# Programming Exercise 13.1

**Name Sorter**

**Purpose**. The purpose is for you to demonstrate that you have mastered arrays, array-based lists, sorting, and Boolean search.

**Requirements**. Write a program named **`nameSorter.cpp`**, to read names in a text file, sort them alphabetically, and output them to the screen. Here are some detailed specifications:

1. The names are to be stored in an input *text file*, one name per line. Prompt the user to enter the filename via the console keyboard.

2. Skip any *blank lines* that may be in the input file.

3. Skip any *exact duplicates* that may be in the input file.

4. Allow *up to 5* names. Stop reading the file after the end-of-file is reached, or the 5th name is added and the list reaches its capacity.

5. *Sort the list alphabetically*, from A to Z, according to the first letter in the name string.

**Optional Requirement.** The sorting and checking for duplicates should both be *case-independent*. But do *not* permanently convert the name case -- compare lowercase (or uppercase) versions of the text for sorting purposes.

**Program I/O.** Input: a filename from the console keyboard, and then open that text file to input names. Output: the list of stored names to the console screen, in sorted order.

**Example.** Your program's console I/O should look something like this, with user input in blue:

```
Enter the name of the file containing names: myFriends.txt

Alex
beth
Carl
Pat
Sasha
```

# Programming Exercise 13.1

**Name Sorter**

**Purpose**. The purpose is for you to demonstrate that you have mastered arrays, array-based lists, sorting, and Boolean search.

**Requirements**. Write a program named **`NameSorter.java`**, to read names in a text file, sort them alphabetically, and output them to the screen. Here are some detailed specifications:

1. The names are to be stored in an input *text file*, one name per line. Prompt the user to enter the filename via the console keyboard.

2. Skip any *blank lines* that may be in the input file.

3. Skip any *exact duplicates* that may be in the input file.

4. Allow *up to 5* names. Stop reading the file after the end-of-file is reached, or the 5th name is added and the list reaches its capacity.

5. *Sort the list alphabetically*, from A to Z, according to the first letter in the name string.

**Optional Requirement.** The sorting and checking for duplicates should both be *case-independent*. But do *not* permanently convert the name case -- compare lowercase (or uppercase) versions of the text for sorting purposes.

**Program I/O.** Input: a filename from the console keyboard, and then open that text file to input names. Output: the list of stored names to the console screen, in sorted order.

**Example.** Your program's console I/O should look something like this, with user input in blue:

```
Enter the name of the file containing names: myFriends.txt

Alex
beth
Carl
Pat
Sasha
```

# Programming Exercise 13.1

**Name Sorter**

**Purpose**. The purpose is for you to demonstrate that you have mastered arrays, array-based lists, sorting, and Boolean search.

**Requirements**. Write a program named **`nameSorter.py`**, to read names in a text file, sort them alphabetically, and output them to the screen. Here are some detailed specifications:

1. The names are to be stored in an input *text file*, one name per line. Prompt the user to enter the filename via the console keyboard.

2. Skip any *blank lines* that may be in the input file.

3. Skip any *exact duplicates* that may be in the input file.

4. Allow *up to 5* names. Stop reading the file after the end-of-file is reached, or the 5th name is added and the list reaches its capacity.

5. *Sort the list alphabetically*, from A to Z, according to the first letter in the name string.

**Optional Requirement.** The sorting and checking for duplicates should both be *case-independent*. But do *not* permanently convert the name case -- compare lowercase (or uppercase) versions of the text for sorting purposes.

**Program I/O.** Input: a filename from the console keyboard, and then open that text file to input names. Output: the list of stored names to the console screen, in sorted order.

**Example.** Your program's console I/O should look something like this, with user input in blue:

```
Enter the name of the file containing names: myFriends.txt

Alex
beth
Carl
Pat
Sasha
```