

ECE551  
Midterm Version 2

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. You may not use any external resources.

**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		7
2	Reading Code		4
3	Fixing Valgrind Errors		10
4	Coding 1: Recursion		10
5	Coding 2: Strings		14
6	Coding 3: Arrays		12
7	Coding 4: A Program		18
	Total		75
	Percent		100

The next two pages contains 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 *s1);`  
The `strdup` function allocates sufficient memory for a copy of the string `s1`, 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.
- `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, frees 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, delimited by the character delimiter. 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 `malloced` 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 [7 pts]

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

1. If you declare an array `float myArray[2][3];`, what is the value of `sizeof(myArray[0])`?
  - (a) 4 bytes
  - (b) 8 bytes
  - (c) 12 bytes
  - (d) 24 bytes
  - (e) None of the above
2. Suppose I declare a pointer `int * *const * q;`. The `const` makes which of the following read only?
  - (a) `q`
  - (b) `*q`
  - (c) `**q`
  - (d) `***q`
  - (e) None of the above
3. In image processing, you can do math on image pixels. This is an example of:
  - (a) The “Everything is a number” principle
  - (b) Abstraction
  - (c) The seven item limit
  - (d) Top-down design
  - (e) None of the above
4. What does `assert(expr);` do?
  - (a) Makes `expr` evaluate to true
  - (b) Sets the boolean variable `expr` to true (1).
  - (c) Causes the compiler to check that `expr` is true, and produce an error otherwise.
  - (d) At runtime, check if `expr` is true, and if not, abort the program.
  - (e) None of the above
5. What does `aliasing` mean?
  - (a) A TV show starring Jennifer Garner.

- (b) When you create a pointer to a pointer.
  - (c) When you copy data from the stack to the heap.
  - (d) When there are two names for the same box.
  - (e) None of the above
6. If you are using gdb to troubleshoot a program with a for loop in the function `printData`, and you would like to stop execution when `i` is 100, how would you do this?
- (a) `break`
  - (b) `break 100`
  - (c) `break if i == 100`
  - (d) `break printData`
  - (e) None of the above
7. If your project lead asks you to write a program that uses a Monte Carlo simulation to design the yield strength of a cantilevered beam and you do not know what a Monte Carlo simulation or yield strength are, what should you do?
- (a) Work Steps 1 and 2 on different examples
  - (b) Find a source of domain knowledge
  - (c) Ask your project lead to further specify the problem
  - (d) Think of a corner case to test
  - (e) None of the above

## Question 2 Reading Code [4 pts]

What is the output of the following code?

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

void f (int v, int * p, int ** q) {
    v = v % 2;
    int temp = (v+1) % 2;
    p[v] = p[temp];
    p = q[v];
    q[v] = q[temp];
    q[temp] = p;
}

int main(void) {
    int * data[2];
    for (int i =0 ; i < 2; i++) {
        data[i] = malloc(2 * sizeof(*data[i]));
        for (int j = 0; j < 2; j++) {
            data[i][j] = i*2+j;
        }
    }
    f(data[1][0], data[0], data);
    for (int i =0 ; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            printf("data[%d][%d]=%d\n", i, j,data[i][j]);
        }
        free(data[i]);
    }
    return EXIT_SUCCESS;
}
```

## Question 3 Fixing Valgrind Errors [10 pts]

A C programmer wrote the following buggy code to read in a file, reverse the order of the lines, and then print the lines in that reversed order.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(void) {
5     char * line = NULL;
6     size_t sz = 0;
7     char ** allstrings=NULL;
8     size_t n =0;
9     while(getline(&line, &sz, stdin)>=0) {
10         allstrings = realloc(allstrings, (n+1)*sizeof(**allstrings));
11         allstrings[n] = line;
12         n++;
13     }
14     free(line);
15     for (size_t i = 0; i < n/2; i++) {
16         size_t j = n-i;
17         char * temp = allstrings[i];
18         allstrings[i] = allstrings[j];
19         allstrings[j] = temp;
20     }
21     for (size_t i= 0; i < n; i++) {
22         printf("%s", allstrings[i]);
23     }
24     free(allstrings);
25     return EXIT_SUCCESS;
26 }
```

**Error 1** The first problem is:

Invalid write of size 8

at 0x400711: main (broken.c:11)

Address 0x5205140 is 0 bytes inside a block of size 1 alloc'd

at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload\_memcheck-amd64-linux.so)  
by 0x4C2FDEF: realloc (in /usr/lib/valgrind/vgpreload\_memcheck-amd64-linux.so)  
by 0x4006F5: main (broken.c:10)

This problem can be correctly fixed by:

- A. Changing the first argument of `realloc` on line 10 to `&allstrings`, instead of `allstrings`.
- B. Changing the operand of `sizeof` on line 10 to `*allstrings` instead of `**allstrings`.
- C. Changing line 7 to initialize `allstrings` with an appropriate call to `malloc`, instead of with `NULL`.
- D. Changing the first argument of `realloc` on line 10 to `*allstrings`, instead of `allstrings`.
- E. None of the above—specify what to do instead:

---

**Error 2** After correctly fixing the first problem, the next error is:

Invalid read of size 8

at 0x4007A2: main (broken.c:18)

Address 0x52062e0 is 0 bytes after a block of size 208 alloc'd

at 0x4C2FD5F: realloc (in /usr/lib/valgrind/vgpreload\_memcheck-amd64-linux.so)  
by 0x4006FD: main (broken.c:10)

This problem can be correctly fixed by:

- A. Changing the initialization of `j` on line 16 from `n-i` to `n-i-1`.
- B. Adding `if (j >= n) {break;}` between lines 16 and 17.
- C. Changing the loop condition of the for loop on line 15 from `i < n/2` to `i < n/2-1`
- D. Changing the initialization of `j` on line 16 from `n-i` to `n-i+1`.
- E. None of the above—specify what to do instead:



**Error 3** After correctly fixing the first two problems, the next error is:

```
Invalid read of size 1
  at 0x4E88CC0: vfprintf (vfprintf.c:1632)
  by 0x4E8F898: printf (printf.c:33)
  by 0x40080D: main (broken.c:22)
Address 0x5204040 is 0 bytes inside a block of size 120 free'd
  at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
  by 0x40074B: main (broken.c:14)
Block was alloc'd at
  at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
  by 0x4EA89E7: getdelim (iogetdelim.c:62)
  by 0x40073A: main (broken.c:9)
```

This problem can be correctly fixed by:

- A. Inserting `line = strdup("");` between lines 12 and 13
- B. Inserting `line = NULL` between lines 12 and 13
- C. Removing line 14 (`free(line);`)
- D. Changing line 14 to `if (n==0) { free(line); }`
- E. None of the above—specify what to do instead:

---

**Error 4** After correctly fixing the first three errors, the code runs, but leaks memory:

```
3,120 bytes in 26 blocks are definitely lost in loss record 1 of 1
  at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
  by 0x4EA89E7: getdelim (iogetdelim.c:62)
  by 0x400742: main (broken.c:9)
```

This error can be corrected by:

- A. Inserting `free(allstrings[i]);` between lines 22 and 23.
- B. Inserting `free(allstrings[n]);` between line 14 and 15.
- C. Inserting `free(allstrings[j]);` between lines 19 and 20.
- D. Inserting `free(temp);` between lines 19 and 20.
- E. None of the above—specify what to do instead:

## Question 4 Coding 1: Recursion [10 pts]

For this problem, you will write a function to count the number of even elements in an array of integers (where `n` is the size of the array) using both iteration, and then tail recursion. First, write this using iteration (`countEvIter`), then implement an equivalent function using tail recursion (`countEvTail`). For the tail recursive function, you should write a helper in the space indicated.

```
int countEvIter(int * toCount, size_t n) {  
//YOUR CODE GOES HERE: Iterative
```

```
}
```

```
//Write a helper function here
```

```
int countEvTail(int * toCount, size_t n) {  
//YOUR CODE GOES HERE: Tail Recursive
```

```
}
```

## Question 5 Coding 2: Strings [14 pts]

For this question, you will implement the `strspn` function, as it behaves in the C library. You MAY NOT USE ANY C LIBRARY FUNCTIONS to implement this. The `strspn` function computes the number of characters from `s1` which are in the string `s2`. Put another way, it returns the index in `s1` of the first character which does not appear in `s2`. If all characters in `s1` appear in `s2`, `strspn` returns the index of the null-terminator in `s1`. For example `strspn("abcde", "abd")` would return 2 because 'c' is not in "abd". You may assume that neither `s1` nor `s2` is NULL.

```
size_t strspn(const char *s1, const char *s2) {  
    //YOUR CODE GOES HERE
```

```
}
```

## Question 6 Coding 3: Arrays [12 pts]

For this problem, you are going to write `findDiagonalWord`, which takes a `const char * const * letters`, two `size_ts`, `r` which specifies how many rows are in `letters`, and `c`, which specifies how many columns in `letters`, and a `const char * word`. This function finds the location of `word` in `letters` on a left-top to right-bottom diagonal. It returns the row and column of the first occurrence that it finds as a `loc_t`, as declared below. If `word` does not occur on a left-top to right-bottom diagonal, this function returns a `loc_t` with both values set to `(size_t)-1` (that is, `-1` cast as `size_t`). For example, if `letters` were the ones shown to the right and `word` is "aop" then your algorithm would return `row=1` and `col=0`. If `word` is `jobs`, your algorithm would return `row=-1 col=-1` since `jobs` is not found on a diagonal.

```
x y z q
a b c d
j o b s
y z p n
```

```
struct loc_tag {
    size_t row;
    size_t col;
};
typedef struct loc_tag loc_t;

loc_t findDiagonalWord(const char * const * letters,
                      size_t r,
                      size_t c,
                      const char * word) {
    //YOUR CODE GOES HERE

}
```

## Question 7 Coding 4: A Program [18 pts]

For this problem, you are going to write a program which takes one command line argument—the name of an input file to read—reads all the lines in it, and prints how many lines in the file were identical to the last line in the file (including itself). For example, given

```
apple
banana
apple
cat
Banana
b a n a n a
banana
```

Your program should print a single line with the number 2 (because `banana` is the last line, and appears twice in the file). Note that this program only consider strings the same if they are **exactly** the same (differences in case and spacing matter). For this problem, you may assume:

- The correct number of command line arguments are given
- `fopen`, `fclose`, `malloc`, and `realloc` always succeed.

**You should write your answer on the next page**

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

//YOUR CODE GOES HERE
```