Session 4 - Optional Session

This is an optional session featuring only exercises; please have a go at the first 3 sessions to learn the language before you attempt these.
For an alternate style of exercise, see Other Exercises !

Good luck and happy coding!

Structs

The first time this course was taught, structures were not introduced so this session briefly introduces structs if you need it!

We know how to group items of the same type together using arrays, however what about items of different types?

In languages like Python, we have tuples. In some, we simply use objects. In **C**, we have structs!

```c
struct Person {
    char name[20];
    int age;
    int earnings;
};
```

Typically, we declare structs like this near the top of the file (outside of main).
This effectively constructs a new type of object, so we can now declare variables of type `Person`! The variables in the struct are called **members** of the struct.

```c
struct Person me;
```

Okay so, how do we access members and initialise them? We can use `aStruct.memberName` and assign!

```c
me.age = 19;
me.earnings = -9250;
```

Arrays in Structs

There is an issue when we directly assign with arrays though: this is where the distinction between an array and a pointer becomes really important. A pointer could be assigned to easily, however you can't assign values to an array; you have to fill each individual element.

```c
me.name[0] ='e';
me.name[1] = 'd';
```

This is kind of annoying for strings, so we can use the `strcpy` (string **copy**) function from `<string.h>`.

```
strcpy(me.name, "Edward");
```

The reason this distinction exists is because structs store their members in contiguous memory locations. You can store a *pointer* in that contiguous memory location, which points to some far-off location though.

Pointers to Structs

Structs can be passed through functions fine; we don't always have to pass a pointer to the struct. However, this may mean copying lots of data and also taking up a lot of stack space which can be extremely important in tiny embedded systems. Of course, if you want a struct to be changed by a function, you have to pass a pointer to it (like an int, a char, or any other primitive type).

Using pointers to structs is very straightforward; the main difference is that instead of using `aStruct.memberName`, we use `aStruct->memberName`.

```
void printingStruct(struct Person* structPointer){
    printf("Name: %s \n", structPointer->name);
    printf("Age: %d \n", structPointer->age);
    printf("Earnings: %d \n", structPointer->earnings);
}
```

Key Points

Key differences between structs and arrays:

- structs have a specific size/number of members
- structs can contain members of different data types

It's time for the spooky end to the C Course!

You're an up-and-coming career politician working for Party C, who has crawled up to the ranks and are primed to be a perfect candidate for Mayor of London - but oh no! You've made a crucial mistake; you coded something in Python! To punish you for using a language other than C, the party leader has relocated you to... Spookesville? You've never heard of this village, but if there's any way you're making it back to the big leagues, it's through making a name for yourself in this village.

Fortunately, you learn on arrival that the mayoral elections are happening this week! If you become the mayor of Spookesville, then *surely* you'll gain the trust of the Party again and be promoted out of this eerie little village in a year or so. The voting will end on Halloween, so you need to gain votes and fast! You learn at the local pub that the big players around the village are the **Haunts of Halloween**; you're not sure why they have such an ominous title, but years of being a career politician have told you to not ask questions and to impress the important people. You've heard that some of them have been having trouble around town, and could use your assistance!

With each exercise, you'll be asked to help a **Haunt of Halloween** and each Haunt will have their own particular demands; solve each of their problems to gain their trust and win the vote to become mayor!

Note: It may be easier for some to solve Problem 2 first.

More Bats, More Problems

You decide to enter the ominous-looking castle as there'll surely be a Haunt there! Lo and behold, a pale-white figure - surrounded by a swarm of bats - is crouched over a laptop and muttering curses; it's Dracula!

1. Dracula has far too many bats to keep track of and is looking to cut down on his flock of bats (fun fact: there are 4 collective nouns for bats!) and he wants you to write a program to store the names of the bats; the problem is that even Dracula does not know how many bats there are! You know from your University days that this requires a data structure that can **grow** to fit more elements.

Dracula races the bats against each other, and feeds you the bat names in order of where they placed (e.g. the first bat is the bat in 1st place, the second bat is the bat in second place etc). Dracula wants to remove the `n` slowest bats from his flock, and gives you this number as the first input. Since you need to remove `n` of the latest elements added to the data structure, you recognise that you could code a **stack** for this problem!

Output the bats you removed (in order of removal), followed by the remaining flock (in order from fastest to slowest). A typical input to this problem will look like this:

```
2
Neal
Patrick
Sharon
Bob
Jonathan
```

In this case, Dracula wants you to remove the 2 slowest bats (Bob and Jonathan), so the output should be:

```
Jonathan
Bob

Neal
Patrick
Sharon
```

Input: An integer `n` (to represent number of bats to eliminate) followed by a number of lines with each line being the name of a bat.

Input Constraints: Each name is guaranteed to be fewer than 20 characters, and all in one word.

Test your solution by feeding it an input file in `txtFiles/BatInputs` using:

```
cat txtFiles/BatInputs/inputFileName.in | ./yourExe
```

Hint 1: There is a function *feof* (in `<stdio.h>`) that lets us test for if a given stream is at the end of a file. For example, `feof(stdin)`.

Hint 2: You can use **malloc** and **realloc** to dynamically allocate memory for an array. You can create a growable Stack (or other data structure) using this.

Hint 3: A string can be thought of as a pointer to a bunch of chars (the string would have type `char*`). An array of strings can be thought of as a pointer to a bunch of pointers which each point to a bunch of chars. This would have type `char**`.

Hint 4: There is an implementation of an IntStack in `cFiles/exerciseHints/IntStack.c`.

Hint 5: Solving the problem, but for integers: `cFiles/exerciseHints/RemovingFromIntStack.c`.

2. Solve the problem for names with an arbitrary length.

Sorting Witches

You see mysterious purple fumes rising above the town centre, from what looks like a building shaped like an enormous pot.

You ask a local and they say, "Oh you don't know? That's the **Potion Factory**, run by the Cauldron Coven. I saw their owner, Esmeralda, looking very stressed..."

You go to the factory and see a witch frantically

1. Esmeralda needs to pay the witches based off how many potions they've produced; she's organised a list of the witches (**W**) and their outputs (**O**), as well as a list of payouts (**P**).

You need to map each witch to their payout, using their known output. The payouts are calculated using a complicated process, from the outputs of each witch.

However, you know that for any two witches, the witch with a higher output has a higher payout i.e. for functions **Pay : W -> P** and **Output : W -> O**, we have **w1,w2 ∈ W. (O(w1) > O(w2)) -> (P(w1) > P(w2))**.

A typical input will look like this:

```
4
8 7 22 31
Agnes 10
Esme 21
Cassandra 7
Circe 36
```

The corresponding output:

```
Cassandra 7
Agnes 8
```

```
Esme 22
Circe 31
```

Input: An integer `n` followed by a line with `n` integers (each a payout), followed by `n` lines with the witch name (one word) and an integer (their output).

Input Constraints: Each witch name is fewer than 15 characters.

Hint 1: You can group together the name and the output using a struct. If you wish to sort these structs, you need a way to compare structs.

Hint 2: `cFiles/exerciseHints/ReadingWitches.c` is a program for reading in the data correctly.

Extension: Are there more efficient algorithms you can use? What about if there are millions of witches?

Other Exercises

There are many resources to practice **C** on, and a nice beginner-friendly one is HackerRank!

Create an account as a **developer**, and go to the C topic.