Name: NumericID#: UID:

CMSC216: Practice Exam 1B

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.*

YOUR CODE HERE

```
#include "get_pn.h"

// Struct to count positive/negative
// numbers in arrays.
typedef struct {
  int poss, negs;
} pn_t;

pn_t *get_pn(char *filename);
// Opens the specified filename which
// should conatin space-separated ASCII
// integers. Allocates a pn_t and
// initializes its field to zero. Scans
// to end of the file incrementing the
// poss field for all positive numbers
// and the negs field for all 0 or
// negative numbers. Returns the
// allocated pn_t.  If the file cannot
// be opened, returns NULL.

int main(){
  // nums1.txt:
  // 3 0 -12 76 -4
  pn_t *pn1 = get_pn("nums1.txt");
  // pn1: {.poss=2, .negs=3}
  free(pn1);

  // nums2.txt:
  // -1 -2 -4 0 -21 -35
  pn_t *pn2 = get_pn("nums2.txt");
  // pn2: {.poss=0, .negs=6}
  free(pn2);

  pn_t *pn3 = get_pn("no-such-file.txt");
  // pn3: NULL

  return 0;
}
```

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
```