**СЕМИНАР 1. Среда создания программ на С++. Компилятор g++**

1. **Среды программирования**

*Среды программирования* (или как их еще называют, среды разработки) - это программы, в которых программисты пишут свои программы. Иными словами, среда программирования служит для разработки (написания) программ и обычно ориентируется на конкретный язык или несколько языков программирования. *Интегрированная* среда программирования (IDE (англ. Integrated development environment)) содержит в себе все необходимое для разработки программ:

* *редактор*. В нем программист пишет текст программы, так называемый программный код;
* *компилятор/интерпритатор*. Компилятор, транслирует программу, написанную на высокоуровневом языке программирования в машинный язык (машинный код), непосредственно понятный компьютеру. Язык С++ относится к компилируемым языкам, поэтому для обработки текстов его программ служит компилятор, иногда вместо компилятора (либо вместе с ним) используется интерпретатор, для программ, написанных на интерпретируемых языках программирования;
* *сборщик*. Автоматизирует процесс сборки кода.
* *отладчик*. Служит для отладки программ. Ошибки в программах допускают абсолютно все: и новички, и профессионалы - они могут быть синтаксическими (обычно они выявляются еще на стадии компиляции) и логическими. Для тестирования программы и выявления в ней логических ошибок служит отладчик.

Выше описана базовая комплектация среды программирования, но иногда в ней присутствуют еще и такие компоненты, как система управления версиями, различные инструменты для конструирования графического интерфейса программы, браузер классов, инспектор объектов и другие.

1. **Этапы реализации программы**

Рассмотрим процесс разработки программы в среде программирования, от момента начала написания кода программы до получения скомпилированного файла, который уже можно непосредственно запускать вне среды разработки. Как правило, для того, чтобы выполнить программу на С++, надо пройти шесть этапов:

1. Редактирование;
2. Предварительная (препроцессорная) обработка;
3. Компиляция;
4. Компоновка;
5. Загрузка;
6. Выполнение.

Рассмотрим пример реализации программы на языке С++, ориентированной на UNIX.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№№** | **Наименование**  **этапа** | **Где исполняется** | **Основные действия** | **Результат**  **выполнения** |
| **1** | Редактирование | Редактор | Набор текста программы, внесение изменений и исправлений | Файл .c/.cpp |
| **2** | Предварительная (предпроцессорная)  обработка | Предпроцессор | Выполнение преобразований, указанных директивами (#) – включение других текстовых файлов в файл, выполнение замен. | Файл .i |
| **3** | Компиляция | Компилятор | Проверка текста программы на синтаксические и логические ошибки. Если все хорошо – преобразование текста программы в машинный (объектный) код. Созданный код содержит дыры (отсутствуют части кода библиотечных функций, подключаемых файлов) | Файл .obj |
| **4** | Компоновка | Компоновщик | Связывание объектного кода с кодами библиотечных функций, функций, находящихся в других файлах и т.д. | Файл  .exe (Windows)  .out (Lunux) |
| **5** | Загрузка | Загрузчик | Отправка программы в оперативную память компьютера | - |
| **6** | Выполнение | Процессор | Пошаговое выполнение команд программы | Зависит от специфики программы |

**Редактирование.** Это первый этап разработки программы в среде программирования и представляет он собой редактирование файла (исходного файла, который в последствии будет содержать код программы). Он выполняется с помощью редактора программ. Программист набирает в этом редакторе свою программу на С++ и, если это необходимо, вносит в нее различные изменения или исправления. Одним словом, работает с кодом программы как с обычным текстом. Имена файлов программ на С++ часто оканчиваются расширением .сс или .срр.

**Предварительная (препроцессорная) обработка.** На этом этапе программист дает команду компилировать программу. Но прежде чем компилятор приступит к компиляции вашей программы, производится предварительная обработка программы. Любая программа в С++ подчиняется специальным командам, именуемым директивами препроцессора (обычно начинаются они со специального символа "#") , которые указывают, что в программе перед ее компиляцией нужно выполнить определенные преобразования. Обычно эти преобразования состоят во включении других текстовых файлов в файл, подлежащий компиляции, и выполнении различных текстовых замен. Создается файл с расширением .i .

**Компиляция.** На этом этапе компилятором проверяется текст программы на наличие синтаксических ошибок и затем, если все хорошо, текст программы с подстановками, сделанными на предыдущем этапе, преобразуется в машинный код (код на языке, уже непосредственно понятный компьютеру). Иногда его еще называют объектным. На этом этапе создается файл с расширением .obj. Также в программе могут использоваться кусочки уже готового машинного кода, расположенного в иных библиотеках (например, в файлах с расширением .lib). На этапе компиляции эти библиотеки еще не будут подключены к только что созданному машинному коду. Они подключаются на следующем этапе.

**Компоновка.** Следующий этап называется компоновка. Программы на С++ обычно содержат ссылки на функции, определенные где-либо вне самой программы, например, в стандартных библиотеках или в личных библиотеках групп программистов, работающих над данным проектом. Объектный код, созданный компилятором, обычно содержит «дыры» из-за этих отсутствующих частей. Компоновщик связывает объектный код с кодами отсутствующих функций, чтобы создать исполняемый загрузочный модуль (без пропущенных частей). Получаем в итоге файл с расширением .exe (для Windows), либо .out (для Linux).

**Загрузка.** Следующий этап называется загрузка. Перед выполнением программа должна быть размещена в оперативной памяти компьютера. Это делается с помощью загрузчика, который забирает исполняемый загрузочный модуль с диска) и перемещает его в оперативную память.

**Выполнение.** И наконец, рассмотрим самый последний этап - выполнение. С этого момента компьютер под управлением своего ЦПУ (центральное процессорное устройство) начинает последовательно выполнять в каждый момент времени по одной команде программы. Эти моменты времени носят название такт, каждый процессор имеет свою тактовую частоту, которую задает его внутренний тактовый генератор. Чем более высокая частота работы вашего процессора, тем, соответственно, лучше и тем быстрее выполняются ваши программы. На маленьких программах это, конечно же, не очень ощутимо, но когда запускаете какую-нибудь новомодную игрушку, то все очень даже заметно.

1. Компилятор/интерпритатор

Есть два способа выполнения программы компьютером: она может быть подвергнута *компиляции* или *интерпретации*.

*Компиля́тор*– [программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), выполняющая *компиляцию*.

*Компиляция* – [трансляция программы](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80), составленной на исходном [языке высокого уровня](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в эквивалентную программу на [низкоуровневом языке](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), близком [машинному коду](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) ([абсолютный код](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D0%BE%D0%BB%D1%8E%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), [объектный модуль](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C), иногда на [язык ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0)). Входной информацией для компилятора ([исходный код](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)) является описание алгоритма или программа на [объектно-ориентированном языке](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F&action=edit&redlink=1), а на выходе компилятора – эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).

В отличие от компилятора интерпретатор не создает никакой объектный код. В простейшем случае интерпретатор читает исходный текст программы по одной строке за раз, выполняет эту строку и только после этого переходит к следующей. В общем случае интерпретируемая программа выполняется медленнее, чем скомпилированная. Необходимо помнить, что компилятор преобразует исходный текст программы в объектный код, который выполняется компьютером непосредственно. Значит, потеря времени на компиляцию происходит лишь единожды, а в случае интерпретации — каждый раз при очередной компиляции фрагмента программы в процессе ее выполнения.

Оптимизация кода при интерпритации выполнена быть не может.

Программа, написанная на любом языке программирования, может как компилироваться, так и интерпретироваться, однако многие языки изначально созданы для выполнения преимущественно одним из этих способов. Например, Java рассчитан в основном на интерпретацию программы, а язык С — на компиляцию.

1. **Компилятор g++**

**GNU Compiler Collection** (обычно используется сокращение **GCC**) — набор [компиляторов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) для различных [языков программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

Был выпущен в 1987 году, а в 1992 году в нему была добавлена возможность компилировать программы, написанные на языке С++.

GCC является [свободным программным обеспечением](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) и используется как стандартный компилятор для свободных [UNIX-подобных операционных систем](https://ru.wikipedia.org/wiki/Unix-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0).

Изначально компилятор поддерживал только язык [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Позднее GCC был расширен для компиляции исходных кодов на таких языках программирования, как [C++](https://ru.wikipedia.org/wiki/C%2B%2B), [Objective-C](https://ru.wikipedia.org/wiki/Objective-C), [Java](https://ru.wikipedia.org/wiki/Java), [Фортран](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD) и [Ada](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).

Существует компилятор gcc под Windows, называемый **MinGW** - Minimalist GNU for Windows.

GCC может работать на микропроцессорах из более 40 разных семейств и генерировать код для них. Это является одним из аргументов критики GCC, которая заключается в том, что это компилятор слишком широкого назначения: возможность применения для столь многих микропроцессоров и языков программирования означает, что результирующий код получается неоптимальным.

Внешний интерфейс GCC является стандартом для компиляторов на платформе UNIX. Пользователь вызывает управляющую программу, которая называется gcc. Она интерпретирует аргументы командной строки, определяет и запускает для каждого входного файла свои компиляторы нужного языка, запускает, если необходимо, [ассемблер](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80) и [компоновщик](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%BE%D0%B2%D1%89%D0%B8%D0%BA).

Компилятор каждого языка является отдельной программой, которая получает [исходный текст](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) и порождает вывод на [языке ассемблера](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%B0%D1%81%D1%81%D0%B5%D0%BC%D0%B1%D0%BB%D0%B5%D1%80%D0%B0). Все компиляторы имеют общую внутреннюю структуру.

Данный компилятор является оптимизирущим то есть в GCC есть модули, оптимизирующие ПО для конкретных процессоров.

Программа отправляется на компиляцию посредством введения команды о запуске в командную строку. В зависимости от параметров, вводимых в командную строку, gcc запускает необходимые процессоры, компиляторы, линкеры и т. д.

1. Оптимизация:

Оптимизация — это процесс преобразования фрагмента кода в другой фрагмент, который функционально эквивалентен исходному, с целью улучшения одной или нескольких его характеристик, из которых наиболее важными являются скорость и размер кода.

В настоящее время логика оптимизации стала очень сложной: компиляторы преобразуют циклы, условные выражения и рекурсивные функции; удаляют целые блоки кода. Они оптимизируют код под процессорную архитектуру, чтобы сделать его действительно быстрым и компактным. И это очень здорово, ведь лучше фокусироваться на написании читабельного кода, чем заниматься ручными оптимизациями, которые будет сложно понимать и поддерживать.

Методы автоматической оптимизации можно разделить на следующие классы:

• Оптимизация, удаляющая "лишний" код. Сюда же относятся такие методы, как вычисление выражений, содержащих константы, на этапе компиляции (constant folding) и, иногда, встраивание функций.

• Оптимизация, улучшающая машинный код на локальном уровне. К этому классу относятся методы оптимизации, уменьшающие число машинных команд, заменяющие короткие последовательности команд более эффективными командами и выполняющие перестановку инструкций с целью уменьшения времени ожидания (peephole optimization).

• Оптимизация работы с памятью. Методы этой группы повышают эффективность кэширования, и организуют упреждающую предвыборку. К этому же классу следует отнести методы выравнивания (alignment) адресов меток и переменных.

• Оптимизация ветвлений и циклов. В этот класс входят методы, удаляющие избыточные условия ветвлений и оптимизирующие операции перехода. К оптимизации циклов относятся методы, разворачивающие циклы, оптимизирующие счетчики и т.п.

• Оптимизация вызова функций. Включает управление формированием стекового фрейма, передачу аргументов в регистрах, оптимизацию кода возврата из функции и встраивание функций (function inlining).

• Интеллектуальная оптимизация. Включает такие методы, как перестановка блоков (block reordering) и исключение концевой рекурсии.

Для запуска оптимизации в командной строке набирают О0, O1, O2 или O3. O0 – самая простая оптимизация

О1 – более сильная оптимизация. Не сильно замедляет работу компилятора, уменьшает размер программы

О2 – оптимизировать все, но только надежными методами – из нее исключены методы, которые ускоряют вычисления за счет увеличения размера кода

О3 – жесткая оптимизация = О2+ оптимизация времени

Каждый более высокий уровень включает в себя более низкий уровень

Os – оптимизировать по размеру

Второй уровень считается наиболее эффективным и безопасным.

С третьим уровнем зачастую возникает нестабильность работы.

1. Пример работы с gcc (применимо и для компилятора g++)

Написать программу, сохранить ее под именем myprogramm.c

Запуск программы выполняется следующим образом

gcc myprogramm.c

В каталоге появился новый файл a.out. Это и есть исполняемый файл. Для запуска необходимо набрать в консоли:

./a.out

Программа должна запуститься.

Компилятор gcc по умолчанию присваивает всем созданным исполняемым файлам имя a.out. Если хотите назвать его по-другому, нужно к команде на компиляцию добавить флаг -o и имя, которым вы хотите его назвать.

gcc myprogramm.c -o myprogramm

В каталоге появится исполняемый файл с названием myprogramm. То есть, получился точно такой же исполняемый файл, только с удобным названием.

Чтобы только скомпилировать объектный код и не пускать его на запуск используют флаг –с:

gcc –с myprogramm.c -o myprogramm

Будет только создан файл myprogramm.о

Если в дальнейшем необходимо скомпонавать несколько объектных файлов в единый исполняемый файл, следует выполнить

Gcc –o F F1.o F2.o

Если программе следует передавать аргументы, это делается следом за именем программы

./F <аргументы>

Флаг -o является лишь одним из многочисленных флагов компилятора gcc. Чтобы просмотреть все возможные флаги, можно воспользоваться справочной системой man. Наберите в командной строке:

man gcc