Федеральное агентство по образованию
Озерский технологический институт (филиал)
ГОУ ВПО «Московский инженерно-физический институт
(государственный университет)»

Кафедра прикладной математики

Теория языков программирования и методы трансляции

Курсовое проектирование

Учебно-методическое пособие

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ МИФИ

Озерск
2008

УДК 681.31

Теория языков программирования и методы трансляции. Курсовое проектирование. Учебно-методическое пособие. Подготовил Вл. Пономарев. Озерск: ОТИ МИФИ, 2008. — 68 с.

Методическое пособие предназначено для студентов специальности 230105 и включает в себя основные требования к выполнению, содержанию и оформлению курсового проекта по дисциплине «теория языков программирования и методы трансляции», а также рекомендации по его защите.

Рецензенты:
1) Зав. кафедрой Информатики ОФ ЮУрГУ, к.т.н. Мосунов С.Е.
2)

Содержание

Обозначения и сокращения

ActiveX - технология Microsoft, основанная на COM

ATL - Active Template Library (библиотека активных шаблонов)

COM - Component Object Model (модель многокомпонентных объектов)

MDI - Multi Document Interface (многодокументный интерфейс)

MS VB - Microsoft Visual Basic 6.0

MS VC++ - Microsoft Visual C++ 6.0

SDI - Single Document Interface (одно-документный интерфейс)

КС - контекстно-свободный (язык, грамматика)

МП автомат - автомат с магазинной памятью

1 Основные положения

1.1 Цели и задачи курсового проектирования

Целями курсового проектирования являются:

- закрепление знаний, полученных в ходе теоретического обучения по дисциплине «теория языков программирования и методов трансляции»;
- приобретение навыков практического программирования систем, построенных с использованием теории формальных языков;
- изучение отдельных разделов предметной области, не вошедших в программу теоретического обучения, формирование навыка поиска информации по конкретной теме, её анализа и использования для решения задачи;
- подготовка к выполнению дипломного проекта.

Курсовой проект позволяет сформировать способности будущего специалиста к самостоятельному решению практических задач и инженерных проблем с использованием теоретических положений, а также знаний и умений, полученных в ходе обучения программированию.

Задачами курсового проектирования являются:

- разработка лексических и синтаксических грамматик;
- разработка представления таблиц токенов;
- разработка представления таблиц символов программы;
- проектирование и практическое программирование конечных автоматов для разбора лексем во время лексического анализа;
- проектирование и практическое программирование автоматов с магазинной памятью для синтаксического разбора;

- выбор способа промежуточного представления программы на этапе анализа и его программная реализация;

- проектирование методов генерации кода на выходном языке.

1.2 Тематика курсового проектирования

В качестве темы для курсового проектирования по дисциплине «теория языков программирования и методы трансляции» студентам могут быть предложены для разработки разные по характеристикам программные компоненты, в целом называемые трансляторами. Трансляторы можно условно разделить на следующие группы:

1) компиляторы; это программы, которые переводят текст на заданном входном языке программирования высокого уровня в эквивалентную программу на языке ассемблер x86 или эквивалентную объектную программу заданного формата;

2) ассемблеры; это программы, которые переводят текст на заданном входном языке низкого уровня в эквивалентную объектную программу заданного формата;

3) кросс-компиляторы; это программы, которые переводят текст на заданном входном языке высокого уровня в эквивалентную программу на выходном языке низкого уровня или в объектный код в той или иной форме для другой целевой машины;

4) интерпретаторы; это программы, которые проверяют лексическую и синтаксическую правильность текста на заданном входном языке высокого уровня и выполняют предписанные программой действия;

5) конвертеры; это программы, которые переводят текст на заданном входном языке программирования высокого уровня в эквивалентную программе на другом языке высокого уровня;

6) контроллеры; это программы, которые проверяют лексическую и синтаксическую правильность текста на некотором входном языке, таком, как HTML, XML, SQL и т.п.;

7) корректоры; это программы, которые проверяют лексическую и синтаксическую правильность текста на заданном входном языке высокого уровня и выполняют преобразования входного текста с целью придать ему предписанную некоторыми соглашениями форму, например, соответствие правилам оформления программных модулей, принятыми в ОТИ МИФИ [1];

8) сопутствующие проекты, связанные с тематикой дисциплины. Это программы, выполняющие функции, аналогичные функциям лексических и синтаксических анализаторов, а также вспомогательные средства трансляторов, такие, как шаблонный поиск в тексте с использованием регулярных выражений, интерпретаторы команд, визуальные редакторы окон, библиотеки стандартных функций и т.п.;

9) исследовательские проекты. Они включают в себя разработку различных программных компонентов, связанных с языками программирования и средами разработки. Примерами могут служить интерпретация языка UML для автоматической генерации программы на языке высокого уровня, средства визуального программирования.

В состав проекта могут быть включены научно-исследовательские работы, в которых студент принимал участие по линии учебно-исследовательской работы студентов (УИРС), в научных учреждениях и на производстве при прохождении практики. Объём этих работ устанавливается руководителями проекта.

В случаях, когда сложность проекта достаточно велика и (или) необходима подробная разработка тем, кафедра имеет право выдавать комплексные задания на курсовое проектирование группе студентов. Такими заданиями могут быть проекты по соз

данию перспективных разработок в области языков программирования и сред разработки программного обеспечения.

В связи со сложностью проектирования реального программного компонента данной категории в курсовых проектах допускаются значительные упрощения и ограничения лексики, синтаксиса и семантики входных языков программирования. При выдаче задания на курсовое проектирование задается примерный синтаксис входного языка и состав типов данных, которые уточняются руководителем курсового проекта совместно со студентом как в начале работы студента над проектом, так и в ходе самого проектирования.

Упомянутые упрощения и ограничения должны быть разными для разных типов трансляторов. Например, для компиляторов и интерпретаторов могут быть приняты соглашения об ограничении типов данных только целочисленными и скалярными, а операций - только числовыми. Для конвертеров ограничения, как правило, незначительные, и касаются составных типов данных. Наименьшие ограничения в отношении лексики и синтаксиса входного языка накладываются на контроллеры и корректоры.

Кроме того, разрабатываемые трансляторы, как правило, не включают в себя библиотеки функций, которые сами по себе могут являться предметом курсового проектирования.

В качестве входных и выходных языков проектируемых трансляторов используются широко известные языки, такие, как Си, Pascal, BASIC, FORTRAN, Ada, LISP, LOGO, PHP а также ассемблеры и кросс-ассемблеры.

1.3 Руководство проектом

Общее методическое руководство курсовым проектированием по дисциплине «теория языков программирования и методы

трансляции» осуществляется кафедрой прикладной математики, а непосредственно курсовым проектом - его руководителем.

Кафедра определяет требования к содержанию курсового проекта, контролирует ход его подготовки, обеспечивает студентов методическими материалами, информирует проректора по учебной работе о выполнении графиков выполнения курсовых проектов и степени их готовности и осуществляет подбор руководителей.

По предложению руководителей кафедра может пригласить консультантов по отдельным разделам курсового проекта. Консультантами могут быть преподаватели высших учебных заведений и высококвалифицированные инженеры и специалисты.

Руководитель курсового проекта выдает задание на проектирование, согласовывает план и график выполнения курсового проекта, оказывает методическую помощь в подборе литературы, справочных материалов, консультирует студента, дает письменный отзыв на курсовой проект.

При невыполнении студентом графика по представлению руководителя кафедра имеет право не допустить работу к защите. В этом случае заведующий кафедрой после ознакомления с отзывом руководителя и содержанием курсового проекта принимает решение о допуске проекта к защите. При этом вопрос рассматривается коллегиально на заседании кафедры с участием студента и руководителя курсового проекта.

1.4 Структура и содержание курсового проекта

Курсовой проект включает в себя:

1) демонстрационную версию программного продукта;

2) расчетно-пояснительную записку;

3) компьютерную презентацию.

Составными частями расчетно-пояснительной записки являются:

1) титульный лист;

2) задание на курсовое проектирование;

3) аннотация на русском языке (одна страница);

4) содержание;

5) список обозначений и сокращений (при необходимости);

6) основной текст;

7) список использованных источников;

8) приложения.

Примечание: дискета или компакт-диск с демо-версией помещается в специальный бумажный карман на задней обложке пояснительной записки.

Объем расчетно-пояснительной записки составляет от 30 до 40 страниц текста (без учета приложений), выполненного в соответствии с требованиями ЕСКД. Руководящим материалом к оформлению расчетно-пояснительной записки служит методическое пособие [2].

Титульный лист расчетно-пояснительной записки оформляется по установленной в ОТИ МИФИ форме (приложение А).

Аннотация не считается разделом и не включается в содержание. Пример выполнения аннотации приведен в приложении Б.

Пример по содержанию пояснительной записки приведен в приложении В.

Если в тексте неоднократно встречаются сокращения и обозначения (больше пяти), после пункта «Содержание» добавляется раздел «Обозначения и сокращения». Обозначения и сокращения формируются в виде списка в алфавитном порядке, при этом иностранные сокращения предшествуют русским. Пример

оформления раздела «Обозначения и сокращения» приведен в приложении Г.

Основной текст расчетно-пояснительной записки состоит из введения, разделов основной части и заключения.

Во введении должна быть раскрыта сущность темы курсового проекта. Необходимо кратко охарактеризовать основные методы лексического и синтаксического анализа, использованные в процессе проектирования инструментальные и программные средства, основные литературные, информационные и другие источники. Рекомендуемый объем введения 1-2 страницы.

Основная часть текста включает в себя исследовательскую и (или) практическую части, а также экспериментальную часть (практическое приложение разработанного программного продукта, его тестирование).

Практическая или исследовательская часть представляет основное содержание записки и составляет примерно 60% её текста. Она предваряется детальным изложением решаемых задач. Далее следует подробное описание выполненной работы, структурированное по подразделам. Каждый подраздел должен иметь свое наименование и отражаться в содержании.

При описании необходимо указать используемые для решения задачи методы, алгоритмы и средства разработки. Следует показать, как развивался процесс разработки, какие решения были проанализированы, и обосновать решение, принятое для окончательного проектирования.

В подразделе «Лексические грамматики» следует привести алфавит входного языка и описание его лексики в форме Бэкуса-Наура. Нетерминальные символы лексических грамматик должны быть приведены на русском языке. Здесь же приводятся категории распознаваемых лексем (категории токенов) с обоснованием их категоризации.

В подразделе «Синтаксические грамматики» приводятся синтаксические грамматики в форме Бэкуса-Наура. Нетерминальные символы синтаксических грамматик должны быть приведены на русском языке. Нетерминальные символы должны описывать все распознаваемые конструкции входного языка.

Подраздел «Семантические соглашения и ограничения» описывает семантические соглашения и ограничения, накладываемые на входной язык.

Подраздел «Таблица токенов» описывает токены и их атрибуты, а также описание реализации таблицы токенов. Подраздел «Таблицы символов» описывает метод построения таблиц символов, их количество и назначение. При описании таблиц токенов и таблиц символов следует проанализировать возможные решения и обосновать выбор решения, принятого для конструирования транслятора.

В подразделе «Лексический анализ» следует привести все разработанные в процессе проектирования конечные автоматы, произвести (при необходимости) их минимизацию, привести соответствующие конечным автоматам грамматики.

Необходимо показать, каким образом выполняется предварительный лексический анализ, обосновать его алгоритм и способ формирования входного потока символов. При необходимости приводится блок-схема предварительного лексического разбора, а также блок-схемы реализации конечных автоматов.

Лексический анализ может быть выполнен с использованием детерминированных и недетерминированных конечных автоматов. В каждом конкретном случае следует показать, является конечный автомат детерминированным или нет, и обосновать выбор того или иного конечного автомата.

В подразделе «Синтаксический разбор» прежде всего необходимо обосновать выбор метода синтаксического разбора.

Основанием для этого служит описание синтаксиса языка, приведенное в подразделе «Синтаксические грамматики», а также анализ синтаксических грамматик, используемых для конструирования распознавателя. Следует заметить, что синтаксические грамматики, используемые для описания синтаксиса языка, и синтаксические грамматики, используемые для выполнения синтаксического разбора, могут в значительной мере различаться.

При описании синтаксического разбора следует отметить алгоритмы, средства и методы, использованные для проектирования синтаксического распознавателя, привести алгоритм распознавателя, подчеркнуть особенности реализации. Если синтаксические грамматики приводились к какому-либо стандартному виду, необходимо описать процесс приведения. Если для распознавания различных синтаксических конструкций были использованы различные методы анализа, нужно показать, как реализовано взаимодействие анализаторов.

При описании семантического анализа следует классифицировать семантические правила, используемые в проекте, указать место и способ их проверки.

Подраздел «Внутреннее представление программы» описывает выбранный метод промежуточного представления программы, получаемый на этапе анализа входного текста.

Подраздел «Генерация кода» описывает, как из внутреннего представления программы генерируется текст на выходном языке. При разработке интерпретатора этот подраздел может иметь название «Вычисление результата».

Заключение должно содержать выводы по выполнению задания на проект. Следует отметить преимущества, связанные с реализацией проектных предложений, практические рекомендации по совершенствованию объекта проектирования, охарактеризовать перспективы дальнейшего развития работы.

Список использованных источников и количество приложений формально не ограничены. Рекомендуемый объем основных используемых литературных источников от 5 до 20 наименований. Сведения об использованных источниках должны быть выполнены в соответствии с требованиями ГОСТ 7.1-2003.

Демонстрационная версия программного продукта предоставляется на дискете или компакт-диске, который прилагается к пояснительной записке. Длительность демонстрации программного продукта при защите курсового проекта составляет 5-7 минут.

Для защиты курсового проекта рекомендуется использовать компьютерную презентацию в виде последовательности слайдов. Демонстрация графического и иллюстративного материала в этом случае выполняется на мультимедийном проекторе.

2 Выполнение курсового проекта

2.1 Планирование времени

Разработка транслятора - очень сложная задача. Перед началом проектирования следует проанализировать, из каких этапов состоит проект, оценить сложность этапов и правильно распределить время, отводимое для выполнения курсового проекта, между отдельными этапами.

Можно выделить следующие основные отдельные составляющие части курсового проекта:

а) подготовка к проектированию;

б) лексический анализ;

в) синтаксический разбор и семантический анализ;

г) генерация кода;

д) оформление расчетно-пояснительной записки и подготовка презентации.

Учитывая, что на разработку курсового проекта отводится до 14 учебных недель, на выполнение одного этапа при равном распределении времени между этапами отводится около 20 календарных дней. Однако следует учитывать также и то, что выполнение этапов «а» и «б» достаточно хорошо описано в литературе и является относительно простым, в то время как выполнение этапов «в» и «г» является наиболее сложной частью проекта, которая описана в литературе недостаточно ясно и полно. При выполнении этапа «в» и особенно этапа «г» студенту придется столкнуться с проблемой разработки собственных, оригинальных методов решения задачи, поэтому с самого начала проектирования нужно выделить больше времени на вторую, более сложную часть проекта (этапы «в» и «г»).

Кроме указанных основных (больших) составляющих частей курсового проекта, при выполнении проекта приходится решать также много мелких задач в рамках отдельных этапов. Это составление спецификации входного языка, разработку среды программирования (среды разработки), проектирование таблиц токенов и таблиц символов, формирование входного потока символов, выбор и реализация способа промежуточного представления программы, разработка вспомогательных классов или функций для, например, организации стека, формирования синтаксического дерева и т.д.

Рекомендуемое распределение времени на выполнение курсового проекта приведено в таблице 2.1.

Таблица 2.1

Этап	Время на выполнение
Подготовка к проектированию	2 недели
Лексический анализ	2 недели
Синтаксический и семантический анализ	4 недели
Генерация кода	4 недель
Расчетно-пояснительная записка и презентация	2 недели

Следует понимать, что разработка курсового проекта по данной дисциплине является одной из самых сложных задач в ходе всего процесса обучения. В связи с этим начинать работу над проектом нужно не после выдачи задания, а до этого момента, в ходе выполнения практических занятий. Учебный процесс по данной дисциплине спроектирован и построен таким образом, чтобы обеспечить студента всей необходимой информацией для выполнения подготовительного этапа курсового проектирования.

Приведенное в таблице 2.1 распределение времени является ориентировочным и соответствует заданию на проектирование компилятора. В некоторых проектах может отсутствовать, например, этап генерации кода. В этих проектах большее время отводится на подготовительный этап и разработку лексического анализатора. Примером может служить проект, осуществляющий контроль текста HTML.

В любом случае распределение времени на выполнение этапов проектирования является предметом тщательного анализа, который выполняется студентом совместно с руководителем проекта во время выдачи задания на курсовое проектирование.

2.2 Спецификация входного языка

Работа над курсовым проектом начинается с определения и описания входного языка. На этом этапе студент совместно с руководителем проекта обсуждает упрощения и соглашения по лексике, синтаксису и семантике входного языка и разрабатывает его спецификацию, состоящую из описания алфавита, лексических и синтаксических грамматик, а также семантических соглашений и ограничений. Выполненная на этом этапе работа фиксируется в задании на проектирование и служит основанием для разработки составляющих частей проекта. В ходе выполнения проекта спецификация входного языка может уточняться.

Алфавит языка предназначен для записи текстов на входном языке. При составлении алфавита следует учитывать символы, которые могут встречаться в комментариях.

Лексические и синтаксические грамматики описываются в форме Бэкуса-Наура с записью нетерминальных символов на русском языке. Описание лексических грамматик начинается с описания категорий распознаваемых лексем, например:

```
<лексема>        ::=  <ключевое_слово>
                  |   <идентификатор>
                  |   <константа>
                  |   <операция>
                  |   <пунктуатор>
                  |   <комментарий>
```

Далее описываются категории лексем в том порядке, в котором они встречаются в описании категорий. Если описание лексемы содержит неопределенные нетерминальные символы, они описываются следом за описанием лексемы.

Рекомендуется для каждого символа привести его информативное описание.

Ниже приводится пример описания категорий лексем.

```
<ключевое_слово>::=  DIM | AS | INTEGER | LONG
                  |   IF | THEN | ELSE | END | PRINT

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

<буква>          ::=  A | B |  ...  | Z | a | b |  ...  | z

<цифра>          ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<константа>      ::=  <цифра>
                  |   <константа> <цифра>

<операция>       ::=  + | - | * | / | =

<пунктуатор>     ::=  ( | )

<комментарий>    ::=  ' <любой_символ> LF
```

Описание синтаксических грамматик начинается с нетерминального символа **<модуль>** или **<программа>**. Далее описываются нетерминальные символы, составляющие синтаксис программы.

Ниже в качестве примера приведено описание синтаксиса простого языка программирования типа BASIC.

```
<модуль>          ::=  <объявления> <операторы>
<объявления>      ::=  <объявление>
                    |  <объявления> <объявление>
<объявление>      ::=  DIM <идентификатор> AS <тип> LF | LF
<тип>             ::=  INTEGER | LONG
<операторы>       ::=  <оператор>
                    |  <операторы> <оператор>
<оператор>        ::=  LF
                    |  <присваивание>
                    |  <печать>
                    |  <условный>
<присваивание>    ::=  <идентификатор> = <выражение> | LF
<печать>          ::=  PRINT <идентификатор> | LF
<условный>        ::=  IF <выражение> THEN LF <операторы> END LF
                    |  IF <выражение> THEN LF <операторы>
                       ELSE <операторы> END LF
<выражение>       ::=  <терм>
                    |  <выражение> + <терм>
                    |  <выражение> - <терм>
<терм>            ::=  <элемент>
                    |  <терм> * <элемент>
                    |  <терм> / <элемент>
<элемент>         ::=  <идентификатор>
                    |  <константа>
                    |  ( <выражение> )
```

При построении грамматики для выражений следует учитывать приоритет операций, принятый во входном языке. Приоритеты операций для некоторых языков приведены в приложении Ж. Приоритет операций может оказать влияние на формируемые в результате лексического анализа токены. Так, если операции отношений (меньше, больше, меньше или равно, больше или равно) в некотором языке имеют одинаковый приоритет, во время лексического анализа все эти операции могут формировать один и тот же токен «операция отношения». В другом языке эти операции могут иметь разный приоритет, поэтому все они будут формировать разные токены.

Составленную грамматику следует проверить посредством построения дерева разбора нескольких примеров программ на

входном языке. Если дерево разбора построить не удается, грамматика составлена неверно. Лексические и синтаксические грамматики могут уточняться в ходе работы над соответствующими анализаторами.

Семантические соглашения и ограничения входного языка описываются произвольно в виде списка. Пример списка семантических ограничений приведен ниже:

1) Идентификатор переменной должен быть описан перед его первым появлением в коде программы;

2) Идентификатор переменной может быть описан один и только один раз в пределах области его видимости;

3) Программа должна содержать функцию «main»;

4) Количество и типы аргументов функции, указанные при ее вызове, должны совпадать с количеством и типами аргументов, указанных при ее описании (определении).

Не все семантические соглашения и ограничения очевидны в начале работы над транслятором. Некоторые из них выявятся в ходе разработки синтаксического анализатора и генератора кода. При этом следует учитывать также все преобразования, которые транслятор добавляет к программе на выходном языке, такие, как неявные преобразования типов. Следует, например, описать, как происходит преобразование типов при вычислении выражений.

2.3 Подготовка программных проектов

Следующим этапом проектирования является выбор средств разработки и подготовка к разработке программных компонентов (подготовка программных проектов).

Как правило, в качестве языка программирования при разработке транслятора используется язык Си (Си++), обладающий средствами низкоуровневого программирования. Это не исклю

чает возможность построения трансляторов на других языках, таких, как Pascal, BASIC, Ada, ассемблер и других. С точки зрения производительности транслятора использование технологий и парадигм программирования, таких, как ActiveX или ООП, может снизить характеристики конечного программного продукта. Однако в методических целях использование этих технологий и парадигм является обоснованным.

Рекомендуемая схема построения транслятора предполагает разработку оконной (многооконной) среды программирования на языке MS VB 6.0 и библиотеки объектов COM, реализуемой в MS VC++ 6.0. Библиотека объектов COM описывает компоненты транслятора, такие, как таблицы токенов, таблицы символов, лексический и синтаксический анализаторы, генератор кода, а также вспомогательные классы и объекты COM.

При данной схеме построения программного проекта сначала следует разработать оконное (многооконное) приложение, являющееся, по сути, редактором исходного кода. В учебных целях это приложение может содержать также окна для отображения результатов отдельных этапов анализа и синтеза, например, список токенов, таблицы идентификаторов, промежуточное представление программы а также результат трансляции.

Рекомендуемая среда разработки для разработки среды транслятора - MS VB. Проект MS VB предусматривает приложение с интерфейсом SDI или MDI. Название проекта задается заданием на курсовое проектирование и не может быть изменено.

В случае использования интерфейса типа SDI главное окно приложения содержит, кроме меню приложения, минимум два текстовых поля - одно для программного кода, другое - для вывода сообщений транслятора (рисунок 2.1).

В случае использования интерфейса MDI для представления кода программы, сообщений транслятора, результата трансляции и т.п. используются дочерние окна приложения.

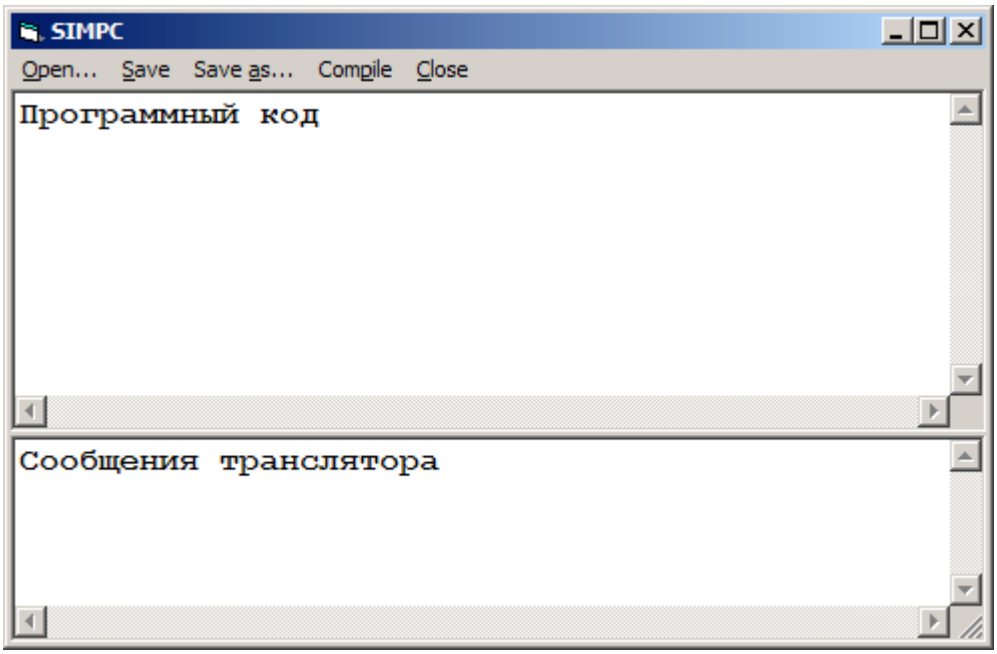


Рисунок 2.1 - Примерный внешний вид среды разработки

Для реализации лексического и синтаксического анализаторов подготавливается проект MS VC++ типа ActiveX DLL на основе библиотеки ATL. Отдельные компоненты транслятора в этом случае разрабатываются, как объекты COM.

Рекомендуемые названия классов COM следующие:

`Token` - класс, описывающий токен;

`Tokens` - класс, описывающий таблицу токенов;

`Symbol` - класс, описывающий символ программы;

`Symbols` - класс, описывающий таблицу символов;

`Lexan` - класс, описывающий лексический анализатор;

`Syntaxan` - класс, описывающий синтаксический анализатор;

Другие классы могут быть добавлены в проект при необходимости во время разработки, например, генерации кода.

Проект библиотеки (на MS VC++) нужно скомпилировать с получением библиотеки типов. Для проекта среды разработки (на MS VB) следует установить флажок «Create symbolic debug info» на вкладке «Compile» в свойствах проекта. Далее проект среды разработки связывается с библиотекой типов при помощи меню «Project-References...». Проект среды разработки должен быть скомпилирован с получением исполняемого файла. Далее в проекте библиотеки классов транслятора (на MS VC++) следует связать полученный .exe модуль среды разработки с проектом библиотеки (меню «Project-Settings», вкладка «Debug»).

2.4 Лексический анализатор

Разработка лексического анализатора производится на основании лексических грамматик. В ходе курсового проектирования необходимо построить алгоритм первичного анализа и разработать конечные автоматы для распознавания категорий лексем, определенных в разделе «Спецификация входного языка».

Прежде всего нужно определить способ формирования входного потока символов. Если размер файла исходного текста программы на входном языке не превышает размер 50-60 Кбайт, можно использовать буфер, в котором анализируемый текст размещается целиком. Использование одного буфера допускается также при построении учебных трансляторов и рассматривается как упрощение спецификации входного языка [3].

Если размер файла исходного текста может иметь произвольный размер, следует проанализировать, какие переходы по тексту относительного текущего символа возможны во время лексического анализа (заглянуть на символ вперед, на два символа вперед, поместить символ назад в поток), и в зависимости от этого выбрать схему буферизации входного потока. В этом слу

чае также понадобится, вероятно, разработать функции для заполнения буферов входного потока, получения текущего символа, для предпросмотра следующего символа, для перемещения указателя текущего символа и т.п.

Алгоритм первичного анализа лексем основывается на предположении, что разные категории лексем начинаются с разных символов. Например, идентификаторы начинаются с буквы, константы - с цифры, строковые литералы - со знака двойной кавычки, комментарии - со знака одинарной кавычки и т.д. В соответствии с этим предположением алгоритм первичного разбора представляет собой детерминированный конечный автомат, состояниями которого являются конечные автоматы для разбора лексем определенных категорий.

При построении алгоритма первичного разбора следует учитывать, какие символы являются значащими для разбора, а какие - нет (являются пробельными). Кроме того, важно определить, является ли символ конца строки распознаваемым токеном.

Важным моментом построения алгоритма первичного разбора является также выбор схемы взаимодействия лексического и синтаксического анализаторов. При последовательной схеме лексический анализатор является самостоятельной и независимой частью транслятора, которая в результате анализа входного потока символов строит выходной поток токенов в виде, например, таблицы токенов. При параллельной схеме взаимодействия лексический анализатор распознает лексемы одна за другой, по мере того, как синтаксический анализатор запрашивает их [3].

При последовательной схеме для формирования потока токенов в классе лексического анализатора создается метод `Compile`, параметрами которого могут являться входной текст и таблица токенов. При параллельной схеме в классе лексического анализатора создается метод `GetToken`, который возвращает очередной

токен и его атрибуты. Организация класса лексического анализатора при этом получается различной.

При последовательной схеме входной текст подается на вход метода `Compile` (при этом текст при необходимости преобразовывается из кодировки Unicode в кодировку ANSI) и сканируется от начала до конца в цикле первичного анализа.

При параллельной схеме входной текст задается, например, как свойство класса `Text` (при этом преобразование текста из кодировки Unicode в кодировку ANSI осуществляется в функции, устанавливающей новое значение свойства `Text`). Заметим, что вместо функции свойства можно использовать метод под названием, например, `SetText`. Метод `GetToken` выполняет распознавание очередной лексемы так же, как одна итерация цикла первичного анализа при последовательной схеме.

Таким образом, общим для последовательной и параллельной схем организации лексического анализа является алгоритм определения категории лексемы.

Определяя схему взаимодействия лексического и синтаксического анализаторов, следует учитывать, сколько проходов будет выполнять транслятор. Однопроходные трансляторы всегда строятся по параллельной схеме.

Распознавание категорий лексем выполняется специально сконструированными конечными автоматами. Конечный автомат может быть разработан на основе лексической грамматики, приведенной к автоматному виду, на основе анализа входного потока (интуитивно), и на основе регулярного выражения. В первом случае нужно разработать грамматику языка, описывающего лексемы данной категории. Например, для распознавания идентификаторов, описание которых приведено в разделе «Спецификация входного языка», можно использовать следующую грамматику:

```
G({a, d}, {A}, {
A ::= a | Aa | Ad
}, A)
```

На основании этой грамматики по известным алгоритмам может быть построен детерминированный конечный автомат, приведенный на рисунке 2.2.

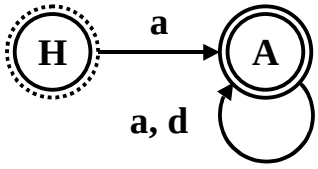


Рисунок 2.2

Для практической реализации этот автомат следует дополнить завершающим «псевдосостоянием» S, в которое автомат переходит по любому недопустимому символу «c». Учитывая ограничение на распознаваемую длину идентификатора, данный автомат должен быть приведен к недетерминированному виду. В результате для построения реального распознавателя используется конечный автомат, приведенный на рисунке 2.3 [3].

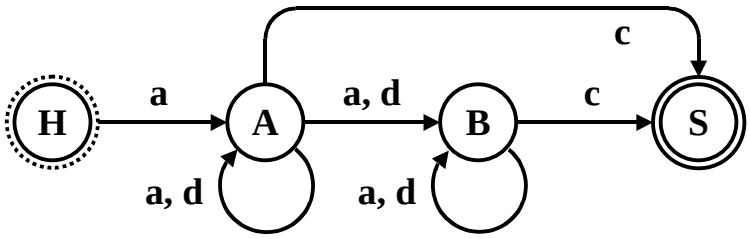


Рисунок 2.3

Разработка грамматики для распознавания языка, описывающего вещественные константы, может оказаться достаточно сложным. В этом случае можно интуитивно построить конечный автомат, основываясь на порядке следования символов в различных предложениях языка. После разработки конечного автомата следует построить на его основе образующую автомат граммати

ку, используя известные алгоритмы преобразования. При необходимости для полученного интуитивным образом конечного автомата должна быть выполнена его минимизация.

При использовании регулярных выражений для построения конечного автомата следует использовать один из известных алгоритмов.

Каждый отдельный конечный автомат реализуется в виде функции класса анализатора. Следует обратить внимание на название функций автоматов. Название функции должно отражать вид распознавания. С этой точки зрения одни конечные автоматы не фиксируют ошибок распознавания, а другие фиксируют. Например, автоматы, распознающие идентификаторы, не фиксируют при распознавании никаких ошибок, а автоматы, распознающие комментарии, строковые литералы, вещественные константы и операции могут фиксировать ошибки.

Автоматы, не фиксирующие ошибок при распознавании, следует называть с использованием префикса «`get`», а автоматы, фиксирующие ошибки, следует называть с использованием префикса «`is`», например, `get_ident`, `is_operator`. Соответственно, автоматы, фиксирующие ошибки, должны возвращать признак успешного (или неуспешного) распознавания данной категории лексем для того, чтобы алгоритм первичного анализа мог принимать решения об откате или об остановке трансляции.

При разработке лексического анализатора следует также определить, какие сообщения об ошибках может формировать лексический анализатор, и действия лексического анализатора в случае обнаружения той или иной ошибки. Каждую ошибку следует описать с указанием ее номера и сообщения.

Важную составляющую часть работы над лексическим анализатором составляет разработка таблицы токенов и таблицы символов программы.

При разработке таблицы токенов следует определить необходимые атрибуты токенов, такие, как позиция в исходном тексте, строковое представление и другие. Далее для представления токена создается тип (структура или класс), и на основе этого типа создается тип, описывающий таблицу.

Таблица символов является важной составляющей частью транслятора. К ней предъявляются противоречивые требования, связанные с частыми обращениями к ней во время всех этапов работы транслятора. В связи с этим разработка таблицы символов сама по себе может служить предметом курсового проектирования. В целях упрощения допустимо строить таблицу символов на основе неэффективных методов, таких, как массивы постоянного размера, инкрементные массивы, связанные списки. Для построения таблицы символов используется тип (класс), описывающий символ и его атрибуты.

На атрибуты символа следует обратить особое внимание. Для символов, описывающих скалярные переменные, атрибуты символа могут включать в себя тип символа, оригинальное имя символа, декорированное (mangled) имя символа, адрес символа в области памяти данных результирующей программы. Для символа, описывающего функцию (процедуру), дополнительно к указанным атрибутам следует добавить описание аргументов. Каждый аргумент, по сути, также является символом, к которому применимы перечисленные выше атрибуты.

Следует также решить, сколько таблиц символов будет иметь разрабатываемый транслятор. Одним из вариантов является использование двух таблиц - одна для символов, описывающих переменные, другая - для символов, описывающих функции. В связи с ограничениями и упрощениями, накладываемыми на входной язык транслятора (таких, например, как распознавание

текста одного модуля), во многих случаях создание одной или двух таблиц является оптимальным решением.

Для формирования (заполнения) таблиц токенов и таблиц символов разрабатываются соответствующие функции (или методы классов). Разрабатываются также функции (методы) для извлечения информации о токенах и символах. Заметим, что формирование и заполнение таблиц символов может быть отнесено как к этапу разработки лексического анализатора, так и к этапу разработки синтаксического анализатора.

Итогом работы лексического анализатора является поток токенов. Главной характеристикой (главным атрибутом) токена является его числовой идентификатор. Этот идентификатор является обозначением терминального символа для последующего синтаксического анализа. Поэтому следует продумать, какие числовые значения будут принимать конкретные токены с тем, чтобы обеспечить непересечение обозначений токенов с обозначениями нетерминальных символов, используемых для синтаксического анализа. Это возможно только если известны синтаксические грамматики, используемые для синтаксического анализа. Их разработка является предметом следующего этапа проектирования транслятора. Тем не менее, на этапе лексического анализа можно проанализировать предполагаемое количество нетерминальных символов синтаксических грамматик и сделать предположение об общем количестве терминальных и нетерминальных символов с тем, чтобы определить тип данных, описывающий числовой идентификатор символа. В реальных компиляторах число токенов составляет величину порядка 1000, в учебных компиляторах, в зависимости от типа разрабатываемого транслятора, эта величина может иметь порядок 100.

2.5 Синтаксический анализатор

Разработка синтаксического анализатора является основной и наиболее сложной задачей на этапе анализа текста на входном языке. Прежде всего нужно проанализировать синтаксические грамматики спецификации входного языка и разработать синтаксические грамматики, используемые для построения синтаксического анализатора (или синтаксических анализаторов). Во многих практических случаях при этом получаются разные грамматики.

Необходимость преобразования исходных синтаксических грамматик обуславливается использованием для распознавания тех или иных методов синтаксического разбора. Так, если для распознавания синтаксических конструкций используется нисходящий метод на основе LL(1) грамматик, то грамматики должны удовлетворять правилу «все правила разбора одного нетерминального символа должны вырождаться в цепочки, начинающиеся с разных терминальных символов». Если же для анализа используется метод рекурсивного спуска, то на грамматики накладывается еще более сильное ограничение - «все правила грамматики для одного нетерминального символа должны начинаться с разных нетерминальных символов». Кроме того, при нисходящем разборе потребуется устранение левой рекурсии, которая недопустима в большинстве методов нисходящего разбора. Для многих классов распознавателей требуется выполнить и другие преобразования грамматик, такие, как устранение цепных правил, пустых правил (правил вида `A ::= λ`) и другие.

Очень трудно дать рекомендации по выбору того или иного конкретного метода распознавания. Можно попробовать сначала преобразовать правила исходной грамматики так, чтобы в правой части правил осталось как можно меньше символов. Например,

известно, что любая КС грамматика может быть приведена к нормальной форме Хомского [4], в которой в правой части правил могут встречаться только терминальные символы или пары нетерминальных символов. Далее можно попытаться доказать, что полученная грамматика удовлетворяет требованиям, предъявляемым для LL(1) грамматик. Для этого необходимо построить множества FIRST для всех нетерминальных символов, а если в грамматике присутствуют пустые правила, то дополнительно построить множества FOLLOW и сравнить полученные множества определенным образом [4].

Одним из эффективных методов синтаксического разбора является расширенный метод рекурсивного спуска [4]. Правила грамматики для применения этого метода должны удовлетворять примерно тем же требованиям, которые применяются к методу рекурсивного спуска. Если эти требования не выполняются, можно попытаться несколько раз выполнить левую факторизацию. Во многих случаях расширенный метод рекурсивного спуска дополнительно можно модернизировать так, чтобы получить возможность однозначно разбирать синтаксические конструкции, записанные в начальном виде. Следует заметить, что расширенный метод рекурсивного спуска является одним из самых эффективных.

Одним из методов, который позволяет реализовать большое количество реальных языков, является метод, основанный на использовании грамматик простого предшествования [5]. Для проверки того, может ли быть использован этот метод в отношении конкретной грамматики, нужно построить матрицу отношений предшествования для данной грамматики. Если при этом не возникает конфликтов (для каждой пары символов матрицы существует одно и только одно отношение предшествования), то грамматика является грамматикой простого предшествования.

Сложнее всего определить принадлежность грамматики к классу LR. Здесь прежде всего нужно построить множество LR(0) пунктов и попытаться составить таблицу синтаксического анализа. Если таблица не имеет конфликтов «перенос-свертка», то грамматика принадлежит классу SLR(1), в противном случае строится множество LR(1) пунктов и снова составляется таблица синтаксического анализа. Если вновь построенная таблица не содержит конфликтов, то грамматика принадлежит к классу LR и далее ее всегда можно привести к виду LALR(1).

Для построения множеств LR пунктов лучше воспользоваться какой-нибудь вспомогательной программой, так как построение этих множеств для реальных грамматик может оказаться слишком сложной задачей (количество LR пунктов даже для очень простых языков может составлять сотни единиц) [6].

Во многих случаях более простым решением может оказаться построение нескольких распознавателей для разбора разных синтаксических конструкций языка. Например, синтаксические конструкции операторов языка разбираются методом расширенного рекурсивного спуска, а арифметические и логические выражения - методом восходящего разбора на основе грамматики класса SLR(1). При этом во время разработки реализации синтаксического анализатора придется решить задачу согласования работы разных распознавателей, - способ их вызова, передачи параметров, результатов распознавания, фиксации ошибок. Использовать несколько разных распознавателей имеет смысл при построении интерпретатора, который разбирает текст по правилу «оператор за оператором». При этом распознаватель выбирается в зависимости от типа оператора.

При разработке синтаксических грамматик могут встретиться стандартные (известные) проблемы. В их число входят, например, проблема «висячего else» и проблема «унарного мину

са». На эти стандартные ситуации следует обратить особое внимание, следует найти в источниках способы их решения и применить эти способы в своей грамматике.

Необходимо тщательно проверить грамматику для вычисления выражений. Часто допускаются ошибки, связанные с приоритетами. Все операции с одним приоритетом должны быть описаны одним нетерминальным символом, однако следует учитывать также ассоциативность операций. Операции, имеющие меньший приоритет, расположены ближе к нетерминальному символу, описывающему выражение, а операции, имеющие больший приоритет, расположены ближе к нетерминальному символу, описывающему элементы данных.

Не менее часто при проектировании неверно выявляются элементы данных, из которых выражение может состоять. Например, забывают, что элементом данных может являться функциональный вызов, элемент массива, элемент класса и т.д. Распознавание функционального вызова является сложной задачей как в плане построения грамматики, так и в плане определения выполнения семантических соглашений.

После того, как грамматика будет приведена к виду, удовлетворяющему одному из классов КС грамматик, можно приступить к построению распознавателю этого класса.

Здесь прежде всего нужно окончательно определить числовые идентификаторы, определяющие терминальные и нетерминальные символы. Напомним, что терминальными символами на этапе синтаксического разбора являются токены, получившиеся в результате лексического анализа. Следует сформировать такие числовые значения терминальных и нетерминальных символов, чтобы они составляли две неперекрывающиеся области. Например, первая часть пространства числовых идентификаторов образована терминальными символами, а вторая - нетерминальными.

При построении распознавателя это позволит легко определять, является ли символ терминальным или нетерминальным.

Основу большинства распознавателей составляют управляющие таблицы. Эти таблицы для реальных распознавателей могут иметь значительный объем. Например, если общее количество терминальных и нетерминальных символов составляет всего одну сотню, размер таблицы для синтаксического анализа на основе грамматики простого предшествования составит объем до 10000 элементов. Из практики проектирования учебных компиляторов известно, что таблица синтаксического анализа на основе грамматики класса SLR для сравнительно простого языка имеет сопоставимый объем. Даже описание этих таблиц в коде распознавателя может вызвать значительные затруднения, поэтому с целью исключения ошибок опять же лучше использовать какие-нибудь вспомогательные средства.

Если для построения распознавателя используются классы, а процесс распознавания описывается внутри метода, то управляющую таблицу следует размещать не внутри класса, а вне его, например, в специально выделенном модуле. Практика показывает, что несоблюдение этого условия приводит к неоправданно длительному процессу компиляции проекта транслятора (вплоть до часов) и к нестабильной работе конечного продукта.

Построение синтаксического распознавателя выполняется на основе алгоритма его работы, который во многих случаях достаточно хорошо описан. Однако работа большинства распознавателей описывается при помощи обычного или расширенного МП автомата, использующего магазинную память (память со стековой организацией). Поэтому в ходе курсового проектирования потребуется разработать класс стека с возможностью оперирования не только символом на вершине, но и множеством символов.

Возможно, потребуется разработка не одного, а двух или более классов стека для организации стеков разного назначения.

Если проект построен с использованием классов, то для реализации распознавателя создается метод `Compile`. Параметром метода является входная таблица токенов, если используется последовательная схема взаимодействия лексического и синтаксического анализаторов. При параллельной схеме функция синтаксического распознавателя вызывает метод `GetToken` лексического распознавателя для получения от него очередного токена.

В ходе работы над синтаксическим распознавателем необходимо продумать, какие ошибки будут фиксироваться во время синтаксического разбора, и как синтаксический анализатор будет на них реагировать. Каждая ошибка синтаксического разбора должна быть описана с указанием ее номера и сообщения.

На начальном этапе проектирования синтаксического анализатора трудно сказать, что будет являться результатом его работы. Поэтому рекомендуется сначала добиться правильного распознавания синтаксических конструкций, записи примененных правил и фиксации ошибок. После того, как эта часть работы будет выполнена, можно будет перейти к этапу генерации кода, в ходе которого будет определен способ формирования промежуточного представления программы, необходимые для этого структуры (классы), а синтаксические грамматики будут дополнены семантическими правилами.

2.6 Этап генерации кода

На этапе генерации кода определяется способ внутреннего представления программы, синтаксические грамматики дополняются операционными символами и (или) атрибутами, описывающими семантические правила, которые необходимы для про

верки и выполнения семантических ограничений и соглашений, и генерации промежуточного представления программы. Далее разрабатывается генератор кода, который формирует текст на выходном языке, используя полученное в ходе синтаксического разбора внутреннее представление программы.

Для внутреннего представления программы наиболее часто используются следующие способы:

- синтаксические деревья;
- тетрады и (или) триады;
- ПОЛИЗ (польская инверсная запись);
- промежуточный код, например, байт-код JAVA.

Синтаксические деревья описывают программу в виде древовидной структуры. Синтаксическое дерево отличается от дерева вывода отсутствием в нем правил вида $A \rightarrow B$. Синтаксическое дерево может быть построено непосредственно во время синтаксического разбора, в ходе которого определяются номера применяемых правил грамматики. Для построения синтаксического дерева синтаксическая грамматика дополняется множеством операционных символов, записываемых на выходную ленту распознающего автомата. После успешного выполнения синтаксического разбора выходная лента интерпретируется в синтаксическое дерево. Далее синтаксическое дерево может быть использовано для, например, генерации кода на выходном языке, при условии, что нет необходимости выполнять семантические преобразования (в трансляторах и корректорах). Кроме того, синтаксическое дерево может быть использовано для получения обратной польской записи (ПОЛИЗ).

Тетрады и триады используют записи из четырех (трехадресный код) или трех (двухадресный код) полей. Первое поле обозначает выполняемое действие, следующие поля - операнды и результат. В тетрадах два поля обозначают операнды, и одно по

ле - результат. В триадах результат обозначается как один из операндов (записывается вместо операнда), и в качестве операндов могут выступать ссылки на предыдущие триады. Операнды в тетрадах и триадах - это переменные программы (адреса переменных), а также метки, используемые для переходов при организации условных конструкций и циклов. Тетрады и триады наиболее удобны для дальнейшей генерации объектного кода, или кода на языке ассемблера (макроассемблера).

Обратная польская запись ПОЛИЗ наиболее удобна для записи вычисления выражений, хотя ее можно относительно легко модернизировать для записи условных конструкций и циклов. С ее помощью также можно относительно просто построить процесс вычисления выражений с использованием вычислительного стека сопроцессора Intel.

Формирование промежуточного кода соответствует компилирующе-интерпретирующей схеме построения программы. При этом создается последовательность инструкций некоторой абстрактной машины, которая интерпретируется при помощи дополнительного программного обеспечения (компилятора или интерпретатора времени исполнения). В общем можно сказать, что получение промежуточного кода в значительной степени схоже с генерацией выходного текста на языке ассемблера. Различие заключается в том, что язык ассемблера отражает особенности архитектуры конкретной вычислительной среды, в то время как промежуточный код является платформенно-независимым.

Основную сложность на этапе генерации кода составляет проверка семантических правил. К ним относятся, прежде всего, выполнение семантических правил, связанных с порядком следования синтаксических конструкций в исходном тексте (например, объявление переменной должно предшествовать ее использова

нию), преобразованием типов при вычислении выражений, определении области видимости переменных и т.п.

Здесь следует использовать атрибутные грамматики, определяющие семантические правила трансляции и, соответственно, синтаксически-управляемый перевод (синтаксически-управляемую трансляцию). С помощью атрибутов синтаксической грамматики можно передавать информацию, связанную с конкретными токенами и нетерминалами, от одних правил вывода к другим, при этом могут передаваться любые сведения, необходимые разработчику - типы лексем, значения лексем, текущая функция (процедура), вызывающая функция (процедура), текущая синтаксическая конструкция, результат вычисления выражения, результат вычисления оператора управления и т.п.

Генерация кода на макроассемблере

Ассемблер описывает некоторую «целевую машину». Под целевой машиной упрощенно можно понимать процессор, который в конечном итоге, будет выполнять скомпилированную программу, а также программную модель памяти и взаимодействия с внешними устройствами. В ОТИ МИФИ в качестве целевых машин в настоящее время используются процессоры Intel семейств x86 и 8051. Первые используются для разработки учебных компиляторов, вторые - для создания учебных кросс-компиляторов.

Прежде, чем приступить к непосредственно генерации кода, необходимо достаточно хорошо представлять, как строится программа на ассемблере. Если говорить об ассемблере для процессора Intel x86, то с его помощью можно построить две модели программ: сегментированную и плоскую. Первая модель основана на сегментированной модели памяти и применяется для приложений, работающих под управлением операционной системы

типа MS-DOS. При этом процессор работает в режиме совместимости с процессором Intel 80x86, в распоряжении программы находится до 600 Кбайт оперативной памяти, и все процессы, связанные с управлением памятью, программа должна решать сама. При программировании на ассемблере в этом режиме требуется самостоятельно управлять сегментными регистрами. Кроме того, при использовании сегментированной модели программа должна также брать на себя управление векторами прерывания, буферизацию ввода-вывода и т.п. По завершении работы программа должна освободить память и восстановить вектора прерывания.

Плоская модель памяти используется для создания приложений, работающих в защищенном режиме процессора, как правило под управлением операционной системы типа Windows. В этом режиме программе предоставляется до 4 Гбайт виртуальной памяти, имеющей плоскую структуру. Программа представляет должна создавать первичный поток при помощи функций операционной системы, а завершение работы программы осуществляется функцией завершения процесса. Отличительной особенностью программы с плоской моделью является возможность использования функций операционной системы для, например, создания окон, обработки сообщений и т.д.

Независимо от выбранной модели, весь код программы на ассемблере можно разбить на три составляющие части.

Первая часть, называемая здесь прологом, описывает модель памяти, используемый процессор, способ формирования кадра стека (дисплея процедуры), начальный стек, предопределенные константы и переменные, формирование первичного потока или описание метки начала программы. В эпилоге могут быть также описаны макросы, непосредственно отвечающие за генерацию вычислений.

Средняя часть образует собственно код программы, состоящий из отрезков, описывающих содержимое сегмента кода и сегментов данных в произвольном порядке.

Третья часть, называемая здесь эпилогом, описывает заключительную часть программы, в частности, метку начала программы и признак конца файла программы. Эпилог может также содержать описание сегмента данных с предопределенными константными объектами, например, сообщениями об ошибках.

В большинстве случаев пролог и эпилог программы могут быть написаны так, чтобы подходить для всех генерируемых компилятором программ. Это обстоятельство позволяет заранее написать пролог и эпилог программы в требуемом формате, и формировать конечный текст на выходном языке простым включением пролога и эпилога.

В этом случае задача генерации кода заключается в том, чтобы сформировать необходимую последовательность инструкций средней части программы, которая образует процессы вычисления выражений, с одной стороны, и описание структур данных с другой.

Принцип генерации выходной программы на ассемблере для целевой машины 8051 не отличается от описанного. Для этой машины другими будут организация и распределение памяти, элементы данных, машинные инструкции и т.п. Однако программу так же можно поделить на пролог, среднюю часть и эпилог.

Чтобы пояснить сказанное, разберем конкретный пример. Пусть на языке, описанном в 2.2, задана следующая программа:

```
Dim A As Integer
Dim B As Integer
A = 2
B = 3 * A
Print B
```

Компиляция этой программы на ассемблер x86 с сегментированной моделью должно привести, например, к следующему выходному тексту:

```
P286                    ; процессор
MODEL SMALL,PASCAL      ; модель памяти и формат вызова
STACK 1024              ; размер стека программы
INCLUDE MACROSS.INC     ; фукнции и макросы языка
CODESEG                 ; сегмент кода
_START:                 ; метка входа в программу
MOV AX, @DATA           ; инициализация
MOV DS, AX              ; сегмента данных
;-----------------------------------------------------;
DATASEG                 ; сегмент данных
A DW ?                  ; Dim A As Integer
B DW ?                  ; Dim B As Integer
CODESEG                 ; сегмент кода
MOV AX, 2               ; выражение = 2
I2ASS A, AX             ; A = выражение
I2MUL 3, A              ; выражение = 3 * A
I2ASS B, AX             ; B = выражение
MOV AX, B               ; параметр процедуры Print
PUSH AX                 ; помещается в стек
CALL PRINT              ; Print B
;-----------------------------------------------------;
ExitCode 0              ; завершение работы
END _START              ; конец текста и метка входа
```

Три части программы выделены при помощи двух горизонтальных линий. Текст компилируется при помощи макроассемблера Turbo Assembler 3.2.

Пролог определяет параметры программы и внутренние функции проектируемого транслятора. Первая строка программы задает процессор 286 и, соответственно, сегментированную модель памяти. Вторая строка задает модель памяти программы **SMALL** (размером до 64 Кбайт) и формат кадра стека при вызове процедур такой же, как в языке Pascal. Третья строка задает размер стека.

В следующей строке в файл программы включается заранее подготовленный файл, в котором определены функции, реализуемые непосредственно языком программирования, и макросы,

описывающие вычислительные операции. Далее определяется сегмент кода при помощи директивы **CODESEG**, метка входа в программу **_START** и описывается инициализация сегмента данных (две инструкции).

Эпилог программы содержит макровызов **ExitCode**, при помощи которого программа завершает работу, и заключительную директиву **END** с указанием метки входа.

Средняя часть получена в результате синтаксического разбора текста. Заметим, что директивы, переключающие сегменты кода и данных, могут следовать в программе произвольным образом. Поэтому, как только распознана первая строка исходного текста, в выходной текст включается переход к сегменту данных **DATADEG** и определяется первая переменная (выделяется память заданного типом **Integer** размера). Далее аналогично выделяется память для второй переменной.

Оставшаяся часть кода выполняет собственно вычисления, поэтому сегмент данных переключается на сегмент кода при помощи директивы **CODESEG**. Далее в исходной программе расположен оператор присваивания, правая часть которого по синтаксису входного языка является выражением. Распознав, что выражение состоит из константы 2, синтаксический анализатор помещает это значение в рабочий регистр **AX**. Поскольку вычисление выражения закончено, генерируется триада **I2ASS A, AX**, которая вызывает макрос, выполняющий присваивание второго аргумента первому.

Следующая инструкция входной программы тоже присваивание, однако выражение в правой части представляет собой бинарную операцию умножения. Распознав операнды, синтаксический анализатор генерирует триаду умножения **I2MUL 3, A**, и результат умножения снова попадает в рабочий регистр **AX**. Далее

вызывается макрос присваивания, и значение рабочего регистра передается переменной **B**.

В заключение производится вызов процедуры **Print**, которая является внутренней процедурой входного языка (транслятора). Значение параметра процедуры помещается в стек и вызывается метка **PRINT**.

Большое значение приобретает включаемый в выходной текст файл с макросами **MACROSS.INC**. Этот файл является частью разработки генератора кода. Он описывает, кроме встроенных функций языка, таких, как **Print**, триады (или тетрады), при помощи которых и формируется процесс вычислений. Разработка этих триад составляет существенную часть разработки генератора кода и в значительной степени влияет на формат результирующего кода и, в конечном итоге, его качество.

В качестве примера приведем используемые в данной программе триады-макросы. Первая триада-макрос описывает целочисленное присваивание 16-разрядных значений:

```
I2ASS MACRO A, B
    MOV A, B
ENDM
```

Следующая триада-макрос описывает целочисленное умножение 16-разрядных значений:

```
I2MUL MACRO A, B
    MOV AX, A
    MOV BX, B
    MUL BX
ENDM
```

Заметим, что результат вычислений всегда записывается в операционный (рабочий) регистр **AX**. Это соглашение, которое используется для организации вычислений. Другим вариантом является использование специальной области памяти - рабочей переменной.

Для организации вычислений в полном объеме, предусмотренном заданием на курсовое проектирование, потребует разработки множества макросов для выполнения множества операций. В зависимости от количества типов, определенных спецификацией входного языка, количество макросов для выполнения одной и той же операции может быть различным. Кроме того, могут понадобиться специальные триады-макросы, описывающие приведение одних типов к другим.

Важным является вопрос о реализации функций. Для передачи параметров используется так называемый кадр стека (дисплей процедуры). Он представляет собой область стека, в которую перед вызовом процедуры записываются ее фактические параметры и формируется область локальных переменных. Чтобы сформировать кадр стека, после входа в процедуру нужно зафиксировать указатель **BP** на середину кадра стека, на которую при входе указывает указатель стека **SP**. При этом выше указателя (с положительным смещением) в стеке располагаются аргументы функции, и ниже указателя (с отрицательным смещением) в стеке находятся локальные переменные. Рассмотрим пример оформления функции **Print**, имеющей один аргумент с именем **A** и одну локальную переменную с именем **N**. Функция должна быть записана во включаем файле **MACROSS.INC**.

```
PRINT PROC PASCAL NEAR
ARG @@A:WORD
LOCAL @@N:WORD
    ; какие-то действия
RET
ENDP
```

Директива **PASCAL** указывает, что аргументы в стек передаются в прямом порядке, и стек очищается инструкцией **RET**. Заметим, что директива была также указана в прологе программы, поэтому в этом месте ее можно было опустить.

Директива **ARG** описывает аргументы процедуры. Именем аргумента функции является **@@A**. Директива **LOCAL** описывает локальные переменные функции. В данном случае используется одна переменная с именем **@@N**.

3 Оформление курсового проекта

3.1 Расчетно-пояснительная записка

Расчетно-пояснительная записка выполняется с помощью компьютера и распечатывается на лазерном принтере. Оформление текста должно соответствовать стандарту ОТИ МИФИ [1].

Текст располагается на одной стороне белого листа формата А4 (210×297 мм). На каждом листе должна быть выполнена рамка по ГОСТ 2.104 (размер рамки 185×287 мм, толщина линии рамки 0,8-1,2 мм). Для обеспечения пропечатывания рамки на лазерных принтерах разных производителей допускается уменьшить высоту и ширину рамки на максимум 3 мм (до размера 182×284 мм). Рамка имеет отступ от левого края листа 20 мм, от верхнего края листа - 5 мм. Размер и толщина линии рамки на всех листах должны быть одинаковыми.

Лист аннотации на русском языке содержит основную надпись по форме 2 (пример см. приложение Б), остальные листы, за исключением титульного, содержат основную надпись по сокращенной (неполной) форме 2а (пример см. приложение В).

Текст на странице со всех сторон должен иметь отступ от рамки и от основной надписи, равный 4-5 мм.

Первой страницей является титульный лист. Номер страницы на титульном листе не ставится, на других листах номер страницы проставляется в основной надписи. Страницы 2-4 составляют задание на курсовое проектирование. За заданием располагается аннотация на одном листе.

После аннотации размещается содержание, включающее в себя заголовки разделов и подразделов.

Текст выполняется шрифтом Times New Roman, размер шрифта 14 пунктов, междустрочный интервал - полуторный. Вы

равнивание основного текста - «по ширине». Абзацы основного текста выполняются с отступом первой строки, равным 1,5 см (с «красной» строкой).

Не допускаются выделения текста при помощи полужирного и (или) курсивного начертания букв, а также подчеркивания.

Переносы слов в документе должны быть разрешены.

Заголовки разделов, подразделов, пунктов и подпунктов также выполняются с «красной» строкой и выравниваются по левому краю. Заголовки не должны содержать переносов слов. В конце заголовка не допускаются никакие знаки препинания.

Заголовки разделов и подразделов, пунктов и подпунктов должны быть по возможности краткими и раскрывать содержание изложенного материала. Разделы и подразделы должны иметь порядковые номера, обозначенные арабскими цифрами. После каждого номера должна стоять точка, за исключением последнего номера, после которого точка не ставится.

Пункты и подпункты также могут иметь нумерацию арабскими цифрами. Номер пункта состоит из номера раздела, номера подраздела и номера пункта, разделенных точками. Номер подпункта состоит из номера раздела, номера подраздела, номера пункта и номера подпункта, разделенных точками.

Разделы «Обозначения и сокращения», «Введение» и «Заключение» не нумеруются.

Каждый раздел следует начинать с новой страницы.

Расстояние между заголовком раздела, подраздела, пункта, подпункта и основным текстом визуально должно составлять одну пустую строку.

Перечисления оформляются так же, как основной текст.

Пример оформления перечислений:

1) пример элемента перечисления первого уровня;

а) пример элемента перечисления второго уровня, который располагается на двух строчках;

- пример элемента перечисления третьего уровня, который располагается на двух строчках;

- пример элемента перечисления третьего уровня;

б) пример элемента перечисления второго уровня, который располагается на двух строчках;

2) пример элемента перечисления первого уровня, который располагается на двух строчках.

Документ может содержать таблицы, рисунки и формулы. На каждую таблицу, рисунок или формулу в документе непосредственно перед таблицей, рисунком или формулой должна быть ссылка, например, «см. рисунок 2.1», «(таблица 3.1)», «по формуле 3.2».

Рисунок размещается по центру страницы. Под рисунком размещается подпись, состоящая из слова «Рисунок» и номера, состоящего из номер раздела и порядкового номера рисунка в разделе. Если рисунок имеет название, оно размещается за номером рисунка через тире. Пример оформления рисунка см рисунок 3.1.

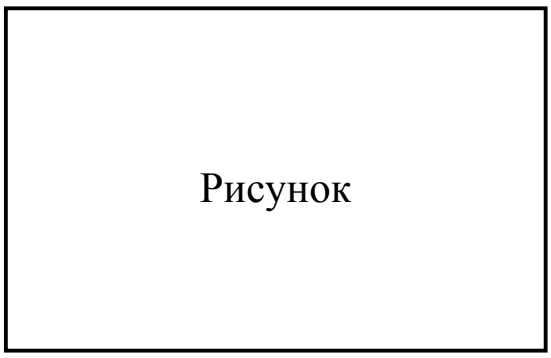


Рисунок 3.1 - Пример оформления рисунка

Знаки препинания в конце названия рисунка недопустимы. Рисунок и подпись рисунка должны располагаться на одной

странице. Расстояние между текстом и рисунком, между рисунком и подписью рисунка, а также между подписью рисунка и последующим текстом визуально должно составлять одну пустую строку.

Таблица выравнивается по левому краю. Перед таблицей должна находиться подпись, состоящая из слова «Таблица» и номера таблицы, состоящего из номера раздела и порядкового номера таблицы в разделе. Если таблица имеет название, оно размещается за номером рисунка через тире.

Знаки препинания в конце названия таблицы недопустимы. Расстояние между текстом и подписью таблицы, а также между таблицей и последующим текстом визуально должно составлять одну пустую строку. Расстояние между подписью таблицы и таблицей визуально должно составлять половину пустой строки. Подпись таблицы и начало таблицы должны располагаться на одной странице.

Если таблица не умещается на одном листе, она разрывается на две или более частей. Перед таблицей, которая является продолжением таблицы, начало которой находится на предыдущей странице, размещается подпись «Продолжение таблицы» или «Окончание таблицы» с указанием номера.

Рамки таблицы выполняются толщиной 1 пункт. Между рамкой таблицы и текстом внутри таблицы должно быть расстояние не менее 2 мм. Числовые данные внутри таблиц выравниваются по правому краю так, чтобы одноименные разряды чисел находились на одной вертикали. Текст внутри таблиц выравнивается по левому краю. Если текст имеет вид предложения, размещенного на нескольких строчках, то он должен иметь отступ «красной» строки, равный 1-1,5 см.

В заголовках граф таблицы не допускаются переносы.

В текст расчетно-пояснительной записки могут быть включены небольшие фрагменты кода, если они помогают раскрыть содержание. При этом фрагменты кода должны оформляться как рисунок, и текст документа должен иметь ссылку на фрагмент кода, как на рисунок. Рекомендуемый шрифт для выполнения фрагмента кода - Courier New, размер шрифта 12-13 пунктов, междустрочное расстояние одинарное, начертание полужирное, масштаб символов подбирается в диапазоне примерно 66% (чтобы отдельные строчки кода умещались по ширине листа). Каждая строка кода должна иметь отступ 1,5 см от левого края. Пример выполнения фрагмента кода приведен на рисунке 3.2.

```
int lexan() {
    while (1) {
        char c = next_char();
        if (END_OF_TEXT == c) {
            /* завершение разбора */
        } else if (CH_SPACE == c || CH_TAB == c) {
            /* пробельный символ */
        } else if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z') {
            /* идентификатор */
        } else if (c >= '0' && '9' >= c) {
            /* константа */
        }
    }
    return 0;
}
```

Рисунок 3.2 - Пример оформления фрагмента кода

Вместо фрагментов кода предпочтительнее вставлять в текст документа алгоритмы, выполненные на виде алгоритмической записи или в виде блок-схемы.

Алгоритмы, выполненные в виде алгоритмической записи, оформляются так же, как и фрагменты кода. Блок-схемы оформляются в соответствии со ГОСТ 19.701 90 (приложение Е). Непосредственно в тексте допускается размещать блок-схемы, целиком умещающиеся на одной странице. При этом блок схема по

мечается, как рисунок, и текст документа ссылается на нее, как на рисунок. Объемные блок-схемы следует размещать в приложениях, помеченных как обязательные, если они необходимы для правильного понимания текста.

В основном тексте нецелесообразно использовать длинные цитаты. Запрещается воспроизведение фрагментов текста, фактов, данных публикаций тех или иных авторов без указания заимствованных источников. На материалы, взятые из литературы и других источников (утверждения, формулы, цитаты и т.п.) должны быть даны ссылки с указанием номера источника по списку использованной литературы. Номер ссылки проставляется арабскими цифрами в квадратных скобках.

Список использованных источников должен содержать только те источники, которые непосредственно использованы студентом и на которую имеются ссылки в тексте. Список составляется в порядке появления ссылок.

В качестве приложений к курсовому проекту может быть иллюстрированный фактический материал, служащий для подтверждения тех или иных положений автора и занимающий определённый объём: схемы данных, схемы программ, схемы взаимодействия программ, схемы ресурсов системы, фрагменты листингов программ, графики, таблицы, диаграммы.

Не рекомендуется в качестве приложений включать в курсовой проект длинные тексты программ.

Каждое приложение начинается с нового листа.

Приложения нумеруются буквами русского алфавита в порядке появления ссылок на них в основном тексте документа. Заголовок приложения состоит из трех строк, составляющих один абзац, выравнивание текста по центру. Первая строка содержит слово «Приложение» и прописную букву, обозначающую номер приложения, например, «Приложение А». Вторая строка содер

жит заключенное в круглые скобки слово «обязательное», «справочное» или «рекомендуемое». Третья строка содержит название приложения с прописной первой буквы. Пример оформления заголовка приложения см. приложение А.

Расчетно-пояснительная записка представляется к защите переплетённой или сброшюрованной.

3.2 Компьютерная презентация

Компьютерная презентация выполняется в Microsoft PowerPoint версии не старше 2003. Она состоит из последовательности слайдов. Рекомендуемое количество слайдов 10-15.

Презентация сопровождает выступление студента и поэтому строится в соответствии с его докладом. Выступление необходимо продумать таким образом, чтобы сформировать 10-15 фрагментов длительностью 30-40 секунд (ориентировочно), на которых будет обращено внимание комиссии. Каждой такой фрагмент выступления должен быть связан соответствующим слайдом презентации.

Каждый слайд должен иметь заголовок из одной строки.

На слайдах следует размещать либо пункты, которые подчеркивают главные моменты фрагмента выступления, либо графический материал в виде рисунков, диаграмм, схем, графиков, формул и т.п. Недопустимо размещать на слайдах текст выступления. Следует помнить о том, что если все слайды презентации содержат только текстовый материал, дублирующий выступление, надобность в презентации отпадает и эффект, который от нее можно было бы получить, полностью утрачивается.

Если слайд содержит пункты, они не должны дословно цитировать выступление. Вместо этого каждый пункт должен являться кратким содержанием предложения, которое выступаю

щий проговаривает полностью. Как правило, пункт слайда содержит не более одной строки, а в идеале - одно слово, являющееся ключевым.

Если слайд содержит графический материал, количество отдельных элементов этого материала должно совпадать с тем, что намеревается показать выступающий. Если докладчик рассказывает о трех составляющих системы, которая отображена на слайде, рисунок при этом должен содержать три элемента, и количество подписей к ним также должно быть равно трем.

Следует помнить о том, что во время выступления докладчик стоит спиной к экрану (и лицом к комиссии). Если во время выступления необходимо показать на какой-то элемент графического материала, следует показать указкой в его сторону, не сходя при этом с места. Чтобы приемная комиссия могла понять, на какой элемент указывает докладчик, этот элемент при необходимости должен быть каким-либо образом выделен. При необходимости выделения нескольких элементов изображения следует подготовить несколько подряд идущих слайдов.

При размещении текста и иллюстративного материала следует помнить о том, что изображение на экране монитора и на экране проектора могут сильно различаться. Поэтому следует избегать мелких надписей, затрудняющих их прочтение, а также сочетания цвета плана и фона, имеющих один цветовой тон. Необходимо обеспечить достаточный контраст так, чтобы все надписи, а также мелкие элементы рисунков, графиков, диаграмм и т.п. были хорошо видны (различимы).

Не рекомендуется использовать для написания текста экзотические шрифты - их может не оказаться на том компьютере, с помощью которого презентация будет демонстрироваться.

При использовании в качестве иллюстративного материала скриншотов экранных форм необходимо помнить о том, что на

формах используются мелкие шрифты. Поэтому полученные рисунки при размещении на слайдах лучше увеличить до масштаба 120-150%. А для этого, вероятно, реальные размеры форм во время получения скриншота нужно уменьшить.

На слайдах недопустимо размещать рекламные элементы изображения, такие, как логотипы.

Анимация слайдов должна быть отключена.

Первый слайд содержит тему курсового проекта, фамилию руководителя и фамилию студента.

Следующий слайд раскрывает предметную область (спецификацию входного языка) и подводит повествование к цели курсового проектирования. Для представления целей и задач курсового проекта отводится один слайд. Далее следует разместить слайд, на котором описываются средства разработки. Оставшиеся слайды посвящаются разработке проекта. В конце могут быть приведены слайды, посвященные результатам проектирования и перспективам проекта. Не следует заключать последовательность слайдом «Спасибо за внимание» или подобным.

Во время выступления следует избегать ситуаций, когда слайд отображается на экране время, недостаточное для его полного прочтения, или отображается дольше необходимого. Время отображения слайда должно в точности соответствовать времени, необходимому для рассказа о его содержимом. Недопустимо рассказывать во время показа слайда о предметах, которые не нашли отображение на слайде, равно как и рассказывать о содержимом слишком долго.

4 Защита курсового проекта

К защите допускаются курсовые проекты, выполненные в установленные сроки и имеющие положительный отзыв руководителя о ходе проектирования. В связи со сложностью проекта к защите могут быть допущены курсовые проекты, выполненные частично (лексический анализ и синтаксический разбор). Решение о возможности допуска проекта к защите принимает руководитель проекта.

Защита (публичная защита) курсовых проектов проводится на открытых заседаниях приемной комиссии, состоящей из трех-четырех преподавателей кафедры в сроки, регламентируемые учебным планом специальности и установленные кафедрой.

Организацией защиты руководит заведующий кафедрой, а в его отсутствие - помощник заведующего кафедрой или один из руководителей курсовых проектов.

Защита включает в себя доклад студента на основе компьютерной презентации и демонстрацию разработанного программного продукта.

В выступлении следует сформулировать цели и задачи курсового проекта, раскрыть его структуру, показать используемые при проектировании решения. Следует уделить внимание выводам, предложениям, рекомендациям, сделанным автором на основе проведенной работы. Длительность выступления составляет 5-7 минут.

После доклада члены комиссии и присутствующие на защите лица задают студенту вопросы, связанные с проектированием. Далее студент демонстрирует работу разработанного программного продукта.

По окончании защиты члены комиссии на закрытом заседании коллективно обсуждают итоги защиты каждого проекта и

оценивают ее большинством голосов по четырехбалльной системе (отлично, хорошо, удовлетворительно, неудовлетворительно). При равном количестве голосов приоритетное право решения предоставлено руководителю проекта.

Итоги обсуждения объявляются открыто.

В тех случаях, когда защита курсового проекта признается комиссией неудовлетворительной, студент может представить к повторной защите тот же проект с доработкой, определяемой комиссией, или же обязан разработать новую тему, которая устанавливается кафедрой. Сроки и условия защиты проекта в этом случае устанавливаются кафедрой по согласованию с заместителем директора по учебной работе.

Список использованных источников

1) Вл. Пономарев. ТПМ. Требования к программным модулям. Методические указания. Озерск: ОТИ МИФИ, 2006. - 68 с.

2) Комаров А.А., Ларьков Н.С., Нуржанова И.А., Пономарев В.В., Сосюрко В.Г. Оформление текстов учебных студенческих работ (общие требования). Методические указания - Озерск: ОТИ МИФИ, 2007. - 44 с.

3) Пономарев В.В. Краткий курс теории языков программирования и методов трансляции. Учебное пособие. В двух книгах. Книга 1. Редакция 1. Озерск: ОТИ МИФИ, 2008. - 136 с., ил.

4) Пономарев В.В. Краткий курс теории языков программирования и методов трансляции. Учебное пособие. В двух книгах. Книга 2. Редакция 1. Озерск: ОТИ МИФИ, 2009. - 136 с., ил.

5) Системное программное обеспечение / А.В. Гордеев, А.Ю. Молчанов. - СПб.: Питер, 2003. - 736 с.: ил.

6) Ахо, Альфред, В., Сети, Рави, Ульман, Джеффри, Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. - М.: Издательский дом «Вильямс», 2003. - 768 с. : ил.

Приложение А (обязательное)
Титульный лист

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ОЗЕРСКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ (филиал)
ГОУ ВПО «МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(государственный университет)»

Кафедра прикладной математики

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Теория языков программирования и методы трансляции»

Тема: «Интерпретатор языка BASIC»

Пояснительная записка
МИФИ.090Х42.1ХХ.ПЗ

Зав. кафедрой к.т.н.	И.О. Фамилия
Руководитель	И.О. Фамилия
Н. контролер	И.О. Фамилия
Рецензент	И.О. Фамилия
Разработал студент гр. 1ПО-ХХД	И.О. Фамилия

20ХХ

Приложение Б (обязательное)
Аннотация на русском языке

Аннотация

М.С. Савельева. Интерпретатор языка BASIC: Курсовой проект. ОТИ МИФИ, 2008, 37 с., 9 ил.

Библиография - 5 наименований.

В курсовом проекте разработан интерпретатор языка BASIC с ограниченными лексикой и синтаксисом.

Предлагается:

а) использовать одну таблицу символов, построенную на основе упорядоченного инкрементного массива;

б) использовать грамматику операторного предшествования для синтаксического разбора выражений;

в) использовать метод расширенного рекурсивного спуска для синтаксического разбора других конструкций языка;

Разработанный интерпретатор позволяет выполнять программы с использованием целых типов данных в пошаговом режиме и в режиме непрерывного исполнения.

Имеется возможность выводить результат вычисления на дисплей.

Изм.	Лист	№ докум.	Подпись	Дата	МИФИ.090Х42.1ХХ.ПЗ		
Разраб.		М.С. Савельева		16.05.08	Интерпретатор языка BASIC	Лит.	Лист 5 / Листов 37
Пров.		И.О. Фамилия				К	
Н. контр		И.О. Фамилия				ОТИ 1ПО-ХХД	
Утв.		И.О. Фамилия					

Приложение В (обязательное)
Оформление пункта «Содержание»

Содержание

Приложение Г (обязательное)
Оформление пункта «Обозначения и сокращения»

LALR - метод восходящего синтаксического анализа Look Ahead LR

LR - метод восходящего синтаксического анализа, основанный на построении множества LR пунктов

SLR- метод восходящего синтаксического анализа Simple LR

ДКА - детерминированный конечный автомат

ДМП - детерминированный автомат с магазинной памятью

КА - конечный автомат

Лексема - лексическая единица текста

МП - автомат с магазинной памятью

НКА - недетерминированный конечный автомат

Токен - лексема, отнесенная к некоторой категории

Приложение Д (обязательное)
Оформление списка использованных источников

Список использованных источников

1) Пономарев В.В. Краткий курс теории языков программирования и методов трансляции. Учебное пособие. В двух книгах. Книга 1. Редакция 28.02.2008. Озёрск: ОТИ МИФИ, 2008. - 136 с., ил.

2) Ахо, Альфред, В., Сети, Рави, Ульман, Джеффри, Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. — М.: Издательский дом «Вильямс», 2003. - 768 с.: ил.

3) (www.wikipedia.ru/basic).

Проверено 12.03.2008.

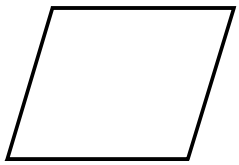
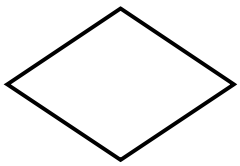
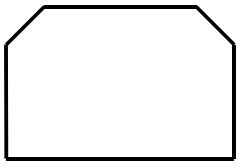
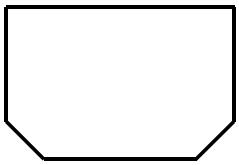
4) Системное программное обеспечение / А.В. Гордеев, А.Ю. Молчанов. - СПб.: Питер, 2003. - 736 с.: ил.

5) Опалева Э.А., Самойленко В.П. Языки программирования и методы трансляции. - СПб.: БХВ-Петербург, 2005. - 480 с.: ил.

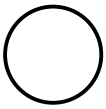
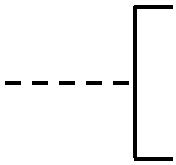
Приложение Е (рекомендуемое)
Выполнение блок-схем

Для выполнения блок-схем (схем программ) используются символы, приведенные в таблице Е.1.

Таблица Е.1 - Основные символы блок-схем

Символ	Название и описание символа
	Данные. Отображает данные алгоритма, например, параметры процедуры, ввод или вывод данных.
	Процесс. Отображает действия алгоритма.
	Предопределенный процесс. Соответствует вызову процедуры.
	Решение. Отображает функцию, имеющую один вход и ряд выходов, из которых один и только один может быть активизирован (соответствует условному оператору, оператору выбора).
	Граница цикла. Обозначает начало цикла. Содержит имя цикла и условие для цикла с предусловием.
	Граница цикла. Обозначает конец цикла. Содержит условие для цикла с постусловием и имя цикла.

Продолжение таблицы Е.1

	Терминатор. Обозначает начало и конец алгоритма. Может содержать слова «Начало» и «Конец».
	Соединитель. Используется для обрыва алгоритмической нити и продолжения ее в другом месте или на другом листе. Содержит уникальный идентификатор.
	Комментарий. Используется для добавления комментариев и примечаний. Пунктирная линия связана с соответствующим символов или может обводить группу символов.

При составлении блок-схемы следует использовать символы одинакового размера с соотношением сторон 3:2 (ширина:высота).

Все символы должны быть соединены сплошной линий, называемой здесь алгоритмической. Алгоритмическая нить начинается от начального терминатора и заканчивается заключительным терминатором. Алгоритмическая нить должна вертикально входить в центр символа или выходить из центра символа. Если алгоритмической нить направлена снизу вверх или справа налево, она должна завершаться стрелкой.

Алгоритмическая нить по возможности не должна пересекаться. Пересечение двух алгоритмических линий не обозначает её ветвления. В местах соединения (ветвления) алгоритмических нитей они должны быть разнесены по высоте (рисунок Е.1).

Слева над блоком может находиться его буквенное или цифровое обозначение, используемое для ссылки на символ в тексте документа.

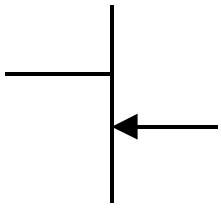


Рисунок Е.1.

Символы содержат текст, поясняющий назначение символа в алгоритме. Если текст не умещается внутри символа, его можно записать в форме комментария или использовать ссылку, которая расшифровывается ниже блок-схемы, но в составе рисунка.

Приложение Ж (справочное)
Приоритеты операций

В таблице Ж.1 приведены приоритеты операций в компиляторе Microsoft Visual C. Высший приоритет имеют операции в начале таблицы, низший - в ее конце. Ассоциативность указывает, в каком порядке выполняются операции одного приоритета.

Таблица Ж.1

Операция	Тип	Ассоциативность
`[] () . -> ++` и `--` (постфиксные)	выражение	слева направо
`++` и `--` (префиксные) `sizeof` `& * + - ~ !`	унарная	справа налево
приведение типа	унарная	справа налево
`* / %`	мультипликативная	слева направо
`+ -`	аддитивная	слева направо
`<< >>`	побитовый сдвиг	слева направо
`< > <= >=`	отношение	слева направо
`== !=`	равенство	слева направо
`&`	побитовое И	слева направо
`^`	побитовое исключающее ИЛИ	слева направо
`|`	побитовое ИЛИ	слева направо
`&&`	логическое И	слева направо
`||`	логическое ИЛИ	слева направо
`? :`	условное выражение	справа налево
`= *= /= %= += -= <<= >>= &= ^= |=`	простое и сложное присваивание	справа налево
`,`	последовательная оценка	слева направо

В таблице Ж.2 приведены приоритеты операций в компиляторе Microsoft Visual Basic 6.0.

Таблица Ж.2

Операция	Тип	Ассоциативность
`^`	экспонента	слева направо
`-`	унарная	слева направо
`* /`	мультипликативная	слева направо
`\`	деление нацело	слева направо
`Mod`	деление по модулю	слева направо
`+ -`	аддитивная	слева направо
`&`	конкатенация	слева направо
`= <> < > <= >= Like Is`	сравнение	слева направо
`Not`	логическая	слева направо
`And`	логическая	слева направо
`Or`	логическая	слева направо
`Xor`	логическая	слева направо
`Eqv`	логическая	слева направо
`Imp`	логическая	слева направо

В таблице Ж.3 приведены приоритеты операций в компиляторе Turbo Pascal 7.1.

Таблица Ж.3

Операция	Тип	Ассоциативность
`@ not`	унарная	слева направо
`* / div mod and shl shr`	мультипликативная	слева направо
`+ - or xor`	аддитивная	слева направо
`= <> <= >= < > in`	отношения	слева направо

Владимир Вадимович Пономарев
Теория языков программирования и методы трансляции.
Курсовое проектирование.
Учебно-методическое пособие.

Отпечатано с готового оригинал-макета.
Издательство ОТИ МИФИ, 2008
Тираж 40 экз.