Lecture 13

Introduction to
High-Level Programming
(S&G, §§7.1–7.6)

From Algorithms to Programs

- Algorithms are essentially abstract things that can have
  - Linguistic realizations (in pseudocode or programming languages)
  - Hardware realizations (in digital circuitry)
What Is a Program?

- Usually, one or more algorithms written in a programming language that can be translated to run on a real machine
- We sometimes call programs *software*

What Is a Programming Language?

- A bit like pseudocode, but with very strict syntax rules
- Examples: Basic, Pascal, C++, Java
From Algorithms to Hardware

Algorithm

Translate (by a human being)

Program

Translate (by another program)

A real computer

The Program Development Process (Data Flow)

Algorithm

Editor

Program in programming language

Compiler

Program in machine’s language

A real computer

Input

Output
The Program Development Process (Control Flow)

Edit \rightarrow Compile \rightarrow Run

Input \rightarrow Syntax errors \rightarrow Output

Syntax errors \rightarrow Runtime errors

Java

• Was developed at Sun Microsystems in the early 1990s
• A general-purpose language but
  – Programs could run in small devices (embedded systems)
  – Internet-ready and Web-ready
  – Standard support for multimedia applications
Compiling and Running a Java Program

- A Java compiler translates a Java program to an equivalent program in byte code.
- A Java virtual machine (JVM) is a program that interprets a Java byte code program to execute it.

```
Java code → Compiler → byte code → JVM → Program outputs
```

Portability

- A Java program translates to a standard byte code.
- A Java byte code can run on any JVM.
- A JVM, and thus a Java program, can run:
  - In a PDA
  - In a desktop computer
  - In a Web browser
Statements in Java: Assignment

Pseudocode:
Set the value of <identifier> to <arithmetic expr>

Java:
<identifier> = <arithmetic expr>;

Examples:
i = i + 1;
force = mass * acceleration;
average = sum / n;

Example Java Statements

// This is a comment.  
// Comments are not executable statements  
// but exist solely for the benefit of human readers
int width = 20;        // Declare an integer variable and set its  
                        // value to 20
int height = 35;

int area;              // Declare an integer variable; its  
                        // default value is 0
area = height * width; // An assignment statement  
                        // Pseudocode: set area to height * width

All variables must be declared before they are used in statements.

Each variable has a data type, such as int, double or String.
Basic Data Types

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Example Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>34, 0, 10000000</td>
</tr>
<tr>
<td>double</td>
<td>3.14, 0.5633333333</td>
</tr>
<tr>
<td>String</td>
<td>&quot;Java rules!&quot;</td>
</tr>
<tr>
<td>char</td>
<td>'a', '9', '%'</td>
</tr>
<tr>
<td>boolean</td>
<td>true, false</td>
</tr>
</tbody>
</table>

Algorithms that use arrays, such as Sequential Search

Given values for Name, \(N_1, \ldots, N_{10}\) and for \(T_1, \ldots, T_{10}\)

Set the value of \(i\) to 1 and set the value of Found to NO

Repeat until either Found = YES or \(i > 10\)

If Name is equal to \(N_i\), then

Print \(T_i\)

Set the value of Found to YES

Else

Increment \(i\)

If (Found = NO) then

Print “Sorry, but the name’s not in the directory”

Stop
Translating to Java

- Here are examples of creating arrays:
  ```java
  int[] years = new int[10]; // a list of 10 integers
  double[] area = new double[2000]; // a list of 2000 real numbers
  String[] name = new String[40]; // a list of 40 strings
  ```

- Here are examples of referencing elements:
  ```java
  firstyear = years[0]; // warning: array elements are numbered starting at zero!
  lastarea = area[1999]; // the last element in the list of area’s
  ```

Statements in Java: Input

Pseudocode:
Ask user to enter a value of `<identifier>`

Java:
```java
<identifier> = Console.readInt(<prompt>);
<identifier> = Console.readDouble(<prompt>);
<identifier> = Console.readChar(<prompt>);
```

Examples:
```java
int speed;
  speed = Console.readInt("Please enter" + " the speed you traveled");
```
Statements in Java: Output

Pseudocode:
Output the value of <identifier>

Java w/ examples:

System.out.println("thanks for the data");
System.out.println("The time for your trip is " + time);
System.out.println("A trip of " + distance + " miles " + "at " + speed + " mph " + "requires " + time + " hours");

Example Problem

Write a program that computes the area of a circle.
From Problem to Algorithm

The algorithm

- uses area = \( \pi r^2 \)
- accepts as input a decimal point number representing the radius of the circle
- computes and displays the area of the corresponding circle

Pseudocode for Program

input radius

area = \( \pi \times radius \times radius \)

output area
Basic Computation

```
radius = Console.readInt ("Radius?");
area = Math.PI * radius * radius;
System.out.println ("Area = " + area);
```

Complete Program

```
public class CircleArea
{
    double radius, area;

    public static void main (String[] args)
    {
        radius = Console.readInt ("Radius?");
        area = Math.PI * radius * radius;
        System.out.println ("Area = " + area);
    }
}
```
Digression: *What Does It All Mean?*

The language machine regulates and adjusts in advance the mode of our possible usage of language through mechanical energies and functions. The language machine is — and above all, is still becoming — one way in which modern technology controls the mode and the world of language as such. Meanwhile, the impression is still maintained that man is the master of the language machine. But the truth of the matter might well be that the language machine takes language into its management and thus masters the essence of the human being.

— Martin Heidegger

Java Methods

```java
// A method is like a little program that performs a complex task
double d = Math.sqrt(25); // d is 5.0
int i = Math.abs(-4); // i is 4
i = Math.pow(2, 10); // i is 1024
System.out.println("Hello there!"); // Displays output in // the terminal window
```

The data in the parentheses are called *parameters*.

Methods receive information in parameters and can return information to the rest of the program.
Structure of Simple Program: The Main Method

public class <name of program>
{
    public static void main (String[] args)
    {
        <declarations>
        <statements>
    } //end of main method
} //end of program

A Java method describes data and an algorithm as a chunk of program code. The main method is run when the program starts up.

Java Control Structures

Conditionals
(if statements)
Syntax of Compound Statements

A compound statement is a sequence of zero or more statements.


document content

Syntax of Selection Statements

if (<Boolean expression>)    // One-way decision
    <statement>

if (<Boolean expression>)    // Two-way decision
    <statement>
else
    <statement>
Semantics of Selection Statements

```plaintext
if (<Boolean expression>)
  <statement>
else
  <statement>
```

Java Control Structures

Iteration/Loops

(while statements and for statements)
Example: Summation

Write a program that computes the sum of the numbers between 1 and 10.

Set counter to 1
Set sum to 0
While counter ≤ 10 do
  Set sum to sum + counter
  Increment counter
Print sum

Example: Summation

Write a program that computes the sum of the numbers between 1 and 10.

```java
int counter = 1;
int sum = 0;
while (counter <= 10)
{
    sum = sum + counter;
    counter = counter + 1;
}
System.out.println (sum);
```
Syntax and Semantics of **while** Statements

while (<Boolean expression>)
<statement>

while (<Boolean expression>)
{
    <statement 1>
    ...  
    <statement n>
}

Count-controlled Loops

// General form
<initialize a counter variable>
while (<test counter for termination condition>){
    <do something>
    <change the value of counter>
}
Count-controlled Loops

// General form
<initialize a counter variable>
while (<test counter for termination condition>){
  <do something>
  <change the value of counter>
}

// Count up
<initialize a counter variable>
while (<counter is less than a limit value>){
  <do something>
  <increase the value of counter>
}

int counter = 1;
while (counter <= 10){
  <do something>
  counter = counter + 1;
}
Count-controlled Loops

// General form
<initialize a counter variable>
while (<test counter for termination condition>){
  <do something>
  <change the value of counter>
}

// Count down
<initialize a counter variable>
while (<counter is greater than a limit value>){
  <do something>
  <decrease the value of counter>
}

int counter = 10;
while (counter > 0){
  <do something>
  counter = counter - 1;
}
Increment and Decrement

```c
int counter = 1;
while (counter <= 10){
    <do something>
    counter = counter + 1;
}

int counter = 10;
while (counter > 0){
    <do something>
    counter = counter - 1;
}
```

Increment and Decrement

```c
int counter = 1;
while (counter <= 10){
    <do something>
    counter++;
}

int counter = 10;
while (counter > 0){
    <do something>
    counter--;
}
```
Designing Correct Loops

- Initialize all variables properly
  - Plan how many iterations, then set the counter and the limit accordingly
- Check the logic of the termination condition
- Update the loop control variable properly

Off-by-One Error

```cpp
int counter = 1;
while (counter <= 10){   // Executes 10 passes
    <do something>
    counter++;
}

int counter = 1;
while (counter < 10){   // Executes 9 passes
    <do something>
    counter++;
}
```
Infinite Loop

```c
int counter = 1;
while (counter <= 10) {
    // Executes 5 passes
    <do something>
    counter = counter + 2;
}
```

```c
int counter = 1;
while (counter != 10) {
    // Runs forever
    <do something>
    counter = counter + 2;
}
```

In general, avoid using `!=` in loop termination conditions.

The `for` Loop

```c
int counter = 1;
while (counter <= 10) {
    <do something>
    counter++;
}
```

```c
for (int counter = 1; counter <= 10; counter++)
    <do something>
```
The **for** Loop

```cpp
int counter = 10;
while (counter > 0) {
    <do something>
    counter--;
}
```

```cpp
for (int counter = 10; counter > 0; counter--)
<do something>
```

### Syntax and Semantics of the **for** Loop

```
for (<initializer>; <termination>; <update>)
<statement>
```

- **Initializer**: Expressions that are evaluated before the loop begins.
- **Termination**: Boolean expressions that are checked before each iteration. If false, the loop ends.
- **Update**: Expressions that are evaluated after each iteration.

The loop body is executed while the termination condition is true.
The Life Cycle of Programs: Waterfall Model

- Analysis
  - Verify
- Design
  - Verify
- Implementation
  - Test
- Integration
  - Test
  - Maintenance

Maintenance

- Programs might have a lifetime of 5-20 years
- Requirements might change, and minor or major changes would be necessary
The Cost of Correcting an Error

Software Development Phase

The Relative Costs of the Phases

Maintenance 68%
Integration 8%
Implementation 8%
Design 8%
Analysis 8%