Midterm
CSE 131
Spring 2010

Page 1  __________ (21 points)
Page 2  __________ (28 points)
Page 3  __________ (32 points)
Page 4  __________ (18 points)
Page 5  __________ (19 points)
Page 6  __________ (8 points)

Subtotal  __________(126 points = 100%)

Page 7  __________ (9 points)
Extra Credit

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

\[
\text{Expr ::= } \text{Des AssignOp Expr \{ } \text{System.out.println("00"); } \}\text{;}
\| \text{Des \{ } \text{System.out.println("0"); } \}\text{;}
\]

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

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

\[
\text{Des3 ::= } \text{T_ID \{ } \text{System.out.println("11"); } \}\text{;}
\]

\[
\text{AssignOp ::= } \text{T_ASSIGN \{ } \text{System.out.println("12"); } \}\text{;}
\]

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

\[
*x = y = &z
\]

In the above grammar, does the postincrement 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 phases of compilation in a typical C compiler as discussed in class

A – Parser (Syntax Analysis)     E – Target language file (for ex., prog.s)
B – Source language file (for example, prog.c)  F – Intermediate Representation(s)
C – Scanner (Lexical Analysis)    G – Parser (Semantic Analysis)
D – Code generation (for ex., Assembly)


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

```cpp
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.

```cpp
foo( ______________________________________________________ a );
```

Using Reduced-C syntax, define an array of an array of bools with dimensions 8x4 named bar such that `bar[7][2] = true;` 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) Modifiable L-val  B) Non-Modifiable L-val  C) R-val

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

___ *d++    ____ &b    ____ (&b)->b    ____ (int)a[3]    ____ a
___ foo    ____ &a[2]    ____ foo()    ____ *foo()    ____ ++*d
___ *++d    ____ (*d)++    ____ d++    ____ b.b    ____ b = *c
```
3. Given the following C++ definitions (similar to Reduced-C):

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

```cpp
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.

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo2( ++*ptr++ );</td>
<td>D</td>
</tr>
<tr>
<td>foo2( *&amp;x );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( *&amp;y );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( (int)&amp;*ptr );</td>
<td>B</td>
</tr>
<tr>
<td>foo2( *&amp;ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( 42 );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( *(int *)&amp;y );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( foo3() );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( *ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( *ptr++ );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( *++ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( ++*ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( ++*ptr++ );</td>
<td>A</td>
</tr>
<tr>
<td>foo2( *&amp;x );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( *&amp;y );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( (int)&amp;*ptr );</td>
<td>B</td>
</tr>
<tr>
<td>foo1( *&amp;ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( 42 );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( *(int *)&amp;y );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( foo3() );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( *ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( *ptr++ );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( *++ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( ++*ptr );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( ++*ptr++ );</td>
<td>A</td>
</tr>
<tr>
<td>foo1( *&amp;x );</td>
<td>A</td>
</tr>
</tbody>
</table>

Using the Right-Left rule write the C definition of a variable named fubaz that is a pointer to a 2-d array of 5 rows by 11 columns where each element is a pointer to a function that takes a pointer to a double as a single parameter and returns a pointer to an array of 20 elements where each element is a pointer to a struct fubaz.
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[2] d;
};

structdef FOO2 {
    int   a;
    float b;
    function : void bar() {
        int x = *this.d;
    }
    FOO2 *c;
    int  *d;
};

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

<table>
<thead>
<tr>
<th>Size</th>
<th>Size</th>
<th>Size</th>
</tr>
</thead>
</table>

State whether constant folding can be performed by the compiler according to this quarter's Reduced-C spec in the following Reduced-C statements (T or F)

```
function : void foo() {
    const int a = 5;
    int b = 3;
    const int c = a + 10;
    int[53 + c] d;
    d[-2 + (a * b)] = c;
    b = d[d[2] + c];
    int e = d[a + c];
    e = d[13 + b];
    e = d[e + a];
    d[5 - 2 + c] = e;
}
```

How did your group test the sizeof operator to make sure it was working correctly?

Using only the following C variable declarations:

```
int a;
float b;
int  *c;
```

Give an example assignment stmt using a non-converting type cast (underlying bit pattern does not change).

Give an example assignment stmt using a converting type cast (underlying bit pattern changes).
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. Underline this sentence for a point. Structs and unions are padded so the total size is evenly divisible by the most strict alignment requirement of its members.

```c
struct foo {
    char a;
    short b[2];
    double c;
    int d;
};
struct fubar {
    int e[2];
    short f[2];
    char g[11];
    struct foo h;
    short i[2];
};
struct fubar fubaz;
```

What is the `sizeof( struct fubar )`? _____  What is the `offsetof( struct fubar, h.d )`? _____

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?

```
( (struct fubar *) ((& fubaz.h.c) + 1) ) -> f[1] ____________
```

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

```
fubaz.____________
```
6. Given the following Reduced-C structdef:

```c
structdef MYSTRUCT {
    int x;

    function : int foo()
    {
        float x;
        /* Body of foo() */
    }

    function : void bar()
    {
        /* Body of bar() */
    }
};
```

Write a simple assignment statement that could occur in the body of foo() using both of the variables named x such that an error will not occur according to this quarter's spec. You must use both x variables in this simple assignment statement. No casts or other operators other than = and . (dot).

Write a simple assignment statement that could occur in the body of foo() using both of the variables named x such that an error will occur according to this quarter's spec. You must use both x variables in this simple assignment statement. No casts or other operators other than = and . (dot).

Write the appropriate error-free code in the body of bar() to call the function foo() and assign the return value of foo() to the first x variable [the x defined to be an int].

_____________________ analysis deals with verifying correct structure of a program.
_____________________ analysis deals with verifying correct meaning of a program.

What are the 3 ways in C/C++ to subvert the typing system as discussed in class.

1)

2)

3)
Extra Credit

What gets printed when the following C program is executed?

```c
#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;
}
```

A portion of the C Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ postfix increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- postfix decrement</td>
<td></td>
</tr>
<tr>
<td>[] array element</td>
<td></td>
</tr>
<tr>
<td>() function call</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>R to L</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td></td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td></td>
</tr>
<tr>
<td>&amp; address-of</td>
<td></td>
</tr>
<tr>
<td>sizeof size of type/object</td>
<td></td>
</tr>
<tr>
<td>(type) type cast</td>
<td></td>
</tr>
<tr>
<td>* multiplication</td>
<td>L to R</td>
</tr>
<tr>
<td>/ division</td>
<td></td>
</tr>
<tr>
<td>% modulus</td>
<td></td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>