USN | | | | | | | | | | |

Internal Assessment Test 1 – December 2023

Sub:	Data Structures and Applications					Sub Code:	BCS304	Branch :	CSE	
Date:	18-12-2023	Duration:	90 mins	Max Marks:	50	Sem / Sec:	III (A, B & C)		OBE	

Answer any FIVE FULL Questions

		MARKS	CO	RBT

1 (a)	Many Registration forms for various websites asks the Users to fill out the address details containing the following details:
	a. Flat No/Building Number b. Street
	c. District d. City
	e. Country f. Pin code
	Create a Structure to maintain the address of a user with above-mentioned details with and without typedef. Also choose appropriate datatype for the same.
	
	SOLUTION
	
	Struct Registration{
	Int flatno;
	Char *street; or char street[20];
	Char * District;
	Char *city;
	Char *country;
	Long int pincode
	};
	
	Typedef Struct{
	Int flatno;
	Char *street; or char street[20];
	Char * District;
	Char *city;
	Char *country;
	Long int pincode
	}Registration;
		4	CO1	L3

1 (b)	Identify an appropriate data structure to handle the data of the followig usecases: a. Phone Contacts in Mobile phone b. Cart in Online Shopping platforms. Also, provide a brief justification about the data structure you have opted with proper reasoning. **SOLUTION** **a.** Data structures for Phone contacts is Array because the phone numbers will be saved in alphabetical order that serves as an index to search through b. Data structures used for cart in online shopping platforms is stack where the newly added item will be added on the top of the other	2	CO1	L3
1(c)	Based on your data structure chosen for 1b. Provide any two operations for each use case that can be performed.	4	CO1	L3

	SOLUTION **For Phone Contacts :** Operations are addContact() – where a new contact can be added, deleteContact()- where an existing contact can be deleted. **For Cart :** Operations are addItem()- where new item can be added to the cart and deleteItem() where an item can be deleted from the cart.			

2 (a)	Define Pointers. List the advantages of pointers over arrays
	
	SOLUTION
	A pointer is a variable that stores the memory address of another variable as its value. A pointer variable points to a data type (like int) of the same type, and is created with the * operator.
	The address of the variable you are working with is assigned to the pointer:
	int myAge = 43; // An int variable
	int* ptr = &myAge; // A pointer variable, with the name ptr, that stores the address of myAge
	// Output the value of myAge (43)
	printf("%d\n", myAge);
	// Output the memory address of myAge (0x7ffe5367e044)
	printf("%p\n", &myAge);
	// Output the memory address of myAge with the pointer (0x7ffe5367e044)
	printf("%p\n", ptr);
	
	Advantages:
	1. sizeof Operator:
	sizeof(array) returns the amount of memory used by all elements in the array where as sizeof(pointer) only returns the amount of memory used by the pointer variable itself
	2. t& operator
	array is an alias for &array[0] and returns the address of the first element in the array &pointer returns the address of the pointer
	3. a string literal initialization of a character array
	char array[] = “abc” sets the first four elements in array to ‘a’, ‘b’, ‘c’, and ‘\0’
	char *pointer = “abc” sets the pointer to the address of the “abc” string (which may be stored in read-only memory and thus unchangeable)
	4. Pointer variable can be assigned a value whereas an array variable cannot be.
	int a[10];
	int *p;
	p=a; /*legal*/
	a=p; /*illegal*/
	5. Arithmetic on pointer variable is allowed.
	p++; /*Legal*/
	a++; /*illegal*/
	6. Array is a collection of similar data types while the pointer variable stores the address of another variable.
		3	CO1	L1

2(b)	Define Dynamic Memory Allocation. List and write with explanation the syntax of dynamic memory allocating functions **SOLUTION**	5	CO1	L1

Dynamic memory Allocation can be defined as a procedure in which the size of a data structure can be changed during run time.

There are 4 library functions provided by C.

- It is defined under <stdlib.h>

The functions are:

1. malloc()
2. calloc()
3. free()
4. realloc()

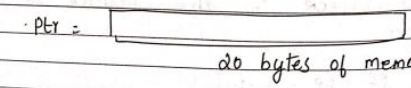
malloc():

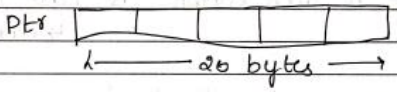
- The malloc() or memory allocation method in C is used to dynamically allocate a single large block of memory with specified size.

- It returns a pointer of type void which can be cast into any form.

Syntax: ptr = (cast-type*) malloc (byte-size)

Example: Ptr = (int*) malloc (5 * sizeof(int));



	2. Calloc() method:
	- contiguous allocation method is used to dynamically allocate the specified number of blocks of memory of specified type.
	Syntax: ptr = (cast-type*) calloc (n, elementSize);
	Example: Ptr = (float*) calloc (25, sizeof (float));
	Ptr = (int*) Calloc (5, sizeof (int));
	 
	3. free() method:
	- free method in C is used to dynamically de-allocate the memory.
	4. realloc():
	reallocation method used in C is used to dynamically change the memory of previously allocated memory.
	Syntax: ptr = realloc (ptr, newSize);
	int ptr = (*int) malloc (5 * size of (int));
	ptr = realloc (ptr, 10 * size of (int));			

2(c)	Is it possible to access the contents of an array using a pointer? Justify your opinion with suitable C Statements **SOLUTION** Yes, It is possible to access the contents of an array using a pointer. Ex: A[5] = {1,2,3,4,5}; Int *ptr; Ptr = A; Ptr+1 //Refers Second Element Ptr+4 //Refers last element.	2	CO1	L3
3(a).	Define Sparse Matrix. Represent the Sparse Matrix as follows: a. Triplet Array b. Fast Transpose $\begin{pmatrix} 7 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix}$	7	CO1	L2

SOLUTION

Sparse matrix is one which has a large number of zero elements.

Eg)
$$\begin{bmatrix} 15 & 0 & 0 & 22 \\ 4 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 4 & 2 & 0 & 0 \end{bmatrix}$$

6 - non-zero element
10 - no zero "

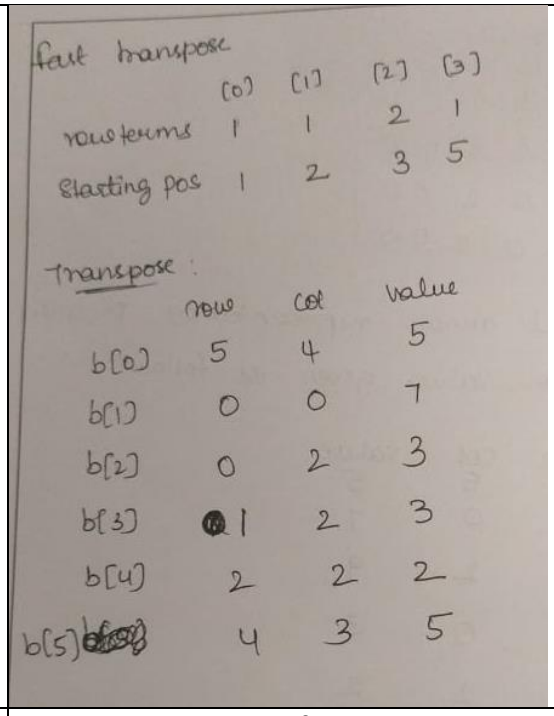
Operations performed on sparse matrix:
matrix creation, addition,
multiplication and transpose.

Given matrix is

	[0]	[1]	[2]	[3]	[4]
[0]	7	0	0	0	0
[1]	0	0	9	0	0
[2]	3	0	2	0	0
[3]	0	0	0	0	5

The triplet array representation includes row, col, values given as follows:

	row	col	value
a[0]	4	5	5
a[1]	0	0	7
a[2]	1	2	9
a[3]	2	0	3
a[4]	2	2	2
a[5]	3	4	5



fast transpose

	[0]	[1]	[2]	[3]
row terms	1	1	2	1
starting pos	1	2	3	5

transpose :

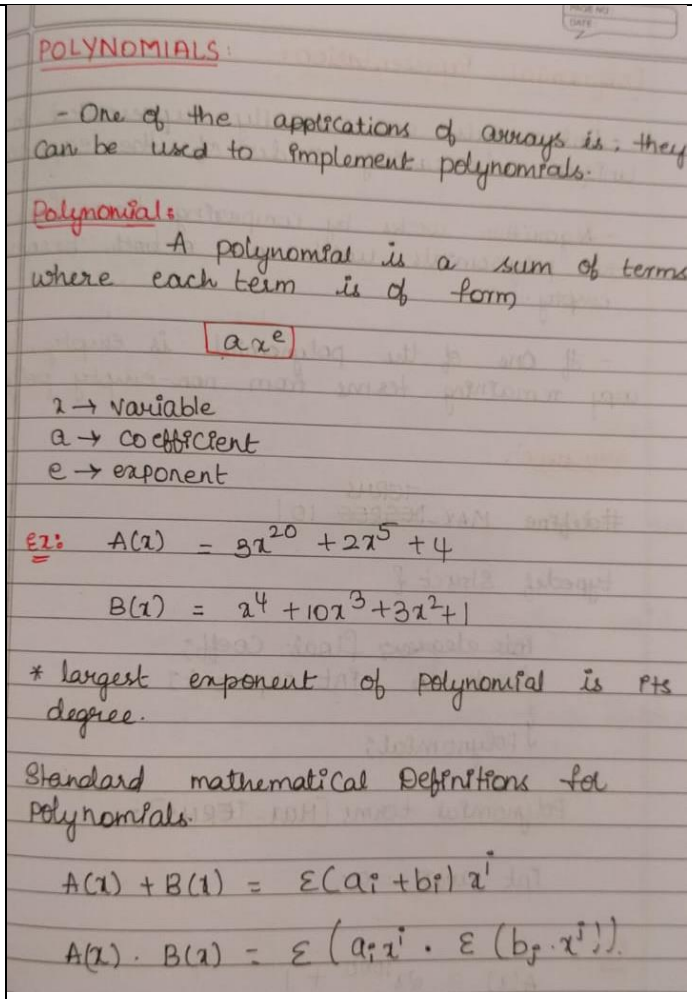
	row	col	value
b[0]	5	4	5
b[1]	0	0	7
b[2]	0	2	3
b[3]	1	2	3
b[4]	2	2	2
b[5]	4	3	5

3(b)	Write a C Function to find the Fast Transpose of the Sparse Matrix **SOLUTION**	3	CO1	L2

```
void fast_transpose(term a[], term b[])
{
/* the transpose of a is placed in b */
   int row_terms[MAX_COL], starting_pos[MAX_COL];
   int i,j, num_cols = a[0].col, num_terms = a[0].value;
   b[0].row = num_cols;  b[0].col = a[0].row;
   b[0].value = num_terms;
   if (num_terms > 0) { /* nonzero matrix */
      for (i = 0; i < num_cols; i++)
         row_terms[i] = 0;
      for (i = 1; i <= num_terms; i++)
         row_terms[a[i].col]++;
      starting_pos[0] = 1;
      for (i = 1; i < num_cols; i++)
         starting_pos[i] =
                     starting_pos[i-1] + row_terms[i-1];
      for (i = 1; i <= num_terms; i++) {
         j = starting_pos[a[i].col]++;
         b[j].row = a[i].col;   b[j].col = a[i].row;
         b[j].value = a[i].value;
      }
   }
}
```

Program 2.8: Fast transpose of a sparse matrix

4(a)	Define a polynomial. Represent the polynomials, $A(x) = 3x^{23} + 3x^4 + 4x^2 + 15$ and $B(x) = x^5+20x^3+2$ using array of Structures. **SOLUTION**	2	CO1	L2



POLYNOMIALS:

- One of the applications of arrays is: they can be used to implement polynomials.

Polynomial:

A polynomial is a sum of terms where each term is of form

ax^e

x → variable
a → coefficient
e → exponent

Ex: $A(x) = 3x^{20} + 2x^5 + 4$

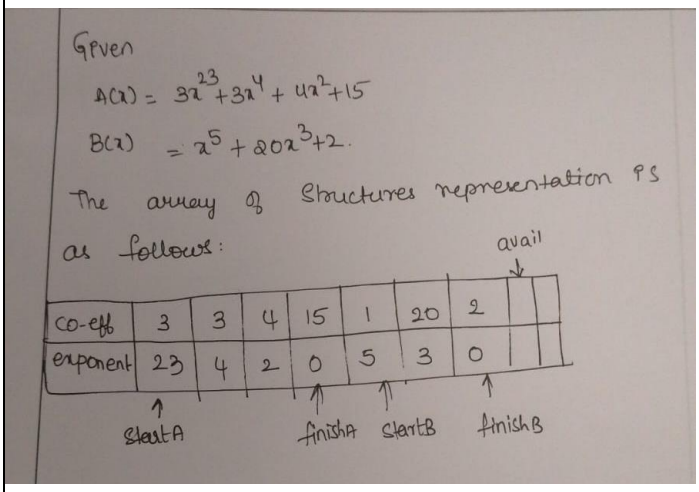
$B(x) = x^4 + 10x^3 + 3x^2 + 1$

* largest exponent of polynomial is its degree.

Standard mathematical Definitions for Polynomials.

$A(x) + B(x) = \Sigma(a_i + b_i)x^i$

$A(x) \cdot B(x) = \Sigma(a_i x^i \cdot \Sigma(b_j \cdot x^j))$.



Given

$A(x) = 3x^{23} + 3x^4 + 4x^2 + 15$

$B(x) = x^5 + 20x^3 + 2$.

The array of structures representation is as follows:

								avail	
co-eff	3	3	4	15	1	20	2		
exponent	23	4	2	0	5	3	0		
	↑			↑	↑		↑		
	startA			finishA	startB		finishB		

4(b)	Define Array. Give Abstract Data Type (ADT) for arrays. **SOLUTION** An array is a collection of items of same data type stored at contiguous memory locations.	3	CO1	L2

structure *Array* is

objects: A set of pairs <*index*, *value*> where for each value of *index* there is a value from the set *item*. *Index* is a finite ordered set of one or more dimensions, for example, {0, ··· , *n*–1} for one dimension, {(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)} for two dimensions, etc.

functions:

for all $A \in Array$, $i \in index$, $x \in item$, j, $size \in$ integer

Array Create(*j*, *list*)	::=	**return** an array of *j* dimensions where *list* is a *j*-tuple whose *i*th element is the the size of the *i*th dimension. *Items* are undefined.
Item Retrieve(*A*, *i*)	::=	**if** ($i \in index$) **return** the item associated with index value *i* in array *A* **else return** error
Array Store(*A*,*i*,*x*)	::=	**if** (*i* in *index*) **return** an array that is identical to array *A* except the new pair <*i*, *x*> has been inserted **else return** error.

end *Array*

Structure 2.1: Abstract Data Type *Array*

4(c)	Write a Menu Driven C Program for the following operations on STACK of integers: a. Push an element on to the stack b. Pop an element from the stack c. Display the contents of the stack d. Exit	5	CO2	L3

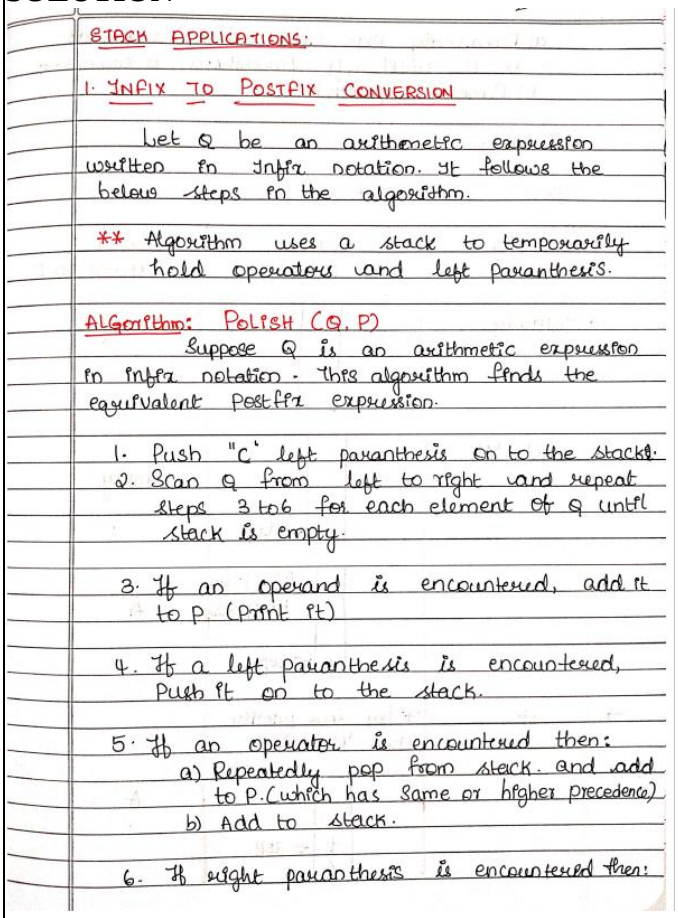
SOLUTION

```
#include<stdio.h>
#define MAX 4

int stack[MAX], item;
int top = -1, status = 0;

void push(int item)
{
if (top == (MAX-1))
printf("\n\nStack is Overflow");
else
  {
        stack[++top] = item;
        status++;
  }
}

int pop()
{
int ret=0;
if(top == -1)
        printf("\n\nStack is Underflow");
else
{
        ret = stack[top--];
        status--;
        printf("\nPopped element is %d", ret);
```

```
}
return ret;
}


void display()
{
int i;
printf("\nThe stack contents are:");
for(i=top; i>=0; i--)
printf("\n ------\n| %d |", stack[i]);
printf("\n");
}

int main()
{
int choice;
while(1)
{
printf("\n");
printf("--------STACK  IMPLEMENTATION----------");
printf("\n1.PUSH 2.POP 3..Exit");
printf("\n----------------------------------------");
printf("\nEnter your choice:\t");
scanf("%d",&choice);
switch(choice)
{
        case 1:
                printf("\nEnter a single digit element to be pushed: ");
                scanf("%d", &item);
                push(item);
                display();
                break;
        case 2:
                        item=pop();
                        if(item !=0)
                        display();
                        break;
case 3: printf("You Exited the program\n\n");
     exit (0);
     break;
default:
        return 0;

}
}
return 0;
}
```

5 (a)	Write an algorithm to convert Infix expression to Postfix Expression.	4	CO2	L2

SOLUTION



STACK APPLICATIONS:

1. INFIX TO POSTFIX CONVERSION

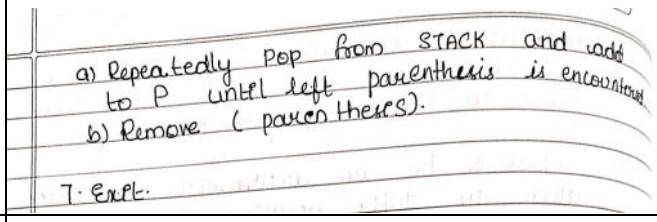
Let Q be an arithmetic expression written in infix notation. It follows the below steps in the algorithm.

** Algorithm uses a stack to temporarily hold operators and left paranthesis.

ALGorithm: POLISH (Q, P)

Suppose Q is an arithmetic expression in infix notation. This algorithm finds the equivalent postfix expression.

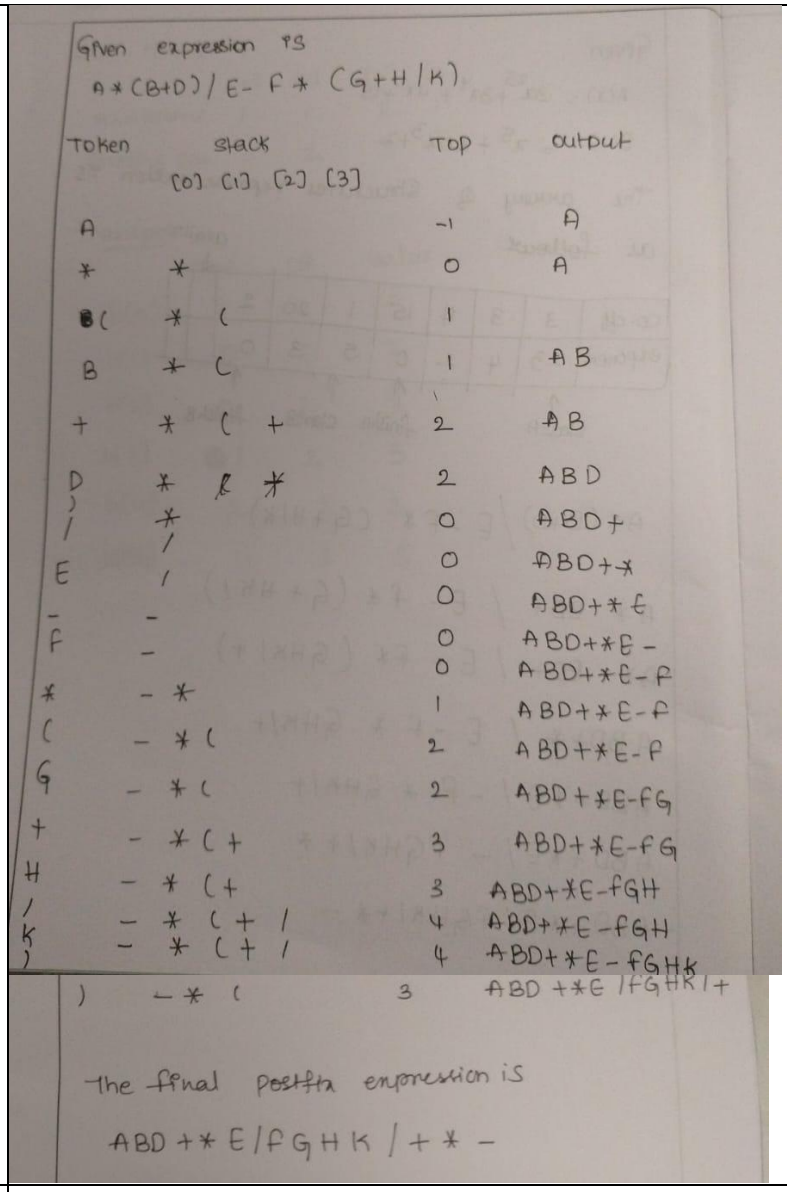
1. Push "(" left paranthesis on to the stack.
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until stack is empty.
3. If an operand is encountered, add it to P. (Print it)
4. If a left paranthesis is encountered, Push it on to the stack.
5. If an operator is encountered then:
   a) Repeatedly pop from stack. and add to P. (which has same or higher precedence)
   b) Add to stack.
6. If right paranthesis is encountered then:



a) Repeatedly POP from STACK and add to P until left parenthesis is encountered
b) Remove (paranthesis).

7. Exit.

5 (b)	Consider the expression A * (B + D) / E – F * (G + H / K). Convert the given infix expression to postfix expression using Stacks. **SOLUTION**	6	CO2	L3



Given expression is

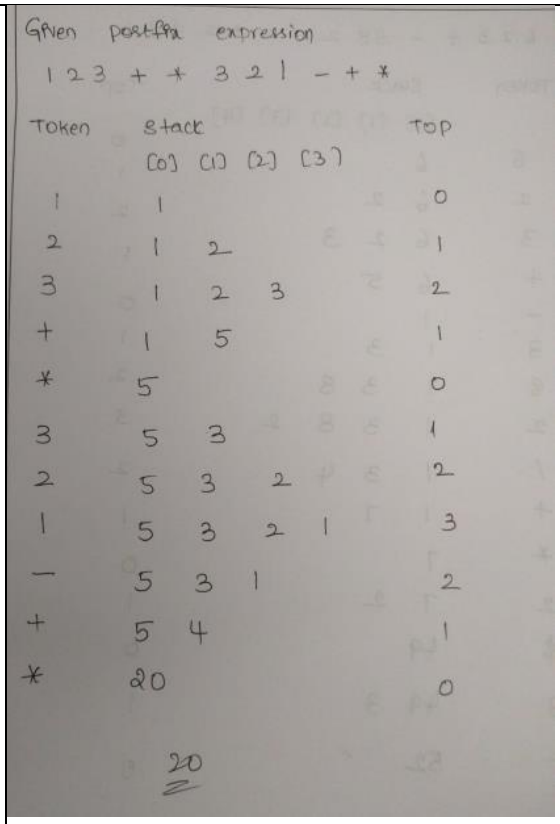
A * (B+D) / E - F * (G+H/K)

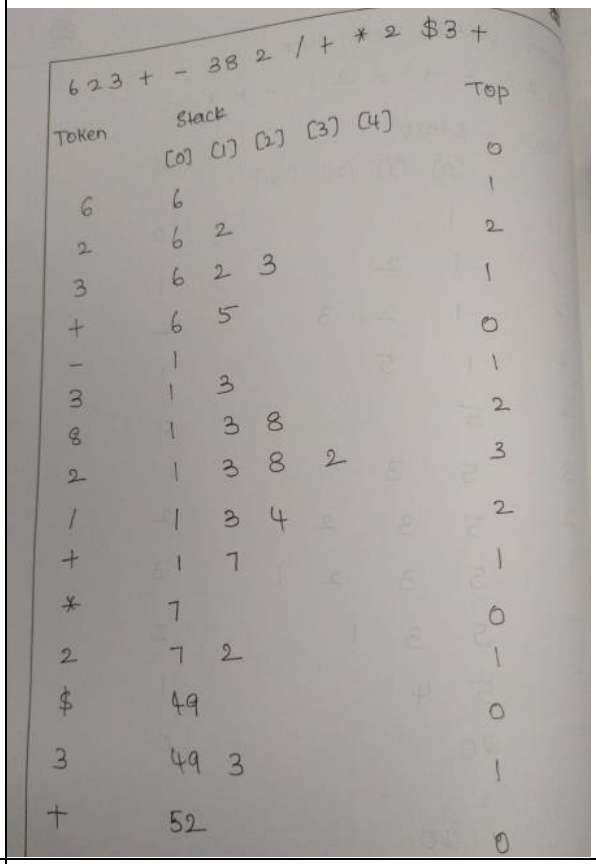
Token	stack [0] [1] [2] [3]	Top	output
A		-1	A
*	*	0	A
(	* (	1	
B	* (	1	AB
+	* (+	2	AB
D	* (+	2	ABD
)	*	0	ABD+
/	/	0	ABD+*
E	/	0	ABD+*E
-	-	0	ABD+*E/
F	-	0	ABD+*E/F
*	- *	1	ABD+*E/F-
(	- * (	2	ABD+*E/F-
G	- * (	2	ABD+*E/FG-
+	- * (+	3	ABD+*E/FG-
H	- * (+	3	ABD+*E/FGH-
/	- * (+ /	4	ABD+*E/FGH-
K	- * (+ /	4	ABD+*E/FGHK-
)	- * (	3	ABD+*E/FGHK/+-

The final postfix expression is

ABD+*E/FGHK/+*-

6(a)	Evaluate the following postfix Expressions using stack. a. 1 2 3 + * 3 2 1 - + * b. 6 2 3 + - 3 8 2 / + * 2 $ 3 + **SOLUTION**	6	CO2	L3





6(b)	Write a C Program to find a pattern in a given string and replace it with a replace string. **SOLUTION**	4	CO1	L3

```
#include<stdio.h>
char str[100], pat[50], rep[50], ans[100];
int i, j, c, m, k, flag=0;

void stringmatch()
{
i = m = c = j = 0;
while(str[c]!='\0')
{
if(str[m]==pat[i])
{
i++; m++;
   if(pat[i]=='\0')
   {
   flag = 1;
   for(k = 0; rep[k] != '\0'; k++, j++)
   ans[j] = rep[k];
   i = 0;c = m;
   }
}
else
   {
   ans[j] = str[c];
   j++;c++; m = c;i = 0;
   }
}
ans[j] = '\0';
}

int main()
{
printf("\nEnter a main string \n");
gets(str);
printf("\nEnter a pattern string \n");
gets(pat);
printf("\nEnter a replace string \n");
gets(rep);
stringmatch();
if(flag==1)
printf("\nThe resultant string is\n %s" , ans);
else
printf("\nPattern string NOT found\n");
return 0;
}
#include<stdio.h>
char str[100], pat[50], rep[50], ans[100];
int i, j, c, m, k, flag=0;

void stringmatch()
{
```

```
i = m = c = j = 0;
while(str[c]!='\0')
{
if(str[m]==pat[i])
{
i++; m++;
   if(pat[i]=='\0')
   {
   flag = 1;
   for(k = 0; rep[k] != '\0'; k++, j++)
   ans[j] = rep[k];
   i = 0;c = m;
   }
}
else
   {
   ans[j] = str[c];
   j++;c++; m = c;i = 0;
   }
}
ans[j] = '\0';
}

int main()
{
printf("\nEnter a main string \n");
gets(str);
printf("\nEnter a pattern string \n");
gets(pat);
printf("\nEnter a replace string \n");
gets(rep);
stringmatch();
if(flag==1)
printf("\nThe resultant string is\n %s" , ans);
else
printf("\nPattern string NOT found\n");
return 0;
}
```

CI **CCI** **HOD**

PO Mapping

CO-PO and CO-PSO Mapping																				
Course Outcomes		Blooms Level	Modules covered	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	Analyze the performance of the algorithms, state the efficiency using asymptotic notations, and analyze mathematically the complexity of the algorithm.	L2	M1	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO2	Apply divide and conquer approaches and decrease and conquer approaches in solving the problems and analyze the same	L3	M2	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO3	Apply the appropriate algorithmic design technique like the greedy method, transform and conquer approaches and compare the efficiency of algorithms to solve the given problem.	L3	M3	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO4	Apply and analyze dynamic programming approaches to solve some problems. and improve an algorithm's time efficiency by sacrificing space.	L3	M4	3	3	2	3	2	-	-	-	-	-	-	2	-	-	-	-
CO5	Apply and analyze backtracking, branch and bound methods to describe P, NP, and NP-complete problems.	L3	M5	3	2	2	3	2	-	-	-	-	-	-	2	-	-	-	-

COGNITIVE LEVEL	REVISED BLOOMS TAXONOMY KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PROGRAM OUTCOMES (PO), PROGRAM SPECIFIC OUTCOMES (PSO)				CORRELATION LEVELS	
PO1	Engineering knowledge	PO7	Environment and sustainability	0	No Correlation
PO2	Problem analysis	PO8	Ethics	1	Slight/Low
PO3	Design/development of solutions	PO9	Individual and team work	2	Moderate/ Medium
PO4	Conduct investigations of complex problems	PO10	Communication	3	Substantial/ High
PO5	Modern tool usage	PO11	Project management and finance		
PO6	The Engineer and society	PO12	Life-long learning		
PSO1	Develop applications using different stacks of web and programming technologies				
PSO2	Design and develop secure, parallel, distributed, networked, and digital systems				
PSO3	Apply software engineering methods to design, develop, test and manage software systems.				
PSO4	Develop intelligent applications for business and industry				