Chapter 6: Programming Languages

- 6.1 Historical Perspective
- 6.2 Traditional Programming Concepts
- 6.3 Procedural Units
- 6.4 Language Implementation
- 6.5 Object Oriented Programming
- 6.6 Programming Concurrent Activities
- 6.7 Declarative Programming
Figure 6.1 Generations of programming languages

- Second-generation: Assembly language
  - A mnemonic system for representing machine instructions
    - Mnemonic names for op-codes
    - Identifiers: Descriptive names for memory locations, chosen by the programmer
Assembly Language Characteristics

- One-to-one correspondence between machine instructions and assembly instructions
  - Programmer must think like the machine
- Inherently machine-dependent
- Converted to machine language by a program called an assembler

Program Example

<table>
<thead>
<tr>
<th>Machine language</th>
<th>Assembly language</th>
</tr>
</thead>
<tbody>
<tr>
<td>156C</td>
<td>LD R5, Price</td>
</tr>
<tr>
<td>166D</td>
<td>LD R6, ShipCharge</td>
</tr>
<tr>
<td>5056</td>
<td>ADDI R0, R5 R6</td>
</tr>
<tr>
<td>30CE</td>
<td>ST R0, TotalCost</td>
</tr>
<tr>
<td>C000</td>
<td>HLT</td>
</tr>
</tbody>
</table>

Copyright © 2012 Pearson Education, Inc.
Third Generation Language

- Uses high-level primitives
  - Similar to our pseudocode in Chapter 5
- Machine independent (mostly)
- Examples: FORTRAN, COBOL
- Each primitive corresponds to a sequence of machine language instructions
- Converted to machine language by a program called a compiler

Figure 6.2  The evolution of programming paradigms
Figure 6.3  A function for checkbook balancing constructed from simpler functions

Inputs: Old_balance Credits Debits

Find_sum Find_sum

Find_diff

Output: New_balance

Figure 6.4  The composition of a typical imperative program or program unit

The first part consists of declaration statements describing the data that is manipulated by the program.

The second part consists of imperative statements describing the action to be performed.

Copyright © 2012 Pearson Education, Inc.
Data Types

- Integer: Whole numbers
- Real (float): Numbers with fractions
- Character: Symbols
- Boolean: True/false

Variable Declarations

```c
float Length, Width;
int Price, Total, Tax;
char Symbol;
```
**Figure 6.5** A two-dimensional array with two rows and nine columns

Scores

Scores (2, 4) in FORTRAN where indices start at one.

Scores [1][3] in C and its derivatives where indices start at zero.

---

**Figure 6.6** The conceptual structure of the aggregate type Employee

- **Meredith W Linsmeyer**
  - Employee.Name

- Employee
  - 23
    - Employee.Age
  - 6.2
    - Employee.SkillRating
Procedural Units

• Local versus Global Variables
• Formal versus Actual Parameters
• Passing parameters by value versus reference
• Procedures versus Functions
Figure 6.8  The flow of control involving a procedure

Figure 6.9  The procedure ProjectPopulation written in the programming language C

```c
void ProjectPopulation (float GrowthRate) {
    int Year;
    Population[0] = 100.0;
    for (Year = 0; Year <= 10; Year++)
        Population[Year+1] = Population[Year] + (Population[Year] * GrowthRate);
}
```

Starting the head with the term void is the way that C programmers specify that the program unit is a procedure rather than a function. We will learn about functions shortly.

The formal parameter list. Note that C, as with many programming languages, requires that the data type of each parameter be specified.

This declares a local variable named Year.

Those statements describe how the populations are to be computed and stored in the global array named Population.
Figure 6.10 Executing the procedure Demo and passing parameters by value

a. When the procedure is called, a copy of the data is given to the procedure.

```
Calling environment
5

Procedure's environment
\rightarrow 5
```

b. and the procedure manipulates its copy.

```
Calling environment
5

Procedure's environment
6
```

c. Thus, when the procedure has terminated, the calling environment has not been changed.

```
Calling environment
5
```

Figure 6.11 Executing the procedure Demo and passing parameters by reference

a. When the procedure is called, the formal parameter becomes a reference to the actual parameter.

```
Calling environment
Actual
5

Procedure's environment
Formal
```

b. Thus, changes directed by the procedure are made to the actual parameter.

```
Calling environment
Actual
6

Procedure's environment
Formal
```

c. and are, therefore, preserved after the procedure has terminated.

```
Calling environment
Actual
6
```
Figure 6.12  The function CylinderVolume written in the programming language C

The function header begins with the type of the data that will be returned.

```c
float CylinderVolume (float Radius, float Height) {
    float Volume;
    Volume = 3.14 * Radius * Radius * Height;
    return Volume;
}
```

- Declare a local variable named Volume.
- Compute the volume of the cylinder.
- Terminate the function and return the value of the variable Volume.

Figure 6.13  The translation process

Source program → Lexical analyzer → Parser → Code generator → Object program

Copyright © 2012 Pearson Education, Inc.
Figure 6.14 A syntax diagram of our if-then-else pseudocode statement

Figure 6.15 Syntax diagrams describing the structure of a simple algebraic expression
Figure 6.16  The parse tree for the string $x + y \times z$ based on the syntax diagrams in Figure 6.17

Figure 6.17  Two distinct parse trees for the statement if $B_1$ then if $B_2$ then $S_1$ else $S_2$
Objects and Classes

• **Object**: Active program unit containing both data and procedures
• **Class**: A template from which objects are constructed

An object is called an **instance** of the class.
Components of an Object

- **Instance Variable**: Variable within an object
  - Holds information within the object
- **Method**: Procedure within an object
  - Describes the actions that the object can perform
- **Constructor**: Special method used to initialize a new object when it is first constructed
Figure 6.21  A class with a constructor

```java
class LaserClass {
    int RemainingPower;
    LaserClass (InitialPower) {
        RemainingPower = InitialPower;
    }
    void turnRight () {
        ... }
    void turnLeft  () {
        ... }
    void fire      () {
        ... }
}
```

Object Integrity

• **Encapsulation**: A way of restricting access to the internal components of an object
  – Private
  – Public
Additional Object-oriented Concepts

- **Inheritance**: Allows new classes to be defined in terms of previously defined classes
- **Polymorphism**: Allows method calls to be interpreted by the object that receives the call
Programming Concurrent Activities

- **Parallel (or concurrent) processing:** simultaneous execution of multiple processes
  - True concurrent processing requires multiple CPUs
  - Can be simulated using time-sharing with a single CPU

---

**Figure 6.23  Spawning threads**

[Diagram showing spawning threads]

---
Controlling Access to Data

- **Mutual Exclusion**: A method for ensuring that data can be accessed by only one process at a time
- **Monitor**: A data item augmented with the ability to control access to itself

Declarative Programming

- **Resolution**: Combining two or more statements to produce a new statement (that is a logical consequence of the originals).
  - Example: \((P \text{ OR } Q) \text{ AND } (R \text{ OR } \neg Q)\) resolves to \((P \text{ OR } R)\)
  - **Resolvent**: A new statement deduced by resolution
  - **Clause form**: A statement whose elementary components are connected by the Boolean operation OR
- **Unification**: Assigning a value to a variable so that two statements become “compatible.”
Figure 6.24  Resolving the statements \((P \lor Q)\) and \((R \lor \lnot Q)\) to produce \((P \lor R)\)

\[ \begin{align*}
P \lor Q & \quad R \lor \lnot Q \\
P \lor R & \\ 
\end{align*} \]

Figure 6.25  Resolving the statements \((P \lor Q), (R \lor \lnot Q), \lnot R,\) and \(\lnot P\)

\[ \begin{align*}
P \lor Q & \quad R \lor \lnot Q \\
\lnot R & \quad \lnot P \\
P \lor R & \\
P & \\
\text{empty clause} & \\
\end{align*} \]
Prolog

• **Fact:** A Prolog statement establishing a fact
  – Consists of a single predicate
  – Form: `predicateName(arguments)`.
    • Example: `parent(bill, mary)`.

• **Rule:** A Prolog statement establishing a general rule
  – Form: `conclusion :- premise`
    • `:-` means “if”
  – Example: `wise(X) :- old(X)`.
  – Example: `faster(X,Z) :- faster(X,Y), faster(Y,Z)`.