**Signature** ____________________ **Name** ________________________

**Login Name** __________________ **Student ID** ___________________

# Midterm
# CSE 131B
# Spring 2006

| | | |
|---|---|---|
| **Page 1** | ___________ | **(19 points)** |
| **Page 2** | ___________ | **(26 points)** |
| **Page 3** | ___________ | **(20 points)** |
| **Page 4** | ___________ | **(20 points)** |
| **Page 5** | ___________ | **(15 points)** |
| **Subtotal** | ___________ | **(100 points)** |
| **Page 6**<br>**Extra Credit** | ___________ | **(5 points)** |
| **Total** | ___________ | |

**1.** Using the following type and variable declarations, indicate the size of each of the variables (in bytes) in the space provided. In the event that a type definition for a variable is an error/indeterminate size, write the word "ERROR" in the blank. (2 pts each)

```
TYPE rec1 = RECORD
       a : INTEGER;
       b : REAL;
       c : ARRAY 10 OF POINTER TO rec1;
       d : POINTER TO rec1;
       e : POINTER TO REAL;
     END;

TYPE rec2 = POINTER TO RECORD
       a : ARRAY 900, 900 OF REAL;
       b : RECORD
         a : REAL;
         b : INTEGER;
         c : ARRAY 20 OF rec2;
       END;
     END;

TYPE rec3 = RECORD
       a : INTEGER;
       b : REAL;
       c : RECORD
         d: rec3;
       END;
       e : POINTER TO rec3;
     END;

TYPE foo1 = ARRAY 10, 10 OF rec2;

TYPE foo2 = RECORD
       a : foo1;
       b : rec1;
       c,d : rec2;
     END;

VAR myRec1 : rec1;      _________

VAR myRec2 : rec2;      _________

VAR myRec3 : rec3;      _________

VAR myFoo1 : foo1;      _________

VAR myFoo2 : foo2;      _________
```

Using the Right-Left rule write the C/C++ definition of a variable named fubar that is a pointer to a function which takes two arguments, a pointer to a char and a pointer to a pointer to a float, and returns a pointer to an array of 4 elements where each array element is of type pointer to struct foobaz. (9 pts)

**2.** The types in Oberon variable definitions are often unnecessary in the sense that it is possible to infer variables' types and detect type errors simply from their use. For each of the following program fragments, find a set of types that makes it legal, and write an Oberon definition for each variable. If there is more than one possible type, choose only one. If there is none, write "NONE". Assume all arrays are of size 8. (2 pts each)

```
    a[b^] := b;

VAR a : _____________________________________________ ;

VAR b : _____________________________________________ ;
```

---

```
    IF a[b] # c& THEN
        b := c + d;
    END

VAR a : _____________________________________________ ;

VAR b : _____________________________________________ ;

VAR c : _____________________________________________ ;

VAR d : _____________________________________________ ;
```

---

```
    IF (a # b) OR c THEN
        b := c;
    END

VAR a : _____________________________________________ ;

VAR b : _____________________________________________ ;

VAR c : _____________________________________________ ;
```

---

```
    IF a[b]& = c THEN
        c^ := b / d;
    END

VAR a : _____________________________________________ ;

VAR b : _____________________________________________ ;

VAR c : _____________________________________________ ;

VAR d : _____________________________________________ ;
```

**3.** Given the following type, variable, and function definitions, state whether each function call would match
(A) the first
(B) the second
(C) the third
(D) none (illegal call)
(E) more than one of the definitions (ambiguous call) ( 2 pts each)

```
TYPE peter = REAL;
TYPE lois = peter;

VAR chris : lois;
VAR meg : INTEGER;


FUNCTION quagmire (brian : peter, stewie : INTEGER) : INTEGER;       (* first *)
BEGIN
   RETURN 0;
END quagmire;

FUNCTION quagmire (REF brian: INTEGER, stewie : peter) : INTEGER;    (* second *)
BEGIN
   RETURN 0;
END quagmire;

FUNCTION quagmire (brian : peter, REF stewie : lois) : INTEGER;      (* third *)
BEGIN
   RETURN 0;
END quagmire;

BEGIN
   quagmire(chris, meg);                    _____

   quagmire(meg, chris);                    _____

   quagmire(meg + meg, chris + chris);      _____

   quagmire(meg, meg);                      _____

   quagmire(meg, chris + meg);              _____

   quagmire(meg + meg, chris);              _____

   quagmire(chris + meg, chris + meg);      _____

   quagmire(chris, chris);                  _____

   RETURN 0;
END.
```

Although we did not allow it in this quarter's project specification, record/struct assignment is relatively straight forward requiring same/equal types for both operands to the binary assignment operator ( := ). It certainly is defined in the C family of languages. So ... if assignment is defined for records/structs in languages like C, why is record equality ( equals / not equals ) not defined? From the compiler's point of view, why is record/struct equality difficult? (4 pts)

**4.** Given the following array definition

```
       /* C */                                   (* Oberon *)
    double x[50][20];                          VAR x : ARRAY 50,20 OF LONGREAL;
```

write the assembly level address calculation expression <u>taking into account scalar arithmetic</u> to access

```
    x[i][j]                                        x[i,j]
```

(( x + ________________________________________ ) + ______________________________________ )

The result is the address of where we can find this array element. (8 points)

Assume the array access was in a nested loop construct to print each array element in order like the following:

```
for ( i = 0; i < 50; ++i )
  for ( j = 0; j < 20; ++j )
    print( x[i][j] );
```

Why would using a traversal pointer be more efficient? (2 pts)

Replace the above nested loop code by writing the definition for a traversal pointer (named ptr), the statement to assign/initialize the traversal pointer, and the loop code to print each element in the array above using this traversal pointer. If you prefer to use pseudo-Oberon syntax instead of C syntax, you can assume pointer arithmetic is defined the same as in C. (10 pts)

**5.** Assume the following definitions are correct: (2 pts each)

```
TYPE rec1 = RECORD
       ptr : POINTER TO INTEGER;
     END;

TYPE rec2 = RECORD
       ptr : POINTER TO rec1;
     END;

VAR ptr : POINTER TO rec2;
VAR a, b : INTEGER;
```

a) What type is `ptr^.ptr^.ptr` ?

b) What type is `ptr^` ?

c) What type is `ptr^.ptr^.ptr^` ?

d) What type is `ptr^.ptr` ?

For the following, assume each statement is independent of each other. You cannot use the result of a previous statement in the answer for a subsequent question. You can only use the above vars and types. You cannot use NIL.

e) Write the Oberon statement to have `ptr` in `rec1` point to variable **b**.

f) Write the Oberon statement to assign variable **a** the value of the integer pointed to by `ptr` in `rec1`.

```
      a :=
```

g) Write a valid statement with variable **c** (defined below) on the lhs (left hand side) of the assignment statement. You cannot use `c` or NIL.

```
VAR c : POINTER TO rec1;

      c :=
```

Do you promise to have a good time tomorrow at Sun God Festival ... or wherever you may be tomorrow ... oh, even at Richard's Discussion Section before hitting SGF hard? (1 pt)

**Extra Credit** (5 points)

Given the following definitions:

```
      int i1 = 5;              /* Assume i1 is at memory location 4000 */
      int i2 = 10;             /* Assume i2 is at memory location 5000 */
      float f = 1.5;           /* Assume f is at memory location 6000 */
      int *i1Ptr = &i1;        /* Assume i1Ptr is at memory location 7000 */
      int *i2Ptr = &i2;        /* Assume i2Ptr is at memory location 8000 */
      float *fPtr = &f;        /* Assume fPtr is at memory location 9000 */
```

```
i1Ptr = (int *) fPtr;

What is the value of i1Ptr after this statement executes?   ________


++fPtr;

What is the value of fPtr after this statement executes?    ________


i1Ptr = i2Ptr;

What is the value of i1Ptr after this statement executes?   ________


i1 = *&*i1Ptr;

What is the value of i1 after this statement executes?      ________


i2 = **&i2Ptr + 1;

What is the value of i2 after this statement executes?      ________
```

**Assume each instruction is executed in the order given.**

## Scratch Paper