Student ID ________________ **Name** ______________________

Login Name _________________ **Signature** _____________________

Final
CSE 131
Winter 2008

Page 1	___________	**(24 points)**
Page 2	___________	**(28 points)**
Page 3	___________	**(37 points)**
Page 4	___________	**(24 points)**
Page 5	___________	**(9 points)**
Page 6	___________	**(12 points)**
Page 7	___________	**(12 points)**
Page 8	___________	**(40 points)**
Subtotal	___________	**(186 points)**
Page 9 Extra Credit	___________	**(12 points)**
Total	___________	

1. 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"); :} AddOp {: System.out.println("D"); :} Designator
     |         Designator {: System.out.println("E"); :}
     ;

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

AddOp ::=      T_PLUS {: System.out.println("+"); :}
     ;
```

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

```
a = b + c;
```

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 minus sign (-) as an operator with higher precedence than the current plus sign (+)? _____

A

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

Op ::=      T_MINUS
     |      T_PLUS
     ;
```

C

```
Expr :: =   Expr T_PLUS Expr1
     |      Expr1
     ;

Expr1 ::=   Expr1 T_MINUS Designator
     |      Designator
     ;
```

B

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

Op ::=      T_PLUS
     |      T_MINUS
     ;
```

D

```
Expr :: =   Expr T_MINUS Expr1
     |      Expr1
     ;

Expr1 ::=   Expr1 T_PLUS Designator
     |      Designator
     ;
```

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

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

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

class Brian extends Lois {
    public void print(Peter p) {
        System.out.println("Brian 1");
    }

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

    public void print(Brian b) {
        System.out.println("Brian 3");
    }
}

public class Overloading_Final_Exam {
    public static void main (String [] args) {
        Lois  famGuy1 = new Lois();
        Peter famGuy2 = new Lois();
        Peter famGuy3 = new Brian();
        Brian famGuy4 = new Brian();
        Peter famGuy5 = new Peter();

        famGuy1.print(new Lois());                        ______________________

        ((Peter)famGuy1).print(new Lois());               ______________________


        famGuy2.print(new Lois());                        ______________________

        ((Lois)famGuy2).print(new Lois());                ______________________


        ((Brian)famGuy3).print(new Brian());              ______________________

        famGuy3.print(new Brian());                       ______________________

        ((Brian)famGuy3).print(new Lois());               ______________________


        famGuy4.print(new Brian());                       ______________________

        ((Peter)famGuy4).print(new Brian());              ______________________

        ((Peter)famGuy4).print(new Lois());               ______________________

        ((Peter)famGuy4).print(new Peter());              ______________________


        famGuy5.print(new Brian());                       ______________________

        famGuy5.print(new Lois());                        ______________________

        famGuy5.print(new Peter());                       ______________________
    }
}
```

3. In your Project 2, how did you (and your partner if you had a partner) handle allocating and accessing global variables? Be specific how your project implemented this!

Which part of the entire compilation sequence clear through to program execution is responsible for:

a) resolving undefined external references with defined global references in other modules _______________

b) ensuring the bss segment is set up and zero-filled _______________

c) translating C source code into assembly target code _______________

d) getting the executable image from disk into memory _______________

e) translating assembly source code into object target code _______________

f) creating an executable from multiple object files _______________

e) expanding #defines and #includes _______________

Using Reduced-C syntax, define a pointer to an array of 5 ints named `foo` such that `(*foo)[4] = 42` is a valid expression. This will most likely take two lines of code.

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

```
r3 = r2 * 17            _____________

                        _____________
```

Optimize the following. `x` represents a memory location.

```
x = r1                  _____________

r2 = x                  _____________
```

Optimize the following

```
r1 = r2 + r4            _____________

r3 = r1 + r5            _____________

r6 = r2 + r4            _____________

r1 = 15 + r6
r6 = ...                r6 = ...
```

4. Given the following Reduced-C code, write the equivalent code generated in SPARC Assembly.

```
        /* Reduced-C */

function : bool foo(int & a)
{
  bool b;
  int c = 5;

  a = c;

  return b;
}
```

```
                    /* Equivalent SPARC Assembly code */
```

Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named bar that is an array of 8 elements where each element is a pointer to a function that takes a pointer to a struct fubar as a single parameter and returns a pointer to an array of 3 elements where each element is a pointer to a pointer to a struct Baz. (9 points)

5. The following program to test linked lists in Reduced-C has three bugs that are manifested at run-time. Identify all three of them and what needs to be changed to fix each bug. Each change should require only one minor modification to the code (you should ***not*** need to do any restructuring of multiple lines of code).

```
structdef LIST {
  int data;
  LIST* next;
};

function : void createList(LIST* list) {
   new list;
   list->data = 77;
   new list->next;
   list->next->data = 99;
}

function : void printList(LIST* &list) {
   while(list != NULL) {
       cout << list->data << endl;
       list = list->next;
   }
}

function : void deleteList(LIST* list) {
   if(list->next == NULL) {
       deleteList(list->next);
   }
   delete list;
}

function : void main() {
   LIST* head;

   createList(head);
   printList(head);
   deleteList(head);
}
```

```
/****************************
Correct output should be:
77
99
(with no memory leaks)
****************************/
```

1)

2)

3)

6. Consider the following error messages and programs as defined in the starterCode and Project I:

```
E1) public static final String error5n_Call  =
       "Number of arguments (%D) differs from number of parameters (%D).";

E2) public static final String error5c_Call  =
       "Non-addressable argument passed to reference parameter %S (type %T).";

E3) public static final String error5v_Call  =
       "Argument of type %T not equivalent to reference parameter %S, of type %T.";

E4) public static final String error5a_Call  =
       "Argument of type %T not assignable to value parameter %S, of type %T.";
```

```
function : bool foo( float a )
{
    return true;
}

function : int main() {
    float a;
    int b;
    bool c;

    foo( /* Pass a var here */ );
    return 0;
}
```

List all variables which can be passed as an argument to foo() without causing a compile error? ________________

Which error message (E1, E2, E3, E4) should be displayed if you tried to pass any of the variables that might cause a compile error? If no vars would cause an error, state "No Error". __________

```
function : bool foo( float & a )
{
    return true;
}

function : int main() {
    float a;
    int b;
    bool c;

    foo( /* Pass a var here */ );
    return 0;
}
```

List all variables which can be passed as an argument to foo() without causing a compile error? ________________

Which error message (E1, E2, E3, E4) should be displayed if you tried to pass any of the variables that might cause a compile error? If no vars would cause an error, state "No Error". __________

```
function : bool foo( float & a )
{
    return true;
}

function : int main() {
    int c;

    foo( c + 2.5 );
    return 0;
}
```

Which error message (E1, E2, E3, E4) should be displayed? If not an error, state "No Error". __________

7. Consider the following two C program files:

```
/* file1.c */
#include <stdio.h>
extern float a;
extern int foo( float z );
static int c = 420;

int main( int argc, char *argv[] ) {
   int i = c;

   for ( i = 0; i < c; ++i )
      (void) printf( "%d ", foo( i ) );
   return 0;
}
```

```
/* file2.c */
#include <stdio.h>
extern int c;

int a = 420;

int foo( void ) {
   static int b = 37;

   ++a;
   (void) printf( "%d ", b );
   return c;
}
```

Trying to separately compile each file and then link the resulting object modules

gcc –c file1.c	file1.c –>file1.o
gcc –c file2.c	file2.c –>file2.o
gcc file1.o file2.o	file1.o & file2.o –> a.out

results in just one error being reported. We discussed some of the problems/complications imposed on the compiler when trying to perform static semantic type checking with separate compilation.

What error will be reported (specify the symbol name and a general description of what the problem is). Hint: The error will be reported in the 3rd gcc call which attempts to link the already compiled and assembled object modules.

Assume we fixed this error so the program will fully compile/link. How many times does the variable **b** in function **foo()** get initialized (once or every time function foo() is called)?

C (unlike C++) requires all global and static variable initializations to be done completely at compile time so the values in the Data segment are known at compile time. Can we change the initialization of **b** in file2.c to be `static int b = bar();` assuming function bar() is properly defined? Why or why not?

Can we change the initialization of **i** in file1.c to be `int i = bar();` assuming function bar() is properly defined? Why or why not?

Identify two other potential semantic errors in this program that the C compiler and linker did not detect, but lint will identify.

1)

2)

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

   int a = 42;
   int b;

   foo2( a, &argc );

/*  1 */ (void) printf( "1: foo2 --> %p\n", foo2 );
/*  2 */ (void) printf( "2: a --> %p\n", &a );
/*  3 */ (void) printf( "3: argv --> %p\n", &argv );
/*  4 */ (void) printf( "4: b --> %p\n", &b );
/*  5 */ (void) printf( "5: argc --> %p\n", &argc );
}

void foo1( int *c, int d ) {

   static int e = 404;
   struct foo {int a; int b;} f;
   int g;

/*  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: f.b --> %p\n", &f.b );
/* 11 */ (void) printf( "11: g --> %p\n", &g );
/* 12 */ (void) printf( "12: f.a --> %p\n", &f.a );
}

void foo2( int h, int *i ) {

   int j[2];
   static int k;

   foo1( j, h );

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

}
```

________	smallest value (lowest memory address)
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	
________	largest value (highest memory addresses)

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

What is the name of Rick's favorite beer? ______________________

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

9. Extra Credit (12 points total extra credit)

What gets printed when this program is executed?

```
#include <stdio.h>

int main( void )
{
  char a[] = "CSE131 Rocks!";
  char *p = a + 3;

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

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

  printf( "%c", *p++ + 1 );              _______
  p = p + 2;
  printf( "%c", *p = *p + 3);            _______

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

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

  return 0;
}
```

For each of the following give the equivalent expression which in most cases is how the C compiler really treats/handles the expression.

```
int a[5];
struct foo { int a; double b; } * p;

p->a          _____________

a[2]          _____________
```

If array a is located at memory location 0x5000, what memory location is a[3]? ____________

If p has the value 0x6000, what memory location is p->b? ____________

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	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