*CSCI3180 Principles of Programming Languages*

# Tutorial 1

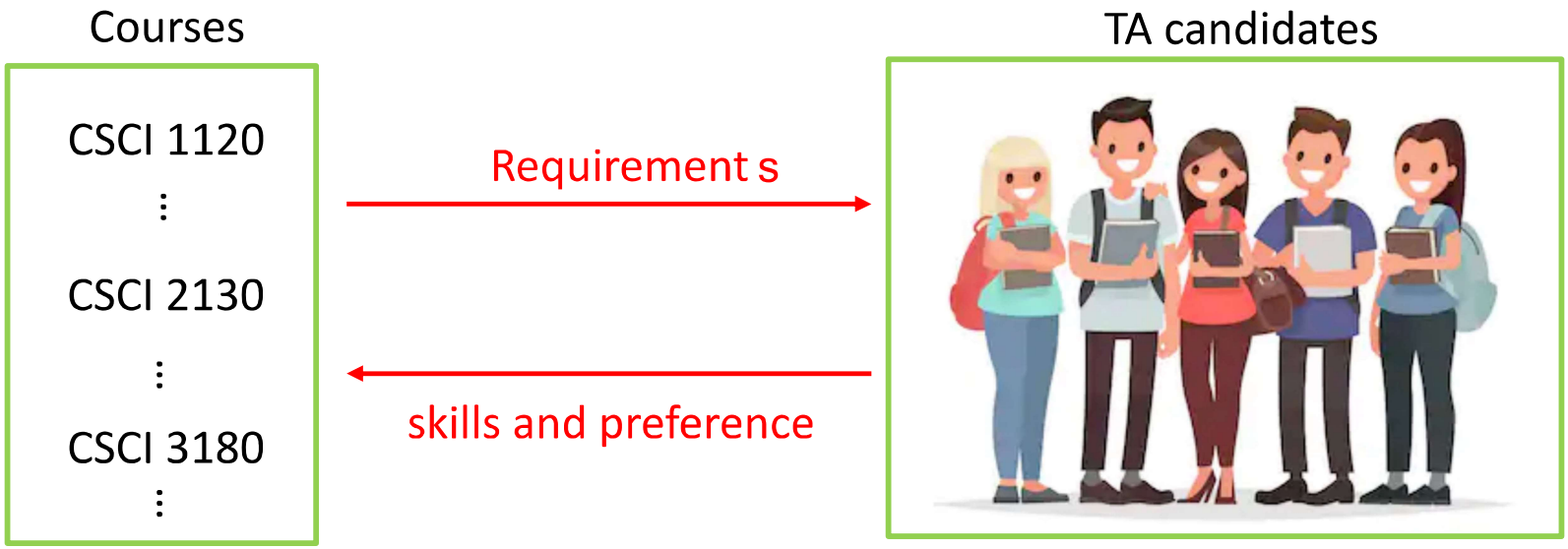
- Assignment 1
- COBOL Programming

# Assignment 1

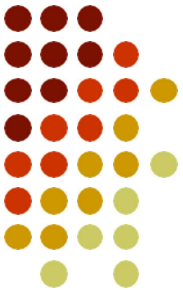
# Background

## TA Matching System

- To find the most suitable TAs for each course



# Introduction



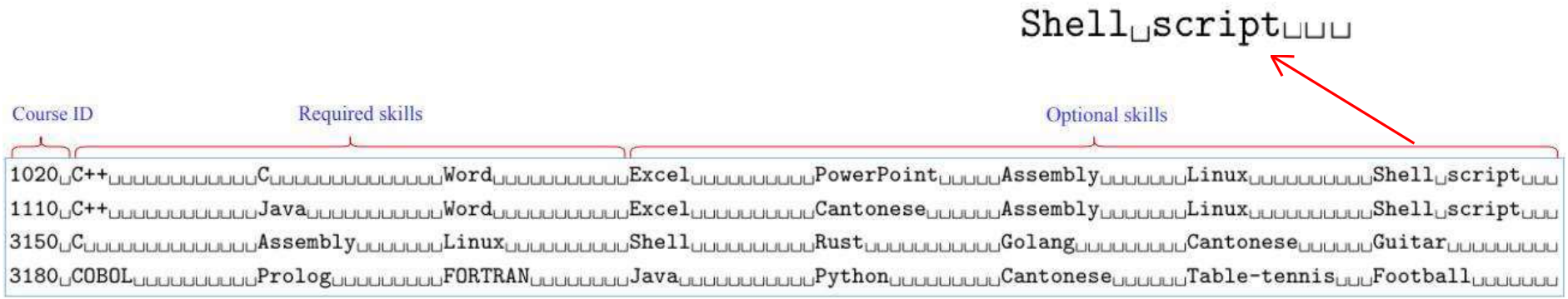
**TA Ranking Module of the Matching System**

- Two inputs
  - The courses requirements
  - The candidate TA' skills and preferences
- One output
  - Top 3 TAs for each course
    - TAs are ranked by matching scores

# Inputs

**The courses requirements (instructors.txt)**

- Course ID (4 digit numbers followed by a space)
- 3 required skills and 5 optional skills
  - Skill names are padded spaces so that each skill name takes exactly 15 characters.



# Inputs

**The candidate TA' skills and preference (candidate.txt)**

- TA ID (10 digit numbers followed by a space)
- 8 skills
  - A string of 15 characters (spaces are padded).
- 3 preferences

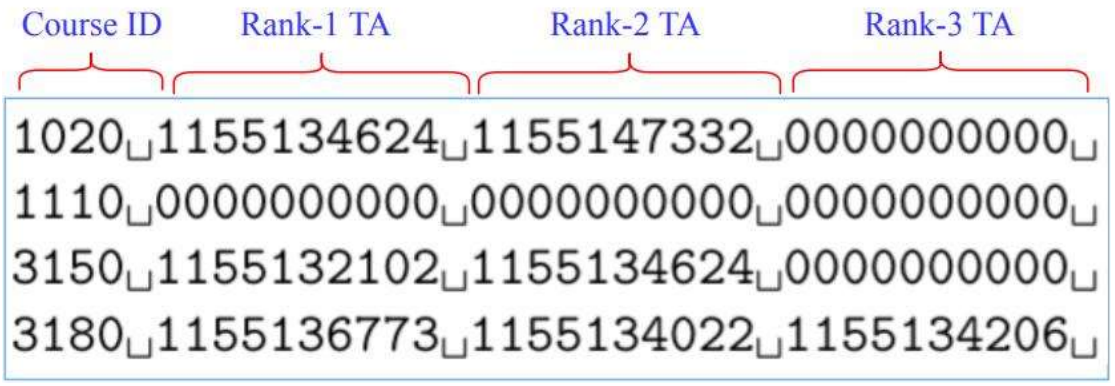
TA ID | Skills | 1st | 2nd | 3rd

```
1155132102 C              Assembly       Linux          Shell script   Rust           Golang         Cantonese      Guitar         3150 3180 1110
1155134022 COBOL          Prolog         FORTRAN        Java           Python         Cantonese      Table-tennis   Football       3320 2110 3180
1155134206 COBOL          Prolog         FORTRAN        Java           Python         Japanese       Table-tennis   Football       2110 3180 3320
1155134624 C++            C              Word           Excel          PowerPoint     Assembly       Linux          Shell script   1020 3150 2102
1155136773 COBOL          Prolog         FORTRAN        Java           Python         Cantonese      Table-tennis   Football       3180 3320 1110
1155147332 C++            C              Word           Excel          PowerPoint     Cantonese      Rust           Football       3180 3150 1020
```

# Output

**output.txt**

- Course ID
- 3 TA IDs



# Matching Score

score(course, TA)= { 1+skill_score+preference_score if all the required skills are satisfied; 0 otherwise

skill_score: number of optional skills satisfied by the TA.

preference_score: TA's preference to the score, according to the following table.

| | 1st preference | 2nd preference | 3rd preference |
|---|---|---|---|
| *preference_score* | 1.5 | 1 | 0.5 |

For example:

Required skills | Optional skills

instructors.txt

| 3180 | COBOL | Prolog | FORTRAN | Java | Python | Cantonese | Table-tennis | Football |
|---|---|---|---|---|---|---|---|---|

candidates.txt

| 1155136773 | COBOL | Prolog | FORTRAN | Java | Python | Cantonese | Table-tennis | Football | 3180 | 3150 | 1020 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1152147332 | C++ | C | Word | Excel | PowerPoint | Cantonese | Rust | Football | 3180 | 3150 | 1020 |

score(3180, 1155136773) = 1 + 5 + 1.5 = 7.5

score(3180, 1152147332) = 0

# COBOL Programming

# Introduction

**COBOL** – **CO**mmon **B**usiness **O**riented **L**anguage

- Advantages:
  - Designed for batch processing
  - Emphasis on the **data formats** (file format and intermediate data structures)
  - The syntax is “close” to English

# Introduction

**COBOL** – **CO**mmon **B**usiness **O**riented **L**anguage

- Disadvantages:
  - Less “structured”
  - Verbose

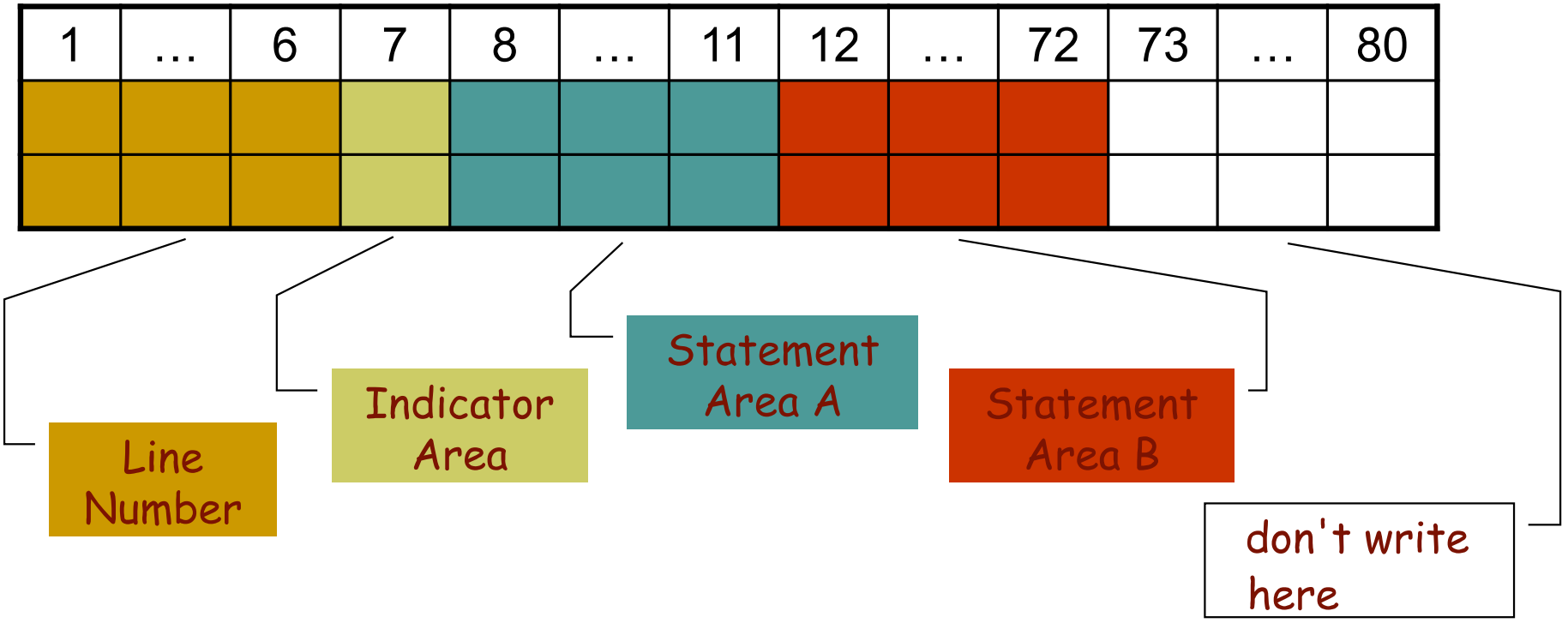
*“The use of COBOL cripples the mind; its teaching should, therefore be regarded as a criminal offense.”*

– Edsger Dijkstra

Feel it youself ☺

# Statement Format

- Format is FIXED!



# Ancient COBOL Coding Form

Program: PROG1 Requested by: Page 2 of 3

Programmer: ROBERT LIEB Date: 1-15-1998 Identification

| Sequence (Page) | Sequence (Serial) | Cont | A | B | COBOL Statement |
|---|---|---|---|---|---|
| | 01 | | WORKING-STORAGE SECTION. | | |
| | 02 | | 01 | DATA-REMAINS-SWITCH | PIC X(02) VALUE SPACES. |
| | 03 | | | | |
| | 04 | | 01 | HEADING-LINE. | |
| | 05 | | | 05 FILLER | PIC X(10) VALUE SPACES. |
| | 06 | | | 05 FILLER | PIC X(12) VALUE 'STUDENT NAME'. |
| | 07 | | | 05 FILLER | PIC X(10) VALUE SPACES. |
| | 08 | | | | |
| | 09 | | 01 | DETAIL-LINE. | |
| | 10 | | | 05 FILLER | PIC X(08) VALUE SPACES |
| | 11 | | | 05 PRINT-NAME | PIC X(25). |
| | 12 | | | 05 FILLER | PIC X(10) VALUE SPACES. |
| | 13 | | | | |
| | 14 | | PROCEDURE DIVISION. | | |
| | 15 | | PREPARE-SENIOR-REPORT. | | |
| | 16 | | | OPEN INPUT STUDENT-FILE | |
| | 17 | | | OUTPUT PRINT-FILE. | |
| | 18 | | | READ STUDENT-FILE | |
| | 19 | | | AT END MOVE 'NO' TO DATA-REMAINS-SWITCH | |
| | 21 | | | END-READ. | |
| | 22 | | | PERFORM WRITE-HEADING-LINE. | |
| | 23 | | | PERFORM PROCESS-RECORDS | |
| | 24 | | | UNTIL DATA-REMAINS-SWITCH = 'NO'. | |
| | 25 | | | CLOSE STUDENT-FILE | |
| | 26 | | | PRINT-FILE. | |
| | 27 | | | STOP-RUN. | |
| | 28 | | | | |
| | 29 | | | | |
| | 30 | | | | |

# Hello World!

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID.    HELLO-WORLD-PROG.
000030 AUTHOR.        TIMOTHY R P BROWN.
000040*The standard Hello world program
000050
000060 ENVIRONMENT DIVISION.
000070
000080 DATA DIVISION.
000090 WORKING-STORAGE SECTION.
000100 01 TEXT-OUT    PIC X(12) VALUE 'Hello World!'.
000110
000120 PROCEDURE DIVISION.
000130 MAIN-PARAGRAPH.
000140         DISPLAY TEXT-OUT.
000150         DISPLAY
000160         'hElLo WoRlD!'
000170         ' ', TEXT-OUT.
000180         STOP RUN.
```

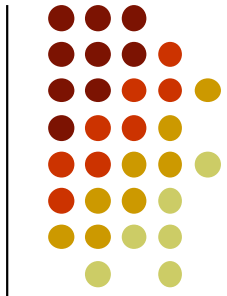
Case insensitive
Usually use capital letters

A statement can span across several lines

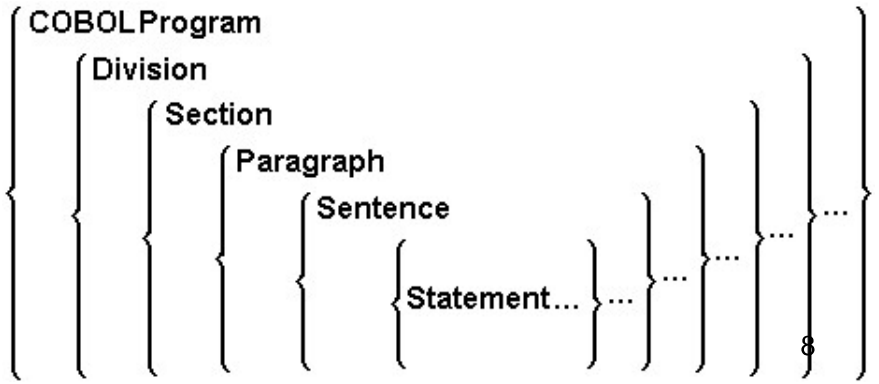
Scope terminator

MANY reserved words with MANY variations!

# Program Structure



- 4 divisions:
  - Identification Division (required)
  - Environment Division (optional)
  - Data Division (optional)
    - File Section
    - Working Storage Section
  - Procedure Division (optional)



# 4 Divisions of “Hello World”

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID.    HELLO-WORLD-PROG.
000030 AUTHOR.        TIMOTHY R P BROWN.
000040*The standard Hello world program
000050
000060 ENVIRONMENT DIVISION.
000070
000080 DATA DIVISION.
000090 WORKING-STORAGE SECTION.
000100 01 TEXT-OUT    PIC X(12) VALUE 'Hello World!'.
000110
000120 PROCEDURE DIVISION.
000130 MAIN-PARAGRAPH.
000140        DISPLAY TEXT-OUT.
000150        DISPLAY
000160        'hElLo WoRlD!',
000170        ' ', TEXT-OUT.
000180        STOP RUN.
```

# Environment Division

- Configuration & I/O

```
000260 ENVIRONMENT DIVISION.
000270 CONFIGURATION SECTION.
000280 SOURCE-COMPUTER. IBM PC.
000290 OBJECT-COMPUTER. IBM PC.
000300 INPUT-OUTPUT SECTION.
000310 FILE-CONTROL.
000320     SELECT INPUT-FILE ASSIGN TO 'input.txt'
000330       ORGANIZATION IS LINE SEQUENTIAL.
```

optional

**Start** in Area A

**Start** in Area B

Be careful Area B starts from column 12!

# Environment Division

- File I/O section
  - Assign identifiers to files/printers
  - Syntax

```
000320      SELECT identifier ASSIGN TO filename
000330        ORGANIZATION IS LINE SEQUENTIAL.
```

Declared in data division

No Period

# Data Division

- File, working-storage & linkage section

```
000400 DATA DIVISION.
000410 FILE SECTION.
000420 FD INPUT-FILE.
000440 01 CUSTOMER-DATA.
000450     03 NAME PIC X(12).
000460     03 ADDRESS.
000470         05 HOUSE-NUMBER PIC 99.
000480         05 STREET PIC X(19).
000500 WORKING-STORAGE SECTION.
000510 01 RECORD-COUNTER PIC 9(5) VALUE ZERO.
```

**Start** in Area A

FILE SECTION

goes first.

Where we define the data we want to manipulate

# Data Division – File Section

- File consists of records
  - Business-oriented language

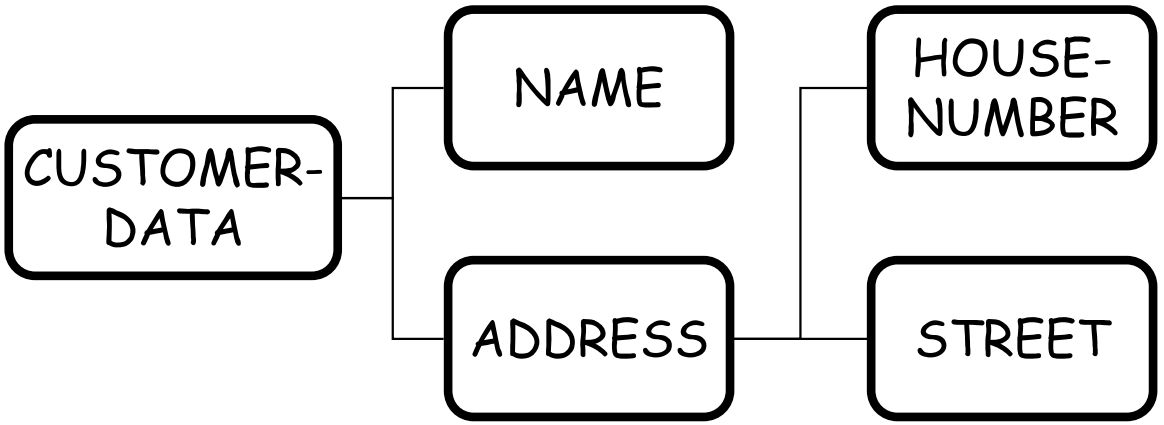
```
000420 FD INPUT-FILE.
000440 01 CUSTOMER-DATA.
000450     03 NAME PIC X(12).
000460     03 ADDRESS.
000470         05 HOUSE-NUMBER PIC 99.
000480         05 STREET PIC X(19).
```

States the file being defined; in Area A

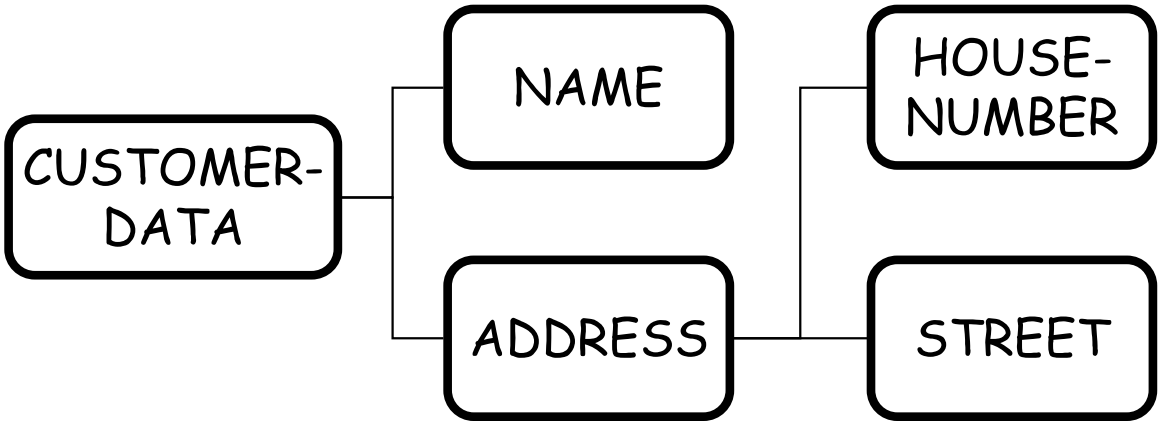
Record name; in Area A

Field name; in Area B

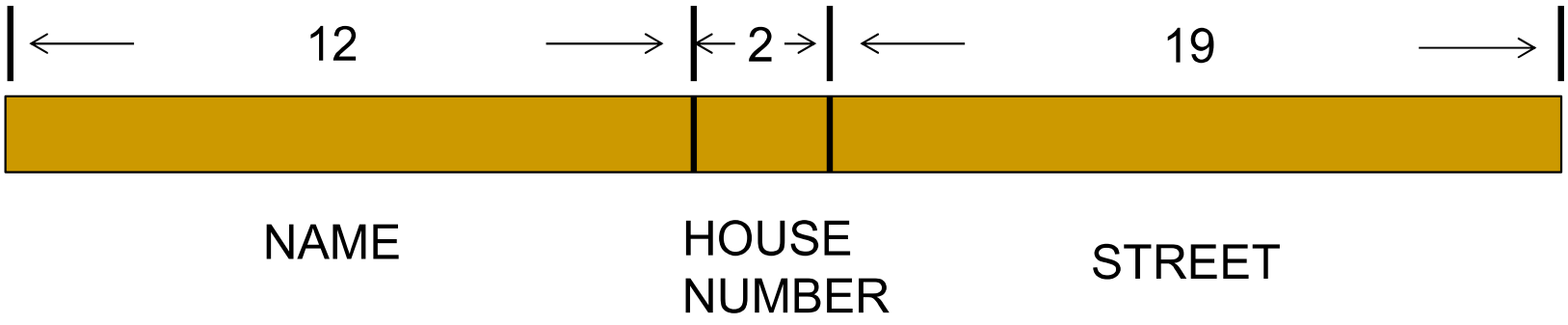
Sub-field name; in Area B



# Data Division – File Section




Each row in the file:



# Data Division – Working Storage Section

- Define variables

```
000420 FD INPUT-FILE.
000440 01 CUSTOMER-DATA.
000450     03 NAME PIC X(12).
000460     03 ADDRESS.
000470         05 HOUSE-NUMBER PIC 99.
000480         05 STREET PIC X(19).
```

- Level number
  - 01 for record name (Area A)
  - Other numbers for fields (Area B)
    - Larger numbers for deeper levels
  - 66, 77, 88 reserved

# Data Division – Variable Declaration

- Identifier
  - Max. 30 characters
  - Alphanumeric & hyphen
- Data Type
  - Character
    - `01 INPUT-STR PIC XXXX VALUES 'ABCD'.`
    - `01 INPUT-STR PIC X(4) VALUES SPACES.`
  - Number
    - `01 INPUT-NUM PIC 9999 VALUES 1000.`
    - `01 INPUT-NUM PIC 9(4) VALUES ZERO.`
    - `01 INPUT-NUM PIC 99V99 VALUES ZERO.`

width

decimal point

# Table (array)

- **One-dimensional**

```
01 BOOK-ID-TABLE.
    03 BOOK-ID PIC 9(5) OCCURS 5 TIMES.
```

- **Usage**

```
MOVE 24781 TO BOOK-ID(3)
MOVE ZEROS TO BOOK-ID-TABLE
```

- **Multi-dimensional**

```
01 SALES-TABLE.
    03 BRANCH-NO OCCURS 4.
        05 SALES PIC 9(4) OCCURS 5.
```

Here a two-dimensional array with 4 rows and 5 columns is defined

Notes: OCCURS clause CANNOT be specified for Level 01 or 77 items

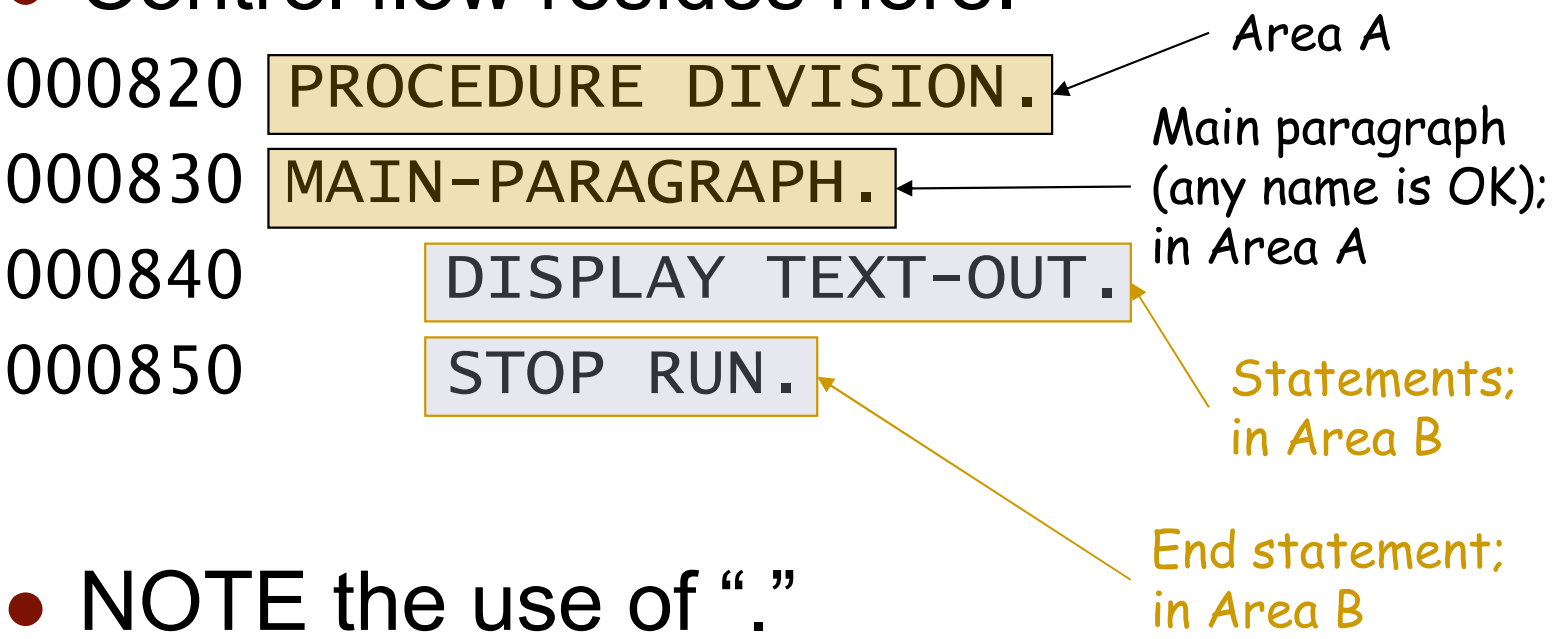
# Boolean Data Type

- NUMBER-SIZE for assignment
  - MOVE 'Y' TO NUMBER-SIZE
  - MOVE 'N' TO NUMBER-SIZE
- BIG-NUMBER for evaluation
  - IF BIG-NUMBER THEN …
- Multi-level boolean

```
01 GRADES-CHECK PIC 999.
    88 A-GRADE VALUE 70 THRU 100.
    88 B-GRADE VALUE 60 THRU 69.
    88 C-GRADE VALUE 50 THRU 59.
    88 FAIL-GRADE VALUE 0 THRU 49.
```

# Procedure Division

- Control flow resides here.




- NOTE the use of “.”

# Assignment Statement

- Syntax

  MOVE *value* TO *variable*

- Example

```
01 DATE-IN.
     03 W-DAY PIC 99.
     03 W-MONTH PIC 99.
...
     MOVE 31 TO W-DAY
     MOVE 12 TO W-MONTH IN DATE-IN
```

# Arithmetic Statement

- Simple arithmetic statement

```
ADD value1 TO value2 GIVING variable
SUBTRACT value2 FROM value1 GIVING variable
MULTIPLY value1 BY value2 GIVING variable
DIVIDE value1 BY value2 GIVING variable
```

- General computation statement

```
COMPUTE variable = arithmetic expression
```

  - Operators: +, -, *, /, **

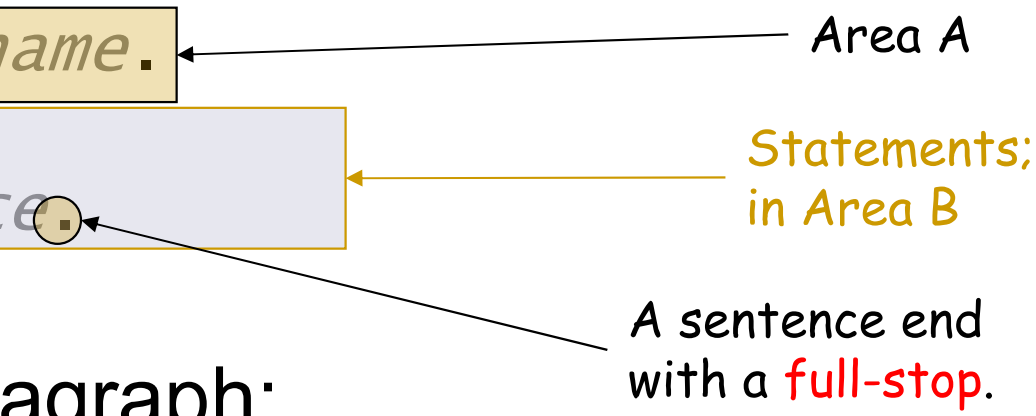
Example:

```
COMPUTE A = (C + B) * D
```

# Paragraph

- Subroutine in COBOL
- No parameter passing
  - Use global variable
  - The scope ends until the occurrence of next paragraph
- Syntax

```
000930 paragraph-name.
000940     ...
000950     sentence.
```




- To invoke the paragraph:

```
PERFORM pname. (return to the caller)
GO TO pname. (not returning, keep going)
```

# GO TO Statement

- Syntax

  GO TO *paragraph-name*

- Example

```
000010 PROCEDURE DIVISION.
000020 MAIN-PARAGRAPH.
000030     DISPLAY TEXT-OUT.
000040     GO TO SUB-PARAGRAPH.
000050     STOP RUN.
000060 SUB-PARAGRAPH.
000070     DISPLAY 'This is a sub-paragraph!'.
```

# Selection Statement

***(ELSE Phrase is Not allowed in Asg1!!!)***

- Syntax

```
IF condition THEN
   statements 1
ELSE
   statements 2
END-IF
```

**In Assignment 1**

```
IF condition THEN
   statements
END-IF
```

**In Assignment 1**

- Logical operators
  - NOT, AND, OR
- Relational operators
  - =, >, <, <=, >=, NOT =

# Repetition Statement – Until Loop *(Not allowed in Asg1!!!)*

- Until Loop

```
PERFORM UNTIL condition
   ...
END-PERFORM
```

- Do-Until Loop

```
PERFORM WITH TEST AFTER
   UNTIL condition
   ...
END-PERFORM
```

# Repetition Statement – For Loop *(Not allowed in Asg1!!!)*

- Version 1

```
PERFORM n TIMES
    ...
END-PERFORM
```

- Version 2

```
PERFORM VARYING counter FROM
    initial BY step UNTIL condition
    ...
END-PERFORM
```

# IF Statement with multiple statements for a condition

- Syntax

```
IF condition THEN
   statements 1
   statements 2
END-IF.
```

```
IF condition1 THEN
   statements 1
   IF condition2 THEN
      statements 2
      statements 3
   END-IF
END-IF.
```

"." terminates the scope.
Use period only at the last END-IF of a nested block

# File I/O – Opening & Closing

- File Opening
  - `OPEN` *`mode filename1 filename2 ...`*
  - Mode - `INPUT`, `OUTPUT`, `I-O`, `EXTEND`
- File Closing
  - `CLOSE` *`filename1 filename2 ...`*

# File I/O - Reading

- Suppose the file structure is:

```
Grace Hopper   000001
William Selden 123456
...
```

- Define the file record structure

```
FD IN-FILE.
01 CUSTOMER-DETAILS.
    03 CUS-NAME       PIC X(15).
    03 CUS-NUM        PIC 9(6).
```

# File I/O - Reading

- To read one record of the file:

  `READ IN-FILE.` ←

  Note it is the identifier **NOT** the record name

  - `CUS-NAME` contains customer name
  - `CUS-NUM` contains customer number
  - To read the next record, execute the statement again.

  You can also use the READ statement in the following form:

  ```
  READ IN-FILE INTO CUSTOMER-DETAILS.
  ```

# File I/O - Writing

- Suppose the file structure is:

```
Grace Hopper    000001
William Selden 123456
...
```

- Define the file record structure

```
FD OUT-FILE.
01 CUSTOMER-DETAILS.
    03 CUS-NAME        PIC X(15).
    03 CUS-NUM         PIC 9(6).
```

# File I/O - Writing

- To write one record into the file:

```
MOVE 'william Selden' TO CUS-NAME.
MOVE 123456 TO CUS-NUM.
WRITE CUSTOMER-DETAILS.
```

Note it is the the record name **NOT** identifier

# GNUCobol 1.1 (Windows)

- Use computers in SHB924/904
- Map network drive with folder name \\ntsvr1\userapps (usually it is already mounted as the drive S:
- Go to S: \OpenCOBOL and run set_env.bat
- To compile and produce executable:

```
cobc -x asg.cob
```

# Other platforms

- Installing GNUCOBOL
  - Linux: compile from source
  - Mac: try brew install

**Assignments will be graded on computers in SHB924/904!!!**

# Resources

Tutorials:

- http://cobol.404i.com/
- https://www.tutorialspoint.com/cobol/
- https://open-cobol.sourceforge.io/ (GnuCobol Manual 2.2)
- http://www.csis.ul.ie/cobol/

- Best Consultant:

Google