Midterm
CSE 131
Winter 2015

Page 1 ________ (30 points)
Page 2 ________ (26 points)
Page 3 ________ (28 points)
Page 4 ________ (28 points)
Page 5 ________ (30 points)

Subtotal ________ (142 points = 100%)

Page 6 ________ (8 points)
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):

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

What is the output when parsing the following expression (you should have 22 lines/letters in your output) [22 pts]:

```
*x[ *y ]++ = ++z[ sizeof(y[0]) ];
```

In the above grammar, does the assignment operator have left-to-right or right-to-left associativity? [2 pts]

__________________________

If variable `z` is defined to be type `float*`, what type must variables `x` and `y` be defined for this expression to be semantically correct? Do not define these variables using arrays. [6 pts]

__________________________

```
2. 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], p[1], p[2], etc. If the member/field is a struct, use the member name followed by its member names (e.g. p.a, p.b, etc.). If the member/field is an array of structs, include both the index and member name (e.g. p[0].a). Place an X in any bytes of padding. Structs/unions are padded so the total size is evenly divisible by the strictest alignment requirement of its members. [16 pts figure; 10 pts questions]

```c
struct bird {
  char    a;
  short   b;
  int     c;
};

struct tree {
  char    d;
  struct bird e[2];
  double  f;
  char    g;
};

struct tree redwood;
```

| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| low memory | | | |
| high memory | | | |
| high memory | | | |
| high memory | | | |
| high memory | | | |
| high memory | | | |
| high memory | | | |
| high memory | | | |
| high memory | | | |

What is the `sizeof(struct tree)`? _______

What is the `offsetof(struct tree, e[1].b)`? _______

If `struct tree` had been defined as `union tree` instead, what would be `sizeof(union tree)`? _______

If you rearrange the order of the struct members in `struct tree` to minimize padding, what would be the size of this modified `struct tree`? _______

How many bytes are saved by rearranging the `struct tree` members to minimize padding? _______

2
### What does the program output if the language uses **static** scoping?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

### What does the program output if the language uses **dynamic** scoping?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

3. Given the following pseudocode, determine the program output based on the specified scoping rule: [28 pts]

```c++
int x = 0;    // global var declaration
int y = 0;    // global var declaration

void sum_xy( int a, int b )
    x = x + a;
    y = y + b;

void gamma()
    sum_xy( 10, 10 );
    cout << x << endl;
    cout << y << endl;

void beta()
    int y = 7;    // local var declaration
    sum_xy( 3, 3 );
    gamma();
    cout << x << endl;
    cout << y << endl;

void alpha()
    int x = 1;    // local var declaration
    sum_xy( 7, 7 );
    beta();
    cout << x << endl;
    cout << y << endl;

sum_xy( 2, 2 );
alpha();
beta();
gamma();
cout << x << endl;
cout << y << endl;
```

---

What does the program output if the language uses **static** scoping?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

What does the program output if the language uses **dynamic** scoping?

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
4. Given the C array declaration on a SPARC architecture

```c
char a[2][3][5];
```

What is the `sizeof(a)`? ______________ [2 pts]

Mark with an A all the memory location(s) where we would find the array element `a[1][0][4]`: [4 pts]

```
a:  
```

(Each box represents a byte in memory)

Given the following Reduced-C code:

```c
structdef RECA {
    RECA * ptr1[5];
    float * ptr2[5];
};
structdef RECB {
    RECB * ptr1[5];
    RECA * ptr2[5];
    bool * ptr3[5];
};
RECB * ptr1[5][5];
RECA ** ptr2[5];
```

Indicate the data type of the following expressions and circle what kind of value it is: [22 pts]

<table>
<thead>
<tr>
<th>Data Type:</th>
<th>Kind of Value (circle one):</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ptr1[0][1]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>ptr2[0][1][2]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>ptr2[0][1][2].ptr2[3][4]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>&amp;ptr1[0][1]-&gt;ptr1[2][3]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>(**+**ptr2[0]).ptr1</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>-ptr2[0][1]-&gt;ptr2[2][3]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>!*ptr1[0][1]-&gt;ptr3[2]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>ptr1[0][1]-&gt;ptr1[2]-&gt;ptr2[3]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>*ptr2[0][1]-&gt;ptr1[2]-&gt;ptr1[3]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>&amp;ptr2[0]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
<tr>
<td>&amp;ptr1[0][1][2].ptr2[0]</td>
<td>[ Mod L-Val / Non-Mod L-Val / R-Val ]</td>
</tr>
</tbody>
</table>
5. Given the following C code, indicate which of the 5 main areas of the C Runtime Environment each expression is located in:

```c
int A = 1, B = 5;
static int C = 1;

int * D ( int E ) {
    int F = 3;
    static int G = E;
    static float H;
    return &G;
}

int main ( int argc, char *argv[] ) {
    static int * I;
    I = (int *) calloc(1, sizeof(int));
    int * J = D(*I);
    int K[50];
    int ** L = &J;
    K[0] = **L;
    <------ determine runtime areas at this point in time
    free(I);
    return 0;
}
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

State whether constant folding will be performed on one or more arithmetic expressions in the following statements by the compiler according to this quarter’s Reduced-C spec (Y or N):

```c
function : void main() {
    int a = 10;
    const int b = 5;
    int c = b + 5;  ------
    const int d = b + b; -----
    int e[d * 5]; -----
    a = a + 2; -----
    c = b + 2; -----
    e[b + d] = a; -----
    e[-5 + (d / 2)]++; -----
    c = e[a + b]; -----
    c = e[a + 10]; -----
    e[c * c] = b; -----
    c = (1 + 2) % 3; -----
    e[e[b]] = e[e[4] + e[5]]; -----
}
```
Extra Credit

What gets printed when the following C program is executed? [8 pts]

```c
#include <stdio.h>
int main() {
    char a[] = "GEORGE";
    char *p = a;

    printf("%c\n", *++p = '-');
    printf("%c\n", (p[-1] -= 5) + 6);
    printf("%c\n", 2[p] = --(p - 1));
    printf("%c\n", ++p - 14);
    printf("%c\n", (2[p] = *p + 2) + 12);
    printf("%c\n", *(p = p + 4) += 18);
    printf("%c\n", *++p = a[1] - 12);

    printf("%s\n", a);
}
```

A portion of the C Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+, -</td>
<td>L to R</td>
</tr>
<tr>
<td>*, /, %</td>
<td>L to R</td>
</tr>
<tr>
<td>+=, -=, *=, /=</td>
<td>R to L</td>
</tr>
<tr>
<td>++ post increment</td>
<td>L to R</td>
</tr>
<tr>
<td>-- post decrement</td>
<td>L to R</td>
</tr>
<tr>
<td>[] array element</td>
<td>R to L</td>
</tr>
<tr>
<td>() function call</td>
<td>R to L</td>
</tr>
<tr>
<td>sizeof size of type/object</td>
<td>R to L</td>
</tr>
<tr>
<td>(type) type cast</td>
<td>R to L</td>
</tr>
<tr>
<td>*= indirection</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>
</tr>
</thead>
<tbody>
<tr>
<td>08 BS</td>
<td>09 HT</td>
<td>0A NL</td>
<td>0B VT</td>
<td>0C NF</td>
<td>0D CR</td>
<td>0E SO</td>
<td>0F SI</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>
</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>
</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>
</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>
</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>
</tr>
<tr>
<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>40 @</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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
<tr>
<td>78 x</td>
<td>79 y</td>
<td>7A z</td>
<td>7B {</td>
<td>7C</td>
<td></td>
<td>7D }</td>
<td>7E ~</td>
</tr>
</tbody>
</table>