Com S 327
Spring 2016
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 may always assume that all required headers are included.
- 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; 3 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.

   Assume that the code below all appears in the given order in one function; thus, state changed in, say, part (c) applies to parts (d)–(j).

   `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.

```
enum {
  arryn,   baratheon, greyjoy,   lannister,
  martell, stark,     targaryen, tully
};

char *words[] = {
  "As␣High␣as␣Honor",          "Ours␣is␣the␣Fury",
  "We␣Do␣Not␣Sow",             "Hear␣me␣Roar!",
  "Unbowed,␣Unbent,␣Unbroken", "Winter␣is␣Coming",
  "Fire␣and␣Blood",            "Family,␣Duty,␣Honor"
};

char lords[][7] = {
  "Jon",   "Robert", "Balon",   "Tywin",
  "Doran", "Eddard", "Viserys", "Hoster"
};

char *s;
int i;
```

(a)
```
printf("%s:␣%s\n", s = lords[stark], words[stark]);
```

(b)
```
s = "Rob";
printf(s);
putchar('\n');
```

(c)
```
printf("%s\n", lords[stark]);
```

(d)
```
strcpy(lords[stark], s);
printf("%s\n", s = lords[stark]);
```

(e)
```
s = "Cersei";
for (i = 0; i < 7; i++) {
  lords[lannister][i] = *s++;
}
printf("%s\n", lords[lannister]);
```

(f)
```
printf("%s\n", lords[targaryen]);
```

(g)
```
strcpy(lords[tully], "Brynden");
printf("%s\n", lords[tully]);
```

(h)
```
printf("%s\n", ((char *) lords) + 14);
```

(i)
```
s = words[4];
printf(s + 19);
putchar('\n');
```

(j)
```
strcpy(words[baratheon], words[lannister]);
printf("words[lannister]\n");
```

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()`, `free()`, a `printf()`-family function, or `atoi()`. 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.

   (a) `times_tables()` returns a dynamically allocated array of `int` large enough to hold `rows` $\times$ `columns` elements containing times tables defined as follows: In the one-dimensional output array is the row-major, two-dimension matrix containing at zero-indexed row i and zero-indexed column j the value $(i+1)\times(j+1)$ (thus, no multiples of zero appear in the table). The function will return `NULL` if allocation fails.

```
#include <stdio.h>  /* printf(), fprintf(), etc. */
#include <stdlib.h> /* malloc(), free(), atoi()  */

int *times_tables(int rows, int columns)
{









}
```

(b) This `main()` function checks that there are two command line arguments. If there are not, it will print a hilarious error message and exit with exit code `-1`. You may then assume that the arguments are well formed and convert them to integers with the first argument to contain a number of rows and the second a number of columns. Call `times_tables()` on the converted arguments and print the output in a tab-delimited `rows` $\times$ `columns` format. You should not need any more variables than are already defined, but you are welcome to create more if you like.

```
int main(int argc, char *argv[])
{
  int *times_table;
  int rows, cols;
  int i, j;









}
```

3. (30 pts; 3 each) Each of the code snippets below is independent. For this exercise, code is considered to contain an error if it overflows a buffer, leaks memory, makes an error calling `free()`, or invokes undefined behavior. Assume that `malloc()` always succeeds and that `int`s are 4 bytes.

   For each problem, indicate that it contains an error by writing "error" or indicate that it is error free with "no error".

   (a)
   ```
   int *a;
   *a = 7;
   ```

   (b)
   ```
   int *a = malloc(1);
   a[0] = 5;
   free(a);
   ```

   (c)
   ```
   int i;
   int **a = malloc(3 * sizeof (*a));
   for (i = 0; i < 3; i++) {
     a[i] = malloc(sizeof (**a));
     a[i][0] = i;
     free(a[i]);
   }
   free(a);
   ```

   (d)
   ```
   int *a;
   int b[5];
   a = &b;
   free(a);
   ```

   (e)
   ```
   int *a;
   int b[5];
   a = b;
   free(a);
   ```

(f)
```
int *a;
int b[5];
a = b;
```

(g)
```
int *a;
a = malloc(4);
a[0] = 5;
free(a);
```

(h)
```
int i;
int **a = malloc(3 * sizeof (*a));
for (i = 0; i < 3; i++) {
  a[i] = malloc(sizeof (**a));
  a[i][0] = i;
}
free(a);
for (i = 0; i < 3; i++) {
  free(a[i]);
}
```

(i)
```
int *a, *b;
a = b = malloc(5 * sizeof (int));
free(b);
free(a);
```

(j)
```
int *a, b;
a = malloc(sizeof (*a));
b = (int) a;
a = NULL;
free((void *) b);
```

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—and have double-checked!—everything else.

Give the output of the following program:

```
int main(int argc, char *argv[])
{

  enum {
    arryn,   baratheon, greyjoy,   lannister,
    martell, stark,     targaryen, tully
  };

  char *words[] = {
    "As␣High␣as␣Honor",          "Ours␣is␣the␣Fury",
    "We␣Do␣Not␣Sow",             "Hear␣me␣Roar!",
    "Unbowed,␣Unbent,␣Unbroken", "Winter␣is␣Coming",
    "Fire␣and␣Blood",            "Family,␣Duty,␣Honor"
  };

  char lords[][7] = {
    "Jon",   "Robert", "Balon",   "Tywin",
    "Doran", "Eddard", "Viserys", "Hoster"
  };

  int i;
  char *s = words[6];

  i = -7;
  putchar(s[-2] & 0xdf);
  lords[0][18] = '␣';
  printf(lords[2] + 3);
  putchar(s[i++]);
  putchar(lords[-i][-(i + 1)]);
  putchar(s[i - 1] | 0x20);
  putchar(s[(i *= -2) - 2]);
  putchar(s[--i]);
  s = lords[i / 2];
  s[19]++;
  s[17] -= (++i / 2);
  printf("%s", s + 17);
  putchar(words[i / 4 + 1][-2]);
  putchar('\n');

  return 0;
}
```