Data Structures Fall 2023

Lab Task 01: Templates

Introduction to Templates:

Templates are a powerful feature in C++ that allow you to write generic code capable of working with different data types. They enable you to define functions and classes without specifying the exact data type they will operate on. Templates facilitate code reusability and flexibility, as they eliminate the need to duplicate code for different data types.

Advantages of Templates:

1. **Code Reusability:** Templates enable you to write a single piece of code that works with various data types, reducing code duplication.
2. **Flexibility:** You can use templates to create versatile functions and classes that adapt to different data types as needed.
3. **Performance**: Templates allow the compiler to generate specialized code for each data type, resulting in efficient code execution.

Types of Templates:

There are two main types of templates in C++:

1. **Function Templates:** These are used to define generic functions that can operate on different data types.
2. **Class Templates:** These are used to define generic classes that can store and manipulate different data types.

Syntax of Function Templates:

```
template <typename T>
T Add(T a, T b) {
return a + b;
}
```

Syntax of Class Templates:

```
template <typename T>
class MyContainer {
private:    T
elements[100];    int
size; public:
  //Constructor, methods, etc.
};
```

Example 1: Function Template for Addition

```
#include <iostream>

template <typename T>
T Add(T a, T b) {
return a + b;
}

int main() {    int sum_int
= Add(5, 7);
  double sum_double = Add(3.14, 1.618);

  std::cout << "Sum (int): " << sum_int << std::endl;
std::cout << "Sum (double): " << sum_double << std::endl;

  return 0;
}
```

Example 2: Class Template for Container

```
#include <iostream> template
<typename T, int Size> class
MyContainer { private:    T
elements[Size];
  int count;

public:
  MyContainer() : count(0) {}
```

```
    void AddElement(T  element) {
if (count < Size) {
elements[count] = element;
count++;
        }
    }

    void PrintElements()  {           for
(int i = 0; i < count; i++)  {
            std::cout <<  elements[i]  << " ";
        }
        std::cout <<  std::endl;
    }
};

int main()  {
    MyContainer<int, 5> intContainer;
intContainer.AddElement(42);
intContainer.AddElement(73);         std::cout
<< "Elements  (int): ";
    intContainer.PrintElements();

    MyContainer<std::string, 3> stringContainer;
stringContainer.AddElement("Hello");
stringContainer.AddElement("World");          std::cout
<< "Elements  (string): ";
stringContainer.PrintElements();

    return  0;
}```
```

Lab Work

Task-1:

You are provided with the dataset of university students. This datasets contains names, ages, course major and pending fee. Your task is to write a function that removes duplicates of any type of dataset provided. Write a template function **RemoveDups()** that removes duplicate vales form the provided dataset.

Example :

Age_data = {21, 32, 22 , 33, 22}

Name data = {sara, mary, larry, sara}

The function should return {21,32,22,33} and {sara, mary,larry} when provided with the respective dataset.

Task-2:

You are hired by the Examination Control Center to develop an algorithm that rotates the students seating arrangement every exam term. The problem is that the student's dataset can be their names or their roll numbers.

You have to write a template function **Rotate (Students)** that accepts a template array. The function will perform a k left rotations on the provided array. Where k in an integer $0 < k <$ size of array

Example

Input: nums = [1,2,3,4,5,6,7], k = 3

Output: [4,5,6,7,1,2,3]

Input: nums = [-1,-100,3,99], k = 2

Output: [3,99,-1,-100

Task-3:

Define a template class **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 ful**l**
- **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

Task-4

You are provided with a set of scores of students. Your task is to return the set of two scores that add up to a given target. Write a template function for an array of Numbers "scores" and a number score target, that return indices of the two numbers such that they add up to target. The array should be a template array and the score variable should also be template variable as well.

Example 1:

Input: student_ scores = [10,7,0,7], target_score = 14
Output: [1,3]
Explanation: Because nums[1] + nums[3] == 14, we return [1, 3].

Example 2:

Input: student scores = [3.0,2.0,4], target_score = 6.0 **Output:**
[1,2]

Example 3:

Input: student_scores = [3.0 , 3], target_score = 6
Output: [0,1]