Unit 2 Basics of Programming

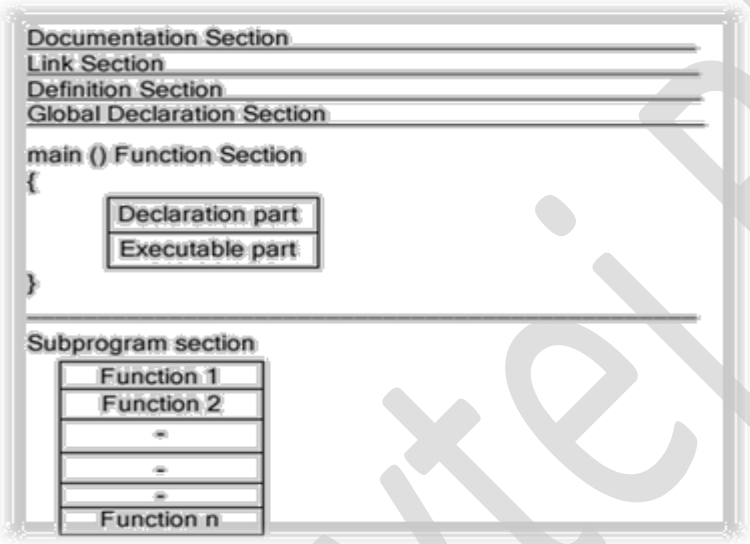
Problem Analysis

When we are going to develop any solution to the problem, we must fully understand the nature of the problem and what we want the program to do. Without the comprehension and definition of the problem at hand, program design might turn into a hit-or-miss approach. We must carefully decide the following at this stage:

- What kind of data will go in
- What kind of outputs are needed, and
- What are the constraints and conditions under which the program has to operate?

Basic Structure of C Program

C program can be viewed as group of building blocks called functions. A function is a subroutine that may include one or more statement designed to perform a specific task. A C program may contain one or more section as shown below:



Documentation Section

This section contains set of comments lines consist of details like program name, author name and purpose or functionality of the program.

Link Section

This section consists of instructions to be given to the compiler to link functions from the system library. For example if you want to use some mathematical function then you have to define link for math.h file in link section. For Example

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

Definition Section

This section defines all the symbolic constants. For example PI=3.14. By defining symbolic constant one can use these symbolic constant instead of constant value.

```
# define PI 3.14
# define Temp 35
```

Global Declaration Section

This section contains the declaration of variables which are used by more than one function of the program.

Main Function Section

A main () function is a heart of any 'C' language program. Any C program is made up of 1 or more than 1 function. If there is only 1 function in the program then it must be the main program. An execution of any C program starts with main () function. Your main ( ) function contains mainly two parts.

1. Declaration section: In this section various variables are declared.
2. Execution section: In this section various executable statements are included.

Subprogram or Sub Function Section

They are the code section which is defined outside the boundary of main function. This function can be called from any point or anywhere from the program. Generally they are defining to solve some frequent tasks.

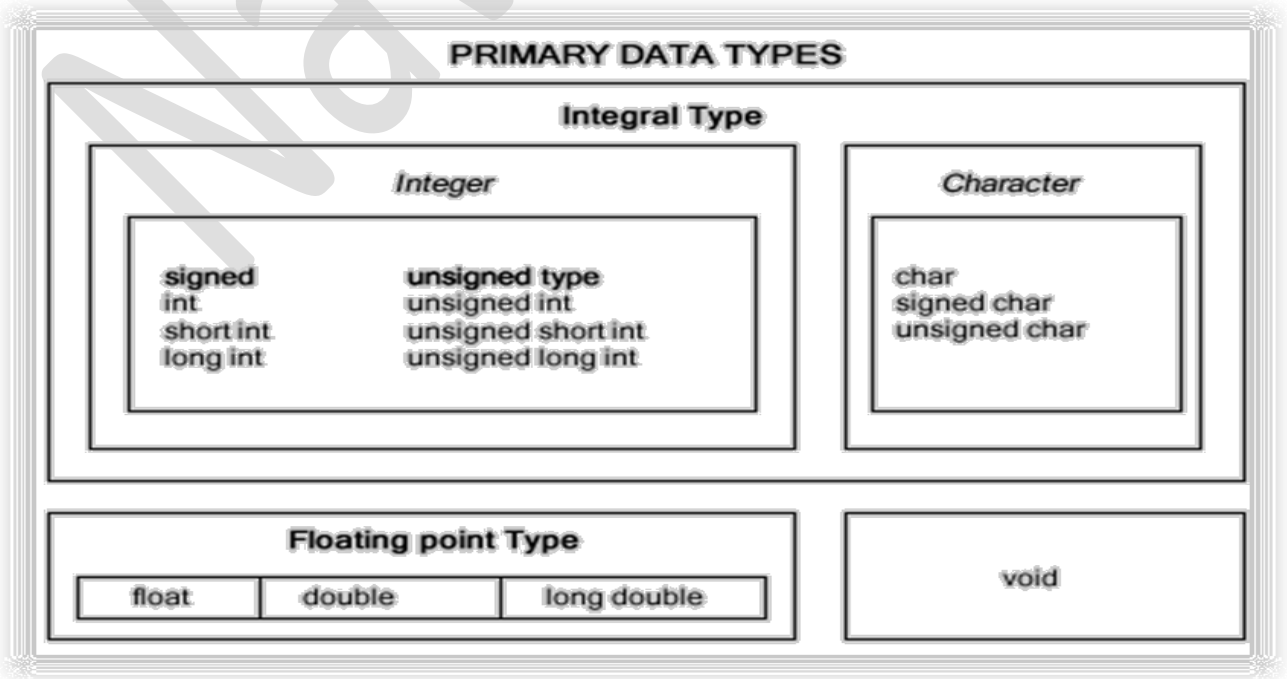
Data Types

"The variable is used to store the different type of data value such as integer, fractional number, character, string, etc. This data value determines the type of variable, which is known as data type." ANSI C supports three classes of data types:

1. Primary (or fundamental) data types
2. Derived data types
3. User-defined data types

PRIMARY DATA TYPE

In C language primary data types are mainly four: int (Integer value), float (single precision value), double (double precision value), char (Character value). There are also some other data types but it is combination of all these four data types such as long int, long float, unsigned int, etc.



Integer Types

Integers are whole numbers with a machine dependent range of values. A good programming language is to support the programmer by giving a control on a range of numbers and storage space. C has 3 classes of integer storage namely short int, int and long int. All of these data types have signed and unsigned forms. A short int requires half the space than normal integer values. Unsigned numbers are always positive and consume all the bits for the magnitude of the number. The long and unsigned integers are used to declare a longer range of values.

For integer variable, we require 2 bytes of memory for 16-bit computer machine so we can store the value from –215 to (215) - 1 i.e. from –32,768 to +32,767. In 32-bit computer machine the size of integer type variable is 4 bytes that means it can store the value from –231 to (231) -1 i.e. from –2,14,74,83,648 to +2,14,74,83,647.

Floating Point Types

Floating point number represents a real number with 6 digits precision. Floating point numbers are denoted by the keyword float. When the accuracy of the floating point number is insufficient, we can use the double to define the number. The double is same as float but with longer precision. To extend the precision further we can use long double which consumes 80 bits of memory space.

Void Types

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

Character Types

A single character can be defined as a defined as a character type of data. Characters are usually stored in 8 bits of internal storage. The qualifier signed or unsigned can be explicitly applied to char. While unsigned characters have values between 0 and 255, signed characters have values from –128 to 127.

Size and Range of Data types on a 16-bit Machine

Datatype	Specifier	Size (In bytes) 16-bit Computer	Range of value for 16-bit Computer
singed integer	%d	2	**-32768 to +32767 (-2^{15} to $2^{15}-1$)**
unsigned integer	%u	2	**0 to 65535 (0 to $2^{16}-1$)**
long integer	%ld	4	**-2147483648 to +2147483647 (-2^{31} to $2^{31}-1$)**
unsigned long integer	%lu	4	**0 to $2^{32}-1$**
Float	%f or %g	4	**3.4E-38 to 3.4E+38**
Double	%lf	8	**1.7E-308 to 1.7E+308**
long double	%Lf	10	**3.4E-4932 to 1.1E+4932**
Singed character	%c	1	**-128 to 127**
unsigned character	%c	1	**0 to 255**

Variables

"A variable is a data name that may be used to store a value."

The value of variable will not remain fixed during entire program. As program is executed statement by statement, value of variable may be varies.

For the variable computer has to allocate memory location depending on the variable type. For example for integer variable it requires two bytes memory location, for float variable it requires four bytes memory location. To provide the memory location for the variable first we have to declare the variable. Declaration statement is always on the beginning of function or program. During declaration each and every variable has given one name. We have to follow certain rules to give the variable name. These rules can be applied to generally any identifier in C programming language.

Rules to define variable in C

1. It contains only alphabet, digits, and underscore (_)
2. The first character is always alphabet.
3. Minimum length of variable name is 1 character and maximum is 32 characters.
4. There is a difference between uppercase and lowercase letter name. For example, "Sum" and "sum" are two different variables.
5. The variable name should not be keyword.
6. White space is not allowed in the variable name. Space, tab, and enter key are the white space character.

Examples

The following are some valid variable names:

Int	**computer**	**b_ca**	**CO2**	**line**
Fybca	**bca**	**I_value**	**emp_name**	**char_c**

The following are some invalid variable names with reasons:

Variable Name	Reason
A's	**Illegal character**
Int	**Keyword**
group one	**Blank not allowed**
4marks	**First character should be a letter.**
C++	**+ not allowed**
emp-name	**Hyphen (-) not allowed.**

Declaration of Variable

We must declare the variable to use them in program. Declaration does two things:

1. It tells the compiler what the variable name is.
2. It specifies what type of data the variable will hold.

The declaration of the variables must be done before they are used in the program. The syntax for declaring a variable is as follows:

data-type v1,v2,…,vn;

v1,v2,…,vn are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For example, valid declarations are:

```
int count;
int number, total;
double ratio;
```

Initialization of variables

"The process of giving initial value to your variable is known as initialization."

Variable initialization also can be performed during the variable declaration as under:

```
data-type variable_name = some constant value
```

Also you can perform the initialization after declaration of variable. For example,

```
int     tot_marks=100;     // Initialization during declaration
float   pi=3.14;
double distance;
distance=34.244;    // Initialization after declaration
```

When you simply declare the variable and not assign any value then your variable contains some random value that is known as garbage value.

Assignment Statement

Values can be assigned to variables using the assignment operator = as follows:

```
variable_name = constant;
```

Some examples of assignment are:

```
initial_value = 0;
final_value = 100.25;
yes = 'x';
```

I/O Statements

I/O statements are generally used to take input from user and print output to screen. C provides two basic functions that are very useful for I/O operations. Function printf is used for output purpose and scanf function is used to take input from user. Both the function is from the header file stdio.h.

printf()

printf is a predefined standard C function for printing output. The printf function causes everything between the starting and the ending quotation marks to be printed out. Consider the following example.

```
printf("This is printf function example");
```

in the example, the output will be:

```
This is printf function example
```

Suppose we want to print the following:

This is printf function.
printf function is used to print output to screen.

This can be achieved by adding another printf function as show below:

```
printf("This is printf function. \n");
printf("printf function is used to print output to screen.");
```

The information contained between the parentheses is called the argument of the function. In this case it contains the combination of two character i.e. '\' and 'n' at the end of argument. This combination is called the newline character, which is used to start the printing from the newline.

The printf statement provides certain features that can be effectively exploited to control the variable values on the terminal while printing. The general form of printf statement is:

```
printf("control string",arg1,arg2,arg3, … ,argn);
```

Control string consists of three types of items:

- Characters that will be printed on the screen as they appear.
- Format specifications that define the output format for display each item.
- Escape sequence characters such as \n,\t.

The control string indicates how many arguments follow and what their types are. The arguments arg1, arg2, …, argn are the variables whose values are printed according to the specifications of the control string. The argument should match in number, order and type with the format specification.

Integer Variable

```
int a=10;
printf("%d",a);
```

Here %d is format string to print some integer value and a is the integer variable whose value will be printed by printf() function. This will print value of a "10" on the screen. You can make this output interactive by writing them

```
int a=10;
printf("a=%d",a);
```

This will print "a=10" on the screen

To print multiple variable's value one can use printf() function in following way.

```
int p=1000,r=10,n=5;
printf("amount=%d rate=%d yeat=%d",p,r,n);
```

This will print "amount=1000 rate=10 year=5" on the screen

Float Variable

```
float per=70.20;
printf("Percentage=%f",per);
```

Here %f is format string to print some float(real) value and per is the float variable whose value will be printed by printf() function. This will print value of a "Percentage=70.20" on the screen.

Character Variable

```
char ans='Y';
printf("Answer=%c",ans);
```

Here %c is format string to print single character value and ans is the character variable whose value will be printed by printf() function. This will print value of a "Answer=Y" on the screen.

Suppose we want to print "Amar" on the screen with character variable

```
char c1='A',c2='m',c3='a',c4='r' ;
printf("Name = %c %c %c %c",c1,c2,c3,c4);
```

This will print "Name=A m a r" on the screen.

Mixed Variable

You can use single printf() function to print different data type variable's value also.

```
int rno=10;
char res='P';
float per=75.70;
printf("Rollno=%d Result=%c Percentage=%f",rno,res,per);
```

This will print message "Rollno=10 Result=P Percentage=75.70" on the screen.

scanf()

scanf() function is use to read data from keyboard and to store that data in the variables. The general syntax for scanf() function is as follows.

```
scanf("Format String",&variable);
```

Here format string is used to define which type of data it is taking as input this format string can be %c for character, %d for integer variable and %f for float variable. Whereas variable the name of memory location or name of the variable and & sign is an operator that tells the compiler the address of the variable where we want to store the value. One can take multiple input of variable with single scanf() function but it is recommended that there should be one variable input with one scanf() function.

```
scanf("Format string1,format string2",&variable1,&variable2);
```

Integer Variable

```
int rollno;
printf("Enter rollno=");
scanf("%d",&rollno);
```

Here in scanf() function %d is a format string for integer variable and &rollno will give the address of variable rollno to store the value at variable rollno location..

Float Variable

```
float per;
printf("Enter Percentage=");
scanf("%f",&per);
```

Here in scanf() function %f is a format string for float variable and &per will give the address of variable per to store the value at variable per location.

Character Variable

```
char ans;
printf("enter answer=");
scanf("%c",&ans);
```

Here in scanf() function %c is a format string for character variable and &ans will give the address of variable ans to store the value at variable ans location.

Operators

Operator is symbol, which represents, a particular operation that can be performed on some data. The data itself (which can be either a variable or a constant) is called the "operand". The operator thus operates on operand. They are classified as:

(1) Arithmetic (2) relational (3) logical
(4) Assignment(Short hand) (5) increment & decrement (6) conditional

Arithmetic operators

C provides basic arithmetic operators *, +, -, /, and %. These all operators can work with only built-in data type (int, float, char, double, etc.)

OPERATOR	PURPOSE
+ (Addition)	Addition
- (Subtraction)	Subtraction
* (Multiplication)	Multiplication
/ (Division)	Division
% (Modulo)	Reminder after integer division

Addition Operator

It has two format, one is unary plus and other is binary plus. Unary plus works with only one operand. Unary plus was not supported in older version of 'C'. For example, in following example first we have operand 'b' and '+' operator is used with it. In second expression 30 is operand and '+' operator is used with it.

```
a = +b;
a = +30;
```

Another format is binary plus in which two operands is used. For example,

```
a = b + c;
a = 23 + 50;
```

Subtraction operator

It has also two format, one is unary minus which multiply your operand with –1. For example, in first expression if the value of 'b' is –5 before this statement execution then after execution value of 'a' will be +5.

```
a = -b;
a = -30;
```

Another format is binary minus in which two operands are used. For example,

```
a = b – c;
a = 34 – 30;
```

Multiplication Operator

It has only binary format. So we can use this operator with only two operands. For example,

```
a = b * c;
a = 5 * 3;
```

Division Operator

Division operator is only work in binary operation. So we have to use this operator with only two operands. For example,

```
a = b / c;
a = 6 / 3;
```

Modulo Operator

Modulo operator is used to determine the remainder after division operation. Modulo operator cannot be used for the float value. If any of the operand is float then compilation error is generated.

For example,

```
a = 6 % 4;          // Answer will be 2
a = 7.8 % 4;        // Compilation error
```

Also if we are using modulo operator for the negative value then sign of the answer is sign of the first operand. For example

a = -10 % 3 = - 1 (Sign of 1st operand 10 is minus so answer is minus)
a = - 10 % -3 = -1 (Sign of 1st operand 10 is minus so answer is minus)
a = 10 % -3 = 1 (Sign of 1st operand 10 is plus so answer is plus)

Integer Arithmetic

When both the operands in a single arithmetic expression (such as a + b) are integers, the expression is called integer expression, and the operation is called integer arithmetic. Integer arithmetic always results an integer value.

Example: If a & b are integer & its value is a = 14 and b=14 then we have the following results:

a -b results 10
a + b results 18
a*b results 56
a/b results 3 (decimal part truncated)
a % b results 2 (remainder)

Real Arithmetic

An arithmetic operation involving only real operands is called real arithmetic.

Example: If a, b and c are floats then we will have following results:

a = 6.0/7.0 (a becomes 0.857143)
b = 1.0/3.0 (b becomes 0.33333)
c = -2.0/3.0 (c becomes 0.66667)

The operator % cannot be used with real operand.

Mixed Mode Arithmetic

When one of the operand is real and the other is integer the expression is called Mixed-mode arithmetic expression.

If either operand is of the real type, then only the real operation is performed and the result is always a real number. Thus 15/10.0 gives 1.5 while 15/10 gives 1.

Relational operator

A relation operator is used to make comparisons between two expressions. The relational operator returns zero values or nonzero values. The zero value is taken as false while the nonzero value is taken as true. The expression containing the relational operator is known as the relational expression. C language supports six relational operators.

Operator	Meaning
<	**Is less than**
<=	**Is less than or equal to**
>	**Is greater than**
>=	**Is greater than or equal to**
==	**Is equal to**
!=	**Is not equal to**

Suppose i, j and k are integer variable with 1,2 and 3 as a value respectively. Following table shows the expression and its interpretation and its value.

Expression	Interpretation	Value
i<j	True	1
(i+j)>=k	True	1
(j+k)>(I+5)	False	0
K!=3	False	0
J = = 2	True	1

Logical Operator

Logical operators are useful in combining one or more conditions. C language has the following three logical operators.

Operator	Meaning
&&	**Logical AND(True only if all conditions are true, otherwise false)**
||	**Logical OR(False only if all conditions are false, otherwise true)**
!	**Logical NOT(True if exp. is false or vice - versa)**

Example on Logical AND:

A =5; B=10; X =10;

The condition (A < B && X == 10) evaluates true.
The condition (A < B && X!= 10) evaluates false.
The condition (A > B && X ==10) evaluates false.
The condition (A > B && X!= 10) evaluates false.

Example on Logical OR:

A =5; B =10; X = 10;

The condition (A < B || X ==10) evaluates true.
Tl1ccondition (A < B || X != 10) evaluates true.
The condition (A > B || X ==10) evaluates true.
The condition (A > B || X != 10) evaluates false.

Example on Logical NOT:

The condition !(A <B) evaluates false and !(A> B)evaluates true.

Assignment Operator

Assignment operators are used to assign the result of an expression to a variable. The general form of assignment operator is

variable = expression;

C has a set of “Shorthand" assignment operators of the form

V op = exp ;

Where V is a variable, exp is an expression and op is a binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

The assignment V op =exp; is equivalent to V = V op (exp);

Example:

Simple Expression	Shorthand Operators
a = a + 1	**a += 1**
a = a –23	**a -= 23**
a = a * 4	**a * = 4**
a = a / 5	**a /= 5**
a = a % 10	**a %= 10**

The use of shorthand assignment operator has three advantages:

- What appears on the left hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise and easier to read.
- The statement is more efficient.

Increment Decrement Operator

The C language uses two operators for incrementing and decrementing variables.

Increment Operator

- A symbol for an increment operator is ++.
- It increases the value of a variable by 1.
- It is a unary operator.
- It can be prefix or postfix.

	Prefix	Postfix
Syntax:	++VariableName ;	VariableName++ ;
Example:	int a=5 ; ++a; printf("Value of a=%d", a) ; Output: Value of a = 6	int a=5 ; a++; printf("Value of a=%d", a) ; Output: Value of a = 6

When we used a prefix in an expression, the value of a variable is incremented before used in expression. But when used as a postfix, its value is first used in expression and then the value is incremented.

Example

Prefix	Postfix

int a = 0, b = 10:
a = ++b;
printf("A=%d, B=%d",a,b);
	int a = 0, b = 10:
	a = b++;
	printf("A=%d, B=%d",a,b);

Output: A = 11, B = 11	Output: A = 10, B = 11

Decrement Operator

- A symbol for a decrement operator is --.
- It decreases the value of a variable by 1.
- It is a unary operator.
- It can be prefix or postfix.

	Prefix	Postfix
Syntax:	- -VariableName ;	VariableName- - ;
Example:	int a=5 ; - -a; printf("Value of a=%d", a) ;	int a=5 ; a- -; printf("Value of a=%d", a) ;
	Output: Value of a = 4	Output: Value of a = 4

When we used a prefix in an expression, the value of a variable is decremented before used in expression. But when used as a postfix, its value is first used in expression and then the value is decremented.

Example

Prefix	Postfix

int a = 0, b = 10:
a = - -b;
printf("A=%d, B=%d",a,b);
	int a = 0, b = 10:
	a = b- -;
	printf("A=%d, B=%d",a,b);

Output: A = 9, B = 9	Output: A = 10, B = 9

Conditional Operator

It is also known as **Ternary Operator**. The general form of ternary operator is

exp 1 ? exp2 : exp3 ; Where exp1, exp2, exp3 are expressions.

The operator **? :** Works as follows :
exp1 is evaluated first.

- If it is non-zero (true), then the expression exp2 is evaluated and becomes the value of the expression.
- If it is zero (false), then the expression exp3 is evaluated and becomes the value of the expression.

Example

```
A = 10;
B = 15;
X = (A>B) ? A : B;
```

Another example finding maximum using conditional operator

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

```
void main()
{
        int a, b, max;
        printf('\n Enter a….");
        scanf("%d" ,&a);
        printf("\n Enter b...");
        scanf("%d", &b);
        max = ( a > b ) ? a : b ;
        printf(\n Maximum = %d" ,max);
        getch();
}
```

Output:

Enter a… 10
Enter b… 5
Maximum = 10

Arithmetic Expression

An arithmetic expression is a combination of variables, constants and operations arranged as per the syntax of the language. Some of the examples of C expression are show in following table:

Algebraic expression	C expression
a x b – c	a * b -c
(m+n)(x+y)	(m+n)*(x+y)
$\left(\frac{ab}{c}\right)$	a*b/c
$3x^2 + 2x + 1$	3 * x * x + 2 * x + 1
$\left(\frac{x}{y}\right) + c$	x/y+c

Evaluation of Expression

Expressions are evaluated using an assignment of the form:

Variable =expression;

Example: A = 10; B=2; C=3;

X= A * B - C; Now, X contains 17

Rules for Evaluation of Expression

- First, parenthesized sub expression from left to right is evaluated.
- If parentheses are nested, the evaluation begins with the innermost sub-expression.
- The precedence rule is applied in determining the order of application of operators in evaluation sub-expressions.
- The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When parentheses are used, the expressions within parentheses assume highest priority.

Examples of some Arithmetic Expression with its solutions

1.	9/2-5%3 = 4 – 2 = 2	**2.**	8 – 12 / *3* = 8 – 4 = 4	**3.**	18 % 5 - 6 % 4 = 3 - 2 = 1	**4.**	2.95 + 0.05 * 10 = 2.95 + 0.50 = 3.45
5.	7 + 5 / 3 / 3 = 7 + 1/3 = 0	**6.**	5 * 9/5 * 5 = 45/5*5 = 9*5 = 45	**7.**	5 * 4/ (6-3)+ (10-7) = 5 * 4 /3 + 3 = 20/3 + 3 = 6 + 3 = 9	**8.**	7*4/2+4-3 = 28/2 + 4 – 3 = 14 + 4 – 3 = 15

Following table shows precedence and associativity of operators

Operator	Precedence	Associativity
() [] . -> ++ --	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ -- + - (*type*) * & sizeof	Prefix increment/decrement Unary plus/minus Cast (change *type*) Dereference Address Determine size in bytes	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional	right-to-left
= += -= *= /= %=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus	right-to-left