DATA STRUCTURES AND ALGORITHMS

SIDDHANTH U HEGDE

3RD SEM CSE

18GAEC9062

UVCE

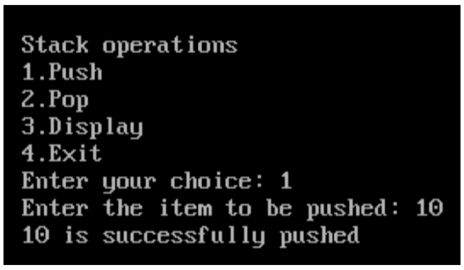
PROGRAMS:

1. Implement the following using arrays:
   a.) Stacks
   b.) Queues
2. Write a program to implement:
   a.) Tower of Hanoi using recursion
   b.) Insertion sort
3. Convert infix expression to prefix expression.
4. Implement Double Ended Queue using singly linked list.
5. Implement Singly Circular Linked List with header node.
6. Perform various operations in Doubly Linked List.
7. Create a binary tree and traverse inorder, preorder and postorder.
8. Perform insert and delete operations in binary search tree.
9. Evaluate expression tree using binary tree.
10. Create right in threaded binary tree.
11. Implement Hash tables.
12. Implement Hashing using open addressing.
13. Write all members of an array of structures to a file using fwrite(). Read the array from file and display on the screen.
14. Compare the contents of two files. Write the difference in another file.

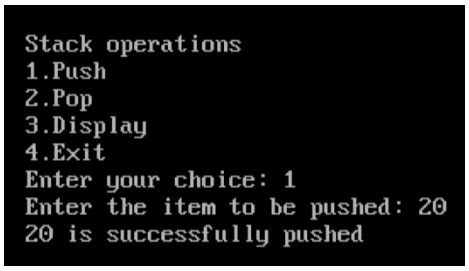
1 a. Implement stacks using arrays:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define SIZE 3
int stack[SIZE];

int push(int);
int pop(int);
void display(int);

void main()
{
 int top=-1,option;
 clrscr();
 while(1)
 {
  printf("\nStack operations\n");
  printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&option);
  switch(option)
  {
   case 1: top=push(top);
           getch();
           clrscr();
           break;
   case 2: top=pop(top);
           getch();
           clrscr();
           break;
   case 3: display(top);
           getch();
           clrscr();
           break;
   case 4: exit(0);   default: printf("Invalid choice\n");
  }
 }
}

int push(int top)
{
 int item;
 if(top==(SIZE-1))
 {
  printf("Stack is full\n");
  return top;
 }
 printf("Enter the item to be pushed: ");
 scanf("%d",&item);
 stack[++top]=item;
 printf("%d is successfully pushed\n",item);
 return top;
}

int pop(int top)
{
 if(top==-1)
 {
  printf("Stack is empty\n");
  return top;
 }
 printf("%d is successfully popped\n",stack[top]);
 top--;
 return top;
}

void display(int top)
{
 int i;
 if(top==-1)
  printf("Stack is empty\n");
 else
 {
  printf("Status of stack\n");
  for(i=top;i>=0;i--)
  printf("  %d\n",stack[i]);
 }
}
```

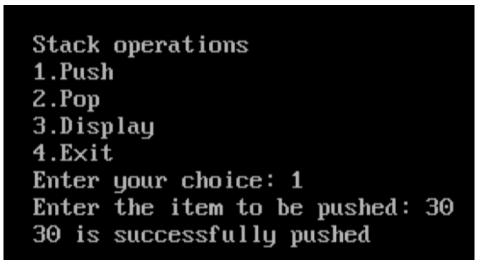
OUTPUT:



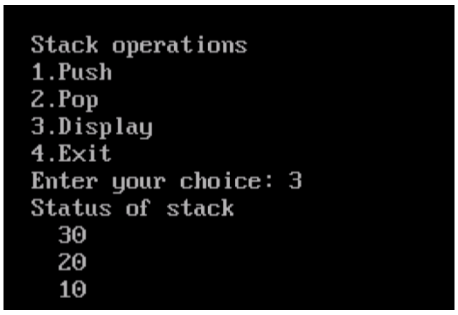
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the item to be pushed: 10
10 is successfully pushed
```



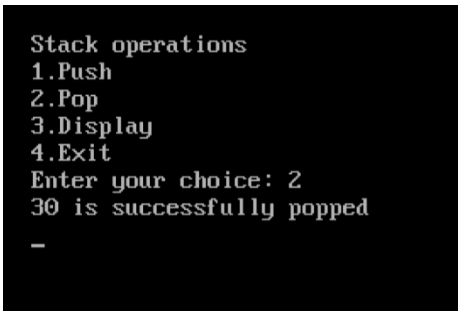
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the item to be pushed: 20
20 is successfully pushed
```



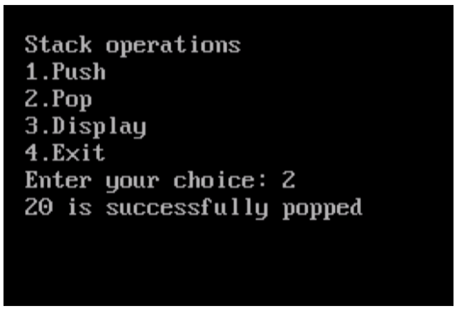
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 1
Enter the item to be pushed: 30
30 is successfully pushed
```



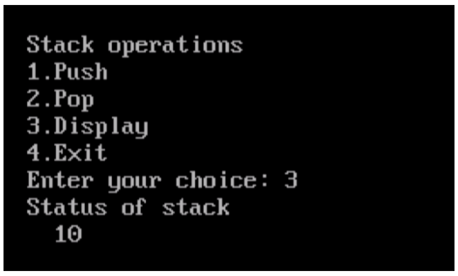
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Status of stack
  30
  20
  10
```



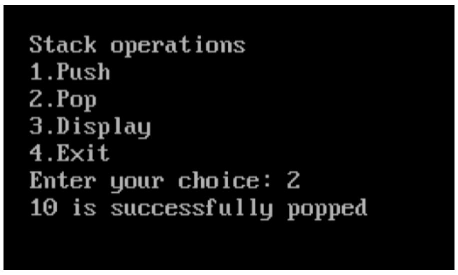
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
30 is successfully popped
_
```



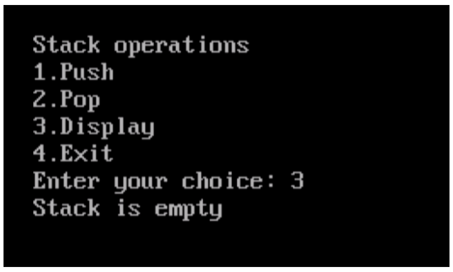
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
20 is successfully popped
```



```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Status of stack
  10
```



```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 2
10 is successfully popped
```



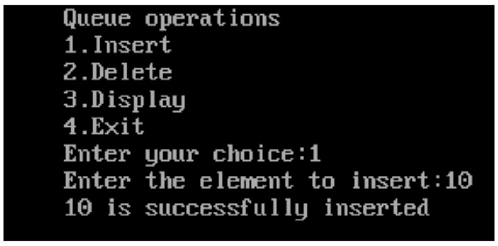
```
Stack operations
1.Push
2.Pop
3.Display
4.Exit
Enter your choice: 3
Stack is empty
```

1 b.Implement Queues using Arrays:

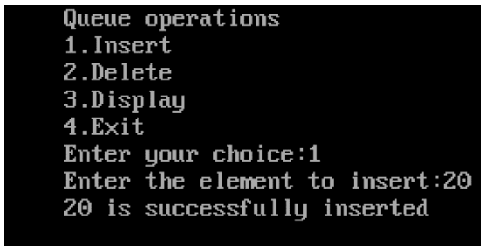
```
#include<stdio.h>
#include<conio.h>
#define SIZE 3
int rear=-1,front=-1,queue[20];

void main()
{
        int ch=1,option;
        clrscr();
        while(ch)
        {
          printf("Queue operations\n");
          printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
          printf("Enter your choice:");
          scanf("%d",&option);
          switch(option)
          {
        case 1:qinsert();
              getch();
              clrscr();
              break;
        case 2:qdelete();
              getch();
              clrscr();
              break;
        case 3:qdisplay();
              getch();
              clrscr();
              break;
        case 4:exit(0);
        default:printf("invalid choice\n");
        break;
                    }
        }
}

qinsert()
{
        int num;
        if(rear==(SIZE-1))
        {
         printf("Queue is full\n");
         return;
         }
          printf("Enter the element to insert:");
          scanf("%d",&num);
          queue[++rear]=num;
          printf("%d is successfully inserted\n",num);
          if(front==-1)
          front++;
          return;
 }

qdelete()
{
        if(front==-1)
        {
         printf("Queue is empty\n");
         return;
         }
         if(front==rear)
         {
         printf("Deleted element:%d",queue[front]);
         front=-1;
         rear=-1;
         return;
         }
         printf("Deleted element:%d",queue[front]);
```

```
          front++;
          return;
  }

  qdisplay()
  {
         int i;
         if(front==-1)
         {
          printf("queue is empty\n");
          return;
          }
          printf("status of queue is\n");
          for(i=front;i<=rear;i++)
            printf("%d\t",queue[i]);
            return;
     }
```

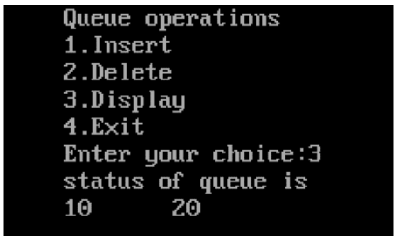
OUTPUT:



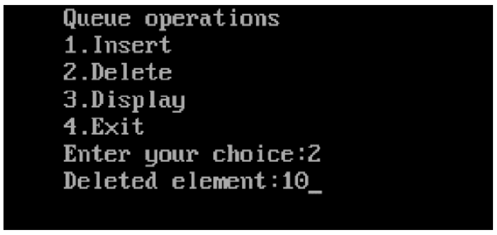
```
Queue operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the element to insert:10
10 is successfully inserted
```



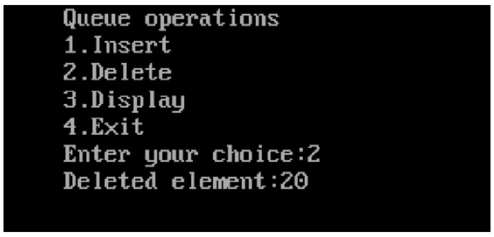
```
Queue operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:1
Enter the element to insert:20
20 is successfully inserted
```



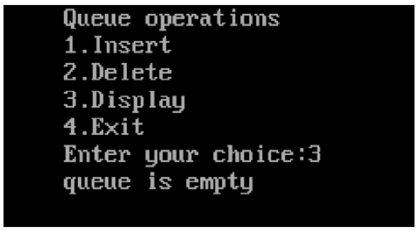
```
Queue operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:3
status of queue is
10      20
```



```
Queue operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:2
Deleted element:10_
```



```
Queue operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:2
Deleted element:20
```

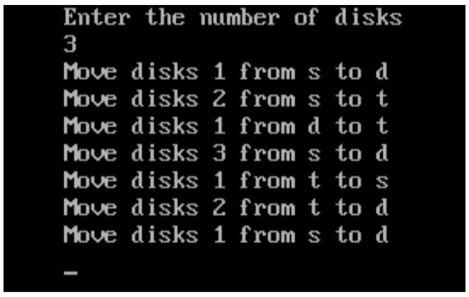


```
Queue operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice:3
queue is empty
```

2 a.Use recursive program to implement Tower of Hanoi:

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

void main()
{
 int n;
 clrscr();
 printf("Enter the number of disks\n");
 scanf("%d",&n);
 tower(n,'s','d','t');
 getch();
}
tower(int n,char source,char dest,char temp)
{
 if(n>0)
 {
 tower(n-1,source,temp,dest);
 printf("Move disks %d from %c to %c\n",n,source,dest);
 tower(n-1,temp,dest,source);
 }
 return;
}
```

OUTPUT:

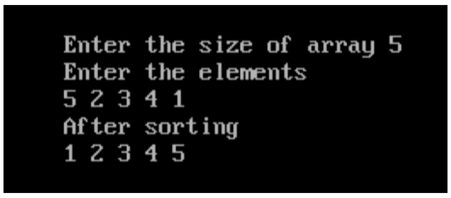


```
Enter the number of disks
3
Move disks 1 from s to d
Move disks 2 from s to t
Move disks 1 from d to t
Move disks 3 from s to d
Move disks 1 from t to s
Move disks 2 from t to d
Move disks 1 from s to d
_
```

2 b. Write a program to implement insertion sort:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
        int a[20],n,k,i,j;
        clrscr();
        printf("\nEnter the size of array");
        scanf("%d",&n);
        printf("Enter the elements\n");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        for(i=1;i<n;i++)
        {
         k=a[i];
         for(j=i-1;j>=0 && k<a[j];j--)
         a[j+1]=a[j];
         a[j+1]=k;
        }
        printf("After sorting\n");
        for(i=0;i<n;i++)
        printf("%d ",a[i]);
        getch();
}
```

OUTPUT:



```
Enter the size of array 5
Enter the elements
5 2 3 4 1
After sorting
1 2 3 4 5
```

3. Write a program to convert a given infix expression to prefix:

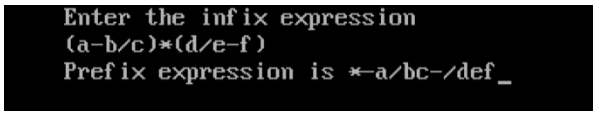
```
#include<stdio.h>
#include<conio.h>
#include<string.h>

int h=0,top=-1,k=0,i=0,j=0,length;
char symbol,infix[20],postfix[20],stack[20];
```

```
void infixtopostfix();
int preceed(char);

void main()
{
int len;
printf("Enter infix expression:");
scanf("%s",infix);
len=strlen(infix)-1;
for(h=0;h<=len/2;h++)
  {
   char temp;
   temp=infix[h];
   infix[h]=infix[len-h];
   infix[len-h]=temp;
  }
for(h=0;h<strlen(infix);h++)
  {
   if(infix[h]==')')
      {
       infix[h]='(';
      }
   else if(infix[h]=='(')
      {
       infix[h]=')';
      }
  }
infixtopostfix();
printf("Pre-fix expression is:  ");
for(k=strlen(postfix)-1;k>=0;k--)
  {
   printf("%c",postfix[k]);
  }
getch();
}


void push(char symbol)
{
 stack[++top]=symbol;
 return;
}

char pop()
{
 char temp;
 temp=stack[top--];
 return temp;
}

void infixtopostfix()
{
 char temp;
 length=strlen(infix);
 push('#');
 while(i<length)
```

```
  {
   symbol=infix[i];
   switch(symbol)
     {
      case '(':
                 push(symbol);
                 break;
      case ')':
                 temp=pop();
                 while(temp!='(')
            {
             postfix[j++]=temp;
             temp=pop();
            }
                 break;
      case '/':
      case '*':
      case '^':
      case '+':
      case '-':
                 while(preceed(stack[top])==preceed(symbol))
           {
            temp=pop();
            postfix[j++]=temp;
           }
                 push(symbol);
                 break;
      default:
                postfix[j++]=symbol;
     }
   i++;
  }
 while(top>0)
  {
   postfix[j++]=pop();
   //return;
  }
}

int preceed(char symbol)
 {
  int p;
  switch(symbol)
   {
    case '^':p=3;
               break;
    case '/':
    case '*':p=2;
               break;
    case '+':
    case '-':p=1;
               break;
    case '(':
    case ')':p=0;
               break;
    case '#':p=-1;
```

```
            break;
  }
 return(p);
}
```

OUTPUT:



```
Enter the infix expression
(a-b/c)*(d/e-f)
Prefix expression is *-a/bc-/def_
```

4.Implement double ended queue using singly linked list:

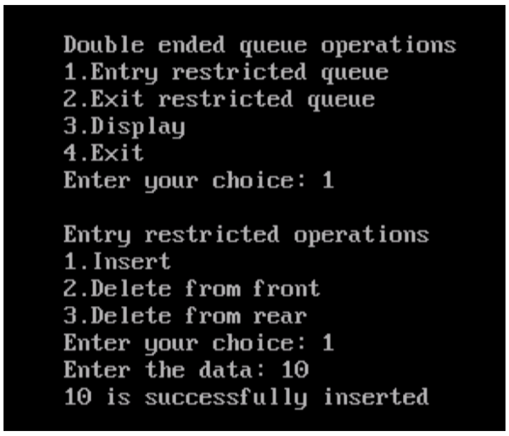
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *link;
};
typedef struct node *NODE;

NODE insert_rear(NODE);
NODE insert_front(NODE);
NODE delete_rear(NODE);
NODE delete_front(NODE);
NODE getnode();
void display(NODE);

void main()
{
 NODE first=NULL;
 int option;
 clrscr();
 while(1)
 {
  printf("\nDouble ended queue operations\n");
  printf("1.Entry restricted queue\n2.Exit restricted queue\n3.Display\n4.Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&option);
  switch(option)
  {
   case 1: printf("\nEntry restricted operations\n");
           printf("1.Insert\n2.Delete from front\n3.Delete from rear\n");
           printf("Enter your choice: ");
           scanf("%d",&option);
           switch(option)
           {
            case 1: first=insert_rear(first);
            getch();
            clrscr();
            break;
            case 2: first=delete_front(first);
            getch();
```

```
             clrscr();
             break;
             case 3: first=delete_rear(first);
             getch();
             clrscr();
             break;
             default: printf("Invalid choice\n");
            }
            break;
   case 2: printf("\nExit restricted operations\n");
            printf("1.Delete\n2.Insert from front\n3.Insert from rear\n");
            printf("Enter your choice: ");
            scanf("%d",&option);
            switch(option)
            {
             case 1: first=delete_front(first);
             getch();
             clrscr();
             break;
             case 2: first=insert_front(first);
             getch();
             clrscr();
             break;
             case 3: first=insert_rear(first);
             getch();
             break;
             default: printf("Invalid choice\n");
            }
            break;
   case 3: display(first);
             getch();
             clrscr();
             break;
   case 4: exit(0);
   default: printf("Invalid choice\n");
  }
 }
}

NODE getnode()
{
 NODE temp;
 temp=(NODE)malloc(sizeof(struct node));
 return temp;
}

NODE insert_front(NODE first)
{
 NODE temp;
 temp=getnode();
 printf("Enter the data: ");
 scanf("%d",&temp->data);
 temp->link=first;
 first=temp;
 printf("%d is successfully inserted\n",temp->data);
 return first;
```

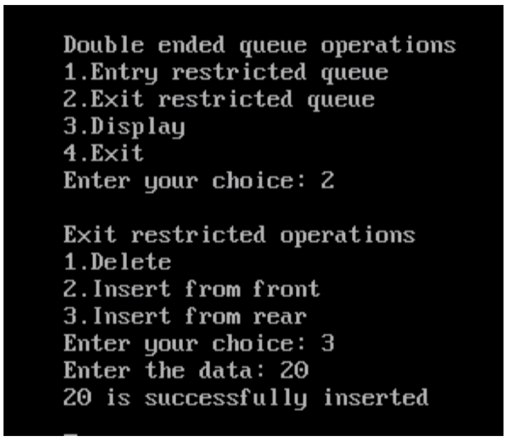
```
}

NODE delete_front(NODE first)
{
 NODE temp;
 if(first==NULL)
 {
  printf("No nodes present");
  return first;
 }
 temp=first;
 first=first->link;
 printf("%d is successfully deleted\n",temp->data);
 free(temp);
 return first;
}

NODE insert_rear(NODE first)
{
 NODE temp,ptr;
 temp=getnode();
 printf("Enter the data: ");
 scanf("%d",&temp->data);
 printf("%d is successfully inserted\n",temp->data);
 temp->link=NULL;
 if(first==NULL)
 {
  first=temp;
  return first;
 }
 ptr=first;
 while(ptr->link!=NULL)
 ptr=ptr->link;
 ptr->link=temp;
 return first;
}

NODE delete_rear(NODE first)
{
 NODE ptr,prev;
 if(first==NULL)
 {
  printf("No nodes present\n");
  return first;
 }
 ptr=first;
 while(ptr->link!=NULL)
 {
  prev=ptr;
  ptr=ptr->link;
 }
 printf("%d is successfully removed\n",ptr->data);
 prev->link=NULL;
 free(ptr);
 return first;
}
```

```
void display(NODE first)
{
 NODE temp;
 if(first==NULL)
  printf("No nodes present\n");
 else
 {
 for(temp=first;temp->link!=NULL;temp=temp->link)
  printf("%d=>",temp->data);
  printf("%d=>",temp->data);
 }
}
```

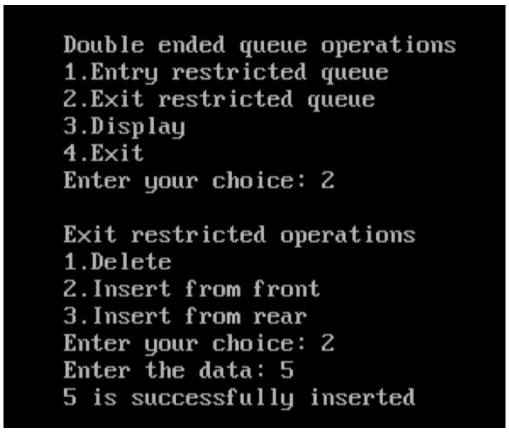
OUTPUT:



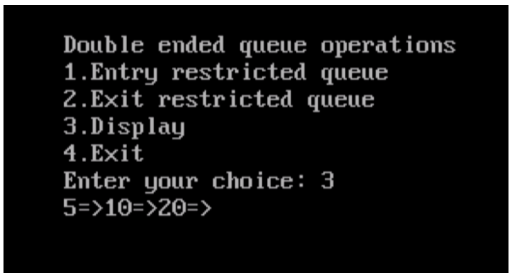
```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 1

Entry restricted operations
1.Insert
2.Delete from front
3.Delete from rear
Enter your choice: 1
Enter the data: 10
10 is successfully inserted
```



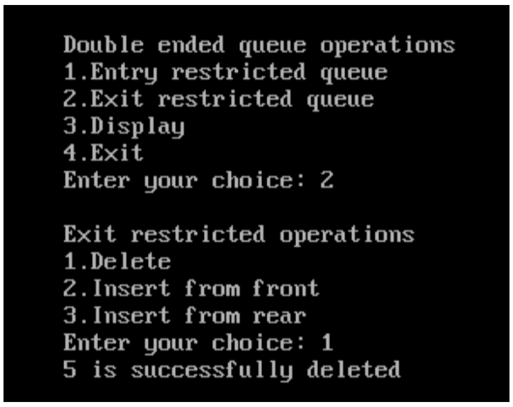
```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 2

Exit restricted operations
1.Delete
2.Insert from front
3.Insert from rear
Enter your choice: 3
Enter the data: 20
20 is successfully inserted
```



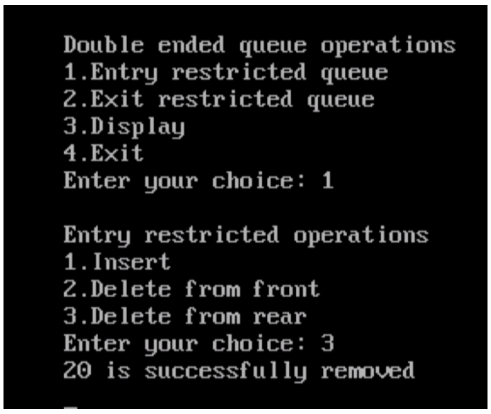
```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 2

Exit restricted operations
1.Delete
2.Insert from front
3.Insert from rear
Enter your choice: 2
Enter the data: 5
5 is successfully inserted
```



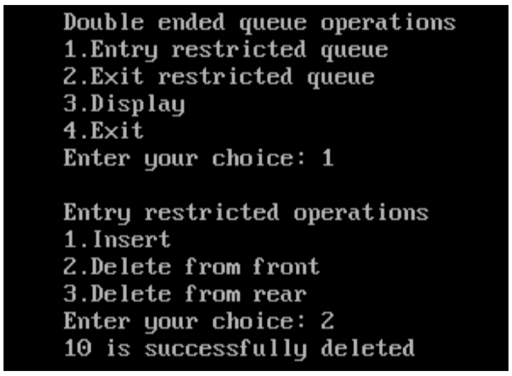
```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 3
5=>10=>20=>
```



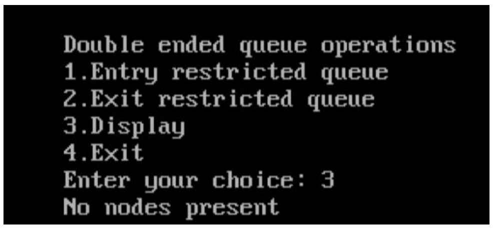
```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 2

Exit restricted operations
1.Delete
2.Insert from front
3.Insert from rear
Enter your choice: 1
5 is successfully deleted
```



```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 1

Entry restricted operations
1.Insert
2.Delete from front
3.Delete from rear
Enter your choice: 3
20 is successfully removed
```



```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 1

Entry restricted operations
1.Insert
2.Delete from front
3.Delete from rear
Enter your choice: 2
10 is successfully deleted
```



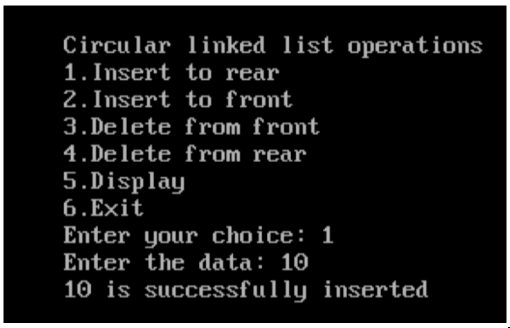
```
Double ended queue operations
1.Entry restricted queue
2.Exit restricted queue
3.Display
4.Exit
Enter your choice: 3
No nodes present
```

5. Write a program to implement circular linked list using header node

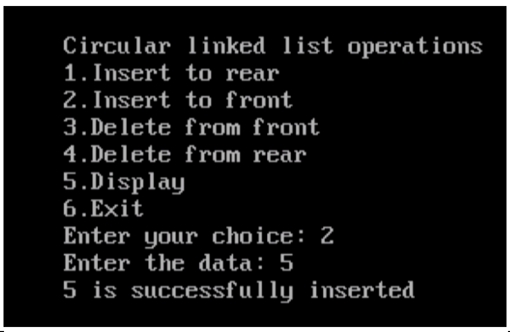
```
#include<stdio.h>
```

```
#include<conio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *link;
};
typedef struct node *NODE;

NODE insert_rear(NODE);
NODE insert_front(NODE);
NODE delete_front(NODE);
NODE delete_rear(NODE);
NODE getnode();
void display(NODE);


void main()
{
 NODE head;
 int option;
 clrscr();
 head=getnode();
 head->link=head;
 while(1)
 {
  printf("\nCircular linked list operations\n");
  printf("1.Insert to rear\n2.Insert to front\n3.Delete from front\n4.Delete from rear\n5.Display\n6.Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&option);
  switch(option)
  {
   case 1: head=insert_rear(head);
		getch();
		clrscr();
		break;
   case 2: head=insert_front(head);
		getch();
		clrscr();
		break;
   case 3: head=delete_front(head);
		getch();
		clrscr();
		break;
   case 4: head=delete_rear(head);
		getch();
		clrscr();
		break;
   case 5: display(head);
		getch();
		clrscr();
		break;
   case 6: exit(0);
   default: printf("Invalid choice\n");
  }
 }
}

NODE getnode()
{
 NODE temp;
 temp=(NODE)malloc(sizeof(struct node));
 return temp;
}

NODE insert_front(NODE head)
{
 NODE temp,first;
 temp=getnode();
 printf("Enter the data: ");
 scanf("%d",&temp->data);
 printf("%d is successfully inserted\n",temp->data);
 temp->link=head->link;
 head->link=temp;
 return head;
}

NODE delete_front(NODE head)
{
 NODE ptr;
 if(head->link==head)
 {
  printf("No nodes\n");
  return head;
 }
 ptr=head->link;
 printf("%d is successfully deleted\n",ptr->data);
 head->link=ptr->link;
 free(ptr);
 return head;
}

NODE insert_rear(NODE head)
{
 NODE ptr,temp;
 ptr=head->link;
 while(ptr->link!=head)
 ptr=ptr->link;
 temp=getnode();
 printf("Enter the data: ");
 scanf("%d",&temp->data);
 printf("%d is successfully inserted\n",temp->data);
 ptr->link=temp;
 temp->link=head;
 return head;
}

NODE delete_rear(NODE head)
{
 NODE ptr,prev;
 if(head->link==head)
 {
  printf("No nodes\n");
  return head;
 }
 ptr=head->link;
 while(ptr->link!=head)
 {
  prev=ptr;
  ptr=ptr->link;
 }
 printf("%d is successfully deleted\n",ptr->data);
 prev->link=head;
 free(ptr);
 return head;
}

void display(NODE head)
{
 NODE temp;
 if(head->link==head)
  printf("No nodes\n");
 else
 {
  for(temp=head->link;temp->link!=head;temp=temp->link)
  printf("%d=>",temp->data);
  printf("%d",temp->data);
 }
}
```

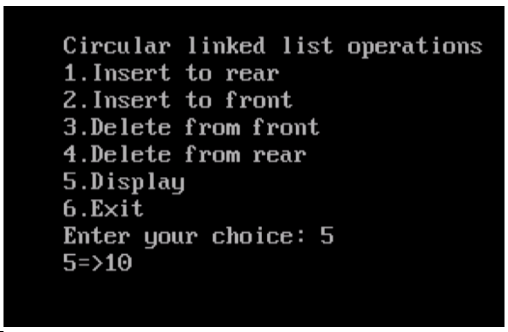
OUTPUT:



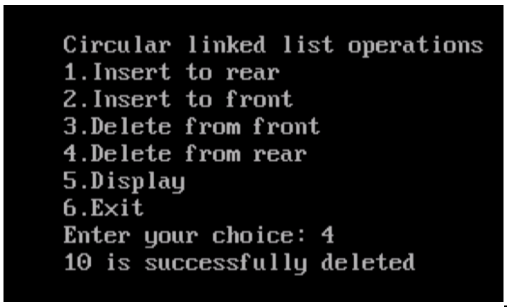
```
Circular linked list operations
1.Insert to rear
2.Insert to front
3.Delete from front
4.Delete from rear
5.Display
6.Exit
Enter your choice: 1
Enter the data: 10
10 is successfully inserted
```



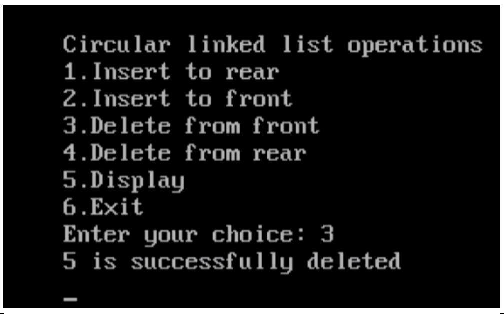
```
Circular linked list operations
1.Insert to rear
2.Insert to front
3.Delete from front
4.Delete from rear
5.Display
6.Exit
Enter your choice: 2
Enter the data: 5
5 is successfully inserted
```



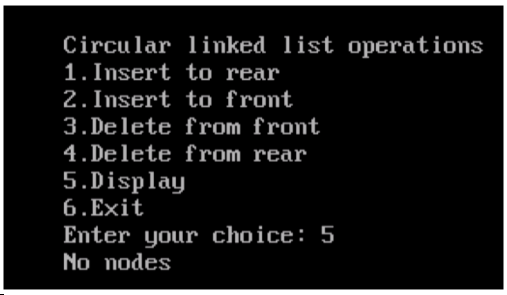
```
Circular linked list operations
1.Insert to rear
2.Insert to front
3.Delete from front
4.Delete from rear
5.Display
6.Exit
Enter your choice: 5
5=>10
```



```
Circular linked list operations
1.Insert to rear
2.Insert to front
3.Delete from front
4.Delete from rear
5.Display
6.Exit
Enter your choice: 4
10 is successfully deleted
```



```
Circular linked list operations
1.Insert to rear
2.Insert to front
3.Delete from front
4.Delete from rear
5.Display
6.Exit
Enter your choice: 3
5 is successfully deleted
_
```



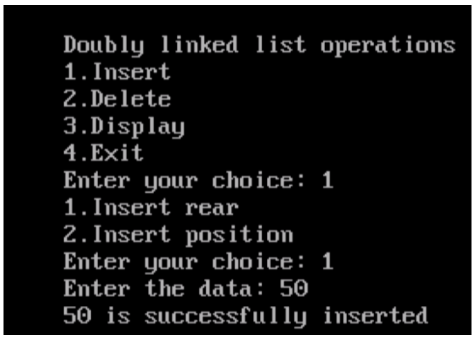
```
Circular linked list operations
1.Insert to rear
2.Insert to front
3.Delete from front
4.Delete from rear
5.Display
6.Exit
Enter your choice: 5
No nodes
```

6.Write a program to implement doubly linked list:

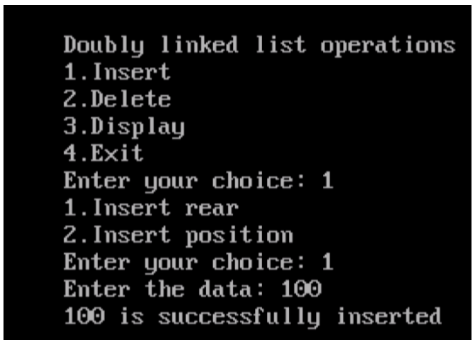
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *llink;
 struct node *rlink;
};
typedef struct node *NODE;

NODE insert_pos(NODE);
NODE delete_pos(NODE);
NODE delete_front(NODE);
NODE insert_rear(NODE);
void display(NODE);
```

```
NODE getnode();

void main()
{
 NODE first=NULL;
 int option;
 clrscr();
 while(1)
 {
  printf("\nDoubly linked list operations\n");
  printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&option);
  switch(option)
  {
   case 1: printf("1.Insert rear\n2.Insert position\nEnter your choice: ");
           scanf("%d",&option);
           switch(option)
           {
            case 1: first=insert_rear(first);
            getch();
            clrscr();
            break;
            case 2: first=insert_pos(first);
            getch();
            clrscr();
            break;
            default:printf("Invalid choice\n");
           }
           break;
   case 2: first=delete_pos(first);
           getch();
           clrscr();
           break;
   case 3: display(first);
           getch();
           clrscr();
           break;
   case 4: exit(0);
   default: printf("Invalid choice\n");
  }
 }
}

NODE getnode()
{
 NODE temp;
 temp=(NODE)malloc(sizeof(struct node));
 return temp;
}

NODE insert_pos(NODE first)
{
 int pos,count;
 NODE temp,ptr;
 temp=getnode();
 temp->llink=temp->rlink=NULL;
 if(first==NULL)
 {
  printf("Enter the data: ");
  scanf("%d",&temp->data);
  printf("%d is successfully inserted\n",temp->data);
  first=temp;
  return first;
 }
 count=1;
 printf("Enter the position: ");
 scanf("%d",&pos);
 ptr=first;
 while(ptr->rlink!=NULL)
 {
  ptr=ptr->rlink;
  count++;
 }
 if(pos>count)
 {
  printf("Invalid position\n");
  return first;
 }
 printf("Enter the data: ");
 scanf("%d",&temp->data);
 printf("%d is successfully inserted\n",temp->data);
 if(pos==1)
 {
  temp->rlink=first;
  first->llink=temp;
  first=temp;
  return first;
 }
 ptr=first;
 count=1;
 while(count<pos)
 {
  ptr=ptr->rlink;
  count++;
 }
 ptr->llink->rlink=temp;
 temp->llink=ptr->llink;
 temp->rlink=ptr;
 ptr->llink=temp;
 return first;
}

NODE delete_pos(NODE first)
{
 NODE ptr;
 int pos,count;
 if(first==NULL)
 {
  printf("No nodes\n");
  return first;
 }
 ptr=first;
 count=1;
 while(ptr->rlink!=NULL)
 {
  ptr=ptr->rlink;
  count++;
 }
 printf("Enter the postion: ");
 scanf("%d",&pos);
 if(pos>count)
 {
  printf("Invalid postion\n");
  return first;
 }
 if(pos==1)
 {
  first=delete_front(first);
  return first;
 }
 ptr=first;
 count=1;
 while(count<pos)
 {
  ptr=ptr->rlink;
  count++;
 }
 printf("%d is successfully deleted\n",ptr->data);
 ptr->llink->rlink=ptr->rlink;
 ptr->rlink->llink=ptr->llink;
 free(ptr);
 return first;
}

void display(NODE first)
{
 NODE temp;
 if(first==NULL)
  printf("No nodes\n");
 else
 {
  for(temp=first;temp->rlink!=NULL;temp=temp->rlink)
  printf("%d=>",temp->data);
  printf("%d",temp->data);
 }
}

NODE insert_rear(NODE first)
{
 NODE temp,ptr;
 temp=getnode();
 printf("Enter the data: ");
 scanf("%d",&temp->data);
 printf("%d is successfully inserted\n",temp->data);
 temp->rlink=temp->llink=NULL;
 if(first==NULL)
 {
  first=temp;
  return first;
 }
 ptr=first;
 while(ptr->rlink!=NULL)
 ptr=ptr->rlink;
 ptr->rlink=temp;
 temp->llink=ptr;
 return first;
}

NODE delete_front(NODE first)
{
 NODE ptr;
 ptr=first;
 first->rlink->llink=NULL;
 first=first->rlink;
 printf("%d is successfully deleted",ptr->data);
 return first;
}
```

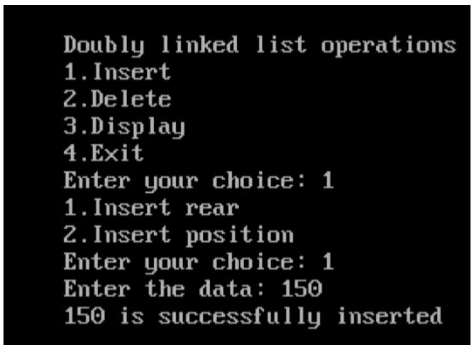
OUTPUT:



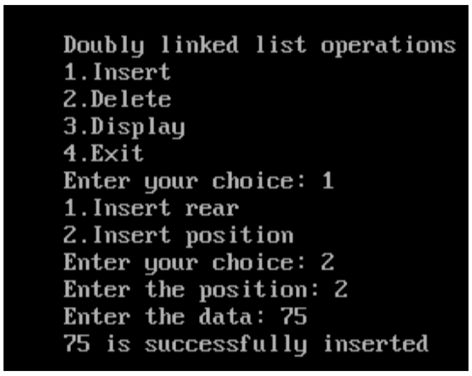
```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
1.Insert rear
2.Insert position
Enter your choice: 1
Enter the data: 50
50 is successfully inserted
```



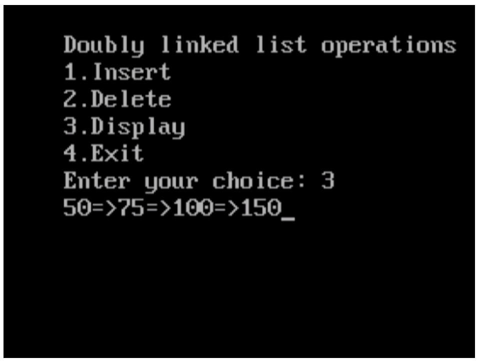
```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
1.Insert rear
2.Insert position
Enter your choice: 1
Enter the data: 100
100 is successfully inserted
```



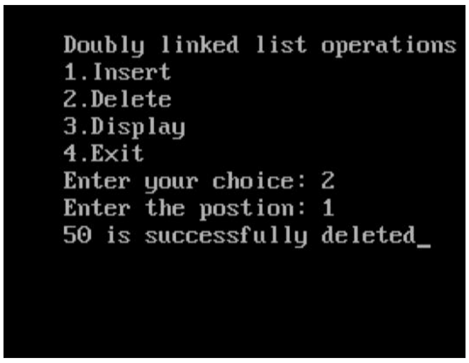
```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
1.Insert rear
2.Insert position
Enter your choice: 1
Enter the data: 150
150 is successfully inserted
```



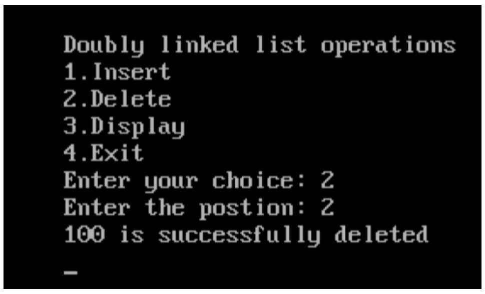
```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
1.Insert rear
2.Insert position
Enter your choice: 2
Enter the position: 2
Enter the data: 75
75 is successfully inserted
```



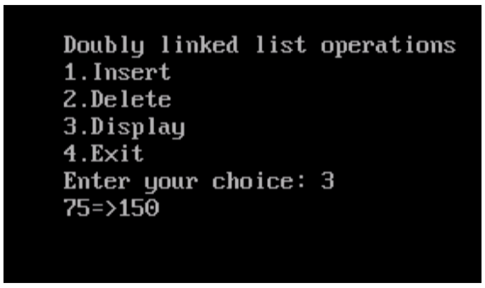
```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 3
50=>75=>100=>150_
```



```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 2
Enter the postion: 1
50 is successfully deleted_
```



```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 2
Enter the postion: 2
100 is successfully deleted
_
```



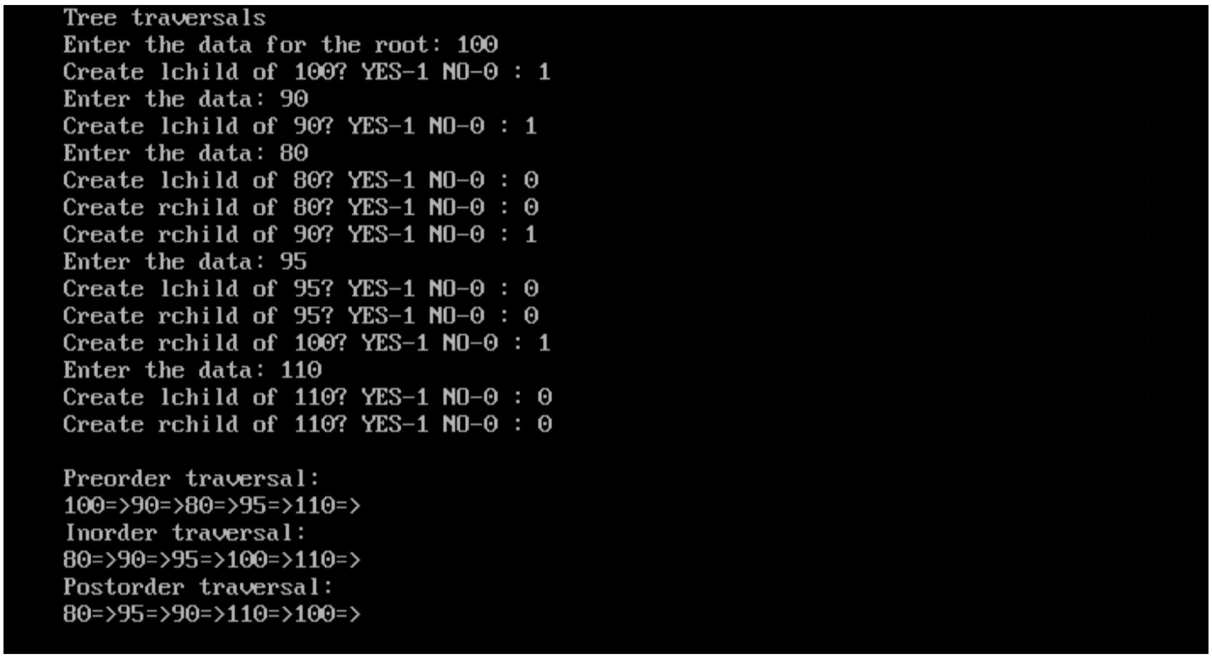
```
Doubly linked list operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 3
75=>150
```

7. Create binary tree and traverse inrorder, preorder and postorder:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *lchild;
 struct node *rchild;
};

typedef struct node *NODE;
NODE root=NULL;

void create(NODE);
int is_lchild(NODE);
int is_rchild(NODE);
void inorder(NODE);
void preorder(NODE);
void postorder(NODE);
NODE getnode();

void main()
{
 clrscr();
 printf("Tree traversals\n");
 printf("Enter the data for the root: ");
 root=getnode();
 scanf("%d",&root->data);
 create(root);
 printf("\nPreorder traversal:\n");
 preorder(root);
 printf("\nInorder traversal:\n");
 inorder(root);
 printf("\nPostorder traversal:\n");
 postorder(root);
 getch();
}

NODE getnode()
{
 NODE temp;
 temp=(NODE)malloc(sizeof(struct node));
 temp->lchild=temp->rchild=NULL;
 return temp;
}

void create(NODE root)
{
 if(is_lchild(root))
 {
  root->lchild=getnode();
  printf("Enter the data: ");
  scanf("%d",&root->lchild->data);
  create(root->lchild);
 }
 if(is_rchild(root))
 {
  root->rchild=getnode();
  printf("Enter the data: ");
  scanf("%d",&root->rchild->data);
  create(root->rchild);
 }
}

int is_lchild(NODE root)
{
 int ch;
 printf("Create lchild of %d? YES-1 NO-0 : ",root->data);
 scanf("%d",&ch);
 if(ch)
 return 1;
 else
 return 0;
}

int is_rchild(NODE root)
```

```
{
 int ch;
 printf("Create rchild of %d? YES-1 NO-0 : ",root->data);
 scanf("%d",&ch);
 if(ch)
 return 1;
 else
 return 0;
}

void inorder(NODE root)
{
 if(root!=NULL)
 {
  inorder(root->lchild);
  printf("%d=>",root->data);
  inorder(root->rchild);
 }
}

void preorder(NODE root)
{
 if(root!=NULL)
 {
  printf("%d=>",root->data);
  preorder(root->lchild);
  preorder(root->rchild);
 }
}

void postorder(NODE root)
{
 if(root!=NULL)
 {
  postorder(root->lchild);
  postorder(root->rchild);
  printf("%d=>",root->data);
 }
}
```

OUTPUT:



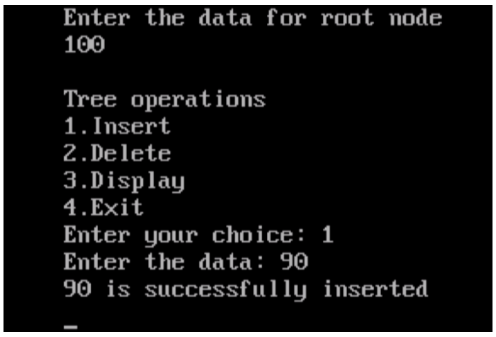
```
Tree traversals
Enter the data for the root: 100
Create lchild of 100? YES-1 NO-0 : 1
Enter the data: 90
Create lchild of 90? YES-1 NO-0 : 1
Enter the data: 80
Create lchild of 80? YES-1 NO-0 : 0
Create rchild of 80? YES-1 NO-0 : 0
Create rchild of 90? YES-1 NO-0 : 1
Enter the data: 95
Create lchild of 95? YES-1 NO-0 : 0
Create rchild of 95? YES-1 NO-0 : 0
Create rchild of 100? YES-1 NO-0 : 1
Enter the data: 110
Create lchild of 110? YES-1 NO-0 : 0
Create rchild of 110? YES-1 NO-0 : 0

Preorder traversal:
100=>90=>80=>95=>110=>
Inorder traversal:
80=>90=>95=>100=>110=>
Postorder traversal:
80=>95=>90=>110=>100=>
```

8. Perform insert and delete operations in binary search tree:

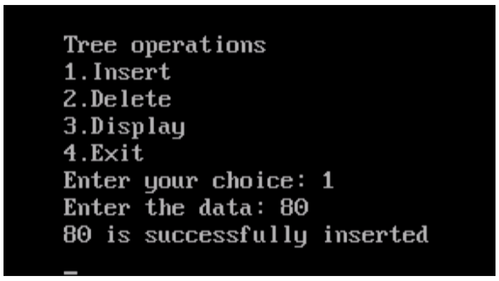
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
 int data;
 struct node *llink;
 struct node *rlink;
};
typedef struct node *NODE;

NODE insertion(NODE);
NODE deletion(NODE);
void inorder(NODE);
NODE getnode();

void main()
{
 int option;
 NODE root;
 clrscr();
 root=getnode();
 printf("Enter the data for root node\n");
 scanf("%d",&root->data);
 while(1)
 {
  printf("\nTree operations\n");
  printf("1.Insert\n2.Delete\n3.Display\n4.Exit\n");
  printf("Enter your choice: ");
  scanf("%d",&option);
  switch(option)
  {
   case 1: root=insertion(root);
           getch();
           clrscr();
           break;
   case 2: root=deletion(root);
```

```
              getch();
              clrscr();
              break;
   case 3: printf("\nInorder traversal\n");
              inorder(root);
              getch();
              clrscr();
              break;
   case 4: exit(0);
   default: printf("Invalid choice\n");
  }
 }
}

NODE insertion(NODE root)
{
 NODE prev,temp,temp1;
 int item;
 temp1=getnode();
 printf("Enter the data: ");
 scanf("%d",&temp1->data);
 printf("%d is successfully inserted\n",temp1->data);
 item=temp1->data;
 prev=NULL;
 temp=root;
 while(temp!=NULL)
 {
  prev=temp;
  if(item<temp->data)
   temp=temp->llink;
  else
   temp=temp->rlink;
 }
 if(item<prev->data)
  prev->llink=temp1;
 else
  prev->rlink=temp1;
 return root;
}

NODE getnode()
{
 NODE temp;
 temp=(NODE)malloc(sizeof(struct node));
 temp->llink=temp->rlink=NULL;
 return temp;
}

NODE deletion(NODE root)
{
 NODE cur,parent,suc,q;
 int item;
 if(root==NULL)
 {
  printf("No elements in tree\n");
  return root;
 }
 printf("Enter the element to be deleted: ");
 scanf("%d",&item);
 parent=NULL;
 cur=root;
 while(cur!=NULL && item!=cur->data)
 {
  parent=cur;
  if(item<cur->data)
   cur=cur->llink;
  else
   cur=cur->rlink;
 }
 if(cur==NULL)
 {
  printf("%d not found",item);
  return root;
 }
 if(cur->llink==NULL)
  q=cur->rlink;
 else
 if(cur->rlink==NULL)
  q=cur->llink;
 else
 {
  suc=cur->rlink;
  while(suc->llink!=NULL)
   suc=suc->llink;
  suc->llink=cur->llink;
  q=cur->rlink;
 }
 if(cur==parent->llink)
  parent->llink=q;
 else
  parent->rlink=q;
 free(cur);
 printf("%d is successfully deleted\n",item);
 return root;
}

void inorder(NODE root)
{
 if(root!=0)
 {
  inorder(root->llink);
  printf("%d=>",root->data);
  inorder(root->rlink);
 }
}
```

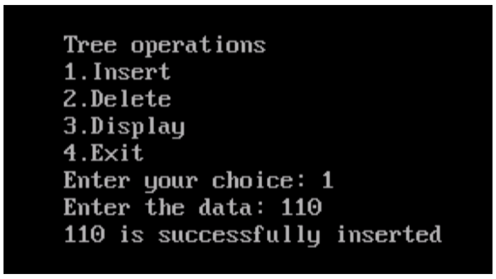
OUTPUT:



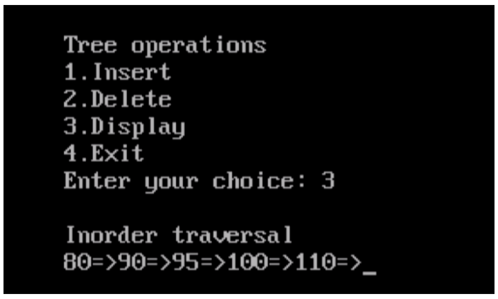
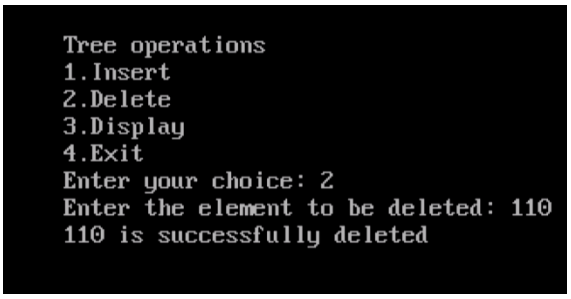
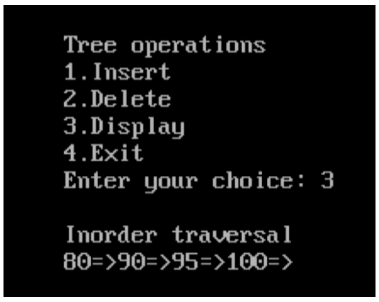
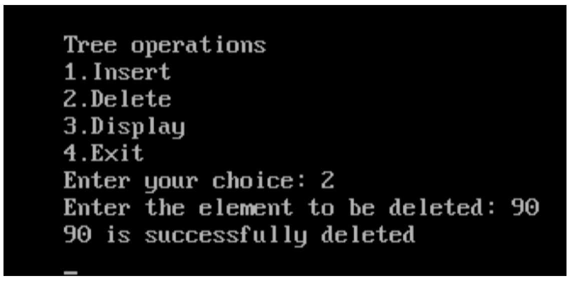
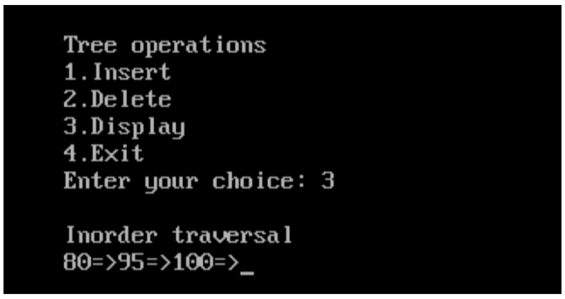
```
Enter the data for root node
100

Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
Enter the data: 90
90 is successfully inserted
_
```



```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
Enter the data: 80
80 is successfully inserted
_
```



```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
Enter the data: 110
110 is successfully inserted
```

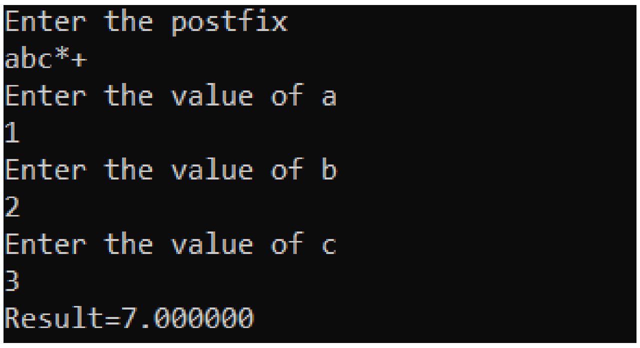

```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 3

Inorder traversal
80=>90=>95=>100=>110=>_
```

```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 2
Enter the element to be deleted: 110
110 is successfully deleted
```

```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 3

Inorder traversal
80=>90=>95=>100=>
```

```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 2
Enter the element to be deleted: 90
90 is successfully deleted
_
```

```
Tree operations
1.Insert
2.Delete
3.Display
4.Exit
Enter your choice: 3

Inorder traversal
80=>95=>100=>_
```

9.Evaluate the given postfix expression using trees:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
struct node
{
 int data;
 struct node *lchild;
 struct node *rchild;
};

typedef struct node *NODE;
NODE root=0;
NODE create_tree(char postfix[]);
float eval(NODE root);
void main()
{
 char postfix[20];
 float result;
 clrscr();
 printf("Enter the postfix\n");
 scanf("%s",postfix);
 root=create_tree(postfix);
 result=eval(root);
 printf("Result=%f\n",result);
 getch();
}

NODE create_tree(char postfix[])
{
 NODE temp,stack[20];
 int i=0,j=0;
 char symbol;
 for(i=0;(symbol=postfix[i])!=0;i++)
 {
  temp=(NODE)malloc(sizeof(struct node));
  temp->lchild=temp->rchild=0;
  temp->data=symbol;
  if(isalnum(symbol))
```

```
   stack[j++]=temp;
  else
  {
   temp->rchild=stack[--j];
   temp->lchild=stack[--j];
   stack[j++]=temp;
  }
 }
 return (stack[--j]);
}

float eval(NODE root)
{
 float num;
 switch(root->data)
 {
  case '+':return eval(root->lchild)+eval(root->rchild);
  case '-':return eval(root->lchild)-eval(root->rchild);
  case '/':return eval(root->lchild)/eval(root->rchild);
  case '*':return eval(root->lchild)*eval(root->rchild);
  case '^':return pow(eval(root->lchild),eval(root->rchild));
  default :if(isalpha(root->data))
              {
               printf("Enter the value of %c\n",root->data);
               scanf("%f",&num);
               return (num);
              }
              else
               return (root->data-'0');
 }
}
```

OUTPUT:



```
Enter the postfix
abc*+
Enter the value of a
1
Enter the value of b
2
Enter the value of c
3
Result=7.000000
```

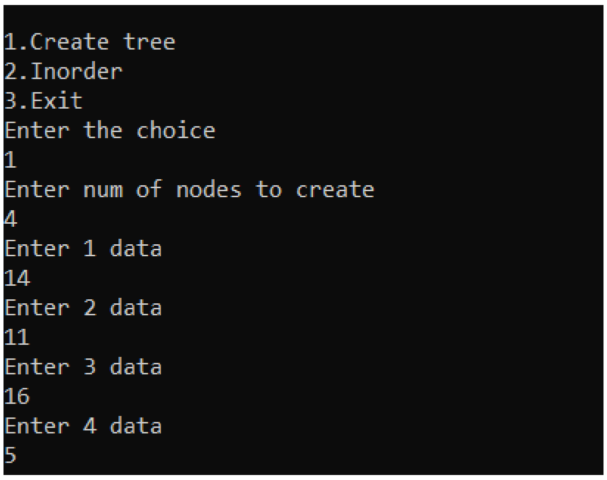
10. Create Right in threaded binary tree:

```
#include<conio.h>
#include<stdio.h>
struct node
{
 int data;
 struct node*left;
 struct node*right;
 int RT;
};
typedef struct node *NODE;
```

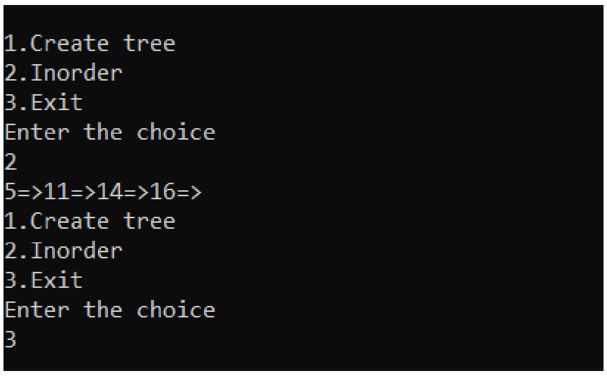
```
NODE head=0;
NODE create(int,NODE);
void insert_left(int,NODE);
void insert_right(int,NODE);
void inorder(NODE);
NODE inorder_successor(NODE);
int ch,i,n,item,choice;

void main()
{
 NODE head;
 clrscr();
 head=(NODE)malloc(sizeof(struct node));
 head->right=head;
 head->left=0;
 head->RT=0;
 while(1)
 {
  printf("\n1.Create tree\n2.Inorder\n3.Exit\n");
  printf("Enter the choice\n");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:printf("Enter num of nodes to create\n");
	  scanf("%d",&n);
	  for(i=1;i<n+1;i++)
	  {
	   printf("Enter %d data\n",i);
	   scanf("%d",&item);
	   head=create(item,head);
	  }
	  break;
   case 2:inorder(head);
	  break;
   case 3:exit(0);
   default: printf("Wrong choice\n");
	    break;
   }
  }
 }

 NODE create(int item,NODE head)
 {
  NODE curptr,ptr;
  if(head->left==0)
  {
   insert_left(item,head);
   return (head);
  }
  curptr=head->left;
  while(curptr!=head)
  {
   ptr=curptr;
   if(item<(curptr->data))
   {
    if(curptr->left!=0)
```

```
      curptr=curptr->left;
    else
     break;
   }
   else
   {
    if(item>(curptr->data))
    {
     if(curptr->RT==0)
      curptr=curptr->right;
     else
      break;
    }
   }
  }
  if(item<(curptr->data))
  {
   insert_left(item,ptr);
   return (head);
  }
  else
  {
   if(item>(curptr->data) &&  curptr->RT==1)
    insert_right(item,ptr);
  }
  return (head);
}

void insert_left(int item,NODE ptr)
{
 NODE temp,newnode;
 newnode=(NODE)malloc(sizeof(struct node));
 newnode->left=0;
 newnode->data=item;
 ptr->left=newnode;
 newnode->right=ptr;
 newnode->RT=1;
}

void insert_right(int item,NODE ptr)
{
 NODE temp,newnode;
 newnode=(NODE)malloc(sizeof(struct node));
 newnode->left=0;
 newnode->data=item;
 temp=ptr->right;
 ptr->right=newnode;
 newnode->right=temp;
 ptr->RT=0;
 newnode->RT=1;
}

void inorder(NODE head)
{
 NODE temp;
 if(head->left==0)
 {
  printf("\n No nodes");
  return;
 }
 temp=head;
 while(1)
 {
  temp=inorder_successor(temp);
  if(temp==head)
  return;
  printf("%d=>",temp->data);
 }
}

NODE inorder_successor(NODE ptr)
{
  NODE temp;
  temp=ptr->right;
  if(ptr->RT==1)
    return (temp);
  while(temp->left!=0)
   temp=temp->left;
  return (temp);
}
```

OUTPUT:



```
1.Create tree
2.Inorder
3.Exit
Enter the choice
1
Enter num of nodes to create
4
Enter 1 data
14
Enter 2 data
11
Enter 3 data
16
Enter 4 data
5
```

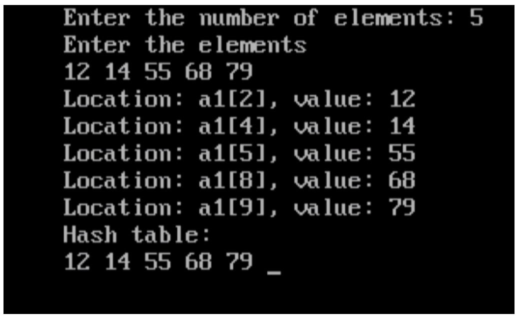


```
1.Create tree
2.Inorder
3.Exit
Enter the choice
2
5=>11=>14=>16=>
1.Create tree
2.Inorder
3.Exit
Enter the choice
3
```

11. Implement Hash Tables:

```
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
void main()
{
 int a[100],a1[100],a2[100],i,result,n;
 clrscr();
 printf("Enter the number of elements: ");
 scanf("%d",&n);
 printf("Enter the elements\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 for(i=0;i<n;i++)
 {
  result=a[i]%10;
  a1[result]=a2[i]=a[i];
  printf("Location: a1[%d], value: %d\n",result,a1[result]);
 }
 printf("Hash table:\n");
 for(i=0;i<n;i++)
 printf("%d ",a2[i]);
 getch();
}
```

OUTPUT:



```
Enter the number of elements: 5
Enter the elements
12 14 55 68 79
Location: a1[2], value: 12
Location: a1[4], value: 14
Location: a1[5], value: 55
Location: a1[8], value: 68
Location: a1[9], value: 79
Hash table:
12 14 55 68 79 _
```

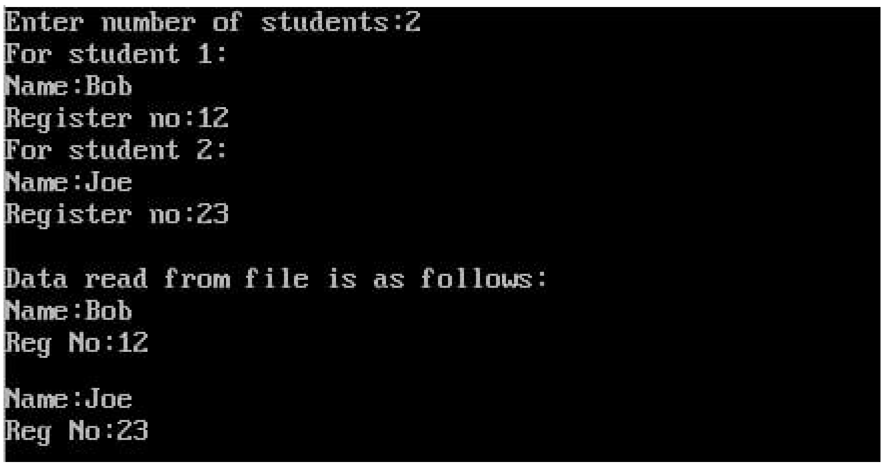
12. Implement Hashing using open addressing:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
 int a[100],a1[100],i,j,result,n,cnt=0;
 clrscr();
 printf("Enter the number of elements: ");
 scanf("%d",&n);
 printf("Enter the elements\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 for(i=0;i<10;i++)
 a1[i]=-1;
 for(i=0;i<n;i++)
 {
  result=a[i]%10;
  if(a1[result]==-1)
  {
   a1[result]=a[i];
   printf("Location: a1[%d], Value: %d\n",result,a1[result]);
  }
  else
  {
   j=result+1;
   while(1)
   {
    if(a1[j]==-1)
    {
     a1[j]=a[i];
     printf("Location: a1[%d], Value: %d\n",j,a1[j]);
     break;
    }
    if(j>n-1)
    j=0;
    if(cnt==n)
    break;
    cnt++;
    j++;
   }
  }
 result=0;
 }
 printf("Hash table using open addressing mode:\n");
 for(i=0;i<10;i++)
 {
  if(a1[i]!=-1)
  printf("%d ",a1[i]);
 }
 getch();
}
```

13. Write all the members of an array of structures to a file using fwrite(). Read the array from file and display on the screen.

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

struct student
 {
  int regno;
  char name[20];
 };

void main()
{
struct student stud[20],temp;
FILE *fout,*fin;
int i,n;
clrscr();
fout=fopen("student.txt","w");
if(fout==NULL)
  {
   printf("Error, could not open file....");
   getch();
   exit(0);
  }
printf("Enter number of students:");
scanf("%d",&n);
for(i=0;i<n;i++)
  {
   clrscr();
   printf("For student %d:\n",i+1);
   printf("Name:");
   scanf("%s",stud[i].name);
   printf("Register no:");
   scanf("%d",&stud[i].regno);
  }
fwrite(stud,sizeof(struct student),n,fout);
fclose(fout);
fin=fopen("student.dat","r");
if(fin==NULL)
  {
   printf("Error, could not read from file....");
   getch();
   exit(0);
  }
clrscr();
printf("\nData read from file is as follows:\n");
while(fread(&temp,sizeof(struct student),1,fin))
  {
   printf("Name:%s\n",temp.name);
   printf("Reg No:%d\n\n",temp.regno);
   getch();
  }
fclose(fin);
}
```

OUTPUT:



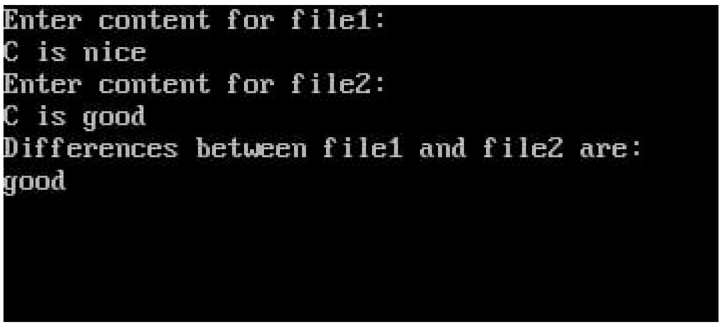
```
Enter number of students:2
For student 1:
Name:Bob
Register no:12
For student 2:
Name:Joe
Register no:23

Data read from file is as follows:
Name:Bob
Reg No:12

Name:Joe
Reg No:23
```

14. Compare the contents of two files. Write the difference in another file.

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

void main()
{
 FILE *fout1,*fout2,*fout3,*fin1,*fin2,*fin3;
 char data1[30],data2[30],ch1,ch2,ch3;
 clrscr();
 fout1 = fopen("file1.txt","w");
 fout2 = fopen("file2.txt","w");
 if(fout1==NULL)
   {
    printf("Could not open file1");
    getch();
    exit(0);
   }
 if(fout2==NULL)
   {
    printf("Could not open file2");
    getch();
    exit(0);
   }
 printf("Enter content for file1:\n");
```

```
gets(data1);
printf("Enter content for file2:\n");
gets(data2);
fprintf(fout1,"%s",data1);
fprintf(fout2,"%s",data2);
fclose(fout1);
fclose(fout2);
fin1=fopen("file1.txt","r");
fin2=fopen("file2.txt","r");
fout3=fopen("file3.txt","w");
if(fin1==NULL)
  {
   printf("Could not read from file 1");
   getch();
   exit(0);
  }
if(fin2==NULL)
  {
   printf("Could not read from file 2");
   getch();
   exit(0);
  }
if(fout3==NULL)
  {
   printf("Could not open file 3");
   getch();
   exit(0);
  }
ch1=getc(fin1);
ch2=getc(fin2);
while(ch1!=EOF && ch2!=EOF)
   {
    if(ch1!=ch2)
      {
         fputc(ch2,fout3);
      }
    ch1=getc(fin1);
    ch2=getc(fin2);
   }
fclose(fin1);
fclose(fin2);
fclose(fout3);
fin3=fopen("file3.dat","r");
if(fin3==NULL)
  {
   printf("Could not read from file 3");
   getch();
   exit(0);
  }
if((ch3=getc(fin3))==NULL)
   {
    printf("No difference between file1 and file2");
    getch();
    exit(0);
   }
 printf("Differences between file1 and file2 are:\n");
 while(ch3!=EOF)
   {
    printf("%c",ch3);
    ch3=getc(fin3);
   }
 fclose(fin3);
 getch();
}
```

OUTPUT:



```
Enter content for file1:
C is nice
Enter content for file2:
C is good
Differences between file1 and file2 are:
good
```