Computer Science 312
Fall 2024
Assignment 6 – C++ Ray Tracer

Due: Saturday, 12/7/2024 11:59 p.m.

For this project, you will port your intermediate ray tracer to C++.

In Project 4, we divided the source code into separate source files (**`sphere.c`**, **`light.c`**, **`vec.c`**, and **`plane.c`**), with corresponding header files. Creating appropriate C++ classes from these source files should be relatively straightforward, though you will need to add other classes, **`Color`** and **`Object`**, to your program. Additionally, you must make most class data (and some class methods) private, taking advantage of operator overloading for vector and color functions, and implementing constructors, inheritance, and dynamic binding.

The following provides a breakdown of the C++ code organization:

`Color` : a new class replacing **`COLOR_T`**

`Color.hpp` – **`R`**, **`G`**, and **`B`** should be private
- two constructors (one receiving **`R`**, **`G`**, **`B`**; one with no parameters)
- **`inline get`** method for each **`R`**, **`G`**, and **`B`** (used **only** when writing to **`img.ppm`**)
- two overloaded **`add`**/**`operator+`** method pairs (one set for adding a **`Color`** and one for adding a **`double`**), all four returning **`Color`**
- overloaded **`mult_scalar`**/**`operator*`** method pair to multiply a **`Color`** times a **`double`**; returns **`Color`**
- **`void cap (void)`**
- don't forget header guards with new file names and to only include files needed (apply this hint to all **`.hpp`** files following)

`Color.cpp` – implementations of non-inline methods; include only necessary header files

`Light` : a new class replacing **`LIGHT_T`**

`Light.hpp` – **`loc`** and any static functions should be private
- include a constructor with one parameter
- one other public method

`Light.cpp` – implementation of constructor
- new interface: **`bool Light::shadow_test (SCENE_T scene, Object *obj, Vec int_pt)`**
  - use new **`Vec`** methods/operators to compute shadow ray
  - loop over **`scene.objs`** similarly to **`trace`**
- new interface: **`Color Light::illuminate (RAY_T ray, SCENE_T scene, Object *obj, Vec int_pt, Vec normal)`**
  - get geometry/color info from **`obj`**
  - use new methods/operators in **`Color`** and **`Vec`** to compute ambient, diffuse, specular

Main : files replacing **rt.h** and **rt.c**

rt.hpp – contains constants and unchanged **RAY_T**

scene.hpp – new header file containing modified **SCENE_T** with the following fields:
- **objs** linked list
- **start_x**, **start_y**, and **pix_size**, as before

Object.hpp – new parent class for **Sphere** and **Plane**; contains only public data/method
- three data items (**color**, **checker**, **color2**; don't forget C++ has **bool** type)
- **virtual intersect** method returning Boolean with parameter ray and reference parameters **t**, **int_pt**, and **normal**
  - default behavior is to return **false**
  - any derived class should override this method with its specialized version
- no corresponding **.cpp** file required
- **next** pointer

Main.cpp – replaces **rt.c**
- **static init** method with pointer to scene (and optional light double pointer) as parameters (returns nothing)
  - read objects from a file, as in Assignment 4, and link them into a linked list
  - set **light** using values from the file; call its constructor to set (optional; otherwise, call the constructor in main during declaration with hardcoded values)
  - set **start_x**, **start_y**, and **pix_size**, as before
- **static trace** method with **ray**, **light**, **scene** parameters (return unchanged)
  - initialize **Color** with constructor
  - loop through **objs** in **scene** linked list
  - use dynamic binding to check object intersect (each item in **objs** will automatically call the correct **intersect** method)
  - remember to update arguments to **intersect** and **illuminate** (now called through an object)
- **main** works as previously with just a few differences
  - you can keep your file I/O the same using C function calls
  - use new **Vec** methods/operators to set **ray**

Sphere : a new class replacing **SPHERE_T**

Sphere.hpp – geometry features should be private
- **Sphere** class declared, inheriting from **Object** (**public**)
- constructor with geometry, **color**, **checker**, and **color2** parameters (default any appropriate parameters to unused values and create objects in line, such as **Vec**, when calling constructor)
- overridden **intersect** method with appropriate reference parameters

Sphere.cpp – implementations of constructor and **intersect**
- constructor sets all five instance variables
- **intersect** should take full advantage of the new methods/operators in **Vec** (use overloaded operators whenever possible); place parentheses around user-defined operators if unsure about precedence

`Plane` : a new class replacing **`PLANE_T`**

`Plane.hpp` – geometry features should be private
- very similar to **`Sphere.hpp`**

`Plane.cpp` – implementations of constructor and **`intersect`**
- very similar to **`Sphere.cpp`**

`Vec` : a new class replacing **`VP_T`**

`Vec.hpp` – **`x`**, **`y`**, and **`z`** should be private; **NO `get` ACCESS methods allowed**
- two constructors (one receiving **`x`**, **`y`**, **`z`**; one inline with no parameters)
- **`set`** method
- **`normalize`** , **`dot`**, and **`len`** methods, as before, but using new methods/operators, if appropriate; also make **`dot`** and **`len`** inline
- new methods (each with one argument only, except where noted):
  - **`add`**/**`operator+`** pair (for adding two **`Vec`**'s)
  - **`sub`**/**`operator-`** pair (for subtracting the second **`Vec`** from the first)
  - **`mult`**/**`operator*`** pair (for multiplying two **`Vec`**'s, coordinate by coordinate)
  - **`scalar_mult`**/**`operator*`** pair (for multiplying a **`Vec`** by a double)
  - **`scalar_divide`**/**`operator/`** pair (for dividing a **`Vec`** by a double)
  - **`sum_components`** (inline, no arguments) to add up **`x`**, **`y`**, and **`z`** (think **`intersect`**)
  - **`sum_floor_components`** (no arguments) to sum the floors of **`x`**, **`y`**, and **`z`** (think checkerboard)

`Vec.cpp` – implementations of methods; use **`Vec`** methods/operators where possible

These detailed specifications are provided to help you complete the project. Do not add any other functions/methods/files to what is listed. I think you will find that your code appears shorter and simpler than before.

Your program must be able to produce the default image from Project 4. As always, be sure that your code is well-structured and commented. Include a creative scene file that can be rendered by your program for extra credit.

To compile your program:

```
g++ Color.cpp Light.cpp Main.cpp Plane.cpp Sphere.cpp Vec.cpp -o rt
```

Submit the following 14 files on Blackboard:

`Color.hpp, Color.cpp`
`Light.hpp, Light.cpp`
`rt.hpp`
`scene.hpp`
`Object.hpp`
`Main.cpp`
`Plane.hpp, Plane.cpp`
`Sphere.hpp, Sphere.cpp`
`Vec.hpp, Vec.cpp`
`Scene2.txt (optional extra credit creative scene file)`