IT2205: Programming I

Section 4

Computer program design (05 hrs)

File Updates

INTRODUCTION

In the last chapter we looked at programs that “match” a master file with a transaction file that holds transient data. Many operations will require matching a relatively static master file with a second file that reflects the current data (e.g., hours worked this week, total sales this sales period, credits registered this term). We know, however, that no file is completely static. Our file of employees may not change on a daily or weekly basis, but it will change. In this chapter, we will look at the means to make the necessary modifications to the “master” file. We are “updating” the master. It is possible that we have hired a new employee (an Add to the master), that an employee has left our company (a Delete from the master), or that an employee has a new address or a new phone number (a Change in the master). The update program allows us to make these modifications to the master file.

INTRODUCTION CONT...

Usually, we also have two output files: a new master file (reflecting all the transaction modifications) and an error report. The new master file will typically utilize the same storage medium as the input master file (disk, tape, etc.) since it will replace the input master as the new, update master file.

The new master file will maintain the sorted order of the original master but will reflect any additions, deletions, and changes from the transaction file. The error report is typically a printed document. The error report identifies transaction processes that could not be performed: You cannot change or delete a record that is not in the master, and you cannot add a record which is already in the master. Some update programs will also generate a printed “audit” report listing all actions taken on the master file. As we will see in the next chapter, an audit report is more important with a non-sequential update.

INTRODUCTION CONT...

We will have two input files: the master that is to be modified and the transaction that holds the modifications to be made. The master file will be the same from of master file that we used in the matching program and will be sorted on unique key field found in each record (name, Social Security Number, ID, etc.).

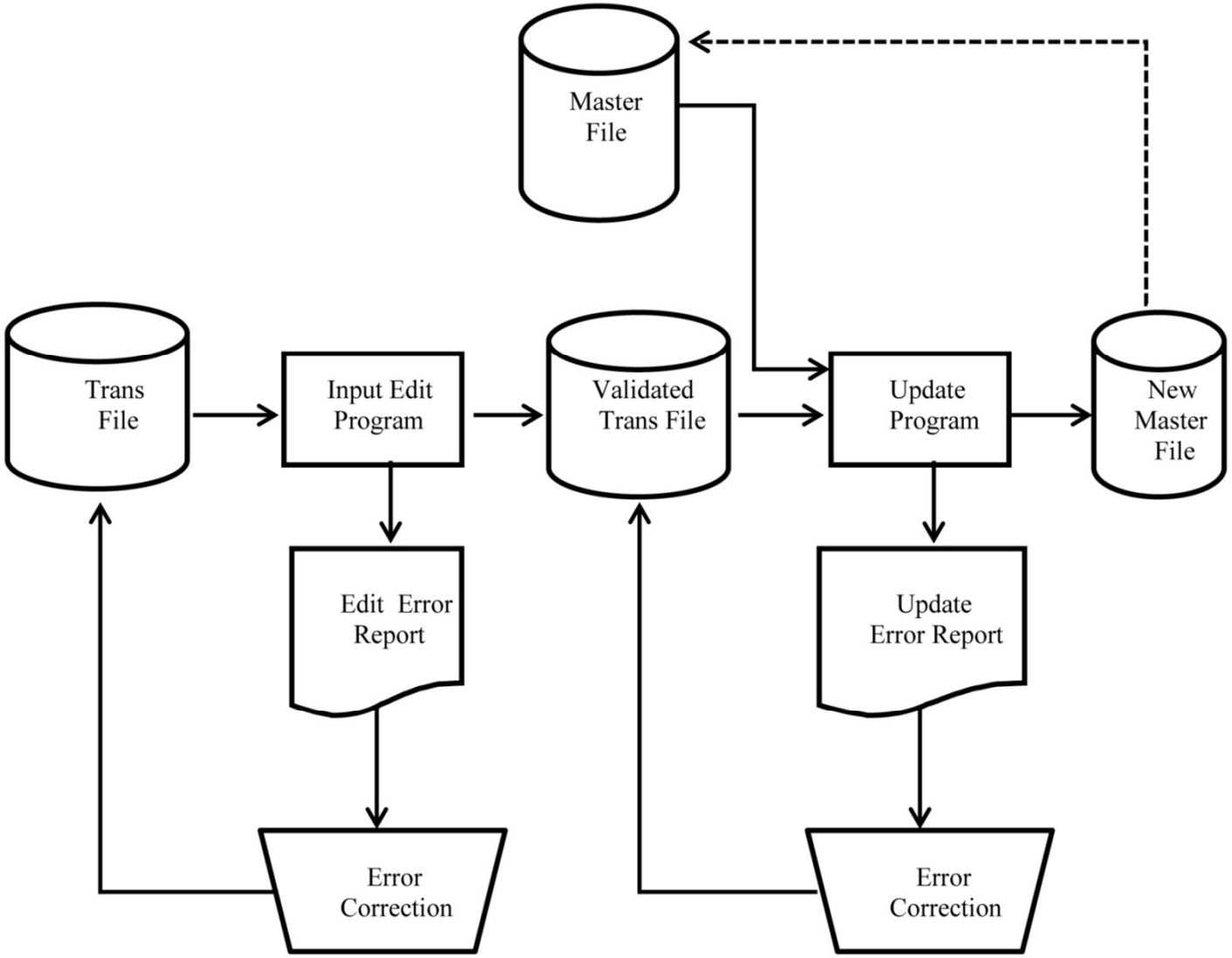
The transaction file must be sorted on the same field because that is the field by which we will match a transaction record to the master record. The update program differs from the matching program in that we now have three different processes to perform. A transaction record may call for an Add, for a Delete, or for a Change. Obviously, there must be same code included in the transaction record which indicates what transaction process is to be performed.

INTRODUCTION CONT...

What follows (on the next page) is a system flowchart for a sequential update program. (See Appendix B for more information about system flowcharts.) Naturally, we would want the transaction file to be as error-free as possible prior to an update so this chart includes an input editing check of the transaction file prior to the update.

The input editing can only check for *field* error (e.g., whether the transaction code is one of the valid codes). We still have the possibility of record errors when we perform the update (e.g., the code is a valid code, but is not valid in the current circumstances). Those errors would be noted in the update Error Report.

INTRODUCTION CONT...



INTRODUCTION CONT...

Notice that there is a broken line going back from the New Master File to the Master File. This line is to indicate that the New Master "becomes" the working master and the next time the update program is run, we will be updating what used to be the "new" master. This process is known as a generational backup. The "old" master along with the transaction file serve as a backup for "new" master. If we somehow lost or damaged the new master file, we could always recreate it by running the update program again.

If we update our master on a weekly basis, we would also then have a weekly backup. Let's say that in week one, we do our first update. The old master becomes the backup for the new master. The old master is sometimes called the "parent" (generational backup, remember). In week two, we do our second update. The current master becomes the old master and we create a two, we do our second update. The current master becomes the old master and we create a new master. We now have three generations: grandparent (the original master), parent, and the current master (child?). Depending on how valuable our data are and how frequent the changes are, we might keep up to five or more generational backups.

ORGANIZING THE UPDATE THE UPDATE POSSIBILITIES

We know that there are three ways in which a given master record and a given transaction record can “match”:

Master = Transaction
Master > Transaction
Master < Transaction

We also know that there are three possible update actions:

Add a record (copying a record from the transaction file to the new master). Delete a record (*not* copying a record from the old master file to the new master). Change a record (making changes before copying a record from the old master file to the new master).

But not all possible actions are valid with each match possibility. If the master field is equal to the transaction field, we could change the master record or we could delete the master record, but we cannot add the transaction record. We cannot add a record that already exists in the master file.

ORGANIZING THE UPDATE THE UPDATE POSSIBILITIES CONT...

If the master field is greater than the transaction field, we know that we will never find a master record to match the current transaction record. Therefore, the transaction cannot be a Change or a Delete because those operations require a match. But the transaction could be an Add since an Add must not have a matching master.

If the master field is less than the transaction field, the transaction record is not relevant at all. We know that the current transaction record is not related to the current master record, and that we will not find a subsequent transaction record to match the current master record. Therefore, there is no transaction record to this master record that needs to be processed. Do we just ignore the current master? No, it still a part of the master file, so we copy it from the old master to the new master.

Two additional questions remain. When do we write an error message in the error report? Any time we have an invalid transaction code (an Add code with a transaction record that matches a master or a Change or Delete code with a transaction record that does not match a master), we must identify that error report. If our transaction file has been through an input-editing program first, we should be able to assume that the code itself is valid (e.g., if our codes are A,C, and D, we have only those three letters). Next, when do we read form an input file? In our matching and merging programs we read a record from a file when we had “processed” the previous record from that file. We will use the same rule in our update program.

ORGANIZING THE UPDATE THE UPDATE POSSIBILITIES CONT...

The following chart summarizes the operations. Remember, we are comparing the equivalent fields from two files, such as an ID field from both files or a name field.

Condition	Code	Action	Read from
Master = Trans	A	Invalid Code, Write Error	Trans file
	C	Make changes to Master Write Master to New Master	Trans & Master Files
	D	No action	Trans & Master Files
Master > Trans	A	Write Trans to New Master	Trans file
	C	Invalid Code, Write Error	Trans file
	D	Invalid Code, Write Error	Trans file
Master < Trans	N/A	Write Master to New Master	Master file

ORGANIZING THE UPDATE THE UPDATE POSSIBILITIES CONT...

A Sample Design

The following program updates an employee master file including ID, name, address, and telephone. The file is in order by ID. Each field will be identified in the design by preceding it with om (old master) as in omID or omName. The updates are found in the transaction file which includes an ID, transaction code, and possibly a name, address, and telephone. The file is sorted on ID. If the transaction code is D (Delete), only the code and ID will be present. If the transaction code is C (Change), the code, ID, and any field which is to be changed will be present. If the transaction code is A (Add), all fields will be included. Transaction fields will identified in the design by preceding the name with a t as in tID,, tname, and so on.

ORGANIZING THE UPDATE THE UPDATE POSSIBILITIES CONT…

Mainline
DO start
WHILE more Data = ‘yes’
    DO ProcessUpdate
ENDWHILE
DO Finish

Start
OPEN files
moreData = ‘yes’
moreTrans = ‘yes’
moreMast = ‘yes’
lineCnt = 60
pageCnt = 0
DO ReadTrans
IF moreTrans = ‘no’
    moreData = ‘no’
    WRITE ‘ERROR – Master file is empty’
ENDIF

```
ProcessUpdate
IF omID < tID
        DO CopyMAster
ELSE
        IF omID = tID
        IF tCode = 'C'
                DO Change
        ELSE IF tCode = 'D'
                DO Delete
        ELSE IF tCode = 'A'
                DO BadAdd
        ENDIF
        ELSE
        IF tCode = 'A'
                DO Add
        ELSE IF rCode = 'C'
                DO BadChange
        ELSE IF tCode = 'D'
                DO BadDelete
        ENDIF
        ENDIF
ENDIF

Finish
CLOSE files
```

Change

```
IF tName not = spaces
        omName = tName
ENDIF
IF tAddr not = spaces
        omAddr = tAddr
ENDIF
IF tPhone not = spaces
        omPhone = tPhone
ENDIF
WRITE omID, omName, omAddr, omPhone (to new master file)
DO ReadTrans
DO ReadMast
```

Delete

```
DO ReadTrans
DO ReadMast
```

Add

```
WRITE tID, tName, tAddr, tPhone (to new master file)
DO ReadTrans
```

CopyMaster
WRITE omID, omName, omAddr, omPhone, (to new master file)
DO ReadMast

BadAdd
msg = ‘Add code with matching record in master
DO ReadTrans

BadChange
Msg = ‘Change code with no matching record in master’
DO ReadTrans

BadDelete
Msg = ‘Delete code with no matching record in master’
DO ReadTrans

ReadTrans

```
If there are data
        READ tCode, tID, tName, tAddr, tPhone
ELSE
        MoreTrans = 'no'
        tID = 99999
        IF moreMast = 'no'
                MoreData = 'no'
        ENDIF
ENDIF
```

ReadMast

```
IF there are data
        READ omID, omName, omAddr, omPhone
ELSE
        moreMast = 'no'
        omID = 99999
IF moreTrans = 'no'
                moreData = 'no'
        ENDIIF
ENDIF
```

WriteErrorLine

```
If lineCnt > = 60
        Do Heads
ENDIF
WRITE tCode, tID, tName, tAddr, tPhone, msg
ADD 1 to lineCnt
```

Heads

```
EJECT page
SKIP 6 lines
IF pageCnt = 0
        WRITE Report Heading
        SKIP 1 line
        lineCnt = 8
ELSE
        LineCnt = 6
ENDIF
ADD 1 to pageCnt
WRITE Page Heading, pageCnt
WRITE Column Headings (dbl spc)
SKIP 1 line
ADD 4 to lineCnt
```

Desk check the design using the following data:

Master

3	Sam	Vienna	281-4318
5	Dot	Reston	437-9117
7	Jim	Herndon	450-9494
9	Sue	Sterling	820-3326
11	Joe	Leesburg	777-1770

Transaction

3	D			
5	C		Herndon	
6	A	Tom	Vienna	281-9143
7	A	Jim	Herndon	450-9494
8	C	Carol		

Loop Iteration #1

Master: Transaction:

3 Sam Viena 281-4318

Tests and Actions:

OmID = tID ⟶ Check tCode

tCode = 'D' ⟶ Read next transaction record

Read next master record

Loop Iteration #2

Master:			Transaction			
5	Dot	Reston	437-9117	5	C	Herndon

Tests and Actions:

omID = tID	→	Check tCode
tCode = 'C'	→	Do Change Module
Check all fields	→	Change omAddr
		Write master to new master
		Read next transaction record
		Read next master record

Loop Iteration #3

Master:			Transaction					
7	Jim	Herndon	450-9494	6	A	Tom	Vienna	281-9143

Tests and Actions:

omID > tID	→	Check tCode
tCode = 'A'	→	Write transaction record to new master
		Read next transaction record

Loop Iteration #4

Master:				Transaction				
7	Jim	Herndon	450-9494	7	A	Jim	Herndon	450-9494

Tests and Actions:

omID = tID	→	Check tCode
tCode = 'A'	→	Write error message (bad add) to report
		Read next transaction record

Loop Iteration #5

Master:				Transaction		
7	Jim	Herndon	450-9494	8	C	Carol

Tests and Actions:

omID < tID	→	Copy master to new master
		Read next master record

Iteration #6

Master:				Transaction		
9	Sue	Sterling	820-336	8	C	Carol

Tests and Actions:

omID > tID	⟶	Check tCode
tCode = 'C'		Write error message (bad change) to report
		Read next transaction record (no more records, so tID set to 99999

Iteration #7

Master:				Transaction
9	Sue	Sterling	820-3326	99999

Tests and Actions:

omID < t-ID	⟶	Copy master to new master
		Read next master record

Iteration #8

Master:				Transaction
11	Joue	Leesburg	777-1770	99999

Tests and Actions:

omID < tID ⟶ Copy master to new master

Read next master record (no more records, so OM-ID set to 99999)

MULTIPLE TRANSACTION RECORDS

We have made a major assumption in this design. After processing matching transaction and master records, we read the next record from both files. We assumed that we would not have another transaction record for that master record. But in reality, we might have several transactions.

If an employee moved, we would get notice of change of address, and we would include a change transaction in the file. Several days later (but before we ran the update program) that employee might get a new phone number and we would receive another change, which we would include as an additional change transaction record. When another change, which we would include as an additional change transaction record. When the update program was run, the master record for that employee should match with two transaction records. With our current design, the second one would not be processed as a match, however, because we would have read a new employee. We would, of course, include an add transaction record in the transaction file. But if that employee came in the next day with the update of telephone number, we would then include a change transaction. When we ran the update program, the change would be seen as invalid because it would not match a master. (It matched the previous transaction record which was already written to the new master.)

MULTIPLE TRANSACTION RECORDS CONT...

To modify the design to account for multiple transaction records, we have to make some adjustment when we write to the new master and when we read the next old master. In the chapter on file match programs, we allowed for multiple transaction matches to the same master by waiting until we read a non-matching transaction record to read from the master file. We will follow the same logic here. With an update program we must consider when do we write the updated record to the new master file. We now cannot write to the new master as soon as we have processed a change or add, because there may be another related transaction. As with the read from the old master file, we will wait to write to the new master file until we read a non-matching transaction record.

Allowing for additional transaction updates to an Add is a bit more complex. We can't simply hold the record in its current location in memory because when we read the next transaction record we will lose the "added" record' One solution is to set up a new location in memory to "hold" the record prior to writing it. The design that follows calls this holding area the newID, newName, and so forth. We could make the following, fairly simply, change to the Change and Add modules:

MULTIPLE TRANSACTION RECORDS CONT...

Change **WARNING: this is not**
REPEAT **the final version**

```
        IF tName not = spaces
                omName = tName
        ENDIF
        IF tAddr not = spaces
                omAddr = tAddr
        ENDIG
        IF tPhone not = spaces
                omPhone = tPhone
        ENDIF
        DO ReadTrans
UNTIL tID < > omID
WRITE omID, omName, omAdde, omPhone (to new master file)
DO ReadMast
```

MULTIPLE TRANSACTION RECORDS CONT...

```
Add
newID = tID
newName = tName
newAddr = tAddr
newPhone = tPhone
DO ReadTrans
WHILE new ID = tID
        IF tName not = spaces
                newName = tName
        ENDIF
        IF tAddr not = spaces
                newAddr = tAddr
        ENDIF
        IF tPhone not = spaces
                newPhone = tPhone
        ENDIF
        DO ReadTrans
ENDWHILE
WRITE newID, newName, newAddr, newPhone (to new master file)
```

MULTIPLE TRANSACTION RECORDS CONT...

This design is consistent with the modifications we made in our match program to account for multiple transaction records – i.e., nested loops within the processing module. The problem is that even this design change is making an assumption. It assumes that any additional transaction records will be Changes – it never checks the tCode! In reality, though, we could have a change and then have a delete for the same master record or an add and then a delete. We must try a different tactic. In the following version, the Change module looks almost the same as the original version; we have just removed the DO ReadMast.

The Add module is quite different. Instead of immediately writing the added transaction record to the new master, we save it in the memory location where the master records is stored. First, of course, we must save the master record in a different location (holdRecord). The advantage of this logic is that the other modules were all written so that they compared fields to the master record. Now the “added record” will be where the master record would have been, and the same comparisons can take place.

MULTIPLE TRANSACTION RECORDS CONT...

We also set a flag, recInHold, to ‘yes,’ indicating that we have a master record already in memory waiting to be processed. When the ReadMast module is called, it first checks to see if a master record is still held in memory.

If a master record is held, the design pulls that record from the holdRecord location to the memory location where the master is stored (and changes recInHold back to ‘no’). The Change and Add modules no longer write a record to the new when the old master is less than transaction record (in CopyMaster). Two other modules ReadMast. In ReadTrans, we need to check to sure recInHold to ‘no’ before the first call to ReadMast. In ReadTrans, we need to check to be sure recInHold is ‘no’ before we change Moredata to ‘no’-the changed module is shown on the next pages. The other modules are not changed from our original design.

MULTIPLE TRANSACTION RECORDS CONT...

Changes

```
IF tName not = spaces
        omName = tName
ENDIF
IF tAddor not = spaces
        OmAddr = tAddr
ENDIF
IF tPhone not = spaces
        omPhone = tPhone
ENDIF
DO ReadTrans
```

Add

```
COPY omRecord tp holdRecord        [copies all fields one memory location to another]
omID = tID
omName = tName
omAddr = tAddr
omPhone = tPhone
recInHold = 'yes'
DO ReadTrans
```

MULTIPLE TRANSACTION RECORDS CONT...

```
ReadMast
IF recInHold = ‘yes’
        COPY holdRecord to omRecord
        recInHold = ‘no’
        IF moreTrans = ‘no’ AND moreMast = ‘no’
                moreData = ‘no’
ENDIF
ELSE
        IF there are data (Master file)
                READ omID, omName, omAddr, omPhone
        ELSE
                moreMast = ‘no’
                omID = 99999
                IF moreTrans = ‘no’
                        moreData = ‘no’
                ENDIF
        ENDIF
ENDIF
```

MULTIPLE TRANSACTION RECORDS CONT...

```
ReadTrans
IF there are data (Transaction file)
        READ tCode, t/D, tName, tAddr, tPhone
ELSE
        moreTrans = 'no'
        tID = 99999
        IF moreMast = 'no' AND reclnHold = 'no'
                moreData = 'no'
        ENDIF
ENDIF
```

MULTIPLE TRANSACTION RECORDS CONT...

We often use flags in a program to help clarify the processing. The above design uses the moreData, moreMast, and moreTrans flags to make the program "readable." Unfortunately, the flags actually add more confusion than clarity. The moreData flag that controls the main processing loop does give the mainline module a simplified look. But the flags can lead to confusion when we move an added transaction record into the master record memory location. If the last record in the transaction file is an Add and the master file has already been completed (moreMast = 'no'), we cannot change moreData to 'no' when we try to do the next read since we haven't finished processing the added record. That's why we have the extra check on recInHold in the ReadTrans module and the extra check on both flags in the ReadMast module when we retrieve a record from the hold area. The final example simplifies the problem by not using the flags. The loop is controlled by the actual values in tID and omID.

Try creating a data file that includes multiple transaction records for a given master and additional transaction records after an Add and desk check this design.

MULTIPLE TRANSACTION RECORDS CONT...

Another variation, much simpler this time, I promise, is to add counters to the design. In some update situations, it is helpful to know how many records were added to the master, or how many were deleted. We can easily add counters for each operation. All counters would be initialized to 0 in the Start module and written in the Finish module. Where would they be incremented? The counter keeping track of the number of deletes would be incremented in the Delete module, the counter for adds would be incremented in the Add module, and so on. It is less likely to need a count of the number of changes, but you should remember that if you increment a counter in the Change module, you are counting the number of change transaction records, not the number of records that received changes (since we could have multiple changes for one record). We could also count the number of records read from the old master fine and the number of records written to the new master file. We then have a way to double-check our numbers since the records read from the old master plus the adds minus the deletions should equal the number of records written to the new master.

MULTIPLE TRANSACTION RECORDS CONT...

One final variation' We have included two output files in our sample design, the new master and the printed error report. Some update programs also include an "audit report" that records every action taken on the master file. Any change to a record, any deletion, and any addition would be recorded in the audit report. The audit report is then used as a written (human readable) record of the actions taken and would be used only if a problem of question occurs. It is not the kind of report that gets circulated. It is field until it is needed. But if it is needed, it may be much easier to refer to printed document than trying to reconstruct what was done by referring to the transaction and master file (which are not in a human-readable format).

A FINAL EXAMPLE

The final example is a revision of our completed update, adding counters for the number of records written to the new master, the number of adds, and the number of deletes, and adding a printed audit report. Notice that the audit report is not a fancy report with headings.

Mainline
DO Start
WHILE tID < > 99999 OR omID < > 99999
    DO ProcessUpdate
ENDWHILE
DO Finish

The End