

1. Implement Stack using array

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

```
#define STACK_SIZE 5
```

```
int top = -1;
```

```
int s[10];
```

```
int item;
```

```
void push();
```

```
{
```

```
if (top == STACK_SIZE - 1)
```

```
{
```

```
printf("Stack overflow\n");
```

```
return;
```

```
top = top + 1;
```

```
s[top] = item;
```

```
}
```

```
int pop()
```

```
{
```

```
if (top == -1)
```

```
return -1;
```

```
return s[top--];
```

```
}
```

```
void display()
```

```
{
```

```
int i;
```

```
int (top == -1)
```

```
{
```

```
printf("Stack is empty\n");
```

```
return;
```

```
}
```

```

printf ("contents of the stack\n");
for (i = top; i >= 0; i--)
{
    printf ("%d\n", s[i]);
}
}

void main()
{
    int item_deleted;
    int choice;
    for (i;)
    {
        printf ("\n 1.push\n 2.pop\n 3.display\n
        4. exit\n");
        printf ("enter the choice");
        switch (choice)
        {
            case 1: printf ("enter the item to be inserted\n");
                    scanf ("%d", &choice);
                    push();
                    break;
            case 2: item_deleted = pop();
                    if (item_deleted == -1)
                        printf ("stack is empty\n");
                    else
                        printf ("item deleted is %d\n", item_deleted);
                    break;
            case 3: display (top, s);
            default: exit (0);
        }
    }
}

```

1. push  
2. pop  
3. display  
4. edit

Output : It takes character, integer

1. push      (char) = (digit) or string

2. pop      (digit)

3. display      set width > number

4. edit      set pointer + stacksize

enter the choice

1.

enter the item to be inserted

12.

enter the choice

enter the item to be inserted

Enter the choice

The elements of stack.

24

12

Enter the choice

Exit point

- 1.PUSH
- 2.POP
- 3.DISPLAY
- 4.EXIT

Enter the Choice:1

Enter a value to be pushed:12



Enter the Choice:1

Enter a value to be pushed:24

Enter the Choice:1

Enter a value to be pushed:98

Enter the Choice:3

The elements in STACK

98

24

12

Press Next Choice

Enter the Choice:2

The popped elements is 98

Enter the Choice:3

The elements in STACK

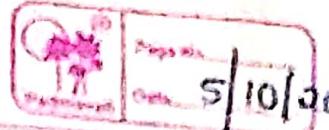
24

12

Press Next Choice

Enter the Choice:4

EXIT POINT



2. Write a program to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), \* (multiply) & / (divide).

$\Rightarrow$  #include <stdio.h>

#include <string.h>

int f (char symbol)

{

switch (symbol)

{

case '+':

case '-' : return 2;

case '\*':

case '/' : return 4;

case '^':

case '\$': return 5;

case 'c': return 8;

case '#': return -1;

default : return 8;

}

} int g (char symbol) {

switch (Symbol)

{

case '+':

case '-' : return 1;

case '\*':

case '/' : return 7;

```
case 'n' :  
case '$' : return 6;  
case '(' : return 9;  
case ')' : return 0;  
default : return 7;  
}
```

}

```
Void infix_postfix (char infix[], char postfix[])  
{
```

```
int top, i, j;  
char s[30], symbol;  
top = -1;  
s[++top] = '#';  
j = 0;  
for (i = 0; i < strlen(infix); i++)
```

```
{
```

```
symbol = infix[i];
```

```
while (F(s[top]) > G(symbol))
```

```
{
```

```
postfix[j] = s[top--];
```

```
j++;
```

```
{
```

```
if (F(s[top]) != G(symbol))
```

```
{
```

```
s[top++] = symbol;
```

```
else
```

```
top--;
```

```
}
```

```

while (s[top] != '#')
{
    postfix[j++] = s[top];
    if (s[top] == ')')
        j--;
    else if (s[top] == '(')
        j++;
    top++;
}

```

Void main ()

```

{ int i;
char infix[20];
char postfix[20];
printf ("enter valid infix expression\n");
scanf ("%s", infix);
infix_to_postfix (infix, postfix);
printf ("the postfix expression is\n");
printf ("%s", postfix);
}

```

Output:

Enter valid infix expression  
 $(A + (B - C) * D)$

the post fix expression is

$ABCD-+*$

**enter the valid infix expression**

**(A+ (B-C) \*D)**



**the postfix expression is**

**....Program finished with exit code 0  
Press ENTER to exit console.**

19/10/2022

Linear Queue

3) ~~as insert & delete of display.~~

```

#ifndef include <stdio.h>
#ifndef include <stdlib.h>
#ifndef include <string.h>
int item, front=0, rear=-1, q[queue_size];
void insert_rear()
{
    if (Count == queue_size + 1)
    {
        printf ("queue overflow");
        return;
    }
    rear = (rear + 1) % queue_size;
    q[rear] = item;
    Count++;
}
int deletefront()
{
    if (front > rear)
    {
        front = 0;
        rear = -1;
        return -1;
    }
    return q[front + 1];
}
void displayq()
{
    int i;
    if (front > rear)

```

**1:PQinsert**

**2:PQdelete**

**3:PQdisplay**

**4:Exit**

**Enter the choice**

**1**

**Enter the priority number**

**10**

**only 3 priority exists 1 2 3**

**PRIORITY QUEUE**

**\*\*\*\*\***

**1:PQinsert**

**2:PQdelete**

**3:PQdisplay**

**4:Exit**

Report No.  
Date 19/10/2020

~~Part 3~~

4) Circular Queue

as 1) insert 2) delete 3) display

```

#include <stdio.h>
#include <stdlib.h>
#define que_SIZE 3
int item, front = 0, rear = -1, q[que_SIZE], count = 0;
void insertrear()
{
    if (count == que_SIZE)
    {
        printf("queue overflow");
        return;
    }
    rear = (rear + 1) % que_SIZE;
    q[rear] = item;
    count++;
}
int deletefront()
{
    if (count == 0) return -1;
    item = q[front];
    front = (front + 1) % que_SIZE;
    count--;
    return item;
}
void display()
{
    int i, f;
    if (count == 0)
    {
        printf("empty");
    }
    else
    {
        for (i = front; i != rear; i = (i + 1) % que_SIZE)
        {
            printf("%d ", q[i]);
        }
        printf("%d", q[rear]);
    }
}

```

```
printf ("queue is empty");
```

```
return;
```

```
}
```

```
f = front;
```

```
printf ("contents of queue\n");
```

```
for (i=0 ; i<= count ; i++)
```

```
{
```

```
printf ("%d\n", q[f]);
```

```
f = (f+1) % que_size;
```

```
}
```

```
}
```

```
void main()
```

```
{
```

```
int choice;
```

```
for (i;)
```

```
{
```

```
printf ("1. Insert rear\n2. Delete front\n
```

```
3. display\n4. exit\n");
```

```
scanf ("enter the choice");
```

```
scanf ("%d", &choice);
```

```
switch (choice)
```

```
{
```

```
case 1 : printf ("Enter the item to be inserted");
```

```
scanf ("%d", &item);
```

```
insertrear();
```

```
break;
```

```
case 2 : item = deletefront();
```

```
if (item == -1)
```

```
printf ("queue is empty\n");
```

```
else  
    cout << "item deleted is %d\n", item);  
break;  
case 3: display();  
break;  
default: exit(0);  
}
```

output:

1. insert rear
2. delete front
3. display
4. exit

Enter the choice : 1

Enter the element to be inserted : 10.

Enter the choice : 1

Enter the element to be inserted : 20

Enter the choice : 3

Content of queue

10

20

- 1.Insert rear
- 2.Delete front
- 3.Display
- 4.exit

Enter the choice : 1

Enter the item to be inserted :10

- 1.Insert rear
- 2.Delete front
- 3.Display
- 4.exit

Enter the choice : 1

Enter the item to be inserted :10

- 1.Insert rear
- 2.Delete front
- 3.Display
- 4.exit

Enter the choice : 1

Enter the item to be inserted :30

- 1.Insert rear
- 2.Delete front
- 3.Display
- 4.exit

Enter the choice : 1

contents of queue

10  
10  
30  
10

- 1.Insert rear
- 2.Delete front
- 3.Display
- 4.exit

Enter the choice :

5) linked list :- inserting, deleting at rear & front  
 searching, concat display  
 #include <stdio.h>  
 #include <stdlib.h>  
 struct node  
 {  
 int info;  
 struct node \*link;  
 };  
 typedef struct node \*Node;  
 Node getNode()  
 {  
 Node n;  
 n = (Node) malloc (sizeof (struct node));  
 if (n == NULL)  
 {  
 printf ("mem full\n");  
 exit (0);  
 }  
 return n;  
 }  
 void freeNode (Node n)  
 {  
 free (n);  
 }  
 Node insertFront (Node first, int item)  
 {  
 Node temp;  
 temp = getNode();
 }

```
temp->info item  
temp->link NULL;  
(if (first == NULL)  
return temp;  
temp->link first;  
first = temp;  
return first;
```

}

node delete front (Node first)

{

Node temp;

if (first == NULL)

{

printf ("list empty can't delete\n");

return first;

}

temp = first;

temp = temp->link

printf ("item deleted at front end is = %d\n",  
temp->info);

free (first);

return temp;

}

Node insert\_recs (Node first, int item)

{

Node temp, cur;

temp = getnode();

temp->info = item;

temp->link = NULL;

```
if (first == NULL);
    return temp;
```

```
cur = first;
```

```
while (cur->link != NULL)
```

```
    cur = cur->link;
```

```
    cur->link = temp;
```

```
return first;
```

b  
Node deletion (Node first)

```
{ Node cur, prev;
```

```
if (first == NULL)
```

```
    printf ("list empty");
```

```
    return first;
```

```
}
```

```
if (first->link != NULL)
```

```
{
```

```
    prev = cur;
```

```
    cur = cur->link;
```

```
}
```

```
printf ("item deleted at rear-end is %d",
```

```
    cur->info);
```

```
free (cur);
```

```
prev->link = NULL;
```

```
return first;
```

```
}
```

c  
Node ordered-list (int item, node first)

```
{
```

Scanned with CamScanner

```
node temp, prev, cur;  
temp = getnode();  
temp->info = item;  
temp->link = NULL;  
if (prev == NULL) return temp;  
if (item <= prev->info)  
    a
```

```
temp->link = first;
```

```
return temp;
```

```
b
```

```
prev = NULL;
```

```
cur = first;
```

```
while (cur != NULL && cur->info > item)
```

```
c
```

```
prev = cur;
```

```
cur = cur->link;
```

```
d
```

```
prev->link = temp;
```

```
temp->link = cur;
```

```
return first;
```

```
e
```

```
node delete_info(int key, node first)
```

```
f
```

```
node prev, cur;
```

```
if (first == NULL)
```

```
g
```

```
printf("list is empty\n");
```

```
return NULL;
```

```
h
```

if (key == first->info)

cu = first;

first = first->link;

freemode(cu);

return first;

}

prev = NULL;

cu = first;

while (cu != NULL)

{

if (key == cu->info) break;

prev = cu;

cu = cu->link;

}

if (cu == NULL)

else

printf ("Search is successful\n");

return first;

}

prev->link = cu->link;

printf ("Key deleted & d: %d", cu->info);

freemode(cu);

return first;

}

Void display (Node first)

{

node temp;

if (first == NULL)

printf ("list empty");  
for (temp = first; temp != NULL; temp = temp->next)  
printf ("%d\n", temp->info);

}

void main ()

{

int item, choice, key;

Node \*first = NULL;

for (i ;

choice = 1; insert\_front (&first, &item) | 2: delete\_front (&first, &item);  
insert\_rear (&first, &item) | 3: delete\_rear (&first, &item);  
order\_list (&first); | 4: display (&first); | 5: exit (0); | 6:  
delete\_info (&first, &item); | 7: display (&first); | 8: (exit (0));

printf ("enter choice (1-8): ");

switch (choice)

{

case 1: printf ("enter the item at front-end\n");

scanf ("%d", &item);

first = insert\_front (first, item);

break;

case 2: first = delete\_front (first);

break;

case 3: printf ("enter the item at rear-end\n");

scanf ("%d", &item);

first = insert\_rear (first, item);

break;

case 4 : pfirst = delete\_rear(first);  
break;

case 5 : printf ("enter the item to be inserted  
at [n]");

scanf ("%d", &item);

first = ordered\_list(item, first);

break;

case 6 : printf ("enter the item to be deleted");

scanf ("%d", &key);

first = delete\_info(key, first);

break;

case 7 : display(first);

break;

default : exit(0);

break;

}

```
1:Insert_Front  
2:Delete_End  
3:Display_List  
4:Count_Items  
5:Search_Items  
6:Order_List  
7:Exit  
enter the choice  
1  
enter the item at front-end  
12  
1:Insert_Front  
2:Delete_End  
3:Display_List  
4:Count_Items  
5:Search_Items  
6:Order_List  
7:Exit  
enter the choice  
1  
enter the item at front-end  
12
```

Scanned with CamScanner

```
1:Display_List  
2:Exit  
enter the choice  
2  
the choice is incorrect so 10  
1:Insert_Front  
2:Delete_End  
3:Display_List  
4:Count_Items  
5:Search_Items  
6:Order_List  
7:Exit  
enter the choice  
2  
1:Insert_Front  
2:Delete_End  
3:Display_List  
4:Count_Items  
5:Search_Items  
6:Order_List  
7:Exit  
enter the choice  
2  
choice of item is the list is 10
```

Scanned with CamScanner

```
1:Enter the item to be searched  
2:  
Search is unsuccessful  
1:Insert_Front  
2:Delete_End  
3:Display_List  
4:Count_Items  
5:Search_Items  
6:Order_List  
7:Exit  
enter the choice  
2  
1:Displaying ordered_List  
2:Displaying ordered_List  
1  
2  
3:  
1:Insert_Front  
2:Delete_End  
3:Display_List  
4:Count_Items  
5:Search_Items  
6:Order_List  
7:Exit  
enter the choice
```

Scanned with CamScanner

6) linked list: creating & reversing

#include <stdio.h>

#include <conio.h>

struct node

{

int info;  
struct node \*link;

};

typedef struct node \*Node;

Node getnode()

{

Node n;

n = (Node) malloc (sizeof (struct node));

if (n == NULL)

.

printf ("mem full");

exit (0);

}

return n;

}

Node insertnay (Node first, int item)

{

Node temp, cur;

temp = getnode();

temp->info = item;

temp->link = NULL;

if (first == NULL)

return temp;

cur = first;

```
while (cur->link != NULL)
```

```
    cur = cur->link;
```

```
    cur->link = temp;
```

```
return first;
```

```
}
```

```
Void display (node first)
```

```
{
```

```
Node temp;
```

```
if (first == NULL)
```

```
printf ("list empty\n");
```

```
for (temp = first; temp != NULL; temp = temp->link)
```

```
printf ("%d\n", temp->info);
```

```
}
```

```
node concat (node first, node second)
```

```
{
```

```
node cur;
```

```
if (first == NULL)
```

```
return second;
```

```
if (second == NULL)
```

```
return first;
```

```
cur = first;
```

```
while (cur->link != NULL)
```

```
    cur = cur->link;
```

```
    cur->link = second;
```

```
    return first;
```

```
}
```

node reverse (node first)

{ node cur, temp;

cur = NULL;

while (first != NULL)

{ temp = first;

first = first → link;

temp → link = cur;

cur = temp;

}

return cur;

}

void main ()

{

int item, choice, pos, i, n;

Node first = NULL, a, b;

for (i; i)

{

printf ("1. insert front\n 2. concat\n 3. reverse\n 4. display\n 5. exit\n");

printf ("enter the choice\n");

scanf ("%d", &choice);

switch (choice) {

{

case 1 : printf ("enter the item\n");

scanf ("%d", &item);

first = insert-rear (first, item);

break;

case 2 : print if entered no. of nodes in  $a[n]$  ;  
scanf ("%d", &h);  
 $a = \text{NULL};$   
 $\text{for } (i=0; i < n; i++)$

d

printf ("enter the item[n]:");  
scanf ("%d", &item);  
 $a = \text{insert\_real}(a, item);$

b

printf ("enter no. of nodes in  $a[n]$  ");  
scanf ("%d", &n);  
 $b = \text{NULL};$   
 $\text{for } (i=0; i < n; i++)$

e

printf ("enter the item[n]:");  
scanf ("%d", &item);  
 $b = \text{insert\_real}(b, item);$

f

$a = \text{concat}(a, b);$

display(a);

break;

case 3 : first = reverse(first);

display(first);

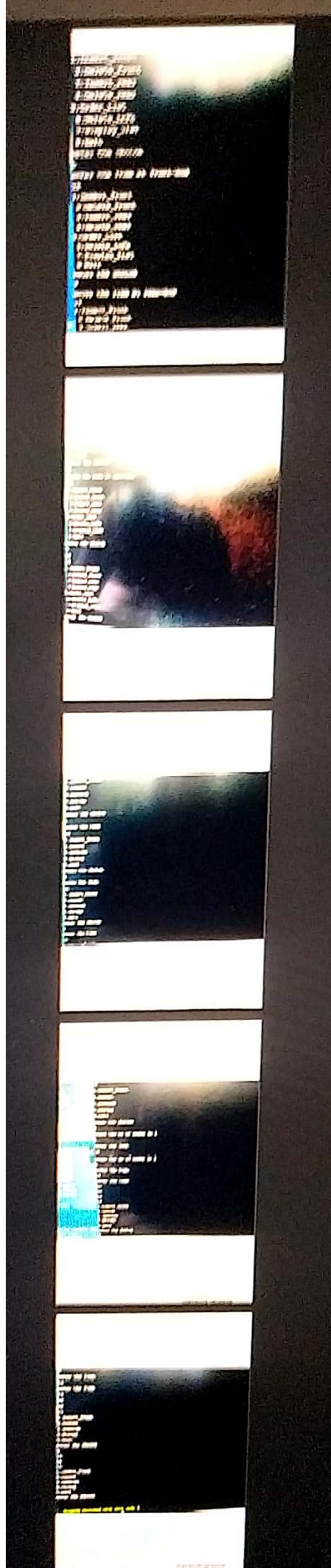
break;

case 4 : display(first);

break;

default & exit (0);

}





~~7) Implement linked list: Insert front, delete rear, display, sort, search, count.~~

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

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

```
struct node
```

```
{
```

```
int info;
```

```
struct node*link;
```

```
}
```

```
typedef struct node* NODE;
```

```
NODE getnode()
```

```
{
```

```
Node x;
```

```
x = (NODE)malloc(sizeof(struct node));
```

```
if(x == NULL)
```

```
{
```

```
printf("mem full\n");
```

```
exit(0)
```

```
}
```

```
return x;
```

```
}
```

```
Void freeNode(NODE x)
```

```
{
```

```
free(x);
```

```
}
```

```
NODE insert_front(NODE first, int item)
```

```
{
```

```
NODE temp;
```

```
temp = getnode();
```

```
temp->info = item;
```

```
temp->link = NULL;
```

if (first == NULL)

return temp;

temp->link = first;

first = temp;

return first;

}

NODE delete rear (NODE first)

{

NODE cur, prev;

if (first == NULL)

{

printf ("list empty can't delete\n");

return first;

}

if (first->link == NULL)

{

printf ("item deleted is %d\n", first->info);

free(first);

return NULL;

}

prev = NULL;

cur = first;

while (cur->link != NULL)

{

prev = cur;

cur = cur->link;

}

printf ("item deleted at rear-end is %d\n", cur->info);

free(cur);

prev->link = NULL;  
return first;

void display (NODE first)

{  
NODE temp;

if (first == NULL)

printf ("list empty can't display\n");

for (temp = first; temp != NULL; temp = temp->link)

{

printf ("%d\n", temp->info);

}

}

int length (NODE first)

{

NODE cur;

int count = 0;

if (first == NULL)

return 0;

cur = first;

while (cur != NULL)

{

count ++;

cur = cur->link;

}

return count;

{

void search (int key, NODE first)

{

NODE cur;  
if(first == NULL)

{  
printf("list empty\n");}

return;

}

cur = first;

while(cur != NULL)

{  
if(key == cur->info)

break;

cur = cur->link;

}

if (cur == NULL)

{

printf("search unsuccessful\n");

return;

}

printf("search successful\n");

}

NODE asc(NODE print)

{

NODE prev=first;

NODE cur = NULL;

int temp;

if(print == NULL){

return 0; }

else {

while(prev != NULL) {

```
curr = prev->link ;  
while (curr != NULL) {  
    if (prev->info > curr->info) {  
        temp = prev->info ;  
        prev->info = curr->info ;  
        curr->info = temp ;  
    }  
    curr = curr->link ;  
}
```

```
prev = prev->link ;
```

```
}  
return first ;
```

```
NODE del (NODE first) -
```

```
{  
    NODE prev = first ;
```

```
    NODE curr = NULL ;
```

```
    int temp ;
```

```
    if (first == NULL)  
    {
```

```
        return NULL ;
```

```
}  
else {
```

```
    while (prev != NULL) {  
        curr = prev->link ;
```

```
        while (curr != NULL) {
```

```
            if (prev->info < curr->info) {  
                temp = prev->info ;
```

prev->info = cur->info;

cur->info = temp;

}

cur = cur->link ;

prev = prev->link ;

} return first ;

}

int main() {

int item, choice, count, key, option ;

NODE first = NULL ;

for(;;) {

printf ("1: Insert front\n 2: Delete rear\n 3: display  
list\n 4: count-item\n 5: search item\n 6: order  
list\n 7: exit\n");

printf ("Enter the choice");

scanf ("%d", &choice);

switch (choice) {

case 1 : printf ("\nEnter the item at front-end\n");

scanf ("%d", &item);

first = insert-front (first, item);

break ;

case 2 : first = delete-rear (first) ; break ;

case 3 : display (first) ; break ;

case 4 : count = length (first) ;

printf ("Length = %d\n", count) ; break ;

case 5 : printf ("Enter the item to be searched\n");

scanf ("%d", &key) ;

search (key, first) ;

break ;



```
case 6 : printf ("\n1: ascending ordered list\n"
2: descending Ordered list\n");
scanf ("%d", &option);
if (option == -1)
{
    first = asc (first);
    display (first);
}
else {
    first = des (first);
    display (first);
    break;
}
default : exit (0);
}
return 0;
}
```

```
1:Insert_front  
2:Delete_rear  
3:Display_list  
4:Count_items  
5:Search_items  
6:Order_list  
7:Exit  
enter the choice  
4  
enter the item at front-end  
12  
  
1:Insert_front  
2:Delete_rear  
3:Display_list  
4:Count_items  
5:Search_items  
6:Order_list  
7:Exit  
enter the choice  
4  
enter the item at front-end  
12
```

Scanned with CamScanner

```
4:Order_list  
5:Exit  
enter the choice  
2  
item deleted in descending list  
1:Insert_front  
2:Delete_rear  
3:Display_list  
4:Count_items  
5:Search_items  
6:Order_list  
7:Exit  
enter the choice  
3  
13  
  
1:Insert_front  
2:Delete_rear  
3:Display_list  
4:Count_items  
5:Search_items  
6:Order_list  
7:Exit  
enter the choice  
4  
Search of items in the list is 3  
item found.
```

Scanned with CamScanner

```
enter the item to be searched  
23  
Search is unsuccessful  
  
1:Insert_front  
2:Delete_rear  
3:Display_list  
4:Count_items  
5:Search_items  
6:Order_list  
7:Exit  
enter the choice  
6  
  
1:ascending ordered_list  
2:descedding ordered_list  
1  
13  
  
1:Insert_front  
2:Delete_rear  
3:Display_list  
4:Count_items  
5:Search_items  
6:Order_list  
7:Exit  
enter the choice
```

Scanned with CamScanner

8) program to complement stack

```
#include <cslib.h>
```

```
#include <conio.h>
```

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

struct node

{

int data

top →

struct node \*next;

}

struct node \*top = NULL;

void push()

{

struct node \*new\_node;

new\_node = (struct node \*) malloc

printf ("enter the element\n");

scanf ("%d", &new\_node->data);

new\_node->next = NULL;

If (top == NULL)

{}

top = new\_node;

}

else

{

new node  $\rightarrow$  next = top;  
 $\downarrow$   
 $\uparrow$   
top = new node

void pop()

```

if (top == null)
    printf ("stack empty\n")
else
{
    temp = top;
    while (temp != null)
    {
        printf ("%d\n", temp->data);
        temp = temp->next;
    }
}

```

void main()

```

{
    int choice;
    while (1)
    {
        printf ("1.push\n");
        printf ("2.pop\n");
        printf ("3.display\n");
        printf ("enter choice");
        scanf ("%d", &choice);
        switch (choice)
        {

```

case 1: push(); break;

case 2: pop(); break;

case 3: display(); break;

default exit();

4:Exit

enter the choice

1  
enter the item at front-end

12

1:Insert\_front

2>Delete\_front

3:Display\_list

4:Exit

enter the choice

2  
item deleted at front-end is=12

1:Insert\_front

2>Delete\_front

3:Display\_list

4:Exit

enter the choice

3

12

1:Insert\_front

2>Delete\_front

3:Display\_list

4:Exit

enter the choice

1). Implementing queue using linked list.

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

```
struct node
```

```
{
```

```
int node
```

```
&
```

```
struct node *next)
```

```
}
```

```
struct node *front=NULL, *rear=NULL;
```

```
void insert()
```

```
{
```

```
struct node *new_node
```

```
new_node=(struct node*)malloc(sizeof(struct node))
```

```
new_node=(struct node*)malloc(sizeof(struct node))
```

```
printf("enter element\n");
```

```
scanf("%d", &new_node->data);
```

```
new_node->next=NULL;
```

```
if(rear==NULL)
```

```
{ new = new_node }
```

```
rear=new_node;
```

```
front=rear;
```

```
else
```

```
{
```

```
rear->next=new_node;
```

```
rear=new_node;
```

```
}
```

```
}
```

```
void del()
```

```
{
```

```
if(front==NULL)
```

```
printf("queue is empty\n");
```

```
else
```

```
printf("deleted element is %d, front deleted");
```

Deleted

~~front = front + next;~~

}

}

void display()

{

struct node \*temp;

if (front == NULL)

printf ("queue is empty");

temp = front;

while (temp != NULL)

{

printf ("%d\n"), temp->data);

temp = temp->next;

}

void main()

{

int choice;

while (1)

{  
printf ("1. insert\n2. delete\n3. display\n4. exit");  
scanf ("%d", &choice);  
switch (choice)

{

case 1: insert(); break;

case 2: del(); break;

case 3: display(); break;

case 4: exit(0);

}

}

4:Exit

enter the choice

1

enter the item at rear-end

12

1:Insert\_rear

2:Delete\_front

3:Display\_list

4:Exit

enter the choice

2

item deleted at front-end is=12

1:Insert\_rear

2:Delete\_front

3:Display\_list

4:Exit

enter the choice

3

list empty cannot display items

1:Insert\_rear

2:Delete\_front

3:Display\_list

4:Exit

enter the choice

# Lab-program

## Doubly Linked List

14/12/2020

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

struct node
{
    int info;
    struct node* link;
    struct node* slink;
};

typedef struct node* NODE;

NODE getnode()
{
    NODE x;
    x = (NODE) malloc(sizeof(struct node));
    if (x == NULL)
    {
        printf("memory full");
        exit(0);
    }
    return x;
}

void freenode(NODE x)
{
    free(x);
}

NODE direct_front(int item, NODE head)
{
    NODE temp, cur;
    temp = getnode();
    temp->info = item;
    temp->link = head;
    temp->slink = NULL;
    if (head != NULL)
        head->slink = temp;
    head = temp;
}
```

```
temp->info = item  
curr->link = temp  
head->link = temp  
temp->link = head  
temp->link = curr  
curr->link = temp  
return head;
```

2) ~~insert rear (int item, NODE head)~~

```
{  
    NODE temp, curr  
    temp = getNode()  
    temp->info = item;  
    curr = head->link;  
    head->link = temp;  
    temp->link = head;  
    temp->link = curr;  
    curr->link = temp;  
    return head;
```

3) ~~delete - rec ( NODE head )~~

```
{  
    NODE curr, prev;  
    if (head->link == head)  
    {  
        cout << "empty[n]"<< endl;  
        return head;  
    }
```

```
    cur = head->link;
    prev = cur->link;
    head->link = prev;
    prev->rlink = head;
    printf("The node deleted is %d, cur->info);
```

freeNode (cur);  
return head;

}  
NODE insert\_leftpos (ent item, NODE head)

{ NODE temp, cur, prev;  
if (head->rlink == head)

printf("list empty\n");  
return head;

cur = head->rlink;

while (cur != head)

{  
if (item == cur->info) break;

cur = cur->rlink;

}

if (cur == head)

q

printf("key not found\n");

return head;

b

prev = cur->link;

printf("enters towards left of %d = ", item);

```

temp = getnode();
scanf ("%d", &temp->info);
prev->link = temp;
temp->link = prev;
curr->link = temp;
temp->link = curr;
return head;
}

NODE insert_rightpos (int item, NODE head)
{
    NODE temp, curr, next;
    if (head->link == head)
    {
        printf ("list empty\n");
        return head;
    }
    curr = head->link;
    while (curr != head)
    {
        if (item == curr->info) break;
        curr = curr->link;
    }
    if (curr == head)
    {
        printf ("key not found\n");
        return head;
    }
    next = curr->link;

```

right of  $\%d =$ , item);

```
    printf (" enter towards\n");
    temp = getnode();
    &canf = (%d, &temp->info);
    cur->glink = temp;
    temp->link = cur;
    next->link = temp;
    temp->glink = next;
    return head;
```

b  
NODE search(NODE head, int item)

d

```
NODE temp, cur;
```

```
int flag = 0;
```

```
if (head->glink == head)
```

```
printf (" list empty\n");
```

```
return head;
```

j

```
cur = head->glink;
```

```
while (cur != head)
```

d

```
if (item == cur->info)
```

```
flag = 1;
```

```
break;
```

j

```
cur = cur->link;
```

j

```
if (cur == head)
```

```
printf ("Search unsuccessful | n");
```

```
if (flag == 1)
```

```
printf ("Search successful | n");
```

```
}
```

```
NODE delete_all (int item, NODE head)
```

```
{
```

```
    NODE prev, cur, next;
```

```
    int count;
```

```
    if (head->link == head)
```

```
{
```

```
    printf ("list empty | n");
```

```
    return head;
```

```
}
```

```
count = 0;
```

```
cur = head->link;
```

```
while (cur != head)
```

```
{
```

```
    if (item == cur->info)
```

```
        cur = cur->link;
```

```
    else
```

```
{
```

```
    count++;
```

```
    prev = cur->link;
```

```
    next = cur->link;
```

```
    prev->link = next;
```

```
    next->link = prev;
```

```
    prenode (cur);
```

```
    cur = next;
```

```
}
```

```
if (count == 0)
    printf ("not found\n");
else
    printf ("found at %d position & deleted", count);
    // between head;
```

b

```
void display (NODE head)
```

    NODE temp;

```
    if (head->link == head)
```

b

```
        printf ("dq is empty\n");
        return;
```

b

```
    printf ("contents of dq\n");
    temp = head->link;
```

```
    while (temp != head)
```

d

```
        printf ("%d", temp->info);
```

```
        temp = temp->link;
```

b

```
    printf ("\n");
```

b

```
void main ()
```

{

```
    NODE head, last;
```

```
    int item, choice;
```

```
head = get_node()
head->link = head;
head->link = head;
for(ii)
```

```
i
ps->lncl.insert front[n2]; insert rear[n3]; update front[n4];
delete node[n5] insert left of key element[n6];
insert right of key element[n7]; search[n8]; delete
replacing elements[n9]; display[n10]; exit[n11];
print("enter the choice[n1]");
scanf("%d", &choice);
switch(choice)
```

```
j
case 1: printf("enter the item at front end[n1]");
scanf("%d", &item);
last = dinsert_front(item, head); break;
case 2: printf("enter the item at rear end[n1]");
scanf("%d", &item);
last = dinsert_rear(item, head);
break;
case 3: last = ddelete_front(head); break;
case 4: last = ddelete_rear(head); break;
case 5: printf("enter the key element[n1]");
scanf("%d", &item);
last = insert_leftpos(item, head); break;
case 6: printf("enter the key element[n1]");
scanf("%d", &item);
last = insert_rightpos(item, head);
break;
```

```
case 7: printf ("enter the search element/n");
    scanf ("%d", &item);
    Search (head, item);
    break;
case 8: printf ("enter the element to be deleted go dupli-
    cate/n");
    scanf ("%d", &item);
    last = delete_all_key (item, head);
Case 9: display (head);
    break;
default: exit (0);
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:insert left of key element
6:insert right of key element
7:search
8:delete repeating occurrences
9:display
10:exit
Enter the choice
1
enter the item at front end
10

1:insert front
2:insert rear
3:delete front
4:delete rear
5:insert left of key element
6:insert right of key element
7:search
8:delete repeating occurrences
9:display
10:exit
Enter the choice
```

```
enter the item at rear end
20

1:insert front
2:insert rear
3:delete front
4:delete rear
5:insert left of key element
6:insert right of key element
7:search
8:delete repeating occurrences
9:display
10:exit
Enter the choice
3
the node deleted is 10
1:insert front
2:insert rear
3:delete front
4:delete rear
5:insert left of key element
6:insert right of key element
7:search
8:delete repeating occurrences
9:display
10:exit
Enter the choice
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:insert left of key element
6:insert right of key element
7:search
8:delete repeating occurrences
9:display
10:exit
Enter the choice
7
enter the key element
15
key not found
```

1. Insert item  
2. Insert item  
3. Insert left of key element  
4. Insert right of key element  
5. Search  
6. Delete repeating elements  
7. Display  
8. Count  
9. Enter the choice

1. Insert item  
2. Insert item  
3. Insert item  
4. Delete item  
5. Insert left of key element  
6. Insert right of key element  
7. Search  
8. Delete repeating elements  
9. Display  
10. Count  
11. Enter the choice

1. Insert item  
2. Insert item  
3. Insert item  
4. Delete item  
5. Insert left of key element  
6. Insert right of key element  
7. Search  
8. Delete repeating elements  
9. Display  
10. Count  
11. Enter the choice  
12. Insert same right of left

1. Insert item  
2. Insert item  
3. Insert item  
4. Delete item  
5. Insert left of key element  
6. Insert right of key element  
7. Search  
8. Delete repeating elements  
9. Display  
10. Count  
11. Enter the choice

1. Insert item  
2. Insert item  
3. Insert item  
4. Delete item  
5. Insert left of key element  
6. Insert right of key element  
7. Search  
8. Delete repeating elements  
9. Display  
10. Count  
11. Enter the choice  
12. Insert same right of left

1. Insert item  
2. Insert item  
3. Insert item  
4. Delete item  
5. Insert left of key element  
6. Insert right of key element

## 10. BINARY SEARCH TREE

```
#include <stdlib.h>
#include <stdio.h>
struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};

typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x = (NODE) malloc (sizeof (struct node));
    if (x == NULL)
        printf ("memory full\n");
    exit(0);
    return x;
}

void freenode (NODE x)
{
    free(x);
}

NODE insert (NODE root,
{
    NODE temp, cur, prev;
    temp = getnode();
```

```

temp->link = NULL;
temp->link = NULL;
temp->info = item;           Root & R
prev = NULL;                 pre :- L|R|RN
return temp;                post :- R
prev = NULL;                cur = root; L|R|RN|R in :
cur = root;                while (cur != NULL)
{
    prev = cur;
    cur = (item < cur->info) ? cur->link
    : cur->link = temp;
    if (item < prev->info)           call
        prev->link = temp;
    else
        prev->link = temp;
    return root;
}
void display (NODE root, int i) call
{
    int j;
    if (root != NULL)
        display (root->link, i+1);      call 3 :- i+1
    for (j=0; j < i; j++)
        printf (" ");
    printf ("%d\n", root->info);
    display (root->link, i+1);      call 4
}
}

```

void preOrder (NODE root)

{  
if (root != NULL)

printf ("%d\n", root->info);

preOrder (root->llink);

preOrder (root->rlink);

}

}

void postOrder (NODE root)

{

if (root != NULL)

{

postOrder (root->llink);

postOrder (root->rlink);

printf ("%d\n", root->info);

}

}

void inOrder (NODE root)

{

if (root != NULL)

{

inOrder (root->llink);

printf ("%d\n", root->info);

inOrder (root->rlink);

}

}

Void main ()

```

int item, choice;
NODE *root = NULL;
for(;;)
{
    printf("1) 1.insert[n 2.display[n 3.preIn 4(post 5.end
    printf("enter the choice(D):");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1: printf("enter the item(n)");
                    scanf("%d", &item);
                    root = insert(root, item);
                    break;
        case 2: display (root, Ø);
                    break;
        case 3: preorder (root);
                    break;
        case 4: postorder (root);
                    break;
        case 5: inorder (root);
                    break;
        default: exit(0);
                    break;
    }
}

```

1. **Memory**  
2. **Display**  
3. **Processor**  
4. **RAM**  
5. **SSD**  
6. **GPU**  
**7. *Enter the choice:***  
A  
B  
C  
D  
E  
  
1. **Memory**  
2. **Display**  
3. **Processor**  
4. **RAM**  
5. **SSD**  
6. **GPU**  
**7. *Enter the choice:***  
A