Hall Ticket Number:

I/IV B.Tech (Regular) DEGREE EXAMINATION

December, 2018 **Common to ECE & MECH**

First Semester **Problem Solving with Programming**

Time: Three Hours **Maximum:** 50 Marks

Answer Question No.1 compulsorily. (1X10 = 10 Marks)

Answer ONE question from each unit. (4X10=40 Marks)

1. Answer all questions (1X10=10 Marks)
   a) Compare declaration and initialization of a variable.
   b) Define Keyword. Name any four.
   c) Differentiate getch() and getche().
   d) Find the output of the following

```
#include<stdio.h>
void main()
{
  int i;
   i=1;
  do
 {
    printf("%d",i);
    i++;
    if(i==5)
    break;
  }while(i<=10);
}
```

   e) Write the general form of ***for*** loop structure.
   f) Drawbacks of arrays.
   g) Consider the 32-bit compiler. We need to store address of integer variable to integer pointer. What will be the size of integer pointer?
   h) Difference between actual parameters and formal parameters.
   i) What are the different ways to access structure members?
   j) Give the purpose of fseek() and ftell().

UNIT I

2. a) Define Datatype. What are the various types of datatypes used in C. Also mention their size, range and format specifiers with suitable examples. 6M
   b) Write a C-program to convert given year to months and remaining days. 4M

(OR)

3. a) Compare and explain ***else-if*** ladder and ***switch()*** case with suitable example. 6M
   b) Write a C-program to identify the class of an input character. 4M

UNIT II

4. a) Explain in detail about 1-D and 2-D character arrays with suitable examples. 6M
   b) Write a C-program to check whether the given is Armstrong or not. 4M

(OR)

5. a) What is the purpose of string handling functions? Explain any four string handling functions with suitable examples. 6M
   b) Write a C-program to read the list of elements and search for a given element. 4M

UNIT III

6. a) Describe various ways to pass parameters to functions with suitable examples. 6M
   b) Write a C-program to count the number of words, lines in a given paragraph. 4M

(OR)

7. a) Define Pointer. Explain various memory allocation functions with suitable examples. 6M
   b) Write a C-program to find GCD of a given number using recursion. 4M

UNIT IV

8. a) Explain array of structures with suitable examples. 6M

   b) Define a Structure called employee that consists of name, date of joining, employee id and salary. Write a C-program to read three employee details and sort them in ascending order based on their employee id. 4M

(OR)

9. a) Write about operations on files with suitable examples. 6M

   b) What is Pre-processor directive? Explain any three with an example. 4M

Scheme & Solutions

1. Answer all questions

a) Compare declaration and initialization of a variable.

Ans: For a **variable**, a **definition** is a **declaration** which allocates storage for that **variable**.

Declaration of variable in c can be done using following syntax:

data_type variable_name; or *data_type variable1, variable2,…,variablen;*

where *data_type* is any valid c data type and *variable_name* is any valid identifier.

Example: int a;

Initialization is the specification of the initial value to be stored in an object, which is not necessarily the same as the first time **you** explicitly assign a value to it.

Syntax: *data_type variable_name=value*;

Example: int a=10; **1M**

b) Define Keyword. Name any four.

Ans: Keywords are predefined, reserved words used in programming that have special meanings to the compiler. **Keywords** are part of the syntax and they cannot be used as an identifier. Some of the keywords are int, float, char ,double, etc. **1M**

c) Differentiate getch() and getche()

Ans: getch(): This is a non-buffer function. This is used to read a single character as input from keyboard. This function directly assigns the reading character to the relevant variable. While giving the data to this function the entered character is not visible on screen (Non-echo function).

getche(): This is a non-buffer function. This is also used to read a single character as input from keyboard. It will not require for ***Enter*** key, the given input visible on the screen. **1M**

d) Find the output of the following

Ans: 1234 **1M**

e) Write the general form of *for* loop structure.

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

The general form of ***for*** loop is as follows:

```
 for ( initialization; condition; increment or decrement )
{
  statement(s);
}
```

f) Drawbacks of arrays.

Ans: 1. Since array is of fixed size, if we allocate more **memory** than requirement then the **memory space** will be wasted.

2. The elements of array are stored in consecutive **memory** locations. Any two--→ **1M**

g) Consider the 32-bit compiler. We need to store address of integer variable to integer pointer. What will be the size of integer pointer?

Ans: 2 bytes= 16 bits. **1M**

h) Difference between actual parameters and formal parameters.

Ans: Actual parameters are the parameters which are present in function call.

Formal parameters are the parameters which are present in function definition or function implementation. **1M**

i) What are the different ways to access structure members?

Ans: There are two ways to access structure members. They are

1. Using dot operator (**.**)
2. Using arrow operator (→) **1M**

j) Give the purpose of fseek() and ftell().

Ans: fseek(): fseek() function is used to move the file position to a desired location within the file.

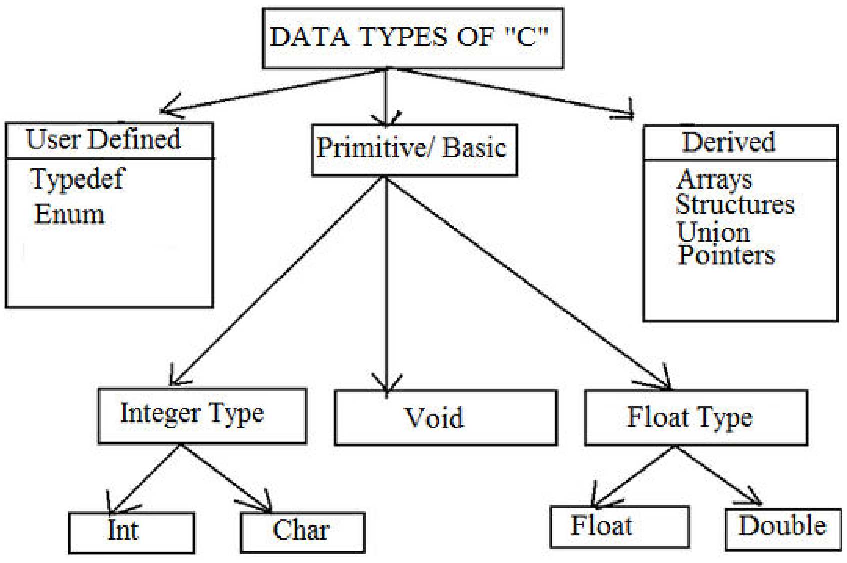
ftell(): ftell() function is **used** to get current position of the file pointer. **1M**

UNIT-I

2. a) Define Datatype. What are the various types of datatypes used in C. Also mention their size, range and format specifiers with suitable examples.
(6M)

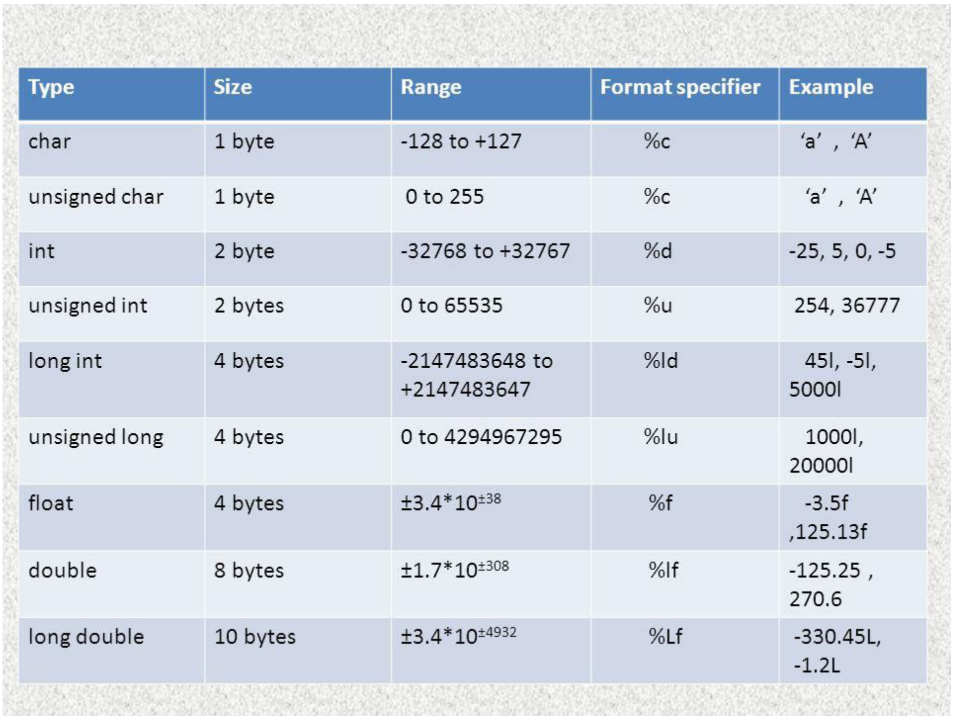
Ans: Datatype: Data types simply refers to the **type** and size of **data** associated with variables and functions. -----→ 1Mark

Classification of datatypes:



------→ 2Marks

Size, Range and format specifiers:



Type	Size	Range	Format specifier	Example
char	1 byte	-128 to +127	%c	'a' , 'A'
unsigned char	1 byte	0 to 255	%c	'a' , 'A'
int	2 byte	-32768 to +32767	%d	-25, 5, 0, -5
unsigned int	2 bytes	0 to 65535	%u	254, 36777
long int	4 bytes	-2147483648 to +2147483647	%ld	45l, -5l, 5000l
unsigned long	4 bytes	0 to 4294967295	%lu	1000l, 20000l
float	4 bytes	$\pm3.4*10^{\pm38}$	%f	-3.5f ,125.13f
double	8 bytes	$\pm1.7*10^{\pm308}$	%lf	-125.25 , 270.6
long double	10 bytes	$\pm3.4*10^{\pm4932}$	%Lf	-330.45L, -1.2L

------→ 3Marks

2. b) Write a C-program to convert given year to months and remaining days.

```
#include <stdio.h>

int main()
{
    int days, years, weeks;

    /* Input total number of days from user */
    printf("Enter days: ");
    scanf("%d", &days);                                          --→ 2Marks

    /* Conversion */
    years = (days / 365);   // Ignoring leap year
    weeks = (days % 365) / 7;
    days  = days - ((years * 365) + (weeks * 7));

    /* Print all resultant values */
    printf("YEARS: %d\n", years);
    printf("WEEKS: %d\n", weeks);
    printf("DAYS: %d", days);
                                                                ---→ 2Marks
    return 0;
}
```

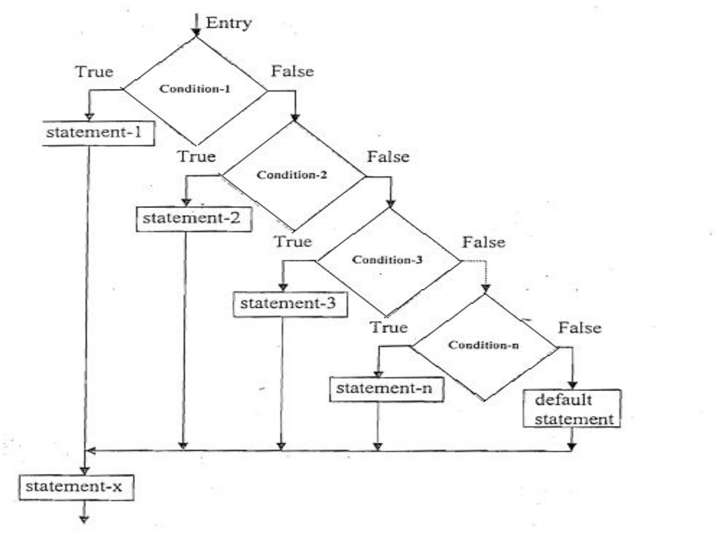
(OR)

3. a) Compare and explain *else-if* ladder and *switch()* case with suitable example. (6M)

Ans: Else If Ladder: *else if* statement can be defined as a control statement which controls the statement(s) to be executed on the basis of some conditions. Whenever the else if statement is used, the compiler or interpreter initially checks the condition whether it is true or false and if the condition is found to be true then, the corresponding statements are executed.

The **syntax and flow chart of else if ladder** can be represented as:

```
if( condition-1)
    statement-1;

else if (condition-2)
    statement-2;

else if (condition-3)
    statement-3;

else if (condition-4)
    statement-4;

else if (condition-n)
    statement-n;

else
    default statement;
```



Features of else if ladder:

- It evaluates an expression and then, the code is selected based on the true value of evaluated expression.
- Each else if has its own expression or condition to be evaluated.
- The variable data type used in the expression of else if is either integer or character.
- The decision making of the else if is dependent on zero or non-zero basis**.**

-----→ 3Marks

Switch Case:

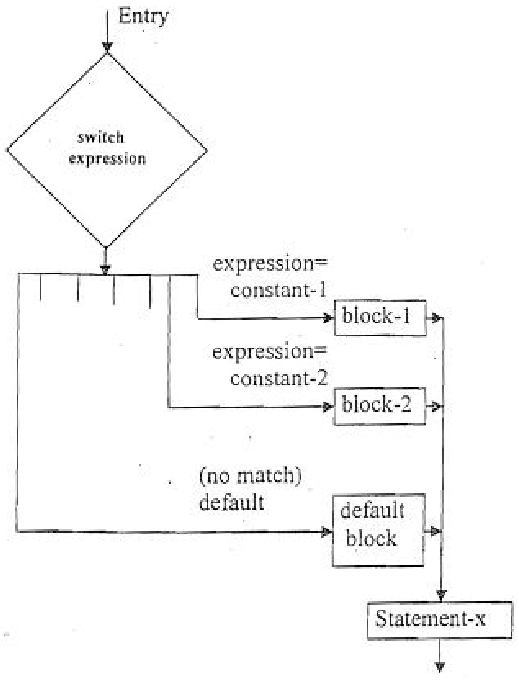
The *switch case* statement is similar to the else-if ladder as it provides multiple branching or multi-conditional processing. But, the basic difference between switch case and else if ladder is that the *switch case* statement tests the value of variable or expression against a series of different cases or values, until a match is found. Then, the block of code with in the match case is executed. If there are no matches found, the optional default case is executed.

Features of switch case:

- The *switch case* statement evaluates the value of an expression and a block of code is selected on the basis of that evaluated expression.
- Each case refers back to the original expression.
- The data type that can be used in switch expression is integer type only.
- Each case has a break statement.
- The switch case takes decision on the basis of equality.

The **syntax and flow chart of switch case** can be represented as:

```
switch(expression)
{
    case constant-1
        block-1;
        break;

    case constant-2
        block-2;
        break;

    case constant-3
        block-3;
        break;

    case constant-n
        block-n;
        break;

    default:
        default_block;
}

statement-x;
```



-----→ 3Marks

Difference between switch case and else if ladder:

So, after going through the two control statements in brief, here are the main difference between switch case and else if ladder:

1. In *else if ladder*, the control goes through the every else if statement until it finds true value of the statement or it comes to the end of the else if ladder. In case of *switch case*, as per the value of the switch, the control jumps to the corresponding case.
2. The switch case is more compact than lot of nested else if. So, switch is considered to be more readable.
3. The use of break statement in switch is essential but there is no need of use of break in else if ladder.
4. The variable data type that can be used in expression of switch is integer only where as in else if ladder accepts integer type as well as character.
5. Another difference between switch case and else if ladder is that the *switch* statement is considered to be less flexible than the else if ladder, because it allows only testing of a single expression against a list of discrete values.

6. Since the compiler is capable of optimizing the switch statement, they are generally considered to be more efficient. Each case in switch statement is independent of the previous one. In case of else if ladder, the code needs to be processed in the order determined by the programmer.
7. *Switch case statement* work on the basis of equality operator whereas *else if ladder*works on the basis of true false(zero/non-zero) basis.

3. b) Write a C-program to identify the class of an input character.

Ans:

```
#include <stdio.h>
int main()
{
   char ch;
   /* Input character from user */
   printf("Enter any character: ");
   scanf("%c", &ch);
                                                       ---→ 2Marks
   /* Alphabet check */
   if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
   {
      printf("'%c' is alphabet.", ch);
   }
   else if(ch >= '0' && ch <= '9')
   {
      printf("'%c' is digit.", ch);
   }
   else
   {
      printf("'%c' is special character.", ch);
   }
     return 0;
  }                                                         ----→ 2Marks
```

UNIT-II

4. a) Explain in detail about 1-D and 2-D character arrays with suitable examples. (6M)

Ans: 1D Character array or String: String is a sequence of characters that is treated as a single data item and terminated by null character '\0'.

Declaring and Initializing a string variables:

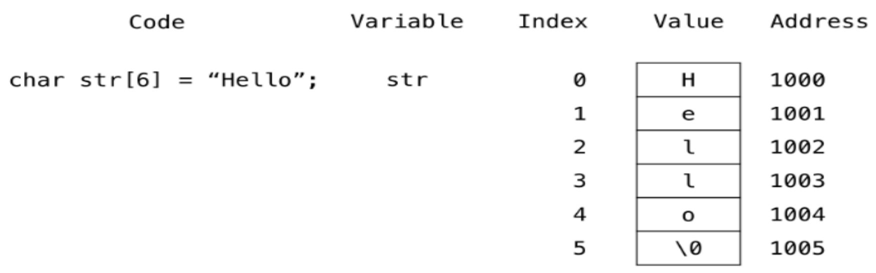
There are different ways to initialize a character array variable.

char name[13] = "StudyTonight";

char name[10] = {'L','e','s','s','o','n','s','\0'};

Memory Map:

For example: char str[6] = “Hello”;`char str[6] = "Hello";`



Code	Variable	Index	Value	Address
char str[6] = “Hello”;	str	0	H	1000
		1	e	1001
		2	l	1002
		3	l	1003
		4	o	1004
		5	\0	1005

Print the elements of the one-dimensional character array:

```
int i;
char str[6] = "Hello";

for (i = 0; str[i] != '\0'; i++) {
  printf("%c\n", str[i]);
}
```

----→ 3Marks

Two-Dimensional Character Arrays or Array of Strings:

Collection of strings is represented using array of strings.

Declaration: The general form of declaring an array of strings is as follows:

char arr[row][col];

where, arr - name of the array

row - represents number of strings

col - represents size of each string.

Initialization: The general form of initializing an array of strings is as follows:

char arr[row][col] = { list of strings };

Example:

char city[5][10] = { “DELHI”, “CHENNAI”, “BANGALORE”, “HYDERABAD”, “MUMBAI” };

Memory Map:

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

In the above storage representation memory is wasted due to the fixed length for all strings.

------→ 3Marks

4. b) Write a C-program to check whether the given is Armstrong or not.

```
#include <stdio.h>
#include <math.h>

void main()
{
    int number, sum = 0, rem = 0, cube = 0, temp;

    printf ("enter a number");
    scanf("%d", &number);
   temp = number;                                                    ----→ 2Marks
   while (number != 0)
   {
      rem = number % 10;
      cube = pow(rem, 3);
      sum = sum + cube;
      number = number / 10;
   }
   if (sum == temp)
      printf ("The given no is armstrong no");
   else
      printf ("The given no is not a armstrong no");
}                                                                    ----→2Marks
```

(OR)

5. a) What is the purpose of string handling functions? Explain any four string handling functions with suitable examples.

(6M)

Ans: String Handling functions: C supports a number of string handling functions. All of these built-in functions are aimed at performing various operations on strings and they are defined in the header file **string.h**. Some of the important string functions are

(i). strlen( ) This function is used to find the length of the string excluding the NULL character. In other words, this function is used to count the number of characters in a string. Its syntax is as follows:

int strlen(string);

Example: char str1[ ] = “WELCOME”;

int n;

n = strlen(str1); **-----→ 1.5 Marks**

(ii). strcpy( ) This function is used to copy one string to the other. Its syntax is as follows:

strcpy(string1,string2);

where string1 and string2 are one-dimensional character arrays. This function copies the content of string2 to string1. E.g., string1 contains master and string2 contains madam, then string1 holds madam after execution of the strcpy (string1,string2) function.

Example: char str1[ ] = “WELCOME”;

char str2[ ] =”HELLO”;

strcpy(str1,str2); **-----→ 1.5 Marks**

(iii). strcmp ( ) This function compares two strings character by character (ASCII comparison) and returns one of three values {-1,0,1}. The numeric difference is „0‟ if strings are equal .If it is negative string1 is alphabetically above string2 .If it is positive string2 is alphabetically above string1. Its syntax is as follows:

int strcmp(string1,string2);

Example: char str1[ ] = “ROM”;

char str2[ ] =”RAM”;

strcmp(str1,str2); (or) strcmp(“ROM”,”RAM”); **-----→ 1.5 Marks**

(iv). strcat ( ) This function is used to concatenate two strings. i.e., it appends one string at the end of the specified string. Its syntax as follows:

strcat(string1,string2);

where string1 and string2 are one-dimensional character arrays. This function joins two strings together. In other words, it adds the string2 to string1 and the string1 contains the final concatenated string. E.g., string1 contains **prog** and string2 contains **ram**, then string1 holds **program** after execution of the strcat() function.

Example: char str1[10] = “VERY”;

char str2[5] =”GOOD”;

strcat(str1,str2); **-----→ 1.5 Marks**

5. b) Write a C-program to read the list of elements and search for a given element. (4M)

Ans:

```
#include <stdio.h>
int main()
{
  int array[100], search, c, n;
   printf("Enter number of elements in array\n");
  scanf("%d", &n);
   printf("Enter %d integer(s)\n", n);
   for (c = 0; c < n; c++)
    scanf("%d", &array[c]);                                    -----→ 2Marks
  printf("Enter a number to search\n");
  scanf("%d", &search);




 for (c = 0; c < n; c++)
  {

   if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);
  return 0;
}                                                         ----→ 2Marks
```

UNIT-III

6. a) Describe various ways to pass parameters to functions with suitable examples.(6M)

Ans: A Parameter is the symbolic name for "data" that goes into a function. There are two ways to pass parameters in C: Pass by Value or call by value, Pass by Reference or pass by reference.

1. Call by value

- Here the values of the variables are passed by the calling function to the called function.
- If any value of the parameter in the called function has to be modified the change will be reflected only in the called function.

- This happens as all the changes are made on the copy of the variables and not on the actual ones.

Example: Call by value

```
#include <stdio.h>
int sum (int n);
void main()
{
    int a = 5;
    printf("\n The value of 'a' before the calling function is = %d", a);
    a = sum(a);
    printf("\n The value of 'a' after calling the function is = %d", a);
}
int sum (int n)
{
    n = n + 20;
    printf("\n Value of 'n' in the called function is = %d", n);
    return n;
}
```

-----→ 3Marks

2. Call by reference

- Here, the address of the variables are passed by the calling function to the called function.
- The address which is used inside the function is used to access the actual argument used in the call.
- If there are any changes made in the parameters, they affect the passed argument.
- For passing a value to the reference, the argument pointers are passed to the functions just like any other value.

Example: Call by reference

```
#include <stdio.h>
int sum (int *n);
void main()
{
    int a = 5;
    printf("\n The value of 'a' before the calling function is = %d", a);
    sum(&a);
    printf("\n The value of 'a' after calling the function is = %d", a);
}
int sum (int *n)
```

```
{
   *n = *n + 20;
   printf("\n value of 'n' in the called function is = %d", n);
}
```

----→ **3Marks**

6. b) Write a C-program to count the number of words, lines in a given paragraph.

Ans:

```
main()
{
char line[81], ctr;
int i,c,
end = 0,
characters = 0,
words = 0,
lines = 0;
printf("KEY IN THE TEXT.\n");


printf("GIVE ONE SPACE AFTER EACH WORD.\n");
printf("WHEN COMPLETED, PRESS 'RETURN'.\n\n");
while( end == 0)
{
/* Reading a line of text */
c = 0;
while((ctr=getchar()) != '\n')
line[c++] = ctr;
line[c] = '\0';
```

-----→ **2Marks**

```
/* counting the words in a line */
if(line[0] == '\0')
break ;
else
{
words++;
for(i=0; line[i] != '\0';i++)
if(line[i] == ' ' || line[i] == '\t')
words++;
}
/* counting lines and characters */
lines = lines +1;
characters = characters + strlen(line);
```

```
        }
        printf ("\n");
        printf("Number of lines = %d\n", lines);
        printf("Number of words = %d\n", words);
        printf("Number of characters = %d\n", characters);
        }
```
-----→ 2Marks

(OR)

7. a) Define Pointer. Explain various memory allocation functions with suitable examples. (6M)

Ans: Pointer: Pointers in C language is a variable that stores/points the address of another variable. A **Pointer in C** is used to allocate memory dynamically i.e. at run time.

The **pointer** variable might be belonging to any of the data type such as int, float, char, double, short etc. ---→ **1Mark**

The C programming language provides several functions for memory allocation and management. These functions can be found in the **<stdlib.h>** header file. These are

1. malloc()

The name "malloc" stands for memory allocation.

The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of type void which can be casted into pointer of any form.

Syntax of malloc()

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

Example:

ptr = (int*) malloc(100 * sizeof(int));

Considering the size of int is 4 bytes, this statement allocates 400 bytes of memory. And, the pointer ptr holds the address of the first byte in the allocated memory**. ----→ 2Marks**

However, if the space is insufficient, allocation fails and returns a NULL pointer.

2. calloc()

The name "calloc" stands for contiguous allocation.

The malloc() function allocates a single block of memory. Whereas, calloc() allocates multiple blocks of memory and initializes them to zero.

Syntax of calloc()

ptr = (cast-type*)calloc(n, element-size);

Example:

ptr = (float*) calloc(25, sizeof(float));

This statement allocates contiguous space in memory for 25 elements each with the size of float. **--→ 2Marks**

3. free()

Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.

Syntax of free()

free(ptr);

This statement frees the space allocated in the memory pointed by ptr. **--→ 1Mark**

7. b) Write a C-program to find GCD of a given number using recursion.

Ans:

```
#include <stdio.h>
int GCD(int n1, int n2);
int main()
{
  int n1, n2;
  printf("Enter two positive integers: ");
  scanf("%d %d", &n1, &n2);
```

-----→ 2Marks

```
  printf("G.C.D of %d and %d is %d.", n1, n2, GCD(n1,n2));
  return 0;
}
int GCD(int n1, int n2)
{
   if (n2 != 0)
      return GCD(n2, n1%n2);
   else
```

```
        return n1;
    }
```
-----→ 2Marks

UNIT-IV

8. a) Explain array of structures with suitable examples. (6M)

Ans: Array of Structure : Structure is collection of different data type. An object of structure represents a single record in memory, if we want more than one record of structure type, we have to create an array of structure or object. As we know, an array is a collection of similar type, therefore an array can be of structure type.

Syntax for declaring structure array

```
struct struct-name
{
     datatype var1;
     datatype var2;
     - - - - - - - - - -
     - - - - - - - - - -
     datatype varN;
};       struct struct-name obj [ size ];
```
-----→ 3Marks

Example for declaring structure array:

```
#include<stdio.h>
struct Employee
{
     int Id;
     char Name[25];
     int Age;
     long Salary;
};
void main()
{
     int i;
     struct Employee Emp[ 3 ];       //Statement   1

     for(i=0;i<3;i++)
     {
     printf("\nEnter details of %d Employee",i+1);
         printf("\n\tEnter Employee Id : ");
```

```
            scanf("%d",&Emp[i].Id);
            printf("\n\tEnter Employee Name : ");
            scanf("%s",&Emp[i].Name);
            printf("\n\tEnter Employee Age : ");
            scanf("%d",&Emp[i].Age);
            printf("\n\tEnter Employee Salary : ");
            scanf("%ld",&Emp[i].Salary);
         }
         printf("\nDetails of Employees");
         for(i=0;i<3;i++)
         printf("\n%d\t%s\t%d\t%ld",Emp[i].Id,Emp[i].Name,Emp[i].Age,Emp[i].Salary);
      }
```

------→ 3Marks

8. b) Define a Structure called employee that consists of name, date of joining, employee id and salary. Write a C-program to read three employee details and sort them in ascending order based on their employee id. (4M)

Ans:

```
 #include <stdio.h>
 /*structure declaration*/
struct employee{
   char    name[30];
   int     empId;
   float   salary;
   float   doj;
};

int main()
{
   int n,i,j,tmp;
   /*declare structure variable*/
   struct employee emp[10];

   /*read employee details*/
```

```
        printf("\nEnter how many employee records you want details :\n");
        scanf("%d",&n);
```

------→2Marks

```
    for(i=0;i<n;i++)
    {
        printf("Name ?:");
            gets(emp[i].name);
        printf("ID ?:");
           scanf("%d",&emp[i].empId);
        printf("Salary ?:");
               scanf("%f",&emp[i].salary);
         printf("Date of joining ?:");
            gets(emp[i].doj);
    }
        /*print employee details*/
              printf("\nEntered detail is:");

     for(i=0;i<n;i++)
    {
        printf("Name: %s"   ,emp[i].name);
        printf("Id: %d"     ,emp[i].empId);
        printf("Salary: %f\n",emp[i].salary);
        printf("Salary: %s\n",emp[i].doj);
    }
      printf("\n Sorting employee records based on their ID's");
        for(i=0;i<n;i++)
              for(j=i+1;j<n;j++)
               if(emp[i].empId> emp[j].empId)
              {
              tmp=emp[i].empId;
              emp[i].empId= emp[j].empId;
              emp[j].empId=tmp;
              }
        return 0;
    }
```

------→2Marks

(OR)

9. a) Write about operations on files with suitable examples. **(6M)**

Ans: Files are used for permanent storage of large amount of data.

The basic operations performed on a file includes

- Naming a file
- Opening a file
- Reading data from a file
- Writing data in to a file
- Appending data to a file
- Closing a file **---→ 2Marks**

Opening a File

The fopen( ) function is used to open a file. It accepts two arguments, the first argument is the name of the file, and the second argument is the mode in which the file is to be opened.

The general form or the syntax of opening a file using the fopen( ) function is

```
FILE *fp;
fp= fopen( "file name", "mode");
```

For example:

```
FILE *fp;
fp=fopen( "system","w");
          ( OR )
File *fp1,*fp2;
fp1=fopen("hello","r");
fp2=fopen("world","w");
```

---→ 2Marks

Closing a File

A file is closed when no more input or output operations are to be performed on it. Closing a file ensures that all outstanding information associated with the file are cleared out from the buffers and releases the space already occupied by the file. All files are closed automatically when a program terminates, so that if the user forgets to close a file, no damage is caused to the file. Closing a file is carried out by using the fclose( ) library function.

The general form or syntax of the fclose( ) function is

```
fclose ( fp);
```

fp is the file pointer, that has been used previously when opening the file.

```
FILE *fp;
fp=fopen( "hello","w");
……………………….
………………………
………………………
fclose(fp);
```

---→ 2Marks

9. b) What is Pre-processor directive? Explain any three with an example. (4M)

Ans: Pre-processor directive is a software program which is used to pre-process the entire code before it passes to the compiler. The following are the various pre-processor directives:

1. Macro expansion directive
2. Conditional compilation directives
3. File inclusion directive **---→ 2Marks**

Explanation **---→2Marks**