National University of Sciences & Technology
School of Electrical Engineering and Computer Science
Department of Computing

CS-212 Object Oriented Programming
Assignment # 3

Generics / Templates			
Maximum Marks: **40**		Instructor: **Muhammad Khurram Shahzad**	
Submission Date: **28th December 2021**		Type: Written/Individual/LMS/Hardcopy	
Name: **Muhammad Umer**	Reg. #: **345834**	Degree: **BEE**	Section: **12C**

Instructions:

- You need to implement the concepts using skills/concepts learned in this course.
- You can get internet help, but assignments are individual
- You can submit on LMS before due date and submit hard copy in the class
- Submission after due date get zero marks

Combine the knowledge from file handing and ASCII table.

- The idea of GENERICS was born when programmers decided to use the same code but with different data types. In contrast to function overloading where data types are hardcoded in the program, in generic programming, a user should be able to pass data types.

Guidelines

- Write a program that shows operator and function overloading.
- Now implement the same program, however, using templates.

Deliverables

- Single word file consisting of programs 1 and 2 (*with line number against each line of code*) along with their output file.

Idea & Objective

The motif behind the writing of the following program is to take advantage of generic programming; that is to create a template of a block of code in order to minimize the number of lines the code takes while also allowing for the same function to be applicable over a range of data types.

For the present case, a simple sorting algorithm, **bubble sort**, has been implemented that sorts an array consisting of numerical data elements which can be either of one of the following data types: ***int, float or double***.

The same program has been implemented twice; once through function overloading and the other through utilizing templates. Although templates do not offer any advantage in terms of *memory allocation*, we can deduce that they increase the simplicity of the program and hence, make it easier to debug.

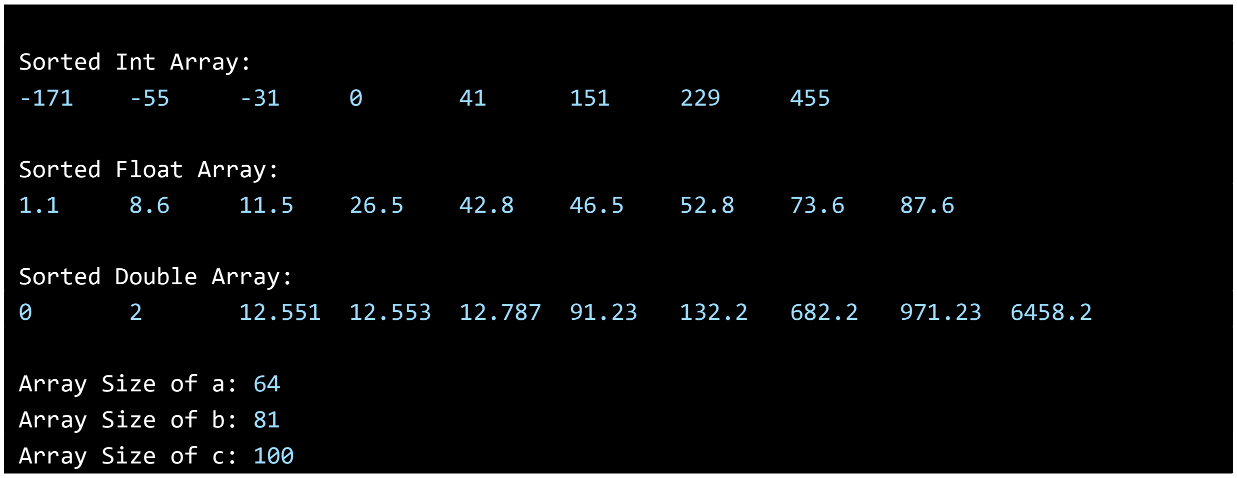
Bubble Sorting through Function Overloading

```
// Program to sort an array of ten elements
// that can be either of int, float, double
// datatype, sorting would work regardless
// *EXCLUDING CHARS as sorting an array of
// char is pointless

// USING FUNCTION OVERLOADING

#include <iostream>
using namespace std;

class ArrAssignment
{
private:
    int n;

public:
    ArrAssignment(int size)
    {
        n = size;
    }

    // Bubble sort for int
    void sortArray(int arr[])
    {
        // to check for validity of sortness
        bool b = true;
        while (b)
        {
            b = false;
            for (int i = 0; i < n - 1; i++)
            {
                // sorting algorithm
                if (arr[i] > arr[i + 1])
                {
                    int temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    b = true;
                }
            }
        }
    }
```

```cpp

    // Display Function for int
    void printArray(int arr[])
    {
        for (int i = 0; i < n; ++i)
            cout << arr[i] << "\t";
        cout << endl;
    }

    // Bubble sort for float
    void sortArray(float arr[])
    {
        // to check for validity of sortness
        bool b = true;
        while (b)
        {
            b = false;
            for (int i = 0; i < n - 1; i++)
            {
                // sorting algorithm
                if (arr[i] > arr[i + 1])
                {
                    int temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    b = true;
                }
            }
        }
    }

    // Display Function for float
    void printArray(float arr[])
    {
        for (int i = 0; i < n; ++i)
            cout << arr[i] << "\t";
        cout << endl;
    }

    // Bubble sort for double
    void sortArray(double arr[])
    {
        // to check for validity of sortness
        bool b = true;
        while (b)
        {
            b = false;
            for (int i = 0; i < n - 1; i++)
            {
                // sorting algorithm
                if (arr[i] > arr[i + 1])
                {
                    double temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    b = true;
                }
            }
        }
    }

    // Display Function for double arr
    void printArray(double arr[])
    {
        for (int i = 0; i < n; ++i)
            cout << arr[i] << "\t";
        cout << endl;
    }

    friend ostream& operator+=(ostream&, const ArrAssignment&);
};

ostream& operator+=(ostream& out, const ArrAssignment& e)
{
    cout << "\nCurrent Array Size: ";
    return out << e.n * e.n;
}

// Code to drive the above class
int main()
{
    // To check for all numerical data types
    ArrAssignment a(8); // 8, 9, 10 represent size of array
    ArrAssignment b(9);
    ArrAssignment c(10);

    // Testint the sorting algorithm through function overloading
    // upon the aforementioned data types
    int intArr[] = { 41, -55, 151, -171, 229, -31, 0, 455 };
    a.sortArray(intArr);
    cout << "\nSorted Int Array:\n";
    a.printArray(intArr);

    float floatArr[] = { 46.5, 73.6, 11.5, 87.6, 8.6, 52.8,
        42.8, 26.5, 1.1 };
    b.sortArray(floatArr);
    cout << "\nSorted Float Array:\n";
    b.printArray(floatArr);

    double doubleArr[] = { 12.553, 6458.2, 2, 91.23, 12.787,
        132.2, 0.00, 971.23, 12.551, 682.2 };
    c.sortArray(doubleArr);
    cout << "\nSorted Double Array:\n";
    c.printArray(doubleArr);

    // Operator overloading that returns the square
    // of number of elements present in each array
    cout << "\nArray Size of a: ";
    cout += a;
    cout << "\nArray Size of b: ";
    cout += b;
    cout << "\nArray Size of c: ";
    cout += c;

    return 0;
}
```

Console Output

```
PS D:\NUST\Semester 3\Object Oriented Programming\Assignments> cd "d:\NUST\Semester 3\Object Oriented Programming\Assignments\Assignment 3\" ; if ($?) { g++ overload.cpp -o overload } ; if ($?) { .\overload }
```



```
Sorted Int Array:
-171    -55     -31     0       41      151     229     455

Sorted Float Array:
1.1     8.6     11.5    26.5    42.8    46.5    52.8    73.6    87.6

Sorted Double Array:
0       2       12.551  12.553  12.787  91.23   132.2   682.2   971.23  6458.2

Array Size of a: 64
Array Size of b: 81
Array Size of c: 100
```

Bubble Sorting through Templates

```cpp
// Program to sort an array of ten elements
// that can be either of int, float, double
// datatype, sorting would work regardless
// *EXCLUDING CHARS as sorting an array of
// char is pointless

// USING TEMPLATES

#include <iostream>
using namespace std;

// Template formed so that sorting of any
// type variable is possible

template <typename T>
class ArrAssignment;

template <typename T>
ostream& operator+=(ostream&, const ArrAssignment<T>&);

template <typename T>
class ArrAssignment {
private:
    int n;

public:
    ArrAssignment(int size) { n = size; }

    // Sorting function
    void sortArray(T arr[])
    {
        // to check for validity of sortness
        bool b = true;
        while (b) {
            b = false;
            for (int i = 0; i < n - 1; i++) {
                // sorting algorithm
                if (arr[i] > arr[i + 1]) {
                    T temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    b = true;
                }
            }
        }
    }

    // Display Function
    void printArray(T arr[])
    {
        for (int i = 0; i < n; ++i)
            cout << arr[i] << "\t";
        cout << endl;
    }

    // template<>
    friend ostream& operator+=<T>(ostream&, const ArrAssignment<T>&);
};

template <typename T>
ostream& operator+=(ostream& out, const ArrAssignment<T>& e)
{
    cout << "\nCurrent Array Size: ";
    return out << e.n * e.n;
}

// Code to drive the above template
int main()
{
    // To check for all numerical data types
    ArrAssignment<int> a(8); // 8, 9, 10 represent size of array
    ArrAssignment<float> b(9);
    ArrAssignment<double> c(10);

    // Testint the sorting algorithm through templates
    // upon the aforementioned data types
    int intArr[] = { 41, -55, 151, -171, 229, -31, 0, 455 };
    a.sortArray(intArr);
    cout << "\nSorted Int Array:\n";
    a.printArray(intArr);

    float floatArr[] = { 46.5, 73.6, 11.5, 87.6, 8.6, 52.8,
    42.8, 26.5, 1.1 };
    b.sortArray(floatArr);
    cout << "\nSorted Float Array:\n";
    b.printArray(floatArr);

    double doubleArr[] = { 12.553, 6458.2, 2, 91.23, 12.787,
    132.2, 0.00, 971.23, 12.551, 682.2 };
    c.sortArray(doubleArr);
    cout << "\nSorted Double Array:\n";
    c.printArray(doubleArr);

    // Operator overloading that returns the square
    // of number of elements present in each array
    cout << "\nArray Size of a: ";
    cout += a;
    cout << "\nArray Size of b: ";
    cout += b;
    cout << "\nArray Size of c: ";
    cout += c;

    return 0;
}
```

Console Output

```
PS D:\NUST\Semester 3\Object Oriented Programming\Assignments> cd "d:\NUST\Semester
3\Object Oriented Programming\Assignments\Assignment 3\" ; if ($?) { g++ template.cpp -
o template } ; if ($?) { .\template }

Sorted Int Array:
-171    -55     -31     0       41      151     229     455

Sorted Float Array:
1.1     8.6     11.5    26.5    42.8    46.5    52.8    73.6    87.6

Sorted Double Array:
0       2       12.551  12.553  12.787  91.23   132.2   682.2   971.23  6458.2

Array Size of a: 64
Array Size of b: 81
Array Size of c: 100
```

Conclusions

Upon comparison of both the implementations; the one through overloading and the one through templates, we deduce that templates indeed take less lines of code and a reduction of 58 lines was observed for implementing a code as simple as bubble sort. It can be argued that for a code of much greater complexity, templates would be equally more effective in reducing the code to its minimal form.