NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CL 217 – Object Oriented Programing Lab

Lab 12

Outline

- Working with template
- Templates functions
- Template classes
- Exercise

Generic Programming!

- **Generic programming** means that you are not writing source code that is compiled as-is but that you write "templates" of source codes that the compiler in the process of compilation transforms into source codes.
- Generic programming is basically the idea that your code should be as generic as possible.
- **Generic programming** is a style of computer programming in which algorithms are written in terms of to-be-specified-later types that are then instantiated when needed for specific types provided as parameters.

Generics can be implemented in C++ using Templates.

Templates!

C++ Templates

- Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates.
- the simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types. For example, a software company may need sort() for different data types. Rather than writing and maintaining the multiple codes, we can write one sort() and pass data type as a parameter.

C++ Templates

- The concept of templates can be used in two different ways:
  - Function Templates
  - Class Templates

How to declare a function template?

- A function template starts with the keyword template followed by template parameter/s inside < > which is followed by function declaration.

```
template <class T>
T someFunction(T arg)
{
   ... .. ...
}
```

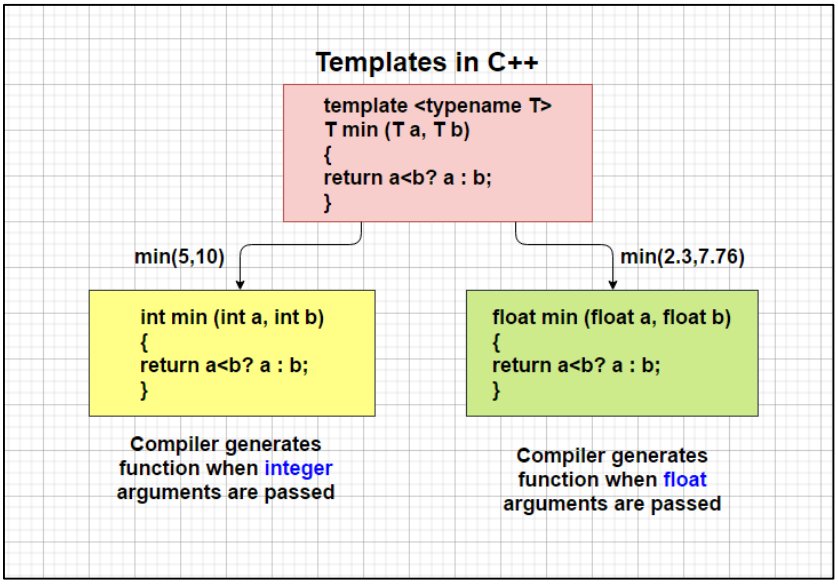
T is a placeholder that the compiler will automatically replace with an actual data type

Function Templates-Generic Function

```
int main()
{
    int i1, i2;
    float f1, f2;
    char c1, c2;
    cout << "Enter two integers:\n";
    cin >> i1 >> i2;
    cout << Large(i1, i2) <<" is larger."
<< endl;
    cout << "\nEnter two floating-point
numbers:\n";
    cin >> f1 >> f2;
    cout << Large(f1, f2) <<" is larger."
<< endl;
    cout << "\nEnter two characters:\n";
    cin >> c1 >> c2;
    cout << Large(c1, c2) << " has larger
ASCII value.";
    return 0;
}
#include <iostream>
using namespace std;
// template function
template <class T>
T  Large(T  n1, T n2)
{
    return (n1 > n2) ? n1 : n2;
}
```

This way, using only a single function template replaced three identical normal functions and made your code maintainable.

How templates work?



Class Templates-Generic Class

- you can also create class templates for generic class operations.
- Sometimes, you need a class implementation that is same for all classes, only the data types used are different.
- Normally, you would need to create a different class for each data type OR create different member variables and functions within a single class.
- This will unnecessarily bloat your code base and will be hard to maintain, as a change is one class/function should be performed on all classes/functions.

- However, class templates make it easy to reuse the same code for all data types.

How to declare a class template?

```
template <class T>
class className
{
   ... .. ...
public:
   T var;
   T someOperation(T arg);
   ... .. ...
};
```

In the above declaration, T is the template argument which is a placeholder for the data type used.

Inside the class body, a member variable var and a member function someOperation() are both of type T.

How to create a class template object?

```
className<dataType> classObject;
```

Example

```
className<int> classObject;
className<float> classObject;
className<string> classObject;
```

Program as Example

```
template <class T>
class Calculator
{
private:
        T num1, num2;

public:
        Calculator(T n1, T n2){
                num1 = n1;
                num2 = n2;}

        void displayResult(){
                cout << "Numbers are: " <<
num1 << " and " << num2 << "." << endl;
                cout << "Addition is: " <<
add() << endl;
                cout << "Subtraction is: " <<
subtract() << endl;
                cout << "Product is: " <<
multiply() << endl;
                cout << "Division is: " <<
divide() << endl;}

        T add() { return num1 + num2; }
        T subtract() { return num1 - num2; }
        T multiply() { return num1 * num2;
}
        T divide() { return num1 / num2; }
};
int main(){
        Calculator<int> intCalc(2, 1);
        Calculator<float> floatCalc(2.4, 1.2);
        cout << "Int results:" << endl;
        intCalc.displayResult();
        cout << endl << "Float results:" <<
endl;
        floatCalc.displayResult();

        return 0;
}
```

Template Function with Two Generic Types

- You can define more than one generic data type in the template statement by using a comma-separated list

```
template <class T1>
void myfunc(T1 a, T2 b)
{
cout << a << “  &  ” << b << '\n';
}
```

Specialized Template

```
template <class T>
void fun(T a)
{
    cout << "The main template fun(): "
         << a << endl;
}

template<>
void fun(int a)
{
    cout << "Specialized Template for int type: "
         << a << endl;
}
```

```
int main()
{
    fun<char>('a');
    fun<int>(10);
    fun<float>(10.14);
}
```

```
The main template fun(): a
Specialized Template for int type: 10
The main template fun(): 10.14

--------------------------------
Process exited after 0.1619 seconds with
```

Overloading a Generic Function

```
// First version of f() template
   template
template <class X>
void f(X a)
{
   cout << "Inside f(X a)";
}
```

```
// Second version of f()

template <class X, class Y>
void f(X a, Y b)
{
cout << "Inside f(X a, Y b)";
}
```

Activity

1. Implement bubble sort algorithm using function template and show the results for the following arrays

- 7, 5, 4, 3, 9, 8, 6
- 4.3, 2.5, -0.9, 100.2, 3.0

2. Create a class template with two generic data types to print their addition. Show the results for following types:

- int and double for example: (10,0.23) would print10.23
- char* and char*
    - For example: (“Now”, “Then”) would print Now Then

3. Create a class containing a function template capable of returning the sum of all the elements in an array being passed as parameter. Show the results for two arrays, one integer type and the other double type.

arr[] ={7,54,3,9,8,6};