Lab Program - 1

8) Write a program to simulate the working of stack using ~~sta~~ array with push, pop and display. The prog should print appropriate ~~prog~~ outputs for stack underflow and overflow conditions.

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

#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;
  if(top==-1)
    { printf("Stack is empty");
      return;
    }
printf("Contents of the stack are");
for(i=0; i<=top; i++)
  { printf("%d \n", S[i]);
}}
void main()
{ int item_deleted;
  int choice;
  for(;;)
{
  printf("\n 1. Push \n 2. Pop \n 3. Display
    \n 4. Exit");
  printf("\n Enter your choice");
  switch(choice);
  scanf("%d \n", &choice);
switch(choice)
{
  case 1: printf("Enter the item to be
  inserted \n");
scanf("%d", &item);
push();
```

```
break;
case 2: printf
 item_deleted = pop();
if(item_deleted == -1)
 printf("Stack is empty");
else
printf("Item deleted is %d \n", item_deleted);
break;
case 3: display();
break;
default: exit(0);
} } }
```

```
1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
2
Stack underflow!

1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
1
Enter the item to be inserted:
1

1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
1
Enter the item to be inserted:
2

1.Push
2.Pop
```

```
3.Display
4.Exit
Enter the choice:
1
Enter the item to be inserted:
2

1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
1
Enter the item to be inserted:
3

1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
1
Enter the item to be inserted:
4

1.Push
2.Pop
3.Display
4.Exit
```

```
Enter the choice:
1
Enter the item to be inserted:
5
Stack overflow!

1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
2
Item deleted is 4

1.Push
2.Pop
3.Display
4.Exit
Enter the choice:
3
Contents of the stack:
3
2
2
1
```

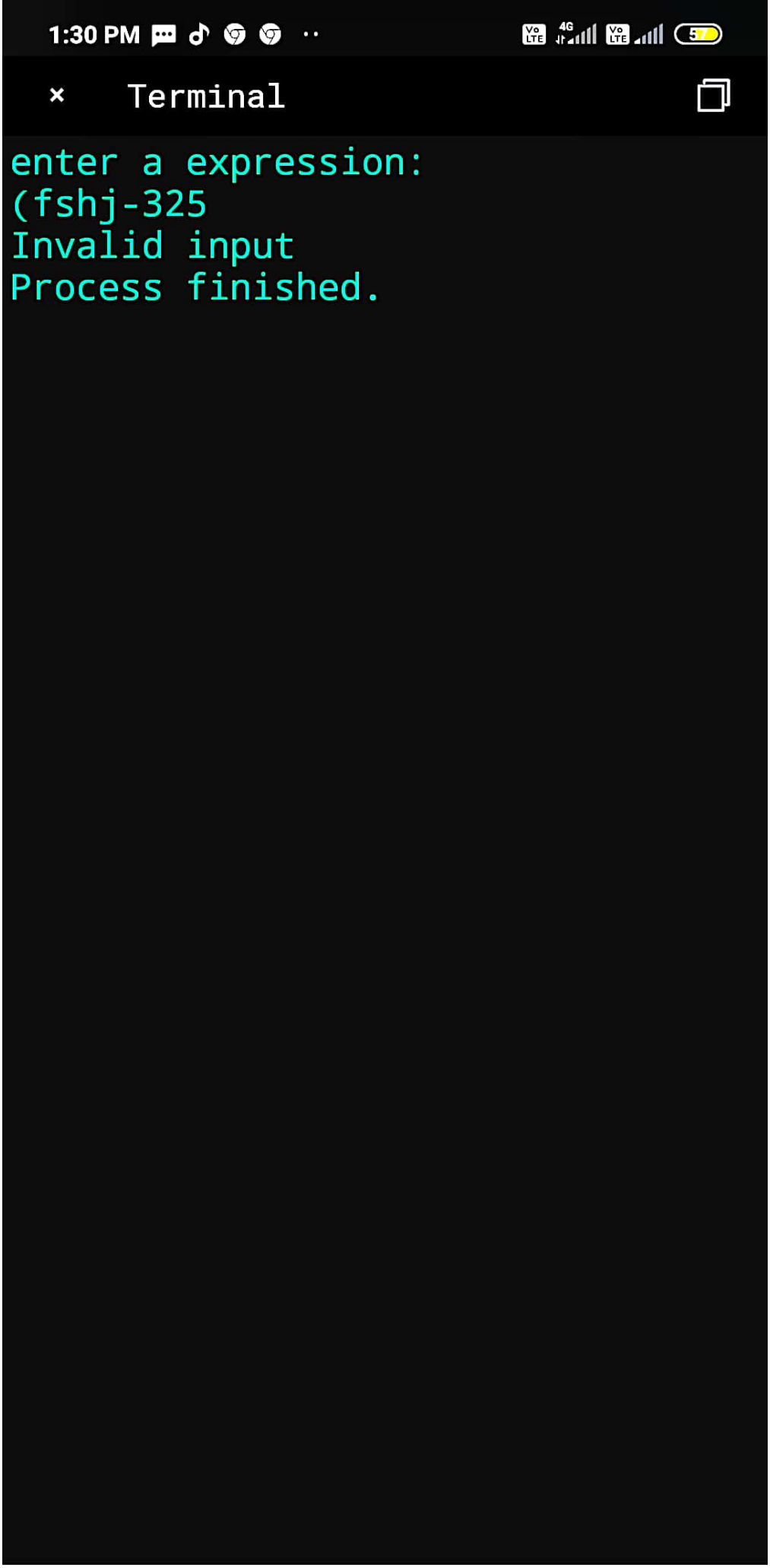
2) WAP to convert a given valid parenthesized infix arithmatic expression to postfix exp. The exp consists of single character operands and the binary operations + (plus), - (minus) * (multiply) and / (divide)

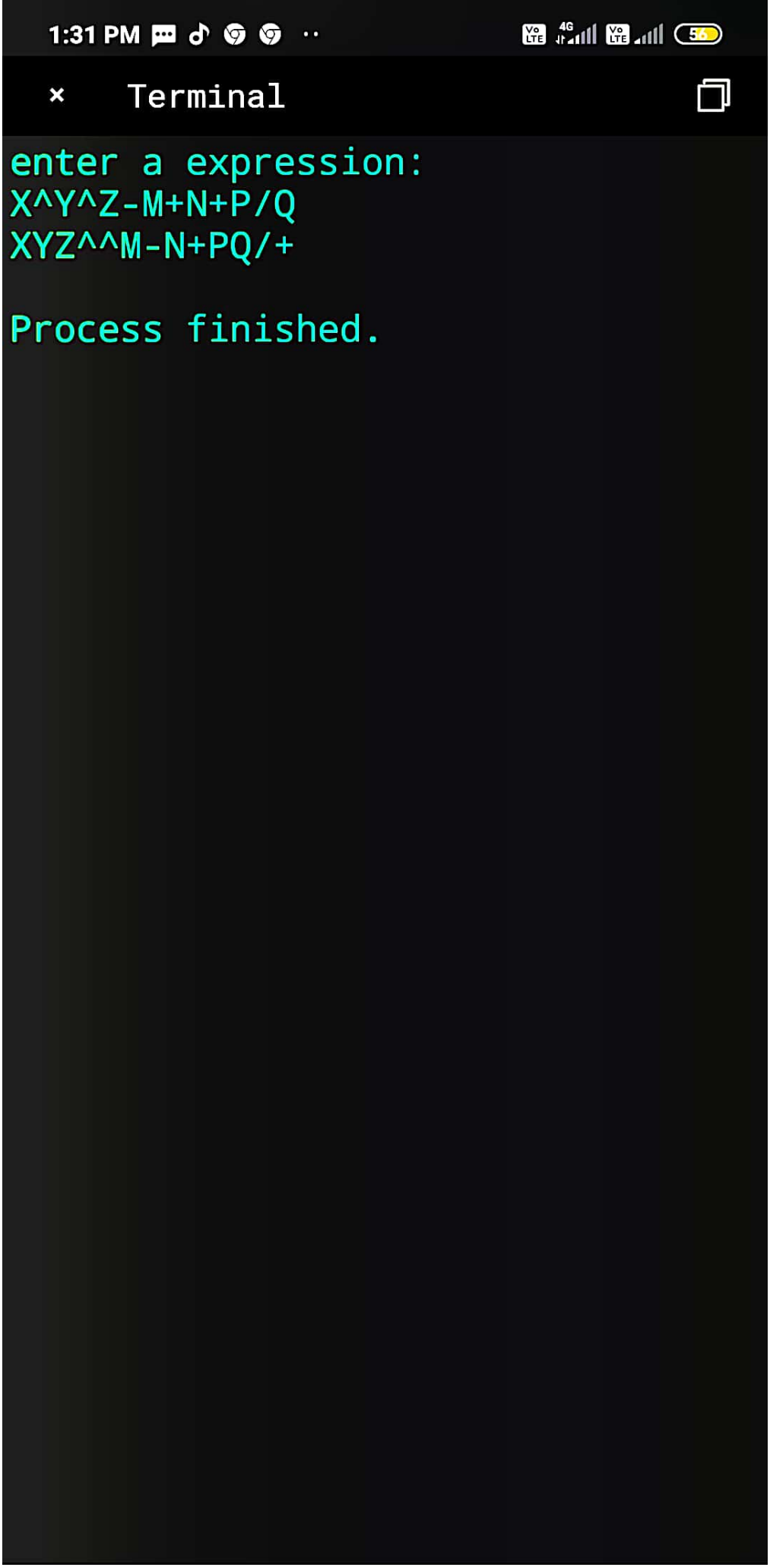
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int F (char symbol) {
    switch (symbol) {
      case '+' :
      case '-' : return 2;
      case '*' :
      case '/' : return 4;
      case '^' :
      case '$' : return 5;
      case '(' : return 0;
      case '#' : return -1;
      default : return 8;
}}

int G(char symbol) {
    switch(symbol) {
  case ):
  case (:
```

```
case '+' :
case '-' : return 1;
case '*' :
case '/' : return 3;
case '^' :
case '$' : return 6;
case '(' : return 9;
case ')' : return 0;
default : return 7;
}
}

void infix_postfix(char infix[]) {
int top, j, i;
char s[30], postfix[30];
char 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++;
    }
else
if (f(s[top]) != g(symbol)) {
    s[++top] = symbol;
```

```
else
    top--;
}
while (s[top] != '#') {
    postfix[j++] = s[top--];
}
Postfix[j] = '\0';
 puts(postfix);
}

int check(char infix[])
int i, c1=0, c2=0;
 for(i=0; i<strlen(infix); ++i)
  {
     char a;
     a = infix[];
      if(a=='(')
         c1++;
      if(a==')')
       return
       c2++;
  }
   if(c1!=c2)
     return -1;
   else
     return 0;
}
```

```
int main()
{ char exp[30];
  int c;
  printf("enter an expression");
  gets(exp);
   c = check(exp);
  if(c==-1)
     puitf("Invalid input");
  else
   infix_postfix(exp);
  return 0;
}
```





Recursion : Factorial of a no.

```
#include <stdio.h>
int factorial(int n);
int main()
{   int n;
    printf("Enter a no.");
    scanf("%d", &n);
    printf("factorial of %d is %d", n, factori
    -al(n));
      return 0;
}
    int factorial(int n) {
    { if(n>=1)
        return n* factorial(n-1);
      else
      return 1;
    }
```

GCD of 2 nos. using recursion.

```
#include <stdio.h>
int hcf(int n1, int n2)
int main()
{ int n1, n2;
  printf("Enter 2 integers");
  scanf("%d %d", &n1, &n2);
  printf("G.C.D of %d and %d is %d", n1, n2,
    hcf(n1, n2));
  return 0;
}
```

```
int hcf(int n1, int n2){
{ if(n2!=0)
    return hcf(n2, n1%n2);

  else
    return 1;
}
```

Towers of hanoi using recursion.

```
#include<stdio.h>
void towerOfHanoi(int n, char from_rod,
char to_rod, char aux_rod);
{  if(n==1)
   { printf("\n Move disc 1 from rod %c to
   to rod %c", from_rod, to_rod);
    return;
   }
towerOfHanoi(n-1, from_rod, aux_rod, to_rod
printf(" move disc %d from rod %c to
  rod %c", n, from_rod, to_rod);
towerOfHanoi(n-1, aux_rod, to rod, from_rod)
}
  int main()
  { int n=4;
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
  }
```

Bubble implimentation

Binary Search-

```
#include <stdio.h>
void binary_search (int [], int, int);
void bubble_sort( int [], int );
int main()
{ int key, size, i;
  int list[25];
  printf("Enter size of list");
  scanf(" %d ", &size);
  printf(" Enter Elements. \n");
    for( i=0; i < size ; i++)
        { scanf(" %d", &list [i]); }
  bubble_sort(list, size) ;
  printf(" Enter key to search ");
  scanf(" %d ", &key);
  binary_search ( list , 0 , size, key);
}

void bubble_sort (int list[], int size)
{ int temp, i, j ;
  for (i=0 ; i<size ; i++)
  { for ( j = i+1 ; j< size ; j++)
      { if( list [i] > list[j])
        { temp = list [i];
          list [i]= list [j];
          list [j]= temp;
        }}}}
void binary_search ( int list [], int low, int high, int
{ int mid;
  if ( low > high)
  { printf(" key not found");
```

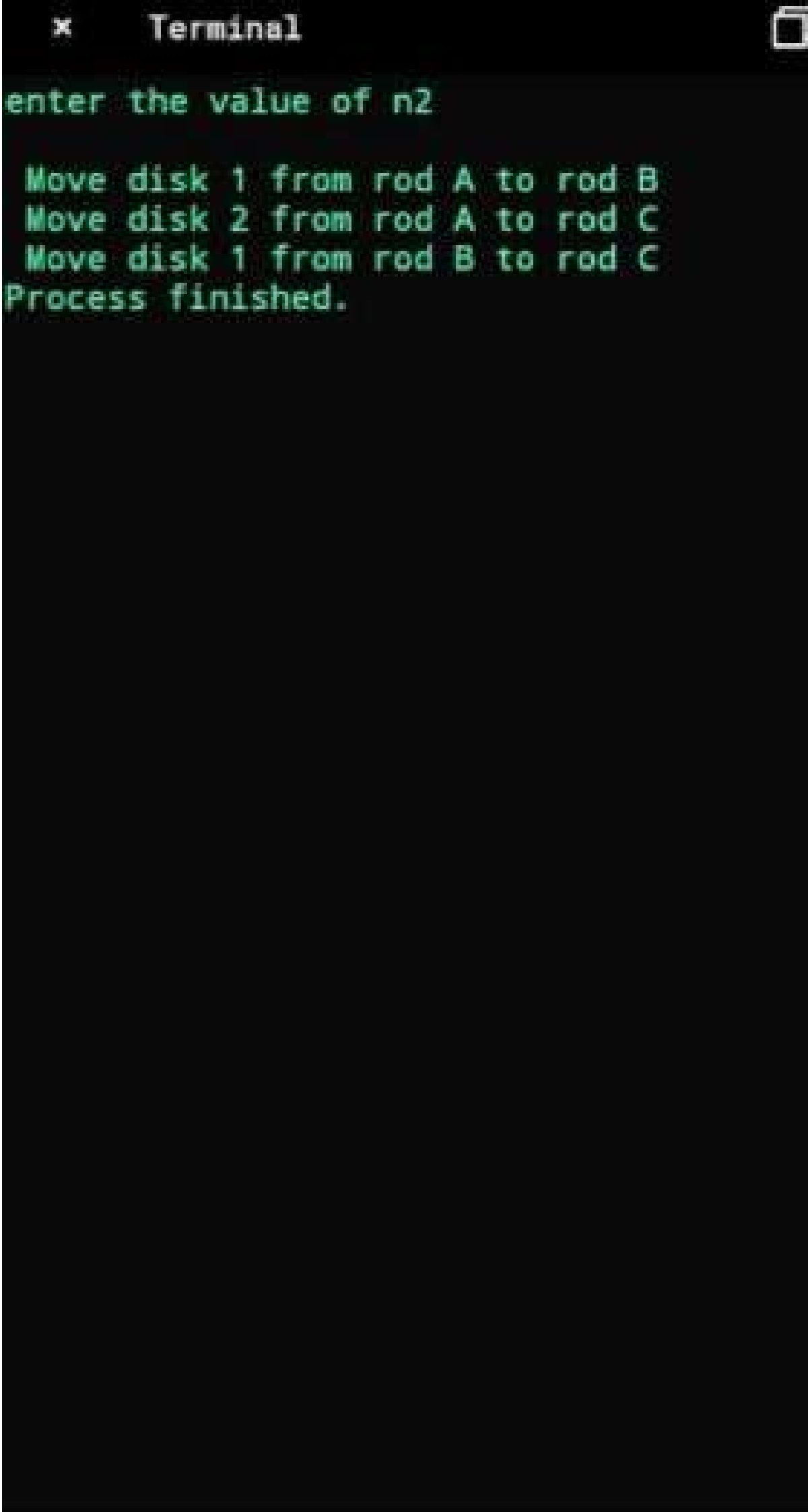
```
return ; }
mid = (low + high) / 2;
if (list[mid] == key)
{ printf(" Key found "); }
  else if (list[mid] < key)
  { binary_search(list, mid + 1, high, key);
  } }
```

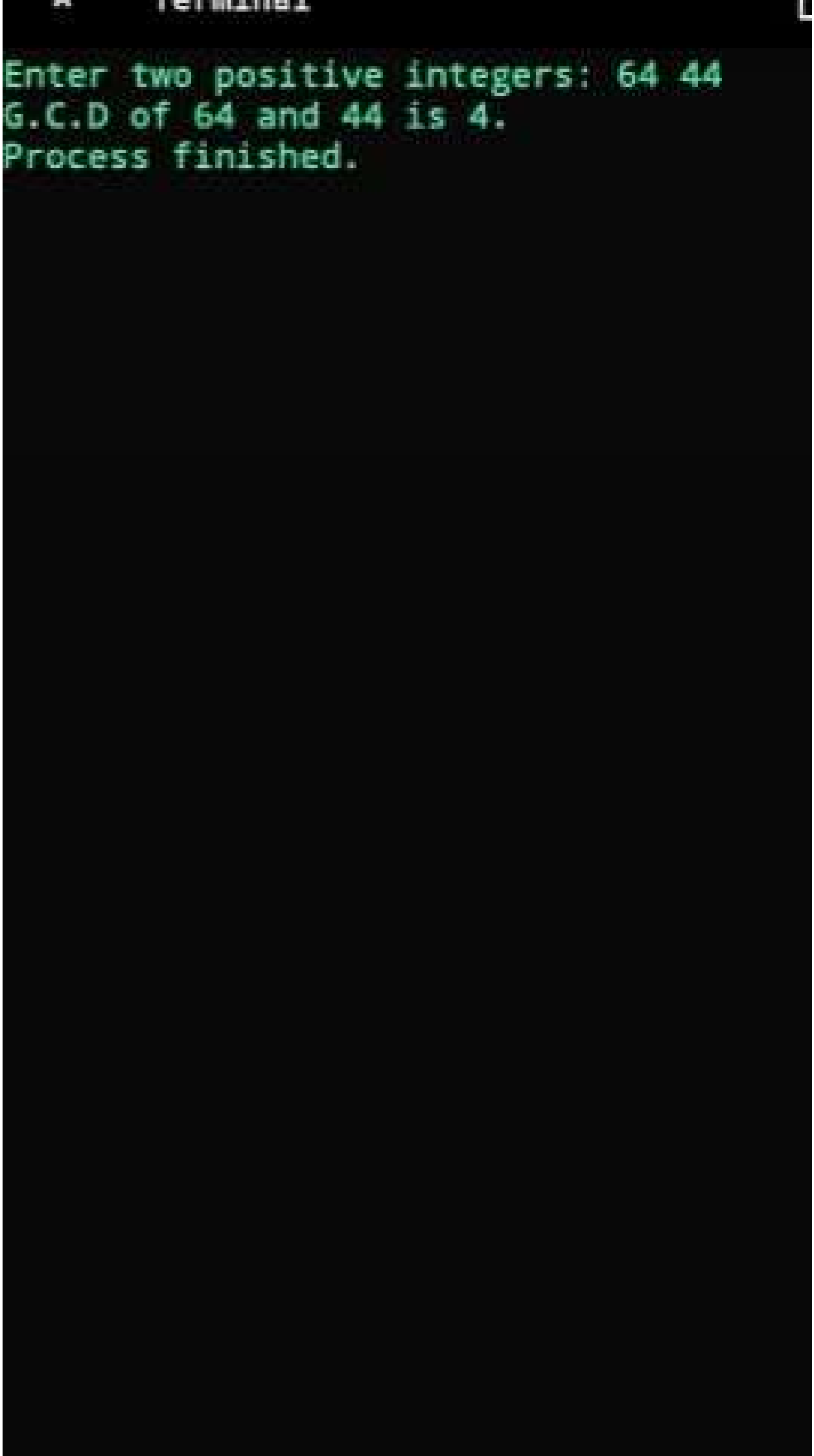

Terminal
Enter size of a list: 3
Enter elements
22 55 78
Enter key to search
55
Key found
Process finished.

Terminal

```
Enter a positive integer: 8
Factorial of 8 = 40320
Process finished.
```


Terminal
Enter n
7
fib nos are
fib[0]=0
fib[1]=1
fib[2]=1
fib[3]=2
fib[4]=3
fib[5]=5
fib[6]=8
fib[7]=13
Process finished.


Terminal
enter the value of n2
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Process finished.


Terminal
Enter two positive integers: 64 44
G.C.D of 64 and 44 is 4.
Process finished.

(1) Program for linear queue

```
#include<stdio.h>
#define CAPACITY 5
int queue[CAPACITY];
int front=0, rear=0;
void main(){
int choice, item;
for(;;){
printf(" 1: Insert 2: delete 3: traverse 4: exit ");
printf("Enter your choice");
scanf("%d", &choice);
switch(choice)
{ case 1: printf("Enter element to be Inserted");
scanf("%d", &item);
insert insert(ele); break;
case 2:
delete(); break;
case 3:
traverse();
break;
case 4: exit(0); break;
default: printf("Invalid Input"); }}}
void insert(int ele)
{ if(rear==CAPACITY-1)
printf("Queue overflow");
else {
ele = queue[rear]
rear++;
printf("Element Inserted"); }}
void delete(){
int i;
```

Spiral

```
if (front == rear)
  printf("Queue empty");
else {
 printf(" deleted item is: ", queue[front]);
  front++;
   for (i=0; i< rear; i++)
     { queue[i] = queue[i+1];
      rear--; }}}
void traverse () {
      int i;
  if (front == rear)
  printf(" queue empty");
   else
  { printf (" Elements are : ");
    for(i=0; for (i = front; i<=rear; i++)
       printf("%d", queue[i]); }}
```

2. Circular queue

```
#include <stdio.h>
#define CAPACITY 5
int cqueue[CAPACITY];
int front = -1, rear = -1, ele;
void main () {
 int ch;
for (;;) {
printf(" Enter your choice 1: Insert 2: delete 3:
 traverse 4: exit ");
scanf("%d", &ch);
switch(ch) {
case 1:
printf(" Enter element to be inserted");
scanf("%d", &ele);
```

```
        insert(ele);    break;
case 2: delete();       break;
case 3: traverse();     break;
case 4: exit(0);        break;
default: printf("Invalid input"); }}}

void insert(int ele){
  if(front==((rear+1)%CAPACITY)
    printf("cqueue is full");
  else if(front==-1 && rear==-1)
    { front=rear=0;
      cqueue[rear]=ele; }
  else if(rear==CAPACITY-1){
    rear=0;
    cqueue[rear]=ele; }
  else {
    rear++;
    cqueue[rear]=ele;}}

void delete(){
  if(front==-1 && rear==-1)
    printf("queue is empty");
  else if(front==rear){
    ele=cqueue[front];
      front=rear=-1;}
  else if(rear==CAPACITY-1){
      ele=cqueue[front];
      front=0;}
  else {
      ele=cqueue[front];
      front++;}}
```

```
void traverse() { int i;
 if (front == -1 && rear == -1)
    printf("queue empty");
 else {
   for (i=0; i<=
    for (i= front ; i <=rear ; i++)
       printf("%d /n ",equeue[i]); }}
```

De-queues

```
#include <stdio.h>
#define CAPACITY 5
int front=0, rear=-1, q[CAPACITY], i=0, item;
  void main() {
  int ch;
 for (;;) {
printf(" Enter your choice \n 1. insertfront \n 2.
 Insertrear \n 3. deletefront \n 4. deleterear \n
5. traverse 6. exit ");
scanf("%d", &ch);
 switch(ch) {
case 1: { printf(" Enter item to be inserted ");
          scanf("%d", &item);
          insertfront(item);
          break; }
case 2: { printf(" Enter item to be inserted ");
          scanf("%d", &item);
          insertrear(item);
          break;() }
case 3: deletefront();
          break;
case 4: deleterear(); break;
```

```
case 5: traverse(); break;
case 6: exit(0); break;
default: printf("Wrong input"); }}}

void insertrear(int item) {
 if(rear == CAPACITY-1)
   printf("queue overflow");
 else {
   rear++;
   q[rear] = item;  printf("Item inserted"); }}

void deletefront() {
 if(rear == -1 && front == 0)
   printf("queue empty");
 else { front++;
 printf("Item deleted from front"); }}

void Insertfront(int item) {
 if(front != 0) {
     front = front-1;
     q[front] = item; }
 else if(front == 0 && rear == -1) {
     rear++;
     q[rear] = item; }
 else
 printf("Item cannot be inserted"); }

void deleterear() {
 if(front == 0 && rear == -1)
   printf("queue empty");
 else if(front > rear) {
   front = 0;     rear = -1; }}
```

Spiral

Multiple pq

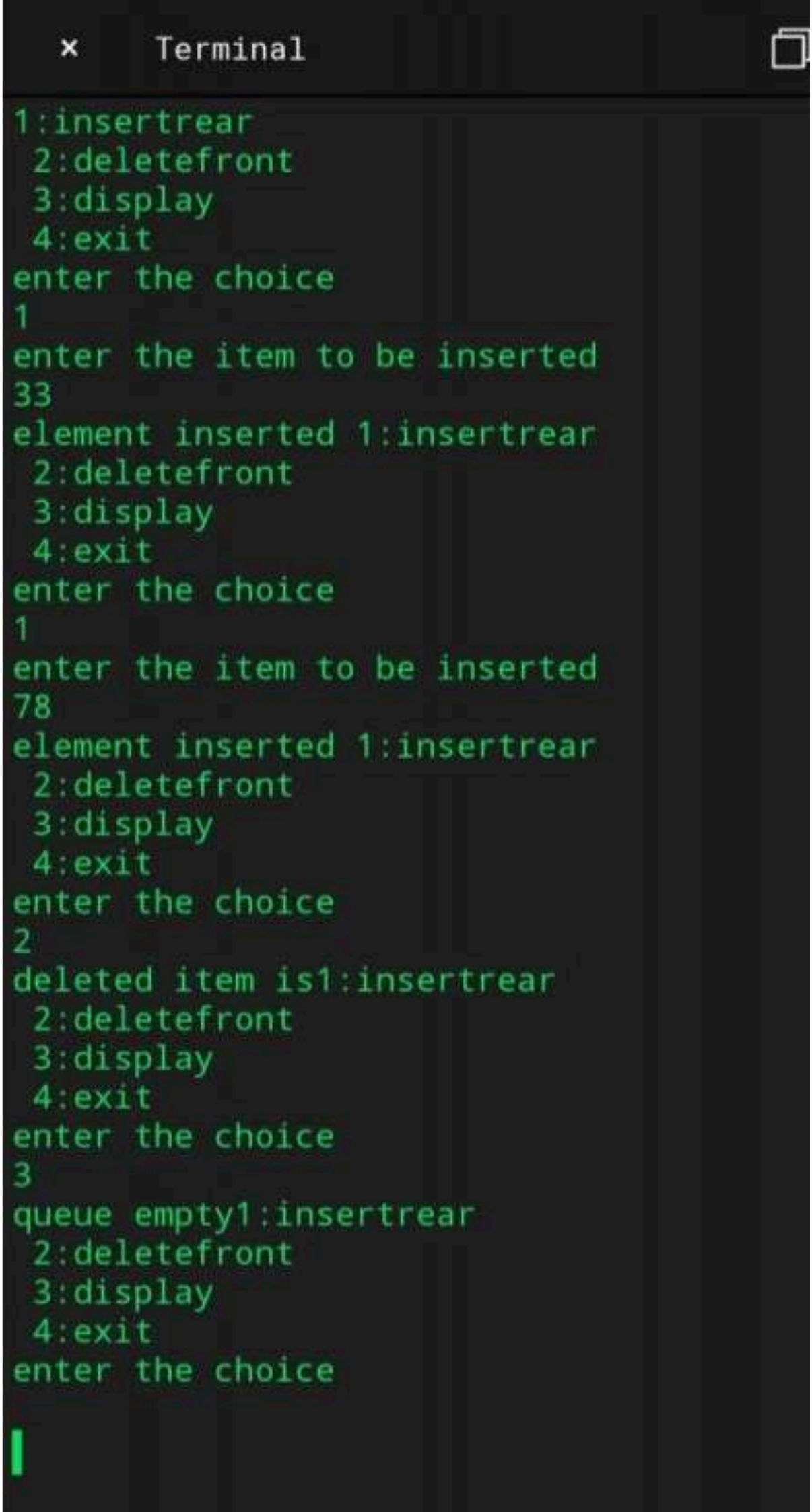
```
#include<stdio.h>
#define N 3
int queue[3],[N];
int front={0,0,0};
int rear={-1,-1,-1};
int item, pr;
void main(){
int ch;
for(;;){
printf("Enter your choice \n 1. insert \n 2. delete
 \n 3. display \n 4. exit");
scanf("%d",&ch);
switch(ch){
```

Date.....

```
case 1 : { printf("enter peiority no.");
           scanf("%d", &pr);
           if(pr > 0 && pr < 4)
             insert(pr-1)
           else
           { printf("priority shd be b/n 1-3");
             break; }}
case 2 : delete();      break;
case 3 : display();     break;
default : printf("wrong input");

void insert(int pr){
   if(rear[pr] == N-1)
     printf("queue overflow");
   else {
   printf("Enter item to be inserted");
   scanf("%d", &item);
   rear[pr]++;
   queue[pr][rear[pr]] = item; }}

void delete(){ int i;
 for(i=0; i<=3; i++){
    if(rear[i] == front[i]-1)
      printf("queue empty");
    else{
 printf("deleted item is %d of queue %d \n",
  queue[i][front[i]], i+1);
    front[i]++; }}

void display(){ int i, j;
   for(i=0; i<3; i++){
      if(rear[i] == front[i]-1)
```

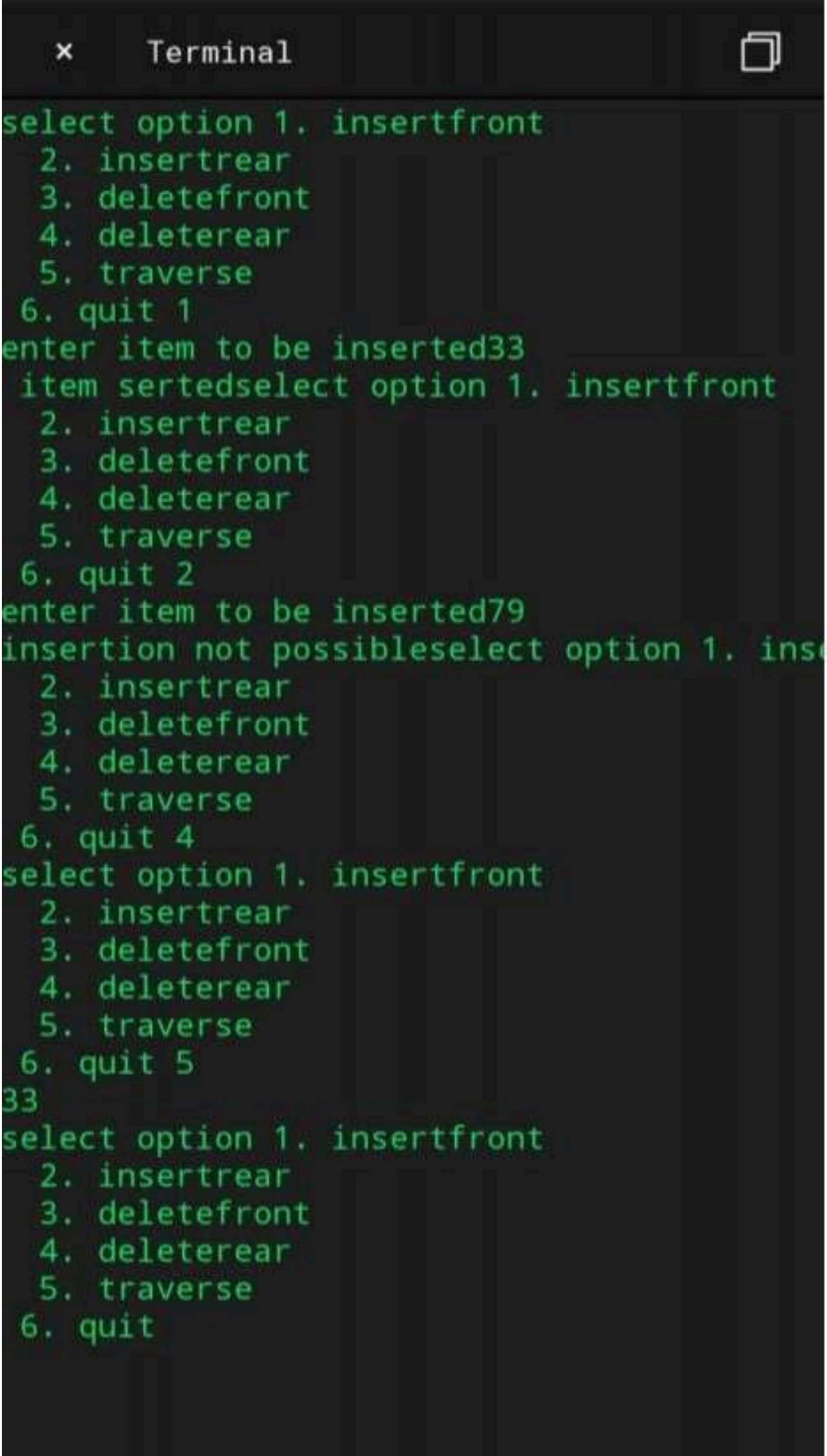
Spiral

```
printf("queue empty");
else { printf("Queue %d", i+1);
for (j=front[i]; j<=rear[i]; j++)
printf("%d \n", queue[i][j]); }}}
```


Terminal
1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
33
element inserted 1:insertrear
2:deletefront
3:display
4:exit
enter the choice
1
enter the item to be inserted
78
element inserted 1:insertrear
2:deletefront
3:display
4:exit
enter the choice
2
deleted item is1:insertrear
2:deletefront
3:display
4:exit
enter the choice
3
queue empty1:insertrear
2:deletefront
3:display
4:exit
enter the choice

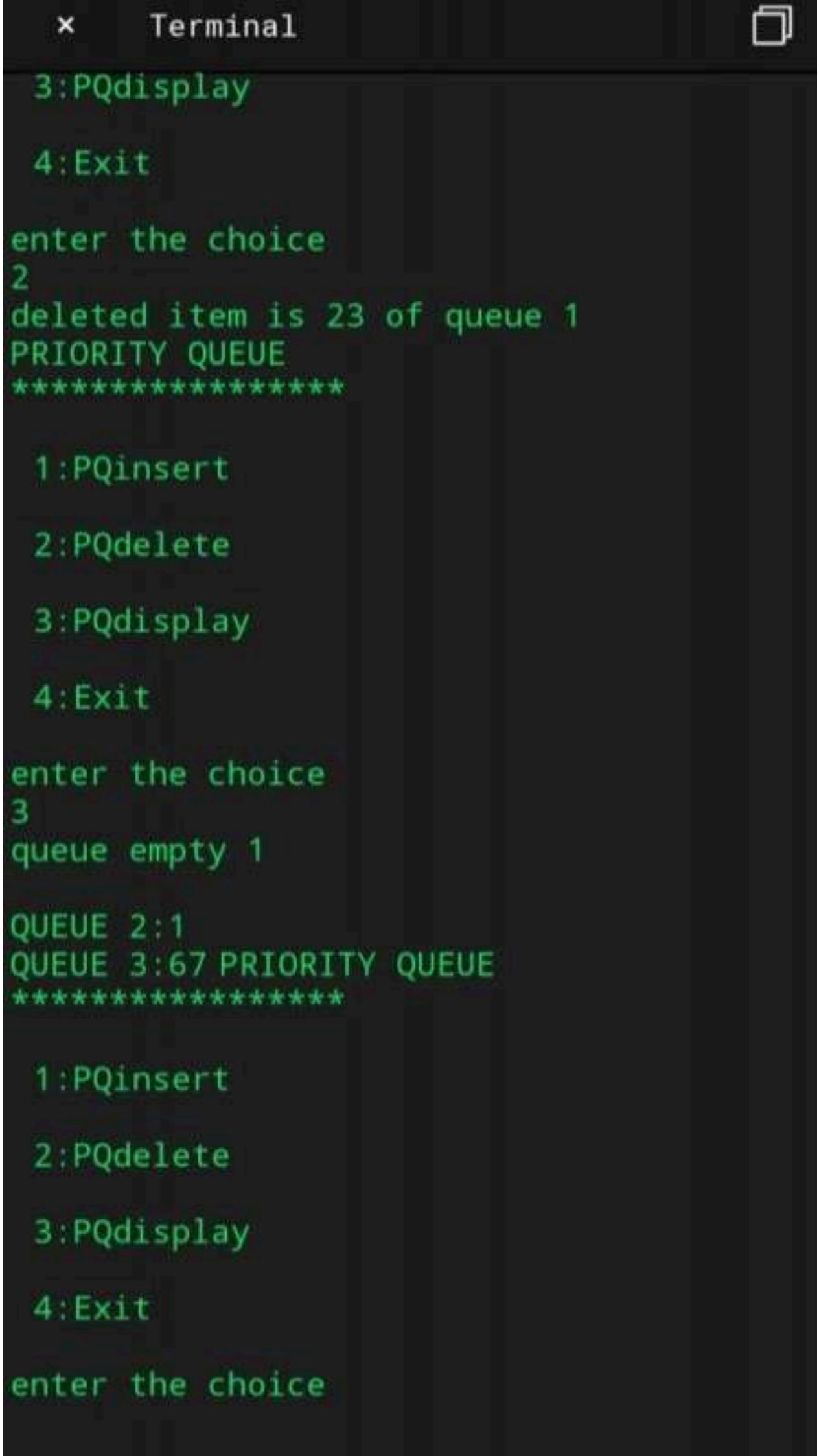
```
Terminal
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
33
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
56
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
2
enter the item
59
insertion not possible
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
```

```
Terminal
5.display
6.exit
enter choice
2
enter the item
59
insertion not possible
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
3
item deleted is 56
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
33
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
```


× Terminal
select option 1. insertfront
2. insertrear
3. deletefront
4. deleterear
5. traverse
6. quit 1
enter item to be inserted33
item sertedselect option 1. insertfront
2. insertrear
3. deletefront
4. deleterear
5. traverse
6. quit 2
enter item to be inserted79
insertion not possibleselect option 1. ins
2. insertrear
3. deletefront
4. deleterear
5. traverse
6. quit 4
select option 1. insertfront
2. insertrear
3. deletefront
4. deleterear
5. traverse
6. quit 5
33
select option 1. insertfront
2. insertrear
3. deletefront
4. deleterear
5. traverse
6. quit

```
× Terminal
PRIORITY QUEUE
******************

 1:PQinsert

 2:PQdelete

 3:PQdisplay

 4:Exit

enter the choice
1

enter the priority number
2

 enter the item
1
PRIORITY QUEUE
******************

 1:PQinsert

 2:PQdelete

 3:PQdisplay

 4:Exit

enter the choice
1

enter the priority number
1

 enter the item
```

```
 enter the item
23
PRIORITY QUEUE
******************

 1:PQinsert

 2:PQdelete

 3:PQdisplay

 4:Exit

enter the choice
1

enter the priority number
3

 enter the item
67
PRIORITY QUEUE
******************

 1:PQinsert

 2:PQdelete

 3:PQdisplay

 4:Exit

enter the choice
2
deleted item is 23 of queue 1
```


Terminal
3:PQdisplay
4:Exit
enter the choice
2
deleted item is 23 of queue 1
PRIORITY QUEUE
****
1:PQinsert
2:PQdelete
3:PQdisplay
4:Exit
enter the choice
3
queue empty 1
QUEUE 2:1
QUEUE 3:67 PRIORITY QUEUE
****
1:PQinsert
2:PQdelete
3:PQdisplay
4:Exit
enter the choice

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

struct node {
 int item;
 struct node *link;};
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 front (NODE front, 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 insert_rear ( NODE first, int item){
NODE cur, temp;
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;
if (first == NULL) {
printf(" list is empty");
return first; }
temp = first;
temp = temp -> link;
printf(" item deleted at front end");
free (first);
return temp;
}

NODE delete_rear ( NODE first) {
NODE cur, prev;
if (first == NULL) {
printf("list is empty");
return first; }
```

```
    if (first->link == NULL)
    { printf("item deleted is %d", first->info);
      free(first);
      return NULL;
    }
    prev = NULL;
    cur = first;
    while(cur->link != NULL)
      { prev = cur;
        cur = cur->link;
      }
    printf("item deleted is %d", cur->info);
    free(cur);
    Prev->link = NULL;
    return first;
}

void display(NODE first){
    NODE temp;
    if (first == NULL)
    printf("list is empty");
    for(temp = first; temp != NULL; temp =
        temp->link){
        printf("%d \n", temp->info);
    }
```

```
NODE insert_pos(int pos, NODE first){
NODE temp, cur, prev;
int count;
temp = getnode();
temp->info = item;
temp->link = NULL;
if(first == NULL && pos == 1
& return temp; }
if(first == NULL)
{ printf("Invalid position");
  return first; }
if(pos == 1)
{ temp->link = first;
  first = temp;
  return temp;
}
count = 1;
prev = NULL;
cur = first;
while(cur != NULL && count != pos)
{ prev = cur;
  cur = cur->link;
  count++;
  }
if(count == pos)
prev->link = temp;
temp->link = cur;
return first;
}
```

```
printf(" invalid position");
return first;
}

NODE delete_pos(NODE first, int pos){
    NODE cur;
    NODE prev;
    int count = 0, flag = 0;
    if (first == NULL || pos <0)
    { printf(" invalid pos");
    }
    if (pos == 1){
    cur = first;
    first = first -> link;
    freenode (cur);
    return first;
    }
    prev = NULL;
    cur = first;
    count = 1;
    while (cur != NULL)
    { if (count == pos){ flag = 1; break;}
    count ++;
    prev = cur;
    cur = cur -> link;}
if (flag == 0){
  printf(" invalid position"); return first;}
printf (" item deleted is %d", cur -> info);
prev -> link = cur -> link;
freenode (cur);
return first;
}
```

```
P void main ( ) {
  int item, pos, ch;
  NODE first = NULL;
  for (;;) {
  printf ("Enter your choice. 1. Insert-front
  \n 2 insert rear \n3 delete front \n 4
   delete rear \n 4 display \n5 insert
  at specific position \n 6 ~~insert a~~
  delete at specific position ");
  scanf ("%d", &ch);
  switch (ch) {
  case 1 : printf ("Enter item to be inserted");
           scanf("%d", &item);
           first = insert-front (first, item);
           break;
  case 2 : printf("Enter item to be inserted");
           scanf("%d", &item);
           first = insert-rear (first, item);
           break;

case 3 :   first = delete-front ( first);
           break;
case 4 :  first = delete-rear (first);

case 5 :  display ( first);
          break;
```

```
case 6: printf("Enter item to be inserted");
        scanf("%d", &item);
        printf("Enter position");
        scanf("%d", &pos);
        first = insert_pos(first, item, pos);
        break;
case 7: printf("Enter ~~item~~ position to be deleted");
        scanf("%d", &pos);
        first = delete_pos(first, pos);
}}}
```

Output

```
1:Insert_front
 2:Insert_rear
3:Insert_AtSpecfiedLocation
4:Delete_front
5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>1
enter the item at front-end
>>>22

1:Insert_front
 2:Insert_rear
3:Insert_AtSpecfiedLocation
4:Delete_front
5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>2
enter the item at front-end
>>>5

1:Insert_front
 2:Insert_rear
3:Insert_AtSpecfiedLocation
4:Delete_front
5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice

enter the item at front-end
```

Output

```
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>1
enter the item at front-end
>>>67

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>2
enter the item at rear-end
>>>7

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>2
enter the item at rear-end
>>>45

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
```

2:32 PM
03-Jan-21

Output

```
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>7
67
8
22
7
56

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>3
enter the item to be inserted at loaction
>>>8
enter the position :    >>>4

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice

67
```

Output

```
enter the position :    >>>4

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>7
67
8
22
5
7
56

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>6
enter the position :    >>>1
item deleted is 22

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
```

2:33 PM

Output

```
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>6
enter the position :    >>>3
item deleted is 22

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>7
67
8
5
7
56

 1:Insert_front
  2:Insert_rear
 3:Insert_AtSpecfiedLocation
 4:Delete_front
 5:Delete_rear
6:Delete_AtSpecfiedLocation
7:Display_list
8:Exit
enter the choice
>>>
```

2:33 PM
03-Jan-21

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main(){
NODE first=NULL, a, b;
int item, ch, n,
for(;;){
printf("Enter your choice.\n 1. insert front
\n2 insert rear \n 3 delete front \n4
delete rear \n5 display \n6 sort \n7
Search reverse \n8 concat);
scanf("%d",&ch);
switch(ch){
case 1: printf("Enter item to be inserted");
        scanf("%d",&item);
        first=insert_front(first,item);
        break;
case 2: printf("Enter item to be inserted");
        scanf("%d",&item);
        first=insert_rear(first,item);
        break;

case 3: first=delete_front(first);
        break;
case 4: first=delete_rear(first);

case 5: display(first);
        break;
```

```
case 6: first = sort(first);
        break;
case 7: first = reverse(first);
        break;
case 8: printf("Enter 1st string");
        scan
        printf("Enter no. of nodes in 1st string");
        scanf("%d", &n);
        for(i=0;i<n;i++){
        printf("Enter the item");
        scanf("%d", &item);
      a = insert_rear(a, item); }
    printf("Enter no. of nodes in 2nd string");
     scanf("%d", &n);
     for(i=0; i<n; i++){
      printf("Enter item");
      scanf("%d", &item);

      b = insert_rear(b, item);

        b }

    a = concat(a, b);
      display(a);
      break;

}}}
```

```
struct node {
 int item;
 struct node *link;};
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_front (NODE front, int pos){
  NODE temp;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL)
  { return temp; }
  temp -> link = first;
  first = temp;
  return first;
  }
```

```
NODE insert_rear ( NODE first, int item){
NODE cur, temp;
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;
if (first == NULL) {
printf(" list is empty");
  return first; }
temp = first;
temp = temp->link;
printf(" item deleted at front end");
free(first);
return temp;
}

NODE delete_rear ( NODE first) {
NODE cur, prev;
if (first == NULL) {
  printf("list is empty");
return first; }
```

```
if (first->link == NULL)
{ printf("item deleted is %d", first->info);
  free(first);
  return NULL;
}
prev = NULL;
cur = first;
while(cur->link != NULL)
 { prev = cur;
   cur = cur->link;
 }
printf("item deleted is %d", cur->info);
free(cur);
prev->link = NULL;
return first;
}

void display(NODE first){
 NODE temp;
 if (first == NULL)
 printf("list is empty");
 for(temp = first; temp != NULL; temp =
   temp->link){
    printf("%d \n", temp->info);
 }
```

```
NODE sort(NODE first){
  NODE cur, temp;
  if (first == NULL)
    return NULL;
  cur = first;
  while(cur != NULL){
   if (temp -> info < cur -> info){
      int info = cur -> info;
      cur -> info = temp -> info; info = 0;}
      temp info = i
      temp = temp -> link;
      }
      cur = cur -> link; }
    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;
 }

NODE concat (NODE first, NODE second){
```

```
{NODE cur;
if (first == NULL)
return second;
if (second == NULL)
return first;
cur = first;
~~cur -> f~~
while (cur != NULL)
  cur = cur -> link;
cur -> link = second;
return first; }
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main(){
NODE first=NULL, a, b;
int item, ch, n,
for(;;){
printf("Enter your choice.\n 1. insert front
\n2 insert rear \n 3 delete front \n4
delete rear \n5 display \n6 sort \n7
Search reverse \n8 concat);
scanf("%d",&ch);
switch(ch){
case 1: printf("Enter item to be inserted");
        scanf("%d",&item);
        first=insert_front(first,item);
        break;
case 2: printf("Enter item to be inserted");
        scanf("%d",&item);
        first=insert_rear(first,item);
        break;

case 3: first=delete_front(first);
        break;
case 4: first=delete_rear(first);

case 5: display(first);
        break;
```

```
case 6: first = sort(first);
        break;
case 7: first = reverse(first);
        break;
case 8: printf("Enter 1st string");
        scan
        printf("Enter no. of nodes in 1st string");
        scanf("%d", &n);
        for(i=0;i<n;i++){
        printf("Enter the item");
        scanf("%d", &item);
      a = insert_rear(a, item); }
    printf("Enter no. of nodes in 2nd string");
     scanf("%d", &n);
     for(i=0; i<n; i++){
      printf("Enter item");
      scanf("%d", &item);

      b = insert_rear(b, item);

        b }

    a = concat(a, b);
      display(a);
      break;

}}}
```

```
struct node {
 int item;
 struct node *link;};
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_front (NODE front, int pos){
  NODE temp;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
  if (first == NULL)
  { return temp; }
  temp -> link = first;
  first = temp;
  return first;
  }
```

```
NODE insert_rear ( NODE first, int item){
NODE cur, temp;
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;
if (first == NULL) {
printf(" list is empty");
  return first; }
temp = first;
temp = temp->link;
printf(" item deleted at front end");
free(first);
return temp;
}

NODE delete_rear ( NODE first) {
NODE cur, prev;
if (first == NULL) {
  printf("list is empty");
return first; }
```

```
if (first->link == NULL)
{ printf("item deleted is %d", first->info);
  free(first);
  return NULL;
}
prev = NULL;
cur = first;
while(cur->link != NULL)
 { prev = cur;
   cur = cur->link;
 }
printf("item deleted is %d", cur->info);
free(cur);
prev->link = NULL;
return first;
}

void display(NODE first){
 NODE temp;
 if (first == NULL)
 printf("list is empty");
 for(temp = first; temp != NULL; temp =
   temp->link){
    printf("%d \n", temp->info);
 }
```

```
NODE sort(NODE first){
  NODE cur, temp;
  if (first == NULL)
    return NULL;
  cur = first;
  while(cur != NULL){
   if (temp -> info < cur -> info){
      int info = cur -> info;
      cur -> info = temp -> info; info = 0;}
      temp info = i
      temp = temp -> link;
      }
      cur = cur -> link; }
    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;
 }

NODE concat (NODE first, NODE second){
```

```
{NODE cur;
if (first == NULL)
return second;
if (second == NULL)
return first;
cur = first;
~~cur -> f~~
while (cur != NULL)
  cur = cur -> link;
cur -> link = second;
return first; }
```

Output

```
1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:dis
enter the choice
>>>4
enter the no of nodes in 1:>>>5
enter the item:>>>1
enter the item:>>>2
enter the item:>>>3
enter the item:>>>4
enter the item:>>>5
enter the no of nodes in list2:>>>3
enter the item:>>>6
enter the item:>>>7
enter the item:>>>8
5
4
3
2
1
8
7
6

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:dis
enter the choice
>>>1
enter the item at front-end
>>>99

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:dis
enter the choice
>>>1
enter the item at front-end
>>>100

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:dis
enter the choice
```

```
1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
>>>1
enter the item at front-end
>>>12

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
>>>1
enter the item at front-end
>>>23

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
>>>1
enter the item at front-end
>>>45

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
>>>1
enter the item at front-end
>>>67

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
>>>1
enter the item at front-end
>>>89

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
>>>4

1:Insert_front  2:Delete_front  3:reverse_list  4:Concate       5:Sort  6:disp
enter the choice
```

Output

enter the choice
>>>5

1:Insert_front 2:Delete_front 3:reverse_list 4:Concate 5:Sort 6:disp
enter the choice
>>>6
4
6
12
23
45

1:Insert_front 2:Delete_front 3:reverse_list 4:Concate 5:Sort 6:disp
enter the choice
>>>3
45
23
12
6
4

1:Insert_front 2:Delete_front 3:reverse_list 4:Concate 5:Sort 6:disp
enter the choice
>>>2
item deleted at front-end is= 45

1:Insert_front 2:Delete_front 3:reverse_list 4:Concate 5:Sort 6:disp
enter the choice
>>>6
23
12
6
4

1:Insert_front 2:Delete_front 3:reverse_list 4:Concate 5:Sort 6:disp
enter the choice
>>>

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

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

typedef struct node *NODE;

NODE ~~getn~~ 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 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(NODE head, int item){
{ 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){
    printf("List empty");
    return head; }
cur = head->rlink;
next = cur->rlink;
head->rlink = next;
next->llink = head;
printf("node deleted is %d", cur->info);
freenode(cur);
  return head; }

NODE ddelete_rear(NODE head){
 NODE cur, prev;
 if(head->rlink == head)
   { printf("List empty"); return head; }
  cur = head->llink;
  prev = cur->llink;
  head->llink = prev;
  prev->rlink= head;
printf("Item deleted is %d", cur->info);
freenode(cur);
return head; }
```

```
void display(NODE head){
NODE temp; temp = head
if(head->rlink==head)
 printf("list empty");
for(temp=head->rlink; temp!=NULL; temp=temp->rlink)
   printf("\n %d", temp->info);
}

NODE dinsert_pos(int item, NODE head, int pos){
{ NODE temp, cur, prev;
  temp=getnode();
  temp->info=item;
  int i=1;
  cur=head->rlink;
  prev=NULL;
  while(i<pos && cur!=head){
  prev=cur;
   cur=cur->rlink;
    i++; }
  if(cur==head){ printf("return pos. not found");
     return head;}
     prev->rlink=temp;
     temp->rlink=cur;
     temp->llink=prev;
     cur->llink=temp;
     return head;
}
```

```
void main()
{
node *head = last;
int item, choice;
head = getnode();
head -> next = head;
head -> link = head;
for(;;)
{
printf("Enter your choice. 1->1. Insert front
2-> Insert rear 3-> delete front 4-> delete rear
5-> display 6-> insert left to");
scanf("%d", &choice);
switch(ch)
{
case 1: printf("Enter item to be inserted");
        scanf("item%d", &item);
        last = insert_front(head, item);
        break;
case 2: printf("Enter item to be inserted");
        scanf("%d", &item);
        last = insert_rear(head, item);
        break;
case 3: last = delete_front(head); break;
case 4: last = delete_rear(head); break;

case5: display(head); break;
case 6: printf("Enter the item and position");
        scanf("%d %d", &item, &pos);
        last = insert_pos(item, head);
        break;
}}}
```

Output

```
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
>>>1
enter the item at front end
>>>12

1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
>>>1
enter the item at front end
>>>34

1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
>>>1
enter the item at front end
>>>56

1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
>>>2
enter the item at rear end
>>>90

1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
>>>2
enter the item at rear end
>>>9

1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
>>>5
contents of dq
56      34      12      90      9
```

2:55 PM

× Output

```
..      --      --      ..
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete rear   5:display
enter the choice
contents of dq
34      22      12      90
1:insert front  2:insert rear   3:delete front  4:delete
```

2:55 PM

Binary Search Tree

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

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

typedef struct node *NODE;

NODE ~~getch~~ 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;
  temp = getnode( );
  temp→info = item;
  temp→llink = NULL;
  temp→rlink = NULL;
  if(root==NULL)
    return temp;
  prev = NULL;
  cur = root;
```

```
while (cur != NULL){
 prev = cur;
cur = (item < cur -> info) ? cur -> llink : cur -> rlink;
}
if (item < prev -> info)
  prev -> llink = temp;
else
  prev -> rlink = temp;
return root; }

void display(NODE root, int i){
 int j;
 if (root != NULL)
 { display (root -> rlink, i+1);
  for (j=0; j<i; j++)
    printf("    ");
  printf("%d", root -> temp);
  printf("    ");
   display (root -> link, i+1); }}
NODE delete (NODE root, int item){
  NODE cur, prev, parent, q, suc;
  if (root == NULL) {
     printf("Empty");
      return root; }
  parent = NULL;
  cur = root;
while (cur != NULL && item != cur -> info){
    parent = cur;
  cur = (item < cur -> info) ? cur -> llink : cur -> rlink;
  }
```

```
~~if (cur->t~~
if (cur==NULL)
{ printf("Element not found");
  return root;}
if (cur->link==NULL)
  q=cur->rlink;
void preorder(NODE root)
{ if(root!=NULL){
 printf("%d", root->info);
 preorder(root->llink);
 preorder(root->rlink);}}
void postorder(NODE root){
 { if(root!=NULL){
   postorder(root->llink);
   postorder(root->rlink);
   printf("%d", root->info);}}
void inorder(NODE root){
  if(root!=NULL){
   inorder(root->llink);
   printf("%d", root->info);
   inorder(root->rlink);}}

void main(){
 int ~~info~~, item, choice;
 NODE root=NULL;
 for(;;){
 printf("Enter your choice. \n1. insert \n2 display
 \n3 preorder \n4 postorder \n5 inorder
   \n6 delete");
 scanf("%d",&choice);
  switch(ch){
```

```
case 1 : printf("Enter the item");
         scanf("%d", &item);
         root = insert(root, item);
         break;
case 2 : display(root, 0);
         break;
case 3 : preorder(root); break;
case 4 : postorder(root); break;
case 5 : inorder(root); break;
case 6 : printf("Enter item to be deleted");
         scanf("%d", &item);
         root = delete(root, item);
         break;
}}}
```

Output

```
1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>1
enter the item
>>>22

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>1
enter the item
>>>33

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice

enter the item


1.insert
```

Output

```
1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>1
enter the item
>>>67

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>1
enter the item
>>>90

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>5
      90
    67
  33
22
  12
```

Output

```
enter the choice
>>>2
      90
    67
  33
22
  12

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>3
22
12
33
67
90

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>4
12
90
67
33
22
```

Paused

Output

```
>>>4
12
90
67
33
22

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>5
12
22
33
67
90

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>6
enter the item
>>>44

1.insert
2.display
3.pre
4.post
```

2:45 PM

Output

```
67
90

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>6
enter the item
>>>67

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>2
    90
  33
22
  12

1.insert
2.display
3.pre
4.post
5.in
6.delete
7.exit
enter the choice
>>>
```