UEE1303(1070) S12: Object-Oriented Programming

Advanced Input and Output

What you will learn from Lab 8

In this laboratory, you will learn about file stream and string stream.

TASK 8-1 MEMBER FUNCTIONS OF ISTREAM

✓ Please compiler and execute the program `lab8-1`

```
// lab8-1.cpp
#include <iostream>
using std::cout;  using std::cin;
using std::endl;

int main()
{
    char c;
    char str[100];

    // first example: getline()
    cout << "Enter a sentence: " << endl;
    cin.getline(str,100,'\n');
    cout << "Tne sentence you enter is: " << str << endl;

    // second example: get()
    cout << "Enter a character: " << endl;
    cin.get(c);
    cout << "The character you type is: " << c << endl;

    return 0;
}
```

- In the first example, if you enter less than 100 characters, `getline()` will insert a NULL character to end this string.
- In the second example, if you enter more than one character, `get()` will remain the first character for you.

TASK 8-2 STRING STREAM

✓ String stream provides an interface to manipulate strings. Remember to include header `#include<sstream>`

```
// lab8-2-1.cpp
#include <iostream>
```

```
#include <sstream>

using std::cout;          using std::endl;
using std::ostringstream;

int main()
{
    int i = 1024;
    double d = 3.14159;

    ostringstream message;
    message << "i = " << i << " d = " << d << endl;

    cout << message << endl;
    cout << message.str() << endl;

    return 0;
}
```

- The line "`cout << message.str() << endl;`" is equal to "`string msg = message.str()`" and `cout << msg << endl;`
- Use member function `str()` to convert an `ostringstream` to a `string` object.

✓ Program `8-2-2` is an example to use `istringstream`.

```
// lab8-2-2.cpp
#include <iostream>
#include <sstream>
using std::cout;          using std::endl;
using std::istringstream; using std::string;

int main()
{
    string s1 = "value 6";
    string s2;

    istringstream format_str(s1);
    int a;

    format_str >> s2 >> a;
    cout << a << endl;


    return 0;
}
```

- You can use `string` object to initialize an `istringstream`.

TASK 8-3 BINARY FILE

✓ In this lab, you will learn how to write and read the binary file.

```
// lab8-3-1.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream out("out.dat", ios::binary);
    for ( int i = 0 ; i < 10 ; ++i) {
        out.write((char*)&i,sizeof(i));
    }
    out.close();
    return 0;
}
```

- Use member function `write( (char*)&var , sizeof(var) )` to write a binary file.

✓

```
// lab8-3-2.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    int num = 0;
    ifstream in("out.dat");
    for ( int i = 0 ; i < 10 ; ++i) {
        in.read((char*)&num,sizeof(i));
        cout << num << endl;
    }
    in.close();
    return 0;
}
```

- Use member function `read( (char*)&var , sizeof(var) )` to read a binary file.

TASK 8-4 EXERCISE

1. DATA ANALYSIS

✓ Please create a text file named "`ex8-1-1.txt`"

```
Elena Crown Mail Bow Treads
Linden Scourge Pike Ring Helmet Braid Treads
Mark Amulet Halberd Goldskin Ring Ring
```

✓ Please create another text file named "ex8-1-2.txt"

```
Item Crown {
    Armor:54-71
}
Item Helmet {
    Armor:72-89
}
Item Mail {
    Armor:87-107
}
Item Goldskin {
    Armor:431-459
}
Item Flamberge {
    Damage:16-17
}
Item Scourge {
    Damage:72-82
}
Item Bow {
    Damage:13
}
Item Ring {
    Damage:5
    Armor:5
}
Item Amulet {
    Damage:20
    Armor:3
}
Item Braid {
    Armor:53-79
}
Item Pike {
    Damage:15-16
}
Item Halberd {
    Damage:41-69
}
Item Treads {
    Armor:130
    Damage:10
}
```

✓ Please write a program to read and show the information. The result is shown as follows,

```
Elena   Damage:23-23    Armor:271-308
Linden  Damage:102-113  Armor:260-303
Mark    Damage:71-99    Armor:444-472
```

2. *TEXT PROCESSING

✓ Please create a text file named "`ex8-2.txt`".

> In computer science, functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data. It emphasizes the application of functions, in contrast to the imperative programming style, which emphasizes changes in state. Functional programming has its roots in lambda calculus, a formal system developed in the 1930s to investigate function definition, function application, and recursion. Many functional programming languages can be viewed as elaborations on the lambda calculus.

✓ Write a program to process "`ex8-2.txt`" and output to "`ex8-2-out.txt`".

```
The number of words read is 78
The longest word has a length of 12
The longest word is elaborations
```

✓ You program executes as follows,

```
> ./ex8-2
Please enter input file name: ex8-2.txt
Please enter output file name: ex8-2-out.txt
```