Login Name _____ **Name _____**

Signature _____ **Student ID _____**

Midterm
CSE 131
Fall 2014

Page 1	**_____ (35 points)**
Page 2	**_____ (30 points)**
Page 3	**_____ (24 points)**
Page 4	**_____ (24 points)**
Page 5	**_____ (32 points)**
Subtotal	**_____ (145 points = 100%)**
Page 6 Extra Credit	**_____ (16 points)**
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"); :}
     ;

Des  ::= T_STAR {: System.out.println("D"); :} Des {: System.out.println("E"); :}
     | T_SIZEOF {: System.out.println("F"); :} T_LPAREN Des T_RPAREN {: System.out.println("G"); :}
     | T_PLUSPLUS {: 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"); :}
     | Des2 {: System.out.println("M"); :} T_LBRACKET Des T_RBRACKET {: System.out.println("N"); :}
     | Des3
     ;

Des3 ::= T_ID
     ;
```

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

```
q[ sizeof(r) ] = r[s] = ++*t++;
```

Output

____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____
____

In the above grammar, which has higher precedence: pointer dereference (i.e. `*`) or array dereference (i.e. `[]`), or do they have the same precedence? [2 pts]

_____

In the above grammar, does the pointer dereference operator (i.e. `*`) have left-to-right or right-to-left associativity? [2 pts]

_____

If variable `r` is defined to be type `float[5]`, what type must variables `q`, `s` and `t` be defined for this expression to be semantically correct? Define these variables to the minimum semantically-correct size using RC syntax. [9 pts]

```
float[5]          r;

_____________________ q;

_____________________ s;

_____________________ t;
```

2. Give the order of the typical GCC compilation stages and on to actual execution as discussed in class: [5 pts]

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

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

Give the order of the general phases of compilation in a typical compiler as discussed in class: [5pts]

A – Scanner (Lexical analysis)
B – Parser (Semantic analysis/intermediate code gen.)
C – Machine-independent code improvement (optional)
D – Machine-specific code improvement (optional)
E – Target language (e.g. assembly)
F – Target code generation
G – Parser (Syntax analysis)
H – Source language (e.g. C)

_____ –> _____ –> _____ –> _____ –> _____ –> _____ –> _____ –> _____

Given the C array declaration

C

```
char * a[2][3];
```

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

a:



low memory high memory

(Each box represents a byte in memory)

Given the following Reduced-C code

```
structdef S1 { bool a; };
structdef S2 { bool a; };
typedef S1 S3;
typedef S3 S4;
S1 a, b;
S2 c, d;
S3 e;
S4 f;
```

identify each of the following statements as either: [15 pts]

(A) No Error (B) Compile-Time Error

____ `a = b;`	____ `c = d;`	____ `a = (S1) c;`
____ `a = c;`	____ `c = e;`	____ `a = * (S1*) &c;`
____ `a = d;`	____ `c = f;`	____ `(S2) b = d;`
____ `a = e;`	____ `e = f;`	____ `* (S2*) &b = d;`
____ `a = f;`	____ `a.a == b.a;`	____ `f = e = b = a;`

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. [14 pts figure; 10 pts questions]

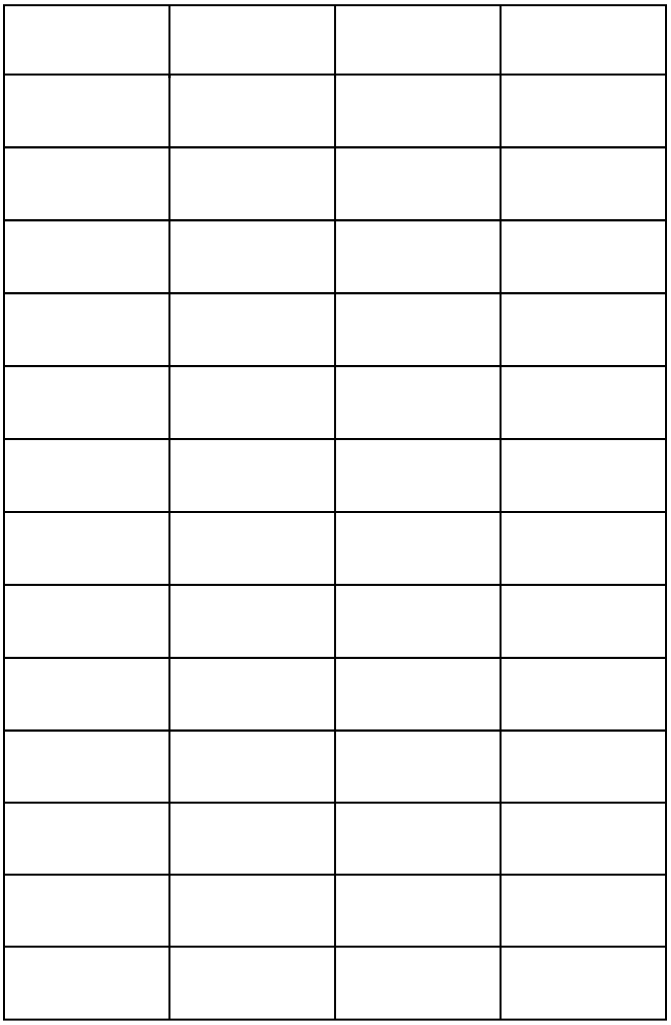
```
struct foo {
  char       a;
  double *   b;
};

struct bar {
  char       c;
  double     d;
  char       e[5];
  struct foo f;
  short      g;
};

struct bar baz;
```

low memory

baz:



high memory

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

What is the `offsetof( struct bar, f.b )`? _____

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? _____

4. Given the following pseudocode, determine the program output based on the specified scoping rule: [24 pts]

```
int x;            -- global var declaration
int y;            -- global var declaration

void set_xy( int a, int b )
      x = a;
      y = b;

void beta()
      int y = 7;  -- local var declaration
      cout << x;
      cout << y;
      set_xy( 5, 5 );
      cout << x;
      cout << y;

void alpha()
      int x = 2;  -- local var declaration
      cout << x;
      cout << y;
      set_xy( 4, 4 );
      cout << x;
      cout << y;
      beta();
      cout << x;
      cout << y;


set_xy( 0, 0 );
alpha();
cout << x;
cout << y;
```

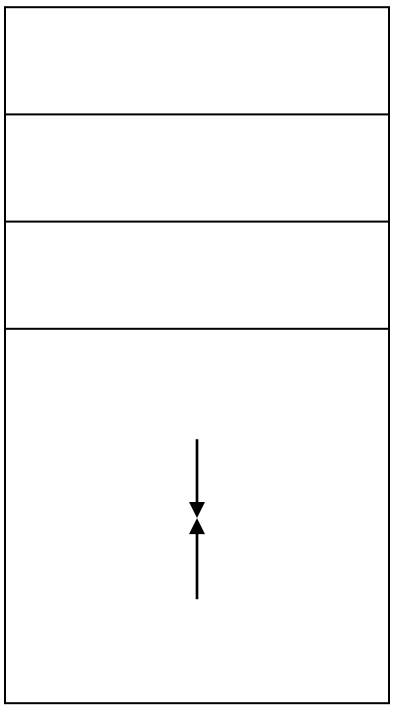
What does the program output if the language uses <u>static</u> scoping?

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

What does the program output if the language uses <u>dynamic</u> scoping?

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

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]



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]

```
int A, B;
static int C, D = 1;

int * E ( int F ) {
    static float G;
    int H = 3;
    int * I = (int *) malloc(sizeof(int));

    return I;
}

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

A	_____	J	_____
B	_____	K	_____
C	_____	L	_____
D	_____	main	_____
E	_____	argc	_____
F	_____	argv	_____
G	_____	*J	_____
H	_____	*K	_____
I	_____	*L	_____

For the above program, while there are no compile-time errors, there is a serious programming mistake that can cause run-time crashes. Explain what the mistake is: [4 pts]

Extra Credit

What gets printed when the following C program is executed? [16 pts]

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

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

  printf( "%c\n", (*p++ = 1[a] + 5) - 1 );                 ______

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

  printf( "%c\n", (*p = *(p + 1) = (*p - 7)) + 10 );       ______

  printf( "%c\n", *--p = (*a + 5) );                       ______

  printf( "%c\n", *p + 7 );                                ______

  printf( "%c\n", *(p - 2) = *(p + 3) = '!' );             ______

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

  return 0;
}
```

A portion of the C Operator Precedence Table

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