# Homework 8

For this homework, you will learn how to use templates by implementing a TreasureMap class. This class accepts a series of steps that will lead to the treasure. Since each step can be of any type such as a string, a number, or another class/struct, the usage of a template is needed. In addition, you will need to incorporate error handling methods such as throwing and catching an exception when reading treasure maps in the form of input text files.

# TreasureMap

TreasureMap class is a template class. It represents a treasure map that holds a series of steps. Each step is of a template class type. The user can traverse backwards and forwards, combine, and compare each maps.

Define TreasureMap class with the following members:

* private:
  + steps is a dynamic array of type T
  + current\_step is an int, starts at 0, indicate the index of the current step in the treasure map
  + total\_steps is an int that indicates how many elements are currently in steps
  + max\_steps is an int that indicates the maximum number of elements steps can hold
* public:
  + default constructor: set total\_steps and current\_step to 0, max\_steps to 10, and initialize steps to an array of 10 elements
  + TreasureMap(int max\_steps) this constructor sets the max\_steps, and initialize steps array to max\_steps
  + getSteps() This function returns the dynamic array T
  + getCurrentStep() This function returns the current\_step
  + getTotalSteps() This function returns the total\_steps
  + (Other getter and setter functions are not required)
  + addStep() a void function that takes in a T element, add it to steps and increment total\_steps. If total\_steps is equal to max\_steps, show this message and don't add.

Cannot add more steps, TreasureMap is full

* + removeStep() a void function that takes in an int index and remove the element in steps at that index (make sure to decrement total\_steps).
  + nextStep() a void function that prints this message and then increment the current\_step

Going from <current element> to <next element>

* Where current element is the T element at current\_step index of steps and next element is current\_step + 1. If current\_step is already the last step, print this message instead:

Cannot goto next step, treasure has already been found!

* If after incrementing current\_step and reaching the last step, print:

Congratulations, you have found the treasure!

* + backtrack() a void function that prints this message and then decrement the current\_step

Backtracking from <current element> to <previous element>

* Where current element is the T element at current\_step index of steps and previous element is current\_step - 1 If current\_step is already the last step, print this message instead:

Cannot backtrack, you're at the very first step!

* + Overload the + operator, if the total\_steps don't match then return a blank map (made using the default constructor). Else make a new TreasureMap and each step in the new map is the addition of map 1's step with map 2's step. For example, given map1 with steps 1, 2, 3 and map2 with steps 1,3,4, map3 = map1 + map2. map3's steps will be 2,5,7.
  + Overload the == operator, returns true if the total\_steps match, and each element in steps is equal to each other. Return false otherwise

NOTE: Please implement all of the above methods of the TreasureMap class in "TreasureMap.h" for this assignment.

# Location class

Since it's quite difficult to follow a treasure map whose step is only either words or numbers, it's better to combine both the name of the location and it's coordinates into the same object. Hence we will create the Location class.

The Location class has been provided to you with no changes needed with the following members:

* name is a private string
* x is a private float
* y is a private float
* A constructor that accepts the name, x, and y
* Functions getName(), getX(), getY() returns name, x, and y respectively
* Functions setName(), setX(), setY() sets name, x, and y respectively
* overload the addition operator (+): create a new Location, it's name, x, and y will be the addition of the original two locations. For example, location A is ("Houston", 1, 2) and location B is ("Austin", 2, 3), A + B = ("HoustonAustin", 3, 5)
* overload the compare operator (==) to return true if the name, x, and y are the same and return false otherwise.

# FileReadException

This class will be the exception thrown and catch during the file reading process

Define FileReadException class with the following members:

* err\_message a string that holds the error message
* public:
* a constructor that takes in a string and assign it to err\_message
* getMessage() returns err\_message

# main.cpp

In the main.cpp, you need to complete the following two functions.

### readMap()

This is a template function that takes in two arguments, a string file\_name and an int for max\_steps. It returns a TreasureMap of type T.

This function first opens the file with the file\_name and if it doesn't exists, throws a FileReadException exception with this message:

Error reading map: <file\_name> does not exists!

There are three types of input files, including string, int, and char. Each line in an input file is an item. For example, an input file with all strings would look like:

Bob

Amy

Houston

(None of the strings will have spaces)

And an input file with all integers looks like:

1

23

14

15

The function will create a new TreasureMap of type T, then for each item in the input file, use the extraction operator (>>) to read in a new element of type T and add it into the map with addStep() member function of the TreasureMap class.

However, if there are more items in the input file than the provided max\_steps (2nd argument of readMap), then throw a FileReadException with this message:

Error reading map: <file\_name> contains more than <max\_steps> steps!

After reading (without issues), the function returns that TreasureMap object.

### About reading the location input file

You don't have to worry about the location input file being different format, an overload of the >> operator has already been provided in the main() function for you. The readMap() function can be applied to the location text file without issues.

### main() function

The main function has been partially completed, firstly it takes in three variables from user input in this order:

* a string (without space) for the file name to be read with readMap()
* a string for the type of input file, can either be "string", "int", "char", or "location" this determines the type of input file so we can call readMap() with the right variable type.
* an int for the max\_steps (second argument in readMap)

Based on the input file type, your task is to create a new TreasureMap and use readMap() to read the new map. Then set a for loop that runs for the number of total\_steps in the new TreasureMap, for each loop, call the map's nextStep() function.

The readMap() functions called inside main() has been enclosed with a try catch block, your task is to define a FileReadException in the catch block and print out it's message with cout and getMessage().

Example main input:

sample\_input.txt string 10

Example main output:

Going from A to B

Going from B to C

Going from C to D

Going from D to E

Congratulations, you have found the treasure!

sample\_input.txt:

A

B

C

D

E

IMPORTANT

* classes and methods names must match exactly for unit testing to succeed.
* Submissions with hard coded answers will receive a grade of 0.

Late submission penalty:

* Code submitted within 24 hours after the deadline will receive a 20% penalty.
* Code submitted within 48 hours after the deadline will receive a 40% penalty.

BONUS

* Code submitted two days before the deadline and pass ALL test cases will receive 10 bonus points.
* Code submitted one day before the deadline and pass ALL test cases will receive 5 bonus points.

Submission Instructions

Downloadable files

main.cpp

,

TreasureMap.h

,

FileReadException.h

,

FileReadException.cpp

,

Location.h

, and

Location.cpp

**Download**

Compile command

g++ main.cpp FileReadException.cpp Location.cpp -Wall -o a.out

*We will use this command to compile your code*