АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Трансляция и компоновка программы

1. Решение задач на ЭВМ
2. Трансляция программы
3. Понятие связывания

1

Решение задач на ЭВМ

Программа = алгоритм + структуры данных

Ключевая задача программирования — создание и использование алгоритмов и структур данных.

ПРОЦЕСС ВЫПОЛНЕНИЯ ПРОГРАММЫ

Поток команд (поток управления)

Последовательность выполнения операций (команд), в которых содержится информация об операндах (данных).

Поток данных

Данные как результат выполнения действий над исходными данными и как источник данных (операнд) для последующих действий.

Этапы решения задач на ЭВМ

Преобразование текста программы

Компиляция — перевод текста программы с исходного формального языка на другой с последующим выполнением полученной программы в виде отдельного шага.

Интерпретация — непосредственное выполнение инструкций (команд, операторов) формального языка.

При компиляции фазы преобразования и выполнения действий разнесены во времени. Каждая из них выполняется над всеми объектами программы одновременно.

При интерпретации преобразование и выполнение действий объединены во времени, но для каждого объекта программы.

*Язык Си является **«чистым компилятором».** Результат трансляции — «чистый» программный код, в который транслятор не включает никаких «лишних» команд.*

Преобразование текста программы

Интерпретатор непосредственно выполняет действия, связанные с определением или преобразованием объектов программы.

Компилятор переводит текст программы на другой (не обязательно машинный язык).

Кросс-компилятор записывает исходный код на своем выходном языке который может быть машинным языком для компьютера с другой архитектурой.

Кросс-системы программирования используются при разработке программ для архитектур, не имеющих собственных операционных систем или систем программирования (например, контроллеры, управляющие микропроцессоры).

Процессор и память любого компьютера являются интерпретатором машинного кода.

Этапы создания программы

Практика построения трансляторов

Пример 1 Универсальный внутренний язык (P-код)
Обеспечение совместимости и переносимости трансляторов на компьютеры с различной архитектурой или с различными операционными системами.
Для каждой архитектуры создается свой интерпретатор P-кода.
Все имеющиеся компиляторы с языков высокого уровня на P-код могут использоваться без каких-либо изменений.

Пример 2 Язык программирования Java
Обеспечение переносимости приложений в среде Интернет.
Исходный текст Java-программы компилируется в байт-код.
Java-машина является интерпретатором байт-кода.
Наличие в любом браузере JVM позволяет передавать по сети и выполнять Java-программу, независимо от архитектуры компьютера и операционной системы.

Байт-код *— это двоичное представление команд виртуального процессора (виртуальной Java-машины или JVM).*

«Чистый» программный код языка Си

Транслятор языка Си — это компилятор, генерирующий программный код целевого процессора.

Транслятор не включает в код никаких дополнительных команд и обращений к внешним функциям, кроме явно прописанных в программе.

Обрабатываемые данные имеют прямое представление в памяти без дополнений или изменений.

Гарантированные свойства программы

Программист контролирует эффективность полученного программного кода.

Программист контролирует размерность и размещение данных в памяти.

Программный код может выполняться без поддержки какой-либо операционной среды на «голой» машине.

2

Трансляция программы

Трансляция программы

Подготовка программы. Редактирование файла, содержащего текст программы, со стандартным расширением для данного языка.

Фазы трансляции (компиляции)

Препроцессор

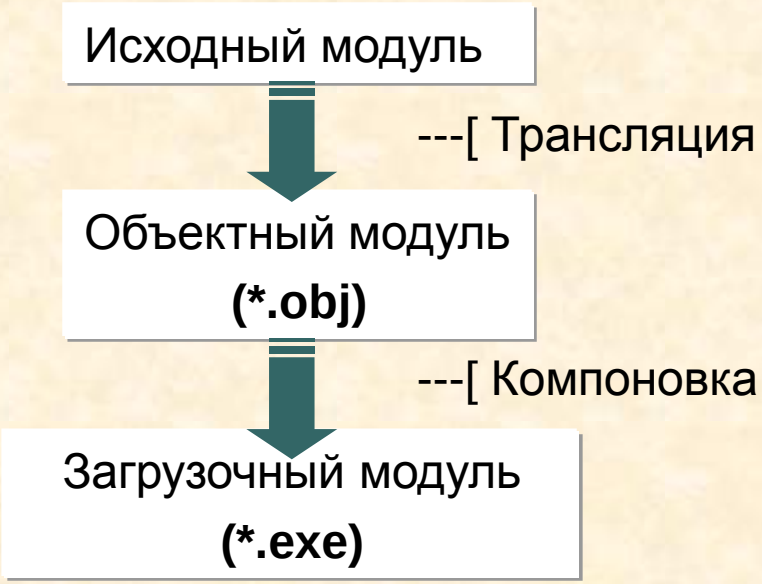
Лексический анализ

Синтаксический анализ

Семантический анализ

Генерация кода

Оптимизация кода



Компоновка (сборка) программы. Объединение одного или нескольких объектных модулей программы и объектных модулей, взятых из библиотечных файлов.

Фазы трансляции

Препроцессор. Предварительная фаза трансляции, которая выполняет замену одних частей текста программы на другие, не вдаваясь в ее содержание.

В языке Си директивы препроцессора оформлены отдельными строками программы, которые начинаются с символа «#».

Лексический анализ разбивает текст программы на лексемы.

Лексика языка программирования — это правила правописания «слов»-лексем программы, таких как идентификаторы, константы, служебные слова, комментарии.

Синтаксический анализ проверяет правильность написания конструкций языка в программе.

Синтаксис языка программирования — это правила составления «предложений» языка из отдельных «слов», например, таких как операции, операторы, определения функций и переменных.

Фазы трансляции

Семантический анализ — это проверка смысловой правильности языковых конструкций.

Семантика языка программирования - это смысл, который закладывается в каждую конструкцию языка.

В компиляторах **синтез** состоит в генерации кода.

В интерпретаторах **синтез** состоит в непосредственном исполнении (интерпретации) полученного внутреннего представления.

Фаза синтеза зависит от способа трансляции.

Генерация кода — это преобразование элементарных действий, полученных в результате лексического, синтаксического и семантического анализа программы, в некоторое внутреннее представление.

В процессе генерации кода производится и его оптимизация.

Ошибки компиляции

Предупреждения компилятора

Ошибки компилятора

Ошибки компоновщика

Предупреждения компилятора не должны останавливать работу программы.

Ошибки — это условия, которые препятствуют завершению компиляции.

Ошибки компилятора всегда включают номер строки, в которой была обнаружена ошибка.

Ошибки компоновщика — это, как правило, проблемы с поиском определения функций, структур, классов или глобальных переменных, которые были объявлены, но не определены в исходном коде.

Сообщения компилятора

Тип сообщения — предупреждение или ошибка

Исходный файл, в котором появилась ошибка

Строка ошибки

Краткое описание того, что работает неправильно

Руководящий принцип вычисления ошибок компилятора:
если сомневаетесь, посмотрите, что было в программе раньше.

Если вы имеете дело со странными проблемами памяти или трудно диагностируемыми ошибками сегментации, — используйте Valgrind на Linux или Purify для Windows.

3

Понятие связывания

Понятие связывания
Пример

Связывание — процесс установления соответствия между объектами и их свойствами в программе на формальном языке (операции, операторы, данные) и элементами архитектуры компьютера (команды, адреса).

```
int a, b;
…
a + b
```

Базовые типы данных языка Си полностью совпадают с соответствующими формами представления данных в компьютере.

Тип переменных **int** связывается с аналогичной формой представления данных в компьютере

Конкретная размерность переменной **int** определяется при реализации соответствующего компилятора.

Понятие связывания
Пример

Переменная определяется в конструкции вида: #define a 0x11FF.

В этом случае псевдо-переменная связывается со своим значением в препроцессоре.

Переменная определяется обычным способом в виде int a.

Связывание переменной с соответствующим ей типом происходит во время трансляции на фазе семантического анализа.

Переменная определяется как внешняя (вне тела функции).

Связывание заключается в распределении памяти под переменную в сегменте данных программы, который создается для текущего модуля.

Понятие связывания
Пример. Этапы распределения памяти

При трансляции переменная привязывается к некоторому относительному адресу в сегменте данных объектного модуля.

При компоновке (связывании) сегменты данных и команд различных объектных модулей объединяются в общий программный файл — образ памяти программы.

При загрузке программы в некоторую область памяти она может размещаться не с самого начала этой области. В этом случае осуществляется привязка адресов переменных, заданных в относительных адресах от начала программного модуля, к адресам памяти с учетом перемещения программного модуля.

Программа работает не в физической, а в виртуальной памяти.

Программный модуль условно считается загруженным в некоторое виртуальное адресное пространство.

Понятие связывания
Пример

Переменная определяется как автоматическая (локальная внутри тела функции), и размещается в стеке программы.

Во время трансляции определяется ее размерность и генерируются команды, которые резервируют под нее память в стеке в момент входа в тело функции.

В процессе трансляции переменная связывается только с относительным адресом в стеке программы.

Связывание локальной переменной с ее адресом в сегменте стека осуществляется при выполнении в момент входа в тело функции.

Тип операции «+» в конкретном выражении **a + b** определяется при трансляции в зависимости от типов операндов.

Инициализация внешних переменных — связывание переменных с их значениями в процессе трансляции программы (**int a = 100;**)