UNIT 2 DATA TYPES, OPERATORS AND EXPRESSIONS

Structure

2.0 INTRODUCTION

As every natural language has a basic character set, computer languages also have a character set, rules to define words. Words are used to form statements. These in turn are used to write the programs.

Computer programs usually work with different types of data and need a way to store the values being used. These values can be numbers or characters. C

language has two ways of storing number values—**variables and constants**—with many options for each. Constants and variables are the fundamental elements of each program. Simply speaking, a program is nothing else than defining them and manipulating them.

A variable is a data storage location that has a value that can change during program execution. In contrast, a constant has a fixed value that can't change.

This unit is concerned with the basic elements used to construct simple C program statements. These elements include the C character set, identifiers and keywords, data types, constants, variables and arrays, declaration and naming conventions of variables.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- define identifiers, data types and keywords in C;
- know name the identifiers as per the conventions;
- describe memory requirements for different types of variables;
- define constants, symbolic constants and their use in programs.write and evaluate arithmetic expressions;
- express and evaluate relational expressions;
- write and evaluate logical expressions;
- write and solve compute complex expressions (containing arithmetic, relational and logical operators), and
- use simple conditions using conditional operators.

2.2 C LANGUAGE 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 character set. These character sets are related, but need not have the same encoding or all the same members.

Every character set contains a distinct code value for each character in the **basic C character set**. A character set can also contain additional characters with other code values. The C language character set has alphabets, numbers, and special characters as shown below:

1. Alphabets including both lowercase and uppercase alphabets - A-Z and a-z.
2. Numbers 0-9
3. Special characters include:

;	:	{	,	‘	“	|
}	>	<	/	\	~	_
[	]	!	$	?	*	+
=	( )	-	%	#	^	@ & .

2.3 IDENTIFIERS AND KEYWORDS

Identifiers are the names given to various program elements such as constants, variables, function names and arrays etc. Every element in the program has its own distinct name but one cannot select any name unless it conforms to valid name in C language. Let us study first the rules to define names or identifiers.

2.3.1 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: **TEXT** is not same as **text**.
4. Except the special character underscore (_), no other special symbols can be used.

For example, some valid identifiers are shown below:

```
 Y
X123
_XI
temp
tax_rate
```

For example, some invalid identifiers are shown below:

123	First character to be alphabet
"X."	. not allowed
order-no	Hyphen not allowed
error flag	Blank space not allowed

2.3.2 Keywords

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

The list of keywords in C language are as follows:

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

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

2.4 DATA TYPES AND STORAGE

To store data inside the computer we need to first identify the type of data elements 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 provides four basic data types viz. int, char, float and double. Using these, we can store data in simple ways as single elements or we can group them together and use different ways (to be discussed later) to store them as per requirement. The four basic data types are described in the following table 2.1:

Table 2.1: Basic Data Types

DATA TYPE	TYPE OF DATA	MEMORY	RANGE
int	Integer	2 Bytes	– 32,768 to 32,767
char	character	1 Byte	– 128 to 128
float	Floating point number	4 bytes	3.4e – 38 to 3.4e +38
double	Floating point number with higher precision	8 bytes	1.7e – 308 to 1.7e + 308

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.

2.5 DATA TYPE QUALIFIERS

Short, long, signed, unsigned are called the data type qualifiers and can be used with any data type. A *short int* requires less space than *int* and *long int* may require more space than *int*. If *int* and *short int* takes 2 bytes, then *long int* takes 4 bytes.

Unsigned bits use all bits for magnitude; therefore, this type of number can be larger. For example ***signed int*** ranges from –32768 to +32767 and ***unsigned int*** ranges from 0 to 65,535. Similarly, ***char*** data type of data is used to store a character. It requires 1 byte. ***Signed char*** values range from –128 to 127 and ***unsigned char*** value range from 0 to 255. These can be summarized as follows:

Data type	Size (bytes)	Range
Short int or int	2	–32768 to 32,767
Long int	4	–2147483648 to 2147483647
Signed int	2	–32768 to 32767
Unsigned int	2	0 to 65535
Signed char	1	–128 to 127
Unsigned char	1	0 to 255

2.6 VARIABLES

Variable is an identifier whose value changes from time to time during execution. It is a named data storage location in your computer's memory. By using a variable's name in your program, you are, in effect, referring to the data stored there. 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. Note that a value must be assigned to the variables at some point of time in the program which is termed as assignment statement. The variable can then be accessed later in the program. If the variable is accessed before it is assigned a value, it may give garbage value. The data type of a variable doesn't change whereas the value assigned to can change. All variables have three essential attributes:

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

2.7 DECLARING VARIABLES

Before any data can be stored in the memory, we must assign a name to these locations of memory. For this we make declarations. Declaration associates a group of identifiers with a specific data type. All of them need to be declared before they appear in program statements, else accessing the variables results in junk values or a diagnostic error. The syntax for declaring variables is as follows:

data- type variable-name(s);

For example,

```
int  a;
short int   a, b;
int  c, d;
long  c, f;
float  r1, r2;
```

2.8 INITIALISING VARIABLES

Variable initialization means assigning a value to the variable. Initial values can be assigned to them in two ways:

a) Within a Type Declaration

The value is assigned at the declaration time.

For example,

```
int        a = 10;
float      b = 0.4 e –5;
char       c = 'a';
```

b) Using Assignment Statement

The values are assigned just after the declarations are made.

For example,
int a;
float b;
char c;

a = 10;
b = 0.4 e –5;
c = 'a';

Check Your Progress 1

1) Identify keywords and valid identifiers among the following:

hello	function	day-of-the-week
student_1	max_value	"what"
1_student	int	union

……

……

……

……

……

2) Declare variables roll no, total_marks and percentage with appropriate datatypes.

……

……

……

……

……

3) How many byte(s) are assigned to store for the following?
a) Unsigned character b) Unsigned integer c) Double

……

……

……

……

……

2.9 CONSTANTS

A constant is an identifier whose value cannot be changed throughout the execution of a program whereas the variable value keeps on changing. In C there are four basic types of **constants**. They are:

1. Integer constants

2. Floating point constants
3. Character constants
4. String constants

Integer and Floating Point constants 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.

2.9.1 Integer Constants

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

1. **Decimal integer constants**

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

   For example,
   1 443 32767
   are valid decimal integer constants.

2. **Invalid Decimal integer Constants**

12 ,45	, not allowed
1 010	Blankspace not allowed
10 – 10	– not allowed
0900	The first digit should not be a zero

3. **Octal integer constants**

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

   Valid octal integer constants are:

   0 01 0743 0777

   Invalid octal integer constants are:

743	does not begin with 0
0438	illegal character 8
0777.77	illegal char

4. **Hexadecimal integer constants**

   To specify a hexadecimal integer constant, start the hexadecimal sequence with a 0 followed by the character X (or x). Follow the X or x with one or more hexadecimal characters (the digits 0 to 9 and the upper or lowercase letters A to F). The value of a hexadecimal constant is computed in base 16 (the letters A to F have the values 10 to 15, respectively).

Valid Hexadecimal integer constants are:

0X0 0X1 0XF77 0xABCD

Invalid Hexadecimal integer constants are:

0BEF	x is not included
0x.4bff	illegal char (.)
0XGBC	illegal char G

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

Long Integer constants: These are used to exceed the magnitude of ordinary integers and are appended by L.

For example,

50000U	decimal unsigned
1234567889L	decimal long
0123456L	octal long
0777777U	octal unsigned

2.9.2 Floating Point Constants

A floating-point constant consists of:

- Decimal integer
- Decimal point
- Decimal fraction
- e or E and a signed integer exponent (optional)
- Type suffix: f or F or l or L (optional)

Either decimal integer or decimal fraction (but not both) can be omitted. Either decimal point or letter e (or E) with a signed integer exponent (but not both) can be omitted. These rules allow conventional and scientific (exponent) notations.

Negative floating constants are taken as positive constants with an unary operator minus (-) prefixed. If there is a need for a floating-point constant that exceeds these limits, user should add l or L suffix, making the constant a long double type.

Here are some examples:

0.	// is equal to 0.0
-1.23	// is equal to -1.23
23.45e6	// is equal to 23.45 X 10^6
2e-5	//is equal to 2.0 X 10^{-5}
3e+10	//is equal to 3.0 X 10^{10}
.09E34	//is equal to 0.09 X 10^{34}

2.9.3 Character Constants

This constant is a single character enclosed in apostrophes (‘ ’) .

For example, some of the character constants are shown below:

'A', 'x', '3', '$'

'\0' is a null character having value zero.

Character constants have integer values 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 characters. The digits 0-9 have ASCII values of 48-57, upper case alphabets from 'A' to 'Z' have ASCII values from 65 to 90 and lower case alphabets 'a' to 'z' have ASCII values from 97 to 122.

Escape Sequence

Many programming languages support a concept called Escape Sequence. When a character is preceded by a backslash (\), it is called an escape sequence and it has a special meaning to the compiler. For example, \n in the following statement is a valid character and it is called a new line character in C language.

2.9.4 String Constants

It consists of sequence of characters enclosed within double quotes. For example,

"red" "Blue Sea" "41213*(I+3)"

2.10 SYMBOLIC CONSTANTS AND OTHERS

Symbolic Constant is a name that substitutes for a sequence of characters or a numeric constant, a character 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.

Examples:

```
#define   printf    print
#define   MAX    100
#define   TRUE   1
#define   FALSE  0
#define    SIZE     10
#define PI 3.141592
```

The # character is used for preprocessor commands. A ***preprocessor*** is a system program, which comes into action prior to Compiler, and it replaces the replacement text by the actual text. This will allow correct use of the statement printf.

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.

Enumerated Data Type

An enumerated type is used to specify the possible values of an object from a predefined list. Elements of the list are called *enumeration constants*. The main use of enumerated types is to explicitly show the symbolic names, and therefore the intended purpose, of objects whose values can be represented with integer values. Objects of enumerated type are interpreted as objects of type signed int , and are compatible with objects of other integral types.

The compiler automatically assigns integer values to each of the enumeration constants, beginning with 0. The following example declares an enumerated object background_color with a list of enumeration constants:

enum colors {black,red,blue,green,white} background_color;

Later in the program, a value can be assigned to the object background_color :

background_color = white;

In this example, the compiler automatically assigns the integer values as follows: black = 0, red = 1, blue = 2, green = 3, and white = 4. Alternatively, explicit values can be assigned during the enumerated type definition:

enum colors { black = 5, red = 10, blue, green = 7, white = green+2 };

Here, black equals the integer value 5, red = 10, blue = 11, green = 7, and white = 9. Note that blue equals the value of the previous constant (red) plus one, and green is allowed to be out of sequential order.

Because the ANSI C standard is not strict about assignment to enumerated types, any assigned value not in the predefined list is accepted without complaint.

Typedef in C Language

typedef keyword is used to assign a new name to a type. This is used just to prevent us from writing more.

For example, if we want to declare some variables of type unsigned int, we have to write *unsigned int* in a program and it can be quite hectic for some of us. So, we can assign a new name of our choice for *unsigned int* using **typedef** which can be used anytime we want to use unsigned int in a program.

typedef current_name new_name;

typedef unsigned in uint;

uint j,k;

Now, we can write ***uint*** in the whole program instead of unsigned int. The above code is the same as writing:

unsigned int j,k;

For example,

```
#include<stdio.h>
int main()
{
typedef unsigned in uint;
uint j=5, k=9;
printf("j=  %d\n",j);
printf("k= %d\n",k);
return 0;
}
```

Check Your Progress 2

1) Write a preprocessor directive statement to define a constant PI having the value 3.14.

……

……

……

……

……

2) Classify the examples into Integer, Character and String constants.

'A'	0147	0xEFH
077.7	"A"	26.4
"EFH"	'\r'	abc

……

……

……

……

……

3) Name different categories of constants C programming language.

……

……

……

……

……

……

……

2.11 EXPRESSIONS AND OPERATORS - AN INTRODUCTION

In the previous sections' we have learnt variables, constants, datatypes and how to declare them in C programming. The next step is to use those variables in expressions. 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
- generates side effects.

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

This unit focuses on different types of operators available in C including the syntax and use of each operator and how they are used in C.

A computer is different from calculator in a sense that it can solve logical expressions also. Therefore, apart from arithmetic operators, C also contains logical operators. Hence, logical expressions are also discussed in the following sections.

2.12 ASSIGNMENT STATEMENT

In the previous unit, we have seen that variables are basically memory locations and they can hold certain values. But, how to assign values to the variables? C provides an assignment operator for this purpose. The function of this operator is to assign the values or 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 conversions are possible. Some examples of assignment statements are as follows:

```
b  = a ;       /* b is assigned the value of a */
b = 5 ;        /* b is assigned the value 5*/
b = a+5;    /* b is assigned the value of expr  a+5 */
```

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. Arithmetic operators, relational operators and logical operators are discussed in the following sections.

For example,

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

2.13 ARITHMETIC OPERATORS

The basic arithmetic operators in C are the same as in most other computer languages, and correspond to our usual mathematical/algebraic symbolism. The following arithmetic operators are present in C:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular Division

Some of the examples of algebraic expressions and their C notation are given below:

Expression	C notation
$\frac{b*g}{d}$	(b *g) / d
a^3+cd	(a*a*a) + (c*d)

The arithmetic operators are all binary operators i.e. all the operators have two operands. The integer division yields the integer result. For example, the expression 10/3 evaluates to 3 and the expression 15/4 evaluates to 3. C provides the modulus operator, %, which yields the reminder after integer division. The modulus operator is an integer operator that can be used only with integer operands. The expression x%y yields the reminder after x is divided by y. Therefore, 10%3 yields 1 and 15%4 yields 3. An attempt to divide by zero is undefined on computer system and generally results in a run- time error. Normally, Arithmetic expressions in C are written in straight-line form. Thus ‘a divided by b’ is written as a/b.

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 conversion 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.
- Operator 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.

For example,

Operation	Result
5/3	1
5.0/3	1.6666666667
5/3.0	1.6666666667
5.0/3.0	1.6666666667

Parentheses can be used in C expression in the same manner as algebraic expression For example,

```
a*(b + c)
```

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 = 4.6;
b = 20;
```

In the first assignment statement, float (4.6) is demoted to int. Hence *i* gets the value 4. In the second assignment statement int (20) is promoted to float, *b* gets 20.0. If we have a complex expression like:

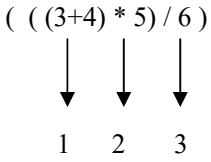
```
float   a, b, c;
int   s;
s = a * b / 5.0 * c;
```

Where some operands are integers and some are float, then int will be promoted or demoted depending on left hand side operator. In this case, demotion will take place since s is an integer.

The rules of arithmetic precedence are as follows:

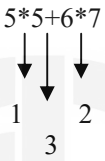
1. Parentheses are at the “highest level of precedence”. In case of nested parenthesis, the innermost parentheses are evaluated first.

For example,
(((3+4)*5)/6)
The order of evaluation is given below:



2. Multiplication, Division and Modulus 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.

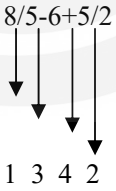
For example,
5*5+6*7
The order of evaluation is given below.



3. Addition, subtraction 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,

8/5-6+5/2
The order of evaluation is given below.



Apart from these binary arithmetic operators, C also contains two unary operators referred to as increment (++) and decrement (--) operators, which we are going to be discussed below:

The two-unary arithmetic operators provided by C are:

- ***Increment operator*** **(++)**
- ***Decrement operator*** **(- -)**

The increment operator increments the variable by one and decrement operator decrements the variable by one. These operators can be written in two forms i.e. before a variable or after a variable. If an ***increment / decrement*** operator is written before a variable, it is referred to as

preincrement / predecrement operators and if it is written after a variable, it is referred to as ***post increment / postdecrement*** operator.

For example,

a++ or ++a is equivalent to a = a+1 and
a-- or - -a is equivalent to a = a -1

The importance of ***pre*** and ***post*** operator occurs while they are used in the expressions. ***Preincrementing (Predecrementing)*** a variable causes the variable to be incremented (decremented) by 1, then the new value of the variable is used in the expression in which it appears***. Postincrementing (postdecrementing)*** the variable causes the current value of the variable is used in the expression in which it appears, then the variable value is incremented (decrement) by 1.

The explanation is given in the table below:

Expression	**Explanation**
++a	Increment a by 1, then use the new value of a
a++	Use value of a, then increment a by 1
--b	Decrement b by 1, then use the new value of b
b--	Use the current value of b, then decrement by 1

The precedence of these operators is right to left. Let us consider the following examples:

```
int a = 2, b=3;
int c;
c = ++a  –  b- -;
printf (“a=%d, b=%d,c=%d\n”,a,b,c);
```

OUTPUT

a = 3, b = 2, c = 0.

Since the precedence of the operators is right to left, first b is evaluated, since it is a post decrement operator, current value of b will be used in the expression i.e. 3 and then b will be decremented by 1.Then, a preincrement operator is used with a, so first a is incremented to 3. Therefore, the value of the expression is evaluated to 0.

Let us take another example,

```
int  a = 1, b = 2, c = 3;
int k;
k = (a++)*(++b) + ++a - --c;
printf(“a=%d,b=%d, c=%d, k=%d”,a,b,c,k);
```

OUTPUT

a = 3, b = 3, c = 2, k = 6

The evaluation is explained below:

```
k  =  (a++) * (++b)+ ++a  - --c
   =  (a++) * (3) + 2 - 2     step1
   =  (2) * (3) + 2 - 2       step2
   =  6                       final result
```

Check Your Progress 3

1. Give the C expressions for the following algebraic expressions:

   i) $\frac{a*4c^2 - d}{m+n}$

   ii) $ab - (e+f)\frac{4}{c}$

……

……

……

……

……

2. Give the output of the following C code:

```
main()
{
   int a=2,b=3,c=4;
   k = ++b +  --a*c + a;
   printf("a= %d b=%d c=%d k=%d\n",a,b,c,k);
}
```

……

……

……

……

……

3. Point out the error:

```
exp = a**b;
```

……

……

……

……

……

2.14 RELATIONAL OPERATORS

Executable C statements either perform actions (such as calculations or input or output of data) or make decision. Using relational operators we can compare two variables in the program. The C relational operators are summarized below, with their meanings. Pay particular attention to the equality operator; it consists of two equal signs, not just one. This section introduces a simple version of C's **if** control structure that allows a program to make a decision based on the result of some condition. If the condition is true then the statement in the body of if statement is executed else if the condition is false, the statement is not executed. Whether the body statement

is executed or not, after the if structure completes, execution proceeds with the next statement after the if structure. Conditions in the **if** structure are formed with the relational operators which are summarized in the Table 2.2.

Table 2.2: Relational Operators in C

Relational Operator	Condition	Meaning
==	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 less than or equal to y
>	x>y	x is greater than y
>=	x>=y	x is greater or equal to y

Relational operators usually appear in statements which are inquiring about the truth of some particular relationship between variables. Normally, the relational operators in C are the operators in the expressions that appear between the parentheses.

For example,

i) if (thisNum < minimumSoFar) minimumSoFar = thisNum

ii) if (job == Teacher) salary == minimumWage

iii) if (numberOfLegs != 8) thisBug = insect

iv) if (degreeOfPolynomial < 2) polynomial = linear

Let us see a simple C program given below containing the ***if statement*** (will be introduced in detail in the next unit). It displays the relationship between two numbers read from the keyboard.

```
/*Program to find relationship between two numbers*/
#include <stdio.h>
main()
{
int a, b;
printf("Please enter two integers: ");
scanf ("%d%d", &a, &b);
if (a <= b)
printf(" %d <= %d\n",a,b);
else
printf("%d > %d\n",a,b);
}
```

OUTPUT

```
Please enter two integers: 12 17
12 <= 17
```

We can change the values assigned to a and b and check the result.

2.15 LOGICAL OPERATORS

Logical operators in C, as with other computer languages, are used to evaluate expressions which may be true or false. Expressions which involve logical operations are evaluated and found to be one of two values: **true or false**. So far we have studied simple conditions. If we want to test multiple conditions in the process of making a decision, we have to perform simple tests in separate IF statements (will be introduced in detail in the next unit). C provides logical operators that may be used to form more complex conditions by combining simple conditions.

The logical operators are listed below:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Thus logical operators (AND and OR) combine two conditions and logical NOT is used to negate the condition i.e. if the condition is true, NOT negates it to false and vice versa.Let us consider the following is:

i) Suppose the grade of the student is 'B' only if his marks lie within the range 65 to 75,if the condition would be:

```
if((marks >=65) && (marks <= 75))
printf ("Grade is B\n");
```

ii) Suppose we want to check that a student is eligible for admission if his PCM is greater than 85% or his aggregate is greater than 90%, then,

```
if((PCM >=85) ||(aggregate >=90))
printf ("Eligible for admission\n");
```

Logical negation (!) enables the programmer to reverse the meaning of the condition. Unlike the && and || operators, which combines two conditions (and are therefore Binary operators), the logical negation operator is a unary operator and has one single condition as an operand. Let us consider an example:

```
if!(grade=='A')
printf ("the next grade is %c\n", grade);
```

The parentheses around the condition grade==A are needed because the logical operator has higher precedence than equality operator. The truth table of the logical AND (&&), OR (||) and NOT (!) are given below.

These table show the possible combinations of zero (false) and nonzero (true) values of x (expression1) and y (expression2) and only one expression in case of NOT operator. The following table 2.3 is the truth table for && operator.

Table 2.3: Truth table for && operator

x	y	x&&y
zero	zero	0
Non zero	zero	0
zero	Non zero	0
Non zero	Non zero	1

The following table 2.4 is the truth table for || operator.

Table 2.4: Truth table for || operator

x	y	x || y
zero	zero	0
Non zero	zero	1
zero	Non zero	1
Non zero	Non zero	1

The following table 2.5 is the truth table for ! operator.

Table 2.5: Truth table for ! operator

x	! x
zero	1
Non zero	0

The following table 2.6 shows the operator precedence and associativity

Table 2.6: (Logical operators precedence and associativity)

Operator	Associativity
!	Right to left
&&	Left to right
||	Left to right

2.16 COMMA AND CONDITIONAL OPERATORS

Conditional Operator

C provides an called as the conditional operator (**?:**) or else called as ***ternary*** operator which is closely related to the **if/else** structure. The conditional operator is C's only ternary operator - it takes three operands. The operands together with the conditional operator form a conditional expression. The first operand is a condition, the second operand represents the value of the entire conditional expression it is the condition is true and the third operand is the value for the entire conditional expression if the condition is false.

The syntax is as follows:

(condition)? (expression1): (expression2);

If condition is true, expression1 is evaluated else expression2 is evaluated. Expression1/Expression2 can also be further conditional expression i.e. the case of nested if statement (will be discussed in the next unit).

Let us see the following examples:

i) x= (y<20) ? 9: 10;

This means, if (y<20), then x=9 else x=10;

ii) printf ("%s\n", grade>=50? "Passed": "failed");

The above statement will print "passed" grade>=50 else it will print "failed"

iii) (a>b) ? printf ("a is greater than b \n"): printf ("b is greater than a \n");

If a is greater than b, then first printf statement is executed else second printf statement is executed.

Comma Operator

A comma operator is used to separate a pair of expressions. A pair of expressions separated by a comma is evaluated left to right, and the type and value of the result are the value of the type and value of the right operand. All side effects from the evaluation of the left operand are completed before beginning evaluation of the right operand. The left side of comma operator is always evaluated to void. This means that the expression on the right hand side becomes the value of the total comma-separated expression. For example,

```
x = (y=2, y - 1);
```

first assigns y the value 2 and then x the value 1. Parenthesis is necessary since comma operator has lower precedence than assignment operator.

Generally, comma operator (,) is used in the for loop (will be introduced in the next unit)

For example,

```
for(i= 0,j= n;i<j; i++,j--)
{
  printf("A");
}
```

In this example **for** is the looping construct (discussed in the next unit). In this loop, i = 0 and j = n are separated by comma (,) and i++ and j—are separated by comma (,). The example will be clear to you once you have learnt for loop (will be introduced in the next unit).

Essentially, the comma causes a sequence of operations to be performed. When it is used on the right hand side of the assignment statement, the value assigned is the value of the last expression in the comma-separated list.

Check Your Progress 4

1. Given a=3, b=4, c=2, what is the result of following logical expressions:

   (a < --b) && (a==c)

   ……..

   ……..

   ……..

   ……..

   ……..

2. Give the output of the following code:

```
main()
{
  int a=10, b=15,x;
  x = (a<b)?++a:++b;
  printf("x=%d a=%d b=%d\n",x,a,b);
}
```

   ……..

   ……..

   ……..

   ……..

   ……

3. What is the use of comma operator?

   ……..

   ……..

   ……..

   ……..

   ……

2.17 TYPE CAST OPERATOR

We have seen in the previous sections and last unit that when constants and variables of different types are mixed in an expression, they are converted to the same type. That is automatic type conversion takes place. The following type conversion rules are followed:

1. All chars and **short ints** are converted to **ints**. All floats are converted to doubles.
2. In case of binary operators, if one of the two operands is a **long double**, the other operand is converted to **long double**,

else if one operand is **double**, the other is converted to **double**,

else if one operand is **long**, the other is converted to **long**,

else if one operand is **unsigned**, the other is converted to **unsigned**,

C converts all operands "up" to the type of largest operand (largest in terms of memory requirement for e.g. **float** requires 4 bytes of storage and **int** requires 2 bytes of storage so if one operand is **int** and the other is **float**, **int** is converted to **float**).

All the above mentioned conversions are automatic conversions, but what if **int** is to be converted to **float.** It is possible to force an expression to be of specific type by using operator called a ***cast***. The syntax is as follows:

(type) expression

where *type* is the standard C data type. For example, if you want to make sure that the expression a/5 would evaluate to type **float** you would write it as

(float) a/5

cast is an unary operator and has the same precedence as any other unary operator. The use of ***cast*** operator is explained in the following example:

```
main()
{
    int   num;
    printf("%f %f %f\n", (float)num/2, (float)num/3, float)num/3);
}
```

Tha ***cast*** operator in this example will ensure that fractional part is also displayed on the screen.

2.18 SIZE OF OPERATOR

C provides a compile-time unary operator called ***sizeof*** that can be used to compute the size of any object. The expressions such as:

sizeof object and ***sizeof(type name)***

result in an unsigned integer value equal to the size of the specified object or type in bytes. Actually the resultant integer is the number of bytes required to store an object of the type of its operand. An object can be a variable or array or structure. An array and structure are data structures provided in C, introduced in latter units. A type name can be the name of any basic type like **int** or **double** or a derived type like a structure or a pointer.

For example,

sizeof(char) = 1bytes

sizeof(int) = 2 bytes

2.19 C SHORTHAND

C has a special shorthand that simplifies coding of certain type of assignment statements. For example: a = a+2;

can be written as: a += 2;

The operator +=tells the compiler that a is assigned the value of a + 2;

This shorthand works for all binary operators in C. The general form is:

variable operator = variable / constant / expression;

These operators are listed below:

Operators	Examples	Meaning
+=	a+=2	a=a+2
-=	a-=2	a=a-2
=	**a***=2	a = a*2
/=	a/=2	a=a/2
%=	a%=2	a=a%2
Operators	**Examples**	**Meaning**
&&=	a&&=c	a=a&&c
||=	a||=c	a=a||c

2.20 PRIORITY OF OPERATORS

Since all the operators we have studied in this unit can be used together in an expression, C uses a certain hierarchy to solve such kind of mixed expressions. The hierarchy and associatively of the operators discussed so far is summarized in Table 2.7. The operators written in the same line have the same priority. The higher precedence operators are written first.

Table 2.7: Precedence of the operators

Operators	Associativity
( )	Left to right
! ++ -- (*type*) sizeof	Right to left
/ %	Left to right
+ -	Left to right
< <= > >=	Left to right
== !=	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &&= ||=	Right to left
,	Left to right

Check Your Progress 5

1. Give the output of the following C code:

```
main( )
{
    int a,b=5;
    float f;
   a=5/2;
   f=(float)b/2.0;
   (a<f)? b=1:b=0;
   printf("b = %d\n",b);
}
```

……

……

……

……

……

2. What is the difference between && and &. Explain with an example.

……

……

……

……

3. Use of Bit Wise operators makes the execution of the program.

……

……

……

……

……

2.21 SUMMARY

To summarize 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. These are used to form words in C language or names or identifiers. Variable are the identifiers, which change their values during execution of the program. Keywords are names with specific meaning and cannot be used otherwise.

We had discussed four basic data types - int, char, float and double. Some qualifiers are used as prefixes to data types like signed, unsigned, short, and long.

The constants are the fixed values and may be either Integer or Floating point or Character or String type. Symbolic Constants are used to define names used for constant values. They help in using the name rather bothering with remembering and writing the values.

In this unit, we discussed about the different types of operators, namely arithmetic, relational, logical present in C and their use. In the following units, you will study how these are used in C's other constructs like control statements, arrays etc. This unit also focused on type conversions. Type conversions are very important to understand because sometimes a programmer gets unexpected results (logical error) which are most often caused by type conversions in case user has used improper types or if he has not type cast to desired type.

C is referred to as a compact language which is because lengthy expressions can be written in short form. Conditional operator is one of the examples, which is the short form of writing the if/else construct (next unit). Also increment/decrement operators reduce a bit of coding when used in expressions.

Since logical operators are used further in all types of looping constructs and if/else construct (in the next unit), they should be thoroughly understood.

2.22 SOLUTIONS / ANSWERS

Check Your Progress 1

1. **Keywords:** int, union

   Valid Identifiers: hello, student_1, max_value

2. int rollno;

   float total_marks, percentage;

3. a) 1 byte b) 2 bytes c) 8 bytes

Check Your Progress 2

1. # define PI 3.14

2. **Integer constant**: 0147

   Character constants: 'A', '\r'

   String constants: "A", "EFH"

Check Your Progress 3

1. C expression would be

   i) ((a*4*c*c)-d)/(m+n)

   ii) a*b-(e+f)*4/c

2. The output would be:

   a=1 b=4 c=4 k=10

3. There is no such operator as **.

Check Your Progress 4

1. The expression is evaluated as under:

   (3 < - -4) && (3== 2)

   (3 < 3) && (3==2)

   0 && 0

   0

   Logical false evaluates to 0 and logical true evaluates to 1.

2. The output would be as follows:

   x=11, a=11, b=16

3. Comma operator causes a sequence of operators to be performed.

Check Your Progress 5

1. Here a will evaluate to 2 and f will evaluate to 2.5 since type cast operator is used in the latter so data type of b changes to float in an expression. Therefore, output would be b=1.
2. && operator is a logical and operator and & is a bit wise and operator. Therefore, && operator always evaluates to true or false i.e 1 or 0 respectively while & operator evaluates bit wise so the result can be any value. For example:

   2 && 5 => 1(true)

   2 & 5 => 0(bit-wise anding)

3. Use of Bit Wise operators makes the execution of the program faster.

2.23 FURTHER READINGS

1. The C Programming Language, *Kernighan & Ritchie*, PHI Publication.
2. Computer Science A structured programming approach using C, *Behrouza A. Forouzan, Richard F. Gilberg*, Second Edition, Brooks/Cole, Thomson Learning, 2001.
3. Programming with C, *Gottfried*, Second Edition, Schaum Outlines,Tata Mc Graw Hill, 2003.