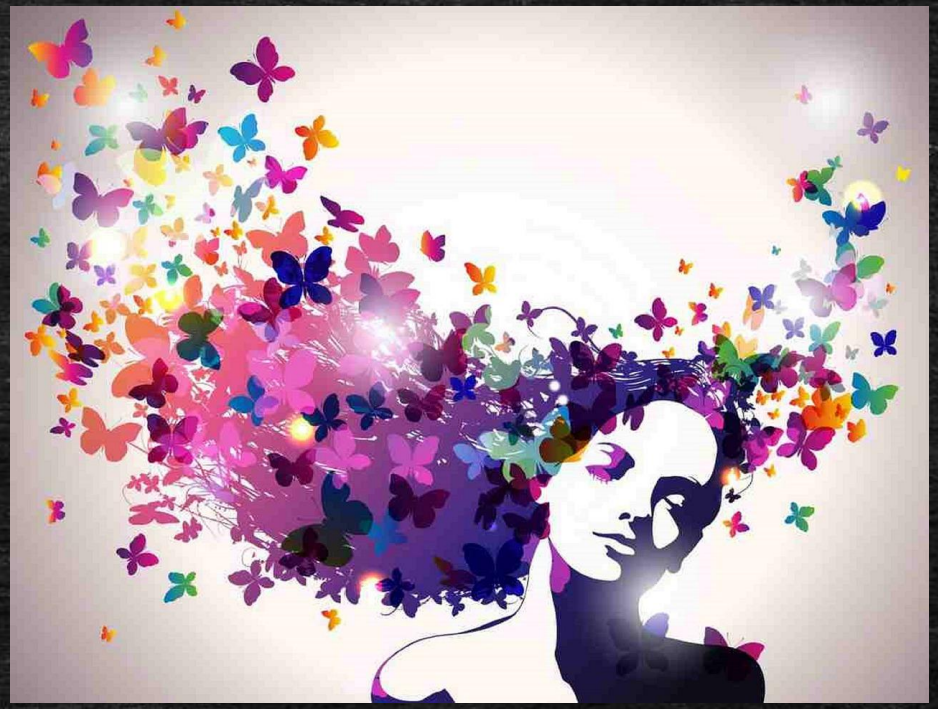
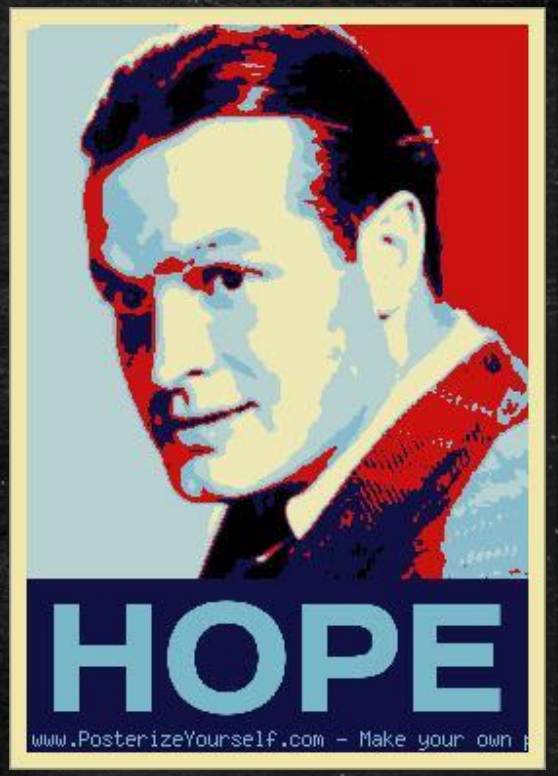
Pointers & Graphics



CS 150 – C++ Programming I
Lecture 18

Pointers and Graphics

- In this chapter we'll use a C graphics library (stb) to read and write images, using pointers to manipulate the image data
- Pointer Topics
  - 1. Pointers as output parameters (load)
  - 2. C functions and *string*
  - 3. Check for *nullptr* instead of *fail()*
  - 4. Address arithmetic instead of size
  - 5. Iterator loops using pointers *beg* and *end*
  - 6. Pointer increment and dereferencing
- Upload your own picture or use one of mine



Pointers as Output Parameters

- You load an image into memory with *stbi_load()*
  - `unsigned char * stbi_load(const char * filename, int * width, int * height, int * bpp, int channels);`
  - C-function - `filename` is a C-style string (`str.c_str()`)
  - Address output parameters (C has no references)
    - Pass the address; filled in in the function
- Returns a pointer to the image data on the heap if succeeds
  - *`unsigned char * const data = stbi_load(...`*
  - Returns *`nullptr`* if can't open file
  - Const pointer because we need to free the memory later

Processing Images with Pointers

- Create a pointer named *beg* initialized with *data*
- Create a second pointer named *end* using address arithmetic
  - If you add an integer (*n*) to an address you get a new address
  - Use *width * height * bits-per-pixel* for *n*
- Use pointer *beg* and an iterator loop to visit each element
  - Each pixel in the image contains 4 *unsigned char* (RGBA).
  - Pass each element to *poster()* function (skip alpha)
- Save the image with *stbi_write_png()*
- Free the data when done: *stbi_image_free()*

Pointers, Pixels & Structures

- Now, let's convert an image to its grayscale value
  - With `stbi_load()`, `data` points to a single `unsigned char`
  - Each pixel in the image has 4 of these (red, green, blue, alpha).
- Create a `Pixel` structure with four members
  - Create a new `Pixel` pointer named `beg`
  - Initialize using `reinterpret_cast` to "look at" `data` as `Pixel`
- Address arithmetic: pointer + n = new address
  - `n` expressed in element size (`Pixel` in our case)
  - Don't need the `bpp` to create a new pointer named `end`

Pointers and Structures

- Increment the `beg` pointer to point to the next `Pixel`
  - Type of the pointer determines how far the pointer moves
  - `unsigned char*` moves only one byte
  - `Pixel*` moves by `sizeof(Pixel)` or 4 bytes
- Access structure members using pointer-to-member
  - `beg->red`, not `*beg.red`
- Pass each `Pixel` to grayscale functions (by address & ref)
  - Write one using average, one with luminance
  - Manually change names of files so you can compare