Midterm
CSE 131
Winter 2008

Page 1  ___________ (20 points)
Page 2  ___________ (21 points)
Page 3  ___________ (12 points)
Page 4  ___________ (19 points)
Page 5  ___________ (14 points)
Page 6  ___________ (14 points)

Subtotal  ___________ (100 points)

Page 6  ___________ (7 points)

Extra Credit

Total  ___________
1. Consider the following C-like code:

```c
int x = 8;

int f()
{
    print(x);
    return x;
}

int g()
{
    int x = 5;
    print(x);
    return f();
}

int main()
{
    print(g());
}
```

What does the program output if the compiler implements **dynamic** scoping? (3 pts)  
What does the program output if the compiler implements **static** scoping? (3 pts)  

_____        _____
_____        _____

Which kind of scoping (dynamic or static) adds run time overhead compared to the other kind of scoping? (1 pt)

Using the Right-Left rule write the C definition of a variable named `crazy` that is a pointer to an array of 6 elements where each array element is of type pointer to a function which takes one argument, a pointer to a pointer to a float, and returns a pointer to an array of 3 elements where each array element is of type pointer to struct `screwball`. (9 pts)

Using the Right-Left rule write the C definition of a variable named `crazy` that is a pointer to an array of 6 elements where each array element is of type pointer to a function which takes one argument, a pointer to a pointer to a float, and returns a pointer to an array of 3 elements where each array element is of type pointer to struct `screwball`. (9 pts)

Fill in the blanks for each letter with the correct name of the C compilation stage (4 pts)

```
gcc foo.c -> A -> B -> C -> D -> a.out
            foo.s foo.o
```

A. _____________________  C. _____________________
B. _____________________  D. _____________________
2. Given the following Reduced-C program and following the Project I spec for parameter passing type checking, for each function call determine if a semantic error will occur (and which kind of error). Use the letters below for your answers. (17 pts)

A. No Error
B. Assignability Error
C. Equivalence Error
D. Addressability Error

function : void foo0 ( int x ) { /* ... */ }
function : void foo1 ( float x ) { /* ... */ }
function : void foo2 ( float & x ) { /* ... */ }
function : void foo3 ( float[5] & x ) { /* ... */ }
function : void foo4 ( float * x ) { /* ... */ }
function : void foo5 ( float * & x ) { /* ... */ }

function : int main()
{
    int a;
    float b;
    int[5] c;
    float[5] d;

    foo0( a );  
    foo0( b );  
    foo1( a );  
    foo1( b );  
    foo2( a );  
    foo2( a + b );  
    foo2( b );  
    foo2( * & b );  
    foo3( c );  
    foo3( d );  
    foo4( c );  
    foo4( d );  
    foo4( & a );  
    foo4( & b );  
    foo4( ( float * ) & a );  
    foo5( & b );  
    foo5( ( float * ) & a );  
}

What four Reduced-C (and C) operators evaluate to a modifiable l-val? (4 pts)

_________  _________  _________  _________
3. The C compiler (unlike the C++ compiler) requires all initialized static data (global and static variables) to be initialized with compile time constant expressions so the compiler can set the initial values for these variables in the Data segment statically at compile time. Non-static data (local variables) only exist at run time, so they do not have this requirement. State which initializations in the C program below will elicit a C compiler error and which initializations will not elicit an error. Use the letters below for your answer. (8 pts)

A. No Compiler Error  B. Compiler Error

```c
#include <stdlib.h>  /* For function prototype of malloc() */

int foo( int x ) { return x + 5; }

int a0 = 5 + 10;  _____
int a1 = foo( a0 );  _____
int a2 = a0 + 5;  _____
int *a4 = (int *) malloc( 5 + 10 );  _____

int main()
{
    int c0 = 5 + 10;  _____
    int c1 = foo( c0 );  _____
    int c2 = c0 + 5;  _____
    int *c4 = (int *) malloc( 5 + 10 );  _____
    return 0;
}
```

4. Given the following Java statements, characterize the class of error detected. (4 pts)

A. Lexical error detected by the scanner
B. Syntax error detected by the grammar
C. Static semantic error detected by the parser/semantic analyzer
D. Dynamic semantic error detected by the run time environment

```java
double[] arr = new double[10];

int n;

/* Read user input (specifically the value 17) into the variable n. */
arr[n] = 42.4012;  _____  /* Array reference out of bounds */

double d0 = 42.4.12;  _____  /* Should be 42.4012; */

int i0 = 42.4012;  _____  /* Should be an int initializer (not 42.4012) */
double d1 + 42.4012;  _____  /* Should be = (not +) */
```
4.

Given the following array definition

```c
/* C */
short x[13][27];
```

write the assembly level address calculation expression taking into account scalar arithmetic to access `x[i][j]`

```
(( x + _________________________________________ ) + _______________________________________ )
```

The result is the address of where we can find this array element. (8 pts)

---

Given the array declaration

```c
int a[3][2];
```

Mark with an A the memory location(s) where we would find `a[1][1]`

```
low memory    high memory
```

Each box represents a byte in memory. (4 pts)

---

Using Reduced-C syntax from Project I, define a pointer to an array of 5 ints named `foo` such that `(*foo)[4]` is a valid expression. This will take two lines of code. (4 pts)

---

Given the following Reduced-C code, state whether the assignment statements are semantically correct or not. (3 pts)

```
structdef REC1 { float a; };
structdef REC2 { float a; };
typedef REC1 REC3;
REC1 r1;
REC2 r2;
REC3 r3;
```

```
r1 = r1; _____ A. Error
r1 = r2; _____ B. No Error
r1 = r3; _____
```
5. Reduced-C (like C++) allows variables to be defined anywhere within a code block with that variable's scope ranging from the point of that variable's definition to the end of the code block, possibly hiding a symbol with the same name defined in an outer scope. You dealt with this scoping mechanism in Project I. What gets printed with the following program? (3 pts)

```c
function : int main()
{
    int i = 2;
    while ( i == 2 ) {
        i = i - 1;
        int i = 3;
        i = i + 2;
        if ( i > 2 ) {
            i = i - 3;
            int i = 4;
            i = i + 4;
            cout << i; _____
        }
        cout << i; _____
    }
    cout << i; _____
    return 0;
}
```

With regard to the following C array and pointer definitions: (3 pts)

```c
double arr[5];
double * ptr = arr;
```

What type is `arr`? ________________

What type is `*&arr[1]`? ________________

If `&arr[0]` is 4000, what is `&arr[2]`? ________________

Write two different statements only using `ptr` (not using `arr`) to assign the value 42.5 to the second array element in the array `arr`. (2 pts)

_________________________ _________________________

Write two different statements only using `ptr` (not using `arr`) to assign `ptr` to point to the last array element in `arr`. (2 pts)

_________________________ _________________________

Explain the difference between a pure declaration and a definition? (2 pts)

Give an example of a pure variable declaration. (1pt) Give an example of a variable definition. (1pt)
6. Show the memory layout of the following C struct/record definition taking into consideration the SPARC data type memory alignment restrictions discussed in class. Fill bytes in memory with the appropriate struct/record 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. 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. (14 pts)

```c
struct foo {   low memory
    char   a[6]; fubar:
    double b;
    float  c;
    short  d;
    int    e[3];
};

struct foo fubar;
```

What is the `offsetof(struct foo, e[1])`? _______

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

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

If you rearranged the order of the struct members in struct foo to minimize padding, how many bytes of padding would you need? _______

And what would be the size of this modified struct? _______

Does it matter whether you arrange the struct members in struct foo from largest to smallest data type or smallest to largest data type to minimize padding? _______
Extra Credit (7 points)

What gets printed by the following C program?

```c
#include <stdio.h>

int main()
{
    char a[] = "CSE131";
    char *p = a;

    printf( "%c", *p++ );  __________
    printf( "%c", ++*p );  __________
    printf( "%c", *++p + 1 );  __________
    p++;
    printf( "%c", *p = *p + 3 );  __________
    printf( "%c", --*p++ );  __________
    printf( "%d", p - a );  __________
    printf( "%s", a );  ________________
    return 0;
}
```

A portion of the Operator Precedence Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>postfix increment L to R</td>
</tr>
<tr>
<td>--</td>
<td>postfix decrement L to R</td>
</tr>
<tr>
<td>*</td>
<td>indirection R to L</td>
</tr>
<tr>
<td>++</td>
<td>prefix increment R to L</td>
</tr>
<tr>
<td>--</td>
<td>prefix decrement R to L</td>
</tr>
<tr>
<td>&amp;</td>
<td>address-of</td>
</tr>
<tr>
<td>*</td>
<td>multiplication L to R</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
</tr>
<tr>
<td>+</td>
<td>addition L to R</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
<tr>
<td>=</td>
<td>assignment R to L</td>
</tr>
</tbody>
</table>