UNIT- I
Data Types & Operators

Before any problem can be solved using a computer, the person writing the program must be familiarized with the problem and with the way in which it has to be solved. The problem solved is to be represented by small and clear steps by using algorithms and flow charts.

A program is a set of instructions to solve a particular problem. C is a programming language, is to be considered as a middle level language. This unit gives an overview of C language and explains about the structure of C program. Variables and constants joined by various operators form an expression, Basic I/O functions are used to accept data and produces output. Different types of values may be passed to the computer from the keyboard. Such different types for the values are called as data types.

This unit will discuss in detail about the data types, variables, constants, various categories of operators, expressions, type modifiers and Input – Output operations

Program – Definition

A computer program is a sequence of instructions written to perform a specified task with a computer.

Programs are written in a programming language. These programs are then translated into machine code by a compiler and linker so that the computer can execute it directly or run it line by line (interpreted) by an interpreter program.

PROGRAM DEVELOPMENT LIFE CYCLE

The process of developing software, according to the desired needs of a user, by following a basic set of interrelated procedures is known as Program Development Life Cycle (PDLC)

Tasks of Program Development

The basic set of procedures that are followed by various organizations in their program development methods are as follows:

1. Program specification.
2. Program Design.
3. Program coding.
4. Program testing.
5. Program documentation.
6. Program Maintenance.

PROGRAMMING LANGUAGES AND FEATURES

Programming language is a set of grammatical rules for instructing a computer to perform specific tasks. The term programming language usually refers to high-level languages, such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal. There are two major types of programming languages. These are Low Level Languages and High Level Languages. Low Level languages are further divided in to Machine language and Assembly language.

LOW LEVEL LANGUAGES

Low level languages are machine oriented and require knowledge of computer hardware and its configuration.

(a) Machine Language

Machine Language is the only language that is directly understood by the computer. It does not need any translator program. It is written as strings of 1's (one) and 0's (zero). For example, a program instruction may look like this:

1011000111101

It is not an easy language to learn because of its difficult to understand. It is efficient for the computer but very inefficient for programmers. It is considered to the first generation language. It is also difficult to debug the program written in this language.

(b) Assembly Language

The computer can handle numbers and letter. Therefore, some combination of letters can be used to substitute for number of machine codes. The set of symbols and letters forms the Assembly Language and a translator program is required to translate the assembly Language to machine language. This translator program is called `Assembler'.

HIGH LEVEL LANGUAGES

The assembly language and machine level language require deep knowledge of computer hardware. But High-level languages are machine independent. Programs are written in English-like statements. As high – level languages are not directly executable; translators (compilers or interpreters) are used to convert them into machine language.

HISTORY OF C

C is a general purpose computer programming language and was developed at AT&Ts bell laboratory of USA in 1972. It was originally created by Dennis Ritchie. By

1960, different computer languages were used for different purposes. So, an International Committee was set up to develop a language that is suitable for all purposes. This language is called ALGOL 60.

To overcome the limitation of ALGOL 60, a new language called Combined Programming Language (CPL) was developed at Cambridge University. CPL has so many features. But it was difficult to learn and implement. Martin Richards of Cambridge University developed a language called "Basic Combined Programming Language" (BCPL). BCPL solves the problem of CPL. But it is less powerful. At the same time, Ken Thomson at AT&T's Bell laboratory developed a language called 'B'. Like BCPL, B is also very specific. Ritchie eliminated the limitations of B and BCPL and developed 'C'.

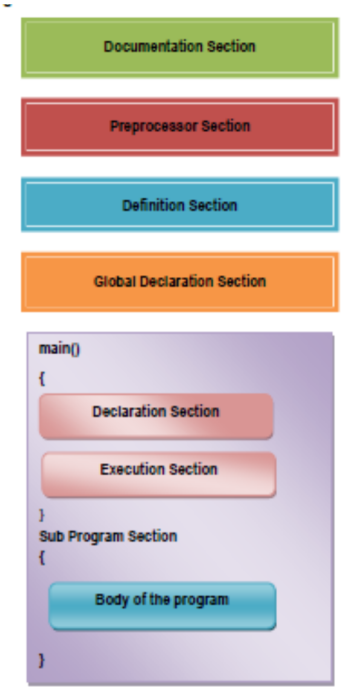
FEATURES OF C LANGUAGE

C has all the advantages of assembly language and all the significant features of modern high-level language. So it is called a "Middle Level Language".

- C language is a very powerful and flexible language.
- C language supports a number of data types and consists of rich set of operators.
- C language provides dynamic storage allocation.
- C Compiler produces very fast object code.
- C language is a portable language. A code written in C on a particular machine can be compiled and run on another machine.

STRUCTURE OF A C PROGRAM

Any C Program consists of one or more function. A function is a collection of statements, used together to perform a particular task. An overview of the structure of a C program is given below: A C program may contain one or more sections. They are illustrated below.



Documentation Section:

The documentation section consists of a set of comment lines giving the name of the program, the author and other details. It consists of a set of comment lines. These lines are not executable. Comments are very helpful in identifying the program features

Pre-processor Section:

It is used to link system library files, for defining the macros and for defining the conditional inclusion.

Definition section:

The definition section defines all symbolic constants.

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.

main () function section:

Every C program must have one main function section. This section contains two parts; declaration part and executable part.

a) Declaration part: The declaration part declares all the variables used in the executable part.

b) 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.

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.

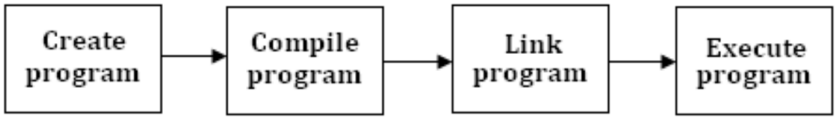
SAMPLE C PROGRAM

```
#include <stdio.h>
int main ()
{
 int number, remainder;
printf ("Enter your number to be tested: ");
scanf ("%d", &number);
remainder = number % 7;
if ( remainder == 0 )
printf ("The number is divided by 7.\n");
else
printf ("The number is not divided by 7.\n");  }
```

General rule for C Programming:

1. Every executable statement must end with semicolon symbol (;).
2. Every C Program, must contain exactly one main method (starting point of the program execution)
3. All the system defined words (keywords) must be used in lowercase letters.
4. Keywords cannot be used as user defined names.
5. For every open brace ({), there must be respective closing brace (}).

EXECUTING A "C" PROGRAM



C program executes in following four steps.

1. Creating a program: Type the program and edit it in standard 'C' editor and save the program with .c as an extension. This is the source program .The file should be saved as '*.c' extension only.
2. Compiling (Alt + F9) the Program:
   - This is the process of converting the high level language program to Machine level Language (Equivalent machine instruction) .
   - Errors will be reported if there is any, after the compilation
   - Otherwise the program will be converted into an object file (.obj file) as a result of the compilation
   - After error correction the program has to be compiled again
3. Linking a program to library: The object code of a program is linked with libraries that are needed for execution of a program. The linker is used to link the program with libraries. It creates a file with '*.exe' extension.
4. Execution of program: This is the process of running (Ctrl + F9) and testing the program with sample data. If there are any run time errors, then they will be reported.

VARIABLES, CONSTANTS AND DATA TYPES

C CHARCTER SET

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters...

1. Alphabets 2. Digits 3.Special Symbols

Alphabets: C language supports all the alphabets from English language. Lower and upper case letters together supports 52 alphabets.

- lower case letters - a to z , UPPER CASE LETTERS - A to Z

Digits: C language supports 10 digits which are used to construct numerical values in C language.

- Digits - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Symbols: C language supports rich set of special symbols that include symbols to perform mathematical operations, to check conditions, white spaces, back spaces and other special symbols.

,	Comma	&	Ampersand
.	Period	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus Sign
?	Question Mark	+	Plus Sign
'	Apostrophe	<	Opening Angle (Less than sign)
"	Quotation Marks	>	Closing Angle (Greater than sign)

!	Exclamation Mark	(	Left Parenthesis
|	Vertical Bar	)	Right Parenthesis
/	Slash	[	Left Bracket
\	Backslash	]	Right Bracket
~	Tilde	{	Left Brace
-	Underscore	}	Right Bracket
$	Dollar Sign	#	Number Sign
|	Vertical Bar	)	Right Parenthesis
/	Slash	[	Left Bracket
%	Percentage sign		

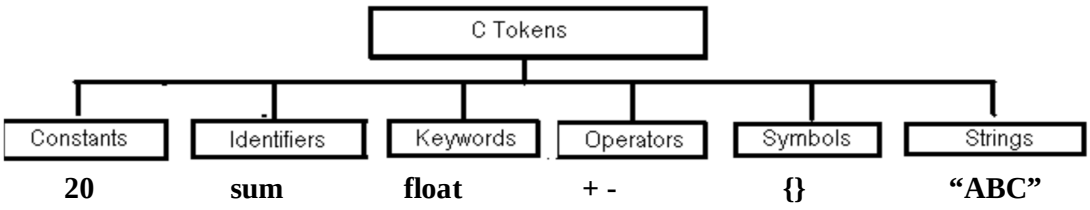
White Space Characters:

The compiler ignores white spaces unless they are a part of a string constant. White spaces may be used to separate words in strings. scanf() uses whitespace to separate consecutive input items from each other.

1. Blank Space 2. Horizontal Tab 3. Carriage Return 4. New Line 5. Form Feed

C TOKENS

C tokens are the basic buildings blocks in C. Smallest individual units in a C program are the C tokens. C tokens are of six types. The below figure shows the C tokens.



KEYWORDS

The meaning of these words has already been explained to the C compiler. All the keywords have fixed meanings. These meanings cannot be changed. So these words cannot be used for other purposes. All keywords are in lower case. The keywords are known as reserved words. Keywords or Reserved words are Pre-defined identifiers. 32 keywords are available in C.

Properties of Keywords

1. All the keywords in C programming language are defined in lowercase letters only.
2. Every keyword has a specific meaning; users cannot change that meaning.
3. Keywords cannot be used as user defined names like variable, functions, arrays, pointers etc...
4. Every keyword in C programming language represents some kind of action to be performed by the compiler.

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			

IDENTIFIERS

The names of variables, functions, labels and various other user-defined objects are called Identifiers. Identifiers are used for defining variable names, function names etc. The general rules to be followed when constructing an identifier are:

1) Identifiers are a sequence of characters. The only characters allowed are alphabetic characters, digits and the underscore character. Special characters are not allowed in identifier name.

2) The first character of an identifier is a letter or an underscore character.

3) Identifiers are case sensitive. For example, the identifiers TOTAL and Total are different.

4) Keywords are not allowed as identifier name.

VALID IDENTIFIERS:

Name area interest circum amount rate_of_interest sum

INVALID IDENTIFIERS

Identifiers	Reason
4^{th}	The first letter is a numeric digit.
int	The Keyword should not be a identifier
First name	Blank space is not allowed.

CONSTANTS

The data values are usually called as Constant. Constant is a quantity that does not change during program execution. This quantity can be stored at a location in the memory of the computer. C has four types of constants: Integer, Floating, String and Character.

INTEGER CONSTANT

An Integer Constant is an integer number. An integer constant is a sequence of digits. **There are 3 types of integer's namely decimal integer, octal integer and hexadecimal integer.**

Decimal Integer consists of a set of digits 0 to 9 preceded by an optional + or - sign. Spaces, commas and non-digit characters are not permitted between digits.

Example for valid decimal integer constants are : 123 -31 0 562321 + 78

Octal Integer constant consists of any combination of digits from 0 through 7 with a O at the beginning.

Examples of octal integers are : O26 O O347 O676

Hexadecimal integer constant is preceded by OX or Ox. They may contain alphabets from A to F or a to f. or a decimal digit (0 to 9).

The alphabets A to F refers to 10 to 15 in decimal digits.

Example of valid hexadecimal integers are : OX2 OX8C OXbcd Ox123

REAL CONSTANT

Real constants are numbers with fractional part. Real constants are often called as Floating Point constants.

RULES

1. A real constant must have at least one digit.
2. It must have a decimal point.
3. It could be either positive or negative.
4. Default sign is positive (If no sign)
5. Special characters are not allowed.
6. Omitting of digit before the decimal point, or digits after the decimal point is allowed. (Ex .655,12.)

Examples: +325.34 426.0 -32.76 -48.5792

Real Constants are represented in two forms:

i) Fractional form

ii) Exponential form

EXPONENTIAL FORM or SCIENTIFIC FORM

The Exponential form is used to represent very large and very small numbers. The exponential form representation has two parts: 1) mantissa and 2) exponent.

The part appears before the letter 'e' is called mantissa and the part following the letter 'e' is called exponent, which represents a power of ten. The general form of exponential representation is.

mantissa e exponent

Where "mantissa" is a decimal or integer quantity and the "exponent" is an integer quantity with an optional plus or minus sign.

The general rules regarding exponential forms are

1. The two parts should be separated by a letter "e" or "E".
2. The mantissa and exponent part may have a positive or a negative sign.
3. Default sign of mantissa and exponent part is positive.
4. The exponent must have at least one digit.

Examples

The value 123.4 may be written as 1.234E+2 or 12.34E+1.

The value 0.01234 may be written as 1.234E-2 or 12.34E-3.

Differences between floating point numbers and integer numbers.

- Integer includes only whole numbers, but floating point numbers can be either whole or fractional.
- Integers are always exact, whereas floating point numbers sometimes can lead to loss of mathematical precision.
- Floating point operations are slower in execution and often occupy more memory than integer operations.

CHARACTER CONSTANTS

A character constant is either a single alphabet, a single digit or a single special character enclosed within a pair of single inverted commas.

The maximum length of a character constant can be 1 character

Examples: 'A' '1' '$' ' ' ';'

Each character constant has an integer value. This value is known as ASCII value. For example, the statement printf (“”%d””,’A’) would print the value of 65. Because the ASCII value of A is 65.

Similarly the statement printf (“%c”,65) would display the letter ‘A’.

STRING CONSTANTS

A combination of characters enclosed within a pair of double inverted commas is called as “String Constant”.

Examples: “Salem” “35.567” “$125” “Rose”

Each string constant ends with a special character ‘\0’. This character is not visible when the string is displayed. Thus “salem” contains actually 6 characters. The ‘\0’ character acts as a string terminator. This character is used to find the end of a string.

Remember that a character constant (e.g., ‘A’) and the corresponding single-character string constant ("A") are not equivalent.

A character constant has an equivalent integer value, whereas a single-character string constant does not have an equivalent integer value and, in fact, consists of two characters - the specified character followed by the null character (\ 0).

VARIABLES – DEFINITION AND RULES

A quantity, which may vary during program execution, is called variable. Variables may be used to store a data value. Variables are actually memory locations , used to store constants. The variables are identified by names. The program may modify the values stored in a variable.

Rules for constructing variable names

1. A variable name is the combination of characters. The length of a variable depends upon the complier.
2. The first character must be an alphabet or underscore.
3. Special characters like comma or blank are not allowed except an underscore character. The only characters allowed are letters, digits and underscore.
4. The variable name should not be a keyword.

Examples: area interest circumference fact date_of_birth

DECLARING VARIABLES

In C, all the variables used in a program are to be declared before they can be used. All variables must be declared in the beginning of the function. Type declaration statement is used to declare the type of various variables used in the program. The syntax for declaration is,

```
data_type variable_name(s)
```

Where data-type is a valid data type plus any other modifiers. variable-name(s) may contain one or more variable names separated by commas.

Examples: int a, b, c; long int interest; unsigned char ch;

Declaration statement is used to allocate memory space for the variable. Declaration statement also provides a name for the location. Declaration statement declares that the

program will use that variable name to identify the value stored at the location.
For example, the declaration

char sub-name

Allocates a memory location of size one byte, of character type. This memory location is given a name of sub-name.

INITIALIZATION OF A VARIABLE (ASSIGNING VALUES TO VARIABLES)

The process of assigning initial values to variables is called initialization of variables. In C, an uninitialised variable can contain any garbage value. Therefore, the programmer must make sure all the variables are initialised before using them in any of the expressions. The value for a particular variable is initialized through declarative statement or assignment statement. For example to initialize the value 10 to an integer variable i, the following two methods are used.

i) int i = 10;
ii) int i;
i = 10;

The above two methods declares that i is an integer variable with an initial value of 10.

Examples : (i) float pi = 3.14; (ii) char alpha = ''h''; (iii) int account = 10;

More than one variable can be initialized in a single statement. For example, the statement

int x = 1, y = 2, z;

declares x to be an int with value 1, y to be an int with value 2 and z to be an int of unpredictable value. Same value can be initialized to several variables with the single assignment statement.

Examples

```
int i, j, k, l, m, n;
float a, b, c, d, e, f;
i = j = k = l = m = n = 20;
a = b = c= d = e = f = 13.56;
```

DECLARING VARIABLES AS CONSTANTS

A constant is a quantity whose value does not change during program execution. The qualifier const is used to declare the variable as constant at the time of initialization. The general form to declare variables as constant is

```
const data type variable = value;
```

Where const - keyword data type - valid type such as int, float etc.

Example : const float PI = 3.14;

const is a new data type qualifier. This tells the computer that the value of the float variable PI must not be modified by the program. But , it can be used on the right-hand side of an assignment like any other variable.

DECLARING VARIABLES AS VOLATILE

The qualifier volatile is used to tell explicitly the compiler that a variable's value may be changed at any time by some external sources not by the program(from outside the program). General form to declare a variable as volatile is

```
volatile data type variable;
```

where volatile -keyword data type - valid types such as int, float etc.

Example : volatile int x;

The value of x may be altered by some external factors even if it does not appear on the left-hand side of an assignment statement. When declaring a variable as volatile, the compiler will examine the value of the variable each time it is encountered to see whether any external alteration has changed the value.

DATA TYPES

C supports several data types. Each data type may be represented differently inside the computer's memory. There are four data types in C language. They are,

Types	Data Types
Basic data types	int, char, float, double
Enumeration data type	enum
Derived data type	pointer, array, structure, union
Void data type	void

Basic Data Types

Basic or Fundamental data types include the data types at the lowest level. i.e. those which are used for actual data representation in the memory. All other data types are based on the fundamental data types.

Examples: char, int, float, double.

int type is used to store positive or negative integers. float data type is used to store a single precision floating-point (real) number. Floating-point numbers are stored in 32 bits with 6 digits of precision. double data type is used to hold real numbers with higher storage range and accuracy than the type float. The data type char is used to store one character.

The size (number of bytes) and range of numbers to be stored in each data type is shown below.

Data Type	Size (Bytes)	Range
char	1	-128 to 127
int	2	-32,768 to 32, 767
float	4	3.4 E-38 to 3.4 E+38
double	8	1.7 E-308 to 3.4 E+308

Derived Data Types

These are based on fundamental data types. i.e. a derived data type is represented in the memory as a fundamental data type.

Examples: pointers, structures, arrays.

Void data type:

1. void is an empty data type that has no value.
2. This can be used in functions and pointers.

DATA TYPES MODIFIERS (QUALIFIERS)

The basic data type may be modified by adding special keywords. These special keywords are called data type modifiers (or) qualifiers. Data type modifiers are used to produce new data types.

The modifiers are: signed unsigned long short

The above modifiers can be applied to integer and char types. Long can also be applied to double type.

Integer Type

A signed integer constant is in the range of -32768 to +32767. Integer constant is always stored in two bytes. In two bytes, it is impossible to store a number bigger than +32767 and smaller than -32768. Out of the two bytes used to store an integer, the leftmost bit is used to store the sign of the integer. So the remaining 15 bits are used to store a number. If the leftmost bit is 1, then the number is negative. If the leftmost bit is 0, then the number is positive.

C has three classes of integer storage namely short int, int and long int. All of these data types have signed and unsigned forms. A short int requires half the space than normal size. Unsigned numbers are always positive and consume all the bits for storing the magnitude of the number. The long and unsigned integers are used to declare a longer range of values.

Floating Point Type

Floating point number represents a real number with 6 digits precision. When the accuracy of the floating point number is insufficient, use the double to define the number. The double is same as float but with longer precision. To extend the precision further , use long double, which consumes 80 bits of memory space.

Character Type

Characters are usually stored in 8 bits of internal storage. The qualifier signed or unsigned can be explicitly applied to char. While unsigned characters have values between 0 and 255, signed characters have values from –-128 to 127.

Size and Range of Data Types on 16 bit machine.

TYPE SIZE (Bits) Range	TYPE SIZE (Bits) Range	TYPE SIZE (Bits) Range
Char or Signed Char	8	-128 to 127
Unsigned Char	8	0 to 255
Int or Signed int	16	-32768 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	-2147483648 to 2147483647
Unsigned long int	32	0 to 4294967295
Float	32	3.4 e-38 to 3.4 e+38
Double	64	1.7e-308 to 1.7e+308
Long Double	80	3.4 e-4932 to 3.4 e+4932

OVERFLOW AND UNDERFLOW OF DATA

Assigning a value which more than the upper limit of the data type is called overflow and less than its lower limit is called underflow.

In case of integer types, overflow results wrapping towards negative side and underflow results wrapping towards positive side.

In case of floating point types, overflow results +INF and underflow results –INF.

Example 1:

```
#include <stdio.h>
void main()
{
int a = 32770;
printf( “%d””,a);
}
```

Output : - 32766

The range of integer is – -32768 to + 32767, assigning 32770 results overflow and wrap towards –negative side.

Example 2:

```
#include <stdio.h>
void main()
{
int a = 33000;
float b = 3.4e50;
printf( “%d%f””,a,b);
}
```

Output : - 32536 +INF

The range of integer is – -32768 to + 32767, assigning 33000 results overflow and wrap towards –negative side and +INF is a result of float overflow.

COMMENTS

Comments are used to make a program more readable. Comments are not instructions. Comments are remarks written in a program. These remarks are used to give more information about the program. The compiler will ignore the comment statements.

In C, there are two types of comments.

1. Single Line Comments: Single line comment begins with // symbol. Any number of single line comments can be written.
2. Multiple Lines Comments: Multiple lines comment begins with /* symbol and ends with */. Any number of multiple lines comments can be included in a program.

In a C program, the comment lines are optional. All the comment lines in a C program just provide the guidelines to understand the program and its code.

Examples:

(e.g.) /* Program to find the Factorial */

Any number of comments can be placed at any place in a program. Comments cannot be nested.

For example

/* Author Arul /* date 01/09/93 */ */ is not possible.

A comment can be split over more than one line. For example, the following is valid.

/* This statement is used to find the sum of two numbers */

ESCAPE SEQUENCES

Character combinations consisting of a backslash (\) followed by a letter or by a combination of digits are called "escape sequences." To represent a newline character, single quotation mark, or certain other characters in a character constant, escape sequences are used. An escape sequence is regarded as a single character and is therefore valid as a character constant.

- The escape sequence characters are also called as backslash character constants.
- These are used for formatting the output

For example, a line feed (LF), which is referred to as a newline in C, can be represented as \n. Such escape sequences always represent single characters, even though they are written in terms of two or more characters. The commonly used escape sequences are listed below.

ESCAPE CHARACTER	MEANING	ESCAPE CHARACTER	MEANING ESCAPE
carriage return	\r	carriage return	\r
backspace	\b	quotation mark (")	*
horizontal tab	\t	apostrophe (')	\’
vertical tab	\v	question mark (?)	\?
newline (line feed)	\n	backslash	\\
form feed	\f	null	\0

The following program outputs a new line and a tab and then prints the string This is a test.

```
#include <stdio.h>
int main()
{
printf("\n\tThis is a test.");
return 0;
}
```

Characteristics

- Although it consists of two characters, it represents single character.
- Every combination starts with back slash(\)
- They are non-printing characters.
- It can also be expressed in terms of octal digits or hexadecimal sequence.
- Each escape sequence has unique ASCII value.

SYMBOLIC CONSTANTS

- Names given to values that cannot be changed. Implemented with the #define pre-processor directive.

```
#define N 3000
#define FALSE 0
```

- Preprocessor statements begin with a # symbol, and are NOT terminated by a semicolon. Traditionally, preprocessor statements are listed at the beginning of the source file.
- Preprocessor statements are handled by the compiler (or preprocessor) before the program is actually compiled. All # statements are processed first, and the symbols (like N) which occur in the C program are replaced by their value (like 3000). Once this substitution has taken place by the preprocessor, the program is then compiled.
- In general, preprocessor constants are written in UPPERCASE. This acts as a form of internal documentation to enhance program readability and reuse.
- In the program itself, values cannot be assigned to symbolic constants.

Use of Symbolic Constants

- Consider the following program which defines a constant called TAXRATE.

```
#include <stdio.h>
#define TAXRATE 0.10
main ()
{
float balance;
float tax;
balance = 72.10;
tax = balance * TAXRATE;
printf("The tax on %.2f is %.2f\n",balance, tax);
}
```

C OPERATORS

C has a very rich set of operators. So C language is sometimes called “the language of operators”. Operators are used to manipulate data and variables. An operator is a symbol, which represents some particular operation that can be performed on some data. The data itself (which can be either a variable or a constant) is called the 'operand'.

Operators operate on constants or variables, which are called operands. Operators can be generally classified as either unary, binary or ternary operators. Unary operators act on one operand, binary operators act on two operands and ternary operators operate on three operands.

Depending on the function performed, the operators can be classified as

Arithmetic	Relational	Logical	Conditional	Assignment
Increment & Decrement	Modulo division	Bitwise	Special operators	

ARITHMETIC OPERATORS

Arithmetic operators are used to perform arithmetic operation in C. Arithmetic operators are divided into two classes:

(i) Unary arithmetic operators (ii) Binary arithmetic operators

Binary operators

Operator	Meaning
+	Addition or Unary Plus
--	Subtraction or Unary Minus
*	Multiplication
/	Division
%	Modulus Operator

Arithmetic Operators +, -, * and / can be applied to almost any built-in data types. Suppose that a and b are integer variables whose values are 10 and 3, respectively. Several arithmetic expressions involving these variables are shown below, together with their resulting values.

Operation	Value
a+ b	13
a–b	7
a* b	30
a/ b	3
a%b	1

Modulo operator: The modulo operator (%) gives the remainder of the division between the two integer values. For Modulo division, the sign of result is always that of the first operand or dividend.

For example, 13%-5 = 3; -13 % - 5 = -3; -13% 5 = -3; 13% 5 = 3;

Example :

```
main()
{
int a, b,c,d;
a = 10;
b = 4;
c= a/b;
d = a % b;
printf(“”%d %d””, c, d);
}
```

Modulo division operator cannot be used with floating point type. C does not have no option for exponentiation.

Unary minus operator

In unary minus operation, minus sign precedes a numerical constant, variable or an expression. A negative number is actually an expression, consisting of the unary minus operator, followed by a positive numeric constant. Unary minus operation is entirely different from the subtraction operator (-). The subtraction operator requires two operands.

INCREMENT AND DECREMENT OPERATORS

C contains two special operators ++ and --. ++ is called Increment operator. -- is called Decrement operator; The above two operators are called unary operators since they operate on only one operand. The operand has to be a variable and not a constant. Thus, the expression 'a++' is valid whereas '6++' is invalid.

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

- Syntax:

  Increment operator: ++var_name; (or) var_name++;

  Decrement operator: – --var_name; (or) var_name –--;

If the operator is used before the operand, then it is called prefix operator. (Example: ++a, --a). If the operator is used after the operand, then it is called postfix operator. (Example: a++, a--).

Difference between pre/post increment & decrement operators in C:

Below table will explain the difference between pre/post increment and decrement operators.

Operator	Description
Pre increment operator (++i)	Pre increment operator (++i)
value of i is incremented before assigning it to the variable i	value of i is incremented before assigning it to the variable i
Post increment operator (i++)	Post increment operator (i++)
value of i is incremented after assigning it to the variable i	value of i is incremented after assigning it to the variable i

Prefix and Postfix operators have the same effect if they are used in an isolated C statement. For example, the two statements

x++; and ++x; have the same effect

But prefix and postfix operators have different effects when they are assigned to some other variable. For example the statements

z = ++x; and z = x++; have different effects.

Assume the value of x to be 10. The execution of the statement z= ++x; will first increment the value of x to 11 and assign new value to z. The above statement is equal to the following two statements.

```
x = x + 1 ;
z = x ;
```

The execution of the statement z = x++; will first assign the value of z to 10 and then increase the value of x to 11. The above statement is equal to

```
z = x ;
x = x + 1 ;
```

The decrement operators are also in a similar way, except the values of x and z which are decreased by 1.

Other Examples

a = 10 , b=6	a=10, b = 6
c = a * b++	c = a * ++b
Output:	
c = 60;	c = 70

The expression n++ requires a single machine instruction such as INR to carry out the increment operation. But n+1 operation requires more instructions to carry out this operation. So the execution of n++ is faster than n+ 1 operation.

RELATIONAL OPERATORS

Relational operators are used to test the relationship between two operands. The operands can be variables, constants or expressions. C has six relational operators. They are

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

An expression containing a relational operator is called as a relational expression. The value of the relational expression is either true or false. If it is false, the value of the expression is 0 and if it is true, the value is 1.

Examples:

Suppose that i,j and k are integer variables whose values are 1, 2 and 3, respectively. Several relational expressions involving these variables are shown below.

Expression	Interpretation	Value
i < j	true	1
(1 + j) >= k	true	1
(j + k) > (i + 5)	false	0
k != 3	false	0
j == 2	true	1

LOGICAL OPERATORS

Logical operators are used to combine or negate expression containing relational expressions. C provides three logical operators.

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

Logical expression is the combination of two or more relational expressions. Logical operators are used to combine the result of evaluation of relational expressions. Like the simple relational expression, a logical expression also gives value of one or zero.

LOGICAL AND (&&)

This operator is used to evaluate two conditions or expressions simultaneously. If both the expressions to the left and to the right of the logical operator is true then the whole compound expression is true.

Example : a > b && x = = 10

The expression to the left is a > b and that on the right is x == 10.The whole expression is true only if both expressions are true i.e., if a is greater than b and x is equal to 10.

LOGICAL OR (||)

The logical OR is used to combine two expressions or the condition. If any one of the expression is true, then the whole compound expression is true.

Example : a < m || a < n

The expression evaluates to true if any one of them is true or if both of them are true. It evaluates to true if a is less than either m or n and when a is less than both m and n.

LOGICAL NOT (!)

The logical not operator takes single expression and evaluates to true if the expression is false and evaluates to false if the expression is true. In other words, it just reverses the value of the expression.

Examples

Suppose that i is an integer variable whose value is 7, f is a floating-point variable whose value is 5.5, and c is a character variable that represents the character 'w '. Several complex logical expressions that make use of these variables are shown below.

Expression	Interpretation	Value
(i >= 6) && (c == ' w')	true	1
(i >= 6) ‖ (c == 119)	true	1
(f < 11) && (i > 100)	false	0
(c != ' p ')‖ ((i + f) <= 10)	true	1

The truth table for the logical operators is shown here using 1's and 0's.

p	q	p && q	p ‖ q	!p
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

CONDITIONAL OPERATORS

Simple conditional operations can be carried out with the conditional operator (?:). The conditional operator is used to build a sionditional expression. The Conditional operator has two parts: ? and:

An expression that uses the conditional operator is called conditional expression. The conditional operator is a ternary operator because it operates on three operands. The general form is

```
expression1? Expression 2: expression 3;
```

The expression1 is evaluated first. If it is true (non – zero), then the expression 2 is evaluated and its value is returned. If expression 1 is false (zero), then expression 3 is evaluated and its value is returned. Only one expression either expression 2 or expression 3 will be evaluated at a time.

Conditional expression frequently appears on the right hand side of a simple assignment statement. The resulting value of the conditional expression is assigned to the identifier on the left.

Example : max = c >d ? c : d;

The purpose of the above statement is to assign the value of c or d to max, whichever is larger. First the condition (c > d) is tested. If it is true, max = c; if it is false max = d;

ASSIGNMENT AND SHORT-HAND ASSIGNMENT OPERATORS

The Assignment Operator (=) evaluates an expression on the right hand side of the expression and substitutes this value to the variable on the left hand side of the expression.
Example : x = a + b ;

Here the value of a + b is evaluated and substituted to the variable x.

In addition, C has a set of shorthand assignment operators. Short hand assignment operators are used to simplify the coding of a certain type of assignment statement. The general form of this statement is

```
var oper = exp;
```

Here var is a variable, exp is an expression or constant or variable and oper is a C binary arithmetic operator. The operator oper = is known as shorthand assignment operator

The above general form translates to : var = var op exp;

The compound assignment statement is useful when the variable name is longer. For example

amount-of-interest = amount-of-interest * 10 ;

can be written as

amount-of-interest * = 10 ;

For example a = a + 1; can be written as a + = 1

The short hand works for all binary operators in C. Examples are

x - = y;	is equal to	x = x - y;
x * = y;	is equal to	x = x * y;
x / = y;	is equal to	x = x / y;
x %= y;	is equal to	x = x % y;

BITWISE OPERATORS

The combination of 8 bits is called as one byte. A bit stores either a 0 or 1. Data are stored in the memory and in the registers as a sequence of bits with 0 or 1.

Bitwise operators are used for manipulation of data at bit level. The operators that are used to perform bit manipulations are called bit operators. C supports the following six bit operators.

Operator	Description	Operator	Description
&	Bitwise AND	~	One's Complement
|	Bitwise OR	<<	Shift left
^	Bitwise X-OR	>>	Shift right

These operators can operate only on integers and not on float or double data types. All operators except ~ operator are binary operators, requiring two operands. When using the bit operators, each operand is treated as a binary number consisting of a series of individual 1s and 0s. The respective bits in each operand are then compared on a bit by bit basis and result is determined based on the selected operation.

Bitwise AND operator

Bitwise AND operator (&) is used to mask off certain bits. The result of AND ing operation is 1 if both the bits have a value of 1; otherwise it is 0. Assume the two variables a and b, whose values are 12 and 24. The result of the statement c = a & b is

a	0	0	0	0	1	1	0	0
b	0	0	0	1	1	0	0	0
c=a & b	0	0	0	0	1	0	0	0

The value of c is 8. To mask the particular bit (s) in a value, AND the above value with another value by placing 0 in the corresponding bit in the second value.

Bitwise OR operator

Bitwise OR operator (|) is used to turn ON certain bits. The result of ORing operation is 1 if any one of the e bits have a value of 1; otherwise it is 0. Assume the two variables a and b, whose values are 12 and 24. The result of the statement c = a | b is

a	0	0	0	0	1	1	0	0
b	0	0	0	1	1	0	0	0
c=a|b	0	0	0	1	1	1	0	0

The value of c is 28. To turn ON a particular bit(s) in a pattern of bits, OR the above value with another value by placing 1 in the corresponding bit in the second value.

Bitwise Exclusive OR operator

The result of bitwise Exclusive OR operator (^) is 1 if only one of the bits is 1; otherwise it is 0. The result of a ^ b is

a	0	0	0	0	1	1	0	0
b	0	0	0	1	1	0	0	0
c= a^b	0	0	0	1	0	1	0	0

The value of c is 20.

Bitwise complement operator

The complement operator ~ complements all the bits in an operand. That is, 0 changes to 1 and 1 changes to 0. If the value of a is 00001100, then ~a is 11110011.

Shift operators

The shift operators are used to move bit pattern either to the left or right. The general forms of shift operators are

Left shift: operand << n Right shift: operand >> n

Where operand is an integer and n is the number of bit positions to be shifted. The value for n must be an integer quantity.

Assume the value of a is 12. Then the result of a << 2 will be follows:

a	0	0	0	0	1	1	0	0
a<<2	0	0	1	1	0	0	0	0

The value of c is 48. The above operation shifts the bits to the left by two places and the vacated places on the right side will be filled with zeros. **Every shift to the left by one position corresponds to multiplication by 2.** Shifting two positions is equal to multiplication by 2*2 i.e, by 4. Similarly, every shift to the right by one position corresponds to division by 2.

SPECIAL OPERATORS

C supports some special operators like comma operator, sizeof operator, pointer operators (& and *) and member selection operators (. And ->).

Comma Operator:

Comma operator is used in the assignment statement to assign many values to many variables.

Example: int a=10, b= 20, c- 30;

Comma operator is also used in for loop in all the three fields.

Example: for (i=0, j=2; i<10; i=i+1;j=j+2);

Size of Operator:

Another unary operator is the sizeof operator . This operator returns the size of its operand, in bytes. This operator always precedes its operand. The operand may be a variable, a constant or a data type qualifier. Consider the following program.

```
main()
{
int sum;
printf("%d", sizeof(float));
printf("%d", sizeof (sum));
printf("%d", sizeof (234L));
printf("%d", sizeof ('A')); }
```

Here the first printf( ) would print out 4 since a float is always 4 bytes long. With this reasoning, the next three printf( ) statements would output 2, 4 and 2. Consider an array school[]= “National” . Then, sizeof (school) statement will give the result as 8.

ORDER OF OPERATIONS (PRECEDENCE AND ACCOCIATIVITY)

The order in which the operations are performed in an expression is called hierarchy of operations. The priority or precedence of operators is given below.

The arithmetic operators have the highest priority. Both relational and logical operators are lower in precedence than the arithmetic operators (except!). The shorthand assignment operators have the lowest priority.

Order	Category	Operator	Operation	Associativity
1.	Highest precedence	( ) [ ] → : : .	Function call	L → R Left to Right
2.	Unary	! ~ + - ++ - - & *	Logical negation (NOT) Bitwise 1’s complement Unary plus Unary minus Pre or post increment Pre or post decrement Address	R → L Right -> Left

		Size of	Indirection Size of operant in bytes	
3.	Multiplication	* / %	Multiply Divide Modulus	L → R Left to Right
4.	Additive	+ -	Binary Plus Binary Minus	L → R Left to Right
5.	Shift	<< >>	Left shift Right shift	L → R Left to Right
6.	Relational	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	L → R Left to Right
7.	Equality	== !=	Equal to Not Equal to	L → R Left to Right
8.	Bitwise AND	&	Bitwise AND	L → R Left to Right
9.	Bitwise XOR	^	Bitwise XOR	L → R Left to Right
10.	Bitwise OR	|	Bitwise OR	L → R Left to Right
11.	Logical AND	&&	Logical AND	L → R Left to Right
12.	Conditional	? :	Ternary Operator	R → L Right -> Left
13.	Assignment	= *= %= /= += -= &= ^= |= <<= >>=	Assignment Assign product Assign reminder Assign quotient Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	R → L Right -> Left
14.	Comma	,	Evaluate	L → R Left to Right

An expression within the parentheses is always evaluated first. One set of parentheses can be enclosed within another. This is called nesting of parentheses. In such cases, innermost parentheses are evaluated first.

Associatively means how an operator associates with its operands. For example, the unary minus associated with the quantity to its right, and in division the left operand is divided by right. The assignment operator '=' associates from right to left. Associativity also refers to the order in which C evaluates operators in an expression having same precedence.

For example, the statement a = b = 20 / 2; assigns the value of 10 to b, which is then assigned to ‘a’, since associativity said to be from right to left.

EVALUATION OF EXPRESSIONS

Example 1:

Determine the hierarchy of operations and evaluate the following expression:

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

Stepwise evaluation of this expression is shown below:

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

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

i = 1 + 4 / 4 + 8 - 2 + 5 / 8 operation: /

i = 1 + 1+ 8 - 2 + 5 / 8 operation: /

i = 1 + 1 + 8 - 2 + 0 operation: /

i = 2 + 8 - 2 + 0 operation: +

Note that 6 / 4 gives 1 and not 1.5. This so happens because 6 and 4 both are integers and therefore would evaluate to only an integer constant. Similarly 5 / 8 evaluates to zero, since 5 and 8 are integer constants and hence must return an integer value.

Example 2:

Determine the hierarchy of operations and evaluate the following expression:

k = 3 / 2 * 4 + 3 / 8 + 3

k = 3 / 2 * 4 + 3 / 8 + 3

k = 1 * 4 + 3 / 8 + 3 operation: /

k = 4 + 3 / 8 + 3 operation: *

k = 4 + 0 + 3 operation: /

k = 4 + 3 operation: +

k = 7 operation +

ASSIGNMENT STATEMENTS

Assignment statements are used to assign the values to variables. Assignment statements are constructed using the assignment operator (=). General form of assigning value to variable is

variable = expression

where expression is a constant or a variable name or the expression (Combinations of constants, variables and operators).

Examples : (i) a = 5; (ii) a = b; (iii) a = a + b; (iv) a = a > b;

Rules to be followed when constructing assignment statements

1. Only one variable is allowed on the left hand side of ‘=’ expression a = b * c is valid, whereas a + b = 5 is invalid.

2. Arithmetic operators can be performed on char, int, float and double data types. For example the following program is valid, since the addition is performed on the ASCII value of the characters x and y.

```
char a,b;
int z;
a = 'x';
b = 'y'
z = a +b;
```

3. All the operators must be written explicitly. For example D = x. y .z is invalid. It must be written as D = x * y * z.

Multiple Assignment Statement

The same value can be assigned to more than one variable in a single assignment. This is possible by using multiple assignments. For example to assign the value 30 to the variable a, b, c, d is

```
a = b = c = d = 30;
```

However, this cannot be done at the time of declaration of variables.

For example

int a = b = c = d = 30 is invalid.

EXPRESSIONS

An expression is the combination of Variables and Constants connected by any one of the arithmetic operator. Examples for expressions are:

(i) a + b (ii) b+5 (iii) a+b*5-6/c (iv) 8.2+ a/b+6.7

Integer expression: If all the variables and constants in an expression are of integer type, then this type of expression is called as integer expression. An integer expression will always give a result in integer value.

Real Expression: If all variables and constants in an expression are of real type, then this type of expression is called as real expression, A real expression will always give a result in real value.

Mixed mode Expression: If the elements in an expression are of both real and integer types, then this type of expression is called as mixed mode expression. A mixed mode expression will always give a result in real value

Examples:

Integer Expression	Real Expression	Mixed Mode Expression
int a,b,c;c = a+b/5 + c/a;	float a,b,c; c = a+b/2.0 + c+ 5.6	int a,b; float c,d; d = 5 + a/c + c/b + 6.1