Creative Software Programming, Assignment 6-1

Handed out: Oct 13, 2021

Due: 23:59, Oct 19, 2021 **(NO SCORE for late submissions!)**

- Only files submitted by **git push to this course project at** https://hconnect.hanyang.ac.kr (<Year>_<Course no.>_<Class code>/<Year>_<Course no.>_<Student ID>.git) will be scored.
- Place your files under the directory structure **<Assignment name>/<Problem no.>/<your files>** just like the following example.
  - **DO NOT push CMake intermediate output files** (CMakeCache.txt, cmake_install.cmake, Makefile, CMakeFiles/*).
  - **DO NOT use cmake_minimum_required()** command in CMakeLists.txt.

```
+ 2020_ITE0000_2019000001
  + 2-1/
    + 1/
      - 1.cpp
      - CMakeLists.txt
    + 2/
      - 2.cpp
      - CMakeLists.txt
    + …
```

- The submission time is determined not when the commit is made **but when the git push is made**.
- Your files must be committed to the **master branch**. Otherwise, it will not be scored.
- Your program should output correct results even for inputs other than those used in the example.
- Basically, assignments are scored based on the output results. If it is not possible to check whether a requirement is implemented because the output is not correct, no score is given for the requirement, even if it is implemented internally. However, even if the output result is correct, no score is given for a requirement if the internal implementation does not satisfy the requirement.

1. Write a program for basic shapes.
    A. Define the class Circle and class Rectangle for circles and rectangles, respectively.
    B. Each class has member functions calculating the area and the perimeter of each shape.
    C. class Circle has the x, y coordinate values of the center point and the radius as member variables.
    D. class Rectangle has the x, y coordinate values of the top-left and bottom-right corners as member variables.
    E. Repeatedly print out the information of the shapes created according to user input
    F. Use 3.14 for pi.
    G. This program should take user input repeatedly
    H.
        i. 'C', x coordinate of center, y coordinate of center, radius - Create a circle.
        ii. 'R', x coordinate of top left corner, y coordinate of top left corner, x coordinate of bottom right corner, and y coordinate of bottom right corner - Create a rectangle
        iii. 'Q' - Quit the program
        iv. Assume that all inputs are valid. You do not need to handle wrong inputs.
    I. Output: The area and perimeter of the shape
    J. Files to submit:
        i. main.cpp – main() must be in this file.
        ii. shapes.h – Class definitions
        iii. shapes.cpp – Class member function definitions (implementations)
        iv. A CMakeLists.txt to generate the executable

```
$ ./simple_shape

shape?
C 1 1 1
area: 3.14, perimeter: 6.28

shape?
R 1 5 5 1
area: 16, perimeter: 16
```

```
shape?
Q
$
```

2. Write a program that counts adjacent mines of each position from a map. (minesweeper)
   A. Click HERE to download minesweeper.h

      i.
      ii. If the download fails, check our website (where you downloaded this document).
   B. Implement the given class "Minesweeper" and use it.
      i. Keep the interface while additional member variables or functions are allowed.
   C. In a map (N*M matrix), ' * ' represents a mine and ' . ' is a blank space.
   D. Calculate another map that represents how many adjacent mines exist from each position.
   E. After setting the adjacency map, allow the user to **toggle** a mine in a specific location.
      i. Recalculate the adjacency map whenever a toggle happens.
   F. Input: Commands that start with ':', a map, toggle positions
   G. Output: A mine adjacency map
   H. Files to submit:
      i. minesweeper_main.cc, minesweeper.h, minesweeper.cc
      ii. A CMakeLists.txt to generate the executable

```
$ ./minesweeper
:set 5 4
*....
....*
.*...
.....
```

```
*1011
2211*
1*111
11100
:toggle 0 0
00011
1111*
1*111
11100
:toggle 3 2
00011
1122*
1*2*2
11211
:set 5 3
.....
.....
.....
00000
00000
00000
:quit
$
```

3. Extend the program to a minesweeper game.

   A. Set the initial map by "**:set**" and start the game by "**:play**".

   B. Every position is represented as ' _ ' right after the **":play"**, and once started, the user cannot **":toggle"**.

      i. When **":set"** is called again(a new game), **":toggle"** is available.

   C. Allow the user to **":touch"** a position.

      i. **":touch # #"** is available after **":play"**.

      ii. If there is no mine in the position, print out a number.

         1. The user can re-touch the position to get the same output.

      iii. If there is a mine in the position, alarm "DEAD" and how many touches the user called so far.

      iv. The user can either **":play"** again with the same map or re- **":set"** from a new map.

      v. The original minesweeper shows the entire map when 0 is given as an input, but DO NOT implement it in this assignment.

      vi. The original minesweeper also terminates after opening(touching) every possible position, but DO NOT implement it in this assignment.

```
./minesweeper
:set 5 4
*....
....*
.*...
.....
*1011
2211*
1*111
11100
:play
_____
_____
```

```
_____
_____
:touch 1 1
_____
_2___
_____
_____
:touch 4 1
DEAD(2)
:quit
```