Name: _____

ECE 222: System Programming Concepts **Fall 2015**
Exam 2 **Monday, November 2**

For all code given on this test,

- Assume all necessary library header files are included.
- Assume the memory is byte-addressable with the big-endian format and a 32-bit architecture.
- Assume all memory is filled contiguously upon declaration.

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
(nul)	0	0x00	(sp)	32	0x20	@	64	0x40	`	96	0x60
(soh)	1	0x01	!	33	0x21	A	65	0x41	a	97	0x61
(stx)	2	0x02	"	34	0x22	B	66	0x42	b	98	0x62
(etx)	3	0x03	#	35	0x23	C	67	0x43	c	99	0x63
(eot)	4	0x04	$	36	0x24	D	68	0x44	d	100	0x64
(enq)	5	0x05	%	37	0x25	E	69	0x45	e	101	0x65
(ack)	6	0x06	&	38	0x26	F	70	0x46	f	102	0x66
(bel)	7	0x07	'	39	0x27	G	71	0x47	g	103	0x67
(bs)	8	0x08	(	40	0x28	H	72	0x48	h	104	0x68
(ht)	9	0x09	)	41	0x29	I	73	0x49	i	105	0x69
(nl)	10	0x0a	*	42	0x2a	J	74	0x4a	j	106	0x6a
(vt)	11	0x0b	+	43	0x2b	K	75	0x4b	k	107	0x6b
(np)	12	0x0c	,	44	0x2c	L	76	0x4c	l	108	0x6c
(cr)	13	0x0d	-	45	0x2d	M	77	0x4d	m	109	0x6d
(so)	14	0x0e	.	46	0x2e	N	78	0x4e	n	110	0x6e
(si)	15	0x0f	/	47	0x2f	O	79	0x4f	o	111	0x6f
(dle)	16	0x10	0	48	0x30	P	80	0x50	p	112	0x70
(dc1)	17	0x11	1	49	0x31	Q	81	0x51	q	113	0x71
(dc2)	18	0x12	2	50	0x32	R	82	0x52	r	114	0x72
(dc3)	19	0x13	3	51	0x33	S	83	0x53	s	115	0x73
(dc4)	20	0x14	4	52	0x34	T	84	0x54	t	116	0x74
(nak)	21	0x15	5	53	0x35	U	85	0x55	u	117	0x75
(syn)	22	0x16	6	54	0x36	V	86	0x56	v	118	0x76
(etb)	23	0x17	7	55	0x37	W	87	0x57	w	119	0x77
(can)	24	0x18	8	56	0x38	X	88	0x58	x	120	0x78
(em)	25	0x19	9	57	0x39	Y	89	0x59	y	121	0x79
(sub)	26	0x1a	:	58	0x3a	Z	90	0x5a	z	122	0x7a
(esc)	27	0x1b	;	59	0x3b	[	91	0x5b	{	123	0x7b
(fs)	28	0x1c	<	60	0x3c	\	92	0x5c	|	124	0x7c
(gs)	29	0x1d	=	61	0x3d	]	93	0x5d	}	125	0x7d
(rs)	30	0x1e	>	62	0x3e	^	94	0x5e	~	126	0x7e
(us)	31	0x1f	?	63	0x3f	_	95	0x5f	(del)	127	0x7f

1. (10 pts) Pointer and Integer Arithmetic

Assume that the starting address for the variable **art[2]** begins at (decimal) **100**.

```
struct Square {
   int area;
   int corners[4];
}
struct Object {
   int volume;
   char *name;
   struct Square pods[3];
};

int main(void)
{
  struct Object art[2];
  struct Object *sold;
  struct Square *copy;
  int *paint;

  sold  = &art[0] + 1;
  copy  = &art[0].pods[0] + 1;
  paint = &art[1].pods[1].corners[1] + 1;
  printf("base address: %u\n", (unsigned int) &art);
  printf("sold: %u\n", (unsigned int) sold);
  printf("copy: %u\n", (unsigned int) copy);
  printf("paint: %u\n", (unsigned int) paint);

}
```

Show your output here (the address of **art** is given, fill in the remaining values):

```
base address: 100
sold:
copy:
paint:
```

2. (20 pts.) Fill in the memory map for the following code. Show all address ranges and all values during execution.

```
struct Colors {
  int trim[2];
};
struct Car {
  int doors;
  struct Colors body;
  int *link;
};
int i = 1;
struct Car race;
int decal[3] = {2, 3, 4};
struct Car *toy, sport[2];
sport[0].doors = 5;         sport[1].doors = 6;
sport[0].body.trim[0] = 7; sport[1].body.trim[0] = 8;
sport[0].body.trim[1] = 9; sport[1].body.trim[1] = 10;

1: sport[0].link = &sport[1].doors;
2: sport[1].link = decal + 4;
3: toy = &race + 2;
4: race = *toy;
5: race.link = &sport[0].doors;
6: toy->body.trim[1] = 11;
7: (*(toy+1)).doors = 12;
8: decal[0] = (sport + 1)->doors + 1;
9: race.doors = toy->link->doors;
```

Label	Address	Value
i	1000	1
race		

3. (20 pts.) Dynamic Memory for Structures

Use the declarations below to write the code necessary to store the details about the textbook for **ece222**. Store the string "System Programming" in memory pointed to by **title**, store the strings "Adam" and "Hoover" as the author's first and last names, and store "Professor", and "Ph.D." into the **rank** and **degree.** Do not use any other variables other than **ece222** which is already declared.

```
struct Person {
  char *first;
  char *last;
};
struct Faculty {
  char rank[10];
  char degree[10];
  struct Person *name;
};
struct Book {
  char title[20];
  struct Faculty *author;
  float cost;
};
int main(void)
{
   struct Book *ece222
```

For the next **four** problems consider code to describe a signal captured by a software-defined radio.

```
struct Complex{
      double real, imag;
};
struct Signal {
      int num_samples;
      struct Complex *samples;
};
int main(int argc, char argv[]) {
   struct Complex c;
   c.real = 0.707; c.imag = 0.707;
   struct Signal waveform;
   int nsam = atoi(argv[1]);
   waveform.num_samples = nsam;
   waveform.samples = (struct Complex *)calloc(nsam, sizeof(struct Complex));
   printf("Example magnitude = %g\n", magnitude(c));

   // add new functions
}
```

4. (10 pts) Magnitude

Write the function **`magnitude(c)`** to find the magnitude of a complex number. For a complex number, c, the magnitude is the square root of the sum of the squares of the real and imaginary parts. That is, $c = (r, i)$ and the magnitude is $|c| = \sqrt{(r)^2 + (i)^2}$. You can use the C-functions `pow(a,b)` which returns a^b, and `sqrt(a)` which returns $\sqrt{a}$ where `a`, `b`, and the return values of these two functions are of type `double`.

```
double magnitude(
{
```

5. (20 pts) Maximum Waveform Magnitude

Write one function **`max_mag()`** to find the sample in the signal with the largest magnitude. The function must return **both** the maximum magnitude (as a **`double`**) and the index of the sample with the maximum value (as an **`int`**).

6. (20 pts.) Copy and Smooth Filtering Function

Write a function called **`filter()`** that takes as input a signal and produces a smoothed copy of the signal. The return value of the function is a pointer to the new filtered signal. The filter function takes each sample in the original signal and produces an average of that sample, the previous sample, and the next sample. That is, for a sample at index position **`i`**, calculate the sum of the real parts of the sample at positions **`i-1`**, **`i`**, and **`i+1`** and divide by **`3`**. Repeat for the imaginary part of the sample. However, the new signal should just contain a copy of the first and last samples in the original signal, because it does not make sense to average the first or last values.