Structures in C

A structure is similar to an array in that it is an aggregate data structure, meaning it is composed of multiple elements or parts. It differs from an array in that each element can be of a different type. (Arrays are homogeneous structures where all elements are of the same type.)

The general syntax of a struct is as follows:

struct tag

{

member1

member2

...

} variable-list;

The tag and variable-list are optional, but the structure declaration must end with a semicolon. The tag is the name of the struct, and is used to reference it later. Unlike an array, this structure declaration does not immediately create an object: it acts like a template, allowing us to create objects later on. In many ways it acts like a built-in type such as int or double, except that it contains multiple variables inside it.

If we don’t include a tag, we don’t have a way to reference the struct and create objects of this type. However, if we know we are only going to use this structure once, we can use the variable-list section to immediately create an object.

In the example below we create a structure declaration that defines a struct type named Time and use that struct type to create an object named timeData.

struct Time

{

int hours;

int minutes;

int seconds;

};

struct Time timeData;

In the next example we create a single struct object named timeData. Since there is no tag, we can’t create any other objects from this structure.

struct

{

int hours;

int minutes;

int seconds;

} timeData;

We can also combine both of these methods and create an object from a named structure.

struct Time

{

int hours;

int minutes;

int seconds;

} timeData;

If we looked at these structures in memory, they would look just like an array of three ints. In both cases three integers are placed in memory one after the other. The difference, however, is how these integers are accessed. Structure members are accessed by name, using the dot operator.

struct Time timeRecord;

timeRecord.hours = 8;

timeRecord.minutes = 45;

timeRecord.seconds = 10;

Another difference is that a structure can be assigned to another structure.

struct Time time1;

struct Time time2;

time2 = time1; /\* all the members of time2 are now identical to time1 \*/

# Initialization

Structures are initialized much like arrays.

struct Time time1 = { 10, 15, 5 }; /\* 10:15:05 \*/

struct Time time2 = { 10, 15 }; /\* 10:15:00 \*/

struct Time time3 = { 10 }; /\* 10:00:00 \*/

struct Time time4 = {}; /\* Illegal \*/

Note: using less initializers than there are fields may give you a compiler warning.

# Nested Structures

Structures can contain any type of variable, including other structures.

struct Date

{

int month;

int day;

int year;

}

struct DateTime

{

int id;

struct Date date;

struct Time time;

}

The members of nested structures are accessed with an additional dot operator, and they can be initialized with an additional pair of curly brackets inside the overall initialization.

struct DateTime dateAndTime = { 1234, { 11, 24, 2000 }, { 0 } };

dateAndTime.date.month = 12;

dateAndTime.time.hour = 12;

# Typedef

We can use typedef to avoid having to use the struct keyword every time we create a new structure variable.

typedef struct tag

{

member1

member2

...

} typedef\_name;

Note that what we put on the last line before the semicolon is now the name that we use to reference the typedef’d struct, not the name of a variable we are creating. We can’t typedef a struct like this and create a variable at the same time.

If we do this for our Time structure:

typedef struct Time

{

int hours;

int minutes;

int seconds;

} TimeStruct;

We can now declare our Time objects without the struct keyword. Because we left the tag on the first line of the declaration, we can also still create variables the old way too.

TimeStruct timeData1;

struct Time timeData2;

If we don’t care about being able to declare variables using the struct keyword, we can leave off the tag on the first line of the structure declaration.

typedef struct

{

int hours;

int minutes;

int seconds;

} TimeStruct;