Student ID _____ **Name _____**

Login Name _____ **Signature _____**

Final
CSE 131
Winter 2010

Page 1	**_____ (26 points)**
Page 2	**_____ (15 points)**
Page 3	**_____ (36 points)**
Page 4	**_____ (25 points)**
Page 5	**_____ (23 points)**
Page 6	**_____ (14 points)**
Page 7	**_____ (15 points)**
Page 8	**_____ (19 points)**
Page 9	**_____ (13 points)**
Page 10	**_____ (22 points)**
Page 11	**_____ (11 points)**
Subtotal	**_____(219 points = 100%)**
Page 12 Extra Credit	**_____ (14 points) [over 6% EC]**
Total	**_____**

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

```
Expr ::=   Des AssignOp Expr {: System.out.println("0"); :}
     |     Des {: System.out.println("1"); :}
     ;

Des  ::=   T_STAR {: System.out.println("2"); :} Des {: System.out.println("10"); :}
     |     T_AMPERSAND {: System.out.println("3"); :} Des {: System.out.println("11"); :}
     |     T_PLUSPLUS {: System.out.println("4"); :} Des {: System.out.println("12"); :}
     |     Des2 {: System.out.println("5"); :}
     ;

Des2 ::=   Des2 {: System.out.println("6"); :} T_PLUSPLUS {: System.out.println("13"); :}
     |     Des3 {: System.out.println("7"); :}
     ;

Des3 ::=   T_ID {: System.out.println("8"); :}
     ;

AssignOp ::=  T_ASSIGN {: System.out.println("9"); :}
     ;
```

What is the output when parsing the follow expression (you should have 24 lines/numbers in your output):

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

Output

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

If variable `x` is defined to be type `float *`, what types must variables `y` and `z` be defined for this expression to be semantically correct?

_____ `y;` _____ `z;`

2. Fill in the blanks with one of the following

A) Front End
B) Intermediate Representation (IR)
C) Back End

The semantic analyzer is considered part of _____.
A complete AST is considered part of _____.
Machine independent code improvements are usually performed in _____.
Machine dependent code improvements are usually performed in _____.
The parser is considered part of _____.
The code generator is considered part of _____.
The lexical analyzer is considered part of _____.

Suppose that in the next version of Reduced-C, we extend support for the types that can be used for if-statement and while-statement conditional expressions. Specifically, we now allow expressions of type int for conditional expressions in addition to type bool. What this means is that the following code

```
int i = 6;
while( i-- )
{
   if( i % 2 )
      cout << i;
}
```

is legal and outputs 531 when executed. Basically, an implicit typecast to bool is assumed for the conditional expressions. Another way to look at it is that an implicit "!= 0" is appended to the conditional expression (for example: `(i--) != 0` and `(i % 2) != 0`), making the resulting expression be of type bool.

(1) Give a high-level explanation of what changes, if any, will need to be made to the semantic analysis part of the compiler (Project I), which is involved with name bindings and type checking.

(2) Give a high-level explanation of what changes, if any, will need to be made to the code generation part of the compiler (Project II).

Circle T for true or F for false:

T or F In our CUP grammar, operators with high precedence are high in the chain of Expr production rules.

T or F In our CUP grammar, operators with high precedence are high in the parse.

T or F In our CUP grammar, unary operator rules are written with left recursion to support left associativity.

T or F In our CUP grammar, we use the maximal munch principle so expressions like a+++++b parse the same as a++ + ++b.

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.

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

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

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

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

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

    public void print(Curly c) {
        System.out.println("Larry 3");
    }
}
```

```
public class Overloading_Final_Exam {
    public static void main (String [] args) {

        Curly stooge1 = new Moe();
        Curly stooge2 = new Larry();
        Curly stooge3 = new Curly();
        Moe   stooge4 = new Larry();
        Moe   stooge5 = new Moe();
        Larry stooge6 = new Larry();

        stooge1.print( stooge6 );                          _____________________   _____________________

        stooge2.print( stooge1 );                          _____________________   _____________________

        stooge3.print( stooge4 );                          _____________________   _____________________

        stooge4.print( stooge5 );                          _____________________   _____________________

        stooge5.print( stooge6 );                          _____________________   _____________________

        stooge6.print( stooge6 );                          _____________________   _____________________


        stooge1.print( (Moe) stooge6 );                    _____________________   _____________________

        stooge2.print( (Moe) stooge1 );                    _____________________   _____________________

        stooge3.print( (Larry) stooge4 );                  _____________________   _____________________

        stooge4.print( (Curly) stooge5 );                  _____________________   _____________________

        stooge5.print( (Moe) stooge6 );                    _____________________   _____________________

        stooge6.print( (Moe) stooge6 );                    _____________________   _____________________


        ( (Moe) stooge1).print( (Moe) stooge6 );           _____________________   _____________________

        ( (Larry) stooge2).print( (Moe) stooge1 );         _____________________   _____________________

        ( (Curly) stooge3).print( (Larry) stooge4 );       _____________________   _____________________

        ( (Moe) stooge4).print( (Curly) stooge5 );         _____________________   _____________________

        ( (Curly) stooge5).print( (Moe) stooge6 );         _____________________   _____________________

        ( (Moe) stooge6).print( (Moe) stooge6 );           _____________________   _____________________
    }
}
```

Now <u>remove</u> the entire method `print(Moe m){}` in `class Moe` and <u>remove</u> the entire method `print(Curly c){}` in `class Larry`. Specify the output of each print() method <u>with these changes</u> below.

4. In your Project 2, how did you (your group) handle code gen for short-circuiting with the logical AND and logical OR operators? Be specific how your project implemented this!

Using Reduced-C syntax, first define a struct A with members of type bool and pointer to struct A named x and ptr, respectively. Then define a variable named `foo` which is an array of an array (with dimensions 7x4) of pointers to struct A such that `foo[0][0]->ptr = foo[6][3];` is a valid expression. This will take more than one line of Reduced-C code. You can use two columns in the space below if you wish: struct definition on the left, array definition on the right.

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 which are most likely a time improvement over the single instruction when it comes to actual code generation. `x` represents a memory location.

```
r5 = r3 + r7
x = r5
r5 = x
r7 = r6 + r5
x = r7
```

_____

_____

_____

What term describes this particular kind of peephole optimization?

Change the following into three instructions which are most likely a time improvement over the single instruction when it comes to actual code generation.

```
r1 = r2 * 130
```

_____

_____

_____

What term describes this particular kind of peephole optimization?

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 = 45;
int b = 67;
int c = 89;

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;
}
```

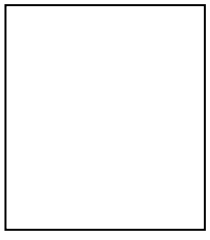
Using the Right-Left rule write the C definition of a variable named foo that is a pointer to an array of 7 elements where each element is a pointer to a function that takes a pointer to a pointer to a double as a single parameter and returns a pointer to a 2-d array of 4 rows and 13 columns where each element is a pointer to a struct bar.

6. Global and static variables are only initialized once and initialization guards can be used to enforce this property for each global and static variable. However, in certain cases, an initialization guard can be avoided by utilizing the data or bss section to initialize a global or static variable. Consider the following Reduced-C program:

```
const int negone = -1;

static int uig;                    //A
static int cig = 5 + negone;       //B
bool rig = uig == cig;             //C

function : int foo( int i )
{
   static int ril = i + 1;         //D
   static bool uil;                //E

   if(uil)
      return cig * ril * negone;

   uil = true;
   return cig;
}

function : void main()
{
   while( --cig > 1 )
   {
      static int cil = 3;          //F
      cout << foo( ++cil + cig ) << endl;
   }
   /* ... May be other code here that does not output anything. */
}
```

Each global and static variable is marked with a letter A-F. For each, indicate below whether an initialization guard is absolutely needed (answer "Y") or not needed (answer "N"). Give a very brief explanation for why or why not.

	Y/N	Reason
(A)	____	_____
(B)	____	_____
(C)	____	_____
(D)	____	_____
(E)	____	_____
(F)	____	_____

What is the output of the above program? 

7. Given the C array declaration

```
      int a[3][2];
```
C

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

```
      a[1][1]
a:
```

```
low memory                                                    high memory
```

Each box above represents a byte in memory.

Why do function pointers in Reduced-C need to be 8 bytes? Be specific.

With respect to the assembly code generated, what is different between a global variable definition and a static variable definition? For example, `int x;` vs. `static int x;` Be specific.

With respect to the assembly code generated, what is different between accessing a global variable that is defined in one file and accessing an extern declaration of a global variable of the same name declared in another file? For example, `int y; y = 5;` vs. `extern int y; y = 5;` This question asks only about the assembly code to access the variable (`y = 5;`) and nothing else. Be specific.

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

```
void foo( int x )
{
   static int y = x - 2;

   cout << y++ << endl;

   if ( x >= 4 && y <= 8 )
       foo( y - 2 );
   else
       cout << "Whoa!" << endl;

   cout << ++y << endl;
}

int main()
{
  foo( 6 );
  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?

```
#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;

   foo1( &argc, a );

/*  1 */ (void) printf( "1: foo1 --> %p\n", foo1 );
/*  2 */ (void) printf( "2: b --> %p\n", &b );
/*  3 */ (void) printf( "3: argc --> %p\n", &argc );
/*  4 */ (void) printf( "4: a --> %p\n", &a );
/*  5 */ (void) printf( "5: argv --> %p\n", &argv );
}

void foo1( int *c, int d ) {

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

   foo2( f, c );

/*  6 */ (void) printf( "6: g.a --> %p\n", &g.a );
/*  7 */ (void) printf( "7: d --> %p\n", &d );
/*  8 */ (void) printf( "8: e --> %p\n", &e );
/*  9 */ (void) printf( "9: f --> %p\n", &f );
/* 10 */ (void) printf( "10: malloc-1 --> %p\n",
                        c = (int *) malloc(50) );
/* 11 */ (void) printf( "11: g.b --> %p\n", &g.b );
/* 12 */ (void) printf( "12: c --> %p\n", &c );
}

void foo2( int h, int *i ) {

   int j[3];
   static int k = 411;

/* 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] );
/* 18 */ (void) printf( "18: malloc-2 --> %p\n",
                        i = (int *) malloc(150) );

}
```

_____ smallest value (lowest memory address)

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

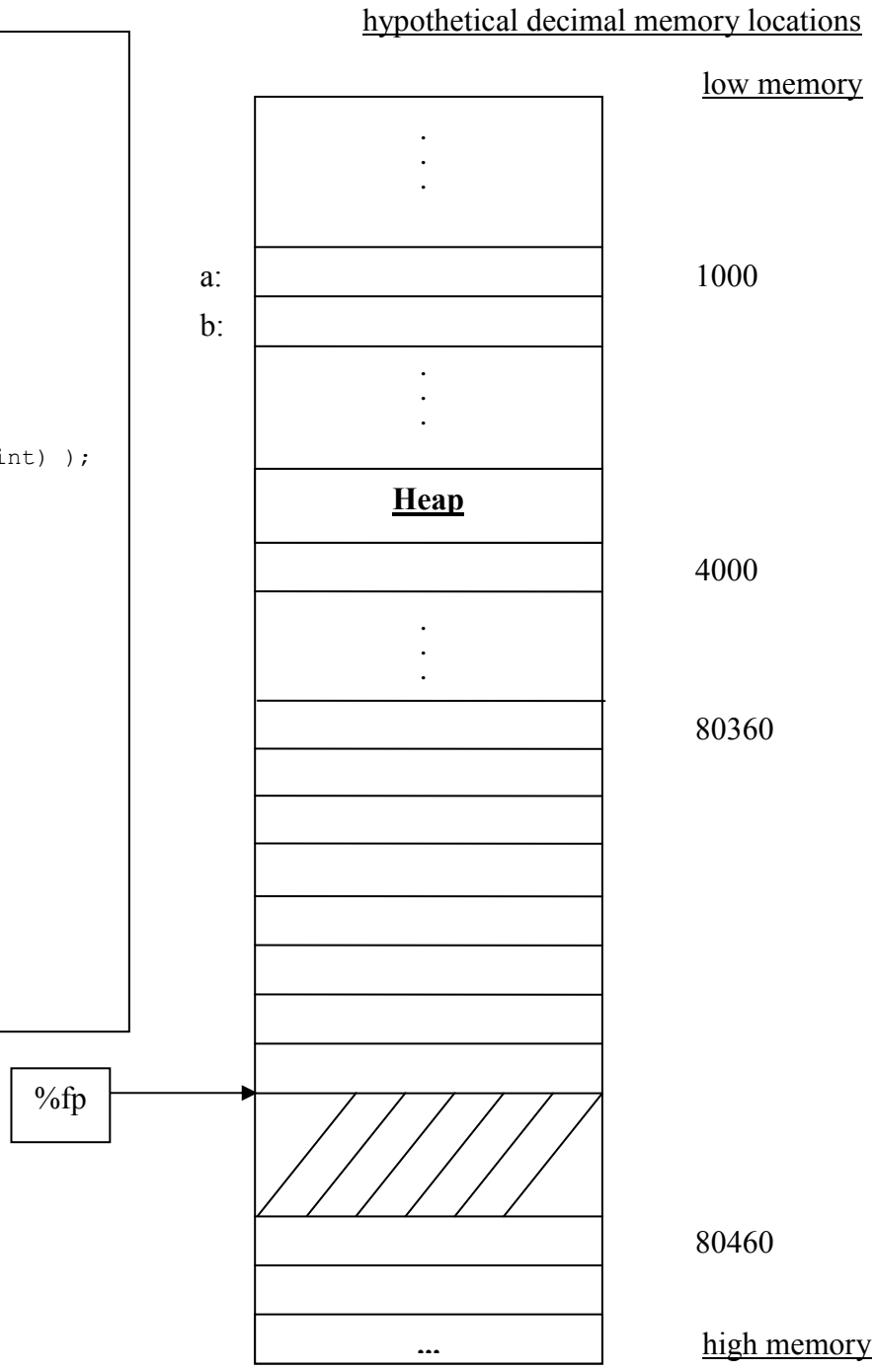
_____

_____ largest value (highest memory addresses)

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.

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

int   a;
float b = 1.25;

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

  var2 = 375;
  var3 = (int *) calloc( 1, sizeof(int) );
  f = -37.5;
  var1[1].c = i + 5;
  *var3 = var2 - 3;
  var1[0].a = &var1[1].c;
  i = 750;
  var1[0].c = a - 5;
  var1[1].a = &i;
  var1[0].b = b;
  var1[1].b = f + 3;

  /* HERE */

  free( var3 );
}

int main() {
  foo( b, a );

  return 0;
}
```



10. Given the following Reduced-C code, write the equivalent non-optimized SPARC assembly code that should be generated by a compliant compiler following this quarter's specs. Be sure to allocate all local variables and temporaries on the run time stack. Store and access all formal parameters on the run time stack.

```
            /* Reduced-C */

function : int foo( int & a, int x )
{
  int b;
  int c = 42;
  int * d;

  b = a;

  d = &x;

  return c - b;
}
```

To make it easiest for us to give you as many points possible, comment each group of instructions. For example:

```
! The next X lines translate the stmt
! b = a
assembly instr 1
assembly instr 2
    .
    .
    .
```

Make it very clear which group of assembly instructions you want us to grade for each part of the original source code. Draw lines between or rectangles around groups of assembly instructions that are logically associated with a task or instruction from the original source code.

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

11. Name the part of the compilation sequence which performs the following. Use the letters below associated with the compilation sequence parts.

A) Compiler B) Loader C) Preprocessor D) Linkage Editor E) Assembler

____ expands # directives & strips comments from its input high-level language (HLL)

____ translates assembly code into machine code

____ performs syntax analysis on its input high-level language (HLL)

____ takes an executable file on disk and makes it ready to execute in memory

____ translates HLL code (for example, C) into assembly code

____ combines all object modules into a single executable file

____ performs semantic analysis on its input high-level language (HLL)

____ resolves undefined external symbols with defined global symbols in other modules

Give an example of something (either in C/C++ or our Reduced-C) that is:

a) an r-value (neither addressable nor assignable)

b) a non-modifiable l-value (an object locator that is addressable but not assignable).

c) a modifiable l-value (an object locator that is addressable and assignable)

12. Extra Credit

What gets printed when this program is executed?

```
#include <stdio.h>

int
main()
{
  char a[] = "Foundations of Computer ";
  char b[] = "Science";
  char *p1 = a + 1;
  char *p2 = b;

  printf( "%c", *p2++ );    ______

  printf( "%c", p2[2] );    ______

  printf( "%c", *++p1 );    ______

  printf( "%c", *(p1 = p1 + 4) - 1 );    ______

  printf( "%c", p1[4] );    ______

  printf( "%c", *++p2 );    ______

  printf( "%c", *--p2 );    ______

  printf( "%c", *p2 - 2 );    ______

  printf( "%c", p1[-3] -2 );    ______

  return 0;
}
```

What is the output of the following Reduced-C program? Assume local variables are allocated in decreasing memory locations. So, for example, i is at a lower memory address than f.

```
function : void main()
{
  float f = 3.25;
  float f2 = (float)&f;
  int i = (int)&f2;

  *(float *)((int)&i + (int)f2 - i) = (*(float *)f2)++;
  ++*(int **)&i;
  i = ((int *)i)[-2] - (int)&i;

  cout << f << " " << f2 << " " << i;  ________ ________ ________
}
```

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 C Operator Precedence Table

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

Scratch Paper

Scratch Paper