Project in C++ OOP

Hergeir Winther Lognberg
Hewi1600

1 Preamble

Assignment was to create a dynamic array class which acted like a queue and held the array using a smart pointer.

2 The Code

2.1 placement

I've decided to keep all files associated with the lab in the root of the project folder. There are in all 12 files 2 for each class and 2 for often used functions.

2.2 code

I chose to implement the class as it was put up to in the Lab. I see that many implemented the array with the type:

```
1 std::unique_ptr<unique_ptr<int>[10]> arr;
```

I didn't see any reason (nor gain) for adding smartpointers for all elements in the array, when we are using a simple standard type, such as int which cleans itself when it goes out of scope. And since the Lab didn't specify this as a requirement I chose simplicity.

2.3 Question

Are we still obligated to comply with the c++11 standard? Using:

```
1 smartPtr = std::make_unique<int[10]>();
```

Breaks that requirement as the feature was not introduced until c++14.

Because I've been told to comply with c++11 I've chosen to make the constructor as such.

```
void TestApp::createQueue()
{
   int size;
   do
   {
      getInt(size,"enter queue size (must be more than 1): ");
   } while (size<1);     //a size under 1 is illegal and doesn't make sense (defensive programming)

   std::unique_ptr <Queue> temp(new Queue(size));
   queue = std::move(temp);
}
```

I create a temporary smartpointer containing a pointer to the new dynamically created int array and move It into our private class variable smartpointer queue. The temporary smartpointer deletes itself as soon as it leaves scope and thereby leaks no memory.

In Queue class I also avoided 'make unique' by using initializer list for getting size of the dynamic int array as follows.

```
Queue::Queue(size_t size)
:queue(new int[size])
{
   maxElem=size;
   head=0;   // first element in array.
   tail=-1; // starting with -1 removes a special case in enqueue method.
   nElem=0; // nElem keeps track of amount of integers in array.
}
```

This upholds the requirements as Queue is not supposed to be able to change size after initialization, but to be initialized with different sizes. Also It's dynamic as it only initializes the array after runtime using the new operator.

2.4 modulo vs if

I believe the modulo operator is implemented by gcc as some kind of bitwise operations (magic) which probably is more effecient than what i'd do (if

overloading basic operators was allowed without user defined types.)

```
//modulo
int mod(int a, int b)
{
    int operator % (const & int a, const & int b)
    {
        return a-(a/b)*b;
    }
}
```

The text asks wether we believe the modulo operator to be faster than the if statement in enqueue function. I didn't run any test, but if they're using something like my definition for modulo the if statement should be faster than the alternative (especially considering branching). All the computer needs to do is pre-increment one variable and compare it with another (essentially 2 fast operations) (*tail* and *maxElem*). Like this:

```
void Queue::enqueue(const Type elem)
{
   if (!full()) //if queue is not full
   {
      if((++tail)<maxElem)
      {
         //already incremented tail in if statement
      }
      else
      {
         tail=0;
      }
      queue[tail]=elem; //inserting element in back of queue.
      ++nElem;          //incrementing number of elements.
   }
}
```

3 Building/Compiling

Just run *make* in the Lab directory. To run the program run *make run* in same directory.

4 Enviroment

I'm programming on an Arch linux 64-bit system. I've got the gcc compiler installed and compile using it's g++ alias which links necessary libraries automatically. To compile I use the recommended flags: "-std=c++11 -Wall -pedantic". The flags let me choose to use c++11 standard and give me useful compiling warnings and errors. For editing of code i use VS code.

5 Backup

And if anything's missing you can find it on:
github: `https://github.com/Hergeirs/Cpp-Obj/tree/master/Skei2`
Cpp-obj/Lab1

September 10, 2017