Starting Out With C++: Early Objects, Ninth Edition
Solutions to End-of-Chapter Review Questions

Chapter 1

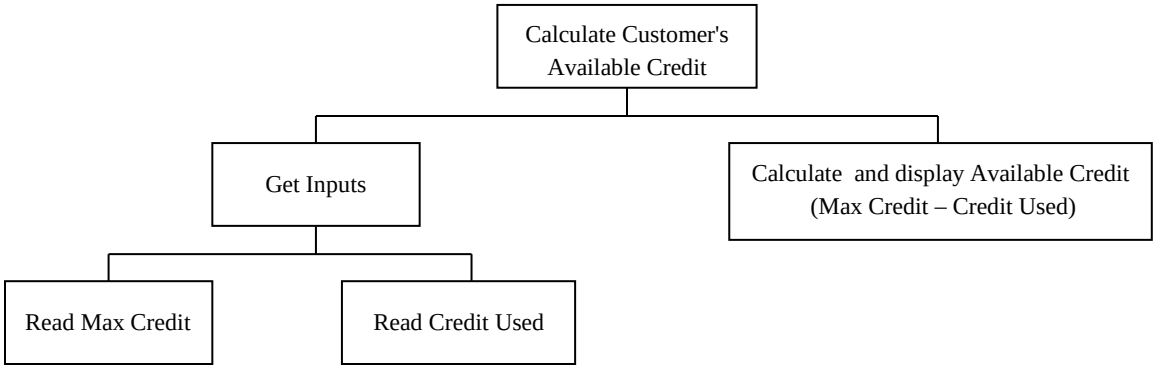
1. programmed
2. CPU
3. arithmetic logic unit (ALU) and control unit
4. disk drive
5. operating systems and application software
6. instructions
7. programming language
8. Machine language
9. High-level
10. Low-level
11. portability
12. key
13. programmer-defined symbols
14. Operators
15. Punctuation
16. syntax
17. variable
18. defined (or declared)
19. input, processing, output
20. Input
21. Output
22. hierarchy chart

23. 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, CD, or flash drive, does not lose its contents when power is removed from the computer.

24. Application software refers to programs used to solve specific problems or perform general operations. System software refers to programs that manage the computer's hardware devices and control their processes or that support the development of application software, such as operating system programs, utility programs, and software development tools.

25. 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 tells the computer to carry out a task incorrectly or to carry out tasks in the wrong order. It causes the program to produce the wrong results.

26. Hierarchy Chart:



27. ***Account Balance High Level Pseudocode***

```
Have user input starting balance
Have user input total deposits
Have user input total withdrawals
Calculate current balance
Display current balance
```

Account Balance Detailed Pseudocode

```
Input startBalance           // with prompt
Input totalDeposits          // with prompt
Input totalWithdrawals       // with prompt
currentBalance = startBalance + totalDeposits - totalWithdrawals
Display currentBalance
```

28. ***Sales Tax High Level Pseudocode***

```
Have user input retail price
Have user input sales tax rate
Calculate tax amount
Calculate sales total
Display tax amount and sales total
```

Sales Tax Detailed Pseudocode

```
Input retailPrice             // with prompt
Input salesTaxRate            // with prompt
taxAmount = retailPrice * salesTaxRate
salesTotal = retailPrice + taxAmount
Display taxAmount, salesTotal
```

29. 45
30. 7
31. 28
32. 365
33. The error is that the program performs its math operation before the user has entered values for the variables `width` and `length`.
34. Some of the questions that should be asked are:
    What standard ceiling height should be used, or is this figure to be input?
    How many square feet should be subtracted out for windows and doors, or do you also want this information input since it could vary by room?
    Are the ceilings also to be painted, or just the walls?
    How many square feet will 1 gallon of paint cover?
    How many coats of paint will you use, or should this information be input?

Chapter 2

1. semicolon
2. `iostream`
3. `main`
4. #
5. braces `{}`
6. literals (also sometimes called constants)
7. 9.7865E14
8. 1, 2
9. A) valid B) invalid C) valid
10. A) valid B) valid C) invalid
11. A) valid B) invalid C) valid only if `Hello` is a variable.
12. A) valid B) invalid C) valid
13. A) 11 B) 14 C) 3 (An integer divide takes place.)
14. A) 9 B) 14 C) 2
15.
```
double temp,
       weight,
       height;
```
16.
```
int months = 2,
    days,
    years = 3;
```
17. A) `d2 = d1 + 2;`
    B) `d1 = d2 * 4;`
    C) `c = 'K';`
    D) `i = 'K';`
    E) `i = i – 1;`
18.
```
A) d1 = d2 – 8.5;
B) d2 = d1 / 3.14;
C) c = 'F';
D) i = i + 1;
E) d2 = d2 + d1;
```
19.
```
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";
```
20.
```
cout << "L\n"
     << "E\n"
     << "A\n"
     << "F\n";
```
    This can also be written as a single string literal: `cout << "L\nE\nA\nF\n";`
21.
```
Input weeks              // with prompt
days = weeks * 7
Display days
```
22.
```
Input eggs              // with prompt
cartons = eggs / 12     // perform integer divide
Display cartons
```

23.
```
Input speed            // with prompt
Input time             // with prompt
distance = speed * time
Display distance
```

24.
```
Input miles            // with prompt
Input gallons          // with prompt
milesPerGallon = miles / gallons
Display milesPerGallon
```

25. A)
```
0
100
```
B)
```
8
2
```
C)
```
I am the incrediblecomputing
machine
and I will
amaze
you.
```

26. A)
```
Be careful!
This might/n be a trick question.
```
B)
```
23
1
```

27. On line 1 the comments symbols are backwards. They should be `/*  */`.
On line 2 `iostream` should be enclosed in angle brackets.
On line 5 there shouldn't be a semicolon after `int main()`.
On lines 6 and 13 the opening and closing braces of function `main` are reversed.
On line 7 there should be a semicolon after `int a, b, c`. In addition, the comment symbol is incorrect. It should be `//`.
On lines 8-10 each assignment statement should end with a semicolon.
On line 11 `cout` begins with a capital letter. In addition, the stream insertion operators should read `<<` instead of `>>` and the variable that is ouput should be `c` instead of capital `C`.

28. Whatever problem a pair of students decides to work with they must determine such things as which values will be input vs. which will be set internally in the program, how much precision is required on calculations, what output will be produced by the program, and how it should be displayed. Students must also determine how to handle situations that are not clear cut. In the paint problem many of these considerations are listed in the teacher answer key (Chapter 1, Question 34). In the recipe program students must determine such things as how to handle quantities, like one egg, that cannot be halved. In the driving program, knowing distance and speed are not enough. Agreement should be reached on how to handle delays due to traffic lights and traffic congestion. Should this be an input value, computed as a percent of overall driving time, or handled some other way?

Chapter 3

1. A) `cin >> description;`
   B) `getline(cin, description);`
2. `char name[35];`
3. A) `cin >> setw(25) >> name;`
   B) `cin.getline(name, 25);`
4. `cin >> age >> pay >> section;`
5. `iostream` and `iomanip`
6. `char city[31];`
7. `string city;`
8. 5, 22, 20, 6, 46, 30, 0, 3, 16
9. A) `a = 12 * x;`
   B) `z = 5 * x + 14 * y + 6 * k;`
   C) `y = pow(x, 4);`
   D) `g = (h + 12) / (4 * k);`
   E) `c = pow(a, 3) / (pow(b, 2) * pow(k, 4));`
10. Two implicit data type conversions occur. First, because `mass` is a `float`, a copy of the `int` value stored in `units` is promoted to a `float` before the multiplication operation is done. The result of `mass * units` will be a `float`. The second data type conversion occurs when the `float` result is promoted to a `double` in order to be stored in `double` variable `weight`.
11. 8
12. Either of these will work:
```
unitsEach = static_cast<double>(qty) / salesReps;
unitsEach = qty / static_cast<double>(salesReps);
```
13. `const int RATE = 12;`
14.
```
x += 5;
total += subtotal;
dist /= rep;
ppl *= period;
inv -= shrinkage;
num %= 2;
```
15. `east = west = north = south = 1;`
16.
```
count = sales = orders = 0;
start = day = 1;
```
17. `int sum = 0;`
18. No. A named constant must be initialized at the time it is defined. It cannot be assigned a value at a later time.

19.
```
cout << fixed << showpoint << setprecision(2);
cout << setw(8) << divSales;
```

20.
```
cout << fixed << showpoint << setprecision(4);
cout << setw(12) << profit;
```

21. A) `cmath` B) `fstream` C) `iomanip` D) `cstdlib`

Note: Once students understand that inputs from the keyboard should *always* be preceded by prompts, the `// with prompt` comment can be omitted from the pseudocode. Therefore, beginning with Chapter 3, we no longer include it.

22.
```
discountPct = .15
Input salesAmt
amtSaved = salesAmt * discountPct
amtDue = salesAmt – amtSaved
Display amtSaved, amtDue
```

23.
```
Input score1, score2, score3
average = (score1 + score2 + score3) / 3.0
Display average
```

24.
```
Input maxCredit
Input creditUsed
availableCredit = maxCredit – creditUsed
Display availableCredit
```

25.
```
Set PI = 3.14
Set cost12In = 14.95
Set cost14In = 17.95
area12 = PI * (12/2)²
area14 = PI * (14/2)²
pricePerSqIn12 = cost12In / area12
pricePerSqIn14 = cost14In / area14
Display pricePerSqIn12, pricePerSqIn14
```

26. A) `Your monthly wages are 3250` `// Some compilers display 3250.0000`
B) `6 3 12`

27. A) `Hello George`
B) `Hello George Washington`

28. A) All the `cin` and `cout` statements must begin with a lowercase `c`, not `C`.
The `<<` operator is used with each `cin`. The `>>` operator should be used instead.
The assignment statement should read:
```
sum = number1 + number2;
```
The last `cout` statement should have `<<` after `cout` and should end with a semi-colon.

B) The `cin` statement should read:
```
cin >> number1 >> number2;
```
The assignment statement should read:
```
quotient = static_cast<double>(number1) / number2;
```
The last `cout` statement is missing a semicolon.

29. A) The variables in the first line should not be declared `const`.
The prompt does not indicate that the inputs must be integers.
The last `cout` statement is missing a semicolon.

B) The `=*` symbol should be `*=`
The string in the final `cout` statement needs to end with a blank to separate it from the variable that follows.

30. Before the price per square inch of a pizza can be calculated, we need to know both the number of square inches it contains and its price. The price for each size pizza can be set at the beginning of the program as constants, since they are known. This can also be done with PI, which is needed for the pizza area calculation. We will use 3.14 for PI because that is precise enough for our calculations. The area of each pizza can be calculated as

$$area = PI * radius^2$$

where the radius of each pizza is half of its diameter. Now that the price of each pizza and its area are known, the price per square inch for each pizza can be found by dividing the price by the area.

If you are unsure what to divide by what to get the answer, try thinking of a simple example using actual numbers. Suppose a pizza contained only 12 square inches and cost $12.00, then it would cost 12 / 12 or $1.00 per square inch. But if it were twice that big for the same price, it would only cost half as much per square inch. Right? Since 24/12 = $2.00 per square inch, that can't be right. But 12 / 24 = $ .50 per square inch. That is clearly correct. So you can see that we need to divide the price by the square inches to get the correct result.

Chapter 4

1. relational
2. false, true
3. false, true
4. braces
5. true, false
6. `default`
7. false
8. true
9. `!`
10. lower
11. `&&`
12. `||`
13. block (or local)
14. integer
15. `break`
16. A) 1 B) 0 C) 0 D) 1

17.
```
if (y == 0)
    x = 100;
```

18.
```
if (y == 10)
    x = 0;
else
    x = 1;
```

19.
```
if (score >= 90)
    cout << "Excellent";
else if (score >= 80)
    cout << "Good";
else
    cout << "Try Harder";
```

20.
```
if (minimum)          // This is the same as if (minimum == true)
    hours = 10;
```

21.
```
if(x < y)
    q = a + b;
else
    q = x * 2;
```

22.
```
switch (choice)
{
    case 1:   cout << fixed << showpoint << setprecision(2);
              break;
    case 2:
    case 3:   cout << fixed << showpoint << setprecision(4);
              break;
    case 4:   cout << fixed << showpoint << setprecision(6);
              break;
    default:  cout << fixed << showpoint << setprecision(8);
              break;
}
```

23. A) T B) F C) T

24. A) T B) F C) T

25.
```
if (grade >= 0 && grade <= 100)
    cout << "The number is valid.";
```

26.
```
if (temperature >= -50 && temperature <= 150)
   cout << "The number is valid.";
```

27.
```
if (hours < 0 || hours > 80)
   cout << "The number is not valid.";
```

28. When using string objects, use the following code:

```
if(title1 <= title2)
   cout << title1 << " " << title2 << endl;
else
   cout << title2 << " " << title1 << endl;
```

With using C-strings, you must replace the above `if` statement with:

```
if (strcmp(title1, title2) <= 0)
```

29.
```
if(sales < 10000)
   commission = .10;
else if (sales <= 15000)
   commission = .15;
else
   commission = .20;
```

30. There are several correct ways to write this. Here is one way.

```
if(dept == 5 && price >= 100)
   discount = .20;
else if (price >= 100)  // but dept is not 5
   discount = .15;
else if(dept == 5)      // but price < 100
   discount = .10;
else                    // dept is not 5 and price < 100
   discount = .05;
```

31. It should read

```
if (!(x > 20))
```

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

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

34. The statement will always be true. If `x` equals neither 1 nor 2, it is clearly true. If `x` equals 1 it is true because `x != 2` is true. If `x` equals 2 it is true because `x != 1` is true. The statement should use && instead of `||`.

35. A) The first `cout` statement is terminated by a semicolon too early.
The definition of `score1`, `score2`, and `score3` should end with a semicolon.
The statement:

```
if(average = 100)
```

needs an `==` sign instead of an assignment operator and should not end with a semicolon.
`perfectScore` is used before it is declared.
The final `if` statement should not be terminated with a semicolon and the conditionally-executed block following it should be enclosed in braces.

B) The conditionally-executed blocks in the `if/else` construct should be enclosed in braces.
The statement

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

should end with a semi-colon, rather than with a `<<`.

C) The trailing `else` statement should come at the end of the `if/else` construct.

D) The variable or expression to be tested in a `switch` construct must be an integer or character, not a double.
The constant value or expresion for each `case` can only be tested for equality with the `switch` variable or expression. Relational operators cannot be used.
The `switch` statement is missing its needed `break` statements.

36. A) An `if/else if` is more appropriate than a `switch` statement when all test expressions do not involve the same variable or when test expressions need to test more than one condition, work with non-integer values, or use relational operators that test for something other than equality.

B) A `switch` statement is more appropriate than an `if/else if` statement when all tests are comparing a variable for equality with just 1 or a small set of integer values. It is a particularly useful construct to use when you want to utilize the “fall through” feature to carry out more than 1 set of actions when a particular condition is true.

C) A set of nested `if/else` statements is more appropriate than either of the other two constructs when the test conditions that determine the actions to be carried out do not fall into a neat set of mutually exclusive cases. For example, if one condition is true, then which set of actions you wish to take may depend on the outcome of a second test.

Chapter 5

1. increment
2. decrement
3. prefix
4. postfix
5. body
6. iteration
7. pretest
8. posttest loop
9. infinite
10. counter
11. running total
12. accumulator
13. sentinel
14. `for`
15. `do-while`
16. `while` and `for`
17. initialization, test, update
18. nested
19. `break`
20. `continue`

21. `fstream`
22. `ofstream`
23. It will be erased and a new file with the same name will be created.
24. `ifstream`
25. It marks the location of the next byte to be read. When an input file is opened, its read position is initially set to the first byte in the file.
26. Close the file.
27.
```
int num;
cin >> num;
num *=2;
while (num < 50)
{  cout << num << endl;
   num *= 2;
}
```
28.
```
do
{   cout << "\nEnter two numbers: ";
    cin  >> num1 >> num2;
    cout << "The sum of these two numbers is "
         << (num1 + num2) << endl << endl;
    cout << "Do you want to add more numbers (y/n)? ";
    cin  >> doMore;
}while (doMore == 'y' || doMore == 'Y');
```
29.
```
for (int num = 0; num <= 1000; num += 10)
   cout << num << "  ";
```
30.
```
total = 0;       // Initialize the accumulator
cout << "Enter 10 numbers separated by spaces "
     << "and I will tell you their total \n";
cout << "Enter your numbers here: "
for (int input = 1; input <= 10; input++)
{   cin >> num;
    total += num;
}
cout << "\nThese 10 numbers add up to " << total << endl;
```

31.
```
for (int row = 0; row < 3; row++)
{  for (int star = 0; star < 5; star++)
      cout << '*';
   cout << endl;
}
```

32.
```
for (int row = 0; row < 10; row++)
{  for (int col = 0; col < 15; col++)
      cout << '#';
   cout << endl;
}
```

33.
```
char doAgain;
int sum = 0;
cout << "This code will increment sum 1 or more times.\n";
do
{  sum++;
   cout << "Sum has been incremented. Increment it again(y/n)? ";
   cin  >> doAgain;
} while ((doAgain == 'y') || (doAgain == 'Y'));
cout << "Sum was incremented " << sum << " times.\n";
```

34.
```
int number;
cout << "Enter an even number: ";
cin  >> number;
while (number % 2 != 0)
{  cout << "Number must be even. Reenter number: ";
   cin  >> number;
}
```

35.
```
for (int count = 0; count < 50; count++)
   cout << "count is " << count << endl;
```

36.
```
int x = 50;
while (x > 0)
{  cout << x << " seconds to go.\n";
   x--;
}
```

37.
```
for (int num = 1; num <= 50; num++)
   outputFile << num << "  ";
```

38.
```
while (inputFile >> num)  // While a value was found and input
   cout << num << "  ";   // (i.e., EOF not yet reached) print it.
```

39. Nothing will print. The erroneous semicolon after the `while` condition causes the `while` loop to end there. Because `x` will continue to remain `1`, `x < 10` will remain true and the infinite loop can never be exited.

40. 10. Because there are no braces only the `x++` statement is in the body of the loop.

41. 2 4 6 8 10

42.
```
$9999
$9999
$9999
```

43. A) The statement `result = ++(num1 + num2);` is invalid. The increment operator cannot operate on a literal.

    B) The `while` loop tests the variable `again` before it has been assigned a value
    The `while` loop is missing its opening and closing braces.

44. A) The `while` statement should not end with a semicolon.
    The `count` variable is never initialized.
    It could also be argued that `BigNum` should be declared a `long int`.

    B) The variable `total` is not initialized to `0`.
    There should be no `count++` inside the body of the loop.
    Ideally, the `num` variable should be defined before the loop, not in it.
    The string `"The average is"` in the final `cout` statement needs a closing quotation mark.

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

    B) The variable `total` is not initialized to 0.
    The `while` loop does not change the value of `count`, so it iterates an infinite number of times.

Chapter 6

1. header
2. `void`
3. `showValue(5);`
4. definition, prototype
5. arguments
6. parameters
7. value
8. prototype
9. local
10. Global
11. Global
12. 0
13. local
14. Static
15. `return`
16. Default
17. last
18. literal
19. reference
20. `&`
21. reference
22. `exit`
23. parameter lists

24. Each function can handle one small, manageable task. This makes it easier to design, code, test, and debug.

25. Arguments appear in the parentheses of a function call. They are the actual values passed to a function. Parameters appear in the parentheses of a function heading. They are the variables that receive the arguments.

26. yes

27. Function overloading means including more than one function in the same program that has the same name. C++ allows this providing the overloaded functions can be distinguished by having different parameter lists.

28. Pass it by value.

29. You want the function to change the value of a variable that is defined in the calling function.

30. with a `return` statement.

31. Yes, but within that function only the local variable can be "seen" and accessed.

32. Use a static variable when you need a local variable to retain its value between function calls.

33.
```
double half(double value)
{
   return value / 2;
}
```

34.
```
result = cube(4);
```

35.
```
void timesTen(int num)
{
   cout << num * 10;
}
```

36.
```
display(age, income, initial);
```

37.
```
void getNumber(int &number)
{
   cout << "Enter an integer between 1 and 100): ";
   cin  >> number;
   while (number < 1 || number > 100)
   {  cout << "This value is out of the allowed range.\n"
           << "Enter an integer between 1 and 100): ";
   }
}
```

38.
```
int biggest(int num1, int num2, int num3)
{
   if (num1 >= num2 && num1 >= num3)
      return num1;
   if (num2 >= num3)
      return num2;
   return num3;
}
```

39. A) The data type of `value2` and `value3` must be declared in the function header.
The function is declared `void` but returns a value.

B) The assignment statement should read:
```
average = (value1 + value2 + value3) / 3.0;
```
The function is declared as a `double` but returns no value.

C) `width` should have a default argument value.
The function is declared `void` but returns a value.

D) The parameter should be declared as:
```
int &value
```
The `cin` statement should read:
```
cin >> value;
```

E) The functions must have different parameter lists.

Chapter 7

1. Abstract Data Type
2. A and B only
3. procedural programming and object-oriented programming
4. Procedural, object-oriented
5. data and procedures (i.e., functions)
6. instance
7. instantiating
8. C
9. member variables
10. member functions
11. encapsulation
12. any function inside or outside of the class, only functions inside the class
13. member variables, member functions
14. accessor
15. mutator
16. inline
17. class
18. created (defined)
19. return
20. default
21. destroyed
22. tilde (~)
23. default
24. parameter list
25. constructor, destructor
26. I/O
27. public
28. private
29. False. It can be both passed to a function and returned from a function.
30. False.
31. separate (i.e., each in their own file)
32. `Canine.h`
33. `Canine.cpp`
34. member functions
35. public
36. declared
37. initialization list, constructor
38. dot (`.`)
39. `Inventory trivet = {555, 110};`
40. `Car hotRod("Ford", "Mustang", 2010, 22495.0);`
41.
```
struct TempScale
{  double fahrenheit;
   double celsius;
};

struct Reading
{  int windSpeed;
   double humidity;
   TempScale temperature;
};

Reading today;

today.windSpeed = 37;
today.humidity = .32;
today.temperature.fahrenheit = 32;
today.temperature.celsius = 0;
```

42.
```
void showReading(Reading value)
{
   cout << "Wind speed: " << value.windSpeed << endl;
   cout << "Humidity: " << value.humidity << endl;
   cout << "Fahrenheit temperature: "
        << value.temperature.fahrenheit << endl;
   cout << "Centigrade temperature: "
        << value.temperature.centigrade << endl;
}
```

43.
```
void inputReading(Reading &r)
{
   cout << "Enter the wind speed: ";
   cin  >> r.windSpeed;
   cout << "Enter the humidity: ";
   cin  >> r.humidity;
   cout << "Enter the fahrenheit temperature: ";
   cin  >> r.temperature.fahrenheit;
   cout << "Enter the celsius temperature: ";
   cin  >> r.temperature.celsius;
}
```

44.
```
Reading getReading()
{
   Reading local;  // Create a temporary local structure
                   // to hold the input data to be returned
   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;
}
```

45. A) valid
    B) invalid (`enum`, like all reserved words, must be lowercase)
    C) invalid (Enumerators represent integer variables. They cannot be strings.)
    D) valid

46. A) valid
    B) valid
    C) valid
    D) invalid (An `int` value cannot be assigned to a variable that is an enumerated data type. You would need to write `floor2 = static_cast<Department>(dnum);`)

47.
```
Inventory(string id = "000", string descrip = "new", int qty = 0)
{ prodID = id; prodDescription = descrip; qtyInStock = qty; }
```

48.
```
int remove(int numUnits)
{
   if (qtyInStock >= numUnits)
   {  qtyInStock -= numUnits;
      return qtyInStock;
   }
   else
      return –1;
}
```

49. A) The structure declaration has no tag.

    B) The semicolon is missing after the closing brace.

50. A) No structure variable has been declared. `TwoVals` is the structure name.

    B) The `ThreeVals` constructor must not have a return type.
    An entire structure variable cannot be sent to `cout`.

51. A) The `Names` structure needs a constructor that accepts 2 strings.

    B) Structure members cannot be initialized in the structure declaration.

52. A) 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.

    B) There should not be a semicolon after the word `Moon` in the class declaration or after the `MoonWeight` and `getMoonWeight` function headers.
    A semicolon must follow the closing brace of the class declaration.
    The first character of the words `private` and `public` should not be capitalized.
    There should be a colon, not a semicolon, following the words `private` and `public`.
    The name of the constructor must be `Moon`, not `moonWeight`.

53. A) The semicolon should not appear after the word `DumbBell` in the class declaration.

    Even though the `weight` member variable is private by default, it should be preceded with the `private` access specifier.

    Because the `setWeight` member function is defined outside the class declaration, its function header must appear as:

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

    The line that reads: `DumbBell.setWeight(200);`
    should read: `bar.setWeight(200);`

    Because the weight member variable is private, it cannot be accessed outside the class, so the `cout` statement cannot legally output `bar.weight`.

    There needs to be a public `getWeight()` function that the main program can call.

    B) Constructors must be public, not private.

    Both constructors are considered default constructors because both can be called with no arguments. This is illegal because there can be only one default constructor.

    All the parameters in the `Change` function header should have a data type.

54. Animal, Doctor, Patient, Medication, Invoice, Client, Nurse, Customer (i.e. all the nouns)

55. A) The nouns are

Bank	Savings Account	Money	Interest rate
Account	Checking Account	Balance	
Customer	Money market account	Interest	

    After eliminating duplicates, objects, and simple values that can be stored in class variables, the potential classes are: `Bank`, `Account`, and `Customer`.

    B) The only class needed for this particular problem is `Account`.

    C) The `Account` class must know its balance and interest rate. The `Account` class must be able to handle deposits and withdrawals and calculate interest earned. It is this last capability, calculating interest earned, that this application will use.

Chapter 8

1. size declarator
2. integer, 0
3. subscript
4. 0
5. size declarator, subscript
6. bounds
7. initialization
8. 0
9. initialization list
10. A) 5
    B) Nothing. Legal subscripts are 0 through 4.
11. subscript
12. assignment (=)
13. value
14. name (which is actually the address of the first array element)
15. multidimensional
16. rows, columns
17. two
18. braces
19. columns
20. 2 14 8

21. A) 10 B) 0 C) 9 D) 40

22. 2 14 8

23. A) 3 B) 0

24. All 3 statements are illegal.

25. All 5 statements are legal.

26. A parallel relationship between two or more arrays is established through the subscripts. Elements in two arrays that are related should have the same subscript.

27. A) 8 B) 10 C) 80 D) `sales[7][9] = 3.52;`

28.
```
Car collection[25];
```

29.
```
Car forSale[35] = { Car("Ford ", "Taurus  ",  2002, 21000),
                    Car("Honda", "Accord  ",  2001, 11000),
                    Car("Jeep ", "Wrangler",  2004, 24000) };
```

30.
```
for (int x = 0; x < 35; x++)
   {  cout << forSale[x].make  << " ";
      cout << forSale[x].model << " ";
      cout << forSale[x].year  << " ";
      cout << forSale[x].cost  << endl;
   }
```

31.
```
for (int index = 0; index < 25; index++)
      array2[index] = array1[index]
```

32. The first total will be correct. The second one will not be because the `total` accumulator was not reset to zero before beginning to sum the values in the second array.

33.
```
struct PopStruct
{   string name;
    long   population;
};

PopStruct country[12];
ifstream dataIn;
dataIn.open("pop.dat");

for (int index = 0; index < 12; index++)
{  getline(dataIn, country[index].name);
   dataIn >> country[index].population;
   dataIn.ignore();    // Clear the linefeed out of the buffer
}                      // before reading the next string.
dataIn.close();
```

34. A)
```
day = 2;
sum = 0;
for (int hour = 0; hour < 24; hour++)
   sum += temp[day][hour];   // hour changes; day does not
average = sum / 24.0;
```

B)
```
hour = 12;
sum = 0;
for (int day = 0; day < 7; day++)
   sum += temp[day][hour];   // day changes; hour does not
average = sum / 7.0;
```

35.
```
int id[10];
double grossPay[10];
for (int emp = 0; emp < 10; emp++)
   cout << id[emp] << "   " << grossPay[emp] << endl;
```

36. A)
```
Struct Payroll
{int id;             // Employee ID
   double grossPay;  // Weekly gross pay amount
};
Payroll employee[10];

// Once all the data has been placed in the array,
// the following code will display it.

for (int emp = 0; emp < 10; emp++)
 cout << employee[emp].id << "   "
      << employee[emp].grossPay << endl;
```

36. B)
```
class Payroll
{
private:
   int id;             // Employee ID
   double grossPay;    // Weekly gross pay amount

public:

   // Constructor with default values
   Payroll(int i = 999, double pay = 0.0)
   {  id = i;
      grossPay = pay;
   }

   void setID(int i)
   {  id = i; }

   void setPay(double pay)
   {  grossPay = pay; }

   int getID()
   {  return id; }

   double getPay()
   {  return grossPay; }
};

Payroll employee[10];

// Once all the data has been placed in the array,
// the following code will display it.

for (int emp = 0; emp < 10; emp++)
 cout << employee[emp].getID()  << "   "
      << employee[emp].getPay() << endl;
```

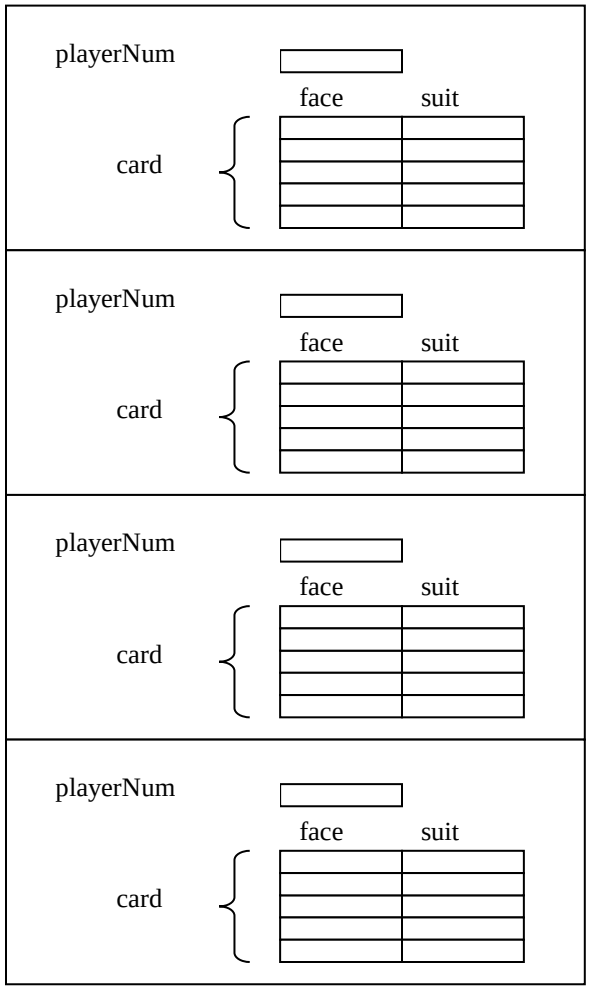
37. A) The size declarator cannot be a variable.
    B) The size declarator cannot be negative.
    C) The initialization list must be enclosed in braces.

38. A) Two of the initialization values are left out.
    B) For the array to be implicitly sized, there must be an initialization list.
    C) All elements of an array must be the same data type and the array declaration must specify what that data type is.

39. A) The parameter should be declared as `int nums[]`.
    Also, spaces should be used to separate the displayed elements.
    B) The parameter must specify the number of columns, not the number of rows. Also, a second parameter is needed to specify the number of rows.

40.

player


playerNum
face
suit
card
playerNum
face
suit
card
playerNum
face
suit
card
playerNum
face
suit
card

Chapter 9

1. linear
2. binary
3. linear
4. binary
5. N/2
6. N
7. first
8. middle
9. 1/8
10. 11
11. ascending
12. descending
13. one
14. one
15. there were no number exchanges on the previous pass
16. N-1

17. Bubble sort normally has to make many data exchanges to place a value in its correct position. Selection sort determines which value belongs in the position currently being filled with the correctly ordered next value and then places that value directly there.

18. bubble sort. Because there would be no data exchanges on the first pass, it would quit after just one pass.

19.

ARRAY SIZE →	100 Elements	1000 Elements	10,000 Elements	100,000 Elements	1,000,000 Elements
Linear Search (Average Comparisons)	50	500	5,000	50,000	500,000
Linear Search (Maximum Comparisons)	100	1000	10,000	100,000	1,000,000
Binary Search (Maximum Comparisons)	7	10	14	17	20

20. Simply swap the corresponding two `empName` elements whenever two `empID` elements are swapped. Here is the pseudocode:

```
do
    Set swap flag to false
    for count = 0 to next-to-last array subscript
       if empID[count] > empID[count+1]
          Swap empID[count] with empID[count+1]
          Swap empName[count] with empName[count+1]
          Set swap flag to true
       end if
    end for
while swap flag is true  // while there was a swap on the previous pass
```

Note that if the data were stored in an array of structures or class objects with `empID` and `empName` members, instead of in two parallel arrays, the sort could be done by comparing the `empID` members and then just swapping two entire structures or objects if they were out of order.

21. A) Map directly from the desired ID to the array location as follows:

```
index = desiredID –101
```

B) Do a linear search starting from the last array element and working backwards until the item is found or until a smaller ID is encountered, which means the desired ID is not in the array. Here is the pseudocode:

```
index = 299        // start at the last element
position = -1
found = false
While index >= 0 and array[index].customerID >= desiredID
                 and not found
   If array[index].customerID = desiredID
      found = true
      position = index
   End If
   Decrement index
End While
Return position
```

Chapter 10

1. address
2. address (&)
3. pointer
4. indirection (*)
5. pointers
6. dynamic memory allocation
7. new
8. an exception
9. null
10. delete
11. new
12. The indirection operator passes from an address to the variable at that address.
13. `cout << *iptr` prints 7, while `cout << iptr` will print the address of the variable `x`.
14. multiplication and indirection
15. You can add or subtract integers using the operators +, ++, -, and --.
16. The pointer will move 4 integers forward in memory.
17. 8
18. The `new` operator is used to allocate memory for variables at runtime.
19. The new operator throws the `bad_alloc` exception when the requested memory is not available. In older compilers, the new operator returns `0` or `NULL` to indicate that the requested memory could not be allocated.
20. It is safe to return a pointer from the function if the variable pointed to is not local to the function, or if the pointer refers to dynamically allocated memory.
21. The `delete` operator de-allocates memory allocated by `new`.
22. With a pointer to a constant, the variable pointed to cannot be changed, but the pointer itself can be changed. With a constant pointer, the pointer itself cannot be changed, although the variable it points to can be changed.
23. A pointer to a constant int: `int const *ptr;`
24. A constant pointer to an int: `int * const ptr;`
25. A smart pointer automatically deallocates the memory it owns when the pointer goes out of scope.
26. The `memory` header file.
27. It deallocates the object it was managing.
28. It returns the raw pointer to the object managed by the smart pointer.
29. `shared_ptr`
30. Pointer arithmetic, initialization from another unique pointer, and assignment from another unique pointer.
31. `make_shared` is more efficient in the way that it allocates memory.
32. You will get a compiler error because current versions of C++ do not support it.

33. `change(&i);`
34. `modify(i);`
35.
```
void exchange(int *p, int *q)
{
   int temp = *p;
   *p = *q;
   *q = temp;
}
```

36. Taking advantage of the `exchange` function of the previous question:
```
void switchEnds(int *array, int size)
{
   exchange(array, array + size - 1);
}
```

37. A) 30 E) 20
B) 30 F) 10
C) 0 G) 10
D) 0 H) 20

38. A) The variable should be declared as `int *ptr;`
B) The assignment statement should read `ptr = &x;`
C) The assignment statement should read `ptr = &x;`
D) The assignment statement should read `*ptr = 100;`
E) The last line should read
```
cout << *(numbers + 2) << endl;
```
F) Multiplication cannot be performed on pointers.
G) Second statement should be `double *dptr = &level;`
H) Order of the two statements should be reversed.
I) The `*` operator is used on the parameter, but it is not a pointer.
J) The second statement should read `pint = new int;`
K) The last statement should be `*pint = 100;`
L) The last line should read `delete [] pint;`
M) The function returns the address of a variable that no longer exists.
N) The type of object pointed to is not specified: should be `unique_ptr<int>`. Also, you cannot use `new` on `unique_ptr`.
O) You cannot assign one unique pointer to another.
P) The parameter passed to `reset()` must be a raw pointer.

39. A list of pointer-related programming guidelines would necessarily be subjective. Items on the list might include:

- Use C++ string objects in preference to C-strings whenever possible
- Pass parameters by reference instead of using pointers
- Avoid complex expressions that involve pointer arithmetic
- Make sure classes that have pointers to dynamically allocated memory have copy constructors, overloaded assignment operators, and destructors. (These topics are discussed in the next chapter).
- Use smart pointers in preference to raw pointers whenever possible!

Chapter 11

1. static
2. outside
3. static
4. before
5. friend
6. forward declaration
7. Memberwise assignment
8. copy constructor
9. `this`
10. overloaded
11. postfix increment (or decrement)
12. composition
13. has-a
14. convert
15. copy constructor
    overloaded = operator
    overloaded = operator
    copy constructor
16. Place the key word `static` before the variable declaration inside the class, then place a separate definition of the variable outside the class.
17. Place the static keyword in the function's prototype. Calls to the function are performed by connecting the function name to the *class* name with the scope resolution operator.
18. 3, 3, 1, 0
    ```
    Thing::putThing(2);
    ```
19. In object composition, one object is a nested inside another object, which creates a has-a relationship. When a class is a friend of another class, there is no nesting. If a class *A* is a friend of a class *B*, member functions of *A* have access to all of *B*'s members, including the private ones.
20. To inform the compiler of the class's existence before it reaches the class's definition.
21. If a pointer member is used to reference dynamically allocated memory, a memberwise assignment operation will only copy the contents of the pointer, not the section of memory referenced by the pointer. This means that two objects will exist with pointers to the same address in memory. If either object manipulates this area of memory, the changes will show up for both objects. Also, if one of the objects frees the memory, the second object will lose access to that memory as well.
22. Because the parameter variable is initialized with a copy of the argument.
23. If an object were passed to the copy constructor by value, a copy of the argument would have to be created before it can be passed to the copy constructor. But then the creation of the copy would require a call to the copy constructor with the original argument being passed by value. This process will continue indefinitely.
24. `Bird& Bird::operator=(Bird right);`
25. ```
    Dollars Dollars::operator++();      // Prefix
    Dollars Dollars::operator++(int);   // Postfix
    ```
26. `bool Yen::operator<(Yen right);`
27. `ostream &operator<<(ostream &strm, Length obj);`
    ```

28. Assume a collection of string objects. Then the subscript operator might be overloaded to return the string at a given position in the collection:

```
string Collection::operator[](int position);
```

29. The overloaded operators offer a more intuitive way of manipulating objects, similar to the way primitive data types are manipulated.

30. A) The last two parameters of the first constructor need to have their type declared as `double`, and the copy constructor's parameter should be a reference variable.

B) The overloaded = operator function header should read

```
Circle& operator=(const Circle right)
```

C) This is really a matter of programming style: the overloaded + operator should not modify its left operand, and should return a value. That will make it work similarly to the built-in + operator:

```
Point operator+(Point right)
{
   Point p= *this;
   p.x = p.x + right.x;
   p.y = p.y + right.y;
   return p;
}
```

D) The overloaded postfix ++ operator function header should read

```
Box operator++(int)
```

and the last two parameters of the constructor should have their types declared. As a matter of programming style, the overloaded increment operators should return values. Also, the parameters to the constructor should have their types declared.

E) The `double` conversion function header should read

```
operator double()
```

31. members
32. `Dog`
33. `Pet`
34. public
35. private
36. private
37. inaccessible, private, private
38. inaccessible, protected, protected
39. inaccessible, protected, public
40. first
41. last
42. scope resolution

43. A) The first line of the class declaration should read

```
class Car : public Vehicle
```

Also, the class declaration should end in a semicolon.

B) The first line of the class declaration should read

```
class Truck : public Vehicle
```

44. Your recommendation would likely have to take into account factors that are not explicitly stated here. If resources are available, one solution would be to assign a team to work on the releases of the software for the next two years, while a second team works on a complete redesign and implementation based on an object-oriented approach. If this is not possible, you recommend a strategy that tries to maximize customer satisfaction while the redesigned product is in progress.

Chapter 12

1. C-string
2. `#include <cstring>`
3. string literal
4. `char *`
5. null terminator
6. `istringstream`
7. `ostringstream`
8. `strlen`
9. concatenate
10. `strcat`
11. `strcpy`
12. `strstr`
13. `strcmp`
14. `strncpy`
15. `atoi`
16. `atol`
17. `atof`
18. `itoa`
19.
```
char lastChar(char *str)
{  //go to null terminator at end
   while (*str != 0)
      str++;
   //back up to last character
   str--;
   return *str;
}
```
20. e
21. h
22. s
23. 9
24. -1
25. Most compilers will print “not equal”. Some compilers store only one copy of each literal string: such compilers will print “equal” because all copies of “a” will be stored at the same address.
26. aaaaabxyzbb
27. abrasion
28. Smith
29. A) C-strings should not be compared with the == operator, and `isupper` does not apply to strings, only single characters.
    B) `atoi` converts a string to an integer, not an integer to a string.
    C) The compiler will not allocate enough space in `string1` to accommodate both strings.
    D) `strcmp` compares C-strings, not characters./
30. A strong argument to keep C-strings can be made on the basis of efficiency: C-strings do not carry the overhead of string objects, and efficiency is critical in some applications. It also makes little sense to drop C-strings if the language will retain arrays and pointers in their current form.

Chapter 13

1. file name
2. opened
3. close
4. `fstream`
5. `ifstream, ofstream, fstream`
6. `ofstream`
7. `ifstream`
8. `fstream`
9. `ofstream people(“people.dat”);`
10. `ifstream pets(“pets.dat”);`
11. `fstream places(“places.dat”);`
12. `ofstream people(“people.dat”, ios::out);`
    `people.open(“people.dat”, ios::out);`
13. `pets.open(“pets.dat”, ios::in);`
    `fstream pets(“pets.dat”, ios::in);`
14. `fstream places(“places.dat”, ios::in | ios::out);`
    `places.open(“places.dat”, ios::in | ios::out);`
15. `null` or 0
16. `fstream employees;`
    `employees.open(“emp.dat”, ios::in | ios::out | ios::binary);`
    `if (!employees)`
    `  cout << “Failed to open file.\n”;`
17. `cout`
18. `eof`
19. `getline`
20. `get`
21. `put`
22. binary
23. text
24. fields
25. structures
26. `write`
27. `read`
28. `char *` typecast
29. sequential
30. random
31. `seekg`
32. `seekp`
33. `tellg`
34. `tellp`
35. `ios::beg`
36. `ios::end`
37. `ios::cur`
38. backward
39. Open the file in binary mode, seek to the end, and then call `tellg` to determine the position of the last byte:

```
ifstream inFile(fileName, ios::binary);
inFile.seekg(0L, ios::end);
long len = inFile.tellg();
```

40. Open the two files for input in binary mode; read the files byte by byte looking for a discrepancy:

```
fstream inFile1(file1name, ios::in|ios::binary);
fstream inFile2(file2name, ios::in|ios::binary);
char ch1, ch2;
while (true)
{  inFile1.read(&ch1, 1);
  inFile2.read(&ch2, 1);
  // Did both reads fail?
  cout << ch1 << " " << ch2 << endl;
  if (inFile1.fail()) cout << "End file 1";
  if (inFile2.fail()) cout << "End of file 2" << endl;
  if (inFile1.fail() && inFile2.fail())
  {  // Yes: both files ended without finding a discrepancy
     cout << "Equal" ; exit(1);
  }
  // Did only one of the reads fail()?
  if (inFile1.eof() || inFile2.eof())
  {
    //Yes: one of the files is shorter than the other
    cout << "Not Equal" ; exit(1);
  }
  // The previous two reads were successful: check if
  // there is a discrepancy
  if (ch1 != ch2)
  {  cout << "Not Equal" ; exit(1);
  }
  // No discrepancy found so far, keep looking
}
```

41. Open the two files in binary mode, the first file for input and the second for output. Seek to the end of the first file, and then keep backing up in the first file while writing to the second.

```
fstream inFile(file1name, ios::in | ios::binary);
fstream outFile(file2name, ios::out | ios::binary);
char ch;
//seek to end of source file
//and then position just before that last character
inFile.seekg(0L, ios::end);
inFile.seekg(-1, ios::cur);
while (true)
{  //we are positioned before a character we need to read
   inFile.get(ch);
   outFile.put(ch);
   //back up two characters to skip the character just read
   //and go to the character before it.
   inFile.seekg(-2, ios::cur);
   if (inFile.fail())
       break;
}
```

42. Read two values, one from each file. Keep comparing the two values and writing the smaller of the two to the output file, and reading a new value to replace the one just written to the output file. When you get to the end of one file, copy the rest of the contents of the other file to the output file.

```
fstream inFile1(filename1, ios::in);
fstream inFile2(filename2, ios::in);
fstream outFile(outfilename, ios::out);
fstream *lastInFile;

int n1, n2;
inFile1 >> n1;
inFile2 >> n2;

while (true)
{
   if (n1 <= n2)
   {
      outFile << n1 << " ";
      inFile1 >> n1;
      if (inFile1.fail())
      {
         outFile << n2 << " ";
         lastInFile = &inFile2;
         break;
      }
   }
   else
   {
      outFile << n2 << " ";
      inFile2 >> n2;
      if (inFile2.fail())
      {
         outFile << n1 << " ";
         lastInFile = &inFile1;
         break;
      }
   }
}
copy rest of lastInFile to outFile
```

43. A) File should be opened as

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

or

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

B) Should not specify `ios::in` with an `ofstream` object. Also, the `if` statement should read

```
if (!File)
```

C) File access flags must be specified with `fstream` objects.

D) Should not write to a file opened for input. Also, the `<<` operator should not be used on binary files.

E) The while statement should read

```
while(!dataFile.eof())
```

Also, `eof()` is not a reliable means of detecting end of file when reading with `>>`. Use the following instead.

```
while (dataFile >> x)
   cout << x << "  " << endl ;
```

F) The last line should be

```
dataFile.getline(line, 81, '\n');
```

G) The `get` member function that takes a single parameter cannot be used to read a string: it can only read single characters.

H) The file access flag should be `ios::out`. Also, the `put` member function cannot be used to write a string, only single characters.

I) The last line should read

```
dataFile.write(&dt, sizeof(date));
```

J) The `seekp` member function should not be used since the file is opened for input.

44. You should make sure your friend understands two things: how pass by value works and how file objects work. A file object keeps track of information needed to access an open file. This information includes the file pointer, which keeps track of the current read or write position in the file. If a file object is passed by value, changes in the file pointer are reflected in the copy that is local to the function, and are lost when the function returns and the local file object disappears.

Chapter 14

1. Indirect recursion. There are more function calls to keep up with.
2. Recursive functions are less efficient due to the overhead associated with each function call.
3. When the problem is more easily solved with recursion, and a better program design is possible.
4. depth
5. direct
6. indirect
7. A) 55
   B) **********
      ****
      ****
      ****
      ******
      *****
      ****
      ***
      **
      *
   C) evE dna madA

8. One example of “recursion” in communication occurs in the procedure used by a dictionary to define the meaning of words. The definition of each word is explained in terms of the definitions of other words. Interestingly, this recursive procedure manages to be useful even though it has no base cases that are handled without recursion!

Chapter 15

1. abstract class
2. pure virtual function, or an abstract function
3. abstract
4. virtual
5. compile
6. run-time
7. polymorphism
8. virtual
9. inheritance
10. composition
11. inheritance
12. application framework
13. final
14. override
15. yes
16. yes

17. The statement compiles correctly.

18. The statement will not compile.

19. `pAnimal = new Dog; pDog = static_cast<Dog *>(pAnimal);`

20. Here is an implementation of the algorithm.

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

// Generic sorting class, using a place holder comparison
class Sorter
{
 private:
   int *p;
   int size;
   void sort(int n);
   virtual bool compare(int x, int y) = 0;
 public:
   void setArray(int *arr, int size)
   {   p = arr; this->size = size;
   }
   void sort() { sort(size); }
 };

// Generic sort function uses recursion and the placeholder
// comparison function
void Sorter::sort(int n)
{
  if (n <= 1) return;
  //find pos of largest in p[0..n-1]
  //and put it at end of p[0..n-1]
  int maxPos = 0;
  for (int k = 1; k < n; k++)
  if (compare(p[k], p[maxPos]))
  {
     maxPos = k;
  }
```

```
    swap(p[n-1], p[maxPos]);
    sort(n-1);
  }

  // Class has concrete comparison for increasing order
  class IncrSorter : public Sorter
  {
  private:
    bool compare(int x, int y)
    {
      return x < y;
    }
  };

  // Class has concrete comparison for decreasing order
  class DecrSorter : public Sorter
  {
  private:
     bool compare(int x, int y)
     {
       return x > y;
     }
  };

  // Prototype for printing arrays.
 void printArray(int [], int);

 // Example of usage
   int main( )
   {
     int a[5] = {23, 56, 23, -45, 52};

     IncrSorter incSorter;
     incSorter.setArray(a, 5);
     incSorter.sort();
     printArray(a, 5);

     DecrSorter decSorter;
     decSorter.setArray(a, 5);
     decSorter.sort();
     printArray(a, 5);

     return 0;
   }

   void printArray(int a[ ], int size)
   {
     for (int k = 0; k < size; k++)
       cout << a[k] << "  ";
     cout << endl;
   }
```

21. A pure virtual function cannot have a body.

22. A direct implementation of this idea will not work because the two base classes have two different array members. When you call `reverse`, it will reverse a different array, not the one that you sorted.

Chapter 16

1. throw point
2. try
3. catch
4. type parameter
5. template prefix
6. actual data type
7. algorithms
8. sequence
9. associative

10. The logic is very similar to that of the copy constructor:

```
template<class T> SimpleVector&
SimpleVector<T>::operator=(SimpleVector obj)
{  if (arraySize > 0)
      delete [ ] aptr;
   arraySize = obj.arraySize;
   aptr = new T[arraySize];
   for (int count = 0; count < arraySize; count++)
     aptr[count] = obj[count];
   return *this;
}
```

11. This solution uses recursion to perform the reversal. It needs the inclusion of the STL algorithm header file to allow use of `swap`.

```
template<class T>
void reverse(T arr[ ], int size)
{  if (size >= 2)
   {  swap(arr[0], arr[size-1]);
      reverse(arr+1, size-2);
   }
}
```

12.
```
template <class T>
T sum(T x, T y)
{
  return x + y;
}
```

13. The array will be reversed.
14. The array will be reversed.
15. A) The `try` and `catch` key words are misplaced
    B) The cout statement should not appear between the try and catch blocks.
    C) The return statement should read `return number * number;`
    D) The type parameter T is not used.
    E) The type parameter T2 is not used.
    F) The declaration should read `SimpleVector<int> array(25);`
    G) The statement should read `cout << valueSet[2] << endl;`
16. There are no simple answers. Try to find out the cause of the problem, and whether the team member can be helped .The team member may be having personal problems that are affecting performance. If the team member's problems are technical in nature, additional training or education may help. Problems with a team member's work ethic may be the most

difficult to deal with. As project leader, you have to figure out how to balance the need to help a team member with the need to ensure success of the project.

Chapter 17

1. head pointer
2. self-referential data structure
3. NULL
4. Appending
5. Inserting
6. Traversing
7. circular
8. doubly-linked
9.
```
void printFirst(ListNode *ptr)
{
   if (!ptr) { cout << “Error”; exit(1);}
   cout << ptr->value;
}
```
10.
```
void printSecond(ListNode *ptr)
{
   if (!ptr || !ptr->next) {cout << “Error”; exit(1);}
   cout << ptr->next->value;
}
```
11.
```
double lastValue(ListNode *ptr)
{
   if (!ptr) { cout << “Error”; exit(1);}
   if (ptr->next == NULL)
      return ptr->value;
   else
      return lastValue(ptr->next);
}
```
12.
```
ListNode *removeFirst(ListNode *ptr)
{
   if (ptr == NULL)
      return NULL;
   ListNode *first = ptr;
   ptr = ptr->next;
   delete first;
   return ptr;
}
```

13.
```
ListNode *ListConcat(ListNode *list1, ListNode *list2)
{
   if (list1 == NULL)
      return list2;
   // Concatenate list2 to end of list1
   ListNode *ptr = list1;
   while (ptr->next != NULL)
      ptr = ptr->next;
   ptr->next = list2;
   return list1;
}
```

14. 34.2
    34.2

15. 56.4

16. 56.4

17. A) The `printList` function should have a return type of `void`. Also, the use of the head pointer to walk down the list destroys the list: use an auxiliary pointer initialized to head instead.

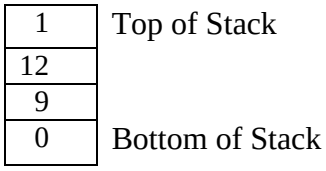
    B) Eventually the pointer `p` becomes `NULL`, at which time the attempt to access `p->NULL` will result in an error. Replace the test `p->next` in the `while` loop with `p`. Also, the function fails to declare a return type of `void`.

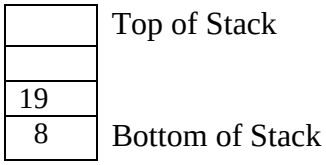
    C) The function should declare a return type of `void`. Also, the function uses `p++` erroneously in place of `p=p->next` when attempting to move to the next node in the list.

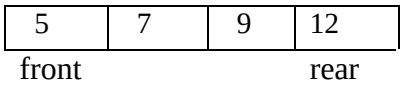
    D) The destructor does not deallocate storage for the nodes. Replace the statement `nodePtr->next = NULL;` with `delete nodePtr;`

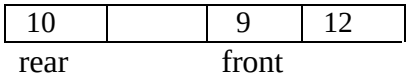
18. There are many reasons that would justify such a workshop if it were needed. Learning how things work on the inside helps programmers make better decisions about when to use them, especially when efficiency is a major consideration. Also, in learning how to code linked lists, stacks and queues, you pick up programming techniques that are useful in solving other problems.

Chapter 18

1. Last-In-First-Out
2. The element at the top
3. A static stack has all its storage allocated at once, when the stack is created. A dynamic stack allocates storage for each element as it is added. Normally, static stacks use array-based implementations, whereas dynamic stacks use linked lists.
4. push adds an element to the stack; pop removes an element from the stack.
5. It takes an existing container and implements a new interface on top of it to adapt it to a different use.
6. vectors, lists, and deques. Default implementations are based on deques.
7. First-In-First-Out
8. The rear of the queue
9. The front of the queue
10. enqueue adds an item to a queue; dequeue takes an item from the queue
11. lists and deques
12. 

13. 

14. 

15. Assuming a circular array buffer:


16. It allows the queue elements to "wrap around" the end of the array. This, in turn, makes the queue more efficient by eliminating the need to copy the queue elements forward each time an item is enqueued.
17. Use two stacks, a main stack and an auxiliary stack. The main stack will store all items that are currently enqueued.
    - To enqueue a new item, push it onto the main stack.
    - To dequeue an item, keep popping items from the main stack and pushing them onto the auxiliary stack until the main stack is empty, then pop and store the top element from the auxiliary stack into some variable *X*. Now keep popping items from the auxiliary stack and pushing them back onto the main stack till the auxiliary stack is empty. Return the stored item *X*.
    - To check if the queue is empty, see if the main stack is empty.
18. Other than the obvious stack of plates in a cafeteria, people put on and remove layers of clothing in a last-in-first-out order. Also, visitors to a party who pack their cars in a long narrow driveway have to remove their cars in last-in-first-out order.

Chapter 19

1. root node
2. children
3. leaf node
4. subtree
5. inorder, preorder, and postorder
6. They are both collections of nodes with links to successor nodes
7.
```
struct TreeNode
{
   int value;
   TreeNode *left, *middle, *right;
};
```

8.
```
struct TreeNode
{
  int value;
  TreeNode *child[100];
};
```

9. To traverse a ternary tree in preorder, visit the root, then traverse the left, middle, and right subtrees.

```
preorder(ternarytree)
   If (ternarytree != NULL)
      visit the root
      preorder left subtree of ternarytree
      preorder middle subtree of ternarytree
      preorder right subtree of ternarytree
   End If
End preorder
```

10. To traverse a ternary tree in postorder, first traverse the left subtree, then the middle subtree, and then the right subtree. Finally, visit the root.

```
postorder(ternarytree)
   If (ternarytree != NULL)
     postorder left subtree of ternarytree
     postorder middle subtree of ternarytree
     postorder right subtree of ternarytree
     visit the root
   End If
End postorder
```

11. We must decide whether to visit the root immediately after the traversal of the left subtree, or immediately after the traversal of the middle subtree.

12.
```
bool search(btree, num)
   bool found; //local flag

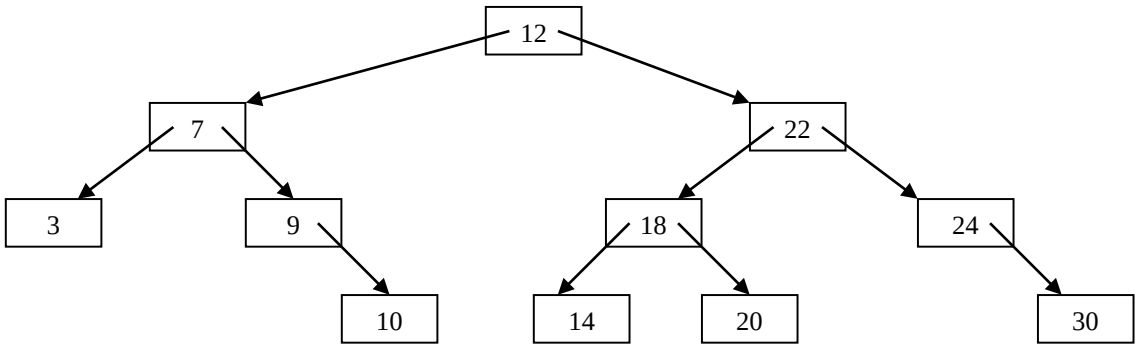
   If btree == NULL Then
      return false
   End If
   If num == value in root of btree Then
      return true
   End If
   found = search(left subtree of btree, num)
   If found Then
      return true
   Else
      return search(right subtree of btree, num)
End search
```

13.
```
int largest(TreeNode *tree)
   Set a pointer p to the root node of tree
   While node at p has a right child Do
      Set p to the right child of the node at p
   End While
   return value in the node at p
End largest
```

14.
```
int smallest(TreeNode *tree)
   Set a pointer p to the root node of tree
   While node at p has a left child Do
      Set p to the left child of the node at p
   End While
   return value in the node at p
End smallest
```

15.
```
void increment(TreeNode *tree)
   If (tree != NULL)
      tree->value = tree->value + 1
      increment(tree->left)
      increment(tree0>right)
   End If
End increment
```

16.



17. 3 7 9 10 12 14 18 20 22 24 30

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

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

20. One way to write programs so they can be easily retargeted for international markets is to separate all strings that will appear in the user interface (as prompts, labels of output, etc) from the rest of the code. Further research into the intended market may be advisable: for example, in some cultures, certain colors have vulgar connotations, so it would be best to avoid them.