Singly-Linked List Deletion is Linear in Size of List

Deletion is slower:
- **to delete player `p`**
- from a list that starts at **`player_list`,**
- we must **walk over the list** to find **`p`**,
- **then change pointer** to **`p`** to **`p->next`**.

In general,
- **with N things** in the list,
- we **examine on average N/2**.

Modify Player Structure to Use Dynamic Allocation

Before writing **`player_delete`**,
- let's **modify our player structure**
- to **use dynamic allocation**
- **for the name*** field.

*We treated the password field as a normal string before, but technically it should be hashed or encrypted to a fixed-length string.

Review: Example Player Structure

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

`name` points to a dynamically allocated block of memory.

`next` is used for the linked list.

Modify **`player_init`** to Dynamically Allocate the Name

Then, in **`player_init`**, we can write…

```
p->name = malloc (strlen (n) + 1);
if (NULL == p->name) { return 0; }
strcpy (p->name, n);
```

or

```
p->name = strdup (n);
if (NULL == p->name) { return 0; }
```

(recall that **`n`** is the new player's name).

