Name: ID#: X.500:

CSCI 2021: Practice Exam 1 SOLUTION

Spring 2023
University of Minnesota

Exam period: 20 minutes
Points available: 50

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.

```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 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.

```
|----------+-----------+------+-------+----------+---------|
| SOLUTION |           |      |       | Unsigned |  Signed |
| Blank #s | Binary    |  Hex | Octal |  Decimal | Decimal |
|----------+-----------+------+-------+----------+---------|
|          |           |      |       |          |         |
| #1 #2 #3 | 0001 1011 | 0x1B |  0033 |       27 |      27 |
|          |           |      |       |          |         |
| #4 #5 #6 | 1010 0101 | 0xA5 |  0245 |      165 |     -91 |
| ~x + 1   | 0101 1011 |      |       |          |         |
| #7 #8 #9 | 1100 0111 | 0xC7 |  0307 |      199 |     -57 |
| ~x + 1   | 0011 1001 |      |       |          |         |
|----------+-----------+------+-------+----------+---------|
NOTES
- Octal shows leading 0 which is not strictly necessary
- Typical twos' complement conversion technique show below
  binary representation: invert bits and add 1
```

Backgound: Write a short C code fragment (1-5 lines) using a C I/O function call to accomplish the stated task. Assume in each case there is a variable `FILE *fh` which has been opened appropriately for the I/O operation. Also assume that any variables mentioned have already been declared.

Problem 3 (5 pts): Read three text floating point values from `fh` formatted as standard decimal point values into `double` variables `r,u,t`. Check if the read fails due to reaching the end of file; if so print the message `None left`.

```
// SOLUTION
int res = fscanf(fh, "%lf %lf %lf", &r, &u, &t);
if(res == EOF){
  printf("None left\n");
}
```

Problem 4 (5 pts): Write 8 characters from the beginning of array `str` to `fh` in binary format.

```
// SOLUTION
fwrite(str, sizeof(char), 8, fh);
```

Problem 5 (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;
}
```

```
// SOLUTION
pn_t *get_pn(int *arr, int len){
  if(arr==NULL || len < 0){
    return NULL;
  }
  pn_t *pn = malloc(sizeof(pn_t));
  pn->negs = 0;
  pn->poss = 0;
  for(int i=0; i<len; i++){
    if(arr[i] < 0){
      pn->negs++;
    }
    else{
      pn->poss++;
    }
  }
  return pn;
}
```