Midterm
CSE 131
Fall 2013

Page 1 __________ (30 points)
Page 2 __________ (24 points)
Page 3 __________ (22 points)
Page 4 __________ (38 points)
Page 5 __________ (32 points)
Page 6 __________ (34 points)

Subtotal __________ (180 points = 100%)

Page 7 __________ (20 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"); : } AssignOp Expr { : System.out.println("B"); : }
| Des { : System.out.println("C"); : }

Des ::= T_STAR { : System.out.println("D"); : } Des { : System.out.println("E"); : }
| T_PLUSPLUS { : 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

Des3 ::= T_ID

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

What is the output when parsing the follow expression (you should have 20 lines/letters in your output) [20 pts]:

```
*z = **x++ = &y;
```

<table>
<thead>
<tr>
<th>Output</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>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

In the above grammar, which has higher precedence: pre-increment operator or post-increment operator, or do they have the same precedence? [2 pts]

```
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 double **, what type must variable x and y be defined for this expression to be semantically correct? [6 pts]
```

```
x;
```

```
y;
```
2. Give the order of the typical GCC compilation stages and on to actual execution as discussed in class [6 pts]:

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

```
```

Given the C array declaration

```c
short a[2][7];
```

Mark with an A all the memory location(s) where we would find the array element `a[1][5]`: [4 pts]

```
a:

low memory ____________________________

(Each box represents a byte in memory)

high memory__________________________
```

Using Reduced-C syntax, define an array of an array of bools with dimension 9x3 named `foo` such that `foo[8][2] = false;` is a valid expression. This will take two lines of code. [4 pts]

```
_______________________________
_______________________________
```

Given the following Reduced-C code:

```c
structdef RECA {
    int * ptr1;
    RECA * ptr2;
};
structdef RECB {
    RECA * ptr1;
    RECB * ptr2;
};
RECB * ptr1;
```

What is the type of the following expressions? [10 pts]

```
*(*(*ptr1).ptr1).ptr1 ____________________________
ptr1->ptr2->ptr2->ptr2 ____________________________
ptr1->ptr2->ptr1->ptr2 ____________________________
&*(ptr1->ptr1->ptr1) ____________________________
*(*(ptr1->ptr1)).ptr2 ____________________________
```
3. 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.). 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. [10 pts figure; 12 pts questions]

![Memory Layout Diagram](image)

**struct foo {**
- `double a;`
- `short b;`
**}**

**struct bar {**
- `char c[9];`
- `double d;`
- `short e;`
- `struct foo f;`
- `int g;`
**}**

**struct bar baz;**

---

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

What is the `offsetof(struct bar, f.a) `? _______

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

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

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

Does it matter whether you arrange the struct members in `struct bar` from largest to smallest data type or smallest to largest data type to minimize padding? _______
4. Given the following Reduced-C definitions:

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

float x; /* global variable */
int y; /* global variable */
bool z; /* global variable */

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. [26 pts]

(A) Argument not assignable to value param  (B) Argument not equivalent to reference param  (C) Argument not a modifiable L-val to reference param  (D) Left-hand operand is not a modifiable L-val  (E) Right-hand-side type not assignable to left-hand-side type  (F) No Error

```
foo1( 4.2 );   ___
foo1( x++ );   ___
foo1( y );    ___
foo1( x + y ); ___
foo1( z );    ___
   x = foo2( x ); ___
   y = foo2( x ); ___
foo2( x ) = x; ___
foo3( x ) = foo2( x ); ___
foo1( (float) y ); ___
    (int) foo2( x ) = y; ___
*&x = foo3( *&x ); ___
foo3( x ) = *(float *) &y; ___
```

Using the Right-Left rule (which follows the operator precedence rules) write the C definition of a variable named alpha that is a pointer to a pointer to a function that takes a pointer to a char and a double as two parameters and returns a pointer to an array of 19 elements where each element is a pointer to int: [6 pts]

```
______________________________;
```

Using the Right-Left rule (which follows the operator precedence rules) write the C definition of a variable named alpha that is a pointer to a pointer to a function that takes a pointer to a char and a double as two parameters and returns a pointer to an array of 19 elements where each element is a pointer to int: [6 pts]

```
______________________________;
```

Given the following definitions:

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

Give a one-line example of a converting type cast using only variables `a` and `b` defined above and any appropriate type cast(s) and any appropriate operators: [3 pts]

```
__________________________________________;
```

Give a one-line example of a non-converting type cast using only variables `a` and `b` defined above and any appropriate type cast(s) and any appropriate operators: [3 pts]

```
__________________________________________;
```
5. 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. [10 pts]

![Diagram showing the 5 main areas of the C Runtime Environment]

low memory

______________________________________________________
______________________________________________________
______________________________________________________
______________________________________________________
______________________________________________________

high memory

Given the following C code, indicate which of the 5 main areas of the C Runtime Environment (which you just described in the question above) each symbol is located in: [18 pts]

```c
int A = 42, B;
static int C = 17, D;

int E ( int F ) {
    int G = 5, H;
    static int I = 3, J;
    return 0;
}

int main ( int argc, char *argv[] ) {
    int K = -44;
    static int L = 88, M;
    int * N = (int *) malloc(sizeof(int));
    E(3);
    return 0;
}
```

For the above program, while there are no compile-time or run-time errors, there is a serious programming mistake that should be avoided. Explain what the mistake is and what needs to be done to fix the issue: [4 pts]
6. 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. [9 pts]

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

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

```
structdef DELTA {
    DELTA *a;
    float b;
    BETA *c;
    int *d;
    function : void bar()
    {
        DELTA *c = this.a;
    }
    int *d;
    int[2] e;
};
```

Size _______ Size _______ Size _______

For the T_AND, T_CARET, and T_BAR operators, the operand types must be equivalent to _____________________, and the resulting type is _____________________.[2 pts]

The result of a type cast is (circle one): [ Modifiable L-Val / Non-Modifiable L-Val / R-Val ] [1 pt]

The result of an address-of is (circle one): [ Modifiable L-Val / Non-Modifiable L-Val / R-Val ] [1 pt]

The result of a dereference is (circle one): [ Modifiable L-Val / Non-Modifiable L-Val / R-Val ] [1 pt]

What occurs when an object lives longer than its binding? _______________________________ [2 pts]

What occurs when a binding lives longer than its object? _______________________________ [2 pts]

Given the following C code, indicate the output for each statement, or -1 if the statement is an error: [10 pts]

```
#include <stdio.h>
void main() {
    int arr[5] = {3, 1, 4, 1, 5};
    printf("%d\n", arr[4]);
    printf("%d\n", arr[arr[arr[0]]]);
    printf("%d\n", 1[arr]);
    printf("%d\n", arr[2[arr]]);
    printf("%d\n", arr[arr[1]+3[arr]]);
}
```

Identify the following C constructs as either: [6 pts]

(A) Pure Declaration
(B) Definition

_____ int ** foo ( int x, float y);
_____ void foo() { }

_____ struct f { int x; } f1;
_____ float * y;

_____ extern int x;
_____ extern int * func1( int x, float y);
Extra Credit

What gets printed when the following C program is executed? [9 pts]

```c
#include <stdio.h>
int main() {
    char a[] = "CSE131";
    char *p = a;
    printf( "%c\n", *(a+2) = *p++ );
    printf( "%c\n", *(p+3) = *p );
    printf( "%c\n", *(p-1) = -*p );
    printf( "%c\n", *p = *p - 3 );
    printf( "%c\n", *(p+3) + 1 );
    printf( "%c\n", *p+3 );
    printf( "%c\n", *(p-2) - 4 + 1 );
    printf( "%c\n", *(++p+1) = '\' );
    printf( "%s\n", a );
    return 0;
}
```

Why do CSE students always confuse Halloween and Christmas? [1 pt]

Consider the following pseudocode and scoping questions [10 pts]:

```c
int x;         // global var declaration

void set_x( int n )
    x = n;

void alpha()
    set_x( 4 );
    cout << x;

void beta()
    int x = 5; // local var declaration
    cout << x;
    set_x( 7 );
    cout << x;
    set_x( 0 );

alpha();
cout << x;
beta();
cout << x;
```

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

_________
_________
_________
_________

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

_________
_________
_________
_________
<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>.</td>
<td></td>
</tr>
<tr>
<td>= assignment</td>
<td>R to L</td>
</tr>
</tbody>
</table>