C Structures

- **Functions**: allow us to organize the structure of the code
- **Structures**: allow us to organize variables in a more logical way

Structures in C are named collections of one or more related variables, possibly of different types
Java vs C Structures: Example

Java Example:
```java
class Slot {
    int x;
    int y;
    int direction;
    methods ...
}
```

In C:
```c
struct Slot {
    int x;
    int y;
    int direction;
};
```

Slot is the name (tag) of the structure
x, y, direction are members of the structure

Of course, no inheritance, associated (private) methods
Structures and types

• Tag name used after struct introduces a new datatype
• `sizeof` operator works on struct
• Continuing the example from previous slide ...
  ```c
  struct Slot s1, s2;
  ```

```c
struct tag {
  list of variables
}
```
Accessing members of a structure

Consider declarations

```c
struct Slot s1, s2;
int i;
```

Allowed

```c
i = s1.x;
```
Structures and pointers

- We can define pointers to structures
  ```c
  struct Slot * s1_ptr = NULL;
  struct Slot s2, s1;
  ```
- Operate with them
  ```c
  s1_ptr = &s2;
  s1 = s2;
  ```
Struct and sizeof

• If the structure contains dynamically allocated members, the size of whole struct may not equal sum of its parts

```c
struct word {
    char * c;
    int     length;
}
```

• Sizeof(struct word) will return …8 bytes. But if char points to some arbitrary string, then the total memory associated with the struct is obviously bigger.

• Internal padding
Padding

1. struct Example
2. {
3.    int a;
4.    char b;
5.    int c;
6.};
7.

What is sizeof(Example)?

What is sizeof(a)+sizeof(b)+sizeof(c)?
Structures and ... structures

- A structure can contain a member of another structure

```c
struct Position{
    int x;
    int y
};

struct Slot {
    struct Position pos;
    int direction;
};

Access x via : slot.pos.x
```
A structure can not refer itself (contain a member of the same structure) UNLESS it is a pointer – such structures are called self-referential (or recursive) structures.

```c
struct tnode {
    char * word;
    int count;
    struct tnode *left;
    struct tnode *right;
}
```
Recursive structures

```c
struct regression
{
    int int_member;
    struct regression self_member;
};
```

What’s wrong with this definition?
Structures and functions

- Structures can be initialized, copied, taking its address and accessing its members;
- They can not be compared
  - Eq vs equal: Do two structures represent the same object vs. Do two structures have the same value
- Functions can return structure instances
  - What is the cost in terms of memory allocation, copying, and performance?
  - What’s the difference between arrays and structures in this sense?
Structures and functions

struct point {
    int x;
    int y
}

struct point createpoint(int x, int y) {
    struct point temp;

    temp.x = x;
    temp.y = y;
    return temp;
}

struct point p1 = createpoint(0, 0);
Typedef

- Allows us to create new data name types;

```c
typedef int Length;
Length l1, l2;
```
typedef and structures

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

Notice the difference. NO struct needed when using the type.
Position p1, p1;
Readings and exercises for this lecture

K&R Chapter 6 till 6.7