C Structs

CST 357/457 – Systems Programming
Michael Ruth, Ph.D.
Associate Professor
Computer Science & I.T.
mruth@roosevelt.edu

Objectives

- Discuss structures in C and their use
- Explain struct definitions, declarations, initializations, and their use in functions
- Discuss the use of arrays of structs
- Explain pointer use with structs including shorthand, declarations, and referencing

Introduction to Structures

- Structure is a collection of one or more variables, possibly of different types which are then grouped under a single name for convenient handling
  - Also called “Records” in other languages
  - Used with pointers, we can create more interested/complicated data structures
    - Linked list
    - Trees
    - Graphs

Definition

- Structures are created using the struct keyword:

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

- The keywords introduces a structure declaration, which is just a list of declarations enclosed in braces
- The variables named in a structure are called its members
- The name for this struct is point

Struct and Type

- A struct declaration defines a "type"
- The right brace may be followed by a list of variables, just like any other type
  - **`struct { … } a,b,c;`**
- A structure definition not followed by lists of variables reserves no storage
  - It simply defines the shape of the type
- However, if the struct is named, the name may be used to declare a variable
  - **`struct point pt;`**

typedef

- **`typedef`** keyword creates synonyms (aliases) for previously defined types
  - Use it to create shorter type names
- Can and will improve readability in almost all cases
  - Unless overused
- Example:
  - **`typedef struct point pt;`**
    - Defines pt to be a new type name of type "struct point"

Struct Initialization

- Structures can be initialized by following its definition with a list of initializers
  - With constant expressions
  - Ex:
    - `struct point maxpt = { 320, 200 }`
- Can also initialize the structure using its member variables
  - Ex:
    - `struct point maxpt;`
    - `maxpt.x = 320`
    - `maxpt.y = 200`

Nested Structures

- Structures can Also be nested:
  - Ex

```
struct rect {
   struct point lowerLeftpt;
   struct point upperRightpt;
}
```


Valid Structure Operations

- There are only 4 valid operations
  - Assigning a structure to a structure of the same type
  - Taking the address (&) of a structure
  - Accessing the members of a structure
  - Using the sizeof operator to determine the size of a structure

Structures and Functions

- There are Only 3 options
  - Pass components
    - Exactly what we've seen before
  - Pass the entire structure
    - `struct point midPt(struct point x, struct point y) { … }`
  - Pass a pointer to the structure
    - Very, similar to above except as pointers
      - Which we'll discuss next!
    - If the structure is large, it is generally more efficient to pass a pointer than to copy the whole structure

Pointers and Structs

- `struct point *origin;`
  - Origin is a pointer to a structure of type point
- Often see this used with **`typedef`**
  - Alias pt for "struct point"
    - `typedef struct point pt;`
  - Alias for a pointer to a "struct point"
    - `typedef pt *ptPtr;`
- Pointers and structs are used so often together, there is a "shorthand" notation for the member variables
  - If p is a pointer to a structure
    - P→<member-of-structure>
    - EX:
      - `ptPtr→x`

Arrays and Structs

- We can also have arrays of structures:

```
#define NPTS 30
struct point {
   int x;
   int y;
}

struct point points[NPTS];
```

- Suppose we wished to determine the size of the array (which was init'd somewhere else)
  - `sizeof(points) / sizeof(struct key)`
  - `sizeof(points)/sizeof(points[0])`
- Never EVER assume that the size of a structure is the sum of its parts!!!

Quick Example

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

typedef struct point pt;
typedef pt * ptPtr;

pt getMidPoint(ptPtr x, ptPtr y) {

   pt result;
   result.x = (.5 * ((x->x) + (y->x)));
   result.y = (.5 * ((x->y) + (y->y)));
   return result;

}
```


Summary

- Discussed structures in C and their use
- Explained struct definitions, declarations, initializations, and their use in functions
- Discussed the use of arrays of structs
- Explained pointer use with structs including shorthand, declarations, and referencing

Questions?