Let's Use Recursion to Evaluate Nim

Here's how our function works:
- **given the number of sticks**
- in each of the three piles,
- the function **`nim` returns**
- the **value of the game**.

Since Nim is a zero-sum game,
- **1** can represent the **first player winning**,
- and **-1** can represent the **second player**.

Value is the Maximum of Negated Recursive Evaluations

In **`nim`**,
- the **current player makes one move**
- then **calls `nim` to evaluate** the new piles.

Since Nim is a zero-sum game,
- the **value returned** by the recursive call
- **is simply negated**:
- the value to one player
- is negative the value to the other player.

The **value of the game is the maximum value over all possible moves** (the best move).

Piles are Empty? Current Player Has Lost

```
int32_t nim (int32_t p[3])
{
    int32_t max = -2;
    int32_t pnum;
    int32_t count;
    int32_t value;
    if (0 == p[0] && 0 == p[1] &&
        0 == p[2]) {
        return -1;
    }
```

max move value seen

pile for a move

sticks for a move

value for a move

stopping condition: piles are empty

Try Every Possible Move and Choose the Best

```
    for (pnum = 0; 3 > pnum; pnum++) {
        for (count = 1;
             p[pnum] >= count;
             count++) {
            // Try one move
            // and update max.
        }
    }
    return max;
}
```

Try each valid number of sticks.

Try moves for each pile.

Return best move.

