Imperial College London

Software Engineering 2: Object Oriented Software Engineering

Week 4 - Objects and Dynamic Memory – Lab*

Sahbi Ben Ismail (s.ben-ismail@imperial.ac.uk)

Static and dynamic `point` objects in `vector`

Add a destructor to our `point` class (printing the state of the object on the standard output).

Write a program which reads from a file a sequence of points (around 20) and stores them into:

- A `vector` of `point` objects. Remember that there are two ways of doing this (with or without temporaries). Try both.
- A `vector` of pointers to `point` (i.e. `point*`), each pointing to a point dynamically constructed using `new`. Call `origin_symmetric()` on each of these points right after they are stored in the vector.

After each `push_back` print size and capacity of the `vector`.

Observe what happens in terms of calls to the destructor both during the program execution (as points are read and stored) and at the end.

You should keep the habit of using `delete` (at the right time) on any memory allocated with `new`, in this exercises try also omitting it.

Try also the following template:

```
int main(){
    {
```

*Lab content originally written by Max Cattafi (first exercise).

```
    // (this is not a typo)

    // write all the rest of the main here

    }
    // (this is not a typo either)

    cout << "end of inner scope" << endl;
    return 0;
}
```

`polynomial` class

Using dynamic arrays, implement a `polynomial` class representing a polynomial of a `degree` n:

$$P(x) = \sum_{k=0}^{k=n} a_k x^k = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$$

One simple way to implement the `polynomial` class is to use an array of `doubles` to store the coefficients $(a_k)_{k=0}^{k=n}$. The index of the array is the exponent of the corresponding term. If a term is missing, then it simply has a zero coefficient[1].

Provide a default constructor, a copy constructor, and a parameterised constructor that enables an arbitrary `polynomial` to be constructed.

Supply an overloaded operator = and a destructor.

Supply a function `at(int i)` to get the value of the coefficient at index `i` (and potentially overwrite it).

Supply a function `evaluate(double d)` to evaluate the `polynomial` at a value d of type `double`.

[1] There are techniques for representing `polynomials` of high degree with many missing terms. These use so-called sparse matrix techniques, don't use them.

Keep in mind dynamic allocation of arrays:

```
double* foo = new double[20];
foo[2] = 2.5; // equivalent to *(foo+2) = 2.5;
delete[] foo;
```

Write a main testing all the functions, including the copy constructor and the assignment operator. To test the constructor for instance, the program could read several coefficients from the user. `at()` and `degree` can be used in a loop to print all the coefficients.

Optional: provide these operations:

`polynomial` + `polynomial`, constant + `polynomial`, `polynomial` + constant
`polynomial` - `polynomial`, constant - `polynomial`, `polynomial` - constant
`polynomial` `polynomial`, constant `polynomial`, `polynomial` constant

`Makefile` with implicit rules and variables

Update your `Makefile` using implicit rules and variables using the template provided on Blackboard.