

CMSC216: Practice Exam 1B SOLUTION

Spring 2024

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 `get_pn()`. Implement this function according to the documentation given. *My solution is about 18 lines plus some closing curly braces.*

```

1 #include "get_pn.h"
2
3 // Struct to count positive/negative
4 // numbers in arrays.
5 typedef struct {
6     int poss, negs;
7 } pn_t;
8
9 pn_t *get_pn(char *filename);
10 // Opens the specified filename which
11 // should contain space-separated ASCII
12 // integers. Allocates a pn_t and
13 // initializes its field to zero. Scans
14 // to end of the file incrementing the
15 // poss field for all positive numbers
16 // and the negs field for all 0 or
17 // negative numbers. Returns the
18 // allocated pn_t. If the file cannot
19 // be opened, returns NULL.
20
21 int main(){
22     // nums1.txt:
23     // 3 0 -12 76 -4
24     pn_t *pn1 = get_pn("nums1.txt");
25     // pn1: { .poss=2, .negs=3 }
26     free(pn1);
27
28     // nums2.txt:
29     // -1 -2 -4 0 -21 -35
30     pn_t *pn2 = get_pn("nums2.txt");
31     // pn2: { .poss=0, .negs=6 }
32     free(pn2);
33
34     pn_t *pn3 = get_pn("no-such-file.txt");
35     // pn3: NULL
36
37     return 0;
38 }

```

YOUR CODE HERE

```

8 // SOLUTION
9 pn_t *get_pn(char *filename){
10     FILE *fin = fopen(filename, "r");
11     if(fin == NULL){
12         return NULL;
13     }
14
15     pn_t *pn = malloc(sizeof(pn_t));
16     pn->negs = 0;
17     pn->poss = 0;
18     while(1){
19         int num;
20         int ret = fscanf(fin, "%d", &num);
21         if(ret == EOF){
22             break;
23         }
24         if(num > 0){
25             pn->poss++;
26         }
27         else{
28             pn->negs++;
29         }
30     }
31     fclose(fin);
32     return pn;
33 }

```

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

```

1 /////////////// SOLUTION /////////////////// 1 >> gcc -g fill_pow2.c
2 #include <stdio.h> 2
3 #include <stdlib.h> 3 >> valgrind ./a.out
4 4 ==6307== Memcheck, a memory error detector
5 int *fill_pow2(int len){ 5 ==6307== Conditional jump or move depends on uninitialised value(s)
6     // malloc the array so it is on 6 ==6307== by 0x48CB13B: printf (in /usr/lib/libc-2.29.so)
7     // the heap instead of stack 7 ==6307== by 0x10927B: main (fill_pow2.c:19)
8     int *arr = malloc(sizeof(int)*len); 8 1
9     int pow = 1; 9 0
10    for(int i=0; i<len; i++){ 10 0
11        arr[i] = pow; 11 0
12        pow = pow * 2; 12 ==6307== Invalid free() / delete / delete[] / realloc()
13    } 13 ==6307== at 0x48399AB: free (vg_replace_malloc.c:530)
14    return arr; 14 ==6307== by 0x109291: main (fill_pow2.c:21)
15 } 15 ==6307== Address 0x1fff000110 is on thread 1's stack
16 int main(){ 16 ==6307==
17     int *twos4 = fill_pow2(4); 17 ==6307== HEAP SUMMARY:
18     for(int i=0; i<4; i++){ 18 ==6307== in use at exit: 0 bytes in 0 blocks
19         printf("%d\n",twos4[i]); 19 ==6307== total heap usage: 0 allocs, 1 frees
20     }
21     free(twos4); // free now
22     return 0; // works fine
23 }

```

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.