## Introduction to C Language:

C language was developed Dennis Ritchie in 1972 at AT&T Bell Laboratories (USA). C language was derived from B language which was developed by Ken Thomson in 1970. This B language was adopted from a language BCPL (Basic Combined Programming Language), which was developed by Martin Richards at Cambridge University. The language B named as so by borrowing the first initial from BCPL language. Dennis Ritchie modified and improved B language and named it as C language, using second initial from BCPL.

C language is not high level language and not a low level language. It is a middle level language with high level and low level features.

C is a general purpose, structured programming language. C language is one of the most popular computer languages today because it is a structured, high level, machine independent language. It allows software developers to develop software without worrying about the hardware platforms where they will be implemented.

**Importance of C**

It is a robust language whose rich set of built in functions and operators can be used to write any complex program. The C compiler combines the capabilities of an assembly language with the features of a higher level language and therefore it is well suited for both system software and business packages.

**Features**

- **C is a general purpose language**
- **C can be used for system programming as well as application programming.**
- **C is middle level language**
- **C is portable language**
- **An application program written on C language will work on many different computers with little or no modifications.**
- **C Block structured language. In C , a block is marked by two curly braces({ })**
- **C programs are compact, fast and efficient**
- **C is case sensitive language**
- **C is function oriented language**

- **C language includes advanced data types like pointers, structures, unions, enumerated types etc.**
- **C language supports recursion and dynamic storage allocation**
- **C can manipulate with bits, bytes and addresses**
- **Parameter passing can be done using call-by-value and call-by-reference**

**Why Use c?**

1. ***Powerful and flexible***
   C is a powerful and flexible language. The language itself places no constraints on you. C is used for projects as diverse as operating systems, Word Processors, graphics, spreadsheets, and even compilers for other languages.
2. ***Popular***
   C is a popular language preferred by professional programmers. As a result, a wide variety of C compilers and helpful accessories are available for use by programmers.
3. ***Portable***
   C is a portable language. Portable means that a C program written for one computer system (say an IBM PC) can be compiled and run on another system (say a DEC VAX system) with little or no modification. Portability is enhanced by the ANSI standard for C, the set of rules for C compilers.
4. ***Minimum keywords***
   C is a language of few words, containing only a handful of terms, called keywords, which serve as the base on which the language's functionality is built. There is a misconception with many that a language with more key words (sometimes called reserved words) would be more powerful. This isn't true.
5. ***Modular***
   C is a modular. C code can (and should) be written in routines called functions. These functions can be reused in other applications or programs. By passing pieces of information to the functions, you can create useful, reusable code.

**Execution Characters**

The meaning of these characters is interpreted at the time of execution. These are also known as “Escape sequences because the backslash (\) is considered as an ‘escape’ character. It causes an escape from normal interpretation of a string. so that the next character is recognized as one having special meaning.

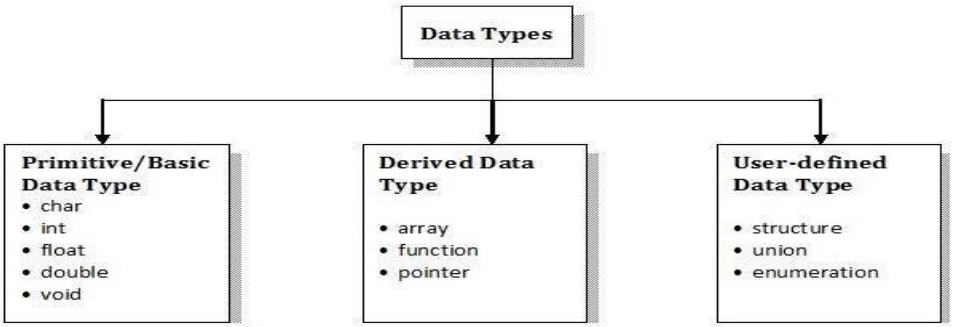
**Escape sequences**

| Escape sequence | Meaning | Result |
|---|---|---|
| **\b** | Backspace | Moves the cursor to the previous position of the current line. |
| **\a** | Bell(alert) | Produces a beep sound for alert |

| | | |
|---|---|---|
| **\r** | Carriage return | Moves the cursor to beginning of the current line. |
| **\n** | New line | Moves the cursor to beginning of the next line. |
| **\0** | Null | Null character |
| **\v** | Vertical tab | Moves the cursor to next vertical tab position. |
| **\t** | Horizontal tab | Moves the cursor to next horizontal tab position. |
| **** | Backslash | Present a character with backslash(\) |

**Data Types of C**

C supports different types of data. The type or Data type refers a type of data that is going to be stored in variable. Data types can be broadly classified as shown in figure




C provides a standard minimal set of data types. Sometimes these are also called as 'Primitive types'. They are:

- '***char***' **is used to store any single character.**
- **'*int*' is used to store integer value.**
- **'*float*' is used to store floating point value and**
- **'*double*' is used for storing long range of floating point number.**

**Type Qualifiers**

In addition, C has four type qualifiers, also known as type modifiers which precede the basic data type. A type modifier alters the meaning of basic data type to yield a new data type. They are as follows:

- **short**
- **long ( to increase the size of an *int* or *double* type)**
- **signed**
- **unsigned**

The qualifiers can be classified into two types:

- Size qualifier (e.g., *short* and *long*)
- Sign qualifier (e.g., *signed* and *unsigned*)

**Size qualifiers:** alters the size of basic data type. The keywords long and short are two size qualifiers. For example:

***long int i;***

The size of int is 2 bytes but, when long keyword is used, that variable will be either 4 bytes or 8 bytes.

**Sign qualifiers:** Whether a variable can hold only positive value or both values is specified by sign qualifiers. Keywords *signed* and *unsigned* are used for sign qualifiers.

- Each of these type modifiers can be applied to base type *int.*
- The modifiers *signed* and *unsigned* can also be applied to base type *char*.
- In addition *long* can also be applied to *double.*
- When base type is omitted from a declaration, *int* is assumed.

**Example**

```
short int a; // 16 bits, range -32,768 to 32,767
unsigned short int b; // 16 bits, range 0 to 65,535
unsigned int c; // 32 bits, range 0 to 4,294,967,295
```

| Data Type | Size (in Bytes) | Range |
|---|---|---|
| char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| signed char | 1 | -128 to 127 |
| int | 2 | -32768 to 32767 |
| unsigned int | 2 | 0 to 65535 |
| signed int | 2 | -32768 to 32767 |
| short int | 2 | -32768 to 32767 |
| unsigned short int | 2 | 0 to 65535 |
| signed short int | 2 | -32768 to 32767 |
| long int | 4 | -2147483648 to 2147483647 |
| unsigned long int | 4 | 0 to 4294967295 |
| signed long int | 4 | -2147483648 to 2147483647 |
| float | 4 | 3.4E-38 to 3.4E +38 |
| double | 8 | 1.7E-308 to 1.7E+308 |
| long double | 10 | 3.4E-4932 to 1.1E+4932 |

***void* type**

This type holds no value and thus valueless. It is primarily used in three cases:

- To specify the return type of a function (when the function returns no value)
- To specify the parameters of the function (when the function accepts no arguments from the caller.)
- To create generic pointers.

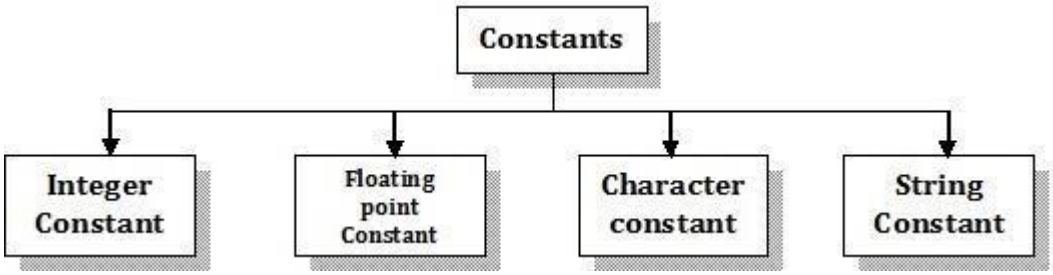
## 2.3 Various kinds of C Data(Tokens)

The data in C language can be divided into

- Constants/literals
- Variables/Identifiers
- Reserved Words/Key words
- Delimiters

### Constants

Constant can be defined as a value that can be stored in memory and cannot be changed during execution of the program. Constants are used to define fixed values like *pi*. C has four basic types of constants. They are:




### Integer Constant

An integer constant must have at least one digit and should not have a decimal point. It can be either positive or negative.

Examples for integer constants

1 9 234 999

### Floating point Constant

A floating point constant is decimal number that contains either a decimal point or an exponent. In the other words, they can be written in 2 forms: fractional and exponential.

When expressed in fractional form, note the following points.

- There should be at least one digit, and could be either positive or negative value. A decimal point is must.
- There should be no commas or blanks.

Examples for fractional form

12.33 -19.56 +123.89 -0.7

When expressed in exponential form, a floating point constant will have 2 parts. One is before *e* and after it. The part which appears before *e* is known as *mantissa* and the one which follows is known as *exponent.* When expressed in this format, note the following points.

- *mantissa* and *exponential* should be separated by letter *E*.
- *mantissa* can have a positive and negative sign.
- default is positive.

**Examples for fractional form**

2E-10 0.5e2 1.2E+3 -5.6E-2

**Character Constant**

These are single character enclosed in single quotes. A character can be single alphabet, single digit, single special symbol enclosed with in single quotes. Not more than a single character is allowed.

**Example**

'a' 'Z' '5' '$'

**String Constant**

A String constant is sequence of characters enclosed in double quotes. So *"a" is* not same as *'a'.* The characters comprising the string constant are stored in successive memory locations.

**Example: " Hello World"**

**Variables/Identifiers**

Variable is named memory location that can be used to store values. These variables can take different values but one value at a time. These values can be changed during execution of a program.

Identifiers are basically names given to program elements such as variables, arrays and functions.

**Rules for naming a Variable/Identifier**

1. The only characters allowed are alphabets, digits and underscore (_).
2. They must start with an alphabet or an underscore (_).
3. It can not include any special characters or punctuation marks (like #, $, ^, ?, ., etc.) except underscore (_).
4. They can be of any reasonable length. They should not contain more than 31 characters. But it is preferable to use 8 characters.
5. There cannot be two successive underscores.
6. Reserved words/keywords cannot be identifiers.
7. Lower case or uppercase letters are significant.

**Key words**

- **These are also called as reserved words.**
- **All Keywords have fixed meanings and these meanings cannot be changed.**
- **There are 32 keywords in C programming.**
- **Keywords serve as basic building blocks for a program statement.**
- **All keywords must be written in lowercase only.**

**ANSI C Keywords**

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

**Delimiters**

Delimiters are used for syntactic meaning in C. These are given below:

| Symbol | Name | Meaning |
|---|---|---|
| : | Colon | Used for label |
| ; | Semicolon | End of statement |
| ( ) | Parenthesis | Used in expression |
| [ ] | Square brackets | Used in expression |
| { } | Curly braces | Used for block of statements |
| # | Hash | Pre-processor directive |
| , | Comma | Variable separator |

## 2.4 Variables:

A variable is named memory location that stores a value. When using a variable, we actually refer to address of the memory where the data is stored. C language supports two basic kinds of variables:

- Numeric variables
- Character variables

**Numeric variables**

Numeric variables can be used to store either integer values or floating point values. While an integer value is a whole number without a fraction part or decimal point. A floating point value can have a decimal point.

Numeric values may be associated with modifiers like short, long, signed, and unsigned.

**Character variables:**

Character variables can include any letter from the alphabet or from ASCII chart and numbers 0-9 that are given within single quotes.

**Variable declaration:**

To declare a variable, specify the data type of the variable followed by its name. The data type indicates the kind of data that the variable will store. Variable names should

always be meaningful and must reflect the purpose of their usage in the program. In C, variable declaration always ends with a semicolon, for example:

***char grade;***
***int emp_no;***
***float salary;***
***double bal_amount***
***unsigned short int acc_no;***

C allows multiple variable of same type to be declared in one statement, so the following statement is absolutely legal in C

***float temp_in_deg, temp_in_farh;***

**Initializing variables**

Assigning value to variable is called variable initialization. For example:

***char grade= 'A';***
***int emp_no=1007;***
***float salary=8750.25;***
***double bal_amount=100000000***

***When variables are declared but not initialized they usually contain garbage values.***

Basically variable declaration can be done in three different places in a C program.

1) Inside main() function are called *local* variables.
2) Outside main() function are called *global* variables.
3) In the function definition header are called *formal* parameters.

**Declaring Constants**

To declare a constant, precede the normal variable declaration with *const* keyword and assign it a value. For example,

*const float pi=3.1415;*

This *const* keyword specifies that the value of cannot change.

## C Input Functions:

C programming language provides built-in functions to perform input operations. The input operations are used to read user values (input) from the keyboard. The c programming language provides the following built-in input functions.

1. **scanf()**
2. **getchar()**
3. **getch()**
4. **gets()**

**1.scanf():**

The scanf() function is used to read multiple data values of different data types from the keyboard. The scanf() function is built-in function defined in a header file called "**stdio.h**".

### Example Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i;
  printf("\nEnter any integer value: ");
  scanf("%d",&i);
  printf("\nYou have entered %d number",i);
}
```

### Example Program:

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

void main(){
  int i;
  float x;
  printf("\nEnter one integer followed by one float value : ");
  scanf("%d%f",&i, &x);
  printf("\ninteger = %d, float = %f",i, x);

}
```

## 2.getchar():

The getchar() function is used to read a character from the keyboard and return it to the program. This function is used to read a single character.

### Example Program

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

```
void main(){
  char ch;
  printf("\nEnter any character : ");
  ch = getchar();
  printf("\nYou have entered : %c\n",ch);
}
```

### 3.getch():

The getch() function is similar to getchar function. The getch() function is used to read a character from the keyboard and return it to the program

### Example Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
  char ch;
  printf("\nEnter any character : ");
  ch = getch();
  printf("\nYou have entered : %c",ch);

}
```

### 4.gets():

The gets() function is used to read a line of string and stores it into a character array. The gets() function reads a line of string or sequence of characters till a newline symbol enters.

### Example Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
  char name[30];
  printf("\nEnter your favourite website: ");
  gets(name);
  printf("%s",name);
}
```

## C Output Functions:

C programming language provides built-in functions to perform output operation. The output operations are used to display data on user screen (output screen) or printer or any file.

1. **printf()**
2. **putchar()**
3. **puts()**

### 1.printf():

The printf() function is used to print string or data values or a combination of string and data values on the output screen (User screen). The printf() function is built-in function defined in a header file called "**stdio.h**".

### Example Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
  int i = 10;
  float x = 5.5;
  printf("%d %f",i, x);
}
```

### 2.Putchar():

The putchar() function is used to display a single character on the output screen. The putchar() functions prints the character which is passed as a parameter to it and returns the same character as a return value. This function is used to print only a single character.

### Example Program:

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

void main(){
  char ch = 'A';
  putchar(ch);
}
```

### 3.Puts():

The puts() function is used to display a string on the output screen. The puts() functions prints a string or sequence of characters till the newline.

### Example Program:

```
#include<stdio.h>
#include<conio.h>
void main(){
  char name[30];
  printf("\nEnter your favourite website: ");
  gets(name);
  puts(name);
}
```

## C Storage Classes:

In C programming language, storage classes are used to define things like **storage location** (whether RAM or REGISTER), **scope**, **lifetime** and the **default value** of a variable.

In the C programming language, the memory of variables is allocated either in computer memory (RAM) or CPU Registers. The allocation of memory depends on storage classes

1. **auto storage class**
2. **extern storage class**
3. **static storage class**
4. **register storage class**

### 1.auto storage class:

The default storage class of all local variables (variables declared inside block or function) is auto storage class. Variable of auto storage class has the following properties...

| Property | Description |
|---|---|
| Keyword | auto |

| Property | Description |
|---|---|
| Storage | Computer Memory (RAM) |
| Default Value | Garbage Value |
| Scope | Local to the block in which the variable is defined |
| Life time | Till the control remains within the block in which variable is defined |

**Example Program 1:**

```
#include<stdio.h>
#include<conio.h>
int main(){
   int i;
   auto char c;
   float f;
   printf("i = %d\tc = %c\tf = %f",i,c,f);
   return 0;
}
```

### 2. extern storage class:

The default storage class of all global varibles (variables declared outside function) is external storage class. Variable of external storage class has the following properties...

| Property | Description |
|---|---|
| Keyword | extern |
| Storage | Computer Memory (RAM) |

| Property | Description |
|---|---|
| Default Value | Zero |
| Scope | Global to the program (i.e., Throughout the program) |
| Life time | As long as the program's execution does not comes to end |

**Example Program 1;**

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

int i;    //By default it is extern variable
int main(){
    printf("%d",i);
    return 0;
}
```

### 3.static storage class:

The static storage class is used to create variables that hold value beyond its scope until the end of the program. The static variable allows to initialize only once and can be modified any number of times. Variable of static storage class has the following properties...

| Property | Description |
|---|---|
| Keyword | static |
| Storage | Computer Memory (RAM) |
| Default Value | Zero |
| Scope | Local to the block in which the variable is defined |

| Property | Description |
|---|---|
| Life time | The value of the persists between different function calls (i.e., Initialization is done only once) |

**Example Program 1:**

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

static int a;
int main(){
    printf("%d",a);
    return 0;
}
```

### 4.register storage class:

The register storage class is used to specify the memory of the variable that has to be allocated in CPU Registers. The memory of the register variable is allocated in CPU Register but not in Computer memory (RAM). The register variables enable faster accessibility compared to other storage class variables. As the number of registers inside the CPU is very less we can use very less number of register variables. Variable of register storage class has the following properties...

| Property | Description |
|---|---|
| Keyword | register |
| Storage | CPU Register |
| Default Value | Garbage Value |
| Scope | Local to the block in which the variable is defined |

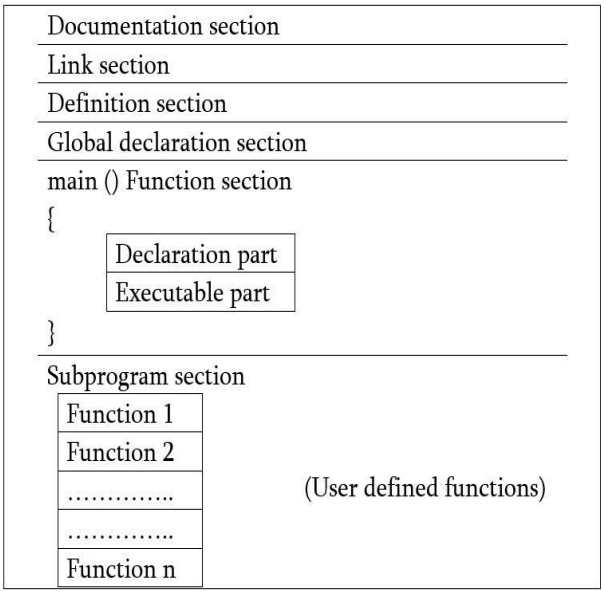
| Property | Description |
|---|---|
| Life time | Till the control remains within the block in which variable is defined |

**Example Program 1:**

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

int main(){
    register int a,b;
    scanf("%d%d",&a,&b);
    printf("%d  %d",a,b);
}
```

**Structure of a C Program**

C program is a collection of one or more functions. Every function is collection of statements, performs a specific task. The structure of a basic C program is:




**Documentation section**

The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later. There are two types of comment lines.

1. Block comment lines /* */ and
2. Single line comment lines. //

**Link section**

The link section provides instructions to the compiler to link functions from the system library.

**#include directive**

The *#include* directive instructs the compiler to include the contents of the file "stdio.h" (standard input output header file) enclosed within angular brackets.

**Definition section**

The definition section defines all symbolic constants. #define PI 3.1413

**Global declaration section**

There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.

**main() function section**

Every C program must have one main function section. This section contains two parts:

1. Declaration part and
2. Executable part

**Declaration part**: It declares all the variables used in the executable part.

**Executable part:** There is at least one statement in the executable part.

These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.

**Subprogram section**

The subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.

**For Example a 'C' program that prints welcome message:**

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

**Output**

Welcome to C Programming

**Expressions**

In C programming, an expression is any legal combination of operators and operands that evaluated to produce a value.Every expression consists of at least one *operand* and can have

one or more *operators*. Operands are either variables or values, whereas operators are symbols that represent particular actions.
In the expression x + 5; x and 5 are operands, and + is an operator.

In C programming, there are mainly two types of expressions are available. They are as follows:

1. Simple expression
2. Complex expression

**Simple expression:** In which contains one operator and two operands or constants.

Example: x+y; 3+5; a*b; x-y etc.

**Complex expression:** In which contains two or more operators and operands or constants.

Example: x+y-z; a+b-c*d; 2+5-3*4; x=6-4+5*2 etc.

Operators provided mainly two types of properties. They are as follows:

1. Precedence and
2. Associativity

**Operator Precedence**

It definesthe order in which operators in an expression are evaluated depends on their relative precedence. Example:

Let us see x=2+2*2
1st pass- 2+**2*2**
2nd pass- **2+4**
3rd pass- 6 that is x=6.

**Associativity** defines the order in which operators with the same order of precedence are evaluated.
Let us see x=2/2*2
1st pass -- **2/2***2
2nd pass - **1*2**
3rd pass - **2**that is x=2

The below table lists C operators in order of *precedence* (highest to lowest) and their *associativity* indicates in what order operators of equal precedence in an expression are applied.

| S.No | Operators | Associativity |
|---|---|---|
| 1 | **( ) [ ] → . ++ (postfix) − −(postfix)** | L to R |

| | | |
|---|---|---|
| 2 | **++ (prefix) -- (prefix) ! ~ sizeof(type) + (unary) − (unary) &(address) * (indirection)** | R to L |
| 3 | *** / %** | L to R |
| 4 | + − | L to R |
| 5 | <<>> | L to R |
| 6 | <<= >>= | L to R |
| 7 | = = **!=** | L to R |
| 8 | **&** | L to R |
| 9 | ^ | L to R |
| 10 | \| | L to R |
| 11 | **&&** | L to R |
| 12 | \|\| | L to R |
| 13 | **? :** | R to L |
| 14 | = += −= *= /= **%**= >>= <<= **&**= ^= \|= | R to L |
| 15 | **,** | L to R |

**Example Program:**

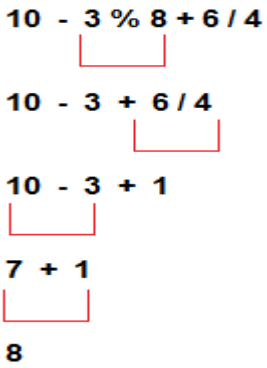
```
#include<stdio.h>
int main()
{
int a, b=4,c=8,d=2,e=4,f=2;

a=b+c/d+e*f;                    // expression1

printf(" The value of a is%d\n", a);

a=(b+c)/d+e*f;                  // expression2

printf("The value of a is%d\n", a);

a=b+c/((d+e)*f);                // expression3
printf("The value of a is%d\n", a);
return 0;

}
```

**Output:**

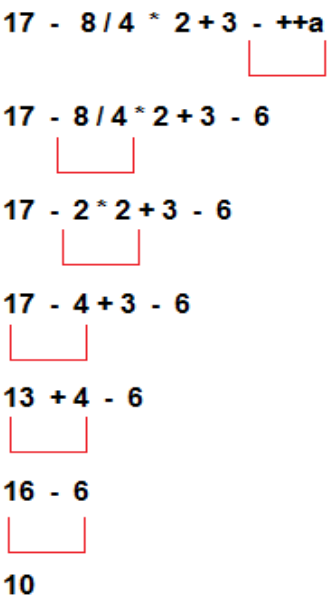
The value of *a* is 16
The value of *a* is 14
The value of *a* is 4

**Evaluating Expressions:**

**Example 1:**




**Example 2:**




**Type Conversion**

**Type casting:**Type casting is a way to convert a variable from one data type to another data type. It can be of two types:

1. Implicit type casting
2. Explicit type casting

**Implicit type casting**

When the type conversion is performed automatically by the compiler without programmer's intervention, such type of conversion is known as **implicit type conversion** or **type promotion**. In this, all the lower data types are converted to its next higher data type.

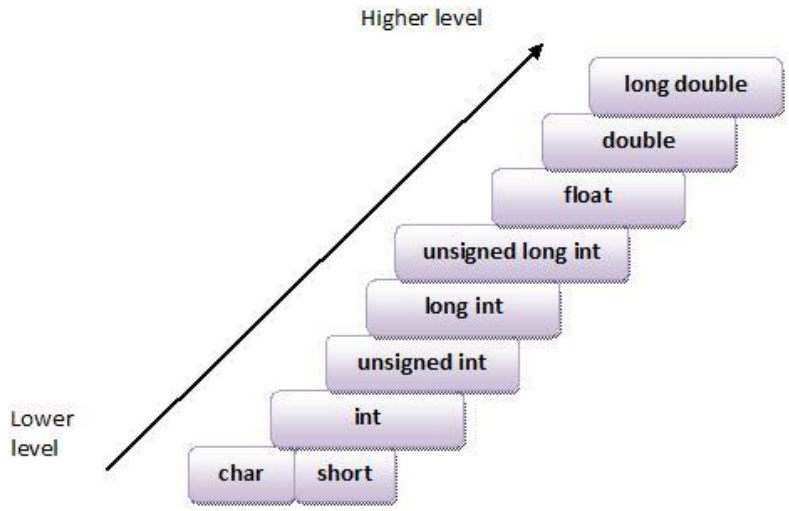
If the both operands are of the same type, promotion is not needed. If they are not, promotion follows these rules:

- ***float* operands are converted to *double*.**
- ***char or short* are converted to *int*.**
- **If anyone operand is *double*, the other operand is also converted to *double*, and that is the type of result.**

  **or**
- **If anyone operand is *long*, the other operand is also converted to *long*, and that is the type of result.**
- **If anyone operand is *unsigned*, the other operand is also converted to *unsigned*, and**

  **that is the type of result**.

The smallest to the largest data types with respect to size are given as follows:




Conversion hierarchy of data types

**A Sample 'C' program that illustares the use implicit type conversion**

```
#include<stdio.h>
int main()
{
        int sum, num=17;
        char ch='A';
        sum=num+ch;
        printf("The value of sum=%d\n", sum);
        return 0;
}
```

**Output**:

The value of sum= 82 i.e. sum=num+ch=>17+65 (ASCII value of 'A'

**Explicit typecasting:** Which is intentionally performed by the programmer for his requirement in a C program?The explicit type conversion is also known as **type casting**.We can convert the values from one type to another explicitly using the **cast operator** as follows:

**Syntax**

**(data_type) expression;**

Where, *data_type* is any valid C data type, and *expression* may be constant, variable or expression.

The following rules have to be followed while converting the expression from one type to another to avoid the loss of information:

1. All integer types to be converted to float.
2. All float types to be converted to double.

All character types to be converted to integer

Let us look at some examples of type casting.

- ✓ res = ( int ) 9.5;
  9.5 is converted to 9 by truncation then assigned to *res.*
- ✓ res = ( int ) 12.3 / ( int ) 4.2 ;
  It evaluated as 12/4 and the value 3 is assigned to *res.*
- ✓ res = ( double ) total / n;
  *total* is converted to *double* and then division is done in float point mode.
- ✓ res = ( int )(a + b);
  The value of *(a + b)* is converted to *integer* and then assigned to *res.*
- ✓ res = ( int )a + b;
  *a* is converted to *int* and then added with *b.*