

CPTC 2016 (ODD)**GROUP – “A”**

1	c	6	a	11	d	16	b
2	a	7	d	12	c	17	a
3	d	8	b	13	a	18	b
4	d	9	c	14	b	19	d
5	b	10	c	15	b	20	c

GROUP – “B”

Q.2. What is function? Explain the need of function with example?

(4)

Ans. : A Function is a self-contained program segment that carries out some specific, well-defined task. Every 'C' program consists of one or more functions. One of these functions must be called main(). If a program contains multiple functions, their definitions may appear in any order, though they must be independent of one another. That is, one function definition cannot be embedded within another. A function will carry out its intended action whenever it is accessed (i.e., Called) from some other portion of the program. Once the function has been carried out its intended action, control will be returned to the point from the function was accessed. Generally, a function will process information passed to it from the calling portion of the program, and return a single value. Information will be passed to the function through special identifier called Arguments or Parameters and returned through return statement.

Need of Functions

Functions are used because of following reasons –

- To improve the readability of code.
- Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
- Debugging of the code would be easier if you use functions, as errors are easy to be traced.
- Reduces the size of the code, duplicate set of statements are replaced by function calls.

For Example:

```
#include <stdio.h>
int addition(int num1, int num2)
{
    int sum;
    sum = num1+num2;
    return sum;
}
```

```
}  
  
int main()  
{  
    int var1, var2, res;  
    printf("Enter number 1: ");  
    scanf("%d",&var1);  
    printf("Enter number 2: ");  
    scanf("%d",&var2);  
    res = addition(var1, var2);  
    printf("Output: %d", res);  
    return 0;  
}
```

Output:

Enter number 1: 100
Enter number 2: 120
Output: 220

Q.2. Explain break and continue statements with one example of each.

(4)

Ans. : Break Statement

The break statement transfers the control out of the block, where it is used. It is also used to exit from a switch or terminate loops. It can be used within a while(), do....while(), for() or a switch() statements.

Syntax : `break ;`

Ex:

```
while (count<=10)  
{  
    scanf("%d",&n);  
    if (n<=0)  
        break;  
    sum = sum + n;  
}
```

Continue Statement

The continue statement is used to bypass the remainder of the current pass through a loop. The loop does not terminate when a continue statement is encountered. Rather, the remainder loop statements are skipped and the computation precedes directly to the next pass through the loop.

Syntax : `continue ;`

Ex:

```
while (count<=10)
{
    scanf("%d",&n);
    if (n<=0)
        continue;
    sum = sum + n;
}
```

Q.3. What is difference between pre-increment and post-increment operators? (4)

Explain with the help of example.

Ans.: Pre-increment operator: A pre-increment operator is used to increment the value of a variable before using it in an expression. In the Pre-Increment, value is first incremented and then used inside the expression.

// Program to demonstrate pre increment operator.

```
#include <stdio.h>
int main()
{
    int x = 10, a;
    a = ++x;
    printf("\nPre Increment Operation\n");
    printf("\n a = %d",a);
    printf("\n x = ",x);
    return 0;
}
```

Output :

Pre Increment Operation

a = 11

x = 11

Post-increment operator: A post-increment operator is used to increment the value of variable after executing expression completely in which post increment is used. In the Post-Increment, value is first used in an expression and then incremented.

Syntax: a = x++;

// Program to demonstrate post increment operator.

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

```
int main()
{
    int x = 10, a;
    a = x++;
```

```

printf("\n\nPost Increment Operation");
printf("\n a = ",a);
printf("\n x = ",x);
return 0;
}

```

Output :

Post Increment Operation

a = 10

x = 11

Q.3. Write a program to print all prime nos. from 1 to 50.

(4)

Ans.:

//Program to check and print all the prime nos. from 1 to 100

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

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

```
main(void)
```

```
{
```

```
int num, j, flag;
```

```
printf("\n\nAll the Prime Nos. from 1 to 100 are:\n\n");
```

```
num=1;
```

```
do
```

```
{
```

```
    j=2;
```

```
    flag=1;
```

```
    while(j<=sqrt(num))
```

```
    {
```

```
        if(num%j==0)
```

```
        {
```

```
            flag = 0;
```

```
            break;
```

```
        }
```

```
        j++;
```

```
    }
```

All the Prime Nos. from 1 to 100 are:

1	2	3	5	7	11	13	17	19	23
29	31	37	41	43	47	53	59	61	67
71	73	79	83	89	97				

```

    if (flag==1)
        printf("%5d",num);
        num++;
    }while(num<=100);
    return 0;
}

```

Q.4. Write a program to exchange values of two variables using call by reference.

(4)

Ans. :

// Program to swap values of two variables without using temporary variable

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

```
int main( )
```

```
{
```

```
    void swap(int *, int *); //Function prototype
```

```
    int a, b;
```

```
    printf("\n\n Enter value for A : ");
```

```
    scanf("%d",&a);
```

```
    printf("\n\n Enter value for B : ");
```

```
    scanf("%d",&b);
```

```
    printf("\n\n Before Swapping A = %d \t B = %d",a,b);
```

```
    //Function call by reference to exchanging values of two variables
```

```
    swap(&a,&b);
```

```
    printf("\n\n After Swapping A = %d \t B = %d",a,b);
```

```
    return 0;
```

```
}
```

//Function Definition to exchange values of two variables

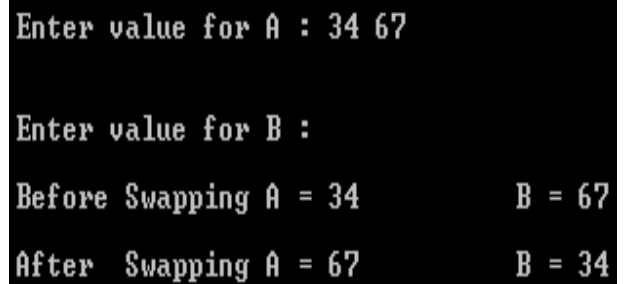
```
void swap(int *x, int *y)
```

```
{
```

```
    int t;
```

```
    t = *x;
```

```
    *x = *y;
```



```

Enter value for A : 34 67

Enter value for B :

Before Swapping A = 34      B = 67

After Swapping A = 67      B = 34

```

```

*y = t;
return;
}

```

Q.4. What is an array? How array can be declared and initialized? write syntax.

(4)

Ans. :

An **ARRAY** is a collection of homogeneous data elements (i.e., of same data type) described by a single variable name. Each individual element of an array is referred by a **Subscripted Variable** or **Integer Constant**. These subscripted variables or indexed variables are enclosed within square brackets and affixed with array variables.

If single subscript is required to refer to an element of an array, the array is known as One Dimensional or Linear Array. If two subscripts are required to refer to an element of an array, the array is known as Two Dimensional Array and so on.

Note: The no. of dimensions supported in an array depends upon RAM capacity and operating system used in the computer system.

Declaring 1-D ARRAY :→

Syntax :

```

DataType  ArrayName [ dim-1 ][ dim-2 ] ... [ dim-n ];

```

An array in a program, must contain following information :→

- ◆ The type and name of the array.
- ◆ The number of subscripts in the array.
- ◆ The total number of memory locations to be reserved, or more specifically, the maximum value of each subscript.

Ex. :--

```

#define    MAX    100

int    mat[MAX];

float  num[20];

char   subject[12];  etc.

```

The first element position in an array is accessed as 0 (zero) and last as (MAX – 1).

Ex. :--

```

num[10];    //The first element is num[0] and last is num[9].

```

Initialising an ARRAY

One Dimensional Array

```

int    num[5] = { 16, -5, -12, 8, 10 };

int    n[ ] = { 2, 4, 7, 1, 6, 8, -9, 10, -45 };

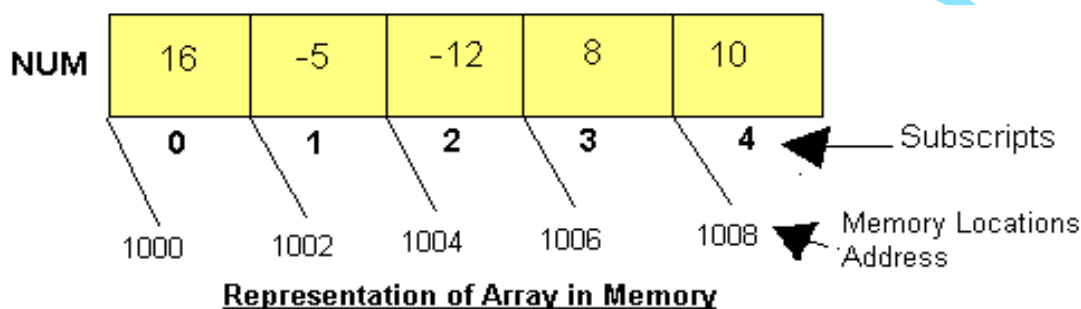
```

In this case, the size of the array will be assumed as the number of elements supplied with it.

`int a[10] = { 1, 2, 4, 7, 8, -3, 9 };` In this case, the Garbage values will be accepted for rest of the elements, because the sufficient data have not been supplied to the array according to its declared size.

Let us understand the array within the memory through this figure :--

`int num[5] = { 16, -5, -12, 8, 10 };`



Formula to Calculate the Address of an Element in 1-D Array

= Base Address + Column No. x Size of (Data Type)

// Data Type Size is implicitly assumed in the formula

Ex. : `&num[3] = 1000 + 3 x 2 = 1006`

A Two Dimensional Array can be treated as array of two parts **ROW** and **COLUMN**. The first subscript represents the ROW position, whereas the second subscript represents the COLUMN position. Similarly, a Three Dimensional Array can be treated as array of two dimensional arrays, in which last two dimensions are treated as one dimensional array.

Ex. :--

```
# define ROW 50
```

```
# define COL 25
```

```
# define DIM1 3
```

```
# define DIM2 3
```

```
# define DIM3 3
```

```
int matrix[ROW][COL];
```

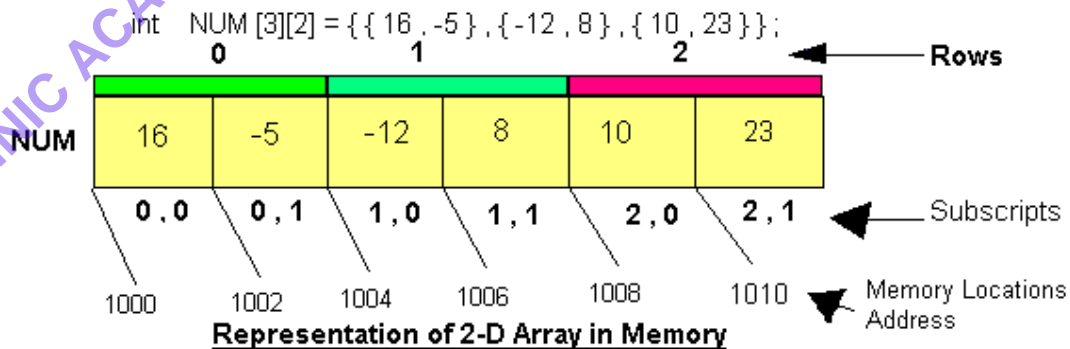
```
int table[20][10];
```

```
char name[100][40]; // 100 String capacity each of 40 letters length
```

```
int num[DIM1][DIM2]; /* Declaration of a 2-D Array */
```

```
int ANK[DIM1][DIM2][DIM3]; /* Declaration of a 3-D Array */
```

Now let us, understand the 2-D Array representation within memory through the following Diagram:



Formula to Calculate the Address of an Element in 2-D Array

= Base Address + Row No. x No. of Columns in a row x Size of (Data Type)

+ Column No. x Size of (Data Type) // Data Type Size is implicitly assumed in the formula

Ex. : $\&\text{num}[1][1] = 1000 + 1 \times 2 \times (2) + 1 \times (2) = 1006$

Q.5. Explain declaration and initialization of string variable using example.

(4)

Ans. :

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

```
char college_nm[] = "POLYTECHNIC";
```

index	0	1	2	3	4	5	6	7	8	9	10	11
-------	---	---	---	---	---	---	---	---	---	---	----	----

college_nm	P	O	L	Y	T	E	C	H	N	I	C	'\n'
------------	---	---	---	---	---	---	---	---	---	---	---	------

address												
---------	--	--	--	--	--	--	--	--	--	--	--	--

Declaration of strings: Declaring a string is as simple as declaring a one-dimensional array. Below is the basic syntax for declaring a string.

```
char str_name[size];
```

In the above syntax str_name is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store. Please keep in mind that there is an extra terminating character which is the Null character ('\0') used to indicate the termination of string which differs strings from normal character arrays.


```
Char state[]={'B','I','H','A','R'};
```

index

0	1	2	3	4								
---	---	---	---	---	--	--	--	--	--	--	--	--

state

B	I	H	A	R								
---	---	---	---	---	--	--	--	--	--	--	--	--

address

--	--	--	--	--	--	--	--	--	--	--	--	--

In the above example the character array state is initialized with individual set of characters enclosed within pair of single quotes with comma separator grouped by pair of curly braces. In this example the size of array will be automatically decided as the count of no. of characters in the set.

Q.5. Write short notes on following

(4)

(a) Difference between while and do—while statement

(b) Nested If--else statement.

Ans. :

(a) Difference between while and do—while statement

	While Loop	Do....while Loop
1	The while loop evaluates the condition first and then executes the statement.	The do...while loop executes the statement first before evaluating the condition.
2	The condition is specified at the beginning of the loop.	The condition is not specified until after the body of the loop.
3	The body is executed only if a certain condition is met and it terminates when the condition is false.	The body is always executed atleast once regardless of whether the condition is met

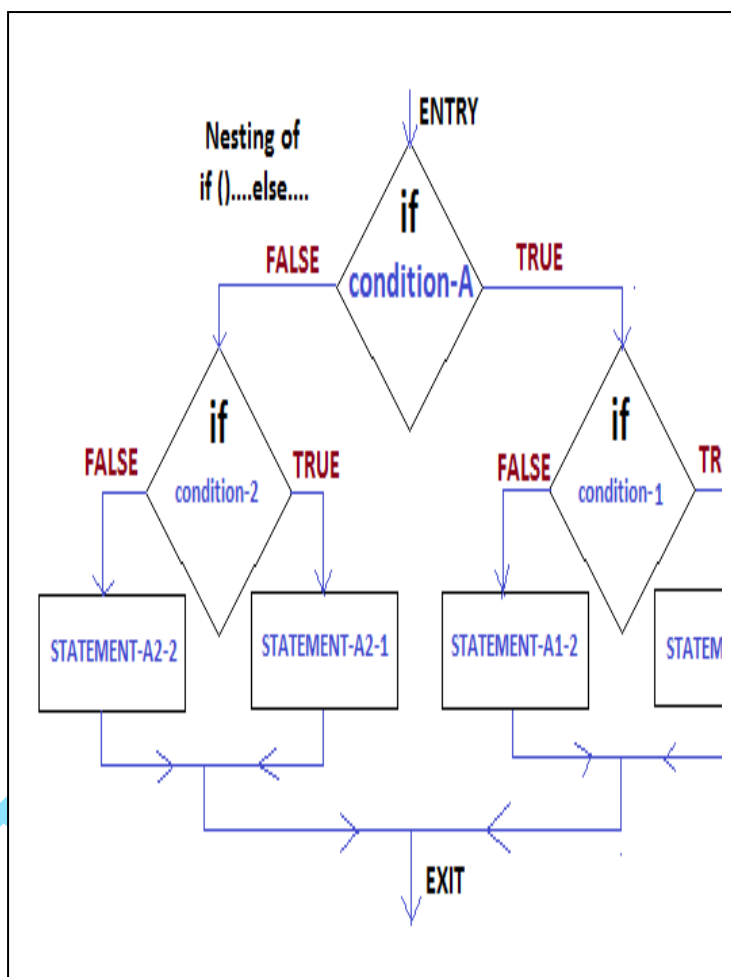
(b) Nested If--else statement.

Using the if()... else... statement in the if block or in the else block is referred as Nesting if... else... statements. nesting can be done in both if and else part. The outer if(condition) is tested first, on getting TRUE, the inner if(condition) is tested thereafter. On getting FALSE, the inner if(condition) in else part is executed. The construct looks like –

```

if (condition A)
{
    if (condition 1)
        statement-A1-1;
    else
        statement-A1-2;
}
else
{
    if(condition 2)
        statement-A2-1;
    else
        statement-A2-2;
}

```



Ex:-

```

if (grade=='A' || grade == 'a')
    if(precent >= 95.0)
        printf ("\n\nExcellent\n\n");
    else
        printf ("\n\nWork Hard for Grade A\n\n");
else
    printf ("\n\nNot good ! Do Work more harder\n\n");

```

Q.6. Write a program in C to print first 10 fibonacci numbers.

(4)

Ans.

//Program to print first 10 Fibonacci Nos using for() Loop

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

```
int main()
```

```
{
```

```

int a=1,b=1,c,j;
printf("\n\nFirst 10 Fibonacci Nos are :\n\n");
printf("\n%5d%5d",a,b);
for(j=2;j<=10;j++)
{
    c = a + b;
    printf("\n%5d",c);
    a = b;
    b = c;
}
return 0;
}

```

First 10 Fibonacci Nos are :

1 1 2 3 5 8 13 21 34 55 89

Q.6. Define Compilation and Interpretation. How do these two processes differ? Comment. (4)

Ans. :

Compilation : The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code. The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

Interpretation : Interpretation operation is performed by an interpreter (a system program) that is used to directly execute program instructions written using one of the many high-level programming languages. The interpreter transforms the high-level program into an intermediate language that it then executes, or it could parse the high-level source code and then performs the commands directly, which is done line by line or statement by statement.

Difference Between Compilation and Interpretation :

- (i) **Interpretation:** An interpreter reads a program line by line, reading every expression and gives output if the program is correct. It stops/gives error when it encounters the first mistake in any line of code. This process is interpretation.

Compilation: In this, a compiler reads the program in one go and gives output if the program is correct else an error. Both, Interpreter & Compiler are special programs that convert a program into machine language.

- (ii) **Interpretation** is for Scripting languages and it translates code to machine language Line-by-line

while **Compilation** is for programming languages and it translates codes at once

- (iii) Interpretation is reading or checking the code line by line and if any error is found you can't go further before correcting it.

Whereas Compilation is reading and checking the whole code at once and reporting all errors found in it.

GROUP - "C"

Q.7. Explain logical operator. Write a C program to check given year is leap year or not. (8)

Ans.: A logical operator is used to form a complex test condition in order to take a decision. An expression is formed by logical operators which combine two more relational expressions is termed as Logical Expression or Combined Relational Expression. A logical expression produces logical value Boolean True (1) or False (0). Logical operators produce (return) value according to the logical Truth Table.

LOGICAL OPERATORS			
OPERATOR	OPERATION PERFORMED	PRECEDENCE	EXAMPLE
!	Negation or Logical NOT	Highest	! digit
&&	Conjunction or Logical AND	Intermediate	(gender=='F') && (age>18)
	Disjunction or Logical OR	Lowest	(status=='1') (age>=30)

*/*Program to test whether a year is leap or not*/*

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

```
int main()
```

```
{
```

```
int year;
```

```
printf ("\n\nEnter year : ");
```

```
scanf ("%d",&year);
```

```
if (year%100==0)
```

```
if (year%400==0)
```

```
printf ("\n\n%d is Leap year",year);
```

```
else
```

```
printf ("\n\n%d is Not leap",year);
```



```
Enter year : 2020
2020 is Leap year
```

```

else
    if (year%4==0)
        printf ("\n\n%d is Leap year",year);
    else
        printf ("\n\n%d is Not leap",year);
return 0;
}

```

Q.7. What is algorithm. Write characteristics of algorithm. Write an algorithm to find smallest of three numbers. (8)

Ans. :

Algorithms are one of the most basic tools that are used to develop the problem solving logic. An algorithm can be defined as a finite sequence of explicit and unambiguous instructions in stepwise logical manner up to a finite number of times until the solution of the problem is achieved. However, algorithms can have steps that repeat (iterate) or require decisions (logic and comparison) until the task is completed.

Characteristics of Algorithm

1. **Finiteness.** An algorithm must always terminate after a finite number of steps.
2. **Definiteness.** Each step of an algorithm must be precisely defined; the actions to be carried out must be rigorously and unambiguously specified for each case.
3. **Input.** An algorithm has zero or more inputs, i.e, quantities which are given to it initially before the algorithm begins.
4. **Output.** An algorithm has one or more outputs i.e, quantities which have a specified relation to the inputs.
5. **Effectiveness.** An algorithm is also generally expected to be effective. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time.

Q.8. Explain the concept of recursion. Write a program to accept two positive integers and computes their GCD using recursive function. (8)

Ans. RECURSION means Self-Reference. A Recursive function is a function whose definition is based upon itself. In other words, a function containing either a call statement to itself or a call statement to a second function that eventually may result in a call statement back to the original function, then that function is called a Recursive function.

In order that the function should not continue indefinitely, a recursive function must have the following properties :--

- 1) There must be certain criteria, called Base Criteria, for which the function does not call itself.
- 2) Each time the function does call itself (directly or indirectly), it must be closer to the base criteria, i.e., it uses an argument(s) smaller than the one it was given.

When a recursive program is executed, the recursive function calls are not executed immediately. Rather, they are placed on a Stack until the condition that terminated the recursion is encountered. The function calls are then executed in reverse order, as they are Popped off the stack.

If a recursive function contains local variables, a different set of local variables will be created during each call. The variables will be of the same name as declared, different set of values each time the function is executed. Each set of values will be stored on the stack, so that they will be available as the recursive process unwinds.

//Program to find GCD of Two Numbers using Recursion

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

```
int hcf(int n1, int n2); // Function Declaration
```

```
int main() {
```

```
    int n1, n2;
```

```
    printf("\n\nEnter two positive integers: ");
```

```
    scanf("%d %d", &n1, &n2);
```

```
    printf("\n\nG.C.D of %d and %d is %d.", n1, n2, hcf(n1, n2));
```

```
    return 0;
```

```
}
```

// Recursive Function Definition

```
int hcf(int n1, int n2) {
```

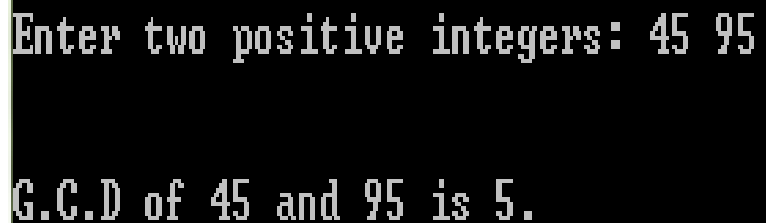
```
    if (n2 != 0)
```

```
        return hcf(n2, n1 % n2);
```

```
    else
```

```
        return n1;
```

```
}
```



Q.8. What do you mean by modularization? Explain strcmp() and strcpy() functions. (8)

Ans. : Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

strcmp() Function

The strcmp() function takes two strings and returns an integer. The strcmp() compares two strings character by character. If the first character of two strings is equal, the next character of two strings are compared. This continues until the corresponding characters of two strings are different or a null character '\0' is reached. It is defined in the **string.h** header file.

Syntax: int strcmp (const char* str1, const char* str2);

Parameters

- str1 – This is the first string to be compared.
- str2 – This is the second string to be compared.

Return Value

This function return values that are as follows –

Return Value	Remarks
0	if both strings are identical (equal)
negative	if the ASCII value of the first unmatched character is less than the second.
positive integer	if the ASCII value of the first unmatched character is greater than the second.

// Example: strcmp() Function Program

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
    int result;
    // comparing strings str1 and str2
    result = strcmp(str1, str2);
    printf("strcmp(str1, str2) = %d\n", result);
    // comparing strings str1 and str3
    result = strcmp(str1, str3);
    printf("strcmp(str1, str3) = %d\n", result);
    return 0;
```

```
}
```

Output

```
strcmp(str1, str2) = 32
```

```
strcmp(str1, str3) = 0
```

strcpy() Function

The strcpy() function copies the string pointed by source (including the null character) to the destination. The strcpy() function also returns the copied string. The strcpy() function is defined in the **string.h** header file.

Syntax : char* strcpy(char* dest, const char* src);

Parameters

- dest – This is the pointer to the destination array where the content is to be copied.
- src – This is the string to be copied.

// Example: strcpy() Function Program

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

```
#include <string.h>
```

```
int main() {
```

```
    char str1[20] = "C programming";
```

```
    char str2[20];
```

```
    // copying str1 to str2
```

```
    strcpy(str2, str1);
```

```
    puts(str2); // C programming
```

```
    return 0;
```

```
}
```

Output

C programming

Q.9. Explain two dimensional array. Write a program to find product of two matrices. (8)

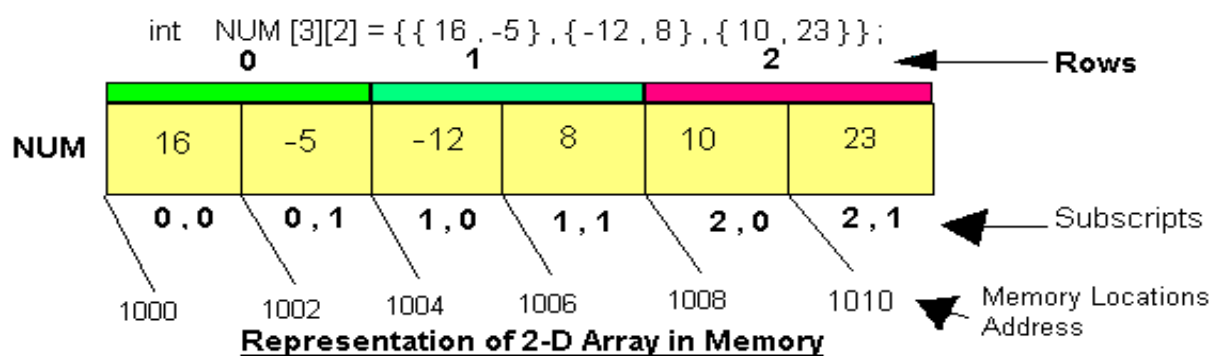
Ans. : A Two Dimensional Array can be treated as array of two parts **ROW** and **COLUMN**. The first subscript represents the ROW position, whereas the second subscript represents the COLUMN position. Similarly, a Three Dimensional Array can be treated as array of two dimensional arrays, in which last two dimensions are treated as one dimensional array.

Ex. :--

```
# define      ROW 50
# define      COL 25
# define      DIM1 3
# define      DIM2 3
# define      DIM3 3

int  matrix[ROW][COL];
int  table[20][10];
char name[100][40];    // 100 String capacity each of 40 letters length
int  num[DIM1][DIM2];    /* Declaration of a 2-D Array */
int  ANK[DIM1][DIM2][DIM3]; /* Declaration of a 3-D Array */
```

The following Diagram showing the memory representation of 2-D array:



Formula to Calculate the Address of an Element in 2-D Array

= Base Address + Row No. x No. of Columns in a row x Size of (Data Type)

+ Column No. x Size of (Data Type) // Data Type Size is implicitly assumed in the formula

Ex. : &num[1][1] = 1000 + 1 x 2 x (2) + 1 x (2) = 1006

//Multiplication of 2 Matrices

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

```
#include<stdlib.h>
```

```
#define MAX 100
```

```
int main()
```

```
{
```

```

int a[MAX][MAX], b[MAX][MAX];
int p[MAX][MAX], i, j, k, r1, c1, r2, c2;
printf("\nEnter Dimension of 1st Matrix : ");
scanf("%d %d", &r1, &c1);
printf("\nEnter Dimension of 2nd Matrix : ");
scanf("%d %d", &r2, &c2);

```

//Compatibility Check

```

if(r2!=c1)
{
    printf("\nMatrices Size do match...");
    printf("\nProgram Aborting....");
    exit(1);
}

```

//Input in Matrix

```

printf("\nEnter %d numbers in 1st Matrix : ", r1*c1);
for(i=0; i<r1; i++)
    for(j=0; j<c1; j++)
        scanf("%d", &a[i][j]);
printf("\nEnter %d numbers in 2nd Matrix : ", r2*c2);
for(i=0; i<r2; i++)

```

```

    for(j=0; j<c2; j++)
        scanf("%d", &b[i][j]);

```

//Multiplying two matrices

```

for(i=0; i<r1; i++)
    for(j=0; j<c2; j++)
    {
        p[i][j]=0;
        for(k=0; k<c1; k++)
            p[i][j]+=(a[i][k] * b[k][j]);
    }

```

```

Enter Dimension of 1st Matrix : 3 2
Enter Dimension of 2nd Matrix : 2 4
Enter 6 numbers in 1st Matrix : 10 20 30 40 50 60
Enter 8 numbers in 2nd Matrix : 2 3 4 5 6 7 8 9

1st Matrix A[3][2]:
10  20
30  40
50  60

2nd Matrix B[2][4]:
2  3  4  5
6  7  8  9

Product Matrix P[3][4]:
140 170 200 230
300 370 440 510
460 570 680 790

```

//Displaying Output

```

printf("\n\n1st Matrix A[%d][%d]:\n",r1,c1);
for(i=0;i<r1;i++)
{
    for(j=0;j<c1;j++)
        printf("%5d",a[i][j]);
    printf("\n");
}
printf("\n\n2nd Matrix B[%d][%d]:\n",r2,c2);
for(i=0;i<r2;i++)
{
    for(j=0;j<c2;j++)
        printf("%5d",b[i][j]);
    printf("\n");
}
printf("\n\nProduct Matrix P[%d][%d]:\n",r1,c2);
for(i=0;i<r1;i++)
{
    for(j=0;j<c2;j++)
        printf("%5d",p[i][j]);
    printf("\n");
}
return 0;
}

```

Q.9. Write short notes on the following**(8)****(a) Difference between array and variable****(b) Difference between loading and linking****Ans. :****(a) Difference between array and variable :**

- Array is the set of an multiple values where as variable can store single value at a time.
- The difference between the definition of array and ordinary variable is the, array is always declared, initialized, and accessed using subscript whereas ordinary variable do not have any subscript.

- The syntax for ordinary variable definition is `data_type v1, v2,;`
- And the syntax for array variable is `data_type v1[N1], v2[N2], ...;` where `v1, v2` are name of variable and `N1, N2` are the integer constants indicating the maximum size of array.

(b) Difference between loading and linking

1. The key difference between linking and loading is that the linking generates the executable file of a program whereas, the loading loads the executable file obtained from the linking into main memory for execution.
2. The linking intakes the object module of a program generated by the assembler. However, the loading intakes the executable module generated by the linking.
3. The linking combines all object modules of a program to generate executable modules it also links the library function in the object module to built-in libraries of the high-level programming language. On the other hand, loading allocates space to an executable module in main memory.

Q.10. What is structure? How it is declared? Give a suitable example to explain. (8)

How do you access member of a structure?

Ans. : STRUCTURE

To store related information in the form of record, 'C' provides a special data type : struct. More than one different data items can be stored and accessed through structure and is accessed and stored by a single variable name.

The structure should be first declared before use. The syntax for declaring a structure is :

Syntax :

```
struct structure_name
{
    data_type    variable 1;
    data_type    variable 2;
    data_type    variable 3;
    :
    :
    :
    data_type    variable n;
};
```

```
struct structure_name
{
    data_type    variable 1;
    data_type    variable 2;
    data_type    variable 3;
    :
    :
    :
    data_type    variable n;
} variable ;
```

Where each declaration of variable inside the structure template is called Member of Structure.

Ex.:-

```
struct student_record
{
    char    roll[10];
    char    name[30];
    float    course_fee;
};
```

```
struct student_rec    studrec;
```

```
struct student_rec    studrec = { "1010023457", "AJAY KUMAR", 4500.75 } ;
```

Accessing Structure Elements :

The elements of a structure variable are accessed using dot operator “.”. It refers an individual element of a structure.

Syntax: `structure_variable_name . structure_member_name ;`

To use for Input :-

1 Record

1010023457	AJAY KUMAR	4500.75
------------	------------	---------

Ex. :-- `studrec.roll; studrec.name; &studrec.course_fee; etc.`

To use for Access :-

Ex. :-- `studrec.roll; studrec.name; studrec.course_fee; etc.`

Arrays and Structures :

A Structure is a collection of related information in the form of Record. The same structure can also be declared through an Array variable to hold a number of different records of same type.

Syntax :	<pre> struct structure_name { data_type variable 1; data_type variable 2; data_type variable 3; : : : data_type variable n; }; </pre>	<pre> struct structure_name { data_type variable 1; data_type variable 2; data_type variable 3; : : : data_type variable n; } variable[m] ; </pre>
-----------------	---	--

Where each declaration of variable inside the structure template is called Member of Structure. The variable m indicates an integer number to set the number of records held by the array variable.

Ex. :--	<pre> struct student_record { char roll[10]; char name[30]; float course_fee; }; struct student_rec studrec[10]; int i; </pre>	<pre> struct student_record { char roll[10]; char name[30]; float course_fee; } studrec[10]; </pre>
---------	--	--

To take input to the array of structure :--

```
for ( j = 0 ; j < 10 ; j++ )
{
    scanf ("%s",studrec[ j ].roll );
    gets (studrec[ j ].name );
    scanf ("%f", &studrec[ j ].course_fee );
}
```

3 Records

0023141245	MANOJ KUMAR	2300.55	0011238871	PRAVEEN	4577.25	0123145229	SURAJ SUMAN	7000.00
Record #0			Record #1			Record #2		

To Access the array of structure :--

```
for ( j = 0 ; j < 10 ; j++ )
    printf ("%s \t %s \t %.2f",studrec[ j ].roll,
            studrec[ j ].name, studrec[ j ].course_fee );
```

Pointers and Structures :

A Structure can be declared through a pointer variable. Here, pointer variable holds the base address of the structure in side memory. Every variable of structure can be accessed through pointer. The “→” operator is used instead of “.” operator while accessing the variables of the structure.

Syntax : struct structure_name structure_variable , *pointer_variable ;

Ex. :-- struct student_record stud_rec, *st ;

 St = &stud_rec ; /* Initialising Pointer variable */

To take input to the elements of structure through pointer :--

st→roll, st→name, &st→course_fee, (st+j)→roll, (st+j)→name, &(st+j)→course_fee

To Access the elements of structure through pointer :--

st→roll, st→name, st→course_fee, *(st+j)→roll, *(st+j)→name, *(st+j)→course_fee

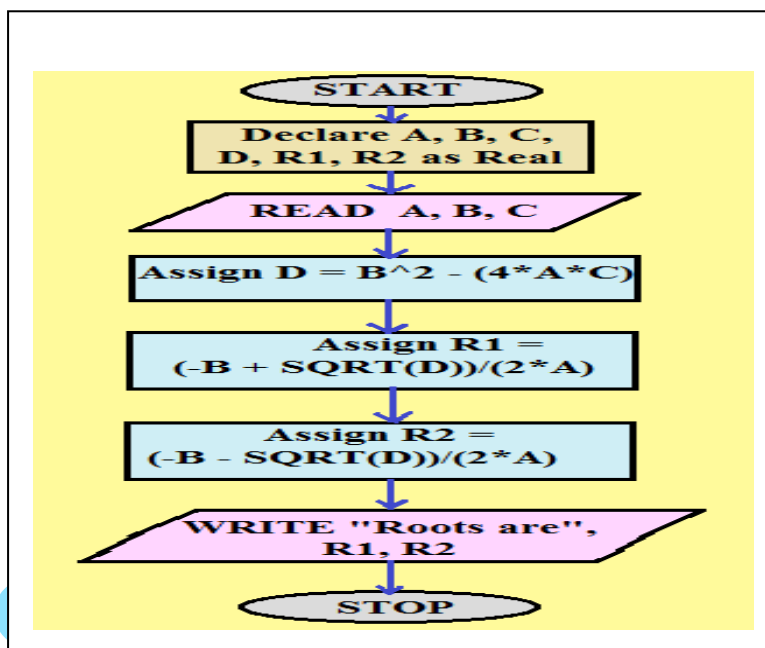
Q.10. Draw a flowchart to find roots of a quadratic equation.

(8)

Ans. :

REM Calculation Square Roots of Quadratic Equation

- Step-1 START
- Step-2 Declare A, B, C, D, R1, R2 as Real
- Step-3 READ A, B, C
- Step-4 Assign $D = B^2 - 4*A*C$
- Step-5 Assign $R1 = (-B + \text{SQRT}(D)) / (2*A)$
- Step-6 Assign $R1 = (-B + \text{SQRT}(D)) / (2*A)$
- Step-7 WRITE "Roots are = ", R1, R2
- Step-8 STOP



Q.11. Write a C program to accept empcode, empname, basic salary of the employees and compute their gross salary. Formula $\text{Gross_Salary} = \text{Basic} + \text{DA} + \text{HRA}$.

(8)

Ans. :

//Program to print Employees Details

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

```
typedef struct employee_records
```

```
{
```

```
    char empcode[10];
```

```
    char empname[20];
```

```
    float basic_sal;
```

```
    float hra;
```

```
    float da;
```

```
    float gross_sal;
```

```
}Employee;
```

```
main()
```

```
{
```

```
    Employee emp[100];
```

```
int n, i;

printf("\nHow many Employees Details : ");

scanf("%d",&n);

printf("\nEnter Employees Details below :\n");

for(i=0;i<n;i++)
{
    fflush(stdin);

    printf("\nEnter Employee Code : ");

    gets(emp[i].empcode);

    fflush(stdin);

    printf("\nEnter Employee Name : ");

    gets(emp[i].empname);

    printf("\nEnter Employee's Basic Salary : ");

    scanf("%f",&emp[i].basic_sal);

    printf("\n");

    emp[i].hra = emp[i].basic_sal * 0.10;

    emp[i].da = emp[i].basic_sal * 0.85;

    emp[i].gross_sal = emp[i].basic_sal + emp[i].da + emp[i].hra;

}

printf("\nThe Employee's Record is as :\n\n");

for(i=0;i<n;i++)

    printf("%3d. %-13s%-25s%10.2f%10.2f%10.2f%10.2f\n",i+1,emp[i].empcode,
        emp[i].empname,emp[i].basic_sal,emp[i].da,emp[i].hra,emp[i].gross_sal);

return 0;

}
```


OUTPUT:

```

How many Employees Details : 5
Enter Employees Details below :
Enter Employee Code : MD99001
Enter Employee Name : RAJESH KUMAR SINHA
Enter Employee's Basic Salary : 150000.00

Enter Employee Code : PM99002
Enter Employee Name : SHRUTI SHARMA
Enter Employee's Basic Salary : 100000.00

Enter Employee Code : SA99005
Enter Employee Name : VISHAL KUMAR YADAV
Enter Employee's Basic Salary : 72950.00

Enter Employee Code : PR99008
Enter Employee Name : MANISHA KUMARI
Enter Employee's Basic Salary : 65000.00

Enter Employee Code : OS99010
Enter Employee Name : RAMAN KUMAR
Enter Employee's Basic Salary : 20500.00

```

The Employee's Record is as :

1.	MD99001	RAJESH KUMAR SINHA	150000.00	127500.00	15000.00	292500.00
2.	PM99002	SHRUTI SHARMA	100000.00	85000.00	10000.00	195000.00
3.	SA99005	VISHAL KUMAR YADAV	72950.00	62007.50	7295.00	142252.50
4.	PR99008	MANISHA KUMARI	65000.00	55250.00	6500.00	126750.00
5.	OS99010	RAMAN KUMAR	20500.00	17425.00	2050.00	39975.00

Q.11. Write short notes on the following -

(8)

(a) Pointer declaration and initialization

(b) Program compilation and debugging

Ans. :

(a) Pointer declaration and initialization :

Pointer variables must be declared before use. When a pointer variable is declared, the variable name must be preceded by an **Asterisk (*)**. This identifies the fact that the variable is a pointer. The data type that appears in the declaration refers to the Object of the pointer, i.e., the item that is stored in the address represented by the pointer, rather than the pointer itself.

Syntax :

data-type *pointer-variable-name ;

Ex.:--
`int *ptr;`
`int x, *px;`
`px = &x; // Storing address of x into pointer variable px`

Initialization of pointer variable :

Datatype *pointer_var = constant;

Ex: `int *p = 45;`
`printf("\nValue through P : %d",*p);`

(b) Program compilation and debugging :

Program Compilation : The compilation is a method whereby the source code is converted into object code. It is achieved with compiler assistance. The compiler tests the source code for syntactic or structural errors and produces the object code if the source code is error-free.

In object code or machine code, the compilation method converts the source code that has been taken as an input. The method of compiling can be divided into four stages, i.e., pre-processing, compilation, assembly, and linking.

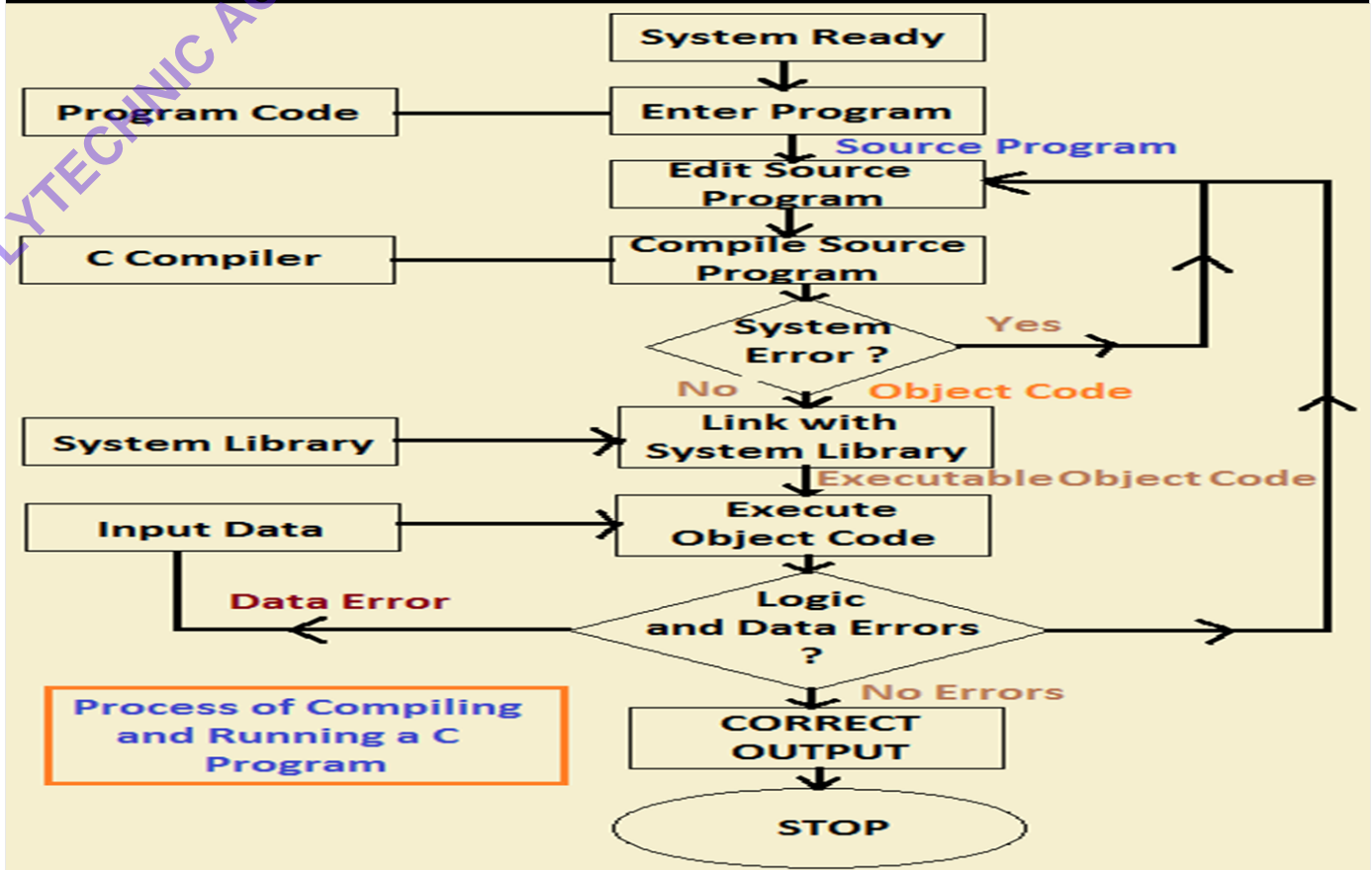
A pre-processor selects the raw data as an input and extracts all the statements from the code. The pre-processor takes in and interprets the pre-processor instruction. For example, if the directive is accessible in the program `<stdio.h>`, then the pre-processor interprets the directive and replaces it with the content of the file 'stdio.h.'

Following are the steps that a program goes through until it is translated into an executable form:

- Preprocessor
- Compiler
- Assembler

- Linker

Flowchart of Process of Compiling and Running a C Program



Program Debugging : Debugging is the process of detecting and removing of existing and potential errors (also called as ‘bugs’) in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another. Sometimes it takes more time to debug a program than to code it.

To debug a program, user has to start with a problem, isolate the source code of the problem, and then fix it. A user of a program must know how to fix the problem as knowledge about problem analysis is expected. When the bug is fixed, then the software is ready to use. Debugging tools (called debuggers) are used to identify coding errors at various development stages. They are used to reproduce the conditions in which error has occurred, then examine the program state at that time and locate the cause. Programmers can trace the program execution step-by-step by evaluating the value of variables and stop the execution wherever required to get the value of variables or reset the program variables. Some programming language packages provide a debugger for checking the code for errors while it is being written at run time.

Here’s the debugging process:

1. Reproduce the problem.
2. Describe the bug. Try to get as much input from the user to get the exact reason.
3. Capture the program snapshot when the bug appears. Try to get all the variable values and states of the program at that time.
4. Analyse the snapshot based on the state and action. Based on that try to find the cause of the bug.
5. Fix the existing bug, but also check that any new bug does not occur.
