University of the Witwatersrand

Computer Science III

Software Engineering III

Course Project Report

Authors:
Donovan Platt (599055)
David Torpey (674425)
Jeroen Schmidt (604309)
Mufaro Simbisayi (563562)

Lecturer:
Prof E.J.Otoo

May 31, 2014

Contents

1 Requirements Analysis

1.1 Introduction

The problem is to design and implement software that will read in data from a text file, process the data in some way (sorting), and save the data in various sources such as a text file. A DBMS (Database Management System) will also be used for storing and retrieving data in the software.

1.2 Outline

The program will read in input from a text file. Each line in this text file will contain the data in a CSV (comma-separated value) format.

The program will read in the aforementioned data, store it in a database (specifically, the SQLite3 Database Management System). The student number of each student will be used as a key in the table, as a student number is a unique, defining characteristic of each student that will (should) never be a duplicate. The data will then be available for future retrieval from the database.

The data will then be retrieved from the database, and will be stored in various arrays. Eventually these arrays will be sorted according to student number via a *QuickSort* algorithm. The user will be able to inform the program, via a flag, whether to sort the data in ascending or descending order. This input flag will be read in upon initial running of the program.

The results of this sorting algorithm will then be written to an output text file in the following format:

Key: FirstName LastName

The data in this output text file will either be in descending or ascending order, depending on what the user requested initially.

There will be various error checks during the duration of program execution, to ensure catastrophic failure or crashing of the software does not occur. These checks will include checking the flag for validity, checking that the database exists and creates it if not, and the same for the various text files used.

The program will run via the command line or via Python's IDLE. There will be no graphical user interface/user interface. The software will be able to run on all common operating systems, such as Windows and Unix-based systems.

1.3 Limitations

For what the program's intended function is, there are no limitations on resources. This program does not perform any CPU-intensive operations. The software requires very basic resources, and will be able to run on any modern computer, such as a personal computer, laptop, mobile cellular telephone, tablet device, etc.

Limitations of the software itself include the fact that only student names of the form *LastName FirstName* are catered for. Essentially, double barrel names and/or surnames are not considered.

1.4 Acceptance Criteria

- The software must be able to read in from an input text file, containing any number of entries.
- The program must be able to store this data in a database, using the database management system SQLite.
- The data must be stored in the database using the student number of each student as a key.
- The program must read from the database.
- The program must allow the user to choose whether to sort the data in ascending or descending order, via an input flag taken in through terminal input.
- The program must sort the retrieved data according to the input flag.

- The data must be sorted using a *QuickSort* algorithm.
- The sorted data must then be written to an output text file.
- The format in this output text file must be *Key: LastName FirstName.*

2 System Design Document

2.1 System Overview

The software will receive input in the form of a plain text file, in which each line has the format: Key, LastName FirstName. The software will ensure that each line is of the correct format and if not, display an error message informing the user and identifying the number of the first line containing an error. Should the text file be of the correct input format, the content of the text file will be split such that each line of the text file forms a record in an SQLite database with 3 fields: Key, FirstName and LastName. The data from the database is then extracted and the user prompted via standard input to enter a flag to indicate if they wish to sort the data in ascending or descending order. If the flag entered is not of the correct format, the user will be informed of this error and the software will terminate. If the flag is correct, the data extracted from the database will be sorted using the quicksort algorithm in descending or ascending order depending on the user's entered flag. The result of this sort will then be outputted to a text file in a similar format to the input text file and the first 15 and last 15 records will be displayed via standard output. Should there be fewer than 15 records, all records will be displayed. The software will account for a variety of errors and at all times inform the user of any errors in input, giving the user clear instructions on what needs to be changed.

2.2 System Architecture

2.2.1 Basic Design and Decomposition Description

The software is to be implemented using the Python programming language.

The software will consist of various modules and use the approach of functional decomposition. Each member of the team will develop one or more of these modules depending on the relative complexity of each module. Each module will contain a function that performs a specific task in the overall software. The modules will consist of:

- CreateDatabase, which creates the database and appropriate tables if either of these do not already exist.
- ReadIn, containing a function that extracts information from the text file and calls another module, CheckFormat to ensure that the format of each line in the text file is correct. It returns a 2D array consisting of the keys and full names if no errors are encountered; otherwise it displays an appropriate error message and exits.
- CheckFormat, checks that each line passed to it is in fact of the correct format in the text file. Makes use of regular expressions.
- DataStore, which stores the content extracted from the text file in the database created using appropriate SQL statements.
- ReadDatabase, which extracts all information stored in the database and fills three arrays for the purposes of sorting.
- Sort, which contains a quicksort function that takes in a flag and the array previously generated and returns the sorted array in ascending or descending order depending on the flag passed to the function.
- OutputFile, which writes the sorted content to a text file in a similar format to the input text file.
- TerminalPrint, which prints the first 15 and last 15 sorted records using standard output. If there are fewer than 15 records, all records are printed.
- TestProg, the main unit which calls all other units and their respective functions to ensure that the software functions. It performs a large amount of the total error checking in the program, before values are passed to the functions in other modules.

2.2.2 Database Design

The database consists of a simple SQLite database. This database consists of a single table, the "People" table, with 3 fields, FirstName, LastName and Key, where Key is the primary key and must be unique. Each field is of the type "text".

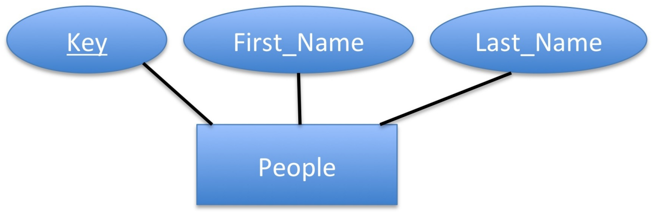


Figure 1: ER Diagram of Database Design

2.2.3 Design Rationale

The Python programming language was chosen over C or C++ because of the fact that it is far easier to manipulate strings in Python. Because a large part of the software involves extracting data from a text file, Python was deemed to be most efficient for this application. SQLite was chosen over GDBM due to its inclusion of SQL, greatly simplifying interaction with the database. A modular structure with functional decomposition was chosen as this allows each member of the team to work independently on a particular part of the software functionality without affecting the progress of other members of the team. It also allows for reuse of code where necessary and improves the efficiency of maintenance and evolution.

2.3 Component Design

The modules that will be included in the software in more detail are as follows:

2.3.1 CreateDatabase Module

This module creates the SQLite database that will be used by the software and also creates all appropriate tables. If the database exists, it simply links the software to the database. It creates a table called "People" with three fields: Key, FirstName and LastName through the use of SQL. If the table already exists, nothing is done.

Identification	CreateDatabase.py
Type	Module Containing Functions
Purpose	Create the SQLite database used by the software and create the ”People” table as described above.
Function	• Create the database used by the software. • Create the ”People” table with fields: Key, FirstName and LastName.
Dependencies	Requires SQLite.
Interface with other components	Interfaces with database used by software and is called by TestProg.
Methods	• createDatabase(), performs all the processes described above when creating the database.
Processing	See Section 2.4, Algorithm 1

2.3.2 ReadIn Module

This module reads the content of an input text file. It reads in each line and checks that the current line is of the correct format by passing it to a function in the CheckFormat module. If the entire text file is of the correct format, the content of the file is split into two arrays, key and fullName. These two arrays are then used to form a two dimensional array which is returned.

Identification	ReadIn.py
Type	Module Containing Functions
Purpose	Check that the input text file is of the correct format. Extract content from the text file.

Function	• Check that each line is of the form: Key, LastName FirstName. • Store the content of the text file in two arrays, fullName and key.
Dependencies	Requires the input text file.
Interface with other components	Called by TestProg.
Methods	• readIn(fileInput), performs all the processes described above when reading in the text file.
Processing	See Section 2.4, Algorithm 2

2.3.3 CheckFormat Module

This module contains a function which takes in a single line of text and a line number and determines that it is of the correct format: Key, LastName FirstName. If the line is correct, it simply returns the line, otherwise it displays an error message, indicating that this line is of the incorrect format.

Identification	CheckFormat.py
Type	Module Containing Functions
Purpose	Check that a specific line is of the correct format.
Function	• Check that each line is of the form: Key, LastName FirstName.
Dependencies	Requires Regular Expressions.
Interface with other components	Called by ReadIn

Methods	• checkFormat(line, lineNumber), performs all the processes described above when checking that each line is valid.
Processing	See Section 2.4, Algorithm 3

2.3.4 DataStore Module

This module contains 2 functions. The first function takes in 3 arrays as parameters: key, firstName and lastName, then connects to the database and deletes any content which may already be in the "People" table using a "Delete" SQL statement. Following this, the first function will call another function that takes in a single key value as a parameter. If this key already exists in the database, an error message will be displayed via standard output and the software will terminate. Otherwise, the function terminates and the function which called it adds the current record to the database using an "Insert" SQL statement.

Identification	DataStore.py
Type	Module Containing Functions
Purpose	Check for duplicate keys and insert records extracted from the text file into the database.
Function	• Connect to database. • Use the content of the 3 passed arrays to build a record that is inserted into the database. • Check if the current record will result in duplicate keys and if so, display an appropriate error message.
Dependencies	Requires SQLite.
Interface with other components	Connects to database and is called by TestProg

Methods	• store(key, firstName, lastName): Takes in 3 arrays, builds a record using array content, calls checkKey on each record and if valid uses an ”Insert” statement to add it to the database. • checkKey(sKey): Check if sKey is already in the database and display an error message and exit if so.
Processing	See Section 2.4, Algorithm 4

2.3.5 ReadDatabase Module

This module contains a function which connects to the project database and uses a ”Select” statement to obtain all records in the ”People” table after which it creates 3 arrays: key, firstName and lastName and populates them with the data extracted from the records. It then returns an array with 3 elements, each element being one of the populated arrays.

Identification	ReadDatabase.py
Type	Module Containing Functions
Purpose	Extract content from ”People” table in database and store in 3 arrays.
Function	• Connect to database. • Obtain all content in the database using a ”Select” SQL statement • Store the content in 3 arrays and return a single array with elements being the populated arrays.
Dependencies	Requires SQLite.
Interface with other components	Connects to database and is called by TestProg

Methods	• readData(): Performs all the functionality described above when extracting information from the database.
Processing	See Section 2.4, Algorithm 5

2.3.6 Sort Module

This module contains a *QuickSort* function which receives an array and a flag as parameters. It sorts this array, which consists of concatenated data from the database in ascending or descending order of keys depending on the flag entered by the user. It then returns this array.

Identification	Sort.py
Type	Module Containing Functions
Purpose	Sort an array in ascending or descending order depending on a passed flag.
Function	• Sort content of the array by key depending on the flag passed.
Dependencies	Requires SQLite.
Interface with other components	Connects to database and is called by TestProg
Methods	• quickSort(arr, flag): Sort the passed array in ascending or descending order by key using the quicksort algorithm.
Processing	See Section 2.4, Algorithm 6

2.3.7 OutputFile Module

This module contains a function which outputs the sorted database contents, received in the form of 3 arrays: key, firstName and lastName to a textfile, where each line has the format: *Key: LastName FirstName.* It begins by creating the output file if it does not exist and clearing it if it does already exist, after which all content is written. If the file cannot be created, the software exits and displays an appropriate error message.

Identification	OutputFile.py
Type	Module Containing Functions
Purpose	Output sorted database content to a text file.
Function	• Create the output file if it does not exist, or clear it if it already exists. • Display an appropriate error message if the output file cannot be created and exit the program. • Write the database contents, passed in the form of the 3 arrays, Key, LastName, FirstName to the text file in the format *Key: LastName FirstName.*
Interface with other components	Called by TestProg
Methods	• writeFile(key, firstName, lastName, fileName): Write database content to an output text file with name stored in the parameter fileName in the format specified above.
Processing	See Section 2.4, Algorithm 7

2.3.8 TerminalPrint Module

This module contains a function which reads the output text file produced by the OutputFile module and displays the first 15 and last 15 sorted records via standard output. If there are fewer than 15 records, all records are displayed.

Identification	TerminalPrint.py
Type	Module Containing Functions
Purpose	Print the first and last 15 records of the output text file to the terminal (standard output)
Function	• Print the first 15 and last 15 sorted records in the output text file. • If fewer than 15 records exist, print all records to the terminal.
Interface with other components	Called by TestProg
Methods	• printOutFile(): Print first 15 and last 15 records, if fewer exist, print all records to the terminal.
Processing	See Section 2.4, Algorithm 8

2.3.9 TestProg Module

This module acts as a main module. It is responsible for receiving all user terminal input and calling all the previously discussed functions in such a way that the overall functionality of the software is achieved. It performs all error checking relating to user terminal input and can be used to run the various test cases for the program by specifying different input text files.

Identification	TestProg.py
Type	Module
Purpose	Act as main module and call all other modules to provide program functionality.

Function	• Receive all user terminal input. • Call all functions in other modules to obtain the desired functionality. • Check that all input formats are correct before passing to any other modules.
Interface with other components	All other units called by TestProg.

2.4 Pseudo Code for Modules

Algorithm 1 Create Database and "People" Table

```
1: function CREATE
2:     if SEProj.db doesn't exist then
3:         Create SEProj.db
4:     if "People" table does not exist then
5:         Create "People" table
6:         Add text fields to "People" table: Key, FirstName, LastName
```

Algorithm 2 Separate Text File Contents, Read Into Arrays

```
Import CheckFormat module (Algorithm 3)
Create key array
Create fullName array

function READIN($fileInput$)
    $lineNumber = 0$

    for $line$ in $fileInput$ do
        $lineNumber$ += 1
        $line$ = checkFormat($line$, $lineNumber$)
        Append key segment of line to key array
        Append full name segment of line to fullName array

    return [$key$, $fullName$]
```

Algorithm 3 Check Format of Individual Lines from Input File

```
function CHECKFORMAT($line$, $lineNumber$)
    if line matches the regular expression d{6}[,][a-zA-Z]{2,25}[ ][a-zA-Z]{2,25}$ then
        Return $line$
    else
        Print "Text file contains line(s) of incorrect format"
        Print "Incorrect format on line " + $lineNumber$
        Print "Please check that the lines in the file match the format specified in the README"
        Exit program
```

Algorithm 4 Add Extracted Content to Database

```
function DataStore($key$, $firstName$, $lastName$)
    $conCursor$ ← Connection to SEProj.db

    for $i$ from 0 to length of $key$ - 1 do
        checkKey($key[i]$)
        Insert key[i], firstName[i], lastName[i] into $conCursor$
    Close $conCursor$ connection to SEProj.db

function checkKey($sKey$)
    $key$ ← keys from SEProj.db
    for $i$ in $key$ do
        if $i = sKey$ then
            Print ”Duplicate keys detected in text file, please ensure that all keys are unique”
            Exit Program
```

Algorithm 5 Read Data from Database

```
function readData
    $conCursor$ ← Connection to SEProj.db

    $key$ ← keys from $conCursor$
    $firstName$ ← first names from $conCursor$
    $lastName$ ← last names from $conCursor$

    Close SEProj.db connection to $conCursor$

    return [key, firstName, lastName]
```

Algorithm 6 Sort Array Using QuickSort

```
function quickSort(arr, flag)
    Create less array
    Create equal array
    Create greater array

    if $arr$ = empty then
        return Empty array

    if length of $arr > 1$ then
        $pivot$ = arr[0]

        if $flag$ = (”ASC” or ”asc”) then
            for $i$ in $arr$ do
                if $i < pivot$ then
                    Append $i$ to $less$
                if $i = pivot$ then
                    Append $i$ to $equal$
                if $i > pivot$ then
                    Append $i$ to $greater$

        if $flag$ = (”DESC” or ”desc”) then
            for $i$ in $arr$ do
                if $i < pivot$ then
                    Append $i$ to $less$
                if $i = pivot$ then
                    Append $i$ to $equal$
                if $i > pivot$ then
                    Append $i$ to $greater$

        return quickSort(less, flag) + $equal$ + quickSort(greater, flag)
    else return $arr$
```

Algorithm 7 Output Text File

```
function writeFile(key, fisrtName, lastName, fileName)

    if Creation of $fileName$.txt succeeds then
        $file$ ← connection to "$fileName$.txt"
    else
        Print "File could not be created"
        Print "Application will now terminate"
        Exit Program
    for $i$ from 0 to length of key - 1 do
        Write line to file: key[i] + ': ' + lastName[i] + ' '+ firstName[i]
    Close connection to fileName.txt
```

Algorithm 8 Print First and last 15 Entries from Text File

```
function PrintOutFile
    $file$ ← connection to "lab3Output.txt"
    Create empty key array
    Create empty fullname array

    for $line$ in $file$ do
        Append key segment of line to key
        Append full name segment of line to fullName

    if Length of key $\geq$ 15 then
        for $i$ from 0 to 14 do
            Print "key[i] + ":" + fullName[i]"

        for $i$ from length of key -16 to length of key - 1 do
            Print "$key[i]$ + ":" + $fullName[i]$"
    else
        for $i$ from 0 to length of key -1 do
            Print $key[i]$ + ":" + $fullName[i]$
```

2.5 Human Interface Design

Due to the relative lack of complexity of the task, the software uses terminal input and output as the development of a graphical user interface is beyond the scale of this software. The software begins by prompting the user to enter the name of the text file which they would like to sort via standard input. If the file does exist or any lines of its are of the wrong format, the user is notified by an appropriate error message using standard output. If the text file is of the correct format, the lines of the text file will be stored in the "People" table of the database. Following this, the user is asked to enter a flag, either "ASC" or "DESC" to indicate the sort order. If the flag is of the wrong format, the user is notified by an error message using standard output and the program exits. If the flag is of the correct format, the records stored in the database are sorted according to key in the order specified by the user flag and then outputted to a text file. The first 15 and last 15 records are then displayed using standard output and if there are fewer than 15 records in the database, all records are displayed. If this text file cannot be created, an error message is displayed using standard output and the program exits.

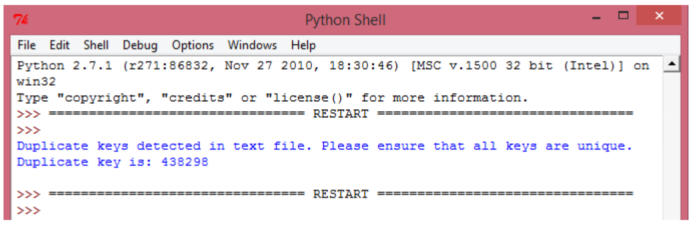
3 Testing

The program will be tested to ensure that no combination of input will result in the software crashing or any form of catastrophic failure of the program. In order to repeat the tests performed, one would run the TestProg module and when prompted to enter the name of the text file to read, enter Testing/ followed immediately by any of the text file names mentioned in the following report, for example: Testing/lab3Input2.txt The program will be tested using "Black Box" testing in which a number of different inputs will be entered and the output checked to ensure that it matches expectations. The output text file may be found in the "Files" folder.

The only standard input the user provides is the input flag which represents whether the data must be sorted in ascending or descending order and the name and path of text file which is to be sorted. This input flag can only be one of the following two values: 1) ASC 2) DESC (or asc and desc).

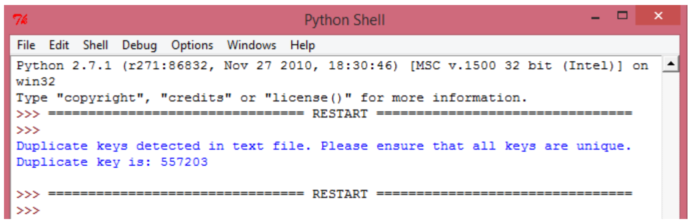
The input text file will also be checked for any invalid line formats or errors. All possible cases of input are tested, and their results shown below:

First execution of program with input text file *lab3Input.txt*:

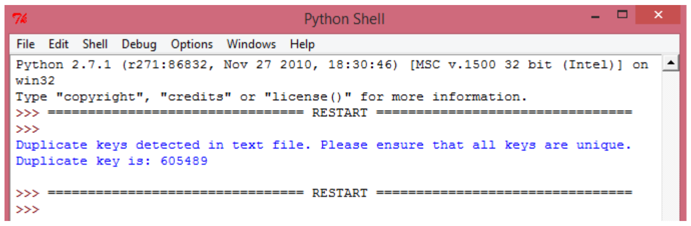


```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Duplicate keys detected in text file. Please ensure that all keys are unique.
Duplicate key is: 438298

>>> ================================ RESTART ================================
>>>
```

As can be seen, the program notifies the user that the input text file contains duplicate student numbers, and informs the user which student number is a duplicate. This is then fixed and the program is executed once again, which results in the same error two more times, as can be seen below:

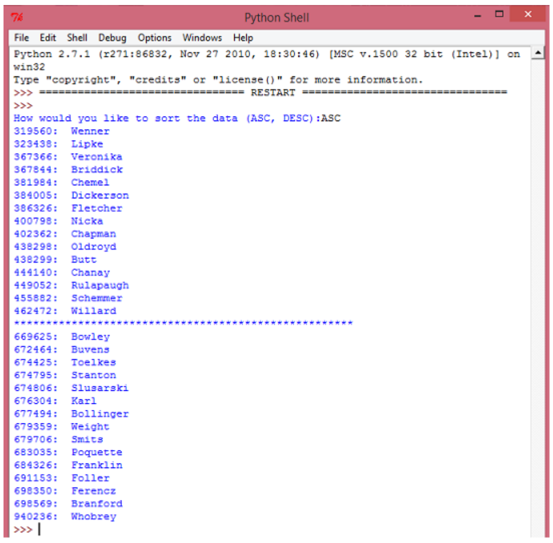


```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Duplicate keys detected in text file. Please ensure that all keys are unique.
Duplicate key is: 557203

>>> ================================ RESTART ================================
>>>
```



```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Duplicate keys detected in text file. Please ensure that all keys are unique.
Duplicate key is: 605489

>>> ================================ RESTART ================================
>>>
```

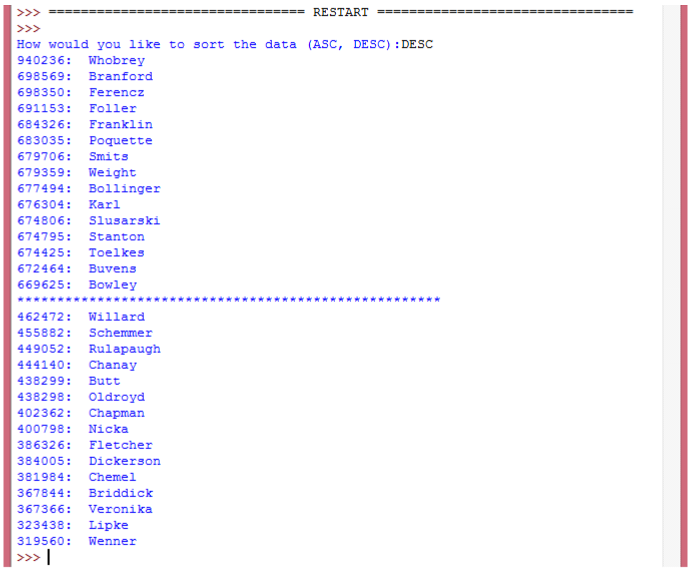
The above two errors are fixed by assigning unique keys, and the program is run again. Now that all keys are unique, we choose to sort in ascending order by typing ”ASC” File Used: lab3InputB.txt



```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
How would you like to sort the data (ASC, DESC):ASC
319560:  Wenner
323438:  Lipke
367366:  Veronika
367844:  Briddick
381984:  Chemel
384005:  Dickerson
386326:  Fletcher
400798:  Nicka
402362:  Chapman
438298:  Oldroyd
438299:  Butt
444140:  Chanay
449052:  Rulapaugh
455882:  Schemmer
462472:  Willard
*****************************************************
669625:  Bowley
672464:  Buvens
674425:  Toelkes
674795:  Stanton
674806:  Slusarski
676304:  Karl
677494:  Bollinger
679359:  Weight
679706:  Smits
683035:  Poquette
684326:  Franklin
691153:  Foller
698350:  Ferencz
698569:  Branford
940236:  Whobrey
>>> |
```

The first fifteen and last fifteen names of the sorted sorted text file are displayed correctly, as can be seen above.

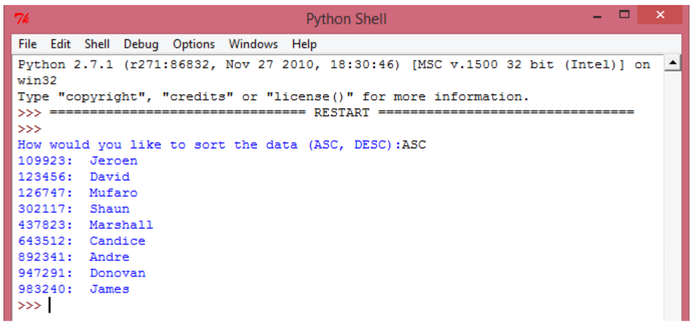
Now we execute the program, this time choosing to sort in descending order by typing ”DESC”:



```
>>> ================================ RESTART ================================
>>>
How would you like to sort the data (ASC, DESC):DESC
940236:  Whobrey
698569:  Branford
698350:  Ferencz
691153:  Foller
684326:  Franklin
683035:  Poquette
679706:  Smits
679359:  Weight
677494:  Bollinger
676304:  Karl
674806:  Slusarski
674795:  Stanton
674425:  Toelkes
672464:  Buvens
669625:  Bowley
*****************************************************
462472:  Willard
455882:  Schemmer
449052:  Rulapaugh
444140:  Chanay
438299:  Butt
438298:  Oldroyd
402362:  Chapman
400798:  Nicka
386326:  Fletcher
384005:  Dickerson
381984:  Chemel
367844:  Briddick
367366:  Veronika
323438:  Lipke
319560:  Wenner
>>> |
```

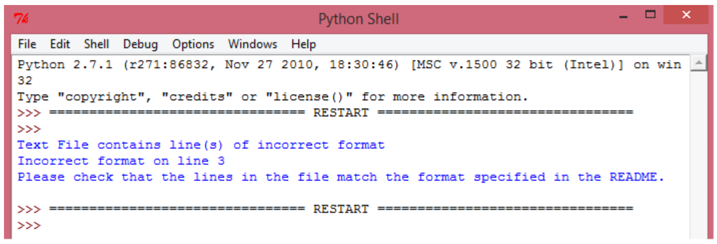
Once again the first and last fifteen names of the sorted list are correctly displayed.

Now to more thoroughly test the software, we create our own text files with errors that will test all of our code. The first text file contains the test of dealing with a text file that has less than 30 entries (for the case of this test, we will sort in ascending order). File used: Testing/lab3Input1.txt



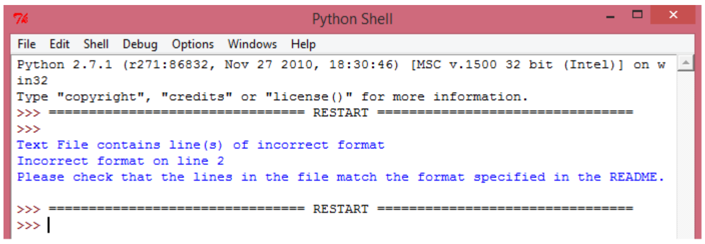
```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
How would you like to sort the data (ASC, DESC):ASC
109923:  Jeroen
123456:  David
126747:  Mufaro
302117:  Shaun
437823:  Marshall
643512:  Candice
892341:  Andre
947291:  Donovan
983240:  James
>>> |
```

The next test is one where a name in the input text file contains a non-alphabetic character: File used: Testing/lab3Input2.txt



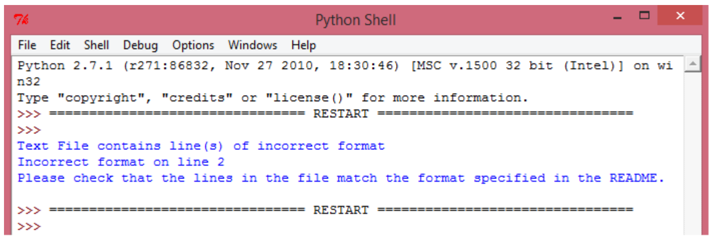
```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Text File contains line(s) of incorrect format
Incorrect format on line 3
Please check that the lines in the file match the format specified in the README.

>>> ================================ RESTART ================================
>>>
```

The following test deals with a very long student name: File used: Testing/lab3Input3.txt



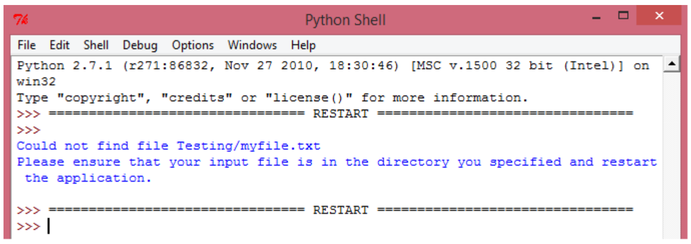
```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on w
in32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Text File contains line(s) of incorrect format
Incorrect format on line 2
Please check that the lines in the file match the format specified in the README.

>>> ================================ RESTART ================================
>>> |
```

The next test will involve a student name with only one word: File used: Testing/lab3Input4.txt



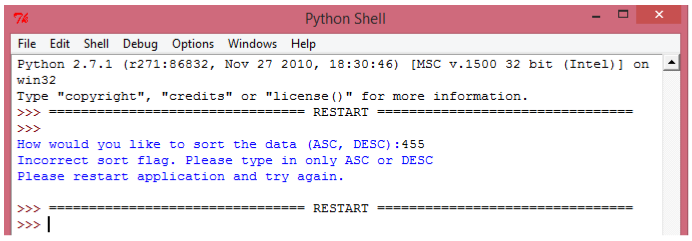
```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on wi
n32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Text File contains line(s) of incorrect format
Incorrect format on line 2
Please check that the lines in the file match the format specified in the README.

>>> ================================ RESTART ================================
>>>
```

The penultimate test will involve a text file that is not found/does not exist:



```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Could not find file Testing/myfile.txt
Please ensure that your input file is in the directory you specified and restart
 the application.

>>> ================================ RESTART ================================
>>> |
```

The final test will involve a valid text file, but inputting an invalid sort flag:



```
Python 2.7.1 (r271:86832, Nov 27 2010, 18:30:46) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
How would you like to sort the data (ASC, DESC):455
Incorrect sort flag. Please type in only ASC or DESC
Please restart application and try again.

>>> ================================ RESTART ================================
>>> |
```

As can be seen, all tests were passed and any invalid input was detected and the user correspondingly informed. Therefore, since the program can handle all of the above, we see that the user requirements have been met and the project is a success.

4 Group Work and Planning

The word was divided as follows:

4.1 Assigned Tasks

- Donovan: ReadIn and DataStore modules, section 2 of this document (Design) and section 4 of this document (Group Work), compilation of all work in Latex.
- David: Sort and ReadDatabase Modules, section 1 and 3 of this document (Requirements and Testing).

- Jeroen: CheckFormat and OutputFile Modules. Contributions to pseudo code, debugging.
- Mufaro: CreateDatabase and TerminalPrint Modules, README file.
- TestProg was a team effort involving all members.

4.2 Work Flow of Project

The project was conducted using the approach of incremental development. The idea was to deliver functionality incrementally. We began by ensuring that a database and appropriate tables could be created. We dealt with any errors, such as the database or tables already existing. We thus created the CreateDatabase module. In the next iteration, we implemented both the ReadIn and DataStore modules, to ensure that we could store the content from the text file in the database. Following this we implemented the ReadDatabase and Sort module to ensure we could read from the database and sort this data. Finally we implemented the OutputFile and TerminalPrint modules that allowed us to output the necessary data. In each cycle, we gradually developed the design and requirements, gaining a greater understanding of the project as we went along. At each stage, we combined the various components using an online GIT repository (GitHub) and we were able to efficiently complete the project. Any errors were identified and dealt with as the project progressed. Some extreme programming measures were also implemented, for example pair programming. For a number of the later modules, Donovan would develop the module while Jeroen would read along and identify any flaws or errors he saw.