Виды текстовых редакторов. Схемы работы трансляторов. Смешанная стратегия трансляции

Редакторы текстов

В соответствии со схемой классической системы программирования можно выделить три начальных элемента процесса создания программы:

- редактор текста исходной программы;
- подсистема автоматизированного проектирования;
- редактор графических форм ведения диалога

Редакторы текстов

Текстовый редактор является наиболее частым начальным элементом процесса создания программы. Он позволяет готовить и вносить изменения в тексты исходных программ, однако в современных системах программирования его функции стали еще более широкими.

Текстовые редакторы стали основой интегрированных сред разработки. Известным примером текстового редактора является редактор vi, входящий в состав стандартной системы программирования операционной системы UNIX.

Редакторы называются текстовыми, поскольку они предназначены для редактирования и хранения в архиве любой текстовой информации – от текстов программ до документации по вычислительной системе.

Редакторы текстов

Текстовые редакторы делятся на две категории по видам запуска: **пакетные и диалоговые**.

Пакетные редакторы не требуют непосредственного присутствия программиста для своей работы.

Они получают на вход исправляемый текст и пакетное задание на редактирование, в котором указано, какие фрагменты текста надо из текста исключить, какие переставить местами, какие фрагменты следует заменить другой информацией, которая также включена в пакетное задание.

Редакторы текстов

Указания могут даваться в терминах номеров строк (заменить строку 15 на текст “…”) или с помощью контекста (перед строкой, в которой найден идентификатор “int”, вставить строку с идентификатором “long”).

Пакетные редакторы особенно удобны при пакетном формировании нескольких версий одних и тех же программ, отличающихся некоторыми важными параметрами, которые должны быть учтены непосредственно в тексте программы.

Запуск пакетного редактора может осуществляться из командной строки или с помощью командного файла, но в любом случае файл с заданием на редактирование должен быть подготовлен заранее.

Редакторы текстов

Диалоговые редакторы отличаются от пакетных редакторов тем, что для них готовить задание на редактирование не требуется.

Пользователь указывает редактору, какой текст он собирается редактировать, далее непосредственно вводятся редактирующие приказы.

В результате формируется исправленный текст, который можно снова записать в системный архив, использовав прежнее имя файла или задав новое имя, сохранив предыдущий вариант в архиве.

Диалоговые редакторы делятся на две категории: **строчные и экранные редакторы**.

Редакторы текстов

При работе со **строчными редакторами** сначала задают номер строки, которая подлежит редактированию, а затем выдают редактирующие приказы, которые могут влиять только на заданную строку.

Экранные диалоговые редакторы позволяют видеть на экране сразу несколько строк.

Экранные редакторы – это самое удобное средство редактирования файлов, а **строчные** обычно применяются в условиях, когда устройство отображения информации не позволяет одновременно показывать сразу несколько строк текста.

Редакторы текстов

Появление интегрированной среды разработки позволило интегрировать в них и текстовые редакторы, точнее **диалоговые экранные редакторы текстов**.

Редакторы тестов стали теснее взаимодействовать с компиляторами, а затем и с отладчиками программ.

Теперь при вводе текстов программ с помощью редактора, эта программа, получив сведения о том, на каком языке программирования написан вводимый текст, выделяет ключевые слова языка особым шрифтом и цветом, "подсвечивает« соответствующие открывающие и закрывающие скобки.

Тем самым, текстовым редакторам передаются некоторые **функции лексических и синтаксических анализаторов**.

Редакторы текстов

Некоторые системы программирования имеют особую структуру, предполагающую ввод исходной информации в виде графических объектов с помощью графического пользовательского интерфейса.

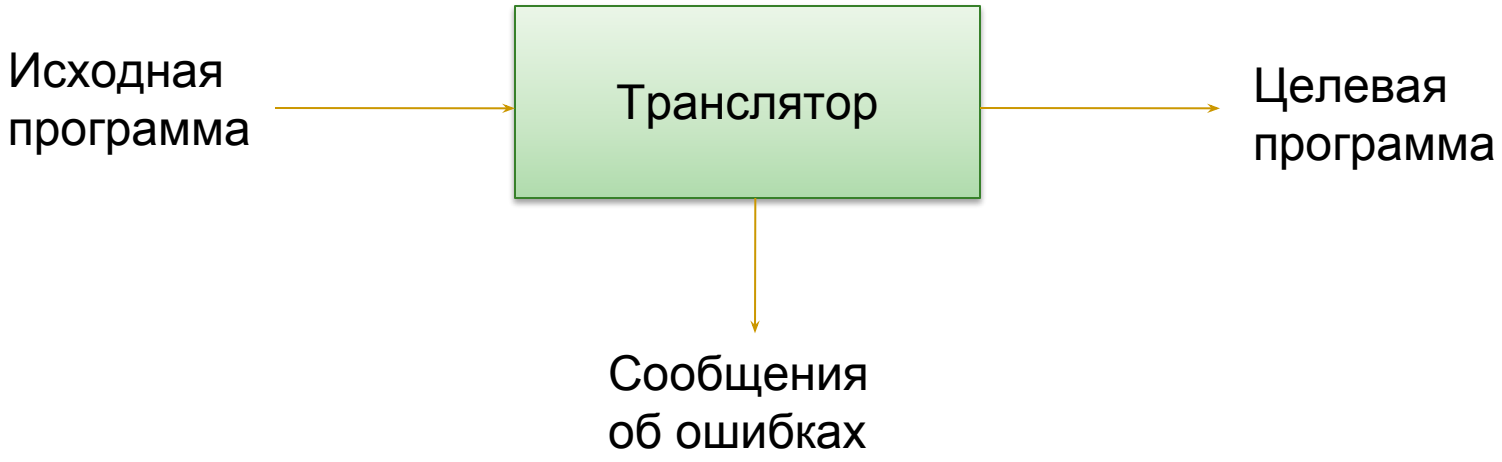
К таким системам относятся, например, системы, работающие с языками UML, Visual Basic.

Графические образы могут впоследствии автоматически преобразовываться в обычные текстовые программы.

Трансляция программы

Трансляция – перевод программы с языка высокого уровня на язык машинных кодов.

Транслятор – это программа, которая переводит программу на исходном (входном) языке в **эквивалентную** ей программу на результирующем (выходном) языке.

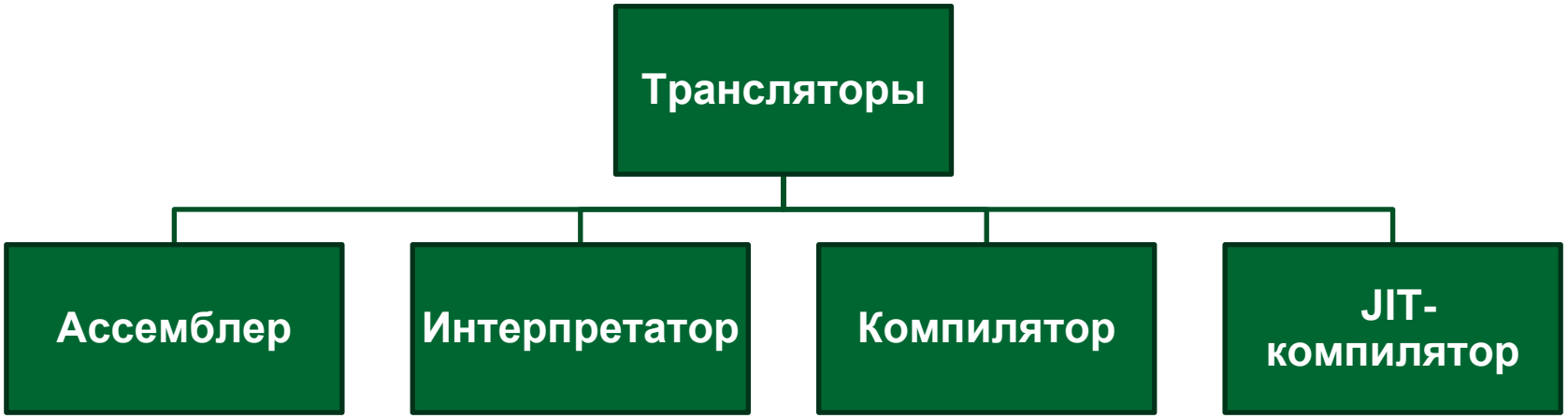


Трансляция программы

Входные данные – программа на исходном языке программирования

Выходные данные – программа на результирующем языке, или сообщение об ошибках

Эквивалентность – совпадение смысла исходной и результирующей программ с точки зрения семантики входного и семантики выходного языка



Ассемблер

Ассемблер - системная обслуживающая программа, которая преобразует символические конструкции в команды машинного языка.

- Осуществляют дословную трансляцию одной символической команды в одну машинную
- Предназначен для облегчения восприятия системы команд компьютера и ускорения программирования в этой системе команд
- Содержит средства для управления ресурсами ЭВМ
- Специфичен для конкретной архитектуры ЭВМ и ОС

Интерпретатор

Интерпретатор — это программа, которая воспринимает исходную программу на входном языке и (сразу же) выполняет ее.

- Не формирует объектный код целой программы
- Трансляция при каждом запуске
- Быстрый старт (не нужно ждать обработку всей программы)
- Медленное выполнение
- Меньше возможностей для оптимизации
- Высокая переносимость между аппаратными платформами
- Необходимо иметь интерпретатор на машине

`PHP, Python, JavaScript, Perl, etc`

Интерпретатор

Компилятор

Компилятор генерирует результирующую программу, предназначенную для непосредственного исполнения на целевой вычислительной системе:
исходная программа – исходный код
результирующая программа – объектная программа (объектный код)

- Создает программу на машинном языке
- Результат – самостоятельная программа (после компоновки)
- Однократные затраты на трансляцию
- Ожидание может быть долгим (большие проекты, **ночные сборки**)
- Тяжело несанкционированно добраться до алгоритма
- Требуется перекомпиляция под каждую платформу

`C, C++, Pascal, Ada, Modula, etc`

Компилятор


Исходная программа
Компилятор
Объектная программа
программа № 2, транслируемая
программа № 1, транслирующая
программа № 3, результирующая
Разрыв во времени и в пространстве
Входные данные
Выполнение
Результат

Смешанная стратегия трансляции

- Иногда интерпретатор сначала производит преобразование исходной программы в некоторое внутреннее представление, которое затем программно интерпретируется.
- Такой подход называется **смешанной стратегией трансляции**, это наиболее часто возникающая на практике ситуация.
- Как и языки ассемблеров, язык внутреннего представления программ в интерпретаторах разрабатывается в таком виде, чтобы на второй фазе (фазе интерпретации) легко его расшифровывать и тратить минимум времени на анализ каждого отдельного предложения внутреннего языка при его выполнении.

Смешанная стратегия трансляции

- Некоторые интерпретаторы построены так, что исполняют исходную программу последовательно, по мере поступления программы на вход интерпретатора.
- Пользователю при этом не надо ждать завершения интерпретации, чтобы увидеть первые результаты работы программы, он может получать результаты постепенно, по мере работы интерпретатора.
- Не все языки программирования допускают такое построение интерпретаторов.

Смешанная стратегия трансляции

- **Для того, чтобы это было возможно, язык должен одновременно допускать возможность существования однопроходного компилятора для этого языка. Это ограничение приводит к таким свойствам языков программирования, как необходимость описаний объектов данных до их первого использования в программе**.
- Например, не могут интерпретироваться таким способом программы на языках программирования, если эти языки допускают использование некоторых объектов прежде, чем в тексте встретится описание этих объектов.

JIT-компилятор

- Абстрактный машинный язык (промежуточный код)
- Высокая переносимость между аппаратными платформами

Алгоритм работы большинства JIT-компиляторов:

1. Компиляция в байт-код виртуальной машины среды исполнения
2. Компиляция байт-кода в машинный код

`Java, .NET, Python (PyPy)`

JIT-компилятор

Если в качестве объектного языка используется промежуточный язык, то возможны два варианта построения транслятора:

- **Первый вариант** – для промежуточного языка имеется (или разрабатывается) другой транслятор с промежуточного языка на машинный, и он используется в качестве последнего блока проектируемого транслятора.
- **Второй вариант** построения транслятора с использованием промежуточного языка – построить интерпретатор команд промежуточного языка и использовать его в качестве последнего блока транслятора.
- Преимущество интерпретаторов проявляется в отладочных и диалоговых трансляторах, обеспечивающих работу пользователя в диалоговом режиме, вплоть до внесений изменений в программу без ее повторной

Этапы трансляции

- Препроцессинг
  - Преобразование исходного текста программы без анализа
  - Выполняется препроцессором (C/C++) или компилятором (C#)
  - **Препроцессор** — это компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы.
  - Директивы – команды препроцессора (`#if #ifdef #define`)
    - Включение файлов в текст программы
    - Определение макросов (текстовой подстановки)
    - Задание параметров условной компиляции

На входе – исходные файлы

На выходе – «единицы трансляции»

Примеры

Включение:

```
#include <iostream>
```

Макрос:

```
#define MAX(a, b) (a) > (b) ? (a) : (b)
```

Условная компиляция:

```
#define DEBUG
#ifdef DEBUG
std::cout << “It is debug mode\n”;
#endif
```

Этапы трансляции

- Компиляция
  - Преобразование единицы трансляции в машинные команды за несколько этапов
  - Независимая обработка отдельных исходных модулей программы

На входе – «единицы трансляции»

На выходе – машинный код (объектные модули)

Этапы трансляции

- Линковка (компоновка, связывание)
  - Формирование единого адресного пространства
  - Размещение всех объектных модулей по соответствующим адресам
  - Изменение относительных адресов функций и переменных каждого объектного модуля на абсолютные

На входе – объектные модули, библиотеки

На выходе – исполняемый файл (или библиотеки)

Этапы трансляции

- Когда выполняется программа на C++, она выполняется последовательно, начиная с метода main(). Когда встречается вызов функции, точка выполнения переходит к началу вызываемой функции. Как ЦП узнает, куда необходимо переходить?
- Когда программа компилируется, компилятор преобразует каждый оператор в программе на C++ в одну или несколько строк машинного языка. Каждой строке машинного языка присваивается собственный уникальный последовательный адрес.
- В случае с функциями нет никаких отличий - когда функция встречается, она преобразуется в машинный язык и снабжается следующим доступным адресом.

Этапы трансляции

- когда компилятор (или компоновщик) встречает вызов функции, он заменяет вызов функции инструкцией машинного языка, которая указывает ЦП перейти к адресу функции.