Login name _____ **Name _____**

Signature _____ **Student ID _____**

Final
CSE 131B
Winter 2006

Page 1	**_____ (27 points)**
Page 2	**_____ (25 points)**
Page 3	**_____ (32 points)**
Page 4	**_____ (24 points)**
Page 5	**_____ (38 points)**
Page 6	**_____ (16 points)**
Page 7	**_____ (26 points)**
Page 8	**_____ (14 points)**
Subtotal	**_____(202 points)**
Page 9 Extra Credit	**_____ (11 points)**
Total	**_____**

1. Give an example of an implicit type coercion (type conversion without an explicit cast). (3 points each)

Give an example of a converting type cast/conversion (underlying bit pattern needs to be changed).

Give an example of a non-converting type cast/conversion (underlying bit pattern does not change).

Give an example of a type inference rule the compiler will perform.

Give an example of a type equivalence error.

Give an example of an addressability error.

Give an example of an assignability error.

Regarding type checking, value parameters require the actual arguments to be _____ to the formal parameter type while reference (VAR) parameters require the actual arguments to be _____ and _____ to the formal parameter type. (2 points each)

2. Given the following Oberon program and expected output, determine whether each parameter is pass-by-reference or pass-by-value. Fill in the blanks with "VAR" if pass-by-reference, leave it blank if pass-by-value. (10 pts)

```
VAR x : INTEGER;
VAR y : INTEGER;

PROCEDURE foo1 ( ____ a : INTEGER);
BEGIN
    a = y + 20;
END foo1;

PROCEDURE foo2 ( ____ a : INTEGER, ____ b : INTEGER);
BEGIN
    a = 3 * y;
    b = 2 * x;
END foo2;

PROCEDURE foo3 ( ____ a : INTEGER, ____ b : INTEGER);
BEGIN
    a = x - y;
    b = x + y;
END foo3;

BEGIN
    x = 10;
    y = 10;

    foo1(y);
    foo2(x, y);
    foo3(x, y);

    OUTPUT x, " ", y;     (* should output -10 10 *)
END.
```

Now in order to get full credit for the above and discourage Jeff Spicoli-like random guesses, what is the output if each VAR parameter was changed to non-VAR and each non-VAR parameter was changed to VAR? (10 pts)

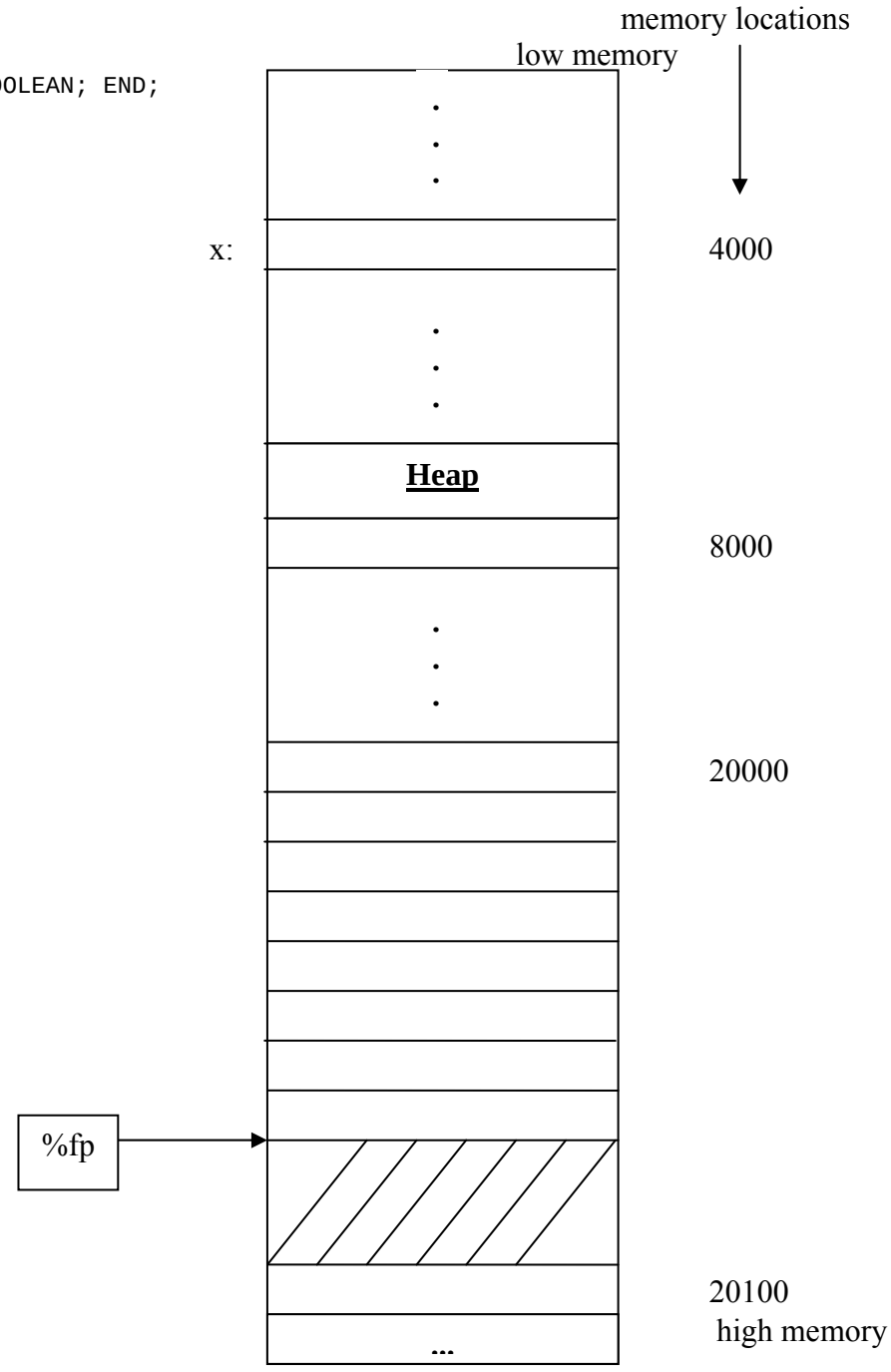
Give an example of a semantic type error in C/C++ that the compiler will not be able to detect due to separate compilation. (3 pts)

Since the compiler cannot detect this type of semantic error, where in the entire compilation sequence could this error be detected? WHY? (2 pts)

3. In your Project 2, how did you (and your partner if you had a partner) implement while loops? There were several possible implementation options outlined in one of the SPARC Supplement handouts. Be specific how your project implemented them! (10 points)

Given the following Oberon program and real compiler code gen, fill in the values of the global and local variables and parameter in the run time environment when the program reaches the label HERE.

```
TYPE t = RECORD a: INTEGER; b: BOOLEAN; END;
VAR x : INTEGER;

PROCEDURE f(VAR i: INTEGER);
    VAR a2: ARRAY 3 OF t;
    VAR r2: POINTER TO INTEGER;
    VAR j: INTEGER;
BEGIN
    NEW(r2);
    r2^ := 420;
    a2[0].a := 7;
    a2[1].a := 18;
    a2[2].b := FALSE;
    a2[0].b := TRUE;
    j := 12;
    a2[2].a := -19;
    a2[1].b := TRUE;

    (* HERE *)
END;

BEGIN
    f(x);
END.
```



4. Describe the main tasks/functions for each part of the C compilation → program execution process:

C Preprocessor ...

1)

2)

C Compiler ...

1) Front end:

2) Back end:

Assembler ...

1)

Linkage Editor ...

1)

2)

Loader ...

1)

2)

Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named foo that is an array of 5 pointers to functions that take a pointer to double as a single parameter and return a pointer to an array of 8 pointers to struct Fubar. (6 points)

5. Given the following Oberon program, emit the **unoptimized** SPARC assembly code that should be generated for the two procedures foo() and main(). Assume no optimizations – treat each instruction separately without any knowledge of any previously computed/loaded/stored values that may still be in a register from a previous instruction. Draw a line between each group of assembly language instructions that represent the emitted code generated for each instruction and label them with the instruction number. All local variables must be allocated and accessed on the Stack (do not map them directly into local registers). Allocate all global variables in the BSS and access them via their own label/name [like real compilers do]. Formal parameters may be accessed via the registers in which they are passed. (38 pts)

```
VAR a, b, c : INTEGER;

PROCEDURE foo(VAR x : INTEGER; y : INTEGER) : INTEGER;
  VAR z : INTEGER;
BEGIN
  z := x - 10;            (* 1 *)
  x := x + y;             (* 2 *)
  y := y * y;             (* 3 *)
  RETURN z;               (* 4 *)
END foo;

BEGIN
  INPUT a;                (* 5 *)
  b := a + 2;             (* 6 *)
  c := foo(a, b);         (* 7 *)
  OUTPUT c;               (* 8 *)
END.
```

```
! allocate global variables here




        .section ".rodata"
intFmt: .asciz "%d"
```

```
        .global foo, main, a, b, c
        .section ".text"
foo:
```

```
main:
```

6. A loop in the C family of languages (and others) may contain a break or continue (really just a glorified goto). Specify the location of the target label of a break branch and the target label of a continue branch in the SPARC assembly code fragments below by writing the name of the label in the appropriate location.

i = 0;
while (i <= j) { ... if (a == 5) break; ... ++i; }

```
loop:
            ! loop body


            cmp   %l0, 5
            be    breakLabel
            nop


            ! more loop body


            cmp   %l3, %l5
            ble   loop
            nop
```

i = 0;
while (i <= j) { ... if (a == 5) continue; ... ++i; }

```
loop:
            ! loop body


            cmp   %l0, 5
            be    continueLabel
            nop


            ! more loop body


            cmp   %l3, %l5
            ble   loop
            nop
```

`for` loops are equivalent to `while` loops except when one of the statements in the loop body is a continue statement. Specify the location of the target label of a continue branch in both a for loop and a while loop in the SPARC assembly code fragments below by writing the name of the label in the appropriate location.

for (i = 0; i <= j; ++i) { ... if (a == 5) continue; ... }

```
forloop:
            ! loop body


            cmp   %l0, 5
            be    continueLabel
            nop


            ! more loop body

            inc   %l3

            cmp   %l3, %l5
            ble   forloop
            nop
```

i = 0;
while (i <= j) { ... if (a == 5) continue; ... ++i; }

```
whileloop:
            ! loop body


            cmp   %l0, 5
            be    continueLabel
            nop


            ! more loop body

            inc   %l3

            cmp   %l3, %l5
            ble   whileloop
            nop
```

What is the Unix command used in class to list symbols (name list) from an object file?

What are the two Unix commands on Sun Solaris used in class (1 used in the linker paper) to dissassemble an object file?

What is the Unix command used in class to print section sizes of an object file?

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. (26 points)

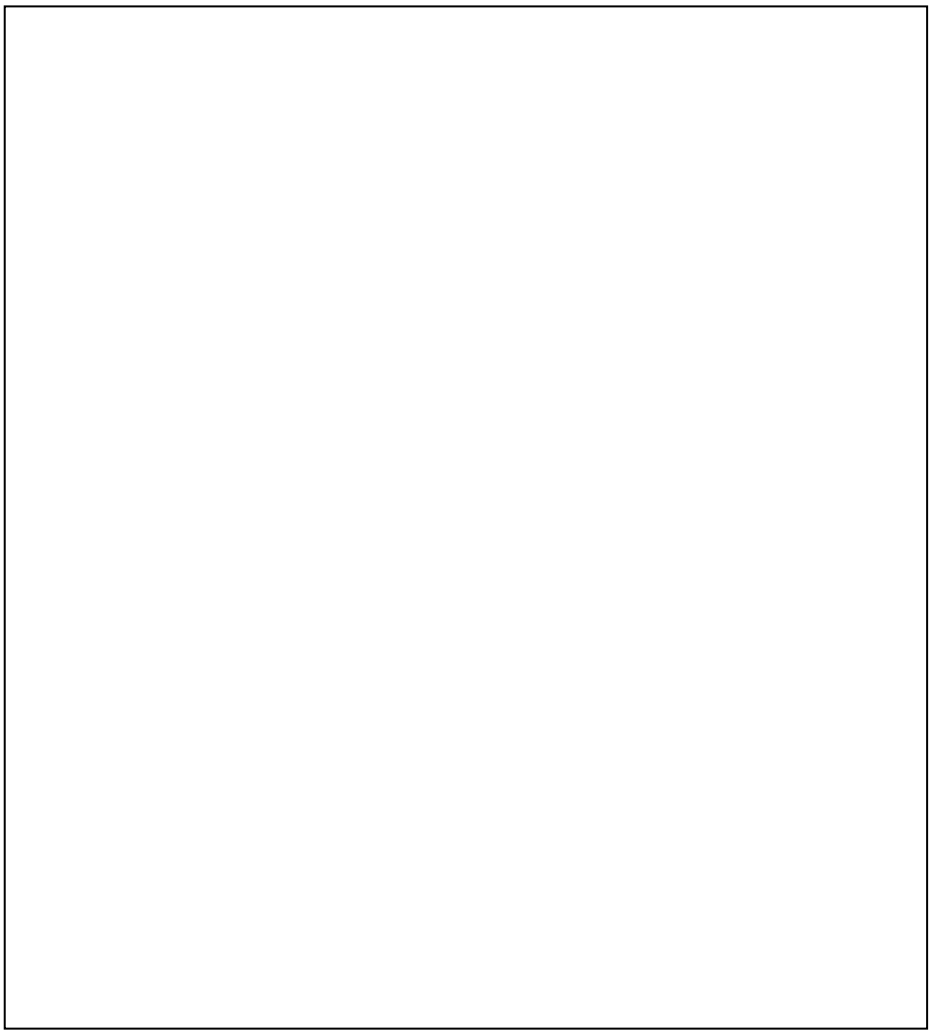
```
(* Oberon Code *)
TYPE recp = POINTER TO RECORD
               a: ARRAY 5, 5 OF INTEGER;
            END;

VAR x : recp;

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

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

    ________________

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, _________

    bne     PtrOK
    nop

    set     errorMsg, %o0
    call    puts              ! similar to printf(string) but safer
    nop                       ! avoids format string exploit

    call    ___________
    nop

PtrOK:
    call    ___________
    nop

    ret
    restore
```

8. When is a caller-save register convention more efficient than a callee-save register convention? (2 pts)

Assume the compiler generated the following SPARC assembly code. Rewrite it to improve the run time speed of this code. Do not change the overall algorithm; just perform basic optimization transformations. (10 pts)

```
        .global main

        .section ".rodata"

code:   .asciz  "102"
fmt:    .asciz  "%s = %d\n"

        .section ".text"
main:
        save    %sp, -(92 + 4) & -8, %sp       ! No changes at or above the save instruction

        set     code, %l0
        ldub    [%l0], %l1
        mov     %g0, %l2
        st      %l2, [%fp - 4]

        cmp     %l1, %g0
        be      .L1
        nop

.L2:
        ld      [%fp - 4], %l2
        mov     %l2, %o0
        mov     8, %o1
        call    .mul
        nop

        ldub    [%l0], %l1
        sub     %l1, 0x30, %l1
        add     %o0, %l1, %l2
        st      %l2, [%fp - 4]

        inc     %l0
        ldub    [%l0], %l1

        cmp     %l1, %g0
        bne     .L2
        nop

.L1:
        set     fmt, %o0
        set     code, %o1
        ld      [%fp - 4], %o2
        call    printf
        nop

        ret
        restore
```



Tell me something you learned in this class that is extremely valuable and that you think you will be able to use for the rest of your programming/computer science career. (2 points if serious; you can add non-serious comments also)

9. Extra Credit (11 points)

What gets printed by the following C program?

```
#include <stdio.h>

int
main()
{
  char a[] = "Me? I want to go";
  char b[] = "to Porter's Pub";
  char c[] = "and don't you, too?";
  char *ptr = a;

  printf( "%c\n", *(ptr + 4) );                              _______

  printf( "%c\n", *(ptr = b + 3) );                          _______

  printf( "%c\n", toupper( *(b + strlen(b) - 1) ) - 1 );     _______

  printf( "%c\n", c[strlen(b)-5] );                          _______

  printf( "%c\n", ptr[10] );                                 _______

  printf( "%c\n", tolower(*a) );                             _______
  return 0;
}
```

Given the following ANSI/ISO C variable definitions, identify which expressions will produce a static semantic compiler error. Hint: Think modifiable l-value.

A) No compiler error
B) Compiler error

```
int i = 5;
float f = 1.5;
int *iPtr = &i;
float *fPtr = &f;

++( (float *) iPtr );   ______

i = *(int *)fPtr;       ______

fPtr = &(i + f);        ______

iPtr = *(int **)&fPtr;  ______
```

Crossword Puzzle (next page)

Scratch Paper

Scratch Paper