CSC209H Worksheet: structs

1. Here is the beginning of a program involving structs. You will need to fill in missing bits. If you can work with a partner with a machine and actually compile your program at each step, do that. If not, work on paper.

```
#define MAX_POSITION_SIZE 16
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct player {
    char *name;
    char position[MAX_POSITION_SIZE];
    int home_runs;
    float avg;
};

int main() {
    // Declare a struct player called p1.
```

struct player p1;

```
    // Initialize it to represent the third baseman Josh Donaldson,
    // whose batting average is 0.270. He has hit 33 home runs.
```

p1.name = "Josh Donaldson";
strcpy(p1.position, "third baseman");
p1.home_runs = 33;
p1.avg = 0.27;

Alternatively, we could have used malloc to allocate space on the heap.

2. Here we have added a declaration for a pointer to a struct player. Allocate space for the struct on the heap and have `p2` point to that memory.

   Error-checking: Look at the man page for `malloc` to see what it returns if it is unable to allocate memory. Now add code to check if the memory allocation for `p2` was successful, and if it failed, exit the program with a non-zero exit status.

```
    struct player *p2;
```

p2 = malloc(sizeof(struct player));
if (p2 == NULL) {
    return 1;
}

```
    // Set the values of p2 to represent shortstop Troy Tulowitzki. His batting average is .249
    // and he has hit 7 home runs.
```

// Equivalent to:
// (*p2).name = ...

p2->name = "Troy Tulowitzki";
strcpy(p2->position, "shortstop");
p2->home_runs = 7;
p2->avg = 0.249;

CSC209H Worksheet: structs

3. Write a function `out_of_the_park` that increments the home-run count for the player passed as the function's argument. Think carefully about what the type of the function parameter should be.

```
void out_of_the_park(struct player *p){
    p->home_runs = p->home_runs + 1;
}
```

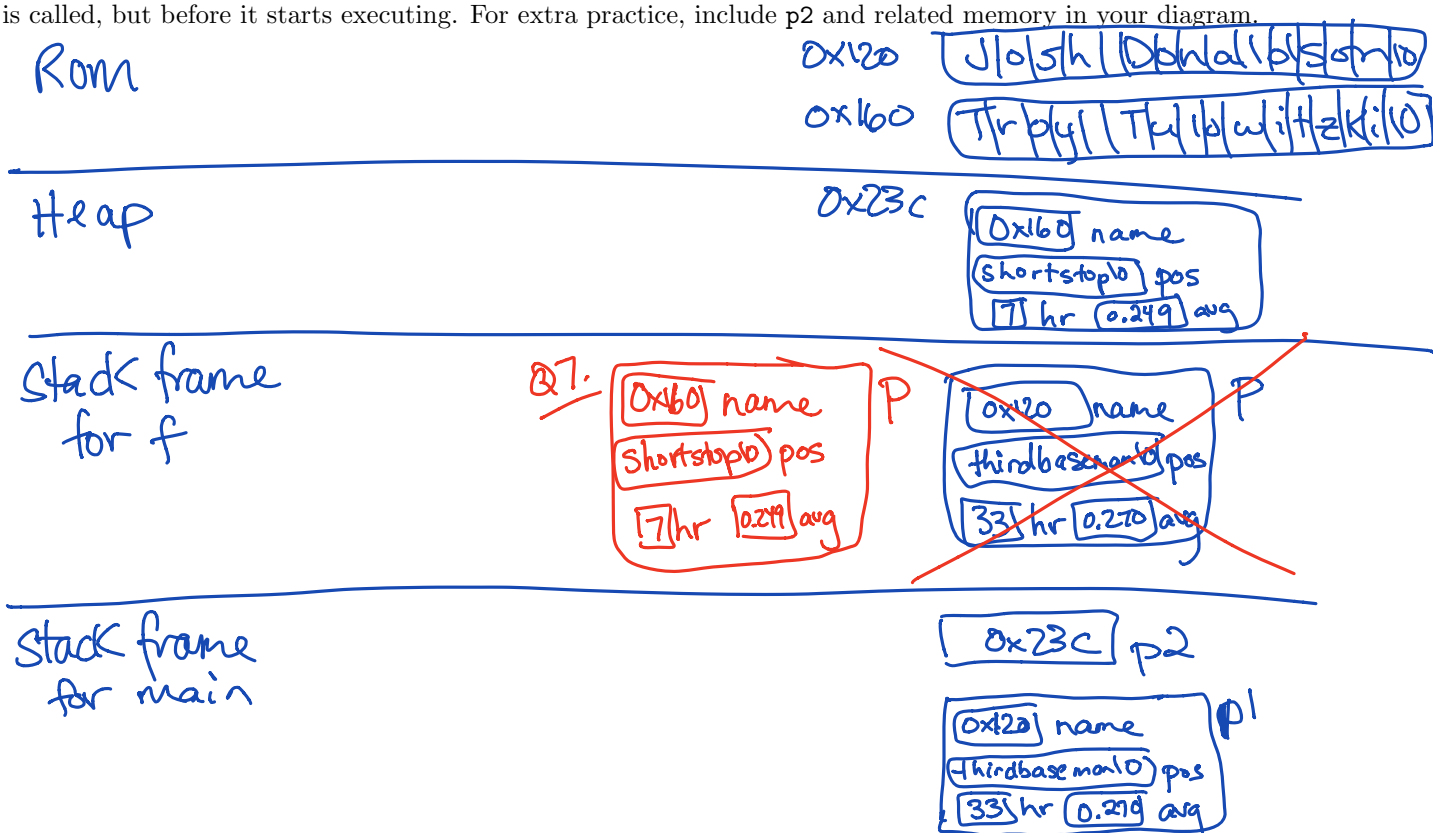
4. Show how to make calls to `out_of_the_park` using `p1` and `p2`.

```
out_of_the_park(&p1);
out_of_the_park(p2);
```

5. Suppose we have the following function declaration.

```
void f(struct player p) { // Body hidden }
```

Now suppose we call it from `main` using `f(p1)`. Draw the memory diagram of the program immediately after `f` is called, but before it starts executing. For extra practice, include `p2` and related memory in your diagram.



6. Something to think carefully about: can the body of `f` affect the local `p1` of `main`? In other words, after `f(p1)` exits, can any data associated with `p1` have changed?

No, not with the way we allocated the names in read-only memory and the rest of the data on the stack. However, if we had malloced space for the names on the heap, then changing p.name could change p1.name.

7. On a new sheet of paper, repeat the previous two questions when you call `f(*p2)` instead.

Only the stack frame for f changes. In this case, p is a copy of the struct from the heap. Note that *p is evaluated to get the initial value of p (a copy of 'struct player' at 0x23c),