Data Structures & Algorithms

Week 8: Templates and STL

Agenda

- Templates in C++
- Function Templates
- Bubble Sort using Function Templates
- Class Templates
- Template Arguments
- Function Overloading vs. Templates
- Intro to Standard Template Library in C++

Templates in C++

- A **template** is a simple yet very powerful tool in C++.
- 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 to sort() for different data types
  - Rather than writing and maintaining multiple codes, we can write one sort() and pass data type as a parameter.

How Do Templates Work?

- C++ adds two new keywords to support templates: *‘template’* and *‘typename’*.
- The second keyword can always be replaced by the keyword ‘class’.
- Templates are expanded at compiler time.
- Compiler does type checking before template expansion.
- The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of the same function/class.

How Do Templates Work?

```
template <typename T>
T myMax(T x, T y)
{
    return (x > y)? x: y;
}

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```

Function Templates

- We write a generic function that can be used for different data types.
- Examples of function templates are sort(), max(), min(), printArray().

Function Templates

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

// One function works for all data types.  This would work
// even for user defined types if operator '>' is overloaded
template <typename T> T myMax(T x, T y)
{
    return (x > y) ? x : y;
}

int main()
{
    cout << myMax<int>(3, 7) << endl; // Call myMax for int
    cout << myMax<double>(3.0, 7.0)
         << endl; // call myMax for double
    cout << myMax<char>('g', 'e')
         << endl; // call myMax for char

    return 0;
}
```

Task: Implement Bubble Sort using templates

Solution

```cpp
// CPP code for bubble sort
// using template function
#include <iostream>
using namespace std;

// A template function to implement bubble sort.
// We can use this for any data type that supports
// comparison operator < and swap works for it.
template <class T> void bubbleSort(T a[], int n)
{
    for (int i = 0; i < n - 1; i++)
        for (int j = n - 1; i < j; j--)
            if (a[j] < a[j - 1])
                swap(a[j], a[j - 1]);
}
```

Solution

```
// Driver Code
int main()
{
    int a[5] = { 10, 50, 30, 40, 20 };
    int n = sizeof(a) / sizeof(a[0]);

    // calls template function
    bubbleSort<int>(a, n);

    cout << " Sorted array : ";
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;

    return 0;
}
```

Class 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.

Class Templates

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

template <typename T> class Array {
private:
    T* ptr;
    int size;

public:
    Array(T arr[], int s);
    void print();
};
```

Class Templates

```
template <typename T> Array<T>::Array(T arr[], int s)
{
    ptr = new T[s];
    size = s;
    for (int i = 0; i < size; i++)
        ptr[i] = arr[i];
}

template <typename T> void Array<T>::print()
{
    for (int i = 0; i < size; i++)
        cout << " " << *(ptr + i);
    cout << endl;
}

int main()
{
    int arr[5] = { 1, 2, 3, 4, 5 };
    Array<int> a(arr, 5);
    a.print();
    return 0;
}
```

Templates Arguments

- We can pass more than one data type as arguments to templates.

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

template <class T, class U> class A {
    T x;
    U y;

public:
    A() { cout << "Constructor Called" << endl; }
};

int main()
{
    A<char, char> a;
    A<int, double> b;
    return 0;
}
```

Templates Arguments

- We can specify a default value for template arguments

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

template <class T, class U = char> class A {
public:
    T x;
    U y;
    A() { cout << "Constructor Called" << endl; }
};

int main()
{
    A<char> a; // This will call A<char, char>
    return 0;
}
```

Function overloading vs. templates

- Both function overloading and templates are examples of polymorphism features of OOP.
- Function overloading is used when multiple functions do quite similar (not identical) operations.
- Templates are used when multiple functions do identical operations.

Notes

- Each instance of a template contains its own static variable.
- We can pass non-type arguments to templates. Non-type parameters are mainly used for specifying max or min values or any other constant value for a particular instance of a template.

Standard Templates in C++

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

Containers in STL

- Containers or container classes store objects and data.
- There are in total seven standards “first-class” container classes and three container adaptor
- Sequence Containers: implement data structures that can be accessed in a sequential manner.
  - vector
  - list
  - deque
  - arrays
  - forward_list(Introduced in C++11)
- Container Adaptors: provide a different interface for sequential containers.
  - queue
  - priority_queue
  - stack

Algorithms in STL

- It is a collection of functions specially designed to be used on a range of elements.
- They act on containers and provide means for various operations for the contents of the containers.
  - Algorithm
  - Sorting
  - Searching
  - Important STL Algorithms
  - Useful Array algorithms
  - Partition Operations

Lecture content adapted from G4G
https://www.geeksforgeeks.org/templates-cpp/