# Lab 12

**Function & Class Templates**

##### Objectives

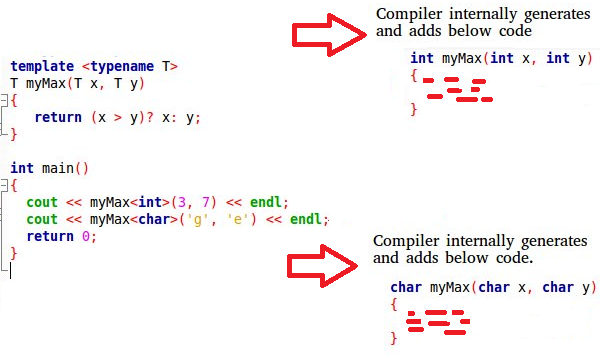
Following programming skills will be acquired in this lab:

* To understand the use of a template.
* To practice the syntax of defining a function template.
* To understand and use class templates.

##### Function Templates

Function templates are special functions that can operate with *generic types*. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

In C++ this can be achieved using *template parameters*. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.



The format for declaring function templates with type parameters is:

template <class identifier>

function\_declaration;

template <typename identifier>

function\_declaration;

The only difference between both prototypes is the use of either the keyword class or the keyword typename. Its use is indistinct, since both expressions have exactly the same meaning and behave exactly the same way.

template <class myType>

myType GetMax(myType a, myType b) { return (a>b?a:b);

}

Here we have created a template function with myType as its template parameter. This template parameter represents a type that has not yet been specified, but that can be used in the template function as if it were a regular type. As you can see, the function template GetMax returns the greater of two parameters of this still-undefined type.

To use this function template, we use the following format for the function call:

function\_name <type> (parameters);

For example, to call GetMax to compare two integer values of type int we can write:

int x,y;

GetMax <int> (x,y);

**“When the compiler encounters this call to a template function, it uses the template to automatically generate a function replacing each appearance of myType by the type passed as the actual template parameter (int in this case) and then calls it. This process is automatically performed by the compiler and is invisible to the programmer.”**

Example 12.1

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  template <class T>  T GetMax(T a, T b) {  return (a > b ? a : b);  } | int main()  {  int i = 5,j=6,k;  long l=10, m=5,n;  k = GetMax(i, j);  n = GetMax(l, m);  cout << k << endl; cout << n << endl;  return 0;  } |

Example 12.2

|  |
| --- |
| #include <iostream>  using namespace std;  //function returns index number of item  //returns -1 if not found  template <class atype>  int find(atype\* array, atype value, int size)  {  for (int j = 0; j < size; j++)  {  if (array[j] == value)  { return j; }  }  return -1;  } |
| int main()  {  char chrArr[] = { 1,3,5,9,11,13 }; //array  char ch = 5; //value to find  int intArr[] = { 1,3,5, 9,11,13 };  int in = 6;  long lonArr[] = { 1L, 3L, 5L, 9L, 11L, 13L };  long lo = 11L;  double dubArr[] = { 1.0, 3.0, 5.0, 9.0, 11.0, 13.0 };  double db = 4.0;  cout << "\n 5 in chrArray: index=" << find(chrArr, ch, 6);  cout << "\n 6 in intArray: index=" << find(intArr, in, 6);  cout << "\n 11 in lonArray: index=" << find(lonArr, lo, 6);  cout << "\n 4 in dubArray: index=" << find(dubArr, db, 6);  cout << endl;  system("pause");  return 0;  } |

Text

Description automatically generated

**Class Templates**

We also have the possibility to write class templates, so that a class can have members that use template parameters as types. For example:

template <class T> class mypair {

T values [2]; public:

mypair (T first, T second)

{

values[0]=first; values[1]=second;

}

};

The class that we have just defined serves to store two elements of any valid type. For example, if we wanted to declare an object of this class to store two integer values of type int with the values 115 and 36 we would write:

mypair<int> myobject (115, 36);

this same class would also be used to create an object to store any other type:

mypair<double> myfloats (3.0, 2.18);

The only member function in the previous class template has been defined inline within the class declaration itself. In case that we define a function member outside the declaration of the class template, we must always precede that definition with the template <...> prefix.

##### Example 12.3

|  |  |
| --- | --- |
| #include <iostream>  using namespace std;  template <class T>  class mypair  {  private:  T a, b;  public:  mypair(T first, T second)  {  a = first;  b = second;  }  T getmax();  }; | template <class T>  T mypair<T>::getmax()  {  T retval;  retval = a > b ? a : b;  return retval; } int main()  {  mypair <int> myobject(100, 75);  cout << myobject.getmax();  system("pause");  return 0;  } |

Example 12.4

|  |  |
| --- | --- |
| #include <iostream> using namespace std;  const int MAX = 100; //size of array  template <class Type>  **class Stack**  {  private:  Type st[MAX]; //stack: array of any type int top; //number of top of stack public:  Stack()  { top = -1; }  **void push(Type var)**  {  st[++top] = var;  }  **Type pop()**  {  return st[top--]; }  }; | int **main**()  {  //s1 is object of class Stack<float> Stack<float> s1;  //push 3 floats, pop 3 floats s1.push(1111.1F); s1.push(2222.2F); s1.push(3333.3F);  cout << "1: "<< s1.pop() << endl; cout << "2: " << s1.pop() << endl; cout << "3: " << s1.pop() << endl;  //s2 is object of class Stack<long> Stack<long> s2;  //push 3 longs, pop 3 longs s2.push(123123123L); s2.push(234234234L); s2.push(345345345L);  cout << "1: " << s2.pop() << endl; cout << "2: " << s2.pop() << endl; cout << "3: " << s2.pop() << endl; system("pause");  return 0;  } |

**Non-type Parameters for Template**

Besides the template arguments that are preceded by the class or typename keywords , which represent types, templates can also have regular typed parameters, similar to those found in functions.

##### Example 12.5

|  |  |
| --- | --- |
| // sequence template  #include <iostream>  using namespace std;  template <class T, int N>  **class mysequence** {  T memblock [N];  public:  void setMember (int x, T value);  T getMember (int x);  };  template <class T, int N>  void mysequence<T,N>::**setMember** (int x, T value)  { memblock[x]=value; }  template <class T, int N>  T mysequence<T,N>::**getMember** (int x)  { return memblock[x]; } | int **main** () {  mysequence <int,5> myints; mysequence <double,5> myfloats;  myints.setMember (0,100);  myfloats.setMember (3,3.1416);  cout << myints.getMember(0) << '\n'; cout << myfloats.getMember(3) << '\n';  return 0;  } |

**Try these Codes too:**

**Code 1:**

#include <iostream>

using namespace std;

template<class T>

void add(T x, T y)

{

cin >> x >> y;

cout << x + y << endl;

}

int main()

{

add(1, 2);

return 0;

}

**Code 2:**

#include <iostream>

using namespace std;

template< typename T>

void InputArray( T \*array, int count )

{

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

cin>>array[ i ] ;

}

template< typename T>

void displayArray( T \*array, int count )

{

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

cout<<array[ i ] ;

}

int main()

{

int a[5];

float b[5];

cout << "Array a contains:" << endl;

// call integer function-template specialization

InputArray( a, 5 );

displayArray( a, 5 );

cout<<"float type array";

InputArray( b, 5 );

displayArray( b, 5 );

return 0;

}

**Lab 12 Exercises**

**Exercise 1:**

Using the C++ template feature, code the following template based functions. A function findAverage( ) which accepts an array of an arbitrary data type and the number of elements in the array and returns the average of the elements in the array.

A function minMax( ) which finds the minimum and maximum values in the array as well as their indices. (You need to print all these values in the main( ) so you may need call by reference) .

In the main program, declare an array of 10 integers and 10 floats, call the above functions and display their outputs in the main.

Code:

#include <iostream>

using namespace std;

template<typename hehe>

hehe findAverage(hehe\*arr,int size){

hehe sum=0;

for (int i = 0; i < size; i++) {

sum = sum + arr[i];

}

return sum / size;

}

template <typename hehe>

hehe minMax(hehe\* arr, int size,hehe &min,hehe &max) {

min=arr[0];

max = arr[0];

for (int i = 0; i < size; i++) {

if (arr[i] < min) {

min = arr[i];

}

}

for (int i = 0; i < size; i++) {

if (arr[i] > max) {

max = arr[i];

}

}

return min;

}

int main()

{

const int size = 10;

int min, max;

int arr[size] = { 20,10,50,60,80,90,100,40,12,2 };

float arr1[size] = { 25.0,25.6,50.2,60.12,80.25,90.91,100.01,40.45,12.88,36.5 };

float min1, max1;

cout << "The average values of the integer array is :" << findAverage(arr, size) << endl;

cout << "The average values of the float array is :" << findAverage(arr1, size) << endl;

minMax(arr1, size, min1, max1);

minMax(arr, size, min, max);

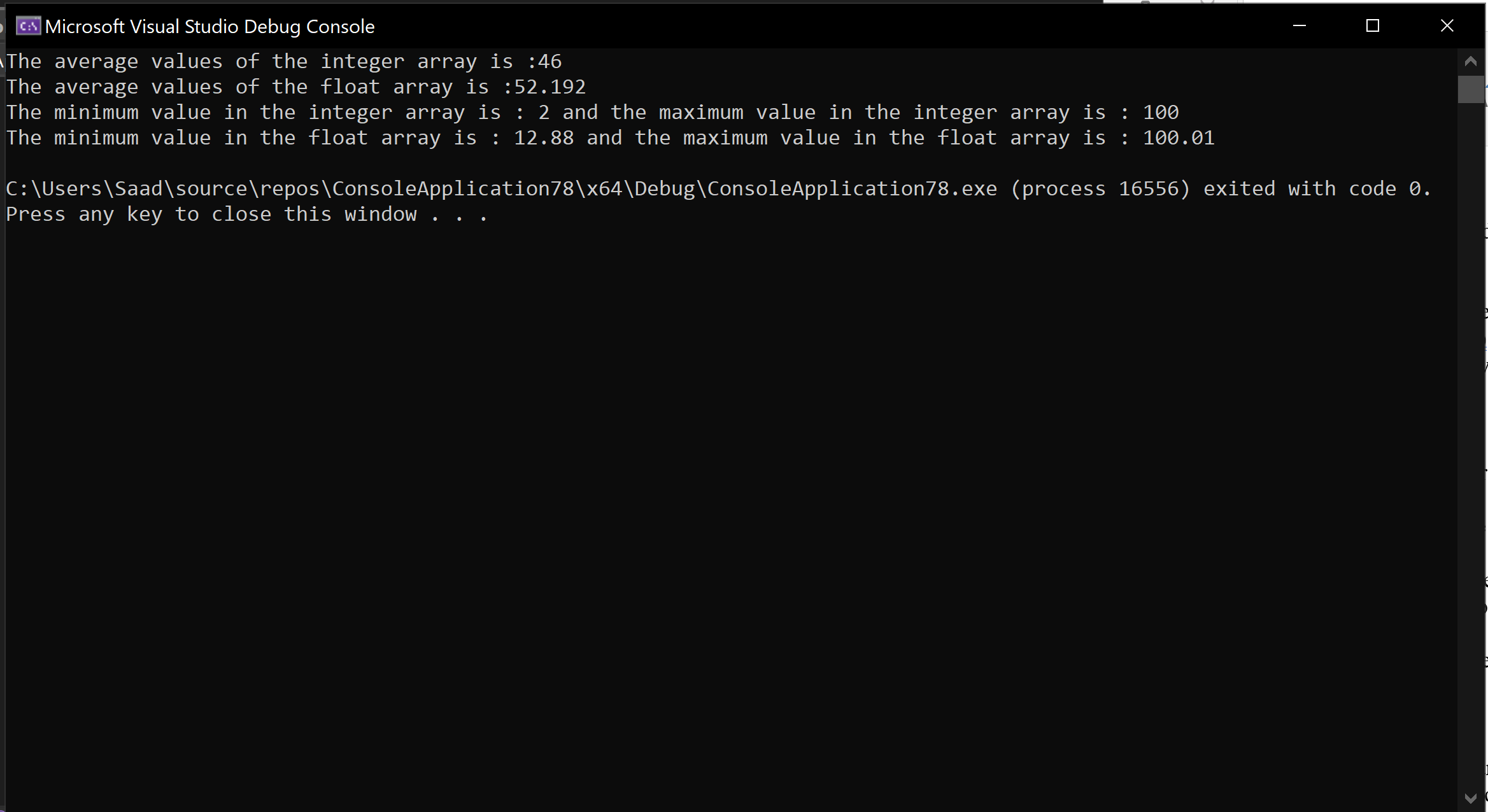
cout << "The minimum value in the integer array is : " << min << " and the maximum value in the integer array is : " << max << endl;

cout << "The minimum value in the float array is : " << min1 << " and the maximum value in the float array is : " << max1 << endl;

return 0;

}

Output:



**Exercise 2:**

Using template feature of C++, program a class ‘Calculator’ with two data members of the user specified data type. The class should have a constructor to initialize these data members and the following functions. (Both function definitions are to be provided outside the class) .

a. A function to add the two data members

b. A function to multiply the data members

From your main program create objects of calculator for integer and float data members, call these functions and display the results.

Code:

#include <iostream>

using namespace std;

template <class hehe>

class Calculator {

hehe a1;

hehe a2;

public:

Calculator(hehe a, hehe b);

hehe multiply();

hehe add();

};

template<class hehe>

Calculator<hehe>::Calculator(hehe a, hehe b) {

a1 = a;

a2 = b;

}

template<class hehe>

hehe Calculator<hehe>::add() {

return a1 + a2;

}

template<class hehe>

hehe Calculator<hehe>::multiply() {

return a1 \*a2;

}

int main() {

Calculator<int> c1(10,20);

Calculator<float> c2(10.5, 20.5);

cout << "Sum : " << c1.add() << endl;

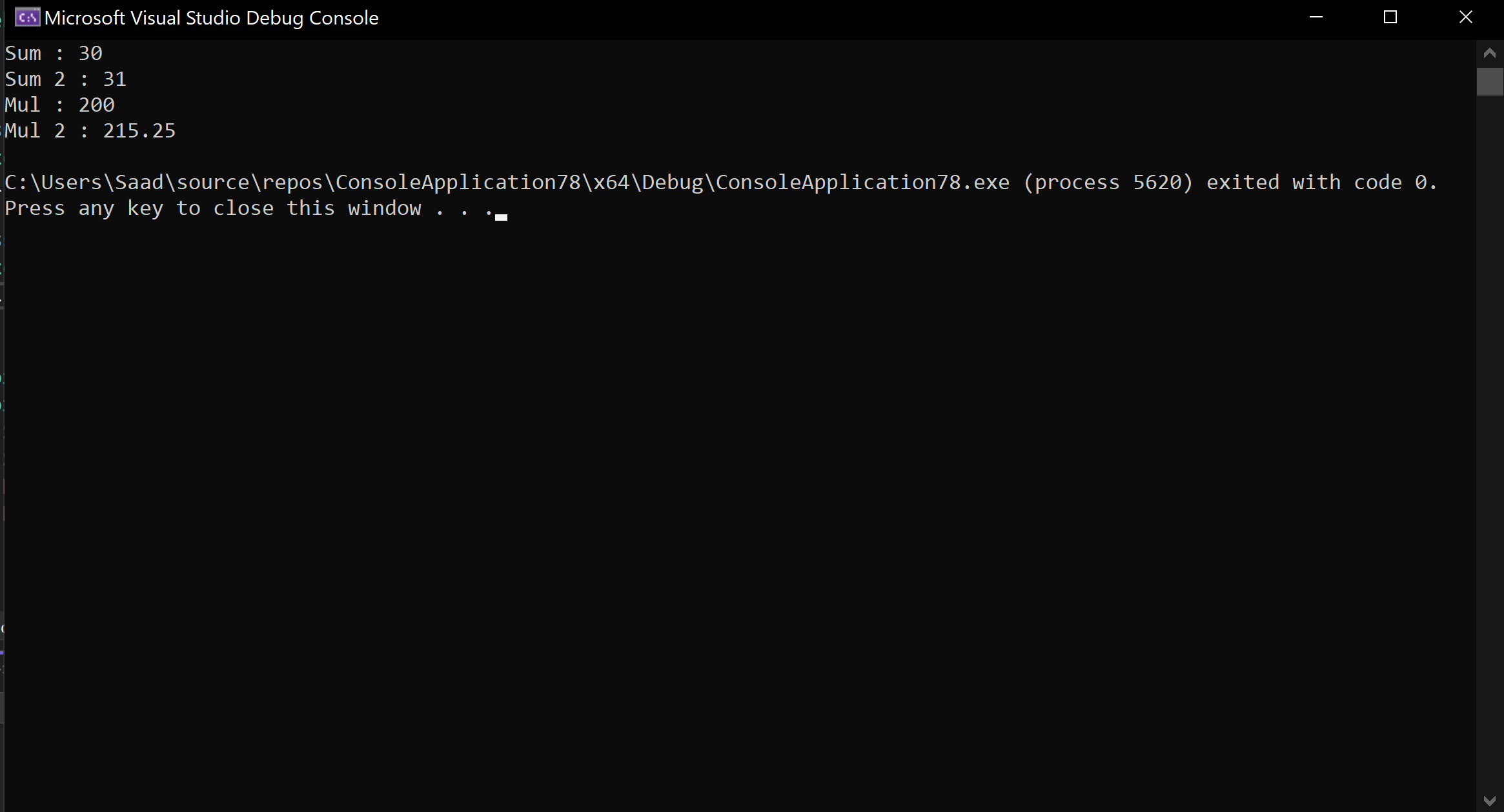
cout << "Sum 2 : " << c2.add() << endl;

cout << "Mul : " << c1.multiply() << endl;

cout << "Mul 2 : " << c2.multiply() << endl;

}

Output:



**Exercise 3:**

Create a template-based class Stack and provide the standard operations of push, pop, top and empty. Also provide a display method to print the elements of the stack on the screen.

In the main program, create a stack of integers and another of floats and call some of its methods.

#include <iostream>

using namespace std;

const int length = 3;

template<class hehe>

class Stack {

hehe arr[length];

int top;

public:

Stack() {

top = -1;

}

void push(hehe value) {

arr[++top] = value;

}

hehe pop() {

return arr[top--];

}

void empty() {

for (int i = 0; i <= top; i++) {

this->pop();

}

}

void display() {

for (int i = 0; i <= top; i++) {

cout << arr[i] << endl;

}

}

void showtop() {

cout << "Top : " << top << " Value : "<< arr[top] << endl;

}

};

int main() {

Stack<int> s1;

s1.push(1);

s1.push(2);

s1.push(3);

s1.display();

cout << s1.pop() << endl;

s1.display();

s1.showtop();

cout << "Stack 2 : " << endl;

Stack<float> s2;

s2.push(1.11);

s2.push(2.22);

s2.push(3.33);

s2.display();

cout << s2.pop() << endl;

s2.display();

}

Output:

