# CSI 201 - Introduction to Computer Science

**Chapter 1**

Introduction to Computers and C++ Programming

Brian R. King
Instructor

---

# What is a computer?

- Today, the term "computer" typically refers to the hardware aspect of your system.
  - Example: 3.0 Ghz Pentium IV Computer with 512 MB RAM, 120 GB hard drive, DVD-RW drive, nVidia GeForce 4 with 64 mb Ram, etc.
- "A programmable electronic device that can store, retrieve, and **process** data." – Merriam Webster Dictionary.
- Simple definition: A device that **computes**.
- To compute something is to "**determine**" by mathematical or numerical methods.
- What does it mean to "determine"?

- Thus, a computer is a device that solves problems that can be solved through mathematical and logical means.
- http://virtualmuseum.dlib.vt.edu/ --History of computers

---

# How does a computer know what to compute?

- A computer by itself can do absolutely nothing without being told HOW to compute.
  - Try to turn on your computer without it's operating system loaded…. what happens? Nothing.
- This is called **software**.
- Software development is essentially the process of writing programs that the computer computes

---

# Classes of computers

- Three main classes of computers we will consider:
  - PCs (Personal Computer)
    - Relatively small used by one person at a time
  - Workstation
    - Larger and more powerful than a PC
  - Mainframe
    - Still larger
    - Requires support staff
    - Shared by multiple users
- But, there are others… What class is a PDA? Or your cell phone?

- What about the computer in your car? Or microwave oven? Or VCR?

(Answers given in class)

# Computer Organization

- Five main components
  - Input devices
    - Allows communication to the computer
  - Output devices
    - Allows communication to the user
  - Processor (or CPU – Central Processing Unit)
    - See http://www.intel.com/museum/index.htm for a very nice retrospective on the history of the microprocessor.
  - Main memory
    - Memory locations containing the running program
  - Secondary memory
    - Permanent record of data often on a disk **Display 1.1**

# Main Memory

**Display 1.2**

- Directly connected to the CPU
- Ordered sequence of cells, called memory cells.
  - Each cell contains zeros and ones – **binary** numbers.
  - The value in the cell can change during program execution
- Each cell has a unique location, called an **address**
- Cells are grouped together to store larger data items.
  - The address of the first cell is used to reference the data.
- Let's look more at the binary number system…

# A little about binary …

- We all know the base-10 number system – the decimal system.
  - Decimal systems have 10 distinct numbers used to represent all numbers in the system: 0-9
  - Every decimal number can be represented using powers of 10. The power depends upon the placement of the number.
  - Example: $354 = 3 * 10^2 + 5*10^1 + 4*10^0$
- Let's consider a binary number system – a **base-2** number system.
  - How many distinct numbers in a binary system?

  - What are the numbers?

- Everything in a computer is always represented in binary. Why binary?
  - The computer represents values using two voltage levels.
  - With two levels, we can represents two different values: 0 or 1

# Counting in binary…

- How do you count in binary?
  - 0, 1, then what? The same rules apply as counting in decimal.
  - 10, 11, then… 100, 101, 110, 111, 1000, ...
- How do you convert from binary to decimal?
  - Instead of applying powers of 10 for each number, you apply powers of 2.
  - What is 1011 in decimal? (What is $1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$?)

- How do you convert from decimal to binary?
  - You keep dividing by 2, until the quotient is zero. The remainders will construct the binary number.
  - Example: What is decimal 22 in binary?
    - 22/2 = 11 rem **0**
    - 11/2 = 5 rem **1**
    - 5/2 = 2 rem **1**
    - 2/2 = 1 rem **0**
    - 1/2 = 0 rem **1**
    - Read the remainders from the bottom going up, and you have your binary representation.
    - 22 decimal = 10110 binary
    - Let's check: $1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 16 + 4 + 2 = 22$. Success!
  - What is decimal number 35 in binary?

## Back to memory

- So, everything in a computer is always represented in binary.
- A **bit** is the smallest "unit" of data on a computer. A bit is a 0 or 1.
- A **byte** is the smallest **addressable** data item in the computer.
  - A byte is **eight** bits.
- Each memory cell holds a **byte** of data.
  - A **bit** is a binary digit, meaning, it can be represented only by a zero or a one
  - Therefore, a memory cell can hold the values 00000000 through 11111111 (or, 0 to 255 in decimal.)
- For example: What is a memory cell with the byte 01000001 actually representing? The CPU will always see this as 01000001.
- But, **how** it's interpreted depends on how your program is interpreting the **meaning** of the value.
  - You may interpret it as a decimal number 65.
  - You may interpret it as a character 'A' if the cell is an ASCII letter (or, as we'll see soon, a *character*).
  - A CPU instruction may also look like 01000001.
  - Interpretation depends on the current instruction being executed

## The Processor

- This is the "brain" of the computer.
- It's a brain without intelligence! It only does what it is told. It can't determine things on its own.
- Who tells the CPU what to do?

### SOFTWARE!

## Computer Software

- Software is defined as a collection of **programs** written to perform specific tasks.
  - A **program** is defined is a set of instructions for the computer to follow.
- Examples of software
  - Word Processors, Spreadsheets, Databases, etc.
  - System drivers (mouse, video, etc…)
  - Compilers

## Operating System

- A computer has a variety of resources available (memory, hard drives, mouse, printer, video, etc…)
  - *CPU Time* is also a very important resource managed by the OS
- The operating system is special software that sits between the hardware and application software.
- The operating system provides access to computer resources through **device drivers** and system **services**.
  - The device driver is loadable software that interacts with the operating system. It gives the operating system capability to access the hardware on behalf of application software.
  - For example, to access your printer (hardware) from a word processor (an application) in Windows, you need to install a device driver, otherwise your word processor will not recognize your printer.
- Examples of Operating Systems – Windows, Mac OS-X, Linux, UNIX, MS-DOS

# Running a program

- A program tells the computer what to do.
- Almost all programs work with some type of data input in order to run properly.
- The program instructs the computer to perform some type of output based on the input it receives.

**Simple View of Running a Program**

Program → Computer
Data → Computer
Computer → Output
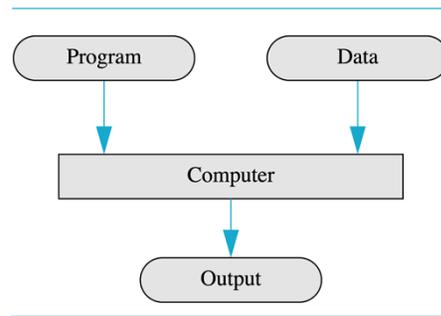
# Giving the CPU instructions

- CPUs have a limited, simplistic instruction set that carry out simple operations, such as increment, decrement, add, subtract, logic operations, comparing values, copying values, etc.
- Each CPU instruction has a unique binary number associated with it. This raw binary format is often referred to as **machine language** or **machine code**. But, we don't want to program a CPU in binary!
- **Assembly language** provides a simplistic language that maps from binary code to readable instruction names, called **mnemonics**. (Example: ADD, MOV, MUL, JE, etc…)
- Fortunately, 99% of programming is done with a **high-level language** – it gives the computer programmer the ability to write a program using a language much closer to human language than assembly language.

# Compiler + Linker = Ready to run code

- **Compiler** translates high-level language **source code** into an intermediate **object code**.
- The **Linker** links together the object code generated by the compiler with other code stored in **libraries** to generate machine code ready for the CPU to execute.

**Preparing a C++ Program for Running**

C++ program → Compiler → Object code for C++ program

Object code for other routines

Object code for C++ program + Object code for other routines → Linker → Complete machine language code ready to run
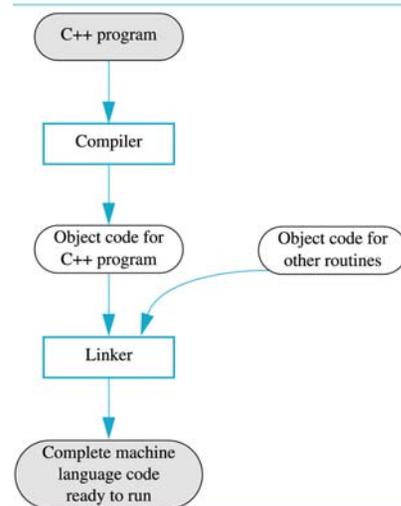
# Writing programs to solve tasks
# The heart of CSI 201

- This course will focus on using the C++ computer programming language, a high-level language, to solve computing problems.
- The most difficult part of this course is not understanding how to solve a problem, but understanding how to translate your solution into C++, thereby translating it into a language the computer will understand.

# Algorithms lead to programs

- An **algorithm** is a sequence of instructions that leads to a solution.
  - A more rigorous C.S. definition is "any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output." – Intro. to Algorithms, Cormen et. al. pg 5.
- Common algorithms we will see later:
  - **Search** for a specific name in a list of names.
  - **Sort** a list of numbers in increasing order.

- A **program** is an algorithm interpreted in the computer language of choice.
- Programming is a creative process. There are no complete set of rules for creating a program.
  - There are often near infinite paths that lead to one solution!
  - One goal of computer science is to find the most optimal solution for a computing problem.
  - Optimizations are usually done regarding the use of time and space as computing resources.
- So, what are the formal steps to the software development process?

# The software life cycle

- Step 1: **Problem Definition** phase. You must perform an analysis of the problem and generate a specification of the task to be solved.
- Step 2: **Design** phase. Design a solution for the problem. (Sometimes called the problem-solving or design phase.)
  - The result of this step is one or more algorithms that work together to solve the problem.
- Step 3: **Implementation** phase
  - Translate the solution into a high-level language.
- Step 4: **Testing**
  - This could often lead to going back to Step 3, Step 2, or if you really screwed up, even Step 1!
- Step 5: **Maintenance** and evolution of the system.
- Step 6: **Obsolesence**

# The Problem with Software Engineering

- So, isn't software engineering just a matter of following those rules?
- **Please read the following quote carefully:**
  "Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product, according to a newly released study commissioned by the National Institute of Standards and Technology (NIST). More than half of the costs are borne by software users, and the remainder by software developers/vendors." See (http://www.nist.gov/public_affairs/update/upd20020624.htm)
- Have you ever had a program crash? (e.g. The infamous "blue-screen-of-death"!) Why do you think are there so many bugs in software today? What do you think about software that runs devices that are a matter of life and death?
- Don't be fooled! CSI201 alone will hardly come close to being able to make you into a true software engineer. It's only a small part of the foundation you will need to help alleviate the ever-growing problem of software failure!
- A few more thoughts for potential CS majors:
  - "It has long been my personal view that the separation of practical and theoretical work is artificial and injurious. Much of the practical work done in computing … is unsound and clumsy because the people who do it have not any clear understanding of the fundamental design principles of their work. Most of the abstract mathematical and theoretical work is sterile because it has no point of contact with real computing." – Christopher Strachey
  - When you complain about the theory and mathematical based CS classes down the road, remember this slide. These classes are part of the foundational material that will help make the difference between a hacker and a true software developer.
- Those that are considering CS as a major, please do not neglect these facts! Be disciplined! Take the discipline seriously. Do something to make a difference.

# The C++ Programming language

- Where did C++ come from?
  - Derived from the C language
  - C was derived from the B language
  - B was derived from the BCPL language
- C was developed by Dennie Ritchie of AT&T Bell Labs in the 1970s, and is closely tied to the UNIX operating system.
- C++ was developed by Bjarne Stroustrup of AT&T Bell Labs in the 1980s.
  - C++ has facilities to allow the programmer to follow object-oriented programming (OOP) techniques, to be discussed later in the course.
- Why the '++'?
  - ++ is an operator in C++ and results in a cute pun intended to depict C++ as the next C.

note

# Basic C++ Program structure

- Here's an example program that prints "Hello, World!" to the screen:
- A C++ programs begins with:
  ```
  #include <iostream>
  using namespace std;

  int main()
  {
  ```
- A **statement** is an instruction that tells the computer to do something:
  ```
  cout << "Hello, World!\n";
  ```
- A C++ program ends with:
  ```
      return 0;
  }
  ```

# ex1.cpp

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

int main()
{
  cout << "Hello, World!\n";

  return 0;
}
```

# Simple application of software life cycle.

- Problem definition:
  - Ask a person for two numbers from the keyboard and output the sum of those numbers on the screen.
- Design of a solution:
  - 1. Ask the user to enter one number
  - 2. Read in the number
  - 3. Ask the user to enter another number.
  - 4. Read in the second number.
  - 5. Add the numbers together.
  - 6. Output the result.

# Implementation

- We start with:
  ```
  #include <iostream>
  using namespace std;

  int main()
  {
  ```
- We end with:
  ```
      return 0;
  }
  ```
- So, what statements go in the middle?

## Implementation continued…

- 1. Ask the user to enter one number
  - We use **cout** to output data to the monitor
  - We use the **<<** operator to send the data to the monitor
  ```
  cout << "Please enter a number:\n";
  ```
  - The \n is a "newline" character. Think of it as the ENTER key.
- 2. Read in the number
  - How do we store what we read in? **Variables!**
  - Variables must be **declared** before they can be used.
  - Variables have a type, followed by a name.
  ```
  int first_num;
  ```
  - We use **cin** for input from the keyboard.
  - We use the **>>** operator with cin to extract data from the keyboard.
  ```
  cin >> first_num;
  ```
- 3. Ask the user to enter another number.
  ```
  cout << "Please enter another number:\n";
  ```

## Implementation continued…

- 4. Read in the second number.
  ```
  int second_num;
  cin >> second_num;
  ```
- 5. Add the numbers together.
  - Simple mathematical statements look as you would expect them to, however, the variable to store the value is always on the left hand side.
  - Example: the statement `c = a + b` causes c to get the result of adding a and b together.
  ```
  int result;
  result = first_num + second_num;
  ```
- 6. Output the result.
  - **endl** is a constant that represents the newline character.
  ```
  cout << "The result is: ";
  cout << result;
  cout << endl;
  ```

## #include and using namespace;

- These lines tell your program where to find information about items used in your program.
- #include <iostream>
  - iostream is the name of a library
  - It includes the definition of cin and cout.
- using namespace std; tells the compiler that we are using standard definitions from the iostream library.
  - Without this, cin and cout will NOT be defined!

## The main( ) function

- Every program must have a `main()` function in order to execute properly.
- It is declared as:
  - int main( )
- It has delimiting brackets { }
- The **body** of your program is placed between the braces.

## ex2.cpp

```
// ex2.cpp
// Written by Brian King
// Date: 01-Jan-2003
//
// Description
// -----------
// This program asks the user to enter two numbers, calculates the sum,
// outputs the result.
//
#include <iostream>
using namespace std;

int main()
{
    // Variables used to store numbers being used
    int first_num, second_num, result;

    cout << "Please enter a number:\n";
    cin >> first_num;
    cout << "Please enter another number:\n";
    cin >> second_num;

    // Calculate the sum
    result = first_num + second_num;
    cout << "The result is: " << result << endl;

    return 0;
}
```

## C++ program structure

- Comments
- Indentation – formatted code makes your program easier to read
- Place only one statement per line.
- Long lines – break them up!
- Variable names – make them meaningful. (More on variables in the next section.)
- Simplify statements together to make your program more concise when possible.

## Errors from testing

- **Syntax errors**
  - Violation of the grammar rules of the language
  - Discovered by the compiler
    - Error messages may not always show correct location of errors
- **Run-time errors**
  - Error conditions detected by the computer at run-time
  - Divide by zero is a common run-time error.
- **Logic errors**
  - Errors in the program's algorithm
  - Most difficult to diagnose
  - Computer does not recognize an error
  - Example: You use a – sign instead of a + sign.
- **Design error**
  - Your program works fine, but it doesn't solve the problem given.
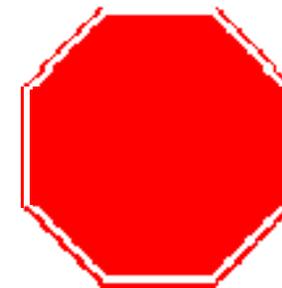
The End

# Display 1.1

**Main Components of a Computer**
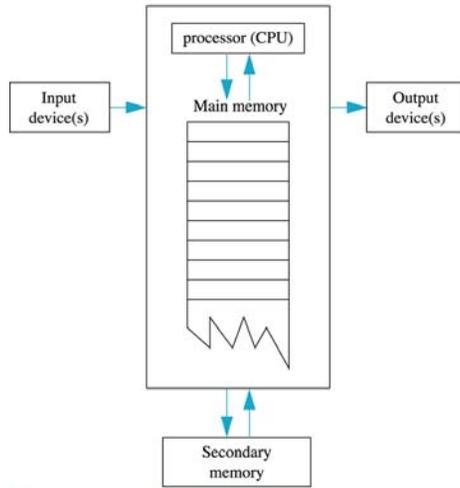
# Display 1.2

**Memory Locations and Bytes**

byte 1
byte 2
byte 3     3 byte location with address 1
byte 4
byte 5     2 byte location with address 4
byte 6     1 byte location with address 6
byte 7
byte 8     3 byte location with address 7
byte 9