

Array of Structs Example

```
// Array of struct version of
// Project 2 for CS 1044 Summer I 2001
//
// Programmer:    William D McQuain
// OS:           Windows NT Workstation 4.0
// System:       Pentium II 400, 256 MB Memory
// Compiler:     Visual C++ 6.0, Service Pack 4
// Last modified: June 19, 2001
//
// Purpose
// This program reads a list of trip data records, including
// origin, destination, distance and time.  For each trip the
// program computes the time in minutes and average MPH.  The
// overall time for all trips and overall average MPH are also
// computed.
//
// The program then writes a summary of its findings to an output
// file.
//
#include <iostream>    // for cout
#include <fstream>     // for file streams
#include <iomanip>     // for formatting manipulators
#include <string>      // for string variables
#include <climits>    // for INT_MAX
using namespace std; // to put all of the above things in scope

const int MAXTRIPS    = 50; // Maximum number of trips to process
const int MINPERHOUR  = 60; // Number of minutes in an hour
```

```
// Declaration of struct data type to hold the information about one trip:
```

```
struct Trip {  
    string Origin;           // trip origin  
    string Destination;     // trip destination  
    int    Mileage;         // trip length, in miles  
    int    Time;           // trip duration, in minutes  
    double avgMPH;         // average speed, in MPH  
};
```

Type declarations are almost always global.

```
// Function prototypes:
```

```
//
```

```
void readTrips(Trip tripList[], int& numTrips);  
void readOneTrip(ifstream& In, Trip& aTrip);  
  
int convertHHMMtoMin(int tripHours, int tripMinutes);  
void calcMPH(Trip tripList[], int numTrips);  
double calcOneMPH(int Miles, int Minutes);  
void calcSummaryData(const Trip tripList[], int numTrips,  
                    int& totalMiles, int& totalTime);  
void sortTripsByMileage(Trip tripList[], int numTrips);  
  
void printIdentification(ofstream& Out);  
void printTableHeader(ofstream& Out);  
void printTripData(ofstream& Out, const Trip tripList[], int numTrips);  
void printTableEnd(ofstream& Out);  
void printSummary(ofstream& Out, int numTrips, int totalMiles, int totalHours,  
                int totalMinutes, double overallMPH);
```

If you compare this to the function version, many of the parameter lists have gotten dramatically shorter because of the "clumping" of data values inside Trip variables.

```

int main() {

    const string resultsFileName = "Summary.txt"; // Output file name.

    ofstream Out(resultsFileName.c_str()); // Attach a stream to the output
                                           // file.
    Out << fixed << showpoint; // Prepare output stream for
                               // decimal output.

    Trip tripList[MAXTRIPS]; // Array to hold trip data

    int numTrips = 0; // Number of trips reported.
    int totalMiles = 0, // Total length of all trips, in miles.
        totalTime = 0; // Total time for all trips, in minutes.

    // Read the data for all the trips:
    readTrips(tripList, numTrips);

    // Check to see if there were any trips; otherwise we'll wind up
    // dividing by zero later on.
    if ( numTrips == 0 ) {
        Out << "No trips were reported." << endl;
        return 0; // No trips, so stop here.
    }

    // Calculate the average MPH for all the trips:
    calcMPH(tripList, numTrips);

    // Update the totals over all trips:
    calcSummaryData(tripList, numTrips, totalMiles, totalTime);
}

```

Here, I grab all the data from the input file.

The overall organization of `main()` has changed dramatically. Now we can separate acquiring data from performing calculations from writing output.

Most of the calculations are performed in these two functions.

```

// Sort the trips by mileage:
sortTripsByMileage(tripList, numTrips);

// Calculate the overall average MPH for all trips:
double overallMPH = calcOneMPH(totalMiles, totalTime);

// Calculate total hours and residual minutes for all trips:
int totalHours,
    totalMinutes;
totalHours    = totalTime / MINPERHOUR;
totalMinutes  = totalTime % MINPERHOUR;

// Write the header data to the output file:
printIdentification(Out);
printTableHeader(Out);

// Print the results for the trips:
printTripData(Out, tripList, numTrips);

// Write a delimiter for the bottom of the output table:
printTableEnd(Out);

// Print the summary information for all trips:
printSummary(Out, numTrips, totalMiles, totalHours,
             totalMinutes, overallMPH);

// Close the output files:
Out.close();
// Terminate a successful execution:
return 0;
}

```

This function sorts the trip records... we'll talk about the details of that later.

Some of the calculations aren't worth moving to separate functions.

All of the output is produced by these function calls, after all the calculations have been finished. The separation of input from processing from output makes the program more understandable.

```

//////////////////////////////////// readTrips
// Reads the data for a all the trips from the input file.
//
// Parameters:
//   tripList[]   array of Trip variables to hold data
//   numTrips     number of trips read
//
// Pre:          Input data is in a file named TripData.txt.
//              tripList[] has dimension MAXTRIPS, a global.
// Post:         All the trip data has been read into the cells of tripList[].
//              numTrips equals the number of trips read.
//
// Returns:     None
//
// Called by:   main()
// Calls:      readOneTrip()
//
void readTrips(Trip tripList[], int& numTrips) {

    const string dataFileName = "TripData.txt"; // Input file name.
    ifstream In(dataFileName.c_str()); // Attach a stream to the input file.

    // If the input file does not exist, this will detect that.
    // We handle that by printing an error message and stopping the program.
    if ( In.fail() ) {
        cout << "Data file not found: " // Write an error message...
             << dataFileName // including the file name.
             << endl // Bang "return"
             << "Exiting now..." << endl; // Finish the message.
        exit(1); // Terminate a failed execution.
    }
}

```

The input file isn't needed outside of this function and the one it calls, so it's opened and closed locally.

```

Trip tempTrip;
numTrips = 0;

In.ignore(INT_MAX, '\n'); // Skip over the two header lines in the
In.ignore(INT_MAX, '\n'); //   input file.

readOneTrip(In, tempTrip);

while ( In && (numTrips < MAXTRIPS) ) { // As long as you got new data,
                                        //   continue...

    tripList[numTrips] = tempTrip; // Store this trip in the list
    numTrips++; // Count this trip.

    // Try to read the data for another trip:
    readOneTrip(In, tempTrip);
}

In.close();
}

```

Pay careful attention to the input logic here.

The function `readOneTrip()` handles reading the data for one trip from the input file. That data is stored in a local variable, NOT directly into the array `tripList`.

If we read data directly into the array we'd risk corrupting the first unused cell when we reach the end of the input.

```

//////////////////////////////////// readOneTrip
// Reads the data for a single trip from the input file.
// Parameters:
//   In      input stream
//   aTrip   Trip variable to hold trip data
// Pre:     Input stream has been opened.
//          Input point is at beginning of a line of trip data.
//          Trip data line is formatted according to specification.
// Post:    Trip data line has been read into aTrip.
//          Input point is at beginning of next input line.
// Returns: None
// Called by: readTrips()
// Calls:    None
//
void readOneTrip(ifstream& In, Trip& aTrip) {

    int tripHours, tripMinutes;

    getline(In, aTrip.Origin, '\t'); // Read: the name of the trip origin.
    getline(In, aTrip.Destination, '\t'); // the name of the trip dest.
    In >> aTrip.Mileage; // length of trip in miles
    In >> tripHours; // hours field for trip time
    In.ignore(1, ':'); // colon separator
    In >> tripMinutes; // minutes field for trip time
    // Calculate and store the trip time
    // in minutes:
    aTrip.Time = convertHHMMtoMin(tripHours, tripMinutes);

    In.ignore(INT_MAX, '\n'); // Skip to beginning of next input line.
    return;
}

```

Note that the Trip parameter must be passed by reference (with a '&') in order to get the data that's read back to the calling function.

```

////////////////////////////////////// convertHHMMtoMin
// Converts a time value in HH:MM format to equivalent total minutes.
// Parameters:
//   tripHours      HH field of time
//   tripMinutes    MM field of time
// Pre:            HH and MM have been initialized.
//                HH and MM are non-negative.
//                MM is in the range [0, 59)
//                The constant MINPERHOUR is in global scope.
// Post:          The equivalent time in minutes has been computed.
// Returns:       Equivalent time in minutes:  MINPERHOUR * HH + MM
// Called by:    readOneTrip()
// Calls:        None
//
int convertHHMMtoMin(int tripHours, int tripMinutes) {

    return (MINPERHOUR * tripHours + tripMinutes);
}

```

```
////////////////////////////////////// calcMPH
// Calculates the average speed, in MPH, for each trip.
// Parameters:
//   tripList[]   array of Trip variables to hold data
//   numTrips     number of trips read
// Pre:           tripList[] stores data for numTrips trips.
//               The constant MINPERHOUR is in global scope.
// Post:          The average MPH has been calculated and stored for each
//               trip in tripList[].
// Returns:      None
// Called by:    main()
// Calls:        None
//
void calcMPH(Trip tripList[], int numTrips) {
```

This function takes care of calculating the average MPH for every trip, not just for one.

```
    int Idx;
    for (Idx = 0; Idx < numTrips; Idx++) {

        tripList[Idx].avgMPH =
            calcOneMPH(tripList[Idx].Mileage, tripList[Idx].Time);
    }
}
```

There's not much reason to have the "helper" function calcOneMPH() instead of just putting the calculation inline here.

It does make it possible to illustrate passing an array element (or a field of one) as a parameter.

Since both those parameters are of type int, they're passed by value by default.

```

//////////////////////////////////// calcOneMPH
// Calculates the average speed, in MPH, for one trip.
// Parameters:
//   Miles      trip mileage
//   Minutes    trip time, in minutes
// Pre:        Miles and Minutes have been initialized.
//            Miles >= 0 and Minutes > 0.
//            The constant MINPERHOUR is in global scope.
// Post:       The average MPH has been calculated and returned.
// Returns:    Avg speed in MPH: Miles / (Minutes / MINPERHOUR)
// Called by: calcMPH()
// Calls:      None
//
double calcOneMPH(int Miles, int Minutes) {

    return (MINPERHOUR * double(Miles) / double(Minutes));
}

```

```

//////////////////////////////////// calcSummaryData
// Calculates the total mileage and total time for all the trips.
// Parameters:
//   tripList[]   array of Trip variables to hold data
//   numTrips     number of trips stored in tripList[]
//   totalMiles   total mileage of all the trips in tripList[]
//   totalTime    total time (in minutes) of all the trips in tripList[]
// Pre:          tripList[] stores data for numTrips trips.
// Post:         totalMiles stores the total mileage of all the trips
//              totalTime stores the total time (in minutes) of all the trips
// Returns:      None
// Called by:   main()
// Calls:       None
//
void calcSummaryData(const Trip tripList[], int numTrips,
                    int& totalMiles, int& totalTime) {

    int Idx;
    totalMiles = 0; // These are probably initialized to zero by the caller,
    totalTime  = 0; // but it doesn't hurt to make sure.

    for (Idx = 0; Idx < numTrips; Idx++) {
        totalMiles += tripList[Idx].Mileage;
        totalTime  += tripList[Idx].Time;
    }
}

```

```

////////////////////////////////////////// sortTripsByMileage
// Sorts the trips in descending order, by mileage, using Bubble Sort.
// Parameters:
//   tripList[]   array of Trip variables to hold data
//   numTrips     number of trips stored in tripList[]
// Pre:          tripList[] stores data for numTrips trips.
// Post:         tripList[] has been sorted.
// Returns:      None
// Called by:    main()
// Calls:        None
//
void   sortTripsByMileage(Trip tripList[], int numTrips) {

    int   Stop, Check;
    Trip  tempTrip;

    for (Stop = numTrips - 1; Stop > 0; Stop--) {

        for (Check = 0; Check < Stop; Check++) {

            if ( tripList[Check].Mileage < tripList[Check + 1].Mileage ) {
                tempTrip           = tripList[Check];
                tripList[Check]    = tripList[Check + 1];
                tripList[Check + 1] = tempTrip;
            }
        }
    }
}

```

```
//////////////////////////////////// printIdentification
// Writes the programmer name and course information to the output file.
// Parameters:
//   Out      output stream to which information is written
// Pre:      The output stream has been opened.
//          Nothing has been written yet.
// Post:     The specified information has been written to the output stream.
// Returns:  None
// Called by: main()
// Calls:    None
//
void printIdentification(ofstream& Out) {

    // Print identification information to output file:
    Out << "Programmer: " << "Bill McQuain" << endl;
    Out << "CS 1044 Summer I 2001" << endl;
    Out << endl;

    return;
}
```

```

//////////////////////////////////// printTableHeader
// Writes the column headers and top delimiter to the output file.
// Parameters:
//   Out      output stream to which information is written
// Pre:      The output stream has been opened.
//          The programmer identification data has been written.
// Post:     The specified information has been written to the output stream.
// Returns:  None
// Called by: main()
// Calls:    None
//
void printTableHeader(ofstream& Out) {

    // Print output column headers to output file:
    Out << "Origin           Destination           Mileage   Minutes   MPH"
        << endl;
    Out << "-----"
        << endl;
    return;
}

```

```


//////////////////////////////////// printTripData
// Writes the data for all of the trips to the output file.
// Parameters:
//   Out          output stream
//   tripList[]  array of Trip variables to hold data
//   numTrips    number of trips stored in tripList[]
// Pre:          Output stream has been opened.
//              Identification and table headers have been written.
//              Output stream is at the beginning of a line.
//              tripList[] contains numTrips records.
// Post:         The specified data has been written to the output file.
// Returns:     None
// Called by:   main()
// Calls:       None
//
void printTripData(ofstream& Out, const Trip tripList[], int numTrips) {

    int Idx;

    for (Idx = 0; Idx < numTrips; Idx++) {
        Out << tripList[Idx].Origin;
        Out << setw(20 - tripList[Idx].Origin.length()) << ' ';
        Out << tripList[Idx].Destination;
        Out << setw(27 - tripList[Idx].Destination.length())
            << tripList[Idx].Mileage
            << setw(10) << tripList[Idx].Time
            << setw(8) << setprecision(1) << tripList[Idx].avgMPH
            << endl;
    }
    return;
}

```

Note the syntax here...



```

//////////////////////////////////// printTableEnd
// Writes the bottom delimiter and a blank line to the output file.
// Parameters:
//   Out      output stream to which information is written
// Pre:      The output stream has been opened.
//          The table of trip data has been written.
// Post:     The specified information has been written to the output stream.
// Returns:  None
// Called by: main()
// Calls:    None
//
void printTableEnd(ofstream& Out) {

    Out << "-----"
        << endl;
    Out << endl;
}

```

```

//////////////////////////////////// printSummary
// Writes a summary of statistics for all the trips.
// Parameters:
//   Out          output stream
//   numTrips     number of trips for which data was read
//   totalMiles   total mileage for all the trips
//   totalHours   HH field of total time for all the trips
//   totalMinutes MM field of total time for all the trips
//   overallMPH   average speed over all the trips
// Pre:          The output stream has been opened.
//              The identification, table headers, table data and table
//              bottom have been written.
//              The numeric parameters have been initialized.
//              The numeric parameters are >= 0, MM is in the range [0, 59).
// Post:        The specified data and labels have been written.
// Returns:     None
// Called by:   main()
// Calls:       None
//
void printSummary(ofstream& Out, int numTrips, int totalMiles, int totalHours,
                 int totalMinutes, double overallMPH) {

    Out << "Number of trips: " << setw(8) << numTrips << endl;
    Out << "Total miles:      " << setw(8) << totalMiles << endl;

    // Print the total time and overall average MPH for all trips:
    Out << "Total time:      " << setw(5) << totalHours << ":";
    Out << setw(2) << setfill('0') << totalMinutes << setfill(' ') << endl;
    Out << "Average MPH:      " << setw(8) << setprecision(2) << overallMPH
        << endl;
}

```