Harsh Kasliwal
2003085
C21

Page No.
Date

## Written Assignment-1

1> Differentiate b/w linear and non-linear data structure.

| | Linear data structure | Non linear data structure |
|---|---|---|
| i> | Elements are arranged sequentially | Elements are not arranged sequentia |
| ii> | Each element is connected to its previous and next element | Elements can have multiple connections with other elements or may have none at all. |
| iii> | Elements can be traversed in a single run since they are connected sequentially | multiple runs may be required to traverse all elements of this data structure. |
| iv> | They can be stored in memory easily, as memory itself is accessed sequentially. | Efficient methods of storing these data structures in memory must be developed. |
| v> | All elements are present on a single level. | Elements may be present on multiple levels. |
| vi> | E.g: Array, list, queue, stack | Eg: Graph, map, tree. |

② Explain ADT of stack:

A stack is a ADT (Abstract Data type) which stores a collection of items. In a stack items can only be added and removed from one end, thus a stack follows LIFO principle. It is based on stacks in real life, for example, a stack of books or plates. The insertion operation is called 'push' and the deletion operation is called 'pop'.

Other operations:

1. peek - Returns the element at the top of stack, without removin it.
2. isFull - Checks if stack is full or not.
3. isEmpty - checks if stack is empty or not.
4. display - lists all the elements in the stack.

To implement stack ADT, we need to use an array to hold the items and two variables: TOP and MAX. TOP stores the index of the element at top and MAX stores capacity. Then, the operation mentioned above are implemented as follows:

1> push(element): 1. If stack is full, print "overflow"
2. Otherwise set TOP=TOP+1 & array[TOP]=element.

2> pop(): 1. If stack is empty, print "Underflow"
2. Otherwise return array[TOP] and set TOP=TOP-1

3> isfull(): If TOP is MAX-1 return True otherwis return False.

4> isEmpty(): If TOP=-1 return True otherwise return False.

⑤ WAP to implement priority queue with the following operations:
1> Insert  2> Delete  3> Search  4> Display.

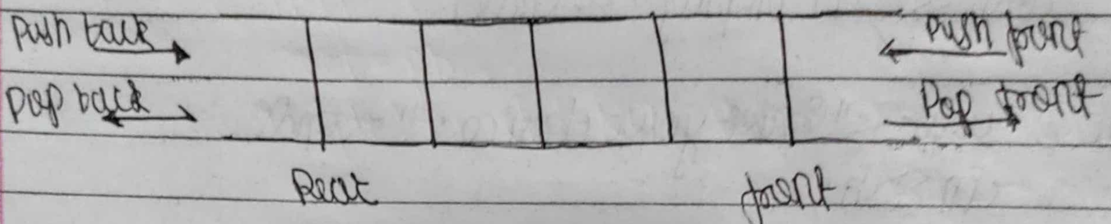
```
#include<iostream>
 using namespace std;
#define n ~~100~~ 5
~~void enqueue~~
int queue[n];
int size = 0, rear = -1, front = -1;

void enqueue(){
 int val;
 if (rear ==n-1)
 cout<<" Queue Overflow "<<endl;
 else {
     if(front==-1)
     front= 0;
     cout<<" Insert the element in queue: " <<endl;
     cin >>val;
     rear ++;
      size ++;
    queue [rear] = val;
  }
}
 void dequeue (){
   if(front == -1 ) || front > rear) {
       cout<<"Queue Underflow" ;
         return;
  }
```

```
        else {
         cout<<"Element deleted from queue is:"<< queue[front]
         front++;
          size--;
        }
    }

void search(){
        int i;
        if (front==-1 && rear==-1)
        cout<< "Queue is empty"<<endl;
        else {
            for (i=front; i <=rear; i++)
                if (queue[i]=element)
              cout<<"Element found inqueue with pos";
            }
        }
void Display(){
       if (front ==-1)
       cout <<"Queue is empty"<<endl;
       else {
         cout<<"Queue elements are:";
          for (int i = front ; i<=rear; i++)
          cout<< queue [i]<<" ";
          cout<<endl;
        }
    }.
```

```
int main() {
  int ch
  cout<<" 1) Insert"<< endl;
  cout <<" 2) Delete" <<endl;
  cout <<" 3) Search" <<endl;
  cout <<" 4) Display" << endl;

do { cout <<"Enter your choice:" <<endl
     cin>>ch;
     switch(ch) {
       case 1 : enqueue();
       break;
       case 2: dequeue();
       break;
       case 3 : search();
       break;
       case 4 : Display();
        break
       default : cout <<"Invalid" <<endl;
    }
 } while (ch!=4);
   return 0;
}
```

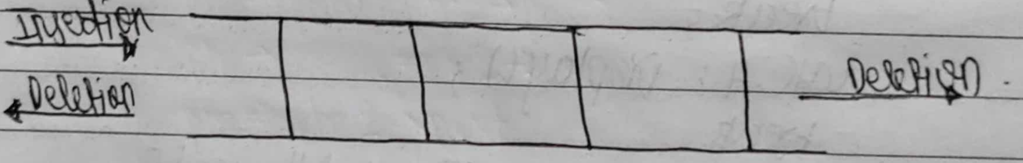
④ Write a short note on Double Ended Queue.

A Double ended queue is an ADT and is a generalized form of a queue in which elements can be added and removed from the front as well as the back.

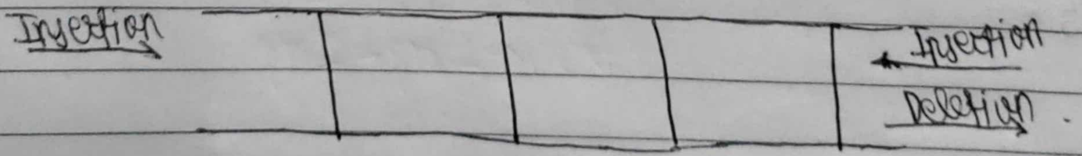


There are two types of deque:

1> Input restricted deque: In an input restricted deque, insertion can only be performed at one end but deletion can be performed at both ends.



2> Output restricted deque: In an output restricted deque, deletion can only be performed at one end, but insertion can be performed at both ends



(5) WAP to implement singly linked list application - Polynomial reperentation and addition.

```
# include<iostream>
  using namespace std;
# include <malloc.h>
typedef struct node{
      int coeff, pow;
      struct node *next;
   } NODE;
   NODE* add (NODE*root 1, NODE *root 2), *input();
          void display ~~Node~~(NODE*root);

int main(){
    NODE*root 1, *root 2, * root;
    cout<< "Enter details of polynomial 1: \n");
    root 1 = input();
    cout<< "Enter details of polynomial 2: \n");
    root 2 = input();
    cout<< " Polynomial 1: "); display(root 1);
    cout<<" \n Polynomial 2: "); display (root 2);
    root = add(root 1, root 2);
    cout<<" \n Addition of polynomials: "); display(root);
    return 0;
  }

NODE *add(NODE *root 1, NODE *root 2){
    NODE *ptr 1= root 1, *ptr 2= root 2, *ptr =NULL, *root=NULL;
    int i = 0;
```

```
while(ptr->next != NULL || ptr2 -> next != NULL){
     if(ptr==NULL){
         ptr=(NODE*)(malloc(sizeof(NODE)));
         ptr -> next=NULL;
     }
     if(i==0)
     root=ptr;
     if(ptr1->next != NULL &&(ptr2->next==NULL || ptr1->pow
                                       >ptr2->pow)){
      ptr -> pow= ptr1->pow;
      ptr->coeff =ptr1->coeff;
      ptr1 =ptr1->next;
   }
   else if (ptr2->next != NULL &&(ptr->next==NULL|| ptr1->pow<
                                                 ptr2->pow)){
      ptr -> pow =ptr2->pow;
      ptr -> coeff =ptr2->coeff;
      ptr ⇒ ptr2 ->next
    }
    else{
       ptr->coeff =ptr1->coeff + ptr2 -> coeff;
       ptr->pow =ptr1->pow;
       ptr1⇒ ptr1 ->next;
       ptr2 =ptr2->next;
    }
    ptr -> next = (NODE*) (malloc (sizeof(NODE)));
     ptr= ptr ->next;
    ptr->next=NULL;
    i++;
  }
  return root; }
```

Harsh Kasliwal
2003086 C21

Page No.
Date

```
NODE *input() {
    NODE * root = NULL, *ptr = NULL;
    int i, n, coeff, pow;
    cout<<" Enter no. of terms in the polynomial : ");
    cin >> n ;
    for(i=0 ; i<n ; i++) {
        cout<< "Enter coefficient & power of term "<<i ;
        cin >> coeff >> pow ;
        if (ptr == NULL) {
          ptr = (NODE*) (malloc (size of(NODE)));
          ptr -> next = NULL;
      }
    if (i==0)
        root = ptr;
      ptr -> coeff = coeff; ptr -> pow = pow;
     ptr -> next = (NODE*) (malloc (size of(NODE)));
      ptr = ptr -> next;
      ptr -> next = NULL;
    }
      return root;
   }
 void display (NODE *root) {
       NODE* ptr = root;
       while (ptr -> next != NULL) {
       cout<<" "<< ptr -> coeff << " x^"<< ptr->pow ;
        ptr = ptr -> next ;
        if (ptr -> next) != Null)
            cout<< {" + " ;
       }
  }
```