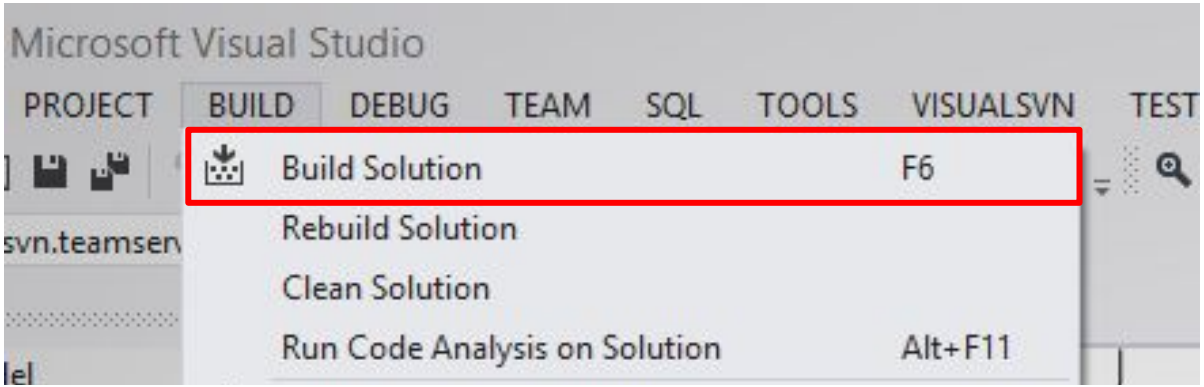
Компилятор: введение

Дмитрий Матвеев, Itseez, 2016

Содержание

Вступление

Что вы знаете о компиляторах?



Современные компиляторы C и C++

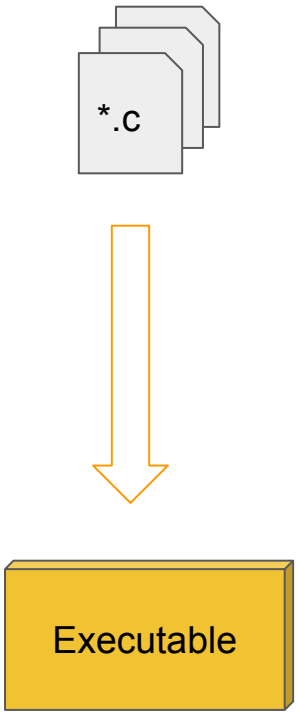
Актуальные:

1. GNU Compiler Collection: http://gcc.gnu.org/
2. LLVM clang: http://clang.llvm.org/
3. Intel C Compiler: https://software.intel.com/en-us/c-compilers
4. Microsoft Visual C++: https://www.visualstudio.com/

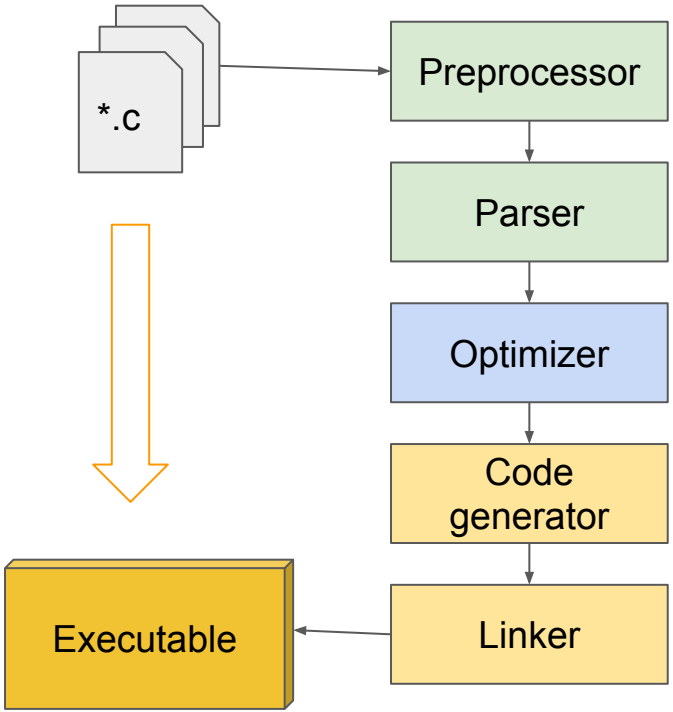
Исторические:

1. Comeau C/C++: http://www.comeaucomputing.com/
2. Watcom C++
3. Portable C Compiler (PCC)

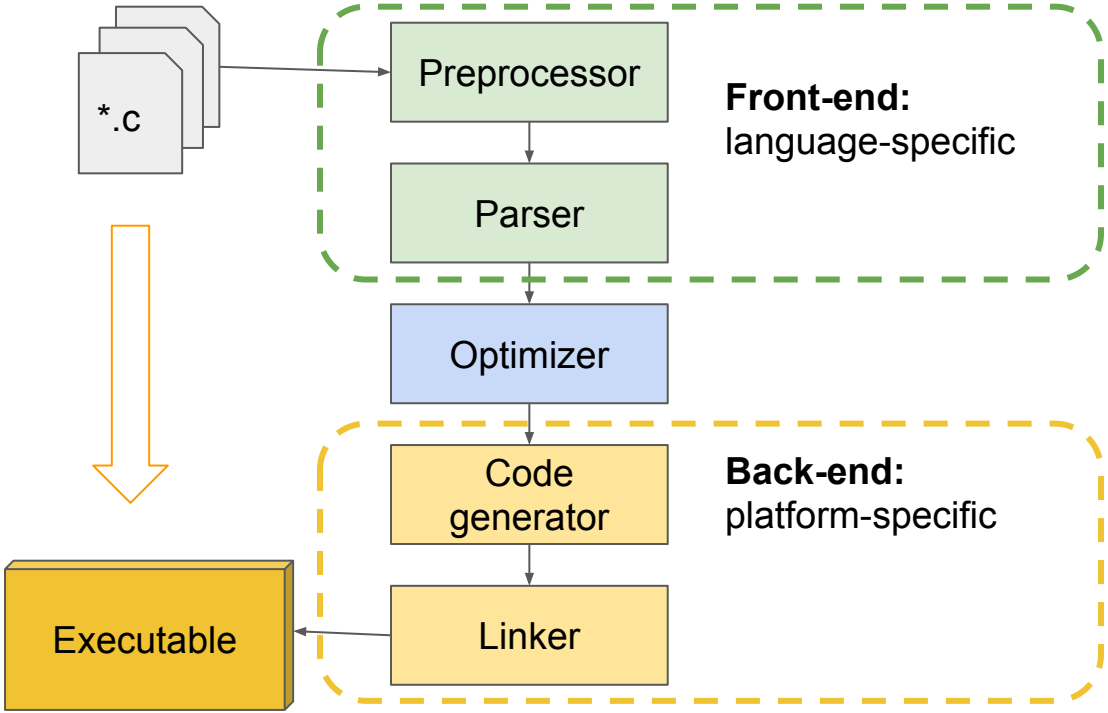
Работа компилятора



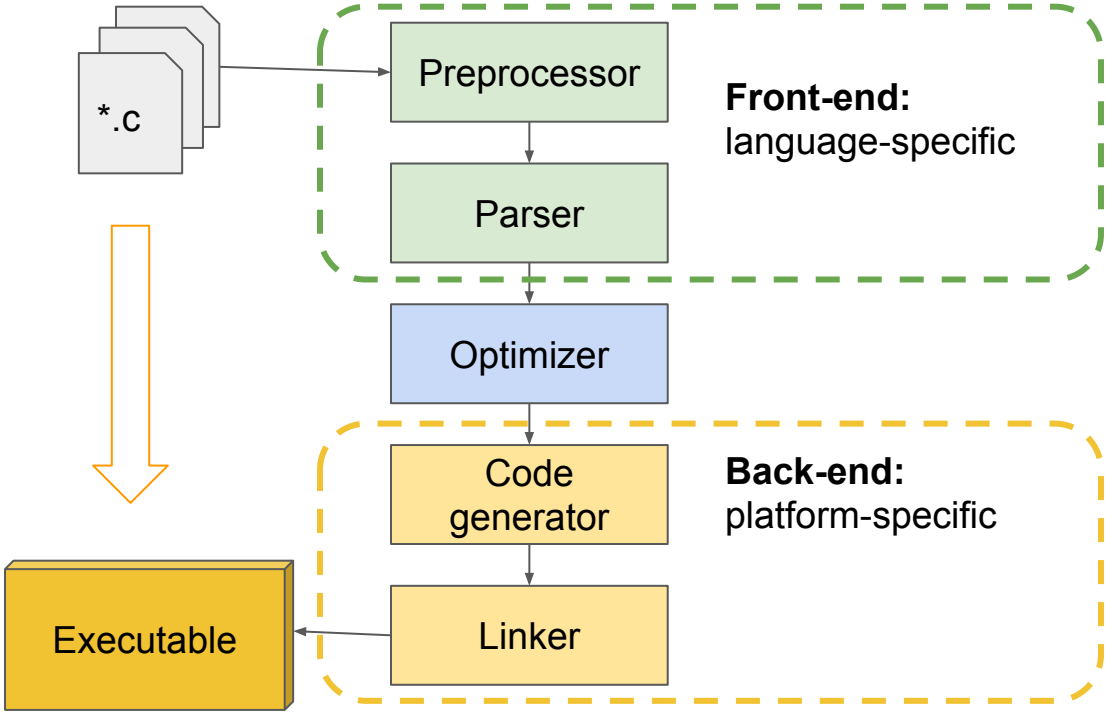
Работа компилятора



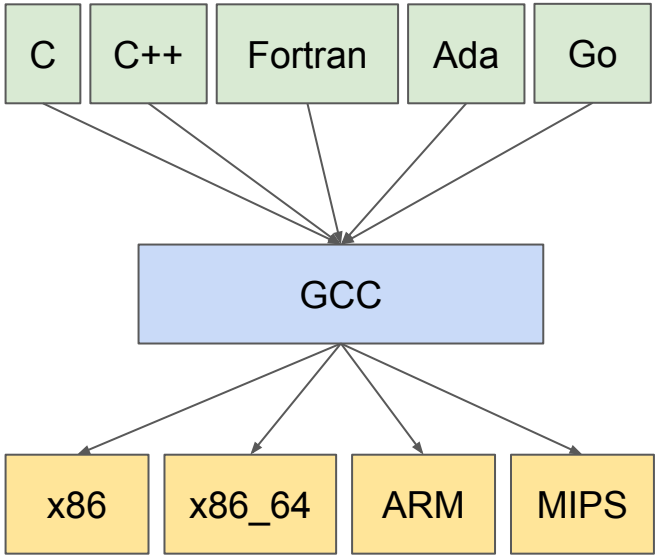
Работа компилятора



Работа компилятора



Пример: GCC



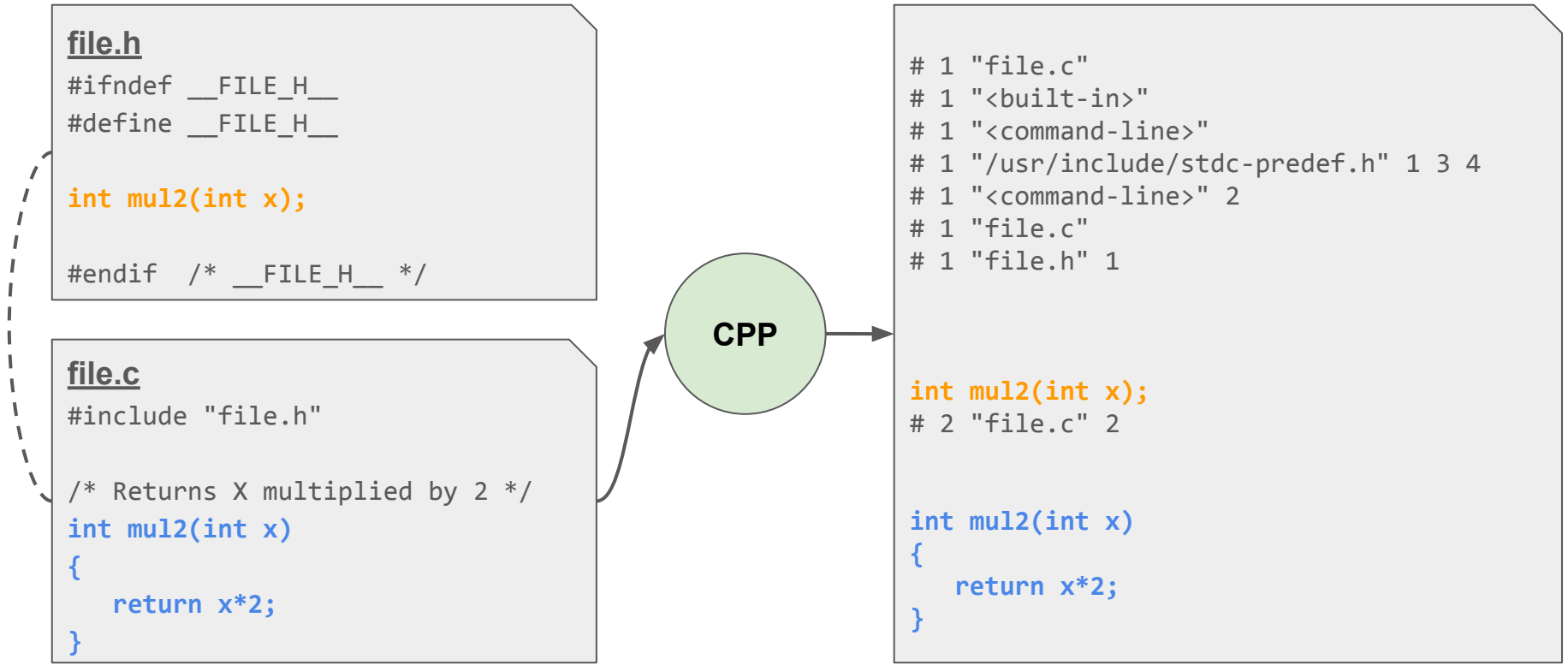
Работа компилятора: препроцессинг

Основное предназначение - создание единицы трансляции:

- Включение всех файлов через #include
- Обработка #ifdef, #define, etc.
- Удаление комментариев

Единица трансляции - минимальный блок программного кода, который физически можно оттранслировать (в т.ч. подать на вход компилятору).

Работа компилятора: препроцессинг



Работа компилятора - синтаксический разбор

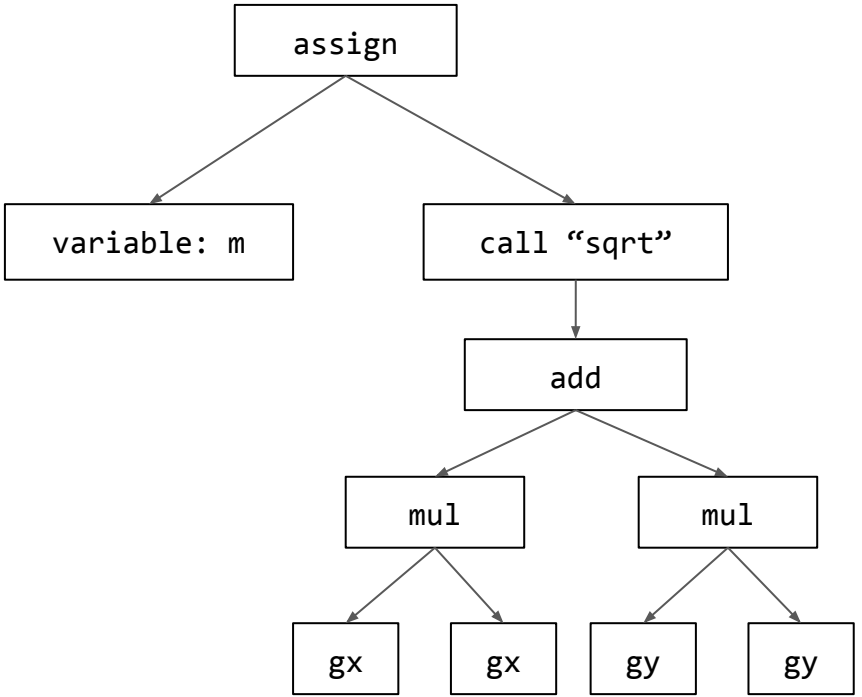
- Код любой валидной программы - не хаотичный набор символов, а текст, написанный по определенным правилам.
- Правила устанавливаются **синтаксисом** языка.
- При помощи синтаксического разбора можно получить структурное представление кода программы, удобное для дальнейшей работы.
- Естественным представлением текста на формальном языке являются деревья.
- AST - Abstract Syntax Tree:
  - Узлы - операторы
  - Листья - операнды

Работа компилятора - синтаксический разбор

```
m = sqrt(gx*gx + gy*gy);
```

Работа компилятора - синтаксический разбор

```
m = sqrt(gx*gx + gy*gy);
```

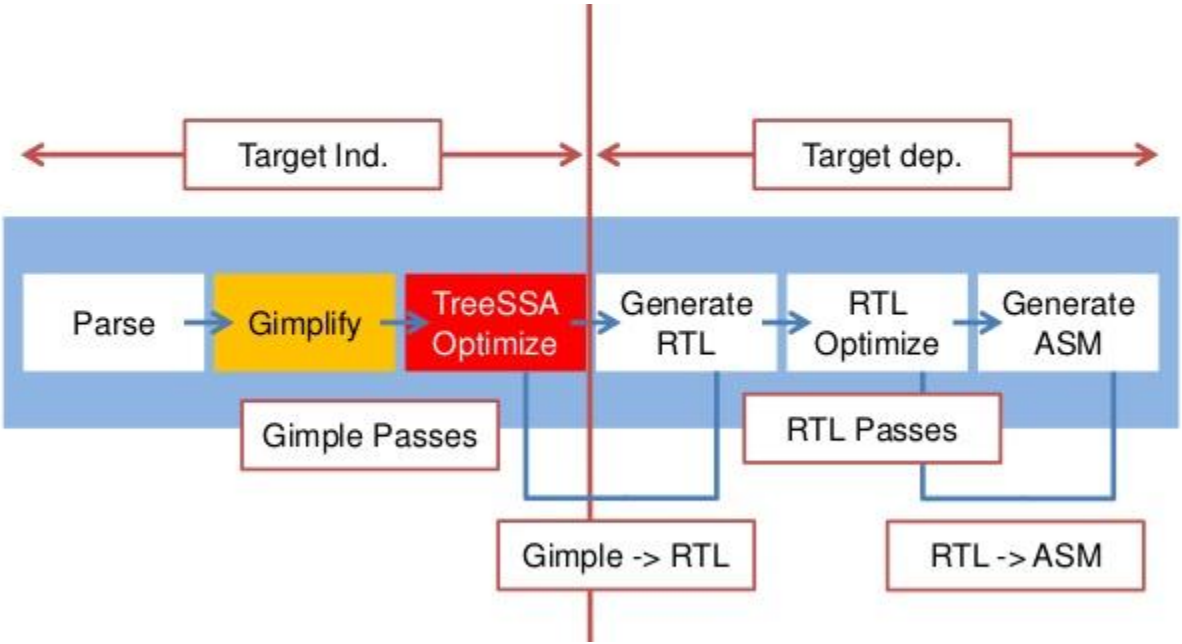


Работа компилятора - оптимизация

- Самый сложный этап, совсем-совсем!
- Происходит в несколько (очень много) проходов на уровне промежуточного представления;
- ...и на разных уровнях промежуточного представления:
  - Платформо-независимые оптимизации
  - Платформо-зависимые оптимизации
- Ваша главная задача - **не мешать** оптимизатору!

Работа компилятора - оптимизация

На примере GCC © Wei-Sheng Chou



Работа компилятора - оптимизация

Платформо-независимые оптимизации (на примере GIMPLE Tree SSA в GCC):

1. Удаление “мертвого” кода;
2. Свертка констант;
3. Избежание ненужных (невостребованных) вычислений;
4. Оптимизация циклов, векторизация;
5. Оптимизация хвостовых вызовов;
6. Оптимизация передачи возвращаемых значением;
7. Как бонус: определение возможных проблем программиста (пример - использование переменных без инициализации)
8. ...И многое другое!

Полный список: https://gcc.gnu.org/onlinedocs/gccint/Tree-SSA-passes.html

Работа компилятора - оптимизация

Платформо-зависимые оптимизации (на примере GCC RTL passes):

1. Оптимизации переходов;
2. Комбинирование инструкций на основе потока данных;
3. Планирование и переупорядочивание команд;
4. Выделение регистров (либо перемещение вычислений на стек);

Полный список: https://gcc.gnu.org/onlinedocs/gccint/RTL-passes.html

Работа компилятора - генерация кода

- Компилятору доступно machine description - описание целевой платформы (какие машинные инструкции может использовать компилятор в генерируемом коде);
- Код генерируется путем сопоставления инструкций промежуточного представления с образцами из machine description и выбора соответствующих машинных команд.
- Опционально в выходной файл включается отладочная информация.

Работа компилятора - линковка

- Объединение нескольких объектных файлов (продуктов компиляции) в результирующий исполняемый файл (или библиотеку)
- Установка связей с внешними библиотеками.

При линковке все составные части приложения объединяются в одно целое, что открывает пути для дополнительных оптимизаций: те фрагменты программы, что были недоступны при обработке отдельной единицы трансляции, становятся известны компилятору.

- LTO - **L**ink **T**ime **O**ptimization
- WHOPR - **WHO**le **P**rogram optimize**R**

Использование GCC

- GCC сегодня - самый распространенный компилятор в мире Open Source
- GCC предоставляет разработчку массу опций для контроля и тонкой настройки процесса компиляции/генерации кода
- Знание и умение использовать GCC **руками (**вне IDE) - полезно и нужно!
- Обычно некоторые этапы компиляции (препроцессинг, линковка) выполняются отдельными приложениями (cpp, ld), но gcc позволяет это скрыть.

Использование GCC: основы

Сборка программы из одного файла:

```
gcc file.c -o myprogram
```

Сборка программы из нескольких файлов:

```
gcc file.c app.c -o myprogram
```

Сборка объектных файлов и линковка:

```
gcc -c file.c
gcc -c app.c
gcc file.o app.o -o myprogram
```

Использование GCC: базовые ключи

Указание директорий поиска заголовочных файлов: `-I`

```
gcc file.c -Iinclude -I/path/to/library/include -o myprogram
```

Установка директив: **`-D`**

```
gcc -c file.c -DENABLE_LOGGING -DBUFFER_SIZE=1024
```

Так же возможна установка этих (и других) ключей в переменной окружения `CFLAGS (CXXFLAGS).`

Использование GCC: базовые ключи

Указание директорий поиска подключаемых библиотек: `-L`

Указание подключаемых библиотек: `-l`

```
gcc file.c -L/path/to/mylib -lmylib -lm -lpthread -o myprogram
```

Так же возможна установка этих (и других) ключей в переменной окружения `LDFLAGS`.

Использование GCC: отладка и профилирование

- `-Wall` - включить все предупреждения о возможных ошибках
- `-Werror` - сделать все предупреждения об ошибках ошибками компиляции
- `-g`, `-ggdb` - включение отладочной информации
- `-p`, `-pg` - включение информации для профилирования (замедляет работу приложения!)

Дополнительно: https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html

Использование GCC: платформа

`-march=`*`cpu-type`* - генерация кода под конкретный набор инструкций, где `cpu-type` может быть:

- native - если приложение оптимизируется для именно вашего компьютера
- i386, i486, i586, …, haswell, broadwell, skylake

По-умолчанию код генерируется для максимальной переносимости (без использования современных расширений, следовательно, не очень быстрый).

Использование GCC: оптимизации

- `-O0` - компилятор старается уменьшить затраты на компиляцию и сделать отладку предсказуемой (режим по-умолчанию);

- `-O`/`-O1`, `-O2`, `-O3` - включение уровней оптимизации, на каждом из которых задействуются дополнительные оптимизации;
- `-Os` - оптимизация кода по размеру (примерно равносильно `-O2`);
- `-Ofast` - оптимизация кода с некоторыми отступлениями от стандарта.

Использование GCC: оптимизации

Опасность! Некоторые используемые оптимизации могут:

- Навредить точности в угоду скорости, если программный код тесно завязан на детали спецификаций IEEE/ISO математических функций;
- Вскрыть проблемы с некорректно написанным кодом;
- Пример: `-ffast-math`

Заключение