

## What Does this Program Do?

```
Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
```

How can you be sure?

## Answer

- Integer division
  - Compute quotient **Q** and remainder **R** of **X** divided by **Y**, for non-negative integer **X** and positive integer **Y**
  - Expressed as a function returning two results **<Q, R>** ← **DIVIDE(X, Y)**
  - Expressed as a relation of four variables **DIVIDE(X, Y, Q, R)**

## Preconditions

- What must be true about the inputs to this program in order for the program to successfully execute?

```
Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
```

## Preconditions

- $X \geq 0 \wedge Y > 0$ 
  - The value of **X** is non-negative and
  - The value of **Y** is positive before execution begins

```
Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
```

## Postconditions

- What must be true about the program output variables after the program has completed execution?
  - Expressed in terms of input and output variables
  - Assuming that it terminates

```
Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
```

## Postconditions

- $Y > 0 \wedge$
- $X \geq 0$

```
Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
```

## Postconditions

- $Y > 0 \wedge$
- $X \geq 0 \wedge$
- $Q \geq 0$

```

Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
    
```

## Postconditions

- $Y > R \geq 0 \wedge$
- $X \geq 0 \wedge$
- $Q \geq 0$

```

Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
    
```

## Postconditions

- $Y > R \geq 0 \wedge$
- $X \geq 0 \wedge$
- $Q \geq 0 \wedge$
- $X = Q * Y + R$

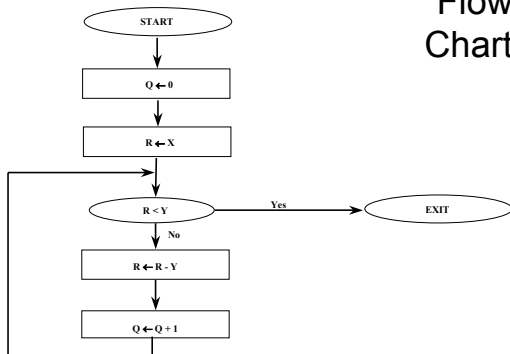
```

Q ← 0
R ← X
while (R ≥ Y)
  R ← R - Y
  Q ← Q + 1
    
```

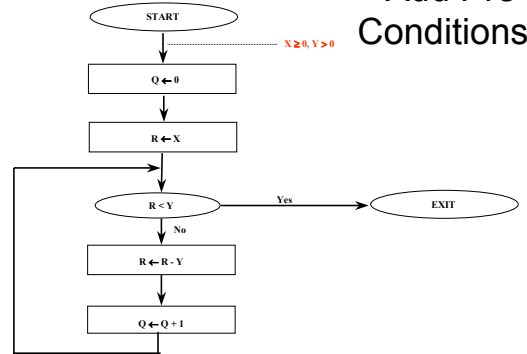
## Proof Plan

- Construct flow chart
- Annotate with preconditions
- Add invariants at intermediate program points based on the type of statement executed
  - Assignment
  - Conditional
  - Loop

## Flow Chart

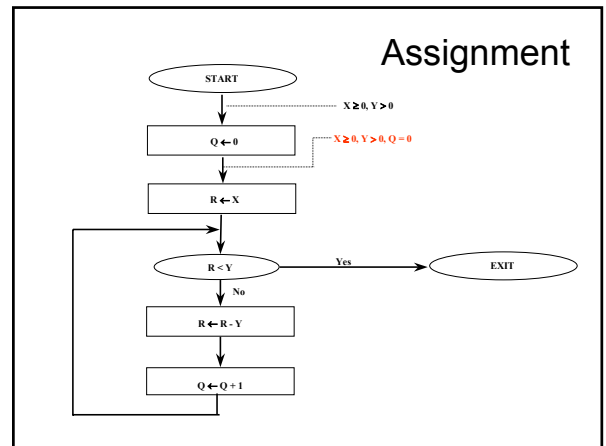


## Add Pre-Conditions



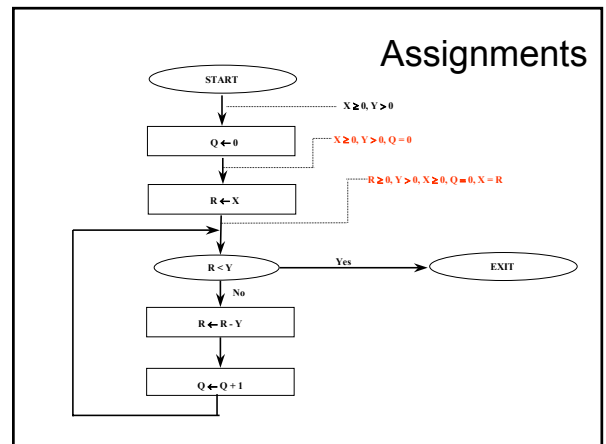
## Simple Assignment

- The statement  $\{Q \leftarrow 0\}$  guarantees that  $Q$  has the value  $0$  after it executes
- Moreover, all other variables remain unchanged
  - Assuming no aliasing of name
- Therefore, the invariant after the assignment execution looks like  $X \geq 0, Y > 0, Q = 0$



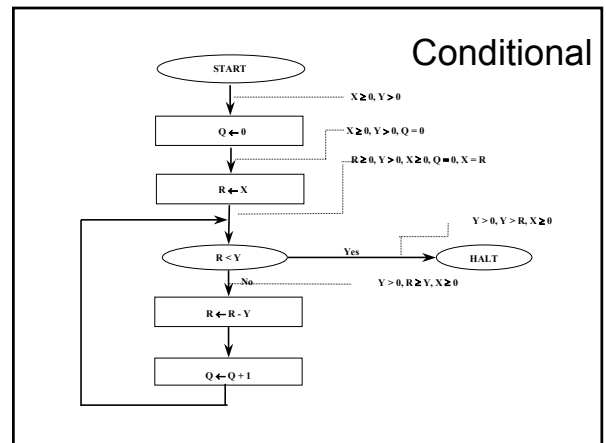
## Slightly More Complex Assignment

- The statement  $\{R \leftarrow X\}$  guarantees that  $R$  has the same value as  $X$  after it executes
  - So,  $X \geq 0, Y > 0, Q = 0$  becomes  $X \geq 0, Y > 0, Q = 0, R = X$
- We can also use the laws of algebra to manipulate other assertions
  - The invariant is algebraically equivalent to  $R \geq 0, Y > 0, X \geq 0, Q = 0, X = R$



## Conditionals

- A conditional statement, like an assertion, describes a possible program state
- Execution branches leading out of a conditional statement can be labeled with an assertion corresponding to the truth or falsity of the Boolean condition tested
- In the example, the conditional tests the value of the expression  $R < Y$
- On the true (**Yes**) branch leading out from this test,  $R < Y$ , can be conjoined with the assertions coming into the conditional
  - Likewise for the **No** branch



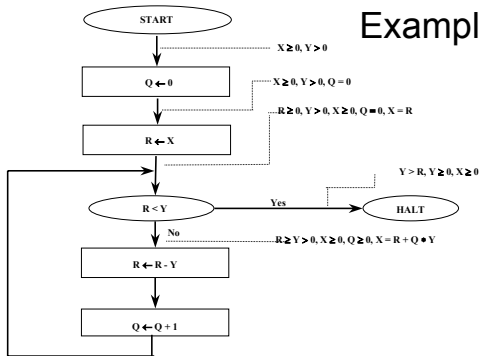
## Loops

- Unlike other statements, a loop, like the one in the example, has to deal with multiple incoming flows of control
  - That is, there are two ways of entering the loop
    - One from the start of the program
    - One after going around the loop at least once
- The statements inside the loop must be true in both circumstances
- In fact, they need to be true no matter how many times the loop is executed
- For this reason, they are called *loop invariants*
  - Normally, you think of loops behaving differently on each iteration
  - But the assertion has to stay the same
- That is, the loop invariant has to generalize over all iterations
  - The analyst has to invent this generalization

## More on Loop Invariants

- A loop invariant has to satisfy three properties
  - It must be true the first time execution reaches it
  - If it is true after some number ( $n$ ) of iterations, it must be true after  $n + 1$
  - It must be strong enough to imply the postcondition
- Recall, the post condition we are looking for is
  - $Y > R \geq 0 \wedge X \geq 0 \wedge Q \geq 0 \wedge X = R + Q * Y$
- We already have  $Y > 0, Y > R, X \geq 0$
- Let's try  $R \geq Y > 0, X \geq 0, Q \geq 0, X = R + Q * Y$

## Example



## More Invariants

- Let's try our first test. Is the invariant true the first time through
  - Condition before entering the loop  
 $R \geq 0, Y > 0, X \geq 0, Q = 0, X = R$
  - Loop invariant  
 $R \geq Y > 0, X \geq 0, Q \geq 0, X = R + Q * Y$
  - $R \geq 0$  and  $Y > 0$  and  $R \geq Y$  implies  $R \geq Y > 0$ ; so we are okay so far
  - $X$  still non-negative
  - If  $Q$  equal to  $0$ , then it is certainly greater than or equal to it
  - Finally, if  $Q = 0$  and  $X = R$  then  $X$  does equal  $R + Q * Y = R + 0 * Y = R$

## Assignments One Last Time

- The algorithm includes the assignment statement  $R \leftarrow R - Y$
- What is interesting about this statement is that the variable on the left hand side ( $R$ ) also occurs on the right hand side
- If we naively state that the postcondition is  $\{R = R - Y\}$ , we would get nonsense
- Instead, we must introduce a little more notation and perform some algebraic manipulations

## Assignments - 2

- Assume that the precondition for the assignment statement is  $\{X = R + Q * Y\}$  and the assignment statement  $R \leftarrow R - Y$
- First, using the assignment statement, annotate the left hand  $R$  with a prime ( $R'$ )
  - $R'$  can be read as "the value of  $R$  after the assignment"
- Then, solve for  $R$  in terms of  $R'$ :  $R' = R - Y \Rightarrow R' + Y = R$
- Substitute this expression ( $R' + Y$ ) into the precondition for all occurrences of  $R$ :  $\{X = (R' + Y) + Q * Y\}$
- Simplify to produce the postcondition (drop the prime):  $\{X = R + (Q + 1) * Y\}$
- The general rule is to solve for the variable without the apostrophe and plug that expression into the precondition



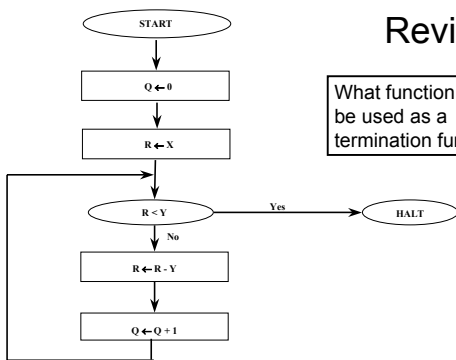
## Termination

- There is one other detail to clear up
- We have proved that the program is correct (matches its specification), assuming that it terminates
- We have to also prove this
- Assignments and conditions always progress, so loops are the concern
- The way to show that loops terminate is to show that they always make progress

## Termination of Loops

- You can show that loops terminate using the following method
- Define a function with the following properties
  - The domain of the function is the set of program variables
  - The range of the function is the integers
  - The value computed by the function gets smaller on every loop iteration
  - When the loop exit condition is reached, the function value is negative
- That is, you are relying on the following property of the integers
  - Any constantly decreasing series of integer values must eventually become negative

## Example Revisited



What function  $f$  can be used as a termination function?

## EXAMPLE - TERMINATION

- For the example program, the termination function is:  
 $f(X, Y, Q, R) = R - Y$
- While the loop continues to execute (on the **No** branch):
  - $R \geq Y \Rightarrow R - Y \geq 0 \Rightarrow f$  is always non-negative
  - The line  $R \leftarrow R - Y$  gets executed on every loop iteration
  - $R$  and  $Y$  are both positive at all times
  - This implies that  $R$  gets smaller on every loop iteration
  - Consequently  $f$  gets reduced on every loop iteration
- On the **Yes** branch:  $R < Y \Rightarrow R - Y < 0 \Rightarrow f$  is negative
- Consequently, the loop must eventually terminate

## Another Exercise

### What Does this Program Do?

Assume  $B$  is a sorted vector of ints of length  $n$

```
I := 1;
P := 1;
while (I != N)
  if (B[I] != B[I - P])
    I := I + 1;
  else
    I := I + 1;
    P := P + 1;
  end if;
end while;
```

## Answer

- $P$  is the length of the longest plateau in  $B$ , where a plateau is a sequence of equal values

## Step 2

- Provide a convincing argument, in English, that this is so