DS LAB

NAME: Muskan Gupta

USN: 1BM19CS091

SECTION: 3B

BATCH: 2

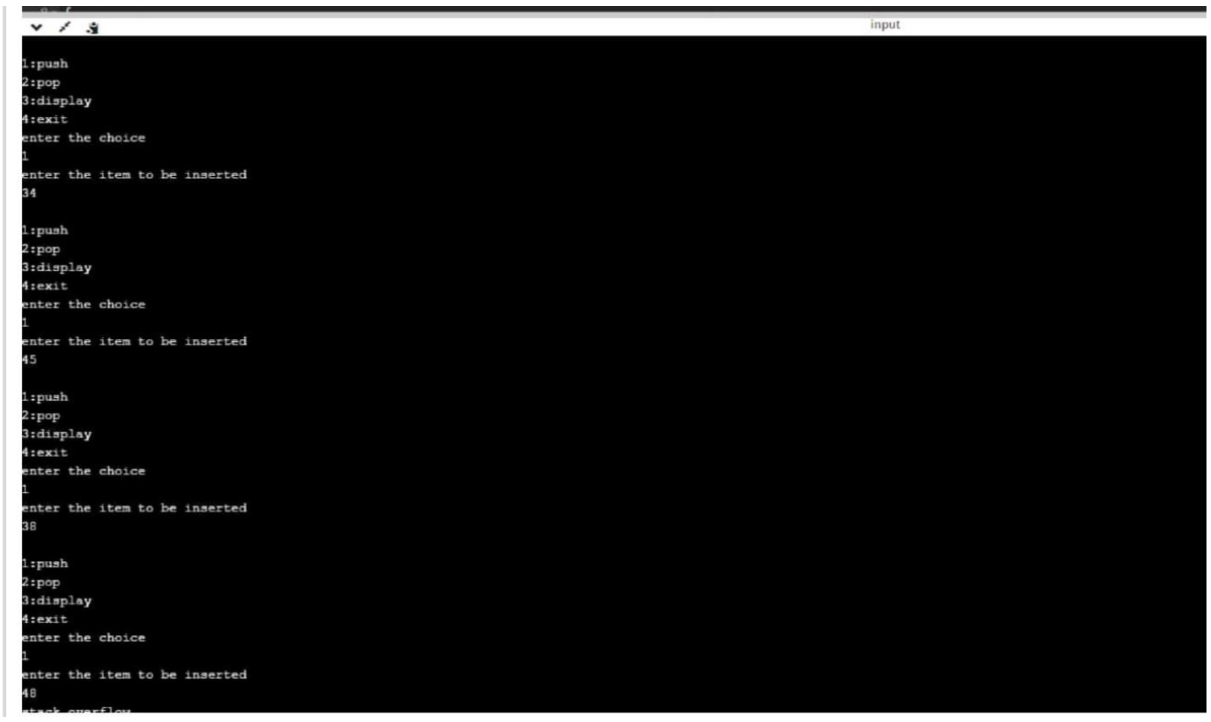
LAB PROGRAM 1

Write a program to simulate the working of stack using an array with the following : a) Push b) Pop c) Display The program should print appropriate messages for stack overflow, stack underflow

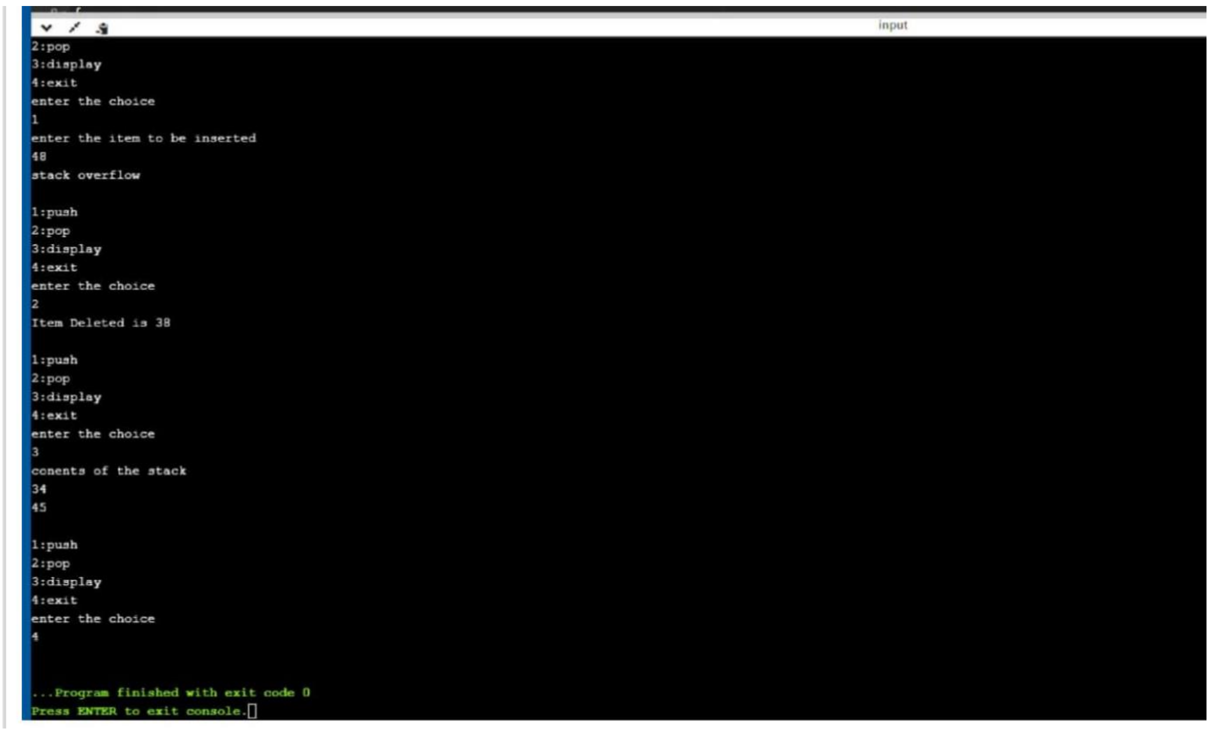
```
#include<stdio.h>

#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;

   }

   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\n");
      return;
   }
   printf("conents of the stack\n");
   for(i=0;i<=top;i++)
   {
      printf("%d\n",s[i]);
   }
}
void main()
{
   int item_deleted;
   int choice;
   for(;;)
   {
      printf("\n1:push\n2:pop\n3:display\n4:exit\n");
      printf("enter the choice\n");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         printf("enter the item to be inserted\n");
         scanf("%d",&item);
         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();
            break;
            default:exit(0);
          }
      }
  }
```



```
1:push
2:pop
3:display
4:exit
enter the choice
1
enter the item to be inserted
34

1:push
2:pop
3:display
4:exit
enter the choice
1
enter the item to be inserted
45

1:push
2:pop
3:display
4:exit
enter the choice
1
enter the item to be inserted
38

1:push
2:pop
3:display
4:exit
enter the choice
1
enter the item to be inserted
48
stack overflow
```

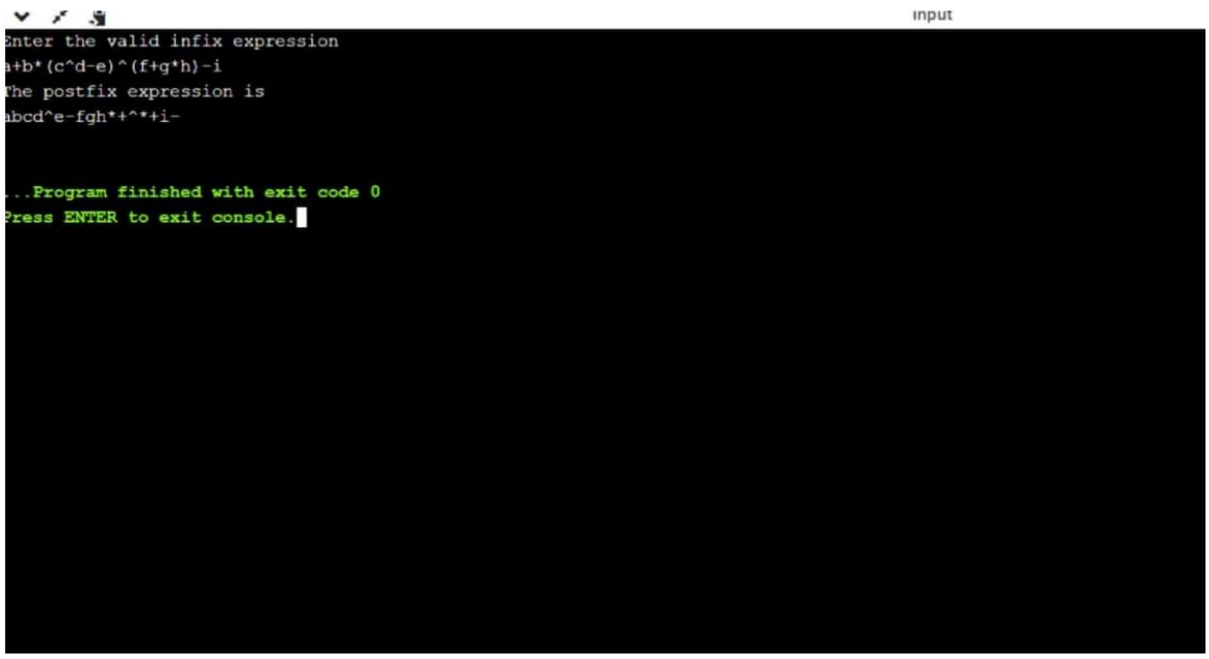


```
2:pop
3:display
4:exit
enter the choice
1
enter the item to be inserted
48
stack overflow

1:push
2:pop
3:display
4:exit
enter the choice
2
Item Deleted is 38

1:push
2:pop
3:display
4:exit
enter the choice
3
conents of the stack
34
45

1:push
2:pop
3:display
4:exit
enter the choice
4


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

LAB PROGRAM 2

WAP 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) and / (divide)

```
#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 '(':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 infix_postfix(char infix[],char postfix[])
{
        int top,j,i;
        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';
}
void main()
{
    char infix[20];
    char postfix[20];
    printf("Enter the valid infix expression\n");
    scanf("%s",infix);
    infix_postfix(infix,postfix);
    printf("The postfix expression is\n");
    printf("%s\n",postfix);
}
```



```
nter the valid infix expression
+b*(c^d-e)^(f+g*h)-i
he postfix expression is
bcd^e-fgh*+^*+i-


..Program finished with exit code 0
ress ENTER to exit console.
```

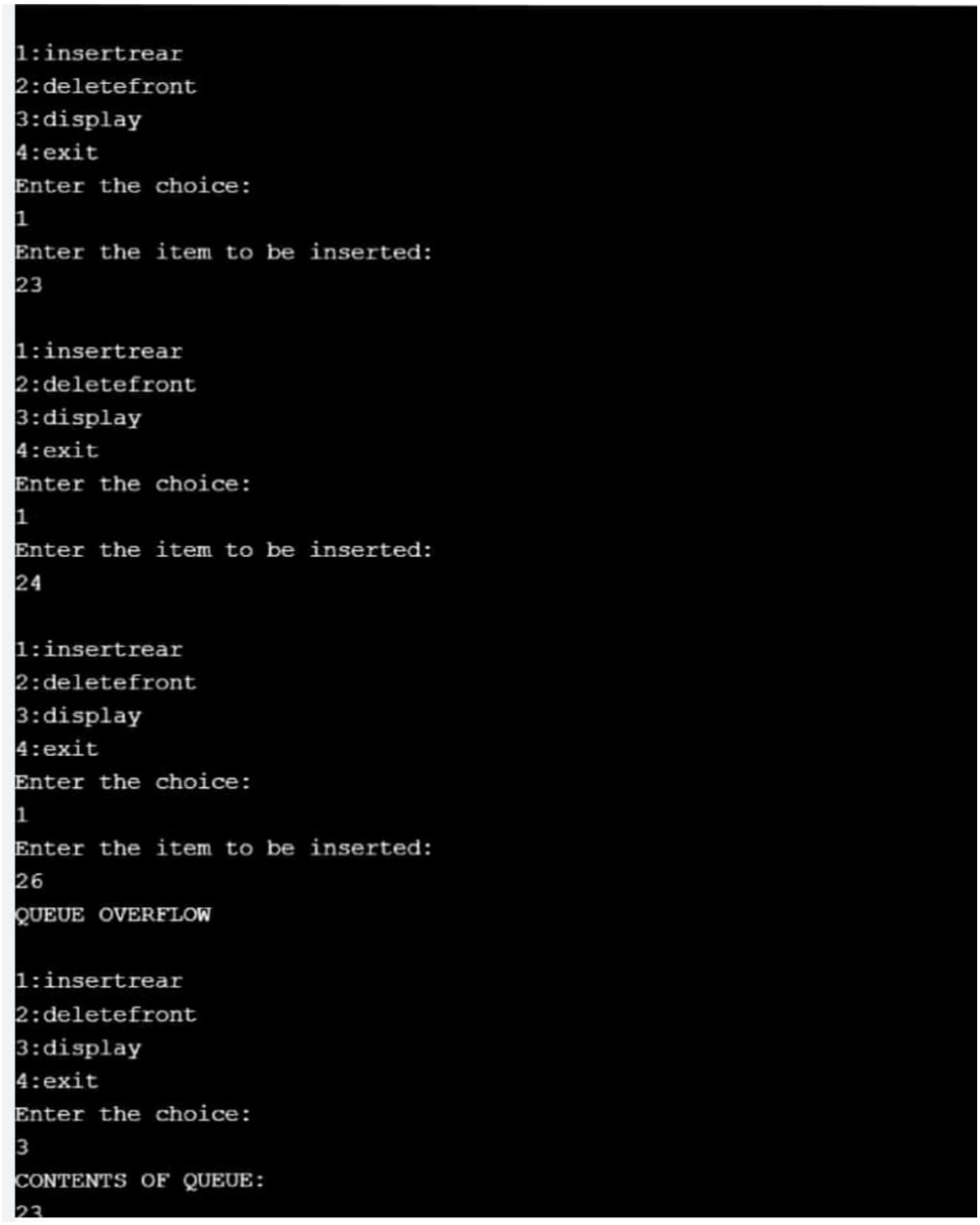
LAB PROGRAM 3

WAP to simulate the working of a queue of integers using an array. Provide the following operations a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions

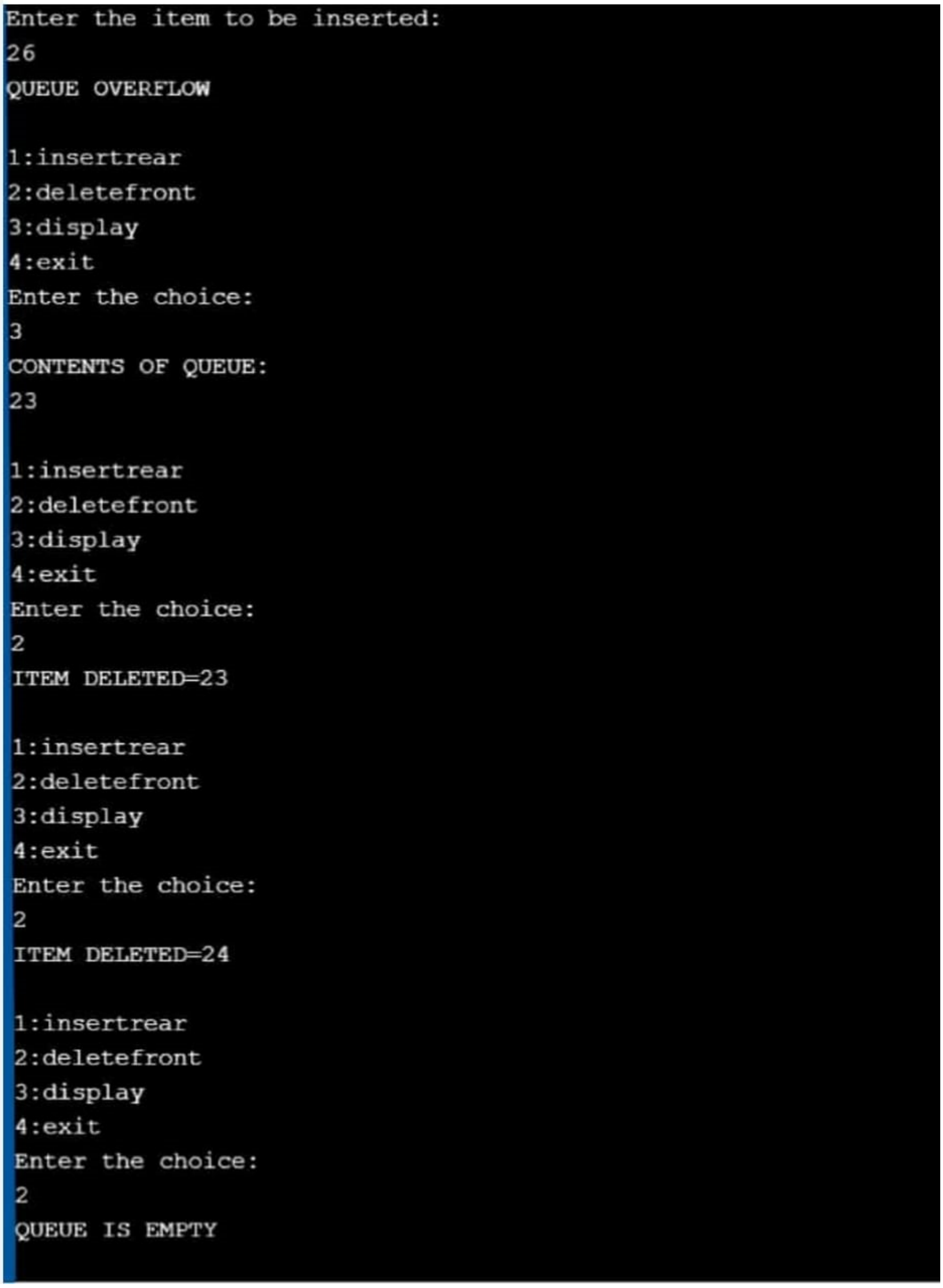
```
#include<stdio.h>

#include<stdlib.h>

#define Q_size 2

int item,front =0,rear=-1,q[10];

void insertrear()

{

  if(rear==Q_size-1)

  {

    printf("QUEUE OVERFLOW\n");

    return;

  }

  rear=rear+1;

  q[rear]=item;

}
```

```
int deletefront()
{
   if(front>rear)
   {
      front=0;
      rear=-1;
      return -1;
   }
   return q[front++];
}
void displayQ()
{
   int i;
   if(front>rear)
   {
      printf("QUEUE IS EMPTY\n");
      return;
   }
   printf("CONTENTS OF QUEUE:\n");
   for(i=front;i<rear;i++)
   {
      printf("%d\n",q[i]);
   }
}
void main()
{
   int choice;
   for(;;)
   {
      printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
      printf("Enter the choice:\n");
```

```
        scanf("%d",&choice);
        switch(choice)
        {
          case 1:printf("Enter the item to be inserted:\n");
          scanf("%d",&item);
          insertrear();
          break;
          case 2:item=deletefront();
          if(item==-1)
          printf("QUEUE IS EMPTY\n");
          else
          printf("ITEM DELETED=%d\n",item);
          break;
          case 3:displayQ();
          break;
          default:exit(0);
        }
    }
}
```



```
1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
1
Enter the item to be inserted:
23

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
1
Enter the item to be inserted:
24

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
1
Enter the item to be inserted:
26
QUEUE OVERFLOW

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
3
CONTENTS OF QUEUE:
23
```



```
Enter the item to be inserted:
26
QUEUE OVERFLOW

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
3
CONTENTS OF QUEUE:
23

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
2
ITEM DELETED=23

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
2
ITEM DELETED=24

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice:
2
QUEUE IS EMPTY
```

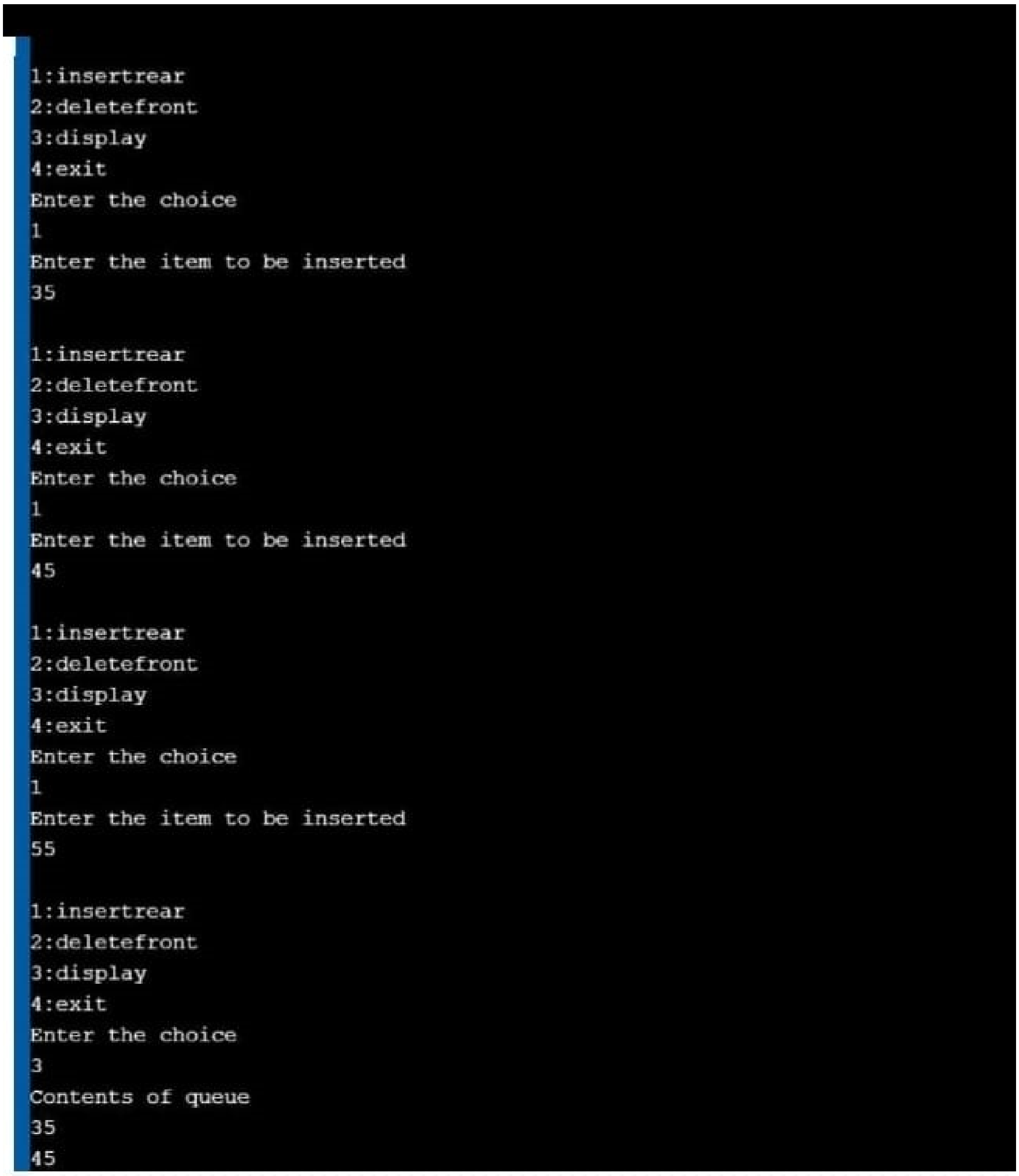
LAB PROGRAM 4

WAP to simulate the working of a circular queue of integers using an array. Provide the following operations. a) Insert b) Delete c) Display The program should print appropriate messages for queue empty and queue overflow conditions

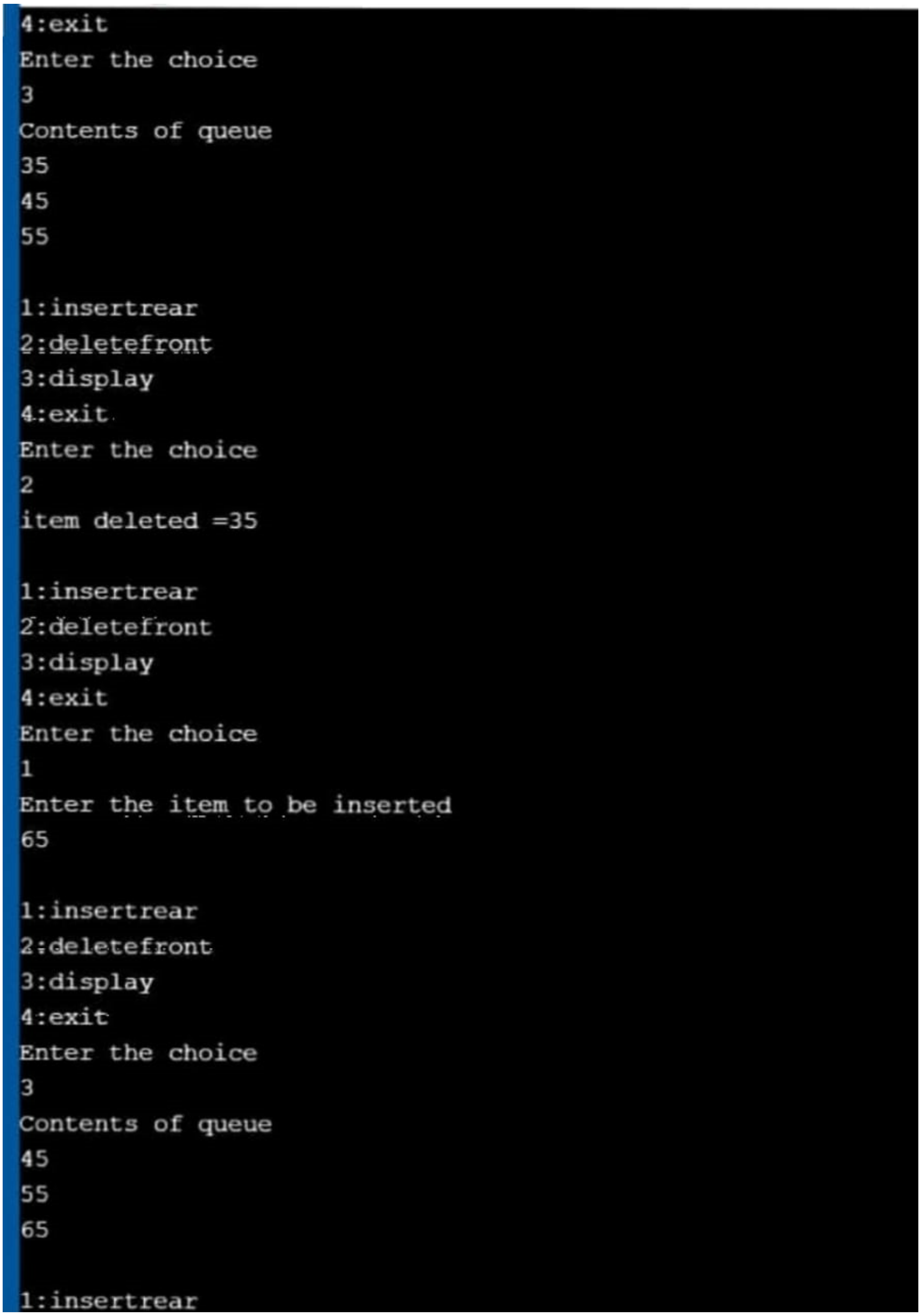
```
#include<stdio.h>

#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\n");

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=count-1;

return item;

}

void displayQ()

{

int i,f;
```

```
if(count==0)
{
printf("queue is empty\n");
return;
}
f=front;
printf("Contents of queue \n");
for(i=1;i<=count;i++)
{
printf("%d\n",q[f]);
f=(f+1)%QUE_SIZE;
}
}
void main()
{
 int choice;


 for(;;)
 {
 printf("\n1:insertrear\n2:deletefront\n3:display\n4:exit\n");
 printf("enter the choice\n");
 scanf("%d",&choice);

 switch(choice)
 {
 case 1:printf("enter the item to be inserted\n");
             scanf("%d",&item);
             insertrear();
             break;
 case 2:item=deletefront();
```

```
            if(item==-1)
            printf("queue is empty\n");
            else
            printf("item deleted =%d\n",item);
            break;
 case 3:displayQ();
            break;
 default:exit(0);
}
}

}
```



```
1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
1
Enter the item to be inserted
35

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
1
Enter the item to be inserted
45

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
1
Enter the item to be inserted
55

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
3
Contents of queue
35
45
```



```
4:exit
Enter the choice
3
Contents of queue
35
45
55

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
2
item deleted =35

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
1
Enter the item to be inserted
65

1:insertrear
2:deletefront
3:display
4:exit
Enter the choice
3
Contents of queue
45
55
65

1:insertrear
```

LAB PROGRAM 5 and 6

WAP to Implement Singly Linked List with following operations a) a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.

WAP to Implement Singly Linked List with following operations a) a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.

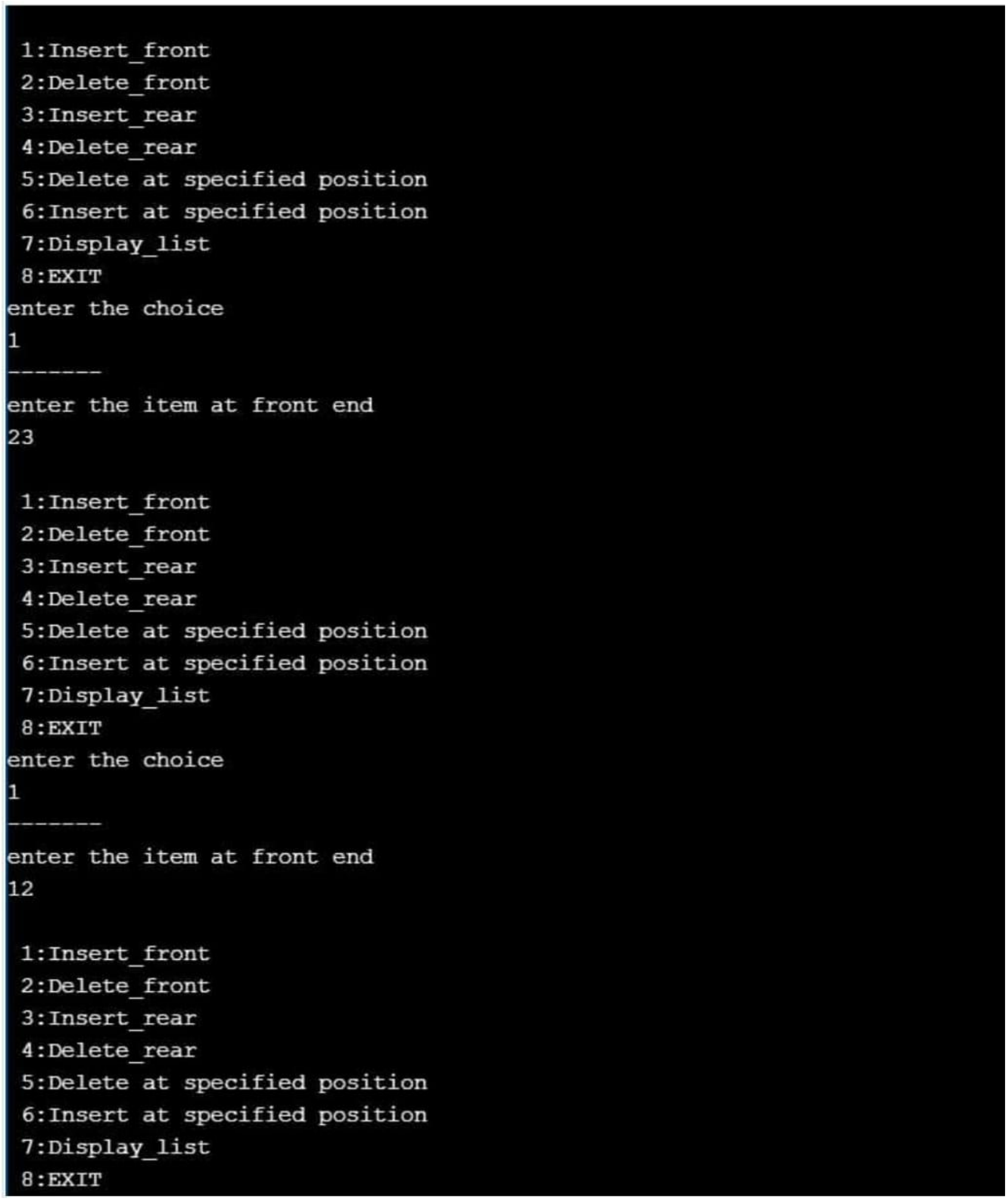
```
#include<stdio.h>

#include<stdlib.h>

#include<conio.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("memory 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_front(NODE first)
{
   NODE temp;
   if(first==NULL)
   {
      printf("list is empty cannot delete\n");
      return first;
   }
   temp=first;
   temp=temp->link;
   printf("item deleted at front end is=%d \n",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;
}
NODE delete_rear(NODE first)
{
   NODE cur,prev;
   if(first==NULL)
   {
      printf("list is empty cannot 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",cur->info);
   free(cur);
   prev->link=NULL;
   return first;
}

NODE delete_pos(int pos,NODE first)
{
            NODE prev,cur;
            int count;
            if (first==NULL || pos<=0)
            {
                        printf("Invalid position\n");
                        return NULL;
            }
            if (pos==1)
            {
                        cur=first;
                        first=first->link;
                        freenode(cur);
                        return first;
            }
            prev=NULL;
            cur=first;
            count=1;
            while (cur!=NULL)
            {
                        if (count==pos)
                        {
                                    break;
```

```
		}
		prev=cur;
		cur=cur->link;count++;
	}
	if (count!=pos)
	{
		printf("Invalid position\n");
		return first;
	}
	prev->link=cur->link;
	freenode(cur);
	return first;
}
NODE insert_pos(int item,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\n");
		return NULL;
	}
	if (pos==1)
	{
```

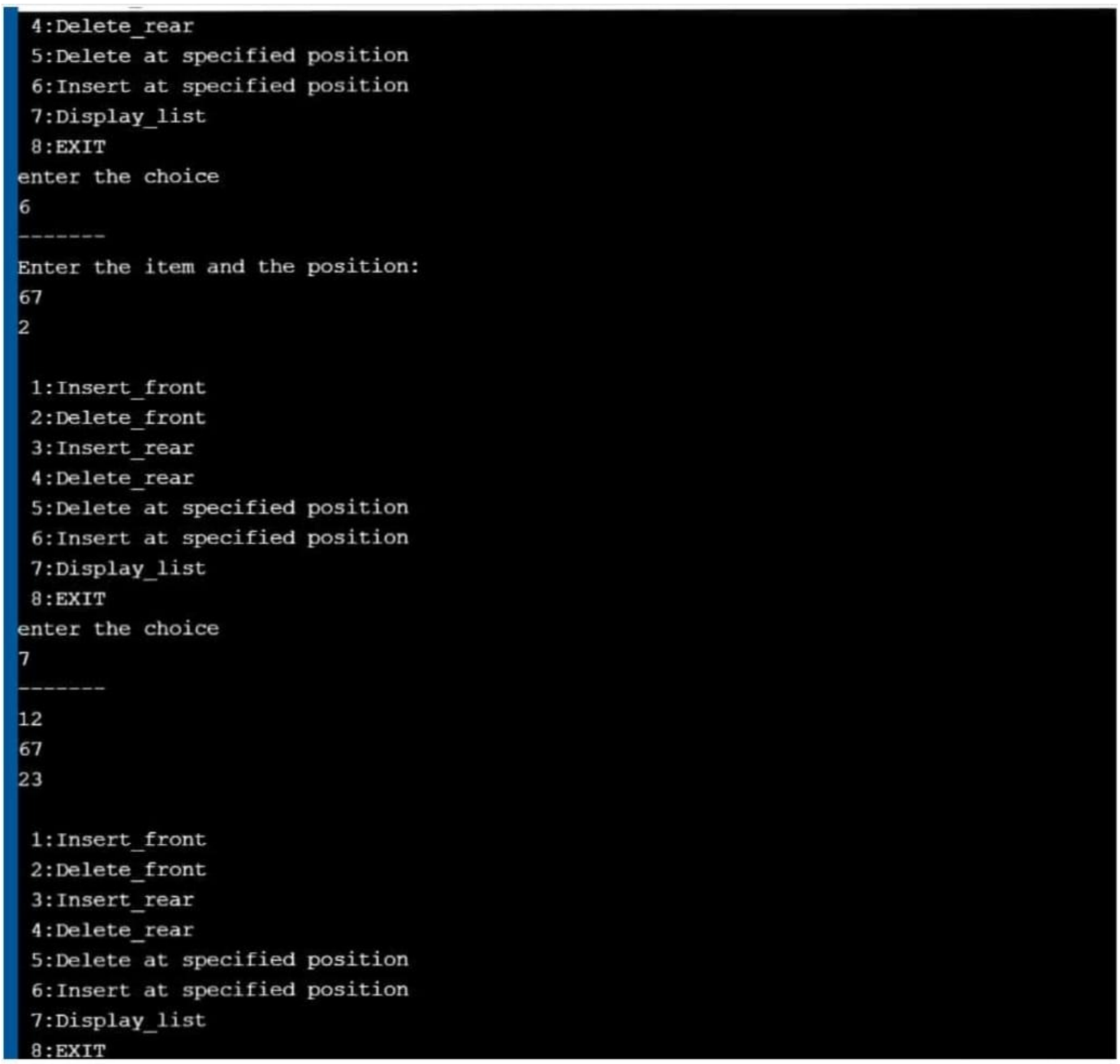
```
                temp->link=first;
                return temp;
        }
        count=1;
        prev=NULL;
        cur=first;
        while (cur!=NULL && count!=pos)
        {
                prev=cur;
                cur=cur->link;
                count++;
        }
        if (count==pos)
        {
                prev->link=temp;
                temp->link=cur;
                return first;
        }
        printf("Invalid position\n");
        return first;
}

void display(NODE first)
{
  NODE temp;
  if(first==NULL)
  printf("list empty cannot display items\n");
  for(temp=first;temp!=NULL;temp=temp->link)
  {
    printf("%d\n",temp->info);
  }
```

```
}
void main()
{
   int item,choice,pos;
   NODE first=NULL;
   for(;;)
   {
       printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n 5:Delete at
specified position\n 6:Insert at specified position\n 7:Display_list\n 8:EXIT\n");
       printf("enter the choice\n");
       scanf("%d",&choice);
       printf("-------\n");
       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:first=delete_rear(first);
          break;
          case 5:printf("Enter the position:\n");
                                       scanf("%d",&pos);
                                       first=delete_pos(pos,first);
                                       break;
                                   case 6:printf("Enter the item and the position:\n");
```

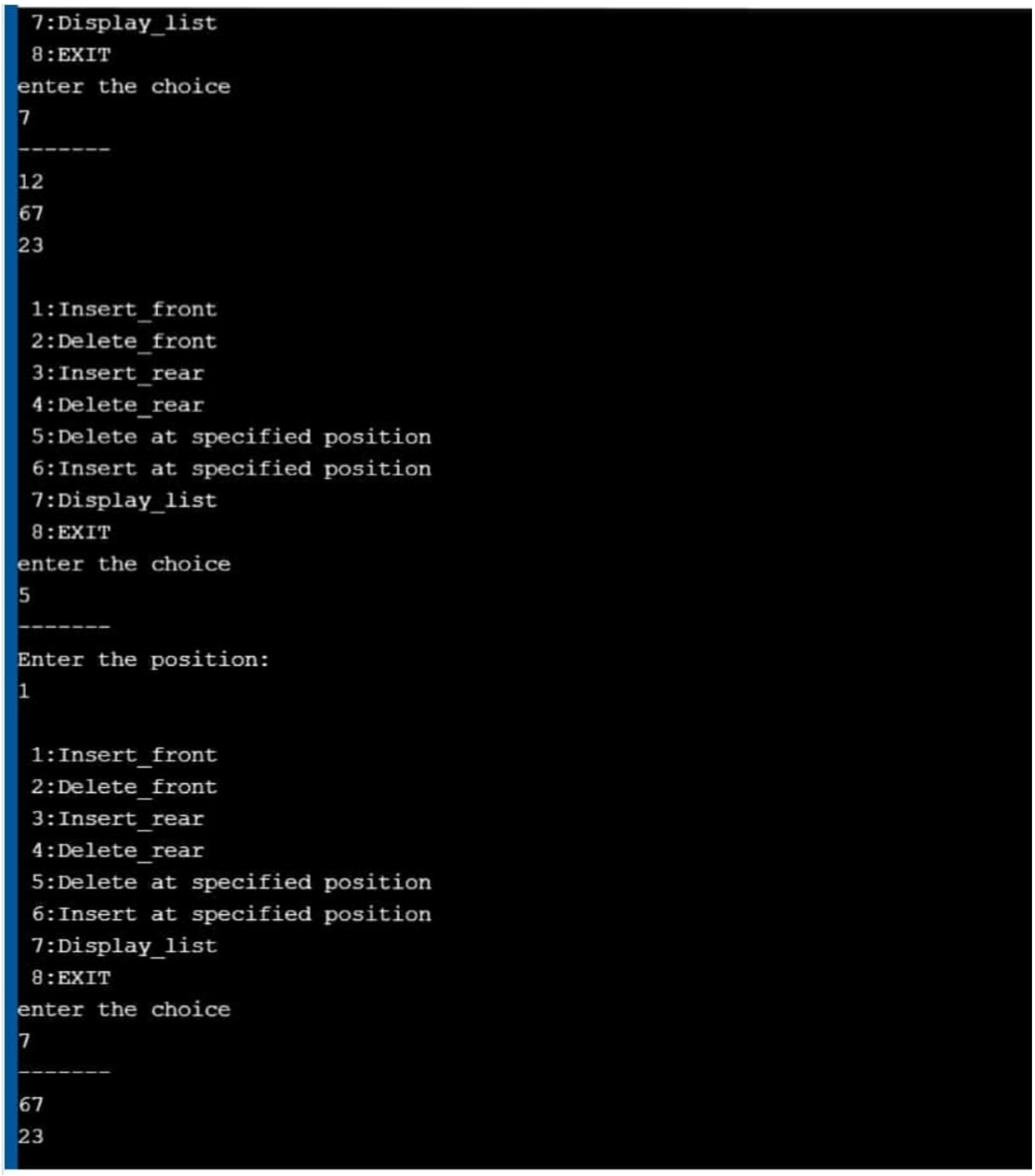
```
                                    scanf("%d%d",&item,&pos);

                                    first=insert_pos(item,pos,first);

                                    break;

        case 7:display(first);

        break;

        default:exit(0);

        break;

      }

    }

}
```



```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
enter the choice
1
-------
enter the item at front end
23

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
enter the choice
1
-------
enter the item at front end
12

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
```



```
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
enter the choice
6
-------
Enter the item and the position:
67
2

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
enter the choice
7
-------
12
67
23

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
```



```
 7:Display_list
 8:EXIT
enter the choice
7
-------
12
67
23

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
enter the choice
5
-------
Enter the position:
1

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Delete at specified position
 6:Insert at specified position
 7:Display_list
 8:EXIT
enter the choice
7
-------
67
23
```

LAB PROGRAM 7

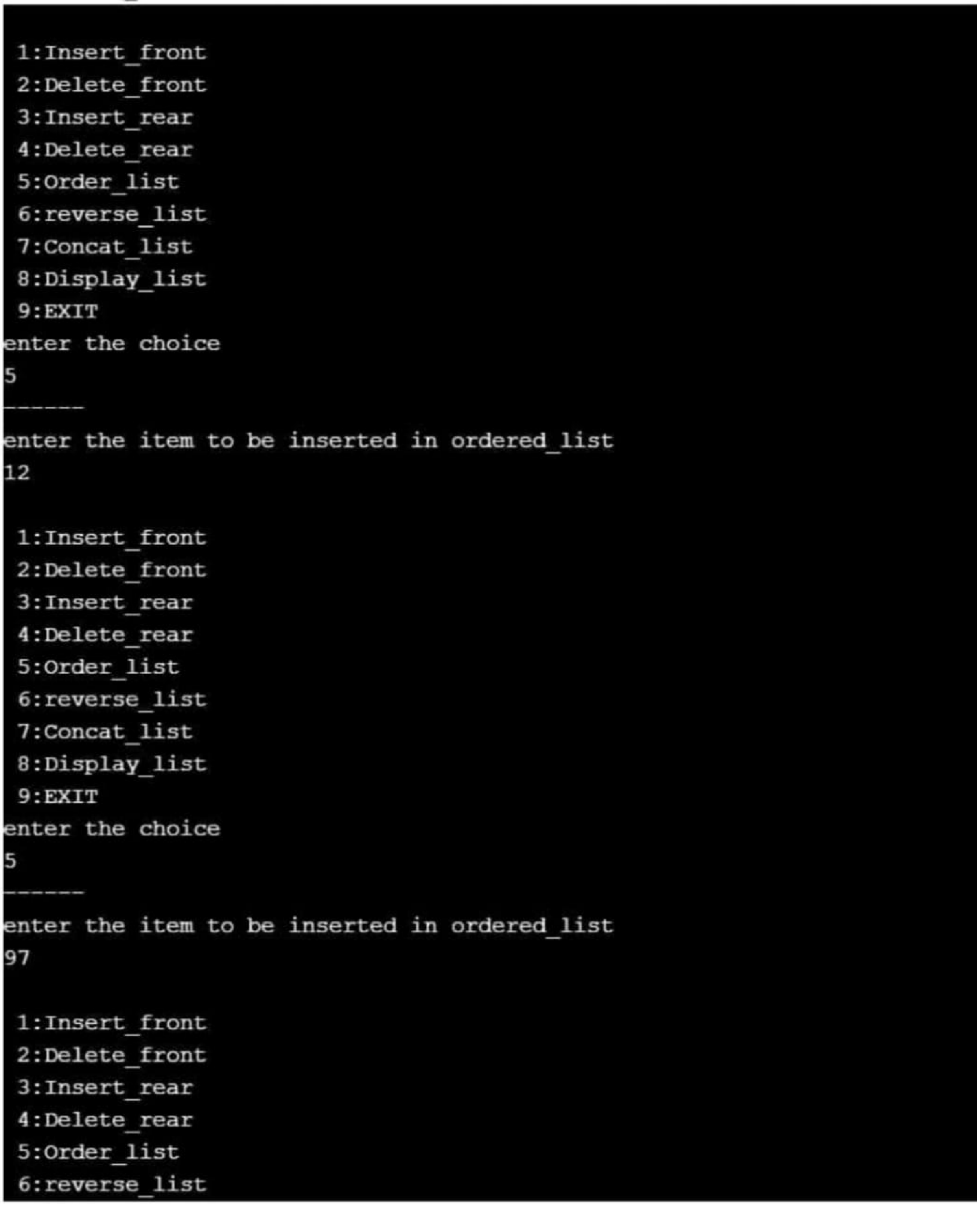
WAP Implement Single Link List with following operations a) a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.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("memory 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;
```

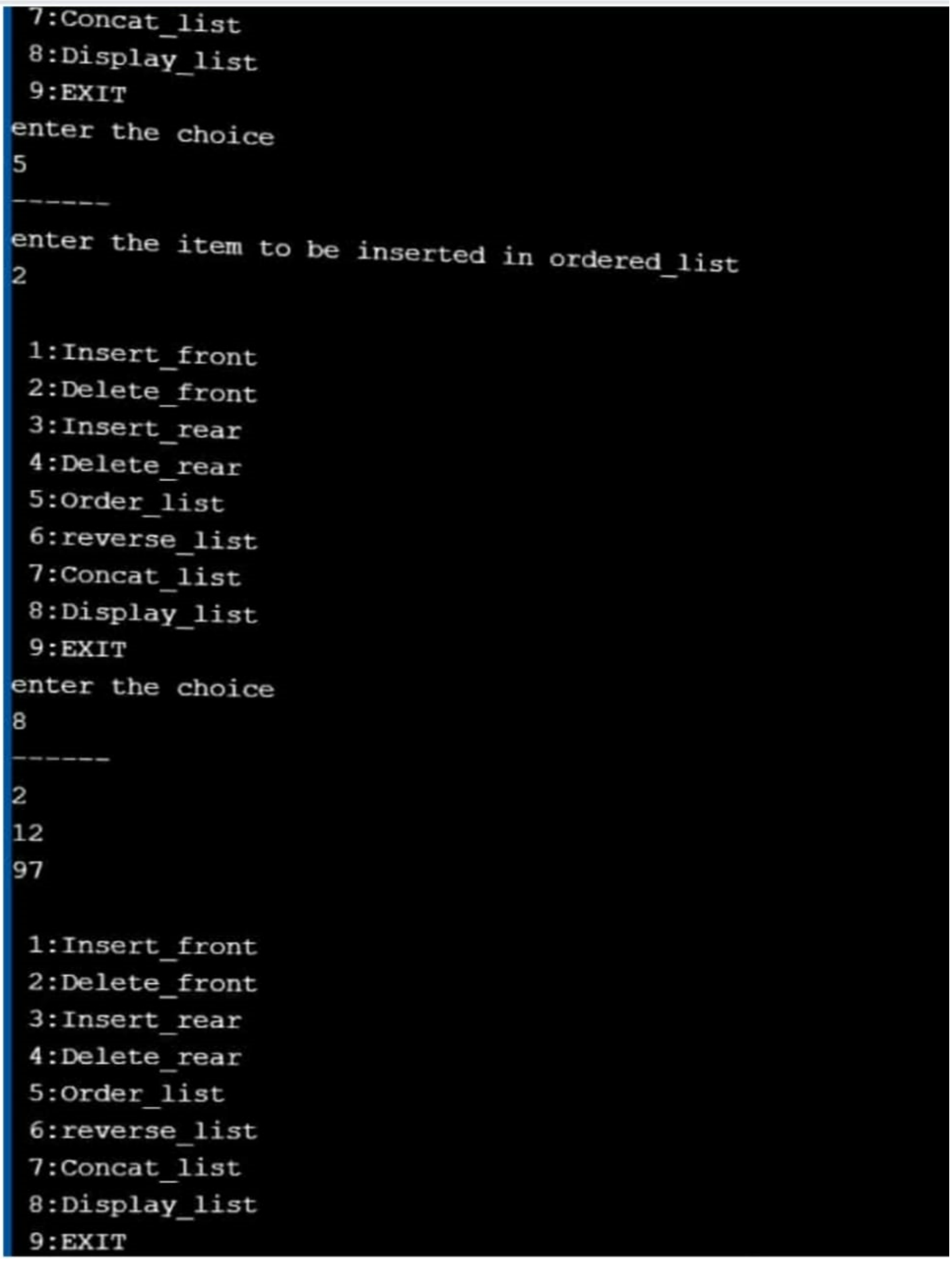
```
    if(first==NULL)
    return temp;
    temp->link=first;
    first=temp;
    return first;
}
NODE delete_front(NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("list is empty cannot delete\n");
        return first;
    }
    temp=first;
    temp=temp->link;
    printf("item deleted at front end is=%d \n",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;
}
NODE delete_rear(NODE first)
{
    NODE cur,prev;
    if(first==NULL)
    {
        printf("list is empty cannot 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",cur->info);
    free(cur);
    prev->link=NULL;
    return first;
}
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 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;
    while(cur->link!=NULL)
    cur=cur->link;
    cur->link=second;
    return first;
}
void display(NODE first)
{
    NODE temp;
    if(first==NULL)
    printf("list empty cannot display items\n");
    for(temp=first;temp!=NULL;temp=temp->link)
    {
        printf("%d\n",temp->info);
    }
}
void main()
{
    int item,choice,n,i;
    NODE first=NULL,a,b;
    for(;;)
```

```
   {
       printf("\n 1:Insert_front\n 2:Delete_front\n 3:Insert_rear\n 4:Delete_rear\n 5:Order_list\n
6:reverse_list\n 7:Concat_list\n 8:Display_list\n 9:EXIT\n");
       printf("enter the choice\n");
       scanf("%d",&choice);
       printf("------\n");
       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:first=delete_rear(first);
         break;
         case 5:printf("enter the item to be inserted in ordered_list\n");
         scanf("%d",&item);
         first=order_list(item,first);
         break;
         case 6:first=reverse(first);
         display(first);
         break;
         case 7:printf("Enter the no of nodes in 1\n");
                    scanf("%d",&n);
                    a=NULL;
                    for(i=0;i<n;i++)
```

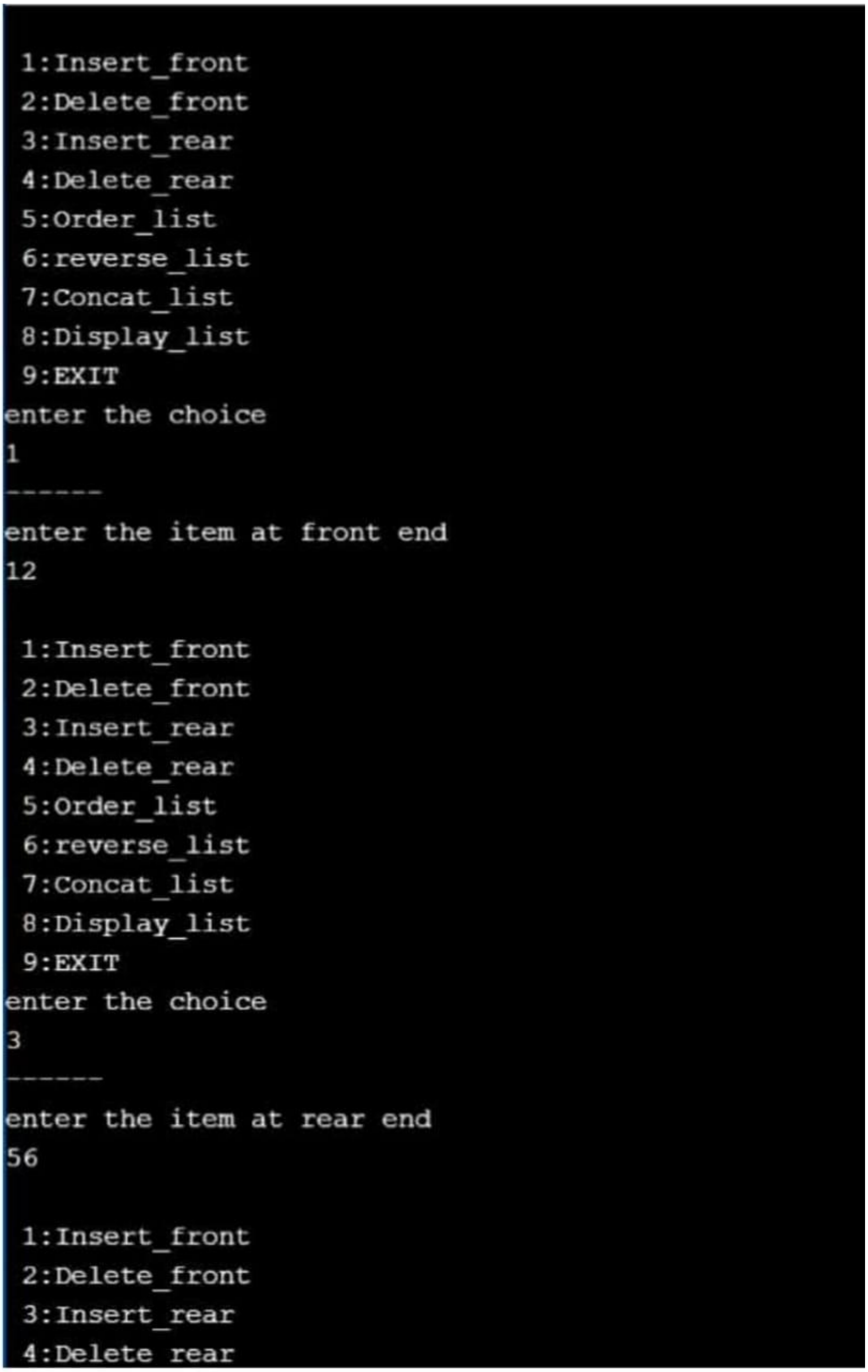
```
                        {
                           printf("Enter the item\n");
                           scanf("%d",&item);
                           a=insert_rear(a,item);
                        }
                        printf("Enter the no of nodes in 2\n");
                        scanf("%d",&n);
                        b=NULL;
                        for(i=0;i<n;i++)
                        {
                          printf("Enter the item\n");
                          scanf("%d",&item);
                          b=insert_rear(b,item);
                        }
                        a=concat(a,b);
                        display(a);
                        break;
          case 8:display(first);
          break;
          default:exit(0);
          break;
        }
    }
}
```



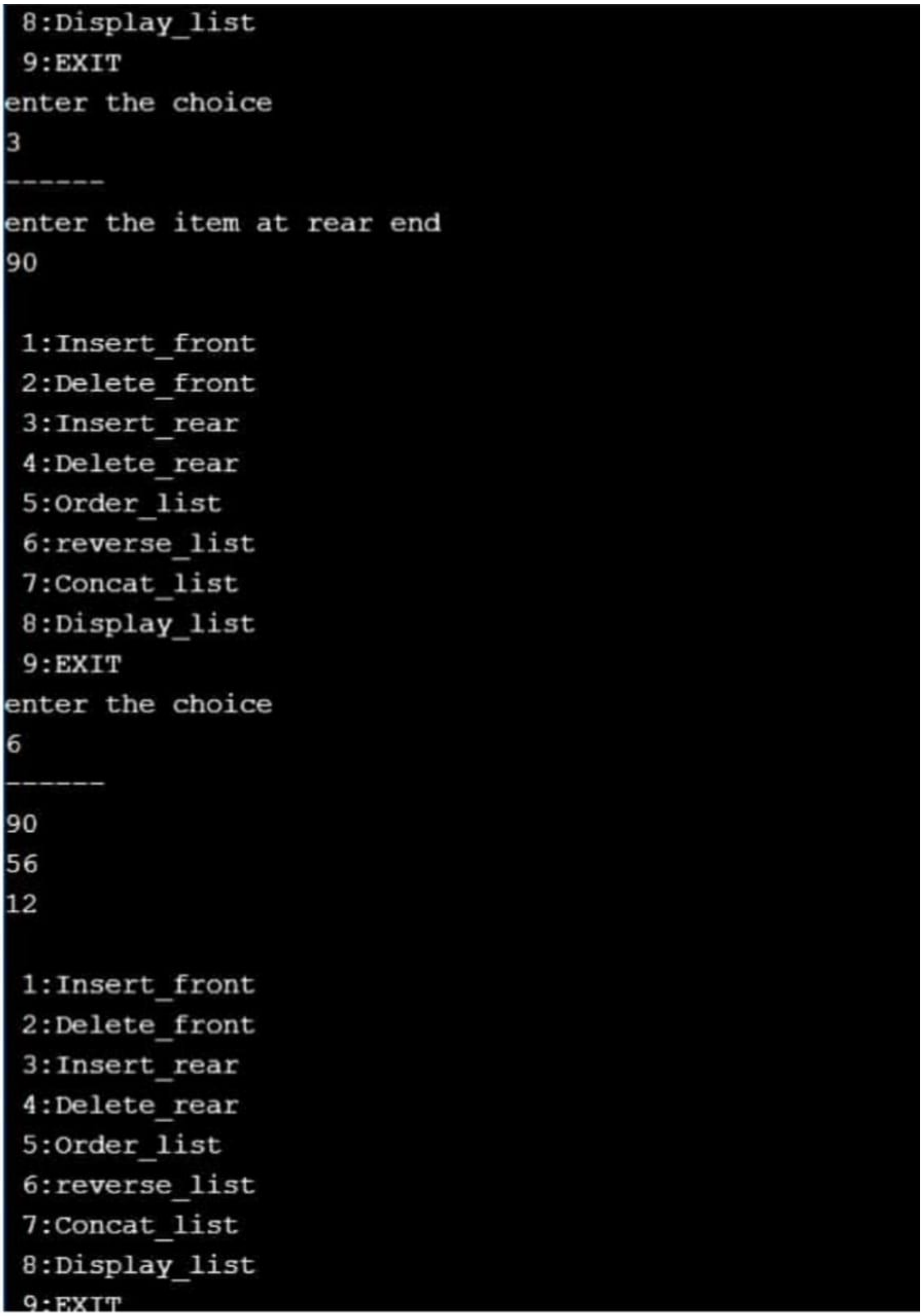
```
 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
5
------
enter the item to be inserted in ordered_list
12

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
5
------
enter the item to be inserted in ordered_list
97

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
```



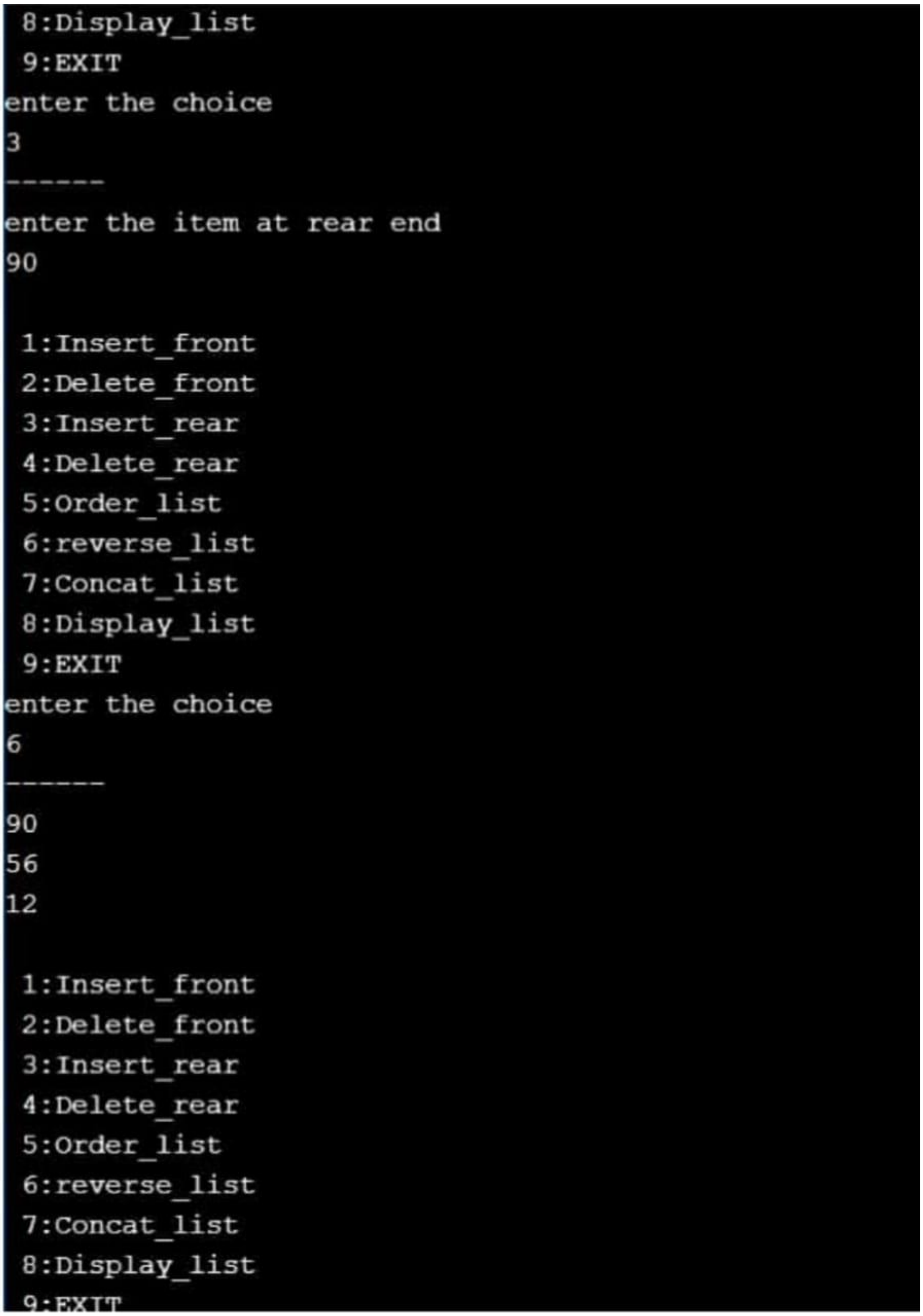
```
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
5
------
enter the item to be inserted in ordered_list
2

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
8
------
2
12
97

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
```



```

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
1
------
enter the item at front end
12

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
3
------
enter the item at rear end
56

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
```



```
 8:Display_list
 9:EXIT
enter the choice
3
------
enter the item at rear end
90

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
6
------
90
56
12

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
```



```
 8:Display_list
 9:EXIT
enter the choice
3
------
enter the item at rear end
90

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
enter the choice
6
------
90
56
12

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:reverse_list
 7:Concat_list
 8:Display_list
 9:EXIT
```

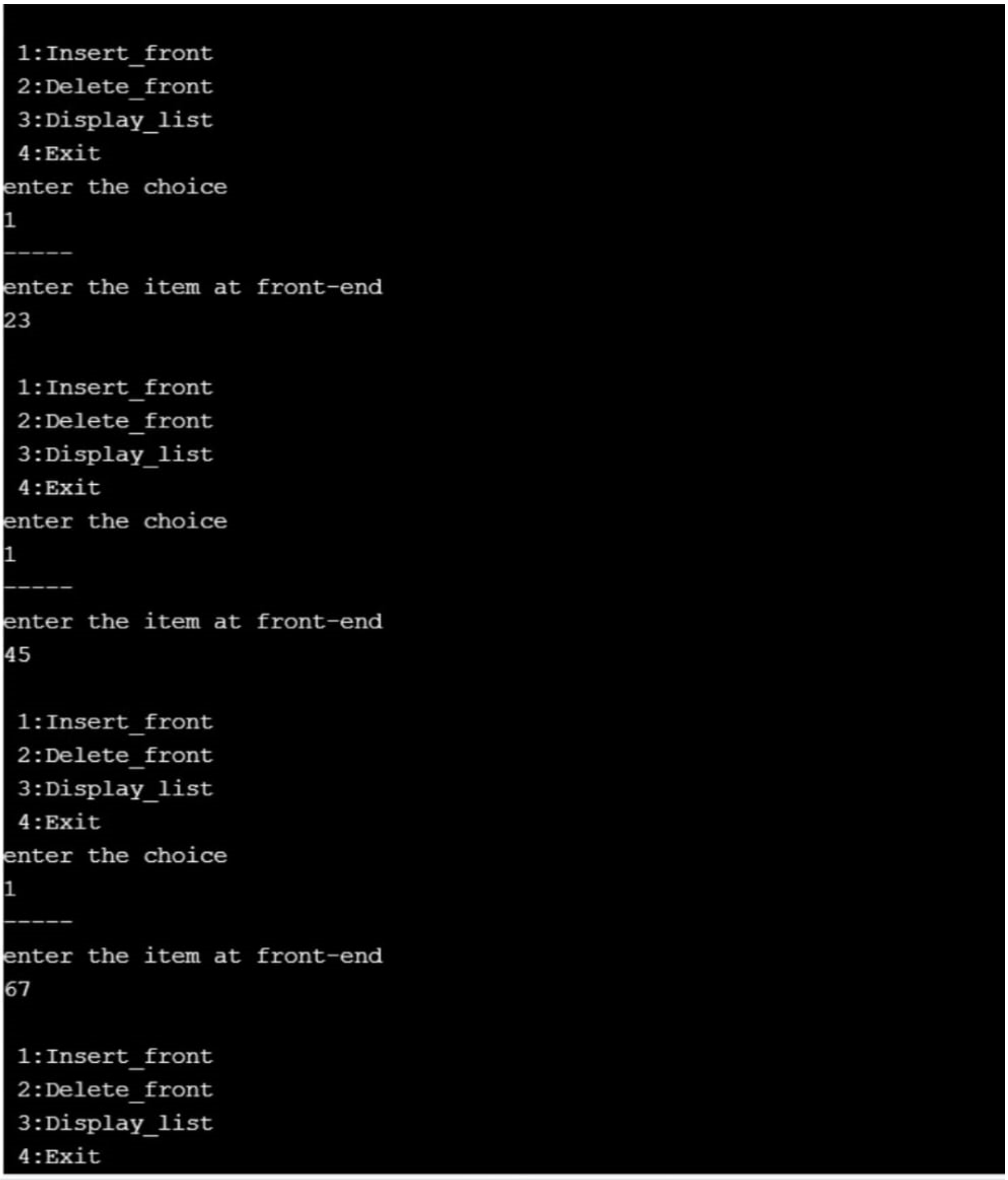
LAB PROGRAM 8

WAP to implement Stack & Queues using Linked Representation

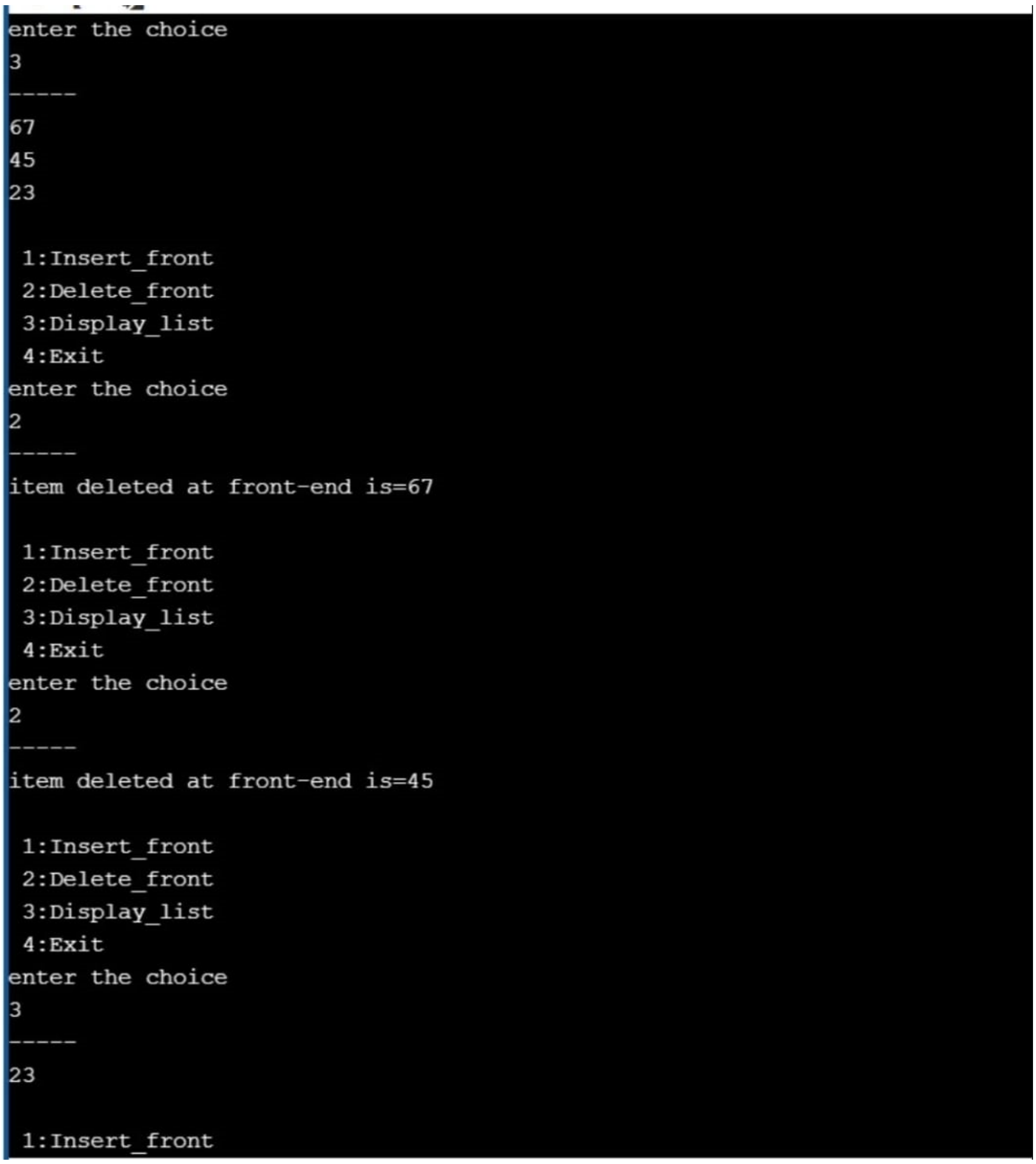
```
//Implementing stack using linked list

#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("mem full\n");
     exit(0);
   }
   return x;
}
void freenode(NODE x)
{
   free(x);
}
NODE insert_front(NODE first,int item)
```

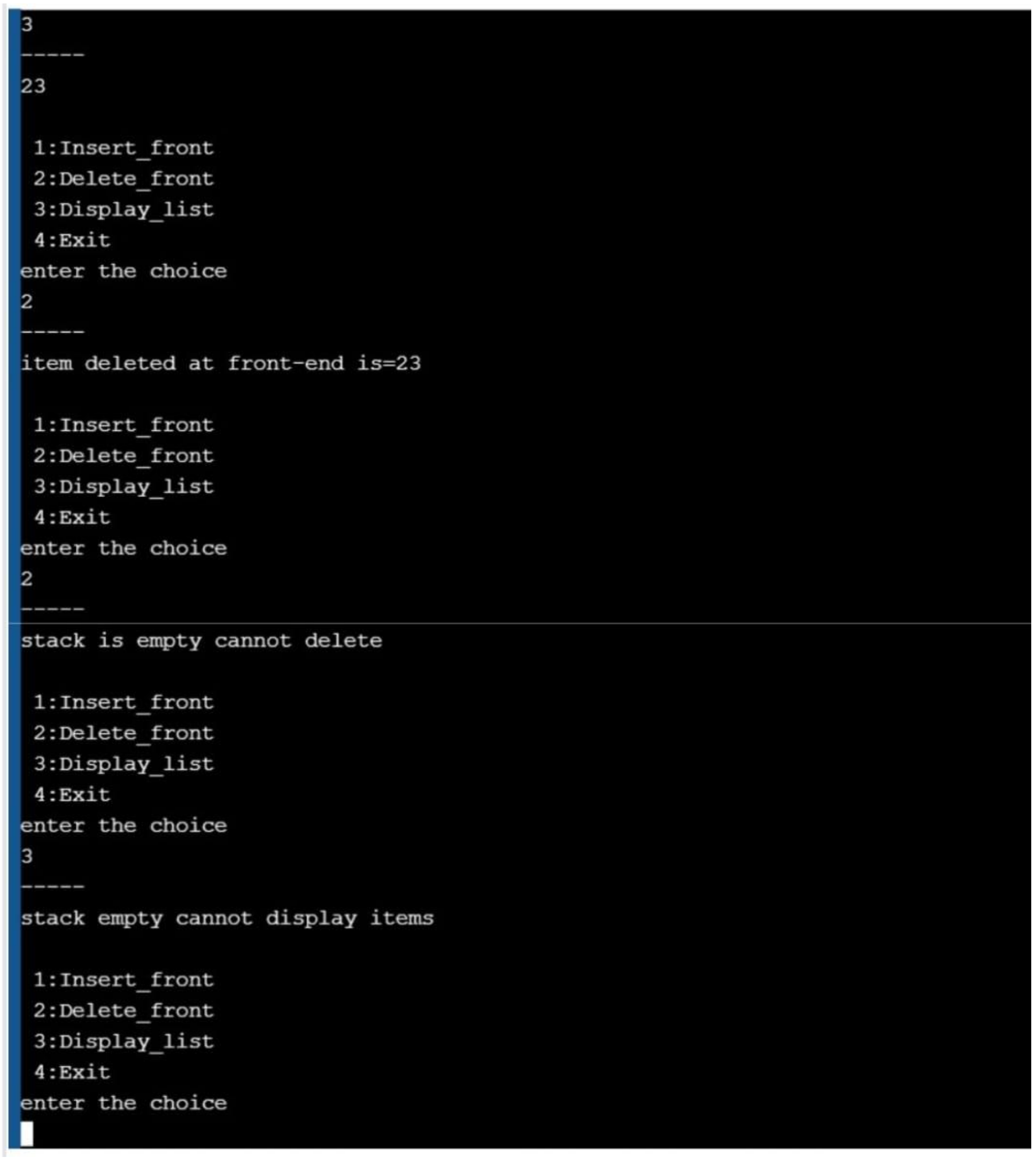
```
{
   NODE temp;
   temp=getnode();
   temp->info=item;
   temp->link=NULL;
   if(first==NULL)
   return temp;
   temp->link=first;
   first=temp;
   return first;
}
NODE delete_front(NODE first)
{
   NODE temp;
   if(first==NULL)
   {
      printf("stack is empty cannot delete\n");
      return first;
   }
   temp=first;
   temp=temp->link;
   printf("item deleted at front-end is=%d\n",first->info);
   free(first);
   return temp;
}
void display(NODE first)
{
   NODE temp;
   if(first==NULL)
   printf("stack empty cannot display items\n");
   for(temp=first;temp!=NULL;temp=temp->link)
```

```
  {
     printf("%d\n",temp->info);
  }
}
int main()
{
  int item,choice,pos;
  NODE first=NULL;
  for(;;)
  {
     printf("\n 1:Insert_front\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
     printf("enter the choice\n");
     scanf("%d",&choice);
     printf("-----\n");
     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:display(first);
        break;
        default:exit(0);
        break;
     }
  }
}
```



```
 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
-----
enter the item at front-end
23

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
-----
enter the item at front-end
45

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
-----
enter the item at front-end
67

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
```


```
enter the choice
3
-----
67
45
23

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
-----
item deleted at front-end is=67

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
-----
item deleted at front-end is=45

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
-----
23

 1:Insert_front
```



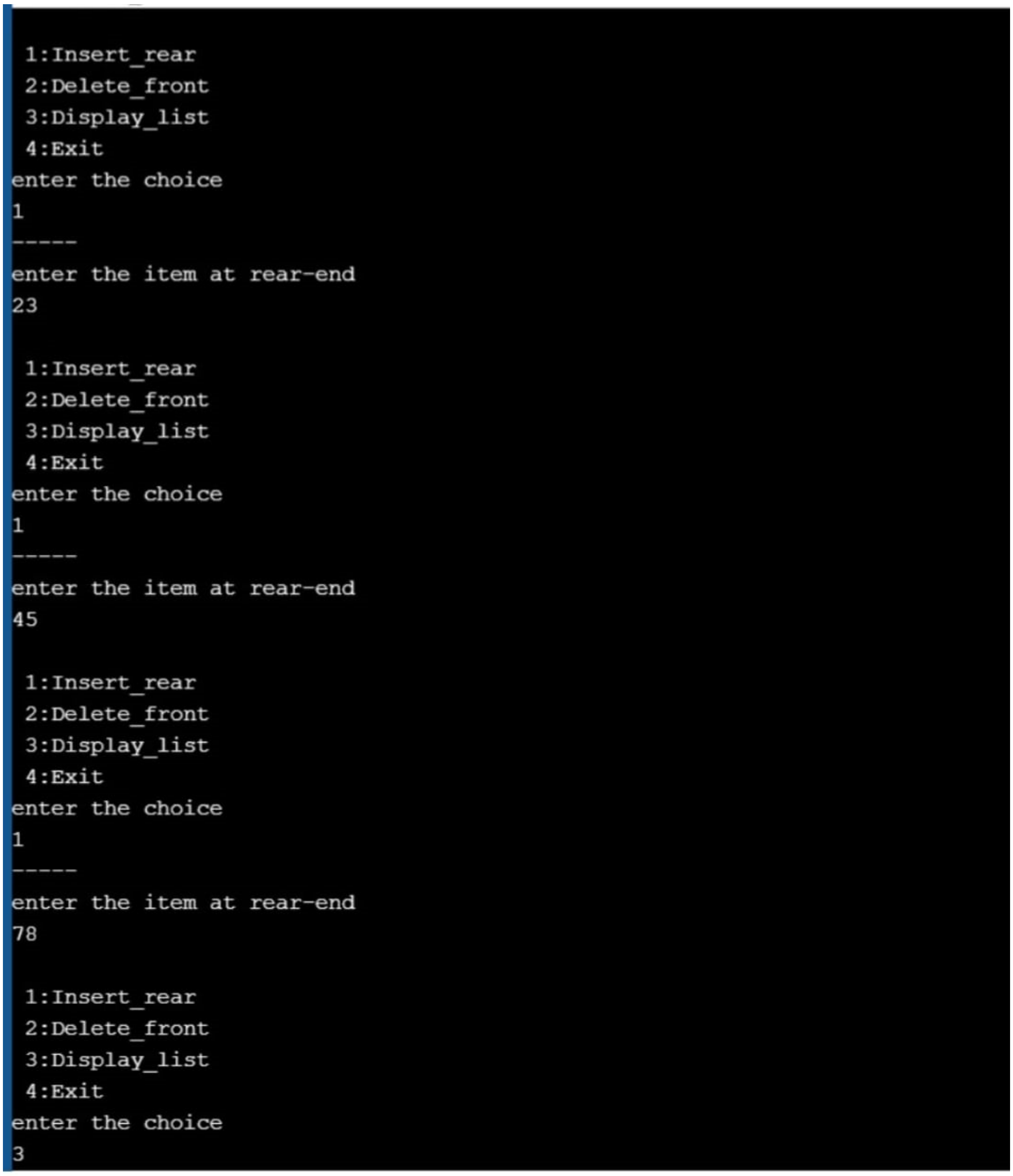
```
3
-----
23

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
-----
item deleted at front-end is=23

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
-----
stack is empty cannot delete

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
-----
stack empty cannot display items

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

```
//Implementing queue using linked list

#include<stdio.h>
#include<stdlib.h>
struct node
{
   int info;
```

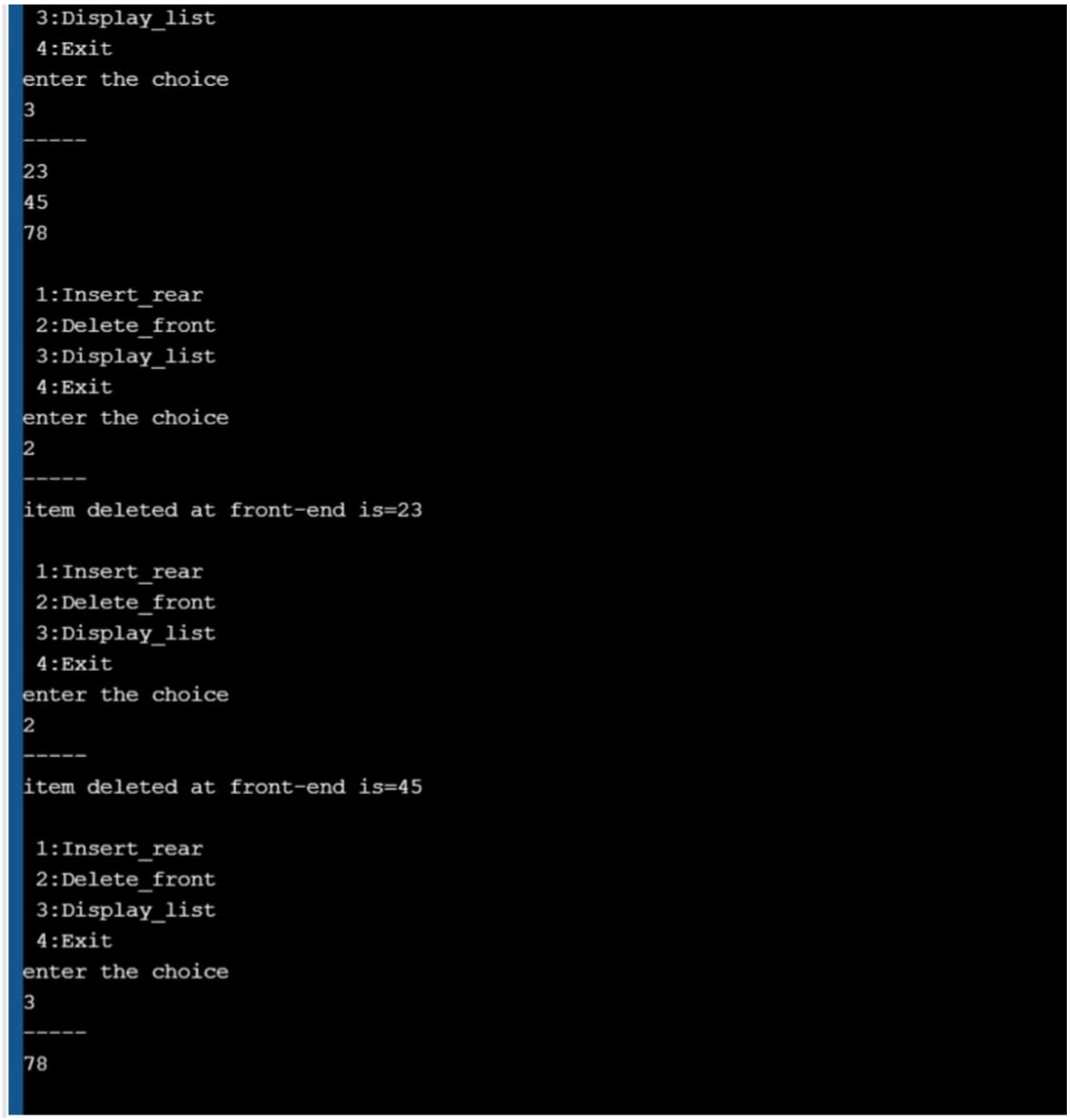
```
    struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE)malloc(sizeof(struct node));
    if(x==NULL)
    {
       printf("mem full\n");
       exit(0);
    }
    return x;
}
void freenode(NODE x)
{
    free(x);
}
NODE insert_rear(NODE first,int item)
{
    NODE 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;
   if(first==NULL)
   {
      printf("list is empty cannot delete\n");
      return first;
   }
   temp=first;
   temp=temp->link;
   printf("item deleted at front-end is=%d\n",first->info);
   free(first);
   return temp;
}
void display(NODE first)
{
   NODE temp;
   if(first==NULL)
   printf("list empty cannot display items\n");
   for(temp=first;temp!=NULL;temp=temp->link)
   {
      printf("%d\n",temp->info);
   }
}
int main()
{
   int item,choice,pos;
   NODE first=NULL;

   for(;;)
```

```
  {
    printf("\n 1:Insert_rear\n 2:Delete_front\n 3:Display_list\n 4:Exit\n");
    printf("enter the choice\n");
    scanf("%d",&choice);
    printf("-----\n");
    switch(choice)
    {
      case 1:printf("enter the item at rear-end\n");
      scanf("%d",&item);
      first=insert_rear(first,item);
      break;
      case 2:first=delete_front(first);
      break;
      case 3:display(first);
      break;
      default:exit(0);
      break;
    }
  }

}
```



```
 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
-----
enter the item at rear-end
23

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
-----
enter the item at rear-end
45

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
-----
enter the item at rear-end
78

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



```
 3:Display_list
 4:Exit
enter the choice
3
-----
23
45
78

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
-----
item deleted at front-end is=23

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
-----
item deleted at front-end is=45

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
-----
78
```

LAB PROGRAM 9

WAP Implement doubly link list with primitive operations a) a) Create a doubly linked list. b) Insert a new node to the left of the node. b) c) Delete the node based on a specific value. c) Display the contents of the 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("mem full\n");
       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(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)
    {
       printf("list empty\n");
       return head;
    }
    cur=head->rlink;
    next=cur->rlink;
    head->rlink=next;
    next->llink=head;
    printf("the node deleted is %d",cur->info);
    freenode(cur);
    return head;
}
NODE ddelete_rear(NODE head)
```

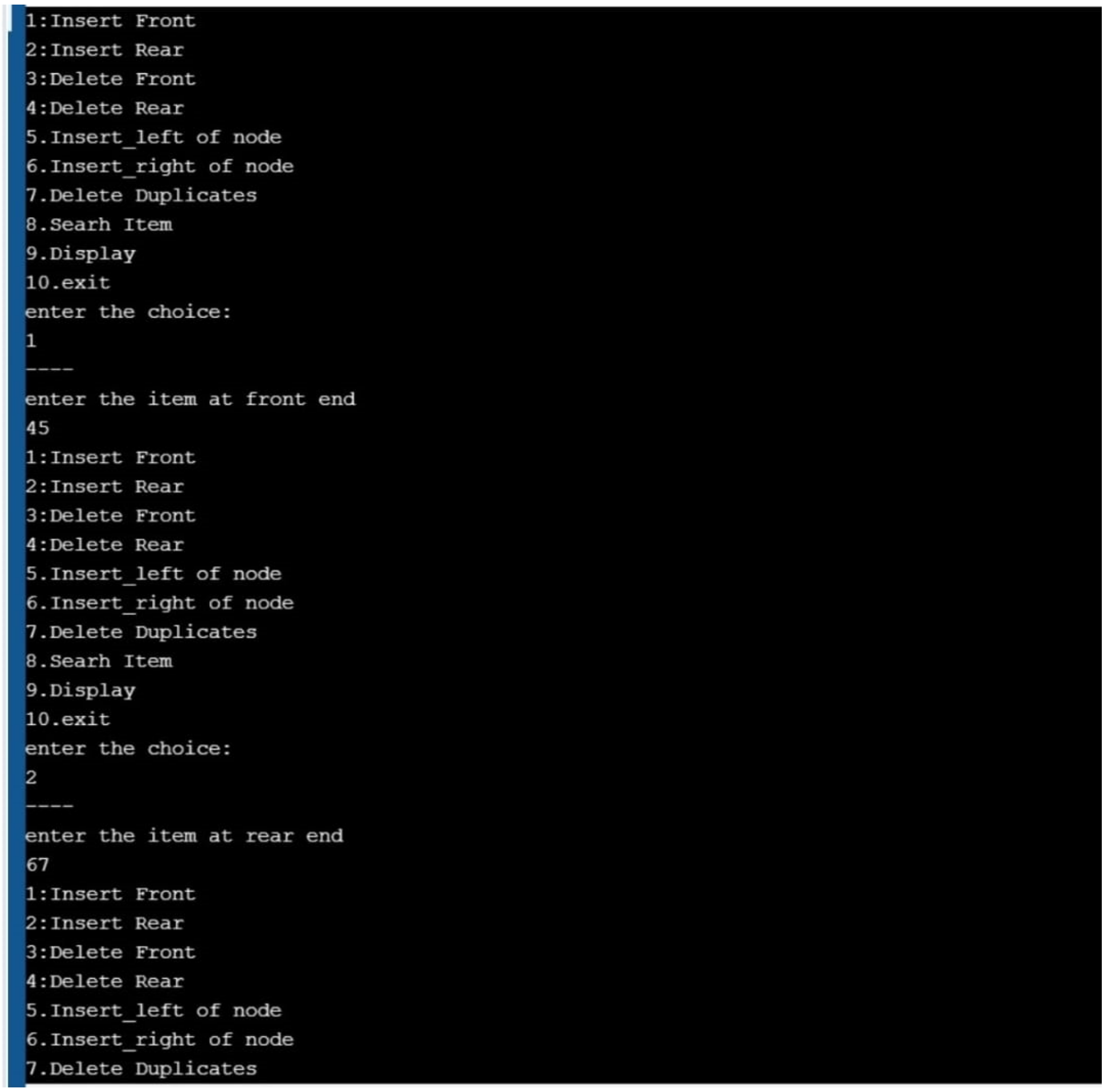
```
{
   NODE cur,prev;
   if(head->rlink==head)
   {
     printf("list empty\n");
     return head;
   }
   cur=head->llink;
   prev=cur->llink;
   head->llink=prev;
   prev->rlink=head;
   printf("the node deleted is %d",cur->info);
   freenode(cur);
   return head;
}
NODE insert_leftpos(int item,NODE 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)
   {
      printf("key not found\n");
      return head;
   }
   prev=cur->llink;
   printf("enter towards left of %d=",item);
   temp=getnode();
   scanf("%d",&temp->info);
   prev->rlink=temp;
   temp->llink=prev;
   cur->llink=temp;
   temp->rlink=cur;
   return head;
}
NODE insert_rightpos(int 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)
   {
      printf("key not found\n");
      return head;
   }
   prev=cur->rlink;
   printf("enter towards right of %d=",item);
   temp=getnode();
   scanf("%d",&temp->info);
   prev->llink=temp;
   temp->llink=cur;
   cur->rlink=temp;
   temp->rlink=prev;
   return head;
}
NODE delete_all_key(int item,NODE head)
{
   NODE prev,cur,next;
   int count;
   if(head->rlink==head)
   {
      printf("List Empty");
      return head;
   }
   count=0;
   cur=head->rlink;
   while(cur!=head)
   {
      if(item!=cur->info)
      cur=cur->rlink;
      else
      {
          count++;
          if(count==1)
          {
```

```
            cur=cur->rlink;
            continue;
          }
          prev=cur->llink;
          next=cur->rlink;
          prev->rlink=next;
          next->llink=prev;
          freenode(cur);
          cur=next;
      }
   }
   if(count==0)
   printf("key not found");
   else
   printf("key found at %d positions\n", count);
   return head;
}

void search_info(int item,NODE head)
{
   NODE cur;
   if(head->rlink==head)
   {
      printf("list empty\n");
   }
   cur=head->rlink;
   while(cur!=head)
   {
      if(item==cur->info)
      {
          printf("Search Successfull\n");
```

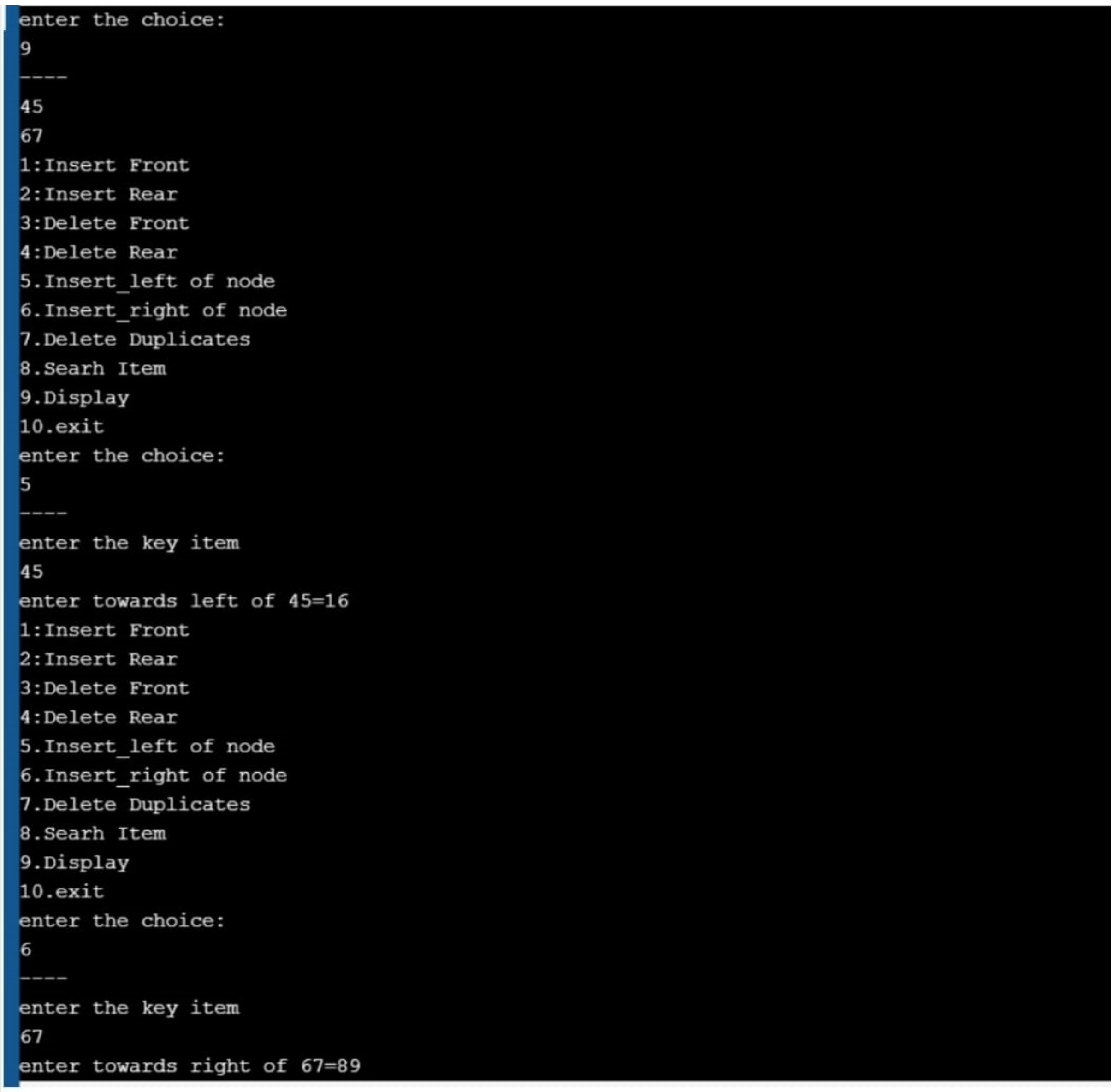
```
            break;
         }
         cur=cur->rlink;
      }
      if(cur==head)
      {
         printf("Element not found\n");
      }
   }
   void display(NODE head)
   {
      NODE temp;
      if(head->rlink==head)
      {
         printf("list empty\n");
         return;
      }
      for(temp=head->rlink;temp!=head;temp=temp->rlink)
      printf("%d\n",temp->info);
   }
   void main()
   {
      int item,choice,key;
      NODE head,last;
      head=getnode();
      head->rlink=head;
      head->llink=head;
      for(;;)
      {
         printf("1:Insert Front\n2:Insert Rear\n3:Delete Front\n4:Delete Rear\n5.Insert_left of node");
         printf("\n6.Insert_right of node\n7.Delete Duplicates\n8.Searh Item\n9.Display\n10.exit\n");
```

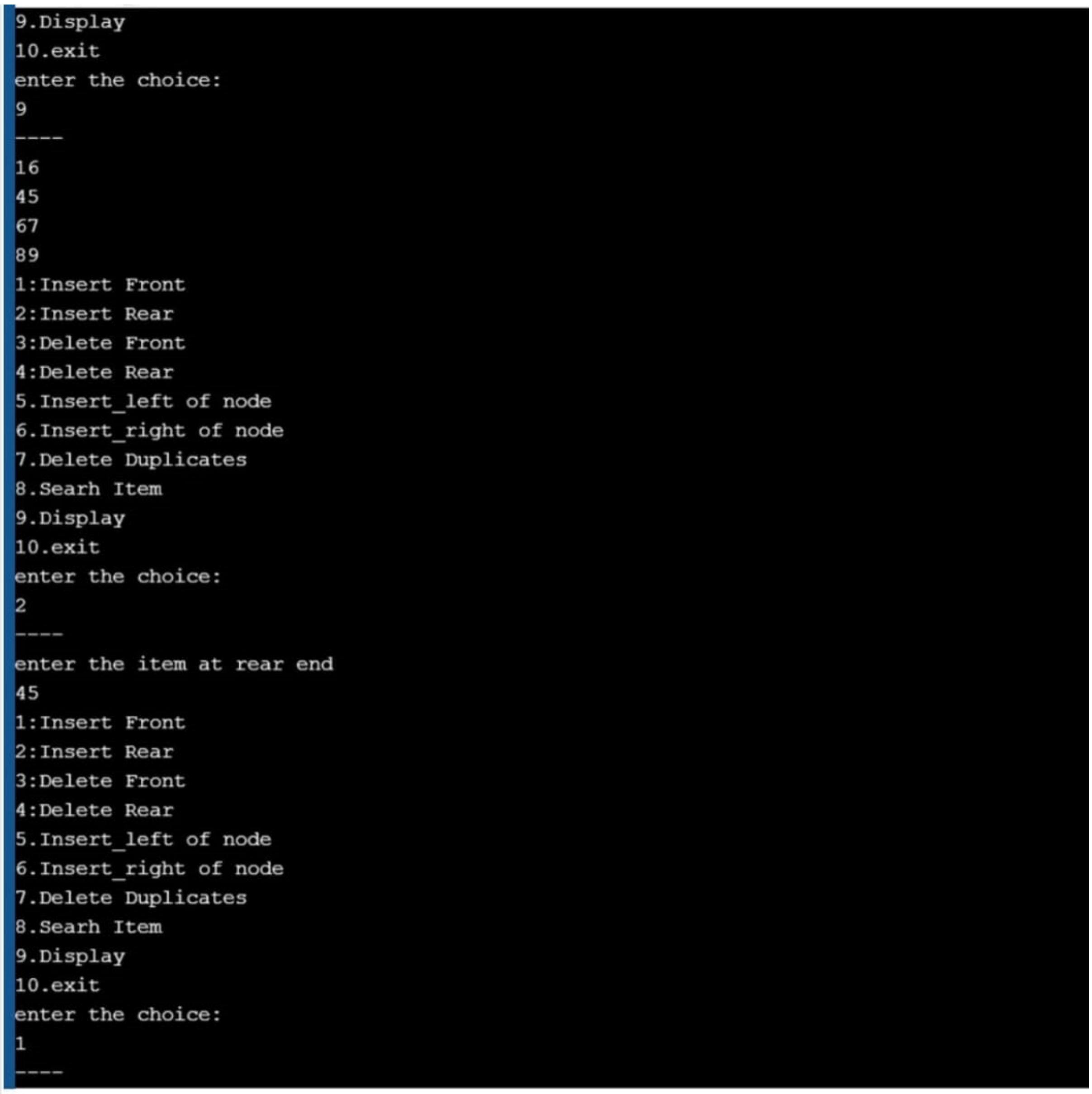
```
printf("enter the choice:\n");
scanf("%d",&choice);
printf("----\n");
switch(choice)
{
   case 1: printf("enter the item at front end\n");
         scanf("%d",&item);
                    last=dinsert_front(item,head);
                    break;
     case 2: printf("enter the item at rear end\n");
                    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 item\n");
                   scanf("%d",&item);
                   head=insert_leftpos(item,head);
                   break;
   case 6:printf("enter the key item\n");
                   scanf("%d",&item);
                   head=insert_rightpos(item,head);
                   break;
   case 7:printf("enter the key item\n");
                   scanf("%d",&item);
                   head=delete_all_key(item,head);
                   break;
   case 8:printf("enter the key item\n");
                   scanf("%d",&item);
```

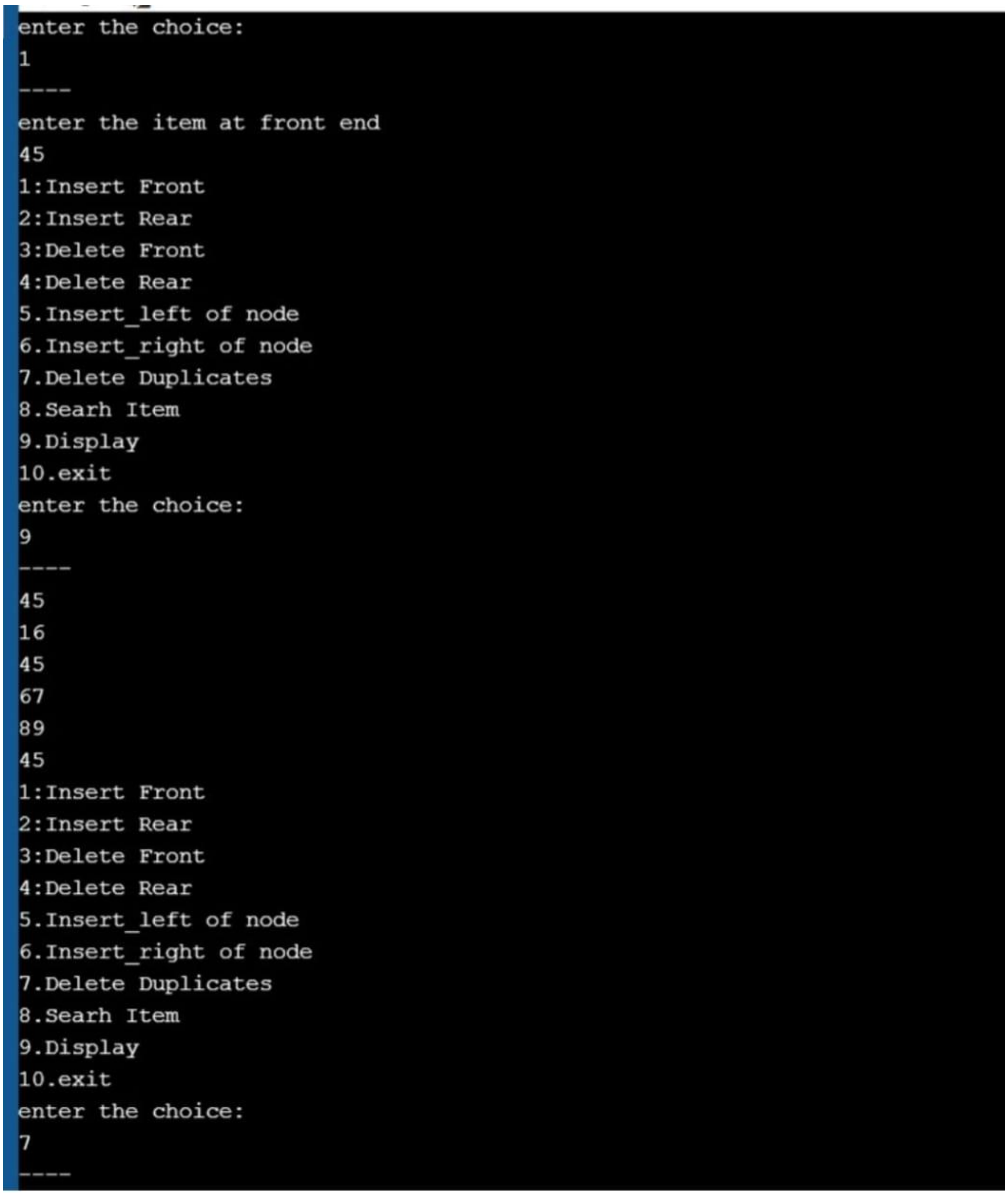
```
                    search_info(item,head);
                    break;
          case 9:display(head);
                    break;
          default:exit(0);
        }
      }
    }
```



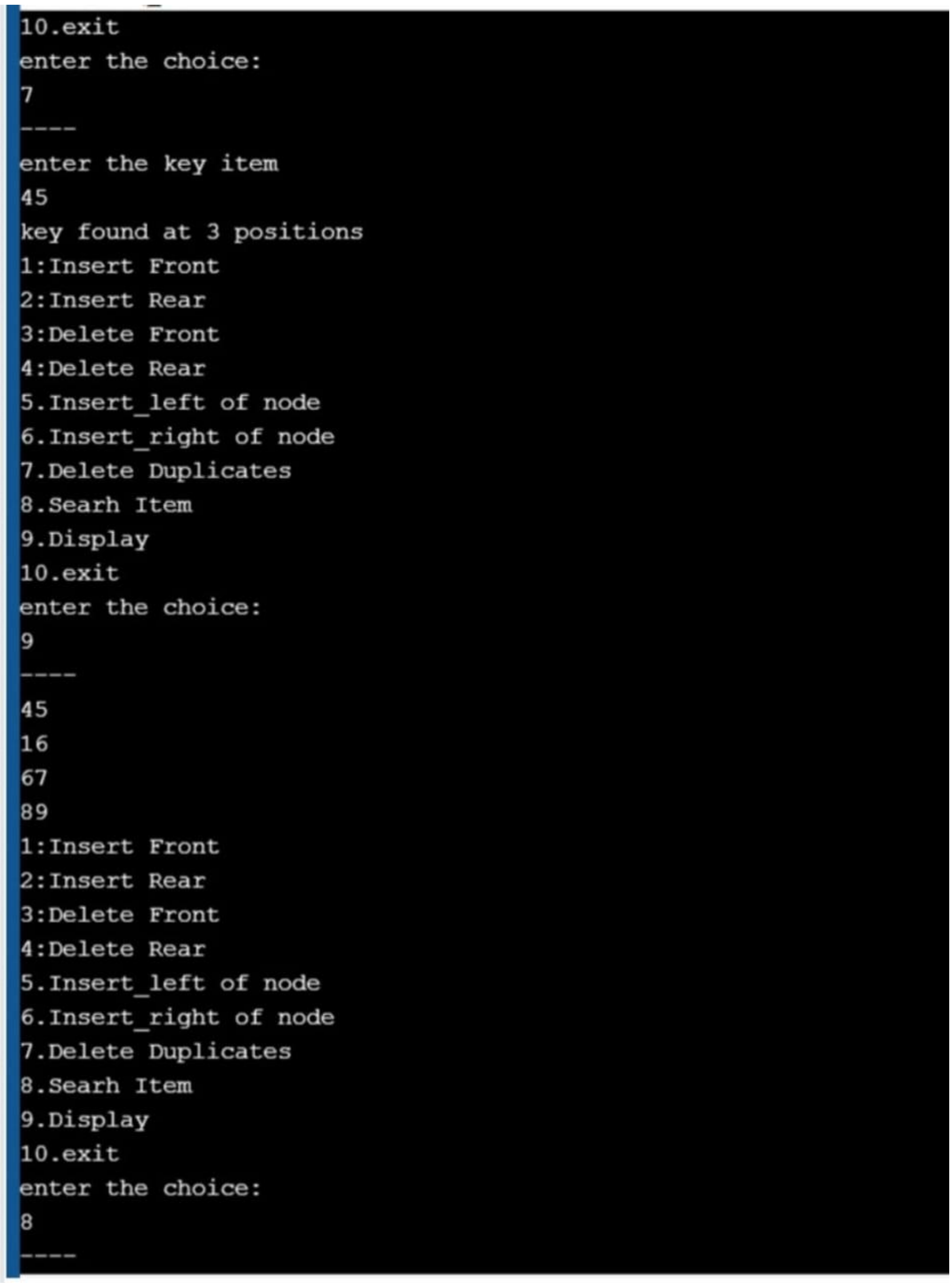
```
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
1
----
enter the item at front end
45
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
2
----
enter the item at rear end
67
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
```


```
enter the choice:
9
----
45
67
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
5
----
enter the key item
45
enter towards left of 45=16
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
6
----
enter the key item
67
enter towards right of 67=89
```

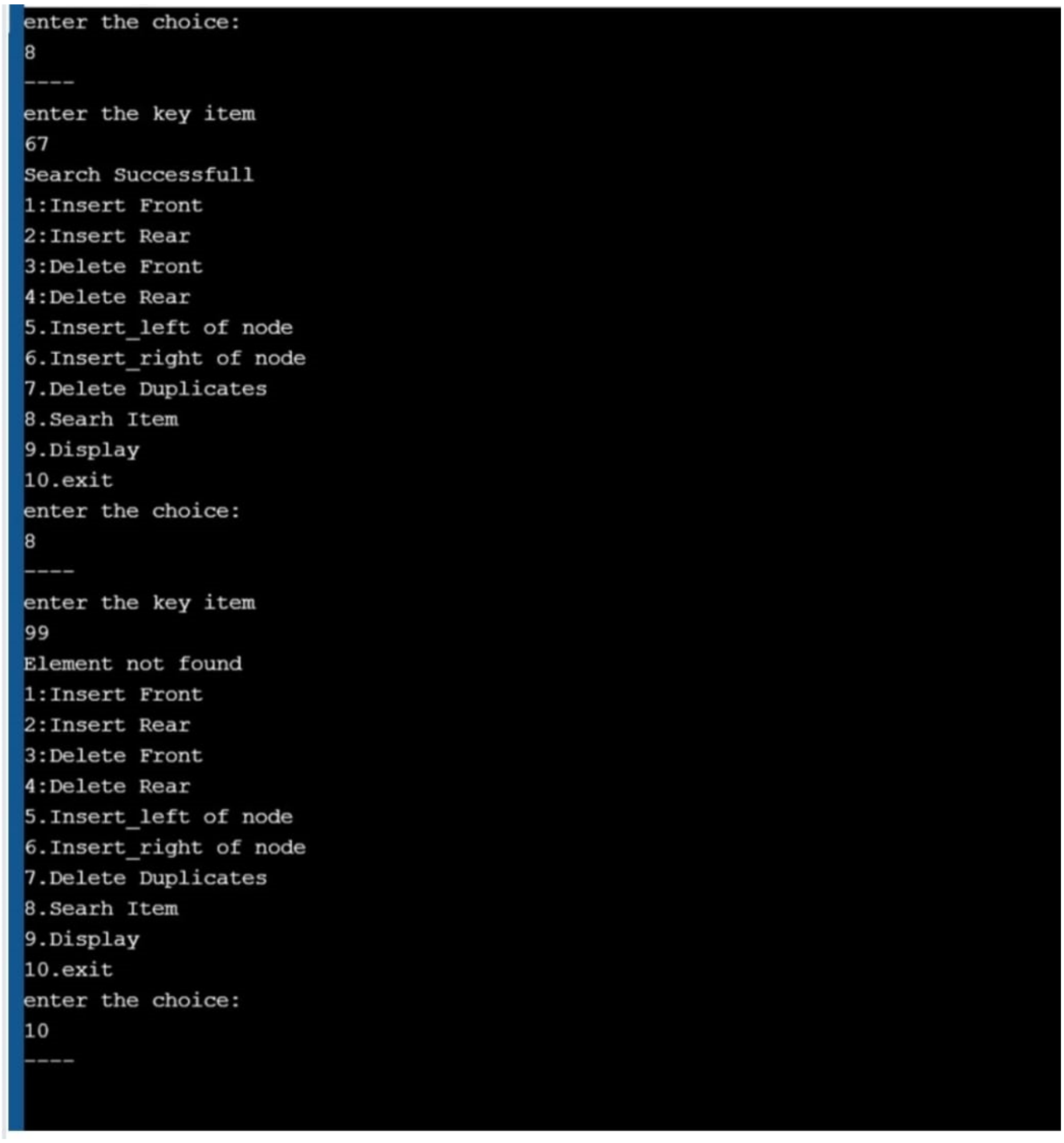

```
9.Display
10.exit
enter the choice:
9
----
16
45
67
89
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
2
----
enter the item at rear end
45
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
1
----
```



```
enter the choice:
1
----
enter the item at front end
45
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
9
----
45
16
45
67
89
45
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
7
----
```



```
10.exit
enter the choice:
7
----
enter the key item
45
key found at 3 positions
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
9
----
45
16
67
89
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
8
----
```



```
enter the choice:
8
----
enter the key item
67
Search Successfull
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
8
----
enter the key item
99
Element not found
1:Insert Front
2:Insert Rear
3:Delete Front
4:Delete Rear
5.Insert_left of node
6.Insert_right of node
7.Delete Duplicates
8.Searh Item
9.Display
10.exit
enter the choice:
10
----
```

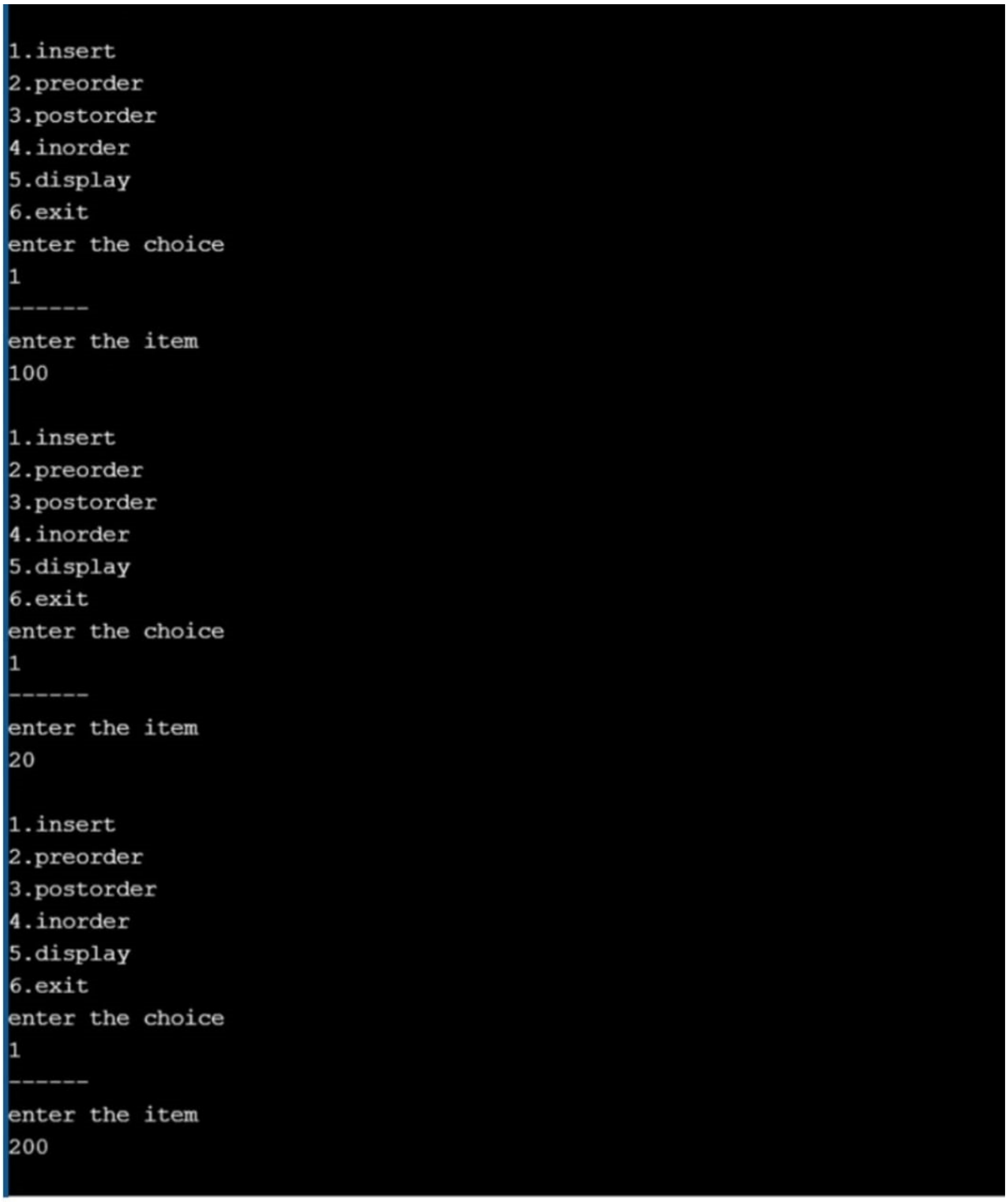
LAB PROGRAM 10

Write a program a) To construct a binary Search tree. b) To traverse the tree using all the methods i.e., in-order, preorder and post order c) To display the elements in the tree.

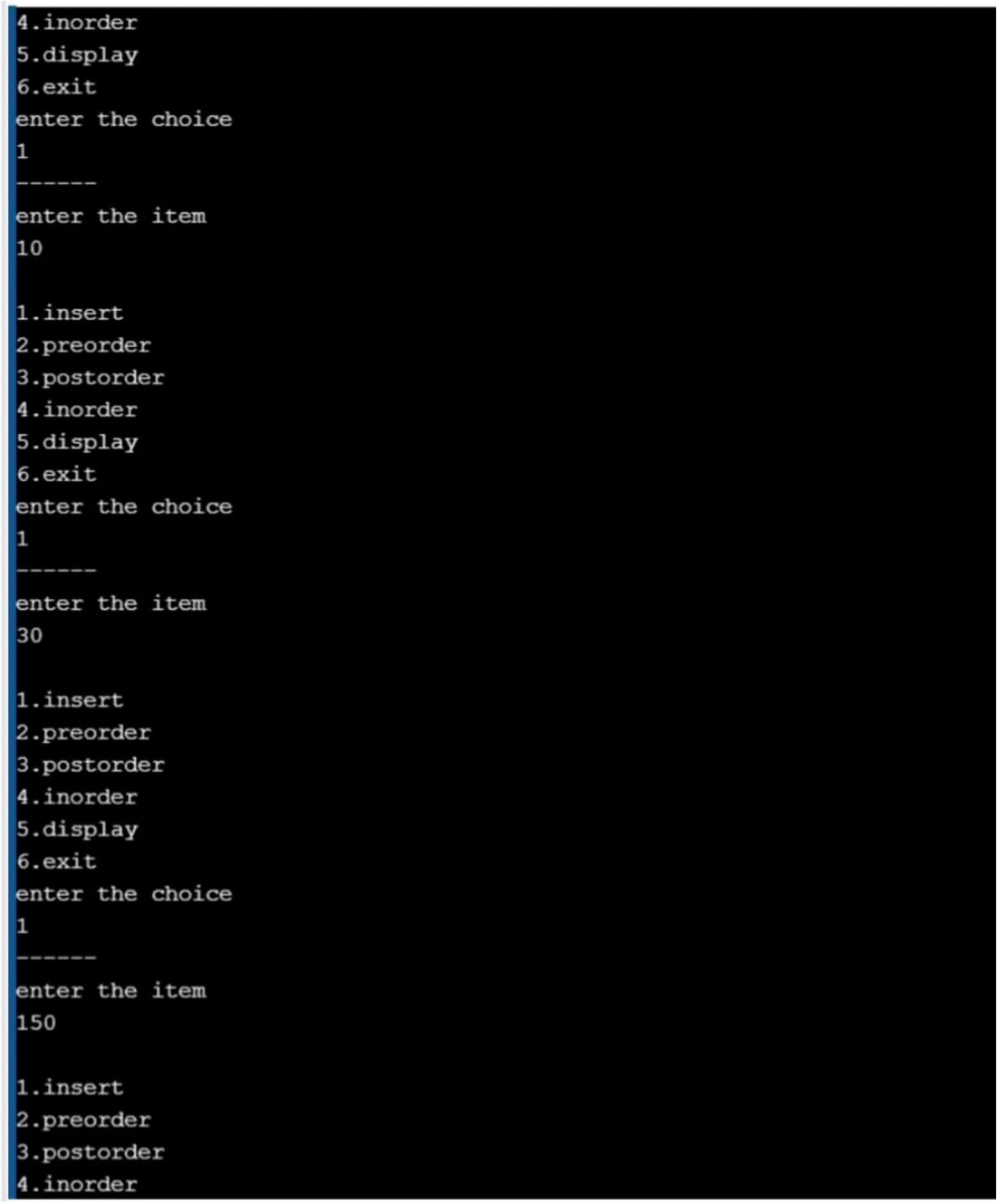
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{
  int info;
  struct node *rlink;
  struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
   NODE x;
   x=(NODE)malloc(sizeof(struct node));
   if(x==NULL)
    {
     printf("mem full\n");
     exit(0);
    }
    return x;
}
void freenode(NODE x)
{
   free(x);
}
NODE insert(NODE root,int item)
{
   NODE temp,cur,prev;
   temp=getnode();
   temp->rlink=NULL;
   temp->llink=NULL;
   temp->info=item;
   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\n",root->info);
                display (root->llink,i+1);
   }
}
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(;;)
    {
```

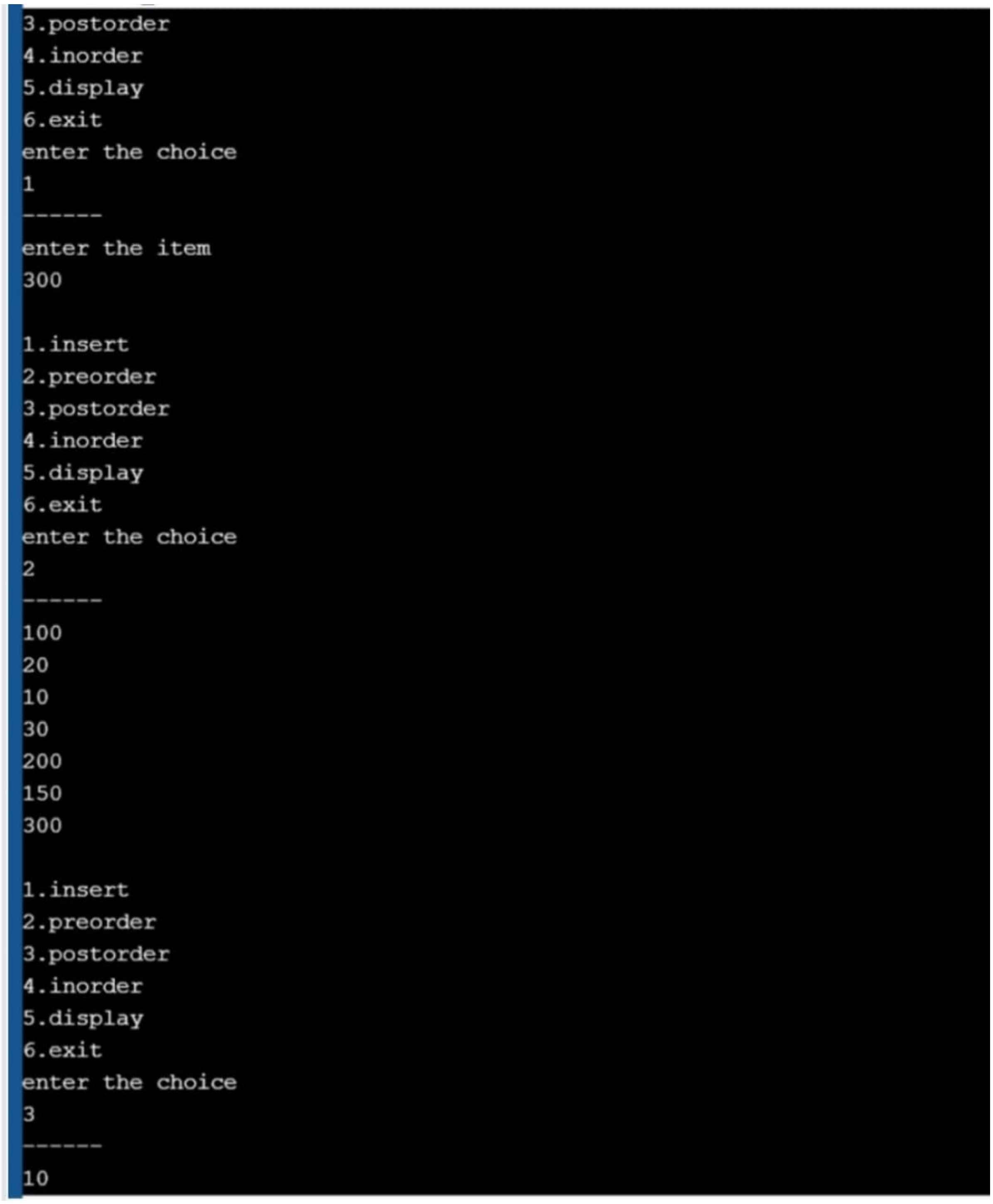
```
printf("\n1.insert\n2.display\n3.preorder\n4.postorder\n5.inorder\n6.exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

printf("------\n");

switch(choice)

{

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

                  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;

   default:exit(0);

                  break;

}

      }
```

```
}
```

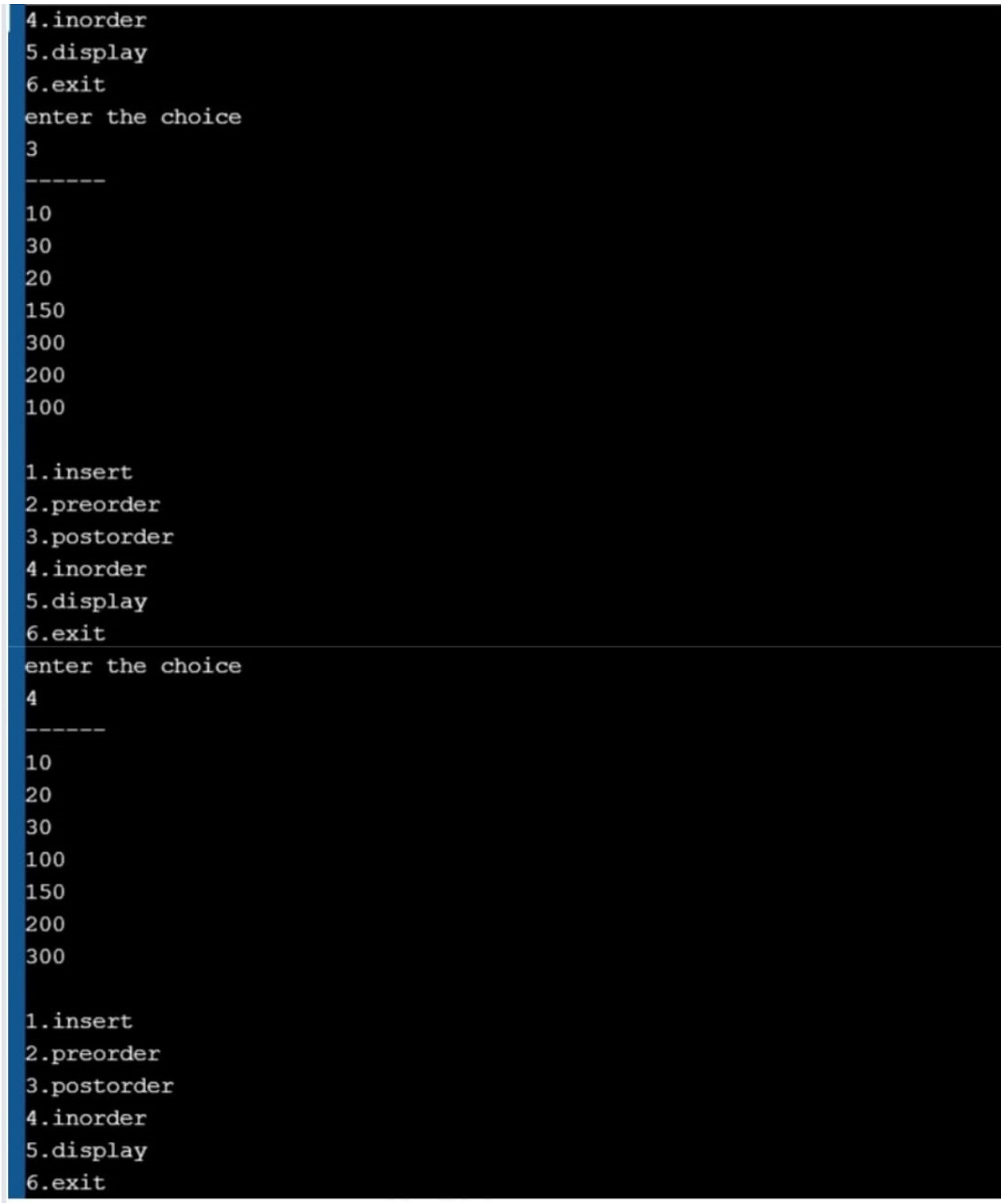


```
1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
100

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
20

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
200
```

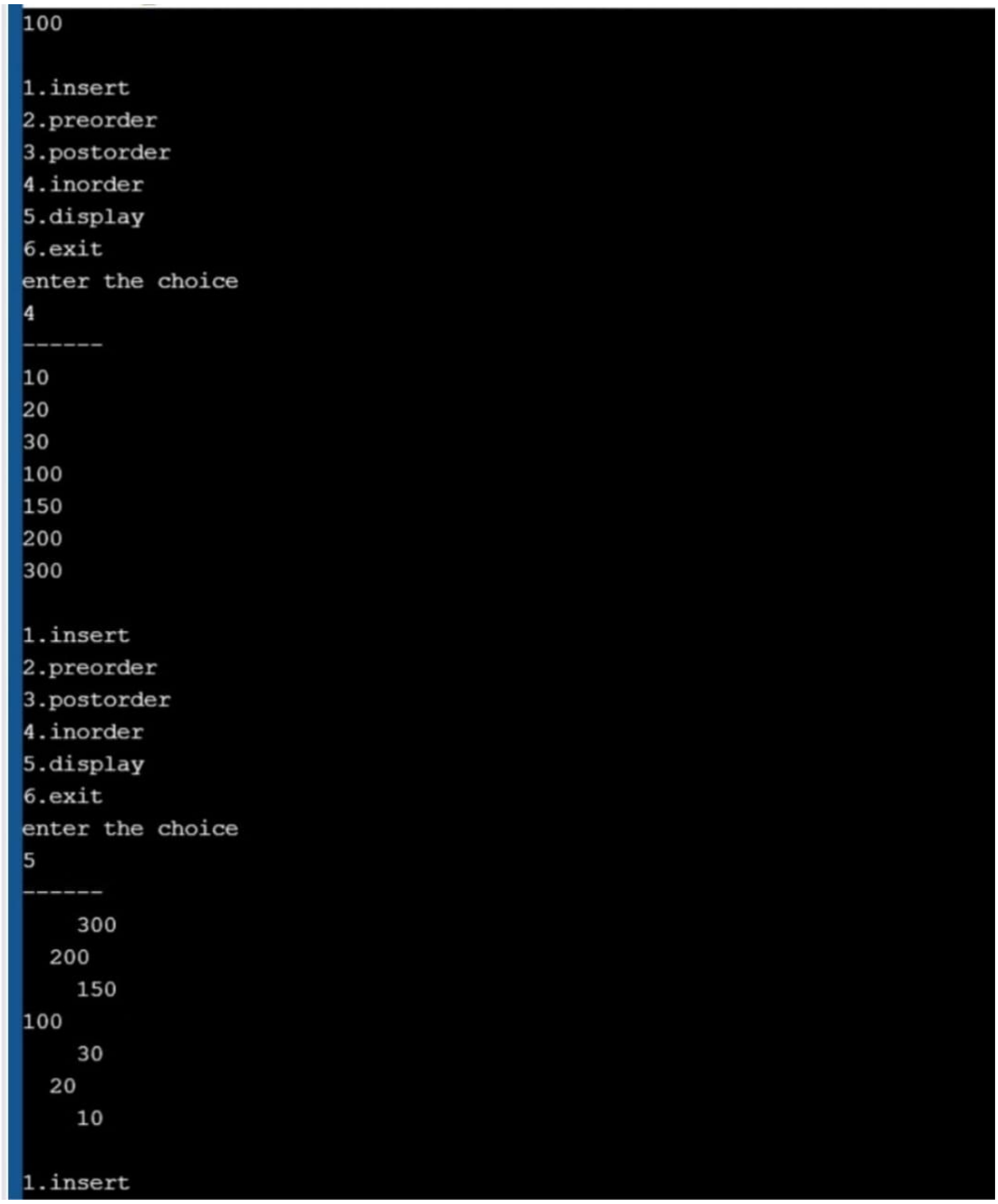

```
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
10

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
30

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
150

1.insert
2.preorder
3.postorder
4.inorder
```



```
3.postorder
4.inorder
5.display
6.exit
enter the choice
1
------
enter the item
300

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
2
------
100
20
10
30
200
150
300

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
3
------
10
```



```
4.inorder
5.display
6.exit
enter the choice
3
------
10
30
20
150
300
200
100

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
4
------
10
20
30
100
150
200
300

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



```
100

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
4
------
10
20
30
100
150
200
300

1.insert
2.preorder
3.postorder
4.inorder
5.display
6.exit
enter the choice
5
------
    300
  200
    150
100
    30
  20
    10

1.insert
```