

Basics Of a Computer Program

i) Algorithm

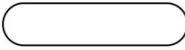


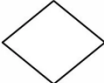

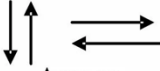



An algorithm is a set of instructions designed to perform a specific task. It can be implemented using a formal language, and such a language is known as a *programming language*. In general terms, an algorithm provides a blueprint to write a program to solve a particular problem. It is considered to be an effective procedure for solving a problem in finite number of steps

ii) Flowchart

A flowchart is a graphical representation of an algorithm. Programmers often use it as a program-planning tool to solve a problem. It makes use of symbols that are connected among them to indicate the flow of information and processing. The process of drawing a flowchart for an algorithm is known as flowcharting.

Flowchart Symbols

Here is a chart for some of the common symbols used in drawing flowcharts.

Name	Symbol	Purpose
Terminal	 oval	Start /stop/begin/end
Input / output	 Parallelogram	Input/output of data
Process	 Rectangle	Any processing to be performed can be represented
Desicion box	 Diamond	Decision operations that determine which of the alternative paths to be followed
Connector	 Circle	Used to connect different parts of flowchart
Flow	 Arrows	Joins 2 symbols and also represents flow of execution
Pre defined process	 Douuble sided rectangle	Modules (or)subroutines specified else where
For loop symbol	 Hexagon	Shows initialization, condition and incrimination of loop variable.
Document	 Printout	Shows the data that is ready for printout

iii) Pseudocode

Pseudo code is a simplified representation of an algorithm that uses the English language to describe coding logic. It allows programmers to plan any algorithm's structure using simple commands.

Advantages of pseudo code

There are several benefits of using pseudo code on a programming project, including:

Evaluation: Writing pseudo code can help you evaluate the logic of an algorithm and solve problems before spending time and resources writing the technical code.

Collaboration: The simple language in pseudocode makes it a valuable tool for explaining complex technical details to team members who may have little programming knowledge.

Translation: Once you complete your pseudocode, you can use it as a clear outline for translating each line into a programming language.

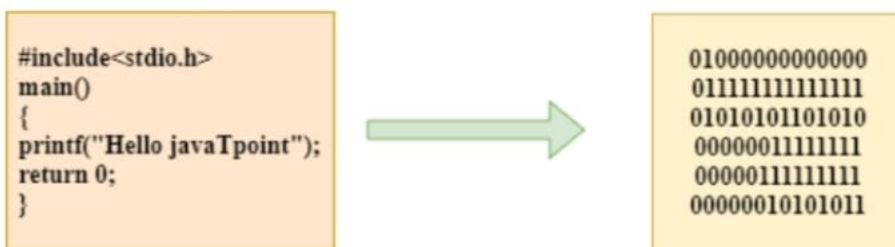
Documentation: Saving your pseudocode documents can help you record project goals, best practices and ideal outcomes for a project.

Example:-Find Area and Perimeter of a Square

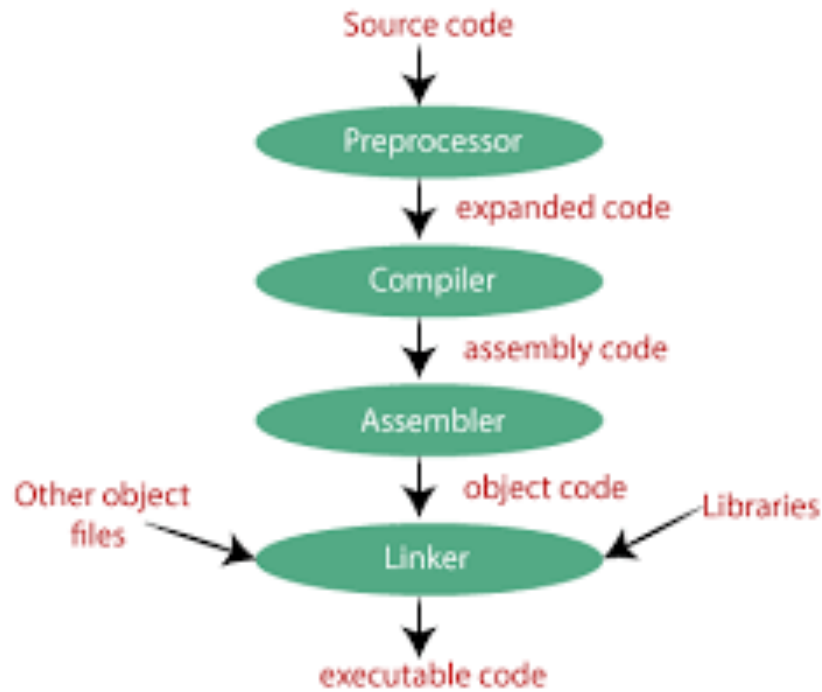
```
BEGIN
NUMBER len, area,perimeter
INPUT len
area = len*len
perimeter = len*4
OUTPUT area
OUTPUT perimeter
END
```

COMPILATION PROCESS IN C

The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.



Preprocessor

The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler

The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' and in UNIX, the extension is '.o'. If the name of the source file is '**hello.c**', then the name of the object file would be 'hello.obj'.

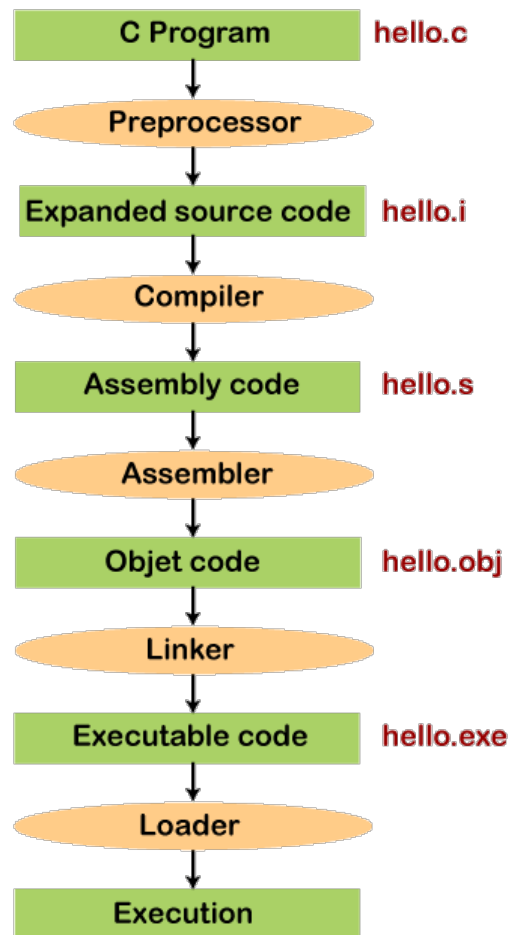
Linker

Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension. The main working of the linker is to combine the object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions. In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as 'a.out'.

Let's understand through an example.

hello.c

```
1. #include <stdio.h>
2. int main()
3. {
4.     printf("Hello ");
5.     return 0;
6. }
```



STRUCTURE OF C PROGRAM

A C-program may contain one or more sections as shown below



Documentation section:

The documentation section consists of a set of comment lines giving the name of the program, the author and other details and so on. In C comments are of two types

Single line comments and double line comments

Single line comments are represented by `//` and it will affected to a single line

Ex:- `// This is a single line comment`

The **Multi-line comment** in C starts with a forward slash and asterisk (`/*`) and ends with an asterisk and forward slash (`*/`). Any text between `/*` and `*/` is treated as a comment and is ignored by the compiler.

```
/*Comment starts  
continues  
continues  
Comment ends*/
```

Link section

The link section provides instructions to the compiler to link functions from the system library.

Example :- `#include<stdio.h>, #include<string.h>, #include<math.h>`

Definition section

The definition section defines all symbolic constants.

Example `#define pi 3.14`

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,

i) declaration part

The declaration part declares all the variables used in the executable part

ii) Executable part.

There is at least one statement in the executable part.

These two parts must appear between the opening and the closing braces. The program execution begins at the opening brace and ends at the closing brace. All statements in the declaration and executable parts end with a semicolon(;).

The 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 sections, except the main function section may be absent when they are not required.

TOKENS

Tokens in C are the element to be used in creating a program in C. We define the token as the smallest individual element in C.

Tokens in C language can be divided into the following categories:

- | | | |
|-------------------|---------------------|----------------------------|
| 1) Keywords in C | 2) Identifiers in C | 3) Constant in C |
| 4) Operators in C | 5) Strings in C | 6) Special Characters in C |

1) KEYWORDS

Keywords in C can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. Since keywords are the pre-defined words used by the compiler, so they cannot be used as the variable names.

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

2) IDENTIFIERS

C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore.

Rules for constructing C identifiers

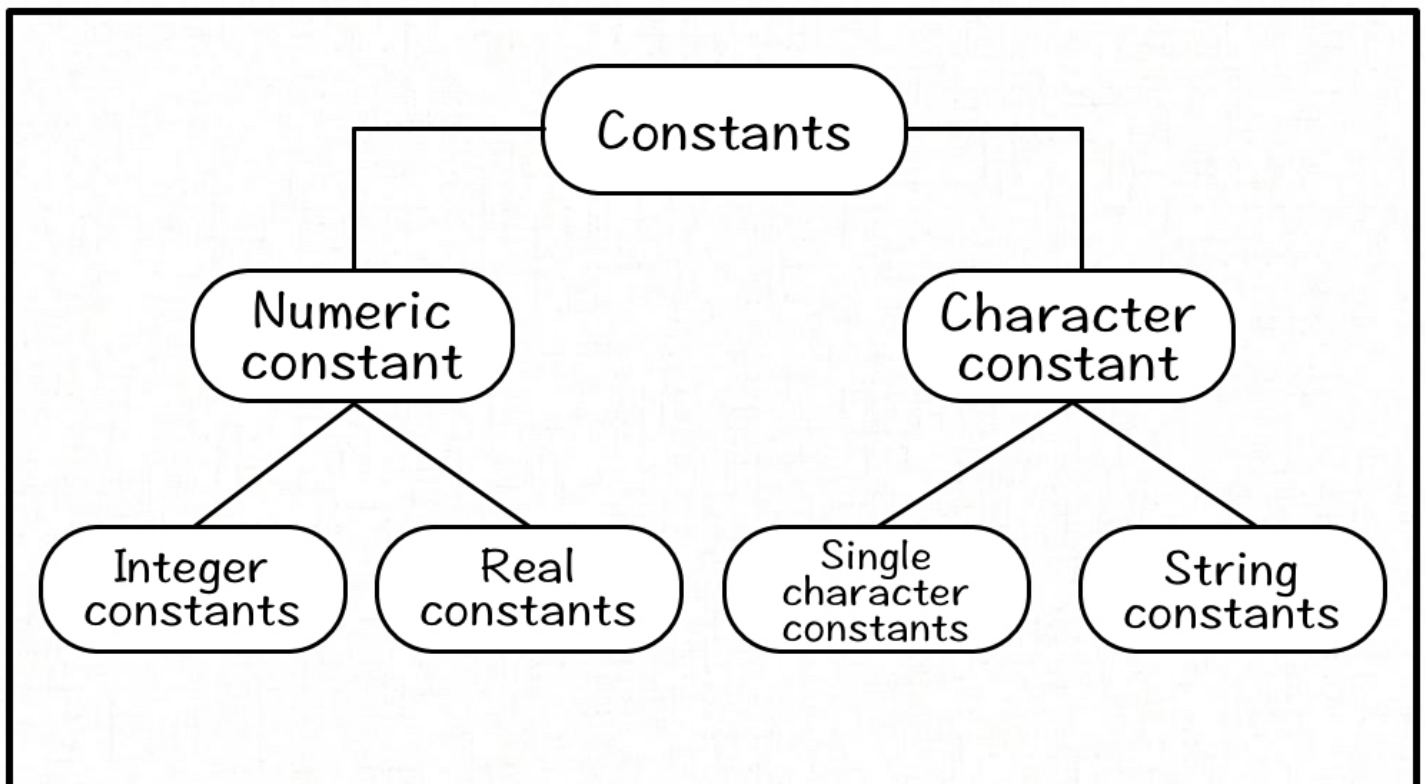
- o The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- o It should not begin with any numerical digit.
- o In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- o Commas or blank spaces cannot be specified within an identifier.
- o Keywords cannot be represented as an identifier.
- o The length of the identifiers should not be more than 31 characters.
- o Identifiers should be written in such a way that it is meaningful, short, and easy to read.

Example of valid identifiers

total, sum, average, _m_, sum_1, etc.

3) CONSTANTS

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



Numeric Constants :-

i) Integer Constants

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

- **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 – 321 0 654321 +78
- 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 0x or 0X 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:
0X2 0x9F 0Xbcd 0x

ii) 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.

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. e2 means multiply by 102. The general form is:

mantissa e exponent

The mantissa is either a real number expressed in decimal notation or an integer. The 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.

Character Constants:-

i) Single-Character Constants

A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Example of character constants are:

'5' 'X' ';' ' '

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

ii) 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 (e.g., 'X') is not equivalent to the single character string constant (e.g., "X").

iii) Backslash 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

Escape Sequences	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

4) OPERATORS

i. Arithmetic operators:-

C provides all the basic arithmetic operators. The operators +, -, *, and / all work the same way as they do in other languages. These can operate on any built-in data type allowed in C.

Arithmetic Operators		Let us assume X and Y are two variables
Operator	Expression	Description
+	$X + Y$	To perform addition
-	$X - Y$	To perform subtraction
*	$X * Y$	To perform multiplication
/	X / Y	To perform division
%	$X \% Y$	To perform modulus

Examples:-

$a - b = 10$

$a + b = 18$

$a * b = 56$

$a / b = 3$ (decimal part truncated)

$a \% b = 2$ (remainder of division)

ii. Relational operators

We often compare two quantities and depending on their relation, take certain decisions. For example, we may compare the age of two persons, or the price of two items, and so on. These comparisons can be done with the help of relational operators.

C supports six relational operators in all.

Relational Operators in C

Operator Symbol	Operator Name
==	Equal To
!=	Not Equal To
<	Less Than
>	Greater Than
<=	Less Than Equal To
>=	Greater Than Equal To

iii. Logical operators

The logical operators are used when we want to test more than one condition and make decisions. In addition to the relational operators, C has the following three logical operators.

Logical Operators in C

Operator	Meaning of Operator	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0
	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) (d>5)) equals to 1
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0

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.

iv. Assignment operators

Assignment operators are used to assign the result of an expression to a variable. The usual assignment operator “=”. In addition, C has a set of shorthand assignment operators of the form **v op= exp;**

Where v is a variable, exp is an expression and op is a C binary arithmetic operator. The operator op= is known as the shorthand assignment operator. The assignment statement `v op= exp;` is equivalent to `v = v op (exp)`. Some of the commonly used shorthand assignment operators are:

Shorthand Assignment operators	
Simple assignment operator	Shorthand operator
<code>a = a+1</code>	<code>a += 1</code>
<code>a = a-1</code>	<code>a -= 1</code>
<code>a = a * (m+n)</code>	<code>a *= m+n</code>
<code>a = a / (m+n)</code>	<code>a /= m+n</code>
<code>a = a %b</code>	<code>a %=b</code>

v. Increment and decrement operators

C allows two very useful operators not generally found in other languages. These are the increment and decrement operators: ++ and --

The operator ++ adds 1 to the operand, while -- subtracts 1. Both are unary operators and takes the following form:

++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;)

Consider the following:

m = 5;

y = ++m;

In this 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 the other hand, a postfix operator first assigns the value to the variable on left and then increments the operand.

vi. 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 exp1 is 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, consider the following statements:

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;

vii. 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. Table 5.5 lists the bitwise operators and their meanings.

Operators	Description	Example	Result
	Bitwise OR	4 2	6
&	Bitwise AND	1&3	1
^	Bitwise exclusive OR	4^3	7
<<	Bitwise Left Shift	2<<1	4
>>	Bitwise Right Shift	4>>1	2
~	Bitwise Not	~6	-7

viii. 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 Operator

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);

```
k = sizeof (235);
```

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.

5) STRINGS

Strings are nothing but an array of characters ended with a null character ('\0'). This null character indicates the end of the string. Strings are always enclosed in double quotes.

Example:-

```
char string[20] = {'w', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};  
char string[20] = "welcome";  
char string [] = "welcome";
```

6) SPECIAL SYMBOLS

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose. Some of these are listed below:

- **Brackets[]:** Opening and closing brackets are used as array element references. These indicate single and multidimensional subscripts.
- **Parentheses():** These special symbols are used to indicate function calls and function parameters.
- **Braces{}:** These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement.
- **Comma (,):** It is used to separate more than one statement like for separating parameters in function calls.
- **Colon(:):** It is an operator that essentially invokes something called an initialization list.
- **Semicolon(;):** It is known as a statement terminator. It indicates the end of one logical entity. That's why each individual statement must be ended with a semicolon.
- **Asterisk (*):** It is used to create a pointer variable and for the multiplication of variables.
- **Assignment operator(=):** It is used to assign values and for logical operation validation.
- **Pre-processor (#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.
- **Period (.):** Used to access members of a structure or union.
- **Tilde(~):** Used as a destructor to free some space from memory.