Midterm
CSE 131
Winter 2014

Page 1 __________ (21 points)
Page 2 __________ (36 points)
Page 3 __________ (28 points)
Page 4 __________ (16 points)
Page 5 __________ (18 points)
Page 6 __________ (20 points)

Subtotal __________ (139 points = 100%)

Page 7 __________ (7 points)

Extra Credit

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

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

Des ::=   T_STAR {: System.out.println("B"); :} Des {: System.out.println("C"); :}
    |   T_AMPERSAND {: System.out.println("D"); :} Des {: System.out.println("E"); :}
    |   T_PLUSPLUS {: System.out.println("F"); :} Des {: System.out.println("G"); :}
    |   Des2 {: System.out.println("H"); :}
    ;

Des2 ::=   Des2 {: System.out.println("I"); :} T_PLUSPLUS {: System.out.println("J"); :}
    |   Des3 {: System.out.println("K"); :}
    ;

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

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

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 post-increment operator have left-to-right associativity or right-to-left associativity? ______________________

If variable z is defined to be type float *, 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 – prog.exe/a.out (Executable image)  
1 – cpp (C preprocessor)  
2 – Program Execution  
3 – ld (Linkage Editor)  
4 – Object file (prog.o)  
5 – Assembly file (prog.s)  
6 – Loader  
7 – ccomp (C compiler)  
8 – as (Assember)  
9 – Source file (prog.c)  
10 – Segmentation Fault (Core Dump) / General Protection Fault

gcc _____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____

Given the C following definitions (same rules in our Reduced-C compiler)

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

indicate whether each of the following statements will cause an equivalence compiler error or not.

A) Error  
B) No Error

```c
_____ a = b;  _____ a.a = b.a;  
_____ a = (struct S1)b;  _____ (struct S2)a = b;  
_____ a = *(struct S1 *)b;  _____ *(struct S2 *)a = b;  
_____ a = *(struct S1 *)&b;  _____ *(struct S2 *)&a = b;
```

Using Reduced-C syntax, define an array of an array of ints with dimensions 8x4 named bar such that

```c
bar[7][3] = 42;
```

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

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

_____ b.a++  _____ *d+1  _____ &b  _____ (int)a[3]  _____ c->a % b.a  
_____ foo()  _____ &a[2]  _____ (R1 *)foo()  _____ a[1] = *foo()  _____ ++*d++  
_____ *foo()  _____ a[7]  _____ *&b  _____ a  _____ *d++
```
3. Given the following Reduced-C code and list of statements, indicate for each numbered statement the type of error that should be reported according to the Project I spec for this quarter (which is similar to C++ rules). Use the letters associated with the available errors in the box on the right, or choose A for No error.

```c
int * [5] a;
const int b = 5;
bool c, d;
function : int foo(){ /* ... */ return 0; }
``` 

<table>
<thead>
<tr>
<th>Statement</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) b = 3;</td>
<td>A</td>
</tr>
<tr>
<td>2) &amp;&amp;a[0];</td>
<td>C</td>
</tr>
<tr>
<td>3) a[b-2] = &amp;b;</td>
<td>D</td>
</tr>
<tr>
<td>4) (int)c = 4;</td>
<td>E</td>
</tr>
<tr>
<td>5) &amp;foo();</td>
<td>A</td>
</tr>
<tr>
<td>6) *a[foo()] = b;</td>
<td>B</td>
</tr>
<tr>
<td>7) a[2] = (int *) &amp;c;</td>
<td>C</td>
</tr>
<tr>
<td>8) ++b;</td>
<td>D</td>
</tr>
<tr>
<td>9) (c = d) = true;</td>
<td>D</td>
</tr>
<tr>
<td>10) c = d = true;</td>
<td>A</td>
</tr>
<tr>
<td>11) (*a[0])++ = *a[1];</td>
<td>A</td>
</tr>
<tr>
<td>12) *a[0]++ = *a[1];</td>
<td>A</td>
</tr>
</tbody>
</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

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) No constant folding</td>
<td>A</td>
</tr>
<tr>
<td>B) Constant folding</td>
<td>B</td>
</tr>
</tbody>
</table>

Using the Right-Left rule write the C definition of a variable named foo that is a pointer to a 2-d array of 3 rows by 14 columns where each element is a pointer to a function that takes a pointer to a pointer to a long as a single parameter and returns a pointer to an array of 5 elements where each element is a pointer to a struct bar.
### 4. Struct Definitions and Memory Locations

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.

<table>
<thead>
<tr>
<th>Struct Definition</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>structdef FOO1</td>
<td></td>
</tr>
<tr>
<td>structdef FOO2</td>
<td></td>
</tr>
<tr>
<td>structdef FOO3</td>
<td></td>
</tr>
</tbody>
</table>

Identify whether each of the following will cause an underlying bit pattern change.

<table>
<thead>
<tr>
<th>Expression</th>
<th>A) Yes – Underlying bit pattern change</th>
<th>B) No – No underlying bit pattern change</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a = 5;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float b = -4.20;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int * ptr1;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>float * ptr2;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>void foo( float x, float &amp; y ) { /* ... */ }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = (int) b;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>foo( a, b );</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = a;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = * (int *) &amp; b;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ptr1 = (int *) &amp; b;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ptr2 = (float *) ptr1;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Given the C array declaration

```c
int b[4][2];
```

Mark with a **B** all the memory location(s) where we would find the array element `b[1][1]`.

```
low memory    high memory
```

Each box represents a byte in memory.

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.

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

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

```c
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.

```c
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;
```

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

What is the `sizeof( struct fubar )`? ____  What is the `offsetof( struct fubar, g.b[3] )`? ____

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. 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

[Diagram showing five boxes with arrows pointing downwards]

Which 3 areas of the above are essentially determined at compile time and are part of an executable image (for example, an a.out or .exe file)?

1) __________________________

2) __________________________

3) __________________________

Give the order of the phases of compilation in a typical C compiler as discussed in class:

0 – Scanner (Lexical Analysis)
1 – Parser (Semantic Analysis)
2 – Parser (Syntax Analysis)
3 – Target language file (for ex., prog.s)
4 – Source language file (for example, prog.c)
5 – Intermediate Representation(s)
6 – Code generation (for ex., Assembly)

_____ → _____ → _____ → _____ → _____ → _____ → _____ → _____
Extra Credit

What gets printed when the following C program is executed?

```c
#include <stdio.h>

int main()
{
    char a[] = "Rohan";
    char *ptr = a;

    printf( "%c\n", *(ptr++ + 2) + 1 );  _____
    printf( "%c\n", *(a+2) = *++ptr + 5 );  _____
    printf( "%c\n", *++ptr );  _____
    printf( "%c\n", *++ptr + 9 );  _____
    printf( "%c\n", *a = ptr[-sizeof(ptr)] + 1); _____
    printf( "%d\n", --ptr - 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>R to L</td>
</tr>
<tr>
<td>-- postfix decrement</td>
<td>L to R</td>
</tr>
<tr>
<td>() function call</td>
<td></td>
</tr>
<tr>
<td>* indirection</td>
<td>L to R</td>
</tr>
<tr>
<td>++ prefix increment</td>
<td>R to L</td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td>L to R</td>
</tr>
<tr>
<td>&amp; address-of (type) sizeof type/objec</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>.</td>
<td></td>
</tr>
<tr>
<td>- assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>

---

Hexadecimal - Character

<table>
<thead>
<tr>
<th></th>
<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>
</tr>
</thead>
<tbody>
<tr>
<td>08 BS</td>
<td>09 HT</td>
<td>0A NL</td>
<td>0B VT</td>
<td>0C SP</td>
<td>0D CR</td>
<td>0E SO</td>
<td>0F SI</td>
<td></td>
</tr>
<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></td>
</tr>
<tr>
<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>
<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></td>
</tr>
<tr>
<td>28 (</td>
<td>29 )</td>
<td>2A *</td>
<td>2B +</td>
<td>2C ,</td>
<td>2D -</td>
<td>2E .</td>
<td>2F /</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></td>
</tr>
<tr>
<td>38 8</td>
<td>39 9</td>
<td>3A :</td>
<td>3B ;</td>
<td>3C &lt;</td>
<td>3D =</td>
<td>3E &gt;</td>
<td>3F ?</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></td>
</tr>
<tr>
<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>
<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></td>
</tr>
<tr>
<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>
<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></td>
</tr>
<tr>
<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>
<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></td>
</tr>
<tr>
<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>
<td></td>
</tr>
</tbody>
</table>
Scratch Paper