**CS-200 Fall 2014**

**LAB 12**

**Time Allowed: 2 Hours and 30 Minutes**

**Warning:** This lab may be lengthy compared to labs in last few weeks so plan accordingly

**Instructions:** After lab you will have 72 hours to submit lab tasks on LMS with 40% deductions.

You can consult the lecture slides in case of any confusion.

**Templates:**

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type. A template is a blueprint or formula for creating a generic class or a function. An example is that of the vector library. There is a single definition of the “vector”, but we can define many different kinds of vectors for example, vector <int> or vector <string>. You can use templates to define functions as well as classes.

The basic syntax is:

template<class T>  
void func\_name (T arg1, T arg2) {

//lines of code…

}

OR

template<typename T>  
void func\_name (T arg1, T arg2) {

//lines of code…

}

Both are identical in this case. Here, “**T**” is a placeholder name for a data type used by the function. The data type T can be used within the function.

The function can be called like this func\_name<int>();

The “typename T” or “class T” sort of an argument that is passed to the template. That means that you can pass arguments to the template other than typename too, e.g. int, char etc. Just like you pass arguments to normal functions, but inside the < > instead of ( ).

**Template specialization:**

It is possible to define a different implementation for a template when a specific type is passed as template argument. This is called a template specialization.

An example is given below:

template<typename T>  
void func\_name (T arg1, T arg2) {

//lines of code…

}

template<> // Note that there is nothing in the **< >**  
void func\_name<string> (string arg1, string arg2) {

//do something specific to string data type …

}

Task 1: 15 Marks

This task is relatively simple. Write a function template named “**print()**”.

The function **does not** have any input arguments.

The **ONLY** template parameter is:

* An integer **n.**

The return argument is:

* Void. As it does not need to return anything.

When called this function prints numbers from **n** to 0 in descending order on the console.

Simple? Maybe! Now here is the twist!

**You can NOT use any if or while or for statements in the function.**

e.g if n = 50 the output is:

50

49

.

.

.

1

0

Task 2: 30 Marks

Write a **RECURSIVE** function template named “**binary\_search**”.

The **ONLY** function arguments are:

* An array of type T. Make sure that you do not modify the initial array. i.e. take a **const** array

The template parameters are (in order):

* Type of the array
* Size of the array
* Item that you need to search

The return argument is:

* An int telling the position of the object or -1 if object is not found.

Give both the function declaration and the definition using templates. The function **must be RECURSIVE.** Alsodo not make any other function i.e no function other than binary\_search would be used**.**

Here is a description of binary search taken from the internet to help in your implementation:

“Binary search relies on a divide and conquer strategy to find a value within an already-sorted collection. The algorithm is deceptively simple. Pretend I was thinking of a number between 1 and 100. Every guess you take, I'll say higher or lower. The most efficient way to discover my number is to first guess 50. Higher. 75. Lower. 62. Higher 68. Yes!”

Test your code by making the following array: {1,2,3,4,5,6,9,24,100,1000}

Also test for array of some other type.

And searching for: 1,2,3,4,5,6,9,24,100,1000,-5,11,20

The output should be: 0,1,2,3,4,5,6,7,8,9,-1,-1,-1

Task 3 (a): 15 Marks

You can use the code from your linked list assignment for this part.

Your task is to modify it so that it is fully compatible with templates.

Task 3 (b): 40 Marks

Your task now is to implement a “**Set**” data structure. You have to do this by making a new class named “Set” that inherits from the linked list class.

The most important property of a set is that it can only contain one instance of a specific item. As in every item in a set is unique.

Your Set class must support the following:

* A void function “insert”

e.g

some\_set.insert(some\_item);

* An overloaded << operator to display the elements in the set in a comma separated way.

e.g

cout << some\_set;

* An overloaded + operator that does set union. Make sure that setA and setB are not modified.

e.g.

setUnion\_AB = setA + setB;

* And another + operator that inserts an item to the set.

e.g

setA = setA + some\_item // you can simply call the insert function in this.

* A function “contains(item)” that returns a Boolean telling whether the set contains the item or not.

Do not declare any operator as a friend, except << operator.

All operators must be declared inside the class.