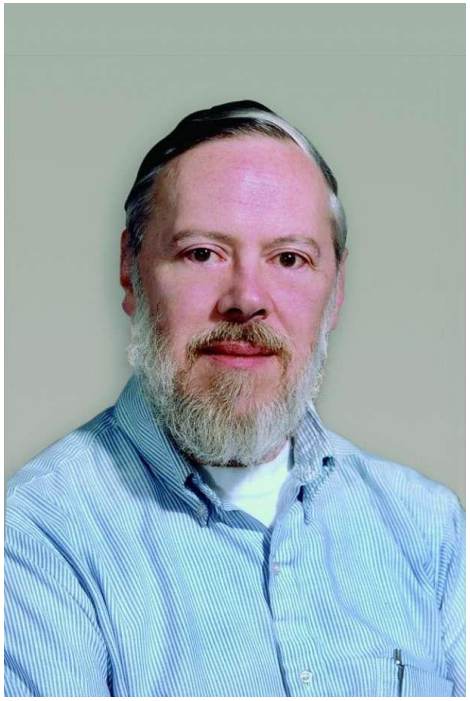
CS100
Introduction to Programming

Lecture 1. C Program Structure

A Brief History of C

- UNIX operating system
  - In 1969, a small group of AT&T Bell Labs led by Ken Thompson and Dennis Ritchie began to develop UNIX
  - In 1973, UNIX kernel was rewritten in C
- Creation of C language
  - From 1969 to 1973, Dennis Ritchie developed C in Bell Labs
  - In 1978, Kernighan and Ritchie published the K&R book "The C Programming Language"
- ANSI C Standard
  - In 1980's the *American National Standards Institute* (ANSI) gave a definition of C and *C standard library*



Dennis M. Ritchie (1941 – 2011)
- The inventor of C language
- Co-inventor of UNIX
- ACM Turing Award (1983) with Ken Thompson for UNIX

Why Learn C?

- Advantages:
  - Powerful, flexible, efficient, portable
  - Small and simple
  - C is closely related with UNIX / Linux
  - Influence on other languages: C++, C#, Java
- Disadvantages:
  - Using **pointers** might be confusing and cause errors
  - Requires attention to low-level details

Plan for Learning C

- Week 1
  - C program structure
  - Data types, operators, expressions
- Week 2
  - Simple input/output (I/O)
  - Control structures
- Week 3 (National Day holidays)
- Week 4
  - Functions
  - Pointers
- Week 5
  - Arrays
  - Character strings
- Week 6
  - Structures
  - Recursion

Learning Objectives (of Lecture 1)

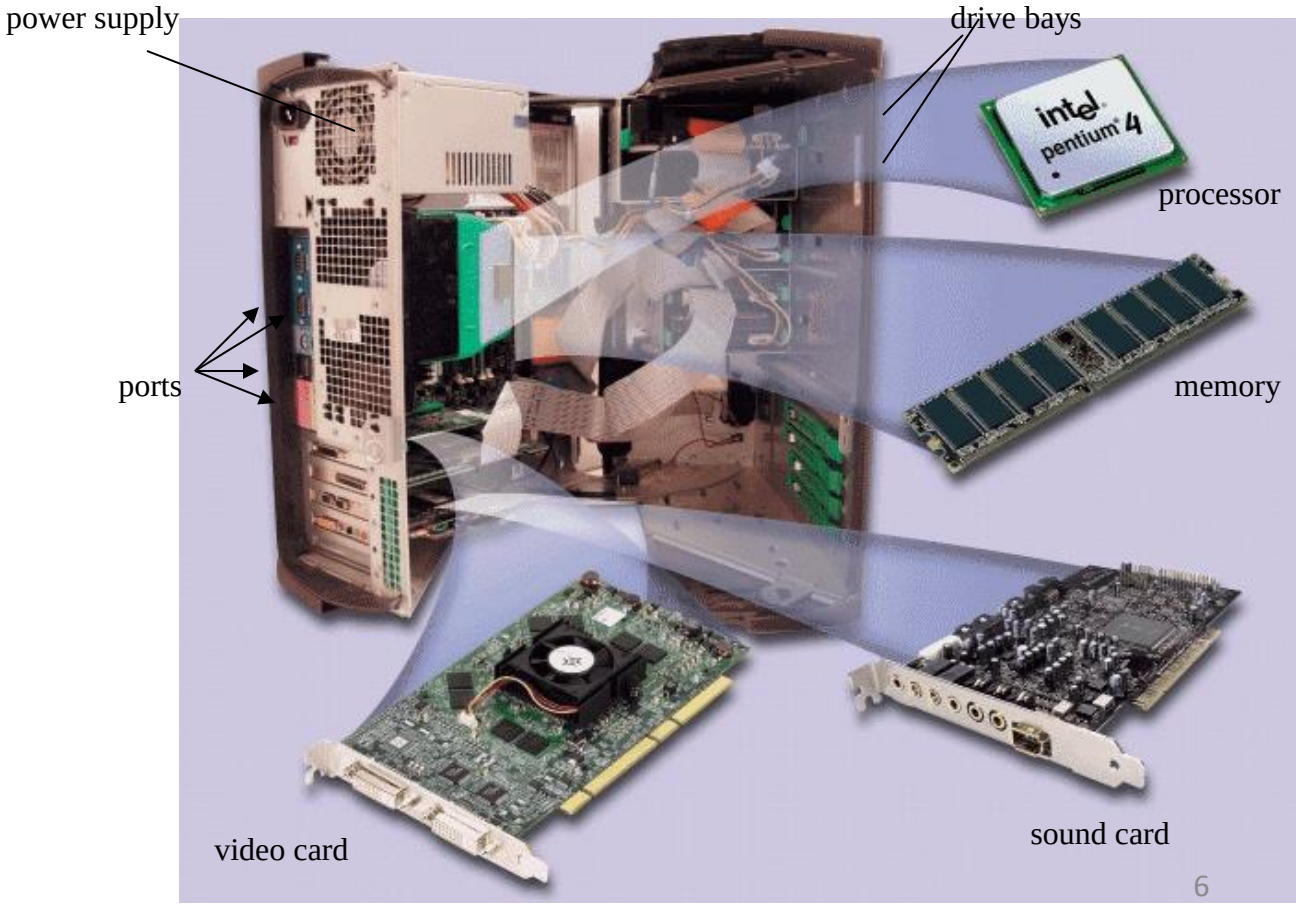
- At the end of this lecture, you will be able to:
  - Know how computer systems manage information
  - Understand how a C program works
  - Recognize C program structures
  - Develop a simple C program

Introduction to Computer Systems

- A **computer system** consists of **hardware** and **systems software** that work together to run **application software**.

Systems software:
- Operating system
- Compiler
- Linker
- Debugger

Application software:
- Word processor
- Web browser
- Media player



The hello Program

```
#include <stdio.h>

int main()
{
    printf("hello, world!\n");
}
```

The above program is saved as a text file named "hello.c"

The text characters are represented by numbers (ASCII code) as:

#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	<sp>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

From book of Bryant and O'Hallaron, 2010, Fig. 1.1, page 2

Information Encoding

- **Bit**: 2 different possibilities, 0 or 1
- **Byte** (8 bits): $2^8 = 256$ different possibilities
- **Word** (2 bytes, or 16 bits): $2^{16} = 65536$
  - Double Word (4 byte, or 32 bits)
  - 32 bits: $2^{32} = 4294967296$
  - 64 bits: $2^{64} = 18446744073709551616$
  - The **word size** (i.e. the number of bytes in a word) is typically 4 bytes (32 bits) or 8 bytes (64 bits).

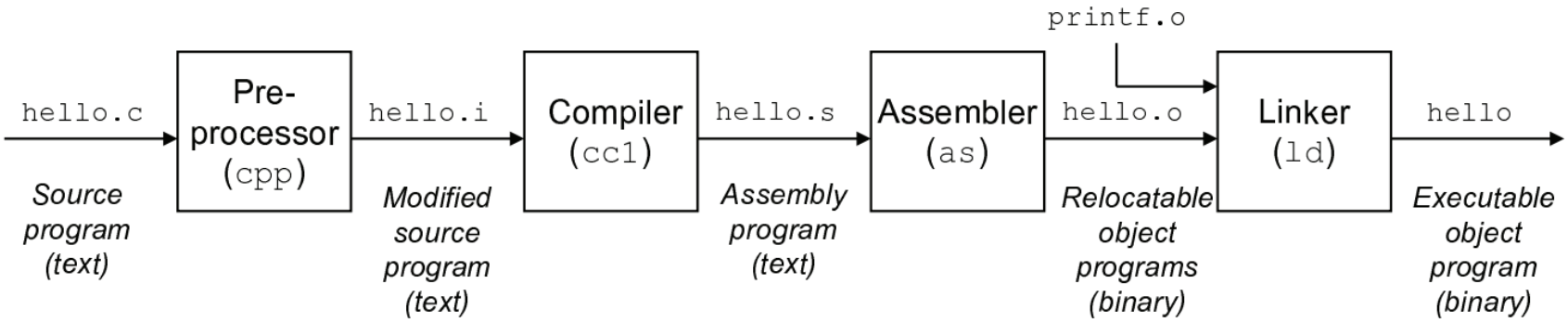
- A **file** is a sequence of bytes.
- A simple program is encoded in a **source file**.

ASCII (American Standard Code for Information Interchange) Code

- One byte for character 'A' : 01000001
- The computer representation in ASCII code for the name "ALICE" is

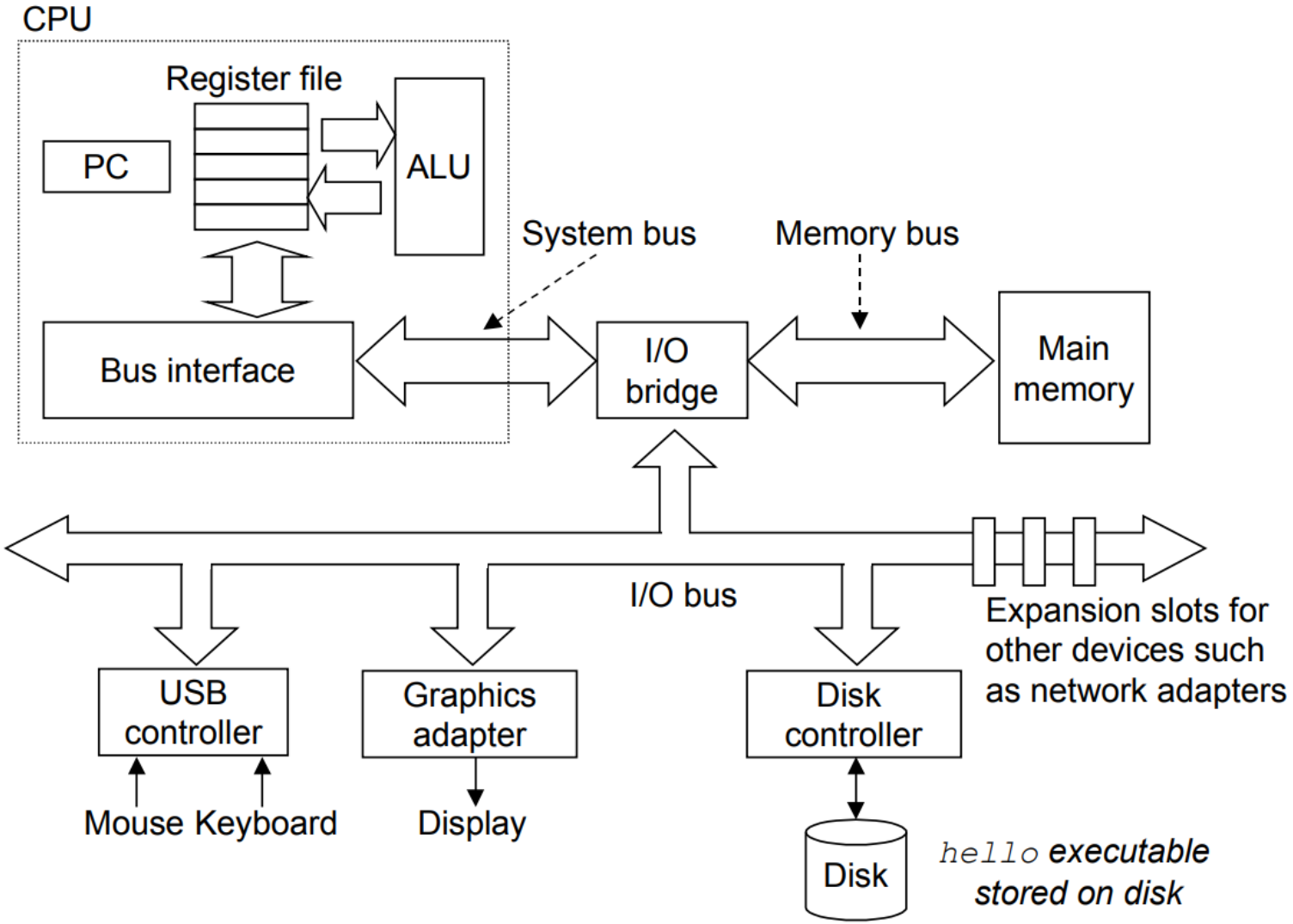
```
01000001    A
01001100    L
01001001    I
01000011    C
01000101    E
```

Compilation System



- **Preprocessing**: Modify C program according to directives starting with **#** (e.g. `#include <stdio.h>` inserts the contents of header file `stdio.h` into the program text).
- **Compilation**: Translate a high-level C program into a low-level assembly-language program.
- **Assembly**: Translate assembly-language program into machine-language instructions, saved in an ***object file***.
- **Linking**: Merge program with precompiled object files into an ***executable object file***.

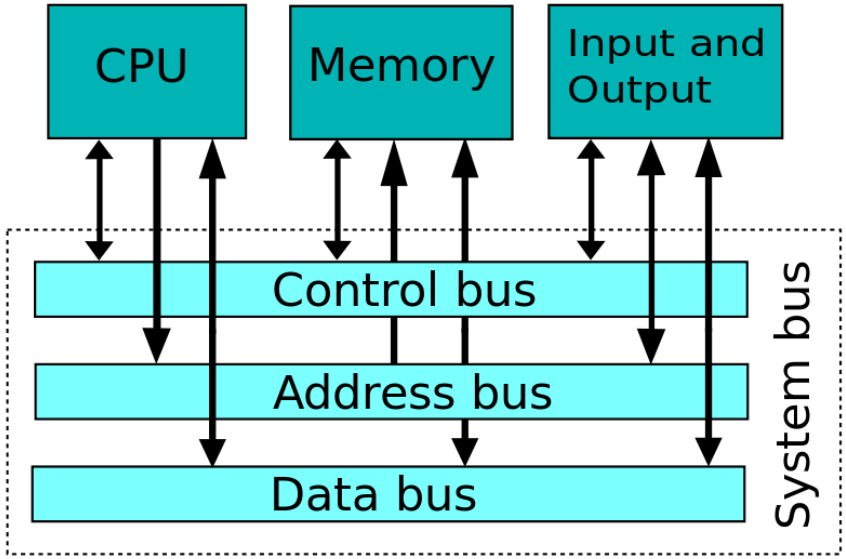
Computer Hardware



From book of Bryant and O'Hallaron, 2010, Fig. 1.4, page 6

Buses

- **Buses** are a collection of electrical conduits (circuits) that carry bytes of information between components.
- Buses transfer fixed-sized chunks of bytes known as **words**.



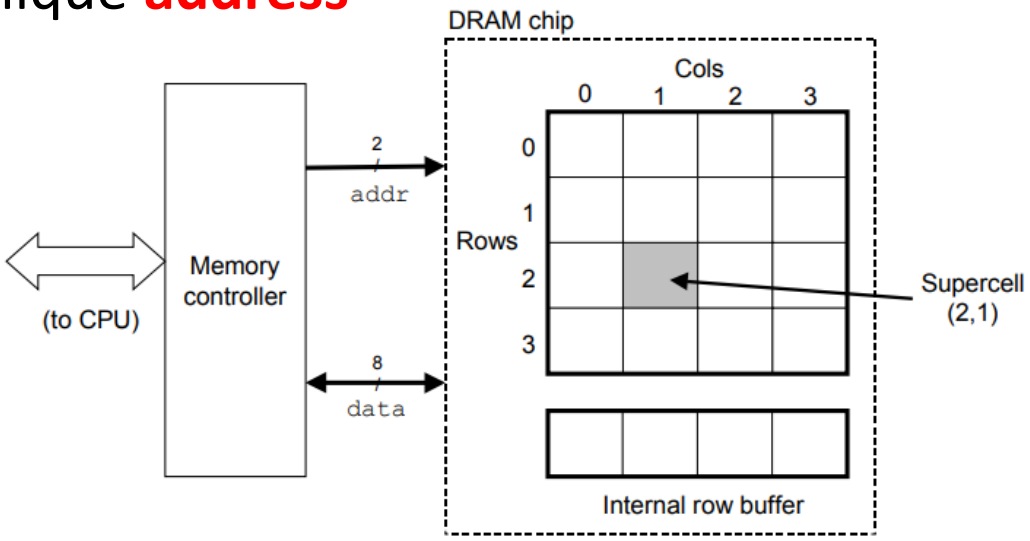
From Wikipedia article "Bus (computing)"

Input/Output (I/O) Devices

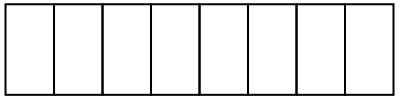
- **I/O devices** are the system's connection to the external world.
- **Input**
  - Keyboard
  - Computer mouse
- **Output**
  - Monitor display
  - Printer
- **Others**
  - Disk drive (or simply disk)
  - Network

Main Memory

- **Main memory** is a temporary storage device that holds both program and data when the program is running.
- **Physically**, main memory is a collection of **dynamic random access memory** (**DRAM**) chips.
- **Logically**, memory is a linear array of bytes, each with its own unique **address** (array index) starting at 0.



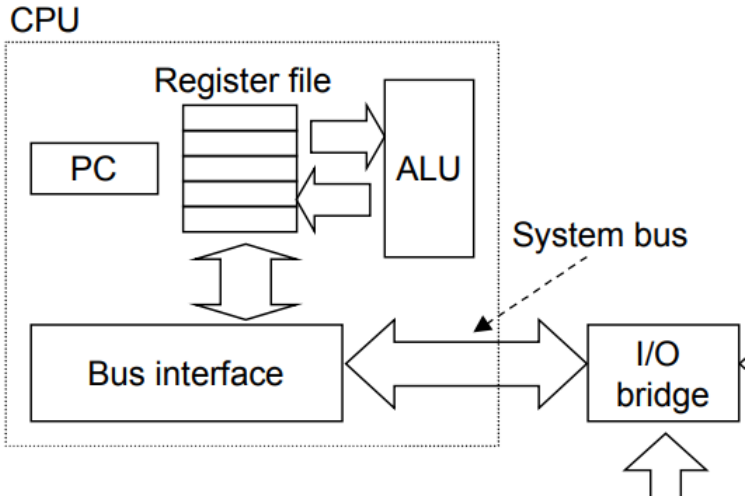
Storage Size Units

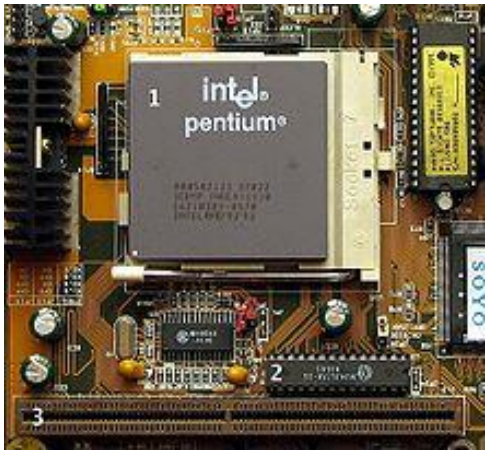
- **Bit** (b): 1 binary digit 
- **Byte** (B): 1B = 8 bits 
- **Kilobyte** (KB):
  1KB = 2^{10}B = **1024**B
- **Megabyte** (MB):
  1MB = 2^{10}KB = 2^{20}B
- **Gigabyte** (GB):
  1GB = 2^{10}MB = 2^{30}B
- **Terabyte** (TB):
  1TB = 2^{10}GB = 2^{40}B

October 24, Chinese Programmer's Day

Processor

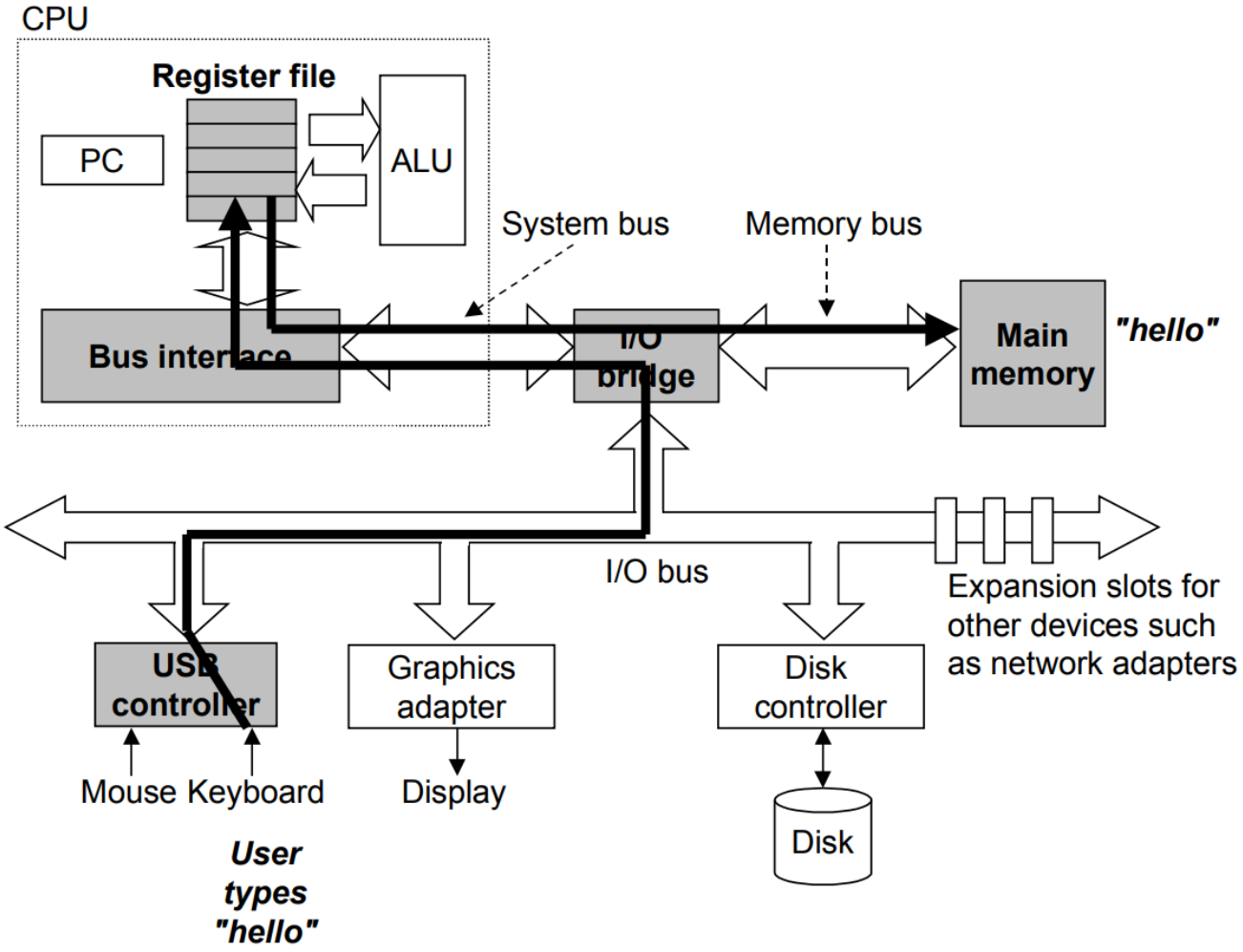
- **Central Processing Unit (CPU)**, also called **processor**, is the engine that interprets (or executes) instructions stored in main memory.
- **Control Unit (CU)**: directs and coordinates operations of other parts.
- **Program Counter (PC)**: a word-sized storage device (**register**) that points at an instruction in the main memory to be executed.
- **Register file**: a small storage device of a collection of word-sized registers.
- **Arithmetic/Logic Unit (ALU)**: a digital circuit that performs principal logical and arithmetic operations (add, subtract, multiply, divide, etc.) to compute new data and address values.





From Wikipedia

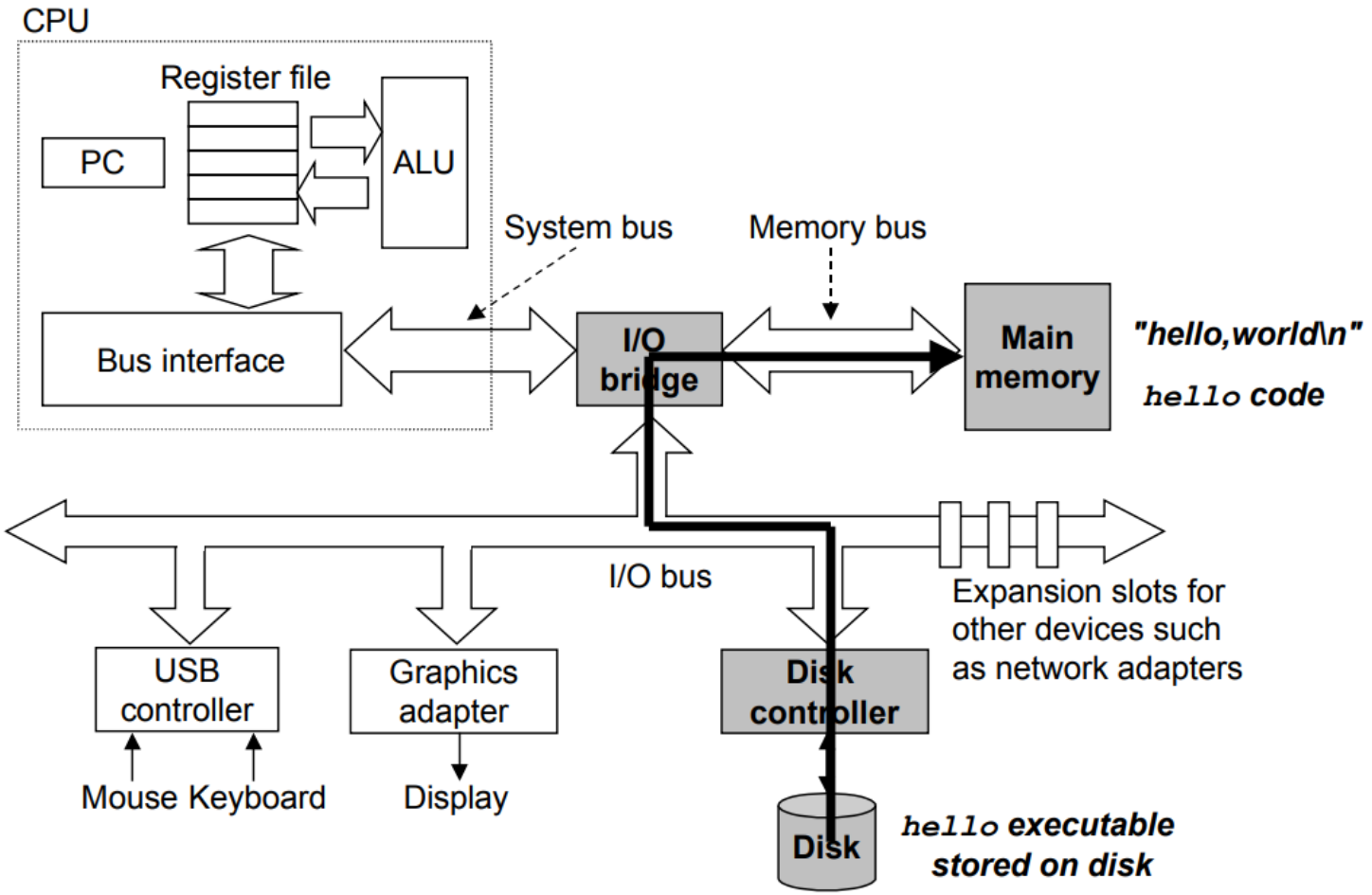
Running the hello Program (1)



Reading the hello command from the keyboard

From book of Bryant and O'Hallaron, 2010, Fig. 1.5, page 9

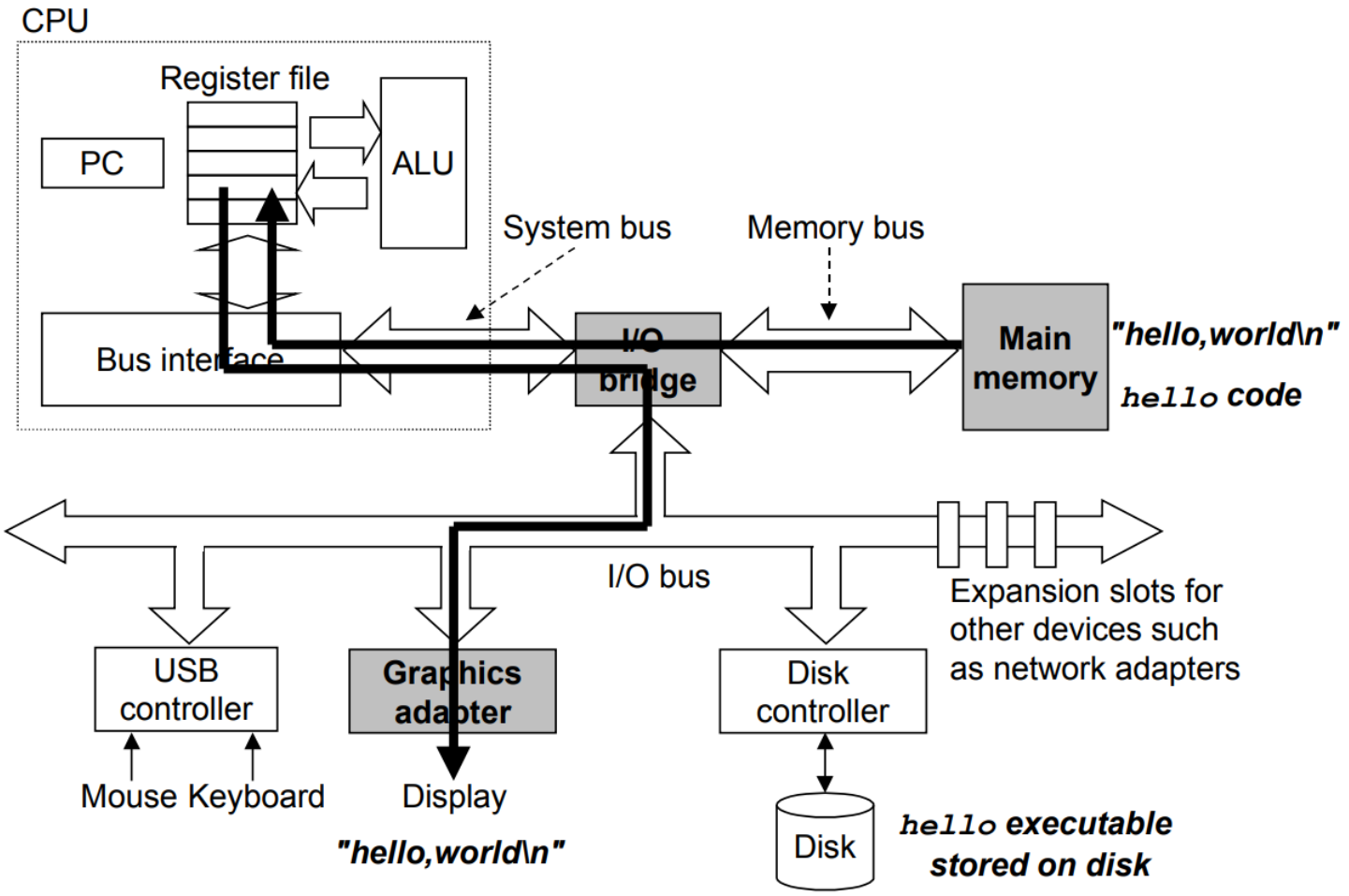
Running the hello Program (2)



Loading the executable from disk into main memory

From book of Bryant and O'Hallaron, 2010, Fig. 1.6, page 10

Running the hello Program (3)

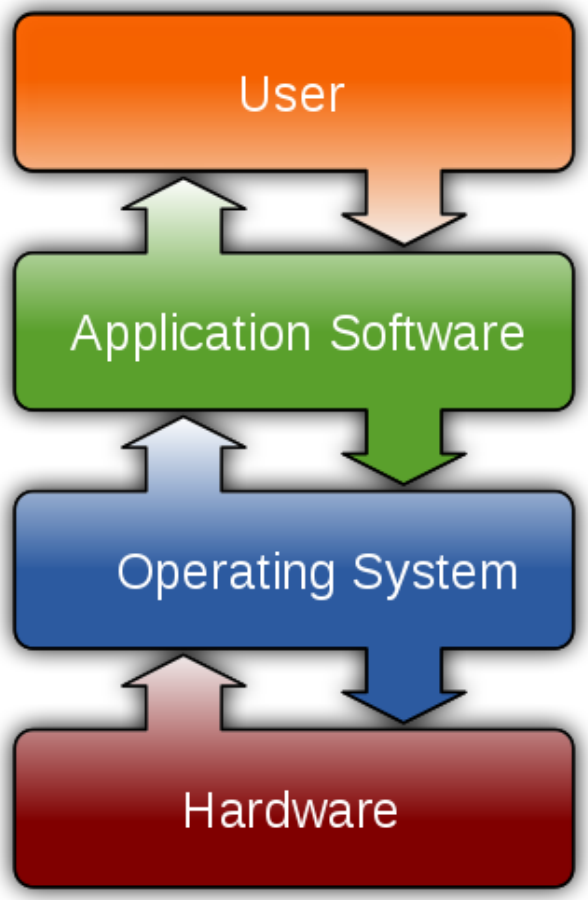


Writing the output string from memory to the display

From book of Bryant and O'Hallaron, 2010, Fig. 1.7, page 10

Computer Software

- **System software** directly operates computer hardware, to provide a platform for running or building application software:
  - Operating systems
  - Compilers
  - Database systems
  - Device drivers
- **Application software** is designed to perform functions or solve problems for the users:
  - Word processor
  - Email software
  - Computer games
- **Firmware** provides the low-level control for a device's specific hardware, e.g. programs in embedded systems like TV remote control, on-board computers in automobiles



From Wikipedia

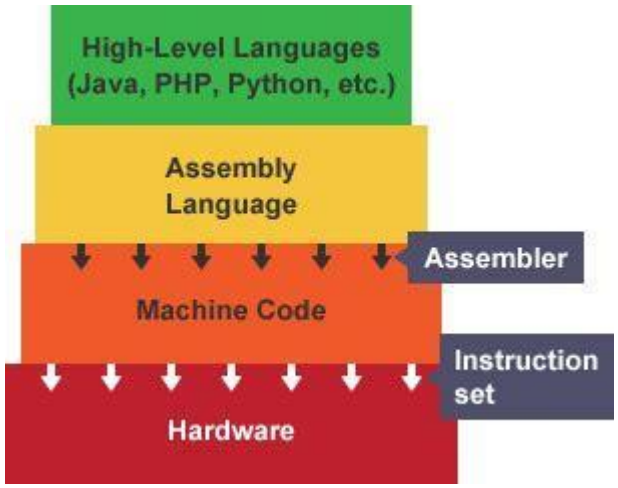
Programming Languages

- A **programming language** is a set of strings of symbols with a set of rules that allow a programmer to instruct a computer to perform certain tasks.

High Level Language **Assembly Language** **Machine Language**

```
i = j + k;                1    ILOAD j    // i = j + k
if (i == 3)               2    ILOAD k
    k = 0;                3    IADD
else                      4    ISTORE i
   j = j – 1;             5    ILOAD i    // if (i < 3)
                          6    BIPUSH 3
                          7    IF_ICMPEQ L1
                          8    ILOAD j    // j = j – 1
                          9    BIPUSH 1
                         10    ISUB
                         11    ISTORE j
                         12    GOTO L2
                         13 L1:          BIPUSH 0
                         14    ISTORE k
                         15 L2:
```

```
10111001 00000000
11010010 10100001
00000100 00000000
10001001 00000000
00001110 10001011
00000000 00011110
00000000 00000010
10111001 00000000
11100001 00000011
00010000 11000011
10001001 10100011
00001110 00000100
00000010 00000000
```



Programming Languages

- A **machine language** consists of instructions executed directly by CPU:
  - Each instruction is a binary strings of 0s and 1s
  - It is machine-dependent, and thus not portable
  - Fast to run, but difficult to read or write
- An **assembly language** uses English-like abbreviations to describe instructions:
  - Assembly code must be converted by **assembler** into machine code, in order to be executed
  - Not portable: tied to a specific computer architecture
- A **high-level language** has strong abstraction from the details of computer hardware. In most cases, C is considered a high-level language.
  - Easier to read and write than assembly and machine languages
  - Source code is converted into machine code, using compiler, assembler, etc.
  - Portable to different machines and operating systems
- Classification of high-level languages:
  - **Compiled languages**: C, C++
  - **Interpreted (scripting) languages**: Python, Perl, JavaScript
  - **Procedural** (such as C) vs. **object-oriented** (such as C++, Java)

Structure of a C Program

- A simple C program has the following structure:

```
/* comment line 1
   comment line 2
*/
preprocessor instructions
int main()
{
   statements;
   return 0;
}
```

An Example Program

```
/* a program to print Hello World! */

#include <stdio.h>  /* preprocessor instruction */
int main()          /* header */
{                   /* begin body */

   /* print message statement */
   printf(“hello, world!\n”);

   return 0;
}                   /* end body */
```

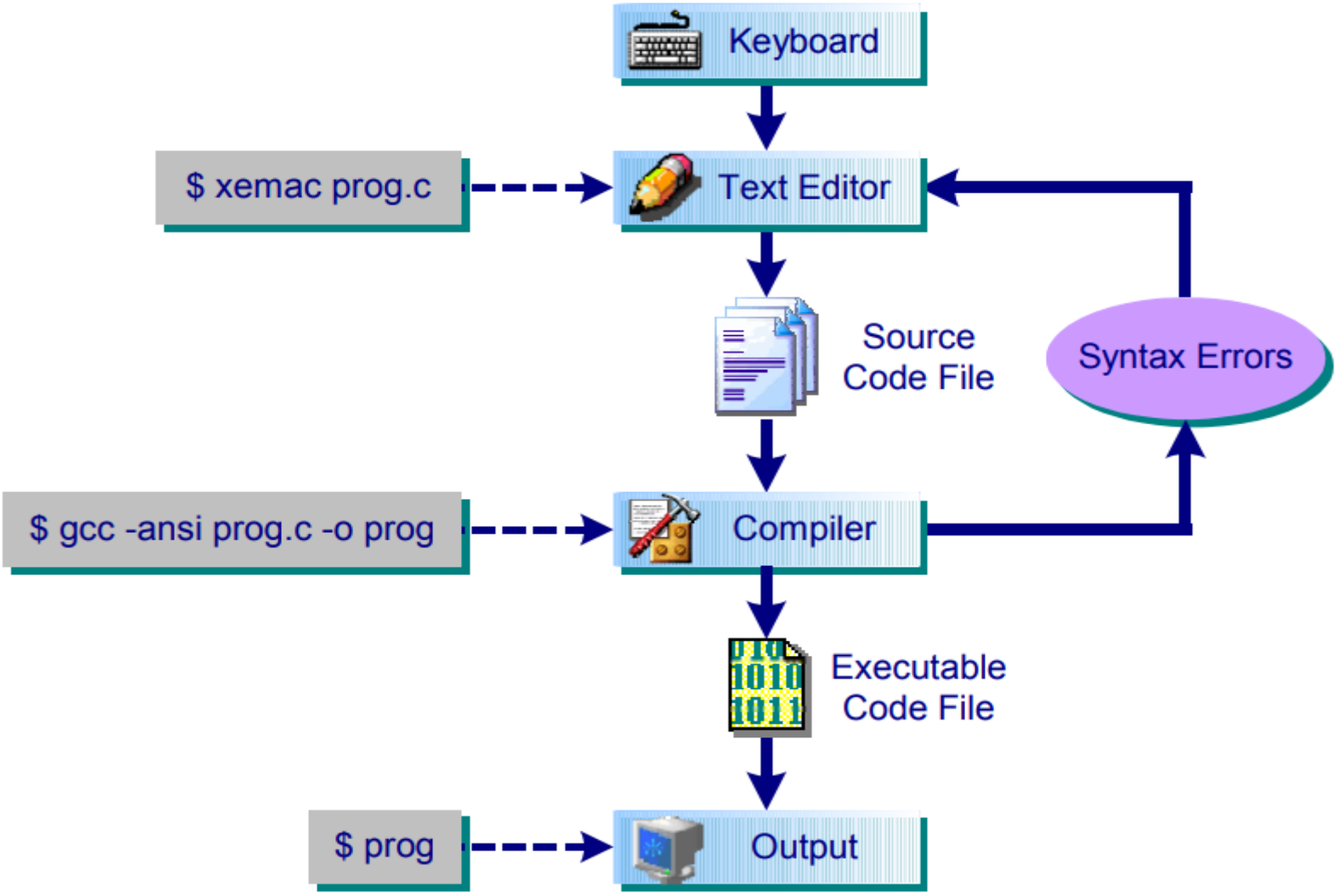
Structure of a C Program

- The **preprocessor instructions** refer to the instructions to the preprocessor of the compiler. All preprocessor instructions start with **#**.
  - The **`#include <filename>`** instruction tells the preprocessor to include the file "filename" into the text of the program file.
  - The **`#define <CONSTANT_NAME> <value>`** instruction defines a constant.
- **`main()`** (or **`int main()`**) is the header of the program. Every program has this header.

Structure of a C Program

- The **body** of the program is enclosed by the braces **{ }**
- A **statement** is a command to the computer. A statement may be a ***simple*** statement or a ***compound*** statement.
- **return 0** is the last statement in the program.
- You may add **comments** to the program to explain what the program is doing, or what a portion of the program is doing.
  - Multi-line comment: enclosed by **/*** and ***/**
  - Single-line comment: can use **//**

3 Steps to Develop a C Program



Step 1: Editing a Program: Hello World!

- May use any **text editor** (e.g. Notepad in Windows or xemacs in Linux), then save the program and name it as **prog.c**.

```
#include <stdio.h>

// a program to print “hello world!” on the screen
int main()
{
   printf(“hello, world!\n”);
   return 0;
}
```

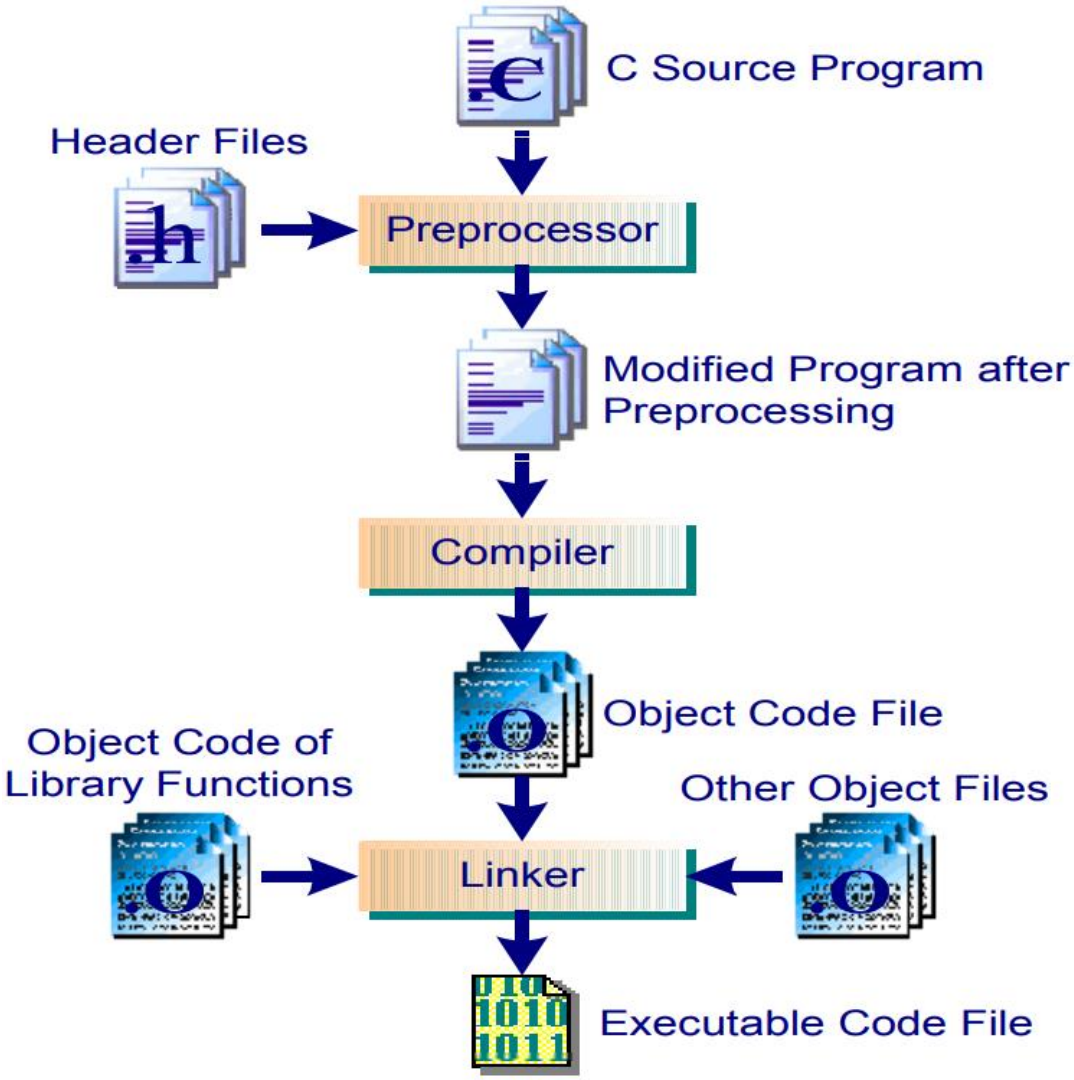
Another Example Program

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

/* a program to print the square root of 2.0 on
   the screen */
main()
{
   printf("The square root of 2 is %f", sqrt(2.0));
   return 0;
}
```

Step 2: Compilation of a C Program

After typing the C program prog.c into the editor, the program need be processed by the ***preprocessor***, the ***compiler*** (including ***assembler***)and the ***linker*** before you can execute the program.



Compilation of a C Program

- To compile your program, type

```
$gcc prog.c
```

  where **prog.c** is your program. **$** is the command prompt. **gcc** is the command to call the C compiler.

- If your program has no error, the compiler will call the linker automatically to do the linking and produce the executable file named **a.out**.
- To compile your program and name your executable file, type

```
$gcc prog.c -o prog
```

  The **–o** option tells the linker to call the executable file **prog** instead of the default name **a.out**.

Compilation of a C Program (that uses a math function)

- If your program uses some library functions like the **`sqrt()`** function from the math library to compute the square root of a number, you need to tell the compiler the library you use. The compilation command will become

```
$gcc prog.c –o prog -lm
```

- The **`-l`** operation is to tell the compiler the library you use. **`m`** indicates the math library. In addition to the change in the **`gcc`** command, you also need to add

```
#include <math.h>
```

At the beginning of your program to tell the preprocessor to include the definition file of the math library.

Step 3: Execution of a C Program

- To execute your program, just type

```
$a.out
```

or, if you have given a name to your executable file, say, **prog**, then just type

```
$prog
```

Your program will be executed.

Develop a C Program: Using Integrated Development Environment

- Major Integrated Development Environments (IDEs) for beginners (free for download):
  - **Visual Studio Code** with C/C++ extension (https://code.visualstudio.com/)
  - **Dev-C++** (version 5.11) (https://sourceforge.net/projects/orwelldevcpp/)

Recap

- The following concepts have been covered in this lecture:
  - A tour of computer systems (hardware, software)
  - Execution of “Hello world” program
  - C Program Structure
  - How to develop a C Program