Computer Programming

Introduction to Computer Programming

Willy Picard, Jacek Chmielewski
Department of Information Technology
The Poznan University of Economics
<{picard, jchmiel}@kti.ae.poznan.pl>

Agenda

- Lecture Goal(s)
- From Assembler to Java
- Programming “styles”
- Interpret vs compile
- Conclusions

Lecture Goal(s)

Overall Goal

- To introduce fundamental concepts of computer programming
- To provide programming knowledge about Java

Lectures Overview

Fundamental Concepts

- 1: Introduction
- 2: Basic data structures & Statements
- 3: Object-oriented programming I
- 4: Object-oriented programming II
- 5: Object-oriented programming III
- 6: Complex data structures
- 7: Threads & Exception handling

Lectures Overview

Java

- 8: Summarizing example
- 9: Standard packages
- 10: GUI – AWT
- 11: GUI – Swing
- 12: IO programming
- 13: Network programming
- 14: Java archives
- 15: Conclusions

Today's Goal

Introduce fundamental concepts of computer programming

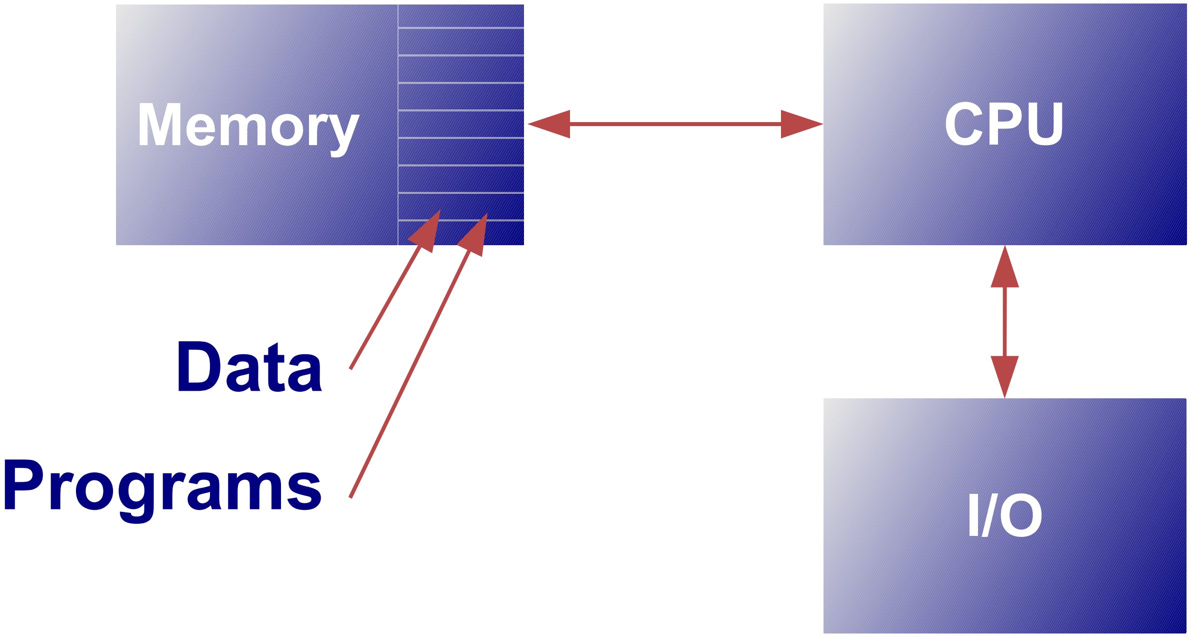
LECTURE GOALS

From Assembler to Java

Princeton vs Harvard

- Harvard architecture
  - Data and programs stored separately
  - ENIAC 1946
- Princeton architecture
  - Data and programs stored in the same way
  - A.k.a the von Neumann architecture

The Princeton Architecture



© Willy Picard

Advantages

- Many jobs for a given computer
  - Not hardwired programs
- Programs generated by programs
  - From source (data) to code (program)
- Self-modifying code

Assembler

- CPU Instructions Set Architecture
  - Instructions
  - Resources accessible to the instructions
    - Registrers
    - Functional units
    - Memory
    - I/O devices
- Assembler
  - Symbolic names

Assembler Example

- x := y + z

- `movl  y,%eax`
- `movl  z,%edx`
- `leal  (%edx,%eax),%ecx`
- `movl  %ecx,x`

Assembler Pros and Cons

- Low level control
- Speed

- Code not portable
- Too many details
- Hard to maintain
- Compilers better at producing large code

Programming languages are a necessity!!!

Java

- Java Virtual Machine
  - Java Virtual Machine specification
  - Java language
  - Java development tools
  - Java Libraries
    - GUI
    - Network
    - IO

Java Example

- Build a new window

```
import javax.swing.*;
...
JFrame frame = new JFrame(“A Window”);
frame.pack();
frame.setVisible(true);
```

Assembler Example

- x := y + z

```
movl  y,%eax
movl  z,%edx
leal  (%edx,%eax),%ecx
movl  %ecx,x
```

Java Pros and Cons

- Low level control
- Speed

- Code portable
- Many hidden details
- Relatively easy to maintain

Let's use high-level programming languages

Programming “Styles”

Language Classification

Declarative Languages	
Functional	*Lisp, XSLT*
Logic programming	*Prolog*
Procedural Languages	
Imperative	*Fortan, Pascal, Basic, C*
Object-Oriented	*C++, ADA, Java*

Declarative Languages

- Relationships between variables in terms of
  - Functions
    - If student is attending “Computer Programming”, her/his satisfaction equals 1, otherwise 0.
    - $f_{satisfaction} = \begin{cases} 1, & \textit{if attending "Computer Programming"} \\ 0, & \textit{otherwise} \end{cases}$
  - Inference rules
    - 1. Either student is attending “Computer Programming” or student is sad
    - 2. The student is not sad,
    - Then student is attending “Computer Programming”
    - Formally: ((A OR B) & not B) => A

Procedural Languages

- Specify explicit sequences of steps to follow to produce a result
  - Manipulation of the state of the computer system
  - Potential side effects

```
private static int counter=0;
public int getCounterValue(){
    return counter++;
}
```

Iteration

- Repetition of a sequence of instruction
  - a set of initial conditions
  - an iterative step
  - a termination condition
- Loops in procedural languages

Iteration Example

- Factorial
- n! = 1 x 2 x 3 x 4 x ... x n, if n ≠ 0, 1 otherwise

```
public int factorial(int n) {
   if (n == 0)
      return 1;
   int factorial = 1;
   for (int i = 1; i<n+1; i++){
      factorial = factorial*i;
   }
   return factorial;
}
```

Recursion

- A function or procedure calls itself
  - a callback to itself
  - a termination condition
- Functional programming

Recursion Example

- Factorial
- n! = 1 x 2 x 3 x 4 x ... x n, if n ≠ 0, 1 otherwise

```
public int factorial(int n) {
    if (n == 0)
        return 1;
    return n*factorial(n-1);
}
```

Recursion vs Iteration

- Recursion and iteration are equivalent
  - Each recursion can be expressed as an iteration and vice-versa
- Recursion cons
  - Uses more resources
  - Often slower because of many procedure calls
- Recursion pros
  - Usually smaller
  - Often better adapted to algorithms
  - Better adapted to some data structures (trees)

Compile vs Interpret

Syntax and Semantics

- Syntax
  - The structural rules of a language that determine the form of a program written in the language
  - e.g. in Java, integer variable names can be followed by two adjacent + symbols
- Semantics
  - The meaning of the various language constructs in the context of a given program
  - e.g. in Java, ‘j = i++;’ means “increment i after assigning its value to j”

Compilation

- Transformation
  - from source code
    - Written by programmer in a programming language
  - to object code
    - Directly executable by a computer
- Syntax and semantics verification
- Generated program
  - executable
  - for a given architecture

Interpretation

- Execution of source code
- Slower than execution of compiled program
- Faster than compilation+execution
  - Well-adapted to prototyping and testing
- Does not rely on a specific architecture
- Controlled environment (the interpreter)

Hybrid

- Two phases
  - Compilation to byte-code
  - Interpretation of byte-code
- Byte-code
  - Optimized and compressed representation of the source code
  - Not machine executable
  - Does not rely on a particular hardware
  - Interpreted by a Virtual Machine

The Java Case

- An hybrid approach
- First, compilation
  - `javac MyProg.java`
  - Creation of `MyProg.class`
- Next, interpretation
  - `java MyProg`
  - `MyProg.class` is interpreted
- Java Virtual Machine
  - Portability

Conclusions

Conclusions

- Many existing programming languages
  - From Assembler
  - To Java
- We chose Java because
  - High-level
  - Portable
  - Widely-used
  - Relatively easy to learn ☺

Example

```
package pl.poznan.ae.compProg;

import java.util.*;

public class Sorter {
  private List _words;

  public void sort(String[] words){
    _words = Arrays.asList(words);
    Collections.sort(_words);
  }
  public String getSortedWords(){
    String sortedString = "";
    for (int i = 0; i< _words.size(); i++){
      sortedString += _words.get(i);
    }
    return sortedString;
  }
 public static void main(String[] args){
    Sorter sorter = new Sorter();
    sorter.sort(args);
    System.out.println(sorter.getSortedWords());
 }
}
```

CONCLUSIONS

See you next week