Section - C

(Long Answer Questions)

Attempt any three questions out of the following five questions. Each question carries 15 marks. Answer is required in detailed.

Q9(a). How can you give command line arguments? Explain with an example.

Ans. ***Command line arguments in C:*** Main ( ) function of a C program accepts arguments from command line or from other shell scripts by the following commands. They are:

(i) argc
(ii) argv [ ]

where argc – number of arguments in the command line including program name.
argv [ ] – this is carries all the arguments.

Example program for argc ( ) and argv [ ] function in C:

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
int main (int argc, char * argv [ ])
  {
    if (argc! = 5)
      {
        printf ("arguments passed through command line 1 not equal to 5");
        return 1;
      }
    else
    printf ("\n program name: %s\n", argv [0]);
    printf ("1st arg: %s\n", argv [1]);
    printf ("2nd arg; %s\n", argv [2]);
    printf ("3rd arg; %s\n", argv [3]);
    printf ("4th arg; %s\n", argv [4]);
    printf ("5th arg; %s\n", argv [5]);
    return 0;
}
```

Output:

```
Program name: test
1st arg: this
2nd arg: is
3rd arg: a
4th arg: program
5th arg: (null)
```

Q9(b). What are different modes in that you can open a file ? Explain in brief.

Ans. In C programming, the **fopen( ) function** is used to create a new file or to open an existing file.

General Syntax:

```
*fp = FILE * fopen (const char * filename, const char * mode);
```

Here, **filename** is the name of the file to be opened and
mode specifies the purpose of opening the file.

File opening mode in C programming can be of the following types:

Mode	Meaning	If file exists	If file not exists
(i) r	Reading	—	Return NULL
(ii) w	Writing	Overwrite on existing	Create new file
(iii) a	Append	—	Create new file
(iv) r+	Reading + writing	New data is written at the beginning overwriting existing data	Create new file
(v) w+	Reading + writing	Overwriting an existing	Create new file
(vi) a+	Reading + Appending	New data is appended at the end of the file	

Generally, we use to open following types of files in C.

Files type	Extension
C source file	.c
Text file	.txt
Data file	.dat

In C programming, File can be opened in basic 3 modes:

1. Reading mode.
2. Writing mode.
3. Appending mode.

In C programming, writing on all files will overwrite all previous content.

Q10(a). What is 'Macro' ? How is it different from a C variable name ? What Is Macros with Arguments ?

Ans. Macro :

A macro is a name given to a block of statements as a preprocessor directive. Being a preprocessor, the block of code is communicated to the compiler before entering into the actual coding (main ( ) functions). A macro is defined with the preprocessor directive, **# define.**

The advantage of using macro is the execution speed of the program fragment. When the actual code snippet is to be used, it can be substituted by the name of the macro. The same block of statements, on the other hand, need to be repeatedly hard code and when required.

Difference between Macro and Variable:

1. Macros are processed at preprocessing time **whereas** a constant variable is processed at compile time.
2. Macros don't have any scope but constant variables has scope. Macros doesn't have the type checking **whereas** constant variables have type checking.

Macro with Arguments :

Preprocessing directive # define can be used to write macro definitions with parameters as well as in the form below:

define identifier (identifier1, ..., identifier n) token-string.

Again, the token string is optional but, are used in almost every case.

Argumented macro is also called as **function macro** as it looks like calling function.

Everytime whenever the macro name is encountered, the arguments are replaced by the actual arguments from the program.

Advantages of argumented macros:

(a) Arguments are not case-sensitive.

(b) Numeric macro executes faster than the function.

Example of argumented macros:

```
# include<stdio.h>
# define SQU (x) (x * x)
void main ( )
  {
    int x;
    float y;
    x = SQU (3);
    y = SQU (3-1);
    printf ("\n square of integer:%d", x)
    printf ("\n square of float: %f", f);
  }
```

Output:

Square of integer: 9
Square of float : 9.610000

Q10(b). Explain the use of '&' and '|' bitwise operators.

Ans. Bitwise AND (&) operator in C :

– The output of logical AND is 1 if both the corresponding bits of operand is 1.

– If either of bit is '0' or both bits are '0' the output will be '0'.

– It is a binary operator.

For example: Bitwise AND operator on 2 integers 12 & 25.

```
12 =    00001100
25 = & 00011001

        00001000 = 8 (in decimal)
```

The corresponding bits of two inputs are checked and & operation is performed.

```
e.g.  # include <stdio.h>
      int main ( )
        {
          int a = 12, b = 39;
          printf ("output = %d", a & b);
          return 0;
        }
```

Output:

Output = 4

Bitwise OR (|) operator in C :

– The output of bitwise OR is 1 if either of the bit is 1 or both the bits are 1. In C programming, bitwise OR operator is denoted by |.

```
e.g.    12 = 00001100
        25 = 00011001

             00011101 = 29 (in decimal)
```

```
e.g.  # include <stdio.h>
      int main ( )
        {
          int a=12, b=25;
          printf ("output=%d", a|b);
          return 0;
        }
```

Output:

Output = 29

Q11(a). Define a structure 'Time' containing three members hour, minute and second. Develop a program that should assign values to the individual members and display the time the following form 19:35:29.

Ans. Program :

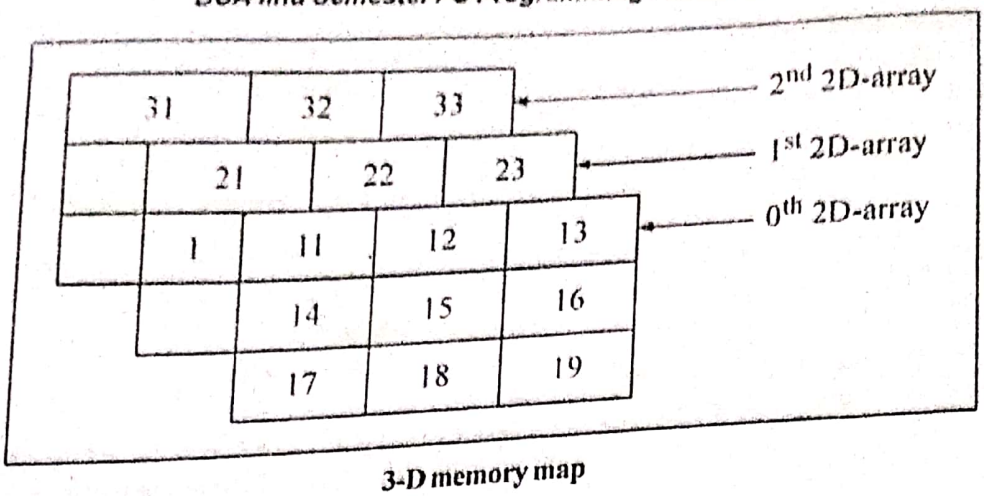
```
# include <stdio.h>
struct time
  {
    int hours;
    int minutes;
    int seconds;
  };
void main ( )
  {
    struct time t1;
    printf("Enter hours for t1:");
    scanf ("%d", t1.hours);
    printf("Enter minutes for t1:");
    scanf("%d", t1.minutes);
    printf("Enter seconds for t1:");
    scanf ("%d", t1.seconds);
    printf("\n Entered time is: %d: %d: %d",
          t1.hours, t1.minutes, t1.seconds);
  }
```

Output:

Enter hours for t1 : 19
Enter minutes for t1 :35
Enter seconds for t1 : 29
Entered times is: 19 : 35 : 29.

Q11(b). What are three-dimensional arrays? How can you initialize them at the time of execution ?

Ans. Three-dimensional array: A 3D-array is essentially an array of arrays. It is an array or collection of 2-D arrays, and a 2D-array is an array of 1D-array.



3-D memory map

The declaration form of 3-D array is:

Datatype array name [size1] [size2] [size3];

Initializing a 3-D array in C: After an array is declared, it must be initialized. Otherwise, it will contain garbage value (any random value). An array can be initialized at either compile time or at runtime.

Compile-time array initialization: Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is type array-name [size] = {list of values};

Run-time array initialization: An array can also be initialized at runtime using scanf( ) function. This approach is usually used for initializing large array or to initialize array with user specified values.

Example for run-time initialization of 3-D array:

```
# include <stdio.h>
# include <conio.h>
void main ( )
{ int number [2][1][3];
  int i, j, k;
  clrscr ( );
  printf ("enter table values \n");
  for (i = 0; i < 2; i++)
  { printf ("table %d: \n", i + 1);
    for (j = 0; j < 1; j++)
    { for (k = 0; k < 3; k++)
      scanf ("%d", & numbers [i][j][k]);
    }
  }
 for (i = 0; i < 2; i++)
 { printf ("table %d: \n", i + 1);
   for (j = 0; j < 1; j++)
   { for (k = 0; k < 3; k++)
     printf ("%d \t", numbers [i][j][k]);
     printf ("\n");
   }
   getch ( );
}
```

Output:

```
Enter table values
table 1:
1
2
3
table 2:
4
5
6
table 1:
1  2  3
table 2:
4  5  6
```

Q12(a). Write a program that can read a text and can count all occurrences of a given word.

Ans. Program to enter a text and count all occurrences of a given string in it.

```
# include <stdio.h>
# include <conio.h>
void main ( )
{ char str[100], find [100];
  int i, j, cnt = 0, flag;
  clrscr ( );
  printf ("enter the text:");
  scanf ("%s", str);
  printf ("enter the string that you want to
        find:");
  scanf ("%s", find);
  for (i = 0; str [i]; i++)
  {
    flag = 0;
```