# О РЕАЛИЗАЦИИ ПРАКТИКУМА ПО ПРОГРАММИРОВАНИЮ ЛЕКСИЧЕСКИХ И СИНТАКСИЧЕСКИХ АНАЛИЗАТОРОВ ПРИ СОЗДАНИИ ЯЗЫКОВЫХ ИНТЕРПРЕТАТОРОВ

А.П. Полищук, С.А. Семериков
г. Кривой Рог, Криворожский государственный педагогический университет
cc@kpi.dp.ua

Требования к квалификации учителя информатики средней школы (что должен знать и уметь) Министерством образования и науки до настоящего времени не определены. Более того, в номенклатуре специальностей педагогических университетов специальность «Информатика» отсутствует, информатика проходит как некоторое дополнение к предметной области в виде «Учитель физики (химии, математики, трудового обучения и т.п.) и основ информатики». При этом под «Основами информатики» часто понимается не система питающих научное направление корней (методов и средств решения основных кибернетических задач), а первичная компьютерная грамотность типа «мышкой по кнопочкам».

В то же время реальные, выдвигаемые практикой, требования к учителю информатики достаточно высоки – он, помимо основной работы по преподаванию курса «Алгоритмизация и программирование», является в школе единственным проводником для внедрения компьютерных технологий в учебный процесс по многим школьным дисциплинам, должен хорошо ориентироваться в системном администрировании и методах защиты программного обеспечения, в приложениях, которые к собственно информатике отношения не имеют (текстовые, графические, музыкальные редакторы и пр.).

Это связано, прежде всего, с созданием и использованием предметно-ориентированных обучающих и тестирующих программ. При работе с такими программными изделиями рано или поздно приходится сталкиваться с необходимостью анализа (лексического, синтаксического, семантического) текстов на формализованных языках и их интерпретации. Этими текстами мо-

гут быть математические формулы или ответы на тестирующие вопросы (не столь простые, как выбор одного из предложенных вариантов ответа), или программы на алгоритмическом языке.

Нелегкие для восприятия курсы системного программирования, связанные с изучением формальных грамматик и их использования в задачах синтаксического анализа, всегда присутствуют в вузовской подготовке специалистов по информатике и обязательно должны быть, по нашему мнению, в арсенале школьного учителя, получающего право преподавания информатики.

Классический курс синтаксического анализа включает в себя:

– теории языков и формальных грамматик (классификация и способы определения языков, грамматики и метаязыки для их описания, генерация языков на основе грамматик);

– постановку и методы решения задач лексического анализа (классификация лексем и их выделение из входного потока);

– постановку и методы решения задач синтаксического анализа (правильность следования лексем и методы вычисления выражений);

– методы создания и использования программ-генераторов программных лексических и синтаксических анализаторов;

– программирование интерпретаторов и компиляторов алгоритмических языков программирования.

Полная реализация этого курса в практикуме по программированию невозможна даже в классических университетах на профильных специальностях, не говоря уж о слишком коротком (и непрерывно укорачиваемом) прокрустовом ложе учебных планов по информатике в педагогических университетах (в лучшем случае – 36 часов на курс синтаксичесого анализа и интерпретации).

Понимая важность и практичекскую полезность курса, мы начали его реализацию еще при изучении дисциплины «Алгоритмизация и программирование»: разбор текстов – прекрасная тренировка для развития алгоритмического мышления и полезнее наборов задач типа «вычислить сумму произведений нечетных элементов четных строк матрицы на четные элементы нечетных столбцов». Первая задача программирования простейшего лексера, поставленная перед студентами, формулировалась так:

составить функцию, осуществляющую ввод и распознавание десятичных, восьмеричных, шестнадцатеричных целых и вещественных чисел. За один вызов функции должно возвращаться одно числовое значение или признак того, что введенные данные не являются числом. Для контроля правильности распознавания распечатывалось как распознанное значение, так и результат его преобразования в числовую форму. Для выполнения работы студентам предоставлялась достаточно подробная методическая помощь, о характере которой можно судить по следующему фрагменту, касающемуся определения системы счисления:

Анализ следующего за знаком считанного символа позволяет предположить тип лексемы:

```
если в ch содержится не '0', то, вероятно, это целое или
вещественное число
|  установим тип лексемы в DEC
|  если ch - не цифра, то
|  | установим тип лексемы в NON
|    -
иначе
|  если в ch содержится '0', то проверяем, число 8-ричное или
|  16-ричное
|  |  добавляем ch в массив parse_result и увеличиваем
|  |  parse_length на 1
|  |  считываем в ch следующий символ
|  |  если в ch содержится 'x' или 'X', то число 16-ричное
|  |  |  установим тип лексемы в HEX
|  |  |  добавляем ch в массив parse_result и увеличиваем
|  |  |  parse_length на 1
|  |  |  считываем в ch следующий символ
|  |  иначе
|  |  |  если в ch содержится '.', то число вещественное и
|  |  |   начинается с 0
|  |  |  |  установим тип лексемы в DBL
|  |  |  |  добавляем ch в массив parse_result и увеличиваем
|  |  |  |  parse_length на 1
|  |  |  |  считываем в ch следующий символ
|  |  |  иначе
|  |  |  |  установим тип лексемы в OCT
|  |  |    -
|  |    -
|  иначе
|  |  если обнаружен признак конца ввода
|  |  |  завершаем работу функции, возвращая -1
|  |    -
|    -
```

-

Дальнейшие действия выполняются в цикле до тех пор, пока лексема не будет распознана полностью либо пока не произойдет ошибка распознавания.

Схематически их можно изобразить следующим образом:

```
 for(;!parse_error && !end;)
  {    switch(mode) //выбор типа лексемы
    { case DEC: //десятичное целое - действия
        break;
      case OCT: //восьмеричное целое - действия
        break;
      case HEX: //шестнадцатеричное целое- действия
        break;
      case DBL: //вещественное число - действия
        break;
      case NON: //не число
        break;
    }
  }
```

По окончании цикла нам останется только проверить, была ли при распознавании ошибка, и не связана ли она с концом ввода. Если это так, признак ошибки можно сбросить установкой значения parse_error в 0.

Доведенная до детализации алгоритма методическая помощь предоставлялась и по каждому распознающему действию. Всего студенты выполняли 3-4 подобных заданий и на следующем этапе ставилась задача программирования интерпретатора функций одной переменной, заданных в параметрическом виде с графической иллюстрацией в заданном диапазоне изменения параметра.

На этом этапе осваивается алгоритм синтаксического разбора выражений, которые могут строиться из следующих элементов: числа, операторы `+ - / * ^ % = () <> ;` , переменные, именованные константы, математические функции из набора, поддерживаемого стандартной математической библиотекой C или C++. Все лексемы классифицируются и переводятся во внутреннее числовое представление:

```
//Коды классов лексем
enum  LexemClass {OPERAND=1,OPERATION,BRACKET,EOL};
//Подклассы (коды) операндов
enum OperandCodes{STRING=1,  //Строка
                  NUMBER,    //Число
                  VARIABLE,  //Переменная
                  CONSTANT,  // Именованная константа
```

```
                  EXPRESS};  // Выражение в скобках
//Коды операций
enum OperationCodes
{  MUL=1,DIV,POW,PLUS,MINUS,ABS,ACOS,ASIN,ATAN, COS,
   COSH, EXP,LOG,LOG10,SIN,SINH,SQRT,TAN,TANH};
//Массив структур для именованных констант
struct cnst {
   char cn[6];//Имя константы
   double cv; //Значение константы
}tc[2]={{"PI",M_PI},{"E",M_E}};
//Все обозначения (имена) операций и их коды сведем в таблицу
//(массив структур по шаблону)
struct {
  char on[10];//Имя операции
  int ov;     //Числовой код операции
}op[]={
      {"ABS",ABS}, {"ACOS",ACOS}, {"ASIN",ASIN}, {"ATAN",ATAN},
      {"COS",COS}, {"COSH",COSH}, {"EXP",EXP}, {"LN",LOG},
      {"LOG",LOG10}, {"SIN",SIN}, {"SINH",SINH}, {"SQRT",SQRT},
      {"TAN",TAN}, {"TANH",TANH}, {"+",PLUS}, {"-",MINUS},
      {"*",MUL}, {"/",DIV}, {"^",POW}, {"",0}};
char *OP="+-*/^"; //Перечень арифметических операций
// Коды ошибок ввода пользователя
enum InputErors {
  SyntaxError, UnpairedParentheses, NotExpression, NotVariable,
  InvalidOperation, InvalidConstName, DivisionByZero
};
```

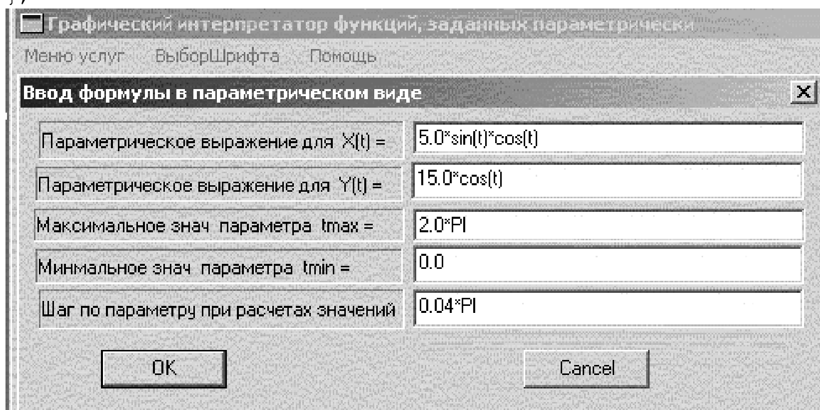


Рис. 1. Вид диалогового окна для ввода интерпретируемых формул

На этом этапе основная трудность состояла в освоении студентами одного из методов синтаксического анализа – рекурсив-

ного спуска или последовательной свертки выражений в соответствии с приоритетами операций. Метод рекурсивного спуска давался трудно и для экономии времени на этом этапе мы ограничились интуитивно легко воспринимаемым методом свертки выражений, используя рекурсию только для раскрытия вложенных скобочных выражений. Материал подавался одновременно с методами программирования интерфейса пользователя, построения графиков функций и других сопутствующих разделов программирования (рис. 1, 2).

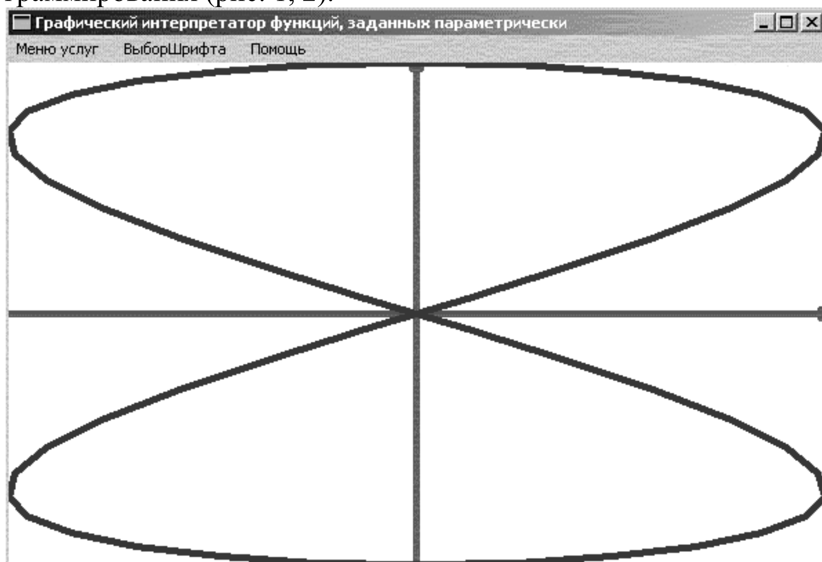


Рис. 2. Результат интерпретации в виде графика функции

После такой предварительной подготовки следующий этап, посвященный синтаксическому анализу формализованных алгоритмических яэыков программирования, реализовывался в составе 36-часового практикума по программированию в составе спецкурса «Синтаксический анализ и интерпретация языков программирования» и включал в себя разработку интерпретатора упрощенного варианта языка С без использования метаязыка для описания его грамматики в форме Бэкуса–Наура. Выбор С в качестве объекта разработки и тестирования объясняется тем, что программирование интерпретатора осуществлялось на этом языке и выбор любого другого языка для интерпретации привел

бы к потерям и без того скудного времени на его освоение. Кроме того, интерпретация того же языка, на котором реализован интерпретатор, позволяет глубже понять его свойства и особенности.

В качестве основы был взят двухпроходный интерпретатор Г. Шилдта [3]. В первом проходе составлялись таблицы глобальных переменных и функций тестируемого текста программы; второй проход начинался с первого оператора главной функции и завершался ее последним оператором с временными переходами в вызываемые вспомогательные подпрограммы любого уровня вложенности. Здесь уже использовался метод рекурсивного спуска в его классическом варианте. В составе программы – три основные функции: лексический и синтаксический анализаторы и собственно интерпретатор, выполняющий действия, предусмотренные блоком операторов, ограниченных парой фигурных скобок с рекурсивными вызовами по уровням вложенности. Эта низкоуровневая фаза во многом перекликается с пройденным ранее программированием интерпретатора формул и поэтому воспринималась относительно неплохо, позволяя в этот период осуществить начитку лекций по формальным грамматикам и генераторам лексических (на примере FLEX, совместимого с широко известным генератором LEX) и синтаксических (на примере BISON, совместимого с YACC) анализаторов.

После этого была поставлена задача выполнить описание грамматики языка в БНФ для LEX, составить операторы распознавания и сопутствующих действий, выполнить генерацию кода лексического анализатора и заменить собственный вариант лексического анализатора анализатором, сгенерированный с помощью LEX. Описание грамматики иллюстрирует следующий фрагмет:

```
LETTER    [A-Z_a-z]
NOLETTER [^A-Z_a-z]
DIGIT    [0-9]
IDENT    {LETTER}({LETTER}|{DIGIT})*
NUMBER   {DIGIT}+
WHISP1   [ \t\r]
WHISP    ({WHISP1}|\n)
.....
```

Секция распознавания и действий иллюстрируется фрагментом программы для ключевых слов, идентификаторов, чисел,

операций, комментариев и пр.:

```
%%
<S>"int" { strcpy(token,yytext);tok = INT;
fprintf(yyout,"%s\n",token);  return token_type=KEYWORD; }
<S>"char{ strcpy(token,yytext);tok = CHAR;
fprintf(yyout,"%s\n",token); return token_type=KEYWORD; }
<S>"while" { strcpy(token,yytext);tok = WHILE;
fprintf(yyout,"%s\n",token);  return token_type=KEYWORD; }
. . . .
<S>{IDENT} { strcpy(token,yytext);tok =
STRING;fprintf(yyout,"%s\n",token); return token_type =
IDENTIFIER; }
<S>{NUMBER} { strcpy(token,yytext);tok=0;
fprintf(yyout,"%s\n",token); return token_type=NUMBER; }
<S>"+"  { strcpy(token,yytext);tok=PLUS;
fprintf(yyout,"%s\n",token); return (token_type = ARITHOP ); }
<S>"-"{ strcpy(token,yytext);tok = MINUS;
fprintf(yyout,"%s\n",token); return(token_type=ARITHOP ); }
<S>"*"{strcpy(token,yytext); tok = MUL;
fprintf(yyout,"%s\n",token); return( token_type=ARITHOP ); }
. . . .
<S>"("{strcpy(token,yytext);tok = 0;
fprintf(yyout,"%s\n",token); return(token_type=DELIMITER ); }
<S>")" { strcpy(token,yytext);tok = 0;
fprintf(yyout,"%s\n",token);  return(token_type=DELIMITER );  }
. . . .
<S>"/*"         { BEGIN COM; }
<COM>[^*]*      { }
<COM>"*"/[^/]   { }
<COM>"*/"       { BEGIN S; }
<S>{WHISP}+     { }
%%
```

При использовании сгенерированного лексера в предварительном проходе по тексту пользовательской программы составлялись таблицы глобальных переменных и функций, а также массив записей со сведениями о лексемах. Записанные в дисковый файл для облегчения отладки, эти таблицы для одного из тестирующих вариантов пользовательских программ имеют вид:

Таблица функций с именем, кодом возвращаемого типа и строкой местонахождения в тексте:

```
FuncName -         main RetType -   2 Loc -  10
FuncName -          sum RetType -   2 Loc - 111
FuncName -   print_alpha RetType -   2 Loc - 147
```

Таблица глобальных переменных (имя, код типа и значение):

```
VarName -            i VarType -   2 Value -   0
```

```
VarName -             j VarType -   2 Value -   0
VarName -            ch VarType -   1 Value -   0
```

Фрагмент таблицы лексем (индекс, обозначение, тип, подтип):

```
Index -     0 Name -        int Type -   3 Tok -   2
Index -     1 Name -          i Type -   1 Tok -   5
Index -     2 Name -          , Type -   0 Tok -   0
Index -     3 Name -          j Type -   1 Tok -   5
Index -     4 Name -          ; Type -   0 Tok -   0
Index -     5 Name -       char Type -   3 Tok -   1
Index -     6 Name -         ch Type -   1 Tok -   5
Index -     7 Name -          ; Type -   0 Tok -   0
Index -     8 Name -        int Type -   3 Tok -   2
Index -     9 Name -       main Type -   1 Tok -   5
Index -    10 Name -          ( Type -   0 Tok -   0
Index -    11 Name -          ) Type -   0 Tok -   0
Index -    12 Name -          { Type -   7 Tok -   0
Index -    13 Name -        int Type -   3 Tok -   2
. . . .
```

Наличие таких таблиц максимально упрощает не только получение очередной лексемы простым наращиванием индекса массива, но и получение любой предыдущей или следующей лексемы при последующем синтаксическом анализе.

Несмотря на описанную выше предварительную подготовку, два следующих этапа (использование генератора синаксических анализаторов BISON и ознакомление с методами программирования генераторов кода) реализованы не были – отведенного учебным планом времени для этого оказалось недостаточно.

**Выводы**

Обобщая опыт реализации описанного спецкурса и ряда других курсов по информатике, направленных на повышение уровня подготовки специалистов в этой области, хотелось бы отметить следующее. Информатика (по существу синоним термина «кибернетика») – очень непростая для освоения наука, которая не должна изучаться как «попутный» с другой наукой предмет, если ставится задача подготовки квалифицированного специалиста в этой области. Учитывая очевидную тенденцию ко все более раннему началу изучения иностранных языков и компьютерных дисциплин в общеобразовательной средней школе (начиная с младших классов) и острый дефицит квалифицированных преподавателей в этой области как в школе, так и в вузах, назрела необходимость внесения в номенклатуру учительских

специальностей для педагогических университетов отдельной квалификации «Учитель информатики» с фундаментальной подготовкой по математичесим методам решения основных кибернетических задач (теории оптимального управления, методам идентификации управляемых систем, исследования операций), анализу алгоритмов и компьютерному программированию и т.д.

При сохранении существующего положения с квалификацией преподавателей информатики удлинение сроков обучения в школе будет иметь тот же плачевный результат, что и многолетнее изучение иностранных (а часто и родных) языков в школе и затем в вузе, когда для перевода студентом простейших англоязычных сообщений компилятора типа «Invalid parameter» приходится предоставлять программу-переводчик.

Литература

1. Ахо А.В., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты. – М.: Издательский дом “Вильямс”, 2001.

2. Кнут Д. Искусство программирования. Т.1. Основные алгоритмы. – 3 е изд. – М.: Издательский дом “Вильямс”, 2000.

3. Шилдт Г. Полный справочник по С. – 4-е издание. – М.: Издательский дом “Вильямс”, 2000.

4. Хантер Р. Основные концепции компиляторов. – М.: Издательский дом “Вильямс”, 2002.