

**CMSC216: Practice Exam 1B**

Spring 2025

University of Maryland

*Exam period: 20 minutes   Points available: 40   Weight: 0% of final grade*

**Problem 1 (15 pts):** Nearby is a `main()` function demonstrating the use of the function `mon_max_type()`. Implement this function according to the documentation given. *My solution is about 29 lines counting closing curly braces.*

YOUR CODE HERE

```

1 #include "mon.h"
2 typedef struct { // info on mon stored in
3     char name[64]; // text files
4     int hp;
5     char type[64];
6 } mon_t;
7
8 mon_t *mon_max_type(char *fname, char *filter);
9 // Opens the file 'fname' which is formatted:
10 //     Fearow 442 Normal
11 //     Arbok 438 Poison
12 //     Raichu 485 Electric
13 //     Sandslash 450 Ground
14 //     ...
15 // as NAME HP TYPE for an arbitrary number of
16 // lines. Reads all mon in the file and
17 // identifies the mon with type 'filter' that
18 // has the maximum HP. Returns a heap-allocated
19 // mon_t struct with the max-HP mon in it. All
20 // HP are assumed to be 0 or more.
21 //
22 // Two failure cases are handled:
23 // 1. If file 'fname' cannot be opened prints
24 //     Failed to open file '<NAME>'
25 //     replacing <NAME> with 'fname', then
26 //     returns NULL.
27 // 2. If no mon match type 'filter' prints
28 //     No matches for filter '<FILTER>'
29 //     replacing <FILTER> with 'filter', then
30 //     returns NULL.
31
32 int main(){
33     mon_t *res;
34     res = mon_max_type("mon-lvl1.txt","Grass");
35     printf("%s %d %s\n",
36         res->name, res->hp, res->type);
37     // Tangela 435 Grass
38     free(res);
39
40     res = mon_max_type("mon-lvl2.txt","Water");
41     printf("%s %d %s\n",
42         res->name, res->hp, res->type);
43     // Gyarados 540 Water
44     free(res);
45
46     res = mon_max_type("mon-all.txt","Computer");
47     // No matches for filter 'Computer'
48     // (nil)
49     printf("%p\n",res);
50
51     res = mon_max_type("no-file.txt","Fire");
52     // Failed to open file 'no-such-file.txt'
53     // (nil)
54     printf("%p\n",res);
55
56     return 0;
57 }

```

**Problem 2 (15 pts):** Nearby is a small C program which makes use of arrays, pointers, and function calls. Fill in the tables associated with the approximate memory layout of the running program at each position indicated. Assume the stack grows **to lower memory addresses** and that the sizes of C variable types correspond to common 64-bit systems.

```

1 #include <stdio.h>
2 void flub(double *ap, double *bp){
3     int c = 7;
4     if(*ap < c){
5         *ap = bp[1];
6     }
7     // POSITION B
8     return;
9 }
10 int main(){
11     double x = 4.5;
12     double arr[2] = {3.5, 5.5};
13     double *ptr = arr+1;
14     // POSITION A
15     flub(&x, arr);
16     printf("%.1f\n",x);
17     for(int i=0; i<2; i++){
18         printf("%.1f\n",arr[i]);
19     }
20     return 0;
21 }

```

POSITION A

Frame	Symbol	Address	Value
main()	x	#3064	
	arr[1]	#3056	
	arr[0]		
	ptr		
	i		?

POSITION B

Frame	Symbol	Address	Value
main()	x	#3064	
	arr[1]	#3056	
	arr[0]		
	ptr		
	i		?
flub()			
			7

**Problem 3 (10 pts):** The code below in fill\_pow2.c has a memory problem which leads to strange output and frequent segmentation faults. A run of the program under Valgrind reports several problems summarized nearby. **Explain these problems in a few sentences and describe specifically how to fix them.** You may directly modify the provided in code.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // allocate and fill an array
5 // with len powers of 2
6 int *fill_pow2(int len){
7     int arr[len];
8     int *ptr = arr;
9     int pow = 1;
10    for(int i=0; i<len; i++){
11        arr[i] = pow;
12        pow = pow * 2;
13    }
14    return ptr;
15 }
16 int main(){
17     int *twos4 = fill_pow2(4);
18     for(int i=0; i<4; i++){
19         printf("%d\n",twos4[i]);
20     }
21     free(twos4);
22     return 0;
23 }

```

```

1 >> gcc -g fill_pow2.c
2
3 >> valgrind ./a.out
4 ==6307== Memcheck, a memory error detector
5 ==6307== Conditional jump or move depends on uninitialised value(s)
6 ==6307==    by 0x48CB13B: printf (in /usr/lib/libc-2.29.so)
7 ==6307==    by 0x10927B: main (fill_pow2.c:19)
8 1
9 0
10 0
11 0
12 ==6307== Invalid free() / delete / delete[] / realloc()
13 ==6307==    at 0x48399AB: free (vg_replace_malloc.c:530)
14 ==6307==    by 0x109291: main (fill_pow2.c:21)
15 ==6307==    Address 0x1fff000110 is on thread 1's stack
16 ==6307==
17 ==6307== HEAP SUMMARY:
18 ==6307==    in use at exit: 0 bytes in 0 blocks
19 ==6307==    total heap usage: 0 allocs, 1 frees

```