ИННОВАЦИОННАЯ ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА

Проект 2: индивидуальная траектория обучения и качество образования

Цель: ориентированное на требования рынка образовательных услуг улучшение качества подготовки и переподготовки специалистов

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Кафедра вычислительной техники

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К КУРСОВОМУ ПРОЕКТИРОВАНИЮ ПО ДИСЦИПЛИНЕ «СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

Составитель
К.В. КУЛИКОВ

Владимир 2008

УДК 681.3.06(076.5)
ББК 32.8я2
М54

Рецензент

Кандидат технических наук доцент кафедры информационных систем и информационного менеджмента Владимирского государственного университета

С.Ю. Кириллова

Печатается по решению редакционного совета Владимирского государственного университета

М54 **Методические** указания к курсовому проектированию по дисциплине «Системное программное обеспечение» / Владим. гос. ун-т ; сост. К. В. Куликов. – Владимир : Изд-во Владим. гос. ун-та, 2008. – 16 с.

Рассматриваются вопросы создания программных средств трансляции, компиляции и интерпретации. Основное внимание уделяется этапам проектирования учебного компилятора. Содержатся теоретические основы курса.

Предназначены для студентов 3-го курса всех форм обучения специальности 230101 – вычислительные машины, комплексы, системы и сети при изучении дисциплины «Системное программное обеспечение», могут быть полезны студентам, аспирантам, специалистам, занимающимся вопросами проектирования и разработки средств компиляции, трансляции и интерпретации.

Ил. 2. Библиогр.: 3 назв.

УДК 681.3.06(076.5)
ББК 32.8я2

Введение

Изучение вопросов проектирования и создания системного программного обеспечения требуется для всех специалистов, связанных с вычислительной техникой. В качестве примера разработки рекомендуется пройти все этапы создания компилирующих или интерпретирующих средств. Это в первую очередь связано с углубленным изучением языков программирования. С одной стороны, современные языки программирования являются высокоабстрактными. С другой стороны, процессор ЭВМ до сих пор способен обрабатывать лишь ограниченный набор операций, выполняемых друг за другом. Для того чтобы создавать более рациональные программы, необходимо понимать, как текст на языке программирования высокого уровня будет преобразован специальными системными программами к виду, удобному для исполнения процессором ЭВМ.

В курсовой работе от студента требуется создать компилятор, преобразующий исходный текст программы на языке Pascal в выходной текст программы на языке Ассемблер. Для выполнения данного задания потребуется применить все ранее приобретенные навыки по данному курсу в ходе выполнения лабораторных работ. Для реализации потребуется использовать сложные информационные структуры данных переменного размера – связанные списки, деревья и, возможно, другие. Целесообразность такого выбора диктуется тем, что эти структуры данных составляют информационную основу многих алгоритмов теории формальных языков – одной из основных теорий, составляющих содержание курса по системному программному обеспечению. Поскольку решить задачи компиляции можно различными способами, не следует рассматривать данные материалы как единственно возможные. Эти методические указания призваны помочь студентам, но не ориентируют на конкретное решение. От студента требуется выбрать в каждом конкретном случае свой путь решения поставленных задач и обосновать такой выбор.

1. ТИПОВОЕ ЗАДАНИЕ

Типовое задание на курсовую работу предполагает разработку системы компиляции. Входным языком является подмножество языка Pascal, определяемое индивидуальным вариантом.

В ходе выполнения курсовой работы должен сформироваться проект. Проект (projectus – брошенный вперёд) – система документов, определяющая процесс создания какого-либо продукта (его бумажная модель). Здесь за основу принят структурный подход с его методом пошаговой детализации, что вполне целесообразно в применении к программным системам (ПС) малого и даже среднего размера (до 10 тыс. операторов).

Основные составляющие проекта и этапы разработки ПС:

Цели и требования

1. Назначение.
2. Определение пользователя.
3. Подробное перечисление функций.
4. Публикации.
5. Эффективность.
6. Совместимость.
7. Конфигурация ТО и БПО.
8. Безопасность.
9. Обслуживание.
10. Установка.
11. Надёжность.
12. Мобильность.

Следует стремиться к использованию измеряемых категорий и чётких критериев.

1.1. *Стадии программирования*

Внешний проект

Содержит описание ожидаемого поведения ПС с точки зрения пользователя в форме внешних спецификаций. Проектируемый продукт на этом этапе рассматривается как чёрный ящик. Общая идея этого этапа состоит в достаточно детальном определении входов

и выходов ПС, включая определение входных и выходных языков, структуры входных и выходных файлов, содержание экранных форм и т.д. Иногда этот этап называют созданием прототипа ПС. В последующем внешний проект служит основой создания проекта архитектуры ПС и одновременно источником информации для специалистов, составляющих комплекс документации для пользователя.

Основные изобразительные средства этого этапа: чертежи, схемы, рисунки, словесные описания, формализованные описания (математические выкладки, БНФ, расширенная БНФ (РБНФ), синтаксические диаграммы Вирта).

РБНФ позволяет заменить рекурсию итерацией. К метасимволам БНФ добавлены скобки факторизации “{“ и “}”, в которые заключают повторяющуюся часть грамматической конструкции, в том числе и нуль раз.

Пример

<Составной оператор> ::= **begin** {<Оператор> ; } <Оператор> **end** .

Синтаксические диаграммы Вирта впервые были использованы при определении синтаксиса АЯ Pascal. Это графическая форма РБНФ.

Проект архитектуры

Под архитектурой понимается распределение функций в системе. Для ПС это означает разбиение её на структурные и функциональные модули, в качестве которых могут быть единицы компиляции (файлы), процедуры, объекты, типы (классы) и т.д.

Проект архитектуры включает:

- определение метода преобразования данных;
- определение основных структур данных;
- разработку иерархической схемы вызовов модулей;
- составление спецификаций модулей.

Проектирование модулей

На этом этапе выполняется проектирование и разработка отдельных модулей ПС. Методология проектирования и разработки подсистемы в общих чертах повторяет методологию проектирования и разработки системы в целом.

Кодирование

Запись программы на языке разработки.

Комплексная сборка, тестирование и отладка

ПС среднего размера обычно отлаживают по мере сборки. Есть два классических метода сборки – **нисходящий** и **восходящий**. Нисходящий метод базируется на использовании **заглушек** – процедур, которые в очень упрощённой форме моделируют поведение реальных процедур нижних уровней ПС. По мере продвижения разработки ПС заглушки заменяются реальными процедурами. Восходящий подход базируется на использовании **драйверов** – программных единиц, моделирующих поведение реальных программных единиц вызывающего уровня. На практике используют всевозможные комбинации этих двух классических методов. ПС малого размера отлаживают обычно **методом большого скачка**, т.е. сразу всю систему целиком, в некоторых случаях предварительно отлаживая лишь процедуры самых нижних уровней.

Проект должен содержать необходимую систему тестов для проведения отладки системы на всех уровнях, включая **внешнее тестирование**. Этот вид тестирования относится ко всей системе в целом.

Сопровождение

Сопровождение ПС предполагает внесение изменений на этапе её эксплуатации. Такая необходимость может быть вызвана изменением условий применения или обнаружением ошибок. Важность этого этапа вытекает из структуры трудозатрат на обеспечение жизненного цикла ПС среднего размера: проектирование – 15, кодирование – 10, тестирование и отладка – 25, сопровождение – 50 %. В этой связи разумно предъявить достаточно жёсткие требования к качеству сопровождающей систему документации.

1.2. *Варианты заданий*

Варианты индивидуальных заданий для курсового проектирования (характеризует набор конструкций входного языка):

1. Массив, FOR, IF, BREAKE.
2. Массив, WHILE, сравнение.
3. Массив, REPEAT, сравнение.
4. Метка, IF, сравнение, GOTO.
5. Константа, CASE, GOTO, метка.
6. Процедура, EXIT, передача данных по указателю, константа.
7. Функция, передача данных по указателю, константа.
8. IF, сравнение, OR, AND.
9. IF, константа, XOR, NAND.

2. ОБЗОР ПРОЦЕССА КОМПИЛЯЦИИ

Транслятором называется системная программа, выполняющая преобразование программы, написанной на одном алгоритмическом языке, в программу на другом алгоритмическом языке, в определенном смысле эквивалентную первой.

Компилятором называется системная программа, выполняющая преобразование программы, написанной на одном алгоритмическом языке, в программу на языке, близком к машинному, и в определенном смысле эквивалентную первой.

Преобразование программы выполняется в несколько этапов (фаз). Один из вариантов приведен ниже (рис. 1).

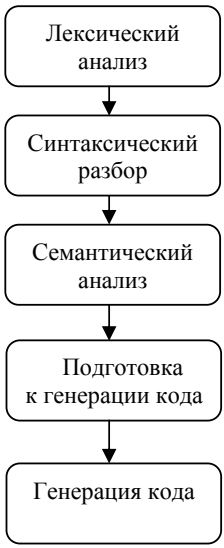

Рис. 1. Этапы компиляции

2.1. *Лексический анализ*

Лексический анализ (ЛА) – это часть общего процесса компиляции, в ходе которого выполняются распознавание и перевод во внутреннее представление конструкций языка типа 3 по Хомскому. Такие конструкции называют **лексемами**. Наиболее типичные классы лексем: имена (идентификаторы), константы, служебные слова. Процедура (сопрограмма), выполняющая ЛА, называется **лексическим анализатором**, или **сканером**.

Точный перечень лексем, которые необходимо распознавать, зависит от языка программирования и от грамматики, используемой для его описания. Например, для языка Pascal лексемами являются PROGRAM, INTEGER, REAL, VAR, DO, READ и т.д. Каждый код лексемы обычно представляется кодом таблицы и спецификатором. Спецификатор задает номер строки в таблице, в которую занесена лексема. Это позволяет избежать дополнительного поиска по таблицам на последующих этапах трансляции. Обычно сканер работает с тремя таблицами: терминальных символов (служебных слов),

символических имен и литералов (констант). Сканер приводит программу к некоторому стандартному виду, что существенно упрощает её последующий грамматический разбор. Таблица терминальных символов в простейшем случае может иметь следующий вид.

Таблица терминальных символов

№ п/п	Терминальный символ	Признак разделителя
1	PROGRAM	-
2	VAR	-
3	BEGIN	-
4	END	-
5	INTEGER	-
6	FOR	-
7	TO	-
8	DO	-
9	WHILE	-
10	DIV	-
11	MOD	-
12	AND	-
13	OR	-
14	:=	+

Таблица символических имен может иметь следующий вид.

Таблица символических имен

№ п/п	Символическое имя	Адрес	Тип	Дополнительная информация
1	I		int	
2	Y		real	
3	X1		int	
4	…	…	…	…

Таблица литералов по структуре аналогична таблице символических имен.

Таблица литералов

№ п/п	Литерал	Адрес	Тип	Дополнительная информация
1	1		int	
2	100		int	
3	…	…	…	…

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

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

будет получена строка

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

где в угловых скобках пара чисел задает код таблицы и спецификатор.

В дополнение к своей основной функции – распознавание лексем – сканер также выполняет чтение строк и печать листинга исходной программы.

Функционально в сканере могут быть выделены следующие модули:

1) выделение из входной строки очередного слова;
2) поиск в таблицах лексем и определение кода лексемы;
3) формирование и вывод выходной строки.

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

Для поиска в таблицах может быть использован как наиболее простой линейный алгоритм, так и более усовершенствованные алгоритмы.

При заполнении таблиц выполняется проверка на наличие в них элементов, совпадающих с выделенным идентификатором или константой, и при их совпадении занесение в таблицу не выполняется.

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

2.2. *Синтаксический разбор*

Синтаксический анализатор (синтаксический разбор) – это часть компилятора, которая отвечает за выявление основных синтаксических конструкций входного языка. Задачи синтаксического анализа: найти и выделить основные синтаксические конструкции в тексте входной программы, установить тип и проверить правильность каждой синтаксической конструкции и, наконец, представить синтаксические конструкции в виде, удобном для дальнейшей генерации текста результирующей программы. Этап синтаксического разбора необходим для преобразования последовательности лексем во внутреннее представление программы. Внутренним представлением могут быть бинарное дерево, триады, тетрады, обратная польская запись или другое. От выбора внутреннего представления программы зависит дальнейшая реализация синтаксического разбора и генерации кода.

В основе синтаксического анализатора лежит распознаватель текста входной программы на основе входного языка. Как правило, синтаксические конструкции языков программирования могут быть описаны с помощью КС-грамматик, реже встречаются языки, которые могут быть описаны с помощью регулярных грамматик. Чаще всего регулярные грамматики применимы к языкам Ассемблера, а языки высокого уровня построены на основе синтаксиса КС-языков. Выходом лексического анализатора является таблица лексем, или цепочка лексем. Эта таблица (цепочка) образует вход синтаксического анализатора, который исследует только один компонент каждой лексемы – её тип. Остальная информация о лексемах используется на более поздних фазах компиляции при семантическом анализе, подготовке к генерации кода и генерации кода результирующей программы. Синтаксический анализ (или разбор) – это процесс, в котором исследуется таблица лексем и устанавливается, удовлетворяет ли она структурным условиям, явно сформулированным в определении синтаксиса языка.

Задача синтаксического анализа упрощается, если программа предварительно обработана лексическим анализатором и поступает на вход синтаксического анализатора в форме последовательности кодов лексем.

Результатом работы распознавателя КС-грамматики входного языка является последовательность правил грамматики, примененных для построения входной цепочки. По найденной последовательности, зная тип распознавателя, можно построить цепочку вывода или дерево вывода. В этом случае дерево вывода выступает в качестве дерева синтаксического разбора и представляет собой результат работы синтаксического анализатора в компиляторе.

В синтаксическом дереве внутренние узлы (вершины) соответствуют операциям, а листья представляют собой операнды. Как правило, листья синтаксического дерева связаны с записями в таблице идентификаторов. Структура синтаксического дерева отражает синтаксис языка программирования, на котором написана исходная программа. Синтаксические деревья могут быть построены для любой части входной программы. В том случае, когда синтаксическому дереву соответствует некоторая последовательность операций, влекущая порождение фрагмента объектного кода, говорят о дереве операций.

Возьмем в качестве примера синтаксическое дерево, построенное из цепочки (a+a)*b в языке Pascal. Семантически незначащими символами в этой грамматике являются скобки (они задают порядок операций и влияют на синтаксический разбор, но результирующего кода не порождают). Знаки операций заданы символами + и *, остальные символы (a и b) являются операндами. В результате получим дерево операций (рис. 2).

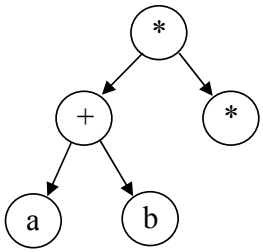

Рис. 2. Пример дерева операций

2.3. *Генерация кода*

Как уже было сказано выше, реализация генератора кода зависит от принятого внутреннего представления программы, получаемого на этапе синтаксического разбора. Но принципы построения

такого генератора схожи. Необходимо, пройдясь по внутреннему представлению программы, получить на выходе однозначно соответствующее ему тело выходного кода.

Соответствие между операторами алгоритмического языка и операторами языка Ассемблер оценивается как 1:N, т.е. один оператор алгоритмического языка заменяется N операторами языка Ассемблер.

Одним из наиболее простых подходов к решению задачи подстановок является представление операторов или операций алгоритмического языка в виде макрокоманд, а операндов – в виде параметров макрокоманды. При обращении к такой макрокоманде выполняется замена ее макроопределением, т.е. последовательностью операторов Ассемблера. Все макроопределения могут сводиться в библиотеку данного транслятора. Например, оператор присваивания S:=F MOD D может быть заменен последовательностью операторов Ассемблера:

```
MOV AX,F
IDIV D
MOV C,DX
```

Поскольку обозначения операндов не являются постоянными, в операторах языка Ассемблер следует проставить обобщенные имена операндов, а при трансляции обобщенные имена заменяют соответствующими им из оператора алгоритмического языка. Для рассматриваемого примера может быть построено следующее соответствие:

<ID1>:=<ID2> MOD <ID3>	**<====>**	**MOV AX,<ID2>**
		IDIV <ID3>
		MOV <ID1>,DX

где ID1, ID2, ID3 обозначают соответственно 1, 2 и 3-й операнды.

В случае разработки кодовых образцов для операторов, содержащих проверку некоторых условий, может возникать проблема обеспечения уникальности меток перехода, которые используются внутри кодового образца: например, для оператора IF ... THEN можно построить следующий кодовый образец:

IF <условие> THEN <оператор> <====> команды вычисления условия
JNZ <метка>
команды вычисления оператора
<метка>: NOP

В кодовом образце предполагается, что при вычислении условия будет установлен флаг равенства нулю, если условие истинно, и сброшен, если ложно. Тогда при каждой замене оператора IF ... THEN кодовым образцом будет тиражироваться <метка>. Для обеспечения уникальности меток может быть использован простой прием: образование метки из какого-либо символа, за которым следует значение счетчика обращений к кодовому образцу. В качестве первого символа необходимо выбирать такой символ, который допустим в качестве первого символа метки в языке Ассемблер, но не допустим в алгоритмическом языке. Например, в языке Ассемблер IBM PC первыми символами метки могут быть: ? , . , @ , $.

В данной работе не описаны этапы семантического анализа, подготовки к генерации кода и оптимизации, которые используются в реальных компиляторах. Студентам рекомендуется их использовать для более детального изучения вопросов курса системного программного обеспечения.

Библиографический список

1. *Гордеев, А. В.* Системное программное обеспечение : учеб. для вузов / А. В. Гордеев, А. Ю. Молчанов. – СПб. : Питер, 2003. – 736 с.

2. *Барков, В. А.* Системное программное обеспечение : конспект лекций / В. А. Барков. – Владимир : Изд-во Владим. гос. ун-та, 2000. – 56 с.

3. *Брежнев, А. М.* Системное программное обеспечение : конспект лекций / А. М. Брежнев. – Северодонецк : Северодонец. технолог. ин-т, 1995. – 42 с.

Оглавление

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К КУРСОВОМУ ПРОЕКТИРОВАНИЮ ПО ДИСЦИПЛИНЕ
«СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

Составитель
КУЛИКОВ Константин Владимирович

Ответственный за выпуск – зав. кафедрой профессор В.Н. Ланцов

Подписано в печать 11.04.08.
Формат 60x84/16. Усл. печ. л. 0,93. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.