Output Variables

{true} S {i = j}

i := j;   or   j := i;

- Which one is the input variable and which one is the output variable?

Ghost Variables

- Suppose we want to specify that the sum of two variables i and j should remain constant
- We specify this by introducing a ghost variable C
- This variable should not be used anywhere else in the program.
- Then S specified by

$\{i + j = C\}$ S $\{i + j = C\}$

Simultaneous Assignment

- The left side is a list of variables and the right side is a list of expressions of the same length as the list of variables.

x, y, z := 2*y, x+y, 3*z

Simultaneous Assignment

- The assignment

  x, y := y, x

  has the effect of

Calculating Assignments

- Suppose that the requirement is to maintain the value of **j + k** constant while incrementing **k** by **1**
- Our task is to calculate and expression X such that

  {j + k = C} j , k := X, k + 1 { j + k = C}
- Applying the assignment axiom, we get

  {X + k + 1 = C} j , k := X, k + 1 { j + k = C}

  j+k = C ⇒ X + k + 1 = C

- Now

 j + k

= j + k + 1 – 1

= (j - 1) + k + 1

- Thus a suitable value of **X** is **j – 1**
- So,

 { j + k = C} j , k := j – 1, k + 1 { j + k = C}

- Suppose variables **s** and **n** satisfy the property

  $\mathbf{s = n^2}$

- We want to increment **n** by 1 whilst maintaining this relationship between s and n
- Our goal is to calculate an expression **X** involving only addition such that

$$\{ s = n^2 \}\ s, n := s + X, n+1\ \{ s = n^2 \}$$

- Applying the assignment axiom we get

$$\{ s + X = (n + 1)^2 \}\ s, n := s + X, n + 1\ \{ s = n^2 \}$$

- So, we need

$s = n^2 \Rightarrow s + X = (n + 1)^2$

- Now

$(n + 1)^2$

$= \quad n^2 + 2*n + 1$

- That is

$s = n^2 \Rightarrow s + 2*n + 1 = (n + 1)^2$

- So,

$\{ s = n^2 \}\ s, n := s + 2*n + 1, n+1\ \{ s = n^2 \}$

Max of two numbers

- two number x and y

  $\mathbf{max(x,y) = x \equiv y \leq x}$

  $\mathbf{max(x,y) = y \equiv x \leq y}$

- pre-condition: **{true}**
- post-condition: **{z = max(x,y)}**
- two cases:

  $\mathbf{(x \leq y)}$ **and** $\mathbf{(y \leq x)}$

  $\mathbf{\{true\} \Rightarrow (x \leq y) \vee (y \leq x)}$

Max of two numbers


```
{true}
if x ≤ y → z := e1 {z = max(x,y)}
[] y ≤ x → z := e2 {z = max(x,y)}
fi
{z = max(x,y)}
```

Max of two numbers

using the assignment axiom we calculate that

$\{e = \max(x,y)\}\ z := e\ \{z = \max(x,y)\}$

in particular

$\{x = \max(x,y)\}\ z := x\ \{z = \max(x,y)\}$

that is

$\{y \leq x\}\ z := x\ \{z = \max(x,y)\}$

and

$\{x \leq y\}\ z := y\ \{z = \max(x,y)\}$

we have thus determined e1 and e2

Iteration

- The do-od statement
- do b $\rightarrow$ S od

Constructing Loops

- Invariant property and bound function
- Loops are designed so that each iteration of the loop body maintains the invariant whilst making progress to the required post-condition by always decreasing the bound function.

Constructing Loops

- Suppose a problem is specified by precondition P and post-condition Q.
- We identify and invariant property *inv* and the bound function *bf*.

Constructing Loops

- The bound function is an integer-value function of the program variables and is a measure of the size of the problem to be solved.
  - It is guaranteed to be greater than zero when loop is executed.
  - A guarantee that the value of such a bound function is always decreased at each iteration is a guarantee that the loop will terminate.

Constructing Loops

- The post condition Q is split into a termination condition, say *done*, and the invariant property *inv*, in such a way that

  $inv \wedge done \Rightarrow Q$

- The invariant property is designed, in combination with the termination condition, by generalizing the required post-condition.

- The termination condition is typically related to the bound function.

Constructing Loops

- The invariant should also guarantee that the value of the bound function is greater than zero unless the loop has terminated. That is:

  $inv \Rightarrow (bf > 0) \vee done$

- The invariant property is chosen so that it is easy to design an initialization statement, S, that establishes the invariant property

  $\{P\}\ S\ \{inv\}$

Constructing Loops

- The design is completed by constructing a loop body T that maintains the invariant whilst making progress towards the termination condition.

$$\{inv \wedge \neg done \wedge bf = C\}\ T\ \{inv \wedge (done \vee bf < C)\}$$

Constructing loops

- If the termination condition, done, the bound function, bf, the invariant, inv, and the loop body, T, have all been constructed as above, then we have

  $\{inv\}$ do $\neg$done $\rightarrow$ T od $\{Q\}$

- Moreover, if S has been constructed then

  $\{P\}$ S; do $\neg$done $\rightarrow$ T od $\{Q\}$

Dutch Flag Problem

- Three Boolean valued functions red, blue, and white
- red(i) evaluate to true if the ball at index i is red. Similarly blue(i) and white(i)
- swap(i,j) swaps the balls in bucket i and j

Dutch Flag Problem

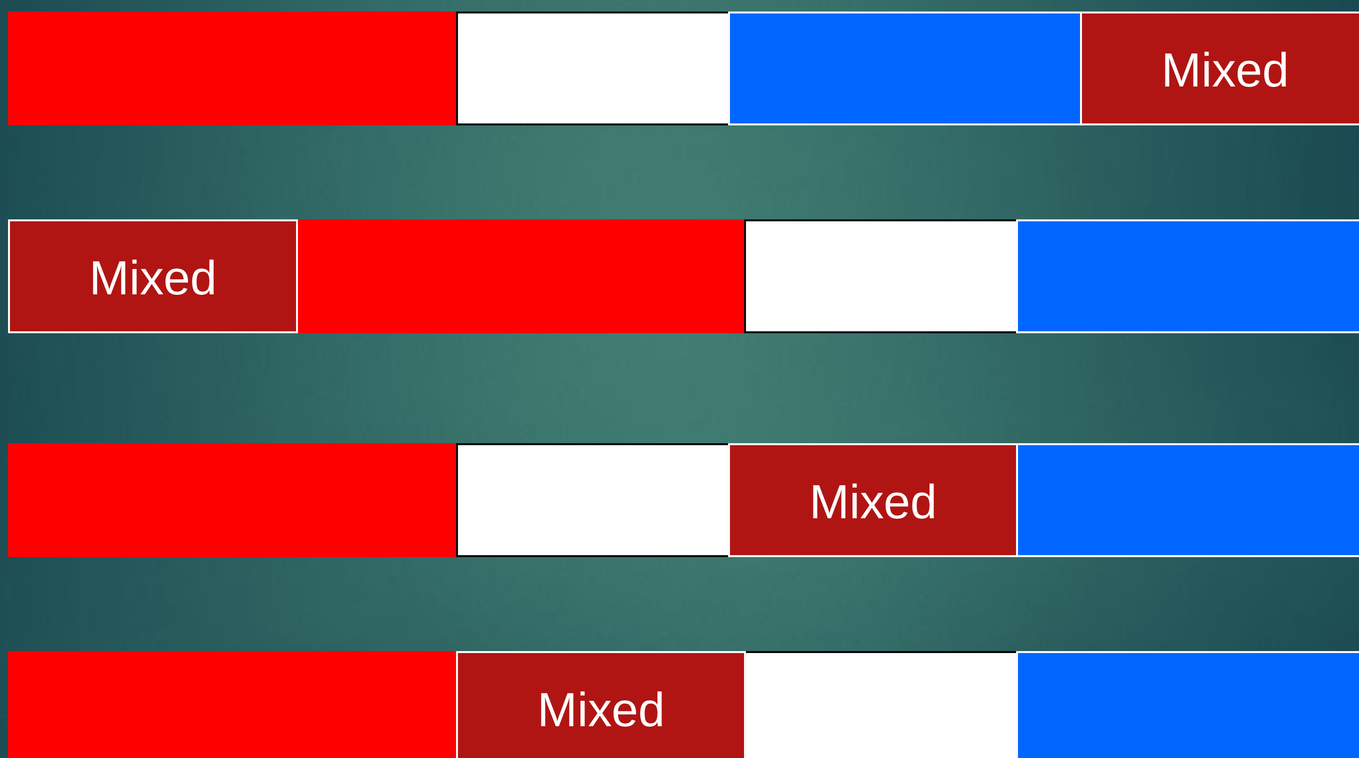
- r and w are two indices such that
- $0 \le r \le w \le N$
- $\wedge\ (\forall i:\ \ 0 \le i < r:\ \ red(i))$
- $\wedge\ (\forall i:\ \ r \le i < w:\ \ white(i))$
- $\wedge\ (\forall i:\ \ w \le i < N:\ blue(i))$

Invariant

- Partition the array into four segments, three of the segments corresponding to the three colors and containing values of that color only, and the fourth containing a mixture of colors.
- Final state: the mixed partition is empty.

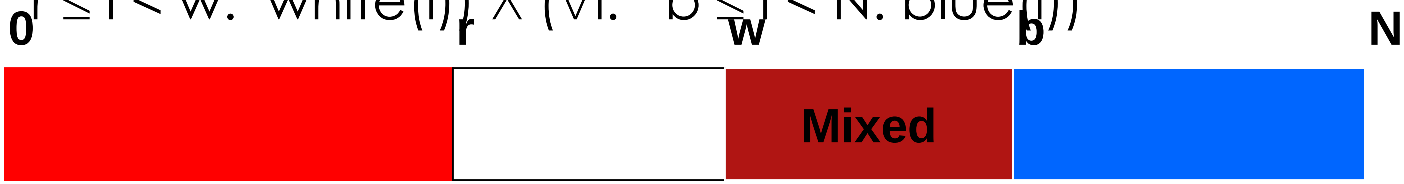
Invariant

- Four Possibilities.



Invariant, initial, and termination condition

- $(0 \le r \le w \le b \le N) \wedge (\forall i: \; 0 \le i < r: \; red(i)) \wedge$
- $(\forall i: \; r \le i < w: \; white(i)) \wedge (\forall i: \; b \le i < N: blue(i))$



- Initial condition: r=0; w=0; b=N;
- Bound Function: b-w
- Done/Termination Condition: w=b

Algorithm

- Progress requires reducing the size of the mixed segment by at least one in each iteration
- Examine the color of the ball at index w and put it in its right partition
- Three cases:

1. white(w) → w := w + 1;
2. red(w) → swap(r, w); r := r + 1; w := w + 1;
3. blue(w) → swap(b - 1,w); b := b – 1;

- $\{0 \le N\}$
- r:= 0; w := 0; b := N;
- {Invariant:
- $0 \le r \le w \le b \le N$
- $\wedge\ (\forall i:\ \ 0 \le i < r:\ \ red(i))$
- $\wedge\ (\forall i:\ \ r \le i < w:\ \ white(i))$
- $\wedge\ (\forall i:\ \ b \le i < N: blue(i))$
- Bound Function: b – w}
- do $w < b \rightarrow$
- if white(w) $\rightarrow$ w := w + 1;
- [] blue(w) $\rightarrow$ swap(b - 1,w); b := b – 1;
- [] red(w) $\rightarrow$ swap(r, w); r := r + 1; w := w + 1;
- fi
- od
- $\{0 \le r \le w \le N$
- $\wedge\ (\forall i:\ \ 0 \le i < r:\ \ red(i))$
- $\wedge\ (\forall i:\ \ r \le i < w:\ \ white(i))$
- $\wedge\ (\forall i:\ \ w \le i < N: blue(i))$
- }

Quiz #2

- Prove that for following program and post condition

  wp(x:= x+1; y := y+1, x = y)

  x=y is the suitable pre-condition.

- Solve following for weakest pre-condition.

  WP(if i <= j then m := i; else m := j, (m ≤ i and m ≤ j) and (m = i or m = j))

  wp(if x>2 then y:=3*x else y:=2*x, (y >= x)>= 0)

- Find loop invariant for following

- ```
  a = 0; i = 0;
  while (i < N)
      a = a + i++;
  ```

```
s = 0;
for i := 1 to n do
    s = s + a[i];
```
```