Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

Lecture 10 Templates : Outline

- ✓ Templates Introduction
- ✓ Function Templates
- ✓ Overloading function templates
- ✓ Multiple arguments in function templates
- ✓ Class templates
- ✓ Multiple arguments class templates
- ✓ Default arguments in class templates

Templates

- One of the powerful feature of C++ is template.
- Template provides support for generic programming which allows generic class and generic functions.
- In short, using templates you can create single function or a class to work with different data types.
- Templates are expanded at compile time and compiler does type checking before this.
- Template concepts is used in two different ways
    - Function Templates
    - Class Templates

Function Template

- A function template works in a similar to a basic functions, with one key difference.
- A single function template can work with different data types at once but, a single normal function can only work with one set of data types.
- Normally, if we need to perform identical operations on two or more types of data, we use function overloading to create two functions with the required function declaration.
- However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.

Source: https://www.programiz.com/cpp-programming/templates

Syntax of Function Template

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

```
template <class T>

return_type function_name (arguments of type T1, T2....)

{

   // body of function.

}
```

- T is a template argument that accepts different data types (int, float), and **class** is a keyword.

Example

- Lets us consider a function which returns maximum of two numbers.

```
int max1(int a, int b)
{ return(a>b)?a:b;  }
float max1(float a, float b)
{ return(a>b)?a:b;  }
Void main()
{ int i,j;
cin>>i>>j;
cout<<max1(i,j);
float k,t;
cin>>k>>t;
cout<<max1(k,t);
}
```

Using template we can write one function for all data types

Function Template

```
//Here we have created a template function with myType as its template parameter.

template <class myType>
myType max1 (myType a, myType b)
{
return (a>b?a:b);
}
void main()
{ int i,j;
cin>>i>>j;
cout<<max1(i,j);
float k,t;
cin>>k>>t;
cout<<max1(k,t);   }
```

Templates with user defined data types

- In previous example, we cannot apply max function on objects of any class. E.g.

```
Class Test
{   int a;
Test(int x)
{ a=x;}
 };
void main()
{ Test T1(3), T2(8);
cout<<max1(T1,T2);
}
```

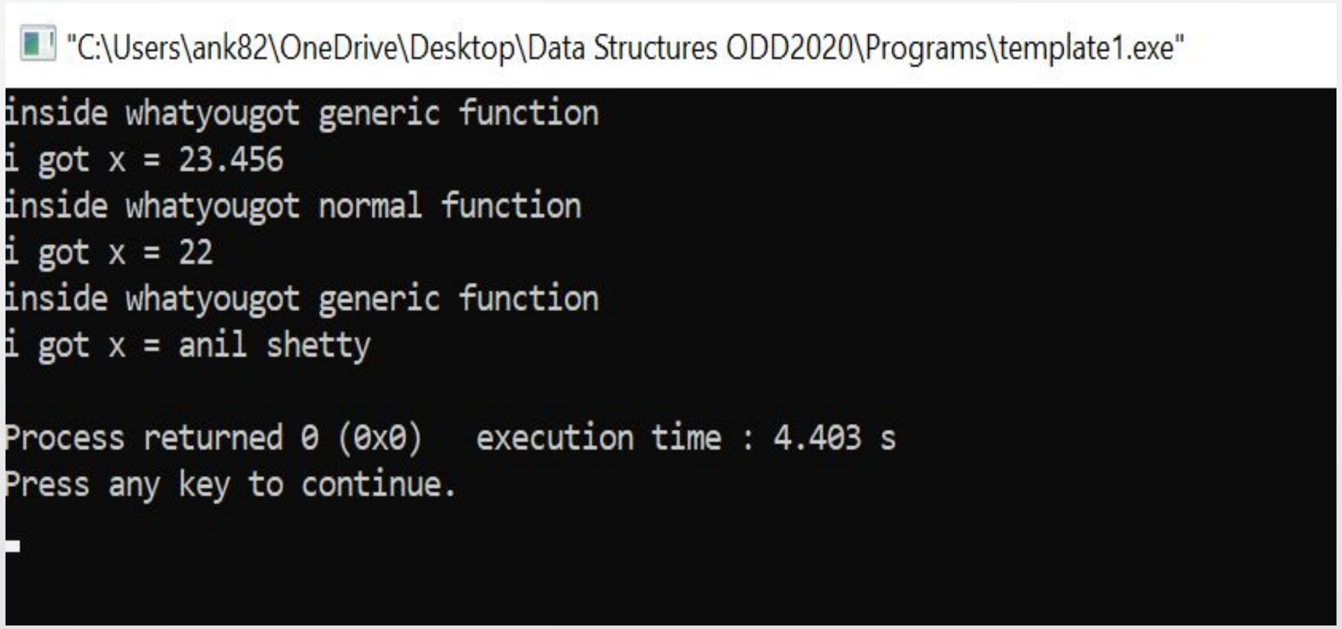
Template function max is not valid with such data type. We need to do operator overloading

Explicitly overloading generic function

```
template <class myType>

myType max1(myType a, myType b)

{ return (a>b?a:b);  }

char* max1(char *a, char *b)

{ if(strcmp(a,b)>0)

return a;

else

return b; }

Void main()

{ cout<<max1(6,10);

Char s1[20],s2[25];

cout<<max1(s1,s2);

}
```

Overloading function templates

```
#include <iostream>

using namespace std;

template <typename T>void whatYouGot(T x);

template <>void whatYouGot (int x);

int main()

{    whatYouGot(23.456);

   whatYouGot(22);

   whatYouGot("anil shetty");

   return 0;

}

template <typename T>void whatYouGot(T x){

cout << "inside whatyougot generic function"<<endl;

cout << "i got x = "<<x<<endl;

}

template <>void whatYouGot(int x){

cout << "inside whatyougot normal function"<<endl;

cout << "i got x = "<<x<<endl;

}
```



"C:\Users\ank82\OneDrive\Desktop\Data Structures ODD2020\Programs\template1.exe"

```
inside whatyougot generic function
i got x = 23.456
inside whatyougot normal function
i got x = 22
inside whatyougot generic function
i got x = anil shetty

Process returned 0 (0x0)   execution time : 4.403 s
Press any key to continue.
```

Source:https://www.learninglad.com/2014/02-explicitly-overloading-generic-functions-with-normal-functions.html

Multiple Arguments Function Templates

```
#include <iostream>
#include <cstring>
using namespace std;
template <class T1, class T2, class T3>
void reverse(T1 x1, T2 x2, T3 x3 )
{
cout<<"original sequence"<<" "<<x1<<" "<<x2<<" "<<x3<<endl;
cout<<"reverse sequence"<<" "<<x3<<" "<<x2<<" "<<x1<<endl;


}
int main()
{
reverse(12, 34, 86);
reverse("JIIT","Best","university");
return 0;}
```

When you call a function template, the compiler tries to *deduce* the template type.

Most of the time it can do that successfully, but every once in a while you may want to help the compiler deduce the right type —
either because it cannot deduce the type at all, or perhaps because it would deduce the wrong type.

For example, you might be calling a function template that doesn’t have any parameters of its template argument types, or you might want to force the compiler to do certain promotions on the arguments before selecting the correct function template.

In these cases you’ll need to explicitly tell the compiler which instantiation of the function template should be called.

```
// template arguments
#include <iostream>
using namespace std;
template <class T, int N>
T fixed_multiply (T val)
{
  return val * N;
}

int main() {
  std::cout << fixed_multiply<int,2>(10) << '\n';
  std::cout << fixed_multiply<int,3>(10) << '\n';
}
```

Class Templates

- **Class Templates** are similar to function templates
- class templates are useful when a class defines something that is independent of the data type.
- Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.
- Whenever there is a need of several classes with similar members, differing only in data types, we use class template.

Need of class Template

```
class charstack
{
char arr[10];
int top;
public:
charstack() { top=-1;}
Void push(char a)
{ arr[top++]=a;}
char pop()
{return arr[top--];  }
};
```

```
class intstack
{
int arr[10];
int top;
public:
intstack() { top=-1;}
Void push(char a)
{ arr[top++]=a;}
int pop()
{return arr[top--];  }
};
```

Syntax of class Template

Syntax:

```
template <class T>
class  ClassName
{………..
public:
T var;
T function1(T args);
….
};
```

A template can have char strings, function names, ints etc. as agruments.

Template class

```
template <class T>

class temp_stack

{ T arr[20];

int top;

public:

Temp_stack()

{ top=-1;}

Void push(T a);

T pop();

};
```

Object creation

To create a class template object, the data type is defined inside a <> when creation.

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

For example:

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

Previous example:

```
temp_stack <int> ob1;

temp_stack <char> ob2;
```

Multiple arguments in class template

```
template <class T, int size>
Class Test1
{ T arr[size];
};
```

Object creation:

```
Test1<int,5> obj;
```

Member functions of Template class

- A member function of a template class is implicitly treated as a template function and it can have template arguments which are same as its class template arguments.

```
template <class T>
class temp_stack
{ void push(T a);
};
```

When defined outside:

```
template <class T>
Void temp_stack<T>::push(T a)
{
}
```

```
template<class T1, class T2>

class Test

{

T1 a;

T2 b;

Public:

Test(T1 x, T2 y)

{ a=x; y=b; }

Void show();

};
```

```
template<class T1, class T2>

void Test<T1, T2> :: show()

{  cout<<a<<" "<<b<<endl;

}

int main()

{ Test<int, double> ob1(20,20.99);

Test<float,char*> ob2(4.3,"JIIT");

ob1.show();

ob2.show();

return 0;

}
```

Using default arguments with template class

```
template<class T, class U = int> class A;

template<class T = float, class U> class A;

template<class T, class U> class A {

   public:

      T x;

      U y;

};

A<> a;
```

The type of member a.x is float, and the type of a.y is int.

You cannot give default arguments to the same template parameters in different declarations in the same scope. For example, the compiler will not allow the following:

```
template<class T = char> class X;

template<class T = char> class X { };
```

Source: https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzarg/default_args_for_templ_params.htm

```
#include<iostream>
using namespace std;

template<class T, class U = char>
class Test
{
public:
    T x;
    U y;
};

int main()
{
    Test<char> ob1;
    Test<int, int> ob2;
    cout<<sizeof(ob1)<<endl;
    cout<<sizeof(ob2)<<endl;
    return 0;
}
```

Home Exercise

- Write a function template to add two data of any type
- Write a function template to swap two data of any type

References

- Bjarne Stroustrup, The C++ Programming Language (4th Edition), Addison-Wesley, May 2013.
- Robert Lafore, Object Oriented Programming in C++, SAMS, 2002
- https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzarg/default_args_for_templ_params.htm
- https://www.programiz.com/cpp-programming/templates