CS61, Fall 2012
Midterm Review Section
(10/16/2012)

Q1: Hexadecimal and Binary Notation

- Solve the following equations and put your answers in hex, decimal and binary.

				Hexadecimal	Decimal	Binary
15	+	0x15	=	_____	_____	_____
99	+	0xAA	=	_____	_____	_____
11101	+	0xF1	=	_____	_____	_____
15	+	0b1010	=	_____	_____	_____

Q2: Address Spaces

- Given the following word sizes, find the largest possible representable memory address in both Little and Big Endian

	Largest Memory Address (Hexadecimal)	
Word Size	Little Endian	Big Endian
2 bytes	_____	_____
17 bits	_____	_____
_____	_____	0x7f ff ff
_____	0xff07	_____

Q3: Signed Arithmetic

- All numbers should fit into 6-bits and be expressed in 2's complement

			Binary	Decimal
0b010101 +	0b010101	=	_____	_____
0b100001 +	0b010101	=	_____	_____

0b101101 + 0b011111 = _____ _____

0b101110 + 0b100111 = _____ _____

Q4: Bitwise Operators

Check if x is even:

```
bool is_odd = x ___ ___
```

Check if *x* is a power of two:

```
bool is_pow2 = 0 ___ ( x ___ (x - 1) )
```

Get the absolute value of *x*:

```
int x;
int mask = x ___ 31;
abs_x = (x + mask) __ mask;
```

Swap the values of *x* and *y*:

```
int x, y;
x = x ___ y;
y = y ___ x;
x = x ___ y;
```

Q5: Struct Size

- Assume a 32-bit machine

```
struct my_struct
{
        char a,b;
        char *c;
        char d[6];
};

struct my_struct list[5];
struct my_struct *ptr = malloc(sizeof(struct my_struct));
```

5.1
What is sizeof(my_struct)?

5.2
How could we reduce this struct's size?

5.3
Suppose **list** lives at memory address 0x100. What is at memory address...
i) 0x111

ii) 0x123

iii) 0x14c

5.4
Suppose ptr = 0x1337. What is the address of...
i) ptr + 3

ii) (int *) (((char *) ptr) + 5)

Q6: Memory Bugs

- Find the problem in the following code that may yield unexpected results.

6.1

```
int* Hamlet() {
	int Ophelia = 42;
	return &Ophelia;
}

void main() {
	int *Horatio = Hamlet();
	printf("The answer is %d\n", *Horatio);
}
```

6.2

```
typedef struct {
	double Jaques;
	int Pages[10];
	char *Duke;
} ExiledCourt;

void main() {
	ExiledCourt *Rosalind;
```

```
        int counter = 0;
        Rosalind = (ExiledCourt*) malloc(sizeof(ExiledCourt));

        for (counter = 0; counter < 10;counter++) {
                Rosalind->Pages[counter] = 0x2a;
        }

        strcpy(Rosalind->Duke, “As You Like It”);
}
```

6.3 - Gangnam Style

Somebody stole Psy’s Gangnam style! Can you help him track down the thief? To find the culprit, Psy needs a piece of software that will copy a CIA list of known criminals (in the form of a singly-linked list of person structs) to his PDA and search the list for suspects living in Seoul. Unfortunately, programmer who developed this software never took CS61 so he left lots of bugs. Find them all to help Psy finally recover his GANGNAM STYLE!

```
1       typdef struct person_struct {
2               char name[20];
3               int age;
4               char *city;
5               struct person_struct *next;
6       } person;
7
8       person *search_suspects (person* first_person, int n) {
9               //Allocate new memory, copy list of people.10
11              person *new_first = malloc(n*sizeof(person));
12              person *new_current_person = NULL;
13              person *current_person = first_person;
14              int i = 0;
15              for (i = 0; i <= n; i++) {
16                      new_current_person = new_first + i*sizeof(person);
17                      strcpy(new_current_person->name, current_person->name);
18                      new_current_person->age = current_person->age;
19                      new_current_person->city = malloc(strlen(current_person->city));
20                      strcpy(*new_current_person->city, *current_person->city);
21                      if(current_person->next == NULL) {
22                              new_current_person->next = NULL;
23                      } else {
24                              new_current_person->next =new_current_person+sizeof(person);
25                              current_person = current_person->next;
26                      }
27              }
28
29              //Search new list for people in Los Angeles.
30
31              person *last_person = NULL;
32              new_current_person = new_first;
33              do {
```

```
34                      if(strcmp(new_current_person->city, "Los Angeles") == 0) {
35                              strcat(new_current_person->name, " (POSSIBLE SUSPECT!)");
36                              if(last_person == NULL) {
37                                      //This is the new head of the linked list
38                                      new_first = new_current_person;
39                              } else {
40                                      //Update pointer in the last node.
41                                      last_person->next = new_current_person;
42                              }
43                              last_person = new_current_person;
44                      } else {
45                              //Free this struct.
46                              free(new_current_person);
47                      }
48                      new_current_person = new_current_person->next;
49              } while(new_current_person != NULL);
50
51              return new_first;
52      }
```

Q7: Garbage Collection

7.1
What are some common memory bugs that garbage collection solves?

- Double free
- Invalid free
- Wild writes (off-the-end writes) (NOT C garbage collection)
- etc.

7.2 Which of the memory bugs in the code above would garbage collection have solved?

Q8: Basic Assembly

Consider the following assembly code:
x at %ebp+8, n at %ebp+12

```
1       movl 8(%ebp), %esi
2       movl 12(%ebp), %ebx
3       movl $-1, %edi
4       movl $1, %edx
5       .L2:
```

```
6       movl %edx, %eax
7       andl %esi, %eax
8       xorl %eax, %edi
9       movl %ebx, %ecx
10      sall %cl, %edx
11      testl %edx, %edx
12      jne .L2
13      movl %edi, %eax
```

The preceding code was generated by compiling C code that had the following overall form:

```
1       int loop(int x, int n)
2       {
3               int result = _____;
4               int mask;
5               for (mask = _____; mask _____; mask = _____) {
6                       result ^= _____;
7               }
8               return result;
9       }
```

a. Which registers hold program values x, n, result, and mask?

b. What are the initial values of result and mask?

c. What is the test condition for mask?

d. How does mask get updated?

e. How does result get updated?

f. Fill in all the missing parts of the C code.

g. Why are the arguments to loop pushed on the stack in reverse order (i.e., x ends up closer to loop's %ebp than n)? Why can't we do it the other way around?

Q9: Procedure Calls

- Let's say we are given the following assembly code for a function:

```
1       pushl %edi
2       pushl %esi
```

```
3       pushl %ebx
4       subl $0x24, %esp
5       movl 24(%ebp), %eax
6       imull 16(%ebp), %eax
7       movl 24(%ebp), %ebx
8       leal 0(, %eax, 4), %ecx
9       addl 8(%ebp), %ecx
10      movl %ebx, %edx
11      subl 12(%ebp), %edx
…...
20      popl %ebx
21      popl %esi
22      popl %edi
```

9.1
Why are %edi, %esi, and %ebx pushed onto the stack at the beginning of this function and popped off at the end?

9.2
What about %eax, %edx, and %ecx? Why aren't they put on the stack?

9.3
What do 24(%ebp) and 16(%ebp) refer to?

9.4
Why do we subtract 0x24 from %esp? What might be put in that area?

Q10: Flags

-For each one of the following, determine which flags are set by the add instruction and why.

10.1

```
movl $0x40, %eax
movl $0xffffffc0, %ebx
addl %eax, %ebx
```

10.2
```
movl $0x2a, %eax
movl $0xfffffffc0, %ebx
addl %eax, %ebx
```

10.3
```
movl $0x7FFFFFF0, %eax
movl $0x2c, %ebx
addl %eax, %ebx
```

Q10: Tail Recursive Optimizations

```
int main(void) {
        printf( " %d\n ",  getSum( 5 ) )
        return 0;
}
int getSum(int num) {
        if ( num == 1 )
                return 1;
        else
                return getSum(num-1) + num;
}
```

10.1
What does the following code do?

10.2
Why is this code inefficient?

10.3
Rewrite this function so it has the same behavior but performs more efficiently.

Q11: Buffer Overflow

- Consider the following function

```
struct client {
    char name[9];
    int money;
};

struct client joinBank ( int initialDeposit );

int main (void) {

    int joiningReward = 50;
    struct client newClient = joinBank( joiningReward );

    printf("%d\n", newClient.money);
    saveClientToDataBase(newClient);
}

struct client joinBank( int joiningReward ) {
   struct client newClient;

   newClient.money = joiningReward;
   printf("What is the new client's name?\n");
   gets(newClient.name);

   return newClient;
}

void deleteAllBankUsers() {
    // deletes records of all bank clients!
}
```

12.1
What input would you input in order to instantly make $1,000,000?

12.2
Suppose the instruction for deleteAllBankUsers() was on line 0x0806578c. What could a malicious user input to delete the records of all the bank clients?