UF UNIVERSITY of FLORIDA
Herbert Wertheim
College of Engineering

COP3504C

Lab 10: Meme Generator

Overview

In this lab, students will use an external library to create a meme generator library and executable. The purposes of this assignment is to give students practice with setting up, importing, using, and writing libraries in C++.

SFML Setup

In this assignment, students will import and use `SFML` (Simple and Fast Multimedia Library). This section describes how to install SFML and integrate it into a project.

Installation

1. Download **GCC 7.3.0 MinGW (SEH) - 64-bit**; decompress into reasonable path (e.g., `C:\Libraries`).
2. Add path (e.g., `C:\Libraries\SFML-2.5.1`) as variable `SFML_INSTALL` to system variables.
3. Add binary path (e.g., `C:\Libraries\SFML-2.5.1\bin`) to `PATH` system variable.

Integration

Under the compiler settings, add the following lines to your CMakeLists.txt to integrate SFML into the project:

```
set(SFML_DIR "C:/Libraries/SFML-2.5.1/lib/cmake/SFML")
find_package(SFML 2.5 COMPONENTS graphics audio REQUIRED)
```

You can add the library to your `memer` library target by specifying link instructions:

```
add_library(memer memer.cpp)
target_link_libraries(memer sfml-graphics sfml-audio)
```

Likewise, you can link your `memer` library to your `memeify` executable:

```
add_executable (memeify memeify.cpp)
target_link_libraries(memeify memer sfml-graphics sfml-audio)
```

Use

To use SFML, you simply include the appropriate header in your code and use SFML constructs in your project:

```
#include <SFML/Graphics.hpp>
```

The `Cave-Story.ttf` open-source font file has also been provided for this project.

Classes
There are a few important classes that you will want to read about in the SFML documentation.

`sf::Image`
This object is an image in system memory (RAM), stored as a series of pixels.

`sf::Texture`
This object represents a read-only version of image data pre-formatted and stored in video memory (VRAM).

`sf::Font`
A font for use in SFML routines. Typically loaded from a file.

`sf::String`
The SFML native String format.

`sf::Sprite`
A drawable class; it references a section of a texture that is used for display / drawing.

`sf::Text`
A drawable text element; incorporates a Font and a String. Positioning can be set as needed.

`sf::RenderTexture`
A read-write texture; data is stored in video memory. This object can be drawn on.

Specification
In this assignment, students will generate two artifacts, a memer library and a memeify executable.

Library
The library will be named **`memer`**. It should incorporate the function below and include **`memer.h`**:

```
sf::Image generateMeme(sf::Image base, sf::String topText, sf::String bottomText = "",
                      int topX = -1, int topY = -1, int bottomX = -1, int bottomY = -1)
```

Takes in an base to be used as the base image. Returns a new `sf::Image` with **`topText`** drawn over it at location (**`topX`**, **`topY`**) in the provided font. If no coordinates are provided, **`topText`** should be centered horizontally and be 1/3 from the top of the image. If it is provided, **`bottomText`** is drawn at location (**`bottomX`**, **`bottomY`**). If provided, the t**`bottomText`** should be placed 1/3 from the bottom of the image.

In general, adding text to an image will consist of the following steps:
1. Converting the **`Image`** into a **`Texture`**
2. Wrapping the **`Texture`** in a **`Sprite`**
3. Drawing the **`Sprite`** on a fresh & empty **`RenderTexture`**
4. Loading a **`Font`**, and using it to construct a **`Text`** element
5. Drawing the **`Text`** on the **`RenderTexture`**
6. Extracting an **`Image`** from a **`Texture`**, derived from the **`RenderTexture`**.

NOTE: graphics are traditionally done differently in 2D and 3D, resulting in the Image from a Texture being upside down; make sure to flip it horizontally before returning it!

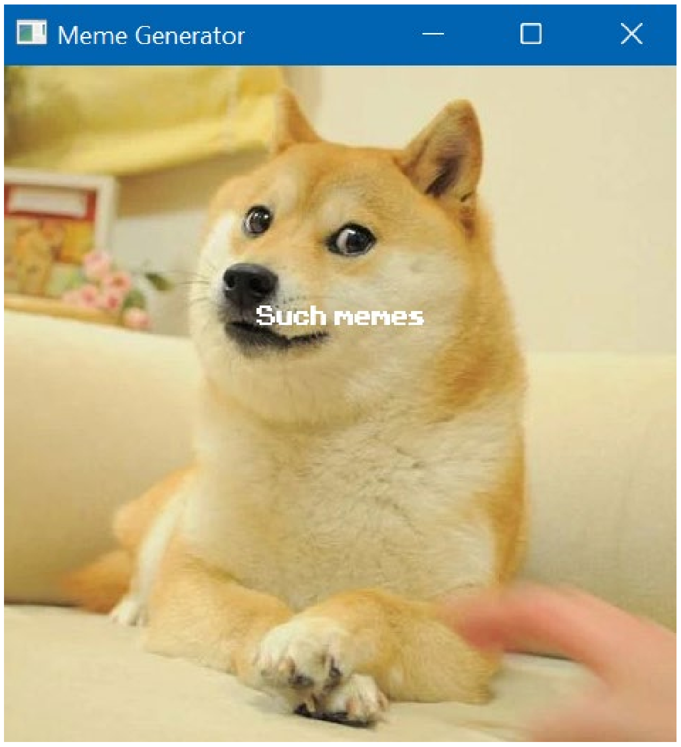
Executable

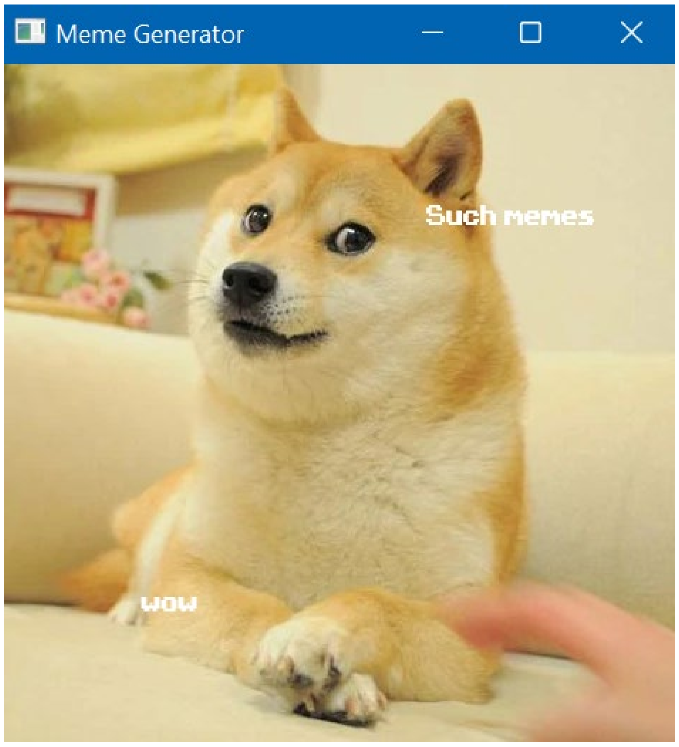
Executable should function with just file & top text:

```
finn@BMO:~$ ./memeify doge.jpg "Such memes"
```

… but should also accept a partial / full complement:

```
finn@BMO:~$ ./memeify doge.jpg "Such memes"\
> "wow" 360 90 120 360
```





The executable should 1) display the image in a window until the window is closed, and 2) save the image with a new name based on the old one in the form of **STEM-meme.EXT**; e.g., if the original image was "**doge.jpg**", the new image saved should be "**doge-meme.jpg**".

Submissions

NOTE: Your output must match the example output *exactly*. If it does not, ***you will not receive full credit for your submission***! (Note that matching sample output is necessary, but not sufficient, for full credit.)

Files: memeify.zip
Method: Submit on Canvas

Compressed Archive (memeify.zip)

We do not list required source files, only headers. You should include additional source or header files in addition to those listed – based on your design – but you must have the listed files at a minimum.

Your compressed file should have the following directory/file structure:

```
memeify.zip
    ↳ memeify (directory)
        ↳ CMakeLists.txt
        ↳ memer.h
        ↳ (Other sources / folders)
```