UNIVERSITY OF WISCONSIN
Computer Sciences Department

CS 537
Fall 2018

Barton Miller

Programming Assignment #4: A Safe malloc/free library

Handed out: Tuesday, November 15
Due: Tuesday, December 4, 5pm

There are several goals for this assignment.

1. Learn about dynamic memory (heap) allocation.
2. Learn about how tools work that check for memory errors.
3. Learn about how Linux (UNIX) signal handlers work for capturing error conditions in your program.
4. Write code that is a component -- a library -- instead of a complete program.

Your assignment is to write a special version of the `malloc` and `free` library routines that checks on common heap allocation and usage mistakes. Your version of `malloc` and `free` will be called `malloc537` and `free537`. You will also write one extra function, `memcheck537`, for checking if a heap allocated memory address range is safe to use.

A programmer will use your versions of `malloc` and `free` wherever they would use the real ones. Your `malloc537` and `free537` will first check to see operation is safe and makes sense, and then call the real `malloc` or `free` to get the job done. In addition, whenever your program needs to access heap memory, they will insert a extra call to `memcheck537` into their code to verify if that address range is valid.

Definitions of `malloc537`, `free537` and `memcheck537`

Here is a description of the functions that you will need to write:

`void *malloc537(size_t size):`

In addition to actually allocating the memory by calling `malloc()`, this function will record a tuple ($addr_i$, len_i), for the memory that you allocate in the heap. (If the allocated memory was previously freed, this will be a bit more complicated.) You will get the starting address, $addr_i$, from the return value from `malloc()` and the length, len_i, from the `size` parameter. You can check the `size` parameter for zero length (this is not actually an error, but unusual enough that it is worth reporting).

`void free537(void *ptr):`

This function will first check to make sure that freeing the memory specified by `ptr` makes sense, then will call `free()` to do the actual free. Some of the error conditions that you should check for include:

1. Freeing memory that has not be allocated with `malloc537()`.
2. Freeing memory that is not the *first* byte of the range of memory that was allocated.
3. Freeing memory that was previously freed (double free).

When an error is detected, then print out a detailed and informative error message and exit the program (with a -1 status). If all checks pass,then this function indicates that the tuple for *addr* = `ptr` is no longer allocated, and then calls `free()`.

`void *realloc537(void *ptr, size_t size):`

If `ptr` is NULL,then this follows the specification of `malloc537()` above. If `size` is zero and `ptr` is not NULL,then this follows the specification of `free537()` above. Otherwise, in addition to changing the memory allocation by calling `realloc()`, this function will first check to see if there was a tuple for the (*addr* = `ptr`, and removes that tuple, then adds a new one where *addr* is the return value from `realloc()` and *len* is `size`

`void memcheck537(void *ptr, size_t size):`

This function checks to see the address range specified by address `ptr` and length `size` are fully within a range allocated by `malloc537()` and memory not yet freed by `free537()`. When an error is detected, then print out a detailed and informative error message and exit the program (with a -1 status).

Here are some examples of its use, showing the `memcheck537()` call that you would insert into your code:

```
char *buff = (char *)malloc537 (10);
char c;
struct node *np = (struct node *)malloc537 (sizeof(struct node));
struct node n;
int *ip = (int *)malloc537 (sizeof(int));

memcheck537 (&(buff[5]), sizeof(char));
c = buff[5];

memcheck537 (buff, strlen("hi mom!"));
strcpy(buff, "hi mom!");

memcheck537 (np, sizeof(struct node));
memcopy (np, &n, sizeof(struct node));
```

Code Details and Testing

An important part of your program will be a data structure that can efficiently handle *range queries*, which means looking up addresses within a range. This is not a simple data structure to implement efficiently.

You will write a module called `537malloc.c` with the four functions described above. In addition, you will provide an include file, `537malloc.h`, with the function prototypes exactly as described above.

Important: You will test your code with various test programs by modifying those programs to use your new heap routines instead of the standard ones, and insert calls to `memcheck537` wherever heap allocated memory is used. You are likely to be able to use the programs you wrote for Program 1 or Program 2 as test program. In addition, we will provide some test programs with which to run your code.

Deliverables

Most of you will work in **groups of two** and turn in a single copy of your program, clearly labeled with the name and logins of both authors. Any exception to working in a pair must be approved by me in advance; there must be some unusual reason for not working in a group.

You will turn in your programs, including all .c and .h files and your Makefile. Also include a README file with your names and indicating on which lab machines you compiled and ran your program.

Note that you must run your programs on the Linux systems provided by the CS Department. You can access these machines from the labs on the first floor of the Computer Sciences Building or from anywhere on

the Internet using the ssh remote login facility. If you do not have ssh on your Windows machine, you can download this client:

SSHSecureShellClient-3.2.9.exe

Make sure to test your code after you copy it to the hand-in directory. This semester, several students have gotten caught by errors in this step.

Handing in Your Assignment

Your CS537 handin directory is `~cs537-1/handin/your_login` where *your_login* is your CS login. Inside of that directory, you need to create a `proj4` subdirectory (unless the TAs created one for you already).

Copying your files to this directory is accomplished with the `cp` program, as follows:

```
shell% cp *.[ch] makefile README ~cs537-1/handin/your_login/proj4
```

You can hand files in multiple times, and later submissions will overwrite your previous ones. To check that your files have been handed in properly, you should list `~cs537-1/handin/your_login/proj4` and make sure that your files are there. The handin directories will be closed after the project is due.

As most of you are working in pairs, you should:

1. Submit only one copy of your code.
2. Make sure that both of your names are in comments at the top of each source file.
3. Both of you should create another file called partner.txt in **both** of your `proj3` directories, where you should have two lines that have each of CS login and netid of you and your partner. You will find a template for this file in your `proj4` directory.

Original Work

This assignment must be the original work of you and your project partner. Unless you have explicit permission from Bart, you may not include code from any other source or have anyone else write code for you.

Use of unattributed code is considered plagiarism and will result in academic misconduct proceedings (and "F" in the course and a notation on your transcript).

Last modified: Mon Nov 26 14:33:54 CST 2018 by bart