UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program design: Introduction

Practical 10 Specifications: Recursion and STL Stacks

Release date: 07-11-2022 at 06:00
Due date: 11-11-2022 at 23:59
Total Marks: 144

Contents

1 General instructions:

- This assignment should be completed individually, no group effort is allowed.
- Be ready to upload your assignment well before the deadline as no extension will be granted.
- You will need to use the stl::stack library for this practical.
- If your code does not compile you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experience a runtime error you will be awarded a mark of zero. Runtime errors are considered as unsafe programming.
- All submissions will be checked for plagiarism.
- Read the entire specification before you start coding.
- Ensure your code compiles with C++98

2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The aim of this practical is to gain experience with recursion and the stl::stack Data Structure.

4 Background

4.1 Stacks

From theory, you should know that a stack is a specialized type of link list. In a stack data structure, element elements are added and removed in a Last In First Out (LIFO) manner meaning elements are added to the top of the stack and removed from the top of the stack.

4.2 stl::stack

The std::stack is a data structure that can be found in the stl library of c++. Note the std::stack's pop() function differs from the standard in the sense that it does not return anything. Use a top pop combination to get and remove the top element of the stack.

4.3 Towers of Hanoi

Towers of Hanoi is a popular puzzel game that consists out of two main components: three towers and n number of disks. The goal of this puzzel game is to move the tower of disks from one tower to another tower with only moving a single disk at a time. Also a constraint that is placed at all times on the game is that no disk may be placed on another disk if the bottom disk is smaller then the top disk. More information can be found at some of the following links:

- `https://en.wikipedia.org/wiki/Tower_of_Hanoi`
- `https://www.mathsisfun.com/games/towerofhanoi.htm`
- `https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi/`
- `https://www.tutorialspoint.com/data_structures_algorithms/tower_of_hanoi.htm`

5 Introduction

Implement the UML diagram and functions as described on the following pages.

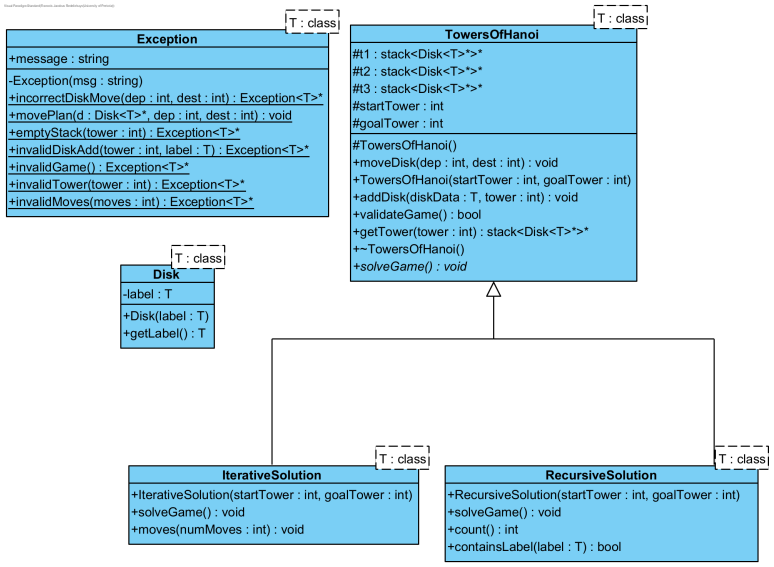

Figure 1: Class diagrams

You have been provided with skeleton header files for all the above classes.

6 Classes:

6.1 Exception

This is a provided class. Do not alter this class as it will be overwritten

6.2 Disk

- Members:
  - label: T
    - This is the label for the disk
- Functions:
  - Disk(label: T)
    - This is the constructor for the Disk class.
    - This function should set the appropriate member with the passed in parameter
  - getLabel() const: T
    - This function is a const function.
    - This function should return the label.

6.3 TowersOfHanoi

- Members:
  - t1: stl::stack<Disk<T>*>*
    - This is the first "tower" in the game.

  - t2: stl::stack<Disk<T>*>*
    - This is the second "tower" in the game.
  - t3: stl::stack<Disk<T>*>*
    - This is the third "tower" in the game.
  - startTower: int
    - This is the starting tower's number for the game.
  - goalTower: int
    - This is the goal tower's number for the game.

- Functions:
  - TowersOfHanoi()
    - This is a constructor for the class.
    - Create the function skeleton in the appropriate cpp file but leave the body empty
  - moveDisk(dep: int, dest: int)
    - This function should move a disk between the two passed-in parameters.
    - The move can be from one parameter to another or visa versa based on the rules.
    - If either of the passed in parameters is not a valid tower (i.e cannot be mapped to the first, second or thrid tower) then the function should throw an Exception<T>::incorrectDiskMove exception. Valid towers are: {1, 2, 3}
    - If a disk is being moved from the same tower to the same tower the same exception should be thrown.
    - You will need to do research about the rules for moving a disk between towers in Towers of Hanoi.
    - When moving a disk between towers the top disk of the selected tower should be poped and pushed onto the newly selected tower.
      Hint: remember about the top function if pop does not return anything
    - If the top disk of the selected tower is null the function should throw Exception<T>::emptyStack exceptions.
    - After the disk has been moved your function should call the Exception<T>::movePlan function such that the move can be printed out.
  - TowersOfHanoi(startTower: int, int goalTower)
    - This is a constructor for the class.
    - Using the passed-in parameters the appropriate member variables should be initialized. Also, initialise all three of the towers with new stacks.
  - addDisk(diskData: T, tower: int): void
    - This function should add a disk to the tower that corresponds to the passed in parameter.
    - Use the diskData as the disk's data.
    - If the passed in tower parameter does not correspond to any tower your code should throw an Exception<T>::invalidDiskAdd exception
    - If the passed in label is bigger then the top disk's label of the tower that will be added to your code should throw Exception<T>::invalidTower exception.
  - validateGame(): bool

    - This function should check if the game is valid.
    - For a game to be valid the following properties need to be satisfied:
        - Only 1 tower contains disks
        - The start and goal tower member variables correspond to towers and these towers are not the same.
    - If the game is valid return true else return false.
- ~TowersOfHanoi()
    - This is the destructor for the class.
    - Each tower should be destroyed. Each disk in each tower should be poped and deallocated.
    - The stacks should also then be deallocated.
- getTower(tower: int): std::stack<Disk<T>*>*
    - This function should return the tower corresponding to the passed in parameter.
    - It can be assumed that the passed in parameter will be valid.
- solveGame(): void
    - This is a pure virtual function that will be implemented in the derived classes.
    - This function will solve the Towers of Hanoi game.

6.4 IterativeSolution

This class has a public inheritance relationship with TowersOfHanoi

- Functions:
    - IterativeSolution(startTower: int, goalTower: int)
        - This is a constructor for the class.
        - Using the passed-in parameters the appropriate member variables should be initialized. Also, initialise all three of the towers with new stacks.
    - solveGame(): void
        - This function should solve the towers of Hanoi game using an iterative algorithm.
        - This function should throw the Extention<T>::invalidGame() if the game is invalid.
        - You will need to do research on such an algorithm.
        - Use the moveDisk function to move your disks between towers.
        - Your algorithm should use the minimum amount of moves to complete the game while maintaining all the rules of towers of Hanoi.
    - moves(numMoves: int): void
        - This function should perform the first numMoves steps in the algorithm used in solveGame.
        - If numMoves is greater then the minimum amount of moves required the output should match that of the above solveGame function.
        - If the numMoves is negative throw Exceptions<T>::InvalidMoves()

6.5 RecursiveSolution

This class has a public inheritance relationship with TowersOfHanoi.
Note: You may not use any other loop apart from recursion for any code in this class. Failure to do so may result in zero. Do not include the following keywords in your files (even as comments):

- for
- while
- do
- goto/jump

You may also not use the stack's size function in this class.You may add any recursive helper functions that you want.

- Functions:
  - RecursiveSolution(startTower: int, goalTower: int)
    - This is a constructor for the class.
    - Using the passed-in parameters the appropriate member variables should be initialized. Also, initialise all three of the towers with new stacks.
  - solveGame(): void
    - This function should solve the towers of Hanoi game using an recursive algorithm.
    - You will need to do research on such an algorithm.
    - Remember to use Exception<T>::movePlan to print out which disk is being moved during each move step.
    - Your algorithm should use the minimum amount of moves to complete the game while maintaining all the rules of towers of Hanoi.
  - count(): int
    - This function should recursively count the amount of disks currently in the game. The disks can be spread over the three towers.
    - Remember that you may not use the size function in this class.
  - containsLabel(label: T): bool
    - This function should recursively check if the passed in label belongs to a disk that is currently in the game. The disks can be spread over the three towers.
    - If the label is contained the function should return true else false.

7 Possible problems

If you are experiencing errors where inherited member variables are not defined please use this-> to access the member variables.

8 Source files

- TowersOfHanoi.h,cpp
- RecursiveSolution.h, cpp
- IterativeSolution.h, cpp
- Exception.h

9 Allowed libraries

- iostream
- stack
- cmath
- string
- sstream

10 Graphical Representation

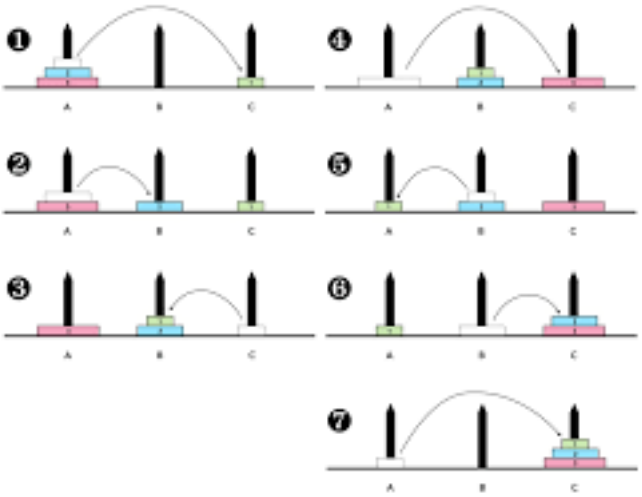

Figure 2: Graphical display of a Tower of Hanoi game with three disks

11 Submission

You need to submit your source files, only the cpp files, on the Fitch Fork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or h files in your submission. Your code should be able to be compiled with the C++98 standard
For this practical you will have 10 upload opportunities. Upload your archive to the Practical 9 slot on the Fitch Fork website.