Post Graduate Government College, Sector 11, Chandigarh

PRACTICAL FILE FOR DATA STRUCTURE

SUBMITTED TO

( )

SUBMITTED BY

NAME:-

ROLL NO:-

CLASS:- BCA 2ND

Q1. DEFINE DATA STRUCTURE WITH VARIOUS TYPE?

Ans. A data structure is a specialized format for organizing, processing, retrieving and storing data. There are several basic and advanced types of data structures, all designed to arrange data to suit a specific purpose. Data structures make it easy for users to access and work with the data they need in appropriate ways. Most importantly, data structures frame the organization of information so that machines and humans can better understand it.

Types of data structures

The data structure type used in a particular situation is determined by the type of operations that will be required or the kinds of algorithms that will be applied. The various data structure types include the following:

Array. An array stores a collection of items at adjoining memory locations. Items that are the same type are stored together so the position of each element can be calculated or retrieved easily by an index. Arrays can be fixed or flexible in length.

Stack. A stack stores a collection of items in the linear order that operations are applied. This order could be last in, first out (LIFO) or first in, first out (FIFO).

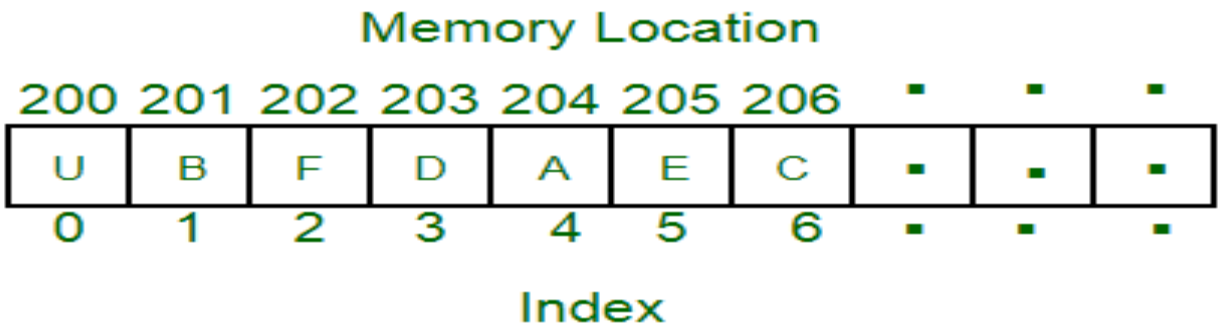
Queue. A queue stores a collection of items like a stack; however, the operation order can only be first in, first out.

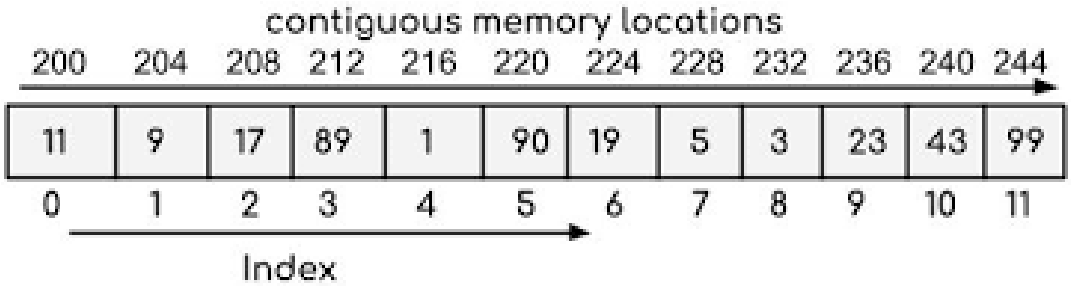
Linked list. A linked list stores a collection of items in a linear order. Each element, or node, in a linked list contains a data item, as well as a reference, or link, to the next item in the list.

Tree. A tree stores a collection of items in an abstract, hierarchical way. Each node is associated with a key value, with parent nodes linked to child nodes -- or subnodes. There is one root node that is the ancestor of all the nodes in the tree.

Q2. Memory representation of an Array?

Ans. An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value.





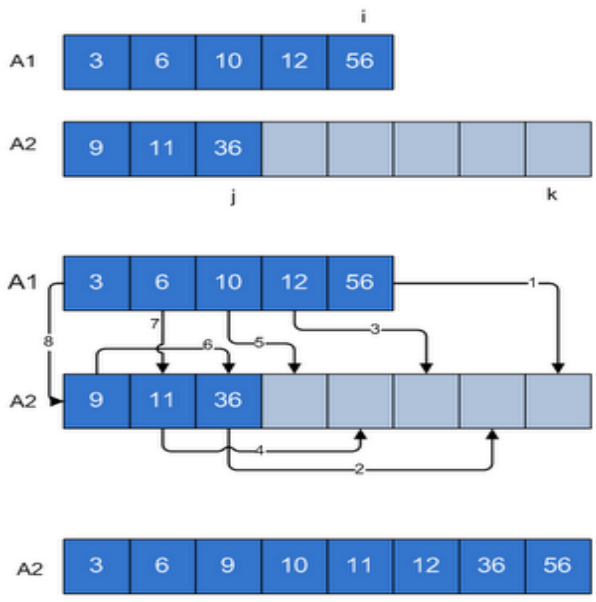
Q3 Explain merge sort and Algorithm?

Ans. An easier approach to merge two sorted arrays would be by traversing both the arrays to keep track of current element in both the arrays, finding the least value among the two and updating the final array with the least value.

Algorithm

Declare two arrays and input the array elements in both the arrays.

Traverse both the arrays. Find the minimum element among the current two elements of both the arrays and update the output array with the least value and increment its index to next position.



Q4. Explain the searching of an Array?

Ans. Finding a specific member of an array means searching the array until the member is found. It's possible that the member does not exist and the programmer must handle that possibility within the logic of his or her algorithm.

The linear search is a very simple algorithm. Sometimes called a sequential search, it uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for, and stops when either the value is found or the end of the array is encountered. If the value being searched for is not in the array, the algorithm will search to the end of the array.

Q5. Difference between array and linked list?

Array	Linked List
An array is a collection of elements of a similar data type.	Linked List is an ordered collection of elements of the same type in which each element is connected to the next using pointers.
Array elements can be accessed randomly using the array index.	Random accessing is not possible in linked lists. The elements will have to be accessed sequentially.
Data elements are stored in contiguous locations in memory.	New elements can be stored anywhere and a reference is created for the new element using pointers.
Insertion and Deletion operations are costlier since the memory locations are consecutive and fixed.	Insertion and Deletion operations are fast and easy in a linked list.
Memory is allocated during the compile time (Static memory allocation).	Memory is allocated during the run-time (Dynamic memory allocation).
Size of the array must be specified at the time of array declaration/initialization.	Size of a Linked list grows/shrinks as and when new elements are inserted/deleted.

Q6. Write Algorithm of Insertion?

At the beg

Algorithm

Step 1: IF PTR = NULL

Write OVERFLOW

Go to Step 7

[END OF IF]

Step 2: SET NEW_NODE = PTR

Step 3: SET PTR = PTR → NEXT

Step 4: SET NEW_NODE → DATA = VAL

Step 5: SET NEW_NODE → NEXT = HEAD

Step 6: SET HEAD = NEW_NODE

Step 7: EXIT

At the end

Algorithm

Step 1: IF PTR = NULL Write OVERFLOW

Go to Step 1

[END OF IF]

Step 2: SET NEW_NODE = PTR

Step 3: SET PTR = PTR - > NEXT

Step 4: SET NEW_NODE - > DATA = VAL

Step 5: SET NEW_NODE - > NEXT = NULL

Step 6: SET PTR = HEAD

Step 7: Repeat Step 8 while PTR - > NEXT != NULL

Step 8: SET PTR = PTR - > NEXT

[END OF LOOP]

Step 9: SET PTR - > NEXT = NEW_NODE

Step 10: EXIT

At the position

Algorithm

STEP 1: IF PTR = NULL

WRITE OVERFLOW

GOTO STEP 12

END OF IF

STEP 2: SET NEW_NODE = PTR

STEP 3: NEW_NODE → DATA = VAL

STEP 4: SET TEMP = HEAD

STEP 5: SET I = 0

STEP 6: REPEAT STEP 5 AND 6 UNTIL I

STEP 7: TEMP = TEMP → NEXT

STEP 8: IF TEMP = NULL

WRITE "DESIRED NODE NOT PRESENT"

GOTO STEP 12

END OF IF

END OF LOOP

STEP 9: PTR → NEXT = TEMP → NEXT

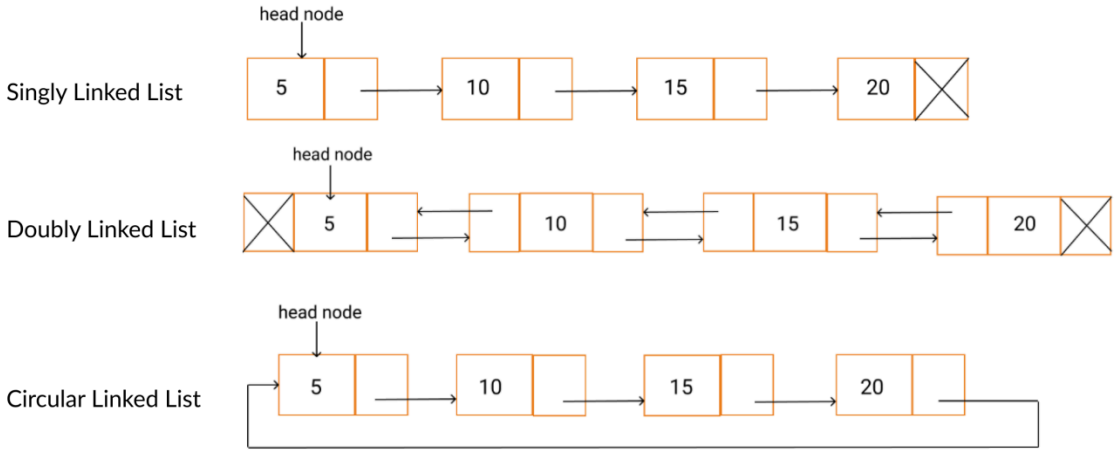
STEP 10: TEMP ⟶ NEXT = PTR

STEP 11: SET PTR = NEW_NODE

STEP 12: EXIT

Q7. Memory Representation of Linked list?

Ans. Linked lists can be represented in memory by using two arrays respectively known as INFO and LINK, such that INFO[K] and LINK[K] contains information of element and next node address respectively. ... It indicates that the node of a list need not occupy adjacent elements in the array INFO and LINK



ARRAYS

1.) To traverse a linear array to calculate the Average of an array?

#include <stdio.h>

```
int main(){
int a[10],n,sum, avg,i;
    printf("Enter the length of the array: ");
    scanf("%d",&n);
    for (i = 0; i <n; i++)
    {
        scanf("%d",&a[i]);
    }  sum=0;
    for ( i = 0; i <n; i++)
    {
        sum=sum+a[i];
        avg = sum/n;  }
    printf("Sum is: %d",sum);
    printf("\nAverage is: %d",avg);
   return 0;
}
```

2.) Program to insert an element at a given position in a linear array?

```
#include <stdio.h>
int main()
{
```

```
    int i,item,n,a[10];
    int pos;
  printf("Enter the length of the Array: ");
    scanf("%d",&n);
    for ( i = 0; i <n; i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Enter the position where you want to add element: ");
    scanf("%d",&pos);
    printf("Enter the element: ");
    scanf("%d",&item);
  for ( i = n; i>=pos; i--)
    {
        a[i]=a[i-1]  }
    a[pos]=item;
n++;
    for ( i = 0; i <n; i++)
    {
        printf("%d ",a[i]);
    }
 return 0;
}
```

3.) Program to insert an element in a sorted linear array?

```
#include <stdio.h>
int main()
{
    int i, item, n, a[10];int pos;

    printf("Enter the length of the Array: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }

    printf("Enter the element: ");
    scanf("%d", &item);
    i = n-1;
     while (item < a[i] && i >= 0)
    {
        a[i]=a[i-1];
```

```
        i--;
    }
    a[i + 1] = item;
    n++;
    for (i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
    return 0;
}
```

4.) Program to delete an element from a given position in linear array?

```
#include <stdio.h>
int a[10], i,  item, ub, lb, n, pos;
void Delete_beg(){
    n--;
    for ( i = 0; i < n; i++)
    {
        a[i]=a[i+1];
```

```
    }}
void Delete_end(){
    n--;
}
void Delete_pos(){
     printf("\nEnter the position for deletion: ");
    scanf("%d", &pos);
    for (i = pos; i < n - 1; i++)
    {
    a[i] = a[i + 1];
    }
    n--;
}
void display(){
    printf(" After Delete Your elements are: ");
    for (i = 0; i < n; i++)
    {
        printf(" %d ", a[i]);
    }}
int main()
{
```

```
    printf("---- Welcome to Deletion of element from array ----");
   printf("\n 1. Press 1 for Delete element at the beginning ");
   printf("\n 2. Press 2 for Delete element at the end ");
   printf("\n 3. Press 3 for Delete element at the any position ");
   int c;
   scanf("%d",&c);
   printf("Enter the length of the Array: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    printf("Your elements are: ");
    for (i = 0; i < n; i++)
    {
        printf(" %d ", a[i]);
    }
   switch (c)
```

```
    {
    case 1:
        Delete_beg();
        display();
        break;
     case 2:
        Delete_end();
        display();
        break;
     case 3:
        Delete_pos();
        display();
        break;
     default:
        break;
    }
    return 0;
}
```

5.) Program of linear search in unsorted linear array?

```
#include <stdio.h>
int main()
{
    int a[10], n, item;
    printf("Enter the length of the array: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
    }
    printf("Element item you wanna find: ");
    scanf("%d", &item);
    for (int i = 0; i < n; i++)
    {
        if (a[i] == item)
        {
            printf("Element Found at %d position", i);
            return 0;
        }
    }
    printf("Opps! Element not found");
```

```
    return 0;
}
```

7.) Program of Binary Search array?

```
#include <stdio.h>
int main()
{
    int i, beg, end, mid, n, find, a[10];

  printf("Enter number of elements: ");
  scanf("%d", &n);

  printf("Enter %d integers: ", n);

  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  printf("Enter value to find: ");
  scanf("%d", &find);
```

```
 beg = 0;
 end = n-1;
mid = beg+end/2;

 while (beg <= end) {
   if (a[mid] < find)
     beg = mid + 1;
   else if (a[mid] == find) {
     printf("%d found at loiation %d.\n",  find, mid+1);
     break;
   }
   else
     end = mid - 1;

   mid = (beg + end)/2;
 }
 if (beg > end)
   printf("Not found! %d isn't present in the list.\n", find);
   return 0;
}
```

8.) Program to Implement Bubble sort ?

```
#include <stdio.h>
void bubble(int a[],int n)
{
    int c,i,j;
    for ( i = 1; i<=n-1; i++)
    {
       for ( j = 0; j <=n-1-i; j++)
       {
           if (a[i]>a[i+1])
           {
              c=a[i];
              a[i]=a[i+1];
              a[i+1]=c;
           }}}}

int main()
{
    int a[]={2,5,3,8,6};
```

```
   bubble(a,5);
   for (int i = 0; i <=4; i++)
  printf("%d ",a[i]);

    return 0;
}
```

8.) Program to Implement Merge sort ?

```
#include <stdio.h>
int n, m;
int a[10], b[10], c[30];
int j, i;
void A_array()
{
    printf("Enter the length of the A array: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("Enter %d element: ",i);
        scanf("%d", &a[i]);
```

```
    }
}
void B_array()
{
    printf("Enter the length of the B array: ");
    scanf("%d", &m);
    for (j = 0; j < m; j++)
    {
        printf("Enter %d element: ",j);
        scanf("%d", &b[j]);
    }
}

void marge_sort(int a[], int b[], int c[], int n, int m)
{
    int i, j, k;
    i = j = k = 0;
    for (i = 0; i < n && j < m;)
    {
        if (a[i] < b[j])
        {
```

```
        c[k] = a[i];
        k++;
        i++;
    }
    else
    {
        c[k] = b[j];
        j++;
        k++;
    }
}

while (i < n)
{
    c[k] = a[i];
    k++;
    i++;
}
while (j < m)
{
    c[k] = b[j];
```

```
        k++;
        j++;
    }
}
void main()
{

    A_array();
    B_array();
    marge_sort(a, b, c, n, m);
    printf("\nAfter merging A and B array into C \n");
    for (i = 0; i < n + m; i++)
    {
        printf("%d ", c[i]);
    }
}
```

Linked List

1.) Program to implement operation like insert, delete and search on single linked list?

```
#include <stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *link;
};
struct node * Insert_beg(struct node *head, int s){
    struct node *e = malloc(sizeof(struct node));

    e->data = s;
    e->link = head;
    return e;
}
struct node * Insert_end(struct node *head, int s){
    struct node *ptr;
```

```
    struct node *e = malloc(sizeof(struct node));

    e->data = s;

    ptr = head;
    while(ptr->link!=NULL){
        ptr = ptr->link;
    }
    ptr->link = e;
    e->link = NULL;

    return head;
}
struct node * Insert_pos(struct node *head, int s, int pos){
    struct node *i;    // i is a position of curser
    struct node *e = malloc(sizeof(struct node));
    i = head;
    int j = 0;
    while(j!=pos-1){
        i = i->link;
        j++;
```

```
    }

    e->data = s;
    e->link = i->link;
    i->link = e;

    return head;
}
void display(struct node *ptr){
    while(ptr!=NULL){
        printf(" %d ",ptr->data);
        ptr = ptr->link;
    }
}

int main()
{
    struct node *head;
    struct node *a;
    struct node *b;
    head = malloc(sizeof(struct node));
```

```
    a = malloc(sizeof(struct node));
    b = malloc(sizeof(struct node));

    head->data = 2;
    head->link= a;

    a->data = 3;
    a->link = b;

    b->data = 4;
    b->link = NULL;
    int m;

printf("Enter the Element to add at the begninning: ");
scanf("%d",&m);
head = Insert_beg(head,m);

printf("Enter the Element to add at the end: ");
    scanf("%d",&m);
head = Insert_end(head,m);
```

```
printf("Enter which position you want to add: ");
int p;
scanf("%d",&p);
printf("Enter the Element to add at the end: ");
scanf("%d",&m);
head = Insert_pos(head,m,p);

display(head);
    return 0;
}
```

Deletion

```
#include <stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *link;

};
struct node *del_beg(struct node *head){
```

```
    struct node *ptr = head;
    head = head->link;
    free(ptr);
    return head;

}
struct node *del_pos(struct node *head, int pos){
    struct node *p = head;
    struct  node *q = head->link;
    for (int i = 0; i < pos-1; i++)
    {
        p = p->link;
        q = q->link;
    }
    p->link = q->link;
    free(q);
    return head;

}
struct node *del_end(struct node * head){
    struct node *p = head;
```

```
    struct  node *q = head->link;
while(q->link != NULL)
    {
        p = p->link;
        q = q->link;
    }
    p->link = NULL;
    free(q);
    return head;

}

void display(struct node *head){
    while(head!=NULL){
        printf(" %d ",head->data);
        head = head->link;
    }
}
int main()
{
```

```
  struct node *head;
struct node *b;
struct node *c;
struct node *d;

head = malloc(sizeof(struct node));
b = malloc(sizeof(struct node));
c = malloc(sizeof(struct node));
d = malloc(sizeof(struct node));

head->data = 1;
head->link = b;

b->data = 2;
b->link = c;

c->data = 3;
c->link = d;

d->data = 4;
d->link = NULL;
```

```
    display(head);
    head = del_beg(head);
head = del_pos(head,4);
head = del_end(head);
    printf("\n");

    display(head);
     return 0;
}
```

2.) Program to implement operation like insert, delete and search on double linked list?

```
#include <stdio.h>
#include<stdlib.h>



struct node{
    int data;
    struct node *prev;
    struct node *link;
```

```
};

void display(struct node *ptr){
    while(ptr!=NULL){
        printf(" %d ",ptr->data);
        ptr = ptr->link;
    }
}
struct node *insert_beg(struct node *head, int s){
    struct node *e = malloc(sizeof(struct node));

    head->prev = e;
    e->link = head;
    e->data = s;



return  e;
}

struct node *insert_end(struct node *head,int s){
    struct node *e = malloc(sizeof(struct node));
```

```
    struct node *ptr = head;
   e->data =s;
   e->link = NULL;
   e->prev = NULL;
   while(ptr->link!=NULL)
   {
       ptr= ptr->link;
   }
   ptr->link = e;
   e->prev = ptr;
   return head;
}

struct node *insert_pos(struct node *head,int s,int pos){
    struct node *e = malloc(sizeof(struct node));
    struct node *ptr = head;
    int j = 0;
    while (j!=pos-1)
    {
        ptr= ptr->link;
        j++;
```

```
    }
    e->data = s;
    e->link = ptr->link;
    e->prev = ptr;
    ptr->link = e;
return head;

}

int main()
{
    struct node *a;
    struct node *b;
    struct node *c;
    struct node *d;

    a = malloc(sizeof(struct node));
    b = malloc(sizeof(struct node));
    c = malloc(sizeof(struct node));
    d = malloc(sizeof(struct node));
```

```
    a->prev = NULL;
    a->data = 1;
    a->link = b;

    b->prev = a;
    b->data = 2;
    b->link = c;

    c->prev = b;
    c->data = 3;
    c->link = d;

    d->prev = c;
    d->data = 4;
    d->link = NULL;
display(a);
printf("\n");
// a = insert_beg(a,0);
// a = insert_end(a,5);
a = insert_pos(a,0,1);
```

```
display(a);
return 0;
}
```

Deletion

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *prev;
    struct node *link;
};

void display(struct node *ptr)
{
    while (ptr != NULL)
    {
        printf(" %d ", ptr->data);
```

```
        ptr = ptr->link;
    }
}
struct node *del_beg(struct node *head)
{

    struct node *ptr = head;
    head = head->link;
    head->prev = NULL;
    free(ptr);

    return head;
}

struct node *del_end(struct node *head)
{
    struct node *ptr = head;
    while (ptr->link != NULL)
    {
        ptr = ptr->link;
    }
```

```
    ptr->prev->link = NULL;
    free(ptr);
    return head;
}

struct node *del_pos(struct node *head, int pos)
{

    struct node *ptr = head;
    struct node *i;
    int j = 0;
    while (j != pos - 1)
    {
        ptr = ptr->link;
        j++;
    }
    i = ptr->link;
    ptr->link = i->link;
    i->link->prev = ptr;

    free(i);
```

```
    return head;
}

int main()
{
    struct node *a;
    struct node *b;
    struct node *c;
    struct node *d;

    a = malloc(sizeof(struct node));
    b = malloc(sizeof(struct node));
    c = malloc(sizeof(struct node));
    d = malloc(sizeof(struct node));

    a->prev = NULL;
    a->data = 1;
    a->link = b;

    b->prev = a;
```

```
    b->data = 2;
    b->link = c;

    c->prev = b;
    c->data = 3;
    c->link = d;

    d->prev = c;
    d->data = 4;
    d->link = NULL;

    display(a);
    printf("\n");
    // a = del_beg(a);
    // a = del_end(a);
    a = del_pos(a, 2);

    display(a);

    return 0;
}
```

Stack

1.) Creation of stack (using array) along with its two operation Push and Top?

```
#include <stdio.h>
#include <stdlib.h>
#define size 5
int stack[size], top = -1;

int isFull()
{
    if (top == size - 1)
    {
        return 1;
```

```
    }

    return 0;
}
int isEmpty()
{
    if (top == -1)
    {
        return 1;
    }

    return 0;
}
void push(int ele)
{

    if (isFull())
    {
        printf("\nStack overflow\n\n");
    }
    else
```

```
    {
        top++;
        stack[top] = ele;
    }
}
int pop()
{
    if (isEmpty())
    {
        return 0;
    }
    else
    {
        return stack[top--];
    }
}
void peek()
{
    if (isEmpty())
    {
        printf("\nStack is empty\n\n");
```

```
    }
    else
    {
        printf("Peek Element is: %d", stack[top]);
    }
}
void display()
{
    if (isEmpty())
    {
        printf("\nStack is empty\n\n");
    }
    else
    {
        printf("Stack Element \n");
        for (int i = 0; i <= top; i++)
        {
            printf(" %d ", stack[i]);
        }
    }
}
```

```
void pos_ele()
{
    int i;
    printf("Which location element you want to see: ");
    scanf("%d", &i);
    printf("%d", stack[i]);
}
int main()
{
    int a, ele;
    while (1)
    {
        printf("\n1. Push\n");
        printf("2. Pop\n");
        printf("3. Peek\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("6. Find Position Element\n");
        printf("Enter your choice: ");
        scanf("%d", &a);
```

```
switch (a)
{
case 1:
    printf("Enter the Element: ");
    scanf("%d", &ele);
    push(ele);
    break;
case 2:
    ele = pop();
    if (ele == 0)
    {
        printf("stack is empty\\underflow\n");
    }
    else
    {
        printf("%d is popped", ele);
    }
    break;
case 3:
    peek();
    break;
```

```
        case 4:
            display();
            break;

        case 5:
            exit(0);
        case 6:
            pos_ele();
            break;

        default:
            printf("Wrong Input!");
            break;
        }
    }
    return 0;
}
```

2.) Creation of stack (using Linked list) along with its two operation Push and Top?

```
#include <stdio.h>
#include<stdlib.h>

struct node{
    int data;
    struct node *link;
};
struct node* top =NULL;
void display(struct node *ptr){
    while(ptr!=NULL){
        printf(" %d\n",ptr->data);
        ptr = ptr->link;
    }
}
void push(){
    struct node *temp;
    temp = malloc(sizeof(struct node));
    printf("Enter the element: ");
    scanf("%d",&temp->data);
    temp->link = top;
    top = temp;
```

```
}
void pop(){
    struct node *t;
    if(top == NULL){
        printf("Stack is empty\n");
    }
    else{
        t = top;
        printf("Popped element: %d\n",top->data);
        top = top->link;
        t->link = NULL;
        free(t);
    }
}
int main()
{
    while(1){
        int a;
        printf("1. Push\n");
        printf("2. PoP\n");
        printf("3. Display\n");
```

```
printf("4. Exit\n");
printf("Enter you choice: ");
scanf("%d",&a);
switch (a)
{
case 1:
    push();
    break;
case 2:
    pop();
    break;
case 3:
    display(top);
    break;
case 4:
    exit(0);
    break;

default:
    printf("Wrong Input!");
    break; } }
```

```
return 0;
}
```

Queue

1.) Implementation of enqueue and dequeue operation using linear queue using array?

```
#include <stdio.h>
#include<stdlib.h>
#define size 5
int queue[size];
int f = 0;
int b = 0;

void enqueue(int val){
    if(size==b){
```

```
        printf("Queue is Full\n");
    }
    else{
        queue[b] = val;
        b++;
    }
}
void dequeue(){
    if(f==b){
        printf("Queue is Empty\n");
    }
    else{
        for (int i = 0; i < b-1; i++)
        {
            queue[i]=queue[i+1];
        }
        b--;

    }
}
void display(){
```

```
    if(f==b){
        printf("Queue is Empty");
    }
    else{
        for (int i = f; i < b; i++)
        {
            printf(" %d ",queue[i]);
        }
    printf("\n");
    }
}
int main()
{

    enqueue(1);
    enqueue(2);
    enqueue(3);
    enqueue(4);
    enqueue(5);

    display();
```

```
    dequeue();
    dequeue();
    dequeue();

    display();
    return 0;
}
```

2.) Implementation of enqueue and dequeue operation using linear queue using Linked list?

```
#include <stdio.h>
#include<stdlib.h>
struct node *f = NULL;
struct node *b = NULL;
struct node{
    int data;
    struct node *link;

};
```

```
void display(struct node *p){
    while(p!=NULL){
        printf(" %d ",p->data);
        p = p->link;
    }
    printf("\n");
}

void enqueue(int val){
    struct node *n = malloc(sizeof(struct node));
    if(n==NULL){
        printf("Queue is Full");
    }
    else{
        n->data = val;
        n->link = NULL;
        if(f==NULL){
            f=b=n;
        }
        else{
            b->link = n;
```

```
            b=n;
        }
    }
}
int dequeue(){
    int val = -1;
    struct node *ptr = f;
    if(f==NULL){
        printf("Queue is Empty");
    }
    else{
        f = f->link;
        val = ptr->data;
        free(ptr);

    }
        return val;
}
int main()
{
```

```
    enqueue(1);
    enqueue(2);
    enqueue(3);
    enqueue(4);
    display(f);
    dequeue();
    display(f);
    return 0;
}
```