PROGRAMMING IN C - UNIT-I

1 Getting Started

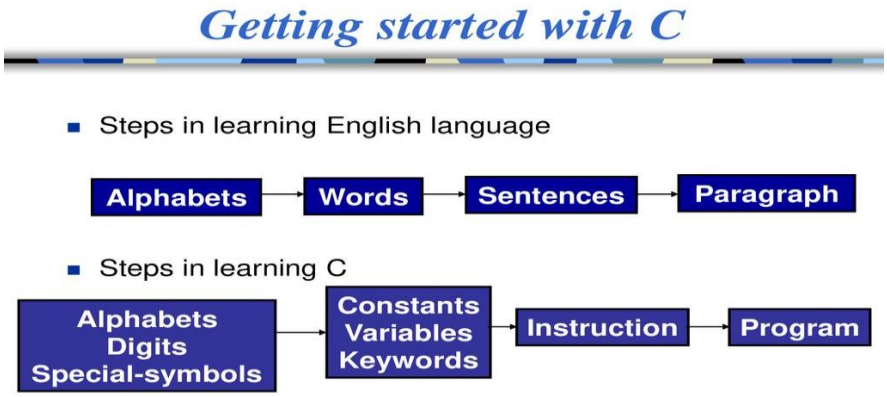
What is C

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by aman named Dennis Ritchie. Possibly C seems so popular is because it is reliable, simple and easy to use.

one should first learn all the language elements very thoroughly using C language before migrating to C++, C# or Java.Major parts of popular operating systems like Windows, UNIX, Linux is still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C.

C language scores over other languages. Many popular gaming frameworks have been built using C language.

Getting Started with C



The C Character Set

A character denotes any alphabet, digit or special symbol used to represent information.

Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Constants, Variables and Keywords

The alphabets, numbers and special symbols when properly combined form constants, variables and keywords

A constant is an entity that doesn't change whereas a variable is an entity that may change.

In any program we typically do lots of calculations. The results of these calculations are stored in computers memory. Like human memory the computer memory also consists of millions of cells. The calculated values are stored in these memory cells. To make the retrieval and usage of these values easy these memory cells (also called memory locations) are given names. Since the value stored in each location may change the names given to these locations are called variable names.

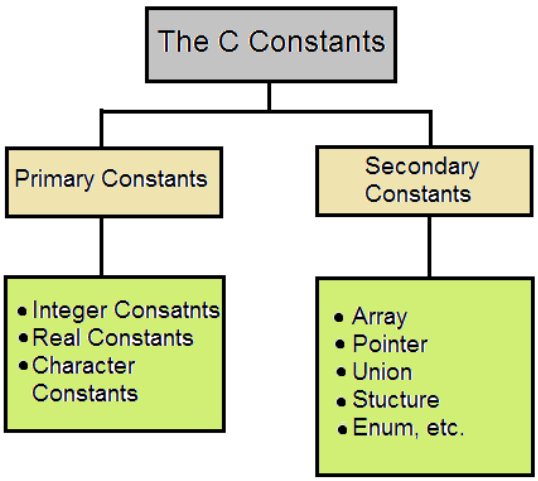
EX: x = 3

a = 5

Types of C Constants

C constants can be divided into two major categories:

(a) Primary Constants
(b) Secondary Constants



Rules for Constructing Integer Constants

(a) An integer constant must have at least one digit.
(b) It must not have a decimal point.
(c) It can be either positive or negative.
(d) If no sign precedes an integer constant it is assumed to be positive.
(e) No commas or blanks are allowed within an integer constant.
(f) The allowable range for integer constants is -32768 to 32767.

Ex.: 426
+782
-8000
-7605

Rules for Constructing Real Constants

Real constants are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

Fractional form

(a) A real constant must have at least one digit.
(b) It must have a decimal point.
(c) It could be either positive or negative.
(d) Default sign is positive.
(e) No commas or blanks are allowed within a real constant.

Ex.: +325.34

426.0

-32.76

-48.5792

Exponential form

(a) The mantissa part and the exponential part should beseparated by a letter e.
(b) The mantissa part may have a positive or negative sign.
(c) Default sign of mantissa part is positive.
(d) The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.
(e) Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Ex.: +3.2e-5

4.1e8

-0.2e+3

-3.2e-5

Rules for Constructing Character Constants

(a) A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas. Both the inverted commas should point to the left. For example, 'A' is a valid character constant whereas 'A' is not.
(b) The maximum length of a character constant can be 1 character.

Ex.: 'A"I'

'5'

'='

Types of C Variables

a) A variable name is any combination of 1 to 31 alphabets, digits or underscores.
b) The first character in the variable name must be an alphabet or underscore.
c) No commas or blanks are allowed within a variable name.
d) No special symbol other than an underscore (as in **gross_sal**) can be used in a variable name.

Ex.: si_int

m_hra
pop_e_89

C Keywords

Keywords are the words whose meaning has already been explained to the C compiler (or in a broad sense to the computer). The keywords **cannot** be used as variable names because if we do so we are trying to assign a new meaning to the keyword, which is not allowed by the computer. The keywords are also called 'Reserved words'.

There are only 32 keywords available in C

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

The First C Program

Rules that are applicable to all C programs:

a) Each instruction in a C program is written as a separate statement.

b) The statements in a program must appear in the same order in which we wish them to be executed.

c) Blank spaces may be inserted between two words to improve the readability of the statement. However, no blank spaces are allowed within a variable, constant or keyword.

d) All statements are entered in small case letters.

e) C has no specific rules for the position at which a statement is to be written. That's why it is often called a free-form language.

f) Every C statement must end with a **;**. Thus **;** acts as a statement terminator.

```
main( )
{
    int x,y,z ;
    x=10;
    y=20;
     z=x+y;
     /* adding two numbers x+y*/
     printf ( "Addition of two numbers %d",z ) ;
    }
```

- Comment about the program should be enclosed within /* */.
- Though comments are not necessary. Any number of comments can be written at any place in the program.
- Often programmers seem to ignore writing of comments. But when a team is building big software well commented code is almost essential for other team members to understand it.
- Comments cannot be nested. For example,

```
/* Cal of SI /* Author sam date 01/01/2002 */ */
```

is invalid.

- A comment can be split over more than one line, as in,

```
/* This is
a jazzy
   comment */
```

main( ) is a collective name given to a set of statements. This name has to be **main( )**, it cannot be anything else. All statements that belong to **main( )** are enclosed within a pair of braces { }

Technically speaking **main( )** is a function

Any variable used in the program must be declared before using it

Any C statement always ends with a ;

Displaying output:

All output to screen is achieved using readymade library functions. One such function is **printf( )**.

The general form of **printf( )** function is,

printf ("<format string>", <list of variables>) ;

Following are some examples of usage of **printf( )** function:

```
EX:  printf ( "%f", si ) ;

printf ( "%d %d %f %f", p, n, r, si ) ;

printf ( "Simple interest = Rs. %f", si ) ;

printf ( "Prin = %d \nRate = %f", p, r ) ;
```

printf( ) can not only print values of variables, it can alsoprint the result of an expression.

```
printf ( "%d %d %d %d", 3, 3 + 2, c, a + b * c – d ) ;
```

Receiving Input

Receiving input can be achieved using a function called **scanf( )**. **scanf( )** receives them from the keyboard. The ampersand (**&**) before the variables in the **scanf( )** function is a must. **&** is an 'Address of' operator. It gives the location number used by the variable in memory.

Ex:

```
scanf ( "%d", &num ) ;
```

Compilation and Execution

To type your C program you needanother program called Editor. Once the program has been typed itneeds to be converted to machine language (0s and 1s) before the machine can execute it. To carry out this conversion we need another program called Compiler. Compiler vendors provide an Integrated Development Environment (IDE) which consists of an Editor as well as the Compiler.

For example, Turbo C, Turbo C++ and Microsoft C are some of the popular compilers that work under MS-DOS

The steps that you need to follow to compile and execute your first C program…

(a) Start the compiler at **C>** prompt. The compiler (TC.EXE is usually present in **C:\TC\BIN** directory).
(b) Select **New** from the **File** menu.
(c) Type the program.
(d) Save the program using **F2** under a proper name (say Program1.c).
(e) Use **Ctrl + F9** to compile and execute the program.
(f) Use **Alt + F5** to view the output.

C Instructions

Now that we have written a few programs let us look at the instructions that we used in these programs. There are basically three types of instructions in C:

(a) Type Declaration Instruction
(b) Arithmetic Instruction
(c) Control Instruction

The purpose of each of these instructions is given below:

(a) Type declaration instruction – To declare the type of variables used in a C program.

(b) Arithmetic instruction – To perform arithmetic operations between constants and variables.

(c) Control instruction – To control the sequence of execution of various statements in a C program.

Type Declaration Instruction

Any variable used in the program must be declared before using it in any statement. The type declaration statement is written at the beginning of **main( )** function.

Ex.: int bas ;

float rs, grosssal ;char
name, code ;

(a) While declaring the type of variable we can also initialize it as shown below.

int i = 10, j = 25 ;

float a = 1.5, b = 1.99 + 2.4 * 1.44 ;

Arithmetic Instruction

A C arithmetic instruction consists of a variable name on the left hand side of = and variable names & constants on the right hand side of =. The variables and constants appearing on the right hand side of = are connected by arithmetic operators like **+, -, *,** and **/**.

(a) Integer mode arithmetic statement - This is an arithmetic statement in which all operands are either integer variables or integer constants.

Ex.: int i, king, issac, noteit ;i
= i + 1 ;

king = issac * 234 + noteit - 7689 ;

(b) Real mode arithmetic statement - This is an arithmetic statement in which all operands are either real constants or real variables.

Ex.: float qbee, antink, si, prin, anoy, roi ; qbee =
antink + 23.123 / 4.5 * 0.3442 ;si = prin *
anoy * roi / 100.0 ;

(c) Mixed mode arithmetic statement - This is an arithmetic statement in which some of the operands are integers and some of the operands are real.

```
Ex.: float  si, prin, anoy, roi, avg ;int
        a, b, c, num ;

        si = prin * anoy * roi / 100.0 ;
        avg = ( a + b + c + num ) / 4 ;
```

(a) Arithmetic operations can be performed on **int**s, **float**s and **char**s.

Thus the statements,

```
char x, y ;
int  z ;

x = 'a' ;

y = 'b' ;

z = x + y
```

Integer and Float Conversions

(a) An arithmetic operation between an integer and integer always yields an integer result.

(b) An operation between a real and real always yields a real result.

(c) An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.

I think a few practical examples shown in the following figure would put the issue beyond doubt.

Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

Type Conversion in Assignments

For example, consider the following assignment statements.

```
int  i ;

float   b ;i
= 3.5 ; b
= 30 ;
```

Here in the first assignment statement though the expression's value is a **float** (3.5) it cannot be stored in **i** since it is an **int**. In such a case the **float** is demoted to an **int** and then its value is stored. Hence what gets stored in **i** is 3. Exactly opposite happens in the next statement. Here, 30 is promoted to 30.000000 and then stored in **b**, since **b** being a **float** variable cannot hold anything except a

float value.

Hierarchy of Operations

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

Now a few tips about usage of operators in general.

(a) Within parentheses the same hierarchy as mentioned in Figure 1.11 is operative. Also, if there are more than one set of parentheses, the operations within the innermost parentheses would be performed first, followed by the operations within the second innermost pair and so on.

(b) We must always remember to use pairs of parentheses. A careless imbalance of the right and left parentheses is a common error. Best way to avoid this error is to type ( ) and then type an expression inside it.

A few examples would clarify the issue further.

Example : Determine the hierarchy of operations and evaluatethe following expression:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8	operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8	operation: /
i = 1 + 1+ 8 - 2 + 5 / 8	operation: /
i = 1 + 1 + 8 - 2 + 0	operation: /
i = 2 + 8 - 2 + 0	operation: +
i = 10 - 2 + 0	operation: +
i = 8 + 0	operation : -
i = 8	operation: +

Control Instructions in C

There are four types of control instructions in C. They are:

(a) Sequence Control Instruction
(b) Selection or Decision Control Instruction
(c) Repetition or Loop Control Instruction
(d) Case Control Instruction

2 The Decision Control Structure

As mentioned earlier, a decision control instruction can be implemented in C using:

(a) The **if** statement
(b) The **if-else** statement
(c) The conditional operators

The *if* Statement

C uses the keyword **if** to implement the decision control instruction. The general form of **if** statement lookslike this:

Syntax

The syntax is given below −

```
if (condition)
{
  Statement (s);
}
```

The keyword **if** tells the compiler that what follows is a decision control instruction. The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true then the statement is not executed; instead the program skips past it.

this expression	is true if
x == y	x is equal to y
x != y	x is not equal to y
x < y	x is less than y
x > y	x is greater than y
x <= y	x is less than or equal to y
x >= y	x is greater than or equal to y

FLOWCHART:

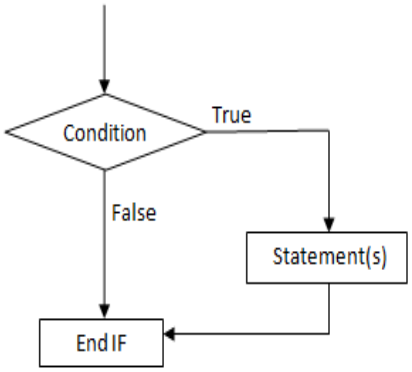


fig: Flowchart for if statement

Here is a simple program, which demonstrates the useof **if** and the relational operators.

```
/* Demonstration of if statement */
main( )
{
    int num ;

    printf ( "Enter a number less than 10 " ) ;
    scanf ( "%d", &num ) ;

    if ( num <= 10 )
        printf ( "hello,you entered the number less than 10 !" ) ;
}
```

Ex:
```
if ( 3 + 2 % 5 )
    printf ( "This works" ) ;

if ( a = 10 )
    printf ( "Even this works" ) ;

if ( -5 )
    printf ( "Surprisingly even this works" ) ;
```

Multiple Statements within *if*

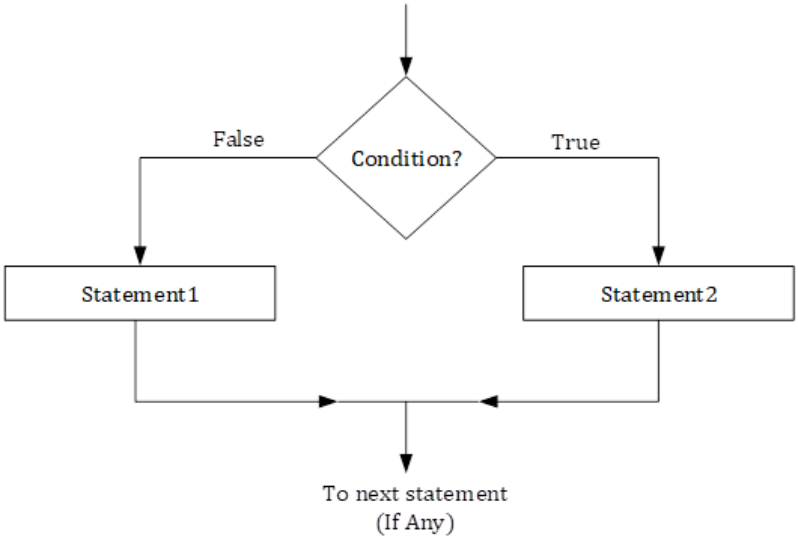
It may so happen that in a program we want more than one statement to be executed if the expression following **if** is satisfied. If such multiple statements are to be executed then they must be placed within a pair of braces as illustrated in the following example.

The *if-else* Statement

The **if** statement by itself will execute a single statement, or agroup of statements, when the expression following **if** evaluates to true. It does nothing when the expression evaluates to false.**syntax:**

```
if (condition)
{
  // do this if condition is true
  // if true statements
}
else
{
  // do this is condition is false
  // if false statements
}
```

Flowchart:



Ex:

```
main( )
{
    int age;
    printf ( "Enter your age: " ) ;
    scanf ( "%d", &age ) ;
    if ( age <= 18 )
   {
        Printf("you are eligible for vote");
        }
        else
   {
        Printf("you are not eligible for vote");
        }
        }
```

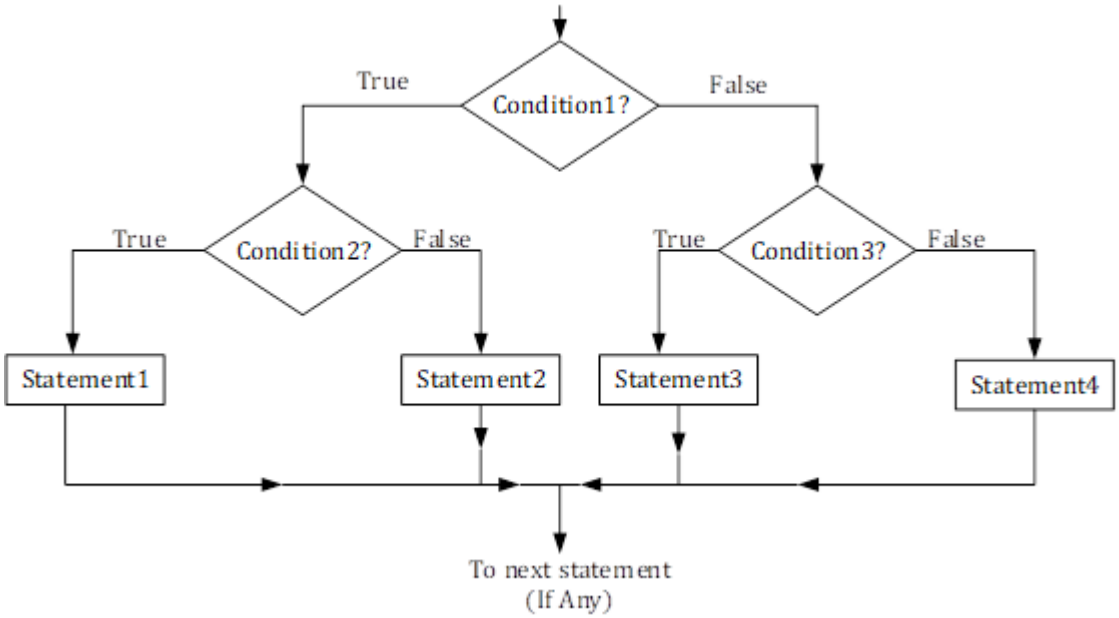
Nested *if-elses*

It is perfectly all right if we write an entire **if-else** construct within either the body of the **if** statement or the body of an **else** statement.This is called 'nesting'of **if**s.

Syntax of Nested if else statement:

```
if(condition) {
   //Nested if else inside the body of "if"
   if(condition2) {
     //Statements inside the body of nested "if"
   }
   else {
     //Statements inside the body of nested "else"
   }
}
else {
   //Statements inside the body of "else"
}
```

Flowchart:



Ex:

```
/* A quick demo of nested if-else */
main( )
{
    int  i ;

    printf ( "Enter either 1 or 2 " ) ;
    scanf ( "%d", &i ) ;

    if ( i == 1 )
        printf ( "You would go to heaven !" ) ;
    else
    {   if ( i == 2 )
        printf ( "Hell was created with you in mind" ) ;
        else
        printf("enter the correct number");
        }
```

Forms of *if*

The **if** statement can take any of the following forms:

(a)
```
if ( condition )
    do this ;
```

(b)
```
if ( condition )
{
    do this ;
    and this ;
}
```

(c)
```
if ( condition )
    do this ;
else
    do this ;
```

(d)
```
if ( condition )
{
    do this ;
    and this ;
}
else
{
    do this ;
    and this ;
}
```

(e)
```
if ( condition )
    do this ;
else
{
    if ( condition )
        do this ;
    }
    else
    {
        do this ;
        and this ;
}
```

(f)
```
if ( condition )
{
    if ( condition )
        do this ;
    else
    {   do this ;
        and this ;
    }
}
else
    do this ;
}
```

Use of Logical Operators

C allows usage of three logical operators, namely, &&, || and !. These are to be read as 'AND' 'OR' and 'NOT' respectively.

There are several things to note about these logical operators. Mostobviously, two of them are composed of double symbols: **||** and **&&**. Don't use the single symbol **|** and **&**. These single symbols also have a meaning

Example 2.6: Write a program to calculate the salary as per the following table:

Gender	Years of Service	Qualifications	Salary
Male	>= 10	Post-Graduate	15000
	>= 10	Graduate	10000
	< 10	Post-Graduate	10000
	< 10	Graduate	7000
Female	>= 10	Post-Graduate	12000
	>= 10	Graduate	9000
	< 10	Post-Graduate	10000
	< 10	Graduate	6000

Figure 2.6

```
main( )
{
    char g ;
    int yos, qual, sal ;

   printf ( "Enter Gender, Years of Service and Qualifications ( 0 = G, 1 = PG ):" ) ;
    scanf ( "%c%d%d", &g, &yos, &qual ) ;

    if ( g == 'm' && yos >= 10 && qual == 1 )
        sal = 15000 ;
    else if ( ( g == 'm' && yos >= 10 && qual == 0 ) ||
        ( g == 'm' && yos < 10 && qual == 1 ) )
        sal = 10000 ;
```

```
    else if ( g == 'm' && yos < 10 && qual == 0 )
        sal = 7000 ;

    else if ( g == 'f' && yos >= 10 && qual == 1 )
        sal = 12000 ;

    else if ( g == 'f' && yos >= 10 && qual == 0 )
        sal = 9000 ;

    else if ( g == 'f' && yos < 10 && qual == 1 )
        sal = 10000 ;

    else if ( g == 'f' && yos < 10 && qual == 0 )
        sal = 6000 ;

        printf ( "\nSalary of Employee = %d", sal ) ;
}
```

The ! Operator

So far we have used only the logical operators **&&** and ||. The third logical operator is the NOT operator, written as **!**.

Ex:

```
If ( y !< 10 )
```

This means "not **y** less than 10". In other words, if **y** is less than10, the expression will be false, since **(y < 10)** is true. We can express the same condition as **(y >= 10)**.

Hierarchy of Operators Revisited

Operators	**Type**
!	Logical NOT
* / %	Arithmetic and modulus
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
||	Logical OR
=	Assignment

The Conditional Operators

The conditional operators **?** and **:** are sometimes called ternary operators since they take three arguments. In fact, they form a kindof foreshortened if-then-else. Their general form is,

expression 1 ? expression 2 : expression 3

What this expression says is: "if **expression 1** is true (that is, if its value is non-zero), then the value returned will be **expression 2**, otherwise the value returned will be **expression 3**". Let us understand this with the help of a few examples:

(a)
```
int x, y ;
scanf ( "%d", &x ) ;

y = ( x > 5 ? 3 : 4 ) ;
```

This statement will store 3 in **y** if **x** is greater than 5, otherwise it will store 4 in y.