Structs II

Chapter 16

Objectives

To be able to

- Access the members of a struct using a pointer to the struct
- Write functions that take pointers to structs as parameters.

Example struct

- We will use the example from the previous structs slide:

```
typedef struct BOX
{
    int length;
    int width;
    int height;
    double weight;
    char contents[32];
} box_t;
```

Accessing members with a pointer

- We often need to access members of a struct using a *pointer* to the struct.

- There is a new operator for this: ->

- Example:

```
box_t* pBox1;
...
pBox1 = &box1;
```

`pBox1->length` is the same as `box1.length`

Accessing members with a pointer

- Use the “arrow” to get at members using a pointer to the struct.
- Use the “dot” to get at members using the name of the struct.

Accessing members with a pointer

```
typedef struct BOX
{
    int length;
    int width;
    int height;
    double weight;
    char contents[32];
} box_t;

int main (void)
{



    int dimension_total;

    box_t box1 = {24, 12, 12, 5.3, "Fine German Wine"};
```

Accessing members with a pointer

```
box_t* pBox1 = &box1;
```

Declare and initialize pointer to struct.

```
    printf ("Length of box1 is %d\n", pBox1->length);

    printf ("Box1 contains %s\n", pBox1->contents);

    dimension_total =
         pBox1->length + pBox1->width + pBox1->height;

    printf ("Sum of the dimensions is %d\n", dimension_total);

    return 0;
}
```

Accessing members with a pointer

- Question: Why don't we use the * to dereference a pointer to a struct?
- Answer: We could.
- **`(*pBox1).length`**

  is the same thing as

  `pBox1->length`
- The parentheses are necessary.
- **`*pBox1.length`** gets a compile error

Accessing members with a pointer

- The “arrow” notation is just nicer syntax.
  - **`(*pBox1).length`** is clunky!
  - **`pBox1->length`**

    is easier to write and easier to read.
- Always use this form rather than the “dot” form when working with pointers to structs.

Passing a pointer to a struct to a function

Function gets a *pointer* to a box struct in the caller's scope.

```
void display_box ( box_t* pBox )
{
    printf ("Box length is %d\n", pBox->length);
    printf ("Box width is %d\n",  pBox->width);
    printf ("Box height is %d\n", pBox->height);
    printf ("Box contains %s\n",  pBox->contents);
}
```

Use pointer to access members

```
int main (void)
{
    box_t box1 = {24, 12, 12, 5.3, "Fine German Wine"};

    box_t* pBox1 = &box1;

    display_box (pBox1);

    return 0;
}
```

This is an example of "Call by Address"

Passing a pointer to a struct to a function

- In this case, the function can modify the caller's struct.

```
void shrink (box_t* pBox)
{
    pBox->length = pBox->length/2;
    pBox->width =  pBox->width/2;
    pBox->height = pBox->height/2;
}
```

Passing a pointer to a struct to a function

```
int main (void)
{
    box_t box1 = {24, 12, 12, 5.3, "Fine German Wine"};
    box_t* pBox1 = &box1;

    display_box (pBox1);
    shrink (pBox1);
    printf ("\nAfter call to shrink\n");
    display_box (pBox1);

    return 0;
}
```

Passing a pointer to a struct to a function

```
turnerr@login0:~/test
[turnerr@login0 test]$
[turnerr@login0 test]$ gcc -Wall shrink.c
[turnerr@login0 test]$ ./a.out
Box length is 24
Box width is 12
Box height is 12
Box contains Fine German Wine

After call to shrink
Box length is 12
Box width is 6
Box height is 6
Box contains Fine German Wine
[turnerr@login0 test]$
[turnerr@login0 test]$
```