Program Declarations

```
//*****************************************************
// Friends program
// Program creates an address book by reading first names,
// last names, phone numbers, & birth dates from standard
// input & writing an alphabetical listing to output file
//*****************************************************
#include <iostream.h>
#include <iomanip.h>    // For setw()
#include <fstream.h>    // For file I/O
#include <string.h>     // For strcmp()
#include <ctype.h>      // For toupper()


const int MAX_FRIENDS = 150;    // Max number of friends
typedef char String8[9];        // 8 characters plus '\0'
typedef char String15[16];      // 15 characters plus '\0'
```

Updated code from "*Programming and Problem Solving with C++*", N. Dale, C. Weems & M. Headington, D.C. Heath, ©1996, 794-808.

friends.cpp

Struct & Prototypes

```
struct EntryType
{
    String15 firstName;
    String15 lastName;
    int      areaCode;      // Range 100..999
    String8  phoneNumber;
    int      month;         // Range 1..12
    int      day;           // Range 1..31
    int      year;          // Range 1900..2100
};

void GetEntry( EntryType& );
void GetName( EntryType& );
void GetPhoneNumber( EntryType& );
void Insert( EntryType[], int&, EntryType );
void OpenForOutput( ofstream& );
void SearchOrd(   EntryType[], EntryType, int,
                  int&, bool& );
void WriteEntries( const EntryType[], int, ofstream& );
```

friends.cpp

main( ) Function

```
int main()
{
  EntryType addressBook[MAX_FRIENDS]; //Friends' recs

    int       length = 0; // # entries in addressBook
    EntryType entry;      // Curr rec being entered
    char      response;   // Response char
    ofstream  friendFile; // Output file of entries

    OpenForOutput(friendFile);
    if ( !friendFile )
        return 1;                  // EXIT_FAILURE
    do {
        GetEntry(entry);
        cout << "Is this entry correct? (Y or N) ";
        cin >> response;
        if (toupper(response) == 'Y')
            Insert(addressBook, length, entry);
        cout << "Do you wish to continue? (Y or N) ";
        cin >> response;
        cin.ignore(100, '\n');

 // Invariant:
            //     addressBook[0..length-1] entries
            //     as input from the user
            //  && 0 <= length <= MAX_FRIENDS
    } while (toupper(response)=='Y' && length<MAX_FRIENDS);

    if (length == MAX_FRIENDS)
        cout << "Address book is full." << endl;
    WriteEntries(addressBook, length, friendFile);
    return 0; //EXIT_SUCCESS
} // end main()
```

friends.cpp

Open Output File

```
void OpenForOutput( /* inout */ ofstream& someFile )
// File to be opened

// Prompts the user for the name of an output file
// and attempts to open the file

// Postcondition:
//     The user has been prompted for a file name
//  && IF the file could not be opened
//         An error message has been printed
// Note:
//     Upon return from function, the caller must test
//     stream state check if file was successfully opened

{
    // User-specified file name (max. 50 chars)
    char fileName[51];  //const int FILENAMEMAX = 51 ;

    cout << "Output file name: ";
    cin.get(fileName, 51);
    cin.ignore(100, '\n');

    someFile.open(fileName);
    if ( !someFile )
        cout << "** Can't open " << fileName
             << " **" << endl;
}
```

friends.cpp

Record Input

```
void GetEntry( /* out */ EntryType& entry )
// Struct being built

// Builds and returns a complete address book entry

// Postcondition:
//     User has been prompted for a friend’s first name,
//     last name, phone number, and birth date
//  && The input values are stored in the corresponding
//     members of entry

{
    GetName(entry);
    GetPhoneNumber(entry);
    cout << "Enter birth date as three integers,”
         << “ separated by"
         << " spaces: MM DD YYYY" << endl;
    cin >> entry.month >> entry.day >> entry.year;
}
```

friends.cpp

Name Input

```
void GetName( /* out */ EntryType& entry )
// Struct receiving name

// Inputs friend's first and last name,
// storing them into entry

// Postcondition:
//     User has been prompted for a friend's first name
//     and last name
//  && entry.firstName == the input string for first name
//  && entry.lastName == the input string for last name

{
    cout << "Enter person's first name." << endl;
    cin.get(entry.firstName, 16);
    cin.ignore(100, '\n');

    cout << "Enter person's last name." << endl;
    cin.get(entry.lastName, 16);
    cin.ignore(100, '\n');
}
```

friends.cpp

Phone Number Input

```
void GetPhoneNumber( /* inout */ EntryType& entry )
// Struct receiving number

// Inputs area code and phone number,
// storing them into entry

// Postcondition:
//     User prompted for the area code and phone number
//  && entry.areaCode == input integer for area code
//  && entry.phoneNumber == input string for phone number

{
    cout << "Enter area code, blank, and the number"
         << " (including '-')." << endl;
    cin >> entry.areaCode;
    cin.ignore(1, ' ');   // Consume blank
    cin.get(entry.phoneNumber, 9);
    cin.ignore(100, '\n');
}
```

friends.cpp

Array Entry Storage

```
void Insert( /*inout*/ EntryType list[], // List to change
             /*inout*/ int&      length, // List Length
             /*in*/    EntryType item   )// Insert Item
// Inserts item into its proper place in the sorted list
// Precondition:
//     length < MAX_FRIENDS
//  && list[0..length-1] are in ascending order
//  && item is assigned
// Postcondition:
//     item is in list
//  && length == length@entry + 1
//  && list[0..length-1] are in ascending order
//  && IF item was already in list@entry
//     item is inserted before the one that was there
{
    bool    placeFound;    // True if item already in list
    int     index;         // Position where item belongs
    int     count;         // Loop control variable

 SearchOrd(list, item, length, index, placeFound);

 // Shift list[index..length-1] down one
 for (count = length - 1; count >= index; count--)
 // Invariant (prior to test):
 //     list[length-1..count+1] have been shifted down
 //  && length - 1 >= count >= index - 1
      list[count+1] = list[count];

    list[index] = item; // Insert item

    length++; // Increment length of list
}
```

friends.cpp

Item Search

```
void SearchOrd(
   /* inout */ EntryType list[],  // List to search
   /* in */    EntryType item,    // Item to be found
   /* in */    int       length,  // Length of list
   /* out */   int&      index,   // Item locn if found
   /* out */   bool&  found  ) // True if item found

// Searches list for item, returning index if item found.
// If item not found, SearchOrd returns index where
// item belongs

// Precondition:
//     length < MAX_FRIENDS
//  && list[0..length-1] are in ascending order
//  && item is assigned
// Postcondition:
//     list is the same as list@entry except that
//     list[length] is
//       overwritten to aid in the search
//  && IF item is in list@entry
//         found == true  &&  list[index] contains item
//     ELSE
//         found == false  &&  index is where item belongs
```

friends.cpp

Item Search (continued)

```
// void SearchOrd continued

    index = 0;

    // Store item at position beyond end of list

    list[length] = item;

    // Exit loop when item is found, perhaps as sentinel

    while (strcmp(item.lastName,list[index].lastName) > 0)

            // Invariant (prior to test):
            //     item is not in list[0..index-1]

        index++;

    // Determine whether item was found prior to sentinel

    found = (index < length &&
             strcmp(item.lastName,list[index].lastName)
                   == 0);
}
```

friends.cpp

Write Output File

```
void WriteEntries(
  /* in */ const EntryType addressBook[], // Entries Array
  /* in */       int       length,        // Num entries
  /* inout */    ofstream& friendFile   ) // File for list

// Writes all entries to the file friendFile
// Precondition:
//     length <= MAX_FRIENDS
//  && addressBook[0..length-1] are assigned
// Postcondition:
//     addressBook[0..length-1] output to friendFile
{
   int counter;        // Loop counter

   for (counter = 0; counter < length; counter++) {
           // Invariant (prior to test):
           //     addressBook[0..counter-1] output
           //  && 0 <= counter <= length
      friendFile<<addressBook[counter].firstName <<' '
                <<addressBook[counter].lastName <<endl;
      friendFile<<'(' <<addressBook[counter].areaCode <<
                << ") " <<addressBook[counter].phoneNumber
                << endl;
      friendFile<< setw(2) << addressBook[counter].month
                << '/' << setw(2)
                << addressBook[counter].day << '/'
                << setw(4) << addressBook[counter].year
                << endl;
      friendFile<< endl;
    }
}
```

friends.cpp

