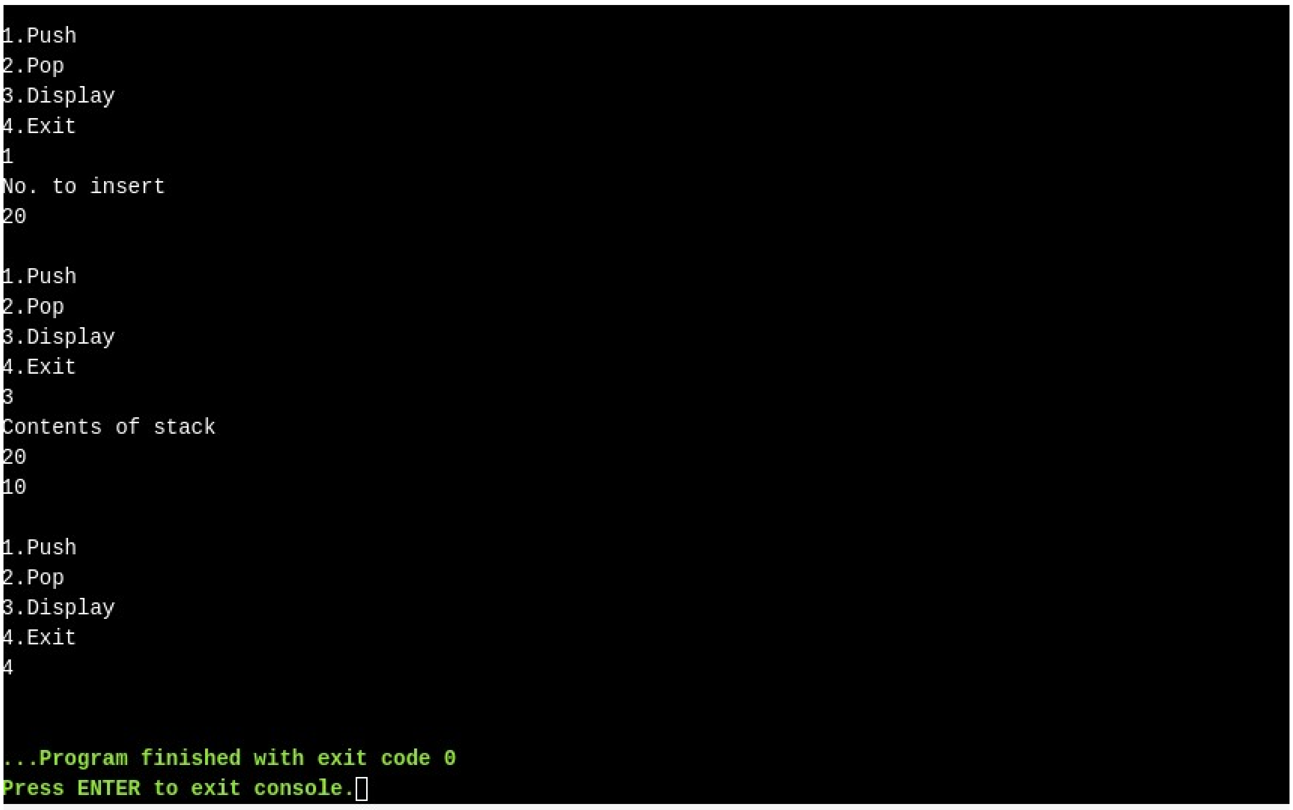
1. Simple Stack Implementation

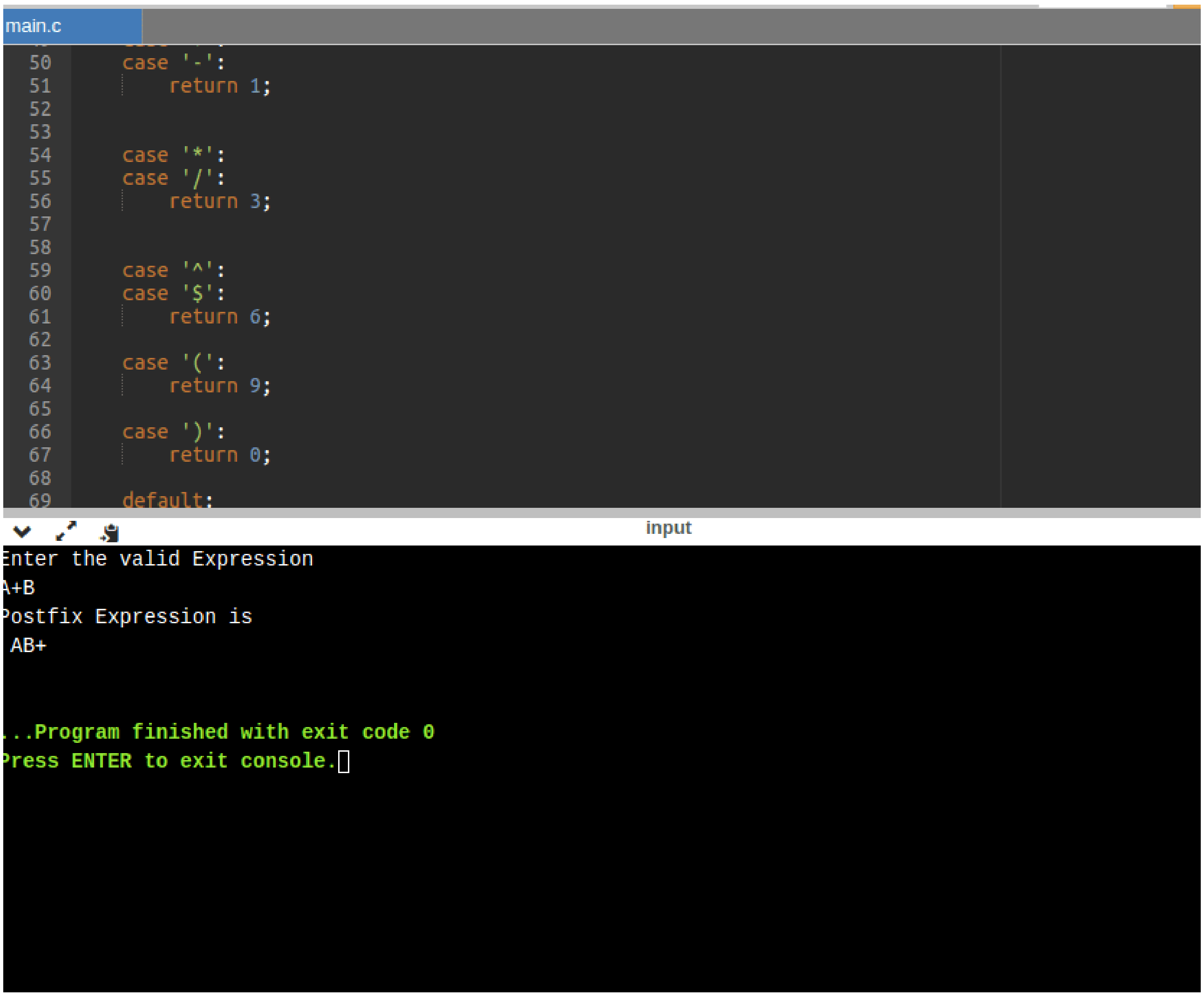
```
#include<stdio.h>
                    #include<conio.h>
                    #define STACK_SIZE 5
                    int top=-1;
                    int s[10];
                    int item;
                    void push()
                    {
                    if(top==STACK_SIZE-1){
                    printf("Overflow\n");
                    return;}
                    top=top+1;
                    s[top]=item;}
                    int pop(){
                    if(top==-1)
                    reutrn -1;
                    return s[top--];}
                    void display(){
                    int i;
                    if(top==-1){
                    printf("Empty\n");
                    return;}
                    printf("Contents of stack\n");
                    for(i=top;i>=0;i--){
                    printf("%d\n",s[i]);}}
                    void main(){
                    int it_d,ch;
                    for(;;){
                    printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
                    scanf("%d",&ch);
                    switch(ch){
                    case 1: printf("No. to insert \n");
                    scanf("%d",&item);
                    push();
                    break;
                    case 2: it_d=pop();
                    if(it_d==-1)
                    printf("empty\n");
                    else
                    printf("Deleted\n");
                    break;
                    case 3: display();
                    break;
                    default: exit(0);
                    }}
                    getch();}
```



```
1.Push
2.Pop
3.Display
4.Exit
1
No. to insert
20

1.Push
2.Pop
3.Display
4.Exit
3
Contents of stack
20
10

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


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

2. Infix to Postfix conversion

```
#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,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];
    printf("Enter the valid Expression\n");
    scanf(" %s",infix);
    infix_postfix(infix,postfix);
    fflush(stdin);
    printf("Postfix Expression is\n");
    printf(" %s\n",postfix);

    return 0;

}
```

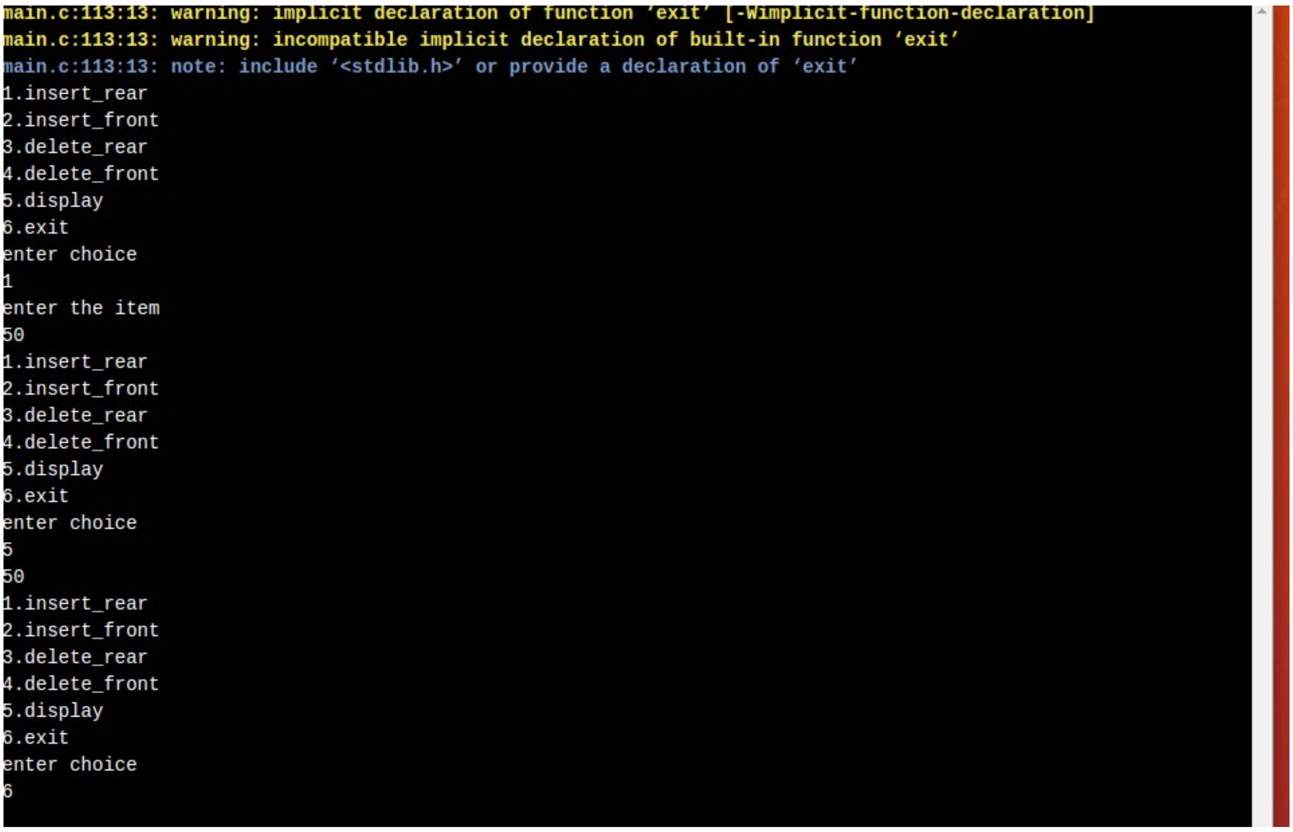

main.c

```
    case '-':
        return 1;


    case '*':
    case '/':
        return 3;


    case '^':
    case '$':
        return 6;

    case '(':
        return 9;

    case ')':
        return 0;

    default:
```

```
Enter the valid Expression
A+B
Postfix Expression is
 AB+


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

3. Simple Queue

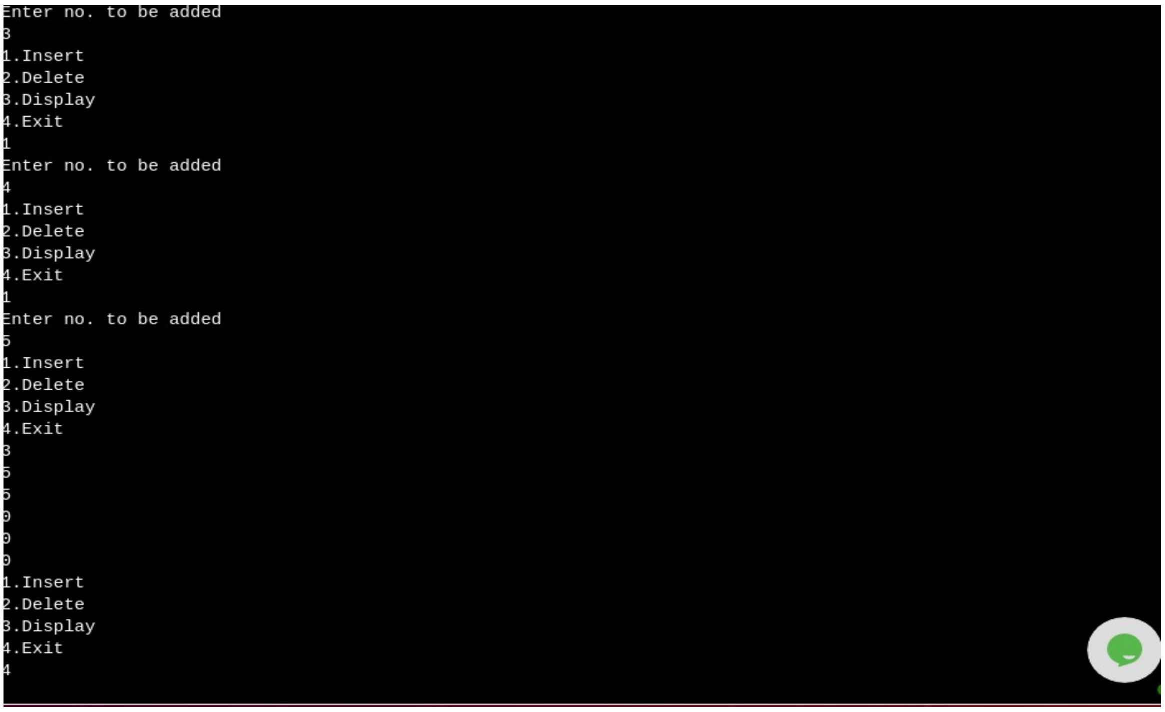
```
#include<stdio.h>#include<stdlib.h>
#define QUE_SIZE 5
int item,front=0,rear=-1,q[10];
void insertrear(){
    if(rear==QUE_SIZE-1){
        printf("queue overflow\n");
        return;}
    printf("Enter the ITEM to be Inserted\n");
    scanf("%d",&item);
    rear=rear+1;
    q[rear]=item;}
void deletefront(){
    if (front>rear){
        front=0;
        rear=-1;
        printf("QUEUE is EMPTY\n");}
    printf("The item deleted is %d",q[front++]);}
void display(){
    int i;if (front>rear){
        printf("Queue is Empty\n");}
        else{
            printf("Contents of queue\n");
            for(i=front;i<=rear;i++){
                printf("%d\t",q[i]);}}}
int main(){
    int choice;
    for(;;)     {
        printf("\n Enter the choice of Operation \n 1:Insert_Rear \
n2:Delete_Front \n3:display\n4:Exit\n");
        printf("enter the choice\n");
        scanf("%d",&choice);
        switch(choice){
            case 1:insertrear();break;
            case 2:deletefront();break;
            case 3:display();break;
            default:exit (0);
                }}}
```



```
main.c:113:13: warning: implicit declaration of function ‘exit’ [-Wimplicit-function-declaration]
main.c:113:13: warning: incompatible implicit declaration of built-in function ‘exit’
main.c:113:13: note: include ‘<stdlib.h>’ or provide a declaration of ‘exit’
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
1
enter the item
50
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
5
50
1.insert_rear
2.insert_front
3.delete_rear
4.delete_front
5.display
6.exit
enter choice
6
```

4. Circular Queue

```
#include<stdio.h>
#include<stdlib.h>
#define que_size 5
int item,front=0,rear=-1,q[que_size],count=0;
void insertrear(){
    if(count==que_size){
        printf("queue overflow\n");
        return;}
    printf("enter the element to be inserted\n");
    scanf("%d",&item);
    rear=(rear+1)%que_size;
    q[rear]=item;
    count++;}void deletefront(){
        if(count==0){
            printf("Queue is EMPTY\n");}
        else{
            item = q[front];
            printf("The ITEM deleted is = %d \n",item);
            front=(front+1)%que_size;
            count=count-1;}}
void displayq(){
    int i,f;
    if(count==0){
        printf("Queue is Empty\n");
        return;}
    f=front;
    printf("ITEMS of Queue \n");
    for(i=0;i<=count;i++){
        printf("%d\t",q[f]);
        f=(f+1)%que_size;}}
int main(){
    int choice;
    for(;;){
        printf("\n Enter the choice of Operation \n1.INSERT REAR \n2.DELETE
FRONT \n3.DISPLAY \n4.EXIT \n ");
        scanf("%d",&choice);
        switch(choice){
            case 1:insertrear();break;
            case 2:deletefront();break;
            case 3:displayq();break;
           default:exit(0);}}}
```



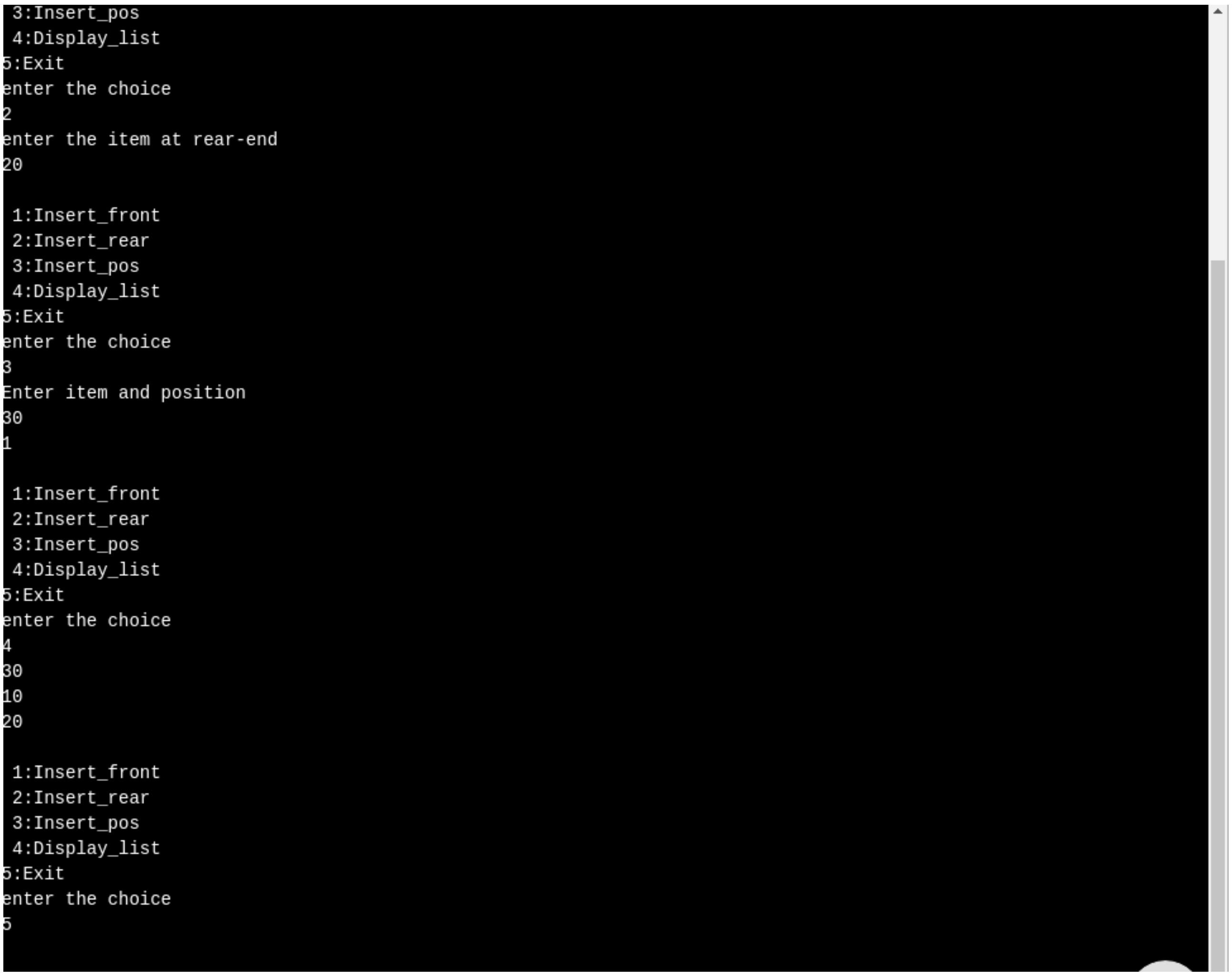
```
Enter no. to be added
3
1.Insert
2.Delete
3.Display
4.Exit
1
Enter no. to be added
4
1.Insert
2.Delete
3.Display
4.Exit
1
Enter no. to be added
5
1.Insert
2.Delete
3.Display
4.Exit
3
5
5
0
0
0
1.Insert
2.Delete
3.Display
4.Exit
4
```

5. Simple Singly Linked List(Insert)

```
#include<stdio.h>
#include<conio.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 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 insert_pos( int item, int pos, NODE first)
{
NODE temp;
NODE prev,cur;
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 first;
```

```
}
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:Insert_rear\n 3:Insert_pos\n 4:Display_list\
n5:Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1:printf("enter the item at front-end\n");
scanf("%d",&item);
first=insert_front(first,item);
break;
case 2:printf("enter the item at rear-end\n");
scanf("%d",&item);
first=insert_rear(first,item);
break;
case 3:printf("Enter item and position\n");
scanf("%d%d",&item,&pos);
first=insert_pos(item,pos,first);
break;
case 4:display(first);
break;
default:exit(0);
break;
}
}
getch();
}
```



```
 3:Insert_pos
 4:Display_list
5:Exit
enter the choice
2
enter the item at rear-end
20

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Display_list
5:Exit
enter the choice
3
Enter item and position
30
1

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Display_list
5:Exit
enter the choice
4
30
10
20

 1:Insert_front
 2:Insert_rear
 3:Insert_pos
 4:Display_list
5:Exit
enter the choice
5
```

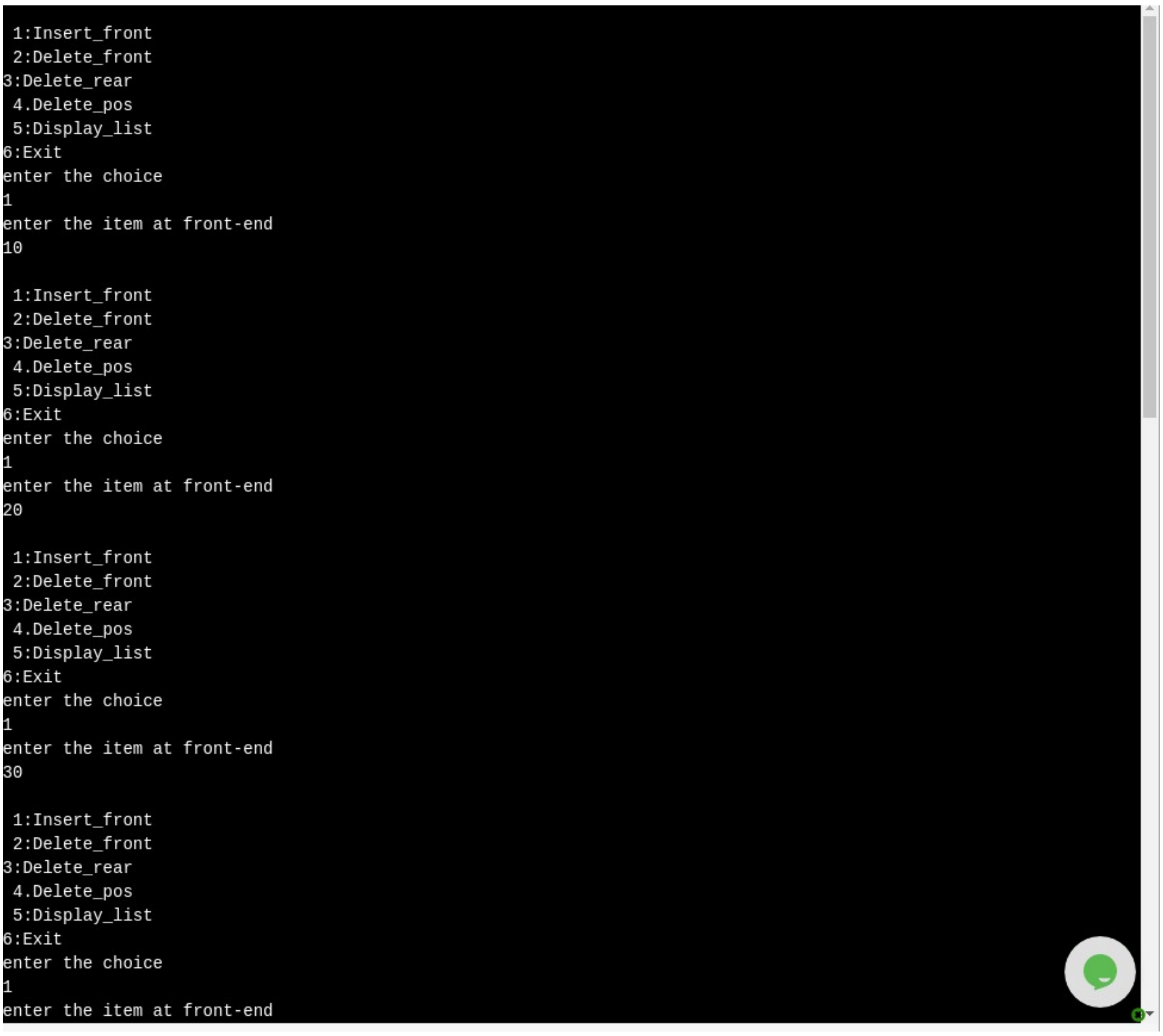
6. Singly Linked list(Delete)

```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node{

int info;

struct node *link;};

typedef struct node *NODE;

NODE getnode(){

NODE x;
```

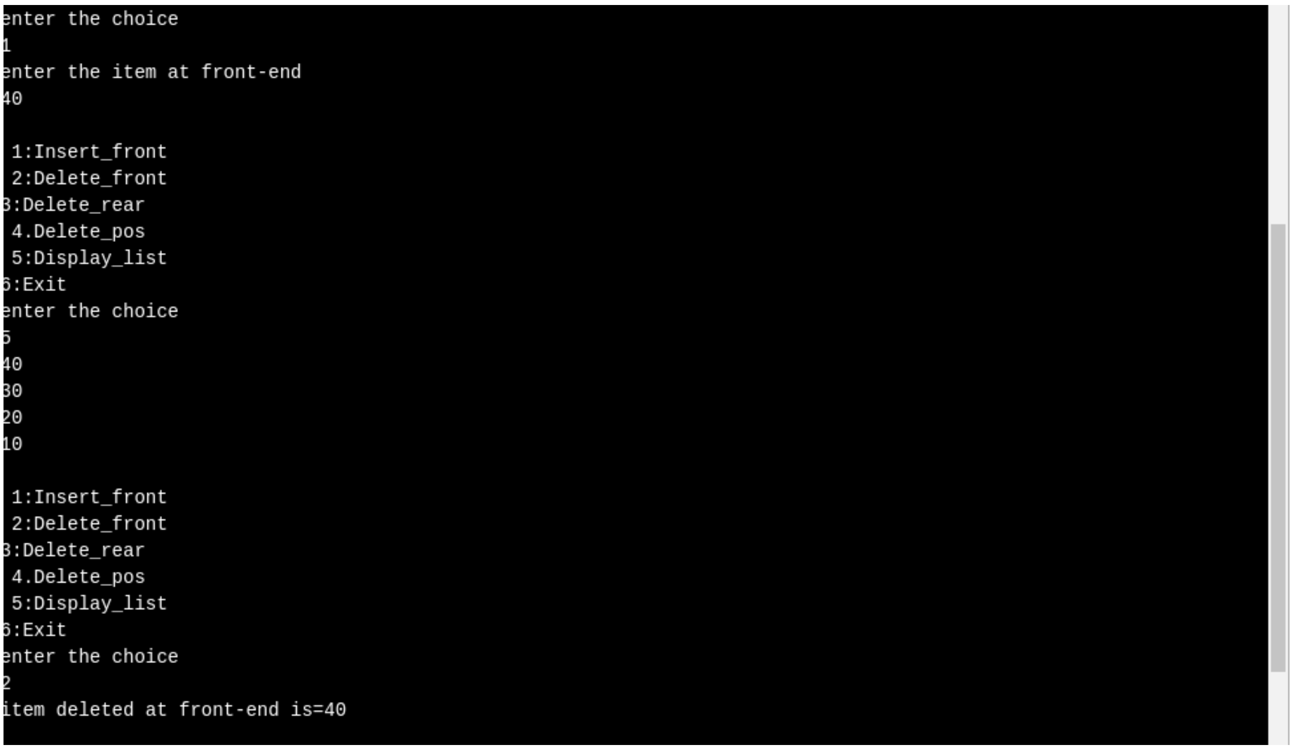
```
x=(NODE)malloc(sizeof(struct node));
if(x==NULL){
printf("mem full\n");
exit(0);}
return x;}
void freenode(NODE x){
free(x);}
NODE insert_front(NODE first,int item){
NODE temp;
temp=getnode();
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;}
NODE delete_front(NODE first){
NODE temp;
if(first==NULL){
printf("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 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("iten deleted at rear-end is %d",cur->info);
free(cur);
prev->link=NULL;
return first;}
NODE delete_pos(int pos, NODE first){
NODE cur;
NODE prev;
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;
```

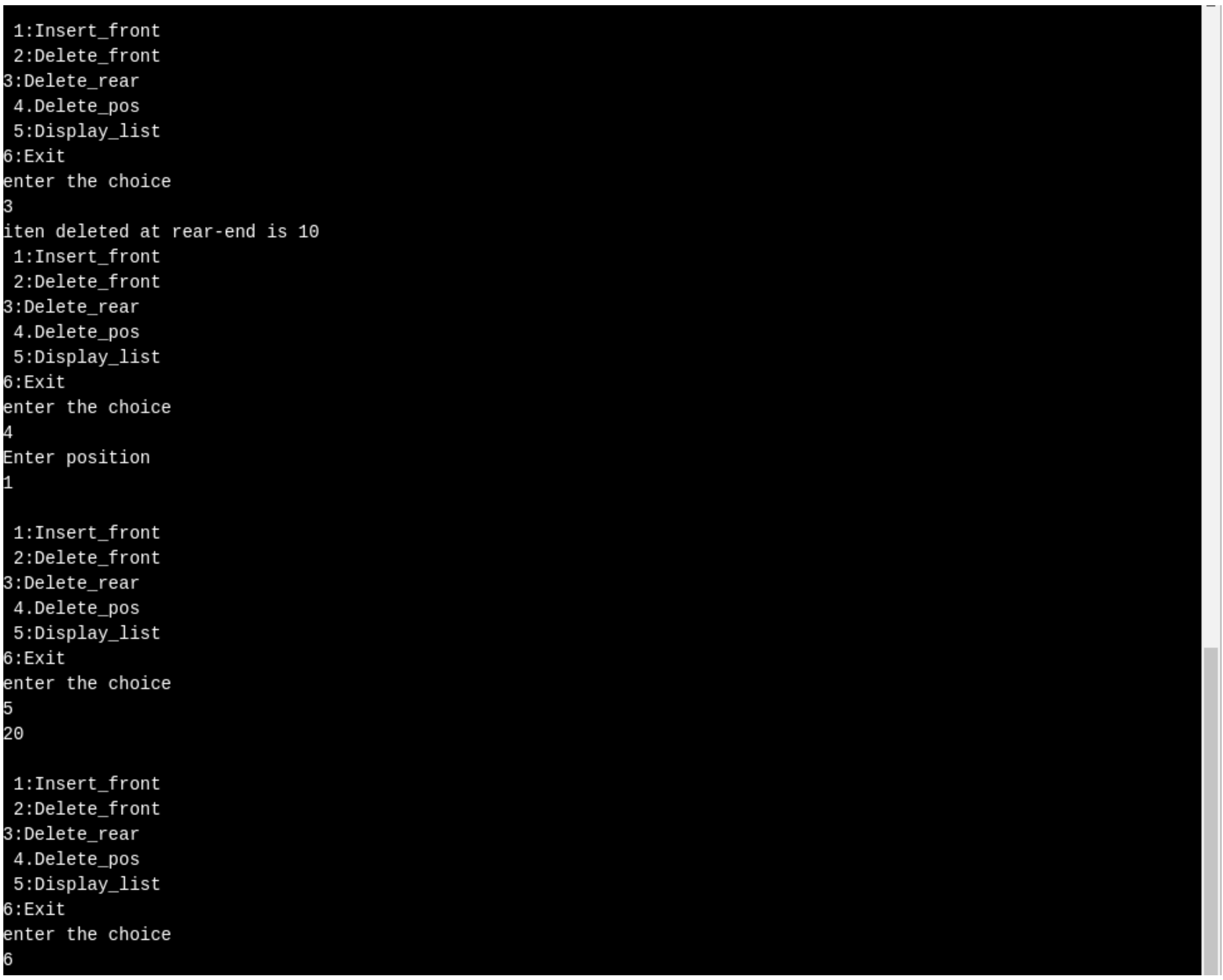
```
prev=cur;
cur=cur->link;
count++;}
if(count!=pos) {
printf("invalid position\n");
return first;}
if(count!=pos){
printf("invalid position specified\n");
return first;}
prev->link=cur->link;
freenode(cur);
return first;}
void display(NODE first){
NODE temp;
if(first==NULL)
printf("list empty cannot 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\n3:Delete_rear\n 4.Delete_pos\n 5:Display_list\n6:Exit\
n");
printf("enter the choice\n");
scanf("%d",&choice);
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:first=delete_rear(first);
break;
case 4:printf("Enter position\n");
scanf("%d",&pos);
first=delete_pos(pos,first);
break;
case 5:display(first);
break;
default:exit(0);
break;}}
getch();}
```



```
 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
1
enter the item at front-end
10

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
1
enter the item at front-end
20

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
1
enter the item at front-end
30

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
1
enter the item at front-end
```



```
enter the choice
1
enter the item at front-end
40

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
5
40
30
20
10

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
2
item deleted at front-end is=40
```

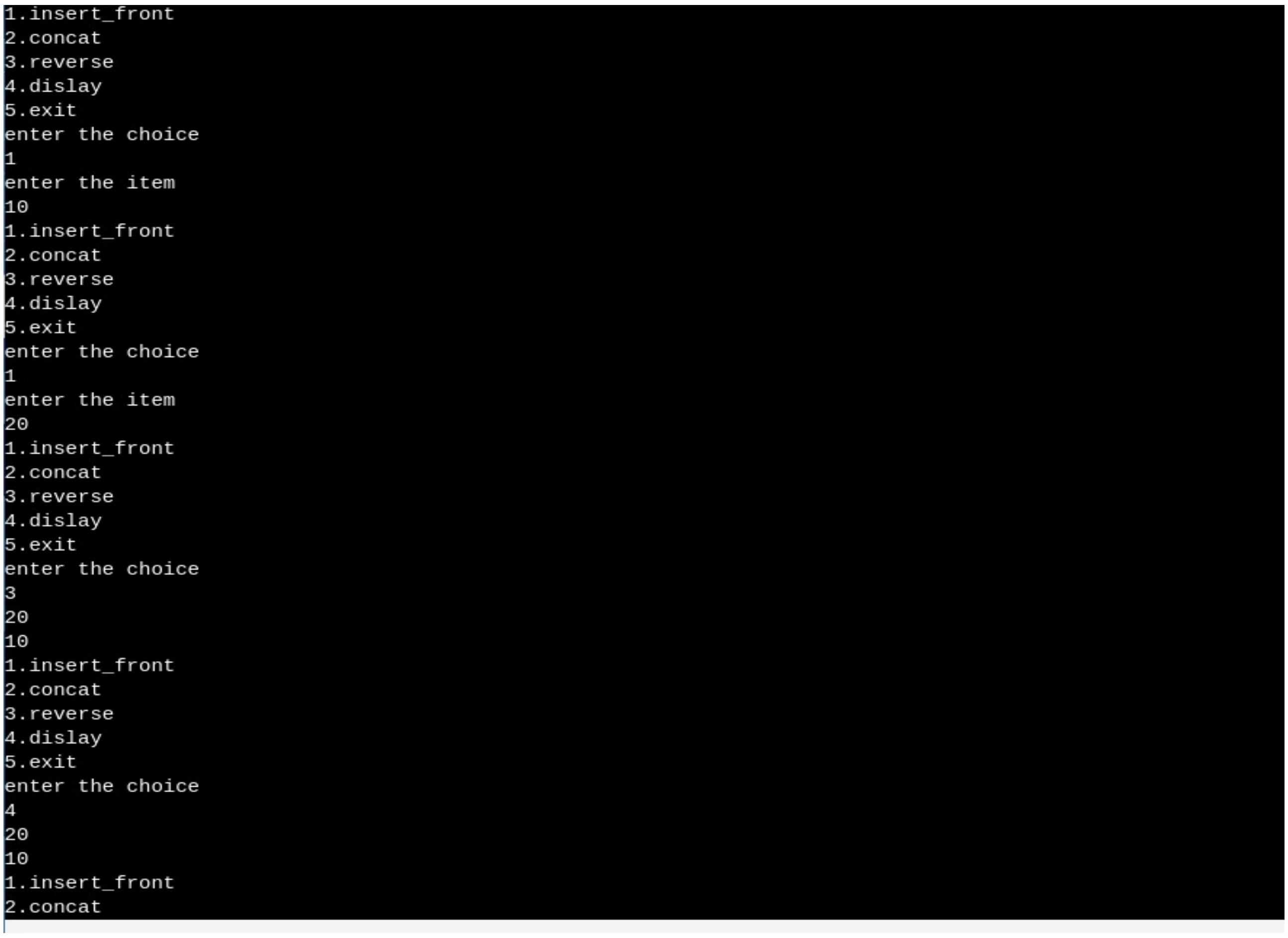


```
 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
3
iten deleted at rear-end is 10
 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
4
Enter position
1

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
5
20

 1:Insert_front
 2:Delete_front
3:Delete_rear
 4.Delete_pos
 5:Display_list
6:Exit
enter the choice
6
```

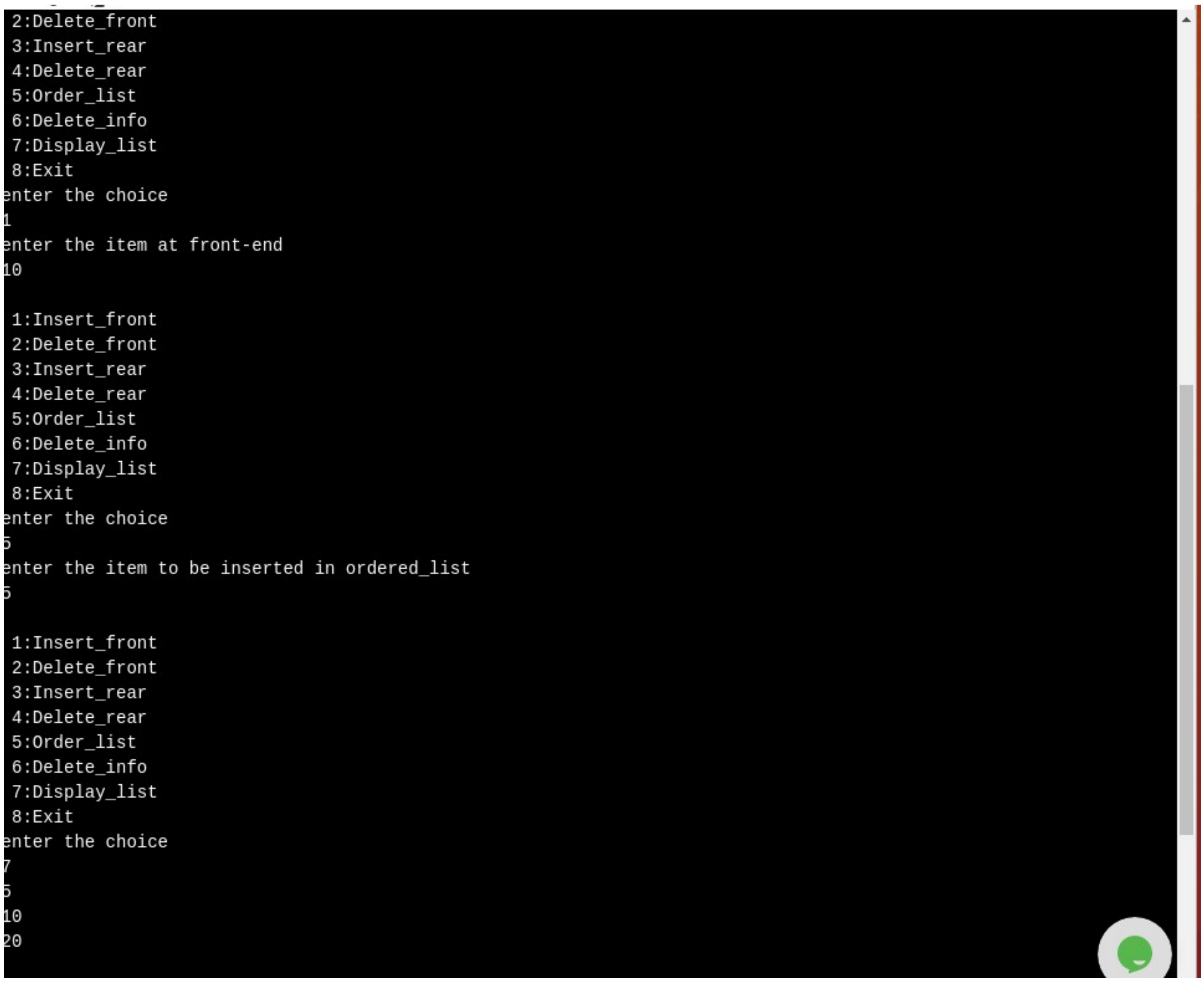
7. Linked List(Sort,Reverse,concat)

```
#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
struct node{
int item;
struct node *next;};
typedef struct node *Node;
Node getNode(){
Node x;
x=(Node)malloc(sizeof(struct node));
return x;}
Node insert_front(Node first,int data){
Node new_node;
new_node=getNode();
new_node->item=data;
new_node->next=NULL;
if(first==NULL){
return new_node;}
new_node->next=first;
first=new_node;
return first;}
Node delete_end(Node first){
Node prev,cur;
if(first==NULL){
printf("List Is Empty And Cannot be deleted\n");
return first;}
cur=first;
while(cur->next!=NULL){
prev=cur;
cur=cur->next;}
prev->next=NULL;
free(cur);
return first;}
void display(Node first){
int count=0;
Node temp;
if(first==NULL){
printf("List Is Empty\n");}
for(temp=first;temp!=NULL;temp=temp->next){
count++;
printf("%d\n",temp->item);}
printf("Number of Nodes In The List Are %d\n",count);}
NODE concat(NODE first,NODE second)
20{
NODE cur;
if(first==NULL)
return second;
if(second==NULL)
return first;
cur=first;
while(cur->link!=NULL)
cur=cur->link;
cur->link=second;
return first;}
NODE reverse(NODE first){
NODE cur,temp;
cur=NULL;
while(first!=NULL){
```

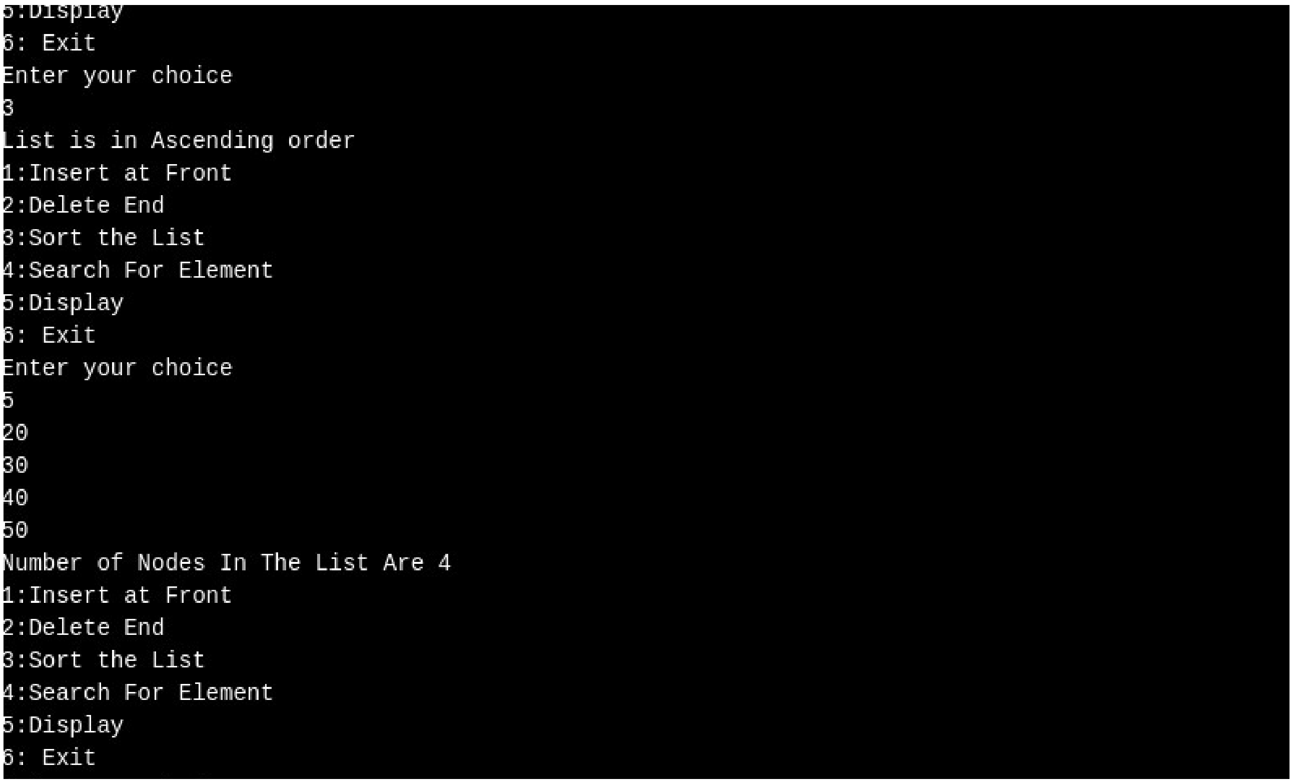
```
temp=first;
first=first->link;
temp->link=cur;
cur=temp;}
return cur;}
void sort(Node first){
int t;Node temp;
if(first==NULL){
printf("List Is Empty\n");
return;}
for(Node i=first;i!=NULL;i=i->next){
for(Node j=i->next;j!=NULL;j=j->next){
if((i->item)>(j->item)){
t=i->item;
i->item=j->item;
j->item=t;}}}
printf("List is in Ascending order\n");}
int main(){
Node first=NULL;
Node a= NULL;Node b=NULL;Node ans=NULL;
int choice,val,pos,n;
do{
printf("1:Insert at Front\n");
printf("2:Delete End\n");
printf("3:Sort the List\n");
printf("4:Search For Element\n");
printf("5:Display\n");
printf("6: Exit\n");
printf("Enter your choice\n");
scanf("%d",&choice);
switch(choice){
case 1:printf("Enter the Value To Be Inserted\n");
scanf("%d",&val);
first=insert_front(first,val);
break;
case 2:
first=delete_end(first);
break;
case 3:sort(first);
break;
case 4:printf("Enter the Number To Searched\n");
scanf("%d",&n);
search(first,n);
break;
case 5:display(first);
break;}}while(choice!=6);}
```



```
1.insert_front
2.concat
3.reverse
4.dislay
5.exit
enter the choice
1
enter the item
10
1.insert_front
2.concat
3.reverse
4.dislay
5.exit
enter the choice
1
enter the item
20
1.insert_front
2.concat
3.reverse
4.dislay
5.exit
enter the choice
3
20
10
1.insert_front
2.concat
3.reverse
4.dislay
5.exit
enter the choice
4
20
10
1.insert_front
2.concat
```



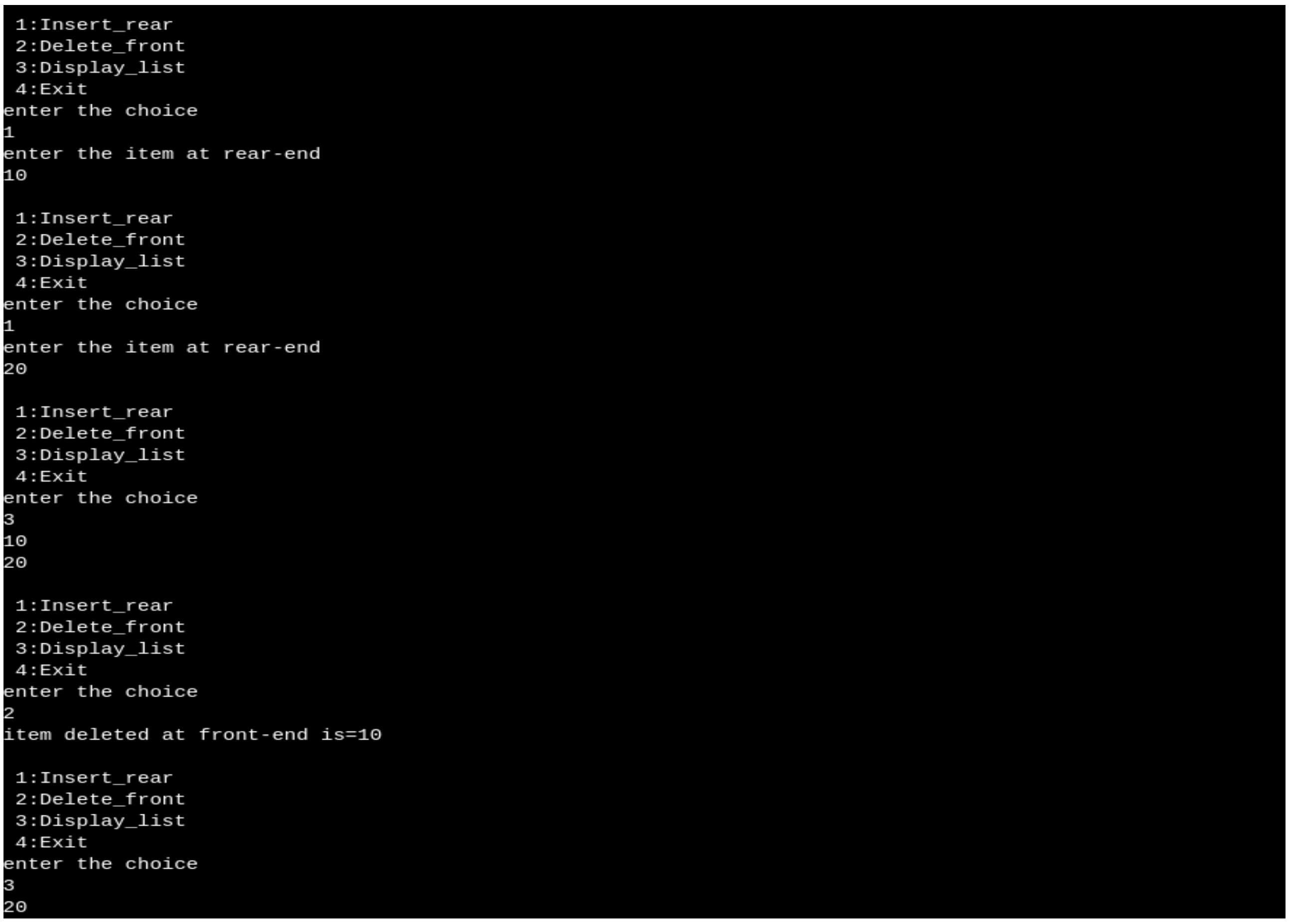
```
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:Delete_info
 7:Display_list
 8:Exit
nter the choice
1
nter the item at front-end
10

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:Delete_info
 7:Display_list
 8:Exit
nter the choice
5
nter the item to be inserted in ordered_list
5

 1:Insert_front
 2:Delete_front
 3:Insert_rear
 4:Delete_rear
 5:Order_list
 6:Delete_info
 7:Display_list
 8:Exit
nter the choice
7
5
10
20
```


```
5:Display
6: Exit
Enter your choice
3
List is in Ascending order
1:Insert at Front
2:Delete End
3:Sort the List
4:Search For Element
5:Display
6: Exit
Enter your choice
5
20
30
40
50
Number of Nodes In The List Are 4
1:Insert at Front
2:Delete End
3:Sort the List
4:Search For Element
5:Display
6: Exit
```

8. Stacks and queue using Linked List

```
#include<stdio.h>
#include<conio.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);}}
void 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);
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;}}
getch();}
```



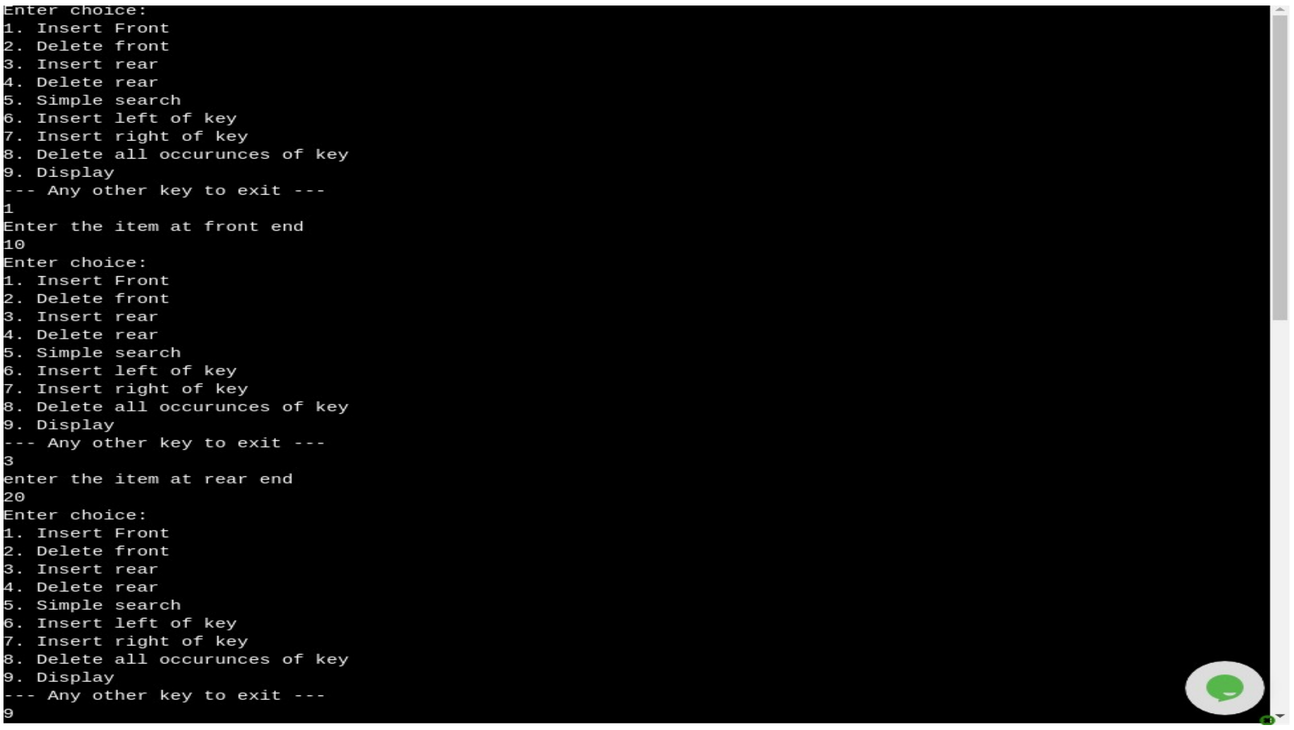
```
 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
10

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
1
enter the item at rear-end
20

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
10
20

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=10

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
20
```

9. Doubly Linked List

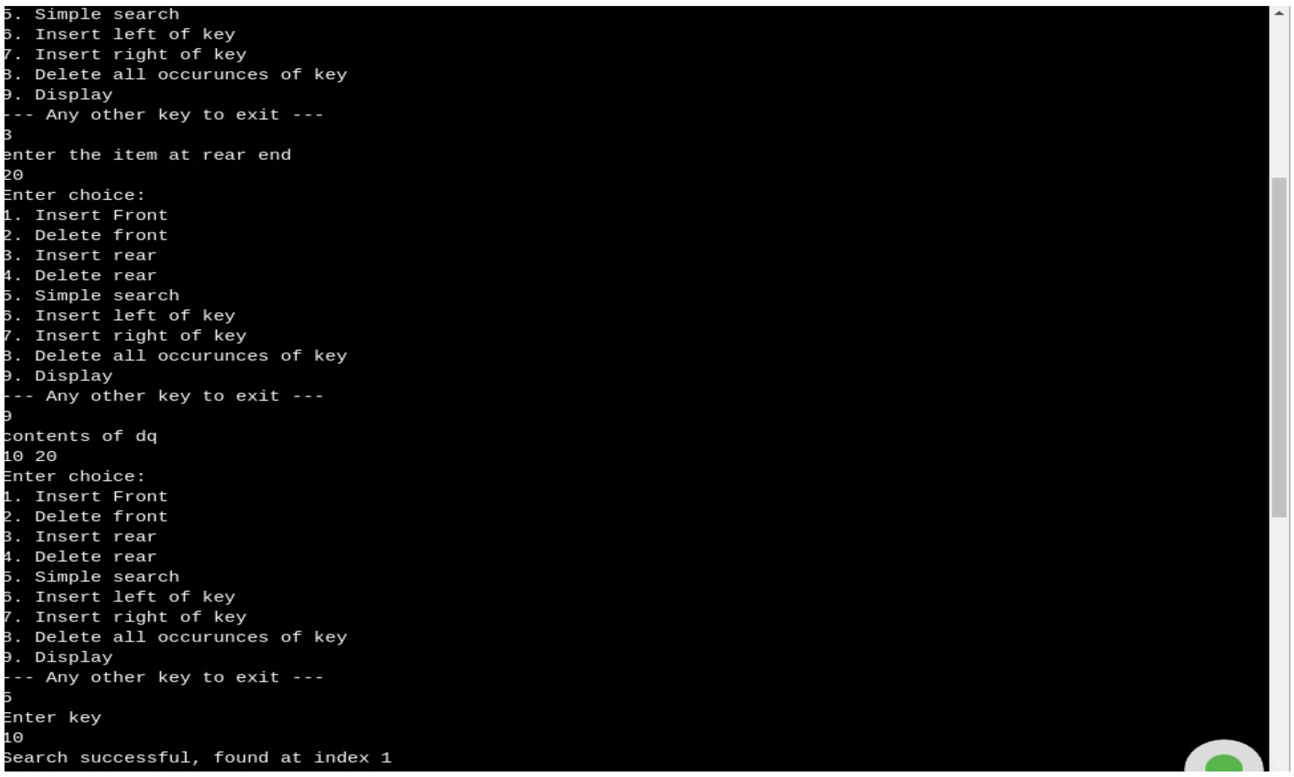
```
#include <stdio.h>
#include <stdlib.h>

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("Memory full\n");
 exit(0);
 }
 return x;
}
NODE dinsert_rear(int item, NODE head){
 NODE temp,cur;
 temp=getnode();
 temp->info=item;
 cur=head->llink;
 temp->llink=cur;
 cur->rlink=temp;
 head->llink=temp;
 temp->rlink=head;
 return head;
}
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 ddelete_front(NODE head)
{
 NODE cur,next;
 if(head->rlink==head)
 {
 printf("dq empty\n");
 return head;
 }
 cur=head->rlink;
 next=cur->rlink;
 head->rlink=next;
 next->llink=head;
 printf("the item deleted is %d\n",cur->info);
 free(cur);
 return head;
```

```
}
NODE ddelete_rear(NODE head)
 {
 NODE cur,prev;
 if(head->rlink==head)
 {
 printf("dq empty\n");
 return head;
 }
 cur=head->llink;
 prev=cur->llink;
 head->llink=prev;
 prev->rlink=head;
 printf("the item deleted is %d\n",cur->info);
 free(cur);
 return head;
}
NODE lsearch(NODE head, int key, int z){
 NODE cur,prev,temp;
 int f=0,c=1;
 if(head->rlink==head)
 {
 printf("list empty\n");
 return head;
 }
 cur=head->rlink;
 while(cur!=head)
 {
 if(cur->info==key){
 f=1;
 break;
 }
 cur=cur->rlink;
 c++;
 }
 if(f==1 && z==0) {
 printf("Search successful, found at index %d\n",c);
 return head;
 }
 if(f==1 && z==1){
 prev=cur->llink;
 printf("enter towards left of %d=",key);
 temp=getnode();
 scanf("%d",&temp->info);
 prev->rlink=temp;
 temp->llink=prev;
 cur->llink=temp;
 temp->rlink=cur;
 return head;
 }
 if(f==1 && z==2){
 prev=cur;
 cur=cur->rlink;
 printf("enter towards right of %d=",key);
 temp=getnode();
 scanf("%d",&temp->info);
 prev->rlink=temp;
 temp->llink=prev;
 cur->llink=temp;
 temp->rlink=cur;
 return head;
 }
 printf("Search unsuccessful\n");
}
NODE delete_all_key(int item,NODE head)
{
NODE prev,cur,next;
int count;
if(head->rlink==head)
 {
printf("List Empty\n");
return head;
}
count=0;
cur=head->rlink;
while(cur!=head)
{
 if(item!=cur->info)
 cur=cur->rlink;
 else
{
 count++;
 prev=cur->llink;
 next=cur->rlink;
 prev->rlink=next;
 next->llink=prev;
free(cur);
 cur=next;
}
}
if(count==0)
 printf("key not found\n");
 else
printf("key found at %d positions and are deleted\n", count);

return head;
}

void display(NODE head)
{
 NODE temp;
 if(head->rlink==head)
 {
 printf("dq empty\n");
 return;
 }
 printf("contents of dq\n");
 temp=head->rlink;
 while(temp!=head)
 {
 printf("%d ",temp->info);
 temp=temp->rlink;
 }
 printf("\n");
}
void main(){
 NODE head, last;
 int item, choice;
 head=getnode();
 head->rlink=head;
 head->llink=head;
 for(;;){
```

```
 printf("Enter choice:\n1. Insert Front\n2. Delete front\n3. Insert rear\n4.
Delete rear\n5. Simple search\n6. Insert left of key\n7. Insert right of key\n8.
Delete all occurunces of key\n9. Display\n--- Any other key to exit ---\n");
scanf("%d",&choice);
switch(choice){
case 1: printf("Enter the item at front end\n");
scanf("%d",&item);
head=dinsert_front(item,head);
break;
case 3: printf("enter the item at rear end\n");
scanf("%d",&item);
head=dinsert_rear(item,head);
break;
case 2:
head=ddelete_front(head);
break;
case 4:
 head=ddelete_rear(head);
break;
 case 5:printf("Enter key\n");
 scanf("%d",&item);
 head=lsearch(head,item,0);
 break;
 case 6:printf("Enter key\n");
 scanf("%d",&item);
 head=lsearch(head,item,1);
 break;
 case 7:printf("Enter key\n");
 scanf("%d",&item);
 head=lsearch(head,item,2);
 break;
 case 8: printf("Enter key\n");
 scanf("%d",&item);
 head=delete_all_key(item,head);
 break;
case 9: display(head);
break;
default:exit(0);}}}
```



```
Enter choice:
1. Insert Front
2. Delete front
3. Insert rear
4. Delete rear
5. Simple search
6. Insert left of key
7. Insert right of key
8. Delete all occurunces of key
9. Display
--- Any other key to exit ---
1
Enter the item at front end
10
Enter choice:
1. Insert Front
2. Delete front
3. Insert rear
4. Delete rear
5. Simple search
6. Insert left of key
7. Insert right of key
8. Delete all occurunces of key
9. Display
--- Any other key to exit ---
3
enter the item at rear end
20
Enter choice:
1. Insert Front
2. Delete front
3. Insert rear
4. Delete rear
5. Simple search
6. Insert left of key
7. Insert right of key
8. Delete all occurunces of key
9. Display
--- Any other key to exit ---
9
```

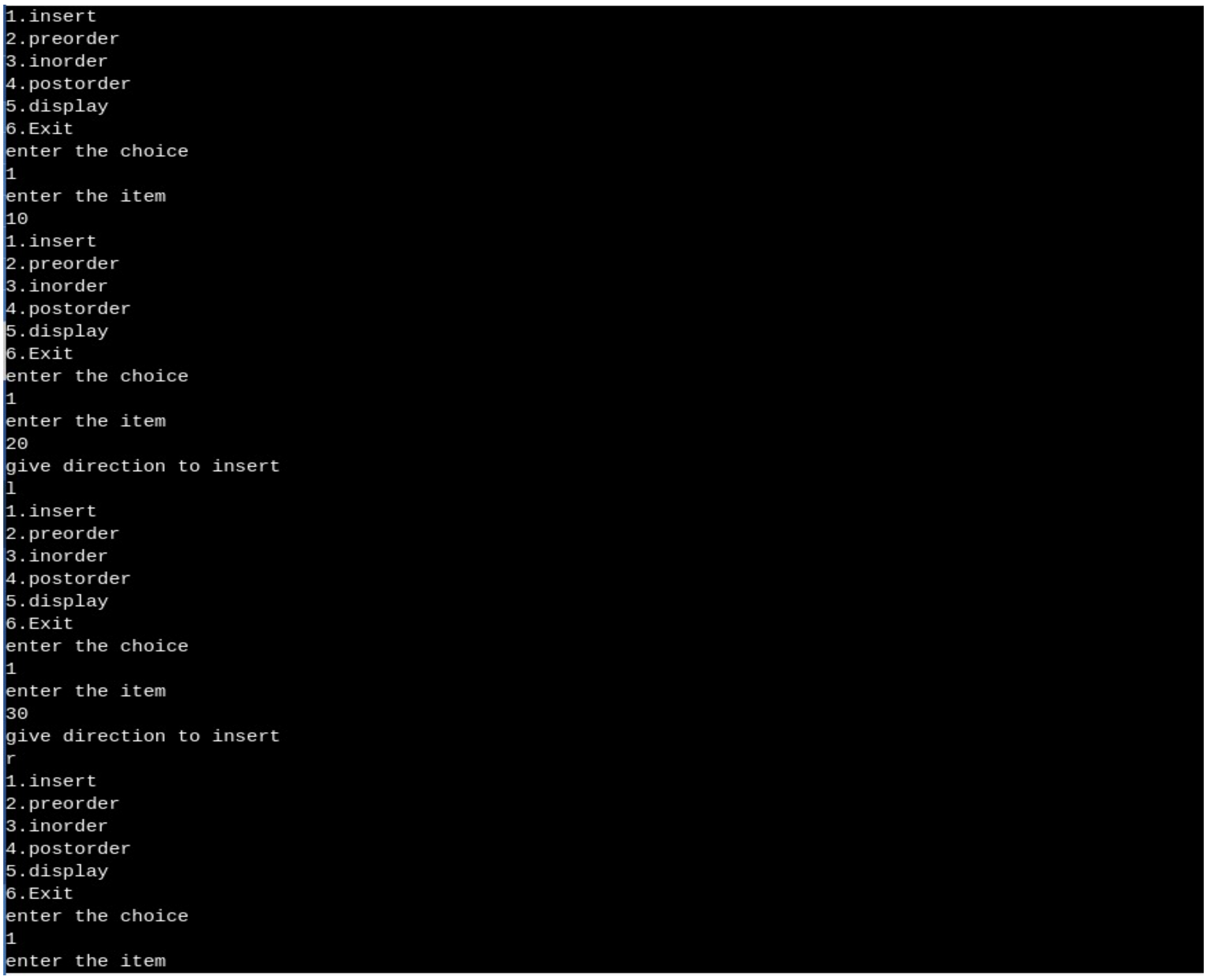


```
5. Simple search
6. Insert left of key
7. Insert right of key
8. Delete all occurunces of key
9. Display
--- Any other key to exit ---
3
enter the item at rear end
20
Enter choice:
1. Insert Front
2. Delete front
3. Insert rear
4. Delete rear
5. Simple search
6. Insert left of key
7. Insert right of key
8. Delete all occurunces of key
9. Display
--- Any other key to exit ---
9
contents of dq
10 20
Enter choice:
1. Insert Front
2. Delete front
3. Insert rear
4. Delete rear
5. Simple search
6. Insert left of key
7. Insert right of key
8. Delete all occurunces of key
9. Display
--- Any other key to exit ---
5
Enter key
10
Search successful, found at index 1
```

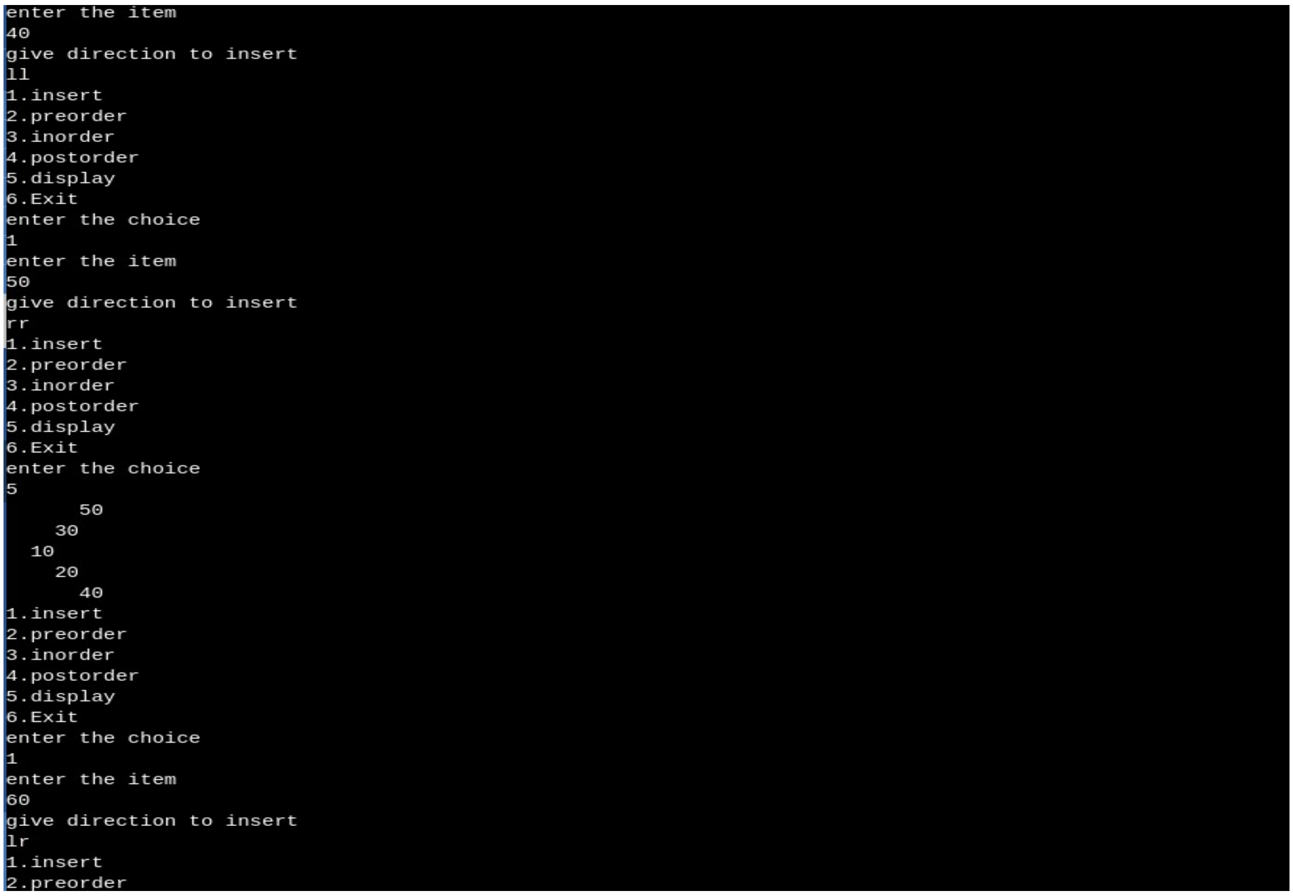
10. Binary Search Tree(Traverse and Display)

```
#include<stdio.h>
#include<conio.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 not available");
exit(0);
}
return x;
}
void freenode(NODE x)
{
free(x);
}
NODE insert(int item,NODE root)
{
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("give direction to insert\n");
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\n");
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)
{
printf("the item is %d\n",root->info);
preorder(root->llink);
preorder(root->rlink);
}
}
void inorder(NODE root)
{
if(root!=NULL)
{
inorder(root->llink);
printf("the item is%d\n",root->info);
inorder(root->rlink);
}
}
void postorder(NODE root)
{
if (root!=NULL)
{
postorder(root->llink);
postorder(root->rlink);
printf("the item is%d\n",root->info);
}
}
void display(NODE root,int i)
```

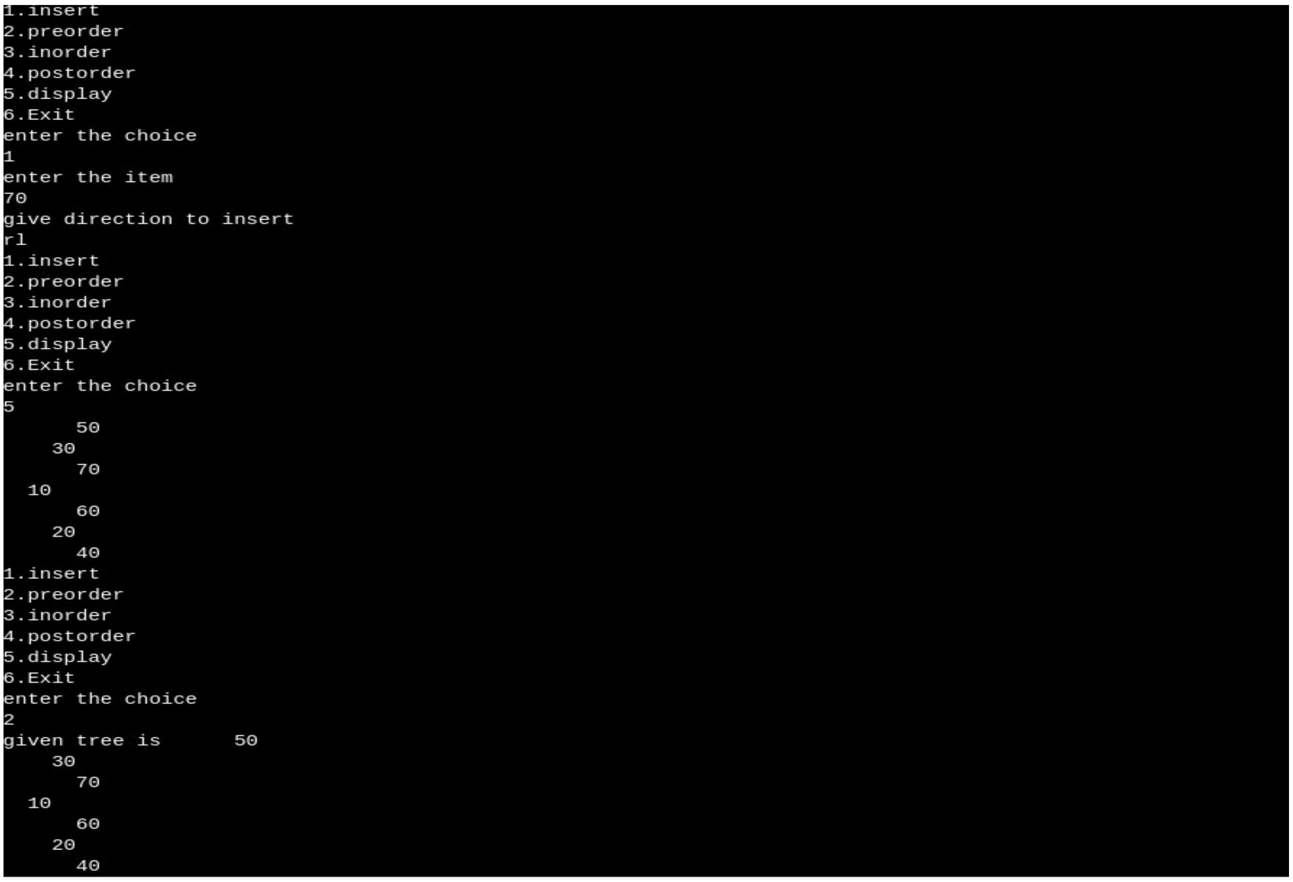
```
{
int j;
if(root!=NULL)
{
display(root->rlink,i+1);
for (j=1;j<=i;j++)
printf("  ");
printf("%d\n",root->info);
display(root->llink,i+1);
}
}

void main()
{
NODE root=NULL;
int choice,i,item;
for(;;)
{
printf("1.insert\n2.preorder\n3.inorder\n4.postorder\n5.display\n6.Exit\n");
printf("enter the choice\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("enter the item\n");
                scanf("%d",&item);
                root=insert(item,root);
                break;
case 2: if(root==NULL)
                {
                 printf("tree is empty");
                }
                else
                {
                 printf("given tree is");
                 display(root,1);
                 printf("the preorder traversal is \n");
                 preorder(root);
                }
                break;
case 3:if(root==NULL)
           {
                printf("tree is empty");
           }
           else
           {
                printf("given tree is");
                display(root,1);
                printf("the inorder traversal is \n");
                inorder(root);
                }
           break;
case 4:if (root==NULL)
                {
                printf("tree is empty");
                 }
           else
           {
                printf("given tree is");
                display(root,1);
                printf("the postorder traversal is \n");
                postorder(root);
           }
          break;
case 5:display(root,1);
           break;
default:exit(0);}}}
```



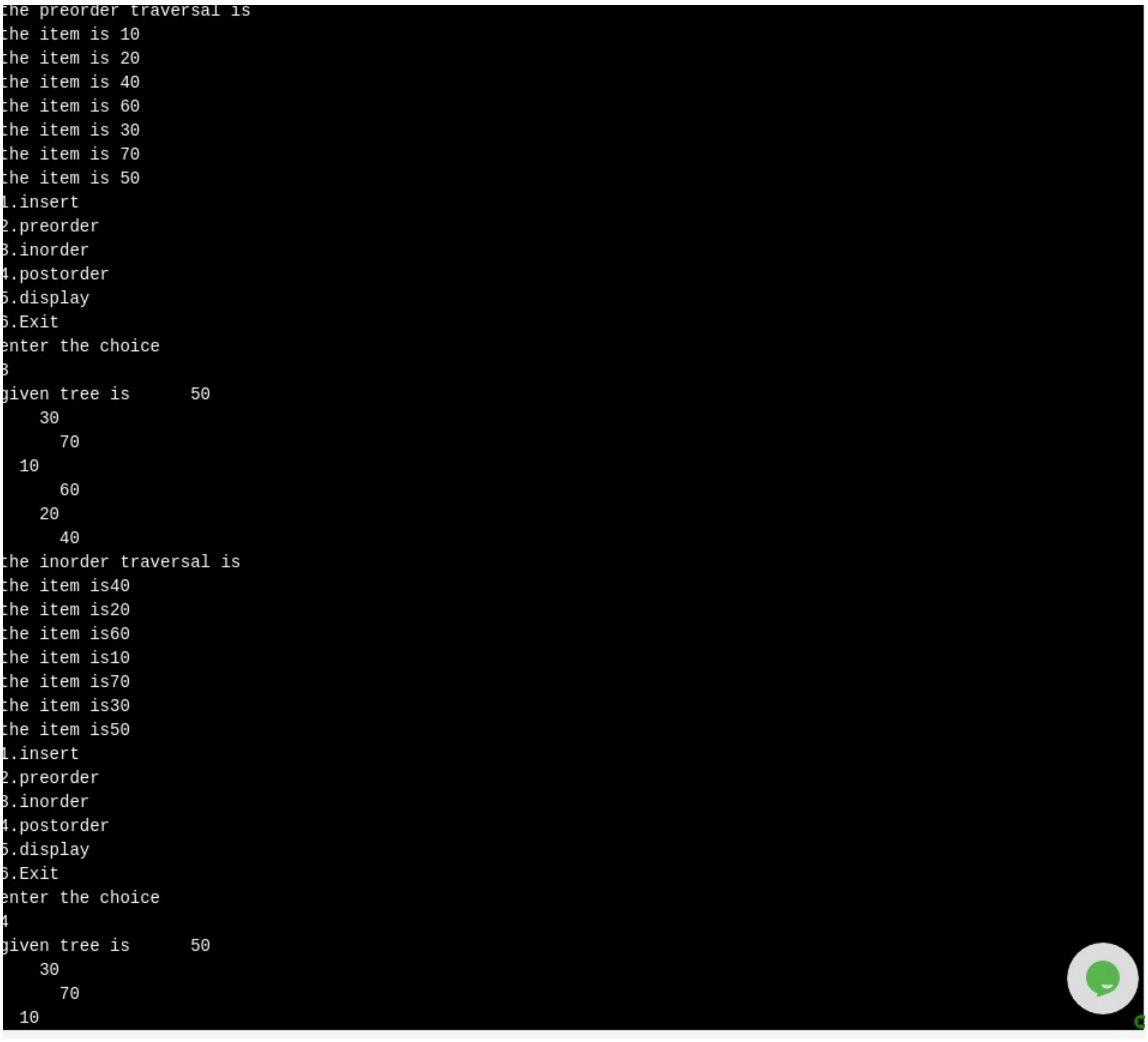
```
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
10
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
20
give direction to insert
l
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
30
give direction to insert
r
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
```



```
enter the item
40
give direction to insert
ll
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
50
give direction to insert
rr
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
5
      50
    30
  10
    20
      40
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
60
give direction to insert
lr
1.insert
2.preorder
```



```
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
1
enter the item
70
give direction to insert
rl
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
5
      50
    30
      70
  10
      60
    20
      40
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
2
given tree is       50
    30
      70
  10
      60
    20
      40
```



```
the preorder traversal is
the item is 10
the item is 20
the item is 40
the item is 60
the item is 30
the item is 70
the item is 50
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
3
given tree is      50
    30
      70
  10
      60
    20
      40
the inorder traversal is
the item is40
the item is20
the item is60
the item is10
the item is70
the item is30
the item is50
1.insert
2.preorder
3.inorder
4.postorder
5.display
6.Exit
enter the choice
4
given tree is      50
    30
      70
  10
```