Data Structures in C

Table of Contents

1. Objectives

Learn the basics of data structures and pointers in the C programming language.

2. Defining Data Structures Using `struct`

In the C programming language, the `struct` keyword is used to define a complex data type as a group of variables. The resulting data type can then be used to declare variables, each of which would contain all of the listed variables in the structure definition.

A C structure variable references a contiguous block of physical memory.

Defining a Structure and Declaring a Variable

```
struct point {
    int x;
    int y;
};

struct point p;
```

In the example above, `p` is an instance of the `struct point` structure.

3. Structure Aliases Using `typedef`

It is possible to use a shorter name to identify the structure type using the `typedef` keyword. The following example results in a variable `p` that is identical to the `p` variable declared in the previous example.

```
struct point {
    int x;
    int y;
};

typedef struct point Point;

Point p;
```

C is case-sensitive. In the example above, `point` is different from `Point`.

4. Combining `typedef` and `struct`

The `struct` and `typedef` statements can be combined into a single statement.

```
typedef struct point {
    int x;
    int y;
} Point;
```

In fact, when combined, the name immediately following the `struct` keyword, also known as the *structure tag*, can be removed.

```
typedef struct {
    int x;
    int y;
} Point;
```

5. Accessing Fields

In the examples above, the variable `p` is of type `Point`, and thus contains two integer fields, `x` and `y`. To access the fields, the dot operator is used (`.`).

```
p.x = 5;
int z = p.y;
```

6. Pointers to Structures

It is common to refer to structure variables by their address, or pointer, instead of the variable itself. This is especially useful when passing a structure instance as an argument to a function to avoid copying possibly-large variables during the function call.

C passes arguments by value, not by reference. Pointers can be used to pass arguments by reference.

Pointers to structure variables are also useful for declaring another structure instance as a field within the structure.

Pointers to structures are used like any other pointers. The `&` operator retrieves the address of a variable, which can be stored in a pointer variable. The `*` operator is used to declare a pointer variable, and to dereference a pointer in order to access the variable it points to.

```
Point p;               /* an instance variable */
Point *pointer = &p;   /* a pointer to the same instance */
p.x = 5;
(*pointer).x = 5;      /* Identical to the previous statement */
```

Because it is very common to refer to structures using pointers instead of structure variables, a special operator, the arrow (`→`), is available to access a field of a structure using its pointer.

```
p.x = 6;
pointer->x = 6;        /* Identical to the previous statement */
```

7. Unions

A union in C is a data type that stores different data types in the same memory location. There are two main uses of unions:

1. Storing mutually-exclusive data. If you never need to store both variables `a` and `b` at the same time, you can define them using a union so that they use the same memory space. This also applies if you want to declare a generic variable that can have multiple types.
2. Accessing the same data in different ways, or as different data types. For example:

   ```
   union {
       uint32_t x;
       struct {
           uint16_t xL;
           uint16_t xH;
       };
   };
   ```

   Here, `x` refers to a 32-bit integer, whereas `xL` and `xH` refer to the low and high 16 bits of that 32-bit integer, respectively. Changing the value of `x` would also change the values of `xL` and `xH`, depending on which bits have changed.