Method contracts

```
void p()
   /*
   requires Pre
   ensures Post
  */
{
   s1;

  while (e)
  /*: invariant I */
  {
    s2;
  }

  s3;
}
```

boolean expressions

- usually written in the same syntax
- pure (no side-effects)
- can use logical connectives

  !, &&, ||, ==>, <==>
- quantifiers

  forall, exists
- special keywords

  e.g. sum, min, max, count, …

Other constructs

- **result**
  - method's return value (used in postcondition)
- **old(x)**
  - value of x before entry to method (used in postcondition)

- **modifies x**
  - frame condition: variable x may be modified in method (usually implicitly no other)
  - defaults: modify all or modify none, depending on convention

- **assert**
  - anywhere in code
  - run-time checks
  - redundant, but useful for code understanding

- **assume**
  - hints for the prover
  - "escape" solution (what happens when you use **assume(false)** in wrong code?)

- **object invariants (more complex)**
  - conditions on object variables that have to hold everywhere

Ex 1 (warm-up)

```
class Mult {

  int fi(int x, int y)
    requires x >= 0 & y >= 0;
    ensures result == x * y;
  {
    int r = 0;
    int i = 0;
    while (i < y)
      invariant ... ;
    {
      i = i + 1;
      r = r + x;
    }
    return r;
  }

}
```

Ex 1 (warm-up)

```
class Mult {

  int fi2(int x, int y)
    requires x >= 0 & y >= 0;
    ensures result == x * y;
  {
    int r = 0;
    int i = y;
    while (i > 0)
      invariant ...;
    {
      i = i - 1;
      r = r + x;
    }
    return r;
  }
}
```

Ex 2 – part 1 *What's missing?*

```
class Account {
  private int checkingBalance;
  private int savingsBalance;

  private void checkingDeposit(int amount)
    ensures checkingBalance == old(checkingBalance) + amount;
  {
    checkingBalance = checkingBalance + amount;
  }

  private void checkingWithdraw(int amount)
    ensures checkingBalance == old(checkingBalance) - amount;
  {
    checkingBalance = checkingBalance - amount;
  }

  private void savingsDeposit(int amount)
    ensures savingsBalance == old(savingsBalance) + amount;
  {
    savingsBalance = savingsBalance + amount;
  }
 …
}
```

Ex 2 – part 2

```
class Account {
  private int checkingBalance;
  private int savingsBalance;
  …

  private void transferSavingsToChecking(int amount)
    requires amount > 0 && savingsBalance >= amount;
    ensures savingsBalance == old(savingsBalance)  - amount;
    ensures checkingBalance == old(checkingBalance) + amount;
    ensures old(checkingBalance) + old(savingsBalance) == checkingBalance +
    savingsBalance;
  {
    savingsBalance = savingsBalance - amount;
    checkingBalance = checkingBalance + amount;
  }

  private void transferCheckingToSavings(int amount)
    requires amount > 0 && checkingBalance >= amount;
    ensures old(checkingBalance) + old(savingsBalance) == checkingBalance +
    savingsBalance;
  {
    checkingWithdraw(amount);
    savingsDeposit(amount);
  }
}
```

Why does it fail to verify?

Ex 3 *Find the invariant!*

```
class Sqrt {

  public int sqrt(int x)
    requires 0 <= x;
    ensures result*result <= x && x < (result+1)*(result+1);
  {
    int r = 0;
    while ((r+1)*(r+1) <= x)
      invariant ... ;
    {
      r++;
    }
    return r;
  }

}
```

Ex 4 *Find the invariant!*

```
class LinearSearch {
  bool search(int[] a, int key)
    ensures result == exists{int i in (0: a.Length); a[i] == key};
  {
    int n = a.Length;
    do
      invariant ...;

    {
      n--;
      if (n < 0) {
        break;
      }
    } while (a[n] != key);
    return 0 <= n;
  }
}
```

Ex 5 *Find the error!*

```
class SomeFunction1 {

  public int f(int x, int y)
    requires 0 <= x && 0 <= y;
    ensures result == 2*x + y;
  {
    int r = x;
    for (int n = 0; n < y; n++)
      invariant r == x+n && n <= y;
    {
      r++;
    }
    return r;
  }
}
```

Ex 6 *Find the invariant!*

```
class GCD {

  int gcd(int a, int b)
    requires a > 0 && b > 0;
    ensures ...;
    ensures ...;
    {
      int i = 1; int res = 1;
      while (i < a+b)
        invariant ...;
      {
        i++;
        if (a % i == 0 && b % i == 0) {
          res = i;
        }
    }
    return res;
  }
}
```