Design and Analysis of Algorithms

Prepared by Dr. Muhammad Imran
Modified by MN Khokhar

Pre-Conditions and Post-Conditions

- **Procedural Abstraction:**
  - Writing a complete operation, method or procedural declaration: Specify HOW.
  - Users do not need to know HOW.
  - Very powerful tool that allows programmers to consider operations one at a time rather than all together.

Pre-Conditions and Post-Conditions

- **Pre-Condition:**
  - A statement or set of statements that outlines a condition that should be true when the operation is called.
  - The operation is not guaranteed to perform as it should unless the pre-conditions have been met.

Pre-Conditions and Post-Conditions

- **Post-Condition:**
  - A statement or statements describing the condition that will be true when the operation has completed its task.
  - If the operation is correct and the pre-condition(s) met, then the post-condition is guaranteed to be true.

Correct Algorithms

- An algorithm is said to be correct if, for every input instance, it halts with the correct output. We say that a correct algorithm solves the given computational problem.
- An incorrect algorithm might not halt at all on some input instances, or it might halt with an answer other than the desired one.

- *Incorrect algorithms can sometimes be useful, if their error rate can be controlled. (An example of this when we study algorithms for finding large prime numbers.)*

What does an algorithm ?

- An algorithm is described by:
  - Input data
  - Output data
  - ***Preconditions***: specifies restrictions on input data
  - ***Postconditions***: specifies what is the result
- Example: Binary Search
  - Input data: `a:array of integer; x:integer;`
  - Output data: `found:boolean;`
  - Precondition: `a` is sorted in ascending order
  - Postcondition: `found` is true if `x` is in `a`, and `found` is false otherwise

Correct algorithms

- **An algorithm is correct if:**
  - for **any** **correct** **input data**:
    - it **stops** and
    - it produces **correct** **output.**

  - Correct input data: satisfies precondition
  - Correct output data: satisfies postcondition

Proving correctness

- An algorithm = a list of actions
- ***Proving that an algorithm is totally correct:***
  1. ***Proving that it will terminate***
  2. ***Proving that the list of actions applied to the precondition imply the postcondition***

  – This is easy to prove for simple sequential algorithms
  – This can be complicated to prove for repetitive algorithms (containing loops or recursivity)
    - use techniques based on ***loop invariants*** and ***induction***

Example – a sequential algorithm

```
Swap1(x,y):
    aux := x
    x := y
    y := aux
```

Precondition:

x = a and y = b

Postcondition:

x = b and y = a

Proof: ***the list of actions applied to the precondition imply the postcondition***

1. Precondition:
   x = a and y = b
2. aux := x => aux = a
3. x : = y => x = b
4. y := aux => y = a
5. x = b and y = a is the Postcondition

Example – a repetitive algorithm

Algorithm Sum_of_N_numbers

Input: a, an array of N numbers

Output: s, the sum of the N numbers in a

```
s:=0;
k:=0;
While (k<N) do
  k:=k+1;
  s:=s+a[k];
end
```

Proof: the list of actions applied to the precondition imply the postcondition

BUT: we cannot enumerate all the actions in case of a repetitive algorithm !

We use techniques based on <u>loop invariants</u> and <u>induction</u>