UNIVERSITY OF CALIFORNIA, LOS ANGELES UCLA

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO SANTA BARBARA • SANTA CRUZ

CS 33

Midterm #2

All answers must be written on the answer sheet (last page of the exam).

All work should be written directly on the exam, use the backs of pages if needed.

This is an open book, open notes quiz – but you cannot share books or notes. An ASCII table is on the second to last page if you need it.

I will follow the guidelines of the university in reporting academic misconduct – please do not cheat.

NAME: __ ________________________

ID: __ ________________________

Problem 1: 30

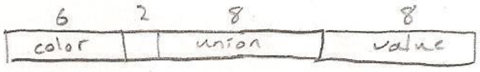
Problem 2: 15 → e.c.

Problem 3: 36

Total: 66/85 +15

1. ***Structured Play (30 points):*** Consider the following declaration:

```
struct node_t {
  char color[6];   6
  union {
    int numeric;    }  8
    char label[8];  }
  } identifier;
  long value; 8
} a[4];
```



Answer the following questions on how this data structure would be laid out on a 64-bit Linux machine:

a) Considering alignment – how much total space (in bytes) would this data structure require?

96

b) If the base address of array *a* is 0x600a60, what would be the string stored in *a[1].color*? Make use of the following gdb output:

6d,61,75,76,65,0a
m a u v e

```
(gdb) x/24x 0x600a60
0x600a60:       0x68636f6d      0x00000a61      0x00000400      0x00000000
0x600a70:       0x46e87ccd      0x00000000      0x7675616d      0x00000a65
0x600a80:       0x00000800      0x00000000      0x3d1b58ba      0x00000000
0x600a90:       0x72757a61      0x00000a65      0x00001000      0x00000000
0x600aa0:       0x507ed7ab      0x00000000      0x7268636f      0x00000a65
0x600ab0:       0x00000200      0x00000000      0x2eb141f2      0x00000000
```

Hint – don't forget that gdb reverses byte ordering within each 4-byte chunk. So in the following dump:

```
(gdb) x/4x 0x00111110
0x111110:    0x33221100   0x77665544   0xBBAA9988   0xFFEEDDCC
```

This prints out 16 bytes of memory starting at address 0x111110. In this example, the 16 bytes of memory starting at 0x111110 would contain, in order from lowest address (0x111110) to highest address (0x11111F):

00112233445566778899AABBCCDDEEFF

So address 0x111110 contains the byte 0x00, address 0x111111 contains the byte 0x11, address 0x111112 contains the byte 0x22, and so on. So in terms of just the least significant hex place of the address, gdb is actually printing out addresses in the following order:

3210 7654 BA98 FEDC

This is useful when reading words, but can be confusing for other values.

2. ***Complete Dis-Array (30 points)***: Consider the following C fragment:

```
#define SIZE 10

int orange[SIZE][SIZE];
int *green;
int *purple[SIZE];
int n;

int main( int argc, const char* argv[] ) {
...
```

The code in …'s will create three two dimensional arrays: orange, purple, and green. Arrays orange and purple are statically sized (i.e. size is known at compile time). Array orange is a nested array and array purple is a multi-level array. Array green is dynamically sized (i.e. a dynamically nested array).

Each array has a function that sets the value of one element of the array. These functions have three parameters – the row position in the array (i), the column position in the array (j), and the value to set (val). They have the following prototypes:

```
                      row   col       value
void setorange(int i, int j, int val);
void setgreen(int i, int j, int val);
void setpurple(int i, int j, int val);
```

For example, *setorange(i,j,val)* will set *orange[i][j]* to the value *val*.

Based on the information above, answer the following questions:

a) Which of these three functions (setorange, setgreen, or [✓ setpurple]) is shown disassembled below?

```
8048450:     55                      push   %ebp
8048451:     8b 15 d4 9a 04 08       mov    0x8049ad4,%edx
8048457:     89 e5                   mov    %esp,%ebp
8048459:     8b 45 08                mov    0x8(%ebp),%eax i
804845c:     0f af 05 d0 9a 04 08    imul   0x8049ad0,%eax
8048463:     8b 4d 10                mov    0x10(%ebp),%ecx val
8048466:     03 45 0c                add    0xc(%ebp),%eax j
8048469:     89 0c 82                mov    %ecx,(%edx,%eax,4)
804846c:     5d                      pop    %ebp
804846d:     c3                      ret
```

b) In the disassembled code above, what variable is located at address 0x8049ad0?

n

1011
0101
0010
0001

3. ***We Want the Func (40 points):*** Consider the following C code:

$$\begin{tikzcd}
\text{foo}(11,0) \arrow[r] & \text{foo}(5,1)+1 \arrow[d] \\
 & \text{foo}(2,2)+1 \arrow[d] \\
 & \text{foo}(1,3)+0 \arrow[d] \\
 & \text{foo}(0,4)+1
\end{tikzcd}$$

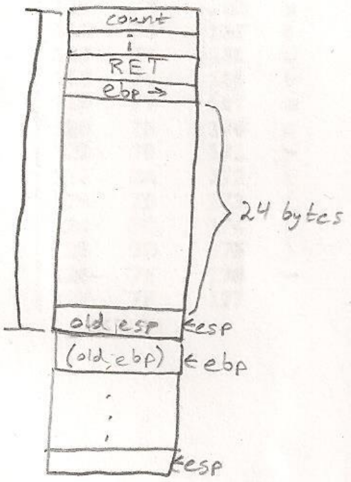
```
int foo(int i, int count) {

  if (count==32)
    return 0;
  else
    return (foo(i>>1,count+1)+(i&1));
}

int main( int argc, const char* argv[] ) {
  fprintf(stderr, "%d\n", foo(atoi(argv[1]), 0));
}
```

a) If this code were compiled, and then executed with the value 11 as a parameter, what would it return?

3

b) Consider the following disassembled code for function foo:

```
08048414 <foo>:
 8048414:       push    %ebp
 8048415:       mov     %esp,%ebp
 8048417:       sub     $0x18,%esp
                                count
 804841a:       cmpl    $0x20,__A__(%ebp)
 804841e:       jne     8048427 <foo+0x13>
 8048420:       mov     $0x0,%eax
 8048425:       jmp     8048446 <foo+0x32>
                              count
 8048427:       mov     __B__(%ebp),%eax
 804842a:       lea     0x1(%eax),%edx count
 804842d:       mov     __C__ i (%ebp),%eax
 8048430:       sar     %eax                         foo
 8048432:       mov count %edx,__D__(%esp)           call
 8048436:       mov   i %eax,__E__(%esp)
 8048439:       call    8048414 <foo>
 804843e:       mov     __F__(%ebp),%edx
 8048441:       and     $0x1,%edx
 8048444:       add     %edx,%eax
 8048446:       leave
 8048447:       ret
```



There are six blanks in the code above – labeled A-F – and all are related to displacements relative to either %esp or %ebp. Fill in the blanks with the appropriate values to make this code work correctly. You should fill in a ***hexadecimal number*** for each blank – it should ***not*** be a register specifier. These are displacements relative to the stack or frame pointers.

A 0xc
B 0xc
C 0x8
D 0x8
E 0x4
F 0x8

Answer Sheet

Name:__ ________

1. a. 96 bytes (24 bytes per element of the array) 15

   b. "mauve\n" 15

2. a. setpurple

   b. n 15 ec

3. a. 3 16

   b. Fill in all blanks below

0xc 4

A

0xc 4

B

0x8 4

C

0x8 2

D

0x4 2

E

0x8 4

F

∞