Login Name _____ **Name** _____

Student ID _____ **Signature** _____

Final
CSE 131
Winter 2012

Page 1	_____	**(29 points)**
Page 2	_____	**(30 points)**
Page 3	_____	**(24 points)**
Page 4	_____	**(38 points)**
Page 5	_____	**(21 points)**
Page 6	_____	**(38 points)**
Page 7	_____	**(27 points)**
Page 8	_____	**(23 points)**
Page 9	_____	**(23 points)**
Page 10	_____	**(22 points)**
Page 11	_____	**(17 points)**
Subtotal	_____	**(292 points) = 100%**
Page 12 Extra Credit	_____	**(17 points) [6% Extra Credit]**
Total	_____	

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

```
Expr ::=   Des AssignOp {: System.out.println("1"); :} Expr {: System.out.println("2"); :}
     |     Des {: System.out.println("3"); :}
     ;

Des  ::=   T_STAR {: System.out.println("4"); :} Des {: System.out.println("5"); :}
     |     T_PLUSPLUS {: System.out.println("6"); :} Des {: System.out.println("7"); :}
     |     T_AMPERSAND {: System.out.println("8"); :} Des {: System.out.println("9"); :}
     |     Des2 {: System.out.println("10"); :}
     ;

Des2 ::=   Des2 {: System.out.println("11"); :} T_PLUSPLUS {: System.out.println("12"); :}
     |     Des3 {: System.out.println("13"); :}
     ;

Des3 ::=   T_ID {: System.out.println("14"); :}
     ;

AssignOp ::=  T_ASSIGN {: System.out.println("15"); :}
     ;
```

What is the output when parsing the follow statement (you should have 26 lines/numbers in your output):

```
**x = *y++ = &z
```

Output

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

In the above grammar, what is the associativity of the operator in the first production rule (the Expr ::= rule)? _____

If variable `z` is defined to be type `int`, what types must variables `x` and `y` be defined to be for this statement to be semantically correct?

_____ x; _____ y;

2. Given the following Reduced-C code fragment:

```
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;
  int c = a;

  b = 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 __________

        .global __________
        .align 4

_______:
        set     ___________________, %g1

        save    ___________________, %g1, ___________________

        /* Initialize the local variables that have explicit initialization in this stack frame */

        set     _____________, %o0

        st      %o0, _____________                  ! int a = 8675309;

        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

        _______  ___________________________, %o2

        call    foo                                 ! Call function foo()

        __________

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

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

        _______  [%fp - 16], ___________________

        _______  _____________, _____________       ! b = foo( ... );

        /* return b; */

        ld       _____________, _____________       ! return b;

        ___________

        ___________

        MAIN_SAVE = -(92 + _______) ________ ________  ! 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.

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

```
class Wind extends Earth
{
    public void print(Earth p)
    {
        System.out.println("Wind 1");
    }

    public void print(Wind p)
    {
        System.out.println("Wind 2");
    }
}
```

```
class Fire extends Wind
{
    public void print(Earth p)
    {
        System.out.println("Fire 1");
    }

    public void print(Wind p)
    {
        System.out.println("Fire 2");
    }

    public void print(Fire p)
    {
        System.out.println("Fire 3");
    }
}
```

Now <u>remove</u> the entire `print(Earth p) {}` method in `class Wind` and <u>remove</u> the entire `print(Wind p) {}` method in `class Fire`. Specify the output of each print() method <u>with these changes</u> below.

```
public class Overloading_Final_Exam {
    public static void main (String [] args) {

        Earth element1 = new Earth();
        Earth element2 = new Wind();
        Earth element3 = new Fire();
        Wind  element4 = new Wind();
        Wind  element5 = new Fire();
        Fire  element6 = new Fire();


        element1.print( element1 );                          _____________________    _____________________

        element2.print( element2 );                          _____________________    _____________________

        element3.print( element3 );                          _____________________    _____________________

        element4.print( element4 );                          _____________________    _____________________

        element5.print( element5 );                          _____________________    _____________________

        element6.print( element6 );                          _____________________    _____________________


        element1.print( (Earth) element6 );                  _____________________    _____________________

        element2.print( (Wind) element6 );                   _____________________    _____________________

        element3.print( (Fire) element6 );                   _____________________    _____________________


        ( (Earth) element1).print( (Earth) element6 );       _____________________    _____________________

        ( (Earth) element2).print( (Wind) element6 );        _____________________    _____________________

        ( (Earth) element3).print( (Fire) element6 );        _____________________    _____________________

    }
}
```

4. Fill in the blanks of the following Reduced-C program with correct types to test if your global scope resolution operator works correctly. If it does, this program should compile without error. If it does not, this program should generate an assignment error at the line y = ::x;

```
_________  x;

function : int main() {

  _________  x;

  int y;

  y = ::x;       // If :: working, this line will not cause an error!
                 // If :: not working, this line will cause an error!
  return 0;
}
```

In Reduced-C (which follows closely the real C standard) all typedefs use _____ name equivalence. Struct operations (like =, ==, !=) use _____ name equivalence.

In RC (and C/C++), we do not support the assignment of an entire array to another array (of the same type) using the assignment operator. However, we do support assignment of an entire struct instance to another struct instance of the same type. Using this fact, fill in the template of the code below, allowing arrays to piggy-back on a struct type to simulate entire-array assignments that are semantically and logically correct.

```
structdef INTARR5 { int [5] a; };
int [5] x, y;
function : void foo()
{
    // x = y would be a semantic error, but ... the following will assign all elements of y into x

     ______ _____________________ ______ x _______ = ______ _____________________ ______ y _______;
}
```

Given the definitions below, indicate whether each expression is either a

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

```
function : int & foo1() { /* Function body not important. */ }
function : int * foo2() { /* Function body not important. */ }
const int x = 5;
int y;
int[5] a;
int *p = &y;
```

____ a[2]	____ &y	____ a	____ x	____ x + y
____ p	____ *p	____ *&*p	____ &*p	____ y
____ 42	____ (float *)p	____ *(float *)p	____ (float *)&y	____ *(float *)&y
____ ::y	____ foo1()	____ foo2()	____ foo1()++	____ y = *foo2()
____ *p++	____ ++*p	____ *++p	____ --*++p	____ ++*p--

What is Rick's favorite cheese? _____

5. What gets printed in the following C++ program (just like Reduced-C without "function : " in front of each function definition)? If a value is unknown/undefined or otherwise cannot be determined by the code given, put a question mark ('?') for that output. Hint: Draw stack frames!

```
int a = 2;
int b = 4;
int c = 6;
int mo;

int & fubar( int x, int & y, int * z )
{
   static int m = x;
   x = x + 3;
   y = y + 3;
   *z = *z + 3;
   mo = ++m;

   return x;
}

void foo1( int & d, int * e, int f )
{
   d = d + 2;
   *e = *e + 2;
   f = f + 2;

   cout << a << endl;    ________

   cout << b << endl;    ________

   cout << c << endl;    ________

   cout << d << endl;    ________

   cout << *e << endl;   ________

   cout << f << endl;    ________

   cout << mo << endl;   ________

   cout << fubar( d, d, &d ) << endl;   ________

   cout << fubar( *e, *e, e ) << endl;  ________

   cout << fubar( f, f, &f ) << endl;   ________

   cout << a << endl;    ________

   cout << b << endl;    ________

   cout << c << endl;    ________

   cout << d << endl;    ________

   cout << *e << endl;   ________

   cout << f << endl;    ________

   cout << mo << endl;   ________
}

int main()
{
   foo1( a, &b, c );

   cout << a << endl;    ________

   cout << b << endl;    ________

   cout << c << endl;    ________

   cout << mo << endl;   ________

   return 0;
}
```

6. Using the load/load/compute/store and internal static variable paradigms recommended in class and discussion sections, complete the SPARC Assembly language statements that might be emitted by a compliant Reduced-C compiler from this quarter for function foo(). Store all formal params on the Stack.

```
function : int foo( int *x, int y, int & z )
{
  static int c = z;

  *x = c - y;

  return z;
}
```

```
        .section        "____________"
        .global         ____________
        .align          4
foo:
        set     foo.SAVE, %g1
        save    %sp, %g1, %sp

        st      %i0, _______________

        st      %i1, _______________

        st      _________, [%fp + 76]

        .section        "____________"
        .align          4
.foo_c:
        .skip           4
.foo_c_flag:
        .skip           4
        .section        "____________"
! Check if internal static var c has
! already been initialized

        set     ____________________, %o0
        ld      [%o0], %o0

        cmp     __________, __________

        ______  .L1             ! skip init
        nop
! Init internal static var c for 1st time

        ld      _______________, %o0

        ______  _______________, %o0

        st      %o0, [%fp - 4] ! tmp1 = z
        ld      [%fp - 4], %o0

        set     ____________________, %o1

! c = z
        ______  _________, __________
! set flag to skip all further inits
        set     ______, %o0

        set     .foo_c_flag, %o1

        st      ________, ____________
```

```
.L1:

! Perform *x = c - y; block

! c - y
        set     _______________, %o0

        ______  [%o0], %o0          ! c

        ld      _______________, %o1 ! y

        ______  %o0, %o1, %o0       ! c - y

! tmp2 <- (c - y)
        st      __________, [%fp - 8]

! previous result from tmp2
        ld      [%fp - 8], %o0

! get param x
        ld      _______________, %o1

! *x = c - y; (store tmp2 into *x)

        _______ %o0, _____________


! return z;
        ld      _____________, %o0

        ld      _____________, %o0

        ______  %o0, __________

        _________

        _________

! save space for 2 temporaries on stack

foo.SAVE = -(92 + _____) _____ ______
```

7. Given the C array declaration

```
      C
int a[2][4];
```

Mark with an **A** the memory location(s) where we would find

```
      a[1][2]
a:
```

```
low memory                                                    high memory
```

Each box represents a byte in memory.

Using the Right-Left rule write the C definition of a variable named foo that is a pointer to an array of 9 elements where each element is a pointer to a function that takes a pointer to a struct RT as the single parameter and returns a pointer to a 3x17 2-D array where each element is a pointer to a pointer to a struct Fubar.

Identify whether each of the following will cause an underlying bit pattern change.

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

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

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

What are the values of `a` and `b` after the following Reduced-C statements?

```
bool a = false;
bool b = true || (a = true);
```

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

Name the part of the compilation sequence which performs each of the following.

_____ takes an executable file on disk and makes it ready to execute in memory.

_____ zero fills the BSS segment in memory.

_____ puts globally defined symbols in the export list of the resulting object file.

_____ translates assembly code into machine code.

_____ combines all object modules into a single executable file.

_____ resolves undefined external symbols with defined global symbols in other modules.

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

8. 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 and Linux system. For example, which line will print the lowest memory address, then the next higher memory address, etc. up to the highest memory address?

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

void foo1( int *, int ); /* Function Prototype */
void foo2( int, int * ); /* Function Prototype */

int a;

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

   int b;
   double c;

   foo2( a, &b );

/*  1 */ (void) printf( "1: argc --> %p\n", &argc );
/*  2 */ (void) printf( "2: c --> %p\n", &c );
/*  3 */ (void) printf( "3: argv --> %p\n", &argv );
/*  4 */ (void) printf( "4: malloc --> %p\n", malloc(50) );
/*  5 */ (void) printf( "5: b --> %p\n", &b );
}

void foo1( int *d, int e ) {

   static struct foo {int a; int b;} f = { 1, 2 };
   int g;

/*  6 */ (void) printf( "6: f.b --> %p\n", &f.b );
/*  7 */ (void) printf( "7: d --> %p\n", &d );
/*  8 */ (void) printf( "8: e --> %p\n", &e );
/*  9 */ (void) printf( "9: f.a --> %p\n", &f.a );
/* 10 */ (void) printf( "10: foo2 --> %p\n", foo2 );
/* 11 */ (void) printf( "11: g --> %p\n", &g );
}

void foo2( int h, int *i ) {

   int j = 411;
   int k[3];

   foo1( i, j );

/* 12 */ (void) printf( "12: k[1] --> %p\n", &k[1] );
/* 13 */ (void) printf( "13: h --> %p\n", &h );
/* 14 */ (void) printf( "14: a --> %p\n", &a );
/* 15 */ (void) printf( "15: i --> %p\n", &i );
/* 16 */ (void) printf( "16: k[0] --> %p\n", &k[0] );
/* 17 */ (void) printf( "17: j --> %p\n", &j );
}
```

_____ smallest value (lowest memory address)

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____ largest value (highest memory addresses)

You are compiling foo1.c and foo2.c together with gcc. If foo1.c has a global variable definition

```
int a = 42;
```

indicate whether each of the following would cause a linkage editor error or not if put in foo2.c?

A) Yes - Linkage Editor Error
B) No - No Linkage Editor Error

____ `int a = 42;`

____ `extern char a;`

____ `static double a;`

____ `extern void a( char * );`

____ `int a( float b) { return (int)b; }`

____ `static int a( float b) { return (int)b; }`

9. Pick one of the following numbers to answer the questions below related to the cdecl calling convention covered in class.

1) Prologue (in callee) 2) Epilogue (in callee) 3) Pre-Call/Call (in caller) 4) Post-Return (in caller)

_____ Allocates space for return value
_____ Copies actual arguments into argument space
_____ Allocates space for actual arguments
_____ Stores return value into return value location
_____ Allocates space for local variables & temps
_____ Saves registers in caller-save scheme
_____ Retrieves return value from return value location
_____ Copies params passed in regs to param stack space
_____ Restores caller-save registers
_____ Saves registers in callee-save scheme
_____ Saves %pc into the return address location
_____ Retrieves saved return address for return
_____ Performs initialization of local variables
_____ Restores callee-save registers
_____ Deallocates argument space in cdecl mode
_____ Deallocates local variable & temps space

Many experienced programmers prefer to use pre-increment/pre-decrement to perform a stand-alone inc/dec of a variable. For example, `++i;` or `for ( i = 0; i < SIZE; ++i )`

Why might a pre-increment/pre-decrement be preferred for these seasoned programmers? Think in terms of code gen from your compiler.

Given the following C type definitions

```
struct foo {
  short  a;
  char   b;
  double c;
  int    d;
};

struct fubar {
  int        e;
  char       f[6];
  struct foo g;
  int        h;
};

struct fubar fubaz;
```

What is the `sizeof( struct fubar )`? _____ What is the `offsetof( struct fubar, g.d )`? _____

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

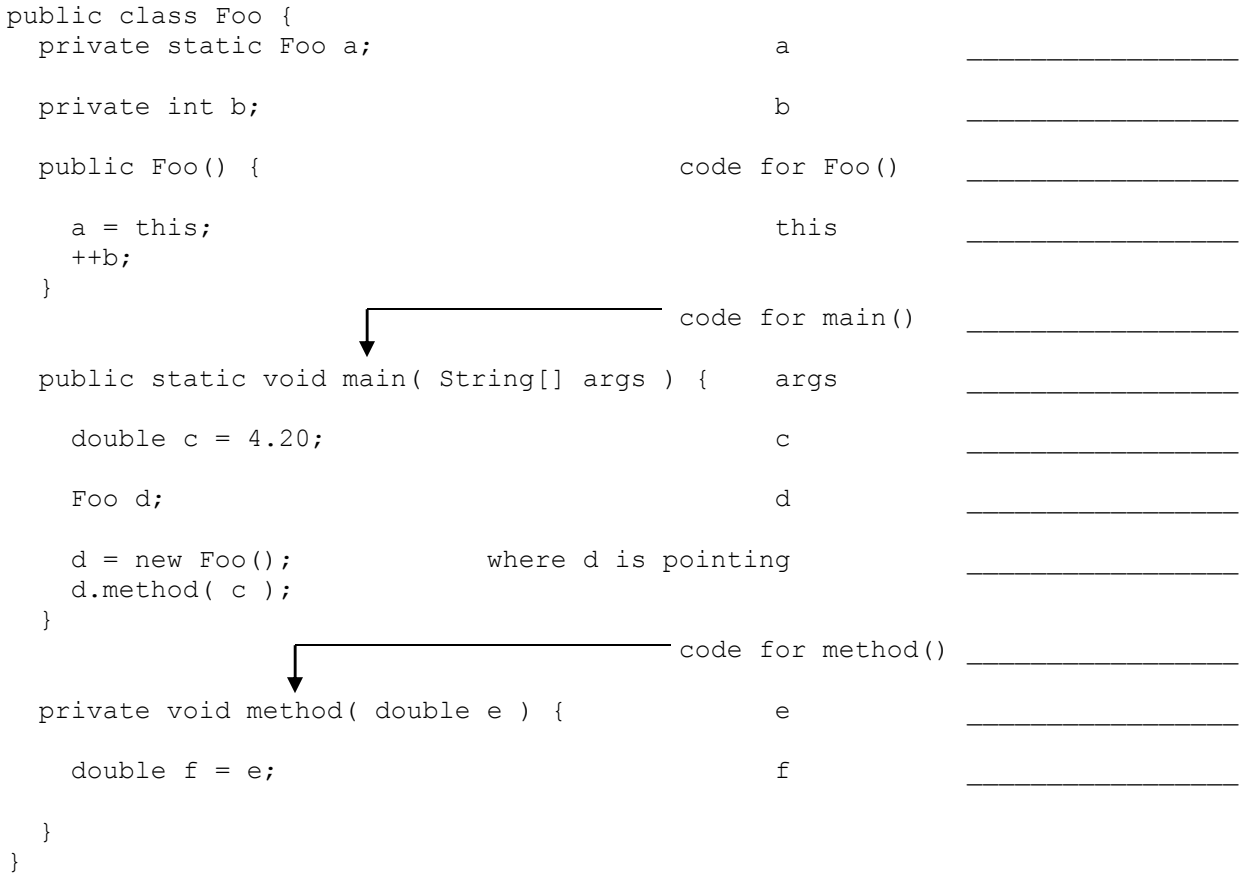
What is the resulting type of the following expression?

```
* (int *) & ( ( (struct fubar *) & fubaz.g.c ) -> g ) _________________
```

Write the equivalent expression that directly accesses this value/memory location without all the fancy casting and `&` operators.

```
fubaz._________________
```

10. Identify <u>where</u> each of the following program parts live in the Java runtime environment as discussed in class.



Write a short, simple Reduced-C program to show how you tested pass-by-reference parameters in your compiler. You will need at least a main() and a function (let's call it foo()) that takes a pass-by-reference parameter. What output would you expect if your compiler implemented pass-by-reference correctly?

11. Use the letters A through D to indicate when you would expect to see each error listed below (assuming a *compiled*, not an interpreted, language).

(A) compile-time (B) link-time (C) load-time (D) run-time

_____ Error message: Left-hand side is not a modifiable l-value.

_____ An "array-index-out-of-bounds" error using a non-constant index expression.

_____ An "array-index-out-of-bounds" error using a constant-valued index expression.

_____ Undeclared identifier "foo".

_____ Segmentation fault.

_____ Running "gcc someModule.o" gives the message "Undefined reference to 'main' ".

_____ Non-addressable argument of type %T to address-of operator.

Use virtual register notation for each of the following.

Change the following instruction into three instructions which are most likely a time improvement over the single instruction when it comes to actual execution time.

```
r2 = r4 * 258
```

_____

_____

_____

What term describes this particular kind of peephole optimization?

Change the following instruction into another single instruction which is most likely a time improvement over the current instruction when it comes to actual execution time.

```
r1 = 16 % 3
```

_____

What term describes this particular kind of peephole optimization?

Change the following instructions into two instructions which are most likely a time improvement over the set of instructions when it comes to actual execution time.

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

r1 = …
r6 = …
```

_____

_____

r1 = …
r6 = …

What terms describe these particular kinds of peephole optimizations? List two that apply.

1)

2)

12. Extra Credit

What gets printed when this C program is executed?

```
#include <stdio.h>

int
main()
{
  char a[] = "Build!";
  char *p = a + 3;

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

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

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

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

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

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

  printf( "%d\n", p - a );             ______

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

  return 0;
}
```

What gets printed if the following function is invoked as `recurse( 2, 10 )` ? (Draw stack frames to help.)

```
int
recurse( int a, int b ) {
  int local = b - a;
  int result;

  printf( "%d\n", local );

  if ( b > 7 )
    result = local + recurse( a, b - 1 );
  else
    result = local;

  printf( "%d\n", result );

  return result;
}
```

Put answers here

Crossword Puzzle (next page) (1 point)

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 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