Introduction to Programming Language Fundamental

Programming Language

In general, a language is a medium of communication between two parties or among parties. Parties may be a person, animal or machineries

The language that is used to communicate with the computers is called **programming language** or computer language. Like any human understandable language, a programming language consists of its own alphabets, words, syntax and semantics.

Evolution of programming Language

(Machine language, Assembly Language, High level language)

Machine Language: This is the only language that a computer can understand directly without help of any interpreter. The machine language consists of strings of 0s and 1s, called binary codes. In this language, some fixed numbers of bits are used to represent an operation (Addition, subtraction etc) and some fixed numbers of bits are used to represent an operand or data on which operation will be applied.

- It is very difficult to write and understand for a human being.
- This language is machine dependent, as code for one computer does not work for another computer of different make.

Assembly Language: Instead of using binary digits for an instruction, in assembly language mnemonics are used to make this language more readable. It uses names for represent the any operation.

Example:

```
MOV  R1, #37
ADD   R1, #34
```

As the computer only understands machine language, A language translator, called assembler is used to convert this language to machine language.This language is easier to write that machine language, but runs slower.

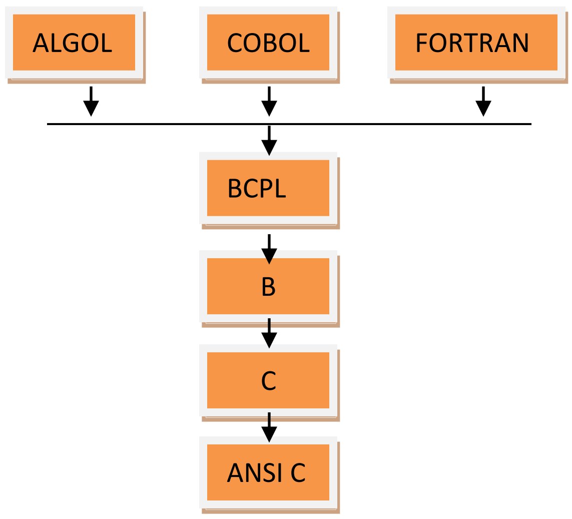
**Both machine and assembly language is called as low level language. To write programs in low level language, programmer must be well conversant with the detail hardware structure of the target machine/computer.*

High level language: High level language is very close to human language that easier to write and more user friendly. The program written in this language is slower as compared to low level language because high level language needs to be converted to machine language before its execution. As language translator, either compiler or interpreter or both are used to convert high level language to low level language.

Programming language come under high level category are Ada, ALGOL, Simula, COBOL, FORTAN, BASIC, pearl, ruby, python, C, C++, JAVA, FoxPro and etc.

Introduction to C Programming language

C is a structured, high level programming language developed by American Computer scientist Dennis MacAlistair Ritchie at AT & T's Bell Lab in during 1972 and 1973.It gained its popularity during 1980 after availability on various compilers from different vendors.



Like any other language, C language has also many antecedents. In 1960, First structured language was developed called ALGOL which gives a light to computer scientist to develop more powerful language. In 1967, Martin Richard developed a general purpose language by using the features of COBOL, FOTRAN and borrowing the structured programming concepts from ALGOL, and call it BCPL(Basic Combined programming Language). Later in 1970 Ken Thompson, Co-worker of Dennis Ritchie modified BCPL by adding some simplicity features to make this language more user friendly. This new language is called B. One major problem of B language is that it is type-less language i,e no data type concepts was incorporated. Otherwise speaking B language has only type i,e integer. Due to this, performance of B language is slow. Then in 1972, Dennis Ritchie make changes in B language by adding the 'Data type" concepts and then name it C. Upto 1972, This version of C is called traditional C. After wide popularity of C in various operating system and platforms, ANSI decided to define a standard for the C compilers and 1989, The C was standardized and referred as ANSI C or C89.In 1990, ISO adopted the ANSI standard of C and named it C90. Then after some amendment and revision C95 was released in 1995, C99 has been released in 1999.Then after a long time C11 was released in 2011 with new additions. Then in 2018, the new version of C, C18 was released with some rectification in C11.

Component of a C program

1. Character set
2. Key word/ Reserved word
3. Variable and constant
4. Data type
5. operators and operands

1. Character sets: - It includes the alphabet A-Z, a-z, digits 0-9 and symbols.

2. Keywords:- Combination of characters that are predefined C library called key word or reserved word. C supports 32 key words

int	double	unsigned	do	continue	default	enum	register
char	long	sizeof	while	goto	return	extern	typedef
float	short	if	for	switch	struct	auto	volatile
void	signed	else	break	case	union	static	const

3. Variable A variable is the name of the temporary memory location (generally RAM) where we store data.

<u>Rules of naming a variable:</u>

1. A variable name consist of alphabets, digits and underscore(_)
2. First letter of variable name must be an alphabet
3. No space is allowed in a variable name.
4. Reserved words cannot be taken as variable name

Constant: Constant is the value that cannot be changed.

Integer constant	: Whole numbers are called integer constant
	66, -234
	034, 020 --→ Octal number system
	oX25, FACE --→ hex decimal system
Real constant	: It includes the fractional type values.
	23.556 , -0.26
	23e2, 2392.98e-3 --→ exponential representation

Character constant: It includes a single character(alphabet, digt, symbol) and is enclosed with a single quotes.

Example: ‘u’ ,’5’.

String constant	: Sequence of characters called string.
	“Hello”, “SSG Mishra”, “7688788990”

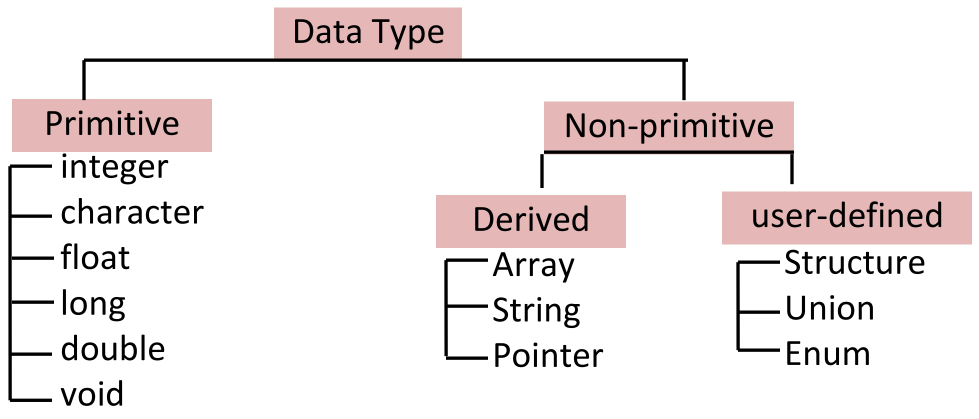
Data type

Data type specifies following about a variable.

type of data it can contain

range of data it can store,

size of memory space it will occupy.



integer: It takes whole numbers.

- keyword – int
- size of integer is compiler dependent. For 16-bit processors it is 2 bytes and for 32 bit / 64 bit processors it is 4 bytes.
- It can take data in the range $-2^{n-1} \quad to \quad +2^{n-1}-1$

  = -2147483648 to 2147483647

  where n = number of bits

unsigned int : An unsigned integer has no signed bit to decide a number is positive or negative. So it cannot take –ve numbers.

It can take data in the range $0 \quad to \quad +2^{n}-1$

signed int : An in type is signed by default.

It has a signed bit. So we can store both +ve and –ve numbers.

float: It takes real numbers e.g : 23.56 , -88.9898

- keyword – float
- size of float is 4 bytes

char : It takes a single character (alphabets, digits, symbols)

- keyword – char
- size of float is 1 byte

long : It can take big integer type data.

- keyword – long / long int
- size of long is 4 bytes.

double : It takes a large number of float type data.

- keyword – double
- size of double is 8 bytes.

void : void mean valueless. It does not occupy any memory space, so declaring a variable as void is not a standard.

- It is used as return type of a function and also used in pointer.
- keyword – void

Operators : Operator operates with the operands. Based on the presence of operands it is three types.

- Unary operator: It takes one operand.
- Binary operator : it takes two operands
- Ternary operator : it takes three operands

Classification of operators on basis of functionality

- Arithmetic operators (+,-,*,/)
- Assignment operators (=)
- Modulo Division operator (%)
- Increment and decrement operators(++ , --)
- Relational operators (<, >, <= ,>= , !=, ==)
- Logical operators (||, &&, !)
- Conditional operator (?:)
- Bit wise operators (&,|,~,^,<<,>>)
- Reference and dereference operator(&,*)
- Member access operators(.,->)
- Others: comma, sizeof (),unary minus

Rules of writing a c program

- A C program must be written in a file with .c extension
- Every program must contain a main() function.
- All the statements of a function must be enclosed with a pair of curly braces {..}
- Statements are case sensitive. (Almost all statements are written in lower case alphabets)
- Every statement must ends with a semi colon.

Part of a C program

1. Start section
2. Declaration section
3. Input section
4. Processing section
5. Output section
6. End

1. Start section: A program starts execution from this part.

```
        main()
            {
```

2. Declaration section: Every variable required throughout the program are declared with their data type.

Syntax: Datatype variable1,variabl2, . . . ;

```
        int num1, num2, result;
```

when a variable is declared, a specific size of memory space (generally RAM) is allocated and is accessed through the name of the variable. This memory is pre-filled with some unknown value, which is called **garbage value.**

3. Input section: In this section, a variable is initialized with some specific value. Two ways, we can input to a variable.

Direct input: Input value is specified directly while designing the program. The programmer knows the input value.

Example :

```
        num1 = 20;
```

Runtime input/ Dynamic input: The input value is given by the program user at the time of execution of program.

scanf() : is a formatted function defined in <stdio.h> , used to make runtime input.

Syntax: scanf("FormatSpecifier",&variableName);

1st argument is format specifier, that specifies the type of data to be input and 2nd argument specifies the memory address, where to store the user data.

Format	Data type
%d	Int
%c	Char
%f	Float
%s	String
%ld	Long int
%lf	Double
%p	Pointer
%u	Unsigned int
%i	Signed int

```
scanf("%d%d",&num1,&num2);
```

4. Processing section: the statements are written according to program requirements.

5. Output section: Results of the programs are sent to the output device.

printf() : This is a formatted output function, defined in <stdio.h>, used to display the data value or any message on monitor. It returns the

Syntax:

1. `printf("Message");`

The statement displays the message at monitor.

2. `printf("FormatSpecifier",variable|expression);`

This statement displays the value of the variable.

Example:

```
printf("Hello! this is our first program");
```

output : Hello! this is our first program

```
int num1 = 20, num2 = 40;
```

```
    printf("%d %d",num1,num2);
```
output : 20 40

```
    printf("The sum is %d",num1+num2);
```
output : The sum is 60

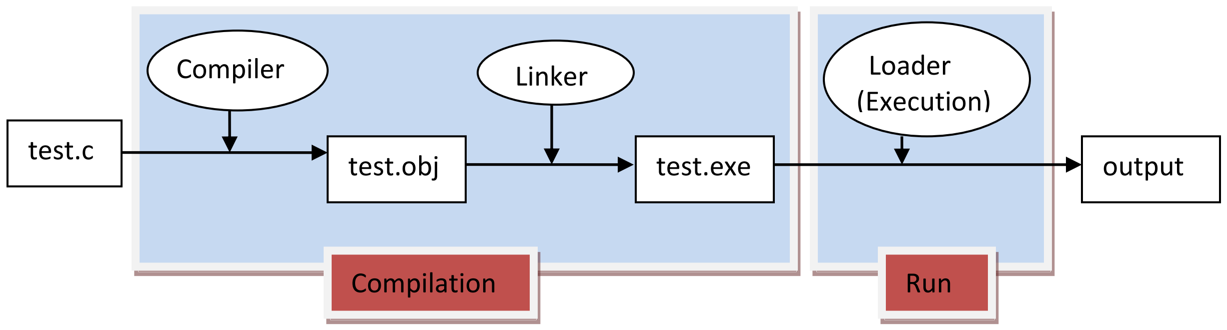
5. End section: Program execution stops when final closing curly brace(}) is encountered.

```
        }
```

Program-Write a program to input length and breadth of a rectangle. calculate its area and then display the area.

```
#include<stdio.h>
int main()
    {
    int length,breadth,area;
    printf("Enter length and breadth:");
    scanf("%d%d",&length,&breadth);
    area = length * breadth;
    printf("Area of Rectangle : %d ",area);
    return 0;
    }
```

<u>Steps to create and execute C program in Windows environment</u>



Extra Information to note:

1. scanf() returns number of input values it scanned.
2. printf() returns the number of characters (including the white space character),it prints.
3. printf("%Nd",variable) : If N is greater than the length of variable, It takes N space to print the variable.

   ```
   printf("%5d",a);
   ```
4. scanf("%Nd",&variable) : if length of the input value is greater that N, then the input value is truncated upto N digits.

   ```
   scanf("%5d",a);
   ```
5. **Comment Lines:** Comment lines are used to describe the program or the statements of the program. It is useful for documentation of the program.

   The comment line are compiled but not executed.

   comment lines cannot be nested.

   For single line comment // symbol is used.

   Example : //This statement finds the area of triangle

   For multi line comments, the statements are enclosed with /*and*/ symbol.

   ```
   /* The program is used to input
   length and breadth of a rectangle
   and  calculate the area . */
   ```

6. **Escape sequence:** is a sequence of two characters started with a '\' and used inside a string literals.

Escape sequence	Meaning
\n	New Line character
\t	Tab character
\a	beep sound
\b	backspace character
\\	backslash
\'	Single quote
\"	Double quote

7. Type casting: Conversion of one data type to another is called typecasting.

Implicit type conversion: Automatic conversion of one type to another is called implicit conversion.

Rules:

- Inside the expression, lower type is converted to higher type.
- In case of assignment, type of right hand side expression is converted to type of left hand side variable.

Explicit type conversion: Force conversion of one type to another by the programmer manually is called explicit type conversion:

Syntax: `(typename)expression.`

Example:
```
float a=18.7;
int i ;
i =(int) (a + 0.5)
```

Program Exercise-1

1. Write a Program to input five real numbers and find out average and sum.
2. Write a Program to input radius of a circle find out its area and circumference.
3. Write a Program to input three side of a triangle find out its area and circumference.
4. Write a Program to input length and breadth of a rectangle; find out its area and circumference.
5. Write a Program to Input a character. Display its ASCII code.
6. Write a Program to find the simple interest for given principal, rate of interest and number of years.
7. Write a Program to find the compound interest for given principal, rate of interest and number of years.
8. Write a Program to solve quadratic equation.
9. Write a Program to calculate gross salary of an employee. Input basic salary through keyboard.
   D.A is 40% of basic salary
   H.R.A is 20% of basic salary.

10. Write a Program to distance between two cities. Convert the distance in meters, feet, inches and centimeters.
11. Write a Program to input temperature of city in Fahrenheit then convert the temperature into centigrade.
12. Write a Program to Input two numbers. Swap them using third variable.
13. Write a Program to Input two numbers. Swap them without using third variable.
14. Write a Program to Input two numbers. Swap them using third variable.
15. Write a Program to input a five-digit number through keyboard find out the sum of its digits.
16. Write a Program to read the price of an item in rupees and print the output in paisa.

Operator

Arithmetic operators (+,-,*,/)

These are used to perform following arithmetic operations on numeric operand.

+	: addition
-	: Subtraction
*	: Multiplication
/	: Division

Type of the result depends upon the type of operands used. If both integer and real type operand are present, then real type supersedes integer type.

example:

float a = 10/3;   printf("%f",a);

float b = 10.0 /3;   printf("%f",b);

Example :

Assignment operators (=)

It assigns the value of right hand side expression in left hand side variable.

a = 10;

Other assignment operators: +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=

<u>Example:</u>

a += 2 implies a = a+2

a -= 2 implies a = a / 2

```
a = 10;
a*=2;
printf("%d",a)
```

```
int a=10,b=20,c;
c=b*=a/=3;
printf("%d%d%d",a,b,c);
```

Modulo Division operator (%)

It returns the reminder on a division operation. It works only on integers. Sign of the reminder is same as the numerator.

```
a = 15 % 6;
b = -15 % 7;
```

Increment and decrement operators (++ , --)

This operator increases/decreases the value of a variable by 1. It can be used only with a variable.

There are two types of increment/decrement operator.

1. pre increment/decrement: If the operator is fixed before a variable in a statement, then the variable is increased or decreased first. Then the statement is executed.

```
    a = 10;
    b = ++a;
    printf ("%d %d ",a,b);
```

2. Post increment/decrement: If the operator is fixed after a variable in a statement, then the statement is executed first. Then the variable is increased or decreased.

```
    a = 10;
    b = a++;
    printf ("%d %d ",a,b);
```

Find output

```
1.  int a = 10,b;
    b = a++ + ++a + a++;
    printf("%d %d ", a,b)
2.  int a = 10,b=6;
    c = a++ + ++b + b++;
    a = c-- + b++;
    printf("%d %d  %d", a,b,c);
3.  int a =10,b=20;
    printf("%d %d", a!=b, a=b);
```

Relational operators : These operators check different types of relation between two values and return a Boolean value indicating the relation is true or false. They return 1 if the relation is true and return 0 otherwise.

Operator	Meaning	Example
<	Less than	23 < 56 returns 1
>	Greater than	12 > 12 returns 0
<=	Less than or equal to	23 <= 56 returns 1
>=	Greater than or equal to	12 >= 12 returns 1
!=	Not equal to	12 != 12 returns 0
==	Exactly equal to	12 == 12 returns 1

Logical operators : There are three logical operators.

Logical AND (&&)

Logical AND (||)

Logical AND (!)

Logical AND (&&) : combines two Boolean expression, returns true(1) if both the expression evaluates true(1). It returns 0 if any one of the Boolean expression evaluates false (0)

If first expression is false, second expression will not be evaluated/not executed.

Logical OR (||) : combines two Boolean expression, returns false(0) if both the expression evaluates false(0). It returns 1 if any one of the Boolean expression evaluates true(1).

If first expression is true, second expression will not be evaluated/not executed.

Exp1	Exp2	Exp1 && Exp2	Exp1 || Exp2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Logical NOT (!)

It is a unary operator. It checks a Boolean expression. It he expression is true (1),it returns 0 and vice-versa.

Exp1	! Exp1
0	1
1	0

Bit wise operators:

These operators perform different type of operations by checking the individual binary digit of numbers. Six operators are

Bitwise AND (&): Compares bits of two numbers. Return 1 if both bits are 1. Returns 0, otherwise.

Bitwise OR (|) : Compares bits of two numbers. Return 0 if both bits are 0. Returns 0, otherwise.

Bitwise XOR(^) : Compares bits of two numbers. Return 0 if both bits are same. Returns 0,if the bits are different.

Bit1	Bit2	Bit1 & Bit2	Bit1 | Bit2	Bit1 ^ Bit2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Complementation (~) : Checks only a bit. Returns 1 if the bit is 0 and vice-versa.

Left shift (<<) : It shifts each bit of a number, a specific number of time towards left.

Syntax: number<<noofshift;

```
int a =25,b;
b= a<<2;              // 100 i,e a * 2^(number of shift)
```

<u>Right shift(>>)</u> : It shifts each bit of a number, a specific number of time towards right.

```
int a =25,b;
b= a>>2;          // 100 i,e a / 2^(number of shift)
```

sizeof operator : This operator returns the memory size required for store a variable, constant.

Syntax: sizeof(variable);
sizeof(constant);
sizeof(datatype);

Note: sizeof() is not a function.

Conditional operator (?:)

It is a ternary operator. It evaluated a Boolean expression. If it is 1, it executes the first expression. If it is false (0), it executes 2nd expression.

Syntax: (Boolean Expression)? <Expression1: expression2 ;

```
    a = 60;
    b= 50;
    (30<40) ? printf("a") ? printf("b");
- ------------- --- ----------------------
    c = a ? b + 10 : b -10;
--------- ----- ------------ ------------------
```

Comma (,):

It separates two expression, variables, and arguments.

When multiple expressions are used with commas, they are executed from left to right.

when multiple expressions are used with comma enclosed with parenthesis, then its value is same as the right most expression.

```
a=10,b=20,c=30;
printf("%d %d %d",a,b,c);
```

```
a=(10,b=20,c=30);
printf("%d %d %d",a,b,c);
--------------------------------
a = (10,20,30);
printf("%d",a);
-----------------------------------
a=10
b=20
d = (a=90,c=a);
printf("%d %d",a,b,c);
```

Program Exercise-2

Using Conditional operator,

1. Write a program to input a number. Check if it is even or odd
2. Write a program to Input a number check if it is +ve or –ve.
3. W.A.P to Input a year check it is a leap year or not.
4. Write a program to read two numbers. Display the larger value.
5. Write a program to read three numbers. Display the largest value.

Precedence and Associativity

precedence of operator determines which operator to be executed first and associativity determines in which direction the operators having same precedence are evaluated.

Operator	Meaning of operator	Associativity
() [] -> .	Functional call Array element reference Indirect member selection Direct member selection	Left to right
! ~ + - ++ -- &	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference (Address)	Right to left

* sizeof(type)	Pointer reference Returns the size of an object Typecast (conversion)	
* / %	Multiply Divide Remainder	Left to right
+ -	Binary plus(Addition) Binary minus(subtraction)	Left to right
<< >>	Left shift Right shift	Left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to right
== !=	Equal to Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to right
||	Logical OR	Left to right
?:	Conditional Operator	Right to left
= *= /= %= += -= &= ^= |= <<= >>=	Simple assignment Assign product Assign quotient Assign remainder Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	Right to left
,	Separator of expressions	Left to right

Control Structure

Controls structure/statement is used to control the sequence of execution of statement.

1. Sequential control statement
2. Decision control statement
3. Case control statement
4. Loop control statement

1. Sequential control statement:

In this structure, the statements are executed as they are written in the editor.

2. Decision control statement

This is used when we want to execute the statements conditionally.

structure :

```
if( condition/Boolean Expression)
    {
    statements;      // If block
    }
else
    {
    statements;      // else block
    }
```

→ if the condition is true, then statements of if block is executed and if the condition is false, then the else block is executed.

→ Else block is optional.

→ If any block is contains a single statement, then enclosing it within pair of curly braces is optional.

→ If and else block can be nested.

Program: Write a program to input a number. Test if the number is even or odd.

```
#include<stdio.h>
int main()
    {
    int number;
    printf("Enter a number:");
    scanf("%d",&number);
    if(number % 2 == 0)
        printf("%d is Even",number);
    else

        printf("%d is Odd",number);
    return 0;
    }
```

Nested If

When an If structure is present within another if structure, it is called nested if.

Structure-1:

```
if(BooleanExpression)
    {
    if(BooleanExpression)
        {
        statements;
        }
    else
        {
        statements;
        }
    }
else
    {
    statements;
    }
```

Structure-2:

```
if(BooleanExpression)
    {
    statements;
    }
else
    if(BooleanExpression)
        {
        statements;
        }
    else
        {
        statements;
        }
    }
```

Structure-3: Else if ladder

```
if(BooleanExpression)
     {
     statements;
     }
else if(BooleanExpression)
     {
     statements;
     }
else if(BooleanExpression)
     {
     statements;
     }
else
     {
     statements;
     }
```

Programs:

1. Input a number. Check if it is even or odd.
2. Input a number. Check if it is +ve or –ve.
3. Input a year. Check it is a leap year or not.
4. Input a character check it lower case alphabet or not.
5. Input cost price and selling price of an item. Determine whether the seller has made profit or loss and also calculate profit or loss amount.
6. Input any character from keyboard to determine whether the character is capital letter, a small case letter, a digit or a special symbol.
7. Input three mark of a student. Calculate average and grade.

    Rules: If avg>=85 grade is 'S'
    If avg>=75 and avg<85 grade is 'A'
    If avg>=65 and avg<75 grade is 'B'
    If avg>=55 and avg<65 grade is 'C'
    If avg>=50 and avg<55 grade is 'D'
    If avg<50 grade is 'F'

8. A cloth show room has announced the following seasonal discounts on
   their clothing products

Purchase Amount	Discount	
	Branded cloth	Local items
0 – 1000	NIL	10%
1001 – 2000	10%	20%
2001 – 3000	20%	25%
Above 3000	25%	50.0%

Calculate the discount amount and net amount to be paid by a customer.

```
Sample run:
Enter purchase amount of branded clothes: 1200
Enter purchase amount of Local clothes: 4000
Net amount payable =       Rs. 3080
Total Discount       =     Rs. 2120
```

Judge the performance of cricketers as per the following rules.

Rules	performance
Average run>40 && avg wicket taken>3	ALL ROUNDER
If avg runs >50	BATSMAN
If avg wicket > 4	BOWLER
Else	FIELDER

To calculate avg run of the following rule.

Avg runs = total run/(number of innings played-number of nut out)

To calculate avg wicket of the following rule.

Avg wickets= number of wickets taken /number of innings played.

Sample run:

```
Enter the name of cricketer:  S.Mishra
Total number of runs second 280
Number of innings played     4
Number of not outs           1
Number of wickets            0
Cricketer S.Mishra is a Batsman
```

3. Case control statement

Another branch control statement used to select one path out of multiple paths.

syntax:
```
switch(expression)
    {
    case constant1 :
         statements;
    case constant2 :
         statements;
    :    :
    :    :
    case constant n :
         statements;
    default:
         statements;
    }
```

Only integer and character constants are allowed.

The value of the expression is matched with the case constants sequentially. The case whose constant is first matched with the expression is executed and then all other cases (including default) below to matching case are executed sequentially.

→ However if you want to restrict the execution to only matching cases, 'break' statement can be used.

→ If no case is matched with the expression, then statements under default are executed.

→ The use of default is optional. It can be used anywhere within the switch block.

```
x =2;
switch(x)
    {
    case 1:
        printf(“A”);
        break;
    case 2:
        printf(“B”);
        break;
    case 3:
        printf(“C”);
        break;
    default:
        printf(“E”);
    }
```

Program :

1. Write a menu design program to input two numbers and make the following operation according to a choice from user

    1.ADD

    2.SUB

    3.MUL

    4.DIV

    ENTER A CHOICE (1,2,3,4)

2. Input an alphabet. test if it is a vowels or a consonant.

4. Iteration control statement / Loop control statement

This control statement is used to execute a set of statement multiple times. Number of times the statements are to be executed, based on a condition/ Boolean expression.

pre-tested loop Vs post-tested.

In a pre-tested loop, condition is checked before execution of the statements

In a post-tested loop, a condition is checked after the statements are executed. If condition is not satisfied loop terminates. If condition is satisfied, loop continues executing statements.

Type of loop:

- while loop
- do while loop
- for loop

while loop

It is pretested and conditional loop used to execute a set of statements until a condition is met. When the condition is not met, loop terminates.

syntax:

```
initialization;
while(condition/Booleanexpression)
     {
     - - - ;
     - - - ;
     increment/decrement;
     }
```

```
i = 1;
while (i<=10)
     {
     printf("\n%d", i);
     i = i + 1;
     }
```

do while Loop:

This is a post tested loop. Here the statement within the do while block is executed first time directly without checking the condition. Then the condition is checked. If the condition is true loop continues. If the condition is false, loop terminates.

Here the do while block is executed at least once.

```
Syntax:  initialization;
         do   {
              - - -;
              - - -;
              increment/decrement;
              }
         while(Booleanexpression);
```

```
     i = 1;
     do   {
          printf("\n%d", i);
          i = i + 1;
          }
     while (i<=4);
```

Program: Add all numbers from 1 to n

```
int main()
    {
    int i,sum=0,n;
    printf("Enter n"):
    scanf("%d",&n);
    i=1;
    do
        {
        sum = sum +i;
        i = i+1;
        }
    while(i<=n);
    printf("Sum = %d",sum);
    }
```

for Loop

This is a counter loop. It is used to execute a set of statements a certain number of time and until the condition is true.

Syntax:

```
for(initialization;condition;increment/decrement)
        {
        -    - ;
        -    - ;
        }
```

```
for ( i =100 ; i>=90; i--)
    {
    printf("%d",i);
    }
```

→ we can add multiple initialization/increment decrement statement in a for loop by using comma as a separator.

```
for(i=1,j=10 ;i<=5; i=i+1,j= j-2)
    {
    printf("\n%d %d", i,j);
    }
```

→ The initialization part and increment decrement are optional.

```
i =5;
for( ; i<=8;)
    {
    printf("%d",++i);
    }
```

→ we may ignore the condition part. But this leads to the infinite loop

```
i=1
for( ;   ; )
    {
    printf("%d", i)
    i = i+1;
    }
```

Other control statements:

break:

it transfers the control to outside of loop or outside of a block skipping remaining statement within that block.

```
     i  =1;
    while (i<=10)
        {
        if(i ==3)
            break;
        printf("%d",i);
        i++;
        }
    printf("CIME");
```

continue:

it transfers the control to beginning of the loop.

```
i  =0;
while (i<=10)
    {
    i = i+1;
    printf("\n%d",i);
    if(i %3 == 0)
        continue;
    printf("CIME");
    }
```

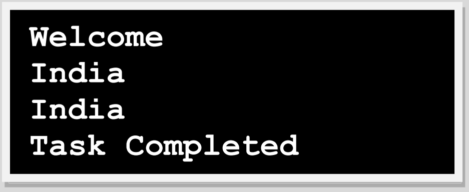
goto :

it transfers the control to a particular section of the program. This particular section is identified by keyword `label`.

Synatx: goto <label>;

goto L; // L is a label.

```
int main()
    {
        int i=1;
        printf("Welcome");
        Again:
        printf("\nIndia");
        if (i ==3)
            goto end;
        i = i+1;
        goto Again;
        end:
        printf("\nTask Completed");
        return 0;
    }
```



```
Welcome
India
India
Task Completed
```

Program Exercise

1. Display all numbers from 1 to n.
2. Add all the numbers from 1 to n and display the sum.
3. Add all even numbers and multiply all odd number from 1 to n.
4. Input a number. Find out its factorial.
5. Input a number. Add the digits.
6. Input a number. Count the digits.
7. Input a number. Reverse it.
8. Input a number Check if it is a palindrome or not.
9. Input a number. Check if it is a Armstrong or not.
10. Input a number. Check if it is a strong or not.
11. Input a number. Check if it is a nelson or not.
12. Input a number. Check if it is a prime or not.
13. Input a number. Check if it is a perfect number or not.
14. Generate a Fibonacci sequence of n numbers.
15. Input a number display following format

    Ex: number=123 output

    1

    2

    3

16. Input a range and find out square, cube and display in following format.
    NUMBER SQUARE CUBE
17. Print multiplication table of numbers from 2 to 12.
17. Read an integer number and find out the sum of all the digits till it reduces to a single digit.
    Ex- n=1256
    Sum=1+2+5+6=14
    Sum=1+4=5

18. Find out the sum of following series
    (i) Sum=$1^2 + 2^2 + 3^2 + \ldots + n^2$
    (ii) Sum=1- 1/1! +2/2! -3/3!..n/n!
    (iii) Sum =$x + x^2/2! + X^4/4! + X^6/6.X^n/n!$
    (iv) Sum=$x - x^3/3! + X^5/5! - X^7/7! +. X^n/n!$
    (v) Sum=$1^2 + 2^2 + 3^2 + \ldots + n^2$

19. Write a program to print all prime numbers from 1 to n.

Pattern/pyramid

```
1
12
123
1234
12345
```

```
1
21
321
4321
54321
```

```
5
45
345
2345
12345
```

```
12345
1234
123
12
1
```

```
54321
5432
543
54
5
```

```
*
**
***
****
*****
```

```
1
00
111
0000
11111
```

```
1
10
101
1010
10101
```

```
1
2 3
4 5 6
7 8 9 10
```

```
1
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8 9
```

```
    1
   12
  123
 1234
12345
```

```
    5
   45
  345
 2345
12345
```

```
12345
 2345
  345
   45
    5
```

```
12345
 1234
  123
   12
    1
```

```
        1
      1  1
    1  1  1
  1  1  1  1
1  1  1  1  1
```

```
56789
 4567
  345
   23
    1
```

```
    1
   121
  12321
 1234321
123454321
```

```
123454321
 2345432
  34543
   454
    5
```

```
123454321
 1234321
  12321
   121
    5
```

```
123454321
1234  4321
123      321
12          21
1            1
```

```
1            1
12          21
123      321
1234  4321
123454321
```

```
1
12
123
1234
12345
1234
123
12
1
```

```
12345
 1234
  123
   12
    1
   12
  123
 1234
12345
```

```
    1
   121
  12321
 1234321
123454321
 1234321
  12321
   121
    1
```

```
    A
   ABC
  ABCBA
 ABCDCBA
ABCDEDCBA
```

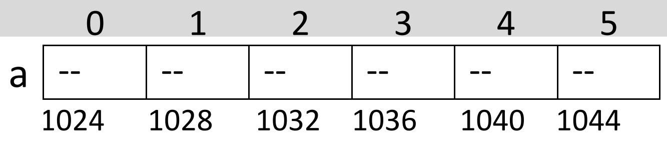
Pascal triangle
```
        1
      1  1
    1  2  1
  1  3  3  1
1  4  6  4  1
```

Array

Array is a data structure that contains multiple numbers of homogenous data elements in continuous memory location.

Declaration :   Datatype ArrayName[Size];

```
    int a[6];
```



Here an array ‘a’ is declared, that can contain maximum of 6 integer type data elements. Here size must be constant expression.

```
    #define size 6
    int a[size];
```

Learn static memory allocation

Direct initialization of array:

Programmer can initialize the array directly by putting the values inside the pair of curly braces. In this case, if array size is not specified, it is determined by the number of elements supplied.

```
    int a[ ] = {10,20,30,40};
```

Accessing element of an array

Array supports index access. This means, each element of an array are accessed and manipulated by their index. Index of an array elements starts with 0. `arrayname[index];` or `index[arrayname];` can be used to access an element.

```
    a[0] = 244 ;
    1[a] = 10;
    printf(“ %d”, a[1]) ;
```

Notes: Array name is a constant pointer that point to the base address of array.

Base address:

Address of first element of an array is called its base address. It is accessed by either using the array name or specifying the address of first element.

```
    printf(“%p”, a);          // Output : 1024
    printf(“%p”, &a[0]);      // Output : 1024
```

Program: Write a program to input n number of data in an array and find the sum of all the data.

```
    #include<stdio.h>
    int main()   {
        int a[20],n,i,sum;
        printf("Enter number of of elements:");
        scanf("%d",&n);
        //Input n data in array
        for(i = 0;i<n;i++)
            {
                printf("Enter the data:");
                scanf("%d",&a[i]);
            }
        //Calculating sum
        sum = 0;
        for(i = 0;i < n; i++)
            sum = sum +a[i];
        // show result
        printf("\nSum = %d",sum);
        return 0;
        }
```

Programs:

Day-1

1. Input n numbers in an array and display all the numbers.
2. Input n numbers in an array and display all the numbers in reverse order.
3. Input n numbers in an array. Add all even numbers and multiply all odd numbers.
4. Input an array of n numbers. Copy it into another array.
5. Input an array of n numbers. Copy it into another array in reverse order.
6. Input an array of n numbers. Reverse itself, without using another array
7. Input n number of data in an array. Store –ve numbers in one array and +ve numbers in another array.
8. Input an array of n elements. Print the biggest element.
9. Input an array of n elements. Print the smallest element.

Day-2

10. Input an array of n numbers. Input a key. Search the key in the array and print its index.
11. Input n number in an array. Input a position. Delete the number present in the position.
12. Input n number in an array. Input a number. Delete the number from array.
13. Input n number in an array. Insert a number in particular position.
    (Hints: Input n, an array, a number to insert, a position)
14. Input an array of n numbers. Check the array is sorted or not.
15. Input n numbers in an array. Sort the array in ascending order.

16. Input n numbers in an array. Sort the array in descending order.
17. Input n number of data in an array. Delete the duplicate data.
18. Input n number in an array. Find out frequency of each number in the array.
19. Take a sorted array. Insert an element according its precedence.
20. Input two arrays of length m and n (Each array contains unique elements). Find the union of two arrays.
21. Input two arrays of length m and n (Each array contains unique elements). Find the intersection of two arrays.
22. Input two arrays of length m and n (Each array contains unique elements). Find the set difference of two arrays.
23. Input two arrays for two vectors. Compute the dot product.
24. Input an array of n elements. Print the n times where each time the array is printed , it circularly shift left as following

    InputArray : 4 6 57

    Output: 6 5 7 4   5 7 4 6   7 4 6 5   4 6 5 7
25. Sort an array using bubble sort.
26. Sort an array using selection sort.
27. Sort an array using insertion sort.

2-Dimesional Array

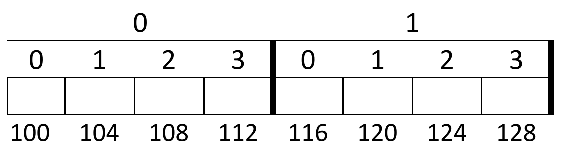
An array of arrays is known as 2D array. We can say, when an array contains data in rows and columns, it is called as 2D array. Like array, index of rows and column starts with 0.

Declaration of 2-D Array

datatype arrayname[rowsize][columnsize];

```
int ar[2][3];
```

A 2D array is declared to store the data in 3 rows and 4 columns.



[Physical representation]

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

[Logical representation]

Direct initialization:

int ar[][3] = { { 10,20,30},   { 40,50 ,60} };

Specifying row size is optional. If not specified, row size is determined by number of 1D array in give. Column size must be specified.

Accessing elements in a 2D array:

Like 1D array, a 2D array supports index accessing. To access an element for a 2D array, its roe index and column index must be specified.

Syntax: arrayname[rowindex][columnindex];

`ar[1][2]` is the 3rd element in the second array.

Program: Write a program to input data into M×N Matrix and display all data in matrix format.

```
#include<stdio.h>
int main()   {
    int a[10][20],m,n,i,j;
    printf("Enter row size and column size of matrix:");
    scanf("%d%d",&m,&n);
    for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
            {
            printf("Enter data :");
            scanf("%d",&a[i][j]);
            }
        }

    for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
             printf("%4d",a[i][j])   ;
        printf("\n");
        }
    return 0;
    }
```

Learn yourself:

What is 3-dimensional array? How to declare it? Explain with a program.

Programs:

1. Input data into a M×N matrix. Show the sum of the elements row wise.
2. Input data into a M×N matrix. Print column wise sum.
3. Input data into a N×N matrix. Display the lower triangle.
4. Input data into a N×N matrix. Display the upper triangle.
5. Input data into a N×N matrix. Display the elements in diagonal.
6. Find the trace of a N×N matrix.
7. Input data into two M×N matrix. Add them and store the result in another matrix. Print the resulting matrix.
8. Input data into two N×N matrix. Multiply them and store the result in another matrix. Print the resulting matrix.
9. Input data into two M×N matrix. Row-wise sort the elements.
10. Input an N×N matrix and an N-element vector. implement Matrix-vector multiplication.