IT2205: Programming I

Section 4

Computer program design (05 hrs)

Merging and matching two input files

INTRODUCTION

In this chapter we will look at programs using two input files. Up to now, any program that used two input files used the first file only in the start module to load an array or to load needed parameter records. After the initial loading, the first file was no longer needed and the main loop module processed just one file. In the designs in chapter, we will need data from both files during the primary loop processing.

Having two input files will require some special attention to a few details. When will we exit our Mainline loop? In most of our past designs, we stayed in the loop while we had data to process. That condition was clear enough because we were only processing data from one file. Now we will have to look at the possibility that the two files are of different lengths and are completed at different times in the design. We will also need to define when we *read* from each file.

MERGING TWO FILES

The simplest form of two-input file processing is a merge program which combines two similar sorted files into one sorted file. Usually, our combined file will utilize the same computer “medium” as the two input files. That is, if we’re reading two disk files, our output file would also be a disk file.

We could just append on file to the end of the other file and then sort the resulting file. That would work, but, since a file sort is a very time-consuming operation and since we already start with sorted files, there is a much more efficient solution. When we start with sorted files, we merely need to pull the records from each file in order and write them to the new file in appropriate order. Notice that we described our files as “two similar sorted files.” We must be able to assume that the files are sorted on the same field. If both files are partial lists of employees, but one is sorted on Social Security Number and the other is sorted on sorted on last name, we will not be able to merge them until we resort one of the files.

MERGING TWO FILES CONT…

In general, our system will be to take the “current” record from each of the two files and decide which of the two should of first in the new, combined file. At any given point, we are comparing only two records, one from each file. Our only question is: Of those two records, which one should come first? Once the selected record is written to the new file, we will read the next record from the input file. The key to remember is that we only read another record from a file when we have “processed” the previous record from that file.

MERGING TWO FILES CONT…

Assume we have a file of the members of the Roller coaster Enthusiasts Club (RCEC) and a second file of the members of the Ferris Wheel Riders of America Club (FWRAC). The two clubs are joining to form the supporters of Amusement Parks (SAP), and the membership lists must be merged. If both files are ordered on the same field (for example, name), our merge will be easy. We will take the current record of each file, compare those two records, and write one of them to the new file.

MERGING TWO FILES CONT…

RCEC File
Abernathy, Adam
Carruthers, Carol
Johnson, Jim
Klover, Kenny
Lucketts, Lucy
Masters, Martha
Opperman, Opal

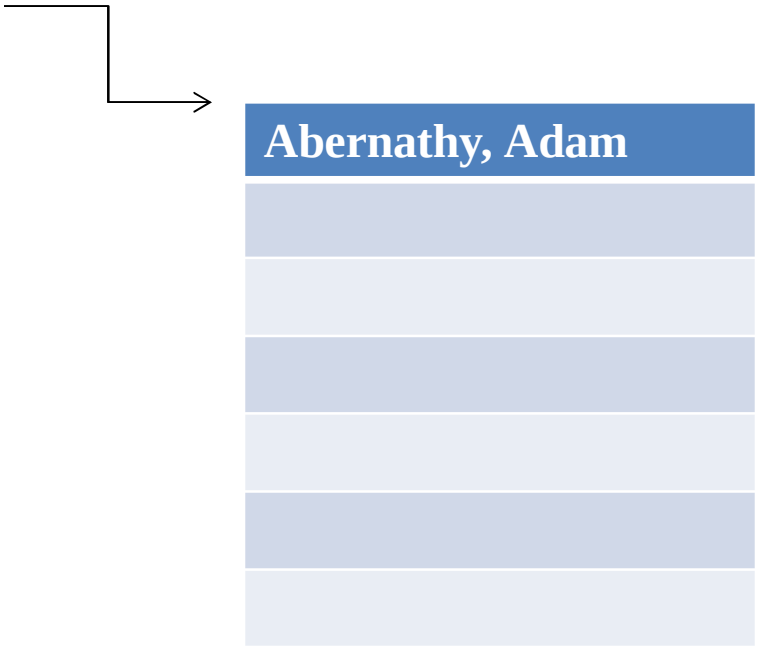
FWRAC File
Bradley, Bertha
Carruthers, Carol
Duddley, David
Frempston, Fred
Ivanoski, Ivan
Lucketts, Lucy
Nestor, Ned
Samuels, Susan

UCSC

BIT

MERGING TWO FILES CONT…

Comparison: Abernathy,Adam : Bradley, Bertha



When the first two records are compared, Abernathy is identified as the record that should come first and Abernathy, Adam is written to the new file. Since a record in the RCEC file has been processes, we read current record from that file. What happens to Bradley, Bertha? Nothing yet. That is still the current in the FWRAC file because it has not yet been processed (Written to the new file). Our next comparison will be between

UCSC

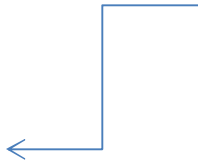
BIT

MERGING TWO FILES CONT…

Comparison: Carruthers, Carol : Bradley, Bertha

Combined file:

Abernathy, Adam
Bradley, Bertha

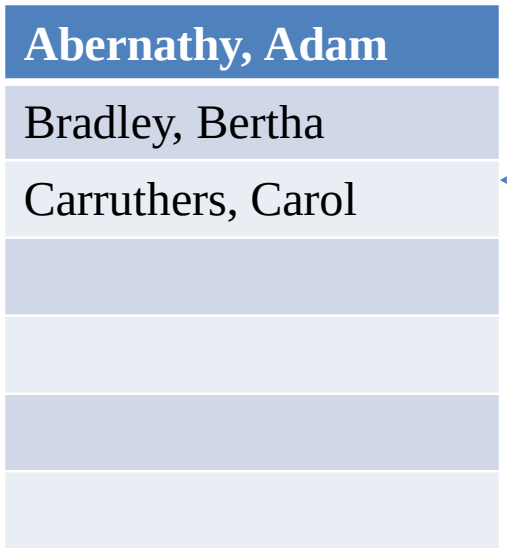


MERGING TWO FILES CONT…

After writing Bradley, Bertha's record to the new file, we read the next record from the FWRAC file: Carruthers, Carol.

Comparison: Carruthers, Carol : Carruthers, Carol

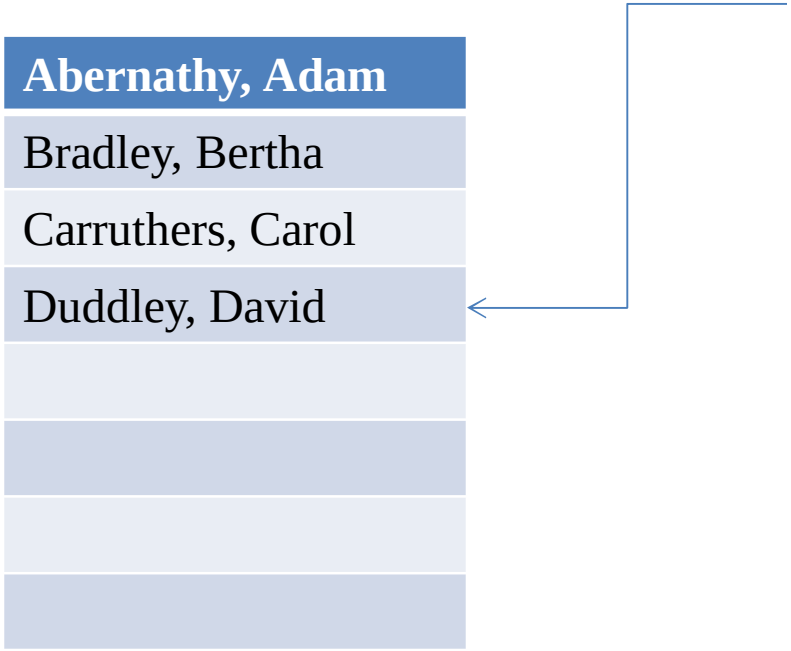
Combined file:



MERGING TWO FILES CONT…

Now we have a very different situation. The two files have the same entry. (Carol is a member of both clubs.) Our design specifications must tell us what action to take if a record occurs in both files. In this case, and in most other cases, we would write the record just once to the new file. But no matter what we do with the new file, we have processed a record from both input files, so we must read the next record from both files:

Comparison: Johonson, Jim : Duddley, David



MERGING TWO FILES CONT…

Since we are always comparing just two records, one of three possibilities must be true: Roller Name is less than FerrisName, RollerName equals FerrisName, or RollerName is greater than FerrisName. We can define what should be done for each case.

Comparison	Write	Read From
Roller < Ferris	Roller record	Roller file
Roller = Ferris	Either record	Both files
Roller > Ferris	Ferris record	Ferris file

We can summarize the chart when the compare fields are unequal, we always write the lesser of the two (assuming our files are in ascending sequence) and read from the same file. When compare fields are equal, our specifications will define the process (usually write either record), and we read from both files.

How do we get started?

Our loop process must begin with the comparison of two records, so we must have read the records prior to the beginning of the loop. In other words, a merge program will involve a priming read for each input file. We should also test to be sure when we begin that both files have data. We cannot merge two files if one is empty, so if either file is empty we will change the flag to indicate an empty file and write an error message.

How do we get started?

Start
OPEN files
moreRollers = ‘yes’
moreFerris = ‘yes’
If there are data (Roller File)
    READ rollerName, rollerAddr
ELSE
    moreRollers = ‘no’
    WRIT ‘ERROR -- Roller File is empty’
ENDIF
If there are data (ferris File)
    READ ferrisName, ferrisAddr
ELSE
    moreFerris = ‘no’
WRITE ‘ERROR – Ferris File is empty’
ENDIF

If either file is empty, the appropriate flag will be changed to ‘no’ and an error message is written. If both files are empty, both flags are changed to ‘no’ and two error messages are written.

WHEN DO WE EXIT THE MAINLINE LOOP”

The simplest design solution is to exit the loop when either file is completed, in other words, to stay in the loop only while both files have data remaining. Our test is then in effect saying, “WHILE we have data in the Roller *file* and we have data in the Ferris file, keep doing this loop.” Of course, if we exit when just one file is completed, we presumably still have data in the other file that has not yet been copied to the new file, and our merge is not yet completed. We will have to finish writing the records to the new file in our Finish module. A completed design follows.

The End