**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 –VII** | |
| **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:** | 3rd April, 2021 (10.00 AM – 12.00 Noon) |
| **Signature** |  |

|  |  |
| --- | --- |
| **All Questions are compulsory** | **Total Marks: 50** |

|  |  |
| --- | --- |
| **Question (s)** | **Maximum Marks** |
| 1. Write a program including mystack.h as a header file into your program, reverse a string, concatenate the reverse of the string with original string and lastly check that the new string is in fact a palindrome. Write separate functions corresponding to each task.   **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 <stdlib.h>**  **#include <string.h>**  **#include"MYSTACK.h"**  **string reverse(string);**  **string concatenate(string,string);**  **int palindrome\_or\_not(string);**  **int main(){**  **string s,output,con\_str;**  **int p\_np;**  **cout<<"Enter the string: ";**  **cin>>s;**  **output=reverse(s);**  **cout<<"The reverse string is: "<<output<<endl;**  **con\_str=concatenate(s,output);**  **cout<<"The concatenate string is: "<<con\_str<<endl;**  **p\_np=palindrome\_or\_not(con\_str);**  **if(p\_np==0){**  **cout<<"The concatenate string is palindrome";**  **}else{**  **cout<<"The concatenate string is not palindrome";**  **}**  **}**  **string reverse(string s1){**  **STACK A;**  **int j;**  **char b;**  **string res;**  **for(int i=0;i<s1.length();i++){**  **push(&A,s1[i]);**  **}**  **while(j<s1.length()){**  **b=pop(&A);**  **res+=b;**  **j++;**  **}**  **return res;**  **}**  **string concatenate(string s,string s1){**  **string result;**  **result=s+s1;**  **return result;**  **}**  **int palindrome\_or\_not(string s2){**  **STACK A;**  **int p\_np;**  **char b;**  **string out;**  **for(int i=0;i<s2.length();i++){**  **push(&A,s2[i]);**  **}**  **int j=0;**  **while(j<s2.length()){**  **b=pop(&A);**  **out+=b;**  **j++;**  **}**  **p\_np=s2.compare(out);**  **return p\_np;**  **}**  **Screenshot of output:** | 20 |
| 1. Write a program to evaluate a postfix expression using a stack of values. [i/p – 12 10 \* 5+ would give an output 125, 5 10\* + would give an error]   **Your code: header file**  **#include <iostream>**  **#include <cstdlib>**  **#include <string.h>**    **using namespace std;**    **struct Stack{**  **int top;**  **unsigned capacity;**  **int\* array;**  **};**    **struct Stack\* STACK(unsigned capacity){**  **struct Stack\* stack = (struct Stack\*) malloc(sizeof(struct Stack));**    **if (!stack) return NULL;**    **stack->top = -1;**  **stack->capacity = capacity;**  **stack->array = (int\*) malloc(stack->capacity \* sizeof(int));**    **if (!stack->array) return NULL;**    **return stack;**  **}**    **int isEmpty(struct Stack\* stack){**  **return stack->top == -1 ;**  **}**    **char peek(struct Stack\* stack){**  **return stack->array[stack->top];**  **}**    **char pop(struct Stack\* stack){**  **if (!isEmpty(stack))**  **return stack->array[stack->top--];**  **}**    **void push(struct Stack\* stack, char op){**  **stack->array[++stack->top] = op;**  **}**  **Code:**  **#include <iostream>**  **#include <cstdlib>**  **#include <string.h>**  **#include "STACK.VSC.h"**    **using namespace std;**      **//returns value of a postfix expression**  **int evaluatePostfix(char\* exp){**  **// stack of capacity equal to expression size**  **struct Stack\* stack = STACK(strlen(exp));**  **int i;**    **// if stack was created CORRECT**  **if (!stack) return -1;**    **// scan all characters one by one**  **for (i = 0; exp[i]; ++i)**  **{**  **/\* If the scanned character is an operand (number here),**  **push it to the stack. \*/**  **if (isdigit(exp[i]))**  **push(stack, exp[i] - '0');**    **/\* If the scanned character is an operator, pop two**  **elements from stack apply the operator \*/**  **else**  **{**  **int a = pop(stack);**  **int b = pop(stack);**  **switch (exp[i])**  **{**  **case '+':{**  **push(stack, b+a);**  **break;**  **}**  **case '-':{**  **push(stack, b-a);**  **break;**  **}**  **case '\*':{**  **push(stack, b\*a);**  **break;**  **}**  **case '/':{**  **push(stack, b/a);**  **break;**  **}**  **}**  **}**  **}**  **return pop(stack);**  **}**    **// no.**  **int main(){**  **char exp[100];**  **cout<<"enter the expression: ";**  **cin>>exp;**  **cout<<"postfix evaluation: "<<evaluatePostfix(exp);**  **return 0;**  **}**  **Screenshot of output:** | 10 |
| 1. Write a program to convert an infix expression into its corresponding postfix expression. The expression contains alphabets, operators and parentheses.   During the conversion all possible checks for the correctness should be checked. [ (a+b)/(c-d) would output ab+cd-/, ((a+b)^c-d would give error as “unmatched parenthesis]  **Your code: header file**  **#include <iostream>**  **using namespace std;**  **#define size 20**  **typedef struct stack{**  **char array[size];**  **int top;**  **}STACK;**  **char 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;**  **}**  **char push(stack \*s1,char element ){**  **s1->top++;**    **s1->array[s1->top]=element;**  **return element;**  **}**  **char pop(stack \*s1){**  **double 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 <cstdlib>**  **#include <cmath>**  **#include<string.h>**  **#include"INFIXTOPOSTFIX.h"**  **using namespace std;**  **#define MAX 20**  **char stk[20];**  **int top=-1;**  **void push(char op){**  **if(top==MAX-1)**  **{**  **cout<<"stackfull!";**  **}**  **else{**  **top++;**  **stk[top]=op;**  **}**  **}**  **char pop(){**  **char ch;**  **if(top==-1){**  **cout<<"stackempty!";**  **}else{**  **ch=stk[top];**  **stk[top]='\0';**  **top--;**  **return(ch);**  **}**  **return 0;**  **}**  **int precedence (char num\_value){**  **//check character is operator symbol or not**  **if(num\_value == '+' || num\_value =='-')**  **{**  **return(1);**  **}**    **if(num\_value == '\*' || num\_value =='/')**  **{**  **return(2);**  **}**    **if(num\_value == '^')**  **{**  **return(3);**  **}**    **return 0;**  **}**  **string convert(string infix){**  **int i=0;**  **string postfix = "";**  **while(infix[i]!='\0'){**  **if(infix[i]>='A' && infix[i]<='Z'|| infix[i]>='a'&& infix[i]<='z'){ //eng alphabet**  **postfix.insert(postfix.end(),infix[i]);**  **i++;**  **}**  **//Function to verify parentheses**  **else if(infix[i]=='(' || infix[i]=='{' || infix[i]=='['){**  **push(infix[i]);**  **i++;**  **}**  **else if(infix[i]==')' || infix[i]=='}' || infix[i]==']'){**  **if(infix[i]==')'){**  **while(stk[top]!='(')**  **{**  **postfix.insert(postfix.end(),pop());**  **}**  **pop();**  **i++;**  **}**  **if(infix[i]==']'){**  **while(stk[top]!='[')**  **{**  **postfix.insert(postfix.end(),pop());**  **}**  **pop();**  **i++;**  **}**    **if(infix[i]=='}'){**  **while(stk[top]!='{'){**  **postfix.insert(postfix.end(),pop());**  **}**  **pop();**  **i++;**  **}**  **}else{**  **if(top==-1){**  **push(infix[i]);**  **i++;**  **}**    **else if(precedence(infix[i]) <= precedence(stk[top])){**  **postfix.insert(postfix.end(),pop());**    **while(precedence(stk[top]) == precedence(infix[i])){**  **postfix.insert(postfix.end(),pop());**    **if(top < 0){**  **break;**  **}**  **}**  **push(infix[i]);**  **i++;**  **}**  **else if(precedence(infix[i]) > precedence(stk[top])) {**  **push(infix[i]);**  **i++;**  **}**  **}**  **}**  **while(top!=-1){**  **postfix.insert(postfix.end(),pop());**  **}**  **cout<<"The final Postfix Expression is : "<<postfix; //it will print postfix expression**  **return postfix;**  **}**  **int main(){**  **int conversion;**  **string infix, postfix;**  **cout<<"\nEnter the infix expression is : "; //enter the expression**  **cin>>infix;**  **postfix = convert(infix);**  **return 0;**  **}**  **Screenshot of output:** | 20 |