

Q1  
Sol.

What is for loop? and its syntax with example

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

SYNTAX :-

```
for(initialization statement; condition; updates)
{
    // statements
}
```

EXAMPLE :-

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

```
int main()
```

```
{
```

```
    int a;
```

```
    for (a=10; a<20; a++)
```

```
    {
```

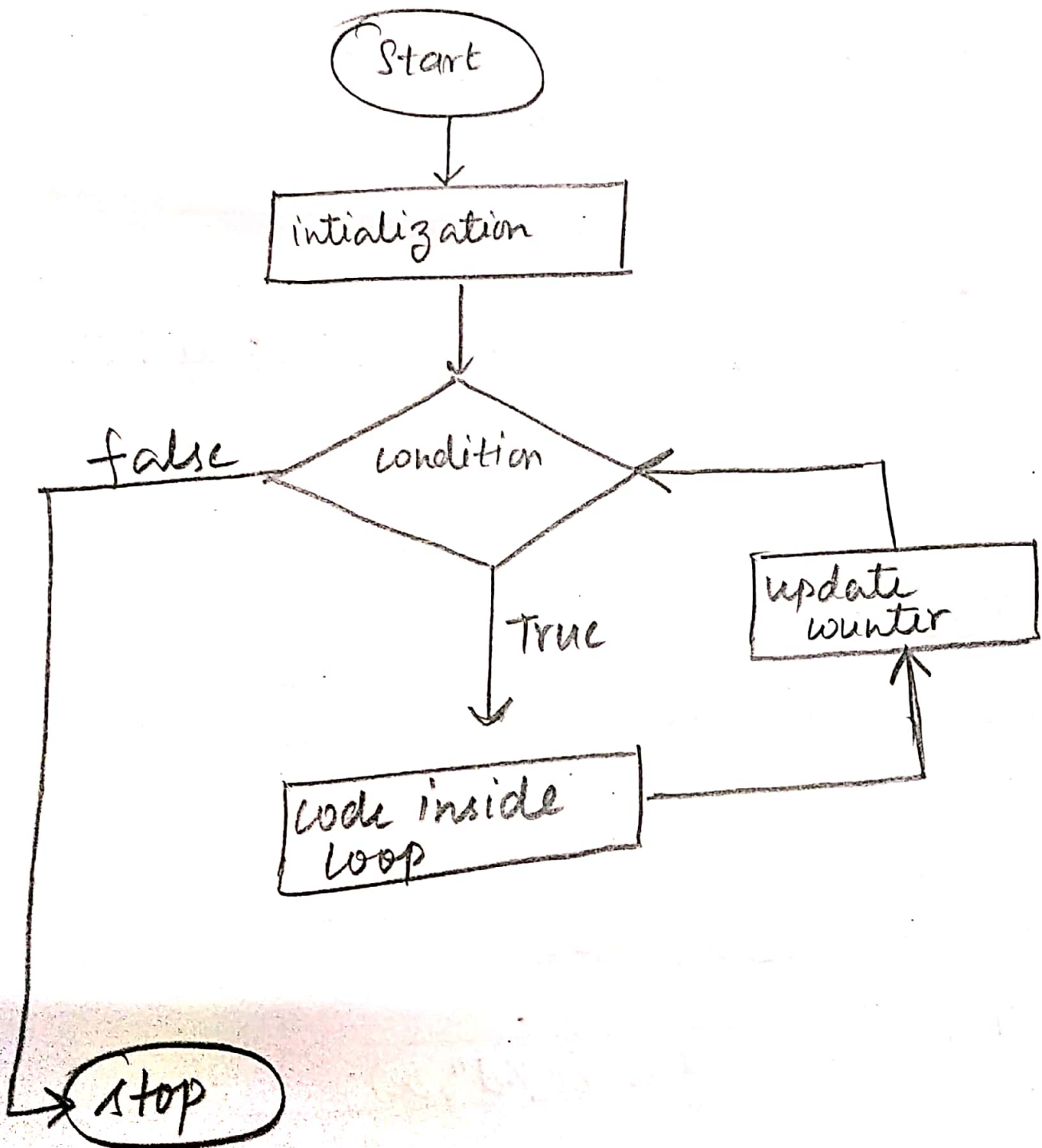
```
        printf("value of a is %d", a);
```

```
    }
```

```
    return 0;
```

```
}
```

## Execution of for loop



OUTPUT :-

x — x — x — x —  
value of a is 10  
" 11  
" 12  
" 13  
" 14  
" 15  
" 16  
" 17  
" 18  
" 19

Q2

What is while loop? and its syntax with an example.

A WHILE LOOP is the most straightforward looping structure. The basic format of while loop is as follow :-

SYNTAX :-

```
while (condition)
{
    statements;
}
```

EXAMPLE :-

```
#include <stdio.h>
void main()
{
    int num = 1;
    while (num <= 10)
    {
        printf("%d \n", num);
        num++;
    }
    return 0;
}
```

Date :

Page No. : 4

## OUTPUT

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

What is DO-while? and its syntax with an example.

A DO-WHILE is similar to the while loop except that the condition is always executed after the body of a loop. it is also called an exit-controlled loop.

### SYNTAX

```
do  
{  
    statement  
}  
while (expression);
```

EXAMPLE

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

```
void main()
{
```

```
    int num = 1;
```

```
    do
```

```
    {
```

```
        printf("%d \n", 2 * num);
```

```
        num ++;
```

```
    }
```

```
    while (num <= 10);
```

```
    getch();
```

```
}
```

OUTPUT

2

18

4

20

6

8

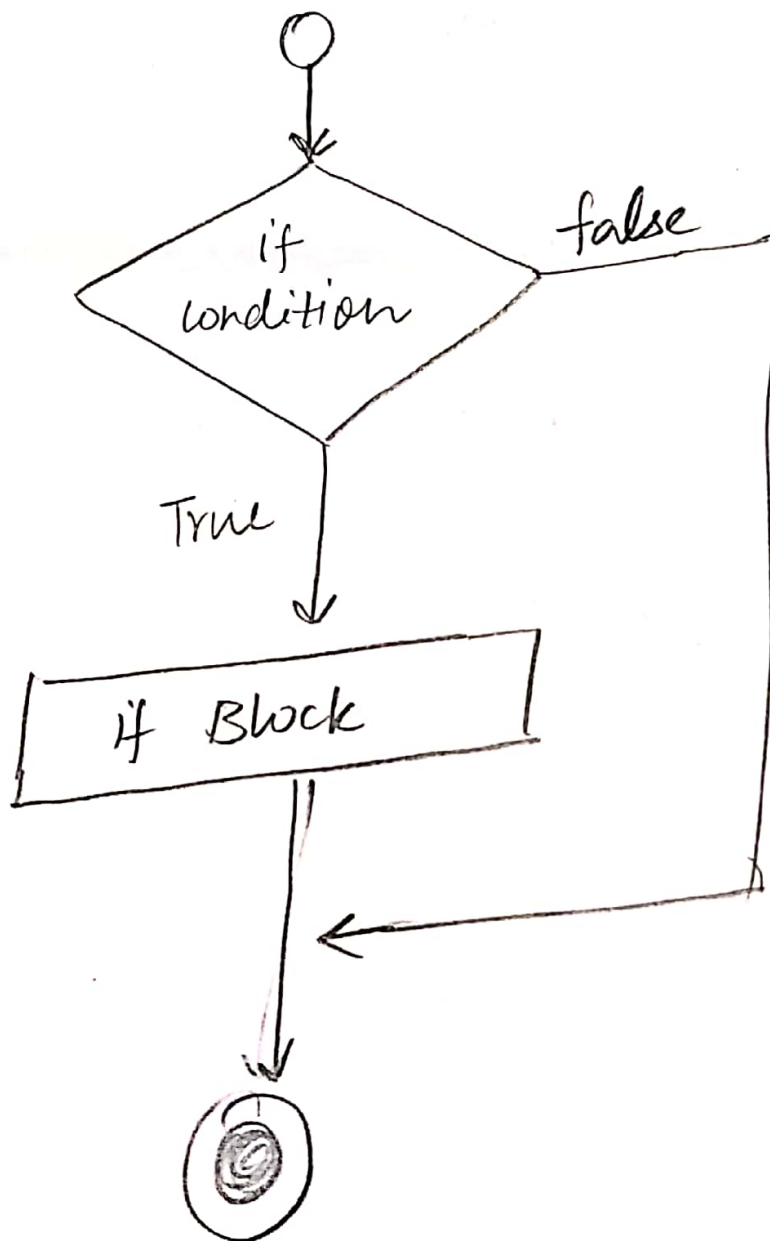
10

12

14

16

# IF STATEMENT



# CONDITIONAL STATEMENTS

Date :  
Page No. : 7

\*

IF STATEMENT :-

The single if statement is used to execute the code if a condition is true. It is also called one-way selection statement.

SYNTAX

```
if (expression)  
{
```

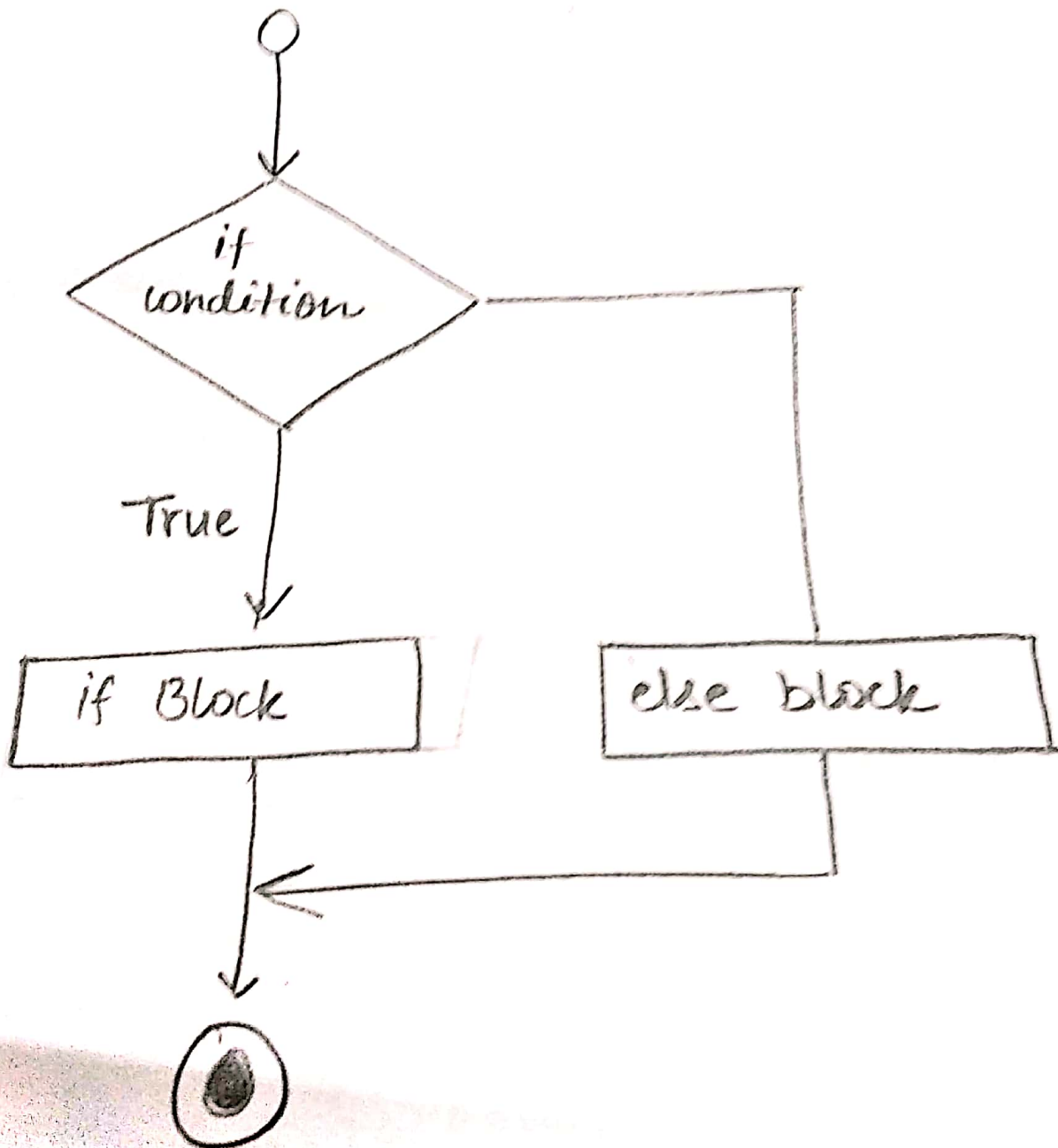
```
    // code to be executed  
}
```

EXAMPLE

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num = 0;
    printf("Enter the number");
    scanf("%d", &num);
    if (num % 2 == 0)
    {
        printf("%d number is even", num);
    }
    getch();
}
```

OUTPUT

## IF - ELSE STATEMENT



\*

## IF - ELSE STATEMENT :-

The If-else statement is used to execute the code if condition is true or false. It is also called two-way selection statement.

### SYNTAX

```
if (expression)
{
    // statements
}
else
{
    // statements
}
```

EXAMPLE

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int num = 0;
    printf("Enter the number");
    scanf("%d", &num);
    if (num % 2 == 0)
    {
        printf("%d number is even", num);
    }
    else
    {
        printf("%d number is odd", num);
    }
    getch();
}
```

~~Output~~

### EXAMPLE

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

```
#include <conio.h>
```

```
void main()
```

```
{
```

```
    int a, b, c;
```

```
    printf("Enter 3 number");
```

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

```
    if (a > b)
```

```
    {
```

```
        if (a > c)
```

```
        {
```

```
            printf("a is greatest");
```

```
        }
```

```
    else
```

```
    {
```

```
        printf("c is greatest");
```

```
    }
```

```
}
```

```
else
```

```
{ if (b > c)
```

```
{
```

```
    printf("b is greatest");
```

```
}
```

```
else
```

```
{
```

```
    printf("c is greatest");
```

```
}
```

```
}
```

```
getch();
```

```
}
```

## \* NESTED IF-ELSE STATEMENT :-

The nested if else statement is used when a program requires more than one test expression. It is also called a multi-way selection statement.

### SYNTAX

```
if (expression)
{
    if (expression)
    {
        statement - block;
    }
    else
    {
        statement - block 2;
    }
}
else
{
    statement - block 3;
}
```

\*

## SWITCH STATEMENT :-

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.

### EXAMPLE

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

```
int main()
```

```
{
```

```
    int i = 2;
```

```
    switch(i)
```

```
    {
```

```
        case 1:
```

```
            printf("No. is 1.\n");
```

```
            break;
```

```
        case 2:
```

```
            printf("No. is 2.\n");
```

```
            break;
```

```
        case 3:
```

```
            printf("No. is wrong");
```

```
    }
```

```
    return 0;
```

```
}
```

## \* BREAK STATEMENT :-

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

### SYNTAX

Break;

### EXAMPLE

```
#include <stdio.h>
int main()
{
    int a;
    for(a=1; a<=10; a++)
    {
        printf("Hello world\n");
        if(a==2)
        {
            break;
        }
    }
    return 0; }
```

## CONTINUE STATEMENT :-

Instead of forcing termination, it forces the next iteration of the loop to take place skipping any code in between.

### SYNTAX

Continue;

### EXAMPLE

```
#include <stdio.h>
int main()
{
    int a;
    for (a=1; a<=10; a++)
    {
        printf("Huy\n");
        if (a==2)
        {
            continue;
        }
        printf("a is not 2\n");
    }
    return 0;
}
```

## \* TYPES OF MICROCONTROLLER IN EMBEDDED SYSTEM

### 1.) BASED ON BIT-CONFIGURATION

i) 8-bit MC :- This type of MC is used to execute arithmetic or logical operations like :-  $+$ ,  $-$ ,  $\%$ ,  $\times$ .

ii) 16-bit MC :- This type of MC is perform arithmetic or logical, where higher accuracy and performance is required.

iii) 32-bit MC :- This is special type of MC which is used in automatically control application like:- automatic operational machine or medical equipment.

## 2.) BASED ON MEMORY CONFIGURATION

i) External memory MC :- This is design in such a way that they do not have program memory in chip.  
eg :- intel 8031

ii) Embedded memory MC :- This is design in such a way that MC has all ~~to~~ program and data memory.  
like :- counter and timer.  
eg :- 8051 MC

## 3.) BASED ON INSTRUCTION SET

i) CISC (Complex instruction set computer)

ii) RISC (Reduced instruction set computer)

## APPLICATION OF MC's :-

MC is used for application like light sensing.

MC has another application like temperature sensing.

examples :-

- i) 8051 MC
- ii) ATMEL MC
- iii) ARM MC
- iv) Arduino MC

## Unit-3

### Embedded 'C' language basics

\* Variables :- A variable is nothing but a name given to a storage area that our programs can manipulate.

Each variable in C has a specific type, which determines the size and layout of the variable's memory. It must begin with either a letter or underscore. upper and lower case letters are distinct because C is case-sensitive.

Variable types :-

1. char
2. int
3. float
4. double
5. void

example:-

```
#include <stdio.h>
#include <conio.h>
void main() {
    int a, b, sum; // declaration of variable

    printf("Enter first no:");
    scanf("%d", &a);
    printf("Enter second no:");
    scanf("%d", &b);
    sum = a + b;
    printf("Print sum:");
    printf("Sum is %d:", sum);
    getch();
}
```

## \* Identifiers \*

Identifiers refers to name given to entities such as variable, functions, structures etc.

Identifiers ~~must~~ must be unique. They are created to give a unique name to an entity to identify it during the execution of the program

examples:-

```
int money;  
double accountBalance;
```

Here money and accountBalance are identifiers  
\*\*\* Also remember, identifiers must be different from keywords.

## Rules for naming identifiers:-

1. A valid identifier can have letter.
2. first letter should be either letter or underscore
3. You can't use keywords as identifiers.
4. There is no rule on how long an identifier can be

\* Give meaningful names to identifier that make sense.

## \* Built in data types \*

C supports a large number of data types.

The built-in or basic data types supported by C are integer, floating point, character, boolean, double, long integer, long, double, short

| Type      | Description               | Size    |
|-----------|---------------------------|---------|
| char      | A character               | 1 byte  |
| int       | An integer                | 4 bytes |
| float     | small real number         | 4 bytes |
| double    | double precision real no. | 8 bytes |
| short     | A short integer           | 2 bytes |
| long      | A double short            | 4 bytes |
| long long | A double long             | 8 bytes |
| BOOL      | boolean (signed char)     | 1 byte  |

10th feb 2020,

Identifiers :- In the C programming language an identifier is a combination of alphanumeric

Data types :- There are 2 types in C language.

- i) Primitive
- ii) Non-primitive

i) Primitive :- A primitive data type is built into language int, char, long etc

ii) Non-primitive :- A NP is an abstract data type that is built from primitive data type union, stack, queue etc

Any  
Operations in C language :-

An operator is a symbol which helps the user to command the computer to do a certain mathematical or logical manipulation.

Some operators are like :-

- i) Arithmetic
- ii) Relational
- iii) logical
- iv) ~~Bitwise~~ Increment/decrement
- v) Conditional
- vi) Bitwise

- i) Arithmetic operators :-
- ① addition (+)
  - ② subtraction (-)
  - ③ multiplication (\*)
  - ④ division (/) (Quotient)
  - ~~sum~~ ⑤ modulus (%) (Remainder)

- ii) Relational operator :-
- ① < (less than)
  - ② <= (less than equal to)
  - ③ > (greater than)
  - ④ >= (greater than equal to)
  - ⑤ ~~sum~~ comparison op. ==
  - ⑥ equal to op. =
  - ⑦ ~~\*~~ != (not equal to)

- iii) Logical operator :-
- ① && (logical and)
  - ② || (logical or)
  - ~~sum~~ ③ ! (logical not)

~~sum~~ iv) **CONDITIONAL OPERATORS** :- This operator is consist of three operands also called ternary operator. The syntax is as follows

$$exp1 ? exp2 : exp3$$

example :-

`num % 2 == 0 ? printf("even") : printf("odd")`

4th feb

## Characteristics of C

1. Small size
2. Extensive use of functions
3. structure language
4. Pointer implementation
5. Middle level language

Imp  
**Debugging** :- It is a way to remove bugs (error) usually logical errors from program that are currently being coded. A debugger used to complete the process. A debugger a computer program that runs your program line by line and examining the value of variable.

**Testing** :- It is a technique for evaluating product quality and also for indirectly improving it, by identifying defects and problems.

Imp  
**Compiler** :- It is a system software that translates HLL (high level language) program into machine ~~code~~ understandable language it also check the syntax error. eg- Turbo C

Imp  
**Interpreter** :- It translates the HLL into machine language. it compile the entire program statement line by line. eg:- Anaconda

V. Imp  
**Linker** :- It combines several object modules into one. this module may either be executable object or library.

## \* Constants and literals \*

- Constants :- As the name suggests the name constants is given to such variables or values in C. which ~~cannot~~ can't be modified once they are defined. They are fixed value in program.  
There can be any types of constants like integer float, octal, etc.  
Every constant have same range.

example:-

|         |           |                  |   |               |
|---------|-----------|------------------|---|---------------|
| const   | int       | var              | = | 5;            |
| ↓       | ↓         | ↓                | ↓ |               |
| keyword | data type | name of constant |   | initial value |

## ● Literals:-

Constant values used within a program are known as Literals. These constants values occupy memory but do not have any reference like variables.

Literal is a notation for representing a fixed value within a source code.  
There are 4 type of literals in C:-

1. Integer
2. float or real
3. Character
4. String

## \* Arrays declaring and manipulating single dimensional array \*

Array :-

An array is a collection of data items, all of the same type, accessed using a common name.

A one-dimensional / single-dimensional array is like a list.

A two dimensional / multi-dimensional array is like a table.

Declaring arrays :-

```
type arrayName [arraysize];
```

This is called single dimensional array. The arraysize must be an ~~integer~~ integer.

example :-

```
double balance[10];
```

Here balance is a variable array which is sufficient to hold up to 10 double numbers.

→ update, insert, delete

## \* manipulating single dimensional array

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

```
#include <conio.h>
```

```
void main()
```

```
{
```

```
    int i;
```

```
    float x[10], value, total;
```

```
    printf(" ENTER 10 REAL NUMBERS\n");
```

```
    for (i=0; i<10; i++)
```

```
    {
```

```
        scanf("%f", &value);
```

```
        x[i] = value;
```

```
    }
```

```
    total = 0.0;
```

```
    for (i=0; i<10; i++)
```

```
        total = total + x[i] * x[i];
```

```
    for (i=0; i<10; i++) → printf("\n");
```

```
    printf("x[%2d] = %.5.2f\n", i+1, x[i]);
```

```
    printf("\n total = %.2f\n", total);
```

```
    getch();
```

```
}
```

Output:-

ENTER 10 REAL NUMBERS

1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.10

x[1] = 1.10

x[2] = 2.20

x[3] = 3.30

x[4] = 4.40

x[5] = 5.50

x[6] = 6.60

x[7] = 7.70

x[8] = 8.80

x[9] = 9.90

x[10] = 10.10

total = 446.86