```
if (justMetYou) {
  crazy = true;
  cout << number << endl;
  int x = rand()%100;
  if (x>=50)
      callMe();
}
```

"Call Me Maybe"

```
private function bad() {
  break;
}
```

"Breaking Bad"

```
class StarWars(int episode) {
  if (episode == 6)
      return Jedi;
}
```

"Star Wars: Episode VI – Return of the Jedi"

```
try
{
    Assert(Life.Real);
    Assert(Life.Fantasy);
}
catch(LandSlideException ex)
{
     #region Reality
     while(true)
     {
         character.Eyes.ForEach(eye => eye.Open().Orient(Direction.Sky).See(););
         self.Wealth = null;
         self.Sex = Sex.Male;

         if(self.ComeDifficulty == Difficulty.Easy && self.GoDifficulty ==
    Difficulty.Easy && self.High < 0.1 && self.Low < 0.1)
         {
             self.Sympathies.Clear();

             switch(wind.Direction)
             {
                 case Direction.North:
                 case Direction.East:
                 case Direction.South:
                 case Direction.West:
                 default:
                 piano.Play();
                 break;
             }
         }
     }
     #endregion
}
```

“Bohemian Rhapsody”

Section 1:
Code Reasoning

Alex Mariakakis
cse331-staff@cs.washington.edu (staff-wide)

INTRO + STORY TIME !!!

Reasoning About Code

- Two purposes
  - Prove our code is correct
  - Understand why code is correct
- Forward reasoning: determine what follows from initial conditions
- Backward reasoning: determine sufficient conditions to obtain a certain result

Forward Reasoning

```
{x >= 0, y >= 0}
y = 16;
{x >= 0, y = 16}
x = x + y
{x >= 16, y = 16}
x = sqrt(x)
{x >= 4, y = 16}
y = y - x
{x >= 4, y <= 12}
```

Forward Reasoning

```
{true}
if (x > 0) {
      {x > 0}
      abs = x
      {x > 0, abs = x}
}
else {
      {x <= 0}
      abs = -x
      {x <= 0, abs = -x}
}
{x > 0, abs = x OR x <= 0, abs = -x}

{abs = |x|}
```

Backward Reasoning

```
{x + 3b - 4 > 0}
a = x + b;
{a + 2b - 4 > 0}
c = 2b - 4
{a + c > 0}
x = a + c
{x > 0}
```

Backward Reasoning

```
{y > 15 || (y <= 5 && y + z > 17)}
if (y > 5) {
      {y > 15}
      x = y + 2
      {x > 17}
}
else {
      {y + z > 17}
      x = y + z;
      {x > 17}
}
{x > 17}
```

Implication

- Hoare triples are just an extension of logical implication
  - Hoare triple: {P} S {Q}
  - P → Q after statement S
- Everything implies true
- False implies everything

P	Q	P → Q
T	T	T
T	F	F
F	T	T
F	F	T

Weaker vs. Stronger

- If P1 → P2, then
  - P1 is stronger than P2
  - P2 is weaker than P1
- Weaker statements are more general
- Stronger statements are more restrictive

Weaker vs. Stronger

`y >= 16` $\boxed{\texttt{y = 16}}$

$\boxed{\texttt{x is even, y = x + 1}}$ `x is even, y is odd`

$\boxed{\texttt{"Alex is an awesome TA"}}$ `"Alex is a TA"`

Weakest Precondition

- The most lenient assumptions such that a postcondition will be satisfied
- If P* is the weakest precondition for {P} S {Q}, then $P \rightarrow P^*$ for all P that make the Hoare triple valid
- Notation: WP = wp(S, Q)

Weakest Precondition

```
wp(x = y*y, x > 4)
|y| > 2

wp(y = x+1;z = y-3, z = 10)
wp(y = x+1, wp(z = y-3, z = 10))
wp(y = x+1, y-3 = 10)
wp(y = x+1, y = 13)
x = 12
```