CS-224 Object Oriented Programming and Design Methodologies

Spring 2021

Homework 4, March 12, 2021

1 Guidelines

This is the last assignment in this course. The due date for submission is **Friday 2nd April, 11:59pm**. Late submissions are allowed till **Sunday 4th April, 11:59pm**, which will be penalized by 20%.

- You need to do this assignment in a group of two persons
- Only one member of the gorup will submit the assignment to LMS
- You need to follow the best programming practices as given in the accompanying document and it is also present on LMS. Failure in doing so will have your marks deducted.
- Submit assignment on time; late submissions will not be accepted.
- Some assignments will require you to submit multiple files. Always Zip and send them.
- It is better to submit incomplete assignment than none at all.
- It is better to submit the work that you have done yourself than what you have plagiarized.
- It is strongly advised that you start working on the assignment the day you get it. Assignments WILL take time.
- Every assignment you submit should be a single zipped file containing all the other files. Suppose your group member's name are Sara Khan, id 0022 and Ali Haider, id 033 so the name of the submitted file should be SaraKhan0022_AliHaider033.zip

- DO NOT send your assignment to your instructor, if you do I will just mark your assignment as ZERO for not following clear instructions.
- You can be called in for Viva for any assignment that you submit

2 HUMania++

Refer to HUMania game you developed in HW3, which had following requirements:

- Create a `Pigeon` class (see the pigeon.hpp/cpp), that will contain attributes and functions (`fly, draw`) related to a pigeon. The `fly` function flies the pigeon gradually to top-right side, and gets back the pigeon to left most corner as they approach at the right most border of the window. Three different images in assets file will be changed back and forth to make the pigeon fly. `draw` is only drawing the Pigeon object.
- Create an `Egg` class (create Egg.hpp and Egg.cpp), that will contain attributes and functions (`drop, draw`) related to egg. `drop` function makes the egg drop on the floor. Its shape changes to broken egg as it reaches to bottom of screen, and it doesn't move further down. `draw` function is only drawing the Egg object.
- Create a `Nest` class (create Nest.hpp and Nest.cpp), that will contain attributes and functions (`wiggle, draw`) related to Nest. `wiggle` is making a nest wiggle on the screen (without dropping down). Look at the three images in assets file to make a nest wiggle. `draw` function is only drawing the Nest object.
- As you click on the screen, one of the above objects is created randomly. You'll maintain three vectors (pigeons, eggs, nests) in `HUMania.hpp/cpp` to store objects of different classes. The object that you create on the click will be pushed into corresponding vector. Refer to `HUMania.cpp` ⇒ `creatObject()`, where you get mouse coordinates.
- Finally, you iterate over all the elements of vectors, and call their corresponding functions to make them animate(fly, drop or wiggle) and draw.

After learning inheritance, and pointers in the course you decided to improve this game as:

- Since `Pigeon, Egg and Nest` are some objects to be drawn on the screen, so a super class, call it `Unit`, can be implemented to take care of drawing aspects for all these objects. Hence the `draw` function and `srcRect, moverRect` can be defined in `Unit` class, and all the other classes will inherit from this class. *Define Unit.cpp/.hpp separately, also srcRect moverRect can be defined protected*
- Storing the objects in vector was a bad idea, because a copy is made to store in the vectors. Also, vectors usually occupy more space than static arrays, because more memory is allocated to handle future growth. Therefore, you decided to create objects dynamically and store only pointers to those objects in a linked list `std::list`.
- If an egg is dropped inside a nest, it hatches a new baby pigeon (a very small size e.g. 5x5, sprout from the same location of nest), that starts flying up-rightwards and increases its size gradually until it becomes adult pigeon.
- The hatched eggs will be destroyed.
- Pigeons which reach to right-most boundary of screen are also destroyed now.
- When certain object is destroyed, remove it from its particular list, and delete it from memory.

- When the game ends, all the objects created dynamically will be destroyed.

3 Binary Search Tree – Time24

Design a class `Time24`, which stores time in 24 hours format. Its constructor initializes the attributes: hours, minutes, seconds. The constructor also adjusts the attributes if any of them is out of range e.g. 65 seconds are adjusted as incrementing 1 in minutes, and setting seconds to 5. Minutes are adjusted as well, and over-flow in minutes are added to hours. Overflow in hours are simply discarded e.g. 26 hours will be adjusted by subtracting 24 from it, hence valid value would be 2.

You have to populate a Binary Search Tree (BST). The structure of a node in BST is defined as:

```
struct BSTNode{
   BSTNode* left;
   Time24 data;
   BSTNode* right;
};
```

You have to read input.txt file, in which every line contains a time value possibly in invalid format. As you read a time from file, you create a Time24 object and load it into the BST to its proper position. Since, the BST requires to compare ($<$) the data starting from its root node, for which you can define the $<$ operator in Time24 class, which compares two time objects with usual notion e.g. 13:20:30 $<$ 13:25:15.

Finally your program should print entire of BST in-order i.e. the time objects are printed in sorted order, and write the output to a file `output.txt`.

Example input and their expected output is given below:

```
Input:
10 20 30      // will be adjusted to   10:20:30
23 45 90      // will be adjusted to   23:46:30
14 55 600        // will be adjusted to  15:05:00
23 1006 300      // will be adjusted to  15:51:00
80 90 1000       // will be adjusted to  09:46:40
12 2000 300      // will be adjusted to  21:25:00



Output (In-Order from BST):
```

```
    09:46:40
    10:20:30
    15:05:00
    15:51:00
    21:25:00
    23:46:30
```

4 std::list demo

```
#include<list>
#include<iostream>

using namespace std;

class Distance{
   int feet, inches;
   public:
   Distance(int ft, int in): feet(ft), inches(in){}
   void show(){
      cout<<feet<<"'"<<inches<<'"'<<endl;
   }
};

int main(){
   list<Distance*> l1; // Create a list that can store Distance
       pointers
   l1.push_back(new Distance(5, 2)); // create an object
       dynamically, and store pointer to l1
   l1.push_back(new Distance(2, 2));
   l1.push_back(new Distance(1, 3));
   l1.push_back(new Distance(8, 5));
   l1.push_back(new Distance(3, 1));

   // It can't be indexed, as it's a linked list
   // We can use iterators to access every element in the list

   for(list<Distance*>::iterator it=l1.begin(); it!=l1.end(); it++)
      (*it)->show(); //*it returns you the pointers stored earlier

   // Let's delete the memory allocated dynamically
   for(list<Distance*>::iterator it=l1.begin(); it!=l1.end(); it++){
```

```
        delete *it; // Pointer is deleted, hence now it's invalid to
            use.
        *it = NULL; //always a good practice to assign NULL to
            invalid pointer.
    }

    // Clear all the contents of the list
    l1.clear();

    return 0;
}
```

5 Guidelines

- Sample code is there for your benefit. If you are going to use it, understand how it works.
- You do not need to follow the code given exactly. You can make changes where you see fit provided that it makes sense.
- Make the class declarations in hpp files, and provide function implementations in cpp files. Don't use hpp files for implementation purposes.
- You need to define separate `*.hpp` and `*.cpp` files for all the classes.
- A general rule of thumb is that all attributes in a class are private/protected.
- Where necessary, declare your own functions inside classes. Make sure why you would keep a function as private or public.
- Lazy Foo tutorial are helpful to learn working in SDL `https://lazyfoo.net/tutorials/SDL/index.php`.
- You should take `www.cplusplus.com` and `www.cppreference.com` as primary web source to search about C++.
- Complete reference for std::list can be found at: `https://en.cppreference.com/w/cpp/container/list`
- Refer to this link for BST `https://www.programiz.com/dsa/binary-search-tree`

- When you delete an object from within a list, it requires some consideration, refer to this document: `https://www.techiedelight.com/remove-elements-list-iterating-cpp/`

6 Rubrics

OOP Concepts	The code followed best OOP practices	2
Inheritance	Implemented inheritance as required	2
Dynamic Memory	Objects created dynamically, and are destroyed properly	2
Logic	Required functionality is fully implemented	4

Table 1: Grading Rubric