Name: ______________________

ECE 222: System Programming Concepts **Fall 2012**
Exam 2 **Thursday, November 1**

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.) Complete the memory map for the following code.

```
int array[5], what[5];
int *ptr;
int offset, j;

for (j=0; j<5; j++)
     array[j] = 2 * j + 10;

ptr = &(array[0]);
offset = 4;

what[0] = *(ptr + offset);
what[1] = ptr[offset - 3];
what[2] = *(array + offset);
what[3] = *ptr+offset;
what[4] = *ptr++;
```

label	address	value
array[0]	1000	10
array[1]	1004	12
array[2]	1008	14
array[3]	1012	16
array[4]	1016	18
what[0]	1020	
what[1]	1024	
what[2]	1028	
what[3]	1032	
what[4]	1036	
ptr	1040	
offset	1044	
j	1048	~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

2. (10 pts) Pointer and Integer Arithmetic

Show the output of the print statement in this code. Give a memory map with values.

```
struct s
{
   double d;
   char *str;
   float f;
   char c[5];
};

int main(void)
{
  struct s a, *t;
  int i, j;

  t = &a;
  i = (int)t;
  j = (int)++t;
  j = j+1;
  i = j-i;

  printf("d = %d\n", i);
}
```

d =

Label	Address	Value

3. (30 pts.) Memory Map

Fill in the memory map for the following code. Show all address ranges and all values during execution.

```
struct s { int i; double d; double *e; int *j; };

struct s *st1;
double d;
struct s st2[2];
int i;
struct s st3;
double *g;
double e[2];

*e = 1.1;
i = sizeof(struct s);
st2[0].d = sizeof(st3.e);
st3.j = &st2[1].i;
*(st3.j) = 2.2;
st3.j--;
*(st3.j) = 3.3;
g = &st3.d;
st1 = &st2[0];
st1->e = g;
*(st1->e) = 4.4;
(*st1).i = 5.5;
st2[0].e = &(st1->d);
*(st2[0].e) = 6.6;
st2[1].d = *((*st1).e);
```

LABEL	ADDRESS	Value
	1000	

LABEL	ADDRESS	Value

4. (10 pts) Dynamic Memory.

What is the problem with the following code?

```
int i, *ptri;
ptri = (int *)malloc(10*sizeof(int));
for (i = 0; i < 10; i++)
     ptri[i] = 2 * i;
ptri = (int *)malloc(100*sizeof(int));
```

5. (10 pts.) Dynamic Memory for Structures

Write the code necessary to dynamically allocate an **`s`** structure named **`mystruct`** and then set the element **`d`** to **`1.0`**, **`i`** to **`2`**, and store the string "**`CLEMSON`**" at **`a`**, given the structure **`s`** below:

```
struct s { char *a; double d; int i; };
```

6. (10 pts.) Passing Values with Functions

The following program fails to print the correct output. The correct output is

```
9/2 = 4 with 1 remainder
```

Fix the bugs. You should not re-write the code but instead just mark the corrections on the printed code. Your solution must use the **`division()`** function and you cannot add any new lines of code or new functions.

```
void division(int  numerator, int  denominator,
              int  dividend,  int  remainder    )
{
     if (denominator == 0) return;
     dividend   =  numerator  /  denominator  ;
     remainder  =  numerator  %  denominator  ;
}
int main(void)
{
     int  x, y ;
     int  div  ;
     int  rem  ;
     x  =  9;
     y  =  2;
     division (  x   ,  y   ,  div   ,  rem  );

     printf("%d/%d = %d with %d remainder\n", x , y , div , rem );
}
```

For the next two problems assume these structure definitions and **main()** code.

```
struct Point {
   double x;
   double y;
};
struct StraightLine {
   struct Point start;
   struct Point end;
   double distance;
};
struct Triangle {
   struct Point p1, p2, p3;
   double perimeter;
};

int main(void)
{
   struct StraightLine line;
   struct Triangle     tri;

   // assume the values in line and tri have been set correctly
   line.distance = euclidian(line.start, line.end);
   calculate_perimeter( &tri );
}
```

7. (10 pts) Distance Function

Write the function **euclidian()** that takes the starting point and ending point of a line and returns the Euclidian (or "ordinary") distance between the two points. The Euclidean distance $d(\boldsymbol{p},\boldsymbol{q})$ between points $\boldsymbol{p} = (p_1, p_2)$ and $\boldsymbol{q} = (q_1, q_2)$ is the length of the line segment connecting them: $d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$. 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`.

Note: your function must match the types of the arguments used here:

```
double euclidian(
{
```

8. (10 pts.) Perimeter Function

Write the function **`calculate_perimeter()`** that takes a pointer to a **`struct Triangle`** structure and sets the member call **`perimeter`** in the structure to the sum of the lengths of the sides of the triangle. You may use the **`euclidian()`** function from Problem 7.

Note that your function must match the parameter type as used in the **`main()`** function on the previous page.

```
void calculate_perimeter(
{
```