CSE 1320 - Intermediate Programming

Structs and Typedefs

Alex Dillhoff

University of Texas at Arlington

Aggregate Data Types

Aggregate data types are design to store multiple values.

We have been using arrays, which is an instance of an aggregate data type.

Aggregate Data Types

Aggregate data types are not necessarily restricted to multiple values of a single type.

C `structs` permit the storage of multiple data types within one entity.

Structs

Structs are created in C to represent complex data. If we wanted a struct to represent a user in a generic database, the declaration would like look this:

```
#define MAX_STR_LEN 128
struct user {
    int id;
    char username[MAX_STR_LEN];
    char password[MAX_STR_LEN];
    char email[MAX_STR_LEN];
}
```

Structs

Each individual data type declared within the `struct` is referred to as a **member**.

The previous example created a `struct` with 4 members.

Structs

A `struct` in C can have almost any data member with a few exceptions:

- A member cannot be a function.
- A member cannot have type `void`.
- The `struct` cannot have a member with the same type as the `struct`.

Structs

Note that the name of the `struct` given in the previous declaration is not the name of an individual variable.

Struct Declaration

To create an instance of the previously declared `struct`, the declaration would be

```
struct user user_var;
```

Here, the type is `struct user` and the identifier is `user_var`.

Struct Declaration

It is possible to declare multiple variables of a `struct` in one line.

```
struct user user1, user2;
```

Struct Declaration

The declaration of the struct can be combined with the declaration of variables.

```
#define MAX_STR_LEN 128
struct user {
    int id;
    char username[MAX_STR_LEN];
    char password[MAX_STR_LEN];
    char email[MAX_STR_LEN];
} user1, user2;
```

Struct Initialization

It is possible to initialize a variable of a `struct`.

For example, we could assign data to a newly declared `struct user` with the following syntax:

```
struct user user_var = {
    1,
    "praxideke",
    "Hy1810",
    "prax@gbr.io"
};
```

Struct Initialization

Similar to other aggregate types, it is possible to initialize partial data by omitting the rest of the members.

Example

```
    struct user user_var = { 1 };
```

The rest of the members are zeroed out.

Accessing Members

The member of a `struct` can be accessed using **dot notation**.

Example

```
user_var.username;
```

Structs

Example: Print struct members.

Structs and Memory

When creating a `struct` in memory, space is allocated for each member.

This implies that the `sizeof()` used with a `struct` variable returns an accurate size.

Structs and Memory

When a `struct` is created, it is possible that there are unused bytes in between each data member.

This is dependent on the system the program is executed on.

Structs and Memory

Further Discussion: Padding and Packing
https://stackoverflow.com/questions/4306186/structure-padding-and-packing

Structs and Memory

Example: Observe the size of the struct and all of its members.

Arrays of Structs

Since a `struct` is a data type, it can be created as an array. Consider the declaration:

```
    struct creature dragons[5];
```

which creates an array of `struct` with size 5 to store creature data.

Arrays of Structs

Accessing individual elements is similar to any other array:

```
dragons[0];   // First member
dragons[1];   // Second member
...
```

Arrays of Structs

Similarly, accessing members of each element is as easy and using the dot notation on the element that was accessed.

```
dragons[0].name;
```

Arrays of Structs

Since a struct can be initialized with an assignment, so can an array of struct.

```
struct creature dragons[5] = {
    { "Brimscythe" },
    { "Vorugal" },
    { "Umbrasyl" },
    { "Raishan" },
    { "Thordak" }
};
```

Struct Pointers

A pointer to `struct` can be created just as a pointer to any other data type.

```
struct creature *creature_ptr;
```

Struct Pointers

When working with a pointer to `struct`, the syntax to access the members changes slightly.

```
creature_ptr->name;
creature_ptr->hp;
```

Struct Pointers

Pointers to `struct` allow a loophole to the previous restriction on member data types.

A `struct` may not have a data member which is of its own type.
However, it may have a pointer to that type.

Struct Pointers

Example: Quake 3 `image_s`

Structs as Function Arguments

Passing a `struct` as an argument to a function is similar to any other data type.

Example: Read data into struct pointer

Typedefs

Typedefs in C are used to associate a given identifier with an existing type.

Its usefulness is immediately apparent when considering `struct`s.

Typedefs

Consider the following example of creating a new struct with a type definition.

```
typedef struct {
    char name[100];
    int hp;
    int ac;
    int speed;
    int cr;
} CREATURE;
```

Typedefs

The corresponding variable declaration for this new type would then be

```
    CREATURE dragon;
```