CS16, 10S, **Handout to go with H18** (structs and pointers) (printable PDF)

Available online at: http://www.cs.ucsb.edu/~pconrad/cs16/10S/homework/H18/handout
The assignment is available at http://www.cs.ucsb.edu/~pconrad/cs16/10S/homework/H18

---

This handout—which is your reading assignment for H18—covers the use of pointers with structs—which is not covered in the textbook until Chapter 7. We are covering them a bit earlier than the textbook coverage, because they will open up some more interesting problem solving opportunities. This builds on the handout from H13 the handout from H14, and the handout from H17.

If you want to read some additional material about structs, you may read Section 7.1 and 7.2 in the Etter text, or Section 12.1 of the online Oualline text (on-campus link, off-campus link)

---

## structs and pointers

Let's look at how structs work with pointers, using the example of struct Time (see box at right.)

```
struct Time
{
  int hours;
  int minutes;
};
```

Suppose we have a variable t1 of type `struct Time`:

```
 struct Time t1
```

If we write `&t1` that means *address of t1*—i.e. the location of `t1` in memory.

- What we end up with is a `struct Time *`
- When reading out loud, we read this as *struct Time star*
- What it means is *the address of a struct Time*, or equivalently, *a pointer to a struct Time*

On the second page of this handout, we'll look at where a `struct Time *` could be useful in practice—but first, let's just review some syntax related to pointers, and introduce one new piece of syntax.

## Declaring a pointer to a struct

When declaring variables, we use:

- `struct Time` for a variable that really *is* a struct Time (i.e. it has space for the `hours` part and the `minutes` part)
- `struct Time *` for a variable that only contains a memory address— someplace a `struct Time` can be found.

Example:

```
struct Time t1;  // contains fields t1.hours and t1.minutes (8 bytes long)
struct Time *tp; // a pointer—32 bits (4 bytes).  Stores an address
```

We can initialize the tp variable using the & operator so that it points to t1:

```
tp = &t1;
```

Once we do that, then t1 and (*tp) refer to the same place in memory—this is the "dereference operation" your read about in Chapter 6 (and worked with in homeworks H15 and H16). We can then access the fields of t1 through the pointer tp in two different ways. That is, in each row, all three expressions are, in this instance, interchangable:

| This expression: | is equivalent to this one: | or this one: |
|---|---|---|
| `t1.hours` | `(*tp).hours` | `tp->hours` |
| `t1.minutes` | `(*tp).minutes` | `tp->minutes` |

In general, for any expression of the form `(*a).b`

- For the expression to be legal, `a` must be a pointer, and `b` must be a field in the thing that `a` points to.
- The expression `(*a).b` is equivalent to `a->b`

**Please turn over for more...**

...continued from other side

### Why do we care about using structs with pointers?

Using structs with pointers turns out to be a very important aspect of C/C++ programming.

Having this capability opens up a lot of possibilities for structs to "refer" to other structs by pointing to them. This allows us to encode *relationships between various pieces of data*—for example, the relationship between a struct for an airline flight (say United Airlines flight 6840) and structs that represent the departure and arrival airports for that flight (SBA and SFO).

In CS24, this concept is extended to relationships between *objects*—which are similar to *structs* in C, but with a few extra rules. In addition to encoding relationships, we can also use pointers in structs to make *linked lists* and *trees*— these are ways of grouping data items together that provide an alternative to arrays. Linked lists and trees have many nice properties that arrays don't have, especially for making programs run fast.

In any case, in CS16 we probably won't get as far as covering structs (or objects) that refer to other structs—even though that's one of the main motivations for covering structs with pointers.

I bring this up because for the examples I will show you, there are alternative ways of solving the problem that don't use pointers with structs—and this may give you the false notion that learning how to use structs with pointers isn't that important. **Nothing could be further from the truth!** Rather, using pointers with structs is so crucial that I want to be sure you understand it. Therefore, we are covering it as early as possible in the course, to give you plenty of practice with this important topic.

### Three ways of assigning a value to a struct—case study in using pointers with structs

One of the ways we use pointers with structs is when we are passing a struct to a function, and we want to give that function access to modify the struct directly. For example, the following function will assign the values of the `h` and `m` parameters to the `hours` and `minutes` members of a `struct Time`.

As a reminder, here's the definition:

```
struct Time
{
  int hours;
  int minutes;
};
```

```
// assign a struct Time a new value, using pointers
void setTime(struct Time *t, int h, int m)
{
  (*t).hours = h; // or we could write t->hours = h;
  (*t).minutes = m; // or we could write t->minutes = m;
}
```

If we have a struct time called t1 declared as follows:

```
struct Time t1 = {7,10}; // initialize to 7:10
```

then we can call this function to change t1 to the time 8:30 with this function call:

```
setTime(&t1, 8, 30); // change t1 to 8:30
```

This would be an alternative to writing:

```
  t1.hours = 8;
  t1.minutes = 30;
```

The program `structPointers.c` in the code directory for this assignment illustrates both of these alternatives in the context of a complete C program—and also shows yet another way to initialize a struct with a function (without using pointers.) You are encouraged to go online and access this complete program through the links provided in the previous sentence. You can also copy this code into your own `~/cs16/H18` directory with this command:

```
mkdir ~/cs16/H18
cp -r /cs/faculty/pconrad/public_html/cs16/10S/homework/H18/handout/code/* ~/cs16/H18
```

After you copy the code, you can compile and run the code for yourself to see how this works. You may like to experiment with the code to investigate further how struct pointers work—this may help you with the H18 homework assignment.

---

End of handout that goes with H18