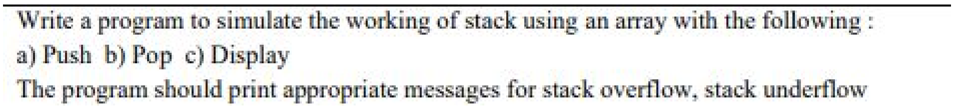
1.



Write a program to simulate the working of stack using an array with the following :

a) Push b) Pop c) Display

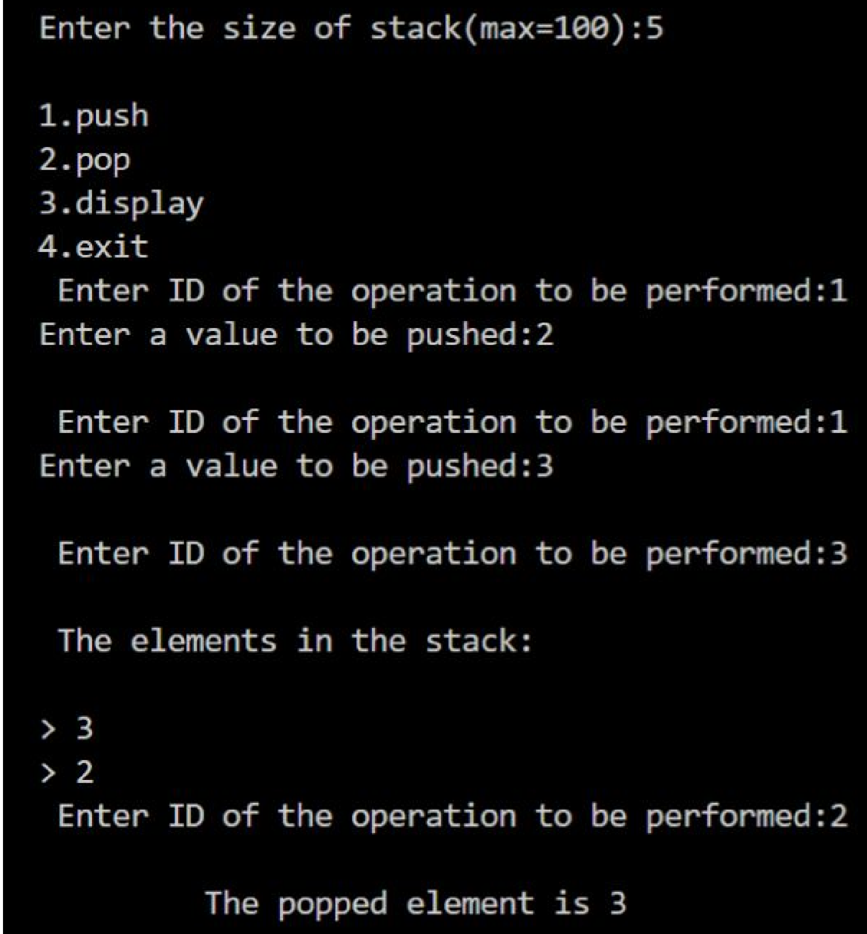
The program should print appropriate messages for stack overflow, stack underflow

Code

```
#include<stdio.h>
void push();
void pop();
void display();
int stack[100],choice,n,top,x,i;

int main()
{
    top=-1;
    printf("\nEnter the size of stack(max=100):");
    scanf("%d",&n);
    printf("\n1.push\n2.pop\n3.display\n4.exit");
    do
    {
        printf("\n Enter ID of the operation to be performed:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("\nexit");
                break;
            }
            default:
            {
                printf ("\nEnter a Valid Choice");
            }

        }
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
        printf("\nStack Overflow");

    }
    else
    {
        printf("Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\nStack underflow");
    }
    else
    {
        printf("\n\t The popped element is %d",stack[top]);
        top--;
    }
}
void display()
{
    if(top>=0)
```

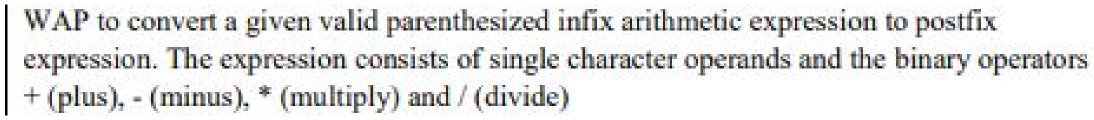
```
  {
      printf("\n The elements in the stack: \n");
      for(i=top; i>=0; i--)
          printf("\n> %d",stack[i]);
  }
  else
  {
      printf("\nStack is empty");
  }
```

OUTPUT



```
Enter the size of stack(max=100):5

1.push
2.pop
3.display
4.exit
 Enter ID of the operation to be performed:1
Enter a value to be pushed:2

 Enter ID of the operation to be performed:1
Enter a value to be pushed:3

 Enter ID of the operation to be performed:3

 The elements in the stack:

> 3
> 2
 Enter ID of the operation to be performed:2

         The popped element is 3
```

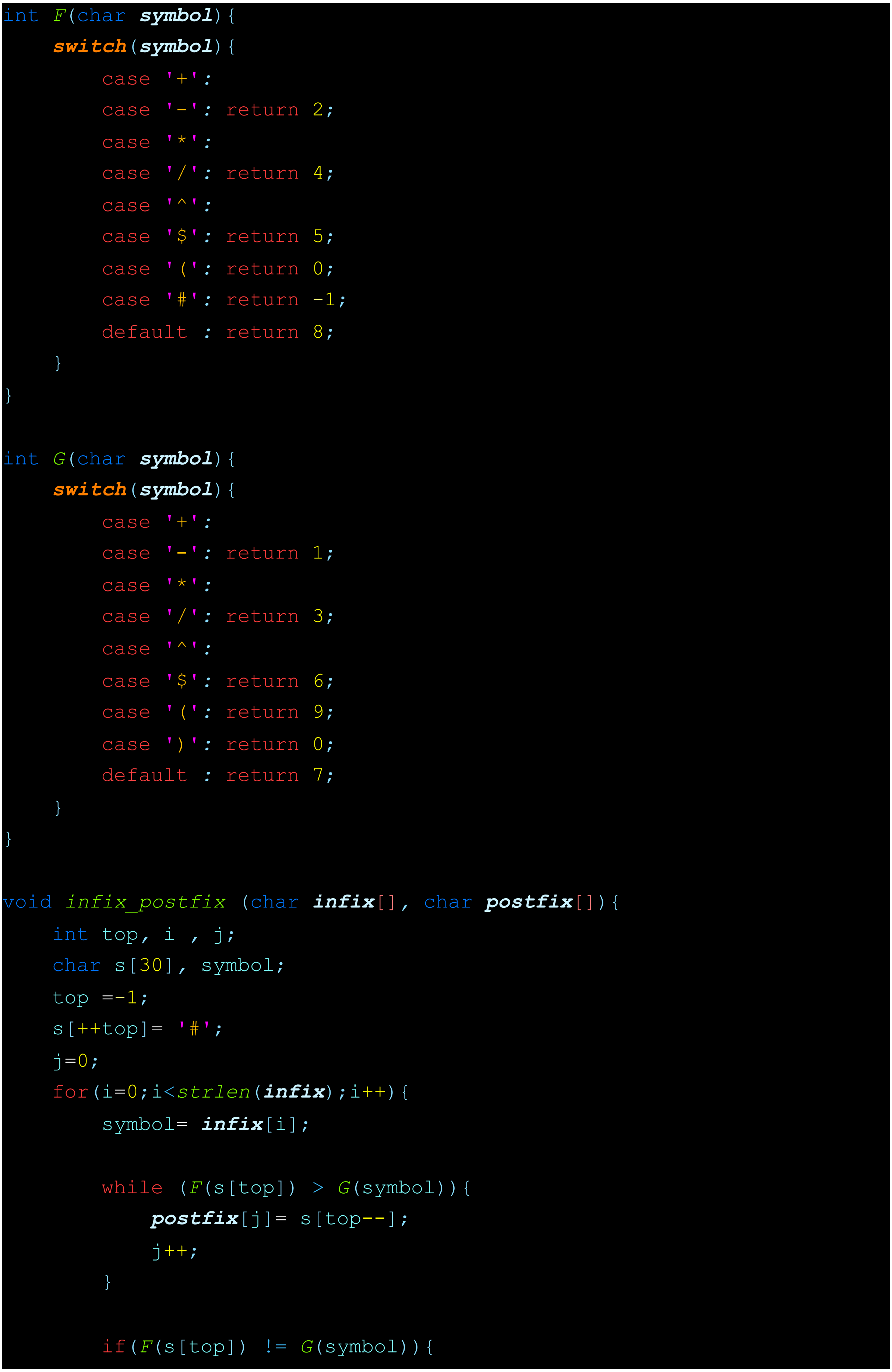
2.



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)

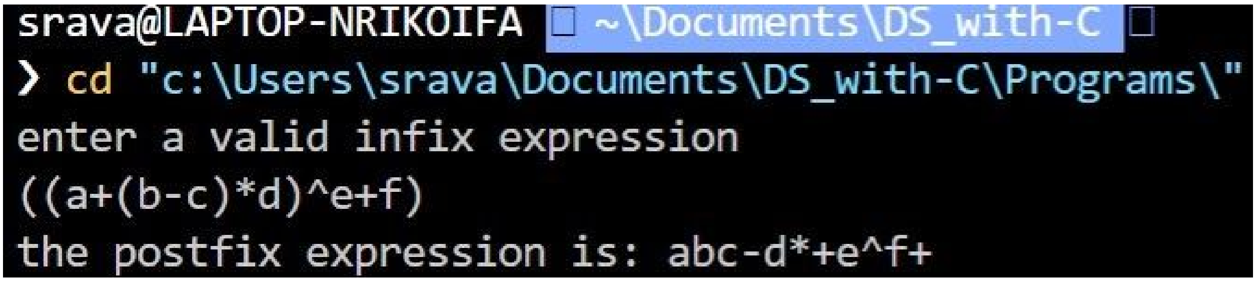
Code

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



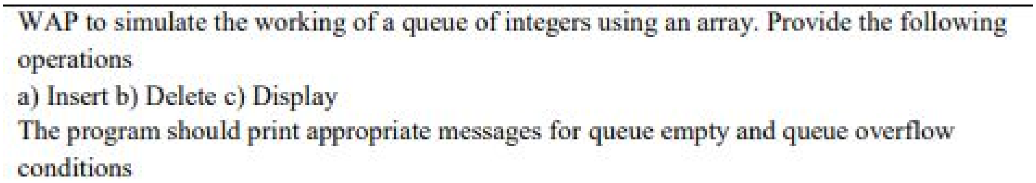
```
int F(char symbol){
    switch(symbol){
        case '+':
        case '-': return 2;
        case '*':
        case '/': return 4;
        case '^':
        case '$': return 5;
        case '(': return 0;
        case '#': return -1;
        default : return 8;
    }
}

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';
}

void main(){
    char infix [20];
    char postfix[20];
    printf("enter a valid infix expression \n");
    scanf("%s", infix);
    infix_postfix(infix, postfix);
    printf("the postfix expression is: ");
    printf("%s", postfix);
}
```

OUTPUT



```
srava@LAPTOP-NRIKOIFA □ ~\Documents\DS_with-C □
❯ cd "c:\Users\srava\Documents\DS_with-C\Programs\"
enter a valid infix expression
((a+(b-c)*d)^e+f)
the postfix expression is: abc-d*+e^f+
```

3.

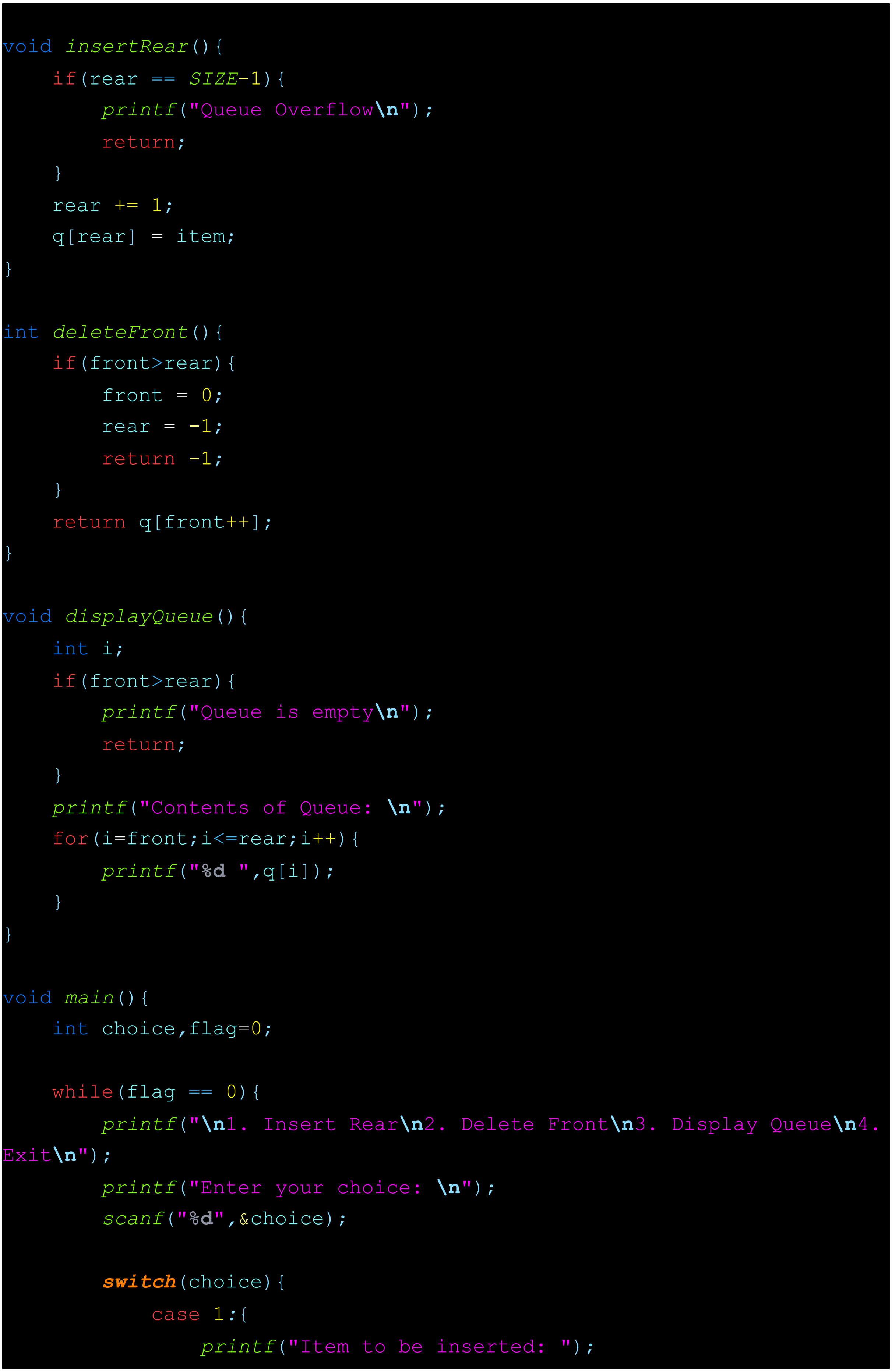


WAP to simulate the working of a queue of integers using an array. Provide the following operations

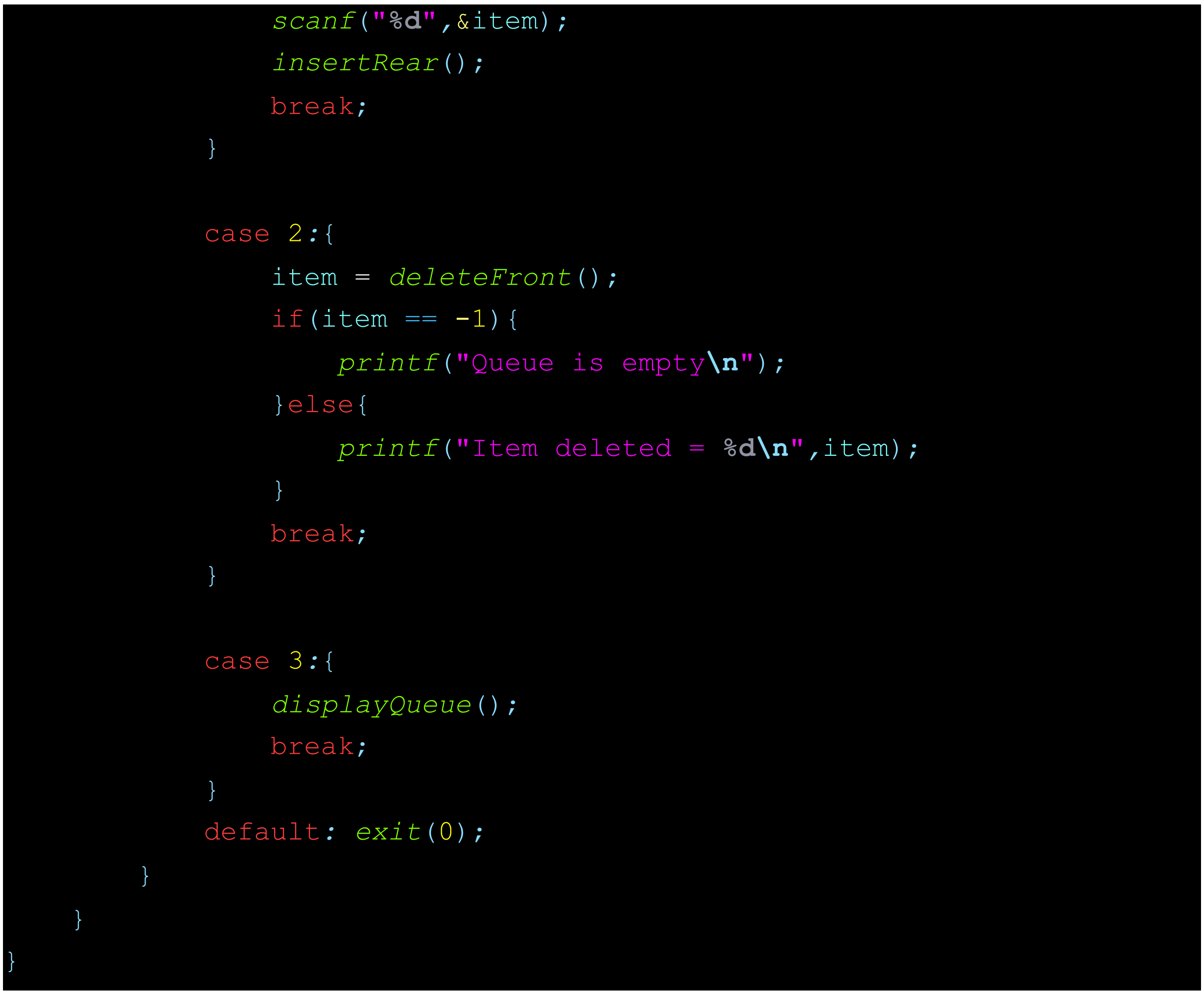
a) Insert b) Delete c) Display

The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#define SIZE 3

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

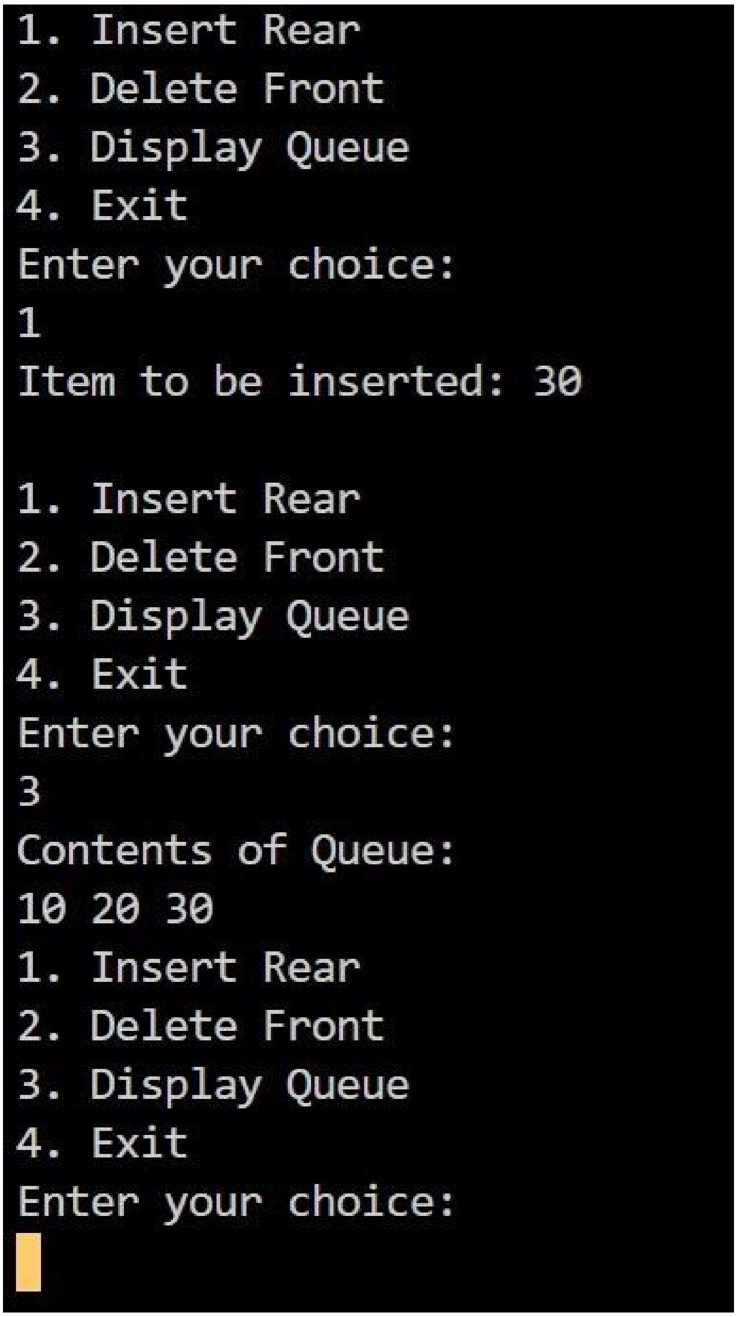


```c
void insertRear(){
    if(rear == SIZE-1){
        printf("Queue Overflow\n");
        return;
    }
    rear += 1;
    q[rear] = item;
}

int deleteFront(){
    if(front>rear){
        front = 0;
        rear = -1;
        return -1;
    }
    return q[front++];
}

void displayQueue(){
    int i;
    if(front>rear){
        printf("Queue is empty\n");
        return;
    }
    printf("Contents of Queue: \n");
    for(i=front;i<=rear;i++){
        printf("%d ",q[i]);
    }
}

void main(){
    int choice,flag=0;

    while(flag == 0){
        printf("\n1. Insert Rear\n2. Delete Front\n3. Display Queue\n4. Exit\n");
        printf("Enter your choice: \n");
        scanf("%d",&choice);

        switch(choice){
            case 1:{
                printf("Item to be inserted: ");
```



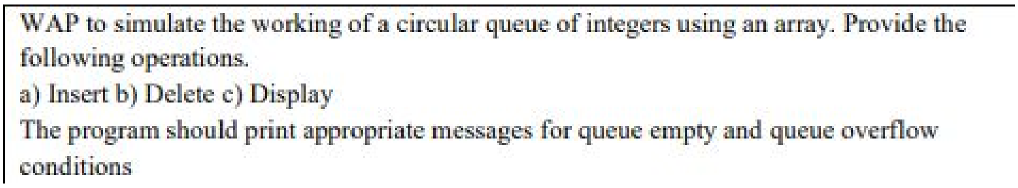
```
                scanf("%d",&item);
                insertRear();
                break;
            }

            case 2:{
                item = deleteFront();
                if(item == -1){
                    printf("Queue is empty\n");
                }else{
                    printf("Item deleted = %d\n",item);
                }
                break;
            }

            case 3:{
                displayQueue();
                break;
            }
            default: exit(0);
        }
    }
}
```

OUTPUT



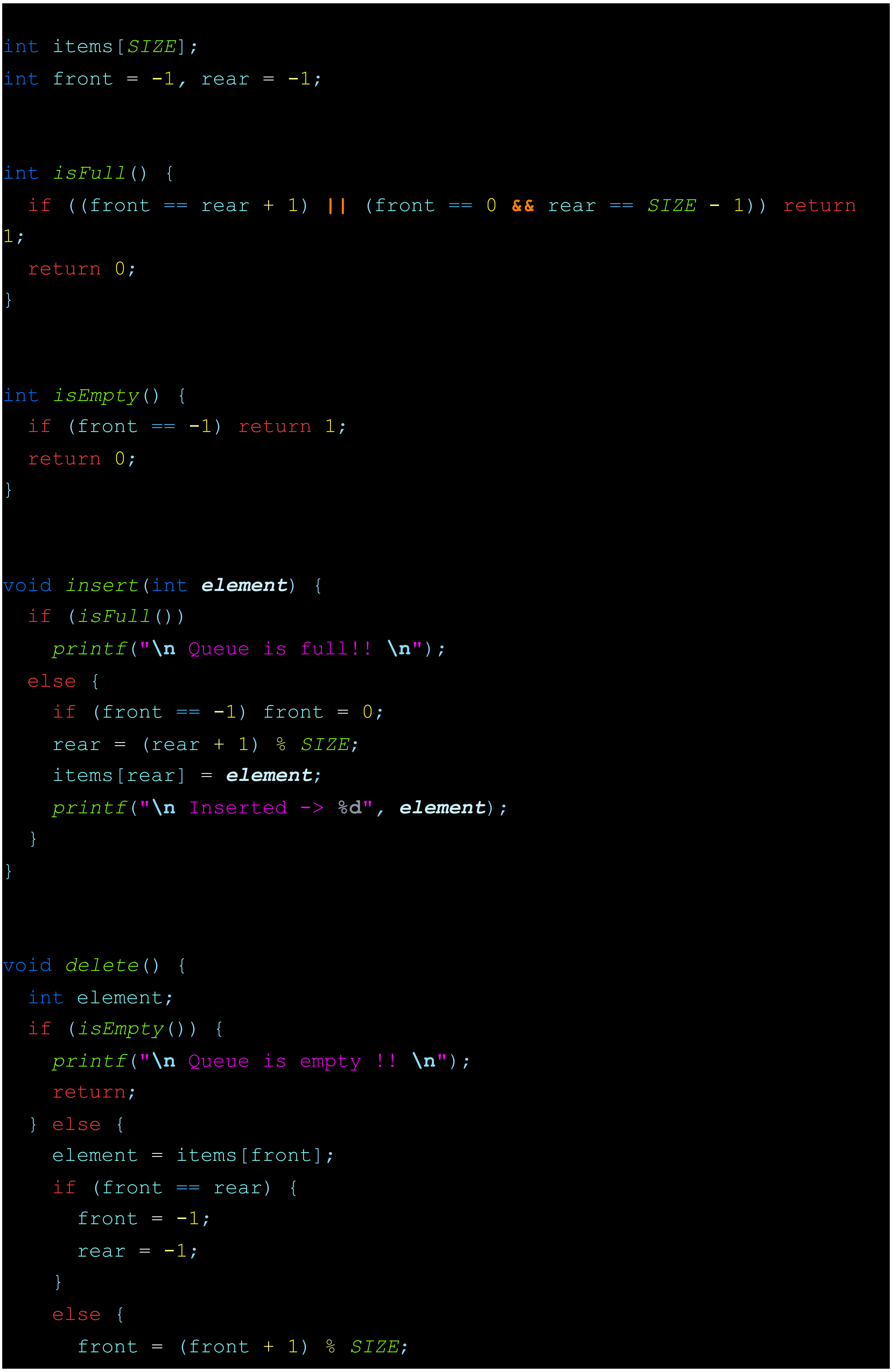
```
1. Insert Rear
2. Delete Front
3. Display Queue
4. Exit
Enter your choice:
1
Item to be inserted: 30

1. Insert Rear
2. Delete Front
3. Display Queue
4. Exit
Enter your choice:
3
Contents of Queue:
10 20 30
1. Insert Rear
2. Delete Front
3. Display Queue
4. Exit
Enter your choice:
```

4.

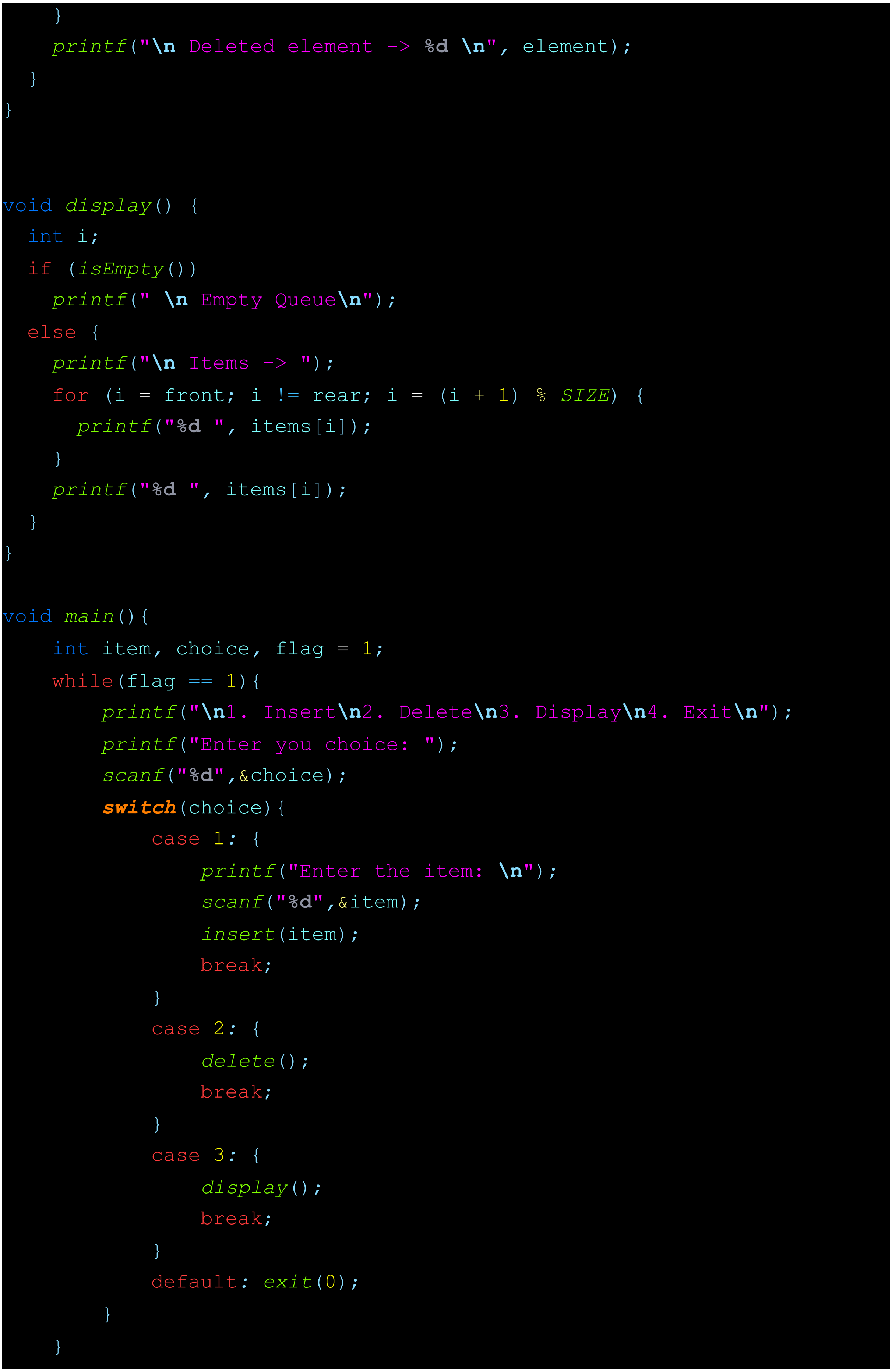


WAP to simulate the working of a circular queue of integers using an array. Provide the following operations.
a) Insert b) Delete c) Display
The program should print appropriate messages for queue empty and queue overflow conditions

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

#define SIZE 5
```



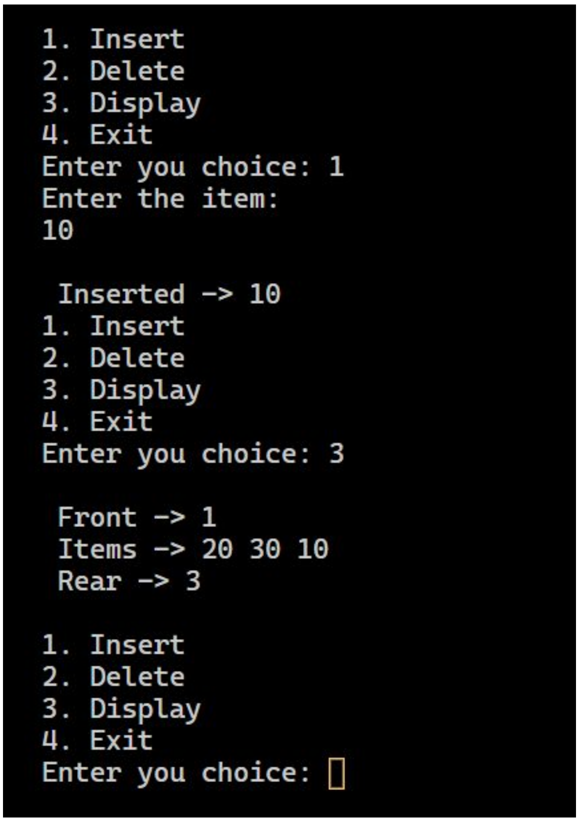
```
int items[SIZE];
int front = -1, rear = -1;


int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return
1;
  return 0;
}


int isEmpty() {
  if (front == -1) return 1;
  return 0;
}


void insert(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    printf("\n Inserted -> %d", element);
  }
}


void delete() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty !! \n");
    return;
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
    else {
      front = (front + 1) % SIZE;
```



```
    }
    printf("\n Deleted element -> %d \n", element);
  }
}


void display() {
  int i;
  if (isEmpty())
    printf(" \n Empty Queue\n");
  else {
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
  }
}

void main(){
    int item, choice, flag = 1;
    while(flag == 1){
        printf("\n1. Insert\n2. Delete\n3. Display\n4. Exit\n");
        printf("Enter you choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: {
                printf("Enter the item: \n");
                scanf("%d",&item);
                insert(item);
                break;
            }
            case 2: {
                delete();
                break;
            }
            case 3: {
                display();
                break;
            }
            default: exit(0);
        }
    }
```

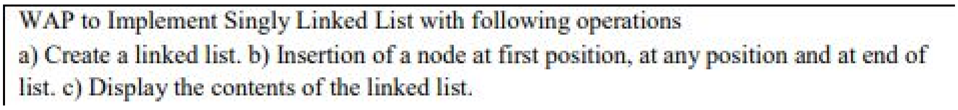
```
}
```

OUTPUT



```
1. Insert
2. Delete
3. Display
4. Exit
Enter you choice: 1
Enter the item:
10

 Inserted -> 10
1. Insert
2. Delete
3. Display
4. Exit
Enter you choice: 3

 Front -> 1
 Items -> 20 30 10
 Rear -> 3

1. Insert
2. Delete
3. Display
4. Exit
Enter you choice: 
```

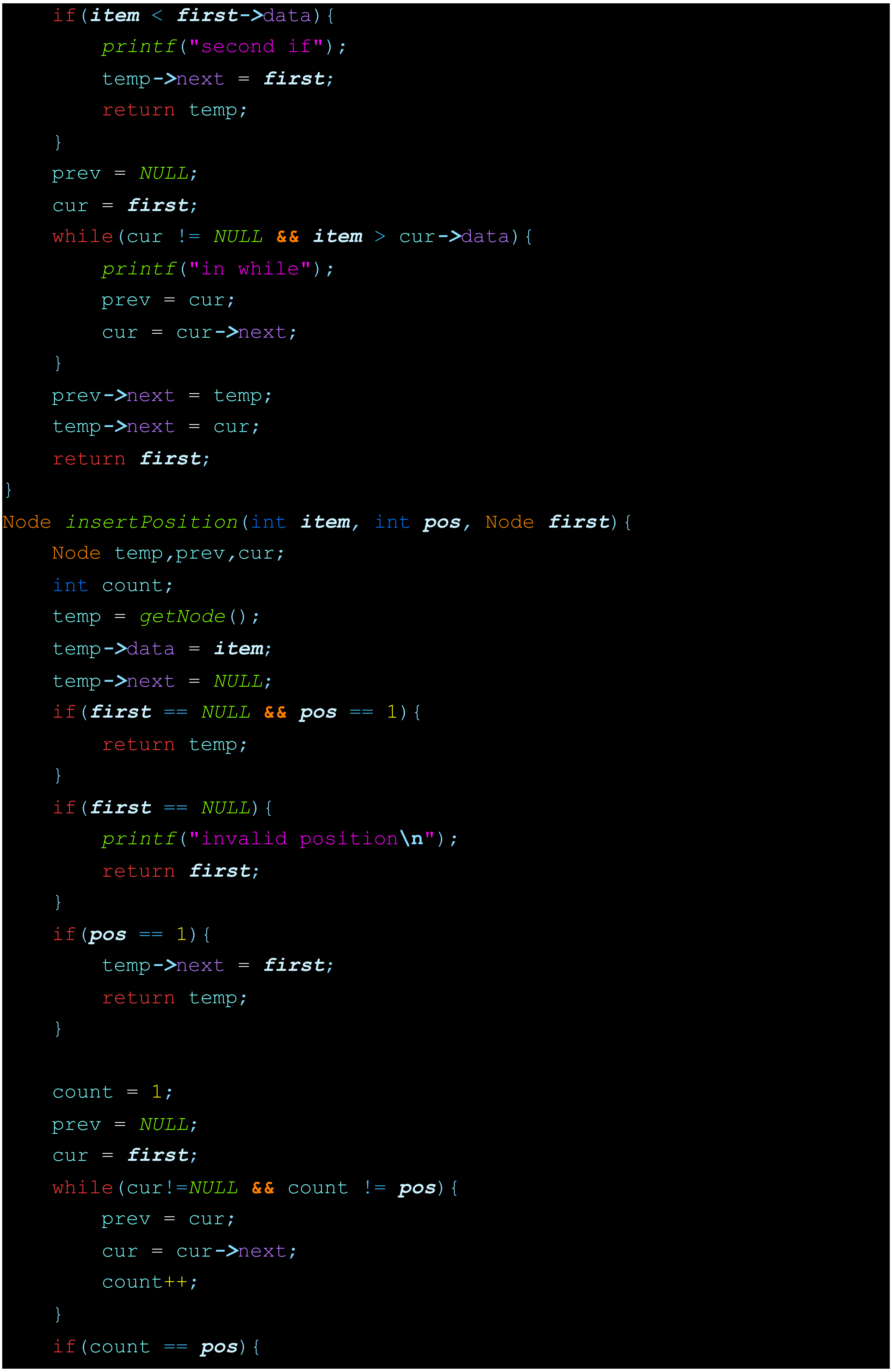
5.



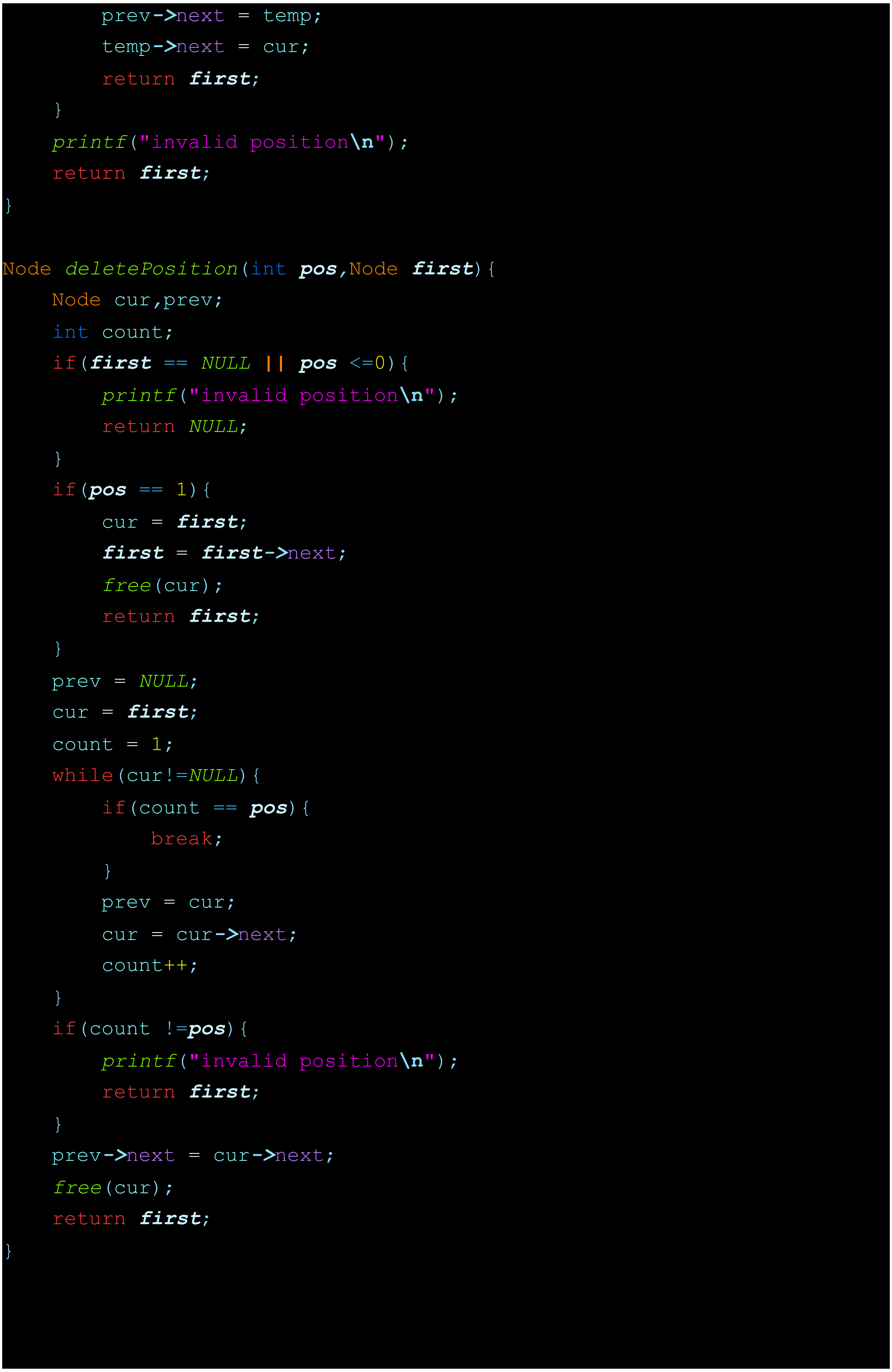
WAP to Implement Singly Linked List with following operations
a) Create a linked list. b) Insertion of a node at first position, at any position and at end of list. c) Display the contents of the linked list.

Code

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

struct node{
    int data;
    struct node *next;
};

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 insertRear(Node first,int item){
    Node temp,current;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    current = first;
    while(current->next != NULL){
        current = current->next;
    }
    current->next = temp;
    return first;
}

Node insertFront(Node first,int item){
    Node temp;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    temp->next = first;
    first = temp;
    return first;
}


Node deleteRear(Node first){
    Node current,previous;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    if(first->next == NULL){
        printf("Item deleted is %d\n",first->data);
        free(first);
        return NULL;
    }
    previous = NULL;
    current = first;
    while(current->next != NULL){
        previous = current;
        current = current->next;
    }
    printf("item deleted at rear end is %d\n",current->data);
    free(current);
    previous->next = NULL;
    return first;
}

Node deleteFront(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    temp = first;
    temp = temp->next;
    printf("Item delted at front end is %d\n",first->data);
    free(first);
    return temp;
}

Node orderList(Node first,int item){
    Node temp,prev,cur;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        printf("in first if\n");
        return temp;
    }
```

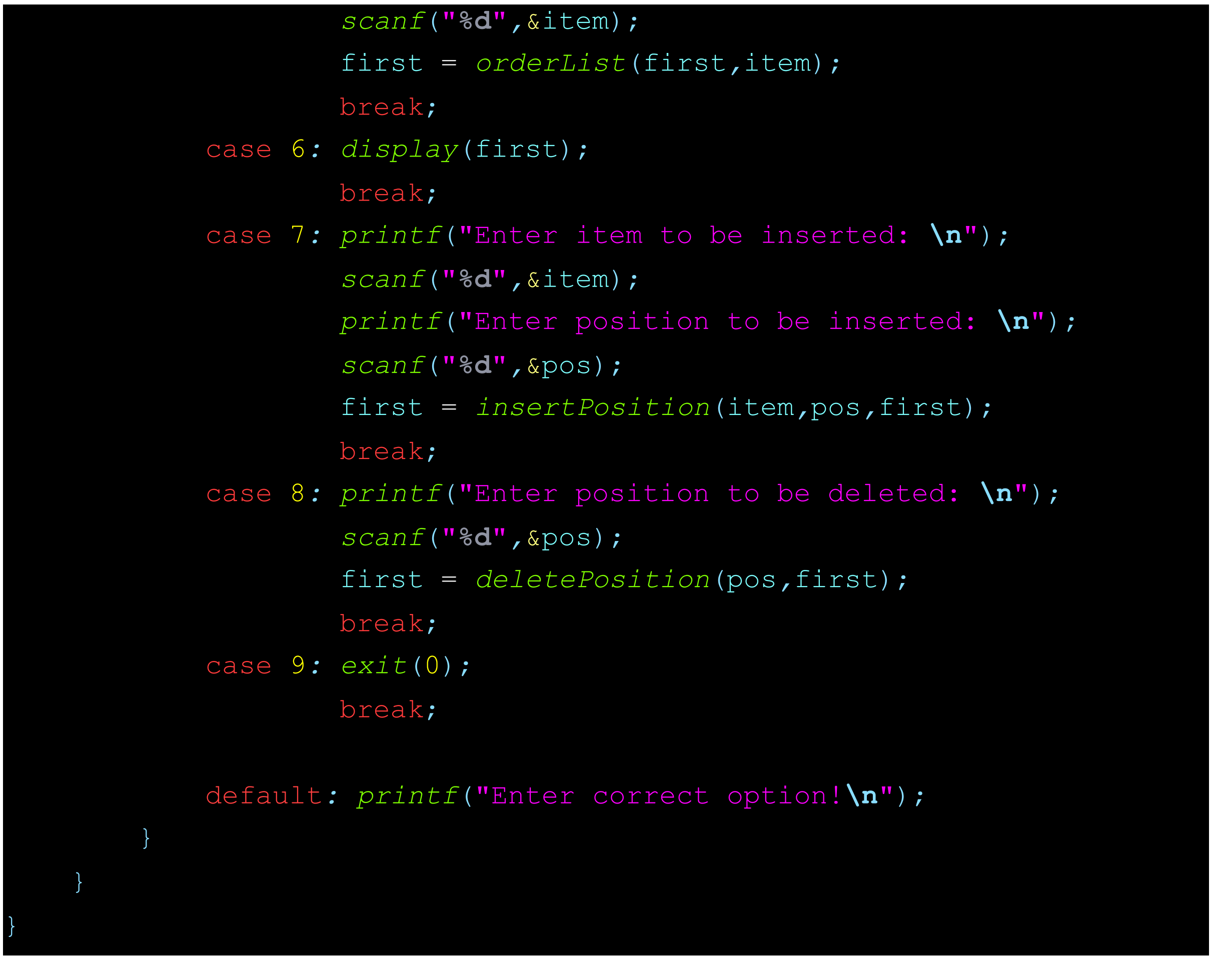


```
    if(item < first->data){
        printf("second if");
        temp->next = first;
        return temp;
    }
    prev = NULL;
    cur = first;
    while(cur != NULL && item > cur->data){
        printf("in while");
        prev = cur;
        cur = cur->next;
    }
    prev->next = temp;
    temp->next = cur;
    return first;
}
Node insertPosition(int item, int pos, Node first){
    Node temp,prev,cur;
    int count;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL && pos == 1){
        return temp;
    }
    if(first == NULL){
        printf("invalid position\n");
        return first;
    }
    if(pos == 1){
        temp->next = first;
        return temp;
    }

    count = 1;
    prev = NULL;
    cur = first;
    while(cur!=NULL && count != pos){
        prev = cur;
        cur = cur->next;
        count++;
    }
    if(count == pos){
```



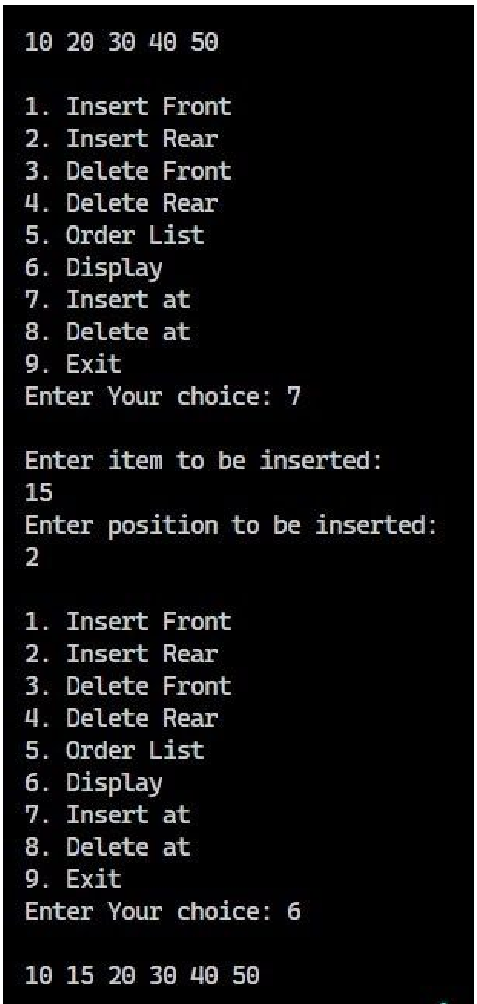
```
        prev->next = temp;
        temp->next = cur;
        return first;
    }
    printf("invalid position\n");
    return first;
}

Node deletePosition(int pos,Node first){
    Node cur,prev;
    int count;
    if(first == NULL || pos <=0){
        printf("invalid position\n");
        return NULL;
    }
    if(pos == 1){
        cur = first;
        first = first->next;
        free(cur);
        return first;
    }
    prev = NULL;
    cur = first;
    count = 1;
    while(cur!=NULL){
        if(count == pos){
            break;
        }
        prev = cur;
        cur = cur->next;
        count++;
    }
    if(count !=pos){
        printf("invalid position\n");
        return first;
    }
    prev->next = cur->next;
    free(cur);
    return first;
}
```

```
void display(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return;
    }
    for(temp=first;temp!=NULL;temp = temp->next){
        printf("%d ",temp->data);
    }
    printf("\n");
}

void main(){
    int item,choice,flag = 1,pos;
    Node first = NULL;

    while(flag == 1){
        printf("\n1. Insert Front\n2. Insert Rear\n3. Delete Front\n4.
Delete Rear\n5. Order List\n6. Display\n7. Insert at\n8. Delete at\n9.
Exit");
        printf("\nEnter Your choice: ");
        scanf("%d",&choice);
        printf("\n");
        switch(choice){
            case 1: printf("Enter the item to be inserted at front:
\n");
                    scanf("%d",&item);
                    first = insertFront(first,item);
                    break;
            case 2: printf("Enter the item to be inserted in the rear:
\n");
                    scanf("%d",&item);
                    first = insertRear(first,item);
                    break;
            case 3: first = deleteFront(first);
                    break;
            case 4: first = deleteRear(first);
                    break;
            case 5: printf("Enter item which will be inserted in
order(ascending): \n");
```



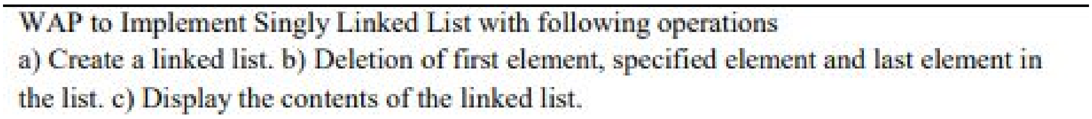
```
                    scanf("%d",&item);
                    first = orderList(first,item);
                    break;
            case 6: display(first);
                    break;
            case 7: printf("Enter item to be inserted: \n");
                    scanf("%d",&item);
                    printf("Enter position to be inserted: \n");
                    scanf("%d",&pos);
                    first = insertPosition(item,pos,first);
                    break;
            case 8: printf("Enter position to be deleted: \n");
                    scanf("%d",&pos);
                    first = deletePosition(pos,first);
                    break;
            case 9: exit(0);
                    break;

            default: printf("Enter correct option!\n");
        }
    }
}
```

OUTPUT

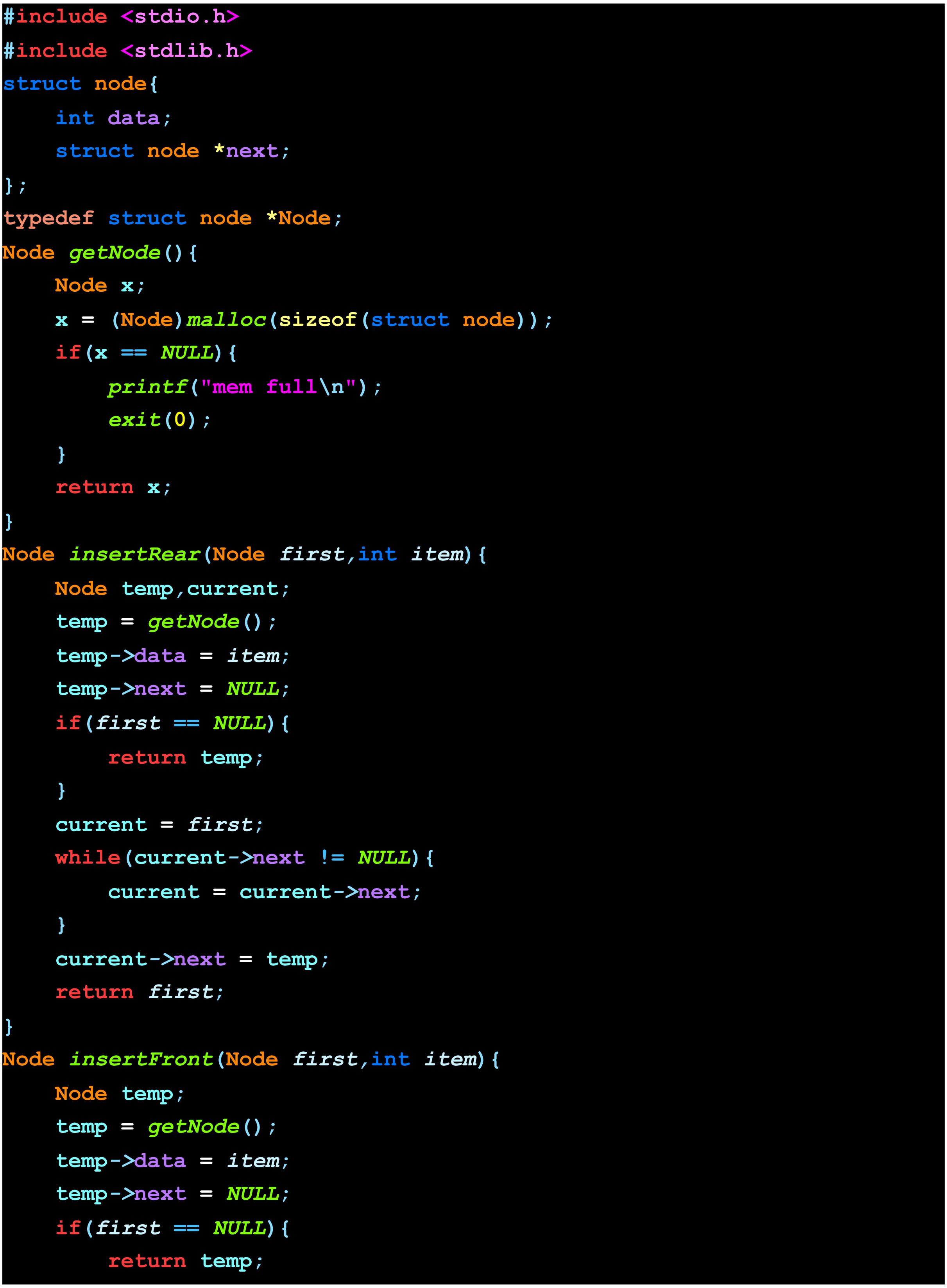


```
10 20 30 40 50

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Order List
6. Display
7. Insert at
8. Delete at
9. Exit
Enter Your choice: 7

Enter item to be inserted:
15
Enter position to be inserted:
2

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Order List
6. Display
7. Insert at
8. Delete at
9. Exit
Enter Your choice: 6

10 15 20 30 40 50
```

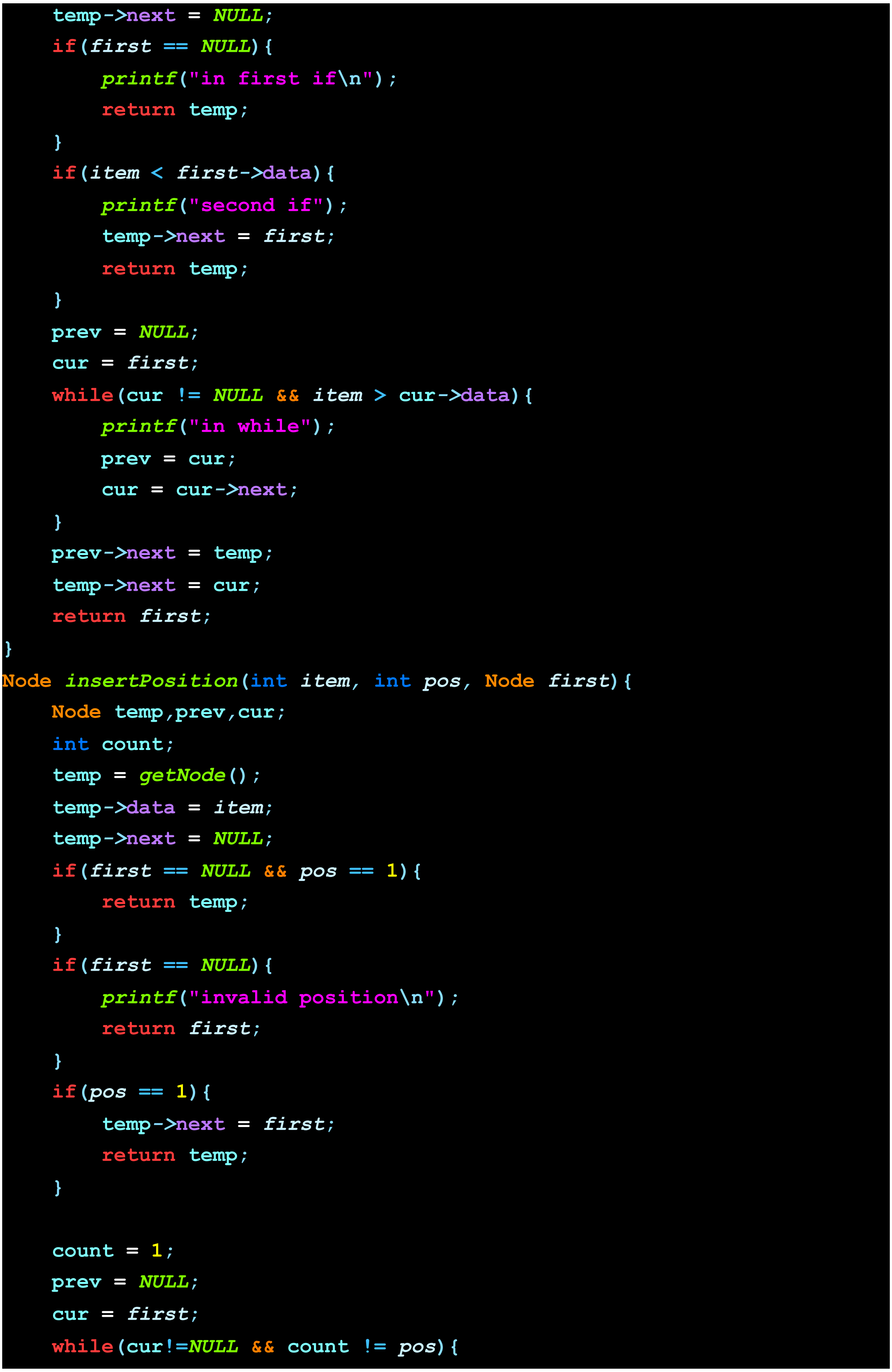
6.



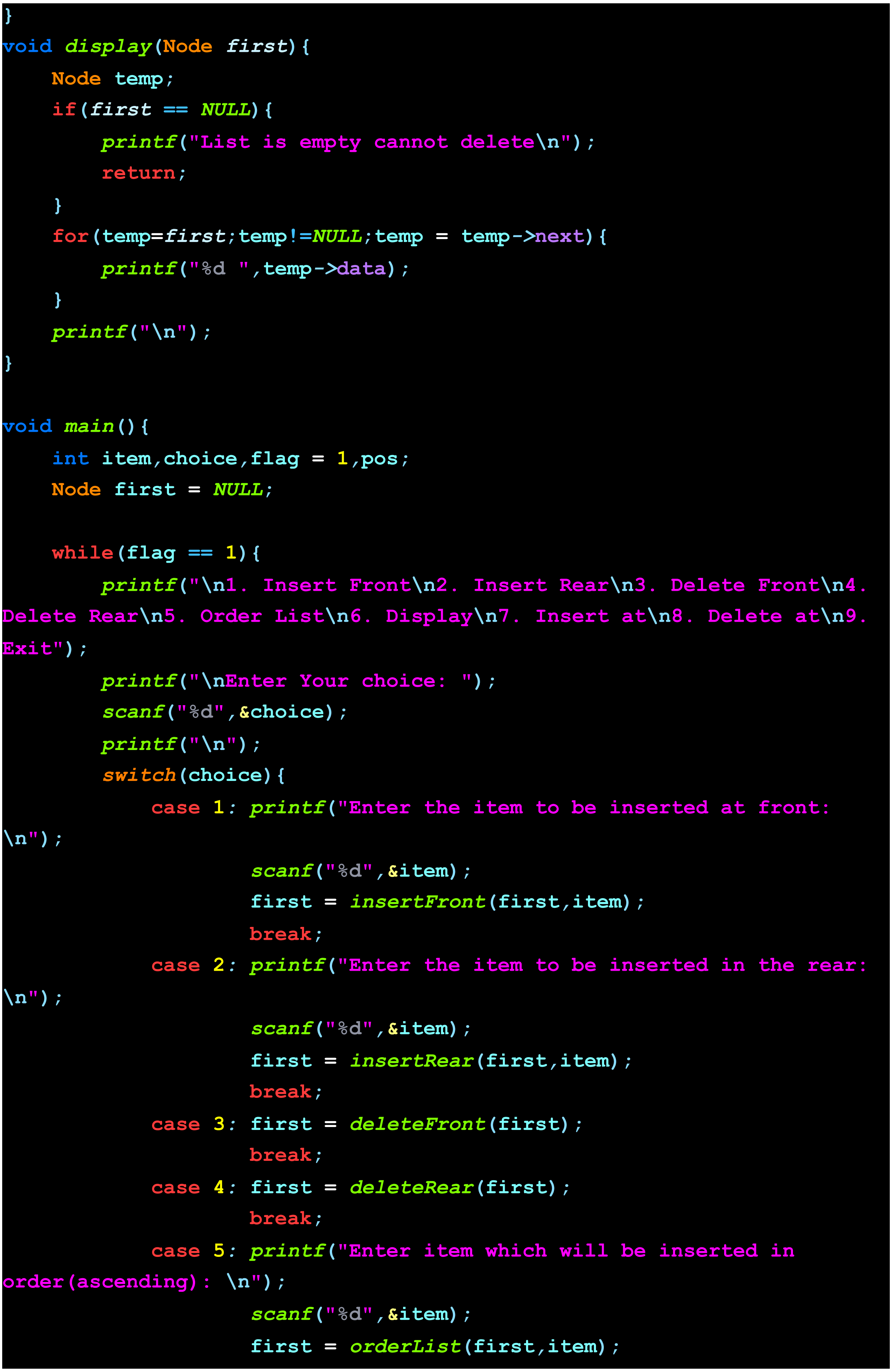
WAP to Implement Singly Linked List with following operations
a) Create a linked list. b) Deletion of first element, specified element and last element in the list. c) Display the contents of the linked list.



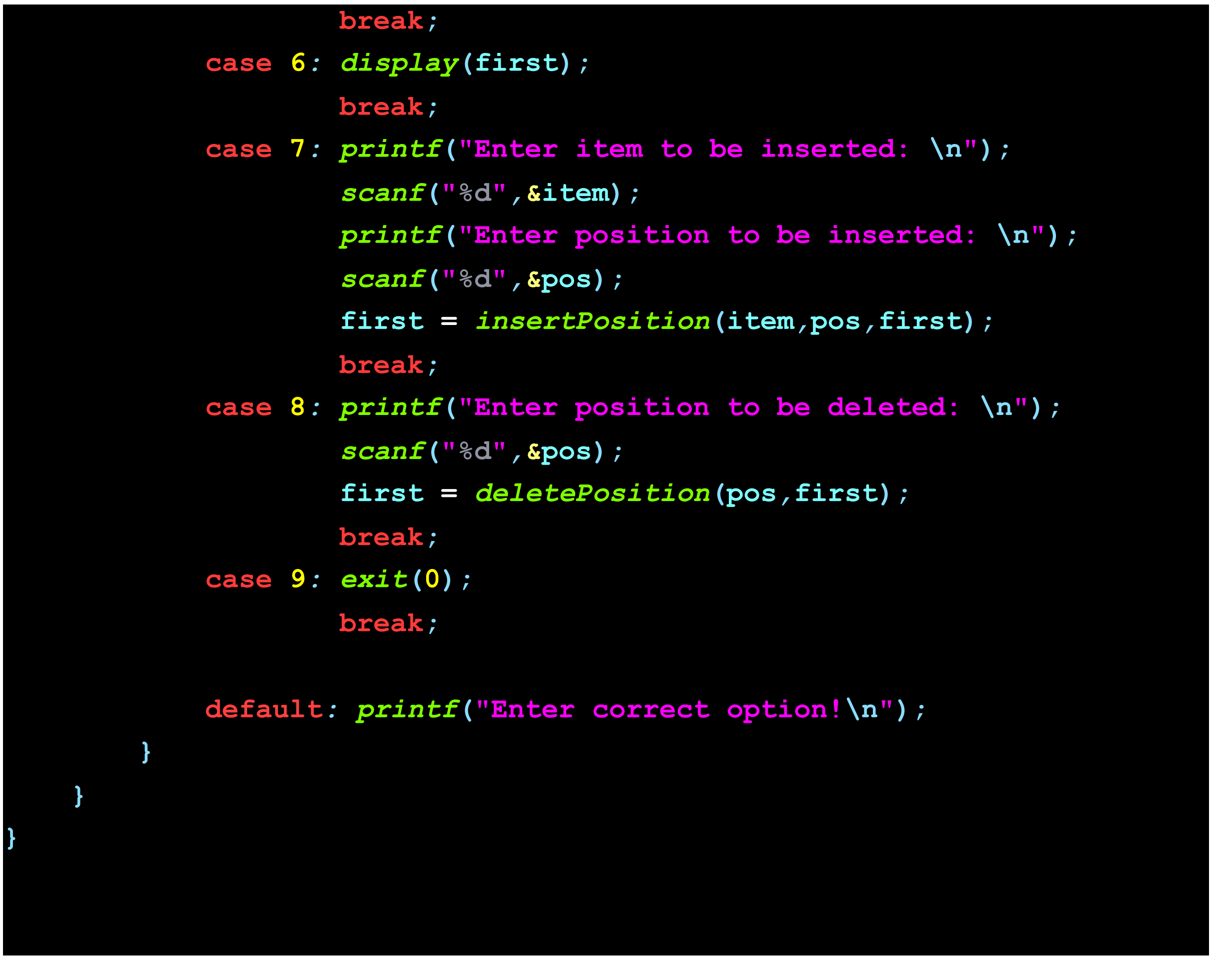
```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
};
typedef struct node *Node;
Node getNode(){
    Node x;
    x = (Node)malloc(sizeof(struct node));
    if(x == NULL){
        printf("mem full\n");
        exit(0);
    }
    return x;
}
Node insertRear(Node first,int item){
    Node temp,current;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    current = first;
    while(current->next != NULL){
        current = current->next;
    }
    current->next = temp;
    return first;
}
Node insertFront(Node first,int item){
    Node temp;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    temp->next = first;
    first = temp;
    return first;
}
Node deleteRear(Node first){
    Node current,previous;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    if(first->next == NULL){
        printf("Item deleted is %d\n",first->data);
        free(first);
        return NULL;
    }
    previous = NULL;
    current = first;
    while(current->next != NULL){
        previous = current;
        current = current->next;
    }
    printf("item deleted at rear end is %d\n",current->data);
    free(current);
    previous->next = NULL;
    return first;
}
Node deleteFront(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    temp = first;
    temp = temp->next;
    printf("Item delted at front end is %d\n",first->data);
    free(first);
    return temp;
}
Node orderList(Node first,int item){
    Node temp,prev,cur;
    temp = getNode();
    temp->data = item;
```



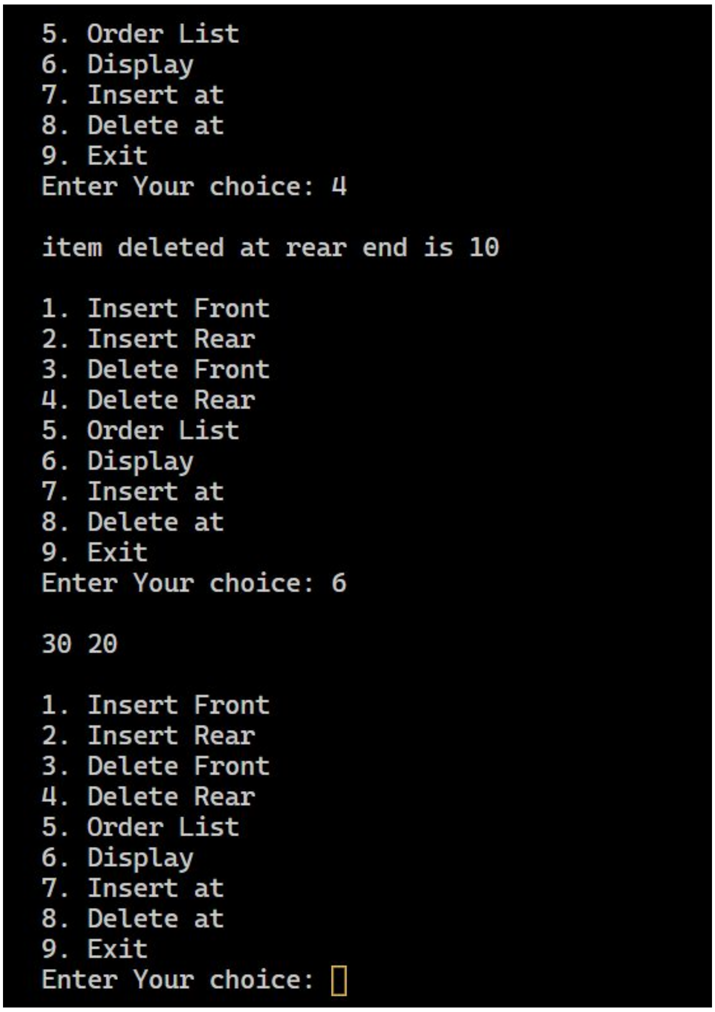
```
    temp->next = NULL;
    if(first == NULL){
        printf("in first if\n");
        return temp;
    }
    if(item < first->data){
        printf("second if");
        temp->next = first;
        return temp;
    }
    prev = NULL;
    cur = first;
    while(cur != NULL && item > cur->data){
        printf("in while");
        prev = cur;
        cur = cur->next;
    }
    prev->next = temp;
    temp->next = cur;
    return first;
}
Node insertPosition(int item, int pos, Node first){
    Node temp,prev,cur;
    int count;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL && pos == 1){
        return temp;
    }
    if(first == NULL){
        printf("invalid position\n");
        return first;
    }
    if(pos == 1){
        temp->next = first;
        return temp;
    }

    count = 1;
    prev = NULL;
    cur = first;
    while(cur!=NULL && count != pos){
        prev = cur;
        cur = cur->next;
        count++;
    }
    if(count == pos){
        prev->next = temp;
        temp->next = cur;
        return first;
    }
    printf("invalid position\n");
    return first;
}
Node deletePosition(int pos,Node first){
    Node cur,prev;
    int count;
    if(first == NULL || pos <=0){
        printf("invalid position\n");
        return NULL;
    }
    if(pos == 1){
        cur = first;
        first = first->next;
        free(cur);
        return first;
    }
    prev = NULL;
    cur = first;
    count = 1;
    while(cur!=NULL){
        if(count == pos){
            break;
        }
        prev = cur;
        cur = cur->next;
        count++;
    }
    if(count !=pos){
        printf("invalid position\n");
        return first;
    }
    prev->next = cur->next;
    free(cur);
    return first;
```



```
}
void display(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return;
    }
    for(temp=first;temp!=NULL;temp = temp->next){
        printf("%d ",temp->data);
    }
    printf("\n");
}

void main(){
    int item,choice,flag = 1,pos;
    Node first = NULL;

    while(flag == 1){
        printf("\n1. Insert Front\n2. Insert Rear\n3. Delete Front\n4.
Delete Rear\n5. Order List\n6. Display\n7. Insert at\n8. Delete at\n9.
Exit");
        printf("\nEnter Your choice: ");
        scanf("%d",&choice);
        printf("\n");
        switch(choice){
            case 1: printf("Enter the item to be inserted at front:
\n");
                    scanf("%d",&item);
                    first = insertFront(first,item);
                    break;
            case 2: printf("Enter the item to be inserted in the rear:
\n");
                    scanf("%d",&item);
                    first = insertRear(first,item);
                    break;
            case 3: first = deleteFront(first);
                    break;
            case 4: first = deleteRear(first);
                    break;
            case 5: printf("Enter item which will be inserted in
order(ascending): \n");
                    scanf("%d",&item);
                    first = orderList(first,item);
```

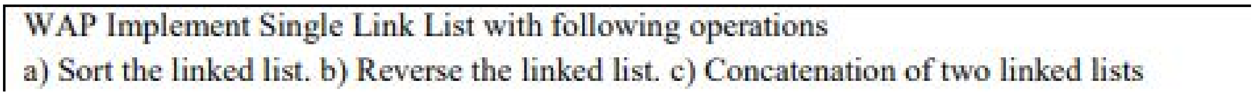


```c
                    break;
            case 6: display(first);
                    break;
            case 7: printf("Enter item to be inserted: \n");
                    scanf("%d",&item);
                    printf("Enter position to be inserted: \n");
                    scanf("%d",&pos);
                    first = insertPosition(item,pos,first);
                    break;
            case 8: printf("Enter position to be deleted: \n");
                    scanf("%d",&pos);
                    first = deletePosition(pos,first);
                    break;
            case 9: exit(0);
                    break;

            default: printf("Enter correct option!\n");
        }
    }
}
```



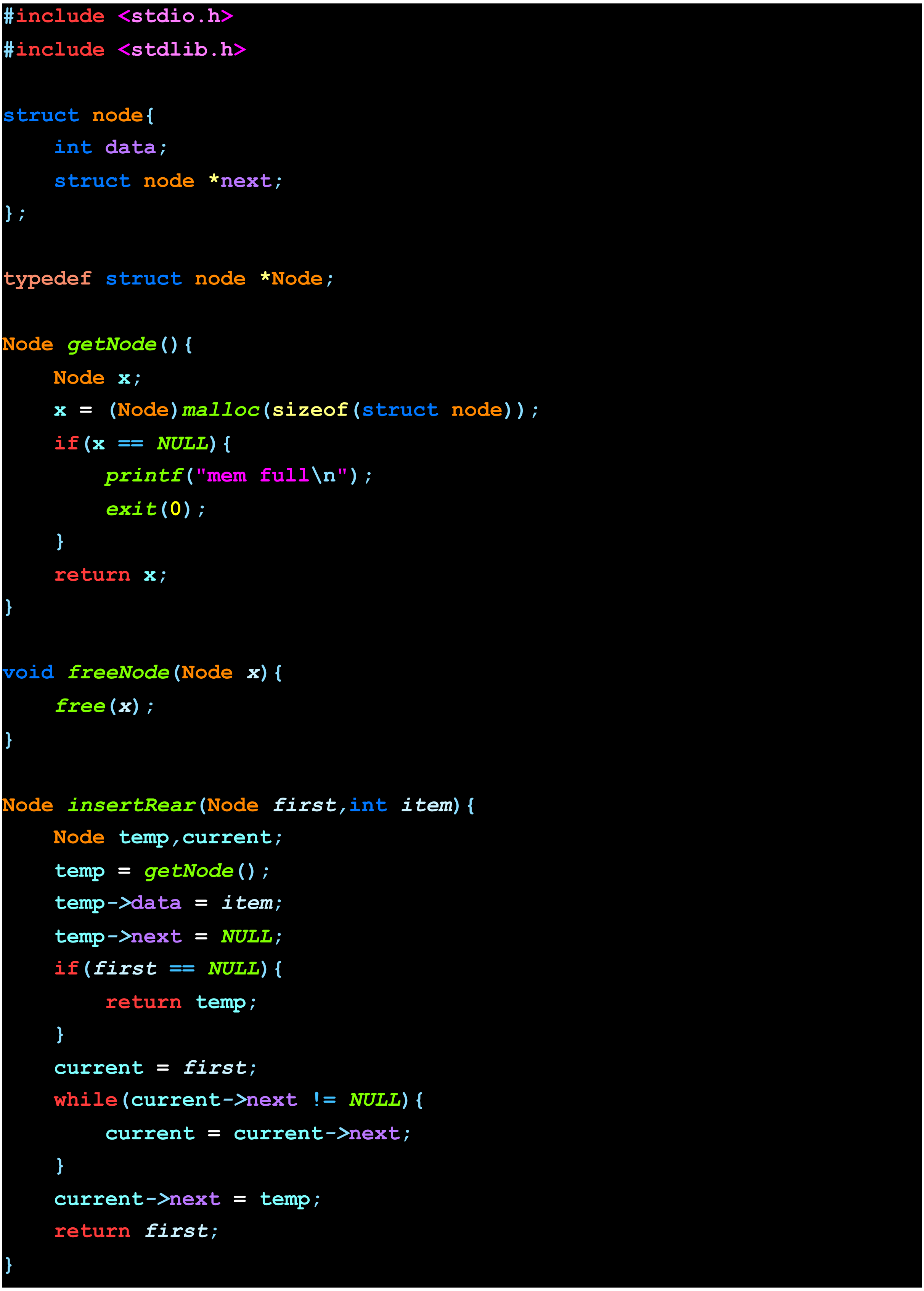
```
5. Order List
6. Display
7. Insert at
8. Delete at
9. Exit
Enter Your choice: 4

item deleted at rear end is 10

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Order List
6. Display
7. Insert at
8. Delete at
9. Exit
Enter Your choice: 6

30 20

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Order List
6. Display
7. Insert at
8. Delete at
9. Exit
Enter Your choice:
```

7.



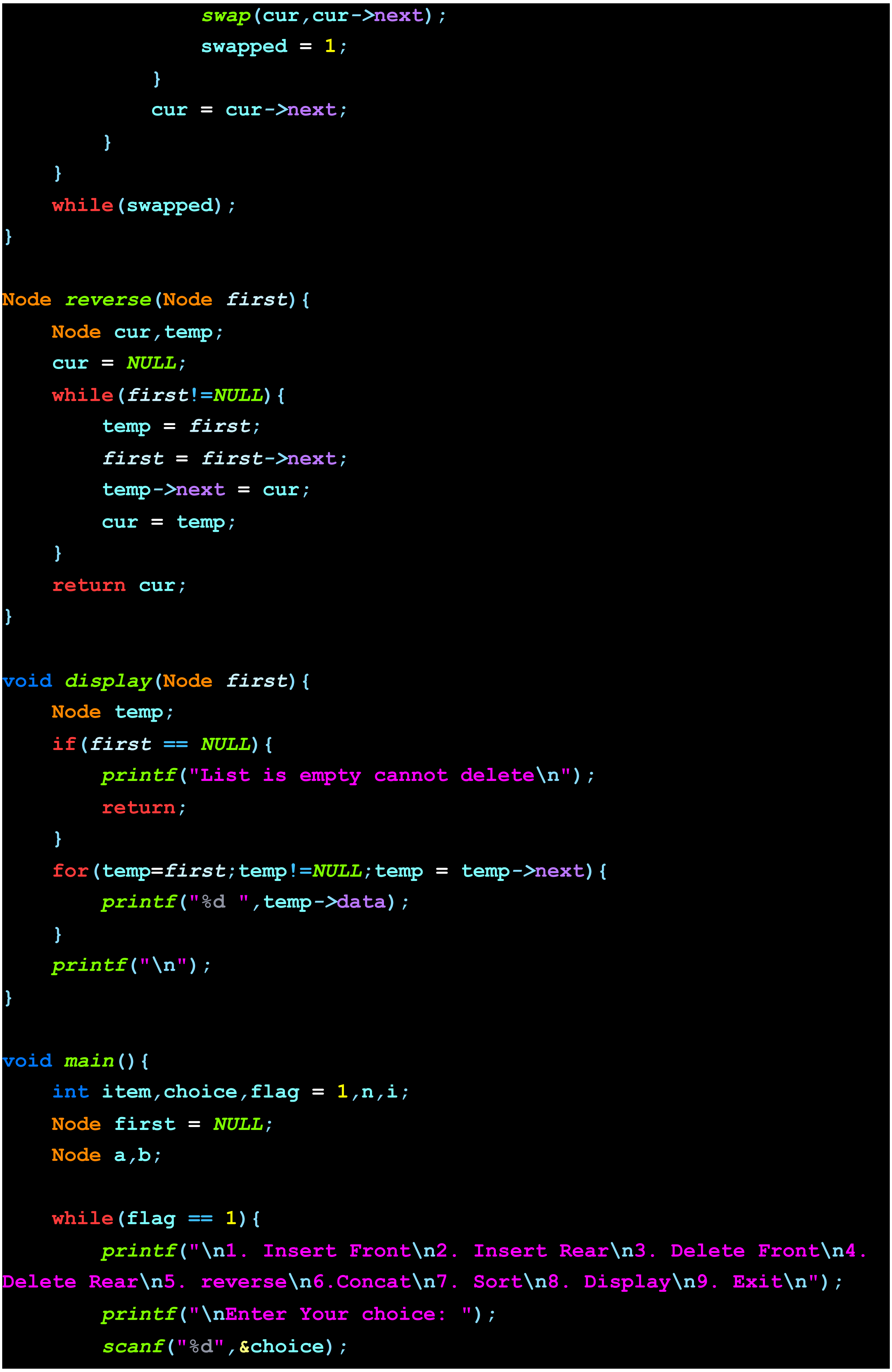
WAP Implement Single Link List with following operations

a) Sort the linked list. b) Reverse the linked list. c) Concatenation of two linked lists

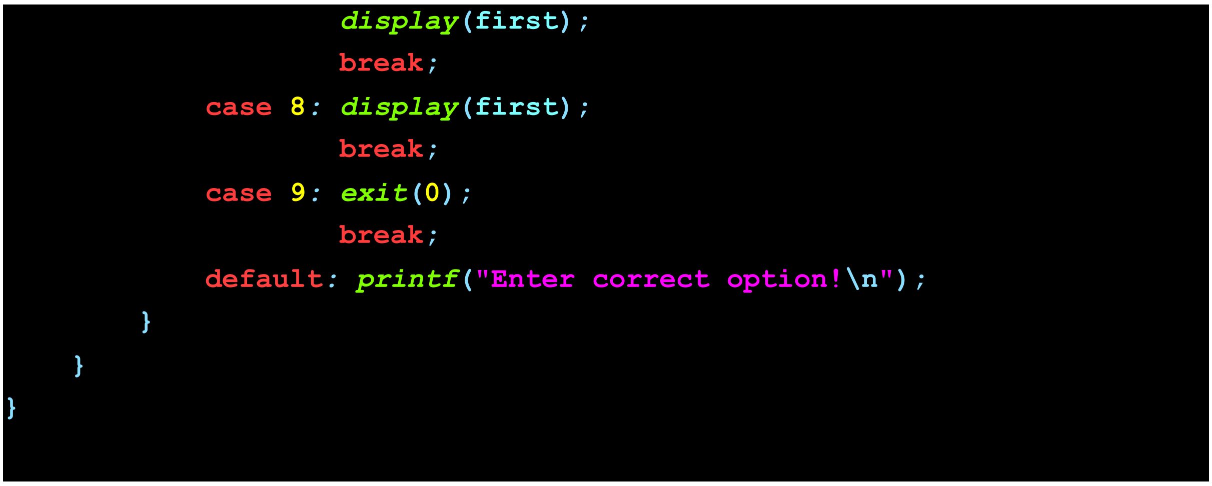


```
#include <stdio.h>
#include <stdlib.h>

struct node{
    int data;
    struct node *next;
};

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 insertRear(Node first,int item){
    Node temp,current;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    current = first;
    while(current->next != NULL){
        current = current->next;
    }
    current->next = temp;
    return first;
}
```

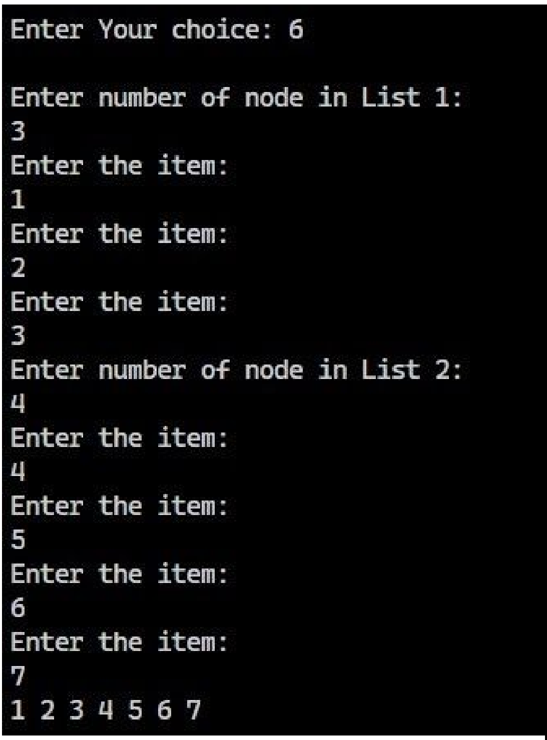
```
Node insertFront(Node first,int item){
    Node temp;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    temp->next = first;
    first = temp;
    return first;
}


Node deleteRear(Node first){
    Node current,previous;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    if(first->next == NULL){
        printf("Item deleted is %d\n",first->data);
        free(first);
        return NULL;
    }
    previous = NULL;
    current = first;
    while(current->next != NULL){
        previous = current;
        current = current->next;
    }
    printf("item deleted at rear end is %d\n",current->data);
    free(current);
    previous->next = NULL;
    return first;
}

Node deleteFront(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    temp = first;
    temp = temp->next;
    printf("Item delted at front end is %d\n",first->data);
    free(first);
    return temp;
}

Node concat(Node first, Node second){
    Node cur;
    if(first == NULL){
        return second;
    }
    if(second == NULL){
        return first;
    }
    cur = first;
    while(cur->next != NULL){
        cur = cur->next;
    }
    cur->next = second;
    return first;
}
Node swap(Node a, Node b){
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
void sort(Node first){
    int swapped, i;
    Node cur;

    if(first == NULL){
        printf("List is empty\n");
        return;
    }

    do{
        swapped = 0;
        cur = first;

        while(cur->next != NULL){
            if(cur->data > cur->next->data){
```



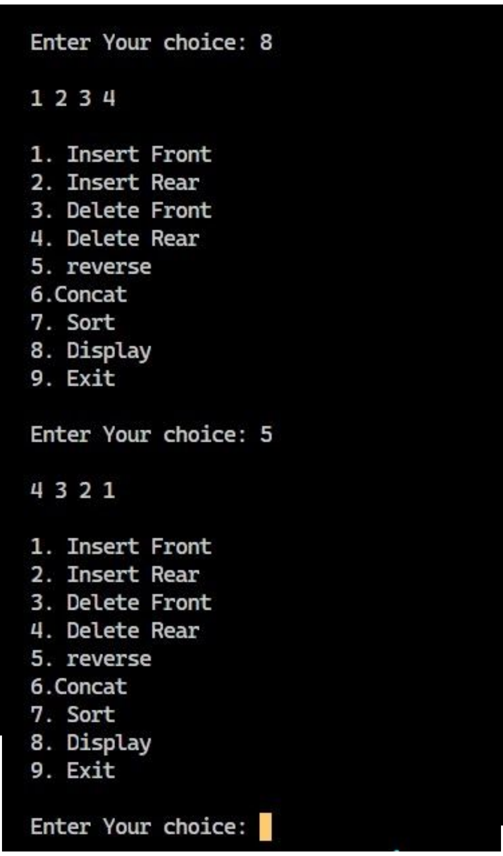
```
                swap(cur,cur->next);
                swapped = 1;
            }
            cur = cur->next;
        }
    }
    while(swapped);
}

Node reverse(Node first){
    Node cur,temp;
    cur = NULL;
    while(first!=NULL){
        temp = first;
        first = first->next;
        temp->next = cur;
        cur = temp;
    }
    return cur;
}

void display(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return;
    }
    for(temp=first;temp!=NULL;temp = temp->next){
        printf("%d ",temp->data);
    }
    printf("\n");
}

void main(){
    int item,choice,flag = 1,n,i;
    Node first = NULL;
    Node a,b;

    while(flag == 1){
        printf("\n1. Insert Front\n2. Insert Rear\n3. Delete Front\n4.
Delete Rear\n5. reverse\n6.Concat\n7. Sort\n8. Display\n9. Exit\n");
        printf("\nEnter Your choice: ");
        scanf("%d",&choice);
        printf("\n");
        switch(choice){
            case 1: printf("Enter the item to be inserted at front:
\n");
                    scanf("%d",&item);
                    first = insertFront(first,item);
                    break;
            case 2: printf("Enter the item to be inserted in the rear:
\n");
                    scanf("%d",&item);
                    first = insertRear(first,item);
                    break;
            case 3: first = deleteFront(first);
                    break;
            case 4: first = deleteRear(first);
                    break;
            case 5: first = reverse(first);
                    display(first);
                    break;
            case 6: {
                printf("Enter number of node in List 1: \n");
                scanf("%d",&n);
                a = NULL;
                for(i=0;i<n;i++){
                    printf("Enter the item: \n");
                    scanf("%d",&item);
                    a = insertRear(a,item);
                }

                printf("Enter number of node in List 2: \n");
                scanf("%d",&n);
                b = NULL;
                for(i=0;i<n;i++){
                    printf("Enter the item: \n");
                    scanf("%d",&item);
                    b = insertRear(b,item);
                }

                a = concat(a,b);
                display(a);
                break;
            }
            case 7: sort(first);
```



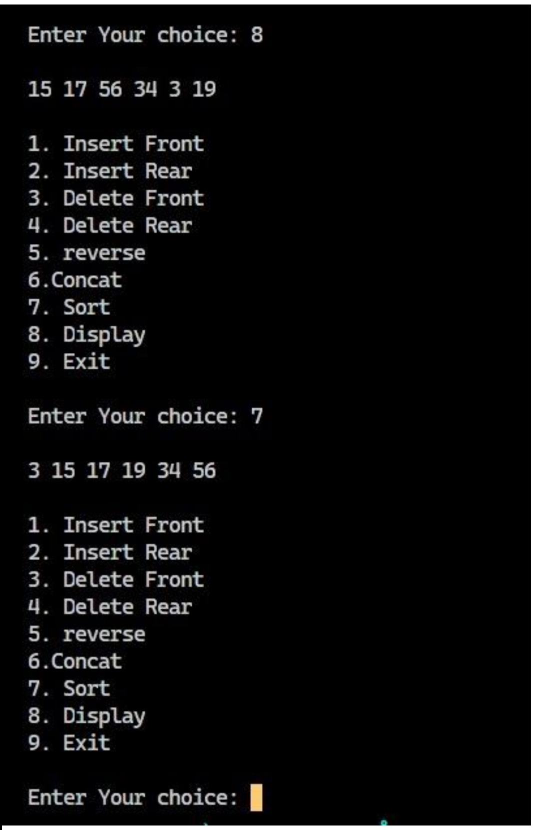
```
                    display(first);
                    break;
            case 8: display(first);
                    break;
            case 9: exit(0);
                    break;
            default: printf("Enter correct option!\n");
        }
    }
}
```



```
Enter Your choice: 6

Enter number of node in List 1:
3
Enter the item:
1
Enter the item:
2
Enter the item:
3
Enter number of node in List 2:
4
Enter the item:
4
Enter the item:
5
Enter the item:
6
Enter the item:
7
1 2 3 4 5 6 7
```



```
Enter Your choice: 8

1 2 3 4

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. reverse
6.Concat
7. Sort
8. Display
9. Exit

Enter Your choice: 5

4 3 2 1

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. reverse
6.Concat
7. Sort
8. Display
9. Exit

Enter Your choice:
```



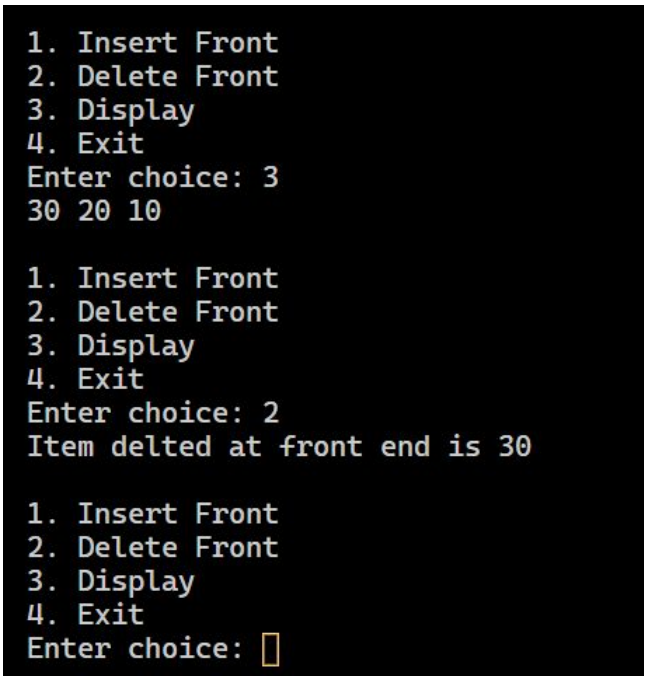
```
Enter Your choice: 8

15 17 56 34 3 19

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. reverse
6.Concat
7. Sort
8. Display
9. Exit

Enter Your choice: 7

3 15 17 19 34 56

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. reverse
6.Concat
7. Sort
8. Display
9. Exit

Enter Your choice:
```

8.

WAP to implement Stack & Queues using Linked Representation

Stack

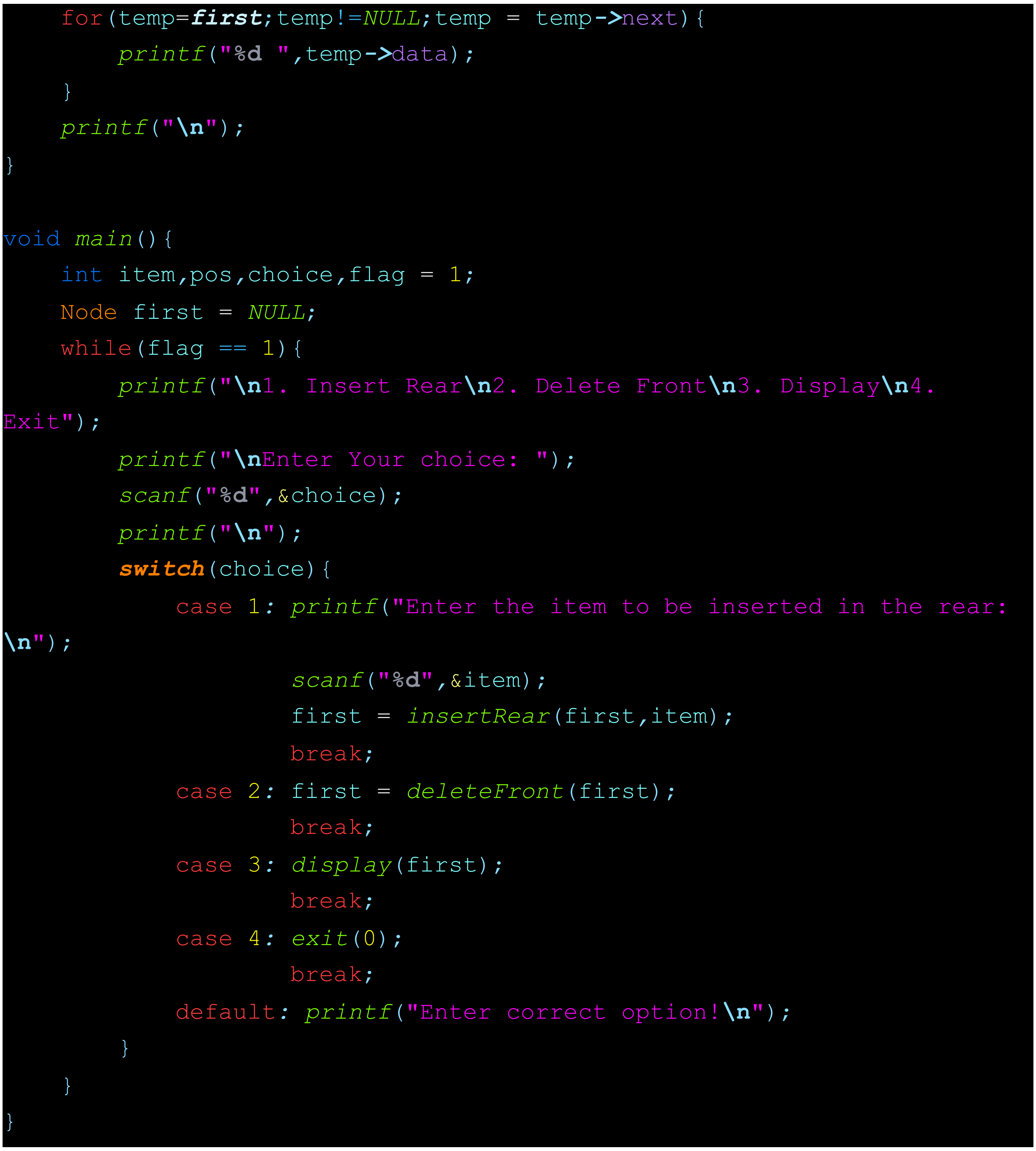
```
#include <stdio.h>
#include <stdlib.h>

struct node{
    int data;
    struct node *next;
    struct node *prev;
};

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;
}

Node insertFront(int item, Node head){
    Node temp,cur;
    temp = getNode();
    temp->data = item;
    cur = head->next;
    head->next = temp;
    temp->prev = head;
    temp->next = cur;
    cur->prev = temp;
    return head;
}

Node deleteFront(Node head){
    Node cur,next;
    if(head->next == head){
        printf("List is empty\n");
        return head;
    }
    cur = head->next;
    next= cur->next;
    head->next = next;
    next->prev = head;
    printf("deleted Node with data: %d",cur->data);
    free(cur);
    return head;
}

void display(Node head){
    Node temp;
    if(head->next == head){
        printf("List is empty\n");
        return;
    }
    printf("Contents of DLL: \n");
    temp = head->next;
    while(temp != head){
        printf("%d ",temp->data);
        temp = temp->next;
    }
    printf("\n");
}

void main(){
    int choice,item,flag = 1,key;
    Node head;
    head = getNode();
    head->next = head;
    head->prev = head;

    while(flag == 1){
        printf("\n1. Insert Front\n2. Delete Front\n3. Display\n4.
Exit\n");
        printf("Enter choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter item: \n");
                    scanf("%d",&item);
                    head = insertFront(item,head);
                    break;
            case 2: head = deleteFront(head);
                    break;
            case 3: display(head);
                    break;
            default: exit(0);
        }
    }
}
```



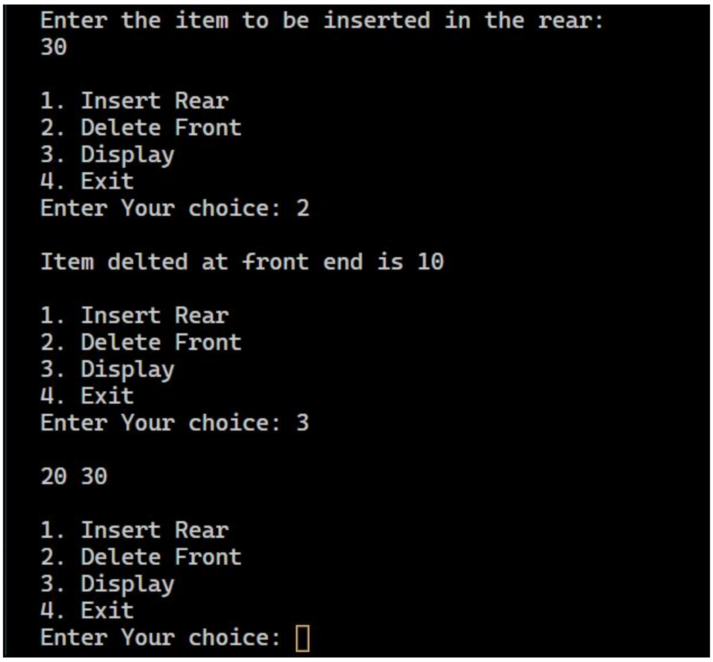
```
1. Insert Front
2. Delete Front
3. Display
4. Exit
Enter choice: 3
30 20 10

1. Insert Front
2. Delete Front
3. Display
4. Exit
Enter choice: 2
Item delted at front end is 30

1. Insert Front
2. Delete Front
3. Display
4. Exit
Enter choice: 
```

Queue

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

struct node{
    int data;
    struct node *next;
};

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 insertRear(Node first,int item){
    Node temp,current;
    temp = getNode();
    temp->data = item;
    temp->next = NULL;
    if(first == NULL){
        return temp;
    }
    current = first;
    while(current->next != NULL){
        current = current->next;
    }
    current->next = temp;
    return first;
}

Node deleteFront(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return first;
    }
    temp = first;
    temp = temp->next;
    printf("Item delted at front end is %d\n",first->data);
    free(first);
    return temp;
}

void display(Node first){
    Node temp;
    if(first == NULL){
        printf("List is empty cannot delete\n");
        return;
    }
```

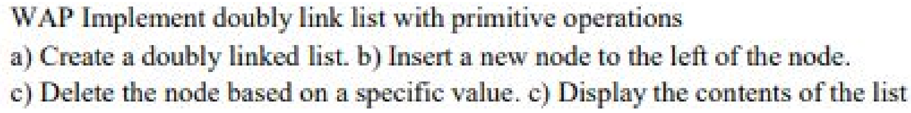


```
    for(temp=first;temp!=NULL;temp = temp->next){
        printf("%d ",temp->data);
    }
    printf("\n");
}

void main(){
    int item,pos,choice,flag = 1;
    Node first = NULL;
    while(flag == 1){
        printf("\n1. Insert Rear\n2. Delete Front\n3. Display\n4.
Exit");
        printf("\nEnter Your choice: ");
        scanf("%d",&choice);
        printf("\n");
        switch(choice){
            case 1: printf("Enter the item to be inserted in the rear:
\n");
                    scanf("%d",&item);
                    first = insertRear(first,item);
                    break;
            case 2: first = deleteFront(first);
                    break;
            case 3: display(first);
                    break;
            case 4: exit(0);
                    break;
            default: printf("Enter correct option!\n");
        }
    }
}
```



```
Enter the item to be inserted in the rear:
30

1. Insert Rear
2. Delete Front
3. Display
4. Exit
Enter Your choice: 2

Item delted at front end is 10

1. Insert Rear
2. Delete Front
3. Display
4. Exit
Enter Your choice: 3

20 30

1. Insert Rear
2. Delete Front
3. Display
4. Exit
Enter Your choice: 
```

9.

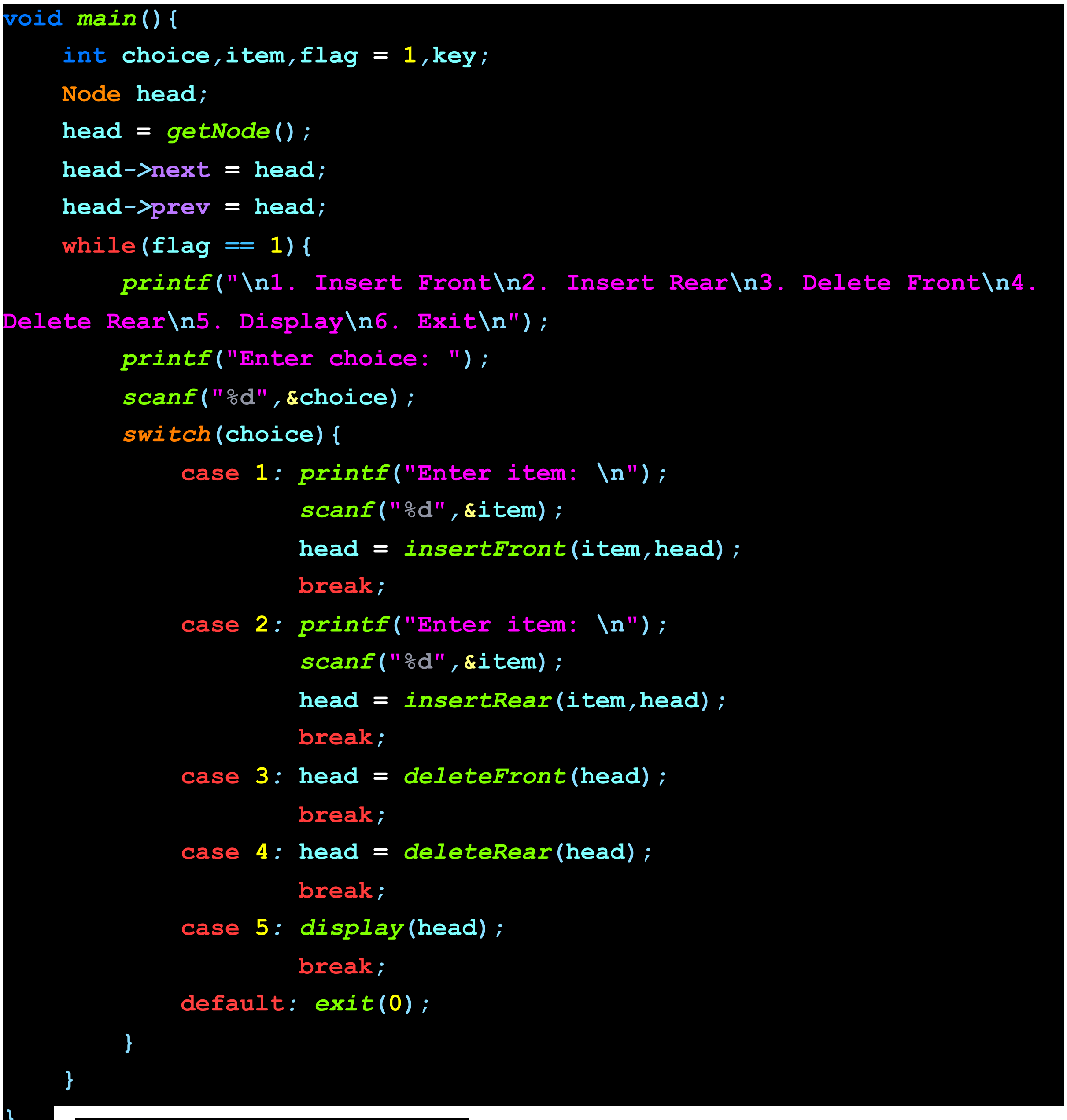


WAP Implement doubly link list with primitive operations

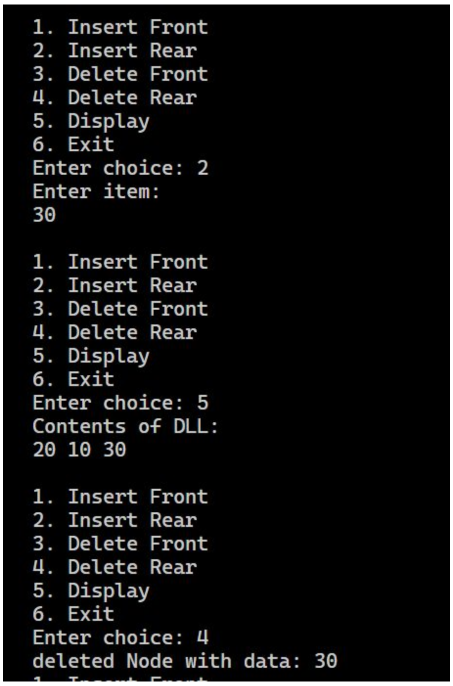
a) Create a doubly linked list. b) Insert a new node to the left of the node.

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 data;
    struct node *next;
    struct node *prev;
};
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;
}
Node insertFront(int item, Node head){
    Node temp,cur;
    temp = getNode();
    temp->data = item;
    cur = head->next;
    head->next = temp;
    temp->prev = head;
    temp->next = cur;
    cur->prev = temp;
    return head;
}
Node insertRear(int item, Node head){
    Node temp,cur;
    temp = getNode();
    temp->data = item;
    cur = head->prev;
    temp->next = head;
    temp->prev = cur;
    cur->next = temp;
    head->prev = temp;
    return head;
}
Node deleteFront(Node head){
    Node cur,next;
    if(head->next == head){
        printf("List is empty\n");
        return head;
    }
    cur = head->next;
    next= cur->next;
    head->next = next;
    next->prev = head;
    printf("deleted Node with data: %d",cur->data);
    free(cur);
    return head;
}
Node deleteRear(Node head){
    Node cur,prev;
    if(head->next == head){
        printf("List is empty\n");
        return head;
    }
    cur = head->prev;
    prev = cur->prev;
    head->prev = prev;
    prev->next = head;
    printf("deleted Node with data: %d",cur->data);
}
void display(Node head){
    Node temp;
    if(head->next == head){
        printf("List is empty\n");
        return;
    }
    printf("Contents of DLL: \n");
    temp = head->next;
    while(temp != head){
        printf("%d ",temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

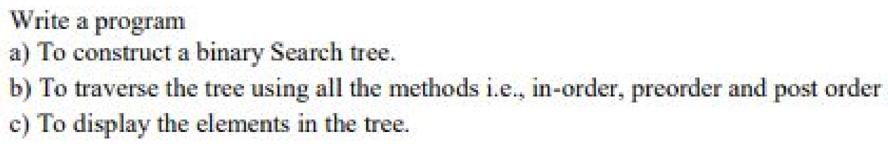


```
void main(){
    int choice,item,flag = 1,key;
    Node head;
    head = getNode();
    head->next = head;
    head->prev = head;
    while(flag == 1){
        printf("\n1. Insert Front\n2. Insert Rear\n3. Delete Front\n4. Delete Rear\n5. Display\n6. Exit\n");
        printf("Enter choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter item: \n");
                    scanf("%d",&item);
                    head = insertFront(item,head);
                    break;
            case 2: printf("Enter item: \n");
                    scanf("%d",&item);
                    head = insertRear(item,head);
                    break;
            case 3: head = deleteFront(head);
                    break;
            case 4: head = deleteRear(head);
                    break;
            case 5: display(head);
                    break;
            default: exit(0);
        }
    }
}
```



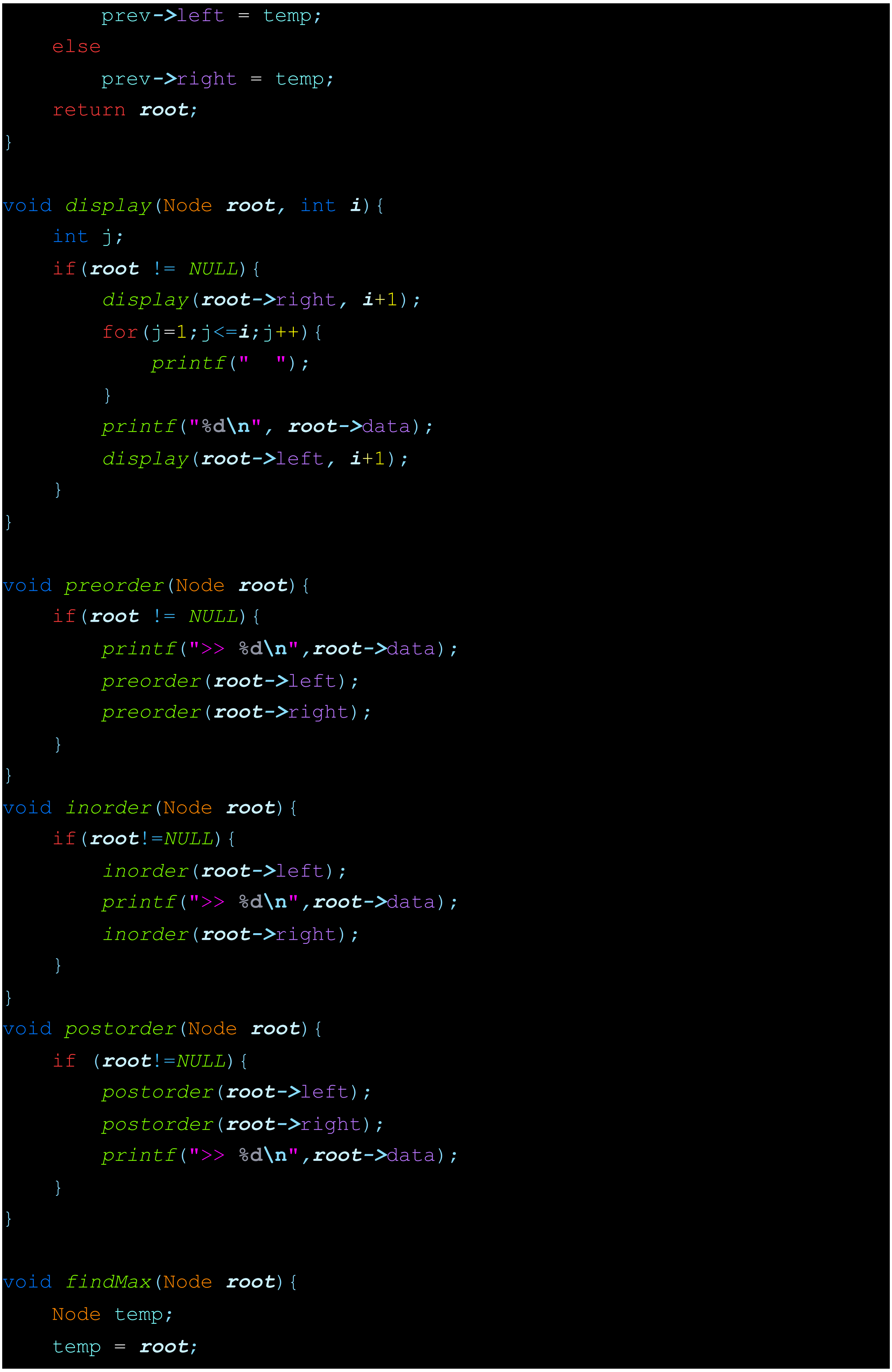
```
1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Display
6. Exit
Enter choice: 2
Enter item:
30

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Display
6. Exit
Enter choice: 5
Contents of DLL:
20 10 30

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Display
6. Exit
Enter choice: 4
deleted Node with data: 30
1. Insert Front
```

10.

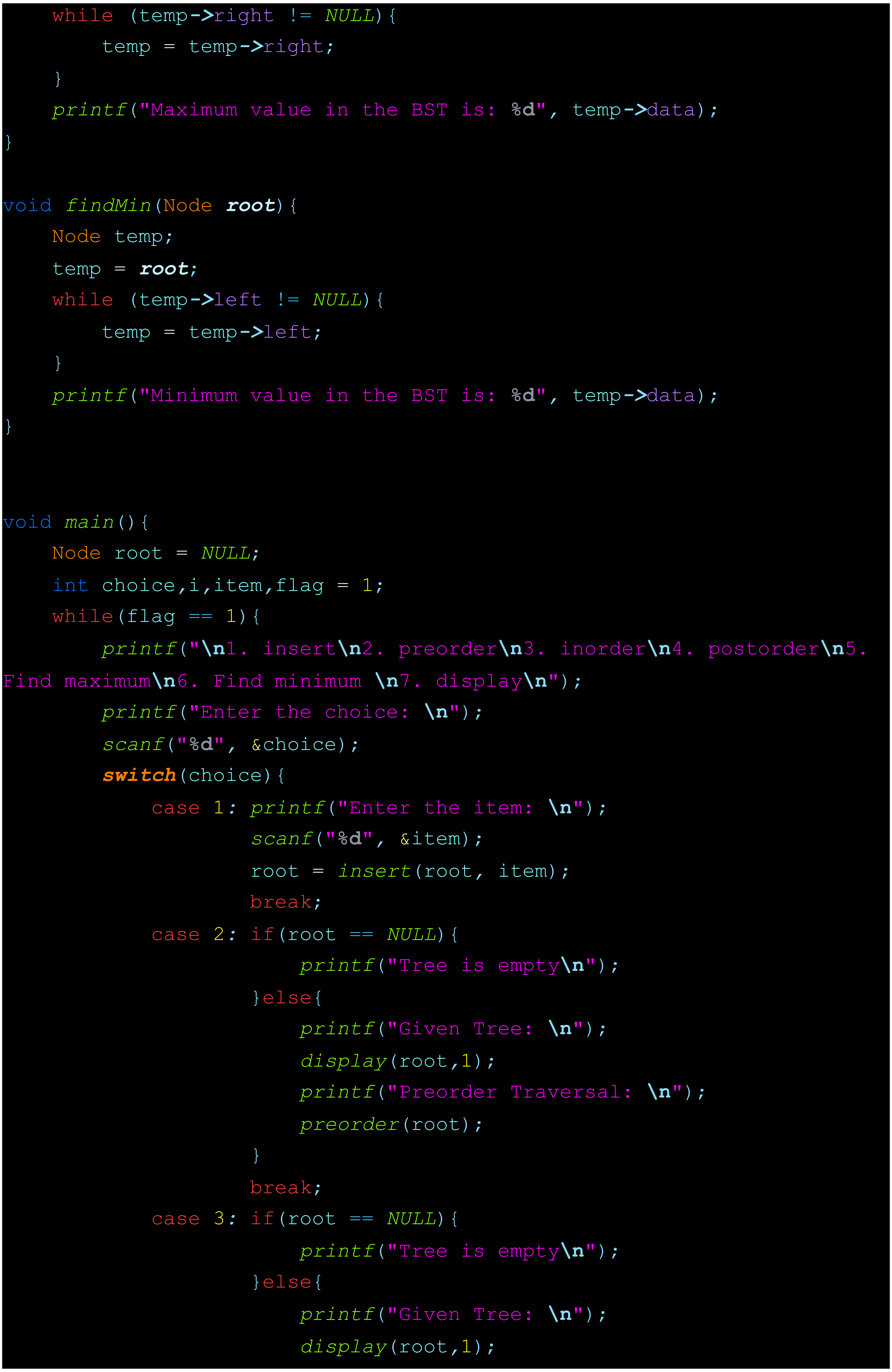


Write a program
a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e., in-order, preorder and post order
c) To display the elements in the tree.

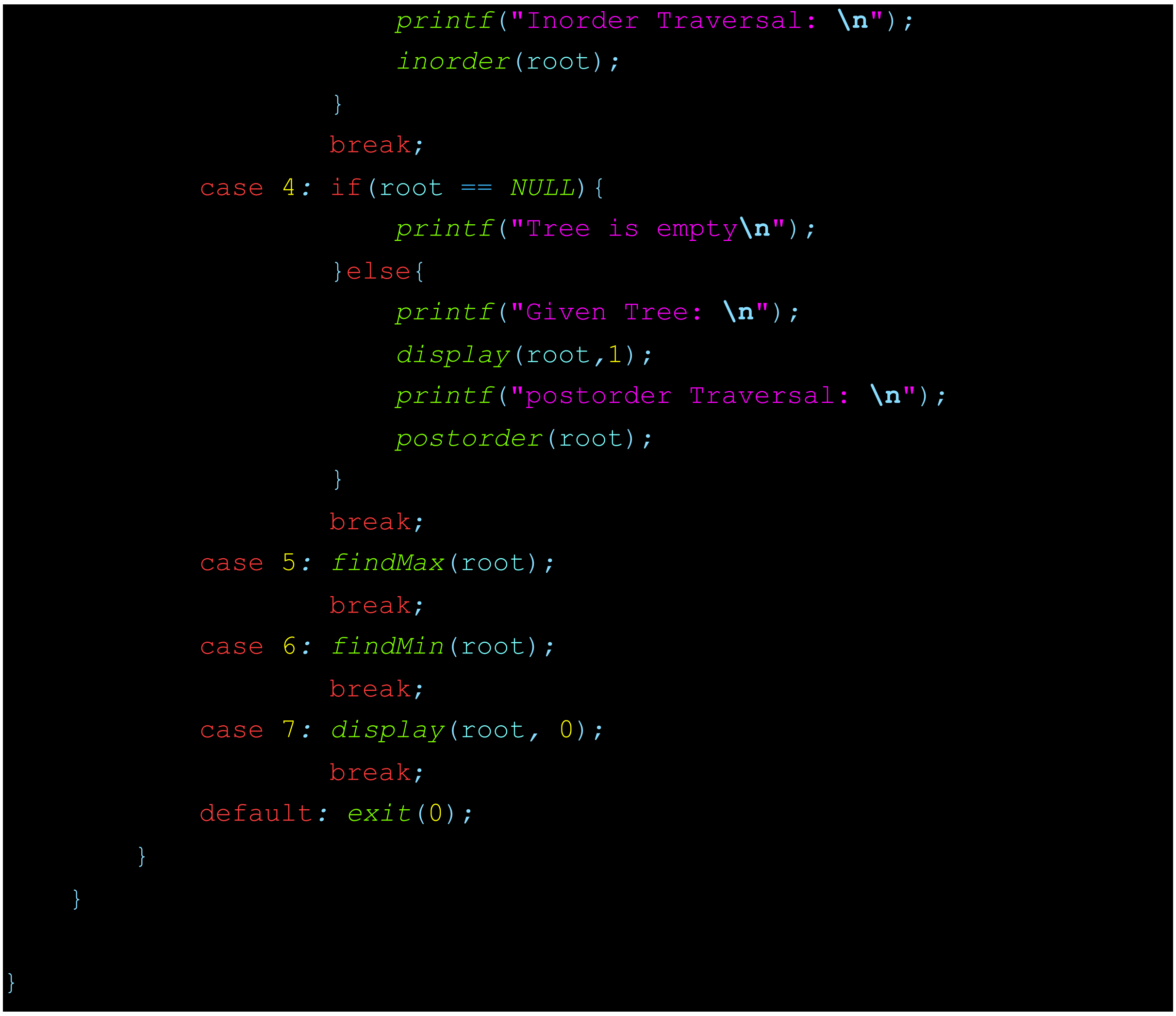
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node{
    int data;
    struct node *left;
    struct node *right;
};

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 insert(Node root, int data){
    Node temp,cur,prev;
    temp = getNode();
    temp->right = NULL;
    temp->left = NULL;
    temp->data = data;
    if(root == NULL){
        return temp;
    }
    prev = NULL;
    cur = root;
    while(cur != NULL){
        prev = cur;
        cur = (data < cur->data) ? cur->left : cur->right;
    }
    if(data < prev->data)
```



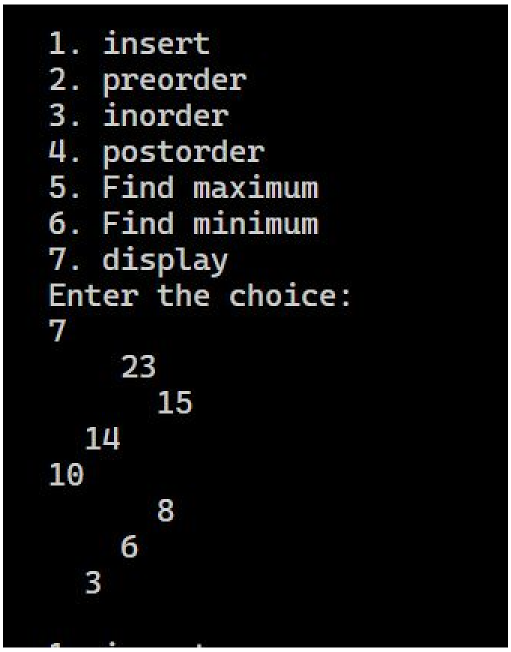
```
        prev->left = temp;
    else
        prev->right = temp;
    return root;
}

void display(Node root, int i){
    int j;
    if(root != NULL){
        display(root->right, i+1);
        for(j=1;j<=i;j++){
            printf("  ");
        }
        printf("%d\n", root->data);
        display(root->left, i+1);
    }
}

void preorder(Node root){
    if(root != NULL){
        printf(">> %d\n",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(Node root){
    if(root!=NULL){
        inorder(root->left);
        printf(">> %d\n",root->data);
        inorder(root->right);
    }
}
void postorder(Node root){
    if (root!=NULL){
        postorder(root->left);
        postorder(root->right);
        printf(">> %d\n",root->data);
    }
}

void findMax(Node root){
    Node temp;
    temp = root;
```



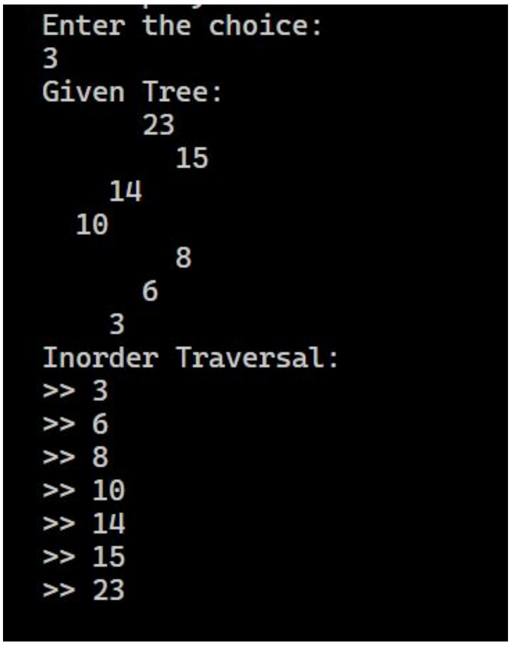
```
    while (temp->right != NULL){
        temp = temp->right;
    }
    printf("Maximum value in the BST is: %d", temp->data);
}

void findMin(Node root){
    Node temp;
    temp = root;
    while (temp->left != NULL){
        temp = temp->left;
    }
    printf("Minimum value in the BST is: %d", temp->data);
}


void main(){
    Node root = NULL;
    int choice,i,item,flag = 1;
    while(flag == 1){
        printf("\n1. insert\n2. preorder\n3. inorder\n4. postorder\n5.
Find maximum\n6. Find minimum \n7. display\n");
        printf("Enter the choice: \n");
        scanf("%d", &choice);
        switch(choice){
            case 1: printf("Enter the item: \n");
                    scanf("%d", &item);
                    root = insert(root, item);
                    break;
            case 2: if(root == NULL){
                        printf("Tree is empty\n");
                    }else{
                        printf("Given Tree: \n");
                        display(root,1);
                        printf("Preorder Traversal: \n");
                        preorder(root);
                    }
                    break;
            case 3: if(root == NULL){
                        printf("Tree is empty\n");
                    }else{
                        printf("Given Tree: \n");
                        display(root,1);
```



```
                        printf("Inorder Traversal: \n");
                        inorder(root);
                    }
                    break;
            case 4: if(root == NULL){
                        printf("Tree is empty\n");
                    }else{
                        printf("Given Tree: \n");
                        display(root,1);
                        printf("postorder Traversal: \n");
                        postorder(root);
                    }
                    break;
            case 5: findMax(root);
                    break;
            case 6: findMin(root);
                    break;
            case 7: display(root, 0);
                    break;
            default: exit(0);
        }
    }

}
```



```
1. insert
2. preorder
3. inorder
4. postorder
5. Find maximum
6. Find minimum
7. display
Enter the choice:
7
    23
      15
  14
10
      8
    6
  3
```



```
Enter the choice:
3
Given Tree:
      23
        15
    14
  10
        8
      6
    3
Inorder Traversal:
>> 3
>> 6
>> 8
>> 10
>> 14
>> 15
>> 23
```