C++ Programming

Trainer : Rohan Paramane

Email: rohan.paramane@sunbeaminfo.com

Template

- If we want to write generic program in C++, then we should use template.
- This feature is mainly designed for implementing generic data structure and algorithm.
- If we want to write generic program, then we should pass data type as a argument. And to catch that type we should define template.
- Using template we can not reduce code size or execution time but we can reduce developers effort.

Template

int num1 = 10, num2 = 20; swap_object<**int**>**(num1, num2);** string str1="Pune", str2="Karad"; swap_object<string>(str1, str2);	In this code, <int> and <string> is considered as type argument.

template<typename T> //or
template<class T> //T : Type Parameter
void swap(b obj1, **T** obj2 **)**
{
T temp = obj1;
obj1 = obj2;
obj2 = temp;
}
	template and typename is keyword in
	C++. By passing datatype as argument
	we can write generic code hence parameterized type is called template

- **Types of Template**
  - Function Template
  - Class Template

Example of Function Template

```
//template<typename T>//T : Type Parameter
template<class T> //T : Type Parameter
void swap_number( T &o1, T &o2 )
{   T temp = o1;
    o1 = o2;
    o2 = temp;
}
 int main( void )
{
    int num1 = 10;
    int num2 = 20;
    swap_number<int>( num1, num2 );     //Here int is type argument
    cout<<"Num1 : "<<num1<<endl;
    cout<<"Num2 : "<<num2<<endl;
    return 0;
}
```

Example of Class Template

```
template<class T>
class Array // Parameterized type
{
private:
   int size;
   T *arr;
   public:
   Array( void ) : size( 0 ), arr( NULL )
   {
   }
   Array( int size )
   {
   this->size = size;
   this->arr = new T[ this->size ];
   }
   void acceptRecord( void ){}
   void printRecord( void ){ }
   ~Array( void ){ }
};
```

```
int main(void)
{
Array<char> a1( 3 );
a1.acceptRecord();
a1.printRecord();
return 0;
}
```

STL

- The Standard Template Library (STL) is a set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays, etc.
- It is a library of container classes, algorithms, and iterators.
- It is a generalized library and so, its components are parameterized.
- Working knowledge of template classes is a prerequisite for working with STL.

- STL has 4 components:
  - Algorithms
  - Containers
  - Functions
  - Iterators

Components of STL

1. **Algorithm**

They act on containers and provide means for various operations for the contents of the containers.

- Sorting
- Searching

2. **containers**

- Containers or container classes store objects and data.

3. **Functions**

- The STL includes classes that overload the function call operator.
- Instances of such classes are called function objects or functors.
- Functors allow the working of the associated function to be customized with the help of parameters to be passed.

4. **Iterators**

- As the name suggests, iterators are used for working upon a sequence of values. They are the major feature that allows generality in STL.
- Iterators are used to point at the memory addresses of STL containers.
- They are primarily used in sequences of numbers, characters etc.
- They reduce the complexity and execution time of the program.

Vector

- Vectors are the same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container.
- Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.
- In vectors, data is inserted at the end. Inserting at the end takes differential time, as sometimes the array may need to be extended. Removing the last element takes only constant time because no resizing happens.
- Inserting and erasing at the beginning or in the middle is linear in time.
- begin() – Returns an iterator pointing to the first element in the vector
- end() – Returns an iterator pointing to the theoretical element that follows the last element in the vector
- size() – Returns the number of elements in the vector.
- empty() – Returns whether the container is empty.
- front() – Returns a reference to the first element in the vector
- back() – Returns a reference to the last element in the vector
- assign() – It assigns new value to the vector elements by replacing old ones
- push_back() – It push the elements into a vector from the back
- pop_back() – It is used to pop or remove elements from a vector from the back.
- insert() – It inserts new elements before the element at the specified position
- erase() – It is used to remove elements from a container from the specified position or range.

Local & Nested Class

- **Local Class**
  - If inside a function you declare a class then such classes are called as local classes.
  - Inside local class you can access static and global members but you cannot access the local members declared inside the function where the class is declared.
  - Inside local classes we cannot declare static data member however defining static member functions are allowed.

- **Nested class**
  - A class declared inside another class is called as nested class
  - A nested class can access all the private and public members of outer class directly on the outer class object
  - An outer class can access only public members of nested inner class on its object.
  - A Nested class can access private static members of outer class directly

Scope

- It decides area/region/boundry in which we can access the element.
- **Types of scope in C++:**
  1. Block scope
  2. Function scope
  3. Prototype scope
  4. Class scope
  5. Namespace scope
  6. File scope
  7. Program scope

Example Scope

```
int num6;                   //Program Scope
static int num5;            //File Scope
```

```
namespace ntest {
    int num4;               //Namespace scope

    class Test {
    int num3;               //Class Scope
};
}
```

```
int main( void ) {
        int num1 = 10; //Function Scope
        while( true ) {
        int temp = 0;    //Block Scope
}
        return 0;
}
```

```
void sum( int num1, int num2 ); //Prototype scope
```

Macro

- Symbolic constant is called as macro
- Expanding macro is a job of pre processor.

- Example:
  - #define SIZE 10
  - #define EOF -1
  - #define MULTIPLY(x,y) x*y

- Few other Macro's in C++
  - __FILE__, __LINE__,__DATE__,__TIME__

Thank You