Brief History of C Language

Section 1

Machine Languages, Assembly Languages and High-Level Languages

- Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time.
- Example: 00010101
  11010001
  01001100
- Each of the lines above corresponds to a specific task to be done by the processor
- Such languages are cumbersome for humans.
- Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations.
- These abbreviations formed the basis of assembly languages.

Machine Languages, Assembly Languages and High-Level Languages

- Enables machine code to be represented in words and numbers
- Translator programs called assemblers were developed to convert early assembly-language programs to machine language at computer speeds.
- Example of a program in assembly language:

```
LOAD A, 9999
LOAD B, 8282
SUB B, A
MOV C, A
```

- Although such code is clearer to humans, it's incomprehensible to computers until translated to machine language.
- Processor and Architecture dependent – **not portable**

Machine Languages, Assembly Languages and High-Level Languages

- Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks.
- To speed the programming process, high-level languages were developed in which single statements could be written to accomplish substantial tasks.
- Translator programs called compilers convert high-level language programs into machine language.
- Processor **independent** - the same code can be run on different processors.

Machine Languages, Assembly Languages and High-Level Languages

- High-level languages allow programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations.
- Examples: Basic, Fortran, Pascal, C, C++ and Java
- Interpreter programs were developed to execute high-level language programs directly (without the delay of compilation), although slower than compiled programs run.

History of C

- BCPL ,1967, Martin Richards
  - as a language for writing operating-systems software and compiler
- B, 1969, Ken Thomson
  - based on BCPL
- C, 1972, Dennis Ritchie
  - based on BCPL and B
  - C initially became widely known as the development language of the UNIX operating system.

C Standard Library

- C programs consist of modules or pieces called functions.
- You can program all the functions you need to form a C program, but most C programmers take advantage of a rich collection of existing functions called the C Standard Library.

C Standard Library

- Avoid reinventing the wheel.
- Instead, use existing pieces—this is called software reusability, and it's a key to the field of object-oriented programming, as you'll see when you study C++.
- When programming in C you'll typically use the following building blocks:
  - C Standard Library functions
  - Functions you create yourself
  - Functions other people have created and made available to you

C++

- C++ was developed by Bjarne Stroustrup at Bell Lab.
  - It has its roots in C, providing a number of features that "spruce up" the C language.
  - provides capabilities for object-oriented programming.
  - Objects are essentially reusable software components that model items in the real world.

Typical C Program Development Environment

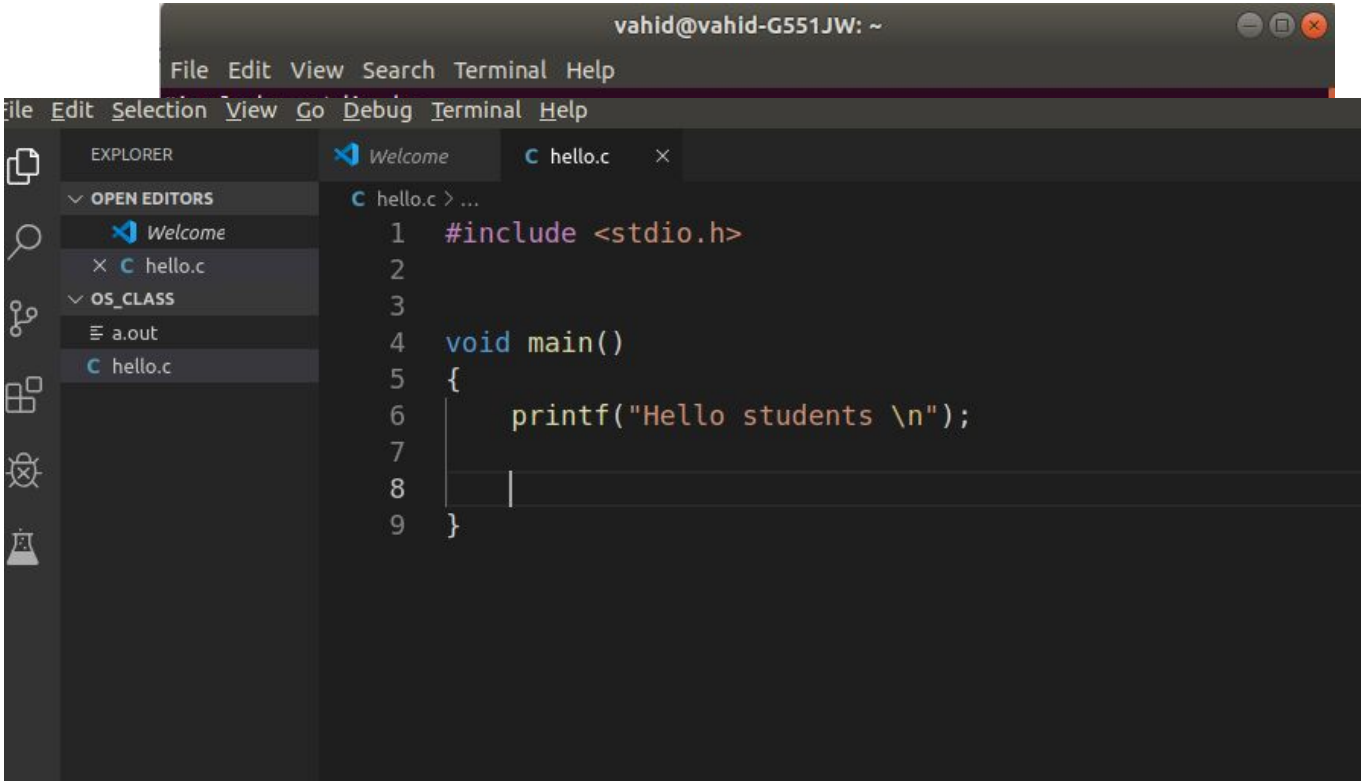
- C systems generally consist of several parts:
  - a program development environment
  - the language
  - the C Standard Library.
- C programs typically go through six phases to be executed:
  - edit, preprocess, compile, link, load and execute.

Typical C Program Development Environment

- Phase 1 consists of editing a file. This is accomplished with an editor program.
- You type a C program with the editor, make corrections if necessary, then store the program on a secondary storage device such as a hard disk.
- C program file names should end with the `.c` extension.
- `gedit sampleProgram.c`

Typical C Program Development Environment

- Editing a file with an editor program



Typical C Program Development Environment

- In Phase 2, the you give the command to compile the program.
- The compiler translates the C program into machine language-code (also referred to as object code).
- `gcc  sampleProgram.c  -o  sampleProgram`

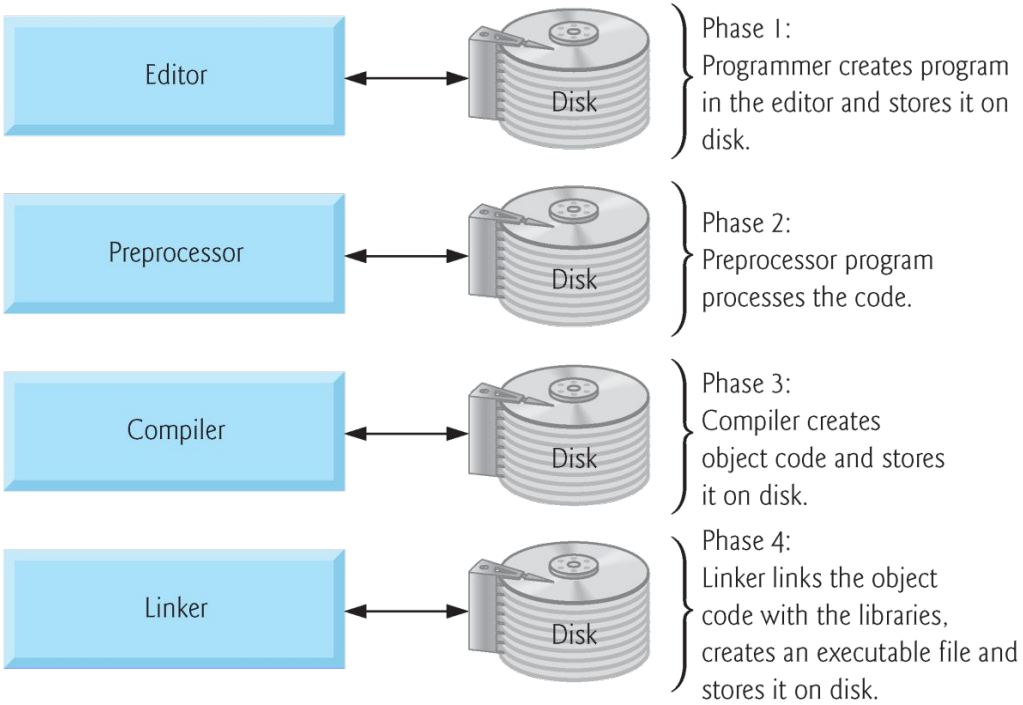


Fig. 1.1 | Typical C development environment. (Part 1 of 2.)

Typical C Program Development Environment

- The next phase is called loading.
- Before a program can be executed, the program must first be placed in memory.
  - This is done by the loader, which takes the executable image from disk and transfers it to memory.
  - Additional components from shared libraries that support the program are also loaded.
- Finally, the computer, under the control of its CPU, executes the program one instruction at a time.

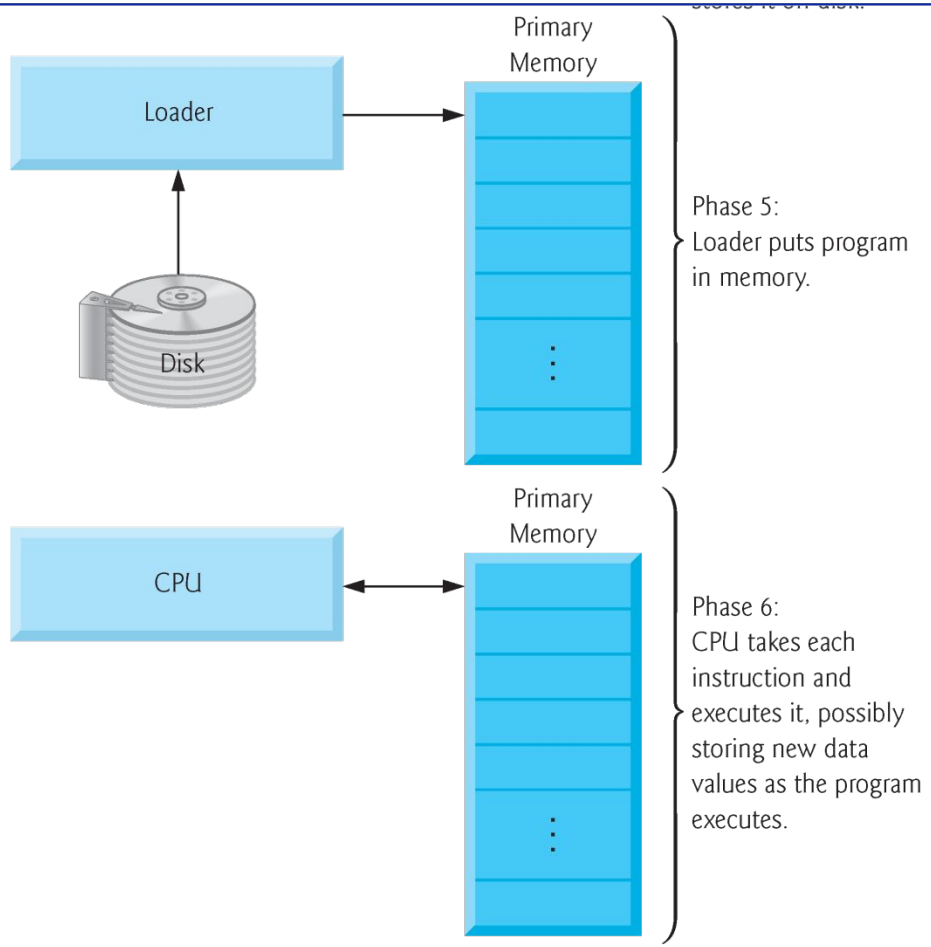


Fig. 1.1 | Typical C development environment. (Part 2 of 2.)

Typical C Program Development Environment

- Programs do not always work on the first try.
- Each of the preceding phases can fail because of various errors that we'll discuss.
- For example, an executing program might attempt to divide by zero.
- This would cause the computer to display an error message.