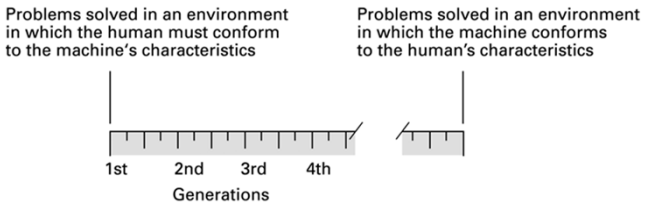
Chapter 6:
Programming Languages

Computer Science: An Overview
Eleventh Edition

by
J. Glenn Brookshear
Dennis Brylow

Chapter 6: Programming Languages

Figure 6.1 Generations of programming languages



Second-generation: Assembly language

- A mnemonic system for representing machine instructions
  - Mnemonic names for op-codes
  - Program **variables** or **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

Machine language	Assembly language
156C	LD R5, Price
166D	LD R6, ShipCharge
5056	ADDI R0, R5 R6
30CE	ST R0, TotalCost
C000	HLT

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

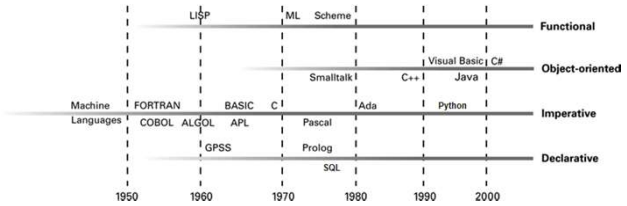


Figure 6.3 A function for checkbook balancing constructed from simpler functions

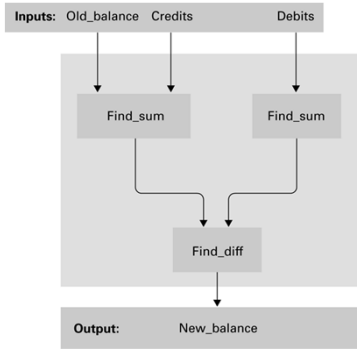
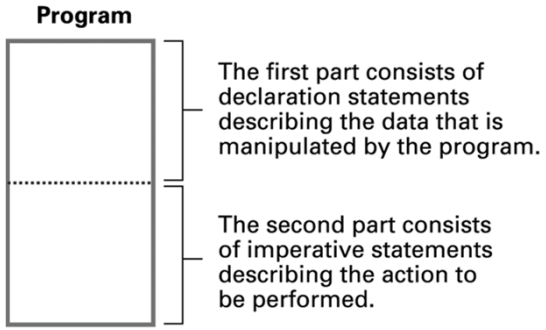


Figure 6.4 The composition of a typical imperative program or program unit



Data Types

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

Variables and Data types

```
float Length, Width;
int   Price, Total, Tax;
char  Symbol;

int   WeightLimit = 100;
```

Data Structure

- Conceptual shape or arrangement of data
- A common data structure is the **array**
  - In C

```
int Scores[2][9];
```

  - In FORTRAN

```
INTEGER Scores(2,9)
```


Figure 6.5 A two-dimensional array with two rows and nine columns

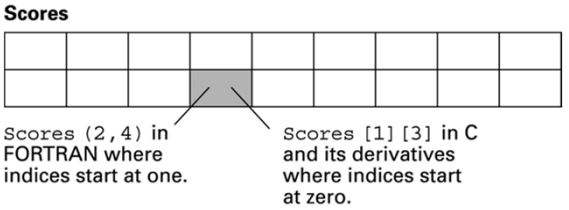
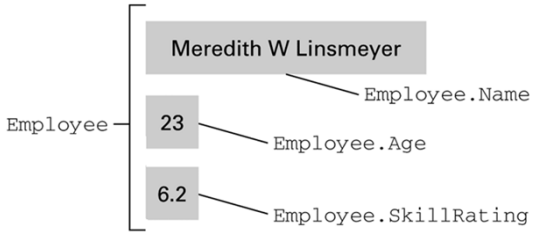


Figure 6.6 The conceptual structure of the aggregate type Employee




```
struct {   char  Name[25];
           int   Age;
           float SkillRating;
      } Employee;
```


Assignment Statements

- In C, C++, C#, Java

```
Z = X + y;
```

- In Ada

```
Z := X + y;
```

- In APL (A Programming Language)

```
Z ← X + y
```

Control Statements

- Go to statement

```
      goto 40
20    Evade()
      goto 70
40    if (KryptoniteLevel < LethalDose) then goto 60
      goto 20
60    RescueDamsel()
70    ...
```

- As a single statement

```
if (KryptoniteLevel < LethalDose):
    RescueDamsel()
else:
    Evade()
```


Control Statements (continued)

- If in Python

```
if (condition):
    statementA
else:
    statementB
```

- In C, C++, C#, and Java

```
if (condition) statementA; else statementB;
```

- In Ada

```
IF condition THEN
    statementA;
ELSE
    statementB;
END IF;
```


Control Statements (continued)

- While in Python

```
while (condition):
    body
```

- In C, C++, C#, and Java

```
while (condition)
{ body }
```

- In Ada

```
WHILE condition LOOP
    body
END LOOP;
```


Control Statements (continued)

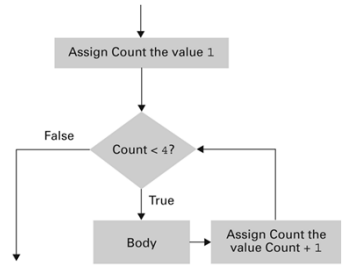
- Switch statement in C, C++, C#, and Java

```
switch (variable) {
    case 'A': statementA; break;
    case 'B': statementB; break;
    case 'C': statementC; break;
    default:  statementD; }
```

- In Ada

```
CASE variable IS
    WHEN 'A'=> statementA;
    WHEN 'B'=> statementB;
    WHEN 'C'=> statementC;
    WHEN OTHERS=> statementD;
END CASE;
```

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




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


Comments

- Explanatory statements within a program
- Helpful when a human reads a program
- Ignored by the compiler

```
/* This is a comment. */

// This is a comment
```


Procedural Units

- Many terms for this concept:
  - Subprogram, subroutine, procedure, method, function
- Unit begins with the function's **header**
- Local versus Global Variables
- Formal versus Actual Parameters
- Passing parameters by value versus reference

Figure 6.8 The flow of control involving a function

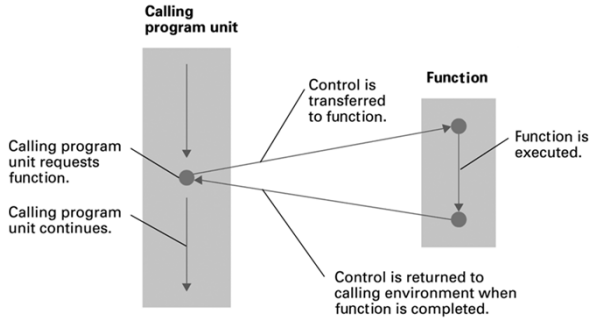


Figure 6.9 The function ProjectPopulation written in the programming language C

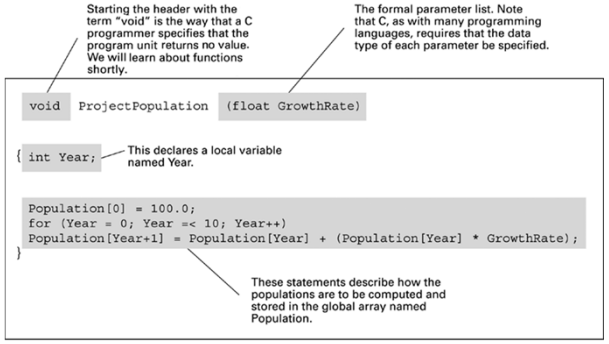


Figure 6.10 Executing the function Demo and passing parameters by value

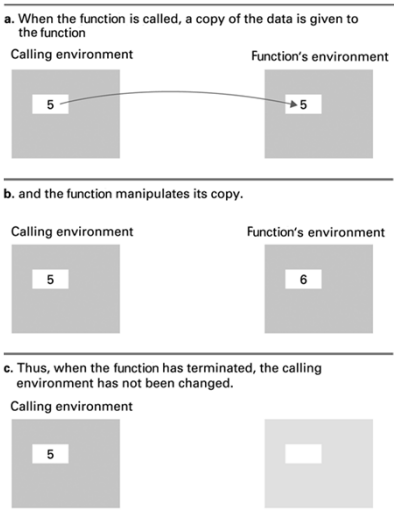


Figure 6.11 Executing the function Demo and passing parameters by reference

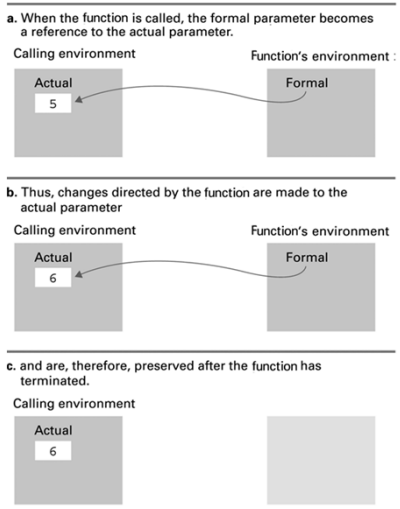


Figure 6.12 The fruitful function CylinderVolume written in the programming language C

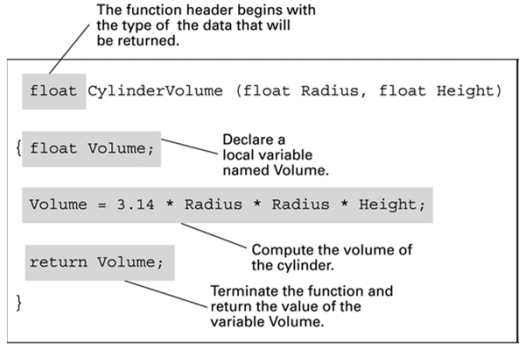


Figure 6.13 The translation process

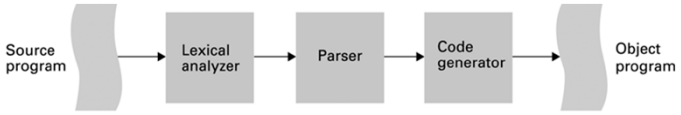


Figure 6.14 A syntax diagram of Python's if-then-else statement

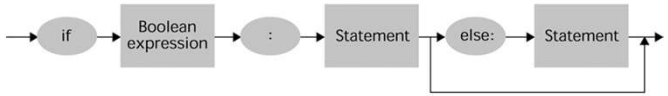


Figure 6.15 Syntax diagrams describing the structure of a simple algebraic expression

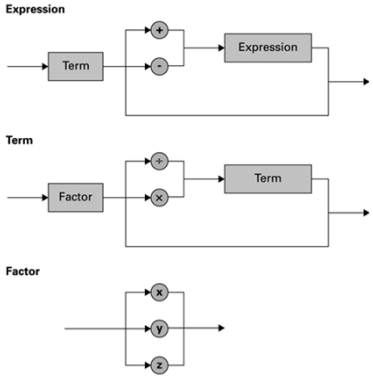


Figure 6.16 The parse tree for the string x + y * z based on the syntax diagrams in Figure 6.17

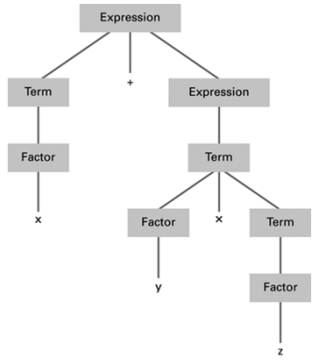


Figure 6.17
Two distinct parse trees for the statement if B1 then if B2 then S1 else S2

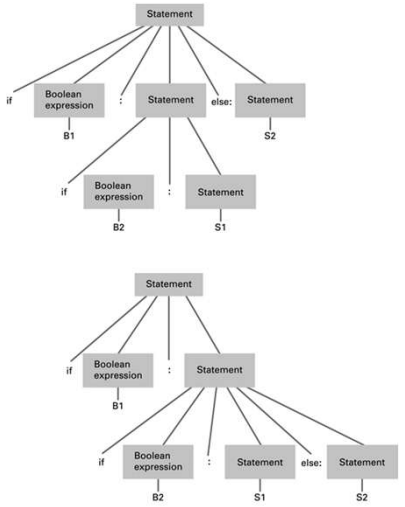
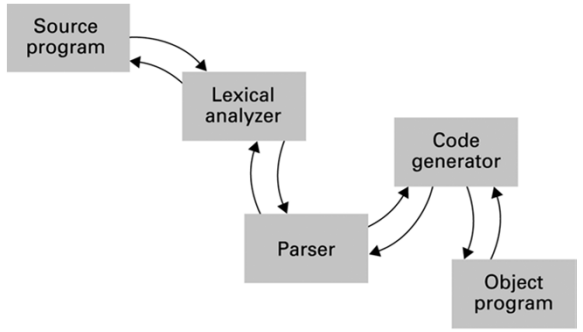


Figure 6.18 **An object-oriented approach to the translation process**



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**

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

Description of the data that will reside inside of each object of this "type."

Methods describing how an object of this "type" should respond to various messages

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

```
class  LaserClass
{ int RemainingPower;

  LaserClass (InitialPower)
  { RemainingPower = InitialPower;
  }

  void turnRight ( )
  { ... }

  void turnLeft ( )
  { ... }

  void fire ( )
  { ... }
}
```

Constructor assigns a value to RemainingPower when an object is created.

Object Integrity

- **Encapsulation:** A way of restricting access to the internal components of an object
  - Private
  - Public

Figure 6.22 Our LaserClass definition using encapsulation as it would appear in a Java or C# program

Components in the class are designated public or private depending on whether they should be accessible from other program units.

```
class  LaserClass
{private int RemainingPower;

 public LaserClass (InitialPower)
 {RemainingPower = InitialPower;
 }

 public void turnRight ( )
 { ... }

 public void turnLeft ( )
 { ... }

 public void fire ( )
 { ... }
}
```

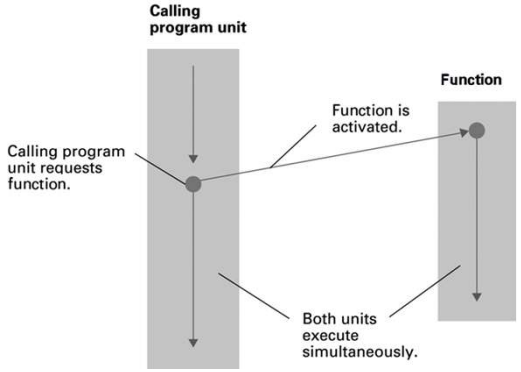
Additional Object-oriented Concepts

- **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



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 OR Q) AND (R OR ¬Q) resolves to (P 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* OR *Q*) and (*R* OR ¬*Q*) to produce (*P* OR *R*)

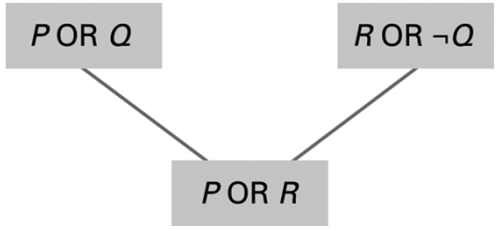
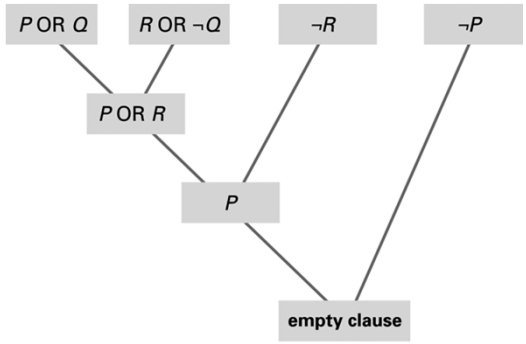


Figure 6.25 Resolving the statements (*P* OR *Q*), (*R* OR ¬*Q*), ¬R, and ¬*P*



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).`