The Icon Memory Monitoring System

Gregg M. Townsend

Department of Computer Science, The University of Arizona

Introduction

The Icon memory monitoring system (''MemMon'') provides tools for displaying Icon's allocated data regions [1]. It consists of instrumentation that produces *allocation history files* (Appendix C) and visualization programs that convert allocation history files to displays that show the sizes, types, and locations of strings as they are allocated. The garbage collection process is shown in detail.

There are several visualization programs; most of them are specific to the University of Arizona environment. Appendix A describes **mmps**, a program for producing displays that can be printed on any PostScript printer.

An allocation history file is produced by setting the environment variable MEMMON to the name of the desired file. No change in the Icon program is necessary and the production of an allocation history file does not change program behavior (except for increasing run time somewhat). On Unix systems, if the value of the MEMMON environment variable begins with '|', the rest of the value is interpreted as a shell command into which the history is piped.

The Display

Icon has two primary allocated data regions: a string region and a block region. On implementations that support region expansion [2], there is also a static region. The display shows the regions as if they were contiguous, which they are on implementations that support region expansion. The static region, if it exists, comes first, followed by the string and block regions. The choice of regions that are displayed can be specified. By default, the static region is not displayed, but the string and block regions are.

Color distinguishes the various uses of memory, and by inference the region boundaries. A legend at the top of the screen gives the meaning of each color. Default colors can be changed by providing an alternate specification file.

Colors and boundaries are set at allocation time; subsequent changes are not reflected until garbage collection occurs. For example, a string constructed in pieces may not show as a concatenated whole.

The line above the color legend gives the program status at the left, the name of the allocation history file in the center, and storage information at the right, such as

```
60480 + 25600 + 51200   (0+0+1+0)
```

The first three numbers give the current sizes (in bytes) of the static, string, and block regions respectively. The ordering reflects that of the display. The first three numbers in parentheses count the garbage collections caused by exhaustion of the regions, with the fourth number counting garbage collections initiated by calls to `collect(0)`.

Garbage Collection

The garbage collection process is shown in detail by producing snapshots at critical points. The first comes at the beginning of garbage collection, and indicates the reason the garbage collection is required.

The next snapshot follows the marking phase. Active data is dark gray; the remaining blocks are garbage to be discarded.

The next snapshot shows the marked (active) data in color, before compaction, with the garbage painted black. This is the inverse of the previous display.

The last snapshot shows the state of memory at the end of garbage collection, after compacting the active data. All garbage is gone, and the string region shows a single unbroken string. At this point the image may be rescaled to handle region expansion.

Alien blocks (such as I/O buffers) in the static region are not subject to marking or garbage collection and instead remain on constant display throughout. Obsolete co-expression blocks are freed during the marking phase, but they are displayed in a manner similar to other blocks so that their disappearance can be noted.

The Programmer's Interface

Three built-in Icon functions write to the allocation history to control a subsequent MemMon run.

`mmpause(s)` generates a snapshot similar to those during garbage collection; the name comes from its effect on interactive visualization programs, which pause at this point. `s`, if supplied, is displayed to identify the pause. The default for `s` is `"programmed pause"`.

`mmshow(x,s)` redraws `x` if `x` is in the managed memory region. This can be used to identify one or more particular data objects on the display. `s` determines the color of `x`:

"w"	white
"g"	gray
"b"	black
"h"	highlight: blinking white and black
"r"	redraw in normal color (the default)

The altered display persists until the next garbage collection. If `x` is outside the managed memory region, no action is taken.

`mmout(s)` writes `s` (without further interpretation) as a separate line in the history file. This can be used to insert comments (beginning with `#`).

All three functions return the null value.

Accessing the Monitoring System

The **mmps** program is built in the **v8/src/memmon** subdirectory of the Icon distribution. Instructions for running **mmps** appear in Appendix B.

That subdirectory also includes some sample color specification files as well as code for building interactive visualization programs. Further information about this appears in [3].

References

1. R. E. Griswold and M. T. Griswold, *The Implementation of the Icon Programming Language*, Princeton University Press, 1986.
2. R. E. Griswold, *Supplementary Information for the Implementation of Version 7.9 of Icon*, The Univ. of Arizona Icon Project Document IPD51d, 1989.
3. G. M. Townsend, *Notes on MemMon Internals*, The Univ. of Arizona Icon Project Document IPD97a, 1989.
4. *Encapsulated PostScript File Format, Version 1.2*, Adobe Systems Incorporated, 1987.

Appendix A: Running mmps

mmps generates Encapsulated Postscript [4] displays of Icon memory. One or more images are produced under control of command options. Output files are Encapsulated PostScript documents suitable either for direct printing or for incorporation into other documents. The output includes full color information, though most PostScript devices print only the black-and-white equivalent.

The default image is 468 × 624 points, or 6.5" × 8.7", centered on a standard page. One line represents 1872 bytes of memory. Smaller or larger images can be specified; larger ones are reduced to fit within the above bounds.

Command format

mmps [options] [file]

Options

−r *regions*	Display the indicated memory regions: f static (fixed) region s string region b block region The default is −r sb.
−p *when*	Produce a snapshot at the indicated points: f memory full (beginning of garbage collection) g showing garbage remaining after marking a showing unmarked, active blocks after marking c after compaction (end of garbage collection) p explicit mmpause() calls d (‘‘done’’) when the program terminates n never The default is −p fgacpd.
−m	Run through the marking phase even when not pausing to display the results. Normally, marking is bypassed if neither of −p ga is selected.
−g *n*	Skip to the end of the *n*th garbage collection before displaying anything.
−q *n*	Quit after completing the *n*th garbage collection.
−Q *n*	Quit after the *n*th snapshot.
−b *n*	Make each horizontal point represent *n* bytes of memory. The default is 4.
−w *n*	Set the display width to *n* points. The default is 468.
−h *n*	Set the display height to *n* points. The default is 624.
−L *n*	Make the legend and status lines *n* points high. −L 0 eliminates the header entirely. The default is 11.
−M *n*	Limit the memory region lines to a maximum of *n* points in height. The default is 20.
−t *title*	Set the display title. The default is the allocation history file name.
−c *file*	Use an alternate color specification file (see Appendix B).
−S *n*	Set the PostScript screen frequency to *n* lines per inch.

Appendix B: Color Specification Files

A *color specification file* can be used to change some or all of the colors produced by **mmps**.

The environment variable MMCOLORS can be used to name a file containing color specifications. **mmps** reads this file and uses it to override the normal defaults. Then, if a file is passed by the **–c** option of **mmps**, it overrides both the built-in defaults and anything from an MMCOLORS file.

Lines in a color specification file contain two whitespace-separated fields, a label and a value, optionally followed by comments. Blank lines are ignored, as are lines beginning with # . For example:

```
#   change the colors for sets
set     657  light purple for set headers
selem   637  medium putple for set elements
```

The label field matches either one of the types shown in the legend or one of these additional keywords:

background	background
bsep	block separator
ssep	string separator
marked	marked block
unmarked	unmarked block (when showing active data)
status	status message
prompt	prompt message
title	title field
regions	region sizes

The value field is a set of three octal digits specifying a color. The digits control the red, green, and blue color components in that order, with a range of 0 to 7 for each. A value of 0 is dark and a value of 7 is light. For example, 070 is green (0% red + 100% green + 0% blue) and 405 is purple (4/7 red + 5/7 blue). Unfortunately, the final colors are somewhat device-dependent because of different responses to the same specification.

Appendix C: Allocation History Files

An allocation history file is composed of printable characters forming a sequence of commands that trace interpreter actions related to memory management. This section describes the overall structure of an allocation history file, using terms and commands that are described later in detail.

An allocation history file begins with a *refresh sequence*, which completely specifies the memory layout at a particular instant. The initial refresh sequence gives a snapshot of memory just before the start of execution. Within a refresh sequence, *item* commands enumerate all the objects within the three regions, as if they are being placed, in order, into initially empty regions.

After the initial refresh sequence, the rest of the file contains any number of the following components, in any order:

item commands
interaction commands
garbage collection sequences

Item commands, when outside other sequences, record new allocations of memory.

Interaction commands are generated by programmed calls to mmshow() and mmpause().

A garbage collection sequence begins with a *marking sequence*, in which item commands identify live objects. The marking sequence is followed by a new refresh sequence giving the memory configuration after compaction. Then, a final marker signals the end of garbage collection.

Comment and *verification* commands may appear at any point in the file.

Command Format

A command has several components:

[*addr*+] [*len*] *cmd* [*etc*]

addr	is an optional address, given as a distance from the start of a region. If the address is omitted, the current end of the region is assumed. An address is always followed immediately by + .
len	is a length. If a length is needed by a command, but none is supplied, then the most recent length specified for that particular command is used.
cmd	is a single character identifying the command.
etc	is additional information needed by a few particular commands.

Addresses and lengths are nonnegative decimal numbers. In the string region, the unit of measurement is a character; in the static and block regions, it is specified in the refresh sequence and usually is 4 bytes.

Whitespace between commands is optional; whitespace within a command is allowed only in *etc* data.

Item Commands

Item commands identify individual allocated objects. The meaning of an item command depends on its context. Within a refresh sequence, item commands enumerate the existing allocations. Within a marking sequence, item commands mark live objects. Otherwise, item commands announce new allocations.

Except during marking, string and block region item commands do not include addresses. Addresses may be obtained by totaling the allocations made since the beginning of the last refresh sequence.

The item commands for the block region are:

[*addr*+] [*len*] c	cset
[*addr*+] [*len*] e	table element trapped variable
[*addr*+] [*len*] E	external block
[*addr*+] [*len*] f	file block
[*addr*+] [*len*] L	list header
[*addr*+] [*len*] l	list element

[*addr*+] [*len*] R	record
[*addr*+] [*len*] r	real number
[*addr*+] [*len*] S	set header
[*addr*+] [*len*] s	set element
[*addr*+] [*len*] T	table header
[*addr*+] [*len*] t	table element
[*addr*+] [*len*] u	substring trapped variable
[*addr*+] [*len*] x	co-expression refresh block

The string item command is:

[*addr*+] [*len*] "	string

The static region item commands always include an address. They are:

addr + [*len*] X	co-expression block
addr + [*len*] A	alien block

Refresh Sequence Commands

[*units*] < *static-region string-region block-region*

Begin a refresh sequence. Each region specification has the form

base : *used* / *max*

where *base* is the beginning address of the region, *used* is the amount of memory used, and *max* is the amount of memory allocated to the region. For the static region, the *used* value is meaningless, and *base* and *max* are zero with fixed-region versions of Icon. All values in this command are given in bytes. The *units* parameter, if present, gives the size of a unit of measurement for other commands referencing the static and block regions. If the *units* parameter is absent, the unit of measurement is four bytes.

> End a refresh sequence

Verification Command

= *static-region block-region string-region*

Region specifications are the same as for the < command. This command provides no new information but confirms the accumulated memory usage after a series of allocations.

Garbage Collection Commands

n { Start a garbage collection and begin the marking phase

n indicates the reason for the collection:

0	collect(0) call
1	static region
2	string region
3	block region

} End the marking phase

! End garbage collection

Interaction Commands

; *string*

Pause (produce a snapshot) and display *string*, which includes all characters up to a newline. This command is generated by a programmed mmpause(*string*) call.

addr + *len* $ *c t*   highlight a string
addr + *len* % *c t*   highlight a block
addr + *len* Y *c t*   highlight a static object

Highlight commands are generated by programmed mmshow(*x,s*) calls. *c* is the first character of the argument string *s*, indicating the kind of highlighting desired. *t* identifies the type of the object being highlighted by giving the character used for an allocation command of that type.

Comment Command

# *comment*

All characters following the #, up to a newline, are ignored.

Example

Here is a small, contrived program that builds a list of strings, then inserts the strings in a set:

```
procedure main ()
   l := list ()
   every put (l, string (-50 to 50))
   s := set ()
   every insert (s, !l)
   end
```

Here is the corresponding history file:

```
4< 234076:60000/60000 294080:0/65024 359104:0/65024
2+2666F2670+2050A4722+2A4726+4A4732+8A4742+256A5000+10000X
0"
>
= 234076:60000/60000 294080:0/65024 359104:0/65024
5L23|L|3"""""""""|""""""""|""""""""31|""""""""""""43|""""2"""""""""1""
"""61|"""""2""""""""""""""""""""""87|"""""""""""""""""""14S10h5sssssss
ssssssssssssssssssssssssssssssssssssshssssssssssssssssssssssssssssssssssss
sssss18hsssssssssssssssssssss
= 234076:60000/60000 294080:233/65024 359104:3524/65024
```