Лекция 2. Среда разработки и компиляция Компиляторы (g++, clang, MSVC), настройка проекта, этапы сборки. препроцессор, компиляция, линковка.

Среда разработки и компиляция в C++

---

1. Введение

Для написания программ на C++ требуется не только знание синтаксиса языка, но и понимание того, **как работает процесс компиляции** и **какие инструменты используются для разработки**.

C++ — это **компилируемый язык**, то есть исходный код (текст программы) должен быть преобразован в **машинный код**, понятный компьютеру.
Этим занимается специальная программа — **компилятор**.

Чтобы упростить этот процесс, программисты используют **интегрированные среды разработки (IDE)**, которые объединяют редактор кода, отладчик, средства компиляции и сборки проектов.

---

2. Среда разработки C++

2.1. Что такое IDE

IDE (Integrated Development Environment) — это интегрированная среда разработки, которая объединяет в себе:

- редактор исходного кода;
- инструменты компиляции и сборки;
- отладчик (debugger);
- средства для управления проектами;
- средства для тестирования и профилирования кода.

Использование IDE значительно упрощает процесс написания, проверки и запуска программ.

2.2. Популярные IDE для C++

Среда	Описание	Особенности
Microsoft Visual Studio	Мощная IDE от Microsoft для Windows	Поддержка MSVC-компилятора, отладка, GUI, интеграция с Git
Code Blocks	Бесплатная и лёгкая IDE	Поддержка GCC, Clang, простота настройки

Среда	Описание	Особенности
CLion (JetBrains)	Профессиональная IDE с умным автодополнением	Использует CMake, кроссплатформенная
Qt Creator	Среда для разработки с графическим интерфейсом	Подходит для приложений с GUI
Dev-C++	Простая и компактная среда для обучения	Использует MinGW (GCC для Windows)
VS Code	Лёгкий редактор с поддержкой плагинов	Может использовать GCC, Clang, MSVC

---

3. Компиляторы C++

Компилятор — это программа, которая переводит исходный код на C++ в машинный код (исполняемый файл).

3.1. Основные компиляторы:

Компилятор	Платформа	Особенности
g++ (GNU Compiler Collection)	Linux, Windows (через MinGW), macOS	Бесплатный, поддерживает стандарты C++11–C++23
Clang (LLVM)	Кроссплатформенный	Быстрая компиляция, улучшенные сообщения об ошибках
MSVC (Microsoft Visual C++)	Windows	Интеграция с Visual Studio, оптимизация под Windows
Intel C++ Compiler	Windows, Linux	Оптимизация под процессоры Intel
Embarcadero C++ Builder	Windows	Разработка GUI-приложений под Windows

---

4. Создание и настройка проекта

4.1. Проект в C++

Проект — это набор файлов исходного кода, библиотек и настроек, необходимых для сборки программы.
Современные IDE (Visual Studio, CLion, Code::Blocks) создают проект автоматически и управляют зависимостями.

Проект обычно включает:

- **исходные файлы (.cpp)** — основной код программы;
- **заголовочные файлы (.h)** — объявления функций, классов, структур;
- **ресурсы** (например, изображения, конфигурации);
- **файл сборки** (CMakeLists.txt, Makefile, проект Visual Studio).

4.2. Ручная компиляция без IDE

Если IDE не используется, программу можно скомпилировать через терминал или командную строку.

Пример:

```
g++ main.cpp -o program
```

- `g++` — компилятор;
- `main.cpp` — исходный файл;
- `-o program` — имя создаваемого исполняемого файла.

Запуск программы:

```
./program
```

Компиляция нескольких файлов:

```
g++ main.cpp utils.cpp -o app
```

Компиляция с указанием стандарта:

```
g++ -std=c++17 main.cpp -o app
```

---

5. Этапы сборки программы

Процесс превращения исходного кода в исполняемую программу состоит из **четырёх этапов**:

1. **Препроцессирование (Preprocessing)**
2. **Компиляция (Compilation)**
3. **Ассемблирование (Assembly)**
4. **Компоновка / Линковка (Linking)**

5.1. Этап 1. Препроцессор

Препроцессор — это программа, выполняющая предварительную обработку исходного кода **до** компиляции.

Она обрабатывает директивы, начинающиеся с `#`, например:

```
#include <iostream>
#define PI 3.14
```

На этом этапе:

- вставляются заголовочные файлы (`#include`);
- заменяются макросы (`#define`);
- исключаются/включаются части кода через условную компиляцию (`#ifdef`, `#endif`).

Результат — единый «расширенный» исходный код без директив `#`.

5.2. Этап 2. Компиляция

Компилятор преобразует предварительно обработанный текст программы в **объектный код** (машинные инструкции, но ещё не исполнимые).

Результат: создаётся файл с расширением `.obj` (Windows) или `.o` (Linux).

Пример команды:

```
g++ -c main.cpp
```

Результат: `main.o`

На этом этапе проверяются:

- синтаксис и типы данных;
- объявления переменных и функций;
- корректность выражений.

Ошибки этого этапа называются **ошибками компиляции** (compile-time errors).

5.3. Этап 3. Ассемблирование

На этом шаге код на языке ассемблера (промежуточное представление) переводится в машинный код.

Этот процесс обычно встроен в компилятор и происходит незаметно для программиста.

5.4. Этап 4. Линковка (Linking)

На этапе линковки все объектные файлы (`.o`) и библиотеки объединяются в **единый исполняемый файл** (`.exe` или без расширения в Linux).

Если программа использует внешние функции (например, из `iostream`), линковщик должен найти их реализации в стандартной библиотеке.

Ошибки на этом этапе — **ошибки линковки** (linker errors), например:

```
undefined reference to 'main'
undefined reference to 'cout'
```

---

6. Виды ошибок при компиляции

Тип ошибки	Где возникает	Пример
Синтаксическая	На этапе компиляции	Пропущена `;` или неправильная структура выражения
Логическая	Во время выполнения	Программа работает, но результат неверен
Ошибка линковки	На этапе сборки	Отсутствует реализация функции
Ошибка времени выполнения	При запуске	Деление на 0, выход за границы массива

---

7. Пример полного цикла сборки

Пусть у нас два файла:

main.cpp

```
#include <iostream>
#include "mathutils.h"

int main() {
    std::cout << "Сумма: " << add(3, 4) << std::endl;
    return 0;
}
```

mathutils.h

```
int add(int a, int b);
```

mathutils.cpp

```
int add(int a, int b) {
    return a + b;
}
```

Шаги сборки вручную:

1️⃣ Препроцессинг и компиляция:

```
g++ -c main.cpp
g++ -c mathutils.cpp
```

→ создаются `main.o` и `mathutils.o`

2️⃣ Линковка:

```
g++ main.o mathutils.o -o app
```

3️⃣ Запуск:

```
./app
```

Результат:

```
Сумма: 7
```

---

8. Флаги компилятора g++

Флаг	Назначение
`-o file`	Указать имя выходного файла
`-c`	Компилировать без линковки
`-Wall`	Включить предупреждения
`-std=c++17`	Указать стандарт C++
`-g`	Добавить отладочную информацию
`-O2`, `-O3`	Оптимизация производительности
`-I path`	Добавить путь к заголовочным файлам
`-L path`	Добавить путь к библиотекам

---

9. Отладка и запуск программы

После успешной компиляции можно выполнять:

- **отладку (debugging)** — поиск ошибок логики;
- **трассировку (tracing)** — пошаговое выполнение кода;
- **анализ памяти** — проверка утечек (`valgrind`, `AddressSanitizer`).

IDE обычно включает встроенный **отладчик** (например, GDB в Code::Blocks или Visual Studio Debugger).

---

10. Заключение

Компиляция — это сердце разработки на C++.
Понимание этапов сборки помогает:

- находить и исправлять ошибки;
- управлять зависимостями и библиотеками;
- оптимизировать программу под конкретные платформы.

Современные IDE делают этот процесс удобным, но хороший программист должен понимать, **что происходит "под капотом"** — от препроцессора до линковки.

---

11. Вопросы для самопроверки

1. Что такое компилятор и для чего он нужен?
2. Какие IDE и компиляторы наиболее популярны для C++?
3. Опишите четыре этапа сборки программы.
4. Что делает препроцессор и какие директивы он обрабатывает?
5. Чем отличается ошибка компиляции от ошибки линковки?
6. Как скомпилировать программу вручную с помощью `g++`?
7. Какие флаги компилятора вы знаете и для чего они нужны?