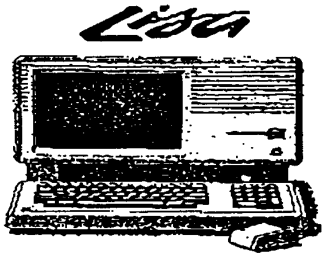

Date: February 26, 1982

To: Lisa Software, NPR, and Technical Support

cc: Wayne Rosing, Pat Marriott, Andy Hertzfeld

DTC 4/2001
Member of
Macintosh team

From: Larry Tesler LT

Subject: Heap Usage Standards and Facilities

This memo discusses the problem of heap fragmentation from a theoretical and practical standpoint. It also proposes more judicious use of non-relocatable blocks in Lisa software. Finally, it summarizes recent changes to the interface of the standard storage allocator (Hz and unit Storage). Somewhat more complete documentation of UnitHz appears in a companion memo.

I have based these proposals and plans on conversations with many, but not all, members of the Applications Group, plus Bud Tribble, Bruce Daniels, and Rich Page. *

FRAGMENTATION

Lest anyone forgets, fragmentation of the heap is even more disastrous in Lisa than in most computer systems.

True, when a heap fragments, its data segment usually can be expanded, but a heap that is both fragmented and expanded leads to serious performance degradation.

The reason that fragmentation is so undesirable on Lisa is that the user is likely to keep a process alive—and very actively allocating and freeing storage—for eight or more hours at a stretch. It would be unfriendly to announce that memory is exhausted, or to slow to a snail's pace, just because most of the heap is occupied by free blocks not quite large enough to fulfill the most recent allocation request.

Fragmentation is an unavoidable problem when non-relocatable blocks of varying sizes are allocated and deallocated in a heap. Theoretical studies have proven that, regardless of allocation strategy, a heap will always fragment. The time it takes is a function of the variance in block sizes. Clever strategies can only defer the unavoidable.

* DTC 4/2001
Bud T - member of 1st Macintosh development team, wrote
core Mac OS library.
Bruce D - Head of Lisa software.
Rich P - Early Lisa team member, wrote Lisa Monitor dev environment.

The theoretical studies have simply validated common experience. Some anecdotes:

> The Burroughs 5000, where buddy-system allocation was a hallmark of the ahead-of-its-time Master Control Program, fragmented storage every few hours and had to be rebooted; not good for a multiprogramming number-cruncher. Burroughs later adopted relocatable storage.
>
> Smalltalk-72 used to die after a few hours, with thousands of words of free space but no block large enough for a disk buffer. Starting with Smalltalk-76, we adopted relocatable storage.
>
> The first implementation of Bravo (forerunner to our Word Processor) would run only a few minutes before its heap became a wasteland of undersized free blocks. The first version they actually released used relocatable storage. At first, they used a complicated system to avoid double-indirection. After a few years of painful experience with it, Tom * switched to an allocator of the same flavor as Hz.

Even in our own group, the folly of using non-relocatable blocks in the heap has been demonstrated by the two applications that have received the most extensive use so far: the Graphics Editor and the Mouse Editor. ** They both die a premature death with memory looking something like this (artificial example):

```
 10 word used block
100 word free block
 10 word used block
110 word free block
 10 word used block
120 word free block
 10 word used block
130 word free block
 10 word used block
100 word free block
 10 word used block
110 word free block
 10 word used block
120 word free block
 10 word used block
130 word free block
```

** DTC 4/2001
"Graphic Editor" ≡ LisaDraw
"Mouse Editor" ≡ LisaWrite
(tho this name was also used for the Lisa Workshop's mouse-based editor which I believe Bruce Daniels worked on).

and then a request for a 140-word block comes along. There are 920 free words in a 1000-word heap, but no room for a 140-word block!

OVERHEAD

The effective overhead for each of the eight 10-word blocks in the above heap is 920/8 or 115 words. So much for the myth that relocatable blocks have a greater overhead than non-relocatable blocks.

* DTC 4/2001
Tom ≡ Tom Malloy: Worked on the Xerox Bravo word processor, then worked on the LisaWrite word processor.

Relocatable blocks do have overhead. In addition to the two-word header and the two-word master pointer, every double-indirect reference requires at least one extra instruction, and the heap has a lot of unused master pointers. But from a system performance standpoint, this price is worth it to avoid fragmentation.

DANGERS OF DOUBLE-INDIRECTION

It is true that double-indirection has its risks. It is always safe to write:

```
    x := h^^.f;
```

but one must take care with constructs such as:

```
    p := h^;
    ...
    x := p^.f;
```

or

```
    with h^^ do
        begin
        ...
        x := p^.f;
        ...;
        end;
```

because a procedure call between the dereference and the access might cause a heap compaction.

Some people get around this by having two heaps: one for objects that are infrequently allocated, but accessed by many statements in their program, and another for objects that are frequently allocated, but accessed by few statements in their program. They then can use "with" for the first type even when calling procedures that may allocate the second type. For example, LisaGraf allocates and deallocates regions and buffers like crazy, but your program never accesses them, so you can give LisaGraf a separate heap and call its procedures almost with abandon.

An apparent disadvantage of this trick is that unused space in one heap is not available to satisfy requests in the other heap. Still, it might work out well if both heaps contain nothing but relocatable blocks, because each data segment can then grow AND SHRINK dynamically.

Another disadvantage is that you can map only a few data segments at a time into your address space. If you have one for LisaGraf, one for you, one shared segment for the Window Manager and Scrap, and three for the Data Base Primitives, then that leaves little for whatever else comes up. We had better watch our data segment usage carefully.

OTHER WAYS TO AVOID FRAGMENTATION

The Data Base Primitives use non-relocatable blocks to reduce overhead, and a special-purpose technique to recover from the inevitable fragmentation. The heap is cleared out and regenerated from redundant information stored elsewhere. Most programs do not have the information to do this.

Sometimes, a program can do its own relocation without using the standard allocator, because it knows where every pointer is. But this is rarely feasible, because one often exhausts memory when some uplevel procedure has a local variable pointing at a heap block, and you'll never get that pointer adjusted.

Sometimes, an application allocates and deallocates objects that are all the same size. In that case, it is possible that an array or a dedicated heap can be an efficient and unfragmentable place to store the objects.

STANDARDS

I propose that everyone find their uses of non-relocatable blocks and change over to relocatable blocks instead, unless:

(1) The blocks are in a heap that contains only non-relocatable blocks, preferably all of the same size, or,

(2) You have a proven recovery technique like the data base primitives, or,

(3) You can prove that fragmentation won't be a problem (the heap only exists for a few minutes and does a known job, or whatever).

You are welcome to consult me about problems that come up in this (or any) effort.

FACILITIES

There have been at* least four relocatable allocators floating around. In Monitor 7, Tom released a new version that consolidates what I consider the most important features. It is segmented so that procedures used only by the Word Processor are not swapped in along with the kernel procedures. It has under 2K bytes of resident code.

Briefly, the new allocator has the following advantages:

(1) A more uniform interface than the old Hz, more like Unit Storage;

(2) Less error-prone than unit Storage, because hz is an explicit, rather than an implicit, argument.

(3) A function to return the heap that a handle references:

```
    hz := HzFromfH(h)
```

See the accompanying memo, "The New Standard Storage Manager Interface", for details.

All former Hz and Storage procedures are still supported. However, I urge people to change to the new ones when they have a chance so that we can reduce the interfaces eventually, and so that software maintenance will be easier.

** Bill is planning to make the following addition to allow a region to be allocated in an explicitly-specified heap:

```
RgnAllocate(hz: THz): RgnHandle
```

The procedure:

```
NewRgn: RgnHandle
```

will still be available, and will use "theHeap". LisaGraf will also use "theHeap" for temporary allocation. Some day, we'll probably rename it "grafHeap".

In a forthcoming release of UnitHz, it will not be possible to mix relocatable and non-relocatable blocks in the same heap. I strongly suggest that everyone who still has non-relocatable blocks put them in a different heap before that release so that conversion to the new UnitHz will be simple.

* Tom ≡ Tom Malloy: LisaWrite, Lisa heap memory manager programmer.

** Bill ≡ Bill Atkinson: Lisa QuickDraw graphics library programmer.

"Monitor" refers to the Lisa's 1st development environment that was very UCSD p-system-like (Rich Page worked on this).

NONRELOCATABLE GRAFPORTS AND EVENTS

At the moment, grafPorts and eventRecords are nonrelocatable and have been causing fragmentation of heaps. Bruce, Bill, and I have analyzed * many alternatives and have decided upon the following solution at least for First Release:

> Event records in the Event Manager will become relocatable. This will affect only the Event Manager and the Alert Manager.
>
> Grafports will not become relocatable. The code changes both in LisaGraf and in applications would hurt schedule and code size too much for First Release. Instead, the Window Manager will limit the number of windows to 20 (plus the dialog box and alert box), and store their nonrelocatable grafPorts in an array that cannot move or grow. There cannot be more than 20 of them in the Tray anyway. (Second release there are several different ways to lift this restriction if necessary.) Applications can still have their own non-window grafPorts in their own global variables as they do now, but they should avoid allocating and deallocating them dynamically on the heap.

FUTURE ENHANCEMENTS

The following allocator enhancements are expected before first release:

(1) ChangeSizeH will grow/shrink in place instead of copying, when possible.

(2) A CheckHeap procedure will scan the data structure and report clobbered free-lists and headers. This can be called during debugging to help isolate code doing wild stores.

(3) A testing tool will be provided to help catch invalid dereferenced handles.

(4) Relocatable and non-relocatable blocks will be forbidden in the same heap.

These changes would not affect the standard interface (except to add CheckHeap).

After first release (or sooner, if forced by performance problems), we might adapt Bud Tribble's assembly-language allocator to increase ** speed. We are not adopting it now because it is slightly incompatible.

* DTC 4/2001
Bruce ≡ Bruce Daniels
Bill ≡ Bill Atkinson
I ≡ Larry Tesler

** Bud Tribble:
Mac OS programmer

o The End o