Chapter 1

1. a. true; b. false; c. false; d. true; e. false; f. true; g. true; h. true; i. false; j. false; k. true; l. true; m. true; n. true; o. true; p. false; q. true; r. true; s. true

3. Base 2 or binary.

5. In linking, an object program is combined with other programs in the library, used in the program, to create the executable code.

7. #

9. Programming is a process of problem solving.

11. (1) Analyze and outline the problem and its solution requirements, and design an algorithm to solve the problem. (2) Implement the algorithm in a programming language, such as C++, and verify that the algorithm works. (3) Maintain the program by using and modifying it if the problem domain changes.

13. 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;

15. 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

17. 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; |

19. 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

21. Suppose x1 and x2 are the real roots 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);

}

23. 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, 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 the highest test score

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

ii. Repeat the following for each of the remaining students in the 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 the 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. true; d. true; e. false; f. false; g. true; h. true; i. false; j. false; k. true; l. false

3. b, e

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 quizno1 is lowercase n. So these identifiers are different

7. a. 7

b. 5.50

c. -1.00

d. Not possible. Both the operands of the operator % must be integers. y + z is of type double. Both operands, y + z and x, of %V must be integers.

e. 13.50

f. 1

g. 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.

h. 3.00

9. x = 9, y = 5, z = 3, w = -3

11. a and c are valid

13. a. 9.0 / 5 \* C + 32

b. static\_cast<int>('+')

c. static\_cast<int>(x + 0.5)

d. str = "C++ Programming is exciting"

e. totalInches = 12 \* feet + inches

f. i++, ++i, or 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) / d \* (e \* f – g \* h)

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

15. x = 101

y = 11

z = 104

w = 159.00

t = 81.50

17. a. 1000

b. 42.50

c. 1.25

d. 11.00

e. 9

f. 88.25

g. -2.00

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 += 5;

b. x \*= 2 \* y

c. totalPay += currentPay;

d. z \*= (x + 2);

e. y /= x + 5;

29.

a b c

a = (b++) + 3; 8 3 und

c = 2 \* a + (++b); 8 2 12

b = 2 \* (++c) – (a++); 9 -3 11

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 = 18;

const char STAR = '\*';

int main()

{

string employeeID;

string department;

int num;

double salary;

cout << "Enter employee ID: ";

cin >> employeeID;

cout << endl;

cout << "Enter department: ";

cin >> department;

cout << endl;

cout << "Enter a positive integer less than 80: ";

cin >> num;

cout << endl;

salary = num \* X;

cout << "ID: " << employeeID << endl;

cout << "Department " << department << endl;

cout << "Star: " << STAR << endl;

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

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

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

return 0;

}

Chapter 3

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

3. a. int1 = 67, int2 = 48, dec1 = 56.5, dec2 = 62.72

b. int1 = 48, int2 = -1, dec1 = 0.5, dec2 = 67

c. int1 = 48, int2 = 62, dec1 = 56.5, dec2 = 67

d. int1 = 56, int2 = 67, dec1 = 0.5, dec2 = 48

e. Input failure: int1 = 56; trying to read the . (period) into int2.

5. a. Samantha 168.5 46

b. Samantha 0.5 168

c. \*\* 2.7 45

Input failure: Trying to read S into dec, which is a double variable. The values of dec, num and str are unchanged.

7. The function pow calculate xy in a program. That is, pow(x, y) = xy. To use this function the program must include the header file. cmath

9. The manipulator scientific is used to output floating-point numbers in scientific format. To use this function the program must include the header file iomanip.

11. The manipulator setw is used to output the value of an expression in a specific number of columns.

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 = " Christy Miller", height = 5.4

b. name = " ", height = 5.4

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("sales.dat ");

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

c. revenue = numOfJuiceBottlesSold \* costOfaJuiceBottle;

d. outfile >> numOfJuiceBottlesSold >> " "

>> costOfaJuiceBottle >> " " >> revenue >> endl;

e. outfile.close();

Chapter 4

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

3. a. false; b. true; c. true; d. 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. 12 / 2 != 4 + 1

c. \*

d. C++

C++

e. low

high

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 << "Pass" << endl;

else

cout << "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. 20 10

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 : (x + y) / 2;

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

: wages = hours \* 7.50;

c. (loanAmount >= 200000) ? closingCosts = 10000 : closingCosts = 8000;

25. a. 40.00

b. 40.00

c. 55.00

27. a. 8 b. 64 c. 1 d. 12

29. a. 7 b. 12167 c. 8000 d. 3

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 ; i. false; j. true

3. 40

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

7. Sum = 22

9. temp = 0

11. a. 20 \*

b. \*

c. 41 70 111 \*

d. 27 44 71 \*

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. 2 3 4 5 6

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. num = 485, y = 15

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. 36 94 260

b. 4 20

c. 30

d. 98 250

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. 18 b. 20.50 c. 87.20 d. 16.00 e. 1717.82 f. 2.80 g. 14.00 h. 11.16 i. 27.00

j. 20.00 k. 19.00 l. -5.00 m. 2.25 n. 4096.00 o. 0.01 p. 3.03

5. a and b

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; double

b. 3; int

c. 3; string

d. 2; char

e. The function third requires 4 actual parameters. The type and the order of these parameters is: string, string, int, double

f. cout << first(2.5, 7.8) << endl;

g. cout << grade(82.50, 92.50) << endl;

h. cout << third("John", "Blair", 26, 132.5) << endl;

11. bool isWhitespace (char ch)

{

if (isspace(ch))

return true;

else

return false;

}

13. a. (i) 72 (ii) -200

b. The function computes *mn*, where *m* and *n* are the arguments of the function.

15. a. 385

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

17. double funcEx17(double x, double y)

{

return pow(x, y) + pow(y, x);

}

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 funcEx23(int num)

{

if (num % 2 == 0)

cout << 2 \* num << endl;

else

cout << 5 \* num << endl;

}

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

{

x = 0;

y = 0;

str = "";

}

27. 7, 0, 0

1, 0, 8

8, 1, 8

2, 1, 1

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 trackVar(double& x, double y, double& z)

{

z = floor(x) + ceil(y);

x = x + z;

y = y - z;

}

33. 3 5

108 0

108 5

35. stVar = 3, u = 3, x = 2

stVar = 9, u = 3, x = 3

stVar = 18, u = 3, x = 4

stVar = 36, u = 3, x = 5

37. a, b, and d 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; \

3. Only a and c are valid.

5.

flowerType readIn()

{

string str;

flowerType flower = 0;

cin >> str;

if (str == "Rose")

flower = ROSE;

else if (str == "Daisy")

flower = DAISY;

else if (str == "Carnation")

flower == CARNATION;

else if (str == "Freesia")

flower = FREESIA;

else if (str == "Gardenia")

flower = GARDENIA;

else if (str == "Allium")

flower = ALLIUM;

else if (str == "Tulip")

flower = TULIP;

else if (str == "Iris")

flower = IRIS;

else if (str == "sunflower")

flower = SUNFLOWER;

else if (str == "Lilac")

bir flower d == LILAC;

else if (str == "Orchid")

flower = ORCHID;

else

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

return flower;

}

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 Line1 1 and 2 should be:

#include <iostream> //Line 1

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. Sammer Vucation

b. Temperary Projoct

c. Nocial Setwork

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 64. first = 0, middle = 32, last = 64

7. 0.00 1.50 9.00 28.50 66.00

57.00 1.50 30.00 28.50 66.00

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. alpha[0] = 3, alpha [1] = 12, alpha [2] = -25, alpha [3] = 72,

alpha [4] = 0.

19. -5 0 10 60 360 600

21. a. Correct.

b. Correct.

c. Incorrect. None of the formal parameters list and sList are of type double while the actual parameter unitPrice is of type double. So there will be mismatch data type error.

d. Incorrect. The size of the array ids is 50, so the call should be printList(ids, 50);

e. Correct.

23. 1 35700.00 714.00

2 96800.00 1936.00

3 55000.00 1100.00

4 72500.00 1450.00

5 87700.00 1754.00

25. list: 810 0 270 180 90

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. 1 0 1 1 1 0 0 1 1

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

35. a. Valid

b. Valid

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

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

37. a. strcpy(myStr, "Summer Vacation");

b. cout << strlen(yourStr) << endl;

c. strcpy(myStr, yourStr);

d. compare = strcmp(myStr, yourStr);

39. double matrix[4][3] = {{2.5, 3.2, 6.0}, {5.5, 7.5, 12.6},

{11.25, 16.85, 13.45}, {8.75, 35.65, 19.45}};

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. computerType newComputer;

newComputer.manufacturer = "Computer Corporation";

newComputer.modelType = "Desk Top";

newComputer.processorType = "Core I 7";

newComputer.ram = 12;

newComputer.hardDriveSize = 500;

newComputer.yearBuilt = 2016;

newComputer.price = 850.00;

5. if (firstHouse.style == secondHouse.style &&

firstHouse.price == secondHouse.price)

cout << "true" << endl;

else

cout << "false" << endl;

7. fruitType fruit;

fruit.name = "banana";

fruit.color = "yellow";

fruit.fat = 1;

fruit.sugar = 15;

fruit.carbohydrate = 22;

9. Assignment statement and function return value.

11. a. classList[0].name.first = "Jessica";

classList[0].name.last = "Miller";

classList[0].gpa = 3.8;

classList[0].course.name = "Data Structure";

classList[0].course.callNum = 8340;

classList[0].course.credits = 3;

classList[0].course.grade = 'B';

b. student = classList[0];

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. sportsType soccer[20];

struct sportsType

{

string sportName;

string teamName;

int numberOfPlayers;

double teamPayroll;

double coachSalary;

};

17. a. void getData(sportsType & sp)

{

cin >> sp[i].sportName >> sp[i].teamName

>> sp[i].numberOfPlayers

>> sp[i].teamPayroll

>> sp[i].coachSalary;

}

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

getData(soccer[i]);

b. void printData(sportsType sp)

{

cout << "Sport Name: " << sp[i].sportName << endl;

cout << "Team Name: " << sp[i].teamName << endl;

cout << "Number of Players: " << sp[i].numberOfPlayers << endl;

cout << "Team Payroll: $" << sp[i].teamPayroll << endl;

cout << "Coach Salary: $" << sp[i].coachSalary << endl;

cout << "-----------------------------------" << endl << endl;

}

for (int j = 0; j < 20; j++)

printData(soccer[i]);

Chapter 10

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

3. A constructor has no type. The statements in Line 6 should be:

syntaxErrors2(int = 0,

double = 0); //Line 6

Also, replace ; after private with :. Line 7 should be:

private: //Line 7

5. The function set must have a return type. A constructor cannot be constant. Replace : after } with ;. The statements in Lines 4, 6, and 12 should be:

void set(string, int, double); //Line 4

syntaxErrors4() const; //Line 6

}; //Line 12

7. a.

void foodType::set(string s, int c, double f, int su,

double cr, double p)

{

name = s;

if (c >= 0)

calories = c;

else

calories = 0;

if (f >= 0)

fat = f;

else

fat = 0;

if (su >= 0)

sugar = su;

else

sugar = 0;

if (cr >= 0)

carbohydrate = cr;

else

carbohydrate = 0;

if (p >= 0)

potassium = p;

else

potassium = 0;

}

b.

void foodType::print() const

{

cout << "Name: " << name << endl;

cout << "Calories: " << calories << endl;

cout << "Fat: " << fat << endl;

cout << "Sugar: " << sugar << endl;

cout << "Carbohydrate: " << carbohydrate << endl;

cout << "potassium: " << potassium << endl;

}

c.

string foodType::getName() const

{

return name;

}

int foodType::getCalories() const

{

return calories;

}

double foodType::getFat() const

{

return fat;

}

int foodType::getSugar() const

{

return sugar;

}

double foodType::getCarbohydrate() const

{

return carbohydrate;

}

double foodType::getPotassium() const

{

return potassium;

}

d.

foodType::foodType()

{

set("", 0, 0.0, 0, 0.0, 0.0);

}

e.

foodType::foodType(string s, int c, double f, int su,

double cr, double p)

{

set(s, c, f, su, cr, p);

}

f.

fruit2.print();

g.

foodType myFruit("Apple ", 52, 0.2, 10, 13.8, 148.0);

9. The functions print, getQuantitiesInStock, getPrice, and getDiscount are accessors; functions set, setQuantitiesInStock, updateQuantitiesInStock, setPrice, and setDiscount 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.

10:17:00

23:59:29

00:00:29

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. It typically inserts the code of an inline function at every location the function is called.

29.

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 it 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 shoe and the derived class is runningShoe.

b. This is private inheritance.

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

9. Omit the word class before employee. The first statement should be:

class hourlyEmployee: public employee

In the third line replace :: with :. This statement should be

public:

Omit the word const from the prototypes of the functions setHoursWorked and setPay because these functions modify the instance variables. These prototypes should be:

void setHoursWorked(double hrsWk);

void setPay();

Replace ; after the label private with :. This statement should be

private:

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 private members of the derived class. They can be directly accessed in the derived class. The protected members of the base class are private 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 public 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 protected 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. a. class yourClass: protected base

{

}

b. The members setXYZ, setX, getX, setY, getY, mystryNum, and print, z, setZ, and secret are protected members of the class yourClass. The private members x and y of the class base are hidden in class yourClass and they can be accessed in class yourClass only through the protected and public members of class base.

27. a. Because the memberAccessSpecifier is not specified, it is a private inheritance.

b. All members of the class base becomes private members in class derived.

29. a.

void base::print() const

{

cout << "num = " << num << ", x = " << x;

}

double base::compute(int n)

{

return n + manipulate(n, n);

}

double base::manipulate(int a, int b)

{

return num \* a + x \* b;

}

b.

void derived::print() const

{

base::print();

cout << ", z = " << z;

}

double derived::compute(int a, double b)

{

return base::compute(a) + z \* manipulate(0, b);

}

c.

num = 2, x = 5.50

59.50

num = 3, x = 1.50, z = 2.00

17.50

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. 33.8 3.8

33.8 3.8

11. The correct code is:

double \*length;

double \*width;

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

length = new double;

\*length = 6.5;

width = new double;

\*width = 3.0;

cout << "Area: " << (\*length) \* (\*width) << ", ";

cout << "Perimeter: " << 2 \* (\*length + \*width) << endl;

Output:

Area: 19.50, Perimeter: 19.00

13. Trip total cost: $550.00

Highest trip cost: $275.00

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 37 78 62 62 13

19. numPtr = 1058 and gpaPtr = 2024

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

23. a. sales = new double[50];

b. for (int i = 0; i < 50; i++)

cin >> sales[i];

c. int maxIndex = 0;

for (int i = 1; i < 50; i++)

if (sales[maxIndex] < sales[i])

maxIndex = i;

d. delete []sales;

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. int \*myList;

int \*yourList;

myList = new int[5];

myList[0] = 8;

for (int i = 1; i < 5; i++)

myList[i] = i \* myList[i - 1];

yourList = new int[5];

for (int i = 0; i < 5; i++)

yourList[i] = 2 \* myList[i];

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. 5

small: --

x: 2, y = 3

\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*

17

small: --

x: 3, y = 5

noSmall--- z: 9

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. b

5. A friend function of a class is a nonmember function of the class, but has access to all the members (public or non-public) of the class.

7. d; Because the left operand of << is a stream object, which is not of the user-defined class type.

9. a. One b. Two

11. object1.operator+(object2)

13. a. bool b. bool

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

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

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

17. In Line 4, the formal parameter of the function operator+ should be of type myClass. The correct statement is:

myClass operator+(const myClass& obj); //Line 4

19. 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

21. In Line 3 and 10, the return type of the function operator should be findErrors. In Line3, the type of the objects a and b must be findErrors. 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 10, is not needed. In Lines 13 and 14, to access the instance variables of the object a, we need to use the object a and the dot operator. The correct statements are:

friend findErrors operator\*(const findErrors& a,

const findErrors& b); //Line 3

double operator\*(const findErrors& a,

const findErrors& b) //Line 10

temp.first = a.first \* b.first; //Line 13

temp.second = b.second \* b.second; //Line 14

23. A reference to an object of the class istream.

25. The function that overloads the pre increment operator has no parameter, while the function that overloads the post increment operator has one (dummy) parameter.

27. a. None b. 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;

}

39. These statements generate and output a random integer between 10 and 25.

**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.At most one.

7. The thrown value then may not be accessible in the catch block exception handling code.

9. The object being thrown can be either a specific object or an anonymous object.

11. The cout statement in Line 12 separates the catch block from the try block. Therefore, the catch block has no associated try block and the try block has no associated catch block. The catch block in Line 13 has no parameters. The correct code is:

double salary = 78000; //Line 1

double raise; //Line 2

try //Line 3

{ //Line 4

cout << "Enter the raise: "; //Line 5

cin >> raise; //Line 6

cout << endl; //Line 7

if (raise < 0.0) //Line 8

throw raise; //Line 9

cout << "Salary increase: $"

<< salary \* raise / 100 << endl; //Line 10

cout << "Exiting the try block." << endl; //Line 11

} //Line 12

catch (double x) //Line 13

{ //Line 14

cout << "Negative raise: " << x << endl; //Line 15

} //Line 16

(In a and b, the user input is shaded.)

a. Enter the raise: 5

Salary increase: $3900

Exiting the try block.

b. Enter the raise: -4

Negative raise: -4

13. (In the following, the user input is shaded.)

a. Enter the number of items: 25

Enter the cost of one item: 5.50

Total cost: $137.50

b. Enter the number of items: -55

Negative number of items: -55

Numer of items must be nonnegative.

c. Enter the number of items: 37

Enter the cost of one item: -4.5

Negative unit cost: -4.50

Unit cost must be nonnegative.

d. Enter the number of items: -10

Negative number of items: -10

Numer of items must be nonnegative.

15. a. 55

Exiting the try block.

b. Exception: Division by 0

c. Exception: Total score is out of range.

d. Exception: Division by 0

17. a. class out\_of\_range

b. class length\_error

c. class runtime\_error

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. A definition in which something is defined in terms of a smaller version of itself.

5. Because a base case stops the recursion.

7. a. The statements from Line 3 to Line 6.

b. The statements in Lines 7 and 8.

c. It is a valid call. The value of recFunc(58) is 32.

d. It is a valid call. The value of recFunc(-24) is 24.

e. It is a valid call. The value of recFunc(0) is 0.

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

11. a. 4 12 28

b. 5 15 34 72 148

c. 2 8 21 47 98

d. It does not produce any output.

13. a. 0

b. 4

c. 8

d. 162

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.

19. A selection control structure.

Chapter 16

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

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

b. true

c. false

d. true

e. true

f. false

9. a. p->link->info = 24;

b. q = current->link;

c. first = first->link;

d. trail = p->link;

e. p = nullptr;

f. temp->link->info = 54;

g. while (first->info != 5)

first = first ->link;

11. a. while (first != nullptr)

first = first->link;

b. q = new nodeType;

q->info = 17;

q->link = current->link;

current->link = q;

c. q = temp->link;

q->link = nullptr;

delete last;

last = q;

d. q = p->link;

p->link = current;

delete q;

e. q = current->link;

q->link = temp->link;

temp->link = q;

current->link = temp;

13. 65 5 78

15. 39 26 78

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.

90 15 65 36 30 27

23.



25.



Chapter 17

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

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

3. a. 8 b. 7 c. dec = stack.top(); d. stack.pop();

5. 13

32 32 13 16 28

temp = 16

7. secretNum = 226

9. a. 16

b. -4

c. 39

d. 12

e. 15

11. a. x \* y + z - t

b. x \* (y + z) - w / u

c. (x - y) \* (z / u) - (t + s)

d. 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. a. 4

b. 21

c.!queue.isEmptyQueue()

d. queue.addQueue("programming")

After the insertion operation the index of the last element is 5

21. cin >> num;

while (cin)

{

switch (num % 2)

{

case 0:

stack.push(num);

break;

case 1: case -1:

if (num % 3 == 0)

queue.addQueue(num);

else

{

if (!stack.isEmptyStack())

stack.pop();

stack.push(num \* num);

}

} //end switch

cin >> num;

} //end while

After processing these numbers, stack and queue are:

stack: 14 289 10 121 28

queue: 15 -9 21 -3 33

23. a. 26

b. queueFront = 35; queueRear = 61.

c. queueFront = 36; queueRear = 60.

25. a. 31

b. queueFront = 25; queueRear = 56.

c. queueFront = 26; queueRear = 55.

27. 51

29. 5 -4 5 -7 1 2 1 4 1 -2 2 -7 7 -6

31. 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);

}

}

33. template <class Type>

int queueType<Type>::queueCount()

{

return count;

}

35.



Chapter 18

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

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

3. a.

template<class elemType>

int seqOrdSearch(const elemType list[], int length,

const elemType& item)

{

int loc;

bool found = false;

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

if (list[loc] >= item)

{

found = true;

break;

}

if (found)

if (list[loc] == item)

return loc;

else

return -1;

else

return -1;

} //end seqOrdSearch

b. i. 2 ii. 8 iii. 10 iv. 9 v. 13

5. a. 6 b. 7 c. 8 d. 7 e. 1 f. 3 g. 8

7. 26

9. 9

11. a. Suppose that the list is of length *n*. Then length = *n*. Consider the first iteration of the outer for loop, that is, when the value of iteration = 1. Now the inner loop executes *n* – 1. Before the execution of the inner for loop, the variable isSorted is set to true, assuming that the list is sorted. If the list is already sorted, then the expression list[index] > list[index + 1] in the if statement always evaluates to false, so the body of the if statement never executes. Because the inner loop executes *n* – 1 times, there are *n* – 1 comparisons. In the second iteration of the outer loop, because the variable isSorted is true, the loop condition, (iteration < length) && !isSorted, evaluates to false, so the outer for loop terminates. It follows that, if the list is already sorted, the outer for loop executes only once. Hence, the total number of comparisons is *n* – 1 = *O*(*n*).

b. 30.

13. 4

15. a. 8, 12, 18, 25, 38, 45, 74, 60, 30

b. 10

17. Bubble sort: 21,121,750; selection sort: 21,121,750; insertion sort: 10,567,374

19. In quick sort, the list is partitioned according to an element, called pivot, of the list. After partition, elements in the first sublist are smaller than the pivot and in the second sublist are larger than the pivot. The merge sort partitions the list by dividing into two sublists of nearly equal size by breaking the list in the middle.

21. a. 48

b. 10, 30, 35, 8, 15, 48, 55, 58, 60, 70, 78, 75, 80, 62, 90

c. 18

d. 5 and 2

23. a. 48

b. 38, 18, 13, 25, 40, 44, 32, 41, 48, 75, 80, 90, 70, 65, 60, 52

c. 27

d. 8 and 7

25.a.8 b. 38 c. 84

27. Quick sort: 10000×log2(10000) = 132878; merge sort: 132878.

**HeapSort-Algorithm**

1. 95, 92, 87, 65, 80, 82, 81, 59, 52, 78, 50, 53, 63, 47, 58, 42, 34, 37, 7, 20

**Chapter 19**

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

3. *LA* = {*B*, *D*, *E*, *H*, *I*, *L*, *M*, *O*, *P*, *Q*, *T*, *U*}.

5. *LC*= {*F*, *J*}.

7. *LE* = { *L*, *O*, *T*, *U* }.

9. 4

11. 6

13. *D*, *E*, *F*, *G*, *K*, *L*.

15. 4

17. *A*, *B*, *D*, *H*, *M*, and *P*

19. *A*, *B*, *D*, *H*, *L*, *O*, *T*, *U*, *M*, *P*, *Q*, *E*, *I*, *C*, *F*, *J*, *G*, *K*, *N*, *R*, *S*

21. a. 7 b. 4 c. 92-82-72-75-80-78

23.Binary search tree after deleting 45. After deleting 45, the height of the tree is 7.



25. The height of the tree after deleting 70 is 7. Binary search tree after deleting 70.



27. Each node has two pointers. Thus, there are 2*n* pointers in all the nodes of the binary tree. The pointer pointing to the root node is an external pointer. Each of the remaining *n* – 1 nodes has one pointer pointing to it, which is a pointer from the parent node. The remaining pointers in all the nodes, which is 2*n* – (*n* – 1) = *n* + 1, are nullptr. This implies that the binary tree has *n* + 1 empty subtrees.

29.



31.



33.



**AVL-Trees**

1. The balance factor of the root node is 0.



3 The balance factor of the root node is 0.



Chapter 20

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

3. 4-7-8-6-3-10; Length of this path is 5.

5. Vertices 2 and 9 are not connected.

7. Vertices 0 and 8 are connected. A path from 0 to 8 is 0-6-11-4-2-9-8

9. 1-2-9-8-3

11. 

13. 0, 1, 4, 2, 5, 6, 3, 7, 8, 9

15. 0, 1, 2, 5, 7, 3, 6, 10, 11, 9, 4, 8

17.



19.



21.



Source Vertex: 0

Edges Weight

(0, 1) 2

(1, 2) 3

(6, 3) 1

(1, 4) 2

(7, 5) 5

(4, 6) 4

(0, 7) 2

(6, 8) 5

Minimal Spanning Tree Weight: 24

**Topological Ordering**

1. 1, 7, 0, 2, 6, 5, 4, 8, 3, 9

Chapter 21

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

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

3. A container is used to store data, while an algorithm is used to manipulate the data stored in a container.

5. A STL function object contains a function that can be treated as a function using the function call operator.

7. 66 2 15 99 37 13 87 18 35 46 32

32 46 35 18 46 35 18 87 13 37 99

9. 0 0 8 12 24

11. a. intList1 = {7, 8, 19, 90, 15, 13, 15, 16, 18,

90, 30, 11, 88, 26}

b. intList1 = {12, 14, 20, 13, 15, 16, 18, 90, 30, 11, 88, 26}

13. A back\_inserter uses the push\_back operation of the container while a front\_inserter uses the push\_front operation of the container to add elements to the container. Furthermore, a front\_inserter cannot be used for the vector container.

15. 5

17. + - = @ ! 1 2 3 4 5

@ ! 1 2 - = + 3 4 5

19. 22 6 52 2 30 24 35 18

21. 9