What are Types?
data types

• A **data type** constrains the **set of valid values** a piece of data can take on
  • An **int** in C can take on values from \([-2^{31}, 2^{31} - 1]\)
  • A **char** in C can take on values from \([0, 255]\)
  • Not always easy to define this set (what are the sets of valid values for floats?)
  • Some times we express this information explicitly:
    ```
    int c = 0
    ```
  • Other times, it’s implicit:
    ```
    x = “Hello from Python”
    ```
data types

- Constraining the set of values helps determine many other things
  - How much space it takes up (ints take up 4 bytes, chars take up 1 byte)
  - How to interpret a sequence of bits: 01000001
    - If the data is an int, this is 65
    - If the data is a char, this is ‘A’
- What kinds of operations you can do on it
  - Can add together two ints
  - Cannot add together two bools
• Pieces of data are not the only things that can have types
• **Functions** can have types too!
  
  ```c
  int foo(int i, char c)
  
  has type \((int \times char) \rightarrow int\)
  ```

• Constrains behavior just like data types do:
  • When I call foo, I need to pass it an **int** and a **char**
  • When I use the return value of foo, I should treat it as an **int**
even more types

- **Arrays:**
  \[ \text{int } a[10] \] : means that an array has exactly 10 items of type int

- **Pointers:**
  \[ \text{float } * * p \] : means a pointer that points to another pointer that points to a float

- **Structs:**
  \[ \text{struct } \{ \text{int } x; \text{ float } f; \} s \] : means a piece of data that contains an int and a float
what can go wrong?

• What can go wrong if we do not pay attention to types?
  • What happens if we generate code to add an int to a float?
  • What happens if we pass the wrong kind of data to a function?
  • What happens if we access past the end of an array?
  • What happens if we use the wrong kind of load to access the first field of a struct?

• In our simulator, many of these operations will trigger a runtime failure (try it!)
  • The simulator does dynamic type checking under the hood, but in reality, in many cases you will just get very strange behavior in your program
types as constraints

- Think of types as imposing constraints on the behavior of your program
  - Operations only between matching types
  - Functions called with appropriate arguments [is the previous point just a special case of this point?]

- Different programming languages, compilers, and runtime systems do different things to enforce these constraints
  - Not all constraints are always enforced!
next: dynamic type checking