Login Name _____ **Name** _____

Student ID _____ **Signature** _____

Final
CSE 131
Winter 2011

Page 1	_____	**(21 points)**
Page 2	_____	**(29 points)**
Page 3	_____	**(24 points)**
Page 4	_____	**(31 points)**
Page 5	_____	**(23 points)**
Page 6	_____	**(38 points)**
Page 7	_____	**(8 points)**
Page 8	_____	**(19 points)**
Page 9	_____	**(13 points)**
Page 10	_____	**(34 points)**
Page 11	_____	**(15 points)**
Subtotal	_____	**(255 points) = 100%**
Page 12 Extra Credit	_____	**(18 points) [7% Extra Credit]**
Total	_____	

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

```
Stmt ::=   Des AssignOp Des T_SEMI {: System.out.println("0"); :}
     ;

Des  ::=   T_STAR {: System.out.println("1"); :} Des {: System.out.println("2"); :}
     |     T_PLUSPLUS {: System.out.println("3"); :} Des {: System.out.println("4"); :}
     |     T_AMPERSAND {: System.out.println("5"); :} Des {: System.out.println("6"); :}
     |     Des2 {: System.out.println("7"); :}
     ;

Des2 ::=   Des2 {: System.out.println("8"); :} T_PLUSPLUS {: System.out.println("9"); :}
     |     Des3 {: System.out.println("10"); :}
     ;

Des3 ::=   T_ID {: System.out.println("11"); :}
     ;

AssignOp ::=  T_ASSIGN {: System.out.println("12"); :}
     ;
```

What is the output when parsing the follow statement (you should have 18 lines/numbers in your output):

```
*++x = &*y++;
```

Output

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

Does the above grammar agree with the C/C++ operator associativity? _____

Does the above grammar agree with the C/C++ operator precedence? _____

If variable `y` is defined to be type `float *`, what type must variable `x` be defined to be for this statement to be semantically correct? _____

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;
  int b = -420;
  int c = b;

  a = foo( a, &b, c );

  return c;
}
```

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.

```
        .section __________

        .global __________
        .align 4

_______:
        set     ___________________, %g1

        save    ________, %g1, ________

        /* Initialize the local variables */

        set     _________, %o0

        st      %o0, _______________                ! int b = -420;

        ld      _______________, %o0

        st      %o0, _______________                ! int c = b;

        /* 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 a */

        _______  [%fp - 16], ____________________

        _______  ____________________________________  ! a = foo( ... );

        /* return c; */

        ld       _______________, _______________

        ___________

        ___________

        MAIN_SAVE = -(92 + _______) ________ ________  ! Save space for 3 local vars + 1 temp
```

3. In object-oriented languages like Java, determining which overloaded method code to bind to (to execute) is done at run time rather than at compile time (this is known as dynamic dispatching 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 Larry
{
    public void print(Larry l)
    {
        System.out.println("Larry 1");
    }
}
```

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

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

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

    public void print(Curly c)
    {
        System.out.println("Moe 2");
    }

    public void print(Larry l)
    {
        System.out.println("Moe 3");
    }
}
```

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

        Larry  stooge1 = new Moe();
        Larry  stooge2 = new Larry();
        Larry  stooge3 = new Curly();
        Curly  stooge4 = new Moe();
        Curly  stooge5 = new Curly();
        Moe    stooge6 = new Moe();


        stooge4.print( stooge5 );                          _____________________    _____________________

        stooge5.print( stooge6 );                          _____________________    _____________________

        stooge6.print( stooge4 );                          _____________________    _____________________

        stooge1.print( stooge6 );                          _____________________    _____________________

        stooge2.print( stooge1 );                          _____________________    _____________________

        stooge3.print( stooge4 );                          _____________________    _____________________


        stooge1.print( (Curly) stooge6 );                  _____________________    _____________________

        stooge2.print( (Moe) stooge1 );                    _____________________    _____________________

        stooge3.print( (Moe) stooge4 );                    _____________________    _____________________


        ( (Moe) stooge1).print( (Curly) stooge6 );         _____________________    _____________________

        ( (Larry) stooge2).print( (Moe) stooge1 );         _____________________    _____________________

        ( (Curly) stooge3).print( (Moe) stooge4 );         _____________________    _____________________

    }
}
```

Now remove the entire `print(Larry l) {}` method in `class Curly` and remove the entire `print(Curly c) {}` method in `class Moe`. Specify the output of each print() method with these changes below.

4. In your Project 2, how did you (and your partner if you had a partner) handle code gen for the address-of operator with an Expression that results in a modifiable l-val? For example, `&*ptr` or `&a[i]` or `&mystruct.a`. Note this question is not asking about handling the address-of operator with an identifier. Be specific how your project implemented this!

Using Reduced-C syntax, first define a struct B with members of type int, float, and pointer to struct B named a, b, and ptr, respectively. Then define a variable named `foobaz` which is an array of an array (with dimensions 8x4) of pointers to struct B such that `foobaz[7][1]->ptr = foobaz[1][3];` is a valid expression. This will take more than one line of code.

Change the following into 3 instructions that is an improvement over the current set of generated instructions

```
r1 = r2 + r3                                          ___________________
x = r1    ! write to mem x
r4 = x    ! read from mem x                           ___________________
r2 = r4 + r5
x = r2                                                ___________________
r1 = x
r2 = r1   ! at this point mem. loc. x should have value of last write and r2 have correct value.
r4 = …    ! r4 is dead; previous value in r4 not needed. Optimize instructions above this point.
r1 = …    ! r1 is dead; previous value in r1 not needed.
```

Suppose you have three source files with the variable declarations and definitions indicated below:

Module A

```
int ddd;
extern float rrr;
static bool zzz;
static int* ppp;
```

Module B

```
static int ddd;
extern float rrr;
static bool zzz;
static float ppp;
```

Module C

```
extern int ddd;
extern float rrr;
extern bool zzz;
static int[5] ppp;
```

An executable is desired from linking just these three modules. Assume that the `main()` function is properly defined in one of these modules. For each module, indicate with a **Yes** or **No** whether you would expect a linker error to occur if the specified variable is used in an expression somewhere in that module.

	Module A	Module B	Module C
ddd	_____	_____	_____
rrr	_____	_____	_____
zzz	_____	_____	_____
ppp	_____	_____	_____

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 = 54;
int b = 43;
int c = 32;

void fubar( int * x, int & y, int z )
{
   ++*x;
   ++y;
   ++z;
}

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

   cout << a << endl;   ______

   cout << b << endl;   ______

   cout << c << endl;   ______

   cout << d << endl;   ______

   cout << e << endl;   ______

   cout << *f << endl;  ______

   fubar( &d, d, d );

   fubar( &e, e, e );

   fubar( f, *f, *f );

   cout << a << endl;   ______

   cout << b << endl;   ______

   cout << c << endl;   ______

   cout << d << endl;   ______

   cout << e << endl;   ______

   cout << *f << endl;  ______
}

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

   cout << a << endl;   ______

   cout << b << endl;   ______

   cout << c << endl;   ______

   return 0;
}
```

Using the Right-Left rule write the C definition of a variable named fubaz that is a pointer to a 2-d array of 19 rows by 4 columns where each element is a pointer to a function that takes a pointer to a pointer to a short as a single parameter and returns a pointer to an array of 8 elements where each element is a pointer to a struct fubar.

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.

```
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[4][2];
```

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

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

```
low memory                                                    high memory
```

Each box represents a byte in memory.

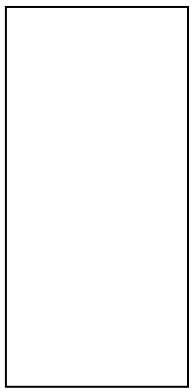
Given the following C code compiled with a compiler that adds no unnecessary padding between local variables (or a similar compliant Reduced-C compiler as described this quarter):

```
int main()
{
  int a[4];
  int b[4];
  int *ptr = a;

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

  return 0;
}
```

Use variable `ptr` (not `a` or `b`) in the `printf` statement above to print the value at memory location `b[1]`.

What is the output of the following C++ program (similar to this quarter's Reduced-C spec)?

```
void foo( int x )
{
   static int y = x - 1;

   cout << y++ << endl;

   if ( x >= 4 && y <= 6 )
       foo( y - 1 );
}

int main()
{
  foo( 5 );
  return 0;
}
```



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 = 42;

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

   int b;
   double c;

   foo1( &a, b );

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

void foo1( int *d, int e ) {

   struct foo {int a; int b;} f;
   int g;

   foo2( g, d );

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

void foo2( int h, int *i ) {

   static int j[3];
   int k = 411;

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

_____ smallest value (lowest memory address)

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

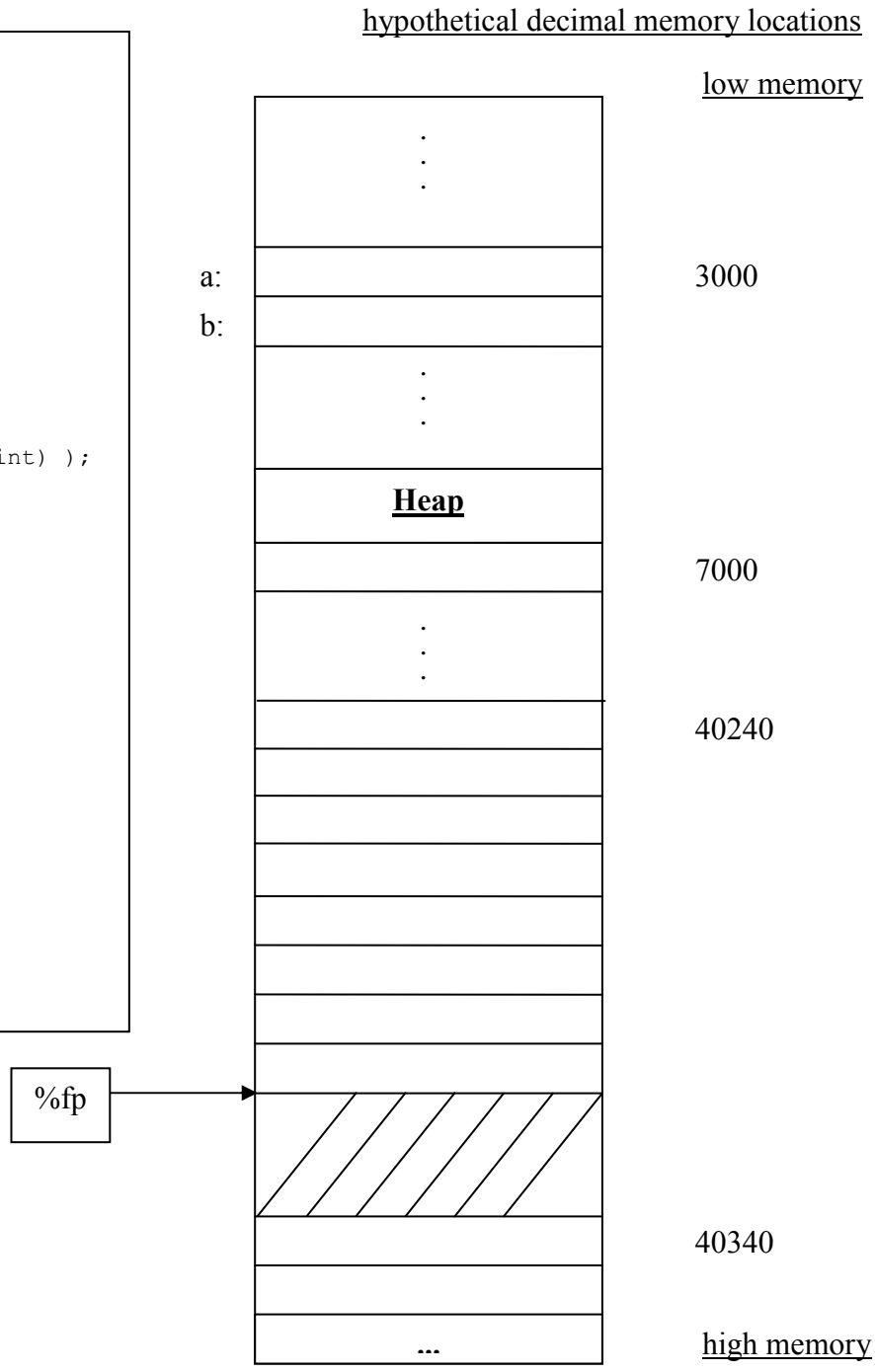
_____ largest value (highest memory addresses)

What is Rick's favorite register? _____

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

9. Given the following C++ program (whose semantics in this case is similar to our Reduced-C) and a real compiler's code gen as discussed in class, fill in the **values** of the global and local variables and parameters in the run time environment for the SPARC architecture when the program reaches the comment `/* HERE */`. Do not add any unnecessary padding.

```
struct fubar {
  int   a;
  int * b;
  float c;
};

int   a = 5;
float b;

void foo( float & f, int i ) {
  int * var1;
  int   var2;
  struct fubar var3[2];

  var2 = -8;
  var1 = (int *) calloc( 1, sizeof(int) );
  f = 1.23;
  var3[0].c = b;
  var3[1].a = i + 6;
  var3[1].b = &var3[1].a;
  i = 73;
  var3[0].a = a;
  var3[0].b = &i;
  var3[1].c = f;
  *var1 = var2 - 1;

  /* HERE */

  free( var1 );
}

int main() {
  foo( b, a );

  return 0;
}
```



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

```
public class Foo {
  private Foo a;                                    a          _____________________

  private static int b;                             b          _____________________

  public Foo() {                                code for Foo()  _____________________

    a = this;                                       this       _____________________
    ++b;
  }
                                                code for main()  _____________________

  public static void main( String[] args ) {      args       _____________________

    Foo c = new Foo();                              c          _____________________

    int d = 3;                                      d          _____________________

    c = new Foo();              where c is pointing            _____________________
    c.method( d );
  }
                                                code for method() _____________________

  private void method( int e ) {                    e          _____________________

    int f;                                          f          _____________________
    f = e;
  }
}
```

Letters A through H below list some types of objects that exist in memory. The numbered items that follow are possible assembly-related actions that may be required in order to define or access these objects. For each numbered item, list ALL possible letters corresponding to those objects for which the action needs to occur.

(A) Global variables (B) Extern declarations (C) External static variables (D) Internal static variables
(E) Local variables (F) Function parameters (G) Global functions (H) Struct member functions

1. _____ List the object name in a ".global" directive

2. _____ Mangle the object name (Do **not** list C or G for this one)

3. _____ Create an assembly label for the object

4. _____ Allocate space for the object using one of the .word, .space, or .skip directives

5. _____ Allocate space for the object using the "save" instruction

6. _____ Use a "set" instruction with the label for the object to get it's address

7. _____ Use a positive offset from %fp to get the object's address

8. _____ Use a negative offset from %fp to get the object's address

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.

_____ A calculator program claims that the sum of 4 and 20 is 420.

_____ A "…/libc.so.2.5 not found" message when trying to run an executable file.

_____ An "Out of memory" message.

Given the following C type definitions

```
struct foo {            struct fubar {
  short  a;               char       e[5];
  char   b;               int        f;
  double c;               struct foo g;
  short  d;               int        h;
};                      };

struct fubar fubaz;
```

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

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?

```
* (short *) & ( ( (struct foo *) & fubaz.e ) -> d ) _________________
```

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

```
fubaz._________________
```

12. Extra Credit

What gets printed when this program is executed?

```
#include <stdio.h>

int
main()
{
  char a[] = "91021";
  char *ptr = a;

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

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

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

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

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

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

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

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

  return 0;
}
```

Tell me something you learned in this class that is extremely valuable to you and that you think you will be able to use for the rest of your computer science career. (1 point if serious; you can add non-serious comments also)

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