```
1   // Title
2   //
3   // A program to demonstrate the management of a small database of operational
4   // amplifiers.
5   //
6   // General description
7   //
8   // The database contains up to DATABASE_MAX operational amplifier elements. Each
9   // element contains the operation amplifier name, the number of pins in the package
10  // and stores the slew rate of the device.
11  //
12  // New elements can be added into the database by the user. The database can be saved
13  // to disk or loaded from disk. The database elements can be sorted either by name or
14  // by slew rate. There is also the facility to display the elements.
15  //
16  // Only a single database is required and the file name is fixed in the code (as
17  // DATABASE_FILENAME). This means that each time the database is saved to disk,
18  // any previous data in the file is overwritten. Also, when a database is loaded
19  // from a file it should overwrite any data already in memory.
20
21  #include <iostream>
22  #include <fstream>
23  #include <string.h>
24  #include <algorithm> //std:: sort
25  using namespace std;
26
27  // Class containing OpAmp parameters
28  // Provides functions that access the private members and overloaded the stream
    operators
29  class OpAmps
30  {
31  private:
32      char Name[20];              // the name of the op - amp (e.g. "741")
33      unsigned int PinCount;      // the number of pins in the package
34      double SlewRate;            // the slew rate in volts per microsecond
35
36  public:
37      OpAmps();                   // constructor
38      ~OpAmps();                  // destructor
39
40      void SetOpAmpValues();      // setting OpAmp parameters function
41      void DisplayOpAmpValues();  // displaying op-amps
42
43      string GetNameOpAmp();      // provides access to private Name
44      int GetPinCountOpAmp();     // provides access to private PinCount
45      double GetSlewRateOpAmp();  // provides access to private SlewRate
46
47      friend ostream &operator << (ostream &, OpAmps &);   // overloaded output
        operator function
48      friend ifstream &operator >> (ifstream &, OpAmps &); // overloaded input
        operator function
49  };
50
51  //Constructor and destructor functions
52  OpAmps::OpAmps() // constructor definition of class-OpAmps with initialised values
53  {
54      Name[0] = '\0'; // the name of the op - amp (e.g. "741")
55      PinCount = 0;   // the number of pins in the package
56      SlewRate = 0;   // the slew rate in volts per microsecond
57  }
58
59  OpAmps::~OpAmps() // destructor function definition
60  {
61      cout << ".Goodbye." << endl;
62  }
63
64  // Structure for ArrayOfOpAmps-objects
65  void OpAmps::SetOpAmpValues() //Input user data to alter database length by adding
    an ArrayOfOpAmps-object
66  {
67      cout << "Add new data" << endl;
68      cout << "------------" << endl;
69      cout << "Enter op-amp name: ";
```

```
70      cin >> Name;
71      cout << "Enter number of pins: ";
72      cin >> PinCount;
73      cout << "Enter slew rate: ";
74      cin >> SlewRate;
75      cout << endl;
76  }
77
78  void OpAmps::DisplayOpAmpValues() // Display current ArrayOfOpAmps-objects in the
    database and their parameters
79  {
80      // display a title
81      cout << endl;
82      cout << "Name   Number of pins  Slew rate" << endl;
83
84      // display the elements
85      cout << Name << "       ";
86      cout << PinCount << "      ";
87      cout << SlewRate << " ";
88      cout << endl;
89  }
90
91  // Fucntions for private member access
92  string OpAmps::GetNameOpAmp() // function to access private members of class-OpAmps
93  {
94      return Name;
95  }
96
97  int OpAmps::GetPinCountOpAmp() // function to access private members of class-OpAmps
98  {
99      return PinCount;
100 }
101
102 double OpAmps::GetSlewRateOpAmp() // function to access private members of
    class-OpAmps
103 {
104     return SlewRate;
105 }
106
107 // Stream operator overloading
108 // Currently these two functions are defined as friend functions which could be
    undesirable
109 // due to breaking the natural encapsulation process , however since it is
    implemented for overloading,
110 // it should not provide anything threatening to the program.
111 ostream &operator<<(ostream& outstream, OpAmps& ArrayOfOpAmps) // output overloaded
    function definition
112 {
113     outstream << ArrayOfOpAmps.Name << endl;
114     outstream << ArrayOfOpAmps.PinCount << endl;
115     outstream << ArrayOfOpAmps.SlewRate << endl << endl;
116     return outstream;
117 }
118
119 ifstream &operator >>(ifstream& instream, OpAmps& ArrayOfOpAmps) // input overloaded
    function definition
120 {
121     instream >> ArrayOfOpAmps.Name;
122     instream >> ArrayOfOpAmps.PinCount;
123     instream >> ArrayOfOpAmps.SlewRate;
124     return instream;
125 }
126
127 // Class containing a pointer to OpAmp object,
128 // also contains functions needed to operate the console.
129 // Sort functions were not succesfully implemented, therefore are commented out
130 class OpAmpDatabase
131 {
132 private:
133     OpAmps *ArrayOfOpAmps;  // Creating a pointer to an op amp object
134     unsigned long database_length;
135
136     //member function prototypes
```

```
137   public:
138       OpAmpDatabase(OpAmps*); // Constructor function initialised
139       ~OpAmpDatabase();       // Destructor
140       void Enter();
141       void Display();
142       void Save();
143       void Load();
144       //  void Sort();
145       //  int SortSlewRate(const void *First, const void* Second);
146       //  int SortName(const void *First, const void* Second);
147   };
148
149   //Construct and destructor functions of the OpAmpDatabase-class
150   OpAmpDatabase::OpAmpDatabase(OpAmps* DatabasePointer)
151   {
152       ArrayOfOpAmps = DatabasePointer;
153       database_length = 0;
154   }
155
156   OpAmpDatabase::~OpAmpDatabase()
157   {
158       cout << ".Goodbye." << endl;
159   }
160
161   // the length of the fixed array to be used for database - must be at least one
162   // and no greater the maximum value allowed in an unsigned long (see the file
163   // limits.h)
164   #define DATABASE_MAX 10
165
166   // file used for the database
167   #define DATABASE_FILENAME "database.txt"
168
169   // Control the entering, saving, loading, sorting and displaying of elements in
170   // the database
171   // Arguments: None
172   // Returns: 0 on completion
173   int main()
174   {
175       OpAmps OpAmpArray[DATABASE_MAX];        // The database being created in a
      format of the array of op amp objects.
176       OpAmpDatabase TheDatabase(OpAmpArray);  // Creates an array of the database
177
178       char UserInput;
179
180       // loop until the user wishes to exit
181       while (1)
182       {
183           // show the menu of options
184           cout << endl;
185           cout << "Op-amp database menu" << endl;
186           cout << "--------------------" << endl;
187           cout << "1. Enter a new op-amp into the database" << endl;
188           cout << "2. Save the database to disk" << endl;
189           cout << "3. Load the database from disk" << endl;
190           cout << "4. Sort the database" << endl;
191           cout << "5. Display the database" << endl;
192           cout << "6. Exit from the program" << endl << endl;
193
194           // get the user's choice
195           cout << "Enter your option: ";
196           cin >> UserInput;
197           cout << endl;
198
199           // act on the user's input
200           switch (UserInput)
201           {
202           case '1':
203               TheDatabase.Enter();
204               break;
205
206           case '2':
207               TheDatabase.Save();
208               break;
```

```
209
210             case '3':
211                 TheDatabase.Load();
212                 break;
213
214             case '4':
215                 //TheDatabase.Sort();
216                 break;
217
218             case '5':
219                 TheDatabase.Display();
220                 break;
221
222             case '6':
223                 return 0;
224
225             default:
226                 cout << "Invalid entry" << endl << endl;
227                 break;
228             }
229         }
230     }
231
232     // Allow the user to enter a new element into the database. Note that the data
233     // is simply added to the end the database (if not full) and no sorting is
234     // carried out.
235     // Arguments:
236     //   (1) the element in the database to be entered
237     //   (2) the position of the element in the database
238     // Returns: void
239     void OpAmpDatabase::Enter()
240     {
241         // if the database is full, inform the user
242         if (database_length == DATABASE_MAX)
243         {
244             cout << "The database is full" << endl;
245         }
246
247         // if the database is not full, get the data from the user and alter the database
248         // length
249         else
250         {
251             ArrayOfOpAmps[database_length].SetOpAmpValues();
252             database_length++;
253         }
254     }
255
256     // Save the database to the file specified by DATABASE_FILENAME. If the file
257     // exists it is simply overwritten without asking the user
258     // Arguments:
259     //   (1) the database
260     //   (2) the length of the database
261     // Returns: void
262     void OpAmpDatabase::Save()
263     {
264         fstream outstream;  // file stream for output
265
266         outstream.open(DATABASE_FILENAME, ios::out); // open the file
267
268         outstream << database_length << endl << endl;
269         for (unsigned long i = 0; i < database_length; i++)
270         {
271             outstream << ArrayOfOpAmps[i];
272         }
273
274         outstream.close();
275     }
276
277     // Load the database from the file specified by DATABASE_FILENAME. If the file
278     // exists it simply overwrites the data currently in memory without asking
279     // the user
280     // Arguments:
281     //   (1) the database
```

```
//   (2) the length of the database
// Returns: void
void OpAmpDatabase::Load()
{
    ifstream instream;  // file stream for input

    instream.open(DATABASE_FILENAME, ios::in);   // open the file

    instream >> database_length;
    for (unsigned long i = 0; i < database_length; i++)
    {
        instream >> ArrayOfOpAmps[i];
    }

    // close the file
    instream.close();
}

// //Sort the database either using the name of the op-amps or using the slew rate
// //values
// //Arguments:
// //  (1) the database
// //  (2) the length of the database
// //Returns: void
//void OpAmpDatabase::Sort()
//{
//  char UserInput;
//
//  // show the menu of options
//  cout << endl;
//  cout << "Sorting options" << endl;
//  cout << "---------------" << endl;
//  cout << "1. To sort by name" << endl;
//  cout << "2. To sort by slew rate" << endl;
//  cout << "3. No sorting" << endl << endl;
//
//  // get the user's choice of sorting operation required
//  cout << "Enter your option: ";
//  cin >> UserInput;
//  cout << endl;
//
//  // act on the user's input
//  switch (UserInput)
//  {
//  case '1':
//      // sort according to name (in alphabetical order)
//      std::sort(ArrayOfOpAmps, (ArrayOfOpAmps + database_length), SortName);
//      break;
//
//  case '2':
//      // sort according to slew rate (in increasing slew rate order)
//      std::sort(ArrayOfOpAmps, (ArrayOfOpAmps + database_length), SortSlewRate);
//      break;
//
//  case '3':
//      return;
//
//  default:
//      cout << "Invalid entry" << endl << endl;
//      break;
//  }
//}

//// Compare function for SORT (C++), to help sort the elements by the Name member of
//// OpAmps.
//// Items should be sorted in alphabetical order.
//// Arguments:
////   (1) a database item
////   (2) a database item
//// Returns: result of the comparison
//int OpAmpDatabase::SortName(const void *First, const void* Second)
//{
//  return strcmp(((OpAmps *)First)->GetNameOpAmp, ((OpAmps *)Second)->GetNameOpAmp);
```

```
355   //}
356
357   //// Compare function for SORT (C++), to help sort the elements by the SlewRate
      member
358   //// of OpAmps.
359   //// Items should be sorted in increasing value of slew rate.
360   //// Arguments:
361   ////    (1) a database item
362   ////    (2) a database item
363   //// Returns: result of the comparison
364   //int OpAmpDatabase::SortSlewRate(const void *First, const void* Second)
365   //{
366   //  return (double)((((OpAmps *)First)->GetSlewRateOpAmp > ((OpAmps
      *)Second)->GetSlewRateOpAmp) ? 1 : -1);
367   //}
368
369
370   // Display all of the messages in the database.
371   // Arguments:
372   //    (1) the database
373   //    (2) the length of the database
374   // Returns: void
375
376   void OpAmpDatabase::Display()
377   {
378       // if the database is empty, display an error statement
379       if (database_length == 0)
380       {
381           cout << "No elements in the database" << endl;
382       }
383
384       // if the database is not empty, display all the elements in the database
385       else
386       {
387           cout << endl;
388           for (unsigned long i = 0; i < database_length; i++)
389           {
390               ArrayOfOpAmps[i].DisplayOpAmpValues(); //display current op amps
                  contained in the database
391           }
392       }
393   }
```