# Programming in C

*A complete introduction to the C programming language*

Stephen G. Kochan

Third Edition

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

# Programming in C

Third Edition

## Developer's Library

Programming in Objective-C
*Stephen G. Kochan*
*0-672-32586-1*

Unix Shell Programming, Third Edition
*Stephen G. Kochan*
*0-672-32490-3*

Microsoft Visual C# .NET 2003 Developer's Cookbook
*Mark Schmidt, Simon Robinson*
*0-672-32580-2*

ASP.NET Developer's Cookbook
*Steven Smith, Rob Howard*
*0-672-32524-1*

PHP and MySQL Web Development, 2nd Edition
*Luke T. Welling, Laura Thomson*
*0-672-32525-X*

Advanced PHP Programming
*George Schlossnagle*
*0-672-32561-6*

Perl Developer's Dictionary
*Clinton Pierce*
*0-672-32067-3*

MySQL, Second Edition
*Paul Dubois*
*0-7357-1212-3*

Apache Administrator's Handbook
*Rich Bowen, Daniel Ridrueio, Allan Liska*
*0-672-32274-9*

HTTP Developer's Handbook
*Chris Shiflett*
*0-672-32454-7*

mod_perl Developer's Cookbook
*Geoffrey Young, Paul Lindner, Randy Kobes*
*0-672-32240-4*

PostgreSQL Developer's Handbook
*Ewald Geschwinde, Hans-Juergen Schoenig*
*0-672-32260-9*

# Programming in C

Third Edition

Stephen G. Kochan

DEVELOPER'S LIBRARY

Sams Publishing, 800 East 96th Street, Indianapolis, Indiana 46240

**Programming in C, Third Edition**

Copyright © 2005 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-32666-3

Library of Congress Catalog Card Number: 2004093272

Printed in the United States of America

First Printing: July 2004

07 06 6 5 4

## Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

## Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

## Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

**U.S. Corporate and Government Sales**
**1-800-382-3419**
**corpsales@pearsontechgroup.com**

For sales outside of the United States, please contact

**International Sales**
**international@pearsoned.com**

**Associate Publisher**
Michael Stephens

**Development Editor**
Mark Renfrow

**Managing Editor**
Charlotte Clapp

**Project Editor**
Dan Knott

**Copy Editor**
Karen Annett

**Indexer**
Chris Barrick

**Proofreader**
Eileen Dennie

**Technical Editor**
Bradley L. Jones

**Publishing Coordinator**
Cindy Teeters

**Interior Designer**
Gary Adair

**Cover Designer**
Alan Clements

❖

*To my mother and father*

❖

# Contents At a Glance

Preface **xvii**

**1** Introduction **1**

**2** Some Fundamentals **5**

**3** Compiling and Running Your First Program **11**

**4** Variables, Data Types, and Arithmetic Expressions **21**

**5** Program Looping **43**

**6** Making Decisions **65**

**7** Working with Arrays **95**

**8** Working with Functions **119**

**9** Working with Structures **165**

**10** Character Strings **195**

**11** Pointers **235**

**12** Operations on Bits **279**

**13** The Preprocessor **299**

**14** More on Data Types **321**

**15** Working with Larger Programs **333**

**16** Input and Output Operations in C **347**

**17** Miscellaneous and Advanced Features **373**

**18** Debugging Programs **389**

**19** Object-Oriented Programming **411**

**A** C Language Summary **425**

**B** The Standard C Library **467**

**C** Compiling Programs with `gcc` **493**

**D** Common Programming Mistakes **497**

**E** Resources **501**

Index **505**

# Table of Contents

**1 Introduction 1**

**2 Some Fundamentals 5**

Programming 5

Higher-Level Languages 6

Operating Systems 6

Compiling Programs 7

Integrated Development Environments 10

Language Interpreters 10

**3 Compiling and Running Your First Program 11**

Compiling Your Program 11

Running Your Program 12

Understanding Your First Program 13

Displaying the Values of Variables 15

Comments 17

Exercises 19

**4 Variables, Data Types, and Arithmetic Expressions 21**

Working with Variables 21

Understanding Data Types and Constants 23

- The Basic Integer Type `int` 23
- The Floating Number Type `float` 24
- The Extended Precision Type `double` 25
- The Single Character Type `char` 25
- The Boolean Data Type `_Bool` 26
- Type Specifiers: `long`, `long long`, `short`, `unsigned`, and `signed` 28

Working with Arithmetic Expressions 30

- Integer Arithmetic and the Unary Minus Operator 33
- The Modulus Operator 35
- Integer and Floating-Point Conversions 36

Combining Operations with Assignment: The Assignment Operators 38
Types `_Complex` and `_Imaginary` 39
Exercises 40

**5 Program Looping 43**
The `for` Statement 44
  Relational Operators 46
  Aligning Output 50
  Program Input 51
  Nested `for` Loops 53
  `for` Loop Variants 54
The `while` Statement 56
The `do` Statement 60
  The `break` Statement 62
  The `continue` Statement 62
Exercises 63

**6 Making Decisions 65**
The `if` Statement 65
  The `if-else` Construct 69
  Compound Relational Tests 72
  Nested `if` Statements 75
  The `else if` Construct 76
The `switch` Statement 84
Boolean Variables 87
The Conditional Operator 91
Exercises 93

**7 Working with Arrays 95**
Defining an Array 96
  Using Array Elements as Counters 100
  Generating Fibonacci Numbers 103
  Using an Array to Generate Prime Numbers 104
Initializing Arrays 106

Character Arrays 108
Base Conversion Using Arrays 109
The `const` Qualifier 111
Multidimensional Arrays 113
Variable-Length Arrays 115
Exercises 117

**8 Working with Functions 119**
Defining a Function 119
Arguments and Local Variables 122
Function Prototype Declaration 124
Automatic Local Variables 124
Returning Function Results 126
Functions Calling Functions Calling… 131
Declaring Return Types and Argument Types 134
Checking Function Arguments 135
Top-Down Programming 137
Functions and Arrays 137
Assignment Operators 142
Sorting Arrays 143
Multidimensional Arrays 146
Global Variables 152
Automatic and Static Variables 156
Recursive Functions 159
Exercises 162

**9 Working with Structures 165**
A Structure for Storing the Date 166
Using Structures in Expressions 168
Functions and Structures 171
A Structure for Storing the Time 177
Initializing Structures 180
Compound Literals 181
Arrays of Structures 182
Structures Containing Structures 185

Structures Containing Arrays 187

Structure Variants 190

Exercises 191

**10 Character Strings 195**

Arrays of Characters 196

Variable-Length Character Strings 198

Initializing and Displaying Character Strings 201

Testing Two Character Strings for Equality 204

Inputting Character Strings 206

Single-Character Input 208

The Null String 213

Escape Characters 216

More on Constant Strings 218

Character Strings, Structures, and Arrays 219

A Better Search Method 222

Character Operations 227

Exercises 230

**11 Pointers 235**

Defining a Pointer Variable 235

Using Pointers in Expressions 239

Working with Pointers and Structures 240

Structures Containing Pointers 243

Linked Lists 244

The Keyword `const` and Pointers 253

Pointers and Functions 254

Pointers and Arrays 259

A Slight Digression About Program Optimization 263

Is It an Array or Is It a Pointer? 264

Pointers to Character Strings 266

Constant Character Strings and Pointers 267

The Increment and Decrement Operators Revisited 268

Operations on Pointers 272

Pointers to Functions 273
Pointers and Memory Addresses 274
Exercises 276

**12 Operations on Bits 279**
Bit Operators 280
The Bitwise AND Operator 281
The Bitwise Inclusive-OR Operator 283
The Bitwise Exclusive-OR Operator 284
The Ones Complement Operator 285
The Left Shift Operator 287
The Right Shift Operator 287
A Shift Function 288
Rotating Bits 290
Bit Fields 292
Exercises 297

**13 The Preprocessor 299**
The `#define` Statement 299
Program Extendability 303
Program Portability 305
More Advanced Types of Definitions 306
The `#` Operator 312
The `##` Operator 313
The `#include` Statement 313
System Include Files 316
Conditional Compilation 316
The `#ifdef`, `#endif`, `#else`, and `#ifndef` Statements 316
The `#if` and `#elif` Preprocessor Statements 318
The `#undef` Statement 319
Exercises 320

**14 More on Data Types 321**
Enumerated Data Types 321
The `typedef` Statement 325

Data Type Conversions 327
Sign Extension 329
Argument Conversion 329
Exercises 330

**15 Working with Larger Programs 333**
Dividing Your Program into Multiple Files 333
Compiling Multiple Source Files from the Command Line 334
Communication Between Modules 336
External Variables 336
`Static` Versus `Extern` Variables and Functions 339
Using Header Files Effectively 341
Other Utilities for Working with Larger Programs 342
The `make` Utility 343
The `cvs` Utility 344
Unix Utilities: `ar`, `grep`, `sed`, and so on 345

**16 Input and Output Operations in C 347**
Character I/O: `getchar` and `putchar` 348
Formatted I/O: `printf` and `scanf` 348
The `printf` Function 348
The `scanf` Function 355
Input and Output Operations with Files 359
Redirecting I/O to a File 359
End of File 361
Special Functions for Working with Files 363
The `fopen` Function 363
The `getc` and `putc` Functions 365
The `fclose` Function 365
The `feof` Function 367
The `fprintf` and `fscanf` Functions 368
The `fgets` and `fputs` Functions 368
`stdin`, `stdout`, and `stderr` 369
The `exit` Function 370
Renaming and Removing Files 371
Exercises 371

**17 Miscellaneous and Advanced Features 373**

Miscellaneous Language Statements 373

The `goto` Statement 373

The `null` Statement 374

Working with Unions 375

The Comma Operator 378

Type Qualifiers 378

The `register` Qualifier 378

The `volatile` Qualifier 379

The `restrict` Qualifier 379

Command-Line Arguments 380

Dynamic Memory Allocation 383

The `calloc` and `malloc` Functions 384

The `sizeof` Operator 385

The `free` Function 387

**18 Debugging Programs 389**

Debugging with the Preprocessor 389

Debugging Programs with `gdb` 395

Working with Variables 398

Source File Display 399

Controlling Program Execution 400

Getting a Stack Trace 405

Calling Functions and Setting Arrays and Structures 405

Getting Help with `gdb` Commands 406

Odds and Ends 408

**19 Object-Oriented Programming 411**

What Is an Object Anyway? 411

Instances and Methods 412

Writing a C Program to Work with Fractions 413

Defining an Objective-C Class to Work with Fractions 414

Defining a C++ Class to Work with Fractions 419

Defining a C# Class to Work with Fractions 422

**A C Language Summary 425**

1.0 Digraphs and Identifiers 425

1.1 Digraph Characters 425

1.2 Identifiers 425

2.0 Comments 426

3.0 Constants 427

3.1 Integer Constants 427

3.2 Floating-Point Constants 427

3.3 Character Constants 428

3.4 Character String Constants 429

3.5 Enumeration Constants 430

4.0 Data Types and Declarations 430

4.1 Declarations 430

4.2 Basic Data Types 430

4.3 Derived Data Types 432

4.4 Enumerated Data Types 438

4.5 The `typedef` Statement 438

4.6 Type Modifiers `const`, `volatile`, and `restrict` 439

5.0 Expressions 439

5.1 Summary of C Operators 440

5.2 Constant Expressions 442

5.3 Arithmetic Operators 443

5.4 Logical Operators 444

5.5 Relational Operators 444

5.6 Bitwise Operators 445

5.7 Increment and Decrement Operators 445

5.8 Assignment Operators 446

5.9 Conditional Operators 446

5.10 Type Cast Operator 446

5.11 `sizeof` Operator 447

5.12 Comma Operator 447

5.13 Basic Operations with Arrays 447

5.14 Basic Operations with Structures 448

5.15 Basic Operations with Pointers 448

5.16 Compound Literals 450

5.17 Conversion of Basic Data Types 451

6.0 Storage Classes and Scope 452
- 6.1 Functions 452
- 6.2 Variables 452

7.0 Functions 454
- 7.1 Function Definition 454
- 7.2 Function Call 455
- 7.3 Function Pointers 456

8.0 Statements 456
- 8.1 Compound Statements 456
- 8.2 The `break` Statement 456
- 8.3 The `continue` Statement 457
- 8.4 The `do` Statement 457
- 8.5 The `for` Statement 457
- 8.6 The `goto` Statement 458
- 8.7 The `if` Statement 458
- 8.8 The `null` Statement 458
- 8.9 The `return` Statement 459
- 8.10 The `switch` Statement 459
- 8.11 The `while` Statement 460

9.0 The Preprocessor 460
- 9.1 Trigraph Sequences 460
- 9.2 Preprocessor Directives 461
- 9.3 Predefined Identifiers 466

**B The Standard C Library 467**

Standard Header Files 467
- `<stddef.h>` 467
- `<limits.h>` 468
- `<stdbool.h>` 469
- `<float.h>` 469
- `<stdint.h>` 469

String Functions 470

Memory Functions 472

Character Functions 473

I/O Functions 473

In-Memory Format Conversion Functions 478

String-to-Number Conversion 479

Dynamic Memory Allocation Functions 481
Math Functions 482
    Complex Arithmetic 488
General Utility Functions 490

**C Compiling Programs with `gcc` 493**
General Command Format 493
Command-Line Options 494

**D Common Programming Mistakes 497**

**E Resources 501**
Answers to Exercises, Errata, etc. 501
The C Programming Language 501
    Books 501
    Web Sites 502
    Newsgroups 502
C Compilers and Integrated Development Environments 502
    gcc 502
    MinGW 502
    CygWin 502
    Visual Studio 503
    CodeWarrior 503
    Kylix 503
Miscellaneous 503
    Object-Oriented Programming 503
    The C++ Language 503
    The C# Language 503
    The Objective-C Language 503
    Development Tools 504

**Index 505**

# Preface

It's hard to believe that 20 years have passed since I first wrote *Programming in C.* At that time the Kernighan & Ritchie book *The C Programming Language* was the only other book on the market. How times have changed!

When talk about an ANSI C standard emerged in the early 1980s, this book was split into two titles: The original was still called *Programming in C*, and the title that covered ANSI C was called *Programming in ANSI C.* This was done because it took several years for the compiler vendors to release their ANSI C compilers and for them to become ubiquitous. I felt it was too confusing to try to cover both ANSI and non-ANSI C in the same tutorial text, thus the reason for the split.

The ANSI C standard has changed several times since the first standard was published in 1989. The latest version, called C99, is the major reason for this edition. This edition addresses the changes made to the language as a result of that standard.

In addition to covering C99 features, this book also includes two new chapters. The first discusses debugging C programs. The second offers a brief overview of the pervasive field of object-oriented programming, or OOP. This chapter was added because several popular OOP languages are based on C: C++, C#, Java, and Objective-C.

For those who have stayed with this text through the years, I am sincerely grateful. The feedback I have received has been enormously gratifying. It remains my main motivation for continuing to write today.

For newcomers, I welcome your input and hope that this book satisfies your expectations.

Stephen Kochan  
June 2004  
steve@kochan-wood.com

## About the Author

**Stephen Kochan** has been developing software with the C programming language for over 20 years. He is the author and coauthor of several bestselling titles on the C language, including *Programming in C*, *Programming in ANSI C*, and *Topics in C Programming*, and several Unix titles, including *Exploring the Unix System*, *Unix Shell Programming*, and *Unix System Security.* Mr. Kochan's most recent title, *Programming in Objective-C,* is a tutorial on an object-oriented programming language that is based on C.

## Acknowledgements

I wish to thank the following people for their help in the preparation of various versions of this text: Douglas McCormick, Jim Scharf, Henry Tabickman, Dick Fritz, Steve Levy, Tony Ianinno, and Ken Brown. I also want to thank Henry Mullish of New York University for teaching me so much about writing and for getting me started in the publishing business.

From Sams Publishing, I'd like to thank my development editor Mark Renfrow and my project editor Dan Knott. Thanks also to my copy editor, Karen Annett, and my technical editor, Bradley Jones. Finally, I'd like to thank all the other people from Sams who were involved on this project, even if I did not work with them directly.

## We Want to Hear from You

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an associate publisher for Sams Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books better.

*Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.*

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@samspublishing.com

Mail: Michael Stephens
Associate Publisher
Sams Publishing
800 East 96th Street
Indianapolis, IN 46240 USA

For more information about this book or another Sams Publishing title, visit our Web site at `www.samspublishing.com`. Type the ISBN (excluding hyphens) or the title of a book in the Search field to find the page you're looking for.

# 3

# Compiling and Running Your First Program

IN THIS CHAPTER, YOU ARE INTRODUCED to the C language so that you can see what programming in C is all about. What better way to gain an appreciation for this language than by taking a look at an actual program written in C?

To begin with, you'll choose a rather simple example—a program that displays the phrase "Programming is fun." in your window. Program 3.1 shows a C program to accomplish this task.

Program 3.1 **Writing Your First C Program**

```
#include <stdio.h>

int main (void)
{
     printf ("Programming is fun.\n");

     return 0;
}
```

In the C programming language, lowercase and uppercase letters are distinct. In addition, in C, it does not matter where on the line you begin typing—you can begin typing your statement at any position on the line. This fact can be used to your advantage in developing programs that are easier to read. Tab characters are often used by programmers as a convenient way to indent lines.

## Compiling Your Program

Returning to your first C program, you first need to type it into a file. Any text editor can be used for this purpose. Unix users often use an editor such as vi or emacs.

Most C compilers recognize filenames that end in the two characters "." and "c" as C programs. So, assume you type Program 3.1 into a file called `prog1.c`. Next, you need to compile the program.

Using the GNU C compiler, this can be as simple as issuing the `gcc` command at the terminal followed by the filename, like this:

```
$ gcc prog1.c
$
```

If you're using the standard Unix C compiler, the command is `cc` instead of `gcc`. Here, the text you typed is entered in bold. The dollar sign is your command prompt if you're compiling your C program from the command line. Your actual command prompt might be some characters other than the dollar sign.

If you make any mistakes keying in your program, the compiler lists them after you enter the `gcc` command, typically identifying the line numbers from your program that contain the errors. If, instead, another command prompt appears, as is shown in the preceding example, no errors were found in your program.

When the compiler compiles and links your program, it creates an executable version of your program. Using the GNU or standard C compiler, this program is called `a.out` by default. Under Windows, it is often called `a.exe` instead.

## Running Your Program

You can now run the executable by simply typing its name on the command line[1]:

```
$ a.out
Programming is fun.
$
```

You can also specify a different name for the executable file at the time the program is compiled. This is done with the `-o` (that's the letter O) option, which is followed by the name of the executable. For example, the command line

```
$ gcc prog1.c -o prog1
```

compiles the program `prog1.c`, placing the executable in the file `prog1`, which can subsequently be executed just by specifying its name:

```
$ prog1
Programming is fun.
$
```

1. If you get an error like this: `a.out: No such file or directory`, then it probably means the current directory is not in your PATH. You can either add it to your PATH or type the following instead at the command prompt: `./a.out`.

## Understanding Your First Program

Take a closer look at your first program. The first line of the program

```
#include <stdio.h>
```

should be included at the beginning of just about every program you write. It tells the compiler information about the `printf` output routine that is used later in the program. Chapter 13, "The Preprocessor," discusses in detail what this line does.

The line of the program that reads

```
int main (void)
```

informs the system that the name of the program is `main`, and that it *returns* an integer value, which is abbreviated "`int`." `main` is a special name that indicates precisely *where* the program is to begin execution. The open and close parentheses immediately following `main` specify that `main` is the name of a *function*. The keyword `void` that is enclosed in the parentheses specifies that the function `main` takes no arguments (that is, it is *void* of arguments). These concepts are explained in great detail in Chapter 8, "Working with Functions."

Now that you have identified `main` to the system, you are ready to specify precisely what this routine is to perform. This is done by enclosing all program statements of the routine within a pair of curly braces. All program statements included between the braces are taken as part of the `main` routine by the system. In Program 3.1, you have only two such statements. The first statement specifies that a routine named `printf` is to be invoked or *called*. The parameter or *argument* to be passed to the `printf` routine is the string of characters

```
"Programming is fun.\n"
```

The `printf` routine is a function in the C library that simply prints or displays its argument (or arguments, as you will see shortly) on your screen. The last two characters in the string, namely the backslash (`\`) and the letter `n`, are known collectively as the *newline* character. A newline character tells the system to do precisely what its name implies—that is, go to a new line. Any characters to be printed after the newline character then appear on the next line of the display. In fact, the newline character is similar in concept to the carriage return key on a typewriter. (Remember those?)

All program statements in C *must* be terminated by a semicolon (`;`). This is the reason for the semicolon that appears immediately following the closing parenthesis of the `printf` call.

The last statement in main that reads

```
return 0;
```

says to finish execution of `main`, and return to the system a status value of `0`. You can use any integer here. Zero is used by convention to indicate that the program completed successfully—that is, without running into any errors. Different numbers can be used to indicate different types of error conditions that occurred (such as a file not being found). This exit status can be tested by other programs (such as the Unix shell) to see whether the program ran successfully.

Now that you've finished analyzing your first program, you can modify it to also display the phrase "And programming in C is even more fun." This can be done by the simple addition of another call to the `printf` routine, as shown in Program 3.2. Remember that every C program statement must be terminated by a semicolon.

Program 3.2

```
#include <stdio.h>

int main (void)
{
     printf ("Programming is fun.\n");
     printf ("And programming in C is even more fun.\n");

     return 0;
}
```

If you type in Program 3.2 and then compile and execute it, you can expect the following output in your program's output window, sometimes called the "console."

Program 3.2 **Output**

```
Programming is fun.
And programming in C is even more fun.
```

As you will see from the next program example, it is not necessary to make a separate call to the `printf` routine for each line of output. Study the program listed in Program 3.3 and try to predict the results before examining the output. (No cheating now!)

Program 3.3 **Displaying Multiple Lines of Output**

```
#include <stdio.h>

int main (void)
{
     printf ("Testing...\n..1\n...2\n....3\n");

     return 0;
}
```

Program 3.3 **Output**

```
Testing...
..1
...2
....3
```

## Displaying the Values of Variables

The `printf` routine is the most commonly used routine in this book. This is because it provides an easy and convenient means to display program results. Not only can simple phrases be displayed, but the values of *variables* and the results of computations can also be displayed. In fact, Program 3.4 uses the `printf` routine to display the results of adding two numbers, namely 50 and 25.

Program 3.4 **Displaying Variables**

```
#include <stdio.h>

int main (void)
{
     int sum;

     sum = 50 + 25;
     printf ("The sum of 50 and 25 is %i\n", sum);

     return 0;
}
```

Program 3.4 **Output**

```
The sum of 50 and 25 is 75
```

In Program 3.4, the first C program statement *declares* the *variable* `sum` to be of type *integer*. C requires that all program variables be declared before they are used in a program. The declaration of a variable specifies to the C compiler how a particular variable will be used by the program. This information is needed by the compiler to generate the correct instructions to store and retrieve values into and out of the variable. A variable declared as type `int` can only be used to hold integral values; that is, values without decimal places. Examples of integral values are 3, 5, –20, and 0. Numbers with decimal places, such as 3.14, 2.455, and 27.0, for example, are known as *floating-point* or *real* numbers.

The integer variable `sum` is used to store the result of the addition of the two integers 50 and 25. A blank line was intentionally left following the declaration of this variable to visually separate the variable declarations of the routine from the program statements; this is strictly a matter of style. Sometimes, the addition of a single blank line in a program can help to make the program more readable.

The program statement

```
sum = 50 + 25;
```

reads as it would in most other programming languages: The number `50` is added (as indicated by the plus sign) to the number `25`, and the result is stored (as indicated by the *assignment operator*, the equal sign) in the variable `sum`.

The `printf` routine call in Program 3.4 now has two items or *arguments* enclosed within the parentheses. These arguments are separated by a comma. The first argument to the `printf` routine is always the character string to be displayed. However, along with the display of the character string, you might also frequently want to have the value of certain program variables displayed. In this case, you want to have the value of the variable `sum` displayed at the terminal after the characters

```
The sum of 50 and 25 is
```

are displayed. The percent character inside the first argument is a special character recognized by the `printf` function. The character that immediately follows the percent sign specifies what *type* of value is to be displayed at that point. In the preceding program, the letter `i` is recognized by the `printf` routine as signifying that an integer value is to be displayed.[2]

Whenever the `printf` routine finds the `%i` characters inside a character string, it automatically displays the value of the next argument to the `printf` routine. Because `sum` is the next argument to `printf`, its value is automatically displayed after the characters "The sum of 50 and 25 is" are displayed.

Now try to predict the output from Program 3.5.

Program 3.5 **Displaying Multiple Values**

```
#include <stdio.h>

int main (void)
{
     int  value1, value2, sum;

     value1 = 50;
     value2 = 25;
     sum = value1 + value2;
     printf ("The sum of %i and %i is %i\n", value1, value2, sum);

     return 0;
}
```

Program 3.5 **Output**

```
The sum of 50 and 25 is 75
```

2. Note that `printf` also allows you to specify `%d` format characters to display an integer. This book consistently uses `%i` throughout the remaining chapters.

The first program statement declares three variables called `value1`, `value2`, and `sum` all to be of type `int`. This statement could have equivalently been expressed using three separate declaratory statements as follows:

```
int value1;
int value2;
int sum;
```

After the three variables have been declared, the program assigns the value `50` to the variable `value1` and then assigns `25` to `value2`. The sum of these two variables is then computed, and the result is assigned to the variable `sum`.

The call to the `printf` routine now contains four arguments. Once again, the first argument, commonly called the *format string*, describes to the system how the remaining arguments are to be displayed. The value of `value1` is to be displayed immediately following the display of the characters "The sum of." Similarly, the values of `value2` and `sum` are to be printed at the appropriate points, as indicated by the next two occurrences of the `%i` characters in the format string.

## Comments

The final program in this chapter (Program 3.6) introduces the concept of the *comment*. A comment statement is used in a program to document a program and to enhance its readability. As you will see from the following example, comments serve to tell the reader of the program—the programmer or someone else whose responsibility it is to maintain the program—just what the programmer had in mind when he or she wrote a particular program or a particular sequence of statements.

Program 3.6 **Using Comments in a Program**

```
/* This program adds two integer values
   and displays the results             */

#include <stdio.h>

int main (void)
{
    // Declare variables
    int  value1, value2, sum;

    // Assign values and calculate their sum
    value1 = 50;
    value2 = 25;
    sum = value1 + value2;
```

Program 3.6 **Continued**

```
    // Display the result
    printf ("The sum of %i and %i is %i\n", value1, value2, sum);

    return 0;
}
```

Program 3.6 **Output**

```
The sum of 50 and 25 is 75
```

There are two ways to insert comments into a C program. A comment can be initiated by the two characters `/` and `*`. This marks the *beginning* of the comment. These types of comments have to be *terminated*. To end the comment, the characters `*` and `/` are used without any embedded spaces. All characters included between the opening `/*` and the closing `*/` are treated as part of the comment statement and are ignored by the C compiler. This form of comment is often used when comments span several lines in the program. The second way to add a comment to your program is by using two consecutive slash characters `//`. Any characters that follow these slashes up to the end of the line are ignored by the compiler.

In Program 3.6, four separate comment statements were used. This program is otherwise identical to Program 3.5. Admittedly, this is a contrived example because only the first comment at the head of the program is useful. (Yes, it is possible to insert so many comments into a program that the readability of the program is actually degraded instead of improved!)

The intelligent use of comment statements inside a program cannot be overemphasized. Many times, a programmer returns to a program that he coded perhaps only six months ago, only to discover to his dismay that he could not for the life of him remember the purpose of a particular routine or of a particular group of statements. A simple comment statement judiciously inserted at that particular point in the program might have saved a significant amount of time otherwise wasted on rethinking the logic of the routine or set of statements.

It is a good idea to get into the habit of inserting comment statements into the program as the program is being written or typed in. There are good reasons for this. First, it is far easier to document the program while the particular program logic is still fresh in your mind than it is to go back and rethink the logic after the program has been completed. Second, by inserting comments into the program at such an early stage of the game, you get to reap the benefits of the comments during the debug phase, when program logic errors are being isolated and debugged. A comment can not only help you read through the program, but it can also help point the way to the source of the logic mistake. Finally, I have yet to discover a programmer who actually enjoyed documenting a program. In fact, after you have finished debugging your program, you will probably not relish the idea of going back to the program to insert comments. Inserting comments while developing the program makes this sometimes tedious task a bit easier to swallow.

This concludes this introductory chapter on developing programs in C. By now, you should have a good feel as to what is involved in writing a program in C, and you should be able to develop a small program on your own. In the next chapter, you begin to learn some of the finer intricacies of this wonderfully powerful and flexible programming language. But first, try your hand at the following exercises to make certain you understand the concepts presented in this chapter.

## Exercises

1. Type in and run the six programs presented in this chapter. Compare the output produced by each program with the output presented after each program in the text.
2. Write a program that prints the following text at the terminal.
    1. In C, lowercase letters are significant.
    2. main is where program execution begins.
    3. Opening and closing braces enclose program statements in a routine.
    4. All program statements must be terminated by a semicolon.
3. What output would you expect from the following program?
```
#include <stdio.h>

int main (void)
{
    printf ("Testing...");
    printf ("....1");
    printf ("...2");
    printf ("..3");
    printf ("\n");

    return 0;
}
```
4. Write a program that subtracts the value `15` from `87` and displays the result, together with an appropriate message, at the terminal.
5. Identify the syntactic errors in the following program. Then type in and run the corrected program to ensure you have correctly identified all the mistakes.
```
#include <stdio.h>

int main (Void)
(
        INT  sum;
        /* COMPUTE RESULT
        sum = 25 + 37 - 19
        /* DISPLAY RESULTS //
        printf ("The answer is %i\n" sum);
        return 0;
}
```

6. What output might you expect from the following program?

```
#include <stdio.h>

int main (void)
{
      int answer, result;

      answer = 100;
      result = answer - 10;
      printf ("The result is %i\n", result + 5);

      return 0;
}
```

# Index

## Symbols

**\' (single quote) escape character, 217**

**\" (double quote) escape character, 217**

**\? (question mark) escape character, 217**

**\a (audible alert) escape character, 216**

**\b (backspace) escape character, 216**

**\n (newline character), program syntax, 13**

**\nnn (octal character value) escape character, 217**

**\t (horizontal tab) escape character, 216**

**\xnn (hexadecimal character value) escape character, 217**

**\\ (backslash) escape character, 217**

**^ (bitwise Exclusive-OR) operator, 284-285**

**_Bool data type, 23, 26**

**_Complex data type, 39**

**_Imaginary data type, 39**

**{} (braces), program syntax, 13**

**| (bitwise Inclusive-OR) operator, 283-284**

**|| (logical OR) operator, compound relationship tests, 72-73**

**; (semicolon)**
- #define statement, 306-307
- program syntax, 13

**! (logical negation) operator, Boolean variables, 90**

**!= (not equal to) operator, 46-50**

**' (single quotation marks), char data type, 25-26**

**" (double quotation marks), char data type, 25-26**

**# operator, macros, 312**

**## operator, macros, 313**

**#define statement, 299-303, 461-463**
- arguments, 308-311
- defined names, 300
- definition types, 306-308
- Introducing the #define Statement (Program 13.1), 300-302
- macros, 308-311
  - *converting character case, 311*
  - *defining number of arguments, 311*
  - *testing for lowercase characters, 310*
- More on Working with Defines (Program 13.2), 302-303
- program extendability, 303-305
- program portability, 305-306
- semicolon (;), 306-307

**#elif statement, 318-319**

**#else statement, conditional compilation, 316-318**

**#endif statement, conditional compilation, 316-318**

**#error statement, 463**

**#if statement, 318-319, 463**

**#ifdef statement, 316-318, 464**

**#ifndef statement, 316-318, 464**

**#include statement, 464-465**
- macro definition collections, 313-315
- Using the #include Statement (Program 13.3), 314-315

**#line statement, 465**

**#pragma statement, 465**

**#undef statement, 319, 465**

**% (modulus) arithmetic operator, 35-36**

**& (address) operator, 236, 260**

**& (bitwise AND) operator, 281-283**

**&& (logical AND) operator, compound relationship tests, 72-73**

*** (indirection) operator, 236**

*** (multiplication sign) arithmetic operator, 30-33**

***/ (closing comments), 18**

***= (times equal) operator, 143**

**+ (plus sign) arithmetic operator, 30-33**

**++ (increment) operator, 49, 262, 268**

**- (minus sign)**
- arithmetic operator, 30-33
- unary arithmetic operator, 33-34

**-- (decrement) operator, 50, 262, 268, 445**

**/ (division sign) arithmetic operator, 30-33**

**/* (opening comments), 18**

**< (less than) operator, 46-50**

**<< (left shift) bitwise operator, 287**

**<= (less than or equal to) operator, 46-50**

**= (assignment) operator, 15**

**== (equal to) operator, 46-50**

**?\ (conditional) operator, ternary nature of, 91-92**

## A

**A Simple Program for Use with gdb (Program 18.4), 396-398**

**abs( ) function, 490**

**absolute value of numbers, calculating, 129-131**

**absolute_value( ) function, 129-131**

**acos( ) function, 483**

**Adding Debug Statements with the Preprocessor (Program 18.1), 389-391**

**address (&) operator, 236, 260**

**adjacent strings, 218**

**algorithms**
- binary search, 223-227
- function of, 5
- Sieve of Erastosthenes (prime numbers), 118

**aligning triangular number output, 50-51**
- field width specification, 51
- right justification, 51

**alphabetic( ) function, 212-214**

**American National Standard Institute.** ***See* ANSI**

**ANSI (American National Standards Institute), 1**
- C standardization efforts, 1
- C99 standard, 425
- Web site, 502

**ar utility, programming functionality, 345**

**argc argument, 380**

**arguments, 16**
- #define statement (macros), 308-311
- argc, 380
- argv, 380
- calling, 13
- command-line, 380
  - *File Copy Program Using Command-Line Arguments (Program 17.1), 382-383*
  - *main( ) function, 380-381*
  - *storing, 383*
- data types, conversion of, 329-330
- format string, 17

functions, 122-123

*declaring, 134-135*

*formal parameter name, 124*

*values, checking, 135-137*

pointer arguments, passing, 254-257

sizeof operator, 385

**argv argument, 380**

**arithmetic operators, 443-444**

associative property, 30

binary, 30-33

division sign (/), 30-33

joining with assignment operators, 38-39

minus sign (-), 30-33

modulus (%), 35-36

More Examples with Arithmetic Operators (Program 4.3), 33-34

multiplication sign (*), 30-33

plus sign (+), 30-33

precedence, 30

*rules example, 34*

type cast, precedence rules, 38

unary minus, 33-34

Using the Arithmetic Operators (Program 4.2), 30-31

**arithmetic right shift, 288**

**array of characters, Concatenating Character Arrays (Program 10.1), 196-198**

**array operators, 447-448**

**array_sum( ) function, 262-264**

**arrays**

characters, 108-109

*memory functions, 472*

const variable, 111-113

containment by structures, 187-189

declaring, 97-98

defining with unions, 376-377

dynamic memory allocation, 117

elements

*as counters, 100-103*

*initializing, 106-108*

*sequencing through, 96-100*

Fibonacci numbers, generating, 103-104

function of, 95

functions, passing multidimensional arrays, 146-152

integer bases, conversion of, 109-111

multidimensional, 113-114, 433-434

*initializing, 114-115*

*passing to functions, 146-152*

multidimensional arrays, declaring, 114

passing to functions, 137-142

*assignment operators, 142-143*

pointers to, 259-260, 449-450

*to character string, 266-267*

*decrement (--) operator, 262, 268*

*increment (++) operator, 262, 268*

*postdecrement operator, 269-271*

*postincrement operator, 269-271*

*predecrement operator, 269-271*

*preincrement operator, 269-271*

*program optimization, 263-264*

*sequencing through pointer elements, 261*

prime numbers, generating, 104-106

programs

*Converting a Positive Integer to Another Base (7.7), 110-111*

*Demonstrating an Array of Counters (7.2), 101-103*

*Finding the Minimum Value in an Array (8.9), 138-140*

*Generating Fibonacci Numbers (7.3), 103-104*

*Generating Fibonacci Numbers Using Variable-Length Arrays (7.8), 115-117*

*Illustrating Structures and Arrays (9.7), 188-189*

*Initializing Arrays (7.5), 107-108*
*Introducing Character Arrays (7.6), 108-109*
*Multidimensional Variable-Length Arrays (8.13A), 150-152*
*Revising the Function to Find the Minimum Value in an Array (8.10), 140-142*
*Revising the Program to Generate Prime Numbers, Version 2 (7.4), 105-106*
*Sorting an Array of Integers into Ascending Order (8.12), 144-146*
*Using Multidimensional Arrays and Functions (8.13), 147-150*
*Working with an Array (7.1), 98-100*
sequencing through elements with pointers to arrays, 261
single-dimensional, 432-433
sorting, 143-146, 490-491
structures
*defining, 182*
*initializing, 183*
*Using the Dictionary Lookup Program (Program 10.9), 220-222*
subscripts, 96
summing elements, 262-264
uses, 95
values, storing, 96
variable-length, 115-117, 433
variables, defining, 96-98
versus pointers, differentiating, 264-265
**asin( ) function, 483**
**asinh( ) function, 483**
**Asking the User for Input (Program 5.4), 51-52**
**assemblers, 6**
programs, compiling, 9
**assigning structure values via compound literals, 181-182**
**assignment operators, 15, 142-143, 446**
joining with arithmetic operators, 38-39
**AT&T Bell Laboratories, 1**
**atan( ) function, 484**
**atan2( ) function, 484**
**atanh( ) function, 484**
**atof( ) function, 480**
**atoi( ) function, 230, 480**
**atol( ) function, 480**
**audible alert (\a) escape character, 216**
**auto keyword, 124-126, 156**
**auto_static( ) function, 157-158**
**automatic local variables, 156**
functions, 124-126

## B

**backslash (\\) escape character, 217**
**backspace (\b) escape character, 216**
**backtrace command (gdb debugger), 405**
**base notations, int data types, 23-24**
**bases, integers, converting via arrays, 109-111**
**basic data types**
C language specifications, 430-432
usual arithmetic conversion, 451-452
**BASIC programming language, 10**
**beginning comments, character syntax, 18**
**binary arithmetic operators, 30-33**
**binary files, opening, 475**
**binary search algorithm, 223-227**
**bit fields, 292-294**
declaring, 296
defining, 294-295
extracting values, 295
units, 296

**bits, 279**
bit fields, 292-294
high-order, 279
least significant, 279
low-order, 279
most significant, 279
operators, 280
*& (bitwise AND), 281-283*
*<< (left shift), 287*
*^ (bitwise Exclusive-OR), 284-285*
*| (bitwise Inclusive-OR), 283-284*
rotating values, 290, 292
**bitwise operators, 445**
Illustrating Bitwise Operators (Program 12.2), 286-287
**book resources**
*The C Programming Language*, 501
*The C Reference Manual*, 501
*C# Programming in the Key of C#*, 503
*C++ Primer Plus, 4th Edition*, 503
*Introduction of Object-Oriented Programming, 3rd Edition*, 503
*Objective-C Programming Language*, 504
*Programming in Objective-C*, 503
*The Standard C Library*, 501
**Boolean variables**
logical negation (!) operator, 90
programs
*Generating a Table of Prime Numbers (6.10), 87-90*
*Revising the Program to Generate a Table of Prime Numbers (6.10A), 90-91*
**braces ({}), program syntax, 13**
**break command (gdb debugger), 400, 409**
**break statement, 62, 84, 456**
**breakpoints in programs, debugging (gdb tool), 400**
**bugs in programs, 9**
**bytes, 279**

## C

**C language**
ANSI standardization efforts, 1
arithmetic operators, 443-444
array pointers, 449-450
arrays
*multidimensional, 433-434*
*operators, 447-448*
*single-dimensional, 432-433*
*variable-length, 433*
assignment operators, 446
AT&T Bell Laboratories, 1
basic data type conversion, usual arithmetic conversion, 451-452
as basis for Unix operating system, 1
bitwise operators, 445
book resources
The C Programming Language*, 501*
The C Reference Manual*, 501*
The Standard C Library*, 501*
character constants, 428
*escape sequences, 428-429*
*wide character, 429*
character string constants, 429
*concatenation, 429*
*multibyte, 429*
comma operators, 447
comments, 426
compound literals, 450-451
conditional operators, 446
constant expressions, 442-443
data types
*basic, 430-432*
*declarations, 430*
*derived, 432-438*
*enumerated, 438*
*modifiers, 439*
*typedef statement, 438-439*
decrement operators, 445

digraph characters, 425
enumeration constants, 430
expressions, specifications, 439
filename extension, 7
floating-point constants, 427-428
fractions program, writing, 413-414
functions
    *calls, 455-456*
    *definition of, 454-455*
    *pointers, 456*
identifiers, 425
    *keywords, 426*
    *universal character names, 426*
increment operators, 445
integer constants, 427
interpreters, 10
ISO standardization efforts, 1
logical operators, 444
operators, summary table, 440-442
origins, 1
pointers
    *declarations, 437-438*
    *operators, 448*
predefined identifiers, 466
preprocessor, 460
    *directives, 461-465*
    *trigraph sequences, 460-461*
relational operators, 444-445
scopes, 452
sizeof operators, 447
statements
    *break, 456*
    *compound, 456*
    *continue, 457*
    *do, 457*
    *for, 457*
    *goto, 458*
    *if, 458*
    *null, 458*
    *return, 459*
    *switch, 459-460*
    *while, 460*
storage classes
    *functions, 452*
    *variables, 452-454*
structures
    *declarations, 434-436*
    *operators, 448*
    *pointers, 450*
text editors, 7
type cast operators, 446
unions, declarations, 436-437
vendor marketing, 1
Web site resources, Kochan-Wood.com, 502

**C preprocessor**
conditional compilation, 316
    *#else statement, 316-318*
    *#endif statement, 316-318*
    *#ifdef statement, 316-318*
    *#ifndef statement, 316-318*
statements, #define, 299-303

***The C Programming Language*, 501**

***The C Reference Manual*, 501**

**C# language**
development history, 422
fractions program, writing, 422-424

***C# Programming in the Key of C#*, 503**

**C++ language**
development history, 419
fractions program, writing, 419-421

***C++ Primer Plus, 4th Edition*, 503**

**cabs( ) function, 488**

**cacos( ) function, 488**

**cacosh( ) function, 488**

**calculating**
- absolute value of numbers, 129-131
- square roots, 131-133
- triangular numbers
  - *nested for loops (program looping), 53-54*
  - *output alignment (program looping), 50-51*
  - *program looping, 43-45*
  - *user input (program looping), 51-52*

**Calculating Factorials Recursively (Program 8.16), 159-161**

**Calculating the 200th Triangular Number (Program 5.2), 44-45**

**Calculating the Absolute Value (Program 8.7), 129-131**

**Calculating the Absolute Value of an Integer (Program 6.1), 66-67**

**Calculating the Average of a Set of Grades (Program 6.2), 67-69**

**Calculating the Eighth Triangular Number (Program 5.1), 43**

**Calculating the nth Triangular Number (Program 8.4), 123**

**Calculating the Square Root of a Number (Program 8.8), 132-133**

**call stacks (traces), 405**

**calling**
- functions, 121-122
  - *C language specifications, 455-456*
- statements, 13

**Calling Functions (Program 8.2), 121**

**calloc( ) function, 386, 481**
- dynamic memory allocation, 384-385

**carq( ) function, 488**

**case sensitivity in programming, 11**

**casin( ) function, 488**

**casinh( ) function, 488**

**catan( ) function, 489**

**catanh( ) function, 489**

**Categorizing a Single Character Entered at the Terminal (Program 6.7), 78-80**

**cc command (Unix), 7-9**

**ccos( ) function, 489**

**ccosh( ) function, 489**

**ceil( ) function, 484**

**cexp( ) function, 489**

**Changing Array Elements in Functions (Program 8.11), 142-143**

**char data type, 23**
- quote usage, 25-26

**character arrays, 108-109, 196-198**

**character constants**
- C language specifications, 428
  - *escape sequences, 428-429*
  - *wide character, 429*
- in expressions, 227-230

**character functions, 473**

**character I/O operations**
- getchar( ) function, 348
- putchar( ) function, 348

**character string constants**
- C language specifications
  - *concatenation, 429*
  - *multibyte, 429*
- pointers, 267-268

**character strings, 195**
- adjacent, 218
- combining with array of structures, 219-222
- comparing, 204-206
- concatenating, 196
- Concatenating Character Strings (Program 10.3), 202-203
- continuation of, 218-219
- Converting a String to its Integer Equivalent (Program 10.11), 228-230
- converting into integers, 228

copying, 266-267, 271
delimiting, 195
displaying, 201-203
escape characters, 216-218
initializing, 201-203
inputting, 206-208
length, 199, 272
null string, 213-215
pointers to, 266-267
Reading Strings with scanf (Program 10.5), 207-208
testing for equality, 204-206
Testing Strings for Equality (Program 10.4), 204-206
universal character name, 218
variable length, 198-200

**characters**
arrays of
*comparing, 472*
*copying, 472*
*initializing, 107*
*memory functions, 472*
*searching, 472*
files
*reading (getc( ) function), 365*
*reading (putc( ) function), 365*
formation of valid variables, 22
pointers to, 238-239
sign extensions, 329
single-character input, 208-212
whitespace, scanf( ) function, 355

**cimaq( ) function, 489**

**classes (OOP)**
instances, 412-413
methods, 412-413

**clear command (gdb debugger), 409**

**clearerr( ) function, 474**

**clearing end of file indicators, 474**

**cloq( ) function, 489**

**closing files, 474**
fclose( ) function, 365-367

**Code Warrior Web site, 503**

**comma operators, 378, 447**

**command lines, multiple source files, compiling, 334-336**

**command-line arguments, 380**
File Copy Program Using Command-Line Arguments (Program 17.1), 382-383
main( ) function, 380-381
storing, 383

**comments**
C language specifications, 426
character syntax
*beginning, 18*
*terminating, 18*
including in programs, 17-19
proper usage of, 18-19
Using Comments in a Program ((Program 3.6), 17-19

**communication between modules**
external variables, 336-338
include files, 341-342
prototype declarations, 336
static variables, 339-340

**compare_strings( ) function, 224**

**comparing**
arrays of characters, 472
character strings, 204-206
strings, 470-471

**compilers, 6-9**
GNU C, 12
Unix C, 12

**compiling programs, 7-12**
assemblers, 9
debugging phase, 9
errors
*semantic, 7-9*
*syntactic, 7-9*
multiple source files from command lines, 334-336

**Compiling the Debug Code (Program 18.2), 391-393**

**complex arithmetic functions, 488-490**

**compound literals, 450-451**

structural values, assigning, 181-182

**compound relationship tests**

if statement, 72-74

*Determining if a Year Is a Leap Year (Program 6.5), 73-74*

*logical AND operator, 72-73*

*logical OR operator, 72-73*

**compound statements, C language specifications, 456**

**concat( ) function, 196, 201-203**

**concatenating**

character string constants, C language specifications, 429

character strings, 196

strings, 470

**Concatenating Character Arrays (Program 10.1), 196-198**

**Concatenating Character Strings (Program 10.3), 202-203**

**conditional (?\) operator, ternary nature of, 91-92**

**conditional compilation, 316**

#else statement, 316-318

#endif statement, 316-318

#ifdef statement, 316-318

#ifndef statement, 316-318

**conditional expression operator, #define statement in macros, 310**

**conditional operators, 446**

**conj( ) function, 489**

**console windows, 9**

**const modifier, C language specifications, 439**

**const variable (arrays), 111-113**

**constant expressions, 23, 442-443**

**constant FILE pointers**

stderr, 369-370

stdin, 369-370

stdout, 369-370

**constant keyword (pointers), 253**

**constants (C language specifications)**

character constants, 428-429

*in expressions, 227-230*

character strings, 267-268, 429

enumeration constants, 430

floating-point constants, 427-428

integer constants, 427

wide character constants, 429

**continue statement, 62-63**

C language specifications, 457

**convert_number( ) function, 152**

**converting**

arguments, data types, 329-330

character strings into integers, 228-229

data types

*float to int, 36-38*

*in expressions, 327-329*

*int to float, 36-38*

*sign extension, 329*

strings to numbers, 479-481

**Converting a Positive Integer to Another Base**

Program 7.7, 110-111

Program 8.14, 153-156

**Converting a String to its Integer Equivalent (Program 10.11), 228-230**

**Converting Between Integers and Floats (Program 4.5), 36-38**

**copy_string( ) function, 266-267, 271**

**copying**

arrays of characters, 472

character strings, 266-267, 271

strings, 470-471

**Copying Characters from Standard Input to Standard Output (Program 16.2), 361-362**

**Copying Files (Program 16.3), 366-367**

**copysign( ) function, 484**

**cos( ) function, 484**

**cosh( ) function, 484**

**counters, array elements, 100-103**

**Counting the Characters in a String (Program 10.2), 199-200**

**Counting Words (Program 10.7), 210-212**

**countWords( ) function, 210-215**

**cpow( ) function, 489**

**cproj( ) function, 489**

**creal( ) function, 489**

**csin( ) function, 489**

**csinh( ) function, 489**

**csqrt( ) function, 489**

**ctan( ) function, 490**

**ctanh( ) function, 490**

**cvs utility, programming functionality, 344**

**CygWin Web site, 502**

## D

**data encapsulation (OOP), 417**

**data types**

- arguments, conversion of, 329-330
- arrays, declaring, 97-98
- C language specifications
  - *basic, 430-432*
  - *declarations, 430*
  - *derived, 432-438*
  - *enumerated, 438*
  - *modifiers, 439*
  - *typedef statement, 438-439*
- char, 23
  - *quote usage, 25-26*
- conversions
  - *order in evaluating expressions, 327-329*
  - *sign extension, 329*
- Converting Between Integers and Floats (Program 4.5), 36-38
- converting to float, 36-38
- converting to int, 36-38
- double, 23-25
- enumerated
  - *defining, 321-322, 324*
  - *Using Enumerated Data Types (14.1), 322-324*
- float, 23
  - *decimal notation, 24*
  - *hexadecimal notation, 25*
  - *scientific notation, 24-25*
- int, 23
  - *base notations, 23-24*
  - *machine-dependent ranges, 24*
  - *ranges, 24*
  - *storage sizes, 24*
  - *valid examples of, 23*
- naming (typedef statement), 325-327
- specifiers
  - *long, 28-30*
  - *long long, 28-30*
  - *short, 28-30*
  - *signed, 28-30*
  - *unsigned, 28-30*
- storage of differing types (unions), 375-378
- Using the Basic Data Types (Program 4.1), 26-27
- void, 128
  - *value storage, 26*

**debugging**

- gdb tool, 395-398
  - *backtrace command, 405*
  - *break command, 400, 409*

*breakpoint deletion, 404-405*
*clear command, 409*
*function calls, 405-406*
*help command, 406-408*
*info break command, 404-405*
*info source command, 409*
*inserting breakpoints, 400*
*list command, 399-400, 409*
*miscellaneous features, 408*
*next command, 409*
*print command, 409*
*program execution controls, 400*
*quit command, 409*
*run command, 400, 409*
*session example, 402-404*
*set var command, 398-399, 409*
*single stepping, 401-404*
*stacktrace retrieval, 405*
*step command, 401-404, 409*
*viewing source files, 399-400*
preprocessor, 389-395
programs
*A Simple Program for Use with gdb (18.4), 396-398*
*Adding Debug Statements with the Preprocessor (18.1), 389-391*
*Compiling the Debug Code (18.2), 391-393*
*Defining the DEBUG Macro (18.3), 393-395*
*Working with gdb (18.5), 401-402*

**decimal notation in float data types, 24**

**declaring**
arguments in functions, 134-135
arrays
*data types, 97-98*
*multidimensional, 114*
bit fields, 296
data types, C language specifications, 430
return types in functions, 126, 134-135
structures, 166
unions, 375
variables, 15
*in for loops, 55-56*

**decrement (--) operator, 50, 262, 268, 445**

**defined names**
NULL, 301
values, 300

**defined values, referencing (#define statement), 307-308**

**defining**
arrays
*of structures, 182*
*with unions, 376-377*
bit fields, 294-295
data types, enumerated, 321-324
external variables, 337
functions, 119-122
local variables in functions, 124-126
pointer variables, 235-239
structures, 166-168
*global structure definition, 173*
unions, members, 376
variables in arrays, 96-98

**Defining the DEBUG Macro (Program 18.3), 393-395**

**deleting files via remove( ) function, 371**

**delimiting character strings, 195**

**Demonstrating an Array of Counters (Program 7.2), 101-103**

**derived data types**
C language specifications, 432
*multidimensional arrays, 433-434*
*pointers, 437-438*
*single-dimensional arrays, 432-433*

*structures, 434-436*
*unions, 436-437*
*variable-length arrays, 433*
**Determining if a Number Is Even or Odd (Program 6.3), 69-71**
**Determining if a Year Is a Leap Year (Program 6.5), 73-74**
**Determining Tomorrow's Date (Program 9.2), 169-171**
**digraph characters, C language specifications, 425**
**directives, 299**
preprocessors, 461
*#define, 461-463*
*#error, 463*
*#if, 463*
*#ifdef, 464*
*#ifndef, 464*
*#include, 464-465*
*#line, 465*
*#pragma, 465*
*#undef, 465*
**dispatch tables, 274**
**display_converted_number( ) function, 152**
**Displaying Multiple Lines of Output (Program 3.3), 14**
**Displaying Multiple Variables (Program 3.5), 16-17**
**Displaying Variables (Program 3.4), 15-16**
**division sign (/), arithmetic operator, 30-33**
**do statement, 60-62**
C language specifications, 457
Implementing a Revised Program to Reverse the Digits of a Number (Program 5.9), 61-62
programming looping usage, 44
**double data type, 23-25**
**double quotation marks (")**
char data types, 25-26
character strings, declaring, 195
**double quote (\") escape character, 217**
**doubly linked lists (pointers), 244-252**
**dynamic memory allocation, 383-384**
arrays, 117
calloc( ) function, 384-386
free( ) function, 387-388
functions, 481
linked lists, 387-388
malloc( ) function, 384-386
returning memory to system, 387-388
sizeof operator, 385-387

## E

**editing programs with modular programming, 333-334**
**elements (array)**
counters, 100-103
initializing, 106-108
sequencing through, 96-100
summing, 262-264
values, storing, 96
**else if construct (if statement), 76-83**
Categorizing a Single Character Entered at the Terminal (Program 6.7), 78-80
Evaluating Simple Expressions (Program 6.8), 80-82
Implementing the Sign Function (Program 6.6), 77-78
Revising the Program to Evaluate Simple Expressions (Program 6.8A), 82-83
sign function, 76
**emacs text editor, 11**
**end-of-file conditions**
clearing, 474
I/O operations, 361-362
**enum keyword, 321-324**

**enumerated data types**
C language specifications, 438
defining, 321-324
Using Enumerated Data Types (14.1), 322-324
**enumeration constants, C language specifications, 430**
**EOF values, getchar( ) function, 362**
**equal to (= =) operator, 46-50**
**equal_strings( ) function, 204**
**errors in programming, troubleshooting, 497-500**
**escape characters, 216-218**
**escape sequences, C language specifications, 428-429**
**evaluating order of operators, 442**
**Evaluating Simple Expressions (Program 6.8), 80-82**
**.exe filename extension, 9**
**exf( ) function, 484**
**exfc( ) function, 484**
**exit( ) function, 490**
programs, terminating, 370-371
**exiting**
loops, 62
programs, 490
**exp( ) function, 484**
**expml( ) function, 484**
**expressions**
C language specifications, 439
*constant expressions, 442-443*
*summary table, 440-442*
character constants in, 227-230
constant, 23
data types, conversion order, 327-329
pointers, 239-240
structures, 168-171
**extensions (filenames), 9**
**external variables**
defining, 337
modules, communicating between, 336-338
versus static variables, 339-340

## F

**fabs( ) function, 484**
**factorial( ) function, 159**
**fclose( ) function, 474**
files, closing, 365-367
**fdim( ) function, 484**
**feof( ) function, 474**
testing files for EOF conditions, 367-368
**ferror( ) function, 474**
**fflush( ) function, 474**
**fgetc( ) function, 474**
**fgetpos( ) function, 474**
**fgets( ) function, 474**
files, reading to, 368
**Fibonacci numbers, generating, 103-104**
Generating Fibonacci Numbers Using Variable-Length Arrays (Program 7.8), 115-117
**field width specification, triangular number output, 51**
**fields, omitting in for loops, 55**
**File Copy Program Using Command-Line Arguments (Program 17.1), 382-383**
**file I/O operations**
fclose( ) function, 365-367
feof( ) function, 367-368
fgets( ) function, 368
fopen( ) function, 363-364
fprint( ) function, 368
fputs( ) function, 368
getc( ) function, 365

putc( ) function, 365
remove( ) function, 371
rename( ) function, 371

**FILE pointers**
stderr, 369-370
stdin, 369-370
stdout, 369-370

**filename extensions, 9**

**files**
a.out (Unix executable), 9
characters
*reading (getc( ) function), 365*
*reading (putc( ) function), 365*
closing (fclose( ) function), 365-367, 474
copying (Program 16.3), 366-367
current position, returning, 474
deleting (remove( ) function), 371
EOF conditions, testing (fclose( ) function), 367-368
executable (Unix), 9
header, 467
I/O operations
*end-of-file conditions, 361-362*
*redirection of, 359-361*
include, 341-342
modular programming organization, 333-334
naming, 7
opening, 475
opening (fopen( ) function), 363-364
printing (fprint( ) function), 368
programming utilities
*ar, 345*
*cvs, 344*
*grep, 345*
*make, 343-344*
*sed, 345*
reading to (fgets( ) function), 368
renaming (rename( ) function), 371, 478
temporary files, creating, 478
writing to (fputs( ) function), 368

**find_entry( ) function, 257**

**Finding the Greatest Common Divisor and Returning the Results (Program 8.6), 127-128**

**Finding the Minimum Value in an Array (Program 8.9), 138-140**

**float data type, 23**
converting to, 36-38
decimal notation, 24
hexadecimal notation, 25
scientific notation, 24-25

**float.h header file, 316, 469**

**floating point numbers, 15**

**floating-point constants, C language specifications, 427-428**

**floor( ) function, 484**

**fma( ) function, 485**

**fmax( ) function, 485**

**fmin( ) function, 485**

**fmod( ) function, 485**

**fopen( ) function, 363-364, 475**

**for statement**
array elements, sequencing through, 98-100
C language specifications, 457
init expression, 45
loop condition, 45
loop expression, 45
nested, 53-54
program looping
*relational operators, 46-50*
*triangular number calculation, 44-45*
programming looping usage, 44
variants, 54
*field omission, 55*
*multiple expressions, 55*
*variable declaration, 55-56*

**formal parameter name, function arguments, 124**

**format string, 17**

**formatted I/O operations**
printf( ) function, 348
*conversion characters, 350*
*flags, 348*
*Illustrating the printf Formats (Program 16.1), 350-355*
*type modifiers, 349*
*width and precision modifiers, 349*
scanf( ) function, 355
*conversion characters, 356-359*
*conversion modifiers, 355*

**FORTRAN (FORmula TRANslation) language, 6**

**fpclassify( ) function, 482**

**fprintf( ) function, 368, 475**

**fputc( ) function, 475**

**fputs( ) function, 368, 475**

**fractions program**
Working with Fractions in C (19.1), 413-414
Working with Fractions in C# (19.4), 422-424
Working with Fractions in C++ (19.3), 419-421
Working with Fractions in Objective-C (19.2), 414-419
writing in C, 413-414
writing in C#, 422-424
writing in C++, 419-421
writing in Objective-C, 414-419

**fread( ) function, 475**

**free( ) function, 387-388, 481**

**freopen( ) function, 475**

**frexp( ) function, 485**

**fscanf( ) function, 476**

**fseek( ) function, 476**

**fsetpos( ) function, 476**

**ftell( ) function, 476**

**function calls (gdb debugger), 405-406**

**functions, 119**
abs( ), 490
absolute value of numbers, calculating, 129-131
absolute_value( ), 129-131
acos( ), 483
alphabetic( ), 212, 214
arguments, 16, 122-123
*checking values, 135-137*
*declaring, 134-135*
*formal parameter name, 124*
*format string, 17*
*pointer arguments, passing, 254-257*
array_sum( ), 262-264
arrays
*passing multidimensional arrays, 146-152*
*passing to, 137-142*
asin( ), 483
asinh( ), 483
atan( ), 484
atan2( ), 484
atanh( ), 484
atof( ), 480
atoi( ), 230, 480
atol( ), 480
auto_static( ), 157-158
automatic local variables, 124-126, 156
C language specifications, 452
*calls, 455-456*
*definition of, 454-455*
*pointers, 456*
cabs( ), 488
cacos( ), 488
cacosh( ), 488
calling, 121-122
calloc( ), 386, 481
carq( ), 488
casin( ), 488
casinh( ), 488

catan( ), 489
catanh( ), 489
ccos( ), 489
ccosh( ), 489
ceil( ), 484
cexp( ), 489
character functions, 473
cimaq( ), 489
clearerr( ), 474
cloq( ), 489
compare_strings( ), 224
complex arithmetic functions, 488-490
concat( ), 196, 201-203
conj( ), 489
convert_number( ), 152
copy_string( ), 266-267, 271
copysign( ), 484
cos( ), 484
cosh( ), 484
count_words( ), 210, 214-215
cpow( ), 489
cproj( ), 489
creal( ), 489
csin( ), 489
csinh( ), 489
csqrt( ), 489
ctan( ), 490
ctanh( ), 490
declaring return value type, 126
defining, 119-122
    *global structure definition, 173*
display_converted_number( ), 152
dynamic memory allocation
    *calloc( ), 384-385*
    *functions, 481*
    *malloc( ), 384-385*
equal_strings( ), 204
exf( ), 484
exfc( ), 484
exit( ), 490
exp( ), 484
expml( ), 484
fabs( ), 484
factorial( ), 159
fclose( ), 474
fdim( ), 484
feof( ), 474
ferror( ), 474
fflush( ), 474
fgetc( ), 474
fgetpos( ), 474
fgets( ), 474
find_entry( ), 257
floor( ), 484
fma( ), 485
fmax( ), 485
fmin( ), 485
fmod( ), 485
fopen( ), 475
fpclassify( ), 482
fprintf( ), 475
fputc( ), 475
fputs( ), 475
fread( ), 475
free( ), 387-388, 481
freopen( ), 475
frexp( ), 485
fscanf( ), 476
fseek( ), 476
fsetpos( ), 476
ftell( ), 476
fwrite( ), 477
get_number_and_base( ), 152
getc( ), 477
getchar( ), 208-210, 477
getenv( ), 490
gets( ), 209-212, 477
global variables, 152-156
hypot( ), 485

I/O, 473-478
    *fclose( ), 365-367*
    *feof( ), 367-368*
    *fgets( ), 368*
    *fopen( ), 363-364*
    *fprint( ), 368*
    *fputs( ), 368*
    *getc( ), 365*
    *getchar( ), 348*
    *printf( ), 348-355*
    *putc( ), 365*
    *putchar( ), 348*
    *remove( ), 371*
    *rename( ), 371*
    *scanf( ), 355-359*
ilogb( ), 485
in-memory format conversion, 478-479
is_leap_year( ), 174
isalnum( ), 473
isalpha( ), 230, 473
iscntrl( ), 473
isdigit( ), 230, 473
isfin( ), 482
isgraph( ), 473
isgreater( ), 482
isgreaterequal( ), 482
isinf( ), 482
islessequal( ), 483
islessgreater( ), 483
islower( ), 230, 473
isnan( ), 483
isnormal( ), 483
isprint( ), 473
ispunct( ), 473
isspace( ), 473
isunordered( ), 483
isupper( ), 230, 473
isxdigit( ), 473
labs( ), 490
ldexp( ), 485
lgamma( ), 485
llabs( ), 490
llrint( ), 486
llround( ), 486
local variables, defining, 124-126
log( ), 485
log1( ), 486
log2( ), 485
logb( ), 485
loglb( ), 485
lookup( ), 219-223
lrint( ), 486
main( ), 120
    *program syntax, 13*
malloc( ), 386, 481
math functions, 482-487
memchr( ), 472
memcmp( ), 472
memcpy( ), 472
memmove( ), 472
memory functions, 472
minimum( ), 138
modf( ), 486
modules, 333
nan( ), 486
nearbyint( ), 486
Newton-Raphson Iteration Technique, 131-133
nextafter( ), 486
nexttoward( ), 486
number_of_days( ), 171-174
passing arrays to assignment operators, 142-143
perror( ), 477
pointers, returning, 257
pointers to, 273-274
pow( ), 486
print_message( ), 120
printf( ), 16, 477
    *program syntax, 13*

programs
*Calculating Factorials Recursively (8.16), 159-161*
*Calculating the Absolute Value (8.7), 129-131*
*Calculating the nth Triangular Number (8.4), 123*
*Calculating the Square Root of a Number (8.8), 132-133*
*Calling Functions (8.2), 121*
*Changing Array Elements in Functions (8.11), 142-143*
*Converting a Positive Integer to Another Base (8.14), 153-156*
*Finding the Greatest Common Divisor and Returning the Results (8.6), 127-128*
*Finding the Minimum Value in an Array (8.9), 138, 140*
*Illustrating Static and Automatic Variables (8.15), 157-158*
*More on Calling Functions (8.3), 122*
*Multidimensional Variable-Length Arrays (8.13A), 150-152*
*Revising the Function to Find the Minimum Value in an Array (8.10), 140-142*
*Revising the Program to Find the Greatest Common Divisor (8.5), 125-126*
*Sorting an Array of Integers into Ascending Order (8.12), 144-146*
*Updating the Time by One Second (9.5), 178-180*
*Using Multidimensional Arrays and Functions (8.13), 147-150*
*Writing a Function in C (8.1), 120-121*
prototype declaration, 124
putc( ), 477
putchar( ), 477
puts( ), 478
qsort( ), 274, 490-491
rand( ), 491
read_line( ), 213-215
realloc( ), 481
recursive, 159-161
remainder( ), 486
remove( ), 478
rename( ), 478
returning results from, 126-135
rewind( ), 478
rint( ), 487
rotate( ), 290-292
round( ), 487
scalar_multiply( ), 147
scalbln( ), 487
scalbn( ), 487
scanf( ), 206, 478
*input values, 51-52*
shift functions, 288-290
shift( ), 289
signbit( ), 483
sin( ), 487
sinh( ), 487
sort( ), 143-144, 146
sprintf( ), 478-479
sqrt( ), 487
square_root( ), 133
srand( ), 491
sscanf( ), 478-479
static functions, 339
static variables, 156
strcat( ), 230, 470
strchr( ), 470
strcmp( ), 230, 470
strcpy, 470-471
strcpy( ), 230
string functions, 470-472
string-to-number conversions, 479-481
string_length( ), 199, 272
string_to_integer( ), 228-230
strlen( ), 230, 471

strncat( ), 471
strncmp( ), 471
strncpy( ), 471
strrchr( ), 471
strstr( ), 471-472
strtod( ), 480
strtol( ), 480
strtoul( ), 481
structures, 171-174, 177
system( ), 491
tan( ), 487
tanh( ), 487
tgamma( ), 487
time_update( ), 178-180, 183
tmpfile( ), 478
tolower( ), 473
toupper( ), 473
trunc( ), 487
ungetc( ), 478
utility functions, 490-491

**fwrite( ) function, 477**

## G

**gcc compiler, command-line options, 493-495**

**gcc Web site, 493, 502**

**gdb tool, debugging with, 395-398**
backtrace command, 405
break command, 400, 409
breakpoint deletion, 404-405
clear command, 409
function calls, 405-406
help command, 406-408
info break command, 404-405
info source command, 409
inserting breakpoints, 400
list command, 399-400, 409
miscellaneous features, 408
next command, 409
print command, 409
program execution controls, 400
quit command, 409
run command, 400, 409
session example, 402-404
set var command, 398-399, 409
single stepping, 401-404
stacktrace retrieval, 405
step command, 401-404, 409
viewing source files, 399-400

**Generating a Table of Prime Numbers (Program 6.10), 87-90**

**Generating a Table of Triangular Numbers (Program 5.3), 47-50**

**Generating Fibonacci Numbers (Program 7.3), 103-104**

**Generating Fibonacci Numbers Using Variable-Length Arrays (Program 7.8), 115-117**

**get_number_and_base( ) function, 152**

**getc( ) function, 365, 477**

**getchar( ) function, 208-210, 348, 477**

**getenv( ) function, 490**

**gets( ) function, 209-212, 477**

**getter methods (OOP), 417**

**global variables**
default initial values, 155
functions, 152-156

**GNU C compiler, 12**

**GNU.org Web site, command-line tools, 504**

**Google Groups Web site, 502**

**goto statement**
C language specifications, 458
execution of, 373
labels, 373
programming abuse, 374

**greater than or equal to (>=) operator, 46-50**

**grep utility, programming functionality, 345**

## H

**header files**
- #include statement, 313-315
- float.h, 316, 469
- limits.h, 316, 468
- math.h, 482
- modular programming, use of, 341-342
- stdbool.h, 469
- stddef.h, 467
- stdint.h, 469-470
- stdlib.h, 490-491

**help command (gdb debugger), 406-408**

**hexadecimal character value (\xnn) escape character, 217**

**hexadecimal notation**
- float data types, 25
- int data type, 23-24

**high-order bit, 279**

**higher-level languages, 6**
- assembly languages, 6
- compilers, 6
- FORTRAN, 6
- interpreters, 10
- syntax standardization, 6

**horizontal tab (\t) escape character, 216**

**hypot( ) function, 485**

## I

**I/O functions, 473-478**

**I/O operations, 347**
- character functions
  - *getchar( ), 348*
  - *putchar( ), 348*
- Copying Characters from Standard Input to Standard Output (Program 16.2), 361-362
- file functions
  - *fclose( ), 365-367*
  - *feof( ), 367-368*
  - *fgets( ), 368*
  - *fopen( ), 363-364*
  - *fprint( ), 368*
  - *fputs( ), 368*
  - *getc( ), 365*
  - *putc( ), 365*
  - *remove( ), 371*
  - *rename( ), 371*
- files
  - *end-of-file conditions, 361-362*
  - *redirecting to, 359-361*
- formatted functions
  - *printf( ), 348-355*
  - *scanf( ), 355-359*
- function calls, 347

**identifiers, C language specifications, 425**
- keywords, 426
- predefined, 466
- universal character names, 426

**IDEs (Integrated Development Environments), 10, 334**
- function of, 10
- Linux, 10
- Mac OS X
  - *CodeWarrior, 10*
  - *Xcode, 10*
- Windows OS,Visual Studio, 10

**if statement, 65**
- C language specifications, 458
- compound relational tests, 72-74
- else if construct, 77-83
- general format, 65
- if-else construct, 69-72
- nested, 75-76

programs

*Calculating the Absolute Value of an Integer (6.1), 66-67*

*Calculating the Average of a Set of Grades (6.2), 67-69*

*Determining if a Number Is Even or Odd (6.3), 69-71*

*Revising the Program to Determine if a Number Is Even or Odd (6.4), 71-72*

**if-else construct (if statement), 69-72**

Determining if a Number Is Even or Odd (Program 6.3), 69-71

Revising the Program to Determine if a Number Is Even or Odd (Program 6.4), 71-72

**Illustrating a Structure (Program 9.1), 166-168**

**Illustrating Arrays of Structures (Program 9.6), 183-184**

**Illustrating Bitwise Operators (Program 12.2), 286-287**

**Illustrating Pointers (Program 11.1), 236-237**

**Illustrating Static and Automatic Variables (Program 8.15), 157-158**

**Illustrating Structures and Arrays (Program 9.7), 188-189**

**Illustrating the Modulus Operator (Program 4.4), 35-36**

**ilogb( ) function, 485**

**Implementing a Revised Program to Reverse the Digits of a Number (Program 5.9), 61-62**

**Implementing a Rotate Function (Program 12.4), 290-292**

**Implementing a Shift Function (Program 12.3), 288-290**

**Implementing the Sign Function (Program 6.6), 77-78**

**in-memory format conversion functions, 478-479**

**include files, modular programming, 341-342**

**include statement, program syntax, 13**

**increment (++) operator, 49, 262, 268, 445**

**index number (arrays), 96**

**indirection, 235-236**

**infinite loops, 65**

**info break command (gdb debugger), 404-405**

**info source command (gdb debugger), 409**

**init expressions (for statement), 45**

**initializing**

arrays

*characters, 107*

*elements, 106-108*

*multidimensional arrays, 114-115*

*of structures, 183*

character strings, 201-203

structures, 180-181

union variables, 376

variables (static), 156-158

**Initializing Arrays (Program 7.5), 107-108**

**input**

programs, 9

single-character, 208-212

**input/output operations.** *See* **I/O operations**

**inputting character strings, 206-208**

**instances, classes (OOP), 412-413**

**instruction sets, 5**

**int data type, 23**

base notations, 23-24

converting to, 36-38

machine-dependent ranges, 24

ranges, 24

storage sizes, 24

valid examples of, 23

**integers**
- base conversion via arrays, 109-111
- constants, C language specifications, 427
- pointers, 239-240

**Integrated Development Environments.** *See* **IDEs**

**International Standard Organization (ISO), 1**

**interpreters, 10**

**Introducing Character Arrays (Program 7.6), 108-109**

**Introducing the #define Statement (Program 13.1), 300-302**

*Introduction of Object-Oriented Programming, 3rd Edition*, **503**

**isalnum( ) function, 473**

**isalpha( ) function, 230, 473**

**iscntrl( ) function, 473**

**isdigit( ) function, 230, 473**

**isfin( ) function, 482**

**isgraph( ) function, 473**

**isgreater( ) function, 482**

**isgreaterequal( ) function, 482**

**isinf( ) function, 482**

**islessequal( ) function, 483**

**islessgreater( ) function, 483**

**islower( ) function, 230, 473**

**isnan( ) function, 483**

**isnormal( ) function, 483**

**ISO (International Standard Organization), 1**

**isprint( ) function, 473**

**ispunct( ) function, 473**

**isspace( ) function, 473**

**isunordered( ) function, 483**

**isupper( ) function, 230, 473**

**isxdigit( ) function, 473**

## J - K

**JAVA programming language, interpretive nature of, 10**

**joining tokens in macros (## operator), 313**

**keywords**
- auto, 124-126, 156
- C language specifications, 426
- enum, 321-324
- static, 156
- void, 128

**Kochan-Wood Web site, 502**
- book exercises and errata resources, 501

**Kylix (Linux IDE), 10**

## L

**labels in goto statements, 373**

**labs( ) function, 490**

**ldexp( ) function, 485**

**least significant bit, 279**

**left shift (<<) bitwise operator, 287**

**length of strings, 471**

**less than (<) operator, 46-50**

**less than or equal to (<=) operator, 46-50**

**lgamma( ) function, 485**

**limits.h header file, 316, 468**

**linked lists**
- dynamic memory allocation, 387-388
- pointers, 244-252

**linking programs, 9**

**Linux, Kylix IDE, 10**

**list command (gdb debugger), 399-400, 409**

**llabs( ) function, 490**

**llrint( ) function, 486**

**llround( ) function, 486**

**loading programs, 9**

**local variables**

automatic (functions), 124-126, 156

defining (functions), 124-126

**log( ) function, 485**

**log1( ) function, 486**

**log2( ) function, 485**

**logb( ) function, 485**

**logical AND (&&) operator, compound relationship tests, 72-73**

**logical negation (!) operator, Boolean variables, 90**

**logical operators, 444**

**logical OR (||) operator, compound relationship tests, 72-73**

**logical right shift, 288**

**loglb( ) function, 485**

**long long specifier (data types), 28-30**

**long specifier (data types), 28-30**

**lookup( ) function, 219-223**

**loop condition (for statement), 45**

**loop expressions (for statement), 45**

**loops**

Boolean variables

*Generating a Table of Prime Numbers (Program 6.10), 87-90*

*Revising the Program to Generate a Table of Prime Numbers (Program 6.10A), 90-91*

break statement, 62, 84

continue statement, 62-63

do statement, 60-62

for statement, 44-45

*field omission, 55*

*multiple expressions, 55*

*nested, 53-54*

*sequencing through array elements, 98-100*

*variable declaration, 55-56*

*variants, 54-56*

if statement, 65

*Calculating the Absolute Value of an Integer (Program 6.1), 66-67*

*Calculating the Average of a Set of Grades (Program 6.2), 67-69*

*if-else construct, 69-72*

infinite, 65

null statement, 374

relational operators, 46-50

switch statement, 84

*Revising the Program to Evaluate Simple Expressions, Version 2 (Program 6.9), 85-86*

while statement, 56-60

**low-order bit, 279**

**lrint( ) function, 486**

## M

**Mac OS X**

IDEs

*CodeWarrior, 10*

*Xcode, 10*

Objective-C language usage, 414

**macros**

# operator, 312

## operator, 313

#define statement, 308-311

*conditional expression operator, 310*

*converting character case, 311*

*defining number of arguments, 311*

*testing for lowercase characters, 310*

#include statement, header files, 313-315

tokens, joining (## operator), 313

**main( ) function, 120**
command-line arguments, 380-381
program syntax, 13
**make utility, programming functionality, 343-344**
**malloc( ) function, 386, 481**
dynamic memory allocation, 384-385
**math functions, 482-487**
**math.h header file, 482**
**members (unions)**
arithmetic rules, 376
defining, 376
**memchr( ) function, 472**
**memcmp( ) function, 472**
**memcpy( ) function, 472**
**memmove( ) function, 472**
**memory, dynamic memory allocation, 383-384**
calloc( ) function, 386
free( ) function, 387-388
functions, 481
linked lists, 387-388
malloc( ) function, 386
returning memory to system, 387-388
sizeof operator, 385-386
**memory addresses (pointers), 274-276**
**memory functions, 472**
**message expressions (OOP), 412-413**
**methods, classes (OOP), 412-413**
getters, 417
setters, 417
**Metrowerks Web site, 503**
**MinGW Web site, 502**
**minimum( ) function, 138**
**minus sign (-), arithmetic operator, 30-33**
**modf( ) function, 486**
**modifiers, C language specifications, 439**
**Modifying the Dictionary Lookup Using Binary Search (Program 10.10), 224-227**
**modular programming**
file organization, 333-334
header files, use of, 341-342
IDE (Integrated Development Environment), 334
multiple source files, compiling from command line, 334-336
**modules, 333**
communicating between
*include files, 341-342*
*static variables, 339-340*
compiling, 334-336
external variables, communicating between, 336-338
prototype declarations, communicating between, 336
**modulus (%) arithmetic operator, 35-36**
**More Examples with Arithmetic Operators (Program 4.3), 33-34**
**More on Calling Functions (Program 8.3), 122**
**More on Working with Defines (Program 13.2), 302-303**
**More Pointer Basics (Program 11.2), 238**
**most significant bit, 279**
**multidimensional arrays, 113-114, 433-434**
declaring, 114
initializing, 114-115
Multidimensional Variable-Length Arrays (Program 8.13A), 150-152
passing to functions, 146-152
variable-length, 150-152
**Multidimensional Variable-Length Arrays (Program 8.13A), 150-152**
**multiple expressions, use in for loops, 55**

**multiple source files, compiling from command line, 334-336**

**multiplication sign (*), arithmetic operator, 30-33**

## N

**naming**
- data types (typedef statement), 325-327
- files, 7
- program constants, #define statement, 299-303
- variables, 21
  - *reserved names, 22*
  - *rules, 22*

**nan( ) function, 486**

**nearbyint( ) function, 486**

**negative numbers, 279-280**

**nested for loops, 53-54**

**nested if statements, 75-76**

**newline character (\n), program syntax, 13**

**newsgroups, C programming resources, 502**

**Newton-Raphson Iteration Technique, 131-133**

**next command (gdb debugger), 409**

**NeXT Software, 414**

**nextafter( ) function, 486**

**nexttoward( ) function, 486**

**not equal to (!=) operator, 46-50**

**null character ('\0'), 199**

**null statement**
- C language specifications, 458
- example of, 374-375
- loop control, 374
- programming uses, 374

**null strings, 213-215**

**number_of_days( ) function, 171-174**

**numbers**
- absolute values, calculating, 129-131
- Fibonacci, generation of, 103-104
- negative, 279-280
- prime, generating with arrays, 104-106
- square roots, calculating, 131-133

## O

**object-oriented programming.** *See* **OOP**

**Objective-C language**
- as basis for Mac OS X, 414
- development history, 414
- fractions program, writing, 414-419

*Objective-C Programming Language*, **504**

**octal character value (\nnn) escape character, 217**

**octal notation, int data type, 23**

**omitting fields in for loops, 55**

**OOP (object-oriented programming), 411**
- C# language, development history, 422
- C++ language, development history, 419
- car analogy, 411-412
- classes
  - *instances, 412-413*
  - *methods, 412-413*
- data encapsulation, 417
- *Introduction of Object-Oriented Programming, 3rd Edition*, 503
- languages, 411
- message expressions, 412-413
- methods
  - *getters, 417*
  - *setters, 417*
- Objective-C language, Working with Fractions in Objective-C (Program 19.2), 414-419
- overview, 411-412
- versus procedural languages, 413

**openf( ) function modes**
append, 364
read, 364
update, 364
write, 364
**opening files**
binary files, 475
fopen( ) function, 363-364
**operating systems**
function of, 6
Unix
*development of, 6*
*spin-offs, 6*
Windows XP, 7
**operators**
#, macro definitions, 312
##, macro definitions, 313
assignment operators, 15, 142-143
*joining with arithmetic operators, 38-39*
bit operators, 280
*& (bitwise AND), 281-283*
*<< (left shift), 287*
*^ (bitwise Exclusive-OR), 284-285*
*| (bitwise Inclusive-OR), 283-284*
C language specifications
*arithmetic operators, 443-444*
*array operators, 447-448*
*array pointers, 449-450*
*assignment operators, 446*
*bitwise operators, 445*
*comma operators, 447*
*conditional operators, 446*
*decrement operators, 445*
*increment operators, 445*
*logical operators, 444*
*pointer operators, 448*
*relational operators, 444-445*
*sizeof operators, 447*
*structure operators, 448*
*structure pointers, 450*
*type cast operators, 446*
comma, 378
conditional expression in macros, 310
evaluation order, 442
pointer operators, 236
precedence rules, 441-442
relational operators, 46-50
sizeof
*arguments, 385*
*dynamic memory allocation, 385-387*
summary table, 440-442
type cast, 69
**output operations**
end-of-file conditions, 361-362
redirecting to files, 359-361

## P

**passing arrays to functions, 137-142**
assignment operators, 142-143
multidimensional arrays, 146-152
**perror( ) function, 477**
**plus sign (+) arithmetic operator, 30-33**
**pointer operators, 448**
**Pointer Version of copyString( ) function (Program 11.13), 266-267**
**pointers, 235**
& (address) operator, 236, 260
* (indirection) operator, 236
arrays, 259-260
*decrement (--) operator, 262, 268*
*increment (++) operator, 262*
*postdecrement operator, 269-271*
*postincrement operator, 269-271*
*predecrement operator, 269-271*
*preincrement operator, 269-271*
*program optimization, 263-264*
*sequencing through array elements, 261*

character string constants, 266-268
const keyword, 253
declarations, 437-438
defining, 235-239
expressions, 239-240
functions, 273-274
*C language specifications, 456*
*passing pointer arguments, 254-257*
*returning pointers, 257*
indirection, 235
integers, 240
memory addresses, 274-276
programs
*Illustrating Pointers (11.1), 236-237*
*More Pointer Basics (11.2), 238*
*Pointer Version of copyString( ) function (11.13), 266-267*
*Returning a Pointer from a Function (11.10), 257-259*
*Revised Version of copyString( ) function (11.14), 271-272*
*Summing the Elements of an Array (11.12), 264-265*
*Traversing a Linked List (11.7), 250-252*
*Using Linked Lists (11.6), 246-250*
*Using Pointers and Functions (11.8), 254-255*
*Using Pointers in Expressions (11.3), 239-240*
*Using Pointers to Exchange Values (11.9), 255-257*
*Using Pointers to Find Length of a String (11.15), 272-273*
*Using Pointers to Structures (11.4), 241-243*
*Using Structures Containing Pointers (11.5), 243-244*
*Working with Pointers to Arrays (11.11), 262-263*
structures, 240-243
*linked lists, 244-252*
*structures containing pointers, 243-244*
subtracting, 272
versus arrays, differentiating, 264-265
**postdecrement operators, 269-271**
**postincrement operators, 269-271**
**pow( ) function, 486**
**precedence rules**
arithmetic operators, 30
operators, 441-442
rules example, 34
**precision modifiers, 69**
**predecrement operators, 269-271**
**predefined identifiers (directives), 466**
**preincrement operators, 269-271**
**preprocessor**
Adding Debug Statements with the Preprocessor (Program 18.1), 389-391
C language specifications, 460
*directives, 461-465*
*trigraph sequences, 460-461*
debugging with, 389-395
**preprocessor statements, 299**
#define, 299-303
*arguments, 308-311*
*definition types, 306-308*
*macros, 308-311*
*program extendability, 303-305*
*program portability, 305-306*
#elif, 318-319
#if, 318-319
#include, macro definition collections, 313-315
#undef, 319
**prime numbers**
Generating a Table of Prime Numbers (Program 6.10), 87-90
generating via arrays, 104-106

Revising the Program to Generate a Table of Prime Numbers (Program 6.10A), 90-91
Sieve of Erastosthenes algorithm, 118

**print_message( ) function, 120**

**print command (gdb debugger), 409**

**printf routine**
output, 14
variables
*displaying multiple values, 16-17*
*displaying values, 15-16*

**printf( ) function, 16, 348, 477**
conversion characters, 350
flags, 348
Illustrating the printf Formats (Program 16.1), 350-355
program syntax, 13
type modifiers, 349
width and precision modifiers, 349

**printing files via fprint( ) function, 368**

**procedural languages versus OOP languages, 413**

**program constants, symbolic names, 299-303**

**program looping**
break statement, 62
Calculating the Eighth Triangular Number (Program 5.1), 43
continue statement, 62-63
do statement, 44, 60-62
for statement, 44
*Generating a Table of Triangular Numbers (5.3), 47-50*
relational operators, 46-50
scanf( ) function, Asking the User for Input (Program 5.4), 51-52
triangular number calculation, 43-45
*nested for loops, 53-54*
*output alignment, 50-51*
*user input, 51-52*
while statement, 44, 56-60

**programming**
algorithms, 5
assembly languages, 6
case sensitivity, 11
common mistakes, troubleshooting, 497-500
higher-level languages, 6
instruction sets, 5
modular programming, 333-334
overview, 5
top-down, 137

***Programming in Objective-C*, 503**

**programming utilities**
ar, 345
cvs, 344
grep, 345
make, 343-344
sed, 345

**programs**
#define statement
*Introducing the #define Statement (13.1), 300-302*
*More on Working with Defines (13.2), 302-303*
A Simple Program for Use with gdb (18.4), 396-398
Adding Debug Statements with the Preprocessor (18.1), 389-391
arguments, calling, 13
arrays
*Converting a Positive Integer to Another Base (7.7), 110-111*
*Generating Fibonacci Numbers Using Variable -Length Arrays (7.8), 115-117*
*Introducing Character Arrays (7.6), 108-109*
*Multidimensional Variable-Length Arrays (8.13A), 150-152*
Asking the User for Input (5.4), 51-52
assemblers, 6, 9

bitwise operators, Illustrating Bitwise Operators (12.2), 286-287
bugs, 9
Calculating the Eighth Triangular Number (5.1), 43
Calculating the 200th Triangular Number (5.2), 44-45
Categorizing a Single Character Entered at the Terminal (6.7), 78-80
comment statements, including, 17-19
compiling, 7-12
  *debugging phase, 9*
  *semantic errors, 7-9*
  *syntactic errors, 7-9*
Compiling the Debug Code (18.2), 391-393
compound relational tests, Determining if a Year Is a Leap Year (Program 6.5), 73-74
Concatenating Character Arrays (10.1), 196-198
Concatenating Character Strings (10.3), 202-203
Converting a String to its Integer Equivalent (10.11), 228-230
Converting Between Integers and Floats (4.5), 36-38
Copying Files (16.3), 366-367
Counting the Characters in a String (Program 10.2), 199-200
Counting Words (Program 10.7), 210-212
Counting Words in a Piece of Text (10.8), 214-215
debugging, 9
Defining the DEBUG Macro (18.3), 393-395
Demonstrating an Array of Counters (7.2), 101-103
Determining if a Number Is Even or Odd (6.3), 69-71
Displaying Multiple Lines of Output (3.3), 14
Displaying Multiple Variables (3.5), 16-17
Displaying Variables (3.4), 15-16
editing (modular programming), 333-334
Evaluating Simple Expressions (6.8), 80-82
exiting, 490
File Copy Program Using Command-Line Arguments (17.1), 382-383
Finding the Greatest Common Divisor (5.7), 58-59
fractions
  *writing in C, 413-414*
  *writing in C#, 422-424*
  *writing in C++, 419-421*
  *writing in Objective-C, 414-419*
functions
  *Calculating Factorials Recursively (8.16), 159-161*
  *Calculating the Absolute Value (8.7), 129-131*
  *Calculating the nth Triangular Number (8.4), 123*
  *Calculating the Square Root of a Number (8.8), 132-133*
  *Calling Functions (8.2), 121*
  *Changing Array Elements in Functions (8.11), 142-143*
  *Converting a Positive Integer to Another Base (8.14), 153-156*
  *defining, 119-122*
  *Finding the Greatest Common Divisor and Returning the Results (8.6), 127-128*
  *Finding the Minimum Value in an Array (8.9), 138-140*
  *Illustrating Static and Automatic Variables (8.15), 157-158*
  *More on Calling Functions (8.3), 122*
  *Revising the Function to Find the Minimum Value in an Array (8.10), 140-142*

*Revising the Program to Find the Greatest Common Divisor (8.5), 125-126*

*Sorting an Array of Integers into Ascending Order (8.12), 144-146*

*Updating the Time by One Second (9.5), 178-180*

*Using Multidimensional Arrays and Functions (8.13), 147-150*

*Writing in Function in C (8.1), 120-121*

Generating a Table of Prime Numbers (6.10), 87-90

Generating a Table of Triangular Numbers (5.3), 47-50

Generating Fibonacci Numbers (7.3), 103-104

I/O operations, Copying Characters from Standard Input to Standard Output (16.2), 361-362

Illustrating Pointers (11.1), 236-237

Illustrating the Modulus Operator (4.4), 35-36

Illustrating the printf Formats (16.1), 350-355

Implementing a Revised Program to Reverse the Digits of a Number (5.9), 61-62

Implementing the Sign Function (6.6), 77-78

Initializing Arrays (7.5), 107-108

input, 9

interpreting, 10

Introducing the while Statement (5.6), 56-58

linking, 9

loading, 9

Modifying the Dictionary Lookup Using Binary Search (10.10), 224-227

More Examples with Arithmetic Operators (4.3), 33-34

More Pointer Basics (11.2), 238

output, 9

Pointer Version of copyString( ) function (11.13), 266-267

portability of, 6

proper termination of, 383

Reading Lines of Input (10.6), 209-210

Reading Strings with scanf (10.5), 207-208

Returning a Pointer from a Function (11.10), 257-259

Reversing the Digits of a Number (5.8), 59-60

Revised Version of copyString( ) function (11.14), 271-272

Revising the Program to Determine if a Number Is Even or Odd (6.4), 71-72

Revising the Program to Evaluate Simple Expressions (6.8A), 82-83

Revising the Program to Evaluate Simple Expressions, Version 2 (6.9), 85-86

Revising the Program to Generate a Table of Prime Numbers (6.10A), 90-91

Revising the Program to Generate Prime Numbers, Version 2 (7.4), 105-106

rotating bit values, Implementing a Rotate Function (12.4), 290-292

running, 12

shift functions, Implementing a Shift Function (12.3), 288-290

statements, calling, 13

structures

*Determining Tomorrow's Date (9.2), 169-171*

*Illustrating a Structure (9.1), 166-168*

*Illustrating Arrays of Structures (9.6), 183-184*

*Illustrating Structures and Arrays (9.7), 188-189*

*Revising the Program to Determine Tomorrow's Date (9.3), 171-174*

*Revising the Program to Determine Tomorrow's Date, Version 2 (9.4), 174-177*

Summing the Elements of an Array (11.12), 264-265

syntax
  *braces ({}), 13*
  *include statement, 13*
  *main( ) function, 13*
  *newline character (\n), 13*
  *printf( ) function, 13*
terminating (exit( ) function), 370-371
Testing Strings for Equality (10.4), 204-206
Traversing a Linked List (11.7), 250-252
undefined exit status, 383
Using Comments in a Program (3.6), 17-19
Using Enumerated Data Types (14.1), 322-324
Using Linked Lists (11.6), 246-250
Using Nested for Loops (5.5), 53-54
Using Pointers and Functions (11.8), 254-255
Using Pointers in Expressions (11.3), 239-240
Using Pointers to Exchange Values (11.9), 255-257
Using Pointers to Find Length of a String (11.15), 272-273
Using Pointers to Structures (11.4), 241-243
Using Structures Containing Pointers (11.5), 243-244
Using the #include Statement (13.3), 314-315
Using the Arithmetic Operators (4.2), 30-31
Using the Basic Data Types (4.1), 26-27
Using the Dictionary Lookup Program (10.9), 220-222
Working with an Array (7.1), 98, 100
Working with Fractions in C (19.1), 413-414
Working with Fractions in C# (19.4), 422-424
Working with Fractions in C++ (19.3), 419-421
Working with Fractions in Objective-C (19.2), 414-419
Working with gdb (18.5), 401-402
Working with Pointers to Arrays (11.11), 262-263
Writing Your First C Program (3.1), 11

**prototype declarations**
  functions, 124
  modules, communicating between, 336
**putc( ) function, 365, 477**
**putchar( ) function, 348, 477**
**puts( ) function, 478**
**Python programming language, 10**

## Q - R

**qsort( ) function, 274, 490-491**
**qualifiers (variables)**
  register, 378-379
  restrict, 379
  volatile, 379
**question mark (\?) escape character, 217**
**quit command (gdb debugger), 409**
**quotation marks, declaring character strings, 195**

**rand( ) function, 491**
**read_line( ) function, 213-215**
**reading files via fgets( ) function, 368**
**Reading Lines of Data (Program 10.6), 209-210**
**Reading Strings with scanf (Program 10.5), 207-208**
**real numbers, 15**
**realloc( ) function, 481**
**recursive functions, 159-161**
**redirecting I/O operations to files, 359-361**
**referencing defined values (#define statement), 307-308**

**register qualifier (variables), 378–379**
**relational operators, 46–50, 444–445**
**remainder( ) function, 486**
**remove( ) function, 478**
  files, deleting, 371
**rename( ) function, 478**
  files, renaming, 371
**renaming files, 478**
  rename( ) function, 371
**reserved names (variables), 22**
**restrict modifier, C language specifications, 439**
**restrict qualifier (variables), 379**
**return statement (functions), 126**
  C language specifications, 459
**returning**
  function results, 126–131
    *declaring return types, 134-135*
  pointers, 257
**Returning a Pointer from a Function (Program 11.10), 257–259**
**Revised Version of copyString( ) function (Program 11.14), 271–272**
**Revising the Function to Find the Minimum Value in an Array (Program 8.10), 140–142**
**Revising the Program to Determine if a Number Is Even or Odd (Program 6.4), 71–72**
**Revising the Program to Determine Tomorrow's Date (9.3), 171–174**
**Revising the Program to Determine Tomorrow's Date, Version 2 (9.4), 174–177**
**Revising the Program to Evaluate Simple Expressions (Program 6.8A), 82–83**
**Revising the Program to Evaluate Simple Expressions, Version 2 (Program 6.9), 85–86**
**Revising the Program to Find the Greatest Common Divisor (Program 8.5), 125–126**
**Revising the Program to Generate a Table of Prime Numbers (Program 6.10A), 90–91**
**Revising the Program to Generate Prime Numbers, Version 2 (Program 7.4), 105–106**
**rewind( ) function, 478**
**right justification, triangular number output, 51**
**right shift ( )  bitwise operator, 287–288**
**rint( ) function, 487**
**Ritchie, Dennis, 1**
**rotate( ) function, 290–292**
**rotating bit values, 290–292**
**round( ) function, 487**
**routines.** *See also* **functions**
  printf, 14
    *displaying multiple variable values, 16-17*
    *displaying variable values, 15-16*
    *output, 14*
**run command (gdb debugger), 400, 409**

## S

**scalar_multiply( ) function, 147**
**scalbln( ) function, 487**
**scalbn( ) function, 487**
**scanf( ) function, 206, 355, 478**
  %s format characters, 206
  conversion characters, 356–359
  conversion modifiers, 355
  input values, 51–52
  skipping fields, 358
**scientific notation, float data types, 24–25**
**scopes, 452**

**search methods**
binary search algorithm, 223-227
lookup( ) function, 222-223
**searches**
arrays of characters, 472
strings, 470-471
**sed utility, programming functionality, 345**
**semantic errors in programs, compiling, 7-9**
**semicolon (;)**
#define statement, 306-307
program syntax, 13
**set var command (gdb debugger), 398-399, 409**
**setters, methods (OOP), 417**
**shell programming language, 10**
**shift functions, 288-290**
programs, Implementing a Shift Function (12.3), 288-290
**shift( ) function, 289**
**short specifier (data types), 28-30**
**Sieve of Erastosthenes algorithm, prime number generation, 118**
**sign bit, 279-280**
**sign extension, data type conversions, 329**
**sign function**
else if construct (if statement), 76-83
*Categorizing a Single Character Entered at the Terminal (Program 6.7), 78-80*
*Implementing the Sign Function (Program 6.6), 77-78*
**signbit( ) function, 483**
**sin( ) function, 487**
**single quotation marks (')**
char data types, 25-26
character strings, declaring, 195
**single quote (\') escape character, 217**
**single-character input, 208-213**
Counting Words (Program 10.7), 210-212
Reading Lines of Data (Program 10.6), 209-210
**single-dimensional arrays, 432-433**
**sinh( ) function, 487**
**sizeof operators, 447**
arguments, 385
dynamic memory allocation, 385-387
**sort( ) function, 143-144**
**sorting arrays, 143-146, 490-491**
**Sorting an Array of Integers into Ascending Order (Program 8.12), 144-146**
**source programs, 7**
**specifiers (data types)**
long, 28-30
long long, 28-30
short, 28-30
unsigned, 28-30
**sprintf( ) function, 478-479**
**sqrt( ) function, 487**
**square roots, calculating, 131-133**
**square_root( ) function, 133**
**srand( ) function, 491**
**sscanf( ) function, 478-479**
*The Standard C Library*, **501**
**statements**
#define, 299-303
*arguments, 308-311*
*definition types, 306-308*
*macros, 308-311*
*program extendability, 303-305*
*program portability, 305-306*
#elif, 318-319
#if, 318-319
#include, macro definition collections, 313-315
#undef, 319

break, 62, 84

C language specifications, 456

*break, 456*

*compound, 456*

*continue, 457*

*do, 457*

*for, 457*

*goto, 458*

*if, 458*

*null, 458*

*return, 459*

*switch, 459-460*

*while, 460*

calling, 13

conditional compilation

*#else, 316-318*

*#endif, 316-318*

*#ifdef, 316-318*

*#ifndef, 316-318*

continue, 62-63

do, 60-62

for, 44-45

*nested, 53-54*

FORTRAN statements, 6

goto

*execution of, 373*

*programming abuse, 374*

if, 65

*Calculating the Absolute Value of an Integer (Program 6.1), 66-67*

*Calculating the Average of a Set of Grades (Program 6.2), 67-69*

*compound relational tests, 72-74*

*else if construct, 76-83*

*general format, 65*

*if-else construct, 69-72*

*nested, 75-76*

include, program syntax, 13

null

*example of, 374-375*

*programming uses, 374*

return (functions), 126

switch, 84

*Revising the Program to Evaluate Simple Expressions, Version 2 (Program 6.9), 85-86*

terminating, 14

typedef, data types, naming, 325-327

while, 56-60

**static functions, 339**

**static keyword, 156**

**static variables, 156**

initializing, 156-158

versus external variables, 339-340

**stdbool.h header file, 469**

**stddef.h header file, 467**

**stderr FILE pointer, 369-370**

**stdin FILE pointer, 369-370**

**stdint.h header file, 469-470**

**stdlib.h header file, 490-491**

**stdout FILE pointer, 369-370**

**step command (gdb debugger), 401-404, 409**

**storage classes**

functions, 452

variables, 452-454

**storing**

different data types (unions), 375-378

time in programs, 177-180

values in arrays, 96

variables via dynamic memory allocation, 383-384

**strcat( ) function, 230, 470**

**strchr( ) function, 470**

**strcmp( ) function, 230, 470**

**strcpy( ) function, 230, 470-471**

**string functions, 470-472**

**string_length( ) function, 199, 272**

**string_to_integer( ) function, 228-230**

**strings**

character strings, 195

*adjacent, 218*

*combining with array of structures, 219-222*

*comparing, 204-206*

*concatenating, 196*

*continuation of, 218-219*

*converting into integers, 228-229*

*copying, 266-267, 271*

*delimiting, 195*

*displaying, 201-203*

*escape characters, 216-218*

*initializing, 201-203*

*inputting, 206-208*

*length, 199, 272*

*pointers to, 266-267*

*testing for equality, 204-206*

*variable-length, 198-200*

comparing, 470-471

concatenating, 470

converting to numbers, 479-481

copying, 470-471

length, 471

null, 213-215

searches, 470-471

searching, 471

**strlen( ) function, 230, 471**

**strncat( ) function, 471**

**strncmp( ) function, 471**

**strncpy( ) function, 471**

**strrchr( ) function, 471**

**strstr( ) function, 471-472**

**strtod( ) function, 480**

**strtol( ) function, 480**

**strtoul( ) function, 481**

**structure operators, 448**

**structure pointers, 241, 450**

**structures**

arrays of, 182

*combining with character strings, 219-222*

*defining, 182*

*initializing, 183*

compound literal values, assigning, 181-182

containing arrays, 187-189

containing other structures, 185-187

containing pointers, 243-244

declarations, 166, 434-436

defining, 166-168

expressions, 168-171

function of, 165

functions, 171-174, 177

initializing, 180-181

pointers to, 240-243

*linked lists, 244-252*

programs

*Determining Tomorrow's Date (9.2), 169-171*

*Illustrating a Structure (9.1), 166-168*

*Illustrating Arrays of Structures (9.6), 183-184*

*Illustrating Structures and Arrays (9.7), 188-189*

*Revising the Program to Determine Tomorrow's Date (9.3), 171-174*

*Revising the Program to Determine Tomorrow's Date, Version 2 (9.4), 174-177*

time, updating, 177-180, 183

uses, 165

variants, 190-191

**subscripts (arrays), 96**

**subtracting pointers, 272**

**summing array elements, 262-264**

**Summing the Elements of an Array (Program 11.12), 264-265**

**switch statement, 84**

C language specifications, 459-460

programs, Revising the Program to Evaluate Simple Expressions,Version 2 (6.9), 85-86

**symbolic names, program constants, 299-303**

**syntactic errors, programs, compiling, 7-9**

**system include files**

float.h file, 316

limits.h file, 316

**system( ) function, 491**

## T

**tan( ) function, 487**

**tanh( ) function, 487**

**temporary files, creating, 478**

**terminating**

comments, character syntax, 18

programs

*exit( ) function, 370-371*

*proper methods, 383*

statements, 14

**ternary operator, 91-92**

**testing**

character strings for equality, 204-206

files for EOF conditions, 367-368

**Testing Strings for Equality (Program 10.4), 204-206**

**text editors**

C programming, 7

emacs, 11

vi, 7, 11

**tgamma( ) function, 487**

**time, updating, 177-180**

with array of structures, 183

**time_update( ) function, 178-180, 183**

**times equal (*=) operator, 143**

**tmpfile( ) function, 478**

**tokens, joining (## operator), 313**

**tolower( ) function, 473**

**top-down programming, 137**

**toupper( ) function, 473**

**Traversing a Linked List (Program 11.7), 250-252**

**trees, pointers, 244-252**

**triangular numbers, calculating, 43-45**

nested for loops (program looping), 53-54

output alignment (program looping), 50-51

user input (program looping), 51-52

**trigraph sequences, preprocessors, 460-461**

**troubleshooting programming errors, common mistakes, 497-500**

**trunc( ) function, 487**

**truth tables**

& (bitwise AND) operator, 281

^ (bitwise Exclusive-OR) operator, 284

| (bitwise Inclusive-OR) operator, 283

**twos complement notation, 279-280**

**type cast operators, 69, 446**

precedence rules, 38

**typedef statement**

C language specifications, 438-439

data types, naming, 325-327

## U

**unary minus arithmetic operator, 33-34**

**undefined exit statuses, 383**

**ungetc( ) function, 478**

**unions**
  arrays, defining, 376-377
  data types, storage, 375-378
  declarations, 436-437
  declaring, 375
  members
    *arithmetic rules, 376*
    *defining, 376*
  variables, initializing, 376
**units, bit fields, 296**
**universal character names, 218**
  C language specifications, 426
**Unix operating system**
  commands, 7
  compiler, 12
  development of, 6
  naming files, 7
  programming utilities
    *ar, 345*
    *grep, 345*
    *sed, 345*
  programs, linking, 9
  roots in C programming language, 1
  spin-offs, 6
**unsigned specifier (data types), 28-30**
**updating time in programs, 177-180**
**Updating the Time by One Second (Program 9.5), 178-180**
**Using Comments in a Program (Program 3.6), 17-19**
**Using Enumerated Data Types (Program 14.1), 322-324**
**Using Linked Lists (Program 11.6), 246-250**
**Using Multidimensional Arrays and Functions (Program 8.13), 147-150**
**Using Nested for Loops (Program 5.5), 53-54**
**Using Pointers and Functions (Program 11.8), 254-255**
**Using Pointers in Expressions (Program 11.3), 239-240**
**Using Pointers to Exchange Values (Program 11.9), 255-257**
**Using Pointers to Find Length of a String (Program 11.15), 272-273**
**Using Pointers to Structures (Program 11.4), 241-243**
**Using Structures Containing Pointers (Program 11.5), 243-244**
**Using the #include Statement (Program 13.3), 314-315**
**Using the Arithmetic Operators (Program 4.2), 30-31**
**Using the Basic Data Types (Program 4.1), 26-27**
**Using the Dictionary Lookup Program (Program 10.9), 220-222**
**usual arithmetic conversion, basic data types, 451-452**
**utilities (programming)**
  a, 345
  cv, 344
  gre, 345
  mak, 343-344
  se, 345
**utility functions, 490-491**

## V

**values**
  arrays, storing, 96
  defined
    *names, 300*
    *referencing (#define statement), 307-308*
**variable-length arrays, 433**
  Generating Fibonacci Numbers Using Variable-Length Arrays (Program 7.8), 115-117
  multidimensional, 150-152
**variable-length character strings, 198-200**

**variables**

arrays, defining, 96–98

Boolean

*Generating a Table of Prime Numbers (Program 6.10), 87-90*

*Revising the Program to Generate a Table of Prime Numbers (Program 6.10A), 90-91*

C language specifications, 452–454

const (arrays), 111–113

data storage types, 21

declarations, 15

*in for loops, 55-56*

external, 336–338

*defining, 337*

*versus static, 339-340*

global (functions), 152–156

initializing static variables, 156–158

local

*automatic (functions), 124-126, 156*

*defining (functions), 124-126*

names, 21

*reserved names, 22*

*rules, 22*

pointers, defining, 235–239

qualifiers

*register, 378-379*

*restrict, 379*

*volatile, 379*

static, 156

*initializing, 156-158*

*versus external, 339-340*

storing via dynamic memory allocation, 383–384

union, initializing, 376

valid characters, 22

**variants, structures of, 190–191**

**vi text editor, 7, 11**

**Visual Studio**

Web site, 503

Windows IDE, 10

**void data type, 128**

**void keyword, 128**

**volatile modifiers, C language specifications, 439**

**volatile qualifiers, 379**

## W - Z

**Web sites**

C language resources

*ANSI.org, 502*

*Code Warrior, 503*

*CygWin, 502*

*gcc compiler, 502*

*Kochan-Wood.com, 502*

*Kylix, 503*

*Metrowerks, 503*

*MinGW, 502*

*newsgroups, 502*

*Visual Studio, 503*

gcc, 493

GNU.org, 504

Google Groups, 502

Kochan-Wood, book exercises and errata, 501

OOP book resources

C# Programming in the Key of C#*, 503*

C++ Primer Plus*, 503*

Code Warrior*, 503*

Programming in Objective-C*, 503-504*

**while statement, 56–60**

C language specifications, 460

Finding the Greatest Common Divisor (Program 5.7), 58–59

Introducing the while Statement (Program 5.6), 56-58
programming looping usage, 44
Reversing the Digits of a Number (Program 5.8), 59-60

**whitespace characters, scanf( ) function, 355**

**wide character constants, C language specifications, 429**

**Working with an Array (Program 7.1), 98, 100**

**Working with Fractions in C (Program 19.1), 413-414**

**Working with Fractions in C# (Program 19.4), 422-424**

**Working with Fractions in C++ (Program 19.3), 419-421**

**Working with Fractions in Objective-C (Program 19.2), 414-419**

**Working with gdb (Program 18.5), 401-402**

**Working with Pointers to Arrays (Program 11.11), 262-263**

**writing**
files with fputs( ) function, 368
programs
*for handling fractions (C language), 413-414*
*for handling fractions (C# language), 422-424*
*for handling fractions (C++ language), 419-421*
*for handling fractions (Objective-C language), 414-419*

**Writing in Function in C (Program 8.1), 120-121**

**Writing Your First C Program (Program 3.1), 11**

**X3J11 committee (ANSI C), 1**

**Xcode, Mac OS X IDE, 10**
*XOR operator, 284-285*