University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Developing a Data Structure

Let's Develop a Data Structure

Let's **develop a data structure together**.

When I was in graduate school,
and the Internet was new (not really),
we played backgammon on FIBS,
the First International Backgammon Server.
It was fun!

Let's Build the First International Blocky Server!

I'm thinking that our MP6 could be **big**.
I see ~~ten~~ ~~a hundred~~ a million
people flocking to a server
to play,
to watch master players play,
and to hang out
and talk about Blocky strategy.
I'm serious!

Start with Some Brainstorming

Let's start with some brainstorming:

- What aspects/information do we want about each player?
- What aspects/information do we want about each game?
- What information about players and games should we record outside of our functions?
- What functions should we define to operate on a player structure?

What Do We Need to Know about a Player?

What aspects/information do we want about each player?

For example,
- name
- password
- age
- # of game played
- distribution of scores
- game in progress

These will become fields in our player structure.

What Do We Need to Know about a Game?

What aspects/information do we want about each game?

For example
- current board
- current score
- current piece type, position, orientation
- next piece type (for hints)
- player

These will become fields in our game structure.

What Information Persists Outside of Functions?

What information about players and games should we record outside of our functions?

For example,
- # of players
- array of player structures
- # of players online now
- # of games in progress
- array of game structures

These will become file-scope variables.

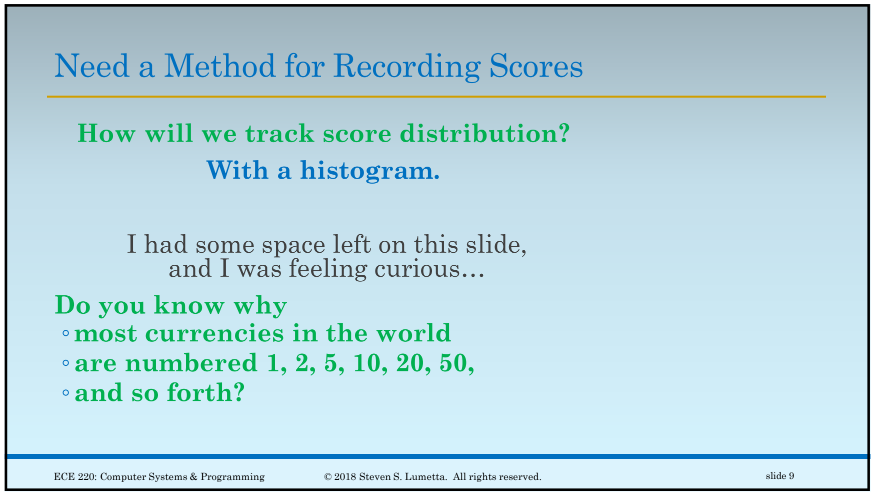
What Operations Do We Need for a Player?

What functions should we define to operate on a player structure?

For example,
- `player_init`
- `player_new_game`
- `player_finish_game`
- `player_delete`

These will be written into a single file, perhaps `player.c`.

We could ask the same question about games, but let's start writing code instead.



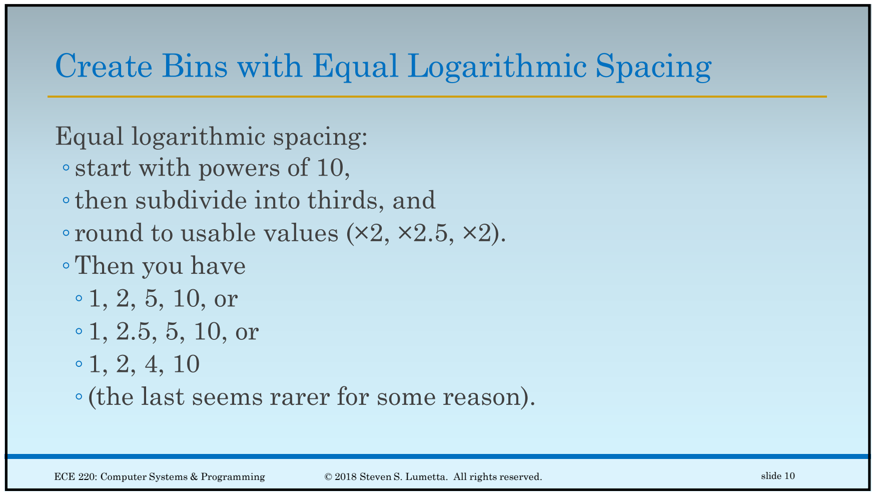
Need a Method for Recording Scores

How will we track score distribution?

With a histogram.

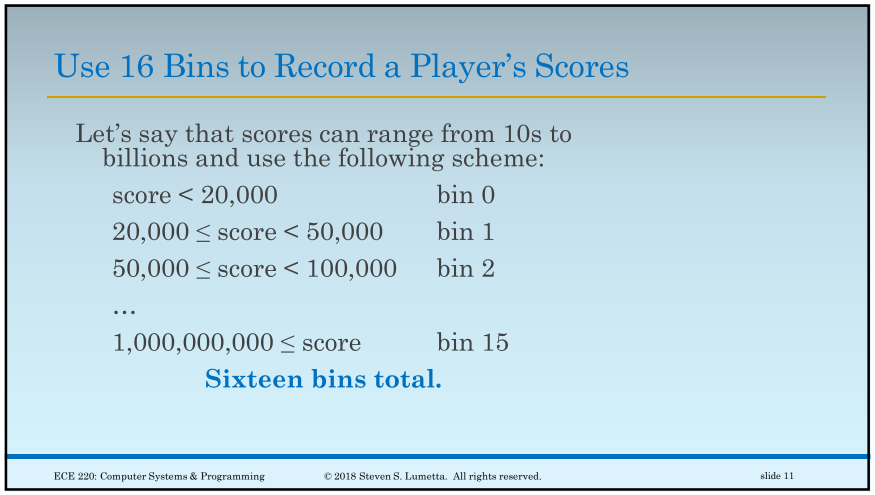
I had some space left on this slide,
and I was feeling curious…

Do you know why
- **most currencies in the world**
- **are numbered 1, 2, 5, 10, 20, 50,**
- **and so forth?**



Create Bins with Equal Logarithmic Spacing

Equal logarithmic spacing:
- start with powers of 10,
- then subdivide into thirds, and
- round to usable values (×2, ×2.5, ×2).
- Then you have
  - 1, 2, 5, 10, or
  - 1, 2.5, 5, 10, or
  - 1, 2, 4, 10
  - (the last seems rarer for some reason).

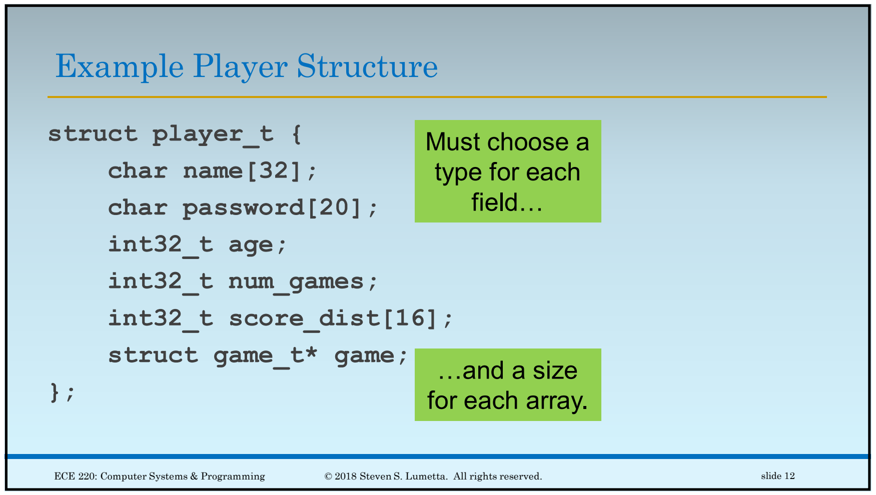


Use 16 Bins to Record a Player's Scores

Let's say that scores can range from 10s to
billions and use the following scheme:

score < 20,000	bin 0
20,000 ≤ score < 50,000	bin 1
50,000 ≤ score < 100,000	bin 2
…	
1,000,000,000 ≤ score	bin 15

Sixteen bins total.



Example Player Structure

```
struct player_t {
    char name[32];
    char password[20];
    int32_t age;
    int32_t num_games;
    int32_t score_dist[16];
    struct game_t* game;
};
```

Must choose a type for each field…

…and a size for each array.

Example Game Structure

```
struct game_t {
    space_type_t board
        [BOARD_HEIGHT][BOARD_WIDTH];
    piece_type_t cur_piece;
    int32_t cur_x;
    int32_t cur_y;
    int32_t cur_orient;
    piece_type_t next_piece;
    struct player_t* player;
};
```

Can Two Structures Have Pointers to One Another?

But can

a `struct player_t` include
a `struct game_t*` field

and

a `struct game_t` include
a `struct player_t*` field

at the same time?

Yes, both are pointers,
and both sizes are known!

Example of File-Scope Variables

```
// in player.c
static int32_t n_players = 0;
static struct player_t players[100];
static int32_t n_players_online = 0;
// in game.c
static int32_t n_games = 0;
static struct game_t games[100];
```

First Function: `player_init` to Initialize a Player

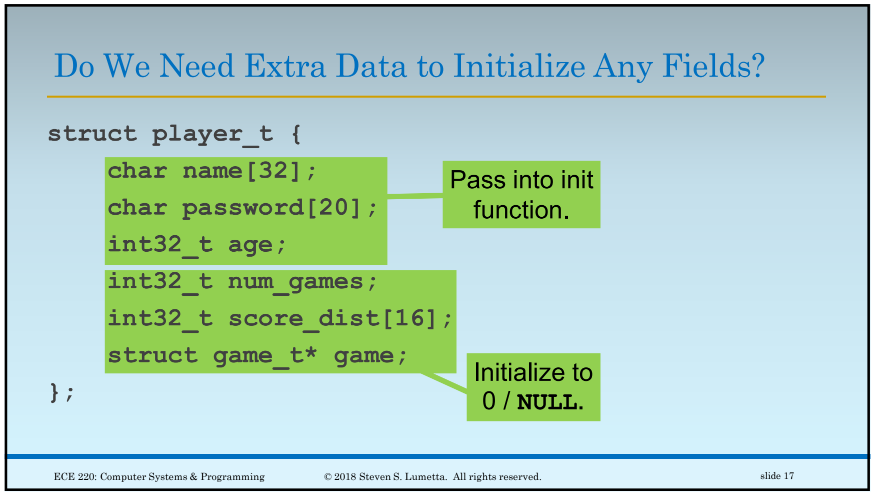
Let's start with
- a function to initialize a player.
- Call it **`player_init`**.

One parameter is a **`struct player_t*`**.

The return value?

Let's say an **`int32_t`**:
0 for failure, 1 for success.

What information do we need for initialization?

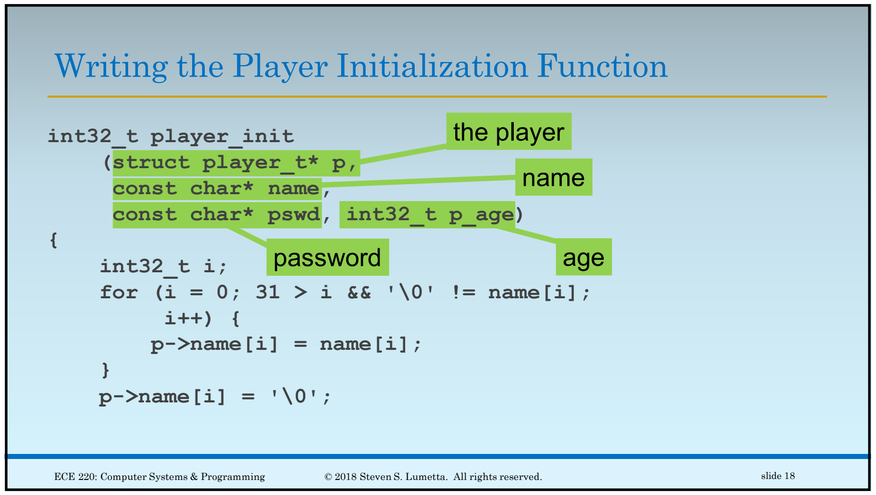


Do We Need Extra Data to Initialize Any Fields?

```
struct player_t {
    char name[32];
    char password[20];
    int32_t age;
    int32_t num_games;
    int32_t score_dist[16];
    struct game_t* game;
};
```

Pass into init function.

Initialize to 0 / **NULL**.



Writing the Player Initialization Function

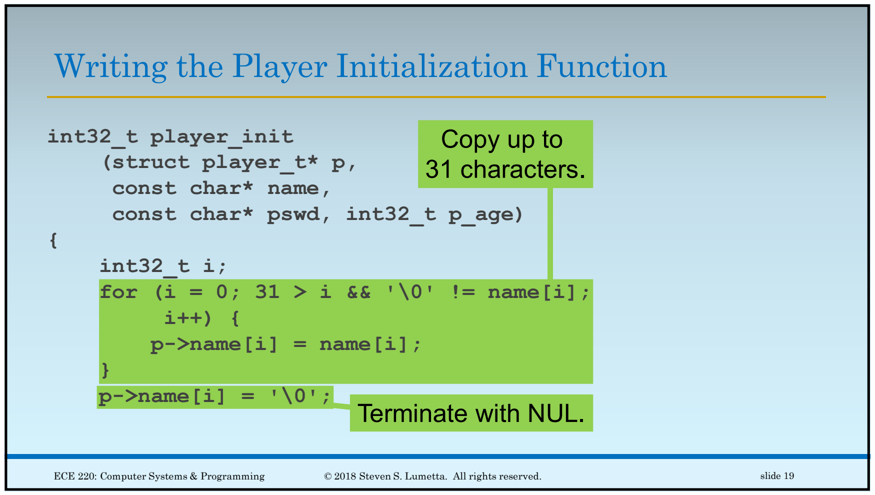
```
int32_t player_init
    (struct player_t* p,
     const char* name,
     const char* pswd, int32_t p_age)
{
    int32_t i;
    for (i = 0; 31 > i && '\0' != name[i];
         i++) {
        p->name[i] = name[i];
    }
    p->name[i] = '\0';
```

the player

name

password

age

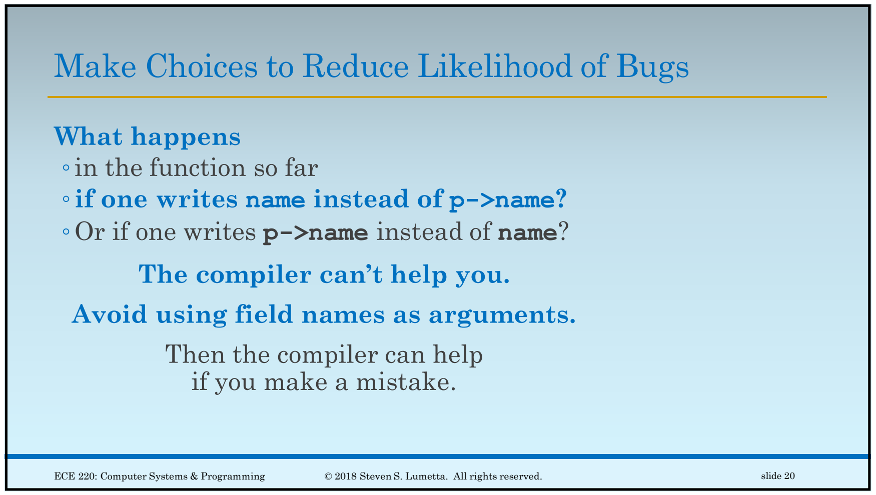


Writing the Player Initialization Function

```
int32_t player_init
    (struct player_t* p,
     const char* name,
     const char* pswd, int32_t p_age)
{
    int32_t i;
    for (i = 0; 31 > i && '\0' != name[i];
         i++) {
        p->name[i] = name[i];
    }
    p->name[i] = '\0';
```

Copy up to 31 characters.

Terminate with NUL.



Make Choices to Reduce Likelihood of Bugs

What happens
- in the function so far
- **if one writes name instead of p->name?**
- Or if one writes **`p->name`** instead of **`name`**?

The compiler can't help you.

Avoid using field names as arguments.

Then the compiler can help
if you make a mistake.

Modified Player Initialization Function (Changes in Blue)

```
int32_t player_init
    (struct player_t* p,
     const char* n,
     const char* pswd, int32_t p_age)
{
    int32_t i;
    for (i = 0; 31 > i && '\0' != n[i];
         i++) {
        p->name[i] = n[i];
    }
    p->name[i] = '\0';
```

Finish Initializing Fields Based on Parameters

```
for (i = 0;
     19 > i && '\0' != pswd[i];
     i++) {
    p->password[i] = pswd[i];
}
p->name[i] = '\0';
p->age = p_age;
```

Copy up to 19 characters.

Terminate with NUL.

Copy age into player struct.

Initialize Remaining Fields with Constant Values

```
    p->num_games = 0;
    for (i = 0; 16 > i; i++) {
        p->score_dist[i] = 0;
    }
    p->game = NULL;
    return 1;
}
```

no games played yet

no scores yet

no game being played

Return success.

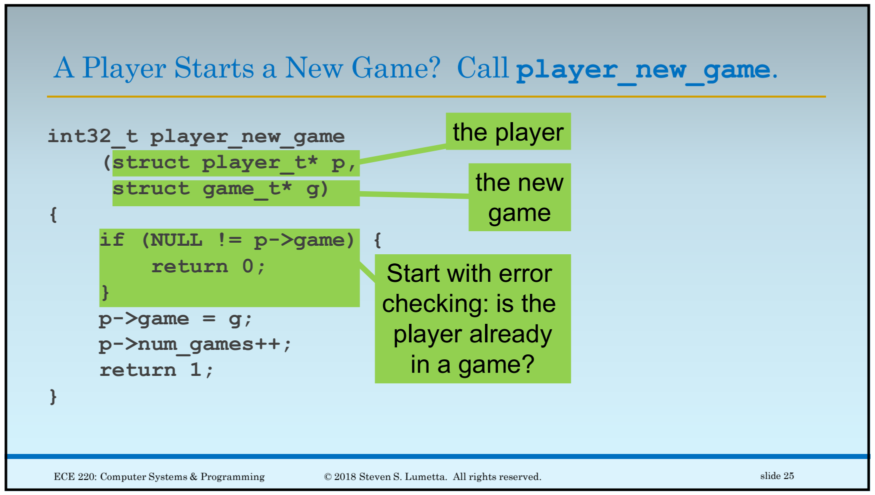
Let's Write Two More Functions for Players

Let's also write

- **`player_new_game`**, for when a player starts a game, **and**
- **`player_finish_game`**, for when a player finishes a game.

Both will **take a `struct player_t*`** as one **parameter**.

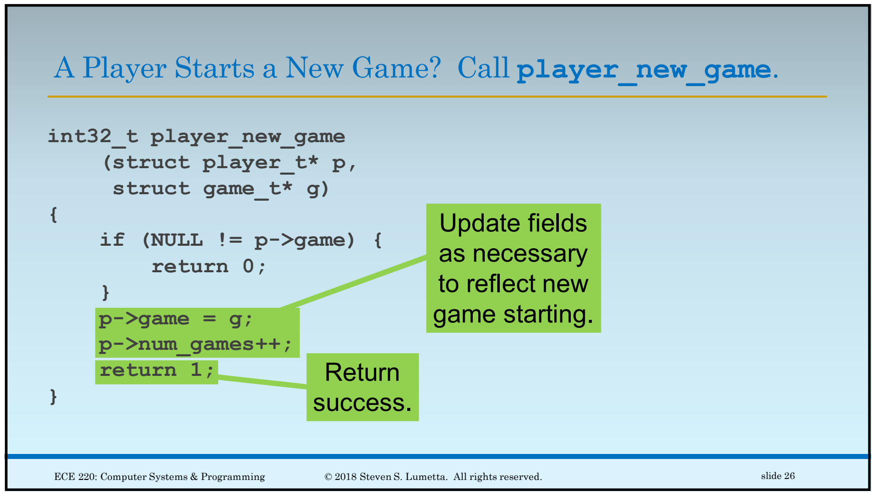
Both will **return an `int32_t`**: 0 for failure, 1 for success.



A Player Starts a New Game? Call **`player_new_game`**.

```
int32_t player_new_game
    (struct player_t* p,
     struct game_t* g)
{
    if (NULL != p->game) {
        return 0;
    }
    p->game = g;
    p->num_games++;
    return 1;
}
```

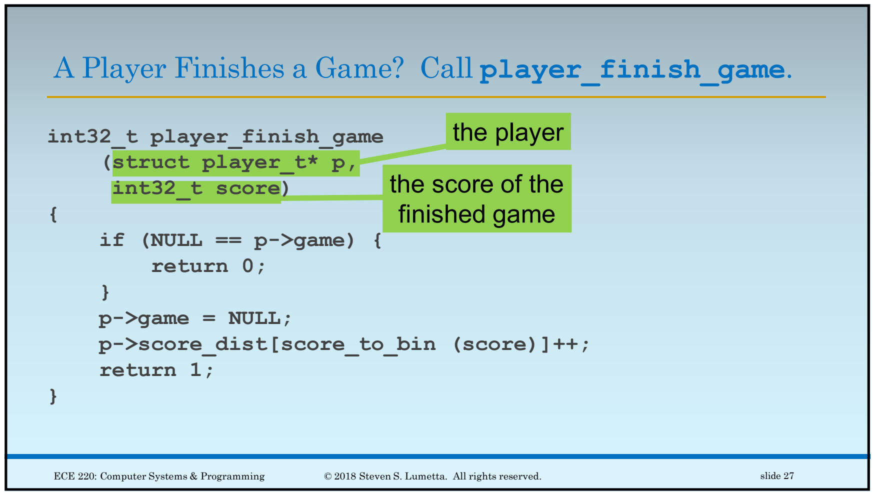
the player
the new game
Start with error checking: is the player already in a game?



A Player Starts a New Game? Call **`player_new_game`**.

```
int32_t player_new_game
    (struct player_t* p,
     struct game_t* g)
{
    if (NULL != p->game) {
        return 0;
    }
    p->game = g;
    p->num_games++;
    return 1;
}
```

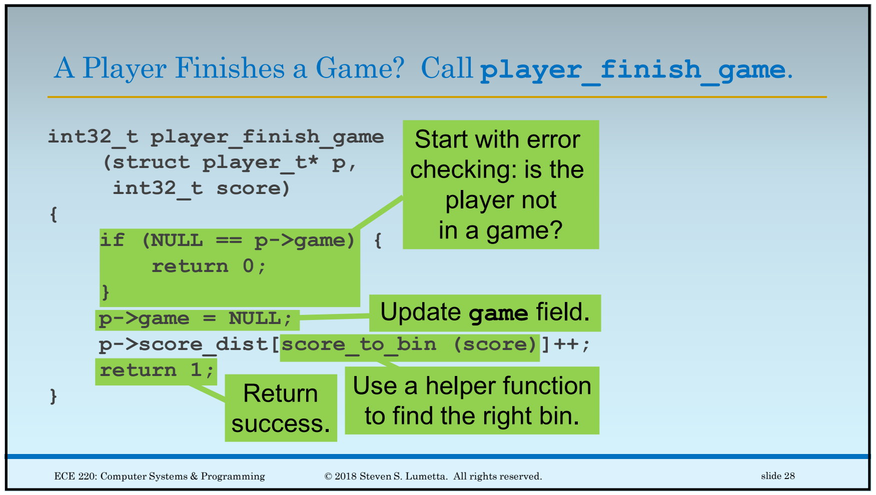
Update fields as necessary to reflect new game starting.
Return success.



A Player Finishes a Game? Call **`player_finish_game`**.

```
int32_t player_finish_game
    (struct player_t* p,
     int32_t score)
{
    if (NULL == p->game) {
        return 0;
    }
    p->game = NULL;
    p->score_dist[score_to_bin (score)]++;
    return 1;
}
```

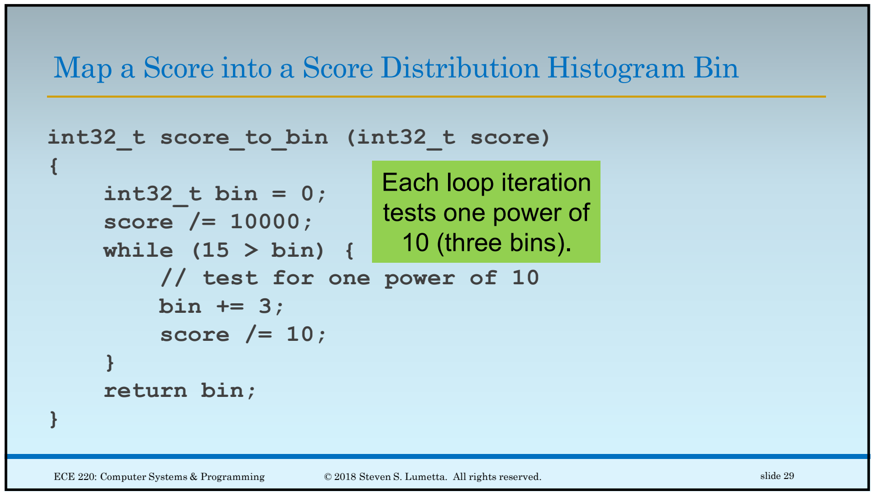
the player
the score of the finished game



A Player Finishes a Game? Call **`player_finish_game`**.

```
int32_t player_finish_game
    (struct player_t* p,
     int32_t score)
{
    if (NULL == p->game) {
        return 0;
    }
    p->game = NULL;
    p->score_dist[score_to_bin (score)]++;
    return 1;
}
```

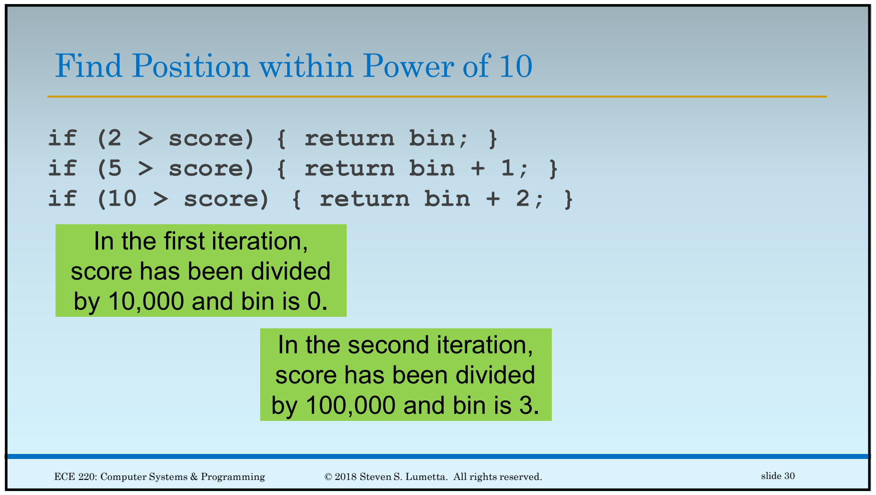
Start with error checking: is the player not in a game?
Update game field.
Use a helper function to find the right bin.
Return success.



Map a Score into a Score Distribution Histogram Bin

```
int32_t score_to_bin (int32_t score)
{
    int32_t bin = 0;
    score /= 10000;
    while (15 > bin) {
        // test for one power of 10
        bin += 3;
        score /= 10;
    }
    return bin;
}
```

Each loop iteration tests one power of 10 (three bins).



Find Position within Power of 10

```
if (2 > score) { return bin; }
if (5 > score) { return bin + 1; }
if (10 > score) { return bin + 2; }
```

In the first iteration, score has been divided by 10,000 and bin is 0.

In the second iteration, score has been divided by 100,000 and bin is 3.