IBM IT Education Services

Session ID: E54

Speaker : Janice Winchell
MICOM Inc

Fun Times in the land of *Dump Reading*

.....and Language Environment

November 10 - 12, 2003 | Hilton, Las Vegas, NV

RETURN TO INDEX

The information presented in this material is for illustration only. There is no implied warranty or correctness for the usability or applicability of the material contained herein. Examples of programming techniques are for demonstration of language elements, and may not necessarily be applicable to specific programming requirements.

Session Objectives

Understand and identify

- WHAT TO DO when you get an ABEND/DUMP
- HOW TO GET the information just in case you DO ABEND/DUMP
- COBOL options that can HELP YOUR DEBUGGING LIFE
- Language Environment RUN-TIME options that can help your DEBUGGING LIFE
- COBOL tricks for debugging
- LE tools for debugging
- Where to FIND THE INFORMATION YOU NEED

Terms & Definitions

- **Program Exception**
  - **Conditions detected by hardware**
  - **Unexpected**
    - **OC7, OC1, OC4 for example**
- **ABEND**
  - **Always “on purpose”**
  - **“User” requested**
  - **“System” requested**

- **Program Exceptions**

  Some of the "not so common" variety

  - **OC2 & OC3**
    - Privileged Operation & EXECUTE exception
    - Not common in COBOL, more likely Assembler
  - **OC5**
    - Addressing Exception
  - **OC6**
    - Specification Exception
- NOTES: there are many more exceptions, but these are the ones typical for application arenas!

Program Exceptions

The “VERY common” variety

- **OC1**
  - Operation Exception
- **OC4**
  - Protection Exception
- **OC7**
  - Data Exception

Program Exceptions

Some “kind of” common ones

- **OC8**
  - Fixed-point overflow
- **OCA**
  - Decimal overflow
- **OC9**
  - Fixed-point Divide Exception
- **OCB**
  - Decimal Divide Exception

- **PSW** – Program Status Word
  - Contains critical information used by the SYSTEM
- **DSA (Stack Frame)**
  - Dynamic Save Area (same as stack frame)
  - Each executing program in the call chain has a stack frame built upon entry to a routine, and “collapsed” upon return from the called module
- **Byte**
  - 8 bits of information
  - Represented by HEX characters (2 chars = 1 byte)
- **Fullword** or **Word**
  - 4 bytes
- **Halfword**
  - 2 bytes
- **Doubleword**
  - 8 bytes

- **DUMP**

Well – lots of definitions for this one ☺

- Webster: “A state of depression, doldrums”
- Winchell: “An opportunity for fun and challenging times”
- In computer terms:
  - Dump contains the contents of internal computer storage areas
  - The areas dumped vary depending upon the options
    - Compiler options
    - Execution or Language Environment Run-Time Options
  - The dump information is the HEX representation of what was occurring at the time of the ABEND or EXCEPTION
- Just one of several TOOLS to help resolve program problems, not the ONLY tool!

- **More terms ...**
  - AMODE – addressing mode
    - What addresses the executable modules “speaks” AND “understands”
    - Effected at LINKEDT time
    - Values possible are 31, 24, ANY
    - Multiple modules statically linked will receive “lowest common denominator”
  - RMODE – Residency mode
    - Where module is “eligible” to live
    - Compiler option RMODE can influence AMODE
    - No AMODE option exists except as PARM override on LINKEDT
    - Values possible: 24, ANY

- Compiler options for debugging and/or problem avoidance
  - **NUMPROC(PFD|NOPFD|MIG)**
  - TRUNC(OPT|BIN|STD)
  - OPTIMIZE(STD|FULL)|NOOPTIMIZE
  - LIST|NOLIST
  - OFFSET|NOOFFSET
  - MAP|NOMAP
  - XREF(SHORT|FULL)|NOXREF
  - VBREF|NOVBREF
  - SSRANGE
  - TEST(NONE,SYM,SEPARATE)

- Compiler options for debugging and/or problem avoidance
  - NUMPROC(PFD|NOPFD|MIG)
  - **TRUNC(OPT|BIN|STD)**
  - OPTIMIZE(STD|FULL)|NOOPTIMIZE
  - LIST|NOLIST
  - OFFSET|NOOFFSET
  - MAP|NOMAP
  - XREF(SHORT|FULL)|NOXREF
  - VBREF|NOVBREF
  - SSRANGE
  - TEST(NONE,SYM,SEPARATE)

- Compiler options for debugging and/or problem avoidance
  - NUMPROC(PFD|NOPFD|MIG)
  - TRUNC(OPT|BIN|STD)
  - **OPTIMIZE(STD|FULL)|NOOPTIMIZE**
  - LIST|NOLIST
  - OFFSET|NOOFFSET
  - MAP|NOMAP
  - XREF(SHORT|FULL)|NOXREF
  - VBREF|NOVBREF
  - SSRANGE
  - TEST(NONE,SYM,SEPARATE)

- Compiler options for debugging and/or problem avoidance
  - NUMPROC(PFD|NOPFD|MIG)
  - TRUNC(OPT|BIN|STD)
  - OPTIMIZE(STD|FULL)|NOOPTIMIZE
  - **LIST|NOLIST**
  - **OFFSET|NOOFFSET**
  - MAP|NOMAP
  - XREF(SHORT|FULL)|NOXREF
  - VBREF|NOVBREF
  - SSRANGE
  - TEST(NONE,SYM,SEPARATE)

- Compiler options for debugging and/or problem avoidance
  - NUMPROC(PFD|NOPFD|MIG)
  - TRUNC(OPT|BIN|STD)
  - OPTIMIZE(STD|FULL)|NOOPTIMIZE
  - LIST|NOLIST
  - OFFSET|NOOFFSET
  - **MAP|NOMAP**
  - **XREF(SHORT|FULL)|NOXREF**
  - **VBREF|NOVBREF**
  - SSRANGE
  - TEST(NONE,SYM,SEPARATE)

- Compiler options for debugging and/or problem avoidance
  - NUMPROC(PFD|NOPFD|MIG)
  - TRUNC(OPT|BIN|STD)
  - OPTIMIZE(STD|FULL)|NOOPTIMIZE
  - LIST|NOLIST
  - OFFSET|NOOFFSET
  - MAP|NOMAP
  - XREF(SHORT|FULL)|NOXREF
  - VBREF|NOVBREF
  - **SSRANGE|NOSSRANGE**
  - **TEST(NONE,SYM,SEPARATE)**

APAR PQ74201 for the TEST(NONE,SYM,SEPARATE) option

A strategy for Debugging

the

Beginning “Begins” HERE!

Problem determination begins with the basics!

1. Has the program executed correctly previously?
2. Was any output produced?
3. ***WHAT HAS CHANGED?***
4. Did the program compile cleanly?
5. Did the program LINK cleanly?
6. Check output messages
7. Did the program produce any output?
8. Who issued the message?
9. If the program abended what is the ABEND CODE?
10. If the program abended due to a program exception, what type of exception?

- **Message Prefix can be useful:**
  - **CEE** – Language Environment
  - **IGY** – COBOL Compiler messages
  - **IGZ** – COBOL run-time messages
  - **EDC** – C/VSE
  - **IBM** – PL/I
  - **DFH** – CICS
  - **DSN** – DB2
  - **ICE** – DFSORT
  - **IDC** – VSAM Access Method Services

- **Language Environment run-time messages**
  - Life has changed!
  - With Language Environment the messages look different
    - Prefixed with CEE indicates a message from LE
    - Prefixed with IGZ is a run-time COBOL message
    - All are documented in the Language Environment Debugging Guide and Run-Time Message
  - Execution options also impact amount of information in the LE produced dump

- With Language Environment & COBOL messages look different
  - New messages at execution
    - CEE3321C Execution Failed with VSE CANCEL CODE 20 and interruption code 07
      - Just means something broke - look further!
  - U4038
    - If TERMTHDACT(DUMP)
  - U4039
    - If TERMTHDACT(UADUMP)
  - CEEnnnn MSG ... maybe
    - Indicates something that Language Environment detected
  - IGZnnnn MSG ... maybe
    - This is a message from COBOL

- **Let's revisit the typical IBM abend codes in COBOL application programs and the "new" LE message code**
  - SOC7 –Data Exception CEE3207S
    - The data is incorrect for the operation being attempted
    - Usually this is storage that is defined as COMP-3 but does not contain valid packed decimal data for some reason
  - SOC4 –Protection Exception CEE3204S
    - Number and/or format of parameters do not match between main and sub program
    - Overwriting your own storage areas (going outside the bounds of a table or going outside the bounds of a data item with reference modification)
    - Attempting to access file I/O without a successful OPEN or following a CLOSE
  - SOC1 – Operation Exception CEE3201S
    - Trying to execute what should be a valid instruction but no instruction is at the location
    - Typically caused by overwriting storage, or a missing subprogram module
  - SOC9 or SOCB – Divide by zero (and no ON OVERFLOW coded) CEE3209S CEE320BS
- Attempting to pass 31-bit address to AMODE(24) program will cause abend as well – but the message is VERY SPECIFIC

- OK – let's take a look at the job output from a processing example where execution resulted in a program check & abend:

```
BG 0000 // JOB COBVSORT
        DATE 08/22/2003, CLOCK 11/11/33
BG 0000 ILU321I J1 COBVSORT SORT  COMPLETE, INSERT 5, DELETE 5, IN 0, OUT 0
BG 0000 COBVSORT - PROGRAM CHECKING NOW!!
BG 0000 ILU321I J1 COBVSORT SORT  COMPLETE, INSERT 5, DELETE 0, IN 0, OUT 0
BG 0000 CEE3321C EXECUTION FAILED WITH VSE CANCEL CODE 20 AND INTERRUPTION
        CODE 07.
BG 0000 0S02I A CANCEL OR CANCEL ALL MACRO WAS ISSUED
BG 0000 0S00I JOB COBVSORT CANCELED
BG 0000 0S07I PROBLEM PROGRAM  PSW = 071D0000 8052123E
END OF LISTLOG UTILITY
1S78I  JOB TERMINATED DUE TO  PROGRAM ABEND
EOJ COBVSORT  MAX.RETURN CODE=0008
```


- Key Sections of a Language Environment Dump
  - Initially the 1st line shows CEE5DMP and the level of LE
    - This is the LE “program” that produces the dump
  - Condition processing indicates this condition was not handled, therefore, “unhandled”
  - ENCLAVE refers to the run-unit currently executing: SMPLSORT
  - Information for thread (the one and only thread)

```
CEE5DMP V1 R4.3: CONDITION PROCESSING RESULTED IN THE UNHANDLED CONDITION.
        08/22/03 11:11:36 AM                    PAGE:    1

INFORMATION FOR ENCLAVE SMPLSORT

 INFORMATION FOR THREAD 8000000000000000
```


- The next key area of the dump is the TRACEBACK area which will be for the current thread (multi-threaded) or only thread
  - **DSA Addr**: Address of the DSA (Dynamic Save Area)
  - **Program Unit**: Enclave Name in REVERSE ORDER OF EXECUTION
  - **PU Addr**: Load Address in memory (the program storage itself is NOT part of the DUMP output)
  - **PU Offset**: Entry point within the Program Unit for the ENCLAVE
  - **Entry**: Entry point name
  - **E Addr**: address of this entry point within the program unit
  - **E Offset**: offset of this entry point within the program unit
  - **Statement**: if you compiled with TEST(NONE,SYM) this is the line number
  - **Service Level** Indicates maintenance level
  - **Status**: shows the reverse sequence of how you got to/through each module

```
INFORMATION FOR ENCLAVE SMPLSORT

  INFORMATION FOR THREAD 8000000000000000

  TRACEBACK:
    DSA ADDR  PROGRAM UNIT  PU ADDR   PU OFFSET  ENTRY       E ADDR    E  OFFSET   STATEMENT  SERVICE  STATUS
    00544478  CEEHDSP       0139F590  +00002B82  CEEHDSP     0139F590  +00002B82              BASELVL  CALL
    01356038  SMPLSORT      00500078  +00000982  SMPLSORT    00500078  +00000982          79           EXCEPTION
    00567960                00000000  +00000000              00000000  +00000000                       CALL
    005441D8                00544358  +00000000              00544358  +00000000                       CALL
    005048E8                00504580  +00000000              00504580  +00000000                       CALL
    00544258                00000000  +00000000              00000000  +00000000                       CALL
    00544018  CEEYSMG       005834A8  +00000418  CEEYSMG     005834A8  +00000418                       CALL
    005675F0  IGZESMG       00593CA8  +00000F7E  IGZESMG     00593CA8  +00000F7E                       CALL
    00567B0C  SMPLSORT      00500078  +0000086C  SMPLSORT    00500078  +0000086C          65           CALL
```


- The Condition Information for Active routines is next:
  - Current Condition – typically an LE message
  - Original Condition – this is the "real deal" – very important
    - Following by the LOCATION of the condition and the
    - Machine State

```
CONDITION INFORMATION FOR ACTIVE ROUTINES
  CONDITION INFORMATION FOR SMPLSORT (DSA ADDRESS 01356038)
    CIB ADDRESS: 00544980
    CURRENT CONDITION:
      CEE0198S THE TERMINATION OF A THREAD WAS SIGNALED DUE TO AN UNHANDLED CONDITION.
    ORIGINAL CONDITION:
      CEE3207S THE SYSTEM DETECTED A DATA EXCEPTION.
    LOCATION:
      PROGRAM UNIT: SMPLSORT ENTRY: SMPLSORT STATEMENT: 79 OFFSET: +00000982
    MACHINE STATE:
      ILC..... 0006    INTERRUPTION CODE..... 0007
      PSW..... 071D3000 80500A00
      GPR0..... 00000000  GPR1..... 0050029C  GPR2..... 0052F7EC  GPR3..... 0052F7EC
      GPR4..... 01356458  GPR5..... 805009A6  GPR6..... 00569088  GPR7..... 00567DB8
      GPR8..... 00569038  GPR9..... 00569138  GPR10.... 00500178  GPR11.... 005005A4
      GPR12.... 0050016C  GPR13.... 01356038  GPR14.... 805009FA  GPR15.... 81360F70
  STORAGE DUMP NEAR CONDITION, BEGINNING AT LOCATION: 005009EA
    +000000 005009EA  B3E45820 D05C58F0 202C4110 A12405EF  FA209078 A0AFF822 90789078
                                        5830D1BC  |.U...*.0.............8.......J.|
```


- The next part of the dump contains the DSA and register information for EVERY ROUTINE on the calling chain sequence
  - The saved registers for CEEHDSP (this is the “dump” routine in LE)
  - Next is SMPLSORT - and you keep going through the phases on the chain in the traceback

```
PARAMETERS, REGISTERS, AND VARIABLES FOR ACTIVE ROUTINES:
  CEEHDSP (DSA ADDRESS 00544478):
    SAVED REGISTERS:
      GPR0..... 00000000  GPR1..... 0054481C  GPR2..... 00000008  GPR3..... 013E8E18
      GPR4..... 00000003  GPR5..... 813A2116  GPR6..... 013A358C  GPR7..... 00545477
      GPR8..... 013A258D  GPR9..... 013A158E  GPR10.... 013A058F  GPR11.... 8139F590
      GPR12.... 0052DF18  GPR13.... 00544478  GPR14.... 80536EEE  GPR15.... 8133EEF8
  SMPLSORT (DSA ADDRESS 01356038):
    SAVED REGISTERS:
      GPR0..... 00000000  GPR1..... 0050029C  GPR2..... 0052F7EC  GPR3..... 0052F7EC
      GPR4..... 01356458  GPR5..... 805009A6  GPR6..... 00569088  GPR7..... 00567DB8
      GPR8..... 00569038  GPR9..... 00569138  GPR10.... 00500178  GPR11.... 005005A4
      GPR12.... 0050016C  GPR13.... 01356038  GPR14.... 805009FA  GPR15.... 81360F70
           (DSA ADDRESS 00567960):
    SAVED REGISTERS:
      GPR0..... 00000000  GPR1..... 00000000  GPR2..... 00000000  GPR3..... 00000000
      GPR4..... 00000000  GPR5..... 005675F0  GPR6..... 00000000  GPR7..... 00000000
      GPR8..... 00000000  GPR9..... 00000000  GPR10.... 00000000  GPR11.... 00000000
      GPR12.... 00000000  GPR13.... 00567960  GPR14.... 8059567E  GPR15.... 00000000
   (DSA ADDRESS 005441D8):
    SAVED REGISTERS:
      GPR0..... 805955F6  GPR1..... 005050A4  GPR2..... 0052DD10  GPR3..... 00506C28
      GPR4..... 0050C270  GPR5..... 005180C4  GPR6..... 805955F6  GPR7..... 00536038
```

- Next come control block areas
  - For COBOL this is DSA and CIB blocks

```
CONTROL BLOCKS FOR ACTIVE ROUTINES:
    DSA FOR CEEHDSP: 00544478
      +000000  FLAGS.... 0808      member... CEE1      BKC...... 01356038  FWC...... 00547018  R14...... 80536EEE
      +000010  R15...... 8133EEF8  R0....... 00000000  R1....... 0054481C  R2....... 00000008  R3....... 013E8E18
      +000024  R4....... 00000003  R5....... 813A2116  R6....... 013A358C  R7....... 00545477  R8....... 013A258D
      +000038  R9....... 013A158E  R10...... 013A058F  R11...... 8139F590  R12...... 0052DF18  reserved. 00000000
      +00004C  NAB...... 00547018  PNAB..... 00544478  reserved. 00000000  00000000  00000000  00000000
      +000064  reserved. 00000000  reserved. 00000000  MODE..... 813A2114  reserved. 00000000  00000000
      +000078  reserved. 00000000  reserved. 00000000
    DSA FOR SMPLSORT: 01356038
      +000000  FLAGS.... 0010      member... 8001      BKC...... 00567960  FWC...... 00568030  R14...... 805009FA
      +000010  R15...... 81360F70  R0....... 00000000  R1....... 0050029C  R2....... 0052F7EC  R3....... 0052F7EC
      +000024  R4....... 01356458  R5....... 805009A6  R6....... 00569088  R7....... 00567DB8  R8....... 00569038
      +000038  R9....... 00569138  R10...... 00500178  R11...... 005005A4  R12...... 0050016C  reserved. F3E3C7E3
      +00004C  NAB...... 00547018  PNAB..... 03000000  reserved. 77420220  00567188  0052F7EC  01356210
      +000064  reserved. 00000003  reserved. 0000007B  MODE..... 80594D20  reserved. 005675F0  00567C80
      +000078  reserved. 00000000  reserved. 00000000
    CIB FOR SMPLSORT: 00544980
      +000000 00544980  C3C9C240 00000000 00000000 010C0004  00000005 00567188 000300C6 59C3C5C5  |CIB ...................h...F.CEE|
      +000020 005449A0  00000000 00544A8C 00030C87 59C3C5C5  00000000 00000004 00567B0C 8137E080  |............g.CEE..........#.a...|
      +000040 005449C0  00000000 01356038 00500A00 013E9160  00000005 00000000 00000000 00000000  |......-..&....j-................|
      +000060 005449E0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
      +000080 00544A00  00000000 00000000 50E0F028 5010F010  4110F02C 4500F014 00000000 17EE05E0  |........&.0.&.0...0...0.........|
      +0000A0 00544A20  12EE4720 E0080A82 0A0258E0 F02807FE  48234000 00000020 00000007 40404040  |.......b....0..... .........    |
      +0000C0 00544A40  40404040 00000000 01356038 01356038  005009FA 5B5BC2D6 D7C5D540 00000067  |    ......-...-..&..$$BOPEN ....|
      +0000E0 00544A60  00567B0C 00000005 00000008 00000014  FFFFFFFC 00000000 00000000 00567E60  |..#.............................=-|
      +000100 00544A80  00000000 013E9244 00500504 E9D4C3C8  00D00001 00000000 0050029C 0052F7EC  |......k..&..ZMCH.........&....7.|
```


- For COBOL, more storage is possible in the dump!
  - Date and time program was compiled
  - Task Global Table (TGT)
    - Explained in the COBOL Programming Guide and the
    - Language Environment Debugging Guide and Run-Time Messages

```
                PROGRAM SMPLSORT WAS COMPILED 08/22/03 11:11:34 AM
COBOL VERSION = 01 RELEASE = 01 MODIFICATION = 01     USER LEVEL = '    '
TGT FOR SMPLSORT: 01356038
  +000000 01356038  00108001 00567960 00568030 805009FA  81360F70 00000000 0050029C 0052F7EC  |.......-.....&..a........&....7.|
  +000020 01356058  0052F7EC 01356458 805009A6 00569088  00567DB8 00569038 00569138 00500178  |..7......&.w...h..'.......j..&..|
  +000040 01356078  005005A4 0050016C F3E3C7E3 00547018  03000000 77420220 00567188 0052F7EC  |.&.u.&.%3TGT..............h..7.|
  +000060 01356098  01356210 00000003 0000007B 80594D20  005675F0 00567C80 00000000 00000000  |...........#..(....0..@.........|
  +000080 013560B8  0052DF18 000001D8 00000000 00000000  00000006 00000001 E2E8E2D6 E4E34040  |.......Q................SYSOUT  |
  +0000A0 013560D8  C9C7E9E2 D9E3C3C4 00000000 00000000  00000000 00000000 00000000 00000000  |IGZSRTCD........................|
  +0000C0 013560F8  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +0000E0 01356118  00000000 013565F8 0050016C 00000001  013561F8 00567408 00500228 01356174  |.......8.&.%....../8.....&..../.|
  +000100 01356138  00500078 00500178 013561B8 00500174  013561EC 00569138 00000000 00000000  |.&...&..../..&..../...j.........|
  +000120 01356158  00569100 00569138 00000000 00569038  00569088 00000000 00567DB8 00000000  |..j...j...........h......'.....|
  +000140 01356178  00500A9C 07FE07FE 00000000 00000000  00001FFF 07FE0000 00000000 00000000  |.&..............................|
  +000160 01356198  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +000180 013561B8  005005EC 0050077E 005008CE 005006FC  005008FC 00500A56 0050077E 005008CE  |.&...&.=.&...&...&...&...&.=.&..|
  +0001A0 013561D8  0050095C 00500A2A 00000000 005005EC  00000001 01356258 01356358 01356458  |.&.*.&.......&..................|
  +0001C0 013561F8  C0000000 00000000 00000000 01356558  00000009 994C0000 01356038 01356258  |....................r<....-.....|
CLLE FOR SMPLSORT: 00567408
  +000000 00567408  E2D4D7D3 E2D6D9E3 00014000 00000000  94810000 00500078 01356038 00500078  |SMPLSORT.. .....ma...&....-..&..|
  +000020 00567428  00000000 00000000 00000000 00000000  00567000 000000E8 000000E0 00000000  |.....................Y........|
```


- One very interesting area is the “Local Variables”
  - Only exists if you compile with TEST(,SYM) to build to symbol tables for your data division names (and now you have SEPARATE as well)

```
LOCAL VARIABLES:
         25 01 SORT-WORK-1-AREA
                                AN-GR
         26  02 SORT-KEY1        X(10) DISP       '0000009998'
         27  02 SORT-KEY2        X(10) DISP       '          '
         28  02 FILLER           X(60) DISP       '
                                                   '
         29 FILE1                                  FILE SPECIFIED AS: ORGANIZATION=VSAM SEQUENTIAL, ACCESS
                                                  MODE=SEQUENTIAL, RECFM=FIXED.  CURRENT STATUS OF FILE IS:
                                                  NOT OPEN, A SUCCESSFUL ACTION SINCE OPEN, LAST
                                                  REQUEST=CLOSED, VSAM STATUS CODE=00, VSAM FEEDBACK=000,
                                                  VSAM RET CODE=000, VSAM FUNCTION CODE=000.
         32 01 RECORD1          X(80) DISP          '0000009994
                                                                '
         33 FILE2                                  FILE SPECIFIED AS: ORGANIZATION=SEQUENTIAL, ACCESS
                                                  MODE=SEQUENTIAL, RECFM=FIXED.  CURRENT STATUS OF FILE IS:
                                                  CLOSED STATUS=FILE, SAM STATUS CODE=00.
         36 01 RECORD2          X(80)               *** Invalid address for this item, no value displayed ***
         37 FILE3                                  FILE SPECIFIED AS: ORGANIZATION=SEQUENTIAL, ACCESS
                                                  MODE=SEQUENTIAL, RECFM=FIXED.  CURRENT STATUS OF FILE IS:
                                                  OPEN STATUS=INPUT, SAM STATUS CODE=10.
         40 01 RECORD3          X(80) DISP          '0000009998
                                                                '
         42 01 F-STATUS1        99 DISP             00
         43 01 F-STATUS2        99 DISP             00
         44 01 F-STATUS3        99 DISP             10
         45 01 KEY-NUMB         9(10) DISP          0000009994
         46 01 EXEC-SORT        X(80) DISP          'OPTION NOSTXIT
                                                                '
         47 01 ABENDIT          S9(5) CMP3          *** Invalid data for this data type *** Hex 000000
```


- Now we can look at the information for SMPLSORT, since we suspect that this program is our “problem child”

```
STORAGE FOR ACTIVE ROUTINES:
 SMPLSORT:
   CONTENTS OF BASE LOCATORS FOR FILES ARE:
        0-00569038        1-00569088        2-00000000        3-00567DB8

   CONTENTS OF BASE LOCATORS FOR WORKING STORAGE ARE:
        0-00569138

   CONTENTS OF BASE LOCATORS FOR THE LINKAGE SECTION ARE:
        1-00000000

   NO VARIABLY LOCATED AREAS WERE USED IN THIS PROGRAM.

   NO EXTERNAL DATA WAS USED IN THIS PROGRAM.

   NO INDEXES WERE USED IN THIS PROGRAM.

FILE RECORD CONTENTS FOR SMPLSORT
TFILE1   (BLF-0): 00569038
  +000000 00569038  F0F0F0F0 F0F0F9F9 F9F84040 40404040  40404040 40404040 40404040 40404040  |0000009998                      |
  +000020 00569058  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                 |
  +000040 00569078  40404040 40404040 40404040 40404040  F0F0F0F0 F0F0F9F9 F9F44040 40404040  |               0000009994        |
TFILE2   (BLF-1): 00569088
  +000000 00569088  F0F0F0F0 F0F0F9F9 F9F44040 40404040  40404040 40404040 40404040 40404040  |0000009994                      |
  +000020 005690A8  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040  |                                 |
  +000040 005690C8  40404040 40404040 40404040 40404040  00000000 00000000 00000000 00000000
```


- Now we can look at the information for SMPLSORT, since we suspect that this program is our “problem child”
  - Statement 79 is the last thing the program did, and, of course the variable ABENDIT is uninitialized, as noted in the dump with the local variables earlier
  - This is a classic case of uninitialized data, which happens because you do not value your storage, or because file data has invalid data - classic SOC7! now a CEE3207S!

```
000077                IN3.
000078                    DISPLAY "COBVSORT - Program Checking NOW!!" upon Console.
000079                    ADD 1 TO ABENDIT.
000080                    CLOSE FILE3.
```


- Here is a different kind of execution abend where COBOL is detecting the error
  - The message IGZ0006S is, of course, from COBOL (you know this because of the IGZ prefix)
  - Notice that you have almost enough information from the message without even looking in the dump!

```
* Execute Program
// OPTION NOSYSDUMP,PARTDUMP
// EXEC COBVTST9,SIZE=COBVTST9,PARM='/CHECK(ON),TER(DUMP)'
1S54I  PHASE COBVTST9 IS TO BE FETCHED FROM GARRYH.BATCH
IGZ0006S THE REFERENCE TO TABLE ALL-FIXED BY VERB NUMBER 01 ON LINE 000095
         ADDRESSED AN AREA OUTSIDE THE REGION OF THE TABLE.
         FROM COMPILE UNIT COBVTST9 AT ENTRY POINT COBVTST9
         AT STATEMENT 95 AT COMPILE UNIT
         OFFSET +0000047C AT ADDRESS 005004F4.
```


- Key Sections of a Language Environment Dump are still the same
  - 1st line shows CEE5DMP and the level of LE
    - This is the LE "program" that produces the dump
  - Condition processing indicates this condition was not handled, therefore, "unhandled"
  - ENCLAVE refers to the run-unit currently executing: COBTST9
  - Information for thread (the one and only thread)

```
CEE5DMP V1 R4.3: CONDITION PROCESSING RESULTED IN THE UNHANDLED CONDITION.
                  08/22/03 11:34:51 AM                  PAGE:    1

INFORMATION FOR ENCLAVE COBVTST9

  INFORMATION FOR THREAD 8000000000000000
```


- The next key area of the dump again is the TRACEBACK area
  - Starts with program COBVTST9
  - Last good thing that happened is at statement 95
  - Notice the EXCEPTION is detected later!

```
TRACEBACK:
    DSA ADDR  PROGRAM UNIT  PU ADDR   PU OFFSET  ENTRY        E ADDR    E  OFFSET   STATEMENT  STATUS
    0052A118  CEEHDSP       013A0420  +00002B78  CEEHDSP      013A0420  +00002B78             CALL
    0052D528  CEEHSGLT      013AA2E0  +0000005C  CEEHSGLT     013AA2E0  +0000005C             EXCEPTION
    0052D018  IGZCMSG       0136DE18  +00000378  IGZCMSG      0136DE18  +00000378             CALL
    01359038  COBVTST9      00500078  +0000047C  COBVTST9      00500078  +0000047C          95 CALL
```


- The Condition Information for Active routines is next:
  - Current Condition – typically an LE message
  - Original Condition – this is the “real deal” – very important
    - Following by the LOCATION of the condition
    - In this case since COBOL detected the error, along with Language Environment, the information is a bit different - no machine state!

```
  CURRENT CONDITION:
       CEE0198S THE TERMINATION OF A THREAD WAS SIGNALED DUE TO AN UNHANDLED CONDITION.
     ORIGINAL CONDITION:
       IGZ0006S THE REFERENCE TO TABLE ALL-FIXED BY VERB NUMBER 01 ON LINE 000095
    ADDRESSED AN AREA OUTSIDE THE REGION OF THE
            TABLE.
LOCATION:
    PROGRAM UNIT: CEEHSGLT ENTRY: CEEHSGLT STATEMENT:  OFFSET: +0000005C
STORAGE DUMP NEAR CONDITION, BEGINNING AT LOCATION: 013AA32C
+000000 013AA32C  F010D20B D0801000 58A0C2B8 58F0A01C  05EFD20B D098B120 41A0D098
    50A0D08C  |0.K.......B..0....K..q.....q&...|
```


- This TRACEBACK area looks a bit different because the “problem” was detected by COBOL, not by LE, but the “parts” are still
  - **DSA Addr**: Address of the DSA (Dynamic Save Area)
  - **Program Unit**: Enclave Name in REVERSE ORDER OF EXECUTION
  - **PU Addr**: Load Address in memory (the program storage itself is NOT part of a CEEDUMP)
  - **PU Offset**: Entry point within the Program Unit for the ENCLAVE
  - **Entry**: Entry point name
  - **E Addr**: address of this entry point within the program unit
  - **E Offset**: offset of this entry point within the program unit
  - **Statement**: if you compiled with TEST(NONE,SYM) this is the line number of the last thing that happened (good or bad)
  - **LoadModule**: Name of the module containing this PU
  - **Service Level** (APAR/PTF level of service)
  - **Status**: shows the reverse sequence of how you got to/through each module

```
TRACEBACK:
   DSA ADDR  PROGRAM UNIT  PU ADDR   PU OFFSET  ENTRY        E ADDR    E  OFFSET   STATEMENT  STATUS
   0052A118  CEEHDSP       013A0420  +00002B78  CEEHDSP      013A0420  +00002B78              CALL
   0052D528  CEEHSGLT      013AA2E0  +0000005C  CEEHSGLT     013AA2E0  +0000005C              EXCEPTION
   0052D018  IGZCMSG       0136DE18  +00000378  IGZCMSG      0136DE18  +00000378              CALL
   01359038  COBVTST9      00500078  +0000047C  COBVTST9     00500078  +0000047C         95   CALL
```


- The next part of the dump contains the DSA and register information for EVERY ROUTINE on the calling chain sequence
  - The saved registers for CEEHDSP (this is the “dump” routine in LE)
  - Next is CEEHSGLT, IGZCMSG, and then COBVTST9 - and you keep going through the phases on the chain in the traceback

```
PARAMETERS, REGISTERS, AND VARIABLES FOR ACTIVE ROUTINES:
  CEEHDSP (DSA ADDRESS 0052A118):
    SAVED REGISTERS:
      GPR0..... 00000000  GPR1..... 0052A4BC  GPR2..... 00000008  GPR3..... 013E8E18
      GPR4..... 00000003  GPR5..... 813A2F9C  GPR6..... 013A441C  GPR7..... 0052B117
      GPR8..... 013A341D  GPR9..... 013A241E  GPR10.... 013A141F  GPR11.... 813A0420
      GPR12.... 00513F18  GPR13.... 0052A118  GPR14.... 8051CEEE  GPR15.... 8132C1B0
  CEEHSGLT (DSA ADDRESS 0052D528):
    SAVED REGISTERS:
      GPR0..... 0052D528  GPR1..... 0052D1E8  GPR2..... 0052D1E8  GPR3..... 00000000
      GPR4..... 0051C038  GPR5..... 0051C038  GPR6..... 00500252  GPR7..... 00000005
      GPR8..... 00515A70  GPR9..... 01359038  GPR10.... 0051C038  GPR11.... 813AA2E0
      GPR12.... 00513F18  GPR13.... 0052D528  GPR14.... 8051CEDA  GPR15.... 80508F48
  IGZCMSG  (DSA ADDRESS 0052D018):
    SAVED REGISTERS:
      GPR0..... 0052D528  GPR1..... 0052D1E8  GPR2..... 0052D1E8  GPR3..... 00000000
      GPR4..... 0051C038  GPR5..... 0051C038  GPR6..... 00500252  GPR7..... 00000005
      GPR8..... 00515A70  GPR9..... 01359038  GPR10.... 0054D188  GPR11.... 8136DE18
      GPR12.... 00513F18  GPR13.... 0052D018  GPR14.... 8052044A  GPR15.... 813AA2E0
  COBVTST9 (DSA ADDRESS 01359038):
    SAVED REGISTERS:
      GPR0..... 0000000F  GPR1..... 00500252  GPR2..... 00000019  GPR3..... 005157EC
      GPR4..... 005000B0  GPR5..... 0054D188  GPR6..... 005151AC  GPR7..... 00000000
      GPR8..... 00515A70  GPR9..... 01351070  GPR10.... 0050017C  GPR11.... 005002FC
      GPR12.... 0050016C  GPR13.... 01359038  GPR14.... 805004F6  GPR15.... 8136DE18
```

- "Local Variables" again has interesting information
  - Remember, this only exists if you compile with TEST(,SYM) to build to symbol tables for your data division names

```
LOCAL VARIABLES:
        62 01 FIVE-13-BYTES-LONG
                              X(513) DISP      '
                                                '
        63 01 ALL-SUBSCRIPT-FIXED
                              AN-GR
        64  02 ALL-FIXED       X(5)  OCCURS 5
             SUB(1)           DISP             '1    '
             SUB(2)                            '2    '
             SUB(3)                            '3    '
             SUB(4)                            '4    '
             SUB(5)                            '5    '
        65 01 NUMERIC-1       99 DISP          06
        66 01 SUBSCRIPT       S9 DISP          +6
```


- STORAGE is the HEX information
  - You need the correct TERMTHDACT run-time option STORAGE!

```
STORAGE FOR ACTIVE ROUTINES:
    COBVTST9:
      NO FILES WERE USED IN THIS PROGRAM.

      CONTENTS OF BASE LOCATORS FOR WORKING STORAGE ARE:
           0-01351070

      CONTENTS OF BASE LOCATORS FOR THE LINKAGE SECTION ARE:
           1-00000000

      NO VARIABLY LOCATED AREAS WERE USED IN THIS PROGRAM.

      NO EXTERNAL DATA WAS USED IN THIS PROGRAM.

      NO INDEXES WERE USED IN THIS PROGRAM.

    WORKING STORAGE FOR COBVTST9
    BLW-0: 01351070
     +000000 01351070  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
     +000020 01351090 - +0001FF 0135126F             SAME AS ABOVE
     +000200 01351270  00000000 00000000 F1404040 40F24040  4040F340 404040F4 40404040 F5404040  |........1    2    3    4    5   |
     +000220 01351290  40000000 00000000 F0F60000 00000000  C6000000 00000000 00000000 00000000  | .......06......F...............|
```


- Here is the compiler listing WITH TEST and MAP options

```
INVOCATION PARAMETERS:
ADATA
PROCESS(CBL) STATEMENTS:
 CBL APOST,LIB,RENT,DATA(31),NOSEQ,TEST,SSRANGE,MAP,OFFSET
OPTIONS IN EFFECT:                                      SSRANGE
      ADATA                                            NOTERM
      ADV                                                   TEST(ALL,SYM,NOSEPARATE) APOST
    NOAWO                                                   TRUNC(STD)
      BUFSIZE(4096)                                    NOVBREF
    NOCMPR2                                                 NOWORD
    NOCOMPILE(S)                                            XREF(SHORT)
    NOCURRENCY                                              YEARWINDOW(1920)
      DATA(31)                                              ZWB
    NODATEPROC
    NODBCS
    NODECK
    NODUMP
    NODYNAM
    NOEXIT
    NOFASTSRT
      FLAG(I)
    NOFLAGMIG
    NOFLAGSAA
    NOFLAGSTD
      INTDATE(ANSI)
      LANGUAGE(UE)
      LIB
      LINECOUNT(60)
    NOLIST
      MAP
    NONAME
    NONUMBER
      NUMPROC(NOPFD)
      OBJECT
      OFFSET
    NOOPTIMIZE
      OUTDD(SYSOUT)
      RENT
      RMODE(AUTO)
    NOSEQUENCE
      SIZE(MAX)
      SOURCE
      SPACE(1)
```


- The compiler listing has important information
  - Allows us to coordinate source with dump information
  - BLW-0 and the offset can be used with the Storage section of the dump

```
 000056         005700*****************************************************************
PP 5686-068 IBM COBOL FOR VSE/ESA 1.1.1                        COBVTST9  DATE 09/08/2003  TIME 13:04:56   PAGE     4
  LINEID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-|--+----8  MAP AND CROSS REFERENCE
  000057         005800*
  000058         005900 environment division.
  000059         006000 data division.
  000060         006100
  000061         006200 working-storage section.
  000062         006300  01  five-13-bytes-long        pic x(513).                              BLW=0000+000         513C
  000063         006400  01  All-subscript-fixed.                                              BLW=0000+208         0CL25
  000064         006500      05 all-fixed              pic x(5) occurs 5 times.                 BLW=0000+208,0000000 5C
  000065         006600  01  numeric-1                 pic 99.                                  BLW=0000+228         2C
  000066         006600  01  subscript                 pic s9.                                  BLW=0000+230         1C
  000067         006700*


  WORKING STORAGE FOR COBVTST9
   BLW-0: 01351070
     +000000 01351070  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
     +000020 01351090 - +0001FF 0135126F             SAME AS ABOVE
     +000200 01351270  00000000 00000000 F1404040 40F24040  4040F340 404040F4 40404040 F5404040  |........1    2    3    4    5   |
     +000220 01351290  40000000 00000000 F0F60000 00000000  C6000000 00000000 00000000 00000000  | .......06......F...............|
```


- From the original dump information we know that statement 95 is the actual offending statement (from the message IGZ0006S indicating that statement 95 addressed an area outside the region of the table

```
  000068         006800 procedure division.
  000069         006900 a00-main.
  000070                    Display 'COBVTST9 Begins.' Upon Console.
  000071                    Move zeros to numeric-1.                                    IMP 65
  000072                    Display 'COBVTST9 - Program Abends with IGZ0006S.' Upon
  000073                            Console.
  000074                    Perform A01-Load-Array 20 times.                            92
  000075         007300*****************************************************************
  000076         007400*  Fixed-length table:                                         *
  000077         007500*   data-name                                                  *
  000078         007600*  11304127  Subscript > length of table                       *
  000079         007700*****************************************************************
  000080         007800     move 7  to numeric-1.                                     65
  000081         007900*                                       VVVVVVVVV
  000082         008000     move function lower-case(all-fixed(numeric-1))            IFN 64 65
  000083         008100          to five-13-bytes-long.                               62
  000084         008200
  000085         007700*****************************************************************
  000086         007500*                                                              *
  000087         007400*   The following Display command should never be executed.    *
  000088         007500*                                                              *
  000089         007700*****************************************************************
  000090                    Display 'COBVTST9 Complete.' Upon Console.
  000091                    Goback.
  000092               A01-Load-Array.
  000093                    compute numeric-1 = numeric-1 + 1.                        65 65
  000094                    Move numeric-1 to subscript.                              65 66
  000095                    Move subscript to all-fixed(subscript).                   66 64 66
  000096               End-A01-Load-Array.
  000097               End program COBVTST9.                                          3
*/ COBVTST9
```


Language Environment

Run-Time options

for

Debugging

- Some of the specific Language Environment run-time options for debugging
  - **ABPERC - IBM-Supplied Default: ABPERC=(NONE)**
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - **ABTERMENC(ABEND)**
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - **ALL31(OFF)**
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - **STACK(128K,128K,BELOW,KEEP)**
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - **CHECK(ON|OFF)**
  - **DEBUG(OFF|ON)**
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - **DEPTHCONDLIMIT=(10)**
  - **ERRCOUNT(20)**
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - **HEAPCHK=(OFF,1,0)**
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - **STACK(128K,128K,BELOW,KEEP)**
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - **STORAGE=(00,NONE,NONE,32K)**
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - **TERMTHDACT(TRACE,,0)**
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - **TEST|NOTEST + suboptions**
  - **TRACE(OFF,4K,DUMP,LE=0)**
  - TRAP(ON|OFF)
  - USRHDLR(userhdlr)

- Some of the specific Language Environment run-time options for debugging
  - ABPERC - IBM-Supplied Default: ABPERC=(NONE)
  - ABTERMENC(ABEND)
  - ALL31(ON|OFF)
  - CHECK(ON|OFF)
  - DEBUG(OFF|ON)
  - DEPTHCONDLIMIT=(10)
  - ERRCOUNT(20)
  - HEAPCHK=(OFF,1,0)
  - STACK(128K,128K,BELOW,KEEP)
  - STORAGE=(00,NONE,NONE,32K)
  - TERMTHDACT(TRACE,,0)
  - TEST|NOTEST + suboptions
  - TRACE(OFF,4K,DUMP,LE=0)
  - **TRAP(ON|OFF)**
  - **USRHDLR(userhdlr)**

- USE FOR DEBUGGING (replace READYTRACE)
  - USE FOR DEBUGGING Declarative module

```
  Environment Division.
  Configuration Section.
  Source-Computer.  IBM-370 WITH DEBUGGING MODE.
 *Source-Computer.  IBM-370.
 ......................
  Data Division.
  Working-Storage Section.
  01 Trace-Switch                 PIC X VALUE 'N'.
        88 TRACE-OFF               VALUE 'N'.
        88 TRACE-ON                VALUE 'Y'.
  Procedure Division.
  Declaratives.
  COBOL-DEBUG SECTION.
        USE FOR DEBUGGING ON ALL PROCEDURES.
  COBOL-DEBUG-PARAGRAPH.
        IF TRACE-ON
              DISPLAY DEBUG-NAME, DEBUG-CONTENTS
        ...any other COBOL statements you want...
        END-IF.
  END DECLARATIVES.
  MAIN-PROCESSING SECTION.
 ....
        SET TRACE-ON TO TRUE
 ....
        SET TRACE-OFF TO TRUE
```

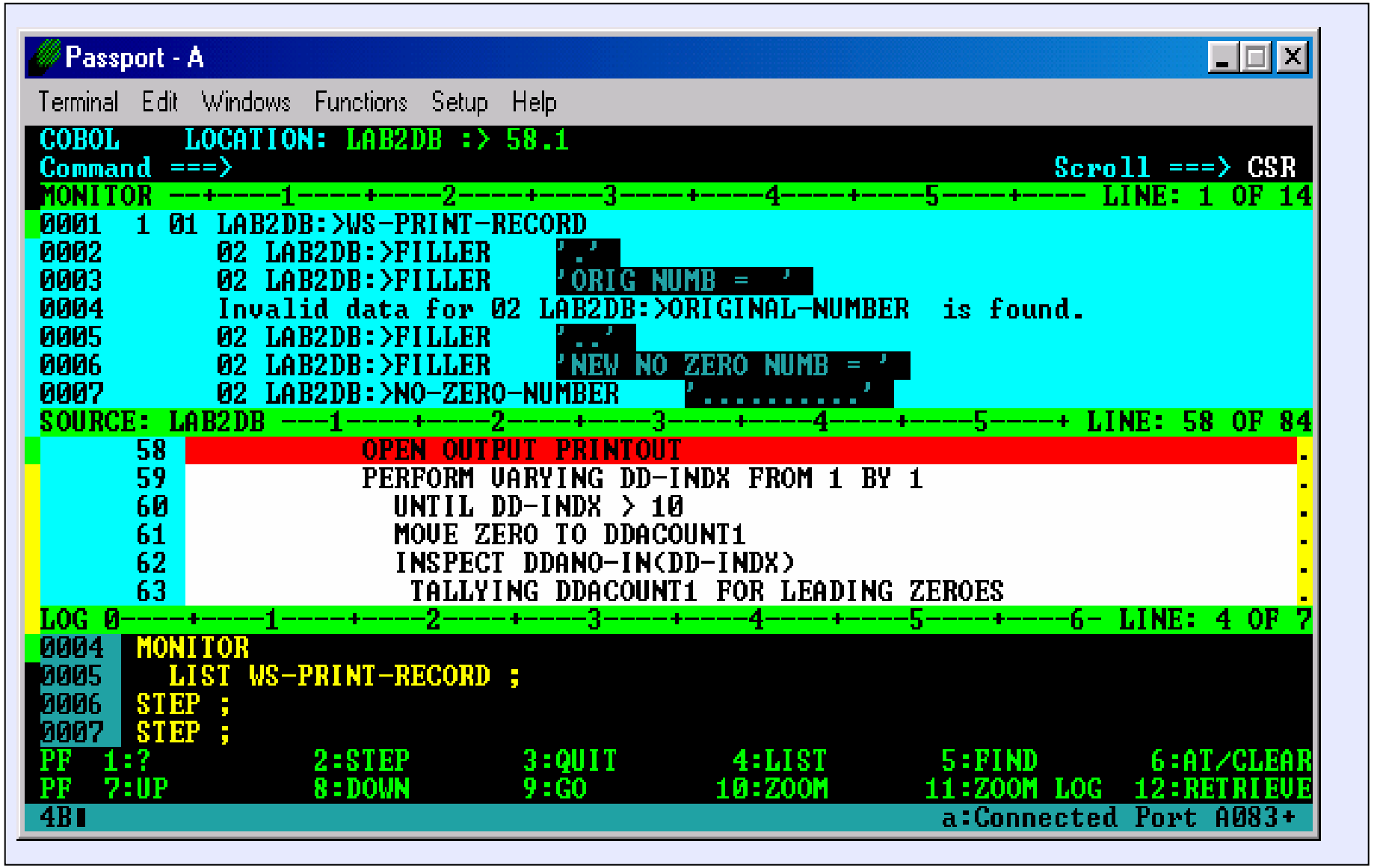

- USE FOR DEBUGGING (replace READYTRACE)
  - Debugging lines
    - Identified with D in column 7
    - Included when compiled with the “WITH DEBUGGING MODE”

```
 Environment Division.
 Configuration Section.
 Source-Computer.  IBM-370 WITH DEBUGGING MODE.
*Source-Computer.  IBM-370.
.....

 PROCEDURE DIVISION.
 100-INITIALIZATION.
D     DISPLAY ‘100-INITIALIZATION’.
.....
 200-BEGIN-PROGRAM.
D     DISPLAY ‘200-BEGIN-PROGRAM’.
D     DISPLAY ’VALUE OF ACCOUNT-NO = ‘, ACCOUNT-NO


.....
```


- Sample of DEBUG TOOL terminal screen




```
     LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-]--+----8  Map/XREF
     000025                PROCEDURE DIVISION USING PARM-DATA.                                    19
     000027                    DISPLAY 'TEST PROGRAM EXTERNAL DATA'
     000030                    IF PARM-LTH = 3                                                    20
     000031
     000032      1                ADD IN-ORDER-QTY TO COB-TOTAL-QUANTITY                         21 12
     000033      1                   ON SIZE ERROR
     000034      2                     DISPLAY 'COB-TOTAL-QUANTITY TOO LARGE'
     000035      2                     DISPLAY 'IN-ORDER-QTY = ', IN-ORDER-QTY                    21
     000036      1                END-ADD
     000037
     000038      1                MULTIPLY IN-ORDER-QTY BY 10 GIVING TEST-QTY                    21 17
     000039      1                DISPLAY 'TEST-QTY = ', TEST-QTY                                17
     000040
     000041      1             ADD IN-ORDER-QTY TO COB-TOTAL-QUANTITY                            21 12
     000042      1               ON SIZE ERROR
     000043      2                 DISPLAY 'COB-TOTAL-QUANTITY TOO LARGE '
     000044      2                 DISPLAY 'IN-ORDER-QTY = ', IN-ORDER-QTY                        21
     000045      2                 ADD 1 TO COB-ERROR-COUNT                                       16
     000046      2             END-ADD
     000047
     000048      2             PERFORM 100-MULTIPLY                                              53
     000049      2             PERFORM 200-CHECK                                                 55
     000050
     000051                    END-IF
   …..
   An "M" preceding a data-name reference indicates that the data-name is modified by this reference.
     Defined   Cross-reference of data names   References
          16   COB-ERROR-COUNT. . . . . . . .  M45
          12   COB-TOTAL-QUANTITY . . . . . .  M32 M41 M54 56
          21   IN-ORDER-QTY . . . . . . . . .  32 35 38 41 44 54
           9   MY-EXTERNAL. . . . . . . . . .  M28
          19   PARM-DATA. . . . . . . . . . .  25
          20   PARM-LTH . . . . . . . . . . .  30
          17   TEST-QTY . . . . . . . . . . .  M38 39
```

```
    LineID  PL SL  ----+-*A-1-B--+----2----+----3----+----4----+----5----+----6----+----7-]--+----8  Map/XREF
    000025               PROCEDURE DIVISION USING PARM-DATA.                                    19
    000027                   DISPLAY 'TEST PROGRAM EXTERNAL DATA'
    000030                   IF PARM-LTH = 3                                                    20
    000031
    000032      1               ADD IN-ORDER-QTY TO COB-TOTAL-QUANTITY                          21 12
    000033      1                  ON SIZE ERROR
    000034      2                    DISPLAY 'COB-TOTAL-QUANTITY TOO LARGE'
    000035      2                    DISPLAY 'IN-ORDER-QTY = ', IN-ORDER-QTY                   21
    000036      1               END-ADD
    000037
    000038      1               MULTIPLY IN-ORDER-QTY BY 10 GIVING TEST-QTY                     21 17
    000039      1               DISPLAY 'TEST-QTY = ', TEST-QTY                                 17
    000040
    000041      1             ADD IN-ORDER-QTY TO COB-TOTAL-QUANTITY                            21 12
    000042      1               ON SIZE ERROR
    000043      2                 DISPLAY 'COB-TOTAL-QUANTITY TOO LARGE '
    000044      2                 DISPLAY 'IN-ORDER-QTY = ', IN-ORDER-QTY                       21
    000045      2                 ADD 1 TO COB-ERROR-COUNT                                      16
    000046      2             END-ADD
    000047
    000048      2             PERFORM 100-MULTIPLY                                              53
    000049      2             PERFORM 200-CHECK                                                 55
    000050
    000051                   END-IF
…..
An "M" preceding a data-name reference indicates that the data-name is modified by this reference.
  Defined   Cross-reference of data names   References
       16   COB-ERROR-COUNT. . . . . .  M45
       12   COB-TOTAL-QUANTITY . . . .  M32 M41 M54 56
       21   IN-ORDER-QTY . . . . . .  32 35 38 41 44 54
        9   MY-EXTERNAL. . . . . .  M28
       19   PARM-DATA. . . . . .  25
       20   PARM-LTH . . . . . .  30
       17   TEST-QTY . . . . . .  M38 39
```

- **Compiler listing output**
  - **MAP - identifies data areas**
    - **BLX- External Data item**
    - **BLF - File fields**
    - **BLW - Working Storage fields**
    - **BLL -Linkage**

    Imbedded in listing at also at bottom
    - **At bottom you get 2 “displacements”**
      - **One within the “structure” of the BLF, BLW, BLL, etc**
      - **2nd is a “structure” displacement relative to each 01 group definition**
    - **Same information is in the STORAGE portion of the dump**
    - **Grouping is in 4K “chunks”**

CEE5DMP callable routine

- **Pass in TITLE**
  - **Appears at top of each page in dump output**
- **OPTIONS indicate what "pieces you want**
- **Can CALL from multiple places in program**
  - **Change the title to indicate "which" dump**

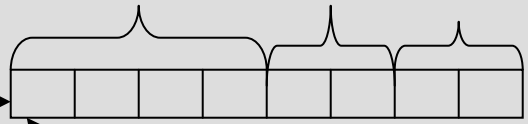
```
01 MY-TITLE          PIC X(80) .
01 MY-OPTIONS        PIC X(255) VALUE 'FILES VARS STOR TRACE.
........
     MOVE 'DUMP ONE' TO MY-TITLE
     CALL 'CEE5DMP' USING MY-TITLE, MY-OPTIONS, FC
.......
     MOVE 'DUMP TWO' TO MY-TITLE
    CALL 'CEE5DMP USING MY-TITLE, MY-OPTIONS, FC
```


- EXTERNAL is a new choice for “sharing”
  - Available for 01 or 77 level
  - Data-name on 01 or 77 MUST BE THE SAME
  - Cannot code a VALUE on EXTERNAL items
  - Data lengths MUST be the same (this is checked)

In *CALLING program*

```
WORKING-STORAGE SECTION.
01  MY-DATE              EXTERNAL.
    05  MY-YYYY          PIC 9(4).
    05  MY-MM            PIC 9(2).
    05  MY-DD            PIC 9(2).
.........

        CALL 'MYDATE'
```



In *CALLED PROGRAM*

```
WORKING-STORAGE SECTION
01  MY-DATE              EXTERNAL.
    05  WS-YYYY          PIC 9(4).
    05  WS-MM            PIC 9(2).
    05  WS-DD            PIC 9(2).
....
PROCEDURE DIVISION
```


Language Environment

and

Condition Handling

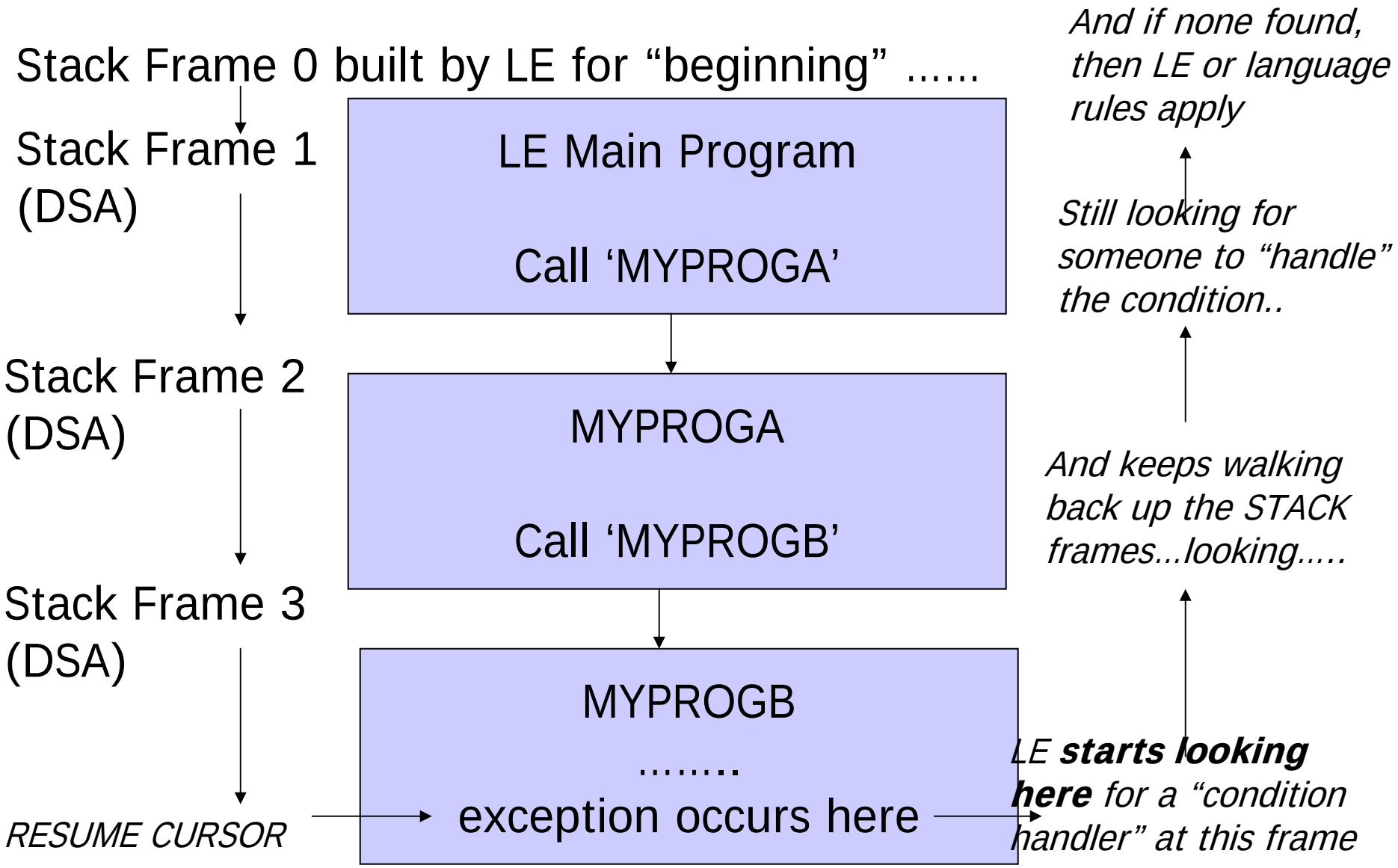
- Condition Handling Services
  - **CEEDCOD** – Decompose Condition Token
  - **CEEGZDT** – Retrieve Q_Data_Token
  - **CEEHDLR** – Register User Condition Handler
  - **CEEHDLU** – Unregister User Condition Handler
  - **CEEITOK** – Return Initial Condition Token
  - **CEEMRCR** – Move Resume Cursor
  - **CEEMRCE** – Move Resume Cursor Explicit
  - **CEESRP** – Set Return Point
  - **CEENCOD** – Construct a Condition Toke
  - **CEE5ABD** – Terminate Enclave with user abend code
  - **CEE5CNC** – Control Nested Conditions
  - **CEE5GRC** – Get Enclave Return Code
  - **CEE5GRN** – Get name of routine incurring condition
  - **CEE5SRC** – Set Enclave Return Code

- **Intrinsic Functions – Callable Services**
  - Similar capabilities
  - LE has more extensive routines and continues to add
  - Intrinsic functions are growing as well
  - With LE, ability to check feedback code
    - For dates – FLEXIBLE FORMAT
    - Parameter format MUST be correct (not just numeric)
    - If something is “wrong” you can catch it programmatically
  - With Intrinsic Functions
    - For dates – FIXED FORMAT
    - Argument requirements less restrictive than LE
    - If something is “wrong” program abends
  - Little performance difference
    - Intrinsic Function often calls LE routine “under the covers”

- **With Language Environment routines**
  - Condition handling routines allow
    - "Trapping" program errors
    - Issuing information
    - Producing same dump information
    - Keeping application running – or not, you choose
    - Works with any LE supported language environment
    - You choose where to start "handling"
      - Can be via RUN-TIME OPTION (USRHDLR)
      - Can be program point (CALL to CEEHDLR)

- **Additional LE concepts**
  - **Stack storage**
    - Save area
    - Local-Storage variables
    - Intrinsic functions
    - Library routines
    - Dynamic variables (PL/I)
  - **Heap storage**
    - COBOL working-storage
    - Variables allocated with malloc(), calloc(), and realloc() in C/VSE
  - **LE run-time option RPTSTG(ON)**
    - Indicates whether you have a problem with either STACK or HEAP

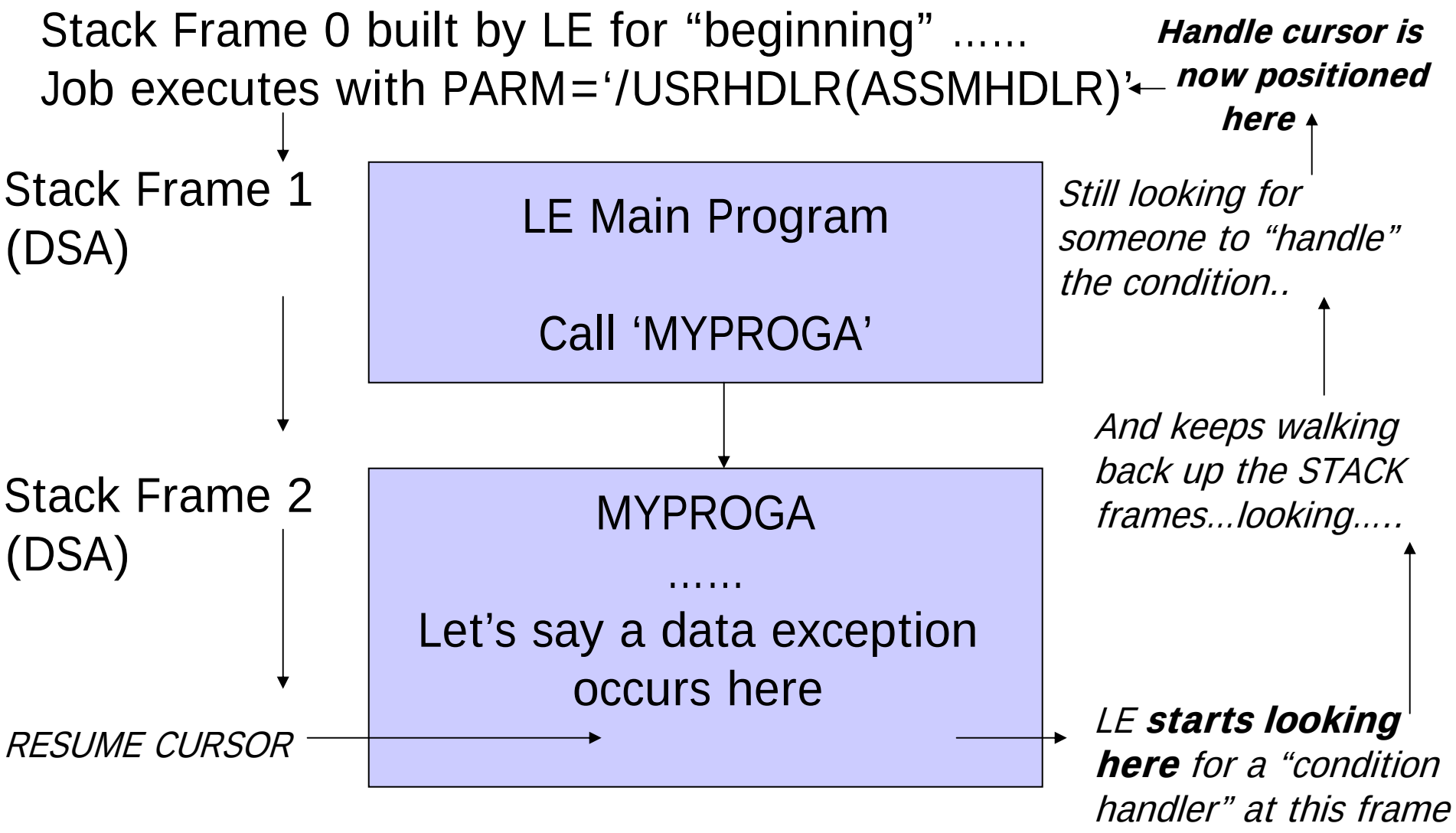
- **OK, so now you get a "condition"....what does LE do?**
  - Starts with the most recently activated stack frame
  - Does not matter how we got the condition
  - LE looks at the most recently activated stack frame
    - Looks for user-written handler for this stack frame
    - Next looks for HLL specific handlers (C/C++ or PL/I)
  - LE "traverses" back through all the active stack frames looking for handlers to process the condition
  - LE continues until there are no more frames
  - If no handlers of any kind are found, then normal LE and/or language rules take over to finish



- **Resume cursor points to the "where to resume" location**
  - This cursor is always *on the move* as the programs execute, tracking the NSI (next sequential instruction)
  - Positioned after the instruction that causes the condition or signal
    - Next *machine instruction*, not necessarily the next "source" instruction
  - May be "moved" with CEEMRCR LE callable routine to move relative to "here"
  - May be "moved" with CEEMRCE callable routine to move to an "explicit" location

- **The other important element in “conditions” is the “handle cursor”**
  - If you CALL CEEHDLR from your routine you have a handle cursor “registered” at this frame (where you issue this CEEHDLR call)
  - If you use the run-time option USRHDLR to identify a handler routine, you have a handle cursor “registered” at Stack Frame 0
  - This is the other piece of the LE CONDITION HANDLING “puzzle”
  - This is how LE knows whether to hand off the condition, and who to hand off to!

- **If all stack frames have been traversed and NO ONE HANDLED THE CONDITION (you did have the chance and the choice)**
  - Language Environment proceeds with termination TIU
  - Return & Reason codes are set based upon “original condition”
  - Message is built and issued (from token)
  - Traceback and dump created (depending on setting of TERMTHDACT run-time option)
  - Thread is terminated (single thread appl means ENCLAVE terminates as well)
- This is why the abend/dump message indicates “thread terminated due to unhandled condition” – *you did have the chance!*



- **CEEHDLR is callable from any point in a user program**
  - You can register multiple condition handlers at multiple "locations"
    - Handlers are processed in a LIFO manner
    - If you have registered more than 1 handler at a specific stack frame level
      - The last shall be first!
  - You can have specific handlers for specific conditions
  - You can have the handler registered at specific "frame" levels
  - You can still use CEEHDLR to be a "top gun"
    - Register your routine RIGHT AWAY in the FIRST PROGRAM
    - If you have no other handlers, this is similar to using the parm, but you get some additional capabilities

Here is an example of using CEEHDLR with COBOL

```
ID DIVISION.
PROGRAM-ID.  LAB1XX.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FC.
        10  FC-SEV                                  PIC S9(4) COMP.
        10  FC-MSG                                  PIC S9(4) COMP.
        10  FC-CTW                                  PIC X.
        10  FC-FAC                                  PIC XXX.
        10  I-S-INF                                 PIC S9(4).
01 MY-DATA                                          PIC s9((9) COMP.
01 CURRENT-RECORD                                   EXTERNAL.
        10  RECORD-IN-PROCESS                       PIC X(80).
01 ERROR-INDICATOR                                  PIC X EXTERNAL.
01 PGMPTR                                           PROCEDURE-POINTER.
.....
PROCEDURE DIVISION.
        SET PGMPTR TO ENTRY 'LAB1HDLR'
        MOVE  9999  TO MY-DATA
        CALL 'CEEHDLR' USING PGMPTR MY-DATA FC
        IF FC-SEV = ZEROES
                DISPLAY 'LAB1HDLR REGISTERED'
        ELSE
                DISPLAY 'LAB1HDLR REGISTRATION FAILED"
                DISPLAY 'FC = ' ,  FC-SEV,  FC-MSG
        END-IF.
100-INITIAL-PROGRAM.
* after reading record or initial processing
* MOVE DATA TO THE EXTERNAL STRUCTURE.....
* This should be data you wish to have available in your User Condition Handler routine
* MY-DATA is 4-byte structure available to pass INTO condition handler
200-SUB-CALL.
        .....
        IF ERROR-INDICATOR = 'Y'
                MOVE +8 to RETURN-CODE
        ELSE
                MOVE +0 to RETURN-CODE
        END-IF
* If there was an error, indicate with non-zero RETURN-CODE
        GOBACK.
```


Here is an example of the LAB1HDLR code (1 of 2)

```
ID DIVISION.
PROGRAM-ID.  LAB1HDLR.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 CURRENT-RECORD                    EXTERNAL.
      10  RECORD-IN-PROCESS                 PIC X(80).
01 ERROR-INDICATOR                          PIC X EXTERNAL.
01  ROUTINE-NAME                    PIC X(80).
01  MSG-DEST                        PIC S9(9) COMP VALUE 2.
01  MSG-STR.
      10  MSG-LEN                   PIC S9(4) COMP VALUE +121.
      10  MSG-STRING                PIC X(121).
01  FC.
      10  FC-SEV                    PIC S9(4) COMP.
      10  FC-MSG                    PIC S9(4) COMP.
      10  FC-CTW                    PIC X.
      10  FC-FAC                    PIC XXX.
      10  I-S-INF                   PIC S9(4).
LINKAGE SECTION.
01  CURRENT-CONDITION.
      10  FBCODE                    PIC X(8).
          88 DATA-EXCEPTION         VALUE X'00030C8759C3C5C5'.
          88 DIVIDE-BY-ZERO         VALUE X'00030C8959C3C5C5'.
      10  FILLER                    PIC X(4).
01  DATA-INFO                       PIC S9(9) COMP.
01  RESULT-CODE                     PIC S9(9) COMP.
      88  RESUME                    VALUE +10.
      88  PERCOLATE                 VALUE +20.
01 NEW-CONDITION                    PIC X(12).
```


Here is an example of the LAB1HDLR code (2 of 2)

```
PROCEDURE DIVISION USING CURRENT-CONDITION,
          DATA-INFO, RESULT-CODE,  NEW-CONDITION.
100-INITIAL.
        IF DATA-EXCEPTION
              CALL 'CEE5GRN' USING ROUTINE-NAME, FC
                 STRING 'The routine ' delimited by size
                              routine-name delimited by ' '
                              'had DATA EXCEPTION.' delimited by size
                              INTO MSG-STRING
                 END-STRING
              CALL 'CEEMOUT' USING MSG-STR , MSG-DEST,  FC
                    STRING 'THE FAILING RECORD is  ' delimited by size
                                RECORD-IN-PROCESS delimited by size
                              INTO MSG-STRING
                    END-STRING
              CALL 'CEEMOUT' USING MSG-STR, MSG-DEST, FC
          SET RESUME TO TRUE
          MOVE 'Y' TO ERROR-INDICATOR
        ELSE
          SET PERCOLATE TO TRUE
          MOVE 'N' TO ERROR-INDICATOR
        END-IF
999- EXIT-PROGRAM.
        GOBACK.
END PROGRAM LAB1HDLR.
```


- When a “condition” occurs
  - The handlers are checking a data field to see what LE is “reporting”
  - This is the condition token that is created when something untoward occurs
  - This is also what LE uses to construct the message you will get regarding the condition
  - The format of the token is consistent:
    - 1st half-word – severity
    - 2nd half-word – message (in binary)
    - 3rd one-byte field – hex codes
    - 4th 3 bytes (char) – who issued the message
      - CEE – LE
      - IGZ – COBOL
      - EDC – C/C++
      - IBM – PL/1

- The handler routines (both COBOL & Assembler) shown here have coded the feedback code as data fields in the program
  - Error messages have this “symbolic feedback code” which represents the 1st 8 positions of the token of the 12-byte field

CEE3207S The system detected a data exception (System Completion Code=0C7).

Explanation: Your program attempted to use a decimal instruction incorrectly. See a Principles of Operation manual for a full list of data exceptions.

Programmer Response: Check the variables associated with the failing statement to make sure that they have been initialized correctly.

System Action: The thread is terminated.

Symbolic Feedback Code: CEE347

- Rather than coding the feedback code you could copy in the appropriate token
  - CEEBALCT for the Assembler “format”
  - CEEIGZCT for COBOL 88’s for LE messages
  - IGZIGZCT for COBOL 88’s for COBOL messages
  - The first 3 positions represent the message origination (LE would be CEE, COBOL would be IGZ)
  - The next 3 positions represent the language format for the conditions
    - IGZ would be the 88’s for COBOL for EVERY symbolic code
    - BAL would be the assembler definitions
  - The CT is for CONDITION TOKEN


```
.............
 88  CEE341      VALUE X'00030C81 59C3C5C5'.
 88  CEE342      VALUE X'00030C82 59C3C5C5'.
 88  CEE343      VALUE X'00030C83 59C3C5C5'.
 88  CEE344      VALUE X'00030C84 59C3C5C5'.
 88  CEE345      VALUE X'00030C85 59C3C5C5'.
 88  CEE346      VALUE X'00030C86 59C3C5C5'.
 88  CEE347      VALUE X'00030C87 59C3C5C5'.
 88  CEE348      VALUE X'00030C88 59C3C5C5'.
 88  CEE349      VALUE X'00030C89 59C3C5C5'.
 88  CEE34A      VALUE X'00030C8A 59C3C5C5'.
 88  CEE34B      VALUE X'00030C8B 59C3C5C5'.
.........
```


Now you SHOULD HAVE THE INFORMATION TO HELP YOU UNDERSTAND:

- WHAT TO DO when you get an ABEND/DUMP
- HOW TO GET the information just in case you DO ABEND/DUMP
- COBOL options that can HELP YOUR DEBUGGING LIFE
- Language Environment RUN-TIME options that can help your DEBUGGING LIFE
- COBOL tricks for debugging
- LE tools for debugging
- Where to FIND THE INFORMATION YOU NEED

RETURN TO INDEX