Recap

{ P }
Point c = null;
int z;
if (y < 0) {
  z = -2*y;
} else {
  z = x;
}
if (z > 10) {
  c = new Point(z,y);
}
{ Q: c != null }

From last time:
• weakest == most permissive
• strongest == most restrictive

What is the weakest precondition P that ensures postcondition Q?

Program Verification

• Weakest Preconditions
  – most permissive assumptions to ensure postcondition is satisfied.

• Verifying functional correctness

  // requires P
  // ensures Q
  method test(...) {
      S
  }

Prove:
P => wp(S, Q)

Example in Dafny

method Test(x : int, y : int) returns (c : Point?)
  requires P;
  ensures c != null;
  {
      c := null;
      var z;
      if y < 0 {
          z := -2 * y;
      } else {
          z := x;
      }
      if z > 10 {
          c := new Point(z,y);
      }
  }
Loops

{\text{n} \geq 0}
i = 0;
y = 0;
{P: \text{n} \geq 0 \land i = 0 \land y = 0}
while(i \neq n) {
i = i+1;
y = y+i;
}
{Q: y = \text{sum}(1, n)}

Loops in Hoare Logic

{\text{n} \geq 0}
i = 0;
y = 0;
{P: \text{n} \geq 0 \land i = 0 \land y = 0}
{\text{invariant: } y = \text{sum}(1,i)}
while(i \neq n) {
i = i+1;
y = y+i;
}
{Q: y = \text{sum}(1, n)}

Version 2

{\text{n} \geq 0}
i = 1;
y = 0;
{P: \text{n} \geq 0 \land i = 1 \land y = 0}
{\text{invariant: } y = \text{sum}(1,i)}
while(i \neq n) {
i = i+1;
y = y+i;
}
{Q: y = \text{sum}(1, n)}
### Version 2

```plaintext
{n \geq 0}
i = 1;
y = 0;
{P: n \geq 0 \land i=1 \land y=0}
{invariant: y = \text{sum}(1,i-1)}
while(i \neq n) {
    i = i+1;
y = y+i;
}
{Q: y = \text{sum}(1,n)}
```

### Version 3

```plaintext
{n \geq 0}
i = 1;
y = 0;
{P: n \geq 0 \land i=1 \land y=0}
{invariant: y = \text{sum}(1,i-1)}
while(i \neq n+1) {
    i = i+1;
y = y+i;
}
{Q: y = \text{sum}(1,n)}
```

### Version 4: Self Check

```plaintext
{n \geq 0}
i = 1;
y = 0;
{P: n \geq 0 \land i=1 \land y=0}
{invariant: y = \text{sum}(1,i-1)}
while(i \neq n) {
    y = y+i;
i = i+1;
}
{Q: y = \text{sum}(1,n)}
```

### Too Strong? Too Weak? Just Right?

- **Loop invariant is too strong:**
  - may not hold on entry.
  - may not be preserved by body
- **Loop invariant is too weak:**
  - can’t prove what you want after the loop
- No automatic procedure for conjuring a loop-invariant...
  - Think about invariant while writing the code
  - If proof doesn’t work, invariant or code or both may need work
Methodology

1. Decide on the invariant first
   - What describes the milestone of each iteration?
2. Write a loop body to maintain the invariant
3. Write the loop test so "false implies postcondition"
4. Write initialization code to establish invariant

Example

Set max to hold the largest value in array items

2. Write a loop body to maintain the invariant
   // i: max holds largest value in items[0..k-1]
   while( ) {
     // i holds
     if(max < items[k]) {
       max = items[k]; // breaks i temporarily
     }
     // max holds largest value in items[0..k]
     k = k+1; // i holds again
   }

Example

Set max to hold the largest value in array items

3. Write the loop test so false-implies-postcondition
   // i: max holds largest value in items[0..k-1]
   while(k != items.count) {
     // i holds
     if(max < items[k]) {
       max = items[k]; // breaks i temporarily
     }
     // max holds largest value in items[0..k]
     k = k+1; // i holds again
   }
   // max holds largest value in items[0..items.count-1]
Example

Set \texttt{max} to hold the largest value in array \texttt{items}

4. Write initialization code to establish invariant
   \begin{verbatim}
   k = 1; max = items[0];
   // i: max holds largest value in items[0..k-1]
   while(k != items.count) {
     // i holds
     if(max < items[k]) {
       max = items[k]; // breaks i temporarily
     }
     // max holds largest value in items[0..k]
     k = k+1; // i holds again
   } // max holds largest value in items[0..items.count-1]
   \end{verbatim}

Example

Edge case!

\begin{verbatim}
   k = 1; max = items[0];
   // i: max holds largest value in items[0..k-1]
   while(k != items.count) {
     // i holds
     if(max < items[k]) {
       max = items[k]; // breaks i temporarily
     }
     // max holds largest value in items[0..k]
     k = k+1; // i holds again
   } // max holds largest value in items[0..items.count-1]
\end{verbatim}

Quotient and Remainder

Set \texttt{q} to be the quotient of \texttt{x/y} and \texttt{r} to be the remainder

\begin{verbatim}
{Pre: x>0 \land y>0}

{Invariant}:
   while ( ) {
     //
   }

{Post: x=q*y+r \land 0 \leq r < y}
\end{verbatim}

Dutch National Flag (classic)

Given an array of red, white, and blue pebbles, sort the array so the red pebbles are at the front, white are in the middle, and blue are at the end

– [Can only swap pebbles, not count them...]

Edsger Dijkstra
Pre- and Post-conditions

Precondition: Any mix of red, white, and blue

Postcondition:
- Red, then white, then blue
- Number of each color same as in original array

More Precise

- Precondition: \( a \) contains red, white, blue
- Postcondition:
  \[ 0 \leq i \leq j < a\text{.count} \quad \land \quad a[0..i-1] \text{ is red} \quad \land \quad a[i..j-1] \text{ is white} \quad \land \quad a[j..a\text{.count-1}] \text{ is blue} \]
- Invariant:
  \[ 0 \leq i \leq j \leq k \leq a\text{.count} \quad \land \quad a[0..i-1] \text{ is red} \quad \land \quad a[i..j-1] \text{ is white} \quad \land \quad a[k..a\text{.count-1}] \text{ is blue} \]

Some Potential Invariants

The Code

```java
i = 0;
j = 0;
k = a.count;
while (j!=k) {
    if(a[j] == White) {
        j = j+1;
    } else if (a[j] == Blue) {
        swap(a,j,k-1);
        k = k-1;
    } else { // a[j] == Red
        swap(a,i,j)
        i = i+1;
        j = j+1;
    }
}"
```
Termination

- **Quotient-and-remainder**
  - $r$ (starts positive, gets strictly smaller)
- **Binary search**
  - size of range still considered
- **Dutch-national-flag**
  - size of range not yet partitioned ($k - j$)
- **Search in a linked list**
  - length of list not yet considered