Login Name _____ **Name _____**

Student ID _____ **Signature _____**

Final
CSE 131
Fall 2014

Page 1	**_____**	**(35 points)**
Page 2	**_____**	**(24 points)**
Page 3	**_____**	**(30 points)**
Page 4	**_____**	**(32 points)**
Page 5	**_____**	**(25 points)**
Page 6	**_____**	**(36 points)**
Page 7	**_____**	**(30 points)**
Page 8	**_____**	**(48 points)**
Subtotal	**_____**	**(260 points) = 100%**
Page 9 Extra Credit	**_____**	**(18 points) [7% Extra Credit]**
Total	**_____**	

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):

```
Expr ::=   Expr T_MINUS {: System.out.println("A"); :} Des {: System.out.println("B"); :}
     |     Des {: System.out.println("C"); :}
     ;

Des  ::=   T_STAR {: System.out.println("D"); :} Des {: System.out.println("E"); :}
     |     T_MINUSMINUS {: System.out.println("F"); :} Des {: System.out.println("G"); :}
     |     T_AMPERSAND {: System.out.println("H"); :} Des {: System.out.println("I"); :}
     |     Des2
     ;

Des2 ::=   Des2 {: System.out.println("J"); :} T_MINUSMINUS {: System.out.println("K"); :}
     |     Des3
     ;

Des3 ::=   T_ID
     ;
```

What is the output when parsing the follow expression (you should have 23 lines/letters in your output) [23pts]:

```
*&*x-- - --*y-- - **z
```

Output

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

In the above grammar, which has higher precedence: post-decrement operator or address-of operator, or do they have the same precedence? [1pt]

_____

What type of associativity does the `Des2` rule have in the above grammar? [1pt]

_____

Fill in the blanks of the below RC program with the appropriate types to be semantically correct: [6pts]

```
function : void main() {

     __________ a = 7.5;

     __________ b = 4.2;

     __________ w = &a;

     __________ x = &a;

     __________ y = &b;

     __________ z = &w;

     cout << *&*x-- - --*y-- - **z;
}
```

What is the output of the above program when executed? [4pts]

_____

2. Given the following Reduced-C code fragment: [24pts]

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

function : int main()
{
  int a = 12345;
  int * b = &a;

  ++foo( a, *b );

  return a;
}
```

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 _________
        .align 4
        .global  _________
_______:
        set     SAVE.main, %g1
        save    %sp, %g1, %sp

        /* Initialize the local variables in this stack frame */

        set     _______________, %o0

        st      %o0, _______________                    ! int a = 12345;

        ______  ___________________________________, %o0   ! large blank can be one or two operands

        st      %o0, _______________                    ! int * b = &a;

        /* Set up the 2 actual arguments to foo() */

        ______  ___________________________________, %o0   ! large blank can be one or two operands

        ______  ___________________________________, %o1   ! large blank can be one or two operands

        ______  ___________________________________, %o1   ! large blank can be one or two operands

        call    foo                                     ! Call function foo()
        nop

        st       _______________, [%fp - 12]            ! Save return value into local temp1

        /* Handle increment after function call */

        ld       [%fp - 12], %o0

        _______  ___________________________________, %o1   ! large blank can be one or two operands

        inc      _______________

        st       _______________, [%o0]                 ! ++foo( ... );

        /* return a; */

        _______  _______________, _______________       ! return a;

        ____________

        ____________

        SAVE.main = -(92 + 12) & -8                     ! Save space for 2 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. [30pts]

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

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

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

class Ugly extends Bad {
    public void print(Good p) {
        System.out.println("Ugly 1");
    }

    public void print(Bad p) {
        System.out.println("Ugly 2");
    }

    public void print(Ugly p) {
        System.out.println("Ugly 3");
    }
}

public class Overloading_Final_Exam {
    public static void main (String [] args) {
        Good cowboy1 = new Ugly();
        Good cowboy2 = new Bad();
        Good cowboy3 = new Good();
        Bad  cowboy4 = new Ugly();
        Bad  cowboy5 = new Bad();
        Ugly cowboy6 = new Ugly();

        cowboy1.print( cowboy1 );                  _____________________________   _____________________________

        cowboy2.print( cowboy2 );                  _____________________________   _____________________________

        cowboy3.print( cowboy3 );                  _____________________________   _____________________________

        cowboy4.print( cowboy4 );                  _____________________________   _____________________________

        cowboy5.print( cowboy5 );                  _____________________________   _____________________________

        cowboy6.print( cowboy6 );                  _____________________________   _____________________________

        cowboy2.print( (Ugly) cowboy6 );           _____________________________   _____________________________

        cowboy4.print( (Bad) cowboy6 );            _____________________________   _____________________________

        cowboy6.print( (Good) cowboy6 );           _____________________________   _____________________________

        ( (Bad) cowboy2 ).print(cowboy2);          _____________________________   _____________________________

        ( (Bad) cowboy2 ).print(cowboy4);          _____________________________   _____________________________

        ( (Bad) cowboy2 ).print(cowboy6);          _____________________________   _____________________________

        ( (Ugly) cowboy1 ).print(cowboy2);         _____________________________   _____________________________

        ( (Ugly) cowboy1 ).print(cowboy4);         _____________________________   _____________________________

        ( (Ugly) cowboy1 ).print(cowboy6);         _____________________________   _____________________________
    }
}
```

Now <u>remove</u> the entire `print(Good p) {}` method in `class Bad` and <u>remove</u> the entire `print(Bad p) {}` method in `class Ugly`. Specify the output of each print() method <u>with these changes</u> below.

4. Give the order of the typical GCC compilation stages and on to actual execution as discussed in class [4pts]:

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

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

Give the order of the general phases of compilation in a typical compiler as discussed in class [4pts]:

A – Target language (e.g. assembly)
B – Machine-independent code improvement (optional)
C – Machine-specific code improvement (optional)
D – Parser (Semantic analysis/intermediate code gen.)
E – Scanner (Lexical analysis)
F – Parser (Syntax analysis)
G – Target code generation
H – Source language (e.g. C)

_____ –> _____ –> _____ –> _____ –> _____ –> _____ –> _____ –> _____

Give an example of a non-converting type cast (underlying bit pattern does not change) [2pts]:

```
int i = 5;
float f = _____ ;
```

What are the values of `a` and `b` after the following two Reduced-C statements? [4pts]

```
int a = 5;
bool b = (++a > 5) && ((a = 2) != 0);
```

Value of `a` is _____ Value of `b` is _____

Fill in the blanks [2pts each]:

_____ analysis deals with verifying correct structure of a program.

_____ analysis deals with verifying correct meaning of a program

A(n) _____ performs thorough analysis and nontrivial transformations on a program in language L1 into an equivalent program in language L2. This is in contrast to a(n) _____ which directly performs the operations implied by the program.

The _____ is the main data structure used in the compiler to store/retrieve information about names (variables, functions, etc) and other attributes (e.g. types) associated with these names.

There are two types of programming errors can easily occur when the programmer is left to deal with dynamic memory management. Garbage collection essentially eliminates these two errors, which are

_____ and _____.

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

The technique of _____ is typically employed by compilers to enable overloaded functions to resolve into unique symbols/labels.

5. Given the Reduced-C code shown in the box below, complete the SPARC Assembly language statements that might be emitted by a compliant Reduced-C compiler from this quarter for function main(). Include the necessary run-time checks. [25pts]

```
     /* Reduced-C */
structdef MS {
  int x;
  int y;
};

function : void main()
{
    MS * x;
    new x;
    delete x;
}
```

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

_______:
        set     SAVE.main, %g1
        save    %sp, %g1, %sp

        ! new x;

        set     1, %o0

        set     __________, %o1

        call    __________

        nop

        ______  __________, __________

        ! delete x;

        ld      __________, %o0

        ______  __________, __________

        ______  .Delete_Error12

        nop

        call    ____________

        nop

        st      __________, __________

        ______  .L47

        nop

   .Delete_Error12:
        !! Assume .DeleteMsg string already defined in ".data" section

        ______  .DeleteMsg, __________

        call    __________

        nop

        set     __________, __________

        call    __________

        nop

   .L47:

        _________

        _________

SAVE.main = -(92 + 4) & -8
```

6. Given the C array declaration

C

```
double * a[2][2];
```

Mark with an **A** the memory location(s) where we would find the array element `a[1][0]`:

`a:` Each box represents a byte in memory. [4pts]



`low memory` `high memory`

Given the definitions below, indicate what type of object each expression is. [25pts]

```
function : int & f1() { /* Function body not important. */ }
function : int * f2() { /* Function body not important. */ }
int[5] a;
int b = 1;
const int c = 5;
int *d = &b;
```

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

____ `a`	____ `a[4]`	____ `&a[4]`	____ `c`	____ `b + c`
____ `d`	____ `*d`	____ `*&*d`	____ `&*d`	____ `b`
____ `::b`	____ `42`	____ `a[a[b]]`	____ `f1() + f1()`	____ `f1()`
____ `&b`	____ `(float) b`	____ `f1()++`	____ `++*f2()`	____ `*f2()`
____ `--*++d`	____ `*(float *)d`	____ `++a[0]`	____ `nullptr`	____ `f2()`

Use virtual register notation for each of the following.

Change the following instruction into three instructions which are most likely a time improvement over the single instruction when it comes to actual execution time. You can assume register `r7` is free. [4pts]

```
r2 = r4 * 1022
```

_____
_____
_____

What term describes this particular kind of peephole optimization?

Change the following instruction into another single instruction which is most likely a time improvement over the current instruction when it comes to actual execution time. [2pts]

```
r1 = 16 % 5
```

_____

What term describes this particular kind of peephole optimization?

Change the following instructions into a single instruction which is most likely a time improvement over the current instructions when it comes to actual execution time. Assume registers `r1`, `r3`, and `r4` are not needed after the last statement. [1pt]

```
r1 = 7
r2 = r2 * 1
r3 = r1 + 13
r4 = r3 - r1
r5 = r4 + r2
```

_____

7. 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** stored in memory for each 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. [26pts]

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

int   a = 88;
float b;

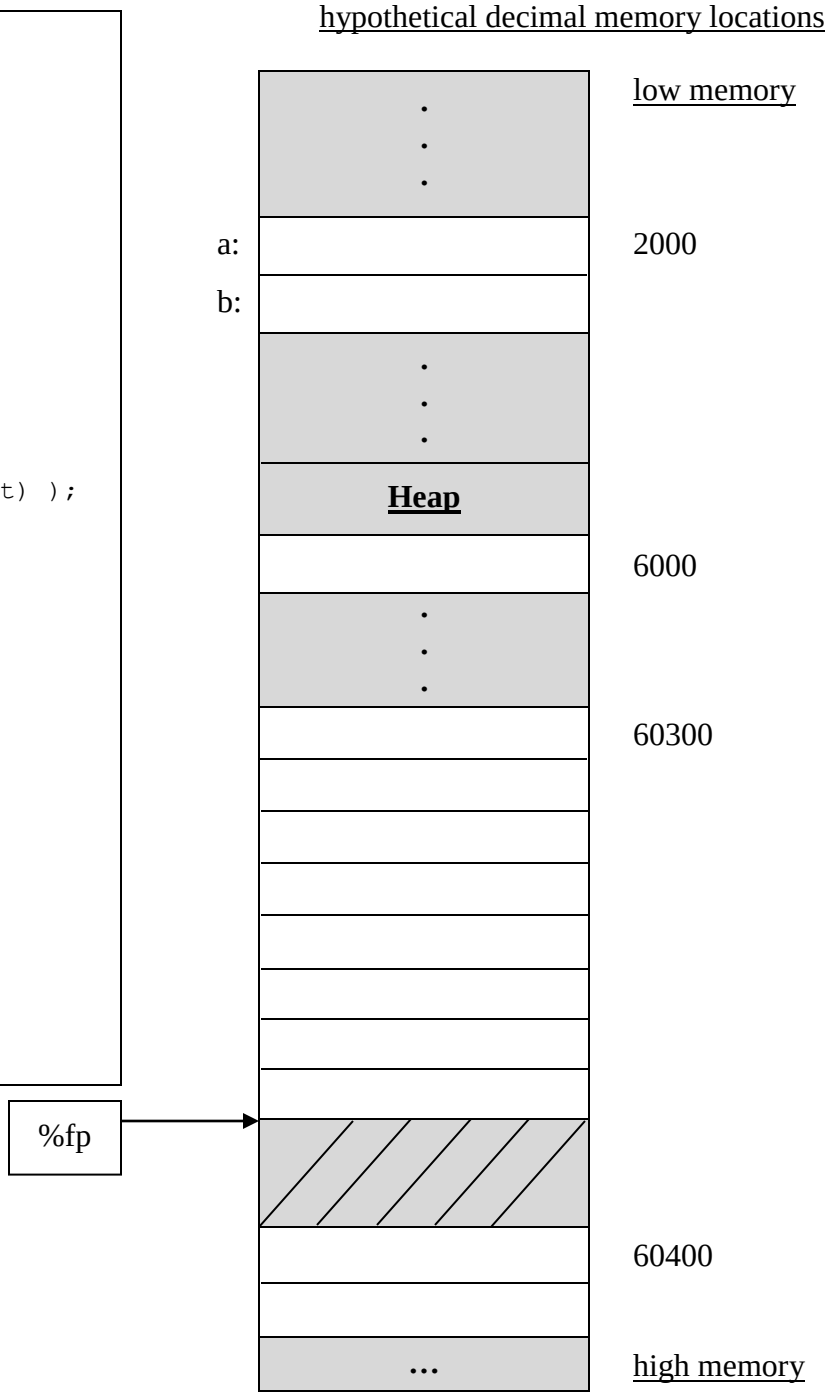
void foo( float & f, int i ) {
  int * var1;
  int   var2;
  struct fubar var3[2];

  var2 = -21;
  var1 = (int *) calloc( 1, sizeof(int) );
  f = 1.23;
  var3[0].c = b;
  var3[1].a = i + 2;
  var3[1].b = &var3[1].a;
  i = 75;
  var3[0].a = a;
  var3[0].b = &i;
  var3[1].c = f;
  *var1 = var2 + 8;

  /* HERE */

  free( var1 );
}

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



Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named fubaz that is a pointer to a 2D array of 19 rows by 4 columns where each element is a pointer to a pointer to a function that takes a pointer to a pointer to a short as a single parameter and returns a pointer to an array of 27 elements where each element is a pointer to an int. [4pts]

_____;

8. Given the following C type definitions: [24pts]

```
struct bar {
  char  a;
  short b;
  char  c;
  int   d;
};

struct foo {
  char       e;
  int *      f;
  struct bar g;
  double     h;
};

struct fubar {
  int        i;
  short      j[3];
  struct foo k;
  char       l;
};

struct fubar fubaz;
```

Hint: Draw the memory layout of these structs (including padding) on the scratch paper at the end of this exam.

What is the `sizeof( struct bar )`? _____ What is the `sizeof( struct foo )`? _____

What is the `sizeof( struct fubar )`? _____ What is the `offsetof( struct fubar, k.g.b )`? _____

If `struct fubar` had been defined as `union fubar` instead, what would be the `sizeof(union fubar)`? _____

What is the `sizeof( struct fubar )` if members of all 3 structs are reordered largest to smallest? _____

Given the following C++ code where ??? may represent different parameter passing modes, what values do you expect to be printed if the parameter passing mode is: [24pts]

```
int x = 6;
int y = 9;

void bar(int * x, int * y) {
    cout << *x << endl;
    cout << *y << endl;
    *x = 5;
    *y = 10;
    y = x;
    cout << *x << endl;
    cout << *y << endl;
    cout << ::x << endl;
    cout << ::y << endl;
}

void foo(int ??? x, int y) {
    cout << x << endl;
    cout << y << endl;
    bar(&x, &y);
    cout << x << endl;
    cout << y << endl;
    cout << ::x << endl;
    cout << ::y << endl;
}

int main()
{
    foo(x, y);
    return 0;
}
```

call-by-value?	**call-by-reference?**
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

9. Extra Credit

What gets printed when the following C program is executed? [18pts]

```
#include <stdio.h>
int main() {
  char a[] = "TRITON!!";
  char *p = a;

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

  printf( "%c\n", (p[6] = p[0]) - 15 );                  ______

  printf( "%c\n", (*p++ = 'A') ? (*a - 2) : 'Z' );       ______

  printf( "%c\n", (*p = --p[3]) - 9 );                   ______

  printf( "%c\n", *(p++ - 2) = '*' );                    ______

  printf( "%c\n", (5[a] = *p++ + 2) - 19 );              ______

  printf( "%c\n", (p[-4] = 'G') + 12 );                  ______

  printf( "%c\n", p[-1] = p[2] = (*p - 10) );            ______

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

  return 0;
}
```

A portion of the C Operator Precedence Table

Operator	Associativity
++ postfix increment -- postfix decrement [] array element () function call	L to R
* indirection ++ prefix increment -- prefix decrement & address-of sizeof size of type/object (type) type cast	R to L
* multiplication / division % modulus	L to R
+ addition - subtraction	L to R
.	
.	
= assignment	R to L

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

Scratch Paper

Scratch Paper