CSCI 2270 – Data Structures

Recitation 6, Sept 2018

Revision for Midterm1

Objectives

1. Pointers
2. Dynamic memory
3. Linked Lists
4. Stacks and Queues

1.Pointers

Refer to Recitation 2 writeup for details on Pointers.
Important concepts you must look at:

a) Address-of Operator (&)
b) What are Pointers?
c) Deferencing Operator (*)
d) Arrays and Pointers
e) Pass-by-value, Pass-by-pointer, Pass-by-reference, Pass-by-array
f) C++ basics: Structs and Classes

Practice MCQs/Objectives -

1) What will happen in this code?

```
Int a=100, b=200;
Int *p=&a, *q=&b;
p=q;
```

a) b is assigned to a
b) p now points to b
c) a is assigned to b
d) q now points to a

2) Consider the structure

```
struct Book {
    string name;
```

Revision for Midterm1

```
};
Book* b = new Book;
b->name = "book1";
cout << b->name << endl;
cout << (*b).name << endl;
```

Do we get the same output from the two cout statements?

3) You want pc to point to c, is this the correct way?

```
int c, *pc;
*pc=&c;
```

4) What is the output of this program?

```
#include <iostream>
    using namespace std;
    void Sum(int a, int b, int & c)
    {
        a = b + c;
        b = a + c;
        c = a + b;
    }
    int main()
    {
        int x = 2, y =3;
        Sum(x, y, y);
        cout << x << " " << y;
        return 0;
    }
```

a) 2 3
b) 6 9
c) 2 15
d) compile time error

2. Dynamic Memory

Refer to Recitation 3 writeup for details on Dynamic Memory.
Important concepts you must look at:

CSCI 2270 – Data Structures

Recitation 6, Sept 2018

Revision for Midterm1

a) Static memory vs Dynamic memory -
   when is the memory allocated and where are the respective variables stored?
b) Why do we need Dynamic memory?
c) New operator
d) De-allocation using delete
e) Why is deleting important?
f) Destructors

Practice MCQs/Objectives -

1)

```
int *a = new int;
*a = 7;
int *b = a;
delete a;
a=nullptr;
```

What would b point to now?

2) Is there a memory leak?

```
int main() {
  int * p = new int;
  delete p;

  int * q = new int;
  }
```

3) How to create a dynamic array of pointers (to integers) of size 10?

a) int *arr = new int *[10];
b) int **arr = new int *[10];
c) int *arr = new int [10];
d) Not Possible

CSCI 2270 – Data Structures

Recitation 6, Sept 2018

Revision for Midterm1

3. Linked Lists

A linked list is a data structure that stores a list, where each element in the list points to the next element.

- **Node** − Each Node of a linked list can store data and a link.
- **Link** − Each Node of a linked list contains a link to the next Node (unit of Linked List), often called 'next' in code.
- **Linked List** − A Linked List contains a connection link from the first link called Head. Every Link after the head points to another Node (unit of Linked List). The last node points to NULL.

Refer to recitation 3 for detailed explanation. Also, revise the following concepts:

1. Representation of a linked list
2. Traversal of a linked list using a **temp** pointer variable.
3. Insertion of a node in different positions of a linked list. (Do remember to update the next pointers properly)
4. Deletion of a node at different positions. (Make sure you store the memory to be freed before you update the next pointers)
5. For all the above problems, be careful with edge cases.

Practice problems -
For below node and linkedlist structure, what will each of the functions return for the inputs provided?

```
struct Node{
      int key;
      Node *next;
};
class List
{
  private:
    Node *head;

  public:
    List(){
       head = NULL;
    }
  int function_1();
  int function_2();
};
```

CSCI 2270 – Data Structures

Recitation 6, Sept 2018

Revision for Midterm1

1) Input:

1 → 3 → 4 → 5 → 7

Function:

```
int function_1(){
Node* temp1 = head;
Node* temp2 = head;

while(temp2 != NULL && temp2->next != NULL){
    temp1 = temp1->next;
    temp2 = temp2->next->next;
  }

return temp1->key;
}
```

Answer:

2)

Input:

1 → 3 → 4 → 5 → 7

Function:

```
int function_2(){
Node * temp = head;
head = head->next;
delete temp;

int total = 0;
int count = 0;

Node * temp2 = head;

while(temp2 != NULL){
```

```
   total += temp2->key;
   count++;
   temp2 = temp2->next;
  }

return total;
}
```

Answer:

Practice Exercise Questions:

1) Sum of alternate nodes of a linked list.
2) Switch the head and tail of a linked list.
3) Delete all occurrences of a key in a linked list.

4. Stacks and Queues

LinkedList implementations:

Stacks: Linear data structures which follow a particular order in which operations are performed. Order is **LIFO (Last in First out)**

- **Insertion** (also called push): occurs at the top. Create a new node. Point the next pointer of newNode node to the stackHead node. Move the stackHead pointer to the new node.
- **Deletion** at the top (also called pop): occurs at the top. Point a temp pointer to the stackHead. Move the stackHead pointer to next node. Delete temp pointer
- **Peek** (return the top element): **Return StackHead**
- **isEmpty** (true if the stack is empty else false): **Check stackHead**. If Null then true else false

Note: Operations in stacks are **ALWAYS** performed from one end **(TOP)**.

CSCI 2270 – Data Structures

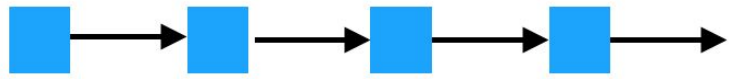
Recitation 6, Sept 2018

Revision for Midterm1

Practice Exercise Questions:

- Validate parenthesis pattern (check for balanced parenthesis)
- Build a simple calculator (like in assignment)
- Convert infix to postfix expression, Convert infix to prefix expression (and vise-versa) (Ex - Infix, prefix, and postfix expression evaluations :
  - Infix : 2+1 = Returns 3
  - Prefix : = + 2 1 returns 3
  - Postfix : 2 1 + = Returns 3)
- Implementing k stacks in a single array
- How do you insert at a particular place (index) in a stack using just the Push and Pop operations?

Queues: Linear data structures which follow a particular order in which operations are performed. Order is **FIFO (First in First out)**



Front **End**

Note: Pointers directions are from Front to End (just for visualisation)

Next pointer of End Node points to Null

Difference between stacks and queues:
In stack (last in first out), you remove the element which is the most recent. Contrary to stacks, elements which is the oldest gets removed from the queues. In other words, LIFO for stacks and FIFO for queues.
Operations:

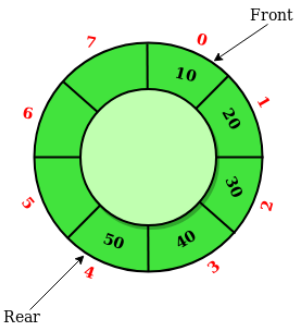
- **Enqueue :** Create a new node. Point the next pointer of End node to the new node. Move the End pointer to the new node.
- **Dequeue :** Point a temp pointer to the front. Move the Front pointer to next node. Delete temp pointer
- **Peek :** Return the front node.

CSCI 2270 – Data Structures

Recitation 6, Sept 2018

Revision for Midterm1

Circular queues: the last position is connected back to the first position to make a circle



Operations (Circular queues using arrays):

- **Peek:** Return the item at the front index of the array.
- **Enqueue :**
  - Check if the array (circular queue) is full or not.
    - If not, then insert the new element to the next index of **end/rear index. Move the end index.**

      arr[(end+1) % queue_size] = new_element
      end = (end+1) % queue_size

    - **If the array is full:** display queue is full
- **Dequeue :**
  - Check if the queue is empty or not.
    - If not,

      if (front == end) // if there is only 1 element in the queue
      - front = -1, end = -1

      Else
      - front = (front+1) % queue_size
    - **If the array is empty:** display queue is empty

Exercises on Queues:

- Implement stacks using queues (implement stacks functions using queues functions)
- Implement queues using stacks

Most of the queues functions are used in tree operations.