CS240: Programming in C

Lecture 9: Structures

C Structures

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

Structures in C are collections of one or more related variables, possibly of different types, for convenient handling

Java vs C Structures: Example

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

```
In 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

Structures and types

- Tag name used after struct introduces a **new datatype**
- **`sizeof`** operator works on struct
- Continuing the example from previous slide …

```
struct Slot s1, s2;
```

```
struct tag {
        list of variables
}
```

Accessing members of a structure

Consider declarations

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

Allowed

```
i = s1.x;
```

Structures and pointers

- We can define pointers to structures

```
struct Slot * s1_ptr = NULL;
struct Slot s2, s1;
```

- Operate with them

```
s1_ptr = (struct Slot *)
malloc(sizeof(struct Slot))


 free(s1_ptr);
 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

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

- Sizeof(struct word) will return …8 bytes. But if char points to some string that was dynamically allocated, the memory occupied by the struct word will be bigger.

Memory layout for a structure

- Data alignment: when cpu accesses the memory reads more than one byte, usually 4 bytes on a 32-bit platform.
- What if the data structure is not a multiple of 4? Padding.
- Many computer languages and computer language implementations handle data alignment automatically.

Structures and … structures

- A structure can contain a member of another structure

```
struct Position{
  int x;
  int y
 }
 struct Slot {
  struct Position pos;
  int direction;
 }
```

Structures and … structures

- A structure can not refer itself (contain a member of the same structure) UNLESS it is a pointer – such structures are called self-referential structures.

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

Structures and functions

- A structure can be initialized, copied, taking its address and accessing its members;
- Structures can not be compared
- Functions can return struct

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;

```
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;
```

Structures summary

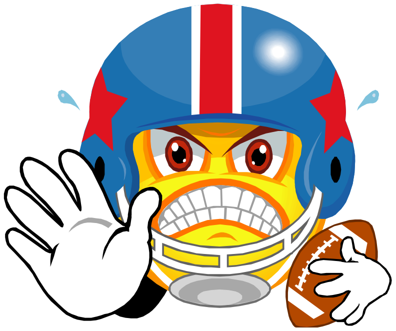
- Holds multiple items as a unit
- Can be returned from functions
- Can be passed to functions
- They can not be compared
- A structure can include
  - a pointer to itself, but not a member of the same structure
  - a member of another structure, the latter has to have the prototype declared before

Structures summary

- Member access
  - Direct: **`s.member`**
  - Indirect: **`s_ptr->member`**
  - Dot operator . has precedence over indirection -> : agenda.contact->name
- Use const to make a structure read-only

Practice

Write a linked list using dynamic memory allocation and structures.



Readings this lecture

K&R Chapter 6 till 6.7