APPENDIX M Solutions to Odd Numbered Review Questions

Chapter 1

1. Main memory, or RAM, is volatile, which means its contents are erased when power is removed from the computer. Secondary memory, such as a disk, does not lose its contents when power is removed from the computer.

3. An operating system

5. Because high level languages are more like natural language.

7. A syntax error is the misuse of a key word, operator, punctuation, or other part of the programming language. A logical error is a mistake that causes the program to produce the wrong results.

9. CPU

11. disk

13. instructions

15. machine language

17. low-level

19. key words

21. operators

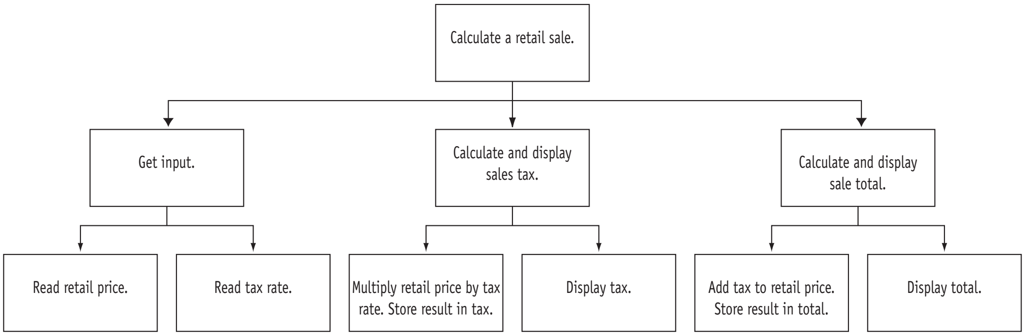
23. syntax

25. defined

27. input

29. hierarchy chart

31. Hierarchy chart:


Calculate a retail sale.
Get input.
Calculate and display sales tax.
Calculate and display sale total.
Read retail price.
Read tax rate.
Multiply retail price by tax rate. Store result in tax.
Display tax.
Add tax to retail price. Store result in total.
Display total.

33. 7

35. 365

Chapter 2

1. 1, 2, 3

3. `int months = 2, days, years = 3;`

5. Multi-line comment

7.
```
#include <iostream>
int main()
{
     cout << "Two mandolins like creatures in the\n\n\n";
     cout << "dark\n\n\n";
     cout << "Creating the agony of ecstasy.\n\n\n";
     cout << " - George Barker\n\n\n";
     return 0;
}
```

9. C

11. B

13. B

15. B, C

17. A) 12
B) 4
C) 2
D) 6
E) 1

19. A

21. A

23. False

25. True

27.
```
int speed, time, distance;
speed = 20;
time = 10;
distance = speed * time;
cout << distance << endl;
```

29. The C-style comments symbols are backwards.
`iostream` should be enclosed in angle brackets.
There shouldn't be a semicolon after `int main`.
The opening and closing braces of function main are reversed.
There should be a semicolon after `int a, b, c`.
The comment `\\ Three integers` should read `// Three integers`.
There should be a semicolon at the end of the following lines:

```
a = 3
b = 4
c = a + b
```

`cout` begins with a capital letter.
The stream insertion operator (that appears twice in the `cout` statement) should read << instead of <.
The `cout` statement uses the variable `C` instead of `c`

Chapter 3

1. `cin >> age >> pay >> section;`

3. `iostream` and `iomanip`

5.
```
a = 12 * x;
z = 5 * x + 14 * y + 6 * k;
y = pow(x, 4);
g = (h + 12) / (4 * k);
c = pow(a, 3) / (pow(b, 2) * pow(k, 4));
```

7. B

9. `const int RATE = 12;`

11. `east = west = north = south = 1;`

13.
```
cout << setw(12) << fixed
     << setprecision(4) << totalAge;
```

15. `cos`

17. `tan`

19. `fmod`

21. `log10`

23. `sqrt`

25.

```
Display “Enter the customer’s maximum amount of credit: ”.
Read maxCredit.
Display “Enter the amount of credit the customer has used: ”.
Read creditUsed.
availableCredit = maxCredit – creditUsed.
Display “The customer’s available credit is $”.
Display availableCredit.
```

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

int main()
{
    double maxCredit, creditUsed, availableCredit;

    cout << "Enter the customer's maximum amount of credit: ";
    cin >> maxCredit;
    cout << "Enter the amount of credit used by the customer: ";
    cin >> creditUsed;
    availableCredit = maxCredit – creditUsed;
    cout << "The customer's available credit is $";
    cout << availableCredit << endl;
    return 0;
}
```

27.

```
Display “Enter the score for the 1st game: ”.
Read score1.
Display “Enter the score for the 2nd game: ”.
Read score2.
Display “Enter the score for the 3rd game: ”.
Read score3.
averageScore = (score1 + score2 + score3) / 3.
Display “The average score is : ”.
Display averageScore.
```

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

int main()
{
    int score1, score2, score3, averageScore;

    cout << "Enter the score for the 1st game: ";
    cin >> score1;
    cout << "Enter the score for the 2nd game: ";
    cin >> score2;
    cout << "Enter the score for the 3rd game: ";
    cin >> score3;
```

```
        averageScore = (score1 + score2 + score3) / 3;
        cout << "The average score is :";
        cout << averageScore << endl;
        return 0;
    }
```

29. The first `cin` statement should read:

    ```
    cin >> number1 >> number2;
    ```

    The assignment statement should read:

    ```
    quotient = static_cast<float>(number1) / number2;
    ```

    The last statement is missing a semicolon.

31. There shouldn't be a semicolon after the `#include` directive.
    The function header for `main` should read:

    ```
    int main()
    ```

    The combined assignment operators improperly used.
    Those statements should be:

    ```
    number1 *= 50;
    number2 *= 50;
    ```

33. There shouldn't be a semicolon after the `#include` directive.
    `name` should be declared as a `string`, and the `#include <string>` directive should be used.
    The following statement:

    ```
    getline >> name;
    ```

    should read:

    ```
    getline(cin, name);
    ```

35. 6 3 12

37. Minutes: 612002.0000
    Hours: 10200.0332
    Days: 425.0014
    Months: 13.9726
    Years: 1.1644

Chapter 4

1. In an `if/else if` statement, the conditions are tested until one is found to be true. The conditionally executed statement(s) are executed and the program exits the `if/else if` statement. In a series of `if` statements, all of the `if` statements execute and test their conditions because they are not connected.

3. A flag is a Boolean variable signaling that some condition exists in the program. When the flag is set to `false` it indicates the condition does not yet exist. When the flag is set to `true` it indicates that the condition does exist.

5. It takes two expressions as operands and creates a single expression that is true only when both subexpressions are true.
7. Because they test for specific relationships between items. The relationships are greater-than, less-than, equal-to, greater-than or equal-to, less-than or equal-to, and not equal-to.
9. relational
11. False, True
13. True
15. True, False
17. nested
19. ||
21. left-to-right
23. ||
25. >
27. integer
29. break
31.
    ```
    if (y == 0)
    x = 100;
    ```
33.
    ```
    if (sales < 10000)
        commission = .10;
    else if (sales <= 15000)
        commission = .15;
    else
        commission = .20;
    ```
35.
    ```
    if (amount1 > 10)
       if (amount2 < 100)
           cout << (amount1 > amount2 ? amount1 : amount2);
    ```
37.
    ```
    if (temperature >= -50 && temperature <= 150)
        cout << "The number is valid.";
    ```
39.
    ```
    if (str1 > str2)
        cout << str1;
    else
        cout << str2;
    ```
41. C, A, B
43. False
45. True
47. True

49. True

51. True

53. False

55. F

57. T

59. The conditionally executed blocks in the `if/else` construct should be enclosed in braces.
The following statement:

```
cout << "The quotient of " << num1 <<
```

should read:

```
cout << "quotient of " << num1;
```

61. A `switch` statement cannot be used to test relational expressions. An `if/else if` statement should be used instead.

63. It should use `&&` instead of `||`.

65. The `:` and `?` are transposed. The statement should read:

```
    z = (a < 10) ? 0 : 7;
```

Chapter 5

1. By indenting the statements, you make them stand out from the surrounding code. This helps you to identify at a glance the statements that are conditionally executed by a loop.

3. Because they are only executed when a condition is true.

5. The `while` loop.

7. The `for` loop.

9. An accumulator is used to keep a running total of numbers. In a loop, a value is usually added to the current value of the accumulator. If it is not properly initialized, it will not contain the correct total.

11. `fstream`

13. `ifstream`

15. A file's read position marks the location of the next byte that will be read from the file. When an input file is opened, its read position is initially set to the first byte in the file.

17. prefix

19. body

21. pretest

23. infinite or endless

25. running total

27. sentinel

29. `while` and `for`

31. initialization, test, update

33. `break`

35.

```
int product = 0, num;
while (product < 100)
{
    cin >> num;
    product = num * 10;
}
```

37.

```
for (int x = 0; x <= 1000; x += 10)
     cout << x;
```

39.

```
for (int row = 0; row < 10; row++)
{
    for (int col = 0; col < 15; col++)
         cout << '#';
    cout << endl;
}
```

41.

```
char sure = 'x';
while (sure != 'Y' && sure != 'N')
{
    cout << "Are you sure you want quit? "
    cin >> sure;
}
```

43.

```
int x = 50;
while (x > 0)
{
    cout << x << " seconds to go.\n";
    x--;
}
```

45.

```
ifstream inputFile("Numbers.txt");
int number;
while (inputFile >> number)
    cout << number << endl;
inputFile.close();
```

47. false

49. false

51. false

53. false

55. false

57. true

59. true

61. false

63. true

65. The statement `result = ++(num1 + num2);` is invalid.

67. The `while` statement should not end with a semicolon.
It could also be argued that `bigNum` should be defined a `long`.
`count` should be initialized to 1.

69. The expression tested by the `do-while` loop should be `choice == 1` instead of `choice = 1`.

Chapter 6

1. Because they are created in memory when the function begins execution, and are destroyed when the function ends.

3. Inside the parentheses of a function header.

5. Yes. The first argument is passed into the parameter variable that appears first inside the function header's parentheses. Likewise, the second argument is passed into the second parameter, and so on.

7. It makes the program easier to manage. Imagine a book that has a thousand pages, but isn't divided into chapters or sections. Trying to find a single topic in the book would be very difficult. Real-world programs can easily have thousands of lines of code, and unless they are modularized, they can be very difficult to modify and maintain.

9. A function such as the following could be written to get user input. The input is stored in the variables that are passed as arguments.

```
void getValues(int &x, int &y)
{
    cout << "Enter a number: ";
    cin >> x;
    cout << "Enter another number: ";
    cin >> y;
}
```

11. `void`

13. arguments

15. value

17. local

19. global

21. local

23. `return`

25. last

27. reference

29. reference

31. parameter lists

33.

```
double half(double num)
{
    return num / 2;
}
```

35.

```
void timesTen(int num)
{
    cout << (num * 10) << endl;
}
```

37.

```
void getNumber(int &num)
{
    cout << "Enter a number in the range 1 – 100 : ";
    cin >> num;
    while (num < 1 || num > 100)
    {
        cout << "That number is out of range.\n";
        cout << "Enter a number in the range 1 – 100 : ";
        cin >> num;
    }
}
```

39. False

41. True

43. True

45. False

47. True

49. True

51. True

53. False

55. True

57. The assignment statement should read:

```
average = (value1 + value2 + value3) / 3.0;
```

The function is defined as a `double` but returns no value.

59. The parameter should be defined as:

```
int &value
```

The `cin` statement should read:

```
cin >> value;
```

Chapter 7

1. The size declarator is used in a definition of an array to indicate the number of elements the array will have. A subscript is used to access a specific element in an array.

3. Because, with the array alone the function has no way of determining the number of elements it has.

5. By providing an initialization list. The array is sized to hold the number of values in the list.

7. Because an array name without brackets and a subscript represents the array's beginning memory address. The statement shown attempts to assign the address of `array2` to `array1`, which is not permitted.

9. By reference.

11. By using the same subscript value for each array.

13. The second size declarator, which is for the number of columns.

15. size declarator

17. subscript

19. size declarator, subscript

21. initialization

23. initialization list

25. =

27. address, or name

29. rows, columns

31. braces

33. Standard Template Library (or STL)

35. sequence

37. `push_back`

39. `pop_back`

41.
```
for (int i = 0; i < 20; i++)
   cout << names[i] << endl;
```

43.
```
const int SIZE = 10;
int id[SIZE];              // To hold ID numbers
double weeklyPay[SIZE];    // To hold weekly pay
// Display each employee's gross weekly pay.
for (int i = 0; i < SIZE; i++)
{
    cout << "The pay for employee "
         << id[i] << " is $" << fixed
         << showpoint << setprecision(2)
         << weeklyPay[i] << endl;
}
```

45.
```
const int SIZE = 12;
// A 2D array to hold the country names
string countries[SIZE];
// An array to hold populations
long populations[SIZE];
// Display each country's name and population.
for (int i = 0; i < SIZE; i++)
{
    cout << "The population of " << countries[i]
         << " is " << populations[i] << endl;
}
```

47.
```
numberArray[0][0] = 145;
numberArray[8][10] = 18;
```

49.
```
const int NUM_ROWS = 29;
const int NUM_COLS = 5;
int row, col,    // Loop counters
    total;       // Accumulator
// Display the sum of each row.
for (row = 0; row < NUM_ROWS; row++)
{
    // Set the accumulator.
    total = 0;
    // Sum a row.
    for (col = 0; col < NUM_COLS; col++)
         total += days[row][col];
    // Display the row's total.
    cout << "The total for row " << row
         << " is " << total << endl;
}
// Display the sum of each column.
for (col = 0; col < NUM_COLS; col++)
{
    // Set the accumulator.
    total = 0;
    // Sum a column.
    for (row = 0; row < NUM_ROWS; row++)
         total += days[row][col];
    // Display the column's total.
    cout << "The total for column "
         << col << " is " << total << endl;
}
```

51. True

53. False

55. False

57. True

59. True

61. False

63. False

65. True

67. True

69. True

71. True

73. False

75. False

77. True

79. True

81. The size declarator cannot be negative.

83. The initialization list must be enclosed in braces.

85. For the array to be implicitly sized there must be an initialization list.

87. The assignment operator cannot be used to assign the contents of one array to another, in a single statement.

89. The parameter must specify the number of columns, not the number of rows.

Chapter 8

1. Because it uses a loop to sequentially step through an array, starting with the first element. It compares each element with the value being searched for, and stops when either the value is found or the end of the array is encountered.

3. N/2 times

5. Ten

7. The selection sort usually performs fewer exchanges because it moves items immediately to their final position in the array.

9. binary
11. binary
13. descending
15. False
17. False

Chapter 9

1. It dereferences a pointer, allowing code to work with the value that the pointer points to.
3. Multiplication operator, definition of a pointer variable, and the indirection operator.
5. It adds 4 times the size of an `int` to the address stored in `ptr`.
7. To dynamically allocate memory.
9. To free memory that has been dynamically allocated with the `new` operator.
11. A pointer to a constant points to a constant item. The data that the pointer points to cannot change, but the pointer itself can change. With a constant pointer, it is the pointer itself that is constant. Once the pointer is initialized with an address, it cannot point to anything else.
13. address
15. pointer
17. pointers
19. `new`
21. null
23. `new`
25. `*(set + 7) = 99;`
27. `delete [] tempNumbers;`
29. `const int *ptr;`
31. True
33. True
35. False
37. False
39. True
41. False
43. True
45. False

47. False

49. The assignment statement should read `ptr = &x;`

51. The assignment statement should read `*ptr = 100;`

53. Multiplication cannot be performed on pointers.

55. `iptr` cannot be initialized with the address of `ivalue`. `ivalue` is defined after `iptr`.

57. The second statement should read `pint = new int;`

59. The last line should read `delete [] pint;`

61. The pointer definition should read:

```
const int *ptr = array;
```

Chapter 10

1. `cctype`

3. ‘A’
   ‘B’
   ‘d’
   ‘E’

5. `cstring`

7. `string`

9. `isupper`

11. `isdigit`

13. `toupper`

15. `cctype`

17. concatenate

19. `strcpy`

21. `strcmp`

23. `atoi`

25. `atof`

27. `if (toupper(choice) == 'Y')`

29.

```
if (strlen(name) <= 9)
     strcpy(str, name);
```

31.

```
int wCount(char *str)
{
     int num = 0;
     while (*str != '\0')
     {
          if (*str == 'w')
```

```
                num++;
        }
        return num;
}
```

33. False

35. False

37. True

39. False

41. True

43. The isupper function can only be used to test a character, not a string.

45. The compiler will not allocate enough space in string1 to accommodate both strings.

Chapter 11

1. A data type that is built into the C++ language, such as int, char, float, etc.

3. The elements of an array must all be of the same data type. The members of a structure may be of different data types.

5. A) `FullName info;`

   B)
   ```
   info.lastName = "Smith";
   info.middleName = "Bart";
   info.firstName = "William";
   ```

   C)
   ```
   cout << Info.lastName << endl;
   cout << info.middleName << endl;
   cout << info.firstName << endl;
   ```

7. A) “Canton”
   B) “Haywood”
   C) 9478
   D) uninitialized

9. All the members of a union occupy the same area of memory, whereas the members of a structure have their own memory locations.

11. 0 1 2

13. declared

15. members

17. tag

19. `Car hotRod = {"Ford", "Mustang", 1997, 20000};`

21.
```
Car forSale[35] = {{"Ford", "Taurus", 1997, 21000},
                   {"Honda", "Accord", 1992, 11000},
                   {"Lamborghini", "Countach", 1997, 200000}};
```

23.

```
struct TempScale
{
    double fahrenheit;
    double centigrade;
};
struct Reading
{
    int windSpeed;
    double humidity;
    tempScale temperature;
};
Reading today;
```

25.

```
void showReading(Reading values)
{
    cout << "Wind speed: " << values.windSpeed << endl;
    cout << "Humidity: " << values.humidity << endl;
    cout << "Fahrenheit temperature: " <<
            values.temperature.fahrenheit << endl;
    cout << "Centigrade temperature: " <<
            values.temperature.centigrade << endl;
}
```

27.

```
Reading getReading()
{
    Reading local;

    cout << "Enter the following values:\n";
    cout << "Wind speed: ";
    cin >> local.windSpeed;
    cout << "Humidity: ";
    cin >> local.humidity;
    cout << "Fahrenheit temperature: ";
    cin >> local.temperature.fahrenheit;
    cout << "Centigrade temperature: ";
    cin >> local.temperature.centigrade;
    return local;
}
```

29.

```
rptr->WindSpeed = 50;
```

31.

```
union Items
{
    char alpha;
    int num;
    long bigNum;
    float real;
};
Items x;
```

33.

```
num = 452;
```

35. `enum Pets{DOGS, CATS, BIRDS, HAMSTERS};`
37. True
39. False
41. False
43. True
45. True
47. True
49. False
51. False
53. True
55. True
57. The structure declaration has no tag.
59. No structure variable has been declared. `TwoVals` is the structure tag.
61. The initialization list of the `customer` variable must be enclosed in braces.
63. Structure members cannot be initialized in the structure definition.
65. The function must define a variable of the `TwoVals` structure. The variable, then, should be used in the assignment statement.
67. Both `x` and `y` cannot be meaningfully used at the same time.

Chapter 12

1. The `fstream` data type allows both reading and writing, while the `ifstream` data type allows only for reading, and the `ofstream` data type allows only for writing.
3. Its contents are erased. (In other words, the file is truncated.)
5. By reference because the internal state of file stream objects changes with most every operation. They should always be passed to functions by reference to ensure internal consistency.
7. When the end of the file has been encountered. The `eof` member function reports the state of this bit.
9. By using the `getline` member function.
11. Two arguments: The starting address of the section of memory where the data will be stored, and the number of bytes to read.
13. The `seekg` function moves a file's write position, and the `seekp` function moves a file's read position.
15. Call the file object's `clear` member function.

17. Use the `seekg` member function to move the read position back to the beginning of the file.

19. NULL or 0

21. `getline`

23. `put`

25. text, ASCII text

27. structures

29. `read`

31. sequential

33. `seekg`

35. `tellg`

37. `ios::beg`

39. `ios::cur`

41.

```
fstream places("places.dat", ios::in | ios::out);
```

43.

```
pets.open("pets.dat", ios::in);
fstream pets("pets.dat" ios::in);
```

45.

```
fstream employees;
employees.open("emp.dat", ios::in | ios::out | ios::binary);
if (!employees)
    cout << "Failed to open file.\n";
```

47.

```
dataFile.seekg(0L, ios::end);
numBytes = dataFile.tellg();
cout << "The file has " << numBytes << " bytes.\n";
```

49. True

51. True

53. True

55. False

57. True

59. True

61. True

63. False

65. File should be opened as

```
fstream file("info.dat", ios::in | ios::out);
```

or

```
fstream file;
file.open("info.dat", ios::in | ios::out);
```

67. File access flags must be specified with `fstream` objects.

69. The file access flag should be `ios::in`. Also, the `get` member function cannot be used to read a string.

71. The file access flag should be `ios::out`. Also, the last line should read

```
dataFile.write(reinterpret_cast<char *>(&dt), sizeof(dt));
```

Chapter 13

1. A class describes a data type. An instance of a class is an object of the data type that exists in memory.

3. `private`

5. A class is analogous to the blueprint.

7. Yes it is. This protects the variables from being directly manipulated by code outside the class, and prevents them from receiving invalid data.

9. When the function is necessary for internal processing, but not useful to the program outside the class. In some cases a class may contain member functions that initialize member variables or destroy their contents. Those functions should not be accessible by an external part of the program because they may be called at the wrong time.

11. A default constructor is a constructor that is called without any arguments. It is not possible to have more than one default constructor.

13. Yes, the constructor executes when the object is created.

15. A class's responsibilities are the things that the class is responsible for knowing and the actions that the class is responsible for doing.

17. procedural programming, object-oriented programming

19. object-oriented

21. `class`

23. access specifier

25. `public`

27. `->`

29. canine.cpp

31. constructor

33. constructors

35. default

37. ~

39. default

41. constructor, destructor

43.

```
class Circle
{
private:
    double radius;
public:
    void setRadius(double r)
        { radius = r; }
    double getRadius()
        { return radius; }
    double getArea()
        { return 3.14159 * radius * radius; }
};
```

45.

```
class Circle
{
private:
    double radius;
public:
    Circle()
         { radius = 0.0; }
    Circle(double r)
         { radius = r; }
    void setRadius(double r)
         { radius = r; }
    double getRadius()
         { return radius; }
    double getArea()
         { return 3.14159 * radius * radius; }
};
```

47. `Circle collection[5] = {12, 7, 9, 14, 8 };`

49.

Animal	Medication	Nurse
Doctor	Invoice	Customer
Patient	Client	

51. False

53. False

55. False

57. True

59. False

61. True

63. True

65. True

67. False

69. True

71. False

73. There should not be a colon after the word `Circle`.

Colons should appear after the words `private` and `public`.

A semicolon should appear after the closing brace.

75. The semicolon should not appear after the word `DumbBell`.

The function header for `setWeight` should appear as:

```
void DumbBell::setWeight(int w)
```

The line that reads:

```
DumbBell(200);
```

should read:

```
bar.setWeight(200);
```

`bar.weight` cannot be accessed outside of the class because no access specifier appeared before it in the class, making the variable private to the class by default. This means the `cout` statement will not work.

Chapter 14

1. Each class object has its own copy of the class's instance member variables. All objects of a class share the class's static member variables.

3. Outside the class declaration.

5. Because every member function of the friend class would have access to the class's private member variables.

7. When an object is initialized with another object's data.

9. When an object has a pointer as a member, and it points to a chunk of dynamically allocated memory. When this object is copied to another object via memberwise assignment, the receiving object's pointer will point to the same chunk of memory.

11. It is a copy constructor that is automatically created for a class, and performs memberwise assignment.

13. The object on the right side of the = operator in the statement that called the overloaded operator function.

15. A dummy parameter is used in the function header of a postfix operator.

17. A Boolean value.

19. Place the key word `static` before the variable declaration (inside the class). Then, place a separate definition of the variable outside the class.

21. 3, 3, 1, 0, `Thing::putThing(2);`

23. To inform the compiler of the class's existence before it reaches the class's definition.

25. Because the parameter variable is created in memory when the function executes, and is initialized with the argument object. This causes the copy constructor to be called.

27. outside

29. before

31. forward declaration

33. copy constructor

35. overloaded

37. aggregation

39. `Bird Bird::operator=(const Bird &right)`

41. `bool Yen::operator<(const Yen &right)`

43. `Collection Collection::operator[](const Collection &sub)`

45. True

47. False

49. True

51. True

53. True

55. False

57. True

59. The copy constructor's parameter should be a reference variable.

61. The overloaded + operator function header should read

    ```
    void operator+(const Point &right)
    ```

63. The `float` conversion function header should read

    ```
    operator float()
    ```

Chapter 15

1. When one object is a specialized version of another object, there is an *"is a" relationship* between them. This indicates that one class "is a" specialized version of the other class.

3. Base class access specification specifies how members of the base class are inherited by the derived class. Member access specification specifies how class members may be accessed by code outside the class.

5. No.

7. When a derived class has a function with the same name as a base class's function, and the base class function is not virtual, it is said that the function is redefined in the derived class. If the base class's function is virtual, however, it is said that the function is overridden.

9. An abstract base class is not instantiated itself, but serves as a base class for other classes. The abstract base class represents the generic, or abstract form of all the classes that are derived from it. A class is abstract when it has one or more virtual functions.

11. `Dog`

13. `public`

15. `private`

17. inaccessible, `protected`, `protected`

19. members

21. last

23. The base class version.

25. `static`

27. polymorphism

29. abstract base class

31. chain

33. override or redefine

35. `class SoundSystem : public CDplayer, public Tuner, public CassettePlayer`

37.

```
class B
{
private:
     int m;
protected:
     int n;
public:
     void setM(int);
     int getM();
     void setN(int);
     int getN();
     virtual int calc()
          { return m * n; }

};
class D : public B
{
protected:
     float q;
     float r;
public:
    void setQ(float);
    float getQ();
    void setR(float);
    float getR();
    virtual float calc()
        { return q * r; }
};
```

39. True

41. True

43. False

45. False

47. True

49. True

51. True

53. The first line of the class declaration should read

    `class Car : public Vehicle`

    Also, the class declaration should end in a semicolon.

55. The constructor function header should read

    `SnowMobile(int h, double w) : Vehicle(h)`

    Also, the constructor parameter `w` is not used.

57. The parameter lists for the `setContents` functions must be different.

Chapter 16

1. A throw point is a line in a program that contains a `throw` statement, thus throwing an exception.

3. A try block contains a block of code executing any statements that might directly or indirectly cause an exception to be thrown. A catch block catches a specific exception and contains code to respond to it.

5. Once an exception has been thrown, the program cannot jump back to the throw point. The function that executes a throw statement will immediately terminate. If that function was called by another function, then the calling function will terminate as well. This process, known as unwinding the stack, continues for the entire chain of nested function calls, from the throw point, all the way back to the try block.

7. By catching the `bad_alloc` exception.

9. Because a class object passed to a function template must support all the operators the function will use on the object.

11. Sequence and associative.

13. throw point

15. catch

17. template prefix

19. specialized

21. associative

23. `bad_alloc`

25.
```
char * allocBlock(int size)
{
     char *ptr;

     try
     {
          ptr = new char[size];
     }

     catch(bad_alloc)
     {
          ptr = 0;
     }
     return ptr;
}
```

27.
```
template <class T>
void displayContents(T arr[], int size)
{
     for (int i = 0; i < size; i++)
          cout << arr[i] << endl;
}
```

29.
```
// Search for the value 7.
binary_search(vect.begin(), vect.end(), 7)
```

31. False

33. True

35. True

37. True

39. False

41. True

43. False

45. True

47. The try block must appear before the catch block.

49. The `return` statement should read `return number * number;`

51. The type parameter `T2` is not used.

53. The statement should read `cout << valueSet[2] << endl;`

Chapter 17

1. A linked list can easily grow or shrink in size. In fact, the programmer doesn't need to know how many nodes will be in the list. They are simply created in memory as they are needed. Also, when a node is inserted into or deleted from a linked list, none of the other nodes have to be moved.

3. A pointer that simply points to the first node in the list.

5. The last node in the list usually points to address 0, the null address.

7. Appending a node means that a new node is added to the end of the list. Inserting a node means that a new node is inserted somewhere in the middle of the list.

9. - Remove the node from the list without breaking the links created by the next pointers.
   - Deleting the node from memory.

11. In a singly linked list each node is linked to a single other node. In a doubly linked list each node not only points to the next node, but also the previous one. In a circularly linked list the last node points to the first.

13. head pointer

15. NULL

17. Inserting

19. circular

21.
```
ListNode *nodePtr;
nodePtr = head;
while (nodePtr)
{
    cout << nodePtr->value << endl;
    nodePtr = nodePtr->next;
}
```

23. `list<float> myList;`

25. `myList.reverse();`

27. False

29. True

31. False

33. `nodePtr` is never properly initialized.

35. The node pointers are simply set to NULL. The nodes themselves are not deleted from memory.

Chapter 18

1. Last in first out

3. A static stack has a fixed size and is usually implemented as an array. A dynamic stack expands as items are added to it. Dynamic stacks are implemented as linked lists.

5. `isFull` and `isEmpty`. The `isFull` operation returns true if the stack is full, and false otherwise. This operation is necessary to prevent a stack overflow in the event a push operation is attempted when all of the stack's elements have values stored in them. The `isEmpty` operation returns true when the stack is empty, and false otherwise. This prevents an error from occurring when a pop operation is attempted on an empty stack.

7. `vector`, `list`, or `deque`. By default it is base on the `deque` type.

9. The rear

11. The two primary queue operations are enqueuing and dequeuing. To enqueue means to insert an element at the rear of a queue, and to dequeue means to remove an element from the front of a queue.

13. last

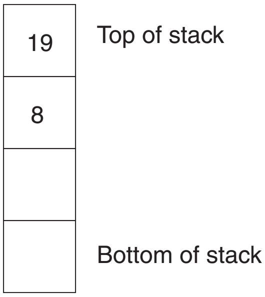
15. static

17. `vectors`, `lists`, and `deques`

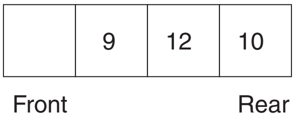
19. enqueuing and dequeuing

21. `deque`

23.



25.



27. Code segment using an if/else statement:

```
if (rear == queueSize - 1)
    rear = 0;
else
    rear++;
```

Code segment using modular arithmetic:

```
rear = (rear + 1) % queueSize;
```

29. False

31. True

Chapter 19

1. For question 12: `num <= 0`

   For question 13: `num > 0`

   For question 14: `pos < size – 1`

3. Recursive functions are less efficient, due to the overhead associated with each function call.

5. The program will eventually run out of stack memory and abort.

7. base case

9. indirect

11.

```
int findLargest(const int arr[], int start, int end)
{
    int largest;

    if(start == end)
       return arr[start];
    else
    {
        largest = findLargest(arr, start + 1, end);
        if(arr[start] >= largest)
           return arr[start];
        else
           return largest;
    }
}
```

13.

```
**********
*********
********
*******
******
*****
****
***
**
*
```

Chapter 20

1. Two others.

3. A node that has no children.

5. The order in which the values are inserted.

7. root node

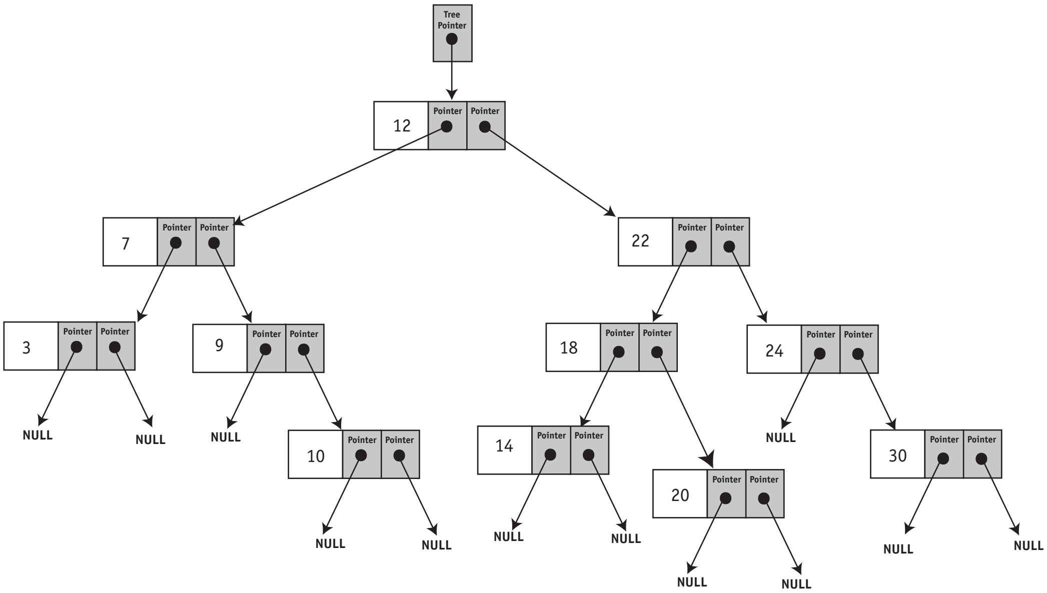
9. leaf node

11. inorder, preorder, and postorder

13.
```
(Recursive Function)
Display In Order(Node Pointer)
    If Node Pointer is not Null
        Display In Order (Node Pointer -> Left).
        Display the node's Value.
        Display In Order (Node Pointer -> Right).
    End If
End Display In Order
```

15.
```
(Recursive Function)
Display Post Order(Node Pointer)
    If Node Pointer is not Null
        Display Post Order (Node Pointer -> Left).
        Display Post Order (Node Pointer -> Right).
        Display the node's Value.
    End If
End Display Post Order
```

17.



19. 12 7 3 9 10 22 18 14 20 24 30

21. True

23. True

25. False