Student ID ___________________ **Name** ________________________

Login Name ____________________ **Signature** _______________________

Final
CSE 131
Spring 2008

Page 1	___________	**(25 points)**
Page 2	___________	**(30 points)**
Page 3	___________	**(33 points)**
Page 4	___________	**(21 points)**
Page 5	___________	**(20 points)**
Page 6	___________	**(22 points)**
Page 7	___________	**(15 points)**
Page 8	___________	**(30 points)**
Subtotal	___________	**(196 points)**
Page 9 **Extra Credit**	___________	**(13 points)**
Total	___________	

1. Given the following Reduced-C code fragment:

Reduced-C

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

function : int main()
{
  int a = 10;
  int b;

  b = foo( a, b );

  return 0;
}
```

Complete the SPARC Assembly language statements that might be emitted by a compliant Reduced-C compiler from this quarter for function main().

```
        .section _________
        .global _________
        .align 4
_______:
        set     _________________, %g1
        save    ________, %g1, ________

        /* int a;  -- stored at %fp - 4  */
        /* int b;  -- stored at %fp - 8  */

        /* Initialize the local variables */
        set     _________, %o0
        st      %o0, [____________]
        st      _________, [____________]

        /* Set up the 2 arguments to foo() */
        _______ _________, __________, %o0
        _______ [____________], %o1

        /* Call function foo() */
        call    foo                     ! Call function foo()
        __________

        /* Save return value into local variable b */
        _______ %o0, [____________]

        /* Return 0 */
        mov     _________, _________
        ___________
        ___________

        MAIN_SAVE = -(92 + _______) ________ ________  ! Save space for 2 local vars
```

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();

        stooge1.print(new Moe());                  ______________________

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


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

        stooge2.print(new Curly());                ______________________


        stooge3.print(new Curly());                ______________________

        stooge3.print(new Moe());                  ______________________

        stooge3.print(new Larry());                ______________________

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

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


        stooge4.print(new Curly());                ______________________

        stooge4.print(new Moe());                  ______________________

        stooge4.print(new Larry());                ______________________


        stooge5.print(new Curly());                ______________________

        stooge5.print(new Larry());                ______________________

        stooge5.print(new Moe());                  ______________________
    }
}
```

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

Give the order of the phases of compilation in a typical C compiler as discussed in class

A – Parser (Semantic Analysis)
B – Target language file (for ex., prog.s)
C – Source language file (for example, prog.c)
D – Code generation (for ex., Assembly)

E – Scanner (Lexical Analysis)
F – Parser (Syntax Analysis)
G – Intermediate Representation(s)

______ –> ______ –> ______ –> ______ –> ______ –> ______ –> ______

Using Reduced-C syntax, define an array of 7 pointers to ints named `foo` such that `*foo[6] = 42` 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

```
r3 = r2 * 511                 ______________

                              ______________
```

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

```
x = r1                        ______________
r2 = r1 + r3
x = r1                        ______________
r3 = x
x = r2                        ______________
```

Optimize the following into a single instruction. Assume r1, r3, and r4 are not needed after last statement.

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

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

```
        /* Reduced-C */

function : void main()
{
  int * x;

  delete x;
}
```

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

What run time check and normal code should your compiler generate for the `delete` statement? Fill in the SPARC assembly instructions to perform this run time check and the actual delete. (Phase III.1 & 2)

```
      ld    [_________], %o0

      _____ %o0, %g0

      _____ .Delete_Error22

      nop

      call  _______

      nop

      st    _______, [_________]

      _____ .L37

      nop

.Delete_Error22:

      /* Assume correct code to output delete error message here, then ... */

      set   ______, %o0

      call  ______

      nop

.L37:
```

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

5. Write a short test program in Reduced-C to verify a change to a call-by-reference parameter in a function immediately updates the object the parameter references. (10 pts)

What output do you expect from this program if the call-by-reference is implemented correctly?

Phase II.3 from Project 2: Assume `struct foo` has been correctly defined and its size has been loaded into register %l3. Given the following C code, how might a compiler generate SPARC code to perform struct assignment without copying each struct member one-by-one?

```
struct foo a;     // Assume local variable a is located at %fp - 96
struct foo b;     // Assume local variable b is located at %fp - 192

    ...           // Other code that may access/modify a and b
                  // sizeof( struct foo ) is in register %l3 at this point
  b = a;          // Write the SPARC Assembly code to perform this struct assignment
```

6. 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 assignability error (and only this error).

Give an example function call to foo() that triggers an addressability error (and only this error).

Give an example function call to foo() that triggers an equivalence error (and only this error).

Given the following C code fragment of local variable definitions:

```
int x = 420;
float y = 4.20;
int * ptr1 = &x;
int * ptr2 = ptr1;
```

for each statement below indicate whether it will cause a compile error or not on a current compliant compiler? Treat each statement individually as if it was the only statement following the definitions above. Remember: the increment operator is performing arithmetic (addition) and assignment. The result of this operation is the incremented value.

A) No Error
B) Compile Error

```
ptr2 = &++x;                  _____

(float *)ptr2 = &y;           _____

ptr2 = ++&x;                  _____

&*ptr2 = ptr1;                _____

ptr1 = *&ptr2;                _____

*&ptr2 = ptr1;                _____

++*(float *)&x;               _____

ptr1 = &*ptr2;                _____
```

7. Show the memory layout of the following C struct/record definition taking into consideration the **SPARC** data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct/record 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]s, p[1]s, p[2]s, etc. If the member/field is a struct, use the member name followed by it's member names (e.g. p.a, p.b). Place an X in any bytes of padding. Structs and unions are padded so the total size is evenly divisible by the most strict alignment requirement of its members.

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

struct fubar {
  float d;
  int   e;
  char  f[5];
  struct foo g;
  short h;
};

struct fubar fubaz;
```

low memory

fubaz:

high memory

What is the `offsetof( struct fubar, g.b )`? ________

What is the `sizeof( struct fubar )`? ________

What is the resulting type of the following expression (pure beauty – my kind of Picasso)?

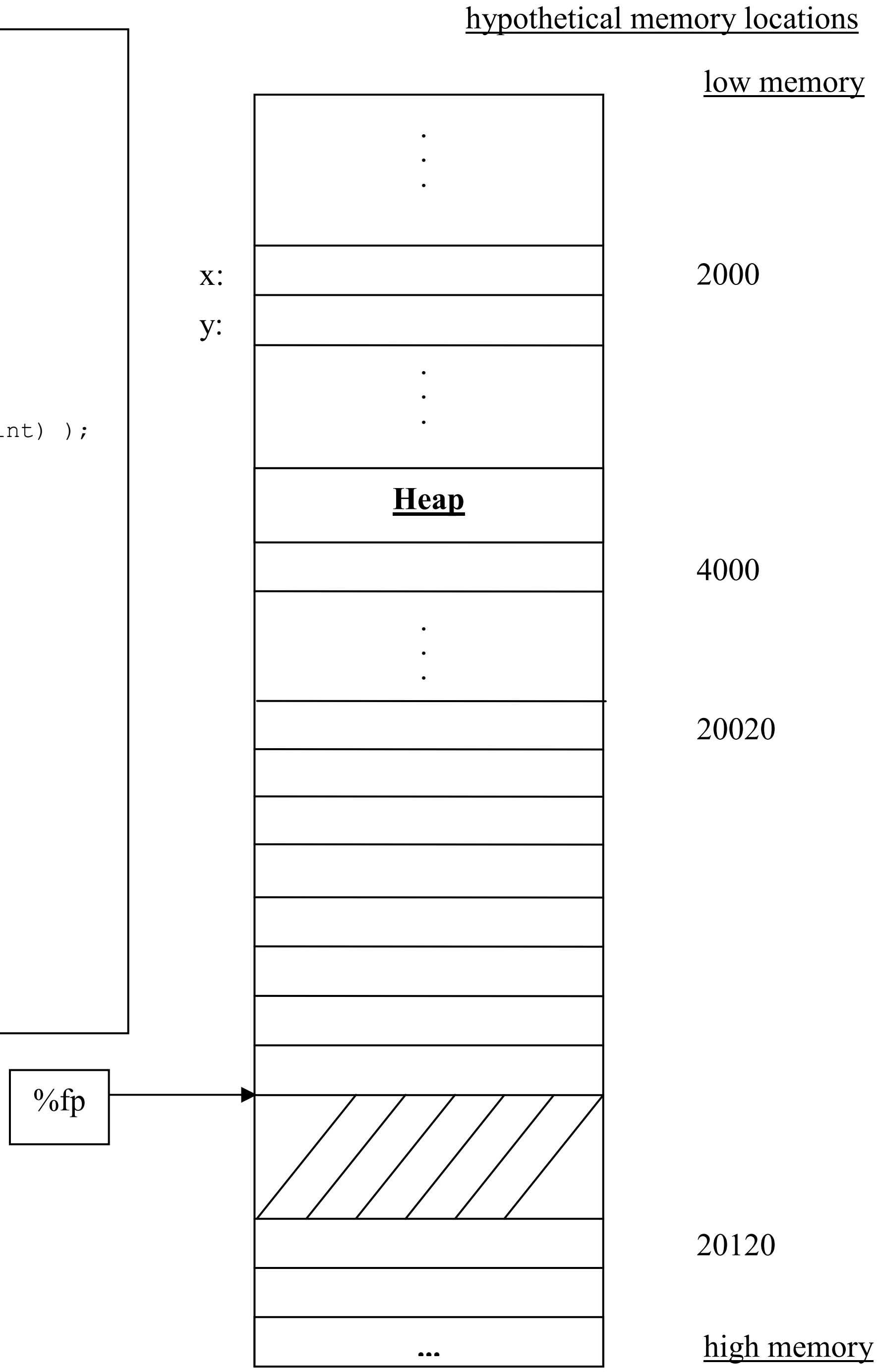
`* (char *) & ( ( (struct foo *) & fubaz ) -> b )` ________________________________

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

`fubaz.`________________

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   a;
  int     b;
  float * c;
};

float x;
int   y;

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

  var1 = (int *) calloc( 1, sizeof(int) );
  f = 98.6;
  var2[1].b = *var1;
  var2[0].c = &x;
  var2[0].a = f;
  var2[0].b = i + 5;
  var2[1].a = -40.5;
  var2[1].c = &var2[0].a;
  var3 = 123;
  i = -99;
  *var1 = var3 - 3;

  /* HERE */

  free( var1 );
}

int main() {
  foo( y, x );

  return 0;
}
```



What is Rick's gangsta name? _______________________

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

9. Extra Credit (13 points total extra credit)

What gets printed when this program is executed?

```
#include <stdio.h>

int
main()
{
  char a[] = "CSE030 Rolls!";
  char *p = a + 2;

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

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

  printf( "%c", ++p[2] );                 ______
  p = p + 4;
  printf( "%c", *++p = a[11] + 2 );       ______
  p++;
  printf( "%c", a[10] = *++p - 7 );       ______

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

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

  return 0;
}
```

With regard to the following C definition:

```
  double x;
```

What type is `(struct bar *) &x` ? ____________________________________

Is the above expression a modifiable l-val? ___________

What type is `*(long *) &x` ? __________________________________

Is the above expression a modifiable l-val? ___________

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

Scratch Paper

Scratch Paper