GeekBrains

Основы C++

Введение

Знакомство, выбор и установка инструментария, компиляция и сборка.

Системы счисления, механика языка, первая программа

На этом уроке

1. Установим инструментарий для программирования на языке C++, рассмотрим вопросы сборки и исполнения кода проекта
2. Изучим основные используемые в информатике системы счисления и напишем свою первую программу.

Оглавление

На этом уроке

Теория урока

Зачем нужен C++

- Почему нужно выбрать именно C++?
- Минимальные требования к студенту

Средства разработки на C++. Что понадобится?

- ТРАНСЛЯТОРЫ
- СРЕДЫ РАЗРАБОТКИ
- МАКРОСБОРЩИК

Системы счисления и основы информатики

- Двоичная система счисления
- Восьмеричная система счисления
- Шестнадцатеричная система счисления

Об основных понятиях и их истории

Как работает компилируемый язык

- Первая программа (program.cpp)
- Трансляция программы
  - Этапы трансляции программы
- Запуск программы
- Методы трансляции программ:
- Стандартная библиотека:

Этапы трансляции программы, сборка

- Простейший сценарий сборки CMake

Практическое задание

Дополнительные материалы

- Пара слов о последних версиях CMake (3.x)

Используемые источники

Теория урока

Трансляторы, среды разработки, макросборщик, системы счисления, основные понятия языка C++, сценарий сборки

Зачем нужен C++

- Возможность программирования на машинном и высоком уровнях одновременно, вплоть до ассемблерных вставок. Язык позволяет полностью игнорировать верхнеуровневые преобразования автоматизированных трансляторов, манипулируя непосредственно ячейками памяти
- Полный контроль за происходящим в программе и её окружении: самостоятельная сборка, отладка и оптимизация программ, в том числе, полученных от других разработчиков.
- Разработка законченных решений под любые аппаратные платформы и операционные системы.

Почему нужно выбрать именно C++?

- Детальный стандарт получения исполняемых программ под любое аппаратное обеспечение и операционную среду.
- Высокая, контролируемая производительность полученных программ, оптимизация компилятором "из коробки".
- Математическая доказательность (что логически вычислимо, то обязательно возможно запрограммировать на C++).
- Высокий спрос на программистов в технологичных отраслях (игрострой, робототехника, автопром, цифровая радиотехника)

Минимальные требования к студенту

- минимальные знания английского языка;
- отсутствие страха перед командной строкой операционной системы, умение ее вызывать;
- готовность к постоянному переключению внимания и анализу логики событий;
- последнее, но не в последнюю очередь: спокойное отношение к своим и чужим ошибкам.

Средства разработки на C++. Что понадобится?

- транслятор языка C++ (далее, следует понимать, что компилятор это тоже самое, что транслятор);
- среда разработки или текстовый редактор;
- терминал командной строки;
- средство сохранения настроек трансляции (макросборщик);
- средство контроля сгенерированного машинного кода (HEX-редактор).

ТРАНСЛЯТОРЫ

Для установки транслятора в Linux-подобных операционных системах нужно ввести строку

```
sudo apt-get install build-essential
```

Узнать версию компилятора - `g++ --version`. Нужно отметить, что компилятор gcc в Windows установить будет невозможно, и в качестве аналогов придется использовать MinGW или Clang/LLVM

Рекомендуемый компилятор - Clang (Возможность собрать программу практически для любой другой операционной системы и любой аппаратной архитектуры). Установка - `sudo apt-get install clang` (Из минусов то, что вместе с LLVM будет занимать около одного гигабайта)

Компилятор MinGW - перенос компилятора gcc на операционную систему Windows, его можно установить, скачав соответствующий пакетный файл с официального сайта разработчика.

Компилятор MSVC - в ранние годы своего существования был предназначен для компиляции своего диалекта C++, который частично отличался от оригинального C++, описанного в международном стандарте. В последние годы поддерживает стандарт и практически не насаждает использование собственных расширений, если они не нужны разработчику.

Важно учесть, что трансляторы для языков C и C++ отличаются, хоть и устанавливаются вместе, поэтому, для корректной компиляции кода на C нужно использовать clang, gcc, а для компиляции C++ кода необходимо использовать g++, clang++, c++.

Какой бы компилятор ни был выбран, проверить его наличие можно набрав в командной строке Вашей операционной системы команду `c++ --version`. При установке транслятора могут возникнуть сложности, поскольку чаще всего программы для разработки являются тесно связанным ис ОС и не всегда тривиальным программным обеспечением. Невозможно описать все возможные варианты поведения для всех возможных операционных систем, но есть самые часто встречающиеся проблемы:

- Команда clang++ (или аналогичная) не существует. Удостоверьтесь, что установили инструментарий, попробуйте скачать другой, например, MinGW, если до этого пробовали clang, и наоборот.
- Компилятор совершенно точно установлен, но терминал “не видит” его. Нужно добавить путь к установленному компилятору в системные переменные среды. Для разных ОС этот процесс отличается, но вопрос настолько часто встречающийся, что совершенно точно подойдёт первая или вторая не рекламная ссылка в Вашем любимом поисковике по запросу “НАЗВАНИЕ-ВАШЕЙ-ОС добавить системную переменную среды”
- Компилятор установлен и я смог скомпилировать программу, но после перезагрузки “всё исчезло”. Смотрите предыдущий пункт, добавив в запрос “с сохранением после перезагрузки”.

СРЕДЫ РАЗРАБОТКИ

Qt Creator - поддерживает работу с gcc, MinGW, Clang. (Рекомендуется: хорошая справочная система, хороший редактор кода)

Eclipse - отличается тем, что разработана на Java, в состоянии интегрироваться с любым компилятором, главное правильно настроить. Сложность самого процесса настройки

KDevelop - используется для очень многих дистрибутивов Linux, которые имеют в своем составе оболочку рабочего стола KDE, поэтому и KDevelop

Набирает популярность среда CodeBlocks и просто “блокноты” с набором соответствующих плагинов. Также есть платная среда разработки CLion от компании JetBrains. Все вышеперечисленные инструменты объединяет одно: автоматизация рутинных процессов и упрощение процесса написания кода.

МАКРОСБОРЩИК

Также (и всё чаще) называемый системой сборки - CMake - будет хранить между сеансом редактирования исходного текста настройки транслятора, которые будут в каждом конкретном случае для нашей программы задавать. Или будет подготавливать готовый проект по тем командам, которые будут прописаны в скрипте запуска

Системы счисления и основы информатики

Данный материал не ставит себе целью закрыть абсолютно все вопросы информатики, но авторы предполагают, что не все могут быть знакомы с понятием систем счисления, поэтому в этом разделе будет представлен необходимый для дальнейшего понимания и работы материал.

Жизнь любого человека так или иначе связана с числами. Числа - это количество съеденных за обедом бутербродов, скорость автомобиля по дороге домой, сумма, потраченная на покупки в магазине. Числа не существуют сами по себе, они связаны определёнными законами математической науки и полностью подчиняются ей. Программирование, как один из аспектов жизни, тоже полностью подчинено математике, но не такой математике, которой пугают людей, а базовой, на уровне «записать число», произвести сложение, вычитание, умножение. Запись чисел осуществляется поразрядно, то есть одно число может содержать несколько цифр, стоящих друг за другом. Разряды в записи чисел увеличивают свой вес справа налево, то есть самая правая цифра - самая маленькая. В любой позиционной системе счисления более значимый разряд всегда находится левее и увеличивается на единицу, если менее значимый разряд «переполняется». Мы можем это увидеть в привычной нам десятичной системе счисления, когда разряд единиц достигает цифры девять, на единицу увеличивается разряд десятков, а разряд единиц «сбрасывается в ноль». В современной математике принята **десятичная система счисления**, это значит, что каждый следующий (более левый) разряд имеет вес в десять раз больше, чем предыдущий (более правый). Для нас привычно называть эти разряды единицами, десятками, сотнями, тысячами и так далее. В программировании также широко используется десятичная система счисления, как наиболее привычная человеку. но кроме этого также часто применяются ещё три: двоичная, восьмеричная, шестнадцатеричная (binary, octal, hexadecimal). Принцип записи и работы этих систем счисления не отличается от десятичной, каждый более старший разряд записывается левее и означает число большее «на основание системы счисления». То есть если в десятичной системе счисления `116 = 1*100 + 1*10 + 6*1`, то в восьмеричной, например `116 = 1*64 + 1*8 + 6*1`. Так мы можем обратить внимание, что системы счисления отличаются только основанием. В двоичной это основание - два, то есть в каждом разряде может содержаться только ноль или единица, в восьмеричной это восемь, то есть от нуля до семи включительно, в десятичной - десять, то есть от нуля до девяти включительно, и в шестнадцатеричной - шестнадцать, то есть от нуля до пятнадцати включительно. Привычная нам математика оперирует числами любой величины, вплоть до бесконечности, а разрядная сетка машин конечна. Отсюда, с одной стороны, возникают переполнения. С другой стороны, этот феномен возродил интерес к конечным полям в алгебре (например, полям Галуа) которые используются в современных и очень распространённых алгоритмах шифрования.

В части базовых знаний об устройстве компьютерной техники (если не вдаваться в подробности) важно запомнить, что **минимальная единица цифровой информации называется бит** и представляет собой информацию о наличии напряжения электричества (например, на контакте

процессора). Такие биты начиная с поздних 70-х принято объединять в регистры по восемь штук и называть **байтами**. Байты в свою очередь показывают размерность тех или иных частей компьютера, например, запоминающих устройств: килобайты (1024 байта), мега-, гига-, тера-, и так далее -байты. Почему килобайт - это не одна тысяча байт, а именно 1024? Здесь снова задействованы биты. С учётом особенностей двоичных систем счисления учёные-математики и информатики (информационная наука довольно молодая и начала существовать примерно после второй мировой войны) подобрали максимально близкое к привычной (для десятичной системы счисления) тысяче число и им оказалась двойка в десятой степени, то есть 1024.

Также важно помнить о битах и байтах следующее: чтобы обращаться к конкретным байтам, компьютер должен знать их адрес в системе, а адреса - это такие же наборы байт. Таким образом, мы можем удобно обращаться к адресам, так или иначе сформированным двоичной системой счисления. Так появились 8ми-битные, 16ти-битные вычислительные процессоры, а в последствии и микроконтроллеры, то есть 8ми-битный, значит позволяющий иметь адреса в диапазоне от нулевого до 255-го включительно, каждый из которых может содержать один байт данных. Современные процессоры 64х-битные, то есть способные напрямую, без дополнительных вычислений адресовать в памяти 2^{64} = 18 446 744 073 709 551 616 байт, то есть около 16 миллионов терабайт. Довольно много.

Двоичная система счисления

Представляет особенный интерес для области информационных технологий, поскольку вся цифровая электроника работает по принципу «есть напряжение или нет напряжения», единица или ноль. Все числа во всех системах счисления в результате преобразуются в двоичную и представляются в виде набора единиц и нулей. Так для записи числа `116` используется двоичная запись `1110100`. Преобразование из системы счисления с бОльшим основанием в систему счисления с меньшим основанием производится последовательным делением исходного числа на основание системы счисления и записи остатков такого деления в младшие разряды. Например:

```
116 /2 = 58(0) /2 = 29(0) /2 = 14(1) /2 = 7(0) /2 = 3(1) /2 = 1(1) /2 = 1 < 2
```

В этом примере полученные остатки от деления записаны в скобках и можно обратить внимание на то, что они полностью повторяют запись числа 116 в зеркальном отражении. Обратное преобразование - это последовательное умножение разрядов числа на величину каждого разряда с их аккумулированием к общему результату:

```
1110100 = 0*1 + 0*2 + 1*4 + 0*8 + 1*16 + 1*32 + 1*64 = 4 + 16 + 32 + 64 = 116
```

В программах двоичный числовой литерал записывается с префиксом `0b`, то есть для нашего примера `0b1110100`. Поскольку двоичная система счисления является основной для компьютерной техники, помнить значения степеней двойки - хорошее подспорье в работе. Такая запись числа появилась в C++ не так давно, в стандарте C++14.

Восьмеричная система счисления

На данный момент практически не встречается (полностью вытеснена шестнадцатеричной)[1]. В программах восьмеричные числа записываются с префиксом `0` (ноль) и не могут содержать цифры больше семи (это ограничение проверяется на этапе трансляции программы). Правила преобразования в восьмеричную систему счисления и обратно точно такие же, но с преобразованием

[1] Бортовые программы для космического аппарата Салют- 5Б, который работал на станции Мир, были написаны именно в восьмеричной системе счисления

в двоичную всё гораздо проще: число `7` (максимальная цифра для одного разряда) в двоичном представлении имеет вид `111`. таким образом `0116` будет записано в виде `001001110`. Каждый разряд восьмеричной записи представляет собой триплет двоичных чисел (`116 = 001.001.110`). Таким образом на уровне представления в двоичном виде мы наблюдаем, что восьмеричные числа имеют меньший вес, чем десятичные: `0b1110100 > 0b1001110`, как `116 > 78`. Убедиться в том, что `0116 = 78` мы можем произведя аналогичные вычисления

```
78 /8 = 9(6) /8 = 1(1) /8 = 1 < 8
```

И обратно

```
116 = 6*1 + 1*8 + 1*64 = 6 + 8 + 64 = 78
```

Шестнадцатеричная система счисления

Представляет собой наиболее популярную для внутреннего использования систему счисления, поскольку позволяет записывать в удобной форме байты данных. Шестнадцатеричная система счисления значительно отличается от привычной десятичной тем, что один её разряд может содержать значения от нуля до пятнадцати. Для представления значений больше чем девять (максимально возможная арабская цифра), с десяти до пятнадцати, используют первые буквы латинского алфавита (A-F). То есть шестнадцатеричное число `B` равно одиннадцати. Представьте, что у вас на руке шестнадцать пальцев, а не десять, и ситуация значительно прояснится. В программах шестнадцатеричные литералы записываются с префиксом `0x`. Например, десятичное число `116` будет записано как `0x74`. Каждый разряд шестнадцатеричной записи представлен четырьмя разрядами двоичной записи (что также полностью подчиняется правилу использования степеней двойки). Так `0x74 = 0b0111_0100`, где `7 = 0111`, а `4 = 0100`. В этой записи мы видим удобство использования шестнадцатеричных чисел для записи двоичных представлений чисел. Правила преобразования из шестнадцатеричной системы счисления и из неё такие же как и для всех предыдущих - десятичное число делится на 16 и последовательно записываются остатки от этого деления, получая шестнадцатеричное число, шестнадцатеричное же число поразрядно умножается на минимальное значение для текущего разряда (16 в степени номера разряда) ($16^0 = 1$, $16^1 = 16$, $16^2 = 256$, и так далее).

Об основных понятиях и их истории

Структура и синтаксис программы на языках C/C++. По большому счету, популярных стилей программирования на языках C/C++ языке всего два, несмотря на то, что языки мультипарадигменные и активно развиваются в сторону, например, поддержки функционального программирования.

Процедурный стиль (известный гуру в области языка Герберт Шилдт в свое время назвал его "старым") ассоциируется обычно с языком C, впервые систематически описанным Брайеном В. Керниганом и Деннисом М. Ричи в 1978 году. Американский институт национальных стандартов (American National Standards Institute - ANSI) в 1983 г. учредил комитет, перед которым была поставлена цель выработать «однозначное и машинно-независимое определение языка Си», полностью сохранив при этом его стилистику. Результатом работы этого комитета явился стандарт ANSI языка C. Последняя версия этого стандарта ISO/IEC 9899:2018 "Information technology - Programming languages - C".

Объектно Ориентированный стиль Программирования (ООП) появился немного позднее процедурного стиля, в 1985 году, и связывается с публикацией Бьёрном Страуструпом (точная транскрипция дат. Bjarne Stroustrup) его книги «Язык программирования C++». Сами принципы ООП, главнейшие из которых инкапсуляция, полиморфизм и наследование были известны, конечно, и до

Бьёрна Страуструпа, но только как абстрактные понятия или понятия из других языков программирования, без применения к языку C++. В 1998 году был ратифицирован первый стандарт языка C++ (ISO/IEC 14882 «Standard for the C++ Programming Language»), на сегодня это ISO/IEC 14882:2017 "Programming languages C++".

Настоятельно рекомендуем слушателям если не приобрести, то хотя бы следить за этими стандартами в интернете, и ознакомиться с книгами Кернигана-Ритчи «Язык программирования C», 34-е переиздание которой вышло в 2017 году и Бьёрна Страуструпа «Язык программирования C++, специальное издание».

Процедурное программирование

Далее, в этом разделе, речь пойдет об основных понятиях процедурного стиля программирования. Этот стиль следует считать "классическим", может быть немного "старым", но ни в коем случае не "устаревшим".

Во первых, процедурный стиль поддерживается синтаксисом всех современных си-подобных языков: C, C++, C#.

Во вторых, по собственному опыту авторов курса, во многих современных и перспективных областях применение техники ООП избыточно и чересчур затратно. В западной литературе существует принцип **YAGNI**, который значит You Ain't Gonna Need It – вам это не понадобится! Его суть в том, чтобы реализовать только поставленные задачи и отказаться от избыточного функционала. Например, в бритву, зубную щетку, или даже пулю для стрелкового оружия сегодня вполне может быть встроен микроконтроллер, естественно, с минимальными аппаратными ресурсами. Опытному C++ программисту такого приложения может и удобнее было бы не заботиться об инициализации своих объектов, а применять конструкторы и деструкторы классов (о чем поговорим позднее). Но не следует забывать, что и конструктор и деструктор по сути своей - та же функция, и требуют для своего исполнения вполне ощутимых затрат памяти и вычислительных ресурсов, а целевая программа и так еле-еле умещается в отведенные аппаратные ресурсы. Вот тут-то опытный программист и вспоминает о "старом стиле".

И, наконец, в третьих - без освоения основ процедурного программирования просто невозможно освоение ООП, как невозможно построить второй и последующие этажи дома без фундамента, первого этажа и крыши.

На рис.1 показана структура программы на языке C++.

Функции в языке

Как видно из рисунка, язык поддерживает модульность программ, что позволяет, при необходимости и в целях экономии времени, компилировать отдельные модули программы (только те, в которые вносились изменения). Функции в языке C++ состоят из последовательности операторов. Оператор представляет собой выражение вида: `ОПЕРАНД ОПЕРАЦИЯ ОПЕРАНД ОПЕРАЦИЯ .... ОПЕРАЦИЯ ОПЕРАНД`

На этапе знакомства с языком нам необходимы только самые общие понятия о функциях в C++, далее этой теме будет посвящено отдельное занятие, где и разберем более конкретные примеры.

Ниже представлен пример оформления параметризованной функции, в функциях принято описывать действия, которые совершает программа:

```
тип_функции имя_функции(параметр_1, параметр_2, ..., параметр_n)
{
   объявление_данных
   /* Комментарий */
   оператор_1;
   оператор_2; // Комментарий
   .....
   оператор_m;
   return (val);
}
```

Параметризованной такая функция называется потому, что после имени функции, в скобках, через запятую, перечисляются типы и имена переменных `(параметр_1, параметр_2, ..., параметр_n)`, которые передаются в качестве аргументов в функцию. Запятые в этом списке являются обязательными разделителями, без них компилятор выдаст сообщение об ошибке, а многоточия ... в этом примере всего лишь показывают, что параметров в скобках после `имя_функции` может быть много (от 0 до n), как много может быть и операторов (от 0 до m). В реальной программе этих многоточий быть не должно, иначе компилятор не сможет транслировать эту программу.

Если нет необходимости передавать аргументы в функцию, то список в скобках остается пустым `()`, но сами скобки при записи `имя_функции()` обязательны. Такая функция называется не параметризованной. Пробел между `имя_функции` и скобками допускается, но не является обязательным. Также допускается размещать несколько коротких операторов на одной строке, обязательно разделяя их символом `;` но, по мнению большинства опытных разработчиков, такой стиль оформления программы не эстетичен и снижает читаемость кода.

`Тип_функции` означает тип переменной `val`, которую функция возвращает своим оператором `return (val);` если ничего возвращать из функции не требуется, оператор `return val;` используется в виде `return;` то есть, никакое значение возвращено не будет. В таком случае, тип функции необходимо обозначить специальным ключевым словом `void` (англ. - пустота). Также `return` в `void` функциях используется если предполагается выход из последовательности операторов не в конце списка, а по какому- либо условию, скажем, сразу после выполнения `оператор_1`. Можно также обратить внимание, что скобки вокруг `val` не являются обязательными.

Комментарии

Об объявлении данных поговорим в следующем разделе. А пока пару замечаний по поводу комментариев и оформления программ. Комментарий - это фрагмент текста программы, который будет проигнорирован компилятором языка. Очень старые компиляторы допускали только комментарии в стиле `/* xxx */`, сейчас допустим также стиль `// xxx`, завершается такой комментарий концом строки (то есть вся оставшаяся строка, после символов `//` будет проигнорирована компилятором). По мнению практикующих специалистов следует меньше увлекаться раскрашиванием текста программы комментариями в старом стиле и использовать последовательность `/* xxx */` на случай, когда будет необходимо "выключить" на время большой фрагмент написанного кода, скажем, при его отладке. А многострочные комментарии можно оформить через `//` в начале каждой новой строки. Придерживаясь этого правила, перед началом первой строки "выключаемого" фрагмента кода будет удобно поставить `/*`, а после последней строки "выключаемого" фрагмента `*/`. Четыре щелчка по клавишам, и много строк, или даже целые страницы кода будут исключены из компиляции. Но если внутри исключённого фрагмента окажется комментарий в стиле `/**/`, то такой трюк, естественно, не пройдет.

Также следует отметить, что текст многострочных операторов в C++ принято записывать "лесенкой", со сдвигом нижних строк оператора на одну табуляцию (или четыре пробела, споры на эту тему не утихают до сих пор) в соответствии со смыслом оператора. В вышеприведенном примере на такую табуляцию сдвинуты все строки внутри функции по отношению к открывающей тело функции и закрывающей её фигурной скобке. Часто настройки редактора, в котором набирается программа, позволяют установить табуляцию размером в четыре пробела, и установить признак "заполнять табуляцию пробелами". При обсуждении стилистики кода важно отметить, что стиль написания кода почти всегда определяется командой и почти всегда различается от команды к команде, но всегда соблюдается в рамках одной команды разработчиков.

Как работает компилируемый язык

Прежде, чем говорить о языках программирования и о том, что такое компиляция, как она работает и прочих интересных вещах, нам необходимо познакомиться с понятием, которое будет сопровождать весь курс и в целом всю программистскую жизнь - это понятие имени. Имя - это некий символьный идентификатор (переменная, контейнер) для некоторого числа (числом в свою очередь является адрес ячейки памяти, куда записывается значение). Простейший пример - запись равенства: `name = 123456`. Различие между именем и числом задает признак числа, в программе признаком числа является первый символ, имя (идентификатор) не должно начинаться с цифры. Таким образом компилятор однозначно может определить, что является именем, а что числом. Это отличие накладывает на программиста очевидное ограничение: невозможность создать идентификаторы, начинающиеся с цифр.

Общий алгоритм работы со всеми компилируемыми языками, в том числе C++ выглядит следующим образом:

- программист пишет текст программы,
- далее при помощи программы-транслятора этот текст преобразуется в исполняемые машинные коды
- после чего исполняемые коды запускаются на компьютере пользователя.

Применим этот общий алгоритм для написания нашей первой программы. Предполагается, что на данный момент у Вас установлена либо среда разработки, либо текстовый редактор и компилятор по отдельности.

Первая программа (`program.cpp`)

```
#include <iostream>
int main(int argc, char** args) {
	std::cout << "Hello, World!\n";
	return 0;
}
```

Эта программа состоит из нескольких обязательных частей:

- подключение стандартной библиотеки ввода-вывода `iostream`, чтобы получать доступ к стандартным средствам ввода-вывода той ОС, на которой будет исполняться наша программа (стандартных потоков три: `cin`, `cout`, `cerr`. для ввода, вывода и вывода сообщений об ошибках, соответственно)
- точка входа в программу функция `int main(int argc, char** args)`. О том, что такое функции, зачем они нужны и как работают мы поговорим позднее, на данный момент нам нужно запомнить, что это отправная точка работы любой нашей программы на языке C++. Никакие действия в C++ нельзя выполнять вне функций
- возвращаемое из функции значение (тип функции, о котором мы говорили ранее). Аналогично предыдущему пункту, более подробно о возвращаемых значениях мы будем говорить позднее. Пока что нам нужно понимать, что таким образом наша программа сообщит операционной системе о том, что всё закончилось хорошо, ошибок не возникло, программа завершилась штатно.

Основная, функциональная часть программы, так называемая "бизнес-логика", то, ради чего программа и пишется, в нашем примере сконцентрировано на четвёртой строке. Здесь мы при помощи оператора `<<` просим ОС вывести в свой стандартный вывод строку `"Привет, мир"` и снабдить её символом перехода на новую строку при помощи стандартной библиотеки. То есть оператор `<<` как бы указывает, какую строку надо поместить и куда. Все строки в языке C++ всегда пишутся в двойных кавычках и выводятся на экран "как есть". Подавляющее большинство операторов языка заканчиваются точкой с запятой (о тех, которые не, мы поговорим позднее), а ставить в конце выводимых строк объект `std::endl` или заканчивать строку переходом на новую - хороший тон. Core guidelines рекомендуют использовать `\n` вместо `endl` там, где не нужен весь функционал объекта `endl`.

Трансляция программы

```
      clang++ -o program program.cpp
(или) g++ -o program program.cpp
(или) c++ -o program program.cpp
```

Трансляция программы состоит из нескольких этапов, таких как препроцессинг (урок 7), компиляция, ассемблирование и линковка (компоновка). В результате вызова приведённой команды транслятор отработает все четыре этапа. Ключ **`-o`** указывает на название выходного файла. Если его не указать (в таком, полном, исполнении), будет создан файл со стандартным названием **`a.out`** (**`a.exe`** для Windows), или если будут указаны другие ключи вывод осуществляется в терминал. В этом примере выбран транслятор (компилятор) **`clang`** идентично ему будет работать **`g++`** (MinGW). В этом примере после работы транслятора и ввода команды **`ls`** (**`dir`** для Windows) можно будет увидеть программу **`program`**. Важная особенность компилятора в том, что он создаётся и работает специфично для каждой ОС, то есть компиляторы языка C++ для ОС Windows, Linux, Mac OS X - это разные компиляторы, со своими специфичными особенностями, некоторые из которых мы рассмотрим на уроке про препроцессинг. То есть, сам язык C++ (основная его часть) является кроссплатформенной,

поскольку компиляторы языка существуют для подавляющего большинства ОС и архитектур, но компилировать исходники в программу нужно чуть сложнее, чем "один раз нажал на кнопочку, и везде работает".

Этапы трансляции программы

Приведенные ниже примеры даны с учётом установки clang. Если установлен g++, изменится только первая, вызываемая программа (на g++ или c++), ключи и параметры останутся такими же.

- **Препроцессинг** - это этап на котором исполняются так называемые директивы препроцессора, в нашей первой программе такая всего одна `#include`. Результатом выполнения этого шага станет файл **`program.ii`**, открыв его можно увидеть как значительно преобразился и дополнился стандартной библиотекой написанный Вами код. Увидеть этот промежуточный файл можно указав компилятору ключ **`-E`**, говорящий о том, что шаг компиляции уже не нужен **`clang++ program.cpp -E > program.ii`** или **`clang++ -E program.cpp -o program.ii`**. Здесь символ **`>`** это указание операционной системе, а не компилятору, куда осуществлять вывод результата работы компилятора, такой, “потоковый” вывод результата работает только на первом этапе. Для этого и всех последующих этапов корректным ключом для компилятора является именно является **`-o`**.
- **Компиляция** - это непосредственное преобразование языка C++ в язык ассемблера. Язык ассемблера не является кроссплатформенным, поэтому этот код будет разным для разных процессорных архитектур и операционных систем. Язык ассемблера всё ещё человекочитаемый, поэтому можно посмотреть результат этого шага вызвав компилятор с ключом **`-S`** для препроцессированного файла **`clang++ -S program.ii -o program.s`**.
- **Ассемблирование** - этап преобразования ассемблерного кода в машинные коды. На этом этапе создаётся объектный файл (имеющий расширение **`.o`**). Такие объектные файлы создаются из всех исходников. Иногда объектные файлы оставляют в таком, скомпилированном и транслированном виде для создания библиотек (чтобы не перекомпилировать их каждый раз). Ассемблирование можно произвести утилитой **`as`**, поставляемой вместе с компилятором **`g++`**. Для этого нужно выполнить команду **`as program.s -o program.o`**. Несмотря на то, что полученный код абсолютно не человекочитаем, мы всё ещё можем изменить наш строковый литерал `"Hello, World!"` на любой другой (для некоторых ОС важна длина литералов в объектных файлах, это тоже важно помнить). На данном этапе, в зависимости от установленного инструментария могут возникать ошибки. Это связано с тем, что если установить clang отдельно от прочего инструментария, он содержит только внутренний ассемблер. Проблема решается дополнительным ключом -as-integrated, то есть команда на ассемблирование будет выглядеть как **`clang++ -as-integrated program.s -o program.o`**
- **Линковка** - это завершающий этап преобразования текста программы в работающее приложение. На данном этапе мы можем объединить несколько объектных файлов и статических библиотек для применения в одном приложении. На этом этапе компилятор создаёт свою собственную *таблицу символов* (таблицу соответствия объектов их типам и

областям видимости). Такая таблица хранит адреса всех объектов, данных и функций, чтобы компоновщик мог связать их между собой. Выполняется этот этап представленной в самом начале командой (но не для файла исходного кода, а для файла множества файлов ассемблированных, объектных) `clang++ program.o -o program`.

Запуск программы

```
./program
```

Запуск программы для разных ОС отличается (как отличаются и трансляторы), и не представляет собой ничего необычного, запускать программы, как пользователи, мы все умеем.

Методы трансляции программ:

1. Компиляция
    a. Выходом компилятора является машинный код для конкретной архитектуры процессоров, ОС
    b. Компилятор не контролирует исполнение программы на целевой машине
    c. Возможна оптимизация времени выполнения отдельных операций за счет процессора
    d. Исходный текст обрабатывается в несколько проходов разными алгоритмами
2. Интерпретация
    a. Компилятор является средой исполнения программы и не имеет выходного кода
    b. Требуется наличие реализации компилятора для исполнения программы на целевой машине
    c. Исполняемая программа по своему представлению идентична исходному коду

Стандартная библиотека:

- Отвечает за связь языка программирования с машиной
- Должна быть составлена как отдельная программа (набор структур классов и функций) для любой аппаратной платформы, на которой транслируется язык программирования
- Стандарт языка программирования полностью описывает реализацию стандартной библиотеки

Этапы трансляции программы, сборка

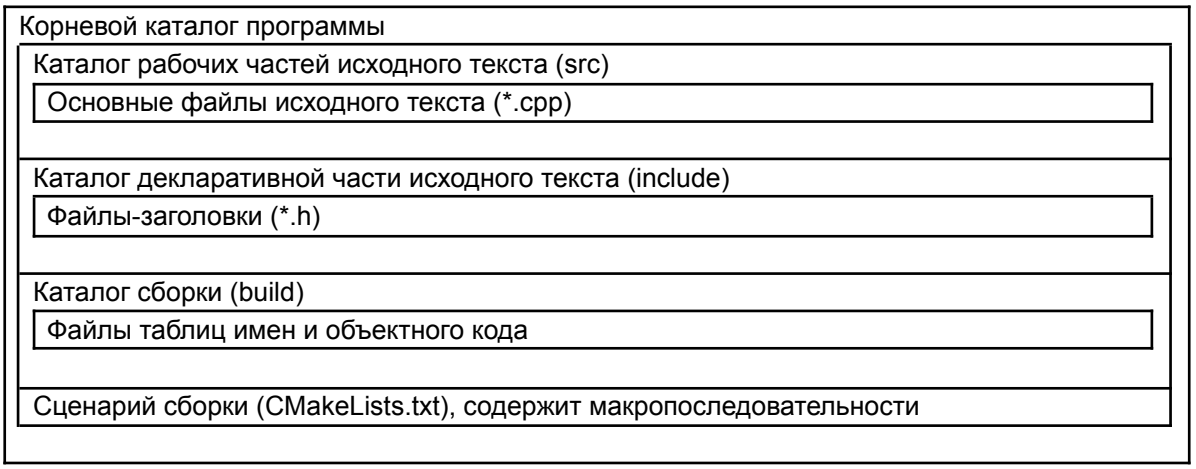
Чтобы понять, как происходит трансляция с точки зрения программиста, стоит понять из чего обычно состоит исходный код программы. Он подразделяется на декларативную и рабочую части

1. Декларативная часть:
    a. Выносится во внешние файлы-заголовки (*.h/*.hpp)
    b. Содержит объявление констант, внешние имена
    c. Может отдаваться другому программисту в качестве внешней таблицы имен для объектной трансляции (проще говоря, чтобы другие знали, какие функции предоставляет наша программа, впоследствии это стало называться программным интерфейсом или API)
2. Рабочая часть:

a. Хранится в основных файлах исходного текста (*.cc, *.cpp)
b. Рассматривается как отдельная единица трансляции в объектный код
c. Хранит описание всех определенных в программе операций

Работа макросборщика в простейшем случае состоит в формировании специфического для конкретной машины или операционной системы набора действий, которые будут являться своего рода средой для облегчения и автоматизации процесса компиляции программы. В том числе макросборщик может разрешить вопрос кроссплатформенной компиляции (вызвать платформозависимые части кода или сборщика). По сути - это набор скриптов, созданных по определенным правилам.

Общая структура программы:



Простейший сценарий сборки CMake

```
#Указываем минимальную версию CMake, которая в состоянии собрать наш проект
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
#Присваиваем имя проекту сборки
PROJECT(program)
#Регистрируем исполняемый файл программы как единицу сборки, называемую TARGET
add_executable(program <Макрос TARGET по умолчанию> src/program.cpp)
#Регистрируем каталог с заголовками специфичный для конкретной единицы сборки
target_include_derictories(program PRIVATE ${CMAKE_CURRENT_SOURCE_DIR}/include)
```

Типовой алгоритм сборки с CMake, пример для GCC/Clang и GNU Automake. О часто встречающейся ошибке при исполнении типового сценария смотрите в конце этого раздела.

```
cd build
cmake ../
make
```

Теперь можно вернуться к уже написанной простейшей программе и выполнить создание папок, для структурирования

```
mkdir build include src
```

Файл **program.cpp** нужно перенести в каталог **src** и создать необходимые файлы в каталогах

```
include/program.h
CMakeLists.txt
```

После чего перейдем в текстовый редактор и разделим исходный текст программы по созданным файлам

Файл **program.h**

```
#include <iostream> /* использование стандартной библиотеки */
#define SUCCESS 0 /* определение константы успешного завершения программы */
```

Файл **program.cpp**

```
#include "program.h"
using namespace std; // подключение пространства имён стандартной библиотеки,
чтобы можно было использовать функции из неё без явного указания
int main(int argc, char** args) {
	cout << "Hello, World!" << endl;
	return SUCCESS;
}
```

Файл **CMakeLists.txt**

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
PROJECT(Program)

add_executable(program src/program.cpp)

target_include_directories(program PRIVATE include)
```

Далее нашу программу можно собирать и запускать

из папки **build/**

выполняем команду **cmake ../** после чего увидим, что в каталоге сборки появился файл сгенерированный Makefile. Часто случается, что установленный инструментарий не совпадает с тем, что ожидает CMake, например, может быть установлен компилятор MSVC, а утилита **make** из состава GNU Automake. В этом случае утилита **make** обычно выдаёт ошибку:

```
make: *** No targets specified and no makefile found. Stop.
```

И необходимо задать ключ сборки (указать сборщику, какой именно Makefile нужно создавать), например **cmake -G "MinGW Makefiles" ..**

выполняем команду **make** - программа соберётся (скомпилируется), появится исполняемый файл **program**

командой **./program** - запускаем исполняемый файл

Практическое задание

1. Скачайте и настройте на компьютере среду программирования. Не важно, какой именно инструментарий Вы выберете, главное, чтобы Вы понимали как с использованием выбранного Вами инструментария выполнять компиляцию и запуск исходного кода, а также осуществлять управление компиляцией. Результат выполнения задания:
    a. Архив с файлами исходного кода приложения “Привет, мир”
    b. Приложите в архив скриншот с результатом выполнения программы с использованием Вашего инструментария.
2. * Приложите в архив промежуточные файлы компиляции
3. ** Скомпилируйте исходный код со строкой “Привет, мир”, а скомпилированный файл ассемблируйте и слинкуйте со строкой “Привет, Geekbrains”, то есть изменённая строка должна быть в ассемблерном файле. Приложите файл программы с изменённым приветствием.

Дополнительные материалы

Пара слов о последних версиях CMake (3.x)

Макросборщик CMake - это не просто инструмент автоматизации. Важно помнить, что настройки сборки - это код. То есть CMake - это, фактически, скрипт, который создаёт скрипт сборки, автоматизатор. Сборка при помощи CMake происходит в несколько этапов:

1. Конфигурация
    a. Выполнение скриптов
    b. создание таргетов (целевых, выходных файлов сборки)
    c. заполнение свойств
    d. заполнение кэша
    e. Не выполняются генераторные выражения
2. Генерация файлов сборочной системы
    a. Выполняются генераторные выражения
    b. Вычисление конечных значений таргетов
    c. Создание кастомных отложенных генерируемых файлов
    d. Собственно, генерация файлов сборочной системы

Эти два этапа обычно в консоли сливаются в один, поскольку выполняются одной командой (с ключом `-G`) например: `cmake -G make ~/MyProjectSrc`. Где `make` - это название финального сборщика. В графическом интерфейсе есть две кнопки Configure и Generate, но при нажатии на вторую конфигурация всё равно произойдёт. Генераторные выражения имеют общий вид `$<SOMETHING>`, могут быть вложенными друг в друга, могут быть использованы только в контексте тех свойств, которые это поддерживают, и с каждым релизом сборщика количество допустимых свойств увеличивается. Такие выражения нужны, например, для разделения кроссплатформенных сборок (путь до конечного файла сборки на разных ОС может отличаться, также могут отличаться системные переменные и многое другое). Типы генераторных выражений:

- булев: для условий
- строковый (в т.ч. список, то есть строки, разделённый точкой с запятой).

Цель генераторных выражений - получить какую-то строку, например, простое условие и условие с двумя исходами:

```
$<condition:true_string>
$<IF:condition, true_string, false_string>
```

Генераторные выражения напоминают очень маленький функциональный язык программирования, потому что кроме условий в нём есть операции манипулирования строками:

```
$<LOWER_CASE:string>
$<UPPER_CASE:string>
```

Работы со списками:

```
$<JOIN:list, string>
$<REMOVE_DUPLICATES:list>
$<FILTER:list, INCLUDE, regex>
$<FILTER:list, EXCLUDE, regex>
```

Также на этапе генерации можно записывать и дописывать файлы:

```
file(WRITE <filename> <content>)
file(APPEND <filename> <content>)
```

Далее после генерации следует третий этап - сборка - это выполнение вспомогательных скриптов и создание конечных. Четвёртый - тесты - то есть выполнение тестов и разделение приложения и отладочной информации, пятый - инсталляция - собственно исполнение сгенерированных конечных скриптов и шестой - упаковка (опционально). Эти этапы выходят далеко за пределы курса основ.

Установка инструментария

Windows/VS/clang, Windows/Qt/gcc, Qt/include iostream

Используемые источники

1. *Брайан Керниган, Деннис Ритчи.* Язык программирования C. — Москва: Вильямс, 2015. — 304 с. — ISBN 978-5-8459-1975-5.
2. Stroustrup, Bjarne The C++ Programming Language (Fourth Edition)