**NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES**

**CL 103-Computer Programming**

**Lab Session 12**

***Topic: Templates***

**Function Templates:**

In C++, **function templates** are functions that serve as a pattern for creating other similar functions. The basic idea behind function templates is to create a function without having to specify the exact type(s) of some or all of the variables. Instead, we define the function using placeholder types, called **template type parameters**. Once we have created a function using these placeholder types, we have effectively created a “function stencil”.

**Creating function templates in C++**

At this point, you’re probably wondering how to actually create function templates in C++. It turns out, it’s not all that difficult.

Let’s take a look at the int version of max() :

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4** | **int max(int nX, int nY)**  **{**  **return (nX > nY) ? nX : nY;**  **}** |

This is a good start — however, it won’t compile because the compiler doesn’t know what “Type” means! In order to tell the compiler that Type is meant to be a placeholder type, we need to formally tell the compiler that Type is a template type parameter. This is done using what is called a **template parameter declaration**:

|  |
| --- |
|  |

**template <typename Type> // this is the template parameter declaration**

**Type max(Type tX, Type tY)**

**{**

**return (tX > tY) ? tX : tY;**

**}**

If the template function uses multiple template type parameter, they can be separated by commas:

|  |  |
| --- | --- |
|  | **template <typename T1, typename T2>**  **// template function here** |

**Using function templates**

Using a function template is extremely straightforward — you can use it just like any other function:

|  |  |
| --- | --- |
|  | int nValue = max(3, 7); // returns 7  double dValue = max(6.34, 18.523); // returns 18.523  char chValue = max('a', '6'); // returns 'a' |

**#include <iostream>**

**#include<string>**

**using namespace std;**

**template <class T> T Max (T a, T b)**

**{**

**return (a>b)?a:b;**

**}**

**int main()**

**{**

**int x=90;**

**int y= 340;**

**cout<<Max(x,y)<<endl;**

**string n = "ABC123";**

**string s = "ABC";**

**cout<<Max(n,s)<<endl;**

**system("pause");**

**return 0;**

**}**

## Class Template:

Just as we can define function templates, we can also define class templates. The general form of a generic class declaration is shown here:

**template <class type> class class-name {**

**.**

**.**

**.**

**}**

Here, **type** is the placeholder type name, which will be specified when a class is instantiated. You can define more than one generic data type by using a comma-separated list.

Example 1

#include <iostream>

#include<string>

using namespace std;

template <class T> T Max (T a, T b)

{

return (a>b)?a:b;

}

int main()

{

int x=90;

int y= 340;

cout<<Max(x,y)<<endl;

string n = "ABC123";

string s = "ABC";

cout<<Max(n,s)<<endl;

return 0;

}

Example 2:

#include <iostream>

using namespace std;

// class template:

template <class T>

class mycontainer {

T element;

public:

mycontainer (T arg) {element=arg;}

T increase () {return ++element;}

};

// class template specialization:

template <>

class mycontainer <char> {

char element;

public:

mycontainer (char arg) {element=arg;}

char uppercase ()

{

if ((element>='a')&&(element<='z'))

element+='A'-'a';

return element;

}

};

int main () {

mycontainer<int> myint (7);

mycontainer<char> mychar ('j');

cout << myint.increase() << endl;

cout << mychar.uppercase() << endl;

return 0;

}

***Topic : File Handling in C++***

**Data File Handling In C++**

**File.** The information / data stored under a specific name on a storage device, is called a file.

**Stream.** It refers to a sequence of bytes.

**Text file.** It is a file that stores information in ASCII characters. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character. When this EOL character is read or written, certain internal translations take place.

## Classes for file stream operation

**ofstream**: Stream class to write on files  
**ifstream**: Stream class to read from files  
**fstream:** Stream class to both read and write from/to files.

## Opening a file

In order to open a file with a stream object we use its member function open():

**open (filename, mode);**

**ofstream** ofile**;** ofile**.open(**“data1”**);  
        
      ifstream** ifile**;** ifile**.open(**“data2”**);**

Where filename is a null-terminated character sequence of type const char \* (the same type that string literals have) representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags:

|  |  |
| --- | --- |
| **ios::in** | Open for input operations. |
| **ios::out** | Open for output operations. |
| **ios::binary** | Open in binary mode. |
| **ios::ate** | Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file. |
| **ios::app** | All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations. |
| **ios::trunc** | If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one. |

All these flags can be combined using the bitwise operator OR (|). For example, if we want to open the file example.bin in binary mode to add data we could do it by the following call to member function open():

|  |  |
| --- | --- |
|  | **ofstream myfile;**  **myfile.open ("example.bin", ios::out | ios::app | ios::binary);** |

**Default modes:**

|  |  |
| --- | --- |
| **Class** | **default mode parameter** |
| **ofstream** | ios::out |
| **ifstream** | ios::in |
| **Fstream** | ios::in | ios::out |

## Closing File

fout**.close();** fin**.close();**

### Text files

These files are designed to store text and thus all values that we input or output from/to them can suffer some formatting transformations, which do not necessarily correspond to their literal binary value.

## Basic Operation On Text File In C++

## **Program to write in a text file**

**#include<fstream.h>  
int main()  
{  
     ofstream fout;  
     fout.open("out.txt");  
     char str[300]="Time is a great teacher but unfortunately it kills all its pupils. Berlioz";  
     fout<<str;  
     fout.close();  
     return 0;  
}**

## **Program to read from text file and display it**

**#include <iostream>**

**#include <fstream>**

**#include <string>**

**using namespace std;**

**int main () {**

**string line;**

**ifstream myfile ("example.txt");**

**if (myfile.is\_open())**

**{**

**while ( myfile.good() )**

**{**

**getline (myfile,line);**

**cout << line << endl;**

**}**

**myfile.close();**

**}**

**else cout << "Unable to open file";**

**return 0;**

**}**

### Checking state flags

In addition to good(), which checks whether the stream is ready for input/output operations, other member functions exist to check for specific states of a stream (all of them return a bool value):

**bad()**

Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.

**fail()**

Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.

**eof()**

Returns true if a file open for reading has reached the end.

**good()**

It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

**Lab Tasks**

1. Define class Stack<> and implement generic methods to push and pop the elements from the stack.
2. Write a program to read bio-data of 5 persons from the console and write to an individual files. Filenames should be accepted from command line.