Student ID _____ **Name _____**

Login Name _____ **Signature _____**

Final
CSE 131
Winter 2009

Page 1	**_____ (34 points)**
Page 2	**_____ (30 points)**
Page 3	**_____ (39 points)**
Page 4	**_____ (39 points)**
Page 5	**_____ (36 points)**
Page 6	**_____ (15 points)**
Page 7	**_____ (20 points)**
Page 8	**_____ (26 points)**
Subtotal	**_____(239 points)**
Page 9 Extra Credit	**_____ (17 points)**
Total	**_____**

1. 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 = 42;
  int b = a;
  int c;

  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 and 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 a = 42;

        ld      [_____________], %o0

        st      %o0, [_____________]                ! int b = a;

        st      _________, [_____________]          ! int c;

        /* 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 c */

        _______  [%fp - 16], ____________________

        _______  ___________________________________  ! c = foo( ... );

        /* return b; */

        ld       [_____________], _______________

        ___________

        ___________

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

2. 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 Moe {
    public void print(Moe p) {
        System.out.println("Moe 1");
    }
}

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

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

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

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

    public void print(Curly b) {
        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();

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

        stooge1.print(new Moe());                           _____________________


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

        stooge2.print(new Curly());                         _____________________


        stooge3.print(new Curly());                         _____________________

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

        stooge3.print(new Larry());                         _____________________

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

        stooge3.print(new Moe());                           _____________________


        stooge4.print(new Curly());                         _____________________

        stooge4.print(new Larry());                         _____________________

        stooge4.print(new Moe());                           _____________________


        stooge5.print(new Curly());                         _____________________

        stooge5.print(new Moe());                           _____________________

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

3. In your Project 2, explain how did you (and your partner if you had a partner) handle code gen of cout with a float variable (as in a statement like: `cout << floatVar`)? Be specific how your project implemented this!

Give the order of the typical C compilation stages and on to actual execution as discussed in class

0 – Loader
1 – Program Execution
2 – as (Assember)
3 – Object file (prog.o)
4 – prog.exe/a.out (Executable image)
5 – Segmentation Fault (Core Dump) / General Protection Fault
6 – ccomp (C compiler)
7 – ld (Linkage Editor)
8 – Source file (prog.c)
9 – Assembly file (prog.s)
10 – cpp (C preprocessor)

gcc ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____

Using Reduced-C syntax, define an array of an array of floats with dimensions 7x4 named `foo` such that `foo[6][3] = 42.24;` is a valid expression. This will take two lines of code.

For each of the following make no assumptions of what may be above or below each window of instructions.

Change the following into two instructions that is an improvement over a single multiply instruction

```
r1 = r4 * 33              _______________

                          _______________
```

Optimize the following into three instructions. `x` represents a memory location.

```
x = r2                    _______________
r3 = r2 * r5
x = r2                    _______________
x = r3
r5 = x                    _______________
```

Optimize the following into two instructions. Assume r1, r3, r4, and r6 are not needed after last statement.

```
r1 = 9
r3 = r1 - 4
r4 = r1 - r3
r6 = r4                   _______________
x = r6                                    /* x represents a memory location */
r2 = r2 + 0               _______________
r6 = x
r5 = r2 + r6
x = r5
```

4. From the Reduced-C Spec (which follows closely the real C language standard), complete the following:

A) Addressable C) Modifiable
B) Not Addressable D) Not Modifiable

A Non-Modifiable L-value is _____ and _____.

An R-value is _____ and _____.

A Modifiable L-value is _____ and _____.

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

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

```
int x;
const int y = 5;
int[5] a;
int *p = &x;
```

____ a[2]	____ &x	____ a	____ y	____ x + y
____ p	____ *p	____ *&*p	____ &*p	____ x
____ 42	____ (float *)p	____ *(float *)p	____ (float *)&x	____ *(float *)&x

Given the following Reduced-C definitions:

```
function : float foo( float & a ) { int b; return b; }

float x;  /* global variables */
int y;
```

For each of the following statements, indicate the type of error (if any) that should be reported according to the Project I spec for this quarter. Use the letters associated with the available errors in the box below.

```
 x = foo( 4.2 );          ____
 x = foo( y );            ____
 x = foo( x );            ____
 x = foo( foo( x ) );     ____
 y = foo( x );            ____
 x = foo( x + y );        ____
&x = foo( x );            ____
 x = foo( &x );           ____
```

A) No Error
B) Arg passed to reference param is not a modifiable L-val
C) Argument not assignable to value param
D) Argument not equivalent to reference param
E) Left-hand operand is not assignable (not a mod L-val)
F) Value of right-hand-side type not assignable to left-hand-side type

Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named bar that is a 2-d array of 9 rows by 7 columns where each element is a pointer to an array of 3 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 5 elements where each element is a pointer to a struct foo. (10 pts)

5. 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. (XXpts)

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

  *x = y + c;

  return &z;
}
```

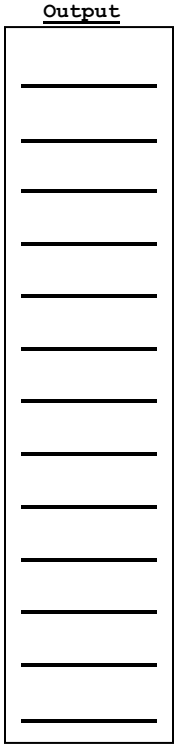
```
        .section        "____________"
        .align          4
.foo_c:
        .skip           4
.foo_c_flag:
        .skip           4

        .section        "____________"

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

        st      %i0, [_______________]

        st      %i1, [_______________]

        st      _________, [%fp + 76]

! 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 = y + c; block

! y + c
        ld      [_______________], %o0  ! y

        set     _______________, %o1

        ______  [%o1], %o1              ! c

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

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

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

! get param x
        ld      [_______________], %o1

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

        _______ %o0, [_____________]


! return &z;
        ld      [_____________], %o0

        ______  %o0, __________

        __________

        __________

! save space for 2 temporaries on stack

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

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

```
Stmt ::=       Designator T_ASSIGN {: System.out.println("A"); :} Expr {: System.out.println("B"); :} T_SEMI
     ;

Expr ::=       Expr {: System.out.println("C"); :} MulOp {: System.out.println("D"); :} Designator
     |         Designator {: System.out.println("E"); :}
     ;

Designator ::= T_ID {: System.out.println("F"); :}
     ;

MulOp ::=      T_STAR {: System.out.println("*"); :}
     ;
```

Output



What is the output on the screen when the follow statement is given as input:

```
a = b * c * d;
```

Rewrite the `Expr` rule so expressions like `a * b * c` will evaluate as `a * (b * c)`. Do not include the actions.

Which of the following would be correct if we wanted to add the divide sign (/) as an operator with lower precedence than the current multiplication sign (*)? _____

A

```
Expr :: =   Expr Op Designator
      |     Designator
      ;

Op ::=      T_SLASH
      |     T_STAR
      ;
```

C

```
Expr :: =   Expr T_STAR Expr1
      |     Expr1
      ;

Expr1 ::=   Expr1 T_SLASH Designator
      |     Designator
      ;
```

B

```
Expr :: =   Expr Op Designator
      |     Designator
      ;

Op ::=      T_STAR
      |     T_SLASH
      ;
```

D

```
Expr :: =   Expr T_SLASH Expr1
      |     Expr1
      ;

Expr1 ::=   Expr1 T_STAR Designator
      |     Designator
      ;
```

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

   int a = 42;
   int b;

   foo1( &argc, a );

/*  1 */ (void) printf( "1: foo1 --> %p\n", foo1 );
/*  2 */ (void) printf( "2: b --> %p\n", &b );
/*  3 */ (void) printf( "3: argc --> %p\n", &argc );
/*  4 */ (void) printf( "4: a --> %p\n", &a );
/*  5 */ (void) printf( "5: argv --> %p\n", &argv );
}

void foo1( int *c, int d ) {

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

   foo2( f, &e );

/*  6 */ (void) printf( "6: d --> %p\n", &d );
/*  7 */ (void) printf( "7: malloc --> %p\n", malloc(50) );
/*  8 */ (void) printf( "8: e --> %p\n", &e );
/*  9 */ (void) printf( "9: c --> %p\n", &c );
/* 10 */ (void) printf( "10: g.a --> %p\n", &g.a );
/* 11 */ (void) printf( "11: f --> %p\n", &f );
/* 12 */ (void) printf( "12: g.b --> %p\n", &g.b );
}

void foo2( int h, int *i ) {

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

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

}
```

_____ smallest value (lowest memory address)

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____ largest value (highest memory addresses)

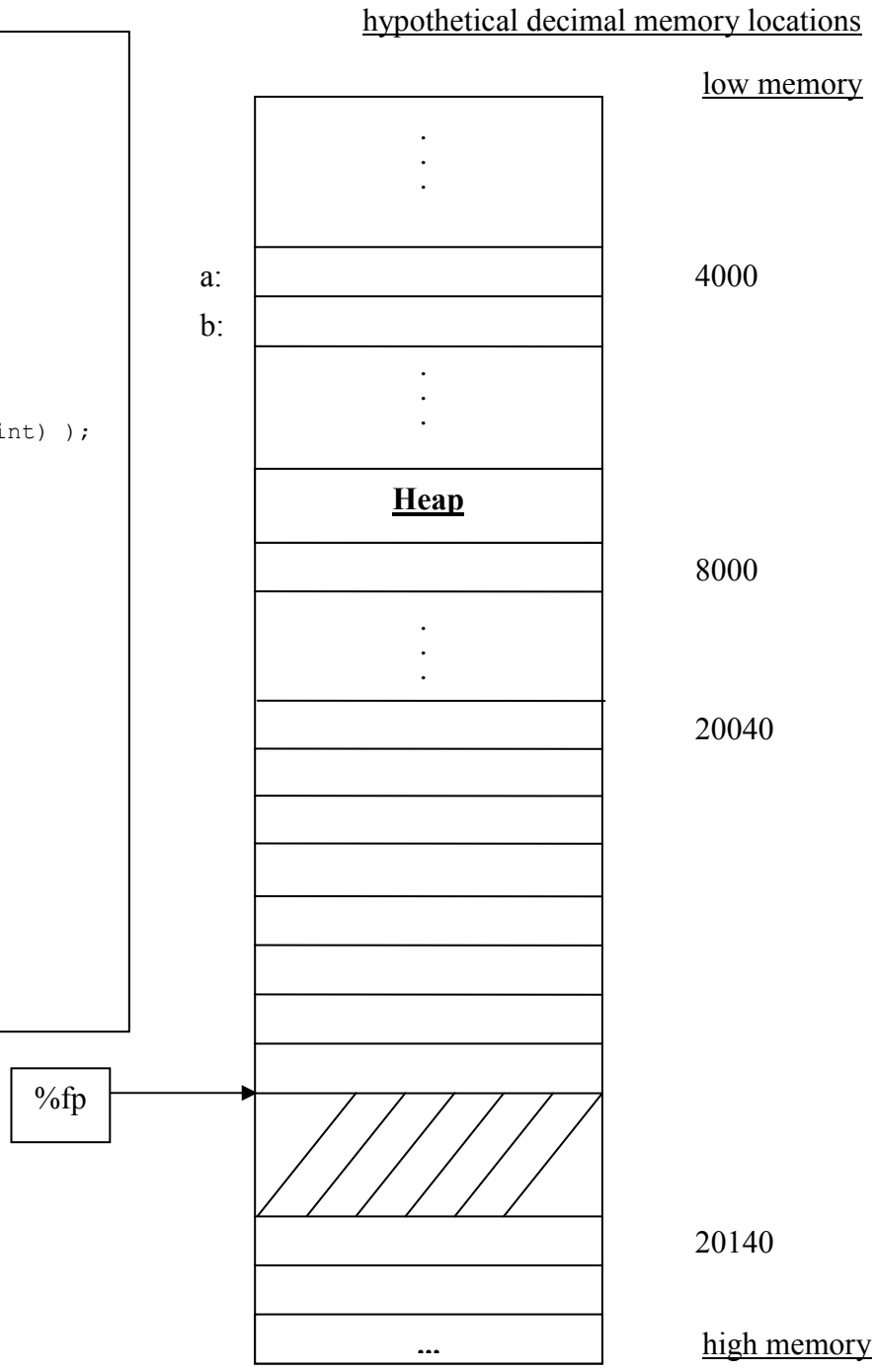
How can you reduce the space required for a large number of local variables of different types on a typical RISC architecture which requires certain alignment restrictions with no optimization compiler flags turned on?

What is Rick's shoe size? _____

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

8. 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 {
  float   x;
  int     y;
  float * z;
};

float a;
int   b;

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

  var3 = (int *) calloc( 1, sizeof(int) );
  f = 98.6;
  var1[1].y = *var3;
  var1[0].z = &a;
  var1[1].x = f;
  var1[0].y = i + 7;
  var1[0].x = -44.25;
  var1[1].z = &var1[1].x;
  var2 = 123;
  i = -99;
  *var3 = var2 + 10;

  /* HERE */

  free( var3 );
}

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

  return 0;
}
```



9. Extra Credit (17 points total extra credit)

What gets printed when this program is executed?

```
#include <stdio.h>

int
main()
{
  char a[] = "Hang 10!";
  char *p = a;

  printf( "%c", *p = toupper( ++p[1] ) );            ______

  printf( "%c", *++p = a[strlen(a) - 6] - 2 );       ______

  printf( "%c", *p++ + 1 );                          ______

  p = p + 3;

  printf( "%c", *p = *p + 3 );                       ______

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

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

  printf( "%s", a );                                 ____________________

  return 0;
}
```

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

_____ expands # directives & strips comments from its input high-level language (HLL)

_____ translates assembly code into machine code

_____ performs syntax analysis on its input high-level language (HLL)

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

_____ translates HLL code (for example, C) into assembly code

_____ combines all object modules into a single executable file

_____ performs semantic analysis on its input high-level language (HLL)

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

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 -- postfix decrement [] array element	L to R
* indirection ++ prefix increment -- prefix decrement & address-of	R to L
* multiplication / division % modulus	L to R
+ addition - subtraction	L to R
. . .	
= assignment	R to L

Scratch Paper

Scratch Paper