Machine Problem 1 Q&A

Fall 2015

CSCE 313 – Introduction to Computer System

**C Programming References**

<http://books.cat-v.org/computer-science/c-programming-language/The.C.Programming.Language.2nd.Edition.pdf>

<http://aelinik.free.fr/c/>

**Seg Fault Due to Null-terminated Test Case**

Line 28 of main.c: " Insert (13, msg, strlen(msg)+1); // insertion of strings, copies the null byte at the end"

Line 46 of main.c: " printf ("Key = %d, Value Len = %d, Value = %s\n", \*(int \*) kv, \*(int \*) (kv+4), kv + 8);"

This line is printing kv(the char\* that was passed back by Lookup), but when it prints the string, I'm getting a segmentation fault. So, to my question:

Are we allowed to pre-process and remove the null byte? What's the best way to handle it?

Answer: You can do that. But for this particular example, to be able to print the string using printf("%s"), the null byte is necessary. If you had implemented your Insert function correctly, this should not cause any segfaults. And, if you are trying to preprocess and remove NULL, you are making assumption about the value part, which is not expected in this MP.

**Using Functions for Linked List**

Can we change the return values given for the functions? For example, I am confused on how to keep track of a new linked list created by Init (int M, int b).

If it doesn't return a pointer to the head node, but returns void, then are we supposed to use global variables or something? I would prefer to change parameters and return values so that the functions act on a linked list declared in the main.

Answer: Yes, you can use global variables for the major pointers (e.g., head-pointer, free-pointer). Having global variables is better, because in that case, your main function is not burdened with too much information (e.g., having to keep the head-pointer returned by Init(), and passing that to each Insert /Delete () call).

**Trouble with using the malloc**

I think I don’t quite understand the exact requirements of part 1 of the project. What I think it to be is:

Allocate some block of memory on the heap (the memory pool) and then fill this block of memory with our headers and payloads as the functions call it.

My problem with this is that new variables, such as the key, value and value length, will not be created on the heap per GNU standards, so the header portion makes sense but how can the payload be assigned to some part of the memory pool? The only thing I can think of is writing directly to the memory address ala

*char \* p = (char \*)0xfffffff;*

*\*p = abc;*

where 0xfffffff would be the actual address inside our block we receive from malloc, p being whatever value we're trying to assign (next memory address, value length, key, value). This doesn't seem right, and I feel like I'm missing something. Thanks for any help,

Answer: You are right about the idea that you have to somehow use the address returned by malloc, say m. You can compute where each block starts from m (i.e., the first block is at m, the second block is at m + 128 (assuming block size is 128) and so on). Once you know where a block starts, you just know that first few bytes starting from there make up a next-pointer, then a 4 byte key, after that a value-length and finally the value.

And once you know all these, the only missing part is how to place a particular data in a given address. Your example is nearly correct. So if you are placing a integer k at an address given by the variable p , you need to do this:

*char \*p = where the block starts, which you computed some how*

*\*(int \*) p = k*

If you are then planning to place a value (pointed to by vp) of length vl immediately following the integer in memory:

*p = p + sizeof (int);*

*memcpy (p, vp, vl);*

I hope this helps.

**What information does value\_len give?**

In the problem it says that each block will consist of a pointer, a key, a value length, and a value. My question is does value\_len contain the number of bytes that represent value? Either way, can't we just use b to figure out how many bytes to advance the pointer anyway?

Answer: You are right. You don't need value length for advancing the pointer. But the value length is kept for the user who does not care to remember the length of every value that he writes.

**Int pointers for MP1**

Can we use int pointers or does it have to be char pointers? Initially I assumed we could, but the return value for the lookup function is a char pointer.

Answer: The type of the pointer does not really matter, as long as you cast it to the required type before using it.

**Structs on MP1**

We can create node structs right?

Answer: In fact, working with structs is more systematic. You are encouraged to used structs.

**int insert()?**

What should insert return since it has a return value of int and its not assigned to anything in main? Can we change it to void?

Answer: The return value indicates success(1) or failure (0) to insert. So, you should keep it that way.