**Sri Sri University, Cuttack, Odisha.**

**Faculty of Science**

|  |  |
| --- | --- |
| **Program: B.Sc. – Computer Science, Data Science, & Environmental Science**  **(2020-23 Batch)**  **Subject Code/Subject Name: Data Structure Laboratory**  **Assignment –IX** | |
| **Full Name of the Student:** | VINAYAK SANJAY CHAVAN |
| **Full Roll Number:** | BCS-011 |
| **Program:** | B.Sc. (Computer Sc.) / B.Sc. (Data Sc.) / B.Sc. (Env. Sc.) |
| **Date:** | 10th – 11th April, 2021 (10.00 AM – 12.00 Noon) |
| **Signature** |  |

|  |  |
| --- | --- |
| **All Questions are compulsory** | **Total Marks: 80** |

|  |  |
| --- | --- |
| **Question (s)** | **Maximum Marks** |
| 1. Write a program to convert an infix expression into its corresponding prefix expression using a Stack.   [i/p – a/b-(c+d)-e would give an output of  --/ab+cde]  **Your code:header file:**  **#include <iostream>**  **using namespace std;**  **#define size 100**  **typedef struct stack{**  **char array[size];**  **int top;**  **}STACK;**  **void stack\_initialisation(STACK\*);**  **void push(STACK\*,char);**  **char pop(STACK\*);**  **char peek(STACK\*);**  **bool full\_stack(STACK\*);**  **int empty\_stack(STACK\*);**  **void display(STACK\*);**  **void stack\_initialisation(stack \*s1){**  **s1->top=-1;**  **}**  **void push(stack \*s1,char element ){**  **s1->top++;**  **s1->array[s1->top]=element;**  **}**  **char pop(stack \*s1){**  **char element;**  **element=s1->array[s1->top];**  **s1->top--;**  **return element;**  **}**  **char peek(stack \*s1){**  **return s1->array[s1->top];**  **}**  **bool full\_stack(stack \*s1){**  **if(s1->top==size-1){**  **cout<<"yes stack is full "<<endl;**  **return true;**  **}**  **else{**  **cout<<"stack has space "<<endl;**  **return false;**  **}**  **}**  **int empty\_stack(stack \*s1){**  **if(s1->top==-1){**  **return 1;**  **}**  **else{**  **return 0;**  **}**  **}**  **void display(stack \*s1) {**  **if(s1->top >-1) {**  **cout<<"Stack elements are:";**  **for(int i=s1->top; i>=0; i--)**  **cout<<s1->array[i]<<" ";**  **cout<<endl;**  **}else**  **cout<<"Stack is empty";**  **}**  **Code:**  **#include <iostream>**  **#include <stdio.h>**  **#include <string.h>**  **#include "Mystack.h"**  **using namespace std;**  **string reverse(string);**  **string infix\_prefix(string);**  **int prece\_table(char);**  **int main(){**  **string s,res,ori\_out;**  **cout<<"Enter the string: ";**  **cin>>s;**  **res=infix\_prefix(s);**  **ori\_out=reverse(res);**  **cout<<ori\_out<<endl;**  **}**  **int prece\_table(char op){**  **if(op=='^'){**  **return 3;// ^ is the highest precedence among this three operator**  **}else if(op=='\*' || op=='/'){**  **return 2;**  **}else if(op=='+' || op=='-'){**  **return 1;//as + and - are lowest precedence among this three operator**  **}else{**  **return 0;**  **}**  **}**  **string reverse(string s2){**  **STACK A;**  **int j;**  **char b;**  **string res;**  **stack\_initialisation(&A);//changed**  **for(int i=0;i<s2.length();i++){**  **push(&A,s2[i]);**  **}**  **while(j<s2.length()){**  **b=pop(&A);**  **res+=b;**  **j++;**  **}**  **return res;**  **}**  **string infix\_prefix(string s1){**  **STACK out;**  **string output,output1;**  **stack\_initialisation(&out);**  **for(int i=s1.length()-1;i>=0;i--){ //Changed**  **if((s1[i]>='A' && s1[i]<='Z') || (s1[i]>='a' && s1[i]<='z') || (s1[i]>='0' && s1[i]<='9') ){**  **output+=s1[i];**  **//operand check**  **}else if(s1[i]==')'){**  **push(&out,s1[i]);**  **}else if(s1[i]=='('){**  **//changed**  **while((empty\_stack(&out)!=true) && peek(&out)!=')'){**  **output+=pop(&out);**  **}**  **pop(&out);//Changed**  **}else if (s1[i]== '^'||s1[i]== '\*' || s1[i]== '/'||s1[i]== '+'||s1[i]== '-') {**    **/\***  **if(empty\_stack(&out)!=true && (prece\_table(s1[i])>=prece\_table(peek(&out)))){**  **push(&out,s1[i]);**  **}else if(empty\_stack(&out)!=true && (prece\_table(peek(&out))>=prece\_table(s1[i]))){**  **char ch=pop(&out);**  **output+=ch;**  **} else {**  **push(&out,s1[i]);**  **}**  **\*/**    **while(!empty\_stack(&out)){**  **if(prece\_table(peek(&out))> prece\_table(s1[i])){**  **char ch=pop(&out);**  **output+=ch;**  **} else{**  **break;**  **}**  **}**  **push(&out,s1[i]);**  **}**  **}**    **while(!empty\_stack(&out)){**  **char ch=pop(&out);**  **output+=ch;**  **}**  **return output;**  **}**  **Screenshot of output:** | 20 |
| 1. Write program to evaluate a prefix expression using Stack.   **Your code: code:**  **/\* Write a program to convert an infix expression into its**  **corresponding prefix expression using a Stack. \*/**  **#include <iostream>**  **#include <cstring>**  **#include "prefix\_stack.h"**  **#include <cstdlib>**  **using namespace std;**  **int infix\_to\_prefix(char\*);**  **bool isOperand(int c)**  **{**    **return isdigit(c);**  **}**  **int infix\_to\_prefix(char\* exp){**  **STACK s1;**  **for(int i = exp[i]-1;i>=0; i--){**  **if(exp[i] == ' '){**  **continue;**  **}else if (isOperand(exp[i])){**  **push(&s1,exp[i] - '0');**    **}else{**  **int val1=pop(&s1);**  **int val2=pop(&s1);**    **switch(exp[i]){**    **case '+':{**  **push(&s1, val1 + val2);**  **break;**  **}**  **case '-':{**  **push(&s1, val1 - val2);**  **break;**  **}**  **case '\*':{**  **push(&s1, val1 \* val2);**  **break;**  **}**  **case '/':{**  **push(&s1, val1 / val2);**  **break;**  **}**    **}**  **}**  **}**  **return peek(&s1); //we need to retrun top element**  **}**  **int main(){**  **char exp[100];**  **cout<<"enter the expression: ";**  **cin>>exp;**  **cout<<"prefix evaluation: "<<infix\_to\_prefix(exp);**  **return 0;**  **}**  **Header file:**  **#include <iostream>**  **using namespace std;**  **#define size 20**  **typedef struct stack{**  **char array[size];**  **int top;**  **}STACK;**  **int push(STACK\*,int);**  **int pop(STACK\*);**  **int peek(STACK\*);**  **bool full\_stack(STACK\*);**  **int empty\_stack(STACK\*);**  **void display(STACK\*);**  **void stack\_initialisation(stack \*str1){**  **str1->top=-1;**  **}**  **int push(stack \*str1,int element ){**  **str1->top++;**    **str1->array[str1->top]=element;**  **return element;**  **}**  **int pop(stack \*str1){**  **int element;**  **element=str1->array[str1->top];**  **str1->top--;**  **return element;**  **}**  **int peek(stack \*str1){**  **return str1->array[str1->top];**  **}**  **bool full\_stack(stack \*str1){**  **if(str1->top==size-1){**  **cout<<"yes stack is full "<<endl;**  **return true;**  **}**  **else{**  **cout<<"stack has space "<<endl;**  **return false;**  **}**  **}**  **int empty\_stack(stack \*str1){**  **if(str1->top==-1){**  **return 1;**  **}**  **else{**  **return 0;**  **}**  **}**  **void display(stack \*str1) {**  **if(str1->top >-1) {**  **cout<<"Stack elements are:";**  **for(int i=str1->top; i>=0; i--)**  **cout<<str1->array[i]<<" ";**  **cout<<endl;**  **}else**  **cout<<"Stack is empty";**  **}**  **Screenshot of output:** | 20 |
| 1. Defining structure of a input restricted double ended queue (DEQUEUE) , write functions for inserting, deleting from front (beginning), deleting from rear (end), in the queue. Write functions IsFull() and IsEmpty() also. Write main function to call them.   **Your code:**  **Header file:**  **#include<iostream>**  **#define max 5**  **using namespace std;**  **typedef struct dequeue{**  **int front;**  **int rear;**  **int arr[max];**  **}DEQUEUE;**  **void intia\_dequeue(DEQUEUE\*);**  **void insert\_dequeue\_rear(DEQUEUE\*);**  **void insert\_dequeue\_front(DEQUEUE\*);**  **int del\_dequeue\_front(DEQUEUE\*);**  **int del\_dequeue\_rear(DEQUEUE\*);**  **bool is\_full\_dequeue(DEQUEUE\*);**  **bool is\_empty\_dequeue(DEQUEUE\*);**  **void display\_dequeue(DEQUEUE\*);**  **int peek\_dequeue(DEQUEUE\*);**  **void intia\_dequeue(DEQUEUE \*dq){**  **dq->front=0;**  **dq->rear=-1;**  **}**  **void insert\_dequeue\_rear(DEQUEUE \*dq,int x){**  **if(dq->rear==max-1){//to check that queue is full or not**  **cout<<"The queue is full";**  **}else{// if queue is not full then increment the rear value and**  **//set the new rear value with the element**  **dq->rear++;**  **dq->arr[dq->rear]=x;**  **}**  **}**  **void insert\_dequeue\_front(DEQUEUE \*dq,int x){// to insert the value with front value**  **if(dq->front==0){**  **cout<<"We cant insert any item from front";**    **}else{**  **dq->front--; //changed**  **dq->arr[dq->front]=x;//now update the front value with the given element**  **//and decrement the front value**  **}**  **}**  **int del\_dequeue\_front(DEQUEUE \*dq){**  **int x=-1;**  **if((dq->front==0) && (dq->rear==-1)){// to check the queue is empty or not**  **x=-99;**  **}else{**  **x=dq->arr[dq->front];//if not empty than delete the front item and increment the front value**  **dq->front++;**  **}**  **return x;**  **}**  **int del\_dequeue\_rear(DEQUEUE \*dq){**  **int n;**  **if (dq->front==dq->rear){**  **dq->front=-1;**  **dq->rear=-1;**  **}**  **else if(dq->rear==-1 && dq->front==0){//to check the queue is empty or not**  **n=-98779;**  **}**  **else if(dq->rear==0){**  **dq->rear=max-1;**  **}else{**  **n=dq->arr[dq->rear];//to store the rear element in new variable**  **dq->rear--;//and decrement the rear value**  **}**  **return n;**  **}**  **bool is\_full\_dequeue(DEQUEUE \*dq){**  **if(dq->rear==max-1){//check queue is full or not**  **return true;**  **}else{**  **return false;**  **}**  **}**  **bool is\_empty\_dequeue(DEQUEUE \*dq){**  **if((dq->front==0) && (dq->rear==-1)){//to check the queue is empty or not**  **return true;**  **}else{**  **return false;**  **}**  **}**  **void display\_dequeue(DEQUEUE \*dq){**  **if(is\_empty\_dequeue(dq)==true){**  **cout<<"The queue is empty";**  **}else{**  **for(int i=dq->front;i<=dq->rear;i++){**  **cout<<dq->arr[i]** **<<" ";//to display the elements**  **}**  **}**  **}**  **int peek\_dequeue(DEQUEUE \*dq){**  **int res;**  **res=dq->arr[dq->front];**  **return res;**  **}**  **Code:**  **#include<iostream>**  **#include"Dequeue\_operation.h"**  **using namespace std;**  **void intia\_dequeue(DEQUEUE\*);**  **void insert\_dequeue\_rear(DEQUEUE\*);**  **void insert\_dequeue\_front(DEQUEUE\*);**  **int del\_dequeue\_front(DEQUEUE\*);**  **int del\_dequeue\_rear(DEQUEUE\*);**  **bool is\_full\_dequeue(DEQUEUE\*);**  **bool is\_empty\_dequeue(DEQUEUE\*);**  **void display\_dequeue(DEQUEUE\*);**  **int peek\_dequeue(DEQUEUE\*);**  **int main(){**  **DEQUEUE q;**  **int choice;**  **intia\_dequeue(&q);**  **do{**  **cout<< "Main menu:"<<endl;**  **cout<< "1. INSERT FROM REAR"<<endl;**  **cout<< "2. INSERT FROM FRONT"<<endl;**  **cout<< "3. DELETE FROM FRONT"<<endl;**  **cout<< "4. DELETE FROM REAR"<<endl;**  **cout<< "5. PEEK"<<endl;**  **cout<< "6. FULL CHECK"<<endl;**  **cout<< "7. DISPLAY"<<endl;**  **cout<< "8. EMPTY CHECK"<<endl;**  **cout<< "9. EXIT"<<endl;**  **cout<<"Enter the option :";**  **cin >>choice;**  **switch(choice){**  **case 1:**  **int element;**  **cout<<"enter the element in the queue ";**  **cin>>element;**  **insert\_dequeue\_rear(&q,element);**  **break;**  **case 2:**  **int ele;**  **cout<<"enter the element in the queue ";**  **cin>>ele;**  **insert\_dequeue\_front(&q,ele);**  **break;**  **case 3:**  **int deleted;**  **deleted=del\_dequeue\_front(&q);**  **cout<<"element deleted is "<<deleted<<endl;**  **break;**  **case 4:**  **int del;**  **del=del\_dequeue\_rear(&q);**  **cout<<"element deleted is "<<del<<endl;**  **break;**  **case 5:**  **cout<<"top element is "<<peek\_dequeue(&q)<<endl;**  **break;**  **case 7:**  **display\_dequeue(&q);**  **break;//changed**  **case 6:**  **bool f\_nf;**  **f\_nf=is\_full\_dequeue(&q);**  **if(f\_nf==true){**  **cout<<"The queue is full."<<endl;**  **}else{**  **cout<<"The queue is not full."<<endl;**  **}**  **break;**  **case 8:**  **bool e\_ne;**  **e\_ne=is\_empty\_dequeue(&q);**  **if(e\_ne==true){**  **cout<<"The queue is empty."<<endl;**  **}else{**  **cout<<"The queue is not empty."<<endl;**  **}**  **break;**  **case 9:**  **exit(0);**  **}**  **}**  **while(choice=9);**  **exit(0);**  **}**  **Screenshot of output:** | 20 |
| 1. **SPARSE MATRIX**   Write a function makesparse that stores a sparse matrix in (k+1) x 3 form where k is the number of non-zero elements. Write functions to add two sparse matrices and subtract one sparse matrix from another where all sparse matrices will be stored in (k+1) x 3 form.  **Your code:** **#include<iostream>**  **# define row 3**  **#define col 3**  **using namespace std;**  **void makesparse(int\*,int\*,int);**  **int count\_nonzero(int\*);**  **void add\_spmat(int\*,int\*,int\*);**  **int main()**  **{**  **int count1,count2;**  **int \*matptr1,\*matptr2,\*spmatptr1,\*spmatptr2;**  **int mat1[row][col];**  **cout<<"Enter the values for the first sparse matrix:"<<endl;**  **for(int i=0;i<row;i++)**  **{**  **for(int j=0;j<col;j++)**  **{**  **cin>>mat1[i][j];**  **}**  **}**  **matptr1=&mat1[0][0];**  **count1=count\_nonzero(matptr1);**  **int spmat1[count1+1][3];**  **spmatptr1=&spmat1[0][0];**  **cout<<"\n The compact sparse of the sparse matrix is: ";**  **makesparse(matptr1,spmatptr1,count1);**      **int mat2[row][col];**    **cout<<"Enter the values for the second sparse matrix:"<<endl;**  **for(int i=0;i<row;i++)**  **{**  **for(int j=0;j<col;j++)**  **{**  **cin>>mat2[i][j];**  **}**  **}**    **matptr2=&mat2[0][0];**  **count2=count\_nonzero(matptr2);**  **int spmat2[count2+1][3];**  **spmatptr2=&spmat2[0][0];**  **cout<<"\n The compact sparse of the sparse matrix is: ";**  **makesparse(matptr2,spmatptr2,count2);**    **int addmat[count1+count2+1][3];**  **int\*addptr;**  **addptr=&addmat[0][0];**    **add\_spmat(spmatptr1,spmatptr2,addptr);**  **return 0;**  **}**  **void makesparse(int\*mat,int\*spmat,int count)**  **{**  **int k=1;**    **for(int i=0;i<row;i++)**  **{**  **for(int j=0;j<col;j++)**  **{**  **if(\*(mat+i\*col+j)!=0)**  **{**  **\*(spmat+k\*3+0)=i;**  **\*(spmat+k\*3+1)=j;**  **\*(spmat+k\*3+2)=\*(mat+i\*col+j);**  **k++;**  **}**    **}**  **}**    **\*(spmat+0+0)=row;**  **\*(spmat+0+1)=col;**  **\*(spmat+0+2)=k-1;**  **for(int i=0;i<k;i++){**  **for(int j=0;j<3;j++){**  **cout<<\*(spmat+i\*3+j)<<" ";**  **}**  **cout<<endl;**  **}**  **}**  **int count\_nonzero(int\*arr)**  **{**  **int count;**  **for(int i=0;i<row;i++)**  **{**  **for(int j=0;j<col;j++)**  **{**  **if(\*(arr+i\*col+j)!=0)**  **{**  **count++;**  **}**    **}**  **}**    **return count;**  **}**  **void add\_spmat(int \*sp1,int \*sp2,int \*sp3)**  **{**  **int totalelement1,totalelement2;**    **if(\*(sp1+0+0)!=\*(sp2+0+0)&& \*(sp1+0+1)!=\*(sp2+0+1))**  **{**  **cout<<"The addition is not possible."<<endl;**  **}**    **totalelement1=\*(sp1+0+2);**  **totalelement2=\*(sp2+0+2);**  **int k1=1,k2=1,k3=1;**    **while(k1<=totalelement1 && k2<=totalelement2)**  **{**  **if(\*(sp1+k1\*3+0)<\*(sp2+k2\*3+0))**  **{**  **\*(sp3+k3\*3+0)=\*(sp1+k1\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp1+k1\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp1+k1\*3+2);**  **k1++;**  **k3++;**  **}**    **else if(\*(sp1+k1\*3+0)>\*(sp2+k2\*3+0))**  **{**  **\*(sp3+k3\*3+0)=\*(sp2+k2\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp2+k2\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp2+k2\*3+2);**  **k2++;**  **k3++;**  **}**    **else if(\*(sp1+k1\*3+0)==\*(sp2+k2\*3+0))**  **{**  **if(\*(sp1+k1\*3+1)<\*(sp2+k2\*3+1))**  **{**  **\*(sp3+k3\*3+0)=\*(sp1+k1\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp1+k1\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp1+k1\*3+2);**  **k1++;**  **k3++;**  **}**    **else if(\*(sp1+k1\*3+1)>\*(sp2+k2\*3+1))**  **{**  **\*(sp3+k3\*3+0)=\*(sp2+k2\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp2+k2\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp2+k2\*3+2);**  **k2++;**  **k3++;**  **}**    **else if(\*(sp1+k1\*3+1)==\*(sp2+k2\*3+1))**  **{**  **\*(sp3+k3\*3+0)=\*(sp1+k1\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp1+k1\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp1+k1\*3+2)+ \*(sp2+k2\*3+2);**  **k1++;**  **k2++;**  **k3++;**  **}**  **}**  **}**    **while(k1<=totalelement1)**  **{**  **\*(sp3+k3\*3+0)=\*(sp1+k1\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp1+k1\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp1+k1\*3+2);**  **k1++;**  **k3++;**  **}**    **while(k2<=totalelement2)**  **{**  **\*(sp3+k3\*3+0)=\*(sp2+k2\*3+0);**  **\*(sp3+k3\*3+1)=\*(sp2+k2\*3+1);**  **\*(sp3+k3\*3+2)=\*(sp2+k2\*3+2);**  **k2++;**  **k3++;**  **}**    **\*(sp3+0+0)=\*(sp1+0+0);**  **\*(sp3+0+1)=\*(sp1+0+1);**  **\*(sp3+0+2)=k3-1;**  **cout<<"The addition of the two matrix is :"<<endl;**  **for(int i=0;i<k3;i++)**  **{**  **for(int j=0;j<3;j++)**  **{**  **cout<<\*(sp3+i\*3+j)<<" ";**  **}**  **cout<<endl;**  **}**  **}**  **Screenshot of output:** | 20 |