

Lecture 13: Programming by Contract

PreConditions and PostConditions

- PreCondition
 - what you assume to be true before an operation or process
- PostCondition
 - what you assert to be true after an operation or process
- Contract: If Preconditions hold before, then Postconditions will hold afterwards

PreConditions and Data Validation

- Precondition client's responsibility
- Input validation is not required
- Documented so it does not slip through the cracks
- Clarifies when and where data needs to be checked
- Redundant checks lead to
 - decreased performance
 - less clarity
 - increased opportunity for errors

Kinds of Postconditions

- Low level
 - Conditions and relationships on variable values
 - Return value properties
- Higher level
 - Objects created or deleted
 - Associations formed or deleted

Applications

- Low level
 - Algorithms
 - Classes
 - class and method properties
- Higher level
 - Use case assumptions and results
 - System and subsystem operations identified during requirements analysis and elaboration

Algorithms and Pre/PostConditions

- Formal specifications
- Proofs of correctness of algorithms
 - verification, formal verification

Assertions

- Assertion: Condition that should be true at an associated location in the algorithm
- PreCondition= input assertion
- PostCondition = output assertion
- Intermediate assertion – intermediate state of system/program
- Loop invariant = assertion about state at some location in a loop

Algorithms and Verification

- Suppose that A_1 is the precondition for an algorithm A and A_n is the postcondition.
- Correctness: A is correct if it is *partially correct* and it always terminates
- Partial correctness of A
 - if A_1 is true at the beginning of A and if A is executed and terminates, then A_n will be true at the end of the execution

Proof Technique

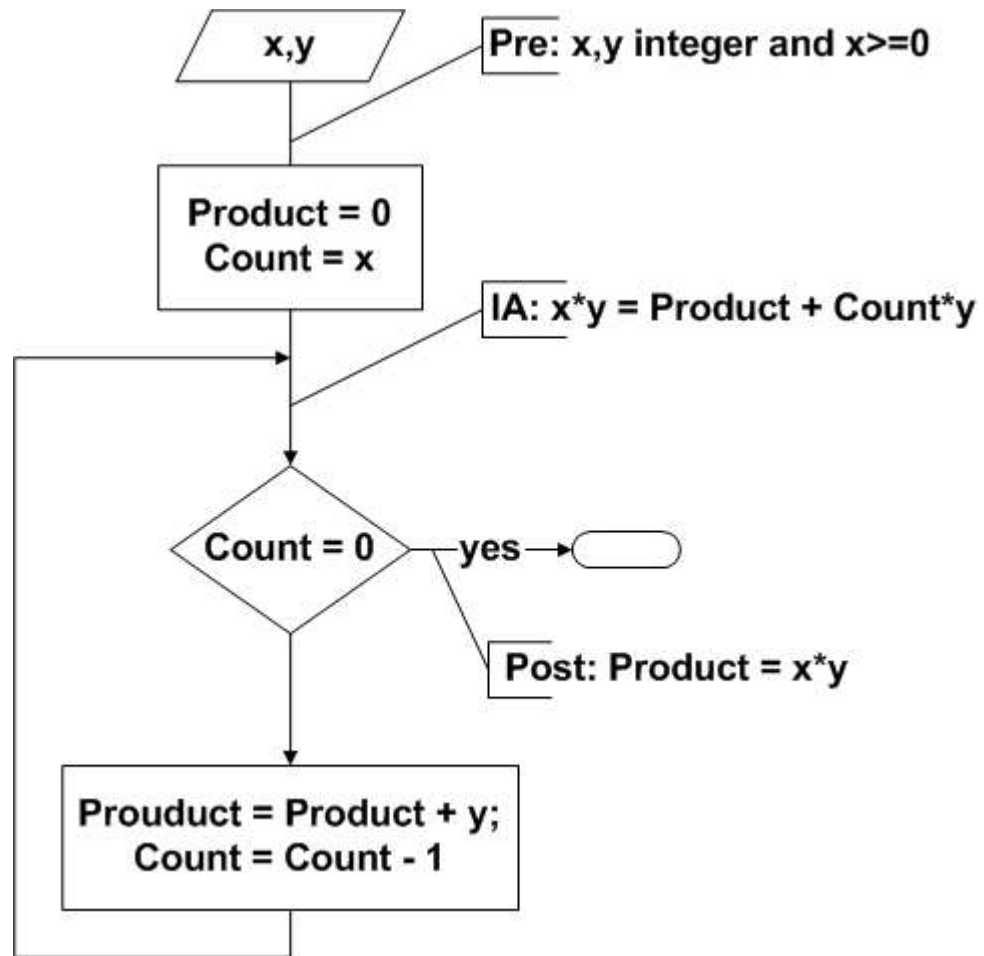
- Add intermediate assertions A_i to the algorithm.
Make sure that each loop has at least one intermediate assertion
- For each pair of assertions (X, Y) where there is a subpath p from X to Y , which has no other assertions on it, prove that if
Control is at the location of X and X is true,
and if control flows along p to Y , then Y will be true

Proof Method Validity

- Assume that we prove validity for each assertion pair (X, Y) joined by a subpath p
- For any execution of the algorithm, a path P will be followed from the precondition to the postcondition
- The path P can be broken up into subpaths p between assertion pairs (X, Y)
- Proofs for the subpaths p implies correctness for whole path P

Sample Algorithm

- Multiplies two numbers x and y by adding up y x times
- Input (precondition) assertion Pre , output (postcondition) assertion $Post$, intermediate loop invariant assertion IA



Verification Conditions for Sample Algorithm

- Prove that
 - if Pre is true then IA will be true
 - if IA is true and Count = 0 then Post will be true
 - If IA is true and Count $\neq$ 0 then IA will be true

Termination Proof?

- Termination proof
 - Count initialized to $x \geq 0$
 - Loop terminates if $\text{Count} == 0$
 - For each iteration Count is decremented by 1 so loop must terminate
- Note: if Precondition $x \geq 0$ is removed and input is negative, algorithm will not always terminate but is still partially correct

Classes and Pre/Post Conditions

- Method Pre and Post conditions
 - algorithm input/output conditions
 - state changes for object
- Class invariants
 - true after constructor, and true before and after each method application
 - could also document as pre and post conditions for each method to ensure compliance

Sample Class Invariants

- e.g. LogOn object created by DomainLogic when someone logs on to DS
- Has a class variable called name which is never null-valued after LogOn is constructed. i.e. it is initialized in the constructor

class invariant: {this.logOn $\neq$ null}

Sample Class

- Stack Class (parameterized)
- Written in a language with
 - class invariant statement
 - precondition and postcondition statements for methods

```

Class Stack[T]
{
    T [ ] Elements
    Int numberElements k
    Int maxSize k
    ...
    invariant 0 <= numberElements; numberElements <= maxSize;

    public void pop()
    {
        precondition notEmpty;
        ...
        postcondition
        {
            notFull;
            numberElements' = numberElements-1;
        }
    }

    public T top()
    {
        precondition notEmpty;
        ....
    }

    public push(T x)
    {
        precondition notFull;
        ....
        postcondition
        {
            notEmpty;
            top() = x;
            numberElements' = numberElements+1;
        }
    }
}

```

Use Cases and Pre/Post Conditions

- Use case is a story that may contain a number of system operations
- Use case pre/postconditions
 - for whole story

DS Example - Precondition

- Use Case: user changing his/her data in the DS data base
- Assume that the use case does not include the log on subtask, which will always be performed before this use case and which will have confirmed that the user was a member before this use case began
- PreCondition: User is a member of the Dating System and has a member record in the Data Base

DS Example – Postcondition

- Use Case: user changing his/her data in the DS data base
- Suppose that m is the current user record for U in the data base, and c specifies changes to the data. Let $m' = \text{change}(m, c)$, the result of making changes c to m .
- Postcondition: User record for $U == m'$

Systems/subsystems and Pre/Postconditions

- Operations determined during requirements analysis and elaboration
- Systems and subsystem events correspond to actor actions or messages received
- Events are responded to by system/subsystem operators
- Pre and post conditions describe required operator functionality

Operator Pre and Postconditions

- High level, more informal than algorithm and method conditions
- More object oriented e.g.
 - Objects created and deleted
 - Associations between instances of classes created and deleted
 - Attributes of objects altered

DS Examples – 1

Subsystem: GUI

Operation: Create(DomainLogic D1)

Preconditions: Domain Logic object is created

Postconditions: GUI object is created

DS Examples - 2

- Subsystem: GUI
- Operation: name entered during Logon
- Precondition: Logon is visible
- Postconditions:
 - LogOn is not visible
 - Appropriate user options dialog (member, admin, unauthorized) created and visible

DS Examples – 3

- Subsystem: GUI
- Operation: EnterMemberData option is chosen from member options dialog
- Preconditions: member options dialog is visible
- Postconditions: EnterMemberData Dialog is visible

DS Examples – 4

- Subsystem: DL
- Operation: logOn
- Preconditions: DataBase is associated with DL
- Postconditions:
 - LogOn object created and associated with DL
 - LogOn userName and userType attributes are set

DS Examples – 5

- SubSystem: DL
 - Operation: addMember(String name)
 - Preconditions: DataBase is associated with DL
 - Postcondition: (none for DL, i.e. no changes for DL)
- /* Recall: postcondition will only refer to object/subsystem under consideration */

DS Examples – 6

- Subsystem: DL
- Operation: deleteMember(String name)
- PreCondition: DataBase is associated with DL
- PostCondition:
/* in the design and implementation a value is
returned by the method for this operation which
would be in a more detailed postcondition */

DS Examples – 7

- Subsystem: DataBase
- Operation: Create (File file)
- Precondition: file exists and contains member records in expected format
- PostCondition: the MemberData[] object associated with DataBase will be filled with the member records from the file

DS Examples – 8

- Subsystem: DataBase
- Operation: getMemberData(String name)
- Precondition:
- Postcondition

/* no changes to DataBase object. Obviously
there is a return value from the operation */

DS Examples – 9

- Subsystem: Data Base
- Operation: addMember(String name)
- Precondition: DataBase has a MemberData [] collection object associated with it
- Postcondition: the MemberData[] object is altered to include a new MemberData instance with this name