Signature ____________________ **Name** ________________________

Login Name __________________ **Student ID** ___________________

Midterm
CSE 131
Winter 2010

Page 1	___________	**(21 points)**
Page 2	___________	**(27 points)**
Page 3	___________	**(32 points)**
Page 4	___________	**(9 points)**
Page 5	___________	**(20 points)**
Page 6	___________	**(16 points)**
Subtotal	___________	**(125 points = 100%)**
Page 7 Extra Credit	___________	**(7 points)**
Total	___________	

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

```
Expr ::=   Des AssignOp Expr {: System.out.println("00"); :}
     |     Des {: System.out.println("0"); :}
     ;

Des  ::=   T_STAR {: System.out.println("1"); :} Des {: System.out.println("2"); :}
     |     T_PLUSPLUS {: System.out.println("3"); :} Des {: System.out.println("4"); :}
     |     T_AMPERSAND {: System.out.println("5"); :} Des {: System.out.println("6"); :}
     |     Des2 {: System.out.println("7"); :}
     ;

Des2 ::=   Des2 {: System.out.println("8"); :} T_PLUSPLUS {: System.out.println("9"); :}
     |     Des3 {: System.out.println("10"); :}
     ;

Des3 ::=   T_ID {: System.out.println("11"); :}
     ;

AssignOp ::=  T_ASSIGN {: System.out.println("12"); :}
     ;
```

What is the output when parsing the follow expression (you should have 18 lines/numbers in your output):

```
x = *y = z++
```

<u>Output</u>

In the above grammar, does the assignment operator have left-to-right associativity or right-to-left associativity? ______________________

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

______________________ `y;` ______________________ `x;`

2. Give the order of the typical C compilation stages and on to actual execution as discussed in class

0 – Loader	6 – ccomp (C compiler)
1 – Program Execution	7 – ld (Linkage Editor)
2 – as (Assember)	8 – Source file (prog.c)
3 – Object file (prog.o)	9 – Assembly file (prog.s)
4 – prog.exe/a.out (Executable image)	10 – cpp (C preprocessor)
5 – Segmentation Fault (Core Dump) / General Protection Fault	

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

Given the following C++ definitions (similar to Reduced-C)

```
struct S1 { int a; };
struct S2 { int a; };

void foo ( struct S2 &b ) { }

struct S1 a;
```

a call to `foo( a )` passing in `a` as the actual argument will cause a compile error. Why?

Fix the function call `foo( a )` below to pass `a` to `foo()` without causing a C++ compile error.

```
foo( ___________________________________________ a );
```

Using Reduced-C syntax, define an array of an array of floats with dimensions 3x9 named bar such that `bar[2][8] = 42.24;` is a valid expression. This will take two lines of code.

Modifiable L-vals, Non-Modifiable L-vals, R-vals

Using the Reduced-C Spec (which closely follows the real C language standard), given the definitions below, indicate whether each expression evaluates to either a

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

```
function : int * foo() { /* Function body not important. */ }
structdef R1 { int a; float b; };
float[9] a;
R1       b;
R1 *     c;
int *    d;
```

____ `++b.a` ____ `c+1` ____ `&b` ____ `(int)a[3]` ____ `c->a % b.a`

____ `foo()` ____ `&a[2]` ____ `(R1 *)foo()` ____ `a[1] = *foo()` ____ `++*d++`

3. Given the following C++ definitions (similar to Reduced-C):

```
void foo1( int a )    { ... }
void foo2( int & a ) { ... }
int  foo3()          { ... }

int x;
float y;
int *ptr;
```

For each of the following function calls, indicate the type of error (if any) that should be reported (using the Project I spec for this quarter which is similar to the C++ rules). Use the letters associated with the available errors from the box to the right.

A) Argument not equivalent to reference param
B) Argument not assignable to value param
C) Arg passed to reference param is not a modifiable L-val
D) No Error

```
foo1( ptr );            ____
foo1( *ptr );           ____
foo1( *ptr++ );         ____
foo1( *++ptr );         ____
foo1( ++*ptr );         ____
foo1( ++*ptr++ );       ____
foo1( *&x );            ____
foo1( *&y );            ____
foo1( (int)&*ptr );     ____
foo1( *&ptr );          ____
foo1( 42 );             ____
```

```
foo2( ptr );            ____
foo2( *ptr );           ____
foo2( *ptr++ );         ____
foo2( *++ptr );         ____
foo2( ++*ptr );         ____
foo2( ++*ptr++ );       ____
foo2( *&x );            ____
foo2( *&y );            ____
foo2( (int)&*ptr );     ____
foo2( *&ptr );          ____
foo2( 42 );             ____
foo2( *(int *)&y );     ____
foo2( foo3() );         ____
```

Using the Right-Left rule write the C definition of a variable named fubaz that is a pointer to a 2-d array of 19 rows by 4 columns where each element is 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 8 elements where each element is a pointer to a struct fubar.

4. Consider the following struct definitions in Reduced-C (similar to C/C++). Specify the size of each struct on a typical RISC architecture (like ieng9) or 0 if it is an illegal definition.

```
structdef FOO1 {
  int   a;
  float b;
  function : void bar()
  {
    FOO1 x;
  }
  FOO1 *c;
  int   d[2];
};
```

Size _______

```
structdef FOO2 {
  int a;
  float b;
  function : void bar()
  {
    FOO2 *x;
  }
  FOO2  c[2];
  int  *d;
};
```

Size _______

```
structdef FOO3 {
  FOO3 *a;
  float b;
  function : void bar( FOO3 &x)
  {
    x.d[0] = *x.c;
  }
  int  *c;
  int   d[2];
};
```

Size _______

Fill in the blanks of the following Reduced-C program with correct types to test if your Phase 0 fix to the scoping bug present in the starterCode works correctly. If the scoping bug is fixed, this program should compile without error. If the bug is not fixed, this program should generate an assignment error at the line x = y;

```
_______  x;      // global x

function : int main() {

  _______  x;    // local x

  bool y;

  x = y;         // If fixed in Phase 0, this line will not cause an error!
                 // If not fixed in Phase 0, this line will cause an error!
  return 0;
}
```

Describe briefly what you/your group did to fix this scoping bug in the starter code.

Given the following Reduced-C code below, fill in the blanks of the compile error that should be reported according to this quarter's Project I spec. Use the letters associated with the words in the box below.

```
typedef float F1;
typedef F1 F2;
typedef int I1;
typedef I1 I2;

I1 x;
I2 y;
F2 z;
```

A. F1	B. F2
C. float	D. I1
E. I2	F. int
G. 5-hour Energy	H. Sleep
I. equivalent	J. assignable
K. addressable	L. modifiable

```
x = z = y; // Compile error reported here. Assume this stmt is inside a function.
```

Value of type ____ not ____ to variable of type ____ .

5. Show the memory layout of the following C struct definition taking into consideration the **SPARC** data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct 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 `p[0]`s, `p[1]`s, `p[2]`s, etc. If the member/field is a struct, use the member name followed by its member names (e.g. p.a, p.b). 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 {
  short  a;
  char   b[4];
  double c;
  short  d;
};

struct fubar {
  char       e[5];
  int *      f;
  struct foo g;
  char       h[3];
  char       i;
};

struct fubar fubaz;
```

low memory
fubaz:

high memory

What is the `sizeof( struct fubar )`? _____ What is the `offsetof( struct fubar, g.b[1] )`? _____

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

What is the resulting type of the following expression?

```
* (char *) (& ( ((struct foo *) fubaz.e )->d ) + 2)
```
____

Write the equivalent expression that directly accesses this value/memory location without all the fancy casting and `&` operators.

`fubaz.`________________

6. Given the following C program:

```
#define X 6
#define Y 4

int   a[X][Y];
int * b[X];

int main()
{
  int i;

  for ( i = 0; i < X; i++ )
    b[i] = malloc( sizeof(int) * Y );

  return 0;
}
```

Match the following expressions with the corresponding type (think type equivalence) from the list A-P.
Use type equivalence rules, not assignability.

```
*a          _____
*b          _____
**a         _____
&b[1][2]    _____
a + 2       _____
&a          _____
&b          _____
b           _____
```

```
A. int
B. int *
C. int[4]
D. int[6]
E. int[6][4]
F. int[4][6]
G. int (*)[6]
H. int (*)[4]
I. int (*)[6][4]
J. int (*)[4][6]
K. int* [6]
L. int* [4]
M. int* [6][4]
N. int* [4][6]
O. int* (*)[6]
P. int* (*)[4]
```

Fill in the blanks to make the array expression below equivalent to the following pointer expression. Note: You cannot use negative numbers in the array expression!

```
*(*(a + 2) - 3) is equivalent to a[___][___]
```

We can access the underlying data associated with `a` and `b` (as defined in the program above) using the same array or pointer expressions. However their underlying structure is different from each other.

What is the total number of bytes allocated to the entire data structure for `a`? ______
What is the total number of bytes allocated to the entire data structure for `b` including any memory dynamically allocated and associated with and reachable by `b`? ______

Assume we want to add a traversal pointer to more efficiently traverse the array `a` above. How would you define and initialize this traversal pointer?

```
____________________ ptr = _______________________ ;
```

Using this traversal pointer you just defined above, write a pointer expression (with no array brackets []) to access the last array element in `a` (last row, last column).

____

Extra Credit

What gets printed when the following C program is executed?

```
#include <stdio.h>

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

  printf( "%c\n", *(ptr = ptr + 9) + 1 );          _____

  printf( "%c\n", *c + 1 );                        _____

  printf( "%c\n", *(a + 1) );                      _____

  printf( "%c\n", *(&b[1] - 1) );                  _____

  printf( "%c\n", *++ptr + 2);                     _____

  printf( "%c\n", ptr[sizeof(ptr) + 2] - 1 );      _____

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

  return 0;
}
```

```
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