Final
CSE 131
Winter 2015

Page 1 __________ (25 points)
Page 2 __________ (24 points)
Page 3 __________ (30 points)
Page 4 __________ (28 points)
Page 5 __________ (24 points)
Page 6 __________ (24 points)
Page 7 __________ (24 points)
Page 8 __________ (24 points)
Page 9 __________ (48 points)

Subtotal __________ (251 points) = 100%

Page 10 __________ (18 points) [7% Extra Credit]

Extra Credit

Total __________

This exam is to be taken by yourself with closed books, closed notes, no electronic devices.
You are allowed both sides of an 8.5"x11" sheet of paper handwritten by you.
1. What gets printed in the following C++ program (just like Reduced-C without "function : " in front of each function definition)? If a value is unknown/undefined or otherwise cannot be determined by the code given, put a question mark ('?') for that output. Hint: Draw stack frames! [25pts]

```cpp
int neo = 1;
int trinity = 3;
int morpheus = 9;
int smith;

int & keymaker( int a, int & b, int * c ) {
    static int merovingian = a + b + *c;
    *c = a++ + ++b;
    smith = ++merovingian;
    return merovingian;
}

void oracle( int * a, int b, int & c ) {
    cout << *a << endl;  
    cout << b << endl;   
    cout << c << endl;   
    cout << (*a = ++c - b++) << endl;    
    cout << neo << endl;  
    cout << trinity << endl;  
    cout << morpheus << endl;  
    cout << smith << endl;   
    cout << *a << endl; 
    cout << b << endl;   
    cout << c << endl;   
    cout << keymaker(*a, *a, a) << endl;  
    cout << keymaker(b, b, &b) << endl;   
    cout << keymaker(c, c, &c) << endl;   
    cout << neo << endl;  
    cout << trinity << endl;  
    cout << morpheus << endl;  
    cout << smith << endl;   
    cout << *a << endl; 
    cout << b << endl;   
    cout << c << endl;   
}

int main() {
    oracle( &neo, trinity, morpheus );
    cout << neo << endl;  
    cout << trinity << endl;  
    cout << morpheus << endl;  
    cout << smith << endl;   
    return 0;
}
```
2. What gets printed in the following Reduced-C program? [24pts]

```c
struct MS {
    int x;
    MS() {
        this.x = 99;
        cout << "A: " << this.x << endl;
    }
    MS(int x) {
        this.x = x;
        cout << "B: " << this.x << endl;
    }
    ~MS() {
        cout << "C: " << ++this.x << endl;
    }
};

MS s1;
MS s2 : (5);

function : void foo() {
    MS s3 : (10);
    static MS s4 : (15);
    return;
    MS s5 : (20);
}

function : void main() {
    MS s6 : (25);
    foo();
    MS s7 : (30);
}
```
3. In object-oriented languages like Java, determining which method code/instructions to bind to (to execute) is done at run time rather than at compile time (this is known as dynamic dispatch or dynamic binding). However, the name-mangled symbol denoting a particular method name is determined at compile time. Given the following Java class definitions, specify the output of each print() method invocation. [30pts]

```java
class Tyrell {
    public void print(Tyrell p) {
        System.out.println("1");
    }
}
class Baratheon extends Tyrell {
    public void print(Tyrell p) {
        System.out.println("2");
    }
    public void print(Baratheon p) {
        System.out.println("3");
    }
}
class Lannister extends Baratheon {
    public void print(Tyrell p) {
        System.out.println("4");
    }
    public void print(Baratheon p) {
        System.out.println("5");
    }
    public void print(Lannister p) {
        System.out.println("6");
    }
}
public class Overloading_Final_Exam {
    public static void main (String [] args) {
        Tyrell margaery = new Tyrell();
        Tyrell loras    = new Tyrell();
        Tyrell myrcella = new Baratheon();
        Baratheon stanis = new Baratheon();
        Baratheon renly  = new Baratheon();
        Baratheon tommen = new Lannister();
        Lannister joffrey = new Lannister();
        joffrey.print( tommen );
        joffrey.print( myrcella );
        joffrey.print( margaery );
        stanis.print( joffrey );
        stanis.print( myrcella );
        renly.print( margaery );
        loras.print( renly );
        margaery.print( tommen );
        margaery.print( (Lannister) tommen );
        margaery.print( joffrey );
        ( (Lannister) tommen ).print( joffrey );
        ( (Baratheon) myrcella ).print( tommen );
        ( (Baratheon) joffrey ).print( joffrey );
        ( (Tyrell) renly ).print( stanis );
        ( (Tyrell) joffrey ).print( tommen );
    }
}
```

Now remove the entire print(Tyrell p) {} method in class Lannister and remove the entire print(Baratheon p) {} method in class Baratheon. Specify the output of each print() method with these changes below.

```java
Now remove the entire print(Tyrell p) {} method in class Lannister and remove the entire print(Baratheon p) {} method in class Baratheon. Specify the output of each print() method with these changes below.
```
4. Given the following pseudocode, determine the program output based on the specified scoping rule: [28pts]

```c
int x = 50;    -- global var declaration
int y = 70;    -- 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 = 10;    -- local var declaration
  sum_xy( 1, 1 );
  gamma();
  cout << x << endl;
  cout << y << endl;

void alpha()
  int x = 9;     -- local var declaration
  sum_xy( 3, 3 );
  beta();
  cout << x << endl;
  cout << y << endl;

sum_xy( 5, 5 );
alpha();
beta();
gamma();
cout << x << endl;
cout << y << endl;
```

<table>
<thead>
<tr>
<th>What does the program output if the language uses <strong>static</strong> scoping?</th>
<th>What does the program output if the language uses <strong>dynamic</strong> scoping?</th>
</tr>
</thead>
<tbody>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
<tr>
<td>__________</td>
<td>__________</td>
</tr>
</tbody>
</table>
5. Give the order of the typical GCC compilation stages and on to actual execution as discussed in class [2pts]:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Object file (prog.o)</td>
</tr>
<tr>
<td>1</td>
<td>prog.exe/a.out (Executable image)</td>
</tr>
<tr>
<td>2</td>
<td>Program Execution</td>
</tr>
<tr>
<td>3</td>
<td>Loader</td>
</tr>
<tr>
<td>4</td>
<td>cpp (C preprocessor)</td>
</tr>
<tr>
<td>5</td>
<td>Assembly file (prog.s)</td>
</tr>
<tr>
<td>6</td>
<td>ccomp (C compiler)</td>
</tr>
<tr>
<td>7</td>
<td>Source file (prog.c)</td>
</tr>
<tr>
<td>8</td>
<td>as (Assembler)</td>
</tr>
<tr>
<td>9</td>
<td>ld (Linkage Editor)</td>
</tr>
</tbody>
</table>

 GCC: ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____ -> ____

Give an example of a **non-converting type cast** (underlying bit pattern does not change) [2pts]:

```
float f = 5.25;
int i = _____________________ ;
```

Fill in the body of the following Reduced-C function definition to initialize all the elements of array `myarr` to the value 123 using a foreach loop: [8pts]

```c
int myarr[100];
function : void init_myarr() {
}
```

In RC (and C/C++), we do not support the assignment of an entire array to another array (of the same type) using the assignment operator. However, we do support assignment of an entire struct instance to another struct instance of the same type. Using this fact, fill in the template of the code below, allowing arrays to piggy-back on a struct type to simulate entire-array assignments that are semantically and logically correct. [8pts]

```
struct def MYARR { int a[50]; };
int x[50];
int y[50];
function : void copy() {
    // x = y would be a semantic error, but....

    _____ _____________________ ______ x _______ = _____ _____________________ ______ y _______
}
```

Using the Right-Left rule (which follows the operator precedence rules) write the definition of a variable named `winter` that is a pointer to a pointer to a function that takes a pointer to a float as a single parameter and returns a pointer to a 2D array of 5 rows by 10 columns where each element is a pointer to a pointer to an short. [4pts]

```
________________________________________________________________________________;
```
Given the Reduced-C code shown in the box below, complete the SPARC Assembly language statements that might be emitted by a compliant Reduced-C compiler from this quarter for function main(). Include the necessary run-time checks. [24pts]

```c
/* Reduced-C */

function : void main()
{
    float * x;
    new x;
    cout << *x;
}
```

```assembly

.section _______
.align 4
.global _______

_____:  
save   %sp, -(92 + 4) & -8, %sp
! new x;
set    1    , %o0
set    ______, %o1
call   ______
nop
st    ______, ______
! *x
ld    ______, %o0
      .NullPtrChk
nop
! cout << *x
ld    ______, ______
call   ______
nop

      ______
      ______
      ______
      .NullPtrChk:
      save   %sp, -96, %sp
      ______
      ______
      ______
      .NullPtrOK
      nop
      ______
      ______
      ______
      .NullPtrMsg, ________  !! Assume .NullPtrMsg string already defined in ".data"
call   ______
nop
set    ______, ______
call   ______
nop

      .NullPtrOK:
      ret
      restore
```
Given the following C++ program (whose semantics in this case is similar to our Reduced-C) and a real compiler’s code gen as discussed in class, fill in the values stored in memory for each of the global and local variables and parameters in the run time environment for the SPARC architecture when the program reaches the comment /* HERE */. Do not add any unnecessary padding. [24pts]

```c
struct crazy {
    int a;
    int * b;
    int ** c;

crazy( int d ) {
    static int * magic;
    static int count;
    if ( count++ == 0 ) {
        magic = (int*) calloc(1, sizeof(int));
    }
    this->a = count;
    this->b = &count;
    this->c = &magic;
    *(this->c) = **(this->c) + d * 100;
}
~crazy() {
    if ( --*(this->b) == 0 ) {
        free( *(this->c) );
    }
}
};

void make_the_pain_stop( int i ) {
    struct crazy pain(++i);
    int hurts = **pain.c;
    struct crazy gain(++i);
    int *** insanity = &gain.c;

    /* HERE */
}

int main() {
    make_the_pain_stop(4);
    return 0;
}
```

**hypothetical decimal memory locations**

<table>
<thead>
<tr>
<th>low memory</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>magic: 2000</td>
<td>6000</td>
</tr>
<tr>
<td>count:</td>
<td></td>
</tr>
<tr>
<td>60300</td>
<td></td>
</tr>
</tbody>
</table>

%fp

<table>
<thead>
<tr>
<th>high memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>60400</td>
</tr>
</tbody>
</table>


8. Given the C array declaration
   
   ```c
   short a[3][5];
   ```

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

   a:  

   Each box represents a byte in memory. [4pts]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   low memory

   high memory

Suppose you have three source files with the variable declarations and definitions below:

<table>
<thead>
<tr>
<th>Module A</th>
<th>Module B</th>
<th>Module C</th>
</tr>
</thead>
<tbody>
<tr>
<td>extern int alpha;</td>
<td>int alpha;</td>
<td>extern int alpha;</td>
</tr>
<tr>
<td>static float beta;</td>
<td>extern float beta;</td>
<td>extern float beta;</td>
</tr>
<tr>
<td>static int * gamma;</td>
<td>static float gamma;</td>
<td>static int gamma;</td>
</tr>
<tr>
<td>int * delta;</td>
<td>static float delta;</td>
<td>extern int * delta;</td>
</tr>
</tbody>
</table>

An executable is desired from linking just these three modules. Assume that the `main()` function is properly defined in one of these modules. For each module, indicate with a Yes or No whether you would expect a linker error to occur if the specified variable is used in an expression somewhere in that module. [12pts]

<table>
<thead>
<tr>
<th>Module A</th>
<th>Module B</th>
<th>Module C</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>beta</td>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>gamma</td>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>delta</td>
<td>_______</td>
<td>_______</td>
</tr>
</tbody>
</table>

Use virtual register notation for each of the following.

Change the following instruction into two instructions which are most likely a time improvement over the single instruction when it comes to actual execution time. [5pts]

\[ r1 = 5 \times r2 \]

What term describes this particular kind of peephole optimization?

Optimize the following instructions into a single instruction which is most likely a time improvement when it comes to actual execution time. [3pts]

\[ r1 = r1 \times 1 \]
\[ r2 = r3 + r4 \]

What term describes this particular kind of peephole optimization?
9. Given the following C type definitions: [24pts]

struct bar {
  char a;
  int b;
  char c;
};

struct foo {
  struct bar d;
  char f;
};

struct fubar {
  char g[5];
  double e;
  char h;
  short i;
};

struct fubar fubaz;

Hint: Draw the memory layout of these structs (including padding) on the scratch paper at the end of this exam.

What is the sizeof( struct bar )? _____ What is the sizeof( struct foo )? _____

What is the sizeof( struct fubar )? _____ What is the offsetof( struct fubar, h.d.c )? _____

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

What is the sizeof( struct fubar ) if members of all 3 structs are reordered largest to smallest? _____

Given the following C++ code where ??? may represent different parameter passing modes, what values do you expect to be printed if the parameter passing mode is: [24pts]

```cpp
int x = 10;
int y = 30;

void baz(int & x, int & y) {
  int tmp = x;
  x = y;
  y = tmp;
}

void bar(int x, int ??? y) {
  cout << x++ << endl;
  cout << y-- << endl;
  baz(x, y);
  cout << x-- << endl;
  cout << y++ << endl;
  baz(x, y);
  cout << ::x << endl;
  cout << ::y << endl;
}

void foo(int x, int y) {
  cout << x << endl;
  cout << y << endl;
  baz(x, y);
  bar(x, y);
  cout << x << endl;
  cout << y << endl;
  baz(x, y);
  cout << ::x << endl;
  cout << ::y << endl;
}

int main() {
  baz(x, y);
  foo(x, y);
  return 0;
}
```

<table>
<thead>
<tr>
<th>call-by-value?</th>
<th>call-by-reference?</th>
</tr>
</thead>
<tbody>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
<tr>
<td>_______</td>
<td>_______</td>
</tr>
</tbody>
</table>

**call-by-value?**

**call-by-reference?**

```cpp
_________  
_________  
_________  
_________  
_________  
_________  
_________  
_________  
```

---

9
10. What gets printed when the following C program is executed? [18pts]

```c
#include <stdio.h>

int main() {
    char a[] = "PASSWORD";
    char *p = a;
    printf( "%c\n", *p++ = --a[7] );
    printf( "%c\n", (*++p = 5[a] - 2) + 6 );
    printf( "%c\n", p[4] = p[-1] + 4 );
    printf( "%c\n", *(p++ + 2) );
    printf( "%c\n", p[1] -= ((p - a) * 4 + 2) );
    printf( "%c\n", (p[4]=*p-1) - (a[3]=p[2]) + '.' );
    printf( "%c\n", (p[2] -= 3) - 23 );
    printf( "%c\n", (p[-2]=(*p++) - (12*(p-a) + 1) );
    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>L to R</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>R to L</td>
</tr>
<tr>
<td>-- prefix decrement</td>
<td>R to L</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>L to R</td>
</tr>
<tr>
<td>% modulus</td>
<td>L to R</td>
</tr>
<tr>
<td>+ addition</td>
<td>L to R</td>
</tr>
<tr>
<td>- subtraction</td>
<td>L to R</td>
</tr>
</tbody>
</table>

Hexadecimal - Character

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