Templates

Chapter 8

Contents

Introduction

- Framework used to create generic functions and classes.
- In a generic function or class, the type of the data that is operated upon is specified as a parameter.
- This allows us to use one function of class with several different types of data without having two explicitly recode a specific version for each different data type.
- Templates allow us to create reusable code.

Function Templates/ Generic Functions

- A generic function defines a general set of operations that will be applied to various types of data.
- A generic function has the type of data that it will operate upon passed to it as parameter.
- Using this mechanism the same general procedure can be applied to a wide range of data.
- Once we create the generic function the compiler automatically generates the correct code for the type of data that is actually used when we execute the function.
- When we create a generic function we are creating a function that can automatically overload itself.

Function Templates/ Generic Functions

- A generic function is created using the keyword “template”. The general form of a template function is given below:

```
template <class Ttype> return_type function_name (parameter list)
      {
              …………. ; //body
      }

                          OR

 template <class Ttype>
   return type function name (parameter list)
       {
             …………. ;  //body
       }
```

Function Templates/ Generic Functions

- Here, Ttype is a place holder name for a data type used by the function.
- The compiler will automatically replace this place holder with an actual data type when it creates a specific version of the function. Instead of the keyword “class” we can also use the keyword “type name” to specify a generic type:

e.g. the line

template <class X> void show (X &a, X &b)

tells the compiler two things:

  - The template is being created.
  - A generic definition is beginning.

- Here, X is a generic type that is used as place holder. After the template portion the function show is declared, using X as a data type of the values that will be swapped.

Function Templates/ Generic Functions

- SOME TERMS
- A generic function is also called a template function.
- When the compiler creates a specific version of this function it is set to have created a generated function.
- The act of generating function is referred as instantiating it.

Notes

1)The template portion of the generic function definition does not have to be on the same line as the functions name.

2)No other statement can occur between the template statement and start of the generic function definition i.e. the template specification must directly continue to the rest of the function definition.

3)We can define more than one generic data type within the template statement, using a (,) comma separated list.

e.g. template <class type 1, class type2>

Class templates/generic classes

- When we define generic class, we create a class that defines all algorithms used by that class but the actual type of the data being manipulated will be specified as a parameter when object of that classes are created.
- Generic classes are useful when a class contains generalizable logic.
- By using generic class, we can create a class that will maintain stack, queue, linked list and so on for any type of data.
- The compiler will automatically generate the correct type of object based upon the type we specify when the object is created.
- The general form of a generic class declaration is:
    - template <class Ttype> class class_name
- Here, Ttype = place holder name that will be specified when a class is instantiated.
- Once we create a generic class we have to create specific instance of that class by using the following general form. Then the type is the type name of data that the class will be operated upon.

Class templates/generic classes

- Member functions of the generic class are themselves, automatically generic.
- C++ provides a library (standard template library or STL) i.e. build upon template classes. It provide generic version of the most commonly used algorithm and data structures.
- A template class can have more than one generic data type separated by comma (,) within the template specification.

```
template < class type 1, class type 2> class myclass
while instantiating in main( )
 myclass <int, double> obj1 (10,70.000)
 myclass <char, char *) obj2 (‘x’, “This is a computer era”) ;
```

Sample Program 1: Function Template

```
#include<iostream>
using namespace std;
template <class x>
void swapargs (x &a, x &b)
{
    x temp;
    temp =a;
    a=b;
    b=temp;
}
```

```
int main ( )
{
    int i=10, j=20;
    float x=10.5, y=15.5;
    cout<<"original values of i, j: "<<i<<"\t"<<j<<endl;
    cout<<"original values of x, y: "<<x<<"\t"<<y<<endl;
    swapargs (i,j);
    swapargs (x,y);
    cout<<"after swapargs the value of i,j= "<<i<<"\t"<<j<<endl;
    cout<<"after swapargs the value of x,y: "<<x<<"\t"<<y<<endl;
    return 0;
}
```

Output:

```
original values of i , j: 10   20
original values of x , y: 10.5   15.5
after swapargs the value of I , j = 20    10
after swapargs the value of x , y:  15.5     10.5
```

Sample Program 2: Class Template

```
#include<iostream>
using namespace std;
template <class T>
class mytemp
{
   private:
   T a,b,c;
   public:
   void getdata ( )
   {
      cout<<"enter first value:";
      cin>>a;
      cout<<"enter second value:";
      cin>>b;
      cout<<endl;
   }
void total ( )
{
   c=a+b;
   cout<<"total value is:"<<c<<endl;
   cout<<endl;
}
};
```

```
int main ( )
{
   mytemp <int> obj1;
   mytemp <float> obj2;
   cout<<"enter two intergers:"<<endl;
   obj1.getdata ( );
   obj1.total ( );
   cout<<"enter two floats:"<<endl;
   obj2.getdata ( );
   obj2.total ( );
   return 0;
}
```

Output:

```
enter two intergers:
enter first value: 1
enter second value: 2

total value is: 3

enter two floats:
enter first value: 0.5
enter second value: 0.6
total value is: 1.1
```

Program 3: Overloaded functions specified for each data type

```
#include <iostream>
using namespace std;
int square (int x)
{
  return x * x;
};

float square (float x)
{
  return x * x;
};

double square (double x)
{
  return x * x;
};
```

```
int main()
{
   int    i, ii;
   float  x, xx;
   double y, yy;
   i = 2;
   x = 2.2;
   y = 2.2;
   ii = square(i);
   cout << i << ": " << ii << endl;
   xx = square(x);
   cout << x << ": " << xx << endl;
   yy = square(y);
   cout << y << ": " << yy << endl;
}
```

Program 4 : A single template to support all data types

```
/* Function template
   implementation of Program 3 */
#include <iostream>
using namespace std;
template <class T>
inline T square(T x)
{
   T result;
   result = x * x;
   return result;
};
```

```
int main()
{
int    i, ii;
float  x, xx;
double y, yy;
i = 2;
x = 2.2;
y = 2.2;
ii = square<int>(i);
cout << i << ": " << ii << endl;
xx = square<float>(x);
cout << x << ": " << xx << endl;
// Explicit use of template
yy = square<double>(y);
cout << y << ": " << yy << endl;
// Implicit use of template
yy = square(y);
cout << y << ": " << yy << endl;
return 0;
}
```

Discussion on Program 3 and Program 4

Note:

- The code used in the overloaded C++ functions example above, is repeated for each data type. The templated function is specified once.
- The templated type keyword specifier can be either "class" or "typename":
- template<class T>
- template<typename T>
- Both are valid and behave exactly the same.
- The template function works using either the explicit or implicit template expression square<int>(value) or square(value).
- In the template definition, "T" represents the data type. The compiler will generate the type specific functions required. This results in a more compact code base which is easier to maintain.
- The code and logic of the functions only has to be specified once in the template function and the parameter "T" is used to represent the argument type.
- The template declaration and definition must reside in the same file, typically an include header file.
- A "C" macro function could also perform this purpose: #define square(x) (x * x)
- The advantage of the template is that it performs type checking while the macro does not.