Московский государственный университет геодезии и картографии (МИИГАиК)

кафедра информационно-измерительных систем (ИС)

дисциплина: «Языки программирования»

Лекция 4:

«Трансляция языков программирования»

1. Компилируемые и интерпретируемые языки программирования
2. Виды трансляторов
3. Фазы трансляции

3.1. Синтаксис и семантика языков программирования

3.2. Фаза анализа программы

3.3. Фаза синтеза программы

4. Выполнимые файлы

бакалавриатура, 2 курс - 2016-2017 уч.год

1. Компилируемые и интерпретируемые языки программирования

Программа, написанная на языке высокого уровня (Object Pascal, Delphi, C++, C#, Java и др.) перед исполнением должна быть преобразована в программу на машинном языке.

Трансляция (или компиляция) программы – *процесс*, в результате которого выполняется разбор кода программы и формируется *выполнимый* или *интерпретируемый* код.

По *типу выходных данных* существует два основных вида трансляторов:
- компилирующие *окончательный выполнимый код*;
- компилирующие *интерпретируемый код* (требуется доп. ПО)

***ПРИМЕРЫ*:**

***Окончательный выполнимый код* – приложения, реализованные как: EXE-файлы, DLL-библиотеки, COM-компоненты**

Интерпретируемый код:

– байт-код Java-программ, выполняемых на виртуальной машине JVM (Java Virtual Machine);

– код управляемых приложений на C# или C++, использующие среду выполнения CLR (Common Language Runtime);

– JavaScript, LISP, Perl, PROLOG, Smalltalk.

1. Компилируемые и интерпретируемые языки программирования

Знать к экзамену(!)

соответственно:

Компилируемые языки программирования **– язык, исходный код которого преобразуется компилятором в машинный код и записывается в файл, с особым заголовком и/или расширением, для последующей идентификации этого файла операционной системой, формирующей *окончательный исполняемый/выполнимый код* для непосредственного выполнения центральным процессом компьютера.**

Примеры: C, C++, Pascal, Object Pascal, Delphi, FORTRAN, Ada и мн.др.

Интерпретируемые языки программирования **– язык, в котором исходный код программы не преобразуется в машинный код для непосредственного выполнения центральным процессором (как в компилируемых ЯП), а выполняется/исполняется с помощью специальной программы-интерпретатора, т.е. это язык, реализующий *интерпретируемый (управляемый) код* .**

Примеры: Java, LISP, Perl, PROLOG, C# и мн. др.

Особенность среды программирования Visual Studio:
позволяет создавать на C++, как традиционные приложения с *выполнимым кодом*, так и приложения *интерпретируемого (управляемого) кода*.

Вывод: Язык программирования C# является и компилируемым, и интерпретируемым.

2. Виды трансляторов

Исходная программа состоит из нескольких программных модулей. Программный модуль – это программный код на языке высокого уровня

***Процесс трансляции (≡ компиляции)* может выполняться в 2-х вариантах:**
– как единое целое – компиляция каждого модуля и редактирования связей;
– как два отдельных этапа (реализован в C и C++):
1) компиляция объектных модулей,
2) вызов редактора связей, создающего окончательный вид.

***Объектный код*, создаваемый компилятором – область данных и область машинных команд, имеющих адреса, которые в дальнейшем «согласуются» редактором связей.**

***Редактор связей* (≡загрузчик) – по отдельности откомпилированные объектные модули и подключаемые библиотеки размещает в едином адресном пространстве.**

Трансляторы подразделяют на четыре типа:
1. **Ассемблер**
2. **Компилятор**
3. **Загрузчик**
4. **Препроцессор (макропроцессор)**

2. Виды трансляторов

1. Ассемблер – транслятор, выполняющий перевод с языка Ассемблер на машинный язык конкретного компьютера.

***Основные положения*:**

1) **одна инструкция на Ассемблере переводится в одну команду на объектном языке (объектный язык на ассемблере – это машинный язык);**

2) **команды языка ассемблера соответствуют командам *процессора* (в т.н. *мнемокоде* – удобной символьной форме записи команд и аргументов);**

3) **пример инструкции на ассемблере** (записывается в отдельной строке, может иметь метку, комментарий записывается после «;»):

```
LabelA:
push ebp
mov ebp, esp
add esp, 0FFFFFFF8h     ; прибавляется -8 к регистру esp
mov esp, ebp
pop ebp
ret
```

4) **каждая модель *процессора* имеет свой набор команд, поэтому язык ассемблера всегда привязан к конкретной процессорной архитектуре;**

5) **существуют языки ассемблера *высокого* уровня – MASM (Microsoft Macro Assembler). MASM v.8 – в среде программирования Visual Studio .NET.**

2. Виды трансляторов

2. Компилятор – транслятор, выполняющий перевод программы с языка высокого уровня в выполнимую или интерпретируемую форму.

3. Загрузчик – транслятор, выполняющий редактирование связей уже откомпилированных модулей.

***Основные положения*:**

1) исходный программный код для загрузчика представляется на машинном языке, но в «перемещаемой» форме;

2) загрузчик соединяет воедино все программные модули, выполняя согласование их адресов;

4. Препроцессор (или макропроцессор) – транслятор, исходным языком которого является расширение языка высокого уровня, а объектным кодом – программа на языке высокого уровня (ЯВУ).

***Основные положения*:**

1) в большинстве современных языков программирования применяются директивы компиляции, обрабатываемые препроцессором;

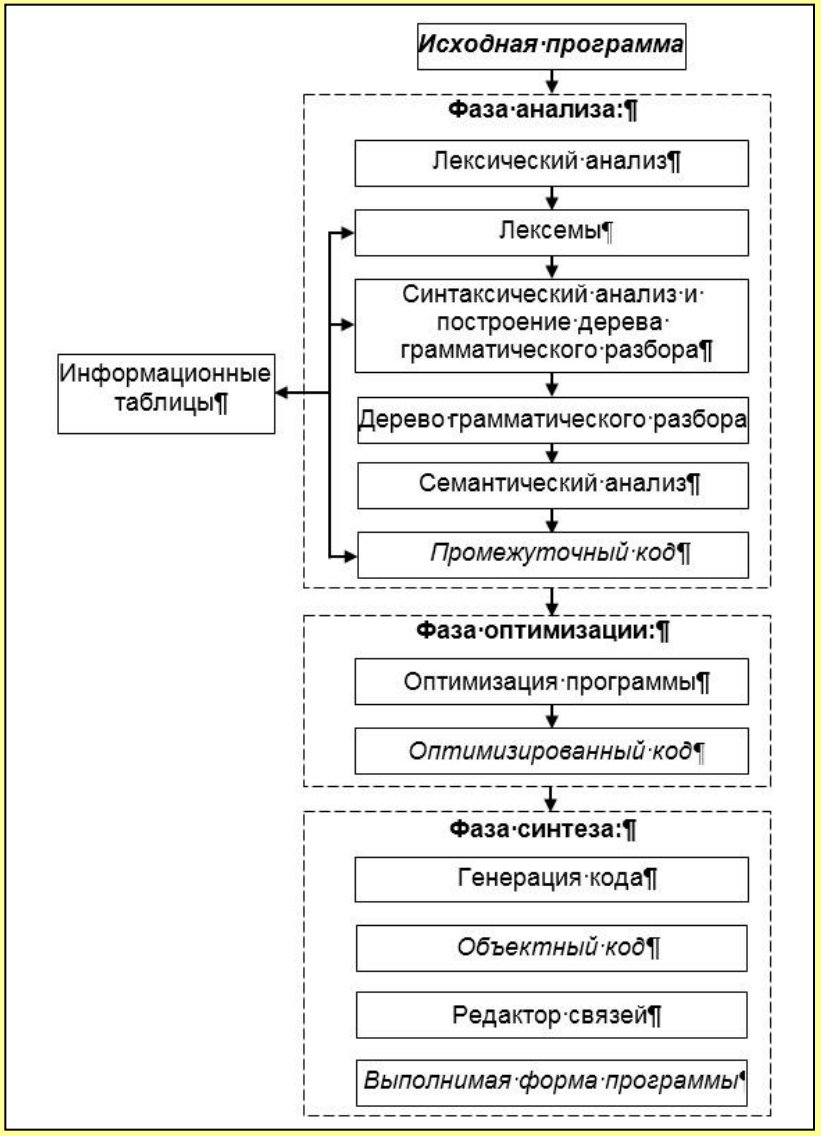
2) для трансляции программы в одной ОС с целью выполнения программы на ЯВУ в другой ОС существуют *кросс-компиляторы*: позволяют получать код для еще только разрабатываемой платформы;

Дизассемблирование – процесс преобразования кода с ассемблера на язык более высокого уровня.

3. Фазы трансляции

Для преобразования исходной программы в ее выполняемую форму (выполнимый файл) транслятор осуществляет определенную последовательность действий, которая зависит от 2-х факторов:
1) языка программирования;
2) конкретной реализации самого транслятора.

В процессе трансляции исходной программы *обязательно* выполняются следующие основные фазы (этапы) трансляции:
I. Анализ исходной программы.
II. Синтез выполнимой формы данной программы.
(Оптимизация программы – не всегда)
В зависимости от числа просмотров исходной программы, выполняемой компилятором, трансляторы разделяются на:
- Однопроходные (высокая скорость компиляции, но не самый эффективный код);
- Двухпроходные (наиб. распространение: при 1-ом проходе выполняется анализ программы и строятся инф.таблицы, которые используются при 2-ом проходе – синтезе для формирования объектного кода);
- Многопроходные (> 2-х проходов)



3.1. Синтаксис и семантика языков программирования

Для любого языка программирования (ЯП) перед разработкой транслятора должна быть определена спецификация данного ЯП – описание синтаксиса и семантики ЯП. Основное назначение синтаксиса языка программирования – формирование системы обозначений, служащей для обмена информацией между программистом и транслятором.

К синтаксическим элементам ЯП относятся (*подготовить к экзамену – примеры*):

- **набор символов (как правило 1-байтовое представление);**
- **идентификаторы (определяются символы, с которых может начинаться идентификатор, разрешенные символы в идентификаторах, длина идентификаторов);**
- **символы операций;**
- **ключевые и зарезервированные слова;**
- **необязательные слова (для облегчения читаемости программы, но при этом «утяжеляется» синтаксис языка);**
- **комментарии;**
- **пробелы (в ЯП C++ пробелы используются как разделители и их число везде игнорируется кроме литералов);**
- **разделители и скобки (синтаксический элемент, определяющий начало или конец синтаксической конструкции; разрешение неоднозначности);**
- **выражения;**
- **операторы (синтаксис операторов определяет регулярность языка и удобство записи программы)**

Семантика ЯП – совокупность правил, определяющих смысл языковых конструкций и программы в целом. (*подготовить к экзамену – примеры*)

3.2. Фаза анализа программы

Фаза анализа программы включает 3 этапа:

1. Лексический анализ;
2. Синтаксический анализ;
3. Семантический анализ.

При анализе исходной программы транслятор последовательно просматривает текст программы в виде набора символов, выполняя разбор структуры программы:

1. Лексический анализ: выделяются основные составляющие программы – лексемы.

***Лексемы* – ключевые слова, идентификаторы, символы операций, комментарии, пробелы и разделители.**

Лексический анализатор не только выделяет, но и определяет тип каждой лексемы. В итоге составляется таблица символов, в которой каждому идентификатору сопоставлен свой адрес, что позволяет при дальнейшем анализе вместо конкретного значения (строки символов) использовать его адрес в таблице символов.

Процесс выделения лексем использует сложные контекстно-зависимые алгоритмы.

3.2. Фаза анализа программы

2. Синтаксический анализ – разбор полученных лексем с целью получения семантически понятных *синтаксических единиц*, которые затем обрабатываются семантическим анализатором.

***Синтаксические единицы* – выражения, объявления, операторы ЯП, вызовы функций и/или процедур.**

3. Семантический анализ – обработка синтаксических единиц и создание промежуточного кода.

В зависимости от наличия или отсутствия Фазы оптимизации результатом семантического анализа является оптимизируемый далее промежуточный код или готовый объектный код.

Для взаимодействия между синтаксическим и семантическим анализаторами может использоваться *стек*: синтаксический анализатор заносит в стек элементы синтаксической структуры, а семантический анализатор извлекает эти элементы и обрабатывает.

Основные задачи, решаемые семантическим анализатором:

- **обнаружение ошибок времени компиляции;**
- **заполнение таблицы символов, созданной на этапе лексического анализа, конкретными значениями, определяющими дополнительную информацию о каждом элементе таблицы;**
- **замена макросов (некоторый предварительно определенный код) их определениями;**
- **выполнение директив времени компиляции (позволяет управлять процессом трансляции).**

3.3. Фаза синтеза программы

Фаза синтеза программы включает 2 этапа:

1. Генерация кода
2. Редактирование связей

1. Генерация кода – преобразование промежуточного кода (или оптимизированного кода) в объектный код. В зависимости от конкретного ЯП получаемый объектный код может быть представлен в выполнимой форме или как объектный модуль, подлежащий дальнейшей обработке редактором связей.

3. Редактирование связей – приведение в соответствие адреса фрагментов кода, расположенных в отдельных объектных модулях: определяются адреса вызываемых внешних функций, адреса внешних переменных, адреса функций и методов каждого модуля. Для редактирования адресов редактор связей использует специальные, создаваемые на этапе трансляции, таблицы загрузчика.

После обработки объектных модулей редактором связей генерируется *выполнимая форма программы*.

4. Выполнимые файлы

- **Изучить самостоятельно:**
  стр. 20-23 (Баженова И.Ю. Языки программирования)