Introduction to Structures

Structures in C are a convenient way to group a set of variables that all work together to describe a certain element. C is not an object oriented programming language, like C++ and Java are, so we can use structures as a very rough approximation of objects. In HW 9, we have the Player structure, this is a collection of variables describing the player, such as name, position, number, and skill. The syntax to define a structure is:

```
struct struct_name {
      struct_member_1;
      ...
      struct_member_n;
}; // don't forget the semi-colon at the end

// now we can use struct struct_name like any other type
struct struct_name *struct;
```

We can also use typedef on the struct so that we do not have to type the struct keyword each time we want to use it.

We can either do it after the struct has been defined:

```
typedef struct struct_name struct_alias; // The struct_alias is
normally just struct_name, but can be anything
```

Or when we are defining it:

```
typedef struct struct_name {
      struct_member_1;
      ...
      struct_member_n;
} struct_alias;
```

In order to access a variable inside of a structure we use one of two operators, '.' if the variable is just of type struct, or '->' if the variable is a struct pointer. The arrow notation is a shorthand way of writing "(*struct_name)." and both work the same, though the arrow is normally preferred.

Eg. `pointerToStruct->member` is the same as `(*pointerToStruct).member`

While you can have structs inside of structs, you cannot have a recursive definition, i.e. a struct can contain a pointer to a struct of the same type, but cannot have a member of the same type that is a normal variable.

```
typedef struct struct_name {
      struct_name *struct1; //this is fine
      struct_name struct; //this is not
} struct_alias;


typedef struct point {
      int x;
      int y;
} point;
```

You can initialize structs either by creating a variable of that type and setting each of its members individually:

```
Point p;

p.x = 3
```

Or set them when making the variable

```
point p = {1,3};
```

We don't have to set all of the members, but they will be set it order

```
point p = {1}; //make a point with x = 1, y uninitialized to 0;
```

We can also just make a point with a y value if we want

```
point p = {.y = 1}; //make a point with y = 1, x uninitialized to 0;
```

Example:

```
#include <stdio.h>

typedef struct point {
      int    x;
      int    y;
} point;

int main(int argc, const char *argv[]){
      point p = {3};
      printf("p.x = %d,p.y = %d\n", p.x,p.y);
      point q = {.y = 3};
      printf("q.x = %d, q.y = %d\n", q.x,q.y);
      return 0;
}
// outputs:
// p.x = 3,p.y = 0
// q.x = 0, q.y = 3
```