Assignment 04

CSE 241

June 2, 2022

1 Description

You are to make a program which maintains a vector (dynamic array) of different types of random number generators. You should use `std::vector` from the STL for your vector.

Your program should present the user with a menu, printed to standard output. The menu should present as:

```
(0) Quit
(1) Add a die
(2) Add a die that cheats
(3) Add a die that cycles 0, 2, 4, 1, 3, 5, ...
(4) Add a sequence of fractions 1, 1/2, 1/3, 1/4, 1/5, 1/6, 1/7, 1/8, 1, 1/2, ...
(5) Add a mixture of the last two generators added
(6) Generate random numbers
Enter your choice:
```

Option 5 should be presented to the user *only* if the vector already has at least 2 elements in it.

You will repeatedly present the user with the menu and respond to their requests until they enter in "0" to quit. You do not have to consider bad input.

For options 1 and 2, you should ask "How many sides? " and read in an int. For option 2, you should additionally ask "Which side should be chosen less often (0-n)? " where n is 1 less than the number of sides of the die.

For option 6, you should ask "Between what and what? " and read in 2 doubles. You may assume that the second double the user enters is larger than the first double. You will then produce random numbers from *all* random number generators currently in the vector, and print out the random values generated, all on one line, with spaces between them. The numbers should not be formatted specially.

2 Random number generators

2.1 class random_gen

Make a `random_gen` class with a `virtual double between(double, double) = 0;` method, which will be used to randomly generate methods. All other classes are descended from this class.

2.2 class die

Represents a fair die with a given number of sides. You may assume that there are at least 2 sides.

Make a protected `roll` method like so:

```
1 virtual int roll() const {
2     return rand() % this->num_sides;
3 }
```

Call that method when writing the `between` method.

You must include `cstdlib` to make use of `rand()`.

2.3 class biased_die

Represents a biased (unfair) die. Like a die, it has a fixed, finite number of sides. Unlike a fair die, there is one side which will be chosen less often.

Let b be the side that is to be chosen less often. Roll as usual. If b is not rolled, then return that value immediately. If b *is* rolled, then roll again. If b is rolled a second time, then you can return b. The ultimate effect is that b will be returned less often than the other sides.

2.4 Template class predetermined

Make a templated class `predetermined`. Unlike the other classes, this one is *not* random.

Instead, it has an array of elements (of some numeric type). Each time `between` is called, it uses the next element in the array to determine the value returned. Once all elements of the array have been exhausted, it cycles back to the beginning of the array.

For example, suppose that we instantiate the `predetermined` class with typename `int` and use the array `{ 5, 6, 10 }` to produce values.

If we called `between(0.0, 1.0)` 7 times, we would get the values 0.0, 0.2, 1.0, 0.0, 0.2, 1.0, 0.0 in sequence.

As another example, if we called `between(-2.0, 2.0)` 4 times, we would get -2.0, -1.2, 2.0, -2.0.

Note that all of the values that are returned are scaled proportionately between the smallest and largest values in the array.

This class must work regardless of the type of the elements in the array. You may only assume that the elements of the array of a numeric type.

2.5 class mix

This represents a random number generator that mixes together two other random number generators. It mixes together their two values in a 50-50 split.

It must have two attributes, both of type `random_gen` .

It does not do any heap allocations or deallocations, as it is not responsible for managing the memory of its constituent objects.

For example, if we call `between(0.0, 1.0)`, it will call `between(0.0, 0.5)` on each of its constituent random number generators and add the two together.

3 Functional requirements

You must have at least 3 files.

main.cpp — contains the `main()` function and does *not* contain any classes

random-gen.h — contains the class descriptions, method prototypes, method "one-liners", and any templated code

random-gen.cpp — contains the implementations of larger class methods

Other functions are methods may be added, as you wish.

Begin your `main` function with:

```cpp
#include <cstdlib>

int main(int argc, char **argv) {
    if (argc == 2)
        srand(strtoul(argv[1], nullptr, 0));
    else
        srand(time(nullptr));
    // ... the rest of your code after this
}
```

This will ensure that you get different random numbers each time you run the program. If you would like to get the same random numbers each time you run the program (for debugging or testing purposes), you can provide an integer as a command-line argument.