CP Lab 7: structured types

Instructions

The purpose of this Lab is to help you get more programming experience with C, and introduce you to using structs in your programming. We will be building on what you have done in the previous lab and in your classes.

Some of you may still be behind with lab 6 from before the mock exam. Even if there are still questions from lab sheet 6 to be solved, we ask that you devote most of today’s lab to this new lab sheet.

If you do finish this lab before the two hours are up, then of course you may go back to earlier unfinished Lab sheets.

All Labs for CP will take place on the DICE machines in AT5.05 (Thursday and Tuesday) or AT6.06 (Monday). For working alone, you may use any of the DICE labs in AT, as long as they are not booked out by a class. You have 24-hour access to AT levels 3,4,5: if your card/pin does not allow you in, use the ITO contact form on our course webpage to contact the ITO.

Aims

- Get you extra experience with the DICE system, and its use for C programming.
- Give you experience of defining your own structure using typedef.
- Program a set of functions with respect to a ‘Running Club’ data structure.
- Follow up by re-visiting earlier unfinished Lab sheets if this is necessary!

To master the basic learning outcomes of the course, you should be able to do all the labs up to and including this one.

Prerequisites

Stage A Getting started

- Log into on of the DICE terminals.
- Bring up a terminal window, move to your home directory, and create a new subdirectory called lab7.
Stage B  Student Database

We will define a student database which includes name, surname, UUN, Department name, gender and age of the students. Your program should be named `studentDB.c`. Its main steps will be:

- Define a struct called `student_t` which contains the given field names `name`, `surname`, `UUN`, `department`, `gender` and `age`.

  In this struct, `name` is a `string`, `surname` is a `string`, `UUN` is a `string`, `department` is a `string`, `gender` is a char. For gender ‘f’ represents female, ‘m’ represents male, and ‘x’ represents other/undisclosed. Lastly, `age` is an `int`.

  You will use `typedef` to make this declaration. Check your lecture notes for lectures 15 and 16 to refresh your knowledge of how to do this.

- Using this struct definition in the main part of your program define an `array` of the `student_t` structs to contain six elements.

- Initialise the first three elements of the student array in the main part of your program. The initialisation should use the following data:
  
  - “John”, “Bishop”, “s1234”, “Inf”, ’m’, 18
  - ”Lady”, ”Cook”, ”s2345”, ”Eng”, ’f’, 21
  - ”James”, ”Jackson”, ”s3456”, ”Eng”, ’m’, 17

- Initialise the other three elements using the `scanf` function by asking the user to enter inputs using a `for` or `while` loop. When you do this, ask for each field separated by spaces, like this:
  
  Enter name surname UUN department gender age:
  so that the user then types
  Marmaduke Featherstonehaugh s9876 Phil x 23

- Write a `findOldest` function that takes a pointer to the start of the array as input, and a length for the array, and returns the details of the oldest student (name, surname, UUN, department, sex and age).

  If there is more than one student with this (maximum) age, you may choose an arbitrary student of that age to have their data returned.

  This function will satisfy the following function prototype:

  ```c
  student_t findOldest(student_t *studentarr, int len)
  ```

  Inside `findOldest` use a `for` loop to traverse each element of the given array to compare each age with each other.

  - To do this you may define a variable, maybe called `max`, to save the index of the student with the max age, initialising this to 0.

  - Then in the `for` loop compare the ages with `max`, and if the age of the current student is bigger than the age of the student at index `max` then reassign the value of `max` to be the current index.

  Finally return the `student_t` entry at index `max` as the result of `findOldest`.

- You will need to use `printf` at the end of your main to print out the details of the student returned by `findOldest`.

After you have written the program, please compile it using

```
gcc -Wall studentDB.c
```
Your result for studentDB.c

☐ Program should be correct. To test:
  – For the following inputs
    * "Alex", "Taylor", "s4567", "Inf", 'f', 23.
    * "Oliver", "Ford", "s5678", "Eng", 'm', 18.
    * "Daniel", "Thomas", "s6789", "Inf", 'm', 20.
  – The result printed out by your 'main' should be:
    Alex Taylor, s4567, Inf, f, 23

☐ Extra: can you alter findOldest to take a length as well as the array, instead of assuming length 6?

☐ Extra: can you write a function called pretty to take an item of type student_t and print out the details of that student in a 'pretty fashion'?

☐ Submit your program with
  submit cp lab7 studentDB.c

Stage C  Running Club database and functions

The starting point for this stage of the lab is the file running.c, which you may download from the course webpage. This file contains a suite of type declarations, as well as the function prototypes for this question (and the code for some helper functions). The first type declaration is a structured type runtime_t to store running results in hours, minutes and seconds. runner_t is another structured type containing the fields name, runid (unique id of the runner, guaranteed to be a positive integer), and two fields pb (personal best) and recent (most recent) of type runtime_t. The structured type runclub_t consists of an array of size MAXRUNNERS of type runner_t, together with a field total which we will use to maintain a count of the current number of runners in the running club. Below are details of the structured data types:

```c
typedef struct {
  int hr;
  int min;
  int sec;
} runtime_t;

typedef struct {
  char name[30];
  int runid;
  runtime_t pb;
  runtime_t recent;
} runner_t;

typedef struct {
  int total;
  runner_t runners[MAXRUNNERS];
} runclub_t;
```
typedef struct {
    char name[30];
    int runid;
    runtime_t result;
} result_t;

There is one extra structured type called result_t which is similar to runner_t except that it only contains one field for a running time. result_t is intended to be used when we are giving updates of recent races, or adding a new runner to the database.

In the file running.c, we have declared one global variable of type runclub_t named slowAC, and this will be the database which all our functions refer to. We assume that if there are \( k \) runners in the database, that slowAC.total will have the value \( k \), and the runners will be stored at indices 0, . . . , \( k - 1 \) of the slowAC.runners array.

The following list describes the tasks you are asked to solve. You will need to use the material you learnt in lectures 13 and 14, which shows you how to access various parts of a structured data type.

1. Your first task is to implement a function which takes a single parameter of type runtime_t, and computes the minutes-per-mile pace represented by this half-marathon time. You should use the fact that a half-marathon is 13.1 miles. The function prototype that you must complete is:

   double minpermile(runtime_t time)

2. The second task is to implement a function which takes no input arguments, but which returns the unique runner id (the value of the runid field) of the runner who has the fastest pb field in the global database slowAC. You must complete the following function prototype:

   int fastest()

3. The final task is to implement a function which takes one argument res of type result_t, and which either uses this input to update the entry of that runner (if res.runid has an entry in the database) or (if the runner with id res.runid is currently missing from the database) to add a new runner entry to the database, and update slowAC.total.

   When updating the entry, you must update the recent field; pb should also be updated if no better time than res.result is previously known.

   You must complete the function prototype below:

   int updateDB(result_t res)

   The value returned of the function should be 1 unless res.runid is a new runner and the database already has MAXRUNNERS (in which case you should return 0).

All your code should be entered into running.c. This file contains some helper functions, including a function to initialize the database with a collection of runners. There is also a detailed main function which runs a menu allowing various functions to be tested.

As you write and debug your functions program, please compile the program using

```
gcc -Wall running.c
```

Submit your program with

```
submit cp lab7 running.c
```

Julian Bradfield after Çiğdem Beyan and Mary Cryan, updated Nov 2016