### Final
**CSE 131**  
**Spring 2009**

<table>
<thead>
<tr>
<th>Page</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 1</td>
<td>(21 points)</td>
</tr>
<tr>
<td>Page 2</td>
<td>(29 points)</td>
</tr>
<tr>
<td>Page 3</td>
<td>(36 points)</td>
</tr>
<tr>
<td>Page 4</td>
<td>(27 points)</td>
</tr>
<tr>
<td>Page 5</td>
<td>(23 points)</td>
</tr>
<tr>
<td>Page 6</td>
<td>(38 points)</td>
</tr>
<tr>
<td>Page 7</td>
<td>(11 points)</td>
</tr>
<tr>
<td>Page 8</td>
<td>(19 points)</td>
</tr>
<tr>
<td>Page 9</td>
<td>(13 points)</td>
</tr>
<tr>
<td>Page 10</td>
<td>(17 points)</td>
</tr>
<tr>
<td>Page 11</td>
<td>(23 points)</td>
</tr>
<tr>
<td>Subtotal</td>
<td>(257 points)</td>
</tr>
<tr>
<td>Page 12</td>
<td>(18 points)</td>
</tr>
<tr>
<td>Extra Credit</td>
<td></td>
</tr>
</tbody>
</table>

**Total**

---

0
1. Given the following CUP grammar snippet (assuming all other Lexing and terminals are correct):

```
Stmt ::= Des AssignOp Des T_SEMI { System.out.println("1"); :} ;

Des ::= T_STAR { System.out.println("2"); :} Des { System.out.println("3"); :} ;
| T_PLUSPLUS { System.out.println("4"); :} Des { System.out.println("5"); :} ;
| T_AMPERSAND { System.out.println("6"); :} Des { System.out.println("7"); :} ;
| Des2 { System.out.println("8"); :}

Des2 ::= Des2 { System.out.println("9"); :} T_PLUSPLUS { System.out.println("10"); :} ;
| Des3 { System.out.println("11"); :}

Des3 ::= T_ID { System.out.println("12"); :}

AssignOp ::= T_ASSIGN { System.out.println("13"); :}
```

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

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

<table>
<thead>
<tr>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>*++x = &amp;*y++;</td>
</tr>
</tbody>
</table>

Does the above grammar agree with the C/C++ operator precedence?  
Does the above grammar agree with the C/C++ operator associativity?  
If variable y is defined to be type int *, what type must variable x be defined to be for this statement to be semantically correct?  

__________
2. Given the following Reduced-C code fragment:

```c
function : int foo( int & x, int * y, int z ) { /* Body of code not important for this question */ }

function : int main()
{
    int a;
    int b = -17;
    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.

```assembly
.equation MAIN_SAVE = -(92 + ______) ________ ________  ! Save space for 3 local vars + 1 temp
```

```
.set     _________________, %g1
.save    ________, %g1, ________
/* Initialize the local variables */
.set     _________, %o0
st      %o0, ______________ ! int b = -17;
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 temp
/* Copy saved return value stored in temp into local var a */
____    [%fp - 16], ______________
____    ______________________________ ! a = foo( ... );
/* return c; */
ld      ______________, ______________
```

```
```
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.

```java
public class Overloading_Final_Exam {
    public static void main (String [] args) {
        Larry stoogel = new Moe();
        Larry stooge2 = new Larry();
        Larry stooge3 = new Curly();
        Curly stooge4 = new Moe();
        Curly stooge5 = new Curly();
        Moe stooge6 = new Moe();

        ( (Moe) stoogel).print( (Curly) stooge6 );
        ( (Larry) stooge2).print( (Moe) stooge1 );
        ( (Curly) stooge3).print( (Moe) stooge4 );
        ( (Moe) stooge4).print( (Larry) stooge5 );
        ( (Curly) stooge5).print( (Larry) stooge6 );
        ( (Moe) stooge6).print( (Moe) stooge4 );

        stoogel.print( (Curly) stooge6 );
        stooge2.print( (Moe) stooge1 );
        stooge3.print( (Moe) stooge4 );
        stooge4.print( (Larry) stooge5 );
        stooge5.print( (Larry) stooge6 );
        stooge6.print( (Moe) stooge4 );

        stoogel.print( stooge6 );
        stooge2.print( stooge1 );
        stooge3.print( stooge4 );
        stooge4.print( stooge5 );
        stooge5.print( stooge6 );
        stooge6.print( stooge4 );
    }
}
```

```java
class Larry {
    public void print(Larry l) {
        System.out.println("Larry 1");
    }
}
```

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

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

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

    public void print(Moe m) {
        System.out.println("Moe 2");
    }
}
```

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 S with members of type int, float, and pointer to struct S named a, b, and ptr, respectively. Then define a variable named fubar which is an array of an array (with dimensions 5x9) of pointers to struct S such that fubar[4][1]->ptr = fubar[1][8]; is a valid expression. This will take more than one line of code.

For each of the following make no assumptions of what may be above or below each window of instructions unless otherwise stated. Use virtual register notation.

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

\[ r_1 = r_4 \times 126 \]

Optimize the following. Assume all registers except r2 are not needed (not alive) after the last statement. Note: Memory access (ld/st) is only between registers and memory.

\[
\begin{align*}
  r_4 &= x \\
  r_7 &= x \\
  r_5 &= r_4 \\
  r_4 &= r_7 - r_5 \\
  r_3 &= 4 \\
  x &= r_3 \\
  r_2 &= r_3 \times 5 \\
  r_7 &= r_2 + r_4 \\
  x &= r_7 \\
  r_2 &= x
\end{align*}
\]

/* x represents a memory location */
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!

```cpp
int a = 23;
int b = 34;
int c = 45;

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 fubar that is a pointer to a 2-d array of 5 rows by 8 columns 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 11 elements where each element is a pointer to a struct fubaz.
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;
}

.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

```c
a[3][0]
```

```
7. Given the C array declaration
int a[4][2];
```

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

```c
a[3][0]
```

```
| low memory | high memory |
```

Each box represents a byte in memory.

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

```
A
Expr :: =   Expr Op Designator
  |      Designator
  |
Op ::=      T_SLASH
  |      T_STAR
  |

B
Expr :: =   Expr Op Designator
  |      Designator
  |
Op ::=      T_STAR
  |      T_SLASH
  |

C
Expr :: =   Expr T_STAR Expr1
  |      Expr1
  |
Exp1 ::=      Expr1 T_SLASH Designator
  |      Designator
  |

D
Expr :: =   Expr T_SLASH Expr1
  |      Expr1
  |
Exp1 ::=      Expr1 T_STAR Designator
  |      Designator
  |
```

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

```c
void foo( int x )
{
    static int y = x + 1;
    cout << y--;  
    if ( x <= 2 && y >= 0 )
        foo( y + 1);
}
```

```c
int main()
{
    foo( 1 );
    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?

```c
#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;
    foo2( a, &b );
    /* 1 */ (void) printf( "1: c --> %p
", &c );
    /* 2 */ (void) printf( "2: argv --> %p
", &argv );
    /* 3 */ (void) printf( "3: malloc --> %p
", malloc(50) );
    /* 4 */ (void) printf( "4: b --> %p
", &b );
    /* 5 */ (void) printf( "5: argc --> %p
", &argc );
}

void foo1( int *d, int e ) {
    struct foo {int a; int b;} f;
    int g;
    /* 6 */ (void) printf( "6: f.b --> %p
", &f.b );
    /* 7 */ (void) printf( "7: d --> %p
", &d );
    /* 8 */ (void) printf( "8: e --> %p
", &e );
    /* 9 */ (void) printf( "9: f.a --> %p
", &f.a );
    /* 10 */ (void) printf( "10: foo2 --> %p
", foo2 );
    /* 11 */ (void) printf( "11: g --> %p
", &g );
}

void foo2( int h, int *i ) {
    static int j[3];
    int k = 411;
    foo1( i, k );
    /* 12 */ (void) printf( "12: j[1] --> %p
", &j[1] );
    /* 13 */ (void) printf( "13: h --> %p
", &h );
    /* 14 */ (void) printf( "14: a --> %p
", &a );
    /* 15 */ (void) printf( "15: i --> %p
", &i );
    /* 16 */ (void) printf( "16: j[0] --> %p
", &j[0] );
    /* 17 */ (void) printf( "17: k --> %p
", &k );
}
```

Who shot Mr. Burns? __________________________________

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.

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

int a;
float b;

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

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

    /* HERE */
    free( var2 );
}

int main() {
    foo( b, a );
    return 0;
}
```

hypothetical decimal memory locations

<table>
<thead>
<tr>
<th>low memory</th>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:</td>
<td>2000</td>
</tr>
<tr>
<td>b:</td>
<td></td>
</tr>
<tr>
<td>Heap</td>
<td>6000</td>
</tr>
<tr>
<td></td>
<td>40140</td>
</tr>
<tr>
<td></td>
<td>40240</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

%fp
10. Identify where each of the following program parts live in the Java runtime environment as discussed in class.

```java
public class Foo {
    private static Foo a;  a  _________________
    private 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 = 5;  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;
    }
}
```

Given the following definitions

```java
struct S1 { int a; };
struct S2 { int a; };

void foo ( struct S1 &a ) { }
struct S2 b;
```

a call to `foo( b )` passing in `b` as the actual argument will cause a compile error. Why?

Fix the call to the function `foo( b )` below to pass `b` to `foo()` without causing a compile error.

```java
foo( _________________________________ b );
```
11. Pick one of the following numbers to answer the questions below related to most calling conventions.

1) Prologue (in callee)  2) Epilogue (in callee)  3) Pre-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
- 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

```c
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.______________
```
12. Extra Credit

What gets printed when this program is executed?

```c
#include <stdio.h>

int main()
{
    char a[] = "10019";
    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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>postfix increment</td>
</tr>
<tr>
<td>--</td>
<td>postfix decrement</td>
</tr>
<tr>
<td>[ ]</td>
<td>array element</td>
</tr>
<tr>
<td>*</td>
<td>indirection</td>
</tr>
<tr>
<td>++</td>
<td>prefix increment</td>
</tr>
<tr>
<td>--</td>
<td>prefix decrement</td>
</tr>
<tr>
<td>&amp;</td>
<td>address-of</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
</tr>
</tbody>
</table>