Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Кемеровский государственный университет»
Новокузнецкий институт (филиал)

Факультет информатики, математики и экономики
Кафедра математики, физики и математического моделирования

Е.В. Решетникова

Разработка трансляторов для языков программирования

Методические указания к выполнению лабораторных работ
для обучающихся по направлению подготовки
02.03.03 Математическое обеспечение и администрирование информационных систем,
профиль «Программное и математическое обеспечение информационных технологий»

Новокузнецк

2020

УДК [378.147.88:004.4'42](072)
ББК 74.484(2Рос-4Кем)я73+32.973я73
Р47

Решетникова Е.В.

Р47 Разработка трансляторов для языков программирования: методические указания к выполнению лабораторных работ для студентов факультета информатики, математики и экономики, обучающихся по направлению подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем, профиль «Программное и математическое обеспечение информационных технологий» / Е.В. Решетникова; Новокузнецкий ин-т (фил.) Кемеров. гос. ун-та. – Новокузнецк : НФИ КемГУ, 2020 – 74 с.

Методические указания содержат индивидуальные задания для лабораторных работ и указания к их выполнению; список основной и дополнительной литературы.

Методические указания предназначены для наиболее рациональной организации аудиторной и внеаудиторной самостоятельной работы студентов.

Рекомендовано на заседании
кафедры математики, физики и
математического моделирования
Протокол № 8 от 16.03.2020
Заведующий каф. МФММ
/ Е.В. Решетникова

Утверждено методической комиссией
факультета информатики, математики и
экономики
Протокол № 8 от 18.05.2020
Председатель методической комиссии
ФИМЭ
_____/ Г.Н. Бойченко

УДК [378.147.88:004.4'42](072)
ББК 74.484(2Рос-4Кем)я73+32.973я73
Р47

Содержание

ПРЕДИСЛОВИЕ

Настоящие методические указания адресованы студентам, получающим квалификацию бакалавр по направлению подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем, профиль «Программное и математическое обеспечение информационных технологий» и направлены на оказание помощи студентам в выполнении лабораторных работ по дисциплине «Разработка трансляторов для языков программирования».

В системном программировании имеется огромное разнообразие специализированных языков: это и языки описания интерфейсов, и командные языки операционных систем, и языки описания протоколов коммуникации в компьютерных сетях. Все системные программисты в своей работе сталкиваются с необходимостью конструирования новых языков. Практически всякая большая программная система включает в себя входной язык.

Разработка трансляторов является областью информатики, объединяющей идеи, методы и алгоритмы синтаксического и семантического анализа искусственных языков. Практической целью разработки трансляторов является построение программных систем специфического назначения – языковых процессоров. Эта область базируется на теории формальных языков и главная цель данного курса это дать возможность студентам освоить основные положения этой области. По итогам изучения дисциплины студент сможет построить собственный транслятор даже для нетривиального языка.

В методические рекомендации включено: варианты индивидуальных лабораторных заданий, методические указания к их выполнению, критерии оценки учебной деятельности студента, список основной и дополнительной литературы.

Таким образом, данные методические материалы позволяют студенту подготовиться к лабораторным занятиям по соответствующим темам и успешно выполнить индивидуальные лабораторные задания. Методические указания могут оказаться полезными при написании курсовых и выпускных квалификационных работ.

1 ФОРМАЛЬНЫЕ ГРАММАТИКИ И РАСПОЗНАЮЩИЕ АВТОМАТЫ

1.1. Лабораторная работа № 1 «Программирование конечного автомата – распознавателя»

1.1.1 Задание к лабораторной работе №1

Задание 1. Построить детерминированный конечный автомат распознавателя многострочных комментариев языка С и выполнить его программную реализацию.

Задание 2. Усовершенствовать распознаватель для корректного удаления комментариев языков С (многострочных) и С++ (однострочных).

1.1.2 Методические указания к выполнению лабораторной работы № 1

Задание 1. Распознаватель многострочных комментариев языка С должен, проведя лексический анализ входного файла, содержащего текст программы на языке С, сформировать выходной (новый) файл, содержащий тот же текст, но с удаленными из него комментариями вида /*...*/. Рассматриваются только комментарии языка С и не производится анализ каких-либо иных лексем. Следует учесть, что такие комментарии не могут быть вложенными.

Конечный автомат распознавателя проще всего представить в виде традиционного графа (диаграммы) состояний, в котором состояния представлены вершинами, а переходы - дугами. Условием перехода из одного состояния в другое (событием, входным сигналом) является значение одного очередного символа, считанного из входного потока. При любом переходе из одного состояния в другое может производиться запись в выходной поток как последнего считанного символа, так и произвольного символа или любой цепочки символов. Таким образом, каждая дуга в графе должна быть маркирована и входным символом *a*, и выходной цепочкой

символов γ, т.е. записью вида «*a*/γ». При этом для компактной и наглядной записи выражений допускается:

- указывать допустимые символы входной цепочки в виде множества, например, запись «{*a*,*b*,*c*,0..9}/...» будет означать принадлежность считанного символа указанному множеству, т.е. его равенство одному из элементов этого множества;

- использовать некоторое обозначение или метасимвол для входного символа в записи условия, например, запись «$c \notin \{0..9\}$/…» будет означать, что считанный символ не является десятичной цифрой (является любым другим возможным символом);

- использовать некоторое обозначение для особых (типичных) ситуаций, например, «else», «default» и т.п. для обозначения всех остальных случаев, т.е. образованных исключением из алфавита языка (полного множества возможных входных символов) всех тех символов, которые указаны в условиях перехода всех остальных дуг, исходящих из этого же состояния;

- использовать квантор $\forall$, если переход осуществляется при чтении любого возможного символа (немаркированная дуга означает безусловный переход из одного состояния в другое, при котором чтение очередного символа не производится);

- использовать некоторое обозначение или метасимвол для обозначения конца входной цепочки (при реализации программы - конца файла), например, «EOF» и т.п.;

- использовать некоторое обозначение или метасимвол для обозначения множества пустых символов - символов-разделителей, - таких как пробел, перевод строки, возврат каретки, табуляция и т.п.;

- не указывать выходную цепочку вместе с символом «/» или указывать «... /λ», если при данном переходе ничего не выводится в выходной поток;

- использовать некоторое обозначение для входного символа при его выводе в выходной поток, например: «$c \in \{0..9\} / c$» - символы входного потока, являющиеся цифрами, дублируются в выходной поток; «$\forall c / c$» - дублирование любого считанного символа в выходной поток.

Для построения распознавателя необходим ***детерминированный и полностью определенный*** конечный автомат. Это означает, что в каждом состоянии для каждого возможного входного символа должен быть определен ровно один переход.

Конечный автомат распознавателя будет являться *недетерминированным*, если хотя бы в одном состоянии некоторый считанный символ приводит к одновременному выбору двух или более направлений перехода, что невозможно реализовать без использования параллельных процессов, рекурсии и/или последующего обратного движения по входной цепочке (возврата).

Конечный автомат распознавателя будет являться *определенным не полностью или недоопределенным*, если хотя бы в одном состоянии существует хотя бы один входной символ из множества возможных, для которого не задан переход, что приведет к неразрешимой ситуации при реализации распознавателя. По этой причине необходимо ввести конец файла как входной символ.

После реализации простейшего лексического анализатора в соответствии с Заданием 1 необходимо оценить его адекватность, проведя анализ контрольных примеров.

Задание 2. При выполнении задания в первую очередь следует уточнить диаграмму состояний распознавателя для многострочных и однострочных

комментариев. При этом необходимо учесть все особенности комментариев и некоторых других лексем языка C++, в частности:

- многострочные комментарии не могут быть вложенными;

- многострочный комментарий может использоваться вместо символа-разделителя и его удаление не должно приводить к «склеиванию» лексем и их неправильной интерпретации транслятором;

- окончанию «*/» многострочного комментария может предшествовать символ «*» в любом количестве;

- комментарии не могут являться частью строковой или символьной константы (начинаться или заканчиваться внутри нее - комментарии не должны распознаваться внутри строковых и символьных констант, а строковые и символьные константы не должны распознаваться внутри комментариев);

- строковые и символьные константы могут включать в себя как символы одинарную и двойную кавычки в виде последовательностей «\'» и «\"», которые не являются завершением записи константы, а также любые управляющие последовательности, запись которых начинается с символа «\»);

- строковая константа может занимать несколько строк в исходном тексте программы;

- любой из символов перевода строки или возврата каретки («\r» или «\n») завершают однострочный комментарий, при этом они являются равнозначными и сохраняются в выходной поток.

Общий вид автомата, соответствующего этим требованиям, представлен на рисунке 2.1 (конечное состояние, соответствующее концу входного потока, и ведущие в него дуги не изображены):

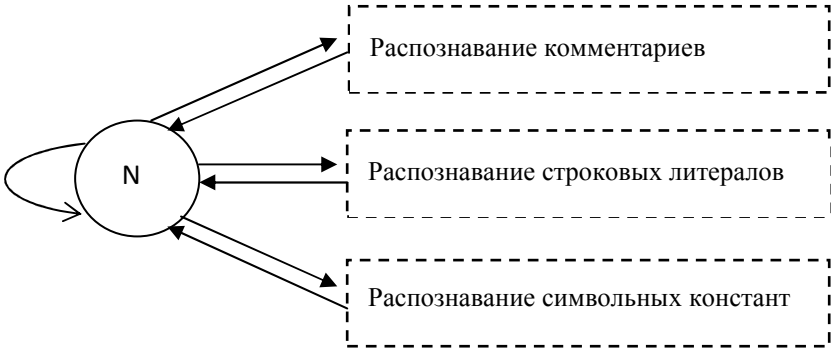

Рисунок 2.1 - Общий вид автомата

На данной диаграмме видно, что переход между состояниями, которые находятся в частях автомата, соответствующих распознаванию комментариев, строковых и символьных констант, невозможен, т.е. эти конструкции не могут: перекрываться или быть вложенными друг в друга.

При реализации необходимо обеспечить корректное завершение программы (запись символов, закрытие файлов и т.п.) при обнаружении конца входного файла в любом состоянии автомата.

Фрагмент возможной реализации (упрощенный) лексического анализатора на языке С с использованием стандартных библиотек показан ниже.

```
typedef enum States { Normal, Slash, Comment,  ...  } States;
                                /*  перечисление всех состояний автомата */
lnt main (lnt arqc,  char ** argv)
{
FILE *  fi,   * fo;                 /*  входной и выходной файлы*/
States State = Normal;              /* текущее состояние*/
lnt c;                              /*  считанный символ(только один текущий)*/
fi = fopen(argv[1], "rb");
If (!fi)
{
```

```
fprintf(stderr,   *Input file \"%s\" open error.\n", argv[1]);
return 1;
}
fo = fopen(argv[2], "wb");
if (!fo)
{
fclose (fi);
fprintf(stderr,  "Output file \"%s\" open error.\n", argv[2]);
return 2;
}
while   ((c=fgetc(fi))   != EOF)       /*  считываем символ и проверяем, */
{                                      /*не   конец ли файла */
      switch   (State)                 /* обрабатываем считанный символ в */
{                                      /* зависимости от текущего состояния */
case Normal:
if   (c ==  '/')                       /* если встретили слэш, */
State = Slash;                /* то перешли в соответствующее состояние*/
else  if   (c =  ...)                  /* если еще что-то,   то аналогично */
State = ...;                           /* и т.д. */
else
fputc (c,   fo);                      /* дублируем считанный символ */
break;                               /* в   выходной поток */
case  Slash:                         /*  если  предыдущим был слэш, то... */
if (c == '*')
State = Comment;
else
State = Normal;
bre ak;
```

```
...                         /* и т.д.   обрабатываем все состояния автомата */
}
}
fclose(fi);             /*  не забывайте закрывать файлы */
fclose(fo);             /*  особенно выходной, т.к. он был открыт для записи */
return 0;
}
```

1.1.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 1

Отчет по лабораторной работе №1 должен содержать:

- Титульный лист.
- Задания.
- Основную часть по заданию 1.
- Основную часть по заданию 2.
- Выводы.
- Список литературы.

Основные части по заданиям 1 и 2 должны содержать таблицы переходов и выходов, а также графы состояний разработанных конечных автоматов и протоколы работы лексических анализаторов на контрольных примерах.

В выводах по работе необходимо привести оценку адекватности разработанных лексических анализаторов, проанализировав протоколы работы при различных ситуациях.

Контрольные вопросы для защиты отчета

1. Дайте определение конечного распознавателя. Что представляют собой входная лента, устройство управления и вспомогательная память?

2. Какой автомат называется недетерминированным?

3. Какой автомат называется детерминированным?

4. Какой автомат называется полностью определенным?

5. Какой автомат называется недоопределенным?

6. Опишите алгоритм построения детерминированного конечного автомата, эквивалентного исходному недетерминированному конечному автомату

7. Приведите решение проблемы принадлежности для конечных автоматов

8. Приведите решение проблемы пустоты языка для конечного автомата

9. Приведите решение проблемы эквивалентности для конечных автоматов

10. Дайте определение недетерминированного конечного преобразователя. Что представляют собой входная лента, устройство управления и выходная лента конечного преобразователя?

11. Дайте определение перевода, определяемого конечным преобразователем.

1.2 Лабораторная работа № 2 «Построение распознавателя с использованием нормальной формы Бэкуса-Наура и детерминированных конечных автоматов»

1.2.1 Задание к лабораторной работе №2

Построить распознаватель числовых констант языков C и C++ (согласно варианту индивидуального задания) с использованием нормальной формы Бэкуса-Наура и детерминированных конечных автоматов, используя в качестве основы распознаватель комментариев, созданный в лабораторной работе № 1.

Задание 1. Формализовать предметную область с использованием естественного языка.

Задание 2. Определить грамматику, порождающую цепочки, соответствующие записи числовых констант языков C/C++. Записать ее правила, используя нормальную форму Бэкуса-Наура и синтаксические диаграммы Вирта.

Задание 3. Построить конечный автомат лексического анализатора этих цепочек (таблицу переходов и граф переходов).

Задание 4. Реализовать лексический анализатор на основе разработанного конечного автомата, построив блок-схему, соответственно графу конечного автомата.

Вариант индивидуального задания выбирается по порядковому номеру в журнале (нечетные – 1, четные – 2):

Вариант 1. Целочисленные константы: 8-я, 10-я и 16-я системы счисления; все возможные целочисленные типы (кроме char) с учетом модификаторов (суффиксов) в записи константы, тип по умолчанию - int.

Вариант 2. Вещественные константы (с плавающей точкой): запись в виде десятичной дроби, показательной формы и их сочетания: все возможные типы с учетом модификаторов (суффиксов) в записи константы, тип по умолчанию - double.

1.2.2 Методические указания к выполнению лабораторной работы № 2

Задание 1. В первую очередь необходимо формализовать предметную область с использованием естественного языка, т.е. определить множество типов данных в соответствии с вариантом индивидуального задания и возможные способы записи констант этих типов. Необходимо учесть, что могут существовать различные способы записи констант для одних и тех же значений, и возможно наличие символов-модификаторов, уточняющих тип

константы. Особое внимание следует обратить на необязательные части в записи константы. (Например, знак числа, дробная часть, символ-модификатор и др.).

Задание 2. В соответствии с формализованным описанием необходимо построить грамматику, порождающую заданные константы, в нормальной форме Бэкуса-Наура. Учитывая, что язык заданных лексем является регулярным, желательно также использовать регулярную (лево- или праволинейную) грамматику, или даже привести ее к виду автоматной грамматики, что упростит в дальнейшем построение конечного автомата и реализацию лексического анализатора. В записи правил грамматики можно использовать метасимвол «|» для компактной записи правых частей правил, но не допускается использование других метасимволов расширений нотации Бэкуса-Наура и регулярных выражений. Построенная грамматика (с целью упрощения) не должна порождать комментарии, символьные константы и строковые литералы, а также числовые константы, не принадлежащие группе типов данных, заданной вариантом индивидуального задания.

Задание 3. В соответствии с построенной грамматикой необходимо определить конечный автомат, который должен идентифицировать успешное распознавание цепочки, ошибки распознавания и пропуск непринятых цепочек. Автомат не должен принимать числовые константы тех типов, которые не соответствуют варианту индивидуального задания – в этих случаях должна идентифицироваться ошибка распознавания или осуществляться пропуск цепочек (их игнорирование без попытки начать распознавание). Например, считав из входного потока цепочку «123.», распознаватель вещественных констант продолжит чтение и анализ символов, а распознаватель целочисленных - выдаст ошибку (символ «точка» не может присутствовать в записи этих констант) и перейдет в некоторое исходное состояние, ожидая начала следующей подходящей

цепочки во входном потоке. Цепочка «123», если за ней следует не алфавитно-цифровой символ (например, пробел), будет принята как константа типа int и не будет принята распознавателем вещественных констант, так как лексема закончилась, но она не содержит признаков константы вещественного типа (десятичной точки или символа «e»). Цепочка «a123» будет просто проигнорирована обоими автоматами, т.к. никакая числовая константа не может начинаться с буквы (эта лексема - идентификатор).

Для корректного распознавания числовых констант необходимо также распознавать комментарии, символьные константы и строковые литералы, т.к. числовые константы как лексемы не могут находиться внутри них. Допустим, автомат в лабораторной работе № 1 распознавал цепочки некоторого языка L' - комбинация цепочек (любых допустимых и в любом порядке) языков комментариев L_R, символьных констант L_C и строковых литералов L_S. Иными словами, $L' = L_R \cup L_C \cup L_S$. Грамматика, построенная в данной работе, задает язык числовых констант L_N и следовательно, автомат будет являться лексическим анализатором для некоторого языка $L = L_N \cup L' = L_N \cup L_R \cup L_C \cup L_S$.

Таким образом, в качестве основы следует взять конечный автомат из лабораторной работы № 1 и дополнить его новыми состояниями (и соответствующими переходами), как изображено на рисунке 2.2.

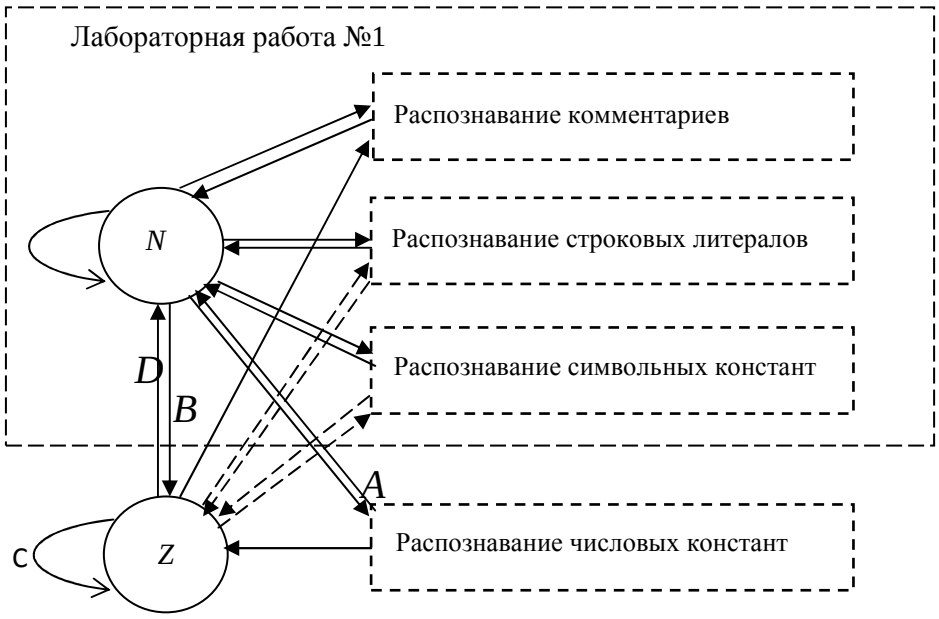

Рисунок 2.2. – Общий вид конечного автомата

На рисунке 2.2 множество *A* содержит символы, которые могут являться первыми в записи распознаваемых констант. Начиная с этого символа и далее, в области «Распознавание числовых констант», входная цепочка должна дублироваться в выходной поток и завершаться либо типом данных константы (цепочка принята), либо сообщением об ошибке (цепочка не принята). Множество *B* - символы, с которых могут начинаться лексемы, внутри которых распознавание констант невозможно (например, идентификаторы и ключевые слова). Таким образом, состояние Z играет роль состояния, блокирующего распознавание констант. Автомат находится в таком состоянии, пока поступают «запрещающие» символы множества *C* (например, буквы, цифры и др.). Следует отметить, что множества *C* и *B* могут отличаться. Вновь распознавание константы может начаться только после «разрешающего» символа из множества *D* (например, символ операции, скобка и т.п.) или разделителя (например, пробел и т.п.).

Задание 4. Распознаватель, реализованный в соответствии с построенным автоматом, должен в результате лексического анализа входного файла (текст программы на языках C/C++) сформировать файл

отчета, содержащий все числовые константы из входного файла в том порядке, в каком они были найдены, с указанием типа этих констант. При отсутствии в записи константы модификаторов (суффиксов, уточняющих тип), полагать, что константа имеет базовый тип вне зависимости от ее значения. Если цепочка не была принята, то необходимо вывести соответствующий результат и показать символ, приведший к ошибке распознавания.

Например, отчет распознавателя целочисленных констант может иметь следующий вид:

123 lnt
123ul unsigned long
123. ERROR
le ERROR
0x123L long
…

1.2.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 2

Отчет по лабораторной работе №2 должен содержать:

- Титульный лист.
- Задания.
- Основную часть по заданию 1.
- Основную часть по заданию 2.
- Основную часть по заданию 3.
- Основную часть по заданию 4.
- Выводы.
- Список литературы.

Основная часть по заданию 1 должна содержать описание предметной области, представляющее собой формализованное описание правил записи лексем на естественном языке.

Основная часть по заданию 2 должна содержать

- запись правил грамматики, порождающей цепочки соответствующие записи числовых констант языков C/C++;

- запись правил в нотации Бэкуса-Наура;

- синтаксические диаграммы правил.

Основная часть по заданию 3 должна содержать

- таблицу переходов и граф состояний конечного автомата для представленной грамматики, включающий распознавание комментариев, символьных констант и строковых литералов.

Основная часть по заданию 4 должна содержать

- блок-схему лексического анализатора;

- протоколы работы лексического анализатора, включая специально внесенные ошибки и случаи, не подлежащие распознаванию (запись чисел внутри строк и комментариев, цифры в идентификаторах и т.п.).

В выводах по работе необходимо привести оценку адекватности разработанного лексического анализатора, в т.ч. в тех случаях, когда невозможно однозначно определить границы лексем при помощи регулярной грамматики (т.е. задача должна решаться при помощи контекстно-свободных грамматик).

Контрольные вопросы для защиты отчета

1. Какое соответствие имеется между классами грамматик из иерархии Хомского и классами распознавателей, допускающих такой же класс языков?

2. Опишите алгоритм определения автоматной грамматики для недерминированного конечного автомата

3. Что такое синтаксис языка?

4. Что такое семантика языка?

5. Что такое «метаязык»?

6. Какие расширения БНФ используются наиболее часто?

7. Определите наиболее характерные типы контекстных условий?

8. Определите понятия «статическая семантика» и «динамическая семантика»

9. Опишите основную концепцию W-грамматик и расскажите, как они используются для задания синтаксиса и семантики языков программирования?

10. Определите основную концепцию операционной семантики

11. Опишите подход к определению семантики языка программирования с использованием аксиоматической семантики.

12. Определите основную концепцию денотационной семантики.

13. Чем отличается операционная семантика от денотационной?

1.3 Лабораторная работа № 3 «Лексический анализатор на основе конечного автомата»

1.3.1 Задание к лабораторной работе №3

Задание 1. В соответствие с заданным вариантом исходных данных разработать алгоритм сканера, составить, ввести и отладить программу, реализующую этот алгоритм.

Задание 2. Составить тестовые наборы данных, получить листинги входных данных и результатов работы;

Варианты исходных данных приведены в Таблице 1.1.

Таблица 1.1 Варианты исходных данных

№ варианта	Исходные данные		
	Ключевые слова	Служебные знаки	Другие лексемы
1	PROGRAM, INPUT, OUTPUT, VAR, INTEGER, BEGIN, IF, THEN, ELSE	«:», «,», «;», «:=», «<», «>», «(», «)»	идентификаторы с количеством символов не более 5
2	PROGRAM, INPUT, OUTPUT, VAR, INTEGER, BEGIN, FOR, TO, DO	«:», «,», «;», «:=», «.», «[», «]», «(», «)»	целые числа с количеством цифр не более 8
3	PROGRAM, INPUT, OUTPUT, VAR, END, BEGIN, WHILE, DO, CHAR	«:», «,», «;», «:=», «.», «=», «<>», «(», апостроф	символьные константы с количеством символов не более 10
4	PROGRAM, INPUT, OUTPUT, VAR, BOOLEAN, END,BEGIN, AND, OR	«:», «,», «;», «:=», «.», «<=», «>=», «(», «)»	константы типа множество символов
5	MAIN, INT, REAL, STDIO, GET, PUT, PRINTF, FOR, DO, AND, OR	«:», «,», «;», «=», «.», «&», «!=», «{», «}», «+», «-»	символические имена с количеством символов не более 10
6	PROCEDURE, OPTIONS, MAIN, BINARY, FIXED, BEGIN, END, IF, THEN,	«:», «,», «;», «=», «.», «&», «<», «(», «)»,	целые числа с количеством цифр не более 4

	ELSE	«+», «-»	
7	DIMENSION, ARRAY, REAL, INTEGER, CONTINUE, DO, IF, GOTO, COMMON	«:», «,», «;», «=», «.», «*», «/», «(», «)», «+», «-»	символические имена с количеством символов не более 4
8	COPY, TO, FIELDS, FOR, WHILE, ALL, REST, REPLACE, USE, IF, ENDIF	«:», «,», «;», «=», «.», «*», «/», «(», «)», «+», «-»"	символьные константы с количеством символов не более 25

1.3.2 Методические указания к выполнению лабораторной работы № 3

Лексический анализ включает в себя сканирование исходного текста программы и распознавание лексем, т.е. элементов, из которых строятся предложения языка, и замену их кодами лексем. Лексемой может быть ключевое слово, имя переменной, число и т.п. Программу, которая выполняет лексический анализ, называют лексическим анализатором или сканером. Точный перечень лексем, которые необходимо распознавать, зависит от языка программирования и от грамматики, используемой для его описания. Например, для языка PASCAL лексемами являются: PROGRAM, INTEGER, REAL, VAR, DO, READ и т.д. Каждый код лексемы обычно представляется кодом таблицы и спецификатором. Спецификатор задает номер строки в таблице, куда занесена лексема. Это позволяет избежать дополнительного поиска по таблицам на последующих этапах трансляции. Сканер работает с тремя таблицами: терминальных символов, символических имен и литералов. Таблица терминальных символов в простейшем случае может иметь вид как в таблице 1.2.

Таблица 1.2 – Пример таблицы терминальных символов

№ строки	Терминальный символ	Признак разделителя
01	PROGRAM	пробел
02	VAR	пробел
03	BEGIN	пробел
04	END	пробел
05	INTEGER	пробел
06	FOR	пробел
07	TO	пробел
08	DO	пробел
09	WHILE	пробел
10	DIV	пробел
11	MOD	пробел
12	AND	пробел
13	OR	пробел
14	:=	+

Таблица символических имен (идентификаторов) может иметь вид как в таблице 1.3.

Таблица 1.3 – Пример таблицы символических имен (идентификаторов)

№ строки	Символическое имя	Адрес	Тип	Другая информация
01	I		int	
02	Y		real	
03	X1		real	
04	...		...	...

Таблица литералов по структуре аналогична таблице символических имен, смотри, например, таблицу 1.4.

Таблица 1.4 – Пример таблицы литералов

№ строки	Литерал	Адрес	Тип	Другая информация
01	1		int	
02	100		int	
03	...	...	...	...

Результатом работы сканера является последовательность кодов лексем. Например, в результате обработки сканером следующего предложения языка Pascal

FOR I := 1 TO 100 DO Y := X1 ;

Будет получена строка:

<1,06><2,1><1,14><3,1><1,07><3,2><1,08><2,2><1,14><2,3>,

где в угловых скобках пара чисел задает код таблицы и спецификатор. В дополнение к своей основной функции – распознавание лексем – сканер также выполняет чтение строк и печать листинга исходной программы. Функционально в сканере могут быть выделены следующие модули:

1. Выделение из входной строки очередного слова.

2. Поиск в таблицах лексем и определение кода лексемы.

3. Формирование и вывод выходной строки.

Для модуля выделения слова важна информация о том, какие символы могут быть признаками начала или конца слова. Например, в языке PASCAL ключевые слова отделяются от других элементов предложения пробелами. Сложнее обстоит дело с выделением идентификаторов и чисел, поскольку разделителем для них может служить любой другой символ, не входящий по определению в идентификатор или число.

Для поиска в таблицах может быть использован, как наиболее простой метод, линейный алгоритм.

При заполнении таблиц выполняется проверка на наличие в ней элемента, совпадающего с выделенным идентификатором или константой, и при совпадении занесение в таблицу не выполняется.

В задачи последнего модуля входит занесение в выходную строку кодов лексем.

1.3.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 3

Отчет по лабораторной работе №3 должен содержать:

- Титульный лист.
- Задания.
- Основную часть по заданию 1.
- Основную часть по заданию 2.
- Выводы.
- Список литературы.

Основная часть по заданию 1 должна содержать блок-схему алгоритма сканера.

Основная часть по заданию 2 должна содержать наборы тестовых данных и листинги результатов тестирования сканера.

В выводах по работе необходимо привести оценку адекватности разработанного сканера.

Контрольные вопросы для защиты отчета

1. Дайте определения цепочек и языка. Какие операции можно выполнять над цепочками символов и над языками?

2. Перечислите и приведите примеры различных способов определения бесконечных языков

3. Что такое синтаксис и семантика языка? Какие средства существуют для описания синтаксиса и семантики?

4. Дайте определение формальной грамматики

5. Каким образом можно классифицировать формальные грамматики?

6. Дайте определение формального языка. Можно ли считать языки программирования формальными?

7. Дайте определение отношения выводимости. Какие виды выводов наиболее часто используются?

8. Определите связь между выводом и деревом вывода.

9. Что такое синтаксис языка?

10. Что такое семантика языка?

11. Что такое «метаязык»?

12. Какие расширения БНФ используются наиболее часто?

2 Методы синтаксического анализа

2.1. Лабораторная работа № 4 «Метод грамматического разбора на основе РБНФ и синтаксических диаграмм»

2.1.1 Задание к лабораторной работе №4

Составить определение заданного подмножества языка (по вариантам из таблицы 2.1) в РБНФ и с помощью синтаксической диаграммы.

Задание 1. На основе РБНФ-определения составить синтаксическую диаграмму, а затем блок-схему алгоритма разбора;

Задание 2. Дополнить программу лексического анализатора (п. 1.3) модулями, реализующими алгоритм разбора;

Задание 3. Составить тестовые наборы данных, получить листинги входных данных и результатов работы.

Таблица 2.1 - Варианты исходных данных.

№ варианта	**Исходные данные**
1	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания и оператор IF ... THEN.
2	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания и оператор FOR.
3	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания и оператор

	WHILE.
4	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания, в правой части которого несколько соотношений соединяются знаками логических операций.
5	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания и оператор REPEAT.
6	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания и оператор CASE.
7	Подмножество языка Pascal. Допускается: – в описаниях – переменные целого типа; – в операторной части – оператор присваивания с арифметическим выражением над целыми переменными.
8	Подмножество языка Pascal. Допускается: - в описаниях - переменные и массивы вещественного типа; - в операторной части - оператор присваивания с выражением над вещественными переменными;

2.1.2 Методические указания к выполнению лабораторной работы № 4

Синтаксическая диаграмма графически изображает структуру конструкций алгоритмического языка. Отдельными элементами диаграммы могут быть основные символы или понятие языка. Из каждого элемента выходит одна или несколько стрелок, указывающих на те элементы, которые могут непосредственно следовать за данным элементом (то есть стрелки указывают возможных преемников каждого из элементов диаграммы). Стрелка, не выходящая из какого-либо элемента, является входом в диаграмму, а стрелка, не ведущая к какому-либо элементу, означает выход из диаграммы, то есть конец синтаксического определения. Чтобы легче было отличить в диаграммах основные символы (терминальные) от понятий (нетерминальных символов) их выделяют каким-либо способом, например, заключают основные символы в кружки или овалы, а понятия – в прямоугольники.

Например, определение идентификатора в БНФ имеет вид:

<имя>::=<буква>{<буква>|<цифра>}

<буква>::=A|B|C|…|Z

<цифра>::=0|1|…|9

Синтаксическая диаграмма для этого определения будет выглядеть как на рисунке 2.1. Блок-схема изображена на рисунке 2.2.

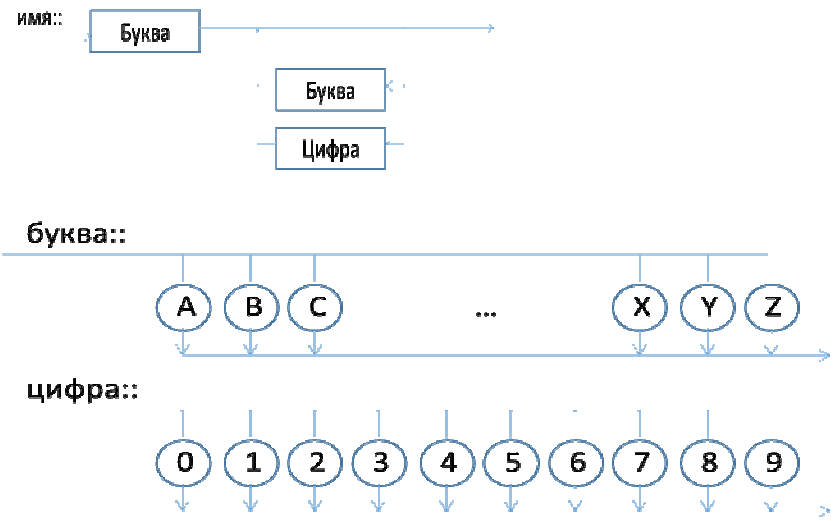

Рисунок 2.1 - Синтаксическая диаграмма определения «имя»

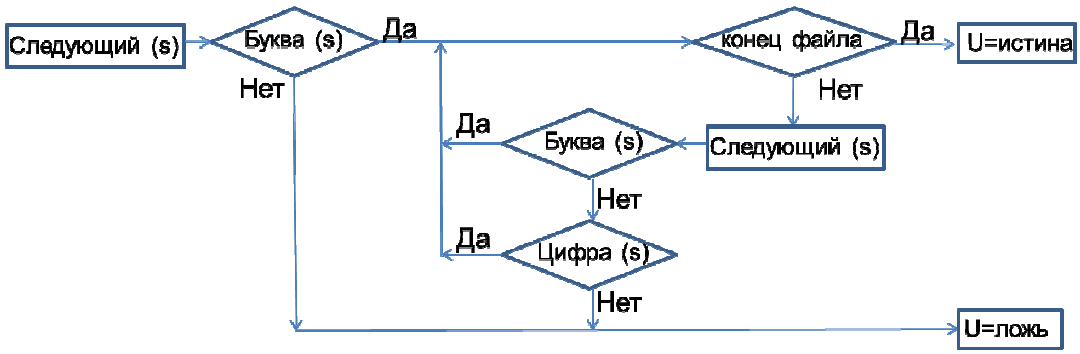

Рисунок 2.2 – Блок-схема программы для определения «имя»

Указанные в блок-схеме подпрограммы «буква()» и «цифра()» выполняют соответственно проверку очередного символа входной строки на принадлежность множеству букв и цифр, «следующий ()» - считывает очередной элемент слова.

Общим недостатком формальных методов является отсутствие средств для задания смысловых правил синтаксиса языка. Примером такого вида правил является требование, чтобы все объекты в PASCAL-программе были описаны. Однако при переходе к алгоритмическому заданию синтаксиса (например, в виде блок-схемы) эта трудность преодолима. Пусть в рассмотренном примере требуется, чтобы количество символов в идентификаторе не превышало 5. Если добавить счетчик символов «счетчик», то можно изменить блок-схему как показано на рисунке 2.3.

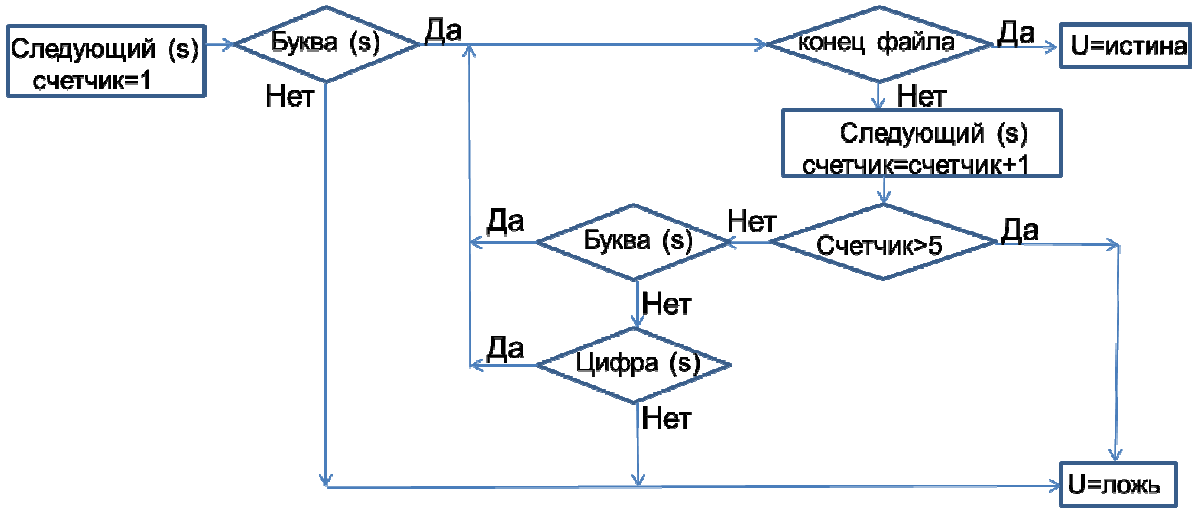

Рисунок 2.3 – Расширенная блок-схема программы для определения «имя»

Задача синтаксического анализа упрощается, если программа предварительно обработана лексическим анализатором и поступает на вход синтаксического анализатора в форме последовательности кодов лексем. Тогда любой модуль блок-схемы, соответствующий элементу диаграммы, в начале по коду таблицы определяет допустимость данного вида лексем в структуре предложения и, если допустим, тогда по спецификатору проверяется допустимость лексемы.

2.1.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 4

Отчет по лабораторной работе №4 должен содержать:

- Титульный лист.
- Задания.
- Основную часть по заданию 1.
- Основную часть по заданию 2.
- Основную часть по заданию 2.
- Выводы.
- Список литературы.

Основная часть по заданию 1 должна содержать

- РБНФ заданного множества;
- синтаксическую диаграмму заданного множества;

Основная часть по заданию 2 должна содержать блок-схему алгоритма разбора.

Основная часть по заданию 3 должна содержать наборы тестовых данных и листинги результатов тестирования программы.

В выводах по работе необходимо привести оценку адекватности разработанного приложения.

Контрольные вопросы для защиты отчета

1. Какое соответствие имеется между классами грамматик из иерархии Хомского и классами распознавателей, допускающих такой же класс языков?

2. Опишите алгоритм определения автоматной грамматики для недетерминированного конечного автомата

7. Определите наиболее характерные типы контекстных условий?

8. Определите понятия «статическая семантика» и «динамическая семантика»

9. Опишите основную концепцию W-грамматик и расскажите, как они используются для задания синтаксиса и семантики языков программирования?

10. Определите основную концепцию операционной семантики

11. Опишите подход к определению семантики языка программирования с использованием аксиоматической семантики.

12. Определите основную концепцию денотационной семантики.

13. Чем отличается операционная семантика от денотационной?

2.2. Лабораторная работа № 5 «Алгоритм синтаксического анализа, с полным возвратом для контекстно-свободных грамматик»

2.2.1 Задание к лабораторной работе №5

Построить транслятор для языка, заданного контекстно-свободной грамматикой, с использованием детерминированного моделирования недетерминированного анализатора.

Задание 1. Записать грамматику, заданную вариантом индивидуального задания (Таблицы 2.2, 2.3), включая полные множества правил, терминальных и нетерминальных символов.

Задание 2. Выполнить преобразование записанной грамматики при необходимости, для исключения левой рекурсии.

Задание 3. Построить транслятор (интерпретатор) программ заданного языка, реализующий алгоритм с полным возвратом.

Варианты индивидуальных заданий

В соответствии с номером варианта индивидуального задания (от 1 до 20) из таблицы 2.2 выбираются варианты 6-ти разделов грамматики $G(N,\Sigma,P,S)$. Затем, в соответствии с вариантами разделов, из таблицы 2.3 выписываются правила, образующие полное множество правил P грамматики. На основе полученного множества правил P строится алфавит грамматики $\Sigma \bigcup N$. Нетерминальный символ S является целевым (начальным) во всех полученных грамматиках.

Таблица 2.2 - Соответствие вариантов разделов грамматики варианту индивидуального задания

Вариант задания	Варианты разделов грамматики						Вариант задания	Варианты разделов грамматики					
	I	II	III	IV	V	VI		I	II	III	IV	V	VI
1	а	б	г	а	б	б	11	б	а	а	а	б	г
2	б	б	г	а	а	б	12	в	г	г	в	б	б
3	а	б	в	г	б	б	13	а	а	в	а	б	г
4	б	б	в	г	а	б	14	б	а	в	а	б	а
5	в	б	б	г	а	в	15	г	а	в	а	б	а
6	б	д	а	а	а	б	16	а	в	б	б	б	а
7	г	г	б	в	б	а	17	г	в	б	б	а	а

8	г	а	а	а	а	г	18	а	д	б	б	б	а
9	г	г	г	в	б	б	19	б	в	б	б	б	а
10	в	г	б	в	б	а	20	в	в	б	б	а	а

Таблица 2.3 - Содержание разделов грамматики по вариантам

Раздел	Вариант	Правила грамматики	Примечания
I	а	$S \to I = E$;	Операторы присваивания переменной с именем I значения выражения E. Переменная с именем I определяется, если она не была определена ранее
	б	$S \to I := E$;	
	в	$S \to (I, E)$	
	г	$S \to I(E)$	
II	а	$E \to E + T \mid E - T \mid T$ $T \to T * M \mid T / M \mid M$ $M \to (E) \mid I \mid C$	Выражения с традиционными арифметическими операциями +, —, *, / и круглыми скобками
	б	$E \to E"\mid"T \mid T$ $T \to T \,\&\, M \mid M$ $M \to\sim M \mid (E) \mid I \mid C$	Выражения с поразрядными логическими операциями 1 (или), & (и), ~ (инверсия) и круглыми скобками
	в	$E \to E + T \mid E - T \mid T$ $T \to M \mid F(E)$ $M \to I \mid C$ $F \to \sin \mid \cos \mid sqr \mid sqrt$	Выражения с операциями суммирования, вычитания и вызова функции (синус, косинус, квадрат числа, квадратный корень)
	г	$E \to -E \mid +(T) \mid *(T) \mid S \mid M$ $T \to E,T \mid E$ $M \to I \mid C$	Выражения с унарным минусом и n-местными операциями суммирования и умножения в префиксной форме
	д	$E \to T > T \mid T < T \mid T == T \mid T$ $T \to T + M \mid T - M \mid M$	Выражения с логическими и арифметическими операциями, семантика

		$M \rightarrow !M \mid (E) \mid I \mid C$	которых соответствует языку С
III	а	$I \rightarrow A \mid AA \mid AD \mid AAD$	Идентификаторы
	б	$I \rightarrow AI \mid A$	
	в	$I \rightarrow AK \mid A$ $K \rightarrow AK \mid DK \mid A \mid D$	
	г	$I \rightarrow AK \mid A$ $K \rightarrow DK \mid D$	
IV	а	$C \rightarrow DC \mid D$	Константы
	б	$C \rightarrow DC \mid D \mid ..R$ $R \rightarrow DR \mid D$	
	в	$C \rightarrow \#R$ $R \rightarrow DR \mid D$	
	г	$C \rightarrow D$	
V	а	$A \rightarrow a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid$ $i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid$ $s \mid t \mid u \mid v \mid w \mid x \mid y \mid z \mid \_$	Буквы
	б	$A \rightarrow a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid$ $i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid$ $s \mid t \mid u \mid v \mid w \mid x \mid y \mid z \mid \_$	
VI	а	$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$	Десятичные цифры
	б	$D \rightarrow 0 \mid 1$	Двоичные цифры
	в	$D \rightarrow true \mid false$	Логические значения
	г	$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7$	Восьмеричные цифры

2.2.2 Методические указания к выполнению лабораторной работы № 5

Заданная грамматика G порождает язык $L(G)$, каждая возможная цепочка которого представляет собой одни оператор присваивания переменной (левый операнд) значения некоторого выражения (правый операнд), причем, при первом использовании в качестве левого операнда,

переменная определяется, а при использовании ее в выражении правого операнда - должна быть определена ранее. Последовательность таких операторов, которые могут быть отделены друг от друга пустыми символами (в любом количестве, в т.ч. 0), представляет собой программу на языке L'. Таким образом, $L' = (L(G) \bigcup \Delta)^*$, где Δ - множество пустых (непечатаемых) символов, выполняющих роль разделителей (в данном случае - между операторами).

Тип данных, значениями которого оперирует программа на языке, порождаемом заданной грамматикой, определяется записью констант этого языка (разделы IV и VI грамматики).

Следует отметить, что правила грамматики, записанные в таблице 2.3, не предполагают наличия разделителей (пустых символов) между лексемами в одном операторе, в то время, как для большинства языков программирования это является нормой. Это сделано для упрощения записи правил грамматики. Например, запись правила $S \to I = E$ из раздела 1,а таблица 2.3 с учетом наличия пустых символов будет иметь вид $S \to \delta_1 I \delta_2 = \delta_3 E \delta_4$, где $\delta_i \in \Delta^*$ и это так же, с формальной точки зрения, должно быть раскрыто при помощи правил грамматики. Очевидно, что ввод разделителей в явном виде излишне усложняет запись грамматики, поэтому их наличие должно быть учтено во время реализации интерпретатора. При выполнении лабораторной работы необходимо самостоятельно определить либо возможность, либо обязательное отсутствие таких символов, как пробел, табуляция и т.п., между терминальными и нетерминальными символами в правых частях правил разделов I и II грамматики, т.е. *между отдельными лексемами* внутри операторов.

Все алгоритмы синтаксического анализа моделируют работу некоторого преобразователя с магазинной памятью. В основе нисходящего анализатора лежит недетерминированный МП-преобразователь.

Алгоритм с полным возвратом - это детерминированное моделирование недетерминированного МП-преобразователя.

Наиболее простой способ состоит в линейном упорядочивании конечного множества последовательность тактов, выполняемых для входной цепочки.

Если целью является получение ***одного разбора***, то обнаружив эту последовательность можно прекратить моделирование.

В случае, когда нужно получить ***все разборы*** моделируются все последовательность тактов.

Конечно же в случае, когда ***входная цепочка не принадлежит*** языку придется выполнить все последовательности тактов для обнаружения этого факта.

При моделировании последовательности тактов необходимо упорядочить таким образом, чтобы можно было вернуться по последним тактам (выполняя их обратное прослеживание) к конфигурации, в которой возможен альтернативный такт.

Недостатком этого алгоритма является выполнение большого числа шагов. Для ускорения можно ***упорядочить правила грамматики*** так, чтобы наиболее вероятные альтернативы испытывались первыми;

Полное линейное моделирование МП-преобразователя **невозможно** если для входной цепочки возможны бесконечные последовательности тактов. Для исключения таких ситуаций на преобразователь накладываются ограничения. Например, левый анализатор не зацикливается тогда и только тогда, когда грамматика не леворекурсивна. Поэтому прежде, чем приступать к построению транслятора, необходимо убедиться, что

грамматика, заданная в соответствии с вариантом индивидуального задания, не содержит левой рекурсии или привести ее к такому виду.

Процесс работы распознавателя соответствует построению дерева вывода сверху вниз, начиная с начального символа S (***активная вершина***). Затем выполняются следующие действия:

- ***Если активная вершина помечена нетерминалом A***, то выбираем первую альтернативу этого символа (правую часть правила). Пусть это $X_1X_2…X_k$, добавляем их к потомкам вершины A и вершину X_1 делаем активной.

В случае k=0 активной становится вершина, расположенная справа от A.

- ***Если активная вершина помечена терминалом a***, то сравниваем текущий входной символ с a. Если они совпадают, то активной становится вершина, расположенная справа от a. Входной указатель сдвигается на один символ вправо.

Если входной символ с a не совпадает, то необходимо:

- вернуться к вершине, в которой применялось предыдущее правило;

- установить входной указатель в позицию, соответствующую предыдущему правилу;

- испытать следующую альтернативу.

Если альтернатив нет, вернуться к предыдущей вершине и т.д.

- ***Если мы окажемся в результате возвратов в корне*** (символ S) и все альтернативы для него будут исчерпаны, делается вывод о том, что входная цепочка синтаксически неправильна.

Для реализации алгоритма используем объекты:

- ***Магазин L_1*** (вершина справа) содержит историю проделанных выборов альтернатив и прочитанные входные символы.

- ***Магазин L_2*** (вершина слева) содержит текущую цепочку с помеченным активным символом.

- ***Счетчик***, хранящий текущую позицию входного указателя.

- ***s - состояния алгоритма*** (*q*-нормальное состояние, *b*-состояние возврата, *t*-заключительное состояние),

Для каждого нетерминала *A* упорядочиваем альтернативы (*A*1, *A*2, …).

Текущая конфигурация алгоритма содержит четыре объекта: (*s*, *i*, α, β):

i-позиция входного указателя,

α-содержимое магазина L_1,

β-содержимое магазина L_2 .

(*q*, *1*, ε, S⌋) – начальная конфигурация.

Типы шагов алгоритма:

- *разрастание дерева* (построение куста из активной вершины)

- *успешное сравнение входного символа с порожденным терминалом*. Входной символ передается из магазина L_2 в магазин L_1. Значение позиции входного указателя увеличивается на единицу.

- *неудачное сравнение входного символа с порожденным терминалом*. Алгоритм переходит в состояние возврата.

- *возврат по входу*. Входные символы переносятся из магазина L_1 в магазин L_2.

- *испытание очередной альтернативы*. В магазине L_2 цепочка, соответствующая предыдущему правилу, заменяется на цепочку нового правила.

- *успешное завершение*. Достигнут конец входной цепочки.

Контекстные зависимости, присущие описанному заданным образом языку (например, идентификаторы, которые определены в предшествующих операторах, зарезервированные слова и т.п.) реализуются посредством семантического анализатора. Семантический анализатор представляет собой

некоторую структуру данных (таблицу идентификаторов), хранящую информацию о текущем контексте, и обслуживающих ее алгоритмов (поиск, извлечение, добавление и т.п.). Выбранный способ программной реализации таблицы идентификаторов не должен вносить существенных ограничений на количество идентификаторов, используемых в программе, и их длину, если это не обусловлено исходной грамматикой.

Транслятор должен после интерпретации каждого оператора программы выводить его порядковый номер, имя переменной и значение, присвоенное ей в этом операторе, а после интерпретации всей программы - список всех переменных и их значений. В случае ошибки необходимо выводить развернутое сообщение о ее причинах.

2.2.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 5

Отчет по лабораторной работе №5 должен содержать:

- Титульный лист.
- Задания.
- Основную часть.
- Выводы.
- Список литературы.

Основная часть отчета по лабораторной работе должна содержать:

- запись исходной грамматики (по таблицам 2.2 и 2.3) в нотации Бэкуса-Наура;
- описание языка, порождаемого заданной грамматикой (на естественном языке с примерами порождаемых конструкций);
- вывод: содержит данная грамматика левую рекурсию или нет;
- описание преобразований грамматики (в случае необходимости) и полученную в результате грамматику без левой рекурсии;

- описание алгоритмов, реализующих типы шагов (в виде блок-схем);

- протоколы работы анализатора, включая как интерпретацию правильных программ языка, так и случаи идентификации специально внесенных ошибок.

Контрольные вопросы для защиты отчета

1. Дайте определение недетерминированного автомата с магазинной памятью и языка, допускаемого МП-автоматом

2. Определите такие понятия, как конфигурация МП-автомата, такт работы МП-автомата, ε-такт.

3. Дайте определение недетерминированного расширенного автомата с магазинной памятью. Что представляют собой входная лента, устройство управления и вспомогательная память расширенного МП-автомата?

4. Определите такие понятия, как конфигурация расширенного МП-автомата, такт работы МП-автомата, ε-такт.

5. Опишите процесс построения МП-автомата по расширенному МП-автомату

6. Дайте определение МП-автомата, допускающего входные цепочки опустошением магазина

7. Опишите процесс построения МП-автомата, допускающего входные цепочки опустошением магазина

8. Опишите алгоритм построения недетерминированного МП-автомата по КС-грамматике

9. Опишите алгоритм построения недетерминированного расширенного МП-автомата по КС-грамматике

10. Опишите алгоритм построения КС-грамматики по недетерминированному МП-автомату

11. Дайте определение детерминированного МП-автомата

12. Дайте определение детерминированного расширенного МП-автомата

13. Докажите лемму о существовании ДМП-автомата, для которого всегда возможен очередной такт

14. Дайте определение зацикливающей конфигурации

15. Опишите алгоритм построения множества зацикливающих конфигураций

16. Дайте определение дочитывающего ДМП-автомата

17. Опишите алгоритм, позволяющий по заданному ДМП-автомату построить эквивалентный ему дочитывающий ДМП-автомат

18. Дайте определение недетерминированного МП-преобразователя и недетерминированного расширенного МП-преобразователя

19. Дайте определение детерминированного МП-преобразователя и детерминированного расширенного МП-преобразователя

20. Будет ли МП-преобразователь детерминированным, если он построен на основе ДМП-автомата

21. Дайте определение перевода, определяемого МП-преобразователем

2.3. Лабораторная работа № 6 «Разработка нисходящего табличного анализатора»

2.3.1 Задание к лабораторной работе №6

Построить транслятор для языка, заданного контекстно-свободной грамматикой (см задание по вариантам из п.1.3.1, таблица 1.1), с использованием моделирования нисходящего табличного анализатора.

Задание 1. В соответствии с вариантом исходных данных разработать формальную грамматику реализуемого подмножества языка Паскаль.

Задание 2. Составить, ввести и отладить программу, реализующую алгоритм синтаксического анализатора;

Задание 3. Составить тестовые наборы данных (позитивные и негативные). Получить листинги входных данных и результатов работы;

2.3.2 Методические указания к выполнению лабораторной работы № 6

Общие требования к синтаксическому блоку:

1. Операции: "+", "-", "/", "*" ("<",">","=","<=",">=","and", "or" – дополнительно по желанию).

2. Операторы: присваивания (обязательно для всех вариантов), условия и циклы (по варианту).

3. Для ввода данных и вывода результатов обязательно реализовать соответствующие функции (операторы), например, "read" и "write".

4. Процедуры, функции и модули могут быть разработаны по желанию.

Синтаксический блок компилятора необходимо объединить с лексическим блоком.

На входе – программа на выбранном языке, разработанная в соответствии с правилами грамматики.

На выходе – сообщение о том, что текст программы соответствует грамматике (программа без ошибок), либо сообщение об ошибке. Дополнительно можно представить дерево синтаксического разбора (например, в виде логического файла, показывающего промежуточные действия компилятора или дерево синтаксического разбора).

В случае разработки типизированного языка необходимо продемонстрировать семантическую работу компилятора.

2.3.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 6

Отчет по лабораторной работе №6 должен содержать:

- Титульный лист.

- Задания.

- Основную часть по заданию 1.

- Основную часть по заданию 2.

- Основную часть по заданию 3.

- Выводы.

- Список литературы.

Основная часть по заданию 1 должна содержать постановку задачи:

– формулировку задачи (ограничения реализуемого подмножества языка Паскаль);

– формальную грамматику пользователя, записанную в РБНФ. Дополнительно можно представить графическое представление в форме диаграмм Вирта.

– преобразованную для реализации формальную грамматику разбора (атрибуты и множество выбора). В грамматике для каждого правила должно быть определено множество «ВЫБОР» (Множество входных символов, для которых может быть применимо правило).

Основная часть по заданию 2 должна содержать описание реализации:

– описание выбора метода решения;

– описание структуры блока синтаксического анализа;

– алгоритмы функционирования компонент (блоков) анализатора.

Основная часть по заданию 3 должна содержать набор позитивных и негативных тестов и листинги результатов тестирования программы.

В выводах по работе необходимо привести оценку адекватности разработанного автомата.

Контрольные вопросы для защиты отчета

1. Какие КС-грамматики называются s-грамматиками?

2. Что такое множества выбора и как они используются при анализе s-грамматик?

3. Какие Кс-грамматики называются LL(1)-грамматиками?

4. Дайте определения s-, q- и LL(1)-грамматик. Как соотносятся классы этих грамматик?

5. Как вычисляются множества выбора?

6. Как описывается такт работы LL(1)-анализатора?

7. Как можно представить управляющую таблицу анализатора?

8. Как определяются множества FIRSTk(β) и FOLLOWk(A)?

9. Какие грамматики называются LL(k)-грамматиками?

10. Что представляет собой головной модуль в синтаксическом анализаторе, реализующем метод рекурсивного спуска?

11. Какие требования предъявляются к языку программирования для реализации метода рекурсивного спуска?

12. Как программируются процедуры для распознавания нетерминалов в методе рекурсивного спуска?

2.4. Лабораторная работа № 7 «Программирование синтаксического анализатора на основе процедуры рекурсивного спуска»

2.4.1 Задание к лабораторной работе №7

Построить транслятор для языка, заданного контекстно-свободной грамматикой (см задание по вариантам из п.2.2.1, таблица 2.3), с использованием метода рекурсивного спуска

Задание1. Записать грамматику, заданную вариантом индивидуального задания, включая полные множества правил, терминальных и нетерминальных символов.

Задание 2. Выполнить преобразование записанной грамматики к виду грамматики рекурсивного спуска (при необходимости).

Задание 3. Построить транслятор (интерпретатор) программ заданного языка, реализующий алгоритм рекурсивного спуска.

2.4.2 Методические указания к выполнению лабораторной работы № 7

Прежде, чем приступать к построению транслятора, необходимо убедиться, что грамматика, заданная в соответствии с вариантом индивидуального задания, является грамматикой рекурсивного спуска или привести ее к такому виду. Грамматикой рекурсивного спуска называется такая грамматика, в которой для любого нетерминального символа $A \in N$ существует либо единственное правило $A \to \alpha$, где $\alpha \in (N \cup \Sigma)^*$, либо в правилах вида $A \to a_1\beta_1 \mid a_2\beta_2 \mid ... \mid a_n\beta_n$, где $a_i \in \Sigma$, $\beta_i \in (N \cup \Sigma)^*$, $a_i \neq a_j$ при $i \neq j$ (т.е. правые части таких правил начинаются с различных терминальных символов). Следовательно, по первому символу каждой цепочки, выводимой из некоторого нетерминального символа A, можно однозначно установить соответствующее правило, сравнивая этот символ с первыми символами правых частей правил, левые части которых представлены символом A.

Если грамматика не удовлетворяет рассмотренному выше требованию необходимо выполнить факторизацию грамматики, которая заключается в замене правил вида:

$A \to a\beta$

$A \to a\gamma$

на новые правила:

$A \to aX$

$X \to \beta$

$X \to \gamma$

В последних правилах введен новый нетерминал X. После такой замены получим грамматику, эквивалентную исходной грамматике.

Построение распознавателя для языков, заданных грамматикой рекурсивного спуска, выполняется по следующим правилам:

1. Для каждого нетерминального символа $A \in N$ строится своя процедура разбора, например **ProcA,** выполняющая распознавание всех цепочек, выводимых из символа *A*.

2. Первый символ входного потока, анализируемый каждой процедурой разбора (т.е. первый символ строки, выводимой из соответствующего нетерминального символа), считывается до ее вызова.

3. При завершении процедуры разбора, ею должен быть считан один символ входного потока, следующий за распознанной цепочкой (т.е. выводимой из соответствующего нетерминального символа).

4. Алгоритм каждой процедуры разбора строится в соответствии с правой частью выбранного правила для соответствующего нетерминального символа:

- если очередной символ правой части правила является терминальным, то он должен быть равен текущему считанному символу входного потока (на первом шаге производится выбор правила), иначе цепочка не принимается (выводится сообщение об ошибке); при успешном сравнении считывается следующий символ входного потока и происходит переход к анализу следующего символа правой части выбранного правила;

- если очередной символ правой части правила является нетерминальным, то вызывается соответствующая ему процедура разбора (текущий считанный символ будет первым анализируемым символом в этой процедуре - правило 2, а после завершения вызванной процедуры текущий символ уже будет считан из входного потока - правило 3).

Каждая процедура, как правило, выполняет некоторые действия в соответствии с распознанной цепочкой и возвращает некоторое значение. Например, в случае интерпретации выражений, процедура распознавания

константы «собирает» символы константы вычисляет и возвращает ее значение, а процедура распознавания выражения с операцией «+» вызывает процедуры для операндов, суммирует возвращенные ими значения и возвращает полученный результат. Очевидно, что операндом может являться как константа, так и другое выражение, в т.ч. содержащее ту же операцию. Таким образом, возможна рекурсия (отсюда и название метода), но единственное ограничение - рекурсия не должна быть левой (как прямой, так и непрямой).

Расширить применение метода рекурсивного спуска можно за счет:

- приведения исходной грамматики к требуемому виду посредством выполнения левой факторизации, устранения левой рекурсии, преобразования к нормальной форме Грейбах и т.п.;

- построения конечных автоматов для распознавания лексем, записи правил в регулярных выражениях, привлечения эвристического анализа, модификации базового алгоритма процедуры разбора и др.

Контекстные зависимости, присущие заданному описанным образом языку, например, идентификаторы, которые определены в предшествующих операторах, зарезервированные слова и т.п., реализуются посредством семантического анализатора. Семантический анализатор представляет собой некоторую структуру данных (таблицу идентификаторов), хранящую информацию о текущем контексте, и обслуживающих ее алгоритмов (поиск, извлечение, добавление и т.п.). Выбранный способ программной реализации таблицы идентификаторов не должен вносить существенных ограничений на количество идентификаторов, используемых в программе, и их длину, если это не обусловлено исходной грамматикой.

Транслятор должен после интерпретации каждого оператора программы выводить его порядковый номер, имя переменной и значение, присвоенное ей в этом операторе, а после интерпретации всей программы - список всех

переменных и их значений. В случае ошибки необходимо выводить развернутое сообщение о ее причинах.

2.4.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 7

Отчет по лабораторной работе №7 должен содержать:

- Титульный лист.
- Задания.
- Основную часть по заданию1.
- Основную часть по заданию2.
- Основную часть по заданию 3.
- Выводы.
- Список литературы.

Основная часть отчета по заданию 1 должна содержать представление формальной грамматики, включая полные множества правил, терминальных и нетерминальных символов.

Основная часть по заданию 2 должна содержать вывод о том, принадлежит или нет заданная грамматика классу грамматик рекурсивного спуска и преобразованную грамматику (при необходимости), с указанием какие преобразования были выполнены.

Основная часть по заданию 3 должна содержать описание алгоритмов процедур разбора (по одной для каждого раздела грамматики). Протоколы работы анализатора, включая как интерпретацию правильных программ языка, так и случаи идентификации специально внесенных ошибок.

Контрольные вопросы для защиты отчета

1. Как программируются процедуры для распознавания нетерминалов в методе рекурсивного спуска?

3 Формальные методы описания и реализации синтаксически управляемого перевода

3.1. Лабораторная работа № 8 «Промежуточные формы представления программ. Преобразование арифметического скобочного выражения в ПОЛИЗ. Вычисление арифметического выражения, записанного в ПОЛИЗ»

3.1.1 Задание к лабораторной работе №8

Задание 1. Разработать программу, осуществляющую перевод арифметического выражения в ПОЛИЗ

Задание 2. Разработать программу вычисления арифметического выражения по его промежуточной записи в ПОЛИЗ.

3.1.2 Методические указания к выполнению лабораторной работы № 8

Компилятор определяет перевод исходной программы (цепочку символов на языке программирования) в объектный код.

Компиляцию можно рассматривать как многоэтапный процесс:

- лексический анализ – цепочка символов преобразуется в цепочку лексических единиц (лексем);

- синтаксический анализ – цепочка лексем преобразуется в промежуточную форму, являющуюся линейной формой записи синтаксического дерева.

Различные формы представления промежуточной программы отличаются друг от друга способами соединения операторов и операндов.

Общепринятыми промежуточными формами являются

- польская инверсная запись;

- тетрады;

- триады;

- связные списочные структуры.

В настоящее время ряд компиляторов в качестве промежуточной формы программы генерирует файлы, содержащие байт-коды, представляющие собой инструкции для виртуальной машины Java. Ядро этой машины реализовано практически для любого компьютера и поэтому байт-коды можно рассматривать как независимые от платформы приложения.

Польская запись (префиксная форма)

Рассмотрим процесс вычисления арифметических выражений со скобками, например: (((*A*+*B*)-*C*)+(*D***E*)).

Для начала необходимо вычислить выражение, стоящее в самых внутренних скобках, далее, заменив его на полученное значение, продолжать процесс пока есть скобки. Для отыскания самых внутренних скобок можно придать каждой открывающейся скобке вес +1, а каждой закрывающейся -1. Тогда при просмотре выражения надо будет просто найти скобку самого большого веса.

При использовании функциональной записи *f*(*x*, *y*), в которой операция предшествует операндам, скобки излишни.

Например, для выражения *A*+*B*, функциональная запись: +(*A*, *B*). В случае когда все функции бинарные можно обойтись без скобок +*AB*.

Тогда выражение (((*A*+*B*)-*C*)+(*D***E*)) в польской записи будет представлено, как +-+*ABC***DE*.

Для записи отдельных операций вводятся дополнительные символы, например, унарный минус может быть заменен на «!», возведение в степень на «^» и т.д.

Например, для выражения $(A+B)*C^{-D/(E+F)}$ префиксная форма записи имеет вид: *+*AB*^*C*/!*D*+*EF*

Для преобразования выражения в префиксную форму оно просматривается справа налево. Трансляторы же в большинстве своем считывают входные цепочки слева направо, поэтому данная форма используется редко. В компиляторах чаще всего используют постфиксную форму.

Постфиксная форма (польская инверсная запись, ПОЛИЗ)

В этой форме операция записывается справа от операндов.

Например, для выражения *A*+*B*, постфиксная форма записи: *AB*+.

Выражение (((*A*+*B*)-*C*)+(*D***E*)) в ПОЛИЗ будет имеет вид: *AB*+*C*-*DE**+.

Для выражения $(A+B)*C^{-D/(E+F)}$ постфиксная форма записи имеет вид: *AB*+*CD*!*EF*+/^*.

Свойства ПОЛИЗ

- однозначное определение порядка выполнения операций;
- простая синтаксическая структура;
- метод определения правильности выражений: если каждому элементу выражения присвоить вес p по правилу: вес переменной или константы p=1, вес знака унарной операции p=0, вес знака бинарной операции p=-1, вес знака *n*-местной операции p=-(*n*-1), тогда сумма весов элементов правильного выражения равна 1.

Например, для выражения: p (*AB*+*CD*!*EF*+/^*)=1+1-1+1+1+0+1+1-1-1-1-1=1

Графическое представление выражений

Выражения можно изображать в виде дерева: вершины – операции, ветви – операнды. Вершины, лежащие ниже, соответствуют операциям, которые выполняются раньше.

Например, дерево выражения *a**(*b*+*c*), изображено на рисунке 3.1.

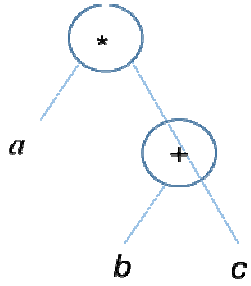

Рисунок 3.1 - дерево выражения *a**(*b*+*c*)

Префиксная форма соответствует обходу дерева сверху вниз (слева направо) – прямой обход. Постфиксная форма (ПОЛИЗ) - обходу снизу вверх (слева направо) – обратный обход.

ПОЛИЗ часто используется в качестве промежуточного языка, так как обладает рядом замечательных свойств:

- действия, заданные в ПОЛИЗ, можно программировать без возврата, так как операнды предшествуют операциям;

- операнды в ПОЛИЗ расположены в том же порядке, что и в выражении, изменяется только порядок знаков операций.

Для представления других конструкций языка программирования вводится расширение ПОЛИЗ.

Для задания элементов массивов введем дополнительную операцию вычисление адреса элемента массива SUBS – операнды операции: имя массива, значения индексов, число операндов (размерность массива +1).

Например, представление выражения *b*[*i*, *j*+1] в ПОЛИЗ имеет вид: *bij*1+3SUBS, графическое представление изображено на рисунке 3.2.

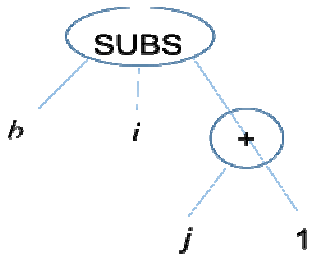

Рисунок 3.2. - графическое представление выражения *bij*1+3SUBS

Для задания функций используем операцию CALL. Структура операции не отличается от SUBS.

Например, представление выражения *f*(*i*, *j*+1) в ПОЛИЗ имеет вид: *fij*1+3CALL.

Для представления условных выражений введем метки, помечающие элементы записи, которые используются при выборе по условию (m_i) и дополнительные операции:

- BF – условный переход по значению «ложь» (например, *am*BF – переход к метке *m*, если значение *a* – «ложь»), если *a* = «истина», продолжается чтение записи (бинарная операция).

- BRL – безусловный переход к выражению, помеченному меткой, которая и является операндом этой унарной операции.

- DEFL – операция определения метки перед некоторым элементом ПОЛИЗ (унарная операция).

Например, представление условного выражения

x + (if a > b then 1 else 2) в ПОЛИЗ имеет вид:

xab>m_1BF1m_2BRLm_1DEFL2m_2DEFL+.

Дерево выражения представлено на рисунке 3.3. В нем появятся пустые вершины.

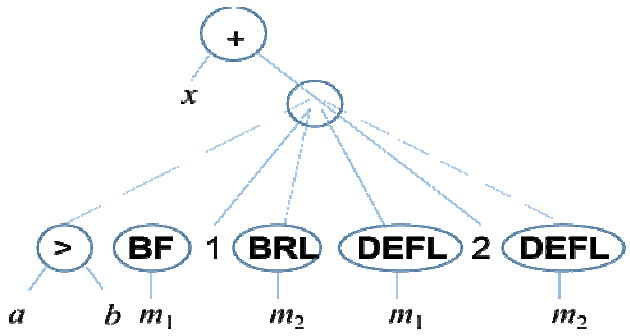

Рисунок 3.3. – Дерево выражения $xab>m_1$BF1m_2BRLm_1DEFL2m_2DEFL+

Рассмотрим другие обозначения операций языка программирования в ПОЛИЗ:

I2A (**A2I**) преобразования типа выражения, являющегося операндом этой унарной операции из целого в вещественный (из вещественного в целый).

Переходы по условию (m,r), бинарные операции (метка, проверяемое выражение):

BZ – равенства нулю;

BP – больше нуля;

BM – меньше нуля;

BPZ – не меньше нуля;

BMZ – не больше нуля;

BLBEG – начало блока;

BLEND – конец блока.

Разработка в ПОЛИЗ других конструкций языка достаточно непростое дело.

Для вычисления выражений, записанных в ПОЛИЗ существует простой алгоритм с использованием стека. Состоит он в том, что в стек записываются операнды по мере поступления, как только встречается знак операции, операнды из стека выбираются для ее выполнения, результат вновь записывается в стек. Для различения операндов и операций можно

использовать множества (операнды, одноместные операции, двуместные операции и т.д.). Схема алгоритма представлена на рисунке 3.4.

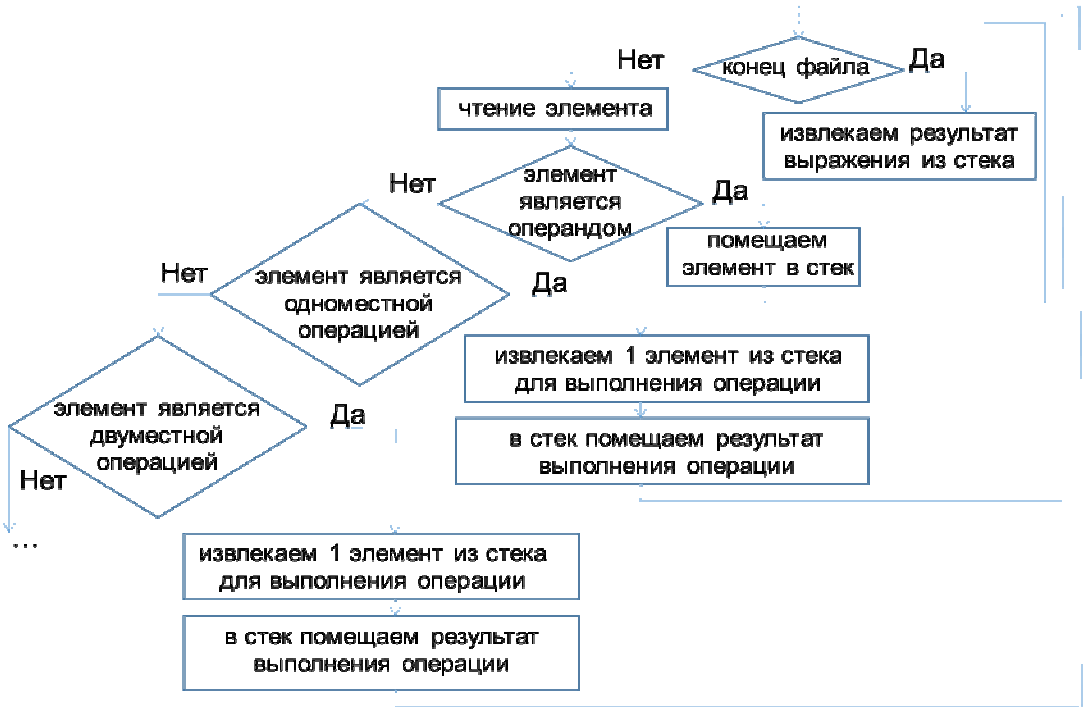

Рисунок 3.4. – Блок-схема алгоритма вычисления выражений, записанных в ПОЛИЗ

Алгоритм Дейкстры (метод стека с приоритетами)

Для перевода выражения в ПОЛИЗ рассмотрим алгоритм Дейкстры (метод стека с приоритетами). Реализуется на основе магазинного автомата преобразователя.

Каждому знаку операции присваивается числовой приоритет, приоритеты операций начинаются с 2, открывающая скобка имеет приоритет 0, закрывающая – 1.

Входная строка считывается слева направо.

Операнды по мере считывания помещаются в выходную строку.

Если приоритет знака операции больше приоритета знака в вершине стека или стек пуст знак добавляется в стек.

Если приоритет знака операции меньше или равен приоритету знака в вершине стека знак из стека выталкивается в выходную строку, после чего текущий знак помещается в стек.

Открывающаяся скобка всегда помещается в стек.

Закрывающая скобка выталкивает все знаки из стека до открывающей скобки, после чего открывающая скобка удаляется из стека (скобки в выходную строку не записываются).

После просмотра всех символов входной строки из стека выталкиваются все оставшиеся знаки.

Блок-схема алгоритма изображена на рисунке 3.5.

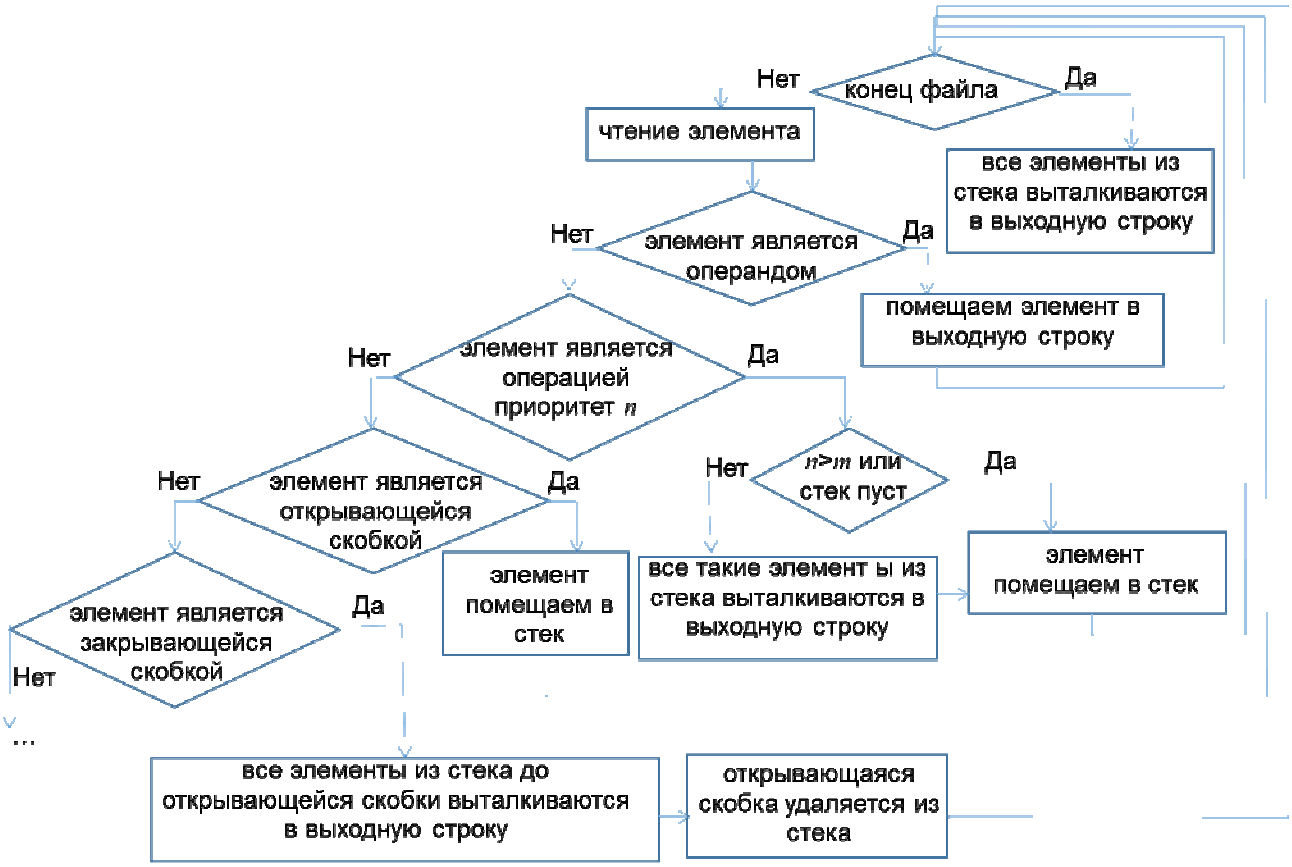

Рисунок 3.5. - Блок-схема алгоритма Дейкстры для перевода выражения в ПОЛИЗ

3.1.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 8

Отчет по лабораторной работе №8 должен содержать:

- Титульный лист.

- Задания.

- Основную часть по заданию 1.

- Основную часть по заданию 2.

- Выводы.

- Список литературы.

Основная часть отчета по заданию 1 должна содержать блок-схему алгоритма функционирования программы. Листинги результатов тестирования программы.

Основная часть отчета по заданию 2 должна содержать блок-схему алгоритма функционирования программы. Листинги результатов тестирования программы.

В выводах по работе необходимо привести оценку адекватности функционирования реализованных алгоритмов.

Контрольные вопросы для защиты отчета

1. Почему исходная программа переводится сначала в промежуточное представление, а затем в объектный код?

2. Как арифметическое выражение записывается в ПОЛИЗ? Как в ПОЛИЗ записывается унарный минус?

3. Почему в компиляторах используется для представления выражений польская инверсная запись?

4. Как производится вычисление выражения, представленного в польской инверсной записи?

5. Как графически представляется польская инверсная запись?

3.2. Лабораторная работа № 9 «Генерация промежуточного кода на основе триад»

3.2.1 Задание к лабораторной работе №9

На основе интерпретатора языка, построенного в лабораторной работе № 7, создать транслятор программы в промежуточный код, представляющий собой список триад.

3.2.2 Методические указания к выполнению лабораторной работы № 9

В компиляторах языков программирования для оптимизации программы и генерации машинного кода используется некоторый промежуточный код, который может быть записан в форме польской инверсной записи, тетрад, триад и др. Триада (тройка) - это совокупность операции и двух ее операндов, представляющая собой некоторую элементарную связку. Триады могут эффективно использоваться как для записи порядка вычисления операций в выражении (в том числе и операторов присваивания), так и для представления управляющих конструкций (условные и безусловные переходы, с использованием которых также строятся циклы), и являются разумным компромиссом между компактностью структуры данных и удобством анализа.

Список триад имеет регулярную структуру, за счет чего к нему легко применяются различные формальные методы анализа и алгоритмы преобразований (в т.ч. оптимизации кода). Каждая триада имеет вид @(op1, op2) или просто @ op1 op2, где @ - обозначение операции, op1 и op2 - первый (левый) и второй (правый) операнды. Например, выражение a + 5 будет представлено триадой +(a, 5) или + a 5.

В качестве обозначения операций могут выступать как символы, используемые для их записи, так и некоторые целочисленные значения или значения перечисления, поставленные в соответствие операциям. При этом

следует учесть, что собственно присваивание тоже рассматривается как операция. Кроме того, с точки зрения реализации распознавателя и дальнейшей оптимизации промежуточного кода, имеет смысл рассматривать получение значения константы и обращение к переменной как элементарные операции, а также предусмотреть ситуации, когда триада не соответствует никакой операции (фиктивная триада).

Операндами триады могут быть: константное значение, переменная, результат вычисления другой триады, адрес перехода в командах передачи управления (в данной работе отсутствует). В случае унарных операций первый операнд является единственным, а второй - фиктивным. Операции с большей арностью (например, выражение (+1 2 3 4 5) языка LISP) сводятся к последовательности бинарных операций. В некоторых случаях это требует соблюдения дополнительных соглашений (например, вызовы процедур и функций с различным числом параметров), использования управляющих конструкций (например, тернарная операция ?: языка C) или применения других методов, поэтому подобные случаи в данной работе не рассматриваются.

В записи триад числовые константы записываются в общепринятой форме, операнды-переменные обозначаются их именами, а результат другой триады - ссылкой на нее (например, ее номером с предшествующим символом «^»). Допустим, дана следующая программа, состоящая из двух операторов присваивания (нумерация операторов приведена условно):

1) a=1;

2) b=2*(a + 5);

Запись этой программы в виде списка триад будет следующей:

1) 1: =(a,1) //a=1;

2) 2: +(a, 5) // a+ 5

3: ×(2, ^2) //2*(a+5)

4: =(b, ^3) //b=2*(a+5);

Однако, с учетом приведенных выше замечаний об элементарных операциях, представляемых отдельными триадами, эта запись должна иметь такой вид:

1) 1: V(a, Ø) //a
2: C(1, Ø) //1
3: =(^1, ^2) //a=1;
2) 4: V(b, Ø) //b
5: C(2, Ø) //2
6: V(a, Ø) //a
7: C(5, Ø) //5
8: +(^6, ^7) //a+5
9: ×(^5, ^8) //2*(a+5)
10: =(^4, ^9) //b=2*(a+5);

Здесь как V и C обозначены операции обращения к переменной и загрузки константы соответственно, а 0 - фиктивный операнд. Необходимость подобной «детальной» записи вызвана тем, что разным компьютерам свойственны различные сочетания инструкций и режимов адресации их операндов. Эти свойства компьютеров учитываются на этапах оптимизации и генерации машинного кода.

При построении транслятора программ в промежуточный код в качестве основы можно использовать интерпретатор, разработанный в лабораторной работе № 8 (рекурсивный спуск). В этом случае необходимо модифицировать процедуры разбора для нетерминальных символов разделов I—IV заданной грамматики (в соответствии с тем же вариантом индивидуального задания). Основу грамматики составляют разделы I и II, правила которых определяют множество операций, их приоритеты и синтаксис их использования. Соответствующие процедуры должны вместо

выполнения операции добавлять в список соответствующую триаду, а возвращаемым значением (результатом работы) процедуры будет являться номер этой триады, который будет использоваться в вышестоящей процедуре в качестве операнда в записи очередной триады и т.д. Следует отметить, что если возможны правила вида A→B, то фактически никакая операция не выполняется, следовательно, новая триада в список не добавляется и процедура разбора для символа A просто возвращает то же значение, которое возвратила вызванная ею процедура для символа B. Аналогично. если в записи выражений для задания приоритетов операций используются скобки, то они влияют на порядок разбора, но операцией не являются.

Процедуры разделов III и IV осуществляют разбор лексем (имен переменных и записи констант) и реализуются, как правило, при помощи автоматной модели с использованием правил разделов V и VI. Эти процедуры образуют лексический анализатор (нижний уровень разбора) и не вызывают других процедур, поэтому они должны просто формировать и добавлять в список триады обращения к переменным и получения констант соответственно. Следует также обратить внимание на то, что в выражении в правой части оператора присваивания могут присутствовать только те переменные, которые уже определены ранее, тогда как новый идентификатор в левой части приводит к определению переменной. Следовательно, процедуры для распознавания идентификаторов в левой и в правой частях оператора присваивания будут отличаться друг от друга по реализации, но, при этом, будут формировать одинаковые триады и возвращать одинаковые значения.

Транслятор должен формировать список триад (аналогично примерам выше, с нумерацией триад, но, конечно, без комментариев) по мере разбора входной программы и по окончании работы создавать файл протокола,

содержащий этот список в виде текста. В случае ошибки в протоколе должно фиксироваться развернутое сообщение о ее причинах (аналогично предыдущей работе).

3.2.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 9

Отчет по лабораторной работе №9 должен содержать:

- Титульный лист.
- Задания.
- Основную часть.
- Выводы.
- Список литературы.

Основная часть отчета должна содержать:

- запись грамматики в нотации Бэкуса-Наура (получена в лабораторной работе №7);
- описание реализованного способа формирования и представления триад;
- протоколы работы лексического анализатора.

Контрольные вопросы для защиты отчета

1. Как можно представить в тетрадах переменную с индексами, указатель функции?

3.3. Лабораторная работа № 10 «Оптимизация промежуточного кода, на основе триад»

3.3.1 Задание к лабораторной работе №10

Построить модуль оптимизации промежуточного кода для транслятора программ.

Задание. Реализовать оптимизацию промежуточного кода (свертку) за счет исключения триад, связанных с константными вычислениями и обращениями к данным.

3.3.2 Методические указания к выполнению лабораторной работы № 10

Оптимизация программы при ее компиляции производится с целью улучшения ее характеристик — уменьшения объема используемой памяти (как для программы, так и для данных) или увеличения производительности. Обычно достижение двух этих критериев противоречит друг другу, а уменьшение объема используемой памяти для кода программы часто вынуждает использовать алгоритмы, требующие больших массивов данных. Тем не менее, есть возможность улучшения всех свойств программы за счет исключения «лишнего» кода.

Оптимизация может быть *машинно-независимой* и *машинно-зависимой*. В первом случае устраняется общая избыточность кола, во втором — используются особенности архитектуры конкретной вычислительной системы для *свертки* последовательностей инструкций в более компактные и/или быстрые.

В качестве машинно-независимой оптимизации необходимо осуществить свертку константных выражений до одной константы, таким образом, значение константного выражения будет вычислено во время трансляции. Свертку необходимо выполнять без учета возможных изменений значений переменных или их отсутствия в транслируемом фрагменте, т.е. выражения, подлежащие свертке, не должны содержать переменных. Таким образом, последовательность триад, соответствующая загрузке констант и вычислению значения константного выражения, будет свернута до одной триады загрузки константы.

Машинно-зависимая оптимизация должна заключаться в учете особенностей архитектуры компьютера, представленных в таблице 4.1.

Таблица 4.1 Допустимые режимы адресации целевой машины

Тип операции	Группа операции	Варианты допустимых сочетаний режимов адресации		Примеры	
		первый операнд	второй операнд	триада	ассемблер
Унарная	Все, кроме пересылки (загрузки)	Регистровый	-	-(^5, Ø)	inc R
		Непосредственный	-	-(8, Ø)	cpl Data
		Прямой	-	-(a, Ø)	inc (Addr)
	Пересылка (загрузка)	Регистровый	-	C(8, Ø)	mov R, Data
		Регистровый	-	V(b, Ø)	mov R, (Addr)
Бинарная	Все, кроме присваивания	Регистровый	Регистровый	+(^6, ^7)	add R, R
		Регистровый	Непосредственный	+(^6, 34)	add R, Data
	Присваивание	Прямой	Регистровый	=(a, ^4)	mov (Addr), R

Прямой режим адресации означает, что за кодом операции следует адрес, по которому производится обращение для считывания или записи данных — эти данные и являются собственно операндом. При *непосредственном* режиме адресации за кодом операции следуют *непосредственно* данные, т.е. значение операнда. Из таблицы 4.1 видно, что в данном компьютере эти режимы не совмещаются для двух операндов одной операции.

В реальных машинах приемником результата обычно является один из операндов — часто это регистр процессора или вершина стека для стековой машины. Таким образом, бинарная операция становится не трех- (операнд 1, операнд 2, результат), а двухадресной (операнд 1 и он же результат, операнд 2). В отличие от инструкций традиционных реальных компьютеров, в триадах не указывается приемник результата — результат просто будет востребован по ссылке на триаду. Поэтому в данном случае регистровый режим адресации может также означать и стековый — это зависит от типа

конкретной машины (регистровая, стековая) и, следовательно, способа реализации передачи результата одной операции для использования его в качестве операнда другой операцией (результат будет помешен либо в регистр, либо на вершину стека).

Идентификаторы (имена) переменных существуют только в исходном коде программы и в ее записи в виде триад. В машинном коде (непосредственно в инструкциях процессора) им соответствуют адреса переменных. Поэтому триаде обращения к переменной соответствует, фактически, некоторая операция подготовки адреса (актуально для некоторых типов микропроцессоров), а случаю прямой адресации будет соответствовать использование имени переменной в качестве операнда триады.

Обобщая эту информацию, получаем, что свертка может выполняться следующим образом:

1. Если единственный операнд унарной операции или второй (правый) операнд бинарной операции, кроме присваивания, является ссылкой на триаду загрузки константы, то он заменяется самой константой, а соответствующая триада загрузки удаляется.

2. Если единственный операнд унарной операции является константой, то вычисляется соответствующая операция, а триада заменяется на триаду загрузки полученного значения.

3. Если первый (левый) операнд бинарной операции, кроме присваивания, является ссылкой на триаду загрузки константы и второй (правый) операнд — константой, то вычисляется соответствующая операция и триада заменяется на триаду загрузки полученного значения, а предшествующая триада загрузки константы удаляется.

4. Если первый (левый) операнд операции присваивания является ссылкой на триаду обращения к переменной, то он заменяется именем переменной, а соответствующая триада загрузки удаляется.

Каждая триада в списке должна быть проанализирована на возможность применения к ней каждого правила, но для адекватной оптимизации промежуточного кода необходима также четкая последовательность этих действий. Так как в качестве операндов могут фигурировать ссылки только на предшествующие триады, то анализировать список триад необходимо «сверху вниз» — последовательно, от меньших номеров триад к большим. Каждую триаду необходимо рассматривать на возможность применения к ней правил в приведенном выше порядке, так как применение правила 1 может повлечь возможность применения правила 2 и т.д., но не наоборот.

Также следует обратить внимание на то, что исключить какую-либо триаду из списка можно только тогда, когда на нее не останется ни одной ссылки. Учитывая, что в рамках данной работы не ставится задача оптимизации общих подвыражений, можно утверждать, что ссылка на каждую триаду является единственной. Таким образом, исключать триады, ставшие «ненужными», можно одновременно с заменой ссылки на нее значением.

При программной реализации рекомендуется не исключать триады из списка, что приведет к изменению нумерации, а заменять их пустыми и пропускать при контрольном выводе или выводе в файл протокола. Это позволит сохранить нумерацию, что облегчает реализацию алгоритма, проверку его работоспособности и адекватности.

Работу алгоритма можно проиллюстрировать на следующем примере:

1) a=1;	1:	V(a, Ø)	//a
	2:	C(1, Ø)	//1

	3:	=(^1, ^2) //a=1;
2) b=(a+2*(5+7))*3;	4:	V(b, Ø) //b
	5:	C(2, Ø) //2
	6:	V(a, Ø) //a
	7:	C(5, Ø) //5
	8:	+(^6, ^7) //a+5
	9:	×(^5, ^8) //2*(a+5)
	10:	=(^4, ^9) //b=2*(a+5);
	11:	+(^5, ^10) //a+2*(5+7)
	12:	C(3, Ø) //3
	13:	×(^11, ^12) //(a+2*(5+7))*3
	14:	=(^4, ^13) //b=(a+2*(5+7))*3

В табл. 4.2 приведен список триад, соответствующий приведенным операторам, и показаны шаги, на которых выполняется свертка. В списке триад выделены те операнды и триады, которые подлежат свертке, а в нижней строке указан номер применяемого правила.

Таблица 4.2 Допустимые режимы адресации целевой машины

	Шаг1	Шаг 2	Шаг3	Шаг 4	Шаг 5	Шаг 6	Шаг 7	Шаг 8	Шаг 9
1	V(a, Ø)								
2	C(1, Ø)	C(1, Ø)	C(1, Ø)	C(1, Ø)	C(1, Ø)	C(1, Ø)	C(1, Ø)	C(1, Ø)	C(1, Ø)
3	=(^1, ^2)	=(a, ^2)	=(a, ^2)	=(a, ^2)	=(a, ^2)	=(a, ^2)	=(a, ^2)	=(a, ^2)	=(a, ^2)
4	V(b, Ø)	V(b, Ø)	V(b, Ø)	V(b, Ø)	V(b, Ø)	V(b, Ø)	V(b, Ø)	V(b, Ø)	V(b, Ø)
5	V(a, Ø)	V(a, Ø)	V(a, Ø)	V(a, Ø)	V(a, Ø)	V(a, Ø)	V(a, Ø)	V(a, Ø)	
6	C(2, Ø)	C(2, Ø)	C(2, Ø)	C(2, Ø)	C(2, Ø)				
7	C(5, Ø)	C(5, Ø)	C(5, Ø)						
8	C(7, Ø)	C(7, Ø)							
9	+(^7, ^8)	+(^7, ^8)	+(^7, 7)	C(12, Ø)					
10	×(^6, ^9)	×(^6, ^9)	×(^6, ^9)	×(^6, ^9)	×(^6, 12)	C(24, Ø)			
11	+(^5, ^10)	+(^5, ^10)	+(^5, ^10)	+(^5, ^10)	+(^5, ^10)	+(^5, ^10)	+(^5, 24)	+(^5, 24)	+(^5, 24)
12	C(3, Ø)	C(3, Ø)	C(3, Ø)	C(3, Ø)	C(3, Ø)	C(3, Ø)	C(3, Ø)		
13	×(^11, ^12)	×(^11, ^12)	×(^11, ^12)	×(^11, ^12)	×(^11, ^12)	×(^11, ^12)	×(^11, ^12)	×(^11, 3)	×(^11, 3)
14	=(^4, ^13)	=(^4, ^13)	=(^4, ^13)	=(^4, ^13)	=(^4, ^13)	=(^4, ^13)	=(^4, ^13)	=(^4, ^13)	=(a, ^13)

	4	1	3	1	3	1	1	4	-

Программная реализация должна быть выполнена на основе транслятора, созданного в предыдущей лабораторной работе. Трансляция в промежуточный код и его оптимизация должны представлять собой два этапа, выполняемых строго последовательно, а передача информации между ними — посредством внутреннего представления списка триад (не через файл протокола). Оптимизированный промежуточный код желательно помещать в тот же файл протокола, вслед за исходным.

3.3.3 Содержание отчета и контрольные вопросы для защиты отчета по лабораторной работе № 10

Отчет по лабораторной работе №10 должен содержать:

- Титульный лист.
- Задание.
- Основная часть.
- Выводы.
- Список литературы.

В основной части отчета необходимо отразить разработанный алгоритм оптимизации промежуточного кода и протоколы работы усовершенствованного транслятора.

Выводы по работе должны содержать обоснование адекватности реализованного алгоритма.

Контрольные вопросы для защиты отчета

1. Чем различаются триады и косвенные триады?
2. Что представляет собой виртуальная машина Java?
3. Как обрабатываются инструкции байт-кода?

4. РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

4.1 Основная литература

1. Малявко, А.А. Системное программное обеспечение. Формальные языки и методы трансляции. [Электронный ресурс]: учебное пособие в 3 частях / А.А. Малявко. - Новосибирск : НГТУ, 2010. - Ч. 1. - 104 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=228974.

2. Малявко, А.А. Системное программное обеспечение. Формальные языки и методы трансляции : [Электронный ресурс]: учебное пособие : в 3-х ч. / А.А. Малявко. - Новосибирск : НГТУ, 2011. - Ч. 2. Синтаксический анализ. - 160 с. Режим доступа:
http://biblioclub.ru/index.php?page=book&id=228973.

3. Малявко, А.А. Системное программное обеспечение. Формальные языки и методы трансляции [Электронный ресурс]: учебное пособие. В 3 ч. / А.А. Малявко. - Новосибирск : НГТУ, 2012. - Ч. 3. - 120 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=228888.

4.2 Дополнительная литература

1. Малявко А.А. Формальные языки и компиляторы [Электронный ресурс]: учебник/А.А. Малявко – Электрон. текстовые дан. – Новосибирск: Изд-во НГТУ, 2014. - 431 с. – Режим доступа: http://biblioclub.ru/index.php?page=book_view_red&book_id=436055

2. Теория и реализация языков программирования [Электронный ресурс]: / В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г. Фуругян – Электрон. текстовые дан. – Москва : Национальный открытый университет «ИНТУИТ», 2007. - 323 с. – Режим доступа: http://biblioclub.ru/index.php?page=book_view_red&book_id=234669

3. Введение в теорию языков и компиляторов [Электронный ресурс]: учеб.пособие / Л.Г. Гагарина, Е.В. Кокорева - Электрон. текстовые дан. – Москва : ИД ФОРУМ: ИНФРА-М, 2011. - 176 с. – Режим доступа:

http://znanium.com/bookread2.php?book=265617

4. Delpfi: программирование в примерах и задачах: Практикум / Г.М. Эйдлина, К.А. Милорадов. - М.: ИЦ РИОР: НИЦ Инфра-М, 2012. - 116 с. http://www.znanium.com/bookread.php?book=319046

5. Программирование на языке высокого уровня. Программир. на языке C++ [Электронный ресурс]: учебн. пособие / Т.И.Немцова [и др.]; под ред. Л.Г.Гагариной – Электрон. текстовые дан. - Москва : ИД ФОРУМ: ИНФРА-М, 2012. - 512 с. – Режим доступа: http://www.znanium.com/bookread.php?book=244875

6. Морохин, Д.В. Основы теории трансляции [Электронный ресурс]: лабораторный практикум / Д.В. Морохин ; Поволжский государственный технологический университет. - Йошкар-Ола : ПГТУ, 2015. - 72 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=439273

7. Сперанский, Д.В. Лекции по теории экспериментов с конечными автоматами [Электронный ресурс]: учебное пособие / Д.В. Сперанский. - М. : Интернет-Университет Информационных Технологий, 2011. - 288 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=233287

8. Серебряков, В.А. Теория и реализация языков программирования [Электронный ресурс] / В.А. Серебряков, М.П. Галочкин, Д.Р. Гончар, М.Г. Фуругян. - М. : Интернет-Университет Информационных Технологий, 2007. - 323 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=234669

9. Сухомлин, В.А. Введение в программирование [Электронный ресурс]: учебное пособие / В.А. Сухомлин, И.Ю. Баженова. - М. : Интернет-Университет Информационных Технологий, 2007. - 327 с. Режим доступа: http://biblioclub.ru/index.php?page=book&id=232982

10. Вирт, Н. Построение компиляторов. [Электронный ресурс] — Электрон. дан. — М. : ДМК Пресс, 2010. — 192 с. — Режим доступа:

http://e.lanbook.com/book/1262