Name: _____

- Instructions:
  - Show your work to receive partial credit.
  - Keep your eyes on your own paper and do your best to prevent anyone else from seeing your work.
  - Do NOT communicate with anyone other than the professor/proctor for ANY reason in ANY language in ANY manner.
  - This exam is closed notes, closed books, no calculator.
  - Turn all mobile devices off and put them away now. You cannot have them on your desk.
  - Write neatly and clearly indicate your answers. What I cannot read, I will assume to be incorrect.
  - Stop writing when told to do so at the end of the exam. I will take 5 points off your exam if I have to tell you multiple times.
  - Academic misconduct will not be tolerated. Suspected academic misconduct will be immediately referred to the Rollins Honor Council. Penalties for misconduct will be a zero on this exam, an F grade in the course, and/or other disciplinary action that may be applied by the Rollins Honor Council.
- Time: This exam has 6 questions on 9 pages including the title page. Please check to make sure all pages are included. You will have 75 minutes to complete this exam.

On my honor, I have not given, nor received, nor witnessed any unauthorized assistance on this work. Also, I have read and understand the above policies for this exam.

Signature: _____

Question:	1	2	3	4	5	6	Total
Points:	23	11	11	7	7	11	70
Score:							

1. Base Conversions: Convert the following numbers.
   (a) (2 points) 52_{10} to 8 bit binary (base 2)

   Solution: 00110100

   (b) (2 points) -52_{10} to 8 bit sign-magnitude binary.

   Solution: `10110100`
   Use positive value in part (a), and change sign bit to 1 to represent negative number.

   (c) (2 points) -52_{10} to 8 bit 2's complement binary.

   Solution: `11001011 + 1 = 11001100`

   Use positive value in part a, flip all bits, add 1.

   (d) (2 points) 011010110100_2 to octal (base 8)

   Solution:
   ```
   bin:011 010 110 100
   oct: 3   2    6   4
   ```
   Common error: grouping by 4 bits instead of 3 bits. You group by 4 to transform to hex, group by 3 for octal.

   (e) (3 points) $E7D_{16}$ to binary (base 2)

   Solution:
   ```
     E    7   D
   1100 0111 1101
   ```

   (f) (4 points) Encode "Zu@" as a C-style string. Give your answer as either hex or binary.

   Solution: binary: 0101 1010 0111 0101 0100 0000 0000 0000
   hex: 5A754000

(g) (4 points) -31.75_{10} to IEEE single precision (32 bit) floating point decimal number.

Solution: negative so sign bit is 1
`1 10000011 11111100000000000000000`

(h) (4 points) 108_{10} to base 4.

Solution: 1230_4

```
108/4 => 27 R0
27/4 => 6 R3
6/4 => 1 R2
1/4 => 0 R1
0 => stop
```

Common error: taking the remainder on the next to last division and carrying that into the last division to calculate 2/4 and ending up with the final answer 2230 instead of 1230.

2. Code Snippets. For each of the following prompts, write a snippet of code (no need for a complete function or program) which accomplishes the task. You can choose variable names unless otherwise specified in the prompt.

(a) (2 points) Write the code to initialize a character array containing all digits needed (0 thru F) to convert a number to hexadecimal.

Solution:

```
char digits[16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8',
  '9', 'a', 'b', 'c', 'd', 'e', 'f'};
```

Common error: adding the character `'\0'` to the end of the array. You only need the nul character for terminating strings. if you have an array of characters, you don't need the nul character at the end (just like you wouldn't need it for an array of ints).

(b) (1 point) Rewrite the following code using a character string:

```
char word[] = {'H','e','l','l','o','!','\0'};
```

Solution: either: `char* str = "Hello!";` `char str[] = "Hello!";`

Common error: combining both the forms of the declaration together (eg. `char* str[] = ...`)

(c) (4 points) Write the code to initialize all the elements in the following array to zeros.

```
int matrix[3][3];
```

Solution: I accepted either the literal value assignment statement:

```
int matrix[3][3] = {{0,0,0},
                    {0,0,0},
                    {0,0,0}};
```

I also accepted a nested loop answer:

```
int row, col;
for(row = 0; row < 3; row++) {
  for(col = 0; col < 3; col++) {
    matrix[row][col] = 0;
  }
}
```

(d) (4 points) Write code to prompt the user to enter their first name. Then read and store their name into a character array.

Solution: This problem is largely taken from the `String_notes.txt` for HW2 (cipher HW)

```
printf("Enter your name: ");
char name[30]; //any reasonable size ok
scanf("%s", name);
       or
fgets(name, sizeof(name), stdin);
```

Scoring:
+1 for prompt
+1 for buffer declaration
+2 for correct reading of input

Common errors: leaving the size out of the array declaration, reading in a character, `%c`, instead of a string, `%s`, with `scanf`, leaving out the size as the 2nd argument to `fgets`.

Note that `scanf` is only acceptable because you're reading a first name only which shouldn't have whitespace in it. If you were reading a set of first name, last name (or anything with spaces in it), you would have to use `fgets`.

3. Explain things to me.

(a) (6 points) The C programming language is both criticized and praised for the freedom it gives the programmer. List 2 specific examples of how C gives the programmer "freedom" and how the programmer must program defensively to avoid errors. You can include code snippets to illustrate your examples if you feel the need.

Solution: Solutions vary, but examples we have discussed include:

* No bounds checking on arrays (buffer overflows). This means that the programmer needs to make sure not to go beyond the end of the array. Storing the length of the array into a variable and then using that variable in all situations relating to the array is a good technique.
* No type-checking for casting variables. This means that the programmer should always use explicit casting operators when going from a more precise datatype (such as a float/double) to

a less precise one (such as an int/char).
* Input via `gets` (discussed in notes and HW2).
* Arrays are uninitialized. Efficient but dangerous. The programmer should either 1) initialize the array themselves or 2) make sure that user entered data will COMPLETELY initialize the values in the array (this may involve "trimming" the array after the user has finished entering data or dynamically allocating the array as the user enters data).
* function declaration order. C will assume declaration of functions which are used before the prototype occurs. This can lead C to make incorrect assumptions about the function which can cause problems (particularly around datatypes of parameters or return values). The solution is to declare and define functions before they are called or to include their prototypes in a .h file (not discussed yet in class but discussed in some readings).

Scoring:
+1 for each example and +2 for each explanation

(b) (2 points) What is the difference between the following statements? What type of data can each store?

```
int* ptr;        int * ptr;        int *ptr;
```

Solution: All statements are the same since C ignores whitespace in this context. Each stores a pointer (address) to an int in memory.

(c) (3 points) Recall the canonical model of a program's address space, which is conceptually a big array of bytes divided into three regions: code and static data, heap, and stack. Draw a picture of the canonical address space. Label each of the three regions.

Solution:

```
A canonical 32-bit address space (taken directly from the course notes)

-----------------------------  Byte index 0
|    Code and static data   |
-----------------------------
|           Heap            |
|  (dynamically-allocated)  |
-----------------------------
|             |             | Heap grows and shrinks as data is dynamically
|             v             | allocated and freed by the program (new  and
|                           | garbage collection in Java, malloc and free
|     Unallocated space     | in C).
|                           |
|             ^             | Stack grows and shrinks as the program calls and
|             |             |  returns from functions.
-----------------------------
|           Stack           |
|   (local variables and    |
|  function call history)   |
-----------------------------  Byte index 2^32 - 1
```

Common error: Swapping the "unallocated space" and "static data and code" sections of memory.

4. Professor Summet is trying to write a method which will increment a variable's value by a specified value. Unfortunately, her program isn't doing what she wants:

```
//** Prof. Summet's version **//
void incrementVariable1(int x, int y) {
  x = x + y;
}

int main() {
  int var = 5;
  incrementVariable1(var, 2);
  printf("result of incrementVariable1 is %d\n", var);

  return 0;
}
```

(a) (1 point) What value for `var` will the program display when run as currently written? 5

(b) (4 points) Time for you to take over. Complete the version below so that it will function as described.

```
void incrementVariable2(                                ) {
```

Solution:

```
  void incrementVariable2(int* x, int y) {
    *x = *x + y;
  }
```

```
}
```

(c) (2 points) Add a call in `main` (after Prof. Summet's call to `incrementVariable1`) which would correctly use your function to increase `var`'s value by 4.

Solution: `incrementVariable2(&var, 4);`

5. (7 points) Assume the following program compiles and runs to completion. Give the output of the following program. If output is unknown, you can simply describe as much as you can about the data or why it is unknown.

```
#include <stdio.h>

int main() {
  char str[] = "Hello\0CMS230\0";
  int num = -5;

  printf("1: %s\n", str);
  printf("2: %c\n", str[6]);
  printf("3: %c\n", str[15]);

  printf("4: %p\n", &num);
  printf("5: %d\n", *(&num));

  if(num) {
    printf("6: Halloween\n");
  } else {
    printf("6: Boo!\n");
  }

  printf("7: Free points!\n");
  return 0;
}
```

Solution:

```
1: Hello    //%s as format string will only print to nul character
2: C  //nul char is 1 character
3: unknown; this accesses memory outside of the array bounds and
prints whatever it finds there as a char
4: unknown; this will print the address of num
5: -5  //dereferences the address of num, leading back to the int value
6. Halloween //any non-zero value is "true" in C (doesn't have boolean
type)
7. Free points!
```

1pt per line of output.

6. (11 points) Write a function named `printNums` which takes a single int as a parameter. This function should print a pyramid as shown below which is the number of rows high as specified by the parameter.

   Examples:

   `printNums(5)` outputs:

   ```
   ....1
   ...22
   ..333
   .4444
   55555
   ```

   `printNums(3)` outputs:

   ```
   ..1
   .22
   333
   ```

   `printNums(7)` outputs:

   ```
   ......1
   .....22
   ....333
   ...4444
   ..55555
   .666666
   7777777
   ```

Solution: Solutions vary. A sample answer:

```
void printNums(int h) {
  int dots = h - 1; //number of dots to print
  int nums = 1;  //number of nums to print

  for(i=0; i < h; i++) {   //each iteration prints a row
    for(j = 0; j < dots; j++) {  //print the dots
      printf(".");
    }

    for(j = 0; j < nums; j++) { //print the numbers
      printf("%d", nums);   // or (i+1)
    }

    printf("\n");
    dots--;
    nums++;
  }
}
```

Scoring:
+2: function header (all parts)
+2: outer loop
+2: inner loop(s) run correct number of times
+2: prints correct number of .
+2: prints correct number of numbers and correct number
+1: ends lines at correct place (newline chars)

Reference Material

Excess-127 Encoding

Bit Pattern	Value Encoded
00000000	-127
00000001	-126
...	...
01111111	0
10000000	1
10000001	2
...	...
11111111	128

Fractions and decimal equivalents

Fraction	Decimal Value
$\frac{1}{2}$	.5
$\frac{1}{4}$	.25
$\frac{1}{8}$	.125
$\frac{1}{16}$	.0625
$\frac{1}{32}$	.03125

`printf` format strings:

Syntax	Datatype
%i, %d	integer
%f	double, float
%c	char
%s	string
%x, %X	hex rep.
%p	pointer

ASCII chart

Dec	Hex	Char
000	00	(nul)
001	01	(soh)
002	02	(stx)
003	03	(etx)
004	04	(eot)
005	05	(enq)
006	06	(ack)
007	07	(bel)
008	08	(bs)
009	09	(tab)
010	0A	(lf)
011	0B	(vt)
012	0C	(np)
013	0D	(cr)
014	0E	(so)
015	0F	(si)
016	10	(dle)
017	11	(dc1)
018	12	(dc2)
019	13	(dc3)
020	14	(dc4)
021	15	(nak)
022	16	(syn)
023	17	(etb)
024	18	(can)
025	19	(em)
026	1A	(eof)
027	1B	(esc)
028	1C	(fs)
029	1D	(gs)
030	1E	(rs)
031	1F	(us)

Dec	Hex	Char
032	20	␣
033	21	!
034	22	"
035	23	#
036	24	$
037	25	%
038	26	&
039	27	'
040	28	(
041	29	)
042	2A	*
043	2B	+
044	2C	’
045	2D	-
046	2E	.
047	2F	/
048	30	0
049	31	1
050	32	2
051	33	3
052	34	4
053	35	5
054	36	6
055	37	7
056	38	8
057	39	9
058	3A	:
059	3B	;
060	3C	<
061	3D	=
062	3E	>
063	3F	?

Dec	Hex	Char
064	40	@
065	41	A
066	42	B
067	43	C
068	44	D
069	45	E
070	46	F
071	47	G
072	48	H
073	49	I
074	4A	J
075	4B	K
076	4C	L
077	4D	M
078	4E	N
079	4F	O
080	50	P
081	51	Q
082	52	R
083	53	S
084	54	T
085	55	U
086	56	V
087	57	W
088	58	X
089	59	Y
090	5A	Z
091	5B	[
092	5C	\
093	5D	]
094	5E	^
095	5F	_

Dec	Hex	Char
096	60	‘
097	61	a
098	62	b
099	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL