CSCB09 - Lab 4: Pointers, Structs and Linked Lists

Introduction

The purpose of this lab is to practice with pointers and structs, and introduce you to the concept of linked lists. Pay attention to the notation so that you are conscious of when you are taking the address of a variable, and when you are dereferencing a pointer. Draw pictures!

Remember, the TAs are there to help you. You should work in pairs and are welcome to ask other students for help too. Try finding a different partner this week. Meet someone new.

Preparation

The C programming concepts you will be using in the lab today are:

- pointers
- structs
- calling malloc and checking return value

The file `structs_lists.c` contains code for you to complete. Copy the file from (`/courses/courses/cscb09w19/nizamnau/labs/lab4/structs_lists.c`) into your directory, compile and run the program as follows:

```
prompt> gcc -Wall -g -o structs_lists structs_lists.c
prompt> ./structs_lists
```

In this lab, you will complete the four functions that are incomplete in structs_lists.c.

You will get the most out of this lab if you think through all of the steps instead of just trying to make the code work. The amount of code you will be writing for this lab is minimal, but it is important that you understand exactly what it does.

Task 1: Warming up with structs

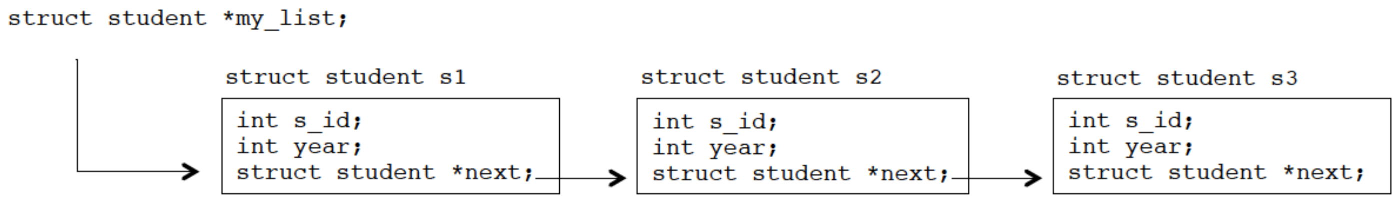
`structs_lists.c` implements code to manage student records in a department. Similar to the example in lecture it defines a struct to hold the data for a student. Open `structs_lists.c` in an editor and take a look at the definition of `struct student`. Don't worry for now about the purpose of the last member of `struct student`, which is a pointer called `next` to another `struct student`. Your first task is to complete the function `CreateStudent`.

Task 2: Intro to linked lists

Once you completed Task 1, your `CreateStudent` function could be used to create a new `struct student` whenever a new student needs to be registered. The question is how to keep track of all the students that are being created. One option would be to store them all in one big array. The disadvantage is that we don't know ahead of time how big the array would have to be since we don't know how many students will register in the future. Frequent calls to `realloc` to adjust the size of the array are wasteful.

Instead we will use a common data structure called linked lists. Linked lists manage a set of elements (called list nodes) by chaining them together using pointers. A node in a linked list is a `struct` that contains members to hold some data (e.g. the student id and student year in our example) and a pointer that points to the next node in the list.

The picture below shows what this data structure would look like for 3 students. There are three variables of type `struct student`, called s1, s2 and s3, that are linked using the `next` pointer provided as the third member of the `struct student` data type. For example, the `next` pointer of `s1` points to `s2`, i.e. the value that the pointer variable `next` in `s1` stores is the address of the variable `s2`. The program that manages the students that have been created now only needs to keep track of the first student in the list (by storing a pointer to it, which is named `my_list` in the example below), since the others can be found by following the chain of pointers.



Your task is to complete the function `BuildThree`, which creates three students and then links them together in a linked list as shown in the figure above.

Task 3: Adding a node at the front of a linked list

Often one wants to add a new node to an existing linked list. For example, after using `BuildThree` to build a list of three students one might want to create a new `struct student` and add it to the existing list. Your task is to complete the `Push` function, which creates a new student and adds it to the front of the list.

Task 4: Adding a node to the end of a linked list

Often lists are used to keep the elements in a particular order. Instead of adding a new node to the front of the list, one might want to add it to the end of the list. In order to append to the end of a linked list you need to be able to traverse the list and move to the end. Take a look at the `PrintList` function that we have provided to see how to do this. `PrintList` walks through the linked list of students and prints out each student. Your task is to complete the `Append` function.

Optional: Food for thought

In addition to adding nodes to a linked list, sometimes one also wants to remove a node from a list. Think about how you would implement a function that takes as an argument a pointer to the first node of a student list and a student ID and then removes the node in the list that corresponds to the specified student ID.

Instead of inserting at the head or the tail of a list, sometimes one wants to add a new node in the middle of a list. Imagine for example that you want to keep your list sorted by the student IDs. Then a new student would have to be inserted into one particular place in the linked list. How would you write a function that does this insertion?

You might want to take a look at chapter 17.5 in the textbook if you are having trouble understanding the concepts of linked lists.