Duration: **2 hours**
Aids Allowed: **none**

Student Number: |_|_|_|_|_|_|_|_|_|_|_|

Family Name(s): _____

Given Name(s): _____

Lecture Section: ☐ LEC01 (with François Pitt)

☐ LEC02 (with Kante Easley)

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This term test consists of 4 questions on 14 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete, fill in the identification section above, write your student number where indicated at the bottom of every odd-numbered page (except page 1), and write your name on the back of the last page.*

Answer each question directly on this paper, in the space provided. If you need more space for one of your solutions, use one of the (mostly) blank pages at the end of the test and *indicate clearly the part of your work that should be marked.*

Write up your solutions carefully! In particular, use notation and terminology correctly and explain what you are trying to do—part marks *will* be given for showing that you know the general structure of an answer, even if your solution is incomplete.

When writing code, assume all input is valid unless otherwise indicated (*i.e.*, do not waste time writing code to perform error-checking). Comments are not required (except where indicated), although they may help us mark your answers more accurately in case you are unable to complete the code.

If you are unable to answer a question (or part), you will get 20% of the marks for that question (or part) if you write "I don't know" and nothing else—you will *not* get those marks if your answer is completely blank, or if it contains contradictory statements (such as "I don't know" followed or preceded by parts of a solution that have not been crossed off).

MARKING GUIDE

1: _____/30

2: _____/10

3: _____/ 8

4: _____/12

TOTAL: _____/60

Good Luck!

OVER...

Question 1. [30 MARKS]

Part (a) [2 MARKS]

In the space on the right, show the output of the following code fragment.

```
int i = 5, j = 3;
printf("%d %d\n", i--, ++j);
printf("%d %d\n", i, j);
```

Part (b) [1 MARK]

Simplify the following `if` statement as much as possible—it can be replaced by a single assignment!

```
if (age >= 13)
    if (age <= 19)  teenager = 1;
    else            teenager = 0;
else
    if (age < 13)   teenager = 0;
```

Part (c) [1 MARK]

Give an *expression* that evaluates to the absolute value of the integer variable `i`.

Part (d) [1 MARK]

The code fragment below contains one small bug. Explain what it is and how to fix it.

```
if (n % 2 == 0);
    printf("%d is even\n", n);
```

Part (e) [1 MARK]

Give the output of the following code fragment.

```
int i;
for (i = 10; i > 1; i /= 2)
    printf("%d ", i);
```

Question 1. (CONTINUED)

Part (f) [3 MARKS]

Complete the following function according to its comment.

```
// Return the number of positive values in a[0..n-1].
int count_pos(int a[], int n)
{




}
```

Part (g) [2 MARKS]

Complete the following function according to its comment.

```
// Set hours to the current number of hours on a 24-hour clock, given
// seconds_since_midnight (where midnight is 00:00:00).
void set_time(long seconds_since_midnight, short *hours)
{




}
```

Question 1. (CONTINUED)

Part (h) [5 MARKS]

Identify every bug in this program and fix it. Then, show the output of the program.

```
void reverse(int *array, int n)
{   int t, *i = array, *j = array + n;
    while (i < j)
    {   t = *i;
        *i++ = *j;
        *j-- = t;
}   }

int main(void)
{   int *s, *e, test_array[] = {1, 2, 3, 4, 5};
    reverse(test_array, sizeof(test_array));
    s = test_array;
    e = test_array + sizeof(test_array) / sizeof(test_array[0]);
    while (s < e)  printf("%d ", *s++);
    printf("\n");
}
```

Part (i) [2 MARKS]

What is the output of the following program? (Hint: draw the memory!)

```
void three_card_monte(char *a, char *b, char *c)
{   *a = *b;
    a = b;
    *a = 'o';
    c = a;
    *c = *a;
}
int main(void)
{   char i = 'o', j = '_', k = '_';
    char *a = &i, *b = &j, *c = &k;
    three_card_monte(a, b, c);
    printf("%c, %c, %c\n", i, j, k);
}
```

Question 1. (CONTINUED)

Part (j) [2 MARKS]

The following function contains an error. Find the bug and fix it.

```
// Return 1 if some element in a[0..n-1] is equal to 0; return 0 otherwise.
int has_zero(int a[], int n)
{   int i;
    for (i = 0; i < n; ++i)
        if (a[i] == 0)
            return 1;
        else
            return 0;
}
```

Part (k) [3 MARKS]

Complete the function below according to its comment (you did something like this for one of the labs).

```
/* FUNCTION palindrome
 * Parameters and preconditions:
 *     str != NULL: points to a valid string (terminated by '\0')
 * Return value:
 *     1 if str points to a palindrome (a string that reads exactly the same
 *     left-to-right as right-to-left, considering every character); 0 otherwise
 */
int palindrome(const char *str) {




} // end palindrome
```

Question 1. (CONTINUED)

Part (l) [2 MARKS]

What is the output of the following code fragment? Write your answer in the space on the right.

```
int a[] = {5, 15, 34, 54, 14, 2, 52, 72};
int *p = &a[1], *q = &a[5];
printf("%d\n", *(p + 3));
printf("%d\n", q - p);
printf("%s\n", *p < *q ? "true" : "false");
```

Part (m) [5 MARKS]

Complete the function below according to its comment (we saw this function in lecture).

```
/* FUNCTION concat
 * Parameters and preconditions:
 *     s1 != NULL: points to a valid string (terminated by '\0')
 *     s2 != NULL: points to a valid string (terminated by '\0')
 * Return value:
 *     A pointer to a new string that contains s1 followed by s2 -- or NULL if memory
 *     could not be allocated for the new string.
 * Notes:
 *     The new string is allocated dynamically to be just large enough.
 *     Assume that <stdlib.h> and <string.h> have already been #included.
 */
char *concat(const char *s1, const char *s2)
{
```

```
} // end concat
```

Question 2. [10 MARKS]

In the space to the right, give a *tight* bound on the worst-case running time of each of the following code fragments. *Show your work*, *i.e.*, indicate how you obtained your answer.

Part (a) [2 MARKS]

```
void some_even(int n, int *array)
{   int i, j;
    for (i = 0; i < n; ++i)
        for (j = n; j > i; j -= 2)
            array[i] = array[j] - 1;
}
```

Part (b) [2 MARKS]

```
int equal(int n, int *a1, int *a2)
{   int *end = a1 + n;
    while (a1 < end)
        if (*a1++ != *a2++)
            return 0;
    return 1;
}
```

Part (c) [2 MARKS]

```
// Let n = number of nodes starting at first.
void free_list(node_t *first)
{   node_t *old;
    while (first != NULL)
    {   old = first;
        first = first->next;
        free(old);
}   }
```

Part (d) [2 MARKS]

```
void some_sum(int n, int *array)
{   int i, j;
    for (i = 1; i < n-1; ++i)
        for (j = i-1; j < i+1; ++j)
            array[i] += array[j];
}
```

Part (e) [2 MARKS]

```
void some_even(int n, int *array)
{   int i, j;
    for (i = 0; i < n; ++i)
        for (j = 1; j < n; j *= 2)
            if (i % 2 == 0)  array[i] += j;
}
```

Question 3. [8 MARKS]

Finish the `magic_square` function **which is started for you on the next page** so that the program below prints an `N x N` magic square (a square arrangement of the numbers $1, 2, 3, \ldots, n^2$ in which the sums of the rows, columns, and diagonals are all the same), for any odd value of `N`.

The algorithm you should use to create a magic square is the following:

(a) start by placing the number 1 in the top middle position;
(b) place each remaining number $(2, 3, \ldots, n^2)$ by moving up one row and to the right one column;
(c) any attempt to go outside the array bounds should "wrap around" to the opposite side of the array;
(d) if a particular array element is occupied, put the number directly below the *previously* stored number.

For example, if `N == 5` then the program should print out:

```
 17  24   1   8  15
 23   5   7  14  16
  4   6  13  20  22
 10  12  19  21   3
 11  18  25   2   9
```

```
#include <stdio.h>

// The size of the square -- must be odd.
#define N   5

void magic_square(); // define this function on the next page

// Print the magic square.
// Parameters and preconditions: square != NULL points to the magic square array
void print_square(int square[N][N])
{
    int i, j;
    for (i = 0; i < N; ++i) {
        for (j = 0; j < N, ++j)
            printf(" %3d", square[i][j]);
            // Note: "%3d" prints each integer using at least 3 characters,
            // padded with spaces as necessary.
        printf("\n");
    }
}

// Create and print an N x N magic square.
int main(void)
{
    magic_square();
    return 0;
}
```

Question 3. (CONTINUED)

```
/* FUNCTION magic_square
 * Create and print an N x N magic square, using the algorithm given in the question.
 * Only works when N is odd.
 */
void magic_square(void)
{
    int i, j, ti, tj, value, square[N][N];

    // Initialize square[][] to be all 0's -- FILL IN THE CODE BELOW.









    // Now, fill the square according to the algorithm -- COMPLETE THE CODE BELOW.

    i = ____________; // fill in the blank with an appropriate initial value

    j = ____________; // fill in the blank with an appropriate initial value

    for (value = 1; value <= N * N; ++value) {









    } // end for

    // Finally, print the magic square.
    print_square(square);

} // end magic_square
```

Question 4. [12 MARKS]

Using the `rot_encode` function from lecture (given below), write a program that does the following:

(a) takes a single command-line argument (in `argv`) containing the name of a text file;
(b) reads the bytes from that file, and encrypts each one using `rot_encode`;
(c) writes the encrypted bytes, one by one, to a new file whose name is the same as the original file, but with `".rot"` appended to the end.

To simplify your task, assume that all file operations and all memory allocation succeeds (*i.e.*, do not bother to write code to check the return value of functions `fopen` and `malloc`). Don't forget to `close` open files and to `free` dynamic memory!

(Hint: split up the work into meaningful chunks/sections and write comments to indicate what each chunk is doing—this will be worth part marks even if you cannot complete a chunk.)

```
#include <stdio.h>

#define ROTATION  13

/* FUNCTION rot_encode
 *  Replace c with its rotated encrypted counterpart if c points to a letter.
 *  Do nothing otherwise.
 */
void rot_encode(char *c)
{
    if ('a' <= *c && *c <= 'z')
        *c = ((*c - 'a' + ROTATION) % ('z' - 'a' + 1)) + 'a';
    else if ('A' <= *c && *c <= 'Z')
        *c = ((*c - 'A' + ROTATION) % ('Z' - 'A' + 1)) + 'A';
}

/* FUNCTION main -- see above. */
int main(int argc, char *argv[])
{
```

Question 4. (CONTINUED)

```
} // end main
```

[Use the space on this page for scratch work, or for any part of an answer that did not fit elsewhere. ***Clearly label each answer with the appropriate question and part number.****]*

[Use the space on this page for scratch work, or for any part of an answer that did not fit elsewhere. ***Clearly label each answer with the appropriate question and part number.****]*

On this page, please write nothing except your name.

Family Name(s): _____

Given Name(s): _____