CS 1713, Intro. to Computer Science
Final Examination

(20) 1. Give a C code segment that will read an integer `n` from `stdin`, and will print the numbers from `n-1` to `0` (inclusive) to `stdout`, on one line, with a space between each number. (Thus if 5 is read, the numbers 4, 3, 2, 1, and 0 should be printed on one line.)

(20) 2. Write a C code segment that will search for a zero value in the first 100 locations of an array `arr` of `doubles`. If a zero is found anywhere in the first 100 locations, your segment should print `"Zero found"`. If there are no zeros in the first 100 locations, it should print `"No zeros"`.

```
double arr[200];
```

(30) 3. (a) Write a complete C program which uses `getchar()` to read the standard input until end-of-file, counts the number of Ascii digits input (`'0'` through `'9'`) and outputs the number of digits. Your program should use the C library function `isdigit()`, which requires the `<ctype.h>` header file. For a character stored in `ch`, `isdigit(ch)` returns a non-zero (true) if `ch` is an Ascii decimal digit character, and returns zero if not. (You do not write the code for `isdigit()`.)

(b) Assume the source program in part (a) is called `digits.c`. Give the Unix command to compile this code to create an executable file named `digits`. Give the Unix command to use the `digits` program to find the number of digits in the program source file `digits.c` itself.

(30) 4. Write a complete C program to calculate and print *Fibonacci* numbers. These numbers start out 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ..., where the sequence starts with 0, 1, and each subsequent number is the sum of the two previous ones. (Thus 34 = 13 + 21.) Use an array `int f[40]`. Store `0` and `1` in positions `f[0]` and `f[1]`. Then use the property of the sequence to calculate and store the proper number in `f[2]` through `f[39]` . Finally, print the forty numbers, one to a line. (You *must* use loops to calculate and print.)

(40) 5. Write a complete C program that will play the game of "Chuck-O-Luck". First one must determine a number to bet on. Let's suppose your program always bets on 4. (Each time you play the game, you will first bet on 4.) At each play, you then roll three dice, and keeps track of each of the rolls. If 4 comes up on all three dice, you win $3. If 4 comes up on only 2 of the dice, you win $2. If 4 comes up on only one of the dice, you win $1. If 4 doesn't come up at all, then you win -$1, that is, you lose $1. Your program should play the game 1000 times. You should keep track of your total winnings for all of the games and print this total at the end. (Thus if you roll 4-3-4, you win $2. If you then roll 2-5-1, you lose $1. If you the roll 4-4-4, you win $3. Your total winnings for these three games is: 2 + (-1) + 3 dollars. Notice that the number of dollars that you win is equal to the number of times your number comes up on the three dice, unless it doesn't come up at all, in which case you lose a dollar.)

(30) 6. Consider the following C program. This program does what is called a *random walk in the plane*. It starts with a 2-dimensional array of characters (75 on a side), and starts in at the 40th row and the 40th column. Then at each step, the program randomly moves to one of the four positions horizontally or vertically from the given position. At each step it inserts a `*` character, so that one can see where it has gone. The program stops when it gets to the boundary.

```
/* Random walk in the plane. By NR Wagner */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <time.h>
#define MAXP 75

void blank_plane(char plane[MAXP][MAXP]);
void one_step(int x, int y, int *new_x, int *new_y);
void print_plane(char plane[MAXP][MAXP]);
void main(void)
{
    char plane[MAXP][MAXP];
    int x = 40, y = 40; /* initial location */
    int new_x, new_y;
    srand48((long)time(NULL));
    blank_plane(plane);
    while( x >= 0 && x < MAXP && y >= 0 && y < MAXP) {
        plane[x][y] = '*';
        one_step(x, y, &new_x, &new_y);
        x = new_x; y = new_y;
    }
    print_plane(plane);
}
void one_step(int x, int y, int *new_x, int *new_y)
{
    if (drand48() < 0.5) { /* change x */
        *new_y = y;
        if (drand48() < 0.5) *new_x = x + 1.0;
        else *new_x = x - 1.0;
    }
    else { /* change y */
        *new_x = x;
        if (drand48() < 0.5) *new_y = y + 1.0;
        else *new_y = y - 1.0;
    }
}
```

Sample output (first 35 lines omitted)

(a) Give the C code for the function `blank_plane`, which will set each character location to a blank character.

(b) Give the C code for the function `print_plane`, which will print out the square array with a vertical line at the start and the end of each line, as shown above.

(c) Explain briefly how the program stops when it gets to the boundary.

(d) Explain briefly how the function `one_step` is getting data back to the `main` function.