Introduction to C programming

* Computer program Set of instructions given to the computer

* Programming Language can be mainly divided Into

(1) Low-level programming Language.

(a) Machine Language (binary Language)

(b) Assembly Language

(2) High-level programming Language.

Low - Level Language

Machine Language (binary Language)

- Machine can understand only 1's and 0's
- Machine dependent
- Computer can directly understand its own machine Language
- Tedius and error-prone for programmers (but the computer can easily understand the errors)

Assembly Language.

- English like abbreviations Called mnemonics formed the basis
- clear to the human but computers can't understand
- Need to translate to machine language using translation programs called assember.

High-level Language

• Instructions look almost like English and mathematical notations.

• Substantial tasks can be accomplished from a single statement.

• easy for humans to understand but the computer can't understand.

• Translator programs convert high-level language into machine language.

• c, c++, python, and Java

$$\begin{tikzcd}\boxed{\text{Source code}} \arrow[r] & \boxed{\text{Translator}} \arrow[r] & \boxed{\text{Object code (machine language code)}}\end{tikzcd}$$

Translator

- Assemblers (convert assembly language programs to machine language)

- Compilers (convert high-level language programs to machine language)

- Interpreters (execute high-level language programs line by line

History of C language

• C language was evolved from two previous languages, BCPL and B by Dennis Ritchie at Bell Laboratories in 1972.

• C initially became widely known as the developing language of the UNIX operating system.

• C99 and C11 are revised standards for C programming language that refines and expands the capabilities of C.

Simple C program

Source code

```
/* First program in C
This program displays a message */
#include < stdio.h >
// function main begins program
execution
int main(void)
{
printf ("welcome to C!");
return 0;
} // end of function main
```

Output

```
welcome to C!
```

Simple C program Description

```
/* First program in C This program displays a message */
```

```
// function main begins program execution
```

These are comments. Comments are used to document programs and it improves the program readability. C compiler ignores comments. Line comments begin with // and continue for the rest of the line. /* … */ multi-line comments can also be used. Everything from /* to */ is a comment.

```
#include < stdio.h >
```

This is a directive to C preprocessor. Lines begin with # are processed by the preprocessor before compiling the program. Here, preprocessor includes the content of stdio.h in the program. stdio.h is a header file which contains information about standard input/output library function calls such as printf.

```
int main(void)
```

Every C program has this main function and it begins executing at main. “int” to left of main indicates that the function returns an integer. “void” in parentheses indicates that main does not receive any information.

```
{
```

Left brace , { , begins the body of every function. Functions ends with a corresponding right brace. The portion of the program between the braces is called a block.

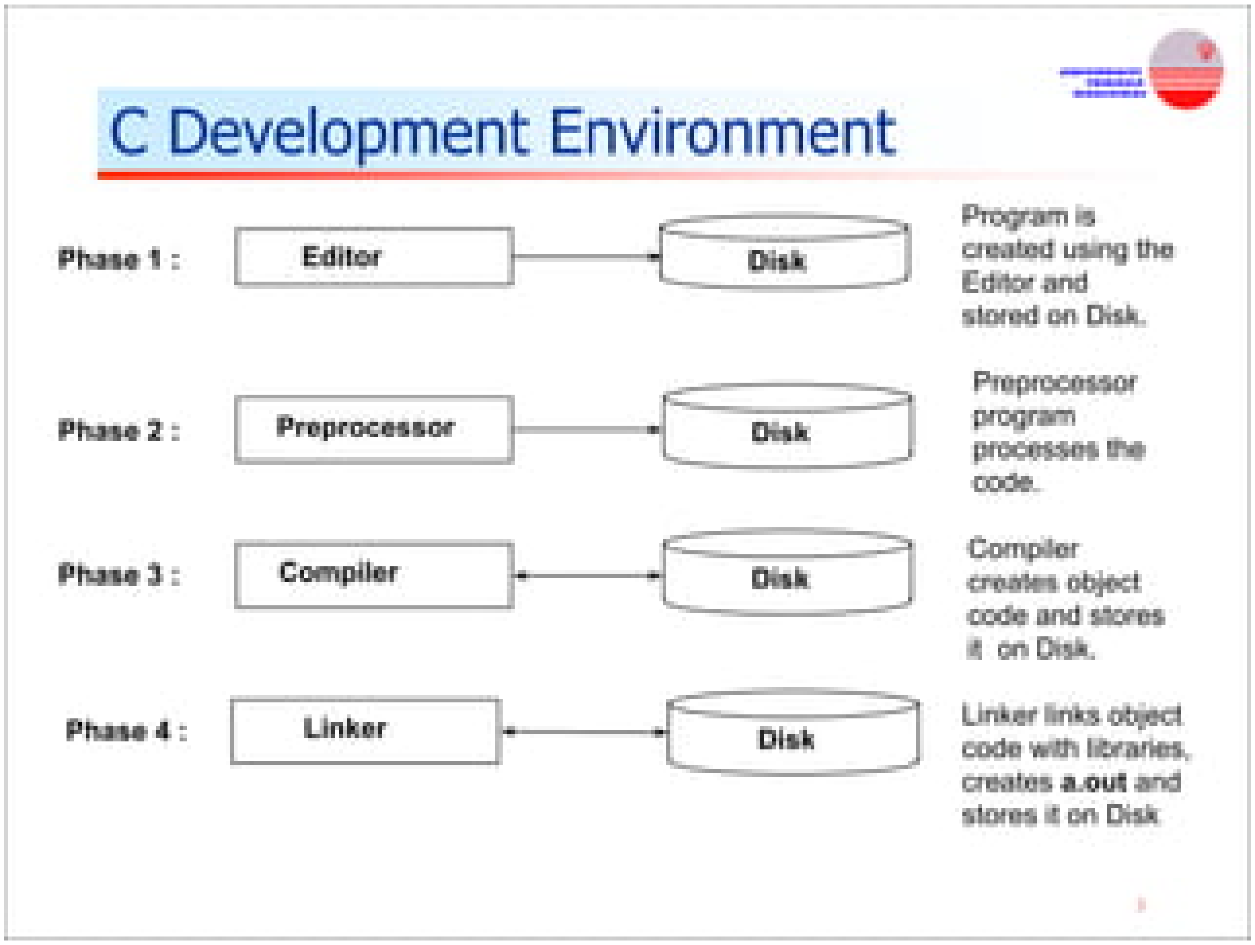
```
printf ( “welcome to C! ” );
```

Entire line is called a statement. It instructs the computer to perform an action. A statement must end with a semicolon. This statement prints a string of characters marked by the quotation marks on the screen.

```
return 0;
```

Included at the end of every main function. It is used to exit the main function and the value 0 indicates a successful termination. Blank Lines and White Space Blank lines, space characters and tab characters are known as white space. White-space characters are normally ignored by the compiler.

C Program Development Environment



C Development Environment

Program is created using the Editor and stored on Disk.

Preprocessor program processes the code.

Compiler creates object code and stores it on Disk.

Linker links object code with libraries, creates **a.out** and stores it on Disk

Printf() function

printf(“welcome to C!\n”);

can be written as,

puts(“welcome to C!”);

puts function adds newline automatically.

printf(“welcome “);

can be written as,

printf (“%s”, “welcome ”);