What is the goal ?

Keyboards track key presses using a matrix, where a "1" in a position mapped to a key indicates that the key is pressed. Keyboards can also detect whether a key is currently down or up. On older, inexpensive keyboards, there is a phenomenon called a "phantom key," where pressing three keys in certain positions can cause the keyboard to falsely register an additional, non-existent key. The goal is to simulate this behavior at a low level.

What is the math ?

- Let N be the number of row lines and P the number of column lines.
- A keyboard state at discrete time (scan index) $t \in \mathbb{Z}_{\geq 0}$ is a matrix

$$M_t \in \{0,1\}^{N \times P},$$

  where

$$M_t[i,j] = \begin{cases} 1 & \text{if the circuit at row } i \text{ and column } j \text{ is closed (interpreted as “key pressed”)} \\ 0 & \text{if it is open (interpreted as “key not pressed”).} \end{cases}$$

- Define the **key-label map**

$$K : \{1,\ldots,N\} \times \{1,\ldots,P\} \to \mathcal{L}$$

  that assigns to each matrix position (i,j) a key label $K(i,j)$ from the set of labels $\mathcal{L}$ (e.g. letters, modifiers).

- The set of keys reported as pressed at time t is

$$S_t = \{\, K(i,j) \mid M_t[i,j] = 1 \,\} \subseteq \mathcal{L}.$$

  (Equivalently, identify S_t with the set of index pairs $\{(i,j) \mid M_t[i,j] = 1\}$.)

Using consecutive scans M_{t-1} and M_t:

- **Press events** (keys that went down at scan t):

$$\text{Press}_t = S_t \setminus S_{t-1} = \{K(i,j) \mid M_{t-1}[i,j] = 0,\ M_t[i,j] = 1\}.$$

- **Release events** (keys that went up at scan t):

$$\text{Release}_t = S_{t-1} \setminus S_t = \{K(i,j) \mid M_{t-1}[i,j] = 1,\ M_t[i,j] = 0\}.$$

- **Held keys** at t (still down):

$$\text{Held}_t = S_t \cap S_{t-1}.$$

These are exact set-theoretic definitions using the matrices.

Define the difference matrix $\Delta_t \in \{-1, 0, 1\}^{N\times P}$ by

$$\Delta_t[i,j] = M_t[i,j] - M_{t-1}[i,j].$$

Then

- $\Delta_t[i,j] = 1$ means a press at (i,j) between $t-1$ and t.
- $\Delta_t[i,j] = -1$ means a release.
- $\Delta_t[i,j] = 0$ means no change.

Press/release sets can be read off Δ_t directly.

The ideal model above assumes the read matrix equals the true physical state. In practice a keyboard controller **scans** rows/columns and the observed matrix M_t^{obs} can differ from the true set of simultaneously pressed keys M_t^{true} because of electrical interactions (ghosting) unless diodes are present.

We can express observation as a (hardware-dependent) function

$$\varphi : \{0,1\}^{N\times P} \to \{0,1\}^{N\times P}, \qquad M_t^{\text{obs}} = \varphi(M_t^{\text{true}}).$$

Properties / examples:

- If diodes are present and the scanner reads each switch independently, typically φ is the identity: $\varphi(M) = M$.
- Without diodes, φ can produce **spurious 1s** (ghost keys). A canonical ghosting condition:

  If there exist distinct rows $i_1 \neq i_2$ and distinct columns $j_1 \neq j_2$ such that

  $$M^{\text{true}}[i_1, j_1] = M^{\text{true}}[i_1, j_2] = M^{\text{true}}[i_2, j_1] = 1,$$

  then the scanner may also observe $M^{\text{obs}}[i_2, j_2] = 1$ even if $M^{\text{true}}[i_2, j_2] = 0$. In words: three pressed corners of a rectangle can make the fourth appear pressed.

You can model ghosting formally by specifying φ; e.g. φ could be defined to set the logical closure along rows/columns according to Kirchhoff-type rules. For many algorithmic purposes it suffices to note that: **if $\varphi \neq \text{Id}$ then derived press/release events from M^{obs} may be incorrect.**

Switches bounce, so a single scan flip may be transient. Model debouncing by requiring stability over a window of w consecutive scans:

- Define a key (i, j) to be *stably pressed* at time t if

$$M_{t'}[i, j] = 1 \quad \text{for all } t' \in \{t - w + 1, \ldots, t\}.$$

- Similarly for *stably released* if all those entries are 0.

Then generate events only when the stability condition is satisfied. This can be encoded as a filtered matrix $\widetilde{M_t}$ derived from recent M-values.

- Reading one scan: $O(NP)$ work to inspect the whole matrix.
- Computing press/release events between scans: $O(k)$ where k is the number of entries that differ (or $O(NP)$ worst case).
- Maintaining stable-state with window w requires $O(w \cdot NP)$ naive memory/time, but can be implemented efficiently with counters per key (one counter per (i, j)).

What is the algorithm ?

Keyboard State Representation

- The keyboard is represented as a two-dimensional matrix of fixed size.
- Each position in the matrix corresponds to a physical key.
- Each entry in the matrix has a binary value:
  - **1** indicates that the corresponding key's circuit is closed (key is pressed).
  - **0** indicates that the circuit is open (key is not pressed).

Algorithm Overview

At each scan cycle, the algorithm operates on the current matrix and, when available, the immediately preceding matrix. The following steps are performed:

1. **Identify Active Keys**
   - Traverse the current matrix to determine which entries are set to "pressed."
   - Map those entries to their corresponding key labels to obtain the set of active keys.
2. **Generate Key Events**
   - Compare the current active keys with the previously recorded set of active keys.

- If a key appears in the current set but not in the previous set, register a **key press event**.
- If a key appears in the previous set but not in the current set, register a **key release event**.
- Keys present in both sets are considered **held keys** (no new event generated).

3. **Detect Phantom Keys**
    - Analyze the pattern of simultaneously pressed keys to detect possible phantom states (ghosting).
    - A phantom key is defined as a key that appears pressed in the matrix but cannot be unambiguously attributed to a physical key press (e.g., when three corners of a rectangle are pressed, and the fourth is falsely detected as pressed).
    - When phantom keys are detected, they are either flagged as invalid or excluded from the set of valid active keys, depending on the system's design.

Event Output

- The algorithm outputs a structured list of events for each scan cycle.
- Each event includes:
    - The key label,
    - The event type (press, release, hold),
    - The scan cycle or timestamp.

The overall architecture :

Here's a clear, formalized version of your text:

The goal of this project is to implement everything from the ground up, completely from scratch, as it is intended to be a low-level project. The implementation will be done using standard C.

The project will include custom implementations of:

- A 2×2 matrix to represent the keyboard state,
- A dynamic list structure,
- A set structure,
- A lookup table for mapping matrix positions to key labels.