Programming in C with Numerical Methods

Unit I

Chapter II

Constants, variables and data types

A programming language is designed to help process certain kinds of data consisting of numbers, characters and strings and to provide useful output known as information. The task of processing of data is accomplished by executing a sequence of precise instructions called a program.

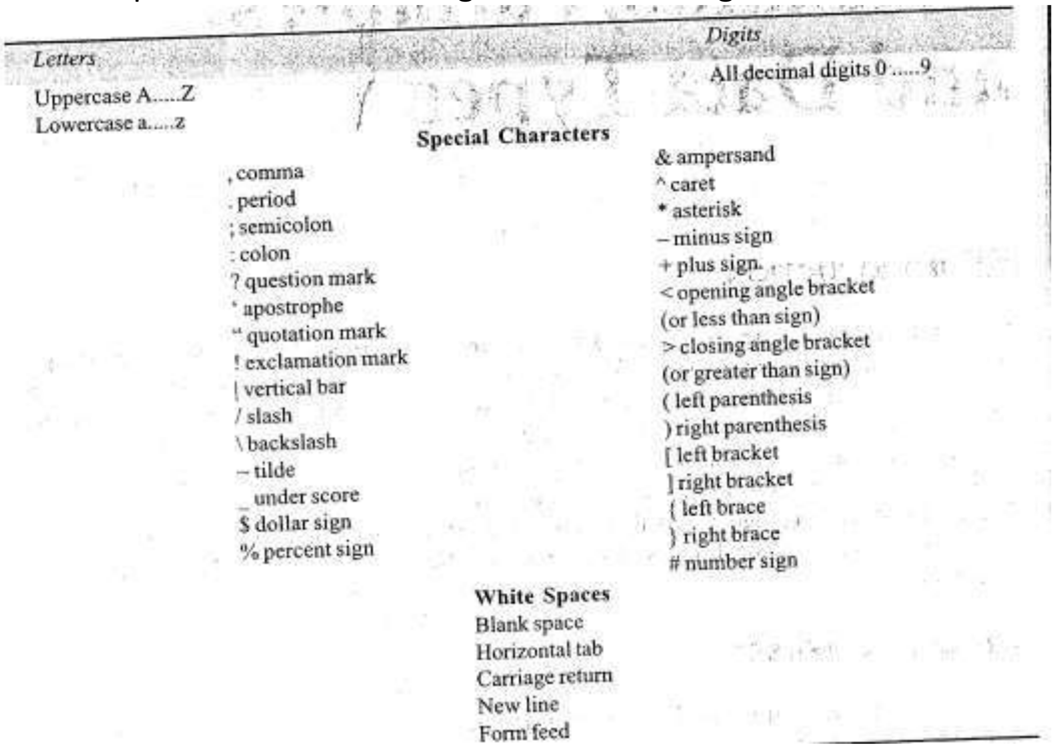
These instructions are formed using certain symbols and words according to some rigid rules known as syntax rules (or grammar).

Character set

The characters that can be used to form words, numbers and expressions depend upon the Computer on which the program is run. However, a subset of characters is available that can be used on most personal, micro, mini and mainframe computers. The characters in C are grouped into the following categories.

1. Letters
2. Digits
3. Special characters
4. White spaces

The complete character set is given in table2.1 given below



Letters	*Digits*
Uppercase A.....Z	All decimal digits 0.....9
Lowercase a.....z	

Special Characters	
, comma	& ampersand
. period	^ caret
; semicolon	* asterisk
: colon	– minus sign
? question mark	+ plus sign
' apostrophe	< opening angle bracket (or less than sign)
" quotation mark	> closing angle bracket (or greater than sign)
! exclamation mark	(left parenthesis
| vertical bar	) right parenthesis
/ slash	[left bracket
\ backslash	] right bracket
~ tilde	{ left brace
_ under score	} right brace
$ dollar sign	# number sign
% percent sign	

White Spaces
Blank space
Horizontal tab
Carriage return
New line
Form feed

Trigraph Characters
ANSI C introduced the concept of Trigraph sequences to provide a way to enter certain characters that are not available on some key boards.Each Trigraph sequence consist of three characters (two questions marks followed by another character)

For example, if a key board does not support square bracket ,we can still use them in a program using the Trigraph??(and??)

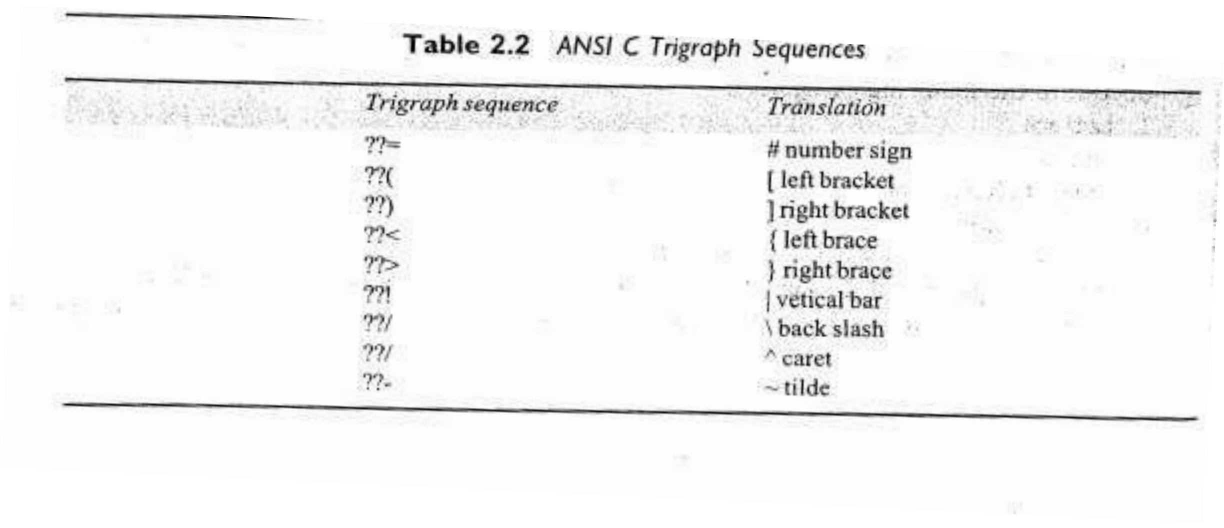


Table 2.2 *ANSI C Trigraph Sequences*

Trigraph sequence	*Translation*
??=	# number sign
??(	[left bracket
??)	] right bracket
??<	{ left brace
??>	} right brace
??!	| vetical bar
??/	\ back slash
??/	^ caret
??-	~ tilde

C TOKENS

In a passage of text, individual words and punctuation marks are called tokens. Similarly, in a C program the smallest individual units are known as C tokens. C has six types of tokens as shown in Fig. 2.1

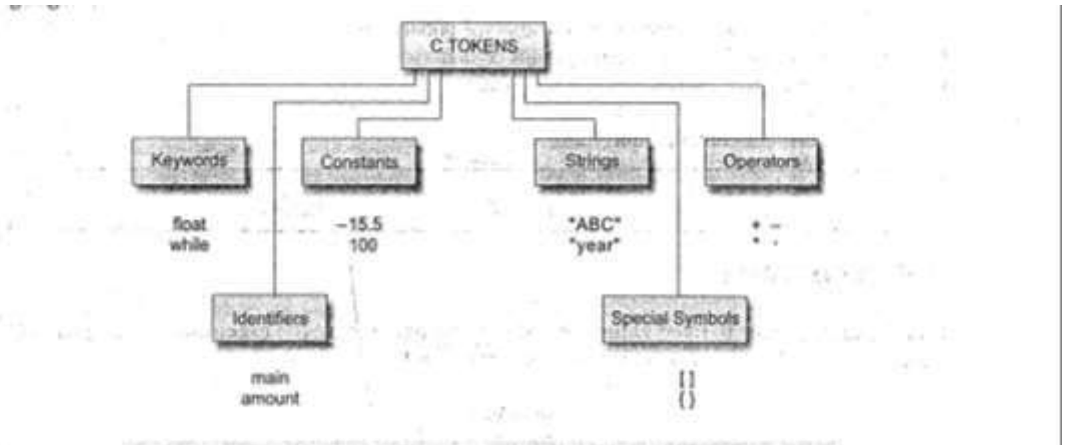
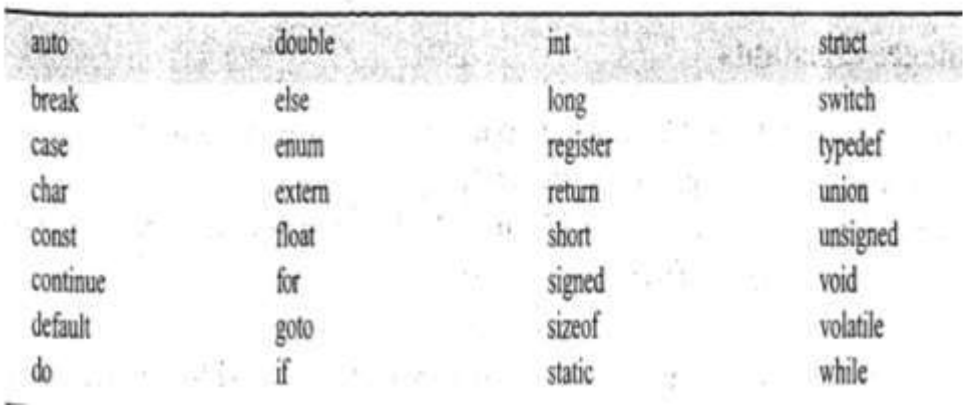


Fig. 2.1 *C tokens and examples*

KEYWORDS AND IDENTIFIERS

Every C word is classified "as either a keyword. or an identifier. All keywords have fixed meanings and these meanings cannot be changed. Key Words serve as basic building blocks for program statements. The list of all keywords of ANSI C is listed in Table 2.3. All key words must be written in lowercase. Some compilers may use additional keywords that must be identified from the C manual.

Table 2.3 *ANSI C Keywords*



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

Identifiers refer to the names of variables, functions and arrays. These are user-defined names and consist of a sequence of letters and digits, with a letter as a first character. Both upper case and lower case are permitted, although lower case letters are commonly used. The underscore character is also permitted in identifiers. It is used as a link between two words in long identifiers

CONSTANTS

Constants in C refer to fixed values that do not change during the execution of the program. C supports several types of constants as illustrated in Fig 2.2.

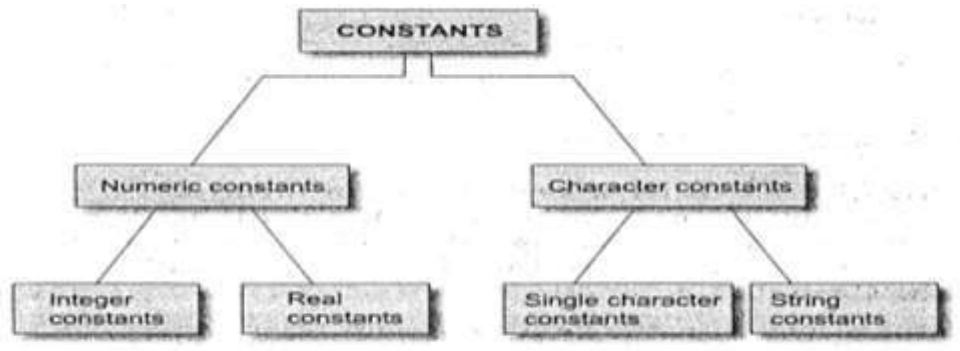


Fig. 2.2 *Basic types of C constants*

INTEGER CONSTANTS.

An Integer constants refers to a sequence of digits. There are three types of Integers, namely,decimal,octal,hexadecimal.

Decimal Integers consist of a set of digits,0 through 9,preceded by an optional - or + sign.Valid examples of decimal integers constants are :

123 -321 0 654321 +78

Embeded spaces,commas,and non digit characters are not permitted between digits.For example

15 750 20,000 $1000

are illegal numbers.

An octal integer constant consists of any combination of digits from the set 0 through 7,with a leading 0. Some examples of octal integer are:

037 0 0435 0551

A sequence of digits preceded by Ox or OX is considered as hexadecimal integer. They may also include alphabets A through F or a through f. The letter A through F represent the numbers 10 through 15. Following are the examples of valid hex integers:

OX2 Ox9F 0Xbcd Ox

We rarely use octal and hexadecimal numbers in programming.

The largest integer value that can be stored is machine-dependent. It is 32767 on 16-bit machines and 2,147,483,647 on 32-bit machines; It is also possible to store larger integer constants on these machines by appending qualifiers such as U,L and UL to the constants.

For examples:

56789U	or	56789u (unsigned integer)
987612347UL	or	98761234ul (unsigned long integer)
9876543L	or	98765431 (long integer)

Real Constants

Integer numbers are inadequate to represent quantities that vary continuously, such as distances, heights, temperatures, prices, and so on. These quantities are represented by numbers containing fractional parts like 17.548. Such numbers are called real (or floating Point) constants. Further examples of real constants are:

0.0083 -0.75 435.36 +247.0

These numbers are shown in decimal notation, having a whole number' followed by a decimal point and the fractional part. It is possible to omit digits before the decimal point, or digits after the decimal point that is

215. .95 -.71 +.5

are all valid real numbers.

A real number may also be expressed in exponential (or scientific)notation.For example,the value 215.65 may be written as 2.1565e2 in exponential notation.

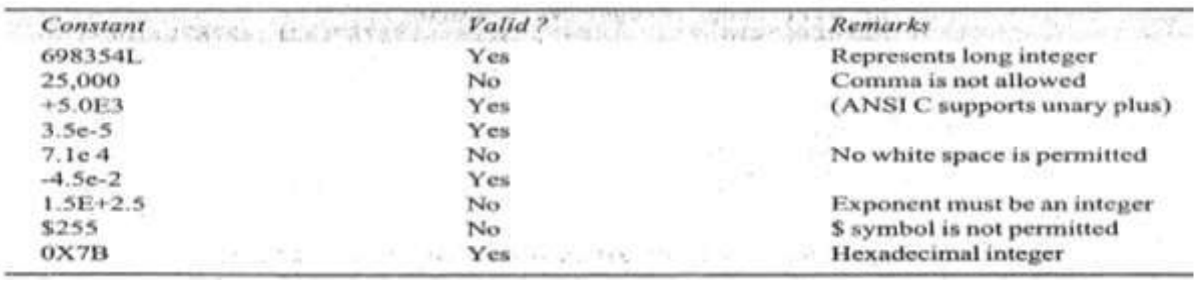
mantissa ***e*** *exponent*

The mantissa is either a real number expressed in decimal notation..or an integer.This exponent is an integer number with an optional plus or minus sign, The letter **e** separating the mantissa and the exponent can be written in either lowercase or uppercase. Since the exponent causes the decimal point to “float”,this notation is said to represent a real number in floating point form,Examples of legal floating point constants are

0.65e4 12e-2 1.5e+5 3.18E3 -1.2E-1

Embedded white space is not allowed.
Exponential notation is useful for representing numbers that are either very large or very small in magnitude. For example 7500000000 may be written as 7.5E9 or 75E8.Similarly -0.000000368 is equivalent to -3.68E -7

Table 2.4 *Examples of Numeric Constants*



Constant	*Valid ?*	*Remarks*
698354L	Yes	Represents long integer
25,000	No	Comma is not allowed
+5.0E3	Yes	(ANSI C supports unary plus)
3.5e-5	Yes	
7.1e 4	No	No white space is permitted
-4.5e-2	Yes	
1.5E+2.5	No	Exponent must be an integer
$255	No	$ symbol is not permitted
0X7B	Yes	Hexadecimal integer

SINGLE CHARACTER CONSTANTS

A single character contants (or simply character constants) contains a single character enclosed within a pair of single quote marks.Examples of character constants are

‘5’ ‘X’ ‘;’ ‘’

Note that the character constant ‘5’ is not same as the number 5.The last constant is a blank space.

Character constants have integer values known as ASCII values.
For example the statement

printf(“%d”, ‘a’);

would print the number 97,the ASCII value of the letter a.

Similarly the statement

printf(“%c”, ‘97’);

would output the letter **‘a’**

STRING CONSTANTS

A string constant is a sequence of characters enclosed in double quotes.The characters may be letters, numbers, special characters and blank space. .Examples are

"Hello!" "1987" "WELL DONE" "?...!" "5+3" "X"

Remember that a character constant (e.g., 'X') is not equivalent to the single character string constant (e.g., "X"). Further, a single character string constant does not have an equivalent integer value while a character constant has an integer value. Character strings are often used in programs to build meaningful programs. Manipulation of character strings are considered in detail in chapter 8.

BACKLASH CHARACTER CONSTANTS

C supports some special backslash character constants that are used in output functions. For example, the symbol '\n' stands for newline character. A list of such backslash character constants is given in Table 2.5. Note that each one of them represents one character although they consist of two characters. These characters combinations are known as escape sequences.

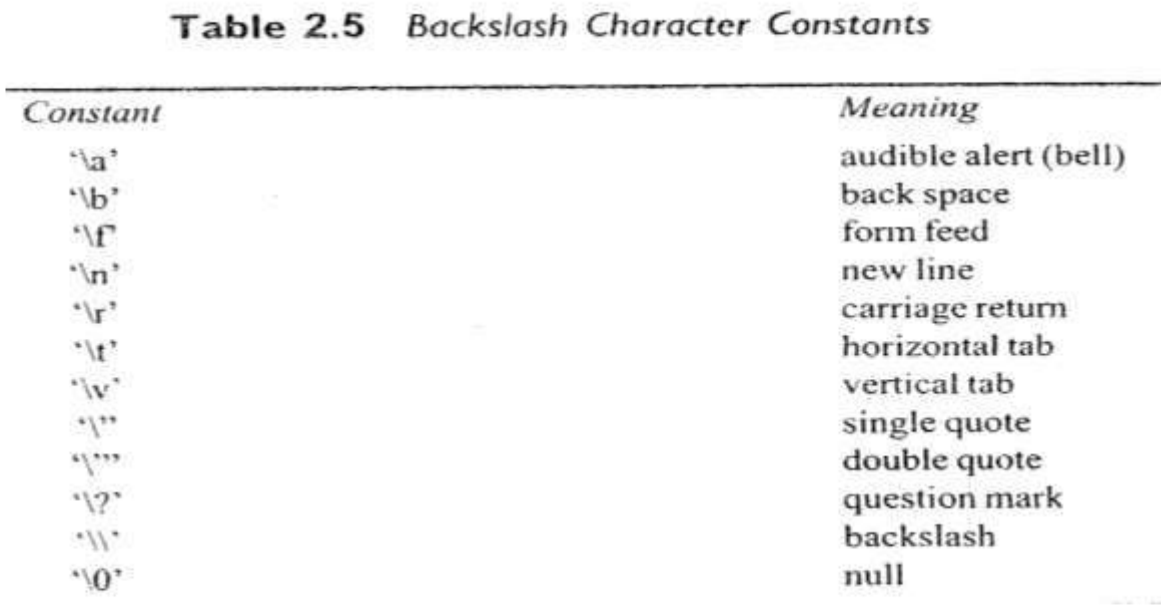


Table 2.5 *Backslash Character Constants*

Constant	*Meaning*
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\t'	horizontal tab
'\v'	vertical tab
'\''	single quote
'\"'	double quote
'\?'	question mark
'\\'	backslash
'\0'	null

VARIABLES

A variable is a data name that may be used to store a data value. Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during execution. In chapter 1,we used several variables. For instance, we used the variables **amount** in sample program 3 to store the value of money at the end of each year after adding the interest earned during that year.

A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. Some examples of such names are

Average   height   Total   Counter_1   class_strength

As mentioned earlier, variable names may consist of letters, digits and the underscore(_)character, subject to the following conditions.

1.They must begin with a letter. Some system permit underscore as the first character.

2.ANSI standard recognizes a length of 31 character.

However, the length should not be normally more than eight character, since only the first eight characters, since only the first eight characters are treated as significant by many compilers.

3.Uppercase and lowercase are significant.That is,the variable **Total** is not the same as **total** or **TOTAL.**

4.The variable name should not be key word.

5.While space is not allowed.

Some examples of valid variable names are

John	Value	T_rise
Delhi	x1	ph_value
Mark	sum1	distance

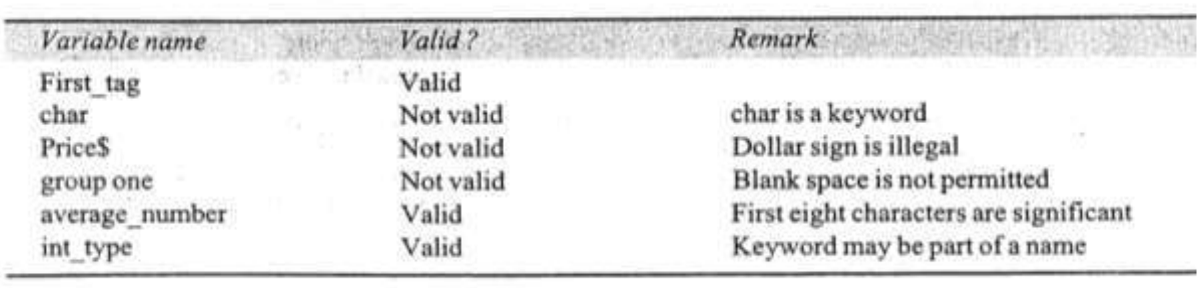
Invalid examples include

123 (area)

% 25th

Further examples of variable names and their correctness are given in Table 2.6

Table 2.6 *Examples of Variable Names*



Variable name	*Valid ?*	*Remark*
First_tag	Valid	
char	Not valid	char is a keyword
Price$	Not valid	Dollar sign is illegal
group one	Not valid	Blank space is not permitted
average_number	Valid	First eight characters are significant
int_type	Valid	Keyword may be part of a name

If only the first eight characters are recognized by a compiler, than the two names average_height

average_height

mean the same thing to the computer. Such names can be rewritten as

avg_height and avg_weight

or

ht_average and wt_average.

Without changing their meanings.

DATA TYPES

Programming in ANSI C

C langua
handle c
the progr
machine.

ANSI
1. Pri
2. De
3. Us

The pr
fined dat
functions

All C
(**char**), fl
them als
and the t
types are

NOTE:

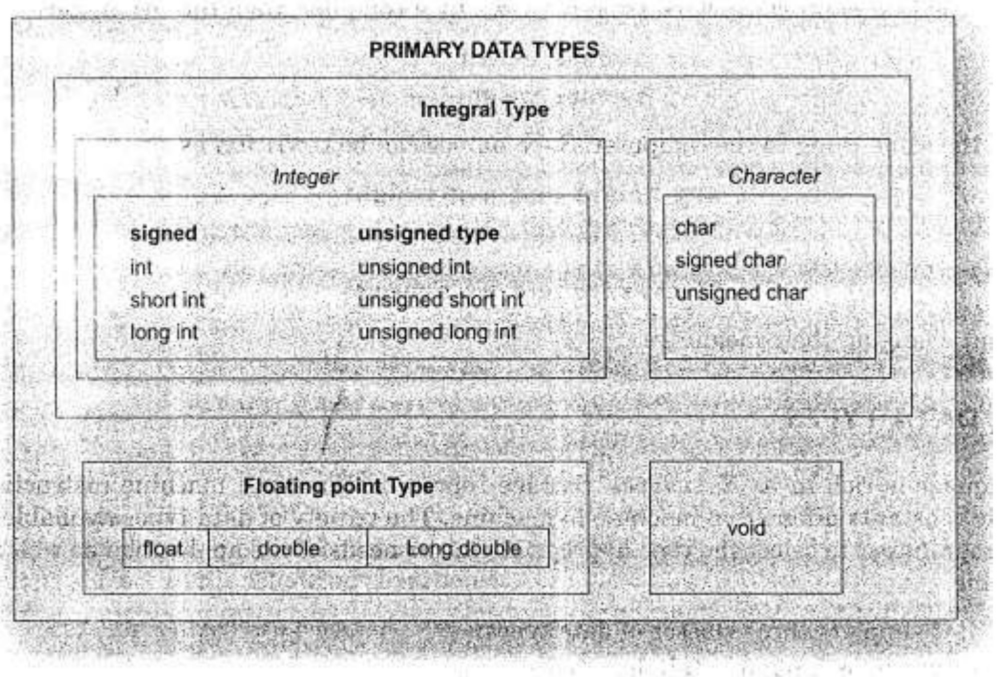


Table 2.7 *Size and Range of Basic Data Types on 16-bit Machines*

Data type	*Range of values*
char	–128 to 127
int	–32,768 to 32,767
float	3.4e–38 to 3.4e+e38
double	1.7e–308 to 1.7e+308

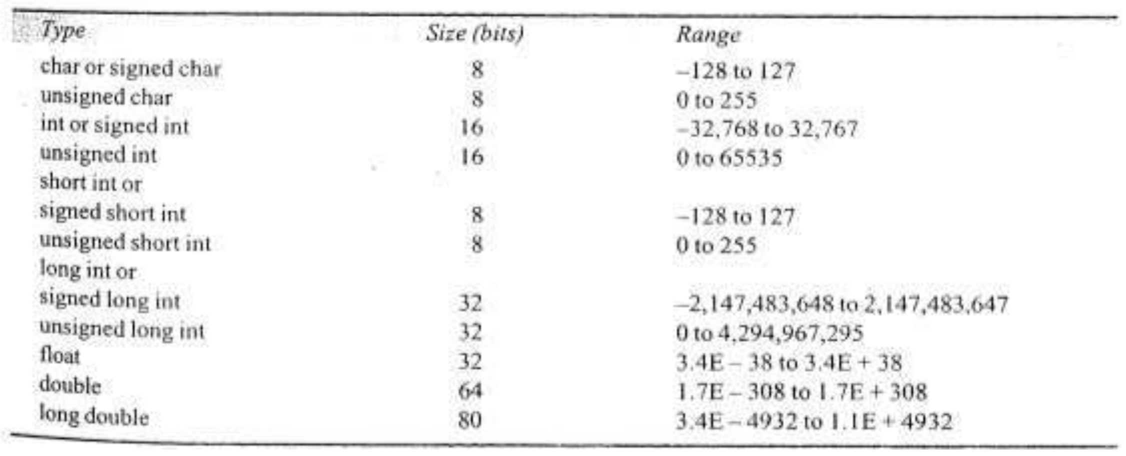
INTEGER TYPES

Integers are whole numbers with a range of values supported by a particular machine. Generally integers occupy one word of storage, and since the word sizes of machines vary(typically,16 or32bits)the size of an integer that can be stored depends on the computer. If we use a 16 bit word length the size of the integer value is limited to the range -32768 to +32767 that is -2^{15} to $+2^{15}$-1.A signed integer uses one bit for sign and 15 bits for the magnitude of the number. Similarly, a 32 bit word length can store an integer ranging from -2,147,483,648 to 2,147,483,647.

In order to provide some control over the range of numbers and storage space, C has three classes of integer storage, namely **short int, int, and long int,** in both signed and unsigned forms. For example **short int** represents fairly small integer values and requires half the amount of storage as a regular **int** number uses.Unlike signed integer,unsigned integers use all bits for the magnitude of the number are always positive.Therefore,for a 16 bit machine, the range of unsigned integer number will be from 0 to 65,535.

We declare **long** and **unsigned** integer to increase the range of values. The use of qualifier **signed** on integer is optional because the default declaration assumes a signed number .Table 2.8 shows all the allowed combination of basic types and qualifiers and their size and range on a 16 bit machine.

Table 2.8 *Size and Range of Data Types on a 16-bit Machine*



Type	*Size (bits)*	*Range*
char or signed char	8	–128 to 127
unsigned char	8	0 to 255
int or signed int	16	–32,768 to 32,767
unsigned int	16	0 to 65535
short int or signed short int	8	–128 to 127
unsigned short int	8	0 to 255
long int or signed long int	32	–2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
float	32	3.4E – 38 to 3.4E + 38
double	64	1.7E – 308 to 1.7E + 308
long double	80	3.4E – 4932 to 1.1E + 4932

FLOATING POINT TYPES

Floating point (or real) numbers are stored in bits (on all 16 bit and 32 bit machines),with 6 digits of precision.Floating point numbers are defined in C by the keyword **float.**When the accuracy provided by the float number is not sufficient ,the type **double** can be used to define the number.

CHARACTER TYPE

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

DECLARATION OF VARIABLES

Declaration does two things

1.It tells the computer what the variable name is

2.It specifies what type of data the variable will hold.

The declaration of variables must done before they are used in the program.

PRIMARY TYPE DECLARATION

A variable can be used to store value of any data type. The syntax for declaring a variable is as follows.

data-type v1,v2……vn ;

v1,v2….vn are the names of the variables. Variables are separated by commas. A declaration statement must end with a semicolon. For example valid decalrations are **int** count ;

int number ,total ;

double ratio ;

int and **double** are the key words to represent integer type and real type .

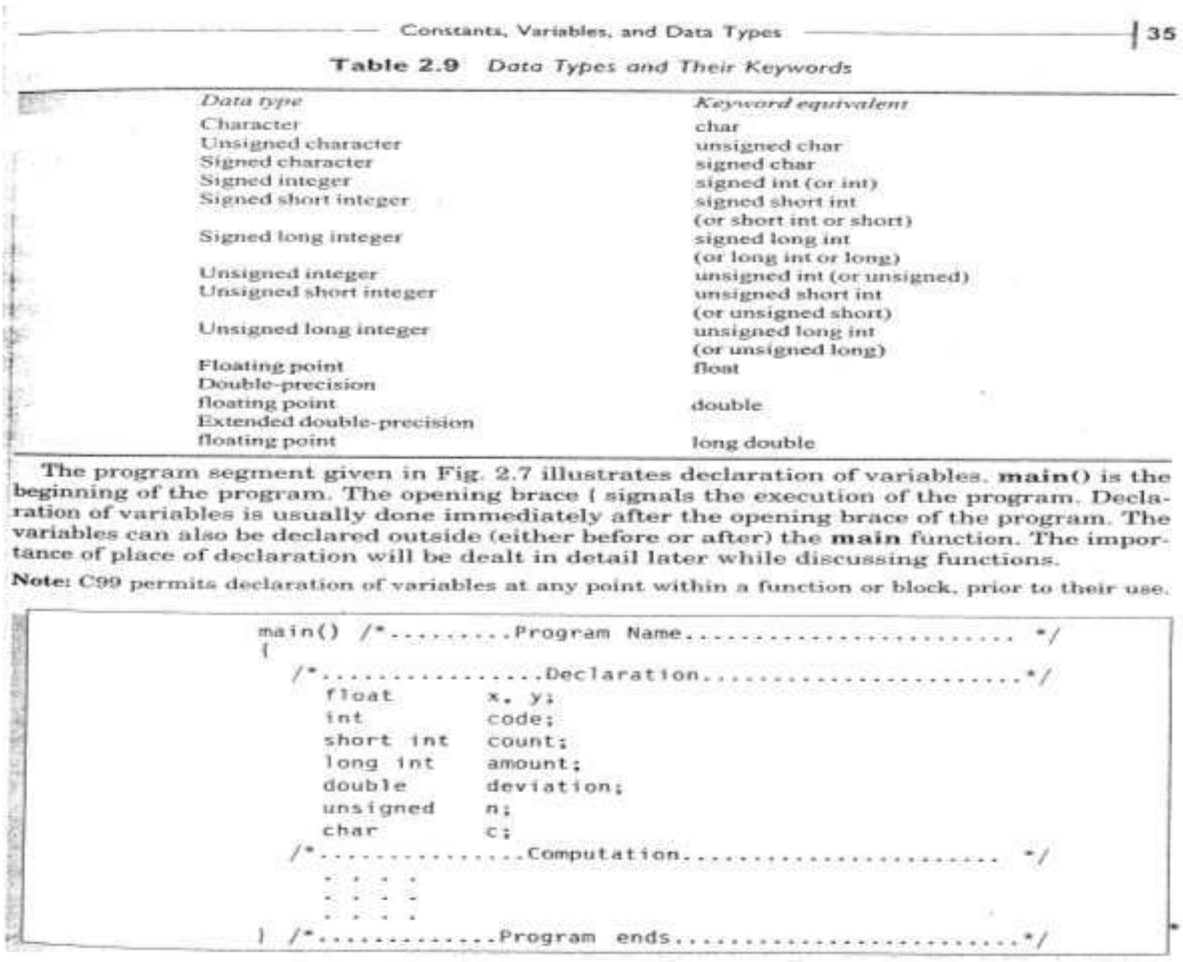


Table 2.9 *Data Types and Their Keywords*

Data type	*Keyword equivalent*
Character	char
Unsigned character	unsigned char
Signed character	signed char
Signed integer	signed int (or int)
Signed short integer	signed short int (or short int or short)
Signed long integer	signed long int (or long int or long)
Unsigned integer	unsigned int (or unsigned)
Unsigned short integer	unsigned short int (or unsigned short)
Unsigned long integer	unsigned long int (or unsigned long)
Floating point	float
Double-precision floating point	double
Extended double-precision floating point	long double

The program segment given in Fig. 2.7 illustrates declaration of variables. **main()** is the beginning of the program. The opening brace { signals the execution of the program. Declaration of variables is usually done immediately after the opening brace of the program. The variables can also be declared outside (either before or after) the **main** function. The importance of place of declaration will be dealt in detail later while discussing functions.

Note: C99 permits declaration of variables at any point within a function or block, prior to their use.

```
main() /*.........Program Name........................ */
{
  /*................Declaration.......................*/
    float       x, y;
    int         code;
    short int   count;
    long int    amount;
    double      deviation;
    unsigned    n;
    char        c;
  /*...............Computation......................  */
    . . . .
    . . . .
    . . . .
} /*.............Program ends.........................*/
```

Fig. 2.7 *Declaration of variables*

USER-DEFINED TYPE DECLARATION.

C supports a feature known as “type definition”that allows user to define an identifier that would represent an existing data type

typedef *type* identifier

The existing data type may belong to any class of type, including the user defined ones

.New type is ‘new’ only in name but not the data type.

typedef cannot create a new type. Some examples are

typedef int units;
typedef float marks;

Here,**units** symbolizes **int** and **marks** symbolizes the **float**. The can be later used to declare variables as follows. They can be later used to declare variables as follows **units** batch1,batch2 ;
marks name 1[50],name2[50]

batch1 and batch 2 are declared as **int** variable and name1[50] and name 2[50] are declared as 50 element floating point array variables. The main advantage of **typedef** is that we can create meaningful data type names for increasing the readability of the program.

Another user –defined data type is enumerated data type provided by ANSI standard. It is defined as follows,

enum identifier (value1,value2….valuen);

The “identifier” is a user -defined enumerated data type which can be used to declare variables that can have one of the values enclosed within the braces(known as enumerated constants)

enum identifier v1,v2,…..vn

The enumerated variables v1,v2…vn can only have one of the values value1,value2….valuen. The assignments of the following types are valid.

Example **enum day** (Monday,Tuesday,……Sunday)

enum day week_st,week_end

week_st = Mon week_end = Fri if week_st = = Tue week_end = = Sat

The compiler automatically assigns integer digits beginning with 0 to all the enumerated constants.That is the enumeration constant value1 is assigned 0,value 2 is assigned 1 and so on.

Here,the constant Monday is assigned the value of 1.The remaining constants are assigned values that increase successively by 1.

The definition and declaration of enumerated variables can be combined in one statement.

enum day (Monday,….Sunday)week_st,week_end

DECLARATION OF STORAGE CLASS

Variables in C can have not only data types but also storage class that provides information about their location and visibility.The storage class decides the portion of the program within which the variables are recognized.

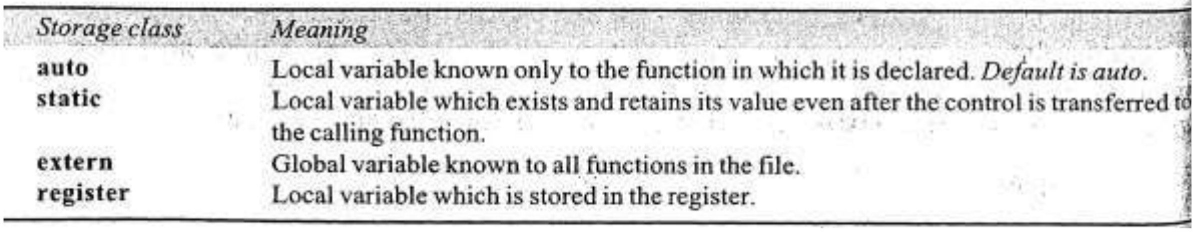
/*Examples of storage classes*/

```
int     m ;
main ( )
{
        int     i ;
        float balance
        …..
        …...
        function 1 ( )
}
Function 1 ( )
{
        int i ;
        float sum
        …..
        …..
}
```

The variable **m** which has been declared before the **main** is called global variable. A global variable is also known as an external variable
It can be used in all the functions in the program. It need not be declared in other functions.
The variables **i, balance** and **sum** are called local variables, because they are declared inside a function. Local variables are visible and meaningful only inside the function in which they are declared. Note that the variable **i** been declared in both the functions. Any change in the value of **i** in one function does not affect its value in the other.

C provides a variety of storage class specifiers that can be used to declare explicitly the scope and lifetime of variables.The concept of scope and lifetime are important only in multifunction and multiple file programs.
Four stages of Class specifiers are (auto,register,static,and extern)

Table 2.10 *Storage Classes and Their Meaning*



Storage class	*Meaning*
auto	Local variable known only to the function in which it is declared. *Default is auto.*
static	Local variable which exists and retains its value even after the control is transferred to the calling function.
extern	Global variable known to all functions in the file.
register	Local variable which is stored in the register.

ASSIGNING VALUES TO VARIABLES

Variables are created for use in program statements such as

```
value  =       amount +   inrate *   amount;
while (year <=  PERIOD)
{
          ……
          ……
year   = year + 1;
}
```

In the first statement, the numeric value stored in the variable **inrate** is multiplied by the value stored in **amount** and the product is added to **amount.** The result is stored in the variable **value**. This process is possible only if the variables **amount** and **inrate** have already been given values. The variable **value** is called the target variable.

ASSIGNMENT STATEMENT

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

```
variable_name = constant;
```

We have already used such statements in Chapter 1.Further examples are:

```
initial_value      =           0;
final_value        =         100;
balance            =       75.84;
yes                =          x;
```

An assignment statement implies that the value of the variable on the left of the 'equal sign' set equal to the value of the quantity (or the expression) on the right.The statement

```
year =   year +  1;
```

means that the 'new value' of year is equal to the 'old value' of **year** plus 1.

It is also possible to assign a value to a variable at the time the variable is declared. This take the following form:

data-type variable_name = constant;

Some examples are

int final_value = 100;
char yes = 'x';
double balance = 75.84;

The process of giving initial values to variables is called initialization. C permits the initialization of more than one variables in one statement using multiple Assignment operators.
For example the statements

P = 9 = S = 0;
X = y = Z = MAX;

are valid. The first stajement initializes the variables p, q, and s to zero while the second inittaeizes
x. y. andz with MAX.

DECLARING A VARIABLE AS CONSTANT
We may like the value of certain variables to remain constant during the execution of a program.We can achieve this by declaring the variable with the qualifier **const** at the time of initialization.
Example:

const int class_size = 40;

const is a new data type qualifier defined by ANSl standard. This tells the compiler that the value of the **int** variable **class_size** must not be modified by the program. However, it can be used on the right_hand side of an assignment statement like any other variable.

DECLARING A VARIABLE AS VOLATILE
ANSI standard defines another qualifier volatile that could be used to tell explicitly the compiler that a variable's value may be changed at any time by some external sources (from outside the program).
For example:

volatile int date;

The value of date may be altered by some external factors even if it does not appear on the left hand side of an assignment statement. When we declare a

variable as **volatile**, the compiler will exmaine the value of the variable each time it is encountered to see whether any external alteration has changed the value.
Remember that the value of a variable declared as **volatile** can be modified by the program as well. If we wish that the value must not be modified by the program while it may be altered by some other process, then we may declare the variable as both **const** and **volatile** as shown below:

```
volatile const int location = 100;
```

Overflow and Underflow of Data

Problem of data overflow occurs when the value of the variable is either too big or too small for the data type to hold.The largest value that a variable can hold depends on the machine.Since the floating point values are rounded off to the number of significant digits allowed (or specified) an overflow normally results in the largest possible real value ,whereas an underflow results in zero.

Reading Data from Keyboard

Giving values to variables is to input data through key board using the **scanf** function, It is a general input function available in C and is very similar in concept to the **printf** function.It works much like an INPUT statement in BASIC.The general format of scanf is as follows

```
scanf (“control string”, &variable2,…..);
```

The control string contains the format of data being received.The ampersand symbol **&** before each variable name is an operator that specifies the variable name address.We must always use this operator ,otherwise unexpected results may occur.

Example **scanf (“%d”, &number)**

When this statement is encountered by the computer,the execution stops and wait for the value of variable **number** to be typed in.

DEFINING SYMBOLIC CONSTANTS

We often use certain unique constants in a program .These constants may appear repeatedly in a number of places in the program. One example of such a constant is 3.142, representing the value of the mathematical constant "pi". Another example is the total number of students whose mark sheets are analyzed by a "test analysis program'. The number of students, say 50, may be used for calculating the class total, class average, standard deviation, etc.

We face two problems in the subsequent use of such programs.

1. Problem in modification of the program.
2 Problem in understanding the program.

Modifiability
We may like to change the value of "pi" from 3.142 to 3.14159 to improve the accuracy of the calculation or the number 50 to 100 to process the test results of another class. In both the cases we will have to search throughout the program and explicitly change the value of the constant wherever it has been used. If any value is left unchanged, the program may produce disastrous outputs.

Understandability
When a numeric value appears in a program, its use is not always clear, especially when the same value means different things in different places.
For example, the number 50 may mean the number of students at one place and the ‘pass marks’ at another place of the same program. We may forget what a certain number meant ,when we read the program some days later.

Assignment of such constants to symbolic name frees us from this problem. For example we may use the name **STRENGHT** to define the number of students and **PASS_MARK** to define the pass mark required in a subject. Constant values are assigned to these names at the beginning of the program. Subsequent use of the names **STRENGTH** and **PASS_MARK** in the program has the effect of causing their defined values to be automatically substituted at the appropriate points.

A constant is defined as follows:

#define symbolic-name value of constant

Valid examples of constant definitions are:

```
#define STRENGTH          100
#define PASS_MARK          50
#deflne MAX               200
#define PI                  3.14159
```

Symbolic names are sometimes called constant identifiers. Since the symbolic names are constants (not variables), they do not appear in declarations. The following rules apply to a **#define** statement which define a symbolic constant.

1. Symbolic names have the same form as variable names. (Symbolic names are written in CAPITALS to visually distinguish them from the normal variable names which are written in lowercase letters. This is only a convention, not a rule.)
2. No blank space between the pound sign ‘#’ and the word **define** is permitted.
3. ‘#’ must be the first character in the line.

4. A blank space is required between **#define** and symbolic name and between the symbolic name and the constant.

5. **#define** statements must not end with a semicolon.

6. After definition, the symbolic name should not be assigned any other value within the program by using an assignment statement. For example, STRENGTH = 200: is illegal.

7. Symbolic names are **NOT** declared for data types. Its data type depends on the type of constant

8. **#define** statements may appear anywhere in the program but before it is referenced in the program (the usual practice is to place them in the beginning of the program).

Operators and Expressions
Chapter: 3

An operator is a symbol that tells the computer to Perform certain mathematical or logical manipulations. Operators are used in program to manipulate data and variables. They usually form a part of the mathematical of logical expressions

C operators can be classified into a number of categories. They include

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators.

Arithmetic operators

C provides all the basic arithmetic operators. The operators +, -, *, and / all work the same way as they do in other language.

Arithmetic operators

Operator	Meaning
+	Addition or unary plus
−	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division.

Integer division truncates any fractional part. The modulo division produces the remainder of an integer division.

Eg: a-b, a+b, a*b, a/b, a%b, -a*b. Here a & b are variables and are known as operands. The modulo division operator % cannot be used on floating point data.

Integer Arithmetic

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

In the above eg, if a and b are integers, then for a=14, b=4 we have the following results

a-b=10, a+b=18, a*b=56, a/b = 3 (decimal part truncated)

a % b = 2 (remainder of division)

During intenger division, if both the opeaands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependent. (ie) 6/7 = 0 and -6/-7 = 0 but -6/7 may be 0 or 1

|||'y during modulo division, the sign of the result is always the sign of the first operand (the dividend)

(ie) -14 % 3 = -2, -14% -3 = -2, 14% -3 = 2.

Real Arithmetic

An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an appl- of correct value.

If x, y and z are floats, then we have

$x = 6.0/7.0 = 0.857143$

$y = 1.0/3.0 = 0.3333$

$z = -2.0/3.0 = -0.666667.$

The operator % cannot be used with real operands.

Mixed-mode Arithmetic

When one of the operand is real and the other is integer, the expression is called a mixed-mode arithmetic expression. If either operand is of the real type, then only the real operation is performed and the result is always a real no. Thus 15/10.0 = 1.5 where as 15/10 = 1.

Relational operators

We often compare two quantities, and depending on their relation, take certain decision. For eg. we may compare the age of two persons, or the price of two items and so on. These comparison can be done with the help of relational operator. An expression such as a<b or 1<20 containing a relational operator is termed as a relational expression. The value of relation expression is either zero or one. It is one if the specified relation is true and zero if the relation is false. For eg: 10<20 is true but 20<10 is false.

Relational operators

Operator	Meaning
<	is less than
<=	is less than or equal to
>=	is greater than or equal to
>	greater than
==	is equal to
!=	is not equal to.

A simple relational expression contains only one relational operator and takes the following form

$\boxed{\text{ae-1 relational operator ae-2}}$

ae-1 and ae-2 are arithmetic expressions, which may be simple constants, variables or combination of them.

Some examples:

4.5 <= 10	True
4.5 < -10	False
-35 >= 0	False
10 < 7+5	True
a+b == c+d	True

only if the sum of values of a and b is equal to the sum of values of c and d.

When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and the result compared.

<u>Logical operators</u>

C has the following three logical operators

&& meaning logical AND

|| meaning logical OR

! meaning logical NOT

* The logical operator && and || are used when we want to test more than one condition and make decision.

* Eg: a>b && x == 10. These expressio is called logical expression or a compound relational expression.

* These expression yields a value of <u>one</u> or <u>zero</u> according to the truth table.

* The above eg is true if a>b is true and x==10 is true.
* If either of them are false, the expression is false

Truth Table

OP-1	OP-2	Value of the expression. op-1 && op-2	op-1 || op-2
Nonzero	Nonzero	1	1
Nonzero	0	0	1
0	Nonzero	0	1
0	0	0	0

* Some egs: If (age > 55 && salary < 1000)

Assignment operators

* It is used to assign the result of an expression to a variable (usual assignment operator "=")
* C has a set of shorthand assignment operators of the form $\boxed{V\ op = exp;}$, V is a variable, exp is an expression and op is a binary arithmetic operators.
* The operator op= is known as shorthand assignment
* Vop = exp; is equivalent to V = V op(exp);
* eg: x + = y+1;

  This is the same as the statement

  x = x+(y+1);

Shorthand Assignment Operators

Statement with simple assignment operator	Statement with shorthand operator
a = a+1	a+=1
a = a-1	a-= 1
a = a*(n+1)	a*= n+1
a = a/(n+1)	a/ = n+1
a = a %b	a%=b

* Three advantage of shorthand Assignment operators.
1. What appears on the left hand side need not be repeated
2. The stt/- is more concise to read
3. The stt/- is more efficient.

Increment and Decrement operators

* The increment and decrement operators : ++ & --
* The operator ++ adds 1 to the operand while -- subtract 1.

```
++m; or m++;
--m; or m--;
```

* we use in for and while loops.

* eg: m=5;
  y=++m; In this case, the value of y and m would be 6
* A prefix operators first adds 1 to the operand and the result is assigned to the variable.

Conditional operator

* Conditional expression of the form

$\boxed{\text{exp1 ? exp2 : exp3;}}$

where exp1, exp2, exp-3 are expression

* The operator ?: works as follows:
exp1 is evaluated first. If it is nonzero (true) then the exp.2 is evaluated and becomes the value of the expression.

* If exp1 is false, exp.3 is evaluated and becomes the value of the expression

* For eg:
```
a=10
b = 15
x = (a>b)?a:b;
```

* This can be achieved using if-else statement as follows;
```
If (a>b)
    x=a;
else
    x=b;
```

Bitwise operator

* C supports a special operator known as bitwise operators for manipulation of data at bit level.

* These operators are used for testing the bits, or shifting them right or left.

Bitwise operators

Operator	Meaning
&	bitwise AND
!	bitwise OR
<<	bitwise exclusive OR
<<	shift left
>>	shift right
~	one's complement.

Special operators

* special operators Such as commo operator, sizeof Operator, pointer operators (& and *)

The Comma operator

* The operator can be used to link the related expression together.
* The expression are evaluated left to right and the Value of rightmost expr is the value of the Combined expression.
* For. eg:
  $Value = (x=10, y=5, x+y)$
  first assigns the Value 10 to x, then assigns 5 to y and finally assigns 15 (ie 10+5) to value.

The sizeof operator

* The sizeof is a compile time operator and, when used with an operand, it returns the number of bytes the operand occupies.

* Example:
```
m = sizeof(sum);
n = sizeof(long int);
k = sizeof(235L);
```

* The sizeof operator is normaly used to determine the length of arrays and structure when their size are not known to the programmer.

Arithmetic Expression

* Arithmetic expression is a combination of variables, constants, and operator

Evaluation of expressions

* Expressions are evaluated using an assignment statement of the form

  variable = experssion;

Precedence of Arithmetic operators

* An arithmetic exp. without paranthesis will be evaluated from left to right using the rules of precedence of operators.

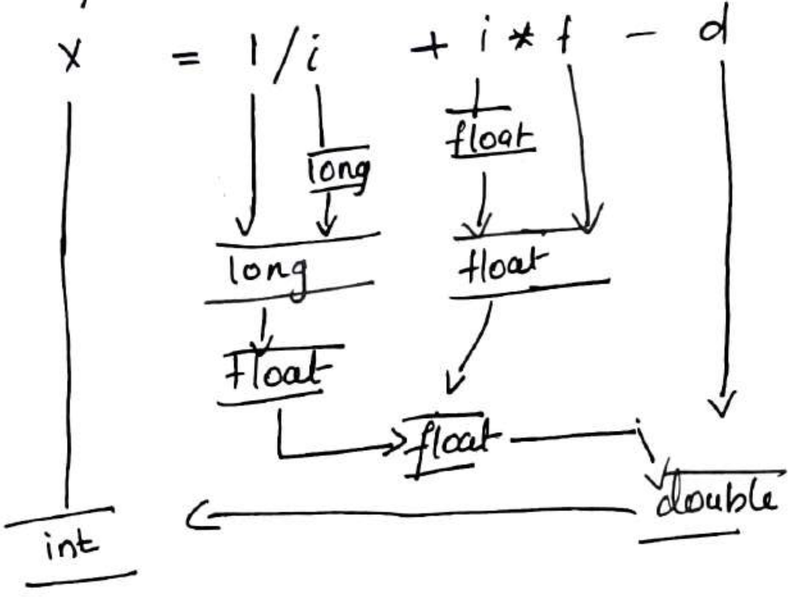
High priority	* / %
Low priority	+ -

Type conversions in expressions

Automatic type conversion.

* If the operands are of different types, the 'lower' type is automatically converted to the 'higher' type before the operation proceeds.

```
int i, x;
float f;
double d;
long int l:
```



* The following changes are introduced during the final statement

1. float to int → fractional part trunation.
2. double to float . causes rounding of digits.
3. long int to int → dropping of the excess higher order bits.

Chapter - 4.

MANAGING INPUT AND OUTPUT OPERATIONS.

(4.1)
Introduction:

Most programs take some data as input and display the processed data. often known as information or results. on a suitable medium. So far we have seen two methods of providing data to the program variables. one method is to assign values to variables through the assignment statements such as x=5: a=0: and so on. Another method is to use the input function scanf which can read data from a terminal. We have used both the methods in most of our earlier example programs.

```
# include <math.h>.

# include < stdio.h>.
```

The file name stdio. h is an abbreration for standard input-output header file. The instruction #include <stdio.h> tells the compiler 'to search for a file named stdio.

Reading A character: (4.2).

(i) The simplest of all input/output operations is reading a character from the standard input unit (usually the keyboard) and writting it to the standard ouptput unit.

(ii) Reading a single character can be done by using the function getchar. With the help of the scanf function which is discussed in the getchar takes the following form

Syntax: variable-name = getchar();

Ex:

```
    char name;
    name = getchar();
```

Example 4.1:

Shows the use of getchar function in an interactive environment.

The program displays a question of YES/NO Type to the user and reads the user's response in a single character (Y or N). If the response is Y. It outputs the message

My name is BUSY BEE.

otherwise, outputs "You are good for nothing"

The getchar function may be called successively to read the characters contained in a line of text.

Example:

```
        . . . . . .
        . . . . . .
        char character;
        character = ' ';
        While (character != '\n')
        {
          character = getchar();
        }
        . . . . . .
        . . . . . .
```

Example:

The user to enter a character and displays a message on the screen telling the user whether the character is an alphabet or digit, or any other special character.

isalpha ~~so~~ assumes a value non-zero (TRUE) if the argument character contains an alphabet: otherwise it assumes 0 (FALSE). Similar is the case with the function isdigit.

C supports many other similar functions which are given in These character functions are contained in the file ctype.h. and therefore the statement.

include <ctype.h>.

must be included in the program

```
Program.
/* * * * * * * * * * * * * * * * * * * * */
/*          TESTING CHARACTER TYPE.          */
/* * * * * * * * * * * * * * * * * * * * * */.
# include  <stdio.h>
# include  < ctype .h>
main ()
{ char character;
   printf ("Press any key \n");
   character = getchar ();
```

```
if (isalpha (character) >0)
printf (" The character is a letter.");

else
if ( isdigit (character) >0).
   printf (" The character is a digit.");
   else.
    Print (" The character is not alphonumeric.");
```

4.3 Writing a character :-

* Like getchar, there is an analogous function the putchar for writing characters one at a time to the terminal.

```
putchar(variable_name);
```

* where variable.name is a type char variable Containing a character. This statement displays the character contained in the variable.name at the terminal. For example, the statements

```
answer = 'y';
putchar(answer);
```

will display the character y on the screen. The statement

```
putchar('\n');
```

would cause the cursor on the screen to move to the beginning of the next line.

Example :

A program that reads a character from keyboard & then prints it in reverse Case is given in below. That is, if the input is upper Case, the output will be lower Case & vice-versa.

* The program uses three new functions:

  * islower
  * toupper
  * tolower.

* The dunction islower is a Conditional function & takes the value TRUE if the argument is a lower case alphabet; otherwise takes the value FALSE. The function toupper Converts the lower Case argument into an upper Case alphabet while the function tolower does the reverse.

Program

```
/* * * * * * * * * * * * * * * * * * */
/*     WRITING A CHARACTER TO THE TERMINAL  */
/* * * * * * * * * * * * * * * * * * * */
#include <stdio.h>
# include <ctype.h>
main( )
{
      char alphabet ;
      printf("Enter an alphabet");
      putchar('\n');
      alphabet = getchar( );
      if (islower (alphabet));
         putchar (toupper (alphabet));
      else
         putchar(tolower (alphabet));
}
```

4.4 : FORMATTED INPUT

Formatted input refers to an input data that has been arranged in a particular format. For example, consider the following data :

15.75 123 John .

This line contains three pieces of data, arranged in a particular form. Such that data has to be read conforming to the format of its appearance. For example, the first part of the data should be read into a variable float, the second into int, and the third part into char. This is possible in c using the scanf function.

We have already used this input function is a number of examples. Here we shall explore all of the options that are available for reading the formatted data with scanf function. The general form of scanf is

scanf ("control string", arg 1, arg 2, ..., arg n) ;

The control string specifies the field format in which the data is to be entered and the argument are arg1, arg2, ..., argn . Specify the address of location where the data is stored.

INPUTTING INTEGER NUMBERS

The field specification for reading an integer number is

%wd .

The percentange sign (%) indicates that a conversion specification follows. w is an integer number that specifies the field width of the number to be read and d, known as data types character, indicates that the number to be read is in integer mode. Consider the following : scanf ("%2d %5d", &num1, &num2);

Data line :

50 31426 .

The value 50 is assigned to num1 and 31426 to num 2. Suppose the input follows : 31426 50

The variable num1 will be assigned 31 and num2 will be assigned 426. The value 50 that is unread will be assigned to the first variable in the next scanf call. This kind of errors may be eliminated if we use the field specifications without the field width specifications. That is, the statement

```
scanf("%d, %d", &num1, &num2)
```

will read the data

```
31426  50
```

correctly and assign 31426 to num1 and 50 to num2.

An input field may be skipped by specifying * in the place of field width. For example, the statement:

```
scanf ("%d %*d %d", &a, &b)
```

will assign the data

```
123  456  789
```

as follows:

123 to a.
456 skipped (because of *)
786 to b.

INPUTTING REAL NUMBERS:

Unlike integer numbers, the field width of real numbers is not to be specified and therefore scanf reads real numbers using the simple specification %f for both the notations, namely decimal part notation and exponential notation. For example, the statement

```
scanf ("%f %f %f", &x, &y, &z);
```

with the input data

```
475.89  43.21E-1 678.
```

will assign the value 475.89 to x, 4.321 to y and 678.0 to z. The input field specifications may be separated by any arbitrary blank spaces.

A number may be skipped using %*f specification.

Example : Reading the real numbers (in both decimal point and exponential notation) is illustrated below.

Program

```
main ()
{
    float x,y;
    double p,q;
    printf ("values of x and y :");
    Scanf ("%f  %e", &x, &y);
    printf ("\n");
    printf ("x = %f \ny = %f \n\n", x,y);
    prinf f ("values of p and q :");
    Scanf ("%lf %lf", &p,&q);
    prinf ("\n\np = %.12lf \np = %.12e", p,q);
}
```

output

```
Values of x and y :12.3456 17.5e-2
x = 12.345600
y = 0.175000

Values of p and q :4.142857142857 18.5678901234567890

P = 4.142857142857
q = 1.856789012846e+001 .
```

INPUTTING CHARACTER STRINGS :

We have already seen how a single character can be read from the terminal using the getchar function. The same can be achieved using the scanf function also. In addition, a scanf function can input strings containing more than one character. Following are the specifications for reading character strings :

% ws or % wc .

The corresponding argument should be a pointer to a character array. However, %c may be used to read a single character when the argument is a pointer to a char variable.

READING MIXED DATA TYPES

It is possible to use one scanf statement to input a data line containing mixed mode data. In such cases, care should be exercised to ensure that the input data items that match the control specifications in order and type. When an attempt is made to read an item that does not match the type expected. the scanf does not read any further and immediately returns the values read. The statement

```
scanf ("%d %c %f %s", &count, &code, &ratio, name);
```

will read the data

```
15 p 1.575 coffee
```

correctly and assigns the values to the variables in the order in which they appear. The input data is converted to the type specified in the control string.

DETECTION OF ERRORS IN INPUT:

When a scanf function completes reading its list, it returns the value of number of items that are successfully read. This value can be used to test whether any errors occured reading the input. For example, the statement

```
scanf("%d %f %s, &a, &b, name);
```

will return the value 3 if the following data is typed in;

```
20 150.25 motor.
```

and will return the value 1 if the following line is entered

```
20 motor 150.25
```

This is because the function would encounter a string when it was expecting a floating-point value, and would therefore terminate its scan after reading the first value.

POINTS TO REMEMBER WHILE USING Scanf.

If we do not plan carefully, som 'crazy' things can happen with Scanf. Since the I/O routines are not a part of C language, they are made available either as a Separate module of the C library or as a part of the operating System (like UNIX). New features are added to these routines from time to time as new versions of systems are released. We should consult the System reference manual before using these routines. Given below are some of the general points to keep in mind while writing a Scanf Statement.

1. All function arguments, except the control String, must be pointers to variables.
2. Format Specifications contained in the control String should match the arguments in order.
3. Input data items must be Separated by spaces and must match variables receiving the input in Same order.
4. The reading will be terminated, when Scanf encounters a 'mismatch' of data or a character that is not valid for the value being read.
5. When Searching for a value, Scanf ignores line boundaries and Simply looks for the next appropriate character.
6. Any unread data items in a line will be considered as part of the data input line to the next Scanf call.
7. When the field width Specifier w is used, it Should be large enough to contain the input data Size.

4.5:-

FORMATTED OUTPUT:-

we have seen the use of printf function for printing captions are numerical results.

The general form of printf statement is alignment and spacing of print-outs on the terminals. The general form of printf statement is

```
Printf("control string", arg1.arg2,.... argn);
```

Control String consists of three types of items:-

1. characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item.
3. Escape sequence characters such as \n, \t, and \b.

A simple format specification has the following form:-

```
%. w.p type-specifier.
```

Output of Integer Numbers:-

The format specification for printing an integer number is

$\boxed{\%\ wd}$

Where w specifies the minimum field width for the output. However, If a number is greater than the specified field width, it will be printed in full, overriding the minimum specification. d specifies that the value to be printed is an integer.

Output of Real Numbers:-

The output of a real number may be displayed in decimal notation using the following format specification:

$\boxed{\%\ w.p\ f}$

we can also display a real number in exponential notation by using the specification

$\boxed{\%\ w.p\ e}$

Some systems also support a special field specification character that lets the user define as field size at run-time. This takes the following form:

Printf("% ","1", width. precision. number);

In this case, both the field width and the precision are given as arguments which will supply the values for w and p. For example.

```
Printf ("%.""f", 7, 2, number);
```

is equivalent to.

```
Printf ("%.7.21", number);
```

The advantage of this format is that the values for width and precision may be supplied a run-time. thus making the format a dynmic ~~the~~ one. For example, the above statement can be used as follows:

```
Int width =7;
Int precision= 2;
......
......
Printf ("%.""f", width, precision, number);
```

Chapter - 5

Decision Making And Branching

Introduction: 5.1

We have seen that a c program is a set of statements which are normally executed sequentially in the order in which they appear. This happens when no options or repetions of certain calculation are necessary

c language possesses such decision making capabilities and supports the following statements known as control or decision making statements

1. If statement
2. switch statement
3. conditional operator statement
4. goto statement

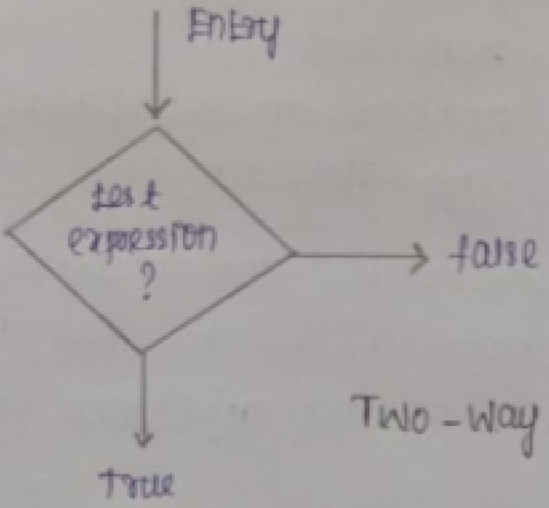
We have already used some of these statements in the earlier examples. Here we shall discuss their features, capabilities and applications in more details.

Decision Making With If statement: 5.2

The if statement is a powerful decision making statement and is used to control the flow of execution of statements. It is basically a two-way decision statement and is used in conjunction with an expression.

If (test expression)

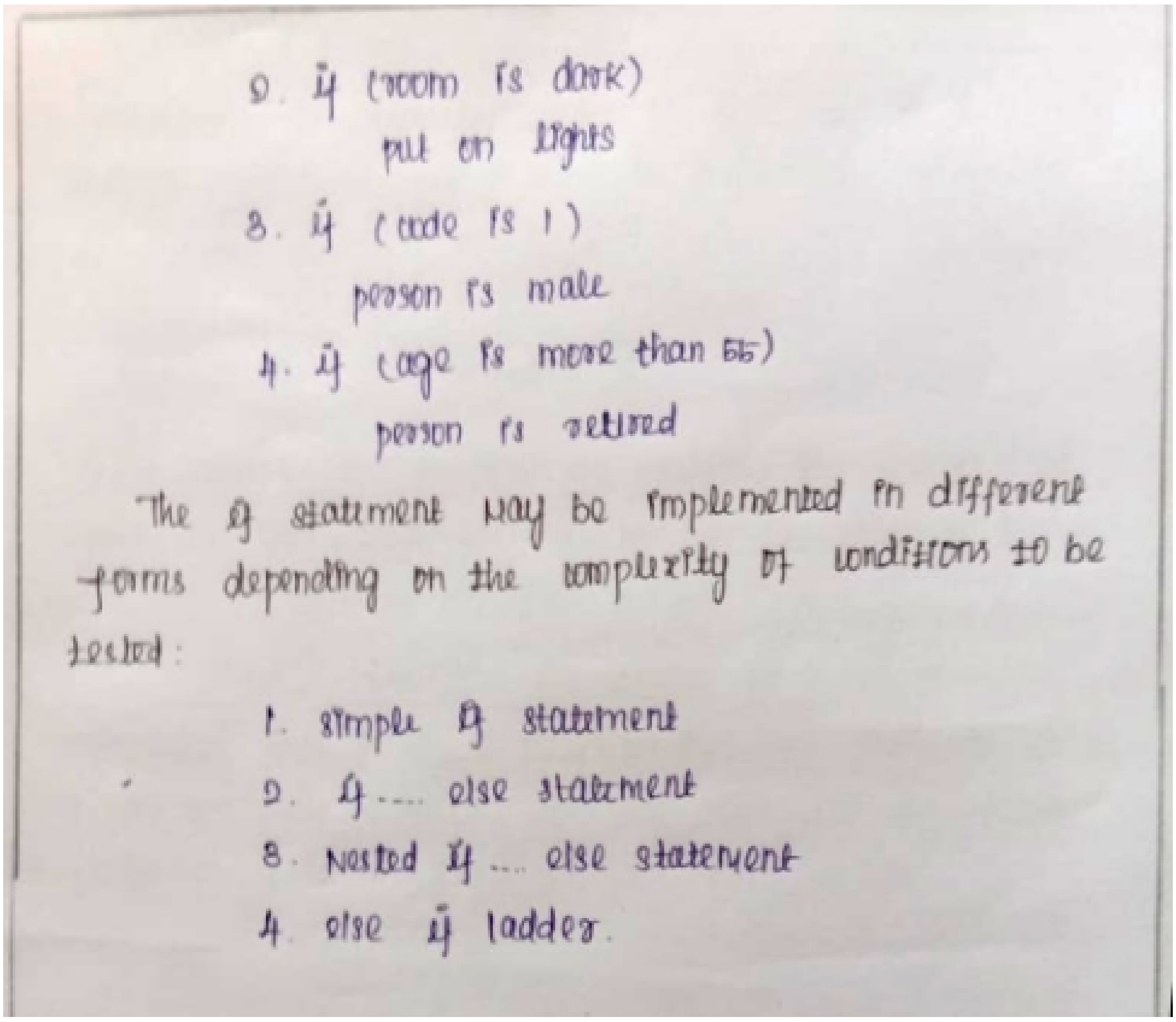
It allows the computer to evaluate the expression first and then, depending on whether the value of the expression (relation or condition) is 'true' (non-zero) or false (zero). it transfers the control to a particular statement. This point of program has two paths to follow one for the true condition and the other for the false condition as shown in fig.



Two-Way branching.

Some example of decision making, using if statement are:

1. if (bank balance is zero)
   borrow money



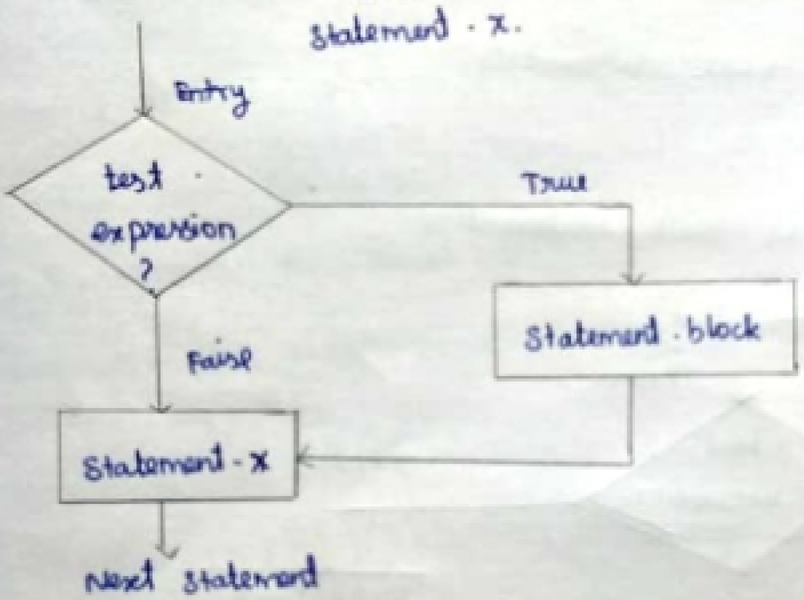
2. if (room is dark)
   put on lights
3. if (code is 1)
   person is male
4. if (age is more than 55)
   person is retired

The if statement may be implemented in different forms depending on the complexity of conditions to be tested:

1. simple if statement
2. if.... else statement
3. Nested if.... else statement
4. else if ladder.

5.3 The general form of a simple if statement is

```
If (test  expression)
  {
  statement -block ;
  }
 statement . x.
```



cassider the following segment of a program that is written for proccesing of marks obtained is an entrance examination.

```
If (category = = sports
   { marks = marks + bonus - marks ;
   }
 print (" %.1" , marks);
```

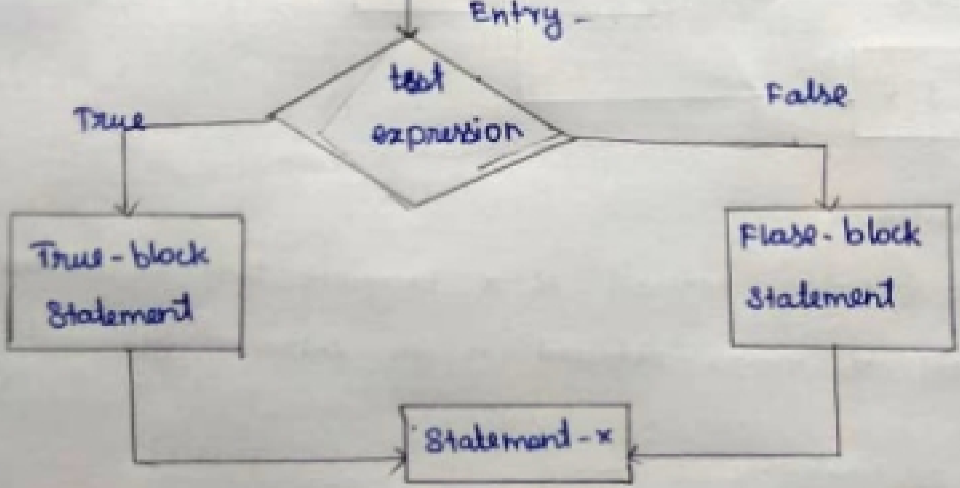
The program tests the type of category of the student. If the student belongs to the SPORTS category, then additional bonus_marks are added to his marks before they are printed For others bonus_marks are not added.

5.4

The IF - ELSE STATEMENT:

The if else statement is an extension of the simple if Statement. The general form is

```
If (test expression)
 {
 True -block statement(S)
 }
else
 {
  Flase - block statement(S)
  }
Statement- x
```



Let us consider an example of counting the number of boys and girls in a class. We use code 1 for a boy and 2 for a girl. The program statement to do this may be written as follows

```
If (code = = 1)
    boy = boy +1 :
If (code = = 2)
    girl = girl +1 :
```

Once a student is identified as a boy. there is a need to test again for a girl. A student can be either a boy or a girl, not both. The above program segment can be modified using the else co clause as follows.

```
If (code = = 1)
    boy = boy +1 :
  else
    girl = girl +1 :
  xxxxxxx xx x

If (c-d != 0)
  {
  ratio = (float) (a+b)/(float) (c-d):
  Print ("Ratio = % f\n", ratio)
  }
  else
  Print"(c-d is zero\n"):
    ........
    ......
```

5.5 Nesting of IF ... Elsestatements.

when a series of decisions are involved, we have to use more than one if ... else statement in nested form as follows:-

Program:-

```
/ .........................................../
/      EXPERIMENT WITH if....else STATMENTS/
/ .........................................../

# define ACCURACY 0.0001
main ()
{
int n, count ;
float x , term, sum ;
Printf (" Enter value of x ");
Scanf ("% f", &x );
n = term = sum = count = 1;
while (n <= 100 )
{
  term = term * x/n
  sum  = sum + term
  count = count + 1;
  if ( term < ACCURACY)
  n = 999;
else
  n = n+1;
}
```

```
   Printf (" Terms= %d Sum = %.f\n", count,sum);
  }
```

out put.

```
    Enter value of x:0
    Terms = 2 Sum =1.000000
    Enter value of x:01
    Terms = 15 Sum = 1.105171
    Enter value of x:05
    Terms = 7Sum =1.648720
    Enter value of x:075
    Terms = 8 Sum = 2.116997
    Enter value of x:0.99
    Terms=9sum = 2 6911232
    Enter value of x:1
    Terms =9Sum =2.718279
```

syntax:-

```
    If (test condition)
    {
     if(test condition 2)
      {
       Satement -1;
      }
     else
```

```
   {
     statement 2;
   }
  }
  else
   {
     Statement 3
   }
  stantement -x;
```

If the condition-1 is false the statement-3 will be executed; otherwise it continues to perform the second test If the condition 2 is true, the statement-1 will be evaluted; otherwise the statement -2 will be evaluated and then the control is transferred to the statement -x.

Example :-

```
        ......
        If (sex is female)
          {
           if (balance > 5000)
             bonus = 0.05 * balance;
           else
             bonus = 0.02 * balance.
          }
         else
          {
            bonus = 0.02 * balance
          }
         balance = balance + bonus;
         .........
```

5.6 THE ElSE IF LADDER

There is another way of putting its together when multipath decisions are involved. A multipath decision is a chain of a its in which the statement associated with each else is an if. It takes the following general form.

```
If {condition 1}
  Statement 1
else if (condition 2}
      Statement - 2
    else if (condition 3?
          Statement - 2
        else if {condition n}
              Statement  n
            else
          default statement
Statement
```

This construct is known as the else if ladder. The conditions are evaluated from the top, downwards As soon as a true conditions is forward. the statement associated with it and control is trans-ferred to the statement

Average marks	Grade
80 to 100	Honours
60 to 79	First Division
50 to 59	Second Division
40 to 49	Third Division
0 to 39	Fail

```
if { marks > 79}
  Grade = " Honours" }
  else if { marks > 59 }
       {grade = "First Division" }
       else if {marks > 49 }
         Grade - "Second Division" }
       else { if {marks > 39 }
            Grade = "Third Division" }
          else
           {grade = "Fail" }
       Printf {" /. \n" , grade }
       ______
       ______
        if {code == 1}
           colour = "RED"
          ==
```

In such situations, the choice of the method is left to the programmer. However, in order to choose an if structure that is both effective and efficient, it is important that the programmer fully aware of the various forms of an if statement and the rules governing their nesting.

5.7 THE SWITCH STATEMENT:

The complexity of such a

Program

```
/*********************************\
/*      USE OF else if LADDER      */
/*********************************/
main()
{
  int units, custnum
  float charges;

  Printf("Enter CUSTOMER NO. and UNITS consumed\n")
  scanf("%d %d", &custnum, & units);
  if(units< =200)
    charges =0.5* units;
  else if(units < =400)
       charges =100+0.65(units -200);
   else if(units < =600)
        charge s =230+0.8*(units-400),
       else
        charges =390+(units-600);

   Printf("\n\n customer NO ; % d; charges =% .2f\n"
        custnum, charges);
 }
output
Enter CUSTOMER NO. and UNITS cunsomed 101 150
customer NO:101 charges= 75.00
Enter CUSTOMER NO. and UNITS cunsumed 202 225
Customer NO :202   charges= 116.25
Enter CUSTOMER NO. and UNITS cunsumed 303 375
customer NO : 303   charges = 213.75
Enter CUSTOMER NO. and UNITS cunsumed 404 520
Customer NO ;404 Charges =326.00
Enter CUSTOMER NO. and UNITS cunsumed 505 625
customer NO :505    Charges=415.00
```

```
------
------
index = Marks/10;
Switch(index)
{
 case10;
 case 9;
 Case 8;
        grade = "Honours";
        break;
 Case 7;
 case 6;
        grade = "First Division";
        break;
 case 5;
        grade = "Second Division";
        break;
Case 4;
        grade = "Third Division";
        break;
defalut:
        grade = "Fail";
        break;
}
Printf("% s
```

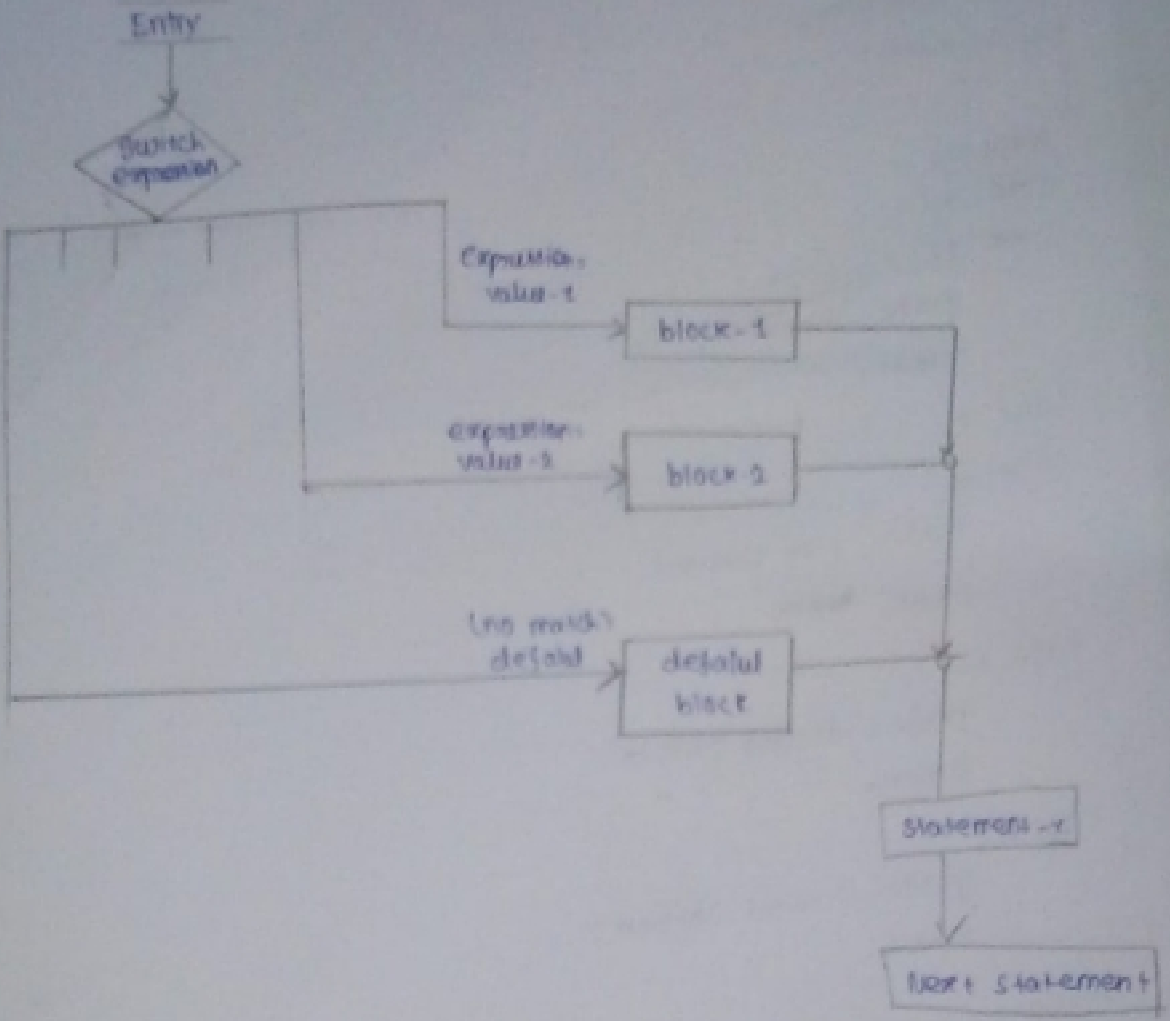
Program increases dramatically when the number of alternatives increases. The program becomes difficulty to read and follow. At times, it may confuse even the person who designed.

C has a built-it multiway decision statement known as a switch statement sets the value of given variable (or expression) against a list of case values and when a match is found, a block of statement associate with that case is executed.

```
switch expression
{  Case value-1;
                    block-1
                    break;

   case value-2;
                    block-2
                    break;

   ------
   ------
   default;
                    defalut block.
                    break;
}
  Statement-x;
```



Note that we have used a conversation Statement

index = marrs /10;

The first three cases will execute the same.

grade = "Honours";
break;

THE ?: OPERATOR

The C language has an unusual operator, useful for making two-way decisions. This operator is a combination of ? and : and takes three operands. This operator is popularly known as the conditional operator. The general form of use of the conditional operator is as follows:

conditional expression ? expression1 : expression2

The conditional expression is evaluated first. If the result is nonzero, expression is evaluated and is returned as the value of the conditional expression. Otherwise, expression 2 is evaluated and its value is returned.

```
If (x<0)
    flag = 0;
else
    flag = 1;
```

can be write as flag = (x<0) ? 0 : 1;

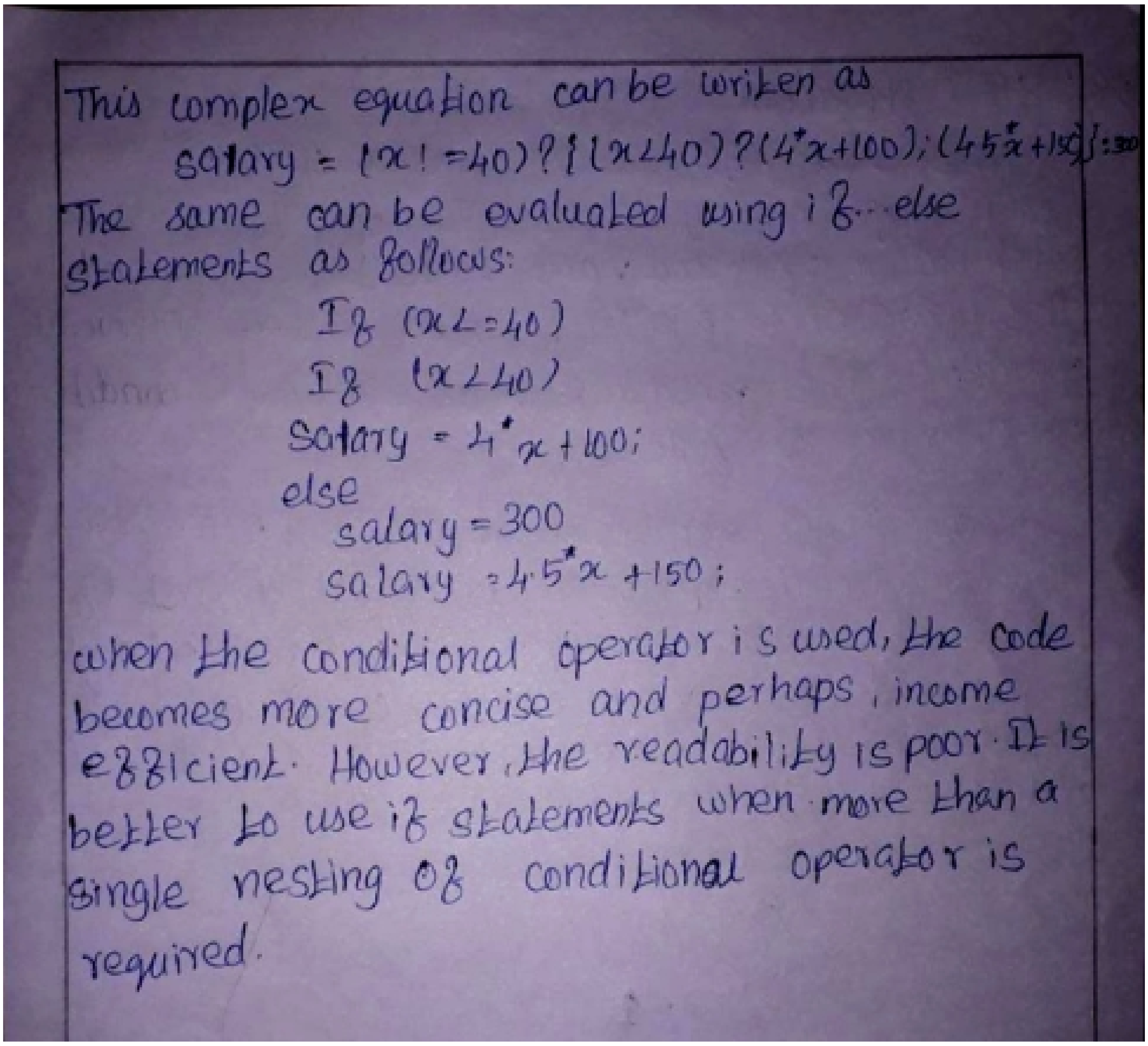
Consider the evaluation of the following function;

$y = 1.5x + 3$ for $x \le 2$

$y = 2x + 5$ for $x > 2$

This can be evaluated using the conditional operators as follows: y = (x>2) ? (2*x+5) : (1.5*x+3);

$$\text{salary} = \begin{matrix} 4x + 100 \\ 300 \\ 4.5x + 150 \end{matrix}$$



This complex equation can be writen as

```
salary = (x!=40)?{(x<40)?(4*x+100);(4.5*x+150)}:300
```

The same can be evaluated using if...else statements as follows:

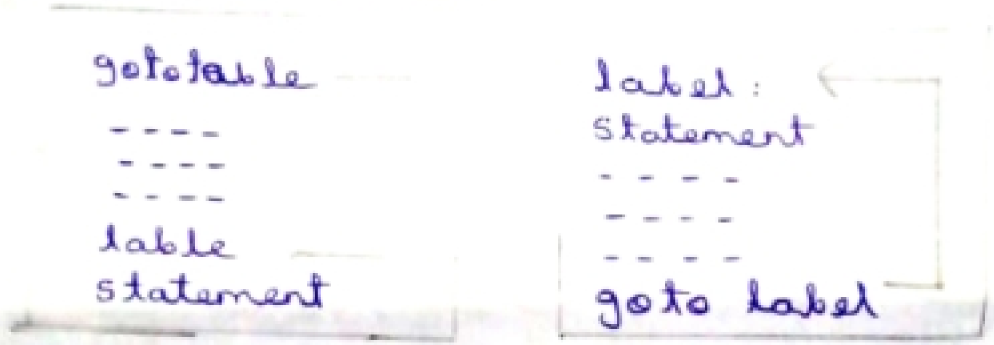
```
          If (x<=40)
          If (x<40)
        salary = 4*x+100;
        else
          salary=300
          salary =4.5*x +150;
```

when the conditional operator is used, the code becomes more concise and perhaps, income efficient. However, the readability is poor. It is better to use if statements when more than a single nesting of conditional operator is required.

THE GOTO STATEMENT

C supports the goto statement to branch uncoundtionally from one point to another in the program.

The goto requiers a label in order to identify the place where the branch is to made A table is any valid variable name. and must be followed by a colon. The label is placed immediately before the statement. The general forms of goto and label statements are shown below.



forward jumb

Backwuark jumb

Can be anywhere in the program either before or after the goto lable, statement. During runnig of a program are show like.

A goto is often used at the end of a Program to direct the control to go to the input statement, to read further data.

The program is written to evalute the square root of a series of numbers read from the terminal. The Program uses to goto statement.