CS 33, Summer 2020, with Prof. Ghaforyfard

Week 5 Worksheet

**1.** What will the following print out?

typedef struct {

char shookie;

int tata;

char cookie;

double chimmy;

} bt;

void main(int argc, char\*\* argv) {

bt band[7];

printf( “%d\n”, (int)sizeof(band));

}

**2.** What is the best\* ordering of the following variables if you want to have a struct that uses all of them? Assume a 64-bit architecture with 4-byte ints.

*\* the ordering that will result in the optimal usage of space.*

char tully;

long stark;

float\* lannister;

double targaryen;

int greyjoy;

float arryn;

struct Westeros{

};

**3.** How many bytes of space would these declarations require?

char vermilion[9];

char\* amber[9];

char\*\* chartreuse[9];

int teal[6];

int violet[6][3];

int\* magenta[6][3];

**4.** Consider the following struct:

typedef struct {

char first;

int second;

short third;

} stuff;

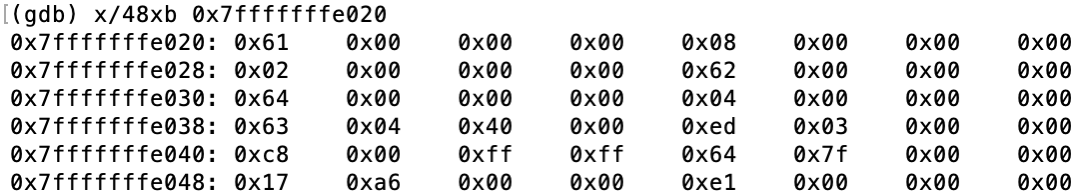
Say we are debugging an application in execution using gdb on a 64-bit, little-endian architecture. The application has a variable called array - defined as:

stuff array[2][2];

Using gdb we find the following information at a particular stage in the application:



And:



What is the value of **array[1][0].second** at this particular stage of the application?

i.e. what would be returned from the statement: printf("%d\n", array[1][0].second);

**5.** Consider the following C code:

typedef struct {

char first;

int second;

short third;

int\* fourth;

} stuff;

stuff array[5];

int func0(int index, int pos, long dist) {

char\* ptr = (char\*) &(array[index].\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

ptr += pos;

\*ptr = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ + dist;

return \*ptr;

}

int func1() {

int x = func0(1, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

return x;

}

Clearly some code is missing - your job is to fill in the blanks! Note that the size of the blanks is not significant. The two functions will be compiled using the following assembly code:

0000000000400492 <func0>:

400492: 8d 04 17 lea (%rdi,%rdx,1),%eax

400495: 48 63 ff movslq %edi,%rdi

400498: 48 63 f6 movslq %esi,%rsi

40049b: 48 8d 14 7f lea (%rdi,%rdi,2),%rdx

40049f: 88 84 d6 60 10 60 00 mov %al,0x601060(%rsi,%rdx,8)

4004a6: 0f be c0 movsbl %al,%eax

4004a9: c3 retq

00000000004004aa <func1>:

4004aa: c6 05 cb 0b 20 00 0d movb $0xd,0x200bcb(%rip)

# 60107c <array+0x1c>

4004b1: b8 0d 00 00 00 mov $0xd,%eax

4004b6: c3 retq