COSEQUENTIAL PROCESSING
Contents of today’s lecture:

• Cosequential processing (Section 8.1),
• Application: a general ledger program (Section 8.2)

Cosequential Processing

Cosequential processing involves the **coordinated processing** of **two or more sequential lists** to produce a single output list.

The two main types of resulting output lists are:

- Matching (intersection) of the items of the lists.
- Merging (union) of the items of the lists.

Examples of applications:

1. Matching:
   - Master file - bank account info (account number, person name, account balance) - sorted by account number
   - Transaction file - updates on accounts (account number, credit/debit info).

2. Merging:
   - Merging two class lists keeping alphabetic order.
   - Sorting large files (break into small pieces, sort each piece and then merge them).
Matching the Names in Two Lists

<table>
<thead>
<tr>
<th>List 1 (Sorted)</th>
<th>List 2 (Sorted)</th>
<th>Matched List (Sorted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>ADAMS</td>
<td>ADAMS</td>
</tr>
<tr>
<td>CARTER</td>
<td>BECH</td>
<td>CARTER</td>
</tr>
<tr>
<td>CHIN</td>
<td>BURNS</td>
<td>DAVIS</td>
</tr>
<tr>
<td>DAVIS</td>
<td>CARTER</td>
<td></td>
</tr>
<tr>
<td>MILLER</td>
<td>DAVIS</td>
<td></td>
</tr>
<tr>
<td>RESTON</td>
<td>PETERS</td>
<td></td>
</tr>
<tr>
<td>End of list</td>
<td>ROSEWALD</td>
<td>SCHIMT</td>
</tr>
<tr>
<td>Detected</td>
<td></td>
<td>WILLIS</td>
</tr>
</tbody>
</table>

Synchronization:

item(i) = current item from list i

if item(1) < item(2) then
  get next item from list 1
if item(1) > item(2) then
  get next item from list 2
if item(1) = item(2) then
  output the item to output list
  get next item from list 1 and list 2

Handling End-of-File/End-of-List Condition
Halt when we get to the end of either list 1 or list 2.
Merging the Names from Two Lists (Elimin. Repetit.)

List 1 (Sorted) | List 2 (Sorted) | Merged List (Sorted)  
--- | --- | ---  
ADAMS | ADAMS | ADAMS  
CARTER | BECH | BECH  
CHIN | BURNS | BURNS  
DAVIS | CARTER | CARTER  
MILLER | DAVIS | DAVIS  
RESTON | PETERS | PETERS  
<HIGH VALUE> | ROSEWALD | ROSEWALD  
SCHIMT | | SCHIMT  
WILLIS | | WILLIS  
<HIGH VALUE> | |  

Modify the synchronization slightly:

if item(1) < item(2) then
    output item(1) to output list
    get next item from list 1
if item(1) > item(2) then
    output item(2) to output list
    get next item from list 2
if item(1) = item(2) then
    output the item to output list
    get next item from list 1 and list 2
Handling End-of-File/End-of-List Condition

1. Using a `<HIGH VALE>` as in the previous example:

   By storing `<HIGH VALUE>` in the current item for the list that finished, we make sure the contents of the other list is flushed to the output list.
   The stopping criteria is changed to :
   Halt when we get to the end of either list 1 and list 2.

2. Reducing the number of comparisons:

   We can perform a similar algorithm with less comparisons without using a `<HIGH VALUE>` as described above.
   The stopping criteria becomes:
   When we get to the end of either list 1 or list 2, we halt the program.

   Finalization: flush the unfinished list to the output list.

   ```
   while (list 1 did not finish)
       output item(1) to output list
       get next item from list 1

   while (list 2 did not finish)
       output item(2) to output list
       get next item from list 2
   ```
Cosequential Processing: A General Ledger Program

Ledger = A book containing accounts to which debits and credits are posted from books of original entry.

Problem: design a general ledger posting program as part of an accounting system.

Two files are involved in this process:

**Master File:** ledger file
- monthly summary of account balance for each of the book-keeping accounts.

**Transaction File:** journal file
- contains the monthly transactions to be posted to the ledger.

Once the journal file is complete for a given month, the journal must be posted to the ledger.

**Posting** involves associating each transaction with its account in the ledger.
Sample Ledger Fragment

<table>
<thead>
<tr>
<th>Account Number</th>
<th>Account Title</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>checking account #1</td>
<td>1032.00</td>
<td>2114.00</td>
<td>5219.00</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>checking account #2</td>
<td>543.00</td>
<td>3094.17</td>
<td>1321.20</td>
<td></td>
</tr>
<tr>
<td>510</td>
<td>auto expense</td>
<td>195.00</td>
<td>307.00</td>
<td>501.00</td>
<td></td>
</tr>
<tr>
<td>540</td>
<td>office expense</td>
<td>57.00</td>
<td>105.25</td>
<td>138.37</td>
<td></td>
</tr>
<tr>
<td>550</td>
<td>rent</td>
<td>500.00</td>
<td>1000.00</td>
<td>1500.00</td>
<td></td>
</tr>
</tbody>
</table>

Sample Journal Entry

<table>
<thead>
<tr>
<th>Account Number</th>
<th>Check Number</th>
<th>Date</th>
<th>Description</th>
<th>Debit/Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1271</td>
<td>April 2, 01</td>
<td>Auto expense</td>
<td>- 79.00</td>
</tr>
<tr>
<td>510</td>
<td>1271</td>
<td>April 2, 01</td>
<td>Tune-up</td>
<td>79.00</td>
</tr>
<tr>
<td>101</td>
<td>1272</td>
<td>April 3, 01</td>
<td>Rent</td>
<td>- 500.00</td>
</tr>
<tr>
<td>550</td>
<td>1272</td>
<td>April 3, 01</td>
<td>Rent for April</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>670</td>
<td>April 4, 01</td>
<td>Office expense</td>
<td>- 32.78</td>
</tr>
<tr>
<td>540</td>
<td>670</td>
<td>April 4, 01</td>
<td>Printer cartridge</td>
<td>32.00</td>
</tr>
<tr>
<td>101</td>
<td>1273</td>
<td>April 5, 01</td>
<td>Auto expense</td>
<td>- 31.00</td>
</tr>
<tr>
<td>510</td>
<td>1273</td>
<td>April 5, 01</td>
<td>Oil change</td>
<td>31.83</td>
</tr>
</tbody>
</table>
Sample Ledger Printout

101 Checking account #1

1271 | April 2, 01 | Auto expense   | - 79.00
1272 | April 3, 01 | Rent          | - 500.00
1273 | April 5, 01 | Auto expense  | - 31.00
Prev. Bal.: 5,219.00 New Bal.: 4,609.00

102 Checking account #2

510 Auto expense

540 Office expense

550 Rent
How to implement the Posting Process?

- Use account number as a **key** to relate journal transactions to ledger records.

- Sort the journal file.

- Process ledger and sorted journal **co-sequentially**.

Tasks to be performed:

- Update ledger file with the current balance for each account.

- Produce printout as in the example.

From the point of view of ledger account:

Merging (unmatched accounts go to printout)

From the point of view of journal account:

Matching (unmatched accounts in journal constitute an error)

The posting method is a combined merging/matching.
Ledger Algorithm

Item(1): always stores the current master record
Item(2): always stores the current transactions record

- Read first master record
- Print title line for first account
- Read first transactions record

While (there are more masters
  or there are more transactions) {
  if item(1) < item(2) then {
    Finish this master record:
    - Print account balances, update master record
    - Read next master record
    - If read successful, then print title line for
      new account
  }
  if item(1) = item(2) {
    Transaction matches master:
    - Add transaction amount to the account balance
      for new month
    - Print description of transaction
    - Read next transaction record  }
  if item(1) > item(2) {
    Transaction with no master:
    - Print error message
    - Read next transaction record  }
}