CMR INSTITUTE OF TECHNOLOGY

Internal Assessment Test 3 – May 2018-SCHEME & SOLUTION

Sub:	Programming in C and Data Structures							**Code:**	17PCD23
Date:	22/ 05/2018	Duration:	90mins	Max Marks:	50	**Sem:**	II	**Branch:**	ALL

Note: Answer any five questions:

1.(a)	**What is a Structure? Explain the following:** **(1) syntax of structure definition and structure variable declaration with example.** **(2) Concept of array of structure with example.**	**Marks**	CO	RBT
		10M	CO 5	L1,L2

Structure is a collection of different data type elements stored continuously in the memory.

(1) Syntax of structure definition:

```
Struct structurename
{
      datatype members;
      ------------------------
};
```

Structure variable declaration

Struct structurename varname;

```
Example: struct student
                 {
                       char name[10];
                       int roll;
                  };
Struct student s;
```

(2) Concept of array of structure with example.

Array of structure is a collection of same structure type elements stored continuously in the memory.

```
Ex: struct student
                 {
                       char name[10];
                       int roll;
                  };
Struct student s[5];
```

<u>Programming example:</u>

```
#include<stdio.h>
int main()
{
         struct student
                 {
                       char name[10];
                       int roll;
                  };
         Struct student s[2];
         int i;
         for (i=0;i<2;i++)
         {
                printf("Enter details of student %d",i+1);
                scanf("%s%d",s[i].name,&s[i].roll);
         }
        printf(" The student details are:\n");
        for (i=0;i<2;i++)
         {
                printf("The details of student %d",i+1);
                printf("%s%d",s[i].name,s[i].roll);
         }
      return 0;
}
```

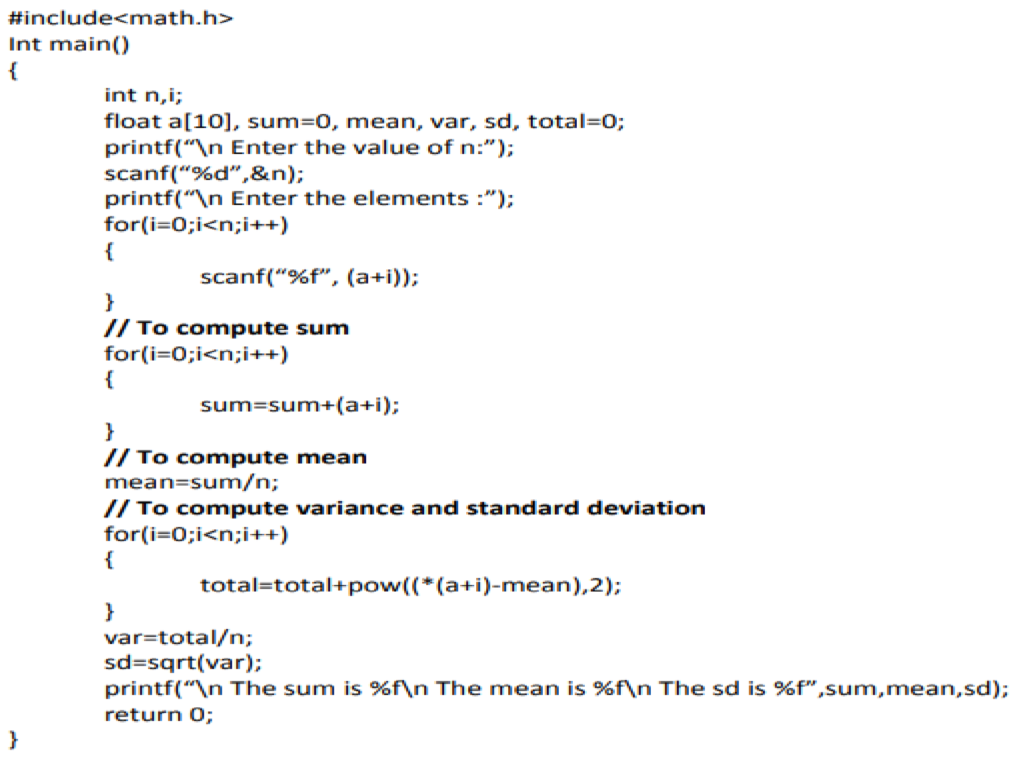
2 (a)	**What is file? Explain the following functions with syntax and examples:**
	i)fopen() ii)fprintf() iii)fscanf() iv)fgets() v)fputs()
	A file is sequence of bytes stored in the secondary storage for use in long run.
	(1) fopen()
	syntax: fopen(“name of file”,”mode”);
	mode: r, w, a
	Ex: FILE *p;
	p=fopen(“a.txt”,”r”);
	
	(2) fprintf()
	syntax: fprintf(fp,”String literal”,varnames);
	Ex: FILE *fp;
	p=fopen(“a.txt”,”w”);
	fprintf(fp,”This is an example”);
	
	(3) fscanf()
	syntax: fscanf(fp,”String literal”,&varnames);
	Ex: FILE *fp;
	char buff[20];
	p=fopen(“a.txt”,”r”);
	fscanf(fp,”%s”,buff);
	
	(4) fgets()
	syntax: fgets(str,n,file_ptr);
	Ex: FILE *fp;
	char buff[20];
	p=fopen(“a.txt”,”r”);
	fgets(buff,20,fp);
	
	(5) fputs()
	syntax: fputs(str,file_ptr);
	Ex: FILE *fp;
	char buff[]=”ani”;
	p=fopen(“a.txt”,”w”);
	fputs(buff,fp);
		10M	CO 5	L1,L 2

3	**Given two university information files “studentname.txt” and “usn.txt” that contains students Name and USN respectively. Write a C program to create a new file called “output.txt” and copy the content of files “studentname.txt” and “usn.txt” into output file in the sequence shown below. Display the contents of output file “output.txt” on to the screen.**	**10M**	CO 5	L2

```c
#include<stdio.h>
int main()
{

        FILE *fp1,*fp2,*fp3;
        char buff1[100],buff2[100];
        fp1=fopen("usn.txt","r");
        fp2=fopen("name.txt","r");
        fp3=fopen("output.txt","w");
fprintf(fp3,"usn\tname\n");
                while(1)
                        {
                                fscanf(fp1,"%s",buff1);
                                fscanf(fp2,"%s",buff2);
                                if(!feof(fp1) && !feof(fp2))
                                fprintf(fp3,"%s\t%s\n",buff1,buff2);
                                else
                                        break;
                        }
fclose(fp1);
fclose(fp2);
fclose(fp3);
}
```

4	**What is a Pointer? Write a program to find and display sum, mean and Standard Deviation of n numbers using pointers.** A pointer is a variable that stores the address of another variable or memory location.	**10M**	CO 4	L1, L3



```c
#include<math.h>
Int main()
{
        int n,i;
        float a[10], sum=0, mean, var, sd, total=0;
        printf(“\n Enter the value of n:”);
        scanf(“%d”,&n);
        printf(“\n Enter the elements :”);
        for(i=0;i<n;i++)
        {
                scanf(“%f”, (a+i));
        }
        // To compute sum
        for(i=0;i<n;i++)
        {
                sum=sum+(a+i);
        }
        // To compute mean
        mean=sum/n;
        // To compute variance and standard deviation
        for(i=0;i<n;i++)
        {
                total=total+pow((*(a+i)-mean),2);
        }
        var=total/n;
        sd=sqrt(var);
        printf(“\n The sum is %f\n The mean is %f\n The sd is %f”,sum,mean,sd);
        return 0;
}
```

5	**What is Dynamic memory allocation? Explain different Dynamic memory allocation function in C with syntax and examples.**	**10M**	Co 4	L2

The process of allocating memory during program execution is called dynamic memory allocation.C language offers 4 dynamic memory allocation functions. They are,

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

S.no	Function	Syntax
i	malloc ()	malloc (number *sizeof(int));
ii	calloc ()	calloc (number, sizeof(int));
iii	realloc ()	realloc (pointer_name, number * sizeof(int));
iv	free ()	free (pointer_name);

i. malloc() function in C:

- malloc () function is used to allocate space in memory during the execution of the program.
- malloc () does not initialize the memory allocated during execution. It carries garbage value.
- malloc () function returns null pointer if it couldn't able to allocate requested amount of memory.

Example program for malloc() function in C:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
char *mem_allocation;
/* memory is allocated dynamically */
mem_allocation = malloc( 20 * sizeof(char) );
if( mem_allocation== NULL )
{
printf("Couldn't able to allocate requested memory\n");
}
else
{
strcpy( mem_allocation,"CMRIT.com");
}
printf("Dynamically allocated memory content : " \ n%s\n", mem_allocation );
free(mem_allocation);

}
```

ii. calloc() function in C:

calloc () function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn t.

Example program for calloc() function in C:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{
char *mem_allocation;
/* memory is allocated dynamically */
mem_allocation = calloc( 20, sizeof(char) );
if( mem_allocation== NULL )
{
printf("Couldn't able to allocate requested memory\n");
}
else
{
strcpy( mem_allocation," CMRIT.com");
}
 printf("Dynamically allocated memory content : " \ "%s\n", mem_allocation );
 free(mem_allocation);
 }
```

```
Output: Dynamically allocated memory content :
CMRIT.com
```

iii. realloc() function in C:

realloc () function modifies the allocated memory size by malloc () and calloc () functions to new size.

If enough space doesn t exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

iv. free() function in C:

free () function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.

6(a)	**What are preprocessor directives? Explain any 5 preprocessor directives in c.**
	The C Preprocessor is not part of the compiler, but is a separate step in the compilation process. In simplistic terms, a C Preprocessor is just a text substitution tool.
	i) **#include** - This directive allows external header files to be processed by the compiler.
	Syntax:
	#include <*header-file*> or #include "*source-file*"
	When enclosing the file with < and >, then the implementation searches the known header directories for the file (which is implementation-defined) and processes it. When enclosed with double quotation marks, then the entire contents of the source-file is replaced at this point. The searching manner for the file is implementation-specific.
	Examples:
	#include <stdio.h> #include "my_header.h"
	ii)#pragma- The #pragma directive allows a directive to be defined. Its effects are implementation-defined. If the pragma is not supported, then it is ignored.
	Syntax: #pragma ***directive***
	#pragma page( ) // Forces a form feed in the listing
	#pragma warning(disable: 2044) // Disables warning number 2044
	#pragma line 100 // Sets the current line number to 100 for listing and reporting purposes.
		05M	Co 6	L2

6(b)	**Write note on a) Stack b) Queue c) Linked list d) Tree.** **STACKS** • A stack is a special type of data structure where elements are inserted from one end and elements are deleted from the same end. • Using this approach, the Last element Inserted is the First element to be deleted Out, and hence, stack is also called LIFO data structure. • The various operations performed on stack: Insert: An element is inserted from top end. Insertion operation is called push operation. Delete: An element is deleted from top end. Deletion operation is called pop operation. Overflow: Check whether the stack is full or not. Underflow: Check whether the stack is empty or not. **APPLICATIONS OF STACK** 1) Conversion of expressions: The compiler converts the infix expressions into postfix expressions using stack. 2) Evaluation of expression: An arithmetic expression represented in the form of either postfix or prefix can be easily evaluated using stack. 3) Recursion: A function which calls itself is called recursive function. 4) Other applications: To find whether the string is a palindrome, to check whether a given expression is valid or not. **QUEUES** • A queue is a special type of data structure where elements are inserted from one end and elements are deleted from the other end. • The end at which new elements are added is called the rear and the end from which elements are deleted is called the front. • The first element inserted is the first element to be deleted out, and hence queue is also called FIFO data structure. • The various operations performed on queue are 1) Insert: An element is inserted from rear end. 2) Delete: An element is deleted from front end. 3) Overflow: If queue is full and we try to insert an item, overflow condition occurs. 4) Underflow: If queue is empty and try to delete an item, underflow condition occurs.	**05M**	CO 6	L2

LINKED LIST

• A linked list is a data structure which is collection of zero or more nodes where each node has some information.

• Normally, node consists of 2 fields

1) info field which is used to store the data or information to be manipulated
2) link field which contains address of the next node.

• Different types of linked list are SLL & DLL

• Various operations of linked lists

1) Inserting a node into the list
2) Deleting a node from the list
3) Search in a list
4) Display the contents of list

SINGLY LINKED LIST

• This is a collection of zero or more nodes where each node has two or more fields and only one link field which contains address of the next node.

BINARY TREE

• A tree in which each node has either zero, one or two subtrees is called a binary tree.

• figure

• Each node consists of three fields

llink: contains address of left subtree

info: this field is used to store the actual data or information to be manipulated

rrlink: contains address of right subtre

BINARY TREE APPLICATIONS

• Tree traversal refers to process of visiting all nodes of a tree exactly once (Figure 5.16).

• There are 3 techniques, namely:

1) Inorder traversal(LVR);
2) Preorder traversal(VLR);
3) Postorder traversal(LRV). (Let L= moving left, V= visiting node and R=moving right).

7	**Write a C program to maintain a record of n student details using an array of structures with four fields (Roll number, Name, Marks, and Grade). Assume appropriate data type for each field. Print the marks of the student, given the student name as input.**	**10M**	Co 5	L3

```
  /* Program to create details of a student using structures. */
  #include<stdio.h>
  #include<string.h>
  int main()
  {
          struct student
          {
                  int rollno;
                  char name [ 30 ];
                  int marks;
                  char grade [ 2 ];
          };
          struct student stud_arr [ 30 ];

 int i, num_studs;
 char stud_name [ 30 ];
 // Input the number of students.
 printf ( "\nEnter the number of students: " );
 scanf ( "%d" , &num_studs );
 // Input student details.
 for ( i = 0 ; i < num_studs ; i++ )
 {
         printf ( "\n\nEnter the following details for Student %d", i );
         printf ( "\nRoll No: " );
         scanf ( "%d" , &stud_arr[i].rollno );
         printf ( "\nName: " );
         scanf ( "%s" , stud_arr[i].name );
         printf ( "\nMarks: " );
         scanf ( "%d" , &stud_arr[i].marks );
         printf ( "\nGrade: " );
         scanf ( "%s" , stud_arr[i].grade );
 }
 // Input the student name that you want to search for.
 printf ( "\n\nEnter the student name you wish to search: " );
 scanf ( "%s" , stud_name );
 // Linear Search.
 for ( i = 0 ; i < num_studs ; i++ )
 {
```

```
                        if ( strcmp ( stud_name, stud_arr[i].name ) == 0 )
                        {
                        printf ( "\nMarks obtained by %s is: %d\n\n", stud_arr[i].name, stud_arr[i].marks );
                        return 0;
                        }
            }
            // Display record not found.
            printf ( "\nStudent Record not Found!!!!\n\n" );
            return 0;
}
```