Administrivia

- Student hours (or by appointment):
  - Listed on Canvas in Syllabus
  - MTWTh 10:30 – 11:30 a.m. via Zoom
- **Note**: this coming Monday student hour is canceled
- Assignment four posted
  - Due Tuesday 4 July by **10pm**

CptS 355- Programming Language Design

Types and Type Checking

Instructor: Jeremy E. Thompson

Types

- How would you define a type? What is it and what does it do?
  - Type defines a collection of values that share a common property - usually a *common set of operations*
  - Type tells you what is *legal* to do with some value in the language

Types

What is a **type error**?

- An attempt to use a value in an operation *inconsistent* with the value's type
- Examples:
  - The following will produce a type error in many languages

    ```
    x = 17
    x (  );
    ```

  - The following *may* produce a type error in *some* programming languages

    ```
    3 + 4.5
    ```

Compile-time (static) type checking *versus* Run-time (dynamic) type checking

- Compile-time (static) type checking:
  - Examples (Haskell):
    - Checks that *all* return values of a function have the same type
    - Checks that *all* patterns are exhaustive
  - Compile-time type checking is necessarily *conservative*:
    - it may flag as an error for something that would not actually cause a run-time error
  - Advantages:
    - Less runtime *overhead*
    - You only do the type-checking *once* (compile-time)
    - The *entire* program is checked

Compile-time (static) type checking *versus* Run-time (dynamic) type checking (cont.)

- Run-time (dynamic) type checking:
  - Run-time type checking is *expensive*
    - must be done each time the program is executed
  - Advantages:
    - Allows certain programming styles not possible with compile-time type checking
    - More *flexible* data structures
      - For example, lists in Python may contain *values of any type* whereas lists in Haskell must have elements of the *same type*

Type Safety: strong typing

- Strong typing ensures that *every* use of a value is compatible with its type
  - Requires *explicit* conversion
  - Static strong typing → compiler error
    - Example : Haskell
  - Dynamic strong typing → error at the point of misuse
    - Example: Python
- In type-safe languages, values are managed "*from the cradle to the grave*":
  - Objects are *created* and *initialized* in a type-safe way
  - An object *cannot be corrupted* during its lifetime
    - its *representation* is in accordance with its *type*
  - Objects are *destroyed*, and their memory reclaimed, in a type-safe way

Type Safety: strong typing

- C doesn't have type safety
  - C <u>heap</u> values are created in a type-unsafe way
  - C *casts* and unchecked array *accesses* can corrupt memory during its lifetime

```
void f(char* char_ptr) {
   double* d_ptr = (double*)char_ptr;
   (*d_ptr) = 3.5;
}
```

← **Breakout**: What happens here?

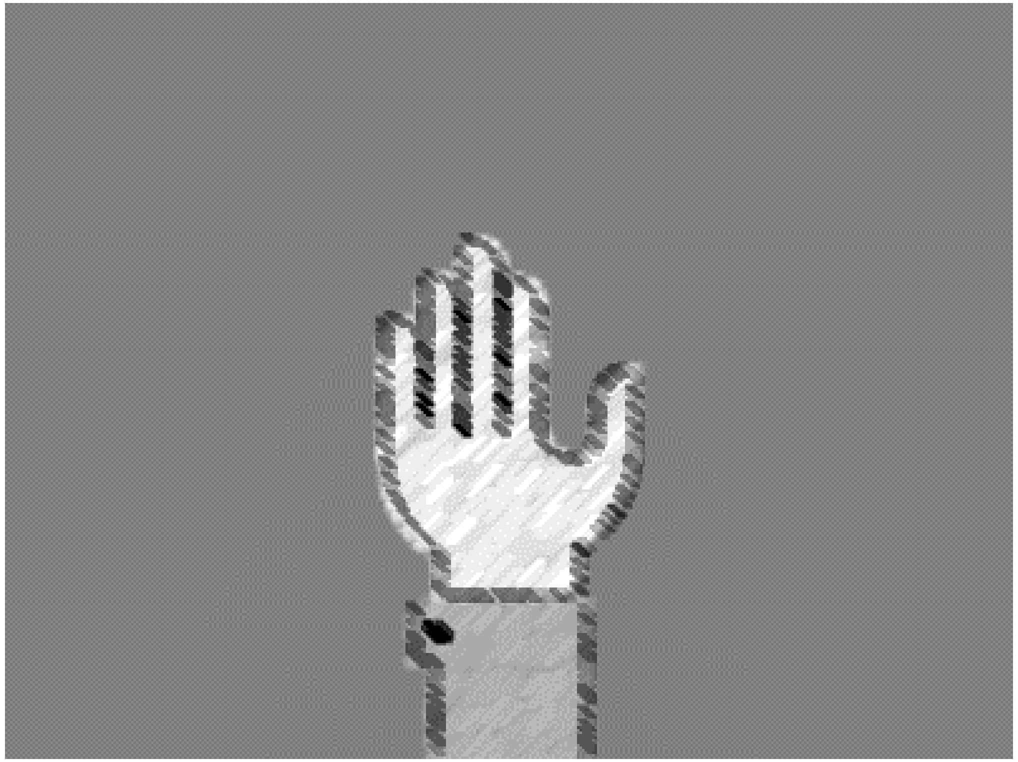
  - C <u>deallocation</u> is unsafe, and can lead to *dangling* pointers

```
int *i = (int*)malloc(sizeof(int));
int j = 0;
*i = 4;
free(i);
/* ... some code that might allocate memory, then: */
*i = j;
```

← And here?

Questions?

Quiz!



Reminder

- Student hours (or by appointment):
  - Listed on Canvas in Syllabus
  - MTWTh 10:30 – 11:30 a.m. via Zoom
- **Note**: this coming Monday student hour is canceled
- Assignment four posted
  - Due Tuesday 4 July by **10pm**