**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**

**001: // Program to sort an array of ten elements**

**002: // that can be either of int, float, double**

**003: // datatype, sorting would work regardless**

**004: // \*EXCLUDING CHARS as sorting an array of**

**005: // char is pointless**

**006:**

**007: // USING FUNCTION OVERLOADING**

**008:**

**009: #include <iostream>**

**010: using namespace std;**

**011:**

**012: class ArrAssignment**

**013: {**

**014: private:**

**015: int n;**

**016:**

**017: public:**

**018: ArrAssignment(int size)**

**019: {**

**020: n = size;**

**021: }**

**022:**

**023: // Bubble sort for int**

**024: void sortArray(int arr[])**

**025: {**

**026: // to check for validity of sortness**

**027: bool b = true;**

**028: while (b)**

**029: {**

**030: b = false;**

**031: for (int i = 0; i < n - 1; i++)**

**032: {**

**033: // sorting algorithm**

**034: if (arr[i] > arr[i + 1])**

**035: {**

**036: int temp = arr[i];**

**037: arr[i] = arr[i + 1];**

**038: arr[i + 1] = temp;**

**039: b = true;**

**040: }**

**041: }**

**042: }**

**043: }**

**044:**

**045: // Display Function for int**

**046: void printArray(int arr[])**

**047: {**

**048: for (int i = 0; i < n; ++i)**

**049: cout << arr[i] << "\t";**

**050: cout << endl;**

**051: }**

**052:**

**053: // Bubble sort for float**

**054: void sortArray(float arr[])**

**055: {**

**056: // to check for validity of sortness**

**057: bool b = true;**

**058: while (b)**

**059: {**

**060: b = false;**

**061: for (int i = 0; i < n - 1; i++)**

**062: {**

**063: // sorting algorithm**

**064: if (arr[i] > arr[i + 1])**

**065: {**

**066: int temp = arr[i];**

**067: arr[i] = arr[i + 1];**

**068: arr[i + 1] = temp;**

**069: b = true;**

**070: }**

**071: }**

**072: }**

**073: }**

**074:**

**075: // Display Function for float**

**076: void printArray(float arr[])**

**077: {**

**078: for (int i = 0; i < n; ++i)**

**079: cout << arr[i] << "\t";**

**080: cout << endl;**

**081: }**

**082:**

**083: // Bubble sort for double**

**084: void sortArray(double arr[])**

**085: {**

**086: // to check for validity of sortness**

**087: bool b = true;**

**088: while (b)**

**089: {**

**090: b = false;**

**091: for (int i = 0; i < n - 1; i++)**

**092: {**

**093: // sorting algorithm**

**094: if (arr[i] > arr[i + 1])**

**095: {**

**096: double temp = arr[i];**

**097: arr[i] = arr[i + 1];**

**098: arr[i + 1] = temp;**

**099: b = true;**

**100: }**

**101: }**

**102: }**

**103: }**

**104:**

**105: // Display Function for double arr**

**106: void printArray(double arr[])**

**107: {**

**108: for (int i = 0; i < n; ++i)**

**109: cout << arr[i] << "\t";**

**110: cout << endl;**

**111: }**

**112:**

**113: friend ostream& operator+=(ostream&, const ArrAssignment&);**

**114: };**

**115:**

**116: ostream& operator+=(ostream& out, const ArrAssignment& e)**

**117: {**

**118: cout << "\nCurrent Array Size: ";**

**119: return out << e.n \* e.n;**

**120: }**

**121:**

**122: // Code to drive the above class**

**123: int main()**

**124: {**

**125: // To check for all numerical data types**

**126: ArrAssignment a(8); // 8, 9, 10 represent size of array**

**127: ArrAssignment b(9);**

**128: ArrAssignment c(10);**

**129:**

**130: // Testint the sorting algorithm through function overloading**

**131: // upon the aforementioned data types**

**132: int intArr[] = { 41, -55, 151, -171, 229, -31, 0, 455 };**

**133: a.sortArray(intArr);**

**134: cout << "\nSorted Int Array:\n";**

**135: a.printArray(intArr);**

**136:**

**137: float floatArr[] = { 46.5, 73.6, 11.5, 87.6, 8.6, 52.8,**

**138: 42.8, 26.5, 1.1 };**

**139: b.sortArray(floatArr);**

**140: cout << "\nSorted Float Array:\n";**

**141: b.printArray(floatArr);**

**142:**

**143: double doubleArr[] = { 12.553, 6458.2, 2, 91.23, 12.787,**

**144: 132.2, 0.00, 971.23, 12.551, 682.2 };**

**145: c.sortArray(doubleArr);**

**146: cout << "\nSorted Double Array:\n";**

**147: c.printArray(doubleArr);**

**148:**

**149: // Operator overloading that returns the square**

**150: // of number of elements present in each array**

**151: cout << "\nArray Size of a: ";**

**152: cout += a;**

**153: cout << "\nArray Size of b: ";**

**154: cout += b;**

**155: cout << "\nArray Size of c: ";**

**156: cout += c;**

**157:**

**158: return 0;**

**159: }**

**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**

**001: // Program to sort an array of ten elements**

**002: // that can be either of int, float, double**

**003: // datatype, sorting would work regardless**

**004: // \*EXCLUDING CHARS as sorting an array of**

**005: // char is pointless**

**006:**

**007: // USING TEMPLATES**

**008:**

**009: #include <iostream>**

**010: using namespace std;**

**011:**

**012: // Template formed so that sorting of any**

**013: // type variable is possible**

**014:**

**015: template <typename T>**

**016: class ArrAssignment;**

**017:**

**018: template <typename T>**

**019: ostream& operator+=(ostream&, const ArrAssignment<T>&);**

**020:**

**021: template <typename T>**

**022: class ArrAssignment {**

**023: private:**

**024: int n;**

**025:**

**026: public:**

**027: ArrAssignment(int size) { n = size; }**

**028:**

**029: // Sorting function**

**030: void sortArray(T arr[])**

**031: {**

**032: // to check for validity of sortness**

**033: bool b = true;**

**034: while (b) {**

**035: b = false;**

**036: for (int i = 0; i < n - 1; i++) {**

**037: // sorting algorithm**

**038: if (arr[i] > arr[i + 1]) {**

**039: T temp = arr[i];**

**040: arr[i] = arr[i + 1];**

**041: arr[i + 1] = temp;**

**042: b = true;**

**043: }**

**044: }**

**045: }**

**046: }**

**047:**

**048: // Display Function**

**049: void printArray(T arr[])**

**050: {**

**051: for (int i = 0; i < n; ++i)**

**052: cout << arr[i] << "\t";**

**053: cout << endl;**

**054: }**

**055:**

**056: // template<>**

**057: friend ostream& operator+=<T>(ostream&, const ArrAssignment<T>&);**

**058: };**

**059:**

**060: template <typename T>**

**061: ostream& operator+=(ostream& out, const ArrAssignment<T>& e)**

**062: {**

**063: cout << "\nCurrent Array Size: ";**

**064: return out << e.n \* e.n;**

**065: }**

**066:**

**067: // Code to drive the above template**

**068: int main()**

**069: {**

**070: // To check for all numerical data types**

**071: ArrAssignment<int> a(8); // 8, 9, 10 represent size of array**

**072: ArrAssignment<float> b(9);**

**073: ArrAssignment<double> c(10);**

**074:**

**075: // Testint the sorting algorithm through templates**

**076: // upon the aforementioned data types**

**077: int intArr[] = { 41, -55, 151, -171, 229, -31, 0, 455 };**

**078: a.sortArray(intArr);**

**079: cout << "\nSorted Int Array:\n";**

**080: a.printArray(intArr);**

**081:**

**082: float floatArr[] = { 46.5, 73.6, 11.5, 87.6, 8.6, 52.8,**

**083: 42.8, 26.5, 1.1 };**

**084: b.sortArray(floatArr);**

**085: cout << "\nSorted Float Array:\n";**

**086: b.printArray(floatArr);**

**087:**

**088: double doubleArr[] = { 12.553, 6458.2, 2, 91.23, 12.787,**

**089: 132.2, 0.00, 971.23, 12.551, 682.2 };**

**090: c.sortArray(doubleArr);**

**091: cout << "\nSorted Double Array:\n";**

**092: c.printArray(doubleArr);**

**093:**

**094: // Operator overloading that returns the square**

**095: // of number of elements present in each array**

**096: cout << "\nArray Size of a: ";**

**097: cout += a;**

**098: cout << "\nArray Size of b: ";**

**099: cout += b;**

**100: cout << "\nArray Size of c: ";**

**101: cout += c;**

**099:**

**100: return 0;**

**101: }**

**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.