UNIT-II

INTRODUCTION TO C PROGRAM & OPERATORS

Topics: First C program - Hello world, How to open a command prompt on Windows or Linux How to read and print on screen - printf(), scanf(), getchar(), putchar() Variables and Data types - Variables, Identifiers, data types and sizes, type conversions, difference between declaration and definition of a variable, Constants.

Life cycle of a C program (Preprocessing, Compilation, Assembly, Linking, Loading, Execution), Compiling from the command line, Macros.

Operators – equality and assignment, Compound assignment operators, Increment and decrement operators, Performance comparison between pre and post increment/decrement operators, bitwise operators (AND, OR, NOT and XOR), Logical Operators, comma operator, precedence and associativity, Logical operators (AND, OR,NOT).

Why Program Computers?

- Computer is a machine. Depends on Instructions to accomplish something.
- Correct Instructions results expected output.
- Series of meaningful instructions is a program.
- People who write these instructions known as programmers.
- Cooking dish certain steps are followed. Particular order, quantity etc. Same in computer programming Wrong instruction, did not give the desired result.
- On RUNNING/EXECUTING a computer program we get desired output
- You need a COMPILER to run a program on a computer. It makes a program understandable to a computer. Since Computer can understand only 0sand 1s
- COMPILATION or BUILD is the process of making a program executable ona computer
- Program written in high level language called a SOURCE CODE.
- Compiler transforms to MACHINE CODE
- We shall use Code::Blocks/Online GDB/Dev C++ Compiler for typing, editing, debugging and executing a program.

Terminology

- Integrated development environment (IDE) seethe program output in one placewith an IDE . Example:Code::blocks, Eclipse, Visual Studio
- Editor
- Compilation
- Debugging: Removing them called debugging.
- Errors in code called as Bugs.

Introduction

We all program our lives in some way. Computer Program is a list of Instructions

Example Program: ISBT (Inter State Bus Terminus) to Graphic Era Univesity, Dehradun

- Head west on NH 7 toward ISBT Circle
- At ISBT Circle, take the 1st exit ontoNH307 2 Kms
- Turn left onto Post Office Rd 1 Kms
- Turn right onto Bell Road 200 Metres
- Make a left towards Graphic Era University

Let us say your parents give you one thousand rupees per month towards your expenses. Assume you hobby is listening to music and you purchase them from a website selling songs. Let us say the cost of a song is 12 rupees. Now you want to write a simple program to tell you the total cost for say N number of songs you download from the site. If you had to write the steps they would be…

Enter "Number of songs you would like to download today?"
NEnter "the cost per song:" 12
Multiply the Number of songs with cost per song: N x
12Show the final cost: Rs. _____

- This sequence of instructions is a program.
- Now computer programs are similar but have a **SYNTAX** and written in some programming language.
- Computer languages may specify that end of each statement be followed by a special character like say ; in C
- Failure to use the **terminating ;** leads to syntax error message
- Every language has its unique syntax
- Logic is known as **SEMANTICS**

```
// C Program
#include <stdio.h>
int main ()
{
   int N_Songs ;
  float Song_Cost = 12, Final_Price;
  printf("Enter Number of Songs \n");
  scanf("%d", &N_Songs);
  Final_Price = N_Songs * Song_Cost;
  printf("Cost of Songs is %0.2f \n",Final_Price);
   return 0;
}
```

Key parts of a program

- Variables and Constant
- Name and types of values
- Variables vary and contain value(s)
- Names for Memory location
- Operators = *
- Data type Character and Non-Character
- Must Declare memory locations to be reserved

Why Learn C Programming Language?

- Close to 50 years still widely used
- Language that is the basis for most modern languages
- Understand the working of a computer system
- Extensive applications. For example, compilers, operating systems embeddedapplications to name a few.
- Major portions of OS like Windows, Linux, UNIX still in C. Integration with new devicesthrough device drivers in C.
- Embedded world mobile phones, washing machines, digital cars, cameras, IoT applications use C.
- Games also due to speed and quick response
- Few languages have the ability to interact with hardware like C
- Lean, Mean and efficient
- Lean, mean efficient language

Features of C language:

- Structured
- High Level/Middle Level
- Robust
- Portable
- Extensible
- Supports pointers

Structured:

C is structured procedure-oriented language, it divides the problem intosmaller modules called functions or procedures.

High Level/Middle Level:

Programs written in C must be translated by the Compiler (system software) into machine level language.
While in middle level languages programmers can write a code that can be translated by the assemblers into machine code, which helps the programmers to interact directly with the hardware.

Portable:

C is portable that is code written on one machine can be easily ported or executed on other machines as long as that machine supports the same C compiler.

Robust:

C is robust language that is programs written in C do not crash so easily. It recovers quickly whenever programs results into an erroneous condition.

Extensible:

C is extensible language that is it allows new features and modifications to be made to the existing programs written in C.

Supports pointers:

C supports the use of pointers that allows the programmers to manipulate the memory directly.

Journey of C Language

- In 1972 Dennis Ritchie worked on improvement of a language called B by Ken Thompson which in turn was derived from BCPL by Martin Richards.
- The result was the C programming language.
- Version 4 of Unix Kernel major portions were coded in C
- First languages used for writing an OS rather than a assembly language
- Language became popular
- 1978 K&R released the first specification via book The C programming language
- To address standardization 1983 ANSI committee formed
- ANSI released standard document in1989
- Adopted by ISO in 1990
- C89 or C90 refer to same standard
- ISO C99 inline functions, one line comments, variable length arrays
- C11 and C18

The only way to learn a programming language is by writing programs in it.

Comments in C Code

```
// C hello world example
/*
Graphic Era University
B.Tech 1st sem
*/


#include <stdio.h>
int main()
{
printf("Welcome to C Programming\n");
 return 0;
}
```

- Can be placed anywhere in a program
- Clarity and explanation to others/self later on
- Only Part of the Source File Not the executable
- Cannot Nest Multiline Comments
- Avoid Trivial Comments Ex: int i
- May use comments to describe the logic in Pseudo code
- Program code change history can be maintained at the top of each program
- Code testing frequently comment out code

```
// C hello world example
/*
 Author: Prof. Mahant @ GEHU
 Purpose: Comments in C Code
*/
#include <stdio.h>
int main()
  {
        printf("Welcome to C Programming\n");
        return 0;
  }
```

Library

- Originally C library was considered as part of the UNIX operating system
- Users provided facilities for I/O, memory management etc
- User Community implementations were shared
- Incorporated as a part of standard C
- Libraries in ANSI C
- Declaration is in header files but actual code is in library files
- C lacks built-in capability to do tasks like memory management, I/O, manipulation ofdata etc
- Arrangement in the form of a standard library exists
- Compile your code and link them for usage
- Functions are specified as a part of the standard ISO C
- Small Library set of functions compared to other languages
- Developed and thoroughly tested,
- Optimised and error free
- Time saving
- Consistency of behaviour across operating platforms
- Header file contains function declarations, data type definitions, and macros
- As of now 29 Header files per C11

Preprocessor Directives (#include)

```
#include <C_HeaderFile.H>

Example
#include <stdio.h>
 OR
#include “U_HeaderFile.H”

Example
#include “projects.h”
```

- <C_HeaderFile.H> Search for header file in the standard system directories(Can showa snapshot of standard system directories on Windows with Code::Blocks compiler)

- “U_HeaderFile.H” Searches in the location where your source file exists followed bythe standard directories

C|D:\Code Blocks\MinGW\include

Eg: stdio.h, math.h

- Contents of the header file are included at that point in the source code. Think of itas a copy paste into the source file
- Step is prior to compilation. Not apart of the C Compiler. That is why no ;
- Files shared across programs
- One header file Per Line
- Function prototype declarations (No code is present), Global variables, Constants
- No Code after # line
- Software Program as the name indicates does pre-processing prior to the actualcompilation of the program in a high level language.
- Includes header file <stdio.h>
- Removes any comments in the source program

Macros Substitution

They are abbreviations of C code. They are substituted for all occurrences in your sourcefile. Example

#define PI 3.142

Ex:

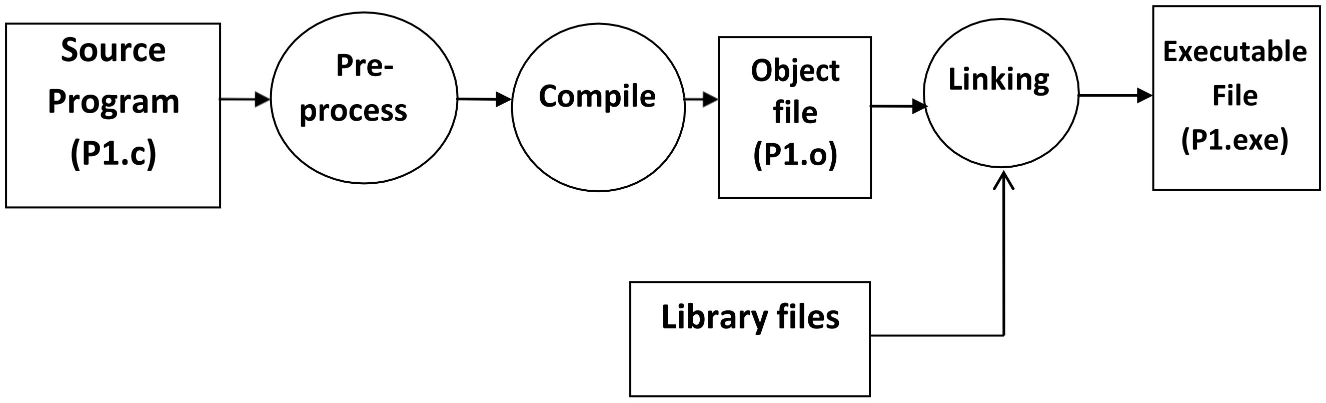
```
#include <stdio.h>
#define STRING "Hello World"
int main(void)
{
/* Using a macro to print “HelloWorld” */
printf(STRING);
 return 0;
}
```

Journey of a C Program from(Source Code to Executable)

- Create a file in code:blocks IDE and save the file with an extension .c For example: **hello.c**
- File is passed onto a software program called pre-processor. File after preprocessingbecomes **hello.i**
- The file is taken in by the compiler and converted to a assembly file called **hello.s**
- The assembler converts the file into **hello.o** (object code)
- Some calls like say printf code not yet part of the program. These references need tobe resolved. Library object code is stored in .a or .lib files.
- A software program called the linker resolves these references (linking the code)
- Output of the linker is .exe on windows and a.out on Linux platforms. The executableimage is stored on the secondary storage device.
- Loader takes the image and loads into RAM. Intimates to the CPU starting address of

the code.

Life cycle of a C program



Source Program: Any C file say **P1.C** program is edited and given to a C compiler. Figure above shows the different phases of execution of C program.

Pre-Processing: This is the first phase through which source code is passed. In this phase any statements defined in this section (before the main() function) are processed, if used in the program.
This phase includes:

- Removal of Comments
- Expansion of Macros
- Expansion of the included files.
- Conditional compilation

For ex. printf and scanf statements, if used in the program, will have been checked with their definitions stored in the header file <stdio.h>.

Compile: The next step is to compile and produce an; intermediate file that contains assembly level instructions **P1.s**.

During this phase the compiler checks for the syntax errors such as declaration of variables, initialization if any, the correct syntax of the C program etc. If any errors are encountered then they get displayed with corresponding line numbers.
The assembler converts the **P1.s** file after correction and successful compilation of the program to an ***object*** *file* **P1.o**.

Linking:
This is the final phase in which all the linking of function calls with their definitions are done. Linker knows where all these functions are implemented. Linker does some extra work also, it adds some extra code to our program which is required when the program starts and ends.

Linking produces the executable file **P1.exe (a.out by default on Linux/Unix machines)**, which is the machine code (binary code) which will be actually executed by the processor.

Loader:
Loader takes the image (**P1.exe**) generated and loads it into RAM and informs the CPU the starting address of the code for execution (running of the program).

BASIC CONCEPTS OF A C PROGRAM

• The structure of a C program is shown below:int main()

```
{
        declaration section;

        statement-1   //   Executable    section
        startsstatement-2
        statement-3
        statement-4 // Executable section ends return
        1;
}
```

User defined function-definition(s)➔*optional*

Documentation Section

- This section allows to document by adding Comments to the program. Comments are portions of the code ignored by the compiler. The comments allow the user to make simple notes in the source-code.
- For ex. // this is an example for single line comment
  /* this is an example for multiple line comment */

Preprocessor Directives

- The *preprocessor* accepts the source program and prepare the source programfor compilation.
- The preprocessor-statements start with symbol#.
- The normal preprocessor used in all programs is **include**.
- The **#include** directive instructs the preprocessor to include the specified file-contents in the beginning of the program.
- Forex:

#include<stdio.h>
main()

Global declaration section

- If user wishes to declare any variable outside the main program which needs to be accessed by any part of the program then he may declare it in this section just before main function.

- Every C program should have a function called as **main()** and is an entry point to the program (a gateway to the program) that is always executed.
- The statements enclosed within left and right curly brace is called body of the function. The main() function is divided into 2parts:

Declaration Section

- The variables that are used within the function main() should be declared in the declaration-section only.
- The variables declared inside a function are called local-variables. The compiler allocates the memory for these variables based on their types for ex. if the variable is an integer then it allocates 4 Bytes, if it's of **char** type then 1-Byte and so on. Ex: int p, t,r;

Executable Section

- This contains the instructions given to the computer to perform a specific task.
- The task may be to:
  - → display a message
  - → read data or
  - → do some task ex. add two numbers etc.

User Defined Function-definition(s): If user has any user defined functions then those definitions are written outside the main function.

Example: **Program to display a message on the screen.**

```
#include<stdio.h>    int
main()
{
        printf("Welcome to C");
        return 1;
}
```

Output:

```
                    Welcome to C
```

<u>**DATA INPUT/OUTPUT FUNCTIONS**</u>

- There are many library functions for input and output operations inC language.
- Forex:

getch( ), putchar( ), scanf( ), printf( )

- For using these functions in a C-program there should be preprocessor statement #include<stdio.h> in the beginning of the program before the declaration of main function.

Input Function

- The input functions are used to read the data from the keyboard and storein memory-location.
- Forex:

scanf(), getchar(), getch(), getche(), gets()

<u>scanf()</u>

Reads values into built-in data types
Syntax: scanf("Format specifier",&var_name);

- Reads data obeying the format specified into a memory address
- scanf(" format string ", &v1,&v2..&vn);
- Format string shall be conversion specifiers without spaces or other characters
- Followed by the location (address) of memory locations.
- &known as reference operator
- number and type of format specifiers is programmers responsibility
- forgetting & can lead to program crash or unexpected behaviour
- ignore spaces, newlines, tabs while reading input

```
#include <stdio.h>
int main()
{
         int a;
        printf("Enter integer number:");
        scanf("%d", &a );
        printf("a is = %d\n", a);
        return 0 ;
}

Enter Integer Number: 55
a is = 55
```

Output Functions

• The output functions are used to receive the data from memory-locations through the variables and display on the monitor.

• Forex:
printf(), putchar(), putch(), puts() Types of I/O Functions:

printf() (Most common Output Function)

- scanf and printf very widely used C functions

```
#include <stdio.h>
int main ()
{
 int i = 10 ;
 float j = 100;
printf("Value of i = %d and j =
%f \n", i, j);
 return 0;
}
```

```
Value of i = 10 and j = 100.000000
```

- Used for printing values of built-in datatypes
- C has no idea about what you are printing. So….
- Need to inform C, how to make sense of the data

printf(" format string ", expr1, expr2,expr3…exprn);

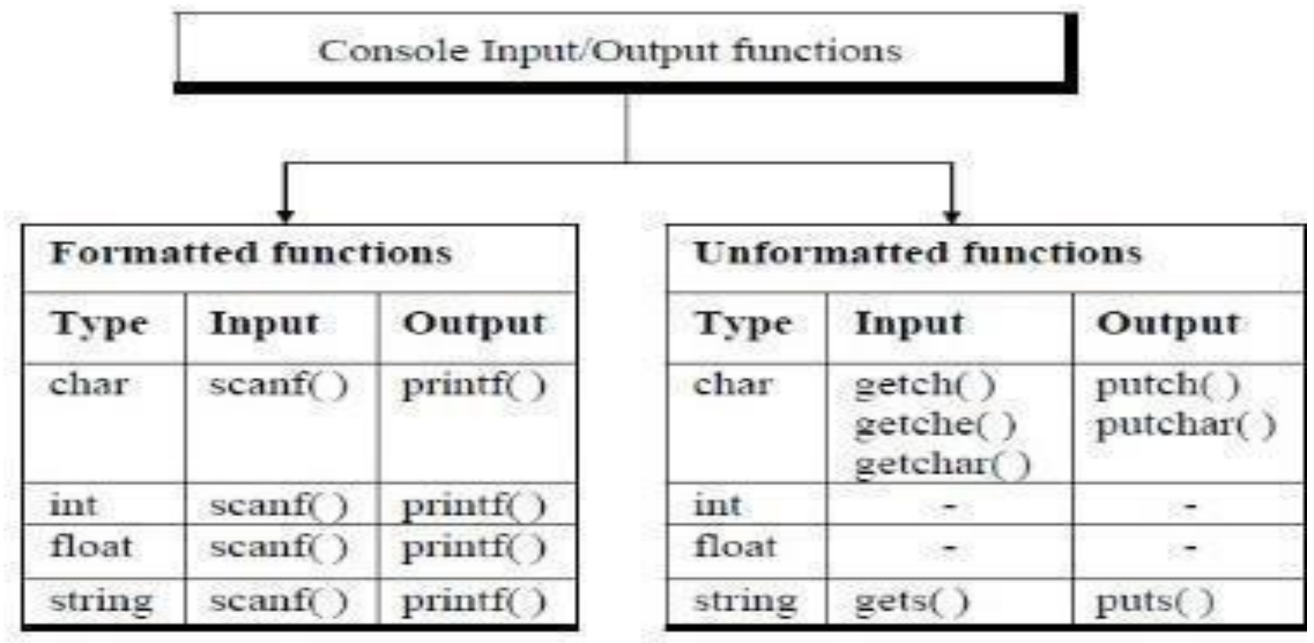
- format string informs C how the values are to be interpreted. It contains conversion specifiers and text
- Conversion specifiers begin with %

- Expressions may contain constants, variables or a combination evaluating to a built-indata type
- At the point of conversion specifiers values are substituted during printing
- Conversion specifiers specify the format of display
- For example %d indicates decimal integer
- Ordinary characters enclosed within “”are printed as is.
- No checking of number of format specifiers and expressions
- C does not perform any type checks in the printf( ) function call
- Programmer job to make sure that the data type of the variables MATCH and CORRESPOND to formatting character
- Programmers responsibility to specify correct specifier and data type
- Otherwise garbage results are printed

ILLUSTRATION

```
printf(“Value of i = %f and j = %d \n”, i,j);// Garbage Output
printf(“Value of i = %d and j = %f \n”, i);// i = 100, meaningless output
```

- There are 2 types of I/O Functions as shown below:



Console Input/Output functions

Formatted functions

Type	Input	Output
char	scanf( )	printf( )
int	scanf( )	printf( )
float	scanf( )	printf( )
string	scanf( )	printf( )

Unformatted functions

Type	Input	Output

char	getch( )
	getche( )
	getchar( )
		putch( )
		putchar( )

int	-	-
float	-	-
string	gets( )	puts( )

Character Set

Character-set refers to the set of alphabets, letters and some special characters that are valid in C language.

- Alphabets
- Digits
- White Spaces Tab Or New line Or Space
- Special Characters

1. ALPHABETS

   Uppercase letters A-Z

   Lowercase letters a-z

2. DIGITS

   0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3. Special Characters

   ~ Tilde

   ! Exclamation mark

   # Number sign

   $ Dollar sign

   % Percent sign

   ^ Caret

   & Ampersand

   * Asterisk

(Left parenthesis)

Right parenthesis

_ Underscore

Special Characters

Symbol Meaning

+ Plus sign

| Vertical bar

\ Backslash

` Apostrophe

– Minus sign

= Equal to sign

{ Left brace

} Right brace

[Left bracket

] Right bracket

: Colon

” Quotation mark

; Semicolon

< Opening angle bracket

> Closing angle bracket
? Question mark
, Comma
. Period
/ Slash

Execution Character Set (Escape Sequence)

These are unprintable, which means they are not displayed on the screen or printer. Those characters perform other functions. Examples are backspacing, moving to a newline, or ringing a bell. Each one of character constants represents one character, although they consist of two characters. These character combinations are called as escape sequence.

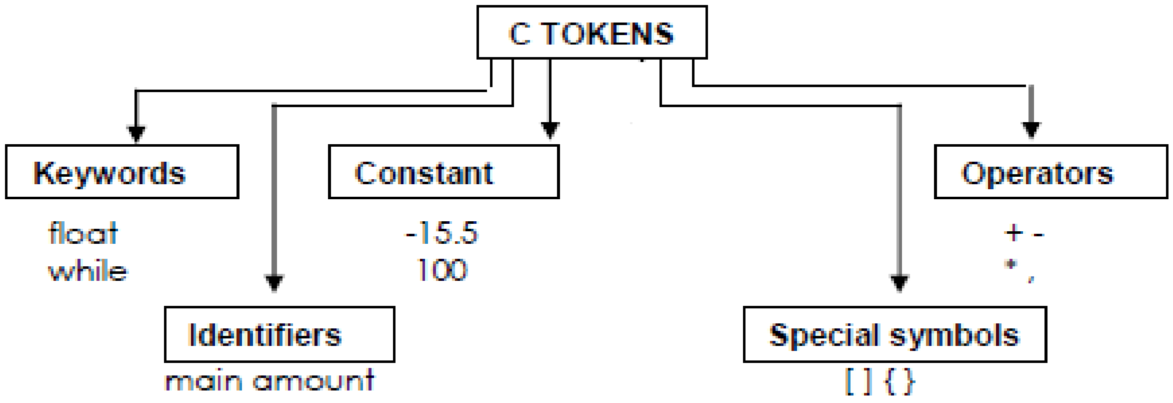
- An escape sequence character begins with a backslash and is followed byone character.
- A backslash (\) along with some characters give rise to special print effectsby changing (escaping) the meaning of some characters.
- The complete set of escape sequences are:

Character	ASCII value	Escape Sequence	
Null	000	\0	Null character
Alarm (bell)	007	\a	Beep Sound
Back space	008	\b	Moves previous position
Horizontal tab	009	\t	Moves next horizontal tab
New line	010	\n	Moves next Line
Vertical tab	011	\v	Moves next vertical tab
Form feed	012	\f	Moves initial position of
next page			
Carriage return	013	\r	Moves beginning of the line
Double quote	034	\"	Present Double quotes
Single quote	039	\'	Present Apostrophe
Question mark	063	\?	Present Question Mark
Back slash	092	\\	Present back slash
Octal number	\00		
Hexadecimal number	\x		

Tokens

- A token is a smallest element of a C program.
- One or more characters are grouped in sequence to form meaningful words. These meaningful words are called tokens.
- The tokens are broadly classified as follows

  → Keywords ex: **if, for, while, int, float, char**
  → Identifiers ex: **sum, length**
  → Constants ex: **10, 10.5, 'a', "sri"**
  → Operators ex: **+ - * /**



→ Special symbols ex: **[], (), {}**

Sample C Code Tokens

```
int main ( )
{
 const float PI = 3.142 ;
 int a = 0 ;
 a = a + 10 ;
 printf( “hello world \n” ) ;
 return 0 ;
}
```

Keywords: int, const, float
Operators: = , +
Identifiers: a, printf, main, PI
Constants: 3.142
Strings: “hello world \n”
Special Characters: { } ( )

Keywords

- Keywords are tokens which are reserved by C for a definite purpose.
- Each keyword has fixed meaning and that cannot be changed by the user.
- C supports about 32keywords.

Rules for using keywords

- Keywords cannot be used as a variable or function.
- All keywords should be written in lower letters.
- Following keywords are as listed below:

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

Identifier

- Identifier is used to represent various part of a program such as variables, constants, functions etc.
- An identifier is a word consisting of sequence of

→ Letters

→ Digits or"_"(underscore)

- Forex:

  Average,_Percent, total100

Few Valid Identifiers

Ex. Length, _breadth, area51, Num5_sol, _percent_ etc.

Few Invalid Identifiers

Ex. 10Average, -Sum, Sol?ution, Div/quot etc.

Rules for an Identifier

- Can start with a letter or underscore
- Can't start with a digit or special symbol or operator
- Can't contain special symbol or operator
- Can't be a keyword

Constants

- A constant is an identifier whose value remains fixed throughout the execution of the program.

- The constants cannot be modified in the program.
- For example:
  1, 3.14512 , ‘z’,“pcdnotes"

Different types of constants are:

1. **Integer Constant:**
   - An integer is a whole number without any fraction part.
   - There are 3 types of integer constants:
     i) Integer & Decimal constants (0 1 2 3 4 5 6 7 89)
     Integer Constants
     For ex: 0, -9, 22
     ii) Octal constants (0 1 2 3 4 5 67)
     For ex: 021, 077, 033
     iii) Hexadecimal constants (0 1 2 3 4 5 6 7 8 9 A B C D EF) For ex:
     0x7f, 0x2a, 0x521

2. **Floating Point Constant**

- The floating point constant is a real number.
- The floating point constants can be represented using 2forms:

i) Fractional Form

- A floating point number represented using fractional form has an integerpart followed by a dot and a fractional part.
- Forex:
  0.5, -0.99

ii) Scientific Notation (Exponent Form)

- The floating point number represented using scientific notation has threeparts namely:
  mantissa E exponent

- Forex:
  9.86E3 imply $9.86*10^3$

3. **Character Constant**

- A symbol enclosed within a pair of single quotes(') is called a character constant.
- Each character is associated with a unique value called an ASCII (American Standard Code for Information Interchange)code.

  Forex: '9', 'a','\n'

4. **String Constant**

- A sequence of characters enclosed within a pair of double quotes(“)is called a string constant.
- The string always ends with NULL character (denoted by \0)character.
- Forex:

  "9" "a" "sri" "\n"

<u>Data in Programming</u>

- Data is collection of raw facts
- Data can be numbers, characters, images, sound
- Programs transform data into meaningful information
- Primarily focus on Integers, Characters and Real Numbers (floating point)
- Ex: Roll Number Integer
- Numbers without decimal place are called as integers
- Example: 9, -8, 1008, -4995

<u>Data Types in C</u>

- The data type defines the type of data stored in a variable and hence inthe memory-location.

Three basic data types in C:

int , float and char

- C supports three classes of datatypes:

1) Primary datatype
   for ex: int, float, char and void
2) Derived datatypes
   For ex: array
3) User defined datatypes For
   ex: structure

Primary data types in C:

1) int

- An int is a keyword, used to define integers.
- Using *int* keyword, the programmer can inform the compiler that the data associated with this keyword should be treated as integer.
- C supports 3 different sizes of integer:
  - Short int
  - int
  - long int

2) float

- A float is a keyword which is used to define floating point numbers.
- Compiler allocates 4 bytes of memory.
- C supports 3 different sizes of float:
  - float
  - double
  - long double
- compiler allocates 8 bytes of memory to the data type **double,** while 16 bytes to **long double**.

3) char

- A **char** is a keyword which allows defining and store a single character.
- Compiler allocates 1 byte of memory.

4) void

- **void** is an empty data type indicates that no value is associated with this data type, and it does not occupy any space in the memory.
- Generally used with functions to indicate that the function does not return any value.

Range of data types for 16-bit processor

Data type	**Bytes**	**Range of data type**
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
float	4 bytes	3.4E-38 to 3.4E38

double	8 bytes	1.7E-308 to1.7E308
long	8 bytes	-223372036854775808 to +9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Qualifiers

- Qualifiers alter the meaning of primary data types to yield a new data type.

Size Qualifiers

- Size qualifiers alter the size of primary datatype.
- The keywords long and short are 2 size qualifiers.For example:

```
long int i; //The size of int is 2 bytes but, when long
                  //keyword is used, variable will be of 4 bytes
short int i; //The size of int is 2 bytes but, when short
                  //keyword is used, that variable will be of 1
                  //byte
```

Sign Qualifiers

- Whether a variable can hold positive value, negative value or both values isspecified by sign qualifiers.
- Keywords signed and unsigned are used for sign qualifiers.

```
unsigned int a; //unsigned variable can hold zero &
                   // positive values only
Signed int b;      //signed variable can hold zero ,positive
                          //and negative values
```

Constant Qualifiers

- Constant qualifiers can be declared with keyword **const**.
- An identifier declared by using a **const** keyword cannot be modified.

```
const int p=20; //the value of p cannot be changed in the
                   //program.
```

Variables

• A variable is an identifier whose value can be changed during execution of the program. Variable is a name given to a memory-location where the data(value) can be stored.

• Using the variable-name, the data can be
→ stored in a memory-location and
→ accessed or manipulated

Variable Declarations:

```
//Showing some variable declarations
#include        <stdio.h> #define   LOWER        0 int   main ()
{
int       lower =       234;
char      c             =       'a',     d,       f;
short     age    ;
float     fahr          =       123.45;
int       start  =              LOWER;
const     double        e       =       2.7049658030;
}
/*Ritchie's advice: Declare each variable type on a    separate line */
```

• Choose variable names that are related to the purpose of the variable, and that are unlikely to get mixed up typographically.

• Use short names for local variables, especially loop indices, and longer names for external variables.

Rules for defining a variable

1) The first character in the variable should be a letter or an underscore.
2) The first character can be followed by letters or digits or underscore.
3) No special symbols are allowed (other than letters, digits and underscore).
4) Length of a variable can be up to a maximum of 31characters.
5) Keywords should not be used as variable-names.

• Valid variables:
area, rate_of_interest, _temperature_, celcius25 Invalid variables:
3fact, ?sum, sum-of-digits, length62$, for, int, if

Declaration of Variable

- The declaration tells the complier
- what is the name of the variable used
- what type of data is held by the variable
- The syntax is shown below:

data_type variable1, variable2, variable3;
where variable1, variable2, variable3 are variable-names of data_type that could be int, float or char type.

- Forex:

```
int a, b, c;
float x, y, z;
```

Initialization of Variable

- The variables are not initialized when they are declared. Hence, variables normallycontain garbage values and hence they have to be initialized with valid data.

- Syntax is shown below:
- Forex:

$$\begin{tikzcd}
\text{int} & \text{xyz} & = & 10; \\
\textbf{data\_type} \arrow[u] & \textbf{variable\_name} \arrow[u] & \textbf{assignment operator} \arrow[u] & \textbf{data (value)} \arrow[u]
\end{tikzcd}$$

float pi=3.1416; char
c='z';

Enum Data Type

- Enumeration: Listing things one after the other
- Variables that possess small range of values
- Represent Integer constant values
- Programmers create their own keyword/type
- Used to make the code readable and easy to maintain
- Two enum names can have same value.
- the compiler by default assigns values starting from 0
- Assign values to some name in any order. All unassigned names get value

- as value of previous name plus one
- value assigned to enum names must be some integral constant, i.e., the value must be in range from minimum
- possible integer value to maximum possible integer value

Suppose we need to assign numbers to days of the week

```
// DECLARATION
        enum WEEK_DAY { SUN, MON, TUE, WED, THU, FRI, SAT};

// INSTANTIATE OBJECT
        enum WEEK_DAY DAY ;
// DAY is variable of type ENUM
        Int main()
        {
                enum WEEK_DAY  DAY;
                DAY=  THU;
                printf("Value of Thursday is %d", DAY);
                return 0;
        }       //      OUTPUT      4
```

Boolean Type Data

In programming TRUE or FALSE is frequently used

```
#define TRUE  1
#define FALSE 0
int test;
test=TRUE;
test=FALSE;


if (test==TRUE)
printf("PASS\n");
if (test==FALSE)
printf("FAIL\n");
```

```
  _Bool test    //_Bool keyword
```

- test takes the values TRUE (1) and FALSE (0)
- _Bool is unsigned integer data type
- Can be assigned values 0 or 1 only.
- Assignment of Non Zero value reverts to 1

- For example test = 3

```
if(test)
printf("TRUE\n");
else
printf("FALSE\n");
```

- bool defined as a macro in <stdbool.h>
- also true and false macros are defined

```
#include        <stdio.h>
#include        <stdbool.h>
//boolasanaliasto     _Bool intmain()
{
      Booltest;
      test=true         ;
      if   (test)
      printf("TRUE     \n");
      else
      printf("FALSE    \n");
}
```

Common Conversion Specifiers

Decimal integer: %d
Octal integer: %o
Hexadecimal: %x
float: %f
double: %lf
char: %c
string: %s
unsigned int : %u

% W.P[Conversion_Specifier]

- W The width specifies the minimum number of characters to be printed
- Width specification is ignored if number of letters is more than the width specified
- P specifies the Precision
- %3d width printing 10 would print b10 (right justified)

- %-3d width printing 10 would print 10b (left justified)

```
//Printing of int and float  using
//various format specifiers and widths #include<stdio.h>

#include<stdlib.h>
int main()
{
        inti=101 ;
        floatj=1234.123f; char
        name[20]="Arjun";
        printf("[%d][%6d][%-6d][%6.4d]\n",i,i, i, i);
        printf("[%f][%12f][%-12f][%12.2f][%12.2e]\n",  j,j,j,j,j);
        printf("[%s][%15s][%-15s][%15.2s]\n", name,name,name,name);

return  0;
}
```

Output of the Code Snippet:

```
[101][   101][101   ][  0101]
[1234.123047][ 1234.123047][1234.123047][ 1234.12][   1.23e+003]
[Arjun][Arjun][Arjun][       Ar]
```

<u>OPERATOR</u>

- An operator can be any symbol like + - * / that specifies what operation need tobe performed on the data.
- Forex:

  + indicates add operation
  * indicates multiplication operation Operand

- An operand can be a constant or a variable.

Expression

An expression is combination of operands and operator that reduces to a single value.

- Forex:

  Consider the following expression a+b here a and b are operands while + is an operator

Operator Table with Precedence and Associativity:

- The order in which different operators are used to evaluate in an expression is called precedence of operators.

Precedence	Operator	Description	Associativity
1	++	Postfix increment	Left-to-right
	--	Postfix decrement	
	()	Function call	
	[]	Array subscripting	
	.	Element selection by reference	
	->	Element selection through pointer	
2	++	Prefix increment	Right-to-left
	--	Prefix decrement	
	+	Unary plus	
	-	Unary minus	
	!	Logical NOT	
	~	Bitwise NOT (One's Complement)	
	(type)	Type cast	

	*	Indirection (dereference)	
	&	Address-of	
	sizeof	sizeof	
3	*	Multiplication	Left-to-right
	/	Division	
	%	Modulo (remainder)	
4	+	Addition	Left-to-right
	-	Subtraction	
5	<<	Bitwise left shift	Left-to-right
	>>	Bitwise right shift	
6	<	Less than	Left-to-right
	<=	Less than or equal to	
	>	Greater than	
	>=	Greater than or equal to	
7	==	Equal to	Left-to-right
	!=	Not equal to	
8	&	Bitwise AND	Left-to-right
9	^	Bitwise XOR	Left-to-right
		(exclusive or)	
10	|	Bitwise OR (inclusive or)	Left-to-right

11	&&	Logical AND	Left-to-right
12	||	Logical OR	Left-to-right
13	?:	Ternary conditional	Right-to-left
14	=	Direct assignment	Right-to-left
	+=	Assignment by sum	
	-=	Assignment by difference	
	*=	Assignment by product	
	/=	Assignment by quotient	
	%=	Assignment by remainder	
	<<=	Assignment by bitwise left shift	
	>>=	Assignment by bitwise right shift	
	&=	Assignment by bitwise AND	
	^=	Assignment by bitwise XOR	
	|=	Assignment by bitwise OR	
15 low est	,	Comma	Left-to-right

Memory Key: **PUMA SRE BIT3 LOG2 TAC**

Illustrations

x = -2 + 11 - 7 * 9 % 6 / 12

Operator	Precedence	Associativity
- (Unary)	**1**	**R-L**
*** , % , /**	**2**	**L-R**
+ , -	**3**	**L-R**
=	**4**	**R-L**

CLASSIFICATION OF OPERATORS

Operator Name	**For Example**
Arithmetic operators	+ - * / %
Increment/decrement operators	++ --
Assignment operators	=
Relational operators	<>== Logical
operators	&& ||~
Conditional operator	?:
Bitwise operators	& |^
Comma operator	,
Special operators	[], sizeof,→

ARITHMETIC OPERATORS

• These operators are used to perform arithmetic operations such asaddition, subtraction, division, multiplication, modulus.

There are 5 arithmetic operators:

Operator	**Meaning of Operator**
+	addition
-	subtraction
*	multiplication
/	division
%	modulos

• Division symbol(/)

→ divides the first operand by second operand and

→ returns the quotient.

Quotient is the result obtained after division operation.

- Modulus symbol (%)

→ divides the first operand by second operand and
→ returns the remainder.

Remainder is the result obtained after modulus operation.

- To perform modulus operation, both operands must be integers.
- Program to demonstrate the working of arithmetic operators.

```
#include<stdio.h>
int main()
{
  int a=11,b=4,c;
  c=a+b;
  printf("a+b=%d \n", c);
  c=a-b;
  printf("a-b=%d     \n",c);
  c=a*b;
  printf("a*b=%d     \n",c);
  c=a/b;
  printf("a/b=%d \n", c);
  c=a%b;
  printf("Remainder when a is divided by b=%d ", c);
  return1;
}
```

Output:

```
a+b=15
a-b=7
a*b=44
a/b=2
a%b=3 Remainder when a is divided by b= 3
```

Integer vs Float Arithmetic

```
int a=20,b=6,result;
    result   =   a   /   b   ;    //3

    result   =   a   %   b   ;    //2
```

```
Float result;

  result=10 /3; //   3.00

  result=10/3;   //   3.333333
```

NOTE:

a) Unary+,-have highest precedence, nextare*,/,%,operators and last are binary+,
-operators and the associativity of allarithmetic operators is left to right.

b) Mod works only on Integer type data

INCREMENT OPERATOR

• ++ is an increment operator.

• As the name indicates, increment means increase, i.e. this operator is used toincrease the value of a variable by1.

• For example:

  If b=5

  then b++ or ++b; // b becomes 6

• The increment operator is classified into 2categories:

1) Post increment Ex:b++

2) Pre increment Ex:++b

• As the name indicates, post-increment means first uses the value of variableand then increases the value of variable by1.

• As the name indicates, pre-increment means first increases the value of variableby 1 and then uses the updated value of the variable.

• Forex:

  If x= 10,

  then z= x++; assigns the value 10 to z & then increments value of x but z = ++x; assigns the value 11 to z

Example: Program to illustrate the use of increment operators. #include<stdio.h>

```
int main()
{
        int x=10,y = 10, z ; z= x++;
        printf(“ z=%d x= %d\n”, z, x);
        z = ++y+x;
        printf(“ z=%d  y=  %d”,  z,  y++);
        return 1;
}
```

Output:

```
z=10    x=11
z=22 y=11
```

DECREMENT OPERATOR

- -- is a decrement operator.
- As the name indicates, decrement means decrease, i.e. this operator is used todecrease the value of a variable by1.
- For example:
  If b=5
      then b-- or --b; // b becomes 4

- Similar to increment operator, the decrement operator is classified into two categories:
  - i) Post decrement Ex:b--
  - ii) Pre decrement Ex:--b

- Forex:

If x=10,
Then z=x--;   // z becomes 10,
but z = --x; // z becomes9

Example: Program to illustrate the use of decrement operators.

```
int main()
{
        int x=10,y = 10, z ; z= x--;
        printf(" z=%d x= %d\n", z, x);
        z = --y;
        printf(" z=%d y= %d", z, y);
}
```

Output:

```
z=10 x=9
z=9 y=9
```

TYPE CONVERSION

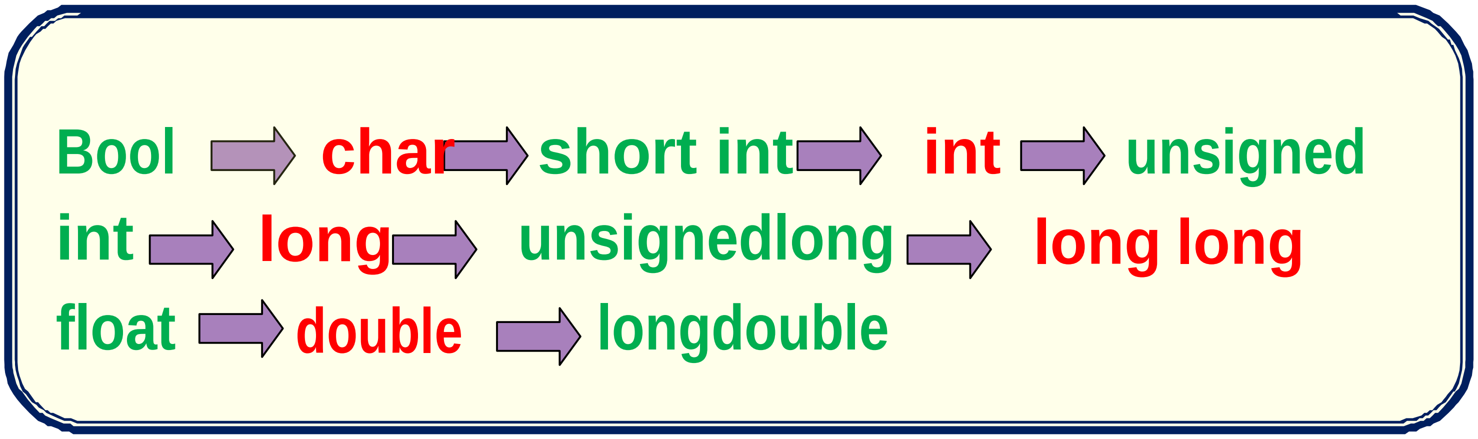
- Type conversion is used to convert data of one type to data ofanother type.
- Type conversion is of 2 types as shown in belowfigure:

Implicit Type Conversion

- If a compiler converts one type of data into another type of data automatically, it is known as implicit conversion.
- There may be data loss whenever in implicit conversion takes place i.e while converting from **float** to **int** the fractional part will be truncated, **double** to **float** causes rounding of digit and **long** to **int** causes dropping of excess higher order bits.
- The conversion always takes place from lower rank to higher rank.

```
int main()//    Implicit        Conversion
{
  intage=2;
  char gender ='A';   //ASCII65
 age=   age*   gender;
 printf("%d    \n",    age); //130
  }        //Lower to a WIDER Type
```

Any implicit type conversions are made by converting the lower type to higher type as shown below:



For ex, int to float as shown in the above datatype hierarchy.

- Forex:

```
int   a   =   22;
float b=11;
float c = a/b; =0.500000
```

Rules (Promote lower to wider type):

- _Bool to character
- character to short int
- short to int
- signed to unsigned int
- unsigned int to long int
- long to unsigned long int
- unsigned long int to long long
- long long to float
- float to double

Type Casting(Explicit Data Type Conversion)

- When the data of one type is converted explicitly to another type with the help of some pre-defined types, it is called as explicit conversion.
- The syntax is shown below:

```
data_type1v1;

data_type2 v2= (data_type2) v1;
```

where v1 can be expression or operand or value

```
#include<stdio.h>
int main()
{
inti=      10,j= 20;
float result;
result   =i/j;
printf("%0.2f\n",      result);//      0.00
  result =(float)i/j;
  printf("%0.2f         \n",    result);//      0.5
  }
```

RELATIONAL OPERATORS

- Relational operators are used to find the relationship between two operands.
- The output of relational expression is either true(1) or false(0).
- For example

a>b //If a is greater than b, then a>b returns 1 else a>b returns 0.

- The 2 operands may be constants, variables or expressions.

There are 6 relational operators:

Operator	**Meaning of Operator**	**Example**
>	Greaterthan7>4	returns true (1)
<	Lessthan7<4	returns false(0)
>=	Greater than or equal to 7>=4	returns true (1)
<=	Less than or equal to7<=4	return false(0)
== Equal to	7==4	returns false(0)
!= Not equal to	7!=4	returns true(1)

- Forex:

Condition	**Return values**
8> 7	1 (or true)
8<7	0 (or false)
8+7<15	0 (or false)

- Example: Program to illustrate the use of all relational operators.

```
#include<stdio.h>
int main()
{    printf("7>4    :    %d    \n",    7>4);
        printf("7>=4  :  %d  \n",  7>=4);
        printf("7<4   :   %d   \n",   7<4);
        printf("7<=4  :  %d  \n",  7<=4);
        printf("7==4  :  %d  \n",  7==4);
        printf("7!=4   :   %d   ",   7!=4);
        return1;

}
```

Output:

```
7>4 : 1
7>=4 : 1
7<4 : 0
7<=4 :0
7==4 :0
7!=4 : 1
```

LOGICAL OPERATORS

- These operators are used to perform logical operations like negation,conjunction and disjunction.

- The output of logical expression is either true(1) or false(0).
- There are 3 logical operators:

Operator Meaning with Examples

&& Logical AND If c=3 and d=1 then ((c==3) && (d>2)) returns false.

|| Logical OR If c=3 and d=1 then ((c==3) || (d>3)) returns true.

! Logical NOT If c=3 then !(c==3) returns false.

- All non zero values(i.e. any value >0 or < 0) will be treated as true, while zerovalue(i.e. 0) will be treated as false.

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

- Example: Program to illustrate the use of all logical operators.

```
#include<stdio.h>
int main()
{
        printf("5  &&  0  :  %d  \n",  5  &&0  );
        printf("5  ||  0  :  %d  \n",  5  ||  0  );
        printf(" !0 : %d", !0 );
        return 1;

}
```

Output:

```
5 &&0 : 0
5 || 0 : 1
!0 : 1
```

- Example: Program to illustrate the use of both relational &Logical operators.

```
#include<stdio.h>    int
main()
{
        printf("7>5 && 5< 8 : %d \n", 7>5 &&5<8);
        printf(" 7<5 || 5!=5 : % d \n", 7<5 || 5!=5);
```

```
          printf("!(3 ==3) : %d ", !(3==3));
          return 1;
     }
```
Output:

```
7>5 && 5<8 :1
7<5 || 5!=5 :0
!(3 ==3) : 0
```

Equality Operators (== !=)

```
int       main ()
{
Int  a=20,b=10;
int result       ;
if(a== 20)
printf("Yes a is 20\n");
if(b!=20)
printf("Yes b is not 20\n");
}
```

TERNARY OPERATOR (CONDITIONAL OPERATOR)

- The conditional operator is also called a ternary operator it has three parts.
- Conditional operators are used for decision making in C.
- The syntax is shown below:

(exp1)? exp2: exp3;

where exp1 is an expression evaluated to true or false;
If exp1 is evaluated to true, exp2 is executed; Ifexp1
is evaluated to false, exp3 is executed.

Example: Program to find biggest of 2 numbers using conditional operator.

```
#include<stdio.h>
int main()
{
        int a,b, big ;
        printf("Enter   two   different   numbers:\n");
        scanf("%d%d", &a, &b);
        big=(a>b)? a :b;
        printf(" Biggest number is ", big);
        return 1;
}
```

Output:

```
Enter two different numbers: 88 79
Biggest number is 88
```

Example: Program to find biggest of 3 numbers using conditional operator.

```
#include<stdio.h>
int main()
{
        int a,b,c big ;
        printf("Enter two different numbers:\n");
        scanf("%d%d%d", &a, &b, &c); big=(a>b)?
        ((a>c)?:a:c) : (b>c)?b:c; printf(" Biggest of three
        number is ", big); return1;
}
```

Output:

```
Enter two different numbers: 88 99
Biggest number is 99
```

ASSIGNMENT OPERATOR

- The most common assignment operator is=.
- This operator assigns the value in right side to the left side.
- The syntax is shown below:

variable=expression;

- Forex:

c=5; //5 is assigned to c

b=c; //value of c is assigned to b 5=c; //

Error! 5 is a constant.

- The operators such as +=,*= are called shorthand assignment operators.
- Forex,

a=a+10: can be written as a+=10;

- In the same way, we have:

Operator Example

a-= a-=b a=a-b a*=

b c*=b a=a*b a/=

a/=b a=a/b

a%=b a%=b a=a%b

NOTE:

```
int i= 2, j = 3; i *= j + 6 ;
i = i * (j + 6);
```

Compact Assignments

=	Direct assignment
+=	Assignment by sum
-=	Assignment by difference
*=	Assignment by product
/=	Assignment by quotient
%=	Assignment by remainder
<<=	Assignment by bitwise left shift
>>=	Assignment by bitwise right shift
&=	Assignment by bitwise AND
^=	Assignment by bitwise XOR
|=	Assignment by bitwise OR

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

BITWISE OPERATORS

- These operators are used to perform logical operation (and, or, not)on individual bits of a binary number.
- There are 6 bitwise operators:

Operators	**Meaning of operators**
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Truth Table

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

Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)



Bit Operation of 12 and 25

  00001100

& 00011001

  ________
00001000  = 8 (In decimal)
```

Example #1: Bitwise AND

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
printf("Output = %d", a&b);
    return 0;
}
```

Output

```
    12 = 00001100 (In Binary)

& 25 = 00011001 (In Binary)
```

```
=================
            00001000 (8 in Binary)
Output = 8
```

Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

Example #2: Bitwise OR

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
printf("Output = %d", a|b);
    return 0;
}
```

Output

```
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)



Bitwise OR Operation of 12 and 25

  00001100

| 00011001

  ________

00011101  = 29 (In decimal)


Output = 29
```

Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

Example #3: Bitwise XOR

```
#include <stdio.h>
int main()
{
    int a = 12, b = 25;
printf("Output = %d", a^b);
    return 0;
}
```

Output

```
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

  00001100

^ 00011001


  ________

00010101  = 21 (In decimal)

Output = 21
```

Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

```
35 = 00100011 (In Binary)



Bitwise complement Operation of 35

~ 00100011


  ________

11011100  = 220 (In decimal)
```

Twist in bitwise complement operator in C Programming

The bitwise complement of 35 (~35) is -36 instead of 220, but why?

For any integer `n`, bitwise complement of `n` will be `-(n+1)`. To understand this, you should have the knowledge of 2's complement.

2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

```
 Decimal         Binary           2's complement

   0            00000000           -(11111111+1) = -00000000 = -0(decimal)

   1            00000001           -(11111110+1) = -11111111 = -256(decimal)

   12            00001100           -(11110011+1) = -11110100 = -244(decimal)

   220            11011100           -(00100011+1) = -00100100 = -36(decimal)


Note: Overflow is ignored while computing 2's complement.
```

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise complement of any number N is -(N+1). Here's how:

```
bitwise complement of N = ~N (represented in 2's complement form)

2'complement of ~N= -(~(~N)+1) = -(N+1)
```

Example #4: Bitwise complement

```
#include <stdio.h>
int main()
{
printf("Output = %d\n",~35);
printf("Output = %d\n",~-12);
    return 0;
}
```

Output

```
Output = -36
Output = 11
```

Shift Operators in C programming

There are two shift operators in C programming:

- Right shift operator
- Left shift operator.

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

```
212 = 11010100 (In binary)

212>>2 = 00110101 (In binary) [Right shift by two bits]

212>>7 = 00000001 (In binary)

212>>8 = 00000000

212>>0 = 11010100 (No Shift)
```

Left Shift Operator

Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is <<.

```
212 = 11010100 (In binary)

212<<1 = 110101000 (In binary) [Left shift by one bit]

212<<0 = 11010100 (Shift by 0)

212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Example #5: Shift Operators

```
#include <stdio.h>
int main()
{
    int num=212, i=1;

   printf("Right shift by %d: %d\n", i, num>>i);

    printf("\n");

    printf("Left shift by %d: %d\n", i, num<<i);

     return 0;
}
```

```
Right Shift by 1: 106

Left Shift by 1:   424
```

```
  Int main()
  {
  int x= 2, y=3, z=-8;
  x=x<< y;
printf("%d\n",x);       //16
  x=x>> 2;
  printf("%d\n",x);    //       4
  z=z>> 2;
  printf("%d\n",z);    //-2    (?)
  }


  int main()     // ~Complement Operator
  {
  unsigned short x= 1;
  unsigned short i;
  i=~x;
  printf("%d\n",i);     //       65534
  int result;    //      signed
  result =~0 ;
  printf("%d\n",result);     //-1
  }


//&, ^   ,|      bitwise       operators

int main()
{

  short i=       2 ,j=3;
  printf("%d\n",      i        &       j);      //      2
  printf("%d\n",      i        ^       j);      //      1
  printf("%d\n",      i        |       j);      //      3
}
```

The sizeof operator

The **sizeof** is a unary operator that returns the size of data (constants, variables, array, structure, etc).

```
#include <stdio.h>
int main()
{
   int a;
   float b;
   double c;
   char d;
printf("Size of int=%lu bytes\n",sizeof(a));
printf("Size of float=%lu bytes\n",sizeof(b));
printf("Size of double=%lu bytes\n",sizeof(c));
printf("Size of char=%lu byte \n",sizeof(d));
printf("Size of char in bits=%d\n",sizeof(d)*8);

   return 0;
}
```

Output

```
Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte
Size of char in bits= 8
```

Comma Operator:

The **comma operator** (represented by the token ,) is a binary **operator** that evaluates its first operand and discards the result, it then evaluates the second operand and returns the value (and type). The **comma operator** has the lowest precedence of any **C operator**.

```
intx,          y;
x=(10,20,30);
printf("x=%d",x);
x=33, y=22;
printf("\nValue: %d\n",(x,(y+10)));
```

Output: x=30

Value: 22

ASCII (American Standard Code for Information Interchange)

- Unique to each Programming Language
- Characters are represented as 0s and1s i.e. in binary numbers
- Each character is mapped to a number
- Different mapping schemes exist like Extended Binary Coded Decimal Interchange Code(EBCDIC, ASCII)
- Uppercase Characters are from 'A' (65) to'Z' (90)
- Lowercase from 'a' (97) to 'z' (122)
- Digits '0' (48) to '9' (57)
- Example AB is stored as 6566 inconsecutive memory locations

```
// C program to print ASCII Value of
Characterint main()
{
 char c = 'A';
printf("The ASCII value of %c is%d", c, c);
}
```

Output

```
The ASCII value of A is 65
```

For example, the ASCII value of 'A' is65.

What this means is that, if you assign 'A' to a character variable, 65 is stored in thevariable rather than 'A' itself.