CS 2230
Programming in COBOL

Lily Chang, Assistant Professor
Computer Science and Software Engineering

Chapter 9
Working with Characters

STRING statement

```
STRING {{identifier-1}}... DELIMITED BY {identifier-2}}...  INTO identifier-3
        {literal-1   }                  {literal-2   }
                                        {SIZE        }
[WITH POINTER identifier-4]
[ON OVERFLOW imperative-statement-1]
[NOT ON OVERFLOW imperative statement-2]
[END-STRING]
```

- The String statement *strings*, or *concatenates*, two or more sending fields into one receiving field.

STRING Statement

- To combine or concatenate several fields into one
- Consider following entries:

```
05  NAME.
    10    LAST-NAME    PIC X(10).
    10    FIRST-NAME   PIC X(10).
    10    MIDDLE-NAME  PIC X(6).
```

STRING Statement

- Suppose name fields have values below

Last-Name									
E	D	I	S	O	N				

First-Name									
T	H	O	M	A	S				

Middle-Name					
A	L	V	A		

- Print name with single blank between parts: `THOMAS ALVA EDISON`

STRING Statement

- identifier-1 or literal-1 is field or value to be combined
- identifier-3 is field in which all identifiers or literals are combined
- DELIMITED BY clause
  - Transmitting of characters ends when value of identifier-2 or literal-2 encountered
  - SIZE means entire contents to be copied

STRING Example

NEW YORK□□□□□□□□□□NY11753

```
STRING
    CITY   DELIMITED BY ' '
    ', '   DELIMITED BY SIZE
    STATE  DELIMITED BY ' '
    ' '    DELIMITED BY SIZE
    ZIP    DELIMITED BY SIZE
    INTO   ADDR-OUT
END-STRING
```

DISPLAY ADDR-OUT ➔ NEW YORK, NY 11753

STRING Example

- To insert a blank between parts of name

```
STRING
    FIRST-NAME DELIMITED BY ' '
    ' ' DELIMITED BY SIZE
    MIDDLE-NAME DELIMITED BY ' '
    ' ' DELIMITED BY SIZE
    LAST-NAME DELIMITED BY ' '
    INTO NAME-OUT
```

Places a blank after each field

OVERFLOW Option

- Specifies operation(s) to be performed if receiving field not large enough to accommodate result
- NOT ON OVERFLOW option may also be used
- END-STRING scope terminator also available

POINTER Option

- To count number of characters moved in STRING statement
- Increments specified field by one for every character moved

POINTER Option Example

```
MOVE 1 TO WS-COUNT
STRING FIRST-NAME DELIMITED BY ' '
     INTO NAME-OUT
     WITH POINTER WS-COUNT
```

- If FIRST-NAME is 'Paul', WS-COUNT is 5 after STRING
- Subtract one from WS-COUNT to get length of actual move (4)

POINTER Option

- Also used to move data to receiving field beginning at some point other than first position
- If WS-COUNT initialized to 15 before STRING, FIRST-NAME moved to NAME-OUT beginning with 15TH position of NAME-OUT

STRING Statement Rules

- DELIMITED BY clause required
- Receiving field must be elementary data item - no editing symbols or JUSTIFIED RIGHT clause
- All literals must be nonnumeric
- Identifier with POINTER clause must be elementary
- Moves data left to right

13

UNSTRING Statement

- To separate a field into its components
- Suppose NAME-IN equals

```
TAFT, WILLIAM, H
```

- The last name, first name and middle initial can be stored in separate fields without commas

14

UNSTRING Example

```
UNSTRING NAME-IN
    DELIMITED BY ‘,’
       INTO LAST-NAME
            FIRST-NAME
            MIDDLE-INITIAL
```

- TAFT will be stored in LAST-NAME, `William` in FIRST-NAME and H in MIDDLE-INITIAL

15

UNSTRING Statement Format

```
UNSTRING identifier-1
      DELIMITED BY [ALL]   { identifier-2 }
                           { literal-1    }

      [ OR [ALL]  { identifier-3 } ] ...
      [           { literal-2    } ]

      INTO identifier-4 …
END-UNSTRING
```

16

UNSTRING Statement

- Sending field, as well as literal, must be nonnumeric
- Receiving fields may be numeric or nonnumeric
- ALL phrase means one or more occurrences of literal or identifier treated as one
- POINTER and OVERFLOW clauses may also be used

17

Reference Modification

```
identifier (offset:length)
```

- Within the parentheses, you code the starting position of the first character in the field that you want to work with (the *offset*) and the number of characters that you want to work with (the *length*)
- If you omit the length, all remaining characters in the field are referred to

18

Reference Modification in MOVE

Example:

```
MOVE SOCIAL-SECURITY-NUMBER (4:2)
     TO USER-PASSWORD (3:2)
MOVE "*" TO WORK-FIELD (20:1)
MOVE ZIP-CODE TO ADDRESS-LINE-3 (15:)
MOVE FULL-NAME (OFFSET-FIELD:LENGTH-FIELD)
     TO LAST-NAME.
MOVE WORK-FIELD (1:COUNT-2)
     TO EDITED-FIELD (6 – COUNT-2:)
```

19

Data Validation

- Avoiding logic errors by validating input
- Even if you have a bug-free program
  - Garbage in Garbage out

20

Program Errors - Syntax

- Due to violations of language rules
- Detected by compiler

21

Program Errors - Logic

- Result from using incorrect instructions or incorrect sequence of instructions
- Also include run-time errors
- Not detected during compilation
- Found by running program with test data and comparing outcome to expected results

22

Avoiding Logic Errors

- Develop comprehensive test data
- Include all condition and types of data tested for in program
- Have someone other than programmer prepare test data to avoid bias
- Manually check computer-produced results for accuracy

23

Debugging Tips

- For every IF statement, include test data that satisfies and does not satisfy condition
- For multi-page report include enough test data to print several pages
- Include test data that produces size errors if ON SIZE ERROR routines are used

24

Debugging Tips

- Used DISPLAY statements during test runs to isolate logic errors
- If program produces disk file, always examine it for accuracy
- Check loops to see that instructions performed exact number of times required

Why Input Must Be Validated

- Risk of data entry errors is high
  - Large volume of data entered
  - Human error keying in data
- Invalid input leads to inaccurate output
  - For example, salary reported incorrectly if entered as 23000 instead of 32000
- Input error can cause program interrupt
  - For example, spaces entered for numeric field used in arithmetic operation

Data Validation Techniques

- Routines to identify various types of input errors
- Error modules to handle each error that occurs

Test Fields for Correct Format

- Use NUMERIC class test to ensure field used in arithmetic operation has numeric value

Example

```
IF AMT IS NOT NUMERIC
    PERFORM 500-ERR-RTN
ELSE
   ADD AMT TO WS-TOTAL
END-IF
```

Test Fields for Correct Format

- Use ALPHABETIC class test if field must be alphabetic
- Use sign test if numeric field is to have
  - Values greater than zero (POSITIVE)
  - Values less than zero (NEGATIVE)
  - Value equal to zero (ZERO)
    - **S** must be included in PIC to store a negative number

Checking for missing data

- Check key fields if they must contain data

Example

```
IF SOC-NO = SPACES
     PERFORM 900-ERR-RTN
END-IF.
```

Testing for Reasonableness

- Use after verifying that numeric fields contain numeric data
- Range test - check that field is within established lower and upper bounds
- Limit test - check that field does not exceed defined upper limit

31

Checking Coded Fields

- Code often stored in field to shorten record and minimize typing
- For example, 'H' or 'S' may represent pay type of 'Hourly' or 'Salaried'
- Use condition names to check validity of coded fields

32

Checking Coded Fields

Example

```
05 Pay-Code       Pic X.
   88  Hourly     Value 'H'.
   88  Salaried   Value 'S'.
```

Data Division entries

```
If Hourly Or Salaried
    Perform Pay-Calc-Rtn
Else
    Perform Pay-Code-Err-Rtn
End-If.
```

Procedure Division statements

33

Typical Validity Checks

- Class test - determine if field contains appropriate type of data (NUMERIC, ALPHABETIC)
- Determine if data is missing by comparing field to SPACES
- Replace spaces in numeric fields with ZEROS using INSPECT statement

34

Actions If Input Errors Occur

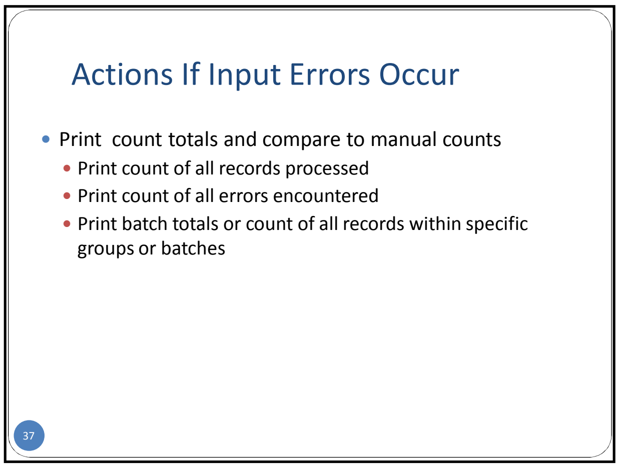
- Print error record displaying key field, field in error and error message
- Stop the run to preserve data integrity
- Partially process or bypass erroneous records
- Stop the run if number of errors exceeds predetermined limit

35

Actions If Input Errors Occur

- Use switch or field to indicate when record contains error
  - Initialize field to 'N' for no errors
  - Set field to 'Y' anytime an error occurs
  - Process record as valid only if switch field still 'N' after all validation checks

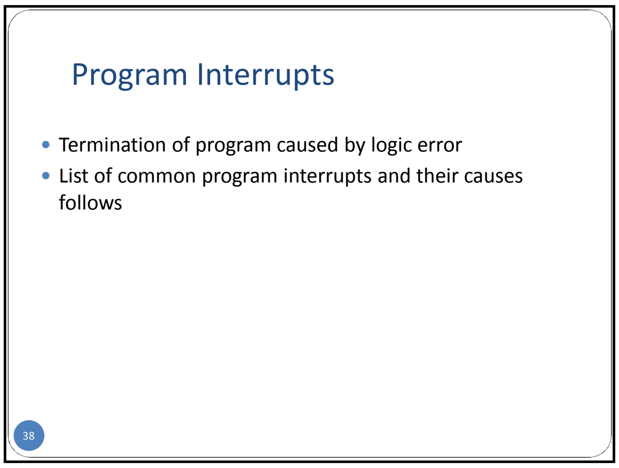
36



Actions If Input Errors Occur

- Print count totals and compare to manual counts
  - Print count of all records processed
  - Print count of all errors encountered
  - Print batch totals or count of all records within specific groups or batches

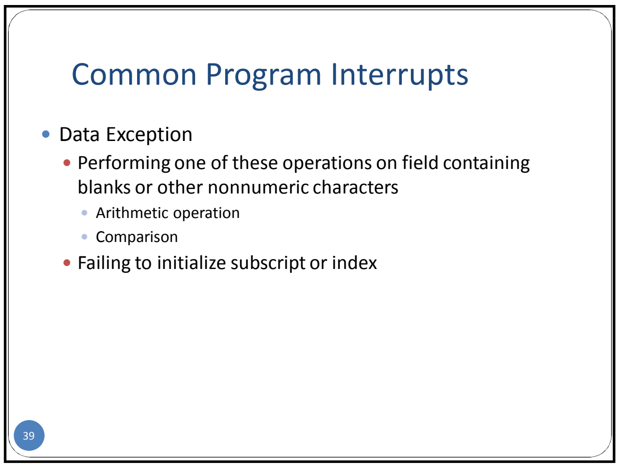
37



Program Interrupts

- Termination of program caused by logic error
- List of common program interrupts and their causes follows

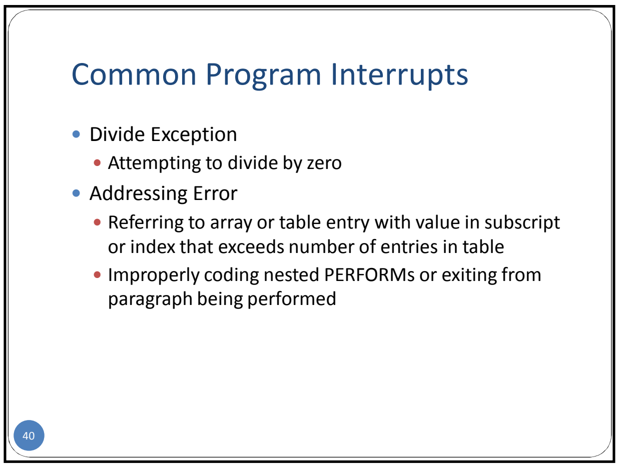
38



Common Program Interrupts

- Data Exception
  - Performing one of these operations on field containing blanks or other nonnumeric characters
    - Arithmetic operation
    - Comparison
  - Failing to initialize subscript or index

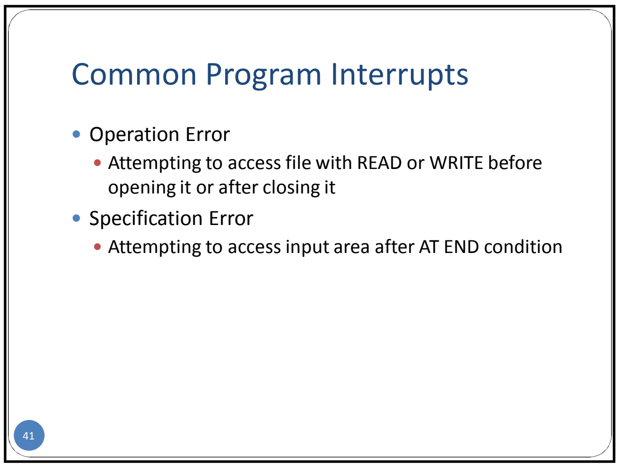
39



Common Program Interrupts

- Divide Exception
  - Attempting to divide by zero
- Addressing Error
  - Referring to array or table entry with value in subscript or index that exceeds number of entries in table
  - Improperly coding nested PERFORMs or exiting from paragraph being performed

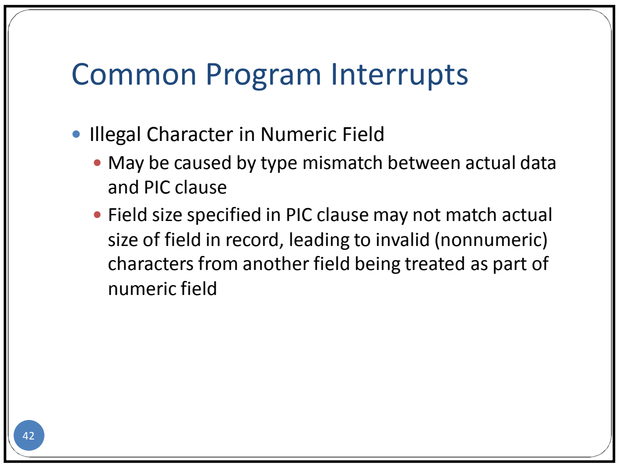
40



Common Program Interrupts

- Operation Error
  - Attempting to access file with READ or WRITE before opening it or after closing it
- Specification Error
  - Attempting to access input area after AT END condition

41



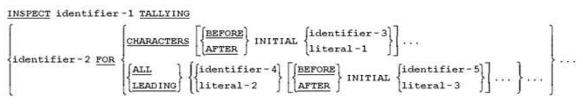
Common Program Interrupts

- Illegal Character in Numeric Field
  - May be caused by type mismatch between actual data and PIC clause
  - Field size specified in PIC clause may not match actual size of field in record, leading to invalid (nonnumeric) characters from another field being treated as part of numeric field

42

INSPECT statement

- Count specific characters in the field that’s inspected



Examples

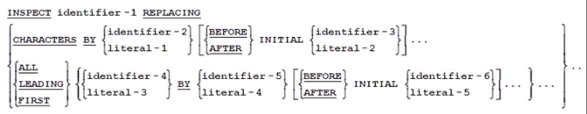
Statement	WORK-FIELD X(8)	COUNT-1 S9(3)	COUNT-2 S9(3)
`INSPECT WORK-FIELD TALLYING`	ANNE	+4	N/A
`    COUNT-1 FOR CHARACTERS BEFORE SPACE.`			
`INSPECT WORK-FIELD TALLYING`			
`    COUNT-1 FOR CHARACTERS BEFORE ","`			
`    COUNT-2 FOR CHARACTERS AFTER ".".`	1,234.56	+1	+2
`INSPECT WORK-FIELD TALLYING`			
`    COUNT-1 FOR ALL ","`			
`                ALL "."`			
`    COUNT-2 FOR CHARACTERS.`	1,234.56	+2	+6
`INSPECT WORK-FIELD TALLYING`			
`    COUNT-1 FOR LEADING "*"`			
`    COUNT-2 FOR CHARACTERS`	***12.**	+3	+5
`INSPECT WORK-FIELD TALLYING`			
`    COUNT-1 FOR ALL "*" BEFORE "."`			
`    COUNT-2 FOR CHARACTERS AFTER "."`	***12.**	+3	+2

INSPECT statement with TALLYING clause

- CHARACTERS refers to all of the characters in the field that’s being inspected
- ALL refers to all occurrences of the specified character or characters
- LEADING refers to all occurrences of the specified character or characters at the start of the field
- You must set the count fields to the starting value you want before issuing the Inspect statement
- A character is only counted once even if it is identified by more than one phrase in the TALLYING clause
- BEFORE or AFTER clause can only specify one character

INSPECT statement with REPLACING clause

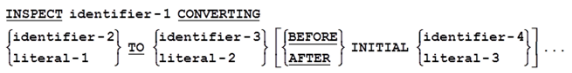
- Replace specific characters in the field that’s inspected.



Examples

Statement	WORK-FIELD X(9) Before	WORK-FIELD X(9) After
`INSPECT WORK-FIELD REPLACING`	$1234.567	$1234.000
`    CHARACTERS BY "0" AFTER ".".`		
`INSPECT WORK-FIELD REPLACING`	***123 CR	000123
`    LEADING "*" BY ZERO`		
`    ALL "CR" BY "  "`		
`    ALL "DB" BY "  ".`		
`INSPECT WORK-FIELD REPLACING`	BACK.BACK	baCK.BAcK
`    ALL "A" BY "a" BEFORE "."`		
`    FIRST "B" BY "b"`		
`    ALL "C" BY "c" AFTER "."`		

INSPECT statement with CONVERTING clause



Statement	WORK-FIELD X(9) Before	WORK-FIELD X(9) After
`INSPECT WORK-FIELD CONVERTING`		
`    "*" TO "0".`	**123.56*	00123.560
`INSPECT WORK-FIELD CONVERTING`		
`    "*" TO "0" BEFORE ".".`	**123.56*	00123.56*
`INSPECT WORK-FIELD CONVERTING`		
`    "ABC" TO "abc".`	BACK.BACK	bacK.bacK
`INSPECT WORK-FIELD CONVERTING`		
`    ".," TO ",.".`	12.345,67	12,345.67