
UNIT 1:Process Of Programming

Historical development of C

Dennis Ritchie of Bell Laboratory USA developed “C” in 1972.

Dennis Richie and Ken Thompson used it to designing the UNIX operating system. “C” came from Ken Thompson's “B” language (**BCPL**-Basic Combined Programming Language).

Until 1978, C was confined to use within Bell Laboratories. In 1978, when Brian Kernighan and Ritchie published a description of the language, known as “k & rc” computer professionals got impressed with C’s many desirable features.

By mid 1980s, popularity of C increased. The important point is that C was created as a tool for working programmers, later turned as a useful programming language.

In the summer of 1983 a committee was established to create an ANSI (American National Standards Institute) standard that would define the C language. The standardization process took six years. The **ANSI C standard** was finally adopted in December **1989**, with the first copies becoming available in early 1990. The standard was also adopted by ISO (International Standards Organization), and the resulting standard was typically referred to as **ANSI/ISO Standard C in 1995**.

Features of C

- C is **reliable, simple and Easy to use** programming Language.
- C is **case sensitive** language.
Capital letters are different than small letters (Ex. ‘A’ is different than ‘a’)
- C is Middle level language, some authors refers C as High level language.
- C is **Platform Independent** language run on all operating system like windows, UNIX, LINUX etc.
- C is compact, fast, powerful language
- C is **portable language** - C program written on one computer can be run on another computer with proper steps of compilation and execution.
- C supports concept of modular programming or **structural programming**.
C provides the ability to split large programming tasks into separate modules.
C is also used for many commercial programs where speed of program execution is important.

Importance of C





The major important factors of C language are as follows:

- C is reliable, simple and Easy to use programming Language.
- C is the base language for some of other programming language like as C++, Java etc.
- To understand Object Oriented features of C++, Java etc. one must be familiar with C programming language.
- The operating systems like UNIX,LINUX, Windows are written in C language
- C is used in compiler designing. Ex.GNU compiler used in UNIX, LINUX.
- The Device driver programs usually written in C programming language.
- Some home appliances like Microwave Oven, Washing machines, Mobile phones, PDA's, Digital Camera's etc, are the embedded system devices their operating programs are usually written in C programming Language.
- Most of the Games, 3D animations are written in C.
- Simulation Software's like Matlab need basic knowledge of C programming language.
- C Programs runs faster as well as consumes a very less memory compare to other language programs.
- C is used in Some of PC applications.
- C used in Robotics.
- Some examples of the use of C might be –
 - Operating Systems
 - Language Compilers
 - Assemblers
 - Text Editors
 - Print Spoolers
 - Network Drivers
 - Modern Programs
 - Databases
 - Language Interpreters
 - Utilities

Editing, Compiling, Error Checking, executing, testing and Debugging

We generally write a computer program using a high-level language. A high-level language is one which is understandable by us humans. It contains words and phrases from the English (or other) language. To type your C program you need program called **Editor**. But a computer does not understand high-level language. It only understands program written in 0's and 1's in binary , called the machine code. A program written in high-level language is called a source code. We need to convert the source code into machine code and this is accomplished by **compilers** and **interpreters**. Hence, a compiler or an interpreter is a program that converts program written in high-level language into machine code understood by the computer.

- **Programming Languages uses following in built programs**

-  **Compiler** – It is a software program which converts the source program into machine code taking **complete program at a time**
-  **Interpreter** – It is a software program converts the source program into machine code taking **program line by line for conversion**
-  **Linker** – It is a software program used to link different modules of the program to generate executable file
-  **Debugger** – A debugging tool or debugger is a software (that's already available in IDE) which allows programmers to stop a program at any point and helps to detect and correct errors. When you have bugs in your program, rather than scratching your head to And the bug, you can use debugger to stop program at any point and the value of variables to detect the bug. Knowing how to use a debugger is an important skill that every programmer should learn.

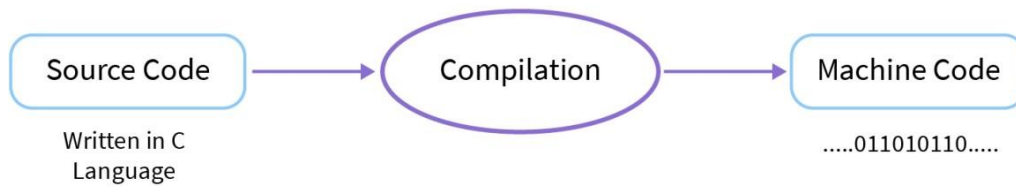
1)Editing

Open Turbo C from your Desktop or Programs menu. Select “File” from Menu bar and select option “New”

If there are any default lines of code present inside editor please remove all of them. The text editor should be blank. Now you may type in the following program in your Turbo C editor. This is a program to print “**Hello World**” on the first line of your output screen

2) Compiling

The compilation process in C is converting an understandable human code into a Machine understandable code and checking the syntax and semantics of the code to determine any syntax errors or warnings present in our C program. Suppose we want to execute our C Program written in an IDE (Integrated Development Environment). In that case, it has to go through several phases of compilation (translation) to become an executable file that a machine can understand.



Compilation process in C involves four steps:

- I. Preprocessing**
- II. Compiling**
- III. Assembling**
- IV. Linking**

I.Preprocessing:

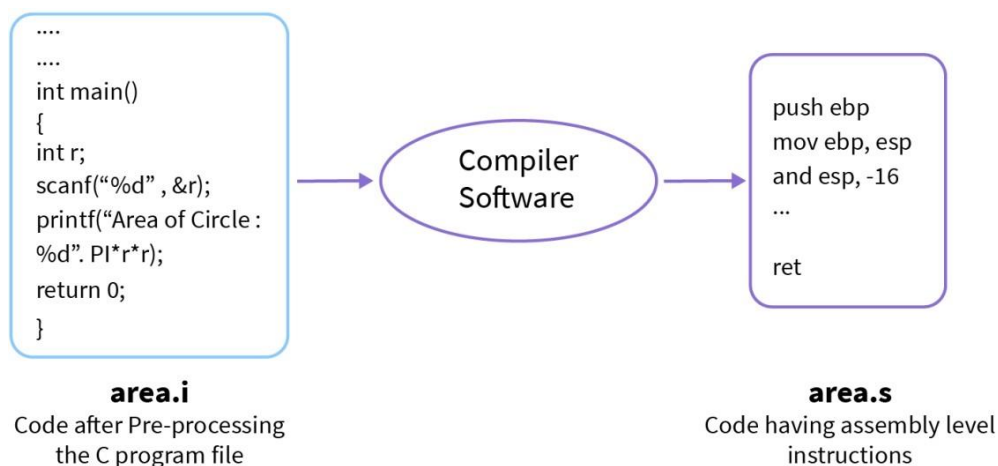
Pre-processing is the first step in the compilation process in C performed using the pre-processor tool (A pre-written program invoked by the system during the compilation). All the statements starting with the # symbol in a C program are processed by the pre-processor, and it converts our program file into an intermediate file with no # statements. Under following pre-processing tasks are performed like Comments Removal, File inclusion

II. Compiling

Compiling phase in C uses an inbuilt compiler software to convert the intermediate (.i) file into an Assembly file (.s) having assembly level instructions (low-level code). To boost the performance of the program C compiler translates the intermediate file to make an assembly file.

Assembly code is a simple English-type language used to write low-level instructions (in micro-controller programs, we use assembly language). The whole program code is parsed (syntax analysis) by the compiler software in one go, and it tells us about any syntax errors or warnings present in the source code through the terminal window.

The below image shows an example of how the compiling phase works.



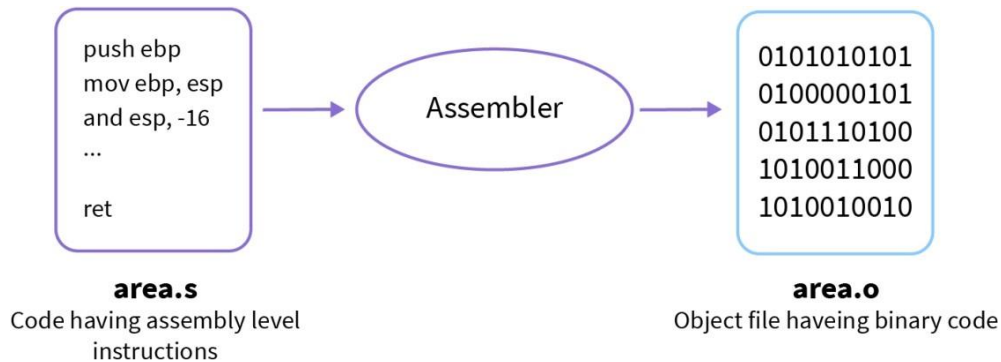
III. Assembling

Assembly level code (.s file) is converted into a machine-understandable code (in binary/hexadecimal form) using an assembler. Assembler is a pre-written program that translates assembly code into machine code. It takes basic instructions from an assembly code file and converts them into binary/hexadecimal code specific

to the machine type known as the object code.

The file generated has the same name as the assembly file and is known as an object file with an extension of .obj in DOS and .o in UNIX OS.

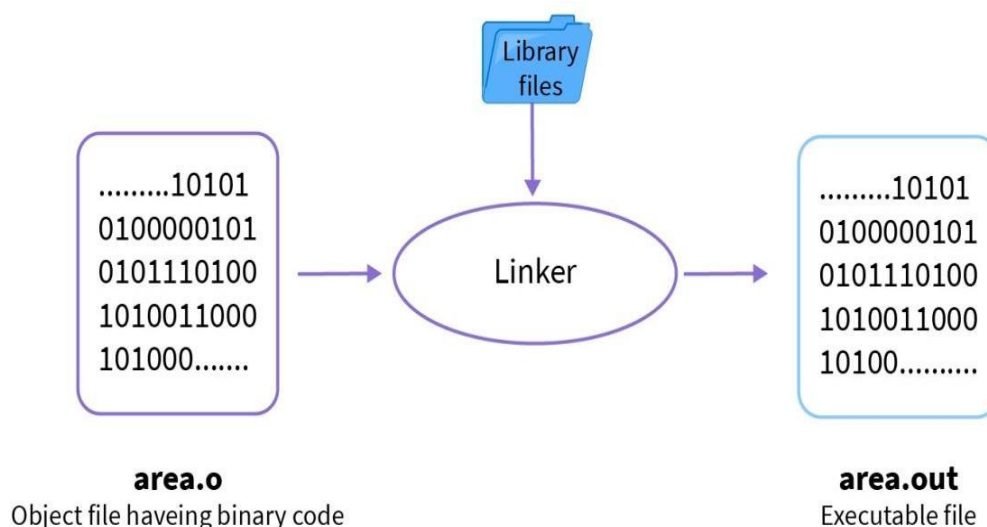
The below image shows an example of how the assembly phase works. An assembly file area.s is translated to an object file area.o having the same name but a different extension.



IV. Linking

Linking is a process of including the library files into our program. Library Files are some predefined files that contain the definition of the functions in the machine language and these files have an extension of .lib. Some unknown statements are written in the object (.o/.obj) file that our operating system can't understand. You can understand this as a book having some words that you don't know, and you will use a dictionary to find the meaning of those words. Similarly, we use Library Files to give meaning to some unknown statements from our object file. The linking process generates an executable file with an extension of .exe in DOS and .out in UNIX OS.

The below image shows an example of how the linking phase works, and we have an object file having machine-level code, it is passed through the linker which links the library files with the object file to generate an executable file.



The difference between an interpreter and a compiler is given below:

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

3) Error Checking

The error are corrected using the editor and the programme is compiled again till we get error free programme

Types of Error

- **Syntax Error or Compiler Time Error** - The errors occurs due to syntactical mistakes while typing the program such as semicolon (;) missing, wrong syntax of statements loops, variable names mismatch, braces mismatch or missing { } , function parameter mismatch etc.
- **Run Time Error or Logical Errors** - Errors occurred due to wrong programming logic such as divide by zero, improper looping condition, prototype missing , directory linking errors, errors caused due to corrupted header and other important files etc. Ex. The following statement generates a run-time error if Count has the value zero, since division by zero is not mathematically defined.

Average = Total/Count;

Program Execution

Assuming that you are using a **Turbo C** or Turbo C++ compiler here are the **steps** that you need to follow to **compile and execute** your first C program...

- Start the compiler at C>prompt.
The compiler (TC.EXE is usually present in C:\TC\BIN\directory).
- Select New from the File menu.
- Type the program.

- Save the program using F2 under a proper name (say Program1.c).
- Use to Alt + F9 compile and Ctrl + F9 execute the program.
- Use Alt + F5 to view the output.

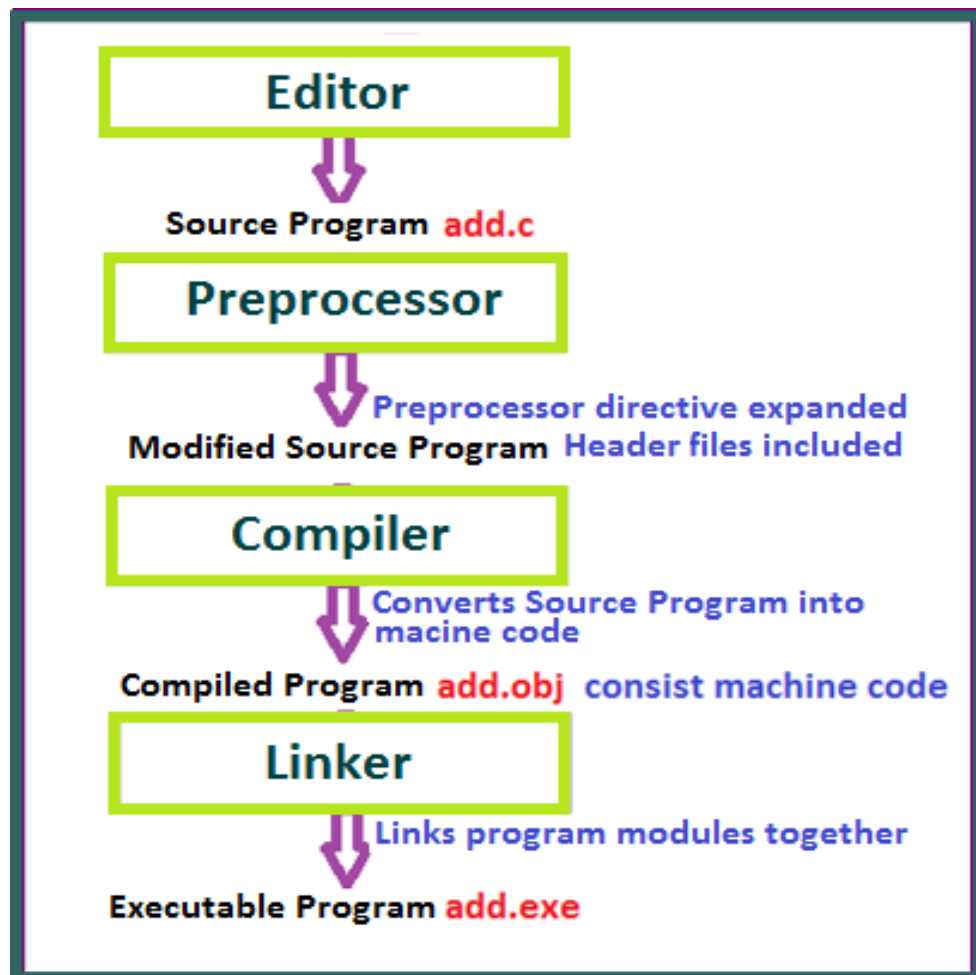


Figure. Internal operations of program execution

Program Testing

- Testing is the process of finding errors ("bugs") in a program.
- Testing can increase your confidence that a program is error-free.
- Testing can find the presence of errors, but, in general, cannot prove the absence of errors.
- Testing small programs is *much* easier than testing large programs.
- When testing code with branches, provide inputs that test all code sections of the if statement.
- You should test your program with **expected** and **unexpected** (invalid) inputs:
 - You should know how your program will perform with good data as well as bad data.
 - Unexpected values may be negative, extremely large, or extremely small values.
- You should test your program with **boundary conditions** values, i.e., values at and around a value for which the behavior of a program should change.
- When testing loops, provide inputs that cause the loop to repeat zero, one, and many times.

Program Debugging

- **Debugging** is the process of locating and correcting errors in a program.
- Debugging is problem-solving and often can be very challenging.
- Thinking carefully about your program is often the best first step when debugging.
- You can help minimize your time spent debugging by:
 - Starting with a good program design,
 - Coding carefully, and
 - Testing your program as you write.
- The development of good debugging skills comes from experience. Therefore, make sure you learn from your mistakes.
- A **symbolic debugger** is a tool for debugging code that allows you to examine your program's behavior as it executes.

Basic Structure of C Program

The following figure shows basic structure of c program

```
/* Program to illustrate general structure of C program */
/*Global variables declaration; */
#include<stdio.h>
#include<conio.h>
Function prototype declaration;
void main()
{
    /* body of program – it includes declaration of variables, constants etc.
    statements, expressions, loops, function declaration */
    getch();
}
Function definition (parameter list) /* Function is Optional */
{
    /* Body of Function */
}
```

Creating a simple C program

The C program written for above algorithm 4.1 and Flowchart figure4.1 is as follows

```
/* C Program for addition of two numbers */
/*ADD.C*/
#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 10, b = 20, c;
    c=a+b;
    clrscr();
    printf("Addition of a & b is = %d",c);
    getch();
}
```

Program1: Program for addition two numbers

Comments

- In the above program the first line in the program starts with `/*` and ends with `*/`. **Anything written between `/*` and `*/` is called a comment.**
- In the C Language comments used by programmer to read and understand a program.
- It is not a statement of the language.
- The compiler ignores comments. It is a good practice to include comments in a program which will help in understanding a program.
- Comment can be of single line or multiple lines.

Preprocessor Directive

include <stdio.h>

- This is called a preprocessor directive. It is written at the beginning of the program.
- The file **stdio.h** is a one header file (indicated by extension **.h**) contains the **standard input/output** functions such as `printf()`, `scanf()` etc.
It commands to compiler that the contents of the file `stdio.h` should be included in the program.
- The file **conio.h** is a another header file contains the console **input/output** functions such as `getch()` – get character, `clrscr()`-clear screen etc.
- All preprocessor directives begin with **Hash sign #** which must be entered in the first column.
- The `# include` line must not end with a semicolon.
- Only one preprocessor directive can appear in one line.

Header files

- **Header files contain definitions of functions and variables** which can be used into any C program by using the pre-processor **#include** statement.
- Standard header files are provided with the compiler
- Header files covers a range of areas: string handling, mathematics, data conversion, printing and reading of variables, etc.
- To use any of the standard functions, the appropriate header file should be included.
- This is done at the beginning of the C source file.
- For example, to use the function **printf()**, **scan()** in a program, the line
#include <stdio.h>
should be at the beginning of the source file, because the declaration for **printf()** is found in the file **stdio.h**.
- All header files have the extension **.h**
- Other header files examples
#include <string.h> - For String operations
#include <math.h> - For Mathematical operations
#include <conio.h> - For console input output operations like getch(), clrscr()
- You can also use " " double quotes instead of angle brackets <>
- Angle brackets <> informs the compiler to search the compiler's include directories for the specified file. The use of the double quotes " " around the filename informs the compiler to start the search in the current directory for the specified file.

main () Function

- C program is made up of many functions.
- The function main () is required in all C programs.
- We will use main () at the beginning of all programs to inform compiler to indicate start of the program.
- No semicolon should be given after the main()
- Here the **main()** **does not return any value** so, we use **void** keyword before main()
If you not written void before main () it will show an error "function should return a value" at the time of execution

Braces pair { }

- Braces pair { } encloses the body by function main ().
- Braces pair { } encloses the body of other functions.
- Braces pair { } encloses the body of the loops, statements such as if-else, switch etc.
- **For every opening brace { there must matching closing brace }**

Declaration of variables and constants

int a = 10, b = 20, c;

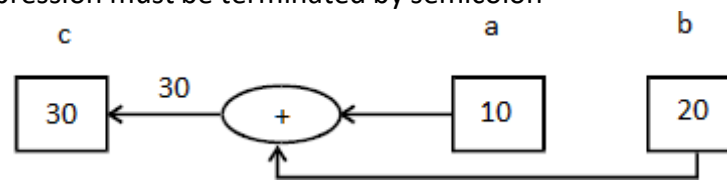
- This line declares the two constant values a=10 and b=20, as well as variable c of the type integer (we used keyword **int**).

- $a=10$ and $b=20$ uses **assignment operator =**.
Here value 10 (at the right side of =) is assigned to variable a (at the left side of =) similar case for the $b=20$.
- The declaration line must separate all constant, variables by comma
- The declaration line must be ended by semicolon.

Expression

$c=a+b;$

- The value of expression $a+b$ (at the right side of =) is assigned to variable c (at the left side of =)
- Every expression must be terminated by semicolon



$clrscr ();$

- Clear screen – this function will clear the console screen
- It helps to show only current program execution output any previous contents on the screen are cleared.
- Requires `#include <conio.h>` declared.

$printf ("Addition of a \& b is = \%d",c);$

- **printf()** is a formatted output function used to display output on console screen
- **"Addition of a & b is = %d"** – This is message part that we want to display on output screen
- **%d** – is format specifier used for integer values.
 - In this case the value of the variable c will be placed on position of %d at final output
 - Since we are displaying the value of c and c is having its type as **int** at declaration line hence we are using **%d** in the printf()
 - Other format specifier are as follows:
- You can print or display any message as like above by enclosing it into the pair of double quotes **"your message"**.
- All alphabets (lowercase, capital), digits, special symbols can be part of your message only thing is that it must enclosed in double quotes **""**.

Table Format Specifier

Format Specifier	Value Type
%d	Integer value
%f	Float value
%c	Character value
%s	String value
%ld	Long integer value
%lf	Double value
%Lf	Long double values
%u	Unsigned value

getch(); - get character

- If you not written this in your program the console output screen will not steadily visible to you.
 - The prototype **#include<conio.h>** must be defined in your program otherwise getch(); will prompt you an error at time of execution.
- ❖ Each line in the program is a statement or expression that must be terminated by a **semicolon;**
- ❖ The statement itself can be written anywhere in a line.
- ❖ More than one statement can be on a line as a semicolon separates them.
It is a good practice to write one statement per line.
- ❖ Your program should be properly indented, neatly written

IDE, Eclipse for C Program development

- An **Integrated Development Environment (IDE)** is a software application that provides comprehensive facilities to computer programmers for software development.
- An IDE normally consists of a source code editor, build automation tools and a debugger.
- This typically provides many features for authoring, modifying, compiling, deploying and debugging software.
- Most modern IDEs have intelligent code completion.
- IDEs present a single program in which all development is done.
- Ex. Some IDEs, such as Eclipse and NetBeans etc.
- Most modern IDEs are graphical, text-based IDEs
- Some IDEs are dedicated to a specific programming language, allowing a feature set that most closely matches the programming paradigms of the language. However, there are many multiple-language IDEs.

Eclipse

- **Eclipse** is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE.
- Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including Ada, ABAP, C, C++, C# etc.
- **Eclipse CDT is used for C/C++.**
The Eclipse CDT is an Eclipse plug-in that transforms Eclipse into a powerful C/C++ IDE.
- Eclipse software development kit (SDK) is **free and open-source software**, released under the terms of the Eclipse Public License.
- Different versions of Eclipse have been given different science-related names

Version name	Date	Platform version
N/A	21 June 2004	3.0 ^[22]
N/A	28 June 2005	3.1
Callisto	30 June 2006	3.2
Europa	29 June 2007	3.3
Ganymede	25 June 2008	3.4
Galileo	24 June 2009	3.5
Helios	23 June 2010	3.6
Indigo	22 June 2011	3.7
Juno	27 June 2012	3.8 and 4.2 ^[30] [Notes 1]
Kepler	26 June 2013	4.3
Luna	25 June 2014	4.4
Mars	24 June 2015	4.5
Neon	22 June 2016	4.6
Oxygen	28 June 2017	4.7
Photon	June 2018 (planned)	4.8

Figure. Versions of Eclipse

Eclipse Installation

For installation you need following:

- **Eclipse**
We're using the CDT C/C++ Development Toolkit (CDT), which is a plug-in to Eclipse, so of course you need Eclipse.
- **Java Runtime Environment (JRE)**
We're building a C++ application, but we're using Eclipse. Eclipse is a Java application itself, so it Environment (JRE). Which requires a JRE of V1.4 or higher.
- **GNU C/C++ Development tools – (Compiler)**
The CDT uses the standard GNU C/C++ tools for compiling your code, building your project, and applications. These tools are GNU Compiler Collection (GCC) for C/C++.

Basic Terms used in eclipse software:

- **Workspaces:**
Workspaces are directories that the Eclipse IDE stores information and files associated with one or more projects, typically grouping related projects together.
- **Projects:**
Projects contain and manage programs that are to be combined together to create a single executable program (or a library). Typically one creates a new project for each programming task rather than re-using existing projects.
- **Source file:**
This is (.c) file where you edit your program.
- **Build**
This command compiles your program.
- **Run**
This command executes your program.
- **Debug**
This command used for debugging your program.

Algorithm

Before explaining the algorithm the some basic things need to be explained they are as follows:

- **Data**- Collection of records, information
- **Instruction** – Command given to the program
- **Program**- Set of Instruction used to perform typical task.
- **Programming Language**- The language used to write, edit, compile, execute (run) the program. Ex. C, C++, Java etc.

An Algorithm is the systematic generalize step by step procedure to write the program.

- ❖ Typically, an algorithm consist of general operation steps that steps we can use for writing the programs in any of the programming languages.
- ❖ The generalize steps of an algorithm can be converted to respective program instruction according to programming language's syntax and semantics.
 - Every programming language uses its own notation of program writing.
- ❖ Every good programmer will first think of an algorithm regarding which operation he has to perform in the program before writing actual program.
 - An algorithm helps programmer to understand the operation to be performed in the program.
 - Decision making will be easier from the algorithm
 - An algorithm gives logic of program

Ex. Generalize Operation:

Suppose we want to perform **simple addition** of two numbers **a = 10** and **b = 20** we will get result **30**. Finally we will display result **c = 30**




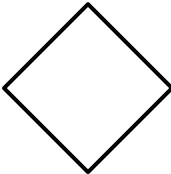

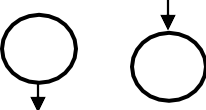
Algorithm: (Algorithm 1)

- | | |
|---------------|------------------------------|
| Step1: | Start |
| Step2: | Declare a,b,c |
| Step3: | Read values a=10, b=20 |
| Step4: | Perform addition $c = a + b$ |
| Step5: | Display result by using c. |
| Step6: | Stop |

Flowchart

Flowchart is the graphical representation of an algorithm or programming steps.

The following notations are used for drawing flowchart:

- | | | |
|--------------------|---|---------------------------------|
| 1. Start / Stop |  | for start / Stop of program |
| 2. Calculation Box |  | For showing calculation |
| 3. Input / Output |  | For showing input / output data |
| 4. Decision Box |  | For showing decision |
| 5. Arrow |  | used to show the links |
| 6. Connectors |  | used in flowchart connection |

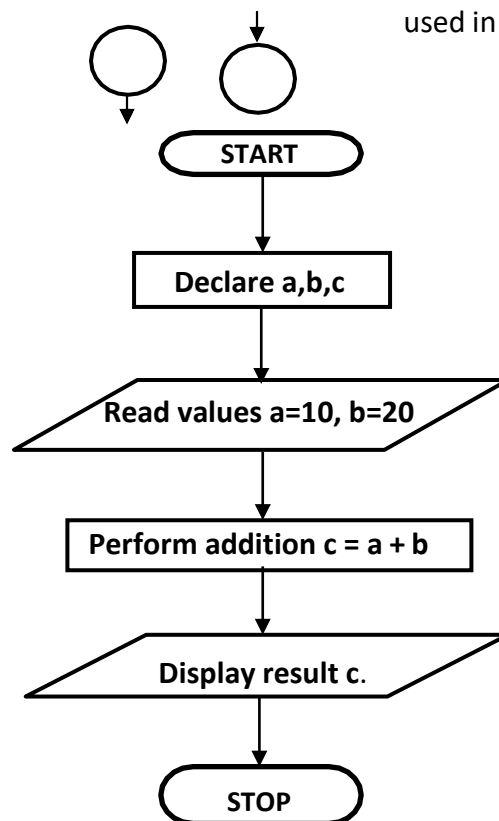


Figure. Flowchart for algorithm 1

Simple steps to make a flowchart:

1. Determine the purpose or function of the flowchart
2. Add steps and connect them with arrows
3. Add decisions or split paths
4. Show any loops back to previous steps
5. Share your flowchart

A good flowchart should communicate a process clearly and effectively. When starting out, it's a good idea to focus on a couple of things.

Here are some of the ways flowcharts are used today.

- Project planning
- Program or system design
- Process documentation
- Audit a process for inefficiencies or malfunctions
- Map computer algorithms
- Documenting workflow