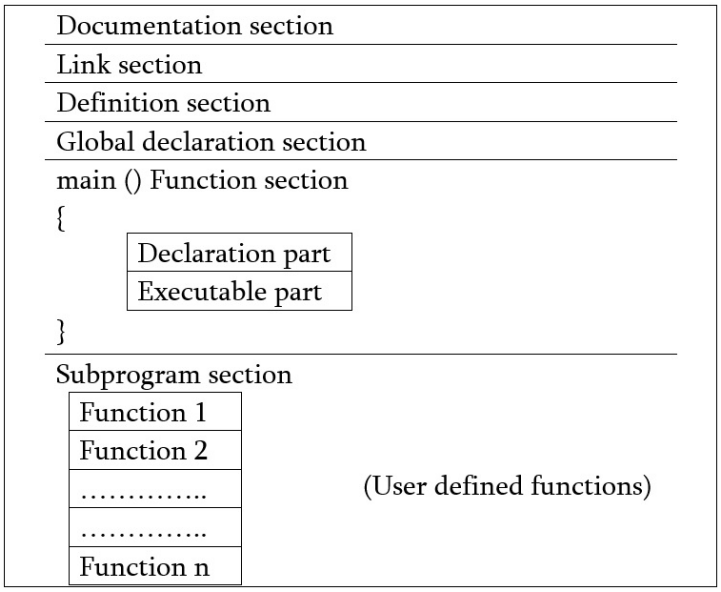
C Programming for Problem Solving (18CPS13/23) – 25 Questions

Model Answer by Dr. P. N. Singh, Professor(CSE)

1. Explain the structure of C program with block diagram and example.

Ans:



- **Documentation Section:** This section is used to write Problem, file name, developer, date etc in comment lines within /*….*/ or separate line comments may start with // . Compiler ignores this section. Documentation enhances the readability of a program.
- **Link section :** To include header and library files whose in-built functions are to be used. Linker also required these files to build a program executable. Files are included with directive # include
- **Definition section:** To define macros and symbolic constants by preprocessor directive #define
- **Global section:** to declare global variables – to be accessed by all functions
- **main() i**s the user defined function which is recognized by the compiler first. So, all C program must have user defined function main() { ……. }. It should have declaration part first then executable part.
- **Sub program section:** There may be other user defined functions to perform specific task when called.

```
/* Example: a program to find area of a circle – area.c
                          - Documentation Section*/

#include <stdio.h>                        /* - Link/Header Section */

#define PI 3.14                         /* definition/global section*/

int main()                              /* main function section */
{
     float r, area;                    /* declaration part */

     printf("Enter radius of the circle : "); /* Execution part*/
     scanf("%f", &r);
     area=PI*r*r; /* using symbolic constant PI */
     printf("Area of circle = %0.3f square unit\n", area);
     return (0);
}
```

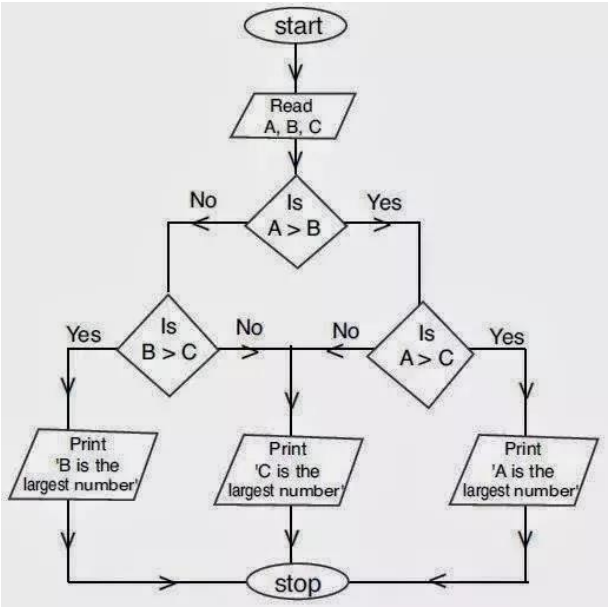
2. Write an algorithm, draw flowchart & write program to find the biggest of three numbers. (using nested if construct)

Ans:

Example of a Algorithm to find maximum in 3 numbers (USING NESTED IF):

1. START
2. INPUT A, B, C
3. IF A > B THEN
    a. IF A > C THEN
        i. MAX=A
    b. ELSE
        i. MAX=C
    c. ENDIF
4. ELSE
    a. IF B > C THEN
        i. MAX=B
    b. ELSE
        i. MAX=C
    c. ENDIF
5. ENDIF
6. PRINT MAX
7. END.

Flowchart




```
/* Example program to find maximum in 3 integers using nested
if*/
#include <stdio.h>
int main()
{
    int n1,n2,n3,max;
    printf("Enter 3 numbers : ");
    scanf("%d%d%d",&n1,&n2,&n3);
    if(n1 > n2)
         if(n1 > n3)
              max=n1;
         else
              max=n3;
   else
         if(n2 > n3)
              max=n2;
         else
              max=n3;
   print("Maximum = %d\n",max);
   return (0);
 }
```

3. Define C tokens. List and explain different C tokens.

Ans:

- Valid Characters (alphabets, numbers or symbols) used in C program are known as C – tokens.
- Each and every smallest individual units in a C program are known as C tokens.
- C tokens are the basic buildings blocks in C language which are constructed together to write a C program.

C tokens are of six types. They are,

I. Keywords (eg: int, while),
II. Identifiers (eg: main, total),
III. Constants (eg: 10, 20),
IV. Strings (eg: “total”, “hello”),
V. Special symbols (eg: (), {}),
VI. Operators (eg: +, /,-,*)

Example program for C - Tokens

```
int main()
{
  int x, y, sum;
  x = 10, y = 20;
  sum = x + y;
  printf ("Total = %d \n", sum);
}
```

where,

- main – identifier
- {,}, (,) – delimiter
- int – keyword
- x, y, sum – identifier
- **main, {, }, (,), int, x, y, sum – tokens**

4. What is an identifier? Explain the rules to construct identifier in C language. Give examples of valid and invalid variables.

Ans:

A variable is an identifier for which a value can be assigned and changed. It is name of the location of memory where some value is stored temporarily.

Rules to construct identifier/variable:

- They may contain A-Z, a-z, and _ (underscore) (no blanks)
- An identifier name but must be started with an alphabet or _ (specific purposes)
- Maximum length of name of identifier should be 31 characters (ANSI C standards) however in newer versions length of identifier name may be more.
- Keywords & compiler constants should not be identifier name

Valid identifiers: num1, a_2

Invalid identifiers: $num1, for, 1st_paper_marks

5. What are operators? Explain the different types of operators supported in C. Explain ternary & bit-wise operator with suitable examples.

Ans:

List of operators used in C Language:

Ans:

i. Arithmetic Operators :
+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulus)

ii. Relational Operators:
< (less than)
<= (less than or equal to)
> (greater than)
>= (greater than or equal to)
==(equal to)
!=(not equal to)

iii. Logical Operators:
&& (and)
|| (or)
! (Not)

iv. Increment & Decrement:
++, --

v. Assignment Operator :
=, +=, -=, */, /=, %=

vi. Pre-processor Operator : #

vii. Member operator: . and ->

viii. Bitwise Operators : & (Bit-wise AND), | (Bit-wise OR), ! (Bit-wise NOT), ^ (Bit-wise XOR), >> (Bit-wise right shift), <<(Bit-wise left shift)

ix. Ternary Operator(Conditional if): ? :

x. Address of operator: &

xi. Pointer (dereference) operator: *

xii. Comma Operator: ,

xiii. Statement terminator operator: ;

Explaining Bitwise Operators:

& - Bit-wise AND : 1 only when both inputs are 1
printf("%d", 20 & 25); /*10100 & 11001 = 10000 will give output 16*/
| - Bit-wise or : 1 when any of the input is 1. It gives 0 when both inputs are 0
printf("%d", 20 | 25); /*10100 | 11001 = 11101 will give output 29*/
^ - Bit wise XOR (Exclusive or): 1 when only one input is 1. If both are 1 or 0, it gives 0
printf("%d", 20 ^ 25); /*10100 ^ 11001 = 01101 will give output 13*/
~ - Bit-wise compliment : Difference in highest & given. In Binary 1 will be 0 and 0 will be 1. If unsigned int has 2 bytes (16 bits) size with highest value 65535 then
printf("%u", ~ 5); /*~ 0000000000000101 = 1111111111111010 will give output 65530*/
>> - Bit wise right shift : Halves the value in integer
printf("%u", 10>>2); /*1010 will 101 in first right shift & again 10 in 2nd right shift so 2 */
<< - Bit wise left shift : doubles the value(within range)
printf("%u", 11<<2); /*1011 will 10110 in first left shift & again 101100 in 2nd left shift so 44 */

Ternary Operator(Conditional if): ? :
Syntax:
<condtion> ? <statement for true> : < statement for false>
int salary = 50000;
Example1:
bonus = salary > 40000 ? 10000 : 20000;
Here salary is greater than 40000 so, bonus will be 20000

6. Explain operator precedence and associativity with examples of arithmetic operators with relational operators.

Ans:

Operator Precedence:

- If more than one operators are involved in a single expression, C language has a predefined rule of priority for the operators. **This rule of priority of operators is called Operators precedence**.

  Example1:

  printf("%d", 3 % 10 + 5); /* gives the output 8 */

  Explanation: Modulas operator % has greater precedence than addition +. So, 3 % 10 will be 3 and adding 5 will give the result 8

  Example2:

  printf("%d", 3 + 4>5); /* gives the output 1 and not 3 */

  Explanation: Arithmetic operator has greater precedence than relational operator so, 3 will be added with 4 first then 7 will be compared with 5 which gives the boolean output 1.

  Example 3:

  printf("%d", 1 > 2 + 3 && 4);

  This expression is equivalent to:

  ((1 > (2 + 3)) && 4)

  i.e, (2 + 3) executes first resulting into 5

  then, first part of the expression (1 > 5) executes resulting into 0 (false)

  then, (0 && 4) executes resulting into 0 (false)

- In C, precedence of arithmetic operators(*, %, /, +, -) is higher than relational operators(==, !=, >, <, >=, <=) and precedence of relational operator is higher than logical operators(&&, || and !).

ii) Associativity:

If two operators of same precedence (priority) is present in an expression, Associativity of operators indicate the order in which they execute (left to right or right to left).

Most of the operators have the asociativity left to right.

Example 1:

printf("%d", 10/5/2); /* output 1 – There should not be confusion that 10 is divided by 5/2 */

Example 2:

```
printf("%d", 1==2 !=3); /* output 1,  1==2 will give 0 and 0 != 3 will give 1 */
```

increment and decrement operators have associativity from right to left:

Example:

```
int n=7;
printf("%d %d %d",n++,n++,n++);          /* gives output 9 8 7 and not 7 8 9 */
```

7. What are data types? Explain the different data types & their sizes supported by ANSI-C language.

Ans:

A data type is a classification of data, which can store a specific type of information.

Primitive data types are primary, fundamental or predefined types of data, which are supported by the programming language.

In C basic/fundamental/primitive data types with their sizes as per ANSI-C standard:

Type	Size
char	1 Byte
int	2 Bytes
float	4 bytes
double	8 Bytes

There are 4 data type specifiers/quantifiers also:

I.signed : Left most bit is reserved as signed bit (1 for negative, 0 for positive) By default all declarations are ***signed***.

II.unsigned: All bits are used(Where only positive values are expected)

III.short : Assures that size of data type will be <= normal size

IV.long : Assures that size of data type will be >= normal size

Programmers can use these data types when creating variables in their programs. For example, a programmer may create a variable called marks and define it as a int data type type.

Examples:

```
int marks;
double num = 0.00;
char c;
float sum;
```

Derived data types are non-primitive data types or composed data types from primary data types. Non-primitive data types are not defined by the programming language, but are instead created by the programmer. Arrays, pointers, structures are examples of derived data type.

Examples of derived/non-primitive data types:

```
int a[10]; /* array of 10 integers */
struct student
{
        int roll;
        char name[20];
        int marks;
};
struct student s1,s2; /* s1 and s2 are structure variables or objects */
int *p; /* p is pointer variable to keep address of integer data */
```

8. What is type conversion? Explain two types of type conversion in C with examples.

Ans:

Type conversion concept in C language is used to modify a variable from one date type to another data type. New data type should be mentioned before the variable name or value in brackets which to be typecast.

Example:

result = (float) 20/3;

- It is best practice to convert lower data type to higher data type to avoid data loss.
- Data will be truncated when higher data type is converted to lower. For example, if float is converted to int, data which is present after decimal point will be lost

Type conversion can be done 2ways:

a) Implicit Type conversion / Coercion / Automatic type conversion

It is done by compiler automatically:

example:

printf("%f", 20/3.00);

b) Explicit Type conversion / A cast

By preceding the expression with type in parenthesis

It may be checked, unchecked or bit pattern.

Syntax:

(type) expression;

9. Explain conditional branching. Explain if, if-else, nested if-else and cascaded if- else with syntax and examples.

Ans:

In C programming language branching statements are:

if, if else & switch

if/if else can be extended to:

ladder if & nested if

if statement: syntax

```
        if (expression/condition)
        {
                statement1;
                statement2;
                ...
        }
```

Example:

```
#include <stdio.h>
int main()
{
    int age;
    printf("Enter your age : ");
    scanf("%d", &age);
    if(age >= 25)
        printf("Celebrate valentine day\n");
    else
        printf("Just keep looking cute only.\n");
    return (0);
}
```

Nested if syntax:

```
if (expression/condition)
{
        if (expression/condition)
        {
                statement1;
                statement2;
                .....
        }
        else
        {
                statement1;
                statement2;
                .....
        }
 }
 else
 {
         if (expression/condition)
         {
                statement1;
                statement2;
                 .....
         }
          else
         {
                statement1;
                statement2;
                .....
         }
}
```

```
// Example: to find maximum in 3 numbers
#include <stdio.h>
int main()
{
    int n1,n2,n3,max;
    printf("Enter 3 numbers : ");
    scanf("%d%d%d", &n1, &n2, &n3);
```

```
    if(n1>n2)
        if(n1>n3)
            max=n1;
        else
            max=n3;
    else
        if(n2>n3)
            max=n2;
        else
            max=n3;

    print("Maximum = %d\n", max);
    return (0);
}
```

switch statement:

C has a built-in multiway decision statement known as a switch. It tests the value of a given variable (or expression) against a list of case values and when a match is found, a block of statements associated with the case is executed. **The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the next statement following the switch (where branch ends with }).** The default is an optional case. When present, it will be executed if the value of the expression does not match with any of the case values.

switch statement syntax:

```
switch (expression)
{       case  condition1
                statement1;
                statement2;
                ......
                break;
        case condition2
                statement1;
                statement2;
                .......
                break;
        .....
        default:
                statement1;
                statement2;
                .......
}
```

```
/* Example of switch */
#include <stdio.h>
int main()
{
    int salary,bonus;
    char grade:
    printf("Enter grade : ");
    scanf("%c", &grade);
    printf("Enter salary : ");
    scanf("%d", &salary);
    switch (grade)
    {
        case 'a':
        case 'A':   bonus=salary;
                break;
        case 'b':
        case 'B':   bonus=salary+5000;
```

```
                break;
    default :  bonus=salary+10000; /*lower grade-more bonus*/
    }
    print("Bonus = %d\n", bonus);
    return (0);
}
```

10. Write a C program

a. To find the roots of quadratic equation.

Ans:

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

int main()
{
   double a, b, c, d, r1, r2, real, imag;
   printf("\nEnter coefficients a, b and c: ");
   scanf("%lf %lf %lf",&a, &b, &c);
   d = b*b-4*a*c;  /* discriminant */

 if (d == 0)
   {
        r1 = r2 = -b/(2*a);
        printf("Roots are real : root1 = root2 = %.2lf;", r1);
   }

  else
    if (d > 0)
    {
         r1 = (-b+sqrt(d))/(2*a);
         r2 = (-b-sqrt(d))/(2*a);
         printf("Roots are distinct: root1 = .2lf root2 = %.2lf\n",r1, r2);
    }

 else
   {
         real = -b/(2*a);
         imag = sqrt(-d)/(2*a);
         printf("Roots are imaginary:\n");
         printf("root1 = %.2lf+%.2lfi and root2 = %.2lf-%.2lfi", real, imag, real, imag);
   }
   return (0);
}
```

Output:

```
Enter coefficients a, b and c:
Enter coefficients a, b and c: 2 4 2
Roots are real : root1 = root2 = -1.00;
```

b. To print Pascal's triangle

Ans:

```
/b. * Pascal's triangle */
#include <stdio.h>
int main()
{
  int x,y,z,p;
  for(x=0;x<9;x++)
  {
     for(y=1;y<20-x;y++)
         printf("  "); /* for blank spaces */
     for(y=0;y<x;y++)
     {
          if(x==0 || y== 0)
               p=1;
          else
               p=p*(x-y)/y;       /* For Bionomial co-efficient */

          printf("%4d",p);      /* printing value taking 4 minimum spaces */

     }
     printf("\n");
  }
  return (0);
}
```

Expected output: Plotting Pascal's triangle

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
    1   6  15  20  15   6   1
  1   7  21  35  35  21   7   1
```

11. What are looping structures? Explain with syntax, flowchart and example.

Ans:

There are 3 looping control statements in C:

- while
- do…while
- for

while statement syntax:

```
while (condition)
{
        statement1;
        statement2;
        ……
}
```

```
/*working example – to display squares of numbers from 1 to 20 */
#include <stdio.h>
int main()
{
    int x;
    x=1;
    while(x<=20)
    {
```

```
        printf("%d  %d\n", x, x*x);
        x++;
    }
    return (0);
}
```

do…while statement syntax: (It is a post tested loop & must executes at least once)

```
    do
    {
        statement1;
        statement2;
        ......
    } while (condition);
```

```
/*working example – to display squares of numbers from 1 to 20 */
#include <stdio.h>
int main()
{
    int x;
    x=1;
    do
    {
        printf("%d  %d\n", x, x*x);
        x++;
    }while(x<=20);
    return (0);
}
```

for statement syntax:

```
    for (initialization; test condition; increment or decrement)
    {
        statement1;
        statement2;
        .....
    }
```

```
/*working example – to display squares of numbers from 1 to 20 */
#include <stdio.h>
int main()
{
    int x;
        for(x=1;x<=20;x++)
    {
        printf("%d  %d\n", x, x*x);
    }
    return (0);
}
```

12. Differentiate between followings with examples.

a. while and do-while loop

Ans:

- ***while*** is a pre-tested (entry-controlled) loop. Condition is checked before entering in the loop.
- ***do-while*** is a post-tested (exit-controlled) loop. Condition is checked after execution of the statements within the loop.

- **The main difference is that do-while loop must execute once (because condition is checked at the end.)**

while statement syntax:

```
    while (condition)
```

```
{
        statement1;
        statement2;
        ……
}
```

```
/*working example – to display squares of numbers from 1 to 20 */
#include <stdio.h>
int main()
{
        int x;
        x=1;
        while(x<=20)
        {
                printf("%d  %d\n", x, x*x);
                x++;
        }
        return (0);
}
```

do…while statement syntax: (It is a post tested loop & must executes at least once)

```
do
{
        statement1;
        statement2;
        ……
} while (condition);
```

```
 /*working example – to display squares of numbers from 1 to 20 */
#include <stdio.h>
int main()
{
        int x;
        x=1;
        do
        {
                printf("%d  %d\n", x, x*x);
                x++;
        }while(x<=20);
        return (0);
}
```

b. break and continue statements

Ans:

break statement would only exit from the loop containing it.

```
//Example program to check a number whether it is prime
#include<stdio.h>
int main()
{

int n,d,prime=1;
printf("Enter a number : ");
scanf("%d", &n);
for (d=2;d<n/2;d++)
{
      if (n%d==0)
      prime=0;
      break; /* there is no need to check further */
}
if (prime)
      printf("Yes %d is a prime number.\n", n);
else
```

```
        printf("No, %d is not a prime number.\n", n);

return (0);
}
```

The continue statement is used in loops to skip the following statements in the loop and to continue with the next iteration (current iteration in for loop).

```
//Example program to enter marks (0-100) in 6 subjects for sum
#include<stdio.h>
int main()
{
int marks,sum=0, x;
for (x=1;x<=6;x++)
{
      printf("Enter marks %d : ", x)
      scanf("%d", &marks);
      if(marks <0 || marks > 100)
      {
            printf("Invalid marks!!!\n");
            continue; /* again read marks for same paper */
      }
      sum+=marks;
}
printf("sum of marks = %d.\n", sum);
return (0);
}
```

13. What is an array? Explain how a single dimensional and two dimensional arrays is declared and initialized?

Ans:

An array is an identifier that refers to the collection of data items which all have the same name. The individual data items are represented by their corresponding array elements. Address of an element (subscript) ranges from 0 to n-1, where n is total number of elements.

i) Declaration of one-dimensional array:

One dimensional array:

syntax:

type variable[size];

Examples:

```
float height[50];
int batch[10];
char name[20];
```

ii) **Initialization of one-dimensional array:**

Examples:

```
int marks[5] = {76,45,67,87,92};
static int count[ ] = {1,2,3,4,5};
static char name[5] = {'S', 'I', 'N', 'G', 'H', '\0'};
```

iii) Declaration of Two-dimensional array:

syntax:

type *arrayname*[rowsize][columnsize];

Examples:

int stud[16][5];

iv) Initialization of Two-dimensional array:

Examples:

```
int marks[3][3] = {
                        {26,57,66},
                        {56,77,48},
```

```
                {76,54,82}
        };
```

```
int marks[3][3] = { {12},{0},{0}};
```

Here first element of each row is explicitly initialised to 12 while other elements are automatically initialized to zero.

14. Write a program to demonstrate working of selection sort & bubble Sort.

Ans:

Selection Sort:

```
/* Code Selection Sort */
#include <stdio.h>
void selesort(int a[], int size)
{
  int x,y,temp;
  for(x=0;x<size-1;x++)
   for(y=x+1;y<size;y++)
     if(a[y] > a[y+1])
      {                    /* swapping adjacent element */
       temp=a[y];
       a[y]=a[y+1];
       a[y+1]=temp;
      }
}

main()
{
 int a[1000], tot, x;
 printf("Total elements : ");
 scanf("%d",&tot);
 for(x=0;x<tot;x++)
 {
     printf("Enter element %d :", x+1);
     scanf("%d",&a[x]);
 }
 selesort(a,tot);
 printf("Sorted array:\n");
 for(x=0;x<tot;x++)
     printf("%5d",a[x]);

return (0);
}
```

Bubble Sort

```
/* Code Bubble Sort */
#include <stdio.h>
void bubsort(int a[], int size)
{
  int x,y,temp;
  for(x=0;x<size-1;x++)
   for(y=0;y<size-x-1;y++)
     if(a[y] > a[y+1])
      {                    /* swapping adjacent element */
       temp=a[y];
       a[y]=a[y+1];
       a[y+1]=temp;
      }
}
```

```
main()
{
 int a[1000], tot, x;
 printf("Total elements : ");
 scanf("%d",&tot);
 for(x=0;x<tot;x++)
 {
    printf("Enter element %d :", x+1);
    scanf("%d",&a[x]);
 }
 selesort(a,tot);
 printf("Sorted array:\n");
 for(x=0;x<tot;x++)
    printf("%5d",a[x]);

return (0);
}
```

15. Write a C program to search a key integer element in the given array of N elements using binary search technique. Assure that numbers are sorted before search operation.

Ans:

```
/* Binary search in given array of N elements */
#include <stdio.h>

int main()
{
  int a[1000],x, y,temp,N, first, last, mid, skey, found;
  printf("Enter number of elements : ");
  scanf("%d", &N);
  for(x=0;x<N;x++)
  {
     printf("Enter element %d : ", x+1);
     scanf("%d",&a[x]);
  }
  /* now sorting - because Binary search requires sorted elements */
  for(x=0;x<N;x++)
     for(y=0;y<N-x-1;y++)
       if(a[y] > a[y+1])
       {
          temp=a[y];
          a[y]=a[y+1];
          a[y+1]=temp;
       }
  printf("The sorted array:\n");
  for(x=0;x<N;x++)
     printf(" %d",a[x]);

  printf("\nEnter search key integer : ");
  scanf("%d",&skey);
  first=0;
  last=N-1;
  found=0;

  while(first <= last && ! found)
  {
     mid=(first+last)/2; /* integer value of mid */
     if(a[mid] > skey)
    last=mid-1;
     else if (a[mid] < skey)
    first=mid+1;
     else
    found=1;
  }
  if(found)
```

```
      printf("Found at %dth position\n",mid+1);
   else
      printf("Not found\n");

   return (0);
}
```

Expected Ouput:

```
Enter number of elements : 11
Enter element 1 : 22
Enter element 2 : 3
Enter element 3 : 44
Enter element 4 : 5
Enter element 5 : 66
Enter element 6 : 7
Enter element 7 : 88
Enter element 8 : 9
Enter element 9 : 111
Enter element 10 : 23
Enter element 11 : 54
The sorted array:
 3 5 7 9 22 23 44 54 66 88 111
Enter search key integer : 88
Found at 10th position
```

16. How are strings declared and initialized? Explain any 5 string manipulation functions with examples.

Ans:

A string is terminated with NULL '\0' character. When a string is entered by default one NULL character is added at the end.

String declaration and initialization:

Declaration:

char str[size];

example:

char str[50];

Initialization

char name[ ]='SINGH';

char name[5]={'S','I','N','G',H','\0'};

String manipulating Functions: (Header file: string.h)

String manipulating Function	Work of Function
strlen()	Calculates the length of string
strcpy()	Copies a string to another string
strcat()	Concatenates(joins) two strings
strcmp()	Compares two string
strrev()	Reverses a string
strlwr()	Converts string to lowercase
strupr()	Converts string to uppercase

String manipulating Function	Work of Function
strchr()	Scans a string to search a character

Now explaining strcat() and strcmp():

strcat(str1, str2) – Concatenates(appends) str2 to str1 at the end.

Syntax:

char* strcat (char* strg1, const char* strg2);

Example:

If two strings are:

char name[]="Roma";

char surname[]="Ritambhara";

Then to append "Ritambhara" with name "Roma"

strcat(name,surname);

printf("%s", name);

Output: Roma Ritambhara

strcmp(str1, str2) – Compares two string and returns:

0 if both strings are identical

else returns difference based on ASCII value of first mismatch.

Syntax:

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

Example:

printf("%d", strcmp("SINGH", "SINHA");

output: - 1

17. What is user defined function? Give the advantages of using functions. Explain two categories of argument passing techniques with examples.

Ans:

- A function is a self-contained block of statements that performs a coherent task of some kind.
- Mathematical definition of a function is like that "A function f(x) returns value for the argument x."
- Functions are classified of two types – In-built function and User/Programmer defined functions.
- A function name is followed by a pair of ( ).
- A function may or may not have arguments. Arguments are written within parentheses separating by , (comma).
- **An user defined function is defined by user as per task is to be performed by function. Now a days user defined functions are also called method as a function is a method to perform a task.**

Yes, main( ) is a user defined function which is specially recognised function in C. Every program must have a function main( ) to indicate where the program has to begin its execution.

Function prototyping : Formal declaration of a function with data type like variables, arrays prior to its use is known as function prototyping. Without it most of the compilers displays an error message "abcd function should have a prototype." Called function should receive the same data type as declared.

Function based on argument:

A function may or may not send back any value to the calling function. The data type void says that the function will not return any value and suppresses the warning message that "Function should return a value." If any function returns any value, it is done through the return statement.

return; or return (expression);

Function without parameter:

```
void gao()
{
    printf("Kudi Punjaban dil chura ke lay gai – Sona Sona, Dil Mera Sona);
}
int main()
{
       gao();  /* calling function */
       return (0);
}
```

A function may have arguments. Formal declaration of the data type of arguments to be passed with function should be done also.

Arguments are passed as i. Pass/call by value ii. Pass/call by reference

```
/*i.  Examples of call by value – Function is called by passing value. It will not effect the value of calling function */
int area( int l, int w) { return (l*w);
int main()
{
    int len, wid;
   printf("Enter length and width of rectangle : ");
  scanf("%d%d",&len,&wid);
  printf("Area of rectangle = %d square unit\n", area(len,wid));
 return (0);
}
```

In the called function arguments are received as parameters declaring their type:

```
            int mul( int a, int b)
            {
                    ......
            }
or          int mul(a,b)
            int a,b;   /* T & R classical style */
            {  ......  }
```

A function may be nested also:

```
int max2(int a, int b) { return (a>b?a:b); }
int main( )
{
     int x,y,z;
     printf("Enter 3 numbers : ");
     scanf("%d%d%d",&x,&y,&z);
     printf("Maximum = %d\n". max2(max2(x,y),z));  /* Nesting of function if more arguments */
     return (0);
}
```

/*ii. Pass/Call by reference

When a function is called by passing address then called function may change value of the variable because it receives address. Thus value of the variable of calling function will change also */

```
int swap(int *a, int *b) { int temp=*a; *a=*b; *b=temp; }
int main()
{
   int a=20, b=30;
   printf(a=%d b=%d\n",a,b); /* a=20 b=30 */
   swap(&a,&b); /* pass/call by reference */
   printf(a=%d b=%d\n",a,b); /*a=30 b=20 */
   return (0);
}
```

18. What is call by value and call by reference? Explain with example.

Ans:

A function may have arguments. Formal declaration of the data type of arguments to be passed with function should be done also.

Arguments are passed as i. Pass/call by value ii. Pass/call by reference

```
/*i.  Examples of call by value – Function is called by passing value. It will not effect the value of calling function */
int area( int l, int w) { return (l*w);
int main()
{
    int len, wid;
   printf("Enter length and width of rectangle : ");
  scanf("%d%d",&len,&wid);
  printf("Area of rectangle = %d square unit\n", area(len,wid));
 return (0);
}
```

In the called function arguments are received as parameters declaring their type:

```
                int mul( int a, int b)
                {
                        ......
                }
or              int mul(a,b)
                int a,b;   /* T & R classical style */
                {  ......  }
```

A function may be nested also:

```
int max2(int a, int b) { return (a>b?a:b); }
int main( )
{
     int x,y,z;
     printf("Enter 3 numbers : ");
     scanf("%d%d%d",&x,&y,&z);
     printf("Maximum = %d\n". max2(max2(x,y),z));  /* Nesting of function if more arguments */
     return (0);
}
```

/*ii. Pass/Call by reference

When a function is called by passing address then called function may change value of the variable because it receives address. Thus value of the variable of calling function will change also */

```
int swap(int *a, int *b) { int temp=*a; *a=*b; *b=temp; }
int main()
{
   int a=20, b=30;
   printf(a=%d b=%d\n",a,b); /* a=20 b=30 */
   swap(&a,&b); /* pass/call by reference */
   printf(a=%d b=%d\n",a,b); /*a=30 b=20 */
   return (0);
}
```

19. What is recursion? Write a function to check if a given number is prime or not, using recursion and without using recursion.

Ans:

Using Recursion:

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

```
#include <math.h>
int checkprime(int n,int d)
{

  if(d==1) return (1);
  else
  if(n%d==0 && n>2)
     return (0);
  else
  if(d<=sqrt(n))
       return checkprime(n,++d);

}


int main()
{
  int n;
  printf("Enter number : ");
  scanf("%d",&n);
  if(checkprime(n,2))
       printf("Yes %d is prime",n);
  else
       printf("No, %d is not prime\n",n);

  printf("\n");
  return (0);
}
```

Without recursion (Non-Recursive)

```
/* to check a number whether it is prime or not */
#include <stdio.h>
#include <math.h>
int main()
{
   int n,d,prime=1;
   printf("Enter number : ");
   scanf("%d",&n);
   for(d=2;d<=sqrt(n);d++)
     if(n%d==0)
     {
   prime=0;
   break;
     }
  if(prime)
        printf("Yes, %d is a prime number\n",n);
  else
        printf("No, %d is not prime number\n",n);
  return (0);
}
```

20. Using recursive functions, write a program for the following

a. Factorial of a given number

Ans:

```
/* Program to find factorial of using recursion */
     #include <stdio.h>
     long int fact(int m)
     {
     if (m == 1 || m == 0)
                return(1);
     else
                return(m * fact(m-1));      /* recursion */
```

```
}

int main()
{
      int n;
            printf("Enter any number for factorial : ");
            scanf("%d",&n);
            printf("Factorial of % d = %ld\n",n,fact(n));
      return (0);
}
```

O/P:

Enter any number for factorial : 6
Factorial of 6 = 720

b. Fibonacci series

Ans:

```
/* Fibonacci series up to nth terms using recirsion */
#include <stdio.h>

int fibo(int n)
{
  if (n==1)
    return (0);
  if (n==2)
    return (1);
  else
    return (fibo(n-1)+fibo(n-2));
}

int main()
{
  int n,x;
  printf("Enter nth term for Fibonacci series : ");
  scanf("%d",&n);
  for(x=1;x<=n;x++)
     printf(" %d",fibo(x));

  return (0);
}
```

Output:

Enter nth term for Fibonacci series : 10
0 1 1 2 3 5 8 13 21 34
Enter nth term for Fibonacci series : 20
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181

21. What is structure? How is it declared, defined and used? Write a C program to store and print Name, USN, Subject and IA Marks of 60 students.

Ans:

Structure is a data structure whose individual elements can differ in type. It may contain integer elements, character elements, pointers, arrays and even other structures can also be included as elements within a structure. struct is keyword to define a structure.

```
struct  tag  { type member1;
                type member2;
                type member3; ..... type member n;};
```

New structure type variables can be declared as follows:

```
struct tag var1, var2, var3, ..... varn, var[10];
```

i) Array of structure:

Whole structure can be an element of the array. A student structure with members roll, name and

marks:

```
struct student
{
    int roll;
    char name[30];
    int marks;
};
```

Now we can use this template to create array of 60 students for their individual roll, name and marks.

struct student s[60]; /* array of structure *

to access roll of student x : **s[x].roll**

ii) Array within structure:

A member of structure may be array. In above example name is also array of characters. We can create a structure of student with members roll, name & marks (array of integers) for 6 papers:

```
struct student
{
    int roll;
    char name[30];
    int marks[6]; /* array within structure */
}s[60];
```

To access the marks of student x in paper y: **s[x].marks[y]**

ii) Structure within Structure:

A structure may be defined within another structure or a structure variable may be declared as a member within another structure . Here date structure is defined within employee structure:

```
struct employee
{
    int eno;
    char name[30];
    long int salary;
    struct date   /*structure within structure */
    {
        int dd, mm, yy;
    }dob,doj;
}e[1000];
```

Or structure variable of date structure can be defined within employee if date structure is separately defined:

```
    struct date
    {
        int dd, mm, yy;
    };
struct employee
{
    int eno;
    char name[30];
    long int salary;
    struct date dob, doj; /* variable of date structure within employee*/
}e[1000];
```

To access day(dd) of date structure we will use member operator (.) dot operator in following way:

e[x].dob.dd

22. Write a program to input & sort array of structure of 60 students with fields roll, name and marks.

Ans:

```
/*sorting array of structure - based on marks in descending order*/
#include <stdio.h>
struct student
{
  int roll;
  char name[20];
  int marks;
}s[1000]; /* array of structure */


int main()
{
  int n,x,y;
  struct student temp;
  printf("How many students : ");
  scanf("%d",&n);
  for(x=0;x<n;x++)
  {
   printf("Enter roll, name and marks for student %d : ", x+1);
   scanf("%d%s%d",&s[x].roll,s[x].name,&s[x].marks);
  }
  /*now sorting */
  for(x=0;x<n-1;x++)
   for(y=0;y<n-x-1;y++)
    if(s[y].marks < s[y+1].marks)
    {
        temp=s[y];
        s[y]=s[y+1];
        s[y+1]=temp;
    }
  printf("roll   name    marks:\n");
  for(x=0;x<n;x++)
    printf("%5d%20s%5d\n",s[x].roll,s[x].name,s[x].marks);

  return (0);
}
```

23. Define pointer variables. Explain with an example declaration and initialization of pointer variable.

Ans:

A pointer keeps address of data. A pointer is variable that represents the location (rather than the value) of a data item, such as a variable or an array element.

Pointers is an important feature of C and frequently used in C. They have a number of useful applications.

Pointer operator is asterisk (*).

To declare a pointer: *datatype *pointervariable;*

To initialize: pointervariable = &data;

& is address of operator.

Example1:

```
int n;
int *p;  /* here p is pointer variable & can be used to keep address */
p=&n;  /* here p keeps address of n */
```

Example2:

```
int a[5] = { 10,20.30,40,50};
int *p;
p = a;   /* It is equivalent to  p = &a[0];   Here it keeps address of first element */
```

24. Distinguish between the following types of variables/storage class with examples:
    a. auto (automatic)
    b. extern (global)
    c. static
    d. register

Ans:

The storage class specifies the portion of the program within which the variables are recognized. variables can be categorized by storage class as well as data type. Storage class refers to the permanence of a variable and its scope within the program, that is, the portion of the program over which the variable is recognized.

i. auto (Automatic variables): They are local variables, available to the function in which it is declared. Their scope is confined to that function only. Value of the variable defined within a function is no way available to other function. Variables declared within a function are by default of auto storage class.

```
int man()
{
    auto int x,y,z;
            int num;        /* num is also interpreted as auto storage class
            /* ….. other code */
    }
```

ii. extern (External/Global variables): They are global variables, known to all the functions of the program from point of definition. Hence, they usually span more than one functions, and often an entire program. Any of the function can change/alter/corrupt extern variables. Variables defined outside of the program are by default of extern storage class.

```
  extern int global = 200; /* by default extern */
void func()
{
        printf("global = %d\n",global); /* displays global = 300 */
        global=400;  /* global variable changed by func() */
}
    int main( )
        {
            printf("global = %d\n",global);
        global=300; /* value of global changed */
        func(); /* calling function */
        printf("global = %d\n",global); /* displays global = 400 */
        return (0);
}
```

iii. static (Static variable): Static variables have the same scope as automatic variables, i.e. they are local to the functions in which they are defined. Unlike automatic variables, however, static variables retain their values throughout the life of the program. As a result, if a function is called again, the static variables defined in the called function will retain their former values **(They will not be reinitialized again. They are confined at the first initialization)**.

```
#include <stdio.h>
int func()
{
  static int k = 0; /* by default also zero */
  k++;
  return (k);
}
```

```
int main()
{
    int x, sum=0;
   for(x=1;x<=5;x++)
     sum+=func();
  print("sum = %d\n", sum);
  return (0);
}
```

iv. register (Register variables) : Registers are special storage area within a computer's central processing unit (In-built memory with CPU). The actual arithmetic and logical operations that comprise a program are carried out within this registers. The execution time can be reduced considerably if certain values can be stored within these registers rather than in computer's memory.

The register variables are local to the function in which they are declared. The address of operator cannot '&' cannot be applied to register variables.

register int x, y, z;

25. What is dynamic memory allocation? Explain different types of dynamic memory allocation and de-allocation built-in functions of C.

Ans:

Memory is allocated/de-allocated dynamically at run time as and when required. Dynamic memory management refers to manual memory management. This allows a programmer to obtain more memory when required and release it when not necessary.

In C there are 4 library functions defined under stdlib.h or alloc.h for the same

malloc( ) - Allocates a block of memory of specified size in bytes and return a pointer of type void which can be casted into pointer of any form

Syntax:

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

Example for p is 2-d array:

p= malloc(sizeof(int)*rows*cols)

If p is cast type then one example

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

calloc() – Contiguous allocation – Allocates multiple blocks of same size – requires two arguments

calloc(number_of_blocks, size_in_bytes_for_each_block)

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

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

realloc( ) : changes/reallocates the size of previously allocated space by malloc() or calloc()

Syntax:

ptr = realloc(ptr, newsize);

if size1 is allocated by malloc: ptr = (int*) malloc(size1 * sizeof(int));

then to rallocate for size2

ptr = realloc(ptr, size2);

free() – To deallocate the previously allocated space

Syntax/example:

free(ptr)

This statement frees the space allocated in the memory pointed by ptr.

26. What is a pre-processor directive? Explain any 5 pre-processor directives in C.

Ans:

A pre-processor represents pre-compilation process. These directives are preceded with # (**and there is no semicolon at the end**).

C has the special feature of pre-processor directives. The pre-processor processes the C source code before it passes to the compiler. Pre-processor directives are executed before the C source code passes through the compiler.

Pre-processor directives can be categorized in 2 groups:

- **General Directives : These are written in header section of the program and there will not be control of compiler over it. They are processed before compilation of program starts.**

Examples: #include, #define, #undef, #pragma

- **Compiler control directives: These are conditional directives and may be written within program also. For example whether a macro is defined or not we can check it within program and redefine (general directives also) if required. Mostly #if family directives are of this category.**

Examples: #if, #endif, #else, #ifdef, #ifndef

Examples of pre-processor directives:

i) #include

This pre-processor is used to include another file. File name is written in angle brackets or within double quotes.

#include <stdio.h>

#include "SORT.H"

File name enclosed within angle brackets are searched in standard directories. File name enclosed within double quotes are searched first in the current directory and if not found then it is searched in the standard directories.

ii) #define

#define directive is used to define the symbolic constant or macro name with macro expression.

#define symconstant value

#define macroname macroexpression

Example

#define PI 3.141

Here the C pre-processor searches for the symbolic constant PI the source code and substitutes 3.141 each time it is encountered.

As the function, a macro can have argument. The argument of the macro name is replaced with actual argument of the macro name, which is encountered in the program.

Examples:

#define MAX(x,y) ((x) > (y) ? (x) : (y))

#define CUBE(x) (x)*(x)*(x)

iii) #ifdef

#ifdef preprocessor directives checks whether a macro is defined by #define. If yes, it executes

the code:

Example:

#ifdef PI

```
        Printf("PI symbolic constant is already defined\n");
#endif
```

iV) #ifndef

#ifndef preprocessor directives checks whether a macro is not defined by #define. If yes, it executes the code:

Example:

```
#ifndef PI
        #define PI 3.14
#endif
```

V) #pragma

#pragma preprocessor directives schedules the execution of function before and after main() function by clauses startup and exit:

Example:

```
#pragma startup func2
#pragma exit func1
```

Here func2() function will execute before main() function and func1() function will execute after main() function. There is no need to call these function from main()

Example program for macros:

```
#include <stdio.h>
#define cube(x) x*x*x
int main()
{
   printf("%d", cube(5+2))
   return (0);
}
```