Chapter 2
Flow of Control

Key Terms
Boolean expression
&& means "and"
|| means "or"
truth tables
parentheses
precedence rules
higher precedence
short-circuit
complete evaluation
integers convert to `bool`
`if-else` statement
parentheses
`if-else` with multiple statements
compound statement
`if` statement
indenting
multiway `if-else`
`switch` statement
controlling expression
`break` statement
`default`
enumeration type
conditional operator
conditional operator expression
loop body iteration
`while` and `do-while` compared
executing the body zero times
comma operator
comma expression
`for` statement
empty statement
infinite loop
`continue` statement
ifstream
stream
file input
text file

Brief Outline

1. Introduction and Teaching Suggestions

Without flow of control, a language is not able to make decisions. The ability to choose a path through code based on data is what gives a computer its power. Two flow of control issues not covered in this chapter are function calls and exception handling. Function calls are covered in Chapter 3 and exceptions in Chapter 18.

This chapter discusses flow of control using both selection and iteration. The if-statement and loops are introduced both in this chapter. With both topics in one chapter, it is conceivable that this chapter will take longer for students to work through than many of the others in the text.

Branching, or selection, is introduced using the if-else statement, the if statement, and the switch statement. Multiple paths and therefore, nested if-else statements are introduced. As the last form of selection available, the conditional operator is introduced at the end of the chapter. Although many students choose not to use this operator, it is useful to cover so students can read code that does use it.

With the introduction of looping and selection, students can write fairly complex programs at the end of this chapter. However, what usually begins to happen is that students will write code that has errors. Finding these errors can be tedious and time-consuming. The sections on common pitfalls (infinite loops, semicolon at the end of for loops, etc.) should be covered as an

introduction to debugging. An option that would be useful to them if it is available would be to introduce and discuss the use of your computing environment's debugger at this time.

The section on file input allows students to write programs that use real-world data. This is significant because it allows students to move beyond toy problems. The book has several problems based on a list of English words. Although this section requires some "magic" code that is not fully explained until later, students should be able to grasp the basic concepts of reading from a text file and can begin writing program that operate on large amounts of data.

2. Key Points

Boolean Expressions and Relational Operators. The Boolean operators && and || are introduced along with ! and their use with ==, !=, <, >, <=, and >=. Truth tables and building complex Boolean expressions are important for the students to learn how to construct. In addition, students must also learn the precedence rules associated with these operators.

If-else statement & Multiway if-else Statement. There are many ways to give the program a sense of choice or branching. First, we can use an if-statement by itself without an else. This allows us the option to do something or skip over it. We can also use an if-else statement, which allows us to take one path or another. Lastly, we can use combinations of these to have more than two choices for execution. As the number of paths increase, so does the complexity of code for the students. Students should be able to follow as well as write these more complicated branching code segments.

The switch Statement. The switch also allows for branching, but it has limitations as to what the condition for branching can be. Also, the syntax contains more keywords and is more structured than the if-else. Discussion of the break statement is needed here as the switch will not function properly without the correct use of break between the cases.

true and false are numbers. True and false can be represented as the numbers 1 and 0. This is sometimes used by setting a variable equal to the result of an Boolean expression, or by testing a variable inside an if-statement.

Syntax for while and do-while Statements. The while and do-while loops are the indefinite loops supported by C++. They also illustrate the differences between an entry-test and an exit-test loop.

The for Statement. The for loop is a definite loop or counting loop that is also an entry-test loop. The syntax for the for loop is different from the other two loops and has a loop counter built right into the construct. However, in C++, we can have more than one statement inside the parts of the for-loop separated by commas and we can also leave parts empty, which can create many different results when using a for-loop.

break and continue. The difference between break and continue are sometimes confused by students, particularly scenarios where continue is useful. It can be instructive to show how the same effect can be achieved by code using if-statements instead of break and continue.

Enumeration Types. These types allow us to define a sequence of constant values that are often used in loops or switch statements. They map to integers.

3. Tips

Use switch statements for menus. The switch statement's advantage over the if-statement is increased clarity, especially for large numbers of items that must be compared. This makes it a good choice for implementing menus, in particular when used with functions introduced in Chapter 3.

Repeat-N Times Loops. A for loop is the best way to implement a count-based loop that repeats N times.

4. Pitfalls

Using = in Place of ==. The instructor should note that the very best of programmers fall into this error. The assignment

```
x = 1;
```

is an expression that has a value. The value of this expression is the value transferred to the variable on the left-hand side of the assignment, in this case, 1.. This value could be used by an `if` or `while` as a condition, just as any other expression can, for example:

```
z = (x = 1);
```

This, in fact, is a typical C/C++ idiom. While this is used, it can be confusing. Many instructors discourage its use, although some programmers do use it. Some compilers warn if an assignment expression is used as the Boolean expression in an `if` statement or within a loop. There is a way to get some further help at the cost of a little discipline. If the programmer will *consistently* write any constant that may appear in a comparison on the left of the `==`, as in

```
if(12 == x)
...;
```

instead of

```
if(x == 12)
...;
```

then the compiler will catch this pitfall of using `=` *instead of* `==.`

```
if(12 = x) // error: invalid l-value in assignment
...;
```

Otherwise this is very hard to see, since this `looks` right. There are also difficult-to-see pitfalls in using only one `&` instead of `&&`, or one `|`, instead of `||`. Each of these is legal C++, so the student won't be warned, and the program will run, and produce incorrect results. Warn them.

Extra Semicolon. If we define an `enum` type, and omit the required semicolon after the ending curly brace, the error message on some compilers is incomprehensible. On other compilers, it is as simple as "missing semicolon in `enum` definition." The reason the `enum` definition requires a semicolon is that one can define a variable of `enum` type between the close curly and the semicolon.

```
enum CompassDir {NORTH, SOUTH, EAST, WEST} direction;
```

Then `direction` has `CompassDir` type. If the semicolon is omitted, the compiler thinks that anything following the semicolon wants to be an identifier of `enum` type, and identifies this as an error, and issues an error message.

Another Semicolon Pitfall. This error occurs when a student adds a semicolon to the end of an if statement or for loop, e.g.:

```
if(x == 12);
  x = 0;
```

After this code segment, the variable `x` has the value 0. The hard-to-see error is the semicolon at the end of the line with the `if`. The `if` statement expects to see a single statement after the closing parenthesis. The semicolon produces a null statement between the close parenthesis and the semicolon. This makes the next statement (`x = 0;`) out of the scope of control of the `if`. If the `if` has an `else`, as in

```
if(x == 12);
  x = 0;
else
  x = 1;
```

Then the error message appears at the `else`. The syntax error really is the presence of the `else`, rather than at the extraneous semicolon, the real perpetrator. As above, the semicolon produces a null statement between the close parenthesis and the semicolon. This makes the next statement (`x = 0;`) out of the scope of control of the `if`, and makes the `else` into an "`else` without an if" error.

The text has a Pitfall section on the extra semicolon in a `for` loop and another that notes that the extra semicolon problem can cause an infinite loop, for example,

```
x = 10;
while(x > 0);
{
  cout << x << endl;
  x--;
}
```

A program that has this loop in it will hang. As before, the extraneous semicolon after the `while` causes the problem. Note that placing assignments in the control of an `if` or a `while` is C (and C++) idiom. Even so, some compilers warn about this. To stop such runaway programs may require killing the process or hitting control-C.

5. Programming Projects Answers

1. Inflation

Write a program to estimate the future cost of an item based on the current inflation rate and the current cost of the item. The inflation rate is to be entered as a percentage, e.g., 5.6 for 5.6 percent. Use a loop to compute the estimated cost based on the (compounded) inflation rate.

```
//inflation estimator
//input: item cost in dollars, inflation as a percentage:
//  for example, 5.6 for 5.6 percent, and an integral number of //  years.
//output: Estimate in dollars of the cost
#include <iostream>
using namespace std;

int main()
{
  double cost;
  double rate; // inflation rate
  int years;   //years to time of requested estimate
  cout << "This is the Future Cost Estimator. \n"
       << "Each entry is to be followed by a <return>\n\n";
  cout << "Enter the cost of the item in dollars: \n";
  cin >> cost;
  cout << "Enter the inflation rate as a percentage: \n"
       << "such as 5.6 for 5.6 percent. ";
  cin >> rate;
  rate = rate / 100.0;
  cout << "Enter an integral number of years to the time \n"
       << "when you want the estimate.";
  cin >> years;

  for( int i = 0; i < years; i++)
    cost = (1 + rate) * cost;

  cout << "The inflated cost is $" << cost << endl;

  return 0;
}
```

2. Installment Loan Time

No down payment, 18 percent / year, payment of $50/month, payment goes first to interest, balance to principal. Write a program that determines the number of months it will take to pay off a $1000 stereo. The following code also outputs the monthly status of the loan.

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

const double PAYMENT=50.00;

int main()
{
  double principal = 1000.;
  double interest, rate = 0.015;

  int months = 0;
```

```
  cout << "months\tinterest\tprincipal" << endl;

  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);
  cout.precision(2);

  while(principal > 0)
  {
    months++;
    interest = principal * rate;
    principal = principal - (PAYMENT - interest);

    if ( principal > 0 )
      cout << months << "\t\t" << interest << "\t\t"
           << principal << endl;
  }
  cout << "number of payments = " << months;
  //undo the action that drove principal negative:
  principal = principal + (PAYMENT - interest);

  //include interest for last month:
  interest = principal * 0.015;
  principal = principal + interest;
  cout << " last months interest = " << interest;
  cout << " last payment = " << principal << endl;

  return 0;
}

A typical run is:

months      interest    principal
1           15.00       965.00
2           14.47       929.48
3           13.94       893.42
4           13.40       856.82
. . .
21          2.86        143.53
22          2.15        95.68
23          1.44        47.12
number of payments = 24 last months interest = 0.71 last payment = 47.83
```

3. Chocolate

Suppose we can buy a chocolate bar from the vending machine for $1 each. Inside every chocolate bar is a coupon. We can redeem 7 coupons for one chocolate bar from the machine. We would like to know how many chocolate bars can be eaten, including those redeemed via coupon, if we have ***n*** dollars.

For example, if we have 20 dollars then we can initially buy 20 chocolate bars. This gives us 20 coupons. We can redeem 14 coupons for 2 additional chocolate bars. These two additional chocolate bars have 2 more coupons, so we now have a total of 8 coupons when added to the six leftover from the original purchase. This gives us enough to redeem for one final chocolate bar. As a result we now have 23 chocolate bars and 2 leftover coupons.

Write a program that inputs the number of dollars and outputs how many chocolate bars you can collect after spending all your money and redeeming as many coupons as possible. Also output the number of leftover coupons. The easiest way to solve this problem is to use a loop.

CodeMate Hint: Use a while loop that redeems coupons for bars and calculates the new number of coupons. Continue the loop as long as you have enough coupons to redeem for a chocolate bar.

```
//chocolate.cpp
//
//This program computes the number of candy bars we can get.
//  Suppose we can buy a chocolate bar from the vending machine for $1
//  each.  Inside every chocolate bar is a coupon.  We can redeem 7
//  coupons for one chocolate bar from the machine.  We would like to
//  know how many chocolate bars can be eaten, including those redeemed
//  via coupon, if we have n dollars.
//
//This program creates a loop that simulates the process by
//  calculating the number of coupons we have leftover after redeeming
//  them for candy bars.  When we don't have enough coupons to redeem,
//  the loop stops.

#include <iostream>
#include <cstdlib>

using namespace std;

// ======================
//     main function
// ======================
int main()
{

 // Variable declarations
 int dollars = 0;
 int bars = 0;
 int newbars = 0;
 int coupons = 0;

 cout << endl << "Enter the number of dollars you have to spend:"
       << endl;
 cin >> dollars;

 //
 // --------------------------------
 // ----- ENTER YOUR CODE HERE -----
 // --------------------------------

 // Buy initial candy bars and number of coupons
 bars = dollars;
 coupons = bars;

 // Continually redeem if we have enough coupons
 while (coupons >= 7)
```

```
 {
      // Redeem 7 coupons for one bar
      newbars = coupons / 7;
      bars = bars + newbars;
      // New number of coupons is the amount leftover +
      // number of coupons we get from the bars we just bought
      coupons = (coupons % 7) + newbars;
 }


 // Output the number of candy bars you can get
 cout << "You can get " << bars << " candy bars " <<
       " with " << coupons << " coupons leftover."
       << endl << endl;

 // --------------------------------
 // --------- END USER CODE --------
 // --------------------------------
```

4. Primes

Write a program that finds and prints all of the prime numbers between 3 and 100. A prime number is a number such that one and itself are the only numbers that evenly divide it (e.g., 3,5,7,11,13,17, …).

One way to solve this problem is to use a doubly-nested loop. The outer loop can iterate from 3 to 100 while the inner loop checks to see if the counter value for the outer loop is prime. One way to see if number *n* is prime is to loop from 2 to *n-1* and if any of these numbers evenly divides *n* then *n* cannot be prime. If none of the values from 2 to *n-1* evenly divide *n*, then *n* must be prime. (Note that there are several easy ways to make this algorithm more efficient).

```
//primes.cpp
//
//This program calculates the prime numbers between 3 and 100.
//  It uses a doubly-nested loop.  The outer loop iterates from 3 to 100
//  while the inner loop checks to see if the counter value for the
//  outer loop is prime.  To check for n to be prime, we loop from 2 to
//  n-1 and if any of these numbers evenly divides n then n cannot be
//  prime.  If none of the values from 2 to n-1 evenly divide n, then n
//  must be prime.  Note that there are several ways to make this
//  algorithm more efficient.

#include <iostream>
#include <cstdlib>

using namespace std;

// ======================
//     main function
// ======================
int main()
{

 // Variable declarations
```

```
int n, i;
bool isprime;

//
// --------------------------------
// ----- ENTER YOUR CODE HERE -----
// --------------------------------

// Outer loop, tests to see if n is prime for n=3 to 100
for (n=3; n<=100; n++)
{
     // Assume n is prime
     isprime = true;
     // Inner loop tests whether or not n is prime
     for (i=2; i<= n-1; i++)
     {
           // n is not prime if n/i has no remainder
           if ((n % i)==0)
           {
                 isprime = false;
           }
     }
     // Check if flag isprime is still true
     if (isprime)
     {
           // Output that n is prime
           cout << n << " is a prime number." << endl;
     }
}

// --------------------------------
// --------- END USER CODE --------
// --------------------------------
cout << endl;
```

5. Cryptarithmetic

In cryptarithmetic puzzles, mathematical equations are written using letters. Each letter can be a digit from 0 to 9, but no two letters can be the same. Here is a sample problem:

SEND + MORE = MONEY

A solution to the puzzle is S = 9, R = 8, O = 0, M = 1, Y = 2, E = 5, N = 6, D = 7
Write a program that finds solutions to the cryptarithmetic puzzle of:

TOO + TOO + TOO + TOO = GOOD

The simplest technique is to use a nested loop for each unique letter (in this case T, O, G, D). The loops would systematically assign the digits from 0-9 to each letter. For example, it might first try T = 0, O = 0, G = 0, D = 0, then T = 0, O = 0, G =0, D = 1, then T = 0, O = 0, G = 0, D = 2, etc. up to T = 9, O = 9, G = 9, D = 9. In the loop body test that each variable is unique and that the equation is satisfied. Output the values for the letters that satisfy the equation.

```
// crypt.cpp
//
// This program calculates the solution to the cryptarithmetic
// puzzle TOO + TOO + TOO + TOO = GOOD where each letter represents
// a single digit with no duplication.  It loops over all possible
// values for each digit, ensures that the digits are unique, computes
// the sum, and if the equation is satisfied outputs the values for
// each digit.
// We must make sure to account for the possibility of carries when
// adding digits.

#include <iostream>
#include <cstdlib>

using namespace std;

// ======================
//     main function
// ======================
int main()
{

 // Variable declarations
 int t, o, g, d;

 //
 // --------------------------------
 // ----- ENTER YOUR CODE HERE -----
 // --------------------------------

 // Loop over all values for "T", "O", "G", and "D"
 for (t = 0; t <= 9; t++)
   for (o = 0; o <=9; o++)
     for (g = 0; g <= 9; g++)
       for (d = 0; d <= 9; d++)
            {
                  // Ensure uniqueness for each digit
                  if ((t != o) && (t != g) && (t != d) &&
                     (o != g) && (o != d) &&
                     (g != d))
                     {
                           // Compute rightmost carry and digit
                           int carry0 = (o + o + o + o) / 10;
                           int digit0 = (o + o + o + o) % 10;
                           // Compute second digit from right
                           int carry1 = (carry0 + o + o + o + o) / 10;
                           int digit1 = (carry0 + o + o + o + o) % 10;
                           // Compute third digit from right
                           int carry2 = (carry1 + t + t + t + t) / 10;
                           int digit2 = (carry1 + t + t + t + t) % 10;
                           // Check if equation matches7
                           if ((carry2 == g) && (digit2 == o) &&
                               (digit1 == o) && (digit0 == d))
                             {
                               cout << "The values are: T = " << t <<
                                  " O = " << o << " G = " << g <<
```

```
                                    " D = " << d << endl;
                                }
                        }
                }

 // --------------------------------
 // --------- END USER CODE --------
 // --------------------------------
 cout << endl;
}
```

6. Buoyancy

Buoyancy is the ability of an object to float. Archimedes' Principle states that the buoyant force is equal to the weight of the fluid that is displaced by the submerged object. The buoyant force can be computed by:

$$F_b = V \times \gamma$$

Where F_b is the buoyant force, V is the volume of the submerged object, and γ is the specific weight of the fluid. If F_b is greater than or equal to the weight of the object then it will float, otherwise it will sink.

Write a program that inputs the weight (in pounds) and radius (in feet) of a sphere and outputs whether the sphere will sink or float in water. Use $\gamma = 62.4$ lb/ft^3 as the specific weight of water. The volume of a sphere is computed by $(4/3)\pi r^3$.

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

int main()
{
      const double SPECIFIC_WEIGHT = 62.4;

      cout << "Input weight of the sphere in pounds." << endl;
      double w;
      cin >> w;
      cout << "Input radius of the sphere in feet." << endl;
      double r;
      cin >> r;

      // Compute volume
      double volume = 4 * 3.14159 * (r*r*r) / 3;
      // Compute buoyant force
      double force = volume * SPECIFIC_WEIGHT;

      // Output if it will sink or float
      if (force >= w)
      {
            cout << "The sphere will float in water." << endl;
      }
      else
      {
```

```
            cout << "The sphere will sink in water." << endl;
      }

      // Type a key and enter to close the program
      char c;
      cin >> c;
}
```

7. Grade Calculator

Write a program that calculates the total grade for *N* classroom exercises as a percentage. The user should input the value for *N* followed by each of the *N* scores and totals. Calculate the overall percentage (sum of the total points earned divided by the total points possible) and output it as a percentage. Sample input and output is shown below.

```
      How many exercises to input? 3

      Score received for exercise 1: 10
      Total points possible for exercise 1: 10

      Score received for exercise 2: 7
      Total points possible for exercise 2: 12

      Score received for exercise 3: 5
      Total points possible for exercise 3: 8

      Your total is 22 out of 30, or 73.33%.
```

```
// ======================
//     Grade Calculator
// ======================
#include <iostream>
using namespace std;

int main( )
{
      int numExercises;
      cout << "How many exercises to input?" << endl;
      cin >> numExercises;

      int totalScore = 0;
      int maxScore = 0;
      for (int i = 0; i < numExercises; i++)
      {
            int score;
            int max;
            cout << "Score received for exercise " << (i+1)
                  << " ";
            cin >> score;
            totalScore += score;
            cout << "Total points possible for exercise " << (i+1)
                  << " ";
            cin >> max;
            maxScore += max;
      }
      cout << "Your total is " << totalScore <<
```

```
            " out of " << maxScore << ", or " <<
            (double) totalScore * 100 / maxScore <<
            " percent." << endl;

      cout << "Enter a character to exit." << endl;
      char wait;
      cin >> wait;
      return 0;
}
```

8. Same Temperature

Write a program that finds the temperature, as an integer, that is the same in both Celsius and Fahrenheit. The formula to convert from Celsius to Fahrenheit is:

$$Fahrenheit = \frac{9}{5}Celsius + 32$$

Your program should create two integer variables for the temperature in Celsius and Fahrenheit. Initialize the temperature to 100 degrees Celsius. In a loop, decrement the Celsius value and compute the corresponding temperature in Fahrenheit until the two values are the same.

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

int main( )
{
      int fahrenheit, celsius;
      celsius = 100;

      fahrenheit = (9 * celsius / 5) + 32;
      while (celsius != fahrenheit)
      {
            celsius--;
            fahrenheit = (9 * celsius / 5) + 32;
      }
      cout << "The temperature is the same at " << fahrenheit << endl;

      cout << "Enter a character to exit." << endl;
      char wait;
      cin >> wait;
      return 0;
}
```

9. Babylonian Algorithm

(This is an extension of an exercise from Chapter 1.) The Babylonian algorithm to compute the square root of a positive number **n** is as follows:

1. Make a **guess** at the answer (you can pick n/2 as your initial guess).
2. Compute ***r* = n / guess.**
3. Set **guess = (guess + r) / 2.**
4. Go back to step 2 for as many iterations as necessary. The more steps 2 and 3 are repeated, the closer **guess** will become to the square root of **n.**

Write a program that inputs a double for **n**, iterates through the Babylonian algorithm until the guess is within 1% of the previous guess, and outputs the answer as a double to two decimal places. Your answer should be accurate even for large values of n.

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

int main()
{
	double n;
	double lastGuess,currentGuess;
	cout << "Enter a number: " << endl;
	cin >> n;

	lastGuess = n / 2;
	bool done = false;

	do
	{
		double r = n / lastGuess;
		currentGuess = (lastGuess + r) / 2;

		// Calculate percent difference
		double diff = lastGuess - currentGuess;
		if (diff < 0)
			diff = diff * -1;
		double percentDiff = diff / lastGuess;
		lastGuess = currentGuess;
		cout << " loop " << endl;
		if (percentDiff < 0.01)
			done = true;
	} while (!done);

	cout.setf(ios::fixed, ios::floatfield);
	cout.precision(2);
	cout << "The square root of " << n << " is approximately " <<
		currentGuess << endl;

}
```

10. Text Replacement

. Create a text file that contains the text "I hate C++ and hate programming!" Write a program that reads in the text from the file and outputs each word to the console but replaces any occurrence of "hate" with "love". Your program should work with any line of text that contains the word "hate", not just the example given in this problem.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main( )
{
    string text;
    fstream inputStream;

    inputStream.open("hate.txt");

    while (inputStream >> text)
    {
          if (text == "hate")
            cout << "love ";
          else
            cout << text << " ";
    }
    inputStream.close();
    cout << endl;
}
```

11. Ideal Body Weight Estimation – File Version

(This is an extension of an exercise from Chapter 1.) A simple rule to estimate your ideal body weight is to allow 110 pounds for the first 5 feet of height and 5 pounds for each additional inch. Create the following text in a text file. It contains the names and heights in feet and inches of Tom Atto (6'3"), Eaton Wright (5'5"), and Cary Oki (5'11"):

```
Tom Atto
6
3
Eaton Wright
5
5
Cary Oki
5
11
```

Write a program that reads the data in the file and outputs the full name and ideal body weight for each person. Use a loop to read the names from the file. Your program should also handle an arbitrary number of entries in the file instead of handling only three entries.

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main( )
{
        string first, last;
        int heightFeet, heightInches;
        fstream inputStream;

        inputStream.open("names.txt");

        while (inputStream >> first)
        {
                inputStream >> last;
                inputStream >> heightFeet >> heightInches;
                int weight = 110 + ((heightFeet - 5)*12 + heightInches) * 5;
                cout << "The ideal body weight for " << first << " " << last
                     << " is " << weight << " pounds." << endl;
        }
        inputStream.close();
    return 0;
}
```

12. Benford's Law

This problem is based on a "Nifty Assignment" by Steve Wolfman (http://nifty.stanford.edu/2006/wolfman-pretid). Consider lists of numbers from real-life data sources, for example, a list containing the number of students enrolled in different course sections, the number of comments posted for different Facebook status updates, the number of books in different library holdings, the number of votes per precinct, etc. It might seem like the leading digit of each number in the list should be 1-9 with an equally likely probability. However, Benford's Law states that the leading digit is 1 about 30% of the time and drops with larger digits. The leading digit is 9 only about 5% of the time.

Write a program that tests Benford's Law. Collect a list of at least one hundred numbers from some real-life data source and enter them into a text file. Your program should loop through the list of numbers and count how many times 1 is the first digit, 2 is the first digit, etc. For each digit output the percentage it appears as the first digit.

No solution provided.

13. File Reading

Create a text file that contains 10 integers with one integer per line. You can enter any 10 integers that you like in the file. Then write a program that inputs a number from the keyboard and determines if any pair of the 10 integers in the text file adds up exactly to the number typed in from the keyboard. If so, the program should output the pair of integers. If no pair of integers adds up to the number then the program should output "No pair found."

```
// There is a much more efficient way to solve this problem, using arrays.
// They are covered later in the book.
//
// For now, the approach is fairly brute force.  We read each number
// into a variable.  Then for each variable, loop over the file again
// comparing the sums of the variable and the number read from the file.
// This version does end up comparing two numbers twice, i.e.
// (n1+n2) and (n2+n1).
//
// There are even more brute force solutions; this is a good problem to
// make the student think about code complexity and to also think
// of how there should be a better way to code the solution more
// efficiently.

#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  int desiredNum;
  fstream inputFile;
  int n1,n2,n3,n4,n5,n6,n7,n8,n9,n10;

  cout << "Enter a number." << endl;
  cin >> desiredNum;

  inputFile.open("file.txt");
  inputFile >> n1;
  inputFile >> n2;
  inputFile >> n3;
  inputFile >> n4;
  inputFile >> n5;
  inputFile >> n6;
  inputFile >> n7;
  inputFile >> n8;
  inputFile >> n9;
  inputFile >> n10;
  inputFile.close();

  // Compare n1 to all other numbers
  inputFile.open("file.txt");
  int temp;
  while (inputFile >> temp)
  {
```

```
  if ((temp + n1)==desiredNum)
      cout << n1 << " " << temp << endl;
}
inputFile.close();

// Compare n2 to all other numbers
inputFile.open("file.txt");
while (inputFile >> temp)
{
  if ((temp + n2)==desiredNum)
      cout << n2 << " " << temp << endl;
}
inputFile.close();

// Compare n3 to all other numbers
inputFile.open("file.txt");
while (inputFile >> temp)
{
  if ((temp + n3)==desiredNum)
      cout << n3 << " " << temp << endl;
}
inputFile.close();
// Compare n4 to all other numbers
inputFile.open("file.txt");
while (inputFile >> temp)
{
  if ((temp + n4)==desiredNum)
      cout << n4 << " " << temp << endl;
}
inputFile.close();

// Compare n5 to all other numbers
inputFile.open("file.txt");
while (inputFile >> temp)
{
  if ((temp + n5)==desiredNum)
      cout << n5 << " " << temp << endl;
}
inputFile.close();

// Compare n6 to all other numbers
inputFile.open("file.txt");
while (inputFile >> temp)
{
  if ((temp + n6)==desiredNum)
      cout << n6 << " " << temp << endl;
}
inputFile.close();

// Compare n7 to all other numbers
inputFile.open("file.txt");
while (inputFile >> temp)
{
  if ((temp + n7)==desiredNum)
      cout << n7 << " " << temp << endl;
}
inputFile.close();
```

```
  // Compare n8 to all other numbers
  inputFile.open("file.txt");
  while (inputFile >> temp)
  {
    if ((temp + n8)==desiredNum)
        cout << n8 << " " << temp << endl;
  }
  inputFile.close();

  // Compare n9 to all other numbers
  inputFile.open("file.txt");
  while (inputFile >> temp)
  {
    if ((temp + n9)==desiredNum)
        cout << n9 << " " << temp << endl;
  }
  inputFile.close();

  return 0;
}
```