Implementation of Basic List Operations in $O(1)$ Time

CREATE: Define a new header, set the first and last fields to nil, set the size field to 0.

DESTROY($L$): Reclaim the memory used by the list header.

EMPTY($L$): Test if the header size field is 0 (or if the first or last field is nil).

SIZE($L$): Return this field of the header.

FIRST($L$), LAST($L$): Return these fields of the header of $L$.

DATA($v$, $L$), NEXT($v$, $L$), PREV($v$, $L$): Return these fields of $v$. 
(implementation of basic operations continued)

\[\text{INSERT}(d,v,L), \quad \text{SPLICE}(L1,v,L2) \quad \text{"hook in"} \quad (\text{and update the header size field}):\]

\[\text{INSERT} \quad \text{SPLICE} \]

\[d \]

\[v \]

\[L1: \]

\[v \]

\[\bullet \]

\[\bullet \]

\[\bullet \]

\[\bullet \]

\[v \]

\[v \]

\[v \]

\[v \]
(implementation of basic operations continued)

DELETE($v, L$), CUT($v_1, v_2, L$) — "bypass" (and update the header size field):

[Diagram of the deletion and cutting operations is shown, illustrating the changes to the linked list structure.]
Notation for "low-level" operations:

If \( p \) is a pointer to a list vertex and \( L \) a pointer to a list header, we use the following notation:

\[
\begin{align*}
  p->data &= \text{data field} \\
  p->next &= \text{next field} \\
  p->prev &= \text{previous field} \\
  L->size &= \text{size field} \\
  L->first &= \text{first field} \\
  L->last &= \text{last field}
\end{align*}
\]

The following two pages show the details of how INSERT and DELETE could be implemented \textit{(without error checking)}.

**NOTE:** We are getting down to a low enough level that this pseudo code looks very much like C, but be careful, it is still not C (e.g., something like if \( x=y \) ... would be if \( x==y \) ... in C).
procedure INSERT(d,v,L):
  Allocate memory for a new vertex, pointed to by x.
  x->data := d
  L->size := (L->size) + 1
  if (L->first) = nil then begin (L is an empty list)
    x->prev = x->next = nil
    L->first = L->last = x
  end
  else if v = nil then begin (x goes at the front of L)
    x->prev = nil
    x->next = L->first
    (L->first)->prev = x
    L->first = x
  end
  else if v = (L->last) then begin (x goes at the end of L)
    x->prev = v
    x->next = nil
    v->next = x
    L->last = x
  end
  else begin (x goes in the middle of L)
    x->prev = v
    x->next = v->next
    (v->next)->prev = x
    v->next = x
  end
end
procedure DELETE(v,L):
\( d := \text{DATA}(v,L) \)
\( L->\text{size} := (L->\text{size}) - 1 \)
\( \text{if } (L->\text{first}) = (L->\text{last}) \text{ then begin (v is the only vertex in } L) \)
\( \quad L->\text{first} = \text{nil} \)
\( \quad L->\text{last} = \text{nil} \)
\( \text{end} \)
\( \text{else if } v = (L->\text{first}) \text{ then begin (v is the first item of } L) \)
\( \quad L->\text{first} = v->\text{next} \)
\( \quad (v->\text{next})->\text{prev} = \text{nil} \)
\( \text{end} \)
\( \text{else if } v = (L->\text{last}) \text{ then begin (v is the last item of } L) \)
\( \quad L->\text{last} = v->\text{prev} \)
\( \quad (v->\text{prev})->\text{next} = \text{nil} \)
\( \text{end} \)
\( \text{else begin (v is in the middle of } L) \)
\( \quad (v->\text{prev})->\text{next} = v->\text{next} \)
\( \quad (v->\text{next})->\text{prev} = v->\text{prev} \)
\( \text{end} \)
Reclaim the memory that was used by the vertex pointed to by \( v \).
\( \text{return } d \)
end