While rule

A while statement includes a condition $C$ and a body of statements $S$. In addition, there is an invariant $I$ (a property expressed like a condition) which must be true (i) before entering the while loop, (ii) at the end of each iteration of the while loop and (iii) immediately after the while loop is terminated. A more formal definition of ‘while rule’ follows:

$$
\begin{align*}
\{P \Rightarrow I\} & \quad \{I \land C\} \quad S \quad \{I\} & \quad \{I \land \neg C \Rightarrow Q\} \\
\{P\} \textbf{while } C \textbf{ do } S \quad \{Q\}
\end{align*}
$$

In the above rule, $P$ indicates a precondition to arrive at the while loop, and $Q$ refers to the post-condition that indicates the consequence of executing the code inside the while loop.

Example 1: Consider the following while loop. Assume that $r \in \mathbb{N} \land y \in \mathbb{N} \land q \in \mathbb{N}$.

$q = 0;
\text{while } (y <= r) \{ \\
\quad r = r - y; \\
\quad q = q + 1;
\}
$

For this while loop, one can derive the following:

$C \equiv y \leq r$
$S \equiv r = r - y; \ q = q + 1$
$Q \equiv q \ast y + r == x$ where $x$ is the initial value of $r$.
$P \equiv r == x$

The invariant for this loop is actually the same as the post-condition, namely $x == q \ast y + r$. Thus,

$I \equiv x == q \ast y + r$

Let us now prove that the invariant is true (i) just before the loop, (ii) at the end of $k^{th}$ iteration of the loop, and (iii) when the loop terminates.

Since the precondition should hold before entering into the loop, $r == x$ is true.

Let the initial value of $y$ be $y_0$. Before entering the loop, the statement $q = 0$ is executed and so $q == 0$ is true. So,

$x == q_0 \ast y_0 + r_0$
$== 0 + x$

$q_0, y_0$ and $r_0$ refer to their initial values respectively.
Therefore, the invariant is true before entering the loop.

At the end of the first iteration, \( q = 1, r = r_0 - y_0 \).

\[
x = 1 \times y_0 + (r_0 - y_0) \\
  = r_0 \\
  = x
\]

Therefore, the invariant is true at the end of the first iteration.

This can be generalized to \( k \) iterations as well.
At the end of the \( k^{th} \) iteration, \( q = k, r = r_0 - k \times y_0 \).

\[
x = k \times y_0 + (r_0 - k \times y_0) \\
  = r_0 \\
  = x
\]

Hence, the invariant is true at the end of the \( k^{th} \) iteration as well.

At the exit of the loop, assume that the loop has been executed \( N \) times.
Substitute \( k = N \) in the previous step; this will prove that the invariant is true at the exit of the loop.

Therefore, \( x = q \times y + r \) is an invariant for this while loop.

Finding Invariants
As seen from the above example, verification of code involving while loop relies in proving that the invariant associated with the while loop is indeed invariant (i.e., it does not change because of the number of iterations of the code). Finding the invariant for the while loop is not an easy task. The invariant heavily depends on the application domain. It is also possible that there could be several invariants for the same loop - some of them may be weaker than others.

Let us try to find an invariant for the while loop in the following problem, and prove it. The code below computes the factorial of \( N \), a natural number (\( \mathbb{N} \)).

```cpp
int i = 0;
int fact = 1;
while (i != N) {
    i = i + 1;
    fact = fact * i;
}
```

From the code, it is easy to observe that at the completion of each iteration, \( fact = i! \). So, this is a loop invariant. Let us next prove this invariant.
Using the rule for \textbf{while} loop,
\[ P \equiv N \in \mathbb{N}, \ i == 0 \quad fact == 1 \]
\[ C \equiv (i! = N) \]
\[ S \equiv i = i + 1; \quad fact = fact \ast i \]
\[ Q \equiv fact == N! \]

At the beginning of the loop,
\[ fact == i! \]
\[ == 0! \]
\[ == 1 \]

This is confirmed by the precondition \( fact == 1 \).

At the end of \( k^{th} \) iteration, \( i == k \)
\[ fact = i! \Rightarrow fact = k! \]. This is true because \( i \) starts at 1 (after the statement \( i = i + 1 \)), and goes up to \( k \), inclusive. Consequently, \( fact \) is the result of multiplication of every value from 1 to \( k \), inclusive (as evident from the statement \( fact = fact \ast i \)).

At the exit of the loop, \( i == N \)
By the same argument above, \( fact = N! \).

Therefore, the invariant \( fact = i! \) is true for this loop.

\textbf{Exercises}

1. The following code is written to find an integer \( k \) within an array of integers \( arr \).

\begin{verbatim}
int i = 0;
boolean flag = false;
while (!flag \&\& i < arr.length) {
    if (arr[i] == k) {
        flag = true;
    }
    else {
        i = i + 1;
    }
}
\end{verbatim}

Find a loop invariant for this problem and prove its correctness.

2. This problem is called 2-color Dutch National Flag Construction. Given an array of two colors, RED and BLUE, the program is supposed to rearrange the colors in such
a way that all the RED colors come before all the BLUE colors. For simplicity of programming, we represent the RED color by ‘0’ and the BLUE color by ‘1’ so that our array consists of zeroes and ones. We randomly initialize the array with zeroes and ones. Following is the program that performs the re-arranging.

```java
int[] A = new int[25];
// initialize A with random number of 0 and 1.
for (int i = 0; i < A.length; i++)
    if ((int)(Math.random()*10) >= 5)
        A[i] = 1;
    else A[i] = 0;
// rearrange the colors
int m = 0, k = 0, temp = 0;
while (m != A.length){
    if (A[m] == 0){
        // swap A[k] with A[m]
        temp = A[k];
        A[k] = A[m];
        A[m] = temp;
        k++;
    }
    m++;
}
```

Prove that “the entries from A[0] to A[k-1] will all be zeroes (or RED)” is a loop invariant for the `while` loop in this code.