UNIT 2

- Character Set, Identifiers and keywords, Data types, Constants, Variables.
- Operators: Arithmetic, Relational and logical, Assignment, Unary, Conditional, Bitwise, Comma, other operators.
- Expression, statements, Library Functions, Preprocessor.
- Data Input and Output: getchar( ), putchar( ), scanf( ), printf( ), gets( ), puts( ), Structure of C program .

2.1 C character set

As every language contains a set of characters used to construct words, statements, C language also has a set of characters which include alphabets, digits, and special symbols.
C language supports a total of 256 characters. Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters.

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

C language supports all the alphabets from the English language. Lower and upper case letters together support 52 alphabets.

Lower case letters - **a to z**

UPPER CASE LETTERS - **A to Z**

Digits

C language supports 10 digits which are used to construct numerical values in C language.
Digits - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Symbols

C language supports a rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, backspaces, and other special symbols.

Special Symbols - **~ @ # $ % ^ & * ( ) _ - + = { } [ ] ; : ' " / ? . > , < \ | tab newline space NULL bell backspace verticaltab etc.,**

Every character in C language has its equivalent ASCII (American Standard Code for Information Interchange) value.

2.2 Identifier

Identifier refers to the name of the variable, function, array, structure. These are user defined names that are used to make program and consists of characters and digits. During the programming user declares various identifiers that serve an important part for the solution of the given problem.

Naming Rules for Identifiers

1. Identifiers may only include the following characters: letters ('a'..'z', 'A'..'Z'), digits ('0'..'9') and underscores ('_'). C is case sensitive, so seaside, SeaSide and SEASIDE are distinct identifiers.
2. The first character must not be a digit. It must be an alphabet or underscore.
3. The identifier must not be one of C's 32 keywords.
4. only first 31 characters are significant.
5. It must not have white spaces.

Example: number, number2, a, x,

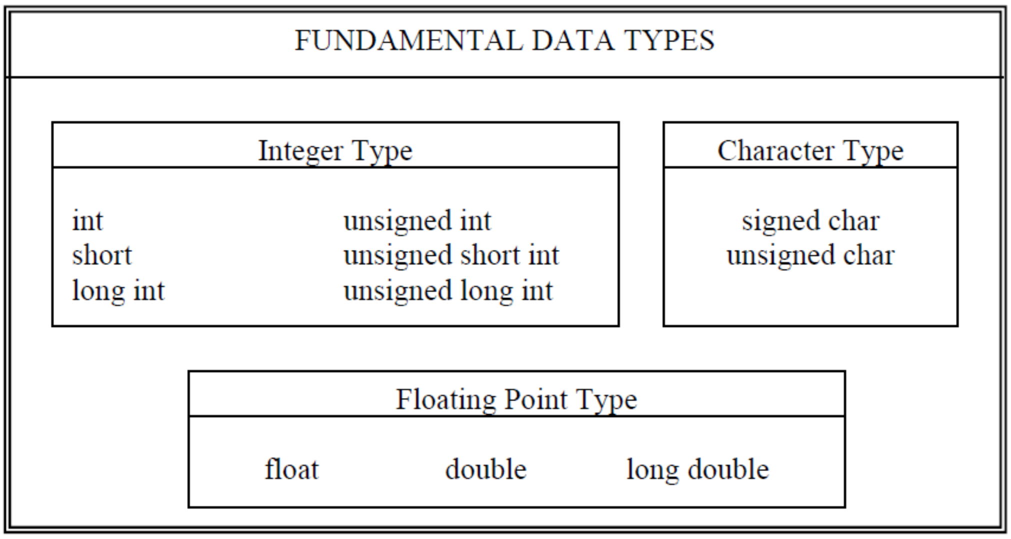
2.3 Keywords

Key word plays an important role in every programming language. These are reserved words have fixed meaning that cannot be changed by the programmer. Whenever compiler encounters these keywords during the time of compilation, it takes necessary action that is predefined. So keywords serve as basic building blocks for programming statements.

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

2.4 Data types

C language is rich in data types. The fundamental data types which can be used in C are integer data types, character data types and floating point data types. A schematic representation of the various data types in C is given below



Each of the data types and their corresponding keywords are given in the table below.

DATA TYPES	**KEYWORD**
Character	Char
Unsigned Character	unsigned char
Signed Character	signed char
Signed Integer	signed int (or int)
Signed Short Integer	signed short int (or short int or short)
Signed Long Integer	signed long int (or long int or long)
Unsigned Integer	unsigned int (or unsigned)
Unsigned Short Integer	unsigned short int (or unsigned short)
Unsigned Long Integer	unsigned long int (or unsigned long)
Floating Point	Float
Double Precision Floating Point	double
Extended Double Precision Floating Point	Long double

Integer data type is used to represent an integer-valued number. Character data type represents one single alphabet or a single-digit integer. Each character type has an equivalent integer representation. Integer and character data types can be augmented by the use of the data type qualifiers, short, long, signed and unsigned. Floating point data types are used to represent real numbers. Basic floating point data type offers six digits of precision. Double precision data type can be used to achieve a precision of 14 digits. To extend the precision furthermore, we may use the extended double precision floating point data type.

In the table below, the memory size required by each data type in bits and the range of values they can possess are listed.

DATA TYPE	SIZE in BITS	RANGE
char or signed char	8	-128 to 127.
unsigned char	8	0 to 255.
int or signed int	16	-32768 to 32767.
unsigned int	16	0 to 65535.
short int or signed short int	8	-128 to 127.
unsigned short int	8	0 to 255.
long int or signed long int	32	-2,147,483,648 to 2,147,483,647.
unsigned long int	32	0 to 4,294,967,295.
float	32	3.4e-38 to 3.4e+38.
double	64	1.7e-308 to 1.7e+308.
long double	80	3.4e-4932 to 3.4e+4932.

2.5 Constants

In C programming language, a constant is similar to the variable but the constant hold only one value during the program execution. That means, once a value is assigned to the constant, that value can't be changed during the program execution. Once the value is assigned to the constant, it is fixed throughout the program. A constant can be defined as “It is a named memory location which holds only one value throughout the program execution.”

Integer constant: An integer constant can be a decimal integer or octal integer or hexadecimal integer. A decimal integer value is specified as direct integer value whereas octal integer value is prefixed with 'o' and hexadecimal value is prefixed with 'OX'.

Floating-point constant: A floating-point constant must contain both integer and decimal parts. Sometimes it may also contain the exponent part.

Character constant: A character constant is a symbol enclosed in single quotation. A character constant has a maximum length of one character.

String constant: A string constant is a collection of characters, digits, special symbols and escape sequences that are enclosed in double quotations.

2.6 Variable

A variable data name that may be used to store data value .unlike constants that remain unchanged during the execution of a program, a variable may take different values at different types during execution. The syntax of the declaration of variables using data type is:

```
data-type vname-1, vname-2,…..vname-n;
```

Where vname-1, vname-2, vname-n are variable names. Some examples for the variable declaration are given below.

```
int number;
char a;
long double big-number;
```

A variable name can be chosen by the programmer for the meaningful way so as to reflect to its function or nature in the program .Some examples of such names are:

```
Average
Height
Total
Counter_1
class_strength
```

As mentioned earlier, variable names may consist of letters, digits, and the underscore(_) character subject to the following condition:

1. They must begin with letter .Some systems permit underscore as the first character.
2. ANSI standard recognizes a length of 31 character, However, length should not be normally for more than eight character, since only the first character are treated as significant by many compilers.
3. It should not be a keyword.
4. While space not allowed.
5. Upper and lower case names are treated as different, as C is case-sensitive, so it is suggested to keep the variable names in lower case.

Some example of valid and invalid variable names are as follows:

John	Value	T_raise
Delhi	x1	Ph_value
Mark	sum1	distance

InvalidExample Include:

123	%area	25TH

Further example of variable names and their correctness are given in table

Variable Name	**Valid?**	**Remarks**
First_tag	Valid	
char	Not Valid	char is a keyword
price$	Not Valid	Dollar sign is illegal
group one	Not Valid	Blank space is not permitted
average_number	Valid	First eight character are significant
int_type	Valid	Keyword may be part of name

The distinction between declaration and definition is an important one, and it is a shame that the two words sound alike enough to cause confusion. From now on they will have to be used for their formal meaning, so if you are in doubt about the differences between them, refer back to this point.

The rules about what makes a declaration into a definition are rather complicated, so they will be deferred for a while. In the meantime, here are some examples and rule-of-thumb guidelines which will work for the examples that we have seen so far, and will do for a while to come.

2.7 Operators

C language supports a rich set of built-in operators. An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

C operators can be classified into following types:

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

Arithmetic operators

C supports all the basic arithmetic operators. 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
++	Increment operator - increases integer value by one
--	Decrement operator - decreases integer value by one

Relational operators

The following table shows all relation operators supported by C.

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 to right operand

Logical operators

C language supports following 3 logical operators. 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. These operators also perform shifting of bits from right to left. Bitwise operators are not applied to float or double.

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

Now lets see 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** operator, 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 have to be shifted. Both operands have the same precedence.

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

Conditional operator

The conditional operators in C language are known by two more names as Ternary Operator and (? : Operator)

It is actually the if condition that we use in C language decision making, but using conditional operator, we turn the if condition statement into a short and simple operator.

The syntax of a conditional operator is :

```
expression 1 ? expression 2: expression 3
```

Explanation:

- The question mark "?" in the syntax represents the if part.
- The first expression (expression 1) generally returns either true or false, based on which it is decided whether (expression 2) will be executed or (expression 3)
- If (expression 1) returns true then the expression on the left side of " : " i.e (expression 2) is executed.
- If (expression 1) returns false then the expression on the right side of " : " i.e (expression 3) is executed.

Special operator

Operator	**Description**	**Example**
sizeof	Returns the size of an variable	**sizeof(x)** return size of the variable **x**
&	Returns the address of an variable	**&x ;** return address of the variable **x**
*	Pointer to a variable	***x ;** will be pointer to a variable **x**

2.8 Expressions

Expressions in C are classified according to the operators used in them. The classifications are

- Arithmetic expression.
- Relational expression.
- Logical expression.

Arithmetic Expressions

An expression is a combination of variables constants and operators written according to the syntax of C language. In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable.

Relational Expression

An expression containing a relational operator is termed as relational expression. In an expression where there are arithmetic and relational operators present, the arithmetic expressions will be evaluated first and then the results are compared.

Logical Expression

Logical expressions are the expressions where logical operators are used to combine more than one relational expression. Logical expressions are also called compound relational expressions. The logical operators used in C are && (logical AND), || (logical OR), and ! (Logical NOT). Logical expressions also yields a value of one or zero, and used in decision statements.

Format of a logical expression:

```
        (rel-expression-1) logical operator (rel-expression-2)
```

Where rel-expression-1 and rel-expression-2 are relational expressions.

2.9 Preprocessor directives

Preprocessor is a program that processes the code before it passes through the compiler. It operates under the control of preprocessor command lines and directives. Preprocessor directives are placed in the source program before the main line before the source code passes through the compiler it is examined by the preprocessor for any preprocessor directives. If there is any appropriate actions are taken then the source program is handed over to the compiler.

Preprocessor directives are lines included in the code of our programs that are not program statements but directives for the preprocessor. These lines are always preceded by a hash sign (#). The preprocessor is executed before the actual compilation of code begins, therefore the preprocessor digests all these directives before any code is generated by the statements.

These preprocessor directives extend only across a single line of code. No semicolon (;) is expected at the end of a preprocessor directive. A unique feature of c language is the preprocessor. A program can use the tools provided by preprocessor to make his program easy to read, modify, portable and more efficient.

The C preprocessor provides four separate facilities that you can use as you see fit:

- Inclusion of header files. These are files of declarations that can be substituted into your program.
- Macro expansion. You can define *macros*, which are abbreviations for arbitrary fragments of C code, and then the C preprocessor will replace the macros with their definitions throughout the program.

- Conditional compilation. Using special preprocessing directives, you can include or exclude parts of the program according to various conditions.
- Line control. If you use a program to combine or rearrange source files into an intermediate file which is then compiled, you can use line control to inform the compiler of where each source line originally came from.

Types of Preprocessor directives:

Directive	**Function**
#define	Defines a macro substitutio
#undef	Undefines a macro
#include	Specifies a file to be included
#ifdef	Tests for macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether the macro is not def
#if	Tests a compile time condition
#else	Specifies alternatives when # if test fails

2.10 Concept of library functions and header files

Library functions are those functions that are stored in the one of the library header files and can be used directly in the source code after including particular header file in the source code. Libraries for use by C programs really consist of two parts:

1. Header files that define types and macros and declare variables and functions

2. Actual library or *archive* that contains the definitions of the variables and functions.

In order to use the facilities in the C library, you should be sure that your program source files include the appropriate header files. This is so that the compiler has declarations of these facilities available and can correctly process references to them. Once your program has been compiled, the linker resolves these references to the actual definitions provided in the archive file. Every header file has some name and followed by file extension (.h)

Header files are included into a program source file by the '#include' preprocessor directive as

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

Typically, '#include' directives are placed at the top of the C source file, before any other code. If you begin your source files with some comments explaining what the code in the file does, put the '#include' directives immediately after comments.

The #include preprocessor directive comes in two flavors:

```
#include <filename>
#include "filename"
```

The use of angle brackets <> informs the compiler to search the compiler's include directories for the specified file. The use of the double quotes "" around the filename informs the compiler to start the search in the current directory for the specified file. Examples of popular header files in C language are as follows:

`#include <string.h>` contains string manipulation functions

`#include <math.h>` contains mathematical functions

`#include<time.h>` contains time manipulation functions

2.11 Console input and output functions(printf ( ) and scanf( )

For interacting with basic console input device(Keyboard) and basic output device (monitor) there are two console input/output standard library functions that are defined in stdio.h header file as

- printf( )- formatted console output function
- scanf( )- formatted console input function

Descriptions of these functions are as follows:

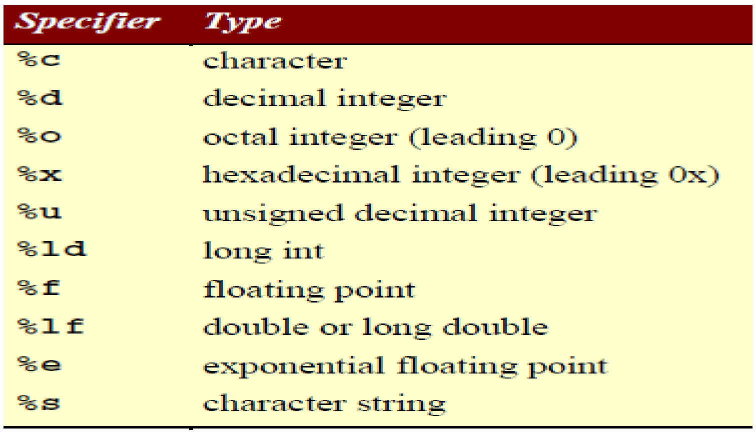
1. printf ( )

Its arguments are, in order; a control string, which controls what get printed, followed by a list of values to be substituted for entries in the control string. The prototype for the printf() is:

```
int printf("control string", argument list);
```

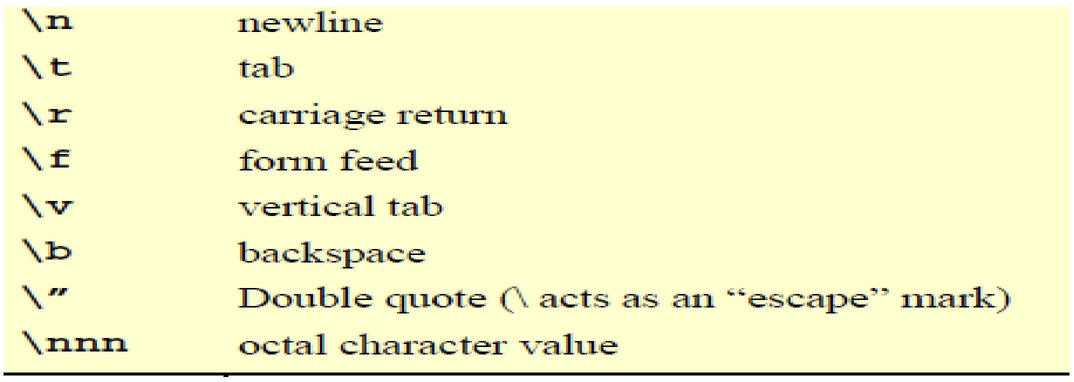
This function returns number of characters printed on to the standard output device. where the *control string* consists of 1) literal text to be displayed, 2) format specifies, and 3)special characters. The arguments can be variables, constants, expressions, or function calls -- anything that produces a value which can be displayed. Number of arguments must match the number of format identifiers. Unpredictable results if argument type does not "match" the identifier.

The following table show what format specifiers should be used with what data types:



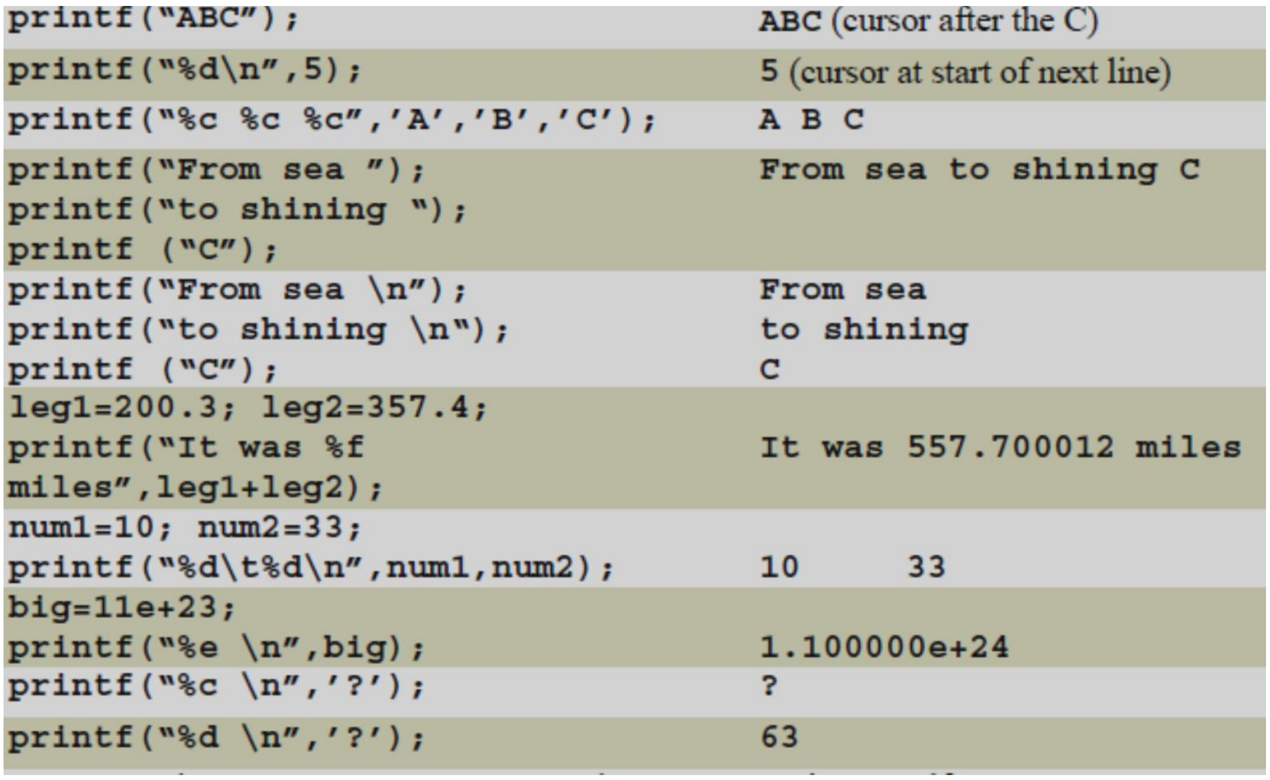
Specifier	*Type*
%c	character
%d	decimal integer
%o	octal integer (leading 0)
%x	hexadecimal integer (leading 0x)
%u	unsigned decimal integer
%ld	long int
%f	floating point
%lf	double or long double
%e	exponential floating point
%s	character string

Some of the special characters as shown in the table-



\n	newline
\t	tab
\r	carriage return
\f	form feed
\v	vertical tab
\b	backspace
\"	Double quote (\ acts as an "escape" mark)
\nnn	octal character value

Example :



`printf("ABC");`	ABC (cursor after the C)
`printf("%d\n",5);`	5 (cursor at start of next line)
`printf("%c %c %c",'A','B','C');`	`A B C`

`printf("From sea ");`
`printf("to shining ");`
`printf ("C");`
	`From sea to shining C`

`printf("From sea \n");`
`printf("to shining \n");`
`printf ("C");`
	`From sea`
	`to shining`
	`C`

`leg1=200.3; leg2=357.4;`
`printf("It was %f`
`miles",leg1+leg2);`
	`It was 557.700012 miles`

`num1=10; num2=33;`
`printf("%d\t%d\n",num1,num2);`
	`10      33`

`big=11e+23;`
`printf("%e \n",big);`
	`1.100000e+24`

`printf("%c \n",'?');`	`?`
`printf("%d \n",'?');`	`63`

2. scanf ( )

This function is used to get data from the standard input device(keyboard). The general syntax for this function is as follows:

```
        int scanf("control string", addresses of variables);
```

The control string specifies the field format which includes format specifications (as applied for printf( ) function) and optional number specifying field width and the conversion character % and also blanks tabs and new lines.

The Blanks tabs and new lines are ignored by compiler. The conversion character % is followed by the type of data that is to be assigned to variable of the assignment. The field width specifier is optional.

Example :

```
scanf("%d %d", &sum1,&sum2);
```

2.12 Type Conversion

Type conversion is the process of converting data in one type in to another type. This process of conversion can be done by two ways;

- Implicit Type Conversion
- Explicit Type Conversion

Implicit type conversion

C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance. This automatic type conversion is know as implicit type conversion

If the operands are of different types the lower type is automatically converted to the higher type before the operation proceeds. The result is of higher type.

The final result is converted to the type of the variable on the left side of the assignment operator before assigning the value to it.

Rule for implicit type conversion;

1. float to integer causes truncation of fractional part.
2. double to float causes rounding of numbers.
3. long integer to integer causes dropping of excess higher order bits.

Explicit Conversion

Many times there may be a situation where we want to force a type conversion in a way that is different from automatic conversion. Consider for example the calculation of number of female and male students in a class.

```
                                female_students
Ratio                      =  -------------------
                                male_students
```

Since if female_students and male_students are declared as integers, the decimal part will be rounded off and its ratio will 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_students / male_students
```

The operator float converts the female_students to floating point for the purpose of evaluation of the expression. Then using the rule of automatic conversion, the division is performed by floating point mode, thus retaining the fractional part of the result. The process of such a local conversion is known as explicit conversion or casting a value. The general form is

```
(type_name) expression
```

2.13 Comments

Comments are a way of inserting remarks and reminders into code without affecting its behavior. Since comments are only read by other humans, you can put anything you wish to in a comment, but it is better to be informative than humorous.

The compiler ignores comments, treating them as though they were *white space* (blank characters, such as spaces, tabs, or carriage returns), and they are consequently ignored. During compilation, comments are simply stripped out of the code, so programs can contain any number of comments without losing speed.

Because a comment is treated as white space, it can be placed anywhere white space is valid, even in the middle of a statement. (Such a practice can make your code difficult to read, however.) . There are two ways to write comment in the source file

- **Single line Comments-** This type of comments starts with // and end till end of the line.
- **Multi line Comments:** Any text sandwiched between /* and */ in C code is a comment. Here is an example of a C comment:

```
/* ...... comment ......*/
```

Comments do not necessarily terminate at the end of a line, only with the characters */. If you forget to close a comment with the characters */, the compiler will display an exterminated comment error when you try to compile your code.

Example:

```
// This is a single line comment
/*
 * This is a mult-line
 * comment.
```

2.14 Statements and Blocks

An expression such as x = 0 or i++ or printf (...) becomes a statement when it is followed by a semicolon, as in

```
x = 0;
i++;
printf (...);
```

In C, the semicolon is a statement terminator. Braces {and} are used to group declarations and statements together into a compound statement, or block, so that they are syntactically equivalent to a single statement. The braces that surround the statements of a function are one obvious example; braces around multiple statements after an if, else, while, or for. There is no semicolon after the right brace that ends a block.

2.15 Basic structure of C Program

Following is the basic structure of a C program.

Documentation	Consists of comments, some description of the program, programmer name and any other useful points that can be referenced later.
Link	Provides instruction to the compiler to link function from the library function.
Definition	Consists of symbolic constants.
Global declaration	Consists of function declaration and global variables.

main( )
{
 	
}
	Every C program must have a main() function which is the starting point of the program execution.

Subprograms	User defined functions.

Example

```
/*
 * file: circle.c
 * author: Harish Tiwari
 * date: 2020-01-29
 * description: program to find the area of a circle using the radius r
 */

#include <stdio.h>                //Link Section for including header file.

#define PI 3.1416                 //Symbolic Constant

float area(float r);              //global declaration section

int main(void)                    // main Function Section
{
  float r = 10;
  printf("Area: %.2f", area(r));
  return 0;
}

float area(float r) {             //subprograms Section
    return PI * r * r;
}
```