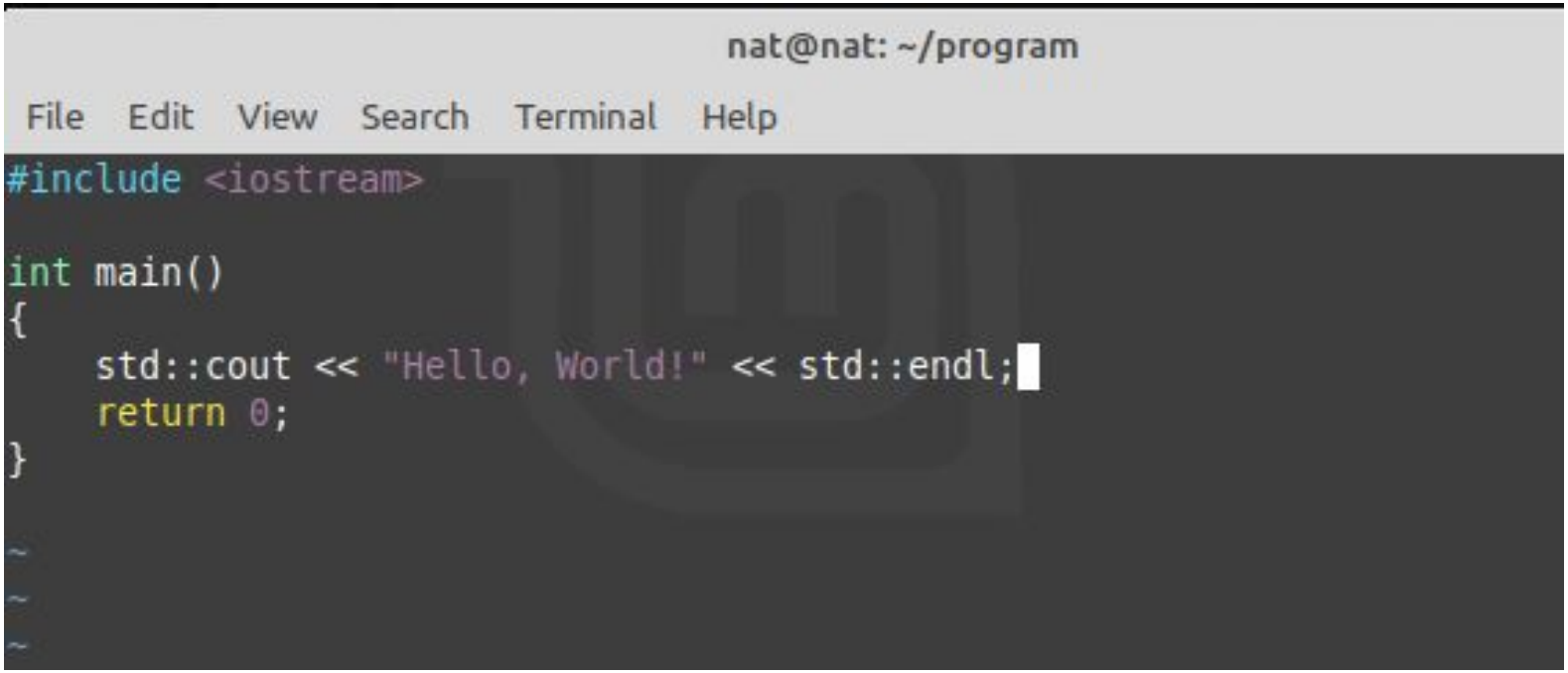
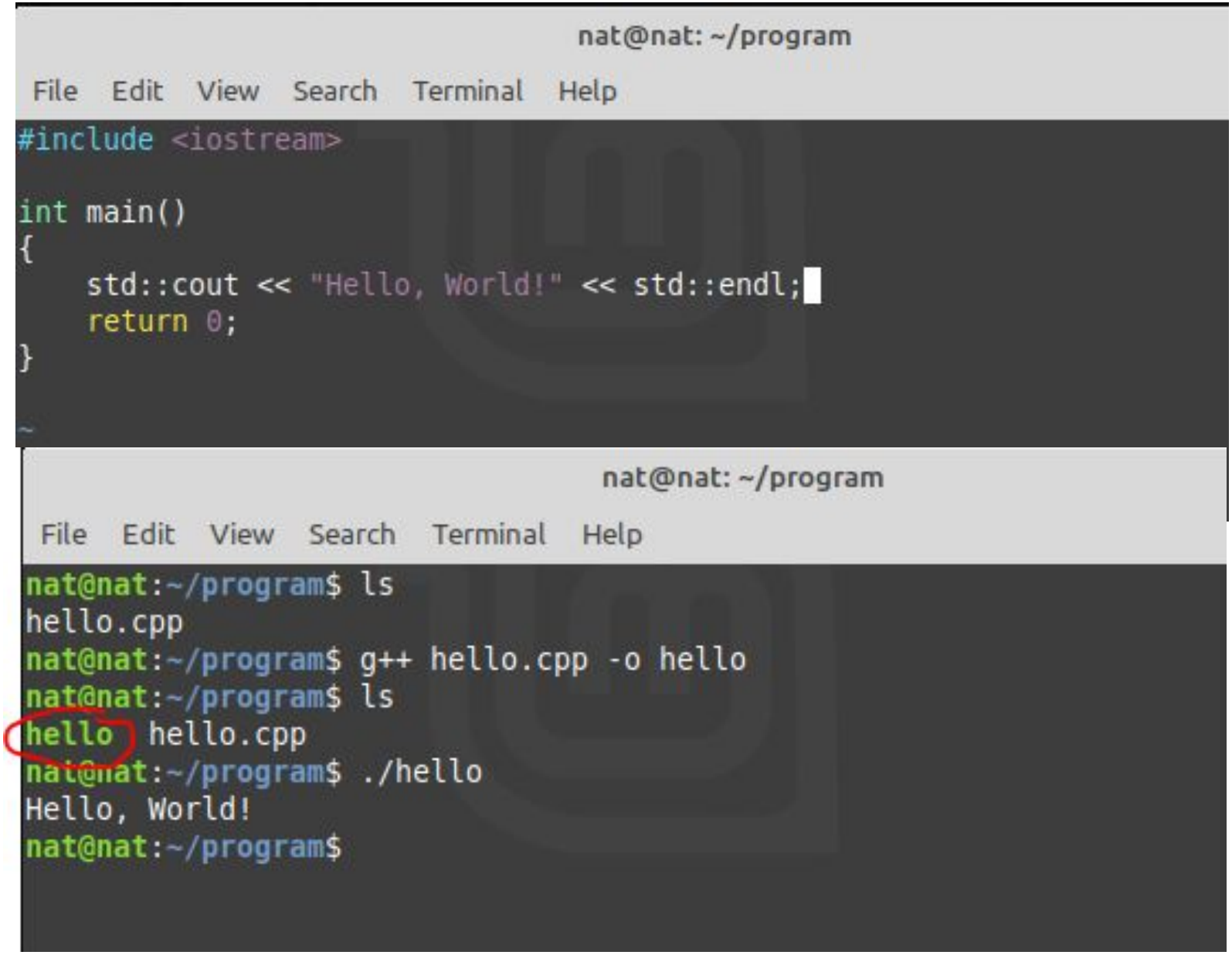
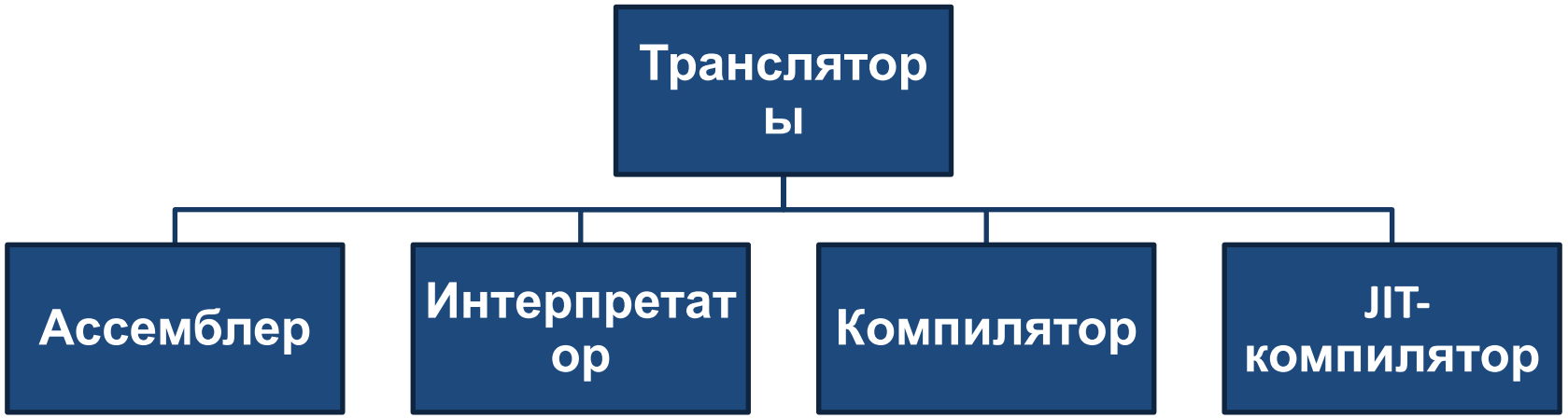
Лекция 3. Трансляция программы


nat@nat: ~/program
File Edit View Search Terminal Help
#include <iostream>
int main()
{
std::cout << "Hello, World!" << std::endl;
return 0;
}
~
~
~


nat@nat: ~/program
File Edit View Search Terminal Help
#include <iostream>
int main()
{
std::cout << "Hello, World!" << std::endl;
return 0;
}
~
nat@nat: ~/program
File Edit View Search Terminal Help
nat@nat:~/program$ ls
hello.cpp
nat@nat:~/program$ g++ hello.cpp -o hello
nat@nat:~/program$ ls
hello hello.cpp
nat@nat:~/program$ ./hello
Hello, World!
nat@nat:~/program$

Трансляция программы

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



Ассемблер

- Вырожденный транслятор. Переводит практически слово в слово
- Содержит средства для управления ресурсами ЭВМ
- Специфичен для конкретной архитектуры ЭВМ и ОС

Интерпретатор

- 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`

Компилятор

- Создает программу на машинном языке (объектный код)
- 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

Разновидность компиляторов

- Кросс-компилятор – создает код для ЭВМ, отличной от той, на которой работает компилятор
- По числу проходов (однопроходные и многопроходные)

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)

Этапы трансляции

- Препроцессинг
  - Преобразование исходного текста программы без анализа
  - Выполняется препроцессором (C/C++) или компилятором (C#)
  - 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)

Человек vs Компилятор

```
void printArray(vector<int> &v)
{
    // comment
    for (auto item : v)
    {
    std::cout << item << " ";
    }
    std::cout << std::endl;
}
```

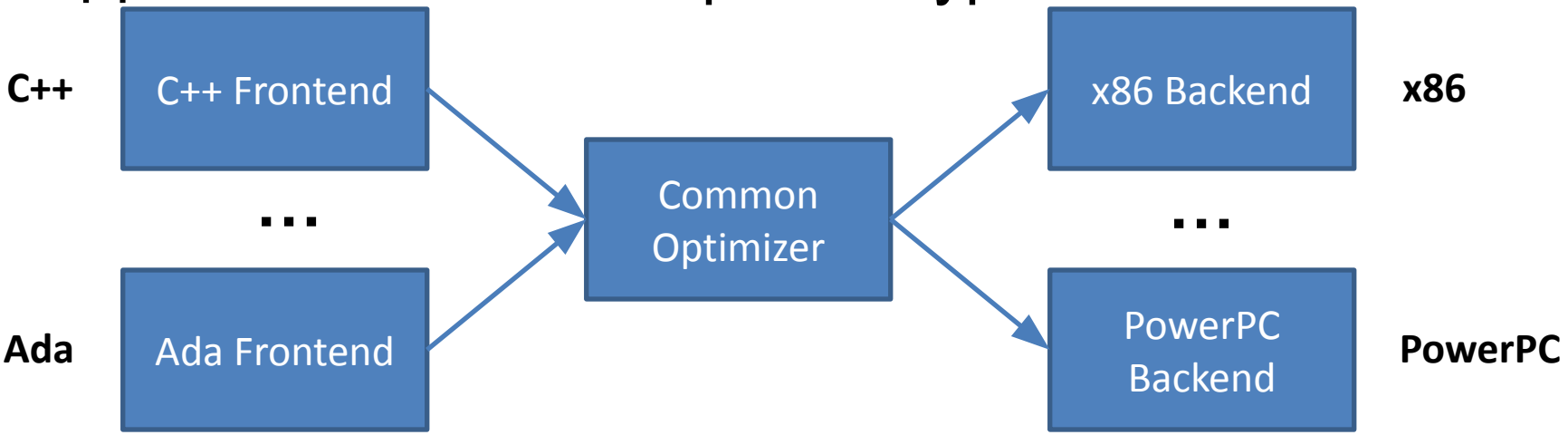
```
void printArray ( vector < int > & v ) { for ( auto
item : v )  { std :: cout << item << " " ; } std ::
cout << std :: endl ; }
```

Структура компилятора

- Frontend
  - парсинг исходного кода
  - синтаксический и семантический анализ
  - построение синтаксического дерева
- Oprimizer
  - преобразование представления с целью устранения избыточных действий (архитекутра не учитывается)
- Backend
  - преобразование оптимизированного кода в машинное представление

Структура компилятора

- Возможность поддержки нескольких языков и нескольких платформ
- Добавление нового языка только новый Frontend
- Добавление новой архитектуры – новый Backend



Этапы компиляции

- Лексический анализ
- Синтаксический анализ
- Семантический анализ

Абстрактное синтаксическое дерево

- Генерация промежуточного кода
- Генерация машинного кода
- Профит

Генерация кода

- Конвертирование синтаксически корректной программы в последовательность исполняемых инструкций
- Может быть два этапа:
  - генерация промежуточного кода
  - генерация кода для целевой архитектуры

Генерация кода

- Генерация промежуточного кода:
  - код для абстрактной машины (часто трехадресной)
  - идеализированный ассемблерный язык
  - бесконечное количество регистров
- Генерация кода для целевой архитектуры:
  - выбор инструкций с учетом системы команд процессора
  - назначение регистров в качестве операндов для инструкций

Инструменты для разработки

Must have:

- Препроцессор
- Текстовый редактор
- Библиотеки
- Компилятор
- Линковщик

По желанию:

- Отладчик
- Профилировщик
- Статический анализатор
- etc

Удобно:

- IDE
  - Редактор кода
  - Компилятор + линкер
  - Отладчик

Online IDE:

- IDEOne
- JDoodle
- etc