Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

А.Н. Стась

Современные проблемы информатики и вычислительной техники

Методические указания по практической и самостоятельной работе для магистрантов направлений 09.04.01 «Информатика и вычислительная техника» и 09.04.02 «Информационные системы и технологии»

Томск
2022

УДК 004.9
ББК 32.973.22
С 77

Рецензенты:
Боровской И.Г., профессор каф. ЭМИС

Стась, Андрей Николаевич
С 77 Современные проблемы информатики и вычислительной техники: метод. указания / А.Н. Стась. – Томск: Томск.гос. ун-т систем упр. и радиоэлектроники, 2022. – 33 с.

Методические указания по выполнению практической и самостоятельной работе для магистрантов направлений 09.04.01 «Информатика и вычислительная техника», 09.04.02 «Информационные системы и технологии».

Одобрено на заседании каф. ЭМИС протокол № 1 от 30.08.2022 г.

УДК 004.9
ББК 32.973.22

Оглавление

Введение

Цель изучения дисциплины: знакомство студентов с современными проблемами информатики, особенностями научной деятельности в данной отрасли знаний.

Задачи дисциплины:

1) знакомство студентов с различными направлениями современных научных исследований в области информатики и вычислительной техники, их результатами и перспективами;

2) развитие у студентов умения изучения и прогнозирования результатов развития научных направлений в области информатики и вычислительной техники.

В результате освоения дисциплины обучающийся должен:

– знать содержание организации и руководства деятельностью рабочего коллектива (группы) разработчиков программного обеспечения, методы командного взаимодействия и стратегического планирования для достижения поставленных целей;

– демонстрировать умение организации работы группы разработчиков программного обеспечения, эффективного руководства данной группой;

– владеть основными методами и приемами командной разработки, методами организации работы группы разработчиков программного обеспечения, приемами социального взаимодействия.

Практическая работа 1. Реализация простейшего алгоритма лексического анализатора

Цель работы: запрограммировать алгоритм лексического анализа на основе конечного автомата.

Теоретический материал

Пример простейшего языка

Все переменные в рассматриваемом языке имеют целочисленный тип, нет описания типов. Константы также целочисленны. Запись идентификатора начинается с символа латинского алфавита, далее латинские символы или цифры. Поддерживаются одномерные массивы и подпрограммы, имеющие один параметр и результирующие значение.

разделители операторов `;`.

Присваивание `:=`.

Операции бинарные: +, −, *, / (div), \ (mod), <, >, <=, >=, !=, = (рез – 0,1).

`A[i]` – доступ к элементу массива, индексация (начальный индекс – 0).

Комментарии: `|*.....*|`.

Команды:

`INPUT` <перем> – ввод;

`OUTPUT` <выр> – вывод;

`IF` <выр> `THEN` <оператор 1> [`ELSE` <оператор 2>];

`WHILE` <выр> `DO` <оператор>;

`{`составной оператор`}` (в паскале `BEGIN…END`);

`FUNC`<назв>`(`<параметр>`)` – начало функции;

`RET` возвр. Знач. – возврат из функции;

<назв. Функ.>`(`<парам>`)` – вызов функции внутри выражения;

`HALT` – прекращение выполнения программы.

Лексика языка описывается с помощью следующей таблицы переходов конечного автомата.

	a..z, A..Z	0..9	;	п	+,-,/,\,	(	)	[	]	:	=	>,<	!	|	*	{	}
S	A	B	S	S	S	S	S	S	S	C	S	D	E	F	S	S	S
A	A	A	S	I	S	S (ф-ция)	S	S	S	C	S	D	E	F	S	S	S
B	Ош	B	S	S	S	Ош	S	Ош	S	Ош	S	D	E	F	S	Ош	S
C	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	S (:=)	Ош	Ош	F	Ош	Ош	Ош
D	A (<,>)	B (<,>)	Ош	S (<, >)	Ош	Ош	Ош	Ош	Ош	Ош	S (<=, >=)	Ош	Ош	F	Ош	Ош	Ош
E	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	S (!=)	Ош	Ош	F	Ош	Ош	Ош
F	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	Ош	G	Ош	Ош
G	G	G	G	G	G	G	G	G	G	G	G	G	G	G	H	G	G
H	G	G	G	G	G	G	G	G	G	G	G	G	G	Кон ком	G	G	G
I	A (нов)	B (нов)	S	I	S	S (ф-ция)	S	S	S	C	S	D	E	F	S	S	S

Понятия, соответствующие нетерминальным символам в грамматике:
S – выделение лексемы
A – выделение идентификатора или ключевого слова
B – выделение числовой константы
C – Возможно присваивание
D – простые или составные операции сравнения <,>,<=,>=
E – возможно операция “!=”
F – возможно комментарий
G – комментарий
H – возможен конец комментария
I – возможен вызов функции

Таблица ключевых слов:

```
K1 – INPUT
K2 – OUTPUT
K3 – IF
K4 – THEN
K5 – ELSE
K6 – WHILE
K7 – DO
K8 – FUNC
K9 – RET
K10 – HALT
```

1-е внутреннее представление, генерируемое лексическим анализатором, имеет свои особенности. Так, ключевые слова, имена переменных, константы заменяются ссылками на элементы соответствующих таблиц, а сложные операнды заменяются однозначными кодовыми символами.

Например: `:= – @, <= – ~, >= $, != – ^`

Также необходимо сохранять значения констант для данной программы, например, в конце описания после спецсимвола `$`. При реализации обычно проще выгрузить таблицу констант в отдельный файл.

Приведем 2 примера соответствия исходного текста программы и текста в первом внутреннем представлении.

Пример 1. Вычисление суммы от `1` до `N`.

```
INPUT n;
s:=0;
i:=1;
WHILE i<=n DO {
 s:=s+i;
 i:=i+1
};
OUTPUT s
HALT;
```

```
K1I1;
I2@C1;
I3@C2;
K6I3~I1K7{
I2@I2+I3;
I3@I3+C3
};
K2I2;
K10;
$C1=0$C2=1$C3=1
```

Пример 2. Вычисление факториала.

```
INPUT n;
OUTPUT fact(n);
HALT;
FUNC fact(k);
IF k=1 THEN RET 1 ELSE RET
k*fact(k-1);
```

```
K1I1;
K2F1(I1);
K10;
K8F1(I2);
K3I2=C1K4K9C2K5
K9I2*F1(I2-C3);
$C1=1$C2=1$C3=1
```

В приложении 1 приведен листинг программы, реализующей лексический анализ на языке Паскаль.

Требования к результатам выполнения работы:

- изучить алгоритм лексического анализа;
- реализовать алгоритм лексического анализа на языке C, C++ или Visual Basic.

Практическая работа 2. Реализация простейшего алгоритма синтаксического анализа

Цель работы:

Запрограммировать алгоритм синтаксического анализа на основе рекурсивного спуска.

Теоретический материал

Описание синтаксиса в виде КС-грамматике.

Приведем описание синтаксиса языка программирования на уровне 1-го внутреннего представления.

S-> A;S| A| л A -> B| C B -> {S} C -> D | E | F |ёG | H| I| J| K D -> L@M E -> k1L F -> k2M G ->k3Mk4A|k3Mk4Ak5A H -> k6Mk7A I -> k8N(O) J -> k9M K -> k10 L -> O| O[M] M -> P | (M)| MRM N -> fT O -> iT P -> L| U| V R -> +| -| *| /| \\| <| >| ~| $| =| ^ T -> 0T| 1T |…..| 9T| 0| 1|…| 9 U -> cT V -> N(M)	S – программа (последовательность операторов) A – оператор B – составной оператор C – простой оператор D – присваивание E – ввод F – вывод G – условный оператор H – цикл с предусловием I – заголовок функции J – возврат из функции K – останов L – переменная M – выражение N – имя функции O – имя переменной P – операнд R – операция T – номер (константы, переменной, ключевого слова или функции) U – константа V – вызов функции

Замечание. Используем малые буквы для обозначения ключевых слов, констант, имен переменных и функций в схематическом описании, т.к. они являются терминальными символами.

Упрощенное описание синтаксиса языка после оптимизации грамматики

S -> A;S| A| л A -> {S}| L@M| k1L| k2M| k3Mk4A| k3Mk4Ak5A| k6Mk7A| k8N(O)| k9M| k10 L -> O| O[M] M -> L | U | V| (M)| MRM N -> fT O -> iT R -> +| -| *| /| \\| <| >| ~| $ =| ^ T -> oT| 1T|….| 9T| 0| 1|…| 9 U -> cT V -> N (M)	S -> A;S| A| л A -> {S}| L@M| k1L| k2M| k3Mk4A| k3Mk4Ak5A| k6Mk7A| k8N(iT)| k9M| k10 L -> iT| iT[M] M -> iT | iT[M]| cT| N(M)| (M)| MRM N -> fT R -> +| -| *| /| \\| <| >| ~| $ =| ^ T -> oT| 1T|….| 9T| 0| 1|…| 9

Метод рекурсивного спуска основан на моделировании LL(k) – анализа за счет использования рекурсивных алгоритмов.

Идея метода

Для каждого нетерминала создается функция, моделирующая анализ, начиная с данного нетерминала. В случае если при раскрытии нетерминала встречается другой нетерминал происходит вызов соответствующей функции. Процесс продолжается до исчерпания входной цепочки (программы) или обнаружения ошибки. Необходимым условием реализации идеи является отсутствие левой рекурсии в грамматике (в этом случае анализ будет циклить). Анализ начинается с нетерминала S.

В нашем случае имеется леворекурсивное правило

```
M->MRM
```

После избавления от левой рекурсии грамматика приобретает следующий вид:

```
S -> A;S| A| л
A -> {S}| L@M| k1L| k2M| k3Mk4A| k3Mk4Ak5A| k6Mk7A| k8N(iT)| k9M| k10
L -> iT| iT[M]
M -> LZ|cTZ| N(M)Z| (M)Z| MZ
N -> fT
R -> +| -| *| /| \| <| >| ~| $ =| ^
T -> oT| 1T|….| 9T| 0| 1|…| 9
Z->RM|л
```

Именно в соответствии с этой грамматикой строится анализатор.

Второе внутреннее представление основано на использовании расширенной обратной польской записи. К множеству арифметических операций и операций сравнения добавлено операции ввода (OI), вывода (OO), прерывания (OH), безусловного перехода (OJ), условного перехода (OZ), вызова подпрограммы (O%), загрузки параметров (OG), возврата из подпрограммы (OR).

Для большинства контрукций языка, которые описываются однозначными грамматическими контрукциями, второе внутреннее представление генерируется непосредственно в процессе синтаксического анализа. Исключением являются выражения, грамматика описания которой неоднозначна (фактическую однозначность этим конструкциям придает налчие приоритетов операций). Поэтому, второе внутреннее представление для выражений формируется с помощью алгоритма Дейкстры, предназначенного для первода выражений из инфиксной формы в постфиксную.

Приведем примеры соответствия вторых внутренних представлений, генерируемых синтаксическим анализатором первому внутреннему представлению и исходному тексту программы.

Пример 1. Вычисление суммы

INPUT n;
s:=0;
i:=1;
WHILE i<=n DO
{
s:=s+i;
i:=i+1
};
OUTPUT s
HALT;
	K1I1;
	I2@C1;
	I3@C2;
	K6I3~I1K7{
	I2@I2+I3;
	I3@I3+C3
	};
	K2I2;
	K10;
	$C1=0$C2=1$C3=1
		I1OI
		I2C1O@
		I3C2O@
		I3I1O~L2OZ (метка L1 – перед началом)
		I2I2I3O+O@
		I3I3C3O+O@
		L1OJ
		I2OO (метка L2 – перед началом)
		OH

Пример 2. Вычисление факториала

INPUT n;
OUTPUT
fact(n);
HALT;
FUNC fact(k);
IF k=1 THEN
RET 1 ELSE RET
k*fact(k-1);
	K1I1;
	K2F1(I1);
	K10;
	K8F1(I2);
	K3I2=C1K4K9C2
	K5K9I2*F1(I2-C3);
	$C1=1$C2=1$C3=1
		I1OIk
		F1I1O%OO
		OH
		I2OG
		I2C1O=L1OZC2ORL2OJ
		I2F1I2C3O-O%O*OR (Метка L1 – перед началом)
		(метка L2 – в конце)

Листинг программы на языке «Паскаль», реализующей алгоритм синтаксического анализатора приведен в приложении 2.

Требования к результатам выполнения работы:

- изучить алгоритм синтаксического анализа;
- реализовать алгоритм синтаксического анализа на языке C, C++ или Visual.

Практическая работа 3. Реализация простого алгоритма интерпретации обратной польской записи.

Цель работы: запрограммировать алгоритм интерпретации второго внутреннего представления на основе обратной польской записи.

Теоретический материал

Алгоритм интерпретации арифметических выражений в постфиксной форме достаточно прост. Строка анализируется слева направо. Если текущий символ – операнд, то он помещается в стек, если текущий символ – операция, то из стека извлекается количество операндов в соответствие с арностью операции, после чего операция выполняется и результат помещается в стек. По окончанию анализа в стеке будет результат выполнения выражения.

Для адаптации данного алгоритма к интерпретации расширенной обратной польской записи необходимо работать с операндами не как с константами, а как с адресами, по которым в памяти хранятся определенные значения.

Листинг программы на языке «Паскаль», реализующий данный алгоритм приведен в приложении 3.

Требования к результатам выполнения работы:

- изучить алгоритм синтаксического анализа;
- реализовать алгоритм синтаксического анализа на языке C, C++ или Visual.

Указания к самостоятельной работе студентов (СРС)

Виды самостоятельной работы:

1. Общий анализ современных проблем в информатике и вычислительной техники. Подготовка к зачету с оценкой.
2. Общий анализ современных проблем в информатике и вычислительной техники. Подготовка к тестированию.
3. Основные тенденции в области эффективного использования ресурсов в IT-отрасли. Подготовка к зачету с оценкой.
4. Основные тенденции в области эффективного использования ресурсов в IT-отрасли. Подготовка к зачету с оценкой. Подготовка к тестированию.
5. Тенденции развития технического обеспечения автоматизированных систем. Подготовка к зачету с оценкой.
6. Тенденции развития технического обеспечения автоматизированных систем. Подготовка к тестированию.
7. Тенденции развития технического обеспечения автоматизированных систем. Подготовка к выступлению (докладу) 15 УК-3 Выступление (доклад) на занятии.

ПРИЛОЖЕНИЕ 1. Реализация лексического анализа

```
program la;
const
 kwCount=10;
 kwTable:array[1..kwCount] of string[7]=('INPUT','OUTPUT','IF','THEN',
          'ELSE','WHILE','DO','FUNC','RET','HALT');
var
 idCount, consCount, funCount: integer;
 curState, curTerm, prevTerm, prevComState: char;
 idTable: array[1..200] of string[15];
 consTable: array[1..200] of integer;
 funTable: array[1..30] of string[15];
 inFile, outFile, consTableFile: Text;
 str, curLex: string[50];
 k,i: integer;

function kwIndex(lex: string): integer;
var
 i: integer;
begin
 kwIndex:=0;
 for i:=1 to kwCount do if lex=kwTable[i] then begin
  kwIndex:=i;
  break
 end
end;

function idIndex(lex: string): integer;
var
 i: integer;
begin
 idIndex:=0;
 for i:=1 to idCount do if lex=idTable[i] then begin
  idIndex:=i;
  break
 end
end;

function funIndex(lex: string): integer;
var
 i: integer;
begin
 funIndex:=0;
 for i:=1 to funCount do if lex=funTable[i] then begin
  funIndex:=i;
  break
 end
end;

begin
 writeln('input source file name:');
 readln(str);
 assign(inFile, str+'.src');
 reset(inFile);
 assign(outFile,str+'.ala');
 rewrite(outFile);
 curState:='S';
 curLex:='';
 idCount:=0;
 consCount:=0;
```

```
while not eof(inFile) do begin
 read(inFile, curTerm);
 if (curTerm<>chr(10))and(curTerm<>chr(13)) then case curState of
  'S': begin
   case curTerm of
    'a'..'z', 'A'..'Z': begin
     curState:='A';
     curLex:=curLex+curTerm
    end;
    '0'..'9': begin
     curState:='B';
     curLex:=curLex+curTerm
    end;
    ' ': begin
     curState:='S';
    end;
    ';','+','-','/','\','(',')','[',']','=','*','{','}': begin
     curState:='S';
     write(outFile, curTerm);
    end;
    ':': begin
     curState:='C';
    end;
    '<','>': begin
     curState:='D';
     prevTerm:=curTerm;
    end;
    '!': begin
     curState:='E';
    end;
    '|': begin
     prevComState:=curState;
     curState:='F';
    end;
    else begin
     writeln('error');
     halt
    end
   end
  end;
  'A': begin
   case curTerm of
    'a'..'z', 'A'..'Z','0'..'9': begin
     curState:='A';
     curLex:=curLex+curTerm
    end;
    ' ': begin
     curState:='I';
    end;
    ';','+','-','/','\',')','[',']','=','*','{','}': begin
     curState:='S';
     k:=kwIndex(curLex);
     if k<>0 then write(outFile,'K',k) else begin
      k:=idIndex(curLex);
      if k=0 then begin
       inc(idCount);
       idTable[idCount]:=curLex;
       write(outFile,'I', idCount)
      end else write(outFile,'I',k)
     end;
     write(outFile, curTerm);
     curLex:='';
```

```
  end;
  '(': begin
   curState:='S';
   k:=funIndex(curLex);
   if k<>0 then write(outFile,'F',k) else begin
    inc(funCount);
    funTable[funCount]:=curLex;
    write(outFile,'F',funCount);
   end;
   write(outFile, curTerm);
   curLex:='';
  end;
  ':': begin
   curState:='C';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
  end;
  '<','>': begin
   curState:='D';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k)  else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
   prevTerm:=curTerm;
  end;
  '!': begin
   curState:='E';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
  end;
  '|': begin
   prevComState:=curState;
   curState:='F';
  end;
  else begin
   write('error');
   halt
  end;
 end;
end;
```

```
'B': begin
 case curTerm of
  '0'..'9': begin
   curState:='B';
   curLex:=curLex+curTerm
  end;
  ' ': begin
   curState:='S';
   inc(consCount);
   val(curLex,k,i);
   consTable[consCount]:=k;
   write(outFile, 'C', consCount);
   curLex:='';
  end;
  ';','+','-','/','\',')',']','=','*','}': begin
   curState:='S';
   inc(consCount);
   val(curLex,k,i);
   consTable[consCount]:=k;
   write(outFile, 'C', consCount);
   write(outFile, curTerm);
   curLex:='';
  end;
  '<','>': begin
   curState:='D';
   inc(consCount);
   val(curLex,k,i);
   consTable[consCount]:=k;
   write(outFile, 'C', consCount);
   curLex:='';
   prevTerm:=curTerm;
  end;
  '!': begin
   curState:='E';
   inc(consCount);
   val(curLex,k,i);
   consTable[consCount]:=k;
   write(outFile, 'C', consCount);
   curLex:='';
  end;
  '|': begin
   prevComState:=curState;
   curState:='F';
  end;
  else begin
   writeln('error');
   halt
  end
 end
end;
'C': begin
 case curTerm of
  '=': begin
   curState:='S';
   write(outFile, '@');
   curLex:='';
  end;
  '|': begin
   prevComState:=curState;
   curState:='F';
  end;
  else begin
```

```
   writeln('error');
   halt
  end
 end
end;
'D': begin
 case curTerm of
  'a'..'z','A'..'Z': begin
   curState:='A';
   write(outFile, prevTerm);
   curLex:=curLex+curTerm
  end;
  '0'..'9': begin
   curState:='B';
   write(outFile, prevTerm);
   curLex:=curLex+curTerm
  end;
  '=': begin
   curState:='S';
   if prevTerm='<' then write(outFile,'~') else write(outFile,'$');
  end;
  '|': begin
   prevComState:=curState;
   curState:='F';
  end;
  else begin
   writeln('error');
   halt
  end
 end
end;
'E': begin
 case curTerm of
  '=': begin
   curState:='S';
   write(outFile,'^');
  end;
  '|': begin
   prevComState:=curState;
   curState:='F';
  end;
  else begin
   writeln('error');
   halt
  end
 end
end;
'F': begin
 case curTerm of
  '*': begin
   curState:='G';
  end;
  else begin
   writeln('error');
   halt
  end
 end
end;
'G': begin
 case curTerm of
  '*': begin
   curState:='H';
```

```
  end;
  else begin
   curState:='G';
  end
 end
end;
'H': begin
 case curTerm of
  '|': begin
   curState:=prevComState;
  end;
  else begin
   writeln('error');
   halt
  end
 end
end;
'I': begin
 case curTerm of
  'a'..'z','A'..'Z': begin
   curState:='A';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
   curLex:=curLex+curTerm;
  end;
  '0'..'9': begin
   curState:='B';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
   curLex:=curLex+curTerm;
  end;
  ' ': begin
   curState:='I';
  end;
  ';','+','-','/','\',')','[',']','=','*','{','}': begin
   curState:='S';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k)  else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
```

```
   write(outFile, curTerm)
  end;
  '(': begin
   k:=funIndex(curLex);
   if k<>0 then write(outFile,'F',k) else begin
    inc(funCount);
    funTable[funCount]:=curLex;
    write(outFile,'F',funCount);
   end;
   curLex:='';
   write(outFile, curTerm);
  end;
  ':': begin
   curState:='C';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
  end;
  '<','>': begin
   curState:='D';
   prevTerm:=curTerm;
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
  end;
  '!': begin
   curState:='E';
   k:=kwIndex(curLex);
   if k<>0 then write(outFile,'K',k) else begin
    k:=idIndex(curLex);
    if k=0 then begin
     inc(idCount);
     idTable[idCount]:=curLex;
     write(outFile,'I', idCount)
    end else write(outFile,'I',k)
   end;
   curLex:='';
  end;
  '|': begin
   prevComState:=curState;
   curState:='F';
  end;
  else begin
   writeln('error');
   halt
  end
 end
end;
```

```
  end
 end;
 assign(consTableFile, str+'.cta');
 rewrite(consTableFile);
 writeln(consTableFile, consCount);
 for i:=1 to consCount do writeln(consTableFile,consTable[i]);
 close(consTableFile);
 close(inFile);
 close(outFile)
end.
```

ПРИЛОЖЕНИЕ 2. Реализация синтаксического анализа

```
program sa;

TYPE
 rec=object
  procedure SAnal;
  procedure AAnal;
  procedure LAnal(isFirst: boolean);
  procedure MAnal(isFirst: boolean);
  procedure NAnal;
  procedure RAnal;
  procedure TAnal;
  procedure ZAnal;
  function prior(oper: char): byte;
  procedure exprAnal(s: string);
 end;


var
 str: array[1..4000] of char;
 inFile, outFile: text;
 FuncFile, LabFile: file of integer;
 fname: string;
 len, cur, offs: integer;
 res: boolean;
 myRec: rec;
 labelCount, funcNum, funcCount, i: byte;
 lastNum: string;
 Lab, funcLab: array[1..255] of integer;


function rec.prior(oper: char): byte;
begin
 case oper of
  '(': prior:=0;
  '<','>','~','$','=','^': prior:=1;
  '+','-': prior:=2;
  '*','/','\': prior:=3;
  '%','[': prior:=4
 end
end;


procedure rec.exprAnal(s: string);
var
 stack: array[1..200] of char;
 digits, operat: SET of char;
 elemType, elemValue: char;
 sN: string;
 sp, pr: integer;
 cr, sNI: byte;
begin
 sp:=0;
 cr:=1;
 digits:=['0','1','2','3','4','5','6','7','8','9'];
 operat:=['+','-
','*','/','\','<','>','~','$','=','^','(',')','[',']','%'];
 while cr<=length(s) do begin
  elemType:=s[cr];
  if elemType IN operat then begin
```

```
  elemValue:=elemType;
  elemType:='O'
 end;
 if elemType<>'O' then begin
  inc(cr);
  sN:='';
  while (s[cr] IN digits)and(cr<=length(s)) do begin
   sN:=sN+s[cr];
   inc(cr)
  end;
  val(sN, sNI, pr);
  elemValue:=chr(sNI);
  write(outFile, elemType, elemValue);
  offs:=offs+2;
  if elemType='F' then begin
   inc(sp);
   stack[sp]:='%';
  end
 end else begin
  if (elemValue='(')or(elemValue='[') then begin
   inc(sp);
   stack[sp]:=elemValue;
   inc(cr);
  end;
  if elemValue in (operat-['(',')','[',']'])  then begin
   while (sp>0)and(prior(elemValue)<=prior(stack[sp])) do begin
    write(outFile, 'O', stack[sp]);
    offs:=offs+2;
    dec(sp)
   end;
   inc(sp);
   stack[sp]:=elemValue;
   inc(cr)
  end;
  if elemValue=')' then begin
   while (sp>0)and(stack[sp]<>'(') do begin
    write(outFile, 'O', stack[sp]);
    offs:=offs+2;
    dec(sp)
   end;
   if (sp>0)and(stack[sp]='(') then dec(sp);
   inc(cr)
  end;
  if elemValue=']' then begin
   while (sp>0)and(stack[sp]<>'[') do begin
    write(outFile, 'O', stack[sp]);
    offs:=offs+2;
    dec(sp)
   end;
   if (sp>0)and(stack[sp]='[') then begin
    dec(sp);
    write(outFile, 'O', 'E');
    offs:=offs+2
   end;
   inc(cr)
  end
 end;
end;
if cr>length(s) then while sp>0 do begin
 write(outFile,'O', stack[sp]);
 offs:=offs+2;
 dec(sp)
```

```
 end;
end;

procedure rec.SAnal;
begin
 if (cur<=len)and(str[cur]<>'}') then begin
  AAnal;
  if (cur<=len)and(str[cur]=';') then begin
   inc(cur);
   SAnal
  end
 end
end;

procedure rec.AAnal;
var
 lab1, lab2, lN: byte;
 offs1, offs2, pr: integer;
begin
 if cur<=len then begin
  case str[cur] of
   '{': begin
    inc(cur);
    SAnal;
    if (cur>len)or(str[cur]<>'}') then begin
     res:=false;
     writeln('error');
     halt
    end else inc(cur)
   end;
   'I': begin
    LAnal(true);
    if (cur>len)or(str[cur]<>'@') then begin
     res:=false;
     writeln('error');
     halt
    end else inc(cur);
    MAnal(true);
    write(outFile,'O','@');
    offs:=offs+2
   end;
   'K': begin
    if (cur+2)>len then begin
     res:=false;
     writeln('error');
     halt
    end else case str[cur+1] of
     '1': begin
      case str[cur+2] of
       'I': begin
        cur:=cur+2;
        LAnal(true);
        write(outFile,'O','I');
        offs:=offs+2
       end;
       '0': begin
        cur:=cur+3;
        write(outFile,'O','H');
        offs:=offs+2
       end;
       else begin
        res:=false;
```

```
   writeln('error');
   halt
  end;
 end;
end;
'2': begin
 cur:=cur+2;
 MAnal(true);
 write(outFile,'O','O');
 offs:=offs+2
end;
'3': begin
 cur:=cur+2;
 MAnal(true);
 if (cur>=len)or(str[cur]<>'K')or(str[cur+1]<>'4') then begin
  res:=false;
  writeln('error');
  halt
 end else cur:=cur+2;
 inc(labelCount);
 write(outFile,'L', chr(labelCount),'O','Z');
 lab1:=labelCount;
 offs:=offs+4;
 AAnal;
 Lab[lab1]:=offs;
 if (cur<len)and(str[cur]='K')and(str[cur+1]='5') then begin
  inc(labelCount);
  lab2:=labelCount;
  write(outFile,'L',chr(labelCount),'O','J');
  offs:=offs+4;
  Lab[lab1]:=offs;
  cur:=cur+2;
  AAnal;
  Lab[lab2]:=offs
 end;
end;
'6': begin
 cur:=cur+2;
 offs1:=offs;
 MAnal(true);
 inc(labelCount);
 write(outFile,'L',chr(labelCount),'O','Z');
 offs:=offs+4;
 lab2:=labelCount;
 if (cur>=len)or(str[cur]<>'K')or(str[cur+1]<>'7') then begin
  res:=false;
  writeln('error');
  halt
 end;
 cur:=cur+2;
 AAnal;
 inc(labelCount);
 write(outFile,'L',chr(labelCount),'O','J');
 offs:=offs+4;
 Lab[labelCount]:=offs1;
 Lab[lab2]:=offs;
end;
'8': begin
 cur:=cur+2;
 offs1:=offs;
 NAnal;
 funcLab[funcNum]:=offs1;
```

```
      if funcNum>funcCount then funcCount:=funcNum;
      if (cur>=len)or(str[cur]<>'(')or(str[cur+1]<>'I') then begin
       res:=false;
       writeln('error');
       halt
      end;
      cur:=cur+2;
      write(outFile,'I');
      inc(offs);
      lastNum:='';
      TAnal;
      val(lastNum, lN, pr);
      write(outFile, chr(lN));
      inc(offs);
      if (cur>len)or(str[cur]<>')') then begin
       res:=false;
       writeln('error');
       halt
      end;
      inc(cur);
      write(outFile,'O','G');
      offs:=offs+2
     end;
     '9': begin
      cur:=cur+2;
      MAnal(true);
      write(outFile,'O','R');
      offs:=offs+2;
     end;
    end
   end;
   else begin
    res:=false;
    writeln('error');
    halt
   end
  end;
 end else begin
  res:=false;
  writeln('error');
  halt
 end
end;


procedure rec.LAnal(isFirst: boolean);
var
 idNum: byte;
 pr: integer;
 expr: string;
 expb, explen, i: byte;
begin
 if (cur>len)or(str[cur]<>'I') then begin
  res:=false;
  writeln('error');
  halt
 end else begin
  expb:=cur;
  inc(cur);
  TAnal;
  if (cur<=len)and(str[cur]='[') then begin
   inc(cur);
```

```
   MAnal(false);
   if (cur>len)or(str[cur]<>']') then begin
    res:=false;
    writeln('error');
    halt
   end else begin
    inc(cur);
   end;
  end;
  if isFirst then begin
   explen:=cur-expb;
   expr:='';
   for i:=1 to explen do expr:=expr+str[i+expb-1];
   exprAnal(expr)
  end
 end
end;

procedure rec.MAnal(isFirst: boolean);
var
 expr: string;
 expb, explen, i: byte;
begin
 if cur<=len then begin
  expb:=cur;
  case str[cur] of
   'I': begin
    LAnal(false);
    ZAnal
   end;
   'C': begin
    inc(cur);
    TAnal;
    Zanal
   end;
   'F': begin
    NAnal;
    if (cur>len)or(str[cur]<>'(') then begin
     res:=false;
     writeln('error');
     halt
    end else inc(cur);
    MAnal(false);
    if (cur>len)or(str[cur]<>')') then begin
     res:=false;
     writeln('error');
     halt
    end else inc(cur);
    ZAnal
   end;
   '(': begin
    inc(cur);
    MAnal(false);
    if (cur>len)or(str[cur]<>')') then begin
     res:=false;
     writeln('error');
     halt
    end else inc(cur);
    ZAnal
   end;
   else begin
    res:=false;
```

```
    writeln('error');
    halt
   end;
  end;
  if isFirst then begin
   explen:=cur-expb;
   expr:='';
   for i:=1 to explen do expr:=expr+str[i+expb-1];
   exprAnal(expr)
  end
 end else begin
  res:=false;
  writeln('error');
  halt
 end
end;

procedure rec.NAnal;
var
 pr: integer;
begin
 if (cur>len)and(str[cur]<>'F') then begin
  res:=false;
  writeln('error');
  halt
 end;
 inc(cur);
 lastNum:='';
 TAnal;
 val(lastNum,funcNum,pr);
end;

procedure rec.RAnal;
var
 operat: set of char;
begin
 operat:=['+','-','*','/','\','<','>','~','$','=','^'];
 if (cur>len)or(not (str[cur] IN operat)) then begin
  res:=false;
  writeln('error');
  halt
 end else inc(cur)
end;

procedure rec.TAnal;
var
 digits: set of char;
begin
 digits:=['0','1','2','3','4','5','6','7','8','9'];
 if (cur>len)or(not (str[cur] IN digits)) then begin
  res:=false;
  writeln('error');
  halt
 end;
 lastNum:=lastNum+str[cur];
 inc(cur);
 if (cur<len)and(str[cur] IN digits) then TAnal
end;

procedure rec.ZAnal;
var
 operat: set of char;
```

```
begin
 operat:=['+','-','*','/','\','<','>','~','$','=','^'];
 if (cur<=len)and(str[cur] IN operat) then begin
  RAnal;
  MAnal(false)
 end
end;

begin
 writeln('input filename:');
 readln(fname);
 assign(inFile, fname+'.ala');
 assign(outFile,fname+'.asa');
 reset(inFile);
 rewrite(outFile);
 offs:=0;
 len:=0;
 labelCount:=0;
 funcCount:=0;
 while not eof(incFile) do begin
  inc(len);
  read(inFile, str[len]);
 end;
 cur:=1;
 res:=true;
 myRec.SAnal;
 writeln('OK');
 close(inFile);
 close(outFile);
 assign(FuncFile, fname+'.fun');
 assign(LabFile, fname+'.lab');
 rewrite(FuncFile);
 rewrite(LabFile);
 for i:=1 to labelCount do write(LabFile,Lab[i]);
 for i:=1 to funcCount do write(FuncFile,funcLab[i]);
 close(FuncFile);
 close(LabFile)
end.
```

ПРИЛОЖЕНИЕ 3. Реализация интерпретации расширенной ОПЗ

```
program interp;
uses crt;

type
 elRec=record
  elType: char;
  elValue: char;
 end;
 arrayType= packed array[0..49] of integer;
 aPointer=^arrayType;

var
 fileName: string;
 inFile: text;
 inFile2: file of integer;
 elType, elValue, c: char;
 stackKA: array[1..1000] of elRec;
 spKA: integer;
 elNum, identCount, i, j, consCount, labCount, funcCount: byte;
 integerCount, arrayCount: byte;
 promCount: integer;
 isArray: array[1..255] of boolean;
 consTable, labTable, funcTable: array[1..255] of integer;
 identTable: array[1..255] of byte;
 intTable, promTable: array[1..255] of integer;
 arrTable: array[1..255] of aPointer;
 ptr1, ptr2: Pointer;
 stackI: array[1..5000] of Pointer;
 spI: integer;
 iPtr1, iPtr2: ^integer;
 rpn: array[1..5000] of char;
 rpnLength, curElem: integer;
 stackC: array[1..5000] of integer;
 spC: integer;

begin
 writeln('Input file name:');
 readln(fileName);
 assign(inFile, fileName+'.asa');
 reset(inFile);
 spKA:=0;
 identCount:=0;
 for i:=1 to 255 do isArray[i]:=false;
 while not eof(inFile) do begin
  read(inFile, elType, elValue);
  case elType of
   'I', 'C', 'L', 'F': begin
    inc(spKA);
    stackKA[spKA].elType:=elType;
    stackKA[spKA].elValue:=elValue;
    if elType='I' then begin
     elNum:=ord(elValue);
     if elNum>identCount then identCount:=elNum
    end
   end;
   '0': begin
    case elValue of
     '+', '-', '*', '/', '\', '<', '>', '=', '^', '~', '$', 'E' :
begin
```

```
      if (elValue='E')and(stackKA[spKA-1].elType='I') then
isArray[ord(stackKA[spKA-1].elValue)]:=true;
      dec(spKA);
      stackKA[spKA].elType:='P';
     end;
     '@', 'Z', '%': spKA:=spKA-2;
     'R': begin
      stackKA[spKA].elType:='P';
     end;
     'I', 'O', 'J', 'G': dec(spKA);
    end
   end
  end
 end;
 close(inFile);
 integerCount:=0;
 arrayCount:=0;
 for i:=1 to identCount do begin
  if isArray[i] then begin
   inc(arrayCount);
   identTable[i]:=arrayCount;
  end else begin
   inc(integerCount);
   identTable[i]:=integerCount;
  end
 end;
 assign(inFile,fileName+'.cta');
 reset(inFile);
 read(inFile,consCount);
 for i:=1 to consCount do read(inFile, consTable[i]);
 close(inFile);
 assign(inFile2,fileName+'.lab');
 reset(inFile2);
 labCount:=0;
 while not eof(inFile2) do begin
  inc(labCount);
  read(inFile2, labTable[labCount])
 end;
 close(inFile2);
 assign(inFile2,fileName+'.fun');
 reset(inFile2);
 funcCount:=0;
 while not eof(inFile2) do begin
  inc(funcCount);
  read(inFile2, funcTable[funcCount])
 end;
 close(inFile2);
 assign(inFile,fileName+'.asa');
 reset(inFile);
 spI:=0;
 spC:=0;
 for i:=1 to arrayCount do begin
  new(arrTable[i]);
  for j:=0 to 255 do arrTable[i]^[j]:=0;
 end;
 writeln('program executing:');
 rpnLength:=0;
 while not eof(inFile) do begin
  read(inFile, rpn[rpnLength+1], rpn[rpnLength+2]);
  rpnLength:=rpnLength+2;
 end;
 curElem:=0;
```

```
 while curElem<rpnLength do begin
  elType:=rpn[curElem+1];
  elValue:=rpn[curElem+2];
  case elType of
   'I': begin
    if isArray[ord(elValue)] then
ptr1:=arrTable[identTable[ord(elValue)]]
    else ptr1:=@(intTable[identTable[ord(elValue)]]);
    inc(spI);
    stackI[spI]:=ptr1;
   end;
   'C': begin
    ptr1:=@(consTable[ord(elValue)]);
    inc(spI);
    stackI[spI]:=ptr1;
   end;
   'L': begin
    ptr1:=@(labTable[ord(elValue)]);
    inc(spI);
    stackI[spI]:=ptr1;
   end;
   'F': begin
    ptr1:=@(funcTable[ord(elValue)]);
    inc(spI);
    stackI[spI]:=ptr1;
   end;
   'O': begin
    case elValue of
     '+', '-', '*', '/', '\', '<', '>', '~', '$', '=', '^': begin
      iPtr1:=stackI[spI-1];
      iPtr2:=stackI[spI];
      dec(spI);
      inc(promCount);
      if elValue='+' then promTable[promCount]:=iPtr1^+iPtr2^;
      if elValue='-' then promTable[promCount]:=iPtr1^-iPtr2^;
      if elValue='*' then promTable[promCount]:=iPtr1^*iPtr2^;
      if elValue='/' then promTable[promCount]:=iPtr1^ div iPtr2^;
      if elValue='\' then promTable[promCount]:=iPtr1^ mod iPtr2^;
      if elValue='<' then begin
       if iPtr1^<iPtr2^ then promTable[promCount]:=1 else
promTable[promCount]:=0;
      end;
      if elValue='>' then begin
       if iPtr1^>iPtr2^ then promTable[promCount]:=1 else
promTable[promCount]:=0;
      end;
      if elValue='~' then begin
       if iPtr1^<=iPtr2^ then promTable[promCount]:=1 else
promTable[promCount]:=0;
      end;
      if elValue='$' then begin
       if iPtr1^>=iPtr2^ then promTable[promCount]:=1 else
promTable[promCount]:=0;
      end;
      if elValue='=' then begin
       if iPtr1^=iPtr2^ then promTable[promCount]:=1 else
promTable[promCount]:=0;
      end;
      if elValue='^' then begin
       if iPtr1^<>iPtr2^ then promTable[promCount]:=1 else
promTable[promCount]:=0;
      end;
```

```
 stackI[spI]:=@(promTable[promCount])
end;
'@': begin
 iPtr1:=stackI[spI-1];
 iPtr2:=stackI[spI];
 spI:=spI-2;
 iPtr1^:=iPtr2^;
end;
'H': halt;
'I': begin
 iPtr1:=stackI[spI];
 dec(spI);
 readln(iPtr1^);
end;
'O': begin
 iPtr1:=stackI[spI];
 dec(spI);
 writeln(iPtr1^);
end;
'J': begin
 iPtr1:=stackI[spI];
 dec(spI);
 curElem:=iPtr1^-2
end;
'Z': begin
 iPtr1:=stackI[spI-1];
 iPtr2:=stackI[spI];
 spI:=spI-2;
 if iPtr1^=0 then curElem:=iPtr2^-2
end;
'E': begin
 ptr1:=stackI[spI-1];
 iPtr1:=stackI[spI];
 dec(spI);
 stackI[spI]:=Pointer(Integer(ptr1)+2*iPtr1^)
end;
'%': begin
 iPtr1:=stackI[spI-1];
 iPtr2:=stackI[spI];
 spI:=spI-2;
 inc(spC);
 stackC[spC]:=curElem+2;
 for i:=1 to integerCount do begin
  inc(spC);
  stackC[spC]:=intTable[i]
 end;
 for i:=1 to arrayCount do for j:=0 to 49 do begin
  inc(spC);
  stackC[spC]:=arrTable[i]^[j]
 end;
 inc(spC);
 stackC[spC]:=iPtr2^;
 curElem:=iPtr1^-2
end;
'G': begin
 iPtr1:=stackI[spI];
 dec(spI);
 iPtr1^:=stackC[spC];
 dec(spC)
end;
'R': begin
 for i:=arrayCount downto 1 do for j:=49 downto 0 do begin
```

```
            arrTable[i]^[j]:=stackC[spC];
            dec(spC)
           end;
           for i:=integerCount downto 1 do begin
            intTable[i]:=stackC[spC];
            dec(spC)
           end;
           curElem:=stackC[spC]-2;
           dec(spc)
          end
         end
        end
       end;
       curElem:=curElem+2
      end;
      for i:=1 to arrayCount do dispose(arrTable[i]);
      close(inFile);
      c:=readkey
end.
```