STRUCTS

LAB 10

FALL 2023

HOW TO DECLARE STRUCTS

```
typedef struct _person {
    char * name;
    int age;
    char gender;
} Person;
// Usage:
//    Person p
//    Person * p;
```

```
struct _person {
    char * name;
    int age;
    char gender;
};
// Usage:
//    struct _person p;
//    struct _person * p;
```

STRUCTS (NON-POINTER DEFINITION)

```
typedef struct _person {
    char * name;
    int age;
    char hair_color;
} Person;
```

- Creating an instance.

```
Person p;
p.age = 100;
p.hair_color = ‘c’;
strcpy(p.name, “My Name”);
```

STRUCTS (POINTER DEFINITION)

```
typedef struct _person {
    char * name;
    int age;
    char hair_color;
} Person;
```

- Suppose we use malloc to create p.

```
Person * p =
malloc(sizeof(Person));
// verify p is given memory.
p->age = 100;
p->hair_color = ‘c’;
strcpy(p->name, “My Name”);
```

INITIALIZE A STRUCT

```
MyStruct myStructInstance = {
        .field1 = 10,
        .field2 = “Something”,
        ...
};
```

Note: The order of which you give values SHOULD follow how they are declared, but it will work in any order.

STRUCTS (OPERATORS)

- `. (Access)`
  - Access a member of a struct
  - `Struct.myInt = 100;`
- `-> (Access)`
  - Access a member using a pointer to a struct
  - `myPtrToMyStruct->myInt = 100;`
- LHS = RHS (Copy)
  - Copy struct on RHS to LHS
  - This happens internally, this is simply done by copying field-by-field from RHS to LHS.
  - `MyStruct newStruct = copyThisStructOver;`

WHAT TYPES CAN STRUCTS CONTAIN?

- Any type. In an indirect manner, we can have “functions” as a member.
- Stick to types we have used so far.