CMR
INSTITUTE OF
TECHNOLOGY

Internal Assessment Test 2 – April 2018-SCHEME & SOLUTION

Sub:	Programming in C and Data Structures					**Code:**	17PCD23
Date:	19/ 03/2018	Duration:	90mins	Max Marks:	50	**Sem:** II	**Branch:** ALL

Note: Answer any five questions:

		Marks	CO	RBT
1.(a)	**What is array? How to initialize single dimension and two dimensional arrays.**	05M	CO 2	L2

• Array is a collection of elements of same data type. **(2M)**
• The elements are stored sequentially one after the other in memory.
• Any element can be accessed by using
→ name of the array
→ position of element in the array
• Arrays are of 2 types:
1) Single dimensional array
2) Multi dimensional array

Single DIMENSIONAL ARRAY(1.5M)

• A single dimensional array is a collection of elements of same data type.
.

Declaration of Single Dimensional Arrays(0.5M)

• The syntax is shown below:
data_type array_name[size];
where data_type can be int, float or char
array name is name of the array
size indicates number of rows in the array
• For ex:
int matrix[3];

Initialization of Single Dimensional Arrays(1M)

For ex:
int matrix[3]={2,4,4};
or

```
for(i=0;i<3i++)
{
            scanf("%d",&matrix[i]);

}
```

2 DIMENSIONAL ARRAY(1.5M)

• A 2 dimensional array is a matrix consisting of related elements of same type.
• In memory, all the elements are stored in continuous memory-location one after the other.

Declaration of 2 Dimensional Arrays(0.5M)

• The syntax is shown below:
data_type array_name[row_size][Col_size];
where data_type can be int, float or char
array name is name of the array
row_size indicates number of rows and col_size indicates number of cols in the array
• For ex:
int matrix[3][3];

Initialization of 2 Dimensional Arrays(1M)

For ex:
int matrix[3][3]={{2,4,4},{1,1,1},{2,2,2}};
or

```
for(i=0;i<3i++)
{
      for(j=0; j<3; j++)
      {
            scanf(“%d”,&matrix[i][j]);
        }
}
```

1.(b)	**Write a program to evaluate polynomial f(x) = a4*x*4 + a3*x*3 + a2*x*2 + a1*x*+ a0, for a given value of *x* and its Coefficients using Horner’s method. (5M)**	**05M**	CO 3	L3

```
 /* Program to evaluate a polynomial using Horner's method. */
#include<stdio.h>
int main()
{
   int order, i;
   float a [ 15 ], x, sum;
  // Input the order of polynomial.
   printf ( "\nEnter the order of polynomial: " );
   scanf ( "%d", &order );
  // Input the coefficients starting from lowest order.
   printf ( "\nEnter %d co-efficients of polynomial, starting with lowest order
coefficient first:\n", (order+1) );
    for ( i = 0 ; i <= order ; i++ )
   {
            scanf ( "%f", &a [ i ] );
   }
  // Input the value of x.
    printf ( "\nEnter the value of x: " );
    scanf ( "%f", &x );
  // Add the constant a (a0) to the sum.
    sum = a [ order ];
  // Compute sum using Horner's method.
   for ( i = order-1 ; i >= 0 ; i-- )
  {
      sum = ( sum * x ) + a [ i ] ;
   }
  // Display the sum of the given polynomial.
    printf ( "\nThe value of polynomial f(%f): %f\n\n", x, sum );
   return 0;
}
```

2	**Explain how strings are declared and initialized. Explain any 4 string manipulation functions with example.**
	Declaring a string- (1M)
	char str_name[size];
	• Str_name is the string name and follows the rules of naming an identifier.
	• Size is the no. of characters in the string.
	Ex: char a[10];
	
	Initialization of string (1M)
	• **gets(str_name);**
	Ex: gets(a);
	Or
	• **scanf("%s",str_name);**
	Ex: scanf("%s",a);
	
	<u>String manipulation functions: (2M PER FUNCTION: 2X4=8)</u>
	strlen()
	• This function calculates the length of string. It takes only one argument, i.e., string-name.
	• The syntax is shown below:
	temp_variable = strlen(string_name);
	
	strcpy()
	• This function copies the content of one string to the content of another string. It takes 2 arguments.
	• The syntax is shown below:
	strcpy(destination,source);
	where source and destination are both the name of the string.
	
	strcat()
	• This function joins 2 strings. It takes two arguments, i.e., 2 strings and resultant string is stored in
	the first string specified in the argument.
	• The syntax is shown below:
	strcat(first_string,second_string);
	
	strcmp()
	• This function compares 2 string and returns value 0, if the 2 strings are equal. It takes 2 arguments,
	i.e., name of two string to compare.
	• The syntax is shown below:
	temp_varaible=strcmp(string1,string2);
	• If string1 is greater than string 2: +1 is returned by the function.
	• If string1 is lesser than string 2: -1 is returned by the function.
		10M	CO 3	L2

3.(a)	**What is a function? Explain function prototype, function call and function definition using programming example.** A **function** is a group of statements that together perform a task. Every **C** program has at least one **function**, which is main(), and all the most trivial programs can define additional **functions**.**(1M)** **FUNCTION DECLARATION//PROTOTYPE- (1.5M)** Syntax: <return type> function_name(Parameter list….); Parameter list contains <data type> var_name separately for every parameter. **FUNCTION CALL: (1M)** Syntax: function_name(parameters) Function call comes inside the main or any sub function. **FUNCTION DEFINITION: (1.5M)** Syntax: <return type> function_name(Parameter list….) { **------------------------------** **------------------------------** **------------------------------** **}**	**05M**	CO 3	L2

3.(b)	**What is a parameter? Explain Actual and formal parameters with example.**
	A parameter is the inputs given to a particular function. **(1M)**
	There are 2 types of parameters:
	1. Actual parameters **(2M)**
	2. Formal parameters **(2M)**
	Actual parameters: The parameters that appear in the function call are called actual parameter. These are the original data.
	Ex: fact(n);--→ n is the actual parameter sent from calling function through the function call
	Formal parameters: The parameters that appear in the function definition are called formal parameter. These are the copied data.
	Ex: int fact(int n)
	{
	------------ 	
	}
	Here int n receives the data from the function call as formal parameter.
		05M	CO 3	L1

4.(a)	**Write a program to perform bubble sort to arrange elements in ascending order using following function. void sort(int n, int a[]); (10M)**	**10M**	Co 3	L3

```
/* Program to perform bubble sort to arrange elements in ascending order*/
#include<stdio.h>
void sort(int n,int a[]);
int main()
{
int a [ 25 ], i,n;
// Input the number of integers that you want to sort.
printf ( "\nEnter the number of integers to be sorted: " );
scanf ( "%d" , &n);
// Input the numbers.
printf ( "\nEnter the numbers:\n" );
for ( i = 0 ; i < n ; i++ )
{
scanf ( "%d" , &a [ i ] );
}
sort(n,a);
return 0;
}
void sort(int n,int a[])
{
  int i, j,  temp;
for ( i = 0 ; i < n - 1 ; i++ ) // Passes.
{
for ( j = 0 ; j < n - 1 - i ; j++ ) // Comparisions.
{
if ( a [ j ] > a [ j + 1 ] ) // If jth element is greater than j+1th element then swap.
{
temp = a [ j ];
a [ j ] = a [ j + 1 ];
a [ j + 1 ] = temp;
}
}
}
// Display sorted array in ascending order.
printf ( "\n\nThe sorted array using Bubble Sort is:\n" );
for ( i = 0 ; i < n ; i++ )
{
printf ( "%d\t", a [ i ] );
}
printf ( "\n" );
}
```

5.(a)	**Explain call by value and call by reference parameter passing mechanism for swapping of two numbers.**	**10M**	Co 3	L2

CALL BY VALUE (5M)

• In this type, value of actual arguments are passed to the formal arguments and the operation is done
on the formal arguments.
• Any changes made in the formal arguments does not effect the actual arguments because formal
arguments are photocopy of actual arguments.
• Changes made in the formal arguments are local to the block of called-function.
• Once control returns back to the calling-function the changes made vanish.

```
/*Program to swap two no.s using call by value*/
#include <stdio.h>
/* function declaration */
void swap(int x, int y);
int main ()
{
 /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   swap(a, b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
/* function definition to swap the values */
void swap(int x, int y) {

   int temp;

   temp = x; /* save the value of x */
   x = y;    /* put y into x */
   y = temp; /* put temp into y */
}
```

CALL BY REFERENCE (5M)

When, argument is passed using pointer, address of the memory-location is passed instead of value. So all the changes are reflected outside the function definition. When we will swap the value of the two variables, output will be Variables with changed values.

```
/*Program to swap two no.s using call by reference*/
#include <stdio.h>
/* function declaration */
void swap(int *x, int *y);
int main ()
{
 /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   /* calling a function to swap the values */
   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
/* function definition to swap the values */
void swap(int *x, int *y) {

   int temp;

   temp = *x; /* save the value of x */
   *x = *y;    /* put y into x */
   *y = temp; /* put temp into y */
}
```

6	**What is recursion? (1M)** **a) Write a program to find factorial of number using recursion. (4.5M)** **b) Write a program to find the sum of n natural no.s using recursion. (4.5M)**	**10M**	Co 3	L3

Recursion is a process of calling a function within itself.

1./* Program to find factorial of number using recursion. */

```
#include<stdio.h>
int fact(int n);
int main()
{
     int n=5;
     printf("The factorial of %d is %d",n,fact(n));
     return 0;
}
int fact(int n)
{
    if(n==0 || n==1)
    return 1;
   else
    return n*fact(n-1);
}
```

2./* Program to find sum of n natural no.s using recursion. */

```
#include<stdio.h>
int sum(int n);
int main()
{
     int n=5;
     printf("The sum of %d natural no.s  is %d",n,fact(n));
     return 0;
}
int sum(int n)
{
    if(n==0)
    return 1;
   else
    return n+sum(n-1);
}
```

7(a)	**Write a C program to find power of a number using loops. (05M)**
	/*Program to find power of a number. */
	#include<stdio.h>
	int main()
	{
	    int a=2,b=3,i;
	    for(i=1;i<=b;i++)
	    {
	        res=res*a;
	    }
	    printf("The result of %d raised to the power %d is %d",a,b,res);
	    return 0;
	}
		05M	CO 2	L3

7(b)	**Explain break and continue statements in c with syntax and example.**
	Break statement: This statement is used to come out of a loop or a switch case statement.
	Syntax: **(2.5M)**
	    While(condition)
	    {
	        if(condition)
	        {
	            break;
	        }
	    }
	Continue statement: This statement is used to skip the current iteration and move to the next iteration.
	Syntax: **(2.5M)**
	    While(condition)
	    {
	        if(condition)
	        {
	            continue;
	        }
	    }
		05M	CO 2	L2