COMP-202
Unit 0: Course Details

CONTENTS:
Focus of the Course and Prerequisites
Outline and Textbook
Course Structure and Grading Scheme
Computer Lab Info and Required Software
Getting started "thinking like a computer scientist"

COMP-202
Intro to Computing I

Reasons for being here?

- Reason #1:
  - My faculty made me
- Reason #2:
  - Learning to understand better computers is exciting!
- Reason #3:
  - I want to conquer the world using robots!

CAN'T YOU DO ANYTHING RIGHT?
GLASBERGEN

Goals of this course

(What you will be able to do in a few months)

- Understanding how computers think
- Being able to describe a task in a way a computer can understand
- Breaking complex tasks into smaller, simpler tasks
- Translating these instructions into the programming language Java
- Pass a required course
- Have fun (?)
-
- Aimed at students with **little** or **no** background in programming and/or knowledge of computer science

I heard from one of my friends that this course is really hard and time consuming!

- Unfortunately, programming computers is a pretty time consuming process. The main challenge is not always **writing** the code, but **debugging** the code when things don't work as they should.
-
- I will do everything I can to make the course load reasonable, but sometimes it is very difficult to figure out how long something will take ahead of time.

I hate computers and am only here because my stupid faculty made me take this @#$# course!

•Even if you never write another program, this course will help you to:

•

•Understand what types of problems computers can and can't solve in practice. (Applications?)

•Improve your logical thinking and problem solving skills

•A new tool to help you with your other courses.

Course Plan

- We will spend about half the course learning the "basics" of programming. This will involve mostly learning about the FLOW of a program
-
- We will then spend the remainder of the course focusing on Object-Oriented Programming.

Non-Goals of this course

(What you will not be able to do at the end. Sorry)

How do I Ace this course?

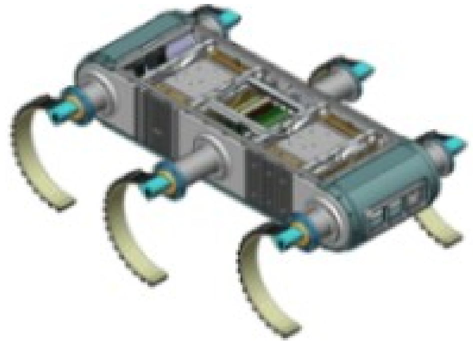
- Practice! When you see an example in class, try it at home.
- Do your assignments!!!!!!!
- Do your assignments!!!!!!!
- Prepare for the lectures ahead of time. Make sure you understand the previous lecture and come prepared to class with questions.
- Ask a question whenever you don't understand something.
  - It is the only way that Dan can figure out what he isn't explaining well.

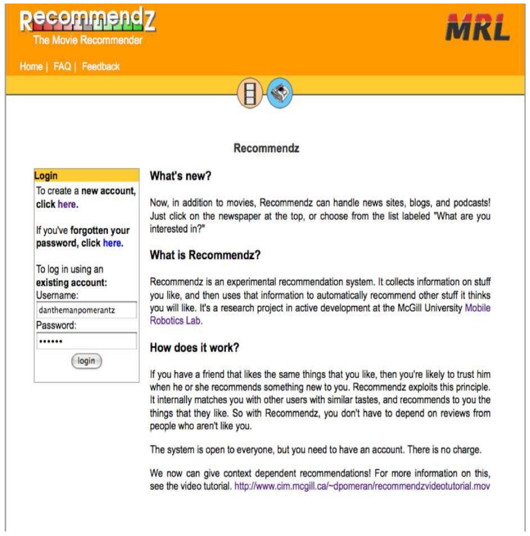
A bit about your instructor (the one minute summary!)

- Dan Pomerantz (course lecturer)
-

Recommender Systems

- Msc. from McGill.
- Worked on movie recommendation systems in MRL lab
- http://www.cim.mcgill.ca/~junaed/video/rollover_follow.mpg
- http://www.recommendz.com





Recommendz

Login

To create a **new account, click** here.

If you've **forgotten your password, click** here.

To log in using an **existing account:**

Username: danthemanpomerantz

Password: ••••••

What's new?

Now, in addition to movies, Recommendz can handle news sites, blogs, and podcasts! Just click on the newspaper at the top, or choose from the list labeled "What are you interested in?"

What is Recommendz?

Recommendz is an experimental recommendation system. It collects information on stuff you like, and then uses that information to automatically recommend other stuff it thinks you will like. It's a research project in active development at the McGill University Mobile Robotics Lab.

How does it work?

If you have a friend that likes the same things that you like, then you're likely to trust him when he or she recommends something new to you. Recommendz exploits this principle. It internally matches you with other users with similar tastes, and recommends to you the things that they like. So with Recommendz, you don't have to depend on reviews from people who aren't like you.

The system is open to everyone, but you need to have an account. There is no charge.

We now can give context dependent recommendations! For more information on this, see the video tutorial. http://www.cim.mcgill.ca/~dpomeran/recommendzvideotutorial.mov

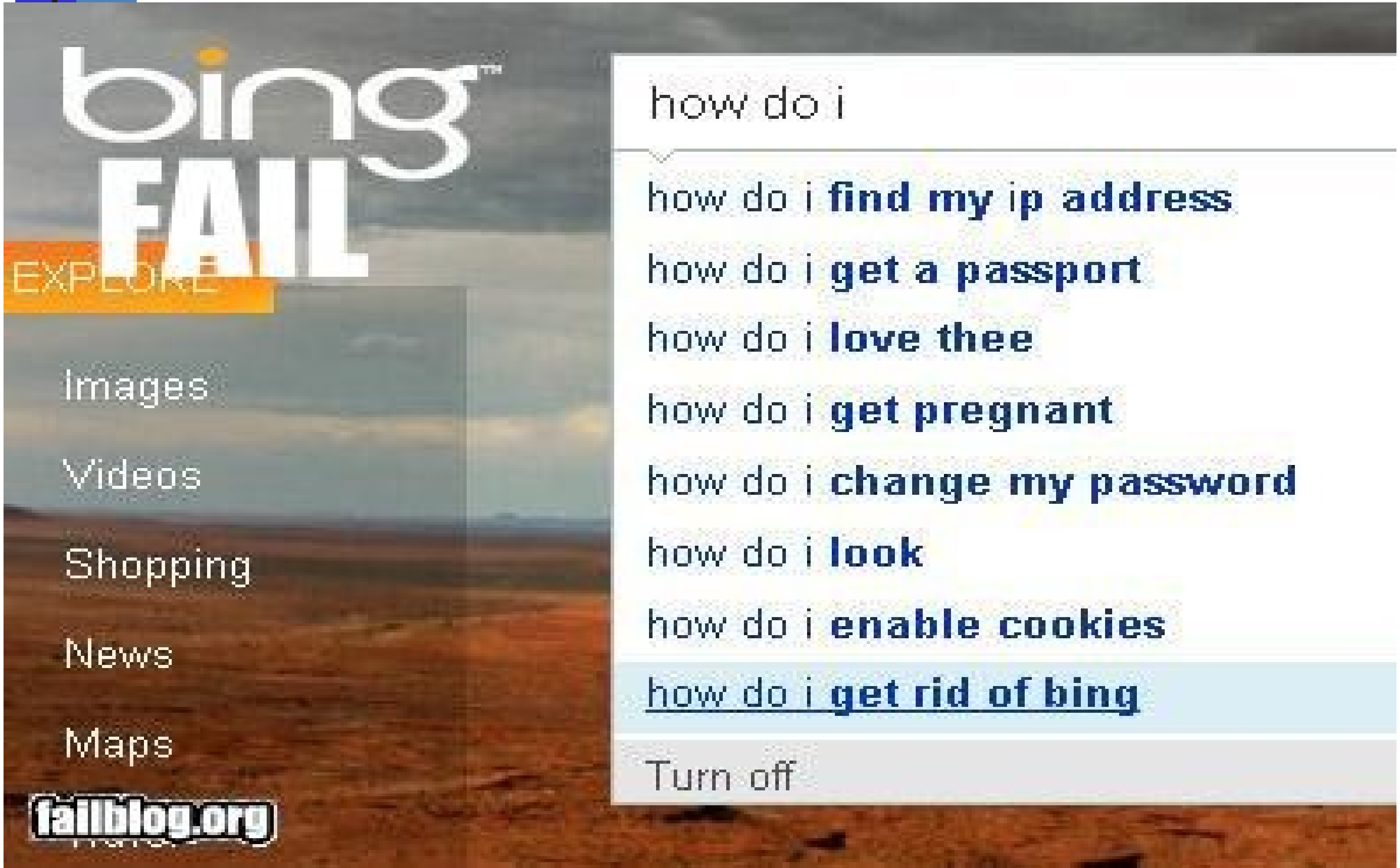
Bing

- Have been doing work on Bing search engine
-

- However, I still use google

Google™

My favourite thing about Bing



how do i

how do i find my ip address
how do i get a passport
how do i love thee
how do i get pregnant
how do i change my password
how do i look
how do i enable cookies
how do i get rid of bing

Turn off

Still annoyed……

Prerequisites

- Prerequisite: A CEGEP-level math course or equivalent
- For those who graduated from high school outside Quebec and thus never attended CEGEP: any upper-level math course
- In any case, the **ability to think logically and rigorously** is more important than calculus, algebra or trigonometry

Course Communication

- Course home page:
    - `http://www.cs.mcgill.ca/~cs202`

–Instructor contact information, course outline, lecture notes, assignment specifications, and other course material

- *my*Courses (WebCT Vista):
    - `http://www.mcgill.ca/mycourses`

Course Structure: Lectures

- Lectures (4.5 hours/week, compulsory)
  - Highlight key concepts; opportunity to **practice** and **ask questions, receive and give feedback, MAKE MISTAKES**.
  - You will get more out of the class if you do the readings before hand and participate.
  - All material taught in lectures is examinable unless otherwise stated
  -
  - **Ask Questions if you don't understand something.**
  -
  - It is the only way I'll know whether I'm explaining something clearly or not.

Assignments

- Assignments are designed to let you practice the things you learn in class.
- It is VERY important that you do the assignments, as they offer you a chance to get feedback on your progress.

Outside of class

- Tutorials (optional but very useful)
  - There will be several tutorials throughout the term. These will be designed to teach material not covered in lectures (e.g. setting up your account with SOCS) and also to give you more practice on it
  - Will be led by a teaching assistant
  - Smaller groups typically
- Office hours (1 instructor, 3 teaching assistants)
  - Approximately 8 office hours per week

Grading Scheme (1)

- 4 assignments: 30%
- Assignment 1: 5%
- Assignment 2: 5%
- Assignment 3: 10%
- Assignment 4: 10%
-

There is also the possibility that some of these assignments will be "split" into 2 parts, depending on timing. In this case, we'd have more assignments but each would be worth less.

-

- All assignments count toward your final grade
  - One of the best ways to learn the material
  - **MUST** be done **INDIVIDUALLY**

Grading Scheme (2)

- 4 assignments: 30%
- 1 midterm examination: 20% (end of May or early June)
  - In class
  - E-mail your instructor as soon as possible if you have a conflict/preference of date!
- 1 final examination: 50%
  - July 7th
  -
  - Students who do poorly on the midterm can transfer the weight to the final exam.

Plagiarism :([1]

-
-
-
-
-
-
-
- [1]*Maja Frydrychowicz.* Lecture. *Date*: January 4th, 2011. Slide number 23

Recommended Textbook (**not** required)

- How to Think Like a Computer Scientist (Java Edition)
-
- http://openbookproject.net/thinkcs/java/english/dist_v4.0/thinkapjava.pdf

SOCS Computer Labs

- If you are officially registered in the course, you can create an account to use the computers on the 3rd floor of Trottier building
- Computer availability:
  - Computers in open areas: physically accessible 24 hours a day, 7 days a week
  - Computers in side rooms: physically accessible on weekdays 10:00 - 20:00, weekends 12:00 - 20:00
- Consultant on duty: weekdays 10:00 - 20:00, weekends 12:00 - 20:00
- Computers run GNU/Linux (Unix-like OS), not MS Windows
  - **Unix seminars** are offered by SOCS Systems staff

Required Software

- You are encouraged to use your personal computer or laptop to complete course work
- Software used in this course
  - Required: Java Development Kit (JDK)
  - Optional: RText, Eclipse (later in the course)
- See course outline for details on how to obtain the above software packages
- All programs you submit for assignments must compile and run using JDK 6 or later
  - JDK is backward compatible; programs that compile and run under previous versions also compile and run under JDK 6

Useful Tips

- Do read everything carefully: slides, notes, textbooks, instructions, assignment specifications, documentation, announcements on *my*Courses, ...
- Do not wait until the last minute to do your assignments
- Do not fall behind; each new concept builds on previous ones
- Contact instructors / TAs if you have difficulties
- Do not expect to be given every single detail; you will have to look things up in the provided material and deduce some things on your own
- **Experiment**
- Practice makes perfect!

With that in mind...

- With that in mind, let's get started (sorry!).

How to Think Like a Computer Scientist

- Most important thing is to *modularize*
- That is: take a big problem and break it down into smaller problems

How to Think Like a Computer Scientist

- We are not going to be able to tell the computer in one line
-
- “Hey Computer! Make Star Craft for me!”
-
- Or explain to a robot:
-
- “Hey Robot, go save the galaxy like R2D2 and C3PO!”
-
- We are going to have to “explain” to it, piece by piece how to do this.

Scenario: Managing a Day's Tasks

- Hypothetically, suppose I am not able to attend lecture or do any of my tasks on Thursday because of a conflict.
-
- I have 2 options:
-
- 1)Find my long lost twin brother that I didn't know I had
- 2)Program a robot to act like me in my place.
-
- Problem: Teaching a robot how to act like me for an entire day is complicated!
-
- The first thing I need to do is break the problem down into smaller problems

Example: Teaching a robot to replace Dan for a day

- The first thing I will want to do is break my day down into smaller, more manageable tasks. For example,
-
- 1)Get ready for work
- 2)Get to school
- 3)Lecture comp 202
- 4)Make dinner
- 5)Do laundry
- 6)~~Watch the Rangers lose again~~ (oops)
- 7)Go to sleep
- Note: Your list of tasks may be different.

Example: Teaching a robot to replace Dan for a day

- Things look a bit more manageable now, but some of the tasks are still a bit complicated.
-
- Ex: Make Dinner

Example: Teaching a robot to replace Dan for a day

-
-

Example: Teaching a robot to replace Dan for a day

- Perhaps to avoid this, we should break the task “make dinner” down further.
-
- How?

Example: Teaching a robot to replace Dan for a day

- Perhaps to avoid this, we should break the task “make dinner” down further.
-
- Examples:
-
- 1)Find a recipe
- 2)Buy food
- 3)Gather ingredients
- 4)Check if anything is flammable (if so, make sure I have a fire extinguisher!)
- 4)Assemble ingredients
- 5)Taste and verify that it tastes good

Example: Teaching a robot to replace Dan for a day

- Even these tasks can be broken down further:
-
- We could break “buy food” down into
-
- 1)Choose the best grocery store
- 2)Go to grocery store
- 3)Find each item
- 4)Pay cashier
- 5)Go home

Example: Teaching a robot to replace Dan for a day

- Continuing this, we could break “pay cashier” into:
-
- 1)Wait in line
- 2)Wait for cashier to give you the sum
- 3)Look for money
- 4)Realize you don't have any
- 5)Plead with the cashier to let you take the items anyway
- 6)Get thrown out of the store for not paying
-

Example: Teaching a robot to replace Dan for a day

- Eventually, you will get down to simple enough tasks that you understand the steps necessary.
-
- Wait in line
-
- could be broken down into
-
- 1)Stand up
- 2)Don't fall
- 3)Check if the cashier is done with the current customer.
- 4)If so, walk forward 2 feet. If not, go back to step 1
-

Programming a Computer

- Now that you convinced me how hard programming computers is, why should I bother? It seems so much easier to explain things to a human being!

Programming a Computer

- Computers are stupid!
-
- We have to explain all our instructions in very simple, unambiguous ways.
-
- In exchange, computers are very fast!
- (you try multiplying a 45 digit number!)
-
- They are also very good at following instructions *IF YOU GIVE THEM THE RIGHT INSTRUCTION TO FOLLOW!*

Programming a Computer

Kasparov vs DeepBlue

Programming a Computer

Humans are good at some things, but in certain cases, computers are more reliable.

Many things, such as stress, can cause us to perform worse when it matters the most.

Programming a Computer

Humans are good at some things, but in certain cases, computers are more reliable.

Many things, such as stress, can cause us to perform worse when it matters the most.

awww---->

(actually these guys just stink in general)

Java vs. Other programming languages

- We are going to study Java. However, most other languages, (e.g. C, C++, C#, Python, Perl, Matlab, etc.) will work the same way
-
- The differences will be:
-
- 1)How simple the steps have to be broken down for the computer
- 2)The *syntax* for translating these simple steps from human language to computers
- 3)The exact procedure for combining these “atomic” steps.
-

Java vs. Other programming languages

- For some people “making dinner” is such a natural step that we don't need to break it down any further than that. For others, you may need to spell it out more clearly.
-
- Programming languages are similar. Some languages require spelling things out more than others.
-

Java vs. Other programming languages

- What is the advantage or disadvantage of either approach?
-
- Would you rather have a chef who required lots of detail or not?
-

Getting started with Java:

- http://www.cs.mcgill.ca/~dpomer/comp202/summer2011/resources.html

-First install JavaSdk at the link above.

This is often already installed on MacOsX by default

Java vs. Other programming languages

- (optional) Then you can install Rtext or Ecclipse or another IDE (integrated development environment)
-
- Personally, I prefer to use Notepad or TexEdit or Emacs as they are simpler to set up.
-
- The IDE matters more when you are using a huge project. For example with hundreds of files in it. Then having 100 windows of Notepad open at once is tough
-

Java vs. Other programming languages

- Next step: Open a command terminal window:
-
- On MacOsX click
-    Applications-->Utilities-->Terminal
- On Windows click start--->run . Then type “cmd”

Java vs. Other programming languages

- Now, write your program either in Notepad or one of the ides (or any other text editor). Be careful if you use microsoft word though.
-
- In this case, just download the file HelloWorld.java from the course webpage

Java vs. Other programming languages

- From the command prompt, first navigate to the directory of your file HelloWorld.java
-
- You do this by typing
-
- cd nameofdirectory
-
- For example
-
- cd C:\Documents\comp202\javafiles
- or
- cd /Documents/comp202

Java vs. Other programming languages

- Then type
-
- javac HelloWorld.java
-
- If nothing shows up, great!
-
- Common problems:
-
- 1)HelloWorld.java not found (make sure you did cd to the directory HelloWorld.java is saved in)
- 2)(windows only) javac : command not found . In this case you have to tell your computer WHERE to look.

Java vs. Other programming languages

- To deal with number 2, there are 2 options:
-
- 1)First fast way to do it is to type the entire path to javac
-
- For example,
-
- C:\Windows\ProgramFiles\java\jdk1.6.0\bin\javac HelloWorld.java
-
-

Java vs. Other programming languages

- Better is to edit the class path variable.
-
- To do this in vista or windows 7, search for something like “edit system/environment variables”
-
- Then click edit path and add C:\Windows\ProgramFiles\..... to it
-
-

Java vs. Other programming languages

- After you do this, you should see a file HelloWorld.class in the same folder as HelloWorld.java
-
- Now you can run your program by typing
-
- java HelloWorld
-
- (Note: If you typed the full path to use javac, you will have to do the same thing here)
-
-

Analyzing what we just wrote:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- Here we see the simplest Java program that does anything.

What do we notice?

- The entire block is surrounded by a set of { } and inside a “class”
- There is another block between { } called a method. In this case it is the “main” method
- System.out.println will print some text to the screen
- Some statements should end in ;

What do we notice?

- Almost every single file you ever write in Java must contain exactly 1 class (minor exception is something called enum which we'll mention later)
-
- A class always looks like:
-
- [description] class NameOfClass {
-
- the definition of the class goes in here!
-
- }

What do we notice?

- A class contains 2 things:
-
- 1) methods
- 2) variables that are not part of any method
-
-
- Note that methods can also contain variables. For the beginning of the course we will only talk about the variables that are inside methods.

What do we notice?

A method looks like

```
[description] [return type] nameOfMethod(arguments) {

    method body goes here

}
```

and is essentially a set of instructions grouped together according to some logical meaning.

What do we notice?

An instruction in Java always ends in a ;

There are some lines of code in Java that are not instructions. For example, the line

```
public static void main(String[] args)
```

is NOT an instruction.

```
System.out.println("HelloWorld!");
```

IS an instruction

What do we notice?

Every time you write

System.out.println

it will print whatever is between the “” to the screen and then a new line.

What is the point of the quotation?

This denotes something in Java called a String . Essentially, this tells Java to treat everything between the “ “ as letters as opposed to part of a different computation

A bit more complex program

- public class CallBelow {
- public static void main(String[] args) {
  - System.out.println("Hello");
  - System.out.println("");
  - System.out.println("");
  - System.out.println("");
  - System.out.println("");
  - System.out.println("");
  - System.out.println("");
  - System.out.println("Down there!");
- }
- }
- What do you think this does?

A bit more complex program

```
public class StringAndIntegers {
    public static void main(String[] args) {
        System.out.println(“4+6”);
        System.out.println(4+6);
        //the next one is tricky!
        System.out.println(3+5*2);
    }
}
```

Programming and Computers

- In order to understand what programming is, we need to know what a computer is
- A computer is a machine that executes lists of instructions
  - We feed a list of instructions to the computer and the computer executes them
  - The computer may apply the instructions on additional information fed to the computer (the *input*)
  - The computer may produce information as a result of executing this list of instructions (the *output*)

Programming and Computers

- Programming a computer involves two things:
-
-
- 1) Designing lists of instructions that will make the computer solve specific problems
- 2) Having the computer execute the instructions
-
-

•The purpose is to have the computer solve the problem for you instead of you solving the problem by hand

Designing instruction list

- Designing an instruction list is largely about breaking problems down into smaller tasks.
-
- It is also about considering the “input and output” of a program

Input vs Output : Input

- The *input to a program is what goes into it. It is whatever is given to the program or problem.*
-
- *This is anything that is necessary to solve it.*

Input vs Output : Output

- The output *from a program is what comes from it.*
-
- *This is anything that is produced as a result of the program running*

Example: f(x) = 2x

- *The input to this mathematical function is "a number"*
-
- *The output to this function is "a number"*
-
- *Note: Input is similar to "domain" in math. Output is similar to "range"*

Example: f(x) = 0

- *What is the input and output?*

Example: $f(x,y) = 2x + 3y$

- *The input to this mathematical function is "two numbers" -- x and y*
-
- *The output is 1 number. The sum of 2x and 3y*

Input/Output

- *Input/Output is not necessarily restricted to math problems.*
-
- *For example, what is the input and output of the problem “cooking scrambled eggs”*
-
-
- *What is the input and output of the problem of counting how many cards you have of a given suit in a deck of cards?*

Amelia Bedilia

Breaking the problem down

- How can we write this suit-counting problem down in simple to understand instructions?

Counting Cards of a Given Suit

- 1)Take a blank piece of paper and the deck
- 2)If there are cards left, take the top card in the deck
- If there are not cards left, skip to step 2
- 3)Check if the card belongs to the suit you are looking for. If so, make a mark on the piece of paper.
- 4)Go back to step 2
- 5)Count the number of marks on your paper. That number is the answer we want.

Counting Cards of a Given Suit

- 1)Take a blank piece of paper and the deck
- 2)If there are cards left, take the top card in the deck
- If there are not cards left, skip to step 2
- 3)Check if the card belongs to the suit you are looking for. If so, make a mark on the piece of paper.
- 4)Go back to step 2
- 5)Count the number of marks on your paper. That number is the answer we want.

an “if statement”

Counting Cards of a Given Suit

- 1)Take a blank piece of paper and the deck
- 2)If there are cards left, take the top card in the deck
- If there are not cards left, skip to step 2
- 3)Check if the card belongs to the suit you are looking for. If so, make a mark on the piece of paper.
- 4)Go back to step 2
- 5)Count the number of marks on your paper. That number is the answer we want.

an "if statement"

a "loop"

Counting Cards of a Given Suit

- 1)Take a blank piece of paper and the deck
- 2)If there are cards left, take the top card in the deck
- If there are not cards left, skip to step 2
- 3)Check if the card belongs to the suit you are looking for. If so, make a mark on the piece of paper.
- 4)Go back to step 2
- 5)Count the number of marks on your paper. That number is the answer we want.

a "variable"

an "if statement"

a "loop"

Instructions and Precision

- The instructions for finding a card in a deck or counting the number of cards of a given suit in a deck are very precise and unambiguous
-
- Amelia would have no room for misunderstanding
-
- Writing lists of instructions like these is the very essence of programming a computer

Programs (1)

- A computer program is essentially a list of instructions telling a computer what to do
- The computer is "stupid" in that it is just following the instructions without knowing what it is doing.
- Thus you must be very precise and omit no details.

Programs (1)

- Programs have an “input” and an “output” as well.
- If you omit the proper instructions or include the wrong instructions, generally 4 things can happen:
- 1)You get incredibly lucky and on that particular input it works anyway
- 2)The program gives the incorrect output
- 3)The program crashes
- 4)The program goes on forever and ever

Case 1

- Sometimes you will give the computer the wrong instructions, but it will work anyway on a particular input.
- For example, in the card program, suppose we ommitted the instruction “make a mark on the piece of paper”
- This means the count will always be 0, no matter what the input.
- When would this still lead to the “right” answer?

Case 2

- The program gives the wrong output
- This would happen in the previous example on any input where the pack of cards had at least one card of the suit in question.

Case 3

- Sometimes the program will crash.
-
- In the card example, this happens if the computer is unable to do a certain step.
-
- For example, at step 1 we have: “Take a blank piece of paper and the deck”
-
- Suppose you don't have any paper. Then this step is impossible.
- In a computer program, when something like this happens, the program will crash, exiting without output

Case 4

- The program goes on forever and ever
-
- What if after step 3, we inserted "Put the card in your hand back on top of the deck"
-
- Then the program would go on forever. We would continually pick up the top card, check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card (the same card!), check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card

Case 4

- check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card, check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card, check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card, check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card, check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top. Then we would pick up the top card, check if it is the suit we are interested in, possibly mark it down on paper, then put it back on top.

Human and Computer Languages

- Compare to the following English sentence:
- ***"The lady hit the man with a baby"***
-
- Does this mean
- 1)A lady hit a man who had a baby? (what a jerk!)
- 2)A lady used a baby to hit a man? (good lord!)
- 3)A lady and a baby ganged up on a man and hit him. (kids today!)
-
- Good programming languages (such as Java), on the other hand, always has only one possible interpretation.

Human and Computer Languages

- One of the challenges is to learn the different interpretations the computer will give to commands.
-
- The computer will not normally tell you how it is interpreting things. It is up to you to figure it out, both by looking at your code and observing the output.

How a Computer is Organized

- Before going into more details on Java, we need to understand a bit about how a computer is organized.
-
- How does a computer perform computations?

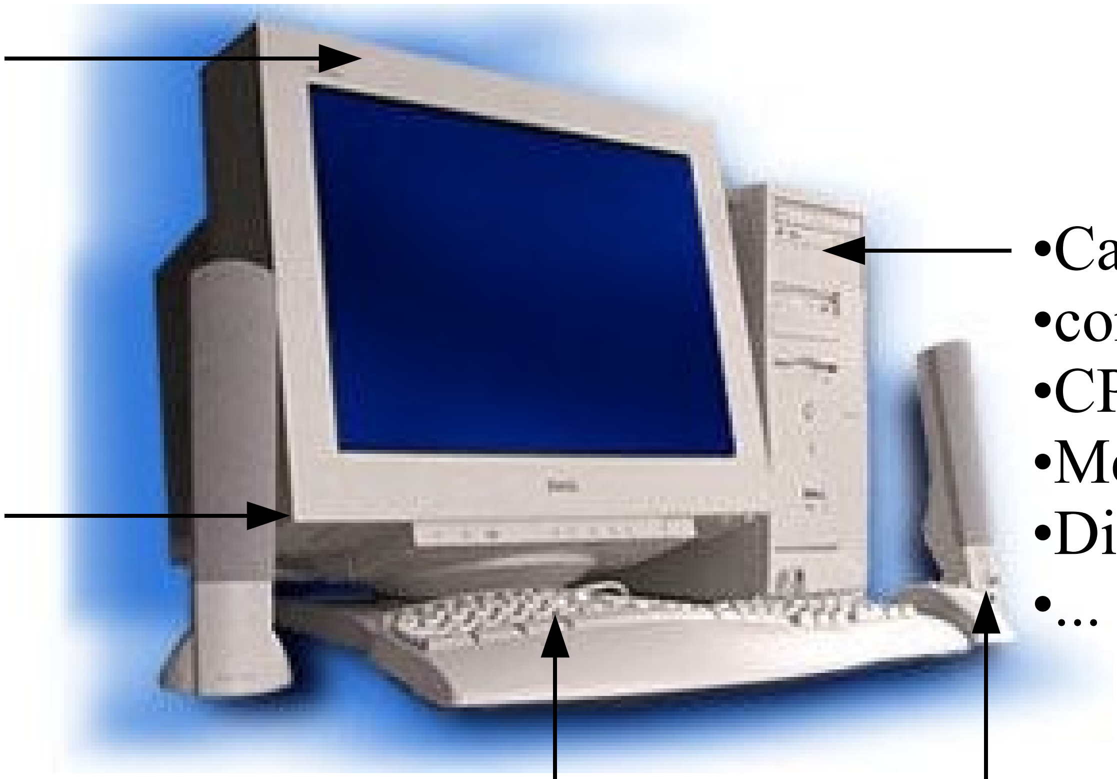
Hardware and Software (1)

- A computer system consists of both *hardware* components and *software*
- Hardware consists of the physical, tangible parts of a computer
  - Cases, monitors, keyboard, mouse, chips
  - Rule of thumb: If you can take it in your hands and it is part of a computer system, then it is hardware
  - It is the hardware which executes the instructions
- Software: Programs and data that they use
- A computer requires both hardware and software
  - Software cannot run without hardware; instructions are useless unless they are performed by someone / something

Hardware and Software (2)

- Hardware will not do anything without software telling it what to do
- Therefore, each is essentially useless without the other

An (old) Personal Computer

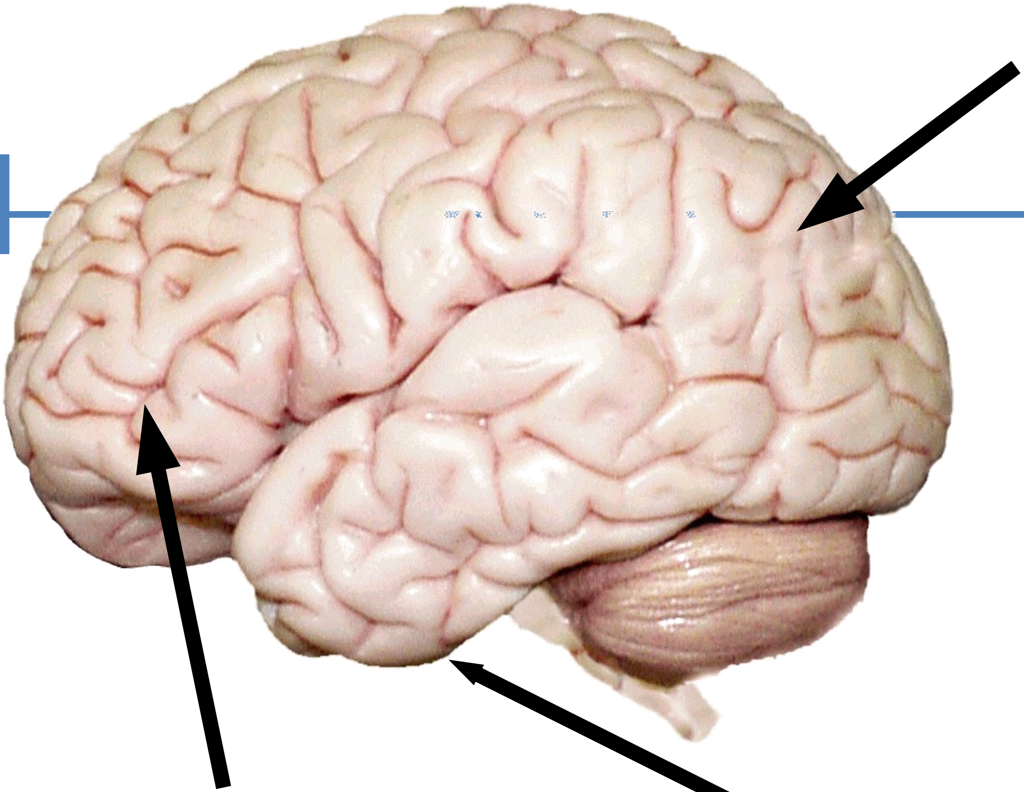


Central Processing Unit (CPU)

- The "operation/action" part of the computer's brain
  - Basically controls the information / data in a computer
- Perform instructions
  - Arithmetic operations
  - Logic operations
  - Decisions
- The instructions it understands are much simpler and fine-grained than those we have seen in previous examples

Memory

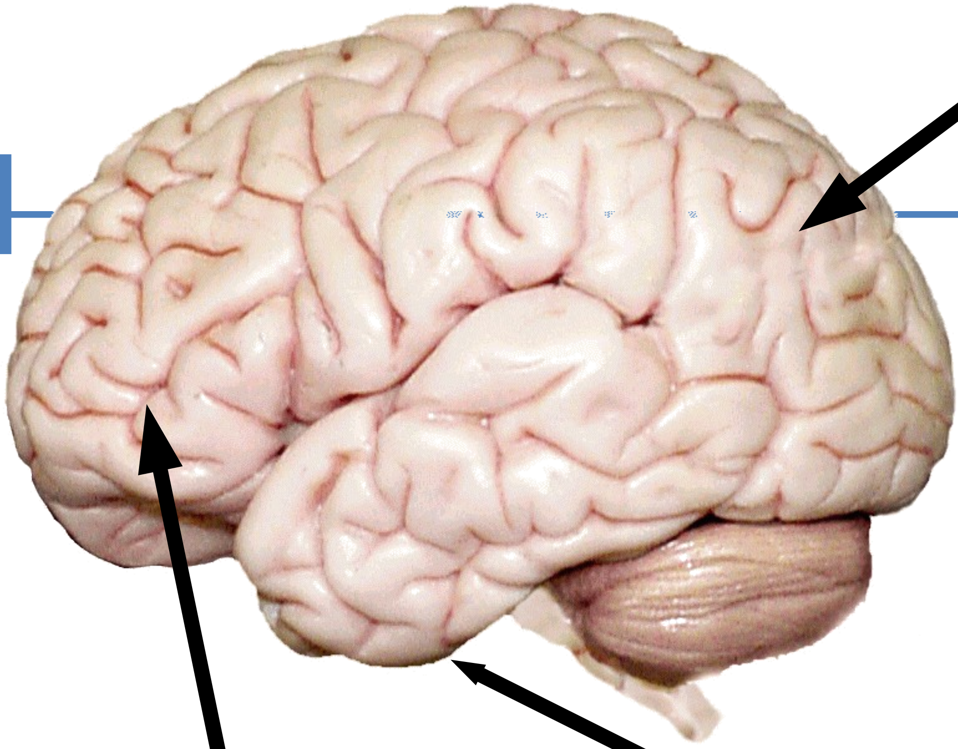
- Memory holds the data
  - Think of a filing cabinet
- Main memory: Most of it is called RAM, which stands for **R**andom-**A**ccess **M**emory
  - Data has to be in main memory so that the CPU can access it
  - Volatile: its contents are lost when the program terminates or when the computer shuts down
- Secondary storage: Hard drive / CD / DVD / Blu-Ray disc / USB memory stick
  - Persistent: its contents will not be lost when the computer shuts down
  - This is where you keep the data for long-term storage
  - Secondary storage has much slower access times than main memory



taking notes; packing a box

remembering what Dan said 2 minutes ago

remembering your name



taking notes;
packing a box
(CPU)

remembering what
Dan said 2 minutes
ago **(RAM)**

remembering your
name **(secondary
storage)**

daydreaming

remembering what Dan said 2 minutes ago

How does a computer store things?

- Computer memory is electronic. It's just a bunch of wires!
-
- All it can recognize is “on” (current goes through) and “off” (no current goes through)
- Using many of these on/off “switches” together, we can *encode* many things.
-
- For example, we can store whether it is morning or afternoon using the following encoding:
- “if the 1st switch is on, then it must be PM. If the 1st switch is off, then it must be AM”

Storing whether it is afternoon or morning

- Pick one electrical switch in memory.
- Whenever it is “on” it is AM
- Whenever it is “off” it is PM
-
- A computer stores billions or trillions of these “switches” By combining many of these, we can control many things.
-
- Note: This is just an example. Your computer probably stores this in an entirely different way.

Encoding the day of the week

- How could we encode the day of the week?
-
- If we just use 1 switch, there will not be enough information.
-
- How many “switches” will we need?

Storage is Exponential

- In general, if there are n possible values to store, we can encode it using
-
- $\log_2(n)$ "switches"
-
- Of course, there is no such thing as a fraction of a switch, so we will always have to round up.
-
- Put another way, if we have n switches, we can store 2^n values

Bits = Switch

- 1 “bit” is the same thing as a “switch”
- It has one or two values “on” or “off”
- For simplicity of notation, we will often just refer to these as 1 (on) and 0 (off)
-
- If you like, you could call them “true/false,” “yes/no,” “oui/non,” or “cats/dogs”

Byte = 8 bits

- A byte is simply 8 bits
-
- Question: How many possible values can we store in a byte?

Other Memory Units

- A kilobyte is 2^{10} bytes (1024 bytes)
- A megabyte is 2^{10} kilobytes (1024 bytes)
- Strangely, a gigabyte is just 1,000,000,000 bytes
-

Storing a number in a computer

- We are used to storing numbers in what is known as “base 10”
-
- What this means, is that every single digit has 10 possible values
-
- 0,1,2,3,4,5,6,7,8,9

Storing a number in a computer

- When we look at a base 10 number, we think of each digit as representing different amounts
-
- 8415
-
- really is:
- 5 ones
- 1 ten
- 4 hundreds
- 8 thousands
- 0 ten-thousands
- 0 hundred-thousands
- ……..

Storing a number in a computer

- When we look at a base 10 number, we think of each digit as representing different amounts
-
- 8415
-
- really is:

5 ones	= 5 * 10^0
1 ten	= 5 * 10^1
4 hundreds	= 5 * 10^2
8 thousands	= 5 * 10^3
0 ten-thousands	= 5 * 10^4
0 hundred-thousands	= 5 * 10^5

- ……..

Storing a number in a computer

- But what is special about 10?

Storing a number in a computer

- But what is special about 10?
-
- A reasonable conclusion to make is that we chose to use 10 digits since we have 10 fingers. This makes it a more natural counting system.

Storing a number in a computer

- Considering that computers think in a sequence of on/off switches, what would be a natural way to count in a computer?

Storing a number in a computer

- Considering that computers think in a sequence of on/off switches, what would be a natural way to count in a computer?
-
- Base 2 (a.k.a. binary)
-
- In this case we will have 2 choices of digits :
-
- 0 and 1

Storing a number in a computer

- 111001111
-
- We can now do the same thing to this number:
-

1 ones	= 1 * 2^0
1 twos	= 1 * 2^1
1 fours	= 1 * 2^2
1 eights	= 1 * 2^3
0 16s	= 0 * 2^4
0 32s	= 0 * 2^5
1 64s	= 0 * 2^6
1 128s	= 0 * 2^7
1 256s	= 0 * 2^8

Storing a number in a computer

- The following number is in base 2. What would it represent in base 10?
-
- 10011

Storing a number in a computer

- The following number is in base 2. What would it represent in base 10?
-
- 10011
-
- = 1 * 2^0 + 1*2^1 + 1 * 2^4 = 19

Other counting systems

- We can do this with many other numbers, although we don't do it as often:
-
- What is
-
- 10011 in base 3?

Other counting systems

- We can do this with many other numbers, although we don't do it as often.
-
- Ex: Hexadecimal : 16 digits (0-9 plus a-f)
-
- What is
-
- 10011 in base 3?
-
- 1 * 3^0 + 1 * 3^1 + 1 * 3^4 = 85

Converting a number to binary

- To convert a number from base 10 to binary, you can use a “greedy” algorithm:
-
- Basically, try to take as many from the bigger columns as possible (this works for going to other bases as well)

Example: Convert 523 to binary

- Start with big powers of 2:
-
- 2^10 = 1024 --->this doesn't fit
- 2^9 = 512 ---> this fits. So we write a 1 in the 2^9 column (i.e. the 10th column from the right). Subtracting this, we are left with 9
- 2^8 = 256-->doesn't fit into 9
- 2^7 = 128, 2^6=64,2^5=32,2^4=16, doesn't fit
- 2^3 = 8 ---> fits into 9, write a 1 in the 2^3 column. Left with 1
- 2^2, 2^1--> don't fit.
- 2^0 fits.

Example: Convert 523 to binary

- 1000001011

Converting from arbitrary base to another

- What if I wanted to convert a base-n number to base-m
-
- A good way to do this is to first convert base-n to base-10 and then convert the base-10 number to base-m

Arithmetic in other bases

- To do arithmetic in other bases, you use the same procedures that we learned in kindergarten.
-
- The only difference is that you have to remember the possible digits are different.
-
- This means, for example, that if you are adding numbers in binary, and you get “2” that you actually have to write “10” ---> But this means you will most likely have to carry a number!

Part 3: Programming Languages

Programming Languages (1)

- We need to expresses our ideas in a form that a computer can understand: a program
- A *programming language* specifies the words and symbols that we can use to write a program
  - e.g. “red” belongs to English; “rouge” belongs to French
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*

–e.g. “Banana red and” in not a valid statement in English.

Programming Languages (2)

- Computers are very intolerant of incorrect programming language statements
  - Humans are much more tolerant of incorrect natural language statements
    - You understand “The kiten is cute” even though kitten is mispelled.

Syntax and Semantics

- The *syntax rules* of a language define what words and symbols are valid in this language, and how they can be combined to make a valid program
  - "The kiten is cute" is not **syntactically** correct.
- The *semantics* of a program statement define what those words, symbols, and statements mean (their purposes or roles in a program)
    - "Banana red and." is not **semantically** correct.

Machine Language

- Each instruction that a CPU understands is represented as a different series of bits
  - The set of all instructions that a CPU understands directly forms the *machine language* for that CPU
- **Each CPU type understands a different machine language**

–In other words, for each different model of CPU, a given series of bits could mean a different instruction

- For example, on an x86-compatible CPU (Intel, AMD), the series of bits `10101010` could mean `ADD`, while on a PowerPC CPU (old Macs, PlayStation 3) it could mean `LOAD`

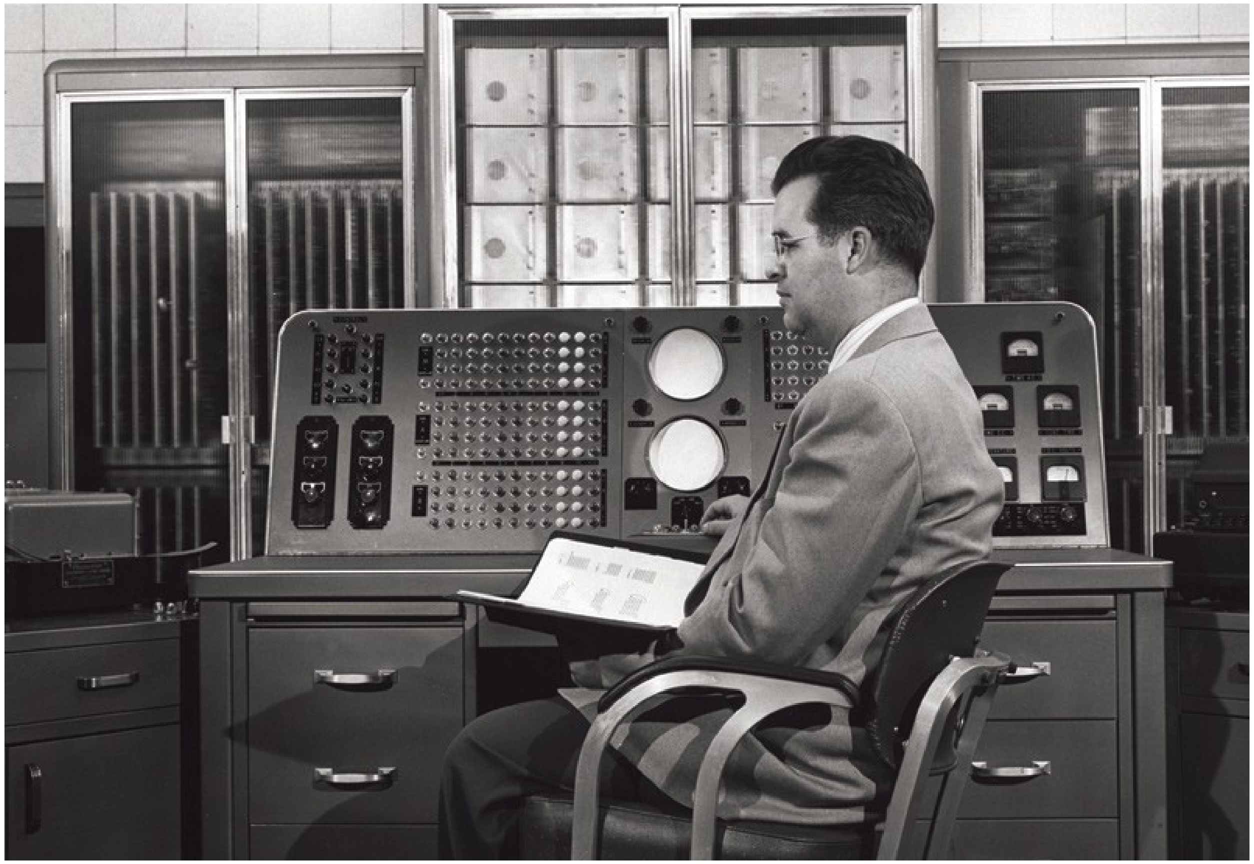
Machine Language Example

•Here are the first 20 bytes of a machine language program that:
–asks the user to enter an integer value using the keyboard
–reads this value from the keyboard
–adds one to this value, and
–displays the new value to the screen

```
01111111 01000101 01001100 01000110 00000001
00000001 00000001 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000
00000000 00000010 00000000 00000011 00000000
```

More the 6500 bytes in total!

Do you think it would be fun or easy to write a program in



Machine Language Disadvantages

•Very tedious and confusing: machine language is extremely difficult for humans to read

•Error-prone

–If you change one bit from 1 to 0 (or vice-versa), or forget a bit, your program's behavior will likely be not even close to what you expected

–Moreover, errors are hard to find and correct

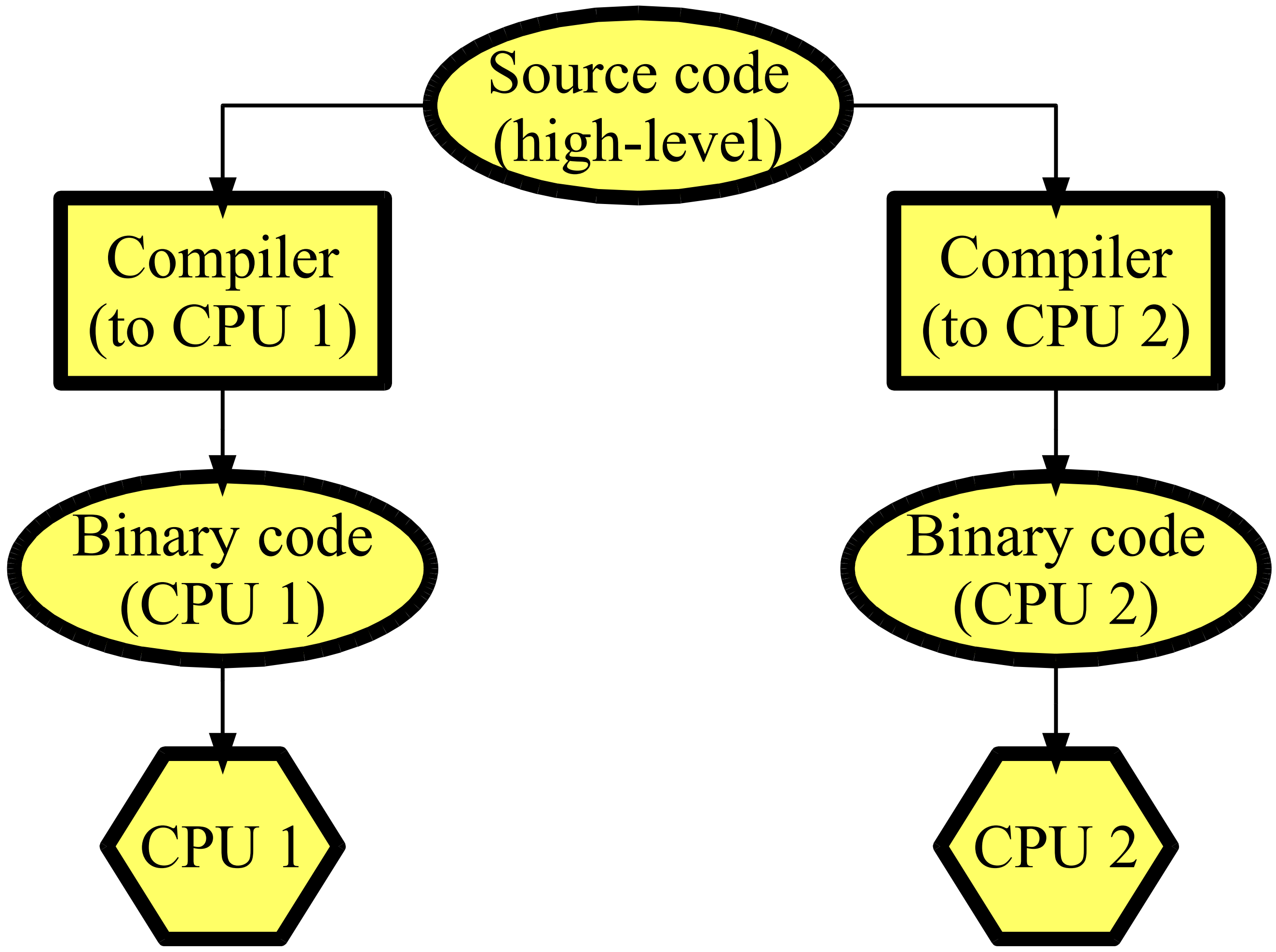
•Programs are not *portable*

–Running the program on a different processor or CPU requires a complete rewrite of the program

High-Level Languages (1)

- To make programming more convenient for humans, *high-level languages* were developed
- No CPU understands high-level languages directly
  - Programs written in these languages must all be translated in machine language before a computer can run them (that's what a **compiler** is for)
- Basic idea:
  - Develop a language that looks like a mix of English and mathematical notation to make it easier for humans to read, understand, and write it
  - For each CPU type, develop a program that translates a program in high-level language to the corresponding machine language instructions (**a compiler)**

Compilers


Source code (high-level)
Compiler (to CPU 1)
Compiler (to CPU 2)
Binary code (CPU 1)
Binary code (CPU 2)
CPU 1
CPU 2

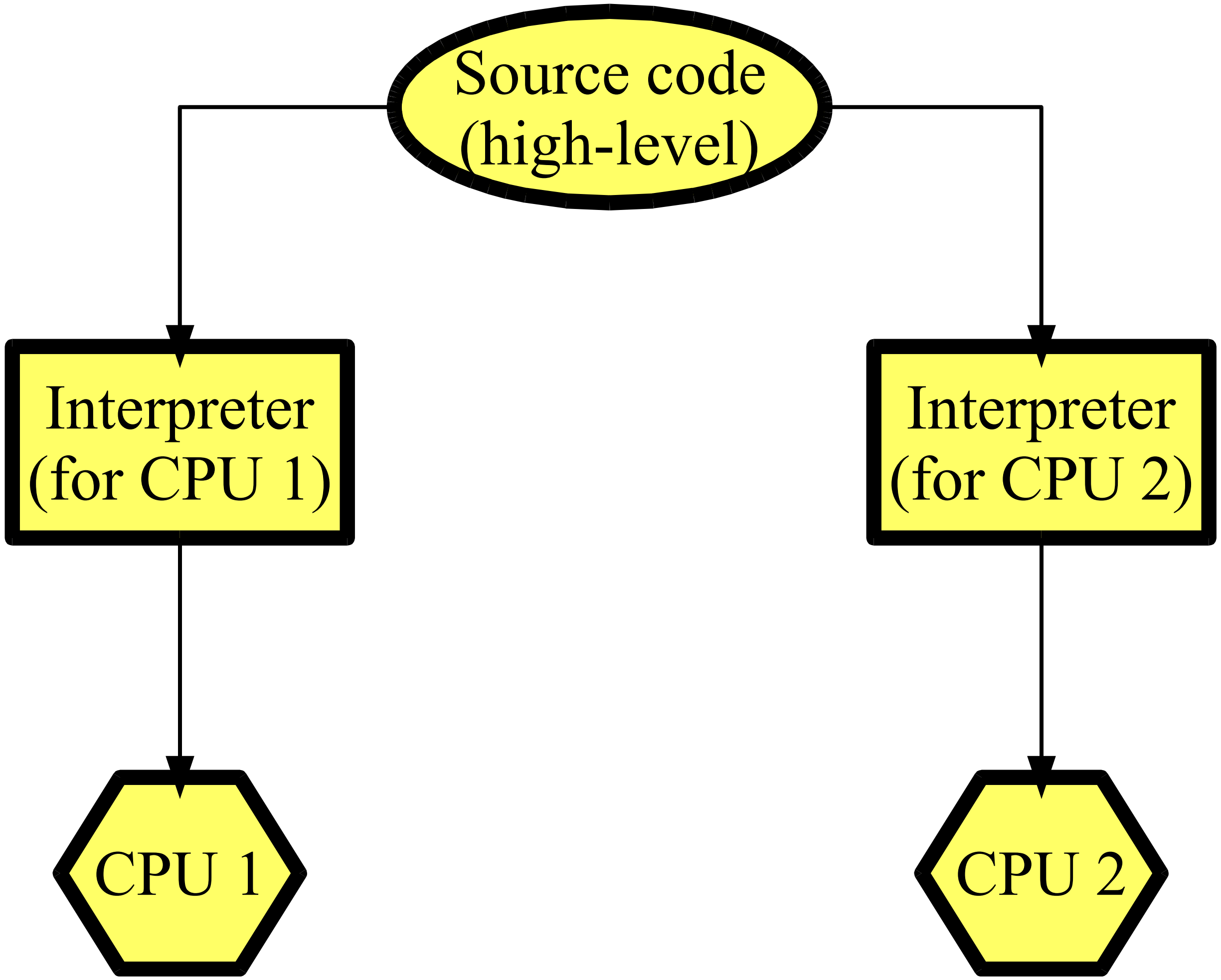
Interpreters (1)

•An *interpreter* is another kind of program. It takes source code and translates it into a target language

–However, the target language instructions it produces are executed **immediately**

–**No executable file is created**

Interpreters (2)



Java combines a compiler with an interpreter

- Java **compiler** (javac, included in JDK 6) takes source and translates it into **bytecode**

$$\begin{tikzcd}\text{foo.java (Java)} \arrow[r, "\text{javac}"] & \text{foo.class (bytecode)}\end{tikzcd}$$

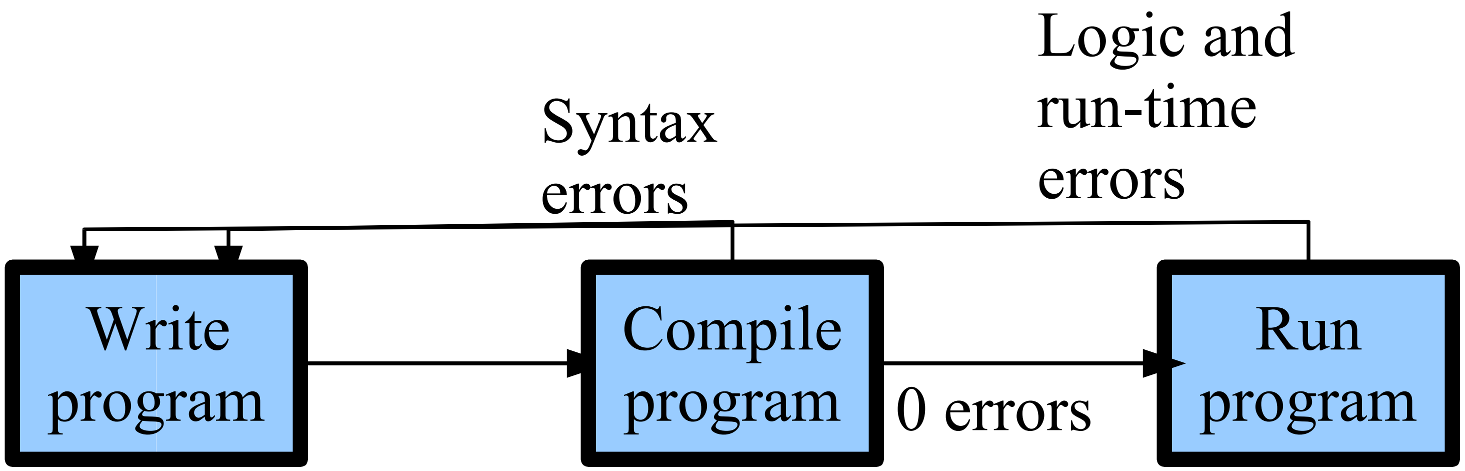
foo.class can than be executed using an **interpreter**, the Java Virtual Machine (JVM)

Programming Errors

- A program can have three types of errors
- **Compile-time errors:** the compiler finds problems with syntax and other basic issues
- **Run-time errors:** a problem occurs during program execution, and causes the program to terminate abnormally (or *crash*)
  - Division by 0
- **Logical errors:** the program runs, but produces incorrect results

```
• celcius = (5.0 / 9.0) * fahrenheit - 32;
    // Incorrect equation; should be
    // (5.0 / 9.0) * (fahrenheit – 32)
```

Development Life Cycle



•Errors may take a long time to debug!

–**Important Note**: When you compile for the first time and see 150 errors, do not despair. Only the first 1 or 2 errors are relevant. Fix those and compile again. There should be fewer errors (like 50). Repeat until there are no more errors

Exercises to practice this at home

- A) Practice breaking the following tasks into smaller pieces. Make sure the pieces are small enough that you can manage them.
- 1)Choosing what channel to watch on TV
- 2)Writing a 5 paragraph essay
- 3)Studying for an exam
- 4)Arguing a speeding ticket in court
-
- B) Look at the resources on the course website and try to compile the HelloWorld program
- Note: The file MUST be called HelloWorld.java (case-sensitive) or else the program will not compile
- C)Take any number in base 10 and convert it to several other bases.

Next Class

- How to store more complex operations in memory using variables
-
- How to group complex operations together using methods