Chapter 1

1. a. true; b. true; c. false; d. true; e. true; f. true; g. false; h. true; i. true; j. false; k. true; l. true; m. false; n. false

3. All programs must be loaded into main memory before they can be executed. Similarly, all data must be brought into main memory before a program can manipulate it.

5. The operating system monitors the overall activity of the computer and provides services.

7. In machine language the programs are written using the binary codes while in high-level language the program are closer to the natural language. For execution, a high-level language program is translated into the machine language while a machine language need not be translated into any other language.

9. To load an executable program into main memory.

11. A compiler reports syntax errors.

13. a. Analyze the problem, outline the problem and its solution requirements, and design an algorithm to solve the problem.

b. Implement the algorithm in a programming language, such as C++, and verify that the algorithm works.

c. Maintain the program by using and modifying it if the problem domain changes.

15. To find the weighted average of the four test scores, first you need to know each test score and its weight. Next, you multiply each test score with its weight, and then add these numbers to get the average. Therefore,

1. Get testScore1, weightTestScore1
2. Get testScore2, weightTestScore2
3. Get testScore3, weightTestScore3
4. Get testScore4, weightTestScore4
5. weightedAverage = testScore1 \* weightTestScore1 +

testScore2 \* weightTestScore2 +

testScore3 \* weightTestScore3 +

testScore4 \* weightTestScore4;

17. To find the price per square inch, first we need to find the area of the pizza. Then we divide the price of the pizza by the area of the pizza. Let radius denote the radius and area denote the area of the circle, and price denote the price of pizza. Also, let pricePerSquareInch denote the price per square inch.

a. Get radius

b. area = π \* radius \* radius

c. Get price

d. pricePerSquareInch = price / area

19. Suppose that radius denotes radius of the sphere, volume denotes volume of the sphere, and surfaceArea denotes the surface area of the sphere. The following algorithm computes the volume and surface area of the sphere.

|  |  |
| --- | --- |
| **Algorithm** | **C++ Instruction (Code)** |
| 1 1. Get the radius. | cin >> radius; |
| 2 2. Calculate the volume. | volume = (4.0 / 3.0) \* 3.1416 \* radius \* radius \* radius; |
| 3 3. Calculate the surface area. | surfaceArea = 4.0 \* 3.1416 \* radius \* radius; |

21. Suppose that billingAmount denotes the total billing amount, numOfItemsOrdered denotes the number of items ordered, shippingAndHandlingFee denotes the shipping and handling fee, and price denotes the price of an item. The following algorithm computes and outputs the billing amount.

a. Enter the number of items bought.

b. Get numOfItemsOrdered

c. billingAmount = 0.0;

d. shippingAndHandlingFee = 0.0;

e. Repeat the following for each item bought.

i. Enter the price of the item

ii. Get price

iii. billingAmount = billingAmount + price;

f. if billingAmount < 200

shippingAndHandlingFee = 10 \* numOfItemsOrdered;

g. billingAmount = billingAmount + shippingAndHandlingFee

i. Print billingAmount

23. Suppose x1 and x2 are the real root of the quadratic equation.

a. Get a

b. Get b

c. Get c

d. if (b \* b – 4 \* a \* c < 0)

Print "The equation has no real roots."

Otherwise

{

temp = b \* b – 4 \* a \* c;

x1 = (-b + temp) / (2 \* a);

x2 = (-b - temp) / (2 \* a);

}

25. Suppose averageTestScore denotes the average test score, highestScore denotes the highest test score, testScore denotes a test score, sum denotes the sum of all the test scores, and count denotes the number of students in class, and studentName denotes the name of a student.

a. First you design an algorithm to find the average test score. To find the average test score, first you need to count the number of students in the class and add the test score of each student. You then divide the sum by count to find the average test score. The algorithm to find the average test score is as follows:

i. Set sum and count to 0.

ii. Repeat the following for each student in class.

1. Get testScore

2. Increment count and update the value of sum by adding the current test score to sum.

iii. Use the following formula to find the average test score.

if (count is 0)

averageTestScore = 0;

otherwise

averageTestScore = sum / count;

b. The following algorithm determines and prints the names of all the students whose test score is below the average test score.

Repeat the following for each student in class:

i. Get studentName and testScore

ii.

if (testScore is less than averageTestScore)

print studentName

c. The following algorithm determines and highest test score

i. Get first student’s test score and call it highestTestScore.

ii. Repeat the following for each of the remaining student in class

1. Get testScore

2.

if (testScore is greater than highestTestScore)

highestTestScore = testScore;

d. To print the names of all the students whose test score is the same as the highest test score, compare the test score of each student with the highest test score and if they are equal print the name. The following algorithm accomplishes this

Repeat the following for each student in class:

i. Get studentName and testScore

ii.

if (testScore is equal to highestTestScore)

print studentName

You can use the solutions of the subproblems obtained in parts a to d to design the main algorithm as follows:

1. Use the algorithm in part a to find the average test score.

2. Use the algorithm in part b to print the names of all the students whose score is below the average test score.

3. Use the algorithm in part c to find the highest test score.

4. Use the algorithm in part d to print the names of all the students whose test score is the same as the highest test score

Chapter 2

1. a. false; b. false; c. false; d. true; e. true; f. false; g. true; h. true; i. false; j. true; k. false

3. c, f

5. The identifiers quizNo1 and quizno1 are not the same. C++ is case sensitive. The fifth letter of quizNo1 is uppercase N while the fifth character of quizno1is lowercase n. So these identifiers are different

7. a. 3

b. -0.5

c. 7.85714

d. Not possible. Both the operands of the operator % must be integers. Because the first operand, x \* z, is a floating-point value, the expression is invalid.

e. 28.5

f. Not possible. Both the operands of the operator % must be integers. Because the second operand, z, is a floating-point value, the expression is invalid.

g. Not possible. Both the operands of the operator % must be integers. Because the second operand, y + z, is a floating-point value, the expression is invalid.

h. 3.57143

9. x = 5, y = 7, z = 4, w = 3, t = 6

11. b and c are valid

13. a. 5.0 / 9 \* (F – 32)

b. 'A'

c. grade = "B+" ;

d. str = "seventh edition"

e. totalInches = 12 \* feet + inches;

f. i = i + 1;

g. v = 4 / 3 \* (3.1416 \* r \* r \* r);

h. s = 2 \* (3.1416 \* r \* r) + 2 \* (3.1416 \* r) \* h;

i. (a + b) / c \* (e \* f) – g \* h

j. (–b + (b \* b – 4 \* a \* c)) / (2 \* a)

15. x = 101

y = 28

z = 28

w = 117.5

t = 29.25

17. a. -40

b. 11.5

c. 17.75

d. 18.7

e. 20

f. 88.25

g. 4.0

19. a and c are correct

21. a. int num1;

int num2;

b. cout << "Enter two numbers separated by spaces." << endl;

c. cin >> num1 >> num2;

d. cout << "num1 = " << num1 << "num2 = " << num2

<< "2 \* num1 – num2 = " << 2 \* num1 – num2 << endl;

23. A correct answer is:

#include <iostream>

using namespace std;

const char STAR = '\*';

const int PRIME = 71;

int main()

{

int count, sum;

double x;

int newNum; //declare newNum

count = 1;

sum = count + PRIME;

x = 25.67; // x = 25.67;

newNum = count \* 1 + 2; //newNum = count \* ONE + 2;

sum++; //(x + sum)++;

sum = sum + count; //sum + count = sum;

x = x + sum \* count; // x = x + sum \* COUNT;

sum += 3; //sum += 3--;

cout << " count = " << count << ", sum = " << sum

<< ", PRIME = " << PRIME << endl;

return 0;

}

25. An identifier must be declared before it can be used.

27. a. x \*= 2;

b. x += y - 2;

c. sum += num;

d. z \*= x + 2;

e. y /= x + 5;

29.

a b c

a = (b++) + 3; 9 7 und

c = 2 \* a + (++b); 9 8 26

b = 2 \* (++c) – (a++); 10 45 27

31. (The user input is shaded.)

firstNum = 62

Enter three numbers: 35 10.5 27

The numbers you entered are 35, 10.5, and 27

z = 33

Enter grade: B

The letter that follows your grade is: C

33.

#include <iostream>

#include <string>

using namespace std;

const double X = 13.45;

const int Y = 34;

const char BLANK = ' ';

int main()

{

string firstName, lastName;

int num;

double salary;

cout << "Enter first name: ";

cin >> firstName;

cout << endl;

cout << "Enter last name: ";

cin >> lastName;

cout << endl;

cout << "Enter a positive integer less than 70: ";

cin >> num;

cout << endl;

salary = num \* X;

cout << "Name: " << firstName << BLANK << lastName << endl;

cout << "Wages: $" << salary << endl;

cout << "X = " << X << endl;

cout << "X + Y = " << X + Y << endl;

return 0;

}

Chapter 3

1. a. false; b. true; c. false; d. false; e. false; f. false; g. true; h. false; i. true; j. false; k. true

3. a. num1 = 28, num2 = 67, x = 35, y = 0.3

b. num1 = 35, num2 = 67, x = 28.3, y = 12.5

c. num1 = 67, num2 = 12, x = 35, y = 28.3

d. num1 = 35, num2 = 28, x = 0.3, y = 67

e. num1 = 35, num2 = 12, x = 28.3, y = 67

f. Input failure: num1 = 28, x = 35, trying to read the . (period) into num2.

5. a. 12.8 15 Bill

b. 0.8 12 Bill

c. Input failure: Trying to read B into y, which is an int variable. x = 12.8, y = 18, and name = "Lisa". The values of y and name are unchanged.

7. cmath

9. iomanip

11. iomanip

13. iostream

15. The function getline reads until it reaches the end of the current line. The newline character is also read but not stored in the string variable.

17. a. name = " Lance Grant", age = 23

b. name = " ", age = 23

19.

#include <iostream>

#include <fstream>

using namespace std;

int main()

{

int num1, num2;

ifstream infile;

ofstream outfile;

infile.open("input.dat");

outfile.open("output.dat");

infile >> num1 >> num2;

outfile << "Sum = " << num1 + num2 << endl;

infile.close();

outfile.close();

return 0;

}

21. fstream

23. a. Same as before.

b. The file contains the output produced by the program.

c. The file contains the output produced by the program. The old contents are erased.

d. The program would prepare the file and store the output in the file.

25. a. outfile.open("travel.dat ");

b. outfile >> fixed >> showpoint >> setprecision(2);

c. outfile >> day >> " " >> distance >> " " >> speed >> endl;

d. travelTime = distance / speed;

outfile >> travelTime;

e. fstream and iomanip.

Chapter 4

1. a. false; b. false; c. false; d. true; e. false; f. false; g. false; h. false; i. false; j. true

3. a. true; b. false; c. true; d. false; e. true

5. a. x == z: 0

b. y != z - 9: 0

c. x - y == z + 10: 1

d. !(z < w): 1

e. w - y < x - 2 \* z: 0

7. a. //-

b. 8 12 - 4

c. ^:

d. ++C

e. Amy

Amy < Bob

f. -12

\*\*

9. a. ?%!!

b. a b c d

##

c. Flying Coding

11. The value of done is: 0

13. Omit the semicolon after else. The correct statement is:

if (score >= 60)

cout << "You pass." << endl;

else

cout << "You fail." << endl;

15. The correct code is:

if (numOfItemsBought > 10)

shippingCharges = 0.0;

else if (5 <= numOfItemsBought && numOfItemsBought <= 10)

shippingCharges = 3.00 \* numOfItemsBought;

else if (0 < numOfItemsBought && numOfItemsBought < 5)

shippingCharges = 7.00 \* numOfItemsBought;

17. 1 18

19. if (sale > 20000)

bonus = 0.10

else if (sale > 10000 && sale <= 20000)

bonus = 0.05;

else

bonus = 0.0;

21. a. The output is: Discount = 10%. The semicolon at the end of the if statement terminates the if statement. So the cout statement is not part of the if statement. The cout statement will execute regardless of whether the expression in the if statement evaluates to true or false.

b. The output is: Discount = 10%. The semicolon at the end of the if statement terminates the if statement. So the cout statement is not part of the if statement. The cout statement will execute regardless of whether the expression in the if statement evaluates to true or false.

23. a. (x >= y) ? z = x – y : z = y – x;

b. (hours >= 40.0) ? wages = 40 \* 7.50 + 1.5 \* 7.5 \* (hours – 40)

: wages = hours \* 7.50;

c. (score >= 60) ? str = "Pass" : str = "Fail";

25. a. 40.00

b. 40.00

c. 55.00

27. a. 0 b. 4 c. 10 d. 25

29. a. 3 b. -20 c. 3 d. 5

31.

#include <iostream>

using namespace std;

const int SECRET = 5;

int main()

{

int x, y, w, z;

z = 9;

if (z > 10)

{

x = 12;

y = 5;

w = x + y + SECRET;

}

else

{

x = 12;

y = 4;

w = x + y + SECRET;

}

cout << "w = " << w << endl;

return 0;

}

33.

switch (classStanding)

{

case 'f':

dues = 150.00;

break;

case 's':

if (gpa >= 3.75)

dues = 75.00;

else

dues = 120.00;

break;

case 'j':

if (gpa >= 3.75)

dues = 50.00;

else

dues = 100.00;

break;

case 'n':

if (gpa >= 3.75)

dues = 25.00;

else

dues = 75.00;

break;

default:

cout << "Invalid class standing code." << endl;

}

Chapter 5

1. a. true; b. false; c. true; d. false; e. true; f. true; g. true; h. false

3. 13

5. if ch > 'Z' or ch < 'A'

7. Sum = 39

9. Sum = 37

11. a. 27

b. 7

c. 31 60 91 151

d. 50 93 143 236 379 615 994 1609 2603 4212

13. Replace the while loop statement with the following:

while (response == 'Y' || response == 'y')

Replace the cout statement:

cout << num1 << " + " << num2 << " = " << (num1 - num2)

<< endl;

with the following:

cout << num1 << " + " << num2 << " = " << (num1 + num2)

<< endl;

15. 5 4 3 2 1

17. 0 3 8 15 24

19. Loop control variable: j

The initialization statement: j = 1;

Loop condition: j <= 10;

Update statement: j++

The statement that updates the value of s: s = s + j \* (j – 1);

21. -1 1 3 5 7 6

23. a. \*

b. infinite loop

c. infinite loop

d. \*\*\*\*

e. \*\*\*\*\*\*

f. \*\*\*

25. The relationship between x and y is: 3y = x.

Output: x = 19683, y = 10

27.

0 - 24

25 - 49

50 - 74

75 - 99

100 - 124

125 - 149

150 - 174

175 - 200

29. a. both

b. do...while

c. while

d. while

31. In a pretest loop, the loop condition is evaluated before executing the body of the loop. In a posttest loop, the loop condition is evaluated after executing the body of the loop. A posttest loop executes at least once, while a pretest loop may not execute at all.

33. int num;

do

{

cout << "Enter a number less than 20 or greater than 75: ";

cin >> num;

}

while (20 <= num && num <= 75);

35. int i = 0, value = 0;

do

{

if (i % 2 == 0 && i <= 10)

value = value + i \* i;

else if (i % 2 == 0 && i > 10)

value = value + i;

else

value = value - i;

i = i + 1;

}

while (i <= 20);

cout << "value = " << value << endl;

The output is: value = 200

37 cin >> number;

while (number != -1)

{

total = total + number;

cin >> number;

}

cout << endl;

cout << total << endl;

39. a.

number = 1;

while (number <= 10)

{

cout << setw(3) << number;

number++;

}

b.

number = 1;

do

{

cout << setw(3) << number;

number++;

}

while (number <= 10);

41. a. 29

b. 2 8

c. 8 13 21 34

d. 28 43 71 114

43 -1 0 3 8 15 24

45. 12 11 9 7 6 4 2 1

Chapter 6

1. a. false; b. true; c. true; d. true; e. false; f. false; g. true; h. false; i. true; j. true; k. false; l. false;

m. false; n. true

3. a. 25 b. 56.3800 c. 13.0000 d. 9.0000 e. 15.4379 f. 5.8788

g. 12.0000 h. 504.0000 i. 13.0000 j. 7.0000 k. 11.0000

l. -8.0000 m. -85.0000 n. -243.0000 o. 0.0041 p. 1.3324

5. b and c

7. a, b, c, d, e are valid. In f, the second argument in the function call is missing. In g and h, the function call requires one more argument.

9. a. 2; int

b. 3; double

c. 4; char

d. 2; string

e. The function func1 requires 2 actual parameters. The type and the order of these parameters is: int, double

f. cout << func1(3, 8.5) << endl;.

g. cout << join("John", "Project Manager") << endl;.

h. cout << static\_cast<char>(static\_cast<int>(func3(4, 3, 17.6, 'A')) + 1)

<< endl;

11. bool isUppercaseLetter (char ch)

{

if (isupper(ch))

return true;

else

return false;

}

13. a. (i) 16 (ii) 27

b. The function computes *mk* - 1, where *m* and *n* are the arguments of the function and *k* = abs(*n*).

15. a. 385

b. This function computes 1+4+9+16+25+36+49+64+81+100

17. double funcEx17(double x, double y, double z)

{

return x \* pow(y, z);

}

19. a. In a void function, a return statement is used without any value such as return;

b. In a void function, a return statement is used to exit the function early.

21. a. A variable declared in the heading of a function definition is called a formal parameter. A variable or expression used in a function call is called an actual parameter.

b. A value parameter receives a copy of the actual parameter’s data. A reference parameter receives the address of the actual parameter.

c. A variable declared within a function or block is called a local variable. A variable declared outside of every function definition is called a global variable.

23. void funcThreeTimes(double x)

{

cout << fixed << showpoint << setprecision(2);

cout << 3 \* x << endl;

}

25. void initialize(int& x, double& y, string& str)

{

x = 0;

y = 0;

str = "";

}

27. 3, 2, 8

3, 7, 24

1190, 7, 187

245157, 1131, 187

29. #include <iostream>

using namespace std;

int secret(int, int);

void func(int x, int& y);

int main()

{

int num1, num2;

\_\_1\_\_ num1 = 6;

\_\_2\_\_ cout << "Enter a positive integer: ";

\_\_3\_\_ cin >> num2;

\_\_4\_\_ cout << endl;

\_\_8\_\_ cout << secret(num1, num2) << endl;

\_\_9\_\_ num2 = num2 – num1;

\_10\_\_ cout << num1 << " " << num2 << endl;

\_15\_\_ func(num2, num1);

\_16\_\_ cout << num1 << " " << num2 << endl;

\_17\_\_ return 0;

}

int secret(int a, int b)

{

int d;

\_\_5\_\_ d = a + b;

\_\_6\_\_ b = a \* d;

\_\_7\_\_ return b;

}

void func (int x, int& y)

{

int val1, val2;

\_11\_\_ val1 = x + y;

\_12\_\_ val2 = x \* y;

\_13\_\_ y = val1 + val2;

\_14\_\_ cout << val1 << " " << val2 << endl;

}

If the input is 10, the output is:

96

6 4

10 24

34 4

31. void traceMe(double& x, double y, double& z)

{

if (x != 0)

z = sqrt(y) / x;

else

{

cout << "Enter a nonzero number: ";

cin >> x;

cout << endl;

z = floor(pow(y, x));

}

}

33. 20 30

5 30

35. 11, 3

16, 2

19, 3

24, 2

37. a, b, c, and e are correct.

Chapter 7

1. a. true; b. false; c. true; d. false; e. false; f. true; g. true; h. true; i. false; j. false; k. false

3. Only a and c are valid.

5. birdType readIn()

{

string str;

birdType bird = 0;

cin >> str;

if (str == "Peacock")

bird = PEACOCK;

else if (str == "Sparrow")

bird = SPARROW;

else if (str == "Canary")

bird == CANARY;

else if (str == "Penguin")

bird = PENGUIN;

else if (str == "Ostrich")

bird = OSTRICH;

else if (str == "Eagle")

bird = EAGLE;

else if (str == "Cardinal")

bird = CARDINAL;

else if (str == "Hummingbird")

bird = HUMMINGBIRD;

else

cout << "Invalid bird name." << endl;

return bird;

}

7. Because there is no name for an anonymous type, you cannot pass an anonymous type as a parameter to a function and a function cannot return an anonymous type value. Also, values used in one anonymous type can be used in another anonymous type, but variables of those types are treated differently.

9. The statement in Line 2 should be:

using namespace std; //Line 2

11. The statement in Line 2 should be:

using namespace std; //Line 2

13. Either include the statement:

using namespace aaa;

before the function main or refer to the identifiers x and y in main as aaa::x and aaa::y, respectively.

15. a. Heelo Thlre

b. Giamond Dold

c. Ca+ J+va

17. Regular exercise

Regular exercise and low fat diet

33

8

8

health insurance

insurance

Regular exercise can reduce health insurance $$$$.

$ocial Nedia!!

14

Social Media!!

**Chapter 8**

**1.** a. true; b. true; c. true; d. false; e. false; f. false; g. false; h. false; i. true; j. false; k. false; l. false

**3.** a. This declaration is correct.

b. Array size must be positive. A correct answer is: int testScores[10];

c. This declaration is correct.

d. Array size must be a positive integer not a range. A correct answer is: int list100[100];

e. gpa is an array of size 50. The expression [50] should be after gpa. The correct statement is: double gpa[50];

f. LENGTH must be declared as integral, such as int . A correct statement is: const int LENGTH = 26;

g. This declaration is correct.

**5.** 0 to 98

**7.** 0 1 4 9 16

32 1 4 32 36

**9.** 1 2 2 4 8 32 224 6944

**11.** int myList[10];

for (int i = 0; i < 10; i++)

myList[i] = i;

**13.** If array index is less than 0 or greater than arraySize – 1, we say that the array index is out-of bounds. C++ does not check for array indices within bound.

**15.** a. double heights[10] = {5.2, 6.3, 5.8, 4.9, 5.2, 5.7, 6.7, 7.1, 5.10, 6.0};

or

double heights[] = {5.2, 6.3, 5.8, 4.9, 5.2, 5.7, 6.7, 7.1, 5.10, 6.0};

b. int weights[7] = {120, 125, 137, 140, 150, 180, 210};

or

int weights[] = {120, 125, 137, 140, 150, 180, 210};

c. char specialSymbols[] = {'$', '#', '%', '@', '&', '! ', '^'};

d. string seasons[4] = {**"**fall**"**, **"**winter**"**, **"**spring**"**, **"**summer"};

or

string seasons[] = {**"**fall**"**, **"**winter**"**, **"**spring**"**, **"**summer"};

**17.** list[0] = 6, list[1] = 10, list[2] = 14, list[3] = 18, list[4] = 22, list[5] = 0, list[6] = 0.

**19.** 16 32 44 56 68 37 20

**21.** a. Correct.

b. Correct.

c. Incorrect. The size of score is 50, so the call should be tryMe(score, 50);

d. Correct.

e. Incorrect. The array gpas is of type double while the parameter x of tryMe is of type int. So there will be mismatch data type error.

**23.** 1 25000.00 750.00

2 36500.00 1095.00

3 85000.00 2550.00

4 62500.00 1875.00

5 97000.00 2910.00

**25.** List elements: 11 16 21 26 30

**27.** 1 3.50 10.70 235.31

2 7.20 6.50 294.05

3 10.50 12.00 791.68

4 9.80 10.50 646.54

5 6.50 8.00 326.73

**29.** No.

**31.** 2 3 3 2 4 7 16 20

**33.** No, because during compile time the formal parameter list has no first and last elements.

**35.** a. Invalid; the assignment operator is not defined for C-strings.

b. Invalid; the relational operators are not defined for C-strings.

c. Invalid; the assignment operator is not defined for C-strings.

d. Valid

**37.** a. strcpy(str1, "Sunny Day");

b. length = strlen(str1);

c. strcpy(str2, name);

d. if (strcmp(str1, str2) <= 0)

cout << str1 << endl;

else

cout << str2 << endl;

**39.** int temp[3][4] = {{6, 8, 12, 9},

{17, 5, 10, 6},

{14, 13, 16, 20}};

**41.** a. 30

b. 5

c. 6

d. row

e. column

**43.** a. beta is initialized to 0.

b. First row of beta: 0 1 2

Second row of beta: 1 2 3

Third row of beta: 2 3 4

c. First row of beta: 0 0 0

Second row of beta: 0 1 2

Third row of beta: 0 2 4

d. First row of beta: 0 2 0

Second row of beta: 2 0 2

Third row of beta: 0 2 0

e. First row of beta: 0 0 0

Second row of beta: 0 1 2

Third row of beta: 0 2 1

Chapter 9

**1.** a. false; b. true; c. false; d. false ; e. false; f. false; g. false; h. true; i. false; j. true; k. true

**3.** carType newCar;

newCar.manufacturer = "GMT";

newCar.model = " Cyclone";

newCar.modelType = "sedan";

newCar.color = "blue"

newCar.numOfDoors = 4;

newCar.cityMilesPerGallon = 28;

newCar.highwayMilesPerGallon = 32;

newCar.yearBuilt = 2006;

newCar.price = 25000.00;

**5.** fruitType fruit;

fruit.name = "banana";

fruit.color = "yellow";

fruit.fat = 1;

fruit.sugar = 15;

fruit.carbohydrate = 22;

**7.** if (firstHouse.style == secondHouse.style &&

firstHouse.price == secondHouse.price)

cout << "true" << endl;

else

cout << "false" << endl;

**9.** Assignment statement and function return value.

**11.** student.name.first = "Linda";

student.name.last = "Brown";

student.gpa = 3.78;

student.course.name = "Calculus";

student.course.callNum = 23827;

student.course.credits = 4;

student.course.grade = 'A';

**13**. a. Invalid; the member name of newEmployee is a struct. Specify the member names to store the value "John Smith". For example,

newEmployee.name.first = "John";

newEmployee.name.last = "Smith";

b. Invalid; the member name of newEmployee is a struct. There are no aggregate output operations on a struct. A correct statement is:

cout << newEmployee.name.first << " "

<< newEmployee.name.last << endl;

c. Valid

d. Valid

e. Invalid; employees is an array. There are no aggregate assignment operations on arrays.

**15**. partsType inventory[100];

**17**. void getData(partsType& pType)

{

for (int j = 0; j < length; j++)

{

cin >> pType.partName;

cin >> pType.partNum;

cin >> pType.price;

cin >> pType.quantitiesInStock;

}

}

for (int j = 0; j < 100; j++)

getData(inventory[i]);

Chapter 10

1. a. false; b. false; c. true; d. false; e. false;

**3.** The type of function print is missing. Also, a constructor has no type. The statements in Lines 5 and 6 should be:

void print() const; //Line 5

temp(int = 0, int = 0); //Line 6

**5.** A class is not a function. Semicolon after private should be a colon, missing semicolon after }; and a constructor has no type. The statements in Lines 1, 8, 9 and 13 should be:

class discover() //Line 1

discover(string, int, int); //Line 8

private: //Line 9

}; //Line 13

**7.** a. void bagType::set(string s, double a, double b, double c, double d)

{

style = s;

l = a;

w = b;

h = c;

price = d;

}

b. void bagType::print() const

{

cout << "Bag Type: " << style << ", length = " << l

<< ", width = " << w << ", height = " << h

<< ", price = $ " << price << endl;

}

c. bagType::bagType()

{

style = "";

l = 0.0;

w = 0.0;

h = 0.0;

price = 0.0;

}

d. newBag.print();

e. bagType tempBag("backPack", 15, 8, 20, 49.99);

**9.** The functions print, getSalary, and getNumOfServiceYears are accessors; functions setData and updateSalary are mutators.

**11.** a. 28; b. 8 c. 1; d. 9;

**13.** a. 14

b. 3

c. The class temporary has only one constructor. Because this is a constructor with default parameters, it can be used to initialize an object without specifying any parameters. For example, the following statement creates the object newObject and its instance variables are initialized to "", 0, and 0, respectively.

temporary newObject;

**15.** The statement in Line 1 creates object1 and initializes the instance variables of this object to "", 0, 0, that is, object1.description = "";, object1.first = 0.0;, and object1.second = 0.0;. The statement in Line 2 creates object2 and initializes the instance variables of this object as follows: object2.description = "rectangle";, object2.first = 3.0;, and object2.second = 5.0;. The statement in Line 3 creates object3 and initializes the instance variables of this object as follows: object3.description = "circle";, object3.first = 6.5;, and object3.second = 0.0;. The statement in Line 4 creates object4 and initializes the instance variables of this object as follows: object4.description = "cylinder";, object4.first = 6.0;, and object4.second = 3.5;.

**17.** There two built-in operations for class objects: Member access (.) and assignment (=).

**19**. a.

int testClass::sum()

{

return x + y;

}

void testClass::print() const

{

cout << "x = " << x << ", y = " << y << endl;

}

testClass::testClass()

{

x = 0;

y = 0;

}

testClass::testClass(int a, int b)

{

x = a;

y = b;

}

b. One possible solution. (We assume that the name of the header file containing the definition of the class testClass is Exercise19Ch10.h.)

#include <iostream>

#include "Exercise19Ch10.h"

int main()

{

testClass one;

testClass two(4, 5);

one.print();

two.print();

return 0;

}

**21.** a. personType student("Buddy", "Arora");

b. student.print();

c. student.setName("Susan", "Gilbert");

**23.**  A constructor is a member of a class and it executes automatically when a class object is instantiated and a call to the constructor is specified in the object declaration. A constructor is included in a class so that the objects are properly initialized when they are declared.

**25.**  A destructor is a member of a class and if it is included in a class, it executes automatically when a class object goes out of scope. Its main purpose is to deallocate the dynamic memory created by an object.

**27**.

a. myClass::count = 0;

b. myClass.incrementCount();

c. myClass.printCount();

d.

int myClass::count = 0;

void myClass::setX(int a)

{

x = a;

}

void myClass::printX() const

{

cout << x;

}

void myClass::printCount()

{

cout << count;

}

void myClass::incrementCount()

{

count++;

}

myClass::myClass(int a)

{

x = a;

}

e. myClass myObject1(5);

f. myClass myObject2(7);

g.

The statements in Lines 1 and 2 are valid.

The statement in Line 3 should be: myClass::printCount();.

The statement in Line 4 is invalid because the member function printX is not a static member of the class, and so cannot be called by using the name of class.

The statement in Line 5 is invalid because count is a private static member variable of the class.

h.

5

2

2

3

14

3

3

Chapter 11

**1.** a. false; b. false; c. true; d. true; e. true; f. true; g. true; h. true; i. false; j. false; k. true

**3.** Some of the member variables that can be added to the class employeeType are: department, salary, employeeCategory (such as supervisor and president), and employeeID. Some of the member functions are: setInfo, setSalary, getSalary, setDepartment; getDepartment, setCategory, getCategory, setID, and getID.

class employeeType: public personType

{

public:

void setInfo(string, string, string, double, string, string);

void setSalary(double);

void setDepartment(string);

void setCategory(string);

void setID(string);

double getSalary() const;

string getDepartment(string) const;

string getCategory()const;

string getID()const;

private:

string department;

double salary;

string employeeCategory;

string employeeID;

};

**5.** a. The base class is computers and the derived class is personalComputers.

b. This is public inheritance.

**7.** Private members of the object newCylinder are xCoordinate, yCoordinate, radius, and height.

**9.** Missing : in the first statement. The first statement should be:

class derivedFromTemp: public temp

Also missing ; after }. It should be

};

**11.** a. void print() const;

b. void set(int, int, int);

void get(int&, int&, int&);

**13.**  First a constructor of class one will execute, then a constructor of class two will execute, and finally a constructor of class three will execute.

**15.** a. Invalid. z is an instance variable of the derived class, it cannot be accessed by the members of the class smart.

b. Invalid. secret is a private member of the class smart. It cannot be accessed directly outside of the class. Also z is a private member of the class superSmart. It cannot be accessed directly outside of the class.

c. Valid

d. Invalid. smart is the name of a class, not an object of this class. It cannot be used to call its member function print.

e. Invalid. superSmart is the name of a class. It cannot be used to access its members.

**17.**  Between the preprocessor directive#ifndef and #endif. The definitions of the classes one and two can be placed between these directives as follows:

#ifndef H\_one #ifndef H\_two

#define H\_one #define H\_two

//place the definition of the //place the definition of the

//class one here //class two here

#endif #endif

**19.** In a private inheritance, the public members of the base class are public members of the derived class. They can be directly accessed in the derived class. The protected members of the base class are protected members of the derived class. They can be directly accessed by the member functions (and friend functions) of the derived class. The private members of the base class are hidden in the derived class. They cannot be directly accessed in the derived class. They can be accessed by the member functions (and friend functions) of the derived class through the public or protected members of the base class.

**21**. In a public inheritance, the public members of the base class are private members of the derived class. They can be accessed by the member functions (and friend functions) of the derived class. The protected members of the base class are private members of the derived class. They can be accessed by the member functions (and friend functions) of the derived class. The private members of the base class are hidden in the derived class. They cannot be directly accessed in the derived class. They can be accessed by the member functions (and friend functions) of the derived class through the public or protected members of the derived class.

**23.** The protected members of a base class can be directly accessed by the member functions of the derived class, but they cannot be directly accessed in a program that uses that class. The public members of a class can be directly accessed by the member functions of any derived class as well as in a program that uses that class.

**25.** The members setX, print, y, and setY are protected members in class third. The private member x of class first is hidden in class third and it can be accessed in class third only through the protected and public members of class first.

**27.** Because the memberAccessSpecifier is not specified, it is a private inheritance. Therefore, all the members of the class first becomes private members in class second.

**29.** a. void two::setData(int a, int b, int c)

{

one::setData(a, b);

z = c;

}

b. void two::print() const

{

one::print();

cout << z << endl;

}

Chapter 12

1. a. false; b. false ; c. false; d. true; e. false; f. true; g. false; h. false; i. true; j. false;

k. true; l. true; m. false; n. true; o. true; p. false;

**3.**  a. To create a pointer, in the variable declaration, operator \* is placed between the data type and the variable name. For example the statement int \*p; declares p to be a pointer of type int.

b. To dereference a pointer, in an expression, the operator \* is placed to the left of the pointer. For example, if p is a pointer of type int, the expression cout << \*p << endl; outputs the data stored in the memory space to which p points.

**5.** \*numPtr given the address of the memory location to which numPtr points, while &numPtr gives the address of numPtr.

**7.** numPtr = &num;

(\*numPtr)++;

**9.** 37 37 28

**11.** 34 27

**13.** 90.00 86.00 88.00

**15.** In Line 6, the operator delete deallocates the memory space to which nextPtr points. So the expression \*nextPtr, in Line 9, does not have a valid value.

**17.** 12 8 7 25 16 24 36

**19.** numPtr = 1058 and gpaPtr = 2024

**21.** The operator delete deallocates the memory space to which a pointer points.

**23.** a. num = new int[10];

b. for (int j = 0; j < 10; j++)

cin >> num[j];

c. delete [] num;

**25**. Because at compile time dynamic arrays have no first and last elements, so the functions begin and end cannot be called on dynamic arrays.

**27**. In a shallow copy of data, two or more pointers point to the same memory space. In a deep copy of data, each pointer has its own copy of the data.

**29**. myList: 3 4 6 9 13

yourList: 7 8 10 13 17 10 6 3 1 0

**31**. The copy constructor makes a copy of the actual variable.

**33**. Classes with pointer data members should include the destructor, overload the assignment operator, and explicitly provide the copy constructor by including it in the class definition and providing its definition.

**35.**

ClassA x: 4

ClassA x: 6

ClassB y: 10

**37**. Yes.

**39**. a. Because employeeType is an abstract class, you cannot instantiate an object of this class. Therefore, this statement is illegal.

b. This statement is legal.

c. This statement is legal.

**Chapter 13**

1. a. true; b. false; c. true; d. false; e. false; f. false; g. false; h. false; i. true; j. false;

k. false; l. true; m. true

**3.** ., .\*, ::, ?:, and sizeof

**5.** The statement return this; returns the address of the object while the statement return \*this;returns the value of the object.

**7.** A friend function is nonmember of a class while a member function is a member of a class.

**9.** Because the left operand of << is a stream object, which is not of the type mystery.

**11**. (), [], ->, and =

**13**. a. friend strange operator+(const strange&, const strange&);

b. friend bool operator==(const strange&, const strange&);

c. friend strange operator++(strange&, int);

**15**. In Line 4, the return type of the function operator\* should be temp. The correct statement is:

temp operator\*(const temp& obj); //Line 4

**17**. In Line 3, the return type of the function operator should be bool. The correct statement is:

friend bool operator<(const mystery& a,

const mystery& b); //Line 3

**19**. In Line 3 and 11, the return type of the function operator should be discover. Also since operator+ is a friend function of the class, the name of the class and the scope resolution operator in the heading of the function, in Line 11, is not needed. The correct statements are:

friend discover operator+(const discover&,

const discover&); //Line 3

discover operator+(const discover& a,

const discover& b) //Line 11

**21**. A reference to an object of the class istream.

**23**. Suppose that a class, say temp, overloads the pre- and post-increment operator++ and tempObj is an object of the class temp. Then the statement ++tempObj; is compiled as tempObj.operator++(); ; and the statement tempObj++; is compiled as tempObj.operator++(0);. The dummy parameter distinguishes between the pre- and post-increment operator functions. There are similar conventions for the pre- and post-decrement operators.

**25**. None.

**27**. One.

**29**.

class complexType

{

//overload the stream insertion and extraction operators

friend ostream& operator<<(ostream&, const complexType&);

friend istream& operator>>(istream&, complexType&);

public:

void setComplex(const double& real, const double& imag);

//set the complex number according to the parameters

//Postcondition: realPart = real; imaginaryPart = imag

complexType(double real = 0, double imag = 0);

//constructor

//initialize the complex number according to the parameters

//Postcondition: realPart = real; imaginaryPart = imag

complexType operator+(const complexType& otherComplex) const;

//overload +

complexType operator\*(const complexType& otherComplex) const;

//overload \*

complexType operator~() const;

double operator!() const;

bool operator==(const complexType& otherComplex) const;

//overload ==

private:

double realPart; //variable to store the real part

double imaginaryPart; //variable to store the imaginary part

};

// Definitions of operator~ and operator!

complexType complexType::operator~() const

{

complexType temp = \*this;

temp.imaginaryPart = -temp.imaginaryPart;

return temp;

}

double complexType::operator!() const

{

return (pow((realPart \* realPart +

imaginaryPart \* imaginaryPart), 0.5));

}

**31**. When the class has pointer data members.

**33**. Error in Line 4. A template instantiation can be for only a built-in type or a user-defined type. The word “type” between the angular brackets must be replaced either with a built-in type or a user-defined type.

**35**. a. 12 b. Sunny Day

**37**.

template <class Type>

void swap(Type &x, Type &y)

{

Type temp;

temp = x;

x = y;

y = temp;

}

**Chapter 14**

**1.** a. false; b. true; c. true; d. false; e. true; f. false; g. false; h. true; i. false; j. true;

k. false; l. true; m. false;

**3.** The program will terminate with an error message.

**5.** If an exception is thrown in a try block, the remaining statements in that try block are ignored. The program searches the catch blocks in the order they appear after the try block, and looks for an appropriate exception handler. If the type of thrown exception matches the parameter type in one of the catch blocks, the code of that catch block executes and the remaining catch blocks after this catch block are ignored.

**7.** At most one.

**9.** throw expression;

where expression is a constant value, variable, or object.

**11.** The catch block has no associated try block, that is, the catch block does not follow any try block. Also, the try block has no associated catch block, that is, there is no catch block that follows the try block. The cout statement just before the catch block disassociates the catch block from the try block. The correct code is:

double radius;

try

{

cout << "Enter the radius: ";

cin >> radius;

cout << endl;

if (radius < 0.0)

throw radius;

cout << "Area: " << 3.1416 \* radius \* radius << endl;

}

catch (double x)

{

cout << "Negative radius: " << x << endl;

}

**13.**  a. Leaving the try block.

b. Current balance: 975

Balance must be greater than 1000.00

c. Current balance: -2000

Balance must be greater than 1000.00

**15**. a.

Entering the try block.

Exception: Lower limit violation.

After the catch block

b.

Entering the try block.

Exception: 0

After the catch block

c.

Entering the try block.

Exiting the try block.

After the catch block

d.

Entering the try block.

Exception: 0

After the catch block

**17**. invalid\_argument

**19.** A throw statement.

**21**. (Assume that the definition of the class tornadoException is in the header file tornadoException.h.)

#include <iostream>

#include "tornadoException.h"

using namespace std;

int main()

{

int miles;

try

{

cout << "Enter the miles: ";

cin >> miles;

cout << endl;

if (miles < 5)

throw tornadoException();

else

throw tornadoException(miles);

}

catch (tornadoException tE)

{

cout << tE.what() << endl;

}

return 0;

}

**23.** A function specifies the exceptions it throws in its heading using the throw clause.

**25.** (1) Do nothing; (2) Partially process the exception and throw the same exception or a new exception; (3) Throw a new exception.

Chapter 15

1. a. true; b. true; c. false; d. false; e. false; f. false; g. true; h. true;

3. The case in which the solution is defined in terms of smaller versions of itself.

5. A function that calls another function and eventually results in the original function call is said to be indirectly recursive.

7. a. The statements in Lines 2 and 3.

b. The statements in Lines 4 and 5.

c. Any nonpositive integer.

d. It is a valid call. The value of mystery(0) is 0.

e. It is an invalid call. Infinite recursion.

f. It is a valid call. The value of mystery(-3) is 6.

9.a. 8 5 2 b. 7 c. 6 3 d. -85

11. a. It does not produce any output.

b. 5 6 7 8 9

c. It does not produce any output.

d. It does not produce any output.

13. a. 2

b. 3

c. 5

d. 21

15. a. 10 b. 21 c. -23 d. 2 e. -56

17.



The base cases are when *n* = 0 or *n* = 1. The general case is specified by the option otherwise.

Chapter 16

1. a. true; b. false; c. false; d. true; e. false; f. false; g. false; h. true

3. int seqOrderedSearch(const int list[], int listLength,

int searchItem)

{

int loc;

bool found = false;

for (loc = 0; loc < listLength; loc++)

if (list[loc] >= searchItem)

{

found = true;

break;

}

if (found)

if (list[loc] == searchItem)

return loc;

else

return -1;

else

return -1;

}

5. List before the first iteration: 38, 60, 43, 5, 70, 58, 15, 10

List after the first iteration: 38, 43, 5, 60, 58, 15, 10, 70

List after the second iteration: 38, 5, 43, 58, 15, 10, 60, 70

List after the third iteration: 5, 38, 43, 15, 10, 58, 60, 70

List after the fourth iteration: 5, 38, 15, 10, 43, 58, 60, 70

List after the fifth iteration: 5, 15, 10, 38, 43, 58, 60, 70

List after the sixth iteration: 5, 10, 15, 38, 43, 58, 60, 70

List after the seventh iteration: 5, 10, 15, 38, 43, 58, 60, 70

7. 3

9. 4, 15, 18, 20, 25, 32, 45, 91, 62, 88, 66

11. Bubble sort: 49,995,000; selection sort: 49,995,000; insertion sort: 25,007,499

13. 26

15. a. 7 b. 3 c. 6 d. 6 e. 8 f. 8

17. To use a vector object in a program, the program must include the header file vector.

19. 1 3 5 7 9

21.

a. vector<int> secretList;

b.

secretList.push\_back(56);

secretList.push\_back(28);

secretList.push\_back(32);

secretList.push\_back(96);

secretList.push\_back(75);

c.

for (unsigned int i = 0; i < secretList.size(); i++)

cout << secretList[i] << " ";

cout << endl;

23. a. cout << myList.front() << " " << myList.back() << endl;

b length = myList.size();

c.

for (int i = 0; i < myList.size(); i++)

cout << myList[i] << " ";

cout << endl;

25. 0 2 6 12 20

Chapter 17

1. a. true; b. false; c. false; d. false; e. false; f. true; g. true; h. false; j. false; k. true;

l. true; m. false; n. false; o. true;

3. nullptr

5. Before deletion the link field of the third node stores the address of the fourth node. After deletion the link field of the third node will store the address of the next node (old) fifth node. If there was no fifth node, after deletion the link field will store the value nullptr. Therefore after deleting the fourth node, the link field of the third node is changed. So a pointer to the third node is needed.

7. a. false

b. true

c. true

d. true

e. true

9. a. current->link->info = 52;

b. current = temp->link;

c. trail = current->link;

d. temp = nullptr;

e. temp->link->link->link->info = 36;

f. while (current->info != 10)

current = current->link;

11. a. while (current != nullptr)

current = current->link;

b. temp = new nodeType;

temp->info = 68;

temp->link = last;

trail->link = temp;

c. delete last;

trail->link = nullptr;

last = trail;

d. trail = temp->link;

temp->link = trail->link;

delete trail;

13. After the execution of the statement in Line 5, trail is NULL, so trail->info does not exist. This code will result in run time error.

15. 33 62 28

17.nodeType head, p, q;

head = new nodeType;

head->info = 72;

head->link = nullptr;

p = new nodeType;

p->info = 43;

p->link = head;

head = p;

p = head->link;

q = new nodeType;

q->info = 8;

q->link = nullptr;

p->link = q;

q = new nodeType;

q->info = 12;

q->link = p;

head->link = q;

p = head;

while (p != nullptr)

{

cout << p->info << " ";

p = p->link;

}

cout << endl;

The output of this code is: 43 12 72 8

19. a. The function begin returns an iterator to the first node of a linked list.

b. The function end returns an iterator one past the last node of a linked list.

21. The item to be deleted is not in the list.

88 72 36 26 48 45

23.



25.



Chapter 18

**1.** a. true; b. false; c. false; d. true; e. false; f. true; g. false; h. false; i. true; j. false;

k. true; l. false; m. false; n. false;

**3.** 4

**5.** 13 5 12

num1 = 15

num2 = 21

**7.** secretNum = 226

**9.** a. 40

b. 11

c. 2

d. 70

**11.** a. (x + y ) \* z - w

b. x \* y / z + w

c. x \* (y + z) - w

**13.** 1 16 27 16 5

**15.**  If the stack is nonempty, the statement stack.top(); returns the top element of the stack and the statement stack.pop(); removes the top element of the stack.

**17.** template <class elemType>

elemType second(stackType<elemType> stack)

{

elemType temp1, temp2;

if (stack.isEmptyStack())

{

cout << "Stack is empty." << endl;

exit(0); //terminate the program

}

temp1 = stack.top();

stack.pop();

if (stack.isEmptyStack())

{

cout << "Stack has only one element." << endl;

exit(0); //terminate the program

}

temp2 = stack.top();

stack.push(temp1);

return temp2;

}

**19.** 48 72 0 15

stack: 36

queue: 0 88 10 52 67

**21**. a. queueFront = 50; queueRear = 0.

b. queueFront = 51; queueRear = 99.

**23**. a. queueFront = 25; queueRear = 76.

b. queueFront = 26; queueRear = 75.

**25**. 51

**27**. 5 -4 5 -7 1 2 1 4 1 -2 2 -7 7 -6

**29**.

template <class Type>

void reverseStack(stackType<Type> &s)

{

linkedQueueType<Type> q;

Type elem;

while (!s.isEmptyStack())

{

elem = s.top();

s.pop();

q.addQueue(elem);

}

while (!q.isEmptyQueue())

{

elem = q.front();

q.deleteQueue();

s.push(elem);

}

}

**31**.

template <class Type>

int queueType<Type>::queueCount()

{

return count;

}

**33**.



**35**.

