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**

- Problems solved in an environment in which the human must conform to the machine’s characteristics
- Problems solved in an environment in which the machine conforms to the human’s characteristics

**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, ShippingCharge</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>

---

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

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

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

**Variable Declarations**
```plaintext
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 heterogeneous array Employee

Employee
  23 Employee.Age
Meredith W Linsmeyer
  Employee.Name
  Employee.SkillRating

Figure 6.7 The for loop structure and its representation in C++, C#, and Java

for (int Count = 1; Count<4; Count++)
body ;

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

- Control is transferred to procedure.
- Procedure is executed.
- Control is returned to calling environment when procedure is completed.

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 a C programmer specifies 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.
- These 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.
- b. and the procedure manipulates its copy.
- c. Thus, when the procedure has terminated, the calling environment has not been changed.

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.
- b. Thus, changes directed by the procedure are made to the actual parameter.
- c. and are, therefore, preserved after the procedure has terminated.
Figure 6.12 The function CylinderVolume written in the programming language C

```c
float CylinderVolume (float Radius, float Height)
{
    float Volume;
    Declare a local variable named Volume.

    Volume = 3.14 * Radius * Radius * Height;
    Compute the volume of the cylinder.

    return Volume;
    Terminate the function and return the value of the variable Volume.
}
```

Figure 6.13 The translation process

Figure 6.14 A syntax diagram of our if-then-else pseudocode statement

```
if Boolean expression then Statement else 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 $B1$ then if $B2$ then $S1$ else $S2$

**Figure 6.18** An object-oriented approach to the translation process

**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.
**Figure 6.19** The structure of a class describing a laser weapon in a computer game

```java
class LaserClass
{
    int RemainingPower = 100;
    void turnRight()
    {
    }
    void turnLeft()
    {
    }
    void fire()
    {
    }
}
```

- **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 6.20** The Structure of a typical object-oriented program

```
Program

main ...
{
...
}

class ...
{
...
}

class ...
{
...
}

class ...
{
...
}
```

- Procedural unit (often called main) that directs the construction of the objects and makes appropriate calls to their methods.
- Class descriptions

**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 versus public

![Image](31x50 to 285x389)

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

![Image](310x50 to 564x389)

Figure 6.22 Our LaserClass definition using encapsulation

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

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
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

Resolution: Combining two or more statements to produce a new statement (that is a logical consequence of the originals).
- Example: \((P \lor Q) \land (R \lor \neg Q)\) resolves to \((P \lor R)\)
- Resolvent: A new statement deduced by resolution
- Clause form: A statement whose elementary components are connected by the Boolean operation \(\lor\) or \(\land\)

Unification: Assigning a value to a variable so that two statements become “compatible.”
**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).`