L03: 9/9 Notes

Scribed by Lily Tsai, Sammy Mehra, Fatma Akcay, Antuan Tran

Problem Set Notes & Hints

- Try to solve problem set from bottom-up. ("test driven problem solving")
  - Worry that graders are 'not that level of idiot'.
- M61_printstatistics already has printf statements! Calls take values from struct 'stats'.
  - Could just update values (with ->) in struct 'stats'.
- Hints!
  - Static struct m61_statistics gstats = {0, 0, 0, 0, 0, 0, NULL, NULL};
    - This initializes all of the fields in gstats in the order they are defined.
  - This means you can assign your global variable to the local in get_statistics, and the written code will print the values out!

DEFINITIONS

OBJECT	region of storage distinct objects *never overlap*	examples: char, int, long
Declaration	In C, creates an object and a name for the object	eg: char ch //creates a character object with name ch

ALIASES	different names for the same object (the names differ in the abstract machine but the information is located in the same place on the physical real machine)	int x = 10;
		int* ptr = &x;
		
		Since data is represented as addresses, both x and ptr are names for the same object!

BYTE	8 bits in memory Represent a number 0-255	01001011

Objects in C (and Hello Kitty)

Why is there a hello kitty picture on the CS61 wiki?
Hello Kitty is not a cat. A Japanese philosopher said she is not a cat because she has a cat, and cats do not have cats. Also she stands up on two legs. This is related to data representation.
WHAT does this have to do with data representation?
An object like 'char' can be assigned to a character like 'A' or to the integer '65'. It is an character and it is not a character (it is an int)….. It is a cat, and its not a cat.

Some thought experiments:

Q: Can we store a value of 256 in a byte?
A: NO! 256 = 0b100000000, only the 8 lowest bits (in red) will be stored by the compiler!

Q: Based upon the previous answer, what does the compiler end up storing in the byte for char ch = 0x149;?
A: 0x49 (each hex digit = 4 bits! lowest 8 bits stored)

---

sizeof():

Q: What does sizeof(object) do?
A: Returns size (in bytes) of object passed in as argument

Q: What type does sizeof() return?
A: size_t (unsigned)

TYPES

Real machine vs. Abstract machine:

- Real machine deals with *bytes* and *addresses* à unsigned/signed ints can have the same representation in bytes, machine can't differentiate
- Compiler is the one that can interpret bytes as particular types!

Fundamental Types

- unsigned [char, int]
- char, int, long, float, double
- size_t (signed/unsigned)

Derived Types: built up from fundamental types, defined by rules of memory!

- ARRAY: laid out sequentially, contiguous blocks of memory
- address of 0^{th} element = address of array
- &(1^{st} element) = &array + sizeof(element)
  &x[1] = x + 1; → if x is an int array, the value of x + 1 = address of x + 1*sizeof(int)!
- See the homepage for examples: here
- array sizes:
  T x[N];
  sizeof(x) == N x sizeof(T);

Data Storage

Little Endian: Highest order (least significant) digit written first
e.g. 0x149 = 49 01 00 00
Big Endian: Lowest order (most significant) digit written first
e.g. 0x149 = 00 00 01 49

Testing!

Use hexdump to see the hexadecimal value of the address of the variable, and the value of the variable itself in hexadecimal, decimal, and ASCII.

Possible tests:

- verify if compiler is using little or big endian data storage!
- see for yourself how arrays are represented as pointers, and how pointer arithmetic works on arrays!