1996

Improved Data Migration and Processing for Projecting the Financial Aid Budget

Jeffery L. Olson '96
Illinois Wesleyan University

Improved Data Migration and Processing for Projecting the Financial Aid Budget

by Jeffery L. Olson

Improved Data Migration and Processing for Projecting the Financial Aid Budget,

an Overview by Jeffery L. Olson

The foundation for this research was laid during the 1993-94 school year by Amy N. Baird. Her project mode it possible to move raw financial aid information from the AS400 to a Quattro Pro 4 spreadsheet on a PC in the Financial Aid Office. The spreadsheet was then used to generate a series of averages that were in turn used to predict the following year's financial aid expenditures.

During the 1994-95 academic year, Stephanie M. Pannier resumed the project. Working with the Director of Financial Aid, she migrated from Quattro Pro® 4 to Quattro Pro® 6 for Windows®. More importantly, her intensive work with spreadsheet formulas improved the methodology used in the computer-aided budget projection.

Last fall, I again resumed work on the budget projection model that encompassed five spreadsheets. Four of these sheets generated a set of statistical averages for each class. Each one consisted of 101 columns containing data for the four- to five-hundred students(rows) in each class. In addition, a fifth sheet used these averages to generate a highly accurate prediction for expenditures in the upcoming year. However, there were two main areas of improvement that became readily apparent: importing data and the sheets themselves.

The BDGTPREP.EXE Program for sorting and formatting input files

The process for importing data from the AS400 to Quattro Pro® was highly error prone and labor intensive. The three ASCII files containing the financial aid information

for each student (one containing FAF information, one with scholarship information, and one with loan information) were imported into three columns. Unfortunately, not all students who filed a FAF with IWU received aid (and therefore did not appear in the second two files). As a result, the user had to go through "by hand" and align the names in the three files to ensure that the correct information was associated with each row in the spreadsheet.

Additionally, the entire data stream making up a student's record in the imported files needed to be split into columns. Quattro Pro® 6 has a "parse" option for dealing with this situation. When the "parse" option is selected, Quattro Pro® 6 produces a default template which can then be edited to ensure proper parsing. For example, the template "L>>>L>>>V>>>>V>>>>>" would parse an eighteen character data stream into 4 columns: 2 alphanumeric each containing 4 characters; and 2 numeric containing 5 numbers.

However, the files that were being imported do not contain delimiters of any kind between fields (i.e., "0100001000" actually represents two fields: "01000" and "01000"). As a result the template line could not be automatically generated by Quattro Pro® and needed to be typed in manually. This required one line 86 characters in length and two 197 characters in length much like the example in the preceding paragraph to be entered error free.

After these steps were completed, the sheet was then copied three times. This created four sheets which each contained information about all four classes. Each sheet then had three classes edited out of it, leaving four sheets which each contained

information pertaining to one class. This data was then used to generate statistical averages.

I immediately came to a realization after performing these steps last fall. If the incoming data files could be programmatically combined and sorted, many hours of labor could be saved. After sorting the input files, spaces were inserted into the output files between fields allowing Quattro Pro® to produce a parsing template automatically. The result was BDGTPREP.EXE. This DOS program can also be run through Windows® 3.x, Windows NT®, Windows® 95, or OS/2®. It takes the three text files downloaded from the AS400, splits them by class, sorts the resulting 12 files by name, and combines them into four output files. These are then imported into Quattro Pro®. This provides a method where errors due to typos and human mistakes are largely eliminated.

New Layout of Quattro Pro® worksheets.

Besides improved data importation methods, many improvements have been made to the budget projection spreadsheets as well. The "Current Year Budget" and "Projected Year Budget @ X%" columns have been removed. Since this information is the same for each student, it has been moved to 2 cells in the top of the sheet reducing data redundancy. This required that formulas in other columns referring to the removed columns be updated to the new cell location using *absolute cell addressing*. This allows these formulas to copied without the cell locations referring to the budget being changed.

Color shading and boldface type has been added to help make more sense of the vast amount of data contained in the model. Columns have also been added to the projected freshman sheet. These columns are duplicate IWU Award and work study

columns. These can be employed to run "scenarios" for financial aid. For example, if all IWU Talent Awards were increased by $100, what would the projected budget be?

Additionally, each class spreadsheet is actually two sheets in a "common notebook". Sheet B of each notebook contains the raw data that was parsed from the input file. Sheet A contains copies of these columns as well as other columns which mathematically manipulate the data. This provides several advantages. Columns in sheet A can be moved around arbitrarily since their order is no longer physically dependent on the order they are placed when parsed. Automatic updates of input data is now possible by modifying the formulas that copy data from sheet B to sheet A. For example, students who have no financial need have Contribution IM values of '99999'. In order for the projection model to run properly, these high values must not be allowed to skew the averages. Stephanie Pannier solved this problem by replacing these values with 'NN' for 'no need' in what she referred to as "a tedious but important procedure." Now, however, these values are simply replaced when they are copied over with an IF statement.

The budget projection spreadsheet that uses the information found in the class spreadsheets has also been modified. Previously, data was manually copied from the class spreadsheets into the budget projection spreadsheet. One solution could have involved putting all the classes and the projection sheet (9 pages total) into a common spreadsheet notebook. Unfortunately, each class sheet takes up approximately 10Mb of space, making the overhead of such a scheme quite high. However, Quattro Pro® has the ability to address information from another spreadsheet file in its formulas. For example

"+[FRESHMAN]A:B10" means that the cell containing this formula should be equal to cell B10 on sheet A in the FRESMAN.WB2 spreadsheet notebook.

This allows the derived averages to be automatically copied (through formulas) to the spreadsheet file that actually projects the budget. Pursuant to this, the rows containing the totals and averages have been moved to the top of each class sheet. The bottom of each class sheet can never be known before importing the data since it is dependent on the number of students in that class. By moving these to the top, they can by referenced in the projection sheet without yearly updates.

Data movement between Quattro Pro® files requires that the names of the class spreadsheet files always be the same from year to year. This mandates that new directories be created each year to sore that year's projection. Interestingly enough, that solves another lament of the Financial Aid Director. Hobbled by the eight character naming convention, cryptic names had been used for files in the past. For example "PFR9697" translates to "projected freshman class for the 1996-97 year." Now the path in the directories helps clue us in : "\FILES \ BUDGET\ 1996-97\ PROJECTN\ FRESHMEN.WB2." While not as good as 255 character naming conventions, this is certainly a step in the right direction.

The resulting product is one that has taken a great projection methodology to the next level. A process that took more than two weeks to complete last fall can now be easily completed within an afternoon. I would like to thank Amy N. Baird and Stephanie M. Pannier for making this project possible with their hard work and innovation. In addition, my thanks goes to Dr. Susan Anderson-Freed, Dr. Lisa Brown, and Dr. Lionel Shapiro for providing me the technical background and assistance to turn my ideas into a

reality. I would also like to express my gratitude to Lynn Nichelson whose vision directed this ongoing project, and whose dedication and enthusiasm made this project such a great experience.

Illinois Wesleyan University

Computer Aided Budget Projection:

Reference & Methodology

Illinois Wesleyan University

Computer Aided Budget Projection:

Reference & Methodology

by

Jeffery L. Olson

I would like to thank the following people, whose contributions have made this revised methodology possible:

Mr. Lynn Nichelson
Director of Financial Aid
Illinois Wesleyan University

Dr. Susan-Anderson Freed
Chairperson, Department of Computer Science
Illinois Wesleyan University

Amy N. Baird &
Stephanie M. Pannier
Previous designers of the methodology this project is based on.

Table of Contents

Introduction

Welcome! These guidelines will help you set up the Budget Projection Model in a relatively short period of time using an efficient series of steps. In addition, details concerning the manipulation, modification, or printing of sheets in the model is also detailed.

Notation:

1) Clicking refers to putting the arrow of he mouse on something and then pressing the left mouse button. Double-clicking refers to this same process, except clicking twice in rapid succession.

2) When referring to selecting items from pull down menus found at the top of windows, the Menu Name and then the menu choice to select is printed as follows:
   - **Menu | Choice**

3) For Menus where multiple levels of Menus pop out:
   - **Menu | Choice1 | Choice2** ...

4) To execute **Menu | Choice**, the user would click on the Menu menu name on the horizontal bar at the top of the window, and then click on Choice from the menu that pops down.

5) In Quattro Pro, selecting a cell can be done in two ways
   - clicking on it with the mouse cursor
   - using the arrow keys to highlight the border around the cell

Part One- Setup

In order to for project the current year financial aid budget, several preparatory steps must first be taken. First, the financial aid information must be downloaded into the ASCII files on a floppy disk. These files are named BDGTOTHR.TXT, BDGT3115.TXT, and BDGT4321.TXT. Additionally, you must have approximately 2.5MB (2,500,000 Bytes)of Hard Disk space free. This can be verified by going to the DOS prompt typing 'dir' and pressing enter. A list of files will be printed out. At the bottom of this list you will see a line reading '####### bytes free'. This number must be greater than 2,500,000.

Next, new directories to store this year's budget projection must be created. The first directory will group files of the same academic year (e.g., 1996-97), while the second, PROJECTN will group all the files used in the budget projection model. Open the "File Manager" program from within Windows by double-clicking on the file cabinet icon associated with it.

On the left side of the "File Manager" you will see a hierarchy of yellow folder icons. Above this is a gray bar with boxes, representing disk drives, and their corresponding letters next to them.

1) Click on the box labeled "C:". This selects the computers hard drive.
2) In the left window click on the "QPW" folder to open it.
3) Click "FILES"
4) Click "BUDGET".
5) Select **File | Create Directory**.
   - You will be prompted to enter a new directory name. Type '199x-9y' where x & y represent the next academic year (e.g. '1996-97').
   - You should now have a folder within 'BUDGET' with the name you just entered.
6) Click on the folder you have just created to open it.
7) Select **File | Create Directory**

- Name this directory 'PROJECTN' to indicate that it will contain budget projection information.
- You should now have a folder within your newly created year folder called PROJECTN

8) Select **File | Exit** to close the Program Manager

Part Two- The 'Budget Prep' Program

Now that you have your directories set up, and the import files in hand; you must now run "Budget Prep". This program sorts and formats the AS400 ASCII files into four text files which can be easily imported into Quattro Pro.

1) Insert the floppy disk containing the three AS400 ASCII files into drive A: (your computer's 3½ inch disk drive.

2) Start the "Budget Prep" program by double clicking on its Windows icon.
   - The following should be displayed:

```
Welcome to automated ASCII file processing for Budget Projection!

In order to proceed, this program needs to know the
location of the input files. Additionally, this program
needs to know where it should put the processed output
files.

Current File Locations:
1. (Input) BDGTOTHR.TXT ----> A:\BDGTOTHR.TXT
2. (Input) BDGT3115.TXT ----> A:\BDGT3115.TXT
3. (Input) BDGT4321.TXT ----> A:\BDGT4321.TXT
4. (Output) FRESH.TXT ------> C:\QPW\FILES\ASCII\FRESH.TXT
5. (Output) SOPH.TXT -------> C:\QPW\FILES\ASCII\SOPH.TXT
6. (Output) JUNIOR.TXT -----> C:\QPW\FILES\ASCII\JUNIOR.TXT
7. (Output) SENIOR.TXT -----> C:\QPW\FILES\ASCII\SENIOR.TXT

Enter 1-7 to modify a file location, 8 to continue:
```

3) Make sure all file locations are correct
   - If one or more of the file locations is incorrect, press the number corresponding to it followed by ENTER. You will then be prompted to enter the new file location. When referring to output files, make sure that

your filename has the extension ".TXT" on the end of it. Quattro Pro will not import files without this extension.

- When you are satisfied with the location of all of the files. Press '8' and ENTER to continue.

4) File Processing will now begin. Some student records in the file will have blank "Year" fields. As a result, the Budget Prep program will not know which output file to put the student in.

- A screen like the following will appear:

```
No year has been found with this student
First Name: "Michael     "
Last Name : "Chadwick        "

Please choose one of the following:
 1. Put the student in the Freshman file.
 2. Put the student in the Sophomore file.
 3. Put the student in the Junior file.
 4. Put the student in the Senior file.
 5. Delete this student from the Budget Projection

Press 1-5, followed by ENTER:
```

- If the name fields are blank, choose option 5. This will discard the blank record from the processed output.
- If the name fields are not blank, look this student up on the AS400, or student directory and place them in the correct class. You may see the same student's name more than once if the year was not filled in on more than one input file.
- If a mistake is made *simply let the program finish its run.* Then run Budget Prep again. The files on your next run will replace those that were incorrect.

5) When all students have been assigned a class, the program will sort and match student records. You will see messages on the screen indicating the program's progress.

6) When you see the message "Process complete" close the window that the program was running in. This is accomplished by double-clicking the gray box in the upper left-hand corner of the window.

Part Three- ASCII File Importation

Now that you have processed ASCII files, you are ready to import them into Quattro Pro 6.

1) Open Quattro Pro 6 by double-clicking on its Windows icon.

2) Select **File | Open**

- You will see a list of file names on the left, and a list of directory folders on the right.
- Double click the directory folders in this order: OFFICE\ QPW\ FILES\ SHELLS the folders will appear to open as you do this revealing the next layer of folders.
- In the file list on the right you will see a file called BPCLASS.WB2

  Click on this file name, highlighting it
- Click OK- the shell will open

3) We now need to create 4 copies of this to use in our budget projection

- Select **File | Save As..**

  This will bring up directory folder and file lists similar to the **File | Open** option

  Double-click on the folders to open OFFICE\ QPW\ FILES\ BUDGET\ 199x-9y\ PROJECTN

  Note that the last two folders are the ones you created at the beginning of this process
- Make Sure the PROJECTN folder is both highlighted *and* looks "open"
- Click above the file list box, a cursor will appear allowing you to enter a file name
- Type 'SENIORS' and then Click OK
- After a brief pause the file is saved
- Repeat step 3) to create files named 'JUNIORS', 'SOPHS', and 'FRESHMEN'

4) You should now be in the newly created 'FRESHMEN.WB2' file. If not Select **File | Open** and open the file

5) Near the bottom of the window you should see some white lettered tabs:
   - Click on the one lettered 'B'
   - You should see a screen with a dark red top border, with green headings

6) Select Cell A16 (top white cell)

7) Select **Notebook | Text** Import
   - Make sure the 'ASCII Text File' button is selected in the option box
   - Double-click on the folders to find the ASCII Files.

     The default location is OFFICE\QPW\ FILES\ ASCII
   - Highlight the FRESH.TXT file
   - Click OK

8) All of the data will be imported into column A of this sheet.

9) Press the END key on the keyboard, then press the '↓' key. This will take you to the bottom row in the sheet.
   - Write this Number down, you will need in the next step

10) Select **Notebook | Parse**
   - On the line labeled 'Input' type : 'B:A16..A#' where # is the number you just recorded as being the last row in the spreadsheet.
   - Click the Create button. A parse line will be inserted into line 16.
   - Click Edit- A new window will appear containing the parse line in the top portion and student information in the bottom.

     In the Parse Line there should be an 'L' above the first letter in each students last name. There should also be an 'L' above the first letter in each student's first name. However, there *should not be* any L's between the aforementioned ones.

     If additional L's do appear, replace them with an '>'. This should leave you with an L followed by 15 '>' or '*' characters, followed by another L.

ex) L>>>>>>>>>>>>>>>L>>>>>>>>>>>L.....

- Click OK at the bottom of this window to save any changes you made. This will close the edit parse line window.
- On the line labeled 'Output' type: 'B:A16' to tell Quattro pro where to start the parsed data
- Click OK
- The information should now be parsed into the appropriate columns.
- Select **File | Save**
- Select **File | Close**

11) Fill in the Current Budget and Projected Budget values in A:C5..C6

12) Repeat steps 4-11 for 'SOPHS.WB2', 'JUNIORS.WB2' , and 'SENIORS'.WB2. Use FRESH.TXT for the SOPHS.WB2, SOPH.TXT for JUNIORS.WB2, and JUNIORS.TXT for SENIORS.WB2

- In other words, *this* years sophomores will be *next* years juniors.. etc. Since we don't yet have the data for next year's freshmen we just use this years freshmen for this sheet as well.

Part Four- Processing Bin Blocks

Four workbooks representing each class should now have their data parsed. However, one crucial step remains. Bin Blocks must be used to count the number of non-zero entries for a given column. The result can then be used to compute the statistical averages.

1) Click on the Sheet 'C' tab at the bottom of the window. Sheet C is where all the Bin Blocks are computed, their results are the automatically copied into Sheet A to compute the formulas. To assist you, a template listing the correct Value Box entry is listed above each Bin Block.

2) For each Bin Block:
   - Select **Tools | Numeric Tools | Frequency**
   - In the area labeled 'Value Block' type the header string appearing over the Bin Block (e.g. A:C16..C999)
   - In the area labeled 'Bin Block' enter the location of the 0 and 1 appearing under the header string-- remember you are on page C! (e.g. C:A4..A5)
   - Click OK

3) Once this is done for each Bin Block, the class sheet is complete. Don't forget to save your work before closing.

Part Five- The Budget Projection Summary

The final worksheet in the projection which collates all of the projection data must now be set up.

1) Select **File | Open**
   - Click OFFICE\ QPW\ FILES\ SHELLS\ BPSUM.WB2
2) Select **File | Save As..**
   - Make sure the OFFICE\ QPW\ FILES\ BUDGET\ 199x-9y\ PROJECTN folder is highlighted & open by double-clicking on it
   - Click in the box above the file list box and type the filename 'BDGTPRJN'
   - Click OK
3) This summary of the Budget Projection should already have data in it from the class files developed above.

Appendix A. Basic Spreadsheet Commands

Here are a few of the more common operations that you will use when working with Quattro Pro 6. If any command has undesired results, simply select **Edit | Undo...** to return the sheet to the state it was in before your action. They are listed in alphabetical order for your convenience:

Align

This command allows the user to cheese whether the information in the selected cells is left justified, right justified, or centered, etc.

1) Select the cells to be aligned by painting them (click and hold the left mouse button and move the mouse). The easiest way to paint an entire column is to click on the gray letter heading above it.
2) In the third tool bar, the property band, click the fourth option from the left.
3) Click on the desired alignment

Bin Blocks

This command can be used to count the number of times a given value appears in a block.

1) Find an open space in a spreadsheet.
2) Type a zero in the cell.
3) In the cells directly below this, enter the values that you want to count in ascending order. This is your bin box. After completing the following steps, number of times the bin box value appears is printed to the right of bin box. The number of "other" values (those not found in the bin box) are listed at the bottom.

   0 (Number of Zeros found in the column)

   1 (Number of Ones found in the column)

   (Number of Values > 1 found in the column)
4) Select **Tools | Numeric Tool | Frequency**

5) In the area labeled Value Block, type the column block of the values you wish to count. (e.g. A:C16..C99)

6) In the area labeled Bin Block, enter the block where your bin block is.(e.g. A:C100..C101)

Block Copy

Copies a value from a cell (or cells) to many other cells. This function is most useful for copying formulas down an entire column.

1) Select **Block | Copy**

2) The 'From' area should contain the block address (e.g. A:A10) that you wish to copy.

3) The 'To' area should contain the block address (e.g. A:A11..A99) that you wish to copy the information to.

Block Delete

Deletes rows/columns from the spreadsheet.

1) Select **Block | Delete**

2) Enter the range of cells to delete (e.g. A:A10..A12 or A:A1..B99)

3) Select Rows or Columns to delete.

4) If the 'Entire' button is filled in, the entire row/column that a deleted cell is part of will be removed. If partial is selected, only the portion specified in part two will be deleted, with the rest of the rows/columns moving left/up to fill the gap.

Tip: if you delete a cell referenced in a formula that remains in the spreadsheet afterwords, you will receive ERR messages.

Block Insert

Inserts rows/columns into the spreadsheet. The rows are inserted in the spreadsheet before the location mentioned.

1) Select **Block | Insert**

2) Click whether you want to insert new rows or columns.

3) Enter the space you want to insert at.

4) Click OK.

Close

Closes the current spreadsheet. The spreadsheet will not be saved unless you click yes when prompted (if you just saved the spreadsheet, you won't be prompted).

1) Select **File | Close**

Exit

Exits Quattro Pro. If you have any spreadsheets that have not been closed, Quattro Pro will ask you if you want to save them before exiting.

1) Select **File | Exit**

Fit

This command adjusts the width of a column so that the widest entry in that column will fit in it.

1) Place your cursor in the desired column.

2) Click the small box in the toolbar that has two arrows in it

Locking Titles

This keeps certain portions of the spreadsheet on the screen at all times. This allows you to view your titles, no matter where you are in the spreadsheet.

1) Place your cursor in a "cornerstone position". All rows above this position and/or all columns to the left of this position will remain fixed.

2) Select **View | Locked Titles**

3) Select whether you would like to lock horizontally(columns), vertically(rows), or both. If the clear option is selected, the locked titles are removed.

4) Click OK

New

Creates a new, blank Quattro Pro worksheet.

1) Select **File | New**

Open

This command opens an existing Quattro Pro worksheet.

1) Select **File | Open**
2) Move to the desired directory by double-clicking on folders to open them. You should see the file you want to open in the file list on the right.
3) Click on the desired file to highlight it.
4) Click OK.

Print

See Appendix C.

Save and Save As..

The Save command saves a notebook. If the spreadsheet has been saved in the past, and you wish to save it under the *same* name, select **File | Save**. No other action is necessary.

If you wish to save it under another name or in a new location:

1) Select **File | Save As..**
2) Click on the desired directory folder locations by double-clicking.
3) Click in the empty box labeled File Name.
4) Type in the name you want to use.
5) Click OK

Style

The Style option allows you to modify the way information is displayed. You can cause numbers to be displayed as Currencies, with a '$' and commas.

1) Paint the portion of the Spreadsheet that you wish to apply the style to.

2) In the Property Band (third bar of the toolbar), click the third option from the left- the style list. A variety of formatting styles will become available.

3) Choose the desired style.

Appendix B. Spreadsheet Formula Development

Quattro Pro provides formulas to manipulate the data found in your spreadsheets. The following is a brief introduction in the topic. For a more detailed discussion, consult the Quattro Pro documentation.

All formulas must start with a '+' or an @ Function. Mathematical operators (+,-,*,/) can be used to perform calculations. For example, '+A12/A14 might compute an average.

@ Functions are always named with an @, followed by the name of the function, followed by values enclosed in parentheses. For example: @SUM (A1,A2) adds the numbers in parentheses together. Alternatively, instead of putting just two values in the parentheses, a range could be addressed as @SUM(A1..A999).

Another useful function is the @IF function. This function uses three arguments: @IF(<this condition is true>, <this cell should = this value>, <otherwise the cell should = this>)For example @IF(A1>A2, A1-A2, 300) means that this cell should = A1-A2 if A1>A2, otherwise it should equal 300.

The true power of formulas can be utilized by imbedding formulas within formulas. Any time an @Function requires a value as an argument, you can put another argument that returns a value in its place. For example, you can have: @IF(A2>A1, @SUM(B1..B333), @IF(A2=A1, 500, A2 - A1). The number of @Functions makes possibilities nearly endless through combinations.

Some common formulas are detailed below:

@SUM(<value1>,<value2>..<value n>) - adds the values together

@IF (<Condition Statement>, <then value>, <otherwise value>) - if the condition statement is true, this cell = <then value>, otherwise it = <otherwise value>

@ROUND(<value statement>, <number of places to round to>) - rounds off the value in part one to the number of decimal places given in part two.

Appendix C. Printing Spreadsheets

1) Select File | Print (or click the printer icon in the top toolbar)

2) Examine the 'Print Area' Box

- Current Page - if selected, will print the entire page that you were on when you selected File | Print. (e.g.- Page B of a notebook)
- Notebook- if selected, will print all pages of the notebook. (e.g. Pages A,B, & C of a 3 page notebook will be printed.
- Block Selection- Allows the user to define a section of the notebook to be printed. (e.g. 'A:B10..Z50' will print the block defined by these corners) If the spreadsheet cannot be printed on one page, this is the best option to select.

3) Click on the Sheet Options button on the right side of the Print Options Window

- If the sheet will be printed on multiple pages, and you wish the top headings of the columns to appear on all pages; you need to define a top heading. For example, a top heading of 'A:A1..Z10' would cause the top ten rows, columns a..z, of the spreadsheet to appear at the top of every printed page. In effect this is much like locking titles.
- Similarly, if the spreadsheet is too *wide* to fit on one page, you may wish to define a left heading. For example, you could define the Name columns of students to print on all pages.
- Click OK.

NOTE- Be sure that the blocks defined in your headings, and the block defined in the Block Selection do not overlap. If they do, the same information will be printed twice:

This is OK:

Block Selection- A:C10..Z99

Top Heading- A:A1..Z9

Left Heading- A:A10..C99

This is not:

Block Selection- A:A10..Z99

Top Heading- A:A1..Z9

Left Heading- A:A10..C99

4) Click the Page Setup button on the right side of the Spreadsheet Print Window

- Select Portrait or Landscape orientation for the paper by clicking the appropriate button in the bottom center of this window.

  Generally, if your spreadsheet is very wide, Landscape orientation is the best choice. This will help keep the spreadsheet from spilling over 1 or more pages to the side.

- Click Print scaling from the list of buttons on the left hand side of the window.

  By entering a scaling percentage, you can shrink the size of the type being printed. This will reduce the amount of pages needed to print the spreadsheet.

  If the 'Print to Fit' box is checked, Quattro Pro will shrink the text so it fits all on one page. However, this often causes the type to be printed too small to read easily.

  Click OK when done.

5) Click Print Preview to examine the sheet as it will be printed. When done examining the preview, click the red X at the top of the preview.

6) If you are not satisfied with the sheet as it looked in the print preview, repeat steps 2-5 as needed.

7) Click the Print button. Your document will be sent to the printer

Special Characters in headers and footers

Quattro Pro has special characters that can be inserted into the Header or Footer of your document. For example, you may want the pages numbered, or today's date printed on the Top.

1) Follow the instructions for printing as above

2) Click the Page Setup button on the right side of the main print window

3) Click the Header/Footer button from the list on the left side of the Page Setup Window.

4) Type the Special Characters into the Header and/or Footer as desired as outlined below.

You can type words in the Header/Footer Fields and they will be printed at the very top or bottom of every page.

To align text use the | as follows:

This text is on the left side of the page | This is centered | This is on the right

A -prints A on the left side of the page.

|A -centers the letter A on the page

||A -prints the letter A on the right side of the page

These special characters can be used to print out certain data.

#d - Current Date, short international format (DD/MM/YY)

#D -Current Date, long international format (Day Month, Year)

#ds -Current Date in standard short format (MM/DD/YY)

#DS -Current Date in standard long format (Month Day, Year)

#t -Current Time in short international format

#T -Current Time in Long international format

#ts -Current Time in short standard format

#Ts -Current Time in long standard format

#p -Current Page Number

#P -Total Number of Pages in the Document

#f -Notebook Name

#F -Notebook Name including its Directory Path

#n -prints all text following it on a new line

For example, a header of:

Page #p || #Ds

would cause "Page 1" to be printed on the left top corner, and "01/01/96" to be printed in the upper right.

Appendix D. Data Fields Found Within the Import Files

The Processed files returned by the Budget Prep program contain the following fields, separated by spaces:

LastName
FirstName
MI
State
Year
FamilyMembersInCollege
IM
EFC
ParentAGI
StudentAGI
FisapAGI
FinancialNeed
Pell
Map
IWU_Grant
IWU_GrantWithAlumni
IWU_Grant_OS
IWU_Grant_OS_Alumni
OutsideScholorship
NoNeedScholorship
SpecialCorp_NMSC
MeritCorp_NMSC
OneTimeNMSC
AlumniAcademic
AlumniWithNeed

TalentMusic

TalentArt

TalentDrama

MusicWithNeed

ForeignStudent

Ministerial

ParentsAssoc

PreTheology

IWU_NationalMerit

Music

Giese

Alkonis

Stevenson

Rupert

Mahlstedt

Senate

Presser

Brokaw

Shanks

Baker

AlumniAchievement

StateFarmMinority

EBRust

SpecialAward

MusicAward

IWU_NationalMeritWithAlumni

InternationalStudentAchvmnt

NeedBasedGift

MusicHonors

SEOG_Initial

SEOG_Renewal

IllinoisMerit

MeritRecognition_Need

Perkins

Perkins_ST

Nursing

Nursing_ST

Stafford_IS

Stafford_OS

Stafford_IS_ST

Stafford_OS_ST

LULU

LULU_ST

EdgarSmith

Methodist

Massock

Ferguson

Tripp

Myers

IWU_StudentEmployment

IWU_EmploymentPartial

WorkStudy

WorkStudyPartial

Appendix E.

Technical Notes Concerning the "Budget Prep" Program

The following error messages may be returned during a program run.

"Unable to open X file. Program terminated."
-- This indicates that one of the 7 file locations specified in the initial screen of Budget Prep was invalid. This is most likely caused by its path (Drive and directory location) not existing. If the file is an input file, the file may also not have been found at the named location.

The program was compiled using the Borland C++ Compiler version 4.52. It was targeted for DOS, making it highly compatible with a variety of operating systems that run on IBM compatible PCs.

```
// ***************************************************************************
//  Budget Prep- a ASCII file sorting, matching, and formatting program
//               by Jeffery L. Olson
// ***************************************************************************
//
// INPUT FILES:..... BDGTOTHR.TXT- IM,FM,EFC,AGI, and Financial Need
//                   BDGT3115.TXT- Pell,MAP,IWU grants, Misc. Awards
//                   BDGT4321.TXT- Misc.Awards, SEOG, Loans, Work Study
//
// TEMPORARY FILES:  _OTHR_1.TXT- Freshman portion of BDGTOTHR.TXT
//                   _OTHR_2.TXT- Sophomore portion of BDGTOTHR.TXT
//                   _OTHR_3.TXT- Junior portion of BDGTOTHR.TXT
//                   _OTHR_4.TXT- Senior portion of BDGTOTHR.TXT
//
//                   _3115_1.TXT- Freshman portion of BDGT3115.TXT
//                   _3115_2.TXT- Sophomore portion of BDGT3115.TXT
//                   _3115_3.TXT- Junior portion of BDGT3115.TXT
//                   _3115_4.TXT- Senior portion of BDGT3115.TXT
//
//                   _4321_1.TXT- Sorted Freshman portion of BDGT4321.TXT
//                   _4321_2.TXT- Sorted Sophomore portion of BDGT4321.TXT
//                   _4321_3.TXT- Sorted Junior portion of BDGT4321.TXT
//                   _4321_4.TXT- Sorted Senior portion of BDGT4321.TXT
//
//                   _OTHR_1S.TXT- Sorted Freshman portion of BDGTOTHR.TXT
//                   _OTHR_2S.TXT- Sorted Sophomore portion of BDGTOTHR.TXT
//                   _OTHR_3S.TXT- Sorted Junior portion of BDGTOTHR.TXT
//                   _OTHR_4S.TXT- Sorted Senior portion of BDGTOTHR.TXT
//
//                   _3115_1S.TXT- Sorted Freshman portion of BDGT3115.TXT
//                   _3115_2S.TXT- Sorted Sophomore portion of BDGT3115.TXT
//                   _3115_3S.TXT- Sorted Junior portion of BDGT3115.TXT
//                   _3115_4S.TXT- Sorted Senior portion of BDGT3115.TXT
//
//                   _4321_1S.TXT- Sorted Freshman portion of BDGT4321.TXT
//                   _4321_2S.TXT- Sorted Sophomore portion of BDGT4321.TXT
//                   _4321_3S.TXT- Sorted Junior portion of BDGT4321.TXT
//                   _4321_4S.TXT- Sorted Senior portion of BDGT4321.TXT
//
// OUTPUT FILES:     FRESH.TXT- Sorted, combined data for Freshmen
//                   SOPH.TXT- Sorted, combined data for Sophomores
//                   JUNIOR.TXT- Sorted, combined data for Juniors
//                   SENIOR.TXT- Sorted, combined data for Freshmen
//
// FUNCTION:   Will first split each file into four separate files by class:
//                Freshman, Sophomores, Juniors, and Seniors.  Each of the
//                12 resulting files will then be alphabetically sorted by
//                name.  These data files will then be combined by class (ex:
//                OTHR_1.TXT, 3115_1.TXT, & 4321_1.TXT would be combined to
//                form the FRESH.TXT output file).  When the files are combined,
//                a space will be inserted between fields.  This will allow
//                QuattroPro v6.0 to automatically place the fields into
//                separate cells in the spreadsheet.
// ***************************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <fcntl.h>
#include <io.h>
#include <sys\stat.h>
#include <string.h>
#include <errno.h>
#include <conio.h>
```

```
#include "dynarray.h"

// ***************************************************************************
//  Stores the Lengths of each field in a table, they are indexed in the
//    table by their names using an enumerated type
// ***************************************************************************
const LastNameLength =                  16;
const FirstNameLength =                 12;
const MILength =                         2;
const StateLength =                      2;
const YearLength =                       2;
const FamilyMembersInCollegeLength =     2;
const IMLength =                         9;
const EFCLength =                        8;
const ParentAGILength =                  8;
const StudentAGILength =                 8;
const FisapAGILength =                   8;
const FinancialNeedLength =              9;

   // Fields Unique to BDGT3115 (ie NOT Names or Year)
const PellLength =                       5;
const MapLength =                        5;
const IWU_GrantLength =                  5;
const IWU_GrantWithAlumniLength =        5;
const IWU_Grant_OSLength =               5;
const IWU_Grant_OS_AlumniLength =        5;
const OutsideSchclorshipLength =         5;
const NoNeedScholorshipLength =          5;
const SpecialCorp_NMSCLength =           5;
const MeritCorp_NMSCLength =             5;
const OneTimeNMSCLength =                5;
const AlumniAcademicLength =             5;
const AlumniWithNeedLength =             5;
const TalentMusicLength =                5;
const TalentArtLength =                  5;
const TalentDramaLength =                5;
const MusicWithNeedLength =              5;
const ForeignStudentLength =             5;
const MinisterialLength =                5;
const ParentsAssocLength =               5;
const PreTheologyLength =                5;
const IWU_NationalMeritLength =          5;
const MusicLength =                      5;
const GieseLength =                      5;
const AlkonisLength =                    5;
const StevensonLength =                  5;
const RupertLength =                     5;
const MahlstedtLength =                  5;
const SenateLength =                     5;
const PresserLength =                    5;
const BrokawLength =                     5;
const ShanksLength =                     5;
const BakerLength =                      5;

   // Fields Unique to Bdgt4321 (ie NOT Names or Year)
const AlumniAchievementLength =          5;
const StateFarmMinorityLength =          5;
const EBRustLength =                     5;
const SpecialAwardLength =               5;
const MusicAwardLength =                 5;
const IWU_NationalMeritWithAlumniLength =5;
const InternationalStudentAchvmntLength =5;
const NeedBasedGiftLength =              5;
const MusicHonorsLength =                5;
const SEOG_InitialLength =               5;
```

```
const SEOG_RenewalLength =              5;
const IllinoisMeritLength =             5;
const MeritRecognition_NeedLength =     5;
const PerkinsLength =                   5;
const Perkins_STLength =                5;
const NursingLength =                   5;
const Nursing_STLength =                5;
const Stafford_ISLength =               5;
const Stafford_OSLength =               5;
const Stafford_IS_STLength =            5;
const Stafford_OS_STLength =            5;
const LULULength =                      5;
const LULU_STLength =                   5;
const EdgarSmithLength =                5;
const MethodistLength =                 5;
const MassockLength =                   5;
const FergusonLength =                  5;
const TrippLength =                     5;
const MyersLength =                     5;
const IWU_StudentEmploymentLength =     5;
const IWU_EmploymentPartialLength =     5;
const WorkStudyLength =                 5;
const WorkStudyPartialLength =          5;
const CRLength =                        1;


// ***************************************************************************
//  Structure of the data fields in the BdgtOthrFile
// ***************************************************************************

typedef struct BdgtOthr_Form
{
   char  LastName                       [LastNameLength],
         FirstName                      [FirstNameLength],
         MI                             [MILength],
         State                          [StateLength],
         Year                           [YearLength],
         FamilyMembersInCollege         [FamilyMembersInCollegeLength],
         IM                             [IMLength],
         EFC                            [EFCLength],
         ParentAGI                      [ParentAGILength],
         StudentAGI                     [StudentAGILength],
         FisapAGI                       [FisapAGILength],
         FinancialNeed                  [FinancialNeedLength],
         CR                             [CRLength];
                                //  Total: 87
};


// ***************************************************************************
//  Structure of the data fields in the Bdgt3115.TXT File
// ***************************************************************************

typedef struct Bdgt3115_Form
{
   char  LastName                       [LastNameLength],
         FirstName                      [FirstNameLength],
         MI                             [MILength],
         Year                           [YearLength],
         Pell                           [PellLength],
         Map                            [MapLength],
         IWU_Grant                      [IWU_GrantLength],
         IWU_GrantWithAlumni            [IWU_GrantWithAlumniLength],
         IWU_Grant_OS                   [IWU_Grant_OSLength],
         IWU_Grant_OS_Alumni            [IWU_Grant_OS_AlumniLength],
```

```
        OutsideScholorship              [OutsideScholorshipLength],
        NoNeedScholorship               [NoNeedScholorshipLength],
        SpecialCorp_NMSC                [SpecialCorp_NMSCLength],
        MeritCorp_NMSC                  [MeritCorp_NMSCLength],
        OneTimeNMSC                     [OneTimeNMSCLength],
        AlumniAcademic                  [AlumniAcademicLength],
        AlumniWithNeed                  [AlumniWithNeedLength],
        TalentMusic                     [TalentMusicLength],
        TalentArt                       [TalentArtLength],
        TalentDrama                     [TalentDramaLength],
        MusicWithNeed                   [MusicWithNeedLength],
        ForeignStudent                  [ForeignStudentLength],
        Ministerial                     [MinisterialLength],
        ParentsAssoc                    [ParentsAssocLength],
        PreTheology                     [PreTheologyLength],
        IWU_NationalMerit               [IWU_NationalMeritLength],
        Music                           [MusicLength],
        Giese                           [GieseLength],
        Alkonis                         [AlkonisLength],
        Stevenson                       [StevensonLength],
        Rupert                          [RupertLength],
        Mahlstedt                       [MahlstedtLength],
        Senate                          [SenateLength],
        Presser                         [PresserLength],
        Brokaw                          [BrokawLength],
        Shanks                          [ShanksLength],
        Baker                           [BakerLength],
        CR                              [CRLength];
                            // Total: 198
};

// ****************************************************************************
//  Structure of the data fields in the Bdgt4321.TXT File
// ****************************************************************************

typedef struct Bdgt4321_Form
{
   char  LastName                       [LastNameLength],
         FirstName                      [FirstNameLength],
         MI                             [MILength],
         Year                           [YearLength],
         AlumniAchievement              [AlumniAchievementLength],
         StateFarmMinority              [StateFarmMinorityLength],
         EBRust                         [EBRustLength],
         SpecialAward                   [SpecialAwardLength],
         MusicAward                     [MusicAwardLength],
         IWU_NationalMeritWithAlumni    [IWU_NationalMeritWithAlumniLength],
         InternationalStudentAchvmnt    [InternationalStudentAchvmntLength],
         NeedBasedGift                  [NeedBasedGiftLength],
         MusicHonors                    [MusicHonorsLength],
         SEOG_Initial                   [SEOG_InitialLength],
         SEOG_Renewal                   [SEOG_RenewalLength],
         IllinoisMerit                  [IllinoisMeritLength],
         MeritRecognition_Need          [MeritRecognition_NeedLength],
         Perkins                        [PerkinsLength],
         Perkins_ST                     [Perkins_STLength],
         Nursing                        [NursingLength],
         Nursing_ST                     [Nursing_STLength],
         Stafford_IS                    [Stafford_ISLength],
         Stafford_OS                    [Stafford_OSLength],
         Stafford_IS_ST                 [Stafford_OSLength],
         Stafford_OS_ST                 [Stafford_OS_STLength],
         LULU                           [LULULength],
         LULU_ST                        [LULU_STLength],
         EdgarSmith                     [EdgarSmithLength],
```

```
        Methodist                       [MethodistLength],
        Massock                         [MassockLength],
        Ferguson                        [FergusonLength],
        Tripp                           [TrippLength],
        Myers                           [MyersLength],
        IWU_StudentEmployment           [IWU_StudentEmploymentLength],
        IWU_EmploymentPartial           [IWU_EmploymentPartialLength],
        WorkStudy                       [WorkStudyLength],
        WorkStudyPartial                [WorkStudyPartialLength],
        CR                              [CRLength];
                            // Total: 198
};



// ****************************************************************************
// These unions allow each line of a file to be accessed as a continuous
//    string or by field
// ****************************************************************************
enum FileLengths { BdgtOthr_Len = 87, Bdgt3115_Len = 198, Bdgt4321_Len = 198};


union BdgtOthr_Type
{
   char           All[BdgtOthr_Len];
   BdgtOthr_Form  Field;
};


union Bdgt3115_Type
{
   char           All[Bdgt3115_Len];
   Bdgt3115_Form  Field;
};

union Bdgt4321_Type
{
   char           All[Bdgt4321_Len];
   Bdgt4321_Form  Field;
};

typedef struct BdgtOthrNode* BdgtOthrNodePtr;

typedef struct BdgtOthrNode
{
   BdgtOthr_Type Data;
};

typedef struct Bdgt3115Node* Bdgt3115NodePtr;

typedef struct Bdgt3115Node
{
   Bdgt3115_Type Data;
};

typedef struct Bdgt4321Node* Bdgt4321NodePtr;

typedef struct Bdgt4321Node
{
   Bdgt4321_Type Data;
};

// ****************************************************************************
//  Default locations of the files
// ****************************************************************************
```

```
// Input Files
char BdgtOthr_File[80] = "A:\\BDGTOTHR.TXT";
char Bdgt3115_File[80] = "A:\\BDGT3115.TXT";
char Bdgt4321_File[80] = "A:\\BDGT4321.TXT";
// Temporary Files
char* FreshOthrFile   = "C:\\_OTHR_1.TXT";
char* SophOthrFile    = "C:\\_OTHR_2.TXT";
char* JuniorOthrFile  = "C:\\_OTHR_3.TXT";
char* SeniorOthrFile  = "C:\\_OTHR_4.TXT";

char* SortedFreshOthrFile   = "C:\\_OTHR_1S.TXT";
char* SortedSophOthrFile    = "C:\\_OTHR_2S.TXT";
char* SortedJuniorOthrFile  = "C:\\_OTHR_3S.TXT";
char* SortedSeniorOthrFile  = "C:\\_OTHR_4S.TXT";

char* Fresh3115File  = "C:\\_3115_1.TXT";
char* Soph3115File   = "C:\\_3115_2.TXT";
char* Junior3115File = "C:\\_3115_3.TXT";
char* Senior3115File = "C:\\_3115_4.TXT";

char* SortedFresh3115File  = "C:\\_3115_1S.TXT";
char* SortedSoph3115File   = "C:\\_3115_2S.TXT";
char* SortedJunior3115File = "C:\\_3115_3S.TXT";
char* SortedSenior3115File = "C:\\_3115_4S.TXT";

char* Fresh4321File  = "C:\\_4321_1.TXT";
char* Soph4321File   = "C:\\_4321_2.TXT";
char* Junior4321File = "C:\\_4321_3.TXT";
char* Senior4321File = "C:\\_4321_4.TXT";

char* SortedFresh4321File  = "C:\\_4321_1S.TXT";
char* SortedSoph4321File   = "C:\\_4321_2S.TXT";
char* SortedJunior4321File = "C:\\_4321_3S.TXT";
char* SortedSenior4321File = "C:\\_4321_4S.TXT";

//Output Files
char FreshOutputFile[80]   = "C:\\OFFICE\\QPW\\FILES\\ASCII\\FRESH.TXT";
char SophOutputFile[80]    = "C:\\OFFICE\\QPW\\FILES\\ASCII\\SOPH.TXT";
char JuniorOutputFile[80]  = "C:\\OFFICE\\QPW\\FILES\\ASCII\\JUNIOR.TXT";
char SeniorOutputFile[80]  = "C:\\OFFICE\\QPW\\FILES\\ASCII\\SENIOR.TXT";


// Handles the Open, Write, and Close functions use to access files
enum FileIndexes { Freshman, Sophomore, Junior, Senior, Input };

int HandleOthr[5];
int Handle3115[5];
int Handle4321[5];
int HandleOutput[4];

int HandleSortedOthr[4];
int HandleSorted3115[4];
int HandleSorted4321[4];

int HandleTempOutput;

// Set up buffers for file I/O
BdgtOthr_Type  BdgtOthr_Buf;
Bdgt3115_Type  Bdgt3115_Buf;
Bdgt4321_Type  Bdgt4321_Buf;

BdgtOthr_Type  BdgtOthr_OutBuf;
Bdgt3115_Type  Bdgt3115_OutBuf;
Bdgt4321_Type  Bdgt4321_OutBuf;
```

```
//: Define three dynamically allocated heaps using the DynamicArray defined in
//   dynarray.h

DynArray HeapOthr;
DynArray Heap3115;
DynArray Heap4321;

int IndexOthr;
int Index3115;
int Index4321;

char CurrentLast[LastNameLength+1];
char CurrentFirst[FirstNameLength+1];

char TempLast[LastNameLength+1];
char TempFirst[FirstNameLength+1];

char LastOthr[LastNameLength+1];
char FirstOthr[FirstNameLength+1];

char Last3115[LastNameLength+1];
char First3115[FirstNameLength+1];

char Last4321[LastNameLength+1];
char First4321[FirstNameLength+1];

int i,j, Class, Choice;  //counting variables for loop structures
int More3115, More4321;

int ErrorCode = 0;

// ***************************************************************************
//                            Function Prototypes
// ***************************************************************************
int GetClass (int &, char*, char*);

// ***************************************************************************
//                                 Main Program
// ***************************************************************************
void main()
{
   // These are the pointers used in the Dynamic array
   BdgtOthrNodePtr TempOthrPtr;
   Bdgt3115NodePtr Temp3115Ptr;
   Bdgt4321NodePtr Temp4321Ptr;

   char Blank3115[Bdgt3115_Len];  // used to set the file buffer to all blanks
   char Blank4321[Bdgt4321_Len];

   clrscr();
   cout << "Welcome to automated ASCII file processing for Budget Projection!";
   cout << endl << endl;
   cout << "In order to proceed, this program needs to know the  " << endl;
   cout << "location of the input files.  Additionally, this program " << endl;
   cout << "needs to know where it should put the processed output " << endl;
   cout << "files. " << endl << endl;
   Choice = 0;

   while(Choice != 8)
   {
      cout << "Current File Locations:" << endl;
      cout << "1. (Input) BDGTOTHR.TXT ----> " << BdgtOthr_File << endl;
      cout << "2. (Input) BDGT3115.TXT ----> " << Bdgt3115_File << endl;
```

```
cout << "3. (Input) BDGT4321.TXT ----> " << Bdgt4321_File << endl;
cout << "4. (Output) FRESH.TXT ------> " << FreshOutputFile << endl;
cout << "5. (Output) SOPH.TXT -------> " << SophOutputFile << endl;
cout << "6. (Output) JUNIOR.TXT -----> " << JuniorOutputFile << endl;
cout << "7. (Output) SENIOR.TXT -----> " << SeniorOutputFile << endl;
cout << endl;
cout << "Enter 1-7 to modify a file location, 8 to continue: ";
cin >> Choice;

clrscr();

switch (Choice)
{
   case 1   : {
                  cout << "Old Location : " << BdgtOthr_File << endl;
                  while (1)
                  {
                     cout << "Enter New Location : ";
                     cin >> BdgtOthr_File;
                     strupr(BdgtOthr_File);
                     ErrorCode = open(BdgtOthr_File, O_RDONLY);
                     if (ErrorCode < 0)
                        cout << "Invalid File Name"<< endl;
                     else
                     {
                        close(ErrorCode);
                        break;
                     }
                  }
                     clrscr();
              } break;
   case 2   : {
                  cout << "Old Location : " << Bdgt3115_File << endl;
                  while (1)
                  {
                     cout << "Enter New Location : ";
                     cin >> Bdgt3115_File;
                     strupr(Bdgt3115_File);
                     ErrorCode = open(Bdgt3115_File, O_RDONLY);
                     if (ErrorCode < 0)
                        cout << "Invalid File Name" << endl;
                     else
                     {
                        close(ErrorCode);
                        break;
                     }
                  }
                     clrscr();
              } break;
   case 3   : {
                  cout << "Old Location : " << Bdgt4321_File << endl;
                  while (1)
                  {
                     cout << "Enter New Location : ";
                     cin >> Bdgt4321_File;
                     strupr(Bdgt4321_File);
                     ErrorCode = open(Bdgt4321_File, O_RDONLY);
                     if (ErrorCode < 0)
                        cout << "Invalid File Name" << endl;
                     else
                     {
                        close(ErrorCode);
                        break;
                     }
                  }
```

```
                    clrscr();
                } break;
        case 4  : {
                    cout << "Old Location : " << FreshOutputFile << endl;
                    cout << "Enter New Location : ";
                    cin >> FreshOutputFile;
                    strupr(FreshOutputFile);
                    clrscr();
                } break;
        case 5  : {
                    cout << "Old Location : " << SophOutputFile << endl;
                    cout << "Enter New Location : ";
                    cin >> SophOutputFile;
                    strupr(SophOutputFile);
                    clrscr();
                } break;
        case 6  : {
                    cout << "Old Location : " << JuniorOutputFile << endl;
                    cout << "Enter New Location : ";
                    cin >> JuniorOutputFile;
                    strupr(JuniorOutputFile);
                    clrscr();
                } break;
        case 7  : {
                    cout << "Old Location : " << SeniorOutputFile << endl;
                    cout << "Enter New Location : ";
                    cin >> SeniorOutputFile;
                    strupr(SeniorOutputFile);
                    clrscr();
                } break;


    }// switch(Choice)

}// while
cout << "Beginning file processing." << endl;

// ***************************************************************************
//  Split the Othr file by class
// ***************************************************************************
HandleOthr[Input] = open(BdgtOthr_File, O_RDONLY | O_TEXT);

HandleOthr[Freshman] = open(FreshOthrFile, O_RDWR | O_CREAT | O_TRUNC |
                       O_APPEND, S_IREAD | S_IWRITE);
HandleOthr[Sophomore] = open(SophOthrFile, O_RDWR | O_CREAT | O_TRUNC |
                       O_APPEND,  S_IREAD | S_IWRITE);
HandleOthr[Junior] = open(JuniorOthrFile, O_RDWR | O_CREAT | O_TRUNC  |
                       O_APPEND, S_IREAD | S_IWRITE);
HandleOthr[Senior] = open(SeniorOthrFile, O_RDWR | O_CREAT | O_TRUNC  |
                       O_APPEND, S_IREAD | S_IWRITE);

for (i = Freshman; i <= Input; i++)
{
   if (HandleOthr[i] < 0)
   {
      cout << "Unable to open OTHR files, program halted";
      exit(1);
   }
}

Class = -1;
while ( read(HandleOthr[Input], BdgtOthr_Buf.All, BdgtOthr_Len))
{
   if ( BdgtOthr_Buf.Field.Year[0] == 'F' &&
        BdgtOthr_Buf.Field.Year[1] == 'R' )
```

```
      Class = Freshman;

   else if ( BdgtOthr_Buf.Field.Year[0] == 'S' &&
             BdgtOthr_Buf.Field.Year[1] == 'O' )
          Class = Sophomore;

        else if ( BdgtOthr_Buf.Field.Year[0] == 'J' &&
                  BdgtOthr_Buf.Field.Year[1] == 'R' )
               Class = Junior;

             else if ( BdgtOthr_Buf.Field.Year[0] == 'S' &&
                       BdgtOthr_Buf.Field.Year[1] == 'R' )
                  Class = Senior;

                else
                {
                   strncpy(CurrentLast, BdgtOthr_Buf.Field.LastName,
                                                    LastNameLength);
                   strncpy(CurrentFirst, BdgtOthr_Buf.Field.FirstName,
                                                    FirstNameLength);
                   GetClass(Class, CurrentLast, CurrentFirst);
                   switch (Class)
                   {
                      case Freshman  : {
                                        BdgtOthr_Buf.Field.Year[0] = 'F';
                                        BdgtOthr_Buf.Field.Year[1] = 'R';
                                       } break;
                      case Sophomore : {
                                        BdgtOthr_Buf.Field.Year[0] = 'S';
                                        BdgtOthr_Buf.Field.Year[1] = 'O';
                                       } break;
                      case Junior    : {
                                        BdgtOthr_Buf.Field.Year[0] = 'J';
                                        BdgtOthr_Buf.Field.Year[1] = 'R';
                                       } break;
                      case Senior    : {
                                        BdgtOthr_Buf.Field.Year[0] = 'S';
                                        BdgtOthr_Buf.Field.Year[1] = 'R';
                                       } break;
                   }

                }// end else
                         // If Freshman..Senior add it to the corrent file
   if (Class >= Freshman && Class <= Senior)
      write(HandleOthr[Class], BdgtOthr_Buf.All, BdgtOthr_Len);

}


for (i=Freshman; i <= Input; i++)     // closes the files that were processed
   close( HandleOthr[i]);

cout << BdgtOthr_File << " split into four classes." << endl;

// ************************************************************
//  Split 3115 File by class
// ************************************************************
Handle3115[Input] = open(Bdgt3115_File, O_RDONLY | O_TEXT);

Handle3115[Freshman] = open(Fresh3115File, O_RDWR | O_CREAT | O_TRUNC |
                        O_APPEND, S_IREAD | S_IWRITE);
Handle3115[Sophomore] = open(Soph3115File, O_RDWR | O_CREAT | O_TRUNC |
                        O_APPEND, S_IREAD | S_IWRITE);
Handle3115[Junior] = open(Junior3115File, O_RDWR | O_CREAT | O_TRUNC  |
                        O_APPEND, S_IREAD | S_IWRITE);
```

```
Handle3115[Senior] = open(Senior3115File, O_RDWR | O_CREAT | O_TRUNC  |
                          O_APPEND, S_IREAD | S_IWRITE);

for (i = Freshman; i <= Input; i++)
{
   if (Handle3115[i] < 0)
   {
      cout << "Unable to open 3115 files. Program Halted";
      exit(1);
   }
}

Class = -1;
while ( read(Handle3115[Input], Bdgt3115_Buf.All, Bdgt3115_Len))
{
   if ( Bdgt3115_Buf.Field.Year[0] == 'F' &&
        Bdgt3115_Buf.Field.Year[1] == 'R' )
      Class = Freshman;

   else if ( Bdgt3115_Buf.Field.Year[0] == 'S' &&
             Bdgt3115_Buf.Field.Year[1] == 'O' )
          Class = Sophomore;

        else if ( Bdgt3115_Buf.Field.Year[0] == 'J' &&
                  Bdgt3115_Buf.Field.Year[1] == 'R' )
                Class = Junior;

             else if ( Bdgt3115_Buf.Field.Year[0] == 'S' &&
                       Bdgt3115_Buf.Field.Year[1] == 'R' )
                    Class = Senior;

                else
                {
                   strncpy(CurrentLast, Bdgt3115_Buf.Field.LastName,
                                                      LastNameLength);
                   strncpy(CurrentFirst, Bdgt3115_Buf.Field.FirstName,
                                                      FirstNameLength);
                   GetClass(Class, CurrentLast, CurrentFirst);
                   switch (Class)
                   {
                      case Freshman  : {
                                        Bdgt3115_Buf.Field.Year[0] = 'F';
                                        Bdgt3115_Buf.Field.Year[1] = 'R';
                                       } break;
                      case Sophomore : {
                                        Bdgt3115_Buf.Field.Year[0] = 'S';
                                        Bdgt3115_Buf.Field.Year[1] = 'O';
                                       } break;
                      case Junior    : {
                                        Bdgt3115_Buf.Field.Year[0] = 'J';
                                        Bdgt3115_Buf.Field.Year[1] = 'R';
                                       } break;
                      case Senior    : {
                                        Bdgt3115_Buf.Field.Year[0] = 'S';
                                        Bdgt3115_Buf.Field.Year[1] = 'R';
                                       } break;
                   }

                }// end else
                        // If Freshman..Senior add it to the corrent file
   if (Class >= Freshman && Class <= Senior)
      write(Handle3115[Class], Bdgt3115_Buf.All, Bdgt3115_Len);

}
```

```
   for (i=Freshman; i <= Input; i++)     // closes the files that were processed
      close( Handle3115[i]);

   cout << Bdgt3115_File << " split into four classes." << endl;

// ************************************************************
//  Split 4321 file by class
// ************************************************************
   Handle4321[Input] = open(Bdgt4321_File, O_RDONLY | O_TEXT);

   Handle4321[Freshman] = open(Fresh4321File, O_RDWR | O_CREAT | O_TRUNC |
                          O_APPEND, S_IREAD | S_IWRITE);
   Handle4321[Sophomore] = open(Soph4321File, O_RDWR | O_CREAT | O_TRUNC |
                          O_APPEND, S_IREAD | S_IWRITE);
   Handle4321[Junior] = open(Junior4321File, O_RDWR | O_CREAT | O_TRUNC |
                          O_APPEND, S_IREAD | S_IWRITE);
   Handle4321[Senior] = open(Senior4321File, O_RDWR | O_CREAT | O_TRUNC |
                          O_APPEND, S_IREAD | S_IWRITE);

   for (i = Freshman; i <= Input; i++)
   {
      if (Handle4321[i] < 0)
      {
         cout << "Unable to open 4321 files. Program halted.";
         exit(1);
      }
   }

   Class = -1;
   while ( read(Handle4321[Input], Bdgt4321_Buf.All, Bdgt4321_Len))
   {
      if ( Bdgt4321_Buf.Field.Year[0] == 'F' &&
           Bdgt4321_Buf.Field.Year[1] == 'R' )
         Class = Freshman;

      else if ( Bdgt4321_Buf.Field.Year[0] == 'S' &&
                Bdgt4321_Buf.Field.Year[1] == 'O' )
              Class = Sophomore;

           else if ( Bdgt4321_Buf.Field.Year[0] == 'J' &&
                     Bdgt4321_Buf.Field.Year[1] == 'R' )
                   Class = Junior;

                else if ( Bdgt4321_Buf.Field.Year[0] == 'S' &&
                          Bdgt4321_Buf.Field.Year[1] == 'R' )
                        Class = Senior;

                     else
                     {
                        strncpy(CurrentLast, Bdgt4321_Buf.Field.LastName,
                                                        LastNameLength);
                        strncpy(CurrentFirst, Bdgt4321_Buf.Field.FirstName,
                                                        FirstNameLength);
                        GetClass(Class, CurrentLast, CurrentFirst);
                        switch (Class)
                        {
                           case Freshman  : {
                                             Bdgt4321_Buf.Field.Year[0] = 'F';
                                             Bdgt4321_Buf.Field.Year[1] = 'R';
                                            } break;
                           case Sophomore : {
                                             Bdgt4321_Buf.Field.Year[0] = 'S';
                                             Bdgt4321_Buf.Field.Year[1] = 'O';
                                            } break;
```

```
                        case Junior    : {
                                          Bdgt4321_Buf.Field.Year[0] = 'J';
                                          Bdgt4321_Buf.Field.Year[1] = 'R';
                                         } break;
                        case Senior    : {
                                          Bdgt4321_Buf.Field.Year[0] = 'S';
                                          Bdgt4321_Buf.Field.Year[1] = 'R';
                                         } break;
                     }

                  }// end else
                        // If Freshman..Senior add it to the corrent file
      if (Class >= Freshman && Class <= Senior)
         write(Handle4321[Class], Bdgt4321_Buf.All, Bdgt4321_Len);

   }

   for (i=Freshman; i <= Input; i++)    // closes the files that were processed
      close( Handle4321[i]);

   cout << Bdgt4321_File << " split into four classes." << endl << endl;

// ***************************************************************
//   Sort BdgtOthr Files
// ***************************************************************

   HandleOthr[Freshman] = open(FreshOthrFile, O_RDONLY | O_TEXT);
   HandleOthr[Sophomore] = open(SophOthrFile, O_RDONLY | O_TEXT);
   HandleOthr[Junior] = open(JuniorOthrFile, O_RDONLY | O_TEXT);
   HandleOthr[Senior] = open(SeniorOthrFile, O_RDONLY | O_TEXT);

   HandleSortedOthr[Freshman] = open(SortedFreshOthrFile, O_RDWR | O_CREAT |
                           O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
   HandleSortedOthr[Sophomore] = open(SortedSophOthrFile, O_RDWR | O_CREAT |
                           O_TRUNC | O_APPEND,  S_IREAD | S_IWRITE);
   HandleSortedOthr[Junior] = open(SortedJuniorOthrFile, O_RDWR | O_CREAT |
                           O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
   HandleSortedOthr[Senior] = open(SortedSeniorOthrFile, O_RDWR | O_CREAT |
                           O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);


   for (Class = Freshman; Class <= Senior; Class++)
   {

   // ************************************************
   //  Read the current file into a heap
   // ************************************************

   //long lseek(int handle, long offset, int fromwhere);

      //Put the file ptr at start of file
      //lseek(HandleOthr[Class], 0L, SEEK_SET);

      // Keep Reading in records from the current file
      while (read(HandleOthr[Class], BdgtOthr_Buf.All, BdgtOthr_Len))
      {
         // Determine the proper place to add the record in the heap
          TempOthrPtr = (BdgtOthrNodePtr)malloc(sizeof(struct BdgtOthrNode));

          //Copy file info into a new node  to be added to the heap
          strncpy(TempOthrPtr->Data.All, BdgtOthr_Buf.All, BdgtOthr_Len);

          strncpy (CurrentLast, BdgtOthr_Buf.Field.LastName, LastNameLength);
          //CurrentLast[LastNameLength] = '\0';
```

```
      strncpy (CurrentFirst, BdgtOthr_Buf.Field.FirstName, FirstNameLength);
      //CurrentFirst[FirstNameLength+1] = '\0';

      IndexOthr = ( HeapOthr.count() ) + 1;

      while (!(IndexOthr == 1 ||
               //Below performs:  NewLastName > OldLastName
               ((strcmp(CurrentLast,
                        strncpy(TempLast,
                                ((BdgtOthrNodePtr)HeapOthr[IndexOthr / 2])->
                                Data.Field.LastName,
                                LastNameLength))                    < 0) ||

               ((strcmp(CurrentLast,
                        strncpy(TempLast,
                                ((BdgtOthrNodePtr)HeapOthr[IndexOthr / 2])->
                                Data.Field.LastName,
                                LastNameLength))       == 0)  &&
               (strcmp(CurrentFirst,
                        strncpy(TempFirst,
                                ((BdgtOthrNodePtr)HeapOthr[IndexOthr / 2])->
                                Data.Field.FirstName,
                                FirstNameLength))       <=0) )      ) ) )


            // terminate when the root is reached or the element
            // is in its correct place
         {
            // check the next lower level of the heap
            HeapOthr.assign(IndexOthr, HeapOthr[IndexOthr/2]);
            IndexOthr /= 2;
         }
      // end while(heap sort)
      HeapOthr.assign(IndexOthr, TempOthrPtr);

   }// end while(read files)

//  ************************************************
//   Sort the Heap
//  ************************************************
   int parent, child;
   BdgtOthrNodePtr ItemOthrPtr;

   for (IndexOthr = HeapOthr.count() ; IndexOthr > 0; IndexOthr --)

   {

      /* delete element with the highest key from the heap */
      /* save value of the element with the highest key */
      ItemOthrPtr = (BdgtOthrNodePtr)HeapOthr[1];
      TempOthrPtr = (BdgtOthrNodePtr)HeapOthr[IndexOthr];
                        /* use last element in heap to adjust heap */
      parent=1;
      child=2;

      while (child <= IndexOthr)
      {
         if (child < IndexOthr)
         /* find the largest child of the current parent */
            if ((strcmp( strncpy(CurrentLast,
                              ((BdgtOthrNodePtr)HeapOthr[child])->
                                   Data.Field.LastName,
                              LastNameLength),
                         strncpy(TempLast,
                              ((BdgtOthrNodePtr)HeapOthr[child + 1])->
```

```
                              Data.Field.LastName,
                         LastNameLength))            < 0)

        || ((strcmp( strncpy(CurrentLast,
                         ((BdgtOthrNodePtr)HeapOthr[child])->
                              Data.Field.LastName,
                         LastNameLength),
                    strncpy(TempLast,
                         ((BdgtOthrNodePtr)HeapOthr[child + 1])->
                              Data.Field.LastName,
                         LastNameLength))            == 0)     &&

        (strcmp( strncpy(CurrentFirst,
                         ((BdgtOthrNodePtr)HeapOthr[child])->
                              Data.Field.FirstName,
                         FirstNameLength),
                    strncpy(TempFirst,
                         ((BdgtOthrNodePtr)HeapOthr[child + 1])->
                              Data.Field.FirstName,
                         FirstNameLength))            < 0)) )


           child++;

      if ((strcmp( strncpy(CurrentLast,
                         TempOthrPtr -> Data.Field.LastName,
                         LastNameLength),
                    strncpy(TempLast,
                         ((BdgtOthrNodePtr)HeapOthr[child])->
                              Data.Field.LastName,
                         LastNameLength))            > 0)
      // if the current name is greater than the one in the heap
      // then this is the correct position to add it.... OR

       || ((strcmp( strncpy(CurrentLast,
                         TempOthrPtr -> Data.Field.LastName,
                         LastNameLength),
                    strncpy(TempLast,
                         ((BdgtOthrNodePtr)HeapOthr[child])->
                              Data.Field.LastName,
                         LastNameLength))            ==0 ) &&

              (strcmp( strncpy(CurrentFirst,
                         TempOthrPtr -> Data.Field.FirstName,
                         FirstNameLength),
                    strncpy(TempFirst,
                         ((BdgtOthrNodePtr)HeapOthr[child])->
                              Data.Field.FirstName,
                         FirstNameLength))           >=0)))

     // if the last names are the same, but this first name is
     // grater than or equal to the one in the heap... this is
     // the correct position to add it so...

          break;
        else
        {
        // move to the next lower level
           HeapOthr.assign(parent, HeapOthr[child]);
           parent = child;
           child *= 2;
        }
    } // end while
HeapOthr.assign(parent, TempOthrPtr);
HeapOthr.assign(IndexOthr, ItemOthrPtr); // Place the sorted Item at the
```

```
                                                    // end of the heap; don't sort
                                                    // this space on next pass
   } // end for each item sort

   IndexOthr = HeapOthr.count();

   // write out sorted file & empty the heap as you go
   for ( j=1; j<=IndexOthr; j++)
   {
      write (HandleSortedOthr[Class],
            ((BdgtOthrNodePtr)HeapOthr[j]) -> Data.All,
            BdgtOthr_Len);

      delete (BdgtOthrNodePtr)HeapOthr[j];  // free memory at heap address
      HeapOthr.remove(j);                   // remove its poiter from heap

   }

}// end for each class of BgdtOthr file

for (i = Freshman; i <= Senior; i++)
{
   close(HandleOthr[i]);
   close(HandleSortedOthr[i]);
}

cout << "BdgtOthr files sorted by name." << endl;

// ***************************************************************
//   Sort Bdgt3115 Files
// ***************************************************************

Handle3115[Freshman] = open(Fresh3115File, O_RDONLY | O_TEXT);
Handle3115[Sophomore] = open(Soph3115File, O_RDONLY | O_TEXT);
Handle3115[Junior] = open(Junior3115File, O_RDONLY | O_TEXT);
Handle3115[Senior] = open(Senior3115File, O_RDONLY | O_TEXT);

HandleSorted3115[Freshman] = open(SortedFresh3115File, O_RDWR | O_CREAT |
                        O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
HandleSorted3115[Sophomore] = open(SortedSoph3115File, O_RDWR | O_CREAT |
                        O_TRUNC | O_APPEND,  S_IREAD | S_IWRITE);
HandleSorted3115[Junior] = open(SortedJunior3115File, O_RDWR | O_CREAT |
                        O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
HandleSorted3115[Senior] = open(SortedSenior3115File, O_RDWR | O_CREAT |
                        O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);


for (Class = Freshman; Class <= Senior; Class++)
{

// ************************************************
//  Read the current file into a  3115 heap
// ************************************************


   // Keep Reading in records from the current file
   while (read(Handle3115[Class], Bdgt3115_Buf.All, Bdgt3115_Len))
   {
      // Determine the proper place to add the record in the heap
       Temp3115Ptr = (Bdgt3115NodePtr)malloc(sizeof(struct Bdgt3115Node));

       //Copy file info into a new node  to be added to the heap
       strncpy(Temp3115Ptr->Data.All, Bdgt3115_Buf.All, Bdgt3115_Len);

       strncpy (CurrentLast, Bdgt3115_Buf.Field.LastName, LastNameLength);
```

```
        //CurrentLast[LastNameLength] = '\0';

        strncpy (CurrentFirst, Bdgt3115_Buf.Field.FirstName, FirstNameLength);
        //CurrentFirst[FirstNameLength+1] = '\0';

        Index3115 = ( Heap3115.count() ) + 1;

        while (!(Index3115 == 1 ||
                 //Below performs:  NewLastName > OldLastName
                 ((strcmp(CurrentLast,
                          strncpy(TempLast,
                                  ((Bdgt3115NodePtr)Heap3115[Index3115 / 2])->
                                  Data.Field.LastName,
                                  LastNameLength))                < 0) ||

                 ((strcmp(CurrentLast,
                          strncpy(TempLast,
                                  ((Bdgt3115NodePtr)Heap3115[Index3115 / 2])->
                                  Data.Field.LastName,
                                  LastNameLength))       == 0)   &&
                 (strcmp(CurrentFirst,
                          strncpy(TempFirst,
                                  ((Bdgt3115NodePtr)Heap3115[Index3115 / 2])->
                                  Data.Field.FirstName,
                                  FirstNameLength))      <=0) )      ) ) )


                 // terminate when the root is reached or the element
                 // is in its correct place
             {
                 // check the next lower level of the heap
                 Heap3115.assign(Index3115, Heap3115[Index3115/2]);
                 Index3115 /= 2;
             }
        // end while(heap sort)
        Heap3115.assign(Index3115, Temp3115Ptr);

    }// end while(read files)

// ************************************************
//  Sort the Heap3115
// ************************************************
    int parent, child;
    Bdgt3115NodePtr Item3115Ptr;

    for (Index3115 = Heap3115.count() ; Index3115 > 0; Index3115 --)

    {

        /* delete element with the highest key from the heap */
        /* save value of the element with the highest key */
        Item3115Ptr = (Bdgt3115NodePtr)Heap3115[1];
        Temp3115Ptr = (Bdgt3115NodePtr)Heap3115[Index3115];
                        /* use last element in heap to adjust heap */
        parent=1;
        child=2;

        while (child <= Index3115)
        {
            if (child < Index3115)
            /* find the largest child of the current parent */
                if ((strcmp( strncpy(CurrentLast,
                                ((Bdgt3115NodePtr)Heap3115[child])->
                                     Data.Field.LastName,
                                LastNameLength),
```

```
                strncpy(TempLast,
                     ((Bdgt3115NodePtr)Heap3115[child + 1])->
                          Data.Field.LastName,
                     LastNameLength))            < 0)

      || ((strcmp( strncpy(CurrentLast,
                     ((Bdgt3115NodePtr)Heap3115[child])->
                          Data.Field.LastName,
                     LastNameLength),
                strncpy(TempLast,
                     ((Bdgt3115NodePtr)Heap3115[child + 1])->
                          Data.Field.LastName,
                     LastNameLength))            == 0)     &&

      (strcmp( strncpy(CurrentFirst,
                     ((Bdgt3115NodePtr)Heap3115[child])->
                          Data.Field.FirstName,
                     FirstNameLength),
                strncpy(TempFirst,
                     ((Bdgt3115NodePtr)Heap3115[child + 1])->
                          Data.Field.FirstName,
                     FirstNameLength))            < 0)) )


        child++;

   if ((strcmp( strncpy(CurrentLast,
                     Temp3115Ptr -> Data.Field.LastName,
                     LastNameLength),
               strncpy(TempLast,
                     ((Bdgt3115NodePtr)Heap3115[child])->
                          Data.Field.LastName,
                     LastNameLength))            > 0)
    // if the current name is greater than the one in the heap
    // then this is the correct position to add it.... OR

     || ((strcmp( strncpy(CurrentLast,
                     Temp3115Ptr -> Data.Field.LastName,
                     LastNameLength),
               strncpy(TempLast,
                     ((Bdgt3115NodePtr)Heap3115[child])->
                          Data.Field.LastName,
                     LastNameLength))                  ==0 ) &&

           (strcmp( strncpy(CurrentFirst,
                     Temp3115Ptr -> Data.Field.FirstName,
                     FirstNameLength),
               strncpy(TempFirst,
                     ((Bdgt3115NodePtr)Heap3115[child])->
                          Data.Field.FirstName,
                     FirstNameLength))            >=0)))

  // if the last names are the same, but this first name is
  // grater than or equal to the one in the heap... this is
  // the correct position to add it so...

      break;
    else
    {
    // move to the next lower level
       Heap3115.assign(parent, Heap3115[child]);
       parent = child;
       child *= 2;
    }
} // end while
```

```
      Heap3115.assign(parent, Temp3115Ptr);
      Heap3115.assign(Index3115, Item3115Ptr); // Place the sorted Item at the
                                               // end of the heap; don't sort
                                               // this space on next pass
    } // end for each item sort

    Index3115 = Heap3115.count();

    // write out sorted file & empty the heap as you go
    for ( j=1; j<=Index3115; j++)
    {
       write (HandleSorted3115[Class],
             ((Bdgt3115NodePtr)Heap3115[j]) -> Data.All,
             Bdgt3115_Len);

       delete (Bdgt3115NodePtr)Heap3115[j];  // free memory at heap address
       Heap3115.remove(j);                   // remove its poiter from heap

    }

  }// end for each class of Bgdt3115 file

  for (i = Freshman; i <= Senior; i++)
  {
     close(Handle3115[i]);
     close(HandleSorted3115[i]);
  }

  cout << "Bdgt3115 files sorted by name." << endl;

//  ***************************************************************
//    Sort Bdgt4321 Files
//  ***************************************************************

  Handle4321[Freshman] = open(Fresh4321File, O_RDONLY | O_TEXT);
  Handle4321[Sophomore] = open(Soph4321File, O_RDONLY | O_TEXT);
  Handle4321[Junior] = open(Junior4321File, O_RDONLY | O_TEXT);
  Handle4321[Senior] = open(Senior4321File, O_RDONLY | O_TEXT);

  HandleSorted4321[Freshman] = open(SortedFresh4321File, O_RDWR | O_CREAT |
                          O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
  HandleSorted4321[Sophomore] = open(SortedSoph4321File, O_RDWR | O_CREAT |
                          O_TRUNC | O_APPEND,  S_IREAD | S_IWRITE);
  HandleSorted4321[Junior] = open(SortedJunior4321File, O_RDWR | O_CREAT |
                          O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
  HandleSorted4321[Senior] = open(SortedSenior4321File, O_RDWR | O_CREAT |
                          O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);


  for (Class = Freshman; Class <= Senior; Class++)
  {

  //  ************************************************
  //   Read the current file into a heap
  //  ************************************************


     // Keep Reading in records from the current file
     while (read(Handle4321[Class], Bdgt4321_Buf.All, Bdgt4321_Len))
     {
        // Determine the proper place to add the record in the heap
         Temp4321Ptr = (Bdgt4321NodePtr)malloc(sizeof(struct Bdgt4321Node));

         //Copy file info into a new node  to be added to the heap
         strncpy(Temp4321Ptr->Data.All, Bdgt4321_Buf.All, Bdgt4321_Len);
```

```
      strncpy (CurrentLast, Bdgt4321_Buf.Field.LastName, LastNameLength);
      //CurrentLast[LastNameLength] = '\0';

      strncpy (CurrentFirst, Bdgt4321_Buf.Field.FirstName, FirstNameLength);
      //CurrentFirst[FirstNameLength+1] = '\0';

      Index4321 = ( Heap4321.count() ) + 1;

      while (!(Index4321 == 1 ||
               //Below performs:  NewLastName > OldLastName
               ((strcmp(CurrentLast,
                        strncpy(TempLast,
                                ((Bdgt4321NodePtr)Heap4321[Index4321 / 2])->
                                Data.Field.LastName,
                                LastNameLength))                < 0) ||

               ((strcmp(CurrentLast,
                        strncpy(TempLast,
                                ((Bdgt4321NodePtr)Heap4321[Index4321 / 2])->
                                Data.Field.LastName,
                                LastNameLength))       == 0)   &&
               (strcmp(CurrentFirst,
                        strncpy(TempFirst,
                                ((Bdgt4321NodePtr)Heap4321[Index4321 / 2])->
                                Data.Field.FirstName,
                                FirstNameLength))       <=0) )      ) ) )


              // terminate when the root is reached or the element
              // is in its correct place
           {
              // check the next lower level of the heap
              Heap4321.assign(Index4321, Heap4321[Index4321/2]);
              Index4321 /= 2;
           }
      // end while(heap sort)
      Heap4321.assign(Index4321, Temp4321Ptr);

   }// end while(read files)

// ************************************************
//  Sort the Heap
// ************************************************
   int parent, child;
   Bdgt4321NodePtr Item4321Ptr;

   for (Index4321 = Heap4321.count() ; Index4321 > 0; Index4321 --)

   {

      /* delete element with the highest key from the heap */
      /* save value of the element with the highest key */
      Item4321Ptr = (Bdgt4321NodePtr)Heap4321[1];
      Temp4321Ptr = (Bdgt4321NodePtr)Heap4321[Index4321];
                    /* use last element in heap to adjust heap */
      parent=1;
      child=2;

      while (child <= Index4321)
      {
         if (child < Index4321)
         /* find the largest child of the current parent */
            if ((strcmp( strncpy(CurrentLast,
                         ((Bdgt4321NodePtr)Heap4321[child])->
```

```
                        Data.Field.LastName,
                    LastNameLength),
                strncpy(TempLast,
                    ((Bdgt4321NodePtr)Heap4321[child + 1])->
                        Data.Field.LastName,
                    LastNameLength))            < 0)

    || ((strcmp( strncpy(CurrentLast,
                    ((Bdgt4321NodePtr)Heap4321[child])->
                        Data.Field.LastName,
                    LastNameLength),
                strncpy(TempLast,
                    ((Bdgt4321NodePtr)Heap4321[child + 1])->
                        Data.Field.LastName,
                    LastNameLength))            == 0)     &&

    (strcmp( strncpy(CurrentFirst,
                    ((Bdgt4321NodePtr)Heap4321[child])->
                        Data.Field.FirstName,
                    FirstNameLength),
                strncpy(TempFirst,
                    ((Bdgt4321NodePtr)Heap4321[child + 1])->
                        Data.Field.FirstName,
                    FirstNameLength))            < 0)) )


      child++;

  if ((strcmp( strncpy(CurrentLast,
                  Temp4321Ptr -> Data.Field.LastName,
                  LastNameLength),
              strncpy(TempLast,
                  ((Bdgt4321NodePtr)Heap4321[child])->
                        Data.Field.LastName,
                  LastNameLength))              > 0)
  // if the current name is greater than the one in the heap
  // then this is the correct position to add it.... OR

   || ((strcmp( strncpy(CurrentLast,
                  Temp4321Ptr -> Data.Field.LastName,
                  LastNameLength),
              strncpy(TempLast,
                  ((Bdgt4321NodePtr)Heap4321[child])->
                        Data.Field.LastName,
                  LastNameLength))                    ==0 ) &&

         (strcmp( strncpy(CurrentFirst,
                  Temp4321Ptr -> Data.Field.FirstName,
                  FirstNameLength),
              strncpy(TempFirst,
                  ((Bdgt4321NodePtr)Heap4321[child])->
                        Data.Field.FirstName,
                  FirstNameLength))             >=0)))

// if the last names are the same, but this first name is
// grater than or equal to the one in the heap... this is
// the correct position to add it so...

     break;
  else
  {
  // move to the next lower level
     Heap4321.assign(parent, Heap4321[child]);
     parent = child;
     child *= 2;
```

```
            }
         } // end while
         Heap4321.assign(parent, Temp4321Ptr);
         Heap4321.assign(Index4321, Item4321Ptr); // Place the sorted Item at the
                                                  // end of the heap; don't sort
                                                  // this space on next pass
      } // end for each item sort

      Index4321 = Heap4321.count();

      // write out sorted file & empty the heap as you go
      for ( j=1; j<=Index4321; j++)
      {
         write (HandleSorted4321[Class],
               ((Bdgt4321NodePtr)Heap4321[j]) -> Data.All,
               Bdgt4321_Len);

         delete (Bdgt4321NodePtr)Heap4321[j];  // free memory at heap address
         Heap4321.remove(j);                   // remove its poiter from heap

      }

   }// end for each class of Bgdt4321 file

   for (i = Freshman; i <= Senior; i++)
   {
      close(Handle4321[i]);
      close(HandleSorted4321[i]);
   }

   cout << "Bdgt4321 files sorted by name." << endl;



// ***************************************************************************
// Combine sorted files into output Files
// ***************************************************************************

   // Set up blank strings to print out if a matching record is not found
   strncpy(Blank3115, Bdgt3115_Buf.All, Bdgt3115_Len); // make sure the end
   strncpy(Blank4321, Bdgt4321_Buf.All, Bdgt4321_Len); //of line char is at end

   strnset(Blank3115, ' ', Bdgt3115_Len - 1);  // sets entire string to ' '
   strnset(Blank4321, ' ', Bdgt4321_Len - 1);  //  except for eoln char.



   for (Class = Freshman; Class <= Senior; Class++)
   {
      switch (Class)
      {
         case  Freshman:
         {
            HandleSortedOthr[Class]= open(SortedFreshOthrFile,
                                          O_RDONLY | O_TEXT);
            HandleSorted3115[Class]= open(SortedFresh3115File,
                                          O_RDONLY | O_TEXT);
            HandleSorted4321[Class]= open(SortedFresh4321File,
                                          O_RDONLY | O_TEXT);

            HandleOutput[Class]= open(FreshOutputFile, O_RDWR | O_CREAT |
                                      O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
            if (HandleOutput[Class] < 0)
            {
               cout << "Unable to create Freshman output file. Program halted.";
```

```
      exit(1);
    }
  } break;

  case Sophomore:
  {
    HandleSortedOthr[Class]= open(SortedSophOthrFile,
                                                  O_RDONLY | O_TEXT);
    HandleSorted3115[Class]= open(SortedSoph3115File,
                                                  O_RDONLY | O_TEXT);
    HandleSorted4321[Class]= open(SortedSoph4321File,
                                                  O_RDONLY | O_TEXT);

    HandleOutput[Class]= open(SophOutputFile, O_RDWR | O_CREAT |
                              O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
    if (HandleOutput[Class] < 0)
    {
      cout <<"Unable to create Sophomore output file. Program halted.";
      exit(1);
    }
  } break;

  case Junior:
  {
    HandleSortedOthr[Class]= open(SortedJuniorOthrFile,
                                                  O_RDONLY | O_TEXT);
    HandleSorted3115[Class]= open(SortedJunior3115File,
                                                  O_RDONLY | O_TEXT);
    HandleSorted4321[Class]= open(SortedJunior4321File,
                                                  O_RDONLY | O_TEXT);

    HandleOutput[Class]= open(JuniorOutputFile, O_RDWR | O_CREAT |
                              O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
    if (HandleOutput[Class] < 0)
    {
      cout << "Unable to create Junior output file. Program halted.";
      exit(1);
    }
  } break;

  case Senior:
  {
    HandleSortedOthr[Class]= open(SortedSeniorOthrFile,
                                                  O_RDONLY | O_TEXT);
    HandleSorted3115[Class]= open(SortedSenior3115File,
                                                  O_RDONLY | O_TEXT);
    HandleSorted4321[Class]= open(SortedSenior4321File,
                                                  O_RDONLY | O_TEXT);

    HandleOutput[Class]= open(SeniorOutputFile, O_RDWR | O_CREAT |
                              O_TRUNC | O_APPEND, S_IREAD | S_IWRITE);
    if (HandleOutput[Class] < 0)
    {
      cout << "Unable to create Senior output file. Program halted.";
      exit(1);
    }
  } break;

  default: cout <<"PROGRAM ERROR- Invalid switch for final sort."<< endl;
} // end switch

More3115 = read(HandleSorted3115[Class],Bdgt3115_Buf.All, Bdgt3115_Len);
More4321 = read(HandleSorted4321[Class],Bdgt4321_Buf.All, Bdgt4321_Len);
```

```
while( read(HandleSortedOthr[Class],BdgtOthr_Buf.All, BdgtOthr_Len) )
{
   strncpy(LastOthr, BdgtOthr_Buf.Field.LastName, LastNameLength);
   strncpy(Last3115, Bdgt3115_Buf.Field.LastName, LastNameLength);
   strncpy(Last4321, Bdgt4321_Buf.Field.LastName, LastNameLength);

   strncpy(FirstOthr, BdgtOthr_Buf.Field.FirstName, FirstNameLength);
   strncpy(First3115, Bdgt3115_Buf.Field.FirstName, FirstNameLength);
   strncpy(First4321, Bdgt3115_Buf.Field.FirstName, FirstNameLength);

   strncpy(BdgtOthr_OutBuf.All, BdgtOthr_Buf.All, BdgtOthr_Len);
   strncpy(Bdgt3115_OutBuf.All, Bdgt3115_Buf.All, Bdgt3115_Len);
   strncpy(Bdgt4321_OutBuf.All, Bdgt4321_Buf.All, Bdgt4321_Len);

   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.LastName,
                                                    LastNameLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.FirstName,
                                                    FirstNameLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.MI, MILength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.State, StateLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.Year, YearLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.
                              FamilyMembersInCollege,
                              FamilyMembersInCollegeLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.IM, IMLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.EFC, EFCLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.ParentAGI,
                                                    ParentAGILength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.StudentAGI,
                                                    StudentAGILength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.FisapAGI,
                                                    FisapAGILength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], BdgtOthr_OutBuf.Field.FinancialNeed,
                                                FinancialNeedLength);
   write(HandleOutput[Class], " ", 1);

   if ( ((strcmp(LastOthr,Last3115)) != 0) ||
        ((strcmp(FirstOthr, First3115)) !=0 ))
      strncpy(Bdgt3115_OutBuf.All,Blank3115, Bdgt3115_Len) ;
                                          // if this doesn't match the
                                          // Othr file, print blanks

   // If these are the same student, then get the next student record
   else
      if (More3115)
         More3115 = read(HandleSorted3115[Class],Bdgt3115_Buf.All,
                                                    Bdgt3115_Len);


   write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Pell, PellLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Map, MapLength);
   write(HandleOutput[Class], " ", 1);
   write(HandleOutput[Class], Bdgt3115_OutBuf.Field.IWU_Grant,
```

```
                                                    IWU_GrantLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.IWU_GrantWithAlumni,
                                    IWU_GrantWithAlumniLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.IWU_Grant_OS,
                                          IWU_Grant_OSLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.IWU_Grant_OS_Alumni,
                                 IWU_Grant_OS_AlumniLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.OutsideScholorship,
                                    OutsideScholorshipLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.NoNeedScholorship,
                                    NoNeedScholorshipLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.SpecialCorp_NMSC,
                                    SpecialCorp_NMSCLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.MeritCorp_NMSC,
                                    MeritCorp_NMSCLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.OneTimeNMSC,
                                          OneTimeNMSCLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.AlumniAcademic,
                                    AlumniAcademicLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.AlumniWithNeed,
                                       AlumniWithNeedLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.TalentMusic,
                                          TalentMusicLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.TalentArt,
                                          TalentArtLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.TalentDrama,
                                          TalentDramaLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.MusicWithNeed,
                                       MusicWithNeedLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.ForeignStudent,
                                       ForeignStudentLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Ministerial,
                                          MinisterialLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.ParentsAssoc,
                                          ParentsAssocLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.PreTheology,
                                          PreTheologyLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.IWU_NationalMerit,
                                    IWU_NationalMeritLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Music, MusicLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Giese, GieseLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Alkonis,
                                             AlkonisLength);
```

```
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Stevenson,
                                                  StevensonLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Rupert, RupertLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Mahlstedt,
                                                  MahlstedtLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Senate, SenateLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Presser,
                                                    PresserLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Brokaw, BrokawLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Shanks, ShanksLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt3115_OutBuf.Field.Baker, BakerLength);
write(HandleOutput[Class], " ", 1);

if ( ((strcmp(LastOthr,Last4321)) != 0) ||
       ((strcmp(FirstOthr, First4321)) !=0 ))
   strncpy(Bdgt4321_OutBuf.All,Blank4321, Bdgt4321_Len) ;
                                        // if this doesn't match the
                                        // Othr file, print blanks

// If these are the same student, then get the next student record
else
   if (More4321)
      More4321 = read(HandleSorted4321[Class],Bdgt4321_Buf.All,
                                                    Bdgt4321_Len);


write(HandleOutput[Class], Bdgt4321_OutBuf.Field.AlumniAchievement,
                                     AlumniAchievementLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.StateFarmMinority,
                                       StateFarmMinorityLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.EBRust, EBRustLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.SpecialAward,
                                       SpecialAwardLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.MusicAward,
                                                  MusicAwardLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.
                                  IWU_NationalMeritWithAlumni,
                                  IWU_NationalMeritWithAlumniLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.
                                InternationalStudentAchvmnt,
                                InternationalStudentAchvmntLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.NeedBasedGift,
                                          NeedBasedGiftLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.MusicHonors,
                                MusicHonorsLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.SEOG_Initial,
                                          SEOG_InitialLength);
write(HandleOutput[Class], " ", 1);
```

```
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.SEOG_Renewal,
                                        SEOG_RenewalLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.IllinoisMerit,
                                        IllinoisMeritLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.MeritRecognition_Need,
                          MeritRecognition_NeedLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Perkins,
                                        PerkinsLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Perkins_ST,
                                        Perkins_STLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Nursing,
                                        NursingLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Nursing_ST,
                                        Nursing_STLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Stafford_IS,
                                           Stafford_ISLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Stafford_OS,
                                           Stafford_OSLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Stafford_IS_ST,
                                           Stafford_OSLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Stafford_OS_ST,
                                        Stafford_OS_STLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.LULU, LULULength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.LULU_ST,
                                              LULU_STLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.EdgarSmith,
                                           EdgarSmithLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Methodist,
                                           MethodistLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Massock,
                                           MassockLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Ferguson,
                                           FergusonLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Tripp, TrippLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.Myers, MyersLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.IWU_StudentEmployment,
                                  IWU_StudentEmploymentLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.IWU_EmploymentPartial,
                                  IWU_EmploymentPartialLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.WorkStudy,
                                           WorkStudyLength);
write(HandleOutput[Class], " ", 1);
write(HandleOutput[Class], Bdgt4321_OutBuf.Field.WorkStudyPartial,
                                        WorkStudyPartialLength);
```

```
      write(HandleOutput[Class], BdgtOthr_OutBuf.Field.CR, CRLength);
   }

   close(HandleSortedOthr[Class]);
   close(HandleSorted3115[Class]);
   close(HandleSorted4321[Class]);
   close(HandleOutput[Class]);

   switch (Class)
   {
      case Freshman  : cout << "Freshman ";  break;
      case Sophomore : cout << "Sophomore "; break;
      case Junior    : cout << "Junior ";    break;
      case Senior    : cout << "Senior ";    break;
   }

   cout << "data matched by name & collated into output file." << endl;

} // end for each class

// ****************************************
//  Remove temporary files
// ****************************************
remove(FreshOthrFile);
remove(SophOthrFile);
remove(JuniorOthrFile);
remove(SeniorOthrFile);

remove(SortedFreshOthrFile);
remove(SortedSophOthrFile);
remove(SortedJuniorOthrFile);
remove(SortedSeniorOthrFile);

remove(Fresh3115File);
remove(Soph3115File);
remove(Junior3115File);
remove(Senior3115File);

remove(SortedFresh3115File);
remove(SortedSoph3115File);
remove(SortedJunior3115File);
remove(SortedSenior3115File);

remove(Fresh4321File);
remove(Soph4321File);
remove(Junior4321File);
remove(Senior4321File);

remove(SortedFresh4321File);
remove(SortedSoph4321File);
remove(SortedJunior4321File);
remove(SortedSenior4321File);

cout << "Temporary files removed." << endl;
cout << "Process complete.";
} // end main()


// ---------------------------------------------------------------------------
int GetClass (int &Class, char* LastName, char* FirstName)
{
   Class = -1;
   clrscr();
   while ( !(Class >= 1 && Class <= 5))
   {
```

```
      cout << "No year has been found with this student" << endl;
      cout << "First Name: '" << FirstName << "'" << endl;
      cout << "Last Name : '" << LastName << "'" << endl << endl;
      cout << "Please choose one of the following:" << endl;
      cout << " 1. Put the student in the Freshman file." << endl;
      cout << " 2. Put the student in the Sophomore file." << endl;
      cout << " 3. Put the student in the Junior file." << endl;
      cout << " 4. Put the student in the Senior file." << endl;
      cout << " 5. Delete this student from the Budget Projection" << endl;
      cout << endl;
      cout << "Press 1-5, followed by ENTER: ";
      cin >> Class;
   }

   switch (Class)
   {
      case 1   : Class = Freshman;  break;
      case 2   : Class = Sophomore; break;
      case 3   : Class = Junior;    break;
      case 4   : Class = Senior;    break;
   }

   clrscr();
   return Class;
} // end procedure

// ---------------------------------------------------------------------------
```