Arrays

- Data stored in $n$ consecutive words of memory.
- Array variable contains address of first word.
- Access $a[i]$ by computing address $a + i$ then reading or writing there.

Advantages:
- Constant access time,
- no memory overhead,
- coherent memory access.

Disadvantage: inflexible.
Dynamic Arrays

- Data stored in first $n$ of $m$ consecutive words.
- Array stores $n$, $m$, and address $a$ of first word.
- When a value is added, $n$ is incremented.
- If $n$ equals $m$, the data is copied to the first $m$ of $2m$ newly allocated words, the old memory is freed, $m$ is doubled, and $a$ is updated.

\[
\begin{align*}
  n = 1 & \quad m = 1 & [7] \\
  n = 2 & \quad m = 2 & [7, 3] \\
  n = 3 & \quad m = 4 & [7, 3, 5, -] \\
  n = 4 & \quad m = 4 & [7, 3, 5, 1] \\
  n = 5 & \quad m = 8 & [7, 3, 5, 1, 2, -, -, -]
\end{align*}
\]
Dynamic Arrays

- Tiny memory overhead to store $n$ and $m$.
- Read time is constant.
- Write time is constant for $n < m$ and is proportional to $n$ for $n = m$.
- Writing $n = 2^k$ values causes $k$ memory doublings.
- Cost is proportional to
  \[ 1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1 = n - 1. \]
- The average overhead is constant!
Lists

- A node stores a value and an address.
- A list variable stores the address of its first node.
- Each node stores the address of the next node.
- The last node stores a null address.

\[1 \rightarrow 2 \rightarrow 3\]

- A value is appended by creating a new node and setting the address of the old last node.

\[1 \rightarrow 2 \rightarrow 3 \rightarrow 4\]

new

- A value can be inserted anywhere in the list similarly.
Lists

- Memory overhead is one address per value.
- Read and append times are proportional to list length.
- Memory is not coherent.