Семинар #9: Подключение библиотек. Создание окна с помощью библиотеки SFML

Раздельная компиляция

Этапы сборки проекта на языке C++

1. **Препроцессинг**. Обрабатываются директивы компилятора `#include`, `#define` и другие. Удаляются комментарии. Чтобы исполнить только этот шаг, нужно передать компилятору опцию `-E`:

   ```
   g++ -E main.cpp > preprocessed.cpp
   ```

2. **Компиляция**: каждый файл исходного кода (файл расширения `.cpp`) транслируется в код на языке ассемблера. Чтобы исполнить только этапы препроцессинга и компиляции, нужно передать компилятору опцию `-S`:

   ```
   g++ -S main.cpp
   ```

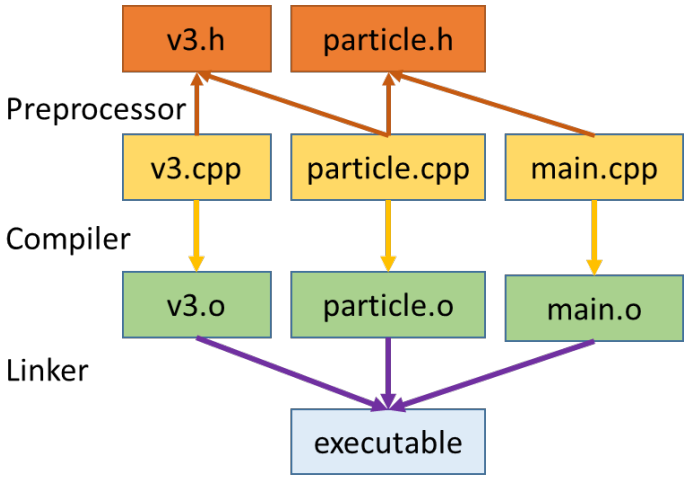
3. **Ассемблирование**: каждый файл на языке ассемблера транслируется в машинный код. В результате создаётся объектный файл с расширением `.o`. Чтобы исполнить процесс до этой стадии включительно нужно передать компилятору опцию `-c`:

   ```
   g++ -c main.cpp
   ```

4. **Линковка**: Все объектные файлы сливаются друг с другом, а также с другими библиотеками. Даже если ваш проект состоит из одного файла, вы наверняка используйте как минимум стандартную библиотеку и на этом этапе ваш код соединяется с другими библиотеками.

   ```
   g++ main.o
   ```

Сборка многофайловой программы



Можно собрать всё сразу:

```
g++ main.cpp particle.cpp v3.cpp
```

Либо можно собрать по частям:

```
g++ -c main.cpp
g++ -c particle.cpp
g++ -c v3.cpp
g++ main.o particle.o v3.o
```

Виды библиотек

1. **header-only библиотеки**
   Весь исходный код хранится в `.h` или `.hpp` файле и подключается с помощью директивы `#include` (очень просто подключить).

2. **Исходный код**
   Библиотека поставляется в виде исходного кода (все `.h` и `.cpp` файлы). Для того чтобы использовать эту библиотеку, её нужно сначала скомпилировать, что может быть очень непросто для больших библиотек, так как процесс сборки может сильно отличаться на разных операционных системах и компиляторах.

3. **Статическая библиотека**
   Библиотека поставляется в виде header-файлов(`.h`) и предварительно скомпилированных файлов библиотеки Расширение статических библиотек на linux: `.a` (archive). Расширение на windows: `.lib` (library). Эти библиотеки подключаются на этапе линковки. После линковки содержимое этих библиотек содержится в исполняемом файле. Такие библиотеки проще подключить к проекту, чем исходный код. Однако, вам обязательно иметь версию библиотеки, скомпилированную на такой же ОС и на таком же компиляторе, иначе она не подключится. Обратите внимание, что статические библиотеки обязательно должны иметь префикс `lib`. Например, если мы хотим получить библиотеку под названием `image`, то файл должен называться `libimage.a`.

4. **Динамическая библиотека**
   Библиотека поставляется в виде header-файлов(`.h`) и предварительно скомпилированных файлов библиотеки Расширение динамических библиотек на linux: `.so` (от shared object). Расширение на windows: `.dll` (от dynamic link library)). Эти библиотеки подключаются на этапе *выполнения программы*. Благодаря тому, что динамическая библиотека подключается на этапе выполнения, если несколько программ будут использовать одну и ту же библиотеку, то она будет загружаться в память лишь один раз.

Создание статических библиотек

Чтобы создать свою статическую библиотеку вам нужно:

1. Создать объектный файл необходимого исходного файла.

2. Превратить объектный файл (или файлы) в библиотеку, используя утилиту `ar`:

   ```
   ar rvs libimage.a image.o
   ```

3. После этого файл `libimage.a` можно будет подключить к любому другому проекту примерно так:

   ```
   g++ main.cpp -I<путь до header-файлов> -L<путь до libimage.a> -limage
   ```

Создайте статическую библиотеку из файла `image.cpp`. Создайте папку `image/` в которой будет храниться наша библиотека. В этой папке создайте ещё 2 папки: `include` и `lib/`. Поместите в папку `image/include` заголовочный файл `image.hpp`. Поместите в папку `image/lib` файл статической библиотеки `libimage.a`. Затем вам нужно удалить файл `image.cpp` и собрать программу используя только статическую библиотеку (не забывайте про опции `-I`, `-L` и `-l`).

Создание динамических библиотек

Чтобы создать динамическую библиотеку из файла исходного кода (`image.cpp`):

```
g++ -c -fPic image.cpp -o image.o
g++ -shared -o libimage.so image.o
```

Чтобы скомпилировать код с подключением динамической библиотеки:

```
g++ -o main.exe main.cpp libimage.so
```

или

```
g++ -o main.exe main.cpp -limage
```

Но для этого понадобится добавить в переменную среды `LD_LIBRARY_PATH` (на Windows нужно добавить в переменную среды `PATH`) путь до папки, содержащий библиотеку.

1. Создайте динамическую библиотеку и скомпилируйте саму программу с подключением динамической библиотеки

2. Проверьте чему равен размеры исполняемых файлов в случае подключения статической и динамической библиотеки.

3. Что будет происходить, если перенести файл динамической библиотеки в другую папку. Запустится ли исполняемый файл?

Подключение библиотеки SFML

Библиотека SFML (Simple and Fast Multimedia Library) - простая и быстрая библиотека для работы с мультимедиа. Кроссплатформенная (т. е. одна программа будет работать на операционных системах Linux, Windows и MacOS). Позволяет создавать окно, рисовать в 2D и 3D, проигрывать музыку и передавать информацию по сети. Для подключения библиотеки вам нужно скачать нужную версию с сайта: sfml-dev.org.

Подключение вручную:

Для подключения библиотеки вручную через опции `g++` нужно задать путь до папок `include/` и `lib/` и названия файлов библиотеки, используя опции `-I`, `-L` или `-l`.

```
g++ .\main.cpp -I<путь до include> -L<путь до lib> -lsfml-graphics -lsfml-window -lsfml-system
```

Например так:

```
g++ .\main.cpp -I./SFLL-2.5.1/include -L./SFLL-2.5.1/lib -lsfml-graphics -lsfml-window -lsfml-system
```

`bash`-скрипт:

Так как постоянно прописывать в терминале сборку проекта может быть затруднительно, то можно положить весь процесс сборки в специальный `bash`-скрипт. `bash`-скрипт - это просто файл кода языка терминала linux. (Для windows есть аналогичные `bat`-скрипты) Пример можно посмотреть в `2sfml/1bash_script`.

Подключение с помощью make (файл Makefile):

Если вы программируйте на Linux/MacOS или на Windows с компилятором MinGW, то можно создать файл под названием `Makefile` в текущей директории. и написать в нём:

```
main.exe:
g++ .\main.cpp -o main.exe -o -I<путь до include> -L<путь до lib> -lsfml-graphics -lsfml-window
-lsfml-system
```

После этого можно будет скомпилировать программу вызвав `make` (на Linux) или `mingw32-make` (на Windows - MinGW)
Пример make-файла можно посмотреть в `classroom_tasks/2sfml/3makefile`

Подключение с помощью cmake (файл CMakeLists.txt):

Система автоматической сборки cmake позволяет собирать большие проекты. Чтобы работать с ней вам нужно её скачать по адресу cmake.org и установить переменную среды `PATH`. Затем нужно создать файл `CMakeLists.txt` в директории вашего проекта и написать в нём:

```
cmake_minimum_required(VERSION 2.8.0)
project(simple_sfml)

# Создадим исполняемый файл по имени simple_sfml из исходного файла main.cpp
add_executable(simple_sfml main.cpp)

# Найдём библиотеку SFML автоматически с компонентами graphics, system и window
find_package(SFML 2.5 COMPONENTS graphics system window)
# Подключим эту библиотеку
target_link_libraries(simple_sfml sfml-graphics sfml-system sfml-window)
```

После этого, проект можно собрать так:

```
cmake -G<генератор> <путь до CMakeLists.txt>
```

Пример make-файла можно посмотреть в папке `classroom_tasks/2sfml/4cmake` и `classroom_tasks/2sfml/5cmake_find_package`

- Соберите проект в папке `0basics`, используя один из приведённых выше способов (предпочтительно - make или cmake).

Работа с библиотекой:

Документация и туториалы по библиотеке SFML можно найти на оффициальном сайте: `https://www.sfml-dev.org/`sfml-dev.org. Пример простой программы, для работы с SFML в папке `1sfml_basics`. Основные классы SFML и их методы:

- `sf::Vector3f, sf::Vector2f, sf::Vector2i` и т. д. Классы для математического вектора с перегруженными операциями. (аналогичные тем, что мы писали на предыдущих занятиях). sfml-dev.org/documentation/2.5.1/classsf_1_1Vector2.php
- `sf::RenderWindow` - класс для окна. sfml-dev.org/documentation/2.5.1/classsf_1_1RenderWindow.php
- `sf::CircleShape` - класс для фигуры - круг. sfml-dev.org/documentation/2.5.1/classsf_1_1CircleShape.php

Часть 2: Библиотека SFML:

Библиотека SFML (Simple and Fast Multimedia Library) - простая и быстрая библиотека для работы с мультимедиа. Кроссплатформенная (т. е. одна программа будет работать на операционных системах Linux, Windows и MacOS). Позволяет создавать окно, рисовать в 2D и 3D, проигрывать музыку и передавать информацию по сети. Для подключения библиотеки вам нужно скачать нужную версию с сайта: sfml-dev.org.

Подключение вручную:

Для подключения библиотеки вручную через опции `g++` нужно задать путь до папок `include/` и `lib/` и названия файлов библиотеки, используя опции `-I`, `-L` или `-l`.

```
g++ .\main.cpp -I<путь до include> -L<путь до lib> -lsfml-graphics -lsfml-window -lsfml-system
```

Например так:

```
g++ .\main.cpp -I./SFLL-2.5.1/include -L./SFLL-2.5.1/lib -lsfml-graphics -lsfml-window -lsfml-system
```

`bash`-скрипт:

Так как постоянно прописывать в терминале сборку проекта может быть затруднительно, то можно положить весь процесс сборки в специальный `bash`-скрипт. `bash`-скрипт - это просто файл кода языка терминала linux. (Для windows есть аналогичные `bat`-скрипты) Пример можно посмотреть в `2sfml/1bash_script`.

Makefile:

`make` – это специальная утилита, предназначенная для упрощения сборки проекта. В `2sfml/3makefile` содержится пример проекта с make-файлом. Содержимое make-файла представляет собой просто набор целей и соответствующих команд оболочки `bash`. Откройте make-файл и просмотрите его содержимое. Чтобы скомпилировать его просто:

```
make <имя цели>
```

либо просто

```
make
```

(в этом случае `make` запустит процесс создания первой цели)