codecademy

Structures

Defining Structures With `struct`

Structures are defined with the `struct` keyword followed by the structure name. Inside the braces, member variables are declared but not initialized. The given code block defines a structure named `Person` with declared member variables `name` and `age`.

```
// `struct` keyword and structure name
struct Person{
  // uninitialized member variables
  char* name;
  int age;
};
```

Initializing Structures With `struct`

Structure data types are initialized using the `struct` keyword with the defined structure type followed by the name of the variable. The given code block shows two ways to initialize `Person` type structures named `person1` and `person2`.

```
// `Person` structure declaration
struct Person{
  char* name;
  int age;
};

// designated initialization with member
variable names
struct Person person1 = {.name = "Cosmo",
.age = 36};

// implicit initialization following
order of member variables
struct Person person2 = {"George", 29};
```

Custom Data Types With Structures

Structures allow the definition of custom data types that are used to represent complex data. Structure customization provides the flexibility to accurately model real-world data, giving you the ability to access and modify the data from a single defined variable.

Grouping Data Types With Structures

Structures can group different data types together into a single, user-defined type. This differs from arrays which can only group the same data type together into

```
// `Person` structure definition
struct Person{
```

a single type. The given code block defines a structure named `Person` with different basic data types as member variables.

```
  // member variables that would be used
  char* name;
  int age;
  char middleInitial;
};
```

Accessing Member Variables With Dot Notation

Initialized structure member variables can be accessed with the dot (`.`) operator. The given code block initializes a `Person` type named `person1` and accesses the `name` member variable within a `printf()` statement.

```
// `Person` structure declaration
struct Person{
  // member variables
  char* name;
  int age;
  char middleInitial;
};

// initialization of `person1`
struct Person person1 = {.name =
"George", .age = 28, .middleInitial =
"C"};

// accessing `name` in `person1`
printf("My name is %s", person1.name);
// OUTPUT: My name is George
```

Structure Member Variables

The variables defined within a structure are known as member variables. The given code block defined a structure named `Person` with member variables `name` of type `char*` , and `age` of type `int` .

```
// Person structure declaration
struct Person{
  // member variables
  char* name;
  int age;
};
```

Structure Type Pointers

Pointers to a structure can be defined using the `struct` keyword, the structure type, and the pointer (`*`) symbol. The memory address of an initialized structure can be accessed using the symbol (`&`). The given code block defines a pointer to a `Person` data type named `person1` .

```
// Person structure declaration
struct Person{
  // member variables
  char* name;
  int age;
};

// person1 initialization
struct Person person1 = {"George", 28};
```

```
// person1Pointer initializated to the
memory address of person1
struct Person* person1Pointer = &person1;
```

Accessing Member Variables With Arrow Notiation

Member variables of a structure can be accessed using a pointer with arrow (`->`) notation. The given code block initializes a `Person` pointer type named `person1Pointer`. Inside the `printf()` statement, the `name` member variable of `person1` is accessed using arrow (`->`) notation.

```
// `Person` structure declaration
struct Person{
  // member variables
  char* name;
  int age;
};

// `person1` intialization
struct Person person1 = {"Jerry", 29};

// `person1Pointer` intialization to
memory address to `person1`
struct Person* person1Pointer = &person1;

// accessing `name` through
`person1Pointer`
printf("My name is %s", person1Pointer-
>name);
// OUTPUT: My name is Jerry
```

Passing Structures To Functions

Structures can be used as parameters of functions by using the `struct` keyword followed by the structure name in the function definition. The given code block defines a function signature named `myFunc()` with a `Person` parameter named `person1`.

```
// Person structure declaration
struct Person{
  // member variables
  char* name;
  int age;
};

// declaring Person type parameter
void myFunc(struct Person person1);
```

Passing Structure Pointers To Functions

Structure pointers can be paramters of functions by using the `struct` keyword, the structure name, and the pointer symbol (`*`) in the function definition. The given code block defines a function signature named

```
// Person structure declaration
struct Person{
  // member variables
  char* name;
```

`myFunc()` with a `Person` pointer parameter named `person1Pointer`.

Save Print Share

```
  int age;
};

// Person pointer parameter declaration
void myFunc(struct Person*
person1Pointer);
```