Login Name _____ **Name _____**

Student ID _____ **Signature _____**

Final
CSE 131
Fall 2013

Page 1	**_____**	**(32 points)**
Page 2	**_____**	**(30 points)**
Page 3	**_____**	**(30 points)**
Page 4	**_____**	**(44 points)**
Page 5	**_____**	**(28 points)**
Page 6	**_____**	**(35 points)**
Page 7	**_____**	**(39 points)**
Page 8	**_____**	**(36 points)**
Page 9	**_____**	**(40 points)**
Subtotal	**_____**	**(314 points) = 100%**
Page 10 Extra Credit	**_____**	**(25 points) [8% Extra Credit]**
Total	**_____**	

This exam is to be taken by yourself with closed books, closed notes, no electronic devices.
You are allowed both sides 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 T_STAR {: System.out.println("A"); :} Expr {: System.out.println("B"); :}
     |     Des {: System.out.println("C"); :}
     ;

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

Des2 ::=   Des2 {: System.out.println("K"); :} T_PLUSPLUS {: System.out.println("L"); :}
     |     Des3
     ;

Des3 ::=   T_ID
     ;
```

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

```
++*x++ * *+++*++y * z
```

Output

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

In the above grammar, which has higher precedence: pre-increment operator or address-of operator, or do they have the same precedence? [2pts]

_____

What type of associativity does the `Expr` rule have in the above grammar? [2pts]

_____

If variable `z` is defined to be type `int`, what type must variables `x` and `y` be defined for this expression to be semantically correct? [6pts]

_____ `x;`

_____ `y;`

2. Given the following Reduced-C code fragment: [30pts]

```
function : int & foo( int x, int * & y, int & z ) { /* Body of code not important for this question */ }

function : int main()
{
  int a = 8675309;
  int * b = &a;
  int c = a;

  c = foo( a, b, c );

  return *b;
}
```

Complete the SPARC Assembly language statements that might be emitted by a compliant Reduced-C compiler from this quarter for function main(). Allocate, store, and access all local variables on the Stack. See comments.

```
        .section __________
        .align 4
        .global  __________
_______:
        set     SAVE.main, %g1
        save    %sp, %g1, %sp
        /* Initialize the local variables in this stack frame */

        set     _______________, %o0

        st      %o0, _______________                    ! int a = 8675309;

        ______  ___________________________________, %o0  ! large blank can be one or two operands

        st      %o0, _______________                    ! int * b = &a;

        ld      _______________, %o0

        st      %o0, _______________                    ! int c = a;

        /* Set up the 3 actual arguments to foo() */

        ______  ___________________________________, %o0  ! large blank can be one or two operands

        ______  ___________________________________, %o1  ! large blank can be one or two operands

        ______  ___________________________________, %o2  ! large blank can be one or two operands

        call    foo                                     ! Call function foo()
        nop

        st       _______________, [%fp - 16]            ! Save return value into local temp1

        /* Copy saved return value stored in temp1 into local var c */

        ld       [%fp - 16], %o0

        _______  ___________________________________, %o0  ! large blank can be one or two operands

        _______  _______________, _______________       ! c = foo( ... );

        /* return *b; */

        _______  _______________, _______________       ! return *b;

        _______  _______________, _______________

        ___________

        ___________

        SAVE.main = -(92 + 16) & -8                     ! Save space for 3 local vars + 1 temp
```

3. In object-oriented languages like Java, determining which method code/instructions to bind to (to execute) is done at run time rather than at compile time (this is known as dynamic dispatch or dynamic binding). However, the name-mangled symbol denoting a particular method name is determined at compile time. Given the following Java class definitions, specify the output of each print() method invocation. If any invocation results in an error, write ERROR in the blank. [30pts]

```
class Moe {
    public void print(Moe m) {
        System.out.println("Moe 1");
    }
}

class Larry extends Moe {
    public void print(Moe m) {
        System.out.println("Larry 1");
    }

    public void print(Larry l) {
        System.out.println("Larry 2");
    }
}

class Curly extends Larry {
    public void print(Moe m) {
        System.out.println("Curly 1");
    }

    public void print(Larry l) {
        System.out.println("Curly 2");
    }

    public void print(Curly c) {
        System.out.println("Curly 3");
    }
}

public class Overloading_Final_Exam {
    public static void main (String [] args) {
        Larry  stooge1 = new Curly();
        Moe    stooge2 = new Larry();
        Moe    stooge3 = new Curly();
        Curly  stooge4 = new Curly();
        Larry  stooge5 = new Larry();

        stooge1.print(new Moe());                    _____________________

        ((Curly)stooge1).print(new Larry());         _____________________

        ((Larry)stooge2).print(new Moe());           _____________________

        stooge2.print(new Curly());                  _____________________

        stooge3.print(new Moe());                    _____________________

        stooge3.print(new Larry());                  _____________________

        stooge3.print(new Curly());                  _____________________

        ((Curly)stooge3).print(new Larry());         _____________________

        ((Curly)stooge3).print(new Curly());         _____________________

        stooge4.print(new Moe());                    _____________________

        stooge4.print(new Larry());                  _____________________

        stooge4.print(new Curly());                  _____________________

        stooge5.print(new Moe());                    _____________________

        stooge5.print(new Larry());                  _____________________

        stooge5.print(new Curly());                  _____________________
    }
}
```

4. Given the following code where ??? may represent different parameter passing modes, what values do you expect to be printed if the parameter passing mode is: [24pts]

```
int x = 6;
int y = 9;
int * z = &x;

void foo( int * ??? p1, int * p2 )
{
    *p1 = 88;
    cout << *p1 << endl;
    cout << *p2 << endl;
    p1 = p2;
    cout << *p1 << endl;
    *p1 = 77;
    cout << x << endl;
    cout << y << endl;
    cout << *z << endl;
}

int main()
{
    cout << x << endl;
    cout << y << endl;
    cout << *z << endl;
    foo( z, &y );
    cout << x << endl;
    cout << y << endl;
    cout << *z << endl;
    return 0;
}
```

call-by-value?	**call-by-reference?**
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

What are the values of `a` and `b` after the following two Reduced-C statements? [4pts]

```
int a = 5;
bool b = (a++ > 5) && ((a = 2) != 0);
```

Value of `a` is _____ Value of `b` is _____

Fill in the blanks [2pts each]:

_____ analysis deals with verifying correct <u>meaning</u> of a program

_____ analysis deals with verifying correct <u>structure</u> of a program.

A(n) _____ performs thorough analysis and nontrivial transformations on a program in language L1 into an equivalent program in language L2. This is in contrast to a(n) _____ which directly performs the operations implied by the program.

The _____ is the main data structure used in the compiler to store/retrieve information about names (variables, functions, etc) and other attributes (e.g. types) associated with these names.

There are two types of programming errors can easily occur when the programmer is left to deal with dynamic memory management. Garbage collection essentially eliminates these two errors, which are

_____ and _____.

Variables declared to be _____ will not be optimized by the compiler.

5. Given the following Reduced-C code and the Project 2 Spec,

```
     /* Reduced-C */
function : void main()
{
    int * x = nullptr;

    delete x;
}
```

What one line of SPARC assembly code (after the `save` instruction) should your compiler generate with regards to the variable definition of `x`? (Phase I.1) [6pts]

_____

For the `delete` statement above, fill in the SPARC assembly instructions below to perform the necessary run time check and the actual delete (Phase III.1 & III.2) [14pts]

```
    ld      __________, %o0
    ______  __________, __________
    ______  .Delete_Error12
    nop
    call    ___________
    nop
    st      __________, __________
    ______  .L47
    nop
.Delete_Error12:
    !! Assume .DeleteMsg string already defined in ".data" section
    set     .DeleteMsg, __________
    call    __________
    nop
    set     __________, __________
    call    __________
    nop
.L47:
```

Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named foo that is a 2D array of 3 rows by 5 columns where each element is a pointer to a pointer to an array of 7 elements where each element is a pointer to a function that takes a pointer to a pointer to a float as a single parameter and returns a pointer to an array of 27 elements where each element is a pointer to a struct baz. [8pts]

_____;

6. Given the C array declaration

C

```
short a[3][4];
```

Mark with an **A** the memory location(s) where we would find the array element `a[2][1]`:

`a:` Each box represents a byte in memory. [4pts]



`low memory` `high memory`

Given the definitions below, indicate what type of object each expression is. [15pts]

```
function : int & f1() { /* Function body not important. */ }
function : int * f2() { /* Function body not important. */ }
const int a = 5;
int b = 1;
int[5] c;
int *d = nullptr;
```

A) Modifiable L-val
B) Non-Modifiable L-val
C) R-val

____ a	____ b	____ c	____ d	____ a + b
____ &b	____ (float) b	____ c[b]	____ *d	____ f1()
____ *&*d	____ *(float *)d	____ ++c[0]	____ nullptr	____ f2()

Identify whether each of the following will cause an underlying bit pattern change. [6pts]

```
int a = 5;
float b = -4.20;
int * ptr1;
float * ptr2;
void foo( int x, float & y ) { /* ... */ }
```

A) Yes – Underlying bit pattern change
B) No – No underlying bit pattern change

`b = a;`	_____	`b = * (float *) & a;`	_____
`a = (int) b;`	_____	`ptr1 = (int *) ptr2;`	_____
`foo( a, b );`	_____	`a + b;`	_____

Phase II.3 from Project 2: Assume structdef `FOO` has been correctly defined and its size has been loaded into register %l7. Given the following Reduced-C code, how might a compiler generate SPARC assembly code to perform struct assignment without copying each struct member one-by-one? [10pts]

```
FOO a;     // Assume local variable a is located at %fp - 192
FOO b;     // Assume local variable b is located at %fp – 384

    ...    // Other code that may access/modify a and b
           // sizeof(FOO) is in register %l7 at this point
b = a;     // Write the SPARC assembly code to perform this struct assignment
```

7. Given the following program, specify the order of the output lines when run and sorted by the address printed with the %p format specifier on a Sun SPARC Unix system. For example, which line will print the lowest memory address, then the next higher memory address, etc. up to the highest memory address? [26pts]

```
#include <stdio.h>
#include <stdlib.h>

void foo( int , int ); /* Function Prototype */

int a = 8675309;

int main( int argc, char *argv[] ) {

   int b = 420;
   int c;

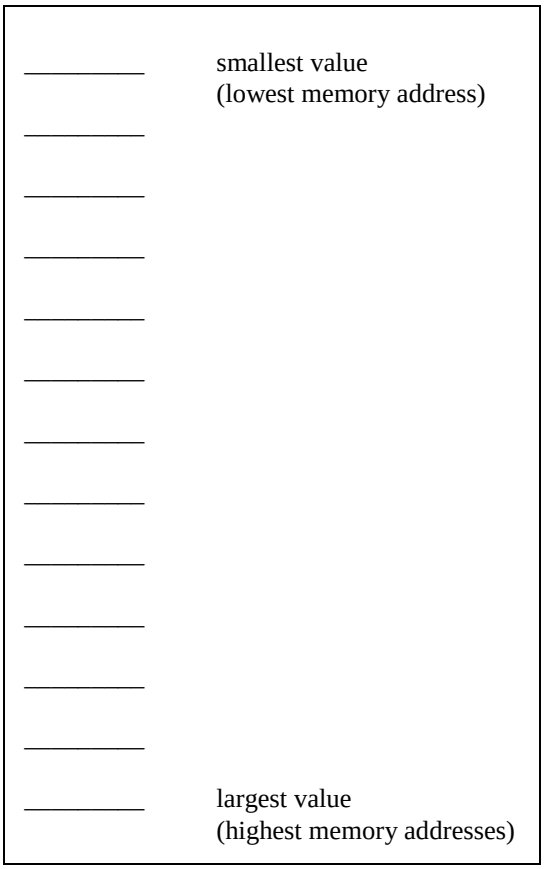
   foo( argc, b );

/*  1 */ (void) printf( "1: argc --> %p\n", &argc );
/*  2 */ (void) printf( "2: foo --> %p\n", foo );
/*  3 */ (void) printf( "3: malloc --> %p\n", malloc(4) );
/*  4 */ (void) printf( "4: b --> %p\n", &b );
/*  5 */ (void) printf( "5: a --> %p\n", &a );
/*  6 */ (void) printf( "6: c --> %p\n", &c );
}

void foo( int d, int e ) {

   int f;
   static struct foo {int a; int b;} g;
   int h[3];

/*  7 */ (void) printf( "7: g.b --> %p\n", &g.b );
/*  8 */ (void) printf( "8: h[0] --> %p\n", &h[0] );
/*  9 */ (void) printf( "9: f --> %p\n", &f );
/* 10 */ (void) printf( "10: d --> %p\n", &d );
/* 11 */ (void) printf( "11: e --> %p\n", &e );
/* 12 */ (void) printf( "12: g.a --> %p\n", &g.a );
/* 13 */ (void) printf( "13: h[1] --> %p\n", &h[1] );
}
```



Which of the following Java programs will generally be faster / require less space if we create several instances of a Foo object and access variable **x** many times? Explain why (time and space). [5pts]

Program 1:

```
public class Foo {
   static int[][] x = new int[100][100];
}
```

Program 2:

```
public class Foo {
  int[][] x = new int[100][100];
}
```

List 2 disadvantages of using open/macro subroutines as opposed to closed subroutines: [4pts]

List 2 disadvantages of using leaf subroutines as opposed to closed subroutines: [4pts]

8. Given the following Reduced-C code fragment:

```
bool a;
int b;

function : void foo( int & x, float y )
{ /* function body */ }
```

Using variables `a`, `b`, and the expression `(b + 3)` as possible arguments to the function `foo()`:
Give an example function call to foo() that triggers an <u>assignability error</u> (and only this error). [3pts]

Give an example function call to foo() that triggers an <u>addressability error</u> (and only this error). [3pts]

Give an example function call to foo() that triggers an <u>equivalence error</u> (and only this error). [3pts]

Optimize the following into a single instruction. Assume r1, r3, and r4 are <u>not needed</u> after the last statement [3pts]

```
r1 = 15
r2 = r2 * 1
r3 = 7 + r1      _____________________
r4 = r3 - r1
r5 = r4 + r2
```

Optimize the following into three instructions. Assume all registers shown <u>are live (needed)</u> after the last statement [6pts]

```
r2 = 128         _____________________
x = r2
r1 = r5 + r2     _____________________
x = r1
r2 = r4 - r3     _____________________
```

Perform step-wise peephole optimization on the following window of pseudo three-address instructions: [6pts]

```
...other instructions...
r3 = r2 * 32          _________________     (instruction eliminated)
r0 = r3               r0 = r3               _________________      (same inst as STEP 2)
r3 = 13 + 5           r3 = 13 + 5           r3 = 13 + 5            _________________
...other instructions...    STEP 1              STEP 2                    STEP 3
```

Optimize the following into four instructions. Assume any loads will be a L1 cache hit with 1 cycle delay before the result is ready. Also assume only %l3 is <u>live (needed)</u> after the last statement: [8pts]

```
ld [%l0], %l1          ____________________
set 0x00FF0000, %l2
set 0x0000FF00, %l3    ____________________
and %l1, %l2, %l2
and %l1, %l3, %l3      ____________________
srl %l2, 16, %l2
or %l2, %l3, %l3       ____________________
```

Optimize the following into two instructions. Assume all registers shown <u>are live (needed)</u> after the last statement. Maintain the last two statements as shown: [4pts]

```
r1 = r2 * r5      _____________________
r3 = r1
r6 = r5 * r2      _____________________
r4 = r6

r1 = ...          r1 = ...
r6 = ...          r6 = ...
```

9. Given the following C type definitions: [24pts]

```
struct bar {
  char  a[13];
  short b[2];
  char  c;
  int   d;
};

struct foo {
  short      e;
  double     f;
  char       g;
  struct bar h;
};

struct fubar {
  int        i;
  char       j[6];
  struct foo k;
  int        l;
};

struct fubar fubaz;
```

Hint: Draw the memory layout of these structs (including padding) on the scratch paper at the end of this exam.

What is the `sizeof( struct bar )`? _____ What is the `sizeof( struct foo )`? _____

What is the `sizeof( struct fubar )`? _____ What is the `offsetof( struct fubar, k.h.a[7] )`? _____

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

What is the `sizeof( struct fubar )` if members of all 3 structs are reordered largest to smallest? _____

Generate the SPARC assembly code for the 3 labeled statements below based on the requirements of this quarter's Reduced-C programming project (<u>do not include the NULL pointer checks</u>). Assume the two local variables have been allocated on the Stack with an appropriate `save` instruction. Draw a line between and label the group of translated instructions for each labeled line. [15pts]

```
function : void foo() {
    int i;
    int * ptr;

    new ptr;      /* 1 */

    *ptr = 5000;  /* 2 */

    i = *ptr;     /* 3 */
}
```

What programming bug exists in the above code? [1pt]

10. Extra Credit

What gets printed when the following C program is executed? [9pts]

```
#include <stdio.h>
int main() {
  char a[] = "CSE131";
  char *p = a;

  printf( "%c\n", *(p+5) = *a + 1 );                  ______

  printf( "%c\n", *a = *p++ + 5 );                    ______

  printf( "%c\n", ++*p++ );                           ______

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

  printf( "%c\n", 4[a] = *(p-2) );                    ______

  printf( "%c\n", *(p-2) = *a );                      ______

  printf( "%c\n", *(p-3) = *(a+4) );                  ______

  printf( "%c\n", *++p = 'N' );                       ______

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

  return 0;
}
```

Looking back at the expression in Question #1 (`++*x++ * *++*++y * z`), write a valid C declaration for variables `x`, `y`, and `z` and initialize them in such a way that the following code will output the number 48. You can use dynamic memory and must ensure all memory locations spanned by the expression are valid. But, you <u>do not</u> need to worry about memory leaks. [16pts]

```
#include <stdio.h>

int main() {
    /* write the declaration and initialization for x, y, and z below */









    printf("%d\n", ++*x++ * *++*++y * z);
    /* should result in the number 48 being output to the screen */
    return 0;
}
```

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

A portion of the C Operator Precedence Table

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

Scratch Paper

Scratch Paper