# 15213 Lecture 8: Data in Memory

## 1 Getting Started

To obtain a copy of today's activity, log into a shark machine and do the following:

1. `$ wget http://www.cs.cmu.edu/~213/activities/lec8.tar`
2. `$ tar xf lec8.tar`
3. `$ cd lec8`

Now run `$ ./act8` and follow the instructions on your screen. It will occasionally ask you discussion questions, whose answers you can record in the following section. Feel free to refer to the activity sheet from last week if you need a reference of GDB commands.

## 2 Discussion Questions

**Use GDB's `c` command to progress through the stages.** These questions accompany the program; as it poses each one, discuss with your partner and write your answer here.

### 2.1 Integers

1. Imagine we needed to take the address of the variable `local`. What problem might we run into, and what do you expect the compiler to do about it?
   The variable local is stored in register %edi. The problem is, registers do not have addresses. The compiler will instead store the variable on the stack, since stack locations have addresses.

### 2.2 Arrays

2. What is the stride of this array (the number of bytes occupied by each entry)? ___4___ bytes
3. Assuming you already have a pointer to the beginning of a C string, how do you determine where it ends?
   C terminates strings with the NUL character (ASCII character code 0, or `'\0'`). You have to scan the string, counting bytes, until you reach the NUL.

### 2.3 Structs

4. Did you notice anything familiar about the layout?
   The layout of this 4 integer struct looks the same as the layout of the 4 integer array.

5. Write '`a`', '`b`', '`c`', or '`d`' in each box based on your prediction of what that byte will contain. If you expect any bytes to be unused, leave them empty.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | | | | | | | | |
| 0x08 | | | | | | | | |
| 0x10 | | | | | | | | |
| 0x18 | | | | | | | | |

6. If you were incorrect, lightly cross out the previous table and use this one to record the correct layout as shown in the dump.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | a | x | b | b | c | c | c | c |
| 0x08 | d | d | d | d | d | d | d | d |
| 0x10 | - | - | - | - | - | - | - | - |
| 0x18 | - | - | - | - | - | - | - | - |

7. Will this type take up more or less space than the first?
   This type will take up more space because the order of its elements produces more padding.

## 2.4 Arrays of Structs

8. What stride do you expect this array to have? 8 bytes
9. How will this struct's size compare to that of `pair`?
   This struct's size is smaller (6 bytes) since it requires less padding.

## 2.5 2-D Arrays

10. What stride do the "inner" arrays have? 1 bytes How about the "outer" ones? 3 bytes
11. Do you think this function would be useful for an array declared as: `int8_t flipped[3][2]`?
    No, the dimensions (specifically, the outer stride) do not match. So the compiler will not be able to access the fields of the array correctly.

12. What stride does the outer array have this time? 8 bytes
13. Do you think this function would still be useful if `first` and `second` each had 4 elements? How about if they had two different lengths?
    This function would still work because its assembly does not use the lengths of first and second in computation.

14. What effect would we observe if we modified an element of `first`?
    The modification would occur on both multilevel[0] and multilevel[1].

## 2.6 Endianness (Optional)

15. What disadvantage of little-endian did you just observe?
    Little endianness is harder to read in a byte-by-byte memory dump.

16. How would the assembly of this function differ if x86-64 were a big-endian architecture?
    `mov 4(%rdi), %eax`