Оглавление

Введение

Цикл лабораторных работ призван закрепить теоретические знания по дисциплине «Теория языков программирования и методы трансляции» и способствовать выработке у студентов профессиональных компетенций в области разработки сложного и комплексного программного обеспечения, каким является транслятор (компилятор).

Цикл лабораторных работ построен таким образом, что охватывает все этапы проектирования трансляторов (лексический и синтаксический анализ, двухэтапную генерацию машинного кода). Поэтому при выполнении всего цикла работ в результате будет построен компилятор, а поскольку студент получает индивидуальный вариант задания, то у него появляется уникальная возможность проявить элементы инженерного творчества как в выборе программного инструментария, так и в самой технологии разработки.

1 Лабораторная работа «Построение лексического анализатора»

1.1 Основные понятия лексического анализа

Лексический анализ всегда предшествует синтаксическому анализу и заключается в предварительной обработке программы и ее перекодировании к виду, удобному для синтаксического анализа. Лексический анализ принято отделять от синтаксического анализа для сокращения времени компиляции, поскольку синтаксис (морфология) слов всегда проще, чем синтаксис программ.

Лексический анализ сводится к разбиению текста программы на лексические единицы (слова, лексемы) – конструкции, которые выступают в качестве терминальных символов для синтаксического анализа. Лексемы еще иногда называют символами или атомами.

Большинство лексем в языках программирования можно сгруппировать в следующие классы:

- класс идентификаторов;
- класс служебных слов;
- класс констант (числовых или символьных);
- класс операций (одно-, дву- или многолитерных);
- класс разделителей (однолитерных или двулитерных).

В лексическом анализе лексема представляется в виде пары (класс, значение). Значение может быть непосредственным (литеральным) или представлять собой ссылку на некоторую таблицу, в которой хранятся значения лексем.

1.2 Лексемы простого PL-подобного языка программирования

Для каждого из классов можно построить автоматную грамматику, полностью описывающую правила построения его лексем. Например, грамматика, описывающая идентификаторы, может быть построена следующим образом:

<Идентификатор>::= *Буква* | <Идентификатор> *Буква* |

| <Идентификатор> *Цифра*

Для чисел с фиксированной точкой:

<Число>::= <Целое> | <Смешанное число>

<Смешанное число>::= <Точка> ***Цифра*** | <Целое> . |

<Смешанное число> ***Цифра***

<Целое>::= ***Цифра*** | <Целое> ***Цифра***

<Точка>::= .

Для разделителей {‘/’ , ‘//’ , ‘/*’ , ‘,’ , ‘;’}:

<Разделитель>::= / | / <Второй символ> | , | ;

<Второй символ>::= / | *

Как выше указывалось, во внутреннем представлении все лексемы имеют вид пары, в которой первый элемент – это признак принадлежности к классу (буква или цифра), а второй – значение в описанном выше понимании.

Рассмотрим пример построения лексического анализатора для языка, являющегося некоторым подмножеством языка PL/1. Определим его конструкции:

1. *Идентификатор* – любая последовательность букв и цифр, начинающаяся с буквы;
2. *Описание данных* (ограничимся только одним типом числовых данных DEC FIXED, то есть вещественное число с фиксированной точкой) имеет вид:

```
DCL Список переменных DEC FIXED;
```

где *Список переменных* – это идентификатор или перечисление идентификаторов через запятую в круглых скобках.

3. *Описание процедур* имеет вид

```
PROC Имя процедуры;
   Тело процедуры
END;
```

4. *Операторы*:

- присваивания =;
- безусловного перехода имеет вид

  `GOTO` ***`Идентификатор;`***

- условный логический оператор

  `IF` ***`Условие`*** `THEN` ***`Оператор;`***

- оператор конца процедуры END;

5. *Выражения*:

- операции

  сложения +;

  умножения *;

  отношений <,>, <>;

- скобки (,);

6. Константы:

- числовые (целые и вещественные с фиксированной точкой);
- символьные (произвольная последовательность символов, заключенная в апострофы);

7. *Метки* вида

Идентификатор: ;

8. *Разделители*:

пробел, конец строки, « , » « ; » « . »

9. *Комментарии*: произвольная последовательность символов от знака // до конца строки.

Построим таблицу классов лексем для рассматриваемого языка (табл. 1.1).

Таблица 1.1. Классы лексем подмножества языка PL/1.

Типы лексем	Обозначение
Служебные слова	W
Идентификаторы	I
Операции	O
Разделители	R
Константы	N
	C

Для оптимизации и ускорения поиска данных часто таблицу констант разделяют на две: таблицу N (для числовых констант) и таблицу C (для символьных констант).

1.3 Функции и таблицы лексического анализа

Основная функция лексического анализатора (сканера) состоит в выделении лексем из исходной программы и передаче их синтаксическому анализатору в некотором внутреннем представлении. При этом лексический анализатор использует таблицы, соответствующие классам лексем. Таблицы служебных слов, разделителей и операций определяются входным языком и должны быть сформированы при построении сканера, они являются постоянными и неизменяемыми. Таблицы идентификаторов и констант являются временными и создаются непосредственно в процессе лексического разбора исходной программы.

Комментарии в процессе лексического анализа игнорируются, так как они специфицируют исходную программу только для программиста и не влияют на семантику программы.

Для рассматриваемого языка сформированные таблицы служебных слов (табл. 1.2), таблица операций (табл. 1.3) и таблица разделителей (табл. 1.4) имеют вид:

Таблица 1.2

Служебное слово	Код
DCL	2
END	3
FIXED	4
GOTO	5
IF	6
PROC	7
THEN	8
MAIN	9

Таблица 1.3

Операция	Код
+	1
*	2
<	3
>	4
=	5
:	6
<>	7

Таблица 1.4

Разделитель	Код
пробел	1
,	2
;	3
(	4
)	5
.	6
конец строки	7

Каждая лексема, обработанная сканером, преобразуется к виду:

<буква><код>,

где: <буква> – это признак класса лексемы (W, I, O, R, N или C),

<код> – номер лексемы в соответствующей таблице.

Например, если на вход сканера поступила цепочка вида

```
IF C > 2.79 THEN B = A * E3;
```

то на выходе сканера будет сформирована следующая последовательность кодов лексем:

Код лексемы	Класс лексемы	Значение лексемы
W6	Служебное слово	IF
I1	Идентификатор	C
O4	Операция	>
N1	Число	2.79
W8	служебное слово	THEN
I2	Идентификатор	B
O5	Операция	=
I3	Идентификатор	A
O2	Оператор	*
I4	Идентификатор	E3
R3	Разделитель	;

При лексическом разборе входной цепочки пробелы не кодируются как разделители и в выходную цепочку не попадают. Это связано с тем, что в исходном тексте программы пробелы служат только для структурирования текста программы и не несут дополнительной смысловой нагрузки.

Рассмотрим работу сканера на примере исходной программы, представленной ниже:

```
//Вычисление суммы и произведения
PROC MAIN;
    DCL (A1, A2) DEC FIXED;
    A1=378;

    A2=.73;
```

```
        PROC CALC;
             DCL (SUM, MULT) DEC FIXED;
             IF A1+A2<>3.2 THEN GOTO P;
             SUM=(A1+A2)*A2;
        P:   MULT=A1*A2;
        END;
    END.
```

Для данной исходной программы лексический анализатор сформирует следующую выходную последовательность лексем:

Входная последовательность текста исходной программы	Выходная последовательность лексем во внутреннем представлении
//Вычисление суммы и произведения	
PROC MAIN;	W7 W9 R3
DCL (A1, A2) DEC FIXED;	W2 R4 I1 R2 I2 R5 W1 W4 R3
A1=378;	I1 O5 1 R3
A2=.73;	I2 O5 N2 R3
PROC CALC;	W7 I3 R3
DCL SUM, MULT DEC FIXED;	W2 R4 I4 R2 I5 R5 W1 W4 R3
IF A1+A2<>3.2 THEN GOTO P;	W6 I1 O1 I2 O7 N3 W8 W5 I6 R3
SUM=A1+A2;	I4 O5 I2 O1 I1 R3
P: MULT=A1*(A2+SUM);	I6 O6 I5 O5 I1 R4 I2 O1 I4 R5 R3
END;	W3 R3
END.	W3 R6

И, кроме того, лексический анализатор сформирует таблицу чисел N (табл. 1.5) и таблицу идентификаторов I (табл. 1.6).

Таблица 1.5

Код	Число
1	378
2	.73
3	3.2

Таблица 1.6

Код	Идентификатор	Номер процедуры	Уровень процедуры	Номер идентификатора в процедуре	Тип идентификатора	Объем памяти
1	A1					
2	A2					
3	CALC					
4	SUM					
5	MULT					
6	P					

1.4 Диаграмма состояний лексического процессора

Лексический анализатор представляет собой процессор или, иначе говоря, сканер (конечный автомат) для разбора и классификации лексем, связанный с некоторыми семантическими процедурами. На вход такого процессора последовательно подаются символы исходной программы, причем каждый входной символ вызывает изменение состояния сканера.

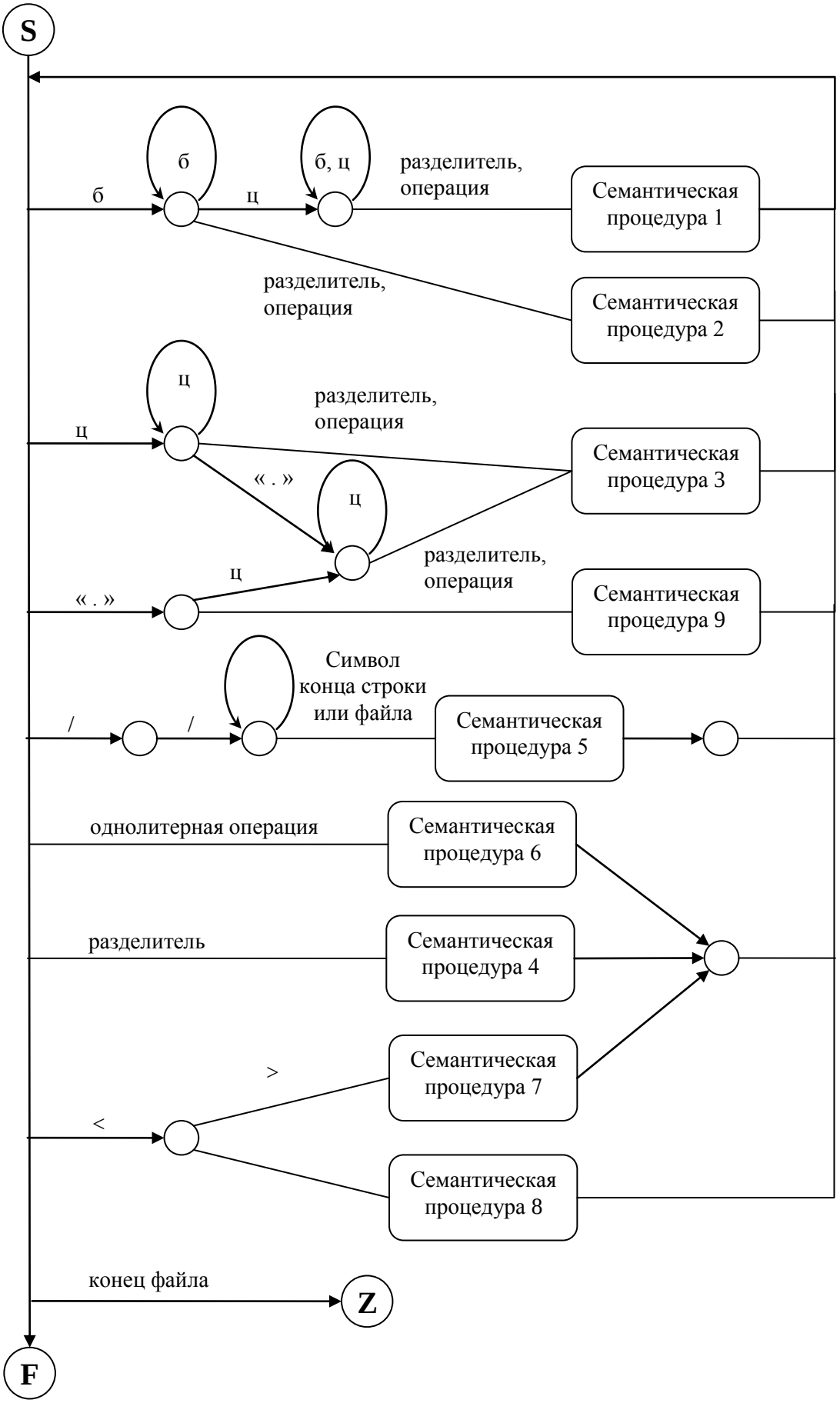
Если анализируемый символ означает конец разбираемой лексемы, то с переходом связывается некоторая семантическая процедура, позволяющая либо определить код лексемы по таблице (в случае служебных слов, разделителей, операций), либо пополнить таблицы (для констант и идентификаторов), а затем в выходную строку выдать очередной код лексемы во внутреннем представлении.

Для эффективного разбора и оптимизации сканера, а также для его программной реализации в алфавите автомата выделяют несколько подмножеств символов, по которым сканер выполняет одинаковые переходы и действия. В нашем случае такими подмножествами являются:

- буквы {A, …, Z};
- цифры {0, …, 9};
- символы однолитерных операций { + , * , > , = , : };
- символы, служащие началом двулитерных операций {<};
- разделители { пробел , конец строки , « , » , « ; » };
- точка (она играет в нашем примере двойную роль: десятичная точка в числе и точка в конце программы).

В некоторых языках программирования точка не имеет роли самостоятельного разделителя.

С учетом вышесказанного диаграмма состояний сканера для рассматриваемого языка имеет вид, представленный на рисунке 1.1.


S
б
б
ц
б, ц
разделитель,
операция
Семантическая
процедура 1
разделитель,
операция
Семантическая
процедура 2
ц
ц
разделитель,
операция
Семантическая
процедура 3
« . »
ц
ц
« . »
разделитель,
операция
Семантическая
процедура 9
Символ
конца строки
или файла
/
/
Семантическая
процедура 5
однолитерная операция
Семантическая
процедура 6
разделитель
Семантическая
процедура 4
>
Семантическая
процедура 7
<
Семантическая
процедура 8
конец файла
Z
F

На данной диаграмме состояния сканера представлены вершинами, а переходы между состояниями - дугами (направленными линиями). Каждый переход связан с чтением очередного символа их текста входной программы. Поэтому дуга взвешена (помечена) символом или множеством символов, которые вызывают данный переход. Если в диаграмме состояний есть невзвешенная дуга, ведущая из какого-либо состояния, то считается, что она взвешена любыми символами кроме тех, которыми взвешены другие дуги, исходящие из данного состояния. Это позволяет не перегружать диаграмму лишними символами. Скругленный прямоугольник в разрыве дуги указывает на семантическую процедуру, выполняемую при данном переходе. Если переход не сопровождается семантической процедурой, то текущий символ добавляется к буферу, в котором формируется лексема. Работа каждой семантической процедуры завершается очищением буфера.

Сканер всегда содержит три стандартных состояния:

S – начальное состояние сканера. Лексический разбор всегда начинается из этого состояния;

Z – заключительное состояние сканера. Если в процессе разбора достигнуто данное состояние, это означает, что разбор успешно завершен;

F – состояние ошибки. Если встретился символ, не входящий во входной алфавит, то сканер переходит в состояние ошибки и прекращает разбор. Кроме того, если в каком-либо состоянии на входе появился символ, по которому не предусмотрен переход из этого состояния, сканер также переходит в состояние ошибки (на диаграмме эти переходы не изображаются, чтобы избежать лишнего загромождения рисунка).

За семантическими процедурами процессора закреплены следующие функции:

Семантическая процедура 1: Провести поиск сформированного слова в таблице идентификаторов. Если такое слово в таблице

идентификаторов не найдено, то занести сформированное слово в таблицу идентификаторов. Сформировать и выдать в выходную последовательность лексему идентификатора во внутреннем представлении.

Семантическая процедура 2: Провести поиск сформированного слова в таблице служебных слов. Если такое слово в таблице служебных слов не найдено, то выполнить Семантическую процедуру 1, иначе сформировать и выдать в выходную последовательность лексему служебного слова во внутреннем представлении.

Семантическая процедура 3: Занести сформированное слово в таблицу констант. Сформировать и выдать в выходную последовательность лексему константы во внутреннем представлении.

Семантическая процедура 4: Провести поиск текущего символа в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя во внутреннем представлении.

Семантическая процедура 5: Удалить сформированную последовательность символов.

Семантическая процедура 6: Провести поиск текущего символа в таблице операций. Сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

Семантическая процедура 7: Добавить текущий символ к сформированному слову. Провести поиск сформированного слова в таблице операций. Если такое слово в таблице операций не найдено, то перейти в состояние ошибки. Если поиск успешен, то сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

Семантическая процедура 8: Провести поиск сформированного слова в таблице операций. Сформировать и выдать в выходную последовательность лексему операции во внутреннем представлении.

Семантическая процедура 9: Провести поиск сформированного слова в таблице разделителей. Сформировать и выдать в выходную последовательность лексему разделителя во внутреннем представлении.

2 Лабораторная работа «Перевод исходной программы в обратную польскую запись»

2.1 Понятие обратной польской записи

Обратная польская запись (ОПЗ) - представляет собой одну из форм записи выражений и операторов, отличительной особенностью которой является расположение аргументов (операндов) перед операцией (оператором).

Например, выражение, записанное в обычной скобочной записи,

```
(a+d)/c+b*(e+d),
```

в ОПЗ имеет следующее представление:

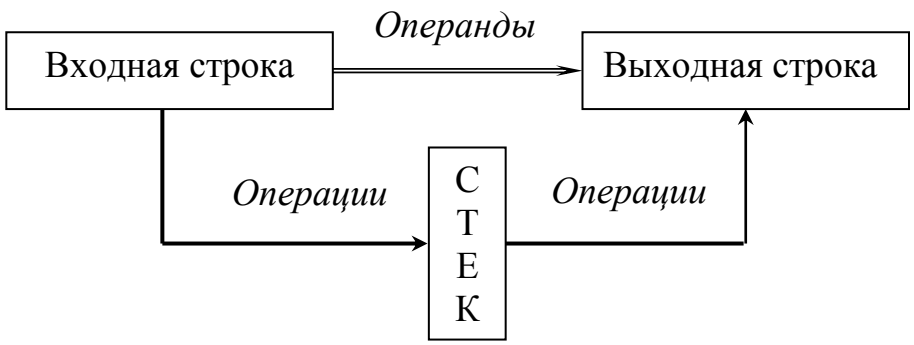
```
ad+c/bed+*+.
```

Обратная польская запись получила широкое распространение благодаря своему основному преимуществу ОПЗ может быть вычислена за один просмотр цепочки слева направо, который часто называют проходом.

2.2 Алгоритм Дейкстры

Исследованию формальных способов преобразования арифметических и логических выражений в ОПЗ посвящены многочисленные исследования, однако в практике системного программирования наибольшее распространение получили способы преобразования на основе алгоритма Дейкстры.

Суть алгоритма Дейкстры можно представить следующим рисунком:



Из этого рисунка следует, что на вход алгоритма посимвольно поступает исходное выражение. Операнды исходного выражения пропускаются на выход и формируют так же посимвольно выходную строку. Операции обрабатываются по определенным правилам на основе стека.

Для реализации такой обработки известное в системном программировании понятие стека используется также в алгоритме Дейкстры для размещения в нем операций. При этом предварительно каждой операции приписывается свой приоритет на основе таблицы приоритетов, которая приведена ниже (табл. 2.1).

Таблица 2.1

Входной элемент	Приоритет
(	0
)	1
V	2
&	3
¬	4
Отношения	5
+ -	6
* /	7
возведение в степень	8

С учетом этой таблицы сформулируем правила работы со стеком.

Если рассматриваемый символ является операцией, то с ним производятся следующие действия путем сравнения его приоритета с приоритетом верхнего символа стека:

- Если стек пуст, операция заносится в стек.

- Если в стеке верхний элемент (операция) имеет более высокий приоритет, то рассматриваемая операция проталкивается в стек.
- Если в стеке верхний элемент (операция) имеет более низкий или равный приоритет, то из стека в выходную строку выталкиваются все элементы (операции) до тех пор, пока не встретится операция с приоритетом выше, чем у рассматриваемой операции, после чего операция из входной строки проталкивается в стек.
- Если входной символ "(", то он всегда проталкивается в стек. Если входной символ ")", то он выталкивает из стека все символы операций до ближайшей "(". Сами скобки взаимно уничтожаются и в выходную строку не попадают.

Приведем пример перевода выражения в ОПЗ:

Выходная строка	a		b	+		5	-	<	2		c	-	1		q	+	=	&
СТЕК		+		<	-		&			-		=		+				
					<					&		&		=				
														&				
Входная строка	a	+	b	<	-	5	&		2	-	c	=	1	+	q	ν		

2.3 Перевод выражений, содержащих переменные с индексами, в ОПЗ

В языках программирования используются массивы и их элементы, то есть переменные с индексами. Сами массивы могут быть как одномерными, так и многомерными. Реально в памяти ЭВМ любые массивы размещаются только в виде одномерных массивов, поэтому при трансляции программ необходимо решать проблему отображения многомерного массива в одномерный и организации доступа к его элементам.

Для решения этой проблемы вводится операция *вычисления адреса элемента массива* (АЭМ).

Операция АЭМ имеет *k*-мерного массива имеет *k +1* операнд:

- имя массива;
- индексы массива,

и ОПЗ представляется следующим образом:

<операнд1><операнд2> ... <операнд k+1> k+1 АЭМ,

где: <операнд1> - имя массива, <операнд2>,..,<операндk+1> - индексы массива, k+1 – счетчик операндов.

Например, ОПЗ выражения

(a+b[i+20,j])*c+d

будет иметь вид

a b i 20 + j 3АЭМ + c * d +

В этой ОПЗ операнды операции АЭМ подчеркнуты.

Для реализации алгоритма Дейкстры с учетом операции АЭМ, ее необходимо ввести в таблицу приоритетов, которая теперь выглядит следующим образом (табл. 2.2).

Таблица 2.2

Входной элемент	Приоритет
([АЭМ	0
,)]	1
V	2
&	3
¬	4
Отношения	5
+ -	6
* /	7
возведение в степень	8

Правила работы со стеком при переводе выражений с индексами в ОПЗ принимают следующий вид:

- обычные знаки операций обрабатываются так же, как и в предыдущем случае;
- с открытием массива (идентификатор и «[») в стек заносится операция АЭМ со значением счетчика операндов равным 2;
- знак «,» выталкивает из стека все символы до ближайшего АЭМ и наращивает счетчик операндов на 1;
- закрывающая скобка «]» выталкивает из стека операцию АЭМ со значением счетчика операндов, но сама в выходную строку не попадает.

Перевод рассмотренного выше выражения в ОПЗ по алгоритму Дейкстры имеет следующий вид:

Выходная строка		a		b		i		20	+	j	3АЭМ	+		c	*	d	+
С Т Е К	(		+		2АЭМ		+		3АЭМ		+		*		+		
			(		+		2АЭМ		+		(						
						(	+		(								
							(										
Входная строка	(	a	+	b	[	i	+	20	,	j	]	)	*	c	+	d	■

Примечание. При трансляции исходной программы в объектный код на машинном языке адреса обращения к элементам массива должны быть правильно оформлены. Для этого необходимо вычислить так называемую функцию упорядочивания, которая устанавливает взаимно однозначное соответствие между многомерным массивом и отображающим его одномерным массивом. Но поскольку в программах может использоваться множество массивов, то нужно еще и выбрать соответствующую данному массиву функцию упорядочивания. Обычно вся эта информация, касающаяся каждого массива, собирается транслятором в определяющий вектор массива, а сами определяющие вектора размещаются во временную таблицу транслятора. Таким образом, обращение к соответствующему элементу многомерного массива связано с обработкой определяющих векторов и вычислением функции упорядочивания, что является семантической процедурой, скрывающимся за символом операции АЭМ.

2.4 Перевод в ОПЗ выражений, содержащих указатели функций

В арифметических и логических выражениях могут использоваться функции.

Введем операцию *функция*, которая так же, как и АЭМ, имеет k операндов и записывается в ОПЗ в виде:

<операнд1><операнд2> ... <операндk> k Ф,

где <операнд1> - имя функции, а <операнд2>,..,<операндk> - операнды функции.

Пример. Обратная польская запись выражения

```
y-f(x,z,y+2)
```

имеет вид

y f x z y 2 + 4 Ф -

Таблица приоритетов для алгоритма Дейкстры модифицируется следующим образом (табл. 2.3).

Таблица 2.3

Входной элемент	Приоритет
([АЭМ Ф	0
,)]	1
V	2
&	3
¬	4
Отношения	5
+ -	6
* /	7
Возведение в степень	8

При обработке оператора "Функция" выполняются следующие действия:

- по входному символу '(', следующему за идентификатором, в стек заносится оператор Ф со значением счетчика, равным 1;
- по входному символу ',' из стека выталкивается все до последнего оператора Ф и значение счетчика наращивается на 1;
- по входному символу ')' из стека выталкивается все до последнего оператора Ф, значение счетчика наращивается на 1 и оператор Ф выталкивается в выходную строку.

Например, для приведенного выражения процесс перевода в ОПЗ имеет вид:

Выходная строка	y		f		x		z		y		2	+	4Ф	-
С Т Е К		-		1Ф -		2Ф -		3Ф -		+ 3Ф -		4Ф -	-	
Входная строка	y	-	f	(	x	,	z	,	y	+	2	)		■

Примечание. При трансляции исходной программы в объектный код на машинном языке функции должны быть оформлены как подпрограммы. Обращение к данным подпрограммам строится на основе обработки операции *функция* в ОПЗ. Поэтому операция *функция* является символом в ОПЗ, а за этим символом скрывается ряд семантических процедур, связанных с правильным обращением к соответствующим подпрограммам. Обычно поиск требуемой подпрограммы организовывается следующим образом: заводится таблица имен функций и в этой таблице указываются адреса, по которым располагаются эти подпрограммы. При распознавании оператора функция в ОПЗ из нее извлекается имя подпрограммы и ее операнды, а далее в объектном коде вставляются стандартные процедуры обращения к подпрограмме.

2.5 Перевод условных выражений в ОПЗ

Пусть имеется условие вида:

$$y = \begin{cases} a + b, & x =< 0 \\ c + d, & x > 0 \end{cases}$$

Это условие в языках программирования записывается как условное выражение вида:

```
y=if x=<0 then a+b else c*d;
```

В отличие от простых выражений, где порядок выполнения действий определяется старшинством операций или скобками и в процессе выполнения программы этот порядок не меняется, в условных выражениях он изменяется в зависимости от значения условия (в нашем примере x <= 0).

Таким образом, понятие условного выражения должно быть также введено в ОПЗ представления исходной программы. Для этого вводятся так называемые динамические деревья.

Рассмотрим ряд понятий и особенностей для динамических деревьев:

- узлы и ветви деревьев могут помечаться метками;
- пустой узел - это такой узел, в котором отсутствует операция. Из пустого узла может исходить любое количество ветвей. Пустой узел вводится для объединения нескольких последовательных действий или разветвления некоторого вычислительного процесса;
- условный оператор перехода по значению 'ложь'. Оператор имеет следующее представление:

<условие> <метка> УПЛ

- оператор безусловного перехода:

<метка> БП

С учетом введенных понятий ОПЗ выражения

```
y+(IF a>b THEN x+y ELSE x+2)
```

принимает следующий вид:

```
yab>M1 УПЛ x1+M2 БП M1: x2+M2:+
```

Для обобщения алгоритма Дейкстры на условные выражения требуется модифицировать таблицу приоритетов, исходя из следующих соображений. Нетрудно заметить, что IF играет роль открывающей скобки, а THEN и ELSE - запятых. Теперь с учетом данного обстоятельства модифицированная таблица приоритетов примет следующий вид (табл. 2.4).

Таблица 2.4

Входной элемент	Приоритет
IF ([АЭМ Ф	0
,)] THEN ELSE	1
V	2
&	3
¬	4
Отношения	5
+ -	6
* /	7
возведение в степень	8

Обработка символов операций будет дополнена правилами обработки ключевых слов условных выражений следующим образом:

- символ IF из входной строки заносится в стек;

- символ THEN выталкивает в выходную строку все операции из стека до ближайшего IF. В стеке к символу IF добавляется рабочая метка Mi и после этого в выходную строку записывается часть Mi УПЛ;
- ELSE выталкивает из стека все символы до ближайшего IF Mi. IF Mi превращается в IF Mi+1 и в выходную строку выталкивается часть Mi+1 БП Mi:
- Символ «)» указывает на конец условного выражения и выталкивает из стека все символы до ближайшего символа «(», при этом сам IF уничтожается, а в выходную строку помещается метка Mi+1: .

Например, для приведенного выражения процесс перевода в ОПЗ имеет вид:

Выходная строка	y				a		b	>	M1 УПЛ	x		1	+	M2 БП M1:	x		2	+	M2:	-
С Т Е К		-	(	IF		>			IF M1		+			IF M2		+				
			-	(		IF			(		IF M1			(		IF M2				
				-					-		(			-		(		-		
											-					-				
Входная строка	y	-	(	IF	a	>	b	THEN		x	+	1	ELSE		x	+	2	)		■

Примечание. Описанная выше обработка условных выражений связана с правильной организацией передачи управления внутри объектного кода, что и составляет внутреннюю семантику обработки условных выражений в ОПЗ.

2.6 Перевод оператора присваивания в ОПЗ

Формат оператора присваивания имеет вид:

<переменная> := <выражение>

а его ОПЗ представляется следующим образом:

<переменная> <выражение> :=

Для включения в ОПЗ оператора присваивания необходимо решить вопрос о месте этого оператора в таблице приоритетов. При выборе приоритета будем руководствоваться следующими соображениями:

- приоритет оператора присваивания должен быть выше, чем у любой арифметической или логической операции, чтобы знаки не могли вытолкнуть оператор присваивания из стека;
- приоритет должен быть меньше, чем у знака ';'.

С учетом этого, таблица приоритетов операций принимает следующий вид (табл. 2.5).

Таблица 2.5

Входной элемент	Приоритет
IF ([АЭМ Ф	0
, ;)] THEN ELSE	1
:=	2
V	3
&	4
¬	5
Отношения	6
+ -	7
* /	8
возведение в степень	9

2.7 Перевод оператора безусловного перехода и меток в ОПЗ

Формат оператора безусловного перехода имеет вид:

```
GOTO <метка>
```

а его ОПЗ представляется следующим образом:

```
<метка> БП
```

При установлении приоритета оператора безусловного перехода руководствуются такими соображениями, как и в случае оператора присваивания. Поэтому приоритеты этих операторов одинаковы.

Для перевода в ОПЗ меток из исходной программы вводится *операция метки*, которую для простоты обозначают символом ':'. Эта операция имеет наименьший приоритет.

Таблица приоритетов с учетом оператора безусловного перехода и операции метки принимает следующий вид (табл. 2.6).

Таблица 2.6

Входной элемент	Приоритет
IF ([АЭМ Ф	0
, ;)] THEN ELSE	1
:= GOTO	2
V	3
&	4
¬	5
Отношения	6
+ -	7
* /	8
Возведение в степень	9
:	10

Рассмотрим перевод в ОПЗ фрагмента программы, содержащего выше рассмотренные операторы и операции. Фрагмент исходной программы имеет следующий вид:

```
R1:=A+B;
```

```
GOTO M1;
R2:=C*D-A;
M1:
```

Перевод данного фрагмента программы в ОПЗ по алгоритму Дейкстры представлен ниже:

Выходная строка	R1		A		B	+	:=		M1	БП	R2		C		D	*	A	-	:=	M1		:
СТЕК		:=		+		:=		GOTO					:=	*		-		:=			:	
				:=										:=		:=						
Входная строка	R1	:=	A	+	B	;		GOTO	M1	;	R2	:=	C	*	D	-	A	;		M1	:	■

2.8 Перевод условного оператора в ОПЗ

Существует два вида условного оператора: полный

```
IF условие THEN оператор1 ELSE оператор2 ;
```

и неполный

```
IF условие THEN оператор ;
```

По аналогии с рассмотренным выше условным выражением для представления условного оператора также используются динамические деревья. В результате обхода данного динамического дерева ОПЗ условного оператора принимают вид:

- для полного условного оператора

  условие M1 УПЛ оператор1 M2 БП M1: оператор2 M2:

- для неполного оператора

  условие M1 УПЛ оператор M1:

В общем случае перевод условного оператора в ОПЗ по алгоритму Дейкстры совпадает с переводом условного выражения. Однако следует заметить, что в разных языках программирования в условных операторах принято различное употребление ограничителей ';' в условном операторе. В

этом случае функции ';' как концевого маркера оператора являются в разных языках различными. Например, в языке программирования Паскаль ';' всегда закрывает условный оператор. В этом случае этот знак должен вытолкнуть из стека IF, расставить соответствующие метки, но при этом сам IF в выходную строку не попадает.

2.9 Перевод описаний переменных и процедурных блоков в ОПЗ

В компиляторах обработка описаний переменных необходима для:

- выявления типов данных;
- подготовки информации для распределения памяти;
- выделения в объектной программе места для размещения данных.

Очевидно, что распределение памяти в объектной программе всегда должно предшествовать ее генерации. Поскольку исходная программа может содержать вложенные процедуры, то область действия идентификаторов, описанных во вложенных процедурах, является локализованной в пределах процедуры. Компилятор должен учитывать это обстоятельство и располагать информацией о начале и конце процедуры.

Приведем пример обработки описаний данных на базе описанного выше подмножества языка PL/1.

В этом языке программирования имеется несколько типов данных:

`DEC FLOAT - десятичное число с плавающей точкой;`

`DEC FIXED - десятичное число с фиксированной точкой;`

`BIN FIXED FIXED - двоичное число с фиксированной точкой;`

`CHAR – символьная строка.`

Сопоставим этим конструкциям языка в ОПЗ *операцию типа*, формат которой имеет вид:

Список переменных k ТИП,

где *k – количество описываемых переменных;*

$$ТИП = \begin{cases} DFT \\ DFD \\ BFD \\ CHAR \end{cases}$$

Поясним операцию типа, для чего предположим, что в программе описано несколько переменных

```
DCL (A,B,C) DEC FLOAT
```

С учетом операции типа построенная ОПЗ этого описания примет следующий вид:

```
A B C 3 DFT
```

Рассмотрим далее следующие операции описания процедур:

- *начало процедуры* - имеет два операнда: номер процедуры, уровень процедуры

<номер><уровень> НП

- *конец процедуры* - не имеет операндов

КП

- *конец описания* - имеет два операнда: номер процедуры, уровень процедуры

<номер><уровень>КО

Операторы начала и конца процедуры и описания данных в исходной программе являются неисполняемыми. Им можно присвоить самый низкий приоритет в таблица приоритетов, после чего она примет следующий вид (табл. 2.7).

Таблица 2.7

Входной элемент	Приоритет
IF ([АЭМ Ф	0
, ;)] THEN ELSE	1
:= GOTO	2
V	3
&	4
¬	5
Отношения	6
+ -	7
* /	8
возведение в степень	9
: PROC END DCL	10

Сам алгоритм Дейкстры с учетом обработки описательных операторов модифицируется следующим образом:

- символ PROC из входной строки заносится в стек, при этом в стек с ним также заносятся номер и уровень процедуры;
- символа DCL из входной строки заносится в стек с номером и уровнем процедуры и начальным значением счетчика операндов, равным 1;
- символ описания типа помещается в вершину стека;
- символ ',' из входной строки в зависимости от верхнего элемента стека выполняет различные функции:
    - если в вершине стека находится оператор DCL, значение счетчика операндов наращивается на 1 (запятая между операндами);
    - если в вершине стека находится символ описания типа, он выталкивает его из стека, помещает в выходную строку операцию ТИП со значением счетчика из оператора DCL и счетчик операндов принимает начальное значение 1 (запятая между символами описания типа);
- символ END из входной строки всегда заносится в стек;
- символ ';' из входной строки в зависимости от верхнего элемента стека выполняет различные функции:
    - если в вершине стека находится оператор PROC, то рассматриваемый символ выталкивает его из стека, занося в выходную строку операцию начала процедуры с ее номером и уровнем;
    - если в вершине стека находится оператор DCL, точка с запятой выталкивает его из стека, записывая в выходную строку операцию конца описания с номером и уровнем процедуры;
    - если в вершине стека находится символ описания типа, то рассматриваемый символ обрабатывает его аналогично запятой, а затем применяется к следующему в стеке оператору DCL;

- если в вершине стека находится оператор END, точка с запятой выталкивает его из стека, записывая в выходную строку операцию конца процедуры;

- символ '.' выталкивает из стека оператор END, записывая в выходную строку операцию конца процедуры.

Например, алгоритм перевода программы

```
PROC F1;
DCL (A,B) DEC FIXED,
          D DEC FLOAT;
     PROC F2;
          DCL (C,D) BIN FIXED;
     END;
END.
```

выглядит следующим образом:

Выходная строка		F1	1 1 НП			A		B	
С Т Е К	PROC 1,1			DCL 1,1,1	(		(		DCL 1,1,2
					DCL 1,1,1		DCL 1,1,2		
Входная строка	PROC	F1	;	DCL	(	A	,	B	)

Выходная строка		2 DFD	D		DFT	1 1 КО		F2	2 2 НП
С Т Е К	DFD	DCL 1,1,1		DFT	DCL 1,1,1		PROC 2,2		
	DCL 1,1,2			DCL 1,1,1					
Входная строка	DEC FIXED	,	D	DEC FLOAT	;		PROC	F2	;

Выходная строка			C		D			2 BFD	2 2 КО
С Т Е К	DCL 2,2,1	(		(		DCL 2,2,2	BFD	DCL 2,2,1	
		DCL 2,2,1		DCL 2,2,2			DCL 2,2,2		
Входная строка	DCL	(	C	,	D	)	BIN FIXED	;	

Выходная строка		КП		КП
СТЕК	END		END	
Входная строка	END	;	END	.

Таким образом, ОПЗ рассмотренной программы имеет следующий вид:

F1 1 1 НП A B 2 DFD D DFT 1 1 КО F2 2 2 НП C D 2 BFD 2 2 КО КП КП

2.10 Комплексный пример перевода исходной программы в ОПЗ

Рассмотрим комплексный пример перевода исходной программы на языке, описанном в лабораторной работе № 1, в обратную польскую запись.

```
PROC MAIN;
     DCL (A1, A2) DEC FIXED;
```

```
    A1=378;

    A2=.73;

    PROC CALC;
        DCL (SUM, MULT) DEC FIXED;
        IF A1+A2<>3.2 THEN GOTO P;
        SUM=(A1+A2)*A2;
    P:  MULT=A1*A2;
    END;
END.
```

Прежде всего, составим таблицу приоритетов, включив в нее все операции и операторы, имеющиеся во входном языке (табл. 2.8). Для этого воспользуемся обозначениями операторов и операций, которые были введены в лабораторной работе № 1.

Таблица 2.8

Приоритет	Входной элемент	Обозначение
0	IF	W6
	(	R4
1	THEN	W8
	.	R6
	,	R2
	;	R3
	)	R5
2	GOTO	W5
	=	O5
3	<	O3
	>	O4
4	+	O1
5	*	O2
6	PROC	W7
	END	W3
	DCL	W2
	:	O6

Привычное представление процесса перевода исходной программы в ОПЗ на основе алгоритма Дейкстры имеет следующий вид:

Выходная строка		MAIN	1 1 НП			A1		A2	
С Т Е К	PROC 1,1			DCL 1,1,1	(		(		DCL 1,1,2
					DCL 1,1,1		DCL 1,1,2		
Входная строка	PROC	MAIN	;	DCL	(	A1	,	A2	)

Выходная строка		2 DFD	1 1 КО	A1		378	:=	A2		.73	:=		CALC
С Т Е К	DFD	DCL 1,1,1			:=				:=			PROC 2,2	
	DCL 1,1,2												
Входная строка	DEC FIXED	;		A1	:=	378	;	A2	:=	.73	;	PROC	CALC

Выходная строка	2 2 НП			SUM		MULT			2 BFD	2 2 КО
С Т Е К		DCL 2,2,1	(		(		DCL 2,2,2	BFD	DCL 2,2,1	
			DCL 2,2,1		DCL 2,2,2			DCL 2,2,2		
Входная строка	;	DCL	(	SUM	,	MULT	)	BIN FIXED	;	

Выходная строка		A1		A2	+	3.2	>	M1 УПЛ		P	БП	M1:	SUM		A2		A1	+	:=
С Т Е К	IF		+		>		IF	IF M1	GOTO		IF M1			:=		+		:=	
			IF		IF				IF M1							:=			
Входная строка	IF	A1	+	A2	>	3.2	THEN		GOTO	P	;		SUM	:=	A2	+	A1	;	

Выходная строка	P	:	MULT		A1			A2		SUM	+	*	:=		КП		КП
С Т Е К				:=		*	(		+		*	:=		END		END	
						:=	*		(		:=						
							:=		*								
									:=								
Входная строка	P	:	MULT	=	A1	*	(	A2	+	SUM	)	;		END	;	END	.

Полученная в результате перевода ОПЗ программы имеет вид:

MAIN 1 1 НП A1 A2 2 DFD 1 1 КО A1 378 := A2 .73 := CALC2 2 НП

SUM MULT 2 BFD 2 2 КО A1 A2 + 3.2 > M1 УПЛ P БП M1: SUM A2 A1 + :=

P : MULT A1 A2 SUM + * := КП КП

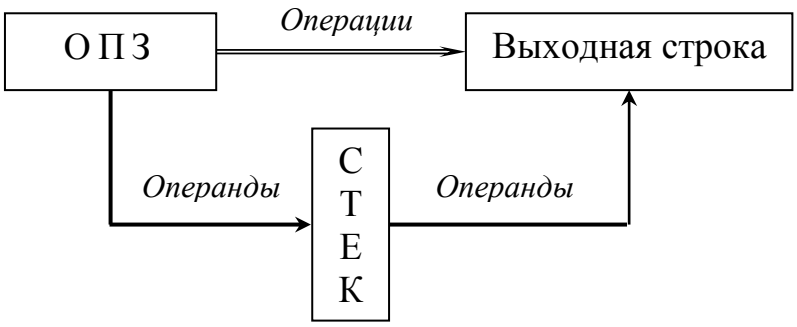
Для наглядности в рассмотренном примере текст программы представлен на исходном языке, но в реальном трансляторе входной текст является результатом работы лексического анализатора. В нем уже выделены и классифицированы лексемы, но в остальном работа протекает в соответствии с описанным выше примером.

3 Лабораторная работа № 3 «Перевод ОПЗ исходного выражения в текст на выходном языке. Генерация машинного кода»

3.1 Базовые понятия

Перевод ОПЗ в текст на выходном (машинном) языке представляет собой следующий этап трансляции исходной программы в машинные коды. Для реализации этой процедуры также используется автомат с магазинной памятью (МП-автомат).

Эту процедуру можно схематично представить следующим образом:



Чтение символов операций из ОПЗ инициирует семантические процедуры, которые генерируют соответствующие заготовки машинного кода. Так же обрабатываются собранные в стеке операнды данной. Например, в случае обработки переменных и констант извлекается соответствующая им информация из таблиц идентификаторов и констант, которая используется для образования правильных адресных частей соответствующих машинных команд.

3.2 Правила генерации машинного кода

Пусть в качестве машинного языка выступает язык программирования Бейсик. Его особенностью является обязательная нумерация строк и отсутствие символьных меток. Поэтому в операторах перехода в качестве меток используют номера строк, на которые нужно передать управление.

Для построения МП-автомата по переводу ОПЗ в машинные коды и его семантических процедур введем ряд внутренних переменных:

P - счетчик вспомогательных переменных;

STR - счетчик строк (этот счетчик вводится ввиду специфики выбранного выходного языка - Бейсика, где обязательна нумерация строк; при трансляции в другие выходные (машинные) языки этот счетчик не нужен);

Кроме того, организуем таблицу меток, которая реализует отображение символьных меток исходного языка в номера строк машинного языка:

Метка	Номер строки

Эта таблица потребуется в дальнейшем для замены символьных меток на номера строк, что также является особенностью Бейсика как выходного языка.

Рассмотрим работу МП-автомата.

1. Если элемент входной строки - идентификатор или константа, то он заносится в стек (в исходном виде, т.е. не условное обозначение, а имя из таблицы идентификаторов или константа из таблицы констант); вспомогательные переменные и константы переносятся без изменения.
2. Для каждой операции и оператора определяется арность, т.е. количество операндов, и соответствующая семантическая процедура.
3. После выполнения каждой семантической процедуры в выходную строку заносится символ <ВК>, счетчик строк STR наращивается на единицу и заносится в начало новой строки.

Семантические процедуры для операторов и операций приведены в табл. 3.1.

Таблица 3.1

Лексема	Действия
НП	Извлечь из стека два элемента, занести в выходную строку текст *"REM Начало процедуры арг2, арг1"*
КП	Занести в выходную строку текст *"REM Конец процедуры"*
DFD BFD DFT CHAR	Извлечь из стека *арг1* - число переменных k; извлечь из стека k аргументов; занести в выходную строку текст *"REM Вещественные переменные арг1, арг2,..,аргk"*
КО	Извлечь из стека два аргумента
УПЛ	Извлечь из стека два аргумента, занести в выходную строку текст *" IF NOT(арг2) THEN GOTO арг1"*
БП	Извлечь из стека один аргумент, занести в выходную строку текст *"GOTO арг1"*
:	Извлечь из стека один аргумент, занести в таблицу меток *арг1* и значение счетчика STR
+ * > <	Извлечь из стека два аргумента, нарастить счетчик вспомогательных переменных P, занести в выходную строку текст *"Rp = арг2 <операция> арг1"*, занести в стек Rp.
:=	Извлечь из стека 2 аргумента и занести в выходную строку текст *" арг2 = арг1 "*

В стеке *арг1* – это элемент, находящийся в вершине стека. Увеличение номера аргумента показывает его удаление от вершины стека и обратно порядку занесения элементов в стек.

Для арифметических выражений в целях уменьшения количества операторов присваивания и временных переменных возможен вариант формирования строки "(арг2 <операция> арг1)" и занесение ее в стек как единого аргумента для последующих операций и операторов. Недостатком такого подхода является избыточность круглых скобок в выражениях.

В языке Бейсик символьных меток не существует, поэтому необходим дополнительный просмотр сгенерированного текста с целью замены в нем символьных меток на номера строк в соответствии с построенной таблицей меток.

3.3 Комплексный пример перевода ОПЗ исходной программы в машинный код

Рассмотрим комплексный пример генерации машинного кода на языке Бейсик по ОПЗ исходной программы, которая была построена в примере, рассмотренном в лабораторной работе № 2.

MAIN 1 1 НП A1 A2 2 DFD 1 1 КО A1 378 := A2 .73 := CALC2 2 НП

SUM MULT 2 BFD 2 2 КО A1 A2 + 3.2 > M1 УПЛ P БП M1: SUM A2 A1 + :=

P : MULT A1 A2 SUM + * := КП КП

Последовательность действий МП-автомата изобразим в виде таблицы (табл. 3.2), в строке которой будем записывать состояние стека, счетчика переменных P, счетчика строк STR после обработки элемента ОПЗ, а также сформированный фрагмент машинного кода.

Таблица 3.2

Шаг	Элемент ОПЗ	Стек	P	STR	Машинный код
0.	-	-	1	1	
1.	MAIN	MAIN	1	1	

Шаг	Элемент ОПЗ	Стек	Р	STR	Машинный код
2.	1	1 MAIN	1	1	
3.	1	1 1 MAIN	1	1	
4.	НП	-	1	2	1 REM Начало процедуры MAIN
5.	A1	A1	1	2	
6.	A2	A2 A1	1	2	
7.	2	2 A2 A1	1	2	
8.	DFD	-	1	3	2 REM Вещественные переменные A1, A2
9.	1	1	1	3	
10.	1	1 1	1	3	
11.	КО	-	1	3	
12.	A1	A1	1	3	
13.	378	378 A1	1	3	
14.	:=	-	1	4	3 A1=378
15.	A2	A2	1	4	
16.	.73	.73 A2	1	4	
17.	:=	-	1	5	4 A2=.73
18.	CALC	CALC	1	5	

Шаг	Элемент ОПЗ	Стек	P	STR	Машинный код
19.	2	2 CALC	1	5	
20.	2	2 2 CALC	1	5	
21.	НП	-	1	6	5 REM Начало процедуры CALC
22.	SUM	SUM	1	6	
23.	MULT	MULT SUM	1	6	
24.	2	2 MULT SUM	1	6	
25.	BFD	-	1	7	6 REM Вещественные переменные SUM, MULT
26.	2	2	1	7	
27.	2	2 2	1	7	
28.	КО	-	1	7	
29.	A1	A1	1	7	
30.	A2	A2 A1	1	7	
31.	+	R1	2	8	7 R1=A1+A2
32.	3.2	3.2 R1	2	8	
33.	>	R2	3	9	8 R2=R1>3.2
34.	M1	M1 R2	3	9	
35.	УПЛ	-	1	10	9 IF NOT(R2) GOTO M1

Шаг	Элемент ОПЗ	Стек	P	STR	Машинный код
36.	P	P	1	10	
37.	БП	-	1	11	10 GOTO P
38.	M1	M1	1	11	
39.	:	-	1	11	*Занести в таблицу меток пару M1, 11*
40.	SUM	SUM	1	11	
41.	A2	A2 SUM	1	11	
42.	A1	A1 A2 SUM	1	11	
43.	+	R1 SUM	2	12	11 R1=A2+A1
44.	:=	-	1	13	12 SUM=R1
45.	P	P	1	13	
46.	:	-	1	13	*Занести в таблицу меток пару P, 13*
47.	MULT	MULT	1	13	
48.	A1	A1 MULT	1	13	
49.	A2	A2 A1 MULT	1	13	
50.	SUM	SUM A2 A1 MULT	1	13	
51.	+	R1 A1 MULT	2	14	13 R1=A2+SUM

Шаг	Элемент ОПЗ	Стек	P	STR	Машинный код
52.	*	R1 MULT	2	15	14 R1=A1*R1
53.	:=	-	1	16	15 MULT=R1
54.	КП	-	1	17	16 REM Конец процедуры
55.	КП	-	1	18	17 REM Конец процедуры

После замены символьных меток на числовые получим следующий машинный код

```
1 REM Начало процедуры MAIN
2 REM Вещественные переменные A1, A2
3 A1=378
4 A2=.73
5 REM Начало процедуры CALC
6 REM Вещественные переменные SUM, MULT
7 R1=A1+A2
8 R2=R1>3.2
9 IF NOT(R2) GOTO 11
10 GOTO 13
11 R1=A2+A1
12 SUM=R1
13 R1=A2+SUM
14 R1=A1*R1
15 MULT=R1
16 REM Конец процедуры
17 REM Конец процедуры
```

Полученный текст на машинном языке соответствует всем требованиям языка Бейсик.

4 Лабораторная работа № 4 «Построение синтаксического анализатора»

Основная задача синтаксического анализа - проверка исходной программы на соответствие грамматике языка программирования. Следует еще раз напомнить, что синтаксический анализ производится над кодом программы, который получен на выходе лексического анализа. Результат синтаксического анализа, который часто называется разбором, представляется в виде дерева разбора. Данное дерево должно демонстрировать вывод исходной программы как цепочки символов из начального символа грамматики.

Для построения дерева разбора используются различные методы синтаксического анализа, каждый из которых имеет свои особенности. В нашем случае будет рассмотрен пример разбора «сверху-вниз», который называется рекурсивным спуском. Основная особенность рекурсивного спуска заключается в том, что для анализа каждого нетерминала используется отдельная семантическая процедура, а многократное обращение к ней в процессе анализа и дало в название методу – «рекурсивный спуск».

Рассмотрим суть синтаксического анализа методом рекурсивного спуска на примере.

Вначале построим грамматику языка программирования, описанного в лабораторной работе № 1. Его грамматика, начальным символом которой является нетерминал <программа>, имеет следующий вид:

<программа> ::= *PROC MAIN;*<текст> *END.*

<текст> ::= (<строка>|<процедура>){<строка>|<процедура>}

<строка> ::= <описание>

|<отд.оператор>

<описание> ::= *DCL*<идентфикатор>{,<идентификатор>}*DEC FIXED;*

<отд.оператор> ::= <идентификатор>:<оператор>

|<оператор>

<оператор> ::= <идентификатор>:=<выражение>

| *GOTO* <идентификатор>

| <условный оператор>

<процедура> ::= *PROC* <идентификатор>;<текст>*END*;

<условный оператор> ::= *IF* <условие> *THEN* <оператор>

<условие> ::= <выражение><неравенство><выражение>

<выражение> ::= <терм>{+<терм>}

<терм> ::= <множитель>{*<множитель>}

<множитель> ::= <аргумент>|(<выражение>)

<аргумент> ::= <идентификатор>

|<константа>

<идентификатор> ::= <буква>{<буква>|<цифра>}

<константа> ::= <число>

|.<число>

|<число>.<число>

|<число>.

<буква> ::= *A* | *B* |...| *Y* | *Z*

<цифра> ::= *0* | *1* |...| *8* | *9*

Теперь введем некоторые термины.

Цель. Под ней будем понимать некоторый нетерминальный символ, который в данный момент времени является для дерева концевым. Задача заключается в том, чтобы из него продолжить построение дерева (найти вывод). И так далее до тех пор, пока в качестве цели не окажется терминальный символ.

Построение дерева разбора в методах синтаксического анализа «сверху-вниз» всегда производится слева направо (в качестве цели всегда выбирается самый левый концевой символ). Здесь очень важно найти в

разбираемой цепочке входной программы самую левую подцепочку, выводимую из цели. Эта подцепочка называется фразой для нетерминала U и в методах синтаксического анализа играет ключевую роль. Для ясности дадим формальное определение понятия фразы.

Фраза. Пусть задана грамматика G[Z], где Z – начальный символ и некоторая цепочка xUy, тогда цепочка *u* называется фразой для нетерминала U, если выполняется два условия:

$$Z \Rightarrow^{+} xUy$$

$$U \Rightarrow^{+} u$$

Если вывод $U \Rightarrow u$ является простым, то и фраза *u* также называется простой.

Итак, при разборе семантическая процедура нетерминального символа (цели) всегда для разбираемой цели ищет самую левую простую фразу, заменяет ее на цель и далее разбор повторяется для новой цели.

Для устранения лишних возвратов в рекурсивном спуске в качестве контекста используется единственный символ, следующий за разобранной частью фразы, который постоянно отслеживается.

Рекурсивная процедура для нетерминала U ищет фразу следующим образом: сравнивает разбираемый символ с правыми частями правил для данного нетерминала U, выбирает подходящее правило грамматики и по мере необходимости вызывает другие процедуры для используемых в правиле нетерминалов. Таким образом происходит спуск до уровня правил, не имеющих в правых частях нетерминалов.

Процедура, удачно завершившая разбор соответствующего ей правила, завершает свою работу, возвращая в точку вызова признак успешного окончания. В противном случае (если соответствующее ей правило (правила) неприменимы в данном контексте), в точку вызова возвращается признак неудачи, и вызывающая процедура переходит к разбору по следующему правилу, если таковое имеется, или, в свою очередь, при его отсутствии возвращает признак неудачи.

При программной реализации данного алгоритма были определены следующие переменные и процедуры, за которыми закреплены определенные функциональные назначения.

1. Переменная NXTSYMB - глобальная переменная, содержащая символ, следующий за разбираемым символом. NXTSYMB содержит символ (лексему) исходной программы, который будет обрабатываться следующим, а при поиске новой цели в переменной NXTSYMB всегда находится первый символ, который должен быть исследован.

При выходе из процедуры с сообщением об успехе символ, следующий за закрытой подцепочкой, помещается в NXTSYMB.

2. Процедура SCAN готовит очередной символ исходной программы и помещает его в NXTSYMB.

3. Процедура ERROR предназначена для обработки ошибочных ситуаций.

Ниже в таблице 4.1 собраны нетерминалы грамматики и имена соответствующих им рекурсивных семантических процедур.

Таблица 4.1

Нетерминальные символы грамматики	Имена рекурсивных процедур
<программа>	PROGRAM
<текст>	TEXT
<строка>	LINE
<описание>	DECLARE
<отд.оператор>	SEP_OPER
<оператор>	OPERATOR
<процедура>	PROCED
<условный оператор>	IF_OPER
<условие>	CONDITION
<выражение>	EXPRESSION
<терм>	TERM
<множитель>	FACTOR
<аргумент>	ARGUMENT
<идентификатор>	IDENT
<константа>	CONST

Процедуры IDENT и CONST в отличие от всех остальных процедур возвращают значение ИСТИНА или ЛОЖЬ в зависимости от того, является ли лексема, содержащаяся в переменной NXTSYMB, идентификатором или константой соответственно.

При инициализации синтаксического анализатора идет обращение к процедуре SCAN, которая помещает первый символ исходной программы в глобальную переменную NXTSYMB и вызывает процедуру PROGRAM.

Для рассмотренной грамматики программа синтаксического анализатора по методу рекурсивного спуска приведена ниже.

```
procedure PROGRAM;
begin
    SCAN;
    if NXTSYMB<>'PROC' then ERROR;
    SCAN;
    if NXTSYMB<>'MAIN' then ERROR;
    if NXTSYMB<>';' then ERROR;
    SCAN;
    TEXT;
    SCAN;
    if NXTSYMB<>'END' then ERROR;
    SCAN;
    if NXTSYMB<>'.' then ERROR;
end.

procedure TEXT;
while NXTSYMB<>'END' do
    if NXTSYMB='PROC' then PROCED
        else LINE;
end;

procedure PROCED;
begin
    SCAN;
    if NOT IDENT (NXTSYMB) then ERROR;
    SCAN;
    if NXTSYMB<>';' then ERROR;
    SCAN;
```

```
    while NXTSYMB<>'END' do begin
        LINE;
        SCAN;
    end;
    SCAN;
    if NXTSYMB<>';' then ERROR;
end;

procedure LINE;
if NXTSYMB='DCL' then DECLARE
    else SEP_OPER;
end;

procedure DECLARE;
begin
    SCAN;
    if NOT IDENT (NXTSYMB) then ERROR;
    SCAN;
    while NXTSYMB=',' do begin
        SCAN;
        if NOT IDENT (NXTSYMB) then ERROR;
        SCAN;
    end;
    if NXTSYMB<>'DEC' then ERROR;
    SCAN;
    if NXTSYMB<>'FIXED' then ERROR;
    SCAN;
    if NXTSYMB<>';' then ERROR;
    SCAN;
end;
```

```
procedure SEP_OPER;
if NXTSYMB='IF' OR NXTSYMB='GOTO' then OPERATOR
    else begin
        if NOT IDENT (NXTSYMB) then ERROR;
        SCAN;
        if NXTSYMB=':' then begin
                SCAN;
                OPERATOR;
        end;
        else if NXTSYMB=':=' then begin
                    SCAN;
                    EXPRESSION;
                end;
            else
                ERROR;
end;

procedure OPERATOR;
if NXTSYMB='IF' then IF_OPER
    else if NXTSYMB='GOTO' then begin
        SCAN;
        if NOT IDENT (NXTSYMB) then ERROR;
    end;
else begin
    IDENTIFIER;
    if NXTSYMB<>':=' then ERROR
        else begin
            SCAN;
            EXPRESSION;
```

```
        end;
    end;
end;

procedure IF_OPER;
begin
    SCAN;
    CONDITION;
    if NXTSYMB<>'THEN' then ERROR
        else begin
            SCAN;
            OPERATOR;
        end;
end;

procedure CONDITION;
begin
    EXPRESSION;
    if NXTSYMB<>'>' then
        if NXTSYMB<>'<' then ERROR;
    SCAN;
    EXPRESSION;
end;

procedure EXPRESSION;
begin
    TERM;
    SCAN;
    while NXTSYMB='+' do begin
        SCAN;
```

```
        FACTOR;
    end;
end;

procedure TERM;
begin
    FACTOR;
    while NXTSYMB='*' do begin
        SCAN;
        FACTOR;
    end;
end;

procedure FACTOR;
if NXTSYMB='(' then begin
        SCAN;
    EXPRESSION;
        if NXTSYMB<>')' then ERROR;
            else
                SCAN;
        end;
    else begin
            ARGUMENT;
            SCAN;
        end;
end;

procedure ARGUMENT;
if NOT CONST(NXTSYMB) then
```

```
        if NOT IDENT(NXTSYMB)) then ERROR;
    end;
```

При выполнении лабораторной работы № 4 студенты должны внимательно ознакомиться с предлагаемым подходом и по аналогии с ним выполнить разработку синтаксического анализатора языка программирования, предложенного в варианте задания к лабораторной работе.

Варианты заданий

Каждый вариант задания представляет собой пару: входной язык и машинный, или выходной, язык (см. табл. 5.1). В качестве входного языка предлагается один из языков программирования высокого уровня, в который должно быть обязательно включено следующее подмножество языковых конструкций и операторов:

- идентификаторы;
- числовые константы целого типа и вещественного типа, представленные с фиксированной и плавающей точкой;
- символьные (строковые) константы;
- переменные с индексами (массивы и элементы массивов);
- комментарии (строчные и блочные);
- имена функций пользователя;
- арифметические операции;
- операции сравнения (меньше, больше, равно, не равно, меньше или равно, больше или равно);
- операторы описания данных (идентификаторов и массивов);
- операторы описания процедур и функций (если предусмотрены в языке);
- операторы условного и безусловного перехода;
- метки (если предусмотрены в языке).

В качестве машинного языка предлагаются различные языки высокого уровня и язык Ассемблера (см. таблицу 5.1).

Таблица 5.1

№ варианта	Входной язык	Выходной язык
1	Паскаль	Си
2	Бейсик	Си
3	Perl	Си

№ варианта	Входной язык	Выходной язык
4	Фортран	Си
5	JavaScript	Си
6	Паскаль	Perl
7	Бейсик	Perl
8	Си	Perl
9	Фортран	Perl
10	JavaScript	Perl
11	Паскаль	Фортран
12	Бейсик	Фортран
13	Perl	Фортран
14	Си	Фортран
15	JavaScript	Фортран
16	Паскаль	Бейсик
17	Си	Бейсик
18	Perl	Бейсик
19	Фортран	Бейсик
20	JavaScript	Бейсик
21	Паскаль	JavaScript
22	Бейсик	JavaScript
23	Perl	JavaScript
24	Фортран	JavaScript
25	Си	JavaScript
26	Паскаль	Ассемблер
27	Си	Ассемблер

№ варианта	Входной язык	Выходной язык
28	Бейсик	Ассемблер
29	Perl	Ассемблер
30	Фортран	Ассемблер
31	JavaScript	Ассемблер
32	Си	Паскаль
33	Бейсик	Паскаль
34	Perl	Паскаль
35	Фортран	Паскаль
36	JavaScript	Паскаль

Приступая к выполнению лабораторных работ, студент должен внимательно ознакомиться со своим вариантом задания, сформировать согласованное с преподавателем подмножество конструкций выбранной версии входного языка и используемые инструментальные средства разработки.

Порядок выполнения лабораторных работ и требования к их оформлению

Цикл состоит из четырех лабораторных работ:

1. Построение программы ***лексического анализатора***, корректно распознающего все перечисленные выше лексемы и формирующего таблицы лексем и внутреннее представление проанализированного текста.

На вход программы подается файл, содержащий тест на входном языке программирования. Результатом работы программы должен быть файл, содержащий последовательность кодов лексем входной программы, а также один или несколько файлов, содержащие все таблицы лексем.

Отчет по работе должен содержать описание правил записи перечисленных выше элементов заданного входного языка, стандартные таблицы лексем, диаграмма состояний сканера, листинг программы и комментарии к нему, пример лексического разбора.

2. Построение программы для ***перевода закодированного текста исходной программы в обратную польскую запись.***

Программа получает на входе файл - результат лексического анализа и строит обратную польскую запись исходной программы.

Отчет по работе должен содержать полное описание алгоритма Дейкстра для конкретного языка: таблицу приоритетов операторов и операций, а также алгоритм работы со стеком. Листинг программы и комментарии к нему, пример.

3. Разработка программы для ***формирования по обратной польской записи текста на выходном (машинном) языке.***

Программа получает на входе файл, содержащий ОПЗ исходной программы, и строит текст программы на машинном язык в соответствии с заданием.

Отчет по работе должен содержать описание правил записи перечисленных выше элементов заданного выходного языка, алгоритм

работы МП-автомата и описание семантических процедур, листинг программы и комментарии к нему, пример.

4. Разработка программы ***синтаксического анализатора*** исходного текста.

Программа получает на входе файл - результат лексического анализа и выполняет синтаксический анализ исходной программы. Результатом работы должно быть сообщение о корректности программы или сообщение об первой обнаруженной ошибке с указанием строки и конструкции языка, при разборе которых обнаружена ошибка.

Отчет по работе должен содержать полную грамматику заданного подмножества входного языка, описание алгоритма синтаксического разбора, листинг программы и комментарии к нему, пример.

5. На рейтинговой неделе для получения допуска к экзамену студент должен представить единый отчет по циклу лабораторных работ, полную версию разработанного программного комплекса и продемонстрировать его работу преподавателю на примере.

Приложение. Описание языков программирования, выбранных в качестве входных языков

Язык программирования Паскаль

(TurboPascal 7.0)

Идентификаторы

Произвольная последовательность букв и цифр, начинающаяся с буквы. Может включать символы подчеркивания.

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида

```
123.45
.25
25.
```

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «e» или «E», а также знак «+» или «-» вида (необязательный):

```
1.23e-25
1.23E-25
1.23e+25
1.23E+25
1.23e2
1.23E2
1E-78
```

```
1e67
```

Символьные (строковые) константы

Последовательность символов, заключенная в апострофы, расположенная в пределах одной строки, вида:

```
'acb 12_& ?tu'
```

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в квадратных скобках через запятую перечислены выражения-индексы, вида:

```
Abc[12, I, i-6]
C[1+i]
```

Комментарии (строчные и блочные)

Только блочные – последовательность символов, заключенная в фигурные скобки, возможно содержащая несколько строк:

```
{ Это комментарий,
Который содержит 2 строки}
```

Обращения к процедурам и функциям пользователя

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Отсутствие аргументов не допускается:

```
F(12, 4, i)
f(av-6)
```

Арифметические операции

Сложение	+
Вычитание	-
Умножение	*
Деление	/
Возведение в степень	^

Операции сравнения

Меньше	<
Больше	>
Равно	=
Не равно	<>
Меньше или равно	<=
Больше или равно	>=

Оператор присваивания

Имеет вид «:=». Слева стоит идентификатор или элемент массива, а справа – выражение. Заканчивается символом «;», например:

```
a:=b+c;
b[2,i-9]:=12;
```

Операторы блока

```
Begin – начало блока
…
End; - конец блока
```

Оператор описания программы

Программа начинается оператором Program с указанием имени программы. Затем могут идти описания даны, процедур и функций, а затем тело программы, заключенное в операторы блока, оканчивающееся точкой.

```
Program <идентификатор>;
…
Begin
…
End.
```

Операторы описания данных (идентификаторов и массивов)

Начинается оператором Var и может содержать несколько строк описаний, состоящих из перечисления идентификаторов через запятую и после двоеточия ключевое слово типа.

```
Var
    A,b: real;
    C: integer;
```

Типы переменных: integer (целый), real (вещественный), string (строковый)

Для массивов после двоеточия указывается ключевое слово массива «array of», в квадратных скобках через запятую перечисляются границы изменения каждого из индексов разделенные символами «..», и затем тип элементов:

```
Var
a,b,c : array of [1..3, 10..20] of integer;
```

Операторы описания процедур и функций

Процедуры имеют заголовок вида

```
procedure <идентификатор> (<список формальных параметров>);
```

и тело – список операторов, заключенный в операторы блока

```
begin … end;
```

Между заголовком и телом может присутствовать оператор описания данных Var. Например:

```
procedure abc (r: real);
var
r1,r2:real;
begin
y:=sinr(r1)/cos(r2)*tan(r);
end;
```

Функции имеют заголовок вида:

```
function <идентификатор> (<список формальных параметров>): <тип возвращаемого значения>;
```

В остальном структура функций аналогична структуре процедур. Исключение составляет обязательное присутствие в теле функции хотя бы одного оператора `return <значение>;`

Оператор безусловного перехода и метки

```
goto <метка> ;
```

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

```
a: str:='ujhti';
```

Оператор условного перехода

Начинается с ключевого слова «if», имеет полный и неполный формат:

```
If <условие> then <оператор_1> else <оператор_2>;
If <условие> then <оператор_1>;
```

Язык программирования Java Script

(ECMA-262 - Netscape)

Идентификаторы

Произвольная последовательность букв и цифр, начинающаяся с буквы. Может включать символы подчеркивания.

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «e» или «E», а также знак «+» или «-» вида (необязательный):

1.23e-25

1.23E-25

1.23e+25

1.23E+25

1.23e2

1.23E2

1E-78

1e67

Символьные (строковые) константы

Набор символов, возможно пустой, заключенный в апострофы или кавычки:

```
"Это строковая константа"
'Это тоже строковая константа'
```

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в квадратных скобках стоит выражение-индекс, вида:

```
Abc[12]
C[1+i]
```

Комментарии (строчные и блочные)

Только строчные – последовательность символов от знаков «//» до конца строки,

```
X=45 //Это комментарий
```

Обращения к функциям пользователя

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Отсутствие аргументов не допускается:

```
F(12, 4, i)
f(av-6)
```

Арифметические операции

Сложение	+
Вычитание	-
Умножение	*
Деление	/
Возведение в степень	^

Операции сравнения

Меньше	<
Больше	>
Равно	==
Не равно	!=
Меньше или равно	<=
Больше или равно	>=

Оператор присваивания

Имеет вид «=». Слева стоит идентификатор или элемент массива, а справа – выражение. Заканчивается символом «;», например:

```
a=b+c;
b[2]=12;
```

Операторы блока

```
{ – начало блока
…
} - конец блока
```

Операторы описания данных (идентификаторов и массивов)

Начинается оператором Var и может содержать список идентификаторов через запятую. Типы отсутствуют.

```
var <имя переменной>;
var x,y,z;
```

Для одномерных массивов после идентификатора указывается ключевое слово массива « = new array» и в круглых скобках указывается количество элементов:

```
var <идентификатор> = new array(количество элементов)
var c = new array(100)
```

Многомерные массивы в языке JavaScript отсутствуют.

Операторы описания функций

Процедуры имеют заголовок вида

```
function <идентификатор> (<список формальных параметров>);
```

и тело – список операторов, заключенный в операторы блока

```
{ … }
```

Например:

```
function sum(a,b)
{
var y;
y=a+b;
return y;
}
```

Оператор безусловного перехода и метки

```
goto <метка> ;
```

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

```
a: str:='ujhti';
```

Оператор условного перехода

Начинается с ключевого слова «if», имеет полный и неполный формат:

```
if (логическое выражение)
{
операторы
}

if (логическое выражение)
{
Операторы_1
```

```
}
else
{
Операторы_2
}
```

Язык программирования Basic

(Microsoft Turbo Basic)

Идентификаторы

Произвольная последовательность букв и цифр, начинающаяся с буквы. Последний символ имени определяет тип идентификатора:

```
a$ - символьный
a% - целый
a& - длинный целый
a! - вещественный обычной точности
a# - вещественный двойной точности
```

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «e» или «E», а также знак «+» или «-» вида (необязательный):

1.23e-25

1.23E-25

1.23e+25

1.23E+25

1.23e2

1.23E2

1E-78

1e67

Символьные (строковые) константы

Последовательность символов, заключенная в кавычки, расположенная в пределах одной строки, вида:

“acb 12_& ?tu”

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в круглых скобках через запятую перечислены выражения-индексы, вида:

```
Abc%(12, I, i-6)
C$(1+i)
```

Комментарии (строчные и блочные)

Только строчные – строка, начинающаяся с оператора «REM».

```
REM Это комментарий
I%=12
```

Обращения к функциям пользователя

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Отсутствие аргументов не допускается:

```
M=F(12, 4, i)
f(av-6)
```

Арифметические операции

Сложение	+
Вычитание	-
Умножение	*

Деление /

Возведение в степень ^

Операции сравнения

Меньше <

Больше >

Равно =

Не равно <>

Меньше или равно <=

Больше или равно >=

Оператор присваивания

Имеет вид «=». Слева стоит идентификатор или элемент массива, а справа – выражение. Заканчивается символом «;», например:

```
a%=b%+c%
b$(2,i-9)=”12”
```

Оператор останова

Прерывает выполнение программы:

```
STOP
```

Оператор окончания программы/процедуры

Завершает текст модуля:

```
END
```

Операторы описания массивов

Описание массивов осуществляется с помощью оператора DIM с указанием размеров. Например, оператор

```
DIM a(10),  b(10:20,  25:45)
```

описывает одномерный массив a, элементы которого имеют индексы от 0 до 10, и двухмерный массив b, элементы которого имеют индексы :

первый от 10 до 20, второй от 25 до 45.

Если нижняя граница индексов в описании не указана, то она считается равной 0.

В описании массива вместо константы может использоваться переменная. Например,

```
DIM a(n)
```

Значение n должно быть предварительно определено.

Оператор безусловного перехода и метки

```
goto <метка> ;
```

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

```
a: str$=”ujhti”;
```

Операторы описания процедур и функций

Подпрограмма - помеченная меткой последовательность операторов, оканчивающаяся оператором RETURN. Выполняется, когда достигнут оператор GOSUB.

```
....
GOSUB aa
....
END
aa:
 <операторы>
RETURN
```

Оператор RETURN осуществляет возврат к оператору, непосредственно следующему за GOSUB.

Оператор условного перехода

Начинается с ключевого слова «if», имеет полный и неполный формат:

```
IF <условие> THEN <оператор1> [ELSE <оператор2>]
```

Например:

```
IF a < b THEN t=15 ELSE t=17
```

Если после THEN или после ELSE располагается целая группа операторов, то можно использовать IF блок, который имеет следующую структуру

```
IF <условие> THEN
      <операторы1>
ELSE
      <операторы2>
END IF
```

При этом ELSE и операторы за ним могут отсутствовать, т.е. возможна конструкция

```
IF  <условие> THEN
      <операторы>
END IF
```

Язык программирования C++

(ISO/IEC 14882)

Идентификаторы

Произвольная последовательность букв и цифр, начинающаяся с буквы. Может включать символы подчеркивания и начинаться с них.

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «e» или «E», а также знак «+» или «-» вида (необязательный):

1.23e-25

1.23E-25

1.23e+25

1.23E+25

1.23e2

1.23E2

Символьные (строковые) константы

Символьная константа – один символ, заключенный в апострофы:

```
'a'
```

Строковая константа - последовательность символов, заключенная в кавычки, расположенная в пределах одной строки, вида:

```
"acb 12_& ?tu"
```

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в квадратных скобках перечислены выражения-индексы, вида:

```
Abc[12][I][i-6]
C[1+i]
```

Комментарии (строчные и блочные)

Блочные – последовательность символов, начинающаяся с «/*» и оканчивающаяся «*/», возможно содержащая несколько строк:

```
/* Это комментарий,
Который содержит 2 строки*/
```

Строчные – от символов «//» до конца строки.

```
i=i+1; // это инкремент
```

Обращения к функциям пользователя

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Скобки могут быть пустыми в случае отсутствия аргументов:

```
F(12, 4, i)
f(av-6)
g()
```

Арифметические операции

Сложение	+
Вычитание	-
Умножение	*
Деление	/

Операции сравнения

Меньше	<
Больше	>
Равно	==
Не равно	!=
Меньше или равно	<=
Больше или равно	>=

Оператор присваивания

Имеет вид «=». Слева стоит идентификатор или элемент массива, а справа – выражение. Заканчивается символом «;», например:

```
a=b+c;
b[2][i-9]=12;
```

Операторы блока

```
{ – начало блока
…
} - конец блока
```

Структура программы

Программа начинается операторами описания данных. Затем могут идти описания данных и функций, а затем основная функция программы void main () и ее тело, заключенное в операторы блока, оканчивающееся точкой.

```
Описания
void main()
{
…
}
```

Операторы описания данных (идентификаторов и массивов)

Начинается с ключевого слова типа и содержит перечисление идентификаторов через запятую. Оканчивается знаком «;»

```
<тип> <список элементов>;
```

Типы переменных: int (целый), float (вещественный), char (символьный)

Элементом списка может быть массив, для которого указывается идентификатор и размерности:

```
int a,b,c;
float d[3][4], c[78];
```

Операторы описания функций

Функции имеют заголовок вида

```
<тип> <идентификатор> (<список формальных параметров>);
```

и тело – список операторов, заключенный в операторы блока

```
{ … };
```

Например:

```
int abc (float r)
{
float r1,r2;
y:=sinr(r1)/cos(r2)*tan(r);
}
```

В теле функции может присутствовать оператор

```
return (<значение>);
```

Оператор безусловного перехода и метки

```
goto <метка> ;
```

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

```
a: str='ujhti';
```

Оператор условного перехода

Начинается с ключевого слова «if», имеет полный и неполный формат:

```
if (логическое выражение) оператор_1 else
    оператор_2;
    if (логическое выражение) оператор_1;
```

Вместо отдельных операторов могут использоваться блоки операторов:

```
    if (логическое выражение)
    {операторы_1}
    else
    {операторы_2}
```

Язык программирования Perl

(5.003 for FreeBSD 2.1.0.)

Идентификаторы

Произвольная последовательность букв и цифр, начинающаяся со специального символа или буквы. Может включать символы подчеркивания и начинаться с них.

Специальный начальный символ определяет тип идентификатора:

Отсутствие символа означает, что идентификатор является именем процедуры

$ - идентификатор обозначает обычную переменную

@ - идентификатор является именем массива (структуры)

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «e» или «E», а также знак «+» или «-» вида (необязательный):

1.23e-25

1.23E-25

1.23e+25

1.23E+25

1.23e2

1.23E2

Символьные (строковые) константы

Строковая константа - последовательность символов, заключенная в апострофы или кавычки, расположенная в пределах одной строки, вида:

```
“acb 12_& ?tu”
‘abc’
```

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в квадратных скобках перечислены выражения-индексы, вида:

```
@Abc[12]
@C[1+i]
```

Комментарии (строчные и блочные)

Блочные комментарии отсутствуют.

Строчные – от символа «#» до конца строки.

```
$i=$i+1; # это инкремент
```

Обращения к подпрограммам

Идентификатор, следующий после знака «&», после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. При отсутствии аргументов скобки не ставятся:

```
&F(12, 4, $i);
&f($av-6);
&g;
```

Арифметические операции

Сложение	+
Вычитание	-

Умножение	*
Деление	/
Возведение в степень	**

Операции сравнения

Меньше	<
Больше	>
Равно	==
Не равно	!=
Меньше или равно	<=
Больше или равно	>=

Оператор присваивания

Имеет вид «=». Слева стоит идентификатор или элемент массива, а справа – выражение. Заканчивается символом «;», например:

```
$a=$b+$c;
@b[i-9]=12;
```

Операторы блока

```
{ – начало блока
…
} - конец блока
```

Структура программы

Программа представляет собой произвольную последовательность операторов и подпрограмм.

Операторы описания данных (идентификаторов и массивов)

Операторы описания данных в языке отсутствуют.

Операторы описания подпрограмм

Подпрограммы имеют заголовок вида

```
sub <идентификатор>
```

и тело – список операторов, заключенный в операторы блока

```
{ … }
```

Например:

```
     sub show_value
         {
           print 'The value    id ', $_[0];
         }
       &show_value(1001);
```

В теле подпрограммы может присутствовать оператор

```
return <значение>;
```

Оператор безусловного перехода и метки

```
goto <метка> ;
```

Метка - идентификатор, расположенный в теле программы в начале строки, после которого стоит знак «:»:

```
a: $str='ujhti';
```

Оператор условного перехода

Начинается с ключевого слова «if», имеет полный и неполный формат:

```
if (<логическое выражение>)
        {
            <оператор_1>;
        }
else
        {
            <оператор_2>;
        }
```

или

```
if (<логическое выражение>)
        {
```

```
    <оператор>;
}
```

Язык программирования Fortran

(Fortran IV):

Идентификаторы

Произвольная последовательность прописных букв и цифр, начинающаяся с буквы.

Числовые константы целого типа

Произвольная последовательность цифр без знака.

Числовые константы вещественного типа, представленные с фиксированной точкой

Последовательность цифр, включающая одну десятичную точку вида

123.45

.25

25.

Числовые константы вещественного типа, представленные с плавающей точкой

Последовательность, включающая цифры, десятичную точку (необязательную), символ «E», а также знак «+» или «-» вида (необязательный):

1.23E-25

1.23E+25

1.23E2

.25E-6

Символьные (строковые) константы

Символьная константа – один символ, заключенный в апострофы:

'a'

Строковая константа - последовательность символов, заключенная в кавычки, расположенная в пределах одной строки, вида:

```
"acb 12_& ?tu"
```

Переменные с индексами (массивы и элементы массивов)

Идентификатор, после которого в квадратных скобках перечислены выражения-индексы, вида:

```
Abc[12][I][i-6]
C[1+i]
```

Комментарии (строчные и блочные)

Блочные комментарии отсутствуют.

Строчные – строка, начинающаяся с символа «C» в первой позиции.

```
C это инкремент
```

Обращения к функциям пользователя

Идентификатор, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Скобки могут быть пустыми в случае отсутствия аргументов:

```
F(12, 4, I)
A3(AV-6)
G()
```

Вызов подпрограмм пользователя

Осуществляется оператором CALL, в котором указывается имя подпрограммы, после которого в круглых скобках следует последовательность выражений-аргументов, разделенных запятыми. Скобки могут быть пустыми в случае отсутствия аргументов:

```
CALL F(12, 4, I)
CALL A3(AV-6)
CALL G()
```

Арифметические операции

Сложение +

Вычитание	-
Умножение	*
Деление	/
Возведение в степень	**

Операции сравнения

Больше	.GT.
Меньше	.LT.
Больше равно	.GE.
Меньше равно	.LE.
Не равно	.NE.
Равно	.EQ.

Оператор присваивания

Имеет вид «=». Слева стоит идентификатор или элемент массива, а справа – выражение, например:

```
A=B+C
B(2,I-9)=12
```

Структура программы

Структура программ является строково-ориентированной. Так, 1-й символ строки служит для маркировки текста как комментария (символом C), с 1-го по 5-й символ располагается область меток, а с 7-го по 72-ой располагается

собственно текст оператора или комментария. Колонки с 73-й по 80-ю

транслятором игнорируются. Если текст оператора не вписывается в отведённое пространство (с 7-й по 72-ю колонку), в 6-ой колонке следующей строки ставится признак продолжения, и затем оператор продолжается на ней.

Располагать два или более оператора в одной строке нельзя.

Программа имеет заголовок вида

```
PROGRAM <ИМЯ ПРОГРАММЫ>
```

Затем следуют операторы описания данных, за ними – исполняемые операторы основной программы, оканчивающиеся операторами

```
STOP
END
```

После основной программы располагаются одна или несколько подпрограмм.

Операторы описания данных (идентификаторов и массивов)

Описание переменных с ключевого слова типа и содержит перечисление идентификаторов через запятую

```
<тип> <список элементов>
```

Типы переменных:

INTEGER (целый),

REAL (вещественный),

CHARACTER (символьный).

Например,

```
REAL A,B
```

Для описания массивов используется оператор DIMENSION, в котором указывается его имя и список размерностей в круглых скобках через запятую:

```
DIMENSION <имя массива> (размерность)
```

Например:

```
DIMENSION MASSIVE(A1,…,An)
```

Операторы описания функций

Функции имеют заголовок вида

```
<тип> FUNCTION <идентификатор>
PARAMETER <список формальных параметров>
```

и тело – список операторов, начинающийся операторами описания данных и оканчивающийся операторами

```
RETURN
END
```

Операторы описания подпрограмм

Подпрограммы имеют заголовок вида

```
SUBROUTINE <идентификатор>
PARAMETER <список формальных параметров>
```

и тело – список операторов, начинающийся операторами описания данных и оканчивающийся операторами

```
RETURN
END
```

Оператор безусловного перехода и метки

```
GO TO <метка>
```

Метка - число, расположенное в области с 1 по 5 символ строки:

```
123 I=25
```

…

```
GOTO 123
```

Оператор условного перехода

Начинается с ключевого слова «IF», имеет только неполный формат:

```
IF (логическое выражение) оператор
```

Список литературы