Introduction to Programming in Go

2. Variables and Basic Data Types

Module Topics

1. Built-in or basic data types
2. Variable declarations
3. Variable scope
4. Constants
5. Pointers

Basic Data Types

Basic Data Types

Data Types

Numeric

integer

signed

int8
int16
int32
int64

unsigned

uint8
uint16
uint32
uint64

floats

float32
float64

complex

complex64
complex128

Non-numeric

string
bool

int - either int32 or int64
uint - either uint32 or uint64

rune alias for int32
byte alias for uint8

About Basic Data Types

1. `int` is either `int32` or `int64` depending on architecture.
2. Similarly, `uint` is either `uint32` or `uint64`.
3. Strings are immutable, like in Java.
4. Only explicit conversions are allowed.
5. No mixed mode calculations are allowed.

Variables

Default Initialization to Zero Value

```
// Example 02-01  Declaring Variables - Zero Inits
package main

import "fmt"

var x float32
var c complex64
var b bool
var str string

func main() {
   var message string = "x=%f c=%f b=%t str=|%s| i=%d \n"
   var i int
   fmt.Printf(message,  x, c, b, str, i)
}
```

```
[Module02]$ go run ex02-01.go
x=0.000000 c=(0.000000+0.000000i) b=false str=|| i=0
```

Explicit Initialization

```
// Example 02-02  Initializing Variables
package main

import "fmt"

var x float32 = 32.0
var c complex64 = 5.0 + 3.1i
var b bool = true
var str string = "Hi"

func main() {
   var message string = "x=%f c=%f b=%t str=|%s| i=%d \n"
   var i int = 3
   fmt.Printf(message,  x, c, b, str, i)
}
```

```
[Module02]$ go run ex02-02.go
x=32.000000 c=(5.000000+3.100000i) b=true str=|Hi| i=3
```

Implicit Initialization

```
// Example 02-03  Implicit Initialization
package main

import "fmt"

var x   = 32.0
var c   = 5.0 + 3.1i
var b   = true
var str = "Hi"

func main() {
   var message  = "x=%T c=%T b=%T str=%T i=%T \n"
   var i  = 3
   fmt.Printf(message,  x, c, b, str, i)
}
```

```
[Module02]$ go run ex02-03.go
x=float64 c=complex128 b=bool str=string i=int
```

Implicit Initialization

```go
// Example 02-04  Implicit Initialization
package main

import "fmt"

var x   = float32(0)
var c   = complex64(0)
var b   = true
var str = "Hi"

func main() {
   var message  = "x=%T c=%T b=%T str=%T i=%T \n"
   var i  = uint8(0)
   fmt.Printf(message,  x, c, b, str, i)
}
```

```
[Module02]$ go run ex02-04.go
x=float32 c=complex64 b=bool str=string i=uint8
```

Variations on a Theme

```
// Example 02-05  Variations
package main

import "fmt"

var i, j int8
var b, str, x = true, "Hi", float32(45)


func main() {
   fmt.Printf( "i=%T j=%T b=%T str=%T x=%T \n",
       i, j, b, str, x)
}
```

```
[Module02]$ go run ex02-05.go
i=int8 j=int8 b=bool str=string x=float32
```

Declaration Short Form

```go
// Example 02-06  Short Form

package main

import  "fmt"

var (
   k int = 1
   m int
)

func main() {
   i, j, k := 3, 4, 5
   x := 1.2
   fmt.Println("i=%d j=%d x=%f k=%d \n", i, j, x, k)
}
```

```
[Module02]$ go run ex02-06.go
i=3 j=4 x=1.200000 k=5
```

Variable Scope

Variable Scope - Package Scope

```
// Example 02-07  Variable Scope

package main

import  "fmt"

var k int = 1

func main() {
   i := 1
   {
       x := 1.2
       fmt.Println("i=%d d x=%f k=%d \n", i, x, k)
   }
}
```

k has Package Scope

```
[Module02]$ go run ex02-07.go
i=1 x=1.200000 k=1
```

Variable Scope - Local Function Scope

```
// Example 02-07  Variable Scope

package main

import  "fmt"

var k int = 1

func main() {
   i := 1
   {
       x := 1.2
       fmt.Println("i=%d d x=%f k=%d \n", i, x, k)
   }
}
```

i has Function Scope

```
[Module02]$ go run ex02-07.go
i=1 x=1.200000 k=1
```

Variable Scope - Local Block Scope

```
// Example 02-07  Variable Scope

package main

import  "fmt"

var k int = 1

func main() {
   i := 1
   {
       x := 1.2
       fmt.Println("i=%d d x=%f k=%d \n", i, x, k)
   }
}
```

x has Block Scope

```
[Module02]$ go run ex02-07.go
i=1 x=1.200000 k=1
```

Variable Scope - Shadowing

```
// Example 02-08  Shadowing

...

var k int = 1

func main() {
        fmt.Println("Package Scope k=%d \n", k)
        k := 2
        fmt.Println("Function Scope k=%d \n", k)
            {
            k := 3
            fmt.Println("Block Scope k=%d \n", k)
            }
        fmt.Println("Function Scope k=%d \n", k)
}
```

```
[Module02]$ go run ex02-08.go
Package Scope k=1
Function Scope k=2
Block Scope k=3
Function Scope k=2
```

Constants

Constants

```
// Example 02-09  Constants
...
const a float32 = 1
const b = 4 / 3

func main() {
   const c string = "Hello Word"
   fmt.Printf("a=%T b=%T c=%T\n", a, b, c)
   fmt.Printf("a=%f b=%d c=%s\n", a, b, c)
}
```

```
[Module02]$ go run ex02-09.go
a=float32 b=int c=string
a=1.000000 b=1 c=Hello Word
```

Enums with iota

1. iota is a built in enumerator for building sets of enumerations.
2. iota starts at 0 and increments by 1 for each constant.
3. An expression to generate iota can be used instead of 0.
4. iota is reset to 0 each time we use it with a new set of constants.
5. Any constant initialized with iota is of type int.
6. The blank _ symbol can be used to make iota start at 1.

Enums with iota

```
// Example 02-10 Enums with iota
...
const (
    a = iota
    b
    c
)

func main() {
   fmt.Printf("a=%T b=%T c=%T\n", a, b, c)
   fmt.Printf("a=%f b=%d c=%s\n", a, b, c)
}
```

```
[Module02]$ go run ex02-10.go
a=int b=int c=int
a=0 b=1 c=2
```

Enums with iota - Blank Variable

```
// Example 02-11 Enums with iota and _
...
const (
    _ = iota
    a
    b
    c
)

func main() {
   fmt.Printf("a=%T b=%T c=%T\n", a, b, c)
   fmt.Printf("a=%f b=%d c=%s\n", a, b, c)
}
```

```
[Module02]$ go run ex02-11.go
a=int b=int c=int
a=1 b=2 c=3
```

Enums with iota - Generator Expression

```
// Example 02-12 Enums with Generator
...
const (
    a = iota * 2
    b
    c
)

func main() {
   fmt.Printf("a=%T b=%T c=%T\n", a, b, c)
   fmt.Printf("a=%d b=%d c=%d\n", a, b, c)
}
```

```
[Module02]$ go run ex02-12.go
a=int b=int c=int
a=0 b=2 c=4
```

Pointers

Pointers

1. Pointers work just like in C or C++.
2. Pointer to an int would be declared as "var p *int".
3. Address-of operator is "&".
4 p := &i would assign a pointer p the address of i.
5. De-referencing operator is "*".
6. *p gives the contents of the location pointed to by p.
7. Pointer assignments and comparisons "==" are allowed.
8. Pointer arithmetic is not allowed.

Pointers

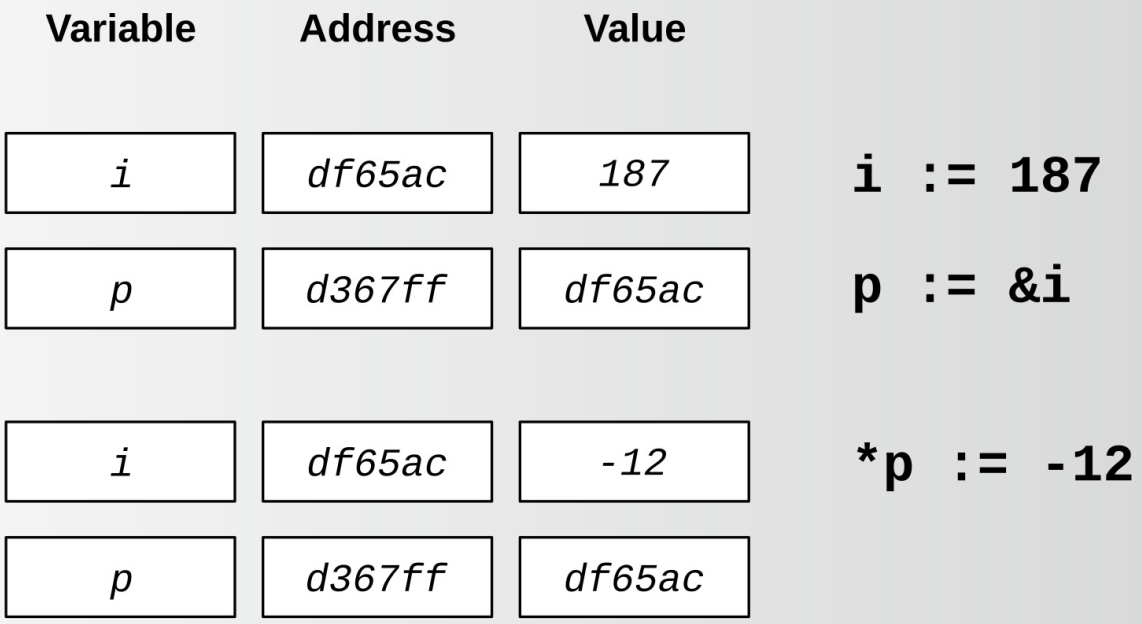
```
// Example 02-13 Pointers
...

func main() {
   var i int = 187
   var p *int

   p = &i
   fmt.Println("i=", i, " &i or p =", p, " *p =", *p)
   *p = -12
   fmt.Println("i=", i, " &i or p =", p, " *p =",
*p)
}
```

```
[Module02]$ go run ex02-13.go
i= 187  &i or p = 0xc82000a398  *p = 187
i= -12  &i or p = 0xc82000a398  *p = -12
```

Pointers


Variable
Address
Value
i
df65ac
187
i := 187
p
d367ff
df65ac
p := &i
i
df65ac
-12
*p := -12
p
d367ff
df65ac

Lab 2: Variables