CC-211L

Object Oriented Programming

Laboratory 15

Custom Templates

Version: 1.0.0

Release Date: 06-05-2023

Department of Information Technology

University of the Punjab

Lahore, Pakistan

Contents:

Learning Objectives:

- Class Templates
- Function Templates
- Arguments to Templates
- Overloading Function Templates

Resources Required:

- Desktop Computer or Laptop
- Microsoft ® Visual Studio 2022

General Instructions:

- In this Lab, you are **NOT** allowed to discuss your solution with your colleagues, even not allowed to ask how is s/he doing, this may result in negative marking. You can **ONLY** discuss with your Teaching Assistants (TAs) or Lab Instructor.
- Your TAs will be available in the Lab for your help. Alternatively, you can send your queries via email to one of the followings.

Teachers:		
Course Instructor	Prof. Dr. Syed Waqar ul Qounain	swjaffry@pucit.edu.pk
Lab Instructor	Azka Saddiqa	azka.saddiqa@pucit.edu.pk
Teacher Assistants	Saad Rahman	bsef19m021@pucit.edu.pk
	Zain Ali Shan	bcsf19a022@pucit.edu.pk

Background and Overview:

Templates:

Templates is defined as a blueprint or formula for creating a generic class or a function. Generic Programming is an approach to programming where generic types are used as parameters in algorithms to work for a variety of data types. A template is a straightforward yet effective tool. To avoid having to write the same code for many data types, the simple concept is to pass the data type as a parameter.

Templates works in such a way that it gets expanded at compiler time, just like macros and allows a function or class to work on different data types without being rewritten.

Function Template:

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.

Class Template:

Template classes are classes that can have members of the generic type, i.e., members that use template parameters as types.

Activities:

Pre-Lab Activities:

Function Template:
Generic functions define a set of operations that can be applied to the various types of data. The type of the data that the function will operate on depends on the type of the data passed as a parameter. A Generic function is created by using the keyword template. The template defines what function will do.

Syntax:
template < class Ttype> return_type func_name(parameter_list)

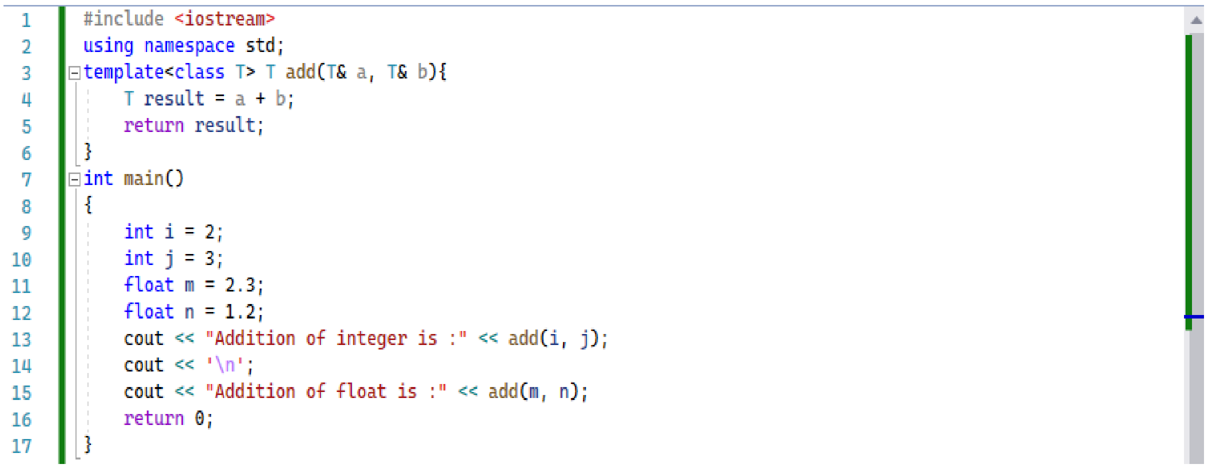
{

// body of function.

}

Ttype is a placeholder name for a data type used by the function. It is used within the function definition. It is only a placeholder that the compiler will automatically replace this placeholder with the actual data type. A **"class"** keyword is used to specify a generic type in a template declaration.

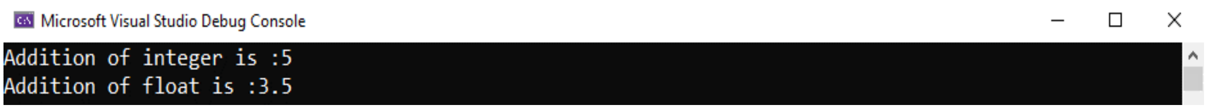
Example:



```cpp
#include <iostream>
using namespace std;
template<class T> T add(T& a, T& b){
    T result = a + b;
    return result;
}
int main()
{
    int i = 2;
    int j = 3;
    float m = 2.3;
    float n = 1.2;
    cout << "Addition of integer is :" << add(i, j);
    cout << '\n';
    cout << "Addition of float is :" << add(m, n);
    return 0;
}
```

Fig. 01 (Function Template)

Output:



```
Addition of integer is :5
Addition of float is :3.5
```

Fig. 02 (Function Template)

Function Template with Multiple Parameters:
We can use more than one generic type in the template function by using the comma to separate the list.

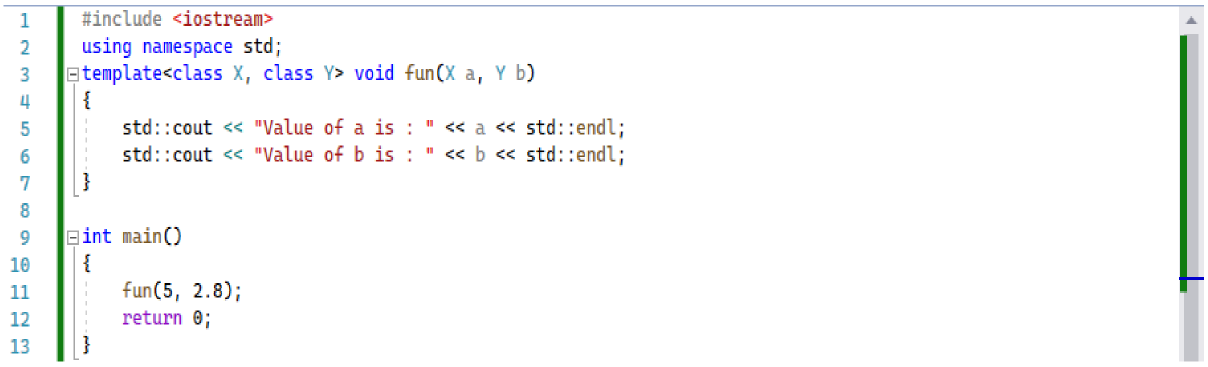
Syntax:
template<class T1, class T2,.....>

return_type function_name (arguments of type T1, T2....) {

// body of function.

}

Example:

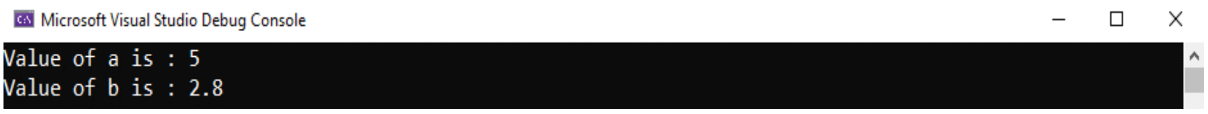


```cpp
#include <iostream>
using namespace std;
template<class X, class Y> void fun(X a, Y b)
{
    std::cout << "Value of a is : " << a << std::endl;
    std::cout << "Value of b is : " << b << std::endl;
}

int main()
{
    fun(5, 2.8);
    return 0;
}
```

Fig. 03 (Function Template with Multiple Parameters)

Output:



```
Value of a is : 5
Value of b is : 2.8
```

Fig. 04 (Function Template with Multiple Parameters)

Task 01: Maximum **[Estimated time 20 minutes / 15 marks]**

Write a program that:

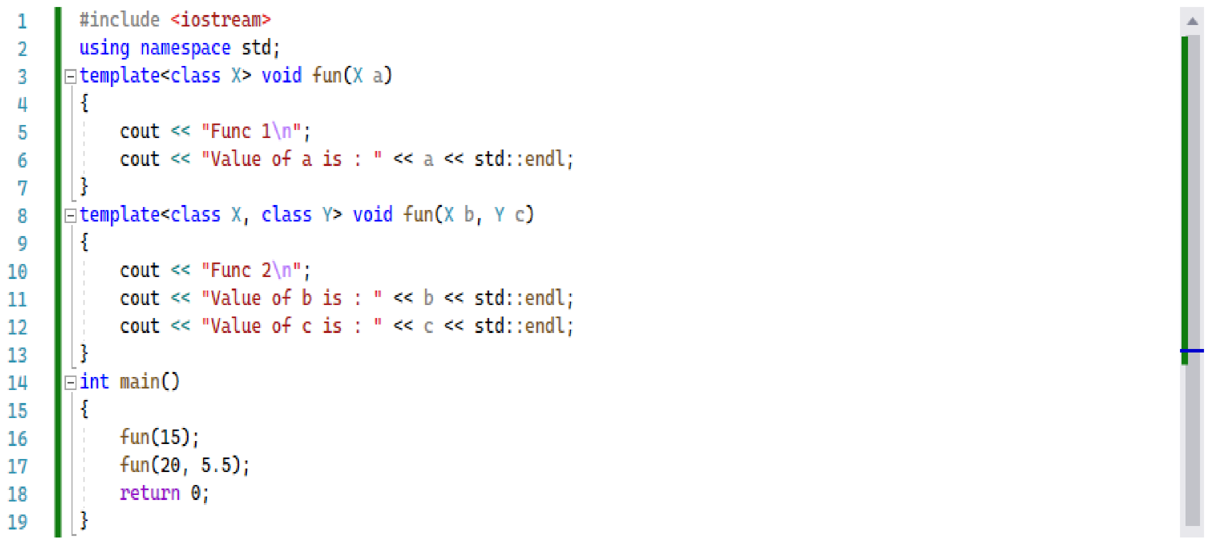
- Declare the function template for finding the maximum value which takes three arguments
- In the main function, declare three variables of different data types - int and float.
- Call the function template twice - once with the int values and once with the float values
- Display the maximum number on the Console

In-Lab Activities:

Overloading Function Template:

We can overload the function template means that overloaded functions can differ in the number of parameters.

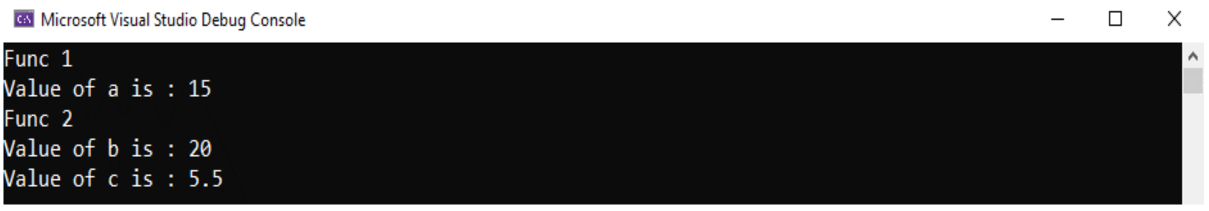
Example:



```cpp
#include <iostream>
using namespace std;
template<class X> void fun(X a)
{
    cout << "Func 1\n";
    cout << "Value of a is : " << a << std::endl;
}
template<class X, class Y> void fun(X b, Y c)
{
    cout << "Func 2\n";
    cout << "Value of b is : " << b << std::endl;
    cout << "Value of c is : " << c << std::endl;
}
int main()
{
    fun(15);
    fun(20, 5.5);
    return 0;
}
```

Fig. 05 (Overloading Function Template)

Output:



```
Func 1
Value of a is : 15
Func 2
Value of b is : 20
Value of c is : 5.5
```

Fig. 06 (Overloading Function Template)

Class Template:

Class templates can also be defined similarly as function templates.

Syntax:

template<class Ttype>

class class_name

{

.

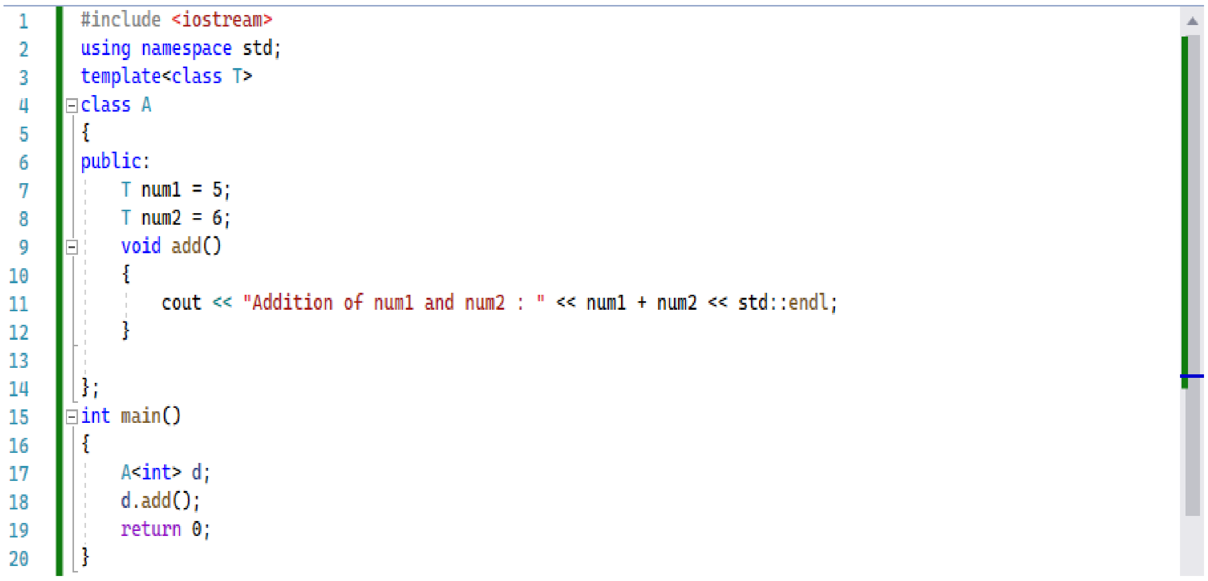
}

To create instance of a class following syntax is used:

class_name<type> ob;

where class_name is the name of the class, type is the type of data the class is operating on and ob is the name of the object.

Example:

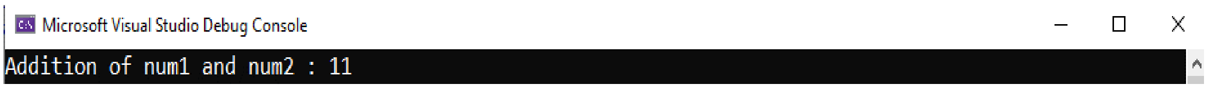


```
#include <iostream>
using namespace std;
template<class T>
class A
{
public:
    T num1 = 5;
    T num2 = 6;
    void add()
    {
        cout << "Addition of num1 and num2 : " << num1 + num2 << std::endl;
    }

};
int main()
{
    A<int> d;
    d.add();
    return 0;
}
```

Fig. 07 (Class Template)

Output:



```
Addition of num1 and num2 : 11
```

Fig. 08 (Class Template)

Class Template with Multiple Parameters:

We can use more than one generic data type in a class template, and each generic data type is separated by the comma.

Syntax:

template<class T1, class T2, ...>

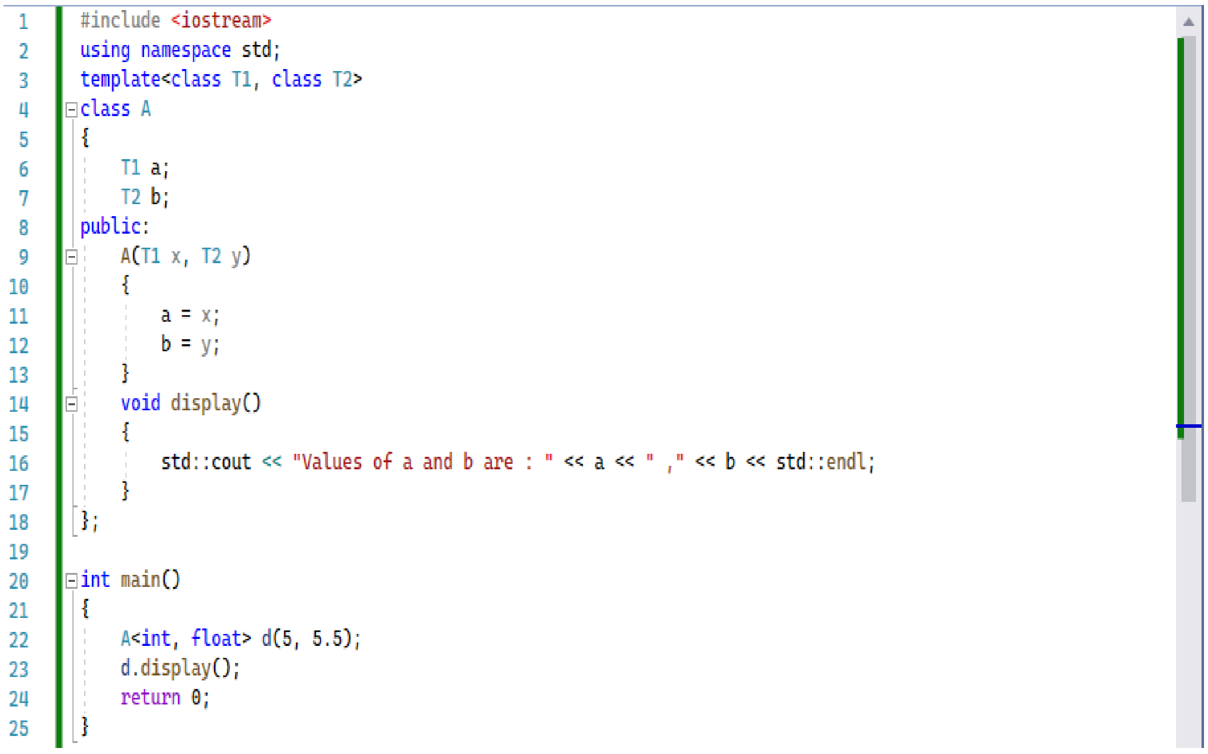
class class_name

{

// Body of the class.

}

Example:

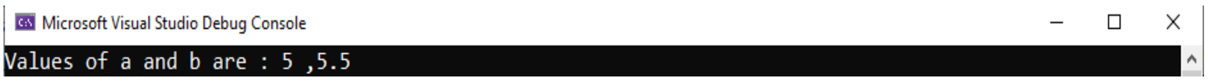


```
#include <iostream>
using namespace std;
template<class T1, class T2>
class A
{
    T1 a;
    T2 b;
public:
    A(T1 x, T2 y)
    {
        a = x;
        b = y;
    }
    void display()
    {
        std::cout << "Values of a and b are : " << a << " ," << b << std::endl;
    }
};

int main()
{
    A<int, float> d(5, 5.5);
    d.display();
    return 0;
}
```

Fig. 09 (Class Template with Multiple Parameters)

Output:



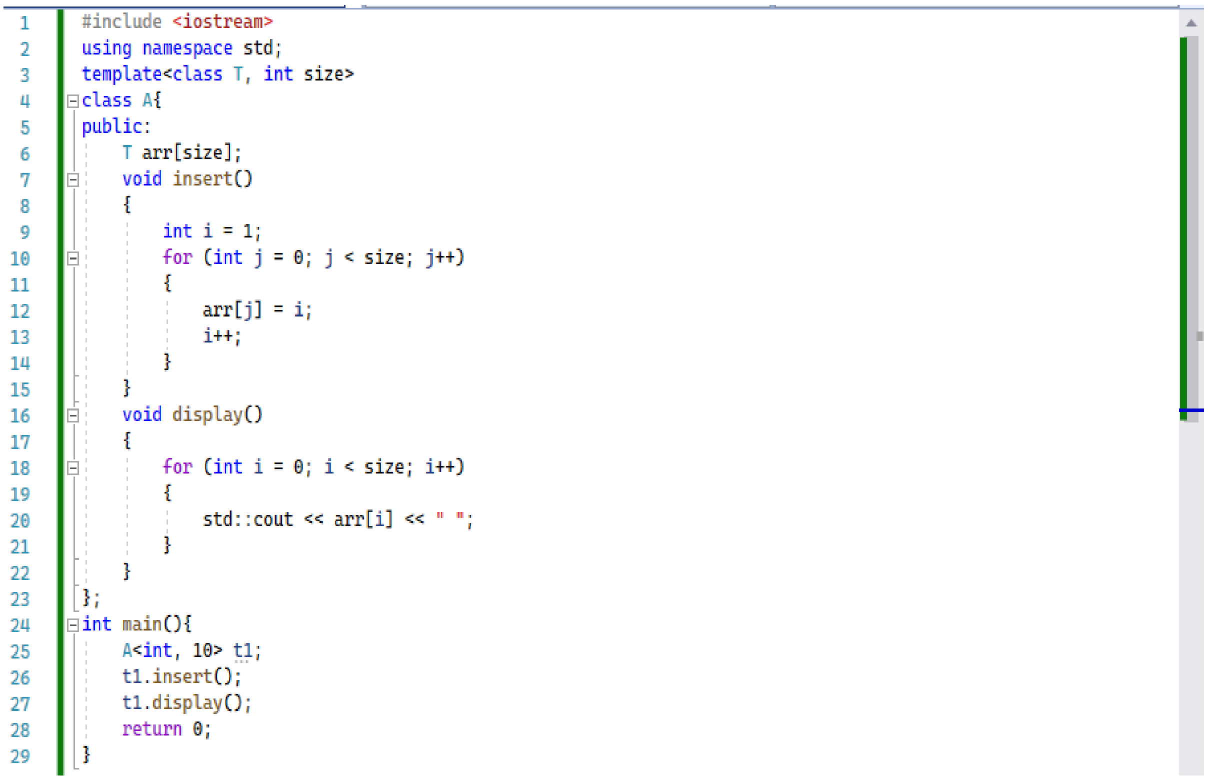
```
Values of a and b are : 5 ,5.5
```

Fig. 10 (Class Template with Multiple Parameters)

Non-type Template Parameters:

The template can contain multiple arguments, and we can also use the non-type arguments In addition to the type T argument, we can also use other types of arguments such as strings, function names, constant expression and built-in types.

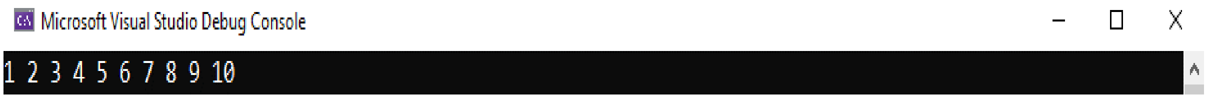
Example:



```
#include <iostream>
using namespace std;
template<class T, int size>
class A{
public:
    T arr[size];
    void insert()
    {
        int i = 1;
        for (int j = 0; j < size; j++)
        {
            arr[j] = i;
            i++;
        }
    }
    void display()
    {
        for (int i = 0; i < size; i++)
        {
            std::cout << arr[i] << " ";
        }
    }
};
int main(){
    A<int, 10> t1;
    t1.insert();
    t1.display();
    return 0;
}
```

Fig. 11 (Non-type Template Parameters)

Output:



Microsoft Visual Studio Debug Console

```
1 2 3 4 5 6 7 8 9 10
```

Fig. 12 (Non-type Template Parameters)

Task 01: Sum of Arrays **[Estimated time 20 minutes / 20 marks]**

Write a program that:

- Declare the function template for finding the sum of two arrays
- This function template takes three arguments - two arrays of the same data type and the size of the arrays. It returns a new array that contains the sum of corresponding elements of the two input arrays
- In the main function, declare two arrays of different data types - int and float
- Call the function template twice - once with the int arrays and once with the float arrays
- Display the output array on the Console

Task 02: Matrix Operations **[Estimated time 30 minutes / 30 marks]**

Write a program that:

- Declare the function templates to add, multiply and subtract two matrices of different data types.
- These function templates take two matrices of types T1 and T2, their size, and creates a new matrix of type T1 to hold the result
- Overload the function templates to add, multiply and subtract two matrices of the same data type
- These overloaded function templates take two matrices of type T, their size, and creates a new matrix of type T to hold the result.
- Show the result on the console

Task 03: Vending Machine **[Estimated time 50 minutes / 50 marks]**

- Create a class template for a vending machine that can dispense different types of items, such as drinks, snacks, and candy. The vending machine should have the following characteristics:
  - It should have a variable for storing the current balance of the user
  - It should have a method for accepting money from the user and adding it to the current balance
  - It should have a method for displaying the current balance to the user
  - It should have a method for dispensing an item to the user, based on the user's current balance and the price of the item
  - The class template should be able to handle any type of item, as long as the item has a name and a price
- Your task is to create a class template that can be used to implement a vending machine for any type of item. The vending machine should have the following methods:
  - **VendingMachine() -** A constructor that initializes the current balance to zero.
  - **void acceptMoney(T amount) -** A method that accepts money from the user and adds it to the current balance. The parameter amount is the amount of money being added to the balance.
  - **T getCurrentBalance() -** A method that returns the current balance of the user.
  - **bool dispenseItem(T price, const std::string& name) -** A method that dispenses an item to the user, based on the user's current balance and the price of the item. If the user has enough money to purchase the item, the method should deduct the price of the item from the current balance and return true. If the user does not have enough money to purchase the item, the method should return false and not deduct any money from the current balance. The name parameter is the name of the item being dispensed and is used for display purposes only.

- You should also create a main program to test the vending machine class template. Your program should do the following:
    - Create an instance of the vending machine class template for each type of item that the vending machine can dispense.
    - Accept money from the user and add it to the current balance of each vending machine instance.
    - Display the current balance of each vending machine instance.
    - Dispense an item from each vending machine instance, based on the user's current balance and the price of the item.
    - Display the current balance of each vending machine instance after each item is dispensed.

Post-Lab Activities:

Task 01: Custom Array [Estimated time 60 minutes / 40 marks]

- Create a class template called **“Array”** that will represent an array of elements of any data type. The class should have the following properties:
  - A private array of template type elements to store the data
  - A private array of template type elements to store the data
- The class should have the following methods:
  - A constructor that takes a single integer parameter to initialize the size of the array
  - A destructor that frees the memory allocated for the array
  - A copy constructor that creates a new object and initializes it with the values of an existing object
  - A move constructor that creates a new object and "steals" the resources (i.e., the dynamically allocated memory) from an existing object
  - An assignment operator that assigns the values of one object to another
  - An index operator that returns the value of the element at a given index
  - A function to get the size of the array
  - A function to set the value of an element at a given index
  - A function to get the value of an element at a given index
  - A function to resize the array, either increasing or decreasing its size
- The implementation of the class should use dynamic memory allocation to allocate the memory for the array

Submissions:

- For In-Lab Activity:
  - Save the files on your PC.
  - TA’s will evaluate the tasks offline.
- For Pre-Lab & Post-Lab Activity:
  - Submit the .cpp file on Google Classroom and name it to your roll no.

Evaluations Metric:

- All the lab tasks will be evaluated offline by TA’s

Division of Pre-Lab marks:	**[15 marks]**
Task 01: Maximum	[15 marks]
Division of In-Lab marks:	**[100 marks]**
Task 01: Sum of Arrays	[20 marks]
Task 02: Matrix Operations	[30 marks]
Task 03: Vending Machine	[50 marks]
Division of Post-Lab marks:	**[40 marks]**
Task 01: Custom Array	[40 marks]

References and Additional Material:

- **C++ Templates**
  https://www.mygreatlearning.com/blog/templates-in-cpp/

Lab Time Activity Simulation Log:

• Slot – 01 – 00:00 – 00:15:	Class Settlement
• Slot – 02 – 00:15 – 00:40:	In-Lab Task
• Slot – 03 – 00:40 – 01:20:	In-Lab Task
• Slot – 04 – 01:20 – 02:20:	In-Lab Task
• Slot – 05 – 02:20 – 02:45:	Evaluation of Lab Tasks
• Slot – 06 – 02:45 – 03:00:	Discussion on Post-Lab Task