Point to Stored Values

Whenever your program creates a container to store data, your computer allocates a space in memory at which to store the data. Like houses, memory locations may be of different size but each one has a unique address.

An initialized variable therefore actually consists of three parts – name, value, and the memory address where the data is stored.

Go programming supports the concept of "pointers". A pointer variable can store the address of another variable and can access the value stored at that address.

The declaration of a pointer variable requires the data type to be prefixed by an * asterisk, to indicate that this will be a pointer:

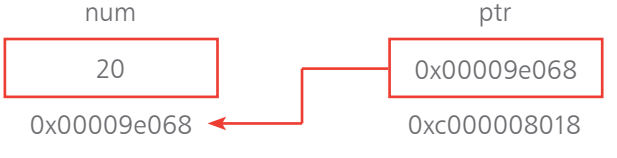
```
var ptr *int        // Declares this variable will point to an int.
```

The address of another variable can be assigned to the pointer by prefixing the other variable's name with an **&** ampersand:

```
ptr = &num          // Assigns an address to the pointer variable.
```

The value at the assigned address can then be accessed by prefixing the pointer name with an * asterisk:

```
*ptr                // Points to the value at the assigned address.
```



The concept of pointers can be difficult to grasp because the * asterisk operator performs two purposes – as a type descriptor in a declaration, and as a dereferencer when placed before a pointer name. You may at this moment be thinking "Pointers, so what?" but pointers are widely used in Go programming, so you should thoroughly understand how they work before proceeding further.

Memory location addresses are hexadecimal numbers that are allocated by your computer. They will almost certainly differ on your system from the addresses shown in this example.

1. Create a directory named "point" inside your "src" folder

src\point\main.go

2. Begin a **main.go** program with package and import declarations

```
package main
import "fmt"
```

3. Add a main function that declares and initializes a regular integer variable and a pointer variable

```
func main( ) {
    var num int = 20
    var ptr *int = &num
    // Statements to be inserted here.
}
```

4. Next, insert statements to display the value and memory address of the regular integer variable

```
fmt.Printf( "num value: %v type: %T \n", num, num )
fmt.Printf( "num address: %v type: %T \n\n", ptr, ptr )
```

5. Now, insert statements to display the dereferenced value and memory address of the pointer variable

```
fmt.Printf( "num via pointer: %v type: %T \n", *ptr, *ptr )
fmt.Printf( "ptr address: %v type: %T \n\n", &ptr, &ptr )
```

6. Finally, insert statements to change the value stored in the integer variable – by assignment to the pointer variable

```
*ptr = 100
fmt.Printf( "new num value: %v type: %T \n", num, num )
```

7. Save the program file in the "point" directory, then run the program to see the variable values and addresses

The most important feature to recognize here is that the pointer changes the original value stored in the variable to which it points, not a copy of that value. This becomes significant when passing variables to functions – see pages 64-67.

```
Go Terminal

C:\Users\mike_\go\src>go run point
num value: 20 type: int
num address: 0xc00009e068 type: *int

num via pointer: 20 type: int
ptr address: 0xc0000c8018 type: **int

new num value: 100 type: int

C:\Users\mike_\go\src>
```