# CSE225 mid.pdf-converted

cse course (North East University)

# NORTH SOUTH UNIVERSITY
## Department of Electrical and Computer Engineering
## B.Sc. in Computer Science and Engineering Program
## Theoretical Assessment (Take-home) – 1
## Spring 2021 Semester

| | |
|---|---|
| **Course:** | **CSE 225 Data Structure and Algorithms, Section-14 and 15** |
| **Instructor:** | **Mohammad Rezwanul Huq (MRH1), PhD, Associate Professor (Part-time)** |
| **Full Marks:** | **100** |
| **Submission Deadline:** | **11:59 PM, 12 April 2021** |

**Note:** There are **10** (**TEN**) questions, answer ALL of them. Mark of each question are mentioned at the right margin.

---

1. **Consider** following program. [10]

```
#include <iostream>
using namespace std;
 int f(int x, int *py, int **ppz)
 {
   int y, z;
   **ppz += 1;
    z  = **ppz;
   *py += 2;
    y = *py;
    x += 3;
    return x + y + z;
 }
 int main()
 {
    int c, *b, **a;
    c = 5;
    b = &c;
    a = &b;
    cout << f(c, b, a) << endl;
    return 0;
 }
```

**Show** all these variables and their allocation of values/addresses appropriately using a suitable figure.

2. Describe the worst-case running time of the following code in “big-Oh” notation in terms of the variable n. You should give the tightest bound possible. [10]

**(a)**

```
void f1(int n) {
    for(int i=0; i < n; i++)
        { for(int j=0; j < n; j++)
        {
            for(int k=0; k < n; k++) {
                    printf("!");
} } }
}
```

**(b)**

```
int f3(int n)
    { int sum =
    73;
    for(int i=0; i < n; i++)
        { for(int j=i; j >= 5; j--)
        {
            sum--;
        }
    }
    for (int k=0; k < n; k++) {
        printf("!");
    }
    return sum;
}
```

**3.** Fill in the blanks in the following table by putting appropriate values of runtime complexity (in Big-O notation) for different operations. [10]

| operation type | using Array | using Singly Linked List (without tail pointer) | using Doubly Linked List (with tail pointer) |
|---|---|---|---|
| accessing any elements arbitrarily | | | |
| inserting an element in the front of a list | | | |
| deleting an element from the end of a list | | | |

**4.** Given two singly linked list, write a function `intersect()` that finds the common elements of both list. Also determine the complexity of your function in terms of Big-Oh notation. [10]

For example: head1 and head2 are the heads of two lists. head1 ⟶ 1 ⟶ 2 ⟶ 3 ⟶ 4 ⟶ 5 ⟶ NULL

head2 ⟶ 1 ⟶ 7 ⟶ 2 ⟶ NULL

The output should be: head ⟶ 1 ⟶ 2 ⟶ NULL

While writing the function, consider the following Node class.

```
class Node{
   public:
      int key;
      Node* next;
}
```

**5.** Given two strings, represented as Singly Linked Lists (without tail pointer) in which every character is a node in the linked list. **Write** a function `isNotEqual()` that returns true if two strings are different; otherwise, returns true. Also determine the complexity of your function in terms of Big-Oh notation. [10]

The function prototype is given below:

```
bool isNotEqual (Node* head1, Node* head2);
```

A few testcases are:

| Testcase | Returned value |
|---|---|
| `head1 → c → s → e → NULL head2 → c → s → e → NULL` | `false` |
| `head1 → c → s → e → NULL head2 → c → s → NULL` | `true` |

While writing the function, consider the following Node class.

```
class Node{
   public:
      char c;
      Node* next;
}
```

**6.** Given a Doubly Linked List of character type values, **write** a function removeAllConsonants() that deletes all nodes from the list which are vowels and returns the head pointer of the list. Also determine the complexity of your function in terms of Big-Oh notation. [10]

The function prototype is given below.

```
Node* removeAllConsonants (Node* head)
```

A testcase is given below:

Input:

```
head --> o <--> r <--> a <--> n <--> g <--> e --> NULL
```

Output:

```
head --> o <--> a <--> e --> NULL
```

While writing the function, consider the following Node class.

```
class Node{
   public:
      char c;
      Node*
      next;
      Node*
      prev;
}
```

**7.** Given a Stack of integers, implemented as a Singly Linked List, **write** a function removeLargest() that removes the smallest element from the stack without changing the order of the other elements of the stack and returns the top pointer of the stack. Also determine the complexity of your function in terms of Big-Oh notation. [10]

The function prototype is given below:

```
Node* removeLargest (Node* top);
```

A testcase is given below:

Input:

top ⟶ 10 ⟶ 5 ⟶ 2 ⟶ 7 ⟶ 12 ⟶ NULL Output:

top ⟶ 10 ⟶ 5 ⟶ 2 ⟶ 7 ⟶ NULL

While writing the function, consider the following Stack and Node class.

| | |
|---|---|
| <pre>class Stack{<br> public:<br>   Node* top;<br>   Stack(){<br>     top=NULL;<br>   }<br>   void push(int data);<br>   void pop();<br>   //returns top most<br>element<br>   int peek();<br>};</pre> | <pre>class Node{<br>   public:<br>     int data;<br>     Node* next;<br>}</pre> |

**8.** **Circular Queue** is efficient than the linear queue as it does not waste storage space. The following figure shows a circular queue with 8 elements. [10]
Assume that, out of every 10 operations on this queue, 4 are Enqueue and the other 4 are Dequeue operations.

Do you think the given circular queue can be completely full or empty based on the assumption mentioned above? Justify your answer with a proper simulation of the operations.

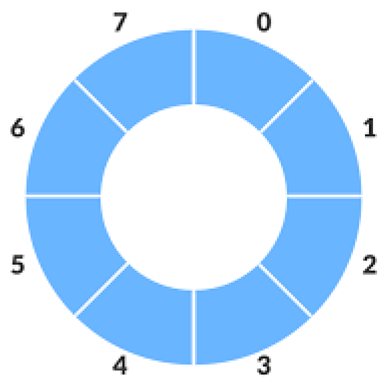


**Figure: A Circular Queue**

**9.** **Apply** the algorithmic method to change the following expression in postfix notation using stack. Show each step of the conversion including stack contents and postfix expression. [10]

$$( ( ( ( 3 * 4 ) - 8 / ( 8 / 2 ) / 2 ) * 3 * 4 ) + ( 9 - 5 ) * 4 )$$

**10.** **Evaluate** the value of the arithmetic expression given in Question 9 using the postfix notation you obtained from the answer of Question 9 through a stack. Show each step of the process including stack contents and relevant parameters. [10]