Tartalomjegyzék

Exercises

Open a new project for each exercise or a new file if you're in the command line.

From this point onward use .cpp file extensions!

Complex class now with operators

Modify the lecture's "http://wiki.math.bme.huComplex"http://wiki.math.bme.hu code so it uses operator+, operator*. Also make sure we can print a complex using cout.

As a reminder:

```
Complex operator+(Complex other);
friend ostream& operator<<(ostream& os, Complex& c);
```

```
#include<iostream>
#include<cmath>

using namespace std;


class Complex {
private:
  float re;
  float im;
public:
  Complex();
  Complex(const Complex& other);
  Complex(float r);
  Complex(float r, float i);

  Complex add(Complex other);
  Complex times(Complex other);
  float abs();

  void print();

  ~Complex();
};

Complex::Complex() {
  re = 0;
  im = 0;
}

Complex::Complex(const Complex& other) {
  re = other.re;
  im = other.im;
}
```

```
Complex::Complex(float r) {
  re = r;
  im = 0;
}

Complex::Complex(float r, float i) {
  re = r;
  im = i;
}

Complex Complex::add(Complex other) {
  return Complex(this->re + other.re, this->im + other.im);
}

Complex Complex::times(Complex other) {
  float real = this->re * other.re - this->im * other.im;
  float imag = this->re * other.im + this->im * other.re;
  return Complex(real, imag);
}

float Complex::abs() {
  return sqrt(this->re * this->re + this->im * this->im);
}

void Complex::print() {
  cout << re << " + " << im << "i" << endl;
}

Complex::~Complex() {
}

int main(void) {
  Complex a;
  Complex b = Complex(1,2);
  Complex c = a.times(b);

  a.print();
  b.print();
  c.print();

  (b.add(c)).print();

  cout << b.abs() << endl;

  return 0;
}
```

String with operators

Extend the previous practical's String class with the operator+, which concatenates the strings. Also operator= which can copy a string into an existing string. Make it work with C strings as well. For example this should work:

```
String s;
s = "batman";
```

Also make the strings printable through cout.

Dictionary

Write a class called Dictionary that implements python's dictionaries partially.

The keys should be strings. The values integers. This is an example of a dynamically created 2 dimensional char array:

```
char** keys = new char*[100];
for(int i = 0; i < 100; i++) {
  keys[i] = new char[20];
}
```

It's fine if the keys have an upper character limit, but try to make it truly dynamic. We should be able to add as many elements as possible.

An example for the operator[]:

```
int& operator[](int index);
```

In our case the parameter would be a char* (C string).

It's important to return a reference for operator[], it makes it possible to use the return values as a left value, for example:

```
dict["batman"] = 1;
```

The following are needed:

- Default constructor: an empty dictionary
- operator[]: make sure we can add new values with this (make it work as a right and a left value as well). If we call it with a key that doesn't have a value yet, we should create that key/value pair.
- Make it cout friendly. When we print a dictionary we should see the whole content of it (all pairs). The formatting doesn't matter.
- remove(char*): removes the given key's key/value pair.
- Destructor

You can do this with a dynamically expanding array or you can use a linked list to store the pairs as well.