Семинар #9: Библиотеки. Домашнее задание.

Задача 1. Стадии компиляции

В файле `simple_image` лежит исходный код простой программы, которая создаёт простое изображение. Пройдите поэтапно все стадии сборки с этой программой. В результате вы должны получить следующие файлы:

- Файл исходного кода на языке C++, который получается после этапа препроцессинга.
- Файл кода на языке ассемблера.
- Объектный файл.

Задача 2. Класс изображения

В файле `simple_image` лежит исходный код программы, которая использует класс `Image`. Это простой класс для работы с изображениями в формате `.ppm`. Скомпилируйте и запустите эту программу.

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

- **Заголовочный файл**
  Создайте файл `image.hpp` и перенесите класс `Image` в этот файл. Используйте директиву `#pragma once`, чтобы избежать в будущем возможного множественного включения. Таким образом получится мини-библиотека. Подключите этот заголовочный файл к вашей основной программу и скомпилируйте программу.

- **Раздельная компиляция:**
  Создайте ещё один файл `image.cpp`. Перенесите из файла `image.hpp` в файл `image.cpp` все определения методов. Таким образом в файле `image.hpp` должны остаться определение класса и объявление всех его методов, а в файле `image.cpp` должны быть определения всех методов. Скомпилируйте эту программу.

- **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
     ```

  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`).

- **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?

Задача 3. Движение по окружности

Напишите программу, которая будет рисовать на экране кружок, двигающийся по окружности. Используйте библиотеку SFML.

Задача 4. Задача n тел

В двумерном пространстве находятся n шариков с различными массами и различными электрическими зарядами. Напишите программу, которая будет моделировать движение таких шариков. Для отрисовки используйте библиотеку `SFML`.

- Шарики действуют друг на друга силой Кулона:

  $$F = \frac{q_1 \cdot q_2}{R}$$

  Где $q_{1,2}$ – заряды шариков, R – расстояние между шариками. Обратите внимание, что сила взаимодействия обратно пропорциональна первой степени расстояния, а не второй. Это правильная формула для силы Кулона в двух измерениях.

- Считайте, что силы гравитации между шариками пренебрежимо малы по сравнения с элекрическими силами. Их можно не учитывать.

- Столкновения шариков друг с другом можно тоже не учитывать. Шарики взаимодействуют только посредством силы Кулона.

- На границах окна поставьте стенки. Шарики должны упруго отскакивать от стенок.

- Расчёт ускорений, скоростей и положений всех шариков проводите с шагом Δt. В этой задаче можно считать, что Δt постоянна и равна $1/fps$. Где значение количества кадров в секунду (fps) задаётся с помощью метода `setFramerateLimit`.

- Если два шарика подойдут слишком близко друг к другу, то сила взаимодействия может стать очень большой. Это приведёт к большой погрешности в вычислениях из-за того, что Δt больше характерного времени изменения силы взаимодействия между шариками. В результате этого шарики начнут чрезмерно быстро двигаться. Чтобы исправить этот баг просто сделайте так, чтобы сила взаимодействия шариков была равна нулю, если расстояние между шариками меньше некоторой величины.

- Начальные значения масс, зарядов, положений шариков задайте случайным образом из некоторых диапазонов. Начальные значения скоростей равны нулю.

- Цвет шарика должен зависит от его заряда. Положительно заряженные шарики рисуйте красным цветом, а отрицательно зарженные – синим.

- (*) Добавьте возможно добавлять шарики, используя мышь. При нажатии левой кнопки мыши, в том месте, где находится курсор, должен создаваться шарик с маленькой массой и с отрицательным зарядом. При нажатии правой кнопки мыши должен создаваться шарик с очень большой массой и положительным зарядом.