LAB programme : 1 :

write a programm to simulate the working of stack using an array using push, pop, and display.
the programm should print message for stack overflow and stack underflow.

```
# include <stdio.h>
# include <stdlib.h>
# define stock_size 5

int top = -1;
int s[10];
int item;

void push()
    if (top == stock_size - 1) {
        printf("stock overflow\n");
        return;
    }
    else {
        printf("Enter the item to be inserted\n");
        scanf("%d", &item);

    top = top + 1;
    s[top] = item;

}
```

```
int pop(){
        if(top==-1){
            printf("Stack Empty");
            return 0;
        }
        else{
            printf("Element removed is : %d \n",
                                    s[top--]);
            return -1;
        }
}

void display(){
        int i;
        if(top==-1){
            printf("Stack is Empty");
            return;
        }
        printf("The Stack items are: \n");
        for(i=top; i>=0; i--){
            printf("%d \n", s[i]);
        }
}
```

```
void main ()
{
    int item_deleted;
    int choice;

    for (;;)
    {
    printf (" Enter \n1. PUSH \n2. POP \n3. DISPLAY \n
                    4. EXIT \n");

    scanf ("%d", &choice);

    switch (choice){
        case 1 : push ();
                 break;
        case 2 : pop ();
                 break;
        case 3 : display ();
                 break;
        default ; exit (0);
    }

}
}
```

```
3.DISPLAY
4.EXIT
Enter your choice for the operation
1
Enter The item to be inserted
54

Enter the corresponding number for the required operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter your choice for the operation
1
Enter The item to be inserted
84

Enter the corresponding number for the required operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter your choice for the operation
3
The Stack Items are:
84
54

Enter the corresponding number for the required operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter your choice for the operation
2
Element removed is : 84

Enter the corresponding number for the required operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter your choice for the operation
2
Element removed is : 54

Enter the corresponding number for the required operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter your choice for the operation
2
Stack is Empty

Enter the corresponding number for the required operation
1.PUSH
2.POP
3.DISPLAY
4.EXIT
Enter your choice for the operation
```

LAB programme : 2 :

Q. write a programme to simulate the working of stack using an array using push, pop, and display.
the programme should print message for stack overflow and stack underflow.

Soln:

```
#include <stdio.h>
#include <stdlib.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;
    }
    else{
        printf("Enter the item to be inserted\n");
        scanf("%d",&item);

    top=top+1;
    s[top]=item;
}
}
```

or

`s[++top]=item;`

* preincrement

convert → void

```
int pop(){
    if(top==-1){
        printf("Stack Empty");
        return 0;
    }
    else{
        printf("Element removed is : %d \n", s[top--]);
        return -1;
    }
}

void display(){
    int i;
    if(top==-1){
        printf("Stack is Empty");
        return 0;
    }
    printf("The Stack items are: \n");
    for(i=top; i>=0; i--){
        printf("%d \n", s[i]);
    }
}
```

* post decrement

[unnecessary]

a[2] ↑
a[1]
a[0]

```
void main ()}

    ~~int item_deleted;~~
    int choice;

    for (;;)}

printf (" Enter\n1. Push\n2. POP\n3. DISPLAY\n
                4. EXIT\n");

scanf ("%d", &choice);

switch (choice)}
    case 1 : push ();
             break;
    case 2 : pop ();
             break;
    case 3 : display ();
             break;
    default ; exit (0); //invalid input//

  }

}

}
```

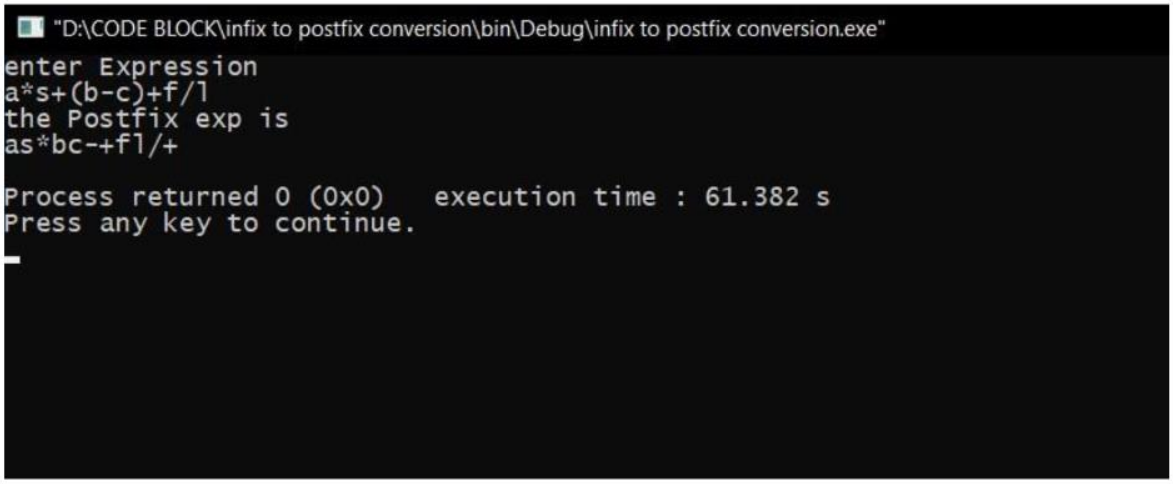
* LAB PROGRAMME - 3:

* Conversion of infix expression to postfix expression :-

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

int F (char symbol)
{
    switch (symbol) {

        case '+' :
        case '-' : return 2;
        case '*' :
        case '/' : return 4;
        case '^' :
        case '$' : return 5;
        case '(' ; return 0;
        case '#' : return -1;
        default : return 8;
    }

}
```

```
int G (char symbol) {
    switch (symbol) {
        case '+':
        case '-': return 1;
        case '*':
        case '/': return 3;
        case '^':
        case '$': return 6;
        case '(': return 9;
        case ')': return 0;
        default : return 7;
    }
}

void Conversion(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--];
    }
postfix[j] = '\0';
}

int main() {
        char infix[20];
        char postfix[20];
        pf("Enter Expression");
        sf("%s", infix);

    Conversion(infix, postfix);

    pf("The postfix Expression is %s\n");
    pf("%s", postfix);
```


"D:\CODE BLOCK\infix to postfix conversion\bin\Debug\infix to postfix conversion.exe"
enter Expression
a*s+(b-c)+f/l
the Postfix exp is
as*bc-+fl/+
Process returned 0 (0x0)   execution time : 61.382 s
Press any key to continue.


"D:\CODE BLOCK\infix to postfix conversion\bin\Debug\infix to postfix conversion.exe"
enter Expression
a+b*(c-d/f)+G
the Postfix exp is
abcdf/-*+G+
Process returned 0 (0x0)   execution time : 27.283 s
Press any key to continue.

★ LAB 6 LINEAR QUEUE IMPLEMENTATION

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

#define SIZE 5

int item, q[10];
int f=0, r=-1;
int count=0;

void Insert_front(){
    if(count == SIZE-1){
        printf("STACK is FULL");
```

```
else {
    printf("Enter Element");
    scanf("%d", &item);
    rear = (rear+1) % SIZE;
    q[rear] = item;
    count++;
}

void delete_rear() {
    if (count == 0) {
        printf("QUEUE Empty");
    }
    else {
        item = q[front];
        front = (front+1) % SIZE;
        count--;
    }
    printf("The Item Deleted is: %d", item);
}

void display() {
    if (count == 0) {
        printf("QUEUE empty");
    }
    else {
        printf("The Items are:");
        for(i=0; i<count; i++) {
            printf("%d \t", q[i]);
        }
    }
```

```
int main(){
    int choice;

    for(;;){
    printf(" Enter the corresponding choice
            1. Insert_front \n
            2. Delete_rear \n
            3. Display \n
            4. Exit \n");

    switch(choice){

        case 1 : insert_front();
                 break;
        case 2 : delete_rear();
                 break;
        case 3; display()
                 break;
        default : exit(0) ;

    }

}
```

```
 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 1
enter the element to be inserted
654

 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 1
enter the element to be inserted
6541

 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 3
ITEMS of Queue
654     6541    0
 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 2
The ITEM deleted is = 654

 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 2
The ITEM deleted is = 6541
```

LAB-5 : CIRCULAR QUEUE :

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 5

int item, front=0, rear=-1, q[SIZE],
        count = 0;

void insertrear()
{
    if (count == SIZE)
    {
        printf("queue overflow");
    }
    printf("enter the element");
    scanf("%d", &item);
    rear = (rear+1) % SIZE;
    q[rear] = item;
    count++;

void deletfront()
{
    if (count == 0)
    {
        printf("queue is Empty");
    }
    else
    {
    item = q[front];
    printf("\n item deleted is = %d\n",
                item);
```

```
        front = (front+1) % SIZE;
        count--;
    }
}

void display(){
    int i,f;
    if(count==0){
        printf("Queue is Empty");
    }
    f=front;
    printf("Items of Queue\n");
    for(i=0;i<count;i++){
        printf("%d\t",q[f]);
        f = (f+1) % SIZE;
    }
}

int main(){
    int choice;
    for(;;){
        printf("\nEnter operations\n
            1. Insert\n2. delete\n3
        4.display\n5.Exit");
    scanf("%d",&choice);
```

```
        front = (front+1) % SIZE;
        count--;
    }
}

void display(){
    int i,f;
    if(count==0){
        printf("Queue is Empty");
    }
    f=front;
    printf("Items of Queue\n");
    for(i=0;i<count;i++){
        printf("%d\t",q[f]);
        f = (f+1) % SIZE;
    }
}

int main(){
    int choice;
    for(;;){
        printf("\nEnter operations\n
            1. Insert\n2. delete\n3
        4.display\n5.Exit");
    scanf("%d",&choice);
```

```
switch (choice) {
        case 1: insertrear()
                break;
        case 2: deletefront()
                break;
        case 3: display();
                break;
        default : exit(0);
    }
}
```

Output:

```
Enter operations
1. insert
2. Delete
3. Display
→ 1
→ Enter item to be inserted
→ 45
Enter operations
1. insert
2. Delete
3. Display
→ 1.
→ Enter item
→ 20
Enter operations
1. insert          → 3.
2. Delete          → Elements are:
3. Display           45  20
```

```
 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 1
enter the element to be inserted
54810

 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 1
enter the element to be inserted
06841

 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 1
enter the element to be inserted
654

 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
4.EXIT
 3
ITEMS of Queue
54810   6841    654     0
 Enter the choice of Operation
1.INSERT REAR
2.DELETE FRONT
3.DISPLAY
```

★ LINKED LIST :

(a) Create a linked list

(b) • Insertion at first pos
• Insertion at any pos
• Insertion at last pos

(c) Display

(d) • Deletion of first
• Deletion of any specified
• Deletion of last.

(e) SORT

(f) Reverse

(g) Concatenate two list.

```
#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)
    {
        pf ("memory full");
        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_Front (NODE first) {

    NODE temp;
    temp = getnode();
    temp -> info

    if (first == NULL) {
        Pf("Empty LIST");
        return first;
    }
    temp = first;
    temp = temp -> link;

    Pf("Item deleted at front end is %d", first -> info);

    free(first);
    return temp;
}
```

```
NODE insert_Rear (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)
        Pf("EMPTY LIST");
    }
    for(temp = first ; temp != NULL; temp = temp->link)
    {
        Pf("%d", temp->info);
    }
}
```

```
NODE delete_rear (NODE first) {

    NODE cur, prev;

    if (first == NULL) {
            P[("EMPTY");
            return first;
    }

    if (first -> link == NULL) {

            P[("deleted item is %d", first->info);
        free (first);
        return NULL;
    }

    prev = NULL;
    cur = first;

    while (cur->link != NULL) {
        prev = cur;
        cur = cur->link
    }

    P[("item deleted is %d", cur->info);

    free (cur);
    prev->link = NULL;

    return first;

}
```

```
void search (int key, NODE first) {
    NODE cur;
    if (first == NULL) {
            pf(" list is EMPTY");
            return;
        }
    cur = first;
    while( cur != NULL)
      {
        if (key == cur->info) break;
        cur = cur->link;
      }
    if (cur == NULL)
      {
        pf(" UNSUCCESSFULL");
        return;
      }
    pf(" SUCCESSFULL");
}
```

```
int
count (NODE first){

    NODE cur;
    int count = 0;

    if (first == NULL){
            Pf("Length zero");
            return 0;}

    cur = first

    if(cur->link == NULL){
            Pf("one element");
            return 1;
    }

    while (cur != NULL){

        cur = cur->link;
        count++;
    }

    return count;

}
```

not necessary

```
NODE order_list (int item, NODE first) {

    NODE temp, prev, cur;
    temp = getnode();

    temp -> info = item;
    temp -> link = NULL;

    if (first == NULL) return temp;

    if (item < first -> info)
    {
        temp -> link = first;
        return temp;
    }

    prev = NULL;
    cur = first;

    while (cur != NULL && item > cur -> info)
    {
        prev = cur;
        cur = cur -> link;
    }

    prev -> link = temp;
    temp -> link = cur;

    return first;
```

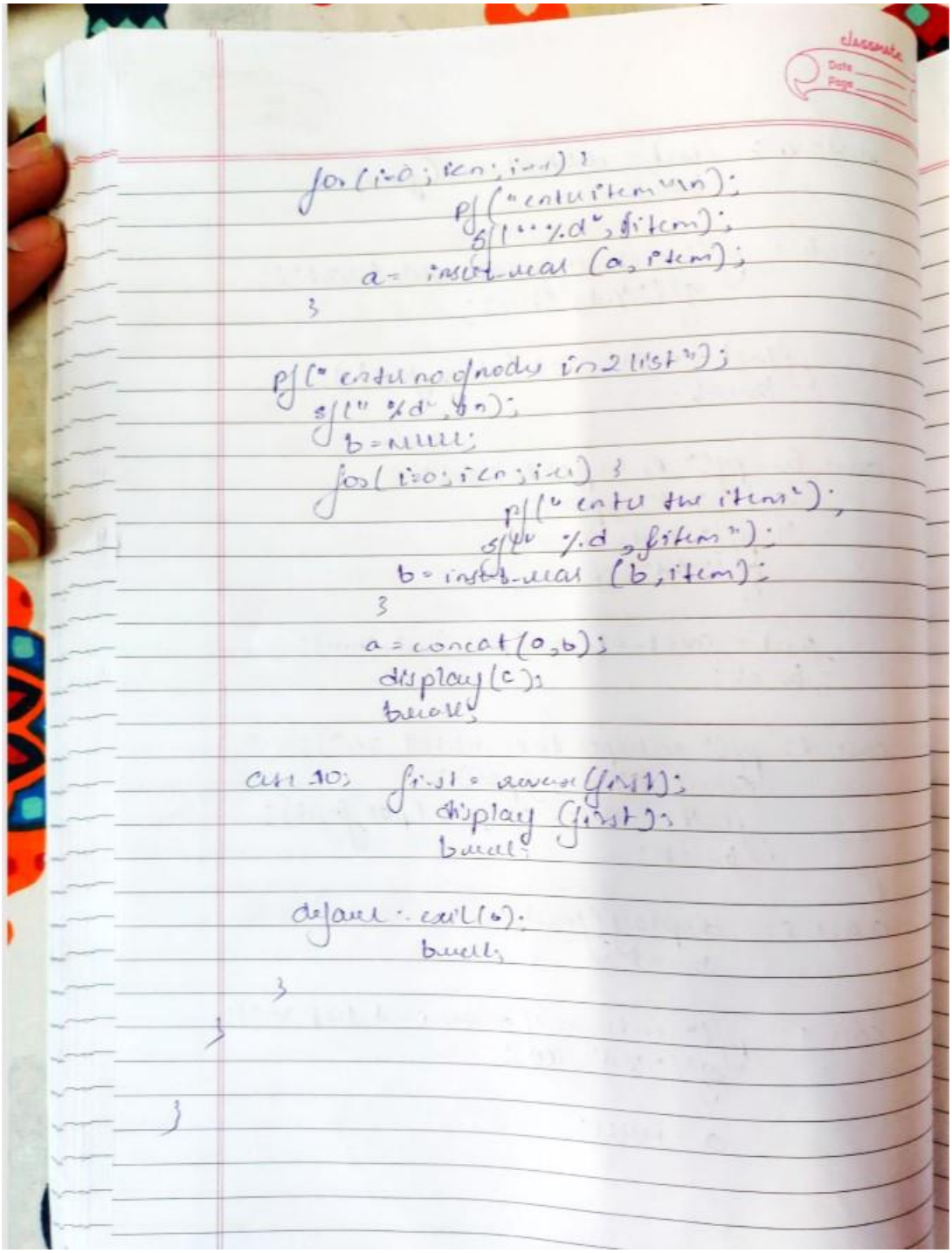
```
NODE delete_info (int key, NODE first)
{
NODE prev, cur;
if (first == NULL) {
        printf("List EMPTY");
        return NULL;
}

if (key == first->info) {
        cur = first;
        first = first->link;
        freenode(cur);
        return first;
}

prev = NULL;
cur = first;
while (cur != NULL) {
        if (key == cur->info) break;
        prev = cur;
        cur = cur->link;
}
if (cur == NULL) {
        printf("UNSUCCESFULL");
        return first;
}
```

```
prev->link = cur->link;
pf("Key deleted is %d", cur->info);
freenode(cur);
return first;
}

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;

}
```

```
int main()
{
    int item, choice, pos;
    NODE first = NULL;
    int n, i;
    NODE a, b;

for (;;)
{
    printf("\n 1. Insert_front \n 2. Delete_front \n
        3. Insert_Rear \n 4. Delete_Rear \n 5.
        Sort List \n 6. insert_at_pos \n 7.
        delete_at_pos \n 8. Display_List \n 9.
        Concate \n 10. Reverse \n");
scanf("%d", &choice);

switch(choice)
{

case 1 : pf("enter item at front");
         scanf("%d", &item);
         first = insert_front(first, item);
         break;

case 2: first = delete_front(first);
        break;

case 3: pf("enter at rear end");
        sf("%d", &item);

        first = insert_rear(first, item);
        break;
```

```
case 4 : first = delet_at_end(first);
         break;

case 5 : printf("enter item into added list");
         scanf("%d", &item);
         first = add_list(item, first);
         break;

case 6: printf("Enter pos to insert");
        scanf("%d", &pos);
        printf("enter item");
        scanf("%d", &item);
        first = insert_at_pos(pos, first, item);
        break;

case 7: printf("enter pos to be deleted \n");
        scanf("%d", &pos);
        first = delete_at_pos(pos, first);
        break;

case 8: display(first);
        break;

case 9: printf("enter no of nodes in a list \n");
        scanf("%d", &n);
        a = NULL;
```



```
for(i=0;i<n;i++){
        pf("enter item\n");
        sf("%d",&item);
    a= insert_rear(a,item);
}

pf("enter no of node in 2 list");
    sf("%d",&n);
    b=NULL;
    for(i=0;i<n;i++){
            pf("enter the item");
            sf("%d",&item");
        b= insert_rear(b,item);
    }
    a=concat(a,b);
    display(c);
    break;

case 10: first=reverse(first);
        display(first);
        break;

default: exit(0);
        break;
}
}
}
```

```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Sort_list
 6:insert_at_pos
 7:delete_at_pos
 8:Display_list
 9:Concate
 10.Reverse
enter the choice
3
enter the item at rear-end
8

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Sort_list
 6:insert_at_pos
 7:delete_at_pos
 8:Display_list
 9:Concate
 10.Reverse
enter the choice
3
enter the item at rear-end
9

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Sort_list
 6:insert_at_pos
 7:delete_at_pos
 8:Display_list
 9:Concate
 10.Reverse
enter the choice
3
enter the item at rear-end
4

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Sort_list
 6:insert_at_pos
 7:delete_at_pos
 8:Display_list
 9:Concate
 10.Reverse
enter the choice
3
enter the item at rear-end
5

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Sort_list
 6:insert_at_pos
 7:delete_at_pos
 8:Display_list
```

* CDD-8. Singly Linked list:

→ Insert front, delete Read and display
→ Count, and sorting

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

struct node
  { int info;
    struct node *link;
  }
typedef struct node *NODE;
```

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

typedef struct node *NODE;

NODE getnode() {

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


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

```
NODE insertFront (NODE first, int
    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) {
            P/("EMPTY");
            return first;
        }

    if (first -> link == NULL) {

            P/("deleted item is %d", first->info);
        free (first);
        return NULL;
    }

    prev = NULL;
    cur = first;

    while (cur -> link != NULL) {
        prev = cur;
        cur = cur -> link
    }

    P/("item deleted is %d", cur->info);

    fr (cur);
    preve -> link = NULL;

    return first;

}
```

```
void display (NODE first) {
    NODE temp;
    if (first == NULL) {
        pf("EMPTY LIST");
    }
    for (temp = first ; temp != NULL; temp
    {
        pf ("%d , temp->info);
    }
}
```

```
void search (int key, NODE first) {

NODE cur;
if (first == NULL) {
        pf(" List is EMPTY");
        return;
        }

cur = first;
while( cur != NULL)
    {
        if (key == cur->info) break;
        cur = cur->link;
    }

if (cur == NULL)
    {
        pf(" UNSUCCESSFULL");
        return;
    }
pf (" SUCCESSFULL ");
;
```

```
Count (NODE first) {

    NODE cur;
    int count = 0;

    if (first == NULL) {
                    Pf("length zero");
                    return 0; }

    cur = first

not necessary { if (cur -> link == NULL) {
                    Pf("one element");
                    return 1; }

    while (cur != NULL) {

        cur = cur -> link;
        count++;
    }

    return count;

}
```

```
order_list (int item, NODE first) {
    NODE temp, prev, cur;
    temp = getnode();
    temp->info = item;
    temp->link = NULL;
// if (first == NULL) return temp;
// if (item < first->info)
    {
        temp->link = first;
        return temp;
    }
    prev=NULL;
    cur = first;
// end of the condition
    while (cur != NULL && item > cur->info)
    {
        prev = cur;
        cur = cur->link;
    }
    prev->link = temp;
    temp->link = cur;
    return first;
}
```

done/ no elements

item less than first

```
int main(){
        int item, choice, count;
        NODE first = NULL;
        int n, i;
        NODE a, b;

    for (;;)
        {
        printf("\n 1. Insert front \n 2. delet
                rear \n 3. display \n
                4. count items \n 5.
                Sort list \n");
    Pf("enter choice");
    sf("%d", &choice);
```

```
switch (choice) {
    case 1 : print("enter the item at
                    front-end \n");
    sc("%d", &item);
    first = insert_front (first, item);
    break;

case 2 : first = delet_end (first);
    break;

case 4 : count = length (first);
    pf("length (items) in the list is %d \n",
        count);
    break;

case 5 : print("enter the item to be
                inserted in ordered list");
    sf("%d", &item);
    first = ord_list (item, first);
    break;

default: exit(0);
    break;

}

}
```

* LAB - 9 :→ DOUBLY LINKED LIST.

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

struct node {
        int info;
        struct node *rlink;
        struct node *llink;
};
typedef struct node *NODE;

NODE getnode() {
    NODE x;
    x = (NODE) malloc (sizeof (struct node));
    if (x==NULL) {
            printf("Empty");
    }
    return x;
}

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

```
NODE dinsert_front(int item, NODE head){

    NODE temp, cur;
    temp = getnode();
    temp -> info = item;
    cur = head -> rlink;
    head -> rlink = temp;
    temp -> llink = head;
    temp -> rlink = cur;
    cur -> llink = temp;
    return head;
}

NODE dinsert_rear(int item, NODE head){

    NODE temp, cur;
    temp = getnode();
    temp -> info = item;
    cur = head -> llink;
    head -> llink = temp;
    temp -> rlink = head;
    temp -> llink = cur;
    cur -> rlink = temp;
    return head;
}
```

```
NODE ddelete_front(NODE head){
    NODE cur, next;
    if(head->rlink == head){
                pf("list empty\n");
                return head; }
cur = head->rlink;
next = cur->rlink;
head->rlink = next;
next->llink = head;
pf("the node deleted is %d", cur->info);
free node(cur);
    return head;
}

NODE ddelete_rear(NODE head){
    NODE cur, prev;

    if(head->rlink == head){ pf(empty);
            return head };

cur = head->llink;
prev = cur->llink;
head->llink = prev;
prev->rlink = head;
pf("node deleted is %d, cur->info);
    free node(cur);
        return head;
}
```

```
NODE insert_before (int item, NODE head)
    NODE temp, cur, prev;

    if (head->rlink == head) {
            pf ("Empty");
            return head;
        }
    cur = head->rlink;
    while (cur != head) {
            if (item == cur->info) break;
            cur = cur->rlink;
        }

    if (cur == head) {
            pf ("key not found");
            return head }

    prev = cur->llink;
    pf ("enter element b/w %d = ", item);

    temp = getnode();
    sf ("%d", &temp->info);
    prev->rlink = temp;
    temp->llink = prev;
    cur->llink = temp;
    temp->rlink = cur;
    return head;
    }
```

```
NODE insert_after (int item, NODE head) {
    NODE temp, cur, prev;
    if (head -> rlink == head) {
            pf("empty");
            return head; }
    cur = head -> rlink;
    while (cur != head) {
            if (item == cur -> info) break;
        cur = cur -> rlink;
    }
    if (cur == head) {
            pf("key not found");
            return head;
            }
    prev = cur -> rlink;
    pf("enter element a/k %d :", item);
      temp = getnode();
    if ("%d", &temp -> info);
    cur -> rlink = temp;
      temp -> llink = cur;
      temp -> rlink = prev;
      prev -> llink = temp;
        return head;
    }
```

```
NODE Search (int item, NODE head) {
    if (head->rlink == head) {
        pf("LL empty");
        return head;
    }
    NODE cur;
    cur = head->rlink;
    while (cur != head) {
        if (item == cur->info) break;
        cur = cur->rlink;
    }
    if (cur == head) {
        pf("Element not found");
        return head;
    }
    pf("Search successful, element found");
    return head;
}
```

```
void delete_dup (NODE head) {

    NODE cur, temp, ptr, prev ;
if (head -> rlink == head) {
                printf ("List Empty ");
                return;
    }

    temp = head -> rlink;
    cur = head -> rlink;

while (temp != head) {

    prev = cur;
    cur = temp -> rlink;

    while(cur != head) {

        if (temp->info == cur->info) {

            ptr = cur -> rlink ;
            ptr -> llink = cur -> llink;
            ptr = cur -> llink
            ptr -> rlink = cur -> rlink;
            free node (cur);
        }
        cur = cur -> rlink;
    }
    temp = temp -> rlink;
}
```

```
void display (NODE head) {
    NODE temp;
    if (head->rlink == head) {
        pf("empty");
        return;
    }
    printf("contents of list \n");
    temp = head->rlink;
    while (temp != head)
    {
        pf("%d \t", temp->info);
        temp = temp->rlink;
    }
    printf("\n");
}

int main() {
    NODE head, last;
    int item, choice;
    head = getnode();
    head->rlink = head;
    head->llink = head;
    for(;;)
```

```
}
pf("1. insert_front 2. insert_rear 3. delete_front
    4. delete_rear 5. display 6. insert_
    before, 7. insert_after. 8. Search
    9. delete Duplicate");

sf("%d", &choice);

switch(choice){

case 1: pf("enter item");
        sf("%d", &item);
        last = insert_front(item, head);
        break;
case 2: pf("enter");
        sf("%d", &item);
        last = insert_rear(item, head);
        break;
case 3: last = delete_front(head);
        break;
case 4: last = delete_rear(head);
        break;
case 5: display(head);
        break;
case 6: pf("enter key");
        sf("%d", &item);
        head = insert_before(item, head);
        break;
case 7: pf("enter key");
        sf("%d", &item);
```

```
case 8: pf("Enter the element for search \n");
        gf("%d", &item);
        head = search(item, head);
        break;

case 9: del_dup(head);
        pf("List after deleting duplicate
            elements");
        display(head);
        break;

default: exit(0);
    }
  }
}
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
6

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
7

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
5
contents of LIST
7       6       9       5       8       9       5       5

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
```

```
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
4
the node deleted is 8
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
5

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
9
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
9

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
8

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
5
contents of LIST
8       9       5       5       8
```

```
1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
5

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
2
enter the item at rear end
8

1:insert front
2:insert rear
3:delete front
4:delete rear
5:display
6:Insert_before
7.Insert_after
8.Search_Element
9.Delete Duplicate Elements
enter the choice
1
enter the item at front end
5
```

* LOB-10 : * Binary Search Tree :

```
#include <stdio.h>
#include <stdlib.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");
                exit(0);
        }
        return x;
}

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

```
NODE insert (NODE root, int item)
{
    NODE temp, cur, prev;
    char direction[10];
    int i;
    temp = getnode();
    temp->info = item;
    temp->llink = NULL;
    temp->rlink = NULL;

    if (root == NULL)
        return temp;

    printf("\n give direction : ");
    scanf("%s", direction);
    prev = NULL;
    cur = root;

    for (i=0; i<strlen(direction) && cur != NULL; i++)
    {
        prev = cur;
        if (direction[i] == 'l')
            cur = cur->llink;
        else
            cur = cur->rlink;
    }
    if (cur != NULL || i != strlen(direction))
    {
        printf("insertion not possible");
        freenode(temp);
        return (root);
```

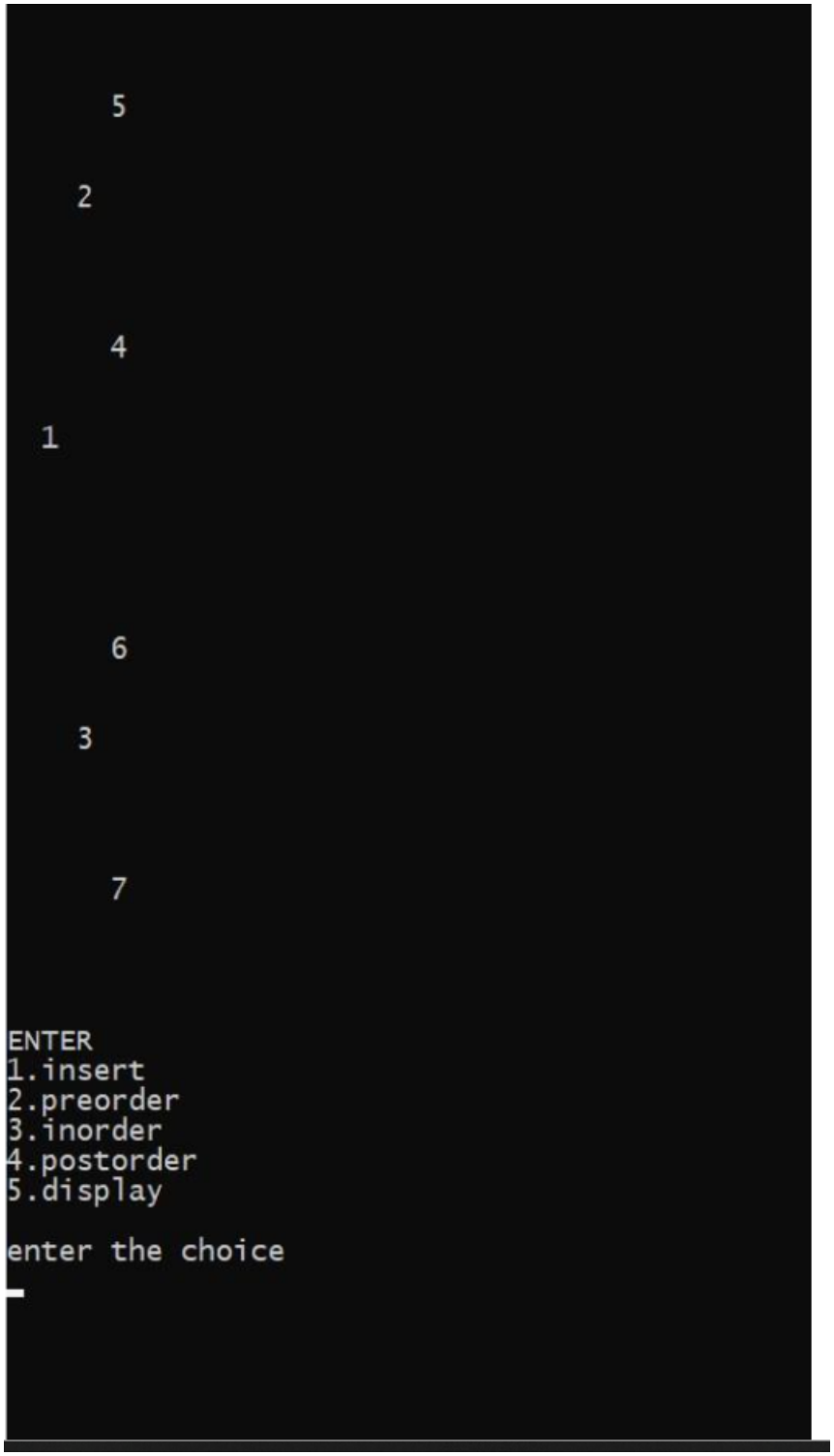
```
if (cur == NULL) {
        if (direction[i-1] == 'l')
            prev->llink = temp;
        else
            prev->rlink = temp;
    }
    return (root);
}

void preorder(NODE root) {
    if (root != NULL) {
        pf("The item is: %d\n", root->info);
        preorder(root->llink);
        preorder(root->rlink);
    }
}

void inorder(NODE root) {
    if (root != NULL) {
        inorder(root->llink);
        pf("The item is: %d\n", root->info);
        inorder(root->rlink);
    }
}
```

```
void postorder(NODE root)
{
    if (root != NULL)
    {
        postorder(root->llink);
        postorder(root->rlink);
        printf("\n The item is : %d",
                root->info);
    }
}

void display(NODE root, int i)
{
    int j;
    printf("\n\n");
    if(root != NULL)
    {
        display(root->rlink, i+1);
        for(j=1; j<=i; j++)
            pf("  ");
        printf("%d\n", root->info);
        display(root->llink, i+1);
    }
```

```
int main ()
{
    NODE root = NULL;
    int choice, i, item;

for (;;)
{
    pf("1.insert 2.preorder 3.inorder,
        4.postorder 5.display");
pf("enter choice");
sf("%d", &choice);

switch (choice)
{
  case 1: pf("enter item");
          sf("%d", &item);
          root = insert(item, root);
          break;

  case 2: if (root == NULL)
          {
            pf("tree is empty");
          }
          else
          {
            pf("given tree");
            display(root, 0);
            pf("\n preorder traversal is \n");
            preorder(root);
          }
          break;
```

```
case 3: if (root == NULL)
        {
            pf ("tree is empty");
        }
        else {
            pf ("given tree is ");
            display (root, 1);
            pf ("\n inorder traversal is \n");
            inorder (root);
        }
        break;

case 4: if (root == NULL)
        {
            pf ("tree is empty");
        }
        else {
            pf ("Given tree:");
            display (root, 1);
            pf ("\n postorder traversal is:");
            postorder (root);
        }
        break;

case : display (root, 1);
       break;

default : exit (0);

    }
}
```

```
      7


the preorder traversal is
The item is : 1
The item is : 3
The item is : 7
The item is : 6
The item is : 2
The item is : 4
The item is : 5

ENTER
1.insert
2.preorder
3.inorder
4.postorder
5.display
```


5
2
4
1
6
3
7
ENTER
1.insert
2.preorder
3.inorder
4.postorder
5.display
enter the choice

```
enter the item
6

give direction to insert:
lr


ENTER
1.insert
2.preorder
3.inorder
4.postorder
5.display

enter the choice
1

enter the item
7

give direction to insert:
ll


ENTER
1.insert
2.preorder
3.inorder
4.postorder
5.display

enter the choice
5









      5
```

```
2.preorder
3.inorder
4.postorder
5.display

enter the choice
1

enter the item
3

give direction to insert:
l


ENTER
1.insert
2.preorder
3.inorder
4.postorder
5.display

enter the choice
1

enter the item
4

give direction to insert:
rl


ENTER
1.insert
2.preorder
3.inorder
4.postorder
5.display

enter the choice
1

enter the item
5

give direction to insert:
rr
```

STACK IMPLEMENTATION USING LINKED LIST :

```
#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("Full");
        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_Front (NODE first) {

    NODE temp;
    ~~temp = getnode();~~
    ~~temp -> info~~

    if (first == NULL) {
                printf("Empty LIST");
                return first;
                }
    temp = first;
    temp = temp -> link;

    printf("Item deleted at front end is %d", first ~~temp~~->info);

    free(first);
      return temp;
}
```

```
void display (NODE first) {
    NODE temp;
    if (first == NULL) {
        Pf("EMPTY LIST");
    }
    for (temp = first ; temp != NULL; temp = temp->link)
    {
        Pf("%d", temp->info);
    }
}
```

```
int main();
    int item, choice, pos;
    NODE first = NULL;

for(;;)
{
pf("1. Insert_front \n, 2Delete_fron
    \n3. Display_list  4: Exit \n
pf("  %d", &choice);

switch (choice){
  case 1: printf("enter item");
    sf("%d", &item);
    first = insert_at (first, item)
    break;
  case 2 : first = delet_front(firs
    break;
  case 3: display(first);
    break;
  default : exit(0);
}
}
```

```
 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
2
item deleted at rear-end is 3
 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
2
item deleted at rear-end is 2
 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
2
item deleted is 1

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
2
stack is empty cannot delete

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
  ;er the choice
```

```
 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
1

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
2

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
3

 1:Insert_rear
 2:Delete_rear
 3:Display_list
 4:Exit
enter the choice
3
1
```

* QUEUE IMPLEMENTATION WITH LINKED LIST :

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

struct node {
        int node;
        struct node *link;
        };
typedef struct node * NODE;

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

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

```
NODE insert_Rear (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;
}
```

```
NODE delete_Front (NODE first) {

    NODE temp;
    temp = getnode();
    temp->info

    if (first == NULL) {
            P/("Empty LIST");
            return first;
    }
    temp = first;
    temp = temp->link;

    P/("Item deleted at front end is %d", first->info)

    free (first);
    return temp;
```

```
void display (NODE first)
{
    NODE temp;
    if (first == NULL)
    {
        pf("EMPTY LIST");
    }
    for(temp = first ; temp != NULL ; temp = temp->link)
    {
        pf("%d", temp->info);
    }
}
```

```
int main()
{
    int item, choice, pos;
    NODE first = NULL;
    for(;;)
    {
        pf("\n1. Insert-rear 2. Delete-front\n
            3. Display-list 4. Exit");
        pf("Enter:");
        sf("%d", &choice);
        switch(choice)
        {
            case 1: pf("enter item:");
                    sf("%d", &item);
                    first = insert_rear(first, item);
                    break;
            case 2: first = delete_front(first);
                    break;
            case 3: display(first);
                    break;
            default: exit(0);
        }
    }
```

```
 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
1

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
2

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
3

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
1
2
```

```
 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=1

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=2

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=3

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
list is empty cannot delete

 1:Insert_rear
 2:Delete_front
 3:Display_list
```