```cpp
// A program to demonstrate the management of a small database of operational
// amplifiers.
//
// General description
//
// The database contains up to DATABASE_MAX operational amplifier elements. Each
// element contains the operation amplifier name, the number of pins in the package
// and stores the slew rate of the device.
//
// New elements can be added into the database by the user. The database can be saved
// to disk or loaded from disk. The database elements can be sorted either by name or
// by slew rate. There is also the facility to display the elements.
//
// Only a single database is required and the file name is fixed in the code (as
// DATABASE_FILENAME). This means that each time the database is saved to disk,
// any previous data in the file is overwritten. Also, when a database is loaded
// from a file it should overwrite any data already in memory.

#include <iostream>
#include <fstream>
#include <string.h>
using namespace std;

// the format of each of the elements in the database
struct OpAmps {
    char Name[20];  // the name of the op-amp (e.g. "741")
    unsigned int PinCount;  // the number of pins in the package
    double SlewRate;  // the slew rate in volts per microsecond
};

// the length of the fixed array to be used for database - must be at least one
// and no greater the maximum value allowed in an unsigned long (see the file
// limits.h).
#define DATABASE_MAX 10

// file used for the database
#define DATABASE_FILENAME "database.txt"

// function prototypes
/////////////////////////<enter code here>

void Enter(OpAmps &OpAmpVal, unsigned long &database_length);

void Save(OpAmps *Savetofile, unsigned long &database_length);

void Load(OpAmps *Loadfromfile, unsigned long &database_length);

void Sort(OpAmps *SortDB, unsigned long &database_length);

void Display(OpAmps *DisplayDB, unsigned long &database_length);

int SortByName(const void*a, const void* b);

int SortBySlewRate(const void*a, const void* b);

// Control the entering, saving, loading, sorting and displaying of elements in the
// database.
// Arguments: None
// Returns: 0 on completion
int main()
{
    OpAmps OpAmp[DATABASE_MAX];   // the database
    unsigned long database_length = 0;  // the number of elements in the database
    char UserInput;

    // loop until the user wishes to exit
    while (1) {

        // show the menu of options
        cout << endl;
        cout << "Op-amp database menu" << endl;
        cout << "--------------------" << endl;
        cout << "1. Enter a new op-amp into the database" << endl;
        cout << "2. Save the database to disk" << endl;
        cout << "3. Load the database from disk" << endl;
        cout << "4. Sort the database" << endl;
        cout << "5. Display the database" << endl;
        cout << "6. Exit from the program" << endl << endl;

        // get the user's choice
        cout << "Enter your option: ";
        cin >> UserInput;
        cout << endl;

        // act on the user's input
        switch (UserInput) {
        case '1':
            Enter(OpAmp[database_length], database_length);
            break;

        case '2':
            Save(OpAmp, database_length);
            break;

        case '3':
            Load(OpAmp, database_length);
            break;

        case '4':
            Sort(OpAmp, database_length);
            break;

        case '5':
            Display(OpAmp, database_length);
            break;

        case '6':
            return 0;

        default:
            cout << "Invalid entry" << endl << endl;
            break;
        }
    }
}


// Allow the user to enter a new element into the database. Note that the data is
// simply added to the end the database (if not full) and no sorting is carried
// out.
// Arguments:
//   (1) the element in the database to be entered
//   (2) the position of the element in the database
// Returns: void

void Enter(OpAmps &OpAmpVal, unsigned long &database_length)

{
    // if the database is full, inform the user

    if (database_length > DATABASE_MAX)
    {
        cout << "The database is full" << endl;
    }
    // if the database is not full, get the data from the user and alter the database
    // length

    else
    {
        OpAmps* pOpAmp = &OpAmpVal;

        cout << "Input the new OpAmp's name" << endl;
        cin >> pOpAmp->Name;
        cout << "Input the new OpAmp's pin number" << endl;
        cin >>  pOpAmp->PinCount;
        cout << "Input the new OpAmp's Slew Rate (V/microseconds)" << endl;
        cin >>  pOpAmp->SlewRate;

        database_length++;
        return;
    }

}
```

```
// Save the database to the file specified by DATABASE_FILENAME. If the file
// exists it is simply overwritten without asking the user.
// Arguments:
//   (1) the database
//   (2) the length of the database
// Returns: void

void Save(OpAmps *Savetofile, unsigned long &database_length)
{
    fstream output_file;  // file stream for output

    output_file.open(DATABASE_FILENAME, ios::out);   // open the file

    if (!output_file.good())
    {
        // The file could not be opened
        cerr << "FATAL ERROR: Could not create file database.";
        exit(1);
    }

    // write length information to file
    else
    {
        output_file << database_length << endl;
    }

    // write data to file
    for (int i = 0; i < database_length; i++)
    {
        output_file << endl;
        output_file << (Savetofile + i)->Name;
        output_file << endl;
        output_file << (Savetofile + i)->PinCount;
        output_file << endl;
        output_file << (Savetofile + i)->SlewRate;
        output_file << endl;
    }

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


// Load the database from the file specified by DATABASE_FILENAME. If the file
// exists it simply overwrites the data currently in memory without asking
// the user.
// Arguments:
//   (1) the database
//   (2) the length of the database
// Returns: void

void Load(OpAmps *Loadfromfile, unsigned long &database_length)
{
    fstream input_file;     // file stream for input

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

```cpp

    if (!input_file.good())
    {
        cerr << "FATAL ERROR: Could not read file database.";
        exit(1);
    }

    // load database length information from file
    else
    {
        input_file >> database_length;
        input_file << endl;
    }

    // load data from file
    for (int i = 0; i < database_length; i++)
    {
        input_file << endl;
        input_file >> (Loadfromfile + i)->Name;
        input_file << endl;
        input_file >> (Loadfromfile + i)->PinCount;
        input_file << endl;
        input_file >> (Loadfromfile + i)->SlewRate;
        input_file << endl;
    }

        // close the file
        input_file.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 Sort(OpAmps *SortDB, unsigned long &database_length)
{
    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':
        qsort(SortDB, database_length, sizeof(OpAmps), SortByName);
        break;

    case '2':
        qsort(SortDB, database_length, sizeof(OpAmps), SortBySlewRate);
        break;

    case '3':
        break;
    }
}

// Compare function for qsort, to help sort the elements by the Name member of
// OpAmps.
// Items should be sorted into alphabetical order.
// Arguments:
//   (1) a database item
//   (2) a database item
// Returns: result of the comparison
//

int SortByName(const void*a, const void* b)
{
    return (*((OpAmps*)a)->Name - *((OpAmps*)b)->Name);
}

// Compare function for qsort, to help sort the elements by the SlewRate member of
// OpAmps.
// Items should be sorted in increasing value of slew rate.
// Arguments:
//   (1) a database item
//   (2) a database item
// Returns: result of the comparison

int SortBySlewRate(const void*a, const void* b)
{
    return (((OpAmps*)a)->SlewRate - ((OpAmps*)b)->SlewRate);
}

// Display all of the messages in the database.
// Arguments:
//   (1) the database
//   (2) the length of the database
// Returns: void

void Display(OpAmps *DisplayMessages, unsigned long &database_length)
{
    fstream input_file;

    input_file.open(DATABASE_FILENAME, ios::in);
    input_file >> database_length;
    input_file << endl;
    // if the database is empty, inform the user
    if (database_length == 0)
    {
        cout << "The database is empty";
        return;

    }

    // if the database is not empty, display all the elements in the database
    else
        for (int i = 0; i < database_length; i++)
        {
            cout << endl;
            cout << (DisplayMessages + i)->Name;
            cout << endl;
            cout << (DisplayMessages + i)->PinCount;
            cout << endl;
            cout << (DisplayMessages + i)->SlewRate;
            cout << endl;
        }
}
```