**Chapter 10**

**The Tower of Babel: Programming Languages**

**A Guide to This Instructor’s Manual:**

We have designed this Instructor’s Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings, you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared toward quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor’s Manual, our Instructor’s Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

|  |
| --- |
| **At a Glance** |

**Instructor’s Manual Table of Contents**

* Overview
* Learning Objectives
* Teaching Tips and Quick Quizzes
* Class Discussion Topics
* Additional Projects
* Additional Resources
* Key Terms
* Solutions to End-of-Chapter Exercises

|  |
| --- |
| Lecture Notes |

**Overview**

Chapter 10 discusses why hundreds of different programming languages have been developed, emphasizing the special features of each language. Those special features are designed to simplify certain kinds of problem solving. A wide range of languages within the procedural paradigm are described, along with the problems they are particularly good at solving. The chapter introduces special-purpose languages, and other paradigms, including functional, logic, and parallel languages. The chapter provides a set of language features that may differ from one language to another, and that provide a framework for comparing languages.

# **Learning Objectives**

* Explain why so many programming languages exist
* List four key procedural languages and the main purpose for the development of each
* Describe the purpose of each special-purpose language: SQL, HTML, JavaScript, and R
* Describe the alternative paradigms for programming languages: functional, logic, and parallel
* Name a functional programming language and a logic programming language
* Describe how logic programming languages work, and explain what facts, rules, and inference are
* Explain how the MIMD model of parallel processing could be used to find the largest number in a list

# **Teaching Tips**

**10.1 Why Babel?**

1. Discuss the reasoning behind the existence of multiple programming languages. Each of the languages, like different models of vehicles, has tasks they perform well. Use C++ and engineering problem solving as an example.
2. Computer scientists and engineers choose a language based on (1) the tasks to be done and (2) the philosophical approach to computation that best fits the task and the person.

|  |  |
| --- | --- |
| ***Teaching Tip*** | To enliven this material, assign students to small groups and have each group research one of the chapters’ languages, presenting a summary to the class. |

**10.2 Procedural Languages**

1. Discuss the key features of procedural language: step-by-step instructions that describe manipulations of data in memory locations. Use the metaphor of the romance language family.
2. Plankalkül was an early language designed for the German computer in the World War II; discuss that it was the first high-level programing language even though it was never implemented.
3. FORTRAN was developed in the mid-1950s by Backus at IBM. It was designed for mathematical computations, particularly floating-point. It contained control structures close to assembly language: GO TO statements. Introduce the term **external libraries** and note that FORTRAN was the first to support them.
   1. Note that FORTRAN is still very active today and that a new release is scheduled for 2018.

|  |  |
| --- | --- |
| ***Teaching Tip*** | Side-by-side comparison of the same algorithm written in different languages can be very helpful here. Consider choosing one of the programs from Chapter 9 and providing a version in each language you intend to feature. |

1. COBOL was developed around 1959 by Hopper of the U.S. Navy.
   1. COBOL focused on business applications, such as inventory control and generating reports.
   2. Due to its natural language phrasing, it is a very verbose programming language.
   3. Introduce the term **legacy code** and talk about the Y2K issues that arose from code outliving its designers’ expectations.
2. C was developed in the early 1970s by Ritchie at AT&T Labs. It was designed to interact closely with the operating system, for system programming.
   1. Discuss the impact of its close connection with the UNIX operating system, and its mixture of high and low-level constructs.
   2. C allows a programmer to access pointers, memory addresses, directly.
   3. Introduce the term **device driver**, as an example of system software that benefits from low-level access as well as high-level features.
3. C++ was developed in the early 1980s by Stroustrup at AT&T Labs. It is explicitly a superset of C. C++ is an object-oriented language, very popular for software development.
4. Ada was developed in the mid-1970s by a team from all the U.S. armed services. Designed as a common language for defense contractors. It supports multiprocessing and enforces good software engineering practices. It is popular in the transportation industry because of its tools for ensuring safety and reliability.
5. SPARK is a subset of Ada that builds on Ada’s strengths but eliminates the ambiguities and security concerns. SPARK provides better support for critical system development.
6. Java was developed in the early 1990s by Sun Microsystems. It was focused particularly on web development. Introduce the terms **applications** and **applets**, and how describe Java implements each. Discuss its emphasis on platform independence and reliability. Introduce the terms **bytecode** and **Java bytecode interpreter**.
7. Python was developed in the early 1990s by van Rossum at Stichting Mathematisch Centrum. Introduce the term **open source** language, and describe why it is different from earlier languages in this chapter. Early applications were for system administrator tasks. The fact that it is easy to learn and use has made it popular.
8. C# was developed in 2000 by Microsoft. It was intended as a successor to C++ but is not backward compatible. It introduced features to be more secure than C++, including **garbage collection.**
9. Introduce the **Microsoft .NET Framework**, which C# and other languages can use: tools for developing software for Windows or the web. Introduce the **Microsoft Intermediate Language (MSIL)** and its use by the **just-in-time (JIT) compiler**.

**Quick Quiz 1**

1. COBOL is an early procedural language intended for \_\_\_\_\_\_\_ applications.

Answer: Business

1. (True or False) A procedural language is not required to support a GO TO statement.

Answer: True

1. (True or False) Python is an open-source programming language.

Answer: True

1. Which language was originally developed as a common language for defense contractors?

Answer: Ada

1. List two special features or purposes of Java.

Answer: Any two of the following: platform independence, emphasis on web applications and applets, more secure than C++, object oriented

**10.3 Special-Purpose Languages**

1. SQL is a query language for accessing information in databases. Provide examples of SQL queries and how web front ends are built to hide the details of SQL from users.
2. HTML is a language for describing the formatting and layout of web pages. Tags are written with angled brackets: <p> and </p>. Tags indicate parts of a document, font and font size, paragraphs, images, and tables. Show students how to view the HTML source from a web browser, and look together at some sample webpages.
   1. Be certain to view webpages that contain visible HTML as many pages hide the HTML behind stylesheets.
   2. While not in the text, discuss the Hyper Card history of HTML and show how Hyper Card stacks became the basis for modern websites.
3. JavaScript’s only relationship to Java is the fact that they are both object-oriented programming languages. JavaScript was developed by the people at Netscape and was originally known as LiveScript.
4. Introduce the term **event handlers**, when bits of JavaScript are invoked by user actions on a webpage.
5. R is a specialized programming language designed for statistical computing and graphics. The name comes from the first name of the authors of the language. R will compute different values based on the data set provided. Go over the examples in the chapter to help build understanding.

|  |  |
| --- | --- |
| ***Teaching Tip*** | Refer students to this tutorial about HTML: <http://www.w3schools.com/html/> |

**10.4 Alternative Programming Paradigms**

1. Introduce the term **paradigm**, and explain its use for philosophical approaches to programming languages. Describe the three alternative paradigms in this chapter: functional, logic, and parallel. Use compilers and interpreters for the languages discussed and demonstrate the different paradigms by running programs from each paradigm.
2. Introduce the term **functional programming language**. Demonstrate that a function is an algorithm that returns a value. Show the Scheme example in figure 10.10. Introduce the term **applicative languages** and explain the reasoning for the term. Discuss list processing in Scheme, and the peculiarly named list operators: car and cdr.Introduce the term **recursive** and work through an example of recursion in action. Introduce the term **side effect**.
3. Introduce the term **Logic programming languages**, sometimes called **declarative languages**. Emphasize that these languages use logical inference to respond to queries about facts in the program’s storehouse. Discuss Prolog and emphasize that it can be used for many purposes, but artificial intelligence is its main purpose. Go through the U.S. presidents example program with students and discuss how it works. Introduce the term **Prolog rule** and demonstrate that along with facts and queries in Prolog (free Prolog interpreters are easily available). Introduce the terms **knowledge base**, **inference engine**, and **modus ponens** to explain how Prolog works.
4. Review MIMD (multiple instruction stream/multiple data stream) model of parallel architecture. Describe large-scale problems that require parallel processing to compute. Introduce the term **multicore computing**, and **cluster computing** as alternate applications. For an example of cluster computing, talk about the SETI project and how it uses machines around the globe to search for Extra Terrestrial signals. Give code examples from the SIMD model of processing and compare to examples from the MIMD model. Introduce the term **divide-and-conquer model**, to describe a common approach to parallel algorithms, and apply to the phone book example.

**Quick Quiz 2**

1. Name the two functional programming languages discussed in this section.

Answer: Scheme and LISP

1. (True or False) Logic programming languages are typically used for large-scale scientific problems.

Answer: False

1. (True or False) The divide-and-conquer model produces inefficient parallel algorithms.

Answer: False

**10.5 New Languages Keep Coming**

1. Go, sometimes called *golang*, is a new language developed by Google to combine ease of use of languages like Python with efficiency and safety. It is an open source language, used for systems programming and multicore programming.
2. Swift was announced by Apple in 2014. Explain that it was designed for building apps on the iOS and OS X operating systems (e.g., iPhone, iPad, Mac). It is similar to Objective-C but faster and safer. Swift became open source in 2015 and by March 2017 it had entered the “top 10” popular languages list.
3. Milk was developed in 2016 by researchers at MIT’s Computer Science and Artificial Intelligence Laboratory. Its focus is on big data. Review the principal of locality, then discuss how Milk uses the cache memory of each processor core in a multicore processor to create a better list of memory addresses so that only the necessary data are retrieved when requested.

**10.6 Conclusion**

1. Remind students that there is a whole spectrum of compute programming out there that meet a myriad of problem solving needs.
2. Go over the various paradigms of programming and provide some examples from the text as a refresher.
3. Look at Figure 10.15 for a list of various programing languages and their paradigms.

# **Class Discussion Topics**

1. Examine the same algorithm implemented in several different programming languages (provided by instructor). Can you identify the language paradigm being used? Can you identify the language? List the similarities and differences between the programs. Can you align instructions or sets of instructions from different programs?
2. What are the pros and cons of open source versus proprietary programming languages? Consider the situation from the perspective of a language designer, a programmer using the language, and a business that needs to hire someone to write a program for it.
3. Look at some of the newer programming languages like Go and Swift and discuss the similarities between the two, and the similarities with some of the older programming languages. What makes them safer than older languages?

# **Additional Projects**

1. Working with a team, create a webpage by writing HTML that focuses on an extracurricular interest of one of the team members.
2. Create a small Prolog knowledge base that describes your family relationships: parents, children, uncles, aunts, and so forth. Try to create general rules that describe when someone is an uncle of someone else. Try out your facts and rules in a Prolog interpreter.
3. Pick a programming language from a list (provided by your instructor) and research the language online. What paradigm does it fit into? What is the language mainly intended to do? What are its features?
4. Research some of the high-level functions that were to be implemented in Plankalkül. How might this programing language have changed the direction of computer programing?

# **Additional Resources**

1. Listing of “hello world” programs in different languages: <http://en.wikibooks.org/wiki/Computer_Programming/Hello_world>
2. HTML tutorial: <http://www.htmltutorialforbeginners.com/>
3. A free Scheme system: <http://racket-lang.org/>
4. A free Prolog interpreter: <http://www.swi-prolog.org/>

**Key Terms**

* **Applet**: (Java) A small application, designed to run from webpages.
* **Application**: (Java) A complete stand-alone program that resides and runs on a self-contained computer.
* **Applicative language**: Another name for a functional programming language, so called because it repeatedly applies functions.
* **Bytecode**: (Java) Low-level code that can be easily translated into any specific machine language.
* **Cluster computing**: A collection of independent systems, such as mainframes, desktops, and laptops which are interconnected by a local area network (LAN) such as Ethernet or a wide area network (WAN) such as the Internet.
* **Declarative programming language**: Another name for a logic programming language, so called because its programs consist of declarations of fact rather than commands.
* **Device driver**: A program to interact with an I/O device.
* **Divide-and-conquer model**: A problem-solving approach that successively partitions the problem into smaller and smaller parts; used in MIMD processing to divide tasks among multiple processors.
* **Event handler**: Code that responds to a particular “event,” often a user action.
* **External library**: A well-written, efficient, and thoroughly tested code module that is separately compiled and then drawn on by any program that wishes to use its capabilities.
* **Functional programming languages**: Views every task in terms of functions. In this context, function means something like a mathematical function—a recipe for taking an argument (or possibly several arguments) and doing something with them to compute a single value.
* **Garbage collection**: Reclaiming memory no longer needed by a program.
* **Inference engine**: Software that is supplied as part of the compiler or interpreter for a logic programming language, which can access a knowledge base, and contains its own rules of deductive reasoning based on symbolic logic. Also called a query interpreter.
* **Java bytecode interpreter**: Software that translates bytecode into machine language and executes it.
* **Java Web Start (JAWS)**: Java code is accessed from the web via the user’s browser but executed in a restricted environment outside the browser itself.
* **Just-in-Time (JIT) compiler**: Part of the .NET framework that compiles MSIL code into object code on the user’s machine.
* **Knowledge base**: Facts and rules about a certain domain of interest.
* **Legacy code**: Old, but still-running, programs.
* **Logic programming language**: A language that, based on facts that are asserted to be true, can infer or deduce other facts.
* **Microsoft Intermediate Language (MSIL)**: Low-level code for a .NET language program that can be easily translated into any specific machine language.
* **Microsoft .NET Framework**: A collection of tools for software development designed so that traditional text-based applications, GUI applications, and web-based programs can all be built with equal ease.
* **Modus ponens**: A rule of deductive reasoning that states that “if A then B” together with “A” must result in “B.”
* **Multicore computing**: In which two or more processors are packaged together on a single integrated circuit.
* **Open source**: Software whose source code is freely available and may be used, distributed, or modified by anyone.
* **Paradigm**: A model or mental framework for representing or thinking about something.
* **Prolog fact**: A property expressed about a single object or a relationship among several objects in the Prolog language.
* **Prolog rule**: A declaration of an “if A then B” form, meaning that if A is true (A is a fact), then be is also true (B is a fact).
* **Recursive**: Something that is defined in terms of “smaller versions” of itself.
* **Side effect**: Occurs when a function, in the course of acting on its argument values to produce a result value, also changes other values that it has no business changing.

**Tags**: Special characters in HTML and other markup languages that achieve formatting, special effects, and links to other documents or webpages.

# **Solutions to End-of-Chapter Exercises**

**1**. 10 (1 + 2 + 3 + 4)

**2**. 8

**3.** 34 = 81

**4.** 301

**5**. The integer 10 has been stored in memory address 500.

**6**. int\* intPointer;  
 intPointer = (int\*) 1000;  
 \*intPointer = SAM;

**7**. The teller object can allow a customer to deposit money into an account, withdraw money from an account, and can tell a customer the current balance in the account.

**8.** After a println statement, the screen cursor goes to the next line. After a print statement, the screen cursor stays on the same line so the user input appears right after the colon.

**9.** The output is

second is bigger

first is bigger

because the else clause consists of just one line.

**10**. a. (3 <= 3) && (7 > 5) This is TRUE (both parts are true)

b. (3 < 3) || (7 > 5) This is TRUE (the second part is true)

c. (4 < 1) && (3 > 2) This is FALSE (the first part is false)

**11.** a. To compute trajectories for a satellite launcher, Fortran would work the best because Fortran is best suited to programs that require heavy mathematical computation.

b. This application requires low-level access to the details of the communications between the input device and the computer. C was designed to give this kind of low-level access to the programmer.

c. To process the day’s transactions at an ATM, COBOL would work best since COBOL was designed to handle data manipulation and report generation tasks.

**12**. It results in the names of all vendors from Dallas with whom the store has done less than $10,000 worth of business in the past quarter.

**13.** SELECT CITY  
FROM VENDOR  
WHERE PURCHASE > 10000

**14**. The sentence "The red dog chased the brown cow across the green field." would appear with the word "red" in red font, the word "brown" in brown font, and the word "green" in green font.

**16**. 21

**17**. (define (toofer input-list)

(list (car (cdr input-list)) (car input-list)))

[Then (toofer (list 3 4 5 6)) results in (4 3)]

**18**. The output is 3. This function returns the size of the list.

**19.** (define (cube x)

(\* (\* x x) x))

(define (double x)

(\* 2 x))

(define (five x)

(\* 5 x))

(define (poly x)

(+ (- (double (cube x)) (five x)) 1))

[Then (poly 2) results in 7]

**20**. The result of the function is 4\*3\*2\*1 = 24. This function returns the factorial of the number input.

**21.** a. earlier(lewisandclark, civilwar).

true b. earlier(worldwarII, firstmoonlanding).  
 true

c. earlier( X, worldwarII).

X = gettysburgaddress  
 X = civilwar

X = lewisandclark

**22**. eli mary  
 \\_\_\_\_\_\_\_\_\_\_\_\_\_/  
 |  
 bill  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_  
 / | \  
 joe betty sarah

**23**. fatherof(X,Y) :- male(X), parentof(X,Y).

**24**. daughterof(X,Y) :- female(X), parentof(Y,X).

**25**. a. ancestorof(X,Y) :- parentof(X,Y).  
 ancestorof(X,Y) :- parentof(X,Z), ancestorof(Z,Y).

b. ancestorof(X, sarah).  
 X = bill X = eli

X = Mary

**26.** Students can copy and paste both their programs and their results for each query into a Word or text document for grading purposes.

**27**. Assume that all units are μsec. Using the sequential algorithm, 7 comparisons will take place. Therefore, it will take the sequential algorithm 7 μsec to complete the Find Largest algorithm.  
  
Using the parallel algorithm, it will take   
 .003\*8 top level split  
 .003\*4 2nd level split  
 .003\*2 3rd level split  
 1 4th level comparison  
 .001 pass up  
 1 3rd level comparison  
 .001 pass up  
 1 2nd level comparison  
 .001 pass up

Total = 3.045 μsec

**Challenge Work**

1. Students should be provided sufficient documentation about Visual Basic.NET to solve this problem. If their only programming experience has been the console programming of the online modules, they will enjoy dragging the user interface objects onto the form and programming a GUI.

**2.** This would be a good chance to remind students that they can't steal images from the web unless they are specifically available for use without fee or permission.

**3.** a. 1, 1, 2, 3, 5, 8

b. 1, 2, 3, 5, 8

c. With //, the same as C or C++

d. The Go compiler automatically inserts semicolons before compilation.

e. The statement

a, b := 0, 1

assigns the value 0 to a and the value 1 to b.

The statement

a, b = b, a+b

computes the value of each expression on the right of the equals sign (i.e., the existing value of b, and the value of a + b using the existing values of a and b), then assigns those two values to a and b, respectively. For example, if a has the value 2 and b has the value 3, the expression b has the value 3 and the expression a + b has the value 2 + 3 = 5. Then the computed b value (3) is assigned to a and the computed a+b value (5) is assigned to b.

**4.** The Prolog simulator at [*http://swish.swi-prolog.org*](http://swish.swi-prolog.org)returns

precedes(fdr, kennedy).  
 true

with an indication that there are more answers, but when you ask to see more answers, the system quickly aborts with the message "out of local stack".

The first part of the "precedes" definition means that the system will check whether before(fdr, kennedy) is a fact in the Prolog database, and it is indeed a fact, so the answer returned from this check is "true", as shown above. But Prolog will go on to check the second definition of "precedes" with the result that in this process, the simulator's processor runs out of space.

In the original version, when trying to match the rule

precedes(X,Y) :- before(X,Z), precedes(Z,Y).

if the second (recursive clause) is ever tested, it checks a "before" relation ahead of trying another recursive call and there is a finite number of "before" facts to check, so this process will eventually terminate. In the new version

precedes(X,Y) :- precedes(Z,Y), before(X,Z).

the recursive part of the rule is checked first. So, where X is fdr and Y is kennedy, trying to match Z with jefferson, for example, would mean checking

**precedes(jefferson, kennedy),**  before(fdr,jefferson)

Checking the simple version of precedes(jefferson, kennedy), that is, before(jefferson,kennedy), fails so the next step is the recursive version

precedes(Z,kennedy) and before(jefferson,Z), and the recursive part of the rule is checked first. So, trying to match Z with jefferson, for example, would mean checking

**precedes(jefferson,kennedy)** and before(jefferson,Z).

And we are now back where we were earlier. The program is in an infinite loop and runs out of space to store all the intermediate results.