Represent the Maze with an Array of Bit Vectors

We can **represent the maze with an array**:

```
static uint8_t maze[10][10];
```

Each space in the array **is a bit vector** composed of the following bits:

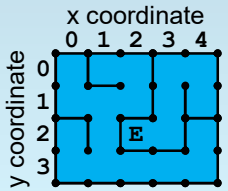
- // **1** – the space has a **left wall**
- // **2** – the space has a **right wall**
- // **4** – the space has an **upper wall**
- // **8** – the space has a **lower wall**
- // **16** – the space is the **exit**

Do You Understand the Representation?

(Reminder: L=1, R=2, U=4, D=8, E=16)

For example,

- `maze[0][0] is 7 (1 | 2 | 4)`
- `maze[0][1] is 9`
- `maze[3][1] is 3`
- `maze[4][2] is 7`
- `maze[2][2] is 29`

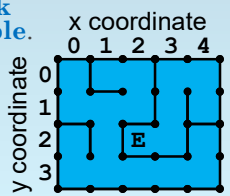


Outline for a Recursive Solution

Let's solve the problem recursively.

Here's the approach:

- **Keep track of reachable locations.**
- Write **a function to mark one location as reachable**.
- Within the function, **call the same function** to mark all "children" (adjacent reachable neighbors) as reachable.



Represent Reachable Locations with a Second Array

Track reachable locations with a second array:

```
static uint8_t found[10][10];
```

Each element is either:

- **0** – the space has **not** been **found**/reached
- **1** – the space has been **found**/reached

And we use one variable for the exit:

```
static int32_t saw_exit;
```

(Both of these should be initialized to all 0s.)

