IT2205: Programming I

Section 4

Computer program design (05 hrs)

Non-Sequential Files

Sequential versus non-sequential files

All of the input files that we have used up to now have been sequential files. When we read from the files, we started reading at the first record and read *sequentially* through the file to the end of-file marker after the last record. We could read a particular record only by first reading all the records that preceded it in the file. With our merging, matching, and update program, we needed to be sure that both input files were sorted on the same field since we would be processing both files sequentially.

A non-sequential file sound like it might be a file that is out of order. But it really means a file that we can process out of order. Instead of needing to read the first 10 records to access the 11th record, we can start with the 11th record if we want. The ability to access any record at any time changes the way we think about the two-input file programs we have been designing. Instead of having to process the master and transaction files in parallel, we can let the transaction file determine the processing order even if the transaction file has not been sorted. If the first record in the transaction file indicates that the 99th record in the master file needs to be changed, we access that 99th record and make the necessary changes. If the second transaction record indicates that we have changes for the 47th master record, we can next access the 47th record. And all without ever reading the first 46 master records!

Sequential versus non-sequential files cont...

Naturally, this kind of advantage, comes at a cost. First, we can only have non-sequential files on hardware devices that provide non-sequential access. Some storage devices are limited to sequential access (tape drives, for example). Some devices can provide either sequential or non-sequential access (disk drives, for example). Second, just having the file on a non-sequential device does not assure non-sequential access. The file must have been created and maintained for non-sequential access. In other words, we can’t decide later that we want to make a file a non-sequential file without completely rewriting the file.

There are other costs as well built into non-sequential access designs. Remember that with our sequential access update program we had an automatic backup system. The very process of performing the update also created a backup of the master file. With a non-sequential update, we will be making the changes directly in the original master, so no backup is created automatically. We can certainly live with this change, but we must be aware of the need to create a backup.

Sequential versus non-sequential files cont...

Non-sequential access sound efficient and easy. But in reality, if we are updating very many of our records (say 25 percent or more), it is more efficient if we can do the update sequentially. Imaging the post office mail carrier delivering the daily mail. It would be possible to take the mail in whatever order it happened to arrive and deliver it. But of course, the mail carrier would be zig-zagging back and forth ?????? delivery area, and the time saved by not having to sorted the mail wouldn't make up for time lost zig-zagging. Obviously, it is much more efficient for the carrier to sort the mail first and then deliver it sequentially, even if only 25 percent of the house actually received mail

Some application require non-sequential updating even though squential would be more efficient. The problem with sequential updating is that it assumes we have all data available at one time. We gather all the transaction records, sort them, then perform the update program. If we must process the transaction immediately instead of "saving" it for the daily or weekly update, we must use a non-sequential file and non-sequential update. Imagine calling an airline to change your reservation and being told that they would process your change the following Friday when they did the weekly update.

Types of non-sequential access

There are a variety of ways to implement non-sequential access to a file. We will look in very general terms at two of the broad categories-each which multiple variations.

Direct access

The simplest form of direct access to a record in a file is analogous data in a positional array. With a positional array, we know the exact location of and element in the array because there is a natural link between the element and the position. We could have inventory numbering system in which our parts are numbered from one the maximum number of parts. Part names could then be sorted on a positional array by part number. Part number 10 would be in the 10th position in the array. Part number 158 would be in the 158th position. If someone asked for the name of part number 432, we would not need to do a search: the name is directly accessible by using the part number as the subscript.

Direct access cont…

Direct access to files could be done in the some way if we have field within each file entry (record) that can be used as the address of the record. Address here refers to the actual location for that record on the storage medium, the address on the disk itself. If we have a file of students enrolled at the college and each student has a suitable student ID number, we could use the ID number as the address of the record. If student got new telephone number and his record was to be updated, the student would give his number to access the record would then be rewritten back to the same location in the file). Of course, if the student came in but couldn't remember his ID number, we would have trouble accessing his record.

This form of direct access, however, usually doesn't work very well. We usually don't have a field in the record that can be used directly as an address. So we do the next best thing-we manipulate a field to create a suitable address from it. A Social Security Number would not be a good number to use for positional, direct access. The range of Social Security Numbers is fat too large for positional, direct access. The range of Social Security Number so that we create a suitable number. We perform a mathematical operation on the field within the record to create the address. As long as we always perfume the same operation we will always be able to find the stored records. The process of creating the address based on a field in the record is called *hashing*.

Direct access cont...

Assume we have a hashing algorithm to convert an ID number to the storage address. To add a record to the file, we apply the algorithm to the new employee’s ID number to “create” the address for that record. The record is then stored at that address. If we later need to access the record, we again apply the algorithm to the ID number. We will get the same address, so we can directly access the needed record. There is one problem. When we convert the ID number to an address, we might derive an address that is already used. All direct access programs have some means for handling these “collisions,” again, as long as they are handled consistently, we will have no problem finding the record.

Fortunately for us, designs, the hard work in performing direct access operations is done for us. When we first create the file, we must identify it as direct access, identify the field to use as the key access field, and identify the maximum number of expected records. The direct access system will build the file for us. A hashing algorithm will be applied and the address determined without any work on our part.

Direct access cont...

But direct access files are not the perfect solution to all file needs. Remember that some operations are best handled sequentially. We might want to do some operations through direct access but other operations, on the same file, sequentially. Can we access a direct access file sequentially; that is, can we read that file from the first record through to the last record? The answer is yes, but you may not want to. The order of the records in the file is determined by the hashing algorithm and will probably not have any meaning for the user.

Indexed Sequential Access

A compromise between the pure sequential file and the direct access file is an indexed sequential file. Indexed files are still stored sequentially based on some key field in the record and can still be accessed, when desired, in that sequential order. In addition, a separate area in storage holds an *index* to the file providing the value of the key field for the last record in each area of the file. Disks are divided into tracks, so an index could show the last record for each track. To access a needed record, we would enter the value of the key field. The system will check the index to determine the correct track and when will sequential read that track looking for the needed record. Obviously, this will be slower than a direct access since the system must first read the index and then read the identified track sequentially, but it allows us to access any record we need non-sequentially while retaining the advantages of sequential processing.

If you were going to visit a friend for the afternoon, you could look for the house *sequentially* by diving systematically up one street and down the next until you got to your friend's house. If you started at the west end of town and your friend lived on the east end, it might be a long time before you got to the right house. Sequential processing a file when we really want to process just one record wastes a lot of time.

Indexed Sequential Access Cont...

If you knew exactly where your friend lived, you could hire a plane to fly over and you could parachute directly to your friend’s house. While that’s not the approach most of us would use to “drop in” for an afternoon, it gives us a reasonable comparison to direct access. A more down to earth view might be driving directly to the friend’s house.

If, on the other hand, you knew which street your friend lived on but didn’t know the exact address, you could go directly to the street and then drive slowly down the street looking for the correct house. This process is rather like an indexed sequential access. It is slower than driving directly to the hoses (although if the record is at the beginning of the identified track or the house is at the first corner it isn’t much slower), but much faster than sequentially checking each house in the town or each record in the file.
Again, fortunately for us, most of the work for indexed sequential files is done for us. When we create the file, we must declare it as an indexed sequential file, and we must provide the records in sequential order. The index is automatically built and maintained for us. There are a few design “oddities” that we must account for, but the index creation and use are not our problem. For the user, the file will look and act just like a direct access file.

Indexed Sequential Access Cont...

One of the oddities is that we don't actually delete records from an indexed sequential file, we *mark* them for deletion. That is, we're doing a "logical" deletion rather than a "physical" deletion. Each record has a delete flag stored with it, and the flag will be set if the record is a "deleted" record. The record remains in the file, however. Therefore, when we retrieve a record, we must be sure that we aren't retrieving a logically deleted record.

Non-sequential file usually assume that we will be able to read from and write to a file in the same program. Therefore, we can read a record, make the necessary changes to the record while it is in memory, and then write the record back to the file. To distinguish between an operation in which we are adding a record to a file and one in which we are replacing an existing record, we will use two different commands. WRITE will be used to indicate a new record being written to the file. REWRITE will be used to indicate the replacement of an existing record.

Finally, all accesses to an indexed sequential file are completely dependent on the validity of the key field. For that reason, some languages require a check on the key field as a part of any file access operation and require that the designer/programmer identify what steps should be taken if the key field is not valid. In these languages, any file access operation, like a READ or a WRITE, will have two steps: the file access itself and the description of what to do if the key is invalid.

UCSC

BIT

An update program using an indexed sequential file

In the last chapter, we designed a sequential update of a master employee file which included ID, name, address, and telephone. Now we'll assume that the master file is an indexed sequential file which uses the ID as the key field. We will update the master using a transaction file containing additions (code A), deletions (code D), and changes (code C). The transaction file need not be in any particular order since we will access only the needed record in the master file. We will assume that logically deleted records have an '*' in the delete flag area and non-deleted records have a space.

<u>Mainline</u>

```
DO Start
WHILE there are transaction data
        DO ProcessUpdate
ENDWHILE
CLOSE Files
```

<u>Start</u>

```
OPEN Master file (Indexed I/O), Trans file (Seq, Input), Report (Printer)
lineCnt = 60
PageCnt = 60
```

The End