Login Name _____ **Name** _____

Signature _____ **Student ID** _____

Midterm
CSE 131
Winter 2012

Page 1	_____	**(22 points)**
Page 2	_____	**(29 points)**
Page 3	_____	**(25 points)**
Page 4	_____	**(34 points)**
Page 5	_____	**(20 points)**
Page 6	_____	**(18 points)**
Subtotal	_____	**(148 points = 100%)**
Page 7	_____	**(9 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 AssignOp {: System.out.println("1"); :} Expr {: System.out.println("3"); :}
     |     Des {: System.out.println("5"); :}
     ;

Des  ::=   T_STAR {: System.out.println("7"); :} Des {: System.out.println("9"); :}
     |     T_AMPERSAND {: System.out.println("11"); :} Des {: System.out.println("13"); :}
     |     T_PLUSPLUS {: System.out.println("15"); :} Des {: System.out.println("17"); :}
     |     Des2 {: System.out.println("19"); :}
     ;

Des2 ::=   Des2 {: System.out.println("21"); :} T_PLUSPLUS {: System.out.println("23"); :}
     |     Des3 {: System.out.println("25"); :}
     ;

Des3 ::=   T_ID {: System.out.println("27"); :}
     ;

AssignOp ::=  T_ASSIGN {: System.out.println("29"); :}
     ;
```

What is the output when parsing the follow expression (you should have 20 lines/numbers in your output):

```
*++x = &*y++
```

Output

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

In the above grammar, what is the associativity of the operators in the second production rule (the Des ::= rule)? _____

If variable `y` is defined to be type `int *`, what type must variable `x` be defined for this expression to be semantically correct?

_____ `x;`

2. For each pair of Reduced-C expressions listed below (using the specs for Project I from this quarter), if the types of the two expressions are *equivalent* to each other, write the letter from the box below for **EQ**. If they are *not* equivalent, but they are *assignable* then write the letter from the box below for **L->R** if the left type is assignable to the right or the letter for **R->L** if the right type is assignable to the left. If they are neither equivalent nor assignable, write the letter for **None**.

```
structdef M { float[4] j; };
structdef N { float[4] j; };
typedef int* IPTR;
typedef int[4] IARR;
typedef N SDN;
IPTR ip;
IARR ia;
float * fp;
M m;
N n;
SDN sdn;
```

A) EQ
B) L->R
C) R->L
D) None

Equivalent/Assignable?

`ia[3]`	`ip[2]`	____	`*ip`	`sizeof(ip)`	____
`m`	`n`	____	`m.j`	`ip`	____
`fp[0]`	`*ip/4.2`	____	`m.j`	`fp`	____
`*fp`	`*ip`	____	`n`	`sdn`	____
`fp`	`ip`	____	`&n`	`&sdn`	____
`&fp`	`&ip`	____	`ia`	`ip`	____
`&fp`	`&m.j[0]`	____	`fp`	`ia`	____

Given the C array declaration

C

```
int a[4][2];
```

Mark with an **A** all the memory location(s) where we would find the array element

```
      a[2][1]
a:
```

low memory high memory

Each box represents a byte in memory.

Give the order of the typical C compilation stages and on to actual execution as discussed in class

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

gcc ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____

3. Given the following Reduced-C definitions:

```
function : float & foo( float & a ) { return a; }

float x;  /* global variables */
int y;
```

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.

```
 x = foo( 4.2 );            ____
 x = foo( y );              ____
 x = foo( x );              ____
 x = foo( foo( x ) );       ____
 y = foo( x );              ____
 x = foo( x + y );          ____
&x = &foo( x );             ____
 foo( x ) = foo( x );       ____
 foo( x ) = x;              ____
 foo( x ) = y;              ____
 (int) foo( x ) = y;        ____
 foo( x ) = &x;             ____
 foo( x ) = (float *) &y;   ____
```

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

Using the Right-Left rule (which follows the operator precedence rules) write the C definition of a variable named foo that is a pointer to an array of 8 elements where each element is a pointer to a function that takes a pointer to a pointer to a char as a single parameter and returns a pointer to an array of 4 elements where each element is a pointer to a pointer to a struct Fubaz. (8 points)

Using Reduced-C syntax, define a variable named fubar that is a pointer to an array of 39 floats. This should take two lines of code.

Now write a syntactically and semantically correct line of Reduced-C code to assign the last value in the array fubar points to into the variable f below (assume the variable f has been properly defined to be of type float and memory allocated for the array and fubar has been properly assigned to point to this array).

```
f = _____ ;
```

4. 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.

```
structdef FOO1 {
  FOO1 *  a;
  float   b;
  function : void bar( FOO1 &x)
  {
    int[10] y;
  }
  int[2]  c;
  int     d;
};
```

Size _____

```
structdef FOO2 {
  int  * a;
  float  b;
  function : void bar()
  {
    int x = *this.a;
  }
  FOO2 * c;
  int  * d;
};
```

Size _____

```
structdef FOO3 {
  int    a;
  float  b;
  function : void bar()
  {
    FOO3 *x;
  }
  int    c;
  int[3] d;
};
```

Size _____

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

1) Yes – Underlying bit pattern change
2) No – No underlying bit pattern change

```
int a = 5;
float b = -4.20;
int * ptr1;
float * ptr2;
void foo( float x, float & y ) { /* … */ }
```

`b = a;`	_____	`a = * (int *) & b;`	_____
`ptr1 = (int *) & b;`	_____	`ptr2 = (float *) ptr1;`	_____
`a = (int) b;`	_____	`foo( a, b );`	_____

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

```
function : int * foo1() { /* Function body not important. */ }
function : int & foo2() { /* Function body not important. */ }
float[9] a;
float x;
const float y = 5.5;
float *p = &x;
```

____ `foo2()`	____ `4.2`	____ `*foo1()`	____ `x = y`	____ `*(int *)p`
____ `p`	____ `a[2]`	____ `(int *)&x`	____ `*(int *)&x`	____ `(float *)foo1()`
____ `x`	____ `*p`	____ `**&p`	____ `y`	____ `foo2() * y`
____ `&x`	____ `a`	____ `&*p`	____ `*p - y`	____ `foo2`
____ `++x`	____ `&a[0]`	____ `a[2]++`	____ `foo1()`	____ `++foo2()`

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. Circle the page number below for an extra point. Structs and unions are padded so the total size is evenly divisible by the strictest alignment requirement of its members.

```
struct foo {
  char   a;
  short  b;
  double c;
  int    d;
};

struct fubar {
  short      e;
  struct foo f;
  char       g[9];
  short      h;
};

struct fubar fubaz;
```

low memory

fubaz:

high memory

What is the `sizeof( struct fubar )`? _____ What is the `offsetof( struct fubar, g[1] )`? _____

What is the alignment restriction for struct foo? _____ What is the alignment restriction for struct fubar? _____

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

What is the resulting type of the following expression?

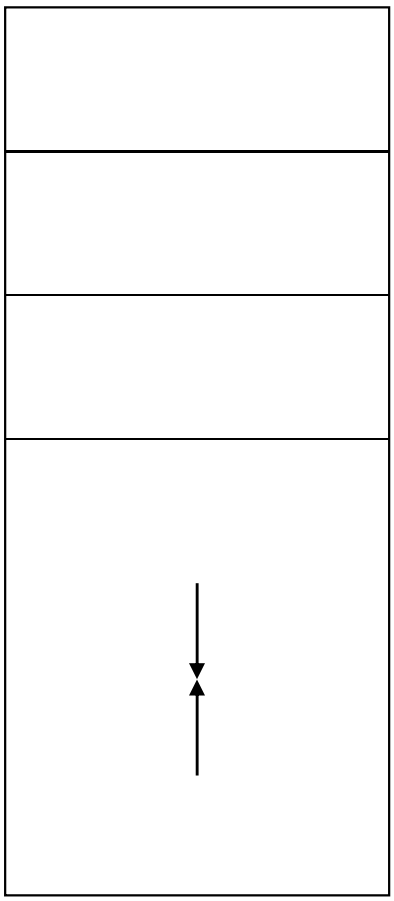
```
* (char *) & ( ( (struct fubar *) & fubaz.f ) -> e ) _________________
```

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



high memory

The following RC input program contains ***FOUR*** erroneous statements. Indicate which lines are erroneous, and what the error would be from the list of available error messages we provided below.

```
01:  int x = 123;
02:  const int y = 420;
03:  const float z = x + y;
04:  const float w = y * y;
05:  int * v = NULL;
06:  function : int main() {
07:      return v[-50];
08:      return z;
09:      v = &(-x);
10:      return sizeof(int[y]);
11:      return sizeof(NULL);
12:      *&*&*&*&y = *&*&*&x;
13:      return *v;
14:  }
```

List of types of errors:
A: An error that wouldn't be tested (WNBT)
B: Incompatible type to binary operator
C: Incompatible type to unary operator
D: Left-hand operand is not assignable (not a modifiable L-value)
E: Value not assignable to variable's type
F: Type of return expression not assignment compatible with function's return type
G: Type of return expression not equivalent to the function's return type
H: Return expression is not a modifiable L-value
I: Initialization value of constant not known at compile-time
J: Initialization value not assignable to constant/variable
K: Index expression value in array declaration must be > 0
L: Index value is outside legal range
M: Invalid operand to sizeof. Not a type or not addressable
N: Non-addressable argument to address-of operator

Error #1: Line # _____ Error Type _____

Error #2: Line # _____ Error Type _____

Error #3: Line # _____ Error Type _____

Error #4: Line # _____ Error Type _____

Extra Credit

What gets printed when the following C program is executed?

```
#include <stdio.h>

int
main()
{
  char a[] = "34589";
  char *p = a;

  printf( "%c\n", *p++ );            ______

  printf( "%c\n", ++*p );            ______

  printf( "%c\n", *p = p[-1] + 1 );  ______

  printf( "%c\n", ++*p++ );          ______

  printf( "%c\n", ++*p );            ______

  printf( "%c\n", --*++p );          ______

  printf( "%c\n", ++*(p+1)  );       ______

  printf( "%d\n", p - a );           ______

  printf( "%s\n", a );               ______________________

  return 0;
}
```

A portion of the C Operator Precedence Table

Operator	Associativity
++ postfix increment	L to R
-- postfix decrement	
[] array element	
() function call	
* indirection	R to L
++ prefix increment	
-- prefix decrement	
& address-of	
sizeof size of type/object	
(type) type cast	
* multiplication	L to R
/ division	
% modulus	
+ addition	L to R
- subtraction	
.	
.	
= assignment	R to L

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	NP	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	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3A	:	3B	;	3C	<	3D	=	3E	>	3F	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4A	J	4B	K	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

Scratch Paper

Scratch Paper