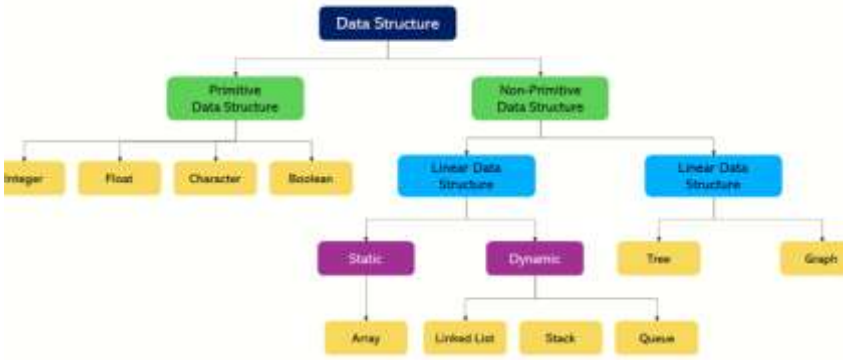
USN										

Internal Assessment Test 1 – December 2023 (Set-1)

Sub:	Data Structures and Applications					Sub Code:	BCS304	Branch:	ISE	
Date:	19/12/2023	Duration:	90 min's	Max Marks:	50	Sem / Sec:	III / A, B & C		OBE	

Answer any FIVE FULL Questions

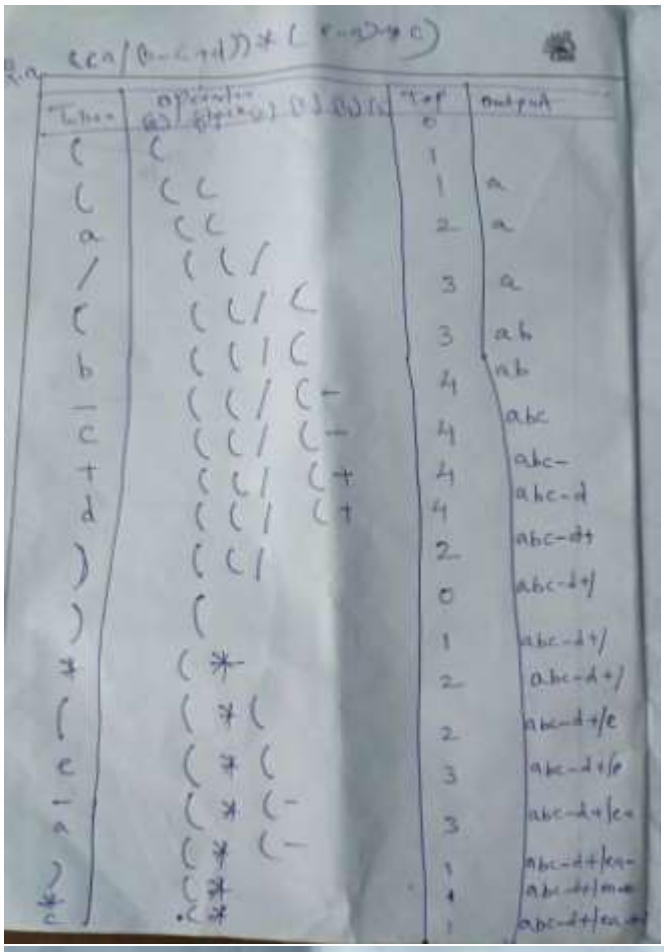
		MARKS	CO	RBT
1 (a)	Define Data Structures. Explain the different types of data structures with examples and list the operations that can be performed. **Solution:** A data structure is a specialized format for organizing, processing, retrieving and storing data. We can classify Data Structures into two categories: 1. Primitive Data Structure 2. Non-Primitive Data Structure  different types of operations that we can perform to manipulate data in every data structure: 1. **Traversal:** Traversing a data structure means accessing each data element exactly once so it can be administered. 2. **Search:** Search is another data structure operation which means to find the location of one or more data elements that meet certain constraints. 3. **Insertion:** Insertion means inserting or adding new data elements to the collection. 4. **Deletion:** Deletion means to remove or delete a specific data element from the given list of data elements. 5. **Sorting:** Sorting means to arrange the data elements in either Ascending or Descending order depending on the type of application. 6. **Merge:** Merge means to combine data elements of two sorted lists in order to form a single list of sorted data elements. 7. **Create:** Create is an operation used to reserve memory for the data elements of the program. 8. **Selection:** Selection means selecting a particular data from the available data.	[04]	CO1	L1

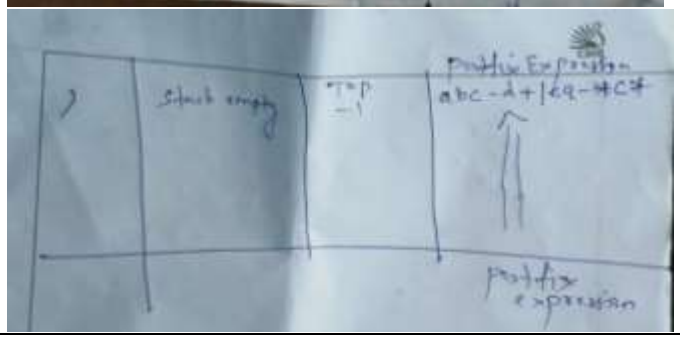
	9. **Update:** The Update operation allows us to update or modify the data in the data structure. 10. **Splitting:** The Splitting operation allows us to divide data into various subparts decreasing the overall process completion time.			

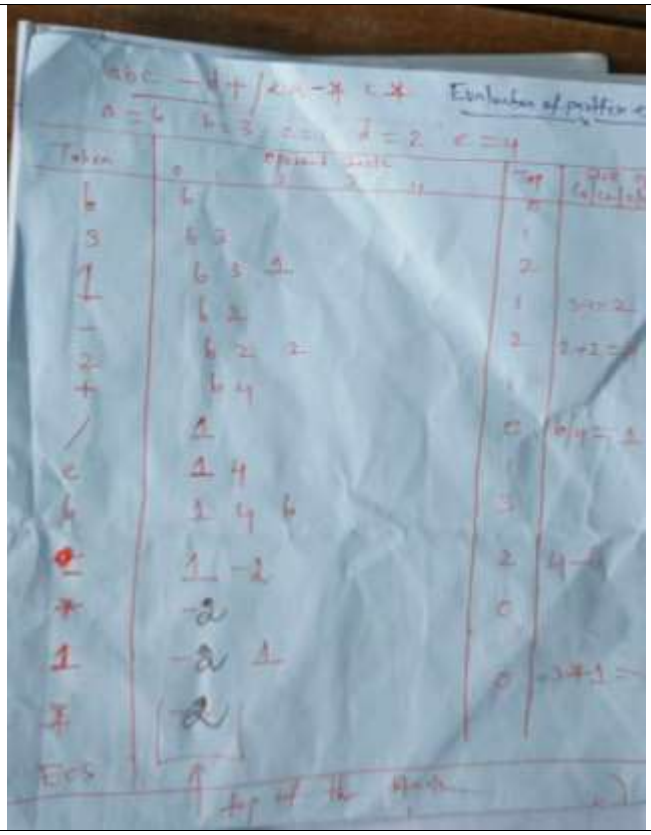
(b)	Write a C function to perform the following operations without using global variables:
	c. Inserting an Element (ELEM) at a given valid Position (POS)
	d. Deleting an Element at a given valid Position (POS)
	Solution:
	#include<stdio.h>
	/* Function definition to insert an element at a given valid position. */
	void insert(int a[], int n)
	{
	int ele, pos, i;
	printf("\nEnter the element to be inserted: ");
	scanf("%d", &ele);
	printf("\nEnter the position at which you want to insert: ");
	scanf("%d", &pos);
	if(pos<1 || pos>n)
	{
	printf("Invalid Position!!!");
	}
	else
	{
	for(i = n-1 ; i >= pos-1 ; i--)
	{
	a[i+1] = a[i]; /* Copy the element in ith location to i+1th location. */
	}
	a[pos-1] = ele; /* Insert the element. */
	n++; /* Increment the count of elements in the array. */
	}
	}
	/* Function definition to delete an element at a given valid position. */
	void delete(int a[], int n)
	{
	int pos, i;
	printf("\nEnter the position from which you want to delete: ");
	scanf("%d", &pos);
	if(pos<1 || pos>n)
	{
	printf("Invalid Position!!!");
	}
	else
	{
	printf("\nThe deleted element is: %d", a[pos-1]);
	for(i = pos-1 ; i < n-1 ; i++)
	{
	a[i] = a[i+1]; /* Copy the element in i+1th location to ith location. */
	}
	n--; /* Decrement the count of elements in the array. */
	}
	}
		[06]	CO3	L3

2 (a)	Convert the infix expression ((a/(b-c+d))*(e-a)*c) to postfix expression and evaluate that postfix expression for given data a=6, b=3, c=1, d=2, e=4 (using stack representation).	[07]	CO3	L3

Solution:







(b)	What is a pointer? How do you declare and initialize a pointer? How do you access the value pointed to by a pointer.
	Solution:
	A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.
	pointer declaration.
	datatype **** ptr***;
	Example
	int ***ptr**;
	The pointer declared here will point to some random memory address as it is not initialized.
	Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the **(&) addressof operator** to get the memory address of a variable and then store it in the pointer variable.
	Example
	int var = 10;
	int * **ptr**;
	ptr = **&**var;
	Pointer Dereferencing
	Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer.
	
	Ex:
	int var = 10;
	
	// declare pointer variable
	int* ptr;
	
	// note that data type of ptr and var must be same
	ptr = &var;
	
	// assign the address of a variable to a pointer
		[03]	CO1	L1

```
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
```

3 What is dynamic memory allocation?. Write a C program to illustrate the same for allocating memory to store n integers and find the sum, mean, variance and standard deviation using dynamic memory allocation. [10] CO1 L2

Soution:

```
#include <stdio.h>
#include <math.h>
#define MAXSIZE 10

void main()
{
   float *x;
   int  i, n;
   float average, variance, std_deviation, sum = 0, sum1 = 0;
   printf("Enter the value of N \n");
   scanf("%d", &n);
x=(float*)malloc(n*sizeof(float));
   printf("Enter %d real numbers \n", n);
   for (i = 0; i < n; i++)
   {
   scanf("%f", &x[i]);
   }
   /*  Compute the sum of all elements */
   for (i = 0; i < n; i++)
   {
      sum = sum + x[i];
   }
   average = sum / (float)n;
   /*  Compute  variance  and standard deviation  */
   for (i = 0; i < n; i++)
   {
      sum1 = sum1 + pow((x[i] - average), 2);
   }
   variance = sum1 / (float)n;
   std_deviation = sqrt(variance);
   printf("Average of all elements = %.2f\n", average);
   printf("variance of all elements = %.2f\n", variance);
   printf("Standard deviation = %.2f\n", std_deviation);
}
```

4 (a) Differentiate between structure and unions. [06] CO1 L1

Solution:

		STRUCTURE	UNION		
	Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.		
	Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**		
	Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.		
	Value Altering	Altering the value of a member will not affed other members of the structure.	Altering the value of any of the member will alter other member values.		
	Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.		
	Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.		

(b)	What is the output of the following code?
	int num[5] = { 3, 4, 6, 2, 1 };
	int *p = num;
	int *q = num + 2;
	int *r = &num[1];
	printf("%d %d", num[2], *(num+2));
	printf("%d %d", *p, *(p+1));
	printf("%d %d", *q, *(q+1));)
	printf("%d %d", *r, *(r+1));)
	output:
	6 6
	3 4
	Syntax Error
	Syntax Error
				[04]	CO1	L3

5	Define stack. Explain different operations that can be performed on the stack with a C program. Also mention the applications.
	Solution:
	#include<stdio.h>
	#define MAX 5
	/*Initially top of the stack will be -1.*/
	int top = -1; /*Points to top of the stack.*/
	int s [MAX - 1];
	/*Function definition to push elements onto stack.*/
	void push (int elem)
	{
	if (top == MAX - 1)
	{
	printf ("\nStack Overflow!!!");
	}
	else
	{
	top = top + 1; /*Increment top of the stack.*/
	s [top] = elem; /*Insert element on top of the stack.*/
	}
	}
	/*Function definition to delete elements from stack.*/
	int pop ( )
	{
	int del;
	if (top == -1)
	{
	printf ("\nStack Underflow!!!");
	return (-1);
	}
	else
				[10]	CO2	L2

```
{
del = s [ top ]; /*Delete the topmost element from the stack.*/
top = top - 1; /*Decrement top of the stack.*/
return ( del );
}
}
/*Function definition to check whether a given number is palindrome or not.*/
int palindrome ( )
{
int n, i, p [ 10 ], flag = 0;
printf ( "\nEnter the number of digits: " );
scanf ( "%d", &n );
printf ( "\nEnter the digits: " );
for ( i = 0; i < n ; i++ )
scanf ( "%d", &p [ i ] );
/*Push all the digits onto the stack.*/
for ( i = 0 ; i < n ; i++ )
push ( p [ i ] );
for ( i = 0 ; i < n ; i++ )
{
/*check if each digit in the array p and the popped element from the stack are equal.*/
if ( p [ i ] != pop ( ) )
{
flag = 1; /*Not equal break.*/
break;
}
}
if ( flag == 0 )
printf ( "\nPALINDROME!!!" );
else
printf ( "\nNOT PALINDROME!!!" );
}
/*Function definition for displaying stack elements.*/
void display ( )
{
int i;
if ( top == -1 )
{
printf ( "\nStack is Empty!!!" );
}
else
{
printf ( "\nThe elements of stack are: " );
for ( i = top ; i >= 0 ; i-- )
printf ( "\t%d", s [ i ] );
}
}
```

6 (a)	Write a C program to add matrices using functions in C by passing 2D array as an argument.
	Solution:
	`#include<stdio.h>`
	`void accept(int x[3][3]);`
	`void Display(int x[3][3]);`
	`void Addmatrix(int x[3][3], int y[3][3]);`
	`void accept(int x[3][3])`
	`{`
	`int i,j;`
	`for(i=0;i<3;i++)`
		[06]	CO3	L3

```
{
for(j=0;j<3;j++)
{
scanf("%d",&x[i][j]);
}
}
}
void Display(int x[3][3])
{
int i,j;
printf("the elements of the matrix  \n");
for(i=0;i<3;i++)
  {
for(j=0;j<3;j++)
    {
printf("%d\t ",x[i][j]);
    }
 printf("\n");
  }
}
void Addmatrix(int x[3][3], int y[3][3])
{
int i,j;
int z[3][3];
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
   z[i][j]=x[i][j]+y[i][j];
}
}
Display(z);
}
int main()
{
int a[3][3],b[3][3];
printf("Enter the elements of matrix A");
accept(a);
printf("Enter the elements of matrix B");
accept(b);
printf("the elements of matrix A");
Display(a);
printf("the elements of matrix B");
Display(b);
printf("The resultant matric is\n");
Addmatrix(a,b);
return 0;
}
```

(b)	What is a postfix expression? What is the need for evaluation of a postfix expression? Write a C program for the evaluation of postfix expression using stack. Soution: • ***Postfix notation*** is also known as ***reverse polish notation***. It is a mathematical notation in which operators come after their operands. • Postfix notation has several advantages over infix notation. • It eliminates the need for parentheses and makes parsing and evaluation of expressions easier.	[04]	CO1	L2

Code:

```
#include <stdio.h>
#include <ctype.h>
#define MAXSTACK 100 /* for max size of stack */
#define POSTFIXSIZE 100 /* define max number of charcters in postfix expression */
int stack[MAXSTACK];
int top = -1;
void push(int item)
{

   if (top >= MAXSTACK - 1) {
      printf("stack over flow");
      return;
   }
   else {
      top = top + 1;
```

splay trees implementation.docx

```
      stack[top] = item;
   }
}

/* define pop operation */
int pop()
{
   int item;
   if (top < 0) {
      printf("stack under flow");
   }
   else {
      item = stack[top];
      top = top - 1;
      return item;
   }
}

/* define function that is used to input postfix expression and to evaluate it */
void EvalPostfix(char postfix[])
{

   int i;
   char ch;
   int val;
   int A, B;

   /* evaluate postfix expression */
   for (i = 0; postfix[i] != ')'; i++) {
      ch = postfix[i];
      if (isdigit(ch)) {
         /* we saw an operand,push the digit onto stack
ch - '0' is used for getting digit rather than ASCII code of digit */
         push(ch - '0');
      }
      else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
         /* we saw an operator
* pop top element A and next-to-top elemnet B
```

```
* from stack and compute B operator A
*/
        A = pop();
        B = pop();

        switch (ch) /* ch is an operator */
        {
        case '*':
           val = B * A;
           break;

        case '/':
           val = B / A;
           break;

        case '+':
           val = B + A;
           break;

        case '-':
           val = B - A;
           break;
        }

        /* push the value obtained above onto the stack */
        push(val);
      }
   }
   printf(" \n Result of expression evaluation : %d \n", pop());
}
```

Faculty CCI HoD