Midterm
CSE 131
Winter 2013

Page 1  (21 points)
Page 2  (21 points)
Page 3  (21 points)
Page 4  (23 points)
Page 5  (18 points)
Page 6  (24 points)

Subtotal  (128 points = 100%)

Page 7  (9 points = 7%)
Extra Credit

Total

This exam is to be taken by yourself with closed books, closed notes, no electronic devices.
You are allowed one side of an 8.5"x11" sheet of paper handwritten by you.
1. Given the following CUP grammar snippet (assuming all other Lexing and terminals are correct):

```java
Expr ::=    Des AssignOp {: System.out.println("A"); :} Expr {: System.out.println("B"); :} |
         Des {: System.out.println("C"); :}
 |
Des ::=    T_PLUSPLUS {: System.out.println("D"); :} Des {: System.out.println("E"); :} |
         T_STAR {: System.out.println("F"); :} Des {: System.out.println("G"); :} |
         T_AMPERSAND {: System.out.println("H"); :} Des {: System.out.println("I"); :} |
         Des2 {: System.out.println("J"); :}
 |
Des2 ::=   Des2 {: System.out.println("K"); :} T_PLUSPLUS {: System.out.println("L"); :} |
         Des3 {: System.out.println("M"); :}
 |
Des3 ::=   T_ID {: System.out.println("N"); :} |
 |
AssignOp ::= T_ASSIGN {: System.out.println("O"); :}
```

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

```java
*x = ++*y++
```

In the above grammar, what is the associativity of the operators in the third production rule (the Des2 ::= rule)? ________________

Which operator in which production has higher precedence in the above grammar:
the T_PLUSPLUS in Des production or the T_PLUSPLUS in Des2 production?

______________________________________________

Is this the pre-increment or the post-increment operator? ________________
2. Assume the following Reduced-C definitions are correct:

```c
structdef RECA
{
    int * ptr;
};

structdef RECB
{
    RECA * ptr;
};

RECB * ptr;
```

a) What type is `(*(*ptr).ptr).ptr`? ______________________________

b) What type is `ptr->ptr`? ______________________________

c) What type is `*(ptr->ptr->ptr)`? ______________________________

d) What type is `*(ptr->ptr)`? ______________________________

From our Reduced-C spec this quarter, name a construct which uses

structural equivalence __________________________

strict name equivalence _________________________

loose name equivalence _________________________

Given the C array declaration

```
int a[2][4];
```

Mark with an A all the memory location(s) where we would find the array element

```
a[1][2]
```

Each box represents a byte in memory.

Now mark with an B all the memory location(s) where we would find the array element

```
a[0][3]
```

Using the Right-Left rule, write the definition of a variable named CSE that is a pointer to an array of 8 elements where each element is a pointer to a function that takes a pointer to a float and returns a pointer to a double.
3. Given the following Reduced-C definitions:

```c
function : float & foo1( float & a ) { return a; }
function : float & foo2( float a )   { return a; }
function : float   foo3( float & a ) { return a; }
function : float   foo4( float a )   { return a; }
```

```c
float x; /* global variables */
int y;
```

For each of the following statements, indicate the type of error (if any) that should be reported according to the Project I spec for this quarter. Use the letters associated with the available errors in the box below.

```c
x = foo1( 4.2 );  ____
x = foo2( y );  ____
y = foo3( x );  ____
x = foo4( foo4( x ) );  ____
x = foo1( x + y );  ____
x = foo2( &x );  ____
x = foo3( y );  ____
foo4( x ) = foo1( x );  ____
foo1( x ) = (int) foo4( x );  ____
y = (int)foo3( (float)y );  ____
*(int *)&foo2( 42 ) = y;  ____
x = foo4( x + y );  ____
&foo2( x ) = (float *)&y;  ____
```

The following RC input program contains **FOUR** erroneous statements per Project I. Indicate which lines are erroneous, and what the error would be from the list of available error messages we provided below.

```c
00:  int a = 0123;
01:  boolean b = true && false;
02:  const int c = 0X420;
03:  const float d = c ^ c;
04:  float * e = nullptr;
05:  function : float & foo() {
06:      return e[-9000];
07:      auto f = &+(+a);
08:      return a;
09:      a = sizeof(e[c]);
10:      return *foo();
11:      return *e;
12:      int[3] g = {0,1,2,3};
13:  }
```

List of types of errors:
A: A syntax error that will not be tested (WNBT)
B: Incompatible type to binary operator
C: Incompatible type to unary operator
D: Left-hand operand is not assignable (not a modifiable L-val)
E: Value not assignable to variable's type
F: Type of return expression not assignment compatible with function's return type
G: Type of return expression not equivalent to the function's return type
H: Return expression is not a modifiable L-value
I: Initialization value of constant not known at compile-time
J: Initialization value not assignable to constant/variable
K: Index expression value in array declaration must be > 0
L: Index value is outside legal range
M: Invalid operand to sizeof. Not a type or not addressable
N: Non-addressable argument to address-of operator
O: Number of initializer expressions exceeds the array size
P: Array initialization expression is not a constant expression

Error #1:  Line #_______  Error Type _______
Error #2:  Line #_______  Error Type _______
Error #3:  Line #_______  Error Type _______
Error #4:  Line #_______  Error Type _______
4. Identify whether each of the following will cause an underlying bit pattern change.

```
int a = 5;
float b = -4.20;
int * ptr1;
float * ptr2;
void foo( float x, float & y ) { /* ... */ }

a = (int) b; __________
foo( a, b ); __________
b = a; __________
a = *(int*) & b; __________
ptr1 = (int*) & b; __________
ptr2 = (float*) ptr1; __________
```

1) Yes – Underlying bit pattern change
2) No – No underlying bit pattern change

According to this quarter's Reduced-C grammar, what two Reduced-C constructs must be uppercase symbols?

__________________________    __________________________

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

```
function : int * foo1() { /* Function body not important. */ }
function : int & foo2() { /* Function body not important. */ }
float[9] f;
float x;
const float c = 5.5;
float *p = &x;

___ x = c ___ p ___ (int*)&x ___ &x ___ c ___ foo2() ___
___ f ___ &*p ___ ++x ___ *(f+2) ___ foo1() ___ ++foo2()
```

Consider the following struct definitions in Reduced-C (similar to C/C++). Specify the size of each struct on a typical ILP-32 compiler mode RISC architecture (like ieng9) or 0 if it is an illegal definition.

```
structdef FOO2 {
    FOO2 * a;
    float b;
    function : void bar( FOO2 &x) {
        int[10] y;
    }
    int[2] c;
    int d;
};
```

```
structdef FOO3 {
    int c;
    int[4] d;
};
```

```
structdef FOO1 {
    int a;
    float b;
    function : void bar() {
        FOO1 *x;
    }
    int c;
    int[4] d;
};
```

Size _____  Size _____  Size _____
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 strictest alignment requirement of its members.

```c
struct foo {
    short a;
    double b;
    char c;
};

struct fubar {
    float d;
    int e;
    char f[5];
    struct foo g;
    short h;
};

struct fubar fubaz;
```

<table>
<thead>
<tr>
<th>low memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

What is the alignment restriction for struct foo? _____ What is the alignment restriction for struct fubar? _____

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 ) -> b
```

__________

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

```
fubaz.
```

__________
6. Fill in the names of the 5 main areas of the C Runtime Environment as laid out by most Unix operating systems (and Solaris on SPARC architecture in particular) as discussed in class. Then state what parts of a C program are in each area.

low memory

_________________________________________________ _____

_________________________________________________ _____

_________________________________________________ _____

_________________________________________________ _____

_________________________________________________ _____

high memory

Identify the following C constructs as either
1) Definition 2) Pure Declaration

___ extern int x;     ___ int foo( int x ) { return x; }

___ struct fubar { int x; } s1;

___ struct fifi;

___ extern int * func1( int x, float y );

When did you use the accessGlobal() method in SymbolTable.java in Project 1?

Using Reduced-C syntax, define a variable named fubaz that is an array of array of int such that fubaz[7][13] is a valid index expression indexing the last element in this array of array. This should take two lines of code.
Extra Credit

What gets printed when the following C program is executed?

```c
#include <stdio.h>

int main()
{
    char a[] = "Zach_and_Mo";
    char *p = a;

    printf( "%c\n", *p++ ); ______
    printf( "%c\n", **p ); ______
    printf( "%c\n", *p = p[-1] - 1 ); ______
    printf( "%c\n", -*p++ ); ______
    printf( "%c\n", ++*p ); ______
    printf( "%c\n", --*p++ ); ______
    printf( "%c\n", ++*(p+6)  ); ______
    printf( "%c\n", p - a ); ______
    printf( "%s\n", a ); ______________________________________________________
    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 sizeof size of type/object (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>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>

Hexadecimal - Character

<table>
<thead>
<tr>
<th>00 NUL</th>
<th>01 SOH</th>
<th>02 STX</th>
<th>03 ETX</th>
<th>04 EOT</th>
<th>05 ENQ</th>
<th>06 ACK</th>
<th>07 BEL</th>
<th>08 BS</th>
<th>09 HT</th>
<th>0A NL</th>
<th>0B VT</th>
<th>0C NP</th>
<th>0D CR</th>
<th>0E SO</th>
<th>0F SI</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 DLE</td>
<td>11 DC1</td>
<td>12 DC2</td>
<td>13 DC3</td>
<td>14 DC4</td>
<td>15 NAK</td>
<td>16 SYN</td>
<td>17 ETB</td>
<td>18 CAN</td>
<td>19 EM</td>
<td>1A SUB</td>
<td>1B ESC</td>
<td>1C FS</td>
<td>1D GS</td>
<td>1E RS</td>
<td>1F US</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>20 SP</td>
<td>21  !</td>
<td>22 &quot;</td>
<td>23 #</td>
<td>24 $</td>
<td>25 %</td>
<td>26 &amp;</td>
<td>27 '</td>
<td>28 (</td>
<td>29 )</td>
<td>2A *</td>
<td>2B +</td>
<td>2C ,</td>
<td>2D -</td>
<td>2E .</td>
<td>2F /</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>30 0</td>
<td>31 1</td>
<td>32 2</td>
<td>33 3</td>
<td>34 4</td>
<td>35 5</td>
<td>36 6</td>
<td>37 7</td>
<td>38 8</td>
<td>39 9</td>
<td>3A :</td>
<td>3B ;</td>
<td>3C &lt;</td>
<td>3D &gt;</td>
<td>3E ~</td>
<td>3F ?</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>40 0</td>
<td>41 A</td>
<td>42 B</td>
<td>43 C</td>
<td>44 D</td>
<td>45 E</td>
<td>46 F</td>
<td>47 G</td>
<td>48 H</td>
<td>49 I</td>
<td>4A J</td>
<td>4B K</td>
<td>4C L</td>
<td>4D M</td>
<td>4E N</td>
<td>4F O</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>50 P</td>
<td>51 Q</td>
<td>52 R</td>
<td>53 S</td>
<td>54 T</td>
<td>55 U</td>
<td>56 V</td>
<td>57 W</td>
<td>58 X</td>
<td>59 Y</td>
<td>5A Z</td>
<td>5B [</td>
<td>5C \</td>
<td>5D ]</td>
<td>5E ^</td>
<td>5F _</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>60 `</td>
<td>61 a</td>
<td>62 b</td>
<td>63 c</td>
<td>64 d</td>
<td>65 e</td>
<td>66 f</td>
<td>67 g</td>
<td>68 h</td>
<td>69 i</td>
<td>6A j</td>
<td>6B k</td>
<td>6C l</td>
<td>6D m</td>
<td>6E n</td>
<td>6F o</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>70 p</td>
<td>71 q</td>
<td>72 r</td>
<td>73 s</td>
<td>74 t</td>
<td>75 u</td>
<td>76 v</td>
<td>77 w</td>
<td>78 x</td>
<td>79 y</td>
<td>7A z</td>
<td>7B {</td>
<td>7C</td>
<td>7D ]</td>
<td>7E ~</td>
<td>7F DEL</td>
</tr>
</tbody>
</table>
Scratch Paper