Chapter 8
(Part 1)

High Level Programming Languages
Layers of a Computing System

Communication

Application

Operating System

Programming

Hardware

Information
Chapter Goals

- Describe the translation process and distinguish between assembly, compilation, interpretation, and execution.
- Name four distinct programming paradigms and name a language characteristic of each.
- Describe the following constructs: stream input and output, selection, looping, and subprograms.
- Construct Boolean expressions and describe how they are used to alter the flow of control of an algorithm.
- . . . Some Hands-On
Compilers

- **Compiler** A program that translates a high-level language program into machine code

- High-level languages provide a richer set of instructions that makes the programmer’s life even easier
Figure 8.1 Compilation process
Interpreters

- **Interpreter** A translating program that translates and executes the statements in sequence
  - Unlike an assembler or compiler which produce machine code as output, which is then executed in a separate step
  - An interpreter translates a statement and then immediately executes the statement
  - Interpreters can be viewed as *simulators*
Java

- Introduced in **1996** and swept the computing community by storm
- **Portability** was of primary importance
- Java is compiled into a standard machine language called **Bytecode**
- A software interpreter called the **JVM** *(Java Virtual Machine)* takes the Bytecode program and executes it
What is a paradigm?

A set of assumptions, concepts, values, and practices that constitute a way of viewing reality
Figure 8.2
Portability provided by standardized languages versus interpretation by Bytecode

(a) A C++ program compiled and run on different systems
Figure 8.2
Portability provided by standardized languages versus interpretation by Bytecode
Programming Language Paradigms

- Imperative or procedural model
  - FORTRAN, COBOL, BASIC, C, Pascal, Ada, and C++

- Functional model
  - LISP, Scheme (a derivative of LISP), and ML
Programming Language Paradigms

- Logic programming
  - PROLOG

- Object-oriented paradigm
  - SIMULA and Smalltalk
  - C++ is as an imperative language with some object-oriented features
  - Java is an object-oriented language with some imperative features
Functionality of Imperative Languages

- **Sequence**  Executing statements in sequence until an instruction is encountered that changes this sequencing
- **Selection**  Deciding which action to take
- **Iteration**  (looping)  Repeating an action

Both selection and iteration require the use of a Boolean expression
Boolean Expressions

- **Boolean expression**  A sequence of identifiers, separated by compatible operators, that evaluates to *true* or *false*

- Boolean expression can be
  - A **Boolean variable**
  - An arithmetic expression followed by a **relational operator** followed by an arithmetic expression
  - A Boolean expression followed by a **Boolean operator** followed by a Boolean expression
Boolean Expressions

• **Variable**  A location in memory that is referenced by an identifier that contains a data value

  Thus, a Boolean variable is a location in memory that can contain either *true* or *false*
Boolean Expressions

- A relational operator between two arithmetic expressions is asking if the relationship exists between the two expressions.

  For example, \( xValue < yValue \)

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>equal to</td>
<td>= or ==</td>
</tr>
<tr>
<td>not equal to</td>
<td>&lt;&gt; or != or /=</td>
</tr>
<tr>
<td>less than or equal to</td>
<td>&lt;=</td>
</tr>
<tr>
<td>greater than or equal to</td>
<td>&gt;=</td>
</tr>
<tr>
<td>less than</td>
<td>&lt;</td>
</tr>
<tr>
<td>greater than</td>
<td>&gt;</td>
</tr>
</tbody>
</table>
Strong Typing

- **Strong typing**  The requirement that only a value of the proper type can be stored into a variable

- **Data type**  A description of the set of values and the basic set of operations that can be applied to values of the type
Data Types

- Integer numbers
- Real numbers
- Characters
- Boolean values
- Strings
Integers

- The **range varies** depending upon how many bytes are assigned to represent an integer value.
- Some high-level languages provide several integer types of different sizes.
- Operations that can be applied to integers are the standard arithmetic and relational operations.
Reals

- Like the integer data type, the range **varies** depending on the number of bytes assigned to represent a real number.
- Many high-level languages have two sizes of real numbers.
- The operations that can be applied to real numbers are the same as those that can be applied to integer numbers.
Characters

- It takes **one byte** to represent characters in the **ASCII character set**
- **Two bytes** to represent characters in the **Unicode character set**
- Our English alphabet is represented in ASCII, which is a subset of Unicode
Characters

• Applying arithmetic operations to characters doesn’t make much sense

• **Comparing characters** does make sense, so the relational operators can be applied to characters

• The meaning of “less than” and “greater than” when applied to characters is “comes before” and “comes after” in the character set
Boolean

• The **Boolean data type** consists of two values: *true* and *false*

• Not all high-level languages support the Boolean data type

• If a language does not, then you can simulate Boolean values by saying that the Boolean value *true* is represented by 1 and *false* is represented by 0
Strings

- A **string is a sequence of characters** considered as one data value.

- For example: **“This is a string.”**
  - Containing 17 characters: one uppercase letter, 12 lowercase letters, three blanks, and a period.

- The operations defined on strings vary from language to language.
  - They include concatenation of strings and comparison of strings in terms of lexicographic order.
• **Declaration** A statement that associates an identifier with a variable, an action, or some other entity within the language that can be given a name so that the programmer can refer to that item by name.
# Declarations

<table>
<thead>
<tr>
<th>Language</th>
<th>Variable Declaration</th>
</tr>
</thead>
</table>
| Ada      | sum : Float := 0; --set up word with 0 as contents  
           | num1: Integer; --set up a two-byte block for num1  
           | num2: Integer; --set up a two-byte block for num2  
           | num3: INTEGER; --set up a two-byte block for num3  
           | ...  
           | num1 := 1; |
| VB.NET   | Dim sum As Single = 0.0F ' set up word with 0 as contents  
           | Dim num1 As Integer ' set up a two-byte block for num1  
           | Dim num2 As Integer ' set up a two-byte block for num2  
           | Dim num3 As Integer ' set up a two-byte block for num3  
           | ...  
           | num1 = 1 |
| C++/Java | float sum = 0.0; // set up word with 0 as contents  
           | int num1; // set up a block for num1  
           | int num2; // set up a block for num2  
           | int num3; // set up a block for num3  
           | ...  
           | num1 = 1; |
Declarations

- **Reserved word**  A word in a language that has special meaning

- **Case-sensitive**  Uppercase and lowercase letters are considered the same
Assignment statement

- **Assignment statement**  An action statement (not a declaration) that says to evaluate the expression on the right-hand side of the symbol and store that value into the place named on the left-hand side.

- **Named constant**  A location in memory, referenced by an identifier, that contains a data value that cannot be changed.
## Constant Declaration

<table>
<thead>
<tr>
<th>Language</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ada</strong></td>
<td><code>Comma : constant Character := ',';</code>&lt;br&gt;    <code>Message : constant String := &quot;Hello&quot;;</code>&lt;br&gt;    <code>Tax_Rate : constant Float := 8.5;</code></td>
</tr>
<tr>
<td><strong>VB.NET</strong></td>
<td><code>Const WORD1 As Char = &quot;&quot;,&quot;c</code>&lt;br&gt;    <code>Const MESSAGE As String = &quot;Hello&quot;</code>&lt;br&gt;    <code>Const TaxRate As Double = 8.5</code></td>
</tr>
<tr>
<td><strong>C++</strong></td>
<td><code>const char COMMA = ',';</code>&lt;br&gt;    <code>const string MESSAGE = &quot;Hello&quot;;</code>&lt;br&gt;    <code>const double TAX_RATE = 8.5;</code></td>
</tr>
<tr>
<td><strong>Java</strong></td>
<td><code>final char COMMA = ',';</code>&lt;br&gt;    <code>final String MESSAGE = &quot;Hello&quot;;</code>&lt;br&gt;    <code>final double TAX_RATE = 8.5;</code></td>
</tr>
</tbody>
</table>
Input/Output Structures

• In our pseudocode algorithms we have used the expressions Read and Write.

• High-level languages view input data as a stream of characters divided into lines.
Input/Output Structures

- The key to the processing is in the data type that determines how characters are to be converted to a bit pattern (input) and how a bit pattern is to be converted to characters (output).

- We do not give examples of input/output statements because the syntax is often quite complex and differs so widely among high-level languages.
A Little Hands On
Hello World

<html>
<body>
<script type="text/javascript">
document.write("Hello World!")
</script>
</body>
</html>
An External JavaScript

```html
<html>
<head>
<script src="xxx.js"></script>
</head>
<body>
</body>
</html>
```
Declaring Variables

You can create a variable with the `var` statement:

```javascript
var strname = some value
```

You can also create a variable without the `var` statement:

```javascript
strname = some value
```

You can assign a value to a variable like this:

```javascript
var strname = "Hello World!"
```

Or like this:

```javascript
strname = "Hello World!"
```
Control Statements

<script type="text/javascript">
//Write a "Good morning" greeting if
//the time is less than 10

var d=new Date()
var time=d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>")
}
</script>
Homework

• Read Chapter Eight, Sections 8.1 – 8.3 (Up to Control Structures)

• “PLAY” with JavaScript
  http://www.w3schools.com/js/js_howto.asp
Mid-Term

- Due Back: **Tonight**
- **No Lateness!!!**
No Class

- There will be **no class on Monday, 10/30**
Good Night

"WHICH CAN'T THEY MAKE A SPELL CHECKER THAT KNOWS HOW TO CHECK SPELLS???