Gap Buffer

A gap buffer is a data structure for the implementation of text editors, which can efficiently move the cursor, as well add and delete characters.

The idea is simple: the editor's content is represented as a character array a of length n, which has a gap of unused entries $a[l], \ldots, a[r-1]$, with respect to two indices $l \leq r$. The data it represents is composed as $a[0], \ldots, a[l-1], a[r], ..., a[n-1]$.

The current cursor position is at the left index l, and if we type a character, it is written to $a[l]$ and l is increased. When the gap becomes empty, the array is enlarged and the data from r is shifted to the right.

Implementation task. Implement the following four operations in the language of your tool: Procedures `left()` and `right()` move the cursor by one character; `insert()` places a character at the beginning of the gap $a[l]$; `delete()` removes the character at $a[l]$ from the range of text.

```
procedure left()
    if l != 0 then
        l := l - 1
        r := r - 1
        a[r] := a[l]
    end-if
end-procedure
```

```
procedure right()
    // your task: similar to left()
    // but pay attention to the
    // order of statements
end-procedure
```

```
procedure insert(x: char)
    if l == r then
        // see extended task
        grow()
    end-if
    a[l] := x
    l := l + 1
end-procedure
```

```
procedure delete()
    if l != 0 then
        l := l - 1
    end-if
end-procedure
```

Verification task. Specify the intended behavior of the buffer in terms of a contiguous representation of the editor content. This can for example be based on strings, functional arrays, sequences, or lists. Verify that the gap buffer implementation satisfies this specification, and that every access to the array is within bounds.

Hint: For this task you may assume that `insert()` has the precondition $l < r$ and remove the call to `grow()`. Alternatively, assume a contract for `grow()` that ensures that this call does not change the abstract representation.

Extended verification task. Implement the operation `grow()`, specify its behavior in a way that lets you verify `insert()` in a modular way (i.e. not by referring to the implementation of `grow()`), and verify that `grow()` satisfies this specification.

Hint: You may assume that the allocation of the new buffer always succeeds. If your tool/language supports copying array ranges (such as `System.arraycopy()` in Java), consider using these primitives instead of the loops in the pseudo-code below.

```
procedure grow()
    var b := new char[a.length + K]

    // b[0..l] := a[0..l]
    for i = 0 to l - 1 do
        b[i] := a[i]
    end-for

    // b[r + K..] := a[r..]
    for i = r to a.length - 1 do
        b[i + K] := a[i]
    end-for

    r := r + K
    a := b
end-procedure
```

Resources

- https://en.wikipedia.org/wiki/Gap_buffer
- http://scienceblogs.com/goodmath/2009/02/18/gap-buffers-or-why-bother-with-1