GENERIC PROGRAMMING – FUNCTION TEMPLATES AND CLASS TEMPLATES

1. Function Template

```
#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()
{
	// Call myMax for int
	cout << myMax<int>(3, 7) << endl;
	// call myMax for double
	cout << myMax<double>(3.0, 7.0) << endl;
	// call myMax for char
	cout << myMax<char>('g', 'e') << endl;
	return 0;
}
```

Output:

```
7
7
g
```

2. Class Template

```
#include <iostream>
using namespace std;
template <typename T>
```

```
class Array {
private:
	T* ptr;
	int size;
public:
	Array(T arr[], int s);
	void print();
};
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;
}
```

Output:

1 2 3 4 5

3. Soring using Function Template

```
#include <iostream>
using namespace std;
template < class Type >
  void bubbleSort(Type arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
      bool swapDone = false;
      for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
          Type temp = arr[j];
          arr[j] = arr[j + 1];
          arr[j + 1] = temp;
          swapDone = true;
        }
      }
      if (!swapDone) return;
    }
  }
int main() {
  int n = 5;
  int arr1[] = {
    11,
    4,
    9,
    2,
    0
  };
  float arr2[] = {
```

```
    3.67,
    9.87,
    1.22,
    2.45,
    4.32
  };

  bubbleSort(arr1, n);
  bubbleSort(arr2, n);
  cout << "Sorting of Integers\n";
  for (int i = 0; i < n; i++) {
    cout << arr1[i] << " ";
  }
  cout << "\nSorting of Floating Point Numbers\n";
  for (int i = 0; i < n; i++) {
    cout << arr2[i] << " ";
  }
  return 0;
}
```

Output:

Sorting of Integers

0 2 4 9 11

Sorting of Floating Point Numbers

1.22 2.45 3.67 4.32 9.87