Beispiel

Wir möchten zu $n > 0$ ein k finden mit $k^2 \leq n < (k+1)^2$.

```
static int sqrt(int n) {
  assert n > 0; // Vorbedingung
  int l = 1, r = n;
  while(r - l > 1) {
    int k = (l + r)/2;
    if(k * k <= n) {
      l = k;
    }
    else {
      r = k;
    }
  }
  assert l * l <= n && n < (l + 1) * (l + 1); // Nachbedingung
  return l;
}
```

Programme können neben dem eigentlichen Ergebnis weitere Informationen liefern, die eine Überprüfung der Korrektheit des Ergebnissen erleichtern.

Beispiel: Berechnung des größten gemeinsamen Teilers

```
int gcd1(int a, int b) {
  while(b > 0) {
    int temp = b;
    b = a%b;
    a = temp;
  }
  return a;
}
```

Dies ist der euklidische Algorithmus.

Wie überprüfen, daß Ergebnis korrekt?

Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler.

Der *erweiterte* euklidische Algorithmus berechnet den größten gemeinsamen Teiler zweier Zahlen a und b und zusätzlich zwei ganze Zahlen x und y mit

$$xa + yb = gcd(a, b).$$

Falls a und b Vielfache von r sind und $xa + yb = r$, dann *muß* $r = gcd(a, b)$ sein.

Mithilfe von x und y können wir also die Korrekheit beweisen.

```
int gcd2(int a, int b) {
  assert a > 0 && b > 0;  // Vorbedingung
  int x = 1, y = 0, xx = 0, yy = 1, quot, temp;
  int r = a, rr = b;
  while (rr != 0) {
    quot = r/rr;
    temp = rr; rr = r - quot * rr; r = temp;
    temp = xx; xx = x - quot * xx; x = temp;
    temp = yy; yy = y - quot * yy; y = temp;
  }
   // Nachbedingung
  assert x * a + y * b == r && a%r == 0 && b%r == 0;
  return r;
}
```

Die **assert**-Anweisung überprüft eine Bedingung.

Vorsicht: Assertions müssen im Java-Laufzeitsystem aktiviert werden (durch `java -ea GCD`).

Java unterstützt Contract Programming nicht direkt. Beispiel in D:

```
int gcd(int a, int b, out int x, out int y)
in {assert(a > 0 && b > 0); }
out(r) {assert(x * a + y * b == r && a%r == 0 && b%r == 0); }
body {
   x = 1; y = 0;
   int xx = 0, yy = 1, quot, temp;
   int r = a, rr = b;
   while (rr != 0) {
      quot = r/rr;
      temp = rr; rr = r - quot * rr; r = temp;
      temp = xx; xx = x - quot * xx; x = temp;
      temp = yy; yy = y - quot * yy; y = temp;
   }
   return r;
}
```

Contract Programming – Zusammenfassung

- Hilft Fehler zu lokalisieren.
- Auch wenn Programm schon Jahre in Verwendung.
- Testen der Nachbedingung muß *effizient* sein.
- Benötigt *Zertifizierende Algorithmen*.
- Außer Vor- und Nachbedingung auch *Invarianten*.

Ein Zertifizierender Algorithmus liefert neben dem Ergebnis ein *Zertifikat*, das eine einfache Überprüfung erlaubt.