|  |
| --- |
| **NAME- ARPIT SUMAN** |
| **USN - 1BM19CS026** |
| **DEPARTMENT - CSE** |
| **SUBJECT – DATA STRUCTURE LAB** |
| **ACADEMIC YEAR – 2020-21** |

**LAB – 1**

WAP for the below given scenario: A university wants to automate their admission process. Students are admitted based on the marks scored in a qualifying exam. A student is identified by student id, age and marks in qualifying exam. Data are valid, if: ● Age is greater than 20 ● Marks is between 0 and 100 (both inclusive) A student qualifies for admission, if ● Age and marks are valid and ● Marks is 65 or more Write a program to represent the students seeking admission in the university.

**PROGRAM**

#include<stdio.h>

struct student

{

int Id,age;

float marks;

}stud[100];

void main()

{

int i,n;

printf("Enter the total no of students\n");

scanf("%d",&n);

printf("enter student info as Id Age Marks\n");

for(i=0;i<n;i++)

{

scanf("%d %d %f",&stud[i].Id,&stud[i].age,&stud[i].marks);

}

printf("Qualified students are: ");

printf("\nId\t\tAGE\t\tMARKS\n");

for(i=0;i<n;i++)

{

if((stud[i].age>20) && (stud[i].marks>=65) && (stud[i].marks<=100))

printf("%d\t\t%d\t\t%f\t\t\n",stud[i].Id,stud[i].age,stud[i].marks);

}

printf("Not qualified students are: ");

printf("\nId\t\tAGE\t\tMARKS\n");

for(i=0;i<n;i++)

{

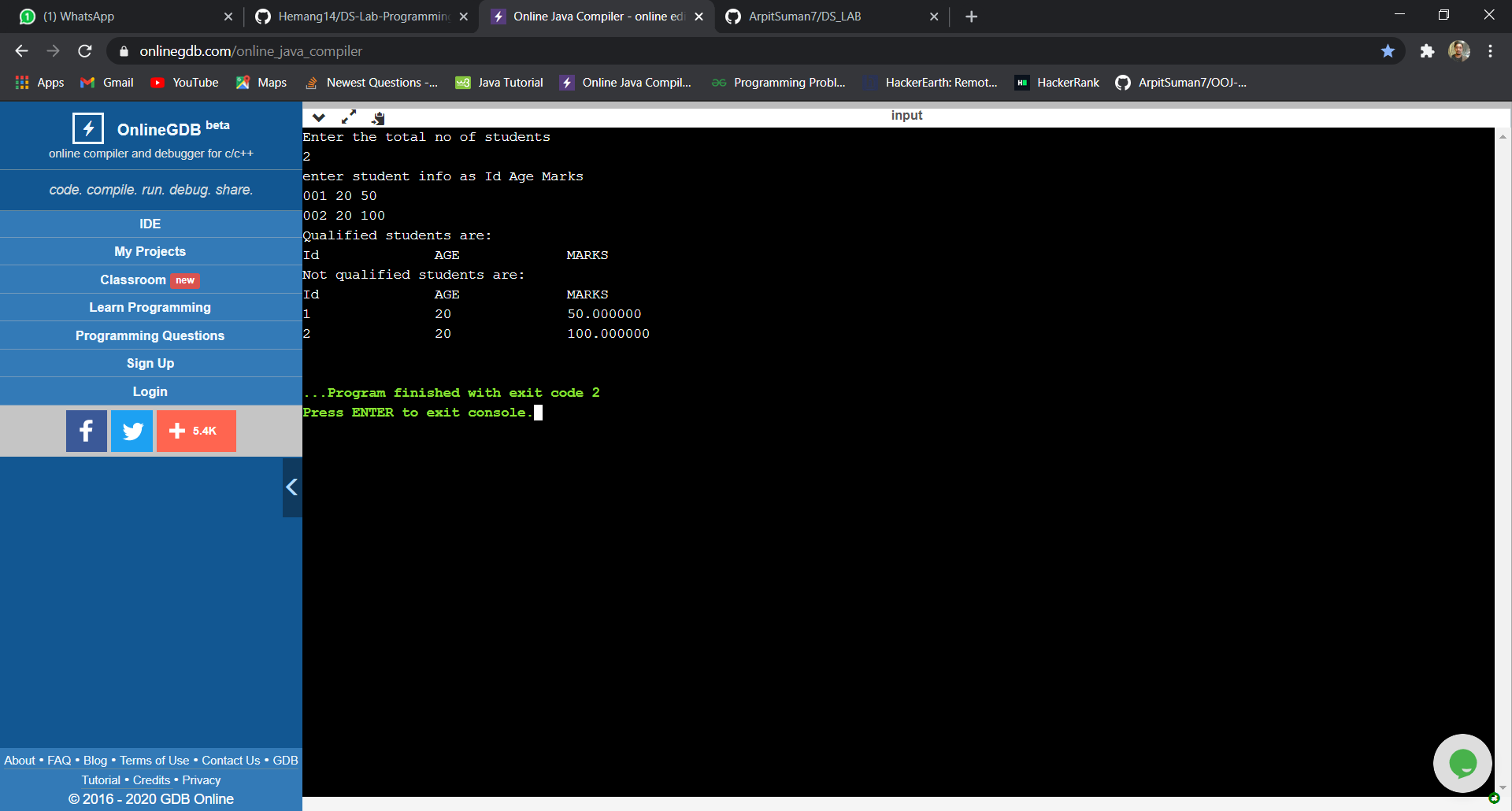
if((stud[i].age<=20) || (stud[i].marks<65))

printf("%d\t\t%d\t\t%f\t\t\n",stud[i].Id,stud[i].age,stud[i].marks);

}

}

**OUTPUT**



**LAB – 2**

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

**PROGRAM**

#include<stdio.h>

int stack[100],choice,n,top,x,i;

void push(void);

void pop(void);

void display(void);

int main()

{

top=-1;

printf("\n Enter the size of STACK[MAX=100]:");

scanf("%d",&n);

printf("\n\t STACK OPERATIONS USING ARRAY");

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

printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");

do

{

printf("\n Enter the Choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("\n\t EXIT POINT ");

break;

}

default:

{

printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");

}

}

}

while(choice!=4);

return 0;

}

void push()

{

if(top>=n-1)

{

printf("\n\tSTACK is over flow");

}

else

{

printf(" Enter a value to be pushed:");

scanf("%d",&x);

top++;

stack[top]=x;

}

}

void pop()

{

if(top<=-1)

{

printf("\n\t Stack is under flow");

}

else

{

printf("\n\t The popped elements is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\n The elements in STACK \n");

for(i=top; i>=0; i--)

printf("\n%d",stack[i]);

printf("\n Press Next Choice");

}

else

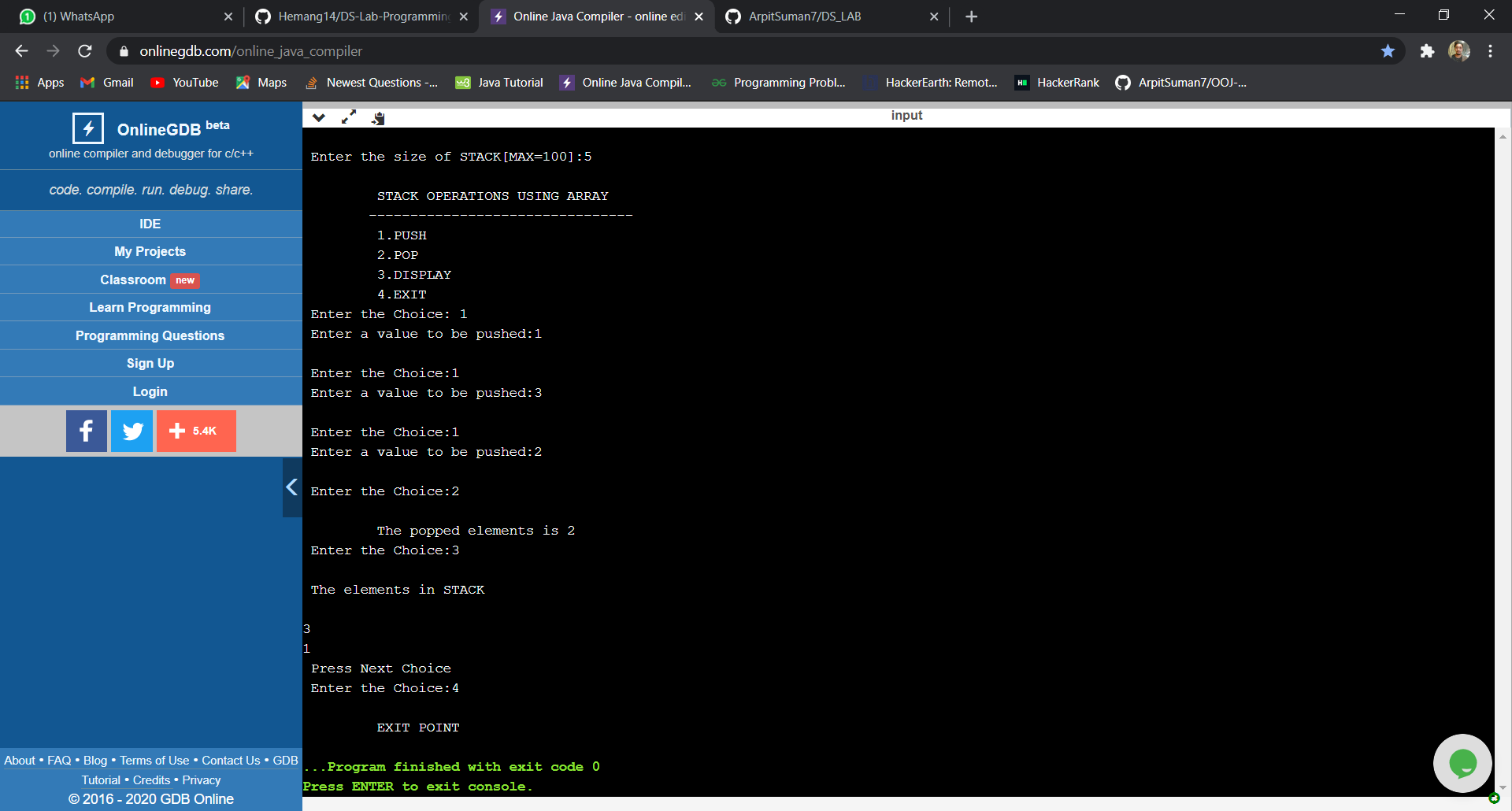
{

printf("\n The STACK is empty");

}

}

**OUTPUT**



**LAB - 3**

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)

**PROGRAM**

#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);

default : return (0);

}

}

int main(int argc, char \*\*argv)

{

char infix[100];

int i, item;

printf("Enter the infix expression :");

scanf("%s",infix);

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 '/':

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);

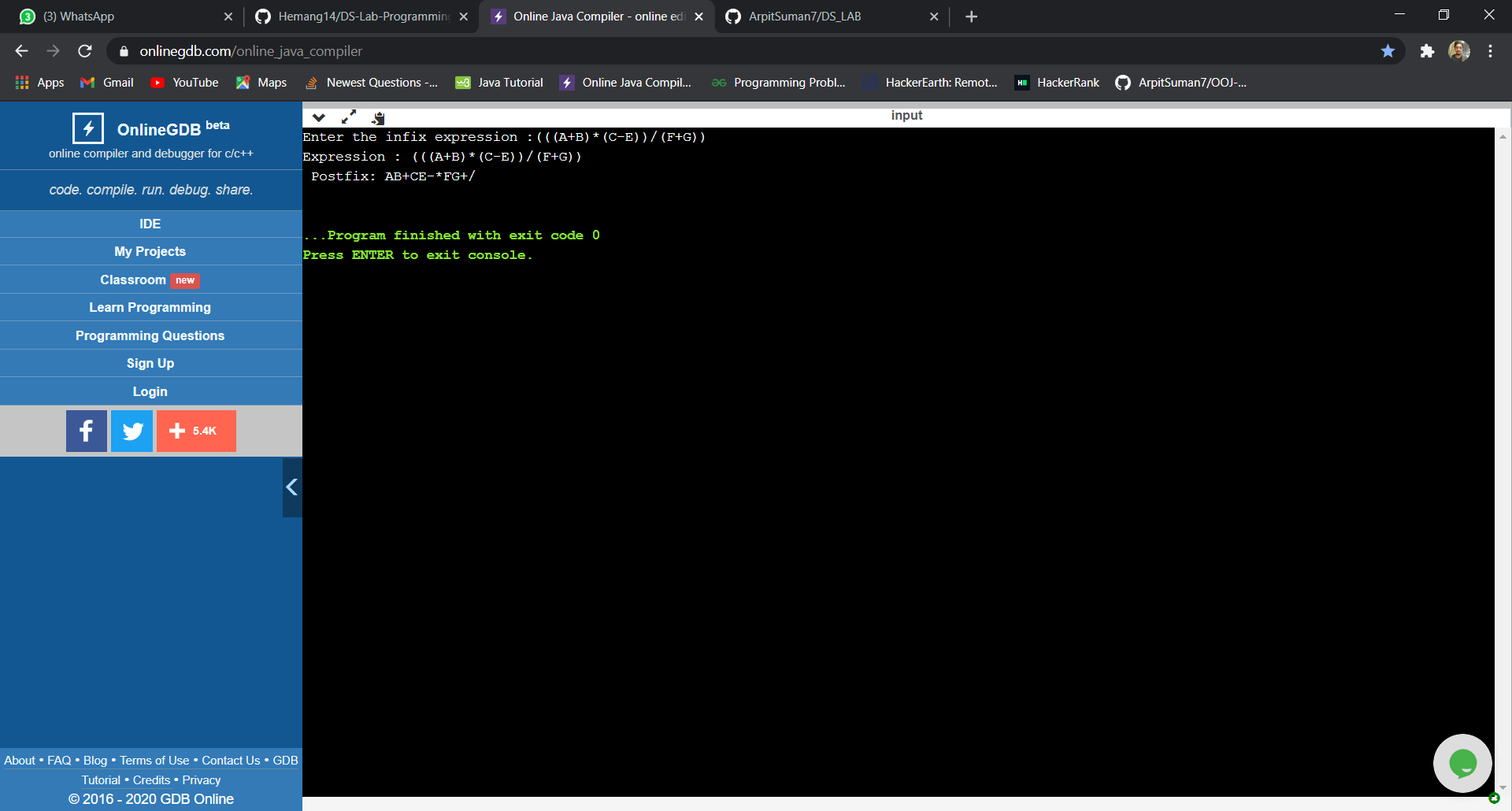
}

printf("\n");

return 0;

}

**OUTPUT**



**LAB - 4**

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

**PROGRAM**

#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 argc, char \*\*argv)

{

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

{

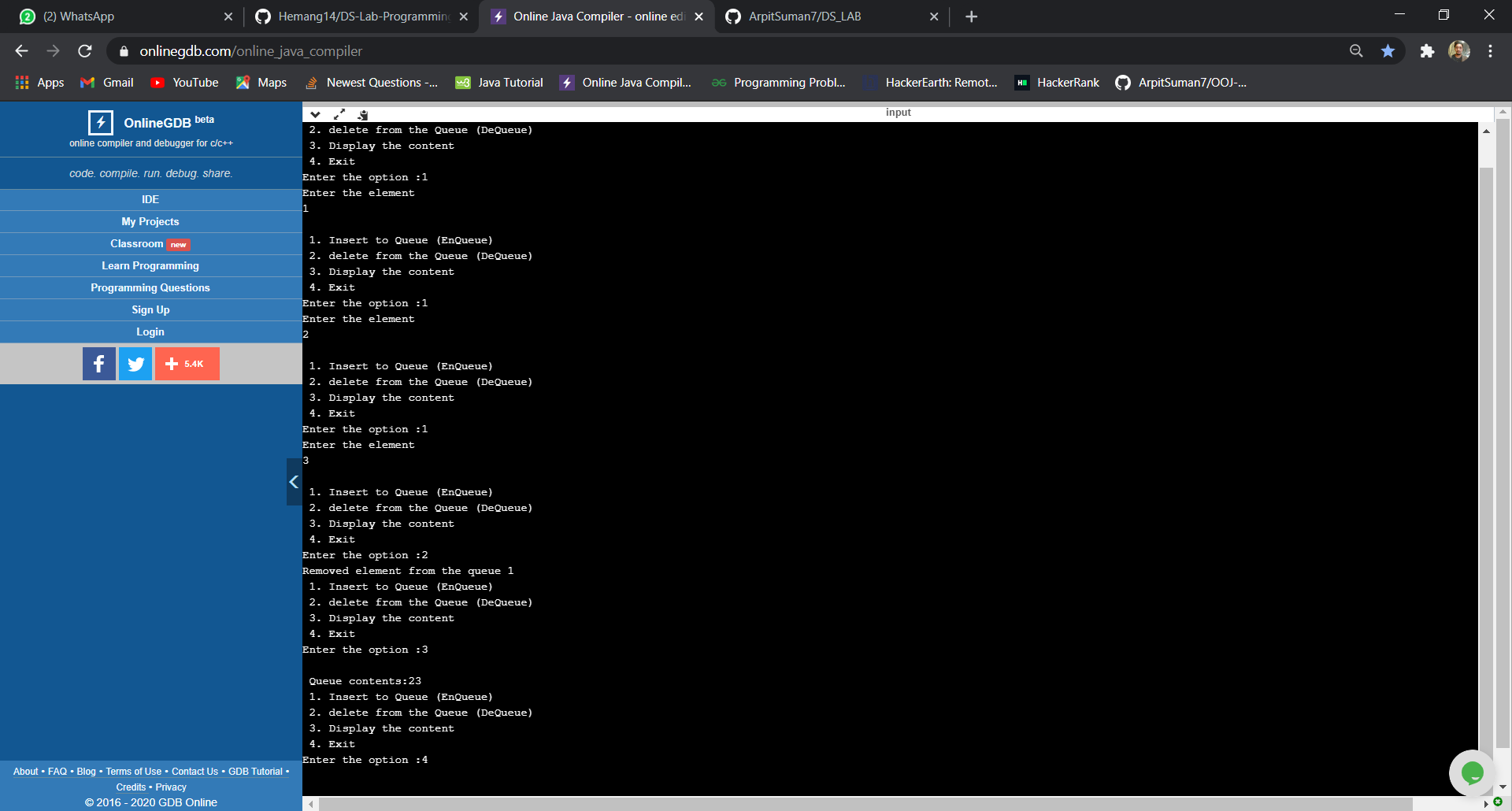
printf("\n Queue contents:");

for(i=front;i<=rear;i++)

printf("%d", queue[i]);

}

**OUTPUT**



**LAB - 5**

Write a program to demonstrate the operations of Circular Queue

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

#define MAX 3

int front=-1;

int rear=-1;

int queue[MAX];

void Enque(int);

int Deque();

void display();

int main(int argc, char \*\*argv)

{

int option;

int item;

do{

printf("Circular Queue\n");

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");

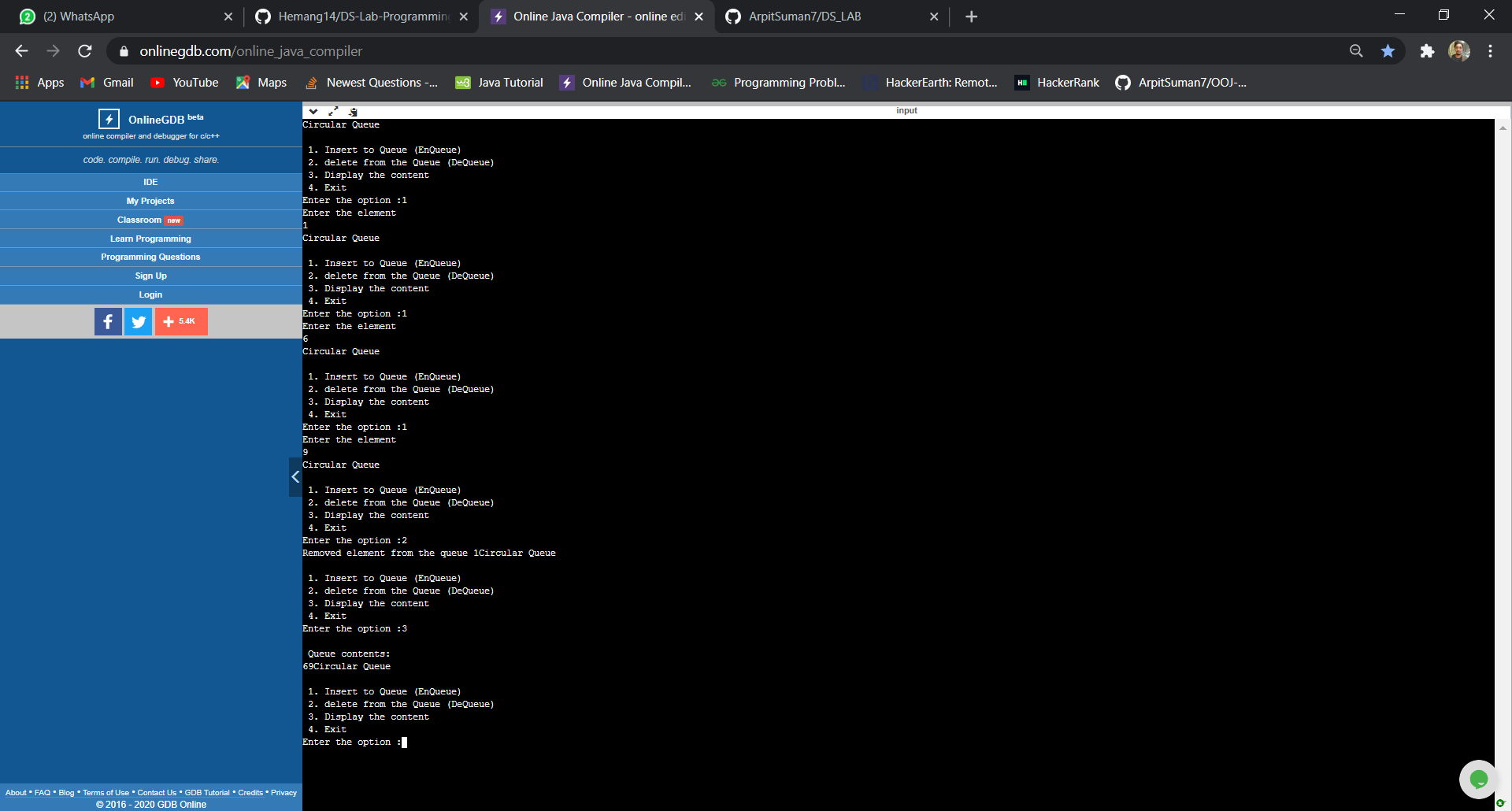
for(i=front;i<=rear;i++)

printf("%d", queue[i]);

}

}

**OUTPUT**



**LAB - 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.

**PROGRAM**

**Insertion Deletion Concatenation**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

};

void create(struct node \*\*);

void display(struct node \*);

void concat(struct node \*, struct node \*);

int main(int argc, char \*\*argv)

{

struct node \*head1=NULL, \*head2=NULL;

printf("Create two list\n");

printf("Creating List one\n");

create(&head1);

printf("Creating List two\n");

create(&head2);

concat(head1,head2);

display(head1);

}

void create(struct node \*\*hptr)

{

struct node \*newnode,\*temp;

int item;

int choice=1;

do

{

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

printf("Enter the data : ");

scanf("%d",&item);

newnode->data=item;

newnode->next=NULL;

printf("Do u want add element in the list:(if yes enter 1)\n");

scanf("%d", &choice);

if (\*hptr==NULL)

{

\*hptr=newnode;

}

else

{

temp=\*hptr;

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

}

}while (choice==1);

}

void concat (struct node \*temp1, struct node \*temp2)

{

while(temp1->next!=NULL)

temp1=temp1->next;

temp1->next=temp2;

}

void display(struct node \*ptr)

{

if(ptr==NULL)

{

printf("Nothing to print\n");

}

else

{

while(ptr!=NULL)

{

printf("%d ",ptr->data);

ptr=ptr->next;

}

}

printf("\n");

}

**Sort & Reverse**

#include <stdio.h>

#include <stdlib.h>

void create();

void display();

void sort();

void reverse();

struct node

{

int data;

struct node \*next;

};

struct node \*head=NULL;

int main()

{

int choice,ele;

do

{

printf("\n1. Create \n2. Display\n3. Sort\n4. Reverse\n5. Exit \n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: create(); break;

case 2: display();break;

case 3: sort();

break;

case 4: reverse();

break;

default: exit(0);

}

}while(choice==1 || choice==2 || choice==3 || choice==4);

return 0;

}

void create()

{

struct node \*newnode,\*temp;

int item;

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

printf("Enter the data : ");

scanf("%d",&item);

newnode->data=item;

if (head==NULL)

{

newnode->next=NULL;

head=newnode;

printf("Node created\n");

}

else

{

temp=head;//transversing

while(temp->next!=NULL)

{

temp=temp->next;

}

temp->next=newnode;

newnode->next=NULL;

printf("Node created\n");

}

}

void display()

{

struct node \*ptr=NULL;

ptr=head;

if(ptr==NULL)

{

printf("list empty!!!\n");

}

else

{

while(ptr!=NULL)

{

printf("%d ",ptr->data);

ptr=ptr->next;

}

}

printf("\n");

}

void sort()

{

struct node \*temp,\*ptr,\*ptr2;

int a;

temp=head;

ptr=head;

for(temp=head;temp!=NULL;temp=temp->next)

{

for(ptr=temp;ptr!=NULL;ptr=ptr->next)

{

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

{

a=temp->data;

temp->data=ptr->data;

ptr->data=a;

}

}

}

}

void reverse()

{

struct node \*prev=NULL,\*current=head, \*next=NULL;

while(current!=NULL)

{

next=current->next;

current->next=prev;

prev=current;

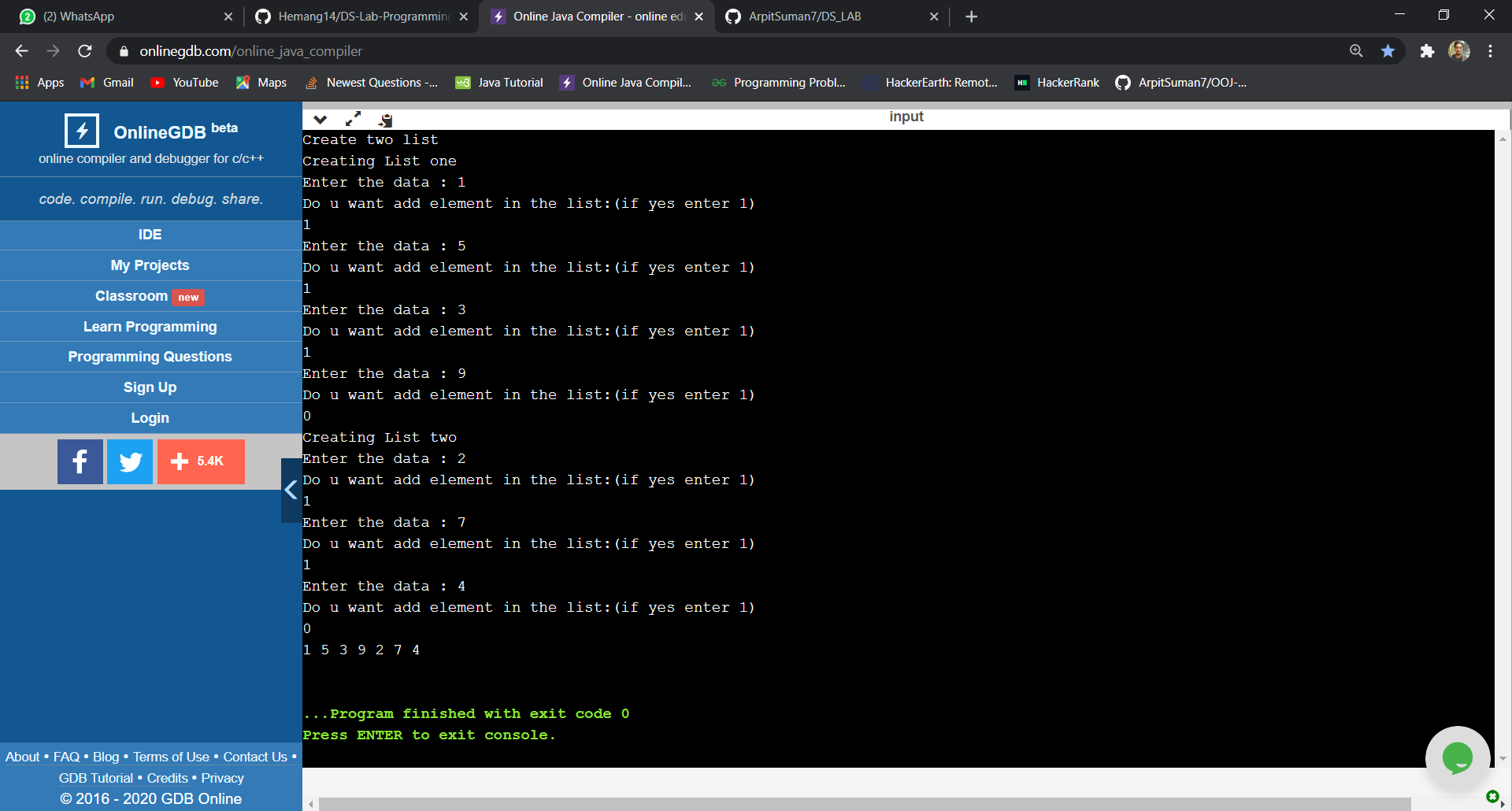
current=next;

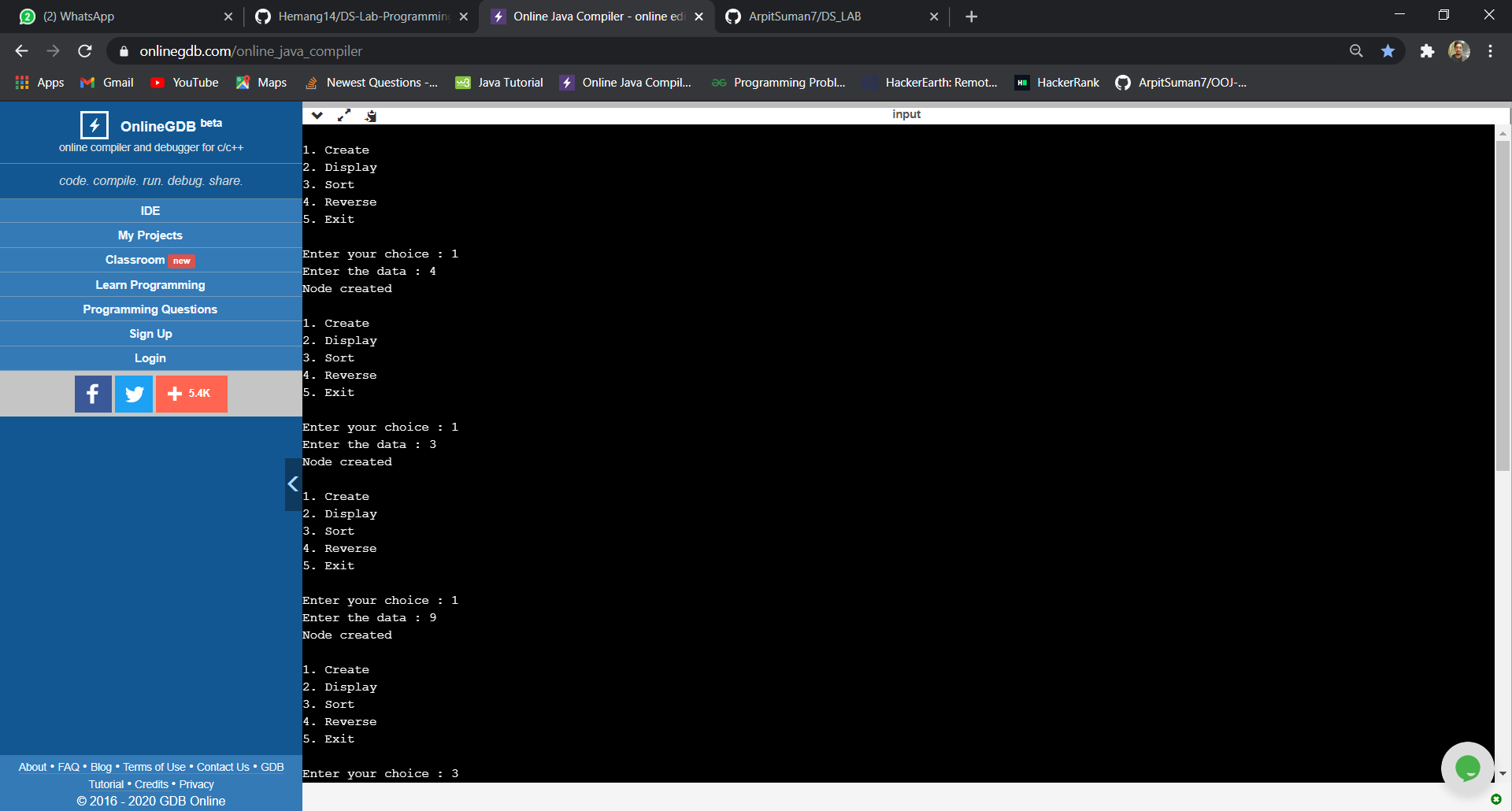
}

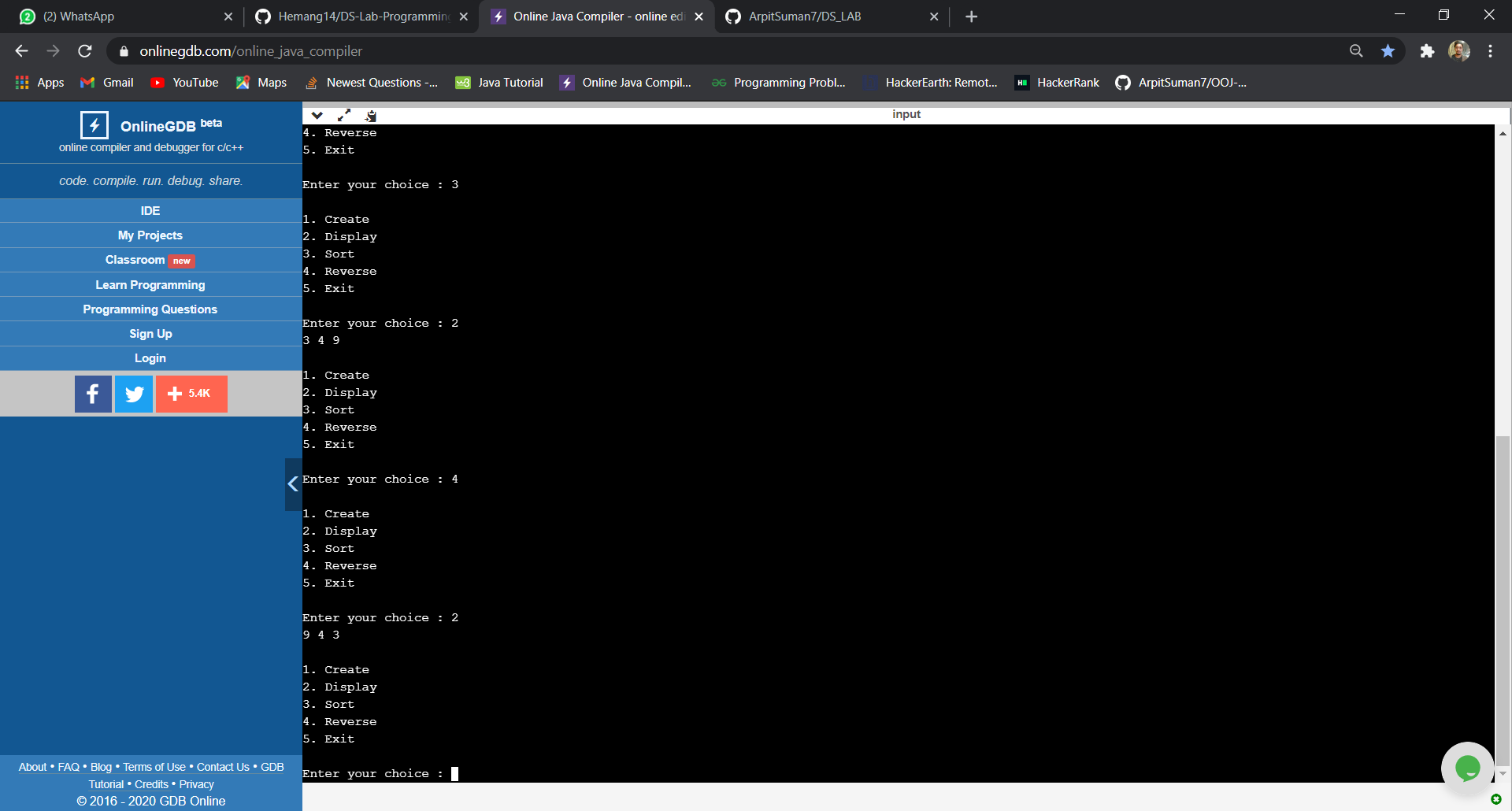
head=prev;

}

**OUTPUT**







**LAB - 7**

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) Deletion of first element, specified element and last element in the list. d) Display the contents of the linked list.

**LAB - 8**

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

**PROGRAM (7 & 8)**

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node\* next;

};

struct node \*rear=NULL, \*front =NULL, \*top=NULL;

struct node\* getnode(int item)

{

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

newn->data = item;

newn->next = NULL;

return newn;

}

void display(struct node\* head)

{

if(head == NULL)

{

printf("List is empty.\n");

return;

}

struct node\* ptr = head;

while(ptr)

{

printf("%d->", ptr->data);

ptr = ptr->next;

}

printf("\b \b\b \n");

}

struct node\* insertfront(struct node\* head, int item)

{

struct node\* newn = getnode(item);

newn->next = head;

head = newn;

return head;

}

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

struct node\* sort (struct node\* head)

{

int sorted;

if(head == NULL) return head;

struct node\* ptr = head;

do

{

ptr = head;

sorted = 0;

while(ptr->next)

{

if(ptr->data > ptr->next->data)

{

swap(&ptr->data, &ptr->next->data);

sorted = 1;

}

ptr = ptr->next;

}

} while(sorted == 1);

return head;

}

void reverse(struct node\*\* head)

{

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;

}

struct node\* concatenate(struct node\* head1, struct node\* head2)

{

struct node\* ptr = head1;

while(ptr->next)

{

ptr = ptr->next;

}

ptr->next = head2;

return head1;

}

void qinsert()

{

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 qdel()

{

if(front==NULL)

{

printf("Queue is empty\n");return;

}

else

{

printf("Deleted ele is %d",front->data);

if(front==rear)

{

printf("Queue is empty\n");

front=NULL; rear=NULL;

}

else

front=front->next;

}

}

void qdisplay()

{

struct node \*temp;

if(front ==NULL)

{

printf("Queue is empty");

return;

}

temp=front;

while (temp !=NULL)

{

printf("%d ",temp->data);

temp=temp->next;

}

}

void spush()

{

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 spop()

{

if(top==NULL)

printf("stack is empty");

else

{

printf("element removed is %d:", top->data);

top=top->next;

}

}

void sdisplay()

{

struct node \*temp;

temp=top;

if(top==NULL)

printf("Stack is empty");

while(temp!=NULL)

{

printf("%d",temp->data);

printf("\n");

temp=temp->next;

}

}

int main()

{

printf("Linked list program containing sort, reverse, and concatenate functions.\n");

int n1, n2, n, ch, flag = 0;

int choice;

struct node\* head1 = NULL; struct node\* head2 = NULL;

do

{

printf("Enter the choice\n1.Stack\n2.Queue\n3: Linked list 1\n4: Linked list 2\n5: Exit\n");

scanf("%d", &n1);

switch(n1)

{

case 1:

{

do

{ printf("\n1. Push \n2. Display \n3. Pop\n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1: spush(); break;

case 2: sdisplay();break;

case 3: spop(); break;

;

}

}while(choice!=10);

}

case 2:

{

do

{ printf("\nQueue implementation using linked list\n");

printf("\n1. Create \n2. Display \n3. Delete \n4. Exit \n");

printf("\nEnter your choice : ");

scanf("%d",&choice);

switch(choice)

{ case 1: qinsert(); break;

case 2: qdisplay();break;

case 3: qdel(); break;

}

}while(choice!=10);

}

case 3:

{

do

{

printf("3: Insert\n4: Sort\n5: Reverse\n6: Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9: Exit\n");

scanf("%d", &n2);

switch(n2)

{

case 3: {

printf("Enter item to be inserted: ");

scanf("%d", &n);

head1 = insertfront(head1, n);

break;

}

case 4: {

head1 = sort(head1);

break;

}

case 5: {

reverse(&head1);

break;

}

case 6: {

head1 = concatenate(head1, head2);

break;

}

case 7: {

display(head1);

break;

}

case 8: {

flag = 1;

break;

}

case 9: {

exit(0);

}

default: printf("Invalid input.\n");

}

if(flag == 1)

{

break;

}

}while(1);

break;

}

case 4: {

flag = 0;

do

{

printf("3: Insert\n4: Sort\n5: Reverse\n6: Concatenate with list 1\n7: Display list\n8: Go back to main menu\n9: Exit\n");

scanf("%d", &n2);

switch(n2)

{

case 3: {

printf("Enter item to be inserted: ");

scanf("%d", &n);

head2 = insertfront(head2, n);

break;

}

case 4: {

head2 = sort(head2);

break;

}

case 5: {

reverse(&head2);

break;

}

case 6: {

head2 = concatenate(head2, head1);

break;

}

case 7: {

display(head2);

break;

}

case 8: {

flag = 1;

break;

}

case 9: {

exit(0);

}

default: printf("Invalid input.\n");

}

if(flag == 1)

{

flag = 0; break;

}

}while(1);

break;

}

case 9: exit(0);

default: printf("Invalid input.\n");

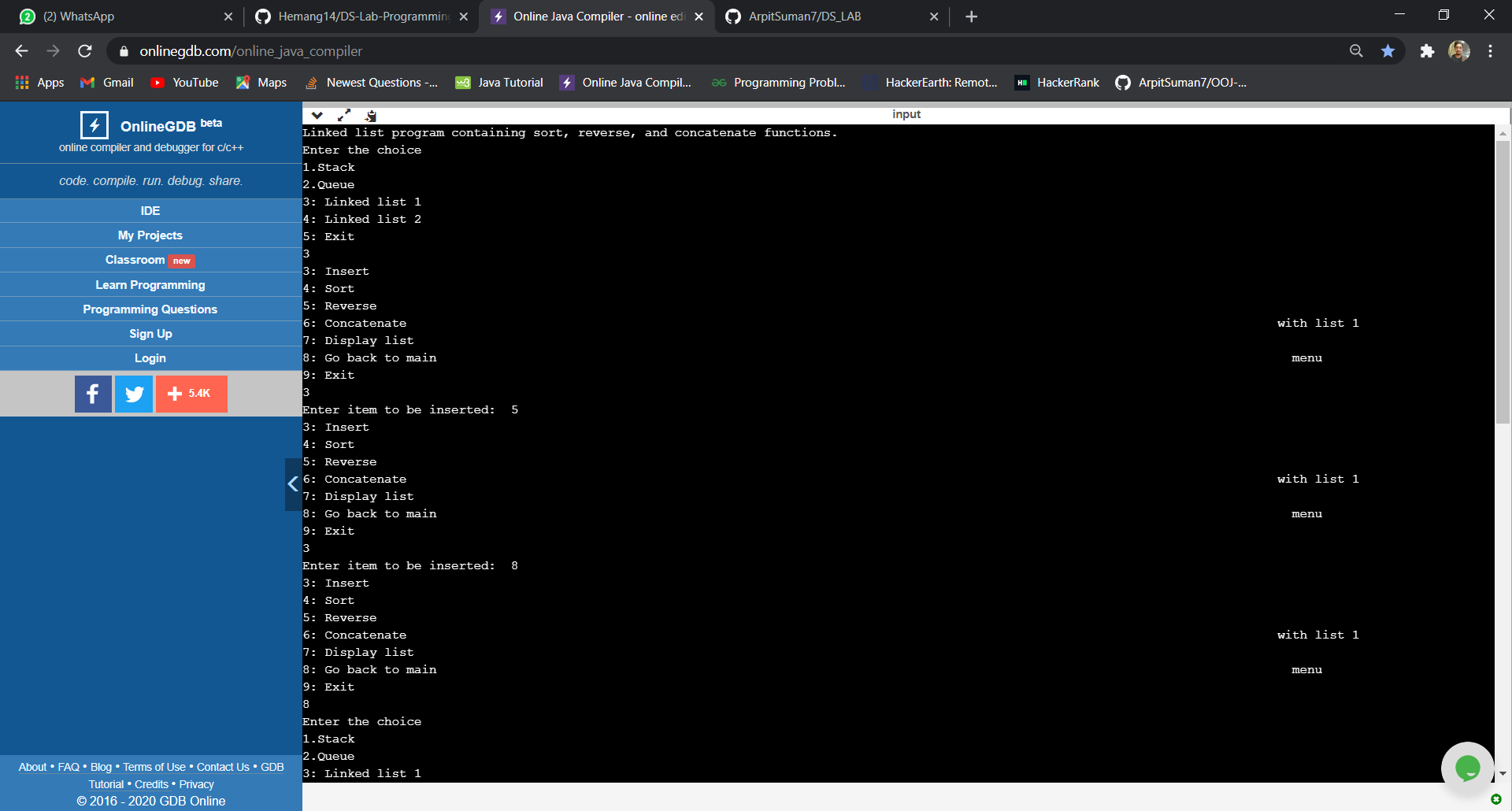
}

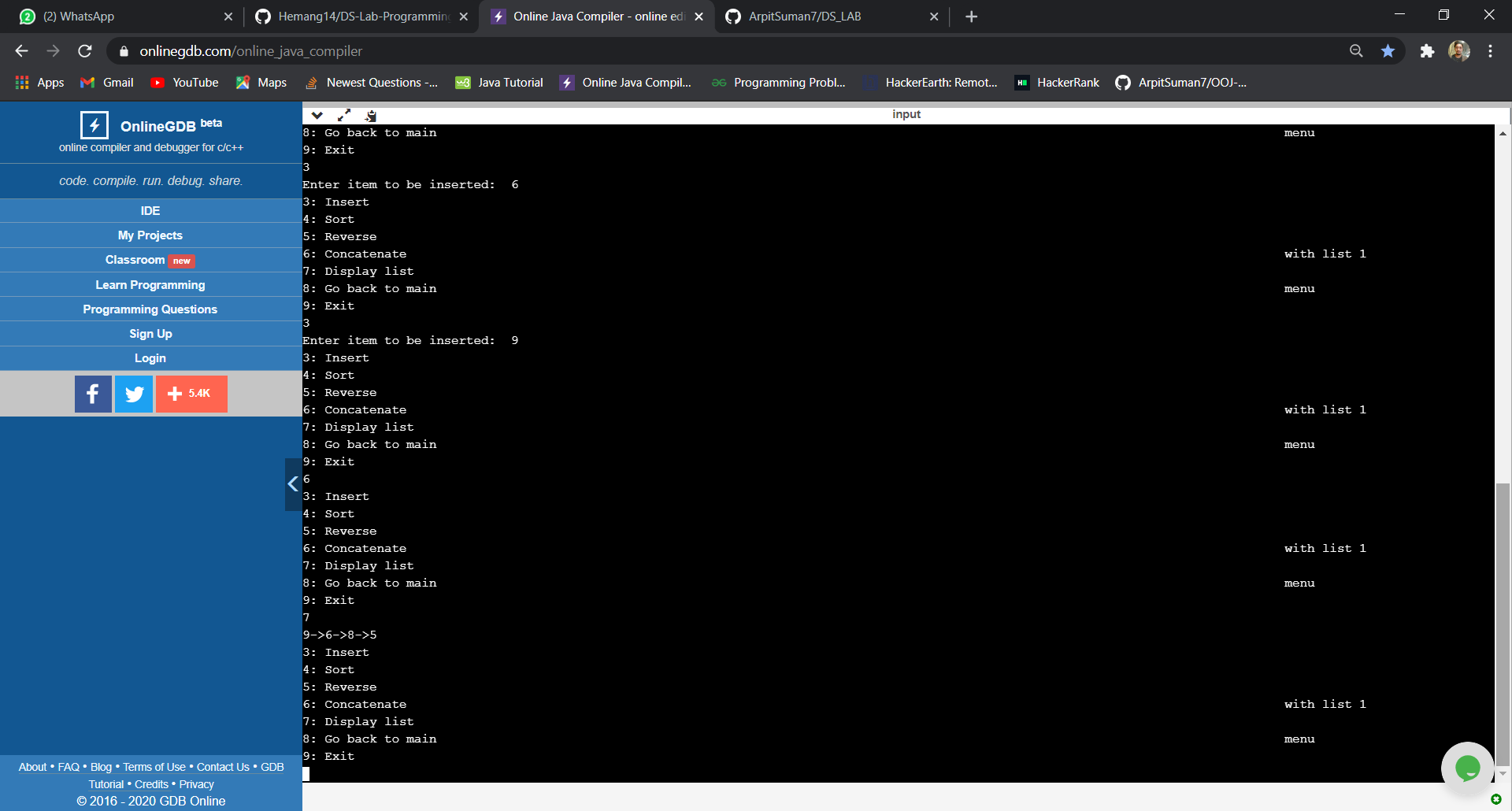
}while(1);

return 0;

}

**OUTPUT**





**LAB - 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 d) Display the contents of the list

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

void insert\_left();

void del();

void display();

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*head=NULL;

int main()

{

int choice;

while(choice!=4)

{

printf(" 1. Insert left \n");

printf(" 2. Delete \n");

printf(" 3. Display\n");

printf(" 4. Exit\n");

printf("Enter your choice\n");

scanf("%d",&choice);

if(choice==1)

insert\_left();

else if(choice==2)

del();

else if(choice==3)

display();

else if(choice==4)

break;

}

return 0;

}

void insert\_left()

{

struct node \*new\_node;

new\_node=(struct node\*)malloc(sizeof(struct node));

printf("Enter the item:");

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 del()

{

struct node \*temp;

int ele;

if(head==NULL)

{

printf("Empty List \n");

return;

}

printf("Enter the element to be deleted:");

scanf("%d",&ele);

temp=head;

while(temp->data!=ele)

{

temp=temp->next;

if(temp==NULL)

{

printf("Element is not in the list\n");

break;

}

}

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;

}

}

void display()

{

struct node \*temp;

temp=head;

while(temp!=NULL)

{

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

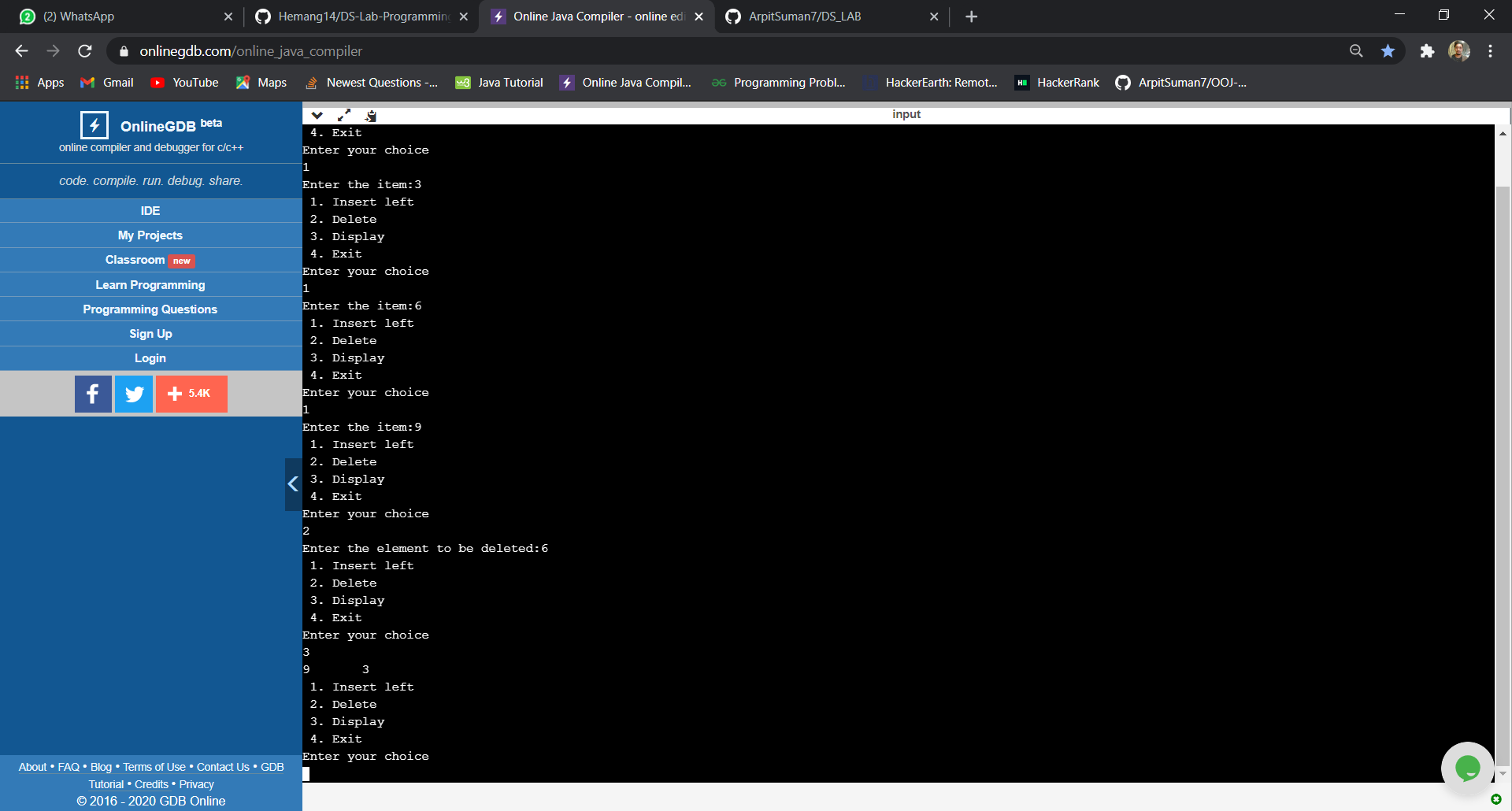
temp=temp->next;

}

printf("\n");

}

**OUTPUT**



**LAB - 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.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node \*left, \*right;

} node;

node \*create(int data) {

node \*temp;

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

temp->data = data;

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

return temp;

}

void inorder(node \*root) {

if (root != NULL) {

inorder(root->left);

printf("%d ", root->data);

inorder(root->right);

}

}

void preorder(node \*root) {

if (root != NULL) {

printf("%d ", root->data);

preorder(root->left);

preorder(root->right);

}

}

void postorder(node \*root) {

if (root != NULL) {

postorder(root->left);

postorder(root->right);

printf("%d ", root->data);

}

}

void insert(node \*root, 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;

}

}

int main(void) {

node \*root = NULL,\*temp;

int choice = 0;

while(choice != 2)

{

temp =

printf("1 - Insert\n");

printf("2 - Exit\n");

printf("Enter your choice:");

scanf("%d",&choice);

if(choice==1)

{

int val;

printf("Enter value:");

scanf("%d",&val);

temp = create(val);

if(root==NULL)

root=temp;

else

insert(root,temp);

}

else if(choice==2)

break;

else

printf("Invalid choice\n");

}

printf("Inorder traversal: ");

inorder(root);

printf("\nPreorder traversal: ");

preorder(root);

printf("\nPostorder traversal: ");

postorder(root);

}

**OUTPUT**

