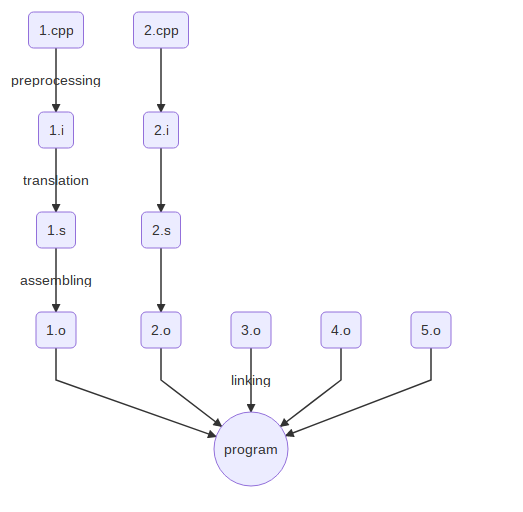
Этапы компиляции программы на C++



В качестве примера рассмотрим следующую программу:

// driver.cpp

#include <iostream>

using namespace std;

#define RETURN return 0

int main() {

// printing “Hello, world!” to console

cout << "Hello, world!" << endl;

RETURN;

}

1. **Препроцессоринг**

**Препроцессор** — это **макропроцессор**, 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, #elif, #else, #endif.

Получим **препроцессированный код** в выходной **файл driver.ii** (прошедшие через стадию препроцессинга C++ файлы имеют расширение .ii), используя флаг -**E**, который сообщает компилятору, что компилировать (об этом далее) файл не нужно, а только провести его препроцессинг:

g++ -E driver.cpp -o driver.ii

**На выходе** препроцессора получаются так называемые **компиляционные листы**, состоящие исключительно из **выражений** языка C++, которых должно быть достаточно для создания объектных файлов на следующих этапах обработки. Последнее означает, что на момент использования каких-либо символов языка из других файлов, **объявления этих символов должны присутствовать** в компиляционном листе выше. Именно такие подстановки и призван осуществлять препроцессор. Часто, на вход препроцессора поступает файл размером в несколько десятков строк, а на выходе получается компиляционный лист из десятков тысяч строк.

Если рассмотреть нашу функцию main() после препроцессинга, то макрос RETURN был заменен:

int main() {

cout << "Hello, world!" << endl;

return 0;

}

1. **Компилирование**

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.

Используя **флаг -S**, который сообщает компилятору остановиться после стадии компиляции, получим ассемблерный код в выходном файле driver.s:

$ g++ -S driver.ii -o driver.s

1. **Ассемблирование**

В данном случае, мы имеем **компилятор** с языка **ассемблера**. Результатом его работы является **объектный файл** полученный на основе всего того текста, что был предоставлен в **компиляционном листе**. Поэтому можно говорить, что каждый объектный файл проекта соответствует одному **компиляционному листу** проекта.

**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.

Получим машинный код с помощью ассемблера (as) в выходной объектный файл driver.o:

$ as driver.s -o driver.o

Ассемблирование **не является обязательным** процессом обработки файлов на языке C++. В данном случае, мы наблюдаем лишь общий подход в архитектуре проекта коллекции компиляторов GNU. Чтобы максимально объединить разные языки в одну коллекцию, **для каждого** из языков реализуется **свой транслятор** на язык ассемблера и, при необходимости, препроцессор, а компилятор с языка ассемблера и линковщик делаются общими для всех языков коллекции.

1. **Линковка**

Также называется **связывание**, **сборка** или **линковка**.

На этапе линковки выполняется **объединение всех** объектных **файлов** проекта, откомпилированных по соответствующим компиляционным листам проекта в **единую сущность** (исполняемый файл). Это может быть приложение, статическая или динамическая библиотека. Разница в бинарных заголовках целевых файлов и несколько различной внутренней организацией. Первичной задачей линковки следует назвать задачу по **подстановке адресов вызова** внешних **объектов**, которые были образованы в объектных файлах проекта. Соответствующие **реализации сущностей с адресами** их размещения должны **находится в видимости** линковщика. Эти сущности должны быть **либо в объектных файлах**, тогда они **должны быть указаны в списке** линковки, **либо во внешних библиотеках** функций, статических или динамических, тогда они должны быть указаны **в списке внешних библиотек**.

Источники:

* <https://habr.com/ru/articles/478124/>
* <https://ejudge.179.ru/tasks/cpp/total/105.html>
* <https://knzsoft.ru/cpp-bgr-ls1/>
* <https://cpp-kt.github.io/cpp-notes/05_compilation.html>

Подготовил **Занин Вячеслав ВМК-21**