APPENDIX P

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 (RAM), secondary storage devices, input devices, and 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 storage location 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 A single tasking operating system is capable of running only one program at a time. All the computer's resources are devoted to the program that is running. A multitasking operating system is capable of running multiple programs at once. A multitasking system divides the allocation of the hardware resources among the running programs using a technique known as time sharing.

1.9 A single user system allows only one user to operate the computer at a time. A multiuser system allows several users to operate the computer at once.

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

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

1.12 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 resembles natural languages.

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

1.14 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 to create an executable file.

1.15 Source file: Contains program statements written by the programmer.

Object file: Contains machine language instructions generated by the compiler.

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

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

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

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

1.19 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. It may be written on 1 or more lines.

1.20 Because their contents may be changed while the program is running.

1.21 The original value is overwritten.

1.22 The variable must be defined in a declaration.

1.23 Input, processing, and output

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

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

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

1.27 A "language" that is a cross between human language and programming languages that is used to express algorithms.

1.28 High-level psuedocode just lists the steps a program must carry out. Detailed psuedocode shows the variables, logic, and computations needed to create the program.

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

1.30 A mistake that causes a program to produce erroneous results. A logic error occurs when what the programmer means for the program to do does not match what the code actually instructs the program to do.

1.31 An error that occurs while the program is running when the system is asked to perform an action it cannot carry out.

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

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
```
// Insert current date here
#include <iostream>
using namespace std;

int main()
{
   cout << "Teresa Jones";
   return 0;
}
```

2.3
```
cout << "red \n" << "blue \n" << "yellow \n" << "green";
```

2.4
```
The works of Wolfgang
include the following:
The Turkish March
and Symphony No. 40 in G minor.
```

2.5
```
#include <iostream>
using namespace std;

int main()
{
   cout << "Teresa Jones\n";
   cout << "127 West 423rd Street\n";
   cout << "San Antonio, TX  78204\n";
   cout << "555-475-1212\n";
   return 0;
}
```

2.6 Only statement **a** is legal. The lefthand side of an assignment statement must be a variable, not a constant.

2.7 Variables: `little` and `big`

Constants: `2`, `2000`, `"The little number is "`, `"The big number is"`, `0`

2.8
```
The little number is 2
The big number is 2000
```

2.9
```
The value is number
```

2.10 `99bottles`: Variable names cannot begin with a number.
`r&d`: Variable names may only use alphabetic letters, digits, and underscores.

2.11 No. Variable names are case sensitive.

2.12 A) `short` or `unsigned short`
B) `int`
C) They both use the same amount of memory.

2.13 `unsigned short`, `unsigned int`, and `unsigned long`

2.14 `int apples 20;`

2.15 67, 70, 87

2.16 'B'

2.17 1 byte, 2 bytes, 6 bytes, 1 byte

2.18 The string literal "`z`" is being stored in the character variable `letter`.

2.19 `string`

2.20
```
// Substitute your name, address, and phone
// number for those shown in this program.
#include <iostream>
#include <string>
using namespace std;

int main()
{
   string name, address, phone;
   name = "George Davis";
   address = "179 Ravenwood Lane";
   phone = "555-6767";
   cout << name << endl;
   cout << address << endl;
   cout << phone << endl;
   return 0;
}
```

2.21 6.31E17

2.22 3

2.23
```
#include <iostream>
using namespace std;

int main()
{
   int age;
   double weight;

   age = 26;
   weight = 168.5;
   cout << "My age is " << age << "and my weight is " << weight;
   cout << weight << " pounds.\n";
   return 0;
}
```

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

2.25 The variable `critter` is assigned a value before it is declared. Correct the program by moving the statement `critter = 62.7;` to the point after the variable declaration.

2.26 11, 5, 24, 2

2.27 Integer division. The value 5 will be stored in `portion`.

Chapter 3

3.1 `iostream`

3.2 The stream extraction operator

3.3 The console (or keyboard)

3.4 True

3.5 3

3.6 `cin >> miles >> feet >> inches;`

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

3.8
```
#include <iostream>
using namespace std;

int main()
{
   double pounds, kilograms;

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

3.9 A) *
B) same
C) same

3.10 *Value*
21
2
31
5
24
2
69
0
30

3.11
```
y = 6 * x;
a = 2 * b + 4 * c;
y = pow(x, 3);
g = (x + 2) / (z * z);  or  g = (x + 2) / pow(z, 2);
y = (x * x) / (z * z);  or  y = pow(x, 2) / pow (z, 2);
```

3.12 *If the user enters…* *The program displays…*

2	6
5	27
4.3	20.49
6	38

3.13
```
#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;
}
```

3.14
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

3.15
```
The ASCII values of uppercase letters are 65 - 90
The ASCII values of lowercase letters are 97 - 122
Enter a letter and I will tell you its ASCII code: B
```
The ASCII code for B is 66

3.16
```
9
9.5
9
```

3.17
```
const double E = 2.71828;
const double MIN_PER_YEAR = 5.256E5;
const double GRAV_ACC_FT_PER_SEC = 32.2;
const double GRAV_ACC_M_PER_SEC = 9.8;
const int METERS_PER_MILE = 1609;
```

3.18
```
#define E 2.71828
#define YEAR_SECS 5.26e5
#define GRAV_ACC_FT_PER_SEC 32.2
#define GRAV_ACC_M_PER_SEC 9.8
#define METERS_PER_MILE 1609
```

3.19
```
This program calculates the number of candy pieces sold.
How many jars of candy have you sold? 6[Enter]
The number of pieces sold: 11160
Candy pieces you get for commission: 2232
```

3.20
```
#include <iostream>
using namespace std;

int main()
{
   const double CONVERSION = 1.467;
   double milesPerHour, feetPerSecond;

   cout << "This program converts miles-per-hour to\n";
   cout << "feet-per-second.\n";
   cout << "Enter a speed in MPH: ";
   cin  >> milesPerHour;
   feetPerSecond = milesPerHour * CONVERSION;
   cout << "That is " << feetPerSecond
        << " feet-per-second.\n";
   return 0;
}
```

3.21
```
total = subtotal = tax = shipping = 0;
```

3.22
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);
```

3.23
```
3
11
1
```

3.24 A)
```
cout << fixed << setprecision(2);
cout << setw(9) << 34.789;
```

B)
```
cout << fixed << showpoint
     << setprecision(3);
cout << setw(5) << 7.0;
```

C)
```
cout << fixed << 5.789e12;
```

D)
```
cout << left << setw(7) << 67;
```

3.25
```
#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 << fixed << showpoint << setprecision(4);
   cout << left << setw(7) << radians << " radians.\n ";
   return 0;
}
```

3.26 No. Space is needed for a fifth character, to hold the null terminator.

3.27 A) Legal (Though no embedded blanks can be input)
B) Illegal (This works for C-strings only)
C) Legal
D) Legal

3.28 A) Legal (Though no embedded blanks can be input)
B) Legal
C) Legal
D) Illegal (Arrays cannot be assigned to variables like this. Use `strcpy()`.)

3.29
```
x = sin(angle1) + cos(angle2);
```

3.30
```
y = pow(x, 0.2);   // 0.2 is equal to 1/5
```

3.31
```
y = 1 / sin(a);
```

3.32 `fstream`

3.33 `ifstream` objects can only be used to read from a file into memory, whereas `ofstream` objects can only be used to write from memory to a file.

3.34 The `open` statement

3.35 C

Chapter 4

4.1 T, T, T, T, T, T, 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 True

4.6 False

4.7 A) Error: There is a semi-colon after the `if` test condition.
Result: The `cout` statement will execute even though `hours` is not greater than 40.
Output: `12 hours qualifies for over-time.`

B) Error: The `if` test condition uses an assignment operator (=) rather than an equality test (==).
Result: `interestRate` will be assigned the value .07 and the `cout` statement will execute even though it shouldn't.
Output: `This account is earning the maximum rate.`

C) Error: The 2 statements that are supposed to be included in the body of the `if` statement are not surrounded by curly braces.
Result: Only the `cout` statement is in the `if` body. The $10 addition to `balance` will always be done, even when `interestRate` is not greater than .07.
Output: `None`

4.8
```
if (y == 20)
   x = 0;
```

4.9
```
if (hours > 40)
   payRate *= 1.5;
```

4.10
```
if (sales >= 10000.00)
   commission = .20;
```

4.11
```
if (max)
   fees = 50;
```

4.12 False

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

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

4.15
```
#include <iostream>
using namespace std;

int main()
{
   double taxRate, saleAmount;
   char residence;

   cout << "Enter the amount of the sale: ";
   cin  >> saleAmount;
   cout << "Enter I for in-state residence or O for out-of-\n";
   cout "state: ";
   cin.get(residence);
   if (residence == 'O')
      taxRate = 0;
   else
      taxRate = 0.05;
   saleAmount += saleAmount * taxRate;
   cout << "The total is " << saleAmount;
   return 0;
}
```

4.16 No. When `x` equals `y` the `if/else` statement causes a 2 to display and the two separate if statments don't display anything.

4.17
```
#include <iostream>
using namespace std;

int main()
{
   int  testScore;
   char grade;
   bool goodScore = true;

   cout << "Enter your numeric test score and I will\n";
   cout << "tell you the letter grade you earned: ";
   cin  >> testScore;

   if (testScore < 0)
      goodScore = false;
   else if (testScore < 60)
      grade = 'F';
   else if (testScore < 70)
      grade = 'D';
   else if (testScore < 80)
      grade = 'C';
   else if (testScore < 90)
      grade = 'B';
   else if (testScore <= 100)
      grade = 'A';
   else
      goodScore = false;
```

```
   if (goodScore)
      cout << "Your grade is " << grade << ".\n";
   else
   {  cout << testScore << " is an invalid score.\n";
      cout << "Please enter a score between 0 and 100.\n";
   }
   return 0;
}
```

4.18 11

4.19

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

4.20

```
if (amount1 > 10)
{  if (amount2 < 100)
   {  if (amount1 > amount2)
         cout << amount1;
      else
   }     cout << amount2;
}
```

4.21

Logical Expression	Result (True or False)
true && false	false
true && true	true
false && false	false
true || false	true
true || true	true
false || false	false
!true	false
!false	true

4.22 T, F, T, T, T

4.23 True (`&&` is done before `||`)

4.24

```
if (!activeEmployee)
```

4.25

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

4.26
```
#include <iostream>
using namespace std;

int main()
{
   int first, second, result;

   cout << "Enter a negative integer: ";
   cin  >> first;
   cout << "Now enter a positive integer: ";
   cin  >> second;
   if (first >= 0 || second < 0)
   {
      cout << "The first number should be negative\n";
      cout << "and the second number should be\n";
      cout << "positive. Run the program again and\n";
      cout << "enter the correct values.\n";
   }
   else
   {
      result = first * second;
      cout << first << " times " << second << " is "
           << result << endl;
   }
   return 0;
}
```

4.27 The variables `length`, `width`, and `area` should be defined before they are used.

4.28
```
Enter your first test score: 40
Enter your second test score: 30
Test 1: 50
Test 2: 40
Sum   : 70
```

4.29
A) True
B) False
C) True
D) False
E) False
F) True

4.30
A) False
B) False
C) True
D) False
E) False
F) True
G) False

4.31
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"));
```

4.32 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 << "Even\n";
else
   cout << "Odd\n";
```

4.33
```
2 2
```

4.34 Because the `if /else` statement tests several different conditions, consisting of different variables and because it tests values with relational operators other than `equal-to`.

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

4.36
```
That is serious.
```

4.37
```
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";
}
```

4.38
```
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";
}
```

4.39 `enum` must be lowercase. There should be no = sign. The symbolic names in the enumeration list should not be in quotes. It should end with a semicolon.

4.40
```
if (color <= yellow)
   cout "primary color \n";
else
   cout "mixed color \n";
```

Chapter 5

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

5.2 A) 4
B) 0
C) 0 Notice the semi-colon after the `while` test expression. This causes an infinite loop that prints nothing.
D) 0 Notice the missing braces. This causes an infinite loop that prints "My favorite day is" over and over.

5.3 `count`

5.4 A)
```
cout << " Enter a menu choice between 1 and 4: ";
cin  >> choice;
while (choice < 1 || choice > 4)
{  cout << "Choice must be between 1 and 4. Re-enter choice: ";
   cin  >> choice;
}
```
B)
```
cout << " Enter Y or N: ";
cin  >> reply;
while (reply != 'Y' && reply != 'y' &&
       reply != 'N' && reply != 'n')
{  cout << "Invalid input. Please enter Y or N: ";
   cin  >> reply;
}
```

5.5 A) Hello World B) 01234 C) 6 10 5

5.6
```
do
{  cout << "Enter an integer: ";
   cin  >> num;

   if (num % 2 == 0)
      cout << "That integer is even.\n";
   else
      cout << "That integer is odd.\n";

   cout << "Do you want to test another number (y/n)? ";
   cin  >> reply;
} while (reply == 'y' || reply == 'Y');
```

5.7 initialization expression, test expression, and update expression

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

5.9 A) 0 2 4 6 8 10
B) -5 -4 -3 -2 -1 0 1 2 3 4
C) 3 6 9 12

5.10
```
for (int count = 1; count <= 10; count++)
   cout << "Put your name here.\n";
```

5.11
```
for (int num = 1; num < 50; num += 2)
   cout << num << endl;
```

5.12
```
for (int num = 0; num <= 100; num += 5)
   cout << num << endl;
```

5.13 x is the counter, y is the accumulator.

5.14
```
int sum = 0;
for (int num = 1; num <= 10; num++)
   sum += num * num;
cout << "The sum of the squares of the integers \n"
     << "from 1 through 10 is " << sum << endl;
```

5.15
```
int sum = 0;
for (int num = 1; num <= 9; num += 2)
   sum += num * num;
cout << "The sum of the squares of the odd integers \n"
     << "from 1 through 9 is " << sum;
```

5.16
```
int count, number, total = 0;
for (count = 0; count < 7; count++)
{
   cout << "Enter a number: ";
   cin  >> number;
   total += number;
}
cout << "The total is " << total << endl;
```

5.17
```
double x, y, quotient, total = 0.0;
for (x = 1, y = 30; x <= 30; x++, y--)
{
   quotient = x / y;
   total += quotient;
}
cout << "The total is " << total << endl;
```

5.18
```
double total = 0.0;
for (int denom = 2; denom <= 1024; denom *= 2)
   total += 1.0 / denom;
cout << "The total of the series is " << total << endl;
```

5.19
```
int score, numScores = 0;
double total = 0.0;

cout << "Enter the first test score (or -99 to quit): ";
cin  >> score;

while (score != -99)
{  numScores++;
   total += score;
   cout << "Enter the next test score (or -99 to quit): ";
   cin  >> score;
}
if (numScores == 0)
   cout << "No scores were entered." << endl;
else
   cout << "The average of the " << numScores
        << " scores is " << total / numScores << endl;
```

5.20 A) `for`
B) `do-while`
C) `while`
D) `while`
E) `for`

5.21 A) 600 times (20 times 30)
B) 220 times (20 times 11)

5.22
```
1
3
7
12
```

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 pay)
{
   cout << fixed << showpoint << setprecision(2);
   cout << "Your wages are $" << pay << 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 defined. A global variable's scope is the portion of the program from its definition to the end of the program.

6.17
```
100
50
100
```

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

6.19 Literals and 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 Different parameter lists

6.25
```
1.2
```

6.26
```
30
```

Chapter 7

7.1 B

7.2 A

7.3 C

7.4
```
class Date
{
private:
   int month;
   int day;
   int year;
public:
   void setDate(int m, int d, int y)
       { month = m; day = d; year = y; }
   int getMonth()
       { return month; }
   int getDay()
       { return day; }
   int getYear()
       { return year; }
}
```
Alternately these could be separate `setMonth`, `setDay`, and `setYear` member functions to validate and set each component of the date separately.

7.5 A constructor is automatically called when the class object is created. It is useful for initializing member variables or performing setup operations.

7.6 A

7.7 A

7.8 `ClassAct sally(25);`

7.9 True

7.10 False

7.11 B

7.12 False

7.13
```
50
50
20
```

7.14
```
4
7
goodbye
```
goodbye

7.15 D

7.16 A

7.17 B

7.18 False. They can be both passed to functions and returned by functions.

7.19 False. Passing it by value *will* ensure it is not changed, but it is best to pass it by constant reference.

7.20 D

7.21
```
class Circle
{  private:
      double radius;          // In inches
   public:
      void setRadius(double r)
      {  radius = r; }
      double getArea()        // In sq. in.
      {  return (3.14.159 * radius * radius); }
}
```

7.22
```
class Pizza
{  private:
      double price;
      Circle size;
   public:
      void setPrice(double p)
      {   price = p; }
      void setSize(double r)
      {   size.setRadius(r); }
      double costPerSqIn()
      {   return (price / size.getArea()); }
}
```

7.23
```
// Student might use other prices and sizes
Pizza myPizza;
myPizza.setPrice(12.99);
myPizza.setSize(14);
cout << "Price per square inch $" << myPizza.costPerSqIn();
```

7.24 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`

7.25 `Basepay.h` and `Overtime.h`

7.26
```
struct Student
{  int   id,
         entryYear;
   double gpa;
};
Student s1(1234, 2008, 3.41);
Student s2(5678, 2010);
```

7.27
```
struct Account
{  string acctNum;
   double acctBal,
          intRate,
          avgBal;

   Account(string num, double bal, double rate, double avg)
   {   acctNum = num;  acctBal = bal;
       intRate = rate; avgBal = avg;
   }
};
Account savings("ACZ42137", 4512.59, .04, 4217.07);
```

7.28
```
#include <iostream>
#include <string>
using namespace std;

struct MovieInfo
{
   string name,
          director;
   int    year;
};

int main()
{
   MovieInfo movie;

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

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

   cout << "Year of Release: ";
   cin  >> movie.year;

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

7.29
```
struct Measurement
{
   int miles;
   long meters;
}
```

7.30
```
struct Destination
{
   string city;
   Measurement distance;
};
Destination place;
```

7.31
```
place.city = "Tupelo";
place.distance.miles = 375;
place.distance.meters = 603375;
```

7.32
```
void showRect(Rectangle r)
{
   cout << r.length << endl;
   cout << r.width << endl;
}
```

7.33
```
void getRect(Rectangle &r)
{
   cout << "Width: ";
   cin  >> r.width;
   cout << "Length: ";
   cin  >> r.length;
}
```

7.34
```
Rectangle getRect()    // Function return type is a Rectangle structure
{
   Rectangle r;
   cout << "Width: ";
   cin  >> r.width;
   cout << "Length: ";
   cin  >> r.length;
   return r;
}
```

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

7.36 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.

7.37 Start by identifying 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.

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

7.39 A) 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 simple data items that can be stored in 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 8

8.1 A) `int empNum[100];`
B) `double payRate[25];`
C) `long miles[14];`
D) `char letter[26];`
E) `double lightYears[1000];`

8.2
```
int readings[-1];         // Size declarator cannot be negative
float measurements[4.5];  // Size declarator must be an integer
int size;                 // This is not an array
char name[size];          // Size declarator must be a constant
```

8.3 0 through 3

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

8.5 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.

8.6
```
1
2
3
4
5
```

8.7
```
#include <iostream>
using namespace std;

int main()
{
   const int NUM_MEN = 10;
   int fish[NUM_MEN], count;

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

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

8.9 A)
```
int numbers[10] = {0, 0, 1, 0, 0, 1, 0, 0, 1, 1};
```
The definition is valid.

B)
```
int matrix[5] = {1, 2, 3, 4, 5, 6, 7};
```
The definition is invalid because there are too many values in the initialization list.

C)
```
double radii[10] = {3.2, 4.7};
```
The definition is valid. Elements 2 through 9 will be initialized to 0.0.

D)
```
int table[7] = {2, , , 27, , 45, 39};
```
The definition is invalid. Values cannot be skipped in the initialization list.

E)
```
char codes[] = {'A', 'X', '1', '2', 's'};
```
The definition is valid. The `codes` array will be allocated space for five characters.

F) `int blanks[];`
The definition is invalid. An initialization list must be provided when an array is implicitly sized.

G) `string suit[4] = {"Clubs", "Diamonds", "Hearts", "Spades"};`
The definition is valid.

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

8.11 A) `10`
B) `3`
C) `6`
D) `14`

8.12 `0`

8.13
```
10.00
25.00
32.50
50.00
110.00
```

8.14
```
1 18 18
2 4 8
3 27 81
4 52 208
5 100 500
```

8.15 `typedef int TenInts[10];`

8.16 The starting address of the array

8.17 `ABCDEFGH`

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

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

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

8.19
```
int grades[30][10];
```

8.20 24

8.21
```
sales[0][0] = 56893.12;
```

8.22
```
cout << sales[5][3];
```

8.23
```
int settings[3][5] = {{12, 24, 32, 21, 42},
                      {14, 67, 87, 65, 90},
                      {19,  1, 24, 12,  8}};
```

8.24

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

8.25
```
void displayArray7(int array[][7], int numRows)
{
   for (int row = 0; row < numRows; row ++)
   {
      for (int col = 0; col < 7; col ++)
      {  cout << array[row][col] << " ";
      }
      cout << endl;
   }
}
```

8.26
```
int vidNum[50][10][25];
```

8.27 vector

8.28
```
vector <int> frogs;
vector <float> lizards(20);
vector <char> toads(100, 'Z');
```

8.29
```
vector <int> gators;
vector <double> snakes(10);
gators.push_back(27);
snakes[4] = 12.897;
```

8.30 False

8.31 False

8.32
```
10
20
50
```

8.33
```
#include <iostream>
using namespace std;

class Yard
{
private:
   int length, width;
public:
   Yard()
      { length = 0; width = 0; }
   void setLength(int len)
      { length = len; }
   void setWidth(int wide)
      { width = wide; }
   int getLength() {return length;}
   int getWidth() {return width;}
};

int main()
{
   const int SIZE = 10;
   Yard lawns[SIZE];
   cout << "Enter the length and width of "
        << "each yard.\n";

   for (int count = 0; count < SIZE; count++)
   {
      int input;
      cout << "Yard " << (count + 1) << ":\n";
      cout << "length: ";
      cin  >> input;
      lawns[count].setLength(input);
      cout << "width: ";
      cin  >> input;
      lawns[count].setWidth(input);
   }
   cout << "\nHere are the yard dimensions.\n";
   for (int yard = 0; yard < SIZE; yard++)
   {
      cout << "Yard " << (yard+1) << " "
           << lawns[yard].getLength() << " X "
           << lawns[yard].getWidth()  << endl;
   }
   return 0;
}
```

8.34
```
Product()                              // Default constructor
{  description = "";
   partNum = cost = 0;
}
Product(string d, int p, double c)   // Constructor
{  description = d;
   partNum = p;
   cost = c;
}
```

8.35
```
Product items[100];
```

8.36
```
items[0].description = "Claw Hammer";
items[0].partNum = 547;
items[0].cost = 8.29;
```

8.37
```
for (int x = 0; x < 100; x++)
{
   cout << items[x].description << endl;
   cout << items[x].partNum << endl;
   cout << items[x].cost  << endl << endl;
}
```

8.38
```
Product items[5] = { Product("Screw driver", 621,  1.72),
                     Product("Socket set",   892, 19.97),
                     Product("Claw hammer",  547,  8.29) };
```

8.39
```
struct Measurement
{
   int miles;
   long meters;
};
```

8.40
```
struct Destination
{
   string city;
   Measurement distance;
};
```

8.41
```
Destination places [20];
places[4].city = "Tupelo";
places[4].distance.miles = 375;
places[4].distance.meters = 603375;
```

Chapter 9

9.1 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.

9.2 10,000

9.3 14

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

9.5 True

9.6 Change the > sign in the `if` statement to a < sign. The line would now read

```
If (array[count] < array[count + 1])
```

9.7 The last value is now in order.

9.8 The first value, in position 0, is now in order.

9.9 selection sort

9.10 A basic operation is one that requires constant time, regardless of the size of the problem that is being solved.

9.11 The worst case complexity function $f(n)$ of an algorithm is a measure of the time required by the algorithm to solve a problem instance of size n that requires the most time.

9.12 Because $10n$ and $25n$ differ by a constant factor and constant factors are not significant, the two algorithms are considered to be equivalent in efficiency.

9.13 To say that $f(n)$ is in $O(g(n))$ means that there exists a positive constant K such that $f(n) \leq Kg(n)$ for all $n \geq 1$. This means that for large problem sizes, an algorithm with complexity function $f(n)$ is no worse than one with complexity function $g(n)$.

9.14 To show that $100n^3 + 50n^2 + 75$ is in $O(20n^3)$, we must show that some constant K exists for which $100n^3 + 50n^2 + 75 \leq K(20n^3)$ for all $n \geq 1$.

Observe that for all $n \geq 1$

$$\frac{100n^3 + 50n^2 + 75}{20n^3} = 5 + \frac{5}{2n} + \frac{75}{20n^3} \leq 5 + 5 + 75 \leq 85$$

Therefore, we have found a constant K that satisfies the inequality, namely $K = 85$.

9.15 Assuming that $g(n) \geq 1$ for all $n \geq 1$, we have $100 \leq 100\, g(n)$ for all $n \geq 1$. This implies that $g(n) + 100 \leq g(n) + 100g(n) = 101g(n)$ for all $n \geq 1$. Now, if $f(n)$ is in $O(g(n)+100)$, there exists a positive K such that $f(n) \leq K(g(n)+100) \leq 101Kg(n)$ for all $n \geq 1$. Taking $K_1 = 101K$, we see that $f(n) \leq K_1 g(n)$ for all $n \geq 1$.

Chapter 10

10.1
```
cout << &count;
```

10.2
```
double *dPtr;
```

10.3 Multiplication operator, pointer declaration, indirection operator

10.4
```
50  60  70
500  300  140
```

10.5
```
for (int x = 0; x < 100; x++)
     cout << *(array + x) << endl;
```

10.6 12040

10.7 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

10.8 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 defined before it is used.

10.9 A) True
B) False
C) True
D) False

10.10
```
makeNegative (&num);
```

10.11
```
void convert(double *val)
{
   *val *= 2.54;
}
```

10.12 A

10.13
```
ip = new int;
delete ip;
```

10.14
```
ip = new int[500];
delete [] ip;
```

10.15 A pointer whose value is the address 0

10.16
```
char *getname(char *name)
{
   cout << "Enter your name: ";
   cin.getline(name, 81);
   return name;
}
```

10.17
```
char *getname()
{
   char name[81];
   cout << "Enter your name: ";
   cin.getline(name, 81);
   return name;
}
```

10.18
```
Rectangle *rptr;
```

10.19
```
cout << rptr->length << endl << rptr->width << endl;
```

10.20 B

Chapter 11

11.1 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 copy of the variable exists in memory. All objects of that class have access to that one variable.

11.2 Outside the class declaration

11.3 Before

11.4 Static member functions can not access instance members unless they explicitly specify an object of the class.

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

11.6 No, but it has access to all of class X's members, just as if it were a member.

11.7 Class X

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

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

11.10 When an object contains a pointer to dynamically allocated memory

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

11.12 The member function has the same name as the class, has no return type, and has a single reference parameter to the same type as the class.

11.13 It performs memberwise assignment.

11.14 `Pet Pet :: operator=(const Pet);`

11.15 `dog.operator=(cat);`

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

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

11.18 Instance member functions

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

11.20 The operator is used in postfix mode.

11.21 They should always return boolean values.

11.22 The object may be directly used with input stream such as `cin` and output streams such as `cout`.

11.23 An `ostream` object should be returned by reference.

11.24 An `istream` object should be returned by reference.

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

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

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

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

11.29 The postfix version has a dummy parameter.

11.30
```
#ifndef INTARRAY_H
#define INTARRAY_H
#include <iostream>
using namespace std;
// Modified Intarry.h
```

```
class IntArray
{
private:
      int *aptr;
      int arraySize;
      void subError();   // Handles subscripts out of range
public:
      IntArray(int);                    // Constructor
      IntArray(const IntArray &); // Copy constructor
      ~IntArray();                          // Destructor
      int size(){ return arraySize; }
      int &operator[](int);       // Overloaded [] operator
      int operator()(int, int); // Overloaded () operator
};
#endif


// Overloaded operator () added to IntArray.cpp
int IntArray::operator()(int i, int j)
{
  int sum = 0;

  if (i < 0 || j >= arraySize)
      subError();
  for(int k = i; k <= j; k++)
    sum = sum + aptr[k];
  return sum;
}


#include <iostream>
#include "IntArray.h"

using namespace std;

int main()
{
   IntArray table(10);

   // Store values in the array.
   for (int x = 0; x < table.size(); x++)
      table[x] = x;

   // Print the sum of the values in the range 3..5
   cout << table(3, 5);

   return 0;
}
```

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

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

11.33 `BlackBox::operator int()`

11.34 `Big::Big (Small sm)`

11.35 The is-a relation

11.36 Because derived class objects can be considered as forming a subset of the set of base class objects. Hence we can think of the base class as a "uperset" or superclass of the derived class.

11.37 The base class access specification determines how members inherited from the base class will be accessed in the derived class.

11.38 A typist is a special case of an employee.

```
class Employee
{
   int yearsOfService;
};
class Typist : public Employee
{
   int wordsPerMinute;
};
```

11.39 Other than to friend functions, private members are only accessible to member functions of the same class. Protected members are accessible to member functions of the class as well as member functions of all derived classes.

11.40 Member access specification determines how a class member is accessible to code outside of the class. Base class access specification determines how members inherited from a base class will be accessed through the derived class.

11.41 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

11.42 Derived class constructors can assume members of the base class object have already been initialized.

11.43 Declarations are for typechecking, definitions are for code generation. The compiler needs the arguments to the base class constructor when it is generating code.

11.44 The same situation arises with composition when an outer class object needs to pass arguments to a constructor of an inner class object. The same syntax is used.

11.45
```
Entering the base.
Entering the camp.
Leaving the camp.
Leaving the base.
```

11.46
```
This base is secure.
The camp is secluded.
Leaving the camp.
Leaving the base.
```

Chapter 12

12.1

`strlen`	Accepts a C-string as an argument. Returns the length of the string (not including the null terminator).
`strcat`	Accepts two C-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 two C-strings as arguments. The function copies the second string to the first string. The second string is left unchanged.
`strncpy`	Accepts two C-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 two C-string arguments. If `string1` and `string2` are the same, this function returns `0`. If `string2` is alphabetically greater than `string1`, it returns a negative number. If `string2` is alphabetically less than `string1`, it returns a positive 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).

12.2
```
4
```

12.3
```
Have a nice day
nice day
```

12.4
```
strcpy(composer, "Beethoven");
```

12.5
```
#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;
}
```

12.6
A) negative
B) negative
C) negative
D) positive

12.7
```
if (strcmp(iceCream, "Chocolate") == 0)
       cout << "Chocolate: 9 fat grams.\n";
else if (strcmp(iceCream, "Vanilla") == 0)
       cout << "Vanilla: 10 fat grams.\n";
else if (strcmp(iceCream, "Pralines and Pecan") == 0)
       cout << "Pralines and Pecan: 14 fat grams.\n";
else
       cout << "That's not one of our flavors!\n";
```

12.8

`atoi`	Accepts a C-string as an argument. The function converts the string to an integer and returns that value.
`atol`	Accepts a C-string as an argument. The function converts the string to a long integer and returns that value.
`atof`	Accepts a C-string as an argument. The function converts the string to a double and returns that value.
`itoa`	Converts an integer to a C-string. The first argument is the integer. The result will be stored at the location pointed to by the second argument. 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.)

12.9
```
num = atoi("10");
```

12.10
```
num = atol("10000");
```

12.11
```
num = atof("7.2389");
```

12.12
```
itoa(127, strValue, 10);
```

12.13

`isalpha`	Returns `true` (a nonzero number) if the argument is a letter of the alphabet. Returns `false` 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 `false`.
`isdigit`	Returns `true` (a nonzero number) if the argument is a digit 0–9. Otherwise it returns `false`.
`islower`	Returns `true` (a nonzero number) if the argument is a lowercase letter. Otherwise, it returns `false`.
`isprint`	Returns `true` (a nonzero number) if the argument is a printable character (including a space). Returns `false` otherwise.
`ispunct`	Returns `true` (a nonzero number) if the argument is a printable character other than a digit, letter, or space. Returns `false` otherwise.
`isupper`	Returns `true` (a nonzero number) if the argument is an uppercase letter. Otherwise, it returns `false`.

`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 `false`.
`toupper`	Returns the uppercase equivalent of its argument.
`tolower`	Returns the lowercase equivalent of its argument.

12.14
```
little = tolower(big);
```

12.15
```
if (isdigit(ch))
   cout << "digit";
else
   cout << "Not a digit.";
```

12.16
```
A
```

12.17
```
char choice;
do
   {
       cout << "Do you want to repeat the program or quit? (R/Q) ";
       cin >> choice;
   } while (toupper(choice) != 'Q');
```

12.18
```
Tom Talbert Tried Trains
     Dom Dalbert Dried Drains
```

Chapter 13

13.1 By filenames

13.2 Open the file, read or write the file, close the file.

13.3 `fstream`

13.4 `ofstream` Output file stream. This data type can be used to create files and write information to them. With the `ofstream` data type, information may only be copied from variables to the file, but not vice versa.

`ifstream` Input file stream. This data type can be used to create files and read information from them into memory. With the `ifstream` data type, information may only be copied from the file into variables, but not vice versa.

`fstream` File stream. This data type can be used to create files, write information to them, and read information from them. With the `fstream` data type, information may be copied from variables into a file or from a file into variables.

13.5 The file must be opened.

13.6
```
diskInfo.open("names.dat", ios::out);
```

13.7
```
diskInfo.open("customers.dat", ios::out | ios::app);
```

13.8
```
diskInfo.open("payable.dat", ios::in | ios::out | ios::app);
```

13.9
```
if (dataBase)
   cout << "The file was successfully opened.\n";
else
   cout << "The file was not opened.\n";
```

13.10
```
outFile.close();
```

13.11
```
A) cout.width(6);
B) cout.precision(4);
C) cout.setf(ios::fixed);
D) cout.setf(ios::left | ios::scientific);
E) cout.unsetf(ios::scientific);
```

13.12
```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main()
{
   string person = "Wolfgang Smith";
   cout.setf(ios::right);
   cout.width(20);
   cout << person << endl;
   cout.setf(ios::left);
   cout << person << endl;
   return 0;
}
```

13.13
```
#include<iostream>
#include <fstream>
using namespace std;

int main()
{
   fstream outFile;
   outFile.open("output.txt", ios::out);
   outFile << "Today is the first day\n";
   outFile << "of the rest of your life.\n";
   return 0;
}
```

13.14 It reports when the end of a file has been encountered.

13.15
```
Run
Spot
run
See
Spot
run
```

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

13.17 The `getline` function reads a line of text; the `get` function reads a single character.

13.18 Writes a single character to a file.

13.19
```
1e+002     1.7     8.6     7.8     5.1
```

13.20
```
#include <iostream>
#include <fstream>
#include <cctype>    // Needed for toupper
using namespace std;

int main()
{
   const int NAME_LENGTH = 81;
             PHONE_LENGTH = 26;
   fstream namesFile;
   namesFile.open("phones.dat", ios::app);
   char name[NAME_LENGTH], phone[PHONE_LENGTH];

   cout << "This program allows you to add names and phone\n";
   cout << "numbers to phones.dat.\n";
   do
   {
      char add;
      cout << "Do you wish to add an entry? ";
      cin >> add;
      if (toupper(add) == 'Y')
      {
         cout << "name: ";
         cin.getline(name, NAME_LENGTH);
         cout << "phone number: ";
         cin.getline(phone, PHONE_LENGTH);
         namesFile << name << endl;
         namesFile << phone << endl;
      }
   } while (toupper(add) == 'Y');
   namesFile.close();
   return 0;
}
```

13.21 You define multiple file stream objects, one for each file you wish to work with.

13.22 Character representation: “479”

ASCII codes: *52 55 57*

13.23 The << operator writes text to a file. The write member function writes binary data to a file.

13.24 The first argument is the starting address of the section of memory, which is to be written to the file. The second argument is the size, in bytes, of the item being written.

13.25 The first argument is the starting address of the section of memory where information read from the file is to be stored. The second argument is the size, in bytes, of the item being read.

13.26 A filed is an individual piece of information pertaining to a single item. A record is made up of fields and is a complete set of information about a single item.

13.27
```
file.write(reinterpret_cast<char> (&cust), sizeof(cust));
```

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

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

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

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

`ios::curr` The offset is calculated from the current position

13.31 0

13.32 `file.seekp(100L, ios::beg);`

Moves the write position to the one hundred first byte (byte 100) from the beginning of the file.

```
file.seekp(-10L, ios::end);
```

Moves the write position to 10 bytes before the end of the file.

```
file.seekp(-25L, ios::cur);
```

Moves the write position 25 bytes backward from the current position.

```
file.seekp(30L, ios::cur);
```

Moves the write position 30 bytes forward from the current position.

13.33 `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 | ios::binary);
```

Input and output, binary mode

Chapter 14

14.1 A simple case of the problem that can be solved without recursion.

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

14.3 `10`

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

Chapter 15

15.1 Let `p` be a pointer pointing to an object `ob` of a class that is part of an inheritance hierarchy. In general, `p` will be a pointer to some base class `B`, and the object `ob` will be an instance of a class `D` derived from `B`. Let f be a member function of `B` that is overridden in `D`. If the call `p->f()` is being made, static binding will call the version of `f` that is in the class `B`. Static binding will select the function to call based on the type of the pointer, and will do so at compile time. Dynamic binding will wait until runtime, and will select the version of `f` that is in `D`, the class of the object.

15.2 Dynamically

15.3
```
1
5
```

15.4
```
2
2
```

15.5
```
2
1
```

15.6
```
2
```

15.7 The body of the function is replaced with `= 0;`

15.8 It cannot be used to instantiate objects.

15.9 A) Inaccessible
B) Protected
C) Protected
D) Inaccessible
E) Protected
F) Public
G) Private
H) Protected
I) Public

15.10
```
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>
int minPosition(T arr[], int size)
{
   int minPos = 0;
   for (int k = 1; k < size; k++)
   {
      if (arr[k] < arr[minPos])
         minPos = k;
   }
   return minPos;
}
```

16.8 That the operator has been overloaded by the class object

16.9 First write a regular, nontemplated 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 successor pointer points to the next node in the list.

17.2 A pointer to the first node in the tree

17.3 The successor pointer in the last node will have a value of NULL.

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 We need a pointer to the previous node so we can set its successor pointer to 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, linked list, or deque

Chapter 19

19.1 A standard linked list is a linear data structure in which each node has at most one successor. A binary tree is nonlinear, because each node can have up to two successors.

19.2 The first node in the tree

19.3 A node pointed to by another node in the tree

19.4 A node that points to no other nodes

19.5 A collection of nodes of the binary tree that consists of some node X, together with all the descendants of X. An empty collection of nodes is also a subtree.

19.6 Information can be stored in a binary tree in a way that makes a form of binary search possible.

19.7
1. The node's left subtree is traversed.
2. The node's data is processed.
3. The node's right subtree is traversed.

19.8
1. The node's data is processed.
2. The node's left subtree is traversed.
3. The node's right subtree is traversed.

19.9
1. The node's left subtree is traversed.
2. The node's right subtree is traversed.
3. The node's data is processed.

19.10 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.

19.11 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.

19.12
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.