INTRODUCTION TO C LANGUAGE

❖ Structure of C Program

Structure	Example Program
Documentation/Comment Section	/*Program to find perimeter of a circle */
Linkage/ Preprocessor directive /File Include Section	#include<stdio.h> #define PI 3.1415
User defined Function / Subprogram prototype/Declaration Section	float perimeter();
Global Declaration Section	float radius, result;

Main Function Section
Local Declaration Part
Executable Code Part
	void main( )
	{
	float p;
	printf(" Enter radius : ");
	scanf(" %f ", & radius);
	p = perimeter( );
	printf(" Perimeter : %f ", p);
	}

User defined Function /Sub Program Definition/Implementation Section
Function 1( )
Local Declaration Part
Executable Code Part
Function 2( )
……………
Function N( )
	float perimeter( )
	{
	result= 2 * PI * radius;
	return (result);
	}

Documentation/ Comment Section: Comments are represented with statements enclosed between /* and */. The statements specify the purpose of the program for better understanding. Any statements enclose within /* and */ are not executed by the compiler. This section is optional and if required can be used in any part of the program.

Linkage/ Preprocessor directive /File Include Section: This section contains instructions which are processed before the compiler executes the program statements. There are two types of pre-processor directives namely File inclusion and Macro (constant) pre-processor directive

User defined Function / Subprogram Prototype/Declaration Section: This section serves the purpose of declaring user defined functions that can be used in the program.

Global Declaration Section: Variables declared here for its type can be used till the end of the program and anywhere in the program.

Main Function Section: Every program should have a main function. Every C program execution begins from main function. The main function consists of Local Declarations and Executable statements. For every program the above two parts should be enclosed in flower braces with { indicates the beginning of the main function and } indicates the end of the main Function.

Local variable declaration: Variables declared here for its type can be used only in the main( ) function.

Executable Statements: The statements can be an input or output or function call or assignment or return statement.

User defined Function /Sub Program Definition/Implementation: Functions defined by the user. These are placed mostly after the main( ) function or above the main( ) function.

❖ <u>Tokens / Elements of ‘C’ programming language:</u>

Tokens: The smallest possible individual valid units or components of a C- program are called Tokens. The Tokens in ‘C’ language are
K**ey words / Reserved words, Identifiers, Constants, Operators, Strings, Comments**

<u>Comments:</u>

- Any set of statements enclosed within /* and */ are taken as comments.
- These statements are not executed by the compiler.
- These statements are given by the programmer to specify the purpose of the program or statement.
- These statements convey information for the programmer and others for proper understanding.
- Comments are optional for a program and can be written anywhere in the program.
- **Example:** /* welcome to programming */

<u>Key words/ Reserve words :</u>

- These words have predefined meaning. The meaning of these words cannot be changed.
- All keywords must be written in small letters only (except additional c99 keywords) and no blank space allowed in keywords
- Keywords should not be used as variable / identifier names.
- These words can be used anywhere in the program.
- **Examples:** auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while etc.

<u>Identifiers :</u>

It is the name given to identify a memory location(variable), function, structure, union, macro, label, array etc., There are **two types** of identifiers
.

Rules for declaring User defines Identifiers:

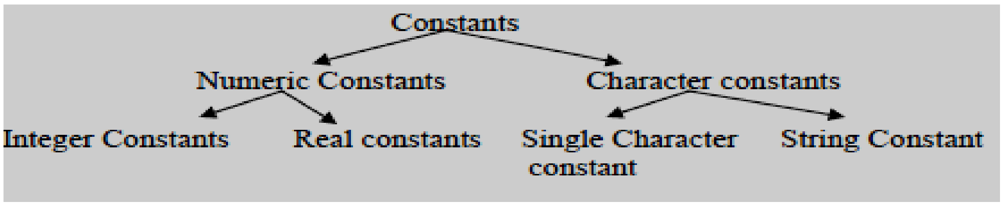
1. An identifier must consist of only alphabetic character , digits, and underscore.
2. First character must be alphabetic character or under score and should not be a digit.
3. Second character onwards can be alphabetic character or digit or under score.
4. Identifier name can be up to 31/63 characters depending on the system.(but the first 8 characters are significant)
5. It cannot be same as key word / reserved word.
6. May not have a white space or any other special symbol except under score.
7. An identifier defined in a C standard library should not be redefined.
8. C – language is Case-sensitive. So that, the variable name should be defined specifically to uppercase or lower case letters.

Examples for User defines Identifiers :

--Valid identifiers	**-- In Valid identifiers**
Rollnumber, Name	4you, Do today, #welcome
Subject1, marks	auto, int
College_name	

Note: Programmer should ensure that the name given for the identifier at the declaration , the same name should be used for every subsequent reference in the program.

<u>Constants:</u> Constants in C refer to fixed values that do not change during the execution of the program. Constants are classified as shown below.



(i). Integer Constants :

An integer constant refers to a sequence of digits without any decimal point. There are three types of integer constants namely decimal integer, octal integer and hexadecimal integer.

Decimal/Integer constant: It consists of set of digits 0 through 9, preceded by an optional – or + sign. No special symbol and spaces is allowed between digits. Examples : 123, -321, +78

Octal Integer constant : It consists of set of digits 0 through 7, preceded with 0. No special symbol and spaces is allowed between digits. Examples : 037, 0435

Hexadecimal Integer constant : It consists of set of digits 0 through 9, and A to F *or* a to f, preceded with 0x or 0X. No special symbol and spaces is allowed between digits. Examples: 0x37, 0x435A

(ii). Real Constants:

Numbers that contain fractional parts and whose value does not change. Such numbers are called real (floating point) constants. It is possible to represent real constant number by omitting digits before decimal point or digits after the decimal point. Examples: 0.0083 435.56 .345 215.

Real number may also be expressed in exponential (Scientific) notation as shown below:

Example: 215.15 can be written as 2.1515e2. (Mantissa, Base, Exponent).

(iii). Single Character Constants:

Any character enclosed in single quotes is considered as single character constant.

Examples: '5', 'x'

The 'C' Programming Language supports backslash character constants that can be used in output functions along with the string. These constants have predefined meaning and they are treated as single character constant.

(iv). String Constant / Strings:

Any sequence of characters enclosed in double quotes is considered as string constant.

Examples: "welcome" , "134".

Note: The 'a' is character constant, where as "a" is string constant.

(v). Boolean constants: The 0 (false) and 1 (true) is called as Boolean constant.

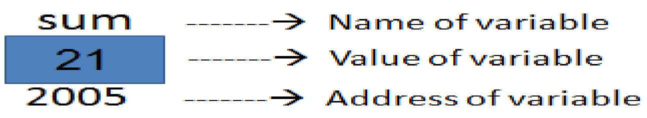
❖ Variables:

- It is an identity for a memory cell, which stores input values, computational values and output values.
- It is named as variable because, its value changes continuously throughout program execution.
- The variable declaration specifies to compiler about what type of value is stored in the variable and also specifies about list of variables used in the program.

Variable Representation:

A variable is a location in memory that is identified by a name and address and can store value of a particular type as shown below.

Example: int sum=21;



Rules for declaring Variable Name :

1. An variable name must consist of only alphabetic character , digits, and underscore.
2. First character must be alphabetic character or under score and should not be a digit.
3. Second character onwards can be alphabetic character or digit or under score.
4. Variable name can be up to 31/63 characters depending on the system.(first 8 characters are significant)
5. It cannot be same as key word / reserved word.
6. May not have a white space or any other special symbol except under score.
7. C – language is Case-sensitive. So that, the variable name should be defined specifically to uppercase or lower case letters.

Examples for User defines Identifiers :

Valid identifiers	*In Valid identifiers*
Rollnumber, Name	4you, Do today, #welcome
Subject1, marks	auto, int
College_name	

Declaration of a Variable:

1. Single variable of specified data type can be declared as
   data_type_name var_name; **Example:** int tot_marks;
2. Multiple variables of same data type can be declared as
   data_type_name var_name1, var_name2,….; **Example:** int a, b, c;

Initialization of Variables:

The variables can be initialized by using assignment operator as shown below.

1. The variable declaration and inialization can be specified separately as shown below
   Syntax : ***data_type_name var_name;***
   var_name= Initial_value;
   Example: int sum;
   sum=0;
2. The variable declaration and inialization can be specified combinely as shown below ***Syntax :***
   data_type_name var_name = Initial_value; **Example:** int sum=0;

❖ Data Types:

The ‘C’ programming language provides a rich set of data types. The Data type defines the set of value that can be stored, type of value and also defines the size of memory allocated to the variable. The ‘C’ language Data type can be classified as:

1. **Primary Data types:** These data types are fundamental data types of the language. They are
   A. Integer data type B. Float (or) real data type
   C. Character data type D. Void data type

2. **Derived Data types:** These data types are designed using the primary data types. They are usually Arrays, Structures, Unions, Pointers, Functions etc,.

3. **User defined Data types:** These are new data types designed by the programmer for specific application. These are designed using typedef and **enum.** The general representation for declaring a variable for a type is done as shown below: **data_type variable_name;**

Integer Data type: In mathematics integers are whole numbers (no decimal point). The integer data type supports three categories of integer data type to support wide verity of values. They are short int, int and long int data types. And each data type can be either signed or unsigned data type to support both positive and negative values. **Example:** 435, -10500, +15, -25, 32767 etc.

The integer data type occupies one word of storage, and since word size of machine/computer varies, the size of the integer data type depends on the computer. And note that a signed integer uses one bit for sign and rest of the bits for magnitude of the number. The table shows notation, size, range of values, data type character.

Data type Notation	Size in Bytes	Range of values	Data type character
signed short int / short int / short	1 Byte	-128 to +127 (-2^7 to $+2^7-1$)	%hd
unsigned short int / unsigned short	1 Byte	0 to 255 (0 to 2^8-1)	%hu
signed int / int	2 Byte	-32,768 to +32,767 (-2^{15} to $+2^{15}-1$)	%d
unsigned int / unsigned	2 Byte	0 to 65,535 (0 to $2^{16}-1$)	%u

signed long int / long int / long	4 Byte	-2,147,483,648 to 2,147,483,647 (-2^{31} to $+2^{31}-1$)	%ld
unsigned long int / unsigned long	4 Byte	0 to 4,294,967,295 (0 to $2^{32}-1$)	%lu
long long int	8 Byte	-2^{63} to $2^{63}-1$	(Added by C99)
unsigned long long int	8 Byte	$2^{64}-1$	(Added by C99)

Float/Real Data type:
The float data type supports larger vales than integer data type and also supports more accuracy/efficiency of value than integer. The float value has a integer part and fractional part that are separated by a decimal point. It supports both decimal point notation and scientific notation to represent float values.

Example: 3.01234, 0.00034, 1234.0 → Decimal point notation.
15.0e-04 (0.0015), 2.345e2 (234.5), 12e5 (1200000.0) → Scientific notation.

The float data type occupies generally 4 words of storage with 6-digits of precision (no. of digits of after decimal point). The precision can be increased to increase accuracy. The table shows notation, size, range of values, data type character.

Data type Notation	Size in Bytes	Range of values	Data type character
float (single precision up to 6 digits)	4 Byte	-3.4×10^{-38} to 3.4×10^{38}	%f or %e
double (double precision up to 14 digits)	8 Byte	-1.7×10^{-308} to $1.7 \times 10^{+308}$	%lf
long double (extended double precision 22 digits)	10 Byte	3.4×10^{-4932} to $1.1 \times 10^{+4932}$	%Lf

Character data type:
The character data type can store a single character value, i.e., a letter, a digit, a special symbol. Each character is enclosed in single quotes. The character data type can be either signed or unsigned. The character data type occupies 1-byte of storage. **Example:** ‘a’, ‘2’, ‘R’, **‘;’** etc.

The table shows notation, size, range of values, data type character.

Data type Notation	Size in Bytes	Range of values	Data type character
signed char / char	1 Byte	-128 to +127	%c
unsigned char	1 Byte	0 to 255	%c

Void data type: The void data type has no values. It can play a role of generic type, meaning that it can represent any of the other data type vales. This void type usually used to specify the type of functions. The type of function is said to be void when it does not return any value to the calling function.

❖ Operators & Expressions:

‘C’ language supports a rich set of built-in operators. An operator indicates an operation to be performed on data. The operator is a special symbol, which denotes some computation.

Example: a+b : where ‘+’ denotes operator to perform addition and a, b are operands.

(i) Arithmetic operators: There are five main arithmetic operators in ‘C’. They are ‘+’ for additions, ‘-' for subtraction, ‘*’ for multiplication, ‘/’ for division and ‘%’ for remainder after integer division. This ‘%’ operator is also known as modulus operator. The + and - operators can be used as unary and binary operators, where as other operators are binary operators.

Operands can be integer quantities, floating-point quantities or characters. The modulus operator requires that both operands be integers & the second operand be nonzero. Similarly, the division operator (/) requires that the second operand be nonzero, though the operands need not be integers. Division of one integer quantity by another is referred to as integer division. With this division the decimal portion of the quotient will be dropped. If division operation is carried out with two floating- point numbers, or with one floating point number. & one integer, the result will be a floating-point quotient.

Example: **int** x=20; and int y=10;

x+y→ 30 x-y→ 10 x*y→ 200 x/y→ 2 x %y→ 0

(ii) Relational operators: Relational operators are symbols that are used to test the relationship between two variables, or between a variable and a constant. All these operators are binary operators. 'C' has six relational operators as follows:

> greater than < less than != not equal to >= greater than or equal to

<= less than or equal to = =equal to

These relational operator are used to form logical expression representing condition that are either true or false. The resulting expression will be of type integer, since true is represented by the integer '1' and false is represented by the value '0'. Let us understand operations of these relational operators with the help of an **Example:** int x=2, y=3, z=4;

x<y true 1, (x+y) >=z true 1, (y+z)>(x+7) false 0, z!=4 false 0, y = =3 true 1

(iii) Logical/Boolen operators: There are three logical operators in C language, they are **and, or** & **not**. They are represented by **&& , ||, !** respectively. These operators are referred to as logical and, logical or, logical not (logical negation) respectively. The ! operator is unary operator, where as &&, || operators are used as binary operators.The result of a logical and operation will be true only if both operands are true, whereas the result of a logical or operation will be true if either operand is true or if both operands are true. The result of a logical not operation will be true if operand is false and vice-versa. The results are shown in the truth table.

A	**B**	**A&&B**	**A||B**	**!A**
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

The logical operators act upon operands that are themselves logical expressions. The logical operators are used to design complex relations and to find the logical relationship between two values.

Example: Suppose x=7, y=5.5, z= 'w'

The Logical expressions using these variables are as follows:

Expression	**Interpretation Value**
(x>=6) &&(z= ='w')	true 1
(x>=6) (y = =119)	true 1
(x<=6) && (z=='w')	false 0

(iv) Assignment operators: There are several different assignment operators in C. All of them are used to form assignment expression, which assign the value of an expression to an identifier. The most commonly used assignment operator is =. The assignment operator is a binary operator. The assignment expressions that make use of this operator are written in the form:

Identifier = expression.

Identifier 1= identifier 2 = - - - - = expression are allowed in 'C'.

In such situations, the assignments are carried out from right to left.

Example : X = 10; It will cause the integer value 10 to be assigned to x .

X = y = 10; It will cause the integer value 10 to be assigned to both x and y.

Compound Assignment Operators: 'C' also contains the five additional assignment operators +=, -=, *=, /= & %=. called as compound assignment operators.

expression1 + = expression2; **is equal to** expression1= expression1 + expression2;

Example: sum+=x; **is equal to** sum=sum + x;

(v) Bitwise operators: The 'C' language includes operators to manipulate memory at the bit level. This is useful for writing low-level hardware or operating system code. The bitwise operations are typically used with unsigned types. The bit-wise negation operator is unary, where as all operators are binary operators.

~ Bitwise Negation (unary) – changes (flips) form 0 to 1 and 1 to 0.

&	Bitwise And
|	Bitwise Or
^	Bitwise Exclusive Or
>>	Right Shift by RHS (divide by power of 2) Ex: x >> y (for positive int same as x/2y)
<<	Left Shift by RHS (multiply by power of 2) Ex: x << y (for positive int same as x*2y)

Note: Do not confuse the Bitwise operators with the logical operators. The bitwise operators are one character wide (&, |) while the logical connectives are two characters wide (&&, ||). The bitwise operators have higher precedence than the logical operators.

Truth table for bit-wise operators:

A	B	A & B	A | B	A ^ B	~A
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

Examples:

Bitwise AND

```
A (42) : 00000000 00101010
B (15) : 00000000 00001111
------------------------------------
& (10) : 00000000 00001010
------------------------------------
```

Bitwise OR

```
A (42) : 00000000 00101010
B (15) : 00000000 00001111
------------------------------------
| (47)  : 00000000 00101111
------------------------------------
```

Bitwise XOR

```
A (42) : 00000000 00101010
B (15) : 00000000 00001111
------------------------------------
& (37) : 00000000 00100101
------------------------------------
```

Bitwise NOT

```
A (42) : 00000000 00101010
------------------------------------
~ (-43) : 11111111 11010101
------------------------------------
```

Left shift(<<) and Right Shift (>>)Operators: The bitwise shift operators '<<' and '>>' shift all bits of the left-hand operand either left or right by the number of bits specified in the right-hand operand.

The '<<'operator is fairly straightforward: all bits are shifted left and 0-bits shift in from the right.

Example:

```
int  i = 2; // In binary, i is: 00000010
i = i << 1; // Shift left by 1 bit:  00000100 ← 0
```

The '>>' operator is slightly more complicated in that its behavior depends upon whether we are operating on a signed or unsigned integer. For unsigned integers, the right-shift is logical, i.e., the uppermost bits are filled with 0's.

Example:

```
int  i = 128; // In binary, i is: 10000000
i = i >> 1; // Shift right by 1 bit:      0→ 01000000
```

(vi) Increment/Decrement operators: The increment operator '++' and decrement operator '--' are shorthand ways of adding or subtracting 1 from an integer-valued variable. Both operators are unary operators. The increment and decrement operators can each be utilized in two different ways, depending on whether the operator is written before or after the operand. They are prefix and postfix form. If the operator precedes the operand, then the value of operand will be altered before it is used for its intended purpose within the program. If, however the operator follows the operand then the value of the operand will be changed after it is used.

(Pre-increment)	**++a**	equivalent to **a = a+1;** returns a after increment
(Post-increment)	**a++**	equivalent to **a = a+1;** returns a before increment
(Pre-decrement)	**- -a**	equivalent to **a = a-1;** returns a after decrement
(Post-decrement)	**a- -**	equivalent to **a = a-1;** returns a before decrement

The difference lies in what the value of the expression is (yes, the statement 'i++' is in fact an *xpression*).

Example:
```
int i,j;    i=3;
j = i++; // j=3, i=4
```

Note that the value of the expression ‘i++’ is the value of i ***before*** incrementing. Now consider:

Example: int i,j;   i=3;

j = ++i; // j=4, i=4

With the prefix form of the operator, the value of the expression ‘++i’ is the value of i ***after*** incrementing.

(vii) Conditional operators: The conditional operator ‘?:’ is like an embedded if...else statement. It is a Ternary Operator. The main difference is a if…else statement cannot be used in an expression, the conditional operator can be used in the expressions.

Syntax : **expression 1 ? expression 2 : expression 3;** (or) **expression ? true_clause : false_clause;**

First, the EXPR expression is evaluated. If true, the value of the conditional operator is the result of executing the TRUE_CLAUSE. Otherwise, the value of the conditional operator is the result of executing the FALSE_CLAUSE.

Example 1: (i< 1) ? 0:200; i is integer variable here.

The expression (i<1) is evaluated first, if it is true the entire conditional expression takes on the value 0. Otherwise, the entire conditional expression takes on the value 200.

Example 2: (x>y) ? z=x : z=y ; is equivalent to if (x > y) z = x; else z = y;

(viii) Special operators:

(a) The Comma (,) Operator: A strange operator, the comma ‘,’ is used to combine related statements/ expressions /any components as specified in the following case the comma operator can be used to separate variables names, if they are belonging to same data type and in input and output statements.

Example: int a, b;

scanf (“ %d%d%f ”, &x, &y, &avg);

(b)The sizeof operator: This operator returns the size of its operand, in bytes(memory allocated). This operator always precedes its operand. The operand may be an expression, or it may be a cast. It is a unary operator.

Example: sizeof (x); sizeof (y);

If x is of integer type variable and y is of floating point variable then the result in bytes is 2 for integer type and 4 for floating point variable.

.

(c) The Cast operator: A cast is also considered to be unary operators. In general terms, a reference to the cast operator is written as (type). It converts the a variable value to the specified data type. It is a unary operator. **Example: x=(int) average;**

(d) Addressing/Pointer operators : The C language defines pointers, and a set of operators which can be applied to them, or which allows them to be built. All these operators are unary operators. And they are

***ptr**	returns the value pointed by the pointer **ptr**
&var	returns the address of variable **var**

(e) Member Selection Operators: There are two member selection operators **dot(.)** and **arrow(→)** operators. Specifically dot(.) operator used with structures, where as arrow(→) operator is used with structures and pointers.

Expressions:

It is defined as the combination of operands and operators where operand can be a variables or constants or direct values. **Example: x=a*(b/c)%d;**

Types of Expressions: An Expression can be classified as specified below.

Type	Example
Arithmetic Expression	c=a+b;
Relational Expression	a>b;
Logical Expression	(a<10) && (b>20)
Assignment Expression	c=b;
Bitwise Expression	a<<2;
Conditional Expression etc.	(a<b) ? c=0 :c=1;

Evaluation of Expressions: Expressions are evaluated using an assignment statement of the form shown below

variable = expression;

When an assignment statement is encountered, the right hand side expression is evaluated first and then replaces the value of the variable on the left hand side. Note that, all variables used in the expression must be assigned values before evaluation is attempted.

Example: let a=9; b=12; and c=3, then

```
x=a*b-c;
x=9*12-3;
x=9*9 = 81;
```

Rules of evaluation of Expressions:

1. When parentheses are used in the expression, then the expression with in parenthesis assumes highest priority and it should be evaluated first.
2. If parentheses are nested, then expression evaluation begins with inner most sub expression.
3. Apply operator precedence rules, if more operators of same type are in the expression.
4. Apply associativity of operators, if more operators of same precedence are in the expression.

To have a complete understanding of how expressions are evaluated one should know about the following things:

- Operator Precedence
- Associativity of operators
- Type Conversion

Note: The operator precedence rules and associativity of operators can be changed by introducing parenthesis in the expression. We know that, parenthesized expression assumes highest priority in evaluation process.

❖ <u>Precedence and Associativity of Operators:</u>

Precdence Group (Highest to Lowest)	**Operators**	**Associativity**
(parenthesis) subscript	( ), [ ], →, .	L → R
Unary operators	-, +, !, ~, ++, – –, (type), *, &, sizeof	R → L
Multiplicative	* , /, %	L → R
Additive	+, –	L → R
Bitwise shift	<<, >>	L → R
Relational	< , <=, >, >=	L → R
Equality	= = , !=	L → R
Bitwise AND	&	L → R
Bitwise exclusive OR	^	L → R
Bitwise OR	|	L → R
Logical AND	&&	L → R
Logical OR	| |	L → R
Conditional	? :	R → L
Assignment	=, +=, –=, *=, /=, %=, &=, ^= |=, <<=, >>=	R → L
Comma	,	L → R

Example: Consider the expression 5 * 4 + 8 / 2

In the above expression we have two operators with the same precedence i.e., * and / so we apply associativity first i.e., we evaluate the expression from left to right based on the above table. Next we consider the first occurrence of the operator with that precedence and evaluate the operands placed before and after that operator for the operation. i.e.,5 * 4.

❖ Type conversion:

The 'C' language provides a facility of mixing different type of variables and constants in expression. But while evaluating those expressions, it is needed to convert the data type one variable may be converted to another data type. This is called as "Type conversion." There are two types of conversions. They are

1. Implicit (Automatic) type conversion. 2. Explicit type conversion (Type Casting).

(i) Implicit (Automatic) type conversion: In this conversion, the user or programmer does not provide any statements to perform conversion process. The compiler automatically converts data type of variables one type to another. This is called as Implicit type conversion.

Normally, in the expression if operands are of different types, then lower data type value is converted into higher data type and the evaluation process proceeds.

Example: let int x; float a=10.5; int b=2;

$$x = \frac{a \rightarrow \text{float}}{b \rightarrow \textbf{int} \rightarrow \textbf{float (Conversion)}}$$

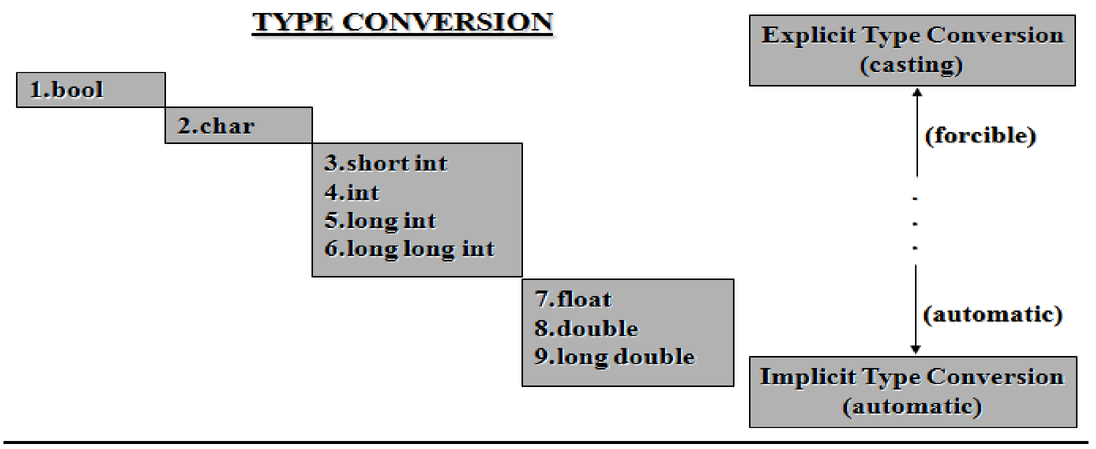
$$x = \frac{10.5 \rightarrow \text{float}}{2 \rightarrow \text{int} \rightarrow \text{float}} \qquad x = \frac{10.5}{2.0} = 5.25 \rightarrow \text{float} \rightarrow \text{int} = 5.$$

(ii) Explicit type conversion (Type Casting): In this conversion, the user or programmer forcibly provides the statements to perform conversion process. The programmer uses the **cast operator** to convert one type of data to another type by specifying the data type required. This is called as Type casting or Explicit type casting.

<u>**Syntax:**</u> **variable = (data_type) expression/variable.**

<u>**Example:**</u> x= (int) 7.5 =7;
b= (double) sum/x;

<u>**Conversion Hierarchy:**</u>


TYPE CONVERSION
1.bool
2.char
3.short int
4.int
5.long int
6.long long int
7.float
8.double
9.long double
Explicit Type Conversion (casting)
(forcible)
(automatic)
Implicit Type Conversion (automatic)