CMR
INSTITUTE OF
TECHNOLOGY

Internal Assesment Test - II
-SCHEME & SOLUTION

Sub:	PROGRAMMING IN C AND DATA STRUCTURES							Code:	15PCD13
Date:	03 / 11 / 2016	Duration:	90 mins	Max Marks:	50	Sem:	I	Sections	I/J/K/L/M/N/O

Answer Any FIVE FULL Questions

		Marks	OBE CO	OBE RBT
1(a)	What is an array? Explain the declaration and initialization of single dimensional and two dimensional arrays.	[05]	CO1	L1

ARRAY

• Array is a collection of elements of same data type.
• 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

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

Declaration of Single Dimensional Arrays

• The syntax is shown below:
data_type array_name[array_size]
where data_type can be int, float or char
array_name is name of the array
array_size indicates number of elements in the array
• For ex:
int age[5];
For ex:
int age[5]={2,4,34,3,4};

MULTI DIMENSIONAL ARRAYS

• Arrays with two or more dimensions are called multi dimensional arrays.
• For ex:
int matrix[2][3];
int matrix[2][3]= {1, 23, 11, 44, 5, 6};

(b)	Explain strings in C and any four string manipulation functions.	[05]	CO1	L1

STRINGS

• Array of character are called strings.
• A string is terminated by null character /0.
• For ex:
"c string tutorial"
• Strings are declared in C in similar manner as arrays.
Only difference is that, strings are of „char“ type.
• For ex:
char s[5];

For ex:

```
char s[5]={'r', 'a', 'm', 'a' };
char s[9]="rama";
```

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);
• Example: Program to illustrate the use of strcmp().

2(a)	Write a C program using function ***void sort (int n, int a [])*** to arrange elements in ascending order. Use 'bubble sort' technique.	[10]	CO4, CO3	L3

```
#include <stdio.h>

void bubble_sort(long [], long);

int main()
{
  long array[100], n, c, d, swap;

  printf("Enter number of elements\n");
  scanf("%ld", &n);

  printf("Enter %ld longegers\n", n);

  for (c = 0; c < n; c++)
    scanf("%ld", &array[c]);

  bubble_sort(array, n);

  printf("Sorted list in ascending order:\n");
```

```
  for ( c = 0 ; c < n ; c++ )
    printf("%ld\n", array[c]);

  return 0;
}

void bubble_sort(long list[], long n)
{
  long c, d, t;

  for (c = 0 ; c < ( n - 1 ); c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (list[d] > list[d+1])
      {
        /* Swapping */

        t       = list[d];
        list[d]   = list[d+1];
        list[d+1] = t;
      }
    }
  }
}
```

3(a)	What is a function? Write a C program to find cube of a number using functions.	[10]	CO4	L3

```
#include <stdio.h>

double cube(double num)
{
    return (num * num * num);
}



int main()
{
    int num;
    double c;

    printf("Enter any number: ");
    scanf("%d", &num);

    c = cube(num);

    printf("Cube of %d is %.2f\n", num, c);

    return 0;
}
```

4 (a)	Explain call by value and call by reference parameter passing mechanism.	[6]	CO1	L1

ARGUMENT PASSING – CALL BY VALUE

• 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.

CALL BY REFERENCE

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.

(b) Program to swap 2 number using call by reference.

```
#include<stdio.h>
void swap(int *a,int *b)
{ // pointer a and b points to address of num1 and num2
respectively
int temp;
temp=*a;
*a=*b;
*b=temp;
}
void main()
{
int num1=5,num2=10;
swap(&num1, &num2); //address of num1 & num2 is passed
to swap function
printf("Number1 = %d \n",num1);
printf("Number2 = %d",num2);
}
```

Output:

```
Number1 = 10
Number2 = 5
```

		[4]	CO4	L3
5(a)	Write a C program to evaluate the polynomial $\mathbf{f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0}$, for a given value of x and coefficients (a1, a2, a3, a4) using Horner's method.	[10]	CO3	L3

```
#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 );
// Check if the order of the polynomial is zero.
if ( order == 0 )
{
printf ( "\nThe sum of polynomial f(%f): %f\n\n", x, a [ 0 ] );
return 0;
}
// Initialize sum to the highest order coefficient.
sum = a [ order ] * x;
// Compute sum using Horner's method.
for ( i = order - 1 ; i > 0 ; i-- )
{
sum = ( sum + a [ i ] ) * x;
}
// Add the constant a (a0) to the sum.
sum = sum + a [ 0 ];
// Display the sum of the given polynomial.
printf ( "\nThe sum of polynomial f(%f): %f\n\n", x, sum );

return 0;
}
```

6(a) What is file? Explain the following functions with syntax and examples: i) fopen( ) ii)fprintf( ) iii) fscanf() [10] CO4 L3

A file is defined as a collection of data stored on the secondary device such as hard disk. An input file contains the same items we might have typed in from the keyboard. An output file contains the same information that might have been sent to the screen as the output from our program.

To open a file for writing, we use the same procedure as given in previous section. But, the file mode has to be changed from “r” to “w”. Suppose, the file “input.txt” has to be opened for writing. We can use the following instructions.

```
#include <stdio.h>

FILE *fp;
fp = fopen(“input.txt”, “w”);
```

fscanf: The function of **fscanf** and **scanf** are exactly same. Only change is that **scanf** is used to get data input from keyboard, whereas **fscanf** is used to get data from the file pointed to by **fp**.

Because input is read from the file, extra parameter file pointer *fp* has to be passed as the parameter. Rest of the functionality of **fscanf** remains same as **scanf.** The syntax of fscanf**()** is shown below:

fscanf(fp, "format string", list);

fprintf: The function of **fprintf** and **printf** are exactly same. Only change is that **printf** is used to display the data onto the video display unit, whereas **fprintf** is used to send the data to the output file pointed to by **fp**. Since file is used, extra parameter file pointer *fp* has to be passed as parameter. Rest of the functionality of **fprintf** remains same as **printf**. The syntax of **fprintf**() is shown below:

fprintf(fp, "format string", list);

7(a)	Explain the following i)Structure data type ii) Concept of array of structure	[05]	CO6	L1

A *structure* is defined as a collection of data of same/different data types. All data items thus grouped are logically related and can be accessed using variables. Thus, structure can also be defined as a group of variables of same or different data types. The variables that are used to store the data are called *members of the structure* or *fields of the structure*. In C, the structure is identified by the keyword **struct.**

```
Struct student
{
   Char name[20];
   Int rollno;
   Char grade[3];
}
```

As variables are declared before they are used in the function, the structures are also should be declared before they are used. A structure can be declared using three different ways as shown below:

1) Tagged Structures
2) Structure variable
3) Type Defined Structures

The structure definition with tag name is called tagged structure. The tag name is the name of the structure. The syntax of tagged structure is shown below:

```
struct   tag name
{
   Type 1 member 1;
   Type 2 member 2;
}
```

The syntax of structure definition and declaration using structure variables is

shown below:

```
struct   tag name
{

   Type 1 member 1;
   Type 2 member 2;

} var1, var2;
```

The structure definition associated with keyword **typedef** is called type-defined structure.

```
Typedef struct
{

   Type 1 member 1;
   Type 2 member 2;

} TYPE_ID;
```

```
typedef struct
{
        char    name[10];
        int     roll_number;
        float    average_marks;
} STUDENT;
```

ARRAY OF STRUCTURES:

Structure is used to store the information of One particular object but if we need to store such N objects then Array of Structure is used where we declare object of type array.

```
#include<stdio.h>

struct bookinfo
{
    bname[20];
    int pages;
    int price;
} book[3];                    variable of type array


int main( )
{

int i;
for(i=0 ; i<3 ; i++)
{

printf("enter name of book:")
gets(book[i].bname);
```

```
printf(“enter number of pages”);
scanf(“%d”book[i].pages);

printf(“enter the price of book”);
scanf(“%f”,book[i].price);

printf(“------------------BOOK DETAILS----------------“);

for(i=0 ; i<3 ; i++)
{
printf(“name of the book:%s”,book[i].name);
printf(“no of the pages:%s”,book[i].pages);
printf(“price of  book:%s”,book[i].price);
}
return 0;
}
```

(b) Write a C program that implements string copy operation ***STRCOPY*** (str1, str2) that copies a string *str1* to another string *str2* without using library functions. [05] CO5 L3

```
#include<stdio.h>
// Function declaration for strcopy.
void strcopy ( char str1 [ ], char str2 [ ] );
int main()
{
char str1 [ 20 ], str2 [ 20 ];
// Input the string that you want to copy.
printf ( "\nEnter string to copy: " );
gets ( str1 );
// Function call.
strcopy ( str1 , str2 );
// Display the contents of str1 and str2.
printf ( "\n\nCopying success!!!!!\n" );
printf ( "\nThe first string is: " );
puts ( str1 );
printf ( "\nThe second string is: " );
puts ( str2 );
// Function definition for strcopy.
void strcopy ( char str1 [ ], char str2 [ ] )
{
int i;
// Copying the contents of str1 to str2 until NULL is encountered.
i = 0;
while ( str1 [ i ] != '\0' )
{
str2 [ i ] = str1 [ i ];
i++;
}
// Append NULL character at the end of str2.
```

```
str2 [ i ] = '\0';
}
```

8(a) What is a Pointer? Write a C program to calculate the sum and standard deviation of five real numbers in an array using pointer. [1+5] C05 L1,L3

POINTER

• A pointer is a variable which holds address of another variable or a memory-location.
• For ex:

```
c=300;
pc=&c;
```

Here pc is a pointer; it can hold the address of variable c
& is called reference operator.

DECLARATION OF POINTER VARIABLE

• Dereference operator(*) are used for defining pointer-variable.
• The syntax is shown below:

```
data_type *ptr_var_name;
```

• For ex:

```
int *a; // a as pointer variable of type int
float *c; // c as pointer variable of type float
```

• Steps to access data through pointers:
1) Declare a data-variable ex: int c;
2) Declare a pointer-variable ex: int *pc;
3) Initialize a pointer-variable ex: pc=&c;
4) Access data using pointer-variable ex: printf("%d",*pc);

```
#include<stdio.h>
#include<math.h>
int main()
{
int n, i;
double a [ 10 ], sum, mean, sd, total, var;
// Read the number of integers.
printf ( "\nEnter the value of n: " );
scanf ( "%d", &n );
// Read the integers.
printf ( "\nEnter %d numbers: ", n );
for ( i = 0; i < 5 ; i++ )
{
scanf ( "%lf", (a+i) );
}
//Compute sum.
sum = 0;
for ( i = 0 ; i < n ; i++ )
{
sum = sum + *(a+i);
}
printf ( "\nThe Sum is: %lf\n", sum );
// Compute mean.
mean = sum / n;
printf ( "\nThe Mean is: %lf\n", mean );
// Compute variance and standard deviation.
total= 0;
for ( i = 0; i < 5 ; i++)
{
```

```
total = total + pow ( (*(a+i)-mean), 2);
}
var = total / n;
sd = sqrt ( var );
printf ( "\nThe Standard Deviation is: %lf\n\n", sd );
return 0;
}
```

(b) What do you mean by dynamic memory allocation? Explain malloc ( ) and calloc ( ) functions with syntax and examples. [1+3] CO5 L1,L2

Dynamic Memory Allocation

• Dynamic memory allocation is the process of allocating memory-space during execution-time i.e. run time.
• If there is an unpredictable storage requirement, then the dynamic allocation technique is used.
• This allocation technique uses predefined functions to allocate and release memory for data during execution-time.
• There are 4 library functions for dynamic memory allocation:
1) malloc()
2) calloc()
3) free()
4) realloc()

MALLOC():

• The name malloc stands for "memory allocation".
• This function is used to allocate the requirement memory-space during execution-time.
• The syntax is shown below:

```
data_type *p;
p=(data_type*)malloc(size);
```

here p is pointer variable
data_type can be int, float or char
size is number of bytes to be allocated.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
int n, i, *ptr, sum=0;
printf("Enter number of elements: ");
scanf("%d",&n);
ptr=(int*)malloc(n*sizeof(int)); //memory allocated using
malloc
printf("Enter elements of array: ");
for(i=0;i<n;++i)
{
scanf("%d ",ptr+i);
sum+=*(ptr+i);
}
printf("Sum=%d",sum);
```

```
free(ptr);
}
```

CALLOC():

• The name calloc stands for "contiguous allocation".
• This function is used to allocate the required memory-size during execution-time and at the same time, automatically initialize memory with 0's.
• The syntax is shown below:

```
data_type *p;
p=(data_type*)calloc(n,size);
```

• If memory is successfully allocated, then address of the first byte of allocated space is returned.
If memory allocation fails, then NULL is returned.
• The allocated memory is initialized automatically to 0's.
• For ex:

```
ptr=(int*)calloc(25,sizeof(int));
```

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
int n,i,*ptr,sum=0;
printf("Enter number of elements: ");
scanf("%d",&n);
ptr=(int*)calloc(n,sizeof(int));
printf("Enter elements of array: ");
for(i=0;i<n;++i)
{
scanf("%d ",ptr+i);
sum+=*(ptr+i);
}
printf("Sum=%d",sum);
free(ptr);
}
```

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1:	Identify variable and their data types for a given problem	1	0	0	0	0	1	0	0	0	0	0	0
CO2:	Use operators to form a computational step.	1	0	0	0	0	1	0	0	0	0	0	0
CO3:	Use Control statement to solve simple algorithms - sort, search.	1	1	1	1	0	1	0	0	0	0	0	0
CO4:	Write functions that solve a given problem.	1	1	0	1	1	1	0	0	0	0	0	0
CO5:	Explain dynamic memory allocation using an example - Array of strings	1	1	0	0	0	1	0	0	0	0	0	0
CO6:	Explain basic data structures used for Programming - Arrays, List, Stack, Queue, Trees.	1	0	0	0	1	1	0	0	0	0	0	0

Cognitive level	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.

PO1 - *Engineering knowledge*; PO2 - *Problem analysis*; PO3 - *Design/development of solutions*; PO4 - *Conduct investigations of complex problems*; PO5 - *Modern tool usage*; PO6 - *The Engineer and society*; PO7- *Environment and sustainability*; PO8 – *Ethics*; PO9 - *Individual and team work*; PO10 - *Communication*; PO11 - *Project management and finance*; PO12 - *Life-long learning*