Login Name _____ **Name _____**

Signature _____ **Student ID _____**

Midterm
CSE 131
Fall 2015

Page 1	**_____ (30 points)**
Page 2	**_____ (26 points)**
Page 3	**_____ (24 points)**
Page 4	**_____ (20 points)**
Page 5	**_____ (28 points)**
Subtotal	**_____ (128 points = 100%)**
Page 6 Extra Credit	**_____ (8 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 ::= Expr {: System.out.println("A"); :} T_STAR Des {: System.out.println("B"); :}
    | Des
    ;

Des  ::= T_STAR {: System.out.println("C"); :} Des {: System.out.println("D"); :}
    | T_PLUSPLUS {: System.out.println("E"); :} Des {: System.out.println("F"); :}
    | T_AMPERSAND {: System.out.println("G"); :} Des {: System.out.println("H"); :}
    | Des2
    ;

Des2 ::= Des2 {: System.out.println("I"); :} T_DOT T_ID {: System.out.println("J"); :}
    | Des2 {: System.out.println("K"); :} T_PLUSPLUS {: System.out.println("L"); :}
    | Des2 {: System.out.println("M"); :} T_LBRACKET Expr T_RBRACKET {: System.out.println("N"); :}
    | Des3
    ;

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

What is the output when parsing the follow expression (you should have 23 lines/letters in your output): [23pts]

```
**a * *b[c*c] * **d
```

Output

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

In the above grammar, does the multiplication (`*`) operator have left-to-right or right-to-left associativity? [2pts]

_____

What type must variables `a`, `b`, `c`, and `d` be defined for the above expression to be semantically correct and result in a type **`int`**? **Do not** define these variables using arrays. [5pts]

_____ a;

_____ b;

_____ c;

_____ d;

2. Given the following pseudocode, determine the program output based on the specified scoping rule: [26pts]

```
int x = 0;        -- global var declaration
int y = 0;        -- global var declaration


int deimos( int dread ) {
      int y = dread + 10;      -- local var declaration
      if ( dread > 0 ) {
            x += deimos(dread – 1);
      }
      cout << x << endl;
      cout << y << endl;
      return y;
}

int phobos ( int fear ) {
      int x = fear + 5;        -- local var declaration
      if ( fear > 0 ) {
            y += phobos(fear – 2);
      }
      cout << x << endl;
      cout << y << endl;
      int z = x + y;           -- local var declaration
      return z + deimos(x % 2);
}

cout << phobos(2) << endl;
```

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?

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

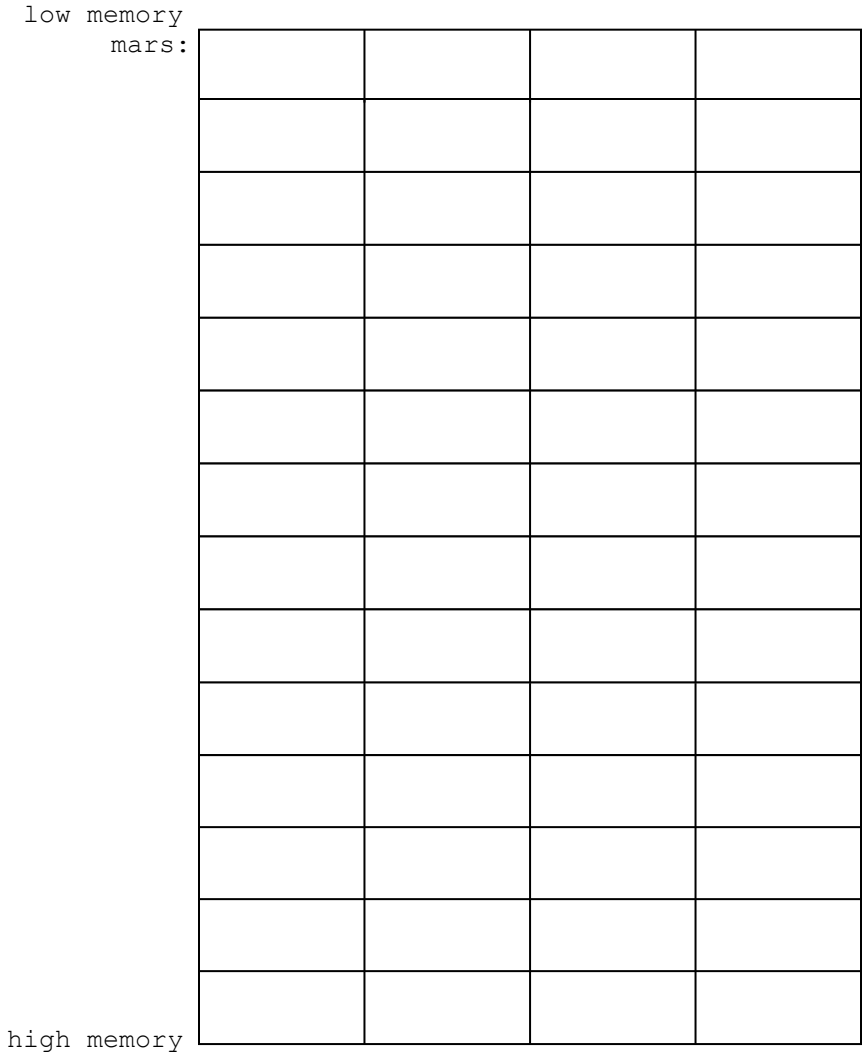
_____

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.). If the member/field is an array of structs, include both the index and member name (e.g. `p[0].a`). 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. [14pts figure; 10pts questions]

```
struct sojourner {
  double              h;
  char                i;
};

struct pathfinder {
  char                m;
  struct sojourner    a;
  struct sojourner * r[3];
  short               k;
};

struct pathfinder mars;
```



What is the `sizeof( mars )`? _____

What is the `offsetof( struct pathfinder, r[2] )`? _____

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

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

How many bytes are saved by rearranging the `struct pathfinder` members to minimize padding? _____

4. Given the following C++ program and taking into consideration how struct constructors and destructors are called as discussed in class, determine the output generated by this program: [16pts]

```
#include <iostream>

using namespace std;

struct MARK {
    int id;

    MARK(int id) {
        this->id = id;
        cout << "MARK CTOR: " << this->id << endl;
    }

    ~MARK() {
        cout << "MARK DTOR: " << this->id << endl;
    }
};

struct MARK Watney1(0);
struct MARK * Watney2[4];

int main() {
    cout << "MAIN: 1" << endl;
    for ( int i = 0; i < 3; ++i ) {
        struct MARK Watney3(3);
        if ( i % 2 ) {
            struct MARK Watney4(4);
            Watney2[i] = new struct MARK(20 + i);
        }
        if ( Watney2[i] != NULL ) {
            delete Watney2[i];
            struct MARK Watney5(5);
        }
    }
    cout << "MAIN: 2" << endl;
    return 0;
}
```

Output

Handling array comparison and assignment in C:

In C, one should call the library function _____ in order to compare if two arrays of arbitrary data type are equal or not. [2pts]

In C, one should call the library function _____ in order to assign the contents of one array of arbitrary data type into another array of that same data type and size, where the two arrays may or may not overlap memory addresses. [2pts]

5. Given the following C code, indicate which of the 5 main areas of the C Runtime Environment each expression is located in: [14pts]

```
int A = 5;
int * B;
int * C;

void D ( int E ) {
    int F;
    static int G = 2;
    B = &G;
    return;
    B = &F;
}

int main ( int argc, char *argv[] ) {
    C = (int *) calloc(1, sizeof(int));
    int ** H = &C;
    D(360);
    <------ determine runtime areas at this point in time
    free(C);
    C = NULL;
    return 0;
}
```

A	_____	F	_____
B	_____	G	_____
*B	_____	H	_____
C	_____	*H	_____
*C	_____	**H	_____
D	_____	argc	_____
E	_____	argv	_____

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

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

State whether constant folding will be performed on one or more arithmetic expressions in the following statements by the compiler according to this quarter's Reduced-C spec (Y or N): [10pts]

```
function : void main() {
    const int a = -1;

    int b = 2;

    int c = a + a;              _______

    int d[30];                  _______

    c = a + 2;                  _______

    c = 1 + 2;                  _______

    d[b + c + 4] = a;           _______

    d[-5 + (2 / c)]++;          _______

    d[(-5 + 2) / c]++;          _______

    d[-a];                      _______

    d[a * a] = a;               _______

    d[b] = a ^ a | a;           _______
}
```

Extra Credit

What gets printed when the following C program is executed? [8pts]

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

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

  printf( "%c\n", p[3] -= 15 );                                       ______

  printf( "%c\n", ((++p)[1] = a[1] - 2) - 1 );                        ______

  printf( "%c\n", (p[3] -= 43) + 44 );                                ______

  printf( "%c\n", ( *p-- = a[3] + (a[0] - a[4]) + 2 ) ? a[3] : a[4] ); ______

  printf( "%c\n", *p = 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