# Семинар #5: Подключение библиотек.

## Часть 1: Многофайловые программы. Библиотеки

### 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
   ```

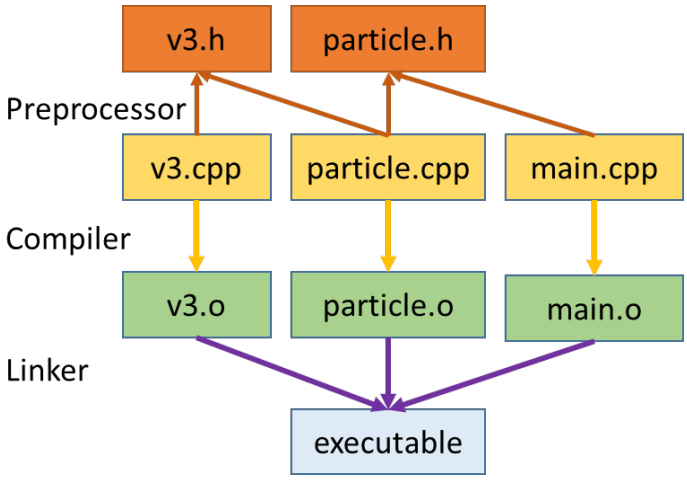
4. **Линковка**: Все объектные файлы сливаются друг с другом, а также с другими библиотеками. Даже если ваш проект состоит из одного файла, вы наверняка используйте как минимум стандартную библиотеку и на этом этапе ваш код соединяется с другими библиотеками.

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

### Задание:

- В папке `0stages` лежит исходный код простой программы. Пройдите поэтапно все стадии сборки с этой программой.

### 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` 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.

## Задания:

- **header:** В папке `1image/0header-only` лежит исходный код программы, которая использует класс `Image`. Это простой класс для работы с изображениями в формате `.ppm`. Скомпилируйте и запустите эту программу.

- **Случайные отрезки:** Используйте этот класс, чтобы создать изображение, состоящее из 100 случайных отрезков случайного цвета. Для случайных чисел используйте функцию `rand()` из библиотеки `<cstdlib>`.

- **Случайные прямоугольники:** Добавьте в этот класс метод `void draw_rectangle(const Vector2i& bottomleft, const Vector2i& torright, const Color& color)`. Используйте этот метод, чтобы создать изображение, состоящее из 100 случайных прямоуголиников случайного цвета.

- **Шум:** Добавьте в этот класс метод `void add_noise(float probability)`, который будет добавлять шум на картинку: каждый пиксель с вероятностью `probability` должен поменять цвет на случайный. Протестируйте этот метод на картинках.

- **Раздельная компиляция:** В папке `1image/1separate_compilation` лежит тот же код, но разделённый на 2 файла исходного кода. Скомпилируйте эту программу с помощью `g++`.Добавьте функции `draw_rectangle` и `add_noise` из предыдущих заданий в этот проект.

- **Статическая библиотека:** 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
     ```

В папке `1image/2static_library` лежит исходный код программы. Вам нужно создать статическую библиотеку из файла `image.cpp` и поместить полученный файл в папку `image/lib`, а header-файл поместить в папку `image/include`. Затем вам нужно удалить файл `image.cpp` и собрать программу используя только статическую библиотеку (не забывайте про опции `-I`, `-L` и `-l`).

- **Статическая библиотека 2:** В папке `1image/3static_test` лежит проект с одной очень маленькой статической библиотекой (содержит 1 функцию). Вам нужно собрать этот проект и запустить исполняемый файл.
- **Динамическая библиотека:** 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?

# 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)

## Задания:

- **Сборка:** скомпилируйте и запустите проект. Используйте `bash`-скрипт или make-файл. Более подробно – в папке `2sfml`.
- **Движение по окружности:** Заставьте кружок двигаться по окружности.
- **Броуновское движение:** Создайте `n = 50` кругов, которые будут двигаться случайным образом (направление и величина движения должны задаваться случайным образом на каждом кадре).
- **Задача n тел:** Создайте `n = 50` кругов, так, чтобы они притягивались друг к другу гравитационной силой. Начальные положения и скорости задайте случайным образом. Сила гравитации в двух измерениях обратно пропорциональна расстоянию между объектами.

$$F \sim \frac{1}{R}$$

- **Задача N тел с массой**
  Добавьте разную массу шарикам. При создании шарика масса должна задаваться случайным образом. Масса шарика должна быть пропорциональна площади (квадрату радиуса).
- **Электрические заряды**
  Смоделируйте взаимодействие заряженных частиц. Для этого нужно добавить поле в структуру `Ball`, которое будет определять величину заряда. Эта величина может быть как положительной, так и отрицательной. В начале работы программы заряд должен задаваться случайно. Заряды должны взаимодействовать по закону Кулона. Гравитацией можно пренебречь. Цвета зарядов должны быть различными (красный для положительного зарада и синий для отрицательного, интенсивность цвета - пропорциональна величине заряда).

- **Нажатие мыши**
  События нажатия мыши можно обработать с помощью следующего синтаксиса:

```
if (event.type == sf::Event::MouseButtonPressed)
{
    if (event.mouseButton.button == sf::Mouse::Right)
    {
        std::cout << "the right button was pressed" << std::endl;
        std::cout << "mouse x: " << event.mouseButton.x << std::endl;
        std::cout << "mouse y: " << event.mouseButton.y << std::endl;
    }
}
```

  Внутри цикла `while (window.pollEvent(event))`.
  Видоизмените вашу программу так, чтобы при нажатии левой кнопки мыши в том месте, где находится мышь, создавался бы шарик со средними массой и средним положительным зарядом зарядом. При нажатии правой кнопки мыши должен создаваться шарик с очень большой массой и очень большим положительным зарядом. При аналогичных нажатиях, но с зажатой клавишой Shift, должны создаваться отрицательные заряды.