Name: **SOLUTION** UID#: DirectoryID:

CMSC216: Practice Exam 1B SOLUTION

Fall 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.*

```
#include "mon.h"
typedef struct {  // info on mon stored in
  char name[64];  // text files
  int bp;
  char type[64];
} mon_t;

mon_t *mon_max_type(char *fname, char *filter);
// Opens the file `fname` which is formatted:
//   Fearow 442 Normal
//   Arbok 438 Poison
//   Raichu 485 Electric
//   Sandslash 450 Ground
//   ...
// as NAME BP TYPE for an arbitrary number of
// lines. Reads all mon in the file and
// identifies the mon with type `filter` that
// has the maximum BP. Returns a heap-allocated
// mon_t struct with the max-BP mon in it. All
// BP are assumed to be 0 or more.
//
// Two failure cases are handled:
// 1. If file `fname` cannot be opened prints
//      Failed to open file '<NAME>'
//    replacing <NAME>  with `fname`, then
//    returns NULL.
// 2. If no mon match type `filter` prints
//      No matches for filter '<FILTER>'
//    replacing <FILTER>  with `filter`, then
//    returns NULL.

int main(){
  mon_t *res;
  res = mon_max_type("mon-lvl1.txt","Grass");
  printf("%s %d %s\n",
         res->name, res->bp, res->type);
  // Tangela 435 Grass
  free(res);

  res = mon_max_type("mon-lvl2.txt","Water");
  printf("%s %d %s\n",
         res->name, res->bp, res->type);
  // Gyarados 540 Water
  free(res);

  res = mon_max_type("mon-all.txt","Computer");
  // No matches for filter 'Computer'
  // (nil)
  printf("%p\n",res);

  res = mon_max_type("no-file.txt","Fire");
  // Failed to open file 'no-such-file.txt'
  // (nil)
  printf("%p\n",res);

  return 0;
}
```

```
////////////////////// SOLUTION //////////////////////
#include "mon.h"

// VARIATION A: break from loop, heap-allocate early
mon_t *mon_max_type(char *fname, char *filter){
  FILE *fin = fopen(fname,"r");
  if(fin == NULL){
    printf("Failed to open file '%s'\n",fname);
    return NULL;
  }
  mon_t *max_mon = malloc(sizeof(mon_t));
  max_mon->bp = -1;
  while(1){
    mon_t mon;
    int ret = fscanf(fin,"%s %d %s", mon.name, &mon.bp, mon.type);
    if(ret == EOF){
      break;
    }
    if(strcmp(filter,mon.type)==0 && mon.bp > max_mon->bp){
      max_mon->bp = mon.bp;
      strcpy(max_mon->name, mon.name);
      strcpy(max_mon->type, mon.type);
    }
  }
  fclose(fin);
  if(max_mon->bp == -1){
    printf("No matches for filter '%s'\n",filter);
    free(max_mon);
    return NULL;
  }
  return max_mon;
}

// VARIATION B: complex loop condition, heap-allocate late
mon_t *mon_max_typeB(char *fname, char *filter){
  FILE *fin = fopen(fname,"r");
  if(fin == NULL){
    printf("Failed to open file '%s'\n",fname);
    return NULL;
  }
  mon_t mon;
  mon_t max ={ .bp=-1 };
  while(fscanf(fin,"%s %d %s", mon.name, &mon.bp, mon.type)
        != EOF)
  {
    if(strcmp(filter,mon.type)==0 && mon.bp > max.bp){
      max.bp = mon.bp;
      strcpy(max.name, mon.name);
      strcpy(max.type, mon.type);
    }
  }
  fclose(fin);
  if(max.bp == -1){
    printf("No matches for filter '%s'\n",filter);
    return NULL;
  }
  mon_t *max_mon = malloc(sizeof(mon_t));
  max_mon->bp = max.bp;
  strcpy(max_mon->name, max.name);
  strcpy(max_mon->type, max.type);
  return max_mon;
}
```

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.

```c
#include <stdio.h>
void flub(double *ap, double *bp){
  int c = 7;
  if(*ap < c){
    *ap = bp[1];
  }
  // POSITION B
  return;
}
int main(){
  double x = 4.5;
  double arr[2] = {3.5, 5.5};
  double *ptr = arr+1;
  // POSITION A
  flub(&x, arr);
  printf("%.1f\n",x);
  for(int i=0; i<2; i++){
    printf("%.1f\n",arr[i]);
  }
  return 0;
}
```

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

```c
////////// SOLUTION ///////////////
#include <stdio.h>
#include <stdlib.h>

int *fill_pow2(int len){
  // malloc the array so it is on
  // the heap instead of stack
  int *arr = malloc(sizeof(int)*len);
  int pow = 1;
  for(int i=0; i<len; i++){
    arr[i] = pow;
    pow = pow * 2;
  }
  return arr;
}
int main(){
  int *twos4 = fill_pow2(4);
  for(int i=0; i<4; i++){
    printf("%d\n",twos4[i]);
  }
  free(twos4); // free now
  return 0;    // works fine
}
```

```
>> gcc -g fill_pow2.c

>> valgrind ./a.out
==6307== Memcheck, a memory error detector
==6307== Conditional jump or move depends on uninitialised value(s)
==6307==    by 0x48CB13B: printf (in /usr/lib/libc-2.29.so)
==6307==    by 0x10927B: main (fill_pow2.c:19)
1
0
0
0
==6307== Invalid free() / delete / delete[] / realloc()
==6307==    at 0x48399AB: free (vg_replace_malloc.c:530)
==6307==    by 0x109291: main (fill_pow2.c:21)
==6307==  Address 0x1fff000110 is on thread 1's stack
==6307==
==6307== HEAP SUMMARY:
==6307==     in use at exit: 0 bytes in 0 blocks
==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.*