Week-7

Lab Session:

Date of the Session: ___/___/___ **Time of the Session:** _____to_____

Pre-lab:

1. Write The postfix form of expression (A+B)*C/D ? Which data structure is required
to convert infix to postfix?

Data Structure Required, use a stack

(1). A+B → AB+

(2). C → AB+C*

(3). D → AB+C*D/

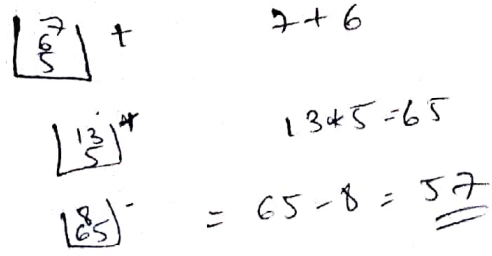
2. What is reverse polish notation and give an example?

It is postfix notation, is a mathematical notation in which operators follow their operands. It eliminates the need for parentheses to define operation precedence, as the order of operation is inherently determined by the sequence of operands and operators. Example is Infix :-(3+4)×5 , Postfix :-*+3 4 5

3. Write steps Evaluate postfix expression using stack?

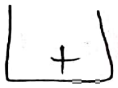
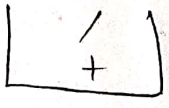
① Initialize an empty stack
② Scan the expression from left to right
③ Repeat this process until the entire expression is scanned.
④. The final value left in the stack is the result.

4. Find the result of postfix expression "5 6 7 + * 8 -" ?



5. Convert the infix expression to a postfix expression

4+2*(8-6)/3

4		4
4		4
2		4 2
*		4 2
(		4 2
8		4 2 8
-		4 2 8
6		4 2 8 6
)		4 2 8 6 -
/		4 2 8 6 - *
3		4 2 8 6 - * 3 / +

In-lab:

1. Transform the Expression

Reverse Polish Notation (RPN) is a mathematical notation where every operator follows all of its operands. For instance, to add three and four, one would write "3 4 +" rather than "3 + 4". If there are multiple operations, the operator is given immediately after its second operand; so the expression written "3 − 4 + 5" would be written "3 4 − 5 +" first subtract 4 from 3, then add 5 to that. Transform the algebraic expression with brackets into RPN form. You can assume that for the test cases below only single letters will be used, brackets [] will not be used and each expression has only one RPN form (no expressions like a*b*c)

Link:https://www.codechef.com/practice/course/stacks-and-queues/STAQUEF/problems/ONP

Program:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX 1000

struct Stack {
    int top;
    char items[MAX];
};

void initStack (struct Stack *s){
    s->top = -1;
}
void push (struct Stack *s, char value){
    if (s->top < MAX -1) {
        s->items[1 + (s->top)] = value;
    }
}
char pop(struct Stack *s) {
    if (s->top != -1){
        return s->items[(s->top)--];
    }
    return '\0';
}
char peek(struct Stack *s){
    if (s->top != -1){
        return s->items[s->top];
    }
    return '\0';
}
int isEmpty (struct Stack *s) {
    return s->top == -1;
}
```

```
int main(){
    int t;
    scanf("%d", &t);
    while (t--) {
        char expression[MAX], result[MAX];
        scanf("%s", expression);
        struct stack my_stack;
        initStack(&my_stack);
        int result_index = 0;
        int length = strlen(expression);
        for(int i=0; i<length; i++) {
            if (isalpha(expression[i])) {
                result[result_index++] = expression[i];
            }
            else if (expression[i] == '(') {
                push(&my_stack, expression[i]);
            }
            else if(expression[i] == ')') {
                while (peek(&my_stack) != '(') {
                    result[result_index++] = pop(&my_stack);
                }
                pop(&my_stack);
            }
            else{
                push(&my_stack, expression[i]);
            }
        }
        result[result_index]='\0';
        printf("%s\n", result);
    }
    return 0;
}
```

Sample Input and Output:

3

(a+(b*c))

((a+b)*(z+x))

((a+t)*((b+(a+c))^(c+d)))

OUTPUT :- abc*+

ab+zx+*

at+bac++cd+^*

2. Infix to Postfix

You are given an Infix expression.
Use the stack data structure to convert it to a Postfix expression.
Stack is already implemented for your convenience in the code editor.
You can use the stack using the following functions-
createStack(capacity) - This function is used to create a stack. It takes one argument capacity which defines the maximum capacity of stack.

isEmpty() - This function returns 1 if the stack is empty, otherwise it returns 0.

peek() - This function returns the top element in the stack.

push(stack, item) - This function can be used to push items into the stack. It takes two arguments, stack - the pointer to the stack, and the item to push in the stack.

pop() - This function can be used to pop the top element of the stack.

Task

Write a program in C to convert an infix expression to postfix expression using stack data structure.

Link:https://www.codechef.com/learn/course/college-data-structures-c/CPDSC02/problems/DSAC29E

Program:

```
#include <stdio.h>
#define MAX 100
typedef struct { int nums[MAX], size; }NoteBook;

void insert(Notebook* n, int num, int idx) {
    if (idx < 0 || idx > n->size || n->size >= MAX) return;
    for(int i=n->size; i>idx; i--) n->nums[i]=n->nums[i
    n->nums[idx]=num, n->size++;
}
void delete(Notebook* n, int num){
    for(int i=0; i<n->size; i++) if(n->nums[i]==num
    for(int j=i; j<n->size-1; j++)n->nums[j]=n->nums[j
    n->size--; return;
}
}
```

```
void display(Notebook *n){
        for(int i=0;i<n->size;i++)printf("%d",n->nums[i]);
        printf("\n");
}
int main(){
    Notebook n = {.size=0};
    printf("Size:"), scanf("%d", &n.size);
    for(int i=0;i<n.size;i++) scanf("%d",&n.nums[i]);

    int ch,num,idx;
    while(1){
        printf("\n1.Insert 2.Delete 3.Display 4.Exit\n");
        scanf("%d", &ch);
        if(ch==4) break;
        if(ch==1) scanf("%d%d",&num,&idx);
        if(ch==2) scanf("%d%d",&num);deletat(&n,idx);
        if(ch==3) display(&n);
    }
}
```

Sample Input and Output:

```
7
A+B*C+D
```

OUTPUT :- ABC*+D+

3.Valid Parenthesis

Give string a S consisting of only (and). Find whether S is a valid parenthesis string.

Note: A **valid parentheses** string is defined as:

- Empty string is valid.
- If P is valid, (*P*) is also valid.
- If P and Q are valid, PQ is also valid.

Link: https://www.codechef.com/practice/course/stacks-and-queues/STAQUEF/problems/PREP59?tab=statement

Program:

```
#include <stdio.h>
#include <stdlib.h>

bool ispar(char *s){
    char stack[10000]; int i=-1;
    for(int j=0; s[j]; j++){
        if(s[j]=='(' || s[j]=='{' || s[j]=='[')
            stack[++i] = s[j];
        else if(i>=0 && ((stack[i]=='(' && s[j]==')')
                || stack[i]=='{' && s[i]=='}'))
            i--;
        else return false;
    }
    return i==-1;
}
int main(){
    int t; scanf("%d",&t);
    while(t--){
        char s[10001];
        scanf("%s", s); printf("%d\n", ispar(s));
    }
}
```

Sample Input and Output

3)(()) (()()))((	OUTPUT 1 0 0

Post Lab

1. Given a string s representing a valid expression, implement a basic calculator to evaluate it, and return the result of the evaluation.
   Note: You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as eval().
   Link: https://leetcode.com/problems/basic-calculator/description/

Program

```
int calc(char **s){
    int result = 0;
    short minus = 1;
    while (**s != 0 && **s != ')'){
        if (isdigit(**s)) {
            result += minus * strtol(*s, &*s, 10);
            minus *= minus;
            (*s)--;
        }else if (**s == '('){
            (*s)++;
            result += minus * calc(s);
            minus *= minus;
        }else if (**s == '-') minus = -1;
        (*s)++;
    }
    return result;
}
int calculate(char *s) {return calc(&s);}
```

Sample Input and output

S:- "1 + 1"

OUTPUT :- 2

2. Given a string s which represents an expression, evaluate this expression and return its value. The integer division should truncate toward zero. You may assume that the given expression is always valid. All intermediate results will be in the range of [-231, 231 - 1].

Link: https://leetcode.com/problems/basic-calculator-ii/description/

Program:

```
int calculate(char *s){
int num[150000];
int top=-1;
int CrrNum=0;
char PrevOperator='+';

for(int i=0;s[i]!='\0';i++){
    char c=s[i];
if(isdigit(c)){
    CrrNum=CrrNum*10+(c-'0');
}
if(c=='+'||c=='-'||c=='*'||c=='/'||s[i+1]=='\0')
{
    if(PrevOperator=='+'){
        nums[++top]=CrrNum;
}else if(PrevOperator=='-'){
        nums[++top]=-CrrNum;
}else if(PrevOperator=='*'){
        nums[top]*=CrrNum;
```

```
}elseif (PrevOperator == '/') {
        nums[top] = nums[top] / CrrNum;
}
PrevOperator = c;
CrrNum = 0;
}
}
int result = 0;
for (int i = 0; i <= top; i++) {
        result += nums[i];
}
return result;
}
```

Sample Input and Output:

S = "3+2*2"

OUTPUT :- 7

1. Evaluate Reverse Polish Notation

You are given an array of strings tokens that represents an arithmetic expression in a Reverse Polish Notation. Evaluate the expression. Return an integer that represents the value of the expression.

Link:https://leetcode.com/problems/evaluate-reverse-polish-notation/description/

Program:

```
int evalRPN (char ** tokens, int tokensSize) {
    typedef struct Stack{
        int item[10000];
        int top;
    }stack;
    Stack s;
    s.top = -1;
    for(int i = 0; i < tokensSize ; i++){
        char *token = token[i];
        if(strcmp(token,"+")==0||strcmp(token,"-")==0||
        strcmp(token,"*")==0||strcmp(token,"/")==0){
            int b = s.item[s.top--];
            int a = s.item[s.top--];
            if( strcmp(token,"+")==0)
                s.item[++s.top] = a+b;
```

```
else if (strcmp (token, "-") == 0)
        s.item [++s.top] = a - b;
else if (strcmp (token, "*") == 0)
        s.item [++s.top] = a * b;
else if (strcmp (token, "/") == 0)
        s.item [++s.top] = a / b;
else {
        s.item [++s.top] = atoi (token);
    }
}
return s.item [s.top];
}
```

Sample Input and Output:

tokens = ["2", "1", "+", "3", "*"]

OUTPUT = 9

2. A bracket is considered to be any one of the following characters: (,), {, }, [, or].

Two brackets are considered to be a *matched pair* if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e.,),], or }) *of the exact same type*. There are three types of matched pairs of brackets: [], {}, and ().

Given strings of brackets, determine whether each sequence of brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

Link: https://www.hackerrank.com/challenges/balanced-brackets/problem

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char * isBalanced (char *s) {
    char stack[10001]; int top = -1;
    for(int i=0; s[i]; i++) {
        if (s[i]=='(' || s[i]=='{' || s[i]=='[')
            stack[++top] = s[i];
        else if (top>=0 && ((stack[top]=='(' && s[i]==')') ||
                            (stack[top]=='{' && s[i]=='}') ||
                            (stack[top]=='[' && s[i]==']')))
            top--;
        else return "NO";
    }
    return (top==-1)? "YES" : "NO";
}
```

Sample Input and Output

3	OUTPUT
{[()]}	YES
{[()]}	NO
{{[[(())]]}}	YES

3.Given a parentheses string s containing only the characters '(' and ')'.

A parentheses string is **balanced** if:

- Any left parenthesis '(' must have a corresponding two consecutive right parenthesis '))'.
- Left parenthesis '(' must go before the corresponding two consecutive right parenthesis '))'.

In other words, we treat '(' as an opening parenthesis and '))' as closing parenthesis.

For example,
"())", "())(()))" and "(())())))" arebalanced, ")()", "()))" and "(()))" are not balanced.

You can insert the characters '(' and ')' at any position of the string to balance it if needed.

Return *the minimum number of insertions* needed to make s balanced.

Link: https://leetcode.com/problems/minimum-insertions-to-balance-a-parentheses-string/submissions/1478544470/

Program:

```
int minInsertions(char *s) {
    int n=strlen(s);
    int left=0, insert_left=0, insert_right=0;
    for(int i=0;i<n;i++){
        if(s[i]=='(') left++;
        else {
            if(i+1<n && s[i+1]==')'){
                if(left<=0) insert_left++;
                else left--;
                i++;
            }else{
                insert_right++;
                if(left>0) left--;
                else insert_left++;
```


3 3 3

return left * 2 + insert_right + insert_left;

}

Sample Input and Output:

Input :- S = "(())"

Output :- 9

4. Evaluate Suffix Expression

You are given an Suffix expression with single digit numbers as operands.
Use stack to evaluate the expression and output the answer.
The operators in the expression may include +, -, *, /, %(modulus), ^(power).
Stack is already implemented for your convenience in the code editor.
You can use the stack using the following functions-

- createStack(capacity) - This function is used to create a stack. It takes one argument capacity which defines the maximum capacity of stack.
- isEmpty() - This function returns 1 if the stack is empty, otherwise it returns 0.
- push(stack, item) - This function can be used to push items into the stack. It takes two arguments, stack - the pointer to the stack, and the item to push in the stack.
- pop() - This function can be used to pop the top element of the stack.

Task

Write a program in C to evaluate the Suffix Expression using stack data structure and print the output.

Link: https://www.codechef.com/learn/course/college-data-structures-c/CPDSC02/problems/DSAC29F?tab=solution

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>

int eval(char *s){
    int stack[100],top=-1,a,b;
```

```
for(int i=0; s[i]; i++) {
    if (isdigit(s[i])) stack[++top] = s[i] - '0';
    else {
        b = stack[top--], a = stack[top--];
        stack[++top] = (s[i]=='+')?(a+b):(s[i]=='
                       (s[i]=='*')?(a*b):(s[i]=='
                       (s[i]=='%')?(a%b):(int)pow(a,b);
    }
}
return stack[top];
}

int main() {
    char s[101];
    scanf("%s", s);
    printf("%d\n", eval(s));
}
```

Sample Input and Output

7

23+45+*

OUTPUT

45

5.Given a **0-indexed** string word and a character ch, **reverse** the segment of word that starts at index 0 and ends at the index of the **first occurrence** of ch (**inclusive**). If the character ch does not exist in word, do nothing.

- For example,

if word = "abcdefd" and ch = "d", then you should **reverse** the segment that starts at 0 and ends at 3 (**inclusive**). The resulting string will be "dcbaefd".

Return *the resulting string.*

Link: https://leetcode.com/problems/reverse-prefix-word/description/?envType=problem-list-v2&envId=stack

Program:

```
#include <stdio.h>
#include< string.h>
void reverseprefix(char *word ,char ch) {
    int i, index=-1;
    for (i=0; word[i]!='\0' ;i++) {
        if (word[i]==ch) {
            index = i;
            break;
        }
    }
    if (index !=-1) {
        int left=0, right=index;
        while (left<right) {
            char temp = word[left];
            word [left] =word[right];
            word [right ]= temp;
            left ++;
            right --;
```

```
}
}
}
int main() {
  char ch;
  printf("Enter a word: ");
  scanf("%s", word);
  printf("Enter a character: ");
  scanf("%c", &ch);
  reverseBefore(word, ch);
  printf("Resulting word: %s\n", word);
  return 0;
}
```

Sample Input and Output:

Enter a word: openai
Enter a character: z
Output: Resulting word: openai