Теория формальных языков и компиляторов

Преподаватель: Достовалов Дмитрий Николаевич, к.т.н., доцент кафедры АСУ.

Лекция 1. Введение. О чем этот курс?

1.1 Формальные языки

Наш курс называется «Теория формальных языков и компиляторов». Рассмотрим значения слов, составляющих название, и поймем, что мы будем изучать в этом курсе.

Теория – это учение, система идей или принципов. Теория включает взаимосвязанные понятия, гипотезы и законы. В теории одни суждения выводятся из других суждений на основе практических подтверждений и/или правил логического вывода.

Обычным способом проверки теорий является прямая *экспериментальная проверка* («эксперимент – это критерий истины»). Однако часто теорию нельзя проверить прямым экспериментом (например, теорию о возникновении жизни на Земле). Либо такая проверка слишком сложна (макроэкономические и социальные теории). К счастью, в нашем курсе все будет достаточно просто. Именно экспериментом (в лабораторных и курсовой работе) мы покажем, что теория действительно работает.

Язык – это знаковая система для обмена информацией.

Язык – это *множество слов и текстов*, составленных из этих слов.

Формальный язык – это язык, который образован некоторым формальным образом. «Формальный» означает:

1. Обобщение (абстрагирование). Математические объекты образуются путем обобщения реальных объектов. Изучая какой-нибудь объект, математик замечает только некоторые его свойства, а от остальных отвлекается.

2. Строгость рассуждений. Выводы не проверяются экспериментальным путем, но принято доказывать их справедливость строгими, подчиняющимися определенным правилам, рассуждениями. Эти рассуждения называются доказательствами и доказательства служат единственным способом обоснования верности того или иного утверждения.

Формальный язык – это *математическая модель* реального языка. Под реальным языком здесь понимается некий способ общения (коммуникации) субъектов друг с другом. Субъектами могут быть, например, люди или компьютеры. Для общения субъекты используют конечный набор знаков (*символов)*, которые проговариваются (*записываются*) в заданном порядке. Такие последовательности обычно называют *словами* или *предложениями*.

Языки программирования являются формальными языками.

Язык программирования – это формальная знаковая система, предназначенная для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит ЭВМ под её управлением.

В теории формальных языков представляется важным изучить законы расположения слов рядом друг с другом, т.е. *синтаксические* свойства текстов. Поэтому формальный язык задается как множество последовательностей (*строк*), составленных из элементов конечного *алфавита*.

Итак, формальные языки – это просто множества строк (цепочек), составленных из символов некоторого конечного алфавита. Но возникает вопрос: *как можно задать формальный язык*? Если язык содержит конечное число строк, то можно просто выписать их все одну за другой (конечно, можно задуматься, имеет ли смысл выписывать цепочки языка, имеющего хотя бы десять тысяч элементов и, вообще, есть ли смысл в таком выписывании?). Но ведь язык может содержать бесконечное число строк.

Например, на языке C++ можно написать бесконечно много программ. Что делать, *если язык бесконечен, как его задавать*? В теории формальных языков имеются различные методы определения (задания) языка. В данном курсе мы изучим три способа:

1. Порождающие грамматики.
2. Конечные автоматы.
3. Регулярные выражения.

Таким образом, мы будем изучать и применять математические методы для построения (создания) и исследования языков.

1.2 Компиляция программ

Вы когда-нибудь задумывались над тем, как именно создаются *исполняемые файлы* с расширением .exe из исходных кодов, например на языке C++ (файлов .cpp и .h)? Какова роль компилятора и компоновщика?

Из курсов информатики и программирования вы знаете, что объединенная единым алгоритмом совокупность описаний переменных и операторов образует программу. Для того чтобы выполнить программу, требуется перевести ее на язык, понятный процессору – в машинные коды. Этот процесс состоит из нескольких этапов. Рисунок ниже иллюстрирует эти этапы для языка C++.

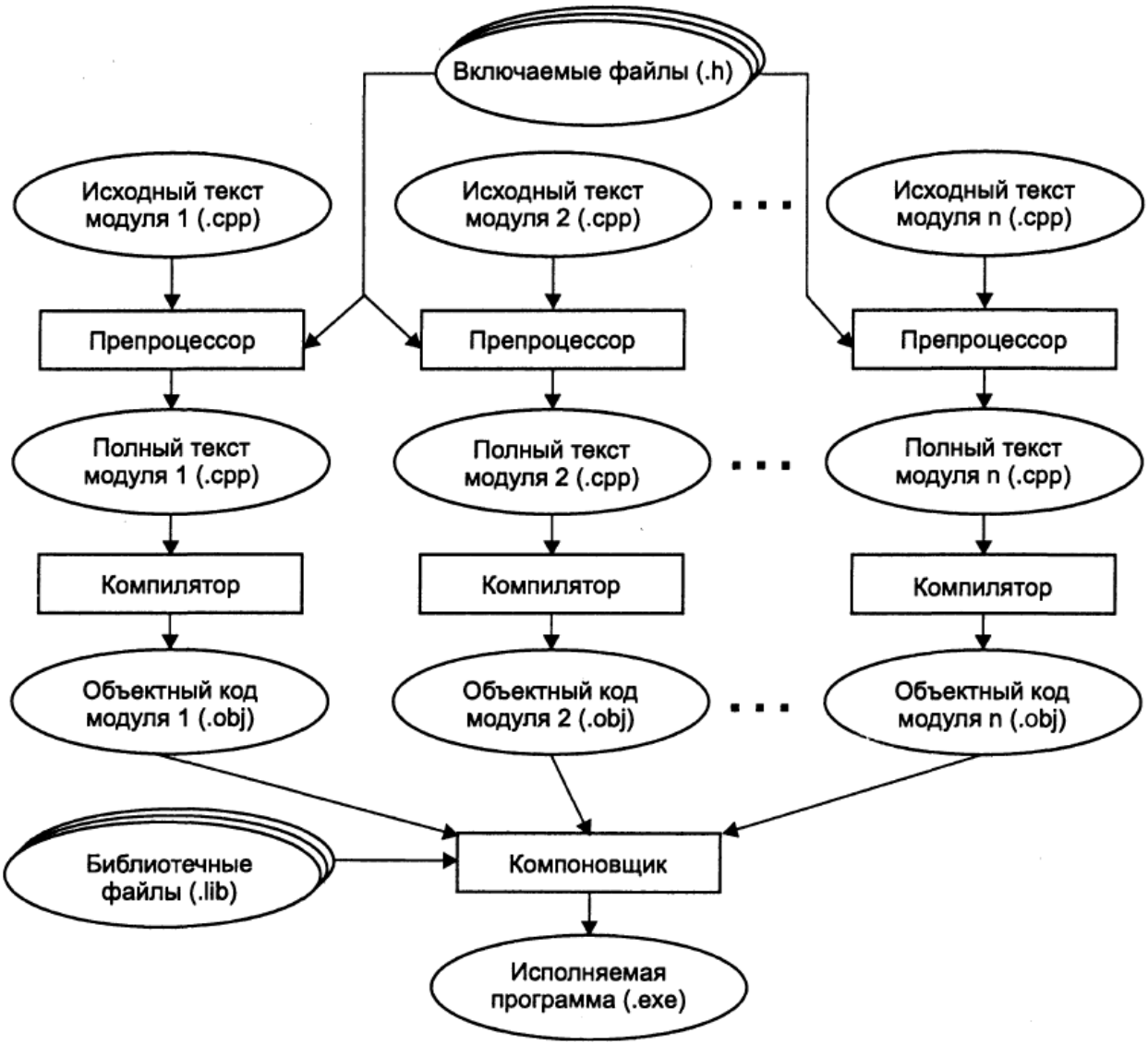

Рис. 1.1. – Этапы создания исполняемой программы

Сначала программа передается *препроцессору*, который выполняет директивы, содержащиеся в ее тексте. Пример директивы препроцессора – команда #include (включение в текст так называемых заголовочных файлов, содержащих описания используемых в программе элементов). Другая часто используемая директива – #define (замена одной последовательности символов на другую).

Получившийся полный текст программы поступает на вход *компилятора*, который выделяет *лексемы* (слова, специальные символы и другие элементы текста программы). Затем на основе грамматики языка компилятор распознает выражения и операторы, построенные из этих лексем.

При этом компилятор выявляет *синтаксические* ошибки, и в случае их отсутствия строит *объектный модуль* (файл .obj).

Компоновщик формирует исполняемый модуль программы, объединяя объектные модули. Например, библиотечные функции содержатся в отдельных объектных модулях. Их нужно объединить с объектными модулями, построенными из исходных кодов. Если программа состоит из нескольких исходных файлов, они компилируются по отдельности и объединяются на этапе компоновки. Исполняемый модуль имеет расширение .exe и запускается на выполнение обычным образом.

Таким образом создаются исполняемые программы на C++. Конечно, это очень общее описание сложного процесса. Здесь нам нужно было определить роль компилятора в процессе создания программы. Кстати, анализируя название курса, мы не обратили внимание на слово «компилятор». Так вот, теперь можно сказать: в курсе также будут изучаться устройство и основные этапы работы компиляторов.

1.3 Виды языковых процессоров

Определение 1.1. *Языковым процессором* называется программа по обработке текстов на одном из языков программирования.

Устройство (структура и логические связи) языковых процессоров непосредственно связаны с языками программирования. Языки программирования, в свою очередь, зависят от предметной области (таблица 1.1).

Таблица 1.1. Использование языков в различных областях

Область применения	**Основные языки**
Обработка деловой информации	COBOL, C/ C++, C#, Java
Научные вычисления	Spreadsheet, PL/1, ALGOL, FORTRAN, BASIC, C/C++, C#, Java, Python
Системное программирование	C/C++, Java, ADA, Modula, Assembler, Forth, Refal
Искусственный интеллект	LISP, Prolog
Издательская деятельность	TeX, Postscript
Создание процессов	UNIX, shell, TCL, Perl, AWK
Языки моделирования	Smalltalk, UML, Modelica

Системная программа выполняет роль перекодировки входного формата в некоторый выходной набор данных. Такие программы разрабатываются на основе синтаксически ориентированных методов. Охарактеризуем некоторые из них – **компиляторы**, **интерпретатоы**, **ассемблеры** и **препроцессоры**.

Определение 1.2. *Компилятор* – это такой языковый процессор, в котором входной формат данных – язык программирования высокого уровня, а выходной формат – объектный код.

Определение 1.3. *Интерпретатор* – это такой языковый процессор, в котором входной формат данных – язык программирования высокого уровня, а выходной формат – промежуточный или исполнимый код.

Процесс интерпретации состоит в обработке каждого оператора и его немедленном исполнении. Интерпретатор используется в отладочных целях. Так как в интерпретаторе обычно используются только блоки анализа исходного текста, то он является более быстрым процессором на стадии

трансляции, нежели компилятор. И как решатель задач используется чаще в пооператорных режимах.

Определение 1.4. *Ассемблер* – это такой языковый процессор, в котором входной формат данных – команды ассемблера, близкие к кодам машины. Выходной формат – исполнительный код или машинные команды, доступные для загрузки в оперативную память и исполнения. Ассемблеры используются в задачах управления ресурсами вычислительной техники. И поскольку входные программы на ассемблере близки к кодам машины, они обладают наивысшей степенью доступа к командам управления всей периферией вычислительной техники.

Определение 1.5. *Препроцессор* – это такой языковый процессор, в котором входной формат данных – специализированные или проблемные языки, выходной – объектный код, как у компилятора, либо данные, аналогичные выходным данным интерпретатора. Структура препроцессора представлена на рис 1.2.

Рис. 1.2. – Структура препроцессора

Препроцессоры используются для решения предметных задач с языков пользователя, когда пользователь не является профессионалом в области вычислительной техники и программирования и имеет доступ лишь к предметным категориям своего языкового процессора.

1.4 Устройство и логические связи компилятора

Рассмотри устройство компилятора как наиболее сложного языкового процессора. На рис. 1.3 приведены логические связи компиляторов. Сплошными стрелками показана передача управления, пунктирными – информационные связи.

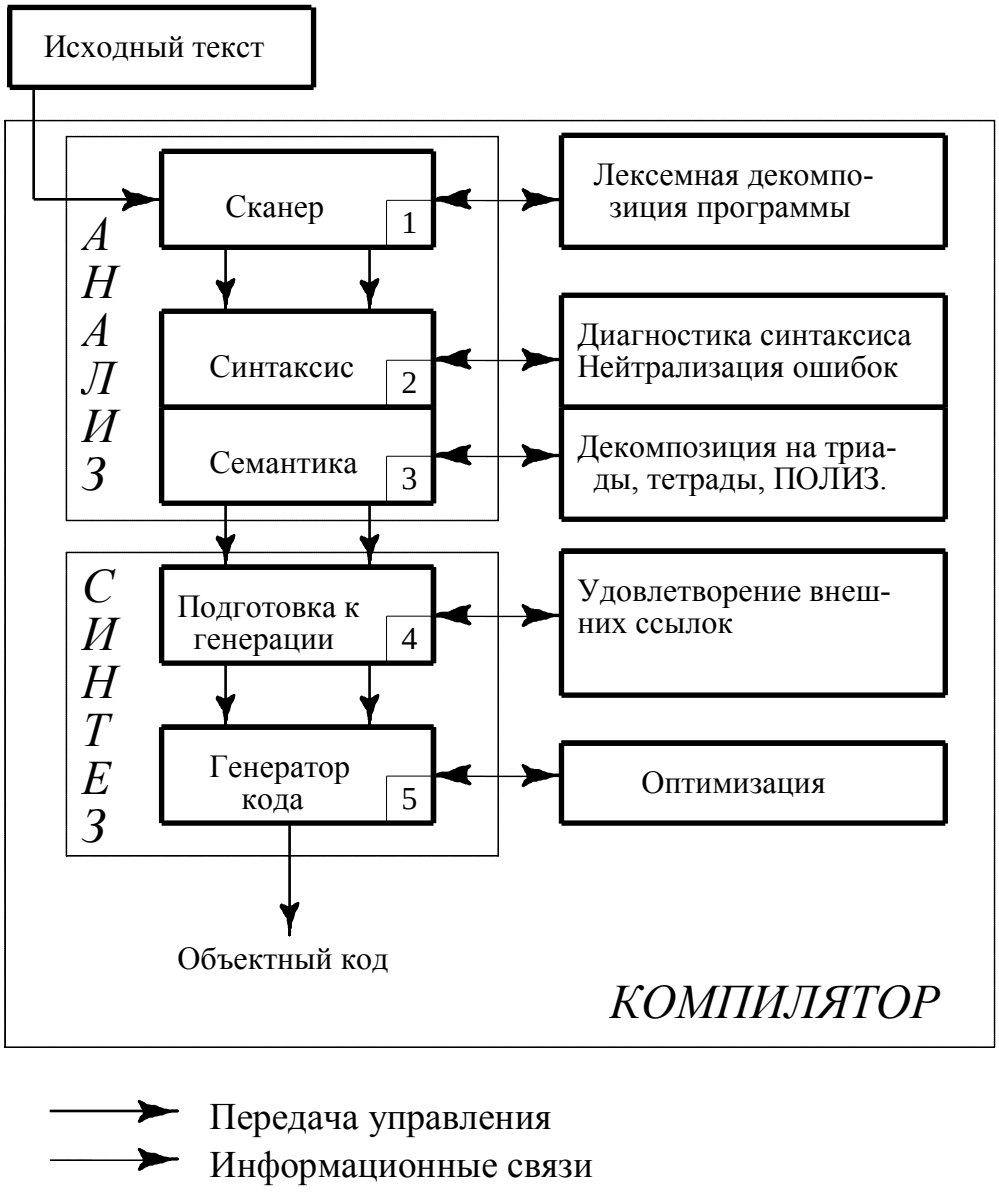

Рис. 1.3. – Логические связи компилятора

Охарактеризуем каждый блок в отдельности и при этом определим сопутствующие понятия:

1. Исходный текст или программа поступает в компилятор на некотором входном языке первоначально в сканер или лексический

анализатор (блок 1), который выполняет декомпозицию текста на лексемы (токены) или просто символы.

Лексема (токен) – некоторый элемент программы, который характеризует определённые категории языка. Традиционно лексемами являются идентификаторы, числовые константы, ключевые слова, знаки операций и т.д. В свою очередь лексема состоят из литер. Понятие «токен» широко используется в иностранной литературе. В отечественных источниках вместо употребляют эквивалентное понятие - лексема. Токен или лексема в общем случае является подмножеством множества терминальных символов, строгое определение которых будет рассмотрено в следующей лекции.

Литера (литерал) – неделимая информационная единица, из которых набирается текст программы. В простом информационном понятии это информационное терминальное поле на клавиатуре компьютера или компьютерный алфавит ASCII и EBCDIC. Иначе говоря, это знаковые клавиши на клавиатуре терминала, исключая служебные и функциональные. Токены (лексемы) могут состоять из одной или более литер, более строгое математическое понятие этой категории будет приведено ниже.

Функция сканера – задать определённый внутренний код определённым объектам программного сегмента. Так, ключевые слова в языке C++ имеют один внутренний код, разделители – другой, числовые константы – третий и так далее. Кроме того, сканер фильтрует (очищает) программу от незначащих символов (пробелов, знаков табуляции и переводов на новую строку).

2. *Синтаксис.* Синтаксический анализатор проводит проверку на соответствие программы на входном языке грамматическим правилам этого языка. Совместно с процессором синтаксических ошибок занимается обработкой несоответствия синтаксиса грамматики, нейтрализацией этих мест в программе и сообщением пользователю об этих ошибках.

3. *Семантика.* Семантический анализатор предполагает смысловую обработку. Из семантически правильных конструкций генерируется промежуточный набор данных, который представлен в виде триад, тетрад, ПОЛИЗ и другом внутреннем представлении (чаще древовидном), удобном для последующей обработки. Промежуточный набор данных служит для упрощения генерации кода.

4. *Подготовка к генерации кода.* Все внешние модули, макросы, встроенные и внешние функции объединяются, и таким образом программа из нескольких функций (процедур) собирается в единый программный сегмент.

5. *Генератор кода.* Преобразует промежуточный код в набор объектных кодов, то есть переводит программный сегмент в объектный код.

На рис 1.3. приведены все стадии обработки до объектного кода. Кодовый образ программы после компиляции оказывается эффективным. В связи с этим компилятор предназначен для решения вычислительных задач, когда загрузочный модуль эффективно распределяет вычислительные ресурсы.

Упражнения

1. Что такое формальный язык?
2. Перечислите, какие современные языки программирования вы знаете?
3. Какие способы задания языка изучаются в этом курсе?
4. Что такое языковой процессор?
5. Перечислите виды языковых процессоров.
6. Перечислите, какие языковые процессоры (например, компиляторы) вы знаете?
7. Какой языковой процессор самый быстрый?
8. Чем сканер отличается от синтаксического анализатора?
9. Какой языковой процессор генерирует самый эффективный код?

10. Чем отличается компилятор от ассемблера?

11. Чем отличается интерпретатор от компилятора?

12. В чем особенности трансляции в препроцессорах?

Список использованных источников

1. Теория. Материал из Википедии – свободной энциклопедии. https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F

2. Лапшин В. Формальные языки и грамматики. Электронный ресурс. Режим доступа: https://habrahabr.ru/post/177109/

3. Этапы компиляции и компоновки программ на языке C++. Электронный ресурс. Режим доступа: http://itandlife.ru/programming/cpp/etapy-kompilyacii-i-komponovki-programm-na-yazyke-c/

4. Шорников Ю.В. Теория и практика языковых процессоров.