CMSC216: Practice Exam 1B SOLUTION
Spring 2026
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.*

```
1  #include "mon.h"
2  typedef struct {  // info on mon stored in
3    char name[64];  // text files
4    int bp;
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 BP 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 BP. Returns a heap-allocated
19 // mon_t struct with the max-BP mon in it. All
20 // BP 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->bp, 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->bp, 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 }
```

```
1  ////////////////// SOLUTION //////////////////
2  #include "mon.h"
3
4  // VARIATION A: break from loop, heap-allocate early
5  mon_t *mon_max_type(char *fname, char *filter){
6    FILE *fin = fopen(fname,"r");
7    if(fin == NULL){
8      printf("Failed to open file '%s'\n",fname);
9      return NULL;
10   }
11   mon_t *max_mon = malloc(sizeof(mon_t));
12   max_mon->bp = -1;
13   while(1){
14     mon_t mon;
15     int ret = fscanf(fin,"%s %d %s", mon.name, &mon.bp, mon.type);
16     if(ret == EOF){
17       break;
18     }
19     if(strcmp(filter,mon.type)==0 && mon.bp > max_mon->bp){
20       max_mon->bp = mon.bp;
21       strcpy(max_mon->name, mon.name);
22       strcpy(max_mon->type, mon.type);
23     }
24   }
25   fclose(fin);
26   if(max_mon->bp == -1){
27     printf("No matches for filter '%s'\n",filter);
28     free(max_mon);
29     return NULL;
30   }
31   return max_mon;
32 }
33
34 // VARIATION B: complex loop condition, heap-allocate late
35 mon_t *mon_max_typeB(char *fname, char *filter){
36   FILE *fin = fopen(fname,"r");
37   if(fin == NULL){
38     printf("Failed to open file '%s'\n",fname);
39     return NULL;
40   }
41   mon_t mon;
42   mon_t max ={ .bp=-1 };
43   while(fscanf(fin,"%s %d %s", mon.name, &mon.bp, mon.type)
44        != EOF)
45   {
46     if(strcmp(filter,mon.type)==0 && mon.bp > max.bp){
47       max.bp = mon.bp;
48       strcpy(max.name, mon.name);
49       strcpy(max.type, mon.type);
50     }
51   }
52   fclose(fin);
53   if(max.bp == -1){
54     printf("No matches for filter '%s'\n",filter);
55     return NULL;
56   }
57   mon_t *max_mon = malloc(sizeof(mon_t));
58   max_mon->bp = max.bp;
59   strcpy(max_mon->name, max.name);
60   strcpy(max_mon->type, max.type);
61   return max_mon;
62 }
```

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 SOLUTION

Frame	Symbol	Address	Value
main()	x	#3064	4.5
	arr[1]	#3056	5.5
	arr[0]	#3048	3.5
	ptr	#3040	#3056
	i	#3036	?

POSITION B SOLUTION

Frame	Symbol	Address	Value
main()	x	#3064	5.5
	arr[1]	#3056	5.5
	arr[0]	#3048	3.5
	ptr	#3040	#3056
	i	#3036	?
flub	ap	#3028	#3064
	bp	#3020	#3048
	c	#3016	7

NOTES
- Both Pos A and B are before i is assigned 0 so i remains undefined

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 specifially how to fix them**. You may directly modify the provided code in place.

```
1  /////////// SOLUTION ///////////////
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  int *fill_pow2(int len){
6    // malloc the array so it is on
7    // the heap instead of stack
8    int *arr = malloc(sizeof(int)*len);
9    int pow = 1;
10   for(int i=0; i<len; i++){
11     arr[i] = pow;
12     pow = pow * 2;
13   }
14   return arr;
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); // free now
22   return 0;    // works fine
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
```

SOLUTION: The memory allocation in `fill_pow2()` *is all on the stack. In order to return an array, the function should use* `malloc()` *to allocate an array as indicated and return a pointer to that array after filling it.*