**B.M.S. College of Engineering**

***(Autonomous Institution affiliated to VTU, Belagavi)***

**Department of Computer Science and Engineering**



**NAME:AKSHATHA SHIVAKUMAR**

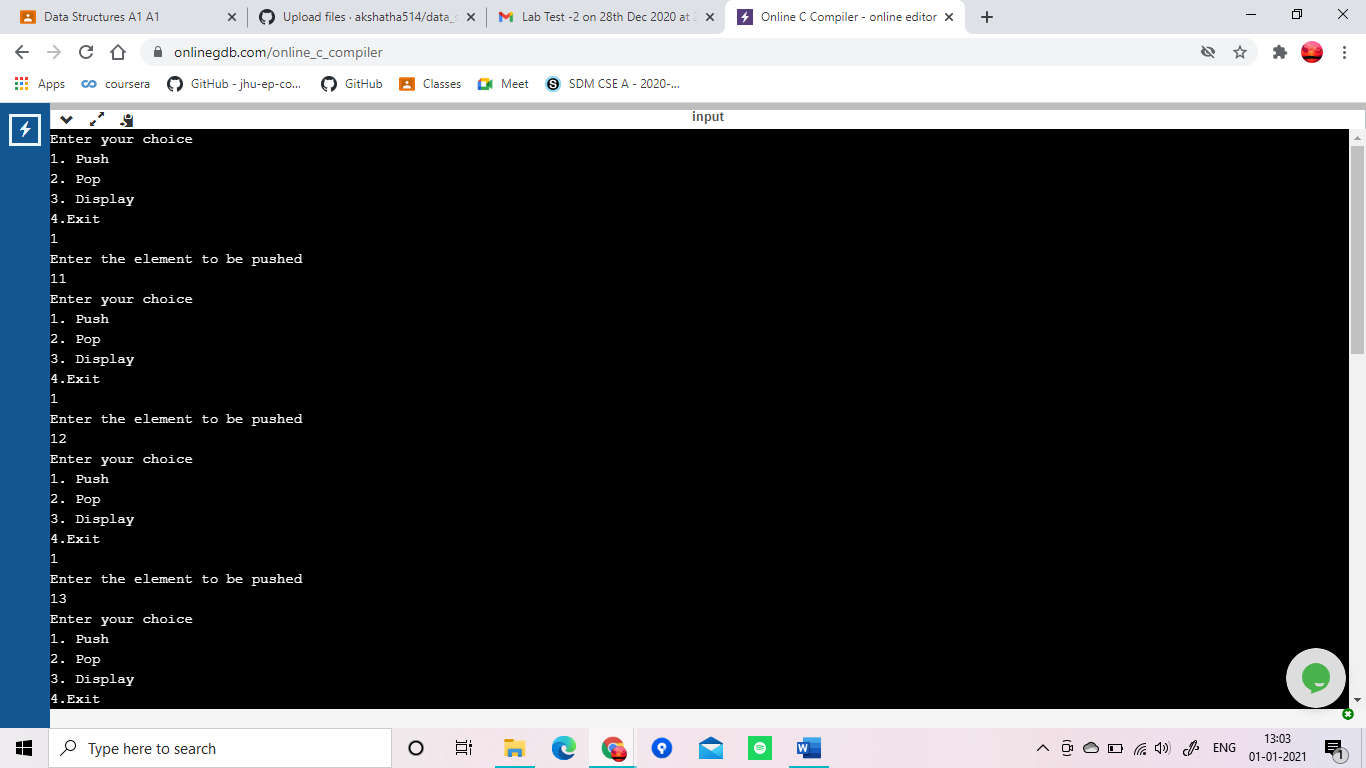
**USN:1BM19CS011**

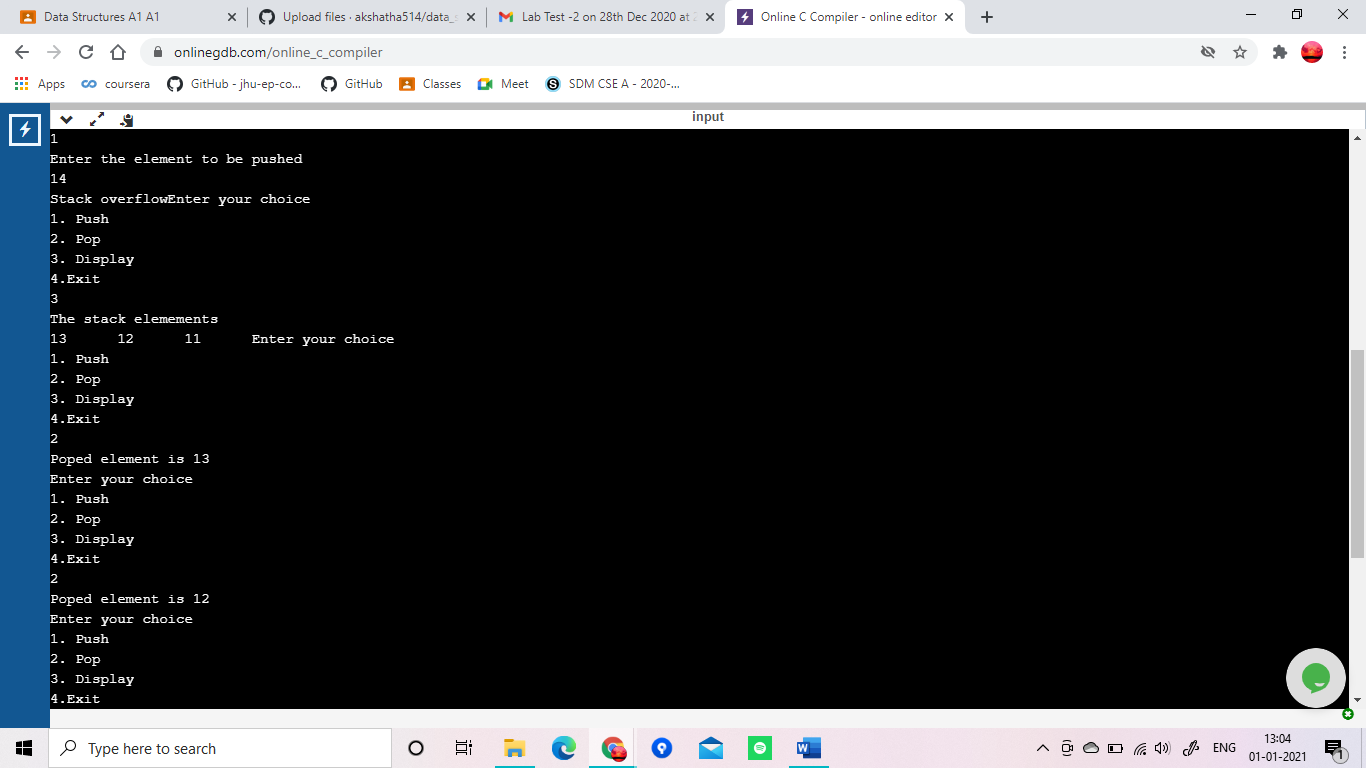
**SUBJECT:DATA STRUCTURES**

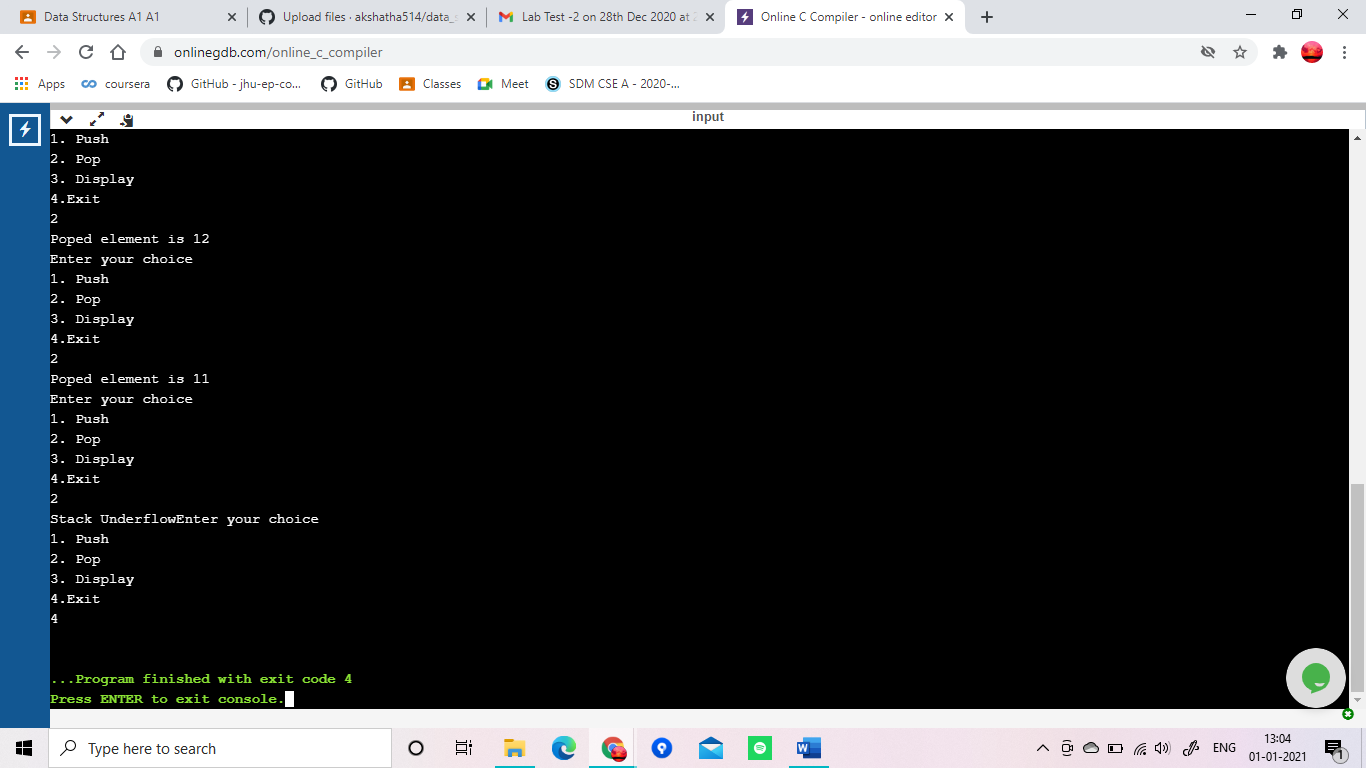
**ACADEMIC YEAR:2020-2021**

Program 1

|  |
| --- |
| #include <stdio.h> |
|  | #define size 3 |
|  | int top=-1; |
|  | void push(int [], int); |
|  | int pop(int[]); |
|  | void display(int []); |
|  | void main() |
|  | { |
|  | int stack[size]; |
|  | int choice,element; |
|  | char ch; |
|  | do |
|  | { |
|  | printf("Enter your choice\n"); |
|  | printf("1. Push\n"); |
|  | printf("2. Pop\n"); |
|  | printf("3. Display\n"); |
|  | printf("4.Exit\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: printf("Enter the element to be pushed \n"); |
|  | scanf("%d",&element); |
|  | push(stack,element); |
|  | break; |
|  | case 2: element=pop(stack); |
|  | if(element==-1) |
|  | printf("Stack Underflow"); |
|  | else |
|  | printf("Poped element is %d \n",element); |
|  | break; |
|  | case 3: display(stack); |
|  | break; |
|  | case 4: |
|  | break; |
|  | default: printf("Invalid choice\n"); |
|  | } |
|  | }while(choice!=4); |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void push(int stack[], int ele) |
|  | { |
|  | if (top==size-1) |
|  | { |
|  | printf("Stack overflow"); |
|  | } |
|  | else |
|  | { |
|  | top++; |
|  | stack[top]=ele; |
|  | } |
|  | } |
|  |  |
|  | int pop(int stack[]) |
|  | { |
|  | int popele; |
|  | if(top==-1) |
|  |  |
|  | return -1; |
|  |  |
|  | else |
|  | { |
|  | popele=stack[top]; |
|  | top--; |
|  | return (popele); |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void display(int stack[]) |
|  | { |
|  | int i; |
|  | printf("The stack elemements\n"); |
|  | for(i=top;i>=0;i--) |
|  | { |
|  |  |
|  | printf("%d\t",stack[i]); |
|  | } |
|  | } |

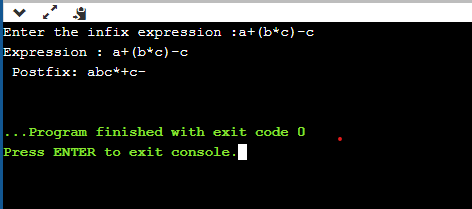


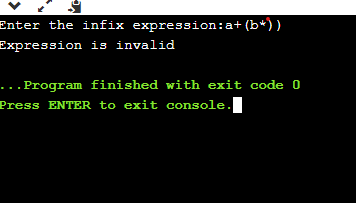


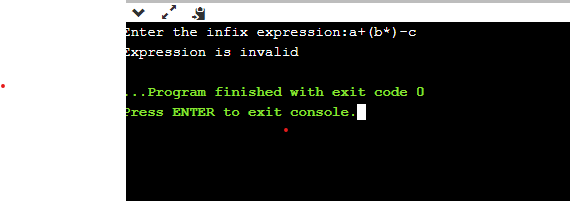
z

Program 2

|  |
| --- |
| #include <stdio.h> |
|  | # define MAX 100 |
|  | char stack[MAX]; |
|  | int top=-1; |
|  |  |
|  | void push(char ch) |
|  | { |
|  | if (top==MAX-1) |
|  | printf("Stack is full\n"); |
|  | else |
|  | { |
|  | top++; |
|  | stack[top]=ch; |
|  | } |
|  | } |
|  | char pop() |
|  | { |
|  | char item; |
|  | if (top==-1) |
|  | printf("\n stack is empty !"); |
|  | else |
|  | { |
|  | item=stack[top]; |
|  | top--; |
|  | return item; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | int stackempty() |
|  | { |
|  | if(top==-1) return 1; |
|  | else return 0; |
|  | } |
|  |  |
|  | char stacktop() |
|  | { |
|  | if( top==-1) |
|  | printf("\n stack is empty!"); |
|  | else |
|  | return stack[top]; |
|  | } |
|  | int priority(char ch) |
|  | { |
|  | switch(ch) |
|  | { |
|  | case '+': |
|  | case '-':return (1); |
|  | case '\*': |
|  | case '/':return (2); |
|  | case '^': return (3); |
|  | default : return (0); |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | int main() |
|  | { |
|  | char infix[100]; |
|  | int i, item; |
|  | printf("Enter the infix expression :"); |
|  | scanf("%s",infix); |
|  | for(int i=0;i<strlen(infix);i++) |
|  | { |
|  | if((infix[i]=='\*'||infix[i]=='+'||infix[i]=='/'||infix[i]=='-'||infix[i]=='^'||infix[i]=='(')&&infix[i+1]=='\*'||infix[i+1]=='+'||infix[i+1]=='/'||infix[i+1]=='-'||infix[i+1]=='^'infix[i]==')')) |
|  | { |
|  | printf("Invalid Expression"); |
|  | exit(0); |
|  | } |
|  | } |
|  | printf("Expression : %s",infix); |
|  | printf("\n Postfix: "); |
|  | i=0; |
|  | while (infix[i]!='\0') |
|  | { |
|  |  |
|  |  |
|  | switch (infix[i]) |
|  | { |
|  | case '(': push(infix[i]); |
|  | break; |
|  | case ')':while(( item=pop())!='(') |
|  | printf("%c",item); |
|  | break; |
|  | case '+': |
|  | case '-': |
|  | case '\*': |
|  | case '/': |
|  | case '^': |
|  | while(!stackempty() && priority(infix[i])<=priority(stacktop())) |
|  | { |
|  | item=pop(); |
|  |  |
|  | printf("%c", item); |
|  | } |
|  |  |
|  | push(infix[i]); |
|  | break; |
|  | default : printf("%c", infix[i]); |
|  | break; |
|  |  |
|  |  |
|  | } |
|  | i++; |
|  | } |
|  |  |
|  | while(!stackempty()) |
|  | { |
|  | char item; |
|  | item=pop(); |
|  | printf("%c", item); |
|  |  |
|  | } |
|  | printf("\n"); |
|  | return 0; |
|  |  |
|  | } |

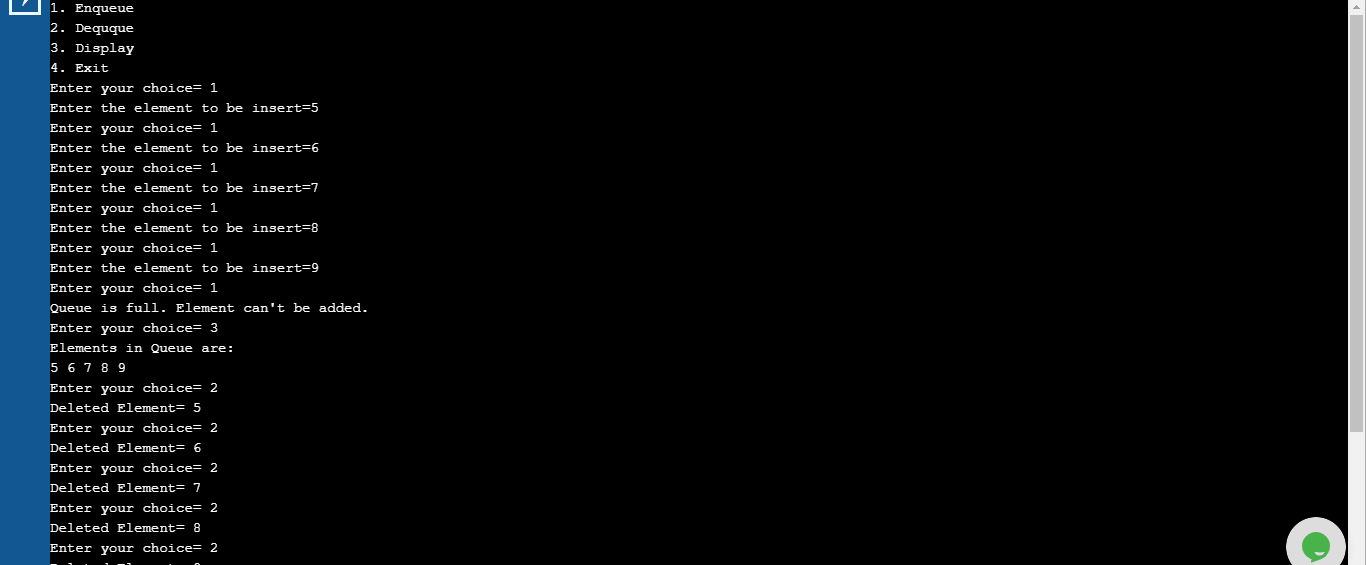


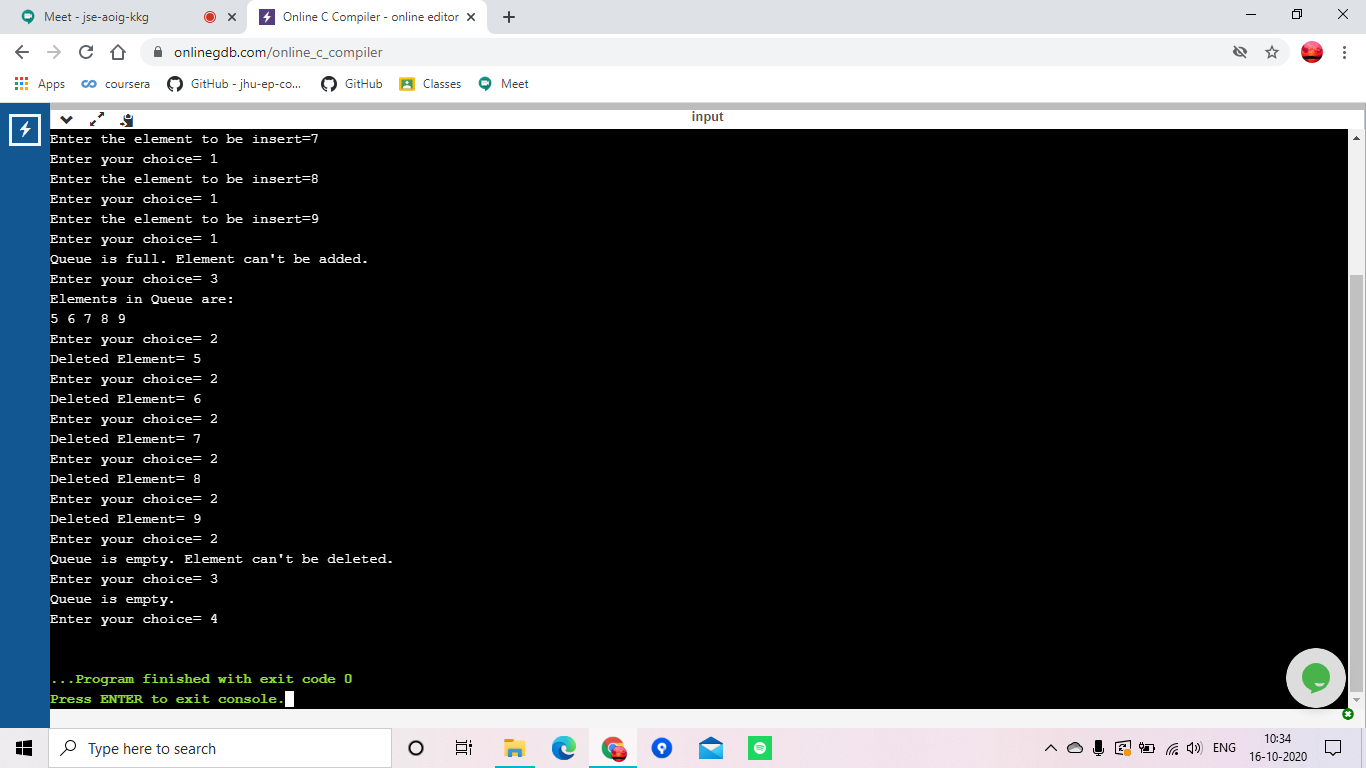




Program 3

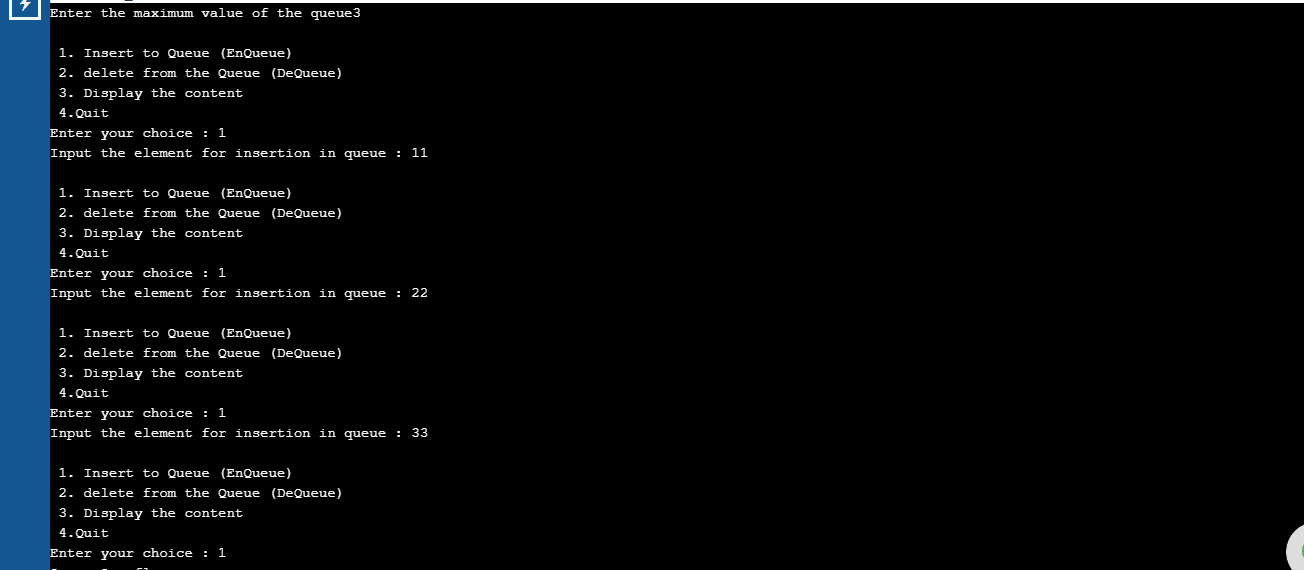
|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #define MAX 5 |
|  |  |
|  | int front=0; |
|  | int rear=-1; |
|  |  |
|  | int queue[MAX]; |
|  |  |
|  | void Enque(int); |
|  | int Deque(); |
|  | void display(); |
|  | int main() |
|  | { |
|  | int option; |
|  | int item; |
|  | do{ |
|  | printf("\n 1. Insert to Queue (EnQueue)"); |
|  | printf("\n 2. delete from the Queue (DeQueue)"); |
|  | printf("\n 3. Display the content "); |
|  | printf("\n 4. Exit\n"); |
|  | printf("Enter the option :"); |
|  | scanf("%d",&option); |
|  | switch(option) |
|  | { |
|  | case 1: printf("Enter the element\n"); |
|  | scanf("%d",&item); |
|  | Enque(item); |
|  | break; |
|  | case 2: item=Deque(); |
|  | if(item==-1) |
|  | printf("Queue is empty\n"); |
|  | else |
|  | printf("Removed element from the queue %d",item); |
|  | break; |
|  | case 3: display(); |
|  | break; |
|  | case 4: exit(0); |
|  | } |
|  | } while (option!=4); |
|  | return 0; |
|  | } |
|  |  |
|  | void Enque(int ele) |
|  | { |
|  | if (rear==MAX-1) |
|  | printf("Queue is full\n"); |
|  | else |
|  | { |
|  | rear++; |
|  | queue[rear]=ele; |
|  |  |
|  | } |
|  | } |
|  | int Deque() |
|  | { |
|  | int item; |
|  | if(front == -1) |
|  | return -1; |
|  | else |
|  | { |
|  | item=queue[front]; |
|  | front++; |
|  | if(front>rear) |
|  | { |
|  | front=-1; |
|  | rear=-1; |
|  | } |
|  | return item; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | int i; |
|  | if(front==-1) |
|  | printf("Queue is empty\n"); |
|  | else |
|  | { |
|  | printf("\n Queue contents:"); |
|  | for(i=front;i<=rear;i++) |
|  | printf("%d", queue[i]); |
|  | } |
|  | } |

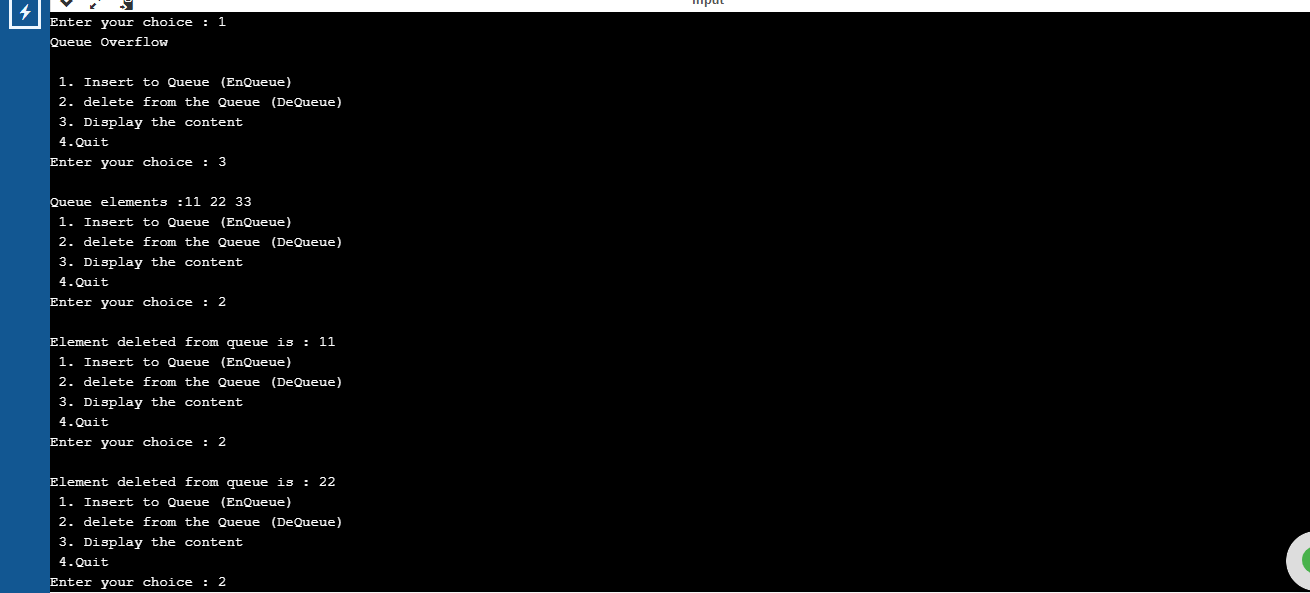




Program 4

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  |  |
|  | int front=-1; |
|  | int rear=-1; |
|  | int MAX=0; |
|  | int queue[MAX]; |
|  |  |
|  | void Enque(int); |
|  | int Deque(); |
|  | void display(); |
|  | int main() |
|  | { |
|  | int option; |
|  | int item; |
|  | do{ |
|  | printf("Enter the maximum value of the queue"); |
|  | scanf("%d",&MAX); |
|  | printf("\n 1. Insert to Queue (EnQueue)"); |
|  | printf("\n 2. delete from the Queue (DeQueue)"); |
|  | printf("\n 3. Display the content "); |
|  | printf("\n 4. Exit\n"); |
|  | printf("Enter the option :"); |
|  | scanf("%d",&option); |
|  | switch(option) |
|  | { |
|  | case 1: printf("Enter the element\n"); |
|  | scanf("%d",&item); |
|  | Enque(item); |
|  | break; |
|  | case 2: item=Deque(); |
|  | if(item==-999) |
|  | printf("Queue is empty"); |
|  | else |
|  | printf("Removed element from the queue %d",item); |
|  | break; |
|  | case 3: display(); |
|  | break; |
|  | case 4: exit(0); |
|  | } |
|  | } while (option!=4); |
|  | return 0; |
|  | } |
|  |  |
|  | void Enque(int ele) |
|  | { |
|  | if(((front == 0 && rear == MAX - 1))|| (front == rear + 1) ) |
|  | { |
|  | printf("Queue is full\n");return; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | rear=(rear+1)%MAX; |
|  | queue[rear]=ele; |
|  | if(front ==-1) |
|  | front=0; |
|  |  |
|  |  |
|  | } |
|  | } |
|  | int Deque() |
|  | { |
|  | int item; |
|  | if((front == -1)&&(rear == -1)) |
|  | { |
|  |  |
|  | return(-999); |
|  | } |
|  | else |
|  | { |
|  | item=queue[front]; |
|  |  |
|  | if(front==rear) |
|  | { |
|  | front=-1; |
|  | rear=-1; |
|  | } |
|  | else |
|  | { |
|  | front=(front+1)%MAX; |
|  | } |
|  | return item; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | int i; |
|  | if(((front==-1)&& (rear==-1))|| (front==rear)) |
|  | { |
|  |  |
|  | printf("Queue is empty\n");return; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | printf("\n Queue contents:\n"); |
|  | for(i=front;i<=rear;i++) |
|  | printf("%d", queue[i]); |
|  | } |
|  | } |

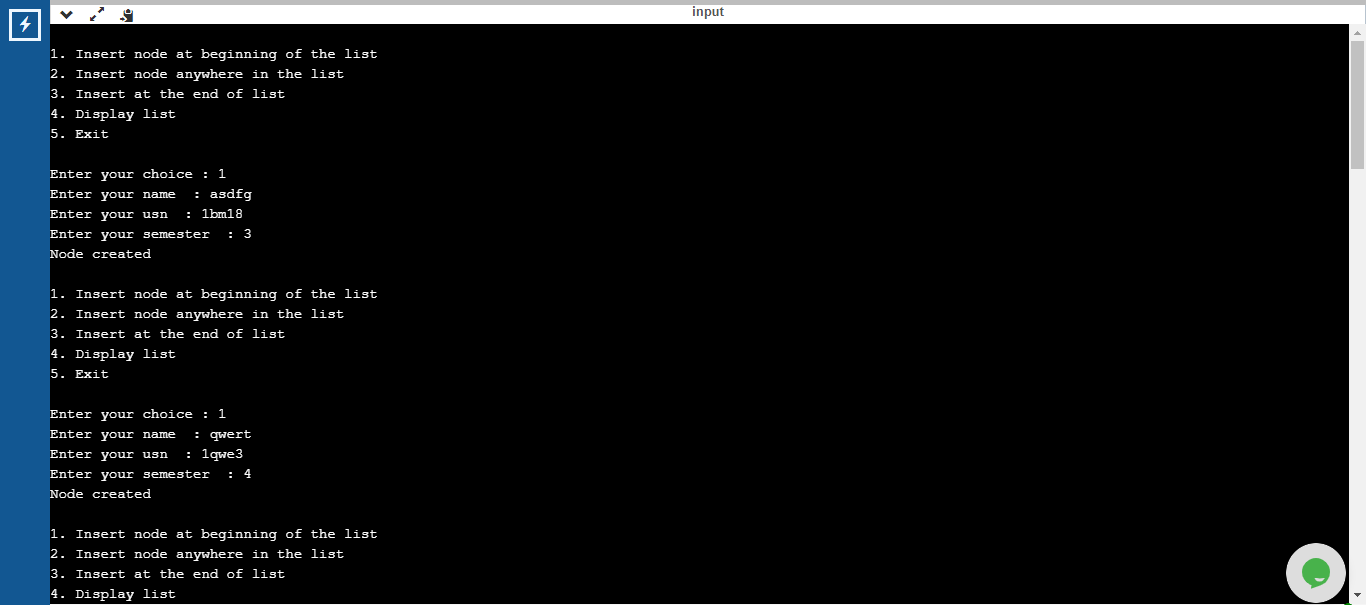




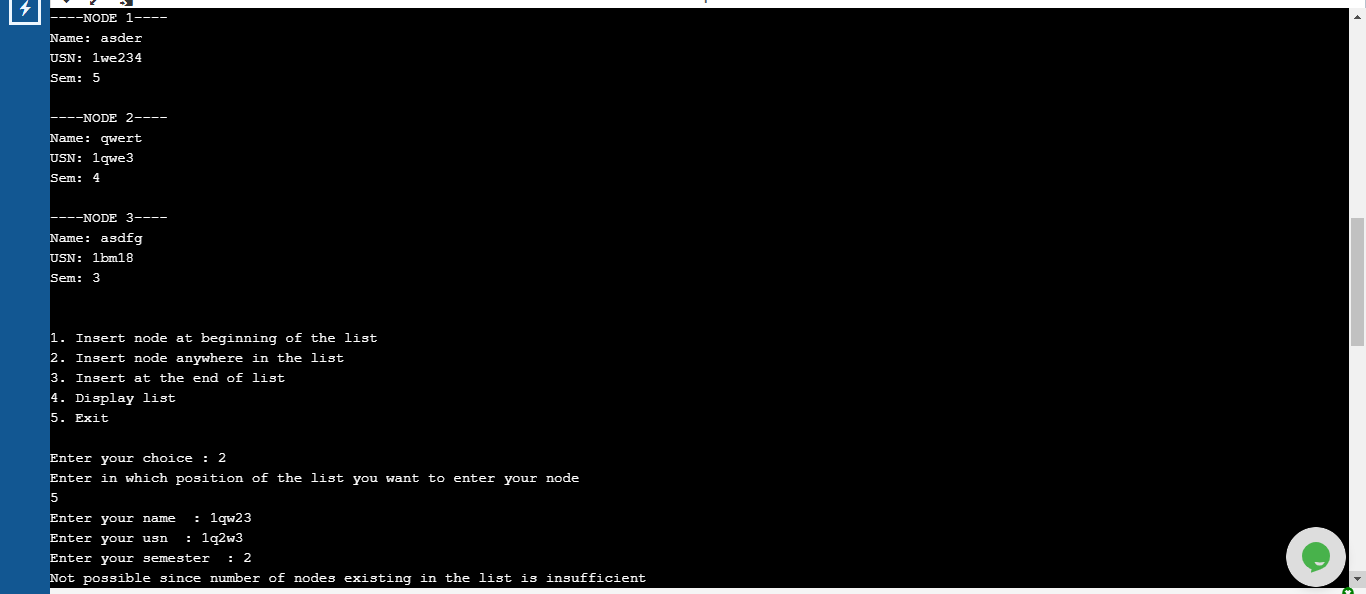


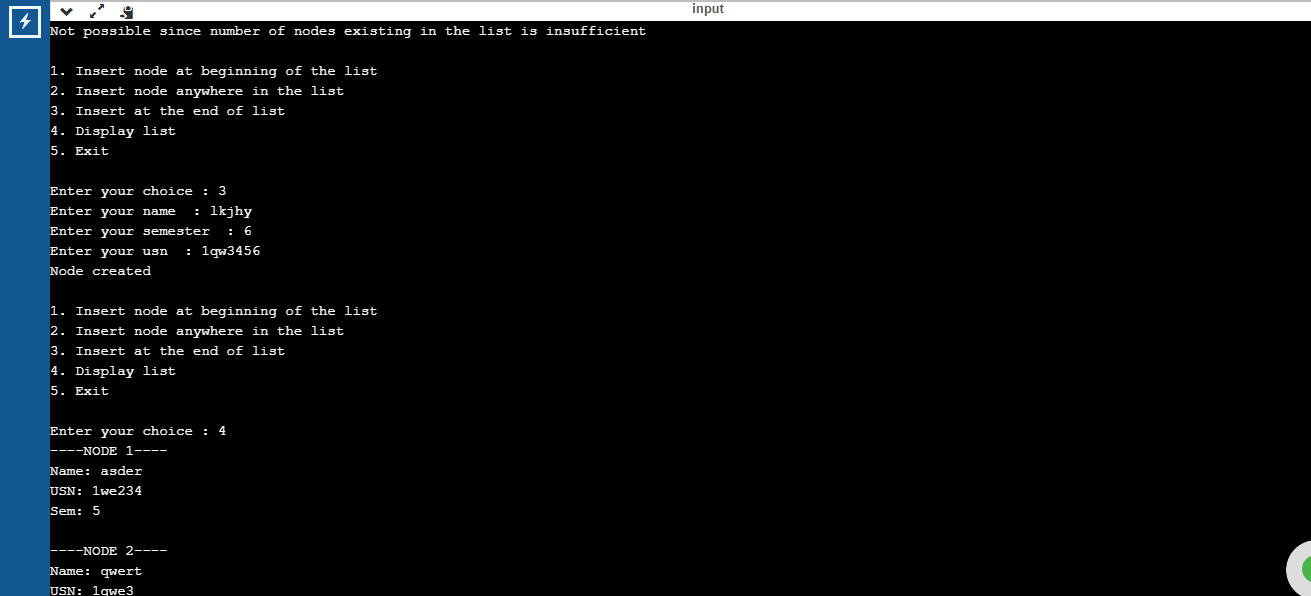
Program 5

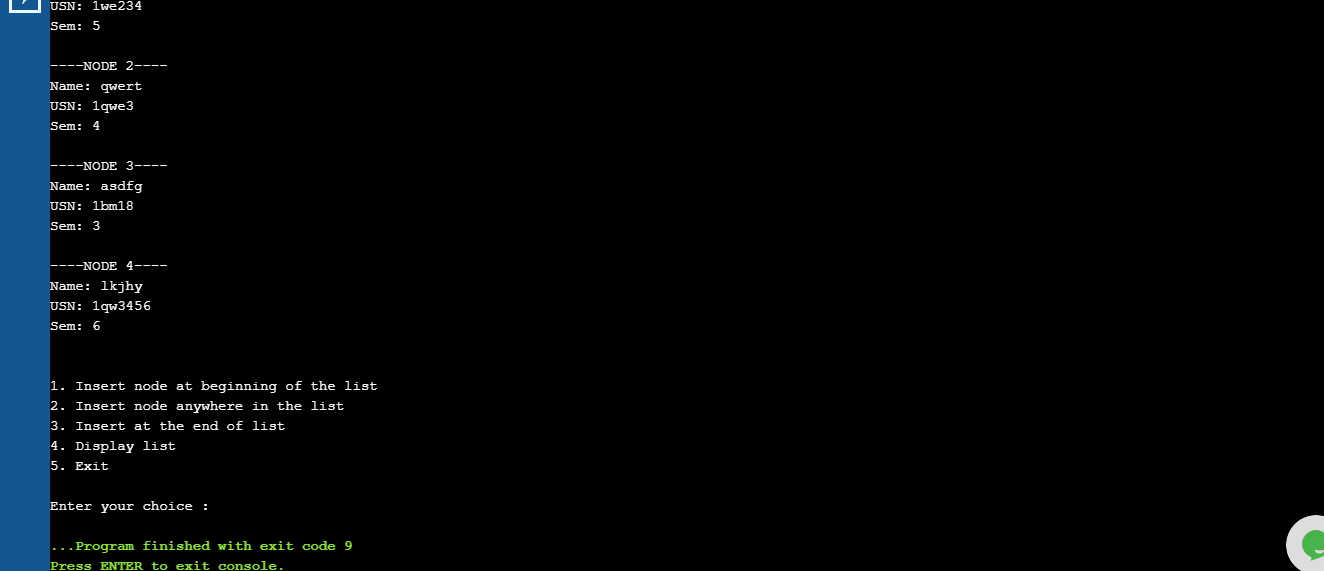
|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  | struct node |
|  | { |
|  | int sem; |
|  | char name[50]; |
|  | char usn[50]; |
|  | struct node \*next; |
|  | }; |
|  | struct node \*head= NULL; |
|  | int c=0; |
|  | void Insertbegining() |
|  | { |
|  | struct node \*newnode; |
|  | int s; |
|  | char a[50],b[50]; |
|  | printf("Enter your name : "); |
|  | scanf("%s",a); |
|  | printf("Enter your usn : "); |
|  | scanf("%s",b); |
|  | printf("Enter your semester : "); |
|  | scanf("%d",&s); |
|  |  |
|  | newnode=(struct node\*)malloc(sizeof(struct node)); |
|  | newnode->sem =s; |
|  | strcpy(newnode->name,a); |
|  | strcpy(newnode->usn,b); |
|  |  |
|  | newnode->next=head; |
|  | head=newnode; |
|  | c++; |
|  | printf("Node created\n"); |
|  | } |
|  | void Insertanypos(int p) |
|  | { |
|  | struct node \*newnode; |
|  | int s; |
|  | char a[30],b[30]; |
|  | printf("Enter your name : "); |
|  | scanf("%s",a); |
|  | printf("Enter your usn : "); |
|  | scanf("%s",b); |
|  | printf("Enter your semester : "); |
|  | scanf("%d",&s); |
|  |  |
|  | newnode=(struct node\*)malloc(sizeof(struct node)); |
|  | newnode->sem =s; |
|  | strcpy(newnode->name,a); |
|  | strcpy(newnode->usn,b); |
|  | if(p==1) |
|  | { |
|  | printf("Node of linked list is inserted in the first position\n"); |
|  | newnode->next=head; |
|  | head=newnode; |
|  | c++; |
|  | } |
|  | else if(head==NULL && p>1) |
|  | { |
|  | printf("the list is empty and node cannot be created\n"); |
|  | return; |
|  | } |
|  | else if(p>(c+1)) |
|  | { |
|  | printf("Not possible since number of nodes existing in the list is insufficient\n"); |
|  | return; |
|  | } |
|  | else |
|  | { |
|  | struct node \*temp1; |
|  | struct node \*temp2; |
|  | int count=1; |
|  | temp1=head; |
|  | while(count<(p-1)) |
|  | { |
|  | temp1= temp1->next; |
|  | count++; |
|  | } |
|  | temp2= temp1->next; |
|  | temp1->next=newnode; |
|  | newnode->next=temp2; |
|  | c++; |
|  | printf("Node inserted at %d position in linked list\n",p); |
|  | } |
|  | } |
|  |  |
|  | void Insertend() |
|  | { |
|  | struct node \*newnode; |
|  | struct node \*temp; |
|  | int s; |
|  | char n[30],u[30]; |
|  | printf("Enter your name : "); |
|  | scanf("%s",n); |
|  | printf("Enter your semester : "); |
|  | scanf("%d",&s); |
|  | printf("Enter your usn : "); |
|  | scanf("%s",u); |
|  | newnode=(struct node\*)malloc(sizeof(struct node)); |
|  | newnode->sem =s; |
|  | strcpy(newnode->name,n); |
|  | strcpy(newnode->usn,u); |
|  | if (head==NULL) |
|  | { |
|  | newnode->next=NULL; |
|  | head=newnode; |
|  | printf("first node of linked list created\n"); |
|  | c++; |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | temp->next=newnode; |
|  | newnode->next=NULL; |
|  | c++; |
|  | printf("Node created\n"); |
|  | } |
|  | } |
|  | void display() |
|  | { |
|  | struct node \*ptr; |
|  | ptr=head; |
|  | int i=1; |
|  |  |
|  | if(ptr==NULL) |
|  | { |
|  | printf("Linked list is empty!\n"); |
|  | } |
|  | else |
|  | { |
|  | while(ptr!= NULL) |
|  | { |
|  | printf("----NODE %d----\n",i); |
|  | printf("Name: %s\n",ptr->name); |
|  | printf("USN: %s\n",ptr->usn); |
|  | printf("Sem: %d\n",ptr->sem); |
|  | printf("\n"); |
|  | i++; |
|  | ptr=ptr->next; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  |  |
|  | int main() |
|  | { |
|  | int choice,pos; |
|  | do |
|  | { |
|  |  |
|  | printf("\n1. Insert node at beginning of the list\n2. Insert node anywhere in the list\n3. Insert at the end of list\n4. Display list\n5. Exit\n"); |
|  | printf("\nEnter your choice : "); |
|  | scanf("%d",&choice); |
|  | if(choice==5) |
|  | break; |
|  | switch(choice) |
|  | { |
|  | case 1: |
|  | Insertbegining(); |
|  | break; |
|  |  |
|  | case 2: |
|  | printf("Enter in which position of the list you want to enter your node\n"); |
|  | scanf("%d",&pos); |
|  | Insertanypos(pos); |
|  | break; |
|  |  |
|  | case 3: |
|  | Insertend(); |
|  | break; |
|  |  |
|  | case 4: |
|  | display(); |
|  | break; |
|  |  |
|  | default: |
|  | printf("Wrong choice!\n"); |
|  | break; |
|  | } |
|  | }while(choice!=5); |
|  | return 0; |
|  | } |











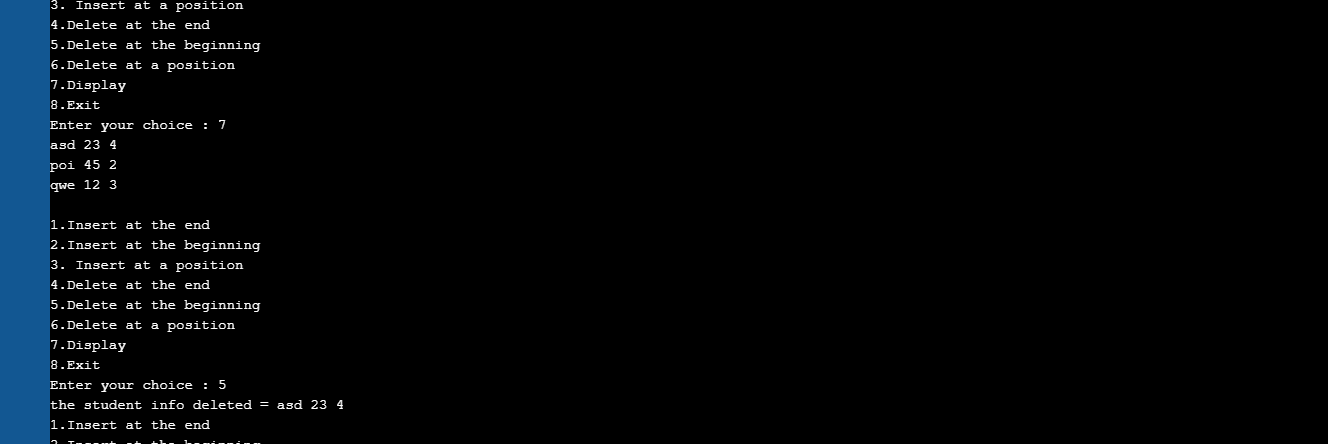
Program 6

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  | void create(); |
|  | void display(); |
|  | void insertpos(int); |
|  | void insert\_beg(); |
|  | void delete(); |
|  | void delpos(int); |
|  | void del\_beg(); |
|  | struct node |
|  | { |
|  | int sem,usn; |
|  | char name[20]; |
|  | struct node \*next; |
|  | }; |
|  |  |
|  | struct node \*head=NULL; |
|  | int count=0; |
|  | int main() |
|  | { |
|  | int choice,ele,a; |
|  |  |
|  | do |
|  | { |
|  | printf("\n1.Insert at the end \n2.Insert at the beginning \n3. Insert at a position \n4.Delete at the end \n5.Delete at the beginning \n6.Delete at a position \n7.Display\n8.Exit"); |
|  | printf("\nEnter your choice : "); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: create(); break; |
|  | case 2:insert\_beg(); |
|  | break; |
|  | case 3: printf("Enter the position to be inserted\n"); |
|  | scanf("%d",&ele); |
|  | insertpos(ele); |
|  | break; |
|  | case 4:delete();break; |
|  | case 5:del\_beg();break; |
|  | case 6:printf("Enter the position to be deleted in"); |
|  | scanf("%d",&a); |
|  | delpos(a); |
|  |  |
|  | break; |
|  | case 7:display(); |
|  | break; |
|  |  |
|  | case 8:exit(0); |
|  | } |
|  |  |
|  | }while(choice!=8); |
|  | } |
|  |  |
|  | void create() |
|  | { |
|  | struct node \*newnode,\*temp; |
|  | int sem1,usn1; |
|  | char name1[20]; |
|  | printf("Enter the name USN semester of student : "); |
|  | scanf("%s %d %d",name1,&usn1,&sem1); |
|  | newnode =(struct node \*) malloc (sizeof(struct node)); |
|  |  |
|  | strcpy(newnode->name,name1); |
|  | newnode->usn=usn1; |
|  | newnode->sem=sem1; |
|  | if (head==NULL) |
|  | { |
|  | newnode->next=NULL; |
|  | head=newnode; |
|  | printf("Node created\n"); |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | temp->next=newnode; |
|  | newnode->next=NULL; |
|  | printf("Node created\n"); |
|  | count++; |
|  | } |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | struct node \*ptr=NULL; |
|  | ptr=head; |
|  |  |
|  | if(ptr==NULL) |
|  | { |
|  | printf("Nothing to print\n"); |
|  | } |
|  | else |
|  | { |
|  | while(ptr!=NULL) |
|  | { |
|  | printf("%s ",ptr->name); |
|  | printf("%d ",ptr->usn); |
|  | printf("%d ",ptr->sem); |
|  | printf("\n"); |
|  | ptr=ptr->next; |
|  | } |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void insertpos(int p) |
|  | { |
|  | struct node \*newnode; |
|  | int sem1,usn1; |
|  | char name1[20]; |
|  | if(count+2<p) |
|  | printf("the position exceeds the number of nodes"); |
|  | else if(head==NULL&& p>1) |
|  | { |
|  | printf("node empty enter in first position"); |
|  | } |
|  | else |
|  | { |
|  | printf("Enter the name USN semester of student : "); |
|  | scanf("%s %d %d",name1,&usn1,&sem1); |
|  | newnode =(struct node \*) malloc (sizeof(struct node)); |
|  |  |
|  | strcpy(newnode->name,name1); |
|  | newnode->usn=usn1; |
|  | newnode->sem=sem1; |
|  |  |
|  |  |
|  |  |
|  | if(p==1) |
|  | { |
|  | printf("inserted at the beginning\n"); |
|  | newnode->next=head; |
|  | head=newnode; |
|  | count++; |
|  |  |
|  | } |
|  |  |
|  | else |
|  | { |
|  |  |
|  |  |
|  | int i; |
|  | struct node \*temp1; |
|  | temp1=head; |
|  | for(i=2;i<p;i++) |
|  | { |
|  | temp1= temp1->next; |
|  | } |
|  | newnode->next=temp1->next; |
|  | temp1->next=newnode; |
|  |  |
|  | printf("Node inserted at %d position in linked list\n",p); |
|  | count++; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void insert\_beg() |
|  | { |
|  | struct node \*newnode; |
|  | int sem1,usn1; |
|  | char name1[20]; |
|  | printf("Enter the name USN semester of student : "); |
|  | scanf("%s %d %d",name1,&usn1,&sem1); |
|  | newnode =(struct node \*) malloc (sizeof(struct node)); |
|  |  |
|  | strcpy(newnode->name,name1); |
|  | newnode->usn=usn1; |
|  | newnode->sem=sem1; |
|  | newnode->next=head; |
|  | head=newnode; |
|  | count++; |
|  | } |
|  |  |
|  | void delete() |
|  | { |
|  | struct node \*temp=NULL; |
|  | int sem1,usn1; |
|  | char name1[20]; |
|  | if(head==NULL) |
|  | printf("linked list is empty"); |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | strcpy(name1,temp->next->name); |
|  | sem1=temp->next->sem; |
|  | usn1=temp->next->usn; |
|  | printf("the student info deleted = %s %d %d",name1,usn1,sem1); |
|  | temp->next=NULL; |
|  | count--; |
|  | } |
|  | } |
|  |  |
|  |  |
|  | void del\_beg() |
|  | { |
|  | struct node \*temp=NULL; |
|  | int sem1,usn1; |
|  | char name1[20]; |
|  | if(head==NULL) |
|  | printf("linked list is empty"); |
|  | else |
|  | { |
|  | strcpy(name1,head->name); |
|  | sem1=head->sem; |
|  | usn1=head->usn; |
|  | printf("the student info deleted = %s %d %d",name1,usn1,sem1); |
|  | temp=head; |
|  | head=temp->next; |
|  | free(temp); |
|  | count--; |
|  | } |
|  | } |
|  | void delpos(int p) |
|  | { |
|  | struct node \*temp=NULL; |
|  | int sem1,usn1; |
|  | char name1[20]; |
|  | if(head==NULL) |
|  | printf("linked list is empty"); |
|  | else if(count+1<p) |
|  | printf("the position exceeds the number of nodes"); |
|  | else if(p==1) |
|  | { |
|  | strcpy(name1,head->name); |
|  | sem1=head->sem; |
|  | usn1=head->usn; |
|  | printf("the student info deleted = %s %d %d",name1,usn1,sem1); |
|  | temp=head; |
|  | head=temp->next; |
|  | free(temp); |
|  | count--; |
|  | } |
|  | else |
|  | { |
|  | int i; |
|  | struct node \*temp,\*ptr; |
|  | temp=head; |
|  | for(i=2;i<p;i++) |
|  | { |
|  | temp= temp->next; |
|  | } |
|  |  |
|  | strcpy(name1,temp->next->name); |
|  | sem1=temp->next->sem; |
|  | usn1=temp->next->usn; |
|  | printf("the student info deleted = %s %d %d",name1,usn1,sem1); |
|  | ptr=temp->next; |
|  | temp->next=temp->next->next; |
|  | free(ptr); |
|  | count--; |
|  | } |
|  |  |
|  | } |

Program 1



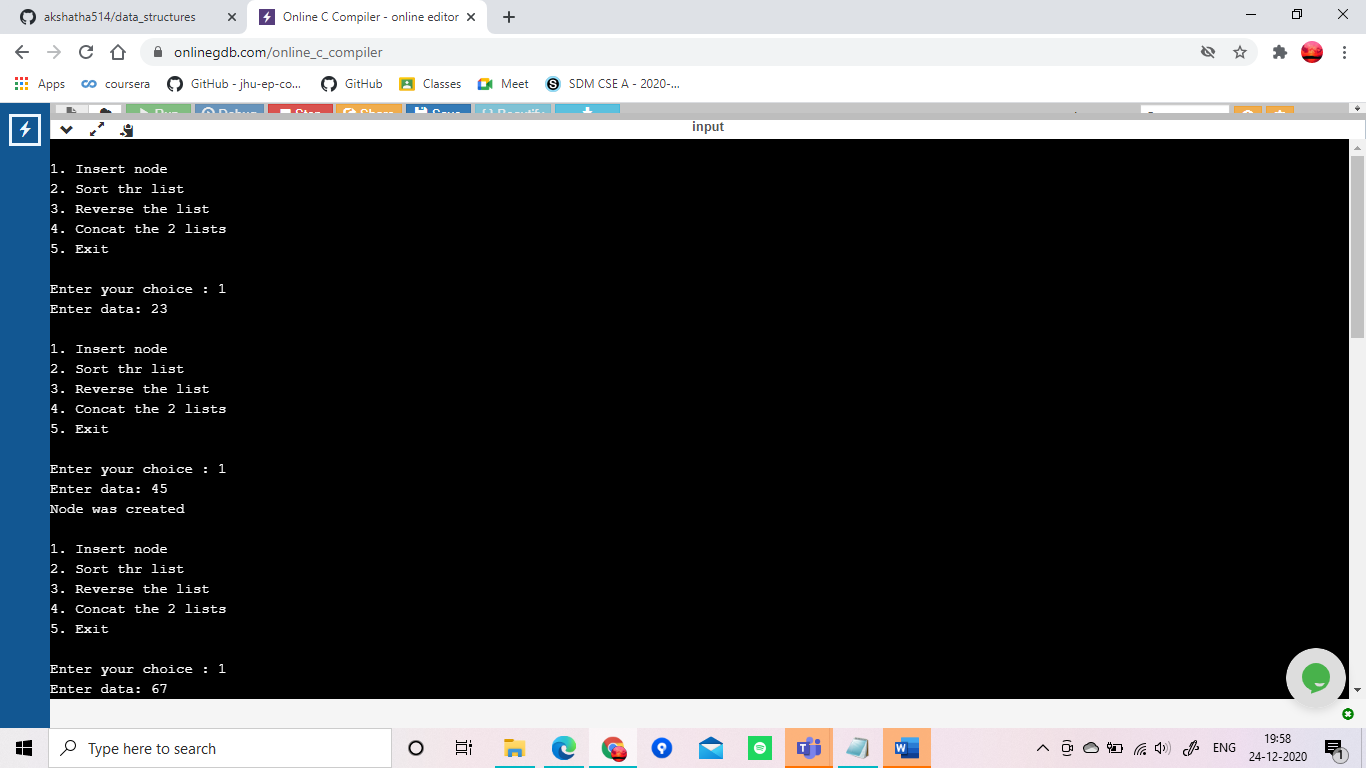


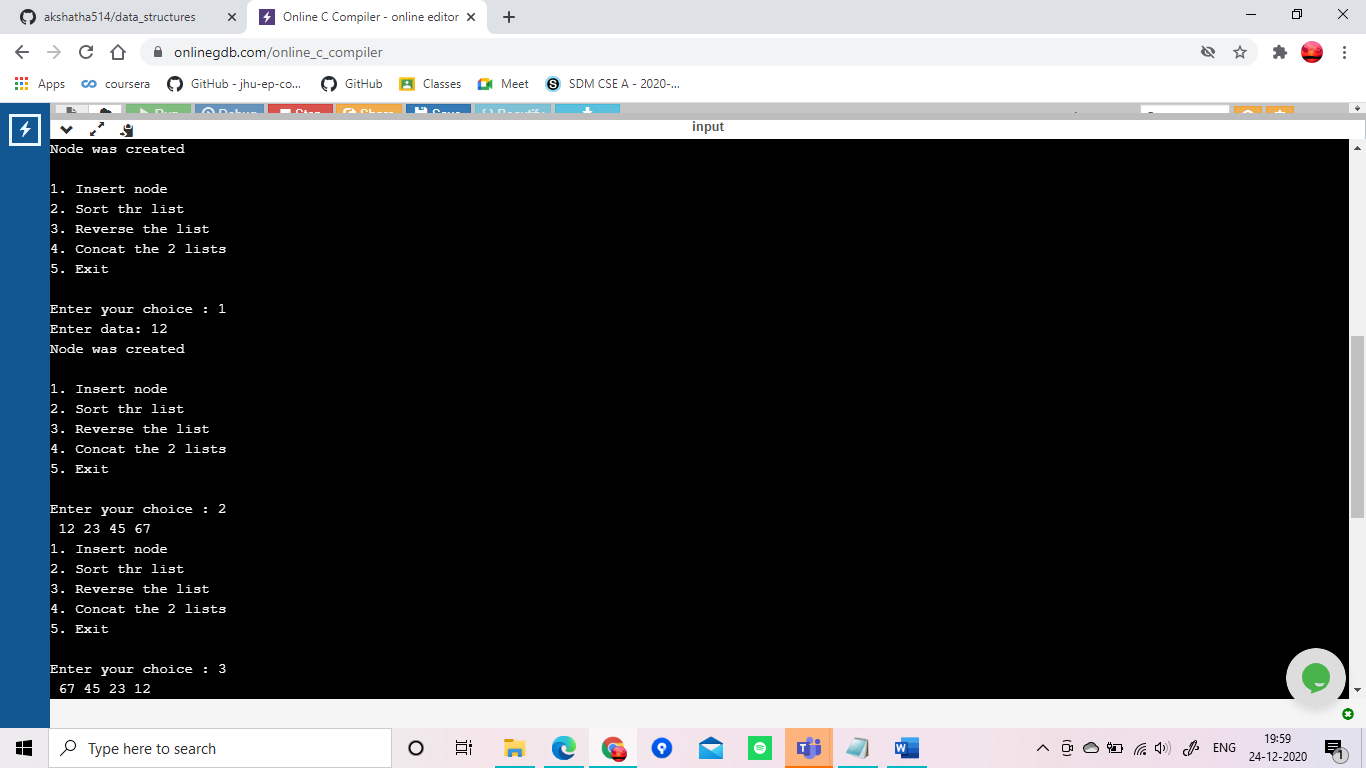


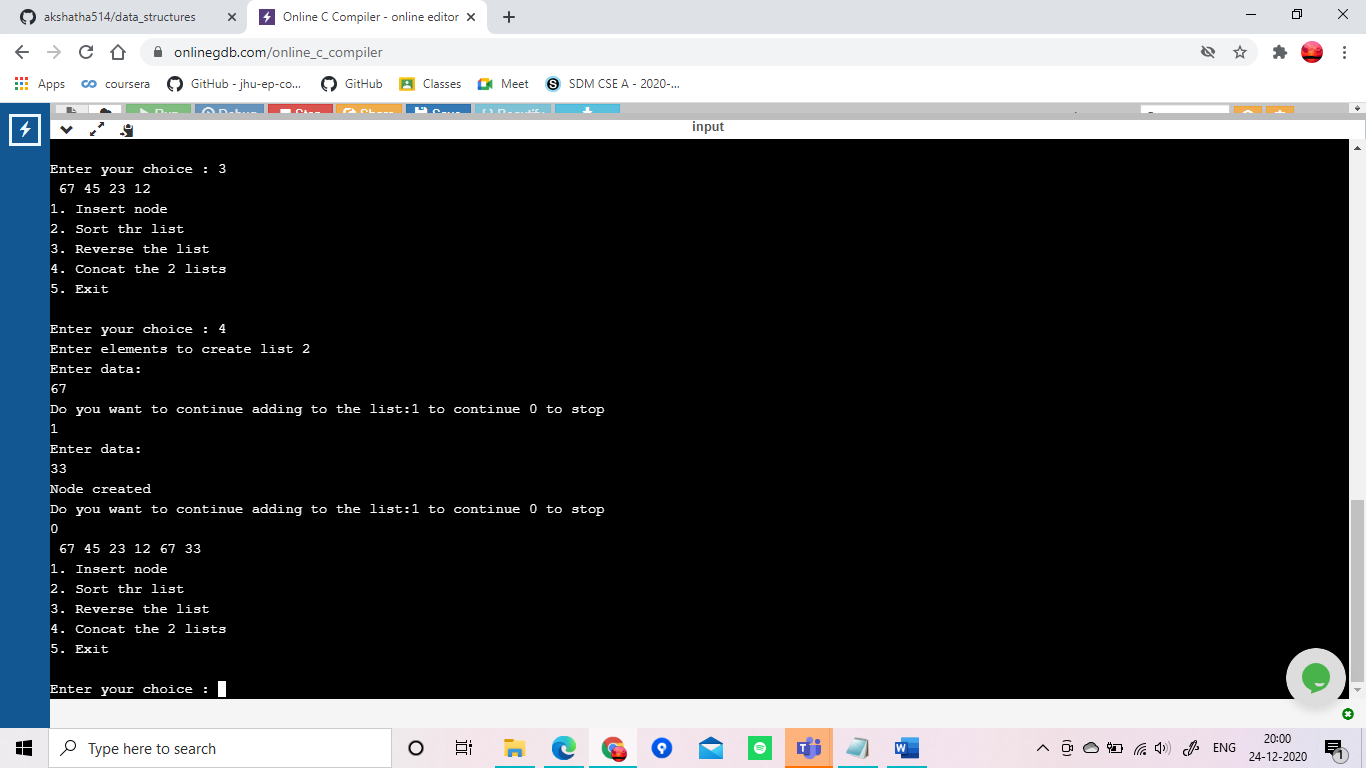


Program 7

|  |
| --- |
|  |
| #include<stdio.h> |
|  | #include <stdlib.h> |
|  | #include <string.h> |
|  | struct node |
|  | { |
|  | int sem; |
|  | struct node \*next; |
|  | }; |
|  | struct node \*head= NULL; |
|  | struct node \*head2= NULL; |
|  | int c=0; |
|  | void Insert() |
|  | { |
|  | struct node \*newnode; |
|  | struct node \*temp; |
|  | int s; |
|  | printf("Enter data: "); |
|  | scanf("%d",&s); |
|  | newnode=(struct node\*)malloc(sizeof(struct node)); |
|  | newnode->sem =s; |
|  | if (head==NULL) |
|  | { |
|  | newnode->next=NULL; |
|  | head=newnode; |
|  |  |
|  | c++; |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | temp->next=newnode; |
|  | newnode->next=NULL; |
|  | c++; |
|  | printf("Node was created\n"); |
|  | } |
|  | } |
|  | void Insert2() |
|  | { |
|  | struct node \*newnode; |
|  | struct node \*temp; |
|  | int s,y; |
|  | printf("Enter elements to create list 2\n"); |
|  | do |
|  | { |
|  | printf("Enter data: \n"); |
|  | scanf("%d",&s); |
|  | newnode=(struct node\*)malloc(sizeof(struct node)); |
|  | newnode->sem =s; |
|  | if (head2==NULL) |
|  | { |
|  | newnode->next=NULL; |
|  | head2=newnode; |
|  |  |
|  | c++; |
|  | } |
|  | else |
|  | { |
|  | temp=head2; |
|  | while(temp->next!=NULL) |
|  | { |
|  | temp=temp->next; |
|  | } |
|  | temp->next=newnode; |
|  | newnode->next=NULL; |
|  | c++; |
|  | printf("Node created\n"); |
|  | } |
|  | printf("Do you want to continue adding to the list:1 to continue 0 to stop\n"); |
|  | scanf("%d",&y); |
|  | }while(y!=0); |
|  | } |
|  |  |
|  |  |
|  | void bubbleSort() |
|  | { |
|  | int swapped, i; |
|  | struct node \*ptr1; |
|  | struct node \*lptr = NULL; |
|  |  |
|  |  |
|  | if (head == NULL) |
|  | return; |
|  |  |
|  | do |
|  | { |
|  | swapped = 0; |
|  | ptr1 = head; |
|  |  |
|  | while (ptr1->next != lptr) |
|  | { |
|  | if (ptr1->sem > ptr1->next->sem) |
|  | { |
|  | int temp = ptr1->sem; |
|  | ptr1->sem = ptr1->next->sem; |
|  | ptr1->next->sem = temp; |
|  | swapped = 1; |
|  | } |
|  | ptr1 = ptr1->next; |
|  | } |
|  | lptr = ptr1; |
|  | } |
|  | while (swapped); |
|  | } |
|  |  |
|  | void reverse() |
|  | { |
|  | struct node\* prev = NULL; |
|  | struct node\* current = head; |
|  | struct node\* next = NULL; |
|  | while (current != NULL) { |
|  | next = current->next; |
|  | current->next = prev; |
|  | prev = current; |
|  | current = next; |
|  | } |
|  | head= prev; |
|  | } |
|  |  |
|  | void concat() |
|  | { |
|  | struct node \*ptr; |
|  | if(head==NULL) |
|  | { |
|  | head=head2; |
|  | } |
|  | if(head2==NULL) |
|  | { |
|  | head2=head; |
|  | } |
|  | ptr=head; |
|  | while(ptr->next!=NULL) |
|  | ptr=ptr->next; |
|  | ptr->next=head2; |
|  | } |
|  | void display1() |
|  | { |
|  | struct node \*ptr; |
|  | ptr=head; |
|  | int i=1; |
|  |  |
|  | if(ptr==NULL) |
|  | { |
|  | printf("The Linked list is empty\n"); |
|  | } |
|  | else |
|  | { |
|  | while(ptr!= NULL) |
|  | { |
|  | printf(" %d",ptr->sem); |
|  | i++; |
|  | ptr=ptr->next; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |
|  | void display2() |
|  | { |
|  | struct node \*ptr; |
|  | ptr=head2; |
|  | int i=1; |
|  |  |
|  | if(ptr==NULL) |
|  | { |
|  | printf("The Linked list is empty\n"); |
|  | } |
|  | else |
|  | { |
|  | while(ptr!= NULL) |
|  | { |
|  |  |
|  |  |
|  | printf(" %d",ptr->sem); |
|  | printf("\n"); |
|  | i++; |
|  | ptr=ptr->next; |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | void main() |
|  | { |
|  | int choice,pos; |
|  | do |
|  | { |
|  |  |
|  | printf("\n1. Insert node \n2. Sort thr list\n3. Reverse the list\n4. Concat the 2 lists \n5. Exit\n"); |
|  | printf("\nEnter your choice : "); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: |
|  | Insert(); |
|  | break; |
|  |  |
|  | case 2: |
|  | bubbleSort(); |
|  | display1(); |
|  | break; |
|  |  |
|  | case 3: |
|  | reverse(); |
|  | display1(); |
|  | break; |
|  |  |
|  | case 4: |
|  | Insert2(); |
|  | concat(); |
|  | display1(); |
|  | break; |
|  |  |
|  | case 5: |
|  | break; |
|  |  |
|  | default: |
|  | printf("Wrong choice!\n"); |
|  | break; |
|  | } |
|  | }while(choice!=5); |
|  | } |
|  |  |



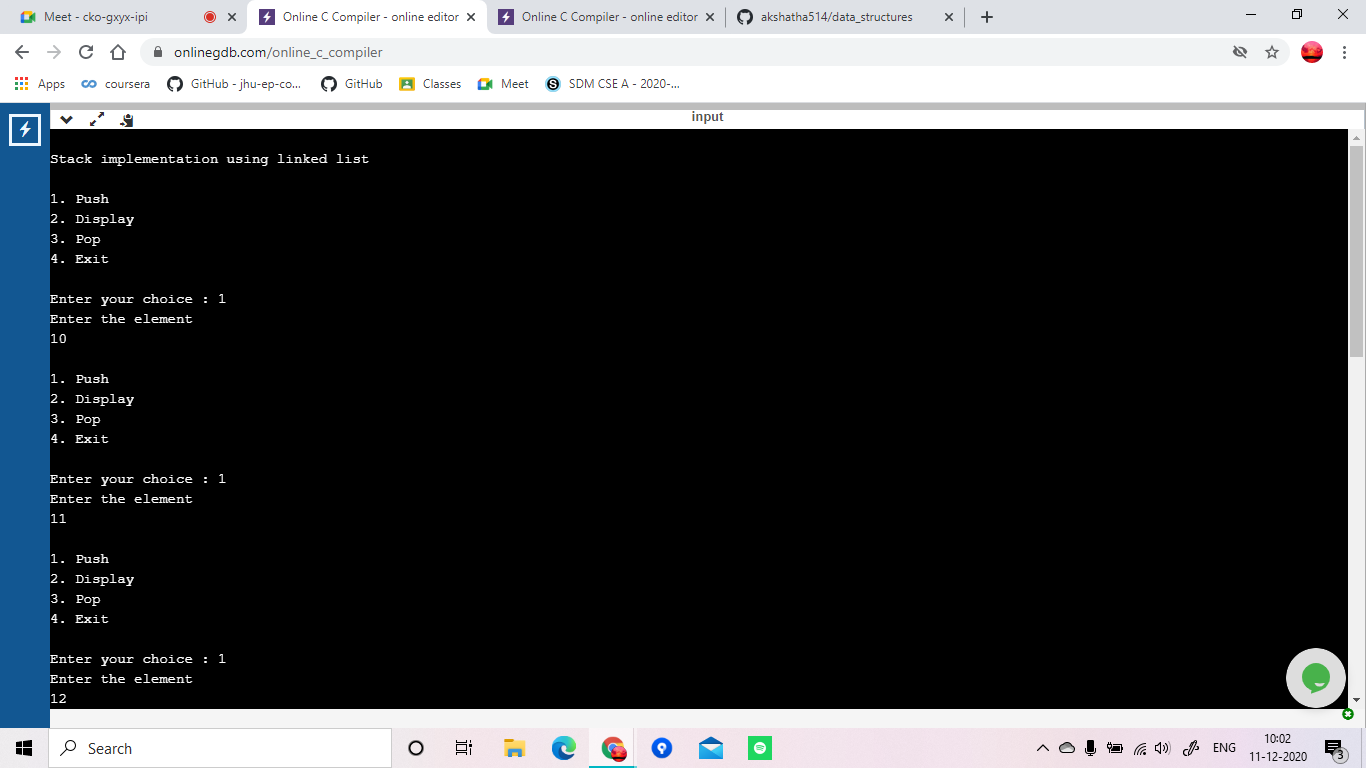


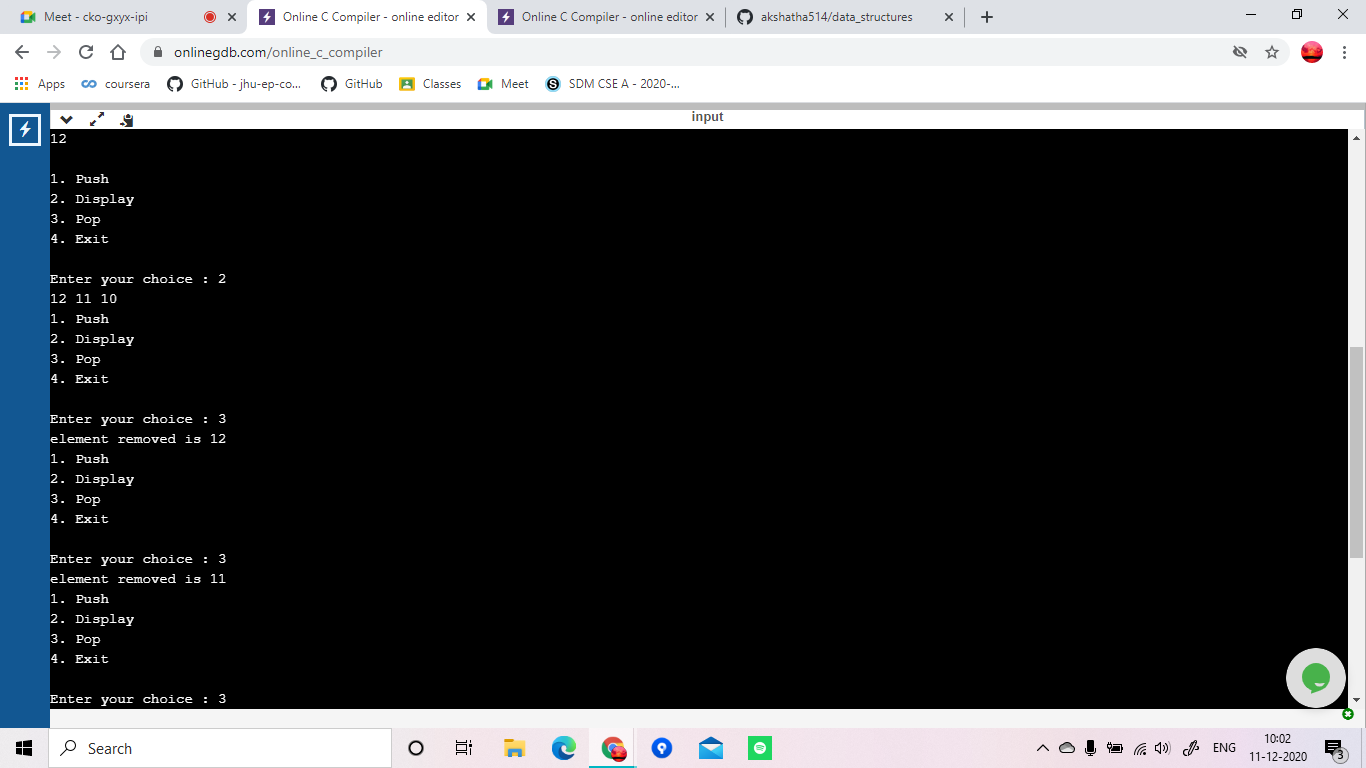


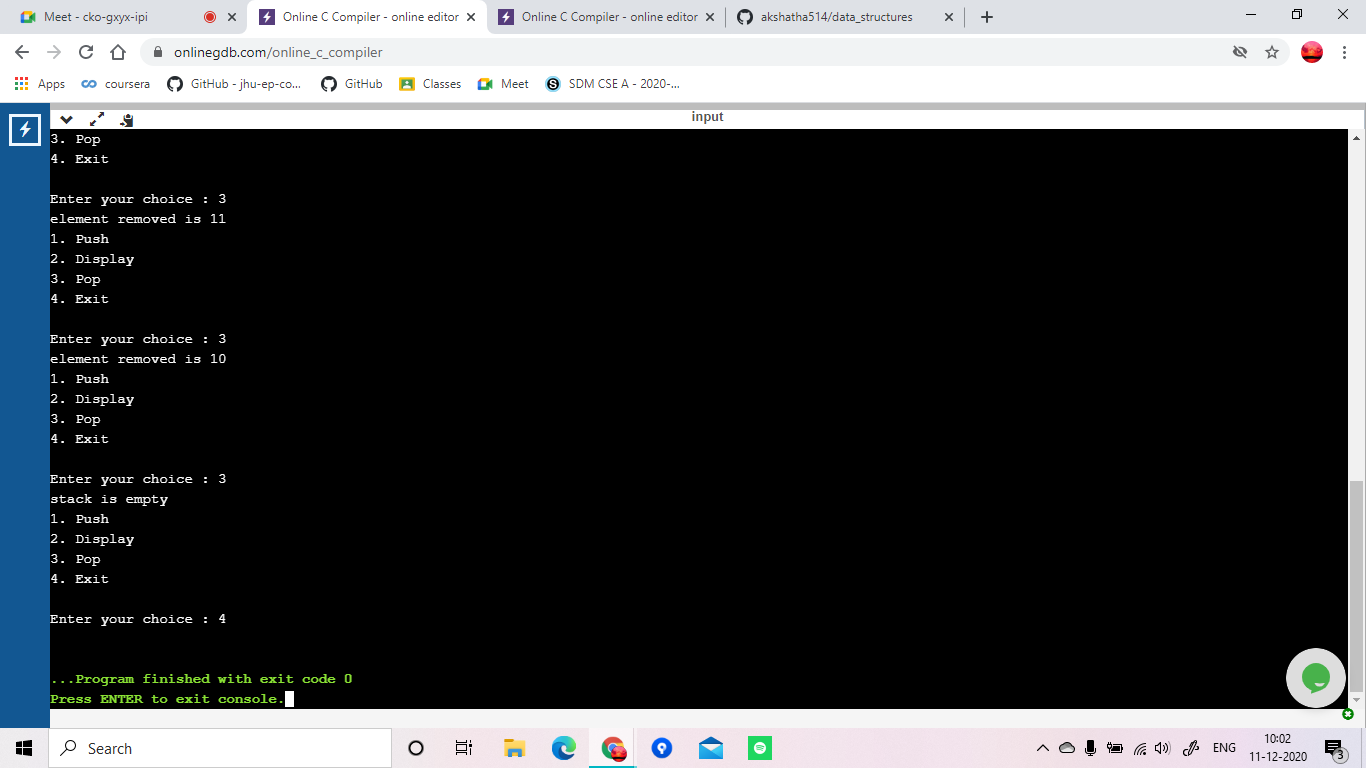
Program 8

|  |
| --- |
| #include <stdio.h> |
|  | #include<stdlib.h> |
|  |  |
|  | void push(); |
|  | void pop(); |
|  | void display(); |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  | struct node \*top=NULL; |
|  |  |
|  | int main() |
|  | { |
|  | int choice; |
|  | printf("\nStack implementation using linked list\n"); |
|  | do |
|  | { |
|  | printf("\n1. Push \n2. Display \n3. Pop\n4. Exit\n"); |
|  | printf("\nEnter your choice : "); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: push(); break; |
|  | case 2: display();break; |
|  | case 3: pop(); break; |
|  | case 4:exit(0); |
|  | } |
|  |  |
|  | }while(choice!=4); |
|  | } |
|  |  |
|  |  |
|  | void push() |
|  | { |
|  | int item; |
|  | struct node \*newnode; |
|  | printf("Enter the element\n"); |
|  | scanf("%d",&item); |
|  |  |
|  | newnode=(struct node\*)malloc(sizeof(struct node)); |
|  | newnode->data=item; |
|  | newnode->next=NULL; |
|  | if(top==NULL) |
|  | top=newnode; |
|  | else |
|  | newnode->next=top; |
|  | top=newnode; |
|  | } |
|  | void pop() |
|  | { |
|  | if(top==NULL) |
|  | printf("stack is empty"); |
|  | else |
|  | { |
|  |  |
|  | printf("element removed is %d", top->data); |
|  |  |
|  | top=top->next; |
|  |  |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | struct node \*temp; |
|  | temp=top; |
|  | if(top==NULL) |
|  | printf("Stack is empty"); |
|  | while(temp!=NULL) |
|  | { |
|  | printf("%d ",temp->data); |
|  | temp=temp->next; |
|  | } |
|  |  |
|  | } |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  | void insert(); |
|  | void display(); |
|  | void del(); |
|  |  |
|  | struct node \*rear=NULL, \*front =NULL; |
|  |  |
|  | int main() |
|  | { |
|  | printf("\nQueue implementation using linked list\n"); |
|  | int choice; |
|  |  |
|  | do |
|  | { |
|  |  |
|  | printf("\n1. Create \n2. Display \n3. Delete \n4. Exit \n"); |
|  | printf("\nEnter your choice : "); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: insert(); break; |
|  | case 2: display();break; |
|  | case 3: del(); break; |
|  | case 4: exit(0); |
|  |  |
|  | } |
|  | }while(choice!=4); |
|  | } |
|  |  |
|  | void insert() |
|  | { |
|  | struct node \*newnode; |
|  | newnode=(struct node \*) malloc(sizeof(struct node)); |
|  | printf("Enter the element:\n"); |
|  | scanf("%d",&newnode->data); |
|  | newnode->next=NULL; |
|  |  |
|  | if(rear==NULL) |
|  | { |
|  | rear=newnode; |
|  | front=newnode; |
|  |  |
|  | } |
|  | else |
|  | { |
|  | rear->next=newnode; |
|  | rear=newnode; |
|  | } |
|  | } |
|  |  |
|  | void del() |
|  | { |
|  | if(front==NULL) |
|  | { |
|  | printf("Queue is empty\n");return; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | printf("Deleted element is %d",front->data); |
|  | if(front==rear) |
|  | { |
|  |  |
|  | front=NULL; rear=NULL; |
|  | } |
|  | else |
|  | front=front->next; |
|  | } |
|  | } |
|  |  |
|  | void display() |
|  | { |
|  | struct node \*temp; |
|  | if(front ==NULL) |
|  | { |
|  | printf("Queue is empty"); |
|  | return; |
|  | } |
|  | temp=front; |
|  | while (temp !=NULL) |
|  | { |
|  | printf("%d ",temp->data); |
|  | temp=temp->next; |
|  | } |
|  |  |
|  | } |

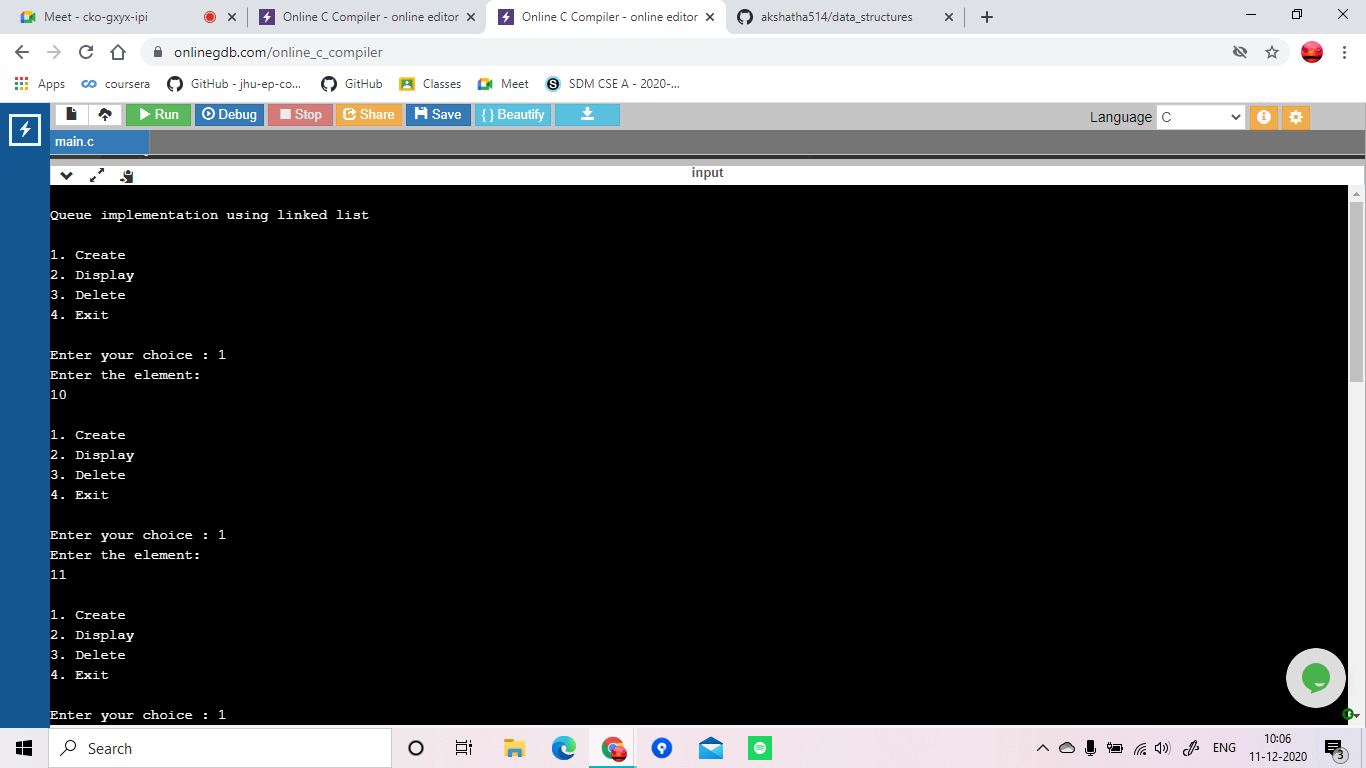
Stack program

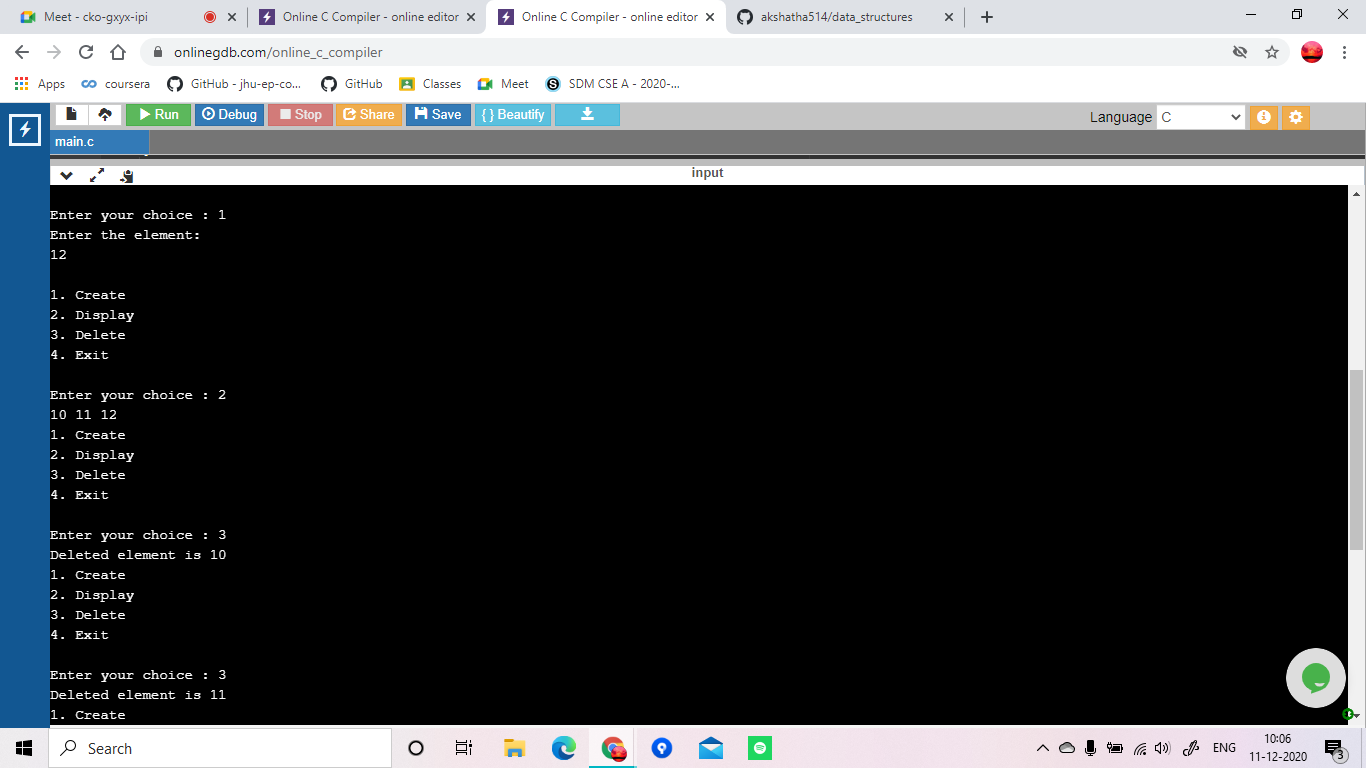


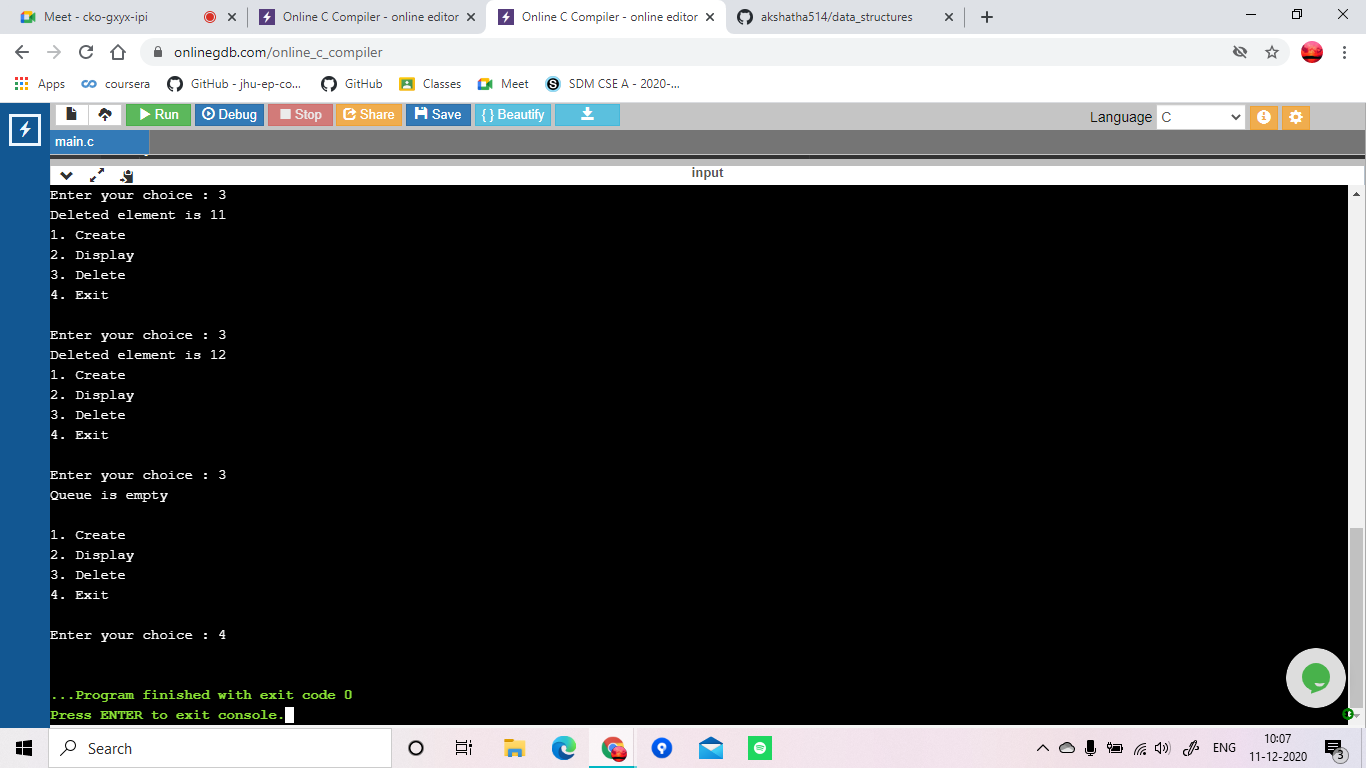




Queue program

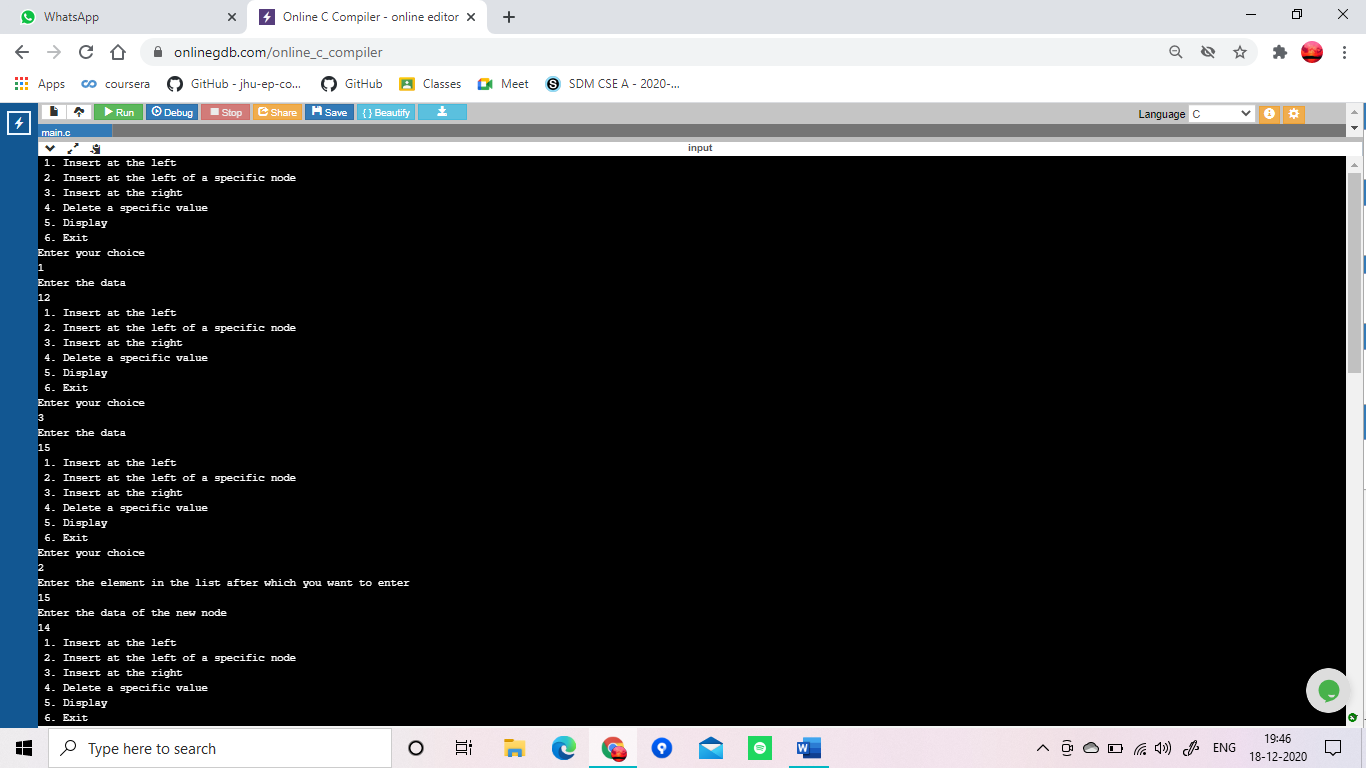


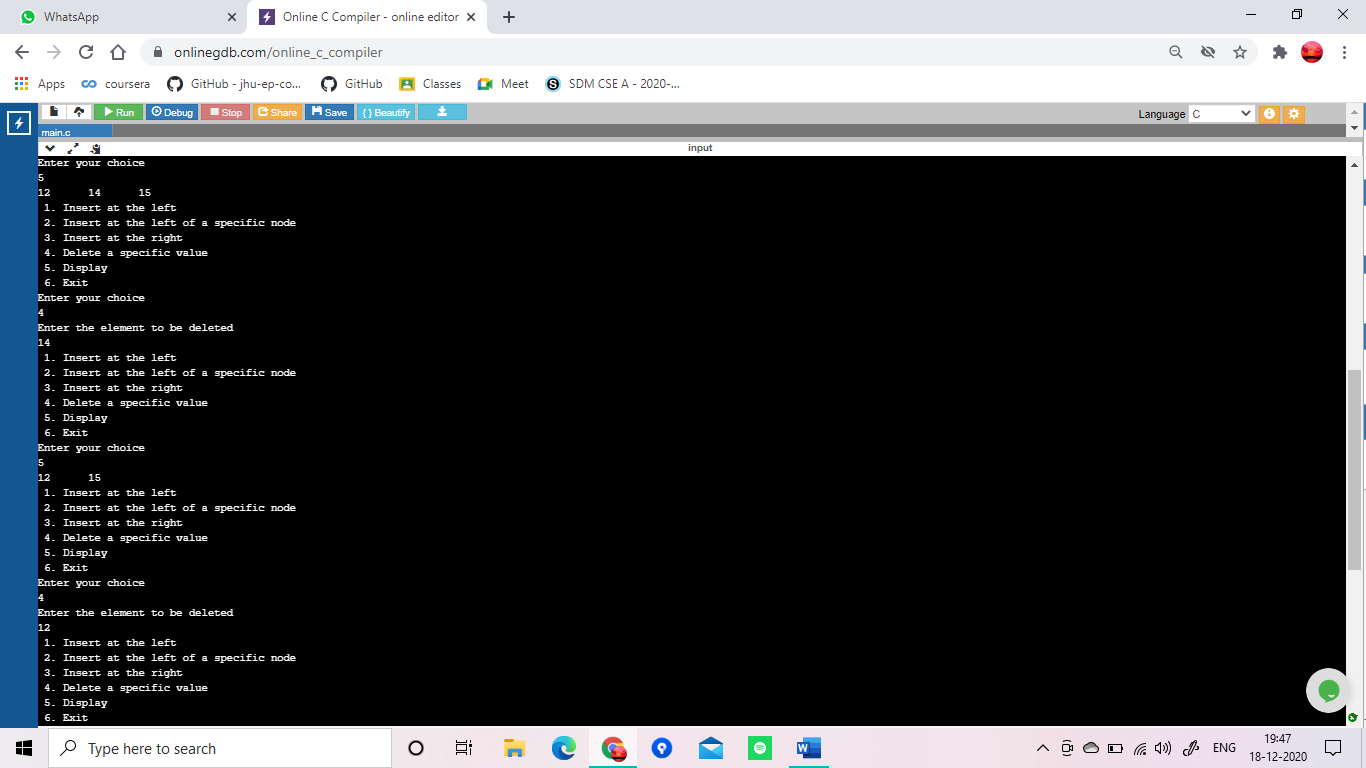


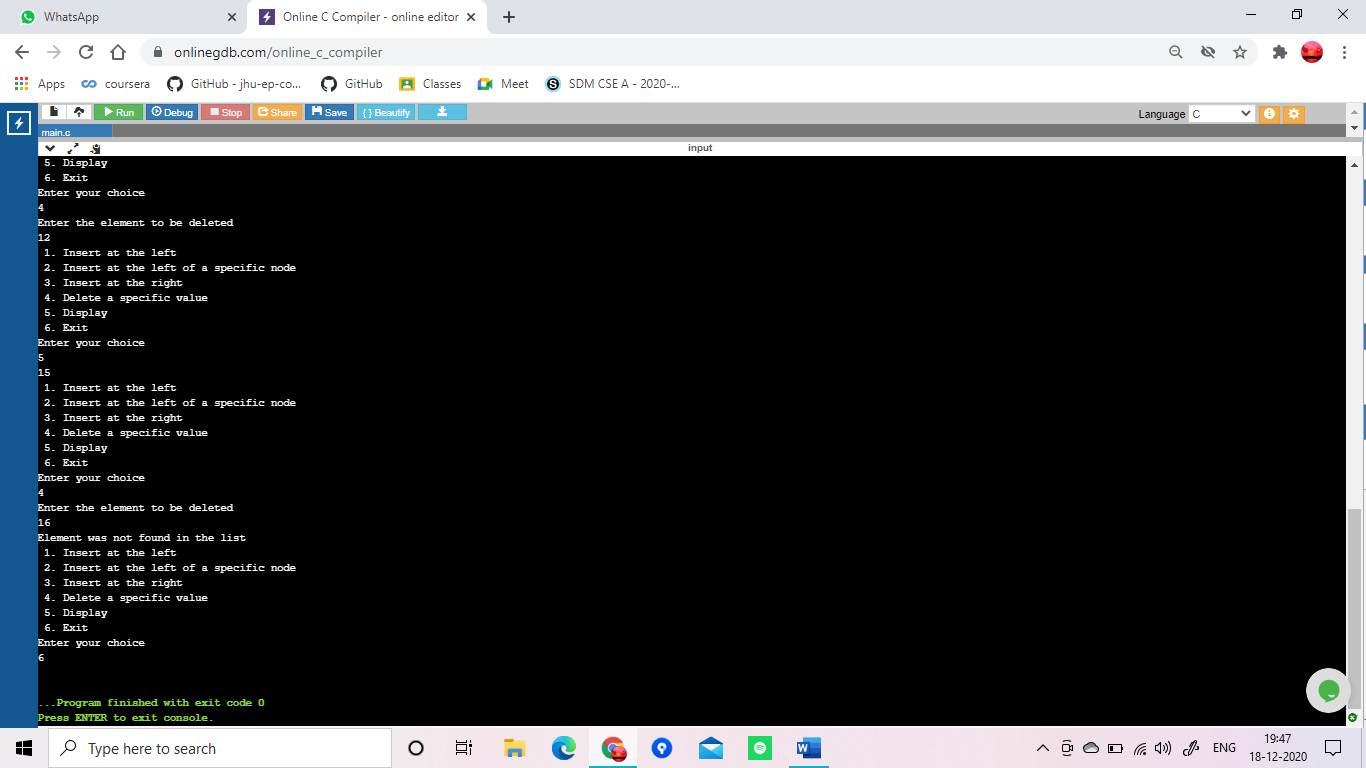


Program 9

|  |
| --- |
|  |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  |  |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | struct node \*prev; |
|  | }; |
|  | struct node \*head=NULL; |
|  |  |
|  | void insert\_left() |
|  | { |
|  | struct node \*new\_node; |
|  | new\_node=(struct node\*)malloc(sizeof(struct node)); |
|  | printf("Enter the data\n"); |
|  | scanf("%d",&new\_node->data); |
|  | new\_node->next=NULL; |
|  | new\_node->prev=NULL; |
|  |  |
|  | if(head==NULL) |
|  | { |
|  | head=new\_node; |
|  | } |
|  | else |
|  | { |
|  | new\_node->next=head; |
|  | head->prev=new\_node; |
|  | head=new\_node; |
|  | } |
|  |  |
|  | } |
|  | void insert\_right() |
|  | { |
|  | struct node \*new\_node,\*temp; |
|  | new\_node=(struct node\*)malloc(sizeof(struct node)); |
|  | printf("Enter the data\n"); |
|  | scanf("%d",&new\_node->data); |
|  | new\_node->next=NULL; |
|  | new\_node->prev=NULL; |
|  | if(head==NULL) |
|  | { |
|  | head=new\_node; |
|  | } |
|  | else |
|  | { |
|  | temp=head; |
|  | while(temp->next!=NULL) |
|  | temp=temp->next; |
|  | temp->next=new\_node; |
|  | new\_node->prev=temp; |
|  |  |
|  | } |
|  |  |
|  | } |
|  | void insert\_leftnode() |
|  | { |
|  | if(head==NULL) |
|  | { |
|  | printf("The list is empty\n"); return; |
|  | } |
|  | int ele; |
|  | struct node \*new\_node,\*temp; |
|  | printf("Enter the element in the list after which you want to enter\n"); |
|  | scanf("%d",&ele); |
|  | new\_node=(struct node\*)malloc(sizeof(struct node)); |
|  | printf("Enter the data of the new node\n"); |
|  | scanf("%d",&new\_node->data); |
|  | new\_node->next=NULL; |
|  | new\_node->prev=NULL; |
|  |  |
|  | temp=head; |
|  | if(temp->data==ele) |
|  | { |
|  | new\_node->next=head; |
|  | head->prev=new\_node; |
|  | head=new\_node; |
|  | } |
|  | else if(temp->next==NULL) |
|  | { |
|  | printf("Element was not found in the list\n"); |
|  | } |
|  | else |
|  | { |
|  | while(temp->next->data!=ele) |
|  | { |
|  | temp=temp->next; |
|  | if(temp==NULL) |
|  | { |
|  | printf("Element was not found in the list\n"); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | new\_node->next=temp->next; |
|  | temp->next=new\_node; |
|  | new\_node->prev=temp; |
|  | new\_node->next->prev=new\_node; |
|  | } |
|  |  |
|  |  |
|  | } |
|  | void delete() |
|  | { |
|  | struct node \*temp; |
|  | int ele; |
|  | if(head==NULL) |
|  | { |
|  | printf("The list is empty \n"); |
|  | return; |
|  | } |
|  | printf("Enter the element to be deleted\n"); |
|  | scanf("%d",&ele); |
|  | temp=head; |
|  | while(temp->data!=ele) |
|  | { |
|  | temp=temp->next; |
|  | if(temp==NULL) |
|  | { |
|  | printf("Element was not found in the list\n"); |
|  | return; |
|  | } |
|  | } |
|  | if(temp==head) |
|  | { |
|  | head=head->next; |
|  | } |
|  | else if(temp->next==NULL) |
|  | { |
|  | temp=temp->prev; |
|  | temp->next=NULL; |
|  | } |
|  |  |
|  | else |
|  | { |
|  | temp->prev->next=temp->next; |
|  | temp->next->prev=temp->prev; |
|  | free(temp); |
|  | } |
|  |  |
|  | } |
|  | void display() |
|  | { |
|  | if(head==NULL) |
|  | { |
|  | printf("List is empty \n"); |
|  | } |
|  | else |
|  | { |
|  | struct node \*temp; |
|  | temp=head; |
|  | while(temp!=NULL) |
|  | { |
|  | printf("%d\t",temp->data); |
|  | temp=temp->next; |
|  | } |
|  | printf("\n"); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | int main() |
|  | { |
|  | int choice; |
|  | do |
|  | { |
|  | printf(" 1. Insert at the left \n"); |
|  | printf(" 2. Insert at the left of a specific node \n"); |
|  | printf(" 3. Insert at the right \n"); |
|  | printf(" 4. Delete a specific value\n"); |
|  | printf(" 5. Display\n"); |
|  | printf(" 6. Exit\n"); |
|  | printf("Enter your choice\n"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: insert\_left(); break; |
|  | case 2: insert\_leftnode(); break; |
|  | case 3: insert\_right(); break; |
|  | case 4: delete(); break; |
|  | case 5: display(); break; |
|  | case 6: exit(0); |
|  | } |
|  | }while(choice!=6); |
|  | } |
|  |  |







Program 10

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* left;

struct node\* right;

}\*root1;

struct node \*create()

{

struct node \*temp;

printf("Enter the root node element: ");

temp=(struct node\*)malloc(sizeof(struct node));

scanf("%d",&temp->data);

temp->left=temp->right=NULL;

return temp;

}

void insert(struct node \*root,struct node \*temp)

{

if(temp->data<root->data)

{

if(root->left!=NULL)

insert(root->left,temp);

else

root->left=temp;

}

if(temp->data>root->data)

{

if(root->right!=NULL)

insert(root->right,temp);

else

root->right=temp;

}

}

void printPostorder(struct node\* node)

{

if (node == NULL)

return;

printPostorder(node->left);

printPostorder(node->right);

printf("%d\t", node->data);

}

void printInorder(struct node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

printf("%d\t", node->data);

printInorder(node->right);

}

void printPreorder(struct node\* node)

{

if (node == NULL)

return;

printf("%d\t", node->data);

printPreorder(node->left);

printPreorder(node->right);

}

int main()

{

int choice;

struct node\* temp;

do

{

printf("1. CREATE\n");

printf("2. INSERT\n");

printf("3. PREORDER TRAVERSAL\n");

printf("4. INORDER TRAVERSAL\n");

printf("5. POSTORDER TRAVERSAL\n");

printf("6. EXIT\n");

printf("Enter your choice : \n");

scanf("%d", &choice);

switch(choice)

{

case 1: root1 = create();

break;

case 2: printf("Enter the value you want to insert: ");

temp=(struct node\*)malloc(sizeof(struct node));

scanf("%d",&temp->data);

insert(root1, temp);

break;

case 3: printPreorder(root1);

break;

case 4: printInorder(root1);

break;

case 5: printPostorder(root1);

break;

case 6: exit(0);

break;

default: printf("Incorrect Choice\n");

}

}while(choice != 6);

return 0;

}

