Middle East Technical University Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2015-2016

Programming Assignment 3

Due date: 8 May 2016, Sunday, 23:55

1 Objectives

This assignment aims to help you get familiar with fundemantal C++ programming concepts.

Keywords: *C++, OOP, Operator overloading*

2 Problem Definition

In this assignment, you are going to provide an implementation for a basic system representing videos. The system and its interfaces are designed for you. Your task is to make it functional by impelementing provided methods, constructors, destructors and overloaded operators.

A video is composed of variable number of sequential frames. A frame, on the other hand is a rectangular still image coded using pixels, which is the basic unit of programmable color on a computer image. The color of each pixel may be different. In our system, it is represented by three component intensities such as red, green and blue. So, an `N` by `M` frame basically contains NxM different pixels. A video may contain any number of such (different) frames.

You will find detailed descriptions for each of these concepts below.

There are several methods in the system to create, mutate or compare these entities along with some utility functions. While implementing all of these according to specifications provided in the following sections you have to be extra careful about the points mentioned in Regulations.

3 Specifications

You are going to implement three C++ classes: `Video, Pixel` and `Frame`. For each class, we are providing you with a header file including the declarations for *public* methods and constructors. You are going to include these header files in your `.cpp` files and provide your constructor, destructor and method bodies in the `.cpp` files.

You are free to define as many *private* members, methods and constructors as you like in the header files as long as you conform with the specifications below. However, do **not** add, remove or modify the given *public* methods and constructors (doing so will result in severe grade deduction).

There will be **no** erroneous input. Therefore, you do not need to worry about that.

Descriptions for each one of the public constructors and methods for all three classes are given in their respective tables.

3.1 Rectangle Struct

The `Rect` struct will be used to access regions of interest inside frames. You should not modify this file - it is already implemented for you.

```
#ifndef rect_h
#define rect_h

struct Rect
{
    int x; // x coordinate of the top-left corner
    int y; // y coordinate of the top-left corner
    int w; // width
    int h; // height
    Rect(int inX, int inY, int inW, int inH) :
        x(inX), y(inY), w(inW), h(inH) { }
};

#endif
```

3.2 Pixel Class

The `Pixel` class is used for representing a single pixel which is composed of *red*, *green*, and *blue* components. As these are defined to be `unsigned char`, valid component values are integers in range[0, 255].

Implement all the public interface functions in a file called `pixel.cpp`. Do not delete anything from this file but if necessary you can add variables and/or functions in the private part.

```
#ifndef pixel_h
#define pixel_h

#include <ostream>

typedef unsigned char uchar;

class Pixel
{
    private:
        // add private members, methods and constructors here as you need
        uchar red, green, blue;
    public:
        // do not make any modifications below
        Pixel(uchar r = 0, uchar g = 0, uchar b = 0);
        uchar& operator[](int index);
        const uchar& operator[](int index) const;
        uchar getR() const;
        uchar getG() const;
        uchar getB() const;
        void setR(uchar r);
        void setG(uchar g);
        void setB(uchar b);
        friend std::ostream& operator<<(std::ostream& os, const Pixel& p);
};

#endif
```

Table 1: Pixel Class

Method	Explanation
`Pixel::Pixel(uchar, uchar, uchar)`	Constructor with three unsigned char arguments. Arguments are *red, green, blue* components of the pixel respectively.
`uchar& Pixel::operator[](int)`	Overloaded [] operator for updating color components. The argument is one of 0, 1, 2 with 0 representing red, 1 green, and 2 blue.
`const uchar& Pixel::operator[](int) const`	Again overloaded [] operator, however it can only access the values rather than changing them.
`uchar Pixel::getR()const`	Getter for the red component
`uchar Pixel::getG()const`	Getter for the green component
`uchar Pixel::getB()const`	Getter for the blue component
`void Pixel::setR(uchar r)`	Setter for the red component
`void Pixel::setG(uchar r)`	Setter for the green component
`void Pixel::setB(uchar r)`	Setter for the blue component
`friend std::ostream& operator<<(std::ostream↩ &, const Pixel&)`	Overloaded << operator used for printing the contents of a pixel. If the pixel colors are 32, 45, and 102, the output should be look like: **(32, 45, 102)**. Do not print a new line after closing paranthesis.

3.3 Frame Class

The `Frame` class is used for representing a frame (i.e. image) which is composed of width*height number of pixels *logically* laid out on a 2D grid. The term logically is used because even though the pixel data is stored in a single array, we assume that the first `width` pixels represent the first row of the frame, the pixel at position `width + 1` represents the first pixel in the second row of the frame and so on.

Implement all the public interface functions in a file called `frame.cpp`. Do not delete anything from this file but if necessary you can add variables and/or functions in the private part. You must ensure that all copies are deep copies.

```
#ifndef frame_h
#define frame_h

#include "pixel.h"
#include "rect.h"

class Frame
{
    private:
        // add private members, methods and constructors here as you need
        Pixel* data;        // pixel data
        int width, height; // dimensions of the frame
    public:
        // do not make any modifications below
        Frame();
        Frame(int width, int height);
        Frame(int width, int height, const Pixel& p);
        Frame(const Frame& rhs);
        ~Frame();
        Frame& operator=(const Frame& rhs);
        Pixel& operator()(int x, int y);
        const Pixel& operator()(int x, int y) const;
        Frame operator()(const Rect& rect) const;
        bool operator==(const Frame& rhs) const;
        bool operator!=(const Frame& rhs) const;
        int getWidth() const;
        int getHeight() const;
        void clear(const Pixel& p);
        void clear(const Rect& rect, const Pixel& p);
        void crop(const Rect& rect);
};

#endif
```

3.4 Video Class

The video class represents a video which is basically a sequence of frames.

Implement all the public interface functions in a file called `video.cpp`. Do not delete anything from this file but if necessary you can add variables and/or functions in the private part. You must ensure that all copies are deep copies.

```
#ifndef video_h
#define video_h

#include "frame.h"

class Video
{
    private:
        // add private members, methods and constructors here as you need
        Frame* frames; // frame data
        int nFrames;   // the number of frames in this video
    public:
        // do not make any modifications below
        Video();
        Video(const Video& rhs);
        ~Video();
        Video& operator=(const Video& rhs);
        Video& operator<<(const Frame& frame);
        Video& operator>>(Frame& frame);
        void resetStream();
        Frame getFrame(int frameIndex) const;
        void dropFrame(int frameIndex);
        void trim(int startIndex, int endIndex);
};

#endif
```

4 Sample Code

All header files with a sample main will be uploaded to COW.

Table 2: Frame Class

Method	Explanation
`Frame::Frame()`	Empty constructor, creates an empty frame (`data` = `NULL`) with zero dimensions.
`Frame::Frame(int, int)`	Constructor with the given dimensions. `data`↩ must be allocated but its contents are irrelevant.
`Frame::Frame(int, int, const Pixel&)`	Constructor with the given dimensions. Each pixel must be set to the given pixel.
`Frame::Frame(const Frame&)`	Copy constructor
`Frame::~Frame()`	Destructor
`Frame& Frame::operator=(const Frame&)`	Overloaded assignment operator. Note that it must return the reference of the current frame to allow chained assignments.
`Pixel& Frame::operator(int x, int y)`	Returns a reference of the pixel at position (x,y). Can be used to set the value of a pixel.
`const Pixel& Frame::operator(int x, int y) const`	Returns a constant reference of the pixel at position (x,y). Can be used to retrieve the value of a pixel.
`Frame Frame::operator(const Rect& rect)const`	Returns a sub-frame of the current frame. The sub-frame position and dimensions are specified in the `rect` parameter.
`bool Frame::operator==(const Frame& rhs)`↩ `const`	Returns true if the current frame is equal to the input frame `rhs`. Conditions for equality are: (1) their dimensions should match **and** (2) they must contain the same pixel data.
`bool Frame::operator!=(const Frame&)const`	The opposite of the equality function above.
`int Frame::getWidth()const`	Getter for width of the frame
`int Frame::getHeight()const`	Getter for height of the frame
`void Frame::clear(const Pixel&)`	Clears the entire frame to the given pixel's color values (Each pixel is updated with the given pixel's colors).
`void Frame::clear(const Rect&, const Pixel&)`	Clears the given sub-region inside the frame (Pixels within the given the given rectangle are updated with the given colors).
`void Frame::crop(const Rect&)`	Crops the current frame to the given rectangle.

5 Regulations

1. **Allowed Libraries:** You are not allowed to use Standard Template Library or any other external library.
2. **Memory Management:** Since you cannot use STL data structures, you have to handle memory management by yourself. While doing so, be careful about properly freeing all of the used heap memory in the class destructors. Any heap block, which is not freed at the end of the program will result in grade deduction. Please check your codes using `valgrind --leak-check=full` for memory-leaks.
3. **Programming Language:** You must code your program in C++. Your submission will be compiled with `g++` on department lab machines. You are expected make sure your code compiles

Table 3: Video Class

Method	Explanation
`Video::Video()`	Empty constructor that creates an empty video (`frames` = `NULL` and zero number of frames)
`Video::Video(const Video&)`	Copy constructor
`Video::~Video()`	Destructor
`Video& Video::operator=(const Video&)`	Assignment operator
`Video& Video::operator<<(const Frame&)`	Inserts the given frame at the end of this video. Can be used by chaining: `video << frame1 ↩ << frame2 << ...`
`Video& Video::operator>>(Frame&)`	Extracts a frame from the given video. Can be used by chaining: `video >> frame1 >> ↩ frame2 >> ....` Note that each application of this operator must return the "next" frame.
`void Video::resetStream()`	Resets the stream such that the first `operator↩ >>` after a call to this function will extract the first frame again.
`Frame Video::getFrame(int)const`	Retrieves the frame given by the frame index
`void Video::dropFrame(int)`	Removes the frame given by the frame index. After removal there should be no "gaps" in the video. That is, the following frames must be shifted to fill this gap.
`void Video::trim(int, int)`	Trims the video such that it contains only the frames between start index inclusive and end index (given as two `int`) exclusive.

successfully with `g++`.

4. **Late Submission:** A penalty of 5 * day * day is applied.

5. **Cheating:** Cheating will result in receiving 0 from all assignments and the university regulations will be applied.

6. **Newsgroup:** You must follow the CENG242 newsgroup for discussions and possible updates on a daily basis.

7. **Grading:** This homework will be graded out of 100.

6 Submission

Submission will be done via COW. Create a zip file named `hw3.zip` that contains all your source code files. Do not submit a file that contains a `main` function. Such a file will be provided and your code will be compiled with it. You are going to submit the following files:

1. `rect.h`

2. `pixel.h`

3. `pixel.cpp`

4. `frame.h`
5. `frame.cpp`
6. `video.h`
7. `video.cpp`

Note: The submitted zip file should not contain any directories! Your codes must successfully compile and run using the command sequence below::

```
$ g++ -pedantic -O3 frame.cpp video.cpp pixel.cpp main.cpp -o hw3
$ ./hw3
```