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 5 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	Total
Points:	23	8	6	12	21	70
Score:						

1. Base Conversions: Convert the following numbers.
   (a) (2 points) 85_{10} to 8 bit binary (base 2)

   Solution: 01010101

   (b) (2 points) -57_{10} to 8 bit sign-magnitude binary.

   Solution: 10111001

   Convert positive value to binary, extend to 8 bits, and change sign bit to 1 to represent negative number.

   (c) (2 points) -57_{10} to 8 bit 2's complement binary.

   Solution: 11000111

   Convert positive value to binary, extend to 8 bits, flip all bits, add 1.

   (d) (2 points) 011000111101_2 to octal (base 8)

   Solution:

   ```
   bin:011 000 111 101
   oct: 3   0   7   5
   ```

   (e) (3 points) $C2F_{16}$ to binary (base 2)

   Solution:

   ```
   hex: C    2    F
   bin: 1100 0010 1111
   ```

   (f) (4 points) -12.875_{10} to IEEE single precision (32 bit) floating point decimal number.

   Solution: negative so sign bit is 1
   12.875 = 1100.111 in binary
   normalize to 1.100111 (exponent 3)
   convert exponent to 8 bit excess 127: 10000010
   drop the leading 1, so mantissa is: 100111

   complete answer: 1 100000010 1001110...0 (17 trailing 0s)

   (g) (4 points) 108_{10} to base 5.

   Solution: 113_5

   ```
   108/5 => 21 R3
   21/5 => 4 R1
   1/5 => 0 R1
   0 => stop
   ```

(h) (4 points) Encode "Hi!" as a C-style string. Give your answer as either hex or binary.

Solution: 0x: 48 69 21 00
bin: 0100 1000 0110 1001 0010 0001 0000 0000
Scoring: 1pt per character.

2. For each of the following, select the single best answer.

   (a) (1 point) Which of the following expressions gives the value stored at the address pointed to by the pointer (reference variable)`a`?

   A. `a`
   B. `*a`
   C. `&a`
   D. `val(a)`
   E. `*(*a)`

   (b) (1 point) Consider the following snippet of code which Dr. Summet has written to print out 20 integer values:

```
int data[20];

int i;
for(i = 0; i <= 20; i++) {
  printf("%i ", data[i]);
}
```

   When Dr. Summet tries to compile her code with `gcc -o firstTry firstTry.c` and run it, what will happen?

   A. The code will contains a syntax error and not compile.
   B. The code will compile, and print 20 integers.
   C. The code will compile, but will not print all 20 integers.
   D. The code will compile and print too many integers.
   E. There are no errors and the code will run as Dr. Summet expects.

   (c) (1 point) Dr. Summet is trying again. Now she has written the following code and wants to count the number of 1's in her array.

```
int data[] = {1, 2, 1, 1, 2};

int sum = 0;
int i;
for(i = 0; i < 5; i++) {
  if(data[i] = 1) {
    sum++;
  }
}
printf("sum is %d\n", sum);
```

   When Dr. Summet tries to compile her code with `gcc -o again again.c` and run her code, what will happen?

   A. The code will contains a syntax error and not compile.
   B. The code will compile and print `sum is 5`.
   C. The code will compile and demonstrate the correct behavior (ie, print `sum is 3`).
   D. The code will compile and print `sum is 7`.
   E. The code will compile and print `sum is 10`.

   (d) (1 point) Which of the following expressions gives the memory address of the integer variable `a`?

   A. `*a`
   B. `a`
   C. `&a`
   D. `address(a)`

E. `*(*a)`

(e) (1 point) Which of the following expressions gives the memory address of a variable pointed to by the pointer (reference variable) `a`?

A. `a`
B. `*a`
C. `&a`
D. `address(a)`
E. `*(*a)`

(f) (1 point) Which of the following is the proper declaration of a pointer in C?

A. `int x;`
B. `int &x;`
C. `ptr x;`
D. `int* x;`
E. `*(int*)x;`

(g) (1 point) Which of the following is the correct way in C to declare an array of three strings and initialize it to contain three strings?

A. `string animals[3] = {"cat", "dog", "giraffe"};`
B. `string* animals[3] = {"cat", "dog", "giraffe"};`
C. `char animals[3] = {"cat", "dog", "giraffe"};`
D. `char* animals[3] = {"cat", "dog", "giraffe"};`
E. `char** animals[3] = {"cat", "dog", "giraffe"};`

(h) (1 point) How many bytes are allocated by the definition below?

```
char txt [20] = "Hello world!\0";
```

A. 12 bytes
B. 13 bytes
C. 14 bytes
D. 20 bytes
E. 21 bytes

3. (6 points) Give 2 similarities and 2 differences between Java and C. These similarities/differences should be substantial differences, not syntax differences.

Solution: Similarities:
* both are typed languages (data must have a type associated with it)
* both provide largely the same syntax for basic operations (variable declaration, basic math, loops, conditionals, etc).

Differences:
* C provides pointers
* C is not object-oriented while Java is (C is procedural)
* Java is "write-once, run anywhere" due to the Java Virtual Machine while C is chip/platform specific.
* Java sourcecode is first compiled to byte code (`.class`) which is then interpretted. C sourcecode is compiled directly to machinecode.

* Java includes automatic memory management and garbage collection. In C, the programmer is responsible for cleaning up. (Haven't really discussed this as much, but it's in your reading.)

4. (12 points) Write a function named `multTable` which prints the multiplication table shown below. Hint: there are 2 spaces between 2 digit numbers, but 3 spaces between single digit numbers to maintain the column alignment. Each row begins with 3 spaces.

```
  1   2   3   4   5   6   7   8   9
  2   4   6   8  10  12  14  16  18
  3   6   9  12  15  18  21  24  27
  4   8  12  16  20  24  28  32  36
  5  10  15  20  25  30  35  40  45
  6  12  18  24  30  36  42  48  54
  7  14  21  28  35  42  49  56  63
  8  16  24  32  40  48  56  64  72
  9  18  27  36  45  54  63  72  81
```

Solution: Solutions vary, but one is:

```
void multTable() {
  int i, j;
  for(i = 1; i < 10; i++) {
    for(j = 1; j < 10; j++) {
      int prod = i * j;
      if(prod < 10) {
        printf("   %d"); //print 3 spaces and then value
      } else {
        printf("  %d"); //print 2 spaces and then value
      }
    }
    printf("\n");  //print newline to start next row of chart
  }
}
```

Scoring:
+1: function definition correct
+2: nested for loops which print something
+2: nested loops have correct bounds (run from 1 to 9 inclusive)
+2: prints product
+2: utilizes correct spacing (2 or 3 as appropriate)
+2: prints rows correctly (newline at end of each row only)
+1: syntax correct (deducted for repeated substantial mistakes)

5. (10 points) Consider the following program:

```
#include <stdio.h>
void functionOne(int a, int b) {
  a = 10;
  b = 20;
  printf("one: a %i, b %i\n", a, b);
}
void functionTwo(int *ptr_a, int *ptr_b) {
  *ptr_a = 1000;
  *ptr_b = 2000;
}
void functionThree(int *ptr_a, int *ptr_b) {
  int c = 10000;
  int d = 20000;
  ptr_a = &c;
  ptr_b = &d;
}
int main() {
  int a = 100;
  int b = 200;
  printf("main1: a %i,b %i\n", a, b);
  functionOne(a, b);
  printf("main2: a %i, b %i\n", a, b);
  functionTwo(&a, &b);
  printf("main3: a %i, b %i\n", a, b);
  functionThree(&a, &b);
  printf("main4: a %i, b %i\n", a, b);
  return 0;
}
```

(a) (7 points) Give the output which is printed to the screen when the following program is run.

Solution:

```
main1: a 100, b 200
one: a 10, b 20
main2: a 100, b 200
main2: a 1000, b 2000
main3: a 1000, b 2000
```

Scoring: 1 pt. each for lines 1, 2, and 5. 2pts for others.

(a) (4 points) Explain why `functionThree` does not change the values of `a` and `b` in `main`.

Solution: C is a pass-by-value language, so changes made to the values of parameter variables do not effect calling variables (ever!). The last 2 lines of `functionThree` set the values of parameter variables to new values (which happen to be the addresses of the ints declared on the 2 previous lines). Once `functionThree` is finished, the parameter variables are removed from the stack and cease to exist. This means that `a` and `b` in `main` retain their original values.

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