Parameter Modes

Parameter Modes

- There are four ***parameter modes***:
  1. `clears`
  2. `replaces`
  3. `restores`
  4. `updates`

- Each mode indicates a possible way that a operation might change the value of the corresponding actual parameter's value

Parameter Modes

- The 4 parameter modes help us as follows:
  - For the caller of an operation, they concisely summarize which actual parameters might have their values modified by the called operation
  - They make requires/ensures clauses shorter

Parameter Modes

- One of the four modes (`clears`, `replaces`, `restores`, `updates`) is always associated with each of the operation's formal parameters

- This includes **`self`**, which represents the object in front of the dot in the call to an class's member function

Clears Mode

- A formal parameter annotated with ***clears*** indicates that its outgoing value is an ***initial value*** for its type, i.e., a value equivalent to what the *constructor* assigns
  - If an operation has **`clears`** *`x`* in its specification, then this is equivalent to adding **`and`** *`x = [an initial value for its type]`* to the ensures clause
  - *`x`* should not appear in the ensures clause
  - *`#x`* may appear in the ensures clause

Example of clears

```
void transferFrom(Integer& n)
  //! replaces self
  //! clears n
  //! ensures: self = #n
```

- Sets **`self`** to the incoming value of `n` and resets `n` to an initial value for Integer

Replaces Mode

- A formal parameter annotated with ***replaces*** indicates that its outgoing value might be changed from its incoming value, and the operation's behavior *does not depend* on its incoming value

  - `x` should not appear in the requires clause
  - `#x` should not appear in the ensures clause

    Why? Because the operation does not depend on the incoming value of `x`

Example of replaces

```
void transferFrom(Integer& n)
  //! replaces self
  //! clears n
  //! ensures: self = #n
```

- Sets **`self`** to the incoming value of `n` and resets `n` to an initial value for Integer

Restores Mode

- A formal parameter annotated with ***restores*** indicates that its outgoing value is equal to its incoming value
  - If an operation has **`restores`** `x` in its specification, then this is equivalent to adding **`and`** `x = #x` to the ensures clause
  - Since `x = #x`, an `#x` should never appear in the ensures clause, instead only `x` should appear in the ensures clause

Example of restores

```
Queue1& operator =(Queue1& rhs)
   //! replaces self
   //! restores rhs
   //! ensures: self = rhs
```

- Sets **`self`** to a copy of the incoming value of `rhs` and the outgoing value of `rhs` is equal to the incoming value of `rhs`

Updates Mode

- A formal parameter annotated with ***updates*** indicates that its outgoing value might be changed from its incoming value in a way that might depend on its incoming value

Example of updates

```
void enqueue(T& x)
   //! updates self
   //! clears x
   //! ensures: self = #self * <#x>
```

- Adds `x` to the rear of queue **`self`** and resets `x` to an initial value for type `T`

Credits

- These slides were adapted from slides obtained from Dr. Bruce W. Weide and Dr. Paolo Bucci.
- Drs. Weide & Bucci are members of the Resolve/Reusable Software Research Group (RSRG) which is part of the Software Engineering Group in the Department of Computer Science and Engineering at The Ohio State University.