ECE 220: Computer Systems and Programming

Fall 2020 – Midterm Exam 2

November 5, 2020

1. This is a closed-book, closed-notes exam
2. Absolutely no interaction between students is allowed
3. Illegible handwriting will be graded as incorrect
4. You must put your name and NetID on your submission page
5. **Use a separate page for each question**
6. Submission is only accepted through Gradescope

Question 1 (40 points): _____

Question 2 (40 points): _____

Question 3 (10 points): 1)_____; 2)_____

Question 4 (10 points): 1)_____; 2)_____

Total Score: _____

Write down your answers in the following format. Each question should be on a separate page.

Name:

NetID:

Q1 (**write down the # and entire line of code highlighted in yellow**)

Q2 (**write down the # and entire line of code highlighted in yellow**)

Q3
1)
2)

Q4
1)
2)

Problem 1 (40 points): Array

A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same value. Given an m×n matrix, your program needs to return 1 if the matrix is Toeplitz, otherwise return 0. **Note the matrix is stored as a 1-D array.**

Example 1:
Input: m=3, n=4
matrix = [1, 4, 3, 9,
6, 1, 4, 3,
5, 6, 1, 4]
Output: 1

Example 2:
Input: m=2, n=2
matrix=[1, 3,
3, 2]
Output: 0

```
int is_toeplitz(int* matrix, int m, int n){
     int r, c;
     // Loop through each row
     (1)for (_______;________; r++){
        // Loop through each column
        (2)for (_______;________; c++){
               if (r > 0 && c > 0){
                   // linear_idx of cell (r, c)
                   (3)int cur_idx = _________________;

                   // linear_idx of cell (r - 1, c - 1)
                   (4)int upper_left_idx = ____________________;

                   // if cell (r,c) and cell (r-1,c-1) is not equal, return 0
                   (5)if(_________________)
                       (6)_____________;
               }
           }
        }
        (7)_____________;
}
```

Problem 2 (40 points): Recursion

In this problem, we compute the area of a fractal shape (that is, a recursively defined shape) and count the number of shapes in this fractal.

The fractal follows these rules.
Given a positive integer *r*:
If *r* mod 3 is 0:
It is a circle with radius = *r*, and it grows two squares with side length = *r*-1
If *r* mod 3 is 1:
It is a 90-45-45 triangle with two equal sides = *r*, and it grows a circle with side length = *r*-1
If *r* mod 3 is 2:
It is a square with sides = *r*, and it will grow a 90-45-45 triangle such that the equal sides have length= *r*-1
If *r* is 0: no new shape is appended.

We want to compute the total area of this fractal shape and also keep count of how many shapes are in this fractal, including circles, triangles, and squares. The formulas for computing the areas of different shapes are listed below.

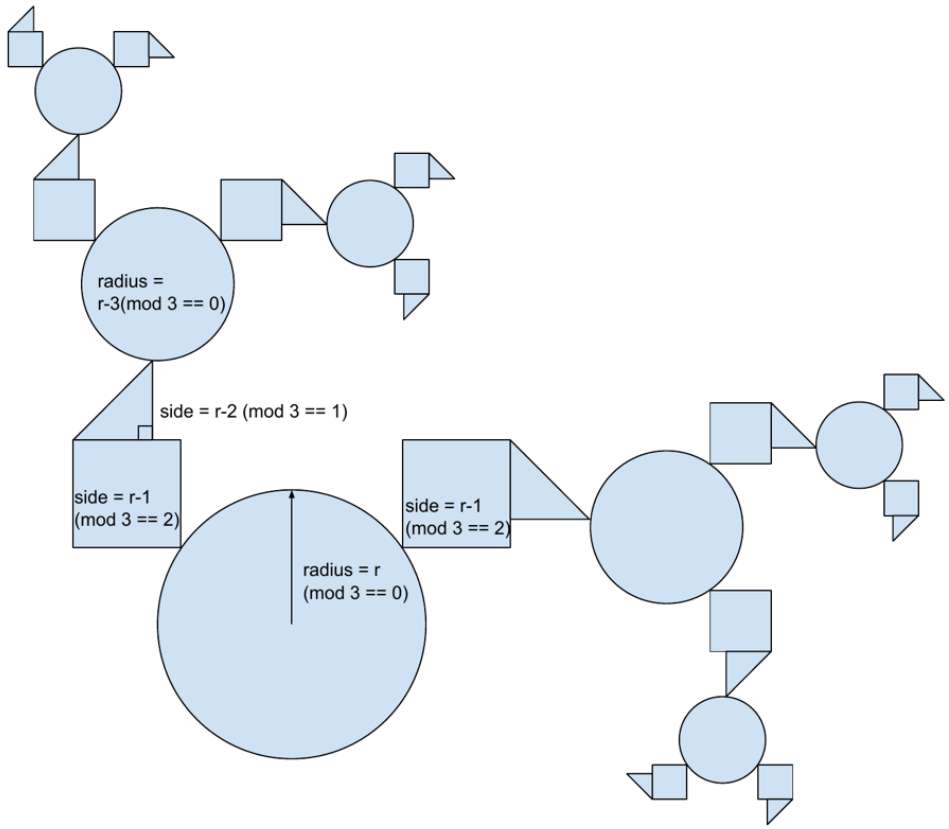
Area of a circle with radius r: $A = \pi r^2$
Area of a 90-45-45 triangle with two equal sides of r: $A = \frac{1}{2} r^2$
Area of a square with sides of r: $A = r^2$

The function **float area(int r, int* count)** takes an integer and an integer pointer; it returns a floating point value.

Inputs
r: radius of a circle, or side length of a square, or equal side length for a 90-45-45 triangle.
count: count for the number of shapes in the fractal (note that it is being passed as a pointer).

Output
area: accumulated area for all the shapes in the fractal

For example, for r = 9 the area is ~= 938.760986 (π is given in the code) and the count = 35 for the number of shapes in the fractal.




```
#include <stdio.h>
#include <stdlib.h>
static float PI=3.14159265358979323846;
// compute the fractal shape area, i.e. area is the output
// r is the parameter for each shape, count is the number of shapes
float area(int r, int* count){
  // deal with r = 0
  if ((1)_____________________________)
      (2)_____________________________;
```

```
 float total_area= 0.0, curr_area = 0.0;
 if((3)_____){  // mod 3 is 0
   curr_area = (4)_____;
   total_area = curr_area + (5)_____;
 }
 else if ( (6)_____){  // mod 3 is 1
   curr_area = (7)_____;
   total_area = curr_area + (8)_____;
 }
 else {  // mod 3 is 2
   curr_area = (9)_____;
   total_area = curr_area + (10)_____;
 }

 // update count for the total number of shapes
 (11)_____;
 return (12)_____;
}



int main (){
 int r1=9, r2=27;
 int  count = 0;
 float area_9 = (13)_____;
 printf("r=9: %f, number = %d\n", area_9, count);
 count = 0;
 float area_27 = (14)_____;
 printf("r=27: %f, number = %d\n", area_27, count);
 return 0;
}
```

Problem 3 (10 points): Debug

The below program contains two functions, main() and norm2(). norm2() calculates the Euclidean norm, or 2-norm, of a one-dimensional array A. That is, given an array of length *n*, it returns:

$$\sqrt{(A_1^2 + A_2^2 + \cdots A_n^2)}$$

In order to test norm2(), we have written a main() function. We successfully compiled the program, which is in file norm2.c, using the command:

```
gcc -g -O0 -lm norm2.c -o norm2
```

We tested norm2() on the 5-item array [1,1,2,0,0], and we expected to get $\sqrt{6} \approx 2.44$.

When we run the program, however, it instead prints 5.

There are two (2) lines with bugs in this program. For each bug, you may do either (or both):

- **a)** Give the bug's line number and describe its effect.
- **b)** Give one (1) line or function where you would set a breakpoint in GDB. Explain both your reasoning and what you would examine at that breakpoint.

Each bug is graded separately.

(a) and (b) will be graded as equivalent and worth equal points when equally correct. **For a given bug, if you provide both (a) and (b), we will go with the answer that scores better** (so coming up with a good breakpoint first may be a good idea).

Example: good explanation for a GDB breakpoint
Breakpoint: line 1005
Reason: In line 1005, which is part of function foo, the program calls function bar and assigns its return value to variable baz. We know from the problem statement that the return value of function bar is correct, but when we return baz in line 1009, it's wrong. So we need to track what happens to baz from line 1005 to 1008. I would step through lines 1005 to 1008, printing baz each time.

For this problem, you may assume:
1. That the use of sqrt(x) in line 23 is correct.
2. That using the compilation instruction provided above, the code compiles without warnings.

Code:

```
 1      #include <stdio.h>
 2      #include <math.h>
 3      #include <stdlib.h>
 4
 5      double norm2(int *arr, int n); //declaration
 6
 7      int main() {
 8            int n = 5;
 9            int arr[] = {1,1,2,0,0};
10            double res = norm2(arr, n);
11            printf("%d\n", res);
12            return 0;
13      }
14
15
16      //instantiation
17      double norm2(int *arr, int n){
18            int i;
19            double squaresum = 0;
20            for (i = 1; i <= n; i++) {
21                  squaresum += arr[i]*arr[i]; //(A_i)^2 added
22            }
23            double sqt = sqrt(squaresum); //take square root
24            return sqt;
25 }
```

1) What is the bug with the smaller line number, and its effect, OR where would you put a breakpoint to find it, and what would you check at that breakpoint?

Your Answer:

2) What is the bug with the larger line number and its effect, OR where would you put a breakpoint to find it, and what would you check at that breakpoint?

Your Answer:

Problem 4 (10 points): Concepts

1) Run-Time Stack Part A

Recall in **MP7 - Sudoku Solver**, the function **is_val_valid** is used by the function **solve_sudoku** to determine whether a value can be filled in a specific cell on the board. **is_val_valid** then uses three other functions to determine the validity of a value at a given cell. An implementation for **is_val_valid** is provided, along with relevant parts of **solve_sudoku**. Assume all functions used are correctly implemented and the provided code is correct. **Given that the function is_val_in_3x3_zone is currently being executed, which functions' activation records are currently on the run-time stack?** Fill in your answer on the stack. An example is given on the left.

```
int is_val_valid(const int val, const int i, const int j,
const int sudoku[9][9]){
     if(is_val_in_row(val, i, sudoku)){
          return 0;
     } else if(is_val_in_col(val, j, sudoku)){
          return 0;
     } else if(is_val_in_3x3_zone(val, i, j, sudoku)){
          return 0;
     } else {
          return 1;
     }
}

int solve_sudoku(int sudoku[9][9]){
     int i, j, num;
     // Parts of the function omitted for simplicity
     for(num = 1; num <= 9; num++){
          if(is_val_valid(num, i, j, sudoku)){
               // omitted for simplicity
          }
     }
     // Rest of the function omitted for simplicity
}
```

Example Run-Time Stack	Fill in your answer here
foo's activation record	
main's activation record	solve_sudoku's activation record

2) Run-Time Stack Part B

Below code attempts to swap the values in variables first and second. We know that the implementation is incorrect and the values in first and second will not change after line 9 is executed. **Explain why using what we learned about the run-time stack and function calls in C.**

```
1 void swap(int a, int b){
2    int temp = a;
3    a = b;
4    b = temp;
5 }
6 int main(){
7    int first = 1;
8    int second = 2;
9    swap(first, second);
10   return 0;
11 }
```

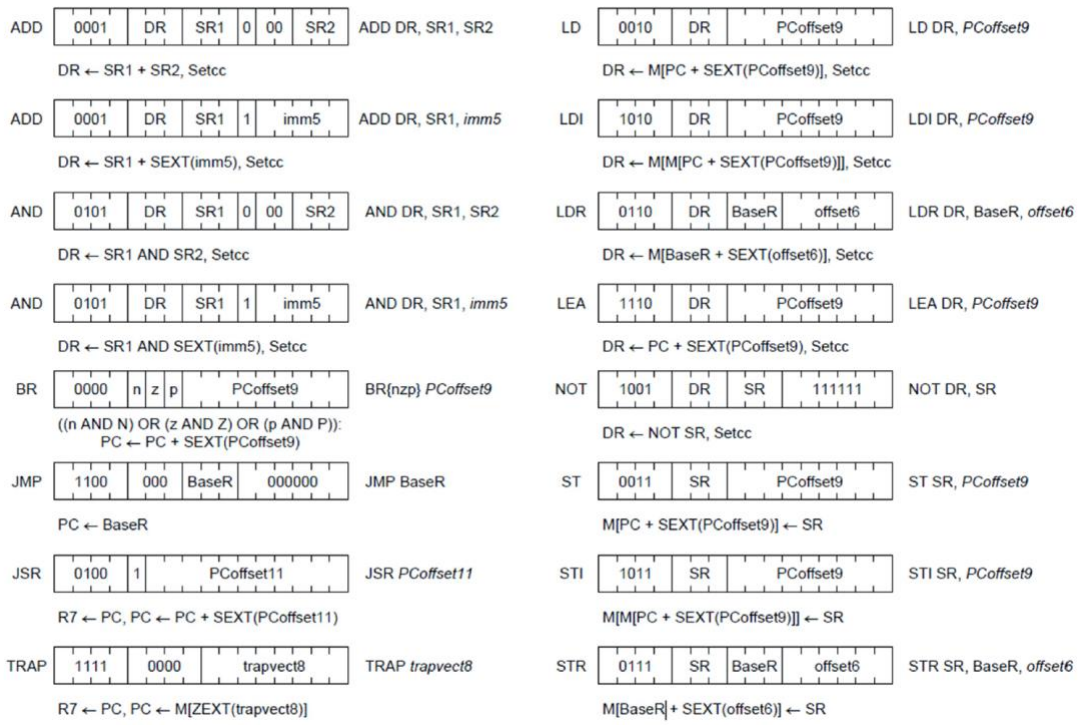
Your Answer:

Table E.2 The Standard ASCII Table

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	'	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F

LC-3 Instructions

NOTES: RTL corresponds to execution (after fetch!); JSRR not shown



End of ECE 220 Midterm Exam 2