Introduction to Programming

Sample Question Paper

Note: Q.No. 1 is compulsory, attempt one question from each unit.

Q. 1. a). What is an algorithm? 2.5 X10

b) What do you mean by Program? Write a small code for any program.

c). What is the difference between while and do while loop.

d). What is the use of getchar and putchar.

e) Explain break and continue statements.

f) What is an array?

g) What is the difference between structure and union.

h) What is a pointer?

i) Explain difference between variables and constants.

j) What is a static variable?

Unit-1

Q. 2. Design an algorithm as well as flowchart for finding out largest number out of three given numbers. 12.5

Q 3. Explain different types of programming languages. 12.5

Unit-2

Q4. What is a data type? Explain different data types in detail with examples. 12.5

Q 5. a) Write a program to check whether a given number is even or odd. 6.5

b) Explain selection statements in C. 6

Unit-3

Q6. What is a loop? Explain in detail with example. 12.5

Q 7. Write a program to multiply two matrices. 12.5

Unit-4

Q. 8. Write a program to swap two values using call by value and call by reference. 12.5

Q.9. Write short notes on:

a) Passing arrays as arguments 6.5

b) File Handling 6

Answers

Ans. 1. a) An algorithm (pronounced AL-go-rith-um) is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850. Al-Khwarizmi's work is the likely source for the word *algebra* as well.A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

b) A computer program can be viewed as an elaborate algorithm. In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

c) The **do while** loop executes the content of the loop once before checking the condition of the while.
Whereas a **while** loop will check the condition first before executing the content.
In this case you are waiting for user input with scanf(), which will never execute in the **while** loop as wdlen is not initialized and may just contain a garbage value which may be greater than 2.

d) getchar() is used to get a single character from the user.....

getchar()- single character can be given to the computer
using 'c' input library function.....

putchar() is used to output a single character and echo's in the output screen

putchar()- used to display one character at a time on the
standard output device.....

Example:1
The program reads five characters (one for each iteration of the for loop) from the keyboard. Note that getchar() gets a single character from the keyboard, and putchar() writes a single character (in this case, ch) to the console screen.

```
#include <stdio.h>

main()
{
int i;
int ch;

for( i = 1; i<= 5; ++i ) {
ch = getchar();
putchar(ch);
}
}
```

Sample Program Output
AACCddEEtt

e) There are two statements built in C programming, break; and continue; to alter the normal flow of a program. Loops perform a set of repetitive task until text expression becomes false but it is sometimes desirable to skip some statement/s inside loop or terminate the loop immediately without checking the test expression. In such cases, break and continue statements are used. The break; statement is also used in switch statement to exit switch statement.

break Statement

In C programming, break is used in terminating the loop immediately after it is encountered. The break statement is used with conditional if statement.

continue Statement

It is sometimes desirable to skip some statements inside the loop. In such cases, continue statements are used.

Syntax of continue Statement

```
continue;
```

f) In programming, a series of objects all of which are the same size and type. Each object in an array is called an *array element.* For example, you could have an array of integers or an array ofcharacters or an array of anything that has a defined data type. The important characteristics of an array are:
Each element has the same data type (although they may have different values).
The entire array is stored contiguously in memory (that is, there are no gaps between elements).
Arrays can have more than one dimension. A one-dimensional array is called a *vector ;* a two-dimensional array is called a *matrix*.

g) union is a way of providing an alternate way of describing the same memory area. In this way, you could have a struct that contains a union, so that the "static", or similar portion of the data is described first, and the portion that changes is described by the union. The idea of a union could be handled in a different way by having 2 different structs defined, and making a pointer to each kind of struct. The pointer to struct "a" could be assigned to the value of a buffer, and the pointer to struct "b" could be assigned to the same buffer, but now a->somefield and b->someotherfield are both located in the same buffer. That is the idea behind a union. It gives different ways to break down the same buffer area.
The difference between structure and union in c are: 1. union allocates the memory equal to the maximum memory required by the member of the union but structure allocates the memory equal to the total memory required by the members. 2. In union, one block is used by all the member of the union but in case of structure, each member have their own memory space
Difference in their Usage:
While structure enables us treat a number of different variables stored at different in memory , a union enables us to treat the same space in memory as a number of different variables. That is a Union offers a way for a section of memory to be treated as a variable of one type on one occasion and as a different variable of a different type

on another occasion.

There is frequent rwquirement while interacting with hardware to access access a byte or group of bytes simultaneously and sometimes each byte individually. Usually union is the answer.

=======Difference With example***** Lets say a structure containing an int,char and float is created and a union containing int char float are declared. struct TT{ int a; float b; char c; } Union UU{ int a; float b; char c; }

sizeof TT(struct) would be >9 bytes (compiler dependent-if int,float, char are taken as 4,4,1)

sizeof UU(Union) would be 4 bytes as supposed from above.If a variable in double exists in union then the size of union and struct would be 8 bytes and cumulative size of all variables in struct.

h) A pointer is a variable that stores a memory address. Pointers are used to store the addresses of other variables or memory items. Pointers are very useful for another type of parameter passing, usually referred to as **Pass By Address**. Pointers are essential for dynamic memory allocation.

Declaring pointers:

Pointer declarations use the * operator. They follow this format:

*typeName * variableName;*

```
  int n;       // declaration of a variable n
int * p;     // declaration of a pointer, called p
```

In the example above, p is a pointer, and its type will be specifically be referred to as "pointer to int", because it stores the address of an integer variable. We also can say its type is: int*

The type is important. While pointers are all the same size, as they just store a memory address, we have to know **what** kind of thing they are pointing TO.

```
  double * dptr;        // a pointer to a double
   char * c1;           // a pointer to a character
float * fptr;           // a pointer to a float
```

Note: Sometimes the notation is confusing, because different textbooks place the * differently. The three following declarations are equivalent:

```
 int *p;
 int* p;
 int * p;
```

All three of these declare the variable p as a pointer to an int.

i)variable is a quantity that can change while constant is a quantity that cannot change during program execution.
variable is a name given to the memory space in which a a constant is stored

eg. int x =10;
here, x is a variable because later in the program you can write
x= 15;

whereas, 10 is constant because you can never write later in a program that
10=20;

Variables value can be changed by you in the program
example
int n=5;
n=4;
whereas constants value cannot be changed
constants are created by the use of keyword final
example
final int n=5;
n=4;//will give error

j) In computer programming, a **static variable** is a variable that has been allocated statically—whose lifetime or "extent" extends across the entire run of the program. This is in contrast to the more ephemeral automatic variables (local variables are generally automatic), whose storage is allocated and deallocated on the call stack; and in contrast to objects whose storage is dynamically allocated in heap memory.

When a program (executable or library) is loaded into memory, static variables are stored in the data segment of the program's address space (if initialized), or theBSS segment (if uninitialized), and are stored in corresponding sections of object files prior to loading.

The static keyword is used in C and related languages both for static variables and other concepts

Ans. 2. **Algorithms**

1. A sequential solution of any program that written in human language,called algorithm.
2. Algorithm is first step of the solution process, after the analysis of problem, programmer write the algorithm of that problem.
3. Example of Algorithms:

Write a algorithem to find out number is odd or even?

Ans.

```
step 1 : start
step 2 : input number
step 3 : rem=number mod 2
step 4 : if rem=0 then
            print "number even"
         else
            print "number odd"
         endif
step 5 : stop
```

Flowchart

1. Graphical representation of any program is called flowchart.
2. There are some standard graphics that are used in flowchart as following:

Figure: Start/Stop terminal box

Figure: Input/Output box

Figure: Process/Instruction box

Figure: Lines or Arrows

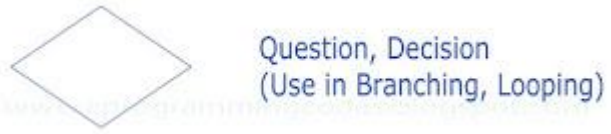


Figure: Decision box

Figure: Connector box

Figure: Comment box

Figure: Preparation box

Ans. 3. Different languages have different purposes, so it makes sense to talk about different kinds, or types, of languages. Some types are:

- Machine languages — interpreted directly in hardware
- Assembly languages — thin wrappers over a corresponding machine language
- High-level languages — anything machine-independent
- System languages — designed for writing low-level tasks, like memory and process management

Machine Code

Most computers work by executing stored programs in a fetch-execute cycle. Machine code generally features

- Registers to store values and intermediate results
- Very low-level machine instructions (add, sub, div, sqrt)
- Labels and conditional jumps to express control flow
- A lack of memory management support — programmers do that themselves

Machine code is usually written in hex. Example for the Intel 64 architecture:

```
89 F8 A9 01 00 00 00 75 06 6B C0

03 FF C0 C3 C1 E0 02 83 E8 03 C3
```

Assembly Language

An assembly language is basically just a simplistic encoding of machine code into something more readable. It does add labeled storage locations and jump targets and subroutine starting addresses, but not much more. Here's the function on the Intel 64 architecture using the GAS assembly language:

```
        .globl  f
        .text
f:
        mov     %edi, %eax      # Put first parameter into eax register
        test    $1, %eax        # Isloate least significant bit
        jnz     odd             # If it's not a zero, jump to odd
        imul    $3, %eax        # It's even, so multiply it by 3
        inc     %eax            # and add 4
```

```
        ret                 # and return it
even:
        shl    $2, %eax         # It's odd, so multiply by 4
        sub    $3, %eax         # and subtract 3
        ret                 # and return it
```

For the SPARC:

```
        .global f
f:
        andcc   %o0, 1, %g0
        bne     .L1
        sll     %o0, 2, %g2
        sll     %o0, 1, %g2
        add     %g2, %o0, %g2
        b       .L2
        add     %g2, 1, %o0
.L1:
        add     %g2, -3, %o0
.L2:
        retl
        nop
```

High-Level Languages

A high-level language gets away from all the constraints of a particular machine. HLLs have features such as:

- Names for almost everything: variables, types, subroutines, constants, modules
- Complex expressions (e.g. 2 * (y^5) >= 88 && sqrt(4.8) / 2 % 3 == 9)
- Control structures (conditionals, switches, loops)
- Composite types (arrays, structs)
- Type declarations
- Type checking
- Easy ways to manage global, local and heap storage
- Subroutines with their own private scope
- Abstract data types, modules, packages, classes
- Exceptions

The previous example looks like this in Fortran 77 (note how the code begins in column 7 or beyond):

```
      INTEGER FUNCTION F(N)
      INTEGER N
      IF (MOD(N, 2) .EQ. 0) THEN
         F = 3 * N + 1
      ELSE
         F = 4 * N - 3
      END IF
      RETURN
      END
```

and like this in Ada:

```
function F (N: Integer) return Integer is
begin
   if N mod 2 = 0 then
      return 3 * N + 1;
   else
      return 4 * N - 3;
   end if;
end F;
```

and like this in Fortran 90 (where the column requirements were finally removed):

```
integer function f (n)
   implicit none
   integer, intent(in) :: n
   if (mod(n, 2) == 0) then
      f = 3 * n + 1
   else
      f = 4 * n - 3
   end if
end function f
```

and like this in C and C++:

```
int f(const int n) {
   return (n % 2 == 0) ? 3 * n + 1 : 4 * n - 3;
}
```

and like this in Java and C#:

```
class ThingThatHoldsTheFunctionUsedInTheExampleOnThisPage {
   public static int f(int n) {
      return (n % 2 == 0) ? 3 * n + 1 : 4 * n - 3;
   }
}
```

and like this in Scala:

```
def f(n: Int) = if (n % 2 == 0) 3 * n + 1 else 4 * n - 3;
```

and like this in JavaScript:

```
function f(n) {
   return (n % 2 === 0) ? 3 * n + 1 : 4 * n - 3;
}
```

and like this in CoffeeScript:

```
f = (n) -> if n % 2 == 0 then 3 * n - 1 else 4 * n + 3
```

and like this in Smalltalk:

```
f
  ^self % 2 = 0 ifTrue:[3 * self + 1] ifFalse:[4 * self - 3]
```

and like this in ML:

```
fun f n = if n mod 2 = 0 then 3 * n + 1 else 4 * n - 3
```

and like this in Lisp and Scheme:

```
(defun f (n)
  (if (= (mod n 2) 0)
    (+ (* 3 n) 1)
    (- (* 4 n) 3)))
```

and like this in Clojure:

```
(defn f [n]
  (if (= (mod n 2) 0)
    (+ (* 3 n) 1)
    (- (* 4 n) 3)))
```

and like this in Prolog:

```
f(N, X) :- 0 is mod(N, 2), X is 3 * N + 1.
f(N, X) :- 1 is mod(N, 2), X is 4 * N - 3.
```

and like this in Perl:

```
sub f {
    my $n = shift;
    $n % 2 == 0 ? 3 * $n + 1 : 4 * $n - 3;
}
```

and like this in Python:

```
def f(n):
    return 3 * n + 1 if n % 2 == 0 else 4 * n - 3
```

and like this in Ruby:

```
def f(n)
  n % 2 == 0 ? 3 * n + 1 : 4 * n - 3;
end
```

and like this in Rust:

```
fn f(n: int) -> int {
    return if n % 2 == 0 {3 * n + 1} else {4 * n - 3}
}
```

***Exercise**: Which of these languages required that variables or functions be declared with types and which did not?*

***Exercise**: Implement this function in PHP, Objective C, Go, D, and Mercury.*

System Languages

System programming languages differ from *application programming languages* in that they are more concerned with managing a computer system rather than solving general problems in health care, game playing, or finance. System languages deal with:

- Memory management
- Process management
- Data transfer
- Caches
- Device drivers
- Operating systems

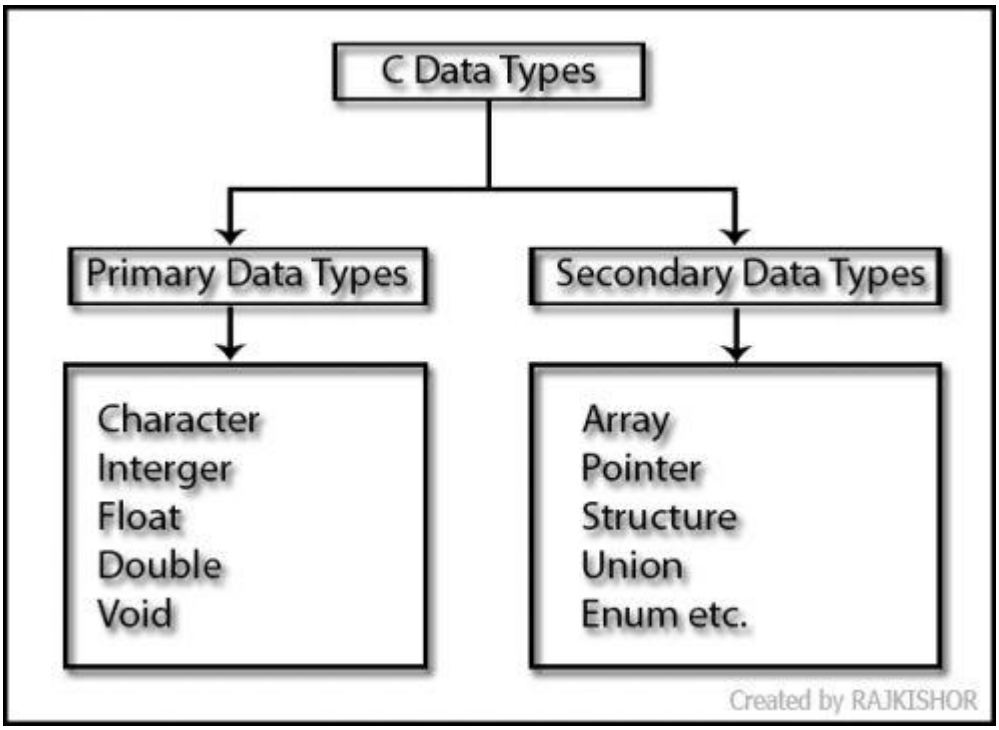
Ans. 4. A **programming language** is proposed to help programmer to process certain kinds of data and to provide useful output. The task of data processing is accomplished by executing series of commands called program.A program usually contains different types of data types (integer, float, character etc.) to store the values being used in the program along with **some library function**and **user defined function (UDF)**to process that stored data. C language is rich of data types and library function. A C programmer has to employ proper data type as per his/her requirement.

C has different data types for different types of data and can be broadly classified as :

1. Primary data types
2. Secondary data types

Primary data types consist following data types.

Data Types in C



Source: Various data types which is available in C language

Integer types:

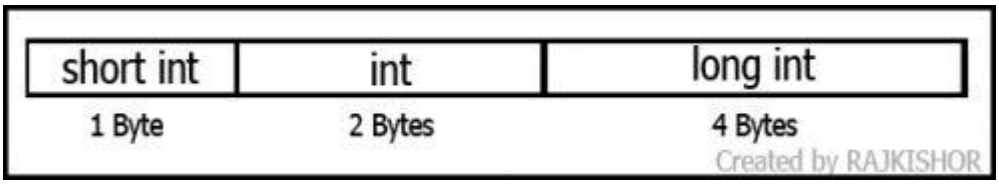
Integers are whole numbers with a range of values, range of values are machine dependent. Generally an integer occupies 2 bytes memory space and its value range limited to -32768 to +32767 (that is, -2^{15} to $+2^{15}-1$). A signed integer use one bit for storing sign and rest 15 bits for number.
To control the range of numbers and storage space, C has three classes of integer storage namely short int, int and long int. All three data types have signed and unsigned forms. A short int requires half the amount of storage than normal integer. Unlike signed integer, unsigned integers are always positive and use all the bits for the

magnitude of the number. Therefore the range of an unsigned integer will be from 0 to 65535. The long integers are used to declare a longer range of values and it occupies 4 bytes of storage space.

Syntax: int <variable name>; like
int num1;
short int num2;
long int num3;

Example: 5, 6, 100, 2500.

Integer Data Type Memory Allocation



Source: Integer data type memory allocation.

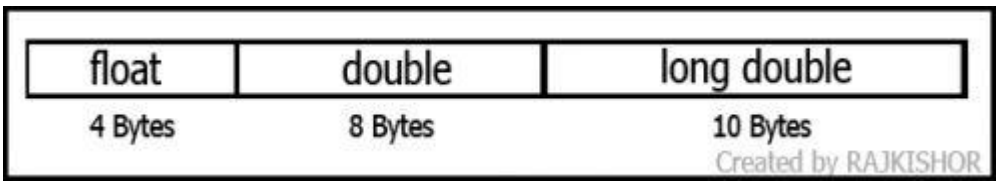
Floating Point Types:

The float data type is used to store fractional numbers (real numbers) with 6 digits of precision. Floating point numbers are denoted by the keyword float. When the accuracy of the floating point number is insufficient, we can use the double to define the number. The double is same as float but with longer precision and takes double space (8 bytes) than float. To extend the precision further we can use long double which occupies 10 bytes of memory space.

Syntax: float <variable name>; like
float num1;
double num2;
long double num3;

Example: 9.125, 3.1254.

Floating Point Data Type Memory Allocation



Source: Float data type memory allocation

Character Type:

Character type variable can hold a single character. As there are singed and unsigned int (either short or long), in the same way there are signed and unsigned chars; both occupy 1 byte each, but having different ranges.

Unsigned characters have values between 0 and 255, signed characters have values from –128 to 127.

Syntax: char <variable name>; like
char ch = ‘a’;

Example: a, b, g, S, j.

Void Type:

The void type has no values therefore we cannot declare it as variable as we did in case of integer and float.

The void data type is usually used with function to specify its type. Like in our first C program we declared “main()” as void type because it does not return any value. The concept of returning values will be discussed in detail in the C function hub.

Secondary Data Types

- Arrays in C Programming (Read it now)
  An array in C language is a collection of similar data-type, means an array can hold value of a particular data type for which it has been declared. Arrays can be created from any of the C data-types int,...
- Pointers in C Programming (Read it now)
  In this tutorial I am going to discuss what pointer is and how to use them in our C program. Many C programming learner thinks that pointer is one of the difficult topic in C language but its not...
- Structure in C Programming (Read it now)
  We used variable in our C program to store value but one variable can store only single piece information (an integer can hold only one integer value) and to store similar type of values we had to declare...

User defined type declaration

C language supports a feature where user can define an identifier that characterizes an existing data type. This user defined data type identifier can later be used to declare variables. In short its purpose is to redefine the name of an existing data type.

Syntax: typedef <type> <identifier>; like
typedef int number;

Now we can use number in lieu of int to declare integer variable. For example: “int x1” or “number x1” both statements declaring an integer variable. We have just changed the default keyword “int” to declare integer variable to “number”.

Data Types in C, Size & Range of Data Types.

Variable Type	Keyword	Bytes Required	Range
Character	char	1	-128 to 127
Unsigned character	unsigned char	1	0 to 255
Integer	int	2	-32768 to 32767
Short Integer	short int	2	-32768 to 32767
Long Integer	long int	4	-2,147,483,648 to 2,147,438,647
Unsigned Integer	unsigned int	2	0 to 65535
Unsigned Short Integer	unsigned short int	2	0 to 65535
Unsigned Long Integer	unsigned long int	4	0 to 4,294,967,295
Float	float	4	1.2E-38 to
Double	double	8	2.2E-308 to
Long Double	long double	10	3.4E-4932 to 1.1E+4932

Ans. 5. a)

```
#include<stdio.h>
main()
{
int no;
printf (“To check a number is even or odd: “);
printf (“Enter a number : “);
scanf (“%d”,&no);
if (no%2==0)
{
printf (“Result: EVEN number.\n”);
}
else
{
printf (“Result: ODD.\n”);
}
}
```

OUTPUT:

To check a number is even or odd:

Enter a number :2

Result: EVEN number.

RUN2:

To check a number is even or odd:

Enter a number :3

Result: ODD number

Ans. b) The ability to control the flow of your program, letting it make decisions on what code to execute, is valuable to the programmer. The if statement allows you to control if a program enters a section of code or not based on whether a given condition is true or false. One of the important functions of the if statement is that it allows the program to select an action based upon the user's input. For example, by using an if statement to check a user-entered password, your program can decide whether a user is allowed access to the program. Without a conditional statement such as the if statement, programs would run almost the exact same way every time, always following the same sequence of function calls. If statements allow the flow of the program to be changed, which leads to more interesting code.

Before discussing the actual structure of the if statement, let us examine the meaning of TRUE and FALSE in computer terminology. A true statement is one that evaluates to a nonzero number. A false statement evaluates to zero. When you perform comparison with the relational operators, the operator will return 1 if the comparison is true, or 0 if the comparison is false. For example, the check 0 == 2 evaluates to 0. The check 2 == 2 evaluates to a 1. If this confuses you, try to use a printf statement to output the result of those various comparisons (for example printf ("%d", 2 == 1);)

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each **switch case**.

The following rules apply to a **switch** statement:

- The **expression** used in a **switch** statement must have an integral or enumerated type, or be of a class type in which the class has a single conversion function to an integral or enumerated type.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The **constant-expression** for a case must be the same data type as the variable in the switch, and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a **break** statement is reached.
- When a **break** statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a **break**. If no **break** appears, the flow of control will *fall through*to subsequent cases until a break is reached.
- A **switch** statement can have an optional **default** case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No**break** is needed in the default case.

Unit-3

Ans. 6. Loops are very basic and very useful programming facility that facilitates programmer to execute any block of code lines repeatedly and can be controlled as per conditions added by programmer. It saves writing code several times for same task.

There are three types of loops in C.

1. For loop
2. Do while loop
3. While loop

1. For Loop Examples

Basic syntax to use 'for' loop is:

```
for (variable initialization; condition to control loop; iteration of variable) {

   statement 1;

   statement 2;

   ..

   ..

}
```

In the pseudo code above :

- **Variable initialization** is the initialization of counter of loop.
- **Condition** is any logical condition that controls the number of times the loop statements are executed.
- **Iteration** is the increment/decrement of counter.

It is noted that when ‘for’ loop execution starts, first **variable initialization** is done, then condition is checked before execution of statements; if and only if **condition** is TRUE, statements are executed; after all statements are executed, **iteration** of counter of loop is done either increment or decrement.
Here is a basic C program covering usage of ‘for’ loop in several cases:

```
#include <stdio.h>



int main () {



   int i = 0, k = 0;

   float j = 0;

   int loop_count = 5;



   printf("Case1:\n");

   for (i=0; i < loop_count; i++) {

      printf("%d\n",i);

   }



   printf("Case2:\n");

   for (j=5.5; j > 0; j--) {
```

```
    printf("%f\n",j);
}

printf("Case3:\n");
for (i=2; (i < 5 && i >=2); i++) {
    printf("%d\n",i);
}

printf("Case4:\n");
for (i=0; (i != 5); i++) {
    printf("%d\n",i);
}

printf("Case5:\n");
/* Blank loop  */
for (i=0; i < loop_count; i++) ;
```

```
    printf("Case6:\n");

    for (i=0, k=0; (i < 5 && k < 3); i++, k++) {

        printf("%d\n",i);

    }



    printf("Case7:\n");

    i=5;

    for (; 0; i++) {

        printf("%d\n",i);

    }



    return 0;

}
```

Do While Loop Examples

It is another loop like ‘for’ loop in C. But do-while loop allows execution of statements inside block of loop for one time for sure even if condition in loop fails.

Basic syntax to use ‘do-while’ loop is:

```
variable initialization;
```

```
do {

statement 1;

statement 2;

..

..

iteration of variable;

} while (condition to control loop)
```

I the pseudo code above :

- **Variable initialization** is the initialization of counter of loop before start of ‘do-while’ loop.
- **Condition** is any logical condition that controls the number of times execution of loop statements
- **Iteration** is the increment/decrement of counter

Here is a basic C program covering usage of ‘do-while’ loop in several cases:

```
#include <stdio.h>

int main () {

int i = 0;

int loop_count = 5;



printf("Case1:\n");
```

```
do {

printf("%d\n",i);

i++;

} while (i<loop_count);



printf("Case2:\n");

i=20;

do {

printf("%d\n",i);

i++;

} while (0);



printf("Case3:\n");

i=0;

do {

printf("%d\n",i);

} while (i++<5);
```

```
printf("Case4:\n");

i=3;

do {

printf("%d\n",i);

i++;

} while (i < 5 && i >=2);

return 0;

}
```

While Loop Examples

It is another loop like ‘do-while’ loop in C. The ‘while’ loop allows execution of statements inside block of loop only if condition in loop succeeds.

Basic syntax to use ‘while’ loop is:

```
variable initialization;

 while (condition to control loop) {

 statement 1;

 statement 2;

 ..
```

```
..

iteration of variable;

}
```

In the pseudo code above :

- **Variable initialization** is the initialization of counter of loop before start of ‘while’ loop
- **Condition** is any logical condition that controls the number of times execution of loop statements
- **Iteration** is the increment/decrement of counter

Basic C program covering usage of ‘while’ loop in several cases:

```
#include <stdio.h>



int main () {



   int i = 0;

   int loop_count = 5;



   printf("Case1:\n");

   while (i<loop_count) {

      printf("%d\n",i);
```

```
    i++;
}

printf("Case2:\n");
i=20;
while (0) {
    printf("%d\n",i);
    i++;
}

printf("Case3:\n");
i=0;
while (i++<5) {
    printf("%d\n",i);
}
printf("Case4:\n");
i=3;
```

```
    while (i < 5 && i >=2) {

        printf("%d\n",i);

        i++;

    }



    return 0;

}
```

Ans. 7. #include <stdio.h>

```
int main()
{
  int m, n, p, q, c, d, k, sum = 0;
  int first[10][10], second[10][10], multiply[10][10];

  printf("Enter the number of rows and columns of first matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");

  for (  c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
      scanf("%d", &first[c][d]);

  printf("Enter the number of rows and columns of second matrix\n");
  scanf("%d%d", &p, &q);
```

```
if ( n != p )
  printf("Matrices with entered orders can't be multiplied with each other.\n");
else
{
  printf("Enter the elements of second matrix\n");

  for ( c = 0 ; c < p ; c++ )
    for ( d = 0 ; d < q ; d++ )
      scanf("%d", &second[c][d]);

  for ( c = 0 ; c < m ; c++ )
  {
    for ( d = 0 ; d < q ; d++ )
    {
      for ( k = 0 ; k < p ; k++ )
      {
        sum = sum + first[c][k]*second[k][d];
      }

      multiply[c][d] = sum;
      sum = 0;
    }
  }

  printf("Product of entered matrices:-\n");

  for ( c = 0 ; c < m ; c++ )
  {
    for ( d = 0 ; d < q ; d++ )
      printf("%d\t", multiply[c][d]);

    printf("\n");
  }
}

return 0;
```

Unit-4

Ans. 8. /Call by Value Example - Swapping 2 numbers using Call by Value

```
1.  #include <stdio.h>
2.
3.
4.  void swap(int, int);
5.
```

```
6.  int main()
7.  {
8.  int x, y;
9.
10. printf("Enter the value of x and y\n");
11. scanf("%d%d",&x,&y);
12.
13. printf("Before Swapping\nx = %d\ny = %d\n", x, y);
14.
15. swap(x, y);
16.
17. printf("After Swapping\nx = %d\ny = %d\n", x, y);
18.
19. return 0;
20. }
21.
22. void swap(int a, int b)
23. {
24. int temp;
25.
26. temp = b;
27. b = a;
28. a = temp;
29. printf("Values of a and b is %d %d\n",a,b);
30. }
```

```
/*swap function to interchange values by call by reference*/
#include<stdio.h>
#include<conio.h>
void swaping(int *x, int *y);
int main()
{
 int n1,n2;
 printf("Enter first number (n1) : ");
 scanf("%d",&n1);
 printf("Enter second number (n2) : ");
 scanf("%d",&n2);
 printf("\nBefore swapping values:");
 printf("\n\tn1=%d \n\tn2=%d",n1,n2);
 swaping(&n1,&n2);
 printf("\nAfter swapping values:");
 printf("\n\tn1=%d \n\tn2=%d",n1,n2);
 getch();
 return 0;
}
void swaping(int *x, int *y)
```

```
{
  int z;
  z=*x;
  *x=*y;
  *y=z;
}
```

Ans. 9.a) If you want to pass a single-dimension array as an argument in a function, you would have to declare function formal parameter in one of following three ways and all three declaration methods produce similar results because each tells the compiler that an integer pointer is going to be received. Similar way you can pass multi-dimensional array as formal parameters.

Way-1

Formal parameters as a pointer as follows. You will study what is pointer in next chapter.

```
void myFunction(int *param)
{
.
.
.
}
```

Way-2

Formal parameters as a sized array as follows:

```
void myFunction(int param[10])
{
.
.
.
}
```

Way-3

Formal parameters as an unsized array as follows:

```
void myFunction(int param[])
{
.
.
.
```

```
}
```

Example

Now, consider the following function, which will take an array as an argument along with another argument and based on the passed arguments, it will return average of the numbers passed through the array as follows:

```
double getAverage(int arr[], int size)
{
  int    i;
  double avg;
  double sum;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = sum / size;

  return avg;
}
```

Now, let us call the above function as follows:

```
#include <stdio.h>

/* function declaration */
double getAverage(int arr[], int size);

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value */
   printf( "Average value is: %f ", avg );

   return 0;
}
```

When the above code is compiled together and executed, it produces the following result:

```
Average value is: 214.400000
```

b) Through file handling, one can perform operations like create, modify, delete etc on system files. Here in this article I try to bring in the very basic of file handling. Hope this article will clear the top layer of this multilayer aspect.

File handling functions

In this article, we will cover the following functions that are popularly used in file handling :

fopen()

```
FILE *fopen(const char *path, const char *mode);
```

The fopen() function is used to open a file and associates an I/O stream with it. This function takes two arguments. The first argument is a pointer to a string containing name of the file to be opened while the second argument is the mode in which the file is to be opened. The mode can be :

- 'r' : Open text file for reading. The stream is positioned at the beginning of the file.
- 'r+' : Open for reading and writing. The stream is positioned at the beginning of the file.
- 'w' : Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.
- 'w+' : Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.
- 'a' : Open for appending (writing at end of file). The file is created if it does not exist. The stream is positioned at the end of the file.
- 'a+' : Open for reading and appending (writing at end of file). The file is created if it does not exist. The initial file position for reading is at the beginning of the file, but output is always appended to the end of the file.

The fopen() function returns a FILE stream pointer on success while it returns NULL in case of a failure.

fread() and fwrite()

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

The functions fread/fwrite are used for reading/writing data from/to the file opened by fopen function. These functions accept three arguments. The first argument is a pointer to buffer used for reading/writing the data. The data read/written is in the form of ‘nmemb’ elements each ‘size’ bytes long.

In case of success, fread/fwrite return the number of bytes actually read/written from/to the stream opened by fopen function. In case of failure, a lesser number of byes (then requested to read/write) is returned.