Preview

- What is Template?
- Function Template.
- Class Template

What is Templates ?

- Many C++ programs use common data structures like stacks, queues and lists. A program may require a queue of any data types.
- Instead of defining a individual data structure for different type, we can accomplish with types parameterization called templates

What is Templates ?

- The idea of template is that pass type as a parameter
- Templates can be used for functions, class as well as any structure data type.
- Templates are very useful when implementing generic constructs like vectors, stacks, lists, queues which can be used with any arbitrary type.
- C++ templates provide a way to re-use source code, instead of defining overload functions

Function Template

- Use function templates to write generic functions that can be used with arbitrary types.
- For example,
  - we can write a function to compare two variables which can be used with any arbitrary type.
  - We can write searching and sorting routines which can be used with any arbitrary type.

Function Template

```
// TemExe.cpp Example for function template
#include <iostream>
using namespace std ;

template <class T>
T Max(T, T);

int main()
{

        cout << "Max(10, 15) = " << Max<int>(10, 15) << endl ;
        cout << "Max('k', 's') = " << Max<char> ('k', 's') << endl ;
        cout << "Max(10.1, 15.2) = " << Max<float>(10.1, 15.2) << endl ;
        return 0;
}

template <class T>
T Max(T a, T b)
{
        return a > b ? a : b ;
}
```



```
// ftemplate.cpp:Using template functions
#include <iostream>

using namespace std;

template< class T >
void printArray( const T *, const int  );

int main()
{
   const int aCount = 5, bCount = 7, cCount = 6;
   int a[ aCount ] = { 1, 2, 3, 4, 5 };
   double b[ bCount ] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
   char c[ cCount ] = "HELLO";  // 6th position for null

   cout << "Array a contains:" << endl;
   printArray<int>( a, aCount );  // integer template function

   cout << "Array b contains:" << endl;
   printArray<float>( b, bCount );  // double template function

   cout << "Array c contains:" << endl;
   printArray<char>( c, cCount );  // character template function

   return 0;
}

template< class T >
void printArray( const T *array, const int count )
{
   for ( int i = 0; i < count; i++ )
      cout << array[ i ] << " ";

   cout << endl;
}
```

Class Templates

- C++ Class Templates are used where we have multiple copies of code for different data types with the same logic.
- If a set of functions or classes have the same functionality for different data types, they becomes good candidates for being written as Templates.
- Ex) Stacks and Queues with different data types

Class Templates

- A class template definition looks like a regular class definition, except it is prefixed by the keyword template.

```
template <class T>
class ClassName{
{
   private:
        T a;
         …
};
```



```
// Class Template Example
#include <iostream>

using namespace std ;

// Class with Template
template <class T>
class Simple {
private:
            T A;
public:
            Simple(T);
            void Print();
};

// Constructor
template <class T>
Simple<T>::Simple(T x)
{
            A = x;
}

// since private part can be different types
template <class T>
void Simple<T>:: Print ()
{
            cout << A<<endl;
}

void main()
{

            Simple <int>A(12);
            Simple <char*>B("Park");
            Simple <float>C(1.23);
            A.Print();
            B.Print();
            C.Print();
}
```



```
// Class Template Example with more than one types
#include <iostream>
using namespace std ;

// Class with Template
template <class T, class Q>
class Simple {
private:
            T A;
            Q B;
public:
            Simple(T, Q);
            void Print();
};

// Constructor
template <class T, class Q>
Simple<T,Q>::Simple(T x, Q y)
{
            A = x;
            B = y;
}

// since private part can be different types
template <class T, class Q>
void Simple<T, Q>:: Print ()
{
            cout << A<<" ";
            cout << B <<endl;
}
int main()
{
            Simple <char*,int> A("Sam",12);
            Simple <float, int> B(1.24, 3);
            Simple <float, char*> C(1.23, "Michael");
            A.Print();
            B.Print();
            C.Print();
            return 0;
}
```