CS100
Introduction to Programming

Lecture 1. C Program Structure

What is a computer system?

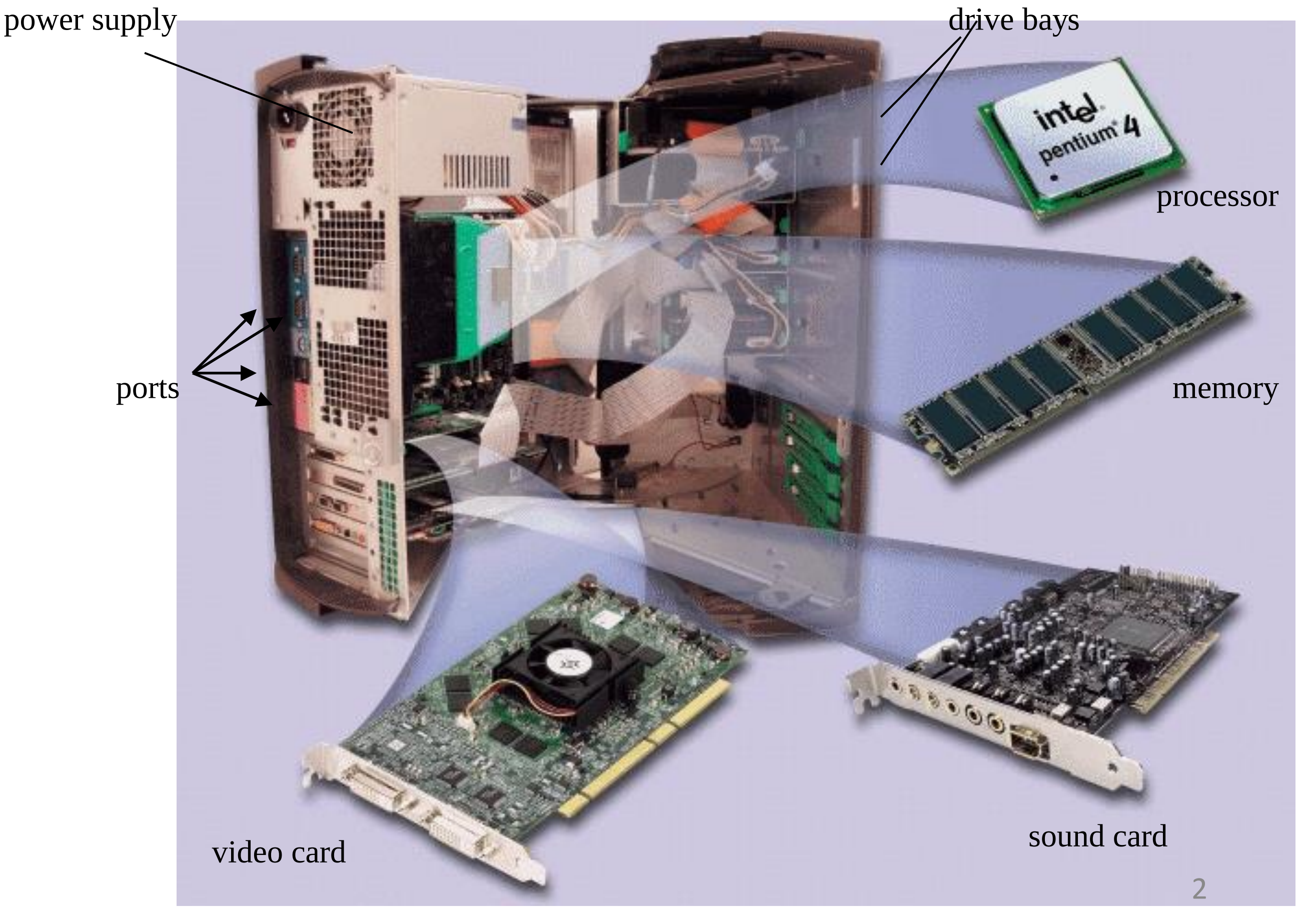
- A **computer system** consists of **hardware** and **system software** that work together to run **application software**.

Systems software:

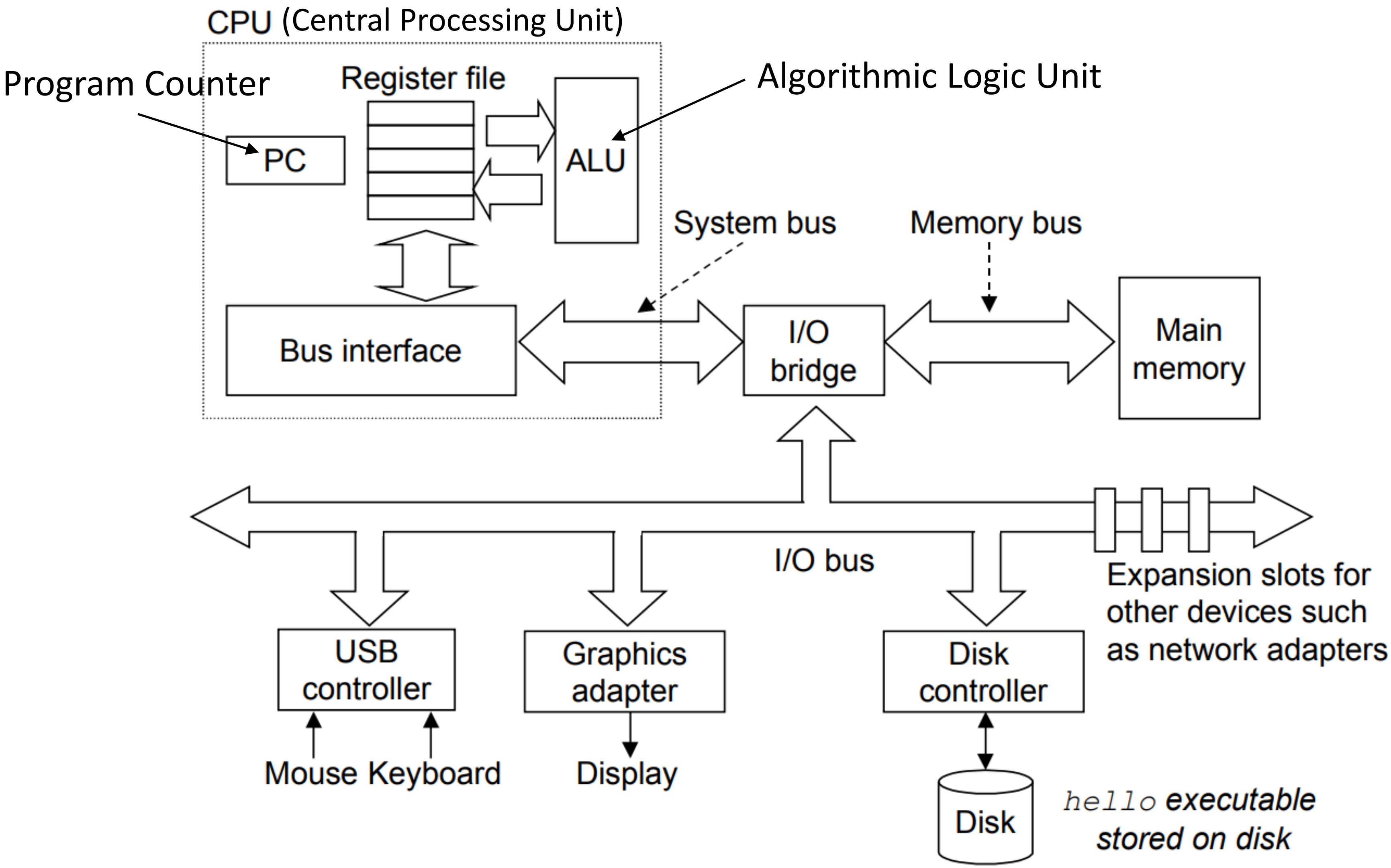
- Operating system
- Compiler
- Linker
- Debugger

Application software:

- Word processor
- Web browser
- Media player



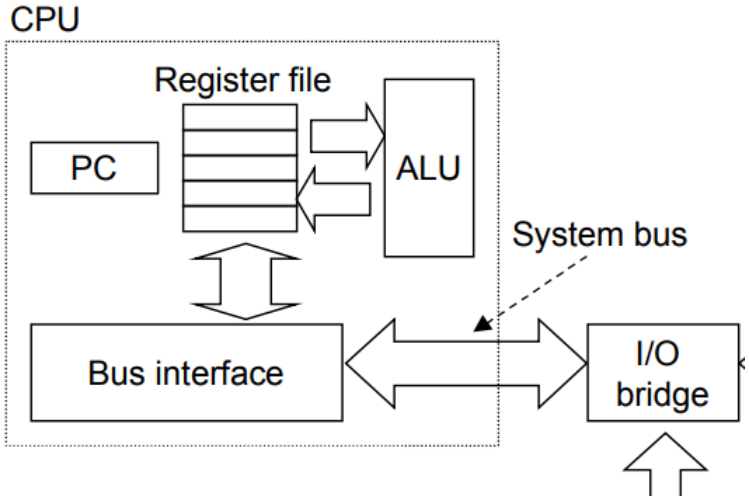
Computer Hardware

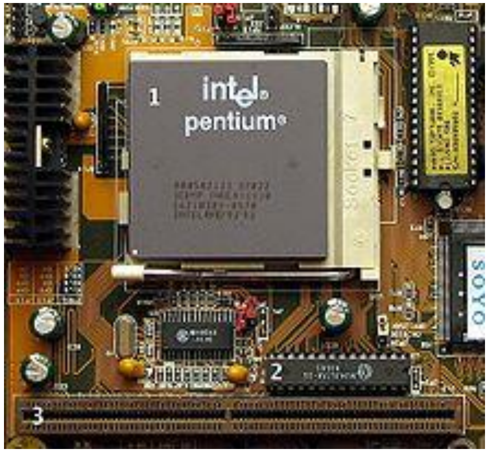


From book of Bryant and O'Hallaron, 2010, Fig. 1.4, page 6

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.

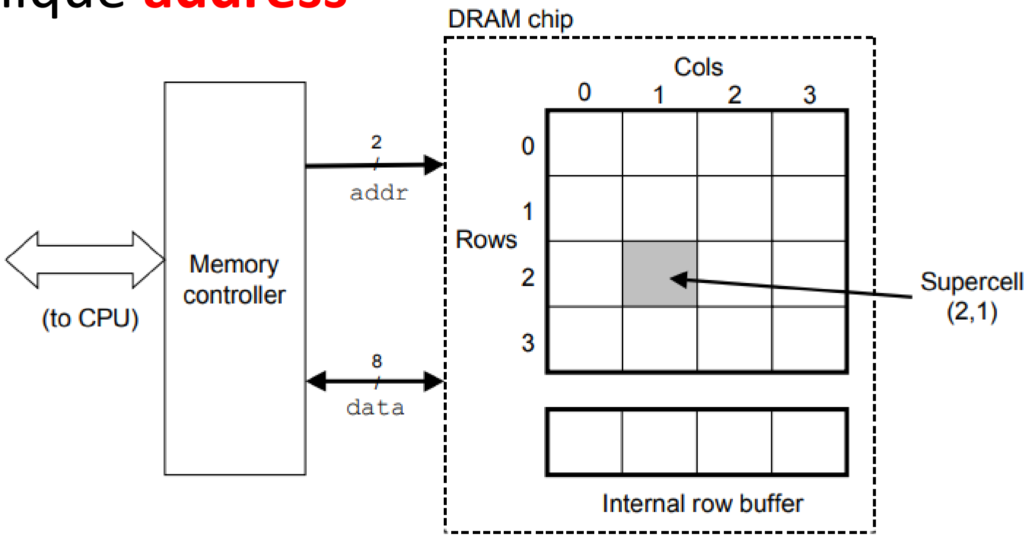




From Wikipedia

Main Memory

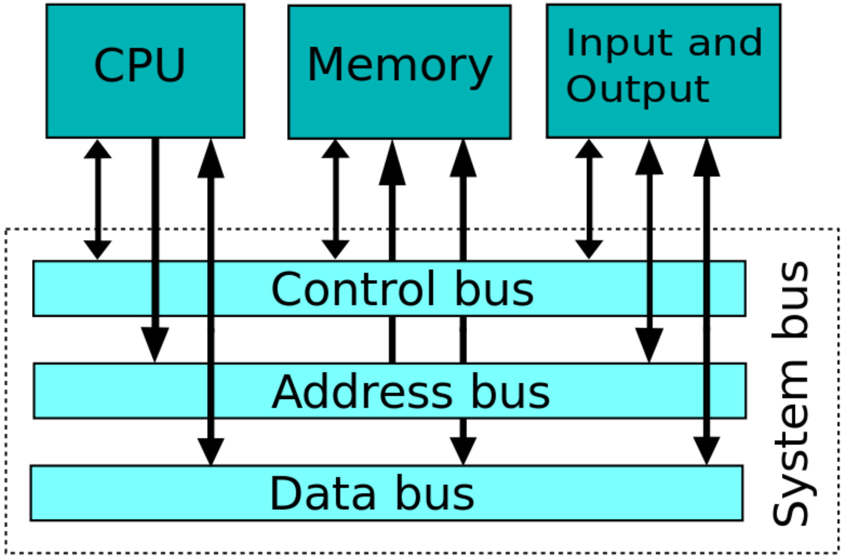
- **Main memory** is a temporary storage device that holds both program and data when the program is running.
- **Physically**, main memory is a collection of **dynamic random access memory** (**DRAM**) chips.
- **Logically**, memory is a linear array of bytes, each with its own unique **address** (array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



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

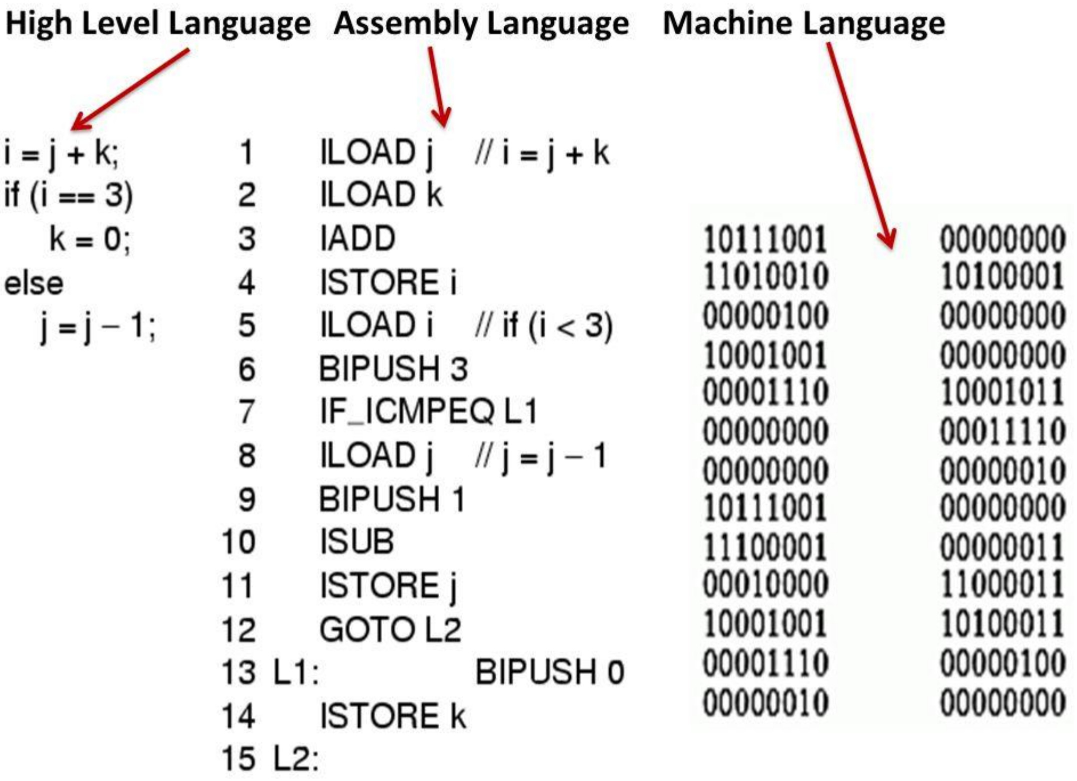
What is a computer program?

- **Instruction**
  - A single operation of a processor (binary code)
  - Defined by the processor instruction set

- **Computer program**
  - A collection of instructions with data (all binary codes)
  - Performs a specific task when executed by a computer

Programming Languages

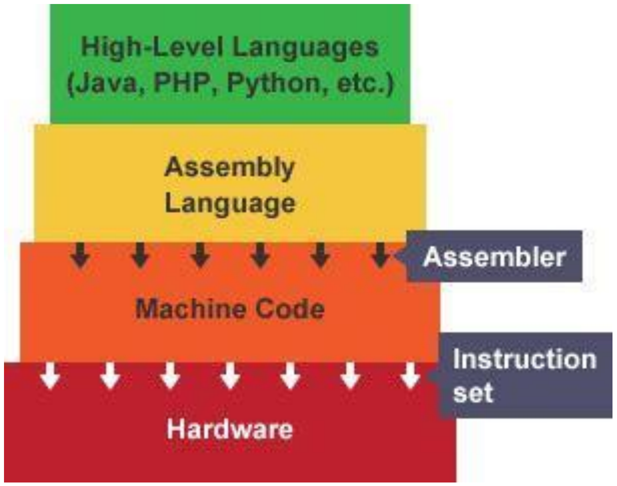
- A **programming language** is a set of strings of symbols with a set of rules that allow a programmer to instruct a computer to perform certain tasks.



```
i = j + k;
if (i == 3)
    k = 0;
else
    j = j – 1;
```

```
    ILOAD j     // i = j + k
    ILOAD k
    IADD
    ISTORE i
    ILOAD i     // if (i < 3)
    BIPUSH 3
    IF_ICMPEQ L1
    ILOAD j     // j = j – 1
    BIPUSH 1
    ISUB
    ISTORE j
    GOTO L2
L1:             BIPUSH 0
    ISTORE k
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

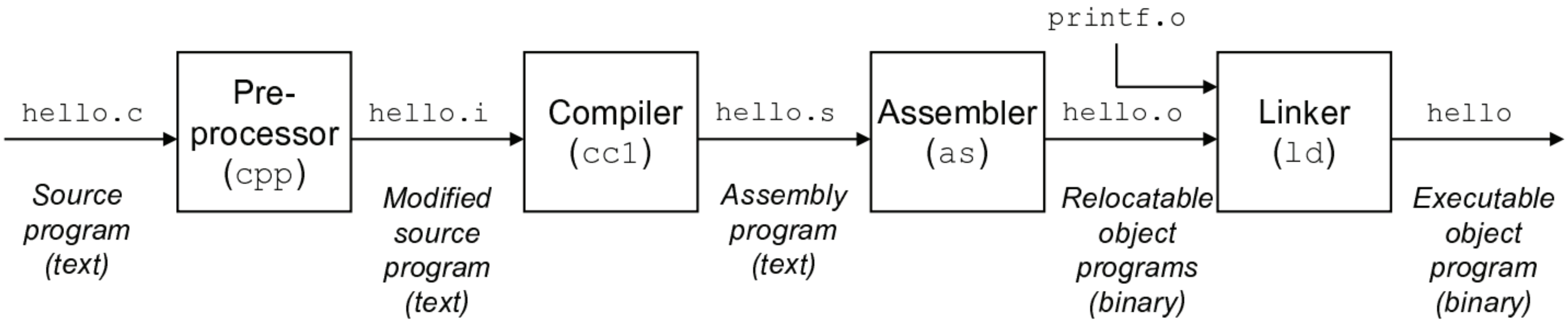
- A **machine language** consists of instructions executed directly by CPU:
  - Each instruction is a binary strings of 0s and 1s
  - It is machine-dependent, and thus not portable
  - Fast to run, but difficult to read or write

- 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)

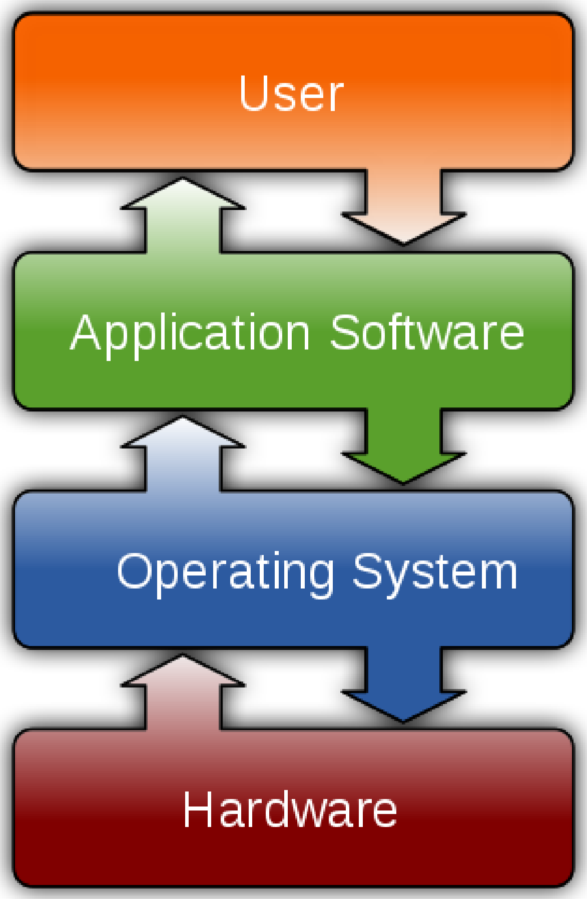
Compilation System



- **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 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

Why C/C++ Programming Language?

- **Advantages**:
  - Powerful, flexible, efficient, portable
  - A high-level language with low-level operations
  - Closely related with UNIX / Linux
  - Influence on other languages: C#, Java

- **Disadvantages**:
  - Using **pointers** might be confusing and cause errors
  - Requires attention to low-level details
  - More difficult to learn, especially for C++

How to learn C/C++ well?

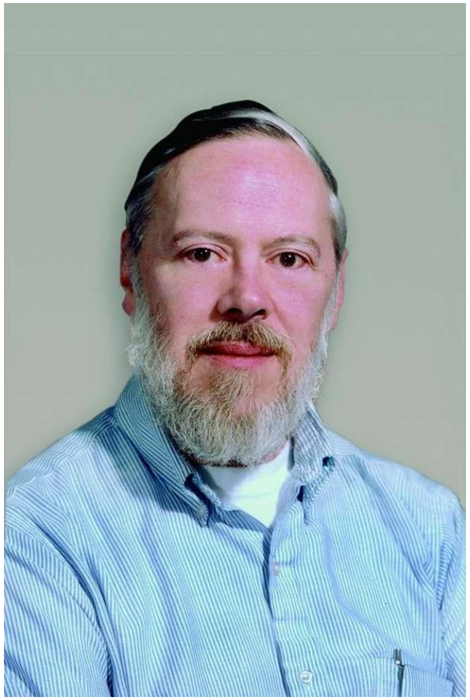
- **Practice, practice, practice!**
  - Only when you program very often can you really get the key experience
  - Refer to other good programming codes
  - Discuss with fellows about specific skills
  - Think more deeply about program design and how to do debugging

Plan for Learning C

- **Week 1**
  - C program structure
  - Data types, operators and expressions
- **Week 2**
  - Input / Output
  - Control flow
- **Week 3**
  - Functions
  - Pointers
- **Week 4**
  - Arrays
  - Character strings
- **Week 5**
  - Structures
  - Recursions
- **Week 6**
  - Basic algorithms and Advanced C
  - Revision on C

A Brief History of C

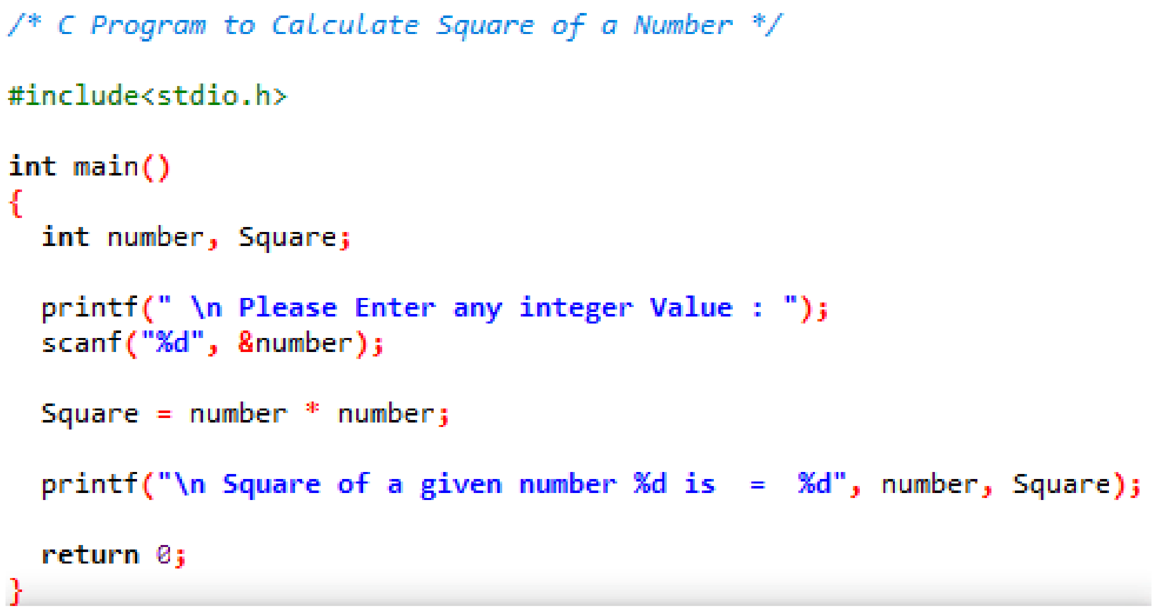
- **UNIX operating system**
  - In 1969, a small group of AT&T Bell Labs led by Ken Thompson and Dennis Ritchie began to develop UNIX
  - In 1973, UNIX kernel was rewritten in C
- **Creation of C language**
  - From 1969 to 1973, Dennis Ritchie developed C in Bell Labs
  - In 1978, Kernighan and Ritchie 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

C Programs

- **A list of character string expressions**
  – Usually saved as text files (named *.c)
  – Sentences are separated by ‘;’



```
/* C Program to Calculate Square of a Number */

#include<stdio.h>

int main()
{
  int number, Square;

  printf(" \n Please Enter any integer Value : ");
  scanf("%d", &number);

  Square = number * number;

  printf("\n Square of a given number %d is  =  %d", number, Square);

  return 0;
}
```

ASCII Code

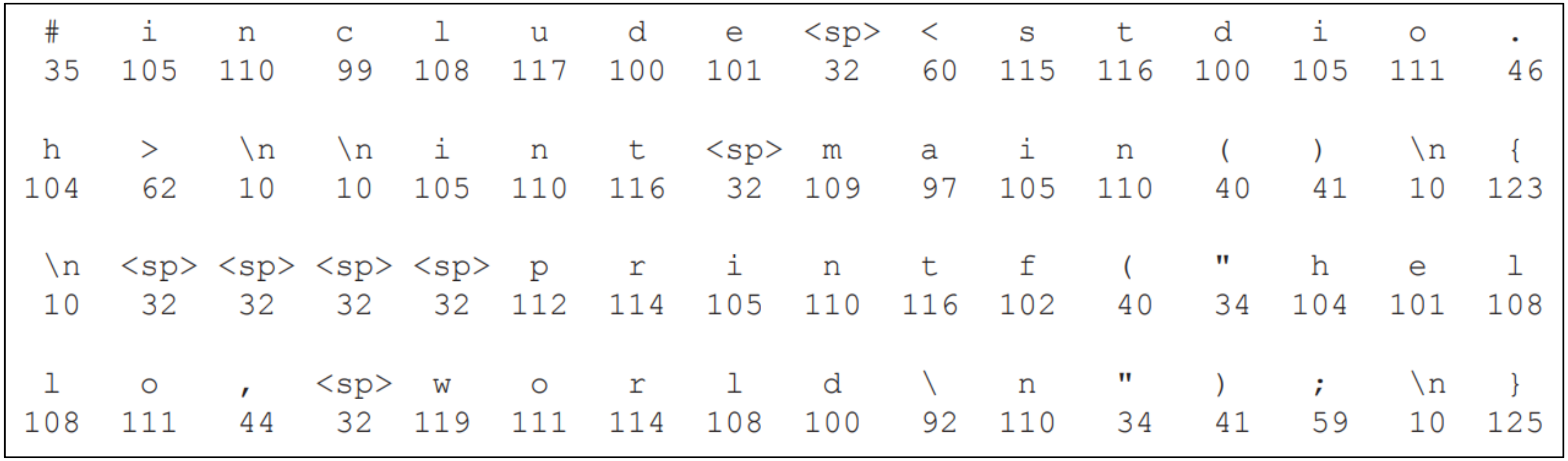
- **American Standard Code for Information Interchange**
  - A character encoding standard for electronic communication
  - ASCII codes represent text in computers, telecommunications equipment, and other devices

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

Program Storage

- **Where do programs store when compiled?**
  - In hard disk
    - A binary file containing all the compiled binary bits
    - Instructions and data
  - When loaded
    - Stored in system memory
    - Operating system can help load and run the program
  - How to measure the size?

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

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 or DWORD (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).

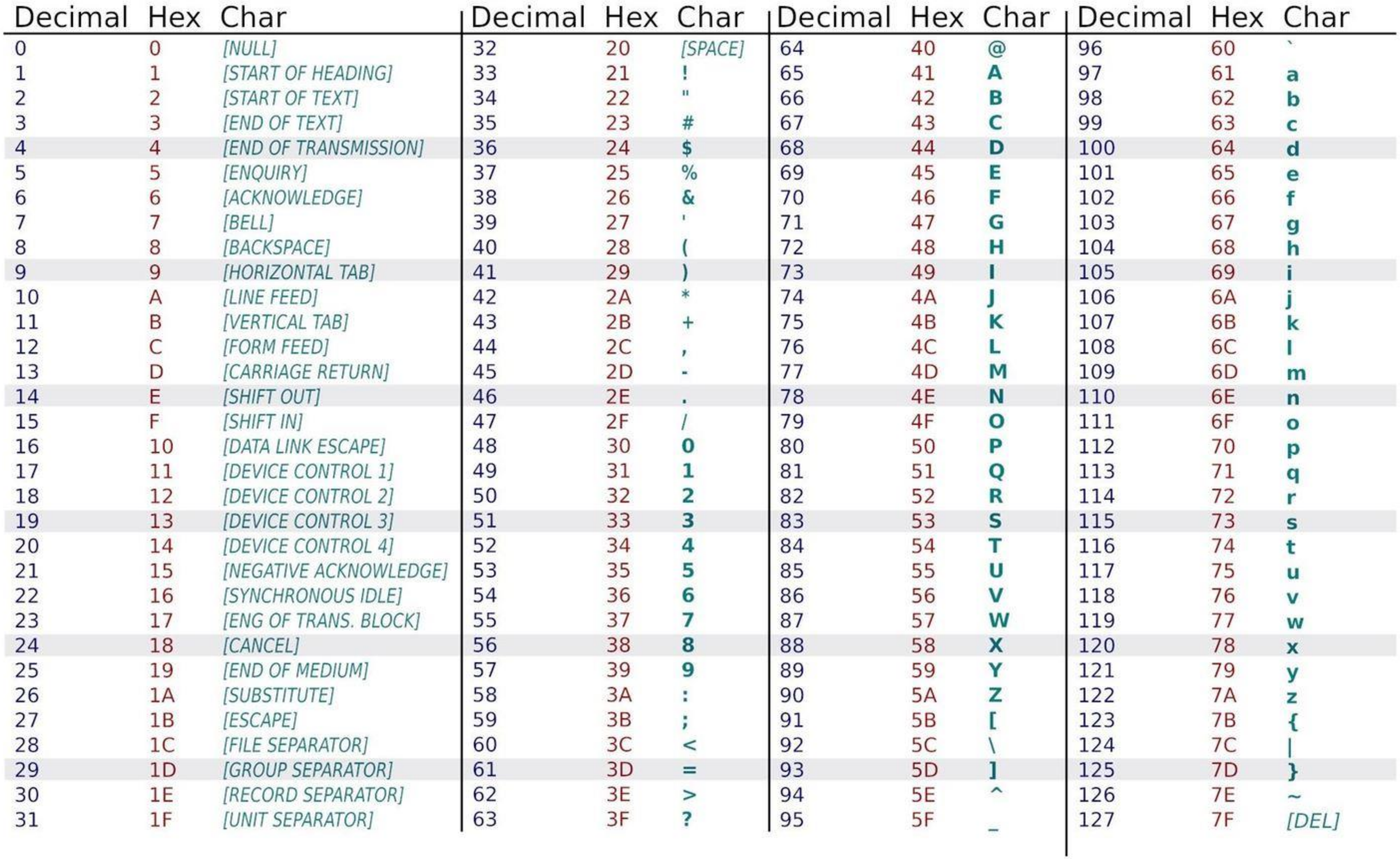
- A **file** is a sequence of bytes.
- A simple program is encoded in a **source file**.

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

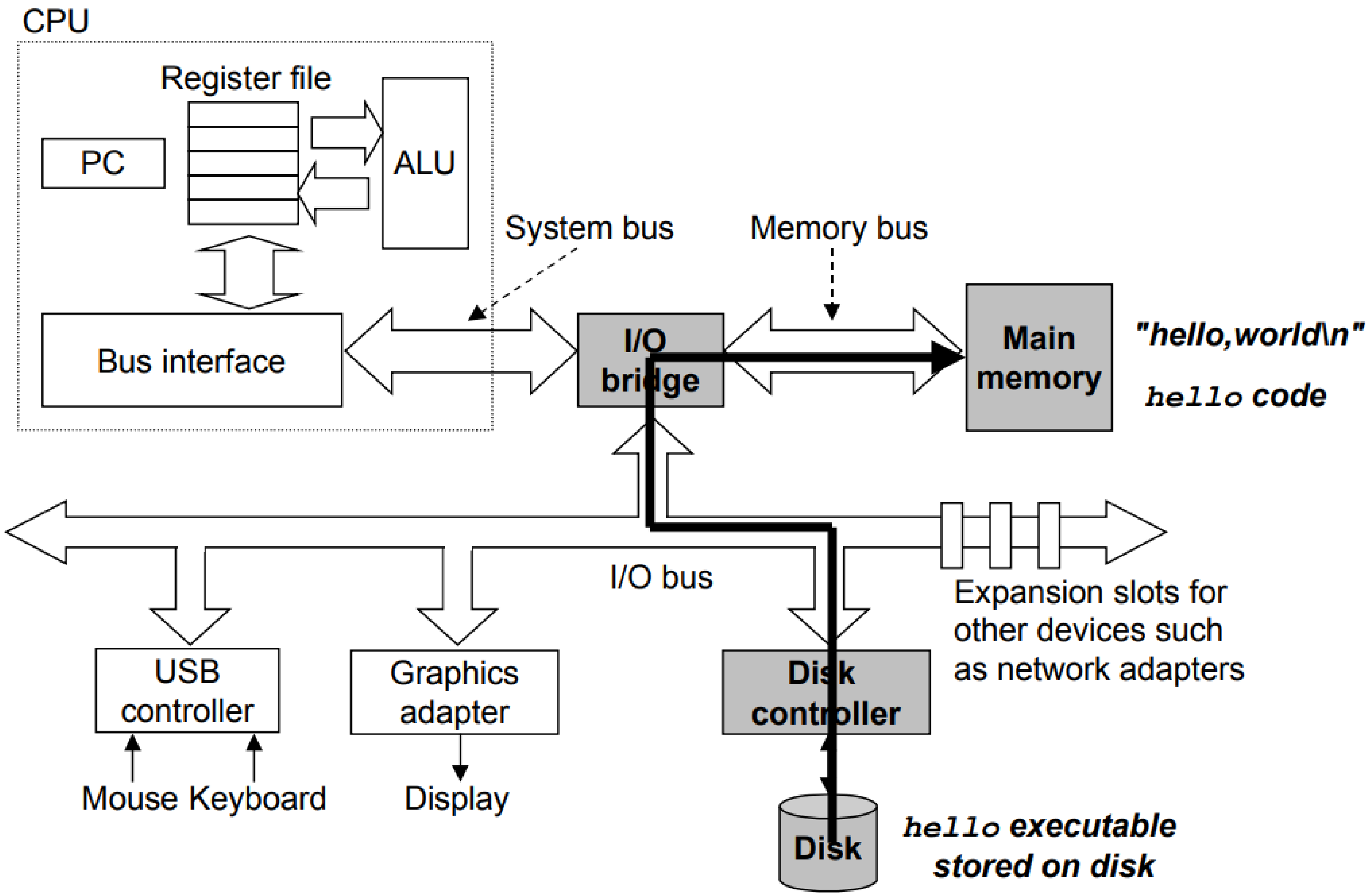
01000001	A
01001100	L
01001001	I
01000011	C
01000101	E

ASCII TABLE



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

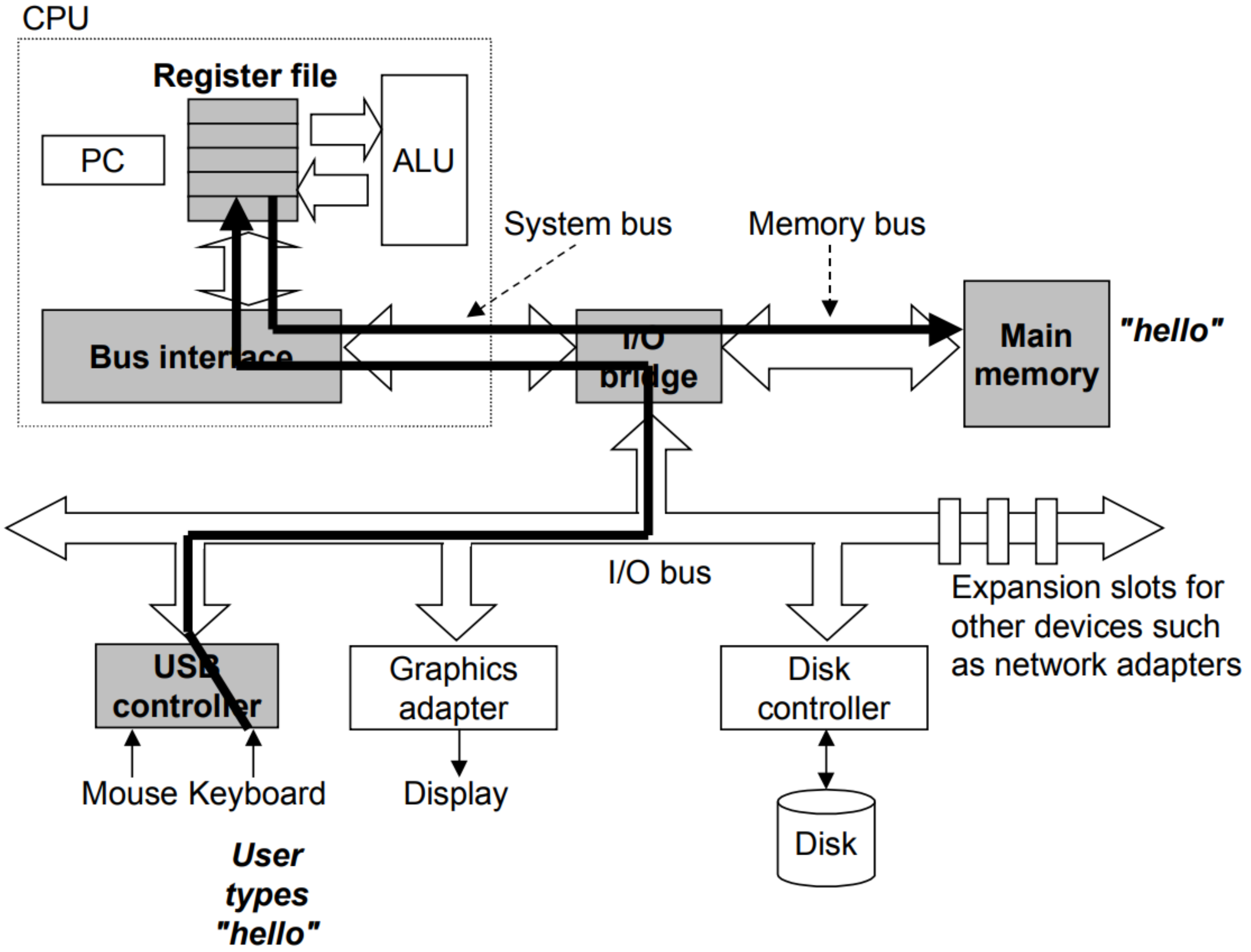
Running the hello Program (1)



Loading the executable from disk into main memory

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

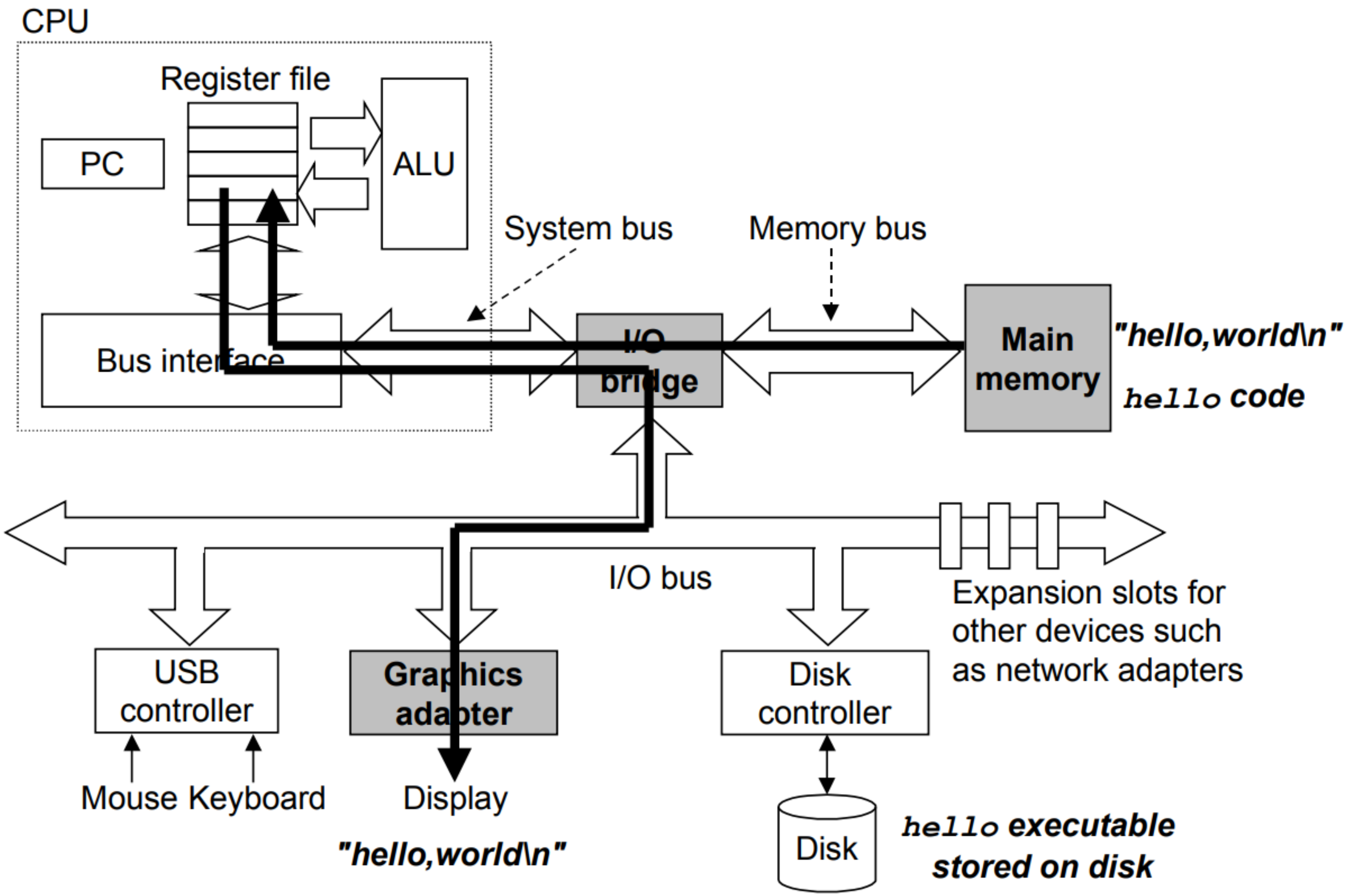
Running the hello Program (2)



Reading the hello command from the keyboard

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

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

Structure of a C Program

- A simple C program has the following structure (always starting from main()):

```
/* 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

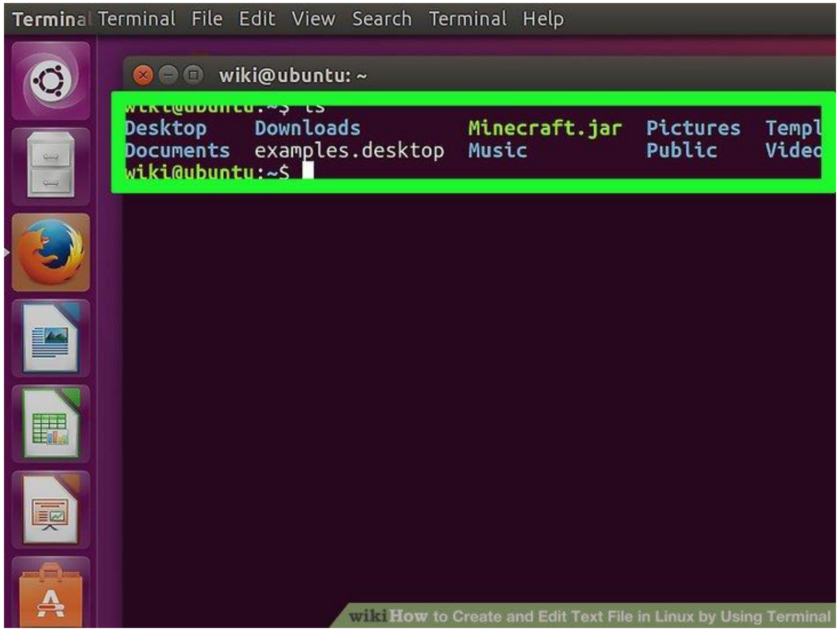
- The **preprocessor instructions** refer to the instructions to the preprocessor of the compiler. All preprocessor instructions start with **`#`**.
  - The **`#include <filename>`** instruction tells the preprocessor to include the file “filename” into the text of the program file.
  - The **`#define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

Structure of a C Program

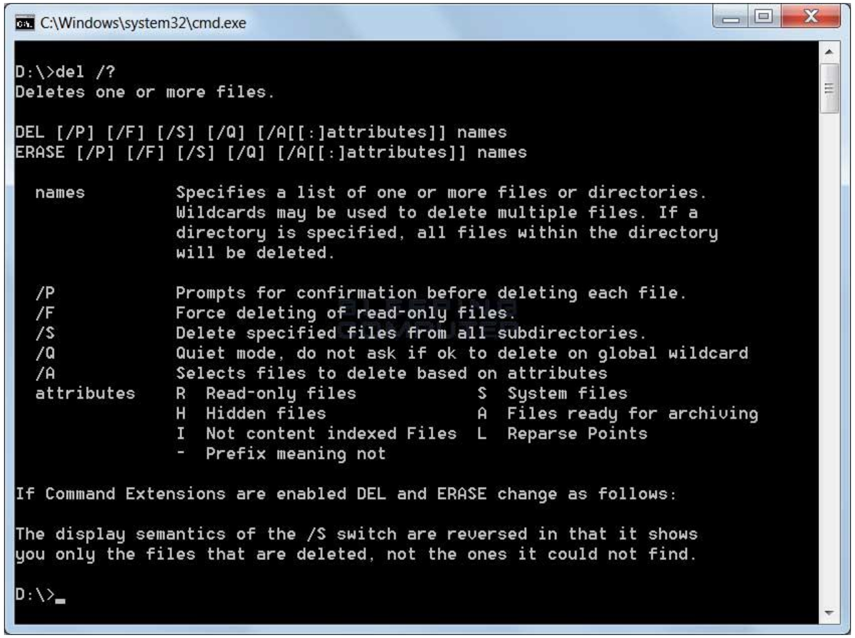
- The **body** of the program is enclosed by the braces **{ }**
- A **statement** is a command to the computer. A statement may be a ***simple*** statement or a ***compound*** statement.
- **return 0** is the last statement in the program.
- You may add **comments** to the program to explain what the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

Console

- **A command line interface**
  - Take input strings and display output strings.

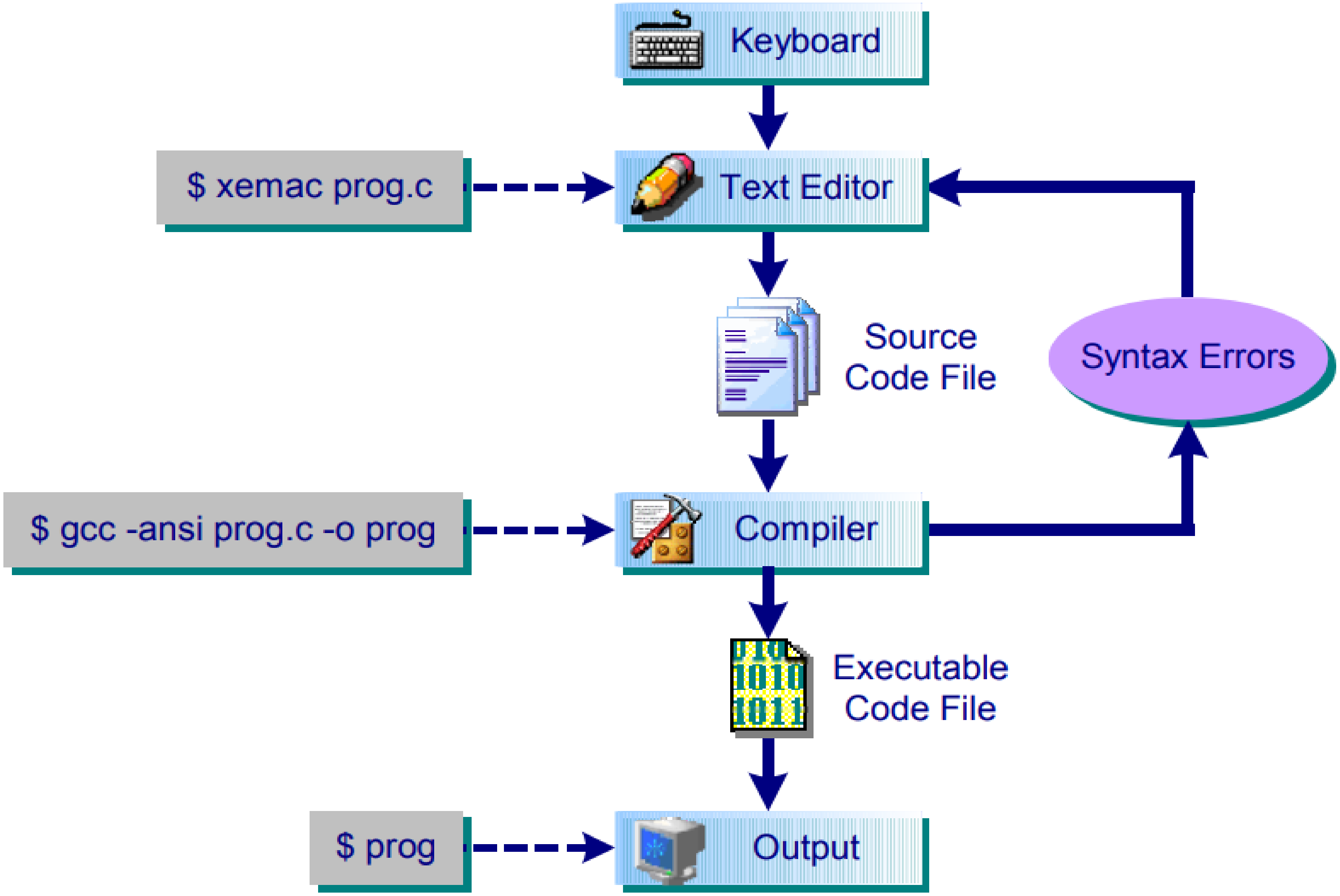

Terminal Terminal File Edit View Search Terminal Help
wiki@ubuntu: ~
wiki How to Create and Edit Text File in Linux by Using Terminal

```
wiki@ubuntu:~$ ls
Desktop    Downloads         Minecraft.jar  Pictures  Templ
Documents  examples.desktop  Music          Public    Video
wiki@ubuntu:~$
```



```
D:\>del /?
Deletes one or more files.

DEL [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names
ERASE [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names

  names         Specifies a list of one or more files or directories.
                Wildcards may be used to delete multiple files. If a
                directory is specified, all files within the directory
                will be deleted.

  /P            Prompts for confirmation before deleting each file.
  /F            Force deleting of read-only files.
  /S            Delete specified files from all subdirectories.
  /Q            Quiet mode, do not ask if ok to delete on global wildcard
  /A            Selects files to delete based on attributes
  attributes    R  Read-only files            S  System files
                H  Hidden files               A  Files ready for archiving
                I  Not content indexed Files  L  Reparse Points
                -  Prefix meaning not

If Command Extensions are enabled DEL and ERASE change as follows:

The display semantics of the /S switch are reversed in that it shows
you only the files that are deleted, not the ones it could not find.

D:\>_
```

3 Steps to Develop a C Program



Keyboard

Text Editor

Source Code File

Syntax Errors

Compiler

Executable Code File

Output

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

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**.

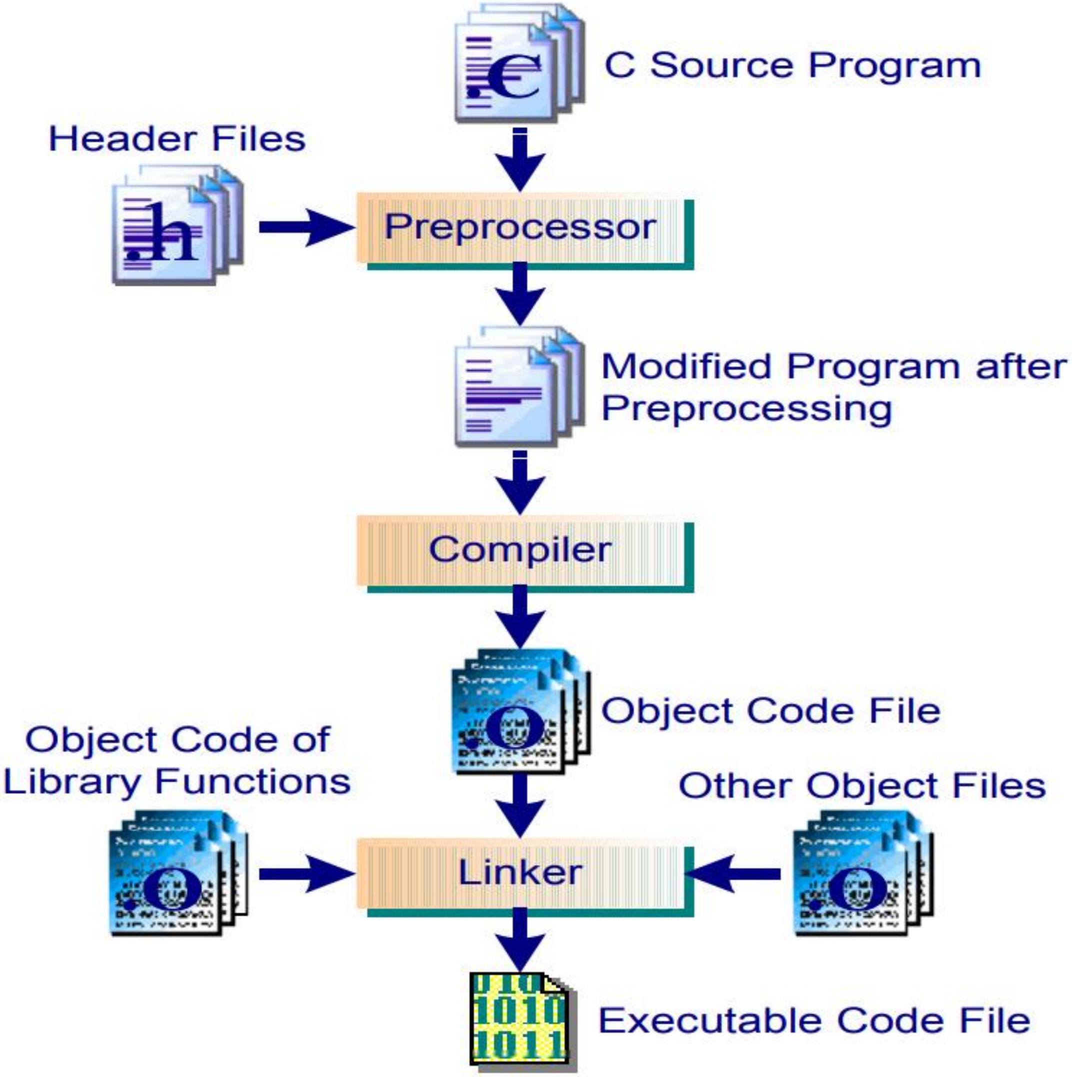
```
#include <stdio.h>

// 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 */

int 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 needs 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 my_program
```

  The **–o** option tells the linker to write to the executable file **my_program** 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.

Structure of a C Program

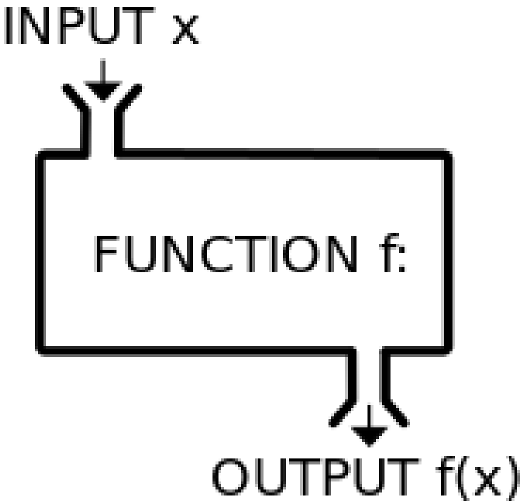
- **C function**
  - Very much like the function in mathematics
  - More abstract

$$y = f(x)$$

$$\{(x, f(x)) : x \in X\}$$

Example:

$$f(x) = \sin(x^2 + 1)$$



Structure of a C Program

- **C function**
  - Declaration

```
float square(float x);
```

  - Implementation

```
float square(float x)
{
     return x*x;
}
```

Structure of a C Program

- **C variable**
  - – Nothing but a name given to a storage area that our programs can manipulate
  - – Variable type: determines the size and layout of the variable's memory

char:	1 byte
int:	4 bytes
long:	8 bytes
float:	4 bytes
double:	8 bytes

Structure of a C Program

- **C variable**
  - Declaration of a variable

```
int a;
float b;
```

  - Assigning variable values

```
a=10;
b=15.6;
```

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/)

C Standard Library

- **The standard library for the C programming language**
  - Provides macros, type definitions and functions
  - Mathematical computations
  - Input/output processing
  - Memory management
  - Several other operating system services