CS152 Midterm

Nick Ding

TOTAL POINTS

65 / 100

QUESTION 1

1 Empty String vs Empty Array 3 / 3

✓ **- 0 pts Claimed**

✓ **- 0 pts Correct**

- 3 pts Blank

- 3 pts Incorrect

- 2 pts Incorrect explanation

QUESTION 2

2 Segmentation Fault 4 / 4

✓ **- 0 pts Claimed**

✓ **- 0 pts Correct**

- 2 pts First incorrect

- 2 pts Second incorrect

QUESTION 3

3 Free Heap Memory 3 / 3

✓ **- 0 pts Claimed**

✓ **- 0 pts Correct**

- 3 pts Blank

- 3 pts Incorrect

- 2 pts Incorrect/no explanation

QUESTION 4

4 Hexadecimal 2 / 2

- 0 pts Claimed

✓ **- 0 pts Correct**

- 2 pts Blank / major mistake

- 1 pts Minor mistake

QUESTION 5

5 flight_new 3 / 5

- 0 pts Claimed

- 0 pts Correct

- 5 pts Incomplete

- 3 pts Correctly constructs a flight struct and then returns the address of that struct. You need allocate memory in order for the struct to be persistent.

✓ **- 2 pts Doesn't allocate memory for a geo struct for src and dst**

- 2 pts Doesn't assign slng, slat, dlng, dlat to allocated src and dst

- 2 pts Doesn't allocate memory for a flight struct

- 2 pts Doesn't assign allocated src and dst to allocated flight

- 1 pts Missing return statement

- 1 pts Dereferences something that doesn't need to be dereferenced. Eg, writes flight->src = *src instead of flight->src = src. Or conversely, flight->*src = src.

- 1 pts Doesn't dereference struct item when assigning to it. Eg, flight.src = src instead of flight->src = src or (*flight).src = src.

- 1 pts Incorrect treatment of nested structs. Eg, writes flight->lng = slng, etc. instead of flight->src->lng = slng.

- 1 pts Incorrect usage of ampersand. Eg, filght->src = &src instead of flight->src = src, where src is a pointer to a geo_t struct that has been allocated with malloc.

- 1 pts Incorrect syntax for assigning items to a struct

- 1 pts Doubly de-references a struct pointer. Eg. (*flight)->src = src instead of flight->src = src or (*flight).src = src.

- 0 pts You should use malloc(sizeof(struct geo)) instead of sizeof(2*double) because for some types of structs padding will be added.

QUESTION 6

6 departed_same_airport 0 / 5

✓ **- 0 pts Claimed**

- 0 pts Correct

✓ **- 5 pts Blank/Incorrect**

- 2.5 pts Incorrect distance calculation (Note: measuring euclidean distance between lat/long and comparing to tol is ok)

- 3 pts Not comparing location of airport a to airport b

- 1 pts Forgot to square/sqrt when doing distance calculation

- 2 pts Incorrect method of accessing a struct pointer

- 4 pts Does not incorporate tol value

- 1 pts Uses destination lat/lng

- 0 pts Does not use proper syntax for power/sqr/abs, etc

- 0 pts Does not multiply tot (by 2 or 4 depending on distance calculation)

QUESTION 7

7 flight_free 4 / 4

✓ **- 0 pts Claim**

✓ **- 0 pts Correct**

- 2 pts did not free f

- 2 pts did not free f->src and f->dst

- 3 pts only freeing non-pointer variables (e.g. free(f->src->lng))

- 2 pts Freeing non-pointer variables, like the members of struct geo

- 1 pts "free" being called in the object rather than the pointer.

- 4 pts no answer/incomplete/incorrect

QUESTION 8

8 square_values 4 / 6

- 0 pts Correct

✓ **- 0 pts Claimed**

- 1 pts Missing the "default" branch in the switch for the error condition.

- 3 pts Squaring is not in-place.

✓ **- 2 pts Wrong syntax for accessing struct/union members.**

- 6 pts Answer missing, or completely ignoring the tag.

- 1 pts Incorrect syntax for accessing tag

- 0.5 pts Missing "break" statement in the "switch".

QUESTION 9

9 Excerpts: what does code print? 3 / 8

✓ **+ 0 pts Claimed**

✓ **+ 2 pts Part I Correct: 54 55 55**

+ 1 pts Part I partially correct (at least two correct values)

+ 2 pts Part II Correct: 0 1 3 2 1

✓ **+ 1 pts Part II Partially Correct or Extra Output: 0 1 3 2 1 1**

+ 2 pts Part III Correct: WXA

+ 1 pts Part III Partially Correct

+ 2 pts Part IV Correct: 121 6c 144

+ 1 pts Part IV Partially Correct

54 55 55
0 1 3 2 1
WXA
121 6c 144

QUESTION 10

10 Find Bugs (Part I) 0.5 / 8

✓ **- 0 pts Claimed**

- 3 pts Missed: f(*a) should be f(&a)

- 1 pts Non-bug reported as bug in f(...) or main(...)

✓ **- 4 pts Completely incorrect analysis of f(...) and main(...)**

✓ **- 2 pts Missed: i < alen/2 - 1 should be i < alen/2 (for statement)**

- 1 pts Missed: a[i] = a[alen-i-1] (instead of a[alen-i])

- 1 pts Missed: a[alen-i-1] = t (instead of a[alen-i] = t)

✓ **- 1 pts Non-bug reported as bug in reverse(...)**

- 4 pts Completely incorrect analysis of reverse(...)

- 0.5 Point adjustment

Need to propose a fix for a[alen-i], in addition to identifying the bug. Declaring int t within the loop is not a bug and is actually stylistically preferable.

QUESTION 11

11 Find Bugs (Part II - subarray) 1.5 / 4

✓ **- 0 pts claim**

- 0 pts Correct

- 2 pts didn't find malloc bug

✓ **- 2 pts didn't find indexing bug**

- 1 pts did not correct malloc bug

- 1 pts did not correct indexing bug

- 0.5 Point adjustment

int * is not a problem

QUESTION 12

12 Find Bugs (Part III - histogram) 0 / 4

✓ **- 0 pts Claim**

✓ **- 2 pts Missing: int* result = (int*)malloc(256*sizeof(int));**

✓ **- 2 pts Missing: while(s) should be while(*s)**

- 1 pts Non-bug reported as bug

- 4 pts Completely incorrect analysis

QUESTION 13

13 Base conversions 11 / 12

✓ **- 0 pts Claimed**

- 1 Point adjustment

QUESTION 14

14 interleave 5 / 8

✓ **- 0 pts Claimed**

- 1 pts Not accounting for terminating character in allocation

- 8 pts Not Answered

✓ **- 3 pts Missing string length computation**

- 2 pts Missing null termination

- 1 pts Missing allocation for null terminator

QUESTION 15

15 local_average 7 / 8

✓ **- 0 pts Claimed**

- 3 pts Before finding local average and changing array in place, must store previously existing value (otherwise averages change)

✓ **- 1 pts Index out of bounds error (i.e. program breaks at alen = 1 because tried to access a[1])**

- 1 pts Unnecessary use of variables

- 1 pts Incorrect accessing of array elements

- 2 pts Not averaging either first or last element correctly

- 1 pts No return values for this function

- 5 pts Misinterpretation of function purpose (i.e. averaging current value and everything before instead of neighbors)

- 1 pts Error case when should not error (i.e. erring when alen = 2 which is a valid case)

- 1 pts If using a stack allocated array to store old values, must initiate it with alen space

- 1 pts Missing proper conditionals

- 2 pts Incorrect storage of old values (but attempted)

- 8 pts Blank Answer

- 1 pts Not averaging middle values correctly

QUESTION 16

16 find_first_substring 8 / 8

✓ **- 0 pts Claimed**

- 2 pts Does not abort inner loop when finding a mismatch

- 2 pts Index out of bounds (i.e. due to failing to reset counter variables)

- 2 pts Doesn't check entirety of substring t and portion of s for match

- 3 pts Missing proper conditionals

- 1 pts Improper dereferencing

- 1 pts Doesn't return length of s when no match

- 1 pts Improper match condition (i.e. index at match should be equal to strlen(t) - 1)

- 3 pts Missing inner loop when first characters of s and t match

- 2 pts Does not return when match (i.e. lack of return causes matches to be ignored)

- 8 pts Blank/Almost entirely incomplete answer

- 2 pts Should not by default, return strlen(s) when no match (matches may appear later)

- 2 pts Missing updating counter of iterations

- 6 pts Missing large majority of the answer but

some correct intuition

QUESTION 17

17 replace_first_substring 6 / 8

- **0 pts** claimed
- **0 pts** Correct
- **8 pts** Incomplete/incorrect
- **3 pts** Incorrect for the case where a != ""

✓ **- 2 pts Incorrect on case where a is not a substring of s**

- **2 pts** You can't concatenate strings with "+" in C
- **2 pts** Incorrect indexing
- **2 pts** Your code overwrites part of S
- **1 pts** You should malloc strlen(s) - strlen(a) + strlen(b) + 1 for the case where find_first_index(s,a) != strlen(s)
- **1 pts** Doesn't add null terminator
- **1 pts** Doesn't add the part of s after the place where the first substring occurs
- **1 pts** Missing return statement
- **1 pts** You must use malloc to allocate memory
- **1 pts** You're not removing the a substring
- **1 pts** Doesn't increment index in while loops
- **0 pts** Make variables for strlen(s), strlen(a), strlen(b), find_first_substring(s,a) to avoid repeated code and execution
- **0 pts** You don't need a special case for if a is the empty string/find_first_substring(s,a) == 0
- **0 pts** Equality comparison should be ==, not =
- **0 pts** Click here to replace this description.
- **1 pts** Missing/incorrect for case of a = ""
- **2 pts** Only malloc necessary space for string

CMSC 15200: Introduction to Computer Science II

The University of Chicago, Spring 2019

Midterm Examination

Monday, April 29, 2019

Please write your name here:

Nick Ding

This exam consists of 16 numbered pages, including this one.
In total, all problems are worth 100 points.

Write helper functions as you see fit. You may use any function you write anywhere on the exam.

Please note that we will not grade anything written on scratch paper. All your responses must appear on this copy of the test itself.

Unless a problem states specific restrictions on calling library functions, you may assume that common library functions from `stdio.h`, `stdlib.h`, `string.h`, `math.h` *etc.* are available; you do not have to write any `#include` directives. You may assume that `malloc` succeeds; you don't need to check if the result of `malloc` is `NULL`.

Type specifications of the forms `int* p` and `int *p` are both correct and mean the same thing. Both styles are used in the text of this exam.

You may not ask clarifying questions during the test; read the text of the test, including these instructions, closely and literally, and respond as best you can. The group of test-takers is too large to allow questions, and it's unfair for certain students to interact with instructors while others don't.

ND

Is an empty string an example of an empty array? If not, how are they different? *[3 points]*

No, since an empty string still contains the special character '\0'.

Describe two different kinds of programming mistakes that might lead to a `Segmentation Fault` at runtime. *[4 points]*

One error would be exceeding an amount of allocated memory

Another error would be infinite recursion

Indicate whether the following statement is true or false and explain why:
A function that allocates memory on the heap should free that memory before it returns. *[3 points]*

False, if you free the memory before returning it then you will return nothing.

Write down the hexadecimal representation of the following binary number *[2 points]*: 1010100101001111010101111011

A93F2EB

ND

A geo consists of a longitude and a latitude, and a flight represents a flight from one location to another, as follows:

```
struct geo {
   double lng;
   double lat;
};
typedef struct geo geo_t;
```

```
struct flight {
   geo_t *src;
   geo_t *dst;
};
typedef struct flight flight_t;
```

Write a constructor `flight_new` to build a fully heap-allocated object from longitudes and latitudes, as follows *[5 points]*:

```
flight_t* flight_new(double slng, double slat, double dlng, double dlat){
  flight_t* new = (flight_t*) malloc(sizeof(flight_t));
  new->src->lng = slng;
  new->src->lat = slat;
  new->dst->lng = dlng;
  new->dst->lat = dlat;
  return new;
}
```

Consider only flights departing from or arriving at airports in the continental United States (far from the poles or prime meridian). Let us assume that each arriving or departing flight has its location measured to be within a distance `tol` from the longitude and latitude defining the center of the airport. Write a function to determine (true or false) whether flight `fa` and flight `fb` departed from the same airport. *[5 points]*

```
int departed_same_airport(flight_t* fa, flight_t* fb, double tol)
```

Write a function `flight_free` to free a flight struct and the structs it points to. *[4 points]*

```
void flight_free(struct flight *f){
    free(f->src);
    free(f->dst);
    free(f);
}
```

```
enum prec_tag { FLOAT, DOUBLE };

union num {
   float f;
   double d;
};

struct tagged_num {
   enum prec_tag tag;
   union num n;
};

typedef struct tagged_num tagged_num_t;
```

This definition of `union num` permits storing floating point values of 32-bit precision (`float`) or 64-bit precision (`double`). Write a function that squares each value in an array of such numbers, preserving the original precision. *[6 points]*

```
void square_values(tagged_num_t a[], unsigned int alen)
```

```
unsigned int i;
float replacef;
double replaced;
for (i=0; i< alen; i++){
    switch( a[i].tag ){
        case FLOAT:
            replacef = a[i] * a[i];
            a[i] = tagged_num_t(FLOAT, replacef);
            break;
        case DOUBLE:
            replaced = a[i] * a[i];
            a[i] = tagged_num_t(DOUBLE, replaced);
            break;
        default:
            fprintf(stderr, "invalid tag");
            break;
    }
}
```

ND

Each of the code excerpts on this page and the next prints something. To the right of each excerpt, write, or, if necessary, describe, what it prints. You do not need to write newlines explicitly. *[2 points each]*

```
int i = 54;
int *q = &i;
int *p = q;
printf("%d\n",i++);
printf("%d\n",(*q)++);
printf("%d\n",--(*p));
```

54
55
55

```
          0 1 2 3 4 5
int a[] = {1, 1, 2, 3, 5, 8};
int i;
for (i = 5; i > 0; i--)
   printf("%d\n", a[i]%4);
```

1
1
3
2
1

```
void foo(int* a, int alen, char* s) {
   while (alen) {
      *s = 'Z' - *a;
      a++;
      s++;
      alen--;
   }
}

int main(int argc, char** argv) {
   int a[] = {3, 2, 25, 'Z'};
   char s[5];
   foo(a, 4, s);
   printf(s);
   return 0;
}
```

prints array of int value from Z with subtraction of each array value

---

```
int y1 = 100;      /* base 10 value */
int y2 = 0154;     /* base 8 value */
int y3 = 0x79;     /* base 16 value */
printf("Decimal: %d\n", y3);
printf("Hexadecimal: %x\n", y2);
printf("Octal: %o\n", y1);
```

value of y3 in decimal
value of y2 in hex
value of y1 in Octal

ND

The following code excerpts may contain one or more errors, including memory management errors. Briefly explain all errors and how to correct them, or indicate that there are no errors. Note that in no case does the code contain simple syntax errors such as missing braces or semicolons. *[4 points each]*

```
void f(int *a)
{
  *a = (*a) * (*a) * (*a);
}

int main(int argc, char *argv[])
{
   int a = 15;
   printf("Cube of side length %d", a);
   f(*a);
   printf(" has volume %d\n", a);
}
```

Will print no value, as supposed to print address

```
/* Reverse the items in the array a in place. */
void reverse(int* a, int alen) {
   int i;
   for (i = 0; i < alen/2 - 1; i++) {
      int t = a[i];
      a[i] = a[alen-i];
      a[alen-i] = t;
   }
}
```

alen = 6

1 2 3 4 5 6

a[alen - i] is outside of memory when i=0.

Creating a new variable int t at every pass in the for loop.

ND

```
/*
   Return a copy of the subarray whose first element is a[start], last
   element is a[end], and which also contains all elements between.
*/
int *subarray(int* a, unsigned int start, unsigned int end) {
   if (start > end) {
      return NULL;
   }
   int* b = (int*)malloc(end - start);
   for (unsigned int i = start; i <= end; i++) {
      b[i] = a[i];
   }
   return b;
}
```

- Declaration line should be int*_ not int _* (_ = space)
- malloc should contain (sizeof(int) *), not just (end - start)
- Doesn't pass in the length of the original array. Start and end parameters could be outside the range/length of a.

ND

```
/*
   Histogram: Compute a table which counts the number of times each
   character occurs in a string.
*/
int* histogram(char* s)
{
   int i;
   int result[256];
   for (i = 0; i < 256; i++)
      result[i] = 0;
   while (s)
   {
      unsigned char bin = *s++;
      result[bin]++;
   }
   return result;
}
```

Segmentation fault,

Infinite recursion,

Mem allocation

Each row in this table corresponds to the same number, and each column indicates a different representation of that number. Convert between the specified bases to fill in all the blanks. *[1 point each]*

binary (unsigned)	octal	decimal	hexadecimal
11011011	333	219	DB
111100	74	60	46
1110100	164	116	74
11111111	177	127	7F

'abc' 3 'wxyz' 4 outlen-1 7 out: a w b x c y z counter: 0 1 2 3 4 5 6 7

(10)

Write a function **interleave** to construct a new heap-allocated string containing the characters of its two argument strings mixed together in an alternating fashion. For example, calling **interleave** on **"abc"** and **"xyz"** builds a new string on the heap containing **"axbycz"** (and returns a pointer to it). If one string is shorter, continue outputting the other. Calling it on **"abc"** and **"wxyz"** returns **"awbxcyz"**. Your implementation should call **malloc** only once, and should not call any other library functions. *[8 points]*

char* interleave(char* s, char* t)

```
unsigned short outlen = strlen(s) + strlen(t) + 1;
char* out = (char*) malloc(sizeof(char) * outlen);
unsigned short i = 0, j = 0, counter = 0;
while (counter < outlen - 1) {
    if (s[i] != '\0') {
        out[counter] = s[i];
        i++;
        counter++;
    }
    if (t[j] != '\0') {
        out[counter] = t[j];
        j++;
        counter++;
    }
}
out[counter] = '\0';
return out;
}
```

Write a function `local_average` that modifies an array in place, replacing each value with the numerical average of itself and the values immediately before and after it in the array. For example, given the input array:

```
-1.00  2.50  4.50  -2.00  -9.00   3.00
```

After a call to `local_average` this array should contain:

```
 0.75  2.00  1.67  -2.17  -2.67  -3.00
```

Note that for the first and last element, taking an average of two, rather than three, values is required.

Your code must modify the array in place. Furthermore, your code may not dynamically allocate memory (calling `malloc` is forbidden). *[8 points]*

`void local_average(float* a, unsigned int alen)`

```
float avg, temp;
unsigned int counter;

for(counter = 0; counter < alen; counter++){
  if(counter == 0){
    temp = a[counter];
    avg = (a[counter] + a[counter+1]) / 2;
    a[counter] = avg;
  } else if(counter == alen-1){
    avg = (temp + a[counter]) / 2;
    a[counter] = avg;
  } else {
    avg = (temp + a[counter] + a[counter+1]) / 3;
    temp = a[counter];
    a[counter] = avg;
  }
}
}
```

This page contains only problem instructions. Your implementation will be written on the subsequent pages.

For both parts of this problem, your code may call `strlen` but should not call any other library functions from `string.h`.

Part I.

Write a function `find_first_substring` that, given strings `s` and `t`, returns the starting position of the first occurrence of substring `t` within `s`. Positions within `s` are numbered from 0 through `(strlen(s)-1)`. If no substring of `s` matches `t`, then return `strlen(s)`. *[8 points]*

Examples:

`find_first_substring("abcdef","cde")` returns 2
`find_first_substring("abcdef","efg")` returns 6 (no match).
`find_first_substring("abcdef","")` returns 0

In the last example, note that the empty string is always a substring of any other string.

Part II.

For this part, you may assume your implementation of `find_first_substring` is correct, and use it as a helper function.

Write a function `replace_first_substring` that, given strings `s`, `a`, and `b`, replaces the first occurrence of substring `a` within `s` with substring `b`, returning the result as a newly created string. Do not modify `s`. *[8 points]*

Examples:

`replace_first_substring("abcdef","cde","wxyz")` returns `"abwxyzf"`
Note that the returned string is longer.

`replace_first_substring("abcdef","efg","wxyz")` returns `"abcdef"`
No replacement is done when there is no match.

`replace_first_substring("abcdef","","wxyz")` returns `"wxyzabcdef"`
Replacing the first match of the empty string is equivalent to prepending the replacement string.

unsigned int find_first_substring(char* s, char* t)

```
unsigned int i, j, counter = 0, itemp = 0;

for(i=0; s[i] != '\0'; i++){
    if(s[i] == t[0]){
        itemp = i+1;
        for(j=1; t[j] != '\0'; j++){
            itemp++;
            if(s[itemp] == t[j]){
                counter++;
            }
        }
        if(counter == strlen(t) - 1){
            return i;
        }
        counter = 0;
    }
}
return 0;
}

/* please put following code where
 * the arrow points! */

if( strlen(s) == 0 || strlen(t) == 0){
    return 0;
}
```

```
char* replace_first_substring(char* s, char* a, char* b){
    unsigned int newlen = strlen(s) - strlen(a) + strlen(b) + 1;
    unsigned int i=0, j=0, counter=0
    char* res = (char*)malloc(sizeof(char) * newlen);
    unsigned int insertat = find_first_substring(s, a);
    while(i < insertat){
        res[counter] = s[i];
        i++;
        counter++;
    }
    while(b[j] != '\0'){
        res[counter] = b[j];
        j++;
        counter++;
    }
    while(s[i] != '\0'){
        res[counter] = s[i];
        i++;
        counter++;
    }
    res[counter] = '\0';
    return res;
}
```