CISS245: Advanced Programming
Assignment a06

Objectives: The purpose of this assignment is to build a simple library for a struct.

1. Declare a struct variable.
2. Access member variables of a struct variable.
3. Write function/operator with struct parameters.
4. Write function/operator with struct return value.

For CISS245, I also emphasize rigorous software testing. Much of the testing strategies that you will learn in this class (although in a small scale) is actually being practiced in the real software engineering world. Specifically, for this and future assignments I will emphasize the testing of the smallest units of a software such as functions. These are called **unit tests**.

Structure Variables

A structure variable is just a variable that contains variables (refer to your notes). Run this:

```
#include <iostream>
#include <iomanip>

struct Time
{
    int hour;
    int min;
    int sec;
}

int main()
{
    Time t0;
    t0.hour = 5;
    t0.min = 18;
    t0.sec = 0;

    std::cout << std::setw(2) << std::setfill('0') << t0.hour
              << ':'
              << std::setw(2) << std::setfill('0') << t0.min
              << ':'
              << std::setw(2) << std::setfill('0') << t0.sec
              << std::endl;

    return 0;
}
```

In this case, I want to work with time which is represented by hour, minute, and second. However, I frequently want to view hour, minute, and second not as three things but one. So I create this new type:

```
struct Time
{
    int hour;
    int min;
    int sec;
}
```

I can then create a `Time` (structure) variable like this:

```
Time t0;
```

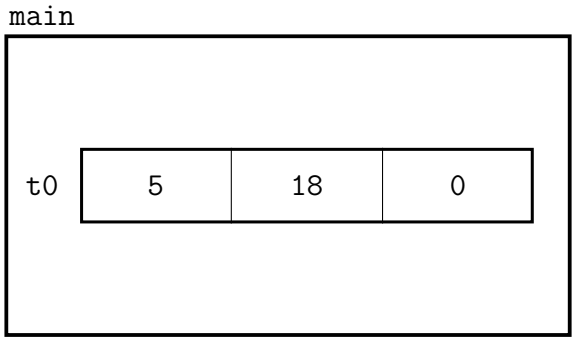
And now `t0` contains three variables. `t0` has:

- `t0.hour` which is an `int` variable.
- `t0.min` which is an `int` variable.
- `t0.sec` which is an `int` variable.

The variables inside a structure variable are usually called **member variables**.

Memory model

Here's the memory model of `main()` for the variable `t0` in the above example:



(Like I said, `t0` is just a variable containing variables.)

Here's another example of a structure type:

```
struct Student
{
    int id;
    double gpa;
}
```

In this case, a `Student` variable contains 2 variables. For instance, after this declaration:

```
Student jdoe;
```

the variable `jdoe` contains:

- `jdoe.id`, an `int` variable and
- `jdoe.gpa`, a `double` variable.

Note that the variables in a structure can have different types. In the `Student` case, there is one `int` variable and one `double` variable.

Note also that to go into a variable inside a structure variable, you use the dot. For instance, to go to the hour of `t0`, you use:

```
t0.hour
```

The variables inside a structure variable are just like any regular variable so you do know how to work with them (input, output, operators etc.) For instance, back in the first `Time` example, we assign values to the variable `t0`:

```
t0.hour = 5;
t0.min = 18;
t0.sec = 0;
```

We also read the values in `t0` and print them:

```
std::cout << std::setw(2) << std::setfill('0') << t0.hour
          << ':'
          << std::setw(2) << std::setfill('0') << t0.min
          << ':'
          << std::setw(2) << std::setfill('0') << t0.sec
          << std::endl;
```

(By the way, although a structure variable is sort of like an array, you see from the above that there are differences. For instance, an array can only contain values of the same type. You can't have an array of 2 integers and 3 doubles. An array is an `int` array or a `double` array. You can't have it both ways! To go into an array, you have the bracket `[]` with an index value. You get into a structure using the dot and a name for the variable inside the structure that you're trying to access.)

Initialization

Instead of doing this:

```
Time t0;
t0.hour = 5;
t0.min = 18;
t0.sec = 0;
```

to initialize structure variables, you can use the same notation as in array initialization:

```
Time t0 = {5, 18, 0};
```

Functions: struct parameters

Just like variables of basic types, it's not surprising that you can pass structure variables into a function. Run this:

```
#include <iostream>
#include <iomanip>

struct Time
{
    int hour;
    int min;
    int sec;
};
```