Appendix N: Answers to Checkpoints

Chapter 1

1.1 Because the computer can be programmed to do so many different tasks.

1.2 The Central Processing Unit (CPU), main memory, secondary storage devices, input devices, output devices.

1.3 Arithmetic and Logic Unit (ALU), and Control Unit

1.4 Fetch: The CPU's control unit fetches the program's next instruction from main memory.

Decode: The control unit decodes the instruction, which is encoded in the form of a number. An electrical signal is generated.

Execute: The signal is routed to the appropriate component of the computer, which causes a device to perform an operation.

1.5 A unique number assigned to each section of memory.

1.6 Program instructions and data are stored in main memory while the program is operating. Main memory is volatile, and loses its contents when power is removed from the computer. Secondary storage holds data for long periods of time—even when there is no power to the computer.

1.7 Operating Systems and Application Software

1.8 The operating system

1.9 A utility program

1.10 application software

1.11 A set of well-defined steps for performing a task or solving a problem.

1.12 To ease the task of programming. Programs may be written in a programming language, and then converted to machine language.

1.13 A low-level language is close to the level of the computer, and resembles the system's numeric machine language. A high-level language is closer to the level of human readability, and resemble natural languages.

1.14 That a program may be written on one type of computer and run on another type.

1.15 The preprocessor reads the source file searching for commands that begin with the # symbol. These are commands that cause the preprocessor to modify the source file in some way. The compiler translates each source code instruction into the appropriate machine language instruction, and creates an object file. The linker combines the object file with necessary library routines.

1.16 Source file: contains program statements written by the programmer.

Object file: machine language instructions, generated by the compiler translated from the source file.

Executable file: code ready to run on the computer. Includes the machine language from an object file, and the necessary code from library routines.

1.17 A programming environment that includes a text editor, compiler, debugger, and other utilities, integrated into one package.

1.18 A key word has a special purpose, and is defined as part of a programming language. A programmer-defined identifier is a word or name defined by the programmer.

1.19 Operators perform operations on one or more operands. Punctuation symbols mark the beginning or ending of a statement, or separates items in a list.

1.20 A line is a single line as it appears in the body of a program. A statement is a complete instruction that causes the computer to perform an action.

1.21 Because their contents may be changed.

1.22 The original value is overwritten.

1.23 The variable must be defined.

1.24 Input, processing, and output.

1.25 The program's purpose, information to be input, the processing to take place, and the desired output.

1.26 To imagine what the computer screen looks like while the program is running. This helps define input and output.

1.27 A chart that depicts each logical step of the program in a hierarchical fashion.

1.28 The programmer steps through each statement in the program from beginning to end. The contents of variables are recorded, and screen output is sketched.

1.29 It translates each source code statement into the appropriate machine language statement..

1.30 A logical error that occurs while the program is running.

1.31 By the compiler

1.32 To determine if a logical error is present in the program.

1.33 Procedural programs are made of procedures, or functions. Object-oriented programs are centered on objects, which contain both data and the procedures that operate on the data.

Chapter 2

2.1
```
// A crazy mixed up program
#include <iostream>
using namespace std;

int main()
{
   cout << "In 1492 Columbus sailed the ocean blue.";
   return 0;
}
```

2.2
```
// It's a mad, mad program
#include <iostream>
using namespace std;

int main()
{
   cout << "Success\n";
   cout << "Success";
   cout << " Success\n\n";
   cout << "Sucess\n";
   return 0;
}
```

2.3
```
The works of Wolfgang
include the following
The Turkish March
and Symphony No. 40 in G minor.
```

2.4
```
// Today's Date: September 3, 2012
#include <iostream>
using namespace std;

int main()
{
   cout << "Teresa Jones\n";
   cout << "127 West 423rd Street\n";
   cout << "San Antonio, TX 55555\n";
   cout << "555-555-1212\n";
   return 0;
}
```

2.5 Variables: `little` and `big`.

Constants: 2, 2000, “The little number is ”, “The big number is ”

2.6 The value is number

2.7 `99bottles`: Variable names cannot begin with a number.

`r&d`: Variable names may only use alphabetic letters, digits, or underscores

2.8 No. Variable names are case sensitive.

2.9 A) `short`, or `unsigned short`.

B) `int`

C) They both use the same amount of memory.

2.10 They both use the same amount of memory.

2.11 67, 70, 87

2.12 ‘B’

2.13 ‘Q’ uses one byte

“Q” uses two bytes

“Sales” uses six bytes

‘\n’ uses one byte

2.14
```
#include <iostream>
using namespace std;

int main()
{
   char first, middle, last;
```

```
   first = 'T';
   middle = 'E';
   last = 'G';
   cout << first << " " << middle << " " << last << endl;
   return 0;
}
```

2.15 The string constant "Z" is being stored in the character variable `letter`.

2.16 The `string` header file

2.17
```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string name = "John Smith";
    string address = "224 Maple Street\nClyde, NC 28721";
    string phone = "555-5050";

    cout << name << endl;
    cout << address << endl;
    cout << phone << endl << endl;
    return 0;
}
```

2.18 No

2.19 6.31E17

2.20
```
#include <iostream>
using namespace std;

int main()
{
   int age;
   float weight;

   age = 26;
   weight = 180;
   cout << "My age is " << age << endl;
   cout << "My weight is " << weight << endl;
   return 0;
}
```

2.21 Invalid. The value on the left of the = operator must be an lvalue.

2.22 `int x = 7, y = 16, z = 28;`

2.23 The variable `number` is assigned a value before it is defined. Correct the program by moving the statement `number = 62.7;` to the point after the variable declaration. Here is the corrected program:

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

int main()
{
   double number;
   number = 62.7;
   cout << number << endl;
   return 0;
}
```

2.24 Integer division. The value 23 will be stored in `portion`.

2.25
```
const float E = 2.71828;
const int MINUTES_IN_A_YEAR = 5.256E5;
const float G_FEET = 32.2;
const float G_METERS = 9.8;
const int METERS_IN_A_MILE = 1609
```

Chapter 3

3.1 `iostream`

3.2 True

3.3 B

3.4 `cin >> miles >> feet >> inches;`

3.5 Include one or more `cout` statements explaining what values the user should enter.

3.6
```
#include <iostream>
using namespace std;

int main()
{
   double pounds, kilograms;

   cout << "Enter your weight in pounds: ";
   cin >> pounds;
   // The following line does the conversion.
   // One kilogram weighs 2.2 pounds.
   kilograms = pounds / 2.2;
   cout << "Your weight in kilograms is ";
   cout << kilograms << endl;
   return 0;
}
```

3.7

Value
21
2
31
5
24
2
69
0
30

3.8
```
y = 6 * x;
a = 2 * b + 4 * c;
y = x * x;  or  y = pow(x, 2.0);
g = (x + 2) / (z * z);  or  g = (x + 2.0) / pow(z, 2.0);
y = (x * x) / (z * z);  or  y = pow(x, 2.0) / pow (z, 2.0);
```

3.9

If the user enters…	*The program displays…*
2	6
5	27
4.3	20.49
6	38

3.10
```
#include <iostream>
#include <cmath>
using namespace std;
```

```
int main()
{
   double volume, radius, height;

   cout << "This program will tell you the volume of\n";
   cout << "a cylinder-shaped fuel tank.\n";
   cout << "How tall is the tank? ";
   cin >> height;
   cout << "What is the radius of the tank? ";
   cin >> radius;
   volume = 3.14159 * pow(radius, 2.0) * height;
   cout << "The volume of the tank is " << volume << endl;
   return 0;
}
```

3.11 A) 2
B) 17.0
C) 2.0
D) 2.4
E) 2.4
F) 2.4
G) 4
H) 27
I) 30
J) 27.0

3.12
```
#include <iostream>
using namespace std;

int main()
{
   char letter;

   cout << "Enter a character: ":
   cin >> letter;
   cout << "The ASCII code for " << letter;
   cout << " is " << static_cast<int>(letter) << endl;
   return 0;
}
```

3.13
```
9
9.5
9
```

3.14
```
total = subtotal = tax = shipping = 0;
```

3.15
```
A) x += 6;
B) amount -= 4;
C) y *= 4;
D) total /= 27;
E) x %= 7;
F) x += (y * 5);
G) total -= (discount * 4);
H) increase *= (salesRep * 5);
I) profit /= (shares – 1000);
```

3.16
```
3
11
1
```

3.17
A) `cout << setw(9) << fixed << setprecision(2) << 34.789;`
B) `cout << setw(5) << fixed << setprecision(3) << 7.0;`
C) `cout << fixed << 5.789e12;`
D) `cout << left << setw(7) << 67;`

3.18
```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string person = "Wolfgang Smith";
    cout << right;
    cout << setw(20);
    cout << person << endl;
    cout << left;
    cout << person << endl;
    return 0;
}
```

3.19
```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   const double PI = 3.14159;
   double degrees, radians;

   cout << "Enter an angle in degrees and I will convert it\n";
   cout << "to radians for you: ";
   cin >> degrees;
   radians = degrees * PI / 180;
   cout << degrees << " degrees is equal to ";
   cout << setw(5) << left << fixed << showpoint
        << setprecision(4) << radians << " radians.\n";
   return 0;
}
```

3.20
`cos`: Returns the cosine of the argument.
`exp`: Returns the exponential function of the argument.
`fmod`: Returns the remainder of the first argument divided by the second argument.
`log`: Returns the natural logarithm of the argument.
`log10`: Returns the base-10 logarithm of the argument.
`pow`: Returns the value of the first argument raised to the power of the second argument.
`sin`: Returns the sine of the argument.
`sqrt`: Returns the square root of the argument.
`tan`: Returns the tangent of the argument.

3.21
```
x = sin(angle1) + cos(angle2);
```

3.22
```
y = pow(x, 0.2);   // 0.2 is equal to 1/5
```

3.23
```
y = 1 / sin(a);
```

Chapter 4

4.1 T, T, F, T, T, F, T

4.2 A) Incorrect
B) Incorrect
C) Correct

4.3 A) yes
B) no
C) no

4.4 0
0
1
0

4.5
```
if (x == 20)
    y = 0;
```

4.6
```
if (price > 500)
    discountRate = 0.2;
```

4.7
```
if (hours > 40)
    payRate *= 1.5;
```

4.8 True

4.9 False

4.10
```
if (sales > 50000)
{
    commissionRate = 0.25;
    bonus = 250;
}
```

4.11 Only the first statement after the `if` statement is conditionally executed. Both of the statements after the `if` statement should be enclosed in a set of braces.

4.12 False

4.13
```
if (y == 100)
    x = 1;
else
    x = 0;
```

4.14
```
if (sales >= 50000.00)
    commissionRate = 0.20;
else
    commissionRate = 0.10;
```

4.15 Zero
Zero Ten
Zero Ten Twenty
No Output

4.16 11

4.17

If the customer purchases this many books...	this many coupons are given.
1	1
2	1
3	2
4	2
5	3
10	3

4.18

Logical Expression	Result (`true` or `false`)
`true && false`	`false`
`true && true`	`true`
`false && true`	`false`
`false && false`	`false`
`true || false`	`true`
`true || true`	`true`
`false || true`	`true`
`false || false`	`false`
`!true`	`false`
`!false`	`true`

4.19 T, F, T, T, T

4.20
```
if (speed >= 0 && speed <= 200)
      cout << "The number is valid.";
```

4.21
```
if (speed < 0 || speed > 200)
      cout << "The number is not valid.";
```

4.22 A) True
B) False
C) True
D) False
E) False
F) True

4.23 A) False
B) False
C) False
D) True
E) False
F) False
G) False

4.24 A)
```
z = x > y ? 1 : 20;
```

B)
```
population = temp > 45 ? base * 10 : base * 2;
```

C)
```
wages *= hours > 40 ? 1.5 : 1;
```

D)
```
cout << (result >= 0 ? "The result is positive\n" :
         "The result is negative.\n");
```

4.25 A)
```
if (k > 90)
   j = 57;
else
   j = 12;
```

B)
```
if (x >= 10)
   factor = y * 22;
else
   factor = y * 35;
```

C)
```
if (count == 1)
   total += sales;
else
   total += count * sales;
```

D)
```
if (num % 2)
   cout << "Odd\n";
else
   cout << "Even\n";
```

4.26 2 2

4.27 Because the `if /else if` statement tests several different conditions, consisting of different variables.

4.28 The case statements must be followed by an integer constant, not a relational expression.

4.29 That is serious.

4.30
```
switch (userNum)
{
   case 1 : cout << "One";
            break;
   case 2 : cout << "Two";
            break;
   case 3 : cout << "Three";
            break;
   default: cout << "Enter 1, 2, or 3 please.\n";
}
```

4.31
```
switch (selection)
{
   case 1  : cout << "Pi times radius squared\n";
             break;
   case 2  : cout << "Length times width\n";
             break;
   case 3  : cout << "Pi times radius squared times height\n";
             break;
   case 4  : cout << "Well okay then, good bye!\n";
             break;
   default : cout << "Not good with numbers, eh?\n";
}
```

Chapter 5

5.1 A) 32
B) 33
C) 23
D) 34
E) It is true!
F) It is true!

5.2
```
int number;
cout << "Enter a number in the range 10 through 25: ";
cin >> number;
while (number < 10 || number > 25)
{
   cout << "Error! The number must be in the range "
        << "of 10 through 25. Enter a valid number: ";
   cin >> number;
}
```

5.3
```
char letter;
cout << "Enter Y for yes or N for no: ";
cin >> letter;
while (letter != 'Y' && letter != 'y' &&
       letter != 'N' && letter != 'n')
{
   cout << "Error! Enter either Y or N: ";
   cin >> letter;
}
```

5.4
```
string input;
cout << "Enter Yes or No: ";
cin >> input;
while ( (input != "Yes") && (input != "No") )
{
     cout << "Error! Enter either Yes or No: ";
     cin >> input;
 }
```

5.5 A) `Hello World`
B) `10`
C) `8 4`

5.6 Initialization, test, and update.

5.7 A) `int count = 1;`
B) `count <= 50;`
C) `count++`
D)
```
for (int count = 1; count <= 50; count++)
   cout << "I love to program\n";
```

5.8 A) `0246810`
B) `-5-4-3-2-101234`
C)
```
5
8
11
14
17
```

5.9
```
for (int count = 1; count <= 10; count++)
   cout << "Chris Coder\n";
```

5.10
```
for (int count = 1; count <= 49; count += 2)
   cout << count << endl;
```

5.11
```
for (int count = 0; count <= 100; count += 5)
   cout << count << endl;
```

5.12
```
int number, total = 0;
for (int count = 0; count < 7; count++)
{
   cout << "Enter a number: ";
   cin >> number;
   total += number;
}
```

5.13 The variable `x` is the counter variable. The variable `y` is the accumulator.

5.14 You must be sure to choose a sentinel value that could not be mistaken as a member of the data list.

5.15 Change the `while` loop to read:
```
while (points >= 0)
```

5.16 An output file is a file that data is written to. An input file is a file that data is read from.

5.17
1. Open the file
2. Process the file
3. Close the file

5.18 A text file contains data that has been encoded as text, using a scheme such ASCII or Unicode. Even if the file contains numbers, those numbers are stored in the file as a series of characters. As a result, the file may be opened and viewed in a text editor such as Notepad. A binary file contains data that has not been converted to text. As a consequence, you cannot view the contents of a binary file with a text editor.

5.19 When you work with a sequential access file, you access data from the beginning of the file to the end of the file. When you work with a random access file, you can jump directly to any piece of data in the file without reading the data that comes before it.

5.20
```
ofstream
```

5.21
```
ifstream
```

5.22
```
ofstream outputFile("num.txt");
for (int num = 1; num <= 10; num++)
     outputFile << num << endl;
outputFile.close();
```

5.23
```
ifstream inputFile("num.txt");
int num;
while (inputFile >> num)
     cout << num << endl;
inputFile.close();
```

5.24
```
ofstream outputFile;
string filename = "Test.txt";
outputFile.open(filename.c_str());
```

Chapter 6

6.1 Function call

6.2 Function header

6.3
```
I saw Elba
Able was I
```

6.4
```
void qualify()
{
   cout << "Congratulations, you qualify for\n";
   cout << "the loan. The annual interest rate\n";
   cout << "is 12%\n";
}

void noQualify()
{
   cout << "You do not qualify. In order to\n";
   cout << "qualify you must have worked on\n";
   cout << "your current job for at least two\n";
   cout << "years and you must earn at least\n";
   cout << "$17,000 per year.\n";
}
```

6.5 Header
Prototype
Function call

6.6
```
void timesTen(int number)
{
   cout << (number * 10);
}
```

6.7
```
void timesTen(int);
```

6.8
```
0       0
1       2
2       4
3       6
4       8
5       10
6       12
7       14
8       16
9       18
```

6.9
```
0 1.5
1.5 0
0 10
0 1.5
```

6.10
```
void showDollars(double amount)
{
   cout << "Your wages are $";
   cout << setprecision(2);
   cout << fixed << showpoint;
   cout << amount << endl;
}
```

6.11 One

6.12
```
double distance(double rate, double time)
```

6.13
```
int days(int years, int months, int weeks)
```

6.14
```
char getKey()
```

6.15
```
long lightYears(long miles)
```

6.16 A static local variable's scope is limited to the function in which it is declared. A global variable's scope is the portion of the program beginning at its declaration to the end.

6.17
```
100
50
100
```

6.18
```
10
11
12
13
14
15
16
17
18
19
```

6.19 Literals or constants

6.20 *Prototype:*
```
void compute(double, int = 5, long = 65536);
```

Header:
```
void compute(double x, int y, long z)
```

6.21 *Prototype:*
```
void calculate(long, double&, int = 47);
```

Header:
```
void calculate(long x, double &y, int z)
```

6.22
```
5 10 15
9 10 15
6 15 15
4 11 16
```

6.23
```
0 00
Enter two numbers: 12 14
12 140
14 15-1
14 15-1
```

6.24
```
1.2
```

6.25
```
30
```

Chapter 7

7.1 A) `int empNums[100];`
B) `float payRates[25];`
C) `long miles[14];`
D) `string cityNames[26];`
E) `double lightYears[1000];`

7.2
```
int readings[-1];  // Size declarator cannot be negative

float measurements[4.5];  // Size declarator must be an integer

int size;
string names[size]; // Size declarator must be a constant
```

7.3 0 through 3

7.4 The size declarator is used in the array declaration statement. It specifies the number of elements in the array. A subscript is used to access an individual element in an array.

7.5 Array bounds checking is a safeguard provided by some languages. It prevents a program from using a subscript that is beyond the boundaries of an array. C++ does not perform array bounds checking.

7.6
```
1
2
3
4
5
```

7.7
```
#include <iostream>
using namespace std;

int main()
{
   const int NUM_FISH = 20;
   int fish[NUM_FISH], count;
```

```
   cout << "Enter the number of fish caught\n";
   cout << "by each fisherman.\n";
   for (count = 0; count < NUM_FISH; count++)
   {
      cout << "fisherman " << (count+1) << ": ";
      cin >> fish[count];
   }
   return 0;
}
```

7.8 A) `int ages[10] = {5, 7, 9, 14, 15, 17, 18, 19, 21, 23};`

B) `float temps[7] = {14.7, 16.3, 18.43, 21.09, 17.9, 18.76, 26.7};`

C) `char alpha[8] = {'J', 'B', 'L', 'A', '*', '$', 'H', 'M'};`

7.9 The definition of `numbers` is valid.

The declaration of `matrix` is invalid because there are too many values in the initialization list.

The definition of `radii` is valid.

The definition of `table` is invalid. Values cannot be skipped in the initialization list.

The definition of `codes` is valid.

The definition of `blanks` is invalid. An initialization list must be provided when an array is implicitly sized.

7.10 A) 0

B) 3

C) 6

D) 14

7.11 0

7.12
```
10.00
25.00
32.50
50.00
110.00
```

7.13
```
1 18 18
2 4 8
3 27 81
4 52 208
5 100 500
```

7.14 No. An entire array cannot be copied in a single statement with the = operator. The array must be copied element-by-element.

7.15 The address of the array.

7.16 No.

7.17 `ABCDEFGH`

7.18 *(The entire program is shown here.)*

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

// Function prototype here
double avgArray(int []);

int main()
{
   const int SIZE = 10;
   int userNums[SIZE];
```

```
   cout << "Enter 10 numbers: ";
   for (int count = 0; count < SIZE; count++)
   {
      cout << "#" << (count + 1) << " ";
      cin >> userNums[count];
   }
   cout << "The average of those numbers is ";
   cout << avgArray(userNums, SIZE) << endl;
   return 0;
}

// Function avgArray
double avgArray(int arr[])
{
   double total = 0.0, average;
   for (int count = 0; count < 10; count++)
      total += arr[count];
   average = total / 10;
   return average;
}
```

7.19
```
int grades[30][10];
```

7.20 24

7.21
```
sales[0][0] = 56893.12;
```

7.22
```
cout << sales[5][3];
```

7.23
```
int settings[3][5] = {{12, 24, 32, 21, 42},
                      {14, 67, 87, 65, 90},
                      {19,  1, 24, 12, 8}};
```

7.24

2	3	0	0
7	9	2	0
1	0	0	0

7.25
```
void displayArray7(int arr[][7], int rows)
{
   for (int x = 0; x < rows; x++)
   {
      for (int y = 0; y < 7; y++)
      {
         cout << arr[x][y] << " ";
      }
      cout << endl;
   }
}
```

7.26
```
int vidNum[50][10][25];
```

7.27
```
vector
```

7.28
```
vector <int> frogs;
```

7.29
```
vector <float> lizards(20);
```

7.30
```
vector <char> toads(100, 'Z');
```

7.31
```
vector <int> gators;

gators.push_back(27);
```

7.32
```
snakes[4] = 12.897;
```

Chapter 8

8.1 The linear search algorithm simply uses a loop to step through each element of an array, comparing each element's value with the value being searched for. The binary search algorithm, which requires the values in the array to be sorted in order, starts searching at the element in the middle of the array. If the middle element's value is greater than the value being searched for, the algorithm next tests the element in the middle of the first half of the array. If the middle element's value is less than the value being searched for, the algorithm next tests the element in the middle of the last half of the array. Each time the array tests an array element and does not find the value being searched for, it eliminates half of the remaining portion of the array. This method continues until the value is found, or there are no more elements to test. The binary search is more efficient than the linear search.

8.2 10,000

8.3 15

8.4 The items frequently searched for can be stored near the beginning of the array.

Chapter 9

9.1
```
cout << &count;
```

9.2
```
float *fltPtr;
```

9.3 Multiplication operator, pointer definition, indirection operator.

9.4
```
50  60  70
500  300  140
```

9.5
```
for (int x = 0; x < 100; x++)
     cout << *(array + x) << endl;
```

9.6 12040

9.7 A) Valid
B) Valid
C) Invalid. Only addition and subtraction are valid arithmetic operations with pointers.
D) Invalid. Only addition and subtraction are valid arithmetic operations with pointers.
E) Valid

9.8 A) Valid
B) Valid
C) Invalid. `fvar` is a float and `iptr` is a pointer to an `int`.
D) Valid
E) Invalid. `ivar` must be declared before `iptr`.

9.9 A) True
B) False
C) True
D) False

9.10
```
makeNegative (&num);
```

9.11
```
void convert(double *val)
{
   *val *= 2.54;
}
```

9.12 A

9.13
```
ip = new int;
delete ip;
```

9.14
```
ip = new int[500];

delete [] ip;
```

9.15 A pointer that contains the address 0.

9.16
```
char *getInitials()
{
   char *initials = new char[3];
   cout << "Enter your three initials: ";
   cin >> initials[0] >> initials[1] >> initials[2];
   return initials;
}
```

9.17
```
char *getInitials()
{
   char[3] initials;
   cout << "Enter your three initials: ";
   cin >> initials[0] >> initials[1] >> initials[2];
   return initials;
}
```

Chapter 10

10.1

`isalpha`	Returns true (a nonzero number) if the argument is a letter of the alphabet. Returns 0 if the argument is not a letter.
`isalnum`	Returns true (a nonzero number) if the argument is a letter of the alphabet or a digit. Otherwise it returns 0.
`isdigit`	Returns true (a nonzero number) if the argument is a digit 0–9. Otherwise it returns 0.
`islower`	Returns true (a nonzero number) if the argument is a lowercase letter. Otherwise, it returns 0.
`isprint`	Returns true (a nonzero number) if the argument is a printable character (including a space). Returns 0 otherwise.
`ispunct`	Returns true (a nonzero number) if the argument is a printable character other than a digit, letter, or space. Returns 0 otherwise.
`isupper`	Returns true (a nonzero number) if the argument is an uppercase letter. Otherwise, it returns 0.
`isspace`	Returns true (a nonzero number) if the argument is a whitespace character. Whitespace characters are any of the following: space..... ‘ ’ newline..... ‘\n’ tab..... ‘\t’ vertical tab..... ‘\v’ Otherwise, it returns 0.
`toupper`	Returns the uppercase equivalent of its argument.
`tolower`	Returns the lowercase equivalent of its argument.

10.2
```
little = tolower(big);
```

10.3
```
if (isdigit(ch))
      cout << "digit";
   else
      cout << "Not a digit.";
```

10.4
```
A
```

10.5
```
char choice;
do
{
      cout << "Do you want to repeat the program or quit? (R/Q) ";
      cin >> choice;
} while (toupper(choice) != 'R' && toupper(choice) != 'Q');
```

10.6

`strlen`	Accepts a pointer to a string as an argument. Returns the length of the string (not including the null terminator).
`strcat`	Accepts pointers to two strings as arguments. The function appends the contents of the second string to the first string. (The first string is altered, the second string is left unchanged.)
`strcpy`	Accepts pointers to two strings as arguments. The function copies the second string to the first string. The second string is left unchanged.
`strncpy`	Accepts pointers to two strings and an integer argument. The third argument, an integer, indicates how many characters to copy from the second string to the first string. If the String2 has fewer than n characters, String1 is padded with '\0' characters.
`strcmp`	Accepts pointers to two string arguments. If String1 and String2 are the same, this function returns 0. If String2 is alphabetically greater than String1, it returns a positive number. If String2 is alphabetically less than String1, it returns a negative number.
`strstr`	Searches for the first occurrence of String2 in String1. If an occurrence of String2 is found, the function returns a pointer to it. Otherwise, it returns a NULL pointer (address 0).

10.7
```
4
```

10.8
```
Have a nice day
nice day
```

10.9
```
strcpy(composer, "Beethoven");
```

10.10
```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   char place[] = "The Windy City";
   if (strstr(place, "Windy"))
      cout << "Windy found.\n";
   else
      cout << "Windy not found.\n";
   return 0;
}
```

10.11

`atoi`	Accepts a string as an argument. The function converts the string to an integer and returns that value.
`atol`	Accepts a string as an argument. The function converts the string to a long integer and returns that value.
`atof`	Accepts a string as an argument. The function converts the string to a float and returns that value.
`itoa`	Converts an integer to a string. The first argument is the integer. The result will be stored at the location pointed to by the second argument, String. The third argument is an integer. It specifies the numbering system that the converted integer should be expressed in. (8 = octal, 10 = decimal, 16 = hexadecimal, etc.)

10.12
```
num = atoi("10");
```

10.13
```
num = atol("10000");
```

10.14
```
num = atof("7.2389");
```

10.15
```
itoa(127, Value, 10);
```

10.16
```
Tom Talbert Tried Trains
Dom Dalbert Dried Drains
```

Chapter 11

11.1
```
struct Account
{
   string acctNum;
   double acctBal;
   double intRate;
   double avgBal;
};
```

11.2
```
Account savings = {"ACZ42137-B12-7",
                   4512.59,
                   0.04,
                   4217.07 };
```

11.3
```
#include <iostream>
#include <string>
using namespace std;

struct Movie
{
    string name;
    string director;
    string producer;
    string year;
};

int main()
{
    Movie favorite;

    cout << "Enter the following information about your\n";
    cout << "favorite movie.\n";
    cout << "Name: ";
    getline(cin, favorite.name);
```

```
    cout << "Director: ";
    getline(cin, favorite.director);

    cout << "Producer: ";
    getline(cin, favorite.producer);

    cout << "Year of release: ";
    getline(cin, favorite.year);

    cout << "Here is information on your favorite movie:\n";
    cout << "Name: " << favorite.name << endl;
    cout << "Director: " << favorite.director << endl;
    cout << "Producer: " << favorite.producer << endl;
    cout << "Year of release: " << favorite.year << endl;
    return 0;
}
```

11.4
```
Product items[100];
```

11.5
```
for (int x = 0; x < 100; x++)
{
   items[x].description = "";
   items[x].partNum = 0;
   items[x].cost = 0;
}
```

11.6
```
items[0].description = "Claw Hammer";
items[0].partNum = 547;
items[0].cost = 8.29;
```

11.7
```
for (int x = 0; x < 100; x++)
{
   cout << items[x].description << endl;
   cout << items[x].partNum << endl;
   cout << items[x].cost  << endl << endl;
}
```

11.8
```
struct Measurement
{
   int miles;
   long meters;
};
```

11.9
```
struct Destination
{
   string city;
   Measurement distance;
};
Destination place;
```

11.10
```
place.city = "Tupelo";
place.distance.miles = 375;
place.distance.meters = 603375;
```

11.11
```
void showRect(Rectangle r)
{
   cout << r.length << endl;
   cout << r.width << endl;
}
```

11.12
```
void getRect(Rectangle &r)
{
   cout << "Width: ";
   cin >> r.width;
   cout << "length: ";
   cin >> r.length;
}
```

11.13
```
Rectangle getRect()
{
   Rectangle r;
   cout << "Width: ";
   cin >> r.width;
   cout << "length: ";
   cin >> r.length;
   return r;
}
```

11.14
```
Rectangle *rptr;
```

11.15 B

11.16
```
union ThreeTypes
{
   char letter;
   int whole;
   double real
};
```

11.17
```
ThreeTypes Items[50];
```

11.18
```
for (int x = 0; x < 50; x++)
   items[x].real = 2.37;
```

11.19
```
for (int x = 0; x < 50; x++)
   items[x].letter = 'A';
```

11.20
```
for (int x = 0; x < 50; x++)
   items[x].whole = 10;
```

11.21 The integers 0, 1, and 2.

11.22 0 7 8

11.23 It is anonymous.

11.24 Z is not greater than X.

11.25 The code will not compile. The assignment statement should be written as:
```
c = static_cast<Color>(0);
```

11.26 The code will not compile. The statement c++ should be written as:
```
c = static_cast<Color>(c + 1);
```

Chapter 12

12.1 `ios::app`

12.2 Place the | operator between them.

12.3 `diskInfo.open("names.dat", ios::out);`

12.4 `diskInfo.open("customers.dat", ios::out | ios::app);`

12.5 `diskInfo.open("payable.dat", ios::in | ios::out);`

12.6
```
fstream dataFile("salesfigures.txt", ios::in);
```

12.7
```
Run
Spot
run
See
spot
run
```

12.8 The >> operator considers whitespace characters as delimiters, and does not read them. The `getline()` member function does read whitespace characters.

12.9
```
100.28      1.72      8.60      7.78      5.10
```

12.10 `seekg` moves the file's read position (for input) and `seekp` moves the file's write position (for output).

12.11 `tellg` reports the file's read position and `tellp` reports the files write position.

12.12 `ios::beg` The offset is calculated from the beginning of the file

`ios::endThe offset is calculated from the end of the file`

`ios::currThe offset is calculated from the current position`

12.13 0

12.14 `file.seekp(100L, ios::beg);`
Moves the write position to the 101st byte (byte 100) from the beginning of the file.

`file.seekp(-10L, ios::end);`
Moves the write position to the 11th byte (byte 10) from the end of the file.

`file.seekp(-25L, ios::cur);`
Moves the write position backward to the 26th byte from the current position.

`file.seekp(30L, ios::cur);`
Moves the write position to the 30th byte (byte 31) from the current position.

12.15 `file.open("info.dat", ios::in | ios::out);`
Input and output.

`file.open("info.dat", ios::in | ios::app);`
Input and output. Output will be appended to the end of the file.

`file.open("info.dat", ios::in | ios::out | ios::ate);`
Input and output. If the file already exists, the program goes immediately to the end of the file.

`file.open("info.dat", ios::in | ios::out);`
Input and output, binary mode.

Chapter 13

13.1 False

13.2 B

13.3 A

13.4 C

13.5
```
class Date
{
private:
   int month;
   int day;
   int year;
```

```
public:
   // Mutators
   void setMonth(int m)
      { month = m; }
   void setDay(int d)
      { day = d; }
   void setYear(int y)
      { year = y; }

   // Accessors
   int getMonth() const
      { return month; }
   int getDay() const
      { return day; }
   int getYear() const
      { return year; }
};
```

13.6 To prevent code outside the class from directly accessing the member variable. This protects the variables from being accidentally modified or used in a way that might adversely affect the state of the object.

13.7 Through public member functions.

13.8 A class specification file is a header file (with a name that ends in `.h`) that contains a class declaration. A class implementation file is a `.cpp` file that contains a class's member function definitions.

13.9 To prevent the contents of a header file from being included more than once in a program.

13.10 The `BasePay` class declaration would reside in `basepay.h`

The `BasePay` member function definitions would reside in `basepay.cpp`

The `Overtime` class declaration would reside in `overtime.h`

The `Overtime` member function declarations would reside in `overtime.cpp`

13.11 A member function whose body is coded in the class declaration.

13.12 A constructor is automatically called when the class object is created. It is useful for initializing member variables, or performing set-up operations.

13.13 A destructor is automatically called before a class object is destroyed. It is useful for performing housekeeping operations, such as freeing memory that was allocated by the class object's member functions.

13.14 A

13.15 B

13.16 B

13.17 A

13.18 True

13.19 True

13.20 False

13.21
```
10
20
50
```

13.22
```
4
7
2
```

2
7
4

13.23 4 (This line is displayed by constructor #2)
7 (This line is displayed by constructor #1)
2 (This line is displayed by constructor #2)
2 (This line is displayed by the destructor)
7 (This line is displayed by the destructor)
4 (This line is displayed by the destructor)

13.24 Some member functions are meant for internal processing, and should be called only from other member functions of the same class. Such member functions may have adverse effects if they are called at the wrong time. For example, a function might initialize member variables, or destroy a member variable's contents. To prevent a member function from being called at the wrong time, it can be made private. Then, it can only be called from another member function, which can determine if it is appropriate to call the private function.

13.25
```
const int SIZE = 3;
InventoryItem items[SIZE];
```

13.26
```
#include <iostream>
using namespace std;

class Yard
{
private:
      int length, width;
public:
      Yard()
         { length = 0; width = 0; }
      setLength(int len)
         { length = len; }
      setWidth(int w)
         { width = w; }
};

int main()
{
      Yard lawns[10];
      cout << "Enter the length and width of "
           << "each yard.\n";
      for (int count = 0; count < 10; count++)
      {
         int input;
         cout << "Yard " << (count + 1) << ":\n";
         cout << "Length: ";
         cin >> Input;
         lawns[count].setLength(input);
         cout << "width: ";
         cin >> input;
         lawns[count].setWidth(input);
      }
   return 0;
}
```

13.27 The problem domain is the set of real-world objects, parties, and major events related to a problem.

13.28 Someone who has an adequate understanding of the problem. If you adequately understand the nature of the problem you are trying to solve, you can write a description of the problem domain yourself. If you do not thoroughly understand the nature of the problem, you should have an expert write the description for you.

13.29 Identify all the nouns (including pronouns and noun phrases) in the problem domain description. Each of these is a potential class. Then, refine the list to include only the classes that are relevant to the problem.

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

13.31 It is often helpful to ask the questions "In the context of this problem, what must the class know? What must the class do?"

13.32 No. Often responsibilities are discovered through brainstorming.

13.33 A) We begin by identifying the nouns: doctor, patients, practice, patient, procedure, description, fee, statement, office manager, name, address, and total charge. After eliminating duplicates, objects, and primitive variables, the list of potential classes is: *Doctor*, *Practice*, *Patient*, *Procedure*, *Statement*, and *Office manager*.

B) The necessary classes for this problem are: *Patient*, *Procedure*, and *Statement*.

C) The *Patient* class knows the patient's name and address.

The *Procedure* class knows the procedure description and fee.

The *Statement* class knows each procedure that was performed.

The *Statement* class can calculate the total charges.

Chapter 14

14.1 Each class object (an instance of a class) has its own copy of the class's instance member variables. If a class's member variable is static, however, only one instance of the variable exists in memory. All objects of that class have access to that one variable.

14.2 Outside the class declaration.

14.3 Before.

14.4 Static member functions can only access member variables that are also static.

14.5 You can call a static member function before any instances of the class have been created.

14.6 No, but it has access to class `X`'s private members, just as if it were a member.

14.7 Class `X`.

14.8 Each member of one object is copied to its counterpart in another object of the same class.

14.9 When one object is copied to another with the = operator, and when one object is initialized with another object's data.

14.10 When an object contains a pointer to dynamically allocated memory.

14.11 When an object is initialized with another object's data, and when an object is passed by value as the argument to a function.

14.12 It has a reference parameter object of the same class type as the constructor's class.

14.13 It performs memberwise assignment.

14.14
```
void operator=(const Pet &);
```

14.15
```
dog.operator=(cat);
```

14.16 It cannot be used in multiple assignment statements or other expressions.

14.17 It is a built-in pointer, available to a class's member functions, that always points to the instance of the class making the function call.

14.18 Non-static member functions

14.19 `cat` is calling the operator+ function. `tiger` is passed as an argument.

14.20 The operator is used in postfix mode.

14.21 They should always return `true` or `false` values.

14.22 The object may be directly used with `cout` and `cin`.

14.23 An `ostream` object.

14.24 An `istream` object.

14.25 The operator function must be declared as a `friend`.

14.26 `list1.operator[](25);`

14.27 The object whose name appears on the right of the operator in the expression.

14.28 So statements using the overloaded operators may be used in other expressions.

14.29 The postfix version has a dummy parameter.

14.30 *(Overloaded operator functions)*

```
// Overloaded prefix -- operator
FeetInches FeetInches::operator--()
{
  --inches;
  simplify();
  return *this;
}

// Overloaded postfix -- operator
FeetInches FeetInches::operator--(int)
{
  FeetInches temp(feet, inches);
  inches--;
  simplify();
  return temp;
}
```

```
(Demonstration program)

// This program demonstrates the prefix and postfix —
// operators, overloaded to work with the FeetInches class.

#include <iostream>
#include "FeetInches.h"
using namespace std;

int main()
{
  FeetInches distance;

  cout << "Enter a distance in feet and inches: ";
  cin >> distance;
  cout << "Demonstrating the prefix — operator: \n";
  cout << "Here is the value: " << --distance << endl;
  cout << "Demonstrating the postfix — operator: \n";
  cout << "Here is the value: " << distance-- << endl;
  cout << "Here is the final value: " << distance << endl;
  return 0;
}
```

14.31 Objects are automatically converted to other types. This ensures that an object's data is properly converted.

14.32 They always return a value of the data type they are converting to.

14.33 `BlackBox::operator int()`

14.34 The `Big` class "has a" `Small` class as its member.

Chapter 15

15.1 The base class is `Vehicle`.

15.2 The derived class is `Truck`.

15.3 A) The `radius` variable. (The `area` variable is inherited, but inaccessible.)

B) The `setArea` function (inherited)

The `getArea` function (inherited)

The `setRadius` function

The `getRadius` function

C) The `area` variable

15.4 Protected members may be accessed by derived classes. Private members are inaccessible to derived classes.

15.5 Member access specification determines if a class member is accessible to statements outside the class. Class access specification determines how the derived class inherits members of the base class.

15.6 A) `a` is inaccessible, the rest are private.

B) `a` is inaccessible, the rest are protected.

C) `a` is inaccessible, `b`, `c`, and `setA` are protected, `setB` and `setC` are public.

D) private

15.7
```
Entering the Sky
Entering the Ground
Leaving the Ground
Leaving the Sky
```

15.8
```
Entering the Sky
Entering the Ground
Leaving the Ground
Leaving the Sky
```

15.9 An overloaded function is one with the same name as one or more other functions, but with a different parameter list. The compiler determines which function to call based on the arguments used. Redefining occurs when a derived class has a function with the same name as a base class function. The two functions can have the same parameter list. Objects that are of the derived class always call the derived class's version of the function, while objects that are of the base class always call the base class's version.

15.10 Static binding means the compiler binds a function call to a function at compile time. Dynamic binding means a function call is bound to a function at runtime.

15.11 Dynamically

15.12
```
1
5
```

15.13
```
2
2
```

15.14
```
2
1
```

15.15 2

15.16 Chain of inheritance

15.17 Multiple inheritance

15.18 A) inaccessible
B) protected
C) protected
D) inaccessible
E) protected
F) public
G) private
H) protected
I) public

15.19
```
class SportUtility : public van, public FourByFour
{
};
```

Chapter 16

16.1 The try block contains one or more statements that may directly or indirectly throw an exception. The catch block contains code that handles, or responds to an exception.

16.2 The entire program will abort execution.

16.3 Each exception must be of a different type. The catch block whose parameter matches the data type of the exception handles the exception.

16.4 With the first statement after the try/catch construct.

16.5 By giving the exception class a member variable, and storing the desired information in the variable. The throw statement creates an instance of the exception class, which must be caught by a catch statement. The catch block can then examine the contents of the member variable.

16.6 When it encounters a call to the function.

16.7
```
template <class T>
double half(T number)
{
   return number / 2.0;
}
```

16.8 That the operator has been overloaded by the class object.

16.9 First write a regular, non-template version of the function. Then, after testing the function, convert it to a template.

16.10
```
List<int> myList;
```

16.11
```
template <class T>
class Rectangle
{
   private:
      T width;
      T length;
      T area;
   public:
      void setData(T w, T l)
         { width = w; length = l;}
      void calcArea()
         { area = width * length; }
```

```
        T getWidth()
           { return width; }
        T getLength()
           { return length; }
        T getArea()
           { return area; }
};
```

Chapter 17

17.1 A data member contains the data stored in the node. A pointer points to another node in the list.

17.2 A pointer to the first node in the list.

17.3 The last node in the list will point to the NULL address.

17.4 A data structure that contains a pointer to an object of the same data structure type.

17.5 Appending a node is adding a new node to the end of the list. Inserting a node is adding a new node in a position between two other nodes.

17.6 Appending

17.7 Because the new node is being inserted between two other nodes, `previousNode` points to the node that will appear before the new node.

17.8 A) Remove the node from the list without breaking the links created by the `next` pointers

B) Delete the node from memory

17.9 Because there is probably a node pointing to the node being deleted. Additionally, the node being deleted probably points to another node. These links in the list must be preserved.

17.10 The unused memory is never freed, so it could eventually be used up.

Chapter 18

18.1 Last-in-first-out. The last item stored in a LIFO data structure is the first item extracted.

18.2 A static stack has a fixed size, and is implemented as an array. A dynamic stack grows in size as needed, and is implemented as a linked list. Advantages of a dynamic stack: There is no need to specify the starting size of the stack. The stack automatically grows each time an item is pushed, and shrinks each time an item is popped. Also, a dynamic stack is never full (as long as the system has free memory).

18.3 Push: An item is pushed onto, or stored in the stack.

Pop: An item is retrieved (and hence, removed) from the stack.

18.4 `vector`, `list`, or `deque`.

Chapter 19

19.1 The function calls itself with no way of stopping. It creates an infinite recursion.

19.2 The solvable problem that the recursive algorithm is designed to solve. When the recursive algorithm reaches the base case, it terminates.

19.3 10

19.4 In direct recursion, a recursive function calls itself. In indirect recursion, function A calls function B, which in turn calls function A.

Chapter 20

20.1 A standard linked list is a linear data structure in which one node is linked to the next. A binary tree is non-linear, because each node can point to two other nodes.

20.2 The first node in the list.

20.3 A node pointed to by another node in the tree.

20.4 A node that points to no other nodes.

20.5 An entire branch of the binary tree, from one particular node down.

20.6 Information can be stored in a binary tree in a way that makes a binary search simple.

20.7
1. The node's left subtree is traversed.
2. The node's data is processed.
3. The node's right subtree is traversed.

20.8
1. The node's data is processed
2. The node's left subtree is traversed.
3. The node's right subtree is traversed.

20.9
1. The node's left subtree is traversed.
2. The node's right subtree is traversed.
3. The node's data is processed.

20.10 The node to be deleted is node D.
1. Find node D's parent and set the child pointer that links the parent to node D, to NULL.
2. Free node D's memory.

20.11 The node to be deleted is node D.
1. Find node D's parent.
2. Link the parent node's child pointer (that points to node D) to node D's child.
3. Free node D's memory.

20.12
1. Attach the node's right subtree to the parent, and then find a position in the right subtree to attach the left subtree.
2. Free the node's memory.