Data Types

- Why do we need data types?
  - Data resides in memory
  - Memory has no specific structure by itself
  - It’s up to interpretation to make sense out of it
  - Sometimes: value ranges

- Type systems help to apply and check for a syntactically consistent interpretation
  - Rule out programs that do not make any sense
  - Allowing certain operations while disallowing others
  - Enforcing syntactic discipline
  - Induce invariants

Data Types in C

- boolean
  - true or false, represented as 0 or (1 or -1)
  - only in C99: bool

- char
  - smallest addressable type
  - can represent a single character

- int
  - integer type, at least 16 bits

- long
  - integer type, at least 32 bits

- long long
  - integer type, at least 64 bits

Today, we will think of types as representation

Which data types does a computer understand?
- Bits?
- Bytes?
- Words?

Which data types do we know from programming languages?
- boolean, scalar types, structs, ...
Signed vs. Unsigned
- all the before mentioned types are signed by default in C
  - One bit is used to store the sign, the rest for the actual value
  - Two’s complement: \(-2^{n-1} \cdot (2^{n-1} - 1)\)
- unsigned types have same representation in memory, different interpretation
  - \(0 - 2^n\)
- Data types have limited range, overflow may occur

Exact-Width Integer Types
- Size of the individual data types is platform-dependent
- Can be checked at runtime with sizeof(...).
- C99 introduced stdint.h
  - int8_t, int16_t, int32_t, int64_t
  - uint8_t, uint16_t, uint32_t, uint64_t

Floating Point Types
- float
  - single precision floating point number, usually IEEE 754
- double
  - double precision floating point number, usually IEEE 754

Floating Point Types
- As was said, 5.2 is represented as a sign bit, an exponent and a mantissa. How do you encode 5.2?
  - 1 0111 0100 1100 1100 1100 1100 1100
  - 5 floats
  - IEEE 754
  - The rest, 0.2 is 1/5, so divide \(6 \times \ldots \) by 5 and you get \(6.333333\ldots\)
  - That should give you:
    - \(0.00000000000000000000000000000000\)
    - Add 1, and you get
      - \(1.00000000000000000000000000000000\)
      - Now shift it and adjust the exponent
        - \(1.00000000000000000000000000000000\ldots \exp +2 = (1.2 \times 2^2)\)
        - Now you only have to add 127 (\(127 = 01111111\)) to the exponent and store it
          - \(1.2 \times 2^{14}\)
          - Forget the top 1 of the mantissa, and you get
            - \(11111111001011001100110011001100\)
            - Now you only have to decide if it is long enough.
            - This is not exactly how it works, but that is more or less what happens when a number like 5.2 is converted to binary.
** Enums**
- Replaces constructs like
  - `#define NORTH 1`
  - `#define EAST 2`
  - `#define SOUTH 3`
  - `#define WEST 4`
- `enum orientation {NORTH, EAST, SOUTH, WEST}`

** Pointers and Pointer Types**
- A variable corresponds to a location in memory
- A memory location has an address
- Pointers allow the programmer to pass or manipulate the address of a variable
  - `x = &y;`  
  - `z = *x;`
- Pointer types
  - `int *x, float **p, ...`

** Pointers**
- `int number = 3;`
- `int *number_p = NULL;`
- `number_p = &number;`
- `*number_p = 10;`

** Arrays**
- Fixed length
  - `int a[4];`  
  - `for (x=0; x<4; x++) { a[i] = x; }`
- Variable length
  - `int *a = (int*) malloc(n * sizeof(int));`
  - `for (x=0; x<n; x++) { a[i] = x; }`
  - `free(a);`
- Arrays = pointer arithmetic
- Multi-dimensional arrays
  - `a[2][2]`
Arrays and Cache Effects
- demo

Strings
- char[] s = “Hello”;
  - char
    - ‘H’
    - ‘e’
    - ‘l’
    - ‘l’
    - ‘o’
    - \0
- char *t = “World”;
  - char*
    - \0

Functions and Data Types
- int a(int x);
- float divdiv(float x, float y, float z);
- void b(void);
- void *c();
- void *malloc(size_t);

Function Pointers
- In C, functions can be treated like data type
- Function pointers
  - void qsort(void *base, size_t nmb, size_t size, int(*compar)(const void *, const void *));
  - (*compar)(a,b);
  - compar(a,b);
Structures

```c
struct person {
    long id;
    char* name;
    char country;
    short department;
    int salary;
};
```

```c
__attribute__((packed))
__attribute__((aligned(<alignment>)))
```

Bitfields

```c
struct on_off {
    unsigned light : 1;
    unsigned toaster : 1;
    int count;
    unsigned ac : 4;
    unsigned : 4;
    unsigned clock : 1;
    unsigned flag : 1;
} kitchen;
```

```c
kitchen.light = 1;
```

Union Types

```c
union {
    struct {
        uint8_t high;
        uint8_t low;
        uint8_t tag;
        uint8_t t;
    } val;
    unit64_t data;
} message;
```

```c
message.val.tag = 0x10;
send(value.data);
```

typedef

```c
typedef int score;
```

```c
struct tree_element {
    score value;
    struct tree_element *right, *left;
};
```

```c
typedef struct tree_element node;
```

```c
typedef struct tree_element {
    score value;
    struct tree_element *right, *left;
} node;
```