```
; Filename: Heap.TEXT  (MACSBUG Heap ROUTINES)
;
;           Modification History
;
;           16-Sep-84   Added type mask to HD (format HD ['H'/'P'/'F'/'R'/'<resource
type>'])
;           23-Sep-84   Abort of heap print w/backspace
;           24-Sep-84   HT works w/heap mask
;           11-Oct-84   Changed heap display line format
;            8-Dec-84   Made BlockMove NOT use saved routine address
;

            .IF         fullSized               ; none here

badHeap
            MOVEQ       #MBadHeap-MText,D0
            BSR         MEight

            MOVE.L      A2,D0                   ; the two block addresses
            BSR         Pr8                     ; 8 then spaces
            MOVE.L      A1,D0
            BSR         PNT8HX
            BSR         WriteLine               ; dump msg

            MOVEQ       #-1,D0                  ; heap is bad
            RTS


CheckHeap
            MOVE.L      bugHeap,A4              ; point to current heap

            LEA         HeapData(A4),A1         ; ptr to first block
            MOVE.L      BufPtr,A0               ; point to top of usable memory
            SUB         A2,A2                   ; previous block ptr

ContHeap
            MOVE.L      tagBC(A1),D0            ; get size/tag 1
            AND.L       MaskBC,D0
            CMP.L       D0,A0                   ; see if past memory top
            BLT         badHeap
            LSR         #1,D0                   ; see if odd
            BCS         badHeap

; case on type
            MOVE.L      tagBC(A1),D0            ; get size/tag 1
            AND.L       #TagMask,D0
            BEQ.S       FreeBlock
            BMI.S       RelBlock

; fixed type
            BRA.S       NextChk

; free block
FreeBlock
            BRA.S       NextChk

; Rel block
RelBlock
            MOVE.L      Handle(A1),D0           ; handle
            LSR         #1,D0                   ; see if odd
            BCS         badHeap
```

```
            MOVE.L      Handle(A1),D0           ; handle
            MOVE.L      0(A4,D0.L),D0           ; master pointer
            AND.L       MaskBC,D0               ; make sure handle is right
            CMP.L       D0,A0                   ; see if past memory top
            BLT         badHeap
            SUBQ.L      #8,D0                   ; this should point back to block ptr
            CMP.L       D0,A1
            BNE         badHeap

NextChk
            MOVE.L      A1,A2                   ; save previous

            MOVE.L      tagBC(A1),D1            ; get size/tag
            AND.L       MaskBC,D1               ; block size in D0
            ADD.L       D1,A1                   ; calc ptr to next block

            CMP.L       D1,A0                   ; see if past memory top
            BLT         badHeap

            CMP.L       bkLim(A4),A1            ; see if at end
            BNE         ContHeap

            MOVEQ       #0,D0                   ; heap is ok
            RTS

;=============================================================================
;
; Heap scrambler
;
;=============================================================================
;
Scram
            MOVE.L      bugHeap,A4              ; point to heap

            MOVE.L      A4,D7                   ; save in D7

            CLR.L       AllocPtr(A4)            ; Hi Larry

            LEA         HeapData(A4),A0         ; ptr to first block

Continue
            MOVE.L      tagBC(A0),D1            ; get size/tag 1

            MOVE.L      D1,D3                   ; get size 1 isolated
            AND.L       MaskBC,D1

            MOVE.L      A0,A1                   ; get ptr to second block
            ADD.L       D1,A1                   ; calc ptr to second block

            MOVE.L      D7,A4                   ; Get limit
            CMP.L       bkLim(A4),A1            ; see if at end
            BEQ         Adioscram

            MOVE.L      A0,A4                   ; save leftmost pointer

            MOVE.L      tagBC(A1),D2            ; get size/tag 1

            MOVE.L      D2,D4                   ; get size 2 isolated
            AND.L       MaskBC,D2

            AND.L       #TagMask,D3             ; get tag 1 isolated
            BEQ         LeftFree
```

```
            BPL         NextOne                 ; get out if non relocatable

            AND.L       #TagMask,D4             ; get tag 2 isolated
            BEQ         RightFree
            BPL         NextOne                 ; get out if non relocatable

;-------------------------------------------------------------------------
; Both blocks are relocatable

            MOVE.L      Handle(A0),A3           ; see if 1st is locked
            TST.L       0(A3,D7.L)              ; see if master is negative
            BMI         NextOne

            MOVE.L      Handle(A1),A3           ; see if 2nd is locked
            TST.L       0(A3,D7.L)              ; see if master is negative
            BMI         NextOne


;-------------------------------------------------------------------------
; swap two blocks (A0,D1) and (A1,D2) from the right

            MOVE.L      D1,D5                   ; Calc total length in D4 to move
            ADD.L       D2,D5                   ; add right length to left length
            SUBQ.L      #2,D5                   ; decrement by 1 word

            MOVE.L      D2,D4                   ; Calc # rotates
            ASR.L       #1,D4                   ; as word count(left length)
            SUBQ.L      #1,D4                   ; pre decrement for DBF

; Set up for rotate right as if right block is smaller
            MOVE.L      A0,A1                   ; A1 = A0 + 2
            ADDQ.L      #2,A1
            MOVE.L      A0,A2                   ; A2 points to left one
            MOVE.L      A0,A3                   ; A3 points to right one
            ADD.L       D5,A3                   ; point to next to last word

            CMP.L       D1,D2                   ; see which assumption is true
            BLE.S       RotLoop                 ; right is bigger

            EXG         A0,A1                   ; Switch all the pointers for other
            EXG         A2,A3                   ; direction

            MOVE.L      D1,D4                   ; Calc # rotates(right length)
            ASR.L       #1,D4                   ; as word count
            SUBQ.L      #1,D4                   ; pre decrement for DBF


RotLoop
            MOVE        (A3),D6                 ; save word
            MOVE.L      D5,D0                   ; get combined length - 2
            BSR         myMove
            MOVE        D6,(A2)                 ; put back saved word

            DBRA        D4,RotLoop              ; continue on for right length

 ; now update the master pointers to reflect new locations
FixupMasters
            BSR         UpdateMaster            ; A4 points to REL block

            MOVE.L      TagBC(A4),D0            ; get next block
            AND.L       MaskBC,D0               ; isolate the size
            ADD.L       D0,A4                   ; get ptr to second block
```

```
            BSR         UpdateMaster            ; A4 points to REL block

            MOVE.L      A4,A0                   ; get second ptr

;---------------------------------------------------------------------------

CheckNext
            MOVE.L      bugHeap,A4              ; point to application heap
            CMP.L       bkLim(A4),A0            ; see if at end
            BNE         Continue

Adioscram
            RTS

;---------------------------------------------------------------------------
;
NextOne
            AND.L       MaskBC,D1               ; block size in D0
            ADD.L       D1,A0                   ; calc ptr to next block
            BRA.S       CheckNext



;---------------------------------------------------------------------------
; Left block is free

LeftFree
            AND.L       #TagMask,D4             ; Make sure right is relocatabale
            BPL         NextOne                 ; if both free skip

            MOVE.L      Handle(A1),A3           ; see if 2nd is locked
            TST.L       0(A3,D7.L)              ; see if master is locked
            BMI         NextOne


; Swap the two(Only have to swap the relocatable lengths worth)
            CMP.L       #12,D2                  ; make sure relocatable is big enough
            BLT         NextOne

; A0 now points to end of free block which is now on left
; A1 now points to end of relocatable which is now on right

            MOVEM.L     tagBC(A0),D3-D5         ; save the free block stuff
                                                ; D3-tagBC, D4-Next, D5-Prev
            EXG         A0,A1
            MOVE.L      D2,D0                   ; move rel length bytes
            BSR         myMove

; A4 points to left
            BSR         UpdateMaster            ; A4 points to REL block

            MOVE.L      TagBC(A4),D0            ; get next block
            AND.L       MaskBC,D0               ; isolate the size
            MOVE.L      A4,A3                   ;
            ADD.L       D0,A3                   ; get ptr to second block

; A3 now points to new free block
FixFree
            MOVEM.L     D3-D5,tagBC(A3)         ; restore the free block stuff


            MOVE.L      A4,A0                   ; point to a valid block
```

```
            MOVE.L      tagBC(A0),D1            ; get size
            BRA         NextOne

;-------------------------------------------------------------------------
; Right block is free

RightFree
            CMP.L       #12,D1                  ; make sure relocatable is big enough
            BLT         NextOne

            MOVE.L      Handle(A0),A3           ; see if 1st is locked
            TST.L       0(A3,D7.L)              ; see if master is locked
            BMI         NextOne

; A0 now points to end of relocatable which is now on left
; A1 now points to end of free block which is now on right

            MOVEM.L     tagBC(A1),D3-D5         ; save the free block stuff
                                                ; D3-tagBC, D4-Next, D5-Prev
            MOVE.L      A0,A1                   ; set up block move
            ADD.L       D2,A1                   ; dest is left ptr + free size
            MOVE.L      A4,A3                   ; save A4 in A3 for free fix
            MOVE.L      A1,A4                   ; save A4 for fix below

            MOVE.L      D1,D0                   ; move rel length bytes
            BSR         myMove

            BSR         UpdateMaster            ; A4 points to REL block

; A4 now points to new free block
            BRA.S       FixFree


;=========================================================================
;
; UpdateMaster routine
;       Entry:
;           A4 is block pointer
;           A0 is trashed
;=========================================================================

UpdateMaster
            MOVE.L      Handle(A4),A0           ; update master pointer
            ADD.L       D7,A0
            MOVE.B      (A0),D0                 ; save upper byte
            MOVE.L      A4,(A0)                 ; save new master
            ADDQ.L      #8,(A0)                 ; point to real data
            MOVE.b      D0,(A0)                 ; put back upper byte
            RTS

myMove
            MOVEM.L     D0/A0-A2,-(SP)          ; save regs
            MOVE.L      $28,-(SP)               ; save debugger A-trap handler
            MOVE.L      SAVEA,$28               ; restore normal A-trap stuff
            _BlockMove                          ; and do the move
            MOVE.L      (SP)+,$28               ; restore debugger A-trap handler
            MOVEM.L     (SP)+,D0/A0-A2          ; restore regs
            RTS                                 ; and return
```

```
;---------------------------------------------------------------
relCount    .EQU        0
ptrCount    .EQU        relCount+2
purgeCount  .EQU        ptrCount+2
purgeSpace  .EQU        purgeCount+2
lockCount   .EQU        purgeSpace+4
freeBytes   .EQU        lockCount+2
stkSpace    .EQU        freeBytes+4

;-------------------------------------------------------------------------------
--
; Routine Name          MaskOK
;
; Registers             D0.L (input)            ; value to check against mask (for
MaskOK)
;                       D0.B (input)            ; (for Mask2OK)
;
; Function              If no mask, or mask matchs D0, or mask is for handle & D0 has
;                       '<rsrc>' or '    ', or mask is for 'R' (all rsrcs) and D0 has
'<rsrc>',
;                       then return, else set printEntry to #1.
;
;-------------------------------------------------------------------------------
--

Mask2OK
            AND.L       #$FF,D0                 ; mask off top three bytes, entry for D0
= 'P'/'F'
                                                ; Mask2OK not called with 'H'
MaskOK
            MOVE.L      D1,-(SP)                ; save D1 on stack
            MOVE.L      heapMask,D1             ; get the heap mask
            TST.L       D1                      ; any mask?
            BEQ.S       MaskExit                ; no, all test values are OK
            CMP.L       D0,D1                   ; is there a match?
            BEQ.S       MaskExit                ; yes, all okay

; mask <> nil, mask <> test value, but if test value is a resource/handle and mask is a
handle, then
; everythings AOKAY

            SWAP        D0                      ; exch low/high words
            TST.W       D0                      ; is it a resource (assume high word <>
0 for true rsrc)
            BNE.S       @0                      ; yes, check for handle/all resource
masks

            TST.L       D0                      ; nil value (rsrc value for no
resource)?
            BNE.S       BadMask                 ; yes, check for handle mask

; it was a resource of some type, or a relocatable non-resource block, so check for
handle mask

@0          CMP.L       #'H',D1                 ; is the mask for a handle?
            BEQ.S       MaskExit                ; yes, exit normally

; test for generic resource mask (print all resources)

            CMP.L       #'R',D1                 ; is the mask for all resources?
            BNE.S       BadMask                 ; no, bail out
            TST.W       D0                      ; do we have a resource?
```

```
            BNE.S       MaskExit                ; yes, still ok.

BadMask
            ADDQ.W      #1,printEntry           ; set printEntry <> 0
            BRA.S       Mask2Exit               ; and return

MaskExit
            MOVE.L      A2,D0                   ; get the size of this entry
            ADD.L       D0,maskTotal            ; bump total size of masked entries
            ADDQ.W      #1,maskCount            ; and increment count of masked entries

Mask2Exit
            MOVE.L      (SP)+,D1                ; restore D1
            RTS                                 ; and return

;-------------------------------------------------------------------------------
--
; Routine Name          DoPrint
;
; Function              Do test of D3 for printing, set CC's correctly.  If not
printing,
;                       but masking, always return EQ.
;-------------------------------------------------------------------------------
--

DoPrint
            TST.L       heapMask                ; is there a heap mask
            BNE.S       @0                      ; yes, EQ exit

            TST.L       D3                      ; test D3 and exit
            RTS

@0          CLR.L       Temp                    ; set EQ CC
            RTS

;-------------------------------------------------------------------------------
--
; Routine Name          PrintTotal
;
; Function              Print out a total count of heap items, masking with D0 (see
below).
;-------------------------------------------------------------------------------
--

PrintTotal
            MOVEQ       #1,D3                   ; no printing
            BRA.S       printAll

;-------------------------------------------------------------------------------
--
; Routine Name          PrintHeap
;
; Registers             D0.L (input)            ; heap mask
;
; Function              Print out the heap specified by bugHeap, masking with D0.  If D0
;                       = 'H', 'P', 'F', display only those types of blocks.  If D0 >
'H',
;                       assume a resource type mask (eg. 'CODE') and display only
resource
;                       blocks of that type.
;-------------------------------------------------------------------------------
--
```

```
PrintHeap
            MOVEQ       #0,D3                   ; do printing

printAll
            SUB         #stkSpace,SP            ; make room for my locals on stack
            MOVE.L      bugHeap,A4              ; point to heap

            CLR.W       relCount(SP)            ; init all locals
            CLR.W       ptrCount(SP)
            CLR.W       purgeCount(SP)
            CLR.L       purgeSpace(SP)
            CLR.W       lockCount(SP)
            CLR.L       freeBytes(SP)

            CLR.W       maskCount
            CLR.L       maskTotal
            MOVE.L      D0,heapMask             ; save heap mask value

            BSR.S       DoPrint                 ; Print?
            BNE.S       @0                      ; no printing

            BSR         FixBuf
            BSR         CRLF                    ; print blank line

            MOVE.L      A4,D0                   ; print zone
            BSR.S       Pr8
            BSR         WriteLine               ;

            BSR         CRLF
@0
            LEA         HeapData(A4),A1         ; ptr to first block

ContPrint
            CLR.W       printEntry              ; assume we'll print this next heap
entry

            MOVE.L      tagBC(A1),D0            ; get size/tag 1
            AND.L       MaskBC,D0
            MOVE.L      D0,A2                   ; save size in A2

            BSR.S       DoPrint                 ; Print?
            BNE.S       @0

            BSR         FixBuf                  ; start a new line

            MOVE.L      A1,D0                   ; print address
            BSR.S       Pr8

            MOVE.L      tagBC(A1),D0            ; get size/tag 1
            AND.L       #TagMask,D0

; case on type of heap entry

@0          MOVE.L      tagBC(A1),D0            ; get size/tag 1
            AND.L       #TagMask,D0
            BEQ.S       FreePrint
            BMI.S       RelPrint

; block is non-relocatable ('P' for mask)

            ADDQ        #1,ptrCount(SP)         ; update ptr count
```

```
            MOVEQ       #'P',D0
            BSR.S       TypeMask                ; print type, size, check mask

            MOVEQ       #32,D0
            BSR         Tab1t                   ; set up for '*' in column 32

            MOVE.B      #$2A,(A6)+              ; asterisk on ptrs
            BRA         nextPrint

; Printing utilities

Pr4
            BSR         Pnt4HX
            BRA.S       PrSpace
Pr8
            BSR         Pnt8HX
PrSpace
            MOVE.B      #$20,(A6)+
            RTS

TypeMask
            BSR.S       JustType                ; print out type & size
            BSR         Mask20K                 ; check on mask
            RTS

JustType
            MOVE.L      D0,-(SP)                ; save reg
            MOVE.B      D0,(A6)+                ; print out 'F'/'P'/'H'
            MOVE.B      #' ',(A6)+              ; print space
            MOVE.L      A2,D0                   ; get size
            BSR.S       Pr8                     ; print it out
            MOVE.L      (SP)+,D0                ; restore reg
            RTS                                 ; and return

; free block ('F' for mask)

FreePrint
            MOVE.L      A2,D0                   ; bump up the size
            ADD.L       D0,freeBytes(SP)

            MOVEQ       #'F',D0
            BSR.S       TypeMask                ; print type, size, check on mask

            BRA.S       nextPrint

; Relocatable block ('H' for mask)

RelPrint
            ADDQ        #1,relCount(SP)         ; update rel count

            MOVEQ       #'H',D0
            BSR.S       JustType                ; print type, size

            MOVE.L      Handle(A1),D0           ; master pointer offset
            MOVE.L      0(A4,D0.L),D1           ; master pointer value
            SPL         D4
            BPL.S       @0
            ADDQ        #1,lockCount(SP)        ; update lock count
            BRA.S       @1                      ; skip purge
@0
            BTST        #30,D1                  ; purgeable?
```

```
            BEQ.S       @1
            ADDQ        #1,purgeCount(SP)       ; update purge count

            MOVE.L      A2,D2
            ADD.L       D2,purgeSpace(SP)
@1
            BSR.S       DoPrint                 ; Print?
            BNE.S       NextPrint

; print out <master ptr high nybble>SP<master ptr loc>SPSP<SP|*>SPSP<refnum>SPSP<id
num>SPSP<type>

            BSR         GetResStuff             ; D5 the resfile refnum, D6 the ID, D7
the type

            MOVE.L      D0,-(SP)                ; save D0 on stack
            MOVE.L      D1,D0                   ; get master ptr value
            ROL.L       #4,D0                   ; put nibble in low order
            BSR         PutHex                  ; print nibble
            MOVE.B      #$20,(A6)+
            MOVE.L      (SP)+,D0                ; restore D0

            ADD.L       A4,D0                   ; D0 = address of master ptr

            BSR.S       Pr8                     ; print location of master ptr

            MOVEQ       #' ',D0
            TST.B       D4                      ; asterisk the locked ones
            BNE.S       @2
            ADD         #$A,D0
@2
            MOVE.B      D0,(A6)+                ; push asterisk or space


            MOVE.L      D7,D0                   ; get the resource type
            BSR         MaskOK                  ; if no match, doesn't return

            TST.L       D7
            BEQ.S       NextPrint               ; only a handle, not a resource file

            BSR.S       PrSpace                 ; print 2 spaces
            MOVE        D5,D0                   ; dump resfile
            BSR         Pnt2Hx
            BSR.S       PrSpace
            MOVE        D6,D0                   ; dump id
            BSR         Pr4

            MOVEQ       #3,D1                   ; get ready to print 4 ascii chars
            MOVE.L      D7,D0                   ; set up for Bin2Char call

@3          ROL.L       #8,D0                   ; shuffle high byte down
            BSR         Bin2Char                ; print the character
            DBRA        D1,@3                   ; and loop

nextPrint
            TST         D3
            BNE.S       @0                      ; don't print anything, keep going
            TST.W       printEntry              ; should we print this entry (set in
MaskOK)
            BNE.S       @0                      ; no, don't print this stuff out
            BSR         WriteLine
            TST.B       AbortPrint              ; did user abort printout?
```

```
            BNE.S       HeapSummary             ; yes, print summary and exit

@0          BSR         FixBuf                  ; for no heap printing, but heap mask,
need to set
            ADD.L       A2,A1                   ; calc ptr to next block
            CMP.L       bkLim(A4),A1            ; see if at end
            BNE         ContPrint

; print a summary of the heap

HeapSummary
            BSR         FixBuf                  ; flush current print buffer
            BSR         WriteLine               ; print blank line, set up for summary
            TST.L       heapMask                ; was there a heap mask
            BEQ.S       @1                      ; no, print full summary

            MOVEQ       #MHeapM-MText,D0        ; print partial (masked) summary
            BSR         MEight

            MOVEQ       #0,D0                   ; don't really need this, right?
            MOVE.W      maskCount,D0
            BSR         Pr4
            MOVE.L      maskTotal,D0
            BRA.S       @2

@1
            MOVEQ       #MHeapT-MText,D0
            BSR         MEight

            MOVEQ       #0,D0
            MOVE        relCount(SP),D0         ; print rel count
            BSR         Pr4
            MOVEQ       #0,D0
            MOVE        lockCount(SP),D0        ; print lock count
            BSR         Pr4
            MOVEQ       #0,D0
            MOVE        purgeCount(SP),D0       ; print purge count
            BSR         Pr4
            MOVE.L      purgeSpace(SP),D0       ; print purge space
            BSR.S       Pr8
            BSR         PrSpace
            MOVEQ       #0,D0
            MOVE        ptrCount(SP),D0         ; print ptr count
            BSR         Pr4
            MOVE.L      freeBytes(SP),D0        ; print free bytes
@2          BSR.S       Pr8

            BSR         WriteLine

            ADD         #stkSpace,SP

            RTS

; Given a handle's block ptr in A1, this guy searches through all maps
; to find a corresponding resource type and ID

GetResStuff
            MOVEM.L     D0-D4/A0-A2,-(SP)
            MOVEQ       #0,D7                   ; assume failure

            MOVE.L      MaskBC,D1               ; nice number ($FFFFFF)
            MOVE.L      Handle(A1),D2           ; handle + offset
```

```
            ADD.L       A4,D2                   ; add heap zone
            AND.L       D1,D2                   ; D2 contains "clean" handle

            MOVE.L      TopMapHndl,A2           ; get top handle
@0
            MOVE.L      A2,D0                   ; are we done?
            BEQ.S       @5

            MOVE.L      (A2),A0                 ; to block beginning
            SUBQ        #8,A0
            MOVE.L      (A0),D7                 ; get the block size
            AND.L       D1,D7
            SUB         #$20,D7                 ; skip header
            ADD         #$20,A0
@1
            MOVE.L      (A0),D0                 ; get long from the map
            AND.L       D1,D0
            CMP.L       D0,D2                   ; right handle?
            BEQ.S       @2                      ; found it

            ADDQ        #2,A0                   ; try next
            SUBQ        #2,D7
            BGT.S       @1                      ; keep trying

            MOVE.L      (A2),A2
            MOVE.L      $10(A2),A2              ; get the next map
            BRA.S       @0

; Found the handle
@2
            SUBQ        #8,A0                   ; point to ID
            MOVE        (A0),D6                 ; and save it
            MOVE.L      (A2),A1                 ; get map pointer
            MOVE        $14(A1),D5              ; return resfile
            ADD         #$1C,A1                 ; add header length
            SUB.L       A1,A0                   ; A0 is offset

            MOVE        (A1)+,D0                ; get # types
            MOVE        #-30000,D4              ; set difference
            SUB.L       A2,A2
@3
            MOVE        6(A1),D1                ; get offset
            SUB         A0,D1                   ; and calc diff
            BGT.S       @4
            CMP         D1,D4                   ; is this less than other
            BGT.S       @4
            MOVE        D1,D4
            MOVE.L      A1,A2                   ; remember type
@4
            ADDQ        #8,A1                   ; next entry
            SUBQ        #1,D0
            BPL.S       @3

; A2 contains types entry
            MOVE.L      (A2),D7                 ; remember type

@5
            MOVEM.L     (SP)+,D0-D4/A0-A2

            RTS
```

```
.ENDC
```