Princeton University
COS 217: Introduction to Programming Systems
The Memstat Tool

What is it?

Memstat is a simple tool to help you analyze your application's dynamic memory management. In particular, it can help you find memory leaks and multiple frees. It may help you to find other dynamic memory management errors as well.

How do I use it?

Suppose you wish to use the memstat tool to help you debug an application named myapp. Further suppose that myapp consists of source code files mysourcecode1.c and mysourcecode2.c. Follow these steps:

(1) Set the **PATH** environment variable so it includes directory /u/cos217/bin/i686. Using the bash shell, you do that by issuing the command:

```
export PATH=/u/cos217/bin/i686:$PATH
```

Note that the file /u/cos217/.bashrc contains that command; if you are using the bash shell and you have copied the /u/cos217/.bashrc file to your home directory, then you need not manually issue the command. You can confirm that the PATH environment variable contains directory /u/cos217/bin/i686 by examining the output of the printenv command.

(2) Set the **MEMSTATDIR** environment variable equal to /u/cos217/bin/i686. Using the bash shell, you do that by issuing the command:

```
export MEMSTATDIR=/u/cos217/bin/i686
```

Again, note that the file /u/cos217/.bashrc contains that command; if you are using the bash shell and you have copied the /u/cos217/.bashrc file to your home directory, then you need not manually issue the command. You can confirm the setting of the MEMSTATDIR environment variable by examining the output of the printenv command.

(3) Use the **gccmemstat** (instead of the **gcc**) command to preprocess, compile, and assemble mysourcecode1.c and mysourcecode2.c:

```
gccmemstat –Wall –ansi –pedantic –c mysourcecode1.c
gccmemstat –Wall –ansi –pedantic –c mysourcecode2.c
```

(4) Use the **gccmemstat** (instead of the gcc) command to link mysourcecode1.o and mysourcecode2.o, thus creating executable file myapp:

```
gccmemstat –o myapp mysourcecode1.o mysourcecode2.o
```

Note that steps 3 and 4 can be combined by issuing a single command:

```
gccmemstat –Wall –ansi –pedantic –o myapp mysourcecode1.c mysourcecode2.c
```

(5) Execute myapp as usual, by typing its name (and command-line arguments, as appropriate):

```
myapp arg1 arg2 ...
```

Doing so generates a text file in the current directory named memstatX.out, where X is the id of the process in which myapp executed.

(6) Optionally, use a text editor to examine the memstatX.out file:

```
xemacs memstatX.out
```

Note that the file contains one line for each call to malloc(), calloc(), realloc(), and free() performed by process X.

(7) Use the **memstatreport** program to generate (to stdout) a summary report of memstatX.out, and thus of process X's dynamic memory management:

```
memstatreport memstatX.out
```

The first part of the report shows the number of bytes allocated and "deallocated" on a line-by-line basis. A positive number indicates a memory allocation; a negative number indicates a memory deallocation. The second part of the report shows the total number of bytes allocated/deallocated by each compilation unit. Usually, the total for each compilation unit should be 0. The last line of the report shows the total number of bytes allocated/deallocated by the entire application. That number certainly should be 0.

If the total number of bytes allocated/deallocated by the entire application is a positive number, then your application contains memory leaks. In that case you should analyze the more detailed information in the report to help you determine which dynamically allocated memory is not being freed.

If the total number of bytes allocated/deallocated by the entire application is a negative number then your application contains multiple frees. In that case you should analyze the more detailed information in the report to help you determine which dynamically allocated memory is being freed more than once.

How does it work?

Memstat is a very simple tool. The code that comprises memstat is available in directory /u/cos217/bin/i686. Please study it. Specifically, directory /u/cos217/bin/i686 contains these files:

memstat.h

memstat.h is the header file for the memstat utility. The gccmemstat command automatically includes memstat.h into each .c file that it preprocesses.

memstat.h declares functions Memstat_malloc(), Memstat_calloc(), Memstat_realloc(), and Memstat_free(). It also uses the C preprocessor to alter your .c files so each instance of the text "malloc" is changed to "Memstat_malloc", each instance of "calloc" is changed to "Memstat_calloc", each instance of "realloc" is changed to "Memstat_realloc", and each instance of "free" is changed to "Memstat_free". In that way, the memstat tool "intercepts" your program's calls to C's standard dynamic memory management functions.

memstat.c

memstat.c contains the definitions of the Memstat_malloc(), Memstat_calloc(), Memstat_realloc(), and Memstat_free() functions.

The first time any of those functions is called, it opens a file named memstatX.out. Subsequently, each function writes a line to memstatX.out indicating that it has been called, along with appropriate data. It then proceeds to call the corresponding standard C function.

With one complication... Unknown to your application, the Memstat_malloc(), Memstat_calloc(), and Memstat_realloc() functions actually allocate a block of memory that is slightly larger than you requested, and store the number of bytes that you requested in a hidden area at the beginning of the memory block. The Memstat_realloc() and Memstat_free() functions then use that hidden information to write appropriate data to memstatX.out.

libmemstat.a

libmemstat.a is a UNIX library (alias archive) that contains the compiled version of memstat.c. It was created from the memstat.o file using the command:

```
ar rs libmemstat.a memstat.o
```

See chapter 4 of our Loukides and Oram textbook for an explanation of UNIX libraries. Page 102 explains the "ar" command.

gccmemstat

gccmemstat is a bash script which calls gcc with appropriate options. It uses the "-include memstat.h" option so gcc includes memstat.h into each .c file that it preprocesses. It uses the "-L$MEMSTATDIR" option to command gcc to look in directory $MEMSTATDIR (that is, /u/cos217/bin/i686) for libraries at link time. It uses the "-lmemstat" option to command gcc to link with the libmemstat.a library.

See page 88 of our Loukides and Oram textbook for more information about the "–L" and "–l" options to gcc.

memstatreport.c

memstatreport.c contains the source code for the memstatreport program.

Note that it uses an ADT named DynArray. The DynArray ADT is discussed in precepts. The source code for the DynArray ADT is provided as a precept handout.

memstatreport

memstatreport is the binary executable file created from memstatreport.c.