Lab Assignment 3

**Templates**

1. Convert the following function to templates

#include <iostream>

using namespace std;

void printIt(int a, int b)

{

int c = a + b;

cout << "You gave me " << a << " and " << b << ".\n";

cout << "Together they make " << c << "." << endl;

}

int main()

{

string sA = "Oh ";

string sB = "noes!";

printIt(1,2);

printIt(2.6, 3.7);

printIt('A','1');

//printIt(sA, sB);

}

1. Make a class template out of the swap function and test it on different data types.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\*

\* Purpose: Demonstrate the use of function templates

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

using namespace std;

//Make a template out of the prototype

int swap(int &one, int &two);

int main()

{

int i\_one = 3, i\_two = 5;

cout << "i" << i\_one << " and " << i\_two << " is "

<< swap(&i\_one, &i\_two) << endl;

//Test your template on float and string types

return 0;

}

Test your swap template on int, double, and string types.

Test output

The swap of 3 and 5 is 5 and 3

The swap of 5.6 and 7.3 is 7.3 and 5.6

The swap of donkey and apple is apple and donkey

1. <https://www.hackerrank.com/challenges/c-class-templates/problem>

A class template provides a specification for generating classes based on parameters. A class template is instantiated by passing a given set of types to it as template arguments. Here is an example of a class, MyTemplate, that can store one element of any type and that has just one member function *divideBy2*, which divides its value by 2.

template <class T>

class MyTemplate {T element;

public:MyTemplate (T arg) {

element=arg;}

T divideBy2 () {

return element/2;}

};

It is also possible to define a different implementation of a template for a specific type. This is called *Template Specialization*. For the template given above, we find that a different implementation for type *char* will be more useful, so we write a function *printElement* to print the *char* element:

// class template specialization:

template <>

class MyTemplate <char> {

char element;

public:MyTemplate (char arg) {element=arg;}

char printElement (){return element;}};

You are given a main() function which takes a set of inputs. The type of input governs the kind of operation to be performed, i.e. concatenation for *strings* and addition for *int* or *float*. You need to write the class template *AddElements* which has a function *add()* for giving the sum of *int* or *float* elements. You also need to write a template specialization for the type *string* with a function *concatenate()* to concatenate the second string to the first string.

**Input Format**

The first line contains an integer

. Input will consist of lines where is the number given in the first line of the input followed by

lines.

Each of the next

line contains the type of the elements provided and depending on the type, either two strings or two integers or two floating point numbers will be given. The type will be one of *int*, *float* or *string* types only. Out of the following two elements, you have to concatenate or add the second element to the first element.

**Constraints**

, where is any float value

, where valueint is any int value

, where

is the length of any string

**Array**

1. Perform the following operations on array
2. Initialize two arrays of size 5 with the following values

{3, 4, 5, 1, 2}, {1, 2, 3, 5}

1. Print the size and max\_size of both the arrays
2. Print the contents of both the arrays.
3. Print the first element and last element of both the arrays
4. Insert 4 in 2nd array {1,2,3,4,5}
5. Sort 1st array and print
6. Swap the contents of 2nd array with 1st array.
7. Replace the whole 2nd array with element 5 and print
8. Replace the whole array with {1,2,3,4,5} and print
9. Use class template Array and search for a number from an Array.

Test output

Simple Class Template Array Program Example : Search Number

Enter 5 Elements for Searching Int :

56

78

10

34

67

Enter Element to Search : 67

Your Data : 56 78 10 34 67

Class Template Search : Element : 67 : Found : Position : 5.

Enter 5 Elements for Searching float :

90.89

89.008

67.45

100.1

90.41

Enter Element to Search : 90.89

Your Data : 90.89 89.008 67.45 100.1 90.41

Class Template Search : Element : 90.89 : Found : Position : 1.

1. Given an array *nums* and a value *val*, remove all instances of that value [**in- 3 0place**](https://en.wikipedia.org/wiki/In-place_algorithm) and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array** [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

**Example 1:**

Given *nums* = **[3,2,2,3]**, *val* = **3**,

Your function should return length = **2**, with the first two elements of *nums* being **2**.

It doesn't matter what you leave beyond the returned length.

**Example 2:**

Given *nums* = **[0,1,2,2,3,0,4,2]**, *val* = **2**,

Your function should return length = **5**, with the first five elements of *nums* containing **0**, **1**, **3**, **0**, and **4**.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.

**Clarification:**

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

// **nums** is passed in by reference. (i.e., without making a copy)

int len = removeElement(nums, val);

// any modification to **nums** in your function would be known by the caller.

// using the length returned by your function, it prints the first **len** elements.

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

    print(nums[i]);

}

1. <https://leetcode.com/problems/remove-duplicates-from-sorted-array/solution/>

Given a sorted array *nums*, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each element appear only *once* and return the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array** [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Example 1:**

Given *nums* = **[1,1,2]**,

Your function should return length = **2**, with the first two elements of *nums* being **1** and **2** respectively.

It doesn't matter what you leave beyond the returned length.

**Example 2:**

Given *nums* = **[0,0,1,1,1,2,2,3,3,4]**,

Your function should return length = **5**, with the first five elements of *nums* being modified to **0**, **1**, **2**, **3**, and **4** respectively.

It doesn't matter what values are set beyond the returned length.

**Clarification:**

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by **reference**, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

// **nums** is passed in by reference. (i.e., without making a copy)

int len = removeDuplicates(nums);

// any modification to **nums** in your function would be known by the caller.

// using the length returned by your function, it prints the first **len** elements.

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

    print(nums[i]);

}