3. Structures

Oliver Diessel
odiessel@cse.unsw.edu.au

Last updated: 10:10 11 Aug 2008
Structures

Topics to be covered:
- Using the keyword `typedef`
- Primitive vs compound data types in C
- Structures
- Pointers to structures
- How to pass structures as arguments to functions
- How to return structures from functions
- Combining structures

Slide credits: Vladimir Pervouchine
typedef – when is it useful?

- Meaningful names:

  ```c
  typedef int phone_number;
  typedef float currency;
  
  phone_number my_number = 57384;
  ```

- Allow for easy changes:

  ```c
  typedef float number;
  
  number complex_calculation(number a, number b) {
      number c = log(a+b);
      ...
      return c;
  }
  ```
**typedef** – when is it useful?

- **Meaningful names:**
  
  ```c
  typedef int phone_number;
  typedef float currency;
  ```
  
  ```c
  phone_number my_number = 57384;
  ```

- **Allow for easy changes:**

  ```c
  typedef double number;
  ```
  
  ```c
  number complex_calculation(number a, number b) {
    number c = log(a+b);
    ...
    return c;
  }
  ```

  **No change to function required!**
typedef as a shorthand for compound types

- Primitive types:
  int, double, float, char, ...

- Compound types:
  arrays and strings
  
  ```c
  char S[MAX_LENGTH];
  float grades[NUM_ELEMENTS];
  ```

  ```c
  typedef char surname_t[MAX_LENGTH];
  typedef float float_array[NUM_ELEMENTS];
  ```

  ```c
  surname_t S;
  float_array grades;
  ```

  ```c
  surname_t class_list[NUM_STUDENTS];
  ```
New compound type: structure

typedef struct {
    surname_t surname;
    surname_t given_name;
    int year_of_birth;
    int month_of_birth, day_of_birth;
    int height;
    int ID;
} person_record;

person_record player;
person_record team[MAX_TEAM_SIZE];
typedef char surname_t[MAX_LENGTH];
typedef struct {
    surname_t surname;
    surname_t given_name;
    int year_of_birth;
    int month_of_birth, day_of_birth;
    int height;
    int ID;
} person_record;

person_record player;
person_record team[MAX_TEAM_SIZE];

player.year_of_birth = 1987;
scanf("%s", team[2].surname);

player.height++; // increment height
Assignment and comparison of structures

- Assignment

  \[
  \text{team}[3] = \text{player}; \quad \text{// valid}
  \]

- Comparison

  \[
  \text{if (team}[2] == \text{team}[3]) \quad \text{// is this valid?}
  \]

  \[
  \text{if (team}[4] > \text{player}) \quad \text{// does this make sense?}
  \]

  Variables of type \text{struct} cannot be compared!
Passing a structure to a function

A function receives its *local copy* of the entire structure variable:

```c
void print_data(person_record person) {
    printf("Name: %s, %s\n", person.surname,
            person.given_name);

    printf("Date of birth: %2d/%2d/%4d\n",
            person.date_of_birth,
            person.month_of_birth,
            person.year_of_birth);

    person.height++; // does this affect the
    // data of the caller ?
}
```
Pointers to structures

- Pointers to primitive types:
  ```
  int* pointer_to_int;
  char* pointer_to_char;
  ```

- Pointers to arrays:
  ```
  char a_string[10];  // a_string is of type char*,
  int numbers[100];  // numbers is of type int*,
  ```

- Pointers to structures:
  ```
  person_data* pointer_to_person_data;
  ```

What’s so useful about them?…
Pointers to structures – size considerations

- Size of a structure in bytes:
  ```c
  int i = sizeof(person_data); // how many bytes?
  ```

  Normally,
  ```c
  sizeof(struct_data) >> sizeof(struct_data*)
  ```

- A function receives its *local copy* of the entire structure variable:
  ```c
  void print_data(person_record person) {...
      20+2*sizeof(surname_t) bytes copied
  }
  ```

- A function can receive only its *local copy of a pointer*:
  ```c
  void print_data(person_record* person) {...
      4 bytes copied
  }
  ```
Pointers to structures
– when we want to change data

- Main program:
  ```
  person_record player;
  // ... input the data

  change_ID(player, 8);
  ```

- Copy of the variable:
  ```
  void change_ID(person_record person, int ID) {
      person.ID = ID;  // Useless: person is a local copy
  }
  ```
Pointers to structures
– when we want to change data

- Main program:
  ```c
  person_record player;
  // … input the data
  
  change_ID(&player, 8);
  ```

- Copy of the variable:
  ```c
  void change_ID(person_record person, int ID) {
    person.ID = ID;  // Useless: person is a local copy
  }
  ```

- Copy of a pointer to it:
  ```c
  void change_ID(person_record* person_p, int ID) {
    (*person_p).ID = ID;  // person_p is a local copy of a pointer
    // It points to a player variable!
  }
  ```
Arrow notation

- Accessing data:
  ```c
  void change_ID(person_record* person_p, int ID) {
    (*person_p).ID = ID;
  }
  ```

- Shorthand:
  ```c
  void change_ID(person_record* person_p, int ID) {
    person_p->ID = ID;
  }
  ```
Returning structures from functions

- Functions can return structures. The value to be returned must be composed in a local variable.

```c
person_record player = get_person();
...

person_record get_person(void) {
    person_record new_person;
    scanf(“%s %s”, new_person.surname,
          new_person.given_name);
    scanf(“%d %d %d”, new_person.date_of_birth,
          new_person.month_of_birth,
          new_person.year_of_birth);
    ...
    return new_person;
}
```
Using arrow notation

- Still need to use parentheses to obtain address of field:

```c
person_record new_person;
...
success = get_person(&new_person);
...
int get_person(person_record* newp) {
    int OK = 1;
    scanf("%s %s", newp->surname,
          newp->given_name);
    scanf("%d %d %d", &newp->date_of_birth,
          &newp->month_of_birth,
          &newp->year_of_birth);
    ...
    return OK;
}
```
Combining data types

typedef struct {
    int year;
    int month;
    int day;
} date;

typedef struct {
    surname_t surname;
    surname_t given_name;
    int year_of_birth;
    int month_of_birth, day_of_birth;
    int height;
    int ID;
} person_record;
typedef struct {
    int year;
    int month;
    int day;
} date;

typedef struct {
    surname_t surname;
    surname_t given_name;
    date date_of_birth
    int height;
    int ID;
} person_record;

person_record player;
date my_date;
...
my_date.year = 1980;
player.height = 182;
player.date_of_birth.day = 1;
player.date_of_birth.year = 1987;
...
player.date_of_birth = my_date;
Exercise

Define a structure to account for this situation:

Cars have six-character registration numbers, and two dates associated with them – the date the car was first registered, and the date that the current registration expires. Each car also has fields (40-byte strings) for manufacturer, make, body type, and colour; and a field to record the number of owners it has had.

(Moffat)
Where to from here?

- Back to learning C….
  - structs (Ch. 8)
  - Dynamic memory allocation (Ch. 10)