Unit : 3 Variables and Constants, Expressions and Operators

Lesson Structure

3.0 Objective

This unit will help you to

- define identifiers, data types and keywords in C;
- name the identifiers as per the conventions;
- describe memory requirements for different types of variables;
- define constants, symbolic constants and their use in programs;
- understand the concept of different operators used in C;

3.1 Introduction

A program consist of certain sequence of instructions which are used for processing

of data to provide useful output known as information. These instruction are formed using certain symbols and words according to some rigid rules known as syntax rules. Every program instruction must confirm precisely to the syntax rules of the language.

Computer programs usually work with different types of data and need a way to store the values being used. C language has two ways of storing values i.e variables and constants. Constants and variables are the fundamental element of each program. A variable is a data storage location that has a value that can change during program execution whereas a constant refer to fixed value that cannot change during program execution.

3.2 Character Set

When you write a program, you express C source files as text lines containing characters from the character set. When a program executes in the target environment, it uses characters from the charater set.

Every character set contains a distinct code value for each character in the basic C character set. A character denotes any alphabates, digit or special symbol. The C language character set are as shown below :

1. Alphabets including both lower case and upper case alphabates - A-Z and a-z.
2. Numbers 0-9
3. Special characters include: ; : {,’ “ | } >< / \ ~ _ [ ] ! $? * + = ( ) - % # ^ @ &.

3.3 Identifiers and Keywords

Identifiers, as the name suggests helps us to identify data and other objects in the program. Identifiers are basically the names given to program elements such as constants, variables, function and arrays. Every element in the program has its own distinct name but one cannot select any name unless it confirms to valid name in C language. Let us study first the rules to define names or identifiers.

3.3.1 Rules for Forming Identifiers

Identifiers are defined according to the following rules:

1. It consists of letters and digits.
2. First character must be an alphabet or underscore.
3. Both upper and lower cases are allowed. Same text of different case is not equivalent, for example: TEMP is not same as temp.

4. Except the special character underscore (_), no other special symbols can be used.
5. There cannot be two successive underscores.
6. Identifiers can be of any reasonable length. they should not contain more than 31 characters. It can be longer but the compiler ignores the character after 31 characters.

Example of some valid identifiers are :

A , A123 , _ BI , temp,s _ name, dept _ code , rollno

Example of some invalid identifiers are shown below:

(i) 123: First digit not allowed.

(ii) % X: first special symbol not allowed.

(iii) roll-no : hyphen not allowed

(iv) first name : space not allowed

3.3.2 Keywords

Keywords are reserved words which have standard, predefined meaning in C. They cannot be used as program-defined identifiers.

The lists of C keywords are as follows :

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

Note : Generally all keywords are in lower case although upper case of same names can be used as identifiers.

3.4Data Types and Storage

To store data inside the computer we need to first identify the type of data elements that we need in our program. There are several different types of data, which may be represented differently within the computer memory.

The data type specifies two things:

1. Permissible range of values that it can store.
2. Memory requirement to store a data type.

C language supports three classes of data types :

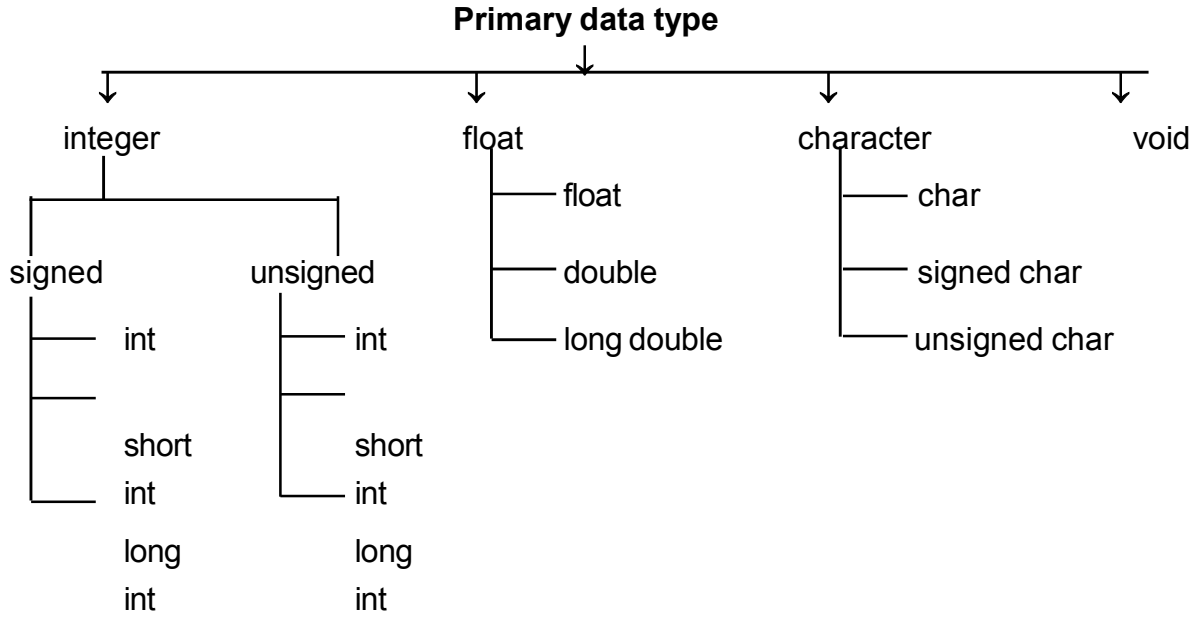
(i) Primary data types.

(ii) Derived data types.

(iii) User defined data types.

Primary data types

C Language provides four basic data types namely integer (int) , character (char) , floating (float) and void. the four basic data types are described below :



integer Types

integers are used to store whole numbers.

Size and range of integer type on 16-bit machine

Type	Size (bytes)	Range
int or signed int	2	-32,768 to 32,767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

floating Types

floating types are used to store real numbers,

Size and range of float type on 16-bit machine

Type	Size (bytes)	Range
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

character Types

character types are used to store characters value.

Size and range of character type on 16-bit machine

Type	Size (bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

void Type

void type means no value. This is usually used to specify the type of functions.

Note: Memory requirements or size of data associated with a data type indicates the range of numbers that can be stored in the data item of that type.

Derived data types

Derived data types are like arrays, functions, structures and pointers. These are dicussed in detail later.

User defined data types

User defined data types allows users to define an identifier that would represent an existing data type. User defined data types are typedef and enum.

typedef

It takes the general form :

```
typedef typc identifier;
```

Where type refers to an existing data type and “identifier” refers to the “new” name given to the existing data type. Some examples of type definition are :

```
typedef int marks;
```

typedef float salary;

Here, marks symbolizes int and salary symbolizes float. They can be later used to declare variables as follows :

```
marks m1, m2;
salary s[5];
```

enum

Another user-defined data type is enumerated data type provided by ANSI standard. It is defined as follows:

```
enum identifier { value1,value2,.........valuen};
```

Where " identifier" is a user-defined enumerateda data type which can be used to declare variables that can have one of the values enclosed within the braces.

An example:

```
enum day {Monday, Tuesday,.....Sunday};
```

The compiler automatically assigns integer digits beginning with 0 to all the enumeration constants.

However, the automatic assignments can be overridden by assigning values explicitly to the enumeration constants. For example:

```
enum day { Monday = 1, Tuesday,.........Sunday};
```

3.5Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable. All variables have three essential attributes :

- the name
- the value
- the memory, where the value is stored.

User defined variable must follow the following rules:

- First character must be a letter or underscore, remaining characters can be letters or digits or underscore.
- Lowercase and upper case letters are different.

- Although there is no restriction on the variable names, but only first 8 character are taken, so the first 8 character of two variable should be different.
- Variable should not be reserved word i.e. keyword.
- Each variable must be declared before its usage within the program, this declaration is achieved by linking variable name to its type.

3.5.1 Declaring Variables

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows :

type variable _list;

Here, **type** must be a valid C data type including char, int, float, double, or any user-defined object,etc. and **variable _ list** may consist of one or more identifier names seperated by commas. Some valid declarations are shown here:

```
int i, j, k;
char c,ch;
float f, salary;
double d;
```

The line int i, j, k; both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

In C variable can be declared at any place in the program but two things must be kept in mind are as following:

1. Variable must be declared before using them.
2. Variables should be declared closet to their first point of use to make the source code easier to maintain.

The three basic places where the variables are declared in C program are :

1. When a variable is declared inside a function it is known as a local variable.
2. When the variable is declared in the definition of function parameters it is known as formal parameter.
3. When the variable is declared outside all functions, it is known as a global variable.

3.5.2 Initializing Variables

Variables can be initialized (assigned an initial value) in their declaration. when variables are declared initial values can be assigned to them in two ways :

(a) Within a Type declaration

The value is assingned at the declaration time.

type variable_name = value;

Some example are :

```
int d = 3, f =5;          / / definition and initializing d and f.
float z = 22.50;          / / definition and initializes z.
char x = 'x';             / / the variable x has the value 'x'.
```

(b) Using Assignment statement

The values are assigned just after the declarations are made.

For example :

```
int x;
float z;
char y;
x = 210;
z = 22.50
 y = 'x';
```

3.6Constants

The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.

Constants can be of any of the basic data types like

- an integer constant,
- a floating constant,
- a charater constant, or
- a string literal.

The **Constants** are treated just like regular variables except that their values cannot be modified after their definition.

3.6.1 Integer and floating point constants

Integer and floating point contants are numeric constants and represent numbers.

Rules to form Integer and floating point constants

- No comma or blankspace is allowed in a constant.
- It can be preceded by - (Minus) sign if desired.
- The value should lie within a minimum and maximum permissible range decided by the word size of the computer.

Integer Constants

Further, these constant can be classified according to the base of the numbers as :

(i) Decimal integer constant

(ii) Octal integer constant

(iii) Hexadecimal integer constants

(iv) Unsigned integer constants

(v) Long Integer constants

Decimal integer constants

These consist of digits 0 through 9 and first digit should not be 0.

For example

Valid decimal integer constants.

8 , 943 ,12767

Invalid Decimal integer constants

82 ,65	(,) is not allowed
36.0	(.) is not allowed
5 0 410	Blankspace not allowed
20 - 100	(-) is not allowed
0987	The first digit should not be a zero

Octal integer constants

These consist of digits 0 through 7. The first digit must be zero order to identify the constant as an octal number.

For example:

Valid Octal integer constants are:

0 , 01, 0743, 0777

Invalid octal integer constants are:

744	is not valid as it does not begin with 0
0338	illegal character 8
0777.77	(.) is illegal char

Hexadecimal integer constants

These constants begin with ox or OX and are followed by combination of digits taken from hexadecimal digits 0 to 9, a to f or A to F.

Valid Hexadecimal integer constants are:

0x0, 0x1 , 0xf77, Oxabcd

Invalid Hexadecimal integer constants are :

OBEF	x is not incluided
Ox.4bff	illegal char (.)
OxgBC	illegal char g

Maximum values these constants can have are as following :

	Integer constants	**Maximum value**
i)	Decimal integer	32767
ii)	Octal integer	77777
iii)	Hexadecimal integer	7FFF

Unsigned integer constants:

Exceed the ordinary integer by magnitude of 2, they are not negative. A character U or u is prefixed to number to make it unsigned.

Long integer constants:

These are used exceed the magnitude of ordinary integers and are appended by L.

Valid unsigned and long integer constants are :

212	decimal
215u	decimal unsigned
0xFeeL	Hexadecimal long
65000U or 40000 u	decimal unsigned
13234667888L or 145637881	decimal long.
0123456L	octal long.
0777777U	octal unsigned.

Invalid unsigned and long integer are :

032UU	Illegal as suffix can not be repeated

Floating point Constants

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

It is a base 10 number containing decimal point or an exponent.

Examples of valid floating point numbers are :

000.89	5.611364	890000.1	0.000621
1.5877E + 4	0.004e-3		

Examples of invalid Floating point numbers are :

35	decimal or exponent required.
14,300.0	comma not allowed.
5E + 10.7	exponent is written after integer quantity.
7E 10	no blank space.

The magnitude of floating point numbers range 3.4E - 38 to a maximum of 3.4E + 38, through 0.0. they are taken as double precision numbers.

3.6.2 Character constants

Charater literals are enclosed in single quotes, e.g., ‘x’ and can be stored in a simple variable of char type.

A character literal can be a plain character (e.g., ‘x’) , an sequence (e.g., ‘ \t’), or a universal character (e.g., ‘ \u02CO’).

Charater constants have integer vlaues associated depending on the character set adopted for the computer. ASCII character set is in use which uses 7-bit code with 2^7 = 128 different charaters.

The digits 0-9 are having ASCII value of 48-56 and ‘A’ have ASCII value from 65 and ‘a’ having value 97 are sequentially ordered.

For example, ‘A’ has 65, Blank has 32

3.6.3 String literals

String literals or constants are enclosed in double quotes “ “. A string contains characters that are similar to character literals: plain characters, escape sequence, and universal characters.

You can break a long line into multiple lines using string literals and separting them using whitespaces.

Here are some examples of string literals. All the three forms are idential strings.

“hello, dear”

“hello, \

dear”

“hello, “ “d” “ear”

3.7Defining Constants

There are two simple ways in C to define constants:

1. Using symbolic constant
2. Using const keyboard

3.7.1 Symbolic Constants

Symbolic Constant is a name that substitues for a sequence of characters or a numeric constant, a charater constant or a string constant. when program is compiled each occurrence of a symbolic constant is replaced by its corresponding character sequence. the syntax is as follows :

define name text

Where

name: implies symbolic name in caps.

text : implies value or the text.

For example

#define MAX 100

#define TRUE 1

#define FALSE 0

#define SIZE 10

The # character is used for preprocessor commands. A preprocessor is a system program, which comes into action prior to compiler, and it replaces the replecement text by the actual text.

Advantages of using Symbolic constants are:

- They can be used to assign names to values
- Replacement of value has to be done at one place and wherever the name appears in the text it gets the value by execution of the preprocessor. This saves time if the symbolic constant appears 20 times in the program; it needs to be changed at one place only.

Example :

```
#include < stdio.h >
#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'
int main ()
{
int area;
area = LENGTH * WIDTH;
printf ("value of area : %d" , area);
printf ("%c", NEWLINE);
return 0;
 }
```

OUTPUT

```
Value of area : 50
```

3.7.2 The const Keyword

You can use const prefix to declare constants with a specific type as follows:

Const type variable = value ;

Example :

```
#include <stdio.h>
int main ()
{
const int LENGTH = 10;
const int WIDTH = 5
```

```
const char NEWLINE = '\n';
int area;
area = LENGTH * WIDTH;
printf ("Value of area : %d," area);
printf ("%c", NEWLINE);
return 0;
}
```

OUTPUT

```
Value of area : 50
```

Note that it is a good programing practice to define constants in CAPITALS.

3.8Escape Sequence

There are certain characters in C when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t).

Escape Sequence are some non-printable charaters that can be printed by preceding them with '\' backslash character. Within character constants and string literals, you can write a variety of escape sequences. Each escape sequence determines the code value for a single character.

You can use escape sequence to represent character codes:

- Which you cannot be otherwise write (such as \n)
- that can be difficult to read properly (such as \t)
- that might change value in different target character sets (such as \a)
- that must not change in value among different target environments (such as \o)

Here, you have a list of some of such escape sequence codes :

Escape sequence	Meaning
\ \	\ character
\'	'character
\''	" character
\?	?character
\a	Alert or bell

Escape sequence	Meaning
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\ooo	Octal number of one to three digits
\xhh	Hexadecimal number of one or more digits.

Following is the example to show few escape sequence characters:

```
#include <stdio.h>
int main ()
{
printf ("Hello \ t World \n\n");
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

OUTPUT

```
Hello          World
```

3.9Input/Output Statement in C

ANSI standard has defined many library functions for input and output in C language. Functions printf( ) and scanf() are the most commonly used to display out and take input respectively.

Let us discuss about them.

Printf ()

The function printf stands for print formatting and is used to display information required by the user and also prints the values of the variables. The printf function takes data values, convert them to text stream using formatting function specified in a control string and passes the resulting text to the standard output.

Syntax of printf function is :

printf ("format string", argument list);

Where:

- **Format string** may be a collection of **escape sequence** or /and conversion specification or \and string constant. The format string directs the printf function to display the entire text enclosed within the double quotes without any change.
  - **Conversion specification** is also a pair of character. It is preceded by % and followed by a quote which may be a character. The conversion specification describes the printf0 function that it could print some value at that location in the text. the Conversion characters supported by C are

Conversion character	Meaning
%d	Data item is displayed as a signed decimal integer.
%i	Data item is displayed as a single decimal integer.
%f	Data item is displayed as a floating-point value without an exponent.
%c	Data item is displayed as a single character.
%o	Data item is displayed as an octal integer, without a leading zero.
%s	Data item is displayed as string.
%u	Data item is displayed as an unsingned decimal integer.
%x	Data item is displayed as a hexadecimal integer, without a leading 0x.

- The argument list contains a list of variables separated by comma. The number of argument is not fixed; however corresponding to each argument there should be a format specifer. Inside the format string the number of argument should tally with the number of format specifier.

Example : if x, y, z is an integer and to display the sum of the two numbers i.e. x and y you may use printf ("The sum of x = %d and y = % is z = %d \n", x,y,z);

scanf()

The function scanf() stands for scan formatting and used to read formatted data from the keyboard. The scanf function takes a text stream from the keyboard, extracts and formats data from the stream according to a format control string and then stores the data in specified program variables.

Syntax of scanf function is

scanf ("format string", argument list);

Where :

- The format string must be a text enclosed in double quotes. It contains the information for interpreting the entire data for connecting it into internal representation in memory.

  Example : integer (%d), float (%f), character (%c) or string (%s).

- The argument list contains a list of variables each preceded by the address operator(&) and separated by comma. The number of argument is not fixed; however corresponding to each argument there should be a format specifier. Inside the format string the number of argument should tally with the number of format specifier.

Example: if i is an integer and j is a floating point number, to input these two numbers you may use scanf ("%d %f", &i, &j);

3.9.1 I/O of Integers in C

The example given below will help you to understand the way of using scanf and printf with integer value :

```
#include<stdio.h>
#include < conio.h > intmain ()
{
int c;              //c is variable of integer data type
printf("Enter a number\n");//printf() method to display text on screen
scanf("%d", &c); //%d denotes input of one integer value from keyboard
printf("Number=%d",c);
return 0;
}
```

OUTPUT

```
Enter a number
4
Number = 4
```

PROGRAM ILLUSTRATION

The scanf() function is used to take input from user. In this program, the user is asked a input and value is stored in variable c. Note the '&' sign before c. & c denotes the address of c and value is stored in that address.

3.9.2 I/O of Float in C

Example :

```
#include <stdio.h>
#include<conio.h>
int main()
{
        float a;
printf("Enter value:");
scanf("%f",&a);
printf("Value=%f",a); //%f is used for floats instead of %d
return 0;
}
```

OUTPUT

```
Enter value: 23.45
Value = 23.450000
```

PROGRAM ILLUSTRATION

Conversion format string "%f" is used for floats to take input and to display floating value of a variable.

3.9.3 I/O of Characters

Example:

```
#include <stdio.h>
#include<conio.h>
int main()
{
char var1;
```

```
printf("Enter character:");
scanf("%c",&var1);
printf("You entered %c,", var 1);
return 0;
}
```

OUTPUT

```
Enter character : g
You entered g.
```

PROGRAM ILLUSTRATION

Conversion format string "%c" is used in case of characters.

3.9.4 I/O of ASCII code

Example:

```
#include <stdio.h>
#include <conio.h>
int main()
{
char var1;
printf ("Enter character :");
scanf("%c", & var1);
printf ("You entered %c.\n", var1);/*\n is an escape sequence and is used to prints the
next line (performs work of enter).*/
printf("ASCII value is %d", var1);
return 0;
}
```

OUTPUT

```
Enter character :
g
You entered g
ASCII value is 103
```

PROGRAM ILLUSTRATION

When character is typed in the above program, the character itself is not recorded a numeric value (ASCII value) is stored. And when we displayed that value by using “%c”, that character is displayed.

When, ‘g’ is entered, ASCII value 103 is stored instead of g.

3.9.5 Variations in Output for Integer and Floats

Example :

```
#include <stdio.h>
#include<conio.h>
int main()
{
int x;
float y;
x = 9876;
y = 987.6543;
printf (“Case 1; % 6d\n”,x);/* Prints the number right justified within 6 columns */
printf (“Case 2; % 3d\n”,x);/* Prints the number to be right justified to 3 columns but,
there are 4 digits so number is not right justified */
printf (“Case 3; % 2f\n”,y);/* Prints the number rounded to two decimal places */
printf (“Case 4; % f\n”,y);/* Prints the number rounded to 0 decimal place, i.e, rounded
to integer */
printf (“Case 5; % e\n”,y);
/* Prints the number in exponential notation (scientific notation) */
getch();
return0;
}
```

OUTPUT

```
Case 1      :      9876
Case 2      :      9876
```

Case 3	:	987.65
Case 4	:	988
Case 5	:	9.876543 e + 002

PROGRAM ILLUSTRATION

Integer and floating-points can be displayed in different formats in C programming as in the above program.

3.10 Operators

The variables, constants are used in expressions and for writing an expression we need operators along with variables. An expression is a sequence of operators and operands that does one or a combination of the following :

- specifies the computation of a value
- designates an object or function

An operator performs an operation (evaluation) on one or more operands. An operand is a subexpression on which an operator acts.

An operator is a symbol that tells the complier to perform specific mathematical or logical manipulations. C language is rich in built-in operators and provides the following types of operators;

- Assignment Operators.
- Arithmetic Operators.
- Increment Operator (++) and Decrement Operator (--)
- Relational Operators or Comparison Operator.
- Logical Operators.
- Comma and conditional Operators.
- Bitwise Operators.
- Misc Operators

3.10.1 Assignment Operators

The function of this operator is to assign the values of values in variables on right hand side of an expression to variables on the left hand side.

The syntax of the assignment expression is as follows :

Variable = constant / variable / expression;

The data type of the variable on left hand side should match the data type of constant/ variable/expression on right hand side with a few exceptions where automatic type conversion are possible.

Some examples of assignment statements are as follows :

```
b = a ;          /* b is assigned the value of a */
b = 88 ;         /* b is assigned the value 88 */
b = a + 10 ;     /* b is assigned the value of expra a + 10 */
```

The expression on the right hand side of the assignment statement can be :

- an arithmetic expression;
- a relational expression;
- a logical expression;
- a mixed expression.

The above-mentioned expressions are different in terms of the type of operators connecting the variables and constants on the right hand side of the variable.

For example :

```
int a;
float b, c, a1, t;
a1 = (b + c)/2;          /* arithmetic expression */
a = b & & c;             /*logical expression */
a = (b + c) & & (b < c); /* mixed expression */
```

There are following assignment operators supported by C language :

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+ =	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
- =	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C – A

Operator	Description	Example
× =	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the results to left operand	C * = A is equivalent to C = C * A
/=	Divide AND assignment operator, it divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C/A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C % = A is equivalent to C = C % A
<< =	Left shift AND assignment operator	C << = 2 is same as C = C << 2
>> =	Right shift AND assignment operator	C >> = 2 is same as C = C >> 2
& =	Bitwise AND assignment operator	C & = 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C^ = 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C | = 2 is same as C = C |2

Example :

```
#include<stdio.h>
#include<conio.h>
int main()
{
int num1 = 3, num2 = 5;
printf ("\n num 1 = %d and num 2 = %d", num1 num2);
num1 + = num2 * 4 – 7;
printf ("\n After evaluation num 1 = %d and num 2 = %d", num 1, num 2);
return 0;
}
```

OUTPUT

```
num 1 = 3 and num 2 = 5
```

After evaluation num 1 = 16 and num 2 = 5

3.10.2 Arithmetic Operators

The operands in arithmetic expressions can be of integer, float, double type. In order to effectively develop C programs, it will be necessary for you to understand the rules that are used for implicit conversation of floating point and integer values in C.

They are mentioned below :

- An arithmetic operator between an integer and integer always yields an integer result.
- Operators between float and float yields a float result.
- Operator between integer and float yields a float result.

If the data type is double instead of float, then we get a result of double data type.

Following table shows all the arithmetic operators supported by language.Assume variable A holds 10 and variable B holds 20 then.

Operator	Description	Example
+	Add two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give - 10
*	Multiplies both operands	A * B will give 200
/	Divides numerator by de-numerator	B / A will give 2
%	Modulus Operator and reminder of after an integer division	B % A will give 0

Parenthese can be used in C expression in the same manner as algebraic expression.

For example, m * ((n + o) + (m*n)).

It may so happen that the type of the expression and the type of the variable on the left hand side of the assignment operator may not be same. In such a case the value for the expression is promoted or demoted depending on the type of the variable on left hand side of = (assignment operator).

For example, consider the following assignment statements:

```
int i;
float b;
i = 22.12;  / / first statement
b = 65;  / / second statement
```

In the first assignment statement, float (22.12) is demoted to int. Hence i gets the value 22. In the second statement int (65) is promoted to float, b gets 65.0.

The rules of arithmetic precedence are as follows :

1. **Parentheses are at the "highest level of precedence".**

   In case of nested parenthesis, the innermost parentheses are evaluated first.

   For example,

   ((8 + 4)* 2) / 6)

   The order of evaluation is given below.

   a) (8+4) is evaluated first i.e we get 12

   b) ((8+4) *2) is evaluated secondally i.e we get 12*2 = 24

   c) (((8 +4)*2) /6) is evaluated lastly to get the correct output i.e 24/6 = 4 is the correct result.

2. **Multiplication, Division and Modulas operators are evaluated next.**

   If an expression contains several multiplication, division and modulus operators, evaluation proceeds from left to right. These three are at the same level of precedence.

   For example,

   8*8+6*7

   The order of evaluation is given below.

   a) 8*8 is evaluated first i.e we get 64

   b) 6*7 is evaluated secondly i.e we get 42

   c) finally 8*8 + 6*7 is evaluated i.e 64 + 42 = 106 is the correct result.

3. **Addition, substraction are evaluated last.**

   If an expression contains several addition and subtraction operators, evaluation proceeds from left to right, Or the associativity is from left to right.

   For example,

   10 /5-6+6/2

   The order of evaluation is given below.

   a) 10/5 is evaluated first i.e we get 2.

   b) 6/2 is evaluated secondly i.e we get 3.

   c) 10 /5-6 is performed next i.e we get -4

   d) -4+6/2 i.e -4+3 is performed lastly i.e we get -1 as the correct output.

Example : C program using arithmetic operator

```
#include <stdio.h>
#include<conic.h>
int main ()
 {
int a, b , add, sub, mul; / / integer type variables
float divide     / / float type variable
printf (“Enter first number:”);
scanf (“%d”, &a);
printf (“Enter second number:”);
scanf (“%d”, &b);
add = ( a+b);
printf (“\n Addition of first and second number is = %d”, add);
sub = (a-b);
printf (“ \ n Substraction of second from first = %d”, sub);
mul = (a*b);
printf (“ \ n Multiplication of first and second number = %d”, mul);
divide = (float) (a/b); / / typecasting as division may result in decimal.
printf ( “\n Division of first number by second number = %. 2f”, divide);
getch();
return 0; / / program ended with zero errors
 }
```

OUTPUT

```
Enter first number : 1
Enter second number : 2
Addition of first and second number is =3
Substraction of second from first = -1
Multiplication of first and second number = 2
Division of first number by second number = 0.50
```

PROGRAM ILLUSTRATION

- Type casting has been used here in order to get the right results. In C language when you divide two integers, the result will be an integer. for example 3/2 will result into 1.

- The fundamental rule that is

  int / int = int.

  int / float = float

  float / int = float

Hence, Typecasting has been used to get the desired results in case of division.

3.10.3 Increment Operator (++) and Decrement Operator (– –)

Operator	Description	Example
++	Increments operator increases integer value by one	When A = 10, A++ will give 11
– –	Decrements operator decreasee integer value by one	When A = 10 A-- will give 9

The increment / decrement operator have two varients ----

i) Prefix

ii) Postfix

In a prefix expression (++a or --a) the operator is applied before an operand is fetched for compultion and thus, the altered value is used for the computation of the expression in which it occurs.

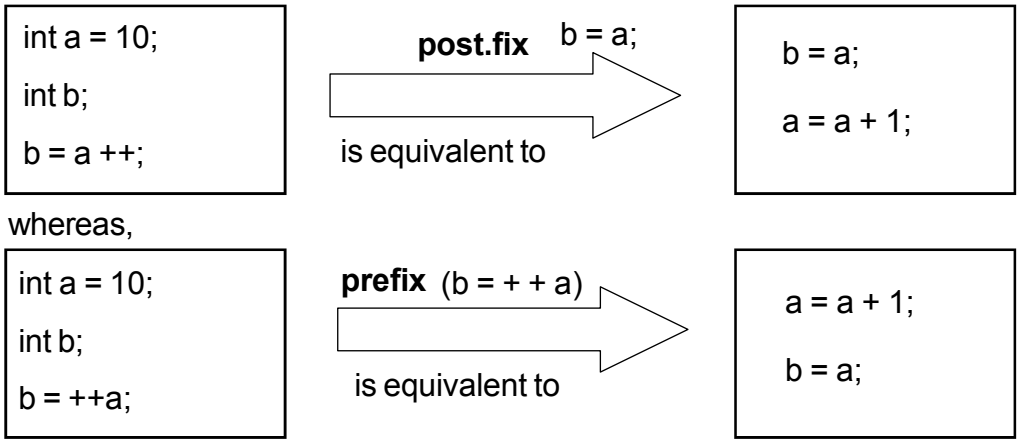
On the contrary, in a postfix expression (a ++or a--) an operator is applied after an operand is fetched for computation that is the unaltered value is used for the computation of the expression in which it occurs.

Example : C program to display the use of increment and decrement operator

```
#include <stdio.h>
int main ()
 {
int c = 2, d = 2.
printf (“%d\n”, c++);
```

```
// this stetement first displays 2 them, only c is incremented by 1 i.e in memory c store 3.
printf ("%d", ++c);
// this statement increments 1 to c in memory first then, only c is displayed.
```

For example :




```
return 0;
}
```

OUTPUT

```
2
4
```

3.10.4 Relational Operators and Comparison Operator

Following table shows all the relational operators supported by C language. Assume variable A holds 10 and variable B holds 20, then :

Operator	Description	Example
= =	Checks if the value of two operands are equal or not, if yes them condition becomes true.	(A = = B) is not true.
! =	Checks if the values of two oprands ar equal or not, if values are not equal then condition becomes true.	(A ! = B) is true
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left oprand is less than the value of right operand, if yes hten condition becomes true.	(A < B) is true.

Operator	Description	Example
>=	Checks if the vlue of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A>= B) is not true.
< =	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Example :

```
#include<stdio.h>
int main()
{
int num1 = 30;
int num 2 = 40;
printf ("value of %d > %d", num1, num2, num1>num2);
printf ("value of %d >=%d is %d," num1, num2, num1>=num2);
printf ("value of %d <=%d", num1,num2,num1<=num2);
printf ("value of %d < %d is %d", num1,num2,num1<num2);
printf ("value of %d = =%d is %d", num1,num2,num1= =num2);
printf ("value of %d ! =%d is %d", num1,num2,num1! = num2);
return (o);
}
```

OUTPUT

```
Value of 30>40 is 0
Value of 30>=40 is 0
Value of 30 <=40 is1
Value of 30 < 40 is 1
Value of 30 = = 40 is 0
Value of 30 ! = 40 is 1.
```

PROGRAM ILLUSTRATION

Whenever we use relational operators in printf statement then we get result of the expression either true or false i.e 1 or 0.

3.10.5 Logical Operators

Following table shows all the logical operators supported by C languaage. Assume variable A holds 1 and variable B holds 0, then;

Operator	Description	Example
& &	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A & & B) is false.
| |	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A | | B)is true.
!	Called Logical NOT Operator. Use to reverse the logical state of its operand. If a condition is true then Logical NOT operator will make false.	! (A& & B) is true.

Example :

```
#include<stdio.h>
int main ()
 {
int num1 = 30
int num2 = 40;
if (num1> = 40 | | num2>=40) / / Logical Or operator
printf (“Or operator Executed”);
if (num1>=20 & & num2> = 20 ) / / Logical AND operator
printf (“And operator Executed”,)
if (! ( num1> =40) ) / / Logical NOT operator
printf (“Not operator Executed”,)
return (0);
 }
```

OUTPUT

```
Or operator Executed
And operator Executed
Not operator Executed
```

3.10.6 Conditional Operators [? :] : Ternary Operator Statement in C

They are called as Ternary operator . They are also called as

? : operator. Ternary operators takes on 3 arguments

Syntax :

expression 1 ? expression 2 : expression 3

Where

expression 1 is Condition

expression 2 is statement followed if condition is true

expression 3 is statement followed if condition is false

Example :

```
#include<stdio.h>
int main()
 {
int num;
printf ("Enter the Number : \n ");
scanf ("%d, &num);
(num %2 = =0)? printf ("Even"):printf ("odd");
 }
```

OUTPUT

```
Enter the number:
4
Even
```

3.10.7 Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

P	Q	p &q	p|q	p^q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

.....

A & B = 0000 1100

A | B = 0011 1101

A ^ B = 0011 0001

~A = 1100 0011

The Bitewise operators supported by C language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then:

Operator	Description	Example
&	Binary AND Operator copies a true bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
|	Binary OR Operator copies a true bit if it exists in either operand.	(A | B) will give 61, which is 0011 1101
^	Binary XOR Operator copies the true bit if it is in one operand but not both.	(A ^ B) will give 49, which is 0011 0001
~	Binary ones Complemelt Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form.
<<	Binary Left shift operator. the left operands value is moved left by the number of bits specified by the right operand.	A <<2 will give 240 which is 1111 0000
>>	Binary Right shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >>2 will give 15 which is 0000 1111

Example :

```
#include<stdio.h>
int main () {
    unsingned char a = -8;
```

```
    unsigned char b=a >> 1;
    printf ("%d \n",b);
return 0;
}
```

OUTPUT

124

PROGRAM ILLUSTRATION

2's compliment of +8 is

1111 1000

Right shifting by 1 yields,

0111 1100 (124 in decimal)

3.10.8 Comma Operator

The comma operator in C takes two operands. It works by evaluating the first and discarding its value and then evaluates the second and returns the value as the result of the expression. Comma separated operands are evaluated in left-to-right sequence with the right most value yielding the result of the expression. Among all the operators, the comma operator has the lowest precedence.

For Example:

```
int a = 2, b = 3,x = 0;
x=(+ +a, b+ = a);
```

Now the value of x = 6

3.10.9 Misc Operators

There are few other important operator including size of and &; supported by C Language.

Examples :

Operator	Description	Example
sizeof()	Returns the size of an variable.	sizeof (a), where a is integer, will return 2.
&	Returns the address af an variable.	&a; will give actual address of the variable
*	Pointer to a variable.	*a; will pointer to a variable.

Example :

```
#include <stdio.h>
int main()
 {
int a;
float b;
double c;
char d;
printf (“size of int = %d bytes \n”,sizeof (a));
printf (“size of float = %d bytes \n”, sizeof (b));
printf (“size of double = %d bytes \n”, sizeof (c));
printf (“size of char = %d byte \n”, sizeof (d));
return 0;
 }
```

OUTPUT

```
Size of int = 2 bytes
Size of float = 4 bytes
Size of double =8bytes
Size of char = 1byte
```

3.11 Operators precendence in C

Operator precedence determines the grouping of terms in an expression. This effects how an expression is evaaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence that the addition operator.

For example x = 7+3 *2; here, x is assigned 13, not 20 because operator * has hiogher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] -> .++ - -	Left to right
Unary	+-! ~ ++--(type)*&size of	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	< <> >	Left to right
Relational	<<= >>=	Left to right
Equality	= = ! =	Left to right
Bitwise AND	&	Left to right
Bitwist XOR	^	Left to right
Bitwist OR	|	Left to right
Logical AND	& &	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	=+= -=*= /=%=>>=<<= &= ^= |=	Right to left
Comma	,	Left to right

3.11Summary

In the above chapter we have learnt certain basics which are required to learn a computer language and form a basis for all languages. Character set includes alphabets, numeric characters, special characters and some graphical characters which are used to form words in C language or names or identifiers. The identifiers which change their values during execution of the program is known as variable. Keywords are reserved word with specific meaning and cannot be used otherwise. C languages uses four basic data types - int, char, float and double. The constants are the fixed values and may be either integer or floating point or character os string type. Symbolic constants are used to deine names used for constant values. some qualifiers are used as prefixes to data types like signed, unsigned, short, and long. This unit also contains the detail of scanf () and printf() with their syntax and use.

The different types of operators present in C, namely arithmetic, relational, logical etc are also discussed in this unit and also their use in processing. Type conversions are very important to understand because sometimes a programmer gets unexcepted results (logical

error) which are most often cause by type conversions in case user has used improper types or if he has not type cast to desired type. Increment / decrement operator reduce a bit of coding when used in expressions. Logical operators are used further in all types of looping constructs.

3.12Questions for Exercise

1. Explain the terms variables and constants? How many type of variables are supported by C?
2. Write a short note on operators available in language.
3. Write a short note on basic data types that the C language supports.
4. Write short notes on printf() and scanf() functions.
5. Write a program that prints a floating point value in exponential format with the following specification:
   a) Correct to two decimal places;
   b) Correct to four decimal places;
   c) Correct to eight decimal places.
6. Write a program to read 10 integers. Displays these numbers by printing three numbers in a line seprated by commas.
7. Write a program to calculate simple interest and compound interest.
8. Write a program to calculate salary of an employee, given his basic pay (to be entered by the user), HRA, = 10% of basic pay, TA=5% of basic pay.
9. Give the output of the following programs.
   i)
```
#include<stdio.h>
int main ()
 {
int x = 3, y = 5, z = 7;
int a,b
a = x *2 + y / 5 - z * y;
b = ++x* (y-3) / 2- z++ * Y;
printf (" \n a =%d",a);
printf ("\n b=%d",b);
return 0;}
```

ii)
```
#include<stdio.h>
int main()
{
int a = 4
printf (" \n %d", 10 + a++);
printf (" \ n%d ",10 + ++a);
return 0;}
```

3.13Suggested Reading

1. Computer science A structed programming approach using C second Edition, Behruouza A. Forouzan, Richard F. Gilberg, Brooks / Cole, Thomson Learning
2. The C programming Language, Kernighan & Richie, PHI publication. 2. Programming with C, second Edition, Byron Gottfried, Tata Mc Graw Hill 2003.
3. The C Complete Reference, Fourth Edition, Herbert Schi;ldt, Tata Mc Graw Hill, 2002.
4. Programming with ANSI and Turbo C, Ashok N. Kamthane, pearson Education Asia, 2002.

Reference Links :

1. www.programiz.com /c-programming /c-operators
2. www.c4learn.com /c- programming
3. http: / / www.thegeekstuff.com

●●●