1: Using One Class: Dice

CSCI 6626 / 4526 Spring 2024

1 Goals

- To build the first part of the term project.
- To build a simple class with all the usual parts.
- To use constructors and destructors.
- To attach your class to the predefined output operator.
- To make a code module comprised of matching .hpp and .cpp files.
- To build a unit-test function that tests all parts of your class.
- To build a main program that calls the unit-test function.
- To follow the coding standards listed in the Introduction of this Canvas site.

Note 1: If you do not understand something on this list, email me. I expect your submission to achieve all of these goals.

Note 2: This assignment provides a lot of guidance and tells you the details of what I expect you to do. As the term goes on, less and less guidance will be provided in the assignments.

2 Dice and Games

Dice. Many games are played with a pair of 6-sided dice. However, dice come in other shapes: 4-sided, 12-sided, and 20-sided dice certainly exist. Also, some games use 1 die, others use more than 2 dice. In this program, you will create a Dice class that can have any reasonable number >= 1 of 6-sided dice. The class members you need are described below.

3 Instructions

In every module, the header file contains variable and function declarations. The data members are always private. The function members are usually public. Most modules also have an implementation (.cpp) file that contains the function definitions.

3.1 The header: Dice.hpp

- Declare a private variable to store *nDice*, the number of dice in the set.
- Declare an *int* pointer variable for the dynamically allocated array of pseudo-random values.
- Declare one constructor *Dice(int n)*. Give your constructor a default value for its argument. This lets you construct dice with or without supplying an argument in the call.
- Declare a *public ostream& print(ostream&)* function
- Declare a function *public const int* roll()*. Because of the *const*, the calling program will have read-only access to the array; changing it will not be possible.

Outside the class, at the end of the .hpp file, declare a method for the output operator,

inline ostream& operator <<(ostream&, Dice&)

It must call your *print()* function with the stream parameter and return the ostream& as the result.

3.2 The implementation: Dice.cpp

Implement all the class functions here.

- Define your constructor. It must store its argument in the corresponding variable, *nDice*, then dynamically allocate an array of *nDice* integers, There is no need to initialize those integers. Finally, call *srand(time(NULL))* to initialize the random number generator.
- Define a destructor that will free the dynamically allocated array.
- Define a public function *const int* roll()*. Fill the dice-value array with *nDice* random values. Return the array of dice-values as a *const int**. Because of the *const*, the calling program will have read-only access to the array; changing it will not be possible.
- Define a public function *ostream& print(ostream&)* that will print the array of dice-values on one line, separated by single spaces. Do not include newlines as part of the output. Newlines are normally supplied by the method that calls print() because it may or may not be desirable in a given context.

4 Testing

Create a test plan – that is, a list of cases that must be tested. For each case, say what input you need and what output or program behavior should be produced. Incorporate this test plan in a function called `unitDice()`, which will be called from `main()`. Use comments to explain the tests from the test plan. It is important to test every part of this program, and to cause every possible error comment to appear in your output.

Open a text file in append mode and check for opening errors. Send your test output to this file. (You may print some or all of it on the screen, but all of the test output should be written to the test-output file.) Learn how to do this now, because screen shots are not adequate for turning in this much test output. If you do this right, all your test outputs will be in one .txt file that you can submit.

Test your program with different values of nDice. Generate enough random values to demonstrate that every value in the range 1..6 can be produced by your dice.

Due: February 6 Put copies of your test plan and your output in the same folder as your source code. The name of the folder should include both the problem number and your name. (Example: P1-Fischer). Zip up your directory and email it to my home.

5 Advice

This is a short, easy program; don't complicate it. PLEASE follow the instructions. The highest grades will be for the simplest programs that follow the specifications. Email for help promptly if you have any trouble of any sort.