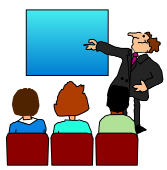
Language Environment Education

Stack and Heap processing

John Monti - March 1999
IBM Poughkeepsie
jmonti@us.ibm.com



Its not easy!

- "Science is built up of facts, as a house is built of stones; but an accumulation of facts is no more a science than a heap of stones is a house." -Henri Poincare -French Mathematician

Agenda

- Introduction
- Overview of LE Stacks
  - Data
  - Layout
  - Processing
- Overview of LE Heap
  - Data
  - Layout
  - Processing
- Heap errors in CEEDUMPs
- Heap errors in SYSTEM DUMPs
- Summary
- Sources and Additional Information

Introduction

- Language Environment
  - Storage Management
    - Stacks
      - Last in first out
      - Allows programs to be reentrant
    - Heaps
      - Completely random access
      - Allows storage to be dynamically allocated at runtime.
    - Enclave level control structures
      - Each 'main' has a separate stack and heap
      - Each 'link' causes a separate stack and heap
      - pthreads share a single heap but have separate stacks.

LE Stacks

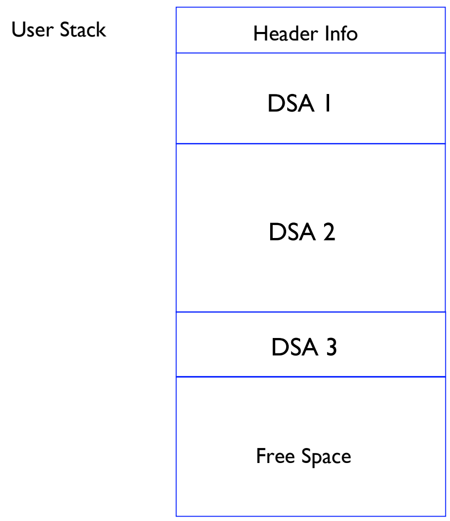
- Overview
  - LE supports 2 independent stacks
    - User stack - most often used
    - Library stack - only used when LE itself needs "below the line" stack.
  - Runtime options dealing with the stacks
    - STACK(init,inc,ANY|BELOW,KEEP|FREE)
    - LIBSTACK(init,inc,KEEP|FREE)
    - STORAGE(...,...,dsa_alloc,...)
    - RPTSTG(ON|OFF)

LE Stacks

- Data
  - DSA
    - Dynamic save area (stack frame)
      - Register save area
      - NAB (next available byte)
      - automatic (local) variables
        - C - int i;
        - PL/I - DCL I FIXED;
        - **NOT** used for COBOL working storage

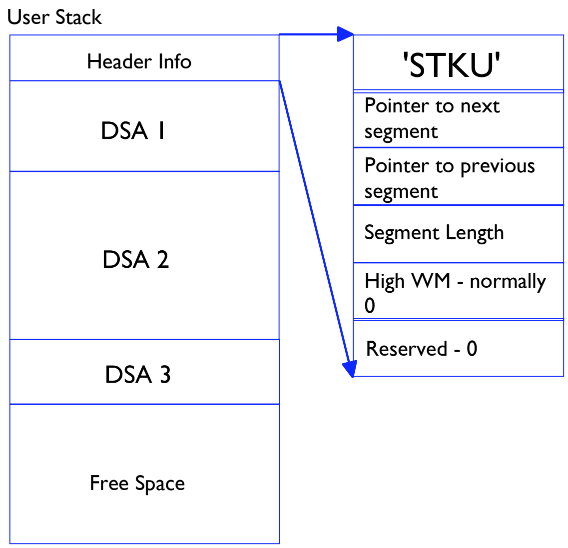
LE Stacks

- Layout - simple stack segment

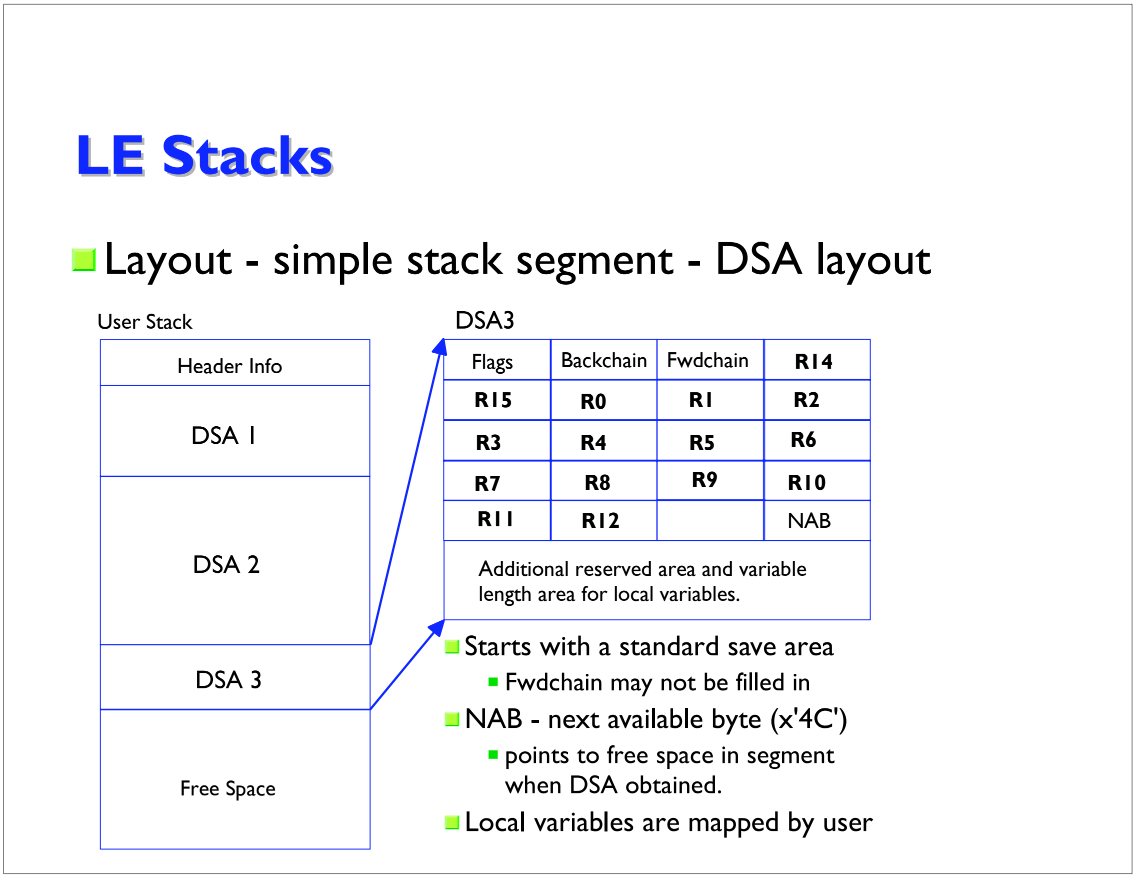


LE Stacks

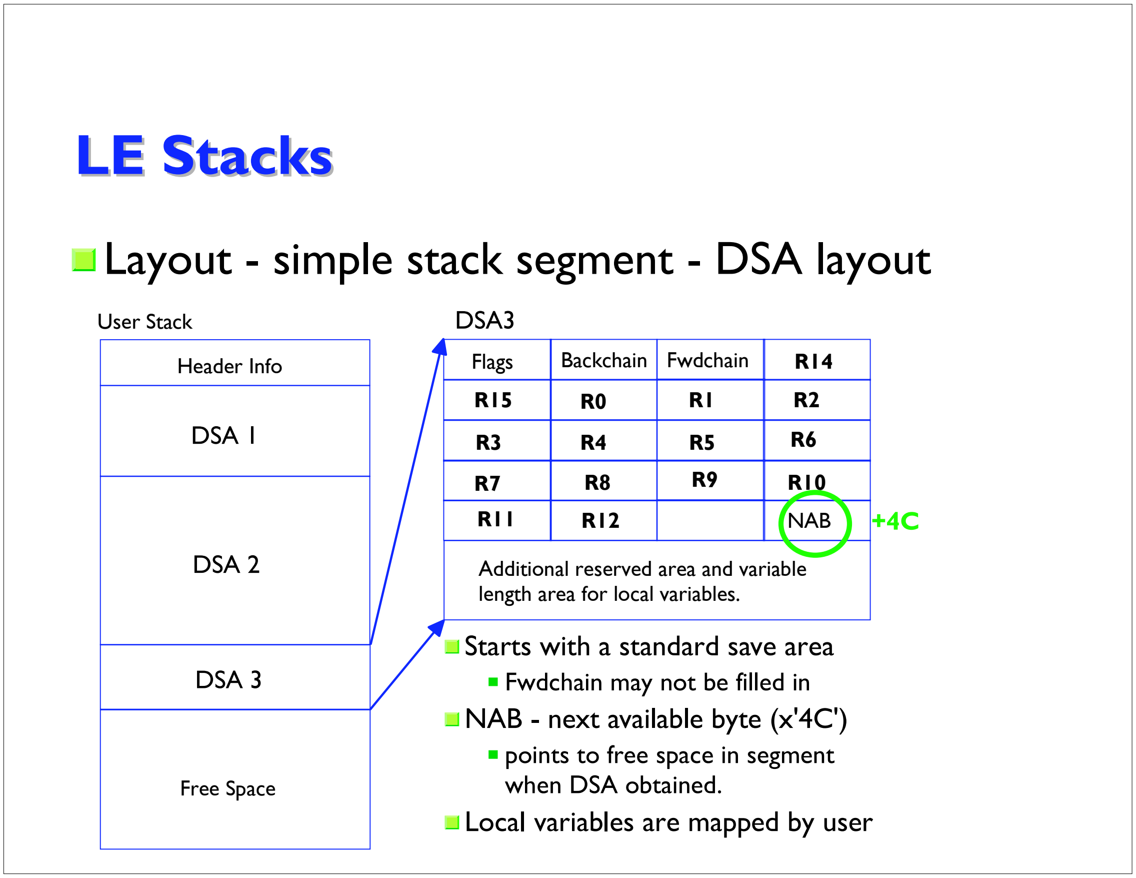
- Layout - simple stack segment - header info



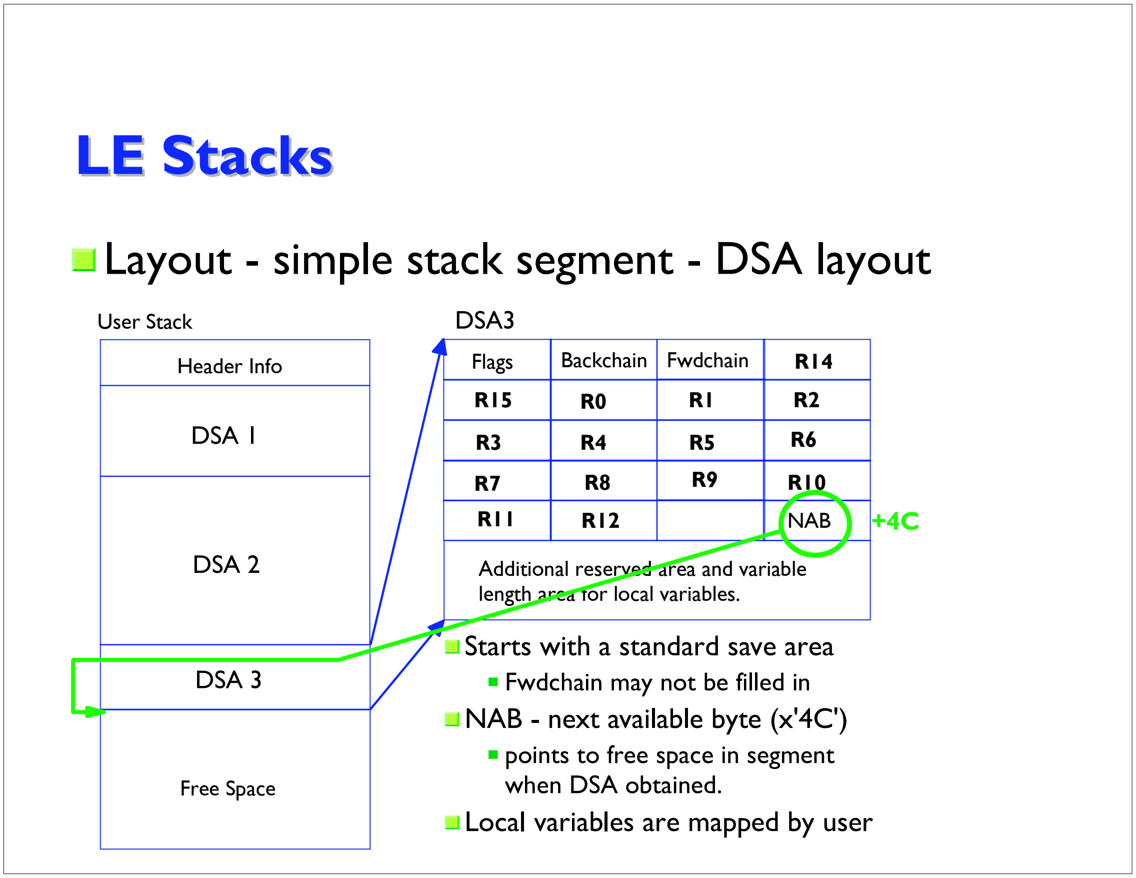
- Eyecatcher will be 'STKL' for LIBSTACK
- If no next and/or prev segment pointers point to SMCB
- Header info is x'18' bytes
- Segment length contains header
  - From STACK or LIBSTACK
  - If request is bigger than runtime option enough storage is obtained.



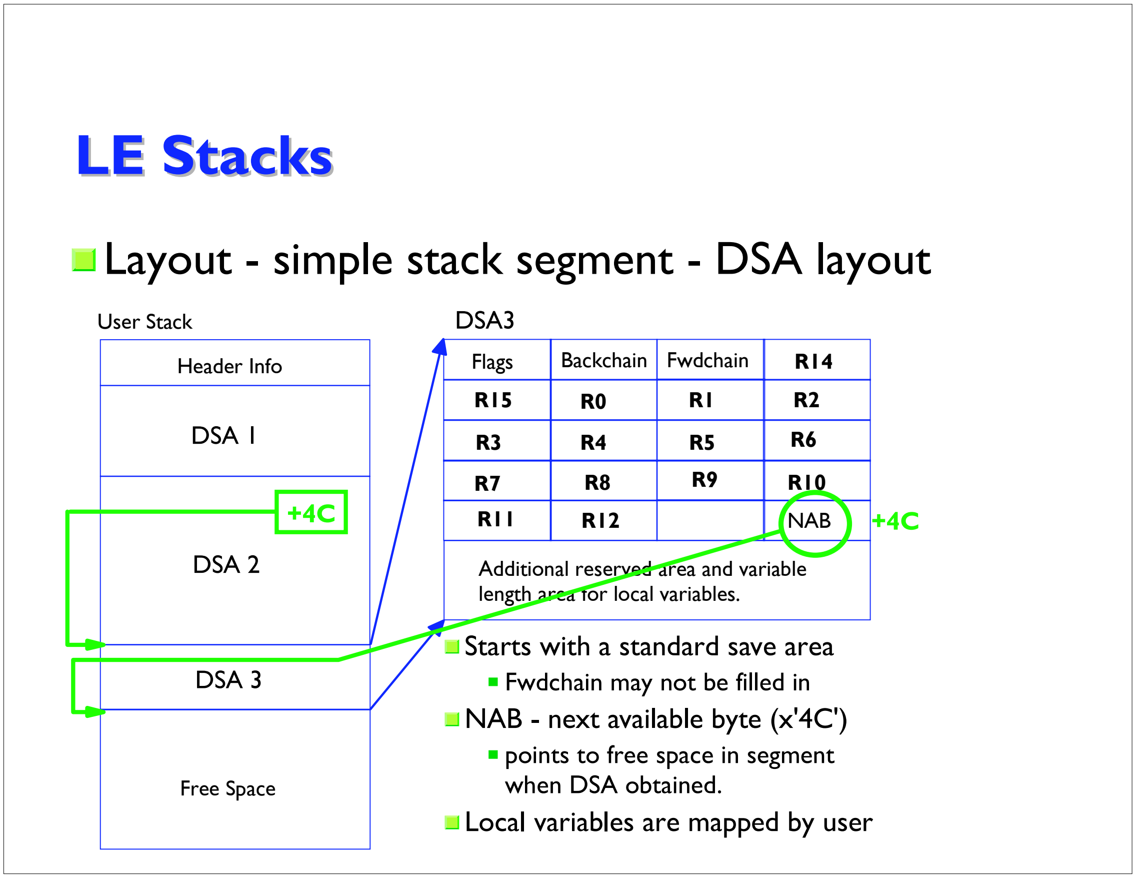
- Go over the chart:
- Then draw point from NAB in DSA3 blow-up to FREE SPACE
- Then draw in other NABs in DSA2 and DSA1 which point to beginning of the next DSAs.
-
- Keep chart available to possibly draw a DSA4 to explain questions



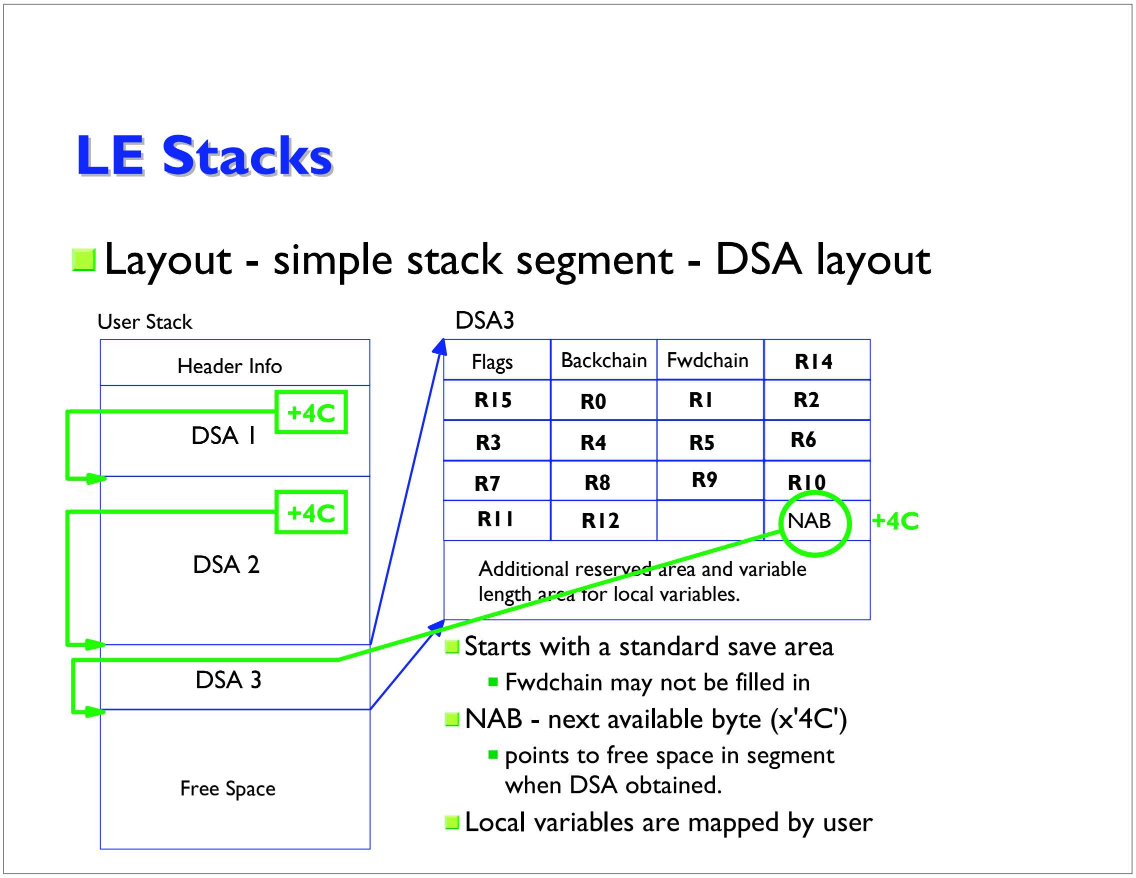
- Go over the chart:
- Then draw point from NAB in DSA3 blow-up to FREE SPACE
- Then draw in other NABs in DSA2 and DSA1 which point to beginning of the next DSAs.
-
- Keep chart available to possibly draw a DSA4 to explain questions



- Go over the chart:
- Then draw point from NAB in DSA3 blow-up to FREE SPACE
- Then draw in other NABs in DSA2 and DSA1 which point to beginning of the next DSAs.
-
- Keep chart available to possibly draw a DSA4 to explain questions



- Go over the chart:
- Then draw point from NAB in DSA3 blow-up to FREE SPACE
- Then draw in other NABs in DSA2 and DSA1 which point to beginning of the next DSAs.
-
- Keep chart available to possibly draw a DSA4 to explain questions



- Go over the chart:
- Then draw point from NAB in DSA3 blow-up to FREE SPACE
- Then draw in other NABs in DSA2 and DSA1 which point to beginning of the next DSAs.
-
- Keep chart available to possibly draw a DSA4 to explain questions

LE Stacks

- Processing - Simple case
  - Program (or function) A calls Program B
    - R13 must point to Prog A's DSA
  - "Prolog Code" in Program B executes
    - Saves Regs in Prog A's DSA
    - Determines size of DSA Prog B requires
    - Checks NAB in Prog A's DSA to determine if stack segment can contain Prog B's DSA
    - Uses area pointed to by NAB as Prog B's DSA
    - Updates Backchain in new DSA
    - Stores new NAB in new DSA

LE Stacks

- Processing - Simple case - continued
  - "Epilog Code" executes to return to Prog A.
    - R13 updated from backchain pointer
    - Registers loaded from Prog A's DSA (R13)
    - Control returned to Prog A via R14
  - NOTES:
    - Prog B's save area not cleared
      - May be useful for debug purposes.
    - No NAB processing takes place
      - NAB in PROG A's DSA again valid

- If question arises you may want to create a DSA4 on the earlier STACK chart and show how DSA3's NAB becomes the active NAB again.

LE Stacks

- Processing - Simple case - continued
  - Sample Prolog Code:

```
00028E  90E6  D00C  STM   r14,r6,12(r13)   Save registers we will be using in caller's DSA
000292  58E0  D04C  L     r14,76(,r13)     Get the NAB from callers save area
000296  4100  E0A8  LA    r0,168(,r14)     Add the size we need (x'A8')
00029A  5500  C314  CL    r0,788(,r12)     Is this still within this segment
00029E  4720  F014  BH    20(,r15)         If not, go get another segment
0002A2  58F0  C280  L     r15,640(,r12)    Return to here. R14=new DSA, R0=new NAB
0002A6  90F0  E048  STM   r15,r0,72(r14)   Update new NAB
0002AA  9210  E000  MVI   0(r14),16        Update flags in DSA
0002AE  50D0  E004  ST    r13,4(,r14)      Update backchain pointer
0002B2  18DE        LR    r13,r14          Set R13 to be the current DSA
0002B4  0530        BALR  r3,r0            Set addressability and go...
0002B6        End of Prolog
```

  - R14 - new DSA
  - R0 - new NAB

LE Stacks

- Processing - Less Simple case
  - Program (or function) A calls Program B
    - R13 must point to Prog A's DSA
  - "Prolog Code" in Program B executes
    - Saves Regs in Prog A's DSA
    - Determines size of DSA Prog B requires
    - Checks NAB in Prog A's DSA to determine if stack segment can contain Prog B's DSA
      - **NOT ENOUGH ROOM!**
      - One of LE's stack overflow routines gets called
      - A new stack segment is created
      - Extra DSA may be inserted into new segment

LE Stacks

- Processing - Less Simple case (continued...)
  - "Prolog Code" in Program B execution...
    - Inserted DSA will be for module CEEVSSFR
      - Used to update SMCB information
      - Used to FREEMAIN stack with FREE option
    - Uses address returned from stack overflow routine as Prog B's DSA
    - Updates Backchain in new DSA
    - Stores new NAB in new DSA residing in new segment

LE Stacks

- Processing - Less Simple case - continued
  - "Epilog Code" executes to return to Prog A.
    - R13 updated from backchain pointer
    - Regs loaded from previous save area (R13)
    - Control is returned via R14
    - Return to CEEVSSFR, segment is collapsed.
    - R13 updated from backchain pointer
    - Registers loaded from Prog A's DSA (R13)
    - Control returned to Prog A via R14
  - NOTES:
    - Prog B's save area not cleared but possibly freed
    - No NAB processing takes place

- That's it for STACKs - ask for questions...

LE Heaps

- Overview
  - Four independently maintained sets of heap segments all with similar layouts:
    - User Heap
      - COBOL W/S
      - C and PL/1 dynamic storage (malloc/allocate)
    - LE Anywhere Heap
      - COBOL and LE above the line CBs
    - LE Below Heap
      - COBOL and LE below the line CBs
    - Additional Heap
      - Defined by the user

LE Heaps

- Overview
  - Runtime options:
    - HEAP(init,inc,ANY|BELOW,KEEP|FREE,...)
    - HEAPCHK(ON|OFF,freq,delay)
    - HEAPPOOLS(ON|OFF,...)
    - ANYHEAP(init,inc,KEEP|FREE)
    - BELOWHEAP(init,inc,KEEP|FREE)
    - STORAGE(heap_alloc,heap_free,...)
    - RPTSTG(ON|OFF)
- DATA
  - User requested storage with a user requested layout

LE Heaps

- Layout
  - Much more complicated than Stack
  - Based on algorithm from Watson Research
    - FAST 1ST!!!, Storage Eff 2nd, Error checks 3rd
  - Header
    - x'20' byte header (8 fullwords of data)
      - Eyecatcher 'HANC'
      - Pointer to previous Heap segment or HPCB
      - Pointer to next Heap segment or HPCB
      - Heapid (user heap = 0)
      - Pointer to beginning of segment
      - Root address (largest free element in segment)
      - Heap Segment Length
      - Root element length (size of largest free element)

- Emphasis the the algorithm is FAST FIRST! This will play in later when we see the error may not be detected immediately
-
- Explain the header - promise a picture later

LE Heaps

- Layout continued…
  - Allocated element
    - 8 byte header
      - Pointer to beginning of heap segment (HANC)
      - Size of element including header
    - User portion
      - Address returned to user

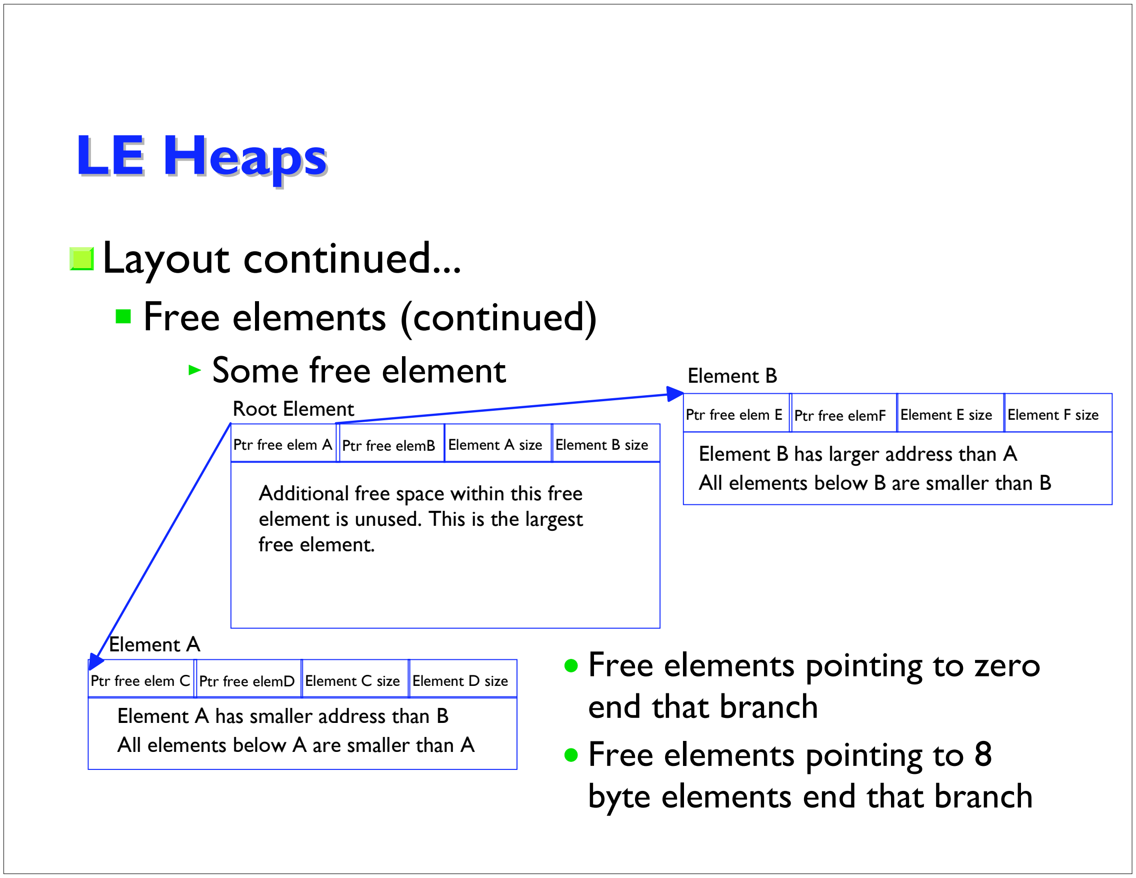
Pointer to HANC	Total size of Element	User data

- The simple stuff first

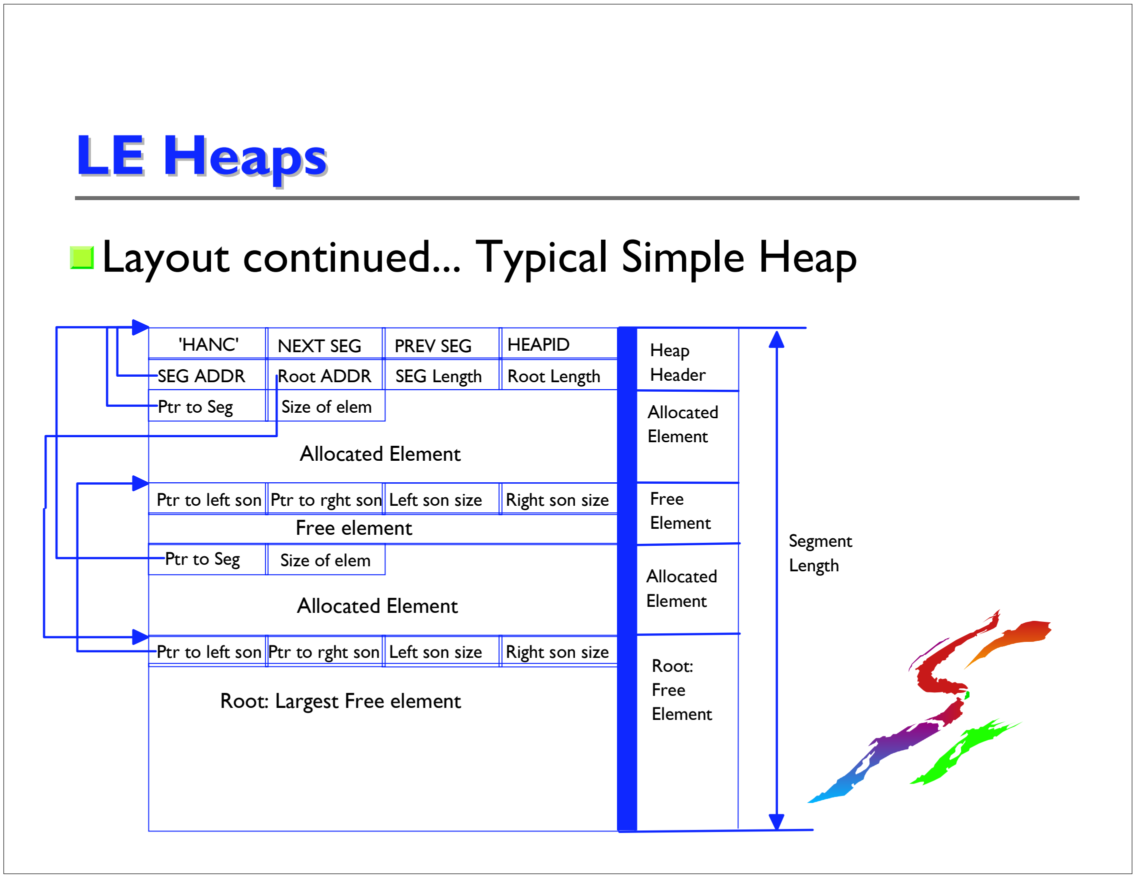
LE Heaps

- Layout continued...
  - Free elements
    - Maintained in a Cartesian Tree
      - Larger elements toward the root
      - Smaller elements toward the leaves
      - Lower addresses to the left
      - Higher addresses to the right
    - Each free area contains x'10' bytes of information about **OTHER** free elements.
      - Left "son" address
      - Right "son" address
      - Left "son" size
      - Right "son" size

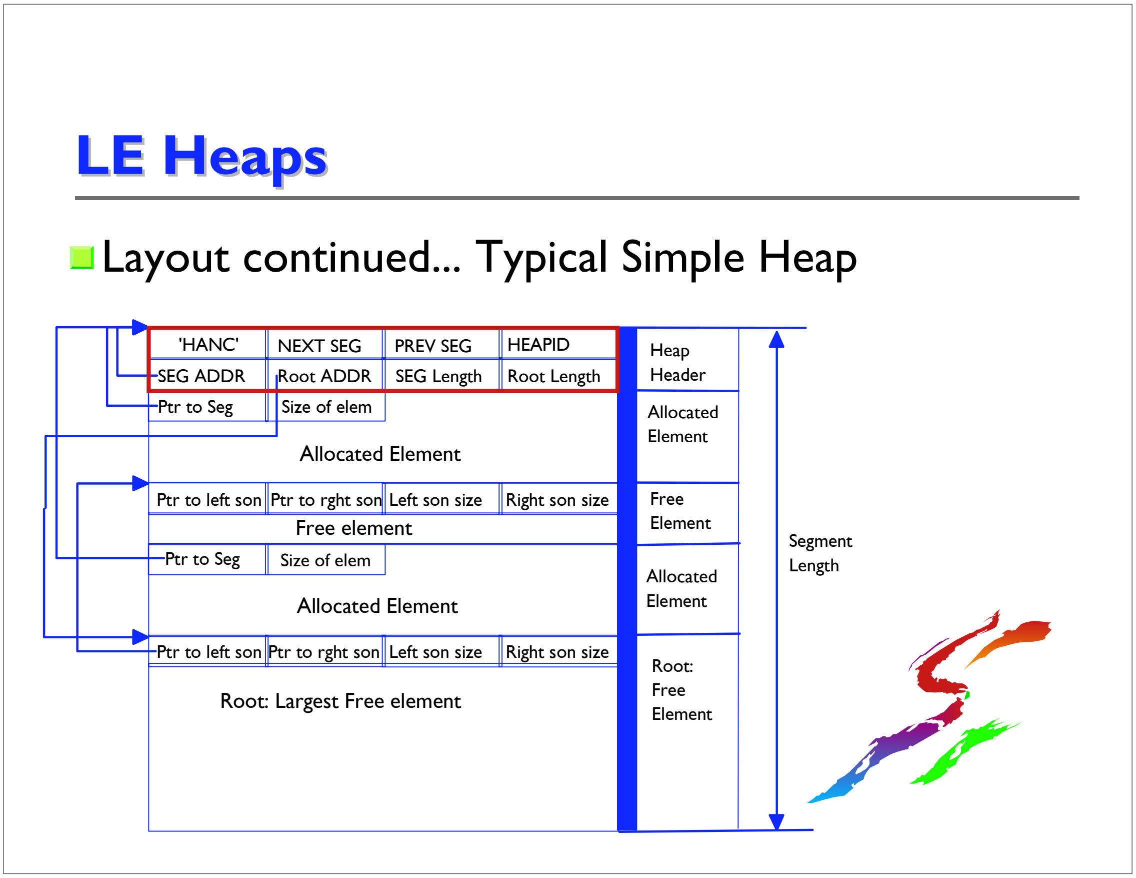
- Free elements are complicated - Don't dwell on them, just explain them at a high level. Pictures are coming....
-



- Explain the pointers and sizes. Stress the info in a free node is about some OTHER free node. Mention both endings to a branch.



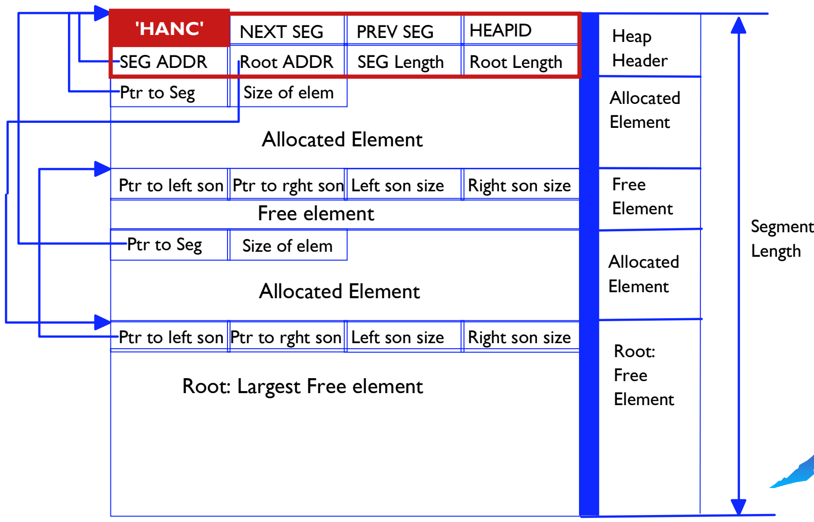
- Highlight the HEADER in RED
- Highlight each allocated element in BLUE
- Highlight each free element in GREEN
- Take your time!



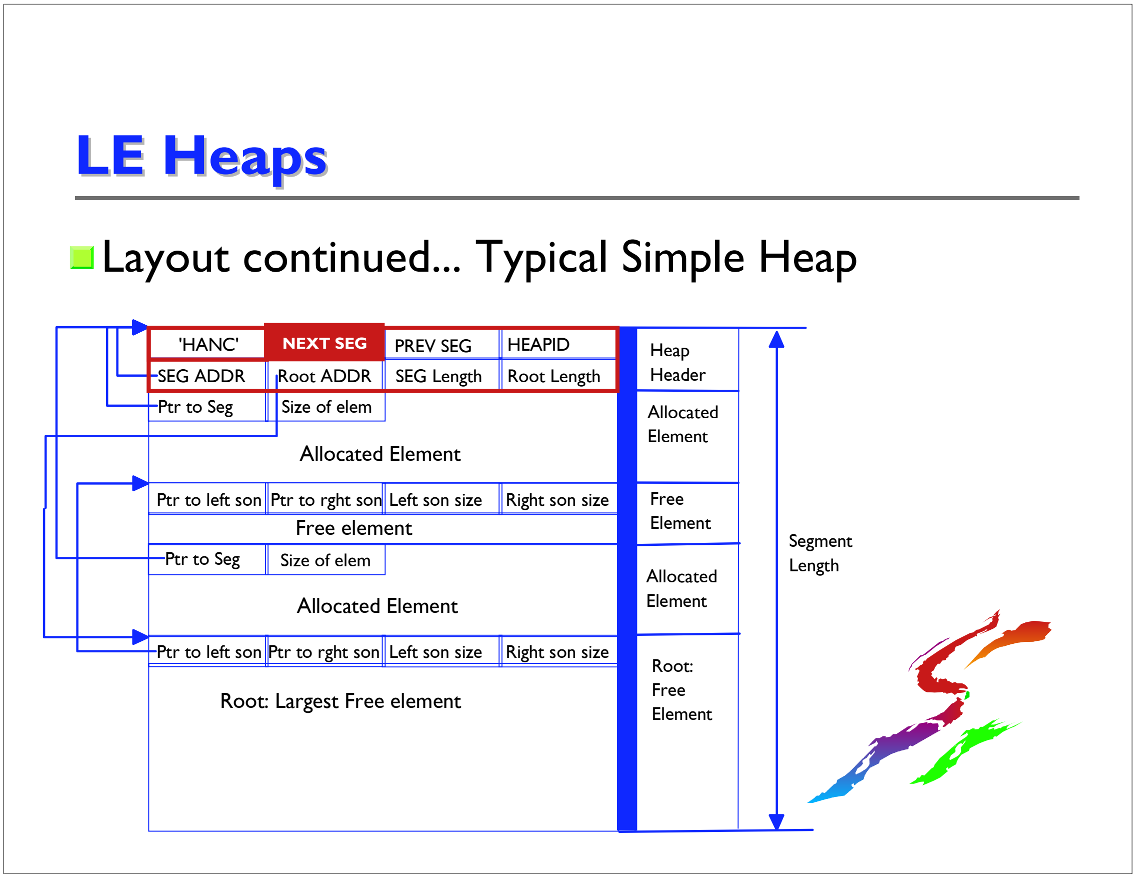
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

- Layout continued… Typical Simple Heap



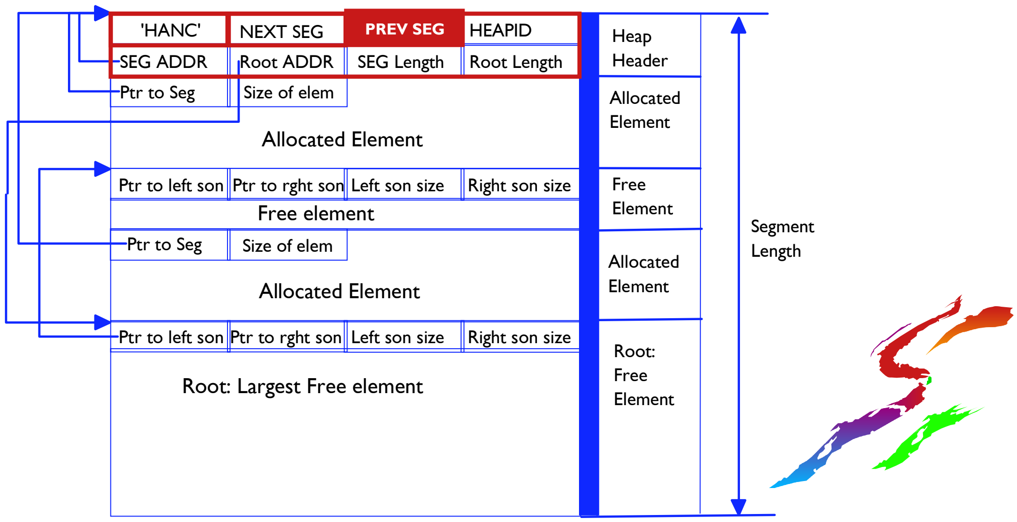
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



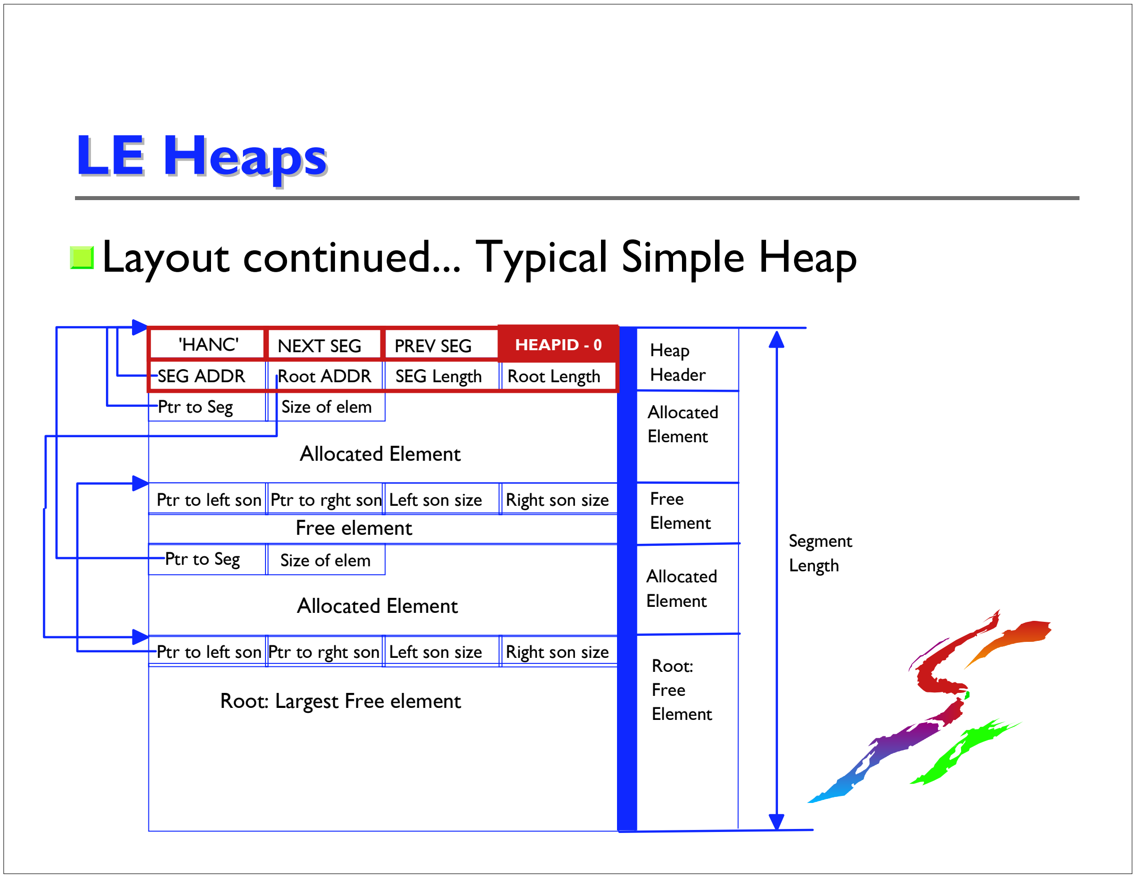
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

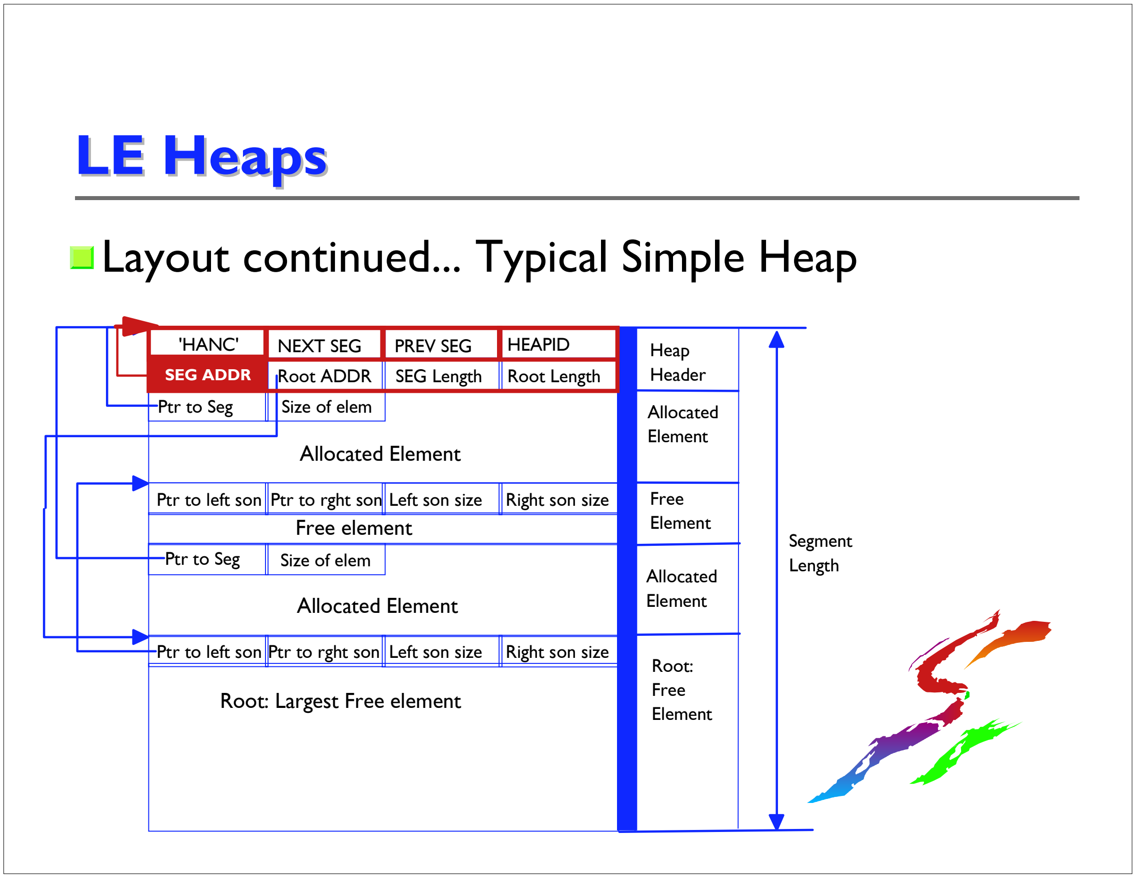
- Layout continued... Typical Simple Heap



- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



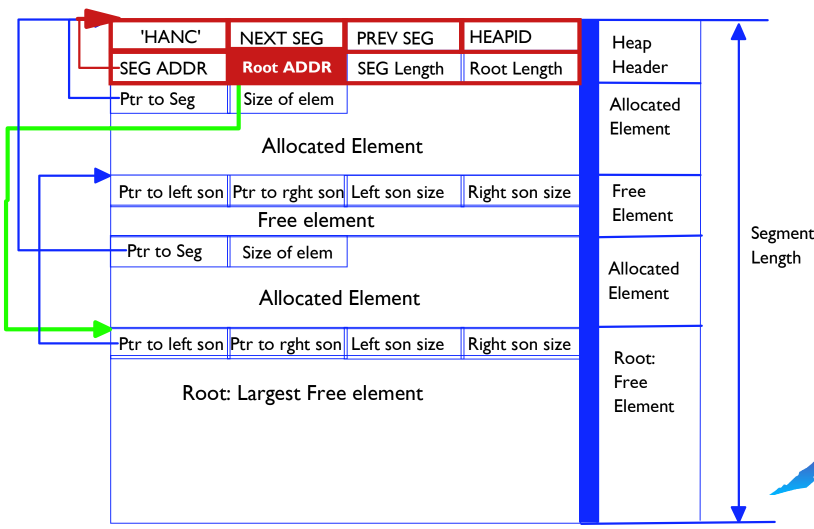
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

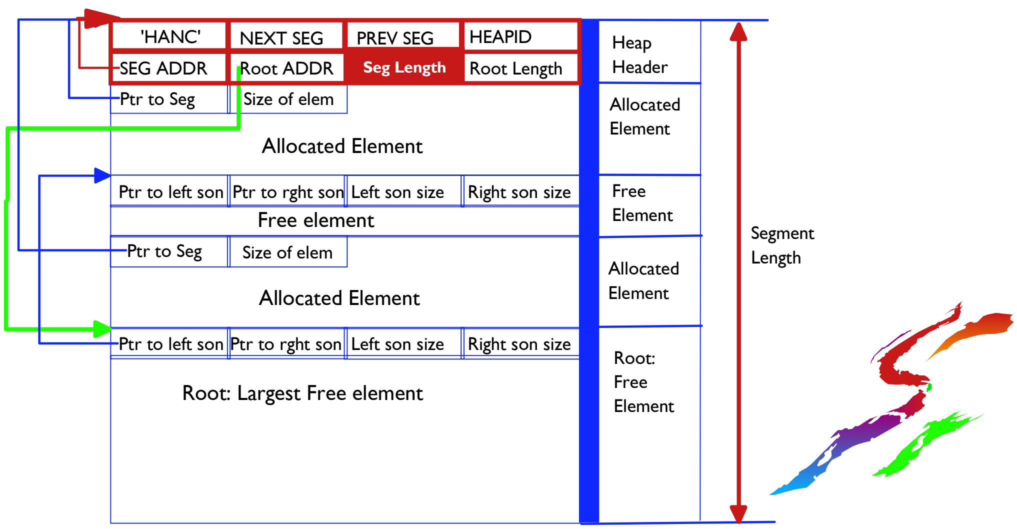
- Layout continued… Typical Simple Heap



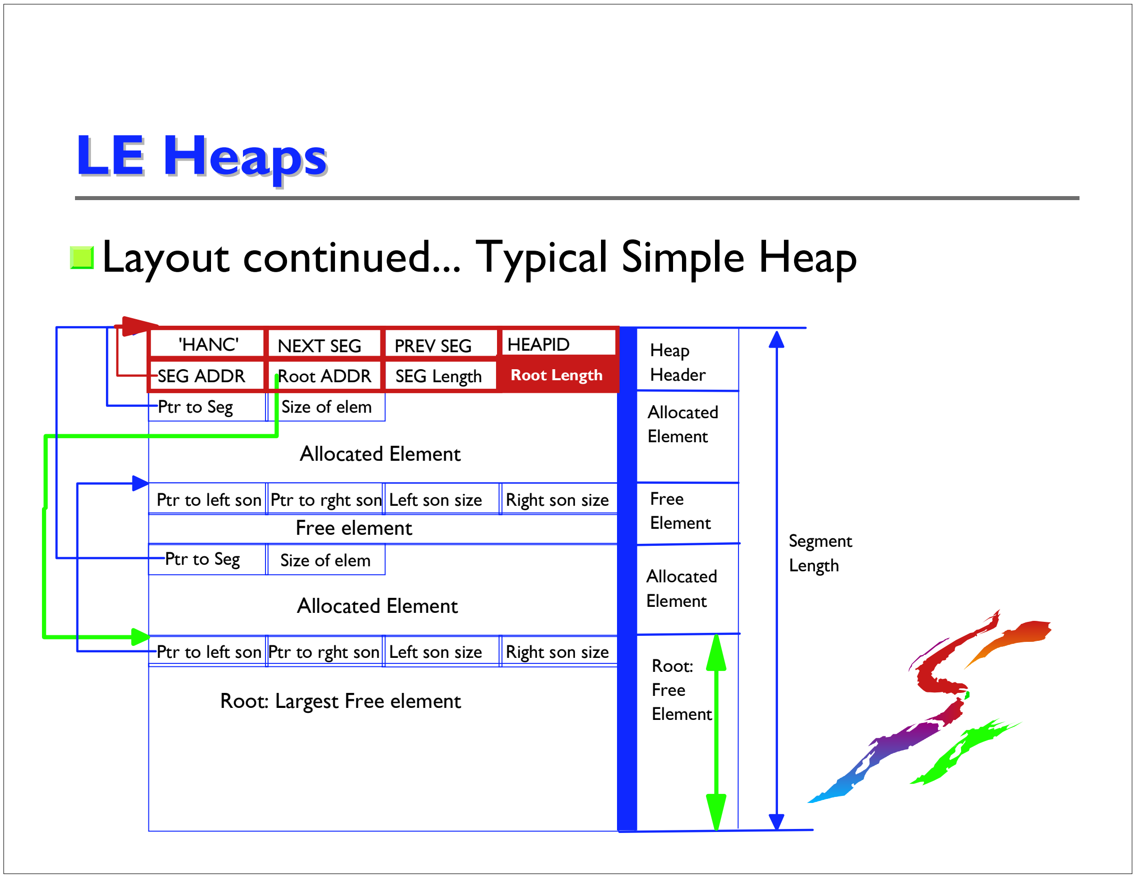
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

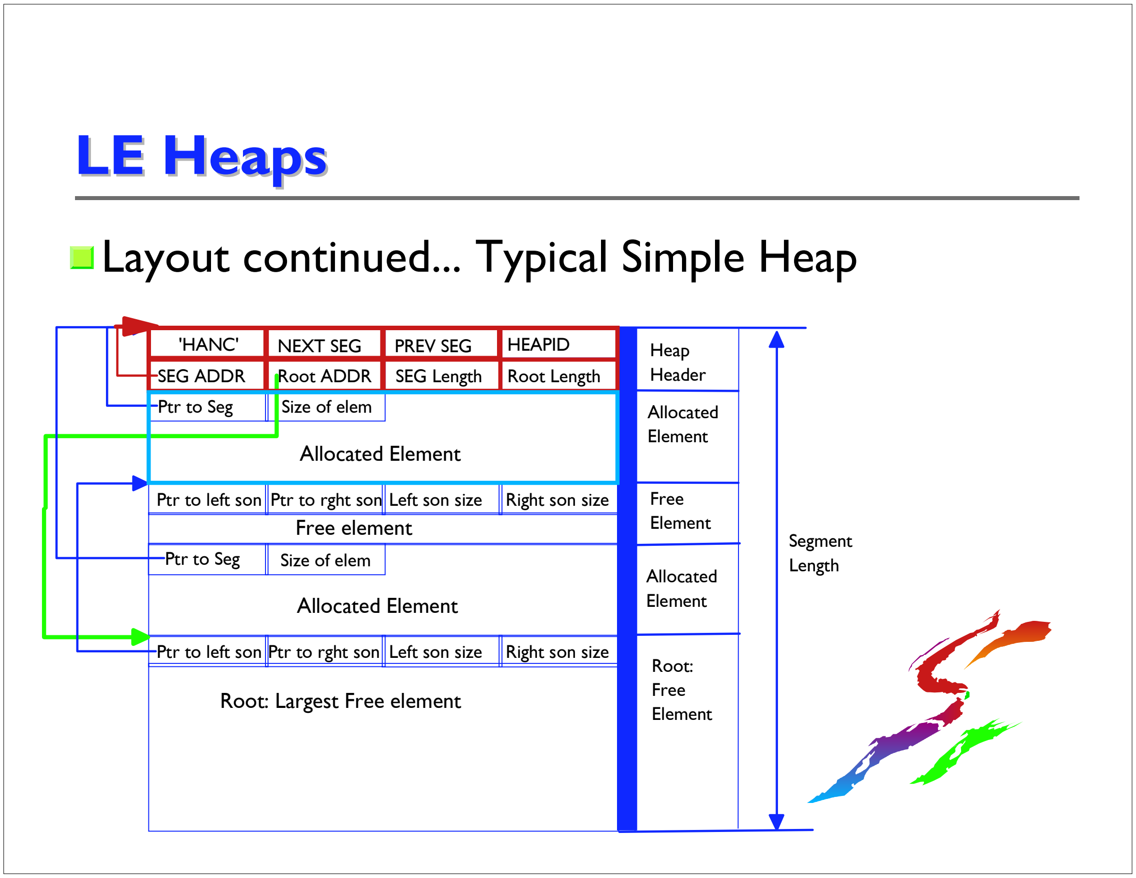
- Layout continued… Typical Simple Heap



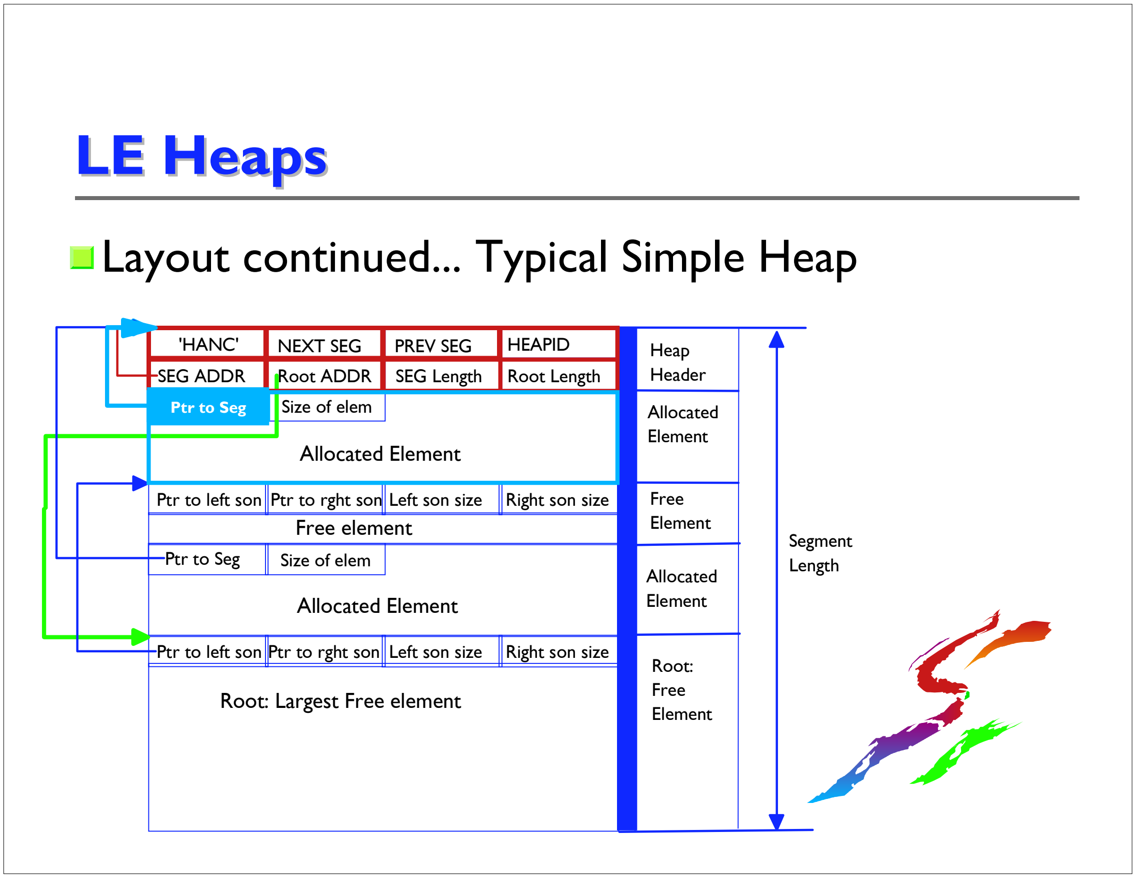
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



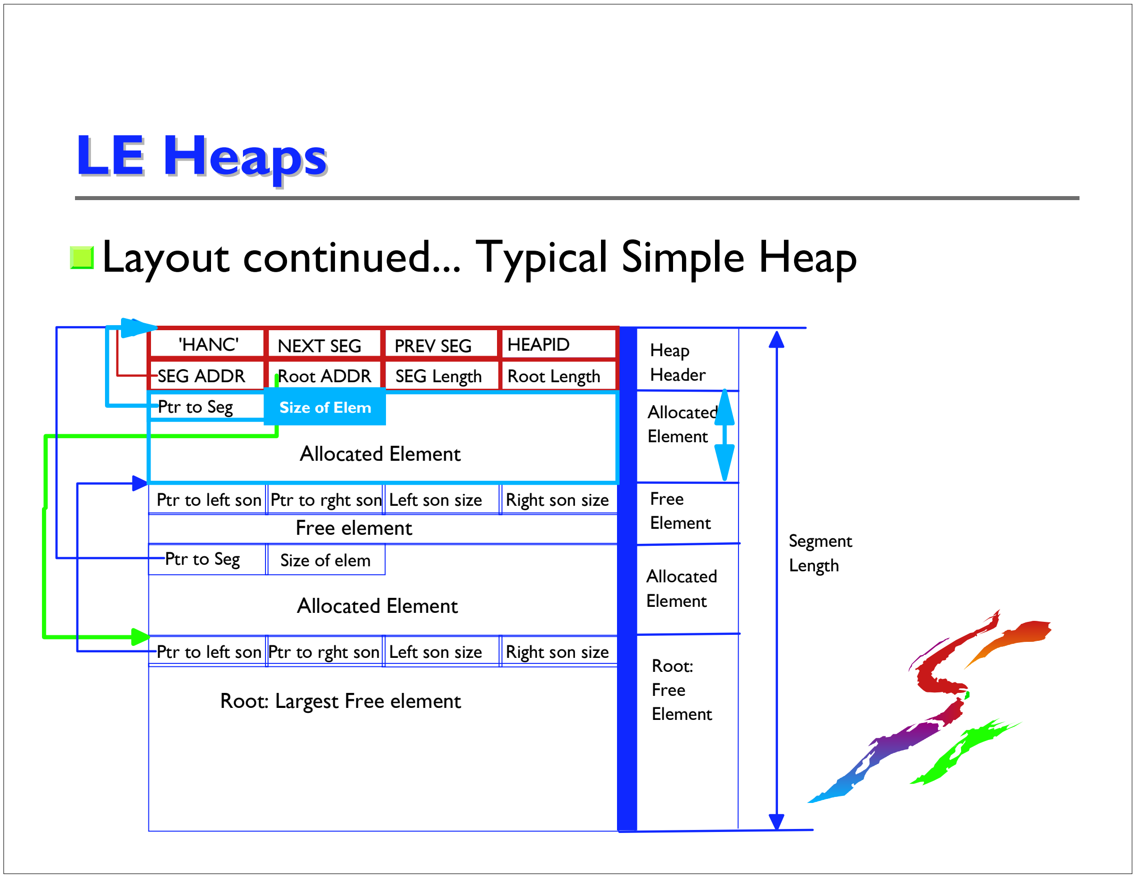
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



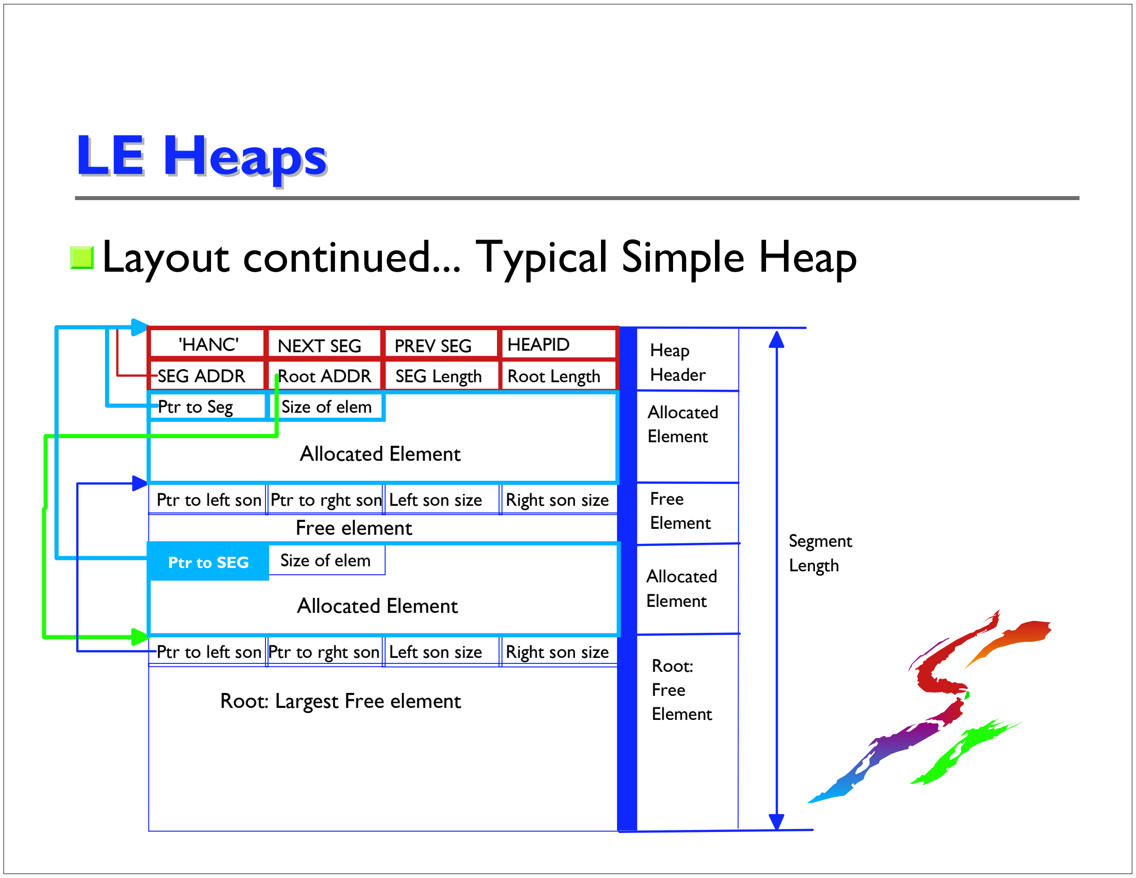
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



- Highlight the HEADER in RED
- Highlight each allocated element in BLUE
- Highlight each free element in GREEN
- Take your time!



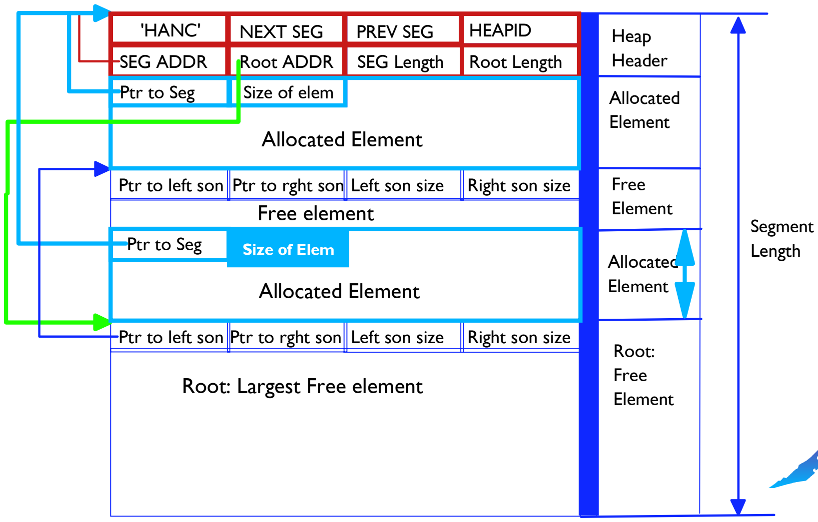
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

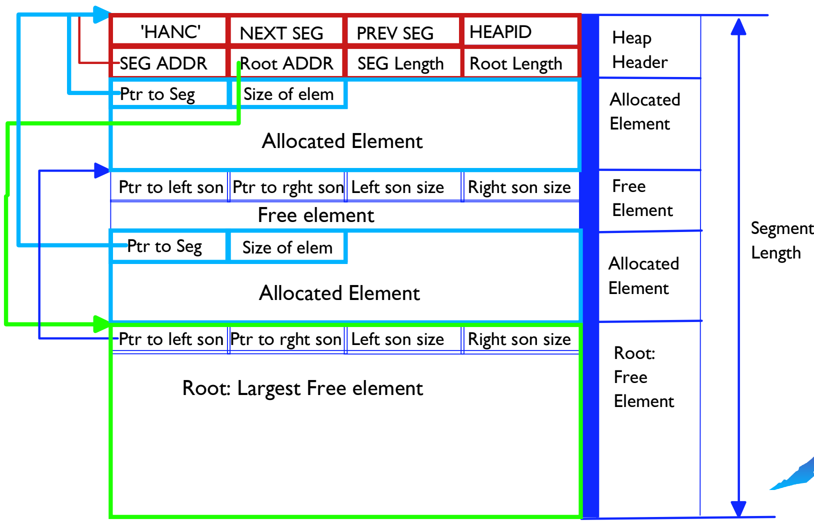
- Layout continued... Typical Simple Heap



- Highlight the HEADER in RED
- Highlight each allocated element in BLUE
- Highlight each free element in GREEN
- Take your time!

LE Heaps

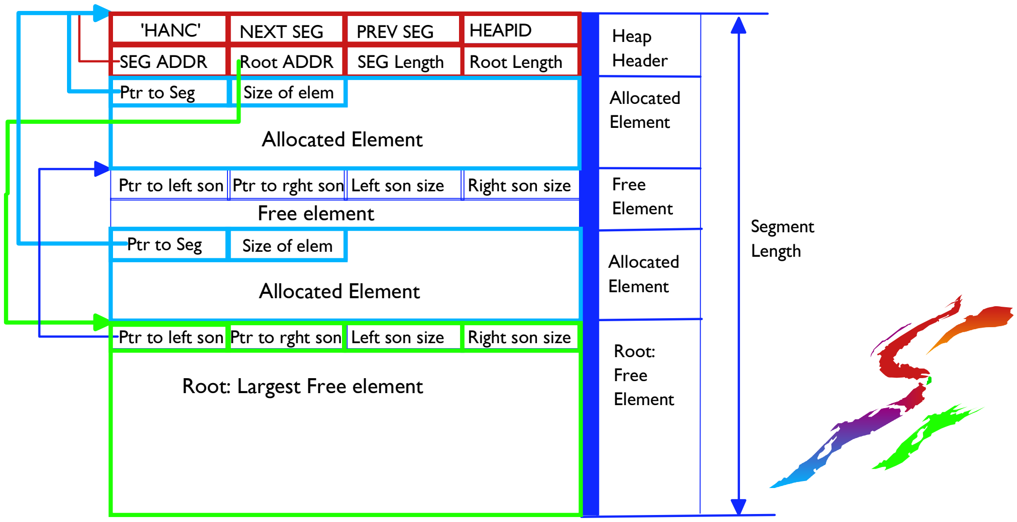
- Layout continued… Typical Simple Heap



- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

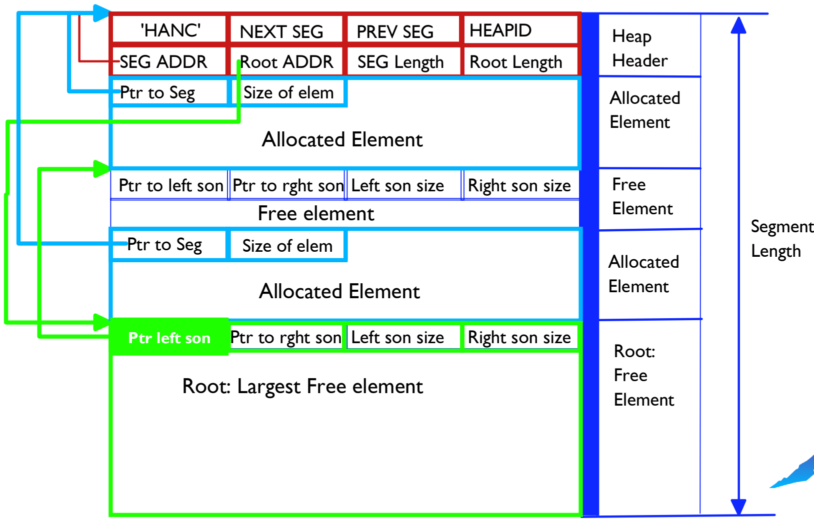
- Layout continued… Typical Simple Heap



- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

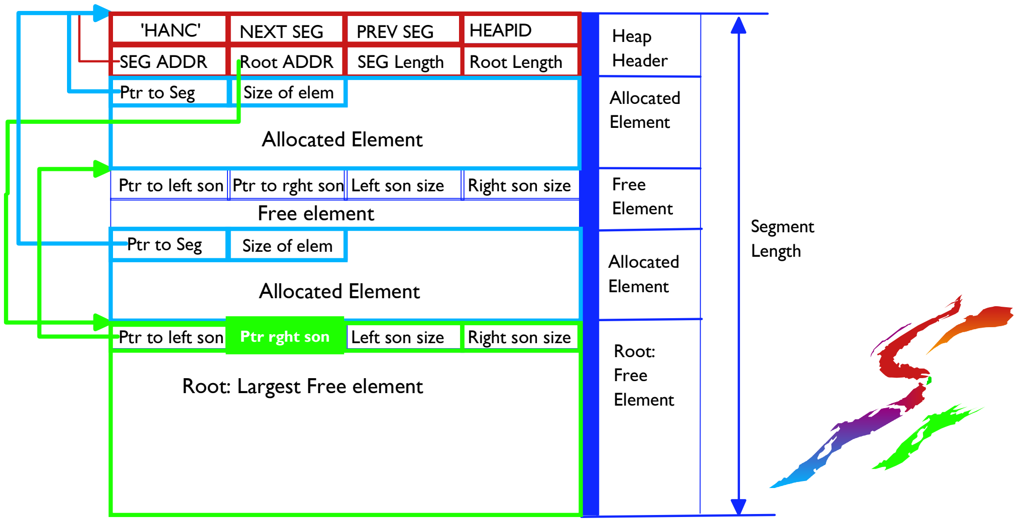
- Layout continued... Typical Simple Heap



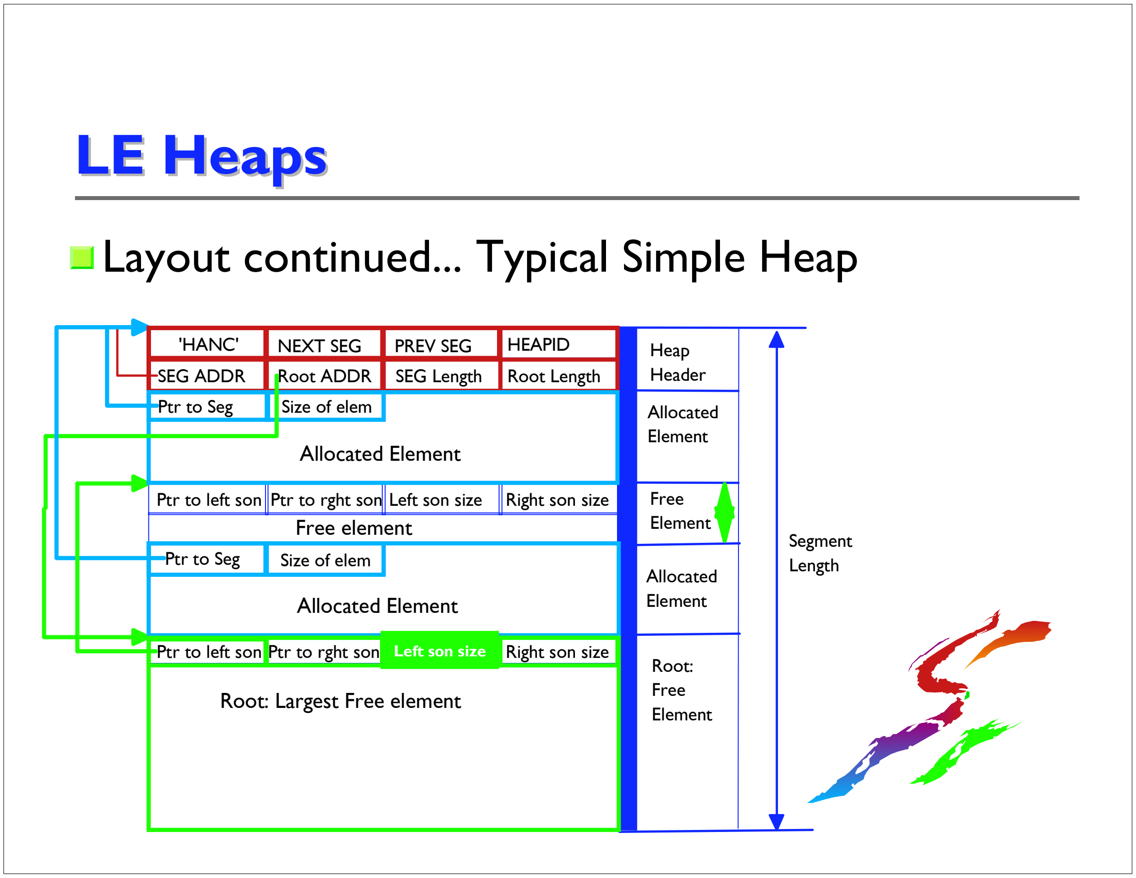
- Highlight the HEADER in RED
- Highlight each allocated element in BLUE
- Highlight each free element in GREEN
- Take your time!

LE Heaps

- Layout continued... Typical Simple Heap



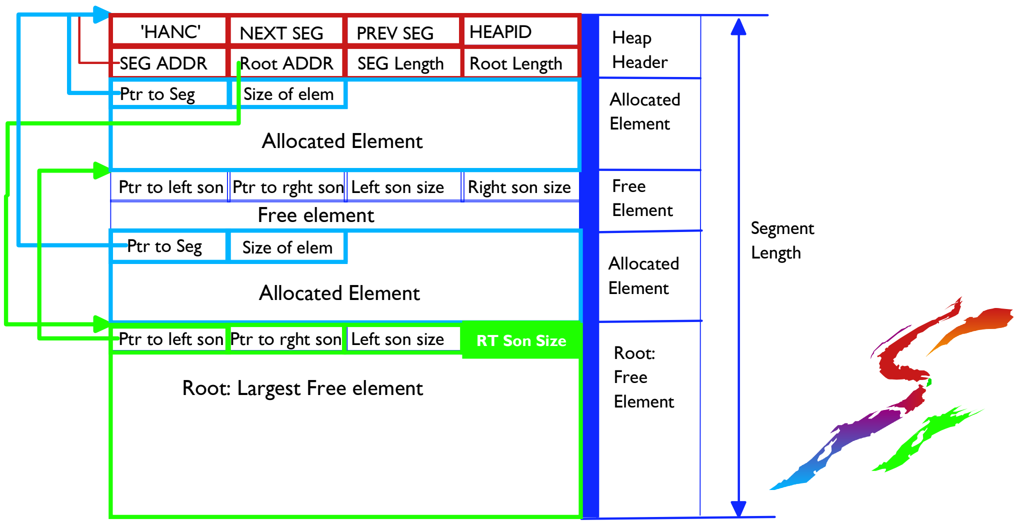
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!



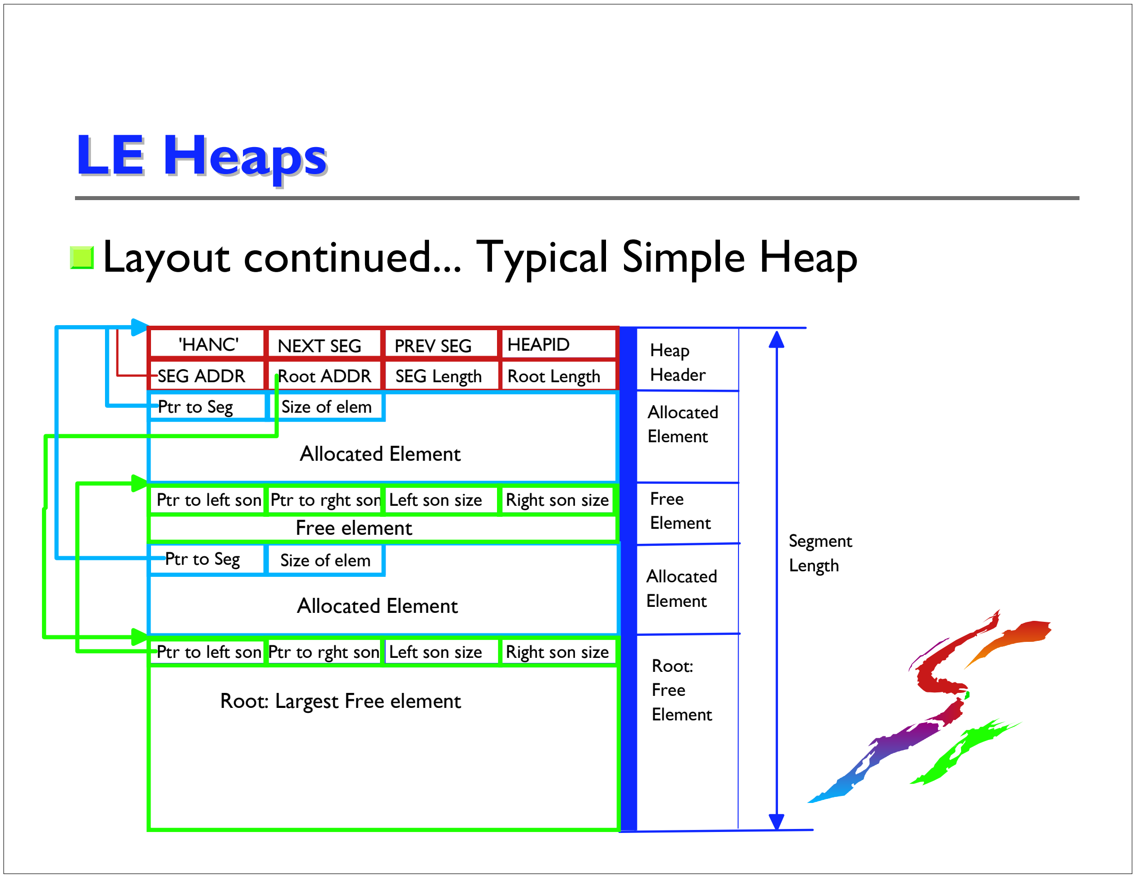
- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

- Layout continued… Typical Simple Heap



- Highlight the HEADER in RED
- Highlight each allocated element in BLUE
- Highlight each free element in GREEN
- Take your time!

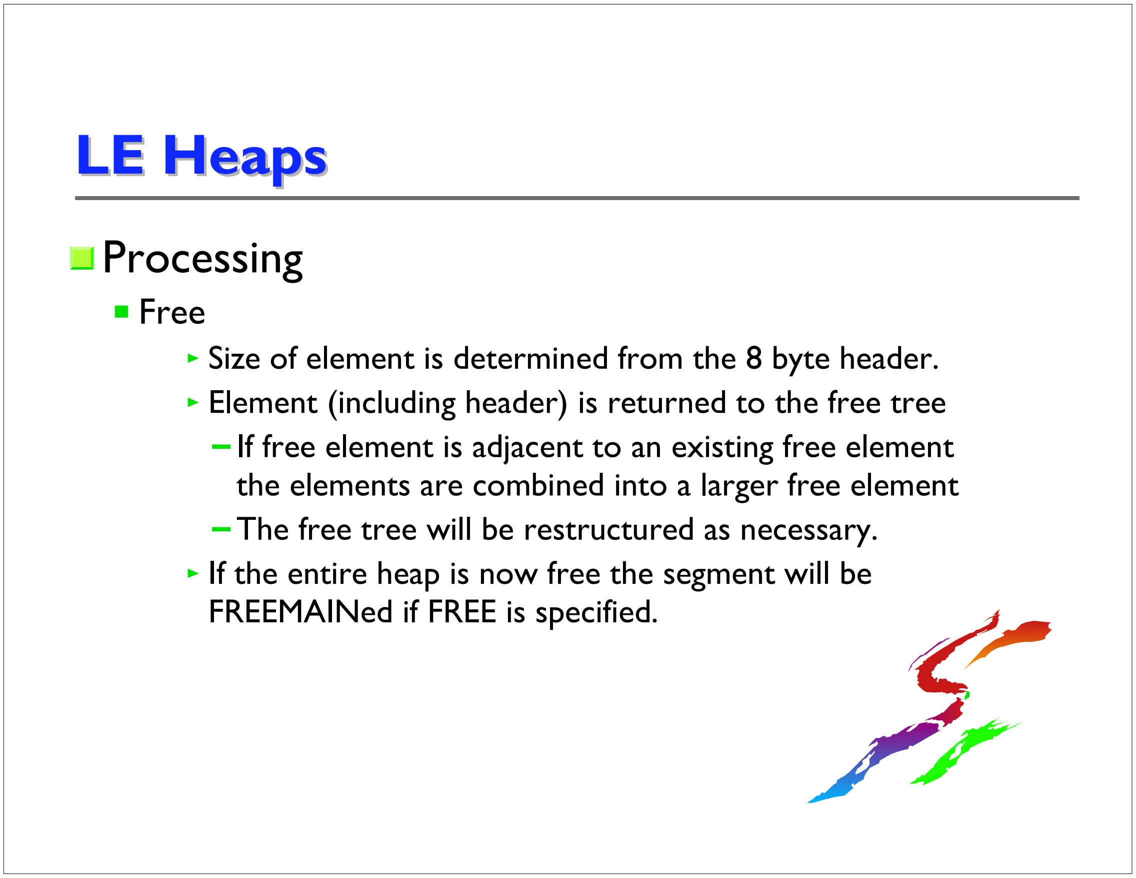


- Highlight the HEADER in RED
-
- Highlight each allocated element in BLUE
-
- Highlight each free element in GREEN
-
- Take your time!

LE Heaps

- Processing
  - Allocation
    - The Free Element tree of the latest heap segment is search starting at the root for the smallest element which will satisfy the request.
    - If no free element found in that segment earlier segments are searched.
    - If no free element is large enough to satisfy the request, an addition heap segment is allocated via GETMAIN.
    - The needed storage from the free element is then allocated (8 byte header filled out).
      - If addition storage is left over in the element it is added to the free tree as a new free element.
    - A pointer to the user storage (after the 8 byte header) is returned to the user.

- Very high level. Explain that it makes sense at a high level
-
- Search the free tree to find a fit
- If not fit search other free trees
- If no fit do a GETMAIN
-
- Allocate the space and return unused portion to free tree
-
- It makes sense...



- Again it makes sense.
- Use the header to determine where the node belongs and free it.
- If adjacent nodes they are combined.
- Simple

Heap Errors in CEEDUMPs

- Sample Program to cause heap damage
  - JMONTI.TEST.C(TESTEDU3)

```
#include <string.h>
void frst_func_called();         /* Function prototypes
*/
void main(int argc, char *argv  )/* Main Program
*/
{
    frst_func_called();
    return;
 }
void frst_func_called()          /* Function: frst_func_called()
*/
{
  char *p1, *p2;                 /* Declare 2 char pointers to use
*/
  p1 = (char *)malloc(16);       /* Malloc 16 bytes of storage for
*/
  p2 = (char *)malloc(16);       /*   each of the 2 pointers
*/
  free(p1);                      /* Free the first pointer
*/
  strcpy(p2,"This string - 16"); /* Initialize the second pointer
*/
  p1 = (char *)malloc(8);        /* Malloc 8 bytes for the 1st
pointer*/
  free(p1);                      /* Free both pointers
*/
  free(p2);
```

Heap Errors in CEEDUMPs

- Sample compile output from TESTEDU3.

```
                        00008 |      *  void frst_func_called()         /* Function: frst_func_called()      */
0002B6  5850  ****      00008 |               L      r5,=A(@STATICC)
                        00009 |      *  {
0002BA  4400  C1B0      00008 |               EX     r0,HOOK..PGM-ENTRY
                                     *    char *p1, *p2;                /* Declare 2 char pointers to use    */
                        00011 |      *    p1 = (char *)malloc(16);      /* Malloc 16 bytes of storage for    */
0002BE  4400  C1AC      00011 |               EX     r0,HOOK..STMT
0002C2  4160  0010      00011 |               LA     r6,16
0002C6  5060  D0A0      00011 |               ST     r6,160(,r13)
0002CA  4110  D0A0      00011 |               LA     r1,160(,r13)
0002CE  58F0  ****      00011 |               L      r15,=V(malloc)
0002D2  4400  C1C0      00011 |               EX     r0,HOOK..CALLBGN
0002D6  05EF            00011 |               BALR   r14,r15
0002D8  4400  C1C4      00011 |               EX     r0,HOOK..CALLRET
0002DC  50F0  D098      00011 |               ST     r15,152(,r13)      ***NOTE:Local variable stored in DSA
                        00012 |      *    p2 = (char *)malloc(16);      /*   each of the 2 pointers          */
0002E0  4400  C1AC      00012 |               EX     r0,HOOK..STMT
0002E4  4160  0010      00012 |               LA     r6,16
0002E8  5060  D0A0      00012 |               ST     r6,160(,r13)
0002EC  4110  D0A0      00012 |               LA     r1,160(,r13)
0002F0  58F0  ****      00012 |               L      r15,=V(malloc)
0002F4  4400  C1C0      00012 |               EX     r0,HOOK..CALLBGN
0002F8  05EF            00012 |               BALR   r14,r15
0002FA  4400  C1C4      00012 |               EX     r0,HOOK..CALLRET
0002FE  50F0  D09C      00012 |               ST     r15,156(,r13)      ***NOTE:Local variable stored in DSA
```

- Chart just to show C local variables are backed in the stack

Heap Errors in CEEDUMPs

- Job Log from program

```
- ======================================================================
-                                    REGION        --- STEP TIMINGS ---
- STEPNAME PROCSTEP PGMNAME     CC     USED      CPU TIME  ELAPSED TIME
- CLG      COMPILE  CBCDRVR     00     176K   00:00:00.60   00:00:19.70
- CLG      LKED     HEWL        00     204K   00:00:00.16   00:00:02.26
IEA995I SYMPTOM DUMP OUTPUT
  USER COMPLETION CODE=4039 REASON CODE=00000000
 TIME=14.11.27  SEQ=03246  CPU=0000  ASID=0206
 PSW AT TIME OF ERROR  478D1400   92B7FA1A  ILC 2  INTC 0D
   ACTIVE LOAD MODULE           ADDRESS=12B19AB8  OFFSET=00065F62
   NAME=CEEPLPKA
   DATA AT PSW  12B7FA14 - 00181610  0A0D58D0  D00498EC
   GPR  0-3  84000000  84000FC7  00020478  12B7FA1A
   GPR  4-7  12B148A0  00000000  000203D0  00021017
   GPR  8-11 12B2A345  12B29346  000203D0  92B7F950
   GPR 12-15 00015910  000225C8  92B299EE  00000000
 END OF SYMPTOM DUMP
IEA993I SYSMDUMP TAKEN TO JMONTI.LEEDU3.SYSMDUMP
IEF450I JMONTI@C GO CLG - ABEND=S000 U4038 REASON=00000001
        TIME=14.12.14
- CLG      GO       PGM=*.DD U4038     296K   00:00:17.34   00:00:49.98
IEF404I JMONTI@C - ENDED - TIME=14.12.14
```

- U4039 in CEEPLPKA - DON'T JUST CALL LE!!!!

Heap Errors in CEEDUMPs

- Output from the job

```
CEE3703I In HANC Control Block, the Eye Catcher is damaged.
CEE3704I Expected data at 00022278 : HANC
  00022258: 01000000 00000000 00000000 00000000  00000000 00000000 00000000 00000004 |................................|
  00022278: 10000000 000221E0 00022320 8001CDAE  92B72BC8 00022320 00022318 92D5C64A |................k..H........kNF.|
CEE0802C Heap storage control information was damaged.
         From compile unit JMONTI.TEST.C(TESTEDU3) at entry point frst_func_called at statement 15 at compile unit
         offset +000000DE at address 12B013B6.
```

  - CEE37xxI messages are new
    - Not designed for this use.
    - Attempt to help with debug, but often not meaningful.
    - We can make them meaningful (later...)
  - CEE0802C message indicates heap damage as well as indicates where call was made to storage routines.

- Show them the 37xx messages are not useful but hint that we can make them meaningful later...

Heap Errors in CEEDUMPs

- CEEDUMP

```
CEE3DMP V1 R8.0: Condition processing resulted in the unhandled condition.        10/02/98 2:11:27 PM                 Page:  1

Information for enclave main

  Information for thread 8000000000000000

  Traceback:
    DSA Addr  Program Unit  PU Addr   PU Offset  Entry         E Addr    E  Offset   Statement  Load Mod  Service  Status
    00020018  CEEHDSP       12B27348  +0000264E  CEEHDSP       12B27348  +0000264E              CEEPLPKA  UQ19695  Call
    00022438  CEEHSGLT      12B9C020  +0000005C  CEEHSGLT      12B9C020  +0000005C              CEEPLPKA  UQ09246  Exception
    00022320  CEEV#GH       12B72BC8  +0000055C  CEEV#GH       12B72BC8  +0000055C              CEEPLPKA  UQ13761  Call
    00022278  JMONTI.TEST.C(TESTEDU3)
                            12B012D8  +000000DE  frst_func_called
                                                               12B012D8  +000000DE         15  TESTEDU            Call
    000221E0  JMONTI.TEST.C(TESTEDU3)
                            12B01070  +0000005E  main          12B01070  +0000005E          5  TESTEDU            Call
    000220C8                12D5C596  -12D461F8  EDCZMINV      12D5C596  -12D461F8              CEEEV003           Call
    00022018  CEEBBEXT      0000E690  +0000013C  CEEBBEXT      0000E690  +0000013C              CEEBINIT  UQ09246  Call
```

  - Not reported as a program check (software detected)
  - CEEV#GH reported CEE0802 condition to handler
  - frst_func_called called (via malloc) "get heap"

- Emphasis that CEEDUMPs are not the way to go. Explain what we see here from that point of view. Its somewhat confusing and sometimes misleading. We need the SYSTEM DUMP with he formatters!!!!

Heap Errors in CEEDUMPs

- CEEDUMP - continued

```
Condition Information for Active Routines
  Condition Information for CEEHSGLT (DSA address 00022438)
    CIB Address: 00020478
    Current Condition:
      CEE0198S The termination of a thread was signaled due to an unhandled condition.
    Original Condition:
      CEE0802C Heap storage control information was damaged.
    Location:
      Program Unit: CEEHSGLT Entry: CEEHSGLT Statement:  Offset: +0000005C
  Storage dump near condition, beginning at location: 12B9C06C
    +000000 12B9C06C  F010D20B D0801000 58A0C2B8 58F0A01C  05EFD20B D098B108 41A0D098 50A0D08C  |0.K.......B..0....K..q.....q&...|
```

  - Condition information not as useful for software detected errors
  - No Machine State information (no program check)

Heap Errors in CEEDUMPs

- CEEDUMP - continued

```
Initial (User) Heap                                       : 12E77000
  +000000 12E77000  C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20
|HANC..(...(......X...X........".|
  +000020 12E77020  12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018
|.X.......................X......|
  +000040 12E77040  00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A
|.....X...X.......X..............|
  +000060 12E77060  12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038
|.X......TESTEDU.........X......|
  +000080 12E77080  C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000  |CDLL....
.........0.............|
  +0000A0 12E770A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
|................................|
  +0000C0 12E770C0  00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6  |.........X......This string -
16|
  +0000E0 12E770E0  00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
|.X..............................|
```

- Errors normally in User Heap so check there first
  - First x'20' bytes are the header
    - Root at 12E770E0
    - x'18' byte element at 12E77020
    - x'18' byte element at 12E77038
    - x'10' byte element at 12E77050
    - x'18' byte element at 12E77060
    - x'38' byte element at 12E77078
    - x'18' byte element at 12E770C8 (p2)
    - Free element at 12E770E0 points to another free element at 00E770B0 of size x'18'
    - The free element has been overlaid!!!!!!!!!

- Go through and mark the header in RED - draw in pointers
-
- Find the allocated elements and highlight in BLUE
- (Indicate the element we miss is probably a free element)
-
- Run the FREE tree and find the error - Ask why???

Heap Errors in CEEDUMPs

- CEEDUMP - continued

```
Initial (User) Heap                                    : 12E77000
  +000000 12E77000  C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20
|HANC..(...(......X...X........".|
  +000020 12E77020  12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018
|.X......................X......|
  +000040 12E77040  00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A
|.....X...X.......X.............|
  +000060 12E77060  12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038
|.X......TESTEDU..........X......|
  +000080 12E77080  C3C4D3D3 00000000 40000000 00000000  12B00EE0 12B01670 00000000 00000000  |CDLL....
.........0...........|
  +0000A0 12E770A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
|................................|
  +0000C0 12E770C0  00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6  |.........X......This string -
16|
  +0000E0 12E770E0  00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
|.X..............................|
```

- Errors normally in User Heap so check there first
  - First x'20' bytes are the header
    - Root at 12E770E0
    - x'18' byte element at 12E77020
    - x'18' byte element at 12E77038
    - x'10' byte element at 12E77050
    - x'18' byte element at 12E77060
    - x'38' byte element at 12E77078
    - x'18' byte element at 12E770C8 (p2)
    - Free element at 12E770E0 points to another free element at 00E770B0 of size x'18'
    - The free element has been overlaid!!!!!!!!!

- Go through and mark the header in RED - draw in pointers
-
- Find the allocated elements and highlight in BLUE
- (Indicate the element we miss is probably a free element)
-
- Run the FREE tree and find the error - Ask why???

```
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20
12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018
00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A
12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038
C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6
00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

```
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20
12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018
00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A
12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038
C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6
00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

HANC

```
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20

12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018

00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A

12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038

C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000

00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000

00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6

00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

HANC NEXT

```
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20

12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018

00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A

12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038

C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000

00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000

00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6

00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

```
HANC      NEXT      PREV
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20

12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018

00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A

12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038

C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000

00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000

00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6

00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

HANC	NEXT	PREV	Heap ID				
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC NEXT PREV Heap ID Ptr Seg

```
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20
12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018
00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A
12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038
C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000
00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6
00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

```
HANC      NEXT      PREV      Heap ID    Ptr Seg   Root @
C8C1D5C3  00014D48  00014D48  00000000   12E77000  12E770E0  00008000  00007F20

12E77000  00000018  00000000  00000000   00000000  00000000  12E77000  00000018

00000001  12E77048  12E77068  00000000   12E77000  00000010  00000000  0001380A

12E77000  00000018  E3C5E2E3  C5C4E400   00000000  00000000  12E77000  00000038

C3C4D3D3  00000000  40000000  00000000   12B00FF0  12B01670  00000000  00000000

00000000  00000000  00000000  00000000   00000000  00000000  00000000  00000000

00000000  00000000  12E77000  00000018   E38889A2  40A2A399  89958740  6040F1F6

00E770B0  00000000  00000018  00000000   00000000  00000000  00000000  00000000
```

HANC NEXT PREV Heap ID Ptr Seg Root @ Seg Size

```
C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20

12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018

00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A

12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038

C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000

00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000

00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6

00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
```

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

Heap Errors in CEEDUMPs

- CEEDUMP - continued

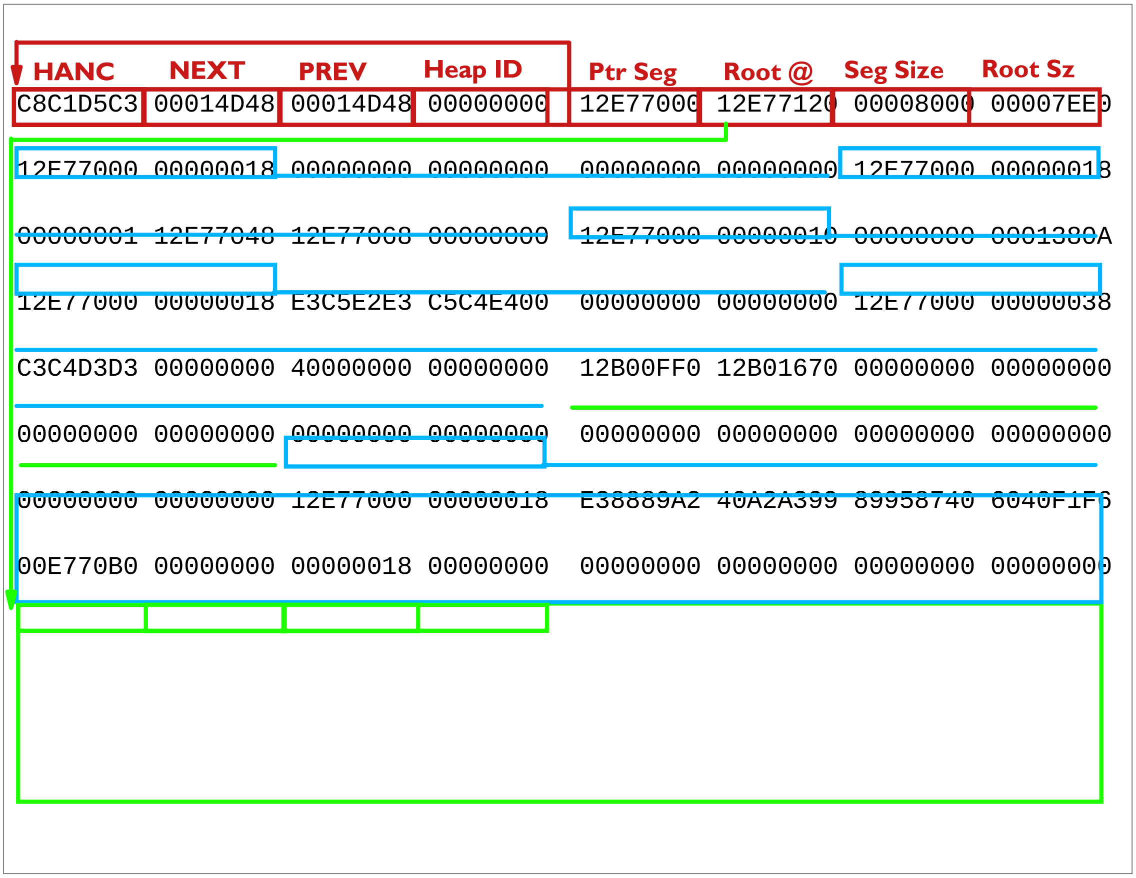
```
Initial (User) Heap                                          : 12E77000
  +000000 12E77000  C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E770E0 00008000 00007F20  |HANC..(...(......X...X........".|
  +000020 12E77020  12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018  |.X.......................X......|
  +000040 12E77040  00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A  |.....X...X.......X..............|
  +000060 12E77060  12E77000 00000018 E3C5E2E3 C5C4E400  00000000 00000000 12E77000 00000038  |.X......TESTEDU..........X......|
  +000080 12E77080  C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000  |CDLL....  .........0............|
  +0000A0 12E770A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |................................|
  +0000C0 12E770C0  00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6  |.........X......This string - 16|
  +0000E0 12E770E0  00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000  |.X..............................|
```

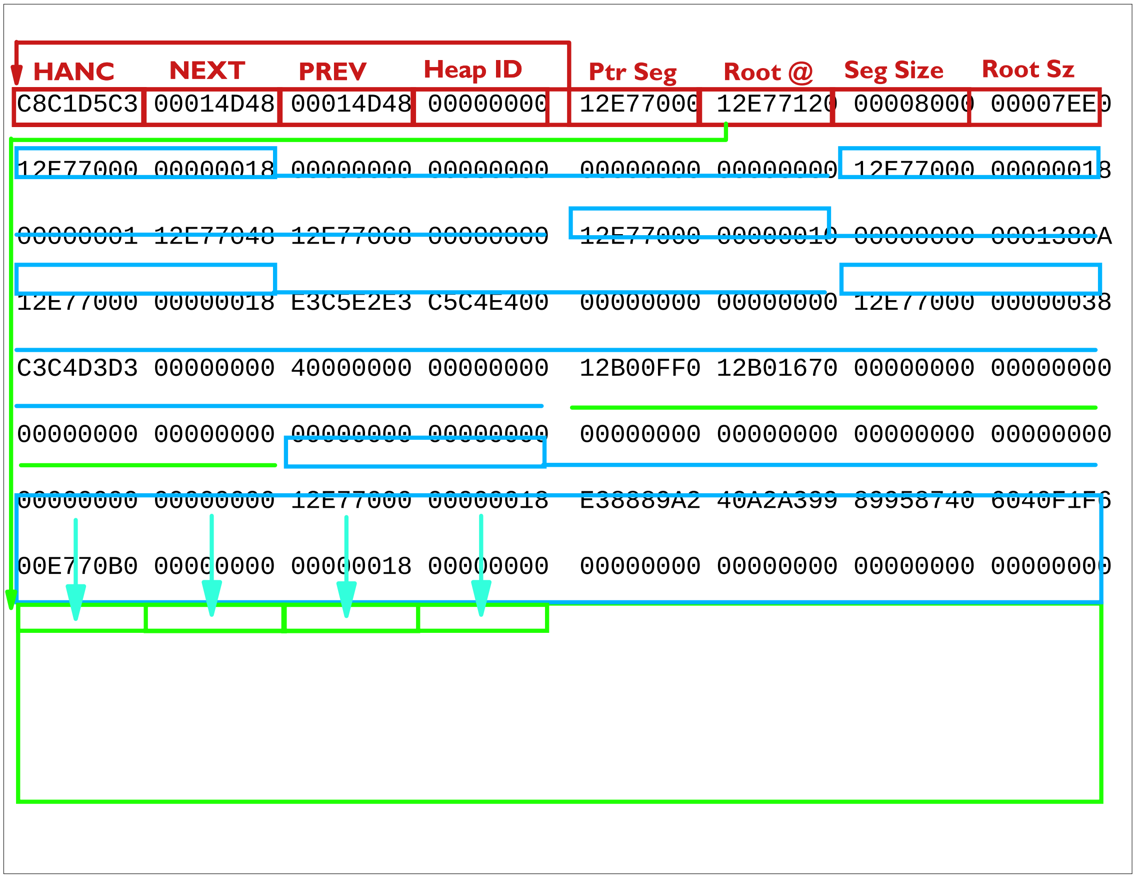
  - Classic "C/C++" problem.
    - Allocation for P2 did not include room for NULL terminator. strcpy() overlaid the free node info.
  - The overlay resulted in the CEE0802C error.
  - This is a simple case with a single small heap.
  - Large heaps and/or multiple heaps are difficult to diagnose by hand.
  - Problem 1!!!!

- Explain problem 1 - Error difficult to find by hand.
-
- Seen dumps with 40 Heaps of 64K each.
-
- Promise a solution.

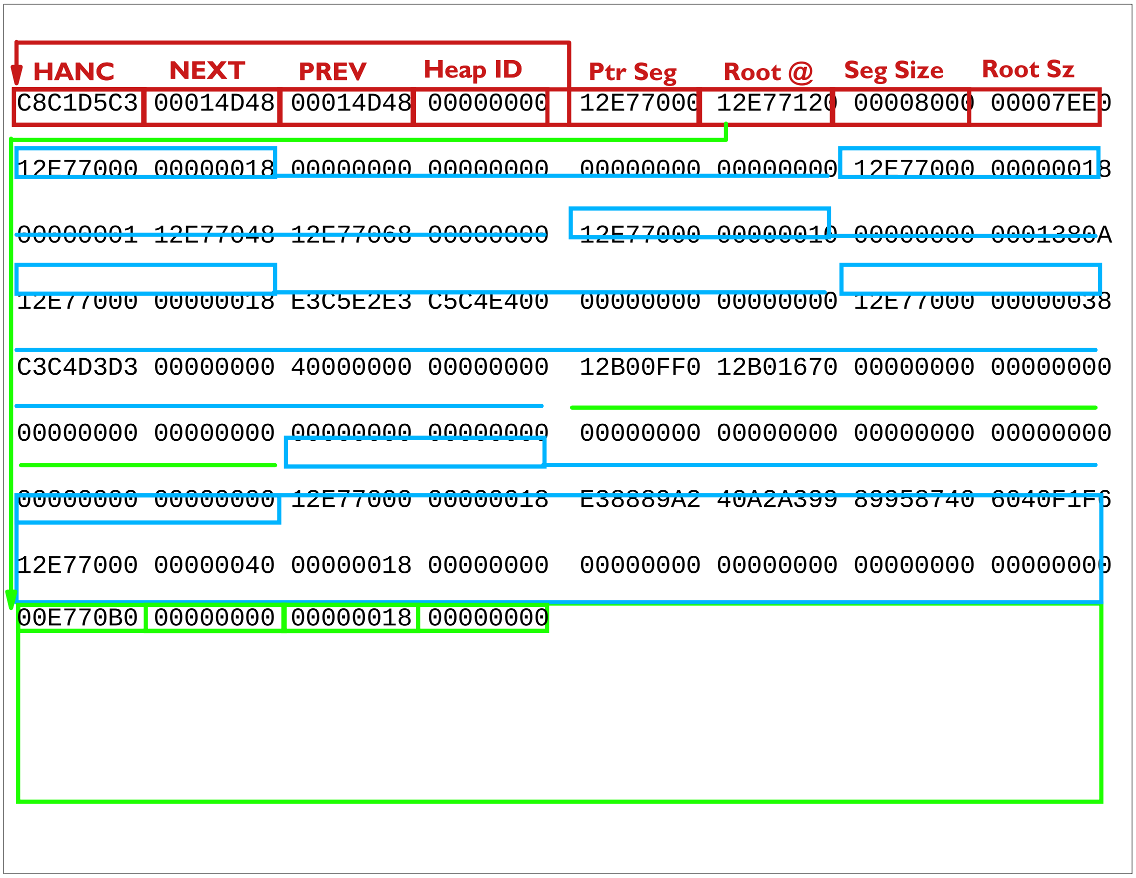
HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E770E0	00008000	00007F20
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000018	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000

HANC	NEXT	PREV	Heap ID	Ptr Seg	Root @	Seg Size	Root Sz
C8C1D5C3	00014D48	00014D48	00000000	12E77000	12E77120	00008000	00007EE0
12E77000	00000018	00000000	00000000	00000000	00000000	12E77000	00000018
00000001	12E77048	12E77068	00000000	12E77000	00000010	00000000	0001380A
12E77000	00000018	E3C5E2E3	C5C4E400	00000000	00000000	12E77000	00000038
C3C4D3D3	00000000	40000000	00000000	12B00FF0	12B01670	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	12E77000	00000018	E38889A2	40A2A399	89958740	6040F1F6
00E770B0	00000000	00000018	00000000	00000000	00000000	00000000	00000000


HANC NEXT PREV Heap ID Ptr Seg Root @ Seg Size Root Sz
C8C1D5C3 00014D48 00014D48 00000000 12E77000 12E77120 00008000 00007EE0
12E77000 00000018 00000000 00000000 00000000 00000000 12E77000 00000018
00000001 12E77048 12E77068 00000000 12E77000 00000010 00000000 0001380A
12E77000 00000018 E3C5E2E3 C5C4E400 00000000 00000000 12E77000 00000038
C3C4D3D3 00000000 40000000 00000000 12B00FF0 12B01670 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 12E77000 00000018 E38889A2 40A2A399 89958740 6040F1F6
00E770B0 00000000 00000018 00000000 00000000 00000000 00000000 00000000


HANC NEXT PREV Heap ID Ptr Seg Root @ Seg Size Root Sz
C8C1D5C3 00014D48 00014D48 00000000 12E77000 12E77120 00008000 00007EE0
12E77000 00000018 00000000 00000000 00000000 00000000 12E77000 00000018
00000001 12E77048 12E77068 00000000 12E77000 00000010 00000000 0001380A
12E77000 00000018 E3C5E2E3 C5C4E400 00000000 00000000 12E77000 00000038
C3C4D3D3 00000000 40000000 00000000 12B00FF0 12B01670 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 12E77000 00000018 E38889A2 40A2A399 89958740 6040F1F6
00E770B0 00000000 00000018 00000000 00000000 00000000 00000000 00000000

```
HANC      NEXT      PREV      Heap ID    Ptr Seg   Root @    Seg Size  Root Sz
C8C1D5C3  00014D48  00014D48  00000000   12E77000  12E77120  00008000  00007EE0

12E77000  00000018  00000000  00000000   00000000  00000000  12E77000  00000018

00000001  12E77048  12E77068  00000000   12E77000  00000010  00000000  0001380A

12E77000  00000018  E3C5E2E3  C5C4E400   00000000  00000000  12E77000  00000038

C3C4D3D3  00000000  40000000  00000000   12B00FF0  12B01670  00000000  00000000

00000000  00000000  00000000  00000000   00000000  00000000  00000000  00000000

00000000  00000000  12E77000  00000018   E38889A2  40A2A399  89958740  6040F1F6

00E770B0  00000000  00000018  00000000   00000000  00000000  00000000  00000000
00E770B0  00000000  00000018  00000000
```



HANC NEXT PREV Heap ID Ptr Seg Root @ Seg Size Root Sz
C8C1D5C3 00014D48 00014D48 00000000 12E77000 12E77120 00008000 00007EE0
12E77000 00000018 00000000 00000000 00000000 00000000 12E77000 00000018
00000001 12E77048 12E77068 00000000 12E77000 00000010 00000000 0001380A
12E77000 00000018 E3C5E2E3 C5C4E400 00000000 00000000 12E77000 00000038
C3C4D3D3 00000000 40000000 00000000 12B00FF0 12B01670 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 12E77000 00000018 E38889A2 40A2A399 89958740 6040F1F6
12E77000 00000040 00000018 00000000 00000000 00000000 00000000 00000000
00E770B0 00000000 00000018 00000000

Heap Errors in CEEDUMPs

- CEEDUMP - continued

```
+000000 12E77000  C8C1D5C3 00014D48 00014D48 00000000  12E77000 12E77120 00008000 00007EE0  |HANC..(...(......X...X........=.|
+000020 12E77020  12E77000 00000018 00000000 00000000  00000000 00000000 12E77000 00000018  |.X......................X......|
+000040 12E77040  00000001 12E77048 12E77068 00000000  12E77000 00000010 00000000 0001380A  |.....X...X.......X.............|
+000060 12E77060  12E77000 00000018 C7D60000 00000000  00000000 00000000 12E77000 00000038  |.X......GO..............X......|
+000080 12E77080  C3C4D3D3 00000000 40000000 00000000  12B00F78 12B01670 00000000 00000000  |CDLL.... .......................|
+0000A0 12E770A0  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |...............................|
+0000C0 12E770C0  00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6  |.........X......This string - 16|
+0000E0 12E770E0  12E77000 00000040 00000018 00000000  00000000 00000000 00000000 00000000  |.X..... .......................|
+000100 12E77100  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |...............................|
+000120 12E77120  00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000  |.X.............................|
+000140 12E77140  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  |...............................|
```

  - In this case after damage is caused we allocate x'38' bytes. Then allocate x'8' bytes.
  - Damaged now moved to 12E77120.
  - An allocated element is between the incorrect string and the heap damage.
  - Problem 2!!! (And harder to solve!!!)

- Explain problem 2. Algorithm is fast first. We don't always check immediately...

SYSTEM DUMPs

- VERBX LEDATA 'STACK' - sample

```
Stack Storage Control Blocks

    SMCB:  00015FF0
  +000000  EYE_CATCHER:SMCB  US_EYE_CATCHER:USTK     USFIRST:00022000
  +00000C  USLAST:00022000   USBOS:00022000    USEOS:00042000
  +000018  USNAB:F0F00000    USINITSZ:00020000       USINCRSZ:00020000
  +000024  USANYBELOW:80000000     USKEEPFREE:00000000     USPOOL:80000002
  +000030  USPREALLOC:00000001     US_BYTES_ALLOC:00000000
  +000038  US_CURR_ALLOC:00000000        US_GETMAINS:00000000
  +000040  US_FREEMAINS:00000000   US_OPLINK:00      LS_THIS_IS:LSTK
  +00004C  LSFIRST:00020000        LSLAST:00020000   LSBOS:00020000
  +000058  LSEOS:00022000    LSNAB:00021F40    LSINITSZ:00002000
  +000064  LSINCRSZ:00001000       LSANYBELOW:80000000
  +00006C  LSKEEPFREE:00000001     LSPOOL:80000001   LSPREALLOC:00000001
  +000078  LS_BYTES_ALLOC:00000000       LS_CURR_ALLOC:00000000
  +000080  LS_GETMAINS:00000000    LS_FREEMAINS:00000000   LS_OPLINK:00
```

  - Storage Management Control Block
    - Points to User and Library stacks

SYSTEM DUMPs

- VERBX LEDATA 'STACK' - sample

```
DSA backchain

   DSA:  000225C8
+000000  FLAGS:0000  MEMD:C64A   BKC:00020018      FWC:00022A28
+00000C  R14:8001806A      R15:92BC3268      R0:00000000
+000018  R1:00022654       R2:00020FA4       R3:00020FA4
+000024  R4:00017038       R5:12B62FBA       R6:12B61FBB
+000030  R7:12B60FBC       R8:12B5FFBD       R9:12B5EFBE
+00003C  R10:12B5DFBF      R11:92B5CFC0      R12:00015910
+000048  LWS:000163B0      NAB:00022650      PNAB:F840F0F0
+000064  RENT:F9F2C4F5     MODE:92B5FB18     RMR:4B4B4B4B

Contents of DSA  at location 000225C8:

+00000000  0000C64A 00020018 00022A28 8001806A  92BC3268 00000000 00022654 00020FA4 ...F.............k..............u.
+00000020  00020FA4 00017038 12B62FBA 12B61FBB  12B60FBC 12B5FFBD 12B5EFBE 12B5DFBF ....u.............................
+00000040  92B5CFC0 00015910 000163B0 00022650  F840F0F0 F0F2F2F3 F2F040F0 F0F0F2F2 .k..............&8 00022320 00022.
+00000060  F3F1F840 F9F2C4F5 C3F6F4C1 92B5FB18  4B4B4B4B 4B4B4B4B 4B4B4B4B 4B4B4B92 .318 92D5C64Ak..................k.
+00000080  000203D0 000210E7                                                       ........X                         .
```

- Next each DSA in backchain is formatted.
  - Somewhat like a CEEDUMP

SYSTEM DUMPs

- VERBX LEDATA 'HEAP' - sample

```
Heap Storage Control Blocks

     ENSM:  00014D30
   +0000A8  ENSM_ADDL_HEAPS:12E80C48

   User Heap Control Blocks

     HPCB:  00014D48
   +000000  EYE_CATCHER:HPCB  FIRST:12E77000    LAST:12E77000

     HANC:  12E77000
   +000000  EYE_CATCHER:HANC  NEXT:00014D48     PREV:00014D48
   +00000C  HEAPID:00000000   SEG_ADDR:12E77000       ROOT_ADDR:12E770E0
   +000018  SEG_LEN:00008000        ROOT_LEN:00007F20

   This is the last heap segment in the current heap.
```

  - Displays HPCB for user heap first
    - Heap Control Block
  - Then HANC header for each heap segment in user heap

SYSTEM DUMPs

- VERBX LEDATA 'HEAP' - sample

```
  Free Storage Tree for Heap Segment 12E77000

          Node      Node      Parent     Left      Right     Left      Right
 Depth  Address    Length     Node       Node      Node     Length    Length
     0  12E770E0  00007F20  00000000  00E770B0  00000000  00000018  00000000
*ERROR*  The left node address does not fall within the current heap segment
```

  - Next each heap segment is detailed.
    - Free tree first
    - Then allocated elements
  - Any errors with the control information are indicated with *ERROR*
    - In large dumps you can just search for *ERROR*

SYSTEM DUMPs

- VERBX LEDATA 'HEAP' - sample

```
  Map of Heap Segment 12E77000

To display entire segment: IP LIST 12E77000 LEN(X'00008000') ASID(X'0206')

12E77020: Allocated storage element, length=00000018. To display: IP LIST 12E77020 LEN(X'00000018') ASID(X'0206')
12E77028: 00000000 00000000 00000000 00000000                                      |................                |

12E77038: Allocated storage element, length=00000018. To display: IP LIST 12E77038 LEN(X'00000018') ASID(X'0206')
12E77040: 00000001 12E77048 12E77068 00000000                                      |.....X...X......                |

12E77050: Allocated storage element, length=00000010. To display: IP LIST 12E77050 LEN(X'00000010') ASID(X'0206')
12E77058: 00000000 0001380A                                                        |........                        |

12E77060: Allocated storage element, length=00000018. To display: IP LIST 12E77060 LEN(X'00000018') ASID(X'0206')
12E77068: E3C5E2E3 C5C4E400 00000000 00000000                                      |TESTEDU.........                |

12E77078: Allocated storage element, length=00000038. To display: IP LIST 12E77078 LEN(X'00000038') ASID(X'0206')
12E77080: C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000 |CDLL.... ..........0............|
```

  - The allocated elements are displayed.

SYSTEM DUMPs

- VERBX LEDATA 'HEAP' - sample

```
 12E77078: Allocated storage element, length=00000038. To display: IP LIST 12E77078 LEN(X'00000038') ASID(X'0206')
 12E77080: C3C4D3D3 00000000 40000000 00000000  12B00FF0 12B01670 00000000 00000000 |CDLL.... ..........0............|

 12E770B0: Allocated storage element, length=00000000. To display: IP LIST 12E770B0 LEN(X'00000008') ASID(X'0206')
 12E770B8: 00000000 00000000                                                       |........                        |
*ERROR*  The heap segment address in the allocated storage element header is not valid
WARNING  This storage element may be a free storage node not found during free storage tree validation
*ERROR*  The length of this storage element is zero
WARNING  Attempting to identify a resume location after encountering a storage element validation error

 12E770C8: Allocated storage element, length=00000018. To display: IP LIST 12E770C8 LEN(X'00000018') ASID(X'0206')
 12E770D0: E38889A2 40A2A399 89958740 6040F1F6                                     |This string - 16                |

 12E770E0: Free storage element, length=00007F20. To display: IP LIST 12E770E0 LEN(X'00007F20') ASID(X'0206')

 Summary of analysis for Heap Segment 12E77000:
    Amounts of identified storage:  Free:00007F20  Allocated:000000A8  Total:00007FC8
    Number of identified areas   :  Free:       1  Allocated:       7  Total:       8
    00000018 bytes of storage were not accounted for.
    Errors were found while processing this heap segment.
    This is the last heap segment in the current heap.
```

  - When errors are detected they are displayed
  - Much simpler to find damaged heap using LEDATA!
  - Problem 1 Solved!!!

HEAPCHK

- HEAPCHK runtime option (LE 1.8 and Up!)
  - Runtime debug tool to help diagnose a heap damage problem
  - In normal cases any heap damage may not be noticed until significant time has passed (problem 2)
  - The HEAPCHK runtime option forces all the heap segments to be validated on a regular basis.
  - Gets a dump closer to the real causer.
  - Generates a U4042 ABEND
    - Set SYSMDUMP DD card
    - Set SLIP
    - Set CEMT

HEAPCHK

- HEAPCHK runtime option ...
  - Syntax
    - HEAPCHK(ON|OFF,freq,delay)
      - ON - turns HEAPCHK on (performance dog)
      - OFF - normal processing
      - freq
        - Defaults to 1, indicates every call to a heap routine (get or free) validates the heap
        - Other values for less frequent checks
      - delay
        - Allows some number of calls to occur prior to 'freq' being used.

- Don't ever use a FREQ > 1 - If you are going to close the window close it all the way
-
- DELAY maybe. If you have a long running program...

HEAPCHK

- Job Log running with HEAPCHK

```
- ======================================================================
-                                    REGION        --- STEP TIMINGS ---
- STEPNAME PROCSTEP PGMNAME     CC      USED      CPU TIME  ELAPSED TIME
- CLG      COMPILE  CBCDRVR     00      176K   00:00:00.57   00:00:11.23
- CLG      LKED     HEWL        00      204K   00:00:00.17   00:00:01.38
IEA995I SYMPTOM DUMP OUTPUT
  USER COMPLETION CODE=4042 REASON CODE=00000000
 TIME=17.15.14  SEQ=04471  CPU=0000  ASID=01E0
 PSW AT TIME OF ERROR  478D1400   92B9B0E0  ILC 2  INTC 0D
   ACTIVE LOAD MODULE           ADDRESS=12B19AB8  OFFSET=00081628
   NAME=CEEPLPKA
   DATA AT PSW  12B9B0DA - 00181610  0A0D47F0  B0F01811
   GPR  0-3  84000000  84000FCA  00000000  00000000
   GPR  4-7  00000000  12B01648  00014558  000148B0
   GPR  8-11 12B9FEDD  12B9EEDE  00000001  12B9B028
   GPR 12-15 00015910  00022438  00000000  00000000
 END OF SYMPTOM DUMP
IEA993I SYSMDUMP TAKEN TO JMONTI.LEEDU3.SYSMDUMP
IEF450I JMONTI@C GO CLG - ABEND=S000 U4042 REASON=00000000
        TIME=17.15.43
- CLG      GO       PGM=*.DD U4042     276K   00:00:14.51   00:00:30.28
```

HEAPCHK

- Job Output running with HEAPCHK

```
CEE3701W Heap damage found by HEAPCHK Run-time option
CEE3707I Left  pointer is bad in the Free Tree at 12E770E0 in the Heap Segment
beginning at 12E77000
  12E770C0: 00000000 00000000 12E77000 00000018  E38889A2 40A2A399 89958740 6040F1F6
|.........X......This string - 16|
  12E770E0: 00E770B0 00000000 00000018 00000000  00000000 00000000 00000000 00000000
|.X..............................|
CEE3702S Program terminating due to Heap damage.
```

  - Note in this case CEE37xx messages are quite meaningful!
    - This was what they were designed for!
  - Full debug still needs to be done with the U4042 SYSMDUMP
  - Problem 2 Solved!

- Read the CEE3707I message!
-
- It doesn't get any better than this!

Summary

- Stack used for save areas and local variables
- Heap used for dynamic storage
- CEEDUMPs contain information on heap errors but they are difficult to find
- SYSTEM DUMPs using LEDATA 'STACK' and LEDATA 'HEAP' make debug much simpler
- Use HEAPCHK runtime option (LE 1.8)

Sources of Additional Information

- All LE documentation available on DISK 1 of OS/390 CD collection and on the LE website
- LE Debug Guide and Runtime Messages
- LE Programming Reference
- LE Programming Guide
- LE Customization
- LE Migration Guide
- LE Writing ILC Applications
- Web site
  - http://www.ibm.com/s390/le

- Books