Introduction

Computer Science
Computer History
Java

Computer Science

- Fundamental areas of CS:
  - Theory (overlaps with some math): csc014, 24, 120, 161
    - Logic and abstract algebra
    - Algorithms
    - Theory of computation
  - Software: csc15, 16, 155, and almost all
    - Programming languages
    - Computer architecture
    - Operating systems
    - Compilers
  - Hardware (overlaps with some Computer & Electrical Eng): 110, 110A
    - Digital logic
    - Computer organization
## Computer science: advanced areas and applications

- Data bases, digital libraries, multimedia
- Distributed systems
- Real-time systems
- Parallel algorithms
- High-performance computer architectures
- Robotics
- Artificial Intelligence
  - Knowledge representation and acquisition
  - Natural language processing
  - Computer Vision
  - Neural networks
- Computational biology
- Computer graphics
- Quantum computing
- Molecular computing, optical computing

## Professional organizations, Student memberships

- **ACM**, Association for Computing Machinery  
  [http://www.acm.org](http://www.acm.org)
- **IEEE**, Institute of Electrical and Electronic Engineers,  
  [http://www.ieee.org](http://www.ieee.org)
- **AAAI**, American Association for Artificial Intelligence,  
  [http://www.aaai.org](http://www.aaai.org)
Focus of the Course

- Object-Oriented Software Development
  - problem solving
  - program design, implementation, and testing
  - object-oriented concepts
    - classes
    - objects
    - interfaces
    - inheritance
    - polymorphism
  - the Java programming language

Fundamentals of CS

- Introduction:
  - computer history
  - components of a computer, how those components interact
  - how computers store and manipulate information
  - programming and problem solving principles
  - programming languages
  - Java

- Read Ch 1
Hardware and Software

- **Hardware**
  - the physical, tangible parts of a computer
  - keyboard, monitor, disks, wires, chips, etc.

- **Software**
  - programs and data
  - a *program* is a series of instructions

- **A computer requires both hardware and software**

- **To develop hardware and software one needs theory too**

- **Each is essentially useless without the other**

Computer History

- **Abacus, calculi: possibly 2000BC – XVII**

- **Automatic calculating devices invented XVII**
  - **Blaise Pascal, 1642,** *automatic adding machine*
    - Mechanical, used counting wheels to represent numbers
  - **Gottfried Leibnitz, 1671,**
    - extended Pascal’s machine to do automatically multiplication and division.
  - **4-function desk calculators**
    - used depressible keys for data and operation entry.
    - Punch the operation and the numbers to be used, one computation at a time. For another computation, again: punch the numbers and the operation, get the result, etc.
    - The calculators perform only a *single arithmetic operation at a time.*
Computer History

- First attempt to build a computing machine capable of automatic multi-step calculations, Difference Engine, 1823, Charles Babbage
  - To computes table of math functions.
  - The input data may change, but the sequence of operations not.

The Analytical Engine, Babbage

- Based on a condition encountered during a calculation, the sequence of operations that the machine executes can change (conditional branching)
  - Technology: mechanical, decimal counting wheels
  - Computer organization:
    - "the store" – main memory, to store data and temp results
    - "the mill" - central processing unit (CPU), gets data, executes instructions, and stores results
  - Computer program: specifies the sequence of instructions (operations) to be preformed
    - Physically, the program for the Analytical Engine was a chain of punch cards
From Analytical Engine to Contemporary Computers

Central Processing Unit executes program commands

Main Memory storage area for data that are in active use

AE, Input / Output Devices

I/O devices facilitate user interaction

Main Memory

Card puncher

Punch card reader
General Purpose Computers

- First built century later, 1930-1940
  - Technology: electro-magnetic relays
    - two stable states: on (1) and off (0)
  - Germany: Z3, Zuse.
    - Introduces the *Binary* number system for use by computers

Binary Numbers

- A single binary digit (0 or 1) is called a *bit*
- Devices that store and move information are cheaper and more reliable if they have to represent only two states
- A single bit can represent two possible states, like a light bulb that is either on (1) or off (0)
- Permutations (sequences) of bits are used to store values
### Bit Permutations

<table>
<thead>
<tr>
<th>1 bit</th>
<th>2 bits</th>
<th>3 bits</th>
<th>4 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>000</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>0001</td>
</tr>
<tr>
<td>10</td>
<td>010</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>11</td>
<td>011</td>
<td>0011</td>
<td>0100</td>
</tr>
<tr>
<td>100</td>
<td>0100</td>
<td>0100</td>
<td>1000</td>
</tr>
<tr>
<td>101</td>
<td>0101</td>
<td>0110</td>
<td>1100</td>
</tr>
<tr>
<td>110</td>
<td>0110</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>111</td>
<td>0111</td>
<td>1110</td>
<td>1110</td>
</tr>
</tbody>
</table>

Each additional bit doubles the number of possible permutations.

### Bit Permutations

- Each permutation can represent a particular item
- There are $2^N$ permutations of $N$ bits
- Therefore, $N$ bits are needed to represent $2^N$ unique items

How many items can be represented by

- 1 bit ?
- 2 bits ?
- 3 bits ?
- 4 bits ?
- 5 bits ?
### Representing Text Digitally

- For example, every character is stored as a number, including spaces, digits, and punctuation.
- Corresponding upper and lower case letters are separate characters.

![Image of character representation](image)

### Digital Information

- Computers store all information digitally:
  - numbers
  - text
  - graphics and images
  - video
  - audio
  - program instructions
- In some way, all information is *digitized* - broken down into pieces and represented as numbers.
Digital Images

Computer History: Generations

- First generation electronic computers
  - based on vacuum tube technology
  - First one: 1942 Iowa State University, John Atanasoff,
    - relatively small, 300 tubes
    - could solve a system of 30 linear equations.
    - It was never used, actually.
  - ENIAC, 1946, first general purpose electronic computer that was actually used extensively
ENIAC Then, UPenn museum

ENIAC on a chip, U Penn
Computer Generations, I: Eniac

- **Huge**: 18000 vacuum tubes, about 30 tons
- Vacuum tube switching circuits capable to do *addition, subtraction, multiplication, division, square root.*
- **Decimal** arithmetic
- Main memory: 20 ten-digit vacuum tube registers, 3’ long
- Fast CPU
- Contradiction:
  - main memory and CPU are fast, high-speed vacuum tube registers,
  - Slow program entry
  - programmed by setting manually thousand of switches, and by plugging and unplugging large number of cables.
- How do you modify such a program? How does one understand what the program is supposed to do?
- Main rule of computer programming: programs should be easy to understand, maintain and modify.

Computer Generations, I: EDVAC

- **VON NEUMANN** computer organization
- Stored-program concept
  - Before program is executed it is loaded into the main memory of the comp
- fixed word size: transfer whole words
- binary representation.
### Memory: holds data and program

- Main memory is divided into many memory locations (or cells called bytes).
- Each memory cell has a numeric address, which uniquely identifies it.

### Storing Information

- Each memory cell stores a set number of bits (usually 8 bits, or one byte).
- Large values are stored in consecutive memory locations.
- Bytes groups in words. Whole words transferred.
Storage Capacity

- Every memory device has a storage capacity, indicating the number of bytes it can hold.
- Capacities are expressed in various units:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Symbol</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobyte</td>
<td>KB</td>
<td>(2^{10} = 1024 = 1K)</td>
</tr>
<tr>
<td>megabyte</td>
<td>MB</td>
<td>(2^{20}) (over 1 million)</td>
</tr>
<tr>
<td>gigabyte</td>
<td>GB</td>
<td>(2^{30}) (over 1 billion)</td>
</tr>
<tr>
<td>terabyte</td>
<td>TB</td>
<td>(2^{40}) (over 1 trillion)</td>
</tr>
</tbody>
</table>

Main memory, CPU, I/O and Secondary storage

VN architecture is the one typical for the contemporary computers.

Information is moved between main memory and secondary memory as needed, and between Main memory and CPU.

Secondary memory devices provide long-term storage.
Memory

- Main memory is volatile - stored information is lost if the electric power is removed.
- Secondary memory devices are nonvolatile.
- Main memory and disks are direct access devices - information can be reached directly.
- The terms direct access and random access often are used interchangeably.
- A magnetic tape is a sequential access device since its data is arranged in a linear order - you must get by the intervening data in order to access other information.

The Central Processing Unit

- The CPU contains:
  - Arithmetic / Logic Unit: Performs calculations and makes decisions.
  - Control Unit: Coordinates processing steps.
  - Registers: Small storage areas.
Computing cycle

- Before program is executed it is loaded into the main memory of the computer.
- This is the cycle in which the computer works
  
  \[
  \text{REPEAT}
  \]
  
  { 
  \text{fetch the next instruction (put it from the memory into the CPU)}
  
  \text{decode the instruction}
  
  \text{fetch the operands}
  
  \text{execute the instruction}
  
  \text{store the result in main memory}
  
  \}
  
  \text{UNTIL the instruction HULT encountered}

The Central Processing Unit

- A CPU in contemporary computers is on a chip called a \textit{microprocessor}
- It continuously follows the \textit{fetch-decode-execute cycle}:

  Retrieve an instruction from main memory

  \begin{tikzpicture}
    \node (fetch) at (0,0) {fetch};
    \node (execute) at (1,-1) {execute};
    \node (decode) at (1,-2) {decode};
    \path[->] (fetch) edge (execute);
    \path[->] (fetch) edge (decode);
    \path[->] (execute) edge (decode);
  \end{tikzpicture}

  Carry out the instruction
  Determine what the instruction is
Computer Generations, I: cont

- The early programs were written in *machine language*, in binary
  - Difficult to write, understand, modify, debug.

- Significant improvement was the introduction of the *assembly language*
  - Uses mnemonics (symbolic names) for the instructions
  - Easier to remember by people
  - CPU does not understand such a language.
  - A special program *assembler* translates a program from assembly to a machine language before the program is executed.

Computer Generations, II

  - High-level programming languages (HLPL) evolved late 50ies.
- People write programs in source code (in HLPL)
- Computers can execute programs in machine language
- COMPILER: translates the whole program from source code to object code (typically machine code)
- LINKER: links object code and precompiled libraries and creates executable code
- INTERPRETER: translates instructions and sends for execution one by one
- JAVA: java compiler compiles java program into object code (java byte code), then java interpreter interprets the object code and executes
Language Levels

- Programming language levels:
  - machine language
  - assembly language
  - high-level language

- Each type of CPU has its own specific *machine language*

Computer Generations, III, IV

- III:
  - Semiconductor technology, integrated circuits;
  - I/O processor;
  - multiprogramming;
  - OPERATING SYSTEM: optimizing the performance, controlling the resources. We will use UNIX/LINUX in class.
  - OS user interface (how people interact with the OS). IV:
  - LSI,VLSI integrated circuits
  - 1970ties, microcomputers, microprocessors
  - 1981, IBM PC, Intel 8088, MS-DOS;
  - 1985, Windows, …
## Software Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
</table>
| System software          | - OS  
                          - System utilities                                                                                                                                 |
| Application program      | - generic term for any other kind of software  
                          - word processors, missile control systems, games, any program you write                                                                 |

## Programming & Problem Solving

- The purpose of writing a program is to solve a problem
- The general steps in problem solving are:
  - Understand the problem
  - Identify input and output
  - Design a strategy for solving the problem — algorithm.
    - dissect the problem into manageable pieces
    - independent of high level prog language
    - Donald Knuth: *The art of programming is the art of organizing complexity*
  - Consider alternatives to the solution and refine it
  - Implement the algorithm in HLPL: write a program
  - Test and debug the program
Problem Solving

- Many software projects fail because the developer didn't really understand the problem to be solved
- We must avoid assumptions and clarify ambiguities
- As problems and their solutions become larger, we must organize our development into manageable pieces
- This technique is fundamental to software development
- We will dissect our solutions into pieces called classes and objects, taking an object-oriented approach

Java

- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- The Java programming language was created by Sun Microsystems, Inc.
- It was introduced in 1995 and its popularity has grown quickly since
- It is an object-oriented language
Java Program Structure

- In the Java programming language:
  - A program is made up of one or more classes
  - A class contains one or more (data declarations) and methods
  - A method contains program statements

- These terms will be explored in detail throughout the course

- A Java application always contains a method called main

Java program example

- Program to print a message on the screen

```java
//*****************************************
// File: Firstclass.java
// First java example
// CSC015-02
// Gerda Kamberova, 9/2/2002
//*****************************************
public class Firstclass {
    public static void main(String[] args)
    {
        System.out.println("Hello there!");
    }
}
```
Java Program Structure

// comments about the class
public class MyProgram
{
    class header
    
    class body

    Comments can be placed almost anywhere. But they must clarify the program, not clutter it.
}

Java Program Structure

// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
        method header
        
        method body
    }
}

43

44
Programming Languages

- A program must be translated into machine language before it can be executed on a particular type of CPU
- This can be accomplished in several ways
- A compiler is a software tool which translates source code into a specific target language
- Often, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

Java Translation

- The Java compiler translates Java source code into a special representation called bytecode
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an interpreter, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be architecture-neutral
Java Translation

Compiling and executing in JDK

In a terminal (shell):

- **Compiling Java program**
  
  ```
  javac Firstclass.java,
  ```
  
  - If messages appear on screen that means there are compiler (syntax) errors. Go back in editor, correct, and compile again till no errors occur
  - Result of the compilation: `Firstclass.class`

- **Executing (running) compiled Java program**
  
  ```
  java Firstclass
  ```
  
  Result of running `Firstclass.java` program
  
  - Message on the screen:
    
    `Hello there!`
Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program.
- The *semantics* of a program statement define what that statement means (its purpose or role in a program).
- A program that is syntactically correct is not necessarily logically (semantically) correct.
- A program will always do what we tell it to do, not what we *meant* to tell it to do.

Comments

- Comments in a program are called *inline documentation*.
- They should be included to explain the purpose of the program and describe processing steps.
- They do not affect how a program works.
- Java comments can take three forms:

```
// this comment runs to the end of the line

/* this comment runs to the terminating symbol, even across line breaks */
```
**White Space**

- Spaces, blank lines, and tabs are called *white space*.
- White space is used to separate words and symbols in a program.
- Extra white space is ignored.
- Programs should be formatted to enhance readability, using consistent indentation.

**Errors**

- A program can have three types of errors.
- The compiler will find syntax errors and other basic problems (*compile-time errors*).
  - If compile-time errors exist, an object code version of the program is not created.
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*).
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*).
Basic Program Development

- Edit and save program
- Compile program
- Execute program and evaluate results

Errors

53