Motivation:

Say you have 2 (or more) files of records (not necessarily the same kind of records) such that:

- both record types include key **k**

- both files are sorted by key **k**

Then there are a number of important operations that you can perform on these files which require:

- only sequential operations

- no data structures in memory

Can you think of such operations?

- merging two or more files into one (called a *union* operation.

- finding records common to both files (called *file intersection*).

- finding where two files differ.

- applying transactions in one file to the master records in another file (called a master-file update or general ledger program).

What we seek is a model of coordinated multiple sequential file processing, or cosequential processing, which performs the above types of operations while reading each file sequentially.

Example 1: Matching Names in Two Lists (FZR 8.1.1)

List 1	**List 2**
Bailey	Ashford
Bolton	Bailey
Borzov	Bolton
Dreshler	Riddick
O'Brian	Surin
Thompson	Williams

Cosequential Process: read each item sequentially from each list, and if the two items match, output the first item as a member of the match set.

File Synchronization:

At each step, we are pointing to one item in List 1, Item(1), and one item from List 2, Item(2). Then our synchronization consists of:

1. If Item(1) is *less than* Item(2), then get the next item from List 1;

2. If Item(1) is *greater than* Item(2), then get the next item from List 2; and

3. If the items are equal, then output the item and get the next items in both lists.

What do we need to construct an algorithm to perform this?

1. initialization

2. the ability to access the next item in a list

3. synchronization - making sure that the current item in one list is never too far ahead of the current item in the other so that a match is missed

4. handling end-of-file conditions - the program must stop when the end of one list is reached

5. error handling - errors in either list (e.g., out of sequence, duplicates, etc.) must be handled

Algorithm (Page 294 FZR)

```
int Match (char * List1Name, char * List2Name,
   char * OutputListName)
{
   int MoreItems; // true if items remain in both lists

   // initialize input and output lists
   InitializeList (1, List1Name); // initialize List 1
   InitializeList (2, List1Name); // initialize List 2
   InitializeList (OutputListName);

   // get first item form both lists
   MoreItems = NextItemInList(1) && NextItemInList(2);

   while (MoreItems) { // loop until one list empty
      if (Item(1) < Item(2))
         MoreItems = NextItemInList(1);
      else if (Item(1) == Item(2)) // Item(1) == Item(2)
      {
         ProcessItem(1); // match found
         MoreItems = NextItemInList(1) &&
                     NextItemInList(2);
      }
      else // Item(1) > Item(2)
         MoreItems = NextItemInList(2);
   }
   FinishUp();
   return(1);
}
```

Example 2: Master-File Update (or General ledger program)

Master File:

- contains the definitive information about accounts
- needs to be updates periodically, e.g., monthly
- is sorted by some key, which is unique, e.g., account number

Transaction File:

- contains the transactions that will be used to update the Master-File, e.g., account transactions that update a client's account balances

- sorted by the Master-File's key

- Master-File key may **not** be unique with respect to the transaction file. Why not?

Cosequential Process: Each month take the sorted transactions and update the Master-File.

- for a given Master-File key, there may be no transactions, 1 transaction, or multiple transactions

- for a given transaction record key, there may be no Master-File records (error, unless we have a new way of initializing MF records), one Master-File record, or multiple Master-File records (error in MF)

Summary of the Model:

Assumptions:

- two or more input files are processed in parallel to produce one or more output files.
- each file must be sorted on one or more key fields, and all files must be sorted identically.
- cosequential processing may require a “high value” that’s higher than any possible record key and a “low value” that’s lower than any possible record key.
- records are processed sequentially in logical sorted order.