DS

Week 1

Structs and

Dynamic Memory Allocation Review

Structs Review

What is a Struct?

A struct is your own data type (like the definition of a record or a class in other languages).

You organize related fields into the struct.

It is not a variable. It is a data type.

Review!

What do you have to do before declaring a struct variable?

Example

```
struct Part
{
    char partNumber[8];
    char name[20];
    double cost;
};     // note semicolon at end
```

This creates a new data type called *struct Part.*

The items within the braces are called *fields*.

Struct Variables

You can declare a struct variable like this:

- struct Part widget;

It can be initialized through an initializer list like this:

- struct Part widget = { "123KDE9", "Grapplegrommet", 133.21 };
- An initializer list can only be used when declaring the variable.

Review!

What’s the data type of *widget?*

Remember ...

Creating a definition for a struct does not create a variable.

Only the declaration of a variable of the struct's data type creates a variable.

The name of the data type includes the keyword *struct*. This is different from C++.

Fields

A struct variable's fields can be accessed by using the dot operator.

- e.g. widget.cost = 14.32;
- This behaves just like a variable of the field's data type.

Review!

What's the difference between a struct and a field?

Arrays of Structs

You can also have arrays of structs.

- e.g. struct Part parts[100];
- e.g. parts[30].cost = 9.33;

Put the array index next to the variable name, before the dot.

Initialization of Arrays of Structs

Arrays of structs can be initialized through nested braces.

```
e.g. struct Part parts[100] = {
 { “123KDE9”, “Grapplegrommet”, 133.21 },
 { “123JKK9”, “Glortznitz Adapter 2.6”, 20.22 },
 { “123JDK9”, “Glortznitz Adapter 5.4”, 21.42 },
 { “123JUK9”, “Glortznitz Adapter 6.4”, 18.22 },
 { “123JIK9”, “Glortznitz Adapter 5.2”, 22.15 },
 { “123JIL9”, “Glortznitz Adapter 4.3”, 20.27 }
};
```

Passing Structs As Parameters

You can pass a struct as a parameter, but it's rarely done

Usually, you pass a pointer to the struct.

Review!

Why is passing a pointer to the struct more efficient?

Review!

Why is passing a pointer to the struct more dangerous?

Notation

There is special notation to access a field of a dereferenced pointer to a struct.

- The -> operator can replace the asterisk and dot around the pointer variable.

The following are equivalent:

- (*pPart).howMany = 9;
- pPart->howMany = 9;

Array of Struct Notation

Referencing elements of an array of structs is done in order, like this:

- thingy[1].screw[3].cost = 9.33;
    - This is the cost of screw #3 of thingy #1.

Dynamic Memory Allocation Review

malloc()

void *malloc(size_t howMuch)

Takes a parameter that says how many bytes you want

Returns a pointer that you can assign to any pointer data type OR NULL if failed

Values being pointed to have random values

Review!

What's the parameter that malloc needs?

Review!

What's another function you can use for memory allocation?

Review!

What are the differences between those two memory allocation functions?

Example

```
if( (pBlock = (char *) malloc(50)) ==
   NULL )
{
        // out of memory
        // write code here to handle it
}
```

Remember to **always** bypass the rest of your code if you run out of memory.

Review!

What happens if you don’t bypass the rest of your code when you run out of memory?

free()

When done with the memory, need to call free(), passing the pointer gotten

Once freed, you can't access that memory any more

Exercise

Write a program that takes in 10 strings containing MP3 filenames and stores the artist and title in an array of structs to be displayed.

The user will input ten strings of the following form:

- artist
- a dash (‘-’)
- title
- “.mp3”

Assume the input string
will be less than
80 characters long.

Assume that the song title and artist both don’t have dashes in them.

Define a structure called *songInfo* that has two fields:

- a pointer to a string for the artist
- a pointer to a string for the title

Both pointers will be set up by using malloc() once you know how long each string is.

Don’t forget to add one to the length for the null-termination.

Define *songInfo* before any of your functions.

This does not violate the banning of global variables since these are data types, not variables.

In main(), declare an array variable of *struct songInfo* of size 10.

Write a function that takes a **pointer to a *struct songInfo*** (which is a pointer to an element of the array) and a string and gets song information from the string, filling in the array element.

- Note the data type of the first parameter. You are taking in a pointer to a struct, not an array and not a struct.

This function is also where you will allocate the two blocks of memory to contain the artist string and title string.

Write a function that takes the array of structs as a parameter and prints all of the information contained within the array in a nicely formatted fashion, one song per line.

You must use printf() width specifiers to help with your formatting.

Both of the preceding functions should be called from main().

Note that the first function must be called once for each song string while the second function must be called only once in total.

Don’t forget to free the memory allocated once you are done.

Have this complete
for next class
(and bring it to class).

Summary

1. You need to remember how to work with structs and dynamic memory allocation.
2. Do the exercise.
3. Bring the exercise to next class.