Процесс компиляции программ на C++

1) Препроцессинг

Самая первая стадия компиляции программы.

Препроцессор — это макро процессор, который преобразовывает вашу программу для дальнейшего компилирования. На данной стадии происходит работа с препроцессорными директивами. Например, препроцессор добавляет хэдеры в код (#include), убирает комментирования, заменяет макросы (#define) их значениями, выбирает нужные куски кода в соответствии с условиями #if, #ifdef и #ifndef.

Хэдеры, включенные в программу с помощью директивы #include, рекурсивно проходят стадию препроцессинга и включаются в выпускаемый файл. Однако, каждый хэдер может быть открыт во время препроцессинга несколько раз, поэтому, обычно, используются специальные препроцессорные директивы, предохраняющие от циклической зависимости.

2) Компиляция

На данном шаге g++ выполняет свою главную задачу — компилирует, то есть преобразует полученный на прошлом шаге код без директив в ассемблерный код. Это промежуточный шаг между высокоуровневым языком и машинным (бинарным) кодом.

Ассемблерный код — это доступное для понимания человеком представление машинного кода.

3) Ассемблирование

Так как x86 процессоры исполняют команды на бинарном коде, необходимо перевести ассемблерный код в машинный с помощью ассемблера.

Ассемблер преобразовывает ассемблерный код в машинный код, сохраняя его в объектном файле.

Объектный файл — это созданный ассемблером промежуточный файл, хранящий кусок машинного кода. Этот кусок машинного кода, который еще не был связан вместе с другими кусками машинного кода в конечную выполняемую программу, называется объектным кодом.

Далее возможно сохранение данного объектного кода в статические библиотеки для того, чтобы не компилировать данный код снова.

4) Компоновка (линковка)

Объектных файлов может быть много и нужно их всех соединить в единый исполняемый файл с помощью компоновщика (линкера). Поэтому мы переходим к следующей стадии.

Компоновщик (линкер) связывает все объектные файлы и статические библиотеки в единый исполняемый файл, который мы и сможем запустить в дальнейшем. Для того, чтобы понять как происходит связка, следует рассказать о таблице символов.

Таблица символов — это структура данных, создаваемая самим компилятором и хранящаяся в самих объектных файлах. Таблица символов хранит имена переменных, функций, классов, объектов и т.д., где каждому идентификатору (символу) соотносится его тип, область видимости. Также таблица символов хранит адреса ссылок на данные и процедуры в других объектных файлах.

Именно с помощью таблицы символов и хранящихся в них ссылок линкер будет способен в дальнейшем построить связи между данными среди множества других объектных файлов и создать единый исполняемый файл из них.

5) Загрузка

Последний этап, который предстоит пройти нашей программе — вызвать загрузчик для загрузки нашей программы в память. На данной стадии также возможна подгрузка динамических библиотек.

Библиотека — это «сборник» кода, который можно многократно использовать в самых разных программах. Как правило, **библиотека в языке C++ состоит из 2-х частей:**

Заголовочный файл, который объявляет функционал библиотеки.

Предварительно скомпилированный бинарный файл, содержащий реализацию функционала библиотеки.

Некоторые библиотеки могут быть разбиты на несколько файлов и/или иметь несколько заголовочных файлов.

Типы библиотек

Библиотеки предварительно компилируют по нескольким причинам. Во-первых, их код редко меняется. Было бы напрасной тратой времени повторно компилировать библиотеку каждый раз при её использовании в новой программе. Во-вторых, поскольку весь код предварительно скомпилирован в машинный язык, то это предотвращает получение доступа к исходному коду (и его изменение) сторонними лицами. Этот пункт важен для предприятий/людей, которые не хотят, чтобы результат их труда (исходный код) был доступен всем (интеллектуальная собственность, все дела).

Есть 2 типа библиотек: статические и динамические.

Статическая библиотека (или **«архив»**) состоит из подпрограмм, которые непосредственно компилируются и линкуются с вашей программой. При компиляции программы, которая использует статическую библиотеку, весь функционал статической библиотеки (тот, что использует ваша программа) становится частью вашего исполняемого файла. В Windows статические библиотеки имеют расширение **.lib** (сокр. от «**lib**rary»), тогда как в Linux статически библиотеки имеют расширение **.a** (сокр. от «**a**rchive»).

Одним из преимуществ статических библиотек является то, что вам нужно распространить всего лишь 1 (исполняемый) файл, чтобы пользователи могли запустить и использовать вашу программу. Поскольку статические библиотеки

становятся частью вашей программы, то вы можете использовать их подобно функционалу своей собственной программы. С другой стороны, поскольку копия библиотеки становится частью каждого вашего исполняемого файла, то это может привести к увеличению размера файла. Также, если вам нужно будет обновить статическую библиотеку, вам придется перекомпилировать каждый исполняемый файл, который её использует.

Динамическая библиотека (или **«общая библиотека»**) состоит из подпрограмм, которые подгружаются в вашу программу во время её выполнения. При компиляции программы, которая использует динамическую библиотеку, эта библиотека не становится частью вашего исполняемого файла — она так и остается отдельным модулем. В Windows динамические библиотеки имеют расширение **.dll** (сокр. от «**d**ynamic **l**ink **l**ibrary» = «библиотека динамической компоновки»), тогда как в Linux динамические библиотеки имеют расширение **.so** (сокр. от «**s**hared **o**bject» = «общий объект»). Одним из преимуществ динамических библиотек является то, что разные программы могут совместно использовать одну копию динамической библиотеки, что значительно экономит используемое пространство. Еще одним преимуществом динамической библиотеки является то, что её можно обновить до более новой версии без необходимости перекомпиляции всех исполняемых файлов, которые её используют.

Поскольку динамические библиотеки не линкуются непосредственно с вашей программой, то ваши программы, использующие динамические библиотеки, должны явно подключать и взаимодействовать с динамической библиотекой. Этот механизм не всегда может быть понятен для новичков, что может затруднить взаимодействие с динамической библиотекой. Для упрощения этого процесса используют **библиотеки импорта.**

Кроме того, динамические библиотеки необязательно загружаются - они обычно загружаются при первом вызове и могут совместно использоваться среди компонентов, которые используют одну и ту же библиотеку (несколько загрузок данных, одна загрузка кода).

Проще говоря:

Статическая библиотека это фактически архив объектных файлов, который используется в процессе статической линковки. В результате статической линковки из многих объектных файлов получается один исполняемый, запускается статическая линковка в момент создания исполняемого файла.

Динамическая библиотека это фактически исполняемый файл (т.е. DLL и EXE в Windows имеют одинаковый формат). Динамическая линковка запускается в момент создания процесса (когда вы запускаете исполняемый файл на выполнение), линкуются между собой несколько исполняемых файлов каждый раз, когда создается новый процесс. Так же возможна динамическая линковка уже после запуска, т.е. новая библиотека может быть подгружена в адресное пространство ужа работающего процесса.

Сделать из статической библиотеки динамическую в принципе можно - необходимо ее (статически) слинковать в динамическую библиотеку, при этом будет создана динамическая библиотека и статическая стаб-библиотека, которую можно использовать в проекте вместо статической библиотеки, чтобы вызывать функции из динамической библиотеки.