**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

филиал федерального государственного бюджетного образовательного учреждения высшего образования «Национальный исследовательский университет «МЭИ» в г. Смоленске

Кафедра вычислительной техники

по курсу: «Трансляторы программных языков»

Тема: «Проектирование трансляторов»

Группа: ПО1-19

Студент: Милославский С.А.

Преподаватель: Курылев В. А.

г. Смоленск, 2022

**ОГЛАВЛЕНИЕ**

[введение 3](#_Toc121991220)

[1 постановка задачи 5](#_Toc121991221)

[2 Формальная модель задачи 6](#_Toc121991222)

[2.1 Описание синтаксиса языка программирования 6](#_Toc121991223)

[2.1.1 Формальная грамматика 6](#_Toc121991224)

[2.1.2 Расширенная форма Бэкуса-Наура (РБНФ) 8](#_Toc121991225)

[2.2 Классы лексем 10](#_Toc121991226)

[2.3 Общая структура компилятора 10](#_Toc121991227)

[3 Лексических анализ 13](#_Toc121991228)

[3.1 Диаграмма состояний 13](#_Toc121991229)

[4 Синтаксический анализ 15](#_Toc121991230)

[4.1 Схемы алгоритма синтаксического анализатора 15](#_Toc121991231)

[5 промежуточный код 17](#_Toc121991232)

[6 Интерпретатор программы 18](#_Toc121991233)

[7 тестирование 19](#_Toc121991234)

[8 руководство пользователя 20](#_Toc121991235)

[заключение 21](#_Toc121991236)

[список литературы 22](#_Toc121991237)

[приложение А – техническое задание 24](#_Toc121991238)

[Приложение Б – Код программы 27](#_Toc121991239)

введение

Данная курсовая работа направлена на изучение проектирования трансляторов программных языков. В рамках работы осуществляется разработка языка программирования и транслятора, переводящего его в промежуточный язык.

Актуальность работы заключается в том, что трансляторы программных языков наряду с операционными системами являются важнейшим программным обеспечением. Несмотря на огромное количество разработанных языков и трансляторов для них новые языки продолжают появляться. Некоторые из них являются языками общего назначения и подходят для выполнения самых разнообразных задач. Другие занимают определенную нишу и являются лучшими только в ней. Сообщество находится в постоянном поиске самых удобных и оптимальных инструментов для удовлетворения своих потребностей. Поэтому с задачей разработки транслятора для программного языка может столкнуться каждый программист. Например, это может быть интерпретатор скриптового языка для прикладного ПО или даже участие в разработке транслятора одного из популярных на данный момент языков программирования, который используют в свой работе большое количество других программистов. Кроме того, понимание принципов работы трансляторов может помочь при проектировании ПО других типов.

Целью курсовой работы является закрепление теоретического материала и приобретение необходимого опыта разработки трансляторов.

Для достижения поставленной цели были сформулированы следующие задачи:

1. Изучить, используя лекции, литературу и Интернет, необходимую для построения транслятора теоретическую информацию.

2. Описать язык программирования в соответствии с вариантом и составить для него формальную грамматику.

3. Составить техническое задание на программу транслятора.

4. Разработать программу транслятора, которая должна состоять из лексического анализатора, синтаксического анализатора и генератора промежуточного кода.

5. Протестировать транслятор на контрольных примерах.

Объектом исследования является проектирование трансляторов программных языков.

Предметом исследования является разработка транслятора для простого языка программирования в соответствии с вариантом.

1. постановка задачи

Разработать компилятор модельного языка, выполнив следующие действия.

1) В соответствии с номером варианта составить формальное описание модельного языка программирования с помощью: формальных грамматик и РБНФ.

2) Написать пять содержательных примеров программ, раскрывающих особенности конструкций учебного языка программирования, отразив в этих примерах все его функциональные возможности.

3) Составить таблицы лексем и диаграмму состояний с действиями для распознавания и формирования лексем языка.

4) По диаграмме с действиями написать функцию сканирования текста входной программы на модельном языке.

5) Разработать программное средство, реализующее лексический анализ текста программы на входном языке.

6) Реализовать синтаксический анализатор текста программы на модельном языке методом рекурсивного спуска.

1. Формальная модель задачи
   1. Описание синтаксиса языка программирования
      1. Формальная грамматика

Описание синтаксиса языка программирования методом формальной грамматики представлена ниже.

* + 1. Расширенная форма Бэкуса-Наура (РБНФ)

Мета язык, предложенный Бэкусом и Науром, использует следующие обозначения:

* символ «::=» отделяет левую часть правила от правой (читается: «определяется как»);
* нетерминалы обозначаются произвольной символьной строкой, заключенной в угловые скобки «<» и «>»;
* терминалы — это символы, используемые в описываемом языке;
* правило может определять порождение нескольких альтернативных цепочек, отделяемых друг от друга символом вертикальной черты «|» (читается: «или»).

Для повышения удобства и компактности описаний, в РБНФ вводятся следующие дополнительные конструкции (метасимволы):

* квадратные скобки «[» и «]» означают, что заключенная в них синтаксическая конструкция может отсутствовать;
* фигурные скобки «{» и «}» означают повторение заключенной в них синтаксической конструкции ноль или более раз;
* сочетание фигурных скобок и косой черты «{/» и «/}» используется для обозначения повторения один и более раз;
* круглые скобки «(» и «)» используются для ограничения альтернативных конструкций.

Описание синтаксиса языка программирования методом РБНФ представлена ниже.

<программа>::= <блок> *end*

<блок>::= {/ (<описание> | <оператор>) /}

<описание>::= *dim* <идентификатор> {, <идентификатор> } <тип>;

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

<тип>::= % | ! | @ | $

<оператор>::= <присваивания с точкой с запятой> | <условный> | <фиксированного\_цикла> | <условного\_цикла> | <ввода> | <вывода>

<присваивание>::= <идентификатор> *as* <выражение>

<присваивание с точкой с запятой>::= <присваивание>;

<выражение>::= <операнд>{<операции\_группы\_отношения> <операнд>}

<операции\_группы\_отношения>:: = < > | = | < | <= | > | >=

<операнд>::= <слагаемое> {<операции\_группы\_сложения> <слагаемое>}

<операции\_группы\_сложения>:: = + | - | *or*

<слагаемое>::= <множитель> {<операции\_группы\_умножения> <множитель>}

<операции\_группы\_умножения>::= \* | / | *and*

<множитель>::= <степень> {<операция\_степени> <степень>}

<степень>::= <идентификатор> | <число> | <строковая константа> | <логическая\_константа> | <унарная\_операция> <множитель> | (<выражение>)

<операция\_степени>::= ^

<унарная\_операция>::= *not*

<число>::= <целое> | <действительное>

<целое>::= <двоичное> | <восьмеричное> | <десятичное> | <шестнадцатеричное>

<двоичное>::= {/ 0 | 1 /} (*B* | *b*)

<восьмеричное>::= {/ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 /} (*O* | *o*)

<десятичное>::= {/ <цифра> /} [*D* | *d*]

<шестнадцатеричное>::= <цифра> {<цифра> | *A* | *B* | *C* | *D* | *E* | *F* | *a* | *b* | *c* | *d* | *e* | *f*} (*H* | *h*)

<действительное>::= <числовая\_строка> <порядок> | [<числовая\_строка>] . <числовая\_строка> [порядок]

<числовая\_строка>::= {/ <цифра> /}

<порядок>::= ( E | e )[+ | -] <числовая\_строка>

<строковая\_константа>::=”{<буква>}”

<логическая\_константа>::= *true* | *false*

<условный>::= *if* <выражение> *then* <блок> { else if <выражение> then <блок> } [ *else* <блок>] endif

<фиксированного\_цикла>::= *for* <присваивания> *to* <выражение> *do* <блок> endfor

<условного\_цикла>::= *while* <выражение> *do* <блок> endwhile

<ввода>::= *read* (<идентификатор> {, <идентификатор> });

<вывода>::= *write* (<выражение> {, <выражение> });

<комментарий>::=\\* <строковая\_константа>

<многострочный комметнтарий>::=\\* <строковая\_константа> \*\

* 1. Классы лексем

Для заданного языка программирования были заданы следующие классы лексем:

* ключевое слово: [«dim», «as», «if», «then», «else», «endif», «for», «to», «do», «while»];
* типы: [«%», «!», «@», «$»];
* ввод: [«read»];
* вывод: [«write»];
* разделители: [«(», «)», «,», «.», «;»];
* арифметические операторы: [«+», «-», «\*», «/», «^»];
* операторы сравнения: [«>», «<», «<>», «=», «>=», «<=»];
* логические операторы №1: [«not»];
* логические операторы №2: [«and», «or»];
* константы;
* идентификаторы.
  1. Общая структура компилятора

Компилятор – это программа, которая осуществляет перевод исходной программы в эквивалентную ей объектную программу на языке машинных команд или языке ассемблере.

Основные функции компилятора:

1) проверка исходной цепочки символов на принадлежность к входному языку;

2) генерация выходной цепочки символов на языке машинных команд или ассемблере.

Процесс компиляции состоит из двух основных этапов: синтеза и анализа.

На этапе анализа выполняется распознавание текста исходной программы и заполнение таблиц идентификаторов. Результатом этапа служит некоторое внутреннее представление программы, понятное компилятору.

На этапе синтеза на основании внутреннего представления программы и информации, содержащейся в таблице идентификаторов, порождается текст результирующей программы. Результатом этого этапа является объектный код.

Данные этапы состоят из более мелких этапов, называемых фазами. Состав фаз и их взаимодействие зависит от конкретной реализации компилятора. Но в том или ином виде в каждом компиляторе выделяются следующие фазы:

1) лексический анализ;

2) синтаксический анализ;

3) семантический анализ;

4) подготовка к генерации кода;

5) генерация кода.

Процесс последовательного чтения компилятором данных из внешней памяти, их обработки и помещения результатов во внешнюю память, называется проходом компилятора.

По количеству проходов выделяют одно-, двух-, трех- и многопроходные компиляторы. В данном пособии предлагается схема разработки трехпроходного компилятора, в котором первый проход – лексический анализ, второй – синтаксический, семантический анализ и генерация внутреннего представления программы, третий – интерпретация программы.

Общая схема работы компилятора представлена на рисунке 1.

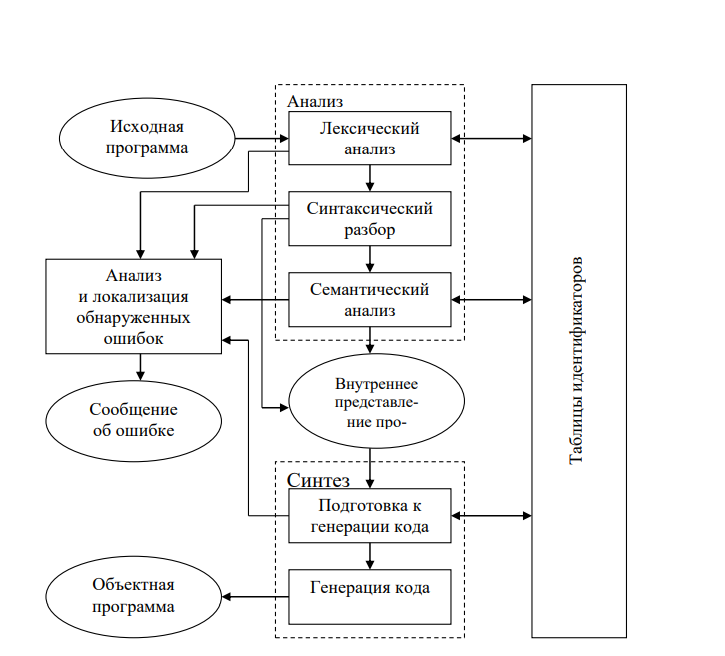


Рисунок 1 – Общая схема работы компилятора

1. Лексических анализ

Лексический анализатор (ЛА) – это первый этап процесса компиляции, на котором символы, составляющие исходную программу, группируются в отдельные минимальные единицы текста, несущие смысловую нагрузку – лексемы.

Задача лексического анализа - выделить лексемы и преобразовать их к виду,

удобному для последующей обработки. ЛА использует регулярные грамматики. ЛА необязательный этап компиляции, но желательный по следующим причинам:

1) замена идентификаторов, констант, ограничителей и служебных слов лексемами делает программу более удобной для дальнейшей обработки;

2) ЛА уменьшает длину программы, устраняя из ее исходного представления несущественные пробелы и комментарии;

3) если будет изменена кодировка в исходном представлении программы, то это отразится только на ЛА.

В процедурных языках лексемы обычно делятся на классы:

1) служебные слова;

2) ограничители;

3) числа;

4) идентификаторы.

Каждая лексема представляет собой пару чисел вида (n, k), где n – номер таблицы лексем, k - номер лексемы в таблице.

Входные данные ЛА – текст транслируемой программы на входном языке.

Выходные данные ЛА – файл лексем в числовом представлении.

* 1. Диаграмма состояний

Для удобства построения ЛА была построена диаграмма состояний, представленная на рисунке 2.

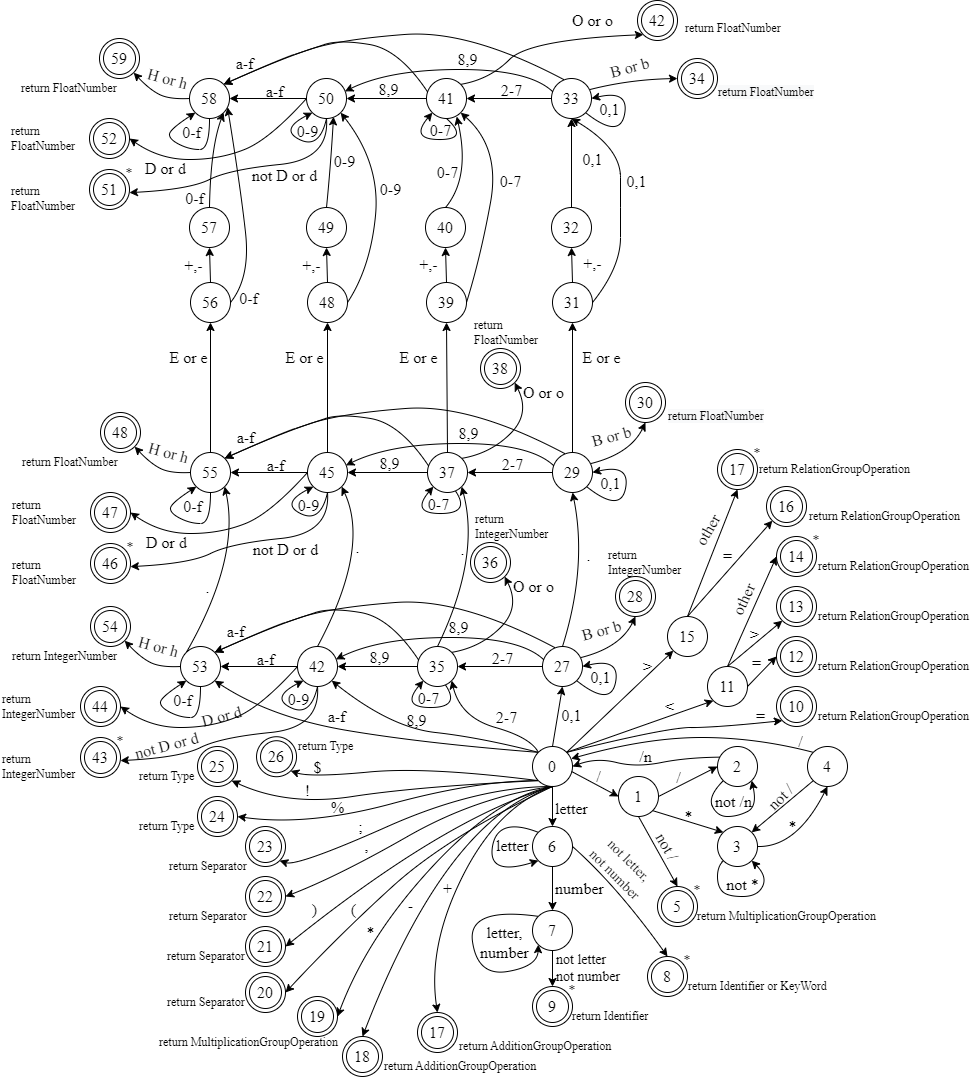


Рисунок 2 – Диаграмма состояний

1. Синтаксический анализ

Задача синтаксического анализатора (СиА) – провести разбор текста программы, сопоставив его с эталоном, данным в описании языка. Для синтаксического разбора используются контекстно-свободные грамматики (КС-грамматики).

Один из эффективных методов синтаксического анализа – метод рекурсивного спуска. В основе метода рекурсивного спуска лежит левосторонний разбор строки языка. Исходной сентенциальной формой является начальный символ грамматики, а целевой – заданная строка языка. На каждом шаге разбора правило грамматики применяется к самому левому нетерминалу сентенции. Данный процесс соответствует построению дерева разбора цепочки сверху вниз (от корня к листьям).

Метод рекурсивного спуска реализует разбор цепочки сверху вниз следующим образом. Для каждого нетерминального символа грамматики создается своя процедура, носящая его имя. Задача этой процедуры – начиная с указанного места исходной цепочки, найти подцепочку, которая выводится из этого нетерминала. Если такую подцепочку считать не удается, то процедура завершает свою работу вызовом процедуры обработки ошибок, которая выдает сообщение о том, что цепочка не принадлежит языку грамматики и останавливает разбор. Если подцепочку удалось найти, то работа процедуры считается нормально завершенной и осуществляется возврат в точку вызова. Тело каждой такой процедуры составляется непосредственно по правилам вывода соответствующего нетерминала, при этом терминалы распознаются самой процедурой, а нетерминалам соответствуют вызовы процедур, носящих их имена.

* 1. Схемы алгоритма синтаксического анализатора

1. промежуточный код

Данный этап реализуется с помощью three-address code (TAC).

TAC представляет собой последовательность операторов формы A-=B op C, где A, B, C — это имена, определенные программистом, константы или временные имена, сгенерированные компилятором, op представляет оператор, который может быть арифметические операторы с константой или с плавающей запятой, либо данные с логическим значением, либо логический оператор. Причина названия «трехадресный код» заключается в том, что каждый оператор обычно включает три адреса, два для операндов и один для результата.

В ТАС не более трех адресов определяют любой оператор. Два адреса для операнда и один для результата.

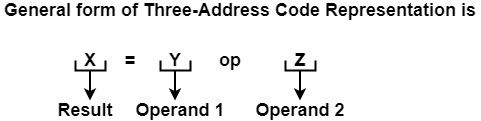


Рисунок 3

Следовательно, op является оператором. Одновременно допускается только одна операция в правой части выражения.

Например — выражение a = b + c + d можно преобразовать в следующий ТАС.

t1 = b + c

t2 = t1 + d

a = t2

где t1 и t2 — временные переменные, сгенерированные компилятором.

1. Интерпретатор программы

Интерпретатор — это компьютерная программа, которая непосредственно выполняет инструкции, написанные на языке программирования или языке сценариев, без необходимости их предварительной компиляции в программу на машинном языке.

На данном этапе интерпретатор используют следующую стратегию преобразует исходный код в некоторое эффективное промежуточное представление или объектный код и немедленно выполняет его.

При реализации этого этапа был создан класс Interpreter, который считывает и выполняет сгенерированный промежуточный код.

1. тестирование
2. руководство пользователя

При запуске программы пользователю предоставляется интуитивно понятный интерфейс программы, в котором можно сразу писать программный код разрабатываемого языка или открыть файл с форматом as.

Программа считывает код программы из компонента или из файла с форматом as.

Затем программа выполняет лексический анализ кода.

Производится синтаксический и семантический анализ, если в ходе данного процесса будет найдена ошибка в коде программы, то работа программы завершится и в консоль будет выведено сообщение об ошибке.

Если во время анализа не было найдено ошибок программа переводит код программы в постфиксную запись для преобразования его в промежуточный код.

Далее следует выполнение промежуточного кода и вывод результата программы в консоль.

заключение

В данной курсовой работе была поставлена задача разработки компилятора к заданной грамматике.

В ходе решения задачи:

* была разработана программа, которая производит анализ исходного текста и выдает сообщения об ошибках, если они есть;
* было произведено кодирование на языке программирования C# в среде объектно-ориентированного программирования Visual Studio и тестирование, в ходе которого было выявлено, что программа решает задачу корректно и устойчиво работает на тестовом наборе данных;
* была составлена пояснительная записка;
* был закреплён соответствующий лекционный материал дисциплины, приобретены практические навыки проектирования программных систем с использованием структурно-объектного подхода.

список литературы

1. Афанасьев А.Н. Формальные языки и грамматики: Учебное пособие. – Ульяновск: УлГТУ, 1997. – 84с.
2. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. – М.: Изд. дом «Вильямс», 2001. – 768с.
3. Братчиков И.Л. Синтаксис языков программирования / Под ред. С.С. Лаврова. – М.: Наука, 1975. - 262с.
4. Вайнгартен Ф. Трансляция языков программирования / Под ред. Мартынюка В.В.- М.: Мир, 1977. - 192с.
5. Вильямс А. Системное программирование в Windows 2000 для профессионалов. – СПб.: Питер, 2001. – 624с.
6. Волкова И.А., Руденко Т.В. Формальные языки и грамматики. Элементы теории трансляции. – М.: Диалог-МГУ, 1999. – 62с.
7. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. – СПб: Питер, 2001. – 736с.
8. Грис Д. Конструирование компиляторов для цифровых вычислительных машин: Пер. с англ. – М.: Мир, 1975. – 544с.
9. Дворянкин А.И. Основы трансляции: Учебное пособие. – Волгоград: ВолгГТУ, 1999. – 80с.
10. Жаков В.И., Коровинский В.В., Фильчаков В.В. Синтаксический анализ и генерация кода. – СПб.: ГААП, 1993. – 26с.
11. Ишакова Е.Н. Теория формальных языков, грамматик и автоматов: Методические указания к лабораторному практикуму. – Оренбург: ГОУ ВПО ОГУ, 2004. – 54с.
12. Компаниец Р.И., Маньков Е.В., Филатов Н.Е. Системное программирование. Основы построения трансляторов. – СПб.: Корона принт, 2000. – 256с.
13. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. - М.: Мир, 1979. - 654с.
14. Пантелеева И.А. Методы трансляции: Конспект лекций. – Новосибирск: Изд-во НГТУ, 1998. – Ч.2. – 51с.
15. Пратт Т., Зелковиц М. Языки программирования: разработка и реализация / Под ред. А. Матросова. – СПб: Питер, 2002. – 688с.
16. Рейуорд-Смит В. Теория формальных языков. Вводный курс: Пер. с англ. – М.: Радио и связь, 1988. – 128с.
17. Серебряков В.И. Лекции по конструированию компиляторов. – М.: МГУ, 1997. – 171с.
18. Соколов А.П. Системы программирования: теория, методы, алгоритмы: Учеб. пособие. – М.: Финансы и статистика, 2004. – 320с.
19. Федоров В.В. Основы построения трансляторов: Учебное пособие. – Обнинск: ИАТЭ, 1995. – 105с.
20. Хантер Р. Проектирование и конструирование компиляторов: Пер. с англ. – М.: Финансы и статистика, 1984. – 232с

приложение А – техническое задание

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

филиал федерального государственного бюджетного образовательного учреждения высшего образования «Национальный исследовательский университет «МЭИ» в г. Смоленске

Кафедра вычислительной техники

по курсу: «Трансляторы программных языков»

**Техническое задание**

Листов

Группа: ПО1-19

Студент: Милославский С.А.

Преподаватель: Курылев В. А.

г. Смоленск, 2022

**1. Введение**

Транслятор, работающий как надстройка над языком программирования.

**2. Основания для разработки**

2.1 Транслятор разрабатывается на основании рабочей программы курса «Трансляторы программных языков» для ПО1-19, утвержденной кафедрой Вычислительной Техники Филиала федерального государственного бюджетного образовательного учреждения высшего образования «Национальный исследовательский университет «МЭИ» в г. Смоленске.

2.2 Наименование работы: «Проектирования транслятора».

2.3 Исполнитель: Милославский С. А.

2.4 Соисполнители: нет.

**3. Назначение**

Программа предназначена для проверки кода и преобразования его в промежуточный код для дальнейшего интерпретирования.

**4. Требования к программе или к программному изделию**

4.1. Требование к функциональным характеристикам.

4.1.1. Программа должна обеспечивать возможность выполнения следующих функций:

* проверку ошибок;
* вывод дескрипторного текста;
* вывод постфиксной записи исходного кода;
* вывод таблиц идентификаторов;
* перевод кода в код другого языка программирования.

4.1.2. Организация входных и выходных данных

* Входные данные: поступают из файла в формате as или читаются с формы.
* Выходные данные: отображаются на экране.

4.2. Требования к надежности

Предусмотреть контроль вводимой информации.

4.3. Требования к составу и параметрам технических средств.

Система должна работать на IBM- совместимых персональных компьютерах Рекомендуемая конфигурация:

* Тип процессора: Pentium и выше.
* Объём ОЗУ: 256 МБ
* Объём свободного места на жестком диске: 30 МБ

4.4. Требования к программной совместимости.

Программа должна работать под управлением операционной системы семейства Windows NT (Windows XP, Vista, 7, 8, 8.1, 10).

**5. Требование к программной документации**

5.1. Разрабатываемые модули должны быть самодокументированы, т.е. текст программы должен содержать комментарии.

5.2. Расчетно-пояснительная записка

**6. Стадии и этапы разработки**

1. Постановка задачи — Необходимо определить цель задачи, ее содержание и общий подход к решению.

2. Анализ задачи и моделирование — определяются исходные данные и результат, выявляются ограничения на их значения, выполняется формализованное описание задачи и построение (выбор) математической модели, пригодной для решения на компьютере.

3. Разработка или выбор алгоритма решения задачи — выполняется на основе ее математического описания.

4. Проектирование общей структуры программы — формируется модель решения с последующей детализацией и разбивкой на подпрограммы, определяется «архитектура» программы, способ хранения информации (набор переменных, массивов и т. п.).

5. Кодирование — запись алгоритма на языке программирования.

6. Отладка и тестирование программы. Под отладкой понимается устранение ошибок в программе. Тестирование позволяет вести их поиск и, в конечном счете, убедиться в том, что полностью отлаженная программа дает правильный результат.

6. Анализ результатов

7. Публикация результатов работы, передача заказчику для эксплуатации.

8. Сопровождение программы — включает консультации представителей заказчика по работе с программой и обучение персонала.

Недостатки и ошибки, замеченные в процессе эксплуатации, должны устраняться.

Таблица 1 – Стадии и этапы разработки

|  |  |  |
| --- | --- | --- |
| Планируемая работа | Трудоемкость % | Сроки выполнения(день) |
| 1. Постановка задачи | 10 | 2 |
| 2. Анализ задачи и моделирование | 20 | 5 |
| 3. Разработка или выбор алгоритма решения задачи | 17 | 5 |
| 4. Проектирование общей структуры программы | 8 | 2 |
| 5. Кодирование | 20 | 9 |
| 7. Анализ результатов | 10 | 3 |
| 8. Публикация работы, передача заказчику | 9 | 4 |
| 9.Сопровождение программы | 6 | 365 |

Приложение Б – Код программы