Examples of creating loops

1. Create a program that sums the first n positive integers up and has postcondition $s = sum(0, n)$.

   1) First, let's try to find some possible loop invariants. Here, replacing a constant by a variable seems to be the best idea. There are two constants 0 and n in the expression so there are two ways to replace.

      a. If we replace n by a variable k, we get $s = sum(0, k)$. Since we need the sum of the first n integers, and k will equal to n after the loop, so we can initialize $k = 0$ and increase it in each iteration until $k = n$. And we will get a loop looks like this:

         $\{\textbf{inv } s = sum(0,k) \wedge 0 \le k \le n\}\{\textbf{bd } n - k\}$
         while $k \ne n$ **do**
             ... make k larger and something else ...
         od
         $\{s = sum(0,k) \wedge 0 \le k \le n \wedge k = n\}$ $\# \, p \wedge \neg B$
         $\{s = sum(0,n)\}$

      b. If we replace 0 by a variable k, we get $s = sum(k, n)$. Since we need the sum of the first n integers, and k will be equal to 0 after the loop, so we can initialize $k = n$ and decrease it in each iteration until $k = 0$. And we will get a loop looks like this:
         $\{\textbf{inv } s = sum(k,n) \wedge 0 \le k \le n\}\{\textbf{bd } k\}$
         while $k \ne 0$ **do**
             ... make k smaller and something else ...
         od
         $\{s = sum(k,n) \wedge 0 \le k \le n \wedge k = 0\}$ $\# \, p \wedge \neg B$
         $\{s = sum(0,n)\}$

   - When we replace a constant c with a variable k, we need to consider the range of values of k can be. We usually end the program with $k = c$, so we need another variable d so that k has the range $[c, d]$ (or $[d, c]$, depend on whether k is increased or decreased in each iteration).

   2) Next, let's consider the precondition of the loop.

      a. If we end with $k = n$, together with $0 \le k \le n$, we are most likely starting the loop with $k = 0$. Then the precondition needs to imply that $s = sum(0,k) \wedge 0 \le k \le n \wedge k = 0$. Then:

         $\{s = 0 \wedge n \ge 0 \wedge k = 0\}$
         $\{\textbf{inv } s = sum(0,k) \wedge 0 \le k \le n\}\{\textbf{bd } n - k\}$
         while $k \ne n$ **do**
             ... make k larger and something else ...
         od
         $\{s = sum(0,k) \wedge 0 \le k \le n \wedge k = n\}$
         $\{s = sum(0,n)\}$

      b. If we end with $k = 0$, together with $0 \le k \le n$, we are most likely starting the loop with $k = n$. Then the precondition needs to imply that $s = sum(k,n) \wedge 0 \le k \le n \wedge k = n$. Then:

         $\{s = n \wedge n \ge 0 \wedge k = n\}$

$\{\mathbf{inv}\ s = sum(k,n) \land 0 \le k \le n\}\{\mathbf{bd}\ k\}$
$\mathbf{while}\ k \ne 0\ \mathbf{do}$
  ... make k smaller and something else ...
$\mathbf{od}$
$\{s = sum(k,n) \land 0 \le k \le n \land k = 0\}$
$\{s = sum(0,n)\}$

3) Then, let's consider the loop body. Other than updating the variable k, we also need $\{p \land B\}\ S\ \{p\}$ being valid.
   a. k will be increased by 1 after each iteration, and we need $s = sum(0,k)$ in the loop invariant. Thus, we can update $s \coloneqq s + sum(0,k+1) - sum(0,k)$. Then:

   $\{s = 0 \land n \ge 0 \land k = 0\}$
   $\{\mathbf{inv}\ s = sum(0,k) \land 0 \le k \le n\}\{\mathbf{bd}\ n-k\}$
   $\mathbf{while}\ k \ne n\ \mathbf{do}$
     $s \coloneqq s + k + 1; k \coloneqq k + 1$
   $\mathbf{od}$
   $\{s = sum(0,k) \land 0 \le k \le n \land k = n\}$
   $\{s = sum(0,n)\}$

   b. k will be decreased by 1 after each iteration, and we need $s = sum(k,n)$ in the loop invariant. Thus, we can update $s \coloneqq s + sum(k-1,n) - sum(k,n)$. Then:

   $\{s = n \land n \ge 0 \land k = n\}$
   $\{\mathbf{inv}\ s = sum(k,n) \land 0 \le k \le n\}\{\mathbf{bd}\ k\}$
   $\mathbf{while}\ k \ne 0\ \mathbf{do}$
     $s \coloneqq s + k - 1; k \coloneqq k - 1$
   $\mathbf{od}$
   $\{s = sum(k,n) \land 0 \le k \le n \land k = 0\}$
   $\{s = sum(0,n)\}$

   - In the above example, we can see that once we decide a loop invariant, the rest of the loop comes naturally.

- We have seen an example in which we find loop invariants by replacing a constant / expression by a variable. The steps include:
  - Choose a constant c / expression e in the postcondition and replace it with some variable x.
  - Think about the range of the values of the variable x and usually one boundary of this range is c (or the value of e).
  - Let the loop ends with $x = c$ or $x = e$ and let the loop start with x equals the other boundary of the range.

- We can also try to create a loop invariant by adding some disjuncts or removing some conjuncts.
  - Adding disjuncts can be very open-ended; but since we need $p \land \neg B \Rightarrow q$, we can try for different loop condition B and let $p \equiv q \lor B$, then we have both $p \land \neg B \Rightarrow q$ and $q \Rightarrow p$. The loop will look like:

    $\{\mathbf{inv}\ q \lor B\}\{\mathbf{bd}\ ...\}$
    $\mathbf{while}\ B\ \mathbf{do}$
      $\{(q \lor B) \land B\}$
      $loop\ body$
      $\{q \lor B\}$

od
$\{(q \vee B) \wedge \neg B\}$
$\{q\}$

- Removing conjuncts can be used when the postcondition is a conjunction $q \equiv q_1 \wedge q_2 \wedge \ldots \wedge q_n$, where $n \geq 2$. It is natural to try to drop some of q_k to get a loop invariant candidate: $p_k \equiv q_1 \wedge q_2 \wedge \ldots q_{k-1} \wedge q_{k+1} \wedge \ldots \wedge q_n$. Then the loop looks like:

  $\{\mathbf{inv}\ p_k\}\{\mathbf{bd} \ldots\}$
  while $\neg q_k$ **do**
  $\quad\{p_k \wedge \neg q_k\}$
  $\quad$*loop body*
  $\quad\{p_k\}$
  od
  $\{p_k \wedge q_k\}\{q\}$

- At the end of the day, adding disjuncts and removing conjuncts are the safe trick: if we have postcondition $p \wedge q$ and we add disjunct $(p \wedge \neg q)$ we will get $(p \wedge q) \vee (p \wedge \neg q) \Leftrightarrow p \vee (q \wedge \neg q) \Leftrightarrow p$; this is equivalent to removing the conjunct q.

2. Create a program that represents the linear search for x in an array slice $b[0 \ldots n-1]$ (note that, in our language, we don't have the expression $b[0 \ldots n-1]$ to represent the first n indices of b). The precondition is that array b has at least n elements ($n \geq 0$) and the value x may or may not appear in $b[0 \ldots n-1]$. The postcondition should be k equals to the index of the leftmost occurrence of x in $b[0 \ldots n-1]$; if x is not found then let $k = n$.

   1) Let us start with creating a postcondition. Let us define $x \notin b[0 \ldots n-1]$ with a predicate function $NotIn(x, b, n) \equiv \forall k. 0 \leq k < n \rightarrow x \neq b[k]$.

      We notice that no matter whether x is in $b[0 \ldots n-1]$ or not, we always have $NotIn(x, b, k)$ if k is in returned index. Thus, postcondition can be written as:
      $$0 \leq k \leq n \wedge NotIn(x, b, k) \wedge (k < n \rightarrow b[k] = x)$$
      $$\Leftrightarrow 0 \leq k \wedge k \leq n \wedge NotIn(x, b, k) \wedge (k < n \rightarrow b[k] = x)$$

   2) The postcondition is a conjunction, we can try to create a loop invariant by dropping some conjuncts. There are four conjuncts, and this means that we can try four candidates.
      a. If we drop off $0 \leq k$, then in the loop body we will have $k < 0$, and this is out of the bound of an array index. This is not a good idea.
      b. Similarly, if we drop of $k \leq n$, then in the loop body we will have $k > n$, which is not a guaranteed index in array b.
      c. Dropping off $NotIn(x, b, k)$ means the loop condition means **while** $x \in b[0 \ldots k-1]$ (note that this is not a legal expression). The problem is how do we start this loop? If we start with $k = 0$, then we are check whether x is in an array slice of length 0, which is fine; but then we need to check with $k = 1$, and we need $x = b[0]$, but it is not guaranteed. If we start with $k = n$, then we need $x \in b[0 \ldots n-1]$, which is also not guaranteed. So, this is not a good idea.

      d. Dropping off $k < n \rightarrow b[k] = x$ could work. The loop condition will become $\neg(k < n \rightarrow b[k] = x) \Leftrightarrow k < n \wedge b[k] \neq x$. We can get a partial outline look like follows:

$\{n \geq 0\}$ ...
$\{\mathbf{inv}\ 0 \leq k \leq n \land NotIn(x,b,k)\}$ # p
$\{\mathbf{bd}$ ... $\}$
$\mathbf{while}\ k < n \land b[k] \neq x\ \mathbf{do}$ # B
$\{0 \leq k \leq n \land NotIn(x,b,k) \land k < n \land b[k] \neq x\}$ # $p \land B$
$loop\ body$
$\{0 \leq k \leq n \land NotIn(x,b,k)\}$ # p
$\mathbf{od}$
$\{0 \leq k \leq n \land NotIn(x,b,k) \land (k < n \to b[k] = x)\}$ # $p \land \neg B \Leftrightarrow q$

3) We can start the loop with $k = 0$, so the precondition of the loop is $0 \leq k \leq n \land NotIn(x,b,k) \land k = 0$. In each iteration, we simply increase k.

$\{n \geq 0\}\ k := 0; \{n \geq k = 0\}$ # forward assignment
$\{\mathbf{inv}\ 0 \leq k \leq n \land NotIn(x,b,k)\}$
$\{\mathbf{bd}$ ... $\}$
$\mathbf{while}\ k < n \land b[k] \neq x\ \mathbf{do}$
$\{0 \leq k \leq n \land NotIn(x,b,k) \land k < n \land b[k] \neq x\}$
$\{0 \leq k + 1 \leq n \land NotIn(x,b,k+1)\}$ # backward assignment
$k := k + 1$
$\{0 \leq k \leq n \land NotIn(x,b,k)\}$
$\mathbf{od}$
$\{0 \leq k \leq n \land NotIn(x,b,k) \land (k < n \to b[k] = x)\}$

4) $n - k$ is a good loop bound expression. Then we full proof outline of program of linear search as follows:

$\{n \geq 0\}\ k := 0; \{n \geq k = 0\}$
$\{\mathbf{inv}\ 0 \leq k \leq n \land NotIn(x,b,k)\}\ \{\mathbf{bd}\ n - k\}$
$\mathbf{while}\ k < n \land b[k] \neq n\ \mathbf{do}$
$\{0 \leq k \leq n \land NotIn(x,b,k) \land k < n \land b[k] \neq n \land n - k = t_0\}$
$\{0 \leq k + 1 \leq n \land NotIn(x,b,k+1) \land n - (k+1) < t_0\}$ # backward assignment
$k := k + 1$
$\{0 \leq k \leq n \land NotIn(x,b,k) \land n - k < t_0\}$
$\mathbf{od}$
$\{0 \leq k \leq n \land NotIn(x,b,k) \land (k < n \to b[k] = n)\}$