Structures in C

(Chapter 19)

1

Structures

•Programs are solving a ‘real world’ problem

- Entities in the real world are real ‘objects’ that need to be represented using some data structure
  - With specific attributes
- Objects may be a collection of basic data types
  - In C we call this a structure

2

2

Data Structures

•A data structure is a particular organization of data in memory.

- We want to group related items together.
- We want to organize these data bundles in a way that is convenient to program and efficient to execute.

•An array is one kind of data structure.

- `struct` – directly supported by C
- linked list – built from `struct` and dynamic allocation

3

3

Variables and Structures

•C gives you freedom to create your own data-types

- We call it a structure in C
- Example:
  - Remember complex #? $Y = real + imaginary\#$
  - C doesn't have a complex # type, but we can create one:

We call these "fields" of the structure ←

```
typedef struct {
   double real, imag;
} complex;
```

- This "structure" acts as a new data-type for variables:
  - Now we can declare a variable like this:

```
complex my_complex_var ;
```

4

4

Structures in C – Ex.2

•A **`struct`** is a mechanism for grouping together related data items of different types.

- Recall that an array groups items of a single type.

•Example:

We want to represent an airborne aircraft:

```
char flightNum[7];
int altitude;
int longitude;
int latitude;
int heading;
double airSpeed;
```

•We can use a **`struct`** to group these data together for each plane.

5

5

Defining a Struct

•We first need to define a new type for the compiler and tell it what our struct looks like.

```
•struct flightType {
   char flightNum[7];   /* max 6 characters */
   int altitude;        /* in meters */
   int longitude;       /* in tenths of degrees */
   int latitude;        /* in tenths of degrees */
   int heading;         /* in tenths of degrees */
  double airSpeed;     /* in km/hr */
•};
```

•This tells the compiler how big our struct is and how the different data items ("members") are laid out in memory.

•But it does not allocate any memory.

6

6

Defining and Declaring at Once

•You can both define and declare a struct at the same time.

```
•struct flightType {
   char flightNum[7];   /* max 6 characters */
   int altitude;        /* in meters */
   int longitude;       /* in tenths of degrees */
   int latitude;        /* in tenths of degrees */
   int heading;         /* in tenths of degrees */
   double airSpeed;     /* in km/hr */
•} maverick;
```

•And you can use the flightType name to declare other structs.

```
•struct flightType iceMan;
```

7

typedef

•C provides a way to define a data type by giving a new name to a predefined type.

•Syntax:

• **typedef <type> <name>;**

•Examples:

```
•  typedef int Color;
•  typedef struct flightType Flight;
•  typedef struct ab_type {
     int a;
     double b;
   } ABGroup;
```

8

Using typedef

•This gives us a way to make code more readable by giving application-specific names to types.

- `Color pixels[500];`
- `Flight plane1, plane2;`

•Typical practice:

•Put typedef's into a header file, and use type names in main program. If the definition of Color/Flight changes, you might not need to change the code in your main program file.

- Pay attention…..need this in your Project 5.6

9

9

Example 3..Structures in C

•Example: We want to represent information pertaining to a student – that uniquely identifies a student

•No two students have the same values in all fields

```
char GWID[9];
char lname[16];
char fname[16];
float gpa;
```

•We can use a **`struct`** to group these data together for each student.

•Student record

```
•struct student {
   char GWID[9];
  char lname[16];
  char fname[16];
  float gpa
  };
```

10

10

Declaring and Using a Struct

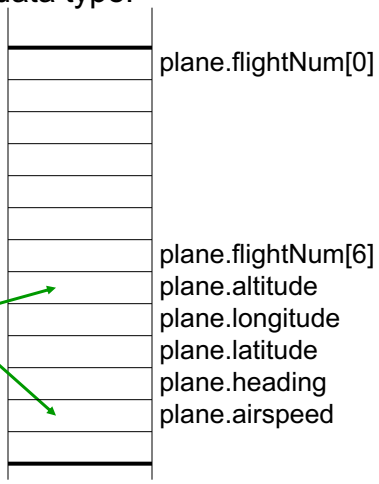
•To allocate memory for a struct,
we declare a variable using our new data type.

```
•struct flightType plane;
```

•Memory is allocated,
and we can access
individual members of this
variable:

```
•plane.airSpeed = 800.0;
plane.altitude = 10000;
```

•A struct's members are laid out
in the order specified by the definition.



11

11

Generating Code for Structs

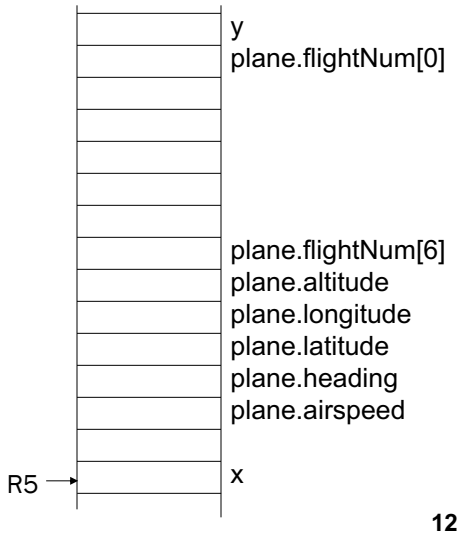
•Suppose our program starts out like this:

```
•int x;
Flight plane;
int y;

plane.altitude = 0;
•...
```

•LC-3 code for this assignment:

```
•AND  R1, R1, #0
ADD  R0, R5, #-13 ; R0=plane
STR  R1, R0, #7   ; 8th word
```



12

12

Array of Structs

•Can declare an array of structs:

```
Flight planes[100];
```

•Each array element is a struct

•To access member of a particular element:

```
planes[34].altitude = 10000;
```

•Because the `[]` and `.` operators are at the same precedence, and both associate left-to-right, this is the same as:

```
(planes[34]).altitude = 10000;
```

13

13

Pointer to Struct

•We can declare and create a pointer to a struct:

```
Flight *planePtr;
planePtr = &planes[34];
```

•To access a member of the struct addressed by Ptr:

```
(*planePtr).altitude = 10000;
```

•Because the . operator has higher precedence than *, this is NOT the same as:

```
*planePtr.altitude = 10000;
```

•C provides special syntax for accessing a struct member through a pointer:

```
planePtr->altitude = 10000;
```

14

14

Structures and Pointers

- It is very common to have a pointer to a structure.

  Example:

```
complex* ptr;// declares ptr to var of‘complex’
                        type
```

  - Works the same way as having a pointer to a variable with a built-in datatype (like int, float, double)

- How do you dereference your pointer to get to fields?

```
ptr = &my_complex_var ;     // assign pointer
(*ptr).real = 5 ;    // interrogate “real” field
(*ptr).imag = 6 ;// interrogate “imag” fields
```

- C offers special syntax to dereference & access fields at same time:

```
ptr = &my_complex_var ;     // assign pointer
ptr->real = 5 ;      // deref & interrogate field
ptr->imag = 6 ;             // interrogate fields
```

15

15

Arrays and Pointers to Struct

- We can declare an array of structs
  - **student enroll[100]**
  - **Enroll[25].GWID=‘G88881234’**
- We can declare and create a pointer to a struct:
  - **student student *stPtr;**
    stPtr = &enroll[34];
- To access a member of the struct addressed by Ptr:
  - **(*stPtr).lname = ‘smith’;**
- Or using special syntax for accessing a struct member through a pointer:
  - **stPtr-> lname = ‘smith’;**

16

16

Ex 1: Passing Structs as Arguments

•Unlike an array, a struct is always passed by value into a function.

- This means the struct members are copied to the function's activation record, and changes inside the function are not reflected in the calling routine's copy.

•Most of the time, you'll want to pass a pointer to a struct.

```
•int Collide(Flight *planeA, Flight *planeB)
{
  if (planeA->altitude == planeB->altitude) {
     ...
  }
  else
    return 0;
}
```

17

17

Ex 2: Passing Structs as Arguments

•Unlike an array, a struct is always passed by value into a function.

- This means the struct members are copied to the function's activation record, and changes inside the function are not reflected in the calling routine's copy.

•Most of the time, you'll want to pass a pointer to a struct.

```
int similar(student *studentA, student *studentB)
{
  if (studentA->lname == studentB->lname) {
     ...
  }
  else
    return 0;
}
```

18

18