Lab 8

Introduction to C
CSCI 112, Spring 2015

Objectives

- Learn to create and use a `struct`
- More practice with creating custom types
- Practice dynamically allocating arrays
- Practice pointer arithmetic
- Learn to redirect a file to `stdin`

Overview

For this lab, you will input and store a set of student records. For each student, you will keep track of their student id (SID), their name, and a list of course grades. From this information, you will calculate each student's GPA. After recording all the student information and calculating the GPA, you will print back all the records in your student database.

Student Record

Begin by defining a student record using a `struct`. In this structure, you will store the following fields:

- student id named `sid` (int)
- last name (char array)
- first name (char array)
- list of grades (float pointer)
- gpa value (float)

For the first and last name arrays, assume a maximum size of 25. Set a preprocessor directive `#define NAME_SIZE 25` and use this value to determine the size of the first and last name character array.

We also wish to store a list of grades in the student record. However, we do not know the size of this array in advance. Instead you should store a pointer to a float. You will later set this pointer to point to the beginning of an array of floating point values (the grades).

Lastly, wrap your structure definition with a type definition and name your type `student`.

> Protip: By using a `typedef` and naming it `student`, we can avoid writing out `struct student` each time we need to refer to the structure, and instead refer to just `student`.

`main` function

1. Prompt the user for the total number of students. Next, prompt the user for the total number of grades to record.

2. Print back to user the total number of students and the total number of grades.

   > PROTIP: Use the `sizeof` function to determine the size of any type, including custom defined types.

3. Using `malloc`, dynamically allocate memory for an array of students. This calculation will be based on the total number of students and the size of each element (the student record). Save this result as a pointer to your custom `student` type. This pointer points to first student in an array of student records. This array is initially empty, but you will populate it in the next step.

   > PROTIP: The `malloc` function returns a void pointer. Do not forget to cast this void pointer to be a pointer to a student record (`student *`).

4. Next, iterate over the total number of students. For each student, call your helper function to input an individual student's information (described below). This function will return a single student record. Save this student record in the appropriate spot in your array of students. Since we only have a pointer to the start of our array, you will need to use pointer arithmetic to access each array element.

   > PROTIP: Since you have a pointer to the start of your array of students records, you can add your loop counter to this address to find the appropriate array offset. Do not forget to dereference in order to set a value at a pointer.
   >
   > ```
   > *(pointer + offset) = ...
   > ```

5. Next, call a helper print function which will be responsible for iterating over all student records, printing out each student record one by one.

6. Lastly, use the `free` function to release the memory associated with your array of students. For this lab, do not worry about freeing each of the array of grades.

Function to Input Student Information

Write a function that is responsible for inputting information pertaining to a single student and storing that information as a single student record. This function will take in the total number of grades (int) to enter as its only argument. This function will return a single student record.

1. Declare a new student record. It is initially empty.
2. Prompt the user for the student id (SID) and store in the student record, in the proper field.
3. Prompt the user for a last name and store in the student record.
4. Prompt the user for a first name and store in the student record.
5. Using the total number of grades, create a new array of grades in which each grade is a floating point. Using `malloc` allocate memory for an array of floating points. Save this pointer in the appropriate spot in your student record.
6. Prompt the user to enter grades. Using a loop and `scanf`, input a list of grades. Store each grade in your array of grades.
7. Calculate the student's GPA using a helper function (described below). Save this value as the gpa field of the student record.
8. Return your (now populated) student record.

Function to Calculate the GPA

Write a function that takes in a single student record and the total number of grades (int) as arguments. This function will return a floating point value that corresponds to the student's GPA. The student's GPA should be calculated as the average of all of the student's grades.

PROTIP: To calculate an average you will loop over all grades in the array, calculate a sum of the grades, and divide by the total number of grades.

Function to Print all Student Records

Lastly, write a function that will print out the records of all the students. This function will return nothing. This function will take as arguments:

- a pointer to start of your array of students,
- the total number of students (int), and
- the total number of grades (int).

PROTIP: Recall the `->` operator is known as a structure dereference. The following two operations are equivalent.

- (* (pointer + offset)).field
- (pointer + offset) `->` field

You will begin by looping up to the total number of students. For each student:

1. Print out the student id.
2. Print out the student name in the format: first name, space, last name.
3. Print out the student grades all on one line with a space between individual values. Format to print **one digit after the decimal place**.
4. Print out the student GPA. Format to print **two digits after the decimal place**.

Redirecting a File to Standard Input

As you test and debug your program, you notice that there is a lot to type in each time. This can be very tedious. Fortunately, unix gives us a way to plug data from a text file in as standard input to a C program. First, create some data (example below) in a text file and save this file as `grades.dat`.

```
4 5
111 Domacese Rand 90.5 96.7 89 96 84.2
187 Uyi Guy 85.3 88.1 82.3 92.8 79.1
242 Wyle Dew 75.3 82.3 88.1 92.4 78.3
616 Pointer Noel 67.2 70.3 45.3 90.1 77.2
```

The first line gives the total number of students followed by the total number of grades. Each subsequent line gives the information for a single student: the SID, the last name, the first name, and 5 grade values. In this example, there are 4 students listed because the first line indicated there would be 4 students. Likewise, there are 5 grades values because the first line indicated that there 5 grades to record for each student.

PROTIP: We intend for you to only use `scanf` in this program, and not the `gets` function. Recall that `scanf` will delineate input based on any white space, but that `gets` uses only the EOL character. For the input data file to work properly as described here, use only `scanf` in your program.

To use file data directly as `stdin` to your `scanf` functions, use the following syntax when executing your program:

```
$ ./program8 < grades.dat
```

PROTIP: When using data from an input file, the values will not be echoed back to the terminal. Therefore some of your formatting may look a bit strange, but that is okay so long as the input data has been properly saved by your code. You can verify you are saving the input data correctly by viewing the output of your print function.

Example Execution – Manual Entry

```
$ ./program8
Enter the number of students:> 2
Enter the number of grades to track:> 3

There are 2 students.
There are 3 grades.

Enter information for student:
        Enter SID:> 101
        Enter last name:> Enright
        Enter first name:> Reed
        Enter grades (separated by space):> 70.1 60 92

Enter information for student:
        Enter SID:> 123
        Enter last name:> Claire
        Enter first name:> Heidi
        Enter grades (separated by space):> 82.5 96.1 89.0

Student ID #101:
        Name:   Reed Enright
        Grades: 70.1 60.0 92.0
        GPA:    74.03
Student ID #123:
        Name:   Heidi Claire
        Grades: 82.5 96.1 89.0
        GPA:    89.20
```

Example Execution – File Input

```
$ ./program8 < grades.dat
Enter the number of students:> Enter the number of grades to track:>
There are 4 students.
There are 5 grades.

Enter information for student:
    Enter SID:>  Enter last name:>  Enter first name:>  Enter grades (separated by space):>
Enter information for student:
    Enter SID:>  Enter last name:>  Enter first name:>  Enter grades (separated by space):>
Enter information for student:
    Enter SID:>  Enter last name:>  Enter first name:>  Enter grades (separated by space):>
Enter information for student:
    Enter SID:>  Enter last name:>  Enter first name:>  Enter grades (separated by space):>

Student ID #111:
        Name:   Rand Domacese
        Grades: 90.5 96.7 89.0 96.0 84.2
        GPA:    91.28
```

```
Student ID #187:
        Name:   Guy Uyi
        Grades: 85.3 88.1 82.3 92.8 79.1
        GPA:    85.52
Student ID #242:
        Name:   Dew Wyle
        Grades: 75.3 82.3 88.1 92.4 78.3
        GPA:    83.28
Student ID #616:
        Name:   Noel Pointer
        Grades: 67.2 70.3 45.3 90.1 77.2
        GPA:    70.02
```

PROTIP: Note that in the second example, the input values were not printed back to terminal. Furthermore, many of your prompts may appear on the same line. This is because there is no user hitting "enter" after each entry. This is the expected behavior when inputting data via a file and you do not need to adjust your program to compensate. Do ensure that when manually inputting information your exchange resembles the "Manual Entry" example.

Compile & Test

Compile your program using this `gcc` command. `c99` is a shortcut for running `gcc -std=c99`, which uses the C99 standard instead of the default C89 standard.

```
$ c99 lab8.c -o program8
```

Run your code, using a data file to input the information.

```
$ ./program8 < grades.dat
```

PROTIP: A segmentation fault is an error related to invalid memory access. This could be caused by passing a function an insufficient number of arguments, or perhaps passing the wrong types of arguments. A segmentation fault could also be caused by dereferencing a null pointer, among other reasons.

In this lab, there are many opportunities to make a memory mistake. To cope with this, we **recommend** you write small portions of your program and test them before moving on. This makes it easier to find your errors. It is also just good programming practice.

Resources

- You should attend the Friday Lab session to seek assistance from the TAs and CAs.
- For general questions, check the D2L `Lab 8 Help Forum` to see if another student has already asked your question. Otherwise, post your question on the D2L forum. This forum is intended for general questions that will benefit all students. Do not paste your code nor give away any specific answers to the lab on this forum.
- For specific questions, attend the weekly lab session or attend the TA's office hours.
- You are encouraged to use resources or tutorials on the internet to learn unix or C. Check the class resource list on D2L for some links to useful resources.

Submission

- Assigned: April 14th
- Lab Day: April 17th
- Due Date: April 24th, 8:00am

1. Make sure you included ample and informative comments – it is 20% of your grade!
2. Rename your C file to `last_first_lab8.c` and substitute your last and first name.

   Protip: If you fail to follow the above file naming conventions, your program may not be graded automatically and you will lose points.

3. Submit your `.c` file to the D2L dropbox Lab8

   Protip: Submit only your C file. Do not submit your object file or your executable program. Do not archive (e.g., `zip`) your file.

Each student will complete and submit this assignment individually. Submit your work before the deadline as late work will receive a 50% penalty. Labs submitted more than 24 hours late will not be accepted. The late deadline is Saturday, April 25th, 8:00am.