|  |  |  |
| --- | --- | --- |
| **Rubrics for Object Oriented Programming Lab** | | |
| **Lab #:** | **10** | |
| **Lab Title:** | **Templetes** | |
| **Submitted by:** | | |
| **Name** | | **Registration #** |
| **AMMAR**  **MUHAMMAD KALEEM ULLAh** | | **FA19-BCE-001**  **FA19-BCE-007** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics name & number** | | | **Marks** | | |
| **In-Lab** | | **Post-Lab** |
| **Engineering Knowledge** | ***R2: Use of Engineering Knowledge and follow Experiment Procedures:***  *Ability to follow experimental procedures, control variables, and record procedural steps on lab report.* | |  | | |
| **Problem Analysis** | | ***R5: Data/Evidence Measurements:***  *Ability to record raw data / evidence.* | |  | |
| **Design** | | ***R8: Best Coding Standards:***  *Ability to follow the coding standards and programming practices.* | |  | |
| **Modern Tools Usage** | | ***R9: Understand Tools:*** *Ability to describe and explain the principles behind and applicability of engineering tools.* | |  | |
| **Individual and Teamwork** | | ***R12: Individual Work Contributions:*** *Ability to carry out individual responsibilities.* | |  | |
| ***R13: Management of Team Work:***  *Ability to appreciate, understand and work with multidisciplinary team members.* | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rubrics #** | R2 | R5 | R8 | R9 | R12 | R13 |
| **In –Lab** |  |  |  |  |  |  |
| **Post- Lab** |  |  |  |  |  |  |

**Lab 10**

**Templates**

1. **Objectives**

To familiarize the students with the concept of templates and how to use one template function to handle many different data types.

1. **Outcome**

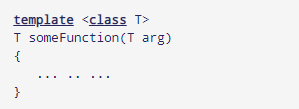
After this lab the students should be able to create template functions and write a single function definition that works with many different data types, instead of having to write a separate function for each data type used.

1. **Introduction** 
   1. **Function Templates:**

A function template is a “generic” function that can work with any data type. The programmer writes the specifications of the function, but substitutes parameters for data types. When the compiler encounters a call to the function, it generates code to handle the specific data type(s) used in the call.

* 1. **How to declare a function template?**

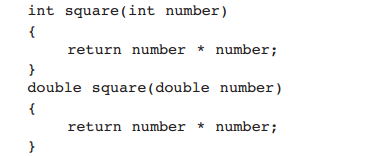
A function template starts with the keyword template followed by template parameter/s inside < > which is followed by function declaration.



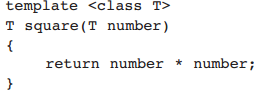
In the above code, T is a template argument that accepts different data types (int, float), and class is a keyword. When, an argument of a data type is passed to someFunction( ), compiler generates a new version of someFunction() for the given data type.

* 1. **Template Function v/s Normal Function:**

A function template works in a similar to a normal function, with one key difference. A single function template can work with different data types at once but, a single normal function can only work with one set of data types.



Here is a function template for the square function:



1. **In-Lab Tasks**

**5.1** Write templates for the two functions minimum and maximum. The minimum function should accept two arguments and return the value of the argument that is the lesser of the two. The maximum function should accept two arguments and return the value of the argument that is the greater of the two. Design a simple program that demonstrates the templates with data types int, char, float and strings.

**Solution:**

* **Code:**

#include<iostream>

using namespace std;

template <class type>

type minimum(type n1,type n2)

{

if(n1>n2)

{

return n2;

}

else

{

return n1;

}

}

template <class type>

type maximum(type n1,type n2)

{

if(n1<n2)

{

return n2;

}

else

{

return n1;

}

}

int main()

{

cout<<"minimum value : "<<minimum(3,2);

cout<<endl<<"Maximum value : "<<maximum(3,2);

cout<<endl<<"Character input : "<<minimum('c','d');

cout<<endl<<"floating input : "<<maximum(4.5,3.5);

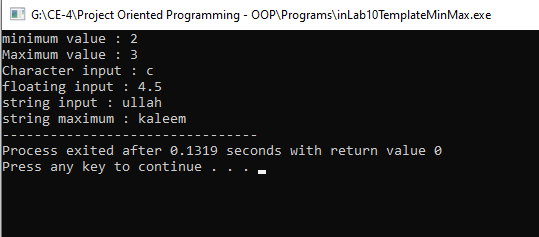
cout<<endl<<"string input : "<<minimum("kaleem","ullah");

cout<<endl<<"string maximum : "<<maximum("kaleem ","ullah");

return 0;

}

* **Output:**

****

**5.2** Write a template for a function called total. The function should keep a running total of values entered by the user, then return the total. The argument sent into the function should be the number of values the function is to read. Test the template in a simple driver program that sends values of various types as arguments and displays the results.

**Solution:**

* **Code:**

#include <iostream>

using namespace std;

template <typename T>

T total(int n, T value)

{

T total;

cin >> value;

total = value;

for(int i = 1; i < n; i++)

{

cin >> value;

total += value;

}

return total;

}

int main()

{

int num;

cout << "How many values you want to enter .... "; cin >> num;

int t1 = total(num, 1);

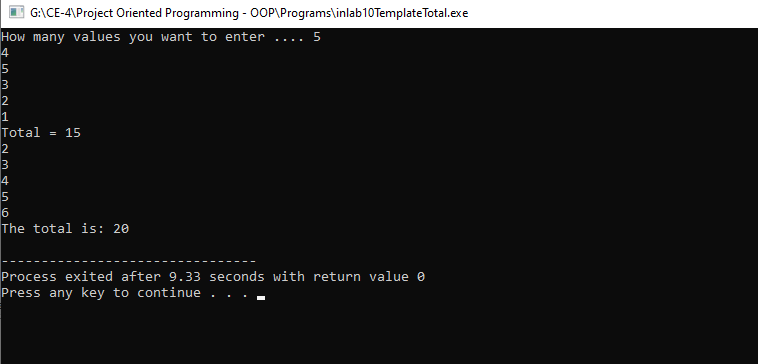
cout << "Total = " << t1 << endl;

double t2 = total(num, 1.0);

cout << "The total is: " << t2 << endl;

}

* **Output:**

****

1. **Post Lab Tasks :**

**6.1** Write a template function that returns the average of all the elements of an array. The arguments to the function should be the array name and the size of the array (type int). In main() , exercise the function with arrays of type int, long, double, and char.

**Code:**

#include <iostream>

using namespace std;

template <class Mean>

Mean Average(Mean \*Name,int Size)

{

Mean Avg=0;

for(int i=0;i<Size;i++)

{

Avg=Name[i]+Avg;

}

return Mean (Avg/Size);

}

int main()

{

// All Arrays Of Differnet Types

int Int\_Array[]={1,2,3,4,5,6,7,8,9,10};

char Char\_Array[]={10,20,30,40,50,60,70,80,90,100};

double Double\_Array[]={1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.15};

long Long\_Array[]={1L,2l,3L,4L,5L,6L,7L,8L,9L,10L};

// Finding There Average

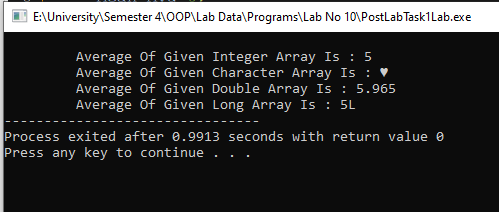
cout<<"\n\t Average Of Given Integer Array Is : "<<Average(Int\_Array,10);

cout<<"\n\t Average Of Given Character Array Is : "<<Average(Char\_Array,10);

cout<<"\n\t Average Of Given Double Array Is : "<<Average(Double\_Array,10);

cout<<"\n\t Average Of Given Long Array Is : "<<Average(Long\_Array,10)<<"L";

}



**Conclusions:**

* Use **templates** in situations that result in duplication of the same code for multiple types.
* You can also use class **templates** to develop a set of typesafe classes.
* **Templates** are sometimes a better solution than C macros and void pointers,