By filling in the above and signing my name, I confirm I will complete this exam with the utmost integrity and in accordance with the Policy on Integrity of Scholarship.

Final
CSE 131
Winter 2014

Page 1 ___________ (21 points)
Page 2 ___________ (30 points)
Page 3 ___________ (24 points)
Page 4 ___________ (35 points)
Page 5 ___________ (21 points)
Page 6 ___________ (13 points)
Page 7 ___________ (24 points)
Page 8 ___________ (23 points)
Page 9 ___________ (23 points)
Page 10 ___________ (16 points)
Page 11 ___________ (20 points)
Page 12 ___________ (18 points)

Total ___________ (268 points)

255 points = 100%
13 points Extra Credit [~5%]

This exam is to be taken by yourself with closed books, closed notes, no electronic devices. You are allowed both sides of an 8.5"x11" sheet of paper handwritten by you.
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_AMPERSAND { : System.out.println("4"); : } Des { : System.out.println("5"); : }
| T_PLUSPLUS { : 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++;
```
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 = 42042;
    int c = b;
    a = foo( c, &a, b );

    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. See comments.

```
.section _________
.global _________
.align 4

_______:
    set     _________________, %g1
    save    _________________, %g1, _________________

    /* Initialize the local variables that have explicit initialization in this stack frame */
    set     _________________, %o0
    st      %o0, _________________  ! int b = 42042;
    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 templ into local var a */
    ___________ [%fp - 16], _________________
    ___________ _________________, _________________  ! a = foo( ... );

    /* return c; */
    ld       _________________, _________________  ! return c;

    ___________

    MAIN_SAVE = -(92 + _______) ___________  ! Save space for 3 local vars + 1 temp
```
3. In object-oriented languages like Java, determining which method code/instructions to bind to (to execute) is done at run time rather than at compile time (this is known as dynamic dispatch 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 - Use the letters A-F below to denote which string is printed.

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

class CPP extends C {
    public void print(CPP p) {
        System.out.println("CPP 1");
    }
    public void print(C p) {
        System.out.println("CPP 2");
    }
}

class JAVA extends CPP {
    public void print(JAVA p) {
        System.out.println("JAVA 1");
    }
    public void print(CPP p) {
        System.out.println("JAVA 2");
    }
    public void print(C p) {
        System.out.println("JAVA 3");
    }
}

public class Overloading_Final_Exam {
    public static void main(String[] args) {
        C lang1 = new C();
        C lang2 = new CPP();
        C lang3 = new JAVA();
        CPP lang4 = new CPP();
        CPP lang5 = new JAVA();
        JAVA lang6 = new JAVA();

        lang1.print(lang1);
        lang2.print(lang2);
        lang3.print(lang3);
        lang4.print(lang4);
        lang5.print(lang5);
        lang6.print(lang6);
        lang4.print((C) lang6);
        lang5.print((CPP) lang6);
        lang6.print((JAVA) lang6);
        (CPP) lang5.print((CPP) lang6);
        (JAVA) lang3.print((C) lang6);
    }
}
```

Now remove the entire `print(C p) {}` method in class CPP and remove the entire `print(CPP p) {}` method in class JAVA. Specify the output of each print() method with these changes below.

A) C 1
B) CPP 1
C) CPP 2
D) JAVA 1
E) JAVA 2
F) JAVA 3
4. Fill in the blanks of the following Reduced-C program with correct types to test if your global scope resolution operator works correctly. If it does, this program should compile without error. If it does not, this program should generate an assignment error at the line `y = ::x;`

```c
function : int main() {
    ______ x;
    int y;
    y = ::x; // If :: working, this line will not cause an error!
           // If :: not working, this line will cause an error!
    return 0;
}
```

In Reduced-C (which again follows closely the real C standard) all typedefs use _____________ name equivalence. Struct operations (like =, ==, ! =) use _____________ name equivalence.

In RC (and C/C++), we do not support the assignment of an entire array to another array (of the same type) using the assignment operator. However, we do support assignment of an entire struct instance to another struct instance of the same type. Using this fact, fill in the template of the code below, allowing arrays to piggy-back on a struct type to simulate entire-array assignment/copy that is semantically and logically correct.

```c
structdef INTARR5 { int [5] a; }
int [5] x;
int [5] y;
function : void foo() {
    // x = y would be a semantic error, but the following will work without an error
    ______ _____________________ ______ x[0] = ______ _____________________ ______ y[0];
}
```

Given the definitions below, indicate whether each expression is either a 1) R-val  2) Modifiable L-val  3) Non-Modifiable L-val

```c
function : int & foo1() { /* Function body not important. */ }
function : int * foo2() { /* Function body not important. */ }
const int x = 5;
int y;
int [5] a;
int *p = &y;

____ a[2]   ____ &y   ____ a   ____ x   ____ x + y
____ p   ____ *p   ____ *&p   ____ &p   ____ y
____ 42   ____ (float *)p   ____ *(float *)p   ____ (float *)&y   ____ *(float *)&y
____ ::y   ____ foo1()   ____ foo2()   ____ foo1()++   ____ y = *foo2()
____ *p++   ____ ++*p   ____ *++p   ____ --*++p   ____ ++*p--
```
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 = 1;
int b = 3;
int c = 5;
int zach;

int & fubar( int * x, int & y, int z )
{
    static int m = *x;
    *x = *x + 3;
    y = y + 3;
    z = z + 3;
    zach = m++;

    return m;
}

void foo1( int d, int * e, int & f )
{
    d = d + 2;
    *e = *e + 2;
    f = f + 2;

    cout << a << endl;    // ______
    cout << b << endl;    // ______
    cout << c << endl;    // ______
    cout << d << endl;    // ______
    cout << *e << endl;   // ______
    cout << f << endl;    // ______
    cout << zach << endl; // ______
    cout << fubar( &d, d, d ) << endl; // ______
    cout << fubar( e, *e, *e ) << endl; // ______
    cout << fubar( &f, f, f ) << endl;  // ______
    cout << a << endl;    // ______
    cout << b << endl;    // ______
    cout << c << endl;    // ______
    cout << d << endl;    // ______
    cout << *e << endl;   // ______
    cout << f << endl;    // ______
    cout << zach << endl; // ______
}

int main()
{
    foo1( a, &b, c );
    cout << a << endl;    // ______
    cout << b << endl;    // ______
    cout << c << endl;    // ______
    cout << zach << endl; // ______
    return 0;
}
```
6. 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 = 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;
}
```

<table>
<thead>
<tr>
<th>hypothetical decimal memory locations</th>
<th>low memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a:</strong></td>
<td>3000</td>
</tr>
<tr>
<td><strong>b:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Heap</strong></td>
<td>7000</td>
</tr>
<tr>
<td><strong>%fp</strong></td>
<td>40240</td>
</tr>
<tr>
<td></td>
<td>40340</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

| **%fp**                                | 40240      |
|                                        | 40340      |
|                                        | ...        |
7. Given the C array declaration

```c
int b[2][4];
```

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

```c
b[1][0]
```

Each box represents a byte in memory.

Using the Right-Left rule write the C definition of a variable named foo that is a pointer to an array of 9 elements where each element is a pointer to a function that takes a pointer to a struct RT as the single parameter and returns a pointer to a 3x17 2-D array where each element is a pointer to a pointer to a struct Fubar.

Identify whether each of the following will cause an underlying bit pattern change.

```c
int a = 5;
float b = -4.20;
int * ptr1;
float * ptr2;
void foo( float x, float & y ) { /* … */ }
```

A) Yes – Underlying bit pattern change
B) No – No underlying bit pattern change

Identify whether each of the following will cause an underlying bit pattern change.

```c
int a = 5;
float b = -4.20;
int * ptr1;
float * ptr2;
void foo( float x, float & y ) { /* … */ }
```

Use the numbers 1 through 4 to indicate when you would expect to see each error listed below (assuming a compiled, not an interpreted, language).

(1) compile-time (2) link-time (3) load-time (4) run-time

| Error message: Left-hand side is not a modifiable l-value. | A) Yes – Underlying bit pattern change |
| Running "gcc a.o b.o" gives the message "Multiple definition of 'main'". | B) No – No underlying bit pattern change |
| An "array-index-out-of-bounds" error using a non-constant index expression. |
| Undeclared identifier "foo". |
| An "array-index-out-of-bounds" error using a constant-valued index expression. |
| Segmentation fault. |
| Running "gcc someModule.o" gives the message "Undefined reference to 'main'". |
| Non-addressable argument of type %T to address-of operator. |
| Bus error. |
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;

int main( int argc, char *argv[] ) {
    int b;
    double c;
    foo2( a, &b );
    /* 1 */ (void) printf( "1: argc --> %p\n", &argc );
    /* 2 */ (void) printf( "2: c --> %p\n", &c );
    /* 3 */ (void) printf( "3: argv --> %p\n", &argv );
    /* 4 */ (void) printf( "4: malloc --> %p\n", malloc(50) );
    /* 5 */ (void) printf( "5: b --> %p\n", &b );
}

void foo1( int *d, int e ) {
    static struct foo {int a; int b;} f = { 1, 2 };
    int g;
    /* 6 */ (void) printf( "6: f.b --> %p\n", &f.b );
    /* 7 */ (void) printf( "7: d --> %p\n", &d );
    /* 8 */ (void) printf( "8: e --> %p\n", &e );
    /* 9 */ (void) printf( "9: f.a --> %p\n", &f.a );
    /* 10 */ (void) printf( "10: foo2 --> %p\n", foo2 );
    /* 11 */ (void) printf( "11: g --> %p\n", &g );
}

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

You are compiling foo1.c and foo2.c together with gcc. If foo1.c has the following statement
extern int a;
indicate whether each of the following would cause a linkage editor error or not if put separately (one at a time) in foo2.c?

A) Yes - Linkage Editor Error
B) No - No Linkage Editor Error

### extern int a;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>___ extern int a;</td>
<td>A) Yes - Linkage Editor Error</td>
</tr>
<tr>
<td>___ int a( float b ) { return (int)b; }</td>
<td>A) Yes - Linkage Editor Error</td>
</tr>
<tr>
<td>___ static double a;</td>
<td>B) No - No Linkage Editor Error</td>
</tr>
<tr>
<td>___ static int a( float b ) { return (int)b; }</td>
<td>A) Yes - Linkage Editor Error</td>
</tr>
<tr>
<td>___ int a = 42;</td>
<td>B) No - No Linkage Editor Error</td>
</tr>
<tr>
<td>___ extern void a( char * );</td>
<td>A) Yes - Linkage Editor Error</td>
</tr>
</tbody>
</table>
9. Pick one of the following numbers to answer the questions below related to the cdecl calling convention covered in class.

1) Pre-Call (Caller) 2) call/jsr 3) Prologue (Callee) 4) Epilogue (Callee) 5) Post-Call (Caller)

_____ Allocates space for return value   _____ Restores caller-save registers
_____ Copies actual arguments into argument space  _____ Saves registers in callee-save scheme
_____ Allocates space for actual arguments   _____ Saves %pc into the return address location
_____ Stores return value into return value location   _____ Retrieves saved return address for return
_____ Allocates space for local variables & temps   _____ Performs initialization of local variables
_____ Saves registers in caller-save scheme   _____ Restores callee-save registers
_____ Retrieves return value from return value location   _____ Deallocates argument space in cdecl mode
_____ Copies args passed in regs to param stack space   _____ Deallocates local variable & temps space

Many older 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 thought of as 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?

```c
* (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.

```c
fubaz.________________
```
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;
    private int b;

    public Foo() { code for Foo() }
        a = this;
        ++b;
    }

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

private void method( double e ) {
    double f = e;
}
}
```

Given this short simple Reduced-C program, show how you tested call-by-reference parameters in your compiler to ensure the implementation is truly call-by-reference vs. any other implementation.

```c
int global = 5;
/* A */
function : void foo( int & param )
    { /* B */
        param = 10;
        /* C */
    }
/* D */
function : void main()
    { foo( global );
        /* E */
    }
/* F */
```

Which two places should you put the following statement to output the value of global to accurately test call-by-reference implementation in the (your) compiler (specify the LETTER associated with the location)? And what value should be printed with each output statement if the (your) compiler correctly implemented call-by-reference?

```
cout << global << endl;
```

_____ Location one  Expected output _____
_____ Location two  Expected output _____
11. Match the compilation process with the various tasks done in the compilation sequence.

1) C++ Preprocessor  2) C++ Compiler      3) Assembler      4) Linkage Editor      5) Loader

_____ combines all object modules into a single executable file.
_____ performs name mangling of function names.
_____ performs semantic analysis on its input high-level language (HLL).
_____ takes an executable file on disk and makes it ready to execute in memory.
_____ translates assembly code into machine code.
_____ expands # directives from its input high-level language (HLL).
_____ performs syntax analysis on its input high-level language (HLL).
_____ resolves undefined external symbols with defined global symbols in other modules.
_____ translates high-level language (HLL) code into assembly code.
_____ zero fills the BSS segment in memory.
_____ puts globally defined symbols in the export list of the resulting object file.

Use virtual register notation for each of the following.

Perform step-wise peephole optimization on the following window of pseudo three-address instructions (max of three operands are allowed in an instruction – up to two source and one destination):

... other instructions ...

\[ r3 = r2 \times 16 \]  \[ r0 = r3 \]  \[ r3 = 13 + 7 \]  \[ r0 = r3 \]  \[ r3 = 13 + 7 \]  \[ r3 = 13 + 7 \]  \[ \ldots \text{other instructions...} \]

step 1 \hspace{1cm} \text{(instruction eliminated)} \hspace{1cm} \text{step 2} \hspace{1cm} \text{step 3}

Change the following instructions into two instructions which are most likely a time improvement over the set of instructions when it comes to actual execution time.

\[ r1 = r2 \times r5 \]
\[ r3 = r1 \]
\[ r6 = r2 \times r5 \]
\[ r4 = r6 \]
\[ r1 = \ldots \]
\[ r6 = \ldots \]
\[ \ldots = \ldots r3 \]
\[ \ldots = \ldots r4 \]

What terms describe these particular kinds of peephole optimizations? List two that apply.

1)  
2)  

What C/C++ compiler option should you use to produce a .o file from a .c file? ______
What C/C++ compiler option should you use to produce a .s file from a .c file? ______
12. What gets printed when this C program is executed?

```c
#include <stdio.h>

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

    printf( "%c\n", *p-- );
    printf( "%c\n", ++*--p );
    printf( "%c\n", 2[a]++ );
    printf( "%c\n", p[-1] = *(a+5) );
    printf( "%c\n", ++p++ );
    printf( "%c\n", +++p );
    printf( "%d\n", p - a );
    printf( "%s\n", a );
    return 0;
}
```

What is Rick's favorite guilty pleasure? _________________________________

What gets printed if the following function is invoked as `recurse( 2, 10 )`? Hint: Draw stack frames.

```c
int recurse( int a, int b ) {
    int local = b - a;
    int result;

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

    if ( b > 7 )
        result = local + recurse( a, b - 1 );
    else
        result = local;

    printf( "%d\n", result );
    return result;
}
```

Put answers here

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>++ postfix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- postfix decrement</td>
<td></td>
</tr>
<tr>
<td>[] array element</td>
<td></td>
</tr>
<tr>
<td>() function call</td>
<td></td>
</tr>
<tr>
<td>-&gt; struct/union pointer</td>
<td></td>
</tr>
<tr>
<td>. struct/union member</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td></td>
</tr>
<tr>
<td>&amp; address-of</td>
<td></td>
</tr>
<tr>
<td>sizeof</td>
<td>size of type/object</td>
</tr>
<tr>
<td>(type)</td>
<td>type cast</td>
</tr>
<tr>
<td>* multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td>/ division</td>
<td></td>
</tr>
<tr>
<td>% modulus</td>
<td></td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>