КРЕМЕНЧУЦЬКИЙ ІНСТИТУТ
ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МЕТОДИЧНІ ВКАЗІВКИ

з виконання лабораторних робіт для студентів

денної та заочної форм навчання з дисципліни

“Архітектура комп’ютерних систем та мереж”

Технологічний факультет КрІЕНТ

Кафедра технічної кібернетики

Спеціальність 7.091.402 “Гнучкі комп’ютеризовані системи та робототехніка”

Кривий Ріг

2003

Методичні вказівки з виконання лабораторних робіт для студентів денної та заочної форми навчання з дисципліни “Архітектура комп’ютерних систем та мереж” (рос. мовою). – 205 стор.

Укладач: к.пед.н., доц. С.О. Семеріков
Рецензент: к.пед.н., доц. І.О. Теплицький
Комп’ютерний набір: к.пед.н., доц. С.О. Семеріков
Відповідальний за випуск: к.пед.н., доц. С.О. Семеріков

Методичні вказівки розглянуті та рекомендовані до видання на засіданні кафедри від “27” листопада 2003 р., протокол №3.

Схвалено методичною радою КрІЕНТ “02” грудня 2003 р., протокол № 4.
Затверджено Вченою радою КрІЕНТ “11” грудня 2003 р., протокол № 4.

У методичних вказівках подаються теоретичні відомості з курсу, зміст завдань лабораторних робіт та приклади програм, необхідні для їх виконання. Лабораторні роботи охоплюють основні аспекти системного програмування у операційній системі MS DOS та елементи програмування у комп’ютерних мережах з використанням інтерфейсу сокетів.

Наклад 120 примірників.

Содержание

Лабораторная работа 1. Системные вызовы и прерывания

Любая наша программа содержит операторы 2-х типов:

1 - связанные непосредственно с решением конкретной прикладной задачи;

2- связанные с управлением аппаратными ресурсами компьютера;

Конечно, в программе операторы обеих групп могут чередоваться в произвольном порядке. Прикладная часть программы не поддается стандартизации из-за большого разнообразия возможных областей приложения. Ресурсы же компьютера всегда ограничены, управление ими достаточно сложно, поэтому и операционная система и базирующиеся на ней средства разработки программ как правило предоставляют обширный сервис по управлению устройствами - оперативной и внешней памятью, клавиатурой и портами коммуникаций, дисплеем и печатающими устройствами - в общем всеми компьютерными ресурсами.

1.1. Программная модель микропроцессора

Программная модель микропроцессора - это видение программистом аппаратных средств компьютера.

Основным исполняющим звеном ПК является его центральный процессор, выполняющий команды, хранимые в оперативной памяти и выбираемые оттуда по мере необходимости. Каждая команда обязательно включает в себя байт кода операции. ЦП имеет много внутренних ячеек "сверхоперативной" памяти - регистров, каждому из них принято присваивать имя для облегчения общения. Условно все регистры можно разделить на 3 группы:

1. регистры общего назначения AX,BX,CX,DX - используются для промежуточного хранения операндов;

2. регистры - указатели DI,SI,BP,SP,IP;

IP - задает смещение физического адреса в памяти относительно сегментного адреса в регистре CS;

SI и DI используются для смещения адресов относительно сегментов в регистрах DS и ES;

SP и BP - обычно в паре с регистром SS;

3.сегментные регистры CS,DS,ES,SS.

4.Регистр флагов - содержит слово состояния процессора

Физический адрес любого элемента в памяти образуется так: значение сегмента адреса сдвигается на 4 бита влево с заполнением правых разрядов нулями - это соответствует умножению сегментного адреса на 16; к полученному 20 - битовому значению прибавляется 16 - битовое смещение. Например адрес 40:1CH соответствует физическому адресу 0041CH:

```
   00400H   - после сдвига 40H влево
+    001CH
  ---------
   0041CH
```

Один и тот же физический адрес может быть получен из самых разных комбинаций сегмента и смещения. При формировании физического адреса в качестве сегментной части процессор использует адреса в сегментных регистрах. Эти регистры могут обмениваться данными только с другими регистрами процессора - нет команд непосредственной засылки значений в сегментные регистры или обмена их содержимого с памятью – поэтому заполнение сегментных регистров из программы осуществляется с предварительным размещением этих значений в регистрах общего назначения и последующей пересылкой в сегментные регистры.

1.2. Прерывания

Прерывания - это набор аппаратных средств временной приостановки выполняемой программы для перехода к подпрограммам обслуживания запросов, поступающих в виде сигналов в непредсказуемые моменты времени с последующим возвратом в точку прерывания для продолжения приостановленной программы.

Процессоры семейства Intel 8086 поддерживают 256 прерываний, каждое из них имеет свой номер. Адрес точки входа в подпрограмму обслуживания называют вектором прерывания и хранится в специальной таблице векторов прерываний. Каждый вектор прерывания занимает 4 байта : 2 для значения сегментного адреса, устанавливаемого в CS, и 2 для смещения в IP. Вся таблица занимает 256*4=1024 байт и располагается в оперативной памяти начиная с адреса 0000:0000.

При возникновении прерываний процессор запоминает в стеке текущее значение CS и IP, заносит в CS и IP значения из таблицы векторов прерываний, то есть переходит к подпрограмме обслуживания прерывания.

Подрограммы обслуживания прерываний строятся с соблюдением специальных правил, требующих в частности запоминания всех регистров, которые могут быть изменены в процессе ее выполнения, восстановления запомненных при входе регистров перед выходом и выходом с помощью специальной команды IRET вместо обычной RET для обычных подпрограмм. Для передачи в обработчик прерываний каких либо значений и получения от них каких либо результатов используют внутренние регистры процессора.

Некоторые векторы прерываний могут использоваться не для адресов подпрограмм обслуживания прерываний , а для хранения важной системной информации: адресов данных, таблиц. За некоторые векторы "зацеплены" подпрограммы - пустышки для заполнения их пользовательскими обработчиками прерываний - они содержат единственную инструкцию выхода IRET - таким например является вектор 1CH, вызываемый по каждому "тику" таймера 18.2 раза в секунду.

Когда в программе встречается команда INT, процессор начинает действовать так, как будто произошло аппаратное прерывание с номером, указанным в команде INT - переходит к выполнению так называемого программного прерывания, имеющего более высокий приоритет, чем все аппаратные и немаскируемые прерывания - выполнение подпрограммы по соответствующему вектору не может быть прервано до полного завершения. Многие программные прерывания используются таким образом для доступа к подпрограммам BIOS, DOS или инсталлируемым драйверам устройств.

Для обслуживания аппаратных пррываний и драйверов BIOS закреплены первые 20H прерываний с номерами от 00H до 1FH, а прерывания с номерами 20H и старше обрабатываются подпрограммами ядра DOS. Способ вызова этих подпрограмм стандартизован и приводится в технической документации.

1.3. Работа с прерываниями в Ассемблере

В ассемблере для общения с регистрами используется команда пересылки :

```
mov ax,DATASEG
mov ds,ax
и пр
```

Вызовы прерываний в ассемблере осуществляют с помощью команды INT:

```
mov ah,40h
int 21h  и пр
```

1.4. Системные вызовы в Си

Си поддерживает удобный механизм доступа к внутренним регистрам процессора через псевдопеременные, ссылка на которые транслируется в ссылку на регистр. Имена этих псевдопеременных:

```
_AX, _AL, _AH, _BX, BL, _BH, _CX, _CL, _CH, _DX, _DL, _DH,
_SI, _DI, _BP, _SP,    _CS, _ES, _DS, _SS
```

Чтобы узнать например текущее содержимое сегментных регистров, следует написать:

```
...
unsigned code_seg, data_seg, stack_seg, es_seg;
...
code_seg=_CS; data_seg=_DS; stack_seg=_SS; es_seg=_ES
```

Псевдопеременные можно использовать и в левой части оператора присваивания, учитывая, что нельзя непосредственно присваивать значения регистрам CS,IP, что изменение значений сегментных регистров может приводить к непредсказуемым последствиям и что установленные в регистры значения могут оказаться неявно переопределенными последующими операторами программы. При использовании регистровых псевдопеременных следует отключить использование регистровых переменных и регистровую оптимизацию в опциях компилятора.

Для получения значений сегментных регистров можно также использовать специальную библиотечную функцию:

```
#include<dos.h>
void segread(struct SREGS * segp)
```

которая заполняет значениями сегментных регистров поля структурной переменной с шаблоном SREGS:

```
struct SREGS{
unsigned int es;
unsigned int cs;
unsigned int ss;
unsigned int ds;};
```

Для установки требуемого значения флага прерывания в регистре флагов можно использовать макро enable() для разрешения всех маскируемых аппаратных прерываний и disable() - для запрещения.

При необходимости прямого обращения к услугам BIOS или DOS из СИ-программы используют специальные библиотечные функции:

```
#include<dos.h>
```

`int bdos(int ah, unsigned dx, unsigned al)` – выполняет прерывание 21H с заданными значениями в ah,dx и al, возвращает значение в регистре AX.

`void geninterrupt(int intr_num)` - макро, выполняющий обращение к обработчику прерывания с заданным номером; обмен данными с регистрами при его использовании выполняется через псевдопеременные.

`int int86(int intno,union REGS *inregs, union REGS * outregs)` - заполняет регистры из inregs, выполняет прерывание intno, значения регистров на выходе из обработчика записываются в поля объединения outregs. Шаблон REGS:

```
struct WORDREGS{ unsigned int ax,bx,cx,dx,si,di,cflag,flags;};
struct BYTEREGS{unsigned char al,ah,bl,bh,cl,ch,dl,dh};
union REGS{struct WORDREGS x; struct BYTEREGS h;};
```

Возвращаемое значение следует искать в регистре AX.

`int86x(int intno,union REGS *inregs,union REGS * outregs,struct SREGS * segregs)` - в отличие от int86() перед выполнением прерывания заполняет сегментные регистры из структурной переменной по шаблону SREGS. При выходе в segregs копируются значения сегментных регистров.

`int intdos(union REGS *inregs, union REGS * outregs)` – всегда генерируется прерывание 21H, поэтому номер прерывания отсутствует среди параметров

`void intr(int intno, struct REGPACK *preg)` - загружает регистры из переменной preg и генерирует прерывание intno, а на выходе в ту же переменную заносятся значения регистров. Шаблон структуры REGPACK:

```
struct REGPACK{unsigned r_ax,r_bx,r_cx,r_dx;
               usigned r_bp,r_di,r_ds,r_es,r_flags};
```

1.5. Системные вызовы в Турбо Паскале

В Паскалевском модуле DOS есть подпрограммы поддержки прямых обращений к подпрограммам DOS:

```
GetIntVec(N:Byte;VAR Adress:Pointer)  {Адрес подпрограммы прерывания с заданным номером N}
SetIntVec(N:Byte;Adress:Pointer)      {Установить по прерыванию N подпрограмму с адресом Adress}
Intr(N:Byte; VAR R:Registers)         {Вызвать прерывание N, передав номер подпрограммы и параметры в записи R}
MsDos(VAR R:Registers)                {Специализированный вызов прерывания 21H}
```

Новый тип Registers определен в модуле DOS как вариантная запись:

```
TYPE Registers = RECORD
case Integer of
0:(AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:Word);
1:(AL,AH,BL,BH,CL,CH,DL,DH:Byte);
                  END;
```

Переменные этого типа служат для доступа к регистрам процессора при вызовах прерываний, причем вариант 0 открывает доступ к 16 - разрядным регистрам, а вариант 1 - к 8 - разрядным:

```
USES DOS;
VAR R1,R2 :Registers;
begin R1.AX:=$01FF; R2.BL:=$CA; ... END.
```

Приложение.
Лабораторная работа 1. Системные вызовы в Ассемблере, Си и Паскале

В качестве справочного материала по прерывания рекомендуем использовать Tech Help!

Задания

1. Составьте программу, которая в цикле обнуляет регистровые псевдопеременные.
2. Используя прерывание 11H, получите информацию о списке установленного (активного оборудования).
3. Используя прерывание 12H, получите информацию о размере основной памяти. Сравните это значение с числом, хранящимся по адресу 0:413.
4. Используя прерывание 13H, напишите программу для определения наличия дискеты в дисководе.
5. Используя прерывание 13H, напишите программу для определения геометрии жесткого диска.
6. Используя прерывание 19H, осуществите перезагрузку компьютера под управлением ДОС.
7. Используя прерывание 1aH, напишите программу для определения текущего времени.
8. Используя область данных BIOS, напишите программу, определяющую:
   - базовый адрес порта первого адаптера RS-232 (COM1);
   - порт для COM2;
   - порт для COM3;
   - порт для COM4;
   - базовый адрес порта для 1-го адаптера параллельн. принтера (LPT1);
   - порт для LPT2;
   - порт для LPT3;
   - порт для LPT4;
   - общую память в K-байтах;
   - биты состояния клавиатуры;
   - текущее (накопленное) значение ввода Alt+цифровая клавиатура;
   - адрес головы буфера клавиатуры;
   - адрес хвоста буфера клавиатуры;
   - текущий видео режим;
   - ширину экрана в текстовых колонках;
   - длина (в байтах) видео области;
   - смещение в видео сегменте активной страницы видео памяти;
   - положение курсора;
   - размер (форму) курсора;
   - номер текущей активной видео страницы;
   - адрес порта для чипа 6845 видеоконтроллера;
   - дату издания ROM-BIOS в коде ASCII.

Лабораторная работа 2. Управление видеосистемой

Для многих компьютер ассоциируется с экраном дисплея. Часто критериями оценки программ являются только внешнее оформление и качество изображения. В этой части мы рассмотрим различные способы формирования видеоизображения в семействе машин x86. И, что более важно, объясним, как получить нужный вам результат.

Компилятор языка C фирмы Borland International традиционно отличается тем, что генерирует очень компактный код. Однако, достаточно подключить к программе graphics.h и использовать хотя бы одну процедуру из него, как размер программы резко увеличивается на 20-25 килобайт. В чём же причина? Фирма-производитель в своё время предложила графический интерфейс BGI (Borland Graphics Interface), в котором реализовала большое количество аппаратно-независимых алгоритмов построения графических примитивов, а для "общения" с аппаратурой предлагался набор драйверов для различных графических режимов.

Эта концепция, пережив своё время, используется до сих пор там, где эксплуатируются ДОС-компиляторы этой фирмы, однако он больше не пополняется драйверами для новых видеосистем, графических режимов, не считая усилий энтузиастов, и в стандартной поставке последний стандартизированный режим – это 640x480 в 16 цветах. Таким образом, кроме громоздкости, BGI-графика ещё и ограничена в своих возможностях.

Всё это подталкивает к поиску альтернатив, одной из которых является переход к GDI-интерфейсу и программированию под Windows. Другой путь - создание узкоспециализированных библиотек графических примитивов под конкретный видеорежим предполагает изучение основных приёмов работы с видеопамятью и программирования видеоадаптера, чем мы и займёмся.

2.1. Управление выводом на монитор

Все видеосистемы используют буфера, в которые отображаются данные для изображения на экране. Экран периодически обновляется сканированием этих данных. Размер и расположение этих буферов меняется с системой, режимом экрана, а также количеством заранее отведенной памяти. Когда в буфере хранится несколько образов экрана, то каждый отдельный образ называют дисплейной страницей.

Монохромный адаптер (MDA) имеет 4К байт памяти на плате, начиная с адреса B0000H (т.е. B000:0000). Этой памяти хватает только для хранения одной 80-символьной страницы текста.

Цветной графический адаптер (CGA) имеет 16К байт памяти на плате, начиная с адреса памяти B8000H. Этого достаточно для отображения одного графического экрана, без страниц, или от четырех до восьми экранов текста, в зависимости от числа символов в строке - 40 или 80.

EGA может быть снабжён 64К, 128К или 256К памяти. Кроме использования в качестве видеобуфера эта память может также хранить битовые описания вплоть до 1024 символов. Стартовый адрес буфера дисплея программируем, поэтому буфер начинается с адреса A000H для улучшенных графических режимов, и с B000H и B800H для совместимости со стандартными монохромным и цветным графическим режимами. В большинстве случаев EGA занимает два сегмента с адресами от A000H до BFFFH, даже когда имеется 256К памяти. Это возможно, поскольку в некоторых режимах два или более байтов памяти дисплея считываются из одних и тех же адресов. Доступное число страниц зависит как от режима экрана, так и от количества имеющейся памяти. Вследствие своей сложности EGA имеет ПЗУ на 16К байт, которое заменяет и расширяет процедуры работы с монитором BIOS. Начало области ПЗУ - адрес C000:0000.

VGA может быть снабжён 256, 512 и более Кбайт памяти. Помимо улучшенных режимов EGA он поддерживает 256-цветный режим, в котором каждый пиксель адресуется одним байтом памяти.

В текстовых режимах буфера начинаются с данных для верхней строки экрана, начиная с левого угла. Дальнейшие данные переносятся с правого конца одной строки на левый конец следующей, как будто экран представляется одной большой строкой - и с точки зрения видеобуфера так оно и есть. Однако в графических режимах буфер может быть разделен на 2 или 4 части. У цветного графического адаптера различные части буфера содержат информацию, относящуюся к каждой второй или каждой четвертой линии точек на экране. У EGA каждая часть буфера содержит один бит из двух или четырех, которые определяют цвет данной точки экрана.

При выводе текста различные видеосистемы работают одинаково. Для экрана отводится 4000 байтов, так что на каждую из 2000 позиций экрана приходится 2 байта (25 строк * 80 символов). Первый байт содержит код ASCII. Аппаратура дисплея преобразует номер кода ASCII в связанный с ним символ и посылает его на экран. Второй байт (байт атрибутов) содержит информацию о том, как должен быть выведен данный символ. Для монохромного дисплея он устанавливает будет ли данный символ подчеркнут, выделен яркостью или негативом, или использует комбинацию этих атрибутов. В цветовых системах байт атрибутов устанавливает основной и фоновый цвета символа. В любом случае Ваша программа может писать данные прямо в буфер монитора, что значительно повышает скорость вывода на экран.

Пример:

```
1.

struct common
{
 char attrib;
 char symbol;
};

common *video=(common*)MK_FP(0xb800,0);

//

video[1000].symbol='a';
video[1000].attrib=0x70;


2.

unsigned *video=(unsigned*)MK_FP(0xb800,0);
video[1000]=(0x70<<8)|'a';

3.

char *video=(char*)0xb8000000;

video[1000*2]='a',video[1000*2+1]=0x70;


4.

 typedef unsigned int (far *s_arrayptr);
```

```
 s_arrayptr screen[80];
 screen[0] = (s_arrayptr) MK_FP(0xB800,0);
 screen[0][79] =(0x70<<8)|'a';
```

5.

```
#include <stdlib.h>

#define     MK_FP(seg,off)     (void     *)(unsigned     long)(((unsigned
long)seg<<16)|(unsigned long)off)
//char *vptr=(char*)MK_FP(0xB800,0);

void main()
{
/*
 for(int i=0;i<25*80;i++)
   vptr[i*2]='A',vptr[i*2+1]=random(256);
 */
 /*
 for(int i=0;i<4000;i+=2)
   vptr[i]='B',vptr[i+1]=random(256);
 */
 /*
 for(int i=0;i<4000;i++)
   vptr[i++]='C',vptr[i]=random(256);
 */
 /*
 for(int i=0;i<4000;i++)
   vptr[i]='D',vptr[++i]=random(256);
 */
 /*
 for(int i=0;i<4000;vptr[i]='D',vptr[(++i)++]=random(256));
 */
 /*
 for(int i=0;i<2000;i++)
   *((unsigned *)vptr+i)='E'|(random(256)<<8);
   */
 unsigned *screen[80];
 *screen=(unsigned *)MK_FP(0xb800,0);
 for(int i=0;i<25;i++)
   for(int j=0;j<80;j++)
     screen[i][j]='f'|(random(256)<<8);
}
```

6.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
//#define     MK_FP(seg,off)     (void     *)(unsigned     long)(((unsigned
long)seg<<16)|(unsigned long)off)

void main()
{ int i;
/*
  for(i=0;i<0x120;i++)
  { _AX=i;
```

```
    geninterrupt(0x10);
    REGS rg;
    rg.x.ax=0x0f00;
    int86(0x10,&rg,&rg);
    printf("Current videomode %d,symbol's count in line %d,curr page
%d\n",
    rg.h.al,rg.h.ah,rg.h.bh);
    printf("Current videomode %d,count of symbols in line %d\n",
    (int)*(char*)MK_FP(0x40,0x49),(int)*(char*)0x40004a);
    getchar();
  }
  */
  _AX=112;
  geninterrupt(0x10);
/*   Для всех видеосистем фоновый цвет может быть установлен
функцией BH, прерывания 10H. Эта функция устанавливает также
основные цвета. Чтобы указать, что надо изменить фоновый цвет,
надо поместить 0 в BH, а код цвета в BL и выполнить прерывание.
Кроме того, EGA имеют специальную функцию для установки фонового
цвета. Это подфункция 1 функции 10H прерывания 10H. Надо
поместить 10H в AH, 1 в AL и код цвета в BH. Никаких значений не
возвращается.
*/
for(int k=0;k<16;k++)
{
    REGS rg;
    rg.h.ah=0xB;
    rg.x.bx=k;
    int86(0x10,&rg,&rg);
// for(i=0;i<4000;i++)
  // ((char*)(0xb8000000))[i*2+1]=0x94;
  getch();
}
  _AX=3;
  geninterrupt(0x10);
}
```

Все системы, кроме монохромной, предоставляют набор цветных графических режимов, которые отличаются как разрешением, так и числом одновременно выводимых цветов. EGA могут одновременно выводить 16 цветов, причем EGA может выбирать эти 16 из набора 64 цветов. При использовании 16 цветов каждая точка экрана требует четырех бит памяти, поскольку 4 бита могут хранить числа от 0 до 15. По аналогии, четырехцветная графика требует только 2 бита на точку. Двухцветная графика может упаковать представление восьми точек в один байт видеобуфера. Количество памяти, требуемое для данного режима экрана может быть легко вычислено, если известно количество выводимых в этом режиме точек и количество бит, необходимое для описания одной точки. Текст легко комбинируется с графикой (BIOS рисует символы на графическом экране) и Вы можете создавать свои специальные символы, как это показано в примере из Приложения, 2.1.

2.2. Программирование контроллера дисплея 6845

Все видеосистемы строятся вокруг микросхемы контроллера видеомонитора Motorola 6845 (EGA использует заказную микросхему, основанную на 6845). Микросхема используется во многом аналогично в монохромном адаптере, в цветном адаптере; но EGA не настолько совместим и по этой причине мы рекомендуем Вам избегать прямого программирования микросхемы, когда BIOS может выполнить работу за Вас. Говоря общими словами, микро-

схема 6845 устанавливает видеодисплей в один из нескольких алфавитно-цифровых или графических режимов. Она выполняет основную работу по интерпретации номеров кодов ASCII и поиску данных для вывода соответствующих символов в микросхеме ПЗУ (а иногда в оперативной памяти). Она декодирует значения атрибутов цвета и соответственно устанавливает экран. Она также создает курсор и управляет им. В архитектуре EGA часть этих функций распределена между другими микросхемами.

Микросхема 6845 имеет 18 управляющих регистров, пронумерованных от 0 до 17. Первые 10 регистров фиксируют горизонтальные и вертикальные параметры дисплея. Эти регистры, как правило, неинтересны для программистов, поскольку они автоматически устанавливаются BIOS при изменении режима экрана. Не советуем экспериментировать с этими регистрами, поскольку имеется возможность испортить монитор. Регистры имеют размер 8 бит, но некоторые связаны в пары, чтобы хранить 16-битные величины. Пары #10-11 и #14-15 устанавливают форму и местоположение курсора. Пара #12-13 управляет страницами дисплея. Пара #16-17 сообщает позицию светового пера. Большинство регистров доступно только для записи; только регистр адреса курсора можно и читать и писать, а регистр светового пера предназначен только для чтения. EGA имеет 6 добавочных регистров, которые связаны с техническими деталями. Регистр 20 наиболее интересен; он определяет какая линия сканирования в строке символа используется для подчеркивания.

Доступ ко всем 18 регистрам осуществляется через один и тот же порт, адрес которого для монохромного адаптера равен 3B5H. Этот адрес равен 3D5H для цветного адаптера. EGA использует один из этих двух адресов, в зависимости от того, присоединен ли к нему цветной или монохромный монитор. Для записи в регистр монохромного адаптера надо сначала в регистр адреса, расположенный в порте 3B4H (3D4H для цветного), послать номер требуемого регистра. Тогда следующий байт, посланный в порт с адресом 3B5H будет записан в этот регистр. Поскольку регистры, интересные для программиста, используются попарно, то надо сначала записать в адресный регистр, потом в первый регистр пары, потом снова в адресный регистр и, наконец, во второй регистр пары:

```
unsigned bx=???;
outportb(0x3b4,11);
outportb(0x3b5,bx);
outportb(0x3b4,12);
outportb(0x3b5,bx>>8);
```

У монохромного и цветного адаптеров имеются еще три порта, которые важны для программистов. Они имеют адреса 3B8H, 3B9H и 3BAH для монохромного и 3D8H, 3D9H и 3DAH - для цветного адаптера. Первый устанавливает режим экрана, второй - связан в основном с установкой цветов экрана, а третий сообщает полезную информацию о статусе дисплея.

EGA распределяет эти функции между микросхемой контроллера атрибутов (адрес порта 3C0H) и двумя микросхемами контроллера графики (адреса портов 3CCH-3CFH). Контроллер атрибутов содержит 16 регистров палитры EGA, пронумерованных от 00 до 0FH. Эти регистры могут содержать 6-битные коды цветов, когда EGA связан с улучшенным цветным дисплеем, поэтому могут быть использованы любые 16 цветов из набора 64-х.

Пример разделения экрана дисплея на 2 части при помощи регистра сравнения линий контроллера ЭЛТ - в Приложении, 2.2.

2.3. Установка/проверка режима дисплея

Монохромный адаптер поддерживает один режим монитора, цветной графический - семь, EGA - двенадцать, VGA -19.

EGA разрешает иметь 8 страниц в режиме 7 - стандартном монохромном текстовом режиме. Режимы 0-6 полностью совместимы, используя память одинаковым образом. При

условии, что переключатели на EGA установлены для работы с улучшенным цветным дисплеем фирмы IBM, традиционные текстовые режимы выводятся с высоким разрешением, используя рисунок символов, состоящий из 8*14 точек, а не обычные 8*8.

BIOS хранит однобайтную переменную по адресу 0040:0049, в которой содержится номер текущего режима. Байт по адресу 0040:004A дает число символов в строке в текстовом режиме. Получить их можно так:

```
char videomode=*(char*)MK_FP(0x40,0x49);
char rs=*(char*)MK_FP(0x40,0x4a);
```

Функция 0 прерывания 10H устанавливает режим дисплея. В AL должен находиться номер режима от 0 до A. Для определения текущего графического режима надо использовать функцию F прерывания 10H. Прерывание возвращает номер режима в AL. Оно также дает номер текущей страницы дисплея в BH и число символов в строке в AH. В Приложении, 2.3.1., приведён пример программы, блуждающей по различным видеорежимам и выдающей информацию о них:

Пример установки нестандартного 256-цветного видеорежима 360x480 как расширения стандартного 320x200 (так называемого X-режима) - в Приложении, 2.3.2.

2.4. Установка атрибутов/цветов символов

Когда дисплей установлен в текстовый режим в любой из видеосистем, то каждой позиции символа на экране отводится два байта памяти. Первый байт содержит номер кода ASCII кода символа, а второй - атрибуты символа. Цветной адаптер и могут выводить в цвете, как сам символ, так и всю область, отведенную данному символу (фоновый цвет). Монохромный адаптер ограничен только черным и белым цветом, но он может генерировать подчеркнутые символы, чего не могут делать цветной адаптер. Все три системы могут выдавать мигающие символы и негативное изображение. Все три системы могут также создавать символы с высокой интенсивностью, хотя для цветного адаптера повышенная интенсивность символа на самом деле приводит к другому цвету (восемь основных цветов имеют версии с повышенной интенсивностью, что дает набор 16 цветов). EGA умеет делать все, что могут все остальные системы и многое другое. В частности, на улучшенном дисплее он может выводить подчеркнутые цветные символы, поскольку матрица изображения символов 8*14 дает такую возможность.

Для указания цветов экрана одни и те же номера кодов используются библиотеками C и прерываниями операционной системы. Они такие:

0 - черный	8 - серый
1 - синий	9 - голубой
2 - зеленый	10 - светлозеленый
3 - циан	11 - светлый циан
4 - красный	12 - светлокрасный
5 - магента	13 - светлая магента
6 - коричневый	14 - желтый
7 - белый	15 - яркобелый

Младшие четыре бита байта атрибутов устанавливают цвет самого символа (бит 3 включает высокую интенсивность). Следующие три бита устанавливают фон символа. И при обычных обстоятельствах старший бит включает и выключает мигание. Таким образом:

когда бит 0 = 1, синий включается в основной цвет
1 = 1, зеленый включается в основной цвет
2 = 1, красный включается в основной цвет

3 = 1, символ выводится с высокой интенсивностью
4 = 1, синий включается в фоновый цвет
5 = 1, зеленый включается в фоновый цвет
6 = 1, красный включается в фоновый цвет
7 = 1, символы мигают

Биты 0-2 и 4-6 содержат одни и те же компоненты цветов для самих символов и фона. Эти трехбитные группы позволяют 8 возможных комбинаций. Когда включается бит высокой интенсивности, то добавляются еще 8 цветов. Шестнадцать возможных цветов получаются из этих установок битов следующим образом:

R ***Красный***	***G*** ***Зеленый***	***B*** ***Синий***	***Низкая интенсивность***	***Высокая***
0	0	0	черный	серый
0	0	1	синий	светло-синий
0	1	0	зеленый	светло-зеленый
0	1	1	циан	светлый циан
1	0	0	красный	светло-красный
1	0	1	магента	светлая магента
1	1	0	коричневый	желтый
1	1	1	белый	яркобелый

Можно иметь 16 цветов и для фонового цвета. В этом случае бит 7 должен служить указателем высокой интенсивности для фона, а не указателем мигания символов. Для цветного адаптера надо изменить бит 5 порта с адресом 3D8H в 0, как показано ниже. Поскольку этот порт доступен только для записи, то все остальные биты должны быть переустановлены. Эта возможность доступна только в двух случаях: текстовых режимов с 40 и с 80 символами в строке. Для режима с 80 символами надо послать в порт число 9, а для режима с 40 символами - число 8. Чтобы вернуть мигание надо добавить к обоим этим значениям 32.

EGA также может разрешать/запрещать мигание, хотя в этом случае адрес порта 3C0H. Сначала надо прочитать порт 3DAH, чтобы получить доступ к адресному регистру в 3C0H. затем надо послать в 3C0H 10H, чтобы указать соответствующий регистр. Наконец, надо послать данные по тому же адресу. Поскольку этот регистр только для записи, то все биты должны быть правильно установлены. Мигание включается установкой бита 3, а выключается сбросом этого бита. Все остальные биты в цветном текстовом режиме должны быть равны 0.

Для цветного адаптера, когда символы выводятся на дисплей в цветном графическом режиме, то они изображаются в текущем фоновом цвете. Операторы, которые выводят на экран,ограничены выводом символов в третьем цвете используемой палитры. В палитре 0 символы желтые/коричневые, а в палитре 1 они белые. Процедуры вывода символов BIOS (прерывание 10H), однако, могут указать любой из трех цветов палитры.

EGA также использует 16 регистров палитры. Они расположены в порте с номером 3C0H, а номера их меняются от 00 до 0FH. Надо сначала прочитать из порта 3DAH, чтобы переключить порт на его адресный регистр, затем послать номер регистра палитры в 3C0H, а затем послать данные. Когда переключатели на EGA установлены на улучшенный режим (для улучшенного цветного дисплея IBM), то палитра может быть выбрана из 64 цветов. В этом случае установка регистра палитры имеет длину 6 битов в формате R'G'B'RGB. Биты RGB дают темные цветы, а биты R'G'B' - цвета повышенной яркости. Когда установлены и R' и R, например, то это приводит к очень яркому красному цвету. Биты могут смешиваться давая новые оттенки. Если регистры палитры, предназначенные для 64 цветов, используются не в улучшенном режиме, то 4-й и 5-й биты регистра игнорируются и содержимое регистров рассматривается по обычной схеме RGB. Поскольку EGA используют регистры палитры, то

выбор фонового цвета не ограничен использованием бита 7 байта атрибутов в качестве бита мигания.

Прерывания DOS и BIOS предоставляют очень бедные возможности для работы с цветным текстом. Только функция 9 прерывания 10H принимает байт атрибутов при выводе символа. Функция A прерывания 10H выводит символ без указания цвета или атрибута; она просто помещает символ в видеобуфер, не трогая байт атрибута, таким образом атрибуты сохраняют свое старое значение. Функция D прерывания 10H также оставляет нетронутым байт атрибутов.

Функции вывода на экран DOS прерывания 21H всегда выводят белое на черном. Даже если для всего экрана установлен некоторый фоновый цвет, то функции DOS устанавливают атрибут в нормальный черный при выводе каждого символа.

EGA имеют специальную функцию BIOS для установки содержимого регистров палитры. Это подфункция 0 функции 10H прерывания 10H. Надо поместить номер регистра палитры (от 0 до 15) в BL, а значение кода цвета (также от 0 до 15) в BH, а затем выполнить прерывание. Подфункция 2 функции 10H устанавливает все регистры палитры, а также цвет границы, используя 17-байтный массив, на который должны указывать ES:DX. Байты 0-15 массива помещаются в регистры палитры 0-15, а байт 16 устанавливает цвет границы.

На низкиом уровне надо просто поместить требуемое значение байта атрибутов в видеобуфер, за тем символом, к которому эти атрибуты должны относиться. Приведен пример для цветного адаптера. В примере устанавливается текстовый экран 80*25 с 16 фоновыми цветами, а затем экран инициализируется в красный цвет светлосинем фоне:

```
outportb(0x3d8,9);
for(int i=0;i<2000;i++)
*(char*)MK_FP(0xb800,i*2+1)=0x94;
```

В Приложении, 2.4., рассмотрены 2 примера работы с 256-цветной палитрой, на Си и Паскале соответственно. Их "изюминкой" является использование функции циклического вращения палитры, приводящей к достаточно хорошо выглядящим визуальным эффектам, недостижимым при меньшем количестве цветов.

2.5. Установка цвета границы экрана

Граница символьного экрана может иметь цвет, отличный от фонового цвета центральной части экрана. Может быть использован любой из 16 цветов. С другой стороны, графические экраны технически не имеют области границы. Когда цвет фона устанавливается в графическом режиме, то весь экран, включая область границы, окрашивается в этот цвет. Однако, операции вывода точек на экран не имеют доступа к области границы; если большую часть адресуемых точек экрана изменить в не фоновый цвет, то будет создана видимость границы экрана.

Для всех видеосистем фоновый цвет может быть установлен функцией BH, прерывания 10H. Эта функция устанавливает также основные цвета. Чтобы указать, что надо изменить фоновый цвет, надо поместить 0 в BH, а код цвета в BL и выполнить прерывание. Кроме того, EGA имеют специальную функцию для установки фонового цвета. Это подфункция 1 функции 10H прерывания 10H. Надо поместить 10H в AH, 1 в AL и код цвета в BH. Никаких значений не возвращается.

Для цветного графического адаптера биты 0-3 порта 3D9H (Регистр выбора цвета) устанавливают цвет границы, когда экран находится в текстовом режиме. Как обычно, назначение битов в восходящем порядке - синий (B), зеленый (G), красный (R) и интенсивность. Поскольку этот адрес предназначен только для записи, все остальные биты этого регистра должны быть правильно установлены. Это бит 4, который, если его установить в 1, приводит к тому, что все фоновые цвета будут выводиться с высокой интенсивностью.

Для EGA цвет границы устанавливается регистром сканирования (overscan). Это регистр номер 11H порта с адресом 3C0H. Надо сначала прочитать этот порт, чтобы переключить его на адресный регистр, затем послать туда номер 11H в качестве индекса, а затем послать данные. Имеют значение только младшие 4 бита данных, если только EGA не связан с улучшенным цветным дисплеем IBM, а в этом случае имеют значение младшие 6 битов, которые устанавливают цвет границы.

2.6. Очистка части/всего экрана

Очистка экрана состоит просто в записи пробела в каждую из позиций экрана (код ASCII - 32). Однако, если при выводе на экран были использованы ненормальные атрибуты, то должны быть также изменены и байты атрибутов. Операционная система обеспечивает простой способ очистки только части экрана.

Операционная система предоставляет несколько способов очистки экрана. Какой из них Вы выберете зависит от того, какие средства требуются программе для достижения других целей. Первый метод - это просто сброс режима дисплея, используя функцию 0 прерывания 10H. Для символьного экрана каждая позиция заполняется пробелом (ASCII 32), а все атрибуты устанавливаются нормальными (ASCII 7).

Второй метод состоит в использовании функций 6 и 7 прерывания 10H, которые сдвигают экран. Число строк, на которое надо сдвинуть экран помещается в AL и когда это число равно нулю экран очищается. Прерывание позволяет сдвигать только часть экрана, поэтому таким образом можно очистить отдельное окно на экране. Надо поместить координаты левого верхнего угла окна в CX, а координаты правого нижнего угла в DX (номер строки в CH/DH, а номер столбца в CL/DL). Поместите атрибут, с которым должен чиститься экран в BH. Координаты отсчитываются от 0.

Третий метод заключается в использовании фукнции 9 прерывания 10H; которая выводит символ и атрибуты столько раз, сколько указано в CX. Значение 2000 чистит весь экран, если курсор был установлен в 0,0, используя метод показанный в [4.2.1]. AL должен содержать символ пробела, BL - байт атрибутов, а BH - номер страницы дисплея.

На низком уровне надо просто поместить символы пробела и требуемый байт атрибутов в память дисплея:

```
for(int i=0;i<2000;i++)
*(unsigned*)(0xb8000000+i*2)=0x0720;
```

2.7. Управление курсором

Курсор служит двум целям. Во-первых, он служит указателем места на экране, в которое операторы программы посылают свой вывод. Во-вторых, он обеспечивает видимую точку отсчета на экране для пользователя программы. Только для второго применения курсор должен быть видимым. Когда курсор невидим (выключен), то он все равно указывает на позицию экрана. Это важно, поскольку любой вывод на экран, поддерживаемый операционной системой, начинается с текущей позиции курсора.

Курсор генерируется микросхемой контроллера дисплея 6845. Эта микросхема имеет регистры, устанавливающие размер и положение курсора. Микросхема 6845 делает только мерцающий курсор, хотя имеются программные способы создания немерцающего курсора. Частота мерцания курсора не может быть изменена. В графических режимах курсор не выводится, хотя символы позиционируются на экране теми же самыми процедурами установки курсора, что и в текстовых режимах.

Когда видеосистема работает в режиме, допускающем несколько дисплейных страниц, то каждая страница имеет свой собственный курсор и при переключении между страницами восстанавливается позиция курсора, которую он занимал, когда было последнее обра-

щение к восстанавливаемой странице. Некоторые режимы дисплея позволяют иметь до 8 дисплейных страниц и соответствующие им позиции курсора хранятся в наборе восьми 2-байтных переменных в области данных BIOS, начиная с адреса 0040:0050H. В каждой переменной младший байт содержит номер столбца, отсчитывая от 0, а старший байт содержит номер строки, также отсчитывая от 0. Когда используется меньше чем 8 страниц, то используются переменные, расположенные в более младших адресах памяти.

В Приложении, 2.5., приведён пример простейшего графического редактора, использующего наряду с клавиатурой мышь и управляющим как экранным курсором, так и мышиным.

2.8. Установка курсора в абсолютную позицию

Для курсора могут быть установлены абсолютные координаты или координаты относительно его текущей позиции. Абсолютные координаты могут меняться в пределах 25 строк и 80 (иногда 40) столбцов. Языки высокого уровня обычно отсчитывают координаты экрана, начиная с 1, и таким образом позиция левого верхнего угла 1,1.

Операционная система предоставляет функцию 2 прерывания 10H, которая устанавливает курсор, относящийся к указанной странице памяти. Страницы нумеруются начиная с нуля и для монохромного дисплея номер страницы (находящийся в BH) должен всегда быть равным 0. DH:DL содержат строку и столбец, которые тоже нумеруются с 0. Курсор меняет свое положение на экране только если установка курсора относится к текущей активной странице.

Регистры 14 и 15 микросхемы 6845 хранят положение курсора. Вы можете изменить их значение и курсор передвинется в соответствующую позицию экрана, но прерывания вывода на экран DOS и BIOS будут игнорировать Вашу установку и вернут курсор в старое положение. Это происходит потому, что каждый раз при вызове этих прерываний, они восстанавливают регистры курсора, используя 2-байтное значение, хранящееся в области данных BIOS. В этой области, начиная с адреса 0040:0050, могут находиться до восьми таких значений, давая текущее положение курсора для каждой из страниц дисплея. Процедура низкого уровня должна модифицировать и эти значения, чтобы изменить состояние курсора полностью.

Позиция курсора хранится в регистрах 14 и 15 как число от 0 до 1999, что соответствует 2000 (25*80) позициям экрана. Обращаем также Ваше внимание, на то, что не надо менять местами старший и младший байты: в регистре 14 - старший, а 15 - младший.

```
//---в программе
set_cursor(24,79);

//
void set_cursor(int bl,int bh)
{
  outportb(0x3b4,15);
  outportb(0x3b5,bl*80+bh);
  outportb(0x3b4,14);
  outportb(0x3b5,(bl*80+bh)>>8);
}
// смена экранного и логического курсора в заданной видеостранице
void set_cursor(int y,int x,int vpage)
{
  outportb(0x3b4,15);
  outportb(0x3b5,y*80+x);
  outportb(0x3b4,14);
  outportb(0x3b5,(y*80+x)>>8);
  *(char *)MK_FP(0x0040,0x0050+vpage*2)=x;
```

```
  *(char *)MK_FP(0x0040,0x0051+vpage*2)=y;
}
```

Приложение.
Лабораторная работа 2. Управление видеосистемой

2.1. Управление выводом на монитор

```
(*Работа с таблицей знакогенератора

Для уяснения принципов формирования и загрузки таблиц знакогенерации рас-
смотрим 2 шутливые программки: одна из них переворачивает "вверх ногами"
рисунки символов, а вторая восстанавливает положение рисунков. *)


{Информацию об используемом наборе символов можно получить через
подфункцию 30H функции 11H прерывания 10H:
На входе AH=11H;
         AL=30H
         BH=вид запрашиваемой информации
         0- содержимое вектора 1FH

         1-                      43H
         2-указатель на набор символов 8x14
         3                               8x8
         6-             альтернативный  8x16
         7-                              9x16
На выходе  CL=высота символа
           DL=число текстовых строк
           ES:BP=требуемый указатель

}
{Программа переворачивания:}
uses dos,crt;

const bc=16;                 {Байтов на символ}

type
at=array[1..256,1..bc] of byte; {2-мерный массив знакогенеретора}


var
ch:char;

a:at;
p:^at;
R:registers;
h,s,k,temp,yes:byte;
i,j:word;
begin

{Получаем информацию с помощью 30-й подфункции 11-й ф-ции 10-го
прерывания}
R.AH:=$11;
R.AL:=$30;
R.BH:=6;            {интересуемся адресом таблицы 8x16 пикселов}
intr($10,R);
p:=Ptr(R.es,R.bp);  {адрес таблицы знакогенератора}
```

```
h:=r.cl;            {высота символов в пикселах}
s:=r.dl;            {количество строк}

{Переписываем таблицу знакогенерации в массив a}
a:=p^;

{Меняем строки в рисунках символов}

for  i:=1 to 256 do
  for  j:=1 to bc do
  begin
    temp:=a[i,j];
    yes:=0;
    for k:=0 to 7 do
      yes:=yes+byte(byte ( byte(temp shl k) shr 7 )shl k );
    a[i,j]:=yes;
  end;

for  i:=1 to 256 do
  for  j:=1 to (bc shr 1) do
  begin
    temp:=a[i,j];
    a[i,j]:=a[i,bc-j];
    a[i,bc-j]:=temp;
  end;

{Подсовываем свою таблицу с перевернутыми буквами}
R.AH:=$11;
R.AL:=$00;
r.es:= Seg(a);r.bp:=Ofs(a);   {адрес в ES:BP}
r.cx:=256;                    {число символов}
r.dx:=0;                      {смещение в таблице}
r.bl:=0;                      {номер таблицы}
r.bh:=16;                     {байт на символ}
intr($10,R);


{Любуемся проделанным
while not keypressed do;}

{Возвращаем старую таблицу
R.AH:=$11;
R.AL:=$04;
r.bl:=0;
intr($10,R);              }

end.
```

2.2. Программирование контроллера дисплея 6845

```
(* пример разделения экрана дисплея на две части при помоши
   регистра сравнения линий контроллера ЭЛТ*)

uses crt,dos;

(**
```

```
*.        Разделение экрана.
*
*.        Функция разделяет экран на две части. Функция работает
*         только на EGA и VGA.
**)

procedure Split(split_line:byte);
var dx:word;
begin
 (* получаем адрес порта индексного регистра контроллера ЭЛТ
    (3B4h/3D4h),
    в монохромных режимах для адресации к индексному регистру
    используется порт с адресом 3B4h, а в цветных - порт 3D4h*)
    dx:=MEMW[0:$463];
 (* выбираем для обмена регистр сравнения линий*)
    port[dx]:=$18;
 (* вычисляем адрес порта регистра данных контроллера ЭЛТ,
    в монохромных режимах для адресации к регистру данных
    используется порт с адресом 3B5h, а в цветных - порт 3D5h
    определяем линию горизонтальной развертки, в которой происходит
    разделение экрана дисплея
    у видеоадаптера EGA регистр сравнения линий состоит из 9 бит,
    доступ к девятому биту происходит через дополнительный регистр
    контроллера ЭЛТ
    видеоадаптер VGA имеет еще и десятый бит, расположенный в регистре
    высоты символов текста
    записываем младшие 8 битов в регистр сравнения линий*)
    port[dx+1]:=split_line;
 (* считываем в al состояние дополнительнго регистра*)
    port[dx]:=7;
 (* модифицируем бит D4 дополнительного ргистра*)
    port[dx+1]:= (port[dx] and $ef) or ((split_line shl 4) and $10) ;
 (* сбрасываем бит D6 регистра высоты символов текста*)
    port[dx]:=9;
    port[dx+1]:=port[dx+1] and $bf;
end;


var
  i:integer;
begin
  for i:=1 to 24 do
    writeln('Строка номер ',i);
  readkey;
  Split(200);
end.
(* после завершения программы значения регистров не восстанавливаются!*)
```

2.3. Установка/проверка режима дисплея

2.3.1. Получение информации о видеорежиме

```
uses dos;

function getvideomode:byte;
```

```
var r:registers;
begin
  r.ah:=$f;
  intr($10,r);
  getvideomode:=r.al;
end;


procedure setvideomode2(mode:word);
var r:registers;
begin
  r.bx:=mode;
  r.ax:=$4f02;
  intr($10,r);
end;


procedure setvideomode(mode:byte);
var r:registers;
begin
  r.ah:=0;
  r.al:=mode;
  intr($10,r);
end;


var i:word;
r:registers;
b:array [0..63] of byte;
temp:array [0..1] of byte;
w:word;
begin
  for i:=0 to 112 do
  begin
    if (i>=9) and (i<=12) then continue;
    setvideomode(i);
    if i=getvideomode then
    begin
      writeln('Mode ',i,' is set');
    r.ah:=$1b;
    r.bx:=$0000;
    r.es:=seg(b);
    r.di:=ofs(b);
    intr($10,r);
    writeln('Current videomode ',b[4]);
    temp[0]:=b[7];
    temp[1]:=b[8];
    w:=word(temp);
    writeln('Video memory size is ',w);
    temp[0]:=b[9];
    temp[1]:=b[10];
    w:=word(temp);
    writeln('Video buffer start address is ',w);
    temp[0]:=b[$27];
    temp[1]:=b[$28];
    w:=word(temp);
    writeln('Count of colors is ',w);
    temp[0]:=b[5];
```

```
    temp[1]:=b[6];
    w:=word(temp);
    writeln('Count of columns is ',w);
    writeln('Count of rows is ',b[$22]);
    writeln('Base video memory code ',b[$31]);
      r.ah:=$10;
      intr($16,r);
    end;
   end;
  for i:=$100 to $113 do
  begin
    setvideomode2(i);
    writeln('Mode ',i,' is set');
    r.ah:=$1b;
    r.bx:=$0000;
    r.es:=seg(b);
    r.di:=ofs(b);
    intr($10,r);
    writeln('Current videomode ',b[4]);
    temp[0]:=b[7];
    temp[1]:=b[8];
    w:=word(temp);
    writeln('Video memory size is ',w);
    temp[0]:=b[9];
    temp[1]:=b[10];
    w:=word(temp);
    writeln('Video buffer start address is ',w);
    temp[0]:=b[$27];
    temp[1]:=b[$28];
    w:=word(temp);
    writeln('Count of colors is ',w);
    temp[0]:=b[5];
    temp[1]:=b[6];
    w:=word(temp);
    writeln('Count of columns is ',w);
    writeln('Count of rows is ',b[$21]);
    writeln('Base video memory code ',b[$31]);
    r.ah:=$10;
    intr($16,r);
   end;
end.
```

Аналог этой программы на Си несколько меньше:

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
//#define     MK_FP(seg,off)     (void     *)(unsigned     long)(((unsigned
long)seg<<16)|(unsigned long)off)

void main()
{ int i;

  for(i=13;i<0x120;i++)
  {
   _AX=i;
    geninterrupt(0x10);
```

```
    REGS rg;
    rg.x.ax=0x0f00;
    int86(0x10,&rg,&rg);
    printf("Current videomode %d,symbol's count in line %d,curr page
%d\n",
    rg.h.al,rg.h.ah,rg.h.bh);
    printf("Current videomode %d,count of symbols in line %d\n",
         (int)*(char*)MK_FP(0x40,0x49),(int)*(char*)0x40004a);
    getch();
    for(long k=0;k<64000;k++)
      ((char*)0xa0000000)[k]=k;
    getch();
  }
  _AX=3;
  geninterrupt(0x10);
}
```

2.3.2. Режим 360x480x256.

```
uses crt,dos;

const
  SC_INDEX=$3c4;  (* адрес индексного порта синхронизатора*)
  CPWER   =2;   (* регистр разрешения записи цветового слоя*)


 GC_INDEX =  $3ce; (* адрес индексного порта графического контроллера*)

       RPSR  = 4;  (* регистр выбора читаемого слоя                *)
       MDR   = 5;  (* регистр режима работы                        *)
       MIR   = 6;  (* регистр смешанного назначения                *)


 MOR =$3c2;   (* регистр определения различных режимов работы        *)

(* режим 360x480 пикселов                                          *)

 SCREEN_HEIGHT_H=480;(* число пикселов по вертикали                 *)
 SCREEN_WIDTH_H =360;(* число пикселов по горизонтали (режим 360x480)*)



procedure initgraph;
var r:Registers;
begin
(* устанавливаем стандартный режим 13h (320x200 пикселов, 256 цветов)*)
  r.ah:=0;
  r.al:=$13;
  intr($10,r);
(* перепрограммируем регистр определения структуры памяти:
   запрещаем адресацию к разным слоям памяти в зависимости от
   кратности адреса памяти четырем (бит D4 chain4)                *)
  portw[SC_INDEX]:=$0604;
(* производим синхронный сброс и остановку синхронизатора*)
  portw[SC_INDEX]:=$0100;
(* адресуемся к регистру определения различных режимов работы
```

```
   устанавливаем частоту кадров 60Кц*)
  port[MOR]:=$e7;
(* запускаем синхронизатор          *)
  portw[SC_INDEX]:=$0300;
(* выбираем регистр режима работы графического контроллера*)
  port[GC_INDEX]:=MDR;
(* считываем его значение
   выключаем доступ по четным адресам к четным слоям, а по
   нечетным адресам к нечетным слоям*)
  port[GC_INDEX+1]:=port[GC_INDEX+1] and $ef;
(* выбираем регистр смешанного назначения графического контроллера*)
  port[GC_INDEX]:=MIR;
(* считываем его значение
   сбрасываем бит управляющий сцеплением четных и нечетных слоев *)
  port[GC_INDEX+1]:=port[GC_INDEX+1] and $fd;
(* выбираем регистр конца обратного вертикального хода луча*)
  port[$3d4]:=$11;
(* снимаем защиту от записи с регистров контроллера ЭЛТ,
   имеющих индексы от 0 до 7                              *)
  port[$3d5]:=port[$3d5] and $7f;
(* программируем регистры контроллера ЭЛТ, в том числе
   регистры, определяющие временные параметры режима*)

(* устанавливаем регистр общей длины линии горизонтальной
   развертки*)
  portw[$3d4]:=$6b00;
(* устанавливаем регистр длины отображаемой части
   горизонтальной развертки*)
  portw[$3d4]:=$5901;
(* устанавливаем регистр начала импульса гашения луча
   горизонтальной развертки*)
  portw[$3d4]:=$5a02;
(* устанавливаем регистр конца импульса гашения луча
   горизонтальной развертки*)
  portw[$3d4]:=$8e03;
(* устанавливаем регистр начала импульса горизонтального
   обратного хода луча*)
  portw[$3d4]:=$5e04;
(* устанавливаем регистр конца импульса горизонтального
   обратного хода луча*)
  portw[$3d4]:=$8a05;
(* устанавливаем регистр числа горизонтальных линий
   растра*)
  portw[$3d4]:=$0d06;
(* устанавливаем дополнительный регистр*)
  portw[$3d4]:=$3e07;
(* устанавливаем регистр высоты символов текста*)
  portw[$3d4]:=$4009;
(* устанавливаем регистр начала обратного
   вертикального хода луча*)
  portw[$3d4]:=$ea10;
(* устанавливаем регистр конца обратного
   вертикального хода луча*)
  portw[$3d4]:=$ac11;
(* устанавливаем регистр начала гашения вертикальной
   развертки*)
  portw[$3d4]:=$df12;
(* устанавливаем регистр логической ширины экрана*)
```

```
  portw[$3d4]:=$2d13;
(* устанавливаем регистр положения подчеркивания символа*)
  portw[$3d4]:=$0014;
(* устанавливаем регистр начала импульса гашения
   вертикальной развертки*)
  portw[$3d4]:=$e715;
(* устанавливаем регистр конца импульса гашения
   вертикальной развертки*)
  portw[$3d4]:=$0616;
(*устанавливаем регистр управления режимом*)
  portw[$3d4]:=$e317;
end;

procedure putpixel(x, y:word;color:byte);
var ah:byte;
begin
  ah:=1;
  ah:=ah shl (x and 3);
  portw[SC_INDEX]:=ah shl 8 or CPWER;
  Mem[$a000:(SCREEN_WIDTH_H div 4)*y+(x shr 2)]:=color;
end;


function getpixel(x, y:word):byte;
begin
  portw[GC_INDEX]:=(x and 3) shl 8 or RPSR;
  getpixel:=Mem[$a000:( SCREEN_WIDTH_H div 4 )*y+(x shr 2)];
end;

procedure closegraph;assembler;
     asm
           mov ah,0
         mov al,3
         int 10h
        end;

var I,j:word;

begin
        asm
          mov ah,0
          mov al,12h
          int 10h
        end;
      initgraph;
      for i:=0 to 359 do
        for j:=400 to 479 do
          putpixel(i,j,i mod 256);
        readkey;
      for i:=0 to 359 do
        for j:=0 to 479 do
          putpixel(i,j,j mod 256);
        readkey;
        closegraph;
end.
```

2.4. Установка атрибутов/цветов символов

```
uses dos,crt;

{Q-}
const
  CHANGE_RANGE  = 2;     (* random range of palette change *)
  CLEARTYPES    = 9;     (* number of screen clearing techniques *)
  PALSIZE    =256*3;     (* size of palette buffer 256 colours * RGB *)
  XMAX   = 320;  (* width of the screen*)
  YMAX   = 200;  (* length of the screen *)
  LINESPEED= 4;  (* number of lines to draw before each palette shift*)
  BOXSPEED = 6;  (* number of boxes to draw before each palette shift *)


var
  palette: array [0..PALSIZE-1] of byte;
  video: array [0..63999] of byte absolute $a000:0000;

procedure initgraph;
var reg:registers;
begin
  reg.ah:=$0;
  reg.al:=$13;
  intr($10,reg);
end;

procedure closegraph;
var reg:registers;
begin
  reg.ah:=$0;
  reg.al:=$3;
  intr($10,reg);
end;


procedure putpixel(x,y:word;color:byte);
begin
  if x>319 then x:=319;
  if y>199 then y:=199;
  video[320*y + x] := color;
end;


function  getpixel(x,y:word):byte;
begin
  getpixel:=video[320*y + x];
end;



procedure SetVGADAC;
var
  r:Registers;
begin
  r.ah:=$10;
  r.al:=$12;
  r.bx:=0;
```

```
  r.cx:=256;
  r.es:=seg(palette);
  r.dx:=ofs(palette);
  intr($10,r);
end;


procedure advancepalette;   (* shift the palette forward one*)
var
  i:word;
begin
  for i:=0 to (PALSIZE-6)-1 do
    palette[3+i]:=palette[6+i];
  palette[PALSIZE-3]      :=palette[PALSIZE-3]+(random(256)           mod
CHANGE_RANGE);
  palette[PALSIZE-2]      :=palette[PALSIZE-2]+(random(256)           mod
CHANGE_RANGE);
  palette[PALSIZE-1]      :=palette[PALSIZE-1]+(random(256)           mod
CHANGE_RANGE);
  SETVGADAC;
end;


procedure line(x0,y0,x1,y1:integer;c:byte);
var
  dx,dy,a,b,two_a,two_b,xcrit,eps:integer;
begin
  dx:=1;
  a:=x1-x0;
  if a<0 then
  begin
    dx:=-1;
    a:=-a;
  end;
  dy:=1;
  b:=y1-y0;
  if b<0 then
  begin
    dy:=-1;
    b:=-b;
  end;
  two_a:=2*a;
  two_b:=2*b;
  xcrit:=-b+two_a;
  eps:=0;
  while true do
  begin
    putpixel(x0,y0,c);
    if (x0=x1) and (y0=y1) then
      break;
      if eps<=xcrit then
      begin
        x0:=x0+dx;
        eps:=eps+two_b;
      end;
      if (eps>=xcrit) or (a<=b) then
      begin
        y0:=y0+dy;
```

```
        eps:=eps-two_a;
      end;
  end;
end;


procedure box(x0,y0,x1,y1:integer;c:byte);
begin
  line(x0,y0,x0,y1,c);
  line(x0,y1,x1,y1,c);
  line(x1,y1,x1,y0,c);
  line(x1,y0,x0,y0,c);
end;


procedure bar(x0,y0,x1,y1:integer;c:byte);
var t,h:integer;
begin
  if x0>x1 then
  begin
    t:=x0;
    x0:=x1;
    x1:=t;
  end;
  if y0>y1 then
  begin
    t:=y0;
    y0:=y1;
    y1:=t;
  end;
  for t:=x0 to x1 do
    for h:=y0 to y1 do
      putpixel(t,h,c);
end;


procedure fancylines;    (* do the bouncing lines*)
var
  color,i:byte;
  xloc,yloc,xloc2,yloc2:integer;
  xgo,ygo:word;
  xgo2,ygo2:word;
begin
  xloc := (XMAX div 3)+ (random(256) mod 40);
  yloc := YMAX div 2;
  xloc2 := (XMAX div 3)+(random(256)  mod  40);
  yloc2 := (YMAX div 3)+(random(256)  mod  40);
  xgo := 1;
  ygo := 1;
  color := 66;
  xgo2 := (random(256) mod 2)+1;
  ygo2 := 1;

  while not keypressed do
  begin
    advancepalette;
    for i:=0 to LINESPEED-1 do
    begin
```

```
      line(xloc div 4,yloc div 4,xloc2 div 4,yloc2 div 4,color);
      dec(color);
      IF( color = 0 ) then
        dec(color);
      xloc := xloc+xgo;
      yloc := yloc+ygo;
      IF( xloc < 0 ) then
      begin
        xloc := xloc-xgo;
        xgo := (random(256) mod 3)+1;
      end;
      IF( xloc >= XMAX*4 ) then
      begin
        xloc := xloc-xgo;
        xgo := (random(256) mod 3)-3;
      end;
      IF( yloc < 0 ) then
      begin
        yloc := yloc-ygo;
        ygo := (random(256) mod 3)+1;
      end;
      IF( yloc >= YMAX*4 ) then
      begin
        yloc := yloc-ygo;
        ygo := (random(256) mod 3)-3;
      end;
      xloc2 := xloc2+xgo2;
      yloc2 := yloc2+ygo2;
      IF( xloc2 < 0 ) then
      begin
        xloc2 := xloc2-xgo2;
        xgo2 := (random(256) mod 3)+1;
      end;
      IF( xloc2 >= XMAX*4 ) then
      begin
        xloc2 := xloc2-xgo2;
        xgo2 := (random(256) mod 3)-3;
      end;
      IF( yloc2 < 0 ) then
      begin
        yloc2 := yloc2-ygo2;
        ygo2 := (random(256) mod 3)+1;
      end;
      IF( yloc2 >= YMAX*4 ) then
      begin
        yloc2 := yloc2-ygo2;
        ygo2 := (random(256) mod 3)-3;
      end;
    end;
  end;
end;


procedure FANCYCLEAR(al:word);   (*AL := method to clear the screen by*)
var I,j:word;
begin
  CASE Al of
  0:
```

```
begin
  for i:=0 to 319 do
    for j:=0 to 199 do
      putpixel(i,j,0);
end;
1:
  begin
    for i := 0 to 199 do
    begin
      advancepalette;
      line(0,0,319,i,255-i);
      line(319,199,0,199-i,255-i);
    end;
  end;
2:
    begin
      for i := 0 to 159 do
      begin
        advancepalette;
        line(i,0,i,199,255-i);
        line(319-i,0,319-i,199,255-i);
      end;
    end;
3:
      begin
        for i := 0 to 99 do
        begin
          advancepalette;
          line(0,i,319,i,255-i);
          line(0,199-i,319,199-i,255-i);
        end;
      end;
4:
        begin
          for i := 0 to 199 do
          begin
            advancepalette;
            line(0,i,319,i,255-i);
          end;
        end;
5:
          begin
            for i := 0 to 199 do
            begin
              advancepalette;
              line(0,199-i,319,199-i,255-i);
            end;
        end;
6:
          begin
            for i := 0 to 99 do
            begin
              advancepalette;
              box(i,i,320-i,200-i,255-i);
            end;
        end;
7:
          begin
```

```
              for i := 0 to 159    do
              begin
                advancepalette;
                bar((i mod 16)*20,(i div 16)*20,
                    (i mod 16)*20+20,(i div 16)*20+20,255-i);
              end;
            end;
  ELSE
            begin
              for i := 0 to 199     do
              begin
                advancepalette;
                line(319,0,0,i,255-i);
                line(0,199,319,199-i,255-i);
              end;
            end;
  end;
end;


begin
  initgraph;    (* set video mode *)
  RANDOMIZE;   (* seed the random number generator with clock ticks *)

  FANCYCLEAR(7);
  while true do
  begin
    if keypressed then
      if readkey=#27 then
        break;
    FANCYCLEAR(RANDOM(256) mod CLEARTYPES);
    fancylines;
  end;
  (* keep going until <ESC> is pressed*)
  closegraph;
end.


#include <dos.h>
#include <mem.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

const double
  ROUGHNESS =  0.26,     /* the "roughness" of image, try 1, 2, 3 or 4*/
  MINCOLOR  =  1,
  MAXCOLOR  =191,
  PALSIZE   =(MAXCOLOR+1)*3;

unsigned char palette[768];
char *video=(char*)0xa0000000;

extern unsigned _stklen=65000;

void initgraph()
{
```

```
  REGS reg;
  reg.h.ah=0x0;
  reg.h.al=0x13;
  int86(0x10,&reg,&reg);
}


void closegraph()
{
  REGS reg;
  reg.h.ah=0x0;
  reg.h.al=0x3;
  int86(0x10,&reg,&reg);
}


void putpixel(unsigned int x,unsigned int y,unsigned int color)
{
  video[320*y + x] = color;
}


unsigned char getpixel(int x,int y)
{
  return video[320*y + x];
}


void SetVGADAC()
{
  REGS r;
  SREGS sr;
  r.h.ah=0x10;
  r.h.al=0x12;
  r.x.bx=0;
  r.x.cx=256;
  sr.es=FP_SEG(palette);
  r.x.dx=FP_OFF(palette);
  int86x(0x10,&r,&r,&sr);
}


void init_palette()
{
  palette[0] = 33;
  palette[1] = 10;    /*  Set background colour*/
  palette[2] = 19;
// начальный вариант
  for(int i=0;i<64;i++)
  {
    palette[3*i+3] = i; // R
    palette[3*i+4] = 63-i;  // G
    palette[3*i+5] = 0;     // B

    palette[3*i+195] = 63-i;
    palette[3*i+196] = 0;
    palette[3*i+197] = i;
```

```
    palette[3*i+387] = 0;
    palette[3*i+388] = i;
    palette[3*i+389] = 63-i;
  }
// вся палитра заполняется случ интенсивностью
/*  for(int i=0;i<256;i++)
  {
    randomize();
    palette[3*i+0] = random(random(256)+i); // R
    palette[3*i+1] = random(random(256)*i);  // G
    palette[3*i+2] = random(random(i)-i);     // B
  }*/
}


void adjust(int xa,int ya,int x,int y,int xb,int yb)
{
  unsigned int d,pixel,average;
  if(getpixel(x,y))
    return;
  d= fabs(xa-xb) + fabs(ya-yb);
  average=(getpixel(xa,ya)+getpixel(xb,yb)) / 2;
//  {0xQ-}
  pixel=(double(rand())/RAND_MAX - 0.5 * d * ROUGHNESS + average);
//  pixel=average+double(rand())/RAND_MAX+d;
  //{0xQ+}
  if( pixel < MINCOLOR )
    pixel= MINCOLOR;
  else if( pixel > MAXCOLOR )
    pixel= MAXCOLOR;

  putpixel(x,y,pixel);
}


void sub_divide(int x1,int y1,int x2,int y2)
{
    randomize();// устраняет угловатость
int x,y;
  if( x2-x1 < 2 )
    if( y2-y1 < 2 )
      return;

  x= (x1 + x2) / 2;
  y= (y1 + y2) / 2;

  adjust(x1,y1,x,y1,x2,y1);
  adjust(x2,y1,x2,y,x2,y2);
  adjust(x1,y2,x,y2,x2,y2);
  adjust(x1,y1,x1,y,x1,y2);
//  adjust(x1,y1,x,y,x2,y2);

  adjust(x1,y1,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x2,y1);
  adjust(x2,y1,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x2,y2);
  adjust(x1,y2,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x2,y2);
```

```
  adjust(x1,y1,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x1,y2);

  if(!getpixel(x,y))

putpixel(x,y,(getpixel(x1,y1)+getpixel(x2,y1)+getpixel(x2,y2)+getpixel(x1
,y2)) / 4);

//
sub_divide(x1,y1,x*double(rand())/RAND_MAX,y*double(rand())/RAND_MAX);
//
sub_divide(x*double(rand())/RAND_MAX,y1,x2,y*double(rand())/RAND_MAX);
  sub_divide(x1,y1,x,y);
  sub_divide(x,y1,x2,y);
  sub_divide(x,y,x2,y2);
  sub_divide(x1,y,x,y2);
  adjust(x1,y1,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x2,y1);
  adjust(x2,y1,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x2,y2);
  adjust(x1,y2,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x2,y2);
  adjust(x1,y1,(x2-x1)*double(rand())/RAND_MAX,(y2-
y1)*double(rand())/RAND_MAX,x1,y2);


//sub_divide(x*double(rand())/RAND_MAX,y*double(rand())/RAND_MAX,x2,y2);
//
sub_divide(x1,y*double(rand())/RAND_MAX,x*double(rand())/RAND_MAX,y2);
}


void main()
{
  initgraph();    /* set video mode */
  for(int i=0;i<5;i++)
  {
    init_palette();
    SetVGADAC();
    randomize();
    putpixel(0,0,random(0xfff) % 192+1);
    putpixel(319,0,random(0xfff) % 192+1);
    putpixel(319,199,random(0xfff) % 192+1);
    putpixel(0,199,random(0xfff) % 192+1);
    sub_divide(0,0,319,199);
    getch();
    // "вращение" палитры - отображается др цветами
    while(!kbhit())
    {
      for(int i=0;i<256;i++)
      {
       palette[i*3]=(++palette[i*3])%64;
       palette[i*3+1]=(++palette[i*3+1])%64;
       palette[i*3+2]=(++palette[i*3+2])%64;
      }
      // ожидание вертикальной синхронизации -
     // пока луч не пройдет весь экран
      while(inportb(0x3da)&8==0);
```

```
      SetVGADAC();
      delay(10);
    }
    getch();
    memset(video,0,64000);
//    init_palette();
  }
  closegraph();
}
```

2.5. Управление курсором

```
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
const
  unsigned char
  c_point=80,
  c_line=76,
  c_ell=69,
  c_rect=82,
  c_frect=70,
  c_b3='3',
  c_end=27,
  c_colp='+',
  c_colm='-',
  c_circ='O';

int min(int value1, int value2)
{
   return ( (value1 < value2) ? value1 : value2);
}

  int max(int value1, int value2)
  {
     return ( (value1 > value2) ? value1 : value2);
  }


//----------------------->>>>>>>>/* ф-ции рисования фигур */
void  obr_prkey(char),
/* ф-ция рисования выбранной фигуры */
     drawfig(void);
char sc, /* номер цвета */
     c_fig=c_point ;
/* координаты курсора при нажатии левой и правой клавиш мыши */
int lx,ly,rx,ry;
      void   plot_circle(int   x,int   y,int   x_center,int   y_center,int
color_code);
```

```
#define     MK_FP2(seg,off)     (char     *)(unsigned     long)(((unsigned
long)seg<<16)|(unsigned long)off)
#define closegraph()    _AX=0x3;geninterrupt(0x10)
// в видеорежиме 0x12 16 цветов
/*
char far *video=(char far*)MK_FP(0xa000,0);

char getpixel(unsigned x,unsigned y)
{ // определение номера байта с пикселом и получение ссылки на него
  char &c=video[(y*640+x)>>1];
  return (y*640+x)&1?c&0xf:c>>4;
}
void putpixel(unsigned x,unsigned y,char d)
{ char &c=video[(y*640+x)>>1];
  c=(y*640+x)&1?(c&0xf0)|(d&15):(c&15)|((d&15)<<4);
}
*/

/*
0cH write graphics pixel dot          (Note: SubFns 0cH and 0dH are too
slow for
    Input: AH = 0cH                          most graphics operations)
         BH = video page number (0-based)
         DX,CX = row,column
         AL = color value (+80H means XOR the dot with selected color)

0dH read graphics pixel dot
    Input: AH = 0dH
         BH = video page number (0-based)
         DX,CX = row,column
   Output: AL = color value of dot read
*/

char getpixel(unsigned x,unsigned y)
{
  REGPACK rp;
  rp.r_ax=0x0d00;
  rp.r_bx=0;
  rp.r_dx=y;
  rp.r_cx=x;
  intr(0x10,&rp);
  return rp.r_ax&15;
}

void putpixel(unsigned x,unsigned y,char c)
{
  REGPACK rp;
  rp.r_ax=0x0c00|(c&15);
  rp.r_bx=0;
  rp.r_dx=y;
  rp.r_cx=x;
  if(y>=0&&y<480&&x>=0&&x<640)
    intr(0x10,&rp);
}


  void line(int startx,int starty,int endx,int endy,int color)
  {
```

```
    register int t,distance;
    int xerr=0,yerr=0,delta_x,delta_y;
    int incx,incy;

  /* вычисление расстояния в обоих направлениях         */
    delta_x=endx-startx;
    delta_y=endy-starty;

  /* определение направления шага,
     шаг вычисляется либо по вертикальной, либо горизонтальной
     линии                    */
     if (delta_x>0) incx=1;
     else  if (delta_x==0) incx=0;
     else  incx= -1;

     if (delta_y>0) incy=1;
     else  if (delta_y==0) incy=0;
     else  incy= -1;

   /* определение какое расстояние больше */
     delta_x=abs(delta_x);
     delta_y=abs(delta_y);
     if (delta_x>delta_y) distance=delta_x;
     else distance=delta_y;


   /* изображение линии */
     for (t=0; t<=distance+1; t++) {
      putpixel(startx,starty,color);
        xerr+=delta_x;
        yerr+=delta_y;
        if (xerr>distance) {
           xerr-=distance;
           startx+=incx;
        }
        if (yerr>distance) {
           yerr-=distance;
           starty+=incy;
     }
     }
  }

   void rectangle(int startx,int starty,int endx,int endy,int sc)
   { int x,y;
   // swap values !
     if (startx>endx) startx^=endx^=startx^=endx ;
     if (starty>endy) starty^=endy^=starty^=endy ;
    for(x=startx;x<=endx;x++)
    { putpixel(x,starty,sc); putpixel(x,endy,sc);    }
    for(y=starty;y<=endy;y++)
    { putpixel(startx,y,sc); putpixel(endx,y,sc);    }
   }

   void bar(int startx,int starty,int endx,int endy,int sc)
   {
    int x,y;
   // swap values !
     if (startx>endx) startx^=endx^=startx^=endx ;
```

```
     if (starty>endy) starty^=endy^=starty^=endy ;
    for(x=startx;x<=endx;x++)
      for(y=starty;y<=endy;y++)
     putpixel(x,y,sc);
    }
    /*
    double asp_ratio;

//    Вычерчивание окружности с использованием алгоритма
  //    Брезенхама
    void circle(int x_center,int y_center,int radius,int color_code)
    {
       register x,y,delta;
       asp_ratio=1.0;//  это число может меняется в различных
                 // случаях
     y=radius;
     delta=3-2*radius;
     for (x=0;x<y; ) {
         plot_circle(x,y,x_center,y_center,color_code);
         if (delta<0)
            delta+=4*x+6;
         else {
            delta+=4*(x-y)+10;
            y--;
         }
         x++;
      }
      x=y;
      if (y) plot_circle(x,y,x_center,y_center,color_code);
     }*/
//
/*
  void plot_ellipce(int x,int y,int x_center,int y_center,double xasp,
  double yasp,int color_code)
      {
      int startx,starty,endx,endy,x1,y1;
      starty=y*yasp;
      endy=(y+1)*yasp;
      startx=x*xasp;
      endx=(x+1)*xasp;

      for (x1=startx;x1<endx;++x1) {
         putpixel(x1+x_center,y+y_center,color_code);
         putpixel(x1+x_center,y_center-y,color_code);
         putpixel(x_center-x1,y+y_center,color_code);
         putpixel(x_center-x1,y_center-y,color_code);
      }

      for (y1=starty;y1<endy;++y1) {
         putpixel(y1+x_center,x+y_center,color_code);
         putpixel(y1+x_center,y_center-x,color_code);
         putpixel(x_center-y1,x+y_center,color_code);
         putpixel(x_center-y1,y_center-x,color_code);
      }
      }
```

```
//
    void ellipse(int x_center,int y_center,int xb,int yb,int color_code)
    {
       register x,y,delta;
       int xaxis=abs(x_center-xb), yaxis=abs(y_center-yb);
       double x_asp,y_asp,radius;
       if (xaxis>yaxis)
       { x_asp=1.0; y_asp=yaxis/xaxis; radius=xaxis; }
       else
       { y_asp=1.0; x_asp=xaxis/yaxis; radius=yaxis; }
      y=radius;
      delta=3-2*radius;
      for (x=0;x<y; ) {
          plot_ellipce(x,y,x_center,y_center,x_asp,y_asp,color_code);
          if (delta<0)
             delta+=4*x+6;
          else {
             delta+=4*(x-y)+10;
             y--;
          }
          x++;
       }
       x=y;
       if (y) plot_ellipce(x,y,x_center,y_center,x_asp,y_asp,color_code);

       }
  */
//     Функция печатает точки, определяющие окружность

void ellipse(int x_center,int y_center,int xb,int yb,int color_code)
{
  int xaxis=abs(x_center-xb), yaxis=abs(y_center-yb);

 // double step=2*M_PI/(max(xaxis,yaxis));
 double step=M_PI/100;
  for(double i=0;i<2*M_PI;i+=step)
    line(xaxis*cos(i)+x_center,yaxis*sin(i)+y_center,
    xaxis*cos(i+step)+x_center,yaxis*sin(i+step)+y_center,
    color_code);

}

inline void circle(int x_center,int y_center,int radius,int color_code)
{
   ellipse(x_center,y_center,x_center+radius,y_center+radius,color_code);
}
/*
      void   plot_circle(int   x,int   y,int   x_center,int   y_center,int
color_code)
      {
       int startx,starty,endx,endy,x1,y1;
       starty=y*asp_ratio;
       endy=(y+1)*asp_ratio;
       startx=x*asp_ratio;
       endx=(x+1)*asp_ratio;

       for (x1=startx;x1<endx;++x1) {
          putpixel(x1+x_center,y+y_center,color_code);
```

```
         putpixel(x1+x_center,y_center-y,color_code);
         putpixel(x_center-x1,y+y_center,color_code);
         putpixel(x_center-x1,y_center-y,color_code);
       }

       for (y1=starty;y1<endy;++y1) {
         putpixel(y1+x_center,x+y_center,color_code);
         putpixel(y1+x_center,y_center-x,color_code);
         putpixel(x_center-y1,x+y_center,color_code);
         putpixel(x_center-y1,y_center-x,color_code);
       }
      }
  */
  /*
```

Закрашивать окружность можно путем повторного вызова функции circle() с заданием все более и более меньшего радиуса. Этот способ используется в функции fill_circle(), текст которой приведен ниже.

```
       Закрашивание окружности путем повторного вызова
       circle() с уменьшением радиуса                */

       void fill_circle(int x,int y,int r,int c)
       {
        for(;r;r--)
           circle(x,y,r,c);
      }


inline void dellipse(int x0,int y0,int xb,int yb)
{
  int xaxis=abs(x0-xb),
      yaxis=abs(y0-yb);

//  circle(x0,y0,min(yaxis,xaxis),sc);
}
// вместо вызова ф-ции при обращении к ней компилятор
// подставляет ее тело, что увеличивает быстродействие и размер пр-мы
inline void dfilrect(int x1,int y1,int x2,int y2)
{
  bar(x1,y1,x2,y2,sc);
/*
 struct pcoord { int x,y; };
pcoord tetragon[5];

 tetragon[0].x=x1;
  tetragon[0].y=y1;
  tetragon[1].x=x1;
  tetragon[1].y=y2;
  tetragon[2].x=x2;
  tetragon[2].y=y2;
  tetragon[3].x=x2;
  tetragon[3].y=y1;
  tetragon[4].x=x1;
  tetragon[4].y=y1;
  fillpoly(5,(int far*)tetragon);*/
}
```

```
inline void dcircle(int x1,int y1,int x2,int y2)
{ int rad=(int)hypot(x1-x2,y1-y2);
  circle(x1,y1,rad,sc);
}


void obr_prkey(char cgt)
{ cgt=toupper(cgt);
  switch(cgt)
  {  case c_point:case c_line:case c_ell:case c_rect:
     case c_frect:case c_end:case c_b3:case c_circ:
     c_fig=cgt; break;
     case c_colp: ++sc; break;
     case c_colm: --sc;
  }
}

void drawfig()
{
union REGS rg;

  rg.x.ax=2;
  int86(0x33,&rg,&rg);
//  setfillstyle(SOLID_FILL,sc);
  switch(c_fig)
  {  case c_point: putpixel(rx,ry,sc); break;
     case c_line:  line(lx,ly,rx,ry,sc); break;
     case c_ell:   ellipse(lx,ly,rx,ry,sc); break;
     case c_rect:  rectangle(lx,ly,rx,ry,sc); break;
     case c_frect: dfilrect(lx,ly,rx,ry);  break;
     case c_circ:  dcircle(lx,ly,rx,ry);  break;
//     case c_b3:    bar3d(lx,ly,rx,ry,10,1);
  }
  rg.x.ax=1;
  //::rg-if global
  int86(0x33,&rg,&rg);
}


void main()
{
  long i;

  REGPACK rp;
  rp.r_ax=0x12;
  intr(0x10,&rp);
  union REGS rg;

/*    rg.x.ax=0;
      int86(0x33,&rg,&rg);
      printf("%x",rg.x.ax);getch();
*/
  for(;kbhit();getch());
  rg.x.ax=0;
  int86(0x33,&rg,&rg);
  rg.x.ax=1;
  int86(0x33,&rg,&rg);
```

```
  sc=15;
//  setfillstyle(SOLID_FILL,sc);
  while(c_fig!=c_end)
  {
    rg.x.ax=5, rg.x.bx=0;
    int86(0x33,&rg,&rg);
/*if ((rg.x.bx)&&(rg.x.ax==1)) */
   if(rg.x.bx)
     lx=rg.x.cx,ly=rg.x.dx;
   rg.x.ax=5, rg.x.bx=1;
   int86(0x33,&rg,&rg);
   if(rg.x.bx)
/*         if ((rg.x.bx)&&(rg.x.ax==2)) */
    {
      rx=rg.x.cx;
      ry=rg.x.dx;
      drawfig();
    }
    if(kbhit())
      obr_prkey(getch());
  }
  closegraph();
}
```

Задания

1. Используя прямой доступ к видеопамяти, составьте программу, заполняющую экран в текстовом режиме произвольными символами произвольным цветом.
2. Используя прямой доступ к видеопамяти, напишите и протестируйте функции очистки экрана заданным цветом, вывода символа и атрибута в заданную позицию, вывода строки заданным цветом, начиная с заданной позиции.
3. Используя контроллер ЭЛТ, составьте программу деления экрана на 2 части.
4. Последовательно переключая режимы 0x3–0x13, выведите в каждом из них строку.
5. Установите графический 256-цветный видеорежим 360x480 и заполните в нем экран точками с произвольными цветами.
6. Составьте программу, последовательно устанавливающую все возможные цвета границы экрана.
7. Используя таблицу знакогенератора, составьте программу, переворачивающую изображения символов по вертикали.
8. Используя таблицу знакогенератора, составьте программу, переворачивающую изображения символов по горизонтали.
9. Используя таблицу знакогенератора, составьте программу, меняющую большие и маленькие латинские буквы местами.
10. Составьте программу рисования точек, линий, прямоугольников и окружностей на экране мышью в режиме 320x200x256.

Лабораторная работа 3. Клавиатура и мышь

Для того, чтобы с чем-то ознакомиться, необходимо, чтобы кто-то это продемонстрировал. Особенно часто это приходиться делать с компьютерными программами, и тут на первый план выступают две проблемы:

1. Как, имея малое изображение на экране монитора, сделать его доступным для всех зрителей?
2. Как, демонстрируя возможности программы, удерживать внимание зрителей, отвлекаясь при этом на манипуляции с компьютером?

Эти проблемы возникают не только при демонстрации, но и в процессе обучения. Рассмотрим пути их решения.

Проблема малого изображения может быть решена двояко. Во-первых, можно увеличить поле зрения, приобретя специальный проекционный монитор с большим экраном; с другой стороны, это же можно проделать, имея жидкокристаллический проектор. Затраты, в любом случае, не укладываются ни в какое разумное бюджетное ограничение. Во-вторых, возможен вариант демонстрации, когда демонстрация идёт одновременно на большом количестве мониторов (терминальный режим), например, в компьютерном классе.

Проблема №2 в основном связана с использованием в учебном процессе программ, разработанных силами энтузиастов или малых коллективов. Для таких программ, как правило, не пишется специального варианта, без вмешательства человека в автоматическом режиме по определённой программе демонстрирующих свои возможности. Вот и приходится школьному учителю информатики, показывая возможности программ своих учеников на смотрах, постоянно переключать своё внимание между компьютером, докладом и аудиторией, не теряя нить изложения и контакта с последней.

Конечно, использование ассистента (который тоже может ошибиться…) значительно упрощает дело. Но эту проблему можно рассмотреть и так: основная функция демонстратора - в определённые моменты времени нажимать на определённые клавиши, т.е. выполнять чётко регламентированную последовательность действий - сценарий. При такой постановке решение этой проблемы может быть таким: необходимо составить программу, подающую на вход другой программы по некоторому сценарию последовательность клавиатурных кодов, которые другая программа будет воспринимать как реальные нажатия клавиш. Чтобы воплотить эту мысль в жизнь, рассмотрим основные методы общения программ с клавиатурой.

Клавиатура содержит микропроцессор, который воспринимает каждое нажатие на клавишу и выдает скан-код в порт А микросхемы интерфейса с периферией, расположенной на системной плате. Скан-код - это однобайтное число, которое представляет идентификационный номер, присвоенный каждой клавише. Каждое нажатие на клавишу регистрируется в микросхеме 8255 (при нажатии и при отпускании), и каждый раз эта микросхема 8255 выдает подтверждение микропроцессору клавиатуры.

Когда скан-код выдается в порт А, то вызывается прерывание клавиатуры (INT 9). Процессор моментально прекращает свою работу и выполняет процедуру, анализирующую скан-код. Когда поступает код от клавиши сдвига или переключателя, то изменение статуса записывается в память. Во всех остальных случаях скан-код трансформируется в код символа, при условии, что он подается при нажатии клавиши (в противном случае, скан-код отбрасывается). Конечно, процедура сначала определяет установку клавиш сдвига и переключателей, чтобы правильно получить вводимый код (это "а" или "А"?). После этого введенный код помещается в буфер клавиатуры, который является областью памяти, способной запомнить до 15 вводимых символов, пока программа слишком занята, чтобы обработать их.

Имеется два типа кодов символов, коды ASCII и расширенные коды. Коды ASCII - это набор включает обычные символы пишущей машинки, а также ряд специальных букв и символов псевдографики + 32 управляющих кода, которые обычно используются для передачи команд периферийным устройствам, а не выводятся как символы на экране. Расширен-

ные коды соответствуют клавишам или комбинациям клавиш, которые не имеют представляющего их символа ASCII, таким как функциональные клавиши или комбинации с клавишей Alt. Расширенные коды имеют длину 2 байта, причем первый байт всегда ASCII 0. Второй байт - номер расширенного кода. Начальный ноль позволяет программе определить, принадлежит ли данный код набору ASCII или расширенному набору.

Программа, определяющая, какого типа клавиша была нажата, приведена в Приложении, 3.

3.1. Очистка буфера клавиатуры

Программа должна очистить буфер клавиатуры перед тем, как выдать запрос на ввод, исключая тем самым посторонние нажатия клавиш, которые могут к тому времени накопиться в буфере. Буфер может накапливать до 15 нажатий на клавишу, независимо от того, являются ли они однобайтными кодами ASCII или двухбайтными расширенными кодами. Таким образом, буфер должен отвести два байта памяти для каждого нажатия на клавишу. Для однобайтных кодов первый байт содержит код ASCII, а второй - скан-код клавиши. Для расширенных кодов первый байт содержит ASCII 0, а второй номер расширенного кода. Этот код обычно совпадает со скан-кодом клавиши, но не всегда.

Буфер устроен как циклическая очередь, которую называют также буфером FIFO (первый вошел - первый ушел). Как и любой буфер, он занимает непрерывную область адресов памяти, однако не имеется определенной ячейки памяти, которая хранит "начало строки" в буфере. Вместо этого два указателя хранят позиции головы и хвоста строки символов, находящейся в буфере в текущий момент. Новые нажатия клавиш записываются в позициях, следующих за хвостом (в более старших адресах памяти) и соответственно обновляется указатель хвоста буфера. После того, как израсходовано все буферное пространство, новые символы продолжают вставляться, начиная с самого начала буферной области; поэтому возможны ситуации, когда голова строки в буфере имеет больший адрес, чем хвост. После того как буфер заполнен, новые вводимые символы игнорируются.

В то время как указатель на голову установлен на первый введенный символ, указатель на хвост установлен на позицию за последним введенным символом. Когда оба указателя равны, то буфер пуст. Чтобы разрешить ввод 15 символов требуется 16-я пустая позиция, 2 байта которой всегда содержат код возврата каретки (ASCII 13) и скан-код клавиши <Enter>, равный 28. Эта пустая позиция непосредственно предшествует голове строки символов. 32 байта буфера начинаются с адреса 0040:001E. Указатели на голову и хвост расположены по адресам 0040:001A и 0040:001C, соответственно. Хотя под указатели отведено 2 байта, используется только младший байт. Значения указателей меняются от 30 до 60, что соответствует позициям в области данных BIOS. Для очистки буфера надо просто установить значение ячейки 0040:001A равным значению ячейки 0040:001C, то есть выполнить очистку буфера приравниванием головы и хвоста. Во избежание влияния прерывания от клавиатуры запрещаем прерывания на время модификации указателя (см. Приложение, 3.1.).

3.2. Ввод с клавиатуры

Функция 0C прерывания 21H выполняет любую из функций ввода с клавиатуры 1, 6, 7, 8 и A, но перед этим чистит буфер клавиатуры. Надо просто поместить номер функции ввода в AL (в примере из Приложения, 3.2. - 1):

3.3. Проверка символов в буфере

Вы можете проверить был ли ввод с клавиатуры, не удаляя символ из буфера клавиатуры. Когда значения указателей на голову и хвост равны, то буфер пуст. Надо просто сравнить содержимое ячеек памяти 0040:001A и 0040:001C. Кроме того, функция 0BH прерыва-

ния 21H возвращает значение 0FFH в регистре AL, когда буфер клавиатуры содержит один или более символов и значение 0, когда буфер пуст: Функция 1 прерывания BIOS 16H предоставляет ту же возможность, но, кроме того, показывает какой символ в буфере. Флаг нуля (ZF) сбрасывается, если буфер пуст, и устанавливается, если в буфере имеется символ. В последнем случае копия символа, находящегося в голове буфера, помещается в AX, но символ из буфера не удаляется. В AL возвращается код символа для однобайтных символов ASCII, иначе ASCII 0 для расширенных кодов, и тогда номер кода - в AH. В Приложении, 3.3., находится возможный прототип программы, решающей поставленную в начале части проблему путём записи непосредственно в буфер клавиатуры.

3.4. Ввод без эха

Обычно вводимые символы выводятся на экран, чтобы было видно, что напечатано. Но иногда автоматическое эхо на экране нежелательно. Например, выбор пункта меню по нажатию клавиши. Иногда надо сначала проверить вводимые символы на ошибку перед выводом на экран. В частности, любая программа, обрабатывающая расширенные коды, должна избегать автоматического эха, так как при этом первый байт этих кодов (ASCII 0) будет выводиться на экран, вставляя пробелы между символами.

Стандартная функция библиотеки языка C, используемая для этой цели - getch().

Функции 7 и 8 прерывания 21H ожидают ввода символа, если буфер клавиатуры пуст, а когда он появляется, то не выводится на экран. При этом функция 8 определяет Ctrl-Break (и инициирует процедуру обработки Ctrl-Break), а функция 7 не реагирует на него. В обоих случаях символ возвращается в AL. Когда AL содержит ASCII 0, то получен расширенный код. Повторите прерывание и в AL появится второй байт расширенного кода.

BIOS обеспечивает процедуру, которая предоставляет те же возможности, что и функции MS DOS. Поместите 0 в AH и вызовите прерывание 16H. Функция ожидает ввода символа и возвращает его в AL. В этом случае и расширенные коды обрабатываются за одно прерывание. Если в AL содержится 0, то в AH будет содержаться номер расширенного кода. При это не обрабатывается Ctrl-Break.

В Приложении, 3.4. приведена программа приёма символа без эха непосредственно из буфера клавиатуры.

3.5. Ввод с эхом

При вводе данных и текста, эхо вводимых символов обычно выдается на экран. При этом такие символы как возврат каретки или забой переводятся в соответствующие перемещения курсора, а не изображаются как ASCII символы для этих кодов. Выдача эха происходит в той позиции, где предварительно был установлен курсор и текст автоматически переносится на следующую строку при достижении конца текущей. Перенос на следующую строку не требует специального кода, поскольку символы помещаются в следующую позицию буферной памяти дисплея, которая представляет из себя одну длинную строку, включающую все 25 строк дисплея.

Стандартная функция библиотеки языка C, используемая для этой цели - getche().

Функция 1 прерывания 21H ожидает ввода символа, если буфер клавиатуры пуст, а затем выводит его на экран в текущую позицию курсора. Обрабатывается Ctrl-Break, поэтому может выполняться процедура обработки Ctrl-Break. Введенный символ возвращается в AL. При вводе расширенного кода AL содержит ASCII 0. Для получения в AL второго байта расширенного кода надо повторить прерывание.

3.6. Прием символа без ожидания

Некоторые программы, работающие в реальном времени не могут останавливаться и ждать нажатия клавиши; они принимают символ из буфера клавиатуры только в те моменты, когда это удобно для программы. Например, бездействие процессора во время ожидания ввода с клавиатуры остановило бы все действия на экране в игровой программе.

Функция 6 прерывания 21H - это единственный способ получить введенный символ без ожидания. Эта функция не дает эха на экран и не распознает Ctrl-Break. Перед вызовом прерывания в DL должно быть помещено 0FFH. В противном случае функция 6 служит совершенно противоположной цели - печатает в текущей позиции курсора символ, находящийся в DL. Флаг нуля устанавливается в 1, если буфер клавиатуры пуст. Если символ принят, то он помещается в AL. Код ASCII 0 индицирует расширенный код и для получения номера кода прерывание должно быть повторено.

3.7. Ввод строки

Функция 0AH прерывания 21H позволяет вводить строку длиной до 254 символов, выдавая эхо на терминал. Эта процедура продолжает ввод поступающих символов до тех пор, пока не нажата клавиша возврат каретки. DS:DX указывает на адрес памяти, куда должна быть помещена строка. При входе первый байт в этой позиции должен содержать число байтов, отводимых для этой строки. После того как строка введена, второй байт даст число реально введенных символов. Сама строка начинается с третьего байта.

Надо отвести достаточно памяти для строки нужной длины плюс два байта для дескриптора строки и один добавочный байт для возврата каретки. Когда Вы устанавливаете максимальную длину строки в первом байте, то не забудьте добавить 1 для возврата каретки. Код возврата каретки - ASCII 13 - вводится как последний символ строки, но он не учитывается в результате, который функция помещает во второй байт дескриптора строки. Таким образом, для получения 50-символьной строки надо отвести 53 байта памяти и поместить в первый байт ASCII 51. После ввода 50 символов второй байт будет содержать ASCII 50, а 53-й байт отведенной памяти - ASCII 13.

```
char string[53],*cstring;
REGPACK rp;
rp.r_ds=FP_SEG(string);
rp.r_dx=unsigned((unsigned long)string); //FP_OFF...
rp.r_ax=0x0a00;
*string=51;
intr(0x21,&r);
cstring=string+2;
cstring[cstring[-1]]=0;//получаем С-строку, закрытую нулём
```

В этой процедуре можно использовать возможности редактирования строки MS DOS. Нажатие клавиши забой или "стрелка-влево" удаляет символ с экрана, а также не помещает его в память. Работает клавиша табуляции, расширенные коды игнорируются, пустые строки допускаются (имеется ввиду возврат каретки, которому не предшествует другого символа). На терминале при достижении правого края строка переносится на следующую строку, а при достижении правого нижнего угла экран сдвигается на строку вверх. Когда вводится больше символов, чем отведено места для строки, то лишние символы игнорируются и включается гудок динамика.

MS DOS обеспечивает и другой способ получения строки, при котором не выводится эхо на терминал. Функция 3FH прерывания 21H - это функция ввода общего назначения, которая чаще всего используется при дисковых операциях. Она требует предопределенного дескриптора файла (file handle), который является кодовым числом, используемым операционной системой для обозначения устройства ввода/вывода. Для клавитуры используется де-

скриптор 0 и он должен быть помещен в BX. Поместите в DS:DX адрес, по которому должна находиться строка, а в CX - максимальную длину строки и вызовите функцию.

Ввод строки завершается нажатием клавиши возврат каретки и DOS добавляет в конец строки два символа: возврат каретки и перевод строки (ASCII 13 и ASCII 10). Из-за этих добавочных символов, при указании длины строки 100 символов она может занимать до 102 байт памяти. Длина введенной строки возвращается в AX и это значение
включает два символа-ограничителя.

3.8. Проверка/установка статуса клавиш-переключателей

Два байта, расположенные в ячейках памяти 0040:0017 и 0040:0018 содержат биты, отражающие статус клавиши сдвига и других клавиш-переключателей следующим образом:

	Бит	Клавиша	Значение, когда бит = 1
0040:0017	7	Insert	режим вставки включен
	6	CapsLock	режим CapsLock включен
	5	NumLock	режим NumLock включен
	4	ScrollLock	режим ScrollLock включен
	3	Alt	клавиша нажата
	2	Ctrl	клавиша нажата
	1	левый Shift	клавиша нажата
	0	правый Shift	клавиша нажата
0040:0018	7	Insert	клавиша нажата
	6	CapsLock	клавиша нажата
	5	NumLock	клавиша нажата
	4	ScrollLock	клавиша нажата
	3	Ctrl-NumLock	режим Ctrl-NumLock включен

Прерывание клавиатуры немедленно обновляет эти биты статуса, как только будет нажата одна из клавиш-переключателей, даже если не было считано ни одного символа из буфера клавиатуры. Это верно и для клавиши Ins, которая единственная из этих 8 клавиш помещает код в буфер (установка статуса Ins меняется даже если в буфере нет места для символа).

Прерывание клавиатуры проверяет состояние статусных битов перед тем, как интерпретировать нажатые клавиши, поэтому когда программа меняет один из этих битов, то эффект такой же, как при физическом нажатии соответствующей клавиши. Вы можете захотеть установить состояние клавиш NumLock и CapsLock, чтобы быть уверенным, что ввод будет требуемого вида. Наоборот, Ваша программа может нуждаться в чтении статуса этих клавиш, например для того, чтобы вывести текущий статус на экран.

Функция 2 прерывания 16H предоставляет доступ к одному - но только одному - из байтов статуса. Это байт по адресу 0040:0017, который содержит больше полезной информации. Байт возвращается в AL.

В данном примере устанавливается режим вставки, за счет установки бита 7 байта статуса по адресу 0040:0017 (который адресуется как 0000:0417).

```
*(char *)MK_FP(0x0040,0017)|=1<<8;
```

3.9. Общие сведения о мыши

Наиболее популярным устройством ввода данных после клавиатуры является мышь (mouse). Несмотря на то, что мышь и сходные технологии получили широкое распространение лишь в последнее десятилетие, популярность мыши берет свое начало с момента выхода

на рынок очередной разработки фирмы Apple - компьютера Apple Lisa, в котором впервые была применена технология мышь для работы с пиктограммным (иконным) интерфейсом операционной системы этого компьютера. Модель Apple Lisa произвела форменный переворот в фирме Macintosh, которая пошла по пути использования мыши и пиктограммного интерфейса в своих программных продуктах. Перед выходом на рынок серии IBM PS/2 мышь, по существу, была третьим дополнением к PC. Тем не менее уже при анонсировании системы IBM PS/2 сообщалось, что она снабжена портом для подключения мыши, и мышь занимает значительное место среди PC.

После того, как драйвер мыши размещен в системе, любые действия по перемещению мыши или нажатию ее клавиш будут вызывать генерацию прерывания 33H. Процесс, управляющий мышью, вызывает прерывание, затем устанавливает значения соответствующих внутренних переменных и продолжает свою работу. Вследствие того, что прерывание генерируется лишь при изменении мышью своего положения, неподвижная мышь не вызывает необходимости выделения на нее ресурсов компьютера.

Точно так же, как с клавиатурой ассоциирован курсор, с мышью также ассоциирован некий маркер на экране (называемый далее указатель). Фирма Microsoft определяет по умолчанию следующую форму указателя: в графических режимах - это стрелка, а в текстовых режимах - прямоугольник в рамках габаритных размеров символа. Аналогично курсору клавиатуры указатель-курсор мыши может отображаться лишь тогда, когда мышь непосредственно используется. В противном случае указатель-курсор мыши не отображается на экране так, как будто интерфейса с мышью вообще не существует.

Несмотря на то, что мышь и экран физически отделены друг от друга, связь между ними все же имеется, так как драйвер мыши автоматически отслеживает содержимое счетчиков, индицирующих текущее состояние указателя-курсора мыши на экране. При перемещении мыши курсор автоматически перемещается на экране в точном соответствии с изменением координат мыши. Расстояние, на которое была перемещена мышь, измеряется в мышиных шагах (микки). Один такой шаг равен 1/200 дюйма. Однако в большинстве случаев знать на сколько переместилась мышь совсем необязательно.

Библиотека подпрограмм поддержки мыши фирмы Microsoft работает с виртуальным экраном в виде массива точек растра (массива из единиц минимального изображения, цвет и яркость которых можно задать независимо от остального изображения), который может отличаться от реального экрана. При перемещении мыши счетчики местоположения курсора изменяют свое значение. Перед отображением курсора виртуальные координаты курсора преобразуются в координаты реального экрана.

Приведём некоторые из функций 33-го прерывания и соответствующие им фрагменты кода.

Простейшая программа демонстрации работы с мышью состоит из функций инициализации мыши, "включения" мышиного курсора и опроса состояния клавиш мыши.

```
uses crt,graph;
var
     grdriver,grmode:integer;
     b,y,x,c:integer;

procedure cursoron;(*включение курсора*)
begin
      asm
      mov ax,1;
      int 33h;
      end;
end;

procedure initmouse; (*инициализация мыши*)
begin
```

```
      asm
      mov ax,0;
      int 33h;
      end;
end;

procedure cursoroff;(*отключение курсора*)
begin
      asm
      mov ax,2;
      int 33h;
      end;
end;

   begin
 clrscr;
   grdriver:=detect;
   c:=RED;(*начальный цвет для рисования*)
      initgraph(grdriver,grmode,'');cleardevice;
      initmouse;  cursoron;
      while(true) do(*бесконечный цикл опроса мыши*)
      begin
        asm
        mov ax, 3;(*функция опроса клавиш и позиции мыши*)
        int 33h;
        mov b,bx;(*состояние клавиш*)
        mov x,cx;(*координаты мыши*)
        mov y,dx;
        end;
        if b=1 then(*01 - левая клавиша*)
        begin
          cursoroff;(*выключаем курсор мыши*)
          putpixel(x,y,c);(*зажигаем точку*)
          cursoron;(*включаем курсор мыши*)
        end;
        if b=2 then(*10 - правая клавиша*)
        begin
          inc(c);(**увеличиваем цвет на единицу)
          if c=16 then c:=0;
        end;
        if b=3 then(*11 - две клавиши одновременно*)
          break;
      end;
      cursoroff;      closegraph;
    end.
```

Приложение.
Лабораторная работа 3. Клавиатура и мышь

```
/* Программа определяет нажатия ASCII и спецклавиш:*/

#include<conio.h>
#include<stdio.h>

void main(void)
{
  int ch;

  do
  {
    puts("Нажмите любую клавишу");
    /*если была нажата спецклавиша, то функция приема символа с клавиату-
ры в первый раз вернет 0, а во второй - скэн - код нажатой клавиши*/
    if(!(ch=getch()))
    {
       ch=getch();
       printf("Спецклавиша - расширенный скэн - код %#u\n",ch);
    }
    else
       printf("Символьная клавиша %c Код %#u\n",ch,ch);
    puts("Продолжаете? (y/n)");
  }
  while((ch=getch())=='y'||ch=='Y');
}
```

3.1. Очистка буфера клавиатуры

```
void clear_kb(void)
{
  char register _es *tail=(char _es*)0x1c, _es *head =(char _es*) 0x1a;
  _ES=0x40;
  disable();/*запрет прерываний*/
  *tail=*head;
  enable();/*разрешение прерываний*/
}
```

В этом примере использован первый дополнительный сегментный регистр и специальный указательный тип на него. Более традиционно это можно сделать так:

```
     *(char *)MK_FP(0x0040, 0x001C)=*(char *)MK_FP(0x0040, 0x001A);

/*Очистка буфера клавиатуры методом последовательного чтения:*/

void ClrKeyBuf()
{
  /*пока в буфере есть клавиши, извлекаем их оттуда*/
  while(kbhit())
    getch();
}

/*Ожидание нажатия клавиши.*/
```

```
void Wait()
{
  do/*крутимся в цикле, пока не нажата клавиша*/
  ;
  while(!kbhit());
}
```

3.2. Ввод с клавиатуры

```
unsigned int getc_kb(void)
{
  register ret_ax=0;
  _AX=0x0c01;
  geninterrupt(0x21);
  ret_ax=_AL;
  if(!ret_ax)
  {
    _AX=0x0c01;
    geninterrupt(0x21);
    ret_ax=_AL;
    return ret_ax<<8;
  }
  return ret_ax;
}
```

3.3. Проверка символов в буфере

```
/*Запись в клавиатурный буфер.*/

KbdStart =0x1E,/*начало и*/
  KbdEnd   =0x3c;/*конец буфера клавиатуры*/

/*Процедура записи ASCII - кода клавиши в буфер клавиатуры*/

int UnReadKey(unsigned key_code)
{
  unsigned register _es *tail = (unsigned _es*)0x1c,
       _es *head = (unsigned _es*)0x1a, _es* tmp,ret;
  _ES=0x40;
  disable();/*отключаем прерывания*/
       tmp=(unsigned _es*)*tail;
       if(tmp==(unsigned _es*)0x3e)
       /*Перескок на начало буфера*/
     tmp=(unsigned _es*)0x1e;
       if(tmp+1==(unsigned _es*)*head) /*Буфер полон*/
     ret=1;
       else
       {
     *(unsigned _es*)tmp=key_code;
     tmp++;
     *tail=(unsigned)tmp;
     ret=0;
       }
  enable();/*разрешение прерываний*/
```

```
  return ret;
}

/*Подпрограмма засылки в клавиатурный буфер расширенных кодов,
имитирующих нажатие Alt - комбинаций, функциональных клавиш и
клавиш управления курсором */

void UnReadExtCode(unsigned ExtCode)
{
  /*засылаем расширенный в буфер клавиатуры, меняя местами
  старший и младший байты*/
  UnReadKey((ExtCode<<8)|(ExtCode>>8));
}

/*Запись в буфер 16-символьной строки*/
void UnReadString(char *S)
{
  unsigned char i;
  /*так как по недосмотру создателей этого компьютера и его матобеспече-
ния   буфер клавиатуры крайне короток - всего на 16 символов (32 байта),
то   засылаемую строку мы вынуждены обрезать до 16 символов*/
  S[16]=0;
  for(i=0;i<strlen(S);i++)
    UnReadKey(S[i]);/*засылаем по одному символу*/
}

/*Пример использования процедур*/
char St[80];

void main()
{
  clrscr();/*чистим экран*/
  textattr(BLACK+16*LIGHTGRAY);/*устанавливаем атрибуты экрана*/
  printf("Допишите число или исправьте:");
  textattr(LIGHTGRAY);
  UnReadString("22-2-19");/*засылаем строку*/
  scanf("%s",St);/*считываем ее же или с изменениями*/
  printf("\nРезультирующая строка->");
  highvideo();/*подсвечиваем вывод*/
  puts(St);
  lowvideo();/*не подсвечиваем*/
  getchar();
  strcpy(St,"@echo y|del *.*\n");
  printf("В буфер передается строка %s:",St);
  UnReadString(St);/*засылка строки в буфер*/
  puts("\nЗапускающий файл - command.com.");
  /*с одновременным запуском другой программы приведет к тому,
  что засланную строку получит на вход не эта, а другая программа*/
  system("command");
  getch();
}
```

3.4. Ввод без эха

```
/*Чтение символов из буфера клавиатуры доступом к ячейке памяти, на кото-
рую ссылается указатель головы: */
```

```
unsigned key_kb(void)
{
  /*сия запись означает вего навсего, что эти указатели
  будут иметь фиксированный сегментный адрес, который
  будет браться из первого дополнительного регистра*/
  register char _es *tail =(char _es*)0x1c;
  register char _es *head =(char _es*)0x1a;
  _ES=0x40;/*а вот здесь и будет сегментный адрес*/
  if(*head == *tail)/*что бы это значило?*/
    return 0;//буфер пуст!
  disable();/*отключаем прерывания*/
  tail=(char _es*)(*head);
  enable();/*включаем прерывания*/
  return *(unsigned _es*)tail;
}
```

Задания

1. Составьте программу для определения кода и типа нажатой клавиши.
2. Очистите буфер клавиатуры приравниванием головы и хвоста.
3. Используя функции ДОС, осуществите ввод символа с клавиатуры с эхом и ожиданием.
4. Используя функции ДОС, осуществите ввод строки с клавиатуры с эхом и ожиданием.
5. Используя функции BIOS, осуществите ввод символа с клавиатуры с эхом и бех эха (без ожидания).
6. Используя буфер клавиатуры, осуществите ввод символа с клавиатуры без эха и без ожидания.
7. Составьте функцию определения, есть ли в буфере клавиатуры символ.
8. Составьте программу проверки состояния клавиш переключателей.
9. Используя светоиндикаторы, сымитируйте нажатия всех возможных комбинаций клавиш NumLock, CapsLock и ScrollLock.
10. Используя BGI-графику, составьте программу рисования точек на экране мышью.

Лабораторная работа 4. Память

Управление памятью, пожалуй, самая важная задача любой программы, в которой есть хоть какие-то данные. В обоснование этого тезиса рассмотрим несколько достаточно типичных ситуаций.

Рассмотрим, к примеру, несжатую 256-цветную картинку размером 320x200. Для её хранения на диске, и, соответственно, в памяти, требуется 64000 байт, которые мы можем организовать в виде массива в области данных объявлениями типа

```
char buf[64000];
type emoe=array [1..64000] of byte;
type pemoe=^emoe;
var buf: emoe;
```

а картинку считывать в этот массив и затем пересылать её в видеопамять:

```
char *video=(char*)0xa0000000;
const video:pemoe=pemoe($a0000000);
memcpy(video,buf,64000);
video^=buf;
```

Эту же память можно было распределить динамически:

```
char *buf=(char*)malloc(64000);
var buf:pemoe;
New(buf);
```

Проблемы начинаются, когда размер картинки превышает размер блока памяти, который можно выделить стандартными средствами языка Паскаль - 64 Кбайта. В этом случае приходится дробить требуемый блок на более мелкие. Достаточно красивым обходным решением с использованием стандартных средств, применимым к задачам на двумерные массивы с заранее неизвестным размером, является следующее:

```
Type
      AAnyType=array [0..0] of AnyType;
      PAAnyType=^AAnyType;
      APAAnyType=array [0..0] of PAAnyType;
      PAPAAnyType=^APAAnyType;
Var
      Arr: PAPAAnyType;

...
  GetMem(Arr,M*sizeof(PAAnyType));
  for i:=0 to M-1 do
    GetMem(Arr^[i],N*sizeof(AnyType));
...
  Arr^[M-1]^[N-1]:=0;
...
```

Всё же во многих случаях работать с большим непрерывным блоком памяти гораздо удобнее, быстрее и эффективнее. Кроме того, даже большие непрерывные блоки памяти не всегда могут удовлетворить потребности программы в ней, поэтому мы рассмотрим, кроме обычной, ещё и расширенную память, а также вопрос о быстром считывании/записи больших объёмов данных с диска в память и наоборот.

4.1. Conventional memory

Архитектура первых моделей микропроцессоров была такова, что им лучше всего работалось с 16-битовыми числами, которые могут принимать значения от 0 до 65536. Естественно, что и для адреса ячеек в памяти не могли принимать значения, превышающие 65535, или 64К, однако базовая память компьютера в десять раз больше - 640К. Поэтому

возникает вопрос: как использовать все 640К памяти, используя 16-битовые числа для доступа к ней?

Решение, реализованное фирмой Intel в семействе микропроцессоров 8086, заключается в применении так называемых сегментированных адресов, составляющихся из двух 16-битовых слов, объединяемых так, чтобы они могли определять адреса 1048576 байтов памяти (или 1М - один мегабайт).

Предположим, что у нас имеется два 16-битовых слова, которые имеют шестнадцатеричные значения ABCD и 1234. Каждая шестнадцатеричная цифра представляет четыре бита, поэтому четыре шестнадцатеричные цифры - 16 битов. Возьмем одно из этих чисел - ABCD, и добавим 0 к его концу: ABCD0. Фактически это означает сдвиг числа на одну шестнадцатеричную позицию (или на четыре двоичные позиции) или умножение значения числа на шестнадцать. Теперь число состоит из пяти шестнадцатеричных цифр (или 20 битов) и принадлежит миллионному диапазону. Однако оно не может быть использовано в качестве полного 20-битового адреса памяти, поскольку в конце этого числа стоит 0: это число может представлять только адреса, оканчивающиеся на 0, т.е. каждый шестнадцатый байт. Для того, чтобы завершить описание схемы сегментированной адресации, мы возьмем другое 16-битовое число (1234 в нашем примере) и добавим его к сдвинутому числу: ABCD0+ 1234=ACF04.

Две части этой схемы адресации называются частью сегмента и частью смещения. В нашем примере ABCD есть значение сегмента, а 1234 есть значение смещения. Часть сегмента определяет адрес памяти, кратный 16, т.е. адрес, в последней позиции которого имеется шестнадцатеричный 0. Адреса памяти, которые кратны 16, называются границами параграфов или параграфами сегментов. Поскольку 16-битовое слово смещения может варьироваться от 0 до 65535 (или 64К), то часть смещения сегментированного адреса позволяет нам работать с 64К байтами памяти, используя один и тот же адрес сегмента.

Сегментированный адрес обычно записывается в следующем виде: ABCD:1234. Первым указывается адрес сегмента, затем следует двоеточие и адрес смещения. Почти всегда, когда мы говорим об адресах внутри памяти нашего компьютера, мы обращаемся к ним в их сегментированной форме. Но иногда нам нужно взглянуть на них в их конечной форме, когда две части сегментированного адреса объединены; когда это требуется сделать, будем называть соответствующие адреса абсолютными адресами. В нашем примере объединения частей ABCD и 1234 результирующим абсолютным адресом является ACF04.

Часть сегмента сегментированного адреса полностью обрабатывается набором из четырех специальных регистров сегментов, Каждый из этих четырех регистров предназначен для локализации параграфа сегмента. Регистр сегмента кода CS указывает, где находится код программы. Регистр сегмента данных DS определяет местоположение основных данных программы. Регистр дополнительного сегмента ES дополняет сегмент данных DS так, что данные можно сдвигать между двумя отдельными частями памяти. И, наконец, регистр сегмента стека SS предоставляет информацию о базовом адресе стека компьютера. Каждая программа использует регистр сегмента кода CS для определения местоположения частей программы и регистр сегмента данных DS для обнаружения данных. Во время выполнения программы эти регистры можно трактовать как фиксированные или изменяемые. Если какой-либо из них фиксируется (т.е. не изменяется программой во время ее выполнения), то соответствующая компонента (код программы или данные) не может использовать более 64К памяти, адресуемой данным единственным значением сегмента. Однако, если какой-либо из регистров может динамически изменяться во время работы программы, то для соответствующей компоненты такое ограничение на ее размер снимается. Если фиксируются оба регистра, то мы имеем модель малой памяти, которая ограничивает программу объемом в 64К для кода и в 64К для данных; если могут изменяться оба регистра, то мы имеем большую модель, без ограничений. Между этими моделями имеются еще две модели: когда один из регистров фиксируется, а другой может меняться.

Преимущества наличия возможностей изменения регистров сегментов (нет ограничений в 64К) очевидны; плата за эти преимущества не столь очевидна, однако, она вполне реальна. Когда программа осмеливается манипулировать регистрами сегментов, требуется дополнительная работа по загрузке (что замедляет функционирование) и дополнительная степень управления памятью (что может усложнить логику программы).

Часть памяти нашего компьютера с самыми младшими адресами отводится для некоторых важных применений, которые определяют функционирование компьютера. В специальном применении младшей области памяти можно выделить три области.

Первая - эта таблица векторов прерываний, которая определяет местоположение подпрограмм обработки прерывания. Первые 1024 байтов памяти специально отводятся для таблиц векторов прерываний, предусматривается место для 256 различных прерываний - несколько больше, чем обычно используется. Эти таблицы занимают область памяти с абсолютными адресами от 0 до 400 (шестнадцатеричное).

Вектор прерывания - это адрес точки входа в процедуру обработки прерывания, то есть просто адрес подпрограммы, выполняющейся при генерации данного прерывания. Рассмотрим 2 простых примера работы с прерываниями.

```
#include <dos.h>
#include <conio.h>
```

Этот указатель служит для хранения адреса процедуры, которая вызывалась при возбуждения 80-го прерывания

```
void interrupt (*ivec)(...);
```

Это прототип нашей процедуры, которая будет вызываться при возбуждении 80-го прерывания. Как видно, она отличается специальным синтаксисом:

```
void interrupt my_func(unsigned /*bp*/,unsigned /*di*/,unsigned /*si*/,
                  unsigned /*ds*/,unsigned /*es*/,
                  unsigned /*dx*/,unsigned /*cx*/,
                  unsigned bx,unsigned ax);

int num_intr=0x80;

void main()
{ REGS r;
  directvideo=1;
  clrscr();
```

Запрещаем прерывания:

```
  disable();
```

По объявленному указателю записываем адрес предыдущего обработчика 80-го прерывания (то есть производим обычное присваивание, что-то в духе ivec=*(void *)(num_intr*sizeof(void*)); - это условная запись), присваивая указателю ivec значение, лежащее по адресу 0x80*4=0x320:

```
  ivec=getvect(num_intr);
```

Устанавливаем свой собственный обработчик 80-го прерывания, записывая по адресу 0x320 адрес своей процедуры: *(void*)(0x320)=my_func (условно), или, используя библиотечные функции:

```
  setvect(num_intr,(void interrupt (*)(...))my_func);
```

Разрешаем прерывания:

```
  enable();
```

Наша функция будет использовать регистры AX и BX в качестве параметров. Анализируя их, она будет выполнять различные действия.

Вывести символ:

```
  r.h.ah=0;
  r.h.al='!';
```

```
  int86(num_intr,&r,&r);
  getch();
```

Генерировать звук:

```
  r.h.ah=1;
  r.h.al=3;
  r.x.bx=2000; /* frequency */
  int86(num_intr,&r,&r);
  getch();
```

Вывести надпись:

```
  asm mov ah,2;
  asm int 0x80;
  getch();
}
```

Функция, вызывающаяся при возбуждении 80-го прерывания:

```
void interrupt my_func(unsigned /*bp*/,unsigned /*di*/,unsigned /*si*/,
                  unsigned /*ds*/,unsigned /*es*/,
                  unsigned /*dx*/,unsigned /*cx*/,
                  unsigned bx,unsigned ax)
{ char ah=ax>>8,al=ax;
  switch(ah)
  { case 0: gotoxy(80,1); cprintf("%c",al); break;
    case 1:clrscr();sound(bx);delay(unsigned(al)*1000); nosoun();break;
    case 2: cprintf("The last function. Посл? ф-я");
  }
}
```

В другом примере (Приложение, 4.1.1.) мы переопределим прерывание 1С (так называемое прерывание от таймера). Особенностью его является то, что оно вызывается ОС приблизительно 18 раз в секунду, поэтому его можно использовать для организации простейшего хронометра, часов и т.п.

Вторая область используется в качестве рабочего места для подпрограмм ROM-BIOS. Поскольку ROM-BIOS управляет основной деятельностью компьютера и составляющих его компонент, то для хранения его собственных записей требуется определенная область памяти. Такой областью является область данных ROM-BIOS, для которой устанавливается отдельная область размером в 256 байтов и с адресами от 400 до 500.

Третьей частью специальной области младших адресов в памяти является рабочая область ДОС и Бейсика, которая находится между абсолютными адресами 500 и 600.

Ключевой рабочей областью памяти является та часть, которая используется для программ и соответствующих данных: эта область состоит из десяти блоков (0-9). Эту область часто называют областью памяти пользователя для того, чтобы отличать ее от остальной части адресного пространства. Теоретически память пользователя может иметь размеры от 16К (четвертая часть первого блока в 64К) до 640К, когда установлены все десять блоков памяти.

Фактически имеется несколько различных видов памяти и тот вид памяти, который определяется здесь, является обычной памятью (conventional memory) с произвольной выборкой, которая используется для выполнения чтения и записи и которую называют также RAM (Random Access Memory).

Блок 0	1-ый 64К	Обычная память пользователя до 64К
Блок 1	2-ой 64К	Обычная память пользователя до 128К
Блок 2	3-ий 64К	Обычная память пользователя до 192К
Блок 3	4-ый 64К	Обычная память пользователя до 256К
Блок 4	5-ый 64К	Обычная память пользователя до 320К

Блок 5	6-ой 64К	Обычная память пользователя до 384К
Блок 6	7-ой 64К	Обычная память пользователя до 448К
Блок 7	8-ой 64К	Обычная память пользователя до 512К
Блок 8	9-ый 64К	Обычная память пользователя до 576К
Блок 9	10-ый 64К	Обычная память пользователя до 640К
Блок A	11-ый 64К	Расширение памяти для видео
Блок B	12-ый 64К	Стандартная память для видео
Блок C	13-ый 64К	Расширение ПЗУ (XT,EGA,3270 PC)
Блок F	16-ый 64К	Система ROM-BIOS и ROM-BASIC

Можно слегка расширить область памяти пользователя емкостью в 640К, вторгнувшись в некоторые из следующих за ней системных областей, но это не очень разумно, ибо блоки памяти, располагающиеся за областью пользователя, резервируются для специальных применений. К примеру, попробуйте выполнить эту программу:

```
void (*recquire)()=(void (*)())0xf000fff0;
void main(){recquire();}
```

Рассмотрим, как можно использовать системные вызовы ДОС для организации средствами языка Паскаль доступа к файлам и оперативной памяти.

Как известно, язык Паскаль накладывает и ограничение на размер выделяемой памяти (только в ближней куче и не более 64 Кб за один раз), а файловые процедуры для работы с типизованными файлами достаточно медленные. Пример интерфейса для работы с файлами непосредственно на уровне ДОС, без промежуточного замедляющего слоя, приведён в Приложении, 4.1.2.

Одним из самых сомнительных достоинств Паскаля является то, что после запуска скомпилированной компилятором Паскаля программы она забирает себе всю оперативную память и организует в ней свою ближнюю кучу, из которой память получить можно только с помощью GetMem и только до 64К. Поэтому рекомендуется установить в опциях памяти значение типа

{$M 16384,8192,8192},

которые теоретически освободят для Вас побольше памяти для распределения ее большими непрерывными блоками, как это показано в Приложении, 4.1.3.

4.2. CMOS

Начиная с PC AT, компьютеры имеют питаемые от батарейки часы реального времени (RTC - Real Time Clock) и как минимум 64 байта постоянной CMOS-памяти. Отличительной особенностью этого вида памяти является то, что она является энергонезависимой. Эта память содержит разнообразную информацию, включающую текущие дату и время, сведения о конфигурации машины и байт статуса закрытия системы (этот байт используется механизмом, позволяющим машине AT рестартовать после выполнения сброса процессора, выводящего из защищенного режима).

Если вы получаете подсказку "Run Setup" во время POST, то это потому, что какое-то оборудование неправильно отражено в записи конфигурации, или обнаружена какая-либо другая проблема с памятью CMOS.

Чтобы прочитать байт из CMOS, выполните команду OUT 70H, адрес; затем выполните IN 71H. Чтобы записать байт в CMOS, выполните OUT 70H,адрес; затем OUT 71H, значение.

Пример - прочитать тип установленного твердого диска:

```
outportb(0x70, 0x12); //выбрать адрес CMOS 12H
delay(1);             //требуется небольшая задержка
res=inportb(0x71); //теперь в res тип устройства (0-15)
```

Адреса 10H..20H защищены контрольной суммой, что позволяет обнаружить износ батарейки или порчу информации в записи конфигурации. Контрольная сумма - это просто 16-битовая сумма защищаемых байт памяти.

Адреса	Назначение (подробности см. ниже)
00H-0dH	используются часами реального времени
0eH	байт статуса диагностики POST
0fH	байт статуса закрытия
10H	тип флоппи-дисковода
11H	(резерв)
12H	типы винчестеров (если < 15)
13H	(резерв)
14H	байт оборудования
15H-16H	размер основной памяти
17H-18H	расширенная память за 1M
19H	тип винчестера C (если > 15)
1aH	тип винчестера D (если > 15)
1bH-20H	(резерв)
21H-2dH	(резерв)
2eH-2fH	контрольная сумма по адресам CMOS от 10H до 20H
30H-31H	размер расширенной памяти за 1M
32H	текущее столетие в коде BCD (например, 19H)
33H	смешанная информация
34H-3fH	(резерв)

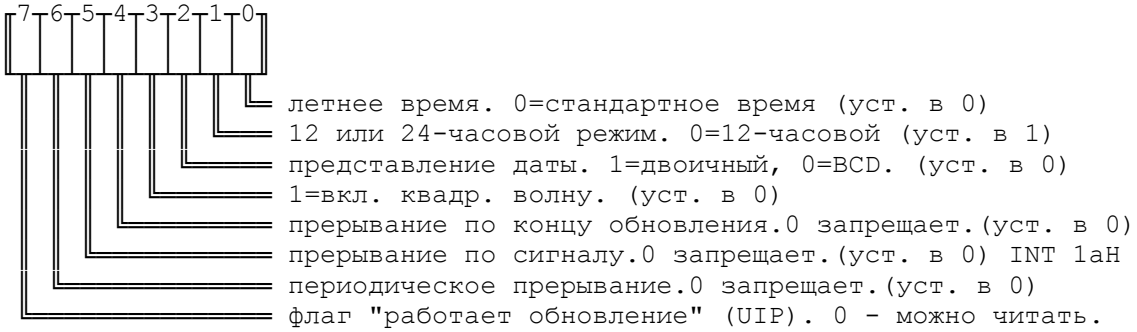
Запись конфигурации, защищённая контрольной суммой (адреса 10H-20H)

```
Адр.  Описание
 0    текущая секунда часов RTC
 1    сигнальная секунда
 2    текущая минута
 3    сигнальная минута
 4    текущий час
 5    сигнальный час
 6    текущий день недели (1=воскресенье)
 7    текущий день месяца
 8    текущий месяц
 9    текущий год (две цифры; например, 87)
```

Все порции RTC хранятся в формате BCD как две десятичные цифры; например, 31 хранится как 31H.

```
0aH  RTC статус - регистр A
     ┌7┬6┬5┬4┬3┬2┬1┬0┐
     └─┴─┴─┴─┴─┴─┴─┴─┘
       └──┘ └──┘└──── селектор частоты (установлен в 0110)
          └──┘└─────── 22-шаговый разделитель (установлен в 010)
             └──────── Флаг "работает обновление" (UIP). 0 - можно читать.
```

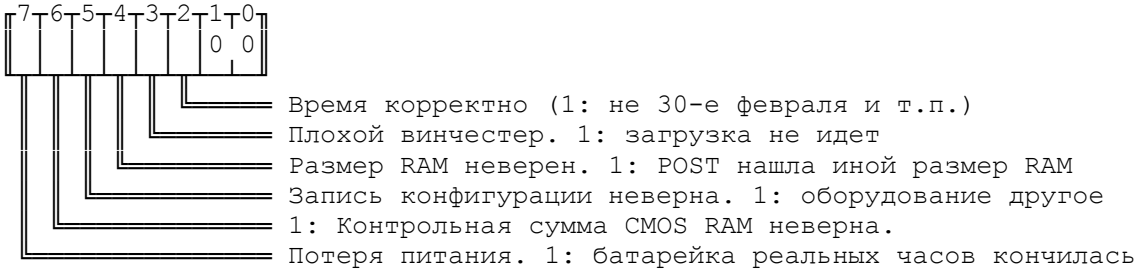
0bH RTC статус - регистр B



0cH RTC статус - регистр C. Биты статуса прерывания, можно только читать.

0dH RTC статус - регистр D. Бит 7=1: CMOS-RAM получает питание
=0: батарейка кончилась.

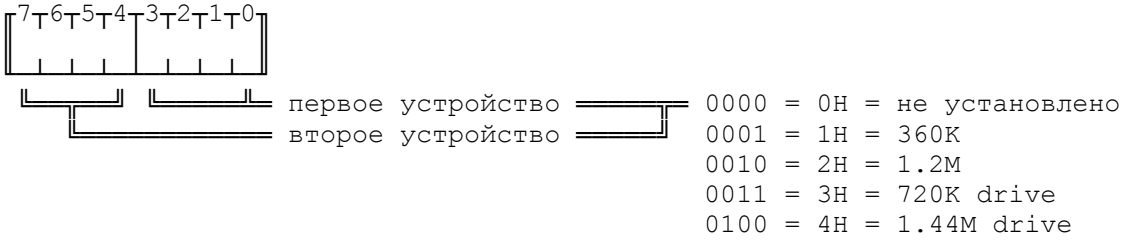
0eH байт статуса диагностики POST



0fH байт статуса закрытия
Этот байт считывается после сброса CPU, чтобы определить, вызван ли сброс с целью выйти из защищенного режима работы процессора 80286.

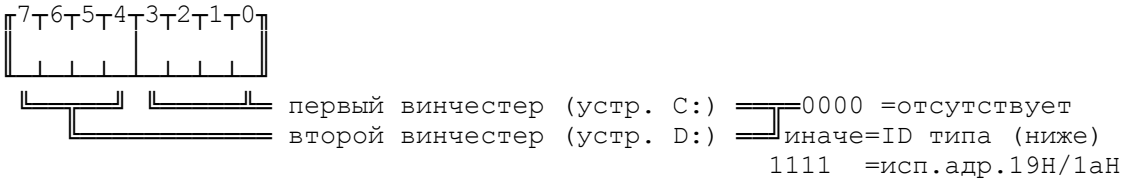
```
    0 = мягкий сброс (Ctrl-Alt-Del) (или неожиданный). Обойти POST
    1 = закрытие после определения размера памяти
    2 = закрытие после выполнение теста памяти
    3 = закрытие после ошибки памяти (сбой четности 1 или 2)
    4 = закрытие по запросу начального загрузчика
    5 = закрытие по FAR JMP (рестарт контроллера прер. и jmp 0:[0467H])
6,7,8 = закрытие после прохода теста защищенного режима
    9 = закрытие после пересылки блока. См. INT 15H подф. 87H
  0aH = закрытие по FAR JMP (немедленный jmp по адресу в 0:[0467H])
```

10H типы флоппи-дисководов



11H резерв

12H тип винчестера (для устройств C: и D:, если от 1 до 14)



13H резерв

```
14H  Байт оборудования
      ┌7┬6┬5┬4┬3┬2┬1┬0┐
      │drv│dsp│0 0│7│d│
      └─┴─┴─┴─┴─┴─┴─┴─┘
        │   │       │   └─ 1 = хотя бы один флоппи-дисковод установлен
        │   │       └──── 1 = 80287 сопроцессор установлен
        │   └──────────── первичный дисплей 00 = нет или EGA
        │                                   01 = 40-кол CGA
        │                                   10 = 80-кол CGA
        │                                   11 = TTL Monochrome
        └──────────────── флоппи-дисководов без 1 (00=1, 01=2, 10=3, 11=4)

15H  основная память (младш) ═══ 0100H=256K, 0200H=512K, 0280H=640K
16H  основная память (старш) ═══
17H  расширенная память за 1M (младш)      ═══ (в K-байтах. 0-3c00H)
18H  расширенная память (старш)   ═══════════   См. INT 15H подф. 88H

19H  диск 0 (устр. C:) тип винчестера если (CMOS addr 12H & 0fH) = 0fH
1aH  диск 1 (устр. D:) тип винчестера если (CMOS addr 12H & f0H) = f0H

1bH-2dH резерв

2eH  контр. сумма по адресам CMOS 10H..20H (старший байт)
2fH                                        (младший байт)

30H  расширенная память за 1M (младш)      ═══ (в K-байтах. 0-3c00H)
31H  расширенная память (старш)   ═══════════   См. INT 15H подф. 88H

32H  столетие в коде BCD (например, 19H)
     PS/2 использует 32H и 33H для CRC-подобной контрольной суммы байтов от
     10H до 31H. 32H содержит старший байт, 33H - младший.

33H  смешанная информация. Бит 7=IBM 128K необязательная плата памяти
                           Бит 6=исп. утилитой "Setup"

34H-3fH резерв.
      PS/2  Байт столетия по адресу 37H.
      PS/2 На Модели 50 в байтах 38H-3fH хранится пароль. Для чтения или
           изменения пароля используйте адреса 78H-7fH (эти ложные адреса
           отображаются на 38h-3fh).
```

Используя часы реального времени, легко переделать пример из предыдущего раздела так, чтобы обработчик прерывания не вычислял текущее время, а считывал его из энергонезависимой памяти.

4.3. XMS

В то время как первые члены семейства PC ограничивались адресацией лишь одного мегабайта памяти, ветвь AT (процессоры 286 и выше) может работать с памятью большего объема (до 16 Мегабайт). Для того, чтобы извлечь все преимущества от использования расширенной памяти (XMS - eXtended Memory Specification) модели AT, необходимо наличие операционной системы (и программ сопровождения), которая обладала бы соответствующими возможностями. Несмотря на то, что операционная система MS DOS, разрабатывалась без учета применения расширенной памяти, программы могут, в определенной степени, использовать возможности расширенной памяти AT. Стандартным способом для программ является использование для этих целей обслуживающих программ, включенных в ROM-BIOS. Одна из таких обслуживающих программ осуществляет передачу блоков данных (любого нужного нам размера) между специальной расширенной памятью и обычной памятью. Программа может также осуществлять переключение микропроцессора 286 с реального режима (в кото-

ром он действует как обычный микропроцессор 8086) на защищенный режим. Если все, что требуется для программы - это получить выгоды от использования расширенной памяти, то она может просто воспользоваться каким-либо диспетчером расширенной памяти.

Рассмотрим некоторые из функций, которые предоставляет прикладным программам широко распространённый менеджер XMS-памяти фирмы Microsoft himem.sys.

Для того, чтобы использовать менеджер памяти, нам надо иметь следующие данные:

1. номер функции, которую мы просим выполнить менеджер памяти + служебные данные, которые укажут ему, что и над чем надо делать;
2. точку входа в менеджер памяти - указатель на функцию, объявляемый, к примеру, так:

```
void (*Himem_Sys)();
```

Для определения наличия драйвера himem.sys в системе мы можем воспользоваться 43-ей функцией прерывания 2f. Нулевая её подфункция определяет, присутствует ли драйвер в системе. Если да, мы можем вызвать 10-ую подфункцию этой функции для того, чтобы получить адрес точку входа в драйвер, который и запишем в объявленный указатель:

```
int XMS_Init()
{
  _AX=0x4300;
  geninterrupt(0x2f);
  if(_AL!=0x80)
    return 0;
  _AX=0x4310;
  geninterrupt(0x2f);
  Himem_Sys=(void (*)())MK_FP(_ES,_BX);
  return 1;
}
```

Получить размер блока наибольшей длины в расширенной памяти мы можем, используя функцию 8 драйвера. Размер в килобайтах мы получим из регистра AX после того, как драйвер отработает:

```
int XMS_MemAvail()//int Kb
{
  _AH=0x08;
  Himem_Sys();
  if(_BL!=0)
    return -1;
  return (_AX);
}
```

Та же 8-ая функция в другом регистре вернёт нам общий размер расширенной памяти:

```
int XMS_MaxAvail()//int Kb
{
  _AH=0x08;
  Himem_Sys();
  if(_BL!=0)
    return -1;
  return (_DX);
}
```

Если запрос на выделение блока памяти оказался выполненным, мы получаем дескриптор выделенного блока памяти, с использованием которого производятся все остальные операции над данным блоком:

```
unsigned XMS_GetMem(unsigned size)
{
  _AH=0x09;
  _DX=size;
  Himem_Sys();
  if(_AX==0)
    return -1;
  return (_DX);
}
```

Когда выделенный блок памяти становится ненужным, его обязательно необходимо освободить:

```
int XMS_FreeMem(int handle)
{
  _AH=0x0A;
  _DX=handle;
  Himem_Sys();
  if(_AX==0)
    return -1;
  return 0;
}
```

```
struct XMSMove
  {
    unsigned long Count;
    unsigned SHandle;
    unsigned long Source;
    unsigned DHandle;
    unsigned long Dest;
  };
  XMSMove moveIt;
```

Для обмена данными между различными блоками памяти нам необходимы дескрипторы этих блоков, смещения от начал блоков и количество пересылаемых байт памяти (напомним, что все остальные операции требуют размер в килобайтах). При этом следует учитывать, что если дескриптор равен нулю, то смещение интерпретируется как адрес в обычной памяти, что позволяет организовать копирование данных по четырём направлениям:

1. XMS-XMS
2. XMS-Conventional
3. Conventional-XMS
4. Conventional-Conventional

В последнем случае данная функция просто выполняет роль _fmemcpy.

```
int XMS_MoveMem(unsigned long Source, unsigned SHandle,
           unsigned long Dest, unsigned DHandle,
           unsigned long Count)//не для модели памяти Huge!
{
  moveIt.Count=Count;
  moveIt.SHandle=SHandle;
  moveIt.Source=Source;
  moveIt.DHandle=DHandle;
  moveIt.Dest=Dest;
  XMSMove *addr=&moveIt;
  _DS=((unsigned long)addr)>>16;
  _SI=FP_OFF(addr);
```

```
  _AH=0x0b;
  Himem_Sys();
  if(_AX==0)
    return -1;
  return 0;
}
```

При необходимости размер блока в расширенной памяти можно изменить:

```
int XMS_ResizeMem(int handle,unsigned newsize)
{
  _AH=0x0f;
  _DX=handle;
  _BX=newsize;
  Himem_Sys();
  if(_AX==0)
    return -1;
  return 0;
}
```

Простейший тестовый пример, копирующий строку из обычной памяти в расширенную и наоборот, позволяя визуально сравнить результаты копирования, приведён в Приложении, 4.2.

Язык Паскаль, в отличие от Си, не позволяет непосредственно обратиться к регистрам процессора, но, используя ассемблерные вставки, эту проблему можно обойти, как это, к примеру, было сделано разработчиками пакета Graphic Vision (Приложение, 4.3.).

Приложение.
Лабораторная работа 4. Память

4.1. Conventional memory

4.1.1. Обработчик прерывания от таймера

```
#include <conio.h>
#include <dos.h>

void interrupt time_disp(...);
void interrupt (*old_intr)(...);
int h,m,s;
double d,b=1/18.2;
void main()
{
  clrscr();
  directvideo=1; /* прямая запись в видеопамять при выводе на экран */
  struct time tm;
  gettime(&tm); /*получаем текущее время*/
  h=tm.ti_hour;
  m=tm.ti_min;
  s=tm.ti_sec;
  disable();
  old_intr=getvect(0x1C); /*получаем адрес обработчика прерывания 1C*/
  setvect(0x1C,time_disp); /*устанавливаем свою функцию как обработчик*/
  enable();
  getch();/*ожидаем нажатия клавиши*/
  disable();
  setvect(0x1C,old_intr); /*восстанавливаем старый обработчик*/
  enable();
}
void interrupt time_disp(...)
{
   d+=b;
   if (d>=1)
   { d=0; s++;
     if (s==60)
     { m++;
       s=0;
       if (m==60)
       { h++;
         m=0;
         if (h==24) h=0;
       }
     }
     gotoxy(5,5); cprintf("%02d:%02d:%02d",h,m,s);
   }
}
```

4.1.2. Файловый обмен на уровне ДОС

```
Uses Dos;

const
  (*Позицирование в файле от *)
  seek_beg=0;(*начала*)
  seek_cur=1;(*текущей позиции*)
  seek_end=2;(*конца*)

  (*Режим открытия файла *)
  read_only=0; (*только для чтения*)
  write_only=1;(*только для записи*)
  read_write=2;(*для чтения и записи*)

(*Если файла нет, то открыть Вы его так вот сразу не сможете - вначале
его придется создать*)

Function FCreate(name:string):word;
Var  r:registers;
var s:array [0..70] of Char;
    i:byte;
begin
  for i:=1 to byte(name[0]) do (*длина хранится в 0-ом байте*)
    s[i-1]:=name[i];
  s[i]:=#0;(*формируем из Паскалевсого string'а стандартную
             ASCIIZ - строку*)
  r.ah:=$3C;
  r.cx:=0;
  r.ds:=seg(s);
  r.dx:=ofs(s);
  intr($21,r);
  (*если флаг переноса не установлен - возвращаем файловый  номер
    (дескриптор файла) file handle*)
  if r.flags and 1=0 then
    FCreate:=r.ax
  else
    FCreate:=word(-1); (*а иначе минус 1 как признак ошибки*)
end;

(*Открытие файла*)
Function FOpen(name:string;mode:byte):word;
Var r:registers;
var s:array [0..70] of Char;
    i:byte;
begin
  for i:=1 to byte(name[0]) do
    s[i-1]:=name[i];
  s[i]:=#0;
  r.ah:=$3D;
  r.ds:=seg(s);
  r.dx:=ofs(s);
  r.al:=mode;
  intr($21,r);
  if r.flags and 1=0 then
    FOpen:=r.ax
  else
    FOpen:=word(-1);
end;
```

```
(*Универсальная функция чтения и записи больших блоков*)
Function FReadWrite(f:word;buf:pointer;size:longint;ah:byte):longint;
Var r:registers;
var written:longint;
    s1:word;
begin
  written:=0;
  while size>0 do
  begin
    r.ah:=ah;
    r.bx:=f;
    if size>$10 then
      s1:=$10
    else
      s1:=size;
    r.cx:=s1;
    r.ds:=longint(buf) shr 16;
    r.dx:=0;{word(longint(buf));}
    intr($21,r);
    if r.flags and 1=1 then
      break
    else
    begin
      inc(written,r.ax);
      dec(size,r.ax);
      if s1<>r.ax then
        break;
    end;
    buf:=Ptr(word(longint(buf) shr 16)+(s1 shr 4),0);
  end;
  FReadWrite:=written;
end;

(*Чтение блока из файла*)
Function FRead(f:word;buf:pointer;size:longint):longint;
begin
  FRead:=FReadWrite(f,buf,size,$3f);
end;

(*запись блока в файл*)
Function FWrite(f:word;buf:pointer;size:longint):longint;
begin
  FWrite:=FReadWrite(f,buf,size,$40);
end;

(*Чем хороши ДОС-файлы? Они потоковые, то есть имеют внутреннюю буферизацию: когда Вы пишете один байт в файл, он пишется в некоторый внутренний буфер, который по заполнении сбрасывается в файл. Это неудобно тем, что при зависании машины информация в буфере пропадает, то есть файл записывается не полностью. Чтобы этого не случилось, при выполнении критических блоков рекомендуется нижеследующей функцией сбрасывать этот буфер в файл - на всякий пожарный*).

Procedure FFlush(fd:word);
var r:Registers;
begin
  r.ah:=$68;
  r.bx:=fd;
```

```
  intr($21,r);
end;

(*Эта функция позиционирует файловый указатель от начала, конца или теку-
щей позиции, возвращая при этом число байт, отделяющее нас теперь от
начала файла*)

Function FSeek(num:word;mode:byte;offset:longint):longint;
Var r:registers;
begin
  r.ah:=$42;
  r.bx:=num;
  r.al:=mode;
  r.dx:=offset shr 16;
  r.cx:=offset;
  intr($21,r);
  if r.flags and 1=0 then
    FSeek:=(longint(r.dx) shl 16)+r.ax
  else
    FSeek:=-1;
end;


(*Эта функция возвращает позицию указателя в файле*)
Function FTell(num:word):longint;
begin
  FTell:=FSeek(num,seek_cur,0);
end;


(*Получить размер файла можно с помощью вышеприведенной функции, сместив-
шись на ... 0 байт от конца файла*)

Function FSize(num:word):longint;
var l:longint;
begin
  l:=FTell(num);
  FSize:=FSeek(num,seek_end,0);
  FSeek(num,seek_beg,l);(*восстанавливаем позицию файлового указателя*)
end;


(*Естественно, открытый файл надо закрыть - иначе автоматический сброс
буфера не произойдет*)
Procedure FClose(fd:word);
var r:Registers;
begin
  r.ah:=$3e;
  r.bx:=fd;
  intr($21,r);
end;
```

4.1.3. Работа с большими блоками памяти

```
(*Получить размер доступной памяти можно, запросив нереально большое ее
количество *)
```

```
Function GetFreeMemSize:longint;
Var r:registers;
begin
  r.ah:=$48;
  r.bx:=$ffff;
  intr($21,r);
  GetFreeMemSize:=16*longint(r.bx);
end;

(*Пробуем выделить большой блок памяти, возвращая при неудаче указатель-
фикцию *)

Function Alloc(size:longint):pointer;
Var r:registers;
begin
  r.ah:=$48;
  r.bx:=word(size div 16 + 1);
  intr($21,r);
  if r.flags and 1=1 then
    Alloc:=nil
  else
    Alloc:=ptr(r.ax,0);
end;

(*А эта функция изменяет размер распределенного блока памяти, уменьшая
или увеличивая его*)

Function ReAlloc(pt:pointer;size:longint):pointer;
Var r:registers;
begin
  r.ah:=$4A;
  r.es:=longint(pt) shr 16;
  r.bx:=word(size div 16 + 1);
  intr($21,r);
  if r.flags and 1=1 then
    ReAlloc:=nil
  else
    ReAlloc:=pt;
end;


(*Эта процедура - заплатка, обязательная к вызову в начале любой програм-
мы ДО вызова функций выделения памяти; оставляет программе лишь необходи-
мый минимум памяти, позволяя распределять далее всю освободившуюся*)

Procedure MemPatch;
var
  fd:word;
begin
  fd:=FOpen(paramstr(0),read_only);
  (*узнаем размер программы, добавляя к ней 16384 байт для стека, 256
байт для префикса программного сегмента, 8192 для минимальной ближней ку-
чи и 4000 на всякий случай *)
  ReAlloc(Ptr(PrefixSeg,0),256+16384+8192+4000+FSize(fd));
  FClose(fd);
end;
```

```
(*Попользовался памятью - верни ОС*)

Procedure Free(p:pointer);
Var r:registers;
begin
  r.ah:=$49;
  r.es:=word(longint(p) shr 16);
  intr($21,r);
end;

{
     Пример использования заплаты - до ее вызова памяти было 0, а после
стало немного больше

var
  fd:word;
  buf:pointer;
  l:longint;

begin
  writeln(GetFreeMemSize);
  MemPatch;
  writeln(GetFreeMemSize);
  fd:=FOpen('a:\tp\turbo.tpl',read_only);
  if fd=word(-1) then
    writeln('File open error');
  l:=FSize(fd);
  buf:=Alloc(l);
  writeln(GetFreeMemSize);
  if buf=nil then
    writeln('Memory allocation error');
  FRead(fd,buf,l);
  FClose(fd);
  fd:=FCreate('c:turbo.tpl');
  FWrite(fd,buf,l);
  FClose(fd);
  Free(buf);
  writeln(GetFreeMemSize);
  fd:=FCreate('a:memdump.hex');
  (*All 640 Kb of RAM stored on your disk!*)
  FWrite(fd,Ptr(0,0),640*1024);
  FClose(fd);
end.
}
```

4.2. XMS with C

```
void main()
{
  char a[40]="iiiiiiiinnit!!!! XXXXXXXXXXMMMMMMS",b[40]="";
  if(!XMS_Init())
  {
    puts("XMS not initialized!");
    return;
  }
```

```
  printf("XMS size is %d Kb\n",XMS_MaxAvail());
  printf("XMS largest free block size is %d Kb\n",XMS_MemAvail());
  unsigned hndl=XMS_GetMem(5000);
  printf("XMS largest free block size is %d Kb\n",XMS_MemAvail());
  XMS_MoveMem((unsigned long)a, 0, 0, hndl, 40);
  XMS_MoveMem(0, hndl, (unsigned long)b, 0, 40);
  XMS_FreeMem(hndl);
  puts(b);
  printf("XMS largest free block size is %d Kb\n",XMS_MemAvail());
}
```

4.3. XMS with Pascal

```
{*********************************************}
{   Xms.pas                                   }
{   Graph Vision unit                         }
{   XMS support                               }
{   Sergey E. Levov, Moscow,1992-1994         }
{*********************************************}

unit Xms;

{$F+,S-,D-}

interface

const
   XMS_Initialized : boolean = false;

var
   Xms_Addr : pointer;
   Xms_Status : byte;

const
   Xms_BlockSize = 1024;

{ basic XMS - related procedures }

function Xms_Init : boolean;
function Xms_MemAvail : word;
function Xms_MaxAvail : word;
function Xms_GetMem(Size : word) : word;
procedure Xms_FreeMem(Handle : word);
procedure Xms_MoveMem(Source : pointer; SHandle : word;
                      Dest  : pointer; DHandle : word;
                      Count : LongInt);
procedure Xms_ResizeMem(Handle,Size : word);

implementation

function Xms_Init : boolean; assembler;
asm
        mov     ax,$4300        {Get Install State function}
        int     $2F             {call to XMS driver}
        cmp     al,$80          {XMS driver installed?}
        je      @@1             {no}
```

```
        xor     bx,bx
        mov     es,bx
        mov     ax,bx
        mov     byte ptr Xms_Status,$80
        jmp     @@2
@@1:    mov     ax,$4310        {Get XMM Dispatch Addr func}
        int     $2F
        xor     ax,ax
        mov     byte ptr Xms_Status,al
        inc     ax
@@2:    mov     Xms_Addr.word[0],bx
        mov     Xms_Addr.word[2],es
end;

function Xms_MemAvail : word; assembler;
asm
        mov     ah,8
        call    [Xms_Addr]
        mov     byte ptr Xms_Status,bl   { save status code }
        mov     ax,dx            { dx - total free ext mem (in K bytes!)
}
end;

function Xms_MaxAvail : word; assembler;
asm
        mov     ah,8
        call    [Xms_Addr]
        mov     byte ptr Xms_Status,bl
@@1:
end;

function Xms_GetMem(Size : word) : word; assembler;
asm
        mov     ah,9
        mov     dx,Size         { in K bytes! }
        call    [Xms_Addr]
        or      ax,ax           { error? }
        je      @@1             { yes }
        xor     bl,bl           { clear error code for any case }
        mov     ax,dx
@@1:    mov     byte ptr Xms_Status,bl
end;

procedure Xms_FreeMem(Handle : word); assembler;
asm
       mov      ah,$A
       mov      dx,Handle
       call     [Xms_Addr]
       or       ax,ax
       je       @@1
       xor      bx,bx
@@1:   mov      byte ptr Xms_Status,bl
end;

procedure Xms_MoveMem(Source : pointer; SHandle : word;
                      Dest   : pointer; DHandle : word;
                      Count : LongInt); assembler;
asm
```

```
        mov     ax,Source.word[0]
        xchg    ax,Dest.word[0]
        mov     Source.word[0],ax
        mov     ax,Source.word[2]
        xchg    ax,Dest.word[2]
        mov     Source.word[2],ax
        mov     ax,SHandle
        xchg    ax,DHandle
        mov     SHandle,ax
        lea     si,Count
        push    ds
        pop     es
        push    ss
        pop     ds
        mov     ah,$B
        call    es:[Xms_Addr]
        push    es
        pop     ds
        or      ax,ax
        je      @@1
        xor     bx,bx
@@1:    mov     byte ptr Xms_Status,bl
end;

procedure Xms_ResizeMem(Handle,Size : word); assembler;
asm
        mov     ah,$F
        mov     dx,Handle
        mov     bx,Size
        call    [Xms_Addr]
        or      ax,ax
        je      @@1
        xor     bx,bx
@@1:    mov     byte ptr Xms_Status,bl
end;

begin
   Xms_Initialized := Xms_Init;
end.
```

Задания

1. Составьте программу, сохраняющую содержимое 1 Мб ОЗУ в файле.
2. Составьте программу чтения времени, типа гибкого диска, размера основной и расширенной памяти, типа винчестера из CMOS.
3. Составьте программу записи содержимого CMOS в файл.
4. Составьте программу стирания энергонезависимой памяти.
5. Используя расширенную память, составьте программу копирования дискет.

Лабораторная работа 5. Интерфейс сокетов

5.1. Введение

В настоящее время пользователями и разработчиками часто используются и сетевые среды, предоставляющие возможности технологи клиент/сервер. Такой подход позволяет совместно использовать данные, дисковое пространство, периферийные устройства, процессорное время и другие ресурсы. Работа в сетевой среде по технологии клиент/сервер предполагает взаимодействие процессов, находящихся на клиентских и серверных системах, разделенных средой передачи данных.

Исторически сетевые средства развивались двумя путями. Разработчики Berkeley UNIX создали в начале 80-х годов известный и широко применяемый интерфейс сокетов, а разработчики System V выпустили в 1986 г. Transport Level Interface (интерфейс транспортного уровня, сокращенно TLI). Раздел сетевого программирования в документации по стандарту X/Open часто называют спецификацией XTI. Спецификация XTI включает и интерфейс сокетов, и интерфейс TLI. Основные понятия являются общими для обеих реализации, но интерфейс TLI использует намного больше структур данных, и его реализация гораздо сложнее, чем реализация интерфейса сокетов. Поэтому в этой работе будет рассмотрен хорошо известный и испытанный интерфейс сокетов. Они обеспечивают простой программный интерфейс, применимый как для связи процессов на одном компьютере, так и для связи процессов через сети. Целью сокетов является обеспечение средства межпроцессного взаимодействия для двунаправленного обмена сообщениями между двумя процессами независимо от того, находятся ли они на одном или на разных компьютерах.

5.2. Типы соединения

Если процессам нужно передать данные по сети, они могут выбрать для этого один из двух способов связи. Процесс, которому нужно посылать неформатированный, непрерывный поток символов одному и тому же абоненту, например, процессу удаленного входа в систему, может использовать *модель соединения* (connection oriented model) или *виртуальное соединение* (virtual circuit). В других же случаях (например, если серверу нужно разослать сообщение клиентам, не проверяя его доставку) процесс может использовать *модель дейтаграмм* (connectionless oriented model). При этом процесс может посылать сообщения (*дейтаграммы*) по произвольным адресам через один и тот же сокет без предварительного установления связи с этими адресами. Термин *дейтаграмма* (datagram) обозначает пакет пользовательского сообщения, посылаемый через сеть. Для облегчения понимания приведем аналогию: модель виртуальных соединений напоминает телефонную сеть, а модель дейтаграмм – пересылку писем по почте. Поэтому в последнем случае нельзя быть абсолютно уверенным, что сообщение дошло до адресата, а если необходимо получить ответ на него, то нужно указать свой обратный адрес на конверте. Модель соединений будет более подходящей при необходимости получения тесного взаимодействия между системами, когда обмен сообщениями и подтверждениями происходит в определенном порядке. Модель без соединений является более эффективной и лучше подходит в таких случаях, как рассылка широковещательных сообщений большому числу компьютеров.

Для того чтобы взаимодействие между процессами на разных компьютерах стало возможным, они должны быть связаны между собой как на аппаратном уровне при помощи сетевого оборудования – кабелей, сетевых карт и различных устройств маршрутизации, так и на программном уровне при помощи стандартного набора сетевых протоколов. Протокол представляет собой просто набор правил, в случае сетевого протокола – набор правил обмена сообщениями между компьютерами. Поэтому в системе UNIX должны существовать наборы правил для обеих моделей – как для модели соединений, так и для модели дейтаграмм. Для

модели соединений используется *протокол управления передачей* (Transmission Control Protocol, сокращенно TCP), а для модели дейтаграмм – *протокол пользовательских дейтаграмм* (User Datagram Protocol, сокращенно UDP).

5.3. Адресация

Чтобы процессы могли связаться по сети, должен существовать механизм определения *сетевого адреса* (network address) компьютера, на котором находится другой процесс. В конечном счете адрес определяет физическое положение компьютера в сети. Обычно адреса состоят из нескольких частей, соответствующих различным уровням сети. Далее будут затронуты только те вопросы, без ответов на которые не обойтись при программировании с использованием сокетов.

5.3.1. Адресация Internet

Сейчас почти во всех глобальных сетях применима *адресация IP* (сокращение от Internet Protocol – межсетевой протокол, протокол сети Интернет).

Адрес IP состоит из четырех десятичных чисел, разделенных точками, например:
`197.124.10.1`

Эти четыре числа содержат достаточную информацию для определения сети назначения, а также компьютера в этой сети; собственно, термин Internet и означает «сеть сетей».

Сетевые вызовы UNIX не могут работать с IP адресами в таком формате. На программном уровне IP адреса хранятся в структуре типа `in_addr_t`. Обычно программистам не нужно знать внутреннее представление этого типа, так как для преобразования IP адреса в структуру типа `in_addr_t` предназначена процедура `inet_addr`.

Описание

```
#include <winsock.h>

in_addr_t inet_addr(const char *ip_address);
```

Процедура `inet_addr` принимает IP адрес в форме строки вида `1.2.3.4` и возвращает адрес в виде структуры соответствующего типа. Если вызов процедуры завершается неудачей из-за неверного формата IP адреса, то возвращаемое значение будет равно `INADDR_NONE`, например:

```
in_addr_t server;

server = inet_addr("197.124.10.1");
```

Для того чтобы процесс мог ссылаться на адрес своего компьютера, определена постоянная `INADDR_ANY`, содержащая локальный адрес компьютера в формате `in_addr_t`.

Обратное преобразование адреса IP в текстовую строку можно при необходимости легко выполнить с помощью функции `inet_ntoa`, имеющей следующий прототип:

```
char FAR * inet_ntoa (struct in_addr  in);
```

При ошибке эта функция возвращает значение `NULL`.

Для хранения информации об адресе и порте адресата (абонента) существуют стандартные структуры. Обобщенная структура адреса сокета определяется следующим образом:

```
struct sockaddr
{
  sa_family_t sa_family;    /* Семейство адресов */
  char sa_data[];           /* Адрес сокета */
};
typedef struct sockaddr SOCKADDR;
typedef struct sockaddr *PSOCKADDR;
typedef struct sockaddr FAR *LPSOCKADDR;
```

Эта структура называется *обобщенным сокетом* (generic socket), так как в действительности применяются различные типы сокетов в зависимости от того, используются ли

они в качестве средства межпроцессного взаимодействия на одном и том же компьютере или для связи процессов через сеть. Сокеты для связи через сеть имеют следующую форму:

```
struct sockaddr_in
{
  sa_family_t      sin_family;   /* Семейство адресов */
  in_port_t        sin_port;     /* Номер порта */
  struct in_addr   sin_addr;     /* IP-адрес */
  unsigned char    sin_zero[8]; /* Поле выравнивания */
};
typedef struct sockaddr_in  SOCKADDR_IN;
typedef struct sockaddr_in  *PSOCKADDR_IN;
typedef struct sockaddr_in  FAR *LPSOCKADDR_IN;
```

Поле `sin_family` определяет тип адреса. Вы должны записать в это поле значение `AF_INET`, которое соответствует типу адреса, принятому в Internet:

```
srv_address.sin_family = AF_INET;
```

Поле `sin_port` определяет номер порта, который будет использоваться для передачи данных.

Особенностью поля `sin_port` является использование так называемого сетевого формата данных. Этот формат отличается от того, что принят в процессорах с архитектурой Intel, а именно, младшие байты данных хранятся по старшим адресам памяти. Напомним, что архитектура процессоров Intel подразумевает хранение старших байтов данных по младшим адресам.

Универсальный сетевой формат данных удобен при организации глобальных сетей, так как в узлах такой сети могут использоваться компьютеры с различной архитектурой.

Для выполнения преобразований из обычного формат в сетевой и обратно в интерфейсе Windows Sockets предусмотрен специальный набор функций. В частности, для заполнения поля `sin_port` нужно использовать функцию `htons`, выполняющую преобразование 16-разрядных данных из формата Intel в сетевой формат.

Ниже показано, как инициализируется поле `sin_port`:

```
#define SERV_PORT 5000
srv_address.sin_port = htons(SERV_PORT);
```

Вернемся снова к структуре `sockaddr_in`.

Поле `sin_addr` этой структуры представляет собой структуру `in_addr`:

```
struct in_addr
{
  union
  {
    struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
    struct { u_short s_w1,s_w2; } S_un_w;
    u_long S_addr;
  } S_un;
};
#define s_addr  S_un.S_addr
#define s_host  S_un.S_un_b.s_b2
#define s_net   S_un.S_un_b.s_b1
#define s_imp   S_un.S_un_w.s_w2
#define s_impno S_un.S_un_b.s_b4
#define s_lh    S_un.S_un_b.s_b3
```

При инициализации сокета в этой структуре вы должны указать адрес IP, с которым будет работать данный сокет.

Если сокет будет работать с любым адресом (например, вы создаете сервер, который будет доступен из узлов с любым адресом), адрес для сокета можно указать следующим образом:

```
srv_address.sin_addr.s_addr = INADDR_ANY;
```

В том случае, если сокет будет работать с определенным адресом IP (например, вы создаете приложение-клиент, которое будет обращаться к серверу с конкретным адресом IP), в указанную структуру необходимо записать реальный адрес.

Датаграммный протокол UDP позволяет посылать пакеты данных одновременно всем рабочим станциям в широковещательном режиме. Для этого вы должны указать адрес как `INADDR_BROADCAST`.

Однако чаще всего пользователь работает с доменными именами, используя сервер DNS или файл HOSTS. В этом случае вначале вы должны воспользоваться функцией `gethostbyname`, возвращающей адрес IP, а затем записать полученный адрес в структуру `sin_addr`:

```
PHOSTENT phe;
phe = gethostbyname ("ftp.microsoft.com");
if(phe == NULL)
{
  closesocket  (srv_socket);
  perror("gethostbyname  Error");
  return;
}
memcpy((char FAR *)&(dest_sin.sin_addr ), phe->h_addr, phe->h_length);
```

В случае ошибки функция `gethostbyname` возвращает `NULL`. При этом причину ошибки можно выяснить, проверив код возврата функции `WSAGetLastError`.

Если же указанный узел найден в базе DNS или в файле HOSTS, функция `gethostbyname` возвращает указатель на структуру `hostent`, описанную ниже:

```
struct  hostent
{
  char  FAR * h_name;             // имя узла
  char  FAR * FAR * h_aliases;    // список альтернативных имен
  short h_addr type;               // тип адреса узла
  short h_length;                 // длина адреса
  char  FAR * FAR * h_addr _list; // список адресов
#define h_addr   h_addr_list[0]   // адрес
};
typedef struct hostent  *PHOSTENT ;
typedef struct hostent  FAR *LPHOSTENT  ;
```

Искомый адрес находится в первом элемента списка `h_addr_list[0]`, на который можно также ссылаться при помощи `h_addr`. Длина поля адреса находится в поле `h_length`.

5.3.2. Порты

Кроме адреса компьютера, клиентская программа должна иметь возможность подключения к нужному серверному процессу. Серверный процесс ждет подключения к заданному *номеру порта* (port number). Поэтому клиентский процесс должен выполнить запрос на подключение к определенному порту на заданном компьютере. Если продолжить аналогию с пересылкой писем по почте, то это равносильно дополнению адреса номером комнаты или квартиры.

Некоторые номера портов по соглашению считаются отведенными для стандартных сервисов, таких как `ftp` или `rlogin`. Эти номера портов записаны в файле `/etc/services`. В общем случае порты с номерами, меньшими 1024, считаются зарезервированными для системных процессов UNIX. Все остальные порты доступны для пользовательских процессов.

5.4. Интерфейс сокетов

В процессе инициализации приложение должно зарегистрировать себя в библиотеке WSOCK32.DLL, которая предоставляет приложениям интерфейс Windows Sockets в среде операционных систем Microsoft Windows 95 и Microsoft Windows NT.

Для инициализации необходимо вызвать функцию `WSAStartup`, определенную следующим образом:

```
int WSAStartup (WORD wVersionRequested, LPWSADATA  lpWSAData);
```

В параметре `wVersionRequested` вы должны указать версию интерфейса Windows Sockets, необходимую для работы вашего приложения. Старший байт параметра указывает младший номер версии (minor version), младший байт – старший номер версии (major version).

Перед вызовом функции `WSAStartup` параметр `lpWSAData` должен содержать указатель на структуру типа `WSADATA`, в которую будут записаны сведения о конкретной реализации интерфейса Windows Sockets.

В случае успеха функция `WSAStartup` возвращает нулевое значение. Если происходит ошибка, возвращается одно из следующих значений:

Значение	Описание
`WSASYSNOTREADY`	Сетевое программное обеспечение не готово для работы
`WSAVERNOTSUPPORTED`	Функция не поддерживается данной реализацией интерфейса Windows Sockets
`WSAEINVAL`	Библиотека DLL, обеспечивающая интерфейсе Windows Sockets, не соответствует версии, указанной приложением указанной в параметре wVersionRequested

Ниже представлен фрагмент исходного текста, выполняющий инициализацию интерфейса Windows Sockets:

```
rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
if(rc != 0)
{
  perror("WSAStartup  Error");
  return 1;
}

// Отображаем описание и версию системы Windows Sockets
printf("Server use %s %s", WSAData.szDescription,WSAData.szSystemStatus);
```

В операционных системах Microsoft Windows 95 и Microsoft Windows NT версии 3.51 встроена система Windows Sockets версии 1.1, поэтому именно это значение указано при вызове функции `WSAStartup`.

Определение структуры `WSADATA` и указателя на нее выглядят следующим образом:

```
typedef struct WSAData
{
  WORD           wVersion;
  WORD           wHighVersion;
  char           szDescription[WSADESCRIPTION_LEN+1];
  char           szSystemStatus[WSASYS_STATUS_LEN+1];
  unsigned short iMaxSockets;
  unsigned short iMaxUdpDg;
  char FAR *     lpVendorInfo;
} WSADATA ;
typedef WSADATA  FAR *LPWSADATA;
```

Использованные выше поля `szDescription` и `szSystemStatus` после вызова функции `WSAStartup` содержат, соответственно, описание конкретной реализации интерфейса Windows Socket и текущее состояние этого интерфейса в виде текстовых строк.

В полях `wVersion` и `wHighVersion` записаны, соответственно, версия спецификации

Windows Socket, которую будет использовать приложение, и версия спецификации, которой соответствует конкретная реализация интерфейса Windows Socket.

Приложение может одновременно создавать несколько сокетов, например, для использования в разных подзадачах одного процесса. В поле `iMaxSockets` хранится максимальное количество сокетов, которое можно получить для одного процесса.

В поле `iMaxUdpDg` записан максимальный размер пакета данных, который можно переслать с использованием датаграммного протокола UDP.

И, наконец, поле `lpVendorInfo` содержит указатель на дополнительную информацию, формат которой зависит от фирмы-изготовителя конкретной реализации системы Windows Sockets.

Перед тем, как завершить свою работу, приложение должно освободить ресурсы, полученные у операционной системы для работы с Windows Sockets. Для выполнения этой задачи приложение должно вызвать функцию `WSACleanup`, определенную так, как это показано ниже:

```
int WSACleanup (void);
```

Эта функция может возвратить нулевое значение при успехе или значение `SOCKET_ERROR` в случае ошибки.

Для получения кода ошибки вы должны воспользоваться функцией с именем `WSAGetLastError`:

```
int WSAGetLastError (void);
```

Функция `WSAGetLastError` позволяет определить код ошибки при неудачном завершении практически всех функций интерфейса Windows Sockets. Вы должны вызывать ее сразу вслед за функцией, завершившейся неудачно.

Если ошибка возникла при выполнении функции `WSACleanup`, функция `WSAGetLastError` может вернуть одно из следующих значений:

Значение	Описание
`WSANOTINITIALISED`	Интерфейс Windows Sockets не был проинициализирован функцией `WSAStartup`
`WSAENETDOWN`	Сбой сетевого программного обеспечения
`WSAEINPROGRESS`	Во время вызова функции `WSACleanup` выполнялась одна из блокирующих функций интерфейса Windows Sockets

Сделаем небольшие пояснения относительно последней ошибки, приведенной в этом списке, и имеющей код `WSAEINPROGRESS`.

Некоторые функции интерфейса Windows Sockets способны блокировать работу приложения, так как они не возвращают управление до своего завершения. В операционных системах, использующих вытесняющую мультизадачность, к которым относятся Microsoft Windows 95 и Microsoft Windows NT, это не приводит к блокировке всей системы. Как вы увидите дальше, можно избежать использования блокирующих функций, так как для них в интерфейсе Windows Sockets существует замена.

5.4.1. Создание сокета

При любых моделях связи клиент и сервер должны создать *абонентские точки* (transport end points,), или сокеты, которые являются дескрипторами, используемыми для установки связи между процессами в сети. Они создаются при помощи системного вызова `socket`.

Описание

```
int socket(int domain, int type, int protocol);
```

Параметр `domain` определяет коммуникационный домен, в котором будет использоваться сокет. Например, значение `AF_INET` определяет, что будет использоваться домен Internet. Интерес может представлять также другой домен, `AF_UNIX`, который используется,

если процессы находятся на одном и том же компьютере.

Параметр `type` определяет тип создаваемого сокета. Значение `SOCK_STREAM` указывается при создании сокета для работы в режиме виртуальных соединений, а значение `SOCK_DGRAM` – для работы в режиме пересылок дейтаграмм. Последний параметр `protocol` определяет используемый протокол. Этот параметр обычно задается равным нулю, при этом по умолчанию сокет типа `SOCK_STREAM` будет использовать протокол TCP, а сокет типа `SOCK_DGRAM` – протокол UDP. Оба данных протокола являются стандартными протоколами UNIX. Поэтому виртуальное соединение часто называют TCP-соединением, а пересылку дейтаграмм – работой с UDP-сокетами.

Системный вызов `socket` обычно возвращает неотрицательное целое число, которое является дескриптором файла сокета, что позволяет считать механизм сокетов разновидностью обобщенного файлового ввода/вывода UNIX.

5.5. Программирование в режиме TCP-соединения

Для того чтобы продемонстрировать основные системные вызовы для работы с сокетами, рассмотрим пример, в котором клиент посылает серверу поток строчных символов через TCP-соединение. Сервер преобразует строчные символы в прописные и посылает их обратно клиенту. В следующих разделах этой работы приведем тот же самый пример, но использующий сокеты UDP-протокола.

Сначала составим план реализации серверного процесса:

```
/* Серверный процесс */

/* Включает нужные заголовочные файлы */

#include <ctype.h>
#include <stdio.h>

main ()
{
  int sockfd;

  /* Установливает абонентскую точку сокета */
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("Ошибка вызова socket");
    exit (1);
  }

/* "Связывание" адреса сервера с сокетом

Ожидание подключения

Цикл
  установка соединения
  создание дочернего процесса для работы с соединением
  если это дочерний процесс,
  то нужно в цикле принимать данные от клиента и посылать ему ответы
*/
}
```

План клиентского процесса выглядит следующим образом:

```
/* Клиентский процесс */

/* Включает нужные заголовочные файлы */
```

```
#include <ctype.h>
#include <winsock.h>

main ()
{
  int sockfd;

  /* Создает сокет */
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("Ошибка вызова socket");
    exit (1);
  }

  /* Соединяет сокет с адресом серверного процесса */
  /* В цикле посылает данные серверу и принимает от него ответы */
}
```

Далее будем постепенно превращать эти шаблоны в настоящие программы, начиная с реализации сервера.

5.5.1. Связывание

Системный вызов `bind` связывает сетевой адрес компьютера с идентификатором сокета.

Описание

```
#include <winsock.h>
int bind (int sockfd, const struct sockaddr *address,
          size_t add_len);
```

Первый параметр, `sockfd`, является дескриптором файла сокета, созданным с помощью вызова `socket`, а второй – указателем на обобщенную структуру адреса сокета. В рассматриваемом примере данные пересылаются по сети, поэтому в действительности в качестве этого параметра будет задан адрес структуры `sockaddr_in`, содержащей информацию об адресе нашего сервера. Последний параметр содержит размер указанной структуры адреса сокета. В случае успешного завершения вызова `bind` он возвращает значение 0. В случае ошибки, например, если сокет для этого адреса уже существует, вызов `bind` возвращает значение -1. Переменная `errno` будет иметь при этом значение `EADDRINUSE`.

5.5.2. Включение приема TCP-соединений

После выполнения связывания с адресом и перед тем, как какой-либо клиент сможет подключиться к созданному сокету, сервер должен включить прием соединений. Для этого служит вызов `listen`.

Описание

```
#include <winsock.h>
int listen(int sockfd, int queue_stze);
```

Параметр `sockfd` имеет то же значение, что и в предыдущем вызове. В очереди сервера может находиться не более `queue_size` запросов на соединение. (Спецификация XSI определяет минимальное ограничение сверху на длину очереди равное пяти.)

5.5.3. Прием запроса на установку TCP-соединения

Когда сервер получает от клиента запрос на соединение, он должен создать новый сокет для работы с новым соединением. Первый же сокет используется только для установки соединения. Дополнительный сокет создается при помощи вызова `accept`, принимающего очередное соединение.

Описание

```
#include <winsock.h>
int accept(int sockfd, struct sockaddr *address, size_t *add_len);
```

Системному вызову `accept` передается дескриптор сокета, для которого ведется прием соединений. Возвращаемое значение соответствует идентификатору нового сокета, который будет использоваться для связи. Параметр `address` заполняется информацией о клиенте. Так как связь использует соединение, адрес клиента знать не обязательно, поэтому можно присвоить параметру `address` значение `NULL`. Если значение `address` не равно `NULL`, то переменная, на которую указывает параметр `add_len`, первоначально должна содержать размер структуры адреса, заданной параметром `address`. После возврата из вызова `accept` переменная `*add_len` будет содержать реальный размер записанной структуры.

После подстановки вызовов `bind`, `listen` и `accept` текст программы сервера примет вид:

```
/* Серверный процесс */

#include <ctype.h>
#include <winsock.h>
#include <stdio.h>

#define SIZE sizeof(struct sockaddr_in)

main ()
{
  int sockfd;
  int rc;
  WSADATA WSAData;
  int newsockfd, size;
  /* Инициализация сокета Internet с номером порта 7000
   * и локальным адресом, заданным в постоянной INADDR_ANY */
  struct sockaddr_in server = {AF_INET, 7000, INADDR_ANY};
  struct sockaddr_in client;

  rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
  if(rc != 0)
  {
    perror("WSAStartup  Error");
    return 1;
  }

  /* Создает сокет */
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("Ошибка вызова socket");
    exit (1);
  }

  /* Связавает адрес с сокетом */
  if (bind (sockfd, (struct sockaddr *) &server, SIZE) == -1)
  {
    perror ("Ошибка вызова bind");
    exit (1);
  }

  /* Включает прием соединений */
  if (listen (sockfd, 5) == -1)
  {
```

```
    perror ("ошибка вызова listen");
    exit (1);
  }

  for (;;)
  {
    /* Принимает очередной запрос на соединение */
    if ((newsockfd = accept (sockfd, (struct sockaddr *) &client, &size))
         == -1)
    {
      perror ("Ошибка вызова accept");
      continue;
    }
  /*
    Создает дочерний процесс для работы с соединением.
    Если это дочерний процесс,
      то в цикле принимает данные от клиента
      и посылает ему ответы.
  */
  }
}
```

Обратите внимание на то, что сервер использует константу `INADDR_ANY`, соответствующую адресу локального компьютера.

Теперь имеется серверный процесс, способный переходить в режим приёма соединений и принимать запросы на установку соединений. Рассмотрим, как клиент должен обращаться к серверу.

5.5.4. Подключение клиента

Для выполнения запроса на подключение к серверному процессу клиент использует системный вызов `connect`.

Описание

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int csockfd, const struct sockaddr *address,
            size_t add_len);
```

Первый параметр `csockfd` является дескриптором сокета клиента и не имеет отношения к дескриптору сокета на сервере. Параметр `address` указывает на структуру, содержащую адрес сервера, параметр `add_len` определяет размер используемой структуры адреса.

Продолжая составление рассматриваемого примера, запишем следующий вариант текста программы клиента:

```
/* Клиентский процесс */

#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SIZE sizeof(struct sockaddr_in)

main ()
{
  int sockfd;
  int c, rc;
  struct sockaddr_in server = {AF_INET, 7000};
  WSADATA WSAData;
```

```
  rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
  if(rc != 0)
  {
    perror("WSAStartup  Error");
    return 1;
  }

  /* Преобразовавает и записывает IP адрес сервера */
  server.sin_addr.s_addr = inet_addr ("127.0.0.1");

  /* Слздать сокет */
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("Ошибка вызова socket");
    exit (1);
  }

  /* Соединяет сокет с сервером */
  if (connect (sockfd, (struct sockaddr *) &server, SIZE) == -1)
  {
    perror ("Ошибка вызова connect");
    exit (1);
  }

  /* Обмен данными с сервером */
}
```

Адрес сервера преобразуется в нужный формат при помощи вызова `inet_addr`. Адреса известных компьютеров локальной сети обычно можно найти в файле `/etc/hosts`.

5.5.5. Пересылка данных

Теперь уже освоена процедура установления соединения между клиентом и сервером. Для сокетов типа `SOCK_STREAM` и клиент, и сервер получают дескрипторы файлов, которые могут использоваться для чтения или записи. В большинстве случаев для этого годятся обычные вызовы `read` и `write`. Если же необходимо задавать дополнительные параметры пересылки данных по сети, то можно использовать два новых системных вызова – `send` и `recv`. Эти вызовы имеют схожий интерфейс и ведут себя точно так же, как вызовы `read` и `write`, если их четвертый аргумент равен нулю.

Описание

```
#include <winsock.h>

ssize_t recv(int sockfd, void *buffer, size_t length,
             int flags);
ssize_t send(int sockfd, const void *buffer, size_t length,
             int flags);
```

Вызов `recv` имеет четыре параметра: дескриптор файла `filedes`, из которого читаются данные, буфер `buffer`, в который они помещаются, размер буфера `length` и поле флагов `flags`.

Параметр `flags` указывает дополнительные опции получения данных. Его возможные значения определяются комбинациями следующих констант:

`MSG_PEEK` Процесс может просматривать данные, не «получая» их

`MSG_OOB` Обычные данные пропускаются. Процесс принимает только срочные данные, например, сигнал прерывания

`MSG_WAITALL` Возврат из вызова `recv` произойдет только после получения всех данных

При аргументе `flags` равном нулю вызов `send` работает точно так же, как и вызов

write, пересылая массив данных буфера buffer в сокет sockfd. Параметр length задает размер массива данных. Аналогично вызову recv параметр flags определяет опции передачи данных. Его возможные значения определяются комбинациями следующих констант:

MSG_OOB	Передать *срочные* (out of band) данные
MSG_DONTROUTE	При передаче сообщения игнорируются условия маршрутизации протокола более низкого уровня. Обычно это означает, что сообщение посылается по прямому, а не по самому быстрому маршруту (самый быстрый маршрут не обязательно прямой и может зависеть от текущего распределения нагрузки сети)

Теперь с помощью этих вызовов можно реализовать обработку данных на серверной стороне:

```
/* Серверный процесс */

main ()
{
  /* Приведенная выше инициализация сокета */
  .
  .
  .
  char c;

  for (;;)
  {
    /* Принимает запрос на установку соединения */
    if ((newsockfd = accept (sockfd, (struct sockaddr *) &client, &size))
        == -1)
    {
      perror ("Ошибка вызова accept");
      continue;
    }

    /* Создает дочерний процесс для работы с соединением */
    CreateThread(NULL,0,child,&newsockfd,0,&child_id);
  }
}


DWORD WINAPI child(void *param)
{
  char c;
  int newsockfd;

  printf("Run child\n");
  newsockfd=*(int *)param;
  while (recv (newsockfd, &c, 1, 0) > 0)
  {
     c = toupper (c);
     send (newsockfd, &c, 1, 0);
  }
  /* После того, как клиент прекратит передачу данных,
   * сокет может быть закрыт и дочерний процесс
   * завершает работу */
  closesocket (newsockfd);
  printf("child exited\n");
  ExitThread(0);
  return 0;
}
```

Использование вызова CreateThread позволяет серверу обслуживать несколько клиентов. Цикл работы клиентского процесса может быть реализован так:

```
/* Клиентский процесс */

main ()
{
  int sockfd;
  char c, rc;

  /* Приведенная выше инициализация сокета и запрос
   * на установку соединения */

  /* Обмен данными с сервером */
  for (rc = '\n';;)
  {
    if (rc == '\n')
      printf ("Введите строчный символ\n");
    c = getchar ();
    send (sockfd, &c, 1, 0);
    recv (sockfd, &rc, 1, 0);
    printf ("%c", rc);
  }
}
```

5.5.6. Закрытие TCP-соединения

При работе с сокетами важно корректно реагировать на завершение работы абонентского процесса. Так как сокет является двусторонним механизмом связи, то нельзя предсказать заранее, когда произойдет разрыв соединения – во время чтения или записи. Поэтому нужно учитывать оба возможных варианта.

В случае разорванной связи вызов read или recv возвращает нулевое значение. Поэтому для вызовов read и recv необходимо всегда проверять возвращаемое значение, чтобы не зациклиться при приеме данных.

Закрываются сокеты при помощи системного вызова closesocket. Для сокета типа SOCK_STREAM ядро гарантирует, что все записанные в сокет данные будут переданы принимающему процессу. Это может вызвать блокирование операции закрытия сокета до тех пор, пока данные не будут доставлены. (Если сокет имеет тип SOCK_DGRAM, то сокет закрывается немедленно.)

Теперь можно привести полный текст примера клиента и сервера, добавив в серверный процесс обработку сигналов и вызов closesocket в обе программы. В данном случае эти меры могут показаться излишними, но в реальном клиент/серверном приложении обязательна надежная обработка всех исключительных ситуаций. Приведем окончательный текст программы сервера:

```
/* Серверный процесс */

#include <ctype.h>
#include <winsock.h>
#include <stdio.h>

#define SIZE sizeof(struct sockaddr_in)

DWORD WINAPI child(void *param);

main ()
{
  int sockfd;
```

```
  int rc;
  struct sockaddr_in server = {AF_INET, 7000, INADDR_ANY}, client;
  WSADATA WSAData;
  int newsockfd, size;

  DWORD child_id;

  rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
  if(rc != 0)
  {
    perror("WSAStartup  Error");
    return 1;
  }


  /* Создает сокет */
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("socket call error");
    exit (1);
  }

  /* Связает адрес с сокетом */
  if (bind (sockfd, (struct sockaddr *) &server, SIZE) == -1)
  {
    perror ("bind call error");
    exit (1);
  }

  /* Включает прием соединений */
  if (listen (sockfd, 5) == -1)
  {
    perror ("listen call error");
    exit (1);
  }

  for (;;)
  {
    /* Принимает запрос на соединение */
    if ((newsockfd = accept (sockfd, (struct sockaddr *) &client, &size))
== -1)
    {
      perror ("accept call error");
      continue;
    }

    printf("Request from %s, processing\n", inet_ntoa(client.sin_addr));
    /* Создает дочерний процесс для работы с соединением */
    CreateThread(NULL,0,child,&newsockfd,0,&child_id);
  }
  //WSACleanup();
}

DWORD WINAPI child(void *param)
{
  char c;
  int newsockfd;
```

```
  printf("Run child\n");
  newsockfd=*(int *)param;
  while (recv (newsockfd, &c, 1, 0) > 0)
  {
     c = toupper (c);
     send (newsockfd, &c, 1, 0);
  }
  /* После того, как клиент прекратит передачу данных,
   * сокет может быть закрыт и дочерний процесс
   * завершает работу */
  closesocket (newsockfd);
  printf("child exited\n");
  ExitThread(0);
  return 0;
}
```

И клиента:

```
/* Клиентский процесс */

#include <ctype.h>
#include <stdio.h>
#include <winsock.h>

#define SIZE sizeof(struct sockaddr_in)

main ()
{
  int sockfd;
  int c, rc;
  struct sockaddr_in server = {AF_INET, 7000};
  WSADATA WSAData;

  rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
  if(rc != 0)
  {
    perror("WSAStartup  Error");
    return 1;
  }

  /* Преобразовывает и сохраняет IP адрес сервера */
  server.sin_addr.s_addr = inet_addr ("127.0.0.1");

  /* Создает сокет */
  if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) == -1)
  {
    perror ("socket call error");
    exit (1);
  }

  /* Соединяет сокет с сервером */
  if (connect (sockfd, (struct sockaddr *) &server, SIZE) == -1)
  {
    perror ("connect call error");
    exit (1);
  }

  /* Цикл обмена данными с сервером */
  for (rc = '\n';;)
```

```
  {
    if (rc == '\n')
      printf ("Input characters\n");
    c = getchar ();
    if (c==-1)
      break;
    send (sockfd, (char*)&c, 1, 0);
    if (recv (sockfd, (char*)&rc, 1, 0) > 0)
      printf ("%c", rc);
    else
    {
      printf ("Server is not response\n");
      break;
    }
  }
  closesocket (sockfd);
  WSACleanup();
  return 1;
}
```

5.6. Программирование в режиме пересылок UDP-дейтаграмм

Перепишем теперь пример, используя модель дейтаграмм. Основное отличие будет заключаться в том, что дейтаграммы (UDP-пакеты), передаваемые между клиентом и сервером, могут достигать точки назначения в произвольном порядке. К тому же, как уже упоминалось, протокол UDP не гарантирует доставку пакетов. При работе с UDP-сокетами процесс клиента должен также сначала создать сокет и связать с ним свой локальный адрес при помощи вызова `bind`. После этого процесс сможет использовать этот сокет для посылки и приема UDP-пакетов. Чтобы послать сообщение, процесс должен знать адрес назначения, который может быть как конкретным адресом, так и шаблоном, называемым «широковещательным адресом» и обозначающим сразу несколько компьютеров.

5.6.1. Прием и передача UDP-сообщений

Для сокетов UDP есть два новых системных вызова – `sendto` и `recvfrom`. Параметр `sockfd` в обоих вызовах задает связанный с локальным адресом сокет, через который принимаются и передаются пакеты.

Описание

```
ssize_t recvfrom(int sockfd, void *message, size_t length,
                 int flags, struct sockaddr *send_addr,
                 size_t *add_len);
ssize_t sendto(int sockfd, const void *message, size_t length,
               int flags, const struct sockaddr *dest_addr,
               size_t dest_len);
```

Если параметр `send_addr` равен `NULL`, то вызов `recvfrom` работает точно так же, как и вызов `recv`. Параметр `message` указывает на буфер, в который помещается принимаемая дейтаграмма, а параметр `length` задает число байтов, которые должны быть считаны в буфер. Параметр `flags` принимает те же самые значения, что и в вызове `recv`. Два последних параметра помогают установить двустороннюю связь с помощью UDP-сокета. В структуру `send_addr` будет помещена информация об адресе и порте, откуда пришел прочитанный пакет. Это позволяет принимающему процессу направить ответ пославшему пакет процессу. Последний параметр является указателем на целочисленную переменную типа `size_t`, в которую помещается длина записанного в структуру `send_addr` адреса.

Вызов `sendto` противоположен вызову `recvfrom`. В этом вызове параметр `dest_addr` задает адрес узла сети и порт, куда должно быть передано сообщение, а параметр

dest_len определяет длину адреса.

Адаптируем пример для модели дейтаграммных посылок.

```
/* Сервер */

#include <ctype.h>
#include <winsock.h>
#include <stdio.h>

#define SIZE sizeof(struct sockaddr_in)

main ()
{
  int sockfd;
  char c, rc;
  WSADATA WSAData;
  /* Локальный серверный порт */
  struct sockaddr_in server = {AF_INET, 7000, INADDR_ANY};
  /* Структура, которая будет содержать адрес */
  /* клиентского процесса */
  struct sockaddr_in client;
  int client_len = SIZE;

  rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
  if(rc != 0)
  {
    perror("WSAStartup  Error");
    return 1;
  }

  /* Создает сокет */
  if ((sockfd = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
  {
    perror ("socket call error");
    exit (1);
  }

  /* Связает локальный адрес с сокетом */
  if (bind (sockfd, (struct sockaddr *) &server, SIZE) == -1)
  {
    perror ("bind call error");
    exit (1);
  }

  /* Бесконечный цикл ожидания сообщений */
  for (;;)
  {
    /* Принимает сообщение и записывает адрес клиента */
    if (recvfrom (sockfd, &c, 1, 0, &client, &client_len) == -1)
    {
      perror ("Server: recvfrom error");
      continue;
    }
    printf("Request from %s: %c\n", inet_ntoa(client.sin_addr), c);

    c = toupper (c);

    /* Посылает сообщение обратно */
    if (sendto (sockfd, &c, 1, 0, &client, client_len) == -1)
```

```
    {
      perror ("Server: sendto error");
      continue;
    }

  }
  //WSACleanup();
}
```

Новый текст клиента:

```
/* Клиентский процесс */

#include <ctype.h>
#include <winsock.h>
#include <io.h>
#include <stdio.h>

#define SIZE sizeof(struct sockaddr_in)

main ()
{
  int sockfd;
  int c,rc;
  WSADATA WSAData;

  /* Локальный порт на клиенте */
  struct sockaddr_in client = {AF_INET, INADDR_ANY, INADDR_ANY};

  /* Адрес удаленного сервера */
  struct sockaddr_in server = {AF_INET, 7000};

  /* Преобразовывает и записыват IP адрес */
  server.sin_addr.s_addr = inet_addr ("127.0.0.1");

  rc = WSAStartup (MAKEWORD(1, 1), &WSAData);
  if(rc != 0)
  {
    perror("WSAStartup  Error");
    return 1;
  }

  /* Создает сокет */
  if ((sockfd = socket (AF_INET, SOCK_DGRAM, 0)) == -1)
  {
    perror ("socket call error");
    exit (1);
  }

  /* Связывает локальный адрес с сокетом */
  if (bind (sockfd, (struct sockaddr *) &client, SIZE) == -1)
  {
    perror ("bind call error");
    exit (1);
  }

  /* Считывает символ с терминала */
  while (read (0, &c, 1) > 0)
  {
```

```
    /* Передает символ серверу */
    if (sendto (sockfd, &c, 1, 0, &server, SIZE) == -1)
    {
      perror ("Client: sendto error");
      continue;
    }

    /* Принимает вернувшееся сообщение */
    if (recv (sockfd, &c, 1, 0) == -1)
    {
      perror ("Client: recv error");
      continue;
    }

    write (1, &c, 1);
  }
  WSACleanup();
  return 0;
}
```

5.7. Различия между двумя моделями

Обсудим различия между двумя реализациями рассматриваемого примера с точки зрения техники программирования.

В обеих моделях сервер должен создать сокет и связать свой локальный адрес с этим сокетом. В модели TCP-соединений серверу следует после этого включить прием соединений. В модели UDP-сокетов этот шаг не нужен, зато на клиента возлагается больше обязанностей.

С точки зрения клиента в модели TCP-соединений достаточно простого подключения к серверу. В модели UDP-сокетов клиент должен создать сокет и связать свой локальный адрес с этим сокетом.

И, наконец, для передачи данных обычно используются различные системные вызовы. Системные вызовы `sendto` и `recvfrom` могут использоваться в обеих моделях, но все же они обычно используются в UDP-модели, чтобы сервер мог получить информацию об отправителе и отправить обратно ответ.

Приложение. Лабораторная работа 5. Интерфейс сокетов

Код демонстрационного Web-сервера под платформу POSIX:

```
/*
    Minimal webserver.c by Pieter Suurmond, februari 4, 2003. No
copyrights.
    Updated november 14, 2003 (<sys/types.h> remark).
    Compile with a UNIX C compiler with POSIX threads, for example with
GNU GCC:
     gcc -Wall -fomit-frame-pointer -ffast-math -O2 -D_REENTRANT \
     mtserver.c -o mtserver -lpthread
    Startup './mtserver' and watch 'http://localhost:1234/BLOOP/' in a
web-
    browser. One can shutdown with something like
'http://localhost:1234/Q'.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <signal.h>         /* For sigprocmask(). */
#include <pthread.h>        /* POSIX threads. */
#include <sys/types.h>      /* For directory-reading and other stuff but
also for */
                            /* bind and accept (thanks to Sam). */
#include <dirent.h>        /* File dir.h for 4.3BSD, dirent.h for System
V versions. */
#include <sys/stat.h>       /* For S_ISDIR(st.st_mode) */

/*-------------------------------------------------------------------------
----------*/
typedef struct                      /* Data common to all threads.
*/
{
    char*           software;       /* Pointer to string with name and
version.    */
    long            in_size;        /* Number of unsigned chars for input
buffer.  */
    long            out_size;       /* Number of unsigned chars for
output buffer. */
    unsigned char   up;             /* Shared memery for quitting (make
it 0).     */
}   GLOBAL_DATA;

/*-------------------------------------------------------------------------
----------*/
typedef struct                      /* Such a structure is passed during
spawning. */
{
    GLOBAL_DATA*    global_data;
```

```
    unsigned char*  in_buff;        /* Allocated and released by parent.
*/
    unsigned char*  out_buff;       /* Allocated and released by parent.
*/
    int             socket;         /* Opened by parent, closed by child.
*/
    unsigned char   active;         /* Not active if 0, otherwise active.
*/
}   MT_SERVICE;

/*-------------------------------------------------------------------------
---*/
int selFunc_dotfilter(struct dirent* de) /* Skip names that begin with
dot. */
{
    if (*de->d_name != '.')
        return(1);
    return(0);
}

/*-------------------------------------------------------------------------*/
int genIndex(char* path, MT_SERVICE* s)
{
    int             dlen, nn, totlen, n, num, result = 0;
    struct dirent** handle = NULL;
    struct stat     statBuf;
    char            *longPath, *what;

    num = scandir(path, &handle, selFunc_dotfilter, alphasort);
    if ((num < 0) || (!handle))             /* Test both to be sure. */
        return -1;
    n = sprintf(s->out_buff, "Index of %s:\n\n", path);
    if (n != send(s->socket, s->out_buff, n, 0))
        { result = -2; goto done; }

    dlen = 1 + strlen(path);        /* 1 extra for separator '/' in
pathnames. */
    totlen = dlen + 128;            /* Guess (and check!) 128 chars for
dir/file-name. */
    longPath = malloc(1 + totlen);  /* 1 extra for terminating '\0'. */
    if (longPath)
        {
        for (n=0; n<num; n++)
            {
            if ((dlen + strlen(handle[n]->d_name)) > totlen)
                { result = -3; goto done; }
            sprintf(longPath, "%s/%s", path, handle[n]->d_name);
            if (stat(longPath, &statBuf))
                { result = -4; goto done; }
            if (S_ISDIR(statBuf.st_mode))
                what = "DIR ";
            else
                what = "FILE";
            nn = sprintf(s->out_buff, " %s %s\n", what, handle[n]-
>d_name);
            if (nn != send(s->socket, s->out_buff, nn, 0))
                { result = -5; goto done; }
            }
```

```
        free(longPath);
        }
    else
        result = -6;
done:
    for (n=0; n<num; n++)
          free(handle[n]);
    free(handle); /* And the array itself. */
    return result;
}

/*--------------------------------------------------------------------*/
typedef struct          /* Using names like CGI environment variables. */
{
    char    *REQUEST_METHOD,
            *SCRIPT_NAME,
            *SERVER_PROTOCOL,          /* What the client's requested. */
            *remains;
}   RQ;

/*--------------------------------------------------------------------*/
static int parseInput(char* in, RQ* out) /* Arguments may not be NULL. */
{
    out->REQUEST_METHOD = in;   /* Returns < 0 in case of failure, */
    in = strchr(in, ' ');       /* 0 for request, +1 for index. */
    if (!in)
        return -1;
    *in++ = (char)0;

    out->SCRIPT_NAME = in;
    in = strchr(in, ' ');
    if (!in)
        return -2;
    *in++ = (char)0;

    if ('/' != out->SCRIPT_NAME[0])             /* SCRIPT_NAME must start
with '/' */
        return -3;                              /* (so at least one char
there).   */

    if (!out->SCRIPT_NAME[1]) /* Request ""--> +1 */
        return 1;

    if (strchr(".,/\\", out->SCRIPT_NAME[1]))   /* Name may not start
with dot, slash... */
        return -4;                              /* True if SCRIPT_NAME[1]
== '\0'. */

    out->SERVER_PROTOCOL = in;
    while (*in)
        {
        if ((*in == 10) || (*in == 13))
            {
            *in++ = (char)0;
            out->remains = in;

            if (strcasecmp("HTTP/1.0", out->SERVER_PROTOCOL) &&
                strcasecmp("HTTP/1.1", out->SERVER_PROTOCOL))
```

```
                return -5;

            return 0;
            }
        in++;
        }
    return -6;
}

/*-------------------------------------------------------------------------
----------*/
void* childServer(void* service)
{
    FILE*           diag = stdout;
    MT_SERVICE*     s = (MT_SERVICE*)service;
    RQ              request;
    int             n, num;
    char            *msg = "?", *bye = "";

    num = recv(s->socket, s->in_buff, (s->global_data->in_size)-1, 0);
    if (num < 0)                        /* One less for termination. */
        {
        msg = "Recv error!"; goto childFinish;
        }
#if 0
    if (num >= s->global_data->in_size) /* Should never occur. */
        {
        msg = "Weird recv error!"; goto childFinish;
        }
#endif
    s->in_buff[num] = (char)0;                  /* Make it a C-string. */
    num = parseInput(s->in_buff, &request);
    if (num < 0)
        {
        fprintf(diag, "Bad request: parseInput() = %d.\n", num);
        n = sprintf(s->out_buff, "\
HTTP/1.0 400 Bad request\nServer: %s\nContent-type: text/plain\n\n\
parseInput() = %d\n", s->global_data->software, num);
        if (n != send(s->socket, s->out_buff, n, 0))
            { msg = "Send error!"; goto childFinish; }
        }
    else if (strcasecmp(request.REQUEST_METHOD, "GET")) /* Block anything
else. */
        {
        fprintf(diag, "Request method not allowed: %s.\n",
request.REQUEST_METHOD);
        n = sprintf(s->out_buff, "\
%s 401 Unauthorized\nServer: %s\nContent-type: text/plain\n\n\
Not allowed REQUEST_METHOD: %s\n", request.SERVER_PROTOCOL, s-
>global_data->software,
                                    request.REQUEST_METHOD);
        if (n != send(s->socket, s->out_buff, n, 0))
            { msg = "Send error!"; goto childFinish; }
        }
    else if (num/*==1*/)
        {
        n = sprintf(s->out_buff, "%s 200 OK\nServer: %s\nContent-type:
text/plain\n\n",
```

```
                    request.SERVER_PROTOCOL, s->global_data->software);
        if (n != send(s->socket, s->out_buff, n, 0))
            { msg = "Send error!"; goto childFinish; }
        if (genIndex("./", s))
            { msg = "GenIndex error!"; goto childFinish; }
        }
    else /* SCRIPT_NAME[0] == '/' */
        {
        if ((request.SCRIPT_NAME[1] == 'q') || (request.SCRIPT_NAME[1] ==
'Q'))
            bye = ", YOU ARE KILLING ME (the next request I'll die)";
                                                    /* Output complete
header: */
        n = sprintf(s->out_buff, "\
%s 200 OK\nServer: %s\nContent-type: text/plain\n\n\
Hi%s!\nI am %s.\n\
You sent me this:\n\n\
REQUEST_METHOD = %s\n\
SCRIPT_NAME = %s\n\
SERVER_PROTOCOL = %s\n\
remains = \n%s\n", request.SERVER_PROTOCOL, s->global_data->software,
                   bye, s->global_data->software, request.REQUEST_METHOD,
                                                   request.SCRIPT_NAME,

request.SERVER_PROTOCOL, request.remains);
        if (n != send(s->socket, s->out_buff, n, 0))
            { msg = "Send error!"; goto childFinish; }
        }
    fprintf(diag, "Served ok.\n");
childFinish:
    close(s->socket);   /* Close socket created by accept() in parent. */
    if (*bye)
        s->global_data->up = 0;     /* Signal to parent to quit. */
    s->active = 0;                  /* signal to parent we're (almost?)
dead. */
    return (void*)NULL;
}

/*--------------------------------------------------------------------
----------*/
int main(void)
{                                                   /* Configuration:
*/
    char*               software = "mtserver 0.20";
    int                 port     = 1234;            /* port number
for server. */
    FILE*               diag     = stdout;          /* diag may not
be NULL.   */
    long                in_size  = 4096;
    long                out_size = 16384;
    short               parallel = 8;              /* Max
simultaneous conns. */
    short               queue    = 16;             /* Max queued
connections. */

    GLOBAL_DATA         global_data;
    MT_SERVICE          *services, *s;
    pthread_attr_t      detach_attr;
```

```
    pthread_t           forget_thread;
    sigset_t            blockSet;                       /* To block
broken-pipe.   */
    struct sockaddr_in  addr, remote_addr;
    int                 sockfd, fd_size, one = 1 /*, zero = 0*/;
    time_t              t;
    char                *tt, *msg = "?";
    short               i, activeCount;
    unsigned char       failed;
    /*---------------------------------------------------------- Time
and date: */
    time(&t);
    tt = ctime(&t);
    fprintf(diag, "%s started on %s", software, tt);
    /*---------------------------------------------------------- Init:
*/
    global_data.software = software;
    global_data.in_size  = in_size;
    global_data.out_size = out_size;
    global_data.up = 1;
    /*----------------------------------------------------------
Memory alloc: */
    services = (MT_SERVICE*)malloc(parallel * sizeof(MT_SERVICE));
    if (!services)
        { msg = "Not enough memory!"; goto finish; }
    s = services;                                       /* Initialise
array. */
    failed = 0;
    for (i = 0; i < parallel; i++)
        {                                   /* s->socket needs no
initialisation. */
        s->global_data = &global_data;
        s->in_buff     = (unsigned char*)malloc(in_size  *
sizeof(unsigned char));
        s->out_buff    = (unsigned char*)malloc(out_size *
sizeof(unsigned char));
        s->active      = (unsigned char)0;
        if ((!s->in_buff) || (!s->out_buff))
            failed = 1;                     /* But go on initialising all
members. */
        s++;
        }
    if (failed)
        { msg = "Not enough memory for buffers!"; goto finish; }
    /*---------------------------------------------------------- Create
socket: */
    sockfd = socket(PF_INET, SOCK_STREAM, 0);          /* Or is 'AF_INET'
better? */
    if (sockfd == -1)
        {
        msg = "Socket error!"; goto finish;
        }                                   /* Alternative for
'perror("socket()");'. */
    addr.sin_family = PF_INET;
    addr.sin_port   = htons(port);
    memset(&addr.sin_zero, 0, 8);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);
    if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (void*)&one,
```

```
sizeof(int)) < 0)
        {
        msg = "Setsockopt error!"; goto finish;
        }
/*
    if (setsockopt(sockfd, SOL_SOCKET, SO_KEEPALIVE, (void*)&zero,
sizeof(int)) < 0)
        {
        msg = "Setsockopt error2!"; goto finish;
        }
*/
    if (bind(sockfd, &addr, sizeof(struct sockaddr)) < 0)
        {
        msg = "Bind error!"; goto finish;
        }
    if (listen(sockfd, queue) < 0)
        {
        msg = "Listen error!"; goto finish;
        }
    /*------------------------------- Prevent broken pipe signals from
exitting program: */
    if (sigemptyset(&blockSet))     /* Clear all signals in specified
set. */
        { msg = "Sigemptyset error!"; goto finish; }
                                    /* Adds sig to the specified set. */
    if (sigaddset(&blockSet, SIGPIPE))
        { msg = "Sigaddset error!";   goto finish; }
                                    /* Manipulate set of signals which
are blocked. */
    if (pthread_sigmask(SIG_BLOCK, &blockSet, NULL)) /* All children
inherit this!  */
        { msg = "Sigprocmask error!"; goto finish; }
    /*------------------------------------ Set thread options: */
    if (pthread_attr_init(&detach_attr))  /* Prepare thread attribute for
automatic de- */
        {                                 /* tachment, so we don't need
pthread_join(). */
        msg = "Pthread_attr_init error!"; goto finish;
        }
    if (pthread_attr_setdetachstate(&detach_attr,
PTHREAD_CREATE_DETACHED))
        {
        msg = "Pthread_attr_setdetachstate error!";
        global_data.up = 0; /* Don't jump over pthread_attr_destroy(). */
        }
    /*---------------------------------------------- Service incoming
requests: */
    fprintf(diag, "Listening on port %d\n\n", port);
    activeCount = 0;
    while (global_data.up)
        {
        s = services;
        i = 0;
        while (s->active && (i < parallel))
            {
            s++;
            i++;
            if (activeCount++ == 1023)
```

```
                {
                fprintf(diag, "Server busy!\n");
                sleep(2); /* Give the machine some rest. */
                }
            }
        if (i < parallel) /* Thus (!s->active). */
            {
            activeCount = 0;
acceptAgain:
            fd_size = sizeof(struct sockaddr_in);
            s->socket = accept(sockfd, &remote_addr, &fd_size); /* Wait
for a request. */
            if (s->socket < 0)
                {
                fprintf(diag, "Accept error!\n");
                goto acceptAgain;
                }                               /* Spawned child closes own
socket. */
            if (pthread_create(&forget_thread, &detach_attr, childServer,
(void*)s))
                {
                close(s->socket);
                fprintf(diag, "Pthread_create error!\n");
                goto acceptAgain;
                }
            else
                s->active = 1; /* Only when we really succeeded, reset by
child. */
            }
        }
/*  I don't see no point in 'destroying'... might it even
    be dangerous? I did not alloc it, it is static object. */
    if (pthread_attr_destroy(&detach_attr))
        msg = "pthread_attr_destroy() failed!";
    else
        msg = "Graceful shutdown.";
finish:
    fprintf(diag, "%s\r\n", msg);
    if (services)                       /* Cleanup array. */
        {
        s = services;
        for (i = 0; i < parallel; i++)
            {
            if (s->active)
                {
                fprintf(diag, "Waiting for thread %d...\n", i);
                activeCount = 0;
                while (s->active)
                    {
                    sleep(1); /* Give children i.e. clients 1 minute to
finish. */
                    if (activeCount++ > 60)
                        {
                        fprintf(diag, "KILLING!\n");
                        exit(1);      /* PANIC. */
                        }
                    }
                }
```

```
            if (s->in_buff)  free(s->in_buff);
            if (s->out_buff) free(s->out_buff);
            s++;
            }
        free(services);
        }
    return 0;
}
```

Задания

1. Запустите приведенную в лабораторной работе программу сервера и несколько клиентских процессов. Что произойдет после того, как все клиентские процессы завершат работу?
2. Измените код программ из лабораторной работы так, чтобы два взаимодействующих процесса выполнялись на одном и том же компьютере. В этом случае сокет должен иметь коммуникационный домен `AF_UNIX`.
3. Запустите сервер и несколько клиентов из лабораторной работы, работающих по протоколу UDP. Как сервер определяет, от какого клиента он принимает сообщение?
4. Создайте программу эхо-сервера и тестовый клиент для нее.
5. Портируйте приведенную в приложении программу под платформу Win32.
6. Создайте программу файлового сервера с функциями просмотра списка файлов, загрузки файла на сервер/с сервера и клиент для нее.

Лабораторная работа 6. Интерфейс CGI

6.1. Введение

Разнообразные справочники по языку HTML детально объясняют, как разместить в документе различную статическую информацию – статическую в том смысле, что она не изменяется во времени без вмешательства администратора сервера WWW. Если сервер WWW содержит только статические документы HTML, то такой сервер называется статическим, или пассивным.

Однако есть приложения, в которых нужны серверы, не просто отображающие информацию, но и способные вести диалог с пользователем в интерактивном режиме. Серверы, которые ведут диалог с удаленным пользователем или выполняют обработку данных пользователя, называют активными.

Наиболее известный способ создания активных серверов WWW заключается в использовании так называемых приложений CGI. В отечественной литературе, посвященной серверам WWW, часто встречается транслитерация “CGI-скрипты”, которая произошла от оригинального термина CGI Scripts.

Что кроется за аббревиатурой CGI?

CGI – это стандартный шлюзовой интерфейс (Common Gateway Interface) для запуска внешних программ под управлением сервера WWW. Соответственно, приложениями CGI называются программы, которые, пользуясь этим интерфейсом, получают через протокол HTTP информацию от удаленного пользователя, обрабатывают ее, и возвращают результат обработки обратно в виде ссылки на уже существующий документ HTML или другой объект (например, графическое изображение) или в виде документа HTML, созданного динамически.

Передача информации от удаленного пользователя приложению CGI обычно выполняется следующим образом.

В документе HTML, который создается для ввода информации, предназначенной для обработки, размещается форма ввода. Эта форма состоит из необходимых органов управления: полей редактирования текстовой информации, переключателей, списков и так далее. Каждому органу управления присваивается произвольное имя. Кроме того, в этой форме должна быть кнопка, которую следует нажать после заполнения формы.

Когда пользователь заполняет форму и нажимает указанную кнопку, данные передаются приложению CGI, путь к которому задается в заголовке формы. Это приложение получает через протокол HTTP данные из полей формы в виде пар значений “имя поля=значение”.

После обработки полученных данных приложение CGI создает документ HTML, и записывает его в стандартное устройство вывода stdout. Этот документ автоматически передается удаленному пользователю.

Так как приложение CGI является ни чем иным, как программой, вы должны оттранслировать ее для той операционной системы, под управлением которой работает ваш сервер WWW. В некоторых случаях вы можете найти более удобным создавать программы CGI с использованием специально предназначенных для этого интерпретаторов, таких как Perl. В этой работе мы сконцентрируемся на использовании для создания программ CGI языка программирования C, транслятор с которого можно найти в любой операционной системе.

6.2. Создание форм

Чаще всего программы CGI применяются для обработки данных, введенных удаленными пользователями при помощи форм. Поэтому изучение программ CGI мы начнем с создания форм.

Каждая форма содержит органы управления, с помощью которых пользователь может вводить текстовые или цифровые значения, выбирать строки из списков. В форме могут располагаться переключатели, обычные или графические кнопки.

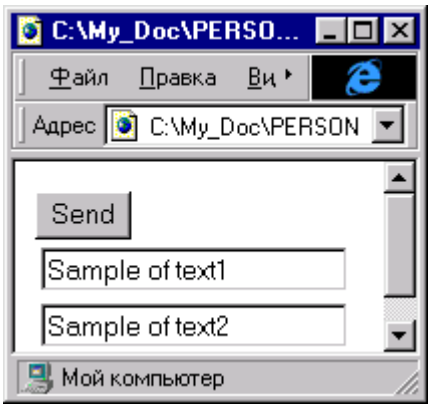
6.2.1. Описание формы

Для того чтобы сделать форму в документе HTML, вы должны воспользоваться оператором <FORM>. Этот оператор используется в паре с оператором </FORM>, завершающим описание формы. Между операторами <FORM> и </FORM> находятся описания органов управления в виде операторов <INPUT>, <TEXTAREA> и <SELECT> с соответствующими параметрами.

Вот пример определения простейшей формы:

```
<FORM METHOD=GET ACTION="http://www.myserver.ru/cgi-bin/form.exe">
  <TABLE>
  <TR>
    <TD><INPUT TYPE=text NAME="text1" VALUE="Sample of text1"></TD>
  </TR>
  <TR>
    <TD><INPUT TYPE=text NAME="text2" VALUE="Sample of text2"></TD>
  </TR>
  <TR>
    <INPUT TYPE=submit VALUE="Send">
  </TR>
  </TABLE>
</FORM>
```

Здесь органы управления размещаются в таблице, состоящей из одного столбца и трех строк. В верхних двух строках мы разместили поля для ввода и редактирования текста, в последней строке – кнопку с названием Send.



Перечислим допустимые параметры оператора <FORM>:

Параметр	Описание
ACTION	Адрес URL для выполнения действий над формой
METHOD	Способ передачи данных из формы в сервер WWW
ENCTYPE	Тип MIME передаваемых данных

Параметр ACTION определяет, какое действие будет выполнено над формой, после того как пользователь ее заполнит и передаст серверу WWW. В примере, приведенном выше, в качестве значения для параметра ACTION мы указали путь к программе CGI, которая будет выполнять обработку данных.

С помощью параметра METHOD вы можете выбрать один из двух методов передачи данных из формы серверу WWW. Если значение этого параметра равно GET (как в нашем примере), программа CGI, указанная в параметре ACTION, получит данные из формы через переменную среды с именем QUERY_STRING. В том случае, когда значение параметра METHOD равно POST, программа CGI получит данные из формы через стандартный поток ввода. Позже мы рассмотрим различия между этими методами более подробно.

И, наконец, третий параметр ENCTYPE, используется очень редко и только для метода POST. Он позволяет указать тип передаваемых данных и по умолчанию имеет значение application/x-www-form-urlencoded.

6.2.2. Создание органов управления для формы

Для создания в форме различных органов управления (полей ввода, переключателей, кнопок и так далее) используются операторы <INPUT>, <TEXTAREA> и <SELECT>.

Оператор <INPUT>

Оператор <INPUT> предназначен для вставки в форму таких органов управления, как поля ввода текстовой информации, переключатели, кнопки (обычные и в виде графических изображений), а также органы управления для передачи локального файла через навигатор в удаленный сервер WWW.

Перечислим параметры оператора <INPUT>:

Параметр	Описание
TYPE	Тип органа управления. В зависимости от значения этого параметра будут создаваться различные органы управления (кнопки, переключатели и так далее)
NAME	Имя органа управления. Это имя посылается программе обработки формы и используется для определения состояния органа управления (для переключателей) или получения других данных (например, для получения строки, введенной в текстовом поле)
VALUE	Начальное состояние или начальное значение для органа управления. Используется для инициализации органа управления при начальном отображении формы
CHECKED	Этот параметр используется для установки начального значения переключателей
SIZE	Ширина поля для ввода текстовой информации в символах. По умолчанию поле имеет ширину 20 символов
MAXLENGTH	Максимальное количество символов, которое можно ввести в поле редактирования текстовой информации. По умолчанию такое ограничение отсутствует
ALIGN	Выравнивание текста, расположенного около формы
SRC	Адрес URL графического изображения, если оно используется в органе управления

Параметр TYPE определяет тип создаваемого органа управления и может иметь следующие значения:

Значение параметра TYPE	Тип органа управления
TEXT	Однострочное поле для ввода текстовой информации. Размер этого поля определяется параметрами SIZE и MAXLENGTH
TEXTAREA	Многострочное поле для ввода текстовой информации. Размер поля также определяется параметрами SIZE и MAXLENGTH
PASSWORD	Этот орган управления предназначен для ввода такой информации, как пароли. Он аналогичен органу управления типа TEXT, но отличается тем, что текст, введенный пользователем, не отображается на экране
CHECKBOX	Переключатель типа Check Box. Предназначен для использования в наборе независимых друг от друга переключателей или отдельно
RADIO	Переключатель для группы зависимых переключателей. Используется для выбора одного значения из нескольких
FILE	Орган управления для выбора и передачи файла. Это значение используется по-разному навигаторами Microsoft Internet Explorer и Netscape Navigator
BUTTON	Кнопка с заданной надписью

SUBMIT	Кнопка, которая предназначена для посылки данных из заполненной формы серверу WWW. Надпись на этой кнопке также можно задавать
RESET	С помощью этой кнопки пользователь может сбросить содержимое полей ввода и состояние переключателей в их начальные значения, заданные операторами VALUE
IMAGE	Для посылки данных из формы в сервер вы можете использовать не только кнопку типа SUBMIT, но и произвольное графическое изображение, заданное параметром SRC. Соответствующий графический орган управления имеет тип IMAGE
HIDDEN	Скрытое поле, которое не отображается. Содержимое этого поля посылается серверу и может быть проанализировано

Оператор <TEXTAREA>

Оператор <INPUT> с параметром TYPE, имеющим значение TEXT, позволяет вставить в форму поле редактирования текстовой строки. Если же вам нужно ввести многострочный текст, лучше воспользоваться оператором <TEXTAREA>, который применяется совместно с оператором </TEXTAREA>.

Заметим, что хотя параметр TYPE оператора <INPUT> позволяет задать многострочное поле редактирования с типом TEXTAREA, созданное таким образом поле работает как однострочное.

Для оператора <TEXTAREA> вы можете задать три параметра:

Параметр	Описание
NAME	Имя многострочного поля, которое посылается программе обработки формы и используется для получения введенных строк текста
ROWS	Размер поля по вертикали (в строках)
COLS	Размер поля по горизонтали (в символах)

Вот пример описания многострочного текстового поля:

```
<TEXTAREA NAME="multi" ROWS=54 COLS=60>
Это
  образец
     многострочного
          текста
</TEXTAREA>
```

Оператор <SELECT>

С помощью оператора <SELECT> вы можете вставить в форму заранее проинициализированный список произвольных текстовых строк. Выбранная строка пересылается серверу WWW наряду с содержимым других полей формы.

Для оператора <SELECT> определены два параметра – NAME и SIZE. Параметр NAME задает имя списка, которое передается серверу WWW в паре с выбранной строкой. С помощью параметра SIZE можно задать высоту списка в строках.

Ниже мы привели пример использования оператора <SELECT>:

```
<SELECT NAME="number">
  <OPTION>Первый
  <OPTION SELECTED>Второй
  <OPTION>Третий
  <OPTION>Последний
</SELECT>
```

Для записи строк в список здесь используется оператор <OPTION>. Строка, отмеченная параметром SELECTED, будет выбрана в списке по умолчанию.

6.2.3. Пример документа HTML с формой

Формы лучше изучать на конкретном примере. В этом разделе мы создадим форму, содержащую почти все органы управления, и рассмотрим исходный текст соответствующего

документа HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Органы управления в формах</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<FORM METHOD=POST ACTION="http://www.someserver.ru/cgi-bin/controls.exe">
  <TABLE>
    <TR>
      <TD VALIGN=TOP>Текстовое поле TEXT</TD>
      <TD><INPUT TYPE=text NAME="text1" VALUE="Sample of text1"
SIZE=30></TD>
    </TR>
    <TR>
      <TD VALIGN=TOP>Текстовое поле PASSWORD</TD>
      <TD><INPUT TYPE=password NAME="pwd" VALUE="Sample of
password"></TD>
    </TR>
    <TR>
      <TD VALIGN=TOP>Текстовое поле TEXTAREA</TD>
      <TD><TEXTAREA NAME="text2" ROWS=4 COLS=30>Sample of
text</TEXTAREA></TD>
    </TR>
    <TR>
      <TD VALIGN=TOP>Переключатели CHECKBOX</TD>
      <TD>
        <INPUT TYPE=CHECKBOX NAME="chk1" VALUE="on" CHECKED>Первый<BR>
        <INPUT TYPE=CHECKBOX NAME="chk2" VALUE="on">Второй<BR>
        <INPUT TYPE=CHECKBOX NAME="chk3" VALUE="on" CHECKED>Третий<BR>
      </TD>
    </TR>
    <TR>
      <TD VALIGN=TOP>Переключатели RADIO</TD>
      <TD>
        <INPUT TYPE=RADIO NAME="rad" VALUE="on1" CHECKED>Первый<BR>
        <INPUT TYPE=RADIO NAME="rad" VALUE="on2">Второй<BR>
        <INPUT TYPE=RADIO NAME="rad" VALUE="on3">Третий<BR>
      </TD>
    </TR>
    <TR>
      <TD VALIGN=TOP>Список</TD>
      <TD>
        <SELECT NAME="sel" SIZE="1">
          <OPTION Value="First Option">First Option</OPTION>
          <OPTION Value="Second Option">Second Option</OPTION>
          <OPTION Value="None">None Selected</OPTION>
        </SELECT>
      </TD>
    </TR>
    <TR>
      <TD VALIGN=TOP>Скрытый орган управления</TD>
      <TD><INPUT TYPE=HIDDEN NAME="hid" VALUE="Hidden"></TD>
    </TR>
  </TABLE>
<BR><INPUT TYPE=submit VALUE="Send"> 
<INPUT TYPE=reset VALUE="Reset">
<P><INPUT TYPE=IMAGE SRC="send.gif" BORDER=0>
```

```
</FORM>
</BODY>
</HTML>
```

Оператор <FORM> здесь имеет два параметра – METHOD и ACTION:

```
<FORM METHOD=POST ACTION="http://www.someserver.ru/cgi-bin/controls.exe">
```

Параметр METHOD имеет значение POST и задает способ передачи данных программе CGI через стандартный поток ввода.

В параметре ACTION указан путь к загрузочному файлу программы CGI, которая находится в каталоге cgi-bin сервера WWW с адресом http://www.someserver.ru.

Заметим, что программы CGI могут находиться не в любом каталоге сервера WWW, а только в таком, для которого разрешено выполнение программ. Если вы создаете виртуальный сервер WWW, который физически располагается у поставщика услуг Internet, возможно, вам придется получить разрешение на создание или использование такого каталога.

К примеру, сервер TinyWeb предполагает, что такой каталог называется cgi-bin и находится в каталоге, используемом в качестве начального при запуске TinyWeb. Команда

```
start tiny inet
```

запускает Web-сервер, указывая в качестве начального каталога inet. Структура каталогов для данного Web-сервера должна включать:

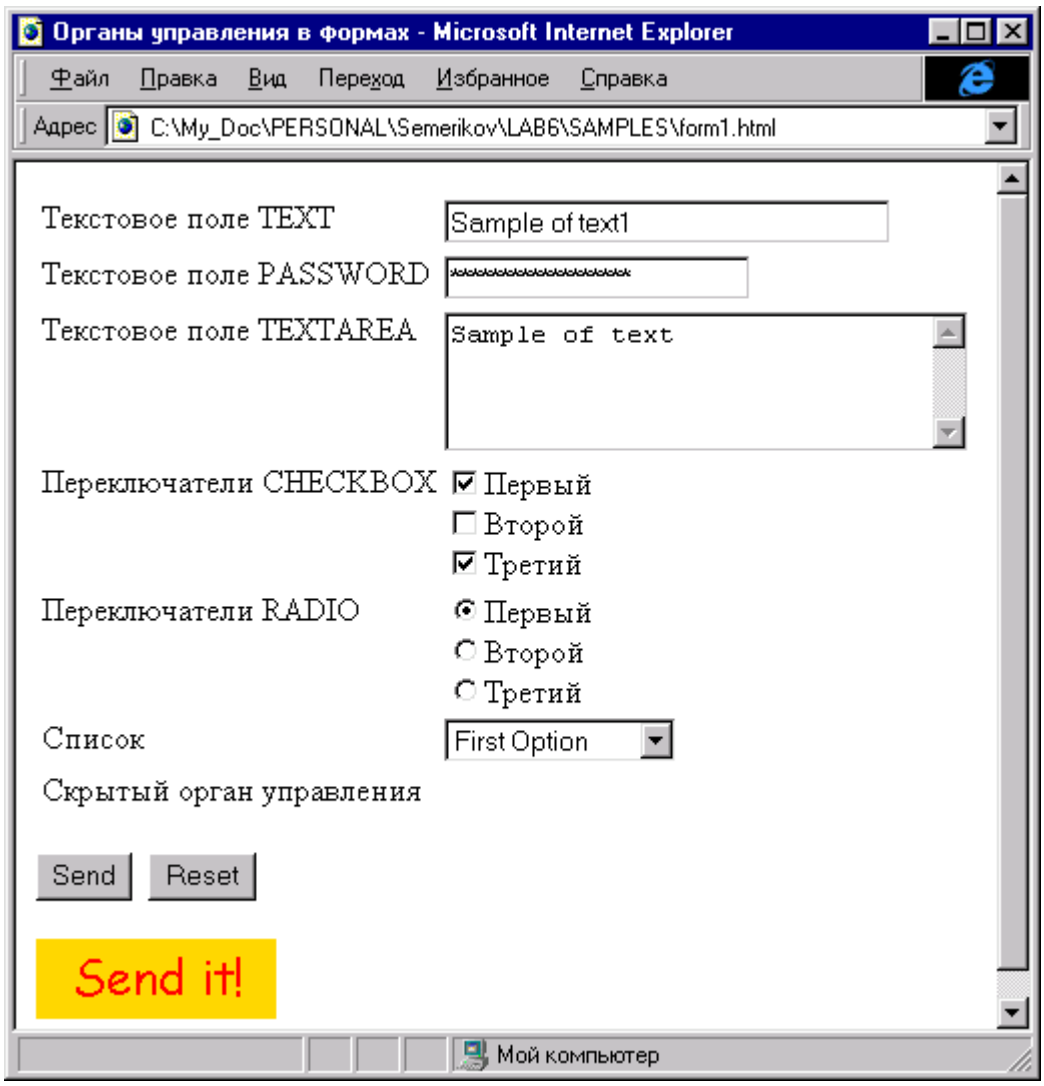
```
tiny.exe
inet
|
|___ index.html
|___ cgi-bin
     |___ controls.exe
```

Можно не указывать полный сетевой путь, ограничившись

```
ACTION=/cgi-bin/controls.exe
```

В этом случае можно тестировать данную форму и на локальном компьютере с произвольным сетевым именем, обратившись к нему по адресу 127.0.0.0 или имени localhost:

```
start http://localhost
```



Теперь займемся органами управления.

Мы разместили все органы управления в таблице. В первой строке этой таблицы находится однострочное поле для ввода текста, которое вставлено в форму оператором <INPUT> и имеет тип TEXT:

```
<TD><INPUT TYPE=text NAME="text1" VALUE="Sample of text1" SIZE=30></TD>
```

Имя поля указано в параметре NAME как “text1”. В качестве начального значения для поля параметром VALUE задана строка "Sample of text1". Эту строку можно будет редактировать после отображения формы. Мы также указали размер поля, равный 30 символам, для чего воспользовались параметром SIZE.

Во второй строке таблицы также при помощи оператора <INPUT> определено поле для ввода пароля:

```
<TD><INPUT TYPE=password NAME="pwd" VALUE="Sample of password"></TD>
```

При вводе символов в этом поле они не отображаются. Аналогично, не отображается и начальная строка, использованная для инициализации поля.

Для ввода многострочного текста в третьей строке таблицы при помощи оператора <TEXTAREA> мы разместили область ввода текста:

```
<TD><TEXTAREA NAME="text2" ROWS=4 COLS=30>Sample of text</TEXTAREA></TD>
```

Имя этого поля задано как “text2”. Поле имеет высоту, равную четырем строкам (параметр ROWS равен 4), и ширину, равную 30 символам (параметр COLS равен 30).

В четвертой строке таблицы мы расположили группу из трех независимых переключателей типа CHECKBOX:

```
<TD>
  <INPUT TYPE=CHECKBOX NAME="chk1" VALUE="on" CHECKED>Первый<BR>
```

```
  <INPUT TYPE=CHECKBOX NAME="chk2" VALUE="on">Второй<BR>
  <INPUT TYPE=CHECKBOX NAME="chk3" VALUE="on" CHECKED>Третий<BR>
</TD>
```

Каждый из этих переключателей имеет собственное имя, заданное параметром NAME. Оператор VALUE задает значение, которое будет послано в сервер WWW при включении переключателя. Кстати, если переключатель выключен, он не посылает в сервер WWW никаких данных.

Ниже, в пятой строке, находится группа из трех переключателей типа RADIO с зависимой фиксацией:

```
<TD>
  <INPUT TYPE=RADIO NAME="rad" VALUE="on1" CHECKED>Первый<BR>
  <INPUT TYPE=RADIO NAME="rad" VALUE="on2">Второй<BR>
  <INPUT TYPE=RADIO NAME="rad" VALUE="on3">Третий<BR>
</TD>
```

Все переключатели, относящиеся к одной группе, должны называться одинаково. В нашем случае при помощи параметра NAME мы задали для всех трех переключателей имя “rad”.

Первый переключатель включен по умолчанию, так как для него задан параметр CHECKED.

Для того чтобы при анализе данных, полученных от формы, программа CGI могла определить, какой из переключателей, входящих в группу, был включен, мы задали для каждого переключателя свое значение параметра VALUE.

Шестая строка таблицы содержит список, состоящий из трех строк. Этот список определен при помощи операторов <SELECT> и <OPTION>, как это показано ниже:

```
<TD>
  <SELECT NAME="sel" SIZE="1">
    <OPTION Value="First Option">First Option</OPTION>
    <OPTION Value="Second Option">Second Option</OPTION>
    <OPTION Value="None">None Selected</OPTION>
  </SELECT>
</TD>
```

Имя списка задано как “sel”, а высота его равна одной строке. Содержимое строк списка задается при помощи параметра VALUE соответствующих операторов <OPTION>.

Последняя строка таблицы содержит скрытый орган управления, который не отображается в окне навигатора:

```
<TD><INPUT TYPE=HIDDEN NAME="hid" VALUE="Hidden"></TD>
```

Он посылает серверу WWW строку “Hidden”, заданную в параметре VALUE.

Под таблицей в форме расположены три кнопки, первые две из которых стандартные, а третья сделана при помощи графического изображения. Эти кнопки вставлены при помощи оператора <INPUT> следующим образом:

```
<BR><INPUT TYPE=submit VALUE="Send"> 
<INPUT TYPE=reset VALUE="Reset">
<P><INPUT TYPE=IMAGE SRC="send.gif" BORDER=0>
```

Кнопка типа SUBMIT имеет надпись “Send” и предназначена для посылки данных из формы в сервер WWW для обработки программой CGI.

Кнопка типа RESET предназначена для того, чтобы пользователь, изменив данные в форме, мог снова вернуться к значениям, заданным по умолчанию при помощи параметра VALUE в операторах определения органов управления. Эта кнопка имеет надпись “Reset”.

Последняя кнопка имеет тип IMAGE. Ее изображение находится в файле send.gif, адрес URL которого (в нашем случае это просто имя файла) указан в параметре SRC. Для того чтобы вокруг изображения кнопки не было рамки, мы указали нулевое значение параметра BORDER.

Заметим, что программа CGI получит от графической кнопки координаты точки, в ко-

торой находился курсор мыши в момент нажатия на эту кнопку. Таким образом возможно создание кнопки в виде сегментированного графического изображения. Программа CGI сможет определить, в какой области изображения был сделан щелчок мышью при отправке заполненной формы на обработку.

6.3. Передача данных программе CGI

Когда пользователь заполняет форму и нажимает на кнопку типа SUBMIT либо на графическую кнопку (которая выполняет аналогичную функцию), данные из полей формы вместе с именами этих полей передаются навигатором серверу WWW. Сервер, в свою очередь, анализирует эти данные и запускает соответствующую программу CGI, путь к файлу которой указан в операторе <FORM>.

Перед запуском программы CGI сервер WWW выбирает в зависимости от значения параметра METHOD оператора <FORM> один из двух способов передачи полученных данных для обработки. Это методы GET и POST.

6.3.1. Метод GET

Метод GET предполагает передачу данных программе CGI через переменные среды (environment variables). Это те самые переменные среды, которые устанавливаются в операционной системе MS-DOS командой SET.

Сервер WWW создает для программы CGI довольно много переменных среды. Имена и назначение всех этих переменных вы узнаете позже, а пока мы расскажем только о самых необходимых.

Прежде всего, метод GET предполагает использование переменной среды с именем QUERY_STRING. Именно сюда попадают данные из полей формы. Эти данные находятся в следующем формате:

```
"Имя1=Значение1&Имя2=Значение2&Имя3=Значение3"
```

Здесь в качестве имен используются значения параметров NAME, задающих имена полей формы. Вместо значений подставляются данные из соответствующих полей. Сканируя содержимое текстовой строки переменной среды QUERY_STRING, программа CGI может найти в ней имя любого нужного поля и соответствующее этому имени значение. Заметим, что никакие данные от выключенных переключателей не передаются, поэтому не следует думать, что в полученной строке вы обязательно встретите имена всех полей, расположенных в форме.

Адрес заданной строки переменной среды в программе, составленной на C, легко получить с помощью функции getenv:

```
char * szQueryString;
szQueryString = getenv("QUERY_STRING");
```

Заметим, что если вы собираетесь модифицировать строку переменной среды, то ее следует скопировать во внутренний буфер. Операционная система сервера WWW может не допустить прямого редактирования блока памяти, содержащего переменные среды.

Строка, передаваемая в переменной среды QUERY_STRING, закодирована с использованием так называемой кодировки URL. В этой кодировке все символы пробелов заменяются на символы "+". Кроме того, для представления кодов управляющих и некоторых других символов используется последовательность символов вида "%xx", где символы "xx" представляют собой шестнадцатеричный код символа в виде двух символов ASCII. В нашей книге вы найдете исходные тексты функций, предназначенные для перекодирования информации, полученной из формы.

6.3.2. Метод POST

При использовании метода POST программа CGI получает данные из формы через стандартный поток ввода STDIN. Если программа CGI составлена на языке программирования C, то для получения данных она может воспользоваться такими функциями, как fread

или scanf.

Что же касается количества байт данных, которые нужно считать из стандартного потока ввода, то эта информация передается программе CGI через переменную среды с именем CONTENT_LENGTH.

Ниже мы привели фрагмент кода для определения размера информации для ввода через стандартный поток STDIN:

```
int Size;
Size = atoi(getenv("CONTENT_LENGTH"));
```

Входные данные могут быть затем получены, например, следующим образом:

```
char szBuf[8196];
fread(szBuf, Size, 1, stdin);
```

Разумеется, буфер для чтения данных можно заказывать и динамически, для чего следует воспользоваться такой функцией, как malloc.

Если в операторе <FORM> не указан параметр ENCTYPE (тип MIME передаваемых данных) или этот параметр имеет значение application/x-www-form-urlencoded, данные, полученные через стандартный поток ввода, закодированы в кодировке URL. Перед использованием вы должны их раскодировать соответствующим образом.

6.3.3. Что лучше – GET или POST

Метод GET обычно используется для обработки небольших форм, так как навигаторы накладывают ограничения для размера данных, передаваемых через переменную среды QUERY_STRING.

В этом отношении метод POST является более предпочтительным, так как не накладывает на размер передаваемых данных никаких ограничений. Только метод POST пригоден для передачи файлов из локального компьютера через навигатор в сервер WWW.

6.4. Передача ответа из программы CGI

Вне зависимости от использованного метода передачи данных (GET или POST) результат своей работы программа CGI должна направить в стандартный поток вывода STDOUT. Если программа составлена на языке программирования C, для записи результат работы она может воспользоваться, например, функцией printf или fwrite.

Чаще всего программы CGI используются для создания динамических документов HTML на основе данных, полученных из формы. В этом случае первой строкой, которую необходимо вывести в стандартный поток вывода STDOUT, должна быть следующая строка заголовка HTTP:

```
Content-type: text/html
```

Сразу за этой строкой необходимо вывести еще одну пустую строку, которая послужит разделителем между заголовком HTTP и данными документа HTML.

Ниже мы привели фрагмент кода, в котором программа CGI динамически формирует документ HTML и выводит его в стандартный поток вывода STDOUT:

```
printf("Content-type: text/html\n\n");
printf("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 3.2//EN\">");
printf("<HTML><HEAD><TITLE>XYZ Incorporation</TITLE></HEAD>"
  "<BODY BGCOLOR=#FFFFFF>");
printf("<H1>Результаты обработки формы</H1>");
. . .
printf("</BODY></HTML>");
```

Обратите внимание на символы перевода строки “\n\n”. Первый из них закрывает строку заголовка HTTP, а второй нужен для создания пустой разделительной строки.

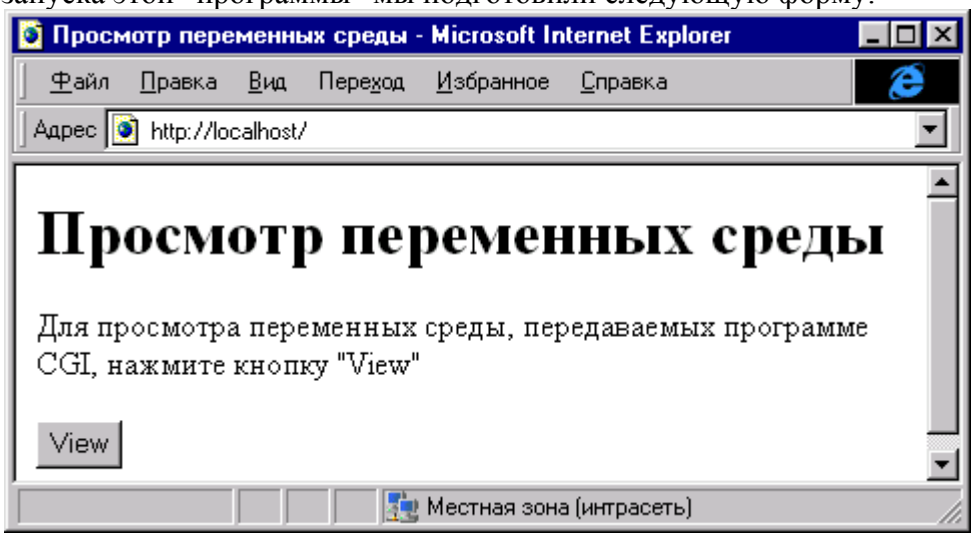
6.5. Переменные среды для программы CGI

Прежде чем перейти к примерам программ CGI, мы изучим переменные среды, кото-

рые формируются для этих программ перед запуском. Через эти переменные помимо данных из полей форм передается и другая очень важная информация, которую не всегда следует игнорировать.

Изучение переменных среды мы начнем с того, что просмотрим их значение с помощью простейшей программы CGI, созданной с использованием языка пакетных заданий операционной системы Microsoft Windows NT, а проще говоря, с помощью обыкновенного файла с расширением имени bat.

Для запуска этой “программы” мы подготовили следующую форму:




```
<HTML>
<HEAD>
 <TITLE>Просмотр переменных среды</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
  <H1>Просмотр переменных среды</H1>
  <P>Для просмотра переменных среды, передаваемых программе
     CGI, нажмите кнопку &quot;View&quot;
  <FORM METHOD=GET ACTION="/cgi-bin/test.bat?param1">
  <INPUT TYPE=SUBMIT VALUE="View">
  </FORM>
</BODY>
</HTML>
```

Обратите внимание, что после имени программы CGI через разделительный символ “?” указана строка параметров param1, которая может быть получена и проанализирована программой CGI.

Исходный текст программы CGI:

```
echo Content-type: text/plain
echo
echo %0 %1 %2 %3
set
```

В первой строке программа выводит в стандартный поток вывода STDOUT строку заголовка HTTP. Эта строка описывает тип передаваемых данных как `text/plain`, то есть обычный текст без оформления с использованием операторов HTML.

Вторая строка выводит в стандартный поток вывода пустую строку, которая отделяет заголовок `HTTP` от передаваемых данных.

Третья строка выводит путь к загрузочному файлу программы CGI и параметры, полученные ей при запуске.

И, наконец, в последней строке вызывается команда `SET`, которая выводит в поток `STDOUT` значения всех переменных среды, определенных в системе перед запуском программы.

Рассмотрим по отдельности назначение переменных среды. Заметим, что набор пере-

менных, создаваемых при запуске программы CGI, зависит от конкретной реализации сервера WWW.

AUTH_TYPE

Технология WWW допускает защиту страниц HTML, когда доступ к отдельным страницам предоставляется только для отдельных пользователей при предъявлении пароля. При этом используется так называемая система аутентификации, или проверки подлинности идентификатора пользователя. Переменная среды AUTH_TYPE содержит тип идентификации, который применяется сервером (например, “NTLM”).

GATEWAY_INTERFACE

В этой переменной находится версия интерфейса CGI, с которой работает данный сервер. В нашем случае интерфейс имеет версию 1.1.

HTTP_ACCEPT

В этой переменной перечислены типы данных MIME, которые могут быть приняты навигатором от сервера WWW. Сервер может передать навигатору, который был использован для работы с программой CGI, графические изображения (image) в формате gif, jpeg, pjpeg, x-xbitmap.

HTTP_REFER

В переменную HTTP_REFER записывается адрес URL документа HTML, который инициировал работу программы CGI.

HTTP_ACCEPT_LANGUAGE

Переменная HTTP_ACCEPT_LANGUAGE содержит идентификатор предпочтительного национального языка для получения ответа от сервера WWW. Обычно это английский язык, хотя сервер может прислать ответ на любом национальном языке.

HTTP_UA_PIXELS

Разрешение видеоадаптера, установленное в компьютере пользователя.

HTTP_UA_COLOR

Допустимое количество цветов в системе пользователя.

HTTP_UA_OS

Операционная система, под управлением которой работает навигатор.

HTTP_UA_CPU

Тип центрального процессора в компьютере удаленного пользователя.

HTTP_USER_AGENT

В эту переменную записывается имя навигатора, с помощью которого запрашивается документ HTML. Анализируя это имя, программа CGI может принимать решение об использовании тех или иных расширений стандарта языка HTML, допустимого для конкретного навигатора.

HTTP_HOST

Имя узла, на котором работает сервер WWW.

HTTP_CONNECTION

Тип соединения.

HTTP_ACCEPT_ENCODING

Метод кодирования, который может быть использован навигатором для формирования ответа серверу WWW.

HTTP_AUTHORIZATION

Информация авторизации от навигатора. Используется навигатором для собственной аутентификации в сервере WWW.

HTTP_FROM

Имя пользователя в виде, как оно было зарегистрировано при настройке навигатора. Используется формат адресов электронной почты.

HTTP_PRAGMA

Специальные команды серверу WWW.

CONTENT_LENGTH

Количество байт данных, которые программа CGI должна получить от навигатора.

CONTENT_TYPE

Тип данных, присланных навигатором.

`PATH_INFO`

Путь к виртуальному каталогу, в котором находится программа CGI.

Как правило, при настройке сервера WWW администратор выделяет один или несколько каталогов для хранения расширений сервера в виде программ CGI. Для файлов, записанных в такие каталоги, устанавливается доступ на запуск.

Администратор создает таблицу соответствия физических каталогов и виртуальных, определяя права доступа к виртуальным каталогам с помощью программы настройки параметров сервера WWW.

`PATH_TRANSLATED`

Физический путь к программе CGI.

`QUERY_STRING`

Строка параметров, указанная в форме после адреса URL программы CGI после разделительного символа “?”.

`REMOTE_ADDR`

Адрес IP узла, на котором работает навигатор удаленного пользователя.

`REMOTE_HOST`

Доменное имя узла, на котором работает навигатор удаленного пользователя. Если эта информация недоступна (например, для узла не определен доменный адрес), вместо доменного имени указывается адрес IP, как в переменной `REMOTE_ADDR`.

`REMOTE_USER`

Имя пользователя, которое используется навигатором для аутентификации. Используется только в том случае, если сервер WWW способен работать с аутентификацией и программа CGI отмечена как защищенная.

`REQUEST_METHOD`

Метод доступа, который используется для передачи данных от навигатора серверу WWW. В своих примерах мы используем методы доступа `GET` и `POST`, хотя протокол `HTTP` допускает применение и других методов доступа, например, `PUT` и `HEAD`.

`SCRIPT_NAME`

В эту переменную записывается путь к виртуальному каталогу и имя программы CGI. Анализируя эту переменную, программа CGI может определить путь к своему загрузочному файлу.

`SERVER_NAME`

Доменное имя сервера WWW или адрес IP сервера WWW, если доменное имя недоступно или не определено.

`SERVER_PROTOCOL`

Имя и версия протокола, который применяется для выполнения запроса к программе CGI.

`SERVER_PORT`

Номер порта, на котором навигатор посылает запросы серверу WWW.

`SERVER_SOFTWARE`

Название и версия программного обеспечения сервера WWW. Версия следует после названия и отделяется от последнего символом “/”.

`REMOTE_IDENT`

Имя, с которым пользователь подключился к серверу WWW. Используется только в том случае, если сервер WWW способен подключать пользователей по именам.

6.6. Примеры программ CGI

В этом разделе мы приведем примеры несложных программ CGI, демонстрирующих динамическое создание документов HTML и обработку данных, введенных при помощи форм.

6.6.1. Программа CGIHELLO

Программа CGIHELLO представляет собой простейшую программу CGI, которая запускается при помощи кнопки в форме, возвращая навигатору документ HTML, созданный динамически.

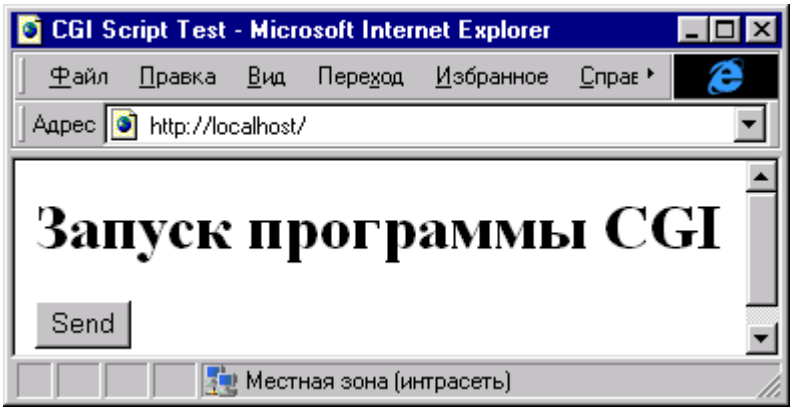
Эта программа хороша для проверки возможности запуска программ CGI на вашем сервере WWW или на сервере вашего поставщика услуг Internet. Так как она очень проста, существует немного причин, по которым она могла бы не работать. Это неправильная настройка прав доступа к виртуальному каталогу, содержащему загрузочный модуль программы CGI, а также неправильная ссылка на этот каталог в параметре ACTION оператора <FORM>.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
  <HEAD>
    <TITLE>CGI Script Test</TITLE>
  </HEAD>
  <BODY BGCOLOR=#FFFFFF>
    <H1>Запуск программы CGI</H1>
    <FORM METHOD=GET ACTION=/cgi-bin/cgihello.exe>
      <INPUT TYPE=submit VALUE="Send">
    </FORM>
  </BODY>
</HTML>
```

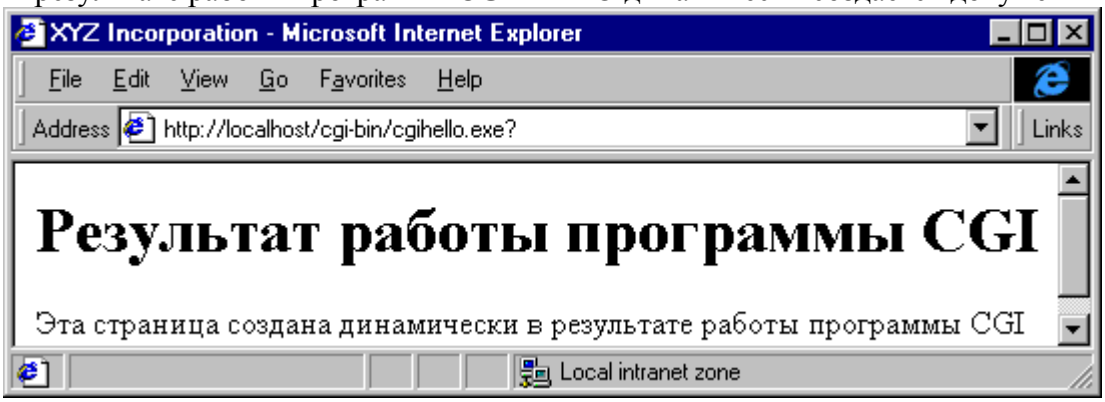
В этом документе определена форма, содержащая единственную кнопку, созданную оператором <INPUT> и имеющую тип SUBMIT.

В параметре ACTION оператора <FORM> мы указали путь к программе CGI, причем этот путь является виртуальным. Для передачи данных используется метод GET.

Внешний вид формы при ее просмотре навигатором Microsoft Internet Explorer представлен на рисунке:



В результате работы программы CGIHELLO динамически создается документ HTML:



Рассмотрим исходный текст программы CGIHELLO:

```
#include <stdio.h>
#include <stdlib.h>
```

```
void main(int argc, char *argv[])
{
  printf("Content-type: text/html\n\n");
  printf("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 3.2//EN\">");
  printf("<HTML><HEAD><TITLE>"
     "XYZ Incorporation</TITLE></HEAD><BODY>");

  printf("<H1>Результат работы программы CGI</H1>");
  printf("<P>Эта страница создана динамически в результате"
    " работы программы CGI");
  printf("</BODY></HTML>");
}
```

Как видите, эта программа не отличается особой сложностью и состоит из ряда вызовов функции `printf`.

В первый раз функция `printf` выводит заголовок `HTTP` и пустую строку-разделитель. Далее программа CGIHELLO записывает построчно в стандартный поток вывода `STDOUT` текст документа HTML.

6.6.2. Программа CONTROLS

Более сложная программа CGI называется CONTROLS и выполняет обработку данных, полученных из формы.

Программа CONTROLS отображает в динамически формируемом документе HTML метод, использованный для передачи (`POST` или `GET`), размер и тип данных, поступающих от формы. Принятые данные показываются как в исходном виде, так и после перекодировки. Кроме того, в документе HTML располагается список значений всех полей, определенных в форме.

Переменные среды

REQUEST_METHOD = POST
CONTENT_LENGTH = 127
CONTENT_TYPE = application/x-www-form-urlencoded

Принятые данные

text1=Sample+of+text1&pwd=Sample+of+password&text2=Sample+of+text&chk1=on&chk3=on&rad=on1&sel=First+Option&hid=Hidden&x=80&y=17

Данные после перекодировки

text1=Sample of text1&pwd=Sample of password&text2=Sample of text&chk1=on&chk3=on&rad=on1&sel=First Option&hid=Hidden&x=80&y=17

Список значений полей

text1=Sample of text1
pwd=Sample of password
text2=Sample of text
chk1=on
chk3=on
rad=on1
sel=First Option
hid=Hidden

```
x=80
y=17
```

Видно, что навигатор прислал серверу WWW 127 байт информации. Так как при этом был использован метод `POST`, данные были направлены в стандартный поток ввода. Данные закодированы в кодировке URL, так как содержимое переменой среды `CONTENT_TYPE` равно `application/x-www-form-urlencoded`.

Обратите внимание на текстовое поле с именем text1. Все пробелы в соответствующей строке в кодировке URL заменены на символ “+”. Что же касается символов “&” и “,”, то они пришли в виде %26 и %2C. Функция перекодирования возвращает строку в исходный вид – “Sample of text1 &,.”.

Форма имеет две кнопки, предназначенные для передачи данных серверу WWW. Это обычная кнопка и кнопка в виде графического изображения. Мы нажали графическую кнопку, поэтому от формы пришла информация о координатах курсора мыши в виде переменных с именами x и y.

Рассмотрим исходный текст программы CONTROLS:.

```
// ================================================
// Программа CGI controls.c
// Демонстрирует методы получения и обработки
// данных от форм, расположенных в документах HTML
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Прототипы функций перекодировки
void DecodeStr(char *szString);
char DecodeHex(char *str);

// -----------------------------------------------
// Функция main
// Точка входа программы CGI
// -----------------------------------------------
void main(int argc, char *argv[])
{
  int lSize;
  FILE * fileReceived;
  char * szMethod;
  char * szQueryString;
  char szBuf[8196];
  char szSrcBuf[8196];
  char * szPtr;
  char * szParam;

  // Вывод заголовка HTTP и разделительной строки
  printf("Content-type: text/html\n\n");

  // Вывод начального форагмента документа HTML,
  // формируемого динамически
  printf("<!DOCTYPE HTML PUBLIC"
    " \"-//W3C//DTD HTML 3.2//EN\">");
  printf("<HTML><HEAD><TITLE>XYZ Incorporation"
    "</TITLE></HEAD><BODY BGCOLOR=#FFFFFF>");

  // Определяем метод передачи данных
```

```
szMethod = getenv("REQUEST_METHOD");

// Обработка метода POST
if(!strcmp(szMethod, "POST"))
{
  // Определяем размер данных, полученных от навигатора
  // при передаче данных из полей формы
  lSize = atoi(getenv("CONTENT_LENGTH"));

  // Читаем эти данные в буфер szBuf из
  // стандартного потока ввода STDIN
  fread(szBuf, lSize, 1, stdin);

  // Создаем файл, в который будут записаны
  // принятые данные
  fileReceived = fopen("received.dat", "w");

  // Выполняем запись принятых данных
  fwrite(szBuf, lSize, 1, fileReceived);

  // Закрываем файл принятых данных
  fclose(fileReceived);

  // Отображаем значения некоторых переменных среды
  printf("<H2>Переменные среды</H2>");

  // Метод доступа
  printf("REQUEST_METHOD = %s", getenv("REQUEST_METHOD"));

  // Размер полученных данных в байтах
  printf("<BR>CONTENT_LENGTH = %ld", lSize);

  // Тип полученных данных
  printf("<BR>CONTENT_TYPE = %s", getenv("CONTENT_TYPE"));

  // Закрываем буфер данных двоичным нулем,
  // превращая его таким образом в строку
  szBuf[lSize] = '\0';

  // Делаем копию принятых данных в буфер szSrcBuf
  strcpy(szSrcBuf, szBuf);

  // Отображаем принятые данные без обработки
  printf("<H2>Принятые данные</H2>");
  printf("<P>%s", szSrcBuf);

  // Выполняем перекодировку принятых данных
  DecodeStr(szSrcBuf);

  // Отображаем результат перекодировки
  printf("<H2>Данные после перекодировки</H2>");
  printf("<P>%s", szSrcBuf);

  // Выводим список значений полей формы
  printf("<H2>Список значений полей</H2>");

  // Дописываем в конец буфера принятых данных
  // символ "&", который используется в качестве
```

```
    // разделителя значений полей
    szBuf[lSize] = '&';
    szBuf[lSize + 1] = '\0';

    // Цикл по полям формы
    for(szParam = szBuf;;)
    {
      // Ищем очередной разделитель
      szPtr = strchr(szParam, '&');

      // Если он найден, раскодируем строку параметров
      if(szPtr != NULL)
      {
        *szPtr = '\0';
        DecodeStr(szParam);

        // Выводим в документ значение параметра
        printf("%s<BR>", szParam);

        // Переходим к следующему параметру
        szParam = szPtr + 1;

        // Если достигнут конец буфера, завершаем цикл
        if(szParam >= (szBuf + lSize))
          break;
      }
      else
        break;
    }

    // Выводим завершающий фрагмент документа HTML
    printf("</BODY></HTML>");
    return;
  }

  // Обработка метода GET
  else if(!strcmp(szMethod, "GET"))
  {
    // Получаем данные, полученные от формы.
    // При использовании метода GET эти данные
    // передаются в переменной среды QUERY_STRING
    szQueryString = getenv("QUERY_STRING");

    // Записываем эти данные в выходной файл
    fileReceived = fopen("received.dat", "w");
    fwrite(szQueryString, strlen(szQueryString) + 1,
       1, fileReceived);
    fclose(fileReceived);

    // Выводим в динамически формируемый документ HTML
    // значения некоторых переменных среды
    printf("<H2>Переменные среды</H2>");

    // Метод передачи данных
    printf("REQUEST_METHOD = %s", getenv("REQUEST_METHOD"));

    // Полученные данные
    printf("<BR>QUERY_STRING = %s", szQueryString);
```

```
    // Копируем принятые данные в буфер szSrcBuf
    strcpy(szSrcBuf, szQueryString);

    // Отображаем принятые данные
    printf("<H2>Принятые данные</H2>");
    printf("<P>%s", szSrcBuf);

    // Перекодируем данные и отображаем результат
    // перекодировки
    DecodeStr(szSrcBuf);

    printf("<H2>Данные после перекодировки</H2>");
    printf("<P>%s", szSrcBuf);

    // Выводим в документ список значений полей формы
    strcpy(szBuf, szQueryString);
    printf("<H2>Список значений полей</H2>");

    szBuf[strlen(szBuf)] = '&';
    szBuf[strlen(szBuf) + 1] = '\0';

    for(szParam = szBuf;;)
    {
      szPtr = strchr(szParam, '&');
      if(szPtr != NULL)
      {
        *szPtr = '\0';
        DecodeStr(szParam);
        printf("%s<BR>", szParam);

        szParam = szPtr + 1;
        if(szParam >= (szBuf + lSize))
          break;
      }
      else
        break;
    }

    printf("</BODY></HTML>");
  }
}

// ------------------------------------------------
// Функция DecodeStr
// Раскодирование строки из кодировки URL
// ------------------------------------------------
void DecodeStr(char *szString)
{
  int src;
  int dst;
  char ch;

  // Цикл по строке
  for(src=0, dst=0; szString[src]; src++, dst++)
  {
    // Получаем очередной символ перекодируемой строки
    ch = szString[src];
```

```
    // Заменяем символ "+" на пробел
    ch = (ch == '+') ? ' ' : ch;

    // Сохраняем результат
    szString[dst] = ch;

    // Обработка шестнадцатеричных кодов вида "%xx"
    if(ch == '%')
    {
      // Выполняем преобразование строки "%xx"
      // в код символа
      szString[dst] = DecodeHex(&szString[src + 1]);
      src += 2;
    }
  }

  // Закрываем строку двоичным нулем
  szString[dst] = '\0';
}

// ------------------------------------------------
// Функция DecodeHex
// Раскодирование строки "%xx"
// ------------------------------------------------
char DecodeHex(char *str)
{
  char ch;

  // Обрабатываем старший разряд
  if(str[0] >= 'A')
    ch = ((str[0] & 0xdf) - 'A') + 10;
  else
    ch = str[0] - '0';

  // Сдвигаем его влево на 4 бита
  ch <<= 4;

  // Обрабатываем младший разряд и складываем
  // его со старшим
  if(str[1] >= 'A')
    ch += ((str[1] & 0xdf) - 'A') + 10;
  else
    ch += str[1] - '0';

  // Возвращаем результат перекодировки
  return ch;
}
```

Функция `main` программы CONTROLS вначале выводит в стандартный поток вывода `STDOUT` заголовок `HTTP` и начальные строки динамически формируемого документа HTML. Для вывода мы использовали функцию `printf`.

Далее функция `main` определяет использованный метод передачи данных, анализируя содержимое переменной среды `REQUEST_METHOD`. Это необходимо, так как при разных методах передачи необходимо использовать различные методы получения входных данных. Значение переменной среды программа получает при помощи функции `getenv`.

Если данные передаются методом `POST`, программа будет считывать их из стандартного потока ввода `STDIN`. Размер данных находится в переменной среды `CONTENT_LENGTH`.

Соответствующая текстовая строка получается функцией getenv и преобразуется в численное значение функцией atoi.

Чтение данных из входного потока выполняется за один вызов функции fread:

```
fread(szBuf, lSize, 1, stdin);
```

Этой функции мы передаем адрес буфера для записи принятых данных, размер данных, количество буферов, которые нужно считать, и входной поток.

Программа CGI может сохранить принятые данные в файле для дальнейшей обработки. Наша программа создает в текущем каталоге файл с названием received.dat:

```
fileReceived = fopen("received.dat", "w");
fwrite(szBuf, lSize, 1, fileReceived);
fclose(fileReceived);
```

Текущим каталогом при запуске этой программы будет каталог с загрузочным модулем программы CONTROLS. Заметим, что для того чтобы программа CGI могла создать файл в каталоге, необходимо соответствующим образом настроить права доступа.

После сохранения принятых данных в файле программа CONTROLS выводит в стандартный поток вывода содержимое некоторых переменных среды: REQUEST_METHOD, CONTENT_LENGTH и CONTENT_TYPE.

Далее наша программа выполняет перекодировку полученных данных из кодировки URL. Перед этим принятые данные копируются в буфер, где они будут обновляться по месту.

Для копирования мы закрываем буфер двоичным нулем, после чего копирование можно выполнить функцией копирования строки strcpy.

Перекодировка выполняется функцией DecodeStr. После перекодирования результат будет находиться в буфере szSrcBuf, откуда он и берется для отображения.

На завершающем этапе обработки данных, полученных от формы, программа CONTROLS записывает в выходной документ HTML значения отдельных полей. Напомним, что эти значения имеют формат &<Имя_поля>=<Значение>, при этом символ “&” используется как разделитель.

Наша программа закрывает исходный буфер с принятыми данными дополнительным символом “&” (для простоты сканирования), после чего запускает цикл по полям формы.

В этом цикле во входной строке с помощью функции strchr ищется символ разделитель. Если этот символ найден, он заменяется на двоичный нуль, после чего полученная текстовая строка значения параметра перекодируется функцией DecodeStr и выводится в выходной поток STDOUT.

Цикл завершается тогда, когда в процессе сканирования указатель текущей позиции выходит за границы буфера данных.

В конце программа CONTROLS закрывает документ HTML, записывая в него команды </BODY> и </HTML>.

Если данные передаются в программу CONTROLS методом GET, входные данные находятся в переменной среды QUERY_STRING, которую мы получаем следующим образом:

```
szQueryString = getenv("QUERY_STRING");
```

Обработка принятых данных выполняется аналогично тому, как это делается при методе POST. Разница заключается лишь в том, что в документе мы отображаем содержимое только переменных REQUEST_METHOD и QUERY_STRING.

Теперь займемся перекодировкой принятых данных, которая выполняется функцией DecodeStr.

Эта функция сканирует входную строку, заменяя символы “+” на пробелы. Если в перекодируемой строке встречается комбинация символов вида "%xx", она заменяются на однобайтовый код соответствующего символа с помощью функции DecodeHex.

Функция DecodeHex комбинирует значение кода символа из старшего и младшего разряда преобразуемой комбинации символов.

6.6.3. Программа AREF

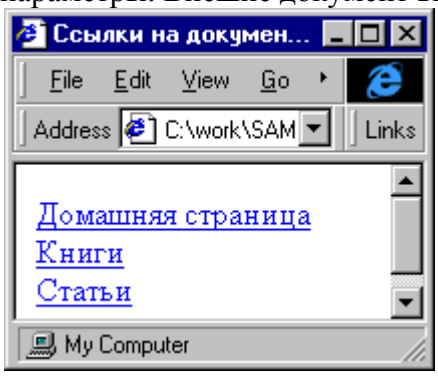
До сих пор в наших примерах мы использовали программы CGI только для обработки данных из полей форм, указывая адрес URL загрузочного файла программы в параметре ACTION оператора <FORM>. Однако есть и другая возможность вызова программ CGI: вы можете указать их адрес в параметре HREF оператора ссылки <A>. В этом случае вы можете передать программе CGI параметры, указав их после имени файла загрузочного модуля через разделительный символ “?”. Программа получит строку параметров методом GET и сможет извлечь ее из переменной среды с именем QUERY_STRING.

Пример документа HTML, в котором демонстрируется вызов программы CGI указанным выше способом:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
  <HEAD>
    <TITLE>Ссылки на документы HTML</TITLE>
  </HEAD>
  <BODY BGCOLOR=#FFFFFF>

    <A HREF=/cgi-bin/aref.exe?page1">Домашняя страница</A><BR>
    <A HREF=/cgi-bin/aref.exe?page2">Книги</A><BR>
    <A HREF=/cgi-bin/aref.exe?page3">Статьи</A><BR>
  </BODY>
</HTML>
```

В этом документе есть три ссылки на программу CGI с именем aref.exe, причем каждый раз ей передаются разные параметры. Внешне документ выглядит так:



Программа CGI принимает параметр и в зависимости от его значения отображает один из документов HTML.

Исходный текст программы AREF достаточно прост и приведен в листинге:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(int argc, char *argv[])
{
  char * szQueryString;

  szQueryString = getenv("QUERY_STRING");

  if(!strcmp(szQueryString, "page1"))
    printf("Location: home.htm\n\n");
  else if(!strcmp(szQueryString, "page2"))
    printf("Location: books.htm\n\n");
  else if(!strcmp(szQueryString, "page3"))
    printf("Location: capital.htm\n\n");
  else
    printf("Location: error.htm\n\n");
```

```
}
```

Программа получает значение переменной среды QUERY_STRING, пользуясь для этого функцией getenv. Далее она сравнивает значение параметра со строками “page1”, “page2” и “page3”. При совпадении программа возвращает навигатору адрес URL соответствующего документа HTML, формируя заголовок HTTP специального вида:

```
Location: <Адрес URL документа HTML или графического изображения>\n\n
```

Когда навигатор получает от сервера WWW такой заголовок, он отображает в своем окне документ или файл графического изображения, адрес URL которого указан в заголовке.

Таким образом, программа CGI может анализировать параметры, поступающие от навигатора через ссылку или поля формы, а затем не только динамически формировать документ HTML для отображения в окне навигатора, но и возвращать ссылки на уже существующие документы в виде их адресов URL.

Эта возможность может пригодиться вам для организации ссылок на документы HTML через списки, создаваемые оператором <SELECT>, расположенном в форме. Программа CGI может определить, какая строка была выбрана в списке в момент посылки заполненной формы серверу WWW, и в зависимости от этого либо возвратить ссылку на тот или иной существующий документ HTML, либо сформировать новый документ HTML динамически.

6.6.4. Программа COUNTER

Почти на каждом сервере WWW в сети Internet вы можете встретить счетчик посещений. По его показаниям можно судить о посещаемости сервера, что имеет, например, значение при выборе сервера для размещения рекламы.

Существуют различные методы создания счетчиков, доступность которых во многом определяется программным обеспечением и настройкой сервера WWW, а также доброй волей поставщика услуг Internet (если ваш сервер WWW виртуальный и физически расположен у поставщика).

Самый простой способ создания счетчика заключается в следующем. Те документы HTML, на которых необходимо разместить счетчик, преобразуется в файл шаблона. В этом файле в том месте, где должно располагаться текстовое значение счетчика, необходимо поместить последовательность заранее определенных символов, например, символы “xxxxx” или “~~~~~”.

Пример такого файла шаблона вы можете найти в листинге:

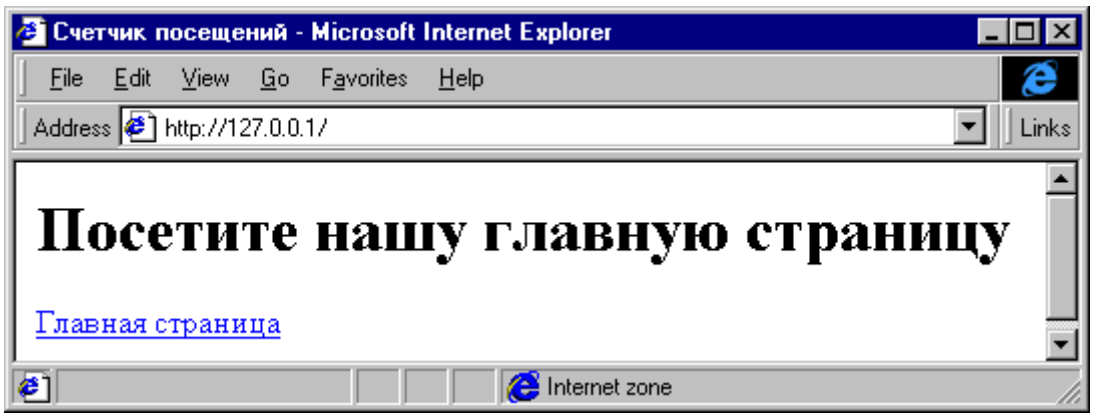
```
<HTML>
  <BODY BGCOLOR="#FFFFFF">
    <H1>Главная страница фирмы XYZ Inc.</H1>
    <P>Добро пожаловать на нашу главную страницу!    <HR>
    <P>Вы посетитель номер <B>~~~~~</B> с 1 января 1913 года
  </BODY>
</HTML>
```

Мы назвали файл home.tm, хотя вы можете выбирать для файлов шаблона любые имена.

Далее вам нужно сделать ссылку на документ, содержащий счетчик. Такая ссылка указывает на программу CGI, выполняющую подсчет посещений. В листинге ссылка на программу счетчика выполнена с использованием оператора <A>.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
  <HEAD>    <TITLE>Счетчик посещений</TITLE>      </HEAD>
  <BODY BGCOLOR=#FFFFFF>
    <H1>Посетите нашу главную страницу</H1>
    <A HREF=/cgi-bin/counter.exe?>Главная страница</A><BR>
  </BODY>
</HTML>
```

Внешний вид этого документа:



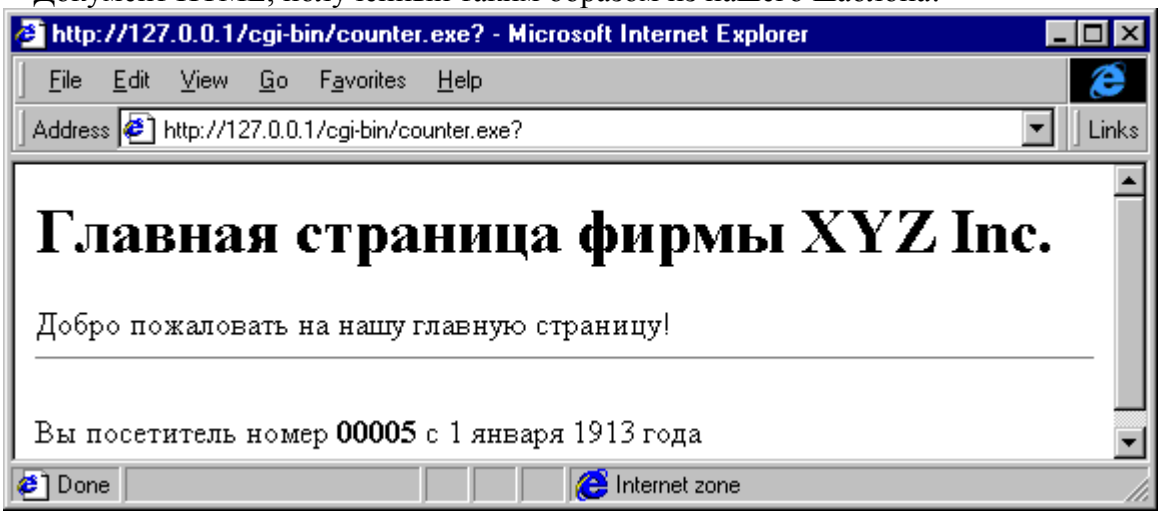
Обратите внимание, что вслед за именем программы мы разместили разделительный символ “?”, после которого вы можете указывать дополнительные параметры, например, номер счетчика. Это может пригодиться, если ваша программа CGI должна выполнять отдельный подсчет посещений для различных документов HTML.

Получив управление, программа счетчика может считать файл шаблона в оперативную память, найти там поле счетчика и заменить его на текущее количество посещений.

А как подсчитывать посещения?

Это тоже несложно. Первоначально вы можете создать при помощи любого текстового редактора файл CNTDAT.DAT, содержащий начальное значение счетчика в текстовом виде, например, 00000. Программа счетчика при обращении к странице могла бы открывать этот файл, считывать значение счетчика, увеличивать его на единицу и снова сохранять в том же файле. После того как текущее значение счетчика посещений будет записано в соответствующее поле шаблона, программа счетчика может вывести шаблон в стандартный поток вывода `STDOUT`.

Документ HTML, полученный таким образом из нашего шаблона:



Исходный текст программы COUNTER, работающей с использованием описанного выше алгоритма, приведен в листинге:

```
// ================================================
// Программа CGI counter.c
// Реализация счетчика посещений
//

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <io.h>
#include <fcntl.h>
```

```
void main(int argc, char *argv[])
{
  // Идентификатор файла шаблона главной страницы
  HFILE hSrcFile;

  // Идентификатор файла счетчика
  FILE *CounterFile;

  // Размер файла шаблона
  DWORD dwFileSize;

  // Адрес шаблона главной страницы
  LPSTR szTemplate;

  // Временный буфер для работы со счетчиком
  CHAR  szBuf[10];

  // Текущее значение счетчика
  INT   nCounter;

  // Указатель на поле счетчика в шаблоне
  LPSTR szCounterPtr;

  // ------------------------------------------------
  // Считываем файл шаблона в оперативную память
  // ------------------------------------------------

  // Открываем файл шаблона
  hSrcFile = _lopen("HOME.TM", OF_READ);

  // Определяем размер файла в байтах
  dwFileSize = _llseek(hSrcFile, 0, 2);

  // Устанавливаем указатель текущей позиции
  // на начало файла шаблона
  _llseek(hSrcFile, 0, 0);

  // Получаем память для буфера шаблона
  szTemplate = malloc(dwFileSize);

  // Загружаем шаблон в буфер
  _hread(hSrcFile, szTemplate, dwFileSize);

  // ------------------------------------------------
  // Увеличиваем значение счетчика в файле
  // ------------------------------------------------

  // Открываем файл счетчика для чтения
  CounterFile = fopen("CNTDAT.DAT", "r");

  // Читаем из файла строку значения счетчика
  fgets(szBuf, 7, CounterFile);

  // Закрываем файл счетчика
  fclose(CounterFile);

  // Преобразуем значение счетчика из текстовой
  // строки в численную величину
```

```
  sscanf(szBuf, "%d", &nCounter);

  // Увеличиваем значение счетчика
  nCounter++;

  // Записываем в буфер szBuf пять цифр нового
  // значения счетчика
  sprintf(szBuf, "%05.5ld", nCounter);

  // Сохраняем новое значение счетчика в файле
  CounterFile = fopen("CNTDAT.DAT", "w");
  fprintf(CounterFile, "%d", nCounter);
  fclose(CounterFile);

  // ------------------------------------------------
  // Заменяем 5 цифр значения счетчика на новые
  // в буфере шаблона
  // ------------------------------------------------

  // Ищем маркер поля счетчика
  szCounterPtr = strstr(szTemplate, "~~~~~");

  // Копируем в это поле новое значение счетчика
  if(szCounterPtr != NULL)
    strncpy(szCounterPtr, szBuf, 5);

  // Выводим заголовок HTTP и разделительную строку
  printf("Content-type: text/html\n\n");

  // Выводим шаблон с измененным значением
  // поля счетчика
  fwrite(szTemplate, dwFileSize, 1, stdout);

  // Освобождаем буфер шаблона
  free(szTemplate);
}
```

Получив управление, программа COUNTER считывает файл шаблона в оперативную память. Для упрощения исходного текста программы мы применили для работы с файлом функции `_lopen`, `_llseek` и `_hread`, которые являются специфичными для операционной системы Microsoft Windows. Вы можете использовать здесь любые другие функции, предназначенные для работы с файлами.

Перед чтением файла шаблона мы определяем его длину в байтах, для чего используем известный прием с установкой текущей позиции на конец файла функцией `_llseek`. После того как размер файла будет сохранен в переменной `dwFileSize`, текущая позиция снова устанавливается на начало файла шаблона. Вслед за этим программа CGI динамически заказывает память для буфера и читает в этот буфер файл шаблона за один вызов функции `_hread`.

Далее программа открывает файл счетчика и считывает из него одну текстовую строку, пользуясь для этого функцией `fgets`. Предполагается, что в этой строке находится значение счетчика в символьном виде. После чтения файл счетчика закрывается, а полученное значение преобразуется в численную величину и увеличивается на единицу.

Далее новое значение счетчика сохраняется в файле счетчика и записывается в символьном виде в буфер `szBuf`, откуда будет выполняться вставка нового значения счетчика в шаблон документа.

Процедура вставки выполняется после обновления содержимого файла счетчика и за-

ключается в том, что найденное с помощью функции `strstr` поле шаблона обновляется из буфера `szBuf`.

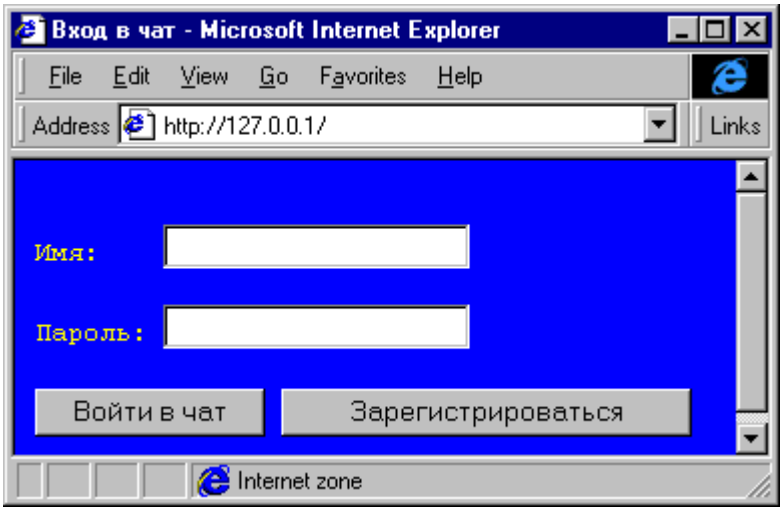
После обновления шаблона в стандартный поток вывода записывается обычный заголовок HTTP (функцией `printf`) и шаблон (функцией `fwrite`). На последнем этапе освобождается буфер памяти, заказанный для буфера шаблона.

Приложение. Лабораторная работа 6. Интерфейс CGI

6.1. Простой чат

Для функционирования данной программы необходимо наличие в корневом каталоге Web-сервера непустого каталога avatar, в котором необходимо разместить один или более файлов, содержащих графические изображения, идентифицирующие пользователей. Для входа в чат используется форма:

```
<html>
  <head>
    <title>Вход в чат </title>
  </head>
  <body bgcolor=blue>
    <font color=yellow>
    <form method=get action=/cgi-bin/chat.exe>
      <pre>
      <p>Имя:    <input type=text name=name>
      <p>Пароль: <input type=password name=pass></pre>
      <input type=submit name=enter value="Войти в чат"> 
      <input type=submit name=reg value="Зарегистрироваться">
    </form>
    </font>
  </body>
</html>
```



Текст программы chat.c:

```
#include <stdio.h>   //для стандартной библиотеки ввода-вывода
#include <stdlib.h>
#include <cstring.h> //для класса "строка"
#include <vector>    //для векторного типа
#include <fstream.h> //для файловых потоков C++

//класс для работы с элементами строки запроса
class convertor
{
  //для хранения пар вида "имя=значение"
  std::vector<string> name,value;
  //функция декодирования строки из кодировки URL
  static string decode(char *);
 public:
```

```
  //конструктор - преобразует строку запроса в набор пар
  //имя-значение, декодируя из кодировки URL
  convertor(char *p);
  //возвращает значение для данного имени поля
  string getvaluebyname(char *);
  //возвращает имя поля по порядковому номеру его появления в форме
  string getnamebyindex(unsigned int);
  //возвращает значение поля по порядковому номеру его появления в форме
  string getvaluebyindex(unsigned int);
  //проверка на существование поля с заданным именем
  bool isnameexists(char *);
  //возвращает число полей в форме
  int getcount() {return name.size();}
  //преобразует строку в кодировку URL
  static string encode(const char *);
};


//конструктор - преобразует строку запроса в набор пар
//имя-значение, декодируя из кодировки URL
convertor::convertor(char *p)
{
  //выделяем память под копию строки запроса
  char *str=new char[strlen(p)+1],*temp;

  //копируем строку запроса в копию
  strcpy(str,p);
  //устанавливаем указатель на начало строки запроса
  temp=str;
  //выделяем из строки запроса пары вида "имя=значение"
  for(char *res=strtok(temp,"&");res;res=strtok(NULL,"&"))
  {
    //выделяем память под копию строки с парой
    char *s=new char[strlen(res)+1];
    //копируем строку с парой в копию
    strcpy(s,res);
    //ищем разделитель пары - знак "="
    char *d=strchr(s,'=');
    //если не находим
    if(!d)
    {
      //освобождаем память из-под копии строки с парой
      delete[] s;
      //продолжаем разбор
      continue;
    }
    //в позицию знака "=" ставим конец строки, разделяя имя и значение
    *d=0;
    //помещаем имя поля в вектор имен
    name.push_back(s);
    //переходим на первый символ значения
    d++;
    //декодировав, помещаем значение поля в вектор значений
    value.push_back(decode(d));
    //освобождаем память из-под копии строки с парой
    delete[] s;
  }
  //освобождаем память из-под копии строки запроса
```

```
  delete[] str;
}


//возвращает значение для данного имени поля
string convertor::getvaluebyname(char *t)
{
  //в цикле перебираем все поля
  for(int i=0;i<getcount();i++)
    //если поле с заданным именем существует
    if(name[i]==t)
      //возвращаем его
      return value[i];
  //если поле с заданным именем не существует, возвращаем пустую строку
  return "";
}


//возвращает имя поля по порядковому номеру его появления в форме
string convertor::getnamebyindex(unsigned int n)
{
  //если номер поля корректен
  if(n<getcount())
    //возвращаем имя поля
    return name[n];
  //пустая строка как признак отсутствия поля
  return "";
}


//возвращает значение поля по порядковому номеру его появления в форме
string convertor::getvaluebyindex(unsigned int n)
{
  //если номер поля корректен
  if(n<getcount())
    //возвращаем значение поля
    return value[n];
  //пустая строка как признак отсутствия имени
  return "";
}


//проверка на существование поля с заданным именем
bool convertor::isnameexists(char* t)
{
  //в цикле перебираем все поля
  for(int i=0;i<getcount();i++)
    //если поле с заданным именем существует
    if(name[i]==t)
      //возвращаем истину
      return true;
  //если поле с заданным именем не существует, возвращаем ложь
  return false;
}


//функция декодирования строки из кодировки URL
string convertor::decode(char *s)
```

```
{
  //строка для хранения результата
  string res;
  //для каждого элемента строки
  for(int i=0;i<strlen(s);i++)
  {
    //знак "+" заменяется на пробел
    if(s[i]=='+')
      res+=' ';
    else
      //обычные символы копируются
      if(s[i]!='%')
        res+=s[i];
      else
      {
        //последовательности, заданные кодом вида "%xx",
        //преобразуются в символы
        char temp[3]={0,0,0};
        int d;
        //копируем два символа кода
        strncpy(temp,s+i+1,2);
        //преобразуем их в числовое значение
        sscanf(temp,"%x",&d);
        i+=2;
        //приклеиваем к строке результата символ с заданным кодом
        res+=(char)d;
      }
  }
  //возвращаем строку результата
  return res;
}


//преобразует строку в кодировку URL
string convertor::encode(const char *s)
{
  //строка для хранения результата
  string res;
  //для каждого элемента строки
  for(int i=0;i<strlen(s);i++)
  {
    //пробелы меняем на знак "+"
    if(s[i]==' ')
      res+='+';
    else
      //латиница копируется
      if(isalpha((unsigned char)s[i]))
        res+=s[i];
      else
      {
        //остальные символы кодируются строкой вида "%xx"
        char temp[4]={0,0,0};
        sprintf(temp,"%%%02X",(unsigned char)s[i]);
        //приклеиваем к строке результата код символа
        res+=temp;
      }
  }
  //возвращаем строку результата
```

```
  return res;
}


//функция печати регистрационной формы;
//параметр - сообщение об ошибке
void printregform(const char *s)
{
  //для хранения указателя на поток ввода из процесса dir
  FILE *fp;
  //для хранения имени файла с аватарой
  char avname[100];
  //определяем тип ответа - программа на HTML
  printf("Content-Type: text/html\n\n");
  printf("<html>\n"
         //выводим заголовок
         "  <head>\n"
         "    <title>Регистрация нового пользователя</title>\n"
         "  </head>\n"
         //цвет фона устанавливаем в синий
         "  <body bgcolor=blue>\n"
         "    <font color=white><b>Ваши данные для регистрации в ча-
те</b></font>%s\n"
         "    <font color=yellow>\n"
         //выводим форму
         "    <form method=get action=/cgi-bin/chat.exe>\n"
         "      <pre>"
         "      <br>Ваше
имя:         <input
type=text name=name>\n"
         "      <br>Введите пароль:   <input type=password
name=pass1>\n"
         "      <br>Повторите пароль: <input type=password
name=pass2>\n"
         "<p><p>       Выберите Ваш аватар:</pre><p>",s);
  //синхронизируемся
  fflush(stdout);
  //открываем поток ввода для чтения результата просмотра командой dir
  //каталога с автарами пользователей чата
  fp=_popen("dir /b ..\\avatar","rt");
  //цикл чтения имен файлов с аватарами
  for(int i=0;fgets(avname,100,fp);i++)
  {
    //при достижении конца файла
    if(feof(fp))
      //прерываем цикл
      break;
    //убираем символ перевода строки
    avname[strlen(avname)-1]=0;
    //выводим поле-переключатель с изображением аватара
    //по 5 изображений в строке
    printf("<input type=radio name=avatar value=%s checked>",avname);
    printf("<img src=/avatar/%s> ",avname);
    if(i==4)
    {
      i=0;
      printf("\n<p>");
    }
```

```
  }
  //закрываем канал с командой просмотра
  _pclose(fp);
  printf("      \n"
         "      <pre><p>Число сообщений, видимых в чате: "
         //задаем список для выбора числа просматриваемых сообщений
         "<SELECT NAME=\"number\">"
         "  <OPTION SELECTED>20"
         "  <OPTION>40"
         "  <OPTION>80"
         "  <OPTION>160"
         "</SELECT>"
         "\n<p>"
         "      <input type=submit value=\"Зарегистрироваться\">\n"
         "    </form>\n"
         "    </font>\n"
         "  </body>\n"
         "</html>\n");
}

//вывод основной формы чата
void printchatform(const char *username)
{
  //вектор для хранения имени и аватара
  std::vector<string> userdata[2],
  //вектор для хранения имени и сообщения
                      chat[2];
  //количество отображаемых сообщений
  int limit=0;
  //открываем файл с профилями пользователей
  ifstream f="passwd";
  while(f)
  {
    char name[200],pwd[200],aw[200],cnt[200];
    //читаем 4 строки -
    //имя
    f.getline(name,199);
    //пароль
    f.getline(pwd,199);
    //файл с аватаром
    f.getline(aw,199);
    //количество отображаемых сообщений
    f.getline(cnt,199);
    //если не конец файла
    if(f)
    {
      //добавляем имя и аватар в вектор
      userdata[0].push_back(name);
      userdata[1].push_back(aw);
      //если это данные текущего пользователя
      if(!strcmp(name,username))
        //считываем его ограничения
        sscanf(cnt,"%d",&limit);
    }
  }
  //определяем тип ответа
  printf("Content-Type: text/html\n\n");
  printf("<html><head>"
```

```
          //в заголовке указываем необходимость автоматического обновления
          //страницы раз в минуту, имитируя ввод пустого сообщения
          "<META HTTP-EQUIV=\"Refresh\" CONTENT=\"60; URL=/cgi-
bin/chat.exe?user=%s&msg=\">"
          "<title>Чат (%s)</title></head><body bgcolor=#111111>\n",
          //для корректной работы имя пользовтаеля в строке запроса
          //преобразуем в кодировку URL
          convertor::encode(username).c_str(),username);
  //открываем лог чата
  ifstream fc="chat";
  while(fc)
  {
    char name[200],msg[300];
    //построчно читаем имя и сообщение
    fc.getline(name,199);
    fc.getline(msg,299);
    //при обнаружении конца файла прерываем цикл
    if(!fc)
      break;
    //добавляем имя и сообщение в вектор
    chat[0].push_back(name);
    chat[1].push_back(msg);
  }
  //если в файле более 300 сообщений
  if(chat[0].size()>300)
  {
    //удаляем сообщения из начала вектора до тех пор,
    //пока их не останется 200
    while(chat[0].size()>200)
    {
      chat[0].erase(chat[0].begin());
      chat[1].erase(chat[1].begin());
    }
    //открываем лог чата для записи
    ofstream f="chat";
    //вывод в файл последние 200 сообщений
    for(int i=0;i<200;i++)
      f<<chat[0][i]<<endl<<chat[1][i]<<endl;
  }
  //удаляем сообщения из начала вектора до тех пор,
  //пока их не останется столько, сколько пользователь
  //выбрал для просмотра
  while(chat[0].size()>limit)
  {
    chat[0].erase(chat[0].begin());
    chat[1].erase(chat[1].begin());
  }
  //выводим форму для ввода сообщения
  printf("<form method=get action=/cgi-bin/chat.exe>\n");
  printf("<input type=hidden name=user value=\"%s\">\n",username);
  printf("<input type=text name=msg size=40 MAXLENGTH=200> ");
  printf("<input type=submit value=\"Отправить!\">\n");
  printf("</form>\n");
  printf("<pre><table>\n");
  //выводим сообщения так, чтобы последнее было сверху
  for(int i=chat[0].size()-1;i>=0;i--)
  {
    printf("<tr>\n<td>");
```

```
    //выводим аватару по имени пользователчч
    for(int k=0;k<userdata[0].size();k++)
      if(chat[0][i]==userdata[0][k])
        printf("<img src=/%s>",userdata[1][k].c_str());
    //сообщения в разных строках выделяем цветами
    printf("</td><td><font color=%s><b>%s</b></font></td>"
           "<td><font color=%s>%s</font></td>\n",
           (i%2)?"yellow":"blue",
           chat[0][i].c_str(), (i%2)?"lime":"silver",
           chat[1][i].c_str());
    printf("</tr>");
  }
  printf("</pre></table>");
  printf("</body></html>\n");
}

//основная программа
main()
{
  //указатель на строку запроса
  char *p;

  //получем строку запроса
  p=getenv("QUERY_STRING");
  //при ошибке диагностируем ее
  if(!p)
  {
    printf("This CGI-program must be runned only by Web-server
software\n");
    exit(0);
  }
//  printf("Content-Type: text/plain\n\n%s\n\n",p);
  //преобразуем строку запроса в список пар
  convertor res(p);
  //проверка имени и пароля пользователя
  if(res.isnameexists("enter"))
  {
    //читаем файл с профилями пользователей
    ifstream f="passwd";
    //пока не конец файла
    while(f)
    {
      char name[200],pwd[200],aw[200],cnt[200];
      //читаем имя, пароль, аватар, ограничения
      f.getline(name,199);
      f.getline(pwd,199);
      f.getline(aw,199);
      f.getline(cnt,199);
      //если пользователь с указанными данными существует

if(name==res.getvaluebyname("name")&&pwd==res.getvaluebyname("pass")&&res
.getvaluebyname("name")!="")
      {
         //входим в чат
         printchatform(name);
         return 0;
      }
    }
```

```
    //если пользователь с указанными данными не существует,
    //предланм повторить ввод
    printf("Content-Type: text/html\n\n");
    printf("<html>\n"
           "  <head>\n"
           "    <title>Вход в чат </title>\n"
           "  </head>\n"
           "  <body bgcolor=blue>\n"
           "    <font color=white>Ошибка при вводе имени или паро-
ля</font>\n"
           "    <font color=yellow>\n"
           "    <form method=get action=/cgi-bin/chat.exe>\n"
           "      <pre>\n"
           "      <p>Имя:    <input type=text
name=name>\n"
           "      <p>Пароль: <input type=password
name=pass></pre>\n"
           "      <input type=submit name=enter value=\"Войти в
чат\"> \n"
           "      <input type=submit name=reg
value=\"Зарегистрироваться\">\n"
           "    </form>\n"
           "    </font>\n"
           "  </body>\n"
           "</html>\n");
    return 0;
  }

  //вывод формы для регистрации
  if(res.isnameexists("reg"))
  {
    printregform("");
    return 0;
  }
  //проверка регистрационных данных на корректность
  if(res.isnameexists("pass1"))
  {
    if(res.getvaluebyname("name")=="")
      printregform("<p><font color=red>Не задано имя, повторите
ввод</font>");
    else
      if(res.getvaluebyname("pass1")=="")
        printregform("<p><font color=red>Не задан пароль, повторите
ввод</font>");
      else
        if(res.getvaluebyname("pass1")!=res.getvaluebyname("pass2"))
          printregform("<p><font color=red>Пароли не совпадают, повторите
ввод</font>");
        else
        {
          ifstream f="passwd";
          while(f)
          {
            char name[200],pwd[200],aw[200],cnt[200];
            f.getline(name,199);
            f.getline(pwd,199);
            f.getline(aw,199);
            f.getline(cnt,199);
```

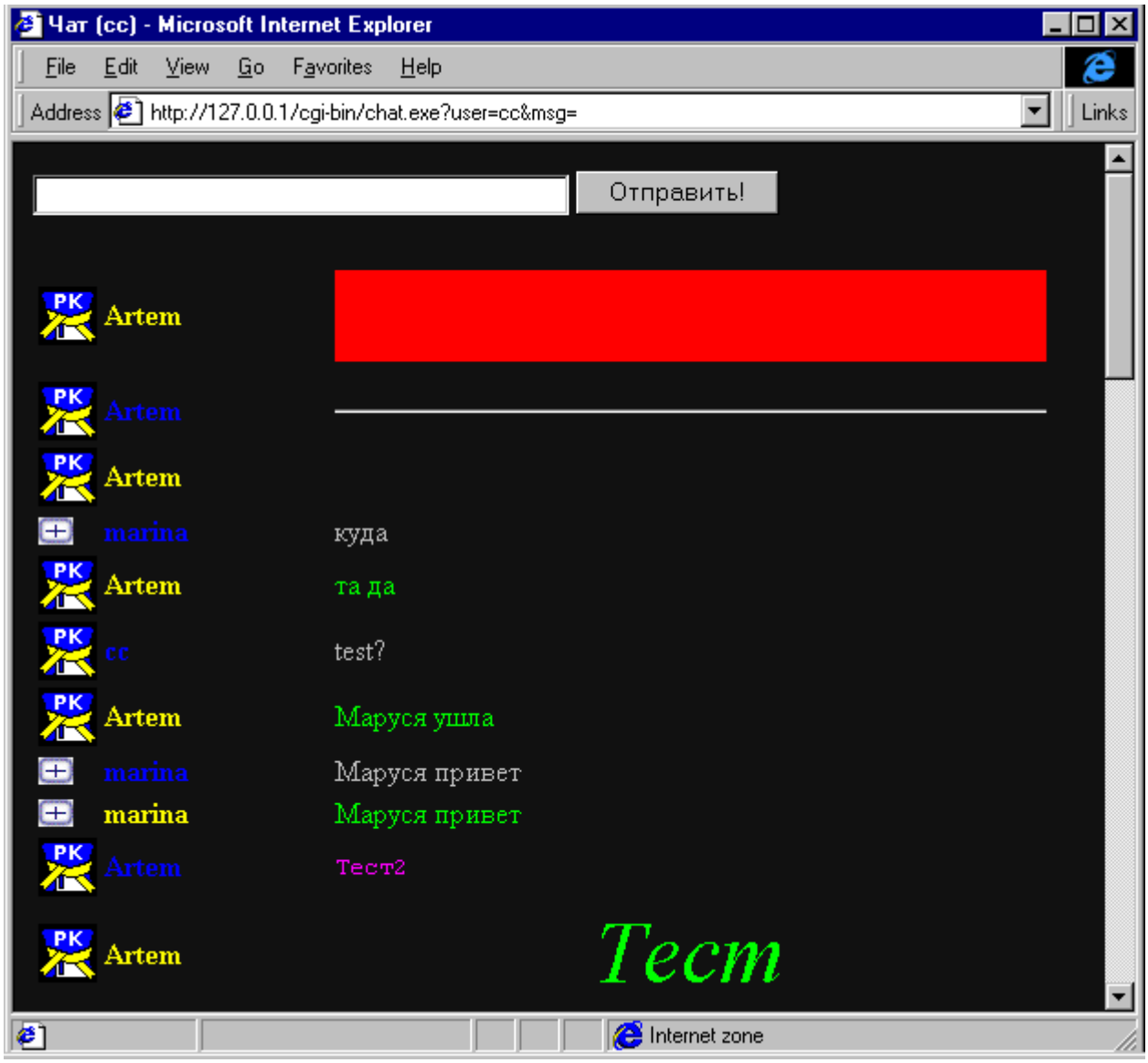
```
            //если такой пользователь уже есть

if(name==res.getvaluebyname("name")&&res.getvaluebyname("name")!="")
            {
              printregform("<p><font color=red>Имя занято, повторите
ввод</font>");
              return 0;
            }
          }
          f.close();
          //добавляем данные нового пользователя в файл профилей
          ofstream out("passwd",ios::app);
          out<<res.getvaluebyname("name")<<endl;
          out<<res.getvaluebyname("pass1")<<endl;
          out<<"avatar/"<<res.getvaluebyname("avatar")<<endl;
          out<<res.getvaluebyname("number")<<endl;
          printchatform(res.getvaluebyname("name").c_str());
        }
    return 0;
  }
  //если введено новое сообщение
  if(res.isnameexists("msg"))
  {
    ifstream f="passwd";
    while(f)
    {
      char name[200],pwd[200],aw[200],cnt[200];
      f.getline(name,199);
      f.getline(pwd,199);
      f.getline(aw,199);
      f.getline(cnt,199);
      //если пользователь существует

if(name==res.getvaluebyname("user")&&res.getvaluebyname("user")!=""&&
         res.getvaluebyname("msg")!="")
      {
        //дабавляем его имя и сообщение в лог чата
        ofstream f("chat",ios::app);

f<<res.getvaluebyname("user")<<endl<<res.getvaluebyname("msg")<<endl;
        printchatform(res.getvaluebyname("user").c_str());
        return 0;
      }
    }
  }
  //если сообщение пустое
  if(res.isnameexists("msg"))
  {
    printchatform(res.getvaluebyname("user").c_str());
    return 0;
  }
  //при ошибках печатаем отладочную информацию
  printf("Content-Type: text/plain\n\nЭтого Вы не должны были уви-
деть:)\n%s\n",p);
  for(int i=0;i<res.getcount();i++)
    printf("name=%s,
value=%s\n",res.getnamebyindex(i).c_str(),res.getvaluebyindex(i).c_str())
;
```

```
}
```



При регистрации пользователя информация о нем добавляется в файл passwd вида:

```
Василиса Прекрасная
уродина
avatar/av2.gif
10
```

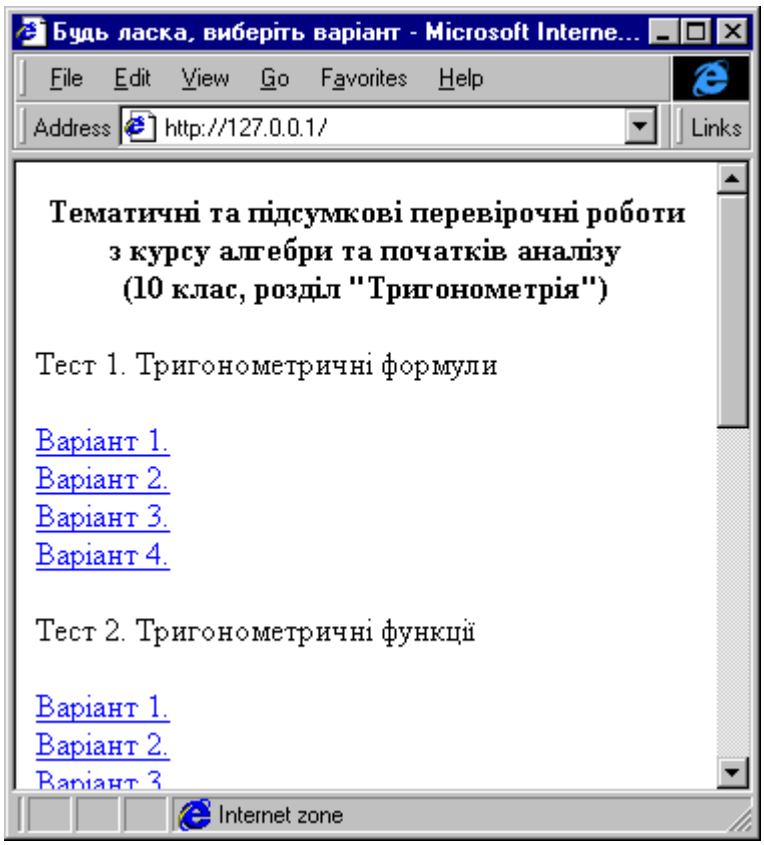
Первая строка определяет имя пользователя, вторая – пароль, третья – ссылку на аватар, а четвертая – количество отображаемых сообщений. Сами сообщения хранятся в файле chat в записях вида

```
marina
Маруся привет
```

В нечетных строках хранятся имена, в четных – однострочные сообщения.

6.2. Тестовая система

В корневом каталоге Web-сервера размещается файл index.html, ссылающийся на файлы test.html в различных каталогах:

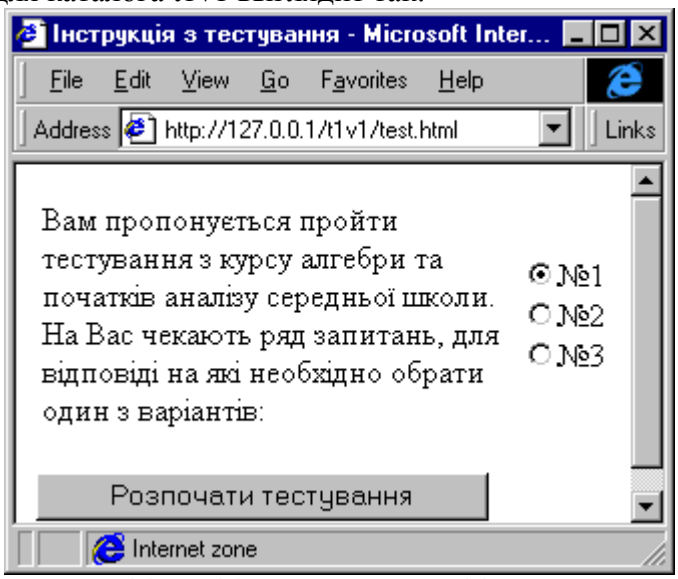



```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-
1251">
<TITLE>Будь ласка, виберіть варіант</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">

<B><FONT FACE="Times New Roman"><P ALIGN="CENTER">Тематичні та підсумкові
перевірочні роботи<BR>
з курсу алгебри та початків аналізу<BR>
(10 клас, розділ "Тригонометрія")</P>
</B><P>Тест 1. Тригонометричні формули</P>
</FONT><P><A HREF="t1v1/test.html"><U><FONT FACE="Times New Roman"
COLOR="#0000ff">Варіант 1.</U></FONT></A><FONT FACE="Times New
Roman"><BR>
<A HREF="t1v2/test.html">Варіант 2.<BR>
<A HREF="t1v3/test.html">Варіант 3.<BR>
<A HREF="t1v4/test.html">Варіант 4.</P></A>
<P>Тест 2. Тригонометричні функції</P>
<P><A HREF="t2v1/test.html">Варіант 1.<BR>
<A HREF="t2v2/test.html">Варіант 2.<BR>
<A HREF="t2v3/test.html">Варіант 3.<BR>
<A HREF="t2v4/test.html">Варіант 4.</P></A>
<P>Тест 3.Обернені тригонометричні функції </P>
<P><A HREF="t3v1/test.html">Варіант 1.<BR>
<A HREF="t3v2/test.html">Варіант 2.<BR>
<A HREF="t3v3/test.html">Варіант 3.<BR>
<A HREF="t3v4/test.html">Варіант 4.</P></A>
<P>Тест 4. Тригонометричні рівняння</P>
<P><A HREF="t4v1/test.html">Варіант 1.<BR>
<A HREF="t4v2/test.html">Варіант 2.<BR>
<A HREF="t4v3/test.html">Варіант 3.<BR>
```

```
<A HREF="t4v4/test.html">Варіант 4.</P></A>
<P>Тест 5. Тригонометричні нерівності</P>
<P><A HREF="t5v1/test.html">Варіант 1.<BR>
<A HREF="t5v2/test.html">Варіант 2.<BR>
<A HREF="t5v3/test.html">Варіант 3.<BR>
<A HREF="t5v4/test.html">Варіант 4.</P></A>
</FONT></BODY>
</HTML>
```

Файл test.html для каталога t1v1 выглядит так:




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
 <TITLE>Інструкція з тестування</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<FORM METHOD=POST ACTION="/cgi-bin/controls.exe">

<input type=hidden name="dir" value="t1v1">
<input type=hidden name="num" value="000">
<input type=hidden name="ans" value="">
<input type=hidden name="psv" value="">

  <TABLE>
    <TR>
      <TD VALIGN=TOP>Вам пропонується пройти тестування з курсу алгебри
та початків аналізу середньої школи. На Вас чекають ряд запитань, для
відповіді на які необхідно обрати один з варіантів:</TD>
      <TD>
        <INPUT TYPE=RADIO NAME="rad" VALUE="01" CHECKED>№1<BR>
        <INPUT TYPE=RADIO NAME="rad" VALUE="02">№2<BR>
        <INPUT TYPE=RADIO NAME="rad" VALUE="03">№3<BR>
      </TD>
    </TR>
  </TABLE>
<BR>
<INPUT TYPE=submit VALUE="Розпочати тестування">
</FORM>
</BODY>
</HTML>
```

Нажатие кнопки вызывает программу controls:

```
#include <io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <vector>
#include <cstring.h>
#include <fstream.h>


void DecodeStr(char *szString);
char DecodeHex(char *str);

struct formdata
{
  string dir,num,ans,psv,rad,chk,txt;
};

void printformdata(formdata data);
formdata getformdata(char *str);


//Структура для хранения вопроса
struct q
{
  string qtype,question,var[30],answers[30];
  int anscount,varcount;

  q():anscount(0),varcount(0){}
  q operator=(const q &x);
  bool operator==(const q &x);
  friend istream &operator>>(istream &is, q& temp);
};


  q q::operator=(const q &x)
  {
    qtype=x.qtype;
    question=x.question;
    anscount=x.anscount;
    varcount=x.varcount;
    for(int i=0;i<30;i++)
    {
      var[i]=x.var[i];
      answers[i]=x.answers[i];
    }
    return *this;
  }


  bool q::operator==(const q &x)
  {
    if(qtype==x.qtype&&
       question==x.question&&
       anscount==x.anscount&&
       varcount==x.varcount)
    {
      for(int i=0;i<30;i++)
```

```
      {
        if(var[i]==x.var[i]&&
           answers[i]==x.answers[i])
           ;
        else
          return false;
      }
      return true;
    }
    return false;
  }

istream &operator>>(istream &is, q& temp)
{
  string answer;
  char tempstr[2000];

  is>>temp.qtype;

  do
  {
    is.getline(tempstr,1998);
    //getline(is,temp.question);
    temp.question=tempstr;
  }while(is&&temp.question=="");

  if(temp.qtype=="txt")
  {
    temp.anscount=1;
    do
    {
      is.getline(tempstr,1998);
      temp.answers[0]=tempstr;
    }while(is&&temp.answers[0]=="");
  }
  else
  {
    is>>temp.varcount;
    for(int z=0;z<temp.varcount;z++)
    {
      do
      {
        is.getline(tempstr,1998);
        temp.var[z]=tempstr;
//        getline(is,temp.var[z]);
      }while(is&&temp.var[z]=="");
    }

    do
    {
      is.getline(tempstr,1998);
      answer=tempstr;
      //getline(is,answer);
    }while(is&&answer=="");

    if(temp.qtype=="rad")
    {
      temp.anscount=1;
```

```
      temp.answers[0]=answer;
    }
    else
    {
      char buf[200],*tok;
      strcpy(buf,answer.c_str());
      tok=strtok(buf," ");
      while(tok!=NULL)
      {
        temp.answers[temp.anscount]=tok;
        temp.anscount++;
        tok=strtok(NULL," ");
      }
    }
  }

  return is;
}


void main()
{
  std::vector<q> tests;
//  FILE * fileReceived, * fileNumbers;
  char * szQueryString;
  char buf[301],filename[300],testdir[300],smth[200],tempstr[2000];
  int count,curtest,i,numbers[1000];
  formdata data;


  // Вывод заголовка HTTP и разделительной строки
  cout<<"Content-type: text/html\n\n";
  randomize();

//  extern char **environ;

  //for(int i=0;environ[i];i++)
    //cout<<"<br>"<<environ[i];
//  return;


  // Вывод начального фрагмента документа HTML,
  // формируемого динамически
  cout<<"<!DOCTYPE HTML PUBLIC"
    " \"-//W3C//DTD HTML 3.2//EN\">"<<endl;


  //szQueryString = getenv("QUERY_STRING");
  int datasize=atoi(getenv("CONTENT_LENGTH"));
  szQueryString=new char[datasize+1];
  if(!szQueryString)
    return;
  read(0,szQueryString,datasize);
  szQueryString[datasize]=0;


  // cout<<szQueryString;//in comment
```

```
  strcpy(tempstr,szQueryString);
  data=getformdata(tempstr);

  sprintf(filename,"../%s/tests.txt",data.dir.c_str());

//  cout<<"<br>"<<filename<<endl;

//  sprintf(filename,"../11/tests.txt");//delete

  ifstream f(filename);

  f>>count;//ввод количества вопросов

  for(int i=0;i<count;i++)
  {
    q temp;
    f>>temp;
    tests.push_back(temp);
    numbers[i]=-1;
  }


  if(data.num=="000") //начинаем тестирование
  {
    curtest=0;
    //tmpnam(received);
    //fileReceived = fopen(received, "w+b");
    //fileNumbers = fopen("numbers", "w+b");
    for(int z=0;z<count;)
    {
      int p=random(count);
      if(numbers[p]==-1)
        numbers[p]=z++;
    }
    //fwrite(numbers,count*sizeof(int),1,fileNumbers);
    for(int z=0;z<count;z++)
    {
      sprintf(tempstr,"%03d ",numbers[z]);
      data.psv+=tempstr;
    }
//    cout<<"<br>"<<data.psv;
  }
  else //уже тестируемся
  {
    curtest=atoi(data.num.c_str());

    //fileReceived = fopen(received, "r+b");

    //fseek(fileReceived,curtest-1,SEEK_SET);

    //fwrite(szQueryString+6, 1, 1, fileReceived);

    //fileNumbers = fopen("numbers", "r+b");
    //fread(numbers,count*sizeof(int),1,fileNumbers);

    strcpy(tempstr,data.psv.c_str());
    char *tok=strtok(tempstr," ");
    numbers[0]=atoi(tok);
```

```
    for(int z=1;z<count;z++)
    {
      tok=strtok(NULL," ");
      //sprintf(tempstr,"%03d ",numbers[z]);
      //data.psv+=tempstr;
      numbers[z]=atoi(tok);
    }

    if(tests[numbers[curtest-1]].qtype=="txt")
      data.ans+="("+data.txt+") ";
    else
      if(tests[numbers[curtest-1]].qtype=="rad")
        data.ans+="("+data.rad+") ";
      else
        data.ans+=data.chk+" ";
  }
  //fclose(fileReceived);
  //fclose(fileNumbers);

  // printformdata(data); //in comment


  if(curtest!=count)
  {
    cout<<"<HTML>\n<HEAD>\n  <TITLE>";
    cout<<"Завдання №"<<(curtest+1)<<endl;
    cout<<"</TITLE>\n</HEAD>\n<BODY BGCOLOR=#FFFFFF>"<<endl;
    // Отображаем принятые данные

    cout<<"<FORM METHOD=POST ACTION=\"/cgi-bin/controls.exe\">"<<endl;

    cout<<"<input type=hidden name=\"dir\" value="<<data.dir<<">"<<endl;
    sprintf(tempstr,"%03d",curtest+1);
    data.num=tempstr;
    cout<<"<input type=hidden name=\"num\" value="<<data.num<<">"<<endl;
    cout<<"<input type=hidden name=\"ans\"
value=\""<<data.ans<<"\">"<<endl;
    cout<<"<input type=hidden name=\"psv\"
value=\""<<data.psv<<"\">"<<endl;

    cout<<"<font
size=+1><P><I>"<<tests[numbers[curtest]].question<<endl<<"</I></font></P>
<br>"<<endl;
    if(tests[numbers[curtest]].qtype=="txt")
    {
      cout<<"\n<input type=text name=\"txt\" value=\"\">\n<br>\n";
    }
    else
    {
//      cout<<"<br>"<<endl;
      for(int z=0;z<tests[numbers[curtest]].varcount;z++)
        smth[z]=0;
      for(int z=1;z<=tests[numbers[curtest]].varcount;)
      {
        int p=random(tests[numbers[curtest]].varcount);
        if(smth[p]==0)
          smth[p]=z++;
      }
```

```
      for(int z=0;z<tests[numbers[curtest]].varcount;z++)
      {
        if(tests[numbers[curtest]].qtype=="rad")
          sprintf(tempstr,"<INPUT TYPE=RADIO NAME=\"rad\" VALUE=\"%02d\">
%s",smth[z],tests[numbers[curtest]].var[smth[z]-1].c_str());
        else
          sprintf(tempstr,"<INPUT TYPE=checkbox NAME=\"chk%02d\"
VALUE=\"%02d\"> %s",smth[z],smth[z],tests[numbers[curtest]].var[smth[z]-
1].c_str());
        cout<<tempstr<<"<br>\n";
      }
    }
    cout<<"<BR><INPUT TYPE=submit VALUE=\"Відповісти\">";
  }
  else
  {
     cout<<"<H2>Тестування завершено</H2>";
     //fileReceived = fopen(received, "r+b");
     //fileNumbers = fopen("numbers", "r+b");
     //fread(numbers,count*sizeof(int),1,fileNumbers);
     int right=0;

     strcpy(tempstr,data.ans.c_str()+1);
     for(int i=0;;i++)
       if(tempstr[strlen(tempstr)-1-i]==')')
       {
         tempstr[strlen(tempstr)-1-i]=0;
         break;
       }

//     cout<<"<br>"<<tempstr<<"<br>"<<endl;

     for(char *p=0;(p=strstr(tempstr,") ("))!=NULL;*p='
',*(p+1)='\n',*(p+2)=' ');
     strcat(tempstr,"\n");

//     cout<<"<br>"<<tempstr<<"<br>"<<endl;

     char *tok=strtok(tempstr,"\n");
     for(int i=0;i<count;i++)
     {
       //fread(buf, 1, 1, fileReceived);
       //cout<<tok<<"<br>";
       while(tok[0]==' ')
         tok++;
       strcpy(buf,tok);
       while(buf[strlen(buf)-1]==' ')
         buf[strlen(buf)-1]=0;
       if(tests[numbers[i]].qtype=="txt")
       {
         if(tests[numbers[i]].answers[0]==buf)
           right++;
       }
       else
       {
         if(tests[numbers[i]].qtype=="rad")
         {
```

```
            if(atoi(tests[numbers[i]].answers[0].c_str())==atoi(buf))
              right++;
          }
          else//"chk"
          {
            char buf[200],smth[200],*t;
            int gotanscount=0;
            int partial=0;
            string gotans[30];

            strcpy(buf,tok);
            t=strchr(buf,' ');
            while(t!=NULL)
            {
              //strncpy(smth,buf,abs(t-buf));
              sscanf(buf,"%s",smth);
              //cout<<"<br>"<<smth;
              gotans[gotanscount]=smth;
              gotanscount++;
              strcpy(smth,t+1);
              strcpy(buf,smth);
              t=strchr(buf,' ');
            }

            if(gotanscount==tests[numbers[i]].anscount)
            {
              for(int k=0;k<gotanscount;k++)
              {
                for(int j=0;j<tests[numbers[i]].anscount;j++)

if(atoi(tests[numbers[i]].answers[j].c_str())==atoi(gotans[k].c_str()))
                  {
                    partial++;
                    break;
                  }
              }
              if(partial==gotanscount)
                right++;
            }

            /*
            for(int k=0;k<gotanscount;k++)
            {
              for(int j=0;j<tests[numbers[i]].anscount;j++)

if(atoi(tests[numbers[i]].answers[j].c_str())==atoi(gotans[k].c_str()))
                {
                  partial++;
                  break;
                }
            }
            */
            //right+=partial/tests[numbers[i]].anscount;
          }
        }

        tok=strtok(NULL,"\n");
        //if(buf[0]-'0'==tests[numbers[i]].answer)
```

```
        //  right++;
      }

      //fclose(fileReceived);
      //fclose(fileNumbers);
      //unlink("numbers");
      //unlink(received);
      cout<<"Ви правильно відповіли на "<<right<<" питань з
"<<count<<endl;
      //cout<<"<P>Оцінка - "<<12*right/count;
      //cout<<"<P><A HREF=\"test.html\" TARGET=\"main page\">Повторити те-
стування</P>";
      cout<<"<P><A HREF=\"/index.html\" TARGET=\"main page\">Повернутися
до вибору варіанту</P>";
   }
   cout<<"</FORM>"<<endl;
   cout<<"</BODY>\n</HTML>";
   delete[] szQueryString;
}

//разбиение данных формы на отдельные поля
formdata getformdata(char *str)
{
   formdata retval;
   char *tok,temp[2000];

   tok=strtok(str,"&");
   retval.chk="( ";
   while(tok!=NULL)
   {
     strcpy(temp,tok+4);
     DecodeStr(temp);
     if(strncmp(tok,"dir",3)==0) retval.dir=temp;
     if(strncmp(tok,"num",3)==0) retval.num=temp;
     if(strncmp(tok,"ans",3)==0) retval.ans=temp;
     if(strncmp(tok,"psv",3)==0) retval.psv=temp;
     if(strncmp(tok,"rad",3)==0) retval.rad=temp;
     if(strncmp(tok,"chk",3)==0)
       retval.chk+=string(strchr(temp,'=')+1)+" ";
     if(strncmp(tok,"txt",3)==0) retval.txt=temp;
     tok=strtok(NULL,"&");
   }
   retval.chk+=")";
   return retval;
}


void printformdata(formdata data)
{
   cout<<"<br>dir="<<data.dir<<endl;
   cout<<"<br>num="<<data.num<<endl;
   cout<<"<br>ans="<<data.ans<<endl;
   cout<<"<br>psv="<<data.psv<<endl;
   cout<<"<br>rad="<<data.rad<<endl;
   cout<<"<br>chk="<<data.chk<<endl;
   cout<<"<br>txt="<<data.txt<<endl;
}
```

```
// ------------------------------------------------
// Функция DecodeStr
// Раскодирование строки из кодировки URL
// ------------------------------------------------
void DecodeStr(char *szString)
{
  int src;
  int dst;
  char ch;

  // Цикл по строке
  for(src=0, dst=0; szString[src]; src++, dst++)
  {
    // Получаем очередной символ перекодируемой строки
    ch = szString[src];

    // Заменяем символ "+" на пробел
    ch = (ch == '+') ? ' ' : ch;

    // Сохраняем результат
    szString[dst] = ch;

    // Обработка шестнадцатеричных кодов вида "%xx"
    if(ch == '%')
    {
      // Выполняем преобразование строки "%xx"
      // в код символа
      szString[dst] = DecodeHex(&szString[src + 1]);
      src += 2;
    }
  }

  // Закрываем строку двоичным нулем
  szString[dst] = '\0';
}

// ------------------------------------------------
// Функция DecodeHex
// Раскодирование строки "%xx"
// ------------------------------------------------
char DecodeHex(char *str)
{
  char ch;

  // Обрабатываем старший разряд
  if(str[0] >= 'A')
    ch = ((str[0] & 0xdf) - 'A') + 10;
  else
    ch = str[0] - '0';

  // Сдвигаем его влево на 4 бита
  ch <<= 4;

  // Обрабатываем младший разряд и складываем
  // его со старшим
  if(str[1] >= 'A')
    ch += ((str[1] & 0xdf) - 'A') + 10;
  else
```

```
    ch += str[1] - '0';

  // Возвращаем результат перекодировки
  return ch;
}
```

Для работы программы в каждом из обслуживаемых ей каталогов должен находится файл tests.txt. Первая строка файла содержит количество задания, в последующих строках идут:

- тип вопроса: 1) rad – выбор одного варианта из нескольких; 2) chk – выбор нескольких вариантов из нескольких; 3) txt – ввод ответа в виде строки;
- текст вопроса (одна строка);
- количество вариантов (для rad, chk) или текст ответа (для txt);
- варианты ответов (для rad, chk), по одному в каждой строке;
- номер(а) вариант(а, ов) с правильным ответами (для rad, chk).

Например:

```
15

rad
Виразити 40<sup>o</sup> у радіанах.
3
pi/9
2pi/9
pi/18
2

txt
Функция отношения прилежащего катета к гипотенузе
cos

chk
Какие из перечисленных функций являются тригонометрическими?
5
sin
cos
exp
tg
ctg
1 2 4 5

rad
pi/90=...
3
4<sup>o</sup>
1<sup>o</sup>
2<sup>o</sup>
3

rad
Яка з трьох рівностей правильна?
3
sin(pi/2)=1
cos(pi)=0
ctg(pi/2)=1
1

rad
```

```
Яка з трьох нерівностей є правильною?
3
tg(1.7pi)>0
cos(5pi/6)<0
sin 195<sup>o</sup>>0
2

rad
sin 310<sup>o</sup>=sin(270<sup>o</sup>+40<sup>o</sup>)=...
3
sin 40<sup>o</sup>
cos 40<sup>o</sup>
-cos 40<sup>o</sup>
3

rad
tg 170<sup>o</sup>=tg(180<sup>o</sup>-10<sup>o</sup>)=...
3
-tg 10<sup>o</sup>
tg 10<sup>o</sup>
ctg 10<sup>o</sup>
1

rad
sin 72<sup>o</sup>*cos 2<sup>o</sup>-sin 2<sup>o</sup>*cos
72<sup>o</sup>=...
3
sin 74<sup>o</sup>
cos 74<sup>o</sup>
sin 70<sup>o</sup>
3

rad
2sin 4<sup>o</sup>*cos 4<sup>o</sup>=...
3
sin 2<sup>o</sup>
sin 8<sup>o</sup>
cos 2<sup>o</sup>
2

rad
sin 3<sup>o</sup>+sin 1<sup>o</sup>=...
3
2sin 1<sup>o</sup>*cos 2<sup>o</sup>
2sin 4<sup>o</sup>*cos 2<sup>o</sup>
2sin 2<sup>o</sup>*cos 1<sup>o</sup>
3

rad
cos 15<sup>o</sup>-cos 25<sup>o</sup>=...
3
2sin 5<sup>o</sup>*sin 20<sup>o</sup>
2cos 5<sup>o</sup>*cos 20<sup>o</sup>
2sin 10<sup>o</sup>*sin 40<sup>o</sup>
1

rad
Довжина дуги OA, зображеної на малюнку, дорівнює 1/6 довжини кола.<Br>
```

```
Записати загальну формулу чисел, що відповідають точкам A,B,C,D.<BR><IMG
SRC="/t1v1/1.gif">
3
<sup>+</sup>-pi/3+pi*n, nєZ
pi/3+pi*n/2, nєZ
pi/6+pi*n/2, nєZ
1

rad
Обчислити площу сектора круга радіуса 5см, дуга якої містить 4 радіан.
3
40 см кв.
50 см кв.
100 см кв.
2

rad
Яка з трьох нерівностей правильна?
3
sin 0.7<0
tg 5>0
cos 2<0
3

rad
sin(x-pi/2)=...
3
cos x
-cos x
sin x
2

rad
Якщо tg x=2, то (sin x+cos x)/(sin x - cos x)=...
3
3
1/2
2
1
```

В текст можно вставлять любые команды (в частности, ссылки на рисунки).

6.3. Психологическое тестирование

В качестве примера приведем программу для оценивания профиля личности. Форма программы выглядит так:

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-
1251">
    <TITLE>МЕТОДИКА МНОГОФАКТОРНОГО ИССЛЕДОВАНИЯ ЛИЧНОСТИ</TITLE>
  </HEAD>
<BODY>

<h1><center>МЕТОДИКА МНОГОФАКТОРНОГО ИССЛЕДОВАНИЯ ЛИЧНОСТИ,</h1>
<p>Или об особенностях вашего характера, склонностях, интересах.</center>
<p align=right>                                    Предложена Р. Кэттел-
```

```
лом</p>
<p>Цель. Опросник предназначен для измерения шестнадцати факторов личности и дает многогранную информацию о личностных чертах, которые называют конституционными факторами.
<p>      Опросник содержит 187 вопросов, на которые предлагается ответить обследуемым (взрослым людям с образованием не ниже 8-9 классов). Испытуемому предлагают занести в регистрационный бланк
<p>      Испытуемому предлагают занести в регистрационный бланк один из вариантов ответа на вопрос: да, нет, не знаю (или a,b,c).

<p><b>Инструкция.</b>  Вам предлагается ответить на ряд вопросов, цель которых выяснить
          особенности вашего характера, склонности и интересы. Не существует вопросом, на
          которые можно дать "правильный" или "неправильный" ответы, так как они
          отражают лишь особенности, присущие различным людям. Если же вы хотите
          рекомендации, правильно отражающие проявлений вашего характера в различных
          ситуациях, старайтесь отвечать как можно более точно и правдиво.
<p>          Отвечая на вопрос, вы можете выбрать один из предложенных вариантов ответов.
<p>          Номер ответа на бланке должен соответствовать номеру вопроса. Выбрав ответ <i>a</i>,
          перечеркните крестиком левый квадратик, если ответ <i>b</i> - то средний квадратик,
          ответу <i>c</i> соответствует правый квадратик.
<p>Отвечая, помните:
<p><ul>
<li>вопросы слишком короткие, чтобы в них содержались необходимые подробности, представляйте типичные ситуации, не задумывайтесь над деталями;
<li>не тратьте времени на раздумья, давайте первый естественный ответ, который приходит вам в голову;
<li>старайтесь отвечать на несколько вопросов в минуту, тогда вы закончите работу примерно за 35 минут;
<li>старайтесь избегать промежуточных, "неопределенных" ответов, кроме тех случаев, когда определенно ответить действительно невозможно (не более одного "неопределенного" ответа на 5-6 вопросов);
<li>не пропускайте ничего, обязательно отвечайте на все вопросы подряд;
<li>возможно, некоторые вопросы вам будет сложно отнести к себе, постарайтесь дать наиболее подходящий ответ. Не старайтесь произвести своими ответами благоприятное впечатление. Свободно выражайте свое собственное мнение
</ul>
<form method=get action=/cgi-bin/psitest.exe>
  <input type=hidden name=num value=0>
  <input type=hidden name=var value=0>
  <input type=hidden name=ans value="">
  <input type=submit value="Начать тестирование">
</form>
</BODY>
</HTML>
```

При нажатии кнопки вызывается программа psitest:

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>

struct vopros
{
  char text[500];
  char var[3][100];
};



vopros getvopros(int num)
{
  FILE *fp;
  int qnum,i,k;
  char temp[500];
  vopros res;

  fp=fopen("psitest.txt","r");
  if(fp==NULL)
  {
    printf("Ошибка открытия файла psitest.txt");
    exit(2);
  }
  fscanf(fp,"%d",&qnum);
  if(qnum<0)
  {
    printf("Неверное число вопросов");
    exit(2);
  }
  if(num>qnum)
  {
    printf("Вопроса %d не существует",num);
    exit(2);
  }

  for(i=0;i<num;i++)
  {
     do
     {
       fgets(temp,490,fp);
       if(feof(fp))
         break;
       if(temp[strlen(temp)]=='\n')
         temp[strlen(temp)]='\0';
     }while(strncmp(temp,"rad",3));

     if(feof(fp))
       break;

     do
     {
       fgets(temp,490,fp);
       if(feof(fp))
         break;
       if(temp[strlen(temp)]=='\n')
         temp[strlen(temp)]='\0';
     }while(strlen(temp)==0);
```

```
      if(feof(fp))
        break;

      strcpy(res.text,temp);


      fscanf(fp,"%d",&k);
      fgets(temp,490,fp);

      if(feof(fp))
        break;

      for(k=0;k<3;k++)
      {
        do
        {
          fgets(temp,490,fp);
          if(feof(fp))
            break;
          if(temp[strlen(temp)]=='\n')
            temp[strlen(temp)]='\0';
        }while(strlen(temp)==0);
        if(feof(fp))
          break;
        strcpy(res.var[k],temp);
      }
      if(feof(fp))
        break;
  }
  if(i!=num)
  {
    printf("Content-Type: text/html\n\n");
    printf("<html>\n");
    printf("<body>\n");
    printf("Вопрос %d в файле не существует",num);
    printf("</body>\n");
    printf("</html>\n");
    exit(2);
  }

  return res;

}


struct key
{
  int qnum;
  int abc[3];
};


key factor_a[]=
{
  {  1, {0,0,0}},
  {  2, {0,0,0}},
  {  3, {2,1,0}},
  { 26, {0,1,2}},
```

```
  { 27, {0,1,2}},
  { 51, {0,1,2}},
  { 52, {2,1,0}},
  { 76, {0,1,2}},
  {101, {2,1,0}},
  {126, {2,1,0}},
  {151, {0,1,2}},
  {176, {2,1,0}},
  NULL
};


key factor_b[]=
{
  { 28, {0,1,0}},
  { 53, {0,1,0}},
  { 54, {0,1,0}},
  { 77, {0,0,1}},
  { 78, {0,1,0}},
  {102, {0,0,1}},
  {103, {0,1,0}},
  {127, {0,0,1}},
  {128, {0,1,0}},
  {152, {1,0,0}},
  {153, {0,0,1}},
  {177, {1,0,0}},
  {178, {1,0,0}},
  NULL
};


key factor_c[]=
{
  {  4, {2,1,0}},
  {  5, {0,1,2}},
  { 29, {0,1,2}},
  { 30, {2,1,0}},
  { 55, {2,1,0}},
  { 79, {0,1,2}},
  { 80, {0,1,2}},
  {104, {2,1,0}},
  {105, {2,1,0}},
  {129, {0,1,2}},
  {130, {2,1,0}},
  {154, {0,1,2}},
  {179, {2,1,0}},
  NULL
};


key factor_e[]=
{
  {  6, {0,1,2}},
  {  7, {2,1,0}},
  { 31, {0,1,2}},
  { 32, {0,1,2}},
  { 56, {2,1,0}},
  { 57, {0,1,2}},
```

```
  { 81, {0,1,2}},
  {106, {0,1,2}},
  {131, {2,1,0}},
  {155, {2,1,0}},
  {156, {2,1,0}},
  {180, {2,1,0}},
  {181, {2,1,0}},
  NULL
};


key factor_f[]=
{
  {  8, {0,1,2}},
  { 33, {2,1,0}},
  { 58, {2,1,0}},
  { 82, {0,1,2}},
  { 83, {2,1,0}},
  {107, {0,1,2}},
  {108, {0,1,2}},
  {132, {2,1,0}},
  {133, {2,1,0}},
  {157, {0,1,2}},
  {158, {0,1,2}},
  {182, {2,1,0}},
  {183, {2,1,0}},
  NULL
};


key factor_g[]=
{
  {  9, {0,1,2}},
  { 34, {0,1,2}},
  { 59, {0,1,2}},
  { 84, {0,1,2}},
  {109, {2,1,0}},
  {134, {2,1,0}},
  {159, {0,1,2}},
  {160, {2,1,0}},
  {184, {2,1,0}},
  {185, {2,1,0}},
  NULL
};


key factor_h[]=
{
  { 10, {2,1,0}},
  { 35, {0,1,2}},
  { 36, {2,1,0}},
  { 60, {0,1,2}},
  { 61, {0,1,2}},
  { 85, {0,1,2}},
  { 86, {0,1,2}},
  {110, {2,1,0}},
  {111, {2,1,0}},
  {135, {2,1,0}},
```

```
  {136, {2,1,0}},
  {161, {0,1,2}},
  {186, {2,1,0}},
  NULL
};


key factor_i[]=
{
  { 11, {0,1,2}},
  { 12, {2,1,0}},
  { 37, {2,1,0}},
  { 62, {0,1,2}},
  { 87, {0,1,2}},
  {112, {2,1,0}},
  {137, {0,1,2}},
  {138, {2,1,0}},
  {162, {0,1,2}},
  {163, {2,1,0}},
  {187, {0,0,0}},
  NULL
};


key factor_l[]=
{
  { 13, {0,1,2}},
  { 38, {2,1,0}},
  { 63, {0,1,2}},
  { 64, {0,1,2}},
  { 88, {2,1,0}},
  { 89, {0,1,2}},
  {113, {2,1,0}},
  {114, {2,1,0}},
  {139, {0,1,2}},
  {164, {2,1,0}},
  NULL
};


key factor_m[]=
{
  { 14, {0,1,2}},
  { 15, {0,1,2}},
  { 39, {2,1,0}},
  { 40, {2,1,0}},
  { 65, {2,1,0}},
  { 90, {0,1,2}},
  { 91, {2,1,0}},
  {115, {2,1,0}},
  {116, {2,1,0}},
  {140, {2,1,0}},
  {141, {0,1,2}},
  {165, {0,1,2}},
  {166, {0,1,2}},
  NULL
};
```

```
key factor_n[]=
{
  { 16, {0,1,2}},
  { 17, {2,1,0}},
  { 41, {0,1,2}},
  { 42, {2,1,0}},
  { 66, {0,1,2}},
  { 67, {0,1,2}},
  { 92, {0,1,2}},
  {117, {2,1,0}},
  {142, {2,1,0}},
  {167, {2,1,0}},
  NULL
};


key factor_o[]=
{
  { 18, {2,1,0}},
  { 19, {0,1,2}},
  { 43, {2,1,0}},
  { 44, {0,1,2}},
  { 68, {0,1,2}},
  { 69, {2,1,0}},
  { 93, {0,1,2}},
  { 94, {2,1,0}},
  {118, {2,1,0}},
  {119, {2,1,0}},
  {143, {2,1,0}},
  {144, {0,1,2}},
  {168, {0,1,2}},
  NULL
};


key factor_q1[]=
{
  { 20, {2,1,0}},
  { 21, {0,1,2}},
  { 45, {0,1,2}},
  { 46, {2,1,0}},
  { 70, {2,1,0}},
  { 95, {0,1,2}},
  {120, {0,1,2}},
  {145, {2,1,0}},
  {169, {2,1,0}},
  {170, {0,1,2}},
  NULL
};


key factor_q2[]=
{
  { 22, {0,1,2}},
  { 47, {2,1,0}},
  { 71, {2,1,0}},
  { 72, {2,1,0}},
```

```
  { 96, {0,1,2}},
  { 97, {0,1,2}},
  {121, {0,1,2}},
  {122, {0,1,2}},
  {146, {2,1,0}},
  {171, {2,1,0}},
  NULL
};


key factor_q3[]=
{
  { 23, {0,1,2}},
  { 24, {0,1,2}},
  { 48, {2,1,0}},
  { 73, {2,1,0}},
  { 98, {2,1,0}},
  {123, {0,1,2}},
  {147, {0,1,2}},
  {148, {2,1,0}},
  {172, {0,1,2}},
  {173, {2,1,0}},
  NULL
};


key factor_q4[]=
{
  { 25, {0,1,2}},
  { 49, {2,1,0}},
  { 50, {2,1,2}},
  { 74, {2,1,0}},
  { 75, {0,1,2}},
  { 99, {2,1,0}},
  {100, {0,1,2}},
  {124, {2,1,0}},
  {125, {0,1,2}},
  {149, {2,1,0}},
  {150, {0,1,2}},
  {174, {2,1,0}},
  {175, {0,1,2}},
  NULL
};


struct factor
{
  char *factor_name;
  key *factor_key;
  int factor_value;
  int stena_value;
};


factor test[20]=
{
  {"A", factor_a},
  {"B", factor_b},
```

```
  {"C", factor_c},
  {"E", factor_e},
  {"F", factor_f},
  {"G", factor_g},
  {"H", factor_h},
  {"I", factor_i},
  {"L", factor_l},
  {"M", factor_m},
  {"N", factor_n},
  {"O", factor_o},
  {"Q<sub><small>1</small></sub>", factor_q1},
  {"Q<sub><small>2</small></sub>", factor_q2},
  {"Q<sub><small>3</small></sub>", factor_q3},
  {"Q<sub><small>4</small></sub>", factor_q4},
  {"F<sub><small>1</small></sub>", NULL},
  {"F<sub><small>2</small></sub>", NULL},
  {"F<sub><small>3</small></sub>", NULL},
  {"F<sub><small>4</small></sub>", NULL}
};


int stena[6][16][10]={
//int women1618[16][10]
{
  { 0, 7, 8, 9,11,12,14,16,17,19 },//A
  { 0, 2, 3, 4, 5, 6, 7, 8,10,11 },//B
  { 0, 7, 9,11,13,15,17,19,21,22 },//C
  { 0, 4, 5, 7, 9,11,13,16,18,20 },//E
  { 0, 7, 9,12,15,17,19,21,23,24 },//F
  { 0, 6, 8,10,12,14,15,17,18,19 },//G
  { 0, 3, 5, 8,10,13,15,18,21,23 },//H
  { 0, 6, 8, 9,11,12,14,15,16,18 },//I
  { 0, 3, 4, 6, 7, 9,10,12,14,15 },//L
  { 0, 7, 8,10,11,13,15,17,18,20 },//M
  { 0, 6, 7, 8, 9,11,12,14,15,16 },//N
  { 0, 5, 6, 8,10,12,14,16,18,20 },//O
  { 0, 4, 5, 6, 7, 9,10,12,13,15 },//Q1
  { 0, 4, 5, 7, 8,10,12,14,16,18 },//Q2
  { 0, 5, 7, 8,10,11,13,14,15,17 },//Q3
  { 0, 4, 6, 9,12,14,17,20,22,24 } //Q4
},


//int men1618[16][10]
{
  { 0, 4, 5, 7, 8,10,12,13,15,17 },//A
  { 0, 2, 3, 4, 5, 6, 7, 8,10,11 },//B
  { 0, 8,10,12,14,16,18,20,21,23 },//C
  { 0, 7, 9,10,12,14,16,18,20,22 },//E
  { 0, 6, 9,12,15,17,19,21,23,24 },//F
  { 0, 5, 7, 9,11,13,15,17,18,19 },//G
  { 0, 3, 5, 8,11,14,17,19,21,23 },//H
  { 0, 3, 4, 5, 7, 9,10,12,14,16 },//I
  { 0, 4, 5, 7, 9,10,12,14,15,17 },//L
  { 0, 5, 7, 8,10,12,14,15,17,19 },//M
  { 0, 6, 8, 9,10,11,13,14,16,17 },//N
  { 0, 4, 5, 7, 9,11,12,14,16,18 },//O
  { 0, 5, 6, 7, 9,10,12,13,14,16 },//Q1
```

```
  { 0, 4, 5, 7, 9,10,12,14,16,18 },//Q2
  { 0, 4, 6, 7, 9,11,12,14,15,17 },//Q3
  { 0, 3, 5, 7,10,13,16,18,20,22 } //Q4
},



//int women1928[16][10]
{
  { 0, 5, 7, 8,10,13,14,16,17,19 },//A
  { 0, 5, 6, 6, 7, 8, 9,10,11,12 },//B
  { 0, 7, 9,11,13,15,17,19,21,23 },//C
  { 0, 4, 5, 7, 9,11,13,15,17,19 },//E
  { 0, 6, 8,11,13,16,18,20,22,23 },//F
  { 0, 5, 7, 9,11,13,14,16,18,19 },//G
  { 0, 3, 5, 8,10,13,16,18,21,23 },//H
  { 0, 6, 7, 9,11,13,14,15,16,18 },//I
  { 0, 2, 4, 5, 6, 8,10,11,13,15 },//L
  { 0, 6, 8, 9,11,13,15,17,18,20 },//M
  { 0, 6, 7, 8, 9,11,12,14,15,17 },//N
  { 0, 4, 5, 7, 8,10,13,15,17,19 },//O
  { 0, 4, 5, 6, 8, 9,10,12,14,15 },//Q1
  { 0, 4, 5, 7, 8,10,12,14,16,18 },//Q2
  { 0, 5, 6, 8,10,11,13,14,15,17 },//Q3
  { 0, 4, 6, 8,11,13,16,19,21,23 } //Q4
},


//int men1928[16][10]
{
  { 0, 4, 5, 7, 8,10,12,14,15,17 },//A
  { 0, 5, 6, 6, 7, 8, 9,10,11,12 },//B
  { 0, 8,10,12,14,16,18,20,22,23 },//C
  { 0, 7, 9,10,12,14,17,19,20,22 },//E
  { 0, 6, 9,11,14,16,18,20,22,24 },//F
  { 0, 5, 7,10,12,13,15,17,18,20 },//G
  { 0, 3, 5, 8,11,14,17,19,21,23 },//H
  { 0, 3, 4, 6, 7, 9,11,13,15,16 },//I
  { 0, 4, 5, 7, 8,10,12,13,15,16 },//L
  { 0, 6, 7, 9,10,12,14,16,18,19 },//M
  { 0, 6, 8, 9,10,11,13,14,16,17 },//N
  { 0, 4, 5, 7, 9,10,12,14,16,18 },//O
  { 0, 5, 6, 7, 9,10,11,13,14,16 },//Q1
  { 0, 4, 5, 7, 8,10,12,14,16,18 },//Q2
  { 0, 4, 6, 7, 9,11,12,14,15,17 },//Q3
  { 0, 4, 5, 8,10,13,15,18,20,22 } //Q4
},


//int women2970[16][10]
{
  { 0, 5, 7, 9,11,12,14,16,17,19 },//A
  { 0, 2, 3, 4, 5, 6, 7, 8,10,11 },//B
  { 0, 8,10,12,14,16,18,21,23,25 },//C
  { 0, 3, 4, 6, 8,10,12,15,17,19 },//E
  { 0, 5, 7, 9,11,14,16,18,20,22 },//F
  { 0, 7, 8,10,12,14,16,17,18,20 },//G
  { 0, 3, 5, 8,10,13,15,18,21,23 },//H
```

```
  { 0, 6, 8, 9,11,12,14,15,17,18 },//I
  { 0, 2, 3, 5, 6, 8, 9,11,12,14 },//L
  { 0, 7, 8,10,12,13,15,17,18,20 },//M
  { 0, 6, 7, 8,10,11,12,14,15,16 },//N
  { 0, 4, 5, 7, 9,11,13,15,17,19 },//O
  { 0, 4, 5, 6, 8, 9,10,12,14,15 },//Q1
  { 0, 4, 5, 7, 9,10,12,14,16,18 },//Q2
  { 0, 6, 8, 9,11,12,14,15,17,18 },//Q3
  { 0, 3, 5, 8,11,13,16,18,21,23 } //Q4
},


//int men2970[16][10]
{
  { 0, 4, 5, 7, 8,10,12,14,15,17 },//A
  { 0, 2, 3, 4, 5, 6, 7, 8,10,11 },//B
  { 0, 8,11,13,15,17,18,20,22,24 },//C
  { 0, 6, 8,10,12,14,16,18,20,22 },//E
  { 0, 4, 6, 9,11,14,16,18,20,21 },//F
  { 0, 5, 8,11,13,14,16,18,19,20 },//G
  { 0, 4, 6, 9,12,15,17,20,22,24 },//H
  { 0, 3, 4, 5, 7, 9,11,13,15,16 },//I
  { 0, 3, 4, 6, 8, 9,11,13,14,16 },//L
  { 0, 6, 8, 9,11,12,14,16,18,20 },//M
  { 0, 7, 8,10,11,12,14,15,16,18 },//N
  { 0, 3, 4, 6, 8,10,12,13,16,18 },//O
  { 0, 5, 7, 8, 9,11,12,14,15,17 },//Q1
  { 0, 4, 5, 7, 9,11,12,14,16,18 },//Q2
  { 0, 5, 7, 9,10,12,13,15,16,18 },//Q3
  { 0, 1, 3, 6, 8,11,13,16,18,20 } //Q4
}

};


enum AGE
{
  WOMEN1618,
  MEN1618,
  WOMEN1928,
  MEN1928,
  WOMEN2970,
  MEN2970
};


enum FACTORS
{
  A, B, C, E, F,
  G, H, I, L, M,
  N, O, Q1, Q2, Q3,
  Q4, F1, F2, F3, F4
};


int getstena(int factornumber, int age)
{
  if(factornumber<F1)
```

```
  {
    for(int i=0;i<9;i++)
      if(stena[age][factornumber][i]<=test[factornumber].factor_value
         &&
test[factornumber].factor_value<stena[age][factornumber][i+1])
         return i+1;
    return 10;
  }
  else
    if(factornumber==F1)
      return ((38+2*test[L].stena_value+30+4*test[Q4].stena_value)-

2*(test[C].stena_value+test[H].stena_value+test[Q3].stena_value))/10;
    else
      if(factornumber==F2)
        return
((2*test[A].stena_value+3*test[E].stena_value+4*test[F].stena_value+
                5*test[H].stena_value)-(2*test[Q2].stena_value+11))/10;
      else
        if(factornumber==F3)
          return ((77+2*test[C].stena_value+2*test[E].stena_value+
                   2*test[F].stena_value+2*test[N].stena_value)-

(4*test[A].stena_value+6*test[I].stena_value+2*test[M].stena_value))/10;
        else //F4
          return ((4*test[E].stena_value+3*test[M].stena_value+
                   4*test[Q1].stena_value+4*test[Q2].stena_value)-
                  (3*test[A].stena_value+2*test[C].stena_value))/10;
}


void counfactor(char *ans,int age)
{
  for(int i=0;i<20;i++)
  {
    test[i].factor_value=0;
    if(test[i].factor_key)
    {
      for(int k=0;test[i].factor_key[k].qnum;k++)
        test[i].factor_value+=
           test[i].factor_key[k].abc[ans[test[i].factor_key[k].qnum-1]-
'1'];
    }
    test[i].stena_value=getstena(i,age);
  }

}


struct comment
{
  char *minus_name;
  char *minus_value;
  char *plus_name;
  char *plus_value;
};
```

```
comment interpretation[20]=
{
//A
  {
    "\"Замкнутость\" (шизотомия)",//-
    "Скрытый, обособленный, критичный, непреклонный, отчужденный,
необщительный, замкнутый, безучастный."
          "<br>Критичный; отстаивает свои идеи; холодный, отчужденный;
точный, объективный; недоверчивый, скептичный; непреклонный; холодный
(жесткий); сердитый, мрачный",
    "\"Общительность\" (аффектотимия)",//+
    "Сердечный, добрый, беспечный, открытый, естественный,
непринужденный."
          "<br>Добродушный, беспечный, готов к содружеству, предпочитает
присоединяться; внимательный к людям; мягкосердечный, небрежный;
доверчивый; легко приспосабливается, идет на поводу; сердечный, веселый"
  },
//B
  {
    "\"Низкий интеллект\"",//-
    "",
    "\"Высокий интеллект\"",//+
    ""
  },
//C
  {
    "\"Эмоциональная нестабильность\" (слабость эго)",//-
    "",
    "\"Эмоциональная стабильность\" (сила эго)",//+
    ""
  },
//E
  {
    "\"Покорность\"",//-
    "",
    "\"Доминирование\"",//+
    ""
  },
//F
  {
    "\"Невозмутимость\" (десургенсия)",//-
    "",
    "\"Импульсивность\" (сургенсия)",//+
    ""
  },
//G
  {
    "\"Корыстный\" (слабость суперэго)",//-
    "",
    "\"Совестливый\" (сила суперэго)",//+
    ""
  },
//H
  {
    "\"Робость\" (тректия)",//-
    "",
    "\"Смелость\" (пармия)",//+
    ""
```

```
  },
//I
  {
    "\"Жесткость\" (харрия)",//-
    "",
    "\"Мягкосердечность\" (премсия)",//+
    ""
  },
//L
  {
    "\"Доверчивость\" (алаксия)",//-
    "",
    "\"Подозрительность\" (протенсия)",//+
    ""
  },
//M
  {
    "\"Практичность\" (праксерния)",//-
    "",
    "\"Мечтательность\" (аутия)",//+
    ""
  },
//N
  {
    "\"Наивность\"",//-
    "",
    "\"Приницательность\"",//+
    ""
  },
//O
  {
    "\"Спокойствие\" (гипертимия)",//-
    "",
    "\"Тревожность\" (гипотимия)",//+
    ""
  },
//Q1
  {
    "\"Консерватизм\"",//-
    "",
    "\"Радикализм\"",//+
    ""
  },
//Q2
  {
    "Зависимость от группы",//-
    "",
    "Самодостаточность",//+
    ""
  },
//Q3
  {
    "Низкое самомнение",//-
    "",
    "Высокое самомнение",//+
    ""
  },
//Q4
```

```
  {
    "Низкая напряженность",//-
    "",
    "Высокая напряженность",//+
    ""
  },
//F1
  {
    "Низкая тревожность",//-
    "",
    "Высокая тревожность",//+
    ""
  },
//F2
  {
    "Интраверт",//-
    "",
    "Экстраверт",//+
    ""
  },
//F3
  {
    "Сензитивность",//-
    "",
    "Реактивная уравновешенность",//+
    ""
  },
//F4
  {
    "Конформность",//-
    "",
    "Независимость",//+
    ""
  }
};



main()
{
  char *p=getenv("QUERY_STRING");
  char ans[200],temp[200];
  int num,var,i;

  if(p==NULL)
  {
    printf("Программа должна вызывать только из-под Web-сервера\n");
    exit(1);
  }
  p+=4;
  for(i=0;*p!='&';i++)
  {
    temp[i]=*p;
    p++;
  }
  temp[i]='\0';
  num=atoi(temp);
  p+=5;
```

```
  var=*p-'0';
  p+=6;
  strcpy(ans,p);
  if(var!=0)
  {
    sprintf(temp,"%s%d",ans,var);
    strcpy(ans,temp);
  }

  if(num<187)
  {
    printf("Content-Type: text/html\n\n");
    printf("<html>\n");
    printf("<head>\n");
    printf("<title>Вопрос №%d из 187</title>\n",num+1);
    printf("<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html; char-
set=windows-1251\">\n");
    printf("</head>\n");
    printf("<body>\n");
    //printf("<p>num=%d,var=%d,ans=%s</p>",num,var,ans);
    vopros q=getvopros(num+1);
    printf("<p>%s\n",q.text);
    printf("<form method=get action=/cgi-bin/psitest.exe>");
    printf("<input type=hidden name=num value=%d>\n",num+1);
    printf("<input type=radio name=var value=1
checked>%s<br>\n",q.var[0]);
    printf("<input type=radio name=var value=2>%s<br>\n",q.var[1]);
    printf("<input type=radio name=var value=3>%s<br>\n",q.var[2]);
    printf("<input type=hidden name=ans value=%s>",ans);
    printf("<br><input type=submit value=\"Ответить\">\n");
    printf("</form>");
    printf("</body>\n");
    printf("</html>\n");
  }
  else
    if(num==187)//запрос данных для анализа
    {
      printf("Content-Type: text/html\n\n");
      printf("<html>\n");
      printf("<head>\n");
      printf("<title>Запрос данных для анализа</title>\n",num+1);
      printf("<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html;
charset=windows-1251\">\n");
      printf("</head>\n");
      printf("<body>\n");
      //printf("<p>num=%d,var=%d,ans=%s</p>",num,var,ans);
      printf("<p>Для анализа результатов тестирования Вам необходимо
выбрать свою возрастную группу\n");
      printf("<form method=get action=/cgi-bin/psitest.exe>");
      printf("<input type=hidden name=num value=%d>\n",num+1);
      printf("<input type=radio name=var value=0 checked>Женщина 16-18
лет<br>\n");
      printf("<input type=radio name=var value=2>Женщина 19-28
лет<br>\n");
      printf("<input type=radio name=var value=4>Женщина 29-70
лет<br>\n");
      printf("<input type=radio name=var value=1>Мужчина 16-18
лет<br>\n");
```

```
      printf("<input type=radio name=var value=3>Мужчина 19-28
лет<br>\n");
      printf("<input type=radio name=var value=5>Мужчина 29-70
лет<br>\n");
      printf("<input type=hidden name=ans value=%s>",ans);
      printf("<br><input type=submit value=\"Выбрать\">\n");
      printf("</form>");
      printf("</body>\n");
      printf("</html>\n");
    }
    else//анализ результатов
    {
      printf("Content-Type: text/html\n\n");
      printf("<html>\n");
      printf("<head>\n");
      printf("<title>Анализ результатов</title>\n",num+1);
      printf("<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html;
charset=windows-1251\">\n");
      printf("</head>\n");
      printf("<body>\n");
      //printf("Еще не сделан, ответы - %s\n",ans);

//counfactor("11123313132133232313211231223211321333221233321221212222232
22323331231112231223231221132221322123131332132122331313213313222323122233
32211313333212332311331321121221332323223311122132231",MEN1928);
      counfactor(ans,var);

      printf("<table align=center border=2><caption>Значения стен для
построения профиля личности:</caption>\n");

      printf("<tr align=center><th>Фактор</th>");
      for(int i=0;i<20;i++)
        printf("<td>%s</td>",test[i].factor_name);

      printf("</tr><tr align=center><th>Стена</th>");
      for(int i=0;i<20;i++)
        printf("<td>%d</td>",test[i].stena_value);

      printf("</tr></table><p>");

      printf("<table align=center border=2><caption>Интерпретация
факторов</caption>\n");
      printf("<tr>");
      printf("<th>Фактор</th>");
      printf("<th>Значение</th>");
      printf("<th>Влияние</th>");
      printf("<th>Интерпретация</th>");
      printf("</tr>");

      for(int i=0;i<20;i++)
      {
        printf("<tr align=center>");
        printf("<td>%s</td>",test[i].factor_name);
        printf("<td>%d</td>",test[i].stena_value);
        if( (test[i].stena_value>=1&&test[i].stena_value<=3) ||
(test[i].stena_value>=8&&test[i].stena_value<=10))
          printf("<td>пиковое</td>");
        else
```

```
          printf("<td>среднее</td>");
        if(test[i].stena_value<5.5)
          printf("<td
align=justify><i>%s</i><p>%s</td>\n",interpretation[i].minus_name,interpr
etation[i].minus_value);
        else
          printf("<td
align=justify><i>%s</i><p>%s</td>\n",interpretation[i].plus_name,interpre
tation[i].plus_value);
        printf("</tr>");
      }

      printf("</table><p>");

      printf("</body>\n");
      printf("</html>\n");
    }
}
```

Файл опросника psitest.txt:

```
187

rad
1. Я хорошо понял инструкцию, которую только что прочитал:
3
да
не уверен
нет

rad
2. Я готов отвечать на каждый вопрос так искренне, как только возможно:
3
да
не уверен
нет

rad
3. Я бы предпочел временами жить в доме, который находиться:
3
в обжитом пригороде
нечто среднее
в одиноко в глухих лесах

rad
4. Я чувствую в себе достаточно сил, чтобы справиться со своими
трудностями:
3
всегда
обычно
редко

rad
5. Я чувствую некоторое беспокойство при виде диких животных, даже если
они находятся в прочных клетках:
3
верно
не уверен
неверно
```

rad
6. Я воздерживаюсь от критики людей и их высказываний:
3
да
иногда
нет

rad
7. Я делаю саркастические(язвительные) замечания по поводу людей, если они этого, по- - моему, заслуживают:
3
обычно
иногда
никогда

rad
8. Мне больше нравиться классическая, чем эстрадная, музыка:
3
верно
не уверен
неверно

rad
9.Если бы я увидел дерущихся соседских детей, то я:
3
дал бы им возможность договориться самим
не уверен
рассудил бы их

rad
10. При общении с людьми я:
3
с готовностью вступаю в разговор
нечто среднее
предпочитаю спокойно оставаться в стороне

rad
11. По-моему, интереснее быть:
3
инженером-строителем
не уверен
драматургом

rad
12. Я остановился бы на улице, скорее чтобы посмотреть работу художника, чем слушать, как ссорятся люди:
3
верно
не уверен
неверно

rad
13. Обычно я могу ладить с самодовольными людьми, несмотря на то что они хвастаются или слишком много о себе воображают:
3
да
нечто среднее

нет

rad
14. По лицу человека всегда можно заметить, что он нечестный:
3
да
не уверен
нет

rad
15. было бы хорошо, если бы отпуск (каникулы) был более продолжителен и каждый был бы обязан его использовать:
3
согласен
не уверен
не согласен

rad
16. Я предпочел бы работу с возможно большим, но не постоянным зароботком, чем работу со скромным, но постоянным окладом:
3
да
не уверен
нет

rad
17. Я говорю о своих чувствах:
3
только если это необходимо
нечто среднее
охотно, когда предоставляется возможность

rad
18. Время от времени у меня возникает чувство неопределенной опасности или внезапного страха по непонятным причинам:
3
да
нечто среднее
нет

rad
19. Когда меня неправильно критикуют за что-то, в чем я не виноват, я:
3
не испытываю чувства вины
нечто среднее
все же чувствую себя виноватым

rad
20. За деньги можно купить почти все:
3
да
не уверен
нет

rad
21. Моим решением руководит больше:
3

сердце
сердце и разум в равной степени
разум

rad
22. Большинство людей были бы гораздо счастливее, если больше доверяли бы друг другу:
3
да
не уверен
нет

rad
23. Иногда, когда я смотрю в зеркало, мне трудно разобраться, где у меня правая, а где левая сторона:
3
верно
не уверен
неверно

rad
24. При разговоре я предпочитаю:
3
высказывать свои мысли так, как они приходят мне в голову
нечто среднее
сначала сформулировать получше свои мысли

rad
25. После того как меня что-то сильно рассердит, я довольно быстро успокаиваюсь6
3
да
нечто среднее
нет

rad
26.При одинаковом рабочем времени и заработке было интереснее работать:
3
плотником или поваром
не уверен
официантом в хорошем ресторане

rad
27. На общественные должности меня выбирали:
3
очень редко
иногда
много раз

rad
28. "Лопата" относиться к "копать", как "нож" относиться к:
3
"острый"
"резать"
"указывать"

rad
29. Иногда я не могу уснуть, потому что какая-нибудь мысль не выходит из

головы:
3
верно
не уверен
неверно

rad
30. В своей жизни я почти всегда достигаю поставленных целей:
3
верно
не уверен
неверно

rad
31. Устаревший закон следует отменить:
3
только после глубокого основательного обсуждения
не уверен
как можно скорее

rad
32. Я чувствую себя не в своей тарелке, когда мне приходиться работать над чем нибудь, что требует быстрых действий, результаты которых могут повлиять на других людей:
3
верно
нечто среднее
неверно

rad
33. Большинство знакомых считают меня интересным рассказчиком:
3
да
не уверен
нет

rad
34. Когда я вижу неряшливых, неопрятных людей, я:
3
принимаю их такими, как они есть
нечто среднее
испытываю отвращение и возмущение

rad
35. Я чувствую себя немного не по себе, если неожиданно оказываюсь в центре внимания группы людей:
3
да
нечто среднее
нет

rad
36. Я всегда рад оказаться среди людей, например в гостях, на танцах, на какой-либо коллективной встрече::
3
да
нечто среднее
нет

rad
37. В школе я предпочитал (или предпочитаю)
3
заниматься музыкой, пением
нечто среднее
выпиливать или мастерить что-либо

rad
38. Если меня назначают руководителем чего-либо, я настаиваю на том, чтобы мои указания выполнялись, иначе я отказываюсь от этой работы:
3
да
иногда
нет

rad
39. Важнее, чтобы родители:
3
помогали детям развивать свои чувства
нечто среднее
обучали детей сдерживать свои чувства

rad
40. Участвуя в групповой деятельности, я бы предпочел:
3
постараться улучшить организацию работы
нечто среднее
следить за результатами и соблюдением правил

rad
41. Время от времени у меня появляется потребность в интересной физической:
3
да
нечто среднее
нет

rad
42. Я предпочел бы скорее общаться с вежливыми людьми, чем с грубоватыми и любящими возражать:
3
да
нечто среднее
нет

rad
43. Я чувствую себя очень униженным , когда меня критикуют в присутствии группы людей:
3
верно
нечто среднее
неверно

rad
44. Если меня вызывает начальство, то я:
3
пользуюсь случаем, чтобы попросить о чем-то нужном мне

нечто среднее
боюсь, что это связано с какой-нибудь оплошностью в моей работе

rad
45. В наше время требуется:
3
больше спокойных, солидных людей;
не уверен;
больше "идеалистов", планирующих лучшее будущее.

rad
46. При чтении я сразу замечаю, когда автор произведения хочет меня в чем-то убедить:
3
да;
иногда;
нет

rad
47. В юности я принимал участие в нескольких спортивных мероприятиях:
3
иногда;
довольно часто;
многократно.

rad
48. Я поддерживаю порядок в моей комнате, все вещи всегда лежат на своих местах:
3
да;
нечто среднее;
нет.

rad
49. Иногда у меня возникает чувство напряжения и беспокойства, когда я вспоминаю, что произошло в течении дня:
3
да;
нечто среднее;
нет

rad
50. иногда я сомневаюсь, действительно ли люди, с которыми я разговариваю, интересуются тем, что я говорю:
3
да;
не уверен;
нет.

rad
51. Если бы пришлось выбирать, то я предпочел бы быть:
3
лесником;
не уверен;
учителем средней школы.

rad
52. На праздники и дни рождения я:

```
3
люблю делать подарки;
неопределенно;
считаю, что делать подарки - довольно неприятная вещь.

rad
53. "Усталый" относиться к "работе", как "гордый" к:
3
"улыбка";
"успех";
"счастливый".

rad
54. Какой из следующих предметов по существу отличается от двух других:
3
свеча;
луна;
электрический свет.

rad
55. Друзья меня подводили:
3
очень редко;
иногда;
довольно часто.

rad
56. У меня есть качества, по которым я определенно выше большинства
людей:
3
да;
не уверен;
нет.

rad
57. Когда я расстроен, я стараюсь скрыть свои чувства от других:
3
верно;
нечто среднее;
неверно.

rad
58. Я склонен посещать зрелищные мероприятия и развлечения:
3
чаще, чем раз в неделю ( т.е. чаще, чем большинство);
примерно раз в неделю ( т.е. как большинство);
реже, чем раз в неделю  (т.е. реже, чем  большинство).

rad
59. Я считаю, что возможность вести себя непринужденно важнее, чем
хорошие манеры и уважение к существующим правилам поведения:
3
да;
не уверен;
неверно.

rad
60. Обычно я молчу в присутствии старших по возрасту, опыту и положению:
```

3
да;
нечто среднее;
нет.

rad
61. Мне трудно говорить или декламировать перед большой группой людей:
3
да;
нечто среднее;
нет

rad
62. У меня хорошее чувство ориентировки в незнакомом месте (мне легко сказать где север - восток - юг - запад):
3
да;
нечто среднее;
нет

rad
63. Если кто-нибудь рассердиться на меня, то я:
3
постараюсь его успокоить;
нечто среднее;
раздражаюсь.

rad
64. Встречаясь с несправедливостью, я скорее склонен забыть об этом, чем реагировать:
3
верно;
не уверен;
неверно.

rad
65. Из моей памяти часто выпадают несущественные тривиальные вещи, например названия улиц, магазинов:
3
да;
нечто среднее;
нет

rad
66. Мне бы понравилась жизнь ветеринара, лечение и операции на животных:
3
да;
не уверен;;
нет

rad
67. Я ем со вкусом, не всегда так аккуратно и тщательно, как другие люди:
3
да;
не уверен;;
неверно.

rad

68. Бывают времена, когда у меня нет настроения видеть кого бы то ни было:
3
очень редко;
нечто среднее;
довольно часто.

rad
69. Иногда меня предупреждают о том, что в моем голосе и манерах слишком проявляется возбуждение:
3
да;
нечто среднее;
нет

rad
70. В юности, когда я расходился во мнении с родителями, то я:
3
оставался при своем мнении:
нечто среднее
соглашался с их авторитетом.

rad
71. Я предпочел бы заниматься самостоятельной работой, а не совместной с другими:
3
да;
не уверен;;
нет

rad
72. Мне бы больше понравилась спокойная жизнь в моем духе, чем слава и шумный успех:
3
верно;
не уверен;
неверно

rad
73. В большинстве случаев я чувствую себя зрелым человеком:
3
верно;
не уверен;
неверно

rad
74. Замечания в мой адрес, которые позволяют себе некоторые люди, меня больше расстраивают, чем помогают:
3
часто;
иногда;
никогда.

rad
75. Я всегда способен управлять проявлением своих чувств:
3
да;
нечто среднее;

нет

rad
76. Начиная работу над полезным изобретением, я бы предпочел:
3
разрабатывать его в лаборатории;
нечто среднее;
заниматься его практической реализацией.

rad
77. "Удивление" относится к "странный", как "страх" относиться к:
3
"смелый";
"тревожный";
"ужасный".

rad
78. Которая из последующих дробей отличается от двух других:
3
3/7
3/9
3/11

rad
79. Кажется, некоторые люди игнорируют и избегают меня, хотя я не знаю почему:
3
верно;
не уверен;
неверно

rad
80. отношение ко мне людей не соответствует моим добрым намерениям:
3
часто;
иногда;
никогда

rad
81. Употребление нецензурных выражений вызывает у меня возмущение, даже если не присутствуют лица другого пола:
3
да;
нечто среднее;
нет

rad
82.У меня определенно меньше друзей, чем у большинства людей:
3
да;
нечто среднее;
нет

rad
83. Я бы очень не хотел находиться в таком месте, где нет таких людей, с которыми можно поговорить:
3
верно;

нечто среднее
нет

rad
84. Люди иногда считают меня небрежным, хотя и думают, что я приятный человек:
3
да;
не уверен ;
нет.

rad
85. Волнение перед выступлением в присутствии многих людей я испытывал:
3
довольно часто;
иногда;
почти никогда.

rad
86. Когда я нахожусь в большой группе людей, то я предпочитаю молчать и предоставляю слово другим:
3
да;
нечто среднее;
нет..

rad
87. Я предрочитаю читать:
3
реалистические описания военных и политических сражений;
нечто среднее;
роман, где много чувств и воображения.

rad
88. Когда люди пытаются мной командовать, то я поступаю как раз наоборот:
3
да;
нечто среднее;
нет.

rad
89. Начальник или члены моей семьи критикуют меня только тогда, когда к этому действительно есть повод:
3
верно;
нечто среднее
неверно.

rad
90. На улицах или в магазинах мне не нравиться, когда некоторые люди пристально разглядывают других:
3
да;
нечто среднее;
нет.

rad
91. Во время длительной поездки я бы предпочел:

3
читать что-нибудь серьезное, но интересное;
неопределенно;
провести время, беседуя с кем-нибудь из пассажиров.

rad
92. В ситуациях, которые могут быть опасными, я громко разговариваю, хотя это выглядит невежливо и нарушает спокойствие:
3
да;
не уверен;
нет

rad
93. Если знакомые плохо обращаются со мной и показывают свою неприязнь , то:
3
меня совершенно это не трогает;
нечто среднее;
я расстраиваюсь.

rad
94. Я смущаюсь, когда меня хвалят или говорят мне комплименты:
3
да;
нечто среднее;
нет.

rad
95. Я бы предпочел иметь работу:
3
с постоянным окладом;
нечто среднее;
с большим окладом, который бы зависел от моей спообности показать людям , чего я стою.

rad
96. Чтобы быть информированным, я предпочитаю получать сведения:
3
в общении с людьми;
нечто среднее;
из литературы.

rad
97. Мне нравиться принимать активное участие в общественной работе:
3
да;
нечто среднее;
нет.

rad
98. При выполнении задания я удовлетворяюсь только тогда, когла должное внимание будет уделено всем мелочам:
3
верно;
не уверен;
неверно.

rad
99. Даже самые незначительные неудачи иногда меня слишком раздражают:
3
да;
нечто среднее;
нет.

rad
100. Сон у меня крепкий, я никогда не хожу и не разговариваю во сне:
3
да;
не уверен;
нет.

rad
101. Для меня интереснее работа, при которой
3
нужно разговаривать с людьми;
нечто среднее;
нужно заниматься счетами и записями.

rad
102. "Размер" так относиться к "длине", как "нечестный" к:
3
"тюрьма";
"нарушение";
"кража".

rad
103. "АБ" так относиться к "ГВ", как "СР" относиться к:
3
"ПО";
"ОП";
"ТУ".

rad
104. Когда люди ведут себя неразумно, то я:
3
молчу;
не уверен;
высказываю свое презрение.

rad
105. Если кто-нибудь громко разговаривает, когда я слушаю музыку:
3
могу сосредоточиться на музыке, не отвлекаться;
нечто среднее;
чувствую, что это портит мне удовольствие и раздражает.

rad
106. Меня лучше характеризовать как:
3
вежливого и спокойного;
нечто среднее;
энергичного.

rad
107. В общественных мероприятиях я принимаю участие только тогда, когда

это нужно, а в иных случаях избегаю их:
3
да;
не уверен;
нет.

rad
108. Быть осторожным и не ждать хорошего лучше, чем быть оптимистом и всегда ждать успеха:
3
верно;
не уверен;
неверно.

rad
109. Думая о трудностях в своей работе, я:
3
стараюсь планировать заранее , чем встретить трудность;
нечто среднее;
считаю, что справлюсь с трудностями по мере того как они возникнут.

rad
110. Мне легко вступить в контакт с людьми во время различных общественных мероприятий:
3
верно;
не уверен;
неверно.

rad
111. Когда требуется немного дипломатии и умения убедить, чтобы побудить людей что-либо сделать, обычно об этом просят меня:
3
верно;
не уверен;
неверно.

rad
112. Интересно быть:
3
консультантом, помогающим молодым людям выбирать профессию;
нечто среднее;
руководителем технического предприятия.

rad
113. Если я уверен, что человек несправедлив или ведет себя эгоистично, я указываю на это, даже если это связано с неприятностями:
3
да;
нечто среднее;
нет.

rad
114. Иногда я говорю глупости ради шутки, чтобы удивить людей и посмотреть, что они на это скажут:
3
да;
не уверен;

нет.

rad
115. Мне бы понравилось быть газетным критиком в разделе драмы, театра, концертов:
3
да;
не уверен;
нет.

rad
116. У меня никогда не бывает потребности что-нибудь рисовать или вертеть в руках, ерзать на месте, когда приходиться долго сидеть на собрании:
3
верно;
не уверен;
неверно.

rad
117. Если кто-нибудь говорит мне что-нибудь неправильное, то я скорее подумаю:
3
"он - лжец";
"не уверен";
"по-видимому, он плохо информирован.

rad
118. Я чувствую, что мне угрожает какое-то наказание, даже когда я ничего плохого не сделал:
3
часто;
иногда;
никогда.

rad
119. Мнение о том, что болезнь так же часто бывает от психических , как и от физических факторов, сильно преувеличено:
3
да;
не уверен;
нет.

rad
120. Торжественность и величие традиционных церемоний следует сохранить:
3
да;
не уверен;
нет.

rad
121. Мысль о том, что люди подумают, будто я веду себя необычно или странно, мея беспокоит:
3
очень;
немного;
совсем не беспокоит.

rad

122. Выполняя какое-либо дело, я бы предпочел работать:
3
в составе коллектива;
не уверен;
самостоятельно.

rad
123. У меня бывают такие периоды, когда мне трудно избавиться от чувства жалости к себе:
3
часто;
иногда;
никогда.

rad
124. Часто я слишком быстро начинаю сердиться на людей:
3
да;
нечто среднее;
нет.

rad
125. Я всегда могу без труда свои привычки и не возвращаться к прежнему:
3
да;
не уверен;
нет.

rad
126. Если бы зарплата была одинакова, то я предпочел бы быть:
3
адвокатом;
не уверен;
пилотом или капитаном судна.

rad
127. "Лучшее" так относиться к "наихудшее", как "медленное" к :
3
"быстрое";
"лучшее";
"быстрейшее".

rad
128. Каким из приведенных ниже сочетаний следует продолжить буквенный ряд:
        POOOOPPOOOPPP::
3
OPPP;
OOPP;
POOO.

rad
129.Когда приходит время осуществить то , что я планировал и на что надеялся, я обнаруживаю, что уже пропало желание делать это:
3
верно;
нечто среднее;

неверно.

rad
130.Большей частью я могу продолжать работать тщательно, не обращая внимания на шум, создаваемый другим:
3
да;
нечто среднее;
нет.

rad
131. иногда я говорю посторонним вещи, кажущиеся мне важными, независимо от того, спрашивают ли они об этом
3
да;
нечто среднее;
нет.

rad
132. много свободного времени я провожу в разговорах с друзьями о прошлых развлечениях, от которых я получал удовольствие:
3
да;
нечто среднее;
нет.

rad
133. Мне нравиться устраивать какие-нибудь смелые рискованные выходки смеха ради:
3
да;
не уверен;
нет.

rad
134. Вид неубранной комнаты раздражает меня:
3
да;
нечто среднее;
нет.

rad
135. Я считаю себя общительным, открытым человеком:
3
да;
нечто среднее;
нет.

rad
136. В общении я:
3
свободно проявляю свои чувства;
нечто среднее;
держу свои переживания про себя.

rad
137. Я люблю .музыку:
3

легкую, живую:
нечто среднее;
чувсвительную.

rad
138. Красота поэмы восхищает меня больше, чем красота хорошо сделанного оружия:
3
да;
не уверен;
нет.

rad
139. Если мое удачное замечание остается незамеченным окружающими, то я:
3
смиряюсь с этим;
нечто среднее;
даю людям возможность услышать его еще раз.

rad
140. Мне бы понравилось работать фотокорреспондентом:
3
да;
не уверен;
нет.

rad
141. Нужно быть осторожным в общении с незнакомыми, так можно, например, заразиться:
3
да;
не уверен;
нет.

rad
142. При поездке за границу я бы предпочел быть под руководством экскурсовода, чем самому планировать маршрут:
3
да;
не уверен;
нет.

rad
143. Меня справедливо считают упорным и трудолюбивым, но не слишком преуспевающим человеком:
3
да;
не уверен;
нет.

rad
144. Если люди пользуются моим хорошим отношением в своих интересах, то я не возмущаюсь этим и вскоре забываю:
3
верно;
нечто среднее;
неверно.

rad
145. Если при обсуждении какого-либо вопроса среди участников возникает ожесточенный спор, то я предпочитаю:
3
увидеть, кто же победил;
нечто среднее;
чтобы спор разрешился мирно.

rad
146. Я предпочитаю планировать что-либо самостоятельно, без вмешательства и предложений со стороны других:
3
да;
нечто среднее;
нет.

rad
147. Иногда чувство зависти влияет на мои действия:
3
да;
не уверен;
нет.

rad
148. Я твердо верю, что начальник может быть не всегда прав, но он всегда имеет право быть начальником:
3
да;
не уверен;
нет.

rad
149. Когда я думаю обо всем, что еще предстоит сделать, у меня появляется чувство напряженности:
3
да;
иногда;
нет.

rad
150. Когда зрители мне что-либо кричат во время игры, меня это не трогает:
3
верно;
нечто среднее;
неверно.

rad
151. Интереснее быть:
3
художником;
не уверен;
организатором культурных развлечений.

rad
152. Которое из следующих слов не относиться к двум другим:
3

любые;
некоторые;
большинство.

rad
153. "Пламя" так относиться к "жар", как "роза" относиться к:
3
"шип";
"красивые лепестки";
"аромат".

rad
154. У меня бывают яркие сновидения, мешающие мне спать:
3
часто;
иногда;
практически никогда.

rad
155. Если по пути к успеху стоят серьезные препятствия, я все-таки предпочетаю рискнуть:
3
да;
нечто среднее;
нет.

rad
156. Когда я нахожусь в группе людей, приступающих к какой-то работе, то само собой получается, что я нахожусь во главе их:
3
да;
нечто среднее;
нет.

rad
157. Мне больше нравиться в душе спокойная корректность, чем бросающаяся в глаза индивидуальность:
3
верно;
нечто среднее;
неверно.

rad
158. Мне больше нравиться провести вечер за спокойным любимым занятием, чем в оживленной компании:
3
верно;
нечто среднее;
неверно.

rad
159. Я не обращаю внимание на доброжелательные советы других, даже когда эти советы могли бы быть полезными:
3
иногда;
почти никогда;
никогда.

rad
160. В своих поступках я всегда стараюсь придерживаться общепринятых правил поведения:
3
да;
нечто среднее;
нет.

rad
161. Мне не очень нравиться, когда смотрят, как я работаю:
3
да;
нечто среднее;
нет.

rad
162. Иногда приходиться применять силу, потому что не всегда возможно добиться результата с помощью утверждения:
3
верно;
нечто среднее;
неверно.

rad
163. В школе я предпочитал (предпочитаю):
3
русский язык и литературу;
не уверен;
математику или арифметику.

rad
164. Меня иногда огорчало, что обо мне за глаза отзывались неодобрительно без всяких к тому причин:
3
да;
нечто среднее;
нет.

rad
165. разговор с простыми людьми, которые всегда придерживаются общепринятых правил и традиций:
3
часто вполне интересен и содержателен;
нечто средне;
раздражает меня тем, что ограничивается мелочами.

rad
166. Некоторые вещи настолько раздражают меня, что я предпочитаю больше не говорить на эти темы:
3
да;
нечто среднее;
нет.

rad
167. В воспитании важнее:
3
относиться к ребенку с достаточной любовью;

нечто среднее;
выработать нужные привычки и отношение к жизни.

rad
168. Люди считают меня положительным, спокойным человеком, которого не трогают превратности судьбы:
3
да;
нечто среднее;
нет.

rad
169. Я считаю, что общество должно руководиться разумом и отбросить старые привычки и ненужные традиции:
3
да;
не уверен;
нет.

rad
170. Думаю, что в современном мире важнее разрешить:
3
вопросы нравственности;
не уверен;
разногласия между странами мира.

rad
171. Я лучше усваиваю материал:
3
читая хорошо написанную книгу;
нечто среднее;
учавствуя в обсуждении вопроса.

rad
172. Я предпочитаю идти своим путем, вместо того, чтобы действовать в соответствии с принятыми правилами:
3
верно;
не уверен;
неверно.

rad
173. Прежде, чем выдвигать какой-либо аргумент, я предпочитаю подождать, пока не буду убежден, что я прав:
3
всегда;
обычно;
только если это целесообразно.

rad
174 . Мелочи иногда невыносимо действуют мне на нервы, хотя я и понимаю, что они неосуществимы:
3
да;
нечто среднее;
нет.

rad

175. Под влиянием момента я нередко говорю вещи, о которых потом очень сожалею:
3
верно;
не уверен;
неверно.

rad
176. Если бы меня попросили участвовать в шефской деятельности, то я бы:
3
согласился;
не уверен;
вежливо сказал, что занят.

rad
177. Которое из следующих слов относиться к двум другим:
3
"широкий";
"зигзагообразный";
"прямой".

rad
178. "Скоро" так относиться к "никогда", как "близко" к:
3
"нигде";
"далеко";
"где-то".

rad
179. Если я невольно нарушил правила поведения, находясь в обществе, то я вскоре забываю об этом:
3
да;
нечто среднее;
нет.

rad
180. Меня считают человеком, которому обычно в голову приходят хорошие идеи, когда нужно разрешить какую-либо проблему:
3
да;
не уверен;
нет.

rad
181. Я способен лучше проявить себя::
3
в трудных ситуациях, когда нужно сохранить самообладание;
не уверен;
когда требуется умение ладить с людьми.

rad
182. Меня считают человеком полным энтузиазма :
3
да;
нечто среднее;
нет.

rad
183. Мне нравиться работа, которая требует перемен, разнообразия, командировок, даже если она связана с некоторой опасностью:
3
да;
нечто среднее;
нет.

rad
184. Я довольно требовательный человек и всегда настаиваю на том , чтобы все делалось по возможности правильно:
3
верно;
нечто среднее;
неверно.

rad
185. Мне нравиться работа, требующая добросовестного отношения, точных навыков и умения:
3
да;
нечто среднее;
нет.

rad
186. Я отношусь к типу энергичных людей, которые всегда заняты:
3
да;
не уверен;
нет.

rad
187. Я уверен в том, что не пропустил ни одного вопроса и на все ответил как следует:
3
да;
не уверен;
нет.

Задания

7. Добавьте в программу счетчика посещений возможность отслеживания различных страниц. Для этого можно использовать уникальный для каждой страницы файл.
8. После нажатия кнопки «Войти в чат» введенный пароль виден в адресной строке – это хороший способ просмотреть чужой пароль. Избавьтесь от этого, изменив метод GET на метод POST.
9. Добавьте в чат возможности выхода и регистрации под другим именем без загрузки нового окна браузера.
10. Добавьте в программу регистрации в чате возможность игнорирования непустого, но состоящего из пробельных символов имени пользователя.
11. Создайте Web-интерфейс с функциями просмотра списка файлов на сервере и загрузки файла с него. Для типов файлов, поддерживаемых браузером, разрешите отображение его средствами.
12. Модифицируйте тестовую систему так, чтобы все вопросы выводились на одной странице.
13. Создайте программу, реализующую функциональность гостевой книги.
14. Расширьте программу гостевой книги до простого форума.