Com S 327
Fall 2023
Midterm Exam
DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: _____

ISU NetID (username): _____

70

Closed book and notes, no electronic devices, no headphones. Time limit ~~45~~ minutes. Partial credit may be given for partially correct solutions.

- Use correct C syntax for writing code.
- You are not required to write comments for your code; however, brief comments may help make your intention clear in case your code is incorrect.
- We do not expect you to memorize APIs; that's what man pages are for. If you have any questions about functions used in the exam problems, please ask.
- '␣' is a "visible space". You do not need to reproduce it in your output, but you may if you like.

If you have questions, please ask!

Question	Points	Your Score
1	30	
2	40	
3	30	
EC	1	
Total	100	

1. (30 pts; 3 each) For each code snippet, either give its output, indicate that a runtime error occurs (which does not necessarily imply that the program crashes), or indicate that it runs cleanly but produces no output. Invoking undefined behavior should be considered a runtime error. Exactly one of these three cases occurs for each problem.

   Be careful! These problems test more than just your knowledge and understanding of how the I/O functions work. In particular, if two of them look essentially the same, you should pay close attention to the differences.

   `char *strcpy(char *dest, const char *src)` copies the string pointed to by `src`, including the terminating null byte (`'\0'`), to the buffer pointed to by `dest`. You should have extensive experience with the rest of the functions used below, but please ask if you need explanations.

   You do not need to write newlines in your answers. They are included in the code only for the sake of completeness.

   Problem parts are fully independent.

```
enum { 0       1        2
  marty,  biff,     doc,
  george, jennifer, libyans
};     3        4        5

char *car[] = {
  "81␣DeLorean␣DMC-12",                "46␣Ford␣Super␣De␣Luxe",
  "48␣Packard␣Custom␣Eight␣Victoria", "84␣BMW␣733i",
  "84␣AMC␣Eagle␣4WD␣Wagon",            "Volkswagon␣Type␣2␣Microbus"
};

/* 0         1     0         1         0         1         *
 * 012345678901    012345678901234    01234567890123456 */
char character[][17] = {
  "Marty␣McFly",  "Biff␣Tannen",      "Emmett␣Brown",
  "George␣McFly", "Jennifer␣Parker", "Libyan␣Terrorists"
};                              McFly 0
```

(a)
```
char *s = "There's␣that␣word␣again.␣␣\"Heavy\".";
printf("%s\n", s);
```

There's that word again. "Heavy".

(b)
```
printf("%s:␣%s\n", character[marty], car[marty]);
```

Marty McFly: 81 DeLorean DMC-12

DAFUTR

(c)
```
strcpy(character[marty], "Marty");
printf("%s\n", character[marty]);
```

Marty

(d)
```
strcpy(character[biff], "Biff");
character[biff][4] = '␣';
printf("%s\n", character[biff]);
```

Biff Tannen

(e)
```
strcpy(car[marty], "85␣Toyota␣Xtra␣Cab␣4x4");
printf("%s\n", car[marty]);
```

Error - Attempt to write to read-only memory

(f)
```
/* Careful! */
printf("%s:␣%s\n", character[libyans], car[libyans]);
```

Error - character[libyans] is not a string (not null terminated) because it is truncated

(g)
```
strcpy(character[jennifer] + 9, character[marty] + 6);
printf("%s:␣%s\n", character[jennifer], car[jennifer]);
```

Jennifer McFly: 84 AMC Eagle 4WD Wagon

(h)
```
printf("%s\n", &car[libyans][-1]);
```

no output - prints a zero-length string

(i)
```
printf("%s\n", ((char *) character) + 68);
```

Jennifer Parker

(j)
```
printf("%s\n", *((char **) car) + 33);
```

De Luxe

2. (40 pts; 20 each) Complete the following functions according to the given specifications. You may not use any other functions (e.g., from the standard library or otherwise assumed) except, if necessary, `malloc()` and `free()`. You may not use any non-local variables. You may not write and use any "helper" functions. You may not leak memory. In all cases, you may assume that all arguments are non-NULL.

   (a) Implement `strncasecmp()` as defined in its man page, given here:

   > The **strcasecmp()** function performs a byte-by-byte comparison of the strings s1 and s2, ignoring the case of the characters. It returns an integer less than, equal to, or greater than zero if s1 is found, respectively, to be less than, to match, or be greater than s2.
   > The **strncasecmp()** function is similar, except it compares only the first n bytes of s1.

   Hint: You can convert uppercase to lowercase by bitwise or'ing with `0x20`, but this conversion is only valid if the character is alphabetical. You can check if a character is alphabetical with the predicate function `isalpha()`; **however, to simplify this exercise, you may assume that all characters are alphabetical and forgo tests for such!**

```
int strncasecmp(const char *s1, const char *s2, size_t n)
{
    int i;
    for (i = 0; i < n && (s1[i] | 0x20) == (s2[i] | 0x20); i++)
        ;
    return (s1[i] | 0x20) - (s2[i] | 0x20);
```

(b) Implement `strcspn()` as defined in its man page, given here:

> The **strcspn**() function returns the number of bytes in the initial segment of s which consists entirely of bytes not in reject.

For instance: `strcspn("flux capacitor", "abcde") == 5.`

`size_t` is an integer type which is guaranteed to be large enough to hold any indexable size on the host system (i.e., 32 bits on a 32-bit system and 64 bits on a 64-bit system).

```
size_t strcspn(const char *s, const char *reject)
{
```

Hello! pso

```
int i, j;

for ( i=0; s[i]; i++)
    for (j=0; reject[j]; j++)
        if (s[i] == reject[j])
            return i;

return i;
```

```
}
```

3. (30 pts; 3 each) Circle the correct statement for each problem that follows. You may assume that all parts are independent, that all code appears in a valid context, and that all *correct code* succeeds.

Documentation for `strcat()` from its man page:

> The **strcat**() function appends the src string to the dest string, overwriting the terminating null byte ('\0') at the end of dest, and then adds a terminating null byte. The strings may not overlap, and the dest string must have enough space for the result. If dest is not large enough, program behavior is unpredictable; buffer overr for attacking secure programs.

and for `strdup()`:

> The **strdup**() function returns a pointer to a new string which is a duplicate of the string s. Memory for the new string is obtained with **malloc**(3), and can be freed with **free**(3).

(a)
```
int *a;
free(a);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(b)
```
int i[5];
free(i);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(c)
```
int *i = malloc(15 * sizeof (*i));
free(i);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(d)
```
int *i = malloc(15 * sizeof (*i));
int **p = &i;
free(*p);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(e)
```
int *i, *j;
i = j = malloc(15 * sizeof (*i));
free(i);
free(j);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(f)
```
int *i = malloc(15 * sizeof (*i));
free(*i);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(g)
```
char s[80];
s = "1.21 gigawatts!";
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(h)
```
char *s = "I'm sure in 1985 plutonium is in every corner drug store";
strcat(s, ", but in 1955, its a little hard to come by!");
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(i)
```
char *s = strdup("Ronald Reagan?! The actor?!");
free(s);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

(j)
```
char *s = strdup("This is heavy.");
s++;
free(s - 1);
```

This code will: (a) run correctly; (b) invoke undefined behavior or crash; (c) not compile

Extra Credit. (1 pt) This is mostly just for fun, and it's only one point, so don't even waste time looking at it unless you're done with everything else. One of the TAs said that there needs to be a "hard question".

Give the output of the following program (Note: this will give different output on big- and little-endian hardware; assume that it is running on a little endian machine):

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  unsigned long long eightyeightmph = 3;
  unsigned long long gets = 4;
  unsigned long long prod;
  unsigned long long thing = 6;
  unsigned long long this = 13;
  unsigned long long to = 37;
  unsigned long long up = 59;
  unsigned long long when = 4514024232797LLU;

  prod = when * this * thing * gets * up * to * eightyeightmph;

  printf("%s\n", (char *) &prod);

  return 0;
}
```

88 MPH

char *s = "88 MPH";

printf("%llu\n%llu\n", *(int *)s, *((int *)s)+1)

"%llu\n", *(unsigned long long *)s