Login Name _____ **Name _____**

Signature _____ **Student ID _____**

Midterm
CSE 131
Winter 2015

Page 1	**_____ (30 points)**
Page 2	**_____ (26 points)**
Page 3	**_____ (28 points)**
Page 4	**_____ (28 points)**
Page 5	**_____ (30 points)**
Subtotal	**_____ (142 points = 100%)**
Page 6 Extra Credit	**_____ (8 points)**
Total	**_____**

This exam is to be taken by yourself with closed books, closed notes, no electronic devices.
You are allowed one side of an 8.5"x11" sheet of paper handwritten by you.

1. Given the following CUP grammar snippet (assuming all other Lexing and terminals are correct):

```
Expr ::= Des {: System.out.println("A"); :} T_ASSIGN Expr {: System.out.println("B"); :}
    | Des {: System.out.println("C"); :}
    | T_INT_LITERAL {: System.out.println("D"); :}
    ;

Des  ::= T_STAR {: System.out.println("E"); :} Des {: System.out.println("F"); :}
    | T_AMPERSAND {: System.out.println("G"); :} Des {: System.out.println("H"); :}
    | T_SIZEOF {: System.out.println("I"); :} T_LPAREN Des T_RPAREN {: System.out.println("J"); :}
    | T_PLUSPLUS {: System.out.println("K"); :} Des {: System.out.println("L"); :}
    | Des2
    ;

Des2 ::= Des2 {: System.out.println("M"); :} T_PLUSPLUS {: System.out.println("N"); :}
    | Des2 {: System.out.println("O"); :} T_LBRACKET Expr T_RBRACKET {: System.out.println("P"); :}
    | Des3
    ;

Des3 ::= T_ID
    ;
```

What is the output when parsing the follow expression (you should have 22 lines/letters in your output) [22 pts]:

```
*x[ *y ]++ = ++z[ sizeof(y[0]) ];
```

Output

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

In the above grammar, does the assignment operator have left-to-right or right-to-left associativity? [2 pts]

_____

If variable `z` is defined to be type `float*`, what type must variables `x` and `y` be defined for this expression to be semantically correct? **Do not** define these variables using arrays. [6 pts]

_____ x;

_____ y;

float * z;

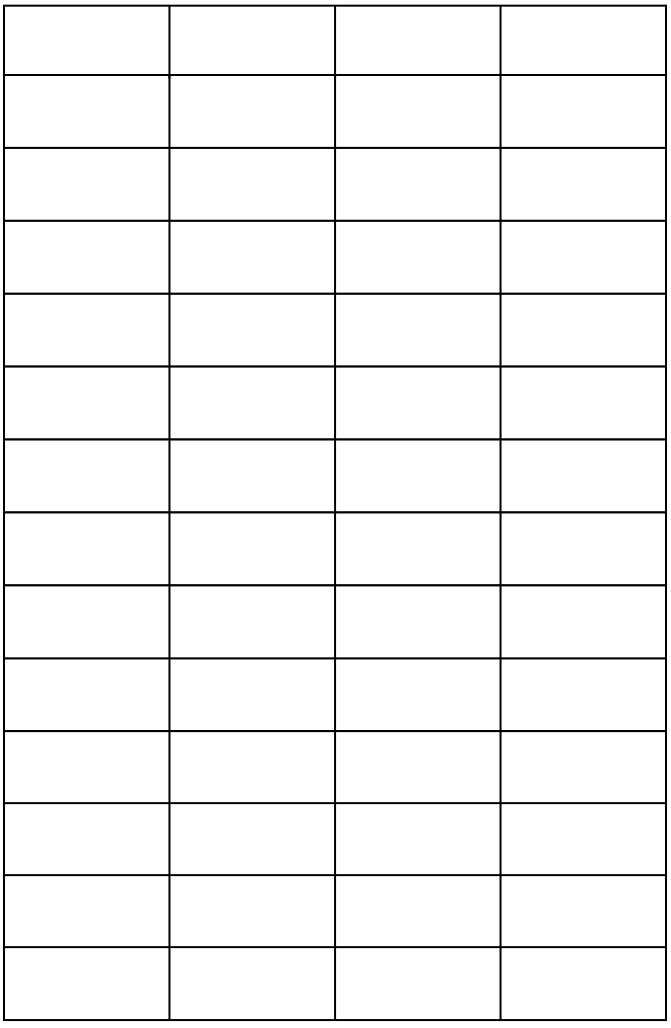
2. Show the memory layout of the following C struct definition taking into consideration the **SPARC** data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct member/field name. For example, if member/field name `p` takes 4 bytes, you will have 4 `p`'s in the appropriate memory locations. If the member/field is an array, use the name followed by the index number. For example, some number of `p[0]`, `p[1]`, `p[2]`, etc. If the member/field is a struct, use the member name followed by its member names (e.g. `p.a`, `p.b`, etc.). If the member/field is an array of structs, include both the index and member name (e.g. `p[0].a`). Place an `X` in any bytes of padding. Structs/unions are padded so the total size is evenly divisible by the strictest alignment requirement of its members. [16 pts figure; 10 pts questions]

```
struct bird {
  char        a;
  short       b;
  int         c;
};

struct tree {
  char        d;
  struct bird e[2];
  double      f;
  char        g;
};

struct tree redwood;
```

low memory
redwood:



high memory

What is the `sizeof( struct tree )`? _____

What is the `offsetof( struct tree, e[1].b )`? _____

If `struct tree` had been defined as `union tree` instead, what would be `sizeof( union tree )`? _____

If you rearrange the order of the struct members in `struct tree` to minimize padding, what would be the size of this modified `struct tree`? _____

How many bytes are saved by rearranging the `struct tree` members to minimize padding? _____

3. Given the following pseudocode, determine the program output based on the specified scoping rule: [28 pts]

```
int x = 0;        -- global var declaration
int y = 0;        -- global var declaration

void sum_xy( int a, int b )
      x = x + a;
      y = y + b;

void gamma()
      sum_xy( 10, 10 );
      cout << x << endl;
      cout << y << endl;

void beta()
      int y = 7;  -- local var declaration
      sum_xy( 3, 3 );
      gamma();
      cout << x << endl;
      cout << y << endl;

void alpha()
      int x = 1;  -- local var declaration
      sum_xy( 7, 7 );
      beta();
      cout << x << endl;
      cout << y << endl;

sum_xy( 2, 2 );
alpha();
beta();
gamma();
cout << x << endl;
cout << y << endl;
```

What does the program output if the language uses <u>static</u> scoping?

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

What does the program output if the language uses <u>dynamic</u> scoping?

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

4. Given the C array declaration on a SPARC architecture

C

```
char a[2][3][5];
```

What is the `sizeof( a )`? _____ [2 pts]

Mark with an **A** all the memory location(s) where we would find the array element `a[1][0][4]`: [4 pts]

a:



lower memory higher memory

(Each box represents a byte in memory)

Given the following Reduced-C code:

```
structdef RECA {
      RECA  * ptr1[5];
      float * ptr2[5];
};
structdef RECB {
      RECB  * ptr1[5];
      RECA  * ptr2[5];
      bool  * ptr3[5];
};
RECB *  ptr1[5][5];
RECA ** ptr2[5];
```

Indicate the data type of the following expressions and circle what kind of value it is: [22 pts]

	Data Type:	*Kind of Value (circle one):*
`++ptr1[0][1]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`ptr2[0][1][2]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`ptr2[0][1][2].ptr2[3][4]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`&ptr1[0][1]->ptr1[2][3]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`(*+++*++ptr2[0]).ptr1`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`-ptr2[0][1]->ptr2[2][3]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`!*ptr1[0][1]->ptr3[2]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`ptr1[0][1]->ptr1[2]->ptr2[3]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`*ptr2[0][1]->ptr1[2]->ptr1[3]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`&ptr2[0]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]
`&ptr1[0][1][2].ptr2[0]`	_____	[Mod L-Val / Non-Mod L-Val / R-Val]

5. Given the following C code, indicate which of the 5 main areas of the C Runtime Environment each expression is located in: [18 pts]

```
int A = 1, B = 5;
static int C = 1;

int * D ( int E ) {
    int F = 3;
    static int G = E;
    static float H;

    return &G;
}

int main ( int argc, char *argv[] ) {
    static int * I;
    I = (int *) calloc(1, sizeof(int));
    int * J = D(*I);
    int K[50];
    int ** L = &J;
    K[0] = **L;
    <------ determine runtime areas at this point in time
    free(I);
    return 0;
}
```

A	_____	J	_____
B	_____	K	_____
C	_____	L	_____
D	_____	main	_____
E	_____	argc	_____
F	_____	argv	_____
G	_____	*I	_____
H	_____	*J	_____
I	_____	*L	_____

State whether constant folding will be performed on one or more arithmetic expressions in the following statements by the compiler according to this quarter's Reduced-C spec (Y or N): [12 pts]

```
function : void main() {
    int a = 10;

    const int b = 5;

    int c = b + 5;                  _______

    const int d = b + b;            _______

    int e[d * 5];                   _______

    a = a + 2;                      _______

    c = b + 2;                      _______

    e[b + d] = a;                   _______

    e[-5 + (d / 2)]++;              _______

    c = e[a + b];                   _______

    c = e[a + 10];                  _______

    e[c * c] = b;                   _______

    c = (1 + 2) % 3;                _______

    e[e[b]] = e[e[4] + e[5]];       _______
}
```

Extra Credit

What gets printed when the following C program is executed? [8 pts]

```
#include <stdio.h>
int main() {
  char a[] = "GEORGE";
  char *p = a;

  printf( "%c\n", *++p = '-' );                    _______

  printf( "%c\n", (p[-1] -= 5) + 6 );              _______

  printf( "%c\n", 2[p] = --*(p - 1) );             _______

  printf( "%c\n", ++*--p + 14 );                   _______

  printf( "%c\n", (2[p] = *p + 2) + 12 );          _______

  printf( "%c\n", *(p = p + 4) += 18 );            _______

  printf( "%c\n", *++p = a[1] - 12 );              _______

  printf( "%s\n", a );              ____________________

  return 0;
}
```

A portion of the C Operator Precedence Table

Operator	Associativity
++ postfix increment -- postfix decrement [] array element () function call	L to R
* indirection ++ prefix increment -- prefix decrement & address-of sizeof size of type/object (type) type cast	R to L
* multiplication / division % modulus	L to R
+ addition - subtraction	L to R
.	
.	
= assignment	R to L

Hexadecimal - Character

00	NUL	01	SOH	02	STX	03	ETX	04	EOT	05	ENQ	06	ACK	07	BEL
08	BS	09	HT	0A	NL	0B	VT	0C	NP	0D	CR	0E	SO	0F	SI
10	DLE	11	DC1	12	DC2	13	DC3	14	DC4	15	NAK	16	SYN	17	ETB
18	CAN	19	EM	1A	SUB	1B	ESC	1C	FS	1D	GS	1E	RS	1F	US
20	SP	21	!	22	"	23	#	24	$	25	%	26	&	27	’
28	(	29	)	2A	*	2B	+	2C	,	2D	-	2E	.	2F	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	4C	L	4D	M	4E	N	4F	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5A	Z	5B	[	5C	\	5D	]	5E	^	5F	_
60	‘	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6A	j	6B	k	6C	l	6D	m	6E	n	6F	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7A	z	7B	{	7C	|	7D	}	7E	~	7F	DEL

Scratch Paper