Login Name ____________________ **Name** ________________________

Signature ______________________ **Student ID** ___________________

Final
CSE 131B
Spring 2006

Page 1	___________	**(22 points)**
Page 2	___________	**(36 points)**
Page 3	___________	**(31 points)**
Page 4	___________	**(31 points)**
Page 5	___________	**(32 points)**
Page 6	___________	**(20 points)**
Page 7	___________	**(19 points)**
Page 8	___________	**(22 points)**
Subtotal	___________	**(213 points)**
Page 9 Extra Credit	___________	**(12 points)**
Total	___________	

1. What keyword is used to modify a variable declaration to indicate to the compiler to not perform any code improvement transformations in expressions containing this variable? (2 points each)

What is the 80/20 rule?

Give an example of a strength reduction optimization the compiler can perform.

In Phase III.2 of our code gen project, we asked you to implement the following dynamic check:

```
Run-time array bounds checks. Out-of-bounds accesses should cause the program to print the
following message to stdout: "Index %D of array is outside legal range [0,%D).\n" and exit
(call exit(1)).
```

Your partner claims that he/she has implemented the above check completely. However, knowing that his/her previous partner divorced him/her, you decide to write your own test cases for this check. Assuming the following array definition:

```
VAR array : ARRAY 240 OF REAL;
BEGIN
   (* Your test code would go here. If the test passes, print "SUCCESS".*)
END.
```

Describe at least five test cases you would write to test thoroughly this dynamic array bounds check.

1)

2)

3)

4)

5)

C and C++ support a wide range of type casts. Given the following C/C++ pseudocode fragment:

```
Type1 x; /* Type1 is some typedef'ed type. */
Type2 y; /* Type2 is some typedef'ed type. */
```

What is the type of the following expressions?

`&x` ______________________________

`* (Type2 *) &x` ______________________________

`(Type2 *) &x` ______________________________

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. (30 pts)

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

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

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

}
public class Overloading_Final_Exam {
    public static void main (String [] args) {
        Peter meg = new Peter();
        Lois chris = new Lois();
        Peter stewie = new Lois();

        meg.print(meg);                           _____
        meg.print(chris);                         _____
        meg.print(stewie);                        _____

        chris.print(meg);                         _____
        chris.print(chris);                       _____
        chris.print(stewie);                      _____

        stewie.print(meg);                        _____
        stewie.print(chris);                      _____
        stewie.print(stewie);                     _____

        ((Peter)chris).print(meg);                _____
        ((Peter)chris).print(chris);              _____
        ((Peter)chris).print(stewie);             _____

        ((Lois)stewie).print(meg);                _____
        ((Lois)stewie).print(chris);              _____
        ((Lois)stewie).print(stewie);             _____
    }
}
```

In C++, static compile time binding is the default. What is the method modifier (keyword) that turns off static binding and turns on dynamic binding? (1 pt)

In Java, what are the three method modifiers (keywords) that turn off dynamic binding and turn on static compile time binding? (3 pts)

Why is static binding more efficient than dynamic binding? (2 pts)

3. In your Project 2, how did you (and your partner if you had a partner) implement the address-of operator? Be specific how your project implemented this! (5 points)

Given the following Oberon program 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 when the program reaches the label `(* HERE *)`. (26 pts)

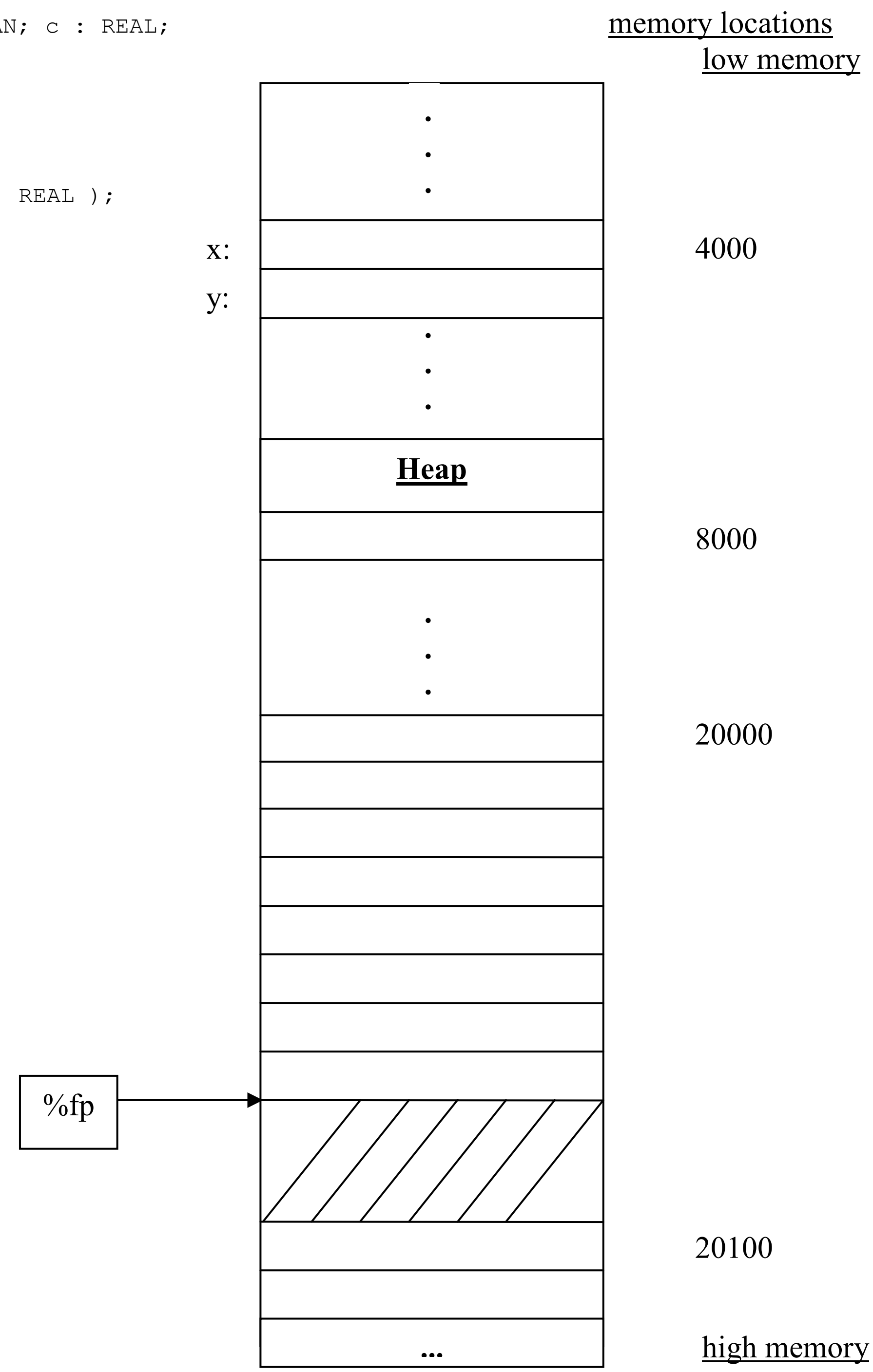
```
TYPE t = RECORD
           a : INTEGER; b : BOOLEAN; c : REAL;
         END;

VAR x : INTEGER;
VAR y : REAL;

PROCEDURE f( REF i : INTEGER; z : REAL );
    VAR  j : INTEGER;
    VAR a2 : ARRAY 2 OF t;
    VAR r2 : POINTER TO INTEGER;
BEGIN
    NEW(r2);
    z := 24.5;
    a2[0].a := -4;
    a2[1].a := 34;
    a2[1].c := 2.40;
    a2[0].b := FALSE;
    j := -73;
    a2[0].c := 32.5;
    a2[1].b := TRUE;
    r2^ := 529;
    i := 37;

    (* HERE *)
END;

BEGIN
    f( x, y );
END.
```



4. Given the following SPARC assembly code, write the equivalent Oberon code that would have generated this. There is one parameter named "a" and one local variable named "b". All types are INTEGER. You do not need to write the BEGIN END. of main(). (18 points)

```
/* SPARC Assembly */

    .section ".text"

foo:
    set   .fooSAVE, %g6
    save  %sp, %g6, %sp

    st    %g0, [%fp - 4]

    ld    [%i0], %l0
    ld    [%fp - 4], %l1
    cmp   %l0, %l1
    bge   .L1
    nop

    set   15, %l2
    st    %l2, [%i0]

    ba    .L2
    nop

.L1:
    set   20, %l2
    st    %l2, [%fp - 4]

.L2:
    ld    [%i0], %i0
    ret
    restore

    .fooSAVE = -(92 + 4) & -8
```

(* Equivalent Oberon code *)

Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named foo that is an array of 9 pointers to functions that take a pointer to char as a single parameter and return a pointer to a pointer to an array of 6 pointers to struct fubaz. (7 points)

In C, ______________________ equivalence is used for ______________________ while all other types use

________________________ equivalence. (6 points)

5. Given the array declaration

C	**Oberon-like**
`short b[4][3];`	`VAR b : ARRAY 4, 3 OF SHORTINT;`

Mark with a **B** the memory location(s) where we would find `b[1][2]`

and

Mark with a **C** the memory location(s) where we would find `b[2][1]`

`b:`

`low memory` `high memory`

Each box above represents a byte in memory.

Show the SPARC memory layout of the following struct/record definition taking into consideration the SPARC data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct/record member/field name. For example, if member/field name `p` takes 4 bytes, you will have 4 `p`'s in the appropriate memory locations. If the member/field is an array, use the name followed by the index number. For example, some number of `p0`'s, `p1`'s, `p2`'s, etc. Place an `X` in any bytes of padding. Structs and unions are padded so the total size is evenly divisible by the most strict alignment requirement of its members.

```
struct foo {
   char   a[6];
   double b;
   float  c;
   short  d;
   int    e[3];
}

struct foo fubar;
```

`fubar:` `low memory`

`high memory`

What is the `offsetof( struct foo, e[1] )`? _______

What is the `sizeof( struct foo )`? _______

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

If you rearranged the order of the struct members in `struct foo` to minimize padding, how many bytes of padding would you need? _______ And what would be the size of this modified struct? _______

6. Given the following function definitions and their already slightly optimized corresponding assembly code:

```
---------------------------------------------------------------------
int f(int x, int y) {           | f:      save %sp, -96, %sp
    int z;                      |         add %i0, %i1, %l0
    z = x + y;                  |         mov %l0, %i0
    return z;                   |         ret
}                               |         restore
---------------------------------------------------------------------
int main() {                    | ! START
    int x,y,w;                  | ! First call to f():
    scanf("%d %d", &x, &y);     |         mov %l0, %o0
                                |         mov %l1, %o1
    // START                    |         call f
    w = f(x,y) + f(y,x);        |         nop
    // END                      |         mov %o0, %l3
                                | ! Second call to f():
    printf("%d\n", x);          |         mov %l1, %o0
}                               |         mov %l0, %o1
                                |         call f
                                |         nop
                                | ! Addition on the results:
                                |         add %o0, %l3, %l2
                                | ! END
---------------------------------------------------------------------
```

Assume x is in %l0 and y is in %l1 at START, and the result w must be in %l2 at END.

A) How many assembly instructions are executed between START and END? _______

B) Can you fill the "nop" in the delay slot of the "call" instructions with the "mov" instruction immediately above the "call f" instruction (the mov instruction with %o1 as the destination register)? _______

Can you fill the "nop" in the delay slot of the "call" instructions with the "mov" instruction two instructions above the "call f" instruction (the mov instruction with %o0 as the destination register)? _______

Assuming you correctly filled the "nop"s to the "call" instructions, now how many assembly instructions are executed between START and END? _______

C) Rewrite f() as a leaf subroutine (no save or restore) and optimize that leaf subroutine implementation as much as you can.

```
f:
```

Using your rewrite of f() as a leaf subroutine, now how many assembly instructions are executed between START and END? _______

D) Further optimization can be performed by in-lining (open-coding) f() as an open subroutine. Rewrite the assembly code between START and END to inline f() and perform any additional optimizations you can.

```
! START



! END
```

Now how many assembly instructions are executed between START and END? _______

7. For the following Oberon code, generate the corresponding unoptimized assembly code. Also, take into account the "Dereference a NIL Pointer" error check before FREEing a pointer, as described in Project II. A framework of the assembly code is provided for your convenience. (19 points)

```
(* Oberon Code *)
TYPE recp = POINTER TO RECORD
                a: ARRAY 20 OF INTEGER; b: ARRAY 5 OF REAL;
            END;

VAR x : recp;

BEGIN
  NEW (x);
  (* ... *)
  FREE (x);
  RETURN 0;
END.
```

```
    .section ".bss"     /* Partial SPARC Assembly */

    _______________

x:  _______________

    .section ".text"
main:
    save    %sp, -96, %sp

    ! NEW (x)
    mov     25, %o0

    mov     _________, _________

    call    ___________
    nop

    set     x, ________                    ! map x into %l2

    st      _________, [_________]

           /* ... other code ... */

    ! FREE (x)
    set     x, ________                    ! map x into %l2

    ld     [_________], %o0

    cmp    %o0, _________

    be     PtrBAD
    nop

    call    ___________
    nop

    st      _________, [_________]

    mov     _________, _________

    ret
    restore

PtrBAD:

    set    errorMsg, %o0                   ! NIL ptr dereference msg
    call   printf
    nop

    mov     _________, _________

    call    ___________
    nop
```

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 system. For example, which line will print the lowest memory address, then the next higher memory address, etc. up to the highest memory address? (16 points)

```
#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;
   int b = 420;

   foo1( &argc, b );

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

void foo1( int *c, int d ) {

   int e = 404;
   static int f;
   int g;

/* 6 */ (void) printf( "6: f --> %p\n", &f );
/* 7 */ (void) printf( "7: g --> %p\n", &g );
/* 8 */ (void) printf( "8: c --> %p\n", &c );
/* 9 */ (void) printf( "9: malloc --> %p\n", malloc(50) );
/* 10 */ (void) printf( "10: d --> %p\n", &d );
/* 11 */ (void) printf( "11: e --> %p\n", &e );

   foo2( f, &d );

}

void foo2( int h, int *i ) {

   int j = 101;
   int k;
   static int l = 37;

/* 12 */ (void) printf( "12: k --> %p\n", &k );
/* 13 */ (void) printf( "13: l --> %p\n", &l );
/* 14 */ (void) printf( "14: h --> %p\n", &h );
/* 15 */ (void) printf( "15: i --> %p\n", &i );
/* 16 */ (void) printf( "16: j --> %p\n", &j );

}
```

_________ smallest value (lowest memory address)

____

____

____

____

____

____

____

____

____

____

____

____

____

____

_________ largest value (highest memory addresses)

Give an example of something (either in C/C++ or our Nano-Oberon) that is: (2 points each)

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

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

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

9. Extra Credit (12 points)

What is the value of each of the following expressions?

```
char *a = "End this, please!";            /*  char a[] = "End this, please!";  */

"I loved Compilers B!"[6]                 ________

a[1]                                      ________

*a                                        ________

toupper( a[strlen( a ) - 2] + 1 )         ________

*"I loved Compilers B!"                   ________

*a + 1                                    ________

*("This Blows Me Away!" + 14)             ________

*(&a[5] + 7)                              ________

*&a[5]                                    ________

0["This Blows Me Away!"]                  ________
```

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 programming/computer science career. (1 point if serious; you can add non-serious comments also)

Crossword Puzzle (next page) (1 point)

Scratch Paper

Scratch Paper