COMP1927 – Ass1

1 WHAT WE DID WRONG

1.1 TREAT THE HEADER STRUCT AS IF IT USED POINTERS

1.1.1 The Problem

So, this is a huge difference in coding that we have to comply with (assignment spec), and we ignored. Consider the struct from lab03:

```
13  typedef struct ListNode {
14      Item value;  // value of this list item
15      struct ListNode *prev;
16                   // pointer previous node in list
17      struct ListNode *next;
18                   // pointer to next node in list
19  } ListNode;
20
21  typedef struct ListRep {
22      int  nitems;      // count of Items in list
23      ListNode *first; // first node in list
24      ListNode *last;  // last node in list
25      ListNode *curr;  // current node in list
26  } ListRep;
```

Note the pointers. Specifically, if you had a ListNode named Bob, and ran the following:

Printf("%p", Bob->next);

You'd get a memory address, something like 0x12345678 (hexadecimal, I don't know that [redacted])

Now consider our struct:

```
typedef unsigned char byte;
typedef u_int32_t vlink_t;
typedef u_int32_t vsize_t;
typedef u_int32_t vaddr_t;

typedef struct free_list_header {
   u_int32_t magic;           // ought to contain MAGIC_FREE
   vsize_t size;              // # bytes in this block (including header)
   vlink_t next;              // memory[] index of next free block
   vlink_t prev;              // memory[] index of previous free block
} free_header_t;
```

Note the terrifying lack of pointers (no *s). By contrast, if you had a free_header_t named Bob, and ran the following:

Printf("%d", Bob->next);

You'd get AN INTEGER, something like 4. THIS SUCKS [redacted]

1.1.2 The Solution

These babies:

```
189  //Helper Functions
190  //Converts an index to a pointer
191  free_header_t* toPointer(vlink_t index) {
192      return ((free_header_t*)(memory + index));
193  }
194
195  //Converts a pointer to an index
196  vlink_t toIndex(free_header_t* pointer) {
197      return (pointer - (free_header_t *)memory);
198  }
```

These functions convert the index (which is useful as shit) into a pointer (which is useful), and vice versa. Here's an example of them being used:

```
248  //print out all headers
249  void printHeaders(void) {
250
251      //Start from the beginning
252      vlink_t curr = free_list_ptr;
253
254      do {
255          //Print this header
256          printf("curr(index): %d\n", curr);
257          printf("curr(pointer): %p\n", toPointer(curr));
258          printf("curr->MAGIC: 0x%08x\n", toPointer(curr)->magic);
259          printf("curr->size: %d\n", toPointer(curr)->size);
260          printf("curr->next: %d\n", toPointer(curr)->next);
261          printf("curr->prev: %d\n\n", toPointer(curr)->prev);
262
263          //Move along
264          curr = toPointer(curr)->next;
265      } while (curr != free_list_ptr);
266
267      return;
268  }
269
```

Basically, if you want to access the curr->next or curr->prev, any of the useful stuff – you need to wrap the curr in the toPointer function. Here's a super lame example where the nesting makes this a headache:

```
229  //Converts a region from free to allocated, and removes it from the free list
230  vlink_t enslaveRegion(vlink_t curr) {
231
232      //Mark header as allocated
233      toPointer(curr)->magic = MAGIC_ALLOC;
234      //Change neighbour's links to skip the enslaved region
235      toPointer(toPointer(curr)->prev)->next = toPointer(curr)->next; //i feel like this
236      toPointer(toPointer(curr)->next)->prev = toPointer(curr)->prev;
237      //Destroy links within the allocated header
238      toPointer(curr)->next = curr;
239      toPointer(curr)->prev = curr;
240
241      return curr;
242  }
```

*Note that I'm not 100% sure this code is correct, although I'm pretty sure the use of toPointer is correct (it compiles at least)

This is needed because curr is of type vlink_t curr. Specifically – curr is an index (vlink_t is a u_int32_t, which is an unsigned 32bit int). In previous exercises, we would've defined curr as a LinkNode or something to that effect, which handled numbers neatly. This time however, it's just a sad old integer.

This took me a long while and much explanation to comprehend – it's a [redacted] pain, but the program is compiling, and so far is at least allocating the first header (nothing beyond that).

1.2 NOT COMPILE AND TEST THE PROGRAM AS WE WENT ALONG

Especially considering the size of this task, we needed to compile and test each section as we wrote it. It's especially erroneous because they gave us a testing program, and debugging tools with sal_stat. Before we start writing a new function, all of the existing ones should A. compile without error and B. actually work. The compile error printouts for allocatorBROKEN.c are insurmountable.

After writing any code, this is what we should have done:

```
wagner % make
gcc -Wall -Werror   -c -o allocator.o allocator.c
gcc   run_sal.o allocator.o   -o run_sal
wagner % ./run_sal < sal_test
+ a 100

passCount = 1, regionFound = 1, curr = 0x8410008, curr->next = 0x8410008
loop escaped
loop escaped
ptr[a] allocated 0x8410108
+ b 50

Memory corruption
sal_stats

Global Variable 'memory' is: 0x8410008
Global Variable 'free_list_ptr' is: 0
Global Variable 'memory_size' is: 4096

curr(index): 0
curr(pointer): 0x8410008
curr->MAGIC: 0xbeefdead
curr->size: 128
curr->next: 0
curr->prev: 0

Aborted
wagner %
```

That's the latest test of the program, you can see that it aborts in sal_malloc (look at the source), and generates a nice list of statistics to help solve the problem (in this case, the search for a free list came up with an allocated one). The file sal_test.txt is in the ass1 directory, its currently just what was on the assignment page but we'll have to upgrade it before we submit.

I think most of the code logic we wrote is correct, but importing it from allocatorBROKEN.c has to be done one step at a time, and each section tested and fixed.

1.3 NOT USE GITHUB'S VERSION CONTROL PROPERLY

So if you look at the last few hours – I've got 10 or so commits. The recommendation I received was that each "commit" operation within github should reflect some new feature – at least a new function() but often just a chunk of meaningful code.

Takeaway: commit far more often. It provides us with a better history to undo changes.

2 WHAT WE NEED TO DO NOW

So, the furthest I got working on allocator.c is stored as allocatorBROKEN.c. This file is – broken, its not much more advanced from previous commits but it contains a lot of the toPointer stuff which is necessary to make allocator.c work. We need to migrate all the remaining code in broken to allocator, one function at a time, testing each one and making sure it works, fixing any errors we find.

2.1 FUNCTIONS

2.1.1 Functions Fully Imported and Fixed

- Sal_init
- Sal_end
- sizeToN
- toPointer
- toIndex
- printHeaders

2.1.2 Functions imported, but not yet working 100%

- Sal_malloc
- EnslaveRegion
- memoryDivide

I'm actually not sure which of these work and don't work, currently the program terminates within sal_malloc but the problem is caused (I think) by one of the other two)

2.1.3 Functions not yet imported

- Sal_free
- Merge

I've got the shell of merge in there, so it doesn't cause compile errors if you call it. Same for sal_free.

2.2 OBSTACLES

I've got an assignment due Wednesday 10am, and it's mostly blank page, so I need to devote all my time to that until it's done. I've put a message on OL clarifying the due date of this task (either Thursday or Friday midnight), but either way I have Thursday free, plus nights

Don't worry too much about the lab – looks like a [redacted] but kinda fun. It's low priority for now.

I'm working today 930am – 3pm, but as this is now my second night without sleep I'm not sure what state I'll be in tonight, and I'll probably still have that assignment to work on. Read the comments on the commits, they (poorly) explain what's happened.

3 WE'RE RUNNING OUT OF TIME

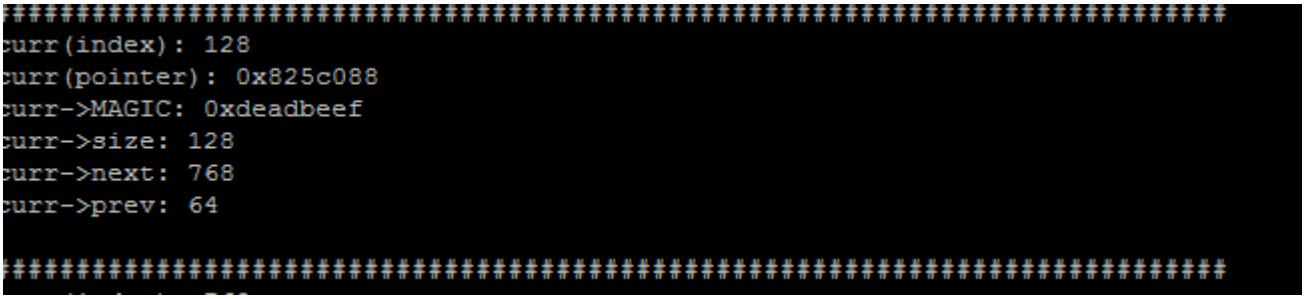
Ok Daniel so, it looks like sal_malloc works (honestly impossible to be sure, but no errors yet). We next need to make sal_free and merge work. I've got work today, so you'll need to debug those functions, and test them to see if it works. There's 3 test scripts you can run:

- Sal_test – this is the exact copy of the one on the assignment page
- Sal_test 721am – this test should be passed all the time from now on, it passed at 721am
- Sal_test_pro – this is one I downloaded off of openLearning. Haven't tried it yet

./run_sal < sal_test

Do this until it works.

The next problem I see is that after initialising a few variables, we start getting non 2^n size headers



The position of next…

If you've got friends who've done this before or have tons of experience, grab their help.