Chapter 9
Introduction to High-Level Language Programming
Objectives

After studying this chapter, students will be able to:

• Explain the advantages of high-level programming languages over assembly language
• Describe the general process of translation from high-level source code to object code
• Compare and contrast the ways languages express basic operations (Ada, C++, C#, Java, and Python)
• Explain the Favorite Number and Data Cleanup examples for each programming language
Objectives (continued)

After studying this chapter, students will be able to:

• Explain why the software development life cycle is necessary for creating large software programs
• List the steps in the software development life cycle, explain the purpose of each, and describe the products of each
• Explain how agile software development differs from the traditional waterfall model
The Language Progression

- Assembly language improves on machine language
- Assembly language disadvantages:
  - Programmer must manage movement of data among memory locations and registers
  - Microscopic view of task
  - Machine specific languages
  - Far from natural language
The Language Progression (continued)

• High-level programming languages improve on assembly language

• Expectations:
  – Programmer need not manage data in memory
  – Macroscopic view of tasks (e.g., “add B and C”)
  – Programs portable from one machine to another
  – Program statements closer to natural language and math notation

• Third-generation languages
The Language Progression (continued)

• Need translator from high level languages
  – Compiler converts **source code** to assembly code or similar
  – Assembler or other converter makes **object code**

• **Code libraries** contain object code for useful tools

• **Linker** integrates multiple files of object code to create an **executable module**
FIGURE 9.1

Transitions of a high-level language program

- High-level language program
- Compiler
- Assembly language or other low-level code
- Assembler
- Object code in machine language
- Source program
- Translation
- Intermediate program
- Translation
- Object program
- Linker
- Complete object code
- Loader
- Complete object code loaded into memory
- Hardware
- Results
- Loading
- Execution
- Library object code
A Family of Languages

• **Procedural (imperative) languages**
  – Popular kind of programming language
  – Programs are sequences of statements
• Examples: Ada, C++, C#, Java, and Python
• Same underlying philosophy/model
• Variations in:
  – **Syntax**: how statements are written
  – **Semantics**: meaning of statements
Two Examples in Five-Part Harmony

- Examine similarities and differences of the languages (Ada, C++, C#, Java, Python) through examples
- Favorite Number
  - Ask user for her favorite number, tell her that your favorite number is one greater than hers
- Data Cleanup
  - Converging Pointers algorithm from Chapter 3
Two Examples in Five-Part Harmony (continued)

- Focus on big picture, not grasping every detail
- Look for commonalities across languages
- Look for these identifiable patterns:
  - Creation/declaration of variables
  - Assignment of variables to values
  - Arithmetic operations
  - Reading input and writing output
  - Markers for beginning and end of sections
  - Conditionals (if statements)
  - Loops (for or while loops)
Two Examples in Five-Part Harmony (continued)

First example: Favorite Number

• Simple program, no loops or conditionals
• Focus on input/output, variable creation and assignment

FIGURE 9.2

1. Get value for the user’s favorite number, \( n \)
2. Increase \( n \) by 1
3. Print a message and the new value of \( n \)

Pseudocode algorithm for favorite number
--Ada program for the
--favorite number algorithm

WITH TEXT_IO;

PROCEDURE FavoriteNumber IS
    PACKAGE INT_IO IS NEW TEXT_IO.INTEGER_IO(INTEGER);

    n : INTEGER;  -- user's favorite number

BEGIN
    -- Get the user's favorite number
    TEXT_IO.PUT("What is your favorite number? ");
    INT_IO.GET(n);

    -- Compute the next number
    n := n + 1;

    -- Write the output
    TEXT_IO.NEW_LINE;
    TEXT_IO.PUT("My favorite number is 1 more than that, ");
    INT_IO.PUT(n, 4);
    TEXT_IO.NEW_LINE;
    TEXT_IO.NEW_LINE;
END FavoriteNumber;
//C++ program for the
//favorite number algorithm

#include <iostream>
using namespace std;

void main()
{
    int n;    //user’s favorite number

    //get the user’s favorite number
    cout << "What is your favorite number? ";
    cin >> n;

    //compute the next number
    n = n + 1;

    //write the output
    cout << endl;
    cout << "My favorite number is 1 more than that, ",
         << n << endl;
    cout << endl << endl;
}

C++ program for favorite number
```csharp
// C# program for the
// favorite number algorithm

using System;

namespace InvitationCSharp
{
    class FavoriteNumber
    {
        static void Main(string[] args)
        {
            int n; // user's favorite number

            // get the user's favorite number
            Console.Write("What is your favorite number? ");
            n = Convert.ToInt32(Console.ReadLine());

            // compute the next number
            n = n + 1;

            // write the output
            Console.WriteLine();
            Console.Write("My favorite number is ");
            Console.WriteLine("1 more than that, "+ n);
            Console.WriteLine();
            Console.WriteLine();
        }
    }
}
```

C# program for favorite number
//Java program for the
//favorite number algorithm

import java.util.*;
public class FavoriteNumber
{
    public static void main(String[] args)
    {
        int n;         //user’s favorite number
        Scanner inp = new Scanner(System.in);   //to read input

        //get the user’s favorite number
        System.out.print("What is your favorite number? ");
        n = inp.nextInt();

        //compute the next number
        n = n + 1;

        //write the output
        System.out.println();
        System.out.println("My favorite number is 1 more "+"than that, "+n);
        System.out.println();
        System.out.println();
    }
}
# Python program for the favorite number algorithm

# get the user's favorite number
n = int(input("What is your favorite number? "))

# compute the next number
n = n + 1

# write the output
print()
print("My favorite number is 1 more than that,", n)

# finish up
input("\n\nPress the Enter key to exit");
Two Examples in Five-Part Harmony (continued)

Second example: Data Cleanup, Converging Pointers
• Focus on loops and conditionals

FIGURE 9.8
1. Get values for $n$ and the $n$ data items
2. Set the value of legit to $n$
3. Set the value of left to 1
4. Set the value of right to $n$
5. While left is less than right do Steps 6 through 10
6. If the item at position left is not 0 then increase left by 1
7. Else (the item at position left is 0) do Steps 8 through 10
8. Reduce legit by 1
9. Copy the item at position right into position left
10. Reduce right by 1
11. If the item at position left is 0, then reduce legit by 1
12. Stop

The converging-pointers algorithm for data cleanup
Figure 9.9
Ada converging-pointers algorithm (part 1)

```ada
--Ada program for the converging-pointers
--data cleanup algorithm

WITH TEXT_IO;

PROCEDURE DataCleanup IS
  PACKAGE INT_IO IS NEW TEXT_IO.INTEGER_IO(INTEGER);

  maxList : constant := 50; --maximum list size
  n : INTEGER;            --max number of data elements
  --in list
  data : array(0..maxList - 1) of INTEGER; --create the empty
                                           --list
  i : INTEGER;            --index variable
  left : INTEGER;         --algorithm left pointer into
                          --the list
  right: INTEGER;         --algorithm right pointer
                          --into the list
  legit: INTEGER;         --counts number of legitimate
                          --(nonzero) data values
```
Figure 9.9 (continued)
Ada converging-pointers algorithm (part 2)

BEGIN
   --Get the values for n and the n data items
   TEXT_IO.PUT("How many numbers are in the list? (maximum is ");
   INT_IO.PUT(maxList,2);
   TEXT_IO.PUT(" ");
   INT_IO.GET(n);
   
   i := 0;
   TEXT_IO.PUT("Enter the first number: ");
   INT_IO.GET(data(i));
   
   while i < n - 1
      loop
         i := i + 1;
         TEXT_IO.PUT("Enter next number: ");
         INT_IO.GET(data(i));
      end loop;
   
   --Set the value of legit, left, and right
   legit := n - 1;
   left := 0;
   right := n - 1;

   TEXT_IO.NEW_LINE;
   TEXT_IO.PUT("The original list is: ");
Figure 9.9 (continued)
Ada converging-pointers algorithm (part 3)

i := 0;
while i <= legit
  loop
    INT_IO.PUT(data(i), 4);
    i := i + 1;
  end loop;
TEXT_IO.NEW_LINE;
TEXT_IO.NEW_LINE;

-- move the pointers together,
-- swapping value at right for 0 at left

while left < right
  loop
    if data(left) /= 0
      then
        left := left + 1;
      else
        legit := legit - 1;
        data(left) := data(right);
        right := right - 1;
    end if;
  end loop;
if data(left) = 0
   then
      legit := legit - 1;
   end if;

--final output
TEXT_IO.PUT("The cleaned list is: ");

i := 0;
while i <= legit
   loop
      INT_IO.PUT(data(i), 4);
      i := i + 1;
   end loop;
TEXT_IO.NEW_LINE;

END DataCleanup;
Figure 9.10
C++ converging-pointers algorithm (part 1)

```cpp
#include <iostream>
using namespace std;

void main() {
    const int MAXLIST = 50; // maximum list size
    int n; // max number of data elements in list
    int data[MAXLIST]; // create the empty list
    int i; // index variable
    int left, right; // algorithm pointers into the list
    int legit; // counts number of legitimate (nonZero) data values

    // Get the values for n and the n data items
    cout << "How many numbers are in the list? ";
    cout << "(maximum is " << MAXLIST << ") ";
    cin >> n;

    i = 0;
    cout << "Enter the first number: ";
    cin >> data[i];
```
while (i < n - 1)
{
    i = i + 1;
    cout << "Enter next number: ";
    cin >> data[i];
}

// Set the value of legit, left, and right
legit = n - 1;
left = 0;
right = n - 1;

cout << endl;
cout << "The original list is" << endl;
i = 0;
while (i <= legit)
{
    cout << data[i] << " ";
    i = i + 1;
}
cout << endl << endl;

// Move the pointers together,
// swapping value at right for 0 at left
while (left < right)
{
    if (data[left] != 0)
        left = left + 1;
    else
    {
        legit = legit - 1;
        data[left] = data[right];
        right = right - 1;
    }
}
if (data[left] == 0)
    legit = legit - 1;

// final output
cout << "The cleaned list is" << endl;
i = 0;
while (i <= legit)
{
    cout << data[i] << " ";
    i = i + 1;
}
cout << endl << endl;

Figure 9.11

C# converging-pointers algorithm (part 1)

```
//C# program for the converging-pointers
//data cleanup algorithm

using System;

namespace InvitationCSharp
{
    class DataCleanup
    {
        static void Main(string[] args)
        {
            const int maxList = 50; //maximum list size
            int n; //max number of data
            int[] data = new int[maxList]; //elements in list
            //create the empty
            //list
            int i; //index variable
```
**Figure 9.11 (continued)**

C# converging-pointers algorithm (part 2)

```csharp
int left, right; //algorithm pointers
int legit; //into the list
//counts number of
//legitimate
//(nonzero) data
//values

//Get the values for n and the n data items
Console.WriteLine("How many numbers are in the list? ");
Console.WriteLine("(maximum is " + maxList + ") ");
n = Convert.ToInt32(Console.ReadLine());

i = 0;
Console.WriteLine("Enter the first number: ");
data[i] = Convert.ToInt32(Console.ReadLine());

while (i < n - 1)
{
    i = i + 1;
    Console.WriteLine("Enter next number: ");
data[i] = Convert.ToInt32(Console.ReadLine());
}
```
Figure 9.11 (continued)
C# converging-pointers algorithm (part 3)

```csharp
// Set the value of legit, left, and right
legit = n - 1;
left = 0;
right = n - 1;

Console.WriteLine();
Console.WriteLine("The original list is");

i = 0;
while (i <= legit)
{
    Console.Write(data[i] + " ");
    i = i + 1;
}
Console.WriteLine();
Console.WriteLine();

// move the pointers together,
// swapping value at right for 0 at left

while (left < right)
{
    if (data[left] != 0)
    {
        left = left + 1;
    }
} else
```
Figure 9.11 (continued)
C# converging-pointers algorithm (part 4)

```csharp
{ 
    legit = legit - 1;
    data[left] = data[right];
    right = right - 1;
}

if (data[left] == 0)
    legit = legit - 1;

//final output
Console.WriteLine("The cleaned list is");
i = 0;
while (i <= legit)
{
    Console.Write(data[i] + " ");
    i = i + 1;
}
Console.WriteLine();
Console.WriteLine();
```
Figure 9.12
Java converging-pointers algorithm (part 1)

```java
import java.util.*;
public class DataCleanup {
    public static void main(String[] args) {
        final int MAXLIST = 50; // maximum list size
        int n;
        int[] data = new int[MAXLIST]; // create the empty list
        int i;
        int left, right;
        int legit;
        // max number of data elements
        // in the list
        // create the empty list
        // index variable
        // algorithm pointers
        // into the list
        // counts number of legitimate (nonzero)
        // data values
    }

    int n;
}
```
Figure 9.12 (continued)
Java converging-pointers algorithm (part 2)

```java
Scanner inp = new Scanner(System.in); //to read input

//Get the values for n and the n data items
System.out.print("How many numbers are in the list? ");
System.out.print(" (maximum is "+ MAXLIST "+") ");
n = inp.nextInt();

i = 0;
System.out.print("Enter the first number: ");
data[i] = inp.nextInt();

while (i < n - 1)
{
    i = i + 1;
    System.out.print("Enter next number: ");
    data[i] = inp.nextInt();
}

//Set the value of legit, left, and right
legit = n - 1;
left = 0;
right = n - 1;
```
Figure 9.12 (continued)
Java converging-pointers algorithm (part 3)

System.out.println();
System.out.println("The original list is");

i = 0;
while (i <= legit)
{
    System.out.print(data[i] + " ");
    i = i + 1;
}
System.out.println();
System.out.println();

// move the pointers together,
// swapping value at right for 0 at left

while (left < right)
{
    if (data[left] != 0)
    {
        left = left + 1;
    }
    else
    {
        legit = legit - 1;
        data[left] = data[right];
        right = right - 1;
    }
}
if (data[left] == 0)
    legit = legit - 1;

// final output
System.out.println("The cleaned list is");
    i = 0;
while (i <= legit)
{
    System.out.print(data[i] + " ");
    i = i + 1;
}

System.out.println();
```python
# Python program for the converging-pointers algorithm
# data cleanup algorithm

# Get the values for n and the n data items
n = int(input("How many numbers are in the list: "))
data = []  # create an empty list

i = 0
number = int(input("Enter first number: "))
data.append(number)  # append a value to the data list
while i < n - 1:
    i = i + 1
    number = int(input("Enter next number: "))
data.append(number)

# Set the value of legit, left, and right
legit = n - 1
left = 0
right = n - 1

print()
print("The original list is")
i = 0
while i <= legit:
    print(data[i], end=" ")
i = i + 1
print()
```

Figure 9.13
Python converging-pointers algorithm (part 1)
Figure 9.13 (continued)
Python converging-pointers algorithm (part 2)

```python
print()

# move the pointers together,
# swapping value at right for 0 at left

while left < right:
    if data[left] != 0:
        left = left + 1
    else:
        legit = legit - 1
        data[left] = data[right]
        right = right - 1

if data[left] == 0:
    legit = legit - 1

# final output
print("The cleaned list is")
i = 0
while i <= legit:
    print(data[i], end=" ")
i = i + 1

# finish up
input("\n\nPress the Enter key to exit");
```
FIGURE 9.14

How many numbers are in the list? (maximum is 50) 10
Enter the first number: 0
Enter next number: 24
Enter next number: 16
Enter next number: 0
Enter next number: 36
Enter next number: 42
Enter next number: 23
Enter next number: 21
Enter next number: 0
Enter next number: 27

The original list is
0 24 16 0 36 42 23 21 0 27

The cleaned list is
27 24 16 21 36 42 23

Output from the various data cleanup implementations
Feature Analysis

- Compare language features, and compare to pseudocode
- Syntax
  - Describing data or variables
  - Grouping things, making loops or conditionals
- Semantics
  - Meaning of function call
  - Meaning of operations
- Deeper structure
  - Modules, classes, scope
Meeting Expectations

• Programmer need not manage data in memory
  – In each language, programmers must declare names (and sometimes types) for variables
  – Program manages movement of data associated with a given variable name
• Macroscopic view of tasks (e.g., “add B and C”)
  – Languages provide statements for high-level math
  – Details of conditionals and loops hidden
Meeting Expectations (continued)

• Programs portable from one machine to another
  – Programming languages are standardized
  – Compiled languages (Ada, C++, Java, C#)
    • Compilers written for particular platform support standard, and translate to machine-specific forms
    • Programmers distribute executable or low level “bytecode”, not source
  – Interpreted languages (like Python) require an interpreter on each machine, and distribute source code
Meeting Expectations (continued)

- Program statements closer to natural language and math notation
  - Math notation fairly standard across languages
  - Conditionals and loops are closer to natural language than assembly
The Big Picture: Software Engineering

- **Software development life cycle**
  - Process required to create large-scale software projects
  - 25% to 40% of time spent on problem specification and program design
  - 10% to 20% of time spent on initial implementation
  - 40% to 65% of time spent reviewing, modifying, fixing, and improving software
FIGURE 9.16

1. Before Implementation
   a. Feasibility study
   b. Problem specification
   c. Program design
   d. Algorithm selection or development, and analysis
2. Implementation
   a. Coding
   b. Debugging
3. After Implementation
   a. Testing, verification, and benchmarking
   b. Documentation
   c. Maintenance

Steps in the software development life cycle
The Big Picture: Software Engineering (continued)

• Large software scale issues
  – Orders of magnitude larger than beginner programs
  – Compare one sentence to 300-page novel

• Software engineering
  – Development of large software projects
  – Requires collaboration, management, organization
  – Formal processes for development
<table>
<thead>
<tr>
<th>Category</th>
<th>Typical Number of People</th>
<th>Typical Duration</th>
<th>Product Size in Lines of Code</th>
<th>Examples</th>
<th>Building Analogy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trivial</td>
<td>1</td>
<td>1–2 weeks</td>
<td>&lt; 500</td>
<td>Student homework assignments</td>
<td>Small home improvement</td>
</tr>
<tr>
<td>Small</td>
<td>1–3</td>
<td>A few weeks or months</td>
<td>500–2,000</td>
<td>Student team projects, advanced course assignments</td>
<td>Adding on a room</td>
</tr>
<tr>
<td>Medium</td>
<td>2–5</td>
<td>A few months to 1 year</td>
<td>2,000–10,000</td>
<td>Research projects, simple production software such as assemblers, editors, recreational and educational software</td>
<td>Single-family house</td>
</tr>
<tr>
<td>Large</td>
<td>5–25</td>
<td>1–3 years</td>
<td>10,000–100,000</td>
<td>Most current applications—word processors, spreadsheets, operating systems for small computers, compilers</td>
<td>Small shopping mall</td>
</tr>
<tr>
<td>Very Large</td>
<td>25–100</td>
<td>3–5 years</td>
<td>100,000–1 M</td>
<td>Airline reservations systems, inventory control systems for multinational companies</td>
<td>Large office building</td>
</tr>
<tr>
<td>Extremely Large</td>
<td>&gt; 100</td>
<td>&gt; 5 years</td>
<td>&gt; 1 M</td>
<td>Large-scale real-time operating systems, advanced military work, international telecommunications networks</td>
<td>Massive skyscraper</td>
</tr>
</tbody>
</table>

Size categories of software products
The Big Picture: Software Engineering (continued)

Life cycle:

• **Feasibility study**
  – Assess costs and benefits
  – Consider alternatives

• **Problem specification**
  – Clear statement of problem to be solved
  – Problem specification document
The Big Picture: Software Engineering (continued)

Life cycle:

• **Program design phase**
  – *Divide-and-conquer, top-down decomposition*
  – Break problem into tasks and subtasks
  – Program design document

• **Algorithm selection/development and analysis**
  – Choose or design an algorithm for each subtask
  – Analyze efficiency
  – Document: describe algorithm in pseudocode, provide efficiency analysis, and rationale
The Big Picture: Software Engineering (continued)

Life cycle:

• **Coding**
  – Translate pseudocode and design into working code
  – Better design = easier coding

• **Debugging**
  – Correcting program errors:
    – **Syntax errors**: ungrammatical code statements
    – **Runtime errors**: illegal operations like divide-by-zero
    – **Logic errors**: errors in the algorithm itself
The Big Picture: Software Engineering (continued)

Life cycle:

• Testing, verification, and benchmarking
  – **Empirical testing**: develop suite of tests to check correctness
  – **Unit testing**: tests on each module/subtask
  – **Integration testing**: test how modules work together
  – **Regression testing**: when changes occur, test to be sure the change did not introduce errors

• **Program verification**: prove code correct

• **Benchmarking**: check performance on inputs
Life cycle:

- **Documentation**
  - **Internal documentation**: comments in code
  - **External documentation**: all earlier documents (problem spec, program design, etc.)
  - **Technical documentation**: information for programmers to understand the code
  - **User documentation**: help users run programs

- **Program maintenance**
  - Add features, fix bugs, improve performance
The Big Picture: Software Engineering (continued)

**Integrated development environment (IDE)**

- Program editor
- File manager
- Compiler/interpreter
- Debugger
- **Prototype** tools
- Version and document management
Agile software development

- Alternative to the waterfall model shown before
- Philosophy:
  - Problem specification is never done
  - Change is expected, respond in agile way
  - Customer/user involved throughout process

- **Pair programming:**
  - Two programmers, one computer
  - Code writer and observer roles, switched frequently
Summary

- High-level languages:
  - Hide details of memory and hardware operations
  - Enable code portability
  - Shift code statements toward natural language and math notation
- Programming languages may share an underlying philosophy, but vary in syntax and semantics details
- Ada, C++, C#, Java, and Python are all procedural languages
Summary (continued)

• Development of large software is a different kind of problem than writing a single algorithm
• Software development life cycles are designed to manage large software development
• Waterfall model starts with feasibility and problem specification and flows through debugging and testing
• Agile development moves quickly through the design-implement-test cycle, repeating it many times for a single project