Data Structures Spring 2023

Lab Task 1: Templates and Introduction to ADT

Introduction to Templates

What is a template?
A template is a very powerful tool in C++. It basically allows you to create a function or class to work with different data types. By using a template, we don't need to write the code for the different data types. We can use the same code by passing data type as a parameter. For example, it can be used for 'int' data type or 'double' data type.

Creating a template:
Templates can be created using keyword 'template' and 'type name'. The pictures below show the syntax to create the templates.The first picture shows how we can create the template T. And the second picture shows how we can use type name T to create a class and/or method.

```
template <typename T>
```

```
T class_name (T x, T y) OR
T method_name (T x, T y)
```

Types of Templates
There are two different ways to use the templates:

1. Function Templates

2. Class Templates

Function Templates

Function template is defined for function. A Function template is basically similar to a normal function but there is one key difference between them. A normal function can work with only

one data type at one time, whereas function template can work with different data types at one time. sort(), min(), max(), printArray() these are some examples of function templates.

Declaring a function template:

```
template <typename T>
T someFunction(T arg)
{
    ..............
}
```

The above picture shows how to declare a function template. It starts with template keyword. <typename T > indicates the template parameter, whereas typename is keyword and T is an argument of template that accepts different data types such as int, float, double, etc. You can also use class keyword instead of typename in template parameter. The function declaration is followed by template parameter. The compiler will generate a new version of someFunction() for given data type as an argument of someFunction().

Example: The following is the example of a function template to find the minimum number

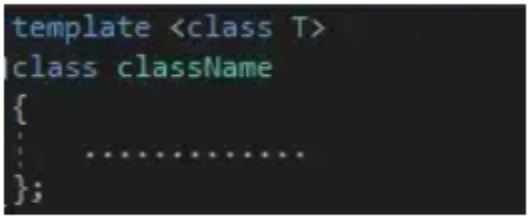
```
#include <iostream>
using namespace std;

//template function
template <typename T>
    T Minimum(T num1, T num2)
{
    return (num1 < num2) ? num1 : num2;
}

int main()
{
    int x, y;
    double a, b;
    char i, j;

    cout << "Enter two integer numbers:\n";
    cin >> x >> y;
    cout << "Minimum integer number is " << Minimum(x, y) <<".\n" << endl; //call Minimum for int

    cout << "Enter two double numbers:\n";
    cin >> a >> b;
    cout << "Minimum double number is " << Minimum(a, b) <<".\n" << endl; //call Minimum for double

    cout << "Enter two characters:\n";
    cin >> i >> j;
    cout << "Minimum ASCII character is " << Minimum(i, j) <<".\n" << endl; //call Minimum for char

    return 0;
}
```

Class Templates

Class template is defined for class. Sometimes, programmer needs to implement the class which is same for all class but the data types that used in class are different. In this case, programmer needs to create the different classes for each data type or create different members and functions within same class. In this situation, it may become hard to maintain the code. Nonetheless, class templates can be used to make it easy to reuse the same code for each data type.

This type of templates are normally used for implementation of containers. It can be instantiated by passing a given type to the argument of template. It can be useful for classes like Stack, Queue, LinkedList, Array, BinaryTree, etc.

Declaring a class template:



Example: The following is the example of class template that calculate the addition, subtraction, multiplication, division of two numbers.

```cpp
#include <iostream>
using namespace std;

template <class T>
class Math
{
private:
    T x, y;

public:
    Math(T num1, T num2)
    {
        x = num1;
        y = num2;
    }

    T add() { return x + y; }

    T sub() { return x - y; }

    T mul() { return x * y; }

    T div() { return x / y; }

    void printResult()
    {
        cout << "Given numbers are: " << x << " and " << y << "." << endl;
        cout << "Sum of two numbers is: " << add() << endl;
        cout << "Subtraction of two numbers is: " << sub() << endl;
        cout << "Multiplication of two numbers is: " << mul() << endl;
        cout << "Division of two numbes is: " << div() << endl;
    }
};
```

```cpp
int main()
{
    Math<int> intMath(8, 5);
    Math<double> doubleMath(6.4, 4.5);

    cout << "Result of Int numbers:" << endl;
    intMath.printResult();

    cout << "\n" << "Result of Double numbers:" << endl;
    doubleMath.printResult();

    return 0;

}
```

LAB-WORK

Task One

Write a C++ template based function that print the largest of the provided data types:

1. Integer
2. Double
3. String
4. Character

Task Two

Write a C++ template based function that will swap the content of two variables provided. The variables will have the data types as follow:

1. Integer
2. Double
3. String
4. Character

Task Three

Write a program that take two arrays as input and find the same elements from the arrays.Write a program that take two arrays as input and concatenate the arrays but the condition is that there are no same elements in the resultant array
Note: input of the array must be of the type int, char, and float.

Task Four

Data types such as int, float, double, long, etc. are considered to be in-built data types and we can perform basic operations with them such as addition, subtraction, division, multiplication, etc. Now there might be a situation when we need operations for our user-defined data type which have to be defined. These operations can be defined only as and when we require them. So, in order to simplify the process of solving problems, we can create data structures along with their operations, and such data structures that are not in-built are known as Abstract Data Type (ADT). Abstract Data type (ADT) is a type (or class) for objects whose behavior is defined by a set of values and a set of operations.

Make an ADT named named Container for storing different type of values e.g. int, double, float etc. This container would keep same type of values/elements at a time. The Container class should consist of the following member variables:

T * values; A pointer to set of values it contains. This member points to the dynamically allocated array (which you will be allocating in constructor).

Capacity; An integer that shows total capacity of the container

Counter; An integer counter variable which increments upon every insertion operation and decrements upon removal of any element; representing total number of elements currently present in the container.

The container should consist of following functions:

constructor with capacity as parameter

isFull which will return true if counter reaches capacity otherwise false

insert which will receive a value of some type (int/double/float) and insert into the container if it's not already full

search which will return true if the container contains value passed as parameter

remove which will remove a value, if exists, from the container and shifts all subsequent values one index back.

print which will print all values contained in the container

*------- END *----------