Final Review -- PART I

Exam

Exam

Time:

- 8/6 14:00 - 15:45

Location:

- F115 & F105

Exam

Part A:

- 25 minutes
- Closed book.

Part B:

- 75 minutes
- Allowed documents:
  - Slides
  - Basic shapes classes, including detailed implementation of move, zoom, rotate, ...
  - OpenGL functions allowing to run the program
  - Makefile/CMakeList.txt similar to the one for p3

C++ Basics

Similarity with C

- Everything from C is valid
  - conditional statement
  - function declaration
  - logical operators

New things

- New data type: `bool`
- namespace
- New libraries
  - iostream: for cin, cout
  - fstream: for file io
  - sstream: to parse strings
  - string: deal with real string instead of c-style char array
  - various containers
- Default arguments for function
- Function and operator overloading

I/O streams

What are streams?

```
Sequence of data that can be accessed sequentially.
```

Extraction operator **>>**: used to remove/get values from the input stream

Insertion operator **<<**: used to put values into the output stream

Stream in c++ is **uni-directional**: If you want to read and write data to the same file or device, you need two streams.

Standard streams

```
#include <iostream>
```

cin: an input stream binded with the standard input (something typed from the keyboard, etc)

cout: an output stream binded with standard output

```
cin >> a;
/*
extract data from standard input stream to a
cin knows how to convert the characters
you type into specific data types
*/
cout << 42;
/*
convert 42 to char and insert it into the output stream

after the program ends, the contents in output stream
will be printed to the terminal
*/
```

File streams

```
#include <fstream>
```

Declare input/output file streams

```
ifstream input(filename);
// connect input file stream `input` to a file `filename`
ofstream output(filename);
```

Read contents from ifstream

```
int a;
input >> a; // extract an int from the ifstream
string str;
getline(input,str); // obtain a whole line
```

Insert contents into the ofstream

```
output << contents;
```

File streams

Other functions

```
input.get(); // get a char from the ifstream
input.seekg(0);
/* move the position back to
the beginning of the file, like `rewind`
*/
```

Reset the state of the fstream: some operation may make the file stream into bad or eof state, which may preventing the normal operations like `seekg`, `tellg`, `getline`, etc.

```
if(input.fail())
    input.clear(); // reset the bad/error state flags
```

String stream

Parse a string: split a space separated string into desired types

```
#include <sstream>
```

Declare input/output string streams

```
istringstream iStream; // declare an input string stream
iStream.str(s); // initialize iStream with string `s`
ostringstream oStream;
```

Example:

```
istringstream iStream;
string s = "love vg101";
iStream.str(s);
string word1, word2;
iStream >> word1 >> word2;
// word1 is "love", word2 is "vg101"
cout << word1 << word2 << endl;
```

Dynamic memory allocation

Similar to malloc/calloc/free

- Single object:

```
int* a = new int;
delete a;
```

- Array:

```
int* A=new int[n];
delete[] A;
```

- 2D Array:

```
int** A=new int*[n];
for(int i=0;i<n;++i)
    A[i]=new int[n];

for(int i=0;i<n;++i)
    delete[] A[i];
delete[] A;
```

Function default argument

Default argument: default value provided for a function parameter

```
int add(int x=5, int y=6);

int main()
{
    // case 1
    add(); // perform 5+6, return value is 11
    // case 2
    add(1); // perform 1+6
    // case 3: normal case
    add(1,2); //perform 1+2, return value is 3
    return 0;
}

int add(int x, int y)
{
    return x+y;
}
```

Function overloading

Define two different functions with exactly the same name, but different argument count and/or argument types.
Example:

```
int add(int x, int y);  // func1
int add(int x, int y, int z);  // func2
double add(double x, double y);  //func3

add(2, 3); // it will call func1
add(1, 2, 3); // it will call func2
add(1.0, 2.5); // it will call func3
```

Compiler tells which function to call based on the actual argument count and types.

Operator overloading

Redefine the meaning of the operators to apply to new data types.

Operators that are member functions is bound to the first operand.

```
class Complex{
private:
    double real,comp;
public:
    Complex(double r, double c):real(r),comp(c) {};

    Complex operator+(Complex oprand2)
    {
        // return *this + oprand2
        Complex result(0,0);
        result.real=this->real+oprand2.real;
        // *this is oprand1
        result.comp=this->comp+oprand2.comp;
        return result;
    }
.....
```

Operator overloading

```
// continue of the class Complex
    Complex &operator*=(float k)
    {
        this->real *= k; this->comp *= k;
        return *this;
    } // why Complex& ?
};
```

Overloading `<<` : non-member function

```
ostream &operator<<(ostream &os, Complex a)
{
    os << a.getReal() << "+" << a.getComp()
    << "i" << endl; // why a.real is not valid?
    return os;
}
```

```
Complex op1(2,3), op2(5,6);
op1 *= 8;
Complex r = op1 + op2;
cout << r;
```