Structure and ~~Union~~ Types

Structure Basics

- Containers for data
- Defined data type
- Contiguous memory for data
- Declare a struct for a person:

```
typedef struct {
        int id;
        char name[NAME_SIZE];
        char addr[ADDR_SIZE];
        char city[CITY_SIZE];
        char state[STATE_SIZE];
        char zip[ZIP_SIZE];
} person_t;
```

Structure Basics

- Declaring a struct
  - person_t this_person;
- Declaring and initializing a struct
  - person_t that_person = {0, “”, “”, “”, “”, “”};

Structure Basics

- Accessing data members (storing)
  - this_person.id = 123;
  - strcpy(this_person.name, “Simpson, Homer”);
  - strcpy(this_person.addr, “742 Evergreen Terrace”);
  - strcpy(this_person.city, “Springfield”);
  - strcpy(this_person.state, “WA”);
  - strcpy(this_person.zip, “12345”);
- Accessing data members (retrieving)
  - printf(“Name: %s\n”, this_person.name);
- Assignment (deep copying)
  - this_person = that_person;
- Reference
  - person_t *ptr = &this_person;

Structures in Functions

- Pass a struct
  - print(this_person);
  - Passed by value
  - equals()
- Pass by reference
  - get_person(&this_person)
  - In function must maintain op precedence strcpy((*this_person).name, “Homer”);
  - Can also use this_person->name
  - Use of scanf(“%d”, &(*this_person).id);
  - Can also use &this_person->id

Structures in Functions

- Returning a struct
  - person_t get_person();
  - this_person = get_person();

Structures in Arrays

- Declare array of structs
  - person_t person_list[SIZE];
  - Similar to primitive arrays but each element can contain multiple types
  - length
  - count
- Array as a struct
  - length
  - count