Chapter 1

OVERVIEW of C

Importance of C:

C is a very important language because of its many useful features and many desirable qualities.

- Robust language whose rich set of built in functions and operators can be used to write any complex program.
- Allows software developers to develop software without worrying about the hardware platforms where they will be implemented.
- Compiler combines the capabilities of an assembly language with the features of a higher level language
- Well suited for both system software and business packages
- Efficient and fast.
- Highly portable.
- Well suited for structured programming.
- Machine independent language.
- Has the ability to extend itself.

Basic structure of a C program

Documentation Section →	/* Finding area and circumference of a circle */
Link Section →	#include<stdio.h>
Definition Section →	#define PI 3.1428
Global declaration Section →	float circum;
Main function Section →	Main()
{	{
Local declaration Section →	float area,r;
Executable Section →	r = 4.35;
	area= PI * r * r
	cir(r);
}	printf("Area of a circle = %d",area);
	}
User Defined functions →	cir(float r)
	{
	circum= 2* PI*r;
	printf(" Circumference = %f",circum);
	}

- **Documentation section:** -
  - Consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.
- **Link section:**
  - Provides instructions to the compiler to link functions from the system library
  - Eg. using #include directive.
- **Definition section:**
  - Defines all symbolic constants such using the #define directive.
- **Global declaration section:**
  - Some variables are to be used in more than one function called 'global variables'
  - Declared in the global declaration section that is outside of all the functions.
  - Also declares all the user-defined functions.

- **main () function section:**
  - Every C program must have one main function section.
  - Declaration part and executable part

    1. **Declaration part:** declares all the variables used in the executable part.
    2. **Executable part:** There is at least one statement in the executable part.

  - These two parts must appear between the opening and closing braces.
  - Program execution begins at the opening brace and ends at the closing brace.
  - Closing brace of the main function is the logical end of the program.
  - All statements in the declaration and executable part end with a semicolon.
- **Subprogram section:**
  - Contains all the user-defined functions that are called in the main () function.
  - Generally placed immediately after the main () function,
  - They may appear in any order

All section, except the main () function section may be absent when they are not required.

Note :

- C makes distinction between uppercase and lowercase letters
- Everything in C is written in lowercase letters
- \n → newline character, instructs to go to next line on the screen

CONSTANTS, VARIABLES AND DATA TYPES

Character Set:

- The various categories of characters are called the character set.
- Grouped into the following categories.'

1. **Letters:**
   - Uppercase: A.....Z
   - Lowercase a.....z
2. **Digits:**
   - All decimal digits. 0.....9
3. **Special characters:**
   - , - comma , . period, ; semicolon, : colon, etc
4. **White spaces:**
   - blank spaces, horizontal tab, new line etc.

C tokens:

- Are the basic buildings blocks in C language which are constructed together to write a C program.
- Each and every smallest individual unit in a C program is known as C tokens.
- Tokens are of six types. They are,

1. Keywords (eg: int, while),
2. Identifiers (eg: main, total),
3. Constants (eg: 10, 20),
4. Strings (eg: "total", "hello"),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*)

Keywords:

- Have fixed meanings and these meanings cannot be changed.
- There are 32 keywords. Some compiler may use additional keywords that must be identified from the C manual.
- Serve as basic building block for a program statement
- Must be written in lowercase letters

Identifiers:

- The names of variables, functions and arrays are **identifiers**.
- These are user-defined names and consist of a sequence of letters and digits. Such as, my_num, _ton etc.

Rules for identifier:

1. Must consist of only letters, digits and underscores.
2. First character must be an alphabet or underscore.
3. Only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white spaces.

Constants:

- Constants are of fixed values
- Do not change during the execution of a program.
- There are various types of constants.

Integer constants:

- An integer constant refers to a sequence of digits. T
- here are three types of integer constants namely : decimal integer, octal integer and hexadecimal integer.

Decimal integer :

- consists of a set of digits from 0 to 9, preceded by an optional + or – sign.
- Examples, 123 -321 0 64932

Octal integer :

- consists of a set of digits from 0 to 7, with a leading 0.
- Examples, 037 0 0437 0551

Hexadecimal integers:

- **are a** sequence of digits preceded by 0x or 0X.
- They may also includes letters from A to F or from a to f.
- The letters represents the numbers from 10 to 15.
- Examples, 0X2 0x9F 0Xbcd 0x

Real constants:

- Real constants are used to represent quantities that are very continuously, such as distances, temperature etc.
- These quantities are represented by numbers containing fractional parts.
- Examples, 0.00832 -0.75 33.337

Single character constants:

- A single character constants contains a single character enclosed within a pair of single quote
- Has an equivalent integer value
- Example, '5' 'X' ';'

String constants:

- Consists of a sequence of characters enclosed in double quotes.
- May consist of any combination of digits, letters, escaped sequences and spaces.
- Note that a character constant 'A' and the corresponding single character string constant "A" are not equivalent.
- Do not have an equivalent integer value
- **Valid S**tring Constants**: -** "W" , "100","24, Kaja Street"
- **Invalid S**tring Constants**: -** "W the closing double quotes missing
  Raja" the beginning double quotes missing

Backslash character constants:

- C supports some special backslash character constants that are used in output functions.
- For example:

Constants	**Meaning**
'\a'	audible alert (bell)
'\n'	new line
'\?'	question mark

Variables

- a variable is a container (storage area) to hold data.
- each variable should be given a unique name (identifier).
- Variable names are just the symbolic representation of a memory location.
- Value of a variable can be changed, hence the name 'variable'
- For example: int playerScore = 95;

Rules for naming a variable in C

1. A variable name can have letters, digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore.
3. There is no rule on how long a variable can be. However, only the first 31 characters of a variable are checked by the compiler.
4. Both uppercase and lowercase letters can be used.
5. Spaces should not be used in a variable name.
6. Keywords like *int, float, struct, if, while* cannot be used as variable names.

Data Types:

- C is rich in data types
- Supports 3 classes of dats types
    - Primary data types
    - Derived data types
    - User Defined data types

Primary Data Types:

- **C supports 5 data types : integer, float ,double, character and void**

Integer Data Types:

- Integers are whole numbers with a range of values
- Values are machine dependent.
- Occupies 2 bytes memory space
- 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.
- C has three classes of integer storage namely short int, int and long int to control the range of numbers and storage space.
- Qualifiers:
    - All three data types have signed and unsigned forms.
    - A short int requires half the amount of storage than normal integer.
    - Signed integer, unsigned integers are always positive and use all the bits for the magnitude of the number
    - Range of an unsigned integer will be from 0 to 65535.
    - The long integers occupies 4 bytes of storage space to store huge values.

Syntax:

int <variable name>;

int num1;
short int num2;
long int num3;

Example: 5, 6, 100, 2500.

Floating Point Data Types:

- The float data type is used to store fractional numbers (real numbers) with 6 digits of precision.
- Denoted by the keyword float.
- Double is same as float but with longer precision of 14 digits. Takes 8 bytes for storage.
- Qualifiers:
    - long double which occupies 10 bytes of memory space with higher precision

Syntax:

float <variable name>;

float num1;
double num2;
long double num3;

Example: 9.125, 3.1254.

Character Data Type:

- Character type variable can hold a single character
- Declared by using the keyword char.
- Supports signed and unsigned chars; both occupy 1 byte each,
- Unsigned characters have values between 0 and 255,
- Signed characters have values from –128 to 127.
- ***Syntax:***

  char <variable name>;

  char ch = 'a';

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

Void Type:

- The void type has no values therefore
- Cannot declare it as variable as we did in case of integer and float.
- Usually used with function to specify its type.

Declaration of Variables

- Tells the compiler what the variable name is
- Specifies what type of data the variable will hold
- Syntax: <data type> v1,v2,….,vn;
  - <data type> can be int, float, char, double
  - V1,v2… are variable names
- Ex: int a,b;
- If qualifiers like short, long, unsigned are used without basic data type , compiler treats the data type as int
  - Ex: short x;

Assigning values to variables

- Uses assignment operator : '='
- Syntax: <var name> = <value>
  - Ex: x=10;
- Value on RHS of = operator is assigned to LHS variable
- Can have multiple statements in one line
  - x=10; y=20;
- Can assign value to a variable during declaration statement
  - Syntax: <data type var_name = <value;
    - int a=10,b=3;
  - called as 'initialization'
- initialization can be done for more than one variables in one statement
  - x=y=z=0;

Declaring a Variable as Constant

- variable will be constant during the execution of the program
- Syntax: const <data type> <variable> = <value>;

        - const int x = 10;
    - x cannot be modified by a program
- can be used on RHS of an assignment statement like other variables

Overflow and Underflow of Data

- value a variable can be either too big or too small
- largest and smallest value a variable can hold depends on the machine
- overflow: if a value that is larger than the value a variable can hold is tried to store
    - int x = 36457;
- underflow : if a value that is smaller than the value a variable can hold is tried to store
    - int x= -35728
    - the range of integer variable is -32768 to 32767

Operators and Expression

- C language supports a rich set of built-in operators.
- An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations.
- Used in program to manipulate data and variables.
- C operators can be classified into following types,

    - Arithmetic operators
    - Relation operators
    - Logical operators
    - Bitwise operators
    - Assignment operators
    - Conditional operators
    - Special operators

Arithmetic operators

- C supports all the basic arithmetic operators.
- Are binary operators: each operator takes two operands
- The following table shows all the basic arithmetic operators.

Operator	Description
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denominator
%	remainder of division, applied for only integer data type

Integer Arithmetic:

- Both the operands are of integer type
- Called as integer expression
- Operation is called integer arithmetic
- Always yields an integer value
- / : integer division truncates the fractional part
    - 7 / 3 = 2
- % : sign of the result is always of the first operand

- Ex: -6 % 2 = -3, 6 % -2 = 3

Real Arithmetic

- Operands are of real type
- may assume the values in decimal or in exponential form
- floating point values are rounded to the number of significant digits permitted
- % cannot be used
- Ex: 6/7 = 0.857143, 1/3 = 0.333333

Mixed mode arithmetic

- One Operand is integer and the other is of float type
- Expression is called mixed mode expression
- Result is always float type
- Ex: 3/2 = 1.5

Relation operators

- The following table shows all relation operators supported by C.
- Are binary operators: each operator takes two operands

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal t

- Return true if the condition is true else returns false
- Relational expression
  - Expression containing relational operator
  - Syntax: <expression> <relational operator> <expression>
    - Expressions will be evaluated first and then their results are compared
  - Ex: a < b , (a+b) > (4+6)
- Used in decision statements

Logical operators

- C language supports following 3 logical operators.
- Used to test more than one conditions and make decisions
- Return true or false depending the condition
- Are binary operators: each operator takes two operands
- Suppose a=1 and b=0,

Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Bitwise operators

- Bitwise operators perform manipulations of data at bit level.
- Are binary operators: each operator takes two operands
- Also perform shifting of bits from right to left.
- Not applied to float or double data type.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

- truth table for bitwise `&`, `|` and `^`

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- The bitwise shift operators shifts the bit value.
- The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted.
- Both operands have the same precedence.

Example :

a = 0001000, b= 2
a << b = 0100000
a >> b = 0000010

Assignment Operators

Assignment operators supported by C language are as follows.

Operator	Description	Example
=	assigns values from right side operands to left side operand	a=b
+=	adds right operand to the left operand and assign the result to left	a+=b is same as a=a+b
-=	subtracts right operand from the left operand and assign the result to left operand	a-=b is same as a=a-b
*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

Increment and Decrement operators

++ Increment operator increases integer value by one

-- Decrement operator decreases integer value by one

- Unary operators
- Postfix (++/--) : expression is evaluated first using the original value and then the variable is incremented/ decremented by one
- Prefix (++/--) : variable is incremented/ decremented by one and then the expression is evaluated
- Eg: m=5
  Y=m++;
  Y contains value 5 and m has 6

  Y=++m
  Y and m has value 6

Conditional operator

- It is also known as ternary operator
- Used to evaluate conditional expression.
- Syntax : epr1 ? expr2 : expr3

  If **epr1** Condition is true ? Then value **expr2** : Otherwise value **expr3**

- Ex: y = (a > b) ? a : b;

Special operator

Operator	**Description**	**Example**
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**

Type conversions in expressions: C permits mixing of constants and variables of different types in an expression, but during evaluation it adheres to very strict rules of type conversion. If the operands are different types, the 'lower type' is automatically converted to the 'higher type' before the operation proceeds. The result is of the higher type. A typical conversion process is illustrated

```
int i, x;
float f;
double d;
long int l;
x = l / i + i * f - d
```

<u>Given below is the sequence of rules that are applied while evaluating expressions.</u>

All short and char are automatically converted to int; then

1. If one of the operands is long double, the other will be converted to long double and the result will be long double.
2. else, if one of the operand is double, the other will be converted to double and the result will be double.

3. else, if one of the operand is float, the other will converted to float and the result will be float.
4. else, if one of the operands is unsigned long int, the other will be converted to unsigned long int and the result will be unsigned long int.
5. else, if one of the operands is long int and the other is unsigned int, then
   (a) if unsigned int can be converted to long int, the unsigned int operand will be converted as such and the result will be long int.
   (b) else, both operands will be converted to unsigned long int and the result will be unsigned long int.
6. else if one of the operand is long int, the other will be converted to long int and the result will be long int.
7. else if one of the operands is unsigned int, the other will be converted to unsigned int and the result will be unsigned int.
   The final result of an expression is converted to the type of the variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.
   1. float to int causes truncation of the fractional part
   2. double to float causes rounding of digits.
   3. long int to int causes dropping of the excess higher order bits.

Casting a value: There are instances when we want to force a type conversion in a way that is different from the automatic conversion. Consider, for example, the calculation of ratio of females to males in a town.

Eg. ratio = female_no./ male_no.

Since female_no. and male_no. are declared as integers, in the program, the decimal part of the result of the division would be lost and ratio would represent a wrong figure. This problem can be solved by converting locally one of the variables to the floating point as shown below.

Ratio = (float) female_no./ male_no.

The operator (float) converts the female_no to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed in floating point mode, thus retaining the fractional part of result.

The type of female_no. remains as int in the other parts of the program.

The process of such a local conversion is known as casting a value. The general form of a case is (type-name) expression.

While type-name is one of the standard C data types. The expression may be a constant, variable or an expression.

Uses of Casts

Example	Action
x= (int)7.5	7.5 is converted to integer by truncation
a = (int)21.3/(int)4.5	Evaluated as 21/4 and the result would be 5
b = (double)sum/n	Division is done in floating point mode
y = (int)(a+b)	The result of a+b is converted to integer
z = (int)a+b	a is converted to integer and then added to b
p =cos((double)x)	Converts x to double before using it.

Operator precedence & associativity: Each operator in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is

evaluated. There are distinct levels of precedence and an operator may belong to one of the levels. The operators at the higher level of precedence evaluated first. The operators of the same precedence are evaluated either from left to right or from right to left depending on the level. This is known as the associativity property of an operator.

Summary of C operators

Operator	Description	Associativity	Rank
( )	Function call	→ L	1
[ ]	Array element reference	R	
+	Unary plus	→ R	2
-	Unary minus	L	
+ +	Increment		
- -	Decrement		
!	Logical negation		
~	Ones complement		
	Pointer reference		
&	Address		
Size of	Size of an object		
(type)	Type cast (conversion)		
*	Multiplication	→ L	3
/	Division	R	
%	Modulus		
+	Addition	→ L	4
-	Subtraction	R	
< <	Left shift	→ L	5
> >	Right shift	R	
<	Less than	→ L	6
< =	Less than or equal to	R	
>	Greater than		
> =	Greater than or equal to		
= =	Equality	→ L	7
! =	Inequality	R	
&	Bit wise AND	→ L	8
		R	
^	Bit wise XOR	→ L	9
		R	
	Bit wise OR	→ L	10
		R	
&&	Logical AND	→ L	11
		R	
	Logical OR	→ L	12
		R	
?:	Conditional expression	→ R	13
		L	
=	Assignment operators	→ R	14
* = /= %=		L	

+= -= &= ^= = <<= >>=			
	Comma operator	⟶ L R	15

Mathematical Functions: Mathematical functions such as cos, sqrt, log etc. are frequently used in programs. Most of the c compilers support these basic math functions.

Math Functions

Trigonometric	Meaning
acos(x)	arc cosine of x
asin(x)	arc sine of x
atan(x)	arc tangent of x
atan2(x,y)	arc tangent of x/y
cos(x)	cosine of x
sin(x)	sine of x
tan(x)	tangent of x
Hyperbolic cosh(x)	hyperbolic cosine of x
sinh(x)	hyperbolic sine of x
tanh(x)	Hyperbolic tangent of x
Other Functions ceil(x)	x rounded up to the nearest integer
exp(x)	e to the power x ex
fabs(x)	absolute value of x
floor(x)	x rounded down to the nearest integer
fmod(x,y)	Remainder of x/y
log(x)	natural log of x , x > 0
log10(x)	base 10 log of x , x > 0
pow(x,y)	x to the power of y (xy)
sqrt(x)	square root of x , x>=0

Note: 1. x and y should be declared as double

2. In trigonometric and hyperbolic functions, x and y are in radians.
3. All the functions return a double.

Question: Identify syntax errors in the following program. After correcting, what output would you expect when you execute it.

```
#include<stdio.h>

#include<conio.h>

#define PI 3.14159

void main()

{

int R,C;

float perimeter;
```

```
float area;

C=PI;

R=5;

perimeter=2.0*C*R;

Area = C*R*R;

printf("%f", "%d",&perimeter,&area)

}
```

List of simple programs:

1. Write a C program to print your name on the screen.
2. Write a C program to print your address in different lines.
3. Write a C program to find addition of two numbers. (subtraction, division, multiplication)
4. Write a C program to find simple interest, given principle, rate and time.
5. Write a C program to find area of a triangle given base and height.
6. Given the radius of a circle, write a program to compute and display its area. Use a symbolic constant to define the π value and assume a suitable value for radius.
7. Given the values of three variables a, b and c, write a program to compute and display the values of x, where X= a / (b - c)
8. Relationship between Celsius and Fahrenheit is governed by the formula : F = (9C/5)+32
   Write a program to convert the temperature
   (a) from Celsius to Fahrenheit and
   (b) from Fahrenheit to Celsius.
9. Area of a triangle is given by the formula: A=sqrt(S(S-a)(S-b)(S-c))
   Where a, b and c are sides of the triangle and 2S=a+b+c. Write a program to compute the area of the triangle given the values of a, b and c.
10. Distance between two points (x1,y1) and (x2,y2) is governed by the formula
    $D_2 = (x_2-x_1)_2+(y_2-y_1)_2$
    Write a program to compute D given the coordinates of the points.
10. A point on the circumference of a circle whose center is (0, 0) is (4, 5). Write a program to compute perimeter and area of the circle.
11. The line joining the points (2,2) and (5,6) which lie on the circumference of a circle is the diameter of the circle. Write a program to compute the area of the circle.
12. Write a program to display the equation of a line in the form: ax+by=c
    for a=5, b=8 and c=18.
13. Write a C program to find area of a circle given its radius.
14. Write a C program to convert Fahrenheit degree to Celsius degree. (5/9*(f-32))
15. Write a C program to find convert rupees to paisa.
16. Write a C program to find simple interest, given principle, rate and time.
17. Write a C program to find area of a triangle given three sides..
18. Write a C program to implement Pythagoras theorem.
19. Write a C program to convert days to months.
20. Write a C program to convert a negative number to a positive number.
21. Write a C program to find square root of a number.
22. Write a C program to print last digit of a number.
23. Write a C program to find sum of first and second digit of a two digit number.
24. Write a C program to find distance given the time and velocity.
25. Write a C program to calculate salary given BASIC, DA, HRA and IT

Managing Input and Output Operation

- C support many input and output statements.
- It also supports 2 classes of input and output.
    - Unformatted : getchar( ), getch( ), putchar( ), putch ( )
    - Formatted : scanf( ) and printf ( )

Unformatted Input/Output statements

getchar( ):

- It reads one character from the standard input.
- Syntax : variable_name = getchar( )
- Example:
```
char c;
….
c=getchar();
```
- A dummy getchar( ) is used to eat unwanted characters given as input

putchar( ):

- It writes one character to the standard output (Monitor).
- Syntax : putchar(variable_name/ character constant)
-

Example:
```
char c;
c=”A”;
putchar(c);
```

Formatted Input:

Scanf( )

- Syntax : scanf (“control string”, argument list);
- Accepts the input from keyboard
- Can be used to enter any combination of numerical values, single character and strings. –
- returns the number of data items that have been entered successfully.

Syntax: scanf(control string, arg1, arg2,….argn)

- control string: consist of control characters, whitespace characters and nor-whitespace characters. The control characters are preceded by a % sign and are listed below.

Control Character	Explanation
%c	A single character
%d	A decimal integer
%i	An integer
%e, %f, %g	A floating-point number
%o	An octal number
%s	A string
%x	A hexadecimal number
%p	A pointer

%n	An integer equal to the number of characters read so far
%u	An unsigned integer
%[]	A set of characters
%%	A percent sign

Example:
```
int i;
float f;
char c;
char str[10];
scanf("%d %f %c %s",&I,&f,&c,str);
```

- format string: must be a text enclosed in double quotes. It contains the information for interpreting the entire data for connecting it into internal representation in memory. Example: integer (%d) , float (%f) , character (%c) or string (%s).
- argument list: contains a list of variables each preceded by the address list and separated by comma. The number of argument is not fixed;
- Inside the format string the number of argument should tally with the number of format specifier.
- Example: if i is an integer and j is a floating point number, to input these two numbers we may use
  scanf ("%d%f", &i, &j);
-
- To input integer numbers:
    - Format specification: %wd
    - % -> conversion character
    - w - > an integer number specifying the field width of the number to be read
    - Ex: scanf(" %2d %3d",&a,&b)
        - Input : 50 123
        - 50 assigned to a and 123 assigned to b
    - Unassigned input value will be assigned to next scanf call
    - Input data must be separated by spaces

- To input real numbers:
    - Format specification: %f
    - Field width for real numbers is not specified
    - Ex: scanf(" %f ",&a)
        - Input : 50.123
        - 50.123 assigned to b
- To input characters:
    - Format specification: %wc or %ws
    - % -> conversion character
    - w - > an integer number specifying the field width of the number to be read
    - %[characters] -> only the characters specified within the brackets are permissible in input data
    - %[^characters] -> the characters specified within the brackets are not permissible in input data

- Ex: scanf(" %[a-z]c %[^0-9]c",&a,&b)
  - a can take only one letter between a to z
  - b can take any character except digits 0-9

Formatted Output printf( ):

- Used to output data onto the screen.
- can be used to output any combination of numerical values, single characters and strings.

Syntax: printf("control string", arg1, arg2, ….argn)

- control string:
  - characters are listed above (same as input control string)
  - Characters that will be printed on the screen as they appear
  - Format specifications for output
  - Escape sequence characters
- Format specification
  - % w.p <type specifier>
- To output integer numbers
  - %wd
  - W- specifies the minimum field width for output
  - If number is greater than the width, prints the whole number
  - Number is printed right justified
  - Leading spaces appear as blanks
  - %-wd : prints the number left justified
  - %0wd : leading spaces are padded with 0's
  - %whd : short integers
  - %wld : long integers

- To output real numbers
  - %w.p f
  - w- specifies the minimum field width for output
  - p-number of digits to be displayed after the decimal point rounding of to p decimal places
  - If number is greater than the width, prints the whole number
  - Number is printed right justified
  - Leading spaces appear as blanks
  - %-wd : prints the number left justified
  - %0wd : leading spaces are padded with 0's
  - %whd : short integers
  - %wld : long integers

Example: printf("%d %o %x\n", 100,100,100);
Will print : 100 144 64

functions supported by C:

- "ctype.h" header file support all the below functions in C language

isalpha( )

- checks whether given character is alphabetic or not.
- Returns TRUE or FALSE accordingly

isspace()
- checks whether character is space or not.

isupper()
- checks whether character is an uppercase character or not.

islower()
- checks whether character is a lowercase character or not.

ispunct()
- checks whether character is a punctuation character or not.
- Punctuation characters are , . : ; ` @ # $ % ^ & * ( ) < > [ ] \ / { } ! | ~ - _ + ? = ' "

isprint()
- checks whether character is printable or not.

toupper()
- returns character in upper case.

tolower()
- returns character in lower case.

Decision making and Branching:

if statement

- a powerful decision making statement
- used to control the flow of execution of statements.
- basically a two-way decision statement
- used in conjunction with an expression.

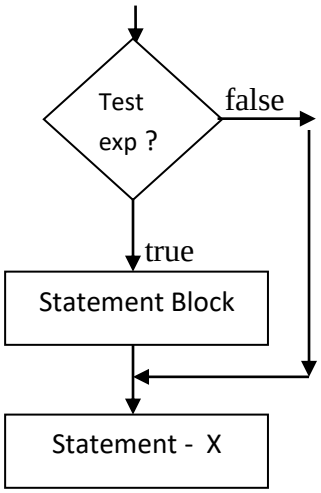
- The if statement may be implemented in different forms depending on the complexity of the conditions to be tested.

1. Simple if statement
2. if...else statement
3. Nested if...else statement
4. else if ladder.

Simple if statement

- syntax:

```
if (test
expression)
{
statement-block;
}
statement-x;
```



Example:

```
.....
if (x == 1)
{
y = y +10;
}
printf("%d", y);
.....
```

- The statement-block may be a single statement or a group of statements.
- If the test expression is true, the statement-block will be executed;
- otherwise the statement-block will be skipped and the execution will jump to statement-x.
- Remember, when the condition is true both the statement-block and the statement-x are executed in sequence.
- **Example**:
    - The program tests the value of x and accordingly calculates y and prints it.
    - If x is 1 then y gets incremented by 1 else it is not incremented
    - if x is not equal to 1. Then the value of y gets printed

      .

The if…else statement

- an extension of the simple if statement.
-
- Syntax

```
if (test expression)
{
True-block statement(s)
}
else
{
False-block statement(s)
}
statement-x
```

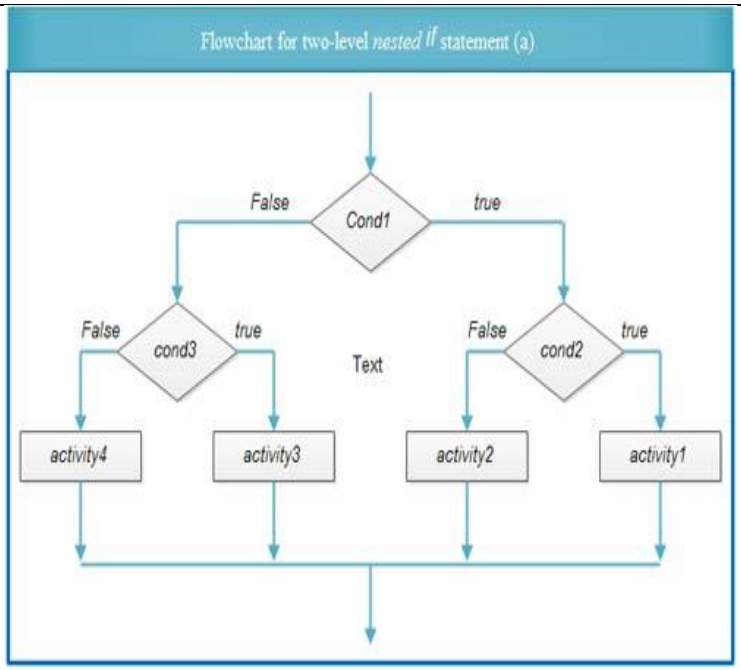
- If the test expression is true , then the true-block statement(s), immediately following the if statement are executed;
- otherwise the false-block statement(s) are executed.
- In either case, either true-block or false-block will be executed, not both.
- **Example:**

```
.....
if (c == 'm')
male = male +1;
else
fem = fem +1;
.....
```

Nesting of if…else statements

- When a series of decisions are involved, we may have to use more than one if.....else statement in nested form as follows:

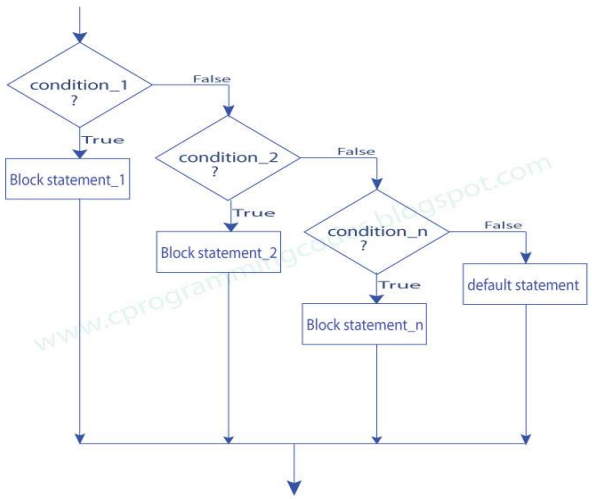
```
if (test condition1)
{
    if (test condition 2)
    {
    statement-1;
    }
    else
    {
    statement-2;
    }
}
Else
{
    if (test condition 3)
    {
    statement-1;
    }
    else
    {
    statement-2;
    }
}
statement-x;
```



- If the condition-1 is false statement-3 will be executed;
- otherwise it continues to perform the second test.
- If the condition-2 is true, the statement-1 will be evaluated;
- otherwise the statement-2 will be evaluated and then the control is transferred to the statement-x.

The else if ladder

- There is another way of putting ifs together when multipath decisions are involved.
- A multipath decision is a chain of ifs in which the statement associated with each else is an if.
- It takes the following general form:

if (condition 1)
  statement 1 ;
else if (condition 2)
  statement 2;
else if (condition 3)
  statement 3;
else if (condition n)
  statement n;
else
  default statement;
  statement x;
	

-
- This construct is known as the else if ladder.
- The conditions are evaluated from the top (of the ladder), downwards.
- As soon as a true condition is found, the statement associated with is executed and the control is transferred to the statement x (skipping the rest of the ladder).
- When all the n conditions become false, then the final else containing the default statement will be executed.

The Switch statement: C has a built-in multi-way decision statement known as a switch. The switch statement 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 that case is executed. The expression is an integer expression or characters.Value-1, value-2….are constants or constant expressions and are known as case labels. Each of these values should be unique within a switch statement. Block-1 , block-2…are statement lists and may contain zero or more statements. There is no need to put braces around these blocks. Case labels end with a colon (:).
When the switch is executed, the value of the expression is successively compared against the values value-1, value-2,…If a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.

```
  switch (expression)
{
    case value-1;
                block1
                break;
……
……
default:
       default-block
       break;
}
statement-X;
```

The break statement at the end of each block signals is the end of a particular case and causes an exit from the switch statement, transferring the control to the statement-x following the switch.

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. If not present, no action takes place if all natches fail and the control goes to the statement X.

What is difference between switch and if-else?

- Simply saying, IF-ELSE can have values based on constraints where SWITCH can have values based on user choice.
- The main difference between switch - case and if - else is we can't compare variables. In the if - else, first the condition is verified, then it comes to else whereas in the switch - case first it checks the cases and then it switches to that particular case.
- The switch branches on one value only, whereas the if-else tests multiple logical expressions.
  So you could say that the switch is a subset of if-else.
  The potential difference if that switch is conceptually an N-way branch point, whereas the if-else is always a (repeated) binary branch.

The conditional operator: The conditional operator is useful for making two-way decisions. This operator is a combination of ? and : and takes three operands. The general form of use of the conditional operator is as follows.

conditional expression ? expression1: expression2

The conditional expression is evaluated first. If the result is nonzero, expression1 is evaluated and is returned as the value of the conditional expression. Otherwise, expression2 is evaluated and its value is returned. For example
the segment

```
if ( x < 0)
   flag = 0;
else
   flag =1;
```

can be written as flag = (x < 0)? 0 :1;

Example2.

Y = 1.5x + 3 for x ≤ 2
Y = 2x +5 for x >2
Y = (x >2) ? (2*x +5): (1.5*x+3);
The conditional operator may be nested for evaluating more complex assignment decisions.
4x +100 for x < 40
Example: salary = 300 for x =40
4.5x + 150 for x > 40
Salary = (x!=40)? ((x<40)? (4*x+100): (4.5*x+150):300;
The same can be evaluated using if … else statement as follows

```
if (x < = 40)
if ( x< 40)
salary = 4*x+100;
else
  salary = 300;
else
  salary = 4.5*x + 150;
```

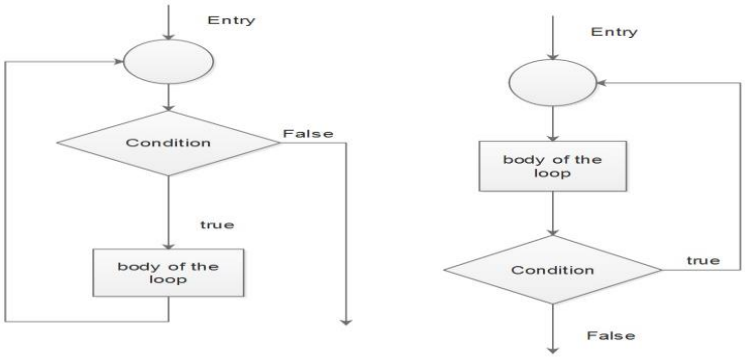
List of simple programs:

1. Write a C program to check whether a given number is even or odd.
2. Write a C program to check whether a given number is positive or negative.
3. Write a C program to find biggest of two numbers.
4. Write a C program to check whether a given year is leap year or not.
5. Write a C program to find square root of a negative number.
6. Write a C program to find sum of first and second digit of a two digit number.
7. Write a C program to find distance given the time and velocity.
8. Write a C program to calculate salary given BASIC, DA, HRA and IT

Decision-Making and Looping

Looping:

- A sequence of statements are executed until some condition is satisfied which is placed for termination of the loop.
- consists of two segments;
    - body of the loop
    - control statement.
        - The control is tested always for execution of the body of the loop.
- Depending on the position of the control statement in the loop,
    - entry-controlled loop
        - first the conditions are tested and if satisfied then only body of loop is executed
    - exit-controlled loop.
        - the test is made at the end of the body,
        - body is executed unconditionally first time.



- A looping process, in general, would involve the following four steps:

    1. Setting and initialization of a counter.
    2. Execution of the statements in the loop.
    3. Test for a specified condition for execution of the loop.
    4. Incrementing / updating the counter.

The C language provides for three loop constructs for performing loop operations. They are:

1. The while statement.
2. The do statement.
3. The for statement.

The while Statement

- Simplest looping structures.
- Syntax:

```
while (test condition)
{
  body of the loop
}
```

- an entry-controlled loop statement.
- The test condition is evaluated
- only if the condition is true the body is executed.
- After execution of the body, the test-condition is once again evaluated
- if it is true, the body is executed once again.
- process of repeated execution of the body continues until the test-condition finally becomes false
- if false,the control is transferred out of the loop.
- On exit, the program continues with the statement immediately after the body of the loop.
- Example:

```
.....
.....
x = 1;                  ----------   Initialization
while (x < 10)          ----------   Test condition
{
 printf("%d",x);        ----------   body of the loop
 x = x+1;
}
.....
```

The do … while statement

- is an exit-controlled loop structure
- in some situations it might be necessary to execute the loop, even if the test-condition fails.
- Evaluates the body of the loop first
- Checks the condition at the end of the loop
- If true, continues the loop else exits
- Body of the loop is always executed at least once
- Sybtax:
-

```
do
{
  body of the loop
}
while (test condition);
```

- **Example:**

```
do
{
  ch = getchar();
  printf("The character entered is %c",ch);
}
while (ch != 'n')
```

- ch is assumed to be a variable of type character.
- Now the loop is executed first under no test
- input by the user is printed.
- Then the entered value would be tested by the condition for further advancement.

The FOR Statement:

- **The for loop is an entry-controlled loop**
- provides a more concise loop control structure.
- Syntax:

```
for (initialization; test-condition; increment)
{
 body of the loop
}
```

The execution of the for statement is as follows:

1. Initialization
    1. control variables is done first
    2. Uses assignment statements such as **i = 1** and **count = 0**.
    3. The variables i and count are known as the loop control variables.

2. test-condition :
    1. The value of the control variable is tested.
    2. is a relational expression, such as **i<10**
    3. determines when the loop will exit.
    4. If the condition is true, the body of the loop is executed;
    5. otherwise the loop is terminated the execution continues with the statement that immediately follows the loop.

3. Body of the loop :
    1. body of the loop is executed,
    2. the control is transferred back to the for statement
    3. the control variable is incremented using an assignment statement such as **i = i+1**
    4. the new value of the control variable is again tested
    5. if true, the body of the loop is again executed.
    6. This process continues till the value of the control variable fails to satisfy the test-condition.

- **Example:**

```
for( i = 0; i<10; i = i+1)
{
  printf("%d",i);
}
```

Additional Features of for Loop

1. More than one variable can be initialized at a time in the for statement.
   - For example:

```
for (p=1,n=0; n<18; ++n)
```

2. Like the initialization section, the increment section may also have more than one part.

   - For example,

```
for (a=2,b=30; n <= m; n=n+1,m=m-1)
{
p = b/a;
printf("%d ",p);
}
```

3. It is also permissible to use expressions in the assignment statements of initialization and increment sections.
   - For example

```
for (w = (a+b); w>0; w=w/2) is valid.
```

4. One or more sections in the for statement can be omitted, if necessary.
   - For example :

```
  e = 2;

for (; e! = 10 ;)
{
  printf("%d",e);
  e = e + 2;
}
```

5. Both the initialization and increment sections are omitted in the for statement.
   - the sections are left blank.
   - the semicolons separating the sections must remain.
   - If the test-condition is not present, the for statement sets up an infinite loop.

The break statement

- terminates the execution of the nearest enclosing **do**, **for**, **switch**, or **while** statement in which it appears.
- Control passes to the statement that follows the terminated statement.

- Within nested statements, the **break** statement terminates only the **do**, **for**, **switch**, or **while** statement that immediately encloses it.
- You can use a **return** or **goto** statement to transfer control elsewhere out of the nested structure.
- Exits only a single loop

The continue statement

- During loop operations it may be necessary to skip a part of the body of the loop under certain conditions.
- 'continue' causes the loop to be continued with the next iteration
- skips any statement in between
- passes control to the next iteration of the nearest enclosing **do**, **for**, or **while** statement in which it appears,
- The next iteration of the loop is determined as follows:
    - In a **do** or **while** loop, the next iteration starts by reevaluating the controlling expression of the **do** or **while** statement.
    - In a **for** loop (using the syntax **for** (*init-expr*; *cond-expr*; *loop-expr*)), **continue** causes *loop-expr* to be executed. Then *cond-expr* is reevaluated and, depending on the result, the loop either terminates or another iteration occurs.

Jumping out of the program: exit ( )

- Breaks out of the program
- Returns to operating system
- Takes an integer value as its aargument
- Syntax : exit (0)
- Needs the header file : < stdlib.h>

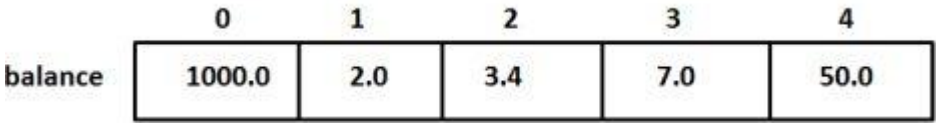
Arrays

- a fixed-size sequential collection of elements of the same type.
- used to store a collection of data of the same type.
- declaring individual variables, such as number0, number1, ..., and number99, is tedious
- one array variable can be used to store number0, number1, ..., and number99 as numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.
- array consists of contiguous memory locations.
- Hence, an array is a collection of data elements of same data type stored in contiguous memory locations

It is described by a single name and each element of an array is referenced by using array name and its index number.

One Dimension array

- List of items with one array name and only one subscript
-



Declaration of One Dimension Array

- Syntax:

  type arrayName [size];

- **Example,**

```
int Age[5] ;
float cost[30];
```

- `Reference to array element outside the limit leads to an error`
- Size should be always an integer constant or a symbolic integer constant

Initialization

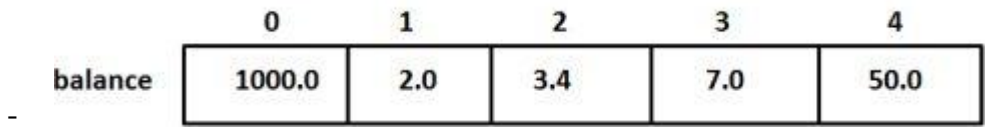
- Compile time
- Run time

Compile time

- An array can be initialized along with declaration.
- For array initialization it is required to place the elements separated by commas enclosed within braces.
- Example

```
int A[5] = {11,2,23,4,15};
```

- The number of values between braces { } cannot be larger than the number of elements that we declare
- It is possible to leave the array size open.
- an array just big enough to hold the initialization is created
- double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};



-

Run time initialization

```
int age [10], i ;
for (i=0 ; i<10; i++)
{
  scanf("%d", &age[i]);
}
```

-

Referring to Array Elements

In any point of a program in which an array is visible, we can access the value of any of its elements individually as if it was a normal variable, thus being able to both read and modify its value. The format is as simple as:
name[index]

Examples:

```
printf("%d", age[4]);      /* print an array element */
age[4] = 55;          /* assign value to an array element */
scanf("%d", &age[4]);       /* input element */
```

Using Loop to input an Array from user

```
int age [10], i ;
for (i=0 ; i<10; i++)
{
  scanf("%d", &age[i]);
}
```

Two dimensional arrays: are declared as follows
type array-name [row-size] [column size];
Initializing two dimensional arrays: Like the one-dimensional arrays, two dimensional array may be initialized by the following their declaration with a list of initial values enclosed in braces.
Eg. static int table [2] [3] = { 0,0,0,1,1,1};
Initializes the elements of the first row to zero and the second row to one. The initialization is done row by row. The above statement can be equivalently written as
static int table [2] [3] = {{0,0,0},{1,1,1}};
or
static int table [2] [3] = {
{ 0,0,0},
{ 1,1,1},

```
};
```

If the values are missing in an initializes, they are automatically set to zero. For example

```
static int table [2] [3] = {
        { 1, 1 },
        { 2}
};
```

will initialize the first two elements of the first row to one, the first element of the second row to two and all other elements to zero.

STRINGS

Handling of character strings: A string is an array of characters. Any group of characters defined between double quotation marks is a constant string.
The common operations performed on character strings are

Reading and writing strings
Combining strings together
Copying one string to another
Comparing strings for equality
Extracting a portion of a string

Declaring and initializing string variable: A string variable is any valid C variable name and is always declared as an array. The general form of declaration of a string variable is

char string-name [size];

The size determines the number of characters in the string name. Some examples are

char city [10];
char name [30];

when the compiler assigns a character string to a character array, it automatically supplies a null character ('\o') at the end of the string. Therefore the size should be equal to the maximum number of characters in the string plus one.
Character arrays may be initialized when they are declared. C permits a character array to be initialized in either of the following two forms.

static char city [9] = " NEW YORK";
static char city [9] = { 'N', 'e', 'w', ' ', 'y','o','r','k','\o'};

when initializing a character array by listing its elements null terminator must be explicitly specified.
C also permits us to initialize a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number of elements initialized.
Eg. static char string [ ] = { 'G', 'o','o', 'D','\o'};
defines the array string has a five element array
Reading strings from terminal.

Reading words: The familiar input function scanf can be used with %s format specification to read in a string of characters.
Eg. char address [15];
scanf ("%s", address);
The problem with the scanf function is that it terminates its input on the first while space it finds. When we need to read a line of text from the terminal. It is not possible to use scanf functions to read a line containing more than one word. This is because the scanf terminates reading as soon as a space is encountered in input. getchar( ) function can be used repeatedly to read successive single character from the terminal at time and place them into a character array. Thus, an entire line of text can be read and stored in an array.

Writing strings to screen: printf function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by the null character.

printf ("%s", name);

can be used to display the entire contents of the array name.

String handling functions:

strlen( ) function: This function counts and returns the number of characters in a string.
n = strlen(string);
where n is an integer variable which receives the value of the length of the string. The argument may be a string constant. The counting ends at the first null character.

strcat( ) function: The strcat function joins two strings together. It takes the following form
strcat (string1, string2);
string1 and string2 are character arrays. When the function strcat is executed. String2 is appended to string1. It does so by removing the null character at the end of string1 and placing string2 from these. The string at string 2 remains unchanged.

str cmp ( ) function: The strcmp function compares two strings identified by the arguments and has a value 0 if they are equal. If they are not, it has the numeric difference between the first non-matching character in the strings.
strcmp (string1, string2);
string1 and string2 may be string variables or string constants.
Eg. strcmp(" their", "there"); will return a value of –9 which is the numeric difference between ASCII "i" and ASCII "r" i.e., "i" minus "r" ASCII code is –9. If the value is negative, string1 is alphabetically above string2.
strcmp ("there", "their"): will return a value of 9. If the value is +ve , string 2 is alphabetically above string1.

strcpy( ) function: The strcpy function works almost like a string assignment operator. It takes the form.
strcpy(string1, string2);
Assigns the contents of string 2 to string 1. string2 may to a character array variable or a string constant.
Eg. strcat(city, " Delhi");
will assign the string "Delhi" to the string variable city.

strrev ( ) function: This function reverses the characters in a string.
strrev (string);
Eg. strrev("program") reverses the characters in string into "margorp".

strlwr( ) function: This function converts all characters in a string from upper case to lower case.
strlwr(string);

strupr( ) function: This function converts all characters in a string from lower case to upper case.
Strupr (string);

User Defined Functions

Function:

A function is a self contained set of statements that perform some specific task when it is called.

Any 'C' program contain at least one function i.e main(). There are basically two types of function:

1. Library function :
   - System defined function that can't be modified, it can only be read and can be used. These functions are supplied with every C compiler
   - Ex: sqrt(), sin(), scanf()
2. User defined function :
   - Defined by the user according to its requirement

- Function definition consists of the whole description and code of the function.
- It tells about what function is doing what are its input and output.
- It consists of two parts: function header and function body

Syntax:-

```
return type function(type 1 arg1, type2 arg2, type3 arg3, ….) /*function header*/
{
        local variable declaration;
        statement 1;
        statement 2;
        …..
        return value
}
```

Elements of user defined functions

Function header:

- Consists of three parts:
  - Function name: a valid C identifier name, appropriate to the task the function performs
  - Return type: the type of value the function returns, if not mentioned, compiler returns integer type
  - Formal parameter list:
    - Declares the variables that are sent by calling function
    - Serves as input to the function
    - Can also be used to send values to the calling function

Function body:

- Declaration:
  - Local variables used by the function are declared
- Statements
  - Function statements that perform the given task

- Return
    - Statement that returns the value after evaluation by the function

Function call

- Function call is made by using just the function name
- Ex:

```
main()
{       …
      Func(a,b);
        ….
}
```

- When he compiler encounters a function call, control is transferred to the function, gets executed
- When return statement is encountered in the function, control returns back to the calling function

Function declaration

- Functions must be declared before they are invoked.
- Syntax
    - <function type> <function name>(arguments list);
- Declaration should be done before calling main()
- Ex:

```
int func(int a, int b);
main() {….}
```

Category of Function based on argument and return type:

i) Function with no argument & no return value

- Function that have no argument and no return value is written as:-

void function(void);
main()
{
function()
}
	void function()
	{
	statement;
	}

ii) Function with no argument but return value

- The calling function calls the function with no arguments, but the called function returns value after computation

Syntax:-

int fun(void);
main()
{
int r;
r=fun();
}
	int fun()
	{
	reurn(exp);
	}

iii) function with argument but no return value

- Here, the calling function send data to the called function as arguments, but called function dosen't return value.

- Syntax:-

void fun (int,int);
main()
{
fun(a,b);
}
	void fun(int x, int y)
	{
	statement;
	}

iv) function with argument and return value

- Here the calling function has the argument to pass to the called function
- The called function returned value to the calling function.
- Syntax:-
-

fun(int,int);
main()
{
int r;
r=fun(a,b);
}
	int fun(int x,int y)
	{
	return(exp);
	}

Call by value and call by reference

- There are two ways through which we can pass the arguments to the function such as call by value and call by reference

1. Call by value

- copy of the actual argument is passed to the formal argument
- the operation is done on formal argument.
- it doesn't affect content of the actual argument.
- Changes made to formal argument are local to block of called function so when the control back to calling function the changes made is vanished.

2. Call by reference

- Instead of passing the value of variable, address or reference of a variable or array is passed
- the function operates through the address of the variable rather than value.

Passing arrays to function

One Dimension array:

- Sufficient to send the name of the array, without any sunscripts to the calling function
- Ex: largest (a,n)
    - Array -> a and limit -> n
- Name of the array represents the address of the first element of the array.
- The function uses the same array for its manipulation
- Changes made in the calling function for the array elements, reflects the array in the called function
- Even called as pass by reference
- I the called function, array has to be declared with the size
- Ex: int largest(int a[], int n)

Two dimensional array

- Similar to 1D array
- Call the function by sending only the array name
- In function definition, the 2D array dimension has to be mentioned by specifying the size
- Its enough to just mention the second dimension of the array
- Ex: int avg(int a[][10],int m, int n)

Structures

structure:

- user defined data type available in C that allows to combine data items of different kinds.
- **Syntax of structure**

```
struct structure_name
{
        data_type member1;
        data_type member2;
                .
                .
        data_type memeber;
};
```

  - Every data member should be ended by ;
  - After defining a structure , semicolon ; in the ending line is must.

- Example : create the structure for a person as mentioned above as:

```
struct person
{
        char name[50];
        int citNo;
        float salary;
};
```

  - creates the derived data type **struct person**.

Structure variable declaration

- When a structure is defined, it creates a user-defined type but, no storage or memory is allocated.
- Structure has to be declared ,
- For the above structure of a person, variable can be declared as:

```
struct person
{
        char name[50];
        int citNo;
        float salary;
};

int main()
{
   struct person person1, person2;
  return 0;
}
```

- Another way of creating a structure variable is:

```
struct person
{
        char name[50];
         int citNo;
        float salary;
} person1, person2, person3[20];
```

Initializing variables of type struct

- Structs variables are initialized in much the same way as arrays, by enclosing multiple data fields in curly braces.
- if the struct is only partially initialized, then remaining fields are initialized to zero:

struct person1 = { "John Doe", 4, 10000 }, person2 = { "Susie Jones", 10,20000 };

Accessing Structure Members

- the **member access operator (.) is used**.
- The member access operator is coded as a period between the structure variable name and the structure member that we wish to access.
- Ex: x = person.salary

POINTERS

What is a Pointer?

- Pointer in C language is a variable that stores/points the address of another variable.
- A Pointer in C is used to allocate memory dynamically i.e. at run time.
- The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc

Declaring a Pointer :

Syntax- Data type *pointer name;
- * before pointer indicate the compiler that variable declared as a pointer.

Example :

- int *p1; //pointer to integer type
- float *p2; //pointer to float type
- char *p3; //pointer to character type
- When pointer declared, it contains garbage value i.e. it may point any value in the memory.

- Two operators are used in the pointer i.e.
    - address operator(&) used to obtain the address of a location
    - indirection operator or dereference operator (*).
      Indirection operator gives the values stored at a particular address.

Initialization of Pointer variables:

- process of assigning the address of a variable to a pointer variable
- Syntax :

    <pointer variable> = & <variable>
    int *p, a=5;
    p = &a

- Cab be even written as : int a, *p=&a;
- Can even define pointer variable with initial value s NULL or 0
  Int *p = NULL; or int *p =0;

Accessing a variable through its pointer:

- * -> indirection operator is used to access value of a variable through its pointer
- Eg : int a, *p,b;
  a=10;
  p=&a; // p contains the address of a
  b= *p; // accesses the value of 'a' through the its pointer 'p'
  b contains the value of a, accessed through the pointer 'p'
- Can be even written as : b = *(&a);

Pointer Expressions:

- Pointer variables cab be used in expressions using * operator (indirection operator)
- Eg: y= *p1 + *p2;
  - Adds the values stored in the locations pointed by pointer variable p1 & p2
- Eg : m= *p1 * *p2 / *p1;
  If (*p1 > *p2) { . . . }
  - Can apply any operations on pointer variables

Example:

```
void main()
{
int i=105;
int *p;
p=&i;
printf("value of i=%d",*p);
printf("address of i=%d",&i);
printf("address of i=%d",p);
printf("address of p=%u",&p);
}
```

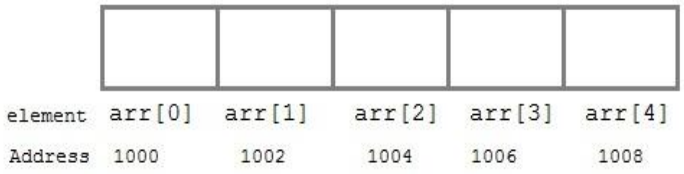
Pointer Increments and Scaling factor

- Pointers can be incremented or decremented
- Eg :
  - P1 = p1+1 // points to next location
  - P2 = p2 - 2 // points to two location backwards
- Cannot apply multiplication and division operation on pointers
- Eg : p1 = p1/2 // not valid
- When a pointer is incremented, its value gets incremented by the length of the data type to which it points to ->' scale factor'
- Eg:
  int *p1, a;
  p1=&a ; // assume address of 'a' is 1200 // 1200 is the address of a location
  p1 = p1 +1
  // P1 now contains the value 1202 , incremented by 2 bytes since it points to int data type
- Similarly , scalar factor of other data types is:

- Float : 4 bytes
- Char : 1 byte
- Int : 2 bytes

Pointers and Arrays

- When array is declared, compiler allocates a base address and sufficient amount of storage
- Base address i.e address of the first element of the array.
  int arr[5] = { 1, 2, 3, 4, 5 };
- Assuming that the base address of `arr` is 1000 and each integer requires two bytes, the five elements will be stored as follows:



- Ex:
  int *p;
  p = arr; // stores the address of the first location of the array
  // or,
  p = &arr[0]; //both the statements are equivalent.

- Ex to print the elements of an array using pointers

```
#include <stdio.h>
int main()
{
   int i;
   int a[5] = {1, 2, 3, 4, 5};
   int *p = a;     // same as int*p = &a[0]
   for (i = 0; i < 5; i++)
   {
      printf("%d", *(p+i));
    }
   return 0;
}
```

- Address of ith location is calculated using the formula
  - Address of ith location = base address + (i * scale factor of the data type)
  - Ex: address a[3] : 1000 + (3 * 2) = 1006

Pointers and Functions

- When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value.
- The parameters at the receiving end should be pointer variables
- Any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as *call by reference* in C.

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

```
void swapnum(int *num1, int *num2)
{
  int tempnum;

  tempnum = *num1;
  *num1 = *num2;
  *num2 = tempnum;
}
int main( )
{
  int v1 = 11, v2 = 77 ;
  printf("Before swapping:");
  printf("\nValue of v1 is: %d", v1);
  printf("\nValue of v2 is: %d", v2);

  /*calling swap function*/
  swapnum( &v1, &v2 );

  printf("\nAfter swapping:");
  printf("\nValue of v1 is: %d", v1);
  printf("\nValue of v2 is: %d", v2);
}
```

Difference between Call by Value and Call by Reference in C

S. No.	Call by Value	Call by Reference
1.	A copy of actual parameters is passed into formal parameters.	Reference of actual parameters is passed into formal parameters.
2.	Changes in formal parameters will not result in changes in actual parameters.	Changes in formal parameters will result in changes in actual parameters.
3.	Separate memory location is allocated for actual and formal parameters.	Same memory location is allocated for actual and formal parameters.