**NATIONAL UNIVERSITY OF SINGAPORE**

*SCHOOL OF COMPUTING*

Practical Examination (PE)

**TIC2001 Data Structure and Algorithm**

28 Oct 2021 Time Allowed: 2 hours

---

## VERY IMPORTANT:

- Please make sure your code can be compiled and runnable after you submitted to Couresmology. Any crashing code will be zero mark.
- You only have 5 chances of submission to each part. So please test out your code in MSVS or XCode first before running them on coursemology. Do not use coursemology as a debugger to test your code.
- You can only use MSVS or XCode
- You should stop modifying your code and start submitting 10 min before the end of the PE in order to avoid the "jam" on Coursemology. Actually, you should submit a version after every time you finished a task. It's your own responsibility to submit the code to Coursemology on time.
- You cannot include other extra libraries such as STL.
- It's your own responsibility to make sure that you have submitted the right code on time.
- The PE consists of three parts. And in each part, you will be provided with a MSVS or XCode project. In each project file, there will be one file called **`partXSubmission.{cpp,hpp}`**. You should modify and submit that file only by "select all" (ctrl-A) in that file and copy and paste it to the corresponding part in Coursemology. Your submission should be able to be complied and run under the assumption that the rest of the files in your project is the same as the original project you downloaded except the `partXSubmission.xpp`.
- **We will close the PE submission sharply when time runs out.** Make sure you save and submit your work a few minutes before it ends. The network will be jammed at the last few min of the assessment because everyone is submitting at the same time.
- *You should let the coursemology **webpage opened** all the time.*
- Any variables used must be declared within some functions. You are not allowed to use global variables (variables that are declared outside all the functions). Heavy penalty will be given (see below) if you use any global variable.

## Advice

- Manage your time well! Do not spend excessive time on any task.
- Read all the questions and plan before you start coding.
- Please save and backup your code regularly during the PE.
- It is a bad idea to do major changes to your code at the last 10 minutes of your PE.

## Brief Summary of the Whole PE

- Part 1: Linked List with Indexing (30 marks)
- Part 2: d-ary Heap (30 marks)
- Part 3: Hash Table with quadratic probing, and count the sum of 3 numbers in an array. (40 marks)

## Part 1 Linked List Access (10 + 20 = 30 marks)

You are provided with the trimmed skeleton code of Assignment 2. Your job is to implement the following three features

- Access the elements in the list like an array using the operator `[]`.
- Insert an element at a certain position/index
- Remove an element at a certain position/index

### Task 1 Operator `[]` (10 marks)

Implement the operator `[]` for a List `L` such that `L[i]` will return the element of the list `L` with index `i` starting with 0 like the array. Here is an example output:

```
Operator [] Test
The linked list so far: 6 5 4 3 2 1 f e d c b a
The char with index 0 is 6
The char with index 2 is 4
The char with index 4 is 2
The char with index 6 is f
The char with index 8 is d
```

Your function should print "`Index out of bound error (operator[])`" if the index is out of bound.

### Task 2 Insert and Remove an item at a Certain Position (20 marks)

Implement the two functions `insertAtPos(int idx, T item)` and `removeAtPos(int idx)` that will insert or remove an item at the positive with index `idx`. Here is an example for insertion. After an insertion, the length of the list will be increased by 1.

```
Original list:f e d c b a
After
   • insertAtPos(4, '#');
   • insertAtPos(6, '#');
   • insertAtPos(8, '+');
New list: f e d c # b # a +
```

And an example for removal. Note that the input of the function is the index, not the item. The items going to be removed are in red.

```
Original list: 6 5 4 3 2 1 f e d c b a
After
   • removeAtPos(7);
   • removeAtPos(3);
   • removeAtPos(0);
New list: 5 4 2 1 f d c b
```

Your function should print "`Index out of bound error (insertAtPos)`" or "`Index out of bound error (removeAtPos)`" if the index is out of bound. Here is the expected input. However, if your list has n elements already, you can add one more item at the position with index n, aka, appending to the list.

## Part 2 d-ary Heap (15 + 15 = 30 marks)

In lecture we learned binary heaps such that every node has at most 2 children. In this part, we will implement the d-ary heap in which every node has at most d children with d > 1. And you are given the trimmed skeleton code of your heap assignment. You should implement the following features for any d on top of d=3 in the example given.

- Inserting an item into the heap
- Extract Max from the heap
- Normally, you should implement the bubble up/down functions.

Notice that you do not have to implement other functions such as delete an item or increase/decrease keys.

A d-ary heap follows all of the binary heap definitions. And here are some extra features/information:

- Each node can have at most d children
- With array implementation, the parent of the node with index `i` is $\lfloor (i - 1)/d \rfloor$
- The d children of the node (with index `i`) is from index `d*i + 1` to index `d*i + d`

### Task 1 Implement the Insert Function for a d-ary heap (15 marks)

The insert function will insert an item into the heap (array) according to the max heap rules as in Assignment 4. From an empty heap, here are the heap arrays with `d = 3` after insertions.

```
Heap Test 2
Insert 1 into the heap.
Heap Array:1

Insert 2 into the heap.
Heap Array:2 1

Insert 3 into the heap.
Heap Array:3 1 2

Insert 4 into the heap.
Heap Array:4 1 2 3

Insert 5 into the heap.
Heap Array:5 4 2 3 1

Insert 6 into the heap.
Heap Array:6 5 2 3 1 4

Insert 7 into the heap.
Heap Array:7 6 2 3 1 4 5
```

```
Insert 8 into the heap.
Heap Array:8 6 7 3 1 4 5 2

Insert 9 into the heap.
Heap Array:9 6 8 3 1 4 5 2 7

Insert 10 into the heap.
Heap Array:10 6 9 3 1 4 5 2 7 8

Insert 11 into the heap.
Heap Array:11 6 9 10 1 4 5 2 7 8 3

Insert 12 into the heap.
Heap Array:12 6 9 11 1 4 5 2 7 8 3 10

Insert 13 into the heap.
Heap Array:13 6 9 12 1 4 5 2 7 8 3 10 11

Insert 14 into the heap.
Heap Array:14 13 9 12 6 4 5 2 7 8 3 10 11 1

Insert 15 into the heap.
Heap Array:15 14 9 12 13 4 5 2 7 8 3 10 11 1 6
```

### Task 2 Implement the ExtractMax Function (15 marks)

Exactly like the one in the assignment, implement the `extractMax()` function with `d=3`.

```
Heap Test 3
Initial heap:
15 14 9 12 13 4 5 2 7 8 3 10 11 1 6
Extract Max: 15
14 13 9 12 6 4 5 2 7 8 3 10 11 1
Extract Max: 14
13 6 9 12 1 4 5 2 7 8 3 10 11
Extract Max: 13
12 6 9 11 1 4 5 2 7 8 3 10
```

## Part 3: Hash Table Quadratic Probing and Sum of 3 Integers (15+5+20 marks)

You are provided with a class Hash Table with a hash function (The hash function is sum of digits of the numeric value but it doesn't really matters for you). In the beginning, your hash table will be initialized by a given table size and allocated some memory dynamically. You can assume that

- We will never insert more items than the table size.
- We will only insert integers > 0
- An entry of the table is 0 if it's empty
- An entry of the table is -1 if it's deleted.
- All access is with the open addressing with **quadratic** probing scheme.

### Task 1 Quadratic Probing Hashing Operations (15 marks)

Your job is to implement the quadratic probing for the functions insert, remove and exist.

- Insert an integer into the table. If the integer is already in the table, it will not modify the table and print that number is `" already exists in the hash table."`.
- Remove an integer from the table. Print `"Fail to remove "` if the integer is not in the table before trying to remove it.
- Check if a certain integer is still in the table. Return true or false accordingly.

### Task 2 Resizing (5 marks)

Implement the resize function such that it will resize the table. For example

```
Current hash table:
0 -1 2 101 4 0 6 7 8 9 0 11 71 -1 0 555 96
After resizing to a table size of 13:
0 96 2 101 4 555 6 7 8 9 0 11 71
```

Note that:

- You will rehash the values in the original hash table into the new table according to their orders in the original hash table, not the order of insertion in their original table.
- We will never resize the table to be smaller than the number of items already stored in the table.

## Task 3 Sum of Three Numbers (20 marks)

Given an array of positive integers with no duplicates, count the number of set of 3 numbers that will sum to a given number `t`. E.g. if the array is

```
{14,52,23,11,12,72,21,22,13,53,54}
```

There are **FIVE** set of numbers that will sum up to `t=88`, namely,

```
(14, 52, 22 ), (14, 21, 53 ), (52, 23, 13 ), (52, 22, 14 ), (12, 53, 23 )
```

Note that the different permutations of a set of 3 numbers will be counted as 1 only. E.g. (14, 52, 22 ), (52, 22, 14 ), (14, 22, 52 ), etc. will be counted as 1 set only.

Write a function int `n3Sum(int* arr, int s, int t)` that will take in an integer array `arr` with `s` elements and return the number of set of 3 numbers with sum `t`.

The true testing objective for this task is for efficiency. You will only get full marks if you can handle an array that is more than 3000 integers within 1 second.

Some more sample output for the above array:

```
n3Sum(test1,10,90) will return 1

n3Sum(test1,10,90) will return 3

n3Sum(test1,10,90) will return 5
```