



# Introduction CISC1600



Dr.Xiaolan Zhang, Fordham Univ

# In this class...

---

- ▶ A tour of computer system
- ▶ The first C++ program and the different stages of programming
  - ▶ An introduction to the programming environment we use
  - ▶ A dissection of a C++ program

# What is a Computer System ?

---

- ▶ **Computer System is a system capable of**
  - ▶ Performing computations, storing data
  - ▶ Making logical decisions:
    - ▶ If income is greater than 25K, use higher tax rate; otherwise, use lower tax rate.
  - ▶ Works billions of times faster than human beings; not smarter: good at *repetitive (tedious)* tasks
  - ▶ Interacts with devices
- ▶ **Runs programs to handle different tasks:**
  - ▶ Balance checkbooks
  - ▶ Process words, display documents
  - ▶ Play games, movies, music

# What is Computer System ?

---

- Components of a computer system:
  - **Hardware:** electronic and mechanical parts.
    - *Input devices:*
    - *Output devices*
    - *System Unit:*
  - **Software:** a collection of programs
    - A *program* tells computer the sequence of steps needed to fulfill a task

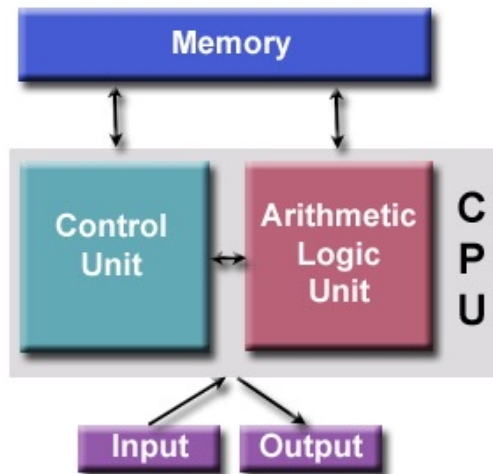
# What is Programming?

---

- ▶ **Programming** — act of designing and implementing programs
  - ▶ Most *users* are **not** programmers
  - ▶ Programmers write instructions that comprise software in various **programming languages**
- ▶ Programming is an *essential skill* for a computer scientist
- ▶ *Not* the only skill required to be a successful computer scientist
- ▶ Once certain skills are developed, even simple programs can be thrilling

# Computer Hardware

- **Hardware:** electronic and mechanical parts.
  - *Input devices:* Obtains data from outside computer.
    - keyboard, mouse, disk or scanner
  - *Output devices:* Makes info available outside computer.
    - Screens, paper printouts, speakers
  - *System Unit:* Disks, Memory, Processor (Central Processing Units)



# Main Hardware Components: **Processor**

---

## ▶ **Central Processing Unit or CPU**

- Brain of a computer system: directly or indirectly controls all the other components.
- Performs program control (locate, and execute program instructions), arithmetic, and data movement
- *One operation at a time* (e.g., a CPU cycle)

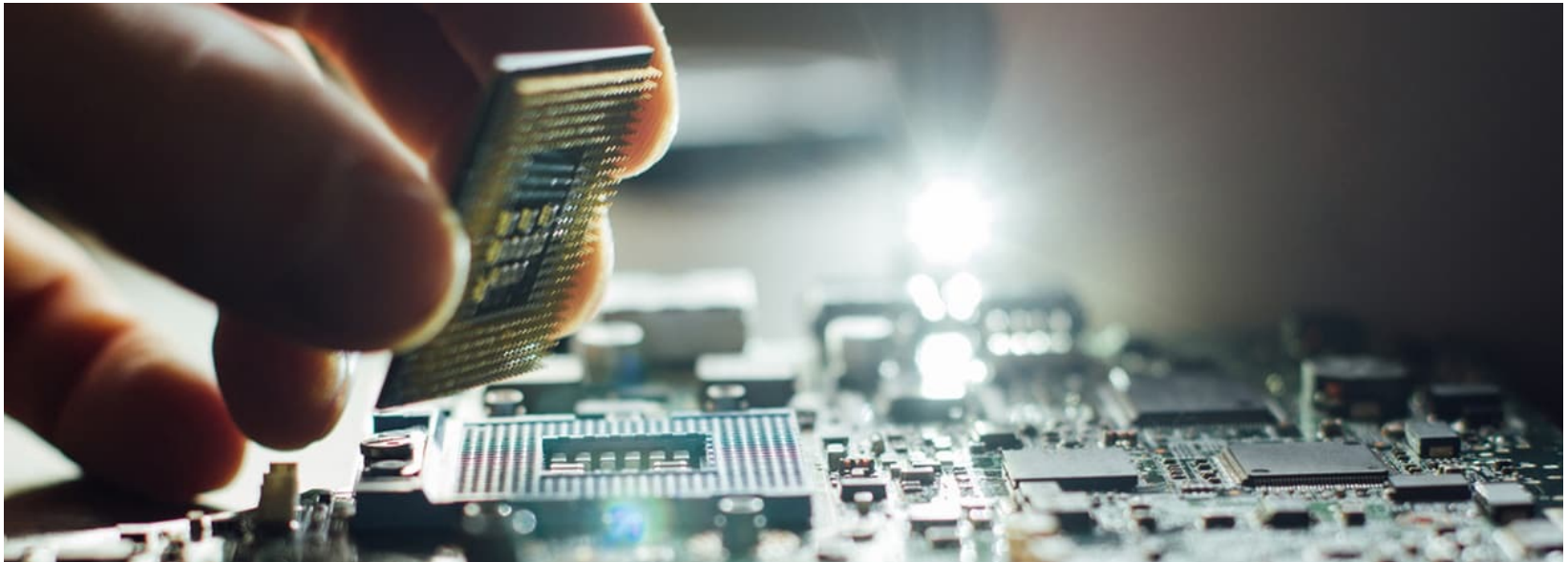
## ▶ **CPU frequency**: the frequency of clock in CPU

- Not same as “instructions per second”, as some processors run multiple instructions per cycle
- One of many factors affecting the CPU speed
  - Within same family of CPUs, an indicator of CPU speed

# CPU

---

- Consists of a single, or small number of, chip(s) made of plastic, metal and mostly silicon
- Composed of several million **transistors with** enormously complicated wiring
- The design of CPU itself is not possible without CAD software





# Major Hardware Components: **Memory**

---

- ▶ RAM (Random Access Memory): read-write memory, **volatile/temporary storage**
  - ▶ Holds data and programs that CPU is using, and save results of programs.
  - ▶ Often called **main memory, primary memory**
  - ▶ ROM (Read Only Memory): contains certain programs that must always be present
- ▶ Secondary storage (e.g., a hard drive) provides ***persistent storage***
  - ▶ **Stores programs or data not currently being used by other units** on *secondary storage devices* (like hard disk and floppy disks)

# Major Hardware Components: Memory

---

- ▶ “4 gigabytes (4GB) of RAM”
  - ▶ 0/1 – one bit, 8 bits = 1 byte
  - ▶ 1024 bytes = 1 kilobyte ( $\sim 10^3$  bytes), 1KB
  - ▶  $1024^2$  bytes = 1 megabyte ( $\sim 10^6$  byte), 1MB
  - ▶  $1024^3$  bytes = 1 gigabyte ( $\sim 10^9$  byte), 1GB
  - ▶  $2^{40}$  or  $1024^4$  bytes = 1 terabyte ( $\sim 10^{12}$  byte)

# Peripherals (Input/Output devices)

---

- ▶ Allow the computer to interact with user and other computers:
  - ▶ Output: Display, Printer, Speakers
  - ▶ Input: Mouse, Keyboard
  - ▶ Input/Output: Network card, Modem

# Types of Programs

---

- ▶ **Application programs:** programs that people use to get their work done
  - Word Processor, Web Browsers, etc.
  - Media Player, Calculator, Notepad, ...
- ▶ **System programs** keep all hardware and software running together smoothly.
  - **Operating System:** software system developed to make using computers more convenient: **Windows, Unix, Linux, MacOS, Android, ...**

# So what is programming?

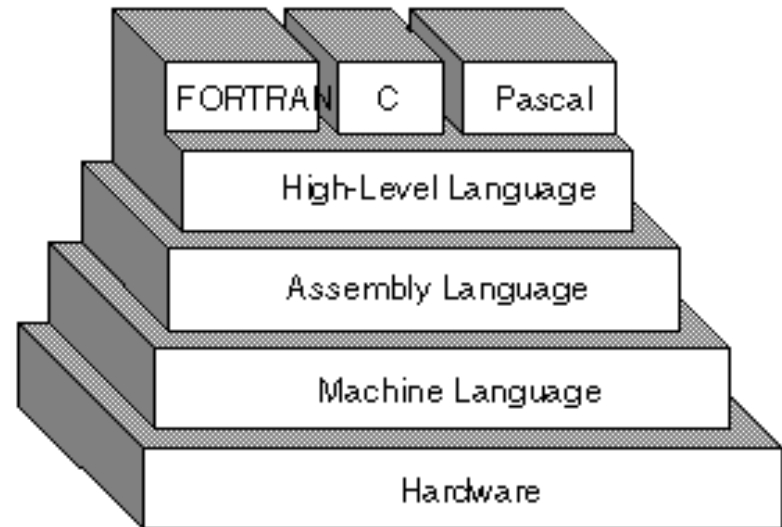
---

- ▶ **Specifying** the structure and behavior of a program via a programming language
- ▶ **Testing** that the program performs its task correctly and with acceptable performance
  - ▶ Never forget to check that “it” works

# Types of Programming Languages

---

- Three general types of programming languages
  - Machine languages
    - machine **dependent**
  - Assembly languages
    - machine **dependent**
  - High-level languages
    - most are **portable**
    - Specific languages include C, C++, Python, Go, and Java



# Programming language: Machine Code

---

- **Machine instructions:** extremely primitive, e.g.
  - Move memory contents to a register
  - Subtract 100 from a register
  - If result is positive, go (jump) to another instruction
- Each type/family of processor has its own set of machine instructions
  - CPU: fetch instructions in machine language (from main memory) and executes them
- Machine instructions are encoded as numbers:  
161 40000 45 100 127 11280
- writing **numeric** codes manually is tedious and error prone
  - thousands of instructions for a simple program

# Programming language: Assembly language

---

- **Assembly language**
  - assigns short names to commands
  - Can give names to memory locations
  - e.g.
    - `mov 40000, %eax sub 100, %eax jg 11280`
- Makes reading easier for humans
- Translated into machine instructions by the **assembler**, a **system program**
- Still processor dependent
- Still a great many instructions



# Higher-level programming languages

---

- ▶ Easiest for humans to read and write:
  - ▶ `if( int_rate > 100 ) cout << "Interest rate error";`
- ▶ Independent of the underlying hardware
  - ▶ You can write a program in C++, and it can work in different Unix machine ...
- ▶ Translated by *compilers* into machine instructions
  - Very sophisticated programs
  - Translate logical statements into sequences of computations  
161 4000 45 100 127 11280
  - Find memory locations for all variable names
- ▶ Much stricter than spoken languages
  - ▶ Compilers don't like to *guess*

# Evolution of C++

---

- Many languages are created with a specific purpose
  - database processing
    - Scripting languages: PERL, Shell, Ruby, R, Python
  - multimedia processing
- General purpose languages can be used for any task
  - C: developed to be translated efficiently into fast machine code, with minimal housekeeping overhead
  - C++: adds "object oriented programming" aspects to C
  - As of 2022 C++ ranked fourth on [TIOBE index](#), a measure of the popularity of programming languages, after [Python](#), [C](#) and [Java](#)

# History of C

---

- Initially designed in 1972 (Kernighan & Ritchie)
- Features were added in response to perceived shortcomings
- Resulted in different dialects
  - Bad for portability
- 1989 — ANSI standard of C completed

# History of C++

---

- ▶ 1979 — Bjarne Stroustrup of AT&T adds object oriented features to C, called *C with Classes*
- ▶ 1985 — rename to C++
- ▶ 1998 — ISO C++ standard published, i.e., C++98
- ▶ 2003 — minor revision C++03.
- ▶ 2011 — major revision C++11 released, adding numerous new features, enlarging the standard library further
- ▶ 2014: a minor revision C++14
- ▶ 2017: major revision C++17
- ▶ 2020: C++20

# Why C++?

---

- C++ is the language that most directly allows you to express ideas from the largest number of application areas
- C++ is the most widely used language in engineering areas
  - Mars rovers, animation, graphics, Photoshop, GUI, OS, compilers, slides, chip design, chip manufacturing, semiconductor tools, etc.
  - <http://www.research.att.com/~bs/applications.html>
- C++ is available on almost all kinds of computers  
Programming concepts that you learn using C++ can be used fairly directly in other languages
  - Including C, Java, C#, and (less directly) Fortran

# In this course

- Teach/learn
  - Fundamental programming concepts
  - Key useful techniques
  - Basic Standard C++ facilities
- After the course, you'll be able to
  - Write small colloquial C++ programs
  - Read much larger programs
  - Learn the basics of many other languages by yourself
  - Proceed with an “advanced” C++ programming course
- After the course, you will **not** (yet) be
  - An expert programmer
  - A C++ language expert
  - An expert user of advanced libraries

# Our programming environment

Introduction to Unix

# Our Programming Environment

---

- ▶ A Linux server
  - ▶ User name and password
  - ▶ Linux server: storm.cis.fordham.edu
  - ▶ Shared by many students, faculty members
- ▶ We access this server using **command line interface**, using
  - ▶ Terminal programs on Mac, Linux
  - ▶ PuTTY, a telnet/ssh client for Windows
    - ▶ a free and open source terminal emulator application a window in your desktop that works like old time terminal

A screenshot of a terminal window titled 'james@volcano ~'. The window shows a login process where the user 'james' has entered their password. The system is 'Linux volcano 2.6.22-14-server #1 SMP Sun Oct 14 23:34:23 GMT 2007 i686'. It displays the Ubuntu logo and a message about the free software included with the system. The prompt 'james@volcano:~>' is visible at the bottom with a green cursor.

```
james@volcano ~
login as: james
james@shell's password:
Linux volcano 2.6.22-14-server #1 SMP Sun Oct 14 23:34:23 GMT 2007 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

james@volcano:~>
```



# Your first encounter: shell

---

- **Shell:** a special-purpose program, **command line interpreter**, read commands typed by a user and execute programs in response to entered commands
- Many different shells:
  - Bourne Shell (sh): oldest,
  - C Shell (csh):
  - Korn Shell (ksh):
  - Bourne again Shell (bash)
- ▶ To change your login shell, use command
  - ▶ chsh

# Shell: interactive mode

---

- A shell session (a dialog between user and shell)
  1. Displays a **prompt** character, and waits for user to type in a **command line**
    - ▶ Prompt depends on shell: sh, ksh, bash: \$ csh: % tcsh: >
    - ▶ May be customized (with current directory, host, ...)
  2. On input of a **command line**, shell extracts **command name and arguments**, searches for the program, and runs it.
  3. When program finishes, shell continues to step 1
  4. The loop continues until user types “exit” or “ctrl-d” to end
- ▶ Note: Log out from storm server when you are done!

# UNIX command line

---

- Command name and arguments:

command [ [ - ] option (s) ] [ option argument (s) ] [ command argument (s) ]

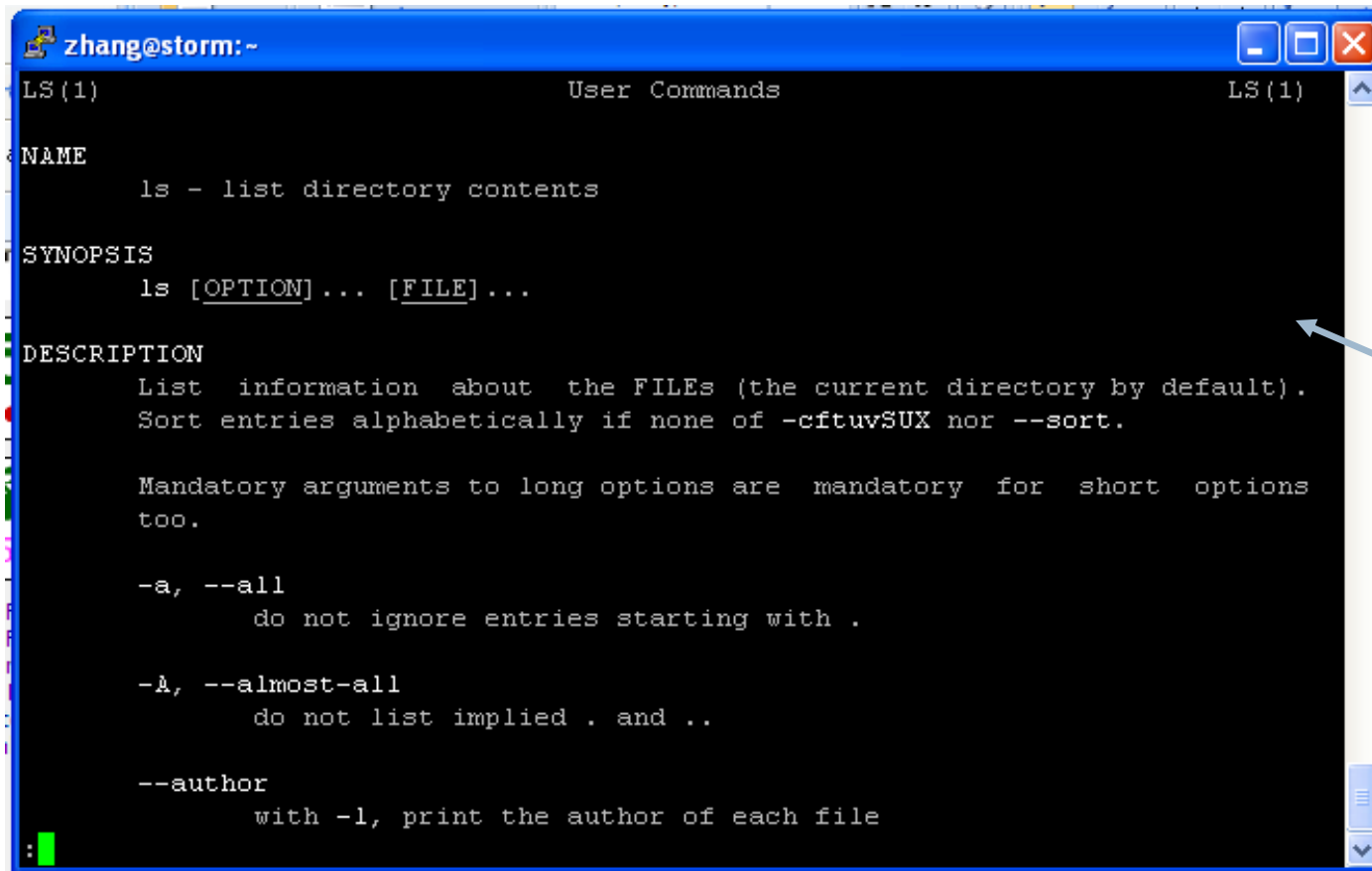
- Command **arguments** are mostly file or directory names

- `cp prog1.cpp prog1.cpp.bak`

- **Options**: used to control behavior of the command

- `head -20 lab1.cpp`
- `wc -w lab2.cpp` // count how many words
- Some options come with **option argument**
  - `sort -k 1 data.txt`
  - // use the first column of data.txt as the key to sort

# The most important command !!!



```
zhang@storm:~  
LS(1) User Commands LS(1)  
  
NAME  
    ls - list directory contents  
  
SYNOPSIS  
    ls [OPTION]... [FILE]...  
  
DESCRIPTION  
    List information about the FILES (the current directory by default).  
    Sort entries alphabetically if none of -cftuvSUX nor --sort.  
  
    Mandatory arguments to long options are mandatory for short options  
    too.  
  
    -a, --all  
        do not ignore entries starting with .  
  
    -A, --almost-all  
        do not list implied . and ..  
  
    --author  
        with -l, print the author of each file  
  
:
```

man ls

- man: displaying online manuals
  - Press **q** to quit, **space** to scroll down, **arrow** keys to roll up/down

# Correcting type mistakes

---

- Shell starts to parse command line only when **Enter** key is pressed
- Delete the whole line (line-kill): **C-u**
- Erase a character: **C-h** or backspace key
- Many more fancy functionalities:
  - **Auto-completion**: press **Tab** key to ask shell to auto-complete command, or path name
  - **History (repeat command)**: use arrow (up and down) keys to navigate past commands
  - ...

# Unix Files

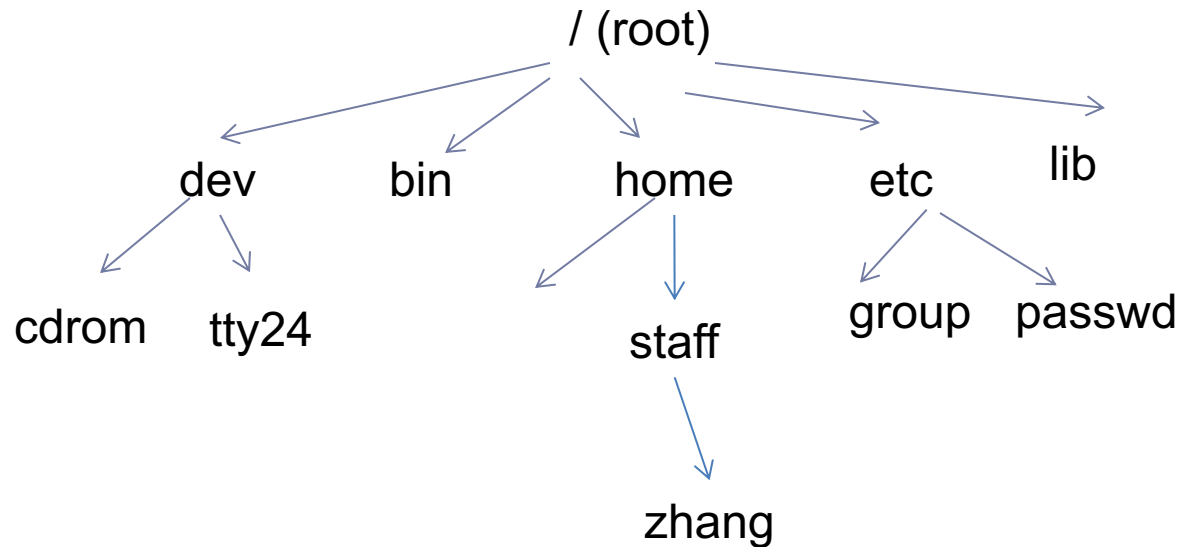
---

- ▶ **Files: store information**
  - ▶ a sequence of 0 or more bytes containing arbitrary information
- ▶ **filename**
  - ▶ Case matters, no longer limited to 14 chars
  - ▶ Special characters such as -, spaces are allowed, but should be avoided
  - ▶ Dot files are hidden, i.e., normally not listed by command `ls`
- ▶ **File type**
  - ▶ Text (ASCII) file (such as your C/C++ source code)
  - ▶ Executable file (commands)
  - ▶ A link to other files, ...

# Hierarchical file system

---

- **Directory:** a file that can hold other files
- ▶ **Advantages of hierarchical file system:**
  - Files can have same names, as long as they are under different directories
  - Easier for protection
  - Organized files



# Home directory

---

- ▶ Every user has a **home directory** created for him/her
  - ▶ When you log in, you are in your home directory
  - ▶ In home directory, a user usually has permission to create files/directories, remove files ..
  - ▶ ~ to refer to current user's home directory
  - ▶ ~username to refer to username's home directory



# Getting around in the file system

---

- To create a subdirectory:
  - mkdir [option]... directory...
  - cd
  - mkdir csI
  - cd csI
  - mkdir labI
- To remove a directory:
  - rmdir [option]... directory...
  - Report failure if directory is not empty
    - Can use rm -rf to remove non-empty directory

# Getting around in the file system

---

- ▶ **ls: list directory contents**

- ▶ `ls [OPTION] ... [FILE]`

ls: list files/directories under current directory

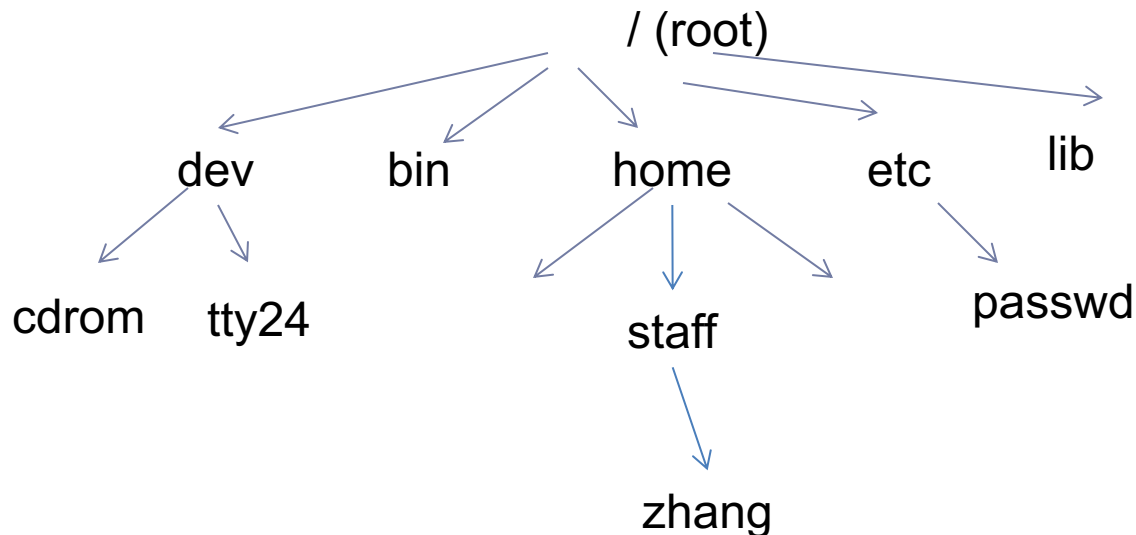
ls -l: long listing,

```
[zhang@storm CISC1600]$ ls -l
```

```
total 59180
```

```
-rw-r--r-- 1 zhang staff 509952 Sep  7 13:02 3_types.ppt
-rw-r--r-- 1 zhang staff 593408 Sep 14 23:38 4_computation.ppt
-rw-r--r-- 1 zhang staff 1297 Sep  2 12:18 account.html
-rw-r--r-- 1 zhang staff 3304448 Nov  7 18:24 ArrayVector1.ppt
drwxr-xr-x 2 zhang staff 4096 Dec  8 22:36 Codes
```

# Absolute pathname, path



- **Pathname** of a file/directory: location of file/directory in the file system
    - How do you tell other where your prog. is located ?
  - **Absolute pathname**: path name specified relative to root, i.e., starting with the root (/)
    - e.g., /home/staff/zhang
- 
- 35 ■ What's the absolute pathname for the "passwd" file?

# Current directory & Relative Pathname

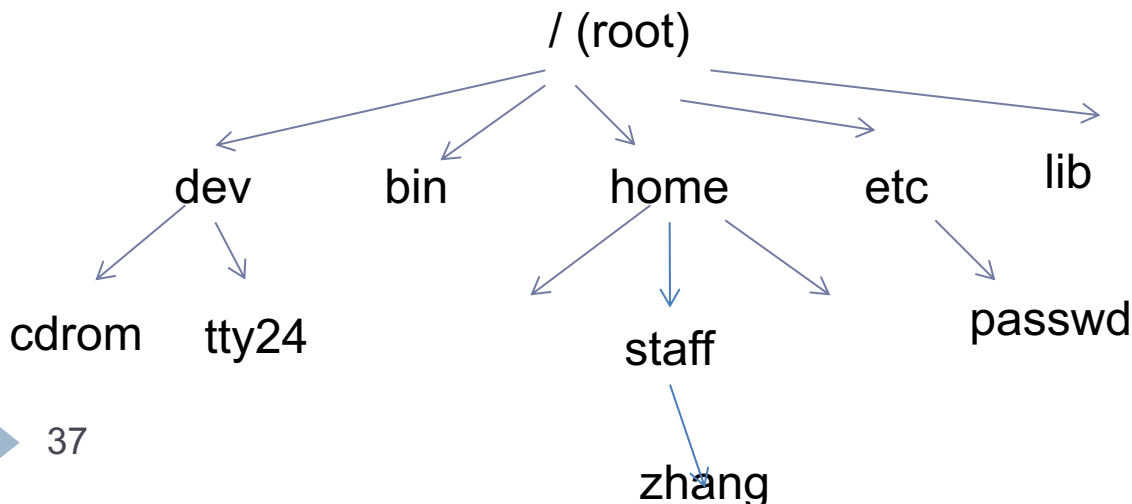
---

- ▶ Tiring to specify **absolute pathname** each time
- ▶ To make life easier: **working directory**
  - ▶ User can move around the file system, shell remembers where the user is (i.e., current directory)
- ▶ To check your current directory, use command:
  - ▶ `pwd`

# Relative pathname

---

- ▶ **Absolute pathname:** specified relative to root
- ▶ **Relative pathname:** specified relative to current directory
  - ▶ . (current directory), .. (parent directory, one level up)
  - ▶ If current directory is at /home/staff/zhang, what is the relative pathname of the file passwd?
    - ▶ ../../etc/passwd: go one level up, go one level up, go one level up, go to etc, passwd is there



# Relative pathname

---

- ▶ For all commands that take file/directory name as arguments, you can use pathnames of the file/directory
- ▶ Example:
  - ▶ `cd /home/staff/zhang/public_html`
  - ▶ `pico cs1600/index.html`
  - ▶ `cd ..` (go up one level to parent directory)
  - ▶ `cp ../prog2.cpp prog2.cpp`

# File Viewing Commands

---

- cat: concatenate files and display on standard output (i.e., the terminal window)
  - cat [option] ... [file] ...
  - cat proj1.cpp
  - cat proj1.cpp proj2.cpp
  - cat -n proj1.cpp // display the file with line #
- more: file perusal filter (i.e., displaying file one screen at a time)
  - more proj1.cpp
- less: similar to more
- head, tail: display the beginning or ending lines of a file
  - tail -f output // display the file, append more lines as the file grows

*[ ] means the argument is optional  
... means there can be multiple arguments of this type*

# File manipulation commands

---

- **rm**: remove one or multiple files or directories
  - **rm [option] ... FILE ...**
  - **rm temp**
  - **rm temp1 temp2**
- **rm -r**: remove directories and their sub-dirs recursively
- **rm -i** : confirm with user before removing files



# File manipulation commands (2)

---

- ▶ **cp: copy file or directory**
  - ▶ `cp [OPTION] SOURCE DESTINATION`
- ▶ To make a backup copy of your program before dramatic change
  - ▶ `cp proj1.cpp proj1.cpp.bak`
- ▶ To make a backup copy of a whole directory
  - ▶ `cp -r lab1_dir lab1_dir_backup`
  - ▶ `-R, -r, --recursive`: copy directories recursively

# File manipulation commands (3)

---

- ▶ **mv: move (rename) files/directories**
  - ▶ mv [OPTION] SOURCE DEST
    - ▶ Rename SOURCE to DEST
    - ▶ mv proj1.cpp lab1.cpp
  - ▶ mv [OPTION]... SOURCE... DIRECTORY
    - ▶ Move SOURCE to DIRECTORY
    - ▶ mv lab1.cpp lab2.cpp CISC3130

# First C++ Program

# Edit, Compile, and Execute C++ Program

---

- ▶ **Edit** the program using C++, save it in a file with suffix .cc, or .cpp, e.g., lab1.cc
  - ▶ Editor we use: **emacs**
- ▶ **Compile** the program (actually multiple steps involved: preprocess, compile and link): type following command **in command line**:
  - ▶ `g++ lab1.cc`
  - ▶ `g++ lab1.cc -o HelloWorld`
- ▶ **Run** the executable file
  - ▶ `./a.out`
  - ▶ `./HelloWorld`

# lab1.cc

---

// Text-printing program.

#include <iostream> // allows program to output data to  
// the screen

// function main begins program execution

int main()

{

std::cout << "Welcome to C++!\n"; // display message

return 0; // indicate that program ended successfully

} // end function main

# Important Parts of a C++ program

---

- ▶ **Comments:** `//`, `/* ... */`
- ▶ **Preprocessor directives :** `#include`
- ▶ **Function `main`**
  - ▶ Body of the function
  - ▶ Return statement
  - ▶ Other statements
- ▶ **White spaces**
  - ▶ Blank lines, space characters and tabs
  - ▶ Delimiter, used to make programs easier to read
  - ▶ Extra spaces are ignored by the compiler

# Comments

---

- Explain programs to other programmers
  - Improve program readability
- Ignored by compiler => does not change behavior of a program
- Single-line comment
  - Begin with `//`
- Multi-line comment
  - Start with `/*`
  - End with `*/`

# Preprocessor Directives

---

- ▶ Processed by **preprocessor** before compiling
- ▶ Begin with **#**
- ▶ Example
  - ▶ `#include <iostream>`
    - ▶ Tells preprocessor to include the **input/output stream header file** `<iostream>`



# Function main

---

- ▶ A part of every C++ program
  - ▶ Every program has **exactly one** main function
  - ▶ **main** is a **keyword**.
    - ▶ **keyword** : a word in code that is reserved by C++ for a specific use.
- ▶ **Header of function** `main : int main( )`
  - ▶ `main( )` is a function that takes no arguments ( )  
and returns an `int` (integer value) to indicate success or failure
- ▶ **Body of a function** is delimited by braces ( { } )

# Statements

---

- ▶ Lines in your program that instruct the computer to **perform an action**
- ▶ All statements end with a semicolon (;)
- ▶ Examples :
  - ▶ `return 0;`
  - ▶ `std::cout << "Hello world!\n ";`

# return Statement

---

- ▶ One of several means to exit a function
- ▶ When used at the end of `main`
  - ▶ The value 0 indicates the program terminated successfully
  - ▶ Example
    - ▶ `return 0;`

# Output Statement (1)

---

- ▶ **std::cout << "Hello world!\n ";**
  - ▶ **std::cout**
    - ▶ Standard output stream object, something that represents your “standard” output device (normally linked to computer screen)
    - ▶ Defined in input/output stream **header file <iostream>**
    - ▶ We are using a name (**cout**) that belongs to “namespace” **std**.
  - ▶ **Stream insertion operator <<**
    - ▶ Value to right (right operand) inserted into left operand.

## Output Statement (2)

---

```
std::cout << "Hello world!\n";
```

- ▶ Quotes delimit a string literal
- ▶ **NOTE:** “smart” quotes “ ” will cause compiler problems.
- ▶ so make sure your quotes are of the style " "

# Output Statement (3)

---

- ▶ **Escape** character : backslash : "\"
- ▶ **Escape sequence** : a character preceded by backslash
  - ▶ Indicates “**special**” **character** output that cause some actions to the output
  - ▶ e.g.
    - ▶ **\n** : newline, move cursor to beginning of next line on the screen
    - ▶ **\t**: horizontal tab, move cursor to the next tab stop.

# “Hello World!”

---

```
// Text-printing program.
```

```
#include <iostream> // allows program to output data  
to                // the screen
```

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

```
int main()
```

```
{
```

```
    std::cout << "Welcome to C++!\n"; // display  
message
```

```
    return 0;    // indicate that program ended  
successfully
```

```
▶ } // end function main
```

# Hello, world!

---

- ▶ The first program help you get used to
  - Editor
  - Compiler
  - Program development environment
  - Program execution environment
- ▶ Type in the program **carefully**
  - After you get it to work, please make a few mistakes to see how the tools respond; for example
    - Forget the header
    - Forget to terminate the string
    - Misspell **return** (e.g. **retrun**)
    - Forget a semicolon
    - Forget { or }
    - ...



# Modifying Our First C++ Program

---

- ▶ **Print text on one line using multiple statements**
  - ▶ Each stream insertion resumes printing where the previous one stopped
  - ▶ Statements:

```
std::cout << "Welcome ";  
std::cout << "to C++!\n";
```

# Modifying Our First C++ Program

---

- ▶ **Print text on several lines using a single statement.**
  - ▶ Each newline escape sequence positions the cursor to the beginning of the next line
  - ▶ Two newline characters back to back outputs a blank line
  - ▶ Example statement :

```
std::cout << "welcome\n to\n\nC++!\n";
```

# Programming

---

- ▶ Programming is fundamentally simple
  - ▶ Just state what the machine is to do
- ▶ So why is programming hard?
  - ▶ We want “the machine” to do complex things
    - ▶ And computers are nitpicking, unforgiving, dumb beasts
  - ▶ The world is more complex than we’d like to believe
    - ▶ So we don’t always know the implications of what we want
  - ▶ “Programming is understanding”
    - ▶ When you can program a task, you understand it
    - ▶ When you program, you spend significant time trying to understand the task you want to automate

# Programming

---

- ▶ Except for the work you hand in as individual contributions, we ***strongly*** encourage you to collaborate and help each other
- ▶ If in doubt if a collaboration is legitimate: ask!
  - ▶ Don't claim to have written code that you copied from others
  - ▶ Don't give anyone else your code (to hand in for a grade)
  - ▶ When you rely on the work of others, explicitly list all of your sources – i.e. give credit to those who did the work
- ▶ Don't study alone when you don't have to
  - ▶ Form study groups
  - ▶ Do help each other (without plagiarizing)

# Comments / Suggestions

---

- ▶ Please mail questions, constructive comments and feedbacks to `xzhang@fordham.edu`
- ▶ On style, contents, detail, examples, clarity, conceptual problems, exercises, missing information, depth, etc.
- ▶ Course website
  - ▶ <http://storm.cis.fordham.edu/~zhang/cs1600>

# In Summary

---

- ▣ We learnt about Computer System Organization
- ▣ Hardware: CPU, Memory, Secondary Storage, Input and Output
- ▣ Software & Programming Languages
  - Application program vs System program
- ▣ Programming in C++
  - Different parts of C++ source code
  - Source code => Preprocessor, compiler, linker => Executable

# The next lecture

---

- ▶ Will talk about types, values, variables, declarations, simple input and output, very simple computations, and type safety.