Chapter 1

Introduction to Computers and C++ Programming

Prof. Chien-Nan (Jimmy) Liu
Dept. of Electrical Engineering
National Chiao-Tung Univ.

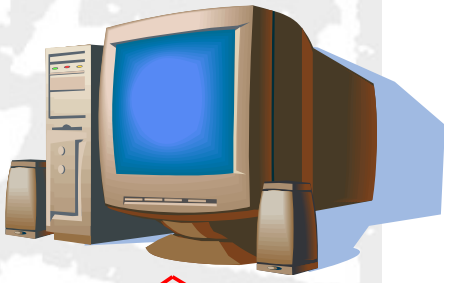
Tel: (03)5712121 ext:31211
E-mail: jimmyliu@nctu.edu.tw
http://mseda.ee.nctu.edu.tw/jimmyliu

Overview

- *1.1 Computer Systems*
- 1.2 Programming and Problem Solving
- 1.3 Introduction to C++
- 1.4 Testing and Debugging

Computer Systems

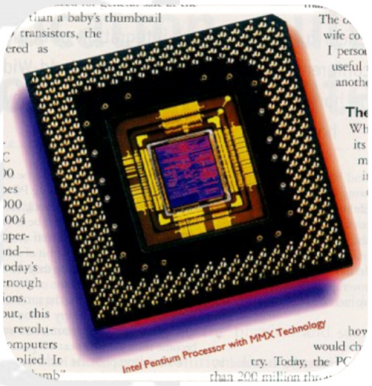
- A computer program is …
  - A set of instructions for a computer to follow



- Computer software is …
  - The collection of programs used by a computer
    - Editors
    - Translators
    - System Managers
    - …

```
main ()
{
   say hello;
   go to internet;
   goodbye;
}
```

Software



Hardware

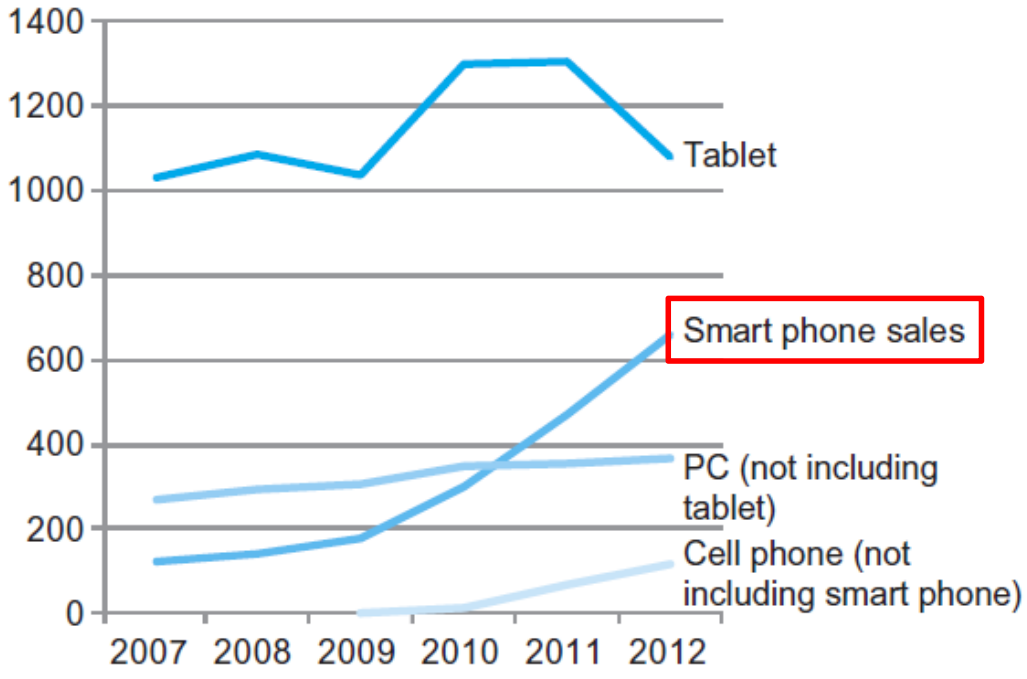
Classes of Computers (1/2)

- Personal computers (PC)
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
  - Deliver good performance to a ***single user*** at low cost
- Server computers
  - Oriented to carrying large workloads and serving ***many users*** through a network
  - High capacity, performance, reliability
  - Range from small servers to building sized

Classes of Computers (2/2)

- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
    ex: washing machines, cars, cell phones, PDA, …
  - Stringent power/performance/cost constraints

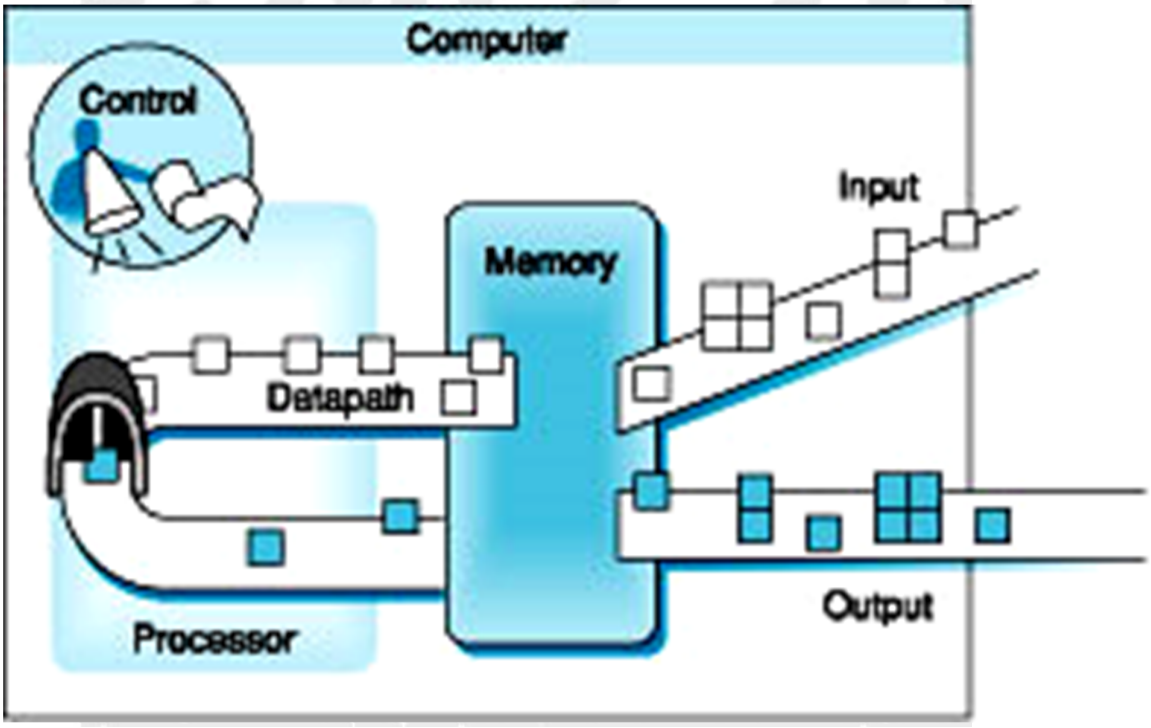
The PostPC Era (1/2)



The PostPC Era (2/2)

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

5 Classic Components of a Computer



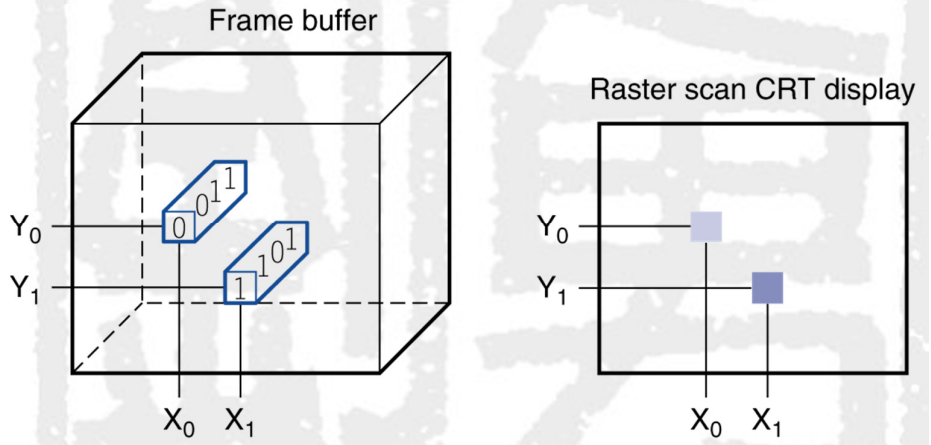
Anatomy of a Computer

Major Components

- Input devices : mouse, keyboard, …
- Output devices : monitor, speaker, …
  - Liquid Crystal Displays (LCD)
  - Touchscreen (Input+Output)
- Motherboard
- Integrated circuits or chips
  - Memory chips : Dynamic Random Access memory (DRAM), Static Random Access memory (SRAM)
  - PC chip set: graphic, sound, I/O controller
  - Central Processor Unit (CPU)
- Data storage
  - Primary memory (main memory) :DRAM, SRAM
  - Secondary memory: (Zip) floppy disk, hard disk, FLASH memory, optical Compact Disk (CD), DVD, …

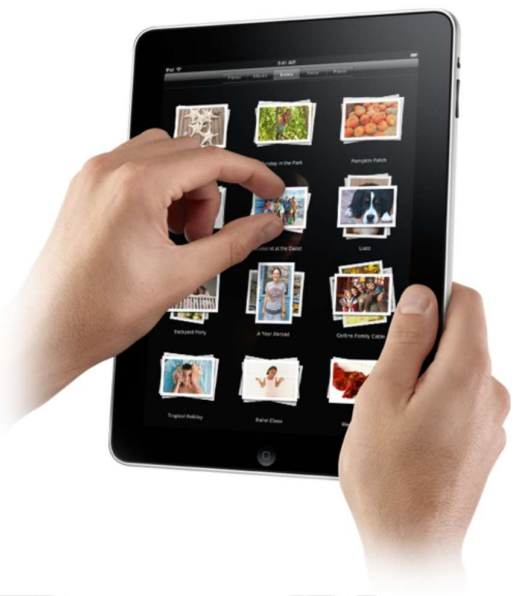
Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory

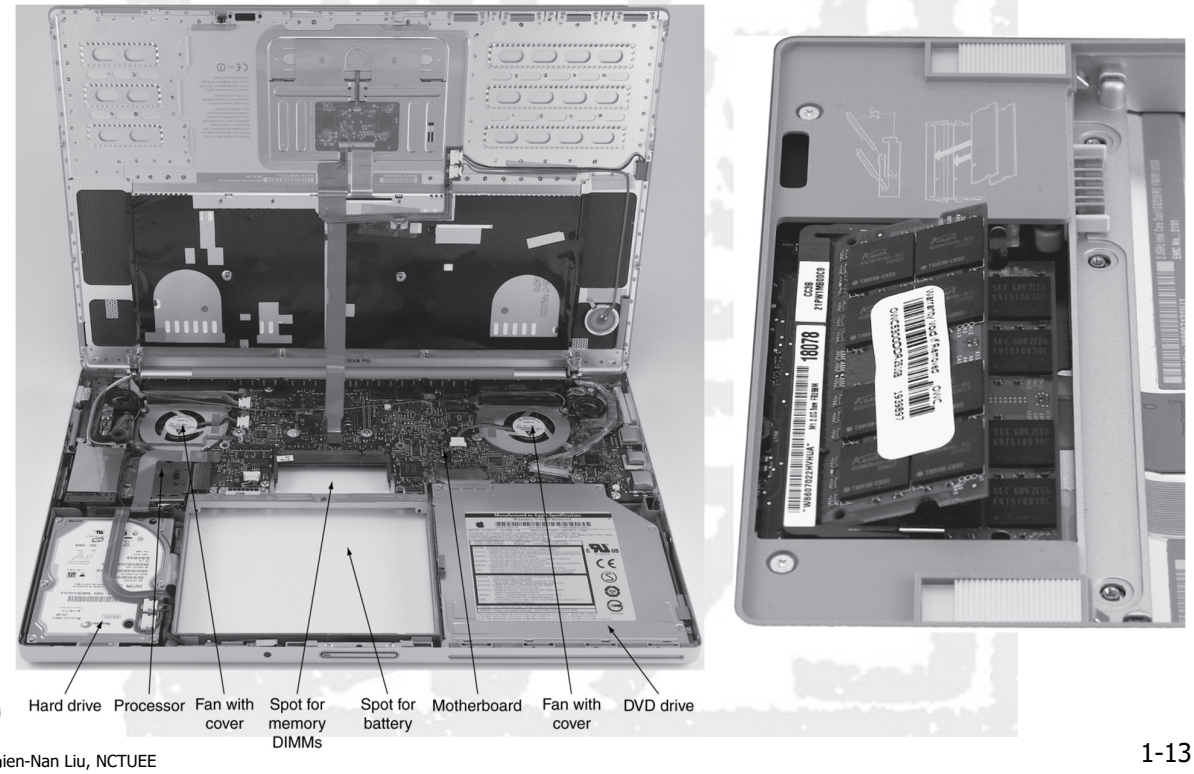


Touchscreen

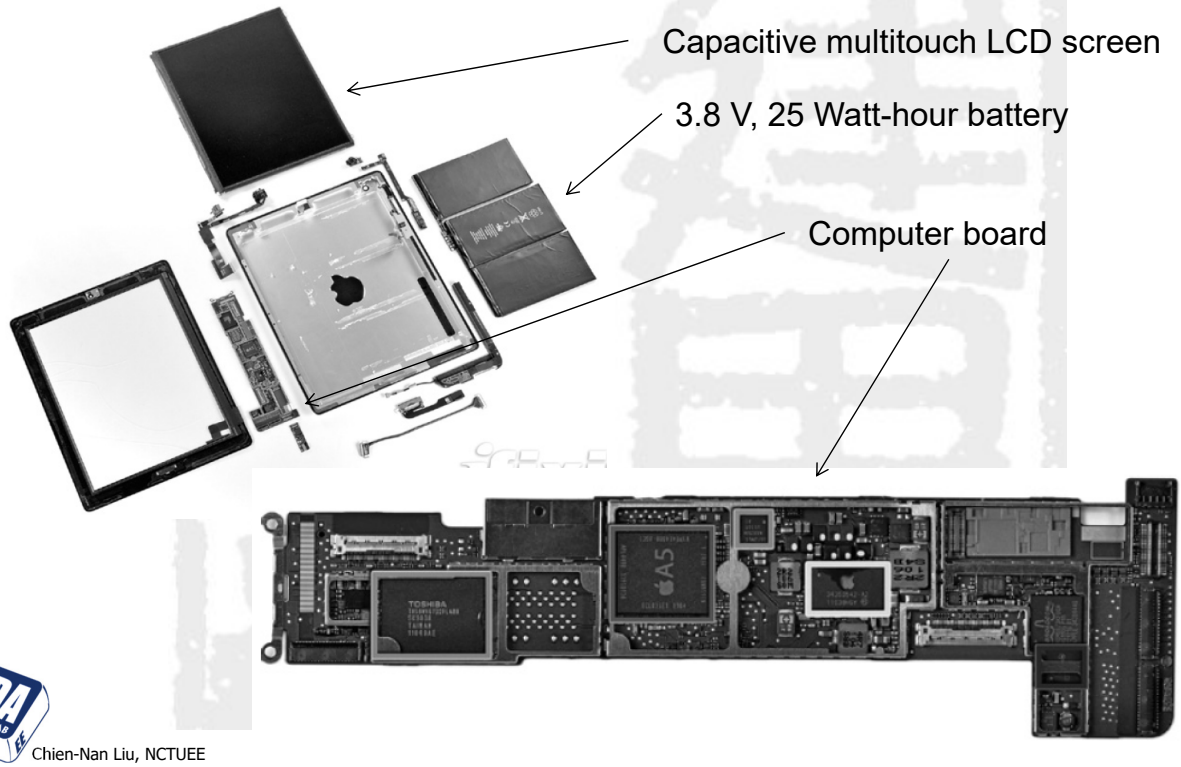
- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously



Opening the Box (Notebook)

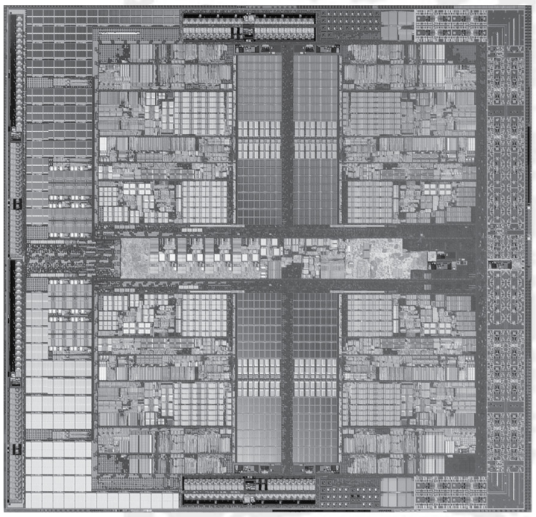


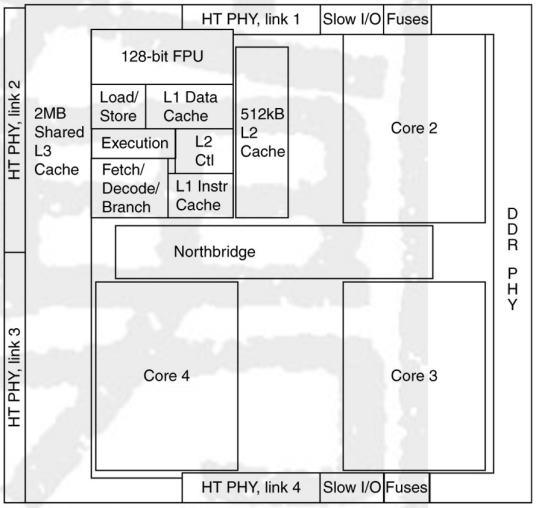
Opening the Box (Apple iPad)



Inside the Processor Chip

- Processor = datapath (execution) + control (commander)
  - Implemented using millions of transistors
  - Impossible to understand by looking at each transistor



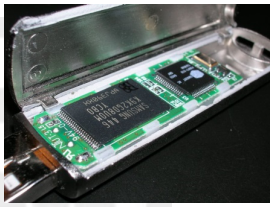


AMD Barcelona: 4 processor cores

Inside the Processor (Apple A5)

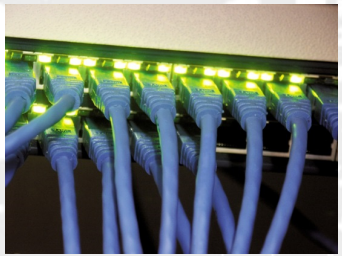
A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



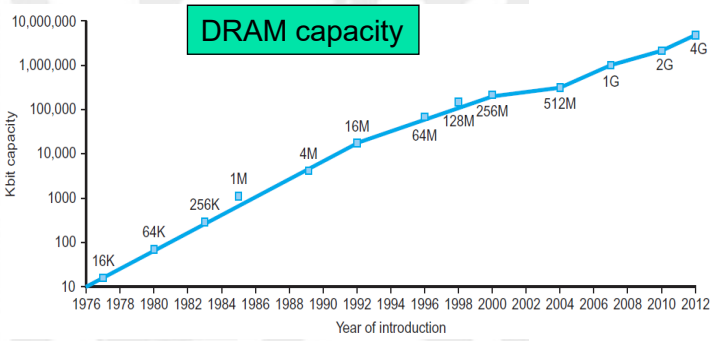
Networks

- ***Networks*** have become the backbone of current computer systems
  - Allow users to extend the power of computing
- Good for communication and resource sharing
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth, 4G, 5G, …



Technology Advance

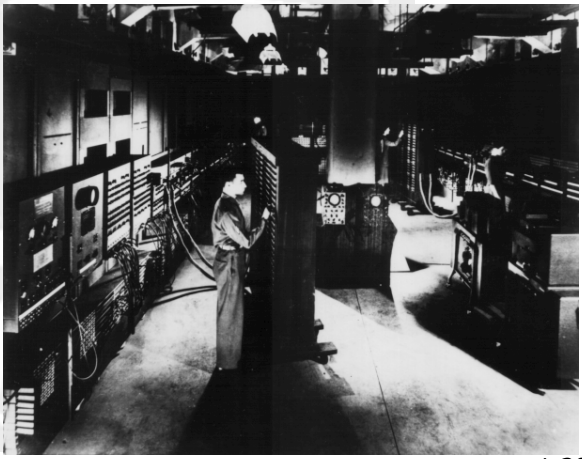
- Moore's Law: logic capacity doubles per IC every 18 months (1975)
- Increased capacity and performance with reduced cost



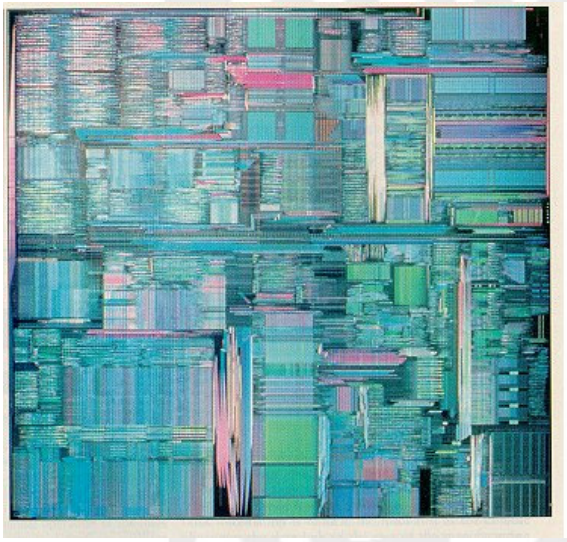
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Revolution of Computers

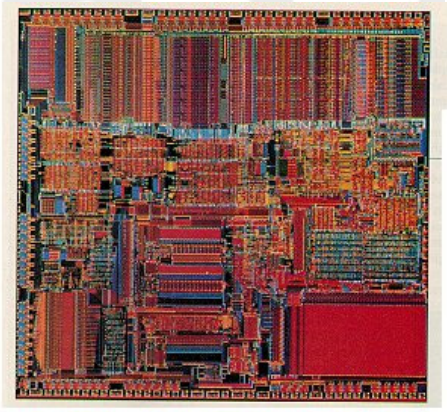
- ENIAC built in World War II (1946) was the first general purpose computer
  - Used for computing artillery firing tables
  - 80 feet long by 8.5 feet high and several feet wide
  - Each of the twenty 10 digit registers was 2 feet long
  - Used 18,000 vacuum tubes
  - Performed 1900 additions per second
- 1948: IBM604
  - 4000 machines were sold
  - The beginning of computer industry
- 1971: Intel 4004 CPU
  - Single-chip processor



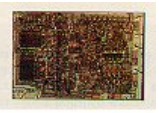
The Dies of Intel CPUs



Pentium Pro

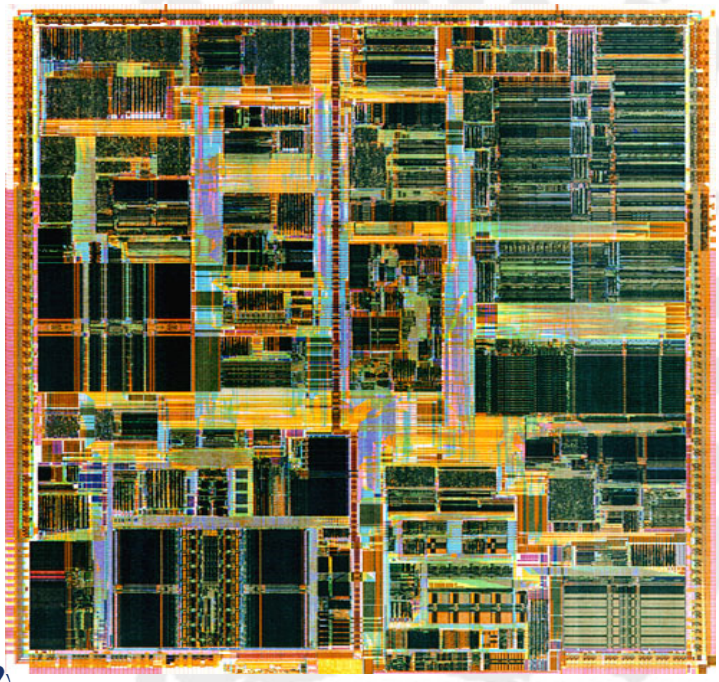


386



4004

Intel Pentium-IV Microprocessor

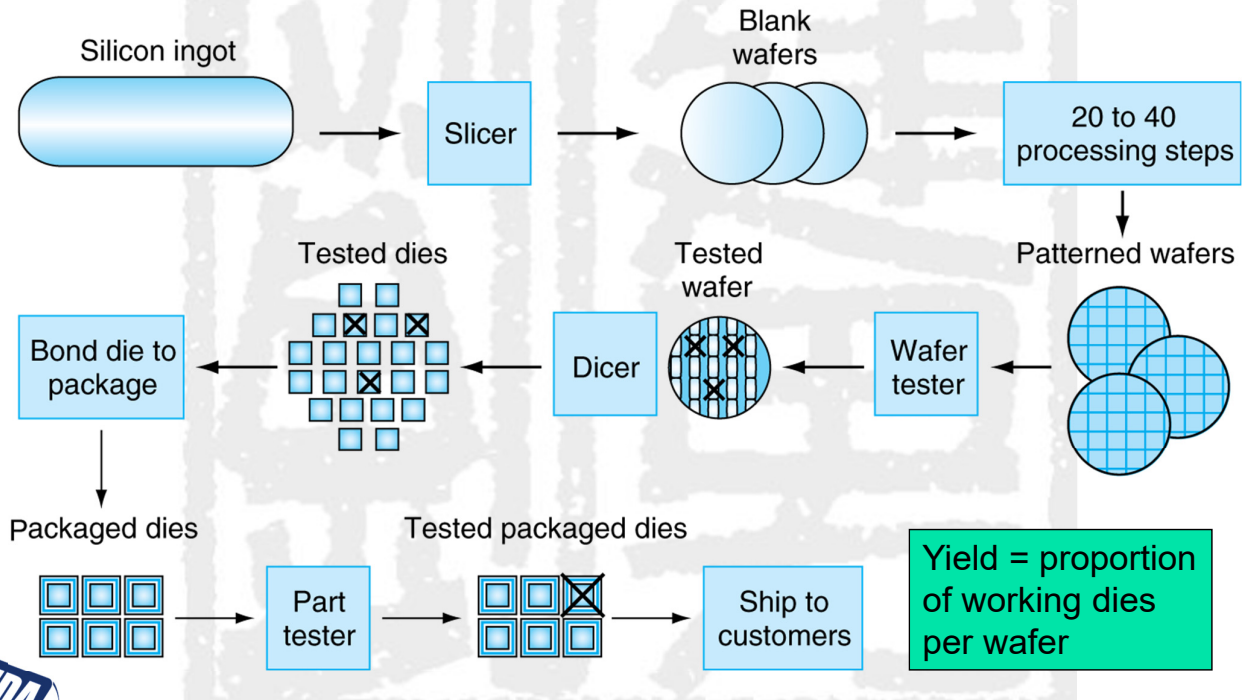


- 2000
- 8 M transistors
- 2800 MHz
- bus width: 64 bits
- Up to 1G Memory
- 512K L2 cache

Intel Core 2 Microprocessor

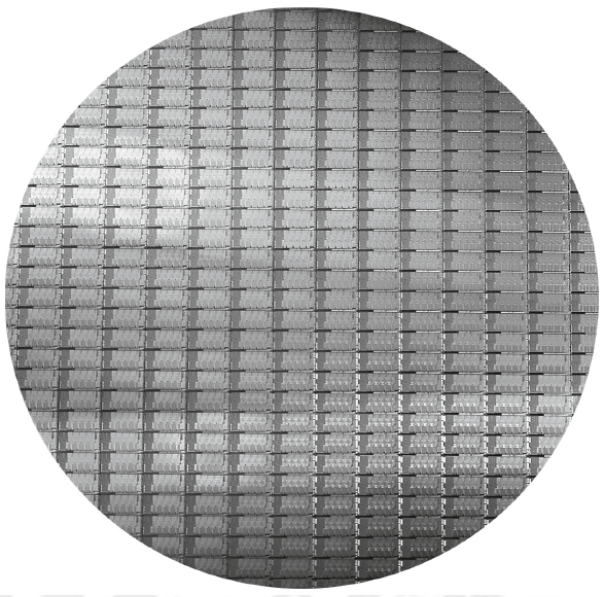
- 2006
- 65nm process
- 291 M transistors
- 3200 MHz
- 64-bit memory
- 4M L2 cache

Chip Manufacturing Process



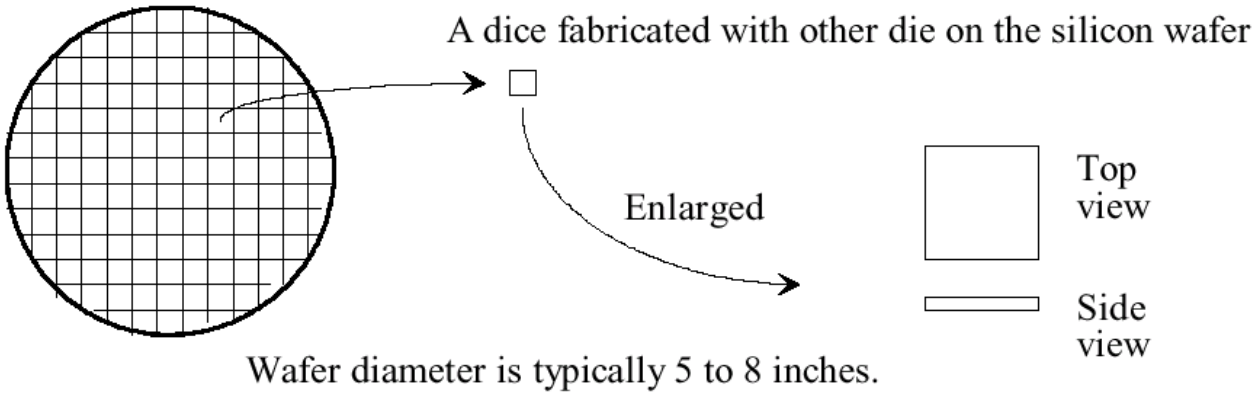
IC Fabrication

Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

Wafer and Dices

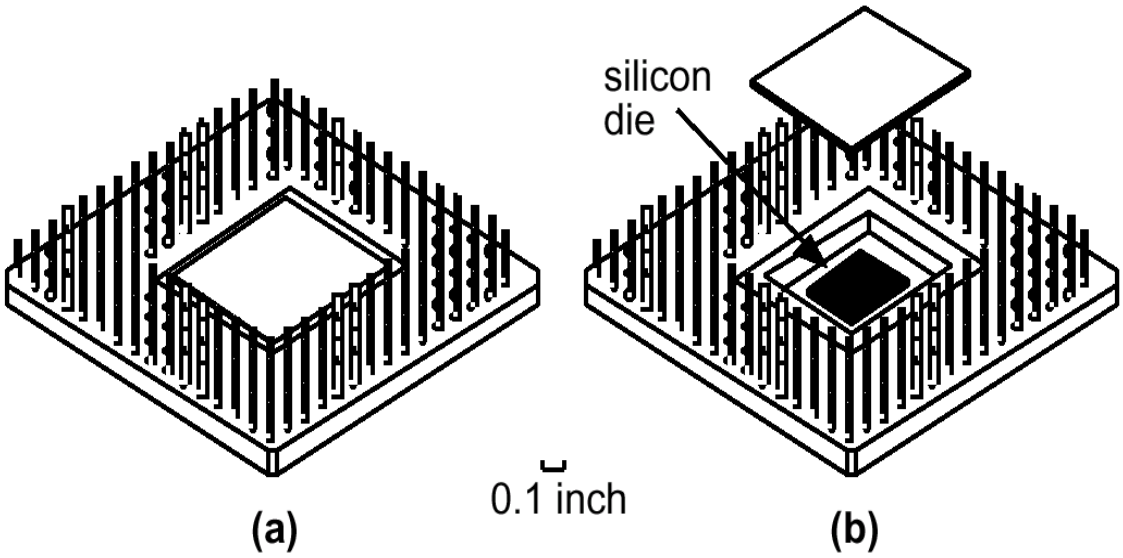


Source: http://cmosedu.com/cmos1/doslasi/doslasi.pdf

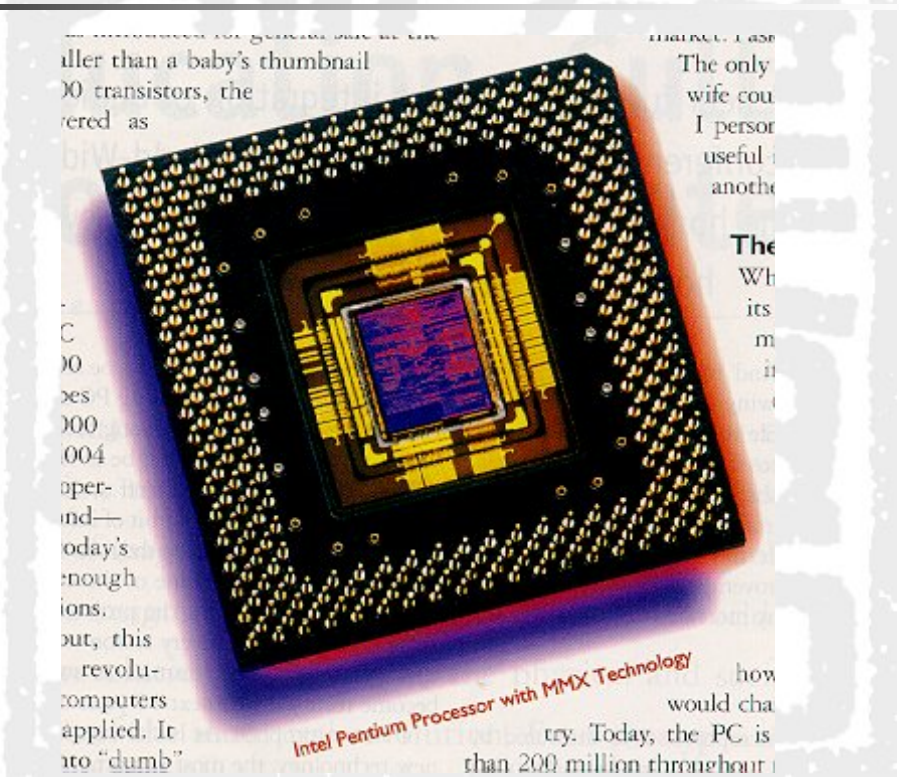
Sawing a Wafer into Chips

Figure 9-33 After testing and sawing, the individual chips are picked up by a robotic arm and placed in the package for die bonding. (Photograph courtesy of Intel Corp.)

IC and Die

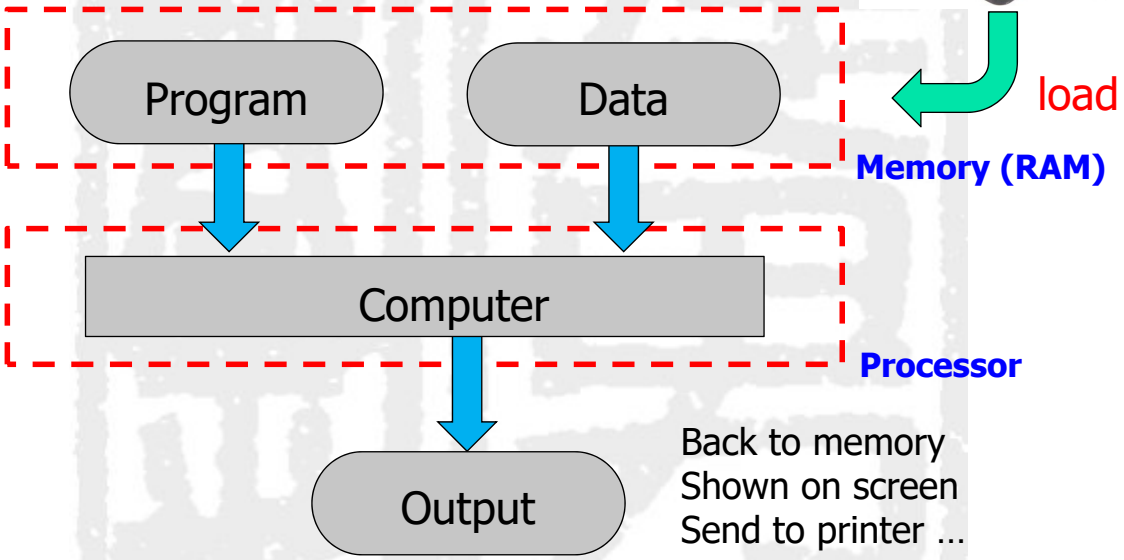


Pentium-MMX with PGA Packaging



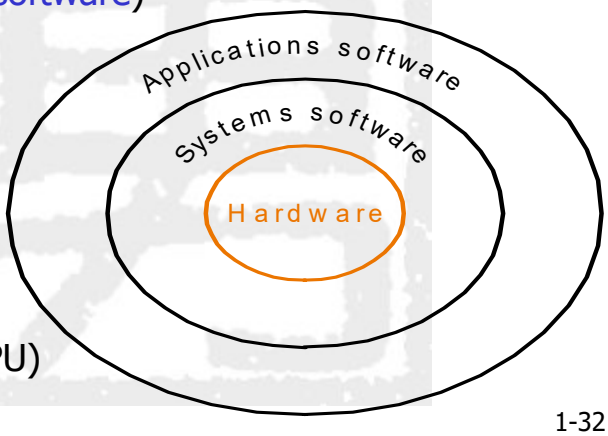
How Software is Executed?

Simple View of Running a Program



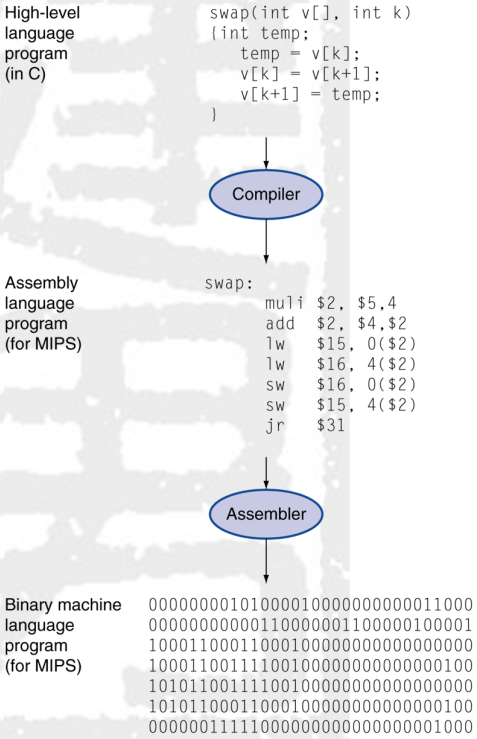
From High-level Language to Hardware

- High-level programming language
  - C, C++, Java, Python, …
  - Write down the concept of your programs (application software)
- Compiler (system software)
  - Translate high-level language into instructions
- Operating Systems (system software)
  - Handle basic I/O and allocate storage/memory
  - The bridge between programs and hardware
  - Windows, Linux, …
- Hardware
  - Execution units (mostly in CPU)

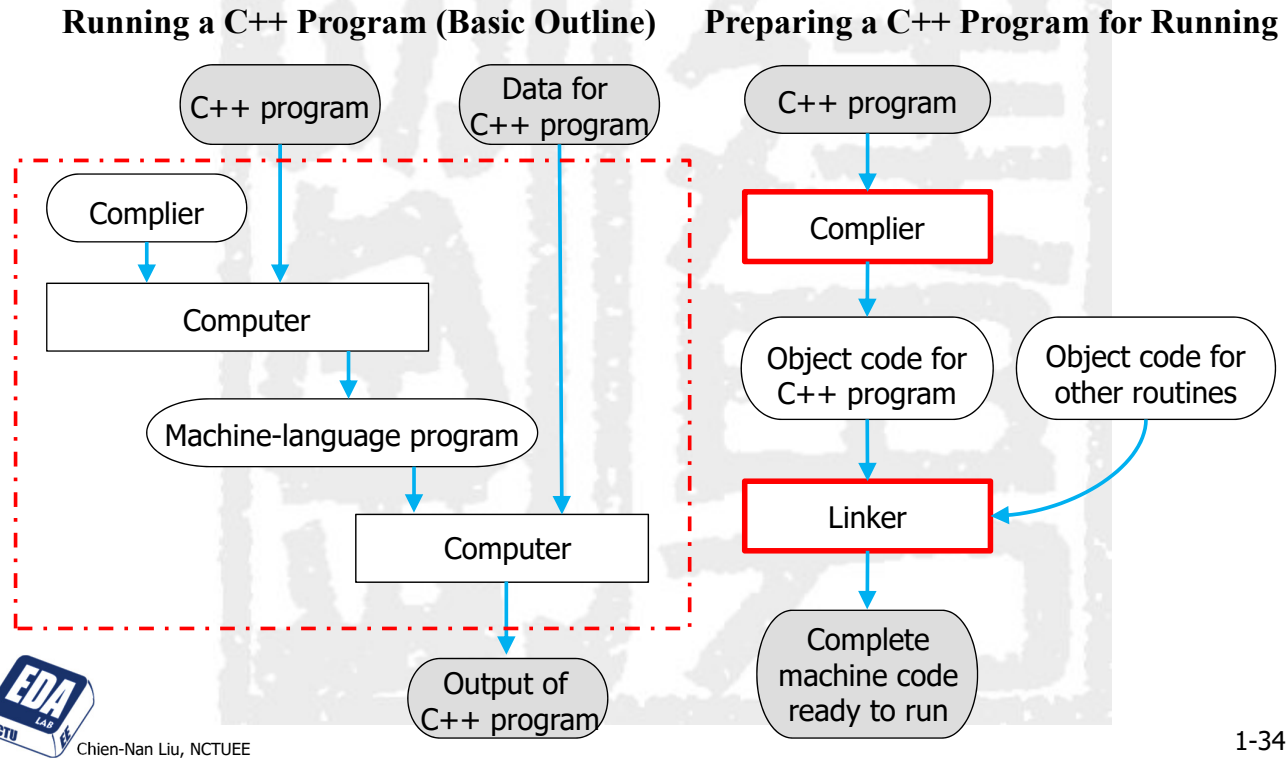


Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Prepare for Execution



Overview

- 1.1 Computer Systems
- *1.2 Programming and Problem Solving*
- 1.3 Introduction to C++
- 1.4 Testing and Debugging

Algorithms

- An algorithm is a sequence of precise instructions that leads to a solution
  - the actions to execute and
  - the order in which the actions execute
- Program: expressed an algorithm in a language the computer can understand
  - Problem solving phase (flow thinking)
  - Implementation phase (translation)

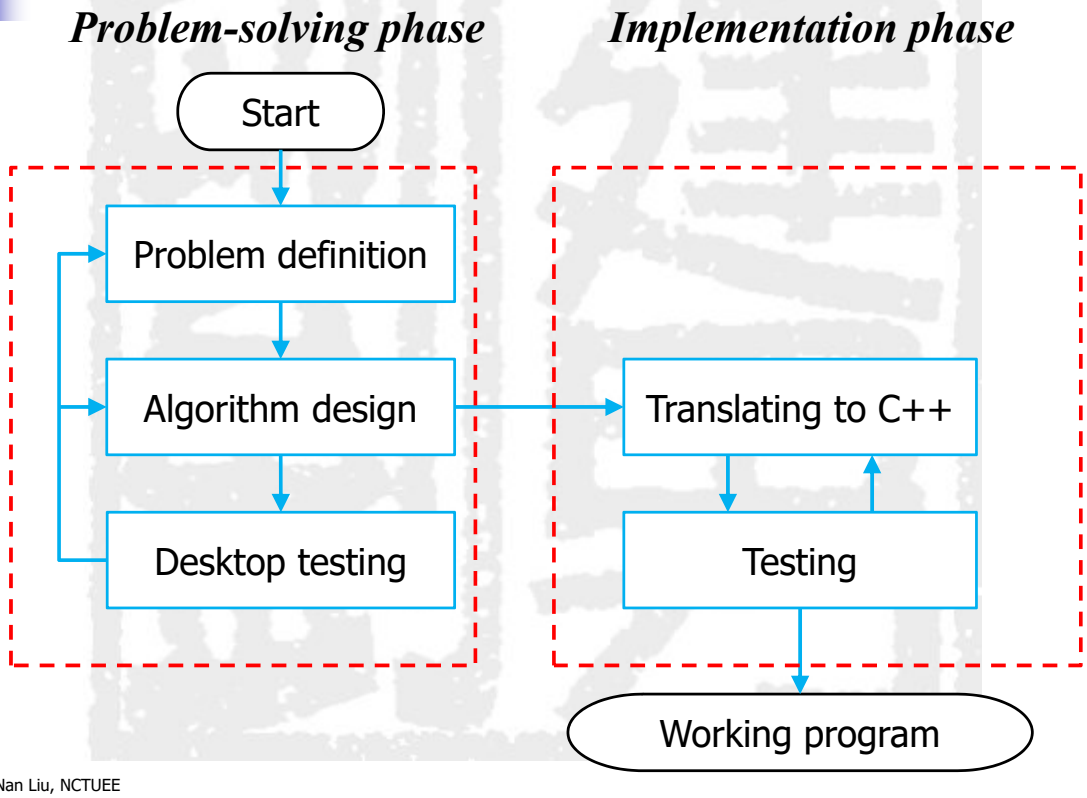
Problem Solving Phase

- Analyze the problem carefully to find:
  - What is the input?
  - What information is in the output?
  - How is the output organized? (format)
- Develop the algorithm before implementation
  - What if you solve this problem manually?? (idea)
  - Make sure this saves time in getting your program to run (analysis)
  - Test the algorithm for correctness on more cases

Implementation Phase

- Translate the algorithm into a programming language
  - Easier as you gain experience with the language
- Compile the source code
  - Locates errors in using the programming language
- Run the program on sample data
  - Verify correctness of results
- Results may require modification of the algorithm and program

Program Design Process



Object Oriented Programming

- Abbreviated as **OOP**
  - Used for many modern programs
- Program is viewed as interacting objects
  - Each object contains algorithms to describe its behavior
  - Program design phase focuses on how to use those objects to build the desired algorithm (LEGO?)
- Key features:
  - Encapsulation: information hiding (easy to use)
  - Inheritance: inherit characteristics from others (reuse)
  - Polymorphism: single name with multiple meanings (change mode automatically)

Overview

- 1.1 Computer Systems
- 1.2 Programming and Problem Solving
- *1.3 Introduction to C++*
- 1.4 Testing and Debugging

C++ History

- C was derived from the B language by Dennis Ritchie at AT&T Bell Labs in the 1970s.
  - Used to maintain UNIX systems
  - Today, most operating systems are written in C/C++
- C++ was developed by Bjarne Stroustrup at AT&T Bell Labs in the 1980s.
  - Overcame several shortcomings of C
  - Incorporated object oriented programming
  - C remains a subset of C++ (backward compatible)
- Why the '++' ?
  - ++ is an operator in C++ and results in a cute pun

C++11

- C++11 is the most recent version of the standard of the C++ programming language
  - Approved on August 12, 2011 by the International Organization for Standardization (ISO)
- C++11 language features are not supported by older compilers
  - Make sure the tool in hand support those new features
- Check the documentation with your compiler to determine if special steps are needed to compile C++11 programs
  - e.g. with g++, use extra flags of –std=c++11

A Simple C++ Program

```
#include <iostream>
using namespace std;

int main()
{
   int numberOfPods, peasPerPod, totalPeas;
   cout << "Press return after entering a number.\n";
   cout << "Enter the number of pods:\n";
   cin >> numberOfPods;
   cout << "Enter the number of peas in a pod:\n";
   cin >> peasPerPod;

   totalPeas = numberOfPods * peasPerPod;

   cout << "If you have ";
   cout << numberOfPods;
   cout << " pea pods\n";
   cout << "and ";
   cout << peasPerPod;
   cout << " peas in each pod, then\n";
   cout << "you have ";
   cout << totalPeas;
   cout << " peas in all the pods.\n";
   return 0;
}
```

Sample Dialogue

```
Press return after entering a number.
Enter the number of pods:
10
Enter the number of peas in a pod:
9
If you have 10 pea pods
and 9 peas in each pod, then
you have 90 peas in all the pods.
```

Program Layout

```
#include <iostream>
using namespace std;
int main()
{
    variable_Declartions

    Statement_1
    Statement_2
     ...
    Statement_Last

    return 0;
}
```

Often begin this way …

Often end this way …

Explanation of code (1/4)

- Include Directives -- #include <iostream>
  - Tells compiler where to find information about items used in the program
  - ***iostream*** is a library containing definitions of basic I/O functions (ex: ***cin*** and ***cout***)
- using namespace std;
  - Tells the compiler to use names in ***iostream*** in a "standard" way
- To begin the main function of the program

  int main() { … }
  - The unique start point for each program
- To end the main function -- return 0;
  - Each function ends with a return statement

Explanation of code (2/4)

- Variable declaration line

  int numberOfPods, peasPerPod, totalPeas;

  - Identifies names of three variables to name numbers
  - ***int*** means that the variables represent integers

- Variables are declared before they are used
  - Typically variables are declared at the beginning of the program → ask for storage space in advance
  - Statements (not always lines) end with a semi-colon

Explanation of code (3/4)

- Program statement for output results

  cout << "Press return after entering a number.\n";

  - ***cout*** (see-out) used for output to the monitor
  - Think of ***cout*** as a name for the monitor
    - "<<" points to where the data is to end up
  - '**\n**' causes a new line to be started on the monitor

- Program statement for input data

  cin >> numberOfPods;

  - ***cin*** (see-in) used for input from the keyboard
  - Think of ***cin*** as a name for the keyboard
    - ">>" points from the keyboard to a variable where the data is stored

Explanation of code (4/4)

- Program statement

```
totalPeas = numberOfPods * peasPerPod;
```

  - Performs a computation
  - '*' is used for multiplication
  - '=' causes totalPeas to get a new value based on the calculation shown on the right of the equal sign
  - Looks similar to mathematical equations??
    - They are called "executable" statements with clear "actions"
    - You will learn more actions in the following chapters

Overview

- 1.1 Computer Systems
- 1.2 Programming and Problem Solving
- 1.3 Introduction to C++
- *1.4 Testing and Debugging*

Testing and Debugging

- Bug
  - A mistake in a program
  - Difficult to find where it hides
- Debugging
  - Eliminating mistakes in programs
  - Term used when a moth caused a failed relay on the Harvard Mark 1 computer.
  - Grace Hopper and other programmers taped the moth in logbook stating:
    "First actual case of a bug being found."
  - Make sure you have tested the program completely

Program Errors

- Syntax errors
  - Violation of the grammar rules of the language
  - Discovered by the compiler
    - Error messages may not always show correct location of errors, but should be nearby
- Run-time errors
  - Error conditions detected by the computer at run-time
- Logic errors
  - Errors in the program's algorithm
  - Most difficult to diagnose
  - Computer does not recognize such errors

Get Hints from Compiler Messages

```
// Testing Your C++ Setup
#include <iostream>
using namespace std;

int main()
{
   cout << "Testing 1, 2, 3\n";
   return 0;
}
```

Sample Dialogue


```
Testing 1, 2, 3
```

- If you cannot compile and run, read error messages and check:
  - Typo: wrong variable names or commands
  - Library: missing #include, wrong library name or namespace
  - Semicolon: missing semicolon often indicates to the previous statement
- Still failed …
  - Try another computer or call for help

Is Your Program Correct ?

- Murphy's Law: If anything can go wrong, it will …
- Run the program on several data sets and check results
  - Have more confidence if those tests are passed
- Still not an absolute guarantee for correctness
  - The program may fail at some other data
- Train yourself to consider all possible cases in the program flow and write program carefully
  - Good disciplines may help → software engineering