# Экстремальный Си.

## Глава 2. Компиляция и компоновка.

В данной главе мы пошагово разберем весь процесс превращения исходного кода на языке C в готовый продукт компиляции. Это очень глубокая тема, и потому я разбил ее на пять разделов.

1. *Стандартный процесс компиляции в C*. Как происходит компиляция в C, из каких этапов она состоит и какова их роль в создании готового продукта;
2. *Препроцессор*. Здесь мы во всех подробностях обсудим препроцессор, который обрабатывает код перед компиляцией;
3. *Компилятор*. Здесь будет рассказано, как компиляторы создают *промежуточное представление* исходного кода и затем транслируют его в язык ассемблера;
4. *Ассемблеры*. Вслед за компиляторами перейдем к ассемблерам, которые играют важную роль в трансляции ассемблерных инструкций, полученных из компилятора, в машинные коды;
5. *Компоновщик* – механизм сборки, который создаёт готовые продукты из проектов C.

*Заголовочный файл* обычно содержит перечисления, макросы и определения типов, а также *объявления* функций, глобальных переменных и структур.

#### К заголовочным файлам можно подключать только другие заголовки, но не исходники. К исходным файлам можно подключать только заголовки. Подключение одних заголовочных файлов к другим считается дурным тоном. Если вы так делаете, то это обычно говорит о серьезной проблеме в архитектуре вашего проекта.

|  |  |
| --- | --- |
| **double** | **average(int\*, int);** |
| Возвращаемый тип | Сигнатура – имя функции со списком входных параметров |

## Сборка проекта на языке Си

### Этап 1. Предобработка

На данном этапе исходному файлу подключается ряд заголовков. Но перед компиляцией препроцессор собирает их содержимое в единый блок кода. Также должны быть выполнены и другие *директивы препроцессора*. Код, прошедший такую обработку, называется *единицей трансляции* (или *единицей компиляции*). Единица трансляции — это отдельный логический блок кода на C, сгенерированный препроцессором и готовый к компиляции.

gcc **-E** example2\_1.c

### Этап 2. Компиляция и ассемблерный код

После получения единицы трансляции можно переходить ко второму этапу — *компиляции*. На вход подается единица компиляции, полученная на предыдущем этапе, а на выходе получается соответствующий *ассемблерный код*. Он все еще может быть прочитан человеком, но уже зависит от аппаратной архитектуры и приближен к оборудованию.

gcc **-S** example2\_1.c  
cat example2\_1.s

### Этап 3. Компиляция в машинные инструкции

Цель — сгенерировать инструкции машинного уровня (или машинный код) на основе ассемблерного кода, созданного компилятором на предыдущем этапе. У каждой архитектуры есть свой ассемблер, который может преобразовать собственный ассемблерный код в машинные инструкции. Файл с машинными инструкциями, который мы сгенерируем в этом подразделе, называется *объектным файлом*.

**as** example2\_1.s **-o** example2\_1.o

Параметр **-c** объединяет три предыдущих шага:

gcc **-c** example2\_1.c

### Этап 4. Компоновка

#### **Синтаксический анализатор** — это, как правило, программа, которая разбирает входные данные и извлекает из них определенные элементы для дальнейшего анализа и обработки. Чтобы разбить входные данные на более мелкие и полезные элементы, анализатор должен понимать их структуру.

### Препроцессор

### Компилятор

### Дерево абстрактного синтаксиса

### Ассемблер

## Резюме

В этой главе вы:

* познакомились с процессом компиляции кода на языке C и его различными этапами. Мы обсудили каждый этап и узнали, что он принимает на вход и какой результат он возвращает;
* узнали определение термина «*платформа*» и то, как разные ассемблеры могут генерировать разные машинные инструкции для одной и той же программы на языке C;
* подробно изучили каждый этап и компонент;
* изучили разделение компилятора на интерфейс и кодогенератор, за счет которого GCC и LLVM поддерживают много разных языков;
* рассмотрели форматы объектных файлов и их зависимости от платформы;
* узнали, что объектные файлы зависят от платформы и должны иметь подходящий формат;
* рассмотрели принцип работы компоновщика, использование символов для поиска недостающих определений в ходе формирования конечного продукта. изучили разные виды продуктов, которые можно получить из проекта на C, и узнали, почему переносимые (или промежуточные) объектные файлы не причисляют к продуктам компиляции;
* узнали, как обмануть компоновщик, если предоставить ему неправильное определение;
* рассмотрели декорирование имен в C++ и как с его помощью можно избежать проблем.