Name: ID#: X.500:

CMSC216: Practice Exam 1

Fall 2023

University of Maryland

Exam period: 20 minutes *Points available:* 50 *Weight:* 0% of final grade

Problem 1 (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 2 (10 pts): Fill in the following table of equivalent ways to write these 8 bit quantities. There are a total of 9 blanks to fill in and the first column indicates which blanks occur in which lines. Assume two's complement encoding for the signed decimal column.

Blank #s	Binary	Hex	Octal	Unsigned Decimal	Signed Decimal
#1 #2 #3	0001 1011		0033		
#4 #5 #6		0xA5	0245		
#7 #8 #9		0xC7			-57

Problem 3 (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 12 lines plus some closing curly braces.*

```
#include <stdio.h>
#include <stdlib.h>

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

pn_t *get_pn(int *arr, int len);
// Allocates a pn_t and initializes
// its field to zero. Then scans array
// arr increment poss for every 0 or
// positive value and negs for every
// negative value. Returns the pn_t
// with poss/negs fields set.  If arr
// is NULL or len is less than 0,
// returns NULL.

int main(){
  int arr1[5] = {3, 0, -1, 7, -4};
  pn_t *pn1 = get_pn(arr1, 5);
  // pn1: {.poss=3, .negs=2}
  free(pn1);

  int arr2[3] = {-1, -2, -4};
  pn_t *pn2 = get_pn(arr2, 3);
  // pn2: {.poss=0, .negs=3}
  free(pn2);

  int *arr3 = NULL;
  pn_t *pn3 = get_pn(arr3, -1);
  // pn3: NULL

  return 0;
}
```

Problem 4 (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
```