Name: NumericID#: UID:

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.*

```
#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;
}
```

YOUR CODE HERE

```
///// SOLUTION /////
pn_t *get_pn(char *filename){
  FILE *fin = fopen(filename,"r");
  if(fin == NULL){
    return NULL;
  }

  pn_t *pn = malloc(sizeof(pn_t));
  pn->negs = 0;
  pn->poss = 0;
  while(1){
    int num;
    int ret = fscanf(fin,"%d",&num);
    if(ret == EOF){
      break;
    }
    if(num > 0){
      pn->poss++;
    }
    else{
      pn->negs++;
    }
  }
  return pn;
}
```

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 in code.

```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.*