Com S 327
Fall 2018
Midterm Exam

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: _____

ISU NetID (username): _____

Closed book and notes, no electronic devices, no headphones. Time limit 70 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.

If you have questions, please ask!

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

1. (30 pts; 5 each) For each code snippet, either give its output, indicate that it produces a compile-time error, 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 four 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.

   (a)
   ```
   printf("You're␣plastic!");
   ```

   (b)
   ```
   int i;
   char *a[] = {
     "happen␣", "to␣", "make␣",
     "stop␣", "\"fetch\"␣", "trying␣",
   };
   int o[] = { 3, 5, 1, 2, 4, 0 };

   for (i = 0; i < 6; i++) {
     printf("%s", a[o[i]]);
   }
   ```

   (c)
   ```
   /*          0         1         2       *
    *          012345678901234567890123456 */
   char s[] = "On␣Wednesdays␣we␣wear␣pink!";
   *(s + 1) = 'O';
   *(s + 2) = 'n';
   *(s + 3) = '␣';
   *(s + 4) = 'S';
   *(s + 5) = 'a';
   *(s + 6) = 't';
   *(s + 7) = 'u';
   *(s + 8) = 'r';
   *(s + 26) = '.';
   printf("%s", s + 1);
   ```

(d)
```
/*          0          1          2          *
 *          01234567890123456789012345678 9 */
char *s = "I␣just␣moved␣here␣from␣Africa.";
s[17] = '.';
s[18] = '\0';
printf("%s", s);
```

(e)
```
char *p, s[] = "!ocoC␣nnelG␣,og␣uoY␣␣!ocoC␣nnelG␣,uoy␣rof␣ruoF";

for (p = s + strlen(s); *p; p--) {
  putchar(*p);
}
```

(f)
```
char *p, s[] = "?brac␣a␣rettub␣sI";

for (p = s + strlen(s) - 1; *p; p--) {
  putchar(*p);
}
```

2. (40 pts; (a) and (b) 10 each, (c) 20) 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()`, `free()`, a `printf()`-family function, or `atoi()`; use any non-local variables; write and use any "helper" functions; or leak memory (however, if the function is defined to return the address of dynamically allocated storage, it is the responsibility of the user to free that storage, so returning that address without freeing it is not considered a leak).

   Hint: There is plenty of space for all of these, even if you write large.

   (a) `swap()` swaps the values of its two integer arguments for the caller. For example:

```
int a = 5, b = 7;
swap(/* parameters are (some form of) a and b */);
/* a is now 7 and b is now 5 */
```

   You also need to complete the function signature for this one.

```
          /* fill in the parameter types */
void swap(              a,              b)
{









}
```

   (b) `freeplusplus()` takes a pointer to an address that was allocated by `malloc()`. It `free()`s the memory at that address and assigns NULL to the pointer such that the caller will receive a NULL pointer back from the function, e.g.:

```
foo_t *f = malloc(sizeof (foo_t));
freeplusplus(/* parameter is (some form of) f */);
/* f has been freed and is now NULL */
```

   You also need to complete the function signature for this one.

```
                  /* fill in the parameter type */
void freeplusplus(                              p)
{









}
```

(c) `strndup()` is a string function in the C standard library. Description of its functionality from its man page follows. Note that its function is described partially in terms of its differences from `strdup()`, so `strdup()` documentation is given as well.

DESCRIPTION

> 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).
>
> The strndup() function is similar, but copies at most n bytes. If s is longer than n, only n bytes are copied, and a terminating null byte ('\0') is added.

RETURN VALUE

> On success, the strdup() function returns a pointer to the duplicated string. It returns NULL if insufficient memory was available.

Note that the specification does not define how much storage to allocate when `n` is larger than is needed to copy `s`, so you may do whatever you feel is most expeditious.

```
char *strndup(const char *s, size_t n)
{









}
```

3. (30 pts; 3 each) Given the data structures and declarations below, give the types of the expressions. Some expressions may have more than one valid answer (for instance, `l` is correctly referred to as both a `struct list` and a `list_t`); you need only give one. Remember that *type* is a static concept and is not related to potential runtime errors created by the expression; we are not concerned with the latter, here.

   This is not an exercise in parsing type names like we practiced in class. In your answer, you would say that `li` is a `list_item_t *`, not "a pointer to a `list_item_t`".

```
#include <stdint.h>

typedef struct list_item {
  struct list_item *pred, *next;
  void *datum;
} list_item_t;

typedef list_item_t * list_iterator_t;

typedef struct list {
  list_item_t *head, *tail;
  uint32_t length;
  int32_t (*compare)(const void *key, const void *with);
  void (*datum_delete)(void *);
} list_t;

list_t l;
list_item_t *li;
list_iterator_t it;
```

   (a) `l.head`

   (b) `l.compare("Cady", "Regina")`

   (c) `&l.length`

   (d) `*l.head->next`

   (e) `*it`

   (f) `(it == li)`

   (g) `l.compare`

   (h) `(*it).next->datum`

   (i) `(&l)[7]`

   (j) `*((char *) li->datum)`

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[])
{
  int i;
  unsigned long long prod;
  unsigned long long r = 1, oo = 2, Ke = 5, in = 5, n = 5,
                     v = 3251, a = 3371, p = 51287, G = 52027;

  prod = Ke * v * in * (G * n * a * p * oo + r);

  for (i = 0; i < 8; i++) {
    putchar(((char *) &prod)[i]);
  }
  putchar('\n');

  return 0;
}
```