ECE 551D
Fall 2018
Midterm Exam

Name: NetID:

There are 7 questions, with the point values as shown below. You have 75 minutes with a total of 75 points. Pace yourself accordingly.

This exam must be individual work. You may not collaborate with your fellow students. However, this exam is open notes, so you may use your class notes, which must be handwritten by you.

I certify that the work shown on this exam is my own work, and that I have neither given nor received improper assistance of any form in the completion of this work.

Signature:

#	**Question**	**Points Earned**	**Points Possible**
1	Concepts		10
2	Testing and Debugging		14
3	Reading Code		9
4	Coding Pictionary		10
5	Coding: Arrays		12
6	Coding: File IO		10
7	Coding: Put Together		10
	Total		75
	Percent		100

These two pages contain some reference (an abbreviated version of the man pages) for common, useful library functions.

- `char * strchr(const char *s, int c);`
  The `strchr` function locates the first occurrence of `c` (converted to a char) in the string pointed to by `s`. It returns a pointer to the located character, or `NULL` if the character does not appear in the string.

- `char * strdup(const char *s);`
  `char * strndup(const char *s, size_t n);`
  The `strdup` function allocates sufficient memory for a copy of the string `s`, does the copy, and returns a pointer to it. The pointer may subsequently be used as an argument to the function `free`. If insufficient memory is available, `NULL` is returned. `strndup` behaves the same way, except the length of the string to be copied is specified.

- `size_t strlen(const char *s);`
  The `strlen` function computes the length of the string `s` and returns the number of characters that precede the terminating `\0` character.

- `int strcmp(const char *s1, const char *s2);`
  `int strncmp(const char *s1, const char *s2, size_t n);`
  The `strcmp` and `strncmp` functions lexicographically compare the null-terminated strings `s1` and `s2`. The `strncmp` function compares not more than `n` characters. Because `strncmp` is designed for comparing strings rather than binary data, characters that appear after a `\0` character are not compared. These functions return an integer greater than, equal to, or less than 0, according as the string `s1` is greater than, equal to, or less than the string `s2`. The comparison is done using unsigned characters.

- `char *strstr(const char *s1, const char *s2);`
  The `strstr` function locates the first occurrence of the null-terminated string `s2` in the null-terminated string `s1`. If `s2` is an empty string, `s1` is returned; if `s2` occurs nowhere in `s1`, `NULL` is returned; otherwise a pointer to the first character of the first occurrence of `s2` is returned.

- `void * malloc(size_t size);`
  The `malloc` function allocates `size` bytes of memory and returns a pointer to the allocated memory.

- `void * realloc(void *ptr, size_t size);`
  The `realloc` function creates a new allocation of `size` bytes, copies as much of the old data pointed to by `ptr` as will fit to the new allocation, `free`s the old allocation, and returns a pointer to the allocated memory. If `ptr` is `NULL`, `realloc` is identical to a call to `malloc` for `size` bytes.

- `void free(void *ptr);`
  The `free` function deallocates the memory allocation pointed to by `ptr`. If `ptr` is a `NULL` pointer, no operation is performed.

- `int fgetc(FILE *stream);`
  The `fgetc` function obtains the next input character (if present) from the stream pointed at by `stream`, or `EOF` if `stream` is at end-of-file.

- `char * fgets(char *  str, int size, FILE *  stream);`
  The `fgets` function reads at most one less than the number of characters specified by `size` from the given `stream` and stores them in the string `str`. Reading stops when a newline character is found, at end-of-file or error. The newline, if any, is retained. If any characters are read and there is no error, a `\0` character is appended to end the string. Upon successful completion, it returns a pointer to the string. If end-of-file occurs before any characters are read, it returns `NULL`.

- `ssize_t getline(char ** linep, size_t * linecapp, FILE * stream);`
  The `getline` function reads a line from `stream`, which is ended by a newline character or end-of-file. If a newline character is read, it is included in the string. The caller may provide a pointer to a `malloc`ed buffer for the line in `*linep`, and the capacity of that buffer in `*linecapp`. These functions expand the buffer as needed, as if via `realloc`. If `linep` points to a `NULL` pointer, a new buffer will be allocated. In either case, `*linep` and `*linecapp` will be updated accordingly. This function returns the number of characters written to the string, excluding the terminating `\0` character. The value -1 is returned if an error occurs, or if end-of-file is reached.

- `void * memcpy(void * dst, const void *src, size_t n);`
  The `memcpy` function copies `n` bytes from memory area `src` to memory area `dst`.

Question 1 Concepts [10 pts]

For all parts of this question, assume that integers and floats occupy 4 bytes and doubles and pointers occupy 8 bytes.

1. Suppose you have a function `int f(int a[])`. How could you write code in `f` to find the number of elements in `a`?

   a) `sizeof(a)`

   b) `sizeof(a[0])`

   c) `int n = sizeof(a)/sizeof(a[0]);`

   d) `sizeof(*a);`

   e) None of the above

2. If `p` is declared as `double * p;`, and the numerical value of `p` is 0x2a73b9, then what is the numerical value of `p` + 4?

   a) 0x2a73bd

   b) 0x2a73c1

   c) 0x2a73c9

   d) 0x2a73d9

   e) None of the above

3. Consider the following declarations:

   ```
   int arr1[4];
   int arr2[4];
   int arr3[4];
   int * mat[3] = {arr1, arr2, arr3};
   ```

   Which one of the following is INCORRECT?

   a) `arr3` is a valid lvalue.

   b) `arr2[3]` is a valid rvalue.

   c) `mat` is a valid rvalue.

   d) `mat[0]` is a valid lvalue.

   e) None of the above

4. Suppose you have the following declaration:

```
const int * * const ptr;
```

After `ptr` is initialized to point at an appropriate data structure, and `something` is of appropriate type (for each answer option), which one of the following is legal?

a) `&ptr = something;`

b) `ptr = something;`

c) `*ptr = something;`

d) `**ptr = something;`

e) None of the above

5. Which one of the following accurately describes the difference between `perror(something)` and using `fprintf(stderr, something)`?

a) `perror` should only be used when you want to describe the value of `errno`, and `fprintf` should be used for other error messages.

b) `perror` should be used only for serious errors, and `fprintf` should be used for less severe errors.

c) `perror` should be used only for pointer-related errors, and `fprintf` should be used for all other errors.

d) `perror` should be used if you are not sure where `stderr` will be redirected, and `fprintf` should be used otherwise.

e) None of the above

Question 2 Testing and Debugging [14 pts]

1. Recall the "match5" testing assignment, where you had to write test cases for a "card game" with two hands of five "cards" each. One of the things you wanted to test was to make sure the program properly handles the error case when the second hand does not have enough cards. Which of the following is the BEST test case to check this behavior?

   a) aaaaaa bbbb

   b) aaaaaa bbb

   c) aaaaa bbbb

   d) aaaaa bbb

   e) None of the above

2. Recall the testing assignment where you had to write test cases for a program that would rotate a 10×10 matrix 90 degrees. Why was entering the character with decimal value 255 (0xFF in hexadecimal) a good test case?

   a) The program might not handle non-printable characters properly.

   b) 0xFF also is used for the End of File (EOF) character.

   c) The program might use an unsigned int instead of a signed int to hold the return value of `fgetc`.

   d) Sign-extending an 8-byte char with value 0xFF gives -1, which is also EOF.

   e) None of the above

3. Consider the following code, which you are debugging:

```
1 int a = 0;
2 for (int i = 0; i < 1000000; i++) {
3   a += getRandomNumber(i);
4 }
```

You hypothesize that something unexpected happens when `i` is 250000. What is the best way to use GDB to gather information about this hypothesis?

a) `display i`

b) `watch i`

c) `break 3`

d) `break 3 if i==250000`

e) None of the above

4. Suppose you needed to write test cases for the following function:

```
int isPalindrome(const char * s);
```

This function should return 1 if `s` is a palindrome and 0 otherwise.

A palindrome is a sequence of characters that is the same forwards and backwards. For example, "radar" *is* a palindrome because it has 'r' at the beginning and end, followed by 'a' as the second and second-to-last characters. The word "book" *is not* a palindrome because 'b' does not match 'k'.

You should write four test cases for this function **on the next page**. For each test case, you should fill in all fields on the testing worksheet we gave you.

Test Case 1
Description of Mistake:
Input:
Expected Behavior:
Behavior if Mistake Were Made:

Test Case 2
Description of Mistake:
Input:
Expected Behavior:
Behavior if Mistake Were Made:

Test Case 3
Description of Mistake:
Input:
Expected Behavior:
Behavior if Mistake Were Made:

Test Case 4
Description of Mistake:
Input:
Expected Behavior:
Behavior if Mistake Were Made:

Question 3 Reading Code [9 pts]

What is the output of the following code?

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

int f(int n, int *p, int ** q) {
  int temp[2];
  temp[0] = n + **q;
  temp[1] = p[1];
  *q = *q + 1;
  printf("%d : %d : %d\n", n, temp[0], temp[1]);
  if (n > 0) {
    int x = f(n-1, temp, q);
    *p = x + 2;
    printf("%d / %d / %d\n", n, temp[1], x);
    return temp[0] + x;
  }
  return 0;
}

int main(void) {
  int a[] = {3,5,7,9,12,44};
  int * p = a;
  int b = f(2,a, &p);
  printf("%d & %d & %d\n", a[0], a[1], b);
  return EXIT_SUCCESS;
}
```

Question 4 Coding Pictionary [10 pts]

Below is a fragment of code with certain lines (that you will reconstruct) removed:

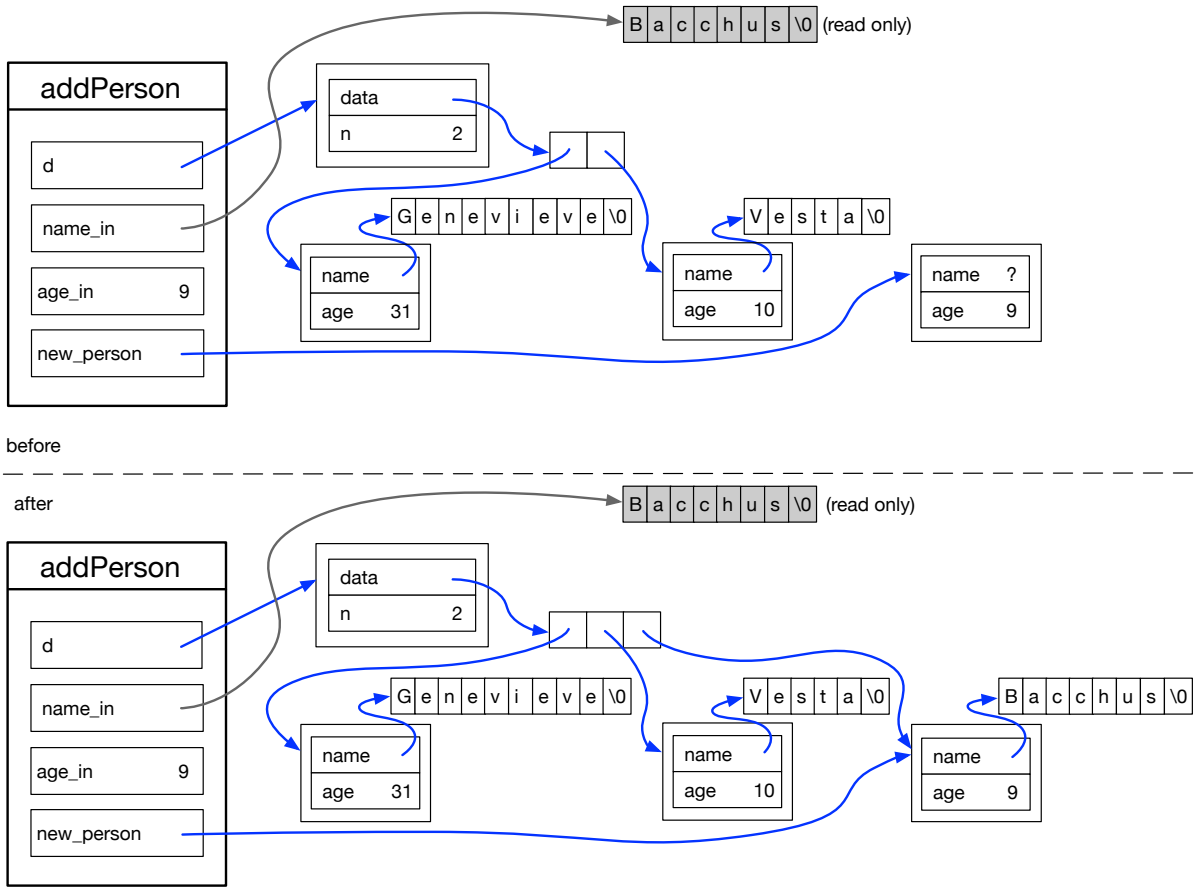
```
1  struct person_tag {
2    char * name;
3    int age;
4  };
5  typedef struct person_tag person_t;
6
7  struct personal_data_tag {
8    // struct fields omitted
9    // fill in for part 1
10 };
11 typedef struct personal_data_tag personal_data_t;
12
13 /* function takes a pointer to a personal_data_t 'd' and adds
14    a person_t with name 'name' and age 'age' */
15 void addPerson(personal_data_t * d, const char * name_in, int age_in) {
16   person_t * new_person = malloc(sizeof(*new_person));
17   new_person->age = age_in;
18   // 3 lines of code omitted
19   // fill in for part 2
20   //
21   d->n++;
22 }
```

To aid you in reconstructing the code, we have given you two diagrams of the program state (shown on the following page). The first shows the state of the program immediately prior to the execution of line 18, while the second shows the state immediately after the execution of line 20.

1. The struct `personal_data_t` has two missing lines (each declaring a field in that struct). What are these two missing lines?

   a)

   b)



2. The function `addPerson` is missing three lines of code. Lines 18–20 have the effect of transforming the "before" picture into the "after" state above. What are the missing lines? Note that you may use any C library functions you like.

   a)

   b)

   c)

Question 5 Coding: Arrays [12 pts]

Given the following struct declarations:

```
struct indpair_tag{
  size_t ind1;
  size_t ind2;
};
typedef struct indpair_tag indpair_t;

struct indarray_tag {
  indpair_t * pairs;
  size_t sz;
};
typedef struct indarray_tag indarray_t;
```

Write the function

```
indarray_t * findIndsThatMult(int * array, size_t n, int product);
```

This function should return a *pointer* to an `indarray_t` (allocated in the heap), which has the fields `pairs`, a pointer to an array of `indpair_t`s (also in the heap), and the number of elements of that array `sz`. An `indpair_t` represents a pair of indices, such that the corresponding elements of `array` (which has number of elements `n`) produce `product` when multiplied together. In these answers, `ind1 <= ind2` (they would be equal if and only if the element at `ind1` can be squared to get the desired product).

For example, if the function were called on the array `{2, 4, 12, 6, 4}`, and `product = 24`, it should return an array with three elements. One of these should have `ind1 = 0` and `ind2 = 2`, since `array[0] * array[2] = 24`. Another should have `ind1 = 1` and `ind2 = 3`, since `array[1] * array[3] = 24`. Likewise, the last element should have `ind1 = 3` and `ind2 = 4`.

Write your answer on the next page.

```
indarray_t * findIndsThatMult(int * array, size_t n, int product) {









}
```

Question 6 Coding: File IO [10 pts]

Given the following struct declaration:

```
struct array_tag {
  int * arr;
  size_t n;
};
typedef struct array_tag array_t;
```

Write the function

```
array_t arrayFromFile(FILE * f);
```

This function should return an `array_t`, which has a pointer to an array of `int`s and the number of elements in that array `n`.

You may assume that the input file contains one number per line, which can be represented as an `int` type. This function should not leak memory.

```
array_t arrayFromFile(FILE * f) {









}
```

Question 7 Coding: Put Together [10 pts]

Now write a complete program, which should put these pieces of code together. This program should take two command line arguments, the name of the file to read and a number X. The program should read the contents of the file (which are one number per line, as in the previous problem). It should then find each pair of numbers whose product is X, and print their sum. For example, if X is 24, and the input file contains:

```
2
4
12
6
4
```

Your program should print

```
14
10
10
```

(because $2 + 12 = 14$, $4 + 6 = 10$, and $6 + 4 = 10$).

You may assume the following things in doing this problem:

- The correct number of command line arguments are given to `main`.
- The first command line argument names a valid file, containing one number per line.
- The second command line argument is a valid number.
- Calls to `fopen` and `fclose` succeed.
- The structs `indpair_t`, `indarray_t`, and `array_t` and prototypes for `findIndsThatMult` and `arrayFromFile` are declared in a header file called `exam.h`.

For full credit, you MUST do the following:

- Use `findIndsThatMult` and `arrayFromFile` from the previous two problems to accomplish the relevant parts of this task, (i.e. do NOT duplicate code).
- `free` all appropriate memory and close all open files.
- Include any header files that you need.

Write your answer on the next page.

Answer for Question 7 Coding: Put Together