CSC209 Lecture 4: Strings and Structs

David Liu, Department of Computer Science, University of Toronto

Navigation tip for web slides: press ***?*** *to see keyboard navigation controls.*

Announcements

- Assignment 2 has been posted!

Strings

A **C string** is a `char` array that ends with a null terminator character, `'\0'`. C strings can be allocated...

- on the stack

```
char name[32];
char greeting[6] = "Hello";
```

- on the heap:

```
char *name = malloc(sizeof(char) * 32);
```

- in “read-only” memory, with a string literal:

```
char *name = "David";
```

Recall:

- Arrays only allocate space for their elements, not their size.
- Typically pass array size to functions:

```
... my_function(int *numbers, int size) { ... }
```

Strings:

- Strings are `char` arrays, so also do not allocate space for their size.
- Instead, strings use the null-terminator `\0` to mark the end of a string.

Whenever you create or modify a string, you **must ensure it is null-terminated**!

Worksheet: strings.pdf

`strings.c` starter code

structs

A **struct (structure type)** describes a sequentially allocated nonempty set of member objects.

```c
// struct type declaration
struct faculty {
    char *name;
    char area[16];
    int num_students;
};
```

```c
// struct object declaration
struct faculty david;

// struct object initialization
david.name = "David";
strcpy(david.area, "Being awesome");
david.num_students = 4;
```

Functions typically take a pointer to a struct rather than a struct:

```
// Yes
... f(struct faculty *prof, ...);

// No
... f(struct faculty prof, ...);
```

1. Avoid making a copy of (potentially large) struct
2. Allows function to modify a struct object declare outside its scope

Syntax note: `obj->member` is equivalent to `(*obj).member`

Worksheet: structs.pdf

`structs.c` starter code