Com S 229
Spring 2015
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 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.

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 you expect—or, at least, would hope for—a runtime error to occur, or indicate that it runs cleanly but produces no output. 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.

   `strlen()` returns the number of bytes in its argument, not including the NULL terminator.

   (a)
```
printf("I␣know␣what␣we're␣gonna␣do␣today.");
```

   (b)
```
int i;
char *a[] = {
  "e␣game!!", "␣are␣bui", "lding␣a␣",
  "Phineas␣", "roguelik", "and␣Ferb",
};
int o[] = { 3, 5, 1, 2, 4, 0 };

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

   (c)
```
char s[] = "I'm␣Lindana␣and␣I␣wanna␣have␣fun!";
*(s + 11) = '\0';
printf(s + 4);
```

(d)

```
char *s = "Hey,␣where's␣Perry?";
*(s + 18) = '\0';
printf(s + 13);
```

(e)

```
char *p, s[] = { '?', '\'', 'n', 'i', 'o', 'd', '␣',
                 'a',  'h', 'c', 't', 'a', 'h', 'W', '\0' };

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

(f)

```
char *p, s[] = { '.', '.', '.', 'p', 'm', 'u', 't', 'h',
                 'c', 's', 'l', 'e', 'm', 'm', 'i', 'G',
                 '␣', 'n', 'i', '␣', 'k', 'c', 'a', 'B', '\0'  };

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

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; 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. In all cases, you may assume that all arguments are non-NULL.

   (a) `strdup()` returns a dynamically allocated copy of `s`, or NULL on failure. `s` is a C string. The returned copy must also be a valid C string using the minimal amount of storage required for the copy.

```
char *strdup(const char *s)
{


}
```

(b) `strchr()` returns the address of of the first occurrence of `c` in `s`. If `s` does not contain an instance of `c`, `strchr()` returns `NULL`.

```
char *strchr(const char *s, int c)
{











}
```

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("Vanessa", "Candace")`

   (c) `&l.length`

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

   (e) `*it`

   (f) `(it == li)`

   (g) `li->next[3]`

   (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:

```
#include <stdio.h>

char *c[] = {
  "THE",
  "ENTIRE",
  "TRI-STATE",
  "AREA!"
};

char **cp[] = { c + 3, c + 2, c + 1, c };
char ***cpp = cp + 1;

int main(int argc, char *argv[])
{
  putchar(*(**++cpp+3));
  putchar(****++cpp-5);
  printf("%c%c", *(**(--cpp-2)+2)+18, *(**(cpp-3)+3));
  printf("␣%s", *cpp[-1]+4);
  printf("%s\n", *cpp[-2]+4);

  return 0;
}
```