# Verification of Functions



- Specification of a system as a set of functions where the internal state is hidden.
- Each function is specified as a set of pre and post conditions.
- Pre-condition must hold if the post-condition is to be true.

# Example – minimum

**function min (X: in INTEGER_ARRAY) return INTEGER**

**Pre: True**

**Post: ∃ j in X'First .. X'Last : min(X) = X( j ) and**

**∀ i in X'First .. X'Last : min(X) ≤ X(i) and X = X"**

# Specification of functions using pre- and post-conditions



**Procedure Search (X: in INTEGER_ARRAY;**

**key: in INTEGER;**

**found: in out Boolean;**

**index: in out INTEGER);**

**Pre: True**

**Post: ((found and X(index) = key) or**

**(NOT found and**

**($\forall$ j in X'First … X'Last : x( j ) $\neq$ key)) and**

**(X = X''))**

# Specification of functions using pre- and post-conditions



**Procedure binarySearch (X: in INTEGER_ARRAY;**

**key: in INTEGER;**

**found: in out Boolean;**

**index: in out INTEGER);**

**Pre:** $\forall$ **j in X'First … X'Last-1 : x( j ) <= x( j+1 )**

**Post: ((found and X(index) = key) or**

**(NOT found and**

**($\forall$ j in X'First … X'Last : x( j ) $\neq$ key)) and**

**(X = X"))**

```
Procedure binary_Search (X: in INTEGER_ARRAY;
                   key: in INTEGER;
                   found: in out Boolean;
                   index: in out INTEGER)
begin
     bot: INTEGER := X'First;
     top: INTEGER := X'Last;
     mid: INTEGER;
     index := (bot + top) / 2;
     found := X(index) = key ;
     while (bot <= top AND NOT found) loop
     begin
          mid := (bot + top)/2;
          if X(mid) = key then
               found := TRUE;
               index := mid;
          elsif X(mid) < key then
               bot := mid + 1;
          else
               top := mid - 1;
          end if;
     end loop;
end binary_Search;
```

## Loop Invariant

**(found AND X(index) = key) OR**
**(NOT found AND**
**$\forall$ j in X'First..bot-1, top+1..X'Last:  X(j) $\neq$ key))**

## Pre Condition

X'Last >= X'First and Ordered(X)

```
begin
    bot: INTEGER := X'First;
    top: INTEGER := X'Last;
    mid: INTEGER;
    index := (bot + top) / 2;
    found := X(index) = key ;
```

## Loop Invariant

(found AND X(index) = key) OR
(NOT found AND
$\forall$ j in X'First..bot-1, top+1..X'Last: X(j) $\neq$ key))

## Loop Invariant

(found AND X(L) = key) OR
(NOT found AND
$\forall$ j in X'First..bot-1, top+1..X'Last:  X(j) $\neq$ key))

```
while (bot <= top AND NOT) found loop
begin
    mid := (bot + top)/2;
    if X(mid) = key then
        found := TRUE;
        L := mid;
    elsif X(mid) < key then
        bot := mid + 1;
    else
        top := mid - 1;
    end if;
end loop;
```

found AND X(index) = key

$\forall$ j in X'First..bot-1: X(index) $\neq$ key

$\forall$ j in top+1..X'Last:  X(index) $\neq$ key

(found AND X(index) = key) OR
(NOT found AND $\forall$ j in X'First..X'Last:
X(index) $\neq$ key))

# Dijkstra's Guarded if Statement



```
if c1 →  S1
[] c2 →  S2
[] c3 →  S3
fi
```

# Dijkstra's Guarded if Statement



if b then S else T

```
if b → S
[] not b → T
fi
```

# Conditional Rule



```
{P}
if b1 → S1
[] b2 → S2
fi
{Q}
```

is equivalent to conjunction of the three propositions:

$P \Rightarrow b1 \lor b2$

$\{P \land b1\}\ S1\ \{Q\}$

$\{P \land b2\}\ S2\ \{Q\}$

# Constructing Conditional Statements



- {P} S {Q} -- P and Q are given, we want to calculate S
- Three step process:

1. Split the precondition into two (or possibly more cases) b1 and b2. That is identify b1 and b2 such that

   $P \Rightarrow b1 \vee b2$

2. Construct a program statement S1 that guarantees termination in a state satisfying Q given the precondition $P \wedge b1$

3. Construct a program statement S2 that guarantees termination in a state satisfying Q given the precondition $P \wedge b2$

# Constructing Conditional Statements



1. Split the precondition into two (or possibly more cases) b1 and b2. That is identify b1 and b2 such that

   $P \Rightarrow b1 \vee b2$

2. Construct a program statement S1 that guarantees termination in a state satisfying Q given the precondition $P \wedge b1$

3. Construct a program statement S2 that guarantees termination in a state satisfying Q given the precondition $P \wedge b2$