Welcome forward to week 2!

Quiz everyone say YAY!

```
while (true) {
   check_feedback();
}
```

How was the quiz?

A. easy
B. mostly fine
C. mostly fine, but not enough time
D. too hard, but finished mostly in time
E. too hard and not enough time
F. too hard regardless of time

Stress

- 439H is **not an easy class**
  - Lots of new material
  - Unfamiliar programming environments
  - Fast, often relentless pace
- Struggling in this course is normal
  - There will be times you won't know the answer or solution
  - This is expected - we want everyone to succeed, but the only way we can help is if you ask for it
- If you find yourself overwhelmed or spending more time on this class than you think you should be, **please reach out** to Dr. Gheith or the TAs
  - We can help out as far as the class goes
  - We can provide other resources if we are not able to help

Mental health resources available at UT

P2

```
while (true) {
    check_feedback();
}

if (feedback.max() == 'A') {
    Debug::panic("");
}
```

How is p2 going?

A. There’s a p2???
B. Cloned the project.
C. Looked through the starter code.
D. Started planning/writing code
E. Done with at least one part of the project
F. p2 speedrun any% glitchless

Event loop

- What is an event loop?
- How would we implement the event loop?
  - How do we add events?
  - How do we actually run events?
  - How do we switch to different events?

Event

- Some work to be run at some time
- Events can be run after a specified delay
  - `pit.h` has a lovely `jiffies` counter

```
class Pit {
    static uint32_t jiffiesPerSecond;
    static uint32_t apitCounter;
public:
    static uint32_t jiffies;
    static void calibrate(uint32_t hz);
    static void init();
    static uint32_t secondsToJiffies(uint32_t secs) {
        return jiffiesPerSecond * secs;
    }
    static uint32_t seconds(void) {
        return jiffies / jiffiesPerSecond;
        return 0;
    }

};

#endif
```

Channels

- What is a channel?

Channels

- What is a channel?
  - Communication method between different events in our case
    - Similar to coroutines
  - Buffer of size 1 (different from coroutines!)
  - `send` optionally takes in a callback with no arguments
  - `receive` optionally takes a callback with **one** argument
  - Callbacks run immediately if a value/space is available
  - Callbacks are **delayed** until they can match up appropriately
- Implementation considerations
  - How do you make sure callbacks are called at the appropriate time?
  - How do you hand off values from senders to receivers?

Futures

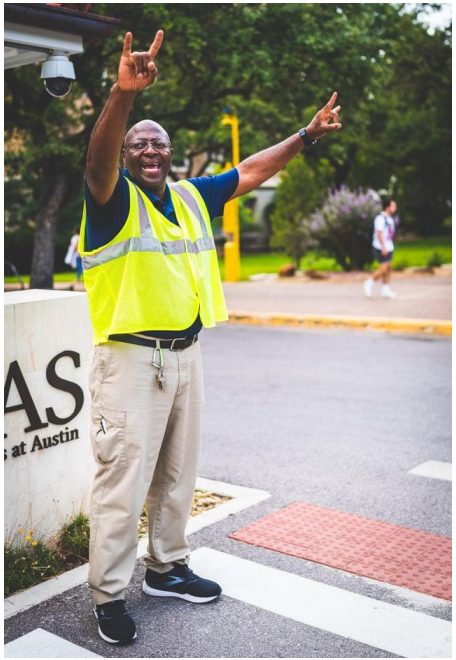
- What is a future?

Futures

- What is a future?
  - Holds a value that will be set at some point by some event
  - `set`ting the value is straightforward
    - `set`ting the value more than once is **undefined behavior**
  - `get`ting the value takes in a callback with **one** argument
    - Only called once the value is actually `set`
- Implementation considerations
  - How do you mark a value as ready or not?
  - How do you make sure callbacks are called at the appropriate time?

Barriers

- What is a barrier?



Barriers

- What is a barrier?
  - Waits until n events have called sync to run all of their callbacks
  - Calling sync more than n times is **undefined behavior**
- Implementation considerations
  - How do you make sure callbacks are called at the appropriate time?

Explain how the event class works (the polymorphism)

- What's `virtual`?
- Every closure has a different type
- `Queue<EventWithWork<???>>`
- What can we call the closure?
- `EventWithWork` inherits from `Event`
- overrides `doit` to call the closure
- You can pass `EventWithWork*` to something that uses `Event*`

```
struct Event {
    Event* next = nullptr;
    virtual void doit() = 0;
    virtual ~Event() {}
};

template <typename Work>
struct EventWithWork: public Event {
    const Work work;
    explicit inline EventWithWork(const Work& work)
      : work(work) {}
    virtual void doit() override {
        work();
    }
};
```

Questions?