C.P.PATEL & F.H.SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA(ITM) & PGDCA PROGRAMME
BCA SEM : 01 US01CBCA21 : Programming Fundamental Using C

Unit 2 – Basics of Programming

Unit 2	Basics of Programming
•	History and Importance of C
•	Basic Structure of C Programming
•	Variables and Constant
•	Data types in a high-level language
•	User Defined Type declaration - Typedef
•	Operators and Expressions & its type conversion
•	Formatted I/O statements
•	Assignment statements

➢ History and Importance of C

C language is one of the most popular computer languages today because it is a structured, high level, machine independent language. It allows software developers to develop software without worrying about the hardware platforms where they will be implemented.

The following flowchart shows the history of ANSI C .

1960	ALGOL	International Group
1967	BCPL	Martin Richards
1970	B	Ken Thompson
1972	Traditional C	Dennis Ritchie
1978	K&R C	Kernighan & Ritchie
1989	ANSI C	ANSI Committee
1990	ANSI/ISO C	ISO Committee

➢ Importance of C:

C is a very important language because of its many useful features. The increasing popularity of C is due to its many desirable qualities. It is a robust language whose rich set of built in functions and operators can be used to write any complex program. It allows software developers to develop software without worrying about the hardware platforms where they will be implemented. The C compiler combines the capabilities of an assembly

language with the features of a higher level language and therefore it is well suited for both system software and business packages. Else these,

1. C language is efficient and fast.
2. C is highly portable.
3. C language is well suited for structured programming.
4. C is a machine independent language.
5. C has the ability to extend itself.

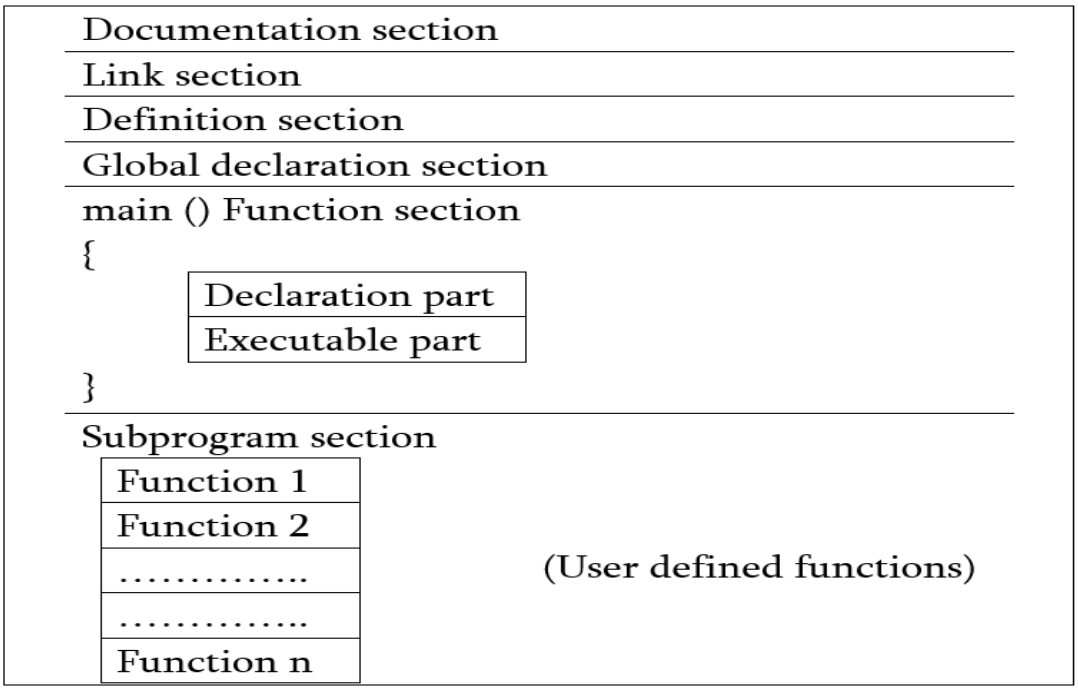
➢ Problems & its type, General structure of C program:

Problem Analysis

If we are to use the computer as a problem-solving tool, then we must have a good analysis of the problem given. Here are some suggested steps on how to go about analyzing a certain problem for computer application:

1. Review the problem carefully and understand what you are asked to do.
2. Determine what information is given(input) and what result must be produced(output).
3. Assign names to each input and output items.
4. Determine the manner of processing that must be done on the input data to come up with the desired output(i.e., determine what formulas are needed to manipulate the given data).

Format of simple C programs:




```
E.g. main()
{
/*……print begins…….*/
        printf("I see, I remember");
/*……printing ends……*/
}
```

1. **Documentation section:** The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later.
2. **Link section:** The link section provides instructions to the compiler to link functions from the system library.
3. **Definition section:** The definition section defines all symbolic constants.
4. **Global declaration section:** There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section that is outside of all the functions. This section also declares all the user-defined functions.
5. **main () function section:** Every C program must have one main function section. This section contains two parts; declaration part and executable part

1. **Declaration part:** The declaration part declares all the variables used in the executable part.
2. **Executable part:** There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.

6. **Subprogram section:** The subprogram section contains all the user-defined functions that are called in the main () function. User-defined functions are generally placed immediately after the main () function, although they may appear in any order.

All section, except the main () function section may be absent when they are not required.

➢ 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:

➢ C Character Set

Letters:
Upper Case A….Z Lower Case a….z
Digits:
All decimal digits 0…9

Special Characters:

, Comma
. Period
; Semicolon
: Colon
? Question mark
' Aphostophe
" Quotation mark
! exclamation mark
| vertical bar
/ slash
\ backslash
~ tilde
_ underscore
$ dollar sign
< opening angle bracket(or less than sign)
> closing angle bracket(or greater than sign)
& ampersand
^ carot
* asterisk
- minus sign
+ plus sign
% per cent sign
# number sign
(left paranthesis
) right paranthesis
[left bracket
] right bracket
{ left brace
} right brace

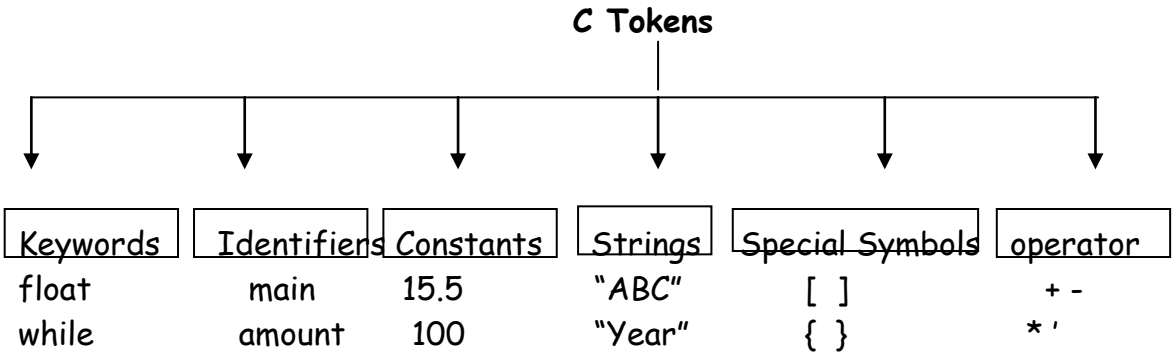
White spaces:

Blank space
Horizontal tab
Carriage return
New line
Form feed

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. C programs are written using these tokens and the syntax of the language.

C tokens and examples



➢ Keywords and identifiers

Every C word is classified as either a keyword or an identifier.
All keywords have fixed meaning and these meanings cannot be changed. Keywords serve as basic building blocks for program statements

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 uppercase and lowercase letters are permitted, although lowercase letters are commonly used. The underscore character is also permitted in identifiers. It is usually used as a link between two words in long identifiers.

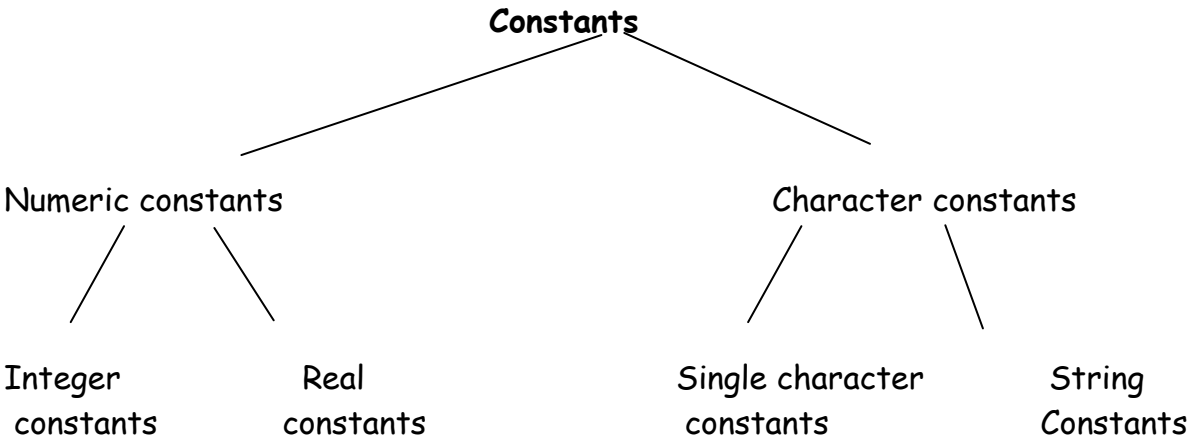
ANSI C Keywords

Auto	break	case	char	const	continue
Default	do	double	else	enum	extern
Float	for	Goto	if	int	long
Register	return	short	signed	sizeof	static
Struct	switch	typedef	union	unsigned	void
Volatile	while				

➢ Constants

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

➢ **Basic types of C constants**



Integer Constants

An integer constant refers to a sequence of digits. There are three types of integers, namely, decimal, octal and hexadecimal.

Decimal integers consist of a set of digits, 0 through 9, preceded by an optional – or + sign. Valid examples of decimal integer constants are : 123,-21,0,564553,+74.

Embedded spaces, commas, and non-digit characters are not permitted between digits. For example, 15,750 , $2000 are illegal numbers. Note that ANSI C supports unary plus which was not defined earlier.

An Octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0. Some examples of octal integers are: 037,0.0456.

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 constants. Further e.g. are: 0.0083,-0.75,+245.0.

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.

Single Character Constants

A single character constant contains a single character enclosed within a pair of single quote marks. E.g. are '5', 'X', ';', ' '.

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

Character constants have integer values known as ASCII values. For e.g. 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'.

Since each character constant represent an integer value it is also possible to perform arithmetic operations on character constants.

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".

Remember that a character constant is not equivalent to the single character string constant. Further, a single character string constant does not have an equivalent integer value while a character constant has an integer value.

Backslash Character Constants

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

Backslash character constants

Constant	Meaning
'\a'	audible alert(bell)
'\b'	back space
'\f'	form feed
'\n'	new line
'\r'	carriage return
'\v'	vertical tab
'\t'	horizontal 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.

A variable name can be chosen by the programmer in a meaningful way so as to reflect its function or nature in the program. E.g. Average, height.

Variable names may consist of letters, digits, and the underscore(_) character, subject to the following conditions:

1. They must begin with a letter. Some systems 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 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 a keyword.
5. White space is not allowed.

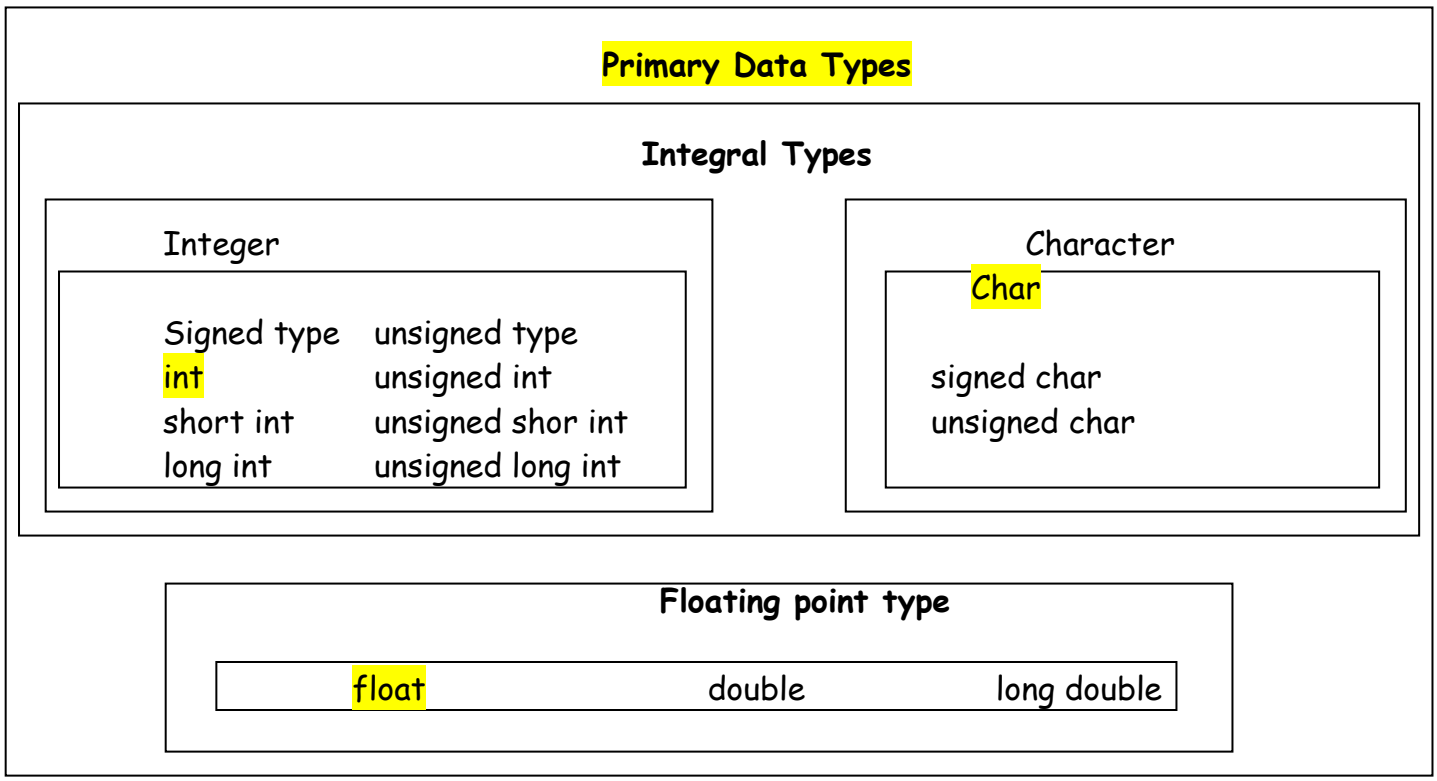
Some examples of valid variable names are: John, Value, T_raise.
Invalid examples include: 123, (area), %, 25^{th}.
If only the first eight characters are recognized by a compiler, then the two names average_height & average_weight mean the same thing to the computer. Such names can be rewritten as avg_height and avg_weight or ht_average and wt_average without changing their meanings.

- **Data types**

Four classes of data types:

1. **Primary data types**
2. **User-defined data types**
3. **Derived data types**
4. **Empty data set**

All C compilers support four fundamental data types, namely integer (int), character (char), floating point (float), and double-precision floating point (double). Many of them also offer extended data types such as long int and long double. Various data types are given.



Integer Types

Integers are whole number 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 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, -215 to +215-1). A signed integer ranging from -2,147,483,648 to 2,147,483,647.

Size and range of data types on a 16-bit machine

Type	Size(bit)	Range
char or signed char	8	-128 to 127
unsigned char	8	0 to 255
int or signed int	16	-32,768 to 32767
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 32 bits (on all 16 bit machines), with 6 digits of precision. Floating point numbers are defined in C by the keyword float. When the accuracy provided by a float number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. Remember that double type represents the same data type that float represents, but with a greater precision. To extend the precision further, we may use long double which uses 80 bits.

Character types

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**

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 to 255, signed chars have values from –128 to 127.

After designing suitable variable names, we must declare them to compiler. Declaration does two things:

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

Primary type declaration

A variable can be used to store a value of any data type. That is, the name has nothing to do with its type. The syntax for declaring a variable is as follows:

data-type v1,v2,….vn;

v1,v2,…vn are the names of variables. Variables are separated by commas. A declaration statement must end with a semicolon. For e.g. int count;
int is the keyword to represent integer type and real type data values respectively.

- **Data types and their keywords**

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

The program segment 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.

User-defined types

The **typedef keyword** in C++ allows creating user-defined data types. To do this, simply specify a new data type name for an already existing data type. The new data type is not created.

A new name for the existing type is defined instead. User-defined types make applications more flexible: sometimes, it is enough to change typedef instructions using substitution macros (#define).

User-defined types also improve code readability since it is possible to apply custom names to standard data types using typedef.

The general format of the entry for creating a user-defined type:

typedef type new_name;

Here, type means any acceptable data type, while new_name is a new name of the type. A new name is set only as an addition (not as a replacement) to an existing type name. MQL5 allows creating pointers to functions using typedef.

- **Declaration of variables**

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

- **Assigning values to variables**

Assigning Statement

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

variable_name = constant;

Example: initial_value = 0;

C permits multiple assignments in one line. For example

intial_value = 0; final_value = 100;

are valid statements.

An assignment statement implies that the value of the variable on the left of the 'equal sign' is 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.

During assignment operation, C converts the right-hand side to the type on the left. This may involve truncation when real value is converted to an integer.

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

data-type variable_name = constant;

Example:

int final_value = 100;

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

p = q = s = 0;

are valid. The statement initializes the variables p, q, and s to zero.

Remember that external and static variables are initialized to zero by default.

Reading data from keyboard

The general format of scanf is as follows:

scanf ("control string", &variable1, &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's 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 waits for the value of the variable number to be typed in. Since the control string "%d" specifies that the integer value is to be read from the terminal, we have to type in the value in integer form. Once the number is typed in and the 'Return' key is pressed, the computer then proceeds to the next statement. Thus, the use of scanf provides an interactive feature and makes the program 'user friendly'.

Printing data

The general purpose of printf statement is to print the message. Example:

printf ("Hello Everybody");

The general format of printf is as follows:

printf ("control string", variable1, variable2,….);

The control string contains the format of data being received. Example:

printf ("%d",number);

When this statement is encountered by the computer, the computer prints the value of the variable that is read by the user. Since the control string "%d" specifies that the integer value is read from the terminal, we will get the value in integer form. Once the number is printed on the monitor, the computer then proceeds to the next statement. Thus, the use of printf provides an interactive feature and makes the program 'user friendly'.

Operators

- **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

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

Integer Arithmetic

If a and b are integers, then for a = 14 and 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)

If one of them is negative, the direction of truncation is implementation dependent. That is, 6/7 = 0 and –6/-7 = 0
but –6/7 may be zero or –1.(Machine dependent)

Similarly, during modulo division, the sign of the result is always the sign of the first operand (the dividend).

That is, -14 % 3 = -2
-14 % -3 = -2
14 % -3 = 2

Real Arithmetic

An arithmetic operation involving only real operands is called real arithmetic. If x, y, and z are floats, then we will have:
x = 6.0/7.0 = 0.857143
y = 1.0/3.0 = 0.333333
z = -2.0/3.0 = -0.666667
The operator % cannot be used with real operands.

Mixed-mode Arithmetic

When one of the operands is real and the other is integer, the expression is called a mixed-mode expression. Thus 15/10.0 = 1.5 where as 15/10 = 1.

Relational operators

The comparisons can be done with the help of relational operators.

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

e.g. 4.5 <= 10 TRUE

Logical operators

In addition to the relational operators, C has the following three logical operators.

&& meaning logical AND
|| meaning logical OR
! meaning logical NOT

The logical operators && and || are used when we want to test more than one condition and make decisions. Example: a > b && x == 10

An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression. Like the simple relational expressions, a logical expression also yields a value of one or zero, according to the truth table. The logical expression given above is true only if a > b is true and x == 10 is true. If either (or both) 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
Non-zero	Non-zero	1	1
Non-zero	0	0	1
0	Non-zero	0	1
0	0	0	0

Some examples of the usage of logical expressions are:

1. if (age > 55 && salary < 1000)
2. if (number < 0 || number > 100)

Assignment operators

Syntax: v op = exp;

where v is a variable, exp is an expression and op is a C binary arithmetic operator.

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

Increment and decrement operators

These are the increment and decrement operators: ++ and –

The operator ++ adds 1 to the operand while – subtracts 1. Both are unary operators and take the following forms:

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

++m; is equivalent to m = m + 1; (or m +=1;)

--m; is equivalent to m = m – 1; (or m -=1;)

While ++m and m++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement. Consider the following:

```
m = 5;
y = ++m;
```

In case, the value of y and m would be 6. Suppose, if we rewrite the above statements as

```
m = 5;
y = m++;
```

then, the value of y would be 5 and m would be 6. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left. On other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

Conditional Operator OR Ternary Operator

A ternary operator pair "?:" is available in C to construct conditional expressions of the form:

exp1? exp2 : exp3;
where exp1, exp2 and exp3 are expressions.

The operator ?: works as follows: exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of the expression. If exp1is false, exp3 is evaluated and its value becomes the value of the expression. note that only one of the expressions (either exp2 or exp3) is evaluated. For example,

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

In this example, x will be assigned the value of b. This can be achieved using the if..else statements as follows:

```
if (a>b)
        x = a;
else
        x = b;
```

Bitwise operators

C has a distinction of supporting special operators 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 may not be applied to float or double.

Bitwise operators

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	Bitwise exclusive OR
<<	shift left
>>	shift right
~	One's complement

Special operators

C supports some special operators of interest such as comma operator, **sizeof** operator, pointer operators (& and *) and member selection operators (. and ->).

The Comma operators

The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression. For example, the statement

```
value = ( x = 10, y = 5, x+y);
```

first assigns the value 10 to x, then assigns 5 to y, and finally assigns 15 (i.e, 10+5) to value. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

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. The operand may be a variable, a constant or a data type qualifier.

Examples:

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

The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmer. It is also used to allocate memory space dynamically to variables during execution of a program.

- **Assignment Statement**

Once you've declared a variable you can use it, but not until it has been declared - attempts to use a variable that has not been defined will cause a compiler error. Using a variable means storing something in it. You can store a value in a variable using:

name = value;
For example:
a=10;
stores the value **10** in the **int** variable **a**. What could be simpler? Not much, but it isn't actually very useful! Who wants to store a known value like 10 in a variable so you can use it later? It is 10, always was 10 and always will be 10. What makes variables useful is that you can use them to store the result of some arithmetic.

Consider four very simple mathematical operations: add, subtract, multiply and divide. Let us see how C would use these operations on two float variables **a** and **b**.

Add **a+b** Subtract **a-b** Multiply **a*b** Divide **a/b**

Note that we have used the following characters from C's character set:

+ for add - for subtract * for multiply / for divide

BE CAREFUL WITH ARITHMETIC!!! What is the answer to this simple calculation?

a=10/3

The answer depends upon how **a** was declared. If it was declared as type **int** the answer will be 3; if **a** is of type **float** then the answer will be 3.333. It is left as an exercise to the reader to find out the answer for **a** of type **char**.
Two points to note from the above calculation:

1. C ignores fractions when doing integer division!
2. when doing **float** calculations integers will be converted into **float**. We will see later how C handles type conversions.

$$$====================$$$====================$$$