МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению лабораторных работ
по дисциплине «ЛИНГВИСТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ»
для студентов направления подготовки 09.03.03 Прикладная информатика

(ЭЛЕКТРОННЫЙ ДОКУМЕНТ)

Ставрополь 2026

Содержание

Порядок выполнения лабораторных работ и требования к их оформлению

Цикл состоит из восьми лабораторных работ, которые можно объединить в четыре блока:

– лабораторные работы 1, 2: построение программы лексического анализатора, корректно распознающего все лексемы, перечисленные в разделе «Варианты индивидуального задания», и формирующей таблицы, описанные в Л/Р 2, и внутреннее представление проанализированного текста;

– лабораторные работы 3, 4, 5, 6: построение программы для перевода закодированного текста исходной программы в обратную польскую запись;

– лабораторная работа 7: разработка программы для формирования по обратной польской записи текста на выходном (или машинном) языке;

– лабораторная работа 8: разработка синтаксического анализатора исходного текста.

Результаты работы каждой, из программ должны быть выведены в соответствующие файлы.

Отчет по каждой лабораторной работе включает в себя теоретические разработки вопроса проектирования, программную реализацию (распечатку текста программы) и пример работы программы.

Варианты заданий

Каждый вариант задания представляет собой пару: входной язык и машинный или выходной язык. Варианты заданий представлены в конце раздела.

В качестве входного языка может быть взят один из языков программирования высокого уровня, в котором должно быть рассмотрено следующее подмножество языковых функций 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);

– 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 (если предусмотрены в языке):

– операторы условного я безусловного перехода;

– метки (если предусмотрены в языке). В качестве машинного языка может быть взят как любой язык высокого уровня, так и язык Ассемблера.

Варианты заданий:

1. Паскаль / Си
2. Бейсик / Си
3. Фортран / Си
4. ПЛ/1 / Си
5. Модула / Си
6. Паскаль / Фортран
7. Бейсик / Фортран
8. Си / Фортран
9. ПЛ/1 / Фортран
10. Модула / Фортран
11. Паскаль / ПЛ/1
12. Бейсик / ПЛ/1
13. Фортран / ПЛ/1
14. Си / ПЛ/1
15. Модула / ПЛ/1

16. Паскаль / Бейсик
17. Си / Бейсик
18. Фортран / Бейсик
19. ПЛ/1 / Бейсик
20. Модула / Бейсик
21. Паскаль / Модула
22. Бейсик / Модула
23. Фортран / Модула
24. ПЛ/1 / Модула
25. СИ / Модула
26. Паскаль / Ассемблер
27. Си / Ассемблер
28. Бейсик / Ассемблер
29. Фортран / Ассемблер
30. ПЛ/1 / Ассемблер
31. Модула / Ассемблер
32. Си / Паскаль
33. Бейсик / Паскаль
34. Фортран / Паскаль
35. ПЛ/1 / Паскаль
36. Модула / Паскаль

ЛАБОРАТОРНАЯ РАБОТА 1

Лексический анализ текста исходной программы.

1. Цель и содержание

Цель лабораторной работы: изучение основных понятий теории регулярных грамматик, ознакомление с назначением и принципами работы лексических анализаторов (сканеров).

2. Теоретическое обоснование

Лексический анализатор (или сканер) – это часть компилятора, которая читает литеры программы на исходном языке и строит из них слова (лексемы) исходного языка. На вход лексического анализатора поступает текст исходной программы, а выходная информация передается для дальнейшей обработки компилятором на этапе синтаксического анализа и разбора.

С теоретической точки зрения лексический анализатор не является обязательной, необходимой частью компилятора. Его функции могут выполняться на этапе синтаксического разбора. Однако существует несколько причин, исходя из которых в состав практически всех компиляторов включают лексический анализ. Эти причины заключаются в следующем:

– упрощается работа с текстом исходной программы на этапе синтаксического разбора и сокращается объем обрабатываемой информации, так как лексический анализатор структурирует поступающий на вход исходный текст программы и выкидывает всю незначащую информацию;

– для выделения в тексте и разбора лексем возможно применять простую, эффективную и теоретически хорошо проработанную технику

анализа, в то время как на этапе синтаксического анализа конструкций исходного языка используются достаточно сложные алгоритмы разбора;

– сканер отделяет сложный по конструкции синтаксический анализатор от работы непосредственно с текстом исходный программы, структура которого может варьироваться в зависимости от версии входного языка – при такой конструкции компилятора при переходе от одной версии языка к другой достаточно только перестроить относительно простой сканер.

Функции, выполняемые лексическим анализатором, и состав лексем, которые он выделяет в тексте исходной программы, могут меняться в зависимости от версии компилятора. В основном лексические анализаторы выполняют исключение из текста исходной программы комментариев и незначащих пробелов, а также выделение лексем следующих типов: идентификаторов, строковых, символьных и числовых констант, ключевых (служебных) слов входного языка.

В простейшем случае фазы лексического и синтаксического анализа могут выполняться компилятором последовательно. Но для многих языков программирования информации на этапе лексического анализа может быть недостаточно для однозначного определения типа и границ очередной лексемы. Иллюстрацией такого случая может служить пример оператора программы на языке Фортран, когда по части текста DO 10 I=1... невозможно определить тип оператора (а соответственно, и границы лексем). В случае DO 10 I=1.15 – это будет присвоение вещественной переменной DO10I значения константы 1.15 (пробелы в Фортране игнорируются), а в случае DO 10 I=1,15 – это цикл с перечислением от 1 до 15 по целочисленной переменной I до метки 10.

В большинстве компиляторов лексический и синтаксический анализаторы – это взаимосвязанные части. Лексический разбор исходного текста в таком варианте выполняется поэтапно так, что синтаксический анализатор, выполнив разбор очередной конструкции языка, обращается к сканеру за следующей лексемой. При этом он может сообщить информацию

о том, какую лексему следует ожидать. В процессе разбора может даже происходить «откат назад», чтобы выполнить анализ текста на другой основе. В дальнейшем будем исходить из предположения, что все лексемы могут быть однозначно выделены сканером на этапе лексического разбора.

Вид представления информации после выполнения лексического анализа целиком зависит конструкции компилятора. Но в общем виде ее можно представить как таблицу лексем, которая в каждой строчке должна содержать информацию о виде лексемы, ее типе и, возможно, значении. Обычно такая таблица имеет два столбца: первый – строка лексемы, второй – указатель на информацию о лексеме, может быть включен и третий столбец – тип лексем.

Лексический анализатор имеет дело с таким объектами, как различного рода константы и идентификаторы (к последним относятся и ключевые слова). Язык констант и идентификаторов в большинстве случаев является регулярным – то есть, может быть описан с помощью регулярных (праволинейных или леволинейных) грамматик. Распознавателями для регулярных языков являются конечные автоматы. Существуют правила, с помощью которых для любой регулярной грамматики может быть построен недетерминированный конечный автомат, распознающий цепочки языка, заданного этой грамматикой.

Недетерминированный конечный автомат задается пятеркой:

$\mathbf{M}=(\mathbf{Q}, \mathbf{\Sigma}, \delta, q_0, \mathbf{F})$,

где:

Q – конечное множество состояний автомата;

Σ – конечное множество допустимых входных символов;

δ – заданное отображение множества $\mathbf{Q}^*\mathbf{\Sigma}$ во множество подмножеств $\mathbf{P}(\mathbf{Q})$ δ: $\mathbf{Q}^*\mathbf{\Sigma}\rightarrow\mathbf{P}(\mathbf{Q})$ (иногда δ называют функцией переходов автомата);

$q_0 \in \mathbf{Q}$ – начальное состояние автомата;

$\mathbf{F} \subseteq \mathbf{Q}$ – множество заключительных состояний автомата.

Работа автомата выполняется по тактам. На каждом очередном такте i автомат, находясь в некотором состоянии $q_i \in \mathbf{Q}$, считывает очередной символ $w \in \mathbf{\Sigma}$ из входной цепочки символов и изменяет свое состояние на $q_{i+1}=\delta(q_i, w)$, после чего указатель в цепочке входных символов передвигается на следующий символ и начинается такт i+1. Так продолжается до тех пор, пока цепочка входных символов не закончится. Конец цепочки символов часто помечают особым символом $\perp$. Считается также, что перед тактом 1 автомат находится в начальном состоянии q_0.

Говорят, что автомат допускает цепочку $\alpha \in \mathbf{\Sigma}^*$, если в результате выполнения всех тактов работы над этой цепочкой он окажется в состоянии $q \in \mathbf{F}$. Язык, определяемый автоматом, является множеством всех цепочек, допускаемых автоматом. Для анализа цепочки с помощью конечного автомата достаточно подать ее на вход автомата, выполнить все такты его работы и определить, перешел ли автомат в результате работы в одно из заданных конечных состояний.

Графически автомат отображается нагруженным однонаправленным графом, в котором вершины представляют состояния, дуги отображают переходы из одного состояния в другое, а символы нагрузки (пометки) дуг соответствуют функции перехода. Если функция перехода предусматривает переход из состояния q в q’ по нескольким символам, то между ними строится одна дуга, которая помечается всеми символами, по которым происходит переход из q в q’.

Недетерминированный автомат неудобен для анализа цепочек, так как в нем могут встречаться состояния, допускающие неоднозначность, т.е. такие, из которых выходит две или более дуги, помеченных одним и тем же символом. Очевидно, что программирование работы такого автомата – нетривиальная задача.

Доказано, что любой недетерминированный автомат может быть преобразован в детерминированный так, чтобы их языки совпадали (говорят, что автоматы эквивалентны).

Детерминированный конечный автомат задается пятеркой:

M'=(**Q'**, **Σ**, δ', q_0', **F'**),

где:

Q' – конечное множество состояний автомата;

Σ – конечное множество допустимых входных символов автомата;

$q_0' \in$ **Q'** – начальное состояние автомата;

δ' – заданное отображение множества **Q'*Σ** во множество **Q'** δ: **Q'*Σ→Q'**;

F⊆Q' – множество заключительных состояний автомата.

После построения конечный детерминированный автомат может быть минимизирован, т.е. для него может быть построен эквивалентный ему автомат с минимальным числом состояний.

Можно написать функцию, отражающую функционирование любого детерминированного конечного автомата. Чтобы запрограммировать такую функцию, достаточно иметь переменную, которая бы отображала текущее состояние автомата, а переходы автомата из одного состояния в другое на основе символов входной цепочки могут быть построены с помощью операторов выбора. Работа функции должна продолжаться до тех пор, пока не будет достигнут конец входной цепочки. Для вычисления результата функции необходимо по ее завершению проанализировать состояние автомата. Если это одно из конечных состояний, то функция выполнена успешно, и входная цепочка принимается, если нет – то входная цепочка не принадлежит заданному языку.

Рассмотрим пример анализа лексем, представляющих собой целочисленные константы без знака в формате языка Си. В соответствии с требованиями языка, такие константы могут быть десятичными,

восьмеричными, либо шестнадцатеричными. Восьмеричной константой считается число, начинающееся с 0 и содержащее цифры от 0 до 7; шестнадцатеричная константа должна начинаться с последовательности символов 0x и может содержать цифры и буквы от A до F (будем рассматривать только прописные буквы). Остальные числа считаются десятичными (правила их записи напоминать, наверное, не стоит). Будем

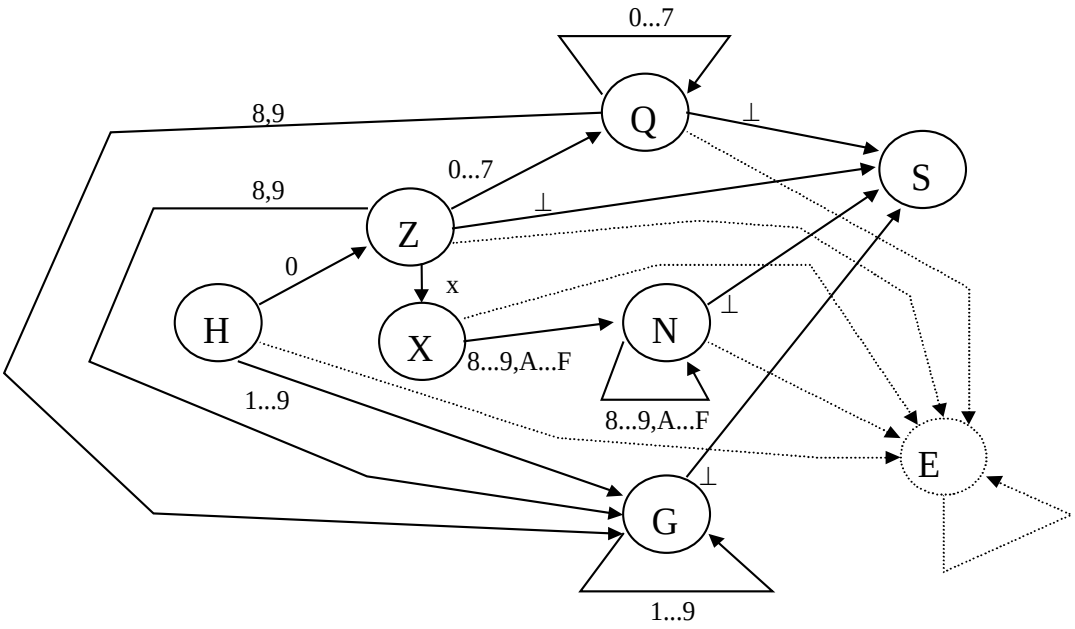

Рисунок 1.1. Граф конечного детерминированного автомата, распознающего грамматику целых чисел языка Си.

считать, что всякое число завершается символом конца строки ⊥.

Рассмотренные выше правила могут быть записаны в форме Бэкуса-Наура в грамматике целочисленных констант без знака языка Си (для избегания путаницы терминальные символы грамматики выделены жирным шрифтом).

G({**0**...**9**,**A**...**F,** ⊥},{S,G,X,N,Q,Z},P,S)

P: S→ G⊥|Z⊥|N⊥|Q⊥

G→ **1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**|G**0**|G**1**|G**2**|G**3**|G**4**|G**5**|G**6**|G**7**|G**8**|G**9**|Z**8**|Z**9**|Q**8**|Q**9**

H→ X**0**|X**1**|X**2**|X**3**|X**4**|X**5**|X**6**|X**7**|X**8**|X**9**|X**A**|X**B**|X**C**|X**D**|X**E**|X**F**|

N**0**|N**1**|N**2**|N**3**|N**4**|N**5**|N**6**|N**7**|N**8**|N**9**|N**A**|N**B**|N**C**|N**D**|N**E**|N**F**

X→ Z**x**

Q→ Z**0**|Z**1**|Z**2**|Z**3**|Z**4**|Z**5**|Z**6**|Z**7**|Q**0**|Q**1**|Q**2**|Q**3**|Q**4**|Q**5**|Q**6**|Q**7**

Z→ **0**

Хорошо видно, что данная грамматика является регулярной грамматикой (леволинейной). Конечный детерминированный автомат **M'**({N, Z, X, N, Q, G, S, ER}, {0...9, A...F, ⊥}, δ, H, {S}), который распознает язык, заданный этой грамматикой, изображен на рисунке 1.1. Начальным состоянием автомата является состояние *H*. В автомат дополнительно введено особое состояние *E*, обозначающее ошибку в распознавании цепочки символов. На графе автомата дуги, идущие в это состояние, не нагружены символами. По принятому соглашению они обозначают функцию перехода по любому символу, кроме тех символов, которыми уже помечены другие дуги, выходящие из той же вершины графа. Такое соглашение принято, чтобы не загромождать обозначениями граф автомата (на рисунке 1.1 такие дуги и состояние *E* выделены пунктиром).

Можно написать программу, моделирующую работу указанного автомата. Ниже приводится текст функции на языке Паскаль, его реализующей. Результат функции истинный (*True*), если входная цепочка символов принадлежит входному языку автомата. Границей цепочки считается символ с кодом 0 (*#0*), в функции он искусственно добавляется в конец цепочки.

В программе переменная iState отображает текущее состояние автомата, переменная i является счетчиком символов входной строки. Конечно, рассмотренная программа может быть оптимизирована (например, можно сразу же прекращать разбор по обнаружению ошибки), но в данном примере оптимизация не выполнялась, чтобы можно было четко отследить соответствие между программой и построенным автоматом.

```
type
```

```
  TAutoState = ( AUTO_H, AUTO_Z, AUTO_X,
                                 AUTO_Q, AUTO_N, AUTO_G,
                                 AUTO_ER, AUTO_S );

function RunAuto (sInput: string): Boolean;
var
  iState : TAutoState;
  i : integer;
begin
  sInput := sInput + #0;
  iState := AUTO_H;
  i := 0;
  repeat
           i := i + 1;
         case iState of
         AUTO_H:
                case sInput[i] of
                       '0':            iState := AUTO_Z;
                       '1'..'9':       iState := AUTO_G;
                       else    iState := AUTO_E;
                end;
         AUTO_Z:
                case sInput[i] of
                       '0'..'7':       iState := AUTO_Q;
                       '8','9':        iState := AUTO_G;
                       'x':            iState := AUTO_X;
                       #0:             iState := AUTO_S;
                       else    iState := AUTO_E;
                end;

         AUTO_X:
                case sInput[i] of
                       '0'..'9':       iState := AUTO_N;
                       'A'..'F':       iState := AUTO_N;
                       else    iState := AUTO_E;
                end;
         AUTO_Q:
                case sInput[i] of
                       '0'..'7':       iState := AUTO_Q;
                       '8','9':        iState := AUTO_G;
                       #0:             iState := AUTO_S;
                       else    iState := AUTO_E;
                end;
         AUTO_N:
                case sInput[i] of
                       '0'..'9':       iState := AUTO_N;
                       'A'..'F':       iState := AUTO_N;
                       #0:             iState := AUTO_S;
                       else    iState := AUTO_E;
                end;
         AUTO_G:
                case sInput[i] of
                       '0'..'9':       iState := AUTO_G;
                       #0:             iState := AUTO_S;
                       else    iState := AUTO_E;
                end;
         AUTO_E:         iState := AUTO_E;
         end {case};
  until (sInput[i] = #0);
  RunAuto := (iState = AUTO_S);
end; { RunAuto }
```

Однако в общем случае задача сканера несколько шире, чем просто проверка цепочки символов лексемы на соответствие ее входному языку. Сканер должен выполнить те или иные действия по запоминанию распознанной лексемы (занесение ее в таблицу лексем). Набор действий определяется реализацией компилятора. Обычно эти действия выполняются сразу же по обнаружению конца распознаваемой лексемы, поэтому их несложно вставить в соответствующие места рассмотренной выше программы-сканера (в те операторы, где обнаруживается символ #0).

Вторая проблема, которая уже обсуждалась выше, это выделение границ лексем. Ведь во входном тексте лексемы не ограничены специальными символами. Если говорить в терминах программы-сканера, то определение границ лексем – это выделение тех строк в общем потоке входных символов, для которых надо выполнять распознавание. В общем случае эта задача может быть сложной, но для простейших входных языков границы лексем распознаются по заданным терминальным символам. Эти символы – пробелы, знаки операций, символы комментариев, а также разделители (запятые, точки с запятой и др.). Набор таких терминальных символов может варьироваться в зависимости от входного языка. Важно отметить, что знаки операций сами также являются лексемами, и необходимо не пропустить их при распознавании текста.

Таким образом, алгоритм работы простейшего сканера можно описать так:

– просматривается входной поток символов программы на исходном языке до обнаружения очередного символа, ограничивающего лексему;

– для выбранной части входного потока выполняется функция распознавания лексемы;

– при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем, и алгоритм возвращается к первому этапу;

– при неуспешном распознавании выдается сообщение об ошибке, а дальнейшие действия зависят от реализации сканера – либо его выполнение

прекращается, либо делается попытка распознать следующую лексему (идет возврат к первому этапу алгоритма).

Работа программы-сканера продолжается до тех пор, пока не будут просмотрены все символы программы на исходном языке из входного потока.

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Содержание отчета и его форма

1. Номер и название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

6. Вопросы для защиты работы

1. Дайте определение термину «Лексический анализатор».

2. Перечислите причины исходя из которых в состав практически всех компиляторов включают лексический анализ.

3. Опишите алгоритм работы простейшего автомата.

4. Опишите структуру и работу недетерминированного конечного автомата.

7. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 2

Построение лексического анализатора.

1. Цель и содержание

Цель лабораторной работы: изучение основных понятий теории регулярных грамматик, ознакомление с назначением и принципами работы лексических анализаторов (сканеров), получение практических навыков построения сканера на примере выбранного языка.

2. Теоретическое обоснование

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 по двум причинам:

– для сокращения времени компиляции;

– синтаксис лексем всегда проще, чей синтаксис программ.

Лексемой называется некоторая конструкция, выступающая как терминальный символ для синтаксического анализа. Лексемы еще иногда называют символами или атомами. Поскольку при генерации машинных команд нужны конкретные значения идентификаторов отдельных лексем, то можно дать следующее определение лексемы.

Лексема – это пара (класс, значение). Значение может быть непосредственным или представлять собой ссылку на некоторую таблицу, в которой хранятся значения лексем. Большинство лексем в языках программирования можно сгруппировать в следующие классы:

– идентификаторы;

– служебные слова;

– числа;

– однолитерные разделители;

– двулитерные разделители.

Классы лексем могут быть описаны простыми правилами, которые либо являются автоматными, либо могут быть приведены к автоматным. Например:

<идентиф>::= буква | <идентиф>буква | <идентиф>цифра

<число>::= <целое> | .<целое> | <целое>. | <целое>.<целое>

<целое>::= цифра | <целое>цифра

<разделитель>::= / | // | /* | . | ;

Во внутреннем представлении все лексемы имеют вид пары, в которой первый знак – 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 ПЛ/1. Определим входной язык.

1. Описание данных: ограничимся только одним типом число выходных данных DEC FIXED, то есть вещественное число с фиксированной точкой:

2. Описание процедур;

3. Операторы :

– присваивания = ;

– безусловного перехода GOTO:;

– условный логический оператор IF THEN;

– оператор конца процедуры END.

4. Выражения:

– операции сложения + ;

– умножения * ;

– отношений <,> ;

– скобки (,) .

5. Метки вида <идентиф>: ;

6. Разделители: , ; .

Построим таблицу классов лексем (таблица 2.1).

Таблица 2.1 – Таблица классов лексем

Типы	Обозн
Служебные	W
Идентифик	I
Операторы	0
Разделител	R
Числа	N

Лексический анализатор (сканер) должен выделять лексемы из исходной программы и передавать их синтаксическому анализатору во внутреннем представлении. При этом используются таблицы, соответствующие каждому классу лексем. Таблицы служебных слов, разделителей, операций определяются входным языком и должны быть сформированы при построении сканера, а таблицы идентификаторов и чисел заполняются в процессе разбора исходной программы.

Для рассматриваемого языка сформированные таблицы служебных слов (табл. 2.2), таблица операций (табл. 2.3) и таблица разделителей (табл. 2.4) имеют вид:

Таблица 2.2 – Таблица служебных слов

Служебное слово	од
DEC	
DCL	
END	
FIXED	
GOTO	
IF	
PROG	
THEM	

MAIN	

Таблица 2.3 – Таблица операций

Операция	код
+	
*	
<	
>	
=	
:	

Таблица 2.4 – Таблица разделителей

Разделитель	код
Пробел	
,	
:	
(	
)	
.	

Каждая лексема, обработанная сканером, преобразуется к виду:

<буква><код>, где:

<буква> – это признак класса лексемы (W, I, 0, R, или N),

<код> – код лексемы в соответствующей таблице.

Например, если на сканер поступила цепочка вида:

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

то сканер выдаст следующую последовательность:

W6 – служебное слово IF,

I1 – идентификатор C,

04 – операция >,

N1 – число 2.79,

W8 – служебное слово THEN

I2 – идентификатор B,

05 – операция = ,

I3 – идентификатор A,

02 – оператор * ,

I4 – идентификатор E3,

R3 – разделитель .

Рассмотрим работу сканера на следующем примере:

```
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.
```

После работы лексического анализатора программа преобразуется к следующему виду:

PROC MAIN;	W7 W9 R3
DCL (A1,A2) DEC	W2 R4 I1 R2 I2 R5 W1 W4 R3
FIXED;	I1 05 N1 R3
A1=378;	I2 05 N2 R3
A2=.73;	W7 I3 R3
PROC CALC;	W2 R4 I4 R2 I5 R5 W1 W4 R3

DCL SUM. MULT DEC FIXED;	W6 I1 01 I2 04 N3 W8 W5 I6 R3
IF A1+A2>3.2 THEN GOTO P;	I4 05 I2 01 I1 R3
SUM=A1+A2;	I6 06 I5 05 I1 R4 I2 01 I4 R5 R3
P: MULT=A1*(A2+SUM);	W3 R3
END;	W3 R6
END.	

В процессе работы лексического анализатора были сформированы таблица чисел N (табл. 2.5) и таблица идентификаторов I (табл. 2.6) .

Таблица 2.5 – Таблица чисел

	4
	3
	.
	3

Таблица 2.6 – Таблица идентификаторов

од	Идентификатор	Номер процедуры	Уровень процедуры	Номер идентиф. в процедуре	Тип идентиф.	Объем памяти
	A1					
	A2					
	CALC					
	SUM					
	MULT					
	P					

Нарисуем диаграмму состояний сканера:

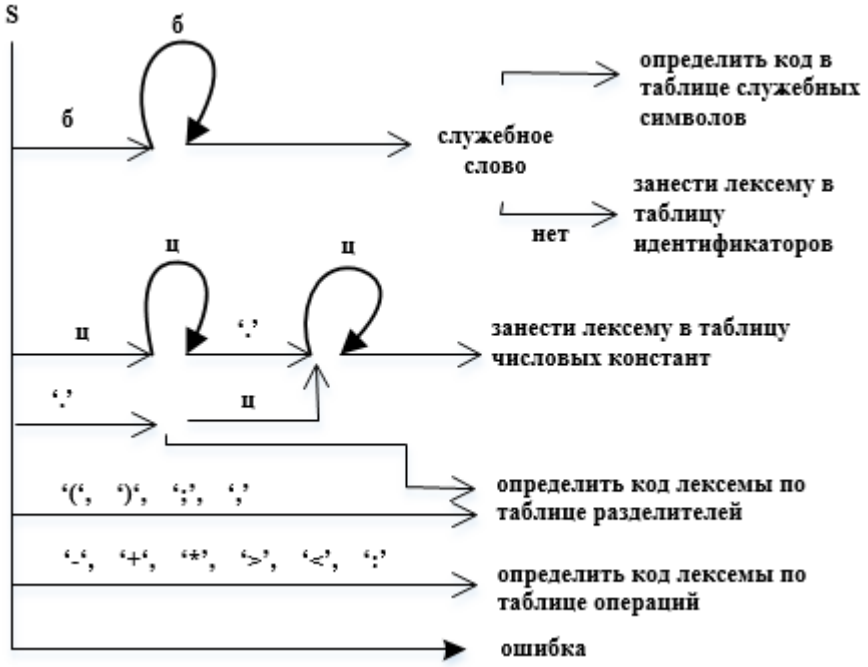

Рисунок 2.1 – Диаграмма состояний сканера

Если в диаграмме состояния есть невзвешенные дуги, то они взвешены любыми символами кроме тех, которыми взвешены другие дуги, исходящие из данного состояния.

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера,

установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Построить программу лексического анализатора, корректно распознающего все лексемы, перечисленные в разделе «Варианты индивидуального задания», и формирующей таблицы, описанные в лабораторной работе, и внутреннее представление проанализированного текста.

2. Результат работы разработанной программы должен сохраняться в файл.

3. Данный файл является файлом со входной информацией для следующей лабораторной работы.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Что такое лексический анализ?

2. Дайте определение термину «лексема».

3. Перечислите классы лексем.

4. Каково назначение лексического анализатора?

5. Опишите диаграмму состояний сканера.

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 3

Перевод исходной программы в обратную польскую запись. Перевод выражений, содержащих указатели функций исходной программы, в обратную польскую запись.

1. Цель и содержание

Цель лабораторной работы: ознакомиться с обратной польской записью и особенностями перевода выражений, содержащих указатели функций в ОПЗ.

2. Теоретическое обоснование

3.1 Перевод исходной программы в обратную польскую запись

Обратная польская запись (ОПЗ) – представляет собой одну из форм записи выражений и операторов, отличительной особенностью которой является то, что все аргументы (или операнды) расположены перед операцией (оператором).

Например, выражение, записанное в обычной скобочной записи, (a+d)/c+b*(e+d), представленное в ОПЗ имеет вид:

ad+c/bed+*+.

Обратная польская запись получила широкое распространение благодаря своему основному преимуществу: выражение (оператор), записанное в виде ОПЗ может быть выполнено за один просмотр цепочки слева направо.

В системном программировании широко применяется обработка цепочек символов, поэтому необходимы алгоритмы получения ОПЗ по принципу обработки строк: преобразование входной строки в выходную, которая является обратной польской записью.

Наиболее известным из таких алгоритмов является алгоритм Дейкстры.

В этом алгоритме используется стек куда помещаются операции. Каждой операции приписывается свой приоритет. В таблице 3.1 приведены приоритеты операций.

Таблица 3.1– Приоритеты операций

Входной	При
(	0
)	1
V	2
&	3
|	4
Отношения	5
+ -	6
* /	7
Возведение в	8

Кроме этого, определяются правила работы со стеком. Входная строка просматривается слева направо. Если рассматриваемый символ является операндом, то он переносится в выходную строку, иначе обрабатывается стеком следующим образом:

– 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 приоритет, то из стека выталкиваются все элементы (операции) до тех пор, пока не встретится операция с приоритетом выше, чем у рассматриваемой операции, после чего эта операция проталкивается в стек;

– если входной символ "(", то он проталкивается в стек. Если входной символ ")", то он выталкивает из стека все знаки до ближайшей "(". Сами скобки взаимно уничтожаются и в выходную строку не попадают.

Приведем пример перевода выражения в ОПЗ:

Таблица 3.2 – Перевод выражений, содержащих переменные с индексами в ОПЗ

Выходная строка	a b + 5 - < 2 c – I q + = &
С Т Е К	+ < - & - = + < & & = &
Вход строка	a + b < - 5 & 2 – c = 1 + q | --

В языках программирования могут использоваться массивы и их элементы, то есть переменные с индексами. Массивы могут быть как одномерными, так и многомерными. Реально в памяти ЭВМ могут быть представлены только одномерные массивы и поэтому возникает проблема: как отобразить многомерный массив в одномерный и как организовать доступ к соответствующему элементу многомерного массива.

Для того, чтобы решить эту проблему вводится операция вычисления адреса элемента массива (АЭМ), которая выполнит следующие действия:

– по имени массива отыскивается определяющий вектор;

– функция упорядочивания вычисляет адрес соответствующего элемента, упорядоченного массив на основе идентификатора и определявшего вектора.

Дополним понятие обратной польской записи операцией определения адреса элемента массива (АЭМ), Операция АЭМ имеет к операндов:

– имя массива;

– индексы массива. В ОПЗ операция АЭМ имеет вид:

<операнд1><операнд2> ... <операндk> К АЭМ, где:

<операнд1> – имя массива, а <операнд2>,...<операндk> – индексы массива.

Например, ОПЗ выражения

(a + b [1 + 20, 1]) * c + d

будет иметь вид:

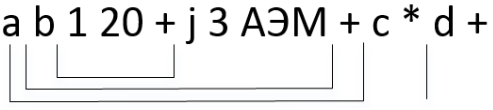

Рисунок 3.1 – Представление выражения в ОПЗ

Для того, чтобы реализовать для данного случая алгоритм Дейкстры необходимо ввести операцию АЭМ в таблицу приоритетов (табл. 3.3)

Таблица 3.3 – Таблица приоритетов

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

Функционирование алгоритма Дейкстры при переводе выражений с индексами в ОПЗ имеет свои особенности:

– обычные знаки операций обрабатывается 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 знаки до ближайшего АЭМ и наращивает счетчик операндов на 1;

– закрывающая скобка] выталкивает из списка операцию АЭМ со значением счетчика операндов, но сама в выходную строку не попадает.

Например, для рассмотренного выражения перевод в ОПЗ имеет вид:

Вых. строка	a b 1 20 + j 3 АЭМ + c * d +
C	(+ 2АЭМ + 3АЭМ + * +

Т Е К	(+ 2АЭМ + ((+ ((
Вход. Строка	(a + b [I + 20, j]) * c + d |--

3.2 Перевод выражений, содержащих указатели функций исходной программы, в обратную польскую запись.

В арифметических и логических выражениях могут использоваться функции, которые сами являются подпрограммами. Поиск требуемой функции реализуется следующим образом: обычно заводится таблица имен функций и в этой таблице указываются адреса, по которым располагаются эти подпрограммы. Обычно подпрограммы вставляются либо перед основным модулем, либо за ним. 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,

<операнд2>... .<операндk> – операнды функции.

Пример. Обратная польская запись выражения:

```
У - f (X, Z, У + 2)
```

имеет вид:

```
У f x z у 2+4 Ф -
```

Таблица приоритетов для алгоритма Дейкстры модифицируется следующим образом (табл.3.4).

При обработке операции ФУНКЦИЯ выполняются следующие действия:

– по входному символу '(', следующему за идентификатором. в стек заносится оператор Ф со значением счетчика, равным 1;

Таблица 3.4 – Модифицированная таблица приоритетов

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

– 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:

Вых. строка	y f x z y 2 + 4Ф –
С Т Е К	– 1Ф 2Ф 3Ф + – – – – 3Ф
Вход. Строка	Y – f (x . z . y + 2) |--

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение:

операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Разработать программу для перевода закодированного текста исходной программы в обратную польскую запись.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Дайте определение термину «обратная польская запись».

2. Назовите наиболее известный алгоритм для формирования ОПЗ?

3. Опишите алгоритм Дейкстры.

4. Каким образом в ОПЗ переводятся многомерные массивы?

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 4

Перевод условных выражений исходной программы в обратную польскую запись.

1. Цель и содержание

Цель лабораторной работы: ознакомиться с обратной польской записью и особенностями перевода условных выражений в ОПЗ.

2. Теоретическое обоснование

Пусть имеется условие вида:

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

Это условие можно записать двумя способами:

– как условное выражение

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

– как условный оператор

```
If x =< 0 the y=a + b else y = 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). Для того, чтобы сформулировать понятие ОПЗ для условных выражений, вводятся динамические деревья. Для динамических деревьев введем ряд понятий:

– узлы, ветви могут помечаться метками;

– введем понятие пустого узла – это такой узел в котором отсутствует операция. Из пустого узла может исходить любое заданное количество ветвей. Пустой узел вводит объединение нескольких последовательных действий или разветвление некоторого вычислительного процесса;

– вводится условный оператор по значению 'ложь', который имеет следующее представление:

<условие> <метка> УПЛ

– вводится операция безусловного перехода:

<метка> БП. Например, ОПЗ выражения

y + (IF a > b THEN x + У ELSE x + 2)

имеет вид:

y a b > Ml УПЛ X 1 + M2 БП Ml: X 2 + M2: +

Для того, чтобы не нарушать концепции Дейкстры внесем введенные выше операции в таблицу приоритетов. Очевидно, что IF играет роль открывающей скобки, а THEN и ELSE – запятых. Модифицированная таблица приоритетов выглядит следующим образом (табл. 4.1).

Обработка ключевых слов при переводе в ОПЗ производится так:

– при входном символе IF он заносится в стек;

– появление THEN выталкивает в выходную строку все операции из стека до ближайшего IF. К оператору IF добавляется рабочая метка M_i и после этого в выходную строку записывается часть M_i УПЛ;

– по прочтению ELSE из стека выталкиваются все символы до ближайшего IF M_i. IF M_i превращается в IF M_{i+1} и в выходную строку выталкивается часть M_{i+1} БП M_i:;

– по прочтению символа), указывающего на конец условного выражения, из стека выталкиваются все знаки до ближайшей (при этом сам IF уничтожается, а в выходную строку помещается M_{i+1}

Таблица 4.1 – Таблица приоритетов

Входной	При
IF ([АЭМ Ф	0
,)] THEN	1
V	2
&	3
I	4
отношения .	5

+ -	6
* /	7
возведение в	8

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

Выходная строка	Y a b > M1 УПЛ x 1 + M2 БП M1: x 2 +
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)

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических

розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Разработать программу для перевода условных выражений исходной программы в обратную польскую запись.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Как производится обработка ключевых слов в алгоритме Дейкстры?

2. Дайте определение термину «обратная польская запись».

3. Назовите наиболее известный алгоритм для формирования ОПЗ?

4. Опишите алгоритм Дейкстры.

5. Каким образом в ОПЗ переводятся многомерные массивы?

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 5

Перевод операторов (присваивания, безусловного перехода, условного оператора) исходной программы в обратную польскую запись.

1. Цель и содержание

Цель лабораторной работы: ознакомиться с обратной польской записью и особенностями перевода операторов в ОПЗ.

2. Теоретическое обоснование

2.1 Перевод оператора присваивания в ОПЗ

ОПЗ оператора присваивания

<переменная> := <выражение> имеет вид:

<переменная> <выражение> :=

Основным вопросом по переводу оператора присваивания в ОПЗ становится вопрос о приоритете этого оператора. При выборе приоритета будем руководствоваться следующими соображениями:

– приоритет должен быть старше, чем у любой арифметической или логической операции (чтобы знаки не могли вытолкнуть оператор присваивания из стека);

– приоритет должен быть меньше, чем у знака ';'. Исходя из этого, таблица приоритетов принимает следующий вид (табл. 5.1).

Таблица 5.1 – Таблица приоритетов

Входной	При
IF ([АЭН Ф	0
. ;)] THEN	1
:=	2
V	3
&	4
I	5

отношения	6
+ -	7
* /	8
возведение в	9

2.2 Перевод оператора безусловного перехода и меток в ОПЗ

ОПЗ оператора безусловного прохода

GOTO <нетка> имеет вид: <метка> БП

Приоритет операции устанавливается исходя из таких же соображений, как и в случае оператора присваивания.

Для обработки метки в ОПЗ вводится операция метки, которую для простоты обозначим символом ':'. Эта операция имеет наименьший приоритет. Таблица приоритетов принимает вид (табл.5.2)

Таблица 5.2 – Таблица приоритетов

Входной	При
IF ([АЭМ Ф	0
. ;)] THEN	1
. := GOTO	2
V	3
&	4
I	5
отношения	6
+ -	7
* /	8
возведение в	9 10

Например, пусть имеется фрагмент программы:

R1 : ° A + B :

GOTO Ml ;

R2 : = C * D - A;

Его перевод в ОПЗ по алгоритму Дейкстры будет осуществляться следующим образом:

Выходная строка	R1 A B + := M1 БП R2 C D * A - :=

С Т Е К	:= + GOTO := * - := := :=
Входная строка	R1 := A + B; GOTO M1; R2 := C * D – A ;

2.6. Перевод условного оператора в ОПЗ

Существует два вида условного оператора: полный

IF условие THEN оператор1 ELSE оператор2 :

и неполный

IF условие THEN оператор ;

ОПЗ условного оператора имеет вид:

– для полного условного оператора

условие M1 УПЛ оператор! M2 БП Ml: оператор2 M2:

– для неполного оператора

условие M1 УПЛ оператор M:

В общем случае трансляция условного оператора совпадает с обработкой условного выражения. В разных языках в условных операторах принято различное употребление ограничителей ':' в условном операторе. В этом случае функции ';' как концевого маркера оператора являются в разных языках различными. В Паскале ':' всегда закрывает условный оператор. В основном она должна вытолкнуть из стека IF, расставить соответствующие метки, при этом сам IF в выходную строку не попадает.

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не

менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Разработать программу для перевода операторов (присваивания, безусловного перехода, условного оператора) исходной программы в обратную польскую запись.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Дайте определение термину «обратная польская запись».

2. Назовите наиболее известный алгоритм для формирования ОПЗ?

3. Опишите алгоритм Дейкстры.

4. Каким образом в ОПЗ переводятся многомерные массивы?

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 6

Перевод описаний переменных и процедурных блоков в ОПЗ.

1. Цель и содержание

Цель лабораторной работы: ознакомиться с процедурой перевода описаний переменных и процедурных блоков в ОПЗ.

2. Теоретическое обоснование

Обработка описаний переменных и производится для следующих целей:

– выявление типов данных;

– подготовка информации для распределения памяти;

– выделение в объектной программе места для размещения данных.

Очевидно, что распределение памяти должно предшествовать генерации объектной программы. Поскольку процедуры могут быть вложенными и область действия имен может быть локализована, то необходимо иметь информацию о начале и конце процедуры.

Рассмотрим обработку описания данных на примере языка PL/1. Имеется несколько типов данных:

DEC FLOAT; DEC FIXED: BIN FIXED; CHAR

DFT ТИП = DFD

BFD CHAR

Выходная строка	2DFD D 1 DFT 1 1 КО F2 2 2 НП
С Т Е К	DCL 1.1.0 DCL 1.1.1 DCL 1.1.0 PROC 2.2

Входная строка	DEC FIXED D DEC FLOAT ; PROC F2 ;

Выходная строка	C D 2 BFD 2 2 KO
C T E K	DCL 2.2.1 ((DCL 2.2.2 DCL 2.2.0 DCL 2.2.1 DCL 2.2.2
Входная строка	DCL (C . D) NIB FIXED ;

Выходная строка	КП КП
C T E K	END END
Входная строка	END ; END ;

2.1 Пример перевода исходной программы в ОПЗ

Рассмотрим пример перевода исходной программы на языке, описанном лабораторной работе 2, в обратную польскую запись.

Прежде всего составим таблицу приоритетов, включив в нее все операции и операторы, имеющиеся во входном языке (табл. 6.1) воспользуемся обозначением операторов и операций, введенным в лабораторной работе 2.

Таблица 6.1 – Операторы и операции входного языка

Приоритет	Входной элемент	Обозначение
0 1	IF (THEN ^ ^)	W6 R4 W8 R6 R2 R3 R5

2	GOTO	W5
	°	05
3	<	03
	/ ••>	04
4	+	01
5	*	02
6	PROC	W7
	END	W3
	DCL	W2
		06

Выходная строка	1 1 НП А1 А2
С Т Е К	PROC 1.1 DCL 1.1.1 ((DCL DCL 1.1.1 DCL 1.1.2
Входная строка	PROC MAIN ; CLD (A1 ; B

Выходная строка	2 DFD 1 1 KO A1 378 = A2 .73 = CALC 2 2
С Т Е К	DCL 1.1.0 = = PROC 2.2
Входная строка	DEC FIXED ; A1 = 378 ; A2 = .73; PROC CALC

Выходная строка	SIM MULT 2 BFD 2
С Т Е К	DCL 2.2.1 ((DCL 2.2.2 DCL 2.2.0 DCL 2.2.1 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 M1 GOTO = + IF IF IF M1 =
Входная строка	IF A1 + A2 > 3.2 TJEN GOTO P ; SUM = A2 + A1

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

Для наглядности в примере мы использовали текст исходной программы, но значительно легче использовать текст, полученный в результате лексического анализа, поскольку в нем уже выделены классы лексем, и, к примеру, встретив обозначение идентификатора или константы, можно без дополнительного анализа переслать его в выходную строку.

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Разработать программу для перевода описаний переменных и процедурных блоков исходной программы в обратную польскую запись.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Дайте определение термину «обратная польская запись».

2. Назовите наиболее известный алгоритм для формирования ОПЗ?

3. Опишите алгоритм Дейкстры.

4. Каким образом в ОПЗ переводятся многомерные массивы?

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 7

Перевод ОПЗ исходного выражения в текст на выходном языке.

1. Цель и содержание

Цель лабораторной работы: ознакомиться с процессом перевода выражения, выраженного в ОПЗ, в текст на выходном языке программирования.

2. Теоретическое обоснование

Перевод ОПЗ в текст на выходном (машинном) языке представляет собой второй этап трансляции исходной программы в машинные коды. Для реализации этой процедуры воспользуемся автоматом с магазинной памятью. МП-автомат имеет следующую структуру:

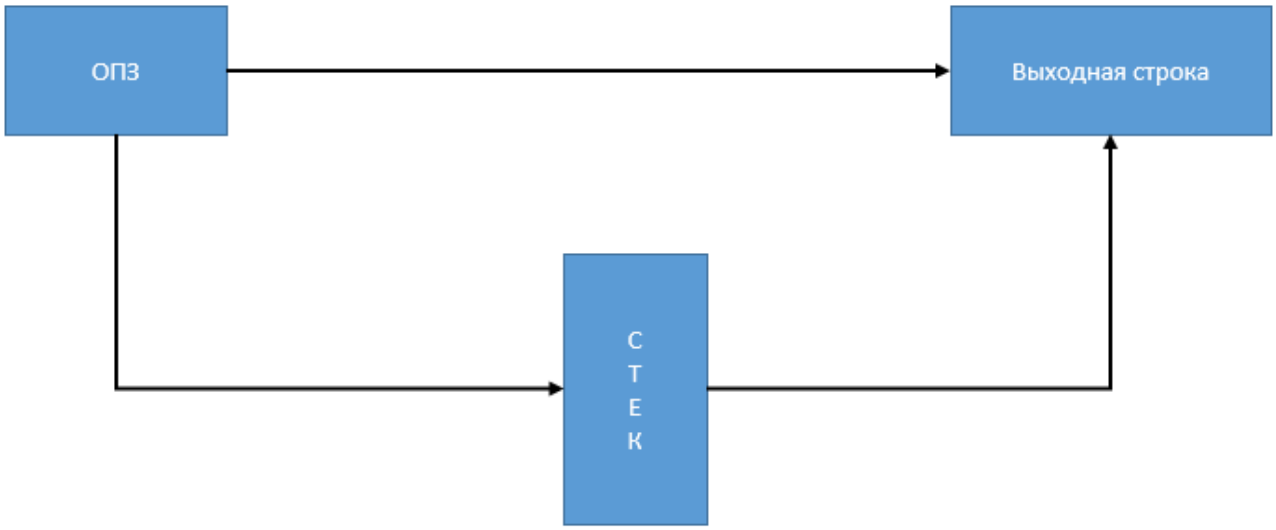

Рисунок 7.1 – Структура МП-автомата

Процесс ОПЗ перевода исходной программы в машинные коды напоминает вычисление по обратной польской записи.

Для работы автомата по переводу ОПЗ в машинные коды нам потребуется 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, нам также необходимо организовать таблицу меток следующей структуры:

Метка	Номер строки

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

2.1 Правила работы 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 – без изменения.

2 Для каждого оператора определяется его арность, т.е. количество операндов, и связанная с ним семантическая процедура.

3 После выполнения каждой семантической процедуры в выходную строку заносится символ <ВК>, счетчик строк наращивается и заносится в качало новой строки. Это правило связано с требованием Бейсика начинать каждый оператор с новой строки.

Семантические процедуры для операторов приведены в таблице 1.

Таблица 7.1 – Семантические процедуры для операторов

Лексема	Действия
НП	Извлечь из стека 2 элемента, занести в выходную строку текст «НЕМ Начало процедуры арг$_2$, арг$_1$»
КП	Занести в выходной строку текст «REM Конец процедуры»
DFD	Извлечь из стека арг$_1$ - число переменных *к*; извлечь из стека *К* аргументов; занести в выходную строку текст «REM Вещественные переменные арг$_i$, арг$_2$, … , арг$_к$»
КО	Извлечь из стека 2 аргумента
УПЛ	Извлечь из стека 2 элемента, занести в выходную строку текст «IF

	NOT(apr_2) THEN GOTO apr_i»
БП	Извлечь из стека один аргумент, занести в выходную строку текст «GOTO $арг_i$»
:	Извлечь из стека 1 аргумент, занести в таблицу $арг_1$ и значение счетчика STR
+ * > <	Извлечь из стека 2 аргумента, нарастить счетчик вспомогательных переменных *P*, занести в выходную строку текст «R_p = $арг_2$ <операция> $арг_1$», занести в стек R_p.
s	Извлечь из стека 2 аргумента и занести в выходную строку текст «$арг_2$ = $арг_1$»

В языке Бейсик символьных меток не существует, поэтому необходимо просмотреть полученный текст и заменить в нем переходы по меткам на переходы по номерам строк (номера строк, соответствующие меткам. хранятся в таблице меток).

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических

розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Разработать программу для формирования по обратной польской записи текста на выходном (или машинном) языке.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.

2. Цели лабораторной работы.

3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Дайте определение термину «обратная польская запись».

2. Назовите наиболее известный алгоритм для формирования ОПЗ?

3. Опишите алгоритм Дейкстры.

4. Каким образом в ОПЗ переводятся многомерные массивы?

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;

2) поясняет ход выполнения индивидуального задания;

3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

ЛАБОРАТОРНАЯ РАБОТА 8

Синтаксический анализ.

1. Цель и содержание

Цель лабораторной работы: ознакомиться с принципом синтаксического анализа текста и алгоритмом его работы.

2. Теоретическое обоснование

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы. Удобным формализмом для описания синтаксических конструкций языка программирования являются контекстно-свободные грамматики.

Грамматики одинаково помогают решать задачи как программистов, использующих язык, так и создателей компиляторов для данного языка:

– Грамматика предоставляет точную и достаточно легкую для понимания синтаксическую спецификацию языка программирования.

– Для некоторых классов грамматик мы можем автоматически сконструировать эффективный анализатор, который определяет, является ли исходная программа синтаксически правильной.

– Аккуратно созданная грамматика может придать языку программирования такую структуру, которая будет полезна и при трансляции исходной программы в правильный объектный код, и при определении ошибок. – Компиляторы, разработанные на базе грамматик, могут быть достаточно легко расширены (это особенно полезно для добавления новых конструкций, появившихся как результат развития языка)

С помощью контекстно-свободных грамматик определяется только так называемая контекстно-свободная составляющая языка программирования, то есть только то, каким образом записывается та или иная конструкция

языка. Другая важная часть определения синтаксической правильности программы – правильность использования типов в программе – не может быть определена с помощью контекстно-свободных грамматик. Поэтому если программа выводима в грамматике, это еще не означает, что она полностью синтаксически правильна.

Синтаксический анализ – это процесс, который определяет, принадлежит ли некоторая последовательность лексем языку, порождаемому грамматикой. В принципе, по любой грамматике можно построить синтаксический анализатор, но грамматики, используемые на практике, имеют специальную форму. Например, известно, что для любой контекстно-свободной грамматики может быть построен анализатор, сложность которого не превышает $O(n^3)$ для входной строки длины n, но в большинстве случаев по заданному языку программирования мы можем построить такую грамматику, которая позволит сконструировать и более быстрый анализатор. Анализаторы реально используемых языков обычно имеют линейную сложность; это достигается, например, за счет просмотра исходной программы слева направо с заглядыванием вперед на один терминальный символ (лексический класс).

Вход синтаксического анализатора – последовательность лексических и таблицы, например, таблица внешних представлений, которые являются выходом лексического анализатора.

Выход синтаксического анализатора – дерево разбора и таблицы, например, таблица идентификаторов и таблица типов, которые являются входом для следующего просмотра компилятора (например, это может быть просмотр, осуществляющий контроль типов).

Для выполнения синтаксического анализа построим грамматику входного языка:

```
<программа> ::=PROC MAIN;<текст> END.
<текст> ::= (<строка> | <Процедура>){<строка>| <процедура>}
```

```
    <строка> ::= <описание>|<отд. оператор> <описание> ::= DCL
<идентификатор>{,<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>

    <буква>::= A | B |... | Y | Z

    <цифра> ::= 0 | 1 |... | 8 | 9
```

2.1 Метод рекурсивного спуска

Одним из наиболее простых и потому одним из наиболее популярных методов нисходящего синтаксического анализа является метод рекурсивного спуска.

Для объяснения принципов, лежащих в основе метода рекурсивного спуска, рассмотрим задачу вычисления значения арифметической формулы. Будем рассматривать формулы, состоящие из целочисленных значений, бинарных операций сложения (+), вычитания (–), умножения (*) и деления нацело (/), а также круглых скобок. Как обычно, приоритеты операций умножения и деления равны и их приоритет больше, чем приоритеты операций сложения и вычитания, причем приоритеты этих операций также равны. Будем называть операции + и – операциями типа сложения, а операции * и / – операциями типа умножения. Круглые скобки используются для изменения стандартного порядка выполнения операций. Наша задача заключается в написании программы, вычисляющей значение формулы.

Изучаемые нами формулы можно представить следующим образом:

T1+T2+…+Tn,

где Ti – это формула вида F1i*F2i*…*Fmi. В свою очередь, Fji – это либо число, либо произвольная формула, заключенная в круглые скобки.

Представим себе процесс вычисления значения формулы. Вначале вычисляется F11, далее мы выясняем, какая операция следует за F_{11}. Если это операция типа умножения, то мы, зная ее левый операнд, вычисляем правый операнд и выполняем операцию. Тем самым, мы получим левый операнд для возможных следующих операций типа умножения. Когда мы закончим вычисление формулы $F_1*F_2*…*F_n$, то, возможно, увидим далее операцию типа сложения, и процесс вычисления такой формулы будет аналогичен только что описанному процессу.

Рассмотрим синтаксический анализ методом рекурсивного спуска применительно к анализу грамматики исходного языка программирования.

В соответствии с методом рекурсивного спуска для каждого нетерминала создается рекурсивная процедура, и каждая такая процедура осуществляет разбор фраз для данного нетерминала. Процедура ищет фразу, сравнивая программу в указанном месте с правыми частями правил для данного нетерминала, вызывая, если нужно, другие процедуры для промежуточных целей.

Для того, чтобы не было возвратов, в качестве контекста используется единственный символ, следующий за разобранной частью фразы. В алгоритме каждому нетерминалу соответствует одна рекурсивная процедура, каждая процедура осуществляет разбор фраз, выводимых из соответствующего нетерминала. Процедура для нетерминала U ищет фразу следующим образом: сравнивает разбираемый символ с правыми частями правил для данного нетерминала и вызывает другие процедуры по мере необходимости. Для того, чтобы работал алгоритм, необходимо определить некоторые понятия.

Введем переменную NXTSYMB – глобальную переменную, содержащую символ, следующий за разбираемым. NXTSYMB содержит символ (лексема) исходной программы, который будет обрабатываться

следующим и при поиске новой цели первый символ, который должен быть исследован, всегда находится в переменной NXTSYMB.

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

Имеется специальная процедура SCAM, которая готовит очередной символ исходной программы и помещает его в NXTSYMB.

Предусмотрена процедура ERROR для обработки ошибочных ситуаций.

Ниже в таблице 1 приведены нетерминалы грамматики и имена соответствующих им рекурсивных процедур.

Процедуры IDENT и CONST в отличие от всех остальных 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.

Таблица 1 – Соответствие нетерминалов грамматики и имен рекурсивных процедур

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

<идентификатор>	IDENT
<константа>	CONST

```
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:
SCAM:
If  NXTSYMB<>'.' then ERROR;
enf.

procedure TEXT;
while NXTSYMB<>'END' do
If NXTSYMB<>'PROF' then PROCED else LINE;
procedure PROCED;
begin
SCAN;
If NOT IDENT (NXT3YMB) then ERROR;
SCAN;
If NXTSYMB<>';' then ERROR;
SCAN;
while NXTSYMBO'END' do begin LINE:
SCAN:
end;
SCAN;
If NXTSYMBO';' then ERROR;
end;
procedure LINE;
If NXTSYMB='DCL' then DECLARE
else SEP_OPER:
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 NXTSYMBO'DEC' then ERROR;
SCAN;
If NXTSYMBO'FIXED' then ERROR;
SCAN;

If NXTSYMBO';' 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;
      procedure OPERATOR;
      If NXTSYHB='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;
      procedure If_OPER;
      begin SCAN;

      CONDITION;
      If NXTSYMB<>'THEN' then ERROR else begin SCAB:
      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 NXTSYNB='*' 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;
      procedure ARGUMENT; If NOT CONST(NXTSYMB) then
      If NOT IDENT(NXTSYMB) then ERROR;
```

3. Аппаратура и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1,5 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система на базе WINDOWS, Microsoft Visual Studio 2008 и старше.

4. Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

5. Индивидуальное задание

1. Разработать программу разработка синтаксического анализатора исходного текста.

6. Содержание отчета и его форма

1. Номер и название лабораторной работы.
2. Цели лабораторной работы.
3. Ответы на контрольные вопросы.

4. Экранные формы, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения:

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

7. Вопросы для защиты работы

1. Дайте определение термину синтаксический анализ?
2. Что является входом для синтаксического анализатора?
3. Опишите метод рекурсивного спуска.
4. Дайте определение термину «контекстно-свободная грамматика».

8. Защита работы

Перед выполнением лабораторной работы каждый студент получает индивидуальное задание.

Защита лабораторной работы происходит только после его выполнения (индивидуального задания). При защите лабораторной работы студент:

1) отвечает на контрольные вопросы;
2) поясняет ход выполнения индивидуального задания;
3) поясняет результаты, полученные в результате выполнения индивидуального задания.

Ход защиты лабораторной работы контролируется преподавателем.

Список использованной литературы

1. Информационные ресурсы и поисковые системы : учебное пособие [Электронный ресурс] / Н.В. Максимов, О.Л. Голицына, Г.В. Тихомиров, П.Б. Храмцов. - М. : МИФИ, 2008. - 400 с. - ISBN 978-5-7262-1049-0. - URL: http://biblioclub.ru/index.php?page=book&id=231125
2. Гетманова, Александра Денисовна. Логика : [учеб. пособие*], 2-е изд. - М. : Академический Проект, 2009. - 712 с. - (Gaudeamus). - На обл.: Учебник, словарь, практикум. - Библиогр.: с. 704-707. - ISBN 978-5- 8291-1148-9.
3. Давыдова, Н.А. Программирование : учебное пособие [Электронный ресурс] / Н.А. Давыдова, Е.В. Боровская. - 2-е изд. (эл.). - М. : БИНОМ. Лаборатория знаний, 2012. - 238 с. - ISBN 978-5-9963-0889-7. - URL: http://biblioclub.ru/index.php?page=book&id=222841
4. Никитин, В.С. Технологии будущего [Электронный ресурс] / В.С. Никитин. - М. : РИЦ "Техносфера", 2010. - 264 с. - ISBN 978-5-94836-256-4. - URL:http://biblioclub.ru/index.php?page=book&id=89015
5. Грекул, В.И. Управление внедрением информационных систем : учебник [Электронный ресурс] / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. - М. : Интернет-Университет Информационных Технологий, 2008. - 224 с. - ISBN 978-5-94774-944-1. - URL: http://biblioclub.ru/index.php?page=book&id=233072