Chapter 6 — Structures

Introduction

In your previous exposures to C++ programming, you probably have learned all about structures. If not, refer to Appendix A for the basics of structures. This chapter begins by providing a review of structures from the view point of how a structure handles the Record type of data structure by grouping a series of related fields together. Next, we will see how an array of structures can be designed in which only limit on the number of elements in the array is the amount available memory on the computer. Finally, we examine some of the advanced features of structures.

Structures as a Record of Data

A structure provides a means to group a series of related fields of information into one entity. The structure template defines which fields are in this entity. Suppose that the program is to process cost records. Each cost record includes the item number, quantity on hand, product description and its cost. Here is what the structure template looks like.

```
const int DescrLen = 21; // max length of description

struct COSTREC {
  long   itemNum;          // item number
  short  qty;              // quantity on hand
  char   descr[DescrLen]; // item description
  double cost;             // item cost
};
```

The structure tag, **COSTREC** in this case, is used to identify this particular structure. Remember that by convention, all structure tags either are wholly uppercase names (usually) or are capitalized. The tag specifies that, when instances of the structure are built, four data members of these types and names are to be created. The member fields are always created and stored in the order shown in the template.

The order of the structure members can sometimes be important. If this program is part of a collection of programs, all sharing the same files, such as a payroll system of programs, or if the data file to be input is in binary format, then the structure members must be in the same order that the data is in the binary file. A **binary** file is one in which all data is stored in internal format; binary files cannot be viewed with text editors such as Notepad. They are discussed in detail in chapter 13. For most problems, the fields can be in any order you choose.

The following creates a structure variable called **costRec**.

```
COSTREC costRec;
```

Figure 6.1 shows the memory layout of **costRec** and its member fields. Notice that the fields are in the same order as in the **COSTREC** template.

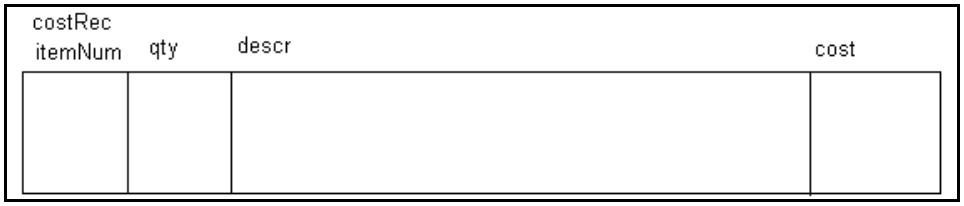

Figure 6.1 The **costRec** Memory Layout

One can have arrays of structures as well. Suppose that the program needed to store a maximum of 1000 cost records. The following defines the array and also shows the location of all the parts of the structure definitions.

File: CostRec.h

```
#ifndef COSTREC_H
#define COSTREC_H

#define MAXRECS 1000
const int DescrLen = 21;  // max length of description

struct COSTREC {
  long   itemNum;          // item number
  short  qty;              // quantity on hand
  char   descr[DescrLen]; // item description
  double cost;             // item cost
};
#endif
```

File: main.cpp

```
#include "CostRec.h"
int main () {
 COSTREC arec[MAXRECS];  // array of 1000 cost records
...
```

A structure can also contain instances of other structures and arrays of other structures. For example, consider a **DATE** structure which represents a calendar date. Using instances of a **DATE** structure would make passing dates very convenient. Further, consider an employee record that contained the employee's id number, his/her salary and the date that he/she was hired. The **EMPLOYEE** structure contains an instance of the **DATE** structure as shown below.

```
struct DATE {
  char month;
  char day;
  short year;
};
```

```
struct EMPLOYEE {
  long id;
  double salary;
  DATE hireDate;
};
```

Suppose that a **CARMAINT** structure must be defined to represent the periodic maintenance requirements for a new car. Here the **CARMAINT** structure contains an array of **DATE** structures.

```
const int numMaint = 10;
struct CARMAINT {
 bool maintenanceDone[numMaint];   // true if the work was done
 int  maintenanceCode[numMaint];   // manufacturer's maint. codes
 DATE maintenanceDueDate[numMaint];// date maintenance is due
};
```

Having defined the structure template and an created instance of it, the next action is to utilize the members within the structure. This is done by using the **dot (.)** operator. To the left of the **dot** operator must be a structure variable and to the right must be a member variable of that structure. We know that to access the **qty** member of the **costRec** instance, one codes

```
costRec.qty
```

To calculate the **totalCost** using the **cost** and **qty** members of the **costRec** instance, do the following.

```
double totalCost = costRec.qty * costRec.cost;
```

To display the description, use

```
cout << costRec.descr;
```

To increment the **costRec**'s quantity member or add another variable to it, one can code

```
costRec.qty++;
costRec.qty += orderedQty;
```

To input a set of data into the **costRec** variable, there are a number of ways. Here is one.

```
cin >> costRec.itemNum >> costRec.qty >> ws;
cin.get (costRec.descr, DescrLen);
cin >> costRec.cost;
```

The above assumes that no description field in the input data contains all blanks. It also assumes that all descriptions contain **DescrLen – 1** number of characters.

Structures can also be dynamically allocated. Here a new **CARMAINT** instance is allocated dynamically.

```
CARMAINT* ptrcar = new CARMAINT;
```

The individual members are accessed using the **pointer operator, ->,** as shown.

```
ptrcar->maintenanceDone[i]
ptrcar->maintenanceCode[i]
ptrcar->maintenanceDueDate[i].month
```

Notice the syntax of the last one above. We desire to get at the i^{th} due date's **month** field. The entire structure instance is pointed to by **prtcar**.

The **address** operator & returns the address of the structure variable. If one codes

```
&costRec
```

then the compiler provides the memory location where the instance begins. Normally, the compiler does this automatically for us when we use reference variables. Here `&costRec` would be a constant pointer to a **COSTREC** structure. Taking the address of a structure instance is sometimes needed when passing a pointer to the instance to a function.

A structure variable can be passed to a function or a reference or pointer to one can be passed. Passing by reference is the best approach to take. However, passing by use of a pointer is also fine. The main issue here is to realize that we are passing the address of the data structure entity, the structure instance.

Suppose that the **main()** program defined the cost record structure as we have been using it thus far. Suppose further that the **main()** function then wanted to call a **PrintRec()** function whose task is to print the data nicely formatted. Of course, we would pass the instance either by reference or by its address to avoid forcing the compiler to make a duplicate copy of this instance. The **main()** function can do one of the following.

```
int main () {
COSTREC crec;
…
PrintRec (outfile, crec);
```

where the **PrintRec()** function begins this way

```
void PrintRec (ostream& outfile, const COSTREC& crec) {
  outfile << crec.itemNum…
```

or

```
int main () {
COSTREC crec;
…
PrintRec (outfile, &crec);
```

and the **PrintRec()** function begins as follows

```
void PrintRec (ostream& outfile, const COSTREC* ptrcrec) {
  outfile << ptrcrec->itemNum…
```

In both cases, the memory location of **crec** is passed to the function. Since **PrintRec()** is not going to modify the data, it is further qualified as being a constant address.

This brings up the **ReadRec()** function whose job it is to input the data and somehow fill up the **main()**'s **costRec** with that data. One way that the **ReadRec()** function can be defined is to have it return a **COSTREC** structure. This is not a good way to do it, because execution time is required to make duplicate copies to return and to copy the returned data into the designated instance in **main()**. We know that instead we should pass a reference to the instance to be filled

or perhaps a pointer to that instance. The return value of such a **ReadRec()** function can then be used to return a reference to the input stream. We do this so that **main()** function has more ways that it can utilize the **ReadRec()** function.

```
istream& ReadRec (istream& infile, COSTREC& crec) {
 if (infile >> ws && !infile.good()) {
  return infile;
 }
 infile >> crec.itemNum >> and so on
 return infile;
}
```

Here the **main()** function can directly check on the input stream's status after an attempt has been made to input all of the fields.

```
int main () {
 COSTREC costRec;
 …
 while (ReadRec (infile, costRec)) {
```

This concept of encapsulating the action of inputting all of the data of a record within one function which then returns a reference to the input stream is a vital one. We will continue to make use of this concept when we write ATDs (classes).

The only obstacle we have faced thus far is with arrays of structure and the need for that constant fixed maximum number of elements in the array. So next, let's examine how an array of structures can be created such that it has no maximum upper bounds subject only to available memory.

A Growable Array of Structures

When writing applications, establishing the upper bounds for an array can often not be foretold with any degree of certainty. When you code

```
COSTREC arec[1000];
```

you are introducing an arbitrary, the maximum number of cost records that the program can handle. If the input contains even one additional record, the program has little choice but to terminate with an error stating the array bounds has been exceeded. With production programs, often a programmer can only make a wild guess at the maximum number that can be found in the input file(s) at run time. Indeed, as companies grow, their data bases likewise increase in volume. Sooner or later, a programmer can expect that he/she will be requested to make that arbitrary maximum number larger.

To counter this uncertainty, programmers sometimes make the array bounds way too large. That is, if the expected maximum is 100,000 elements, they make the array bounds 500,000 elements. Of course, this is self-defeating in that now the program itself always ties up vast amounts of memory that it never uses. However, if we design this array of records properly,

no such limitation occurs. The array just gets larger and larger. This is called a growable array. A growable array just keeps increasing the number of elements in it and is subject only to the total amount of memory available on the computer.

Suppose that we are dealing with customer daily order records. Each week our program inputs a series of daily sales files and calculates weekly sales statistics. Assuming that we have a properly defined **ORDER** structure, the fixed array limit definition would be as follows.

```
const int MAXORDERS = 100000;
ORDER orders[MAXORDERS];
```

How can this fixed approach be converted into an array whose bounds can easily be increased?

The key is dynamic memory allocation. Let's change the constant **MAXORDERS** into a variable, **maxOrders**. Assuming that **maxOrders** has the value of 100 in it at the moment, we could create an array of this size as shown.

```
long maxOrders; // currently contains 100
ORDER* orders;
orders = new ORDER [maxOrders];
```

Here orders is an array of 100 **ORDER** structures. Now, suppose that we input one additional customer order so that the **orders** array needs to become an array of 101 elements?

One way to accomplish this would be to allocate a new array of **maxOrders+1** elements, copy the old array of orders into this larger array, copy in the new order, delete the old array and assign the **orders** pointer to this newly allocated and filled array. The following represents these steps.

```
// assumes newOrderJustInput contains the new order info
// that is to be added to the array
ORDER* temp = new ORDER [maxOrders+1];
for (k=0; k<maxOrders; k++) {
 temp[k] = orders[k];
}
temp[maxOrders] = newOrderJustInput;
delete [] orders;
orders = temp;
maxOrders++;
```

Figures 6.2 illustrates the copy operations and Figure 6.3 shows the results after the last three instructions in the above series are completed, that is the new state of the orders array.

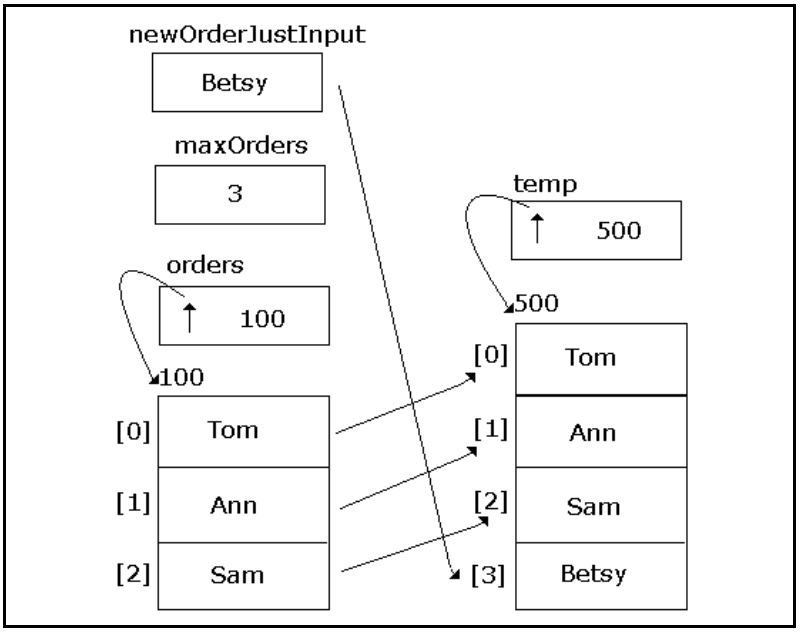

Figure 6.2 Growing the Orders Array — the Copy Operation

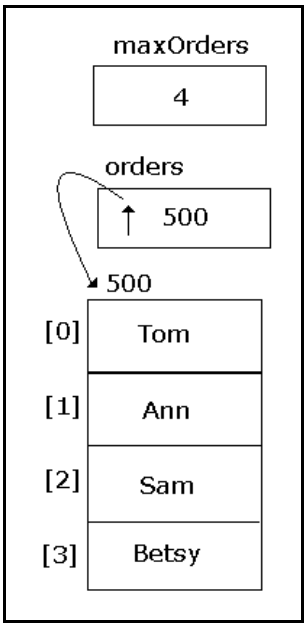

Figure 6.3 New orders

Thus, a simple sequence of allocating a new array one larger than the current array, copying over the current elements and adding the new element, reassigning the array pointer and incrementing the bounds counter allows the array to grow by one element. This simple approach will work just fine, but it has a serious performance problem. Can you spot the execution time inefficiency?

Assume that we start out with **maxOrders** at 0 and that each time the array grows as shown. Supposing that 10,000 orders are input. How many times will element at index 0 be copied into a larger array? 10,000 times! The element at index 1 is copied 9,999 times, and so on. If the order structure is large, our program's execution speed gets slower and slower and slower as more and more elements are added!

In this chapter, since this is our first look at growing an array dynamically at runtime, we will ignore this performance problem. Indeed, if the ultimate size is not too large and the size of the elements is not large, the performance degradation is not noticeably serious. In the ensuing chapters, we will examine alternate schemes to bypass this performance problem.

The next action to consider is how to encapsulate these array growing steps. The obvious choice is to place them into a function, **GrowArray()**. As a first attempt, let's code its prototype as follows.

```
void GrowArray (ORDER* orders, long& maxOrders,
                const ORDER& newOrderJustInput);
```

However, designing the function in this manner represents one of the most common errors programers new to pointers face. It is a very subtle bug. Can you spot the problem just by

looking at the prototype? Extremely well done if you can! Ok. I'll give you a clue. Here is the proposed **GrowArray()** function implementation.

```
void GrowArray (ORDER* orders, long& maxOrders,
                const ORDER& newOrderJustInput) {
 ORDER* temp = new ORDER [maxOrders+1];
 for (long k=0; k<maxOrders; k++) {
  temp[k] = orders[k];
 }
 temp[maxOrders] = newOrderJustInput;
 delete [] orders;
 orders = temp;
 maxOrders++;
}
```

Now, can you spot the insidious bug this design has created? If so, very good indeed. If not, here is a further clue. Here is how the **main()** function invokes **GrowArray()**.

```
ORDER* orders;
long   maxOrders;
ORDER  newOrderJustInput;
...
GrowArray (orders, maxOrders, newOrderJustInput);
```

Now can you spot the gross error? If so, good. If not, pay very close attention.

All parameter variables in C++ are COPIES of what is sent by the caller. **GrowArray()**'s parameter **orders** is a COPY of the memory address of **main()**'s **order**. The error occurs on the second to last line of the **GrowArray()** function.

```
 orders = temp;
```

Where does the new memory address of the new larger array get stored? It is stored in **GrowArray()**'s parameter **orders** variable and NOT in **main()**'s **orders** pointer!

There are several ways to bypass this potential error. We could pass a reference to the orders pointer or we could pass a pointer to the orders pointer. But there is a simpler way. Let's change the design of the function and have **GrowArray()** return the new memory address of the orders array. Here is the new prototype.

```
ORDER* GrowArray (ORDER* orders, long& maxOrders,
                  const ORDER& newOrderJustInput);
```

The **main()** function must now invoke **GrowArray()** this way:

```
ORDER* orders;
long   maxOrders;
ORDER  newOrderJustInput;
...
orders = GrowArray (orders, maxOrders, newOrderJustInput);
```

Here is the new implementation of **GrowArray()**.

```
ORDER* GrowArray (ORDER* orders, long& maxOrders,
                  const ORDER& newOrderJustInput) {
 ORDER* temp = new ORDER [maxOrders+1];
```

```
 for (long k=0; k<maxOrders; k++) {
  temp[k] = orders[k];
 }
 temp[maxOrders] = newOrderJustInput;
 delete [] orders;
 maxOrders++;
 return temp;
}
```

Handling a Variable Number of Command Line Arguments

Before we tackle the full order statistics program, another complexity must be examined. And that has to do with the actual input files themselves. Each day, Acme orders are stored in a daily sales file. This program must input those daily sales files. No problem, we can just open, input, and close each of the five input files. Designing like this is using tunnel vision. What happens on the weeks when the company is closed for a holiday, such as Christmas or Thanksgiving? And what happens on those weeks when the company extends its shopping hours and days to accommodate the expected rush of holiday shoppers? In other words, what I am suggesting is to think about the real-world situation before you dive into program design. Certainly this program will usually expect to input five daily files, but there obviously will be exceptions. Some weeks there will be fewer daily files; some weeks, more daily files.

Let's say that the filenames we need to use will be coded on the command line when the program is launched. The **main()** function has access to these via its parameters usually called **argc** and **argv**, the argument count and the array of string values. Now a design begins to suggest itself. We can write a loop that goes from 1 to **argc** and pass the corresponding filename in **argv** to a **LoadFile()** function whose task is to input that file of orders into the array.

Well, almost. Consider the situation in which the user mis-enters the last filename. If there are a large number of orders in each file, the program will have been executing for some time before it discovers that it cannot open that last file! This is very wasteful of computer and human resources. If a program expects to deal with five input files, then out of consideration for everyone, the program ought to let the user know immediately that one or more of the filenames is incorrect and not make them wait.

One way to do this is to have the **main()** function open and then close each of the proposed input files as its first action, just to verify that there are no user errors on filenames. However, it does take time to open and then close and then later on reopen each input file. So let's get a bit fancier. Let's make an array of **ifstream** objects, one for each of the input files the user has asked us to process. Then, we can open each in turn one time. If all is well, we can then pass those opened file streams on to the **LoadFile()** function. If we encounter an open error, we can notify the user at once.

Pgm06a Acme Weekly Sales Summary Report

Each of the daily sales order files contains the following fields: invoice number, customer account number, order date, item number, quantity, unit cost, tax and the total cost of the order Presumably another company file contains the customer's personal details such as address and credit card information. The company assures us that these daily sales files do not contain any bad data, such as letters where a number should occur.

The sales summary report the program is to create appears as follows. I have attempted to keep the report fairly simple so that we can concentrate on the new actions. Here is the output from the program using the set of sales files I have provided.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Output from Pgm06a                                                          *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 Acme Weekly Sales Summary Report                                        *
*  2                                                                         *
*  3    Item       Qty          Total                                       *
*  4  Number      Sold           Cost                                       *
*  5                                                                        *
*  6   23453       100     $  4250.00                                       *
*  7    3453      1000     $ 50100.00                                       *
*  8    4354       100     $  6065.00                                       *
*  9     253       200     $  2020.00                                       *
* 10   54333        50     $  1667.50                                       *
* 11   54453        50     $  1667.50                                       *
* 12                       ----------                                       *
* 13                     $  65770.00                                        *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Acme has two standard structure definitions that all programs must use. One defines their way of storing date objects and the other defines their order records. Here are those two files that we must use.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Date.h - Acme DATE Structure                                                *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #ifndef DATE_H                                                         *
*  2 #define DATE_H                                                        *
*  3                                                                        *
*  4 /*******************************/                                      *
*  5 /*                             */                                      *
*  6 /* Acme Standard DATE structure */                                     *
*  7 /*                             */                                      *
*  8 /*******************************/                                      *
*  9                                                                        *
* 10 struct DATE {                                                         *
* 11  short month;                                                         *
* 12  short day;                                                           *
* 13  short year;                                                          *
* 14 };                                                                    *
```

```
* 15                                                                          *
* 16 #endif                                                                   *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Order.h - Acme ORDER Structure                                              *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #ifndef ORDER_H                                                         *
*  2 #define ORDER_H                                                        *
*  3                                                                         *
*  4 #include "Date.h"  // needed for the orderDate member                  *
*  5                                                                         *
*  6 /*************************************************************/  *
*  7 /*                                                           */  *
*  8 /* Acme Standard ORDER structure                             */  *
*  9 /*                                                           */  *
* 10 /*************************************************************/  *
* 11                                                                        *
* 12 struct ORDER {                                                         *
* 13                                                                        *
* 14  long   invoiceNum;  // the customer's invoice number                  *
* 15  long   customerNum; // the customer's account number                  *
* 16  DATE   orderDate;   // the date of this order                         *
* 17  long   itemNum;     // the item number of the product purchased       *
* 18  short  quantity;    // the quantity purchased                         *
* 19  double unitCost;    // the price of one of these items                *
* 20  double tax;         // the tax charged on this order                  *
* 21  double totalCost;   // the total cost of this order                   *
* 22 };                                                                     *
* 23                                                                        *
* 24 #endif                                                                 *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

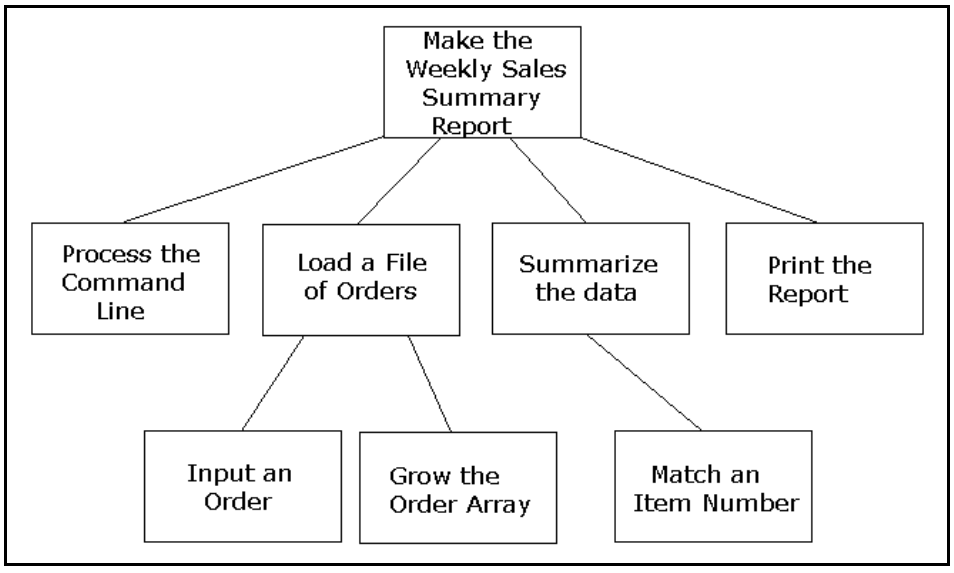

Figure 6.4 The Top-down Design of the Sales Summary Program

As usual, after gaining an understanding of the problem, a Top-down design is created. Figure 6.4 shows the design that I am using.

The **main()** function first calls **ProcessCommandLine()** which dynamically allocates an array of **ifstream** objects and opens all of the user requested files. If one or more files fail to open, then error messages are displayed giving the name of each file that did not open properly and the program is aborted. If all is ok, the address of the array of files is returned to **main()**.

Next, **main()** calls **LoadFile()** for each of the input files. The logic loop for **LoadFile()** is very simple. Call **InputAnOrder()** to actually stream in the next order data. If there is another order, the **GrowArray()** is called to enlarge the array and add in this new order.

Finally, **main()** calls **SummarizeData()** and **PrintReport()** to perform the necessary calculations and display the results. Looking at the report we are to produce, three arrays are needed to store the item number, the corresponding total quantity purchased, and the total cost. However, I do not want to pass three arrays around to these functions much less worry about keeping them parallel arrays. A common simplification to such situations is to define a helper structure to encapsulate these three items. I called this new structure **SALESSUMMARY** and it contains three fields, the item number, quantity, and the cost. Now, **main()** can make a single array of this structure and pass it to the two functions. The maximum array bounds of this summary structure array is determined by the number of individual products the company sells. Here it is set to 100.

SummarizeData() sequentially accesses every order in turn and calls **MatchItemNumber()** to find out if this item number is already in the summary array. If it is, this order's quantity and total are added to the corresponding summary instance. If it is not yet in the summary array, then this item number is added to the summary array along with this order's quantity and total.

Here is the complete Pgm06a program.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Pgm06a.cpp Acme Weekly Sales Summary Program                              *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include <iostream>                                                    *
*  2 #include <iomanip>                                                    *
*  3 #include <fstream>                                                    *
*  4                                                                       *
*  5 using namespace std;                                                 *
*  6                                                                       *
*  7 #include "Order.h"                                                   *
*  8                                                                       *
*  9 ifstream* ProcessCommandLine (int argc, char* argv[]);               *
* 10                                                                       *
* 11 ORDER*    LoadFile (ORDER* orders, long& maxOrders,                   *
* 12                     ifstream& infile);                                *
* 13                                                                       *
```

```
* 14 ifstream& InputAnOrder (ifstream& infile, ORDER& order);          *
* 15                                                                   *
* 16 ORDER*    GrowArray (ORDER* orders, long& maxOrders,              *
* 17                      const ORDER& newOrderJustInput);             *
* 18                                                                   *
* 19 // helper structure for summary totals                           *
* 20 const int MAXITEMS = 100;                                        *
* 21                                                                   *
* 22 struct SALESSUMMARY {                                            *
* 23  long   itemnumber;                                              *
* 24  long   totalQty;                                                *
* 25  double totalCost;                                               *
* 26 };                                                               *
* 27                                                                   *
* 28 void SummarizeData (const ORDER* orders, long numOrders,         *
* 29                     SALESSUMMARY totals[], int& numTotals);      *
* 30                                                                   *
* 31 int  MatchItemNumber (SALESSUMMARY totals[], int numTotals,      *
* 32                       long matchItemNum);                        *
* 33 const int NOMATCH = -1;                                          *
* 34                                                                   *
* 35 void  PrintReport (const SALESSUMMARY totals[], int numTotals);  *
* 36                                                                   *
* 37 /*****************************************************************/  *
* 38 /*                                                               */  *
* 39 /* Pgm06a: Produce Weekly Sales Summary Report                   */  *
* 40 /*                                                               */  *
* 41 /*****************************************************************/  *
* 42                                                                   *
* 43 int main (int argc, char* argv[]) {                              *
* 44                                                                   *
* 45  // construct an array of opened daily sales files              *
* 46  ifstream* files = ProcessCommandLine (argc, argv);              *
* 47                                                                   *
* 48  int    numFiles = argc-1;  // the number of files to process    *
* 49  ORDER* orders = 0;         // the growable array of orders      *
* 50  long   maxOrders = 0;      // the current number of orders      *
* 51  int    i;                                                       *
* 52                                                                   *
* 53  // load all of the orders from all of the files                 *
* 54  for (i=0; i<numFiles; i++) {                                    *
* 55   orders = LoadFile (orders, maxOrders, files[i]);              *
* 56   files[i].close ();                                            *
* 57  }                                                              *
* 58  //remove the files array as it is no longer needed             *
* 59  delete [] files;                                               *
* 60                                                                   *
* 61  // the summarized data helper structure                        *
* 62  SALESSUMMARY totals[MAXITEMS];                                 *
* 63  int          numTotals = 0;                                    *
* 64                                                                   *
* 65  // go summarize all of the order data                          *
```

```
* 66  SummarizeData (orders, maxOrders, totals, numTotals);           *
* 67  // remove the orders array as it is not needed further          *
* 68  delete [] orders;                                               *
* 69                                                                  *
* 70  // finally, display the summarized data                         *
* 71  PrintReport (totals, numTotals);                                *
* 72                                                                  *
* 73  return 0;                                                       *
* 74 }                                                                *
* 75                                                                  *
* 76 /*****************************************************************/  *
* 77 /*                                                               */  *
* 78 /* ProcessCommandLine: verify command line is a proper one      */  *
* 79 /*   consisting of one or more daily sales filenames            */  *
* 80 /*                                                               */  *
* 81 /*   build an array of ifstream objects and attempt to open     */  *
* 82 /*   all of the files                                           */  *
* 83 /*                                                               */  *
* 84 /*   if any file cannot be opened, display an error message     */  *
* 85 /*   that includes that incorrect name, and abort the program*/  *
* 86 /*                                                               */  *
* 87 /*   returns the address of the files array if all is ok        */  *
* 88 /*                                                               */  *
* 89 /*****************************************************************/  *
* 90                                                                  *
* 91 ifstream* ProcessCommandLine (int argc, char* argv[]) {          *
* 92  // guard against no command line files                          *
* 93  if (argc == 1) {                                               *
* 94   cerr << "Error: you must specify one or more daily sales"     *
* 95        << " filenames on the command line\n";                    *
* 96   exit (1);                                                      *
* 97  }                                                               *
* 98                                                                  *
* 99  // allocate the array of file streams                          *
*100  ifstream* files = new ifstream [argc-1];                        *
*101  int i;                                                         *
*102                                                                  *
*103  // attempt to open each file                                   *
*104  for (i=1; i<argc; i++) {                                       *
*105   files[i-1].open (argv[i]);                                     *
*106  }                                                               *
*107                                                                  *
*108  // not display error messages for all that failed to open      *
*109  bool ok = true;                                                *
*110  for (i=1; i<argc; i++) {                                       *
*111   if (!files[i-1]) {                                            *
*112    cerr << "Error: unable to open daily sales file: "          *
*113         << argv[i] << endl;                                      *
*114    ok = false;                                                  *
*115   }                                                             *
*116  }                                                              *
*117                                                                 *
```

```
*118  //if any failed to open, close all files and abort the program  *
*119  if (!ok) {                                                      *
*120   for (i=1; i<argc; i++) {                                       *
*121    files[i-1].close ();                                          *
*122   }                                                              *
*123   delete [] files;                                               *
*124   exit (2);                                                      *
*125  }                                                               *
*126                                                                  *
*127  return files;                                                   *
*128 }                                                               *
*129                                                                  *
*130 /*************************************************************/  *
*131 /*                                                           */  *
*132 /* LoadFile: input all order records in the file and add them*/  *
*133 /*           to the growable orders array                    */  *
*134 /*       return the address of the new larger orders array   */  *
*135 /*                                                           */  *
*136 /*************************************************************/  *
*137                                                                  *
*138 ORDER*    LoadFile (ORDER* orders, long& maxOrders,               *
*139                    ifstream& infile) {                           *
*140  infile >> dec;            // allow for numbers with leading 0's *
*141  ORDER newOrderJustInput;                                        *
*142                                                                  *
*143  while (InputAnOrder (infile, newOrderJustInput)) {              *
*144   orders = GrowArray (orders, maxOrders, newOrderJustInput);     *
*145  }                                                               *
*146  return orders;                                                  *
*147 }                                                               *
*148                                                                  *
*149 /*************************************************************/  *
*150 /*                                                           */  *
*151 /* InputAnOrder: input another set of order data             */  *
*152 /*                                                           */  *
*153 /*************************************************************/  *
*154                                                                  *
*155 ifstream&  InputAnOrder (ifstream& infile, ORDER& order) {       *
*156  char c; // for the / separating mm/dd/yyyy                      *
*157                                                                  *
*158  infile >> order.invoiceNum >> order.customerNum                 *
*159         >> order.orderDate.month >> c                            *
*160         >> order.orderDate.day   >> c                            *
*161         >> order.orderDate.year                                  *
*162         >> order.itemNum                                         *
*163         >> order.quantity                                        *
*164         >> order.unitCost                                        *
*165         >> order.tax                                             *
*166         >> order.totalCost;                                      *
*167                                                                  *
*168  return infile;                                                  *
*169 }                                                               *
```

```
*170                                                                  *
*171 /*****************************************************************/  *
*172 /*                                                               */  *
*173 /* GrowArray: make a new orders array that is one bigger than*/  *
*174 /*            the current one, copy over all the existing    */  *
*175 /*            orders, copy in the new order to be added,     */  *
*176 /*            if the old array exists, delete it,            */  *
*177 /*            increment the number in the array, and return  */  *
*178 /*            the address of the new larger array            */  *
*179 /*                                                           */  *
*180 /*****************************************************************/  *
*181                                                                  *
*182 ORDER* GrowArray (ORDER* orders, long& maxOrders,                 *
*183                         const ORDER& newOrderJustInput) {        *
*184                                                                  *
*185  ORDER* temp = new ORDER [maxOrders+1];                          *
*186                                                                  *
*187  for (long k=0; k<maxOrders; k++) {                              *
*188   temp[k] = orders[k];                                           *
*189  }                                                               *
*190  temp[maxOrders] = newOrderJustInput;                            *
*191                                                                  *
*192  if (orders) delete [] orders;                                   *
*193  maxOrders++;                                                    *
*194  return temp;                                                    *
*195 }                                                                *
*196                                                                  *
*197 /*****************************************************************/  *
*198 /*                                                               */  *
*199 /* SummarizeData: summarize the daily sales information      */  *
*200 /*    builds the sales summary array which contains the item */  *
*201 /*    number and the corresponding quantity and total sales  */  *
*202 /*                                                           */  *
*203 /*   For each order, search the table for a matching item    */  *
*204 /*   number. If not found, add this item. If found, add to   */  *
*205 /*   its quantity and total sales.                           */  *
*206 /*                                                           */  *
*207 /*****************************************************************/  *
*208                                                                  *
*209 void SummarizeData (const ORDER* orders, long numOrders,          *
*210                      SALESSUMMARY totals[], int& numTotals) {    *
*211  int match;                                                      *
*212  for (long i=0; i<numOrders; i++) {                              *
*213   match = MatchItemNumber (totals, numTotals, orders[i].itemNum);*
*214   if (match == NOMATCH) {                                        *
*215    totals[numTotals].itemnumber = orders[i].itemNum;             *
*216    totals[numTotals].totalQty = orders[i].quantity;              *
*217    totals[numTotals].totalCost = orders[i].totalCost;            *
*218    numTotals++;                                                  *
*219   }                                                              *
*220   else {                                                         *
*221    totals[match].totalQty += orders[i].quantity;                 *
```

```
*222    totals[match].totalCost += orders[i].totalCost;                *
*223   }                                                               *
*224  }                                                                *
*225 }                                                                 *
*226                                                                   *
*227 /*************************************************************/  *
*228 /*                                                           */  *
*229 /* MatchItemNumber: match an item number with the array item */  *
*230 /*                  numbers - if no match because this item  */  *
*231 /*                  is not in the array, return NOMATCH      */  *
*232 /*                  otherwise return the subscript of match  */  *
*233 /*                                                           */  *
*234 /*************************************************************/  *
*235                                                                   *
*236 int  MatchItemNumber (SALESSUMMARY totals[], int numTotals,      *
*237                       long matchItemNum) {                        *
*238  for (int i=0; i<numTotals; i++) {                                *
*239   if (totals[i].itemnumber == matchItemNum) return i;             *
*240  }                                                                *
*241  return NOMATCH;                                                 *
*242 }                                                                 *
*243                                                                   *
*244 /*************************************************************/  *
*245 /*                                                           */  *
*246 /* PrintReport: display the summary report                   */  *
*247 /*                                                           */  *
*248 /*************************************************************/  *
*249                                                                   *
*250 void  PrintReport (const SALESSUMMARY totals[], int numTotals) { *
*251  // setup floating point output for dollars and cents             *
*252  cout.setf (ios::fixed, ios::floatfield);                         *
*253  cout << setprecision (2);                                        *
*254                                                                   *
*255  // display heading and two column heading lines                  *
*256  cout << "Acme Weekly Sales Summary Report\n\n";                  *
*257  cout << "   Item       Qty          Total\n"                    *
*258       << " Number      Sold           Cost\n\n";                 *
*259                                                                   *
*260  // accumulator for grand total sales                             *
*261  double grandTotal = 0;                                          *
*262                                                                   *
*263  // display each summary line and accumulate the total dales      *
*264  for (int i=0; i<numTotals; i++) {                                *
*265   cout << setw (7) << totals[i].itemnumber                        *
*266        << setw (10) << totals[i].totalQty                         *
*267        << "     $" << setw (9) << totals[i].totalCost << endl;   *
*268   grandTotal += totals[i].totalCost;                              *
*269  }                                                                *
*270                                                                   *
*271  // display the grand total sales                                 *
*272  cout << "                       ----------\n";                  *
*273  cout << "                    $"                                 *
```

```
*274       << setw (10) << grandTotal << endl;                              *
*275 }                                                                      *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

What about checking for memory leaks? Since I used the new iostreams, if we included the memory leak checking code, it would always report a leak had occurred. That leak is within some Microsoft provided routines and is not ours. Thus, while in development, I checked for leaks, in the final form here, I did not include it because it gives erroneous results that are not easily understood without outside assistance. As a footnote on memory leak checking, I use the third party checker called Bounds Checker made by Nu-Mega Software. When it detects leaks, it also points you to the actual source code where the leak occurred. Thus, with Bounds Checker, you know right away precisely what and where the leak is.

What about testing oracles for **Pgm06a**? What tests would we need to perform to ensure that this program works perfectly? Test1 — run the program with no command line arguments — it should abort with a proper message. Test2 — run with several files that do not exist along with some that do — it should abort with messages on all of the files that do not exist. Test3 — run with a number of files of known data values and see if the totals contain what they should hold. This is the test that I have provided above. Can you think of additional tests that should be performed on this program to ensure it works perfectly?

Unions — an Advanced Feature of Structures

A **union** is a special form of a structure which defines several members but in any given instances of the union, only one of those members is present. Memory for a union is always determined by the size of the largest member it could hold. Consider this union.

```
union Fun {
  int    x;
  long   y;
  double z;
  char   s[10];
};
```

When an instance of **Fun** is allocated, the compiler allocates a total of 10 bytes because member **s** occupies the largest amount of memory.

```
Fun someFun;
```

However, when the program that has allocated **someFun** executes, **someFun** can contain **x**, **y**, **z** or **s**. It can hold only one of these at any moment. But one can place any one of them in **someFun** at will as the following shows.

```
someFun.x = 42;   // it now holds an int x whose value is 42
someFun.z = 98.6; // it now holds the double z, 98.6
strcpy (someFun.s, "Hello"); // it now holds the string s
```

When the union's **x** member is assigned, only 4 bytes of the 10 available are used and contains the integer value of 42. When the union's **z** member is assigned, the contents of **someFun** are

altered and now 8 bytes are used to store the double 98.6. When the union's **s** member is assigned, the **double** is overlaid with the string "hello." Notice that **x**, **y**, **z** and **s** all share the same memory area!

The real problem is knowing which variable is really in the memory at any given moment. But before we can see how this is commonly done, let's examine a special case of a union, called an **anonymous union**. An anonymous union is a union in which no tag and no specific instance is created as the union is being defined. Here is an anonymous union version of the Fun union above.

```
union {
  int    x;
  long   y;
  double z;
  char   s[10];
};
```

Since there is no tag, no other instances of this union can be created; there is no name to use as the data type.

Anonymous unions have a peculiar property. That is, the member names are not local to the union instance as are those in **Fun** above. Instead, the member names are treated as if they were defined immediately outside of the anonymous union and thus take on the definition aspects found there. That is, they act like they are members of the block that surrounds them. For example, here I have defined another structure around the anonymous one.

```
struct FUN {
     char which;
     union {
       int    x;
       long   y;
       double z;
       char   s[10];
     };
};
```

Here the structure **FUN** has two data members, which and either **x**, **y**, **z** or **s**; **FUN** occupies 11 bytes of memory, 1 for the **char** and 10 for the largest of the union members. Now if I create an instance of **FUN** called **fun**, we can get access to the union members because the act as if they were members of the surrounding block, **FUN** in this case.

```
FUN fun;
fun.which = 1;
fun.z = 42;
strcpy (fun.s, "Hello");
```

Variant Records

Ok, but what can we do with these unions? A union gives us the ability to store a single array of heterogenous data types! That is, we can now have an array whose elements are not of the same data type. We force the differences between data types to be in the union portion of a structure. However, with any specific instance of the outer structure, there must be a member outside the union that lets us know which one of the union members is in this specific instance. This concept is known as variant records. Variant records are a very powerful feature and very useful in advanced programming situations. They are widely used in Active-X and COM type programming.

Let's make a realistic example of a variant record. Suppose that our company has several forms that a date may take, depending upon the varying needs of applications. Further, the company wishes to have a uniform method for date processing across all applications. The forms the date may take are shown below.

"01/02/2000 " as a string
1 2 2000 as three shorts
155 2000 as two shorts representing the day of the year and the year
"January 2, 2000" as a string

The company can define a variant **DATE** structure which can hold any of these four forms. All applications can be passed an instance of this single **DATE** structure, independent of the precise format that it contains. Here could be that definition.

```
#ifndef DATE_H
#define DATE_H

enum DateFormat {MonthDayYear, DayYear,
                 StringFormat, EnglishString};

struct DATEMDY {
  short month;
  short day;
  short year;
};

struct DATEDY {
  short day;
  short year;
};

struct DATE {
  DateFormat type;
  union {
   DATEMDY dateMDY;
```

```
        DATEDY  dateDY;
        char    dateString[11];
        char    dateEnglish[19];
      };
};
#endif
```

A user program could then define an instance of **DATE** and store a month, day and year as shown below.

```
DATE d;
d.type = MonthDayYear;
d.dateMDY.month = 1;
d.dateMDY.day = 20;
d.dateMDY.year = 2001;
```

The important factor to note is this abstraction of a date yields a code reuse and commonality across all applications a company has. All applications that need a date object create instances of the **DATE** variant structure. All functions that are passed a date use an instance of this **DATE** structure.

Further code reuse can be achieved by writing a single function to handle all forms of date conversion. This is known as a utility function. In fact, that is precisely what our next application will be, a utility function to handle conversion of a date into the various forms.

Rather than make this utility function overly complex, let's make two assumptions that, while completely not valid in practice, will make this example easier to do so that we can follow this new variant record processing. I completely ignore leap years! No indication is given for the reason for a failure to perform the conversion.

The utility function should be called **ConvertDate()** and is passed a reference to a **DATE** object which holds the date to be converted and an instance of the enumerated data type that defines which conversion is to be performed. The answer will replace the original contents of the passed **DATE** instance, so the function returns a **bool**, **true** if the conversion was successful and **false** if it failed. The enum that defines the conversion is

```
enum DateConversion {ToMDY, ToDY, ToString, ToEnglish};
```

The function prototype is

```
bool ConvertDate (DATE& date, DateConversion which);
```

Here is the **Date.h** header file that is used in every Acme company application that processes a date.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Date.h - DATE Variant Record Definition                                    *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #ifndef DATE_H                                                          *
*  2 #define DATE_H                                                         *
*  3                                                                         *
*  4 /*********************************************************************/ *
```

```
*  5 /*                                                                 */ *
*  6 /* Acme Standard DATE Structure - variant records                  */ *
*  7 /*                                                                 */ *
*  8 /*****************************************************************/ *
*  9                                                                      *
* 10 // DateFormat: identifies which type of data is in the variant      *
* 11 enum DateFormat {MonthDayYear, DayYear,                             *
* 12                  StringFormat, EnglishString};                      *
* 13                                                                     *
* 14 // the variant fields when in M D Y format                         *
* 15 struct DATEMDY {                                                   *
* 16   short month;                                                     *
* 17   short day;                                                       *
* 18   short year;                                                      *
* 19 };                                                                 *
* 20                                                                     *
* 21 // the variant fields when in D Y format                           *
* 22 struct DATEDY {                                                    *
* 23   short day;                                                       *
* 24   short year;                                                      *
* 25 };                                                                 *
* 26                                                                     *
* 27 // the variant DATE structure                                      *
* 28 struct DATE {                                                      *
* 29   DateFormat type; // tells which variant is present               *
* 30   union {                                                          *
* 31    DATEMDY dateMDY;                                                *
* 32    DATEDY  dateDY;                                                 *
* 33    char    dateString[11];                                         *
* 34    char    dateEnglish[19];                                        *
* 35   };                                                               *
* 36 };                                                                 *
* 37                                                                     *
* 38 // DateConversion defines which type of date conversion            *
* 39 //                 is to be done                                   *
* 40 enum DateConversion {ToMDY, ToDY, ToString, ToEnglish};            *
* 41                                                                     *
* 42 // ConvertDate: converts date into the which form of date          *
* 43 // is requested. date is replaced with the new form if             *
* 44 // successful. Function returns true if conversion is ok           *
* 45 // false if it fails for any reason                                *
* 46 bool ConvertDate (DATE& date, DateConversion which);               *
* 47                                                                     *
* 48 #endif                                                             *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Notice how the member **type** is used to let us know which of the four variants is actually present in any given instance of **DATE**. Here is the **ConvertDate()** function. It is rather lengthy because of all of the possible conversion.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* ConvertDate.cpp Converts a DATE into Another Format                  *
```

```
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include <iostream>                                                        *
*  2 #include <iomanip>                                                        *
*  3 #include <strstream>                                                      *
*  4 #include <string>                                                        *
*  5                                                                          *
*  6 using namespace std;                                                     *
*  7                                                                          *
*  8 #include "Date.h"                                                       *
*  9                                                                          *
* 10                                                                          *
* 11 const short daysInMonth[12] = {31, 28, 31, 30, 31, 30,                   *
* 12                                31, 31, 30, 31, 30, 31};                  *
* 13                                                                          *
* 14 const short daysYear[12] = {31, 59, 90, 120, 151, 181,                   *
* 15                             212, 243, 273, 304, 334, 365};               *
* 16                                                                          *
* 17 const char monthNames[12][10] = {"January", "February", "March",         *
* 18         "April", "May", "June", "July", "August", "September",           *
* 19         "October", "November", "December"};                              *
* 20                                                                          *
* 21 /*************************************************************/*
* 22 /*                                                           */*
* 23 /* ConvertDate: Utility function to convert a date from any of */*
* 24 /*              four forms into any of four forms            */*
* 25 /*              It returns true if successful, false if fails  */*
* 26 /*                                                           */*
* 27 /* It is passed the DATE object and the requested to conversion*/*
* 28 /* If the converison is successful, it returns the new form in */*
* 29 /*    the original passed DATE object                        */*
* 30 /*                                                           */*
* 31 /* From Forms: M D Y; D Y; "mm/dd/yyyy"; "mmmmmmmmmm dd, yyyy" */*
* 32 /*      as defined by DateFormat enum:                       */*
* 33 /*      MonthDayYear, DayYear, StringFormat, EnglishString     */*
* 34 /*                                                           */*
* 35 /* To Forms the same as defined by DateConversion enum       */*
* 36 /*      ToMDY, ToDY, ToString, ToEnglish                     */*
* 37 /*                                                           */*
* 38 /*************************************************************/*
* 39                                                                          *
* 40 bool ConvertDate (DATE& date, DateConversion which) {                    *
* 41  // common workareas                                                     *
* 42  short m, d, y;                                                         *
* 43  char c;                                                                *
* 44  char sm[10];                                                           *
* 45  char ansr[11];                                                         *
* 46  char ansrb[19];                                                        *
* 47                                                                         *
* 48  /*************************************************************/*
* 49  /*                                                           */*
* 50  /* Converting to the M D Y trio of numbers                   */*
* 51  /*                                                           */*
```

```
* 52  /*************************************************************/*
* 53                                                                *
* 54  if (which == ToMDY) {                                         *
* 55   // original date already is in M D Y format                  *
* 56   if (date.type == MonthDayYear)                               *
* 57    return true;                                                *
* 58                                                                *
* 59   // original date is in D Y format                           *
* 60   if (date.type == DayYear) {                                  *
* 61    y = date.dateDY.year;                                       *
* 62    int i = 0;                                                  *
* 63    // find the right month and day                             *
* 64    while (i<12) {                                              *
* 65     if (date.dateDY.day <= daysYear[i]) {                      *
* 66      m = i + 1;                                                *
* 67      if (i>0)                                                  *
* 68       d = date.dateDY.day - daysYear[i-1];                     *
* 69      else                                                      *
* 70       d = date.dateDY.day;                                     *
* 71      // fill up the answer instance                            *
* 72      date.type = MonthDayYear;                                 *
* 73      date.dateMDY.month = m;                                   *
* 74      date.dateMDY.day = d;                                     *
* 75      date.dateMDY.year = y;                                    *
* 76      return true;                                              *
* 77     }                                                          *
* 78     i++;                                                       *
* 79    }                                                           *
* 80    return false; // invalid number of days in original date    *
* 81   }                                                           *
* 82                                                                *
* 83   // original date is in 01/01/2000 format                     *
* 84   else if (date.type == StringFormat) {                       *
* 85    istrstream is (date.dateString);                            *
* 86    is >> dec >> m >> c >> d >> c >> y;                         *
* 87    if (!is)                                                    *
* 88     return false; // invalid digits in string date             *
* 89    // fill up the answer instance                              *
* 90    date.type = MonthDayYear;                                   *
* 91    date.dateMDY.month = m;                                     *
* 92    date.dateMDY.day = d;                                       *
* 93    date.dateMDY.year = y;                                      *
* 94    return true;                                                *
* 95   }                                                           *
* 96                                                                *
* 97   // original date is in January 1, 2000 form                 *
* 98   else if (date.type == EnglishString) {                       *
* 99    istrstream is (date.dateEnglish);                           *
*100    is >> sm >> d >> c >> y;                                    *
*101    if (!is)                                                    *
*102     return false; // invalid data in English date string       *
*103    bool found = false;                                         *
```

```
*104    int i=0;                                                        *
*105    while (i<12) {                                                  *
*106     if (stricmp (sm, monthNames[i]) == 0) {                        *
*107      m = i + 1;                                                    *
*108      found = true;                                                 *
*109      break;                                                        *
*110     }                                                              *
*111     i++;                                                           *
*112    }                                                               *
*113    if (!found)                                                     *
*114     return false; // invalid month name spelling in string         *
*115    // fill up the answer instance                                   *
*116    date.type = MonthDayYear;                                       *
*117    date.dateMDY.month = m;                                         *
*118    date.dateMDY.day = d;                                           *
*119    date.dateMDY.year = y;                                          *
*120    return true;                                                    *
*121   }                                                                *
*122                                                                    *
*123   // original date is an invalid form                              *
*124   else                                                            *
*125    return false;                                                   *
*126  }                                                                 *
*127                                                                    *
*128  /*****************************************************************/*
*129  /*                                                               */*
*130  /* Converting to the D Y pair of numbers                         */*
*131  /*                                                               */*
*132  /*****************************************************************/*
*133                                                                    *
*134  else if (which == ToDY) {                                        *
*135   // original date already is in D Y format                        *
*136   if (date.type == DayYear)                                       *
*137    return true;                                                   *
*138                                                                   *
*139   // original date is in M D Y form                               *
*140   if (date.type == MonthDayYear) {                                *
*141    d = date.dateMDY.day;                                          *
*142    m = date.dateMDY.month;                                        *
*143    y = date.dateMDY.year;                                         *
*144    if (m != 1) {                                                  *
*145     d += daysYear[m-2];                                           *
*146    }                                                              *
*147    // fill up the answer instance                                 *
*148    date.type = DayYear;                                           *
*149    date.dateDY.day = d;                                           *
*150    date.dateDY.year = y;                                          *
*151    return true;                                                   *
*152   }                                                               *
*153                                                                   *
*154   // original date is in January 1, 2000 form                     *
*155   else if (date.type == EnglishString) {                          *
```

```
*156    istrstream is (date.dateEnglish);                                *
*157    is >> sm >> d >> c >> y;                                         *
*158    if (!is)                                                         *
*159     return false; // invalid data in English string                 *
*160    bool found = false;                                              *
*161    int i=0;                                                         *
*162    // find the month from the string name                           *
*163    while (i<12) {                                                   *
*164     if (stricmp (sm, monthNames[i]) == 0) {                         *
*165      m = i + 1;                                                     *
*166      found = true;                                                  *
*167      break;                                                         *
*168     }                                                               *
*169     i++;                                                            *
*170    }                                                                *
*171    if (!found)                                                      *
*172     return false; // invalid month spelling                         *
*173    if (m != 1) {                                                    *
*174     d += daysYear[m-2];                                             *
*175    }                                                                *
*176    // fill up the answer instance                                   *
*177    date.type = DayYear;                                             *
*178    date.dateDY.day = d;                                             *
*179    date.dateDY.year = y;                                            *
*180    return true;                                                     *
*181   }                                                                 *
*182                                                                     *
*183   // original string is in 01/01/2000 format                        *
*184   else if (date.type == StringFormat) {                             *
*185    istrstream is (date.dateString);                                 *
*186    is >> dec >> m >> c >> d >> c >> y;                              *
*187    if (!is)                                                         *
*188     return false; // invalid digits in the date string              *
*189    if (m != 1) {                                                    *
*190     d += daysYear[m-2];                                             *
*191    }                                                                *
*192    // fill up the answer instance                                   *
*193    date.type = DayYear;                                             *
*194    date.dateDY.day = d;                                             *
*195    date.dateDY.year = y;                                            *
*196    return true;                                                     *
*197   }                                                                 *
*198                                                                     *
*199   // original date has an invalid format                            *
*200   else                                                              *
*201    return false;                                                    *
*202  }                                                                  *
*203                                                                     *
*204  /*****************************************************************/*
*205  /*                                                               */*
*206  /* Converting to string form of mm/dd/yyyy                       */*
*207  /*                                                               */*
```

```
*208  /*************************************************************/*
*209                                                                 *
*210  else if (which == ToString) {                                 *
*211   // is original date already in string format?                 *
*212   if (date.type == StringFormat)                                *
*213    return true;                                                 *
*214                                                                 *
*215   // original date is in M D Y form                             *
*216   if (date.type == MonthDayYear) {                              *
*217    ostrstream os (ansr, sizeof (ansr));                         *
*218    os << setfill('0') << setw (2) << date.dateMDY.month         *
*219       << "/" << setw (2) << date.dateMDY.day                    *
*220       << "/" << setw (4) << date.dateMDY.year << ends;          *
*221    // fill up the answer instance                               *
*222    date.type = StringFormat;                                    *
*223    strcpy (date.dateString, ansr);                              *
*224    return true;                                                 *
*225   }                                                             *
*226                                                                 *
*227   // original date is in D Y form                               *
*228   else if (date.type == DayYear) {                              *
*229    d = date.dateDY.day;                                         *
*230    y = date.dateDY.year;                                        *
*231    int i = 0;                                                   *
*232    // find the month and day                                    *
*233    while (i<12) {                                               *
*234     if (d <= daysYear[i]) {                                     *
*235      m = i + 1;                                                 *
*236      if (i>0)                                                   *
*237       d = date.dateDY.day - daysYear[i-1];                      *
*238      else                                                       *
*239       d = date.dateDY.day;                                      *
*240      ostrstream os (ansr, sizeof (ansr));                       *
*241      os << setfill('0') << setw (2) << m                        *
*242         << "/" << setw (2) << d                                 *
*243         << "/" << setw (4) << y << ends;                        *
*244      // fill up the answer instance                             *
*245      date.type = StringFormat;                                  *
*246      strcpy (date.dateString, ansr);                            *
*247      return true;                                               *
*248     }                                                           *
*249     i++;                                                        *
*250    }                                                            *
*251    return false; // invalid number of days                      *
*252   }                                                             *
*253                                                                 *
*254   // original date is in January 1, 2000 form                   *
*255   else if (date.type == EnglishString) {                        *
*256    istrstream is (date.dateEnglish);                            *
*257    is >> sm >> d >> c >> y;                                     *
*258    if (!is)                                                     *
*259     return false; // invalid data in string date                *
```

```
*260    bool found = false;                                              *
*261    int i=0;                                                         *
*262    // find the month from the string month name                     *
*263    while (i<12) {                                                   *
*264     if (stricmp (sm, monthNames[i]) == 0) {                         *
*265      m = i + 1;                                                     *
*266      found = true;                                                  *
*267      break;                                                         *
*268     }                                                               *
*269     i++;                                                            *
*270    }                                                                *
*271    if (!found)                                                      *
*272     return false; // invalid month spelling                         *
*273    ostrstream os (ansr, sizeof (ansr));                             *
*274    os << setfill('0') << setw (2) << m                              *
*275       << "/" << setw (2) << d                                       *
*276       << "/" << setw (4) << y << ends;                              *
*277    // fill up the answer instance                                   *
*278    date.type = StringFormat;                                        *
*279    strcpy (date.dateString, ansr);                                  *
*280    return true;                                                     *
*281   }                                                                 *
*282                                                                     *
*283   // original date has an invalid format                            *
*284   else                                                              *
*285    return false;                                                    *
*286  }                                                                  *
*287                                                                     *
*288  /*****************************************************************/*
*289  /*                                                               */*
*290  /* Converting to the string form of January 1, 2000              */*
*291  /*                                                               */*
*292  /*****************************************************************/*
*293                                                                     *
*294  else if (which == ToEnglish) {                                     *
*295   // is it already in English format?                               *
*296   if (date.type == EnglishString)                                   *
*297    return true;                                                     *
*298                                                                     *
*299   // original date is M D Y form                                    *
*300   if (date.type == MonthDayYear) {                                  *
*301    ostrstream os (ansrb, sizeof (ansrb));                           *
*302    os << monthNames[date.dateMDY.month-1] << " "                    *
*303       << date.dateMDY.day << ", "                                   *
*304       << date.dateMDY.year << ends;                                 *
*305    date.type = EnglishString;                                       *
*306    strcpy (date.dateEnglish, ansrb);                                *
*307    return true;                                                     *
*308   }                                                                 *
*309                                                                     *
*310   // original date is D Y form                                      *
*311   else if (date.type == DayYear) {                                  *
```

```
*312    y = date.dateDY.year;                                          *
*313    int i = 0;                                                     *
*314    while (i<12) {                                                 *
*315     if (date.dateDY.day <= daysYear[i]) {                         *
*316      m = i + 1;                                                   *
*317      if (i>0)                                                     *
*318       d = date.dateDY.day - daysYear[i-1];                        *
*319      else                                                         *
*320       d = date.dateDY.day;                                        *
*321      ostrstream os (ansrb, sizeof (ansrb));                       *
*322      os << monthNames[m-1] << " "                                 *
*323         << d << ", " << y << ends;                                *
*324      date.type = EnglishString;                                   *
*325      strcpy (date.dateEnglish, ansrb);                            *
*326      return true;                                                 *
*327     }                                                             *
*328     i++;                                                          *
*329    }                                                              *
*330    return false; // invalid number of days in original date       *
*331   }                                                               *
*332                                                                   *
*333   // original date is in string format 01/01/2000                 *
*334   else if (date.type == StringFormat) {                           *
*335    istrstream is (date.dateString);                               *
*336    is >> dec >> m >> c >> d >> c >> y;                            *
*337    if (!is)                                                       *
*338     return false; // invalid digits in string date                *
*339    ostrstream os (ansrb, sizeof (ansrb));                         *
*340    os << monthNames[m-1] << " "                                   *
*341       << d << ", " << y << ends;                                  *
*342    date.type = EnglishString;                                     *
*343    strcpy (date.dateEnglish, ansrb);                              *
*344    return true;                                                   *
*345   }                                                               *
*346                                                                   *
*347   // invalid original format                                      *
*348   else                                                            *
*349    return false;                                                  *
*350  }                                                                *
*351                                                                   *
*352  // here invalid convert to format                                *
*353  else                                                             *
*354   return false;                                                   *
*355 }                                                                 *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Look over the **ConvertDate()** function. It could have been broken down into a number of subfunctions, one for each specific conversion. The logic is fairly simple; the length is due primarily to the fact the each of four possible conversions has three implementations for a total of twelve conversion sequences. However, they are in part very similar actions. Notice that I made extensive use of **istrstream** and **ostrstream** classes to conveniently extract or insert the data.

The overall logic is first split the conversion request into the four categories of conversion. Next, within a specific category, such as convert to MDY format, then split based upon the four possible current date formats. Of course, if the date is already in MDY format, then there is no conversion necessary, just return true. Let's look at two of these conversions.

One of the harder conversions is to get the month, day and year from a date in the form of day of the year and year.

```
  // original date is in D Y format
  if (date.type == DayYear) {
   y = date.dateDY.year;
   int i = 0;
```

My solution is to loop through the **daysYear** array and find that point where the given days is below the array value. There, the subscript plus one gives the month. The days is dependent on whether it is in January or not.

```
   while (i<12) {
    if (date.dateDY.day <= daysYear[i]) {
     m = i + 1;
     if (i>0)
      d = date.dateDY.day - daysYear[i-1];
     else
      d = date.dateDY.day;
```

With the month, day and year now known, those values are then placed back into the date variant.

```
     date.type = MonthDayYear;
     date.dateMDY.month = m;
     date.dateMDY.day = d;
     date.dateMDY.year = y;
     return true;
    }
    i++;
   }
   return false; // invalid number of days in original date
  }
```

Next, let's see how I got the three numerical values from the English form, such as January 1, 2000. After constructing an **istrstream** on the variant string, the string month name is extracted along with the day and year.

```
  // original date is in January 1, 2000 form
  else if (date.type == EnglishString) {
   istrstream is (date.dateEnglish);
   is >> sm >> d >> c >> y;
   if (!is)
    return false; // invalid data in English date string
```

Then I loop through the array of **monthNames** looking for a string insensitive match. When

found, the subscript plus one is the month and the assignment can be made.

```
   bool found = false;
   int i=0;
   while (i<12) {
    if (stricmp (sm, monthNames[i]) == 0) {
     m = i + 1;
     found = true;
     break;
    }
    i++;
   }
   if (!found)
    return false; // invalid month name spelling in string
   // fill up the answer instance
   date.type = MonthDayYear;
   date.dateMDY.month = m;
   date.dateMDY.day = d;
   date.dateMDY.year = y;
   return true;
  }
```

Finally, we need to devise a set of testing oracles. I created four test files with various dates in each of the four forms. A tester program can then input each date and attempt to convert that date into each of its three other forms, displaying the results. In this manner, we can more easily verify the results. Here is the tester program.

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Pgm06b.cpp - Test Driver to Verify ConvertDate() Works                    *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1 #include <iostream>                                                   *
*  2 #include <iomanip>                                                    *
*  3 #include <fstream>                                                    *
*  4 #include <strstream>                                                  *
*  5 #include <string>                                                     *
*  6                                                                       *
*  7 using namespace std;                                                 *
*  8                                                                       *
*  9 #include "Date.h"                                                    *
* 10                                                                       *
* 11 /*************************************************************/ *
* 12 /*                                                           */ *
* 13 /* Pgm06b: Testing driver for ConvertDate() function         */ *
* 14 /*                                                           */ *
* 15 /*************************************************************/ *
* 16                                                                       *
* 17 int main () {                                                         *
* 18                                                                       *
* 19  /*************************************************************/*
* 20  /*                                                           */*
* 21  /* test forms of original date of M D Y shorts              */*
```

```
* 22  /*                                                                */*
* 23  /******************************************************************/*
* 24                                                                     *
* 25  ifstream infile ("testMDY.txt");                                   *
* 26  if (!infile) {                                                     *
* 27   cerr << "Error: cannot open testMDY.txt file\n";                  *
* 28   return 1;                                                         *
* 29  }                                                                  *
* 30                                                                     *
* 31  DATE originalMDY, d;                                               *
* 32  originalMDY.type = MonthDayYear;                                   *
* 33  infile >> dec;                                                     *
* 34  while (infile >> originalMDY.dateMDY.month                         *
* 35                >> originalMDY.dateMDY.day                           *
* 36                >> originalMDY.dateMDY.year) {                       *
* 37                                                                     *
* 38   d = originalMDY;                                                  *
* 39   cout << "\nOriginal MDY Date from the file: "                      *
* 40        << setfill('0') << setw(2) << d.dateMDY.month << "/"         *
* 41        << setw(2) << d.dateMDY.day << "/" << d.dateMDY.year         *
* 42        << setfill (' ') << endl;                                    *
* 43                                                                     *
* 44   if (ConvertDate (d, ToDY)) {                                      *
* 45    cout << "                                    "                   *
* 46         << d.dateDY.day << "/" << d.dateDY.year << endl;            *
* 47   }                                                                 *
* 48   else cout << "oops!\n";                                           *
* 49                                                                     *
* 50   d = originalMDY;                                                  *
* 51   if (ConvertDate (d, ToString)) {                                  *
* 52    cout << "                                    "                   *
* 53         << d.dateString << endl;                                    *
* 54   }                                                                 *
* 55   else cout << "oops!\n";                                           *
* 56                                                                     *
* 57   d = originalMDY;                                                  *
* 58   if (ConvertDate (d, ToEnglish)) {                                 *
* 59    cout << "                                    "                   *
* 60         << d.dateEnglish << endl;                                   *
* 61   }                                                                 *
* 62   else cout << "oops!\n";                                           *
* 63  }                                                                  *
* 64                                                                     *
* 65  /******************************************************************/*
* 66  /*                                                                */*
* 67  /* test forms of original date of D Y shorts                     */*
* 68  /*                                                                */*
* 69  /******************************************************************/*
* 70                                                                     *
* 71  infile.clear(); // clear eof flag                                  *
* 72                                                                     *
* 73  infile.close();                                                    *
```

```
* 74  infile.open ("testDY.txt");                                        *
* 75  if (!infile) {                                                     *
* 76   cerr << "Error: cannot open testDY.txt\n";                        *
* 77   return 1;                                                         *
* 78  }                                                                  *
* 79                                                                     *
* 80  DATE originalDY;                                                   *
* 81  originalDY.type = DayYear;                                         *
* 82  infile >> dec;                                                     *
* 83  while (infile >> originalDY.dateDY.day                             *
* 84                >> originalDY.dateDY.year) {                        *
* 85                                                                     *
* 86   d = originalDY;                                                   *
* 87   cout << "\nOriginal DY Date from the file:  "                      *
* 88        << d.dateDY.day << "/" << d.dateDY.year << endl;             *
* 89                                                                     *
* 90   if (ConvertDate (d, ToMDY)) {                                     *
* 91    cout << "                                    "                  *
* 92         << setfill('0') << setw(2) << d.dateMDY.month << "/"       *
* 93         << setw(2) << d.dateMDY.day << "/" << d.dateMDY.year       *
* 94         << setfill (' ') << endl;                                   *
* 95   }                                                                 *
* 96   else cout << "oops!\n";                                          *
* 97                                                                     *
* 98   d = originalDY;                                                   *
* 99   if (ConvertDate (d, ToString)) {                                  *
*100    cout << "                                    "                  *
*101         << d.dateString << endl;                                    *
*102   }                                                                 *
*103   else cout << "oops!\n";                                          *
*104                                                                     *
*105   d = originalDY;                                                   *
*106   if (ConvertDate (d, ToEnglish)) {                                 *
*107    cout << "                                    "                  *
*108         << d.dateEnglish << endl;                                   *
*109   }                                                                 *
*110   else cout << "oops!\n";                                          *
*111  }                                                                  *
*112                                                                     *
*113  /*****************************************************************/*
*114  /*                                                               */*
*115  /* test forms of original date of strings 01/01/2000             */*
*116  /*                                                               */*
*117  /*****************************************************************/*
*118                                                                     *
*119  infile.clear(); // clear eof flag                                  *
*120                                                                     *
*121  infile.close();                                                    *
*122  infile.open ("testString.txt", ios::in);                           *
*123  if (!infile) {                                                     *
*124   cerr << "Error: cannot open testString.txt\n";                    *
*125   return 1;                                                        *
```

```
*126  }                                                                  *
*127                                                                     *
*128  DATE originalString;                                              *
*129  originalString.type = StringFormat;                               *
*130  infile >> dec;                                                     *
*131  while (infile >> originalString.dateString) {                     *
*132                                                                     *
*133   d = originalString;                                              *
*134   cout << "\nOriginal String from the file:   "                     *
*135        << d.dateString << endl;                                     *
*136                                                                     *
*137   if (ConvertDate (d, ToMDY)) {                                    *
*138    cout << "                                    "                  *
*139         << setfill('0') << setw(2) << d.dateMDY.month << "/"        *
*140         << setw(2) << d.dateMDY.day << "/" << d.dateMDY.year        *
*141         << setfill (' ') << endl;                                   *
*142   }                                                                *
*143   else cout << "oops!\n";                                          *
*144                                                                     *
*145   d = originalString;                                              *
*146   if (ConvertDate (d, ToDY)) {                                     *
*147    cout << "                                    "                  *
*148         << d.dateDY.day << "/" << d.dateDY.year << endl;            *
*149   }                                                                *
*150   else cout << "oops!\n";                                          *
*151                                                                     *
*152   d = originalString;                                              *
*153   if (ConvertDate (d, ToEnglish)) {                                *
*154    cout << "                                    "                  *
*155         << d.dateEnglish << endl;                                   *
*156   }                                                                *
*157   else cout << "oops!\n";                                          *
*158  }                                                                  *
*159                                                                     *
*160  /*****************************************************************/*
*161  /*                                                               */*
*162  /* test forms of original date of English January 1, 2000        */*
*163  /*                                                               */*
*164  /*****************************************************************/*
*165                                                                     *
*166  infile.clear(); // clear eof flag                                  *
*167                                                                     *
*168  infile.close();                                                    *
*169  infile.open ("testEnglish.txt", ios::in);                          *
*170  if (!infile) {                                                     *
*171   cerr << "Error: cannot open testEnglish.txt\n";                   *
*172   return 1;                                                        *
*173  }                                                                  *
*174                                                                     *
*175  DATE originalEnglish;                                              *
*176  originalEnglish.type = EnglishString;                              *
*177  while (infile.getline(originalEnglish.dateEnglish,                 *
```

```
*178                 sizeof(originalEnglish.dateEnglish))) {          *
*179                                                                  *
*180   d = originalEnglish;                                           *
*181   cout << "\nOriginal English from the file:  "                   *
*182        << d.dateEnglish << endl;                                 *
*183   if (ConvertDate (d, ToMDY)) {                                  *
*184    cout << "                                    "                *
*185         << setfill('0') << setw(2) << d.dateMDY.month << "/"     *
*186         << setw(2) << d.dateMDY.day << "/" << d.dateMDY.year     *
*187         << setfill (' ') << endl;                                *
*188   }                                                              *
*189   else cout << "oops!\n";                                        *
*190                                                                  *
*191   d = originalEnglish;                                           *
*192   if (ConvertDate (d, ToDY)) {                                   *
*193    cout << "                                    "                *
*194         << d.dateDY.day << "/" << d.dateDY.year << endl;         *
*195   }                                                              *
*196   else cout << "oops!\n";                                        *
*197                                                                  *
*198   d = originalEnglish;                                           *
*199   if (ConvertDate (d, ToString)) {                               *
*200    cout << "                                    "                *
*201         << d.dateString << endl;                                 *
*202   }                                                              *
*203   else cout << "oops!\n";                                        *
*204  }                                                               *
*205  infile.close();                                                 *
*206                                                                  *
*207  return 0;                                                       *
*208 }                                                                *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

What dates should be checked? One could put all 365 possibilities in each of the files. However, I picked the extremes and several critical ones on either side of a month change. Here are the test results. After looking these over, are there other dates that should have been tested to help guarantee the function works as expected?

```
+)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))),
* Pgm06b Output Results                                               *
/)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))1
*  1                                                                  *
*  2 Original MDY Date from the file: 01/01/2000                      *
*  3                                  1/2000                          *
*  4                                  01/01/2000                      *
*  5                                  January 1, 2000                 *
*  6                                                                  *
*  7 Original MDY Date from the file: 01/31/2000                      *
*  8                                  31/2000                         *
*  9                                  01/31/2000                      *
* 10                                  January 31, 2000                *
* 11                                                                 *
```

```
* 12 Original MDY Date from the file: 02/01/2000                  *
* 13                                  32/2000                     *
* 14                                  02/01/2000                  *
* 15                                  February 1, 2000            *
* 16                                                              *
* 17 Original MDY Date from the file: 02/28/2000                  *
* 18                                  59/2000                     *
* 19                                  02/28/2000                  *
* 20                                  February 28, 2000           *
* 21                                                              *
* 22 Original MDY Date from the file: 03/01/2000                  *
* 23                                  60/2000                     *
* 24                                  03/01/2000                  *
* 25                                  March 1, 2000               *
* 26                                                              *
* 27 Original MDY Date from the file: 12/01/2000                  *
* 28                                  335/2000                    *
* 29                                  12/01/2000                  *
* 30                                  December 1, 2000            *
* 31                                                              *
* 32 Original MDY Date from the file: 12/31/2000                  *
* 33                                  365/2000                    *
* 34                                  12/31/2000                  *
* 35                                  December 31, 2000           *
* 36                                                              *
* 37 Original DY Date from the file:  1/2000                      *
* 38                                  01/01/2000                  *
* 39                                  01/01/2000                  *
* 40                                  January 1, 2000             *
* 41                                                              *
* 42 Original DY Date from the file:  31/2000                     *
* 43                                  01/31/2000                  *
* 44                                  01/31/2000                  *
* 45                                  January 31, 2000            *
* 46                                                              *
* 47 Original DY Date from the file:  32/2000                     *
* 48                                  02/01/2000                  *
* 49                                  02/01/2000                  *
* 50                                  February 1, 2000            *
* 51                                                              *
* 52 Original DY Date from the file:  59/2000                     *
* 53                                  02/28/2000                  *
* 54                                  02/28/2000                  *
* 55                                  February 28, 2000           *
* 56                                                              *
* 57 Original DY Date from the file:  60/2000                     *
* 58                                  03/01/2000                  *
* 59                                  03/01/2000                  *
* 60                                  March 1, 2000               *
* 61                                                              *
* 62 Original DY Date from the file:  335/2000                    *
* 63                                  12/01/2000                  *
```

```
* 64                                  12/01/2000                  *
* 65                                  December 1, 2000            *
* 66                                                              *
* 67 Original DY Date from the file:  365/2000                    *
* 68                                  12/31/2000                  *
* 69                                  12/31/2000                  *
* 70                                  December 31, 2000           *
* 71                                                              *
* 72 Original String from the file:   01/01/2000                  *
* 73                                  01/01/2000                  *
* 74                                  1/2000                      *
* 75                                  January 1, 2000             *
* 76                                                              *
* 77 Original String from the file:   01/31/2000                  *
* 78                                  01/31/2000                  *
* 79                                  31/2000                     *
* 80                                  January 31, 2000            *
* 81                                                              *
* 82 Original String from the file:   02/01/2000                  *
* 83                                  02/01/2000                  *
* 84                                  32/2000                     *
* 85                                  February 1, 2000            *
* 86                                                              *
* 87 Original String from the file:   02/28/2000                  *
* 88                                  02/28/2000                  *
* 89                                  59/2000                     *
* 90                                  February 28, 2000           *
* 91                                                              *
* 92 Original String from the file:   03/01/2000                  *
* 93                                  03/01/2000                  *
* 94                                  60/2000                     *
* 95                                  March 1, 2000               *
* 96                                                              *
* 97 Original String from the file:   12/01/2000                  *
* 98                                  12/01/2000                  *
* 99                                  335/2000                    *
*100                                  December 1, 2000            *
*101                                                              *
*102 Original String from the file:   12/31/2000                  *
*103                                  12/31/2000                  *
*104                                  365/2000                    *
*105                                  December 31, 2000           *
*106                                                              *
*107 Original English from the file:  January 1, 2000            *
*108                                  01/01/2000                  *
*109                                  1/2000                      *
*110                                  01/01/2000                  *
*111                                                              *
*112 Original English from the file:  January 31, 2000           *
*113                                  01/31/2000                  *
*114                                  31/2000                     *
*115                                  01/31/2000                  *
```

```
*116                                                                          *
*117 Original English from the file:  February 1, 2000                        *
*118                                  02/01/2000                              *
*119                                  32/2000                                 *
*120                                  02/01/2000                              *
*121                                                                          *
*122 Original English from the file:  February 28, 2000                       *
*123                                  02/28/2000                              *
*124                                  59/2000                                 *
*125                                  02/28/2000                              *
*126                                                                          *
*127 Original English from the file:  March 1, 2000                           *
*128                                  03/01/2000                              *
*129                                  60/2000                                 *
*130                                  03/01/2000                              *
*131                                                                          *
*132 Original English from the file:  December 1, 2000                        *
*133                                  12/01/2000                              *
*134                                  335/2000                                *
*135                                  12/01/2000                              *
*136                                                                          *
*137 Original English from the file:  December 31, 2000                       *
*138                                  12/31/2000                              *
*139                                  365/2000                                *
*140                                  12/31/2000                              *
.)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))-
```

Review Questions

1. Assume the following definitions.

```
struct EMPLOYEE {
  long   idNumber;
  char   firstName[21];
  char   lastName[31];
  char   payType;
  double rate;
};

EMPLOYEE emp1;
EMPLOYEE emp2;
```

a. Copy emp1's first name into emp2.

b. Copy emp2's pay type into emp1.

c. Copy “Smith” into emp1's last name.

d. Place an ‘h’ into emp2's pay type.

e. Make emp1's pay rate the same as emp2's pay rate.

f. Write a prototype for an InputEmployee() function whose task is to input a set of employee data from the passed input stream.

g. Write a function to output an employee record to a file.

h. Make emp2 be a copy of emp1.

2. Use the **DATE** variant record defined in this chapter.

a. Create an instance of the date in month, day and year format storing Jan 21, 2002 in it.

b. Create an instance of the date in string format to store February 12, 1995.

c. Create an instance of the date in English format to store 03/04/1998.

d. Create an instance of the date in day-year form to store January 12, 1888.

3. Consider the following tax record. Write the coding sequences to solve the following problems.

```
struct DATE {
  short month;
  short day;
  short year;
};

enum TypeProperty {Home, Automobile, Computer, Appliance,
                   Stocks, Bonds};

enum TypeDepreciation {ThreeYear, FiveYear, TwentyFiveYear};

const double taxRates[3] = {.0789, .055, .000125};

struct ListedProperty {
  DATE             datePlacedIntoService;
  double           originalCost;
  double           depreciationCostThisYear;
  TypeProperty     typeProperty;
```

```
  TypeDepreciation typeDepreciation;
};

ListedProperty item[100];
ListedProperty prop;
```

a. If the i^{th} item is a home, set its depreciation type to 25-year.

b. Copy the i^{th} item into prop.

c. If the i^{th} item is a computer, set its depreciation type to 3-year.

d. If prop's depreciation type is 25-year, then calculate the depreciation cost this year by multiplying the corresponding tax rate by the original cost.

e. Assign type appliance to the i^{th} item's type of property.

f. Write the sequence of statements that would make the 5^{th} element of item be an automobile purchased on March 21, 2000 that originally cost $25,000 and depreciates using the 5 year rule.

g. Write the sequence of statements that would make prop be a computer purchased on 27 June, 1999 that depreciates using the 3-year rule.

Stop! Do These Exercises Before Programming

1. An Acme programmer was given the task to design a phone book structure. He presented the following to the design staff for review. It was not passed by the reviewers because of errors and poor design features.

```
struct NAME {
  char firstName[50];
  char lastName[50];
};
struct ADDRESS {
  char street1[80];
  char street2[80];
  char city[20];
  char state[2];
  char zip[5];
};
struct PHONE {
  short area;   // such as 309
  short prefix; // such as 699
  short number; // such as 9999
```

```
};
struct PHONEBOOK {
  NAME    name;
  ADDRESS addr;
  PHONE   phone;
  char    type; // 0 = normal, 1 = business listing
};
```

Point out the two actual errors in the design. Then, point out the many poor design features. Write an improved definition for PHONEBOOK.

2. Write the function **InputPhoneBookEntry()** that is passed a reference to a **PHONEBOOK** instance and an **istream** reference. The function returns an **istream** reference. On input, all character strings are surrounded by double quote marks, such as "John." The phone number on input appears as (309) 699-9999. Assume that all fields are in the order of occurrence in the PHONEBOOK and other structures.

3. Write the function **LookUpPhoneNumber()** that is passed an array of **PHONEBOOK** objects, the count of the number in the array and a constant reference to a **PHONE** structure to match. The function returns the subscript of the matching entry or –1 if no match occurs.

4. Write a structure definition for a COIN that has four data members. The first member is the count of the number of this coin type. The second is the number of pennies this coin type is worth. The last two members are string pointers, that is, **char*** — one for the English name for one of this coin type and the other for the English plural name of this coin type. For example, if one created an instance of COIN to hold a dime, the count field is 0, the denomination field contains 10. The two **char*** pointers point to the strings "Dime" and "Dimes".

Now create and initialize instances to hold a dime, nickel, and a quarter.

5. Write a structure to hold the information for a bank account. The fields include the account number, the person's name (an array of 60 characters), the current balance, the average daily balance, and the date (stored as three integers). Next write an **InputAccount()** function to input a single bank account record. Assume that the name is surrounded by double quote marks.

Programming Problems

Problem Pgm06-1 The Suspect Matcher Program

Your local law enforcement agency has requested that you write them a suspect matcher program. The program should load a file of suspect information into an array of Suspect structures. Then, the program prompts the user to enter the characteristics of the perpetrator of the crime as described by the witness. The program then displays the suspects that match the perpetrator's characteristics. After waiting for an "Ok to Continue" message, the program prompts for another set of perpetrator characteristics and so on until the end of file is signaled.

The Suspect structure should contain the following fields.

Field	Type	Description
First Name	first name of a person	a 12-character string including the null terminator
Last Name	last name of a person	a 25-character string including the null terminator
Height	height in inches	0 is unknown, 44 to 90 is valid
Eye Color	eye color	0 is unknown, 1 is brown, 2 is blue, 3 is hazel
Hair Color	hair color	0 is unknown, 1 is brown, 2 is black, 3 is red, 4 is gray, 5 blonde
Hat Size	an integer	0 is unknown
Shoe Size	an integer	0 is unknown
Teeth Marks	a characteristic	0 is unknown, 1 is normal, 2 is crooked, 3 is gold filled, 4 is partial, 5 is missing
Facial Scar	a yes/no field	0 is unknown, 1 is yes, 2 is no
Hand Scar	a yes/no field	0 is unknown, 1 is yes, 2 is no
Eye Patch	a yes/no field	0 is unknown, 1 is yes, 2 is no
Bald Patch	a yes/no field	0 is unknown, 1 is yes, 2 is no
Leg Limp	a yes/no field	0 is unknown, 1 is yes, 2 is no
Tattoo	a yes/no field	0 is unknown, 1 is yes, 2 is no

First, make up a test **suspects.txt** data file. The file should contain at least 12 suspect records. One of the records should have all information filled out with none marked as unknown; one should contain all 0's except the first and last name fields. The remainder should contain a

variety of values. Assume that all information in the **suspects.txt** file is correct — there can be no invalid data in this data file.

This is in part a major design problem. You are faced with data entry of a lot of fields that are very similar in nature. You should try to write "generic" or reusable functions to minimize the volume of coding. If you do not, you will be writing volumes of lines of code!

You may use enumerated data types for program maintenance purposes or you may use **#define** values or **const int** values. While the program does not have to check for invalid data in the input file, it will have to detect invalid perpetrator entries being entered and re-prompt the user accordingly.

The matching process is done on all fields except the first and last names. No one characteristic is given more weight than another. If the perpetrator's twelve items match a particular suspect record, then the matching percentage is 100%. If none match, it is 0%. After calculating the matching percentage of the perpetrator to all of the suspects, display first the perpetrator's data followed by the suspects that match with a percentage above 0% in percentage order from highest to lowest. Your output should allow the user to see at a glance which characteristics are matching and which are not. There are many ways this can be done; columnar aligned fields is one way.

Problem Pgm06-2 The TIME Variant Record

Acme Corporation wishes to handle times in a similar manner to the **DATE** variant records discussed in this chapter. That is, times in Acme applications can be stored in several ways.

long total seconds since midnight
three shorts: hours, minutes and seconds
hh.hhhhhh in a double
a string "hh:mm:ss" such as "09:03:06" based on a 24-hour time

All times are 24-hour based beginning at midnight. Thus, the range of values go from 00:00:00 through and including 24:00:00.

Construct a TIME structure to store these variants similar to the DATE structure presented in this chapter. Then create a conversion function called **ConvertTime()** that is passed a reference to a TIME structure instance and a conversion type enum instance that indicates which conversion is to be performed. The conversion enum values should be

TimeToTotalSeconds, TimeToHMS, TimeToFraction, and TimeToString

Also provide a **VerifyTime()** function that is passed three short integers representing the hours, minutes, and seconds. It returns **true** if the time is valid and **false** if it is not valid. This function is intended for the user to call upon inputting a time from the user. Once the data has

been stored in a TIME structure instance, assume that it is a valid time. In other words, when in the **ConvertTime()** function, always assume that the time stored is valid; you do not need to continually re-verify it.

When the **ConvertTime()** function is written, design a testing program to thoroughly test the function. Provide a testing oracle along with the testing program to ensure the function works perfectly in all cases. That is, the testing program should implement your testing oracle.

Problem Pgm06-3 Bank Account Processing

Acme Consolidated Bank wishes a new bank transaction processing program. Their database consists of up to 10,000 accounts. Each record contains the long account number, the current balance, the cumulative daily balance, a count of the number of days in this monthly period that have elapsed and an indicator that defines whether this is a checking (a 1 digit) or savings account (a 0 digit). At the end of each day's processing, that day's current balance is added to the cumulative daily balance field and the count of the number of days is incremented for every bank account. Then, at the end of a monthly period, the cumulative balance is divided by the count of the number of days to find the average daily balance for each account.

The transactions program begins by loading an array with the bank account database called **bankaccount.txt**. Next, the daily transactions file, **trans.txt**, should be processed. The first character of each transaction line contains a C for check or a D for deposit. This is followed by the bank account number and then a monetary amount to be added or subtracted accordingly. If this transaction is a check, then a third number contains the check number.

The following rules apply to each account processed. If the current balance is below $500 and if this transaction is a check, there is a $0.15 service charge to be debited as well. No check can be subtracted if that would lower the balance below 0. If such would occur, the check is bounced. However, if a check bounces, there is an automatic $25.00 service charge debited immediately which could make the balance fall below zero. If this is a deposit, a service charge of $0.10 is applied.

When the end of the transaction file occurs, then add one day to every bank account's count of the number of days that have elapsed. Also add each account's current balance to its cumulative balance member. Finally, the bank account database, **bankaccount.txt**, is re-written to disk. Two reports are generated: the Transactions Log (**TranLog.txt**) and the Accounts Summary (**AcctSum.txt**). These are shown below.

```
Acme Daily Transaction Log

Trans   Account    Current  --Transaction--  Service       New
 Type    Number    Balance    Amount  check   Charge   Balance

Check     12345   $1000.00  $ 100.00 (1234)   $ 0.00  $ 900.00
```

```
Check     23445   $ 200.00  $ 100.00 ( 444)   $ 0.15  $  99.85
Check     45346   $ 100.00  $ 100.00B( 333)   $25.00  $  75.00
Check     56423   $  10.00  $ 100.00B(1211)   $25.00  $ -15.00
Deposit   23454   $2000.00  $ 500.00          $ 0.10  $2499.90


Acme Daily Account Summary Report

Account    Current  Average Daily
 Number    Balance    Balance
  12344   $1234.99   $1500.00
  23434   $ 500.99   $1230.00
```