***NAME : Himanshu Dixit***

***ENROLL NO. : 21103262***

***BATCH : B11***

***Data Structure [15B11CI311]***

***Tutorial Sheet***

***Week 3***

***Q.1.*** *Write pseudo code to reverse a string using Stack, with and without using extra data  structure.*

***Solution :***

***With Extra Data structure Reverse:***

*stack\* reverse\_withDS(stack\* top1)*

*{*

*stack\* top2;*

*stack\* curr=top1;*

*while(curr)*

*{*

*stack\* temp=new stack;*

*temp->data=curr->data;*

*temp->next=NULL;*

*if(top2==NULL)*

*{*

*top2=temp;*

*curr=curr->next;*

*}*

*else*

*{*

*temp->next=top2;*

*top2=temp;*

*curr=curr->next;*

*}*

*}*

*return top2;*

*}*

***Without Extra Data structure :***

*stack\* reverse\_withoutExtraDS(stack\* top1)*

*{*

*stack \*curr,\*prev,\*forwardd;*

*curr=top1;*

*prev=top1;*

*forwardd=top1->next;*

*while(curr)*

*{*

*if(forwardd==NULL)*

*{*

*curr->next=prev;*

*top1=curr;*

*return top1;*

*}*

*if(curr==prev)*

*{*

*curr->next=NULL;*

*}*

*else*

*{*

*curr->next=prev;*

*}*

*prev=curr;*

*curr=forwardd;*

*forwardd=forwardd->next;*

*}*

*}*

***Q.2.*** *How will you insert elements from a stack to a doubly linked list and a circular linked list.*

***Solution :***

*// Q2) insert items in DLL*

*#include <iostream>*

*using namespace std;*

*template <class T>*

*class stack*

*{*

*private:*

*struct node*

*{*

*T item;*

*node \*next;*

*};*

*public:*

*node \*top;*

*stack() { top = NULL; }*

*void push(T data)*

*{*

*node \*temp;*

*temp = new node();*

*temp->item = data;*

*temp->next = top;*

*top = temp;*

*}*

*void pop()*

*{*

*node \*temp;*

*temp = top;*

*top = top->next;*

*delete temp;*

*}*

*};*

*class DLL*

*{*

*private:*

*struct node*

*{*

*int item;*

*node \*prev;*

*node \*next;*

*};*

*node \*head;*

*public:*

*DLL() { head = NULL; }*

*void insert\_first\_DLL(int data)*

*{*

*node \*temp;*

*temp = new node;*

*temp->item = data;*

*temp->next = NULL;*

*temp->prev = NULL;*

*if (head == NULL)*

*{*

*head = temp;*

*}*

*else*

*{*

*temp->next = head;*

*head->prev = temp;*

*head = temp;*

*}*

*}*

*void insert\_last\_DLL(int data)*

*{*

*node \*temp;*

*node \*curr;*

*temp = new node;*

*temp->item = data;*

*temp->next = NULL;*

*temp->prev = NULL;*

*if (head == NULL)*

*{*

*head = temp;*

*}*

*else*

*{*

*curr = head;*

*while (curr->next != NULL)*

*{*

*curr = curr->next;*

*}*

*temp->prev = curr;*

*curr->next = temp;*

*}*

*}*

*void view\_list\_DLL()*

*{*

*node \*temp;*

*if (head == NULL)*

*{*

*cout << "list is empty\n";*

*}*

*else*

*{*

*temp = head;*

*while (temp != NULL)*

*{*

*cout << temp->item << " ";*

*temp = temp->next;*

*}*

*}*

*}*

*};*

*int main()*

*{*

*stack<int> s;*

*DLL dlist;*

*int n, val;*

*cout << "enter no. of elements to insert in stack: ";*

*cin >> n;*

*int i, data;*

*cout << "enter values to stack: ";*

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

*{*

*cin >> data;*

*s.push(data);*

*}*

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

*{*

*val = s.top->item;*

*dlist.insert\_first\_DLL(val);*

*s.pop();*

*}*

*cout << endl*

*<< "stack items inserted in doubly linked list: ";*

*dlist.view\_list\_DLL();*

*return 0;*

*}*

*// Q2) insert items in CLL*

*#include <iostream>*

*using namespace std;*

*template <class T>*

*class stack*

*{*

*private:*

*struct node*

*{*

*T item;*

*node \*next;*

*};*

*public:*

*node \*top;*

*stack() { top = NULL; }*

*void push(T data)*

*{*

*node \*temp;*

*temp = new node();*

*temp->item = data;*

*temp->next = top;*

*top = temp;*

*}*

*void pop()*

*{*

*node \*temp;*

*temp = top;*

*top = top->next;*

*delete temp;*

*}*

*};*

*class CLL*

*{*

*private:*

*struct node*

*{*

*int item;*

*node \*next;*

*};*

*node \*tail;*

*public:*

*CLL() { tail = NULL; }*

*void insert\_first\_CLL(int data)*

*{*

*node \*temp;*

*temp = new node();*

*temp->item = data;*

*temp->next = NULL;*

*if (tail == NULL)*

*{*

*tail = temp;*

*temp->next = temp;*

*}*

*else*

*{*

*temp->next = tail->next;*

*tail->next = temp;*

*}*

*}*

*void insert\_last\_CLL(int data)*

*{*

*node \*temp;*

*temp = new node();*

*temp->item = data;*

*temp->next = NULL;*

*if (tail == NULL)*

*{*

*tail = temp;*

*temp->next = tail;*

*}*

*else*

*{*

*temp->next = tail->next;*

*tail->next = temp;*

*tail = temp;*

*}*

*}*

*void view\_list\_CLL()*

*{*

*node \*temp;*

*if (tail == NULL)*

*{*

*cout << "list is empty\n";*

*}*

*else*

*{*

*temp = tail->next;*

*do*

*{ /\* code \*/*

*cout << temp->item << " ";*

*temp = temp->next;*

*} while (temp != tail->next);*

*}*

*}*

*};*

*int main()*

*{*

*stack<int> s;*

*CLL clist;*

*int n, val;*

*cout << "enter no. of elements to insert in stack: ";*

*cin >> n;*

*int i, data;*

*cout << "enter values to stack: ";*

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

*{*

*cin >> data;*

*s.push(data);*

*}*

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

*{*

*val = s.top->item;*

*clist.insert\_first\_CLL(val);*

*s.pop();*

*}*

*cout << endl*

*<< "stack items inserted in circular linked list: ";*

*clist.view\_list\_CLL();*

*return 0;*

*}*

***Q.3.*** *Compare the reversal of a singly linked list, doubly linked list and a circular linked list in  term of number of operations/swaps.*

***Solution :***

*All the three reversal will take O(n) time.*

***Q.4.*** *How will the elements of a queue be inserted from a stack, using 2 stacks.*

***Solution :***

*Let queue to be implemented be q and stacks used to implement q be stack1 and stack2.*

*This method makes sure that oldest entered element is always at the top of stack 1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used.*

*enQueue(q, x):*

*While stack1 is not empty, push everything from stack1 to stack2.*

*Push x to stack1 (assuming size of stacks is unlimited).*

*Push everything back to stack1.*

*deQueue(q):*

*If stack1 is empty then error*

*Pop an item from stack1 and return it*

***Q.5.*** *What are prefix, postfix and infix notations?*

***Solution :***

***Infix:*** *The typical mathematical form of expression that we encounter generally is known as infix notation. In infix form, an operator is written in between two operands.*

***For example:*** *An expression in the form of* ***A \* ( B + C ) / D*** *is in infix form. This expression can be simply decoded as: “Add B and C, then multiply the result by A, and then divide it by D for the final answer.”*

***Prefix:*** *In prefix expression, an operator is written before its operands. This notation is also known as “Polish notation”.*

***For example:*** *The above expression can be written in the prefix form as* ***/ \* A + B C D****. This type of expression cannot be simply decoded as infix expressions.*

***Postfix:*** *In postfix expression, an operator is written after its operands. This notation is also known as “Reverse Polish notation”.*

***For example:*** *The above expression can be written in the postfix form as* ***A B C + \* D /****. This type of expression cannot be simply decoded as infix expressions.*

***Q.6.*** *Convert an infix notation to prefix and postfix notation. Which Data structure is suitable?*

***Solution :***

*Infix to Postfix Conversion*

*In infix expressions, the operator precedence is implicit unless we use parentheses. Therefore, we must define the operator precedence inside the algorithm for the infix to postfix conversion.*

*---Create a stack.*

*---For each character c in the input stream:*

*If c is an operand*

*{*

*Output c*

*}*

*Else if c is a right parentheses*

*{*

*Pop and output tokens until a left parentheses is popped*

*}*

*Else*

*{ // c is an operator or left parentheses*

*Pop and output tokens until one of the lower priorities than c*

*are encountered, or a left parentheses is encountered, or the stack is empty.*

*Push c*

*}*

*INFIX TO PREFIX*

*Reverse the infix expression i.e A+B\*C will become C\*B+A. Note while reversing each ‘(‘ will become ‘)’ and each ‘)’ becomes ‘(‘.*

*Obtain the “nearly” postfix expression of the modified expression i.e CB\*A+.*

*Reverse the postfix expression.*

*+A\*BC.*

*We have to pop all the operators from the stack which are greater in precedence than that of the scanned operator.*

*in the case of “^” operator,*

*pop operators from the stack which are >= in precedence.*

***Q.7.*** *Solve a given postfix notation using a stack.*

***Solution :***

*#include <iostream>*

*#include <string.h>*

*using namespace std;*

*struct Stack*

*{*

*int top;*

*unsigned capacity;*

*int\* array;*

*};*

*struct Stack\* createStack( 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--] ;*

*return '$';*

*}*

*void push(struct Stack\* stack, char op)*

*{*

*stack->array[++stack->top] = op;*

*}*

*int evaluatePostfix(char\* exp)*

*{*

*struct Stack\* stack = createStack(strlen(exp));*

*int i;*

*if (!stack) return -1;*

*for (i = 0; exp[i]; ++i)*

*{*

*if (isdigit(exp[i]))*

*push(stack, exp[i] - '0');*

*else*

*{*

*int val1 = pop(stack);*

*int val2 = pop(stack);*

*switch (exp[i])*

*{*

*case '+': push(stack, val2 + val1); break;*

*case '-': push(stack, val2 - val1); break;*

*case '\*': push(stack, val2 \* val1); break;*

*case '/': push(stack, val2/val1); break;*

*}*

*}*

*}*

*return pop(stack);*

*}*

*int main()*

*{*

*char exp[] = "231\*+9-";*

*cout<<"postfix evaluation: "<< evaluatePostfix(exp);*

*return 0;*

*}*

***Q.8.*** *Given a sequence of parenthesis, check if the sequence is balanced or not.*

***Solution :***

*#include <iostream>*

*using namespace std;*

*template <class T>*

*class stack*

*{*

*private:*

*struct node*

*{*

*T item;*

*node \*next;*

*};*

*public:*

*node \*top;*

*stack() { top = NULL; }*

*void push(T data)*

*{*

*node \*temp;*

*temp = new node();*

*temp->item = data;*

*temp->next = top;*

*top = temp;*

*}*

*void pop()*

*{*

*node \*temp;*

*temp = top;*

*top = top->next;*

*delete temp;*

*}*

*};*

*void check(string seq)*

*{*

*stack<char> s;*

*int flag = 0;*

*int i = 0;*

*char ch;*

*while (seq[i] != '\0')*

*{*

*if (seq[i] == '(' || seq[i] == '{' || seq[i] == '[')*

*{*

*s.push(seq[i]);*

*}*

*else if (s.top != NULL)*

*{*

*ch = s.top->item;*

*if (seq[i] == ')' && ch == '(')*

*{*

*s.pop();*

*}*

*else if (seq[i] == '}' && ch == '{')*

*{*

*s.pop();*

*}*

*else if (seq[i] == ']' && ch == '[')*

*{*

*s.pop();*

*}*

*else*

*{*

*flag = 1;*

*break;*

*}*

*}*

*else*

*{*

*flag = 1;*

*break;*

*}*

*i++;*

*}*

*if (flag == 1 || s.top != NULL)*

*{*

*cout << "NO\n";*

*}*

*else if (s.top == NULL)*

*{*

*cout << "YES\n";*

*}*

*}*

*int main()*

*{*

*string seq;*

*cout << "enter string: ";*

*cin >> seq;*

*check(seq);*

*return 0;*

*}*

