Go Data Types

Introduction

- Go is a statically typed language.
- Data types define the kind of values a variable can hold.
- Go has several built-in data types to work with.

Data Types

1. **Numeric Types:**
    - `int`: Signed integer type (size depends on the architecture).
    - `int8`, `int16`, `int32`, `int64`: Signed integers with specific sizes.
    - `uint8`, `uint16`, `uint32`, `uint64`: Unsigned integers with specific sizes.
    - `float32`: IEEE-754 32-bit floating-point.
    - `float64`: IEEE-754 64-bit floating-point.
    - `complex64`: Complex number with float32 real and imaginary parts.
    - `complex128`: Complex number with float64 real and imaginary parts.

Data Types

2. **Boolean Type:**
    - `bool`: Represents true or false values.

Data Types

3. **String Type:**
    - `string`: A sequence of Unicode characters.

Data Types

4. **Composite Types:**
    - **Arrays:** Fixed-size sequence of elements of the same type. `[size]type`
    - **Slices:** Dynamic arrays with a flexible size. `[]type`
    - **Maps:** Key-value pairs (associative arrays). `map[keyType]valueType`
    - **Structs:** Composite data type that groups fields with different data types.

Data Types

5. **Pointer Types:**
    - `*type`: A pointer to a value of the specified type.

Data Types

6. **Function Types:**
    - `func`: Represents a function type.

Data Types

7. **Interface Types:**
    - `interface`: Defines a set of methods that a type must implement to satisfy the interface.

Data Types

8. **Channel Types:**
    - `chan`: Used for communication between goroutines.
9. **Custom Types:**
    - You can define your custom data types using the `type` keyword.

Data Types

10. **Byte Types:**
    - `byte`: Alias for `uint8`. Often used to represent single bytes of data.
    - `rune`: Alias for `int32`. Used to represent Unicode code points.

Data Types

11. **Error Type:**
    - `error`: Represents errors in Go programs. Commonly used in error handling.

Data Types

These data types cover a wide range of use cases in Go programming. Understanding when and how to use each type is crucial for effective Go development.

Integer Types

- `int` (depends on the architecture)
- `int8`, `int16`, `int32`, `int64`
- `uint8`, `uint16`, `uint32`, `uint64`

```
// Go
var num1 int = 42

// C++
int num2 = 42;
```

Floating-Point Types

- `float32` (float)
- `float64` (double)

Example (Go vs. C++):

```
// Go
var num1 float64 = 3.14

// C++
double num2 = 3.14;
```

Numeric Data Types (cont.)

Complex Types

- `complex64`
- `complex128`

Example (Go vs. C++):

```
// Go
var comp1 complex128 = 1 + 2i

// C++
std::complex<double> comp2(1, 2);
```

Boolean Data Type

- `bool`: Represents true or false values.

Example (Go vs. C++):

```
// Go
var isTrue bool = true

// C++
bool isTrue = true;
```

String Data Type

- `string`: A sequence of Unicode characters.
- Immutable: You can't change individual characters.

Example (Go vs. C++):

```
// Go
var str1 string = "Hello, Go!"

// C++
std::string str2 = "Hello, C++!";
```

Composite Data Types

Arrays

- Fixed-size sequence of elements of the same type.
- `[size]type`

Example (Go vs. C++):

```
// Go
var arr1 [3]int = [3]int{1, 2, 3}

// C++
int arr2[3] = {1, 2, 3};
```

Composite Data Types (cont.)

Slices

- Dynamic arrays with a flexible size.
- Built on top of arrays.
- `[]type`

Example (Go vs. C++):

```
// Go
slice1 := []int{1, 2, 3}

// C++ (Using vectors)
std::vector<int> vec = {1, 2, 3};
```

Composite Data Types (cont.)

Maps

- Key-value pairs (associative arrays).
- `map[keyType]valueType`

Example (Go vs. C++):

```
// Go
m := make(map[string]int)
m["apple"] = 3

// C++
std::map<std::string, int> m;
m["apple"] = 3;
```

Composite Data Types (cont.)

Structs

- Composite data type that groups fields.
- Fields can have different data types.

Example (Go vs. C++):

```
// Go
type Person struct {
    Name string
    Age  int
}

// C++
struct Person {
    std::string name;
    int age;
};
```

Special Data Types

Pointers

- Store memory addresses.
- Used to indirectly access variables.

Example (Go vs. C++):

```
// Go
var num int = 42
var ptr *int = &num

// C++
int num = 42;
int* ptr = &num;
```

Special Data Types (cont.)

Functions

- Functions are first-class citizens in Go.
- Can be assigned to variables, passed as arguments, and returned from other functions.

Example (Go vs. C++):

```
// Go
func add(a, b int) int {
    return a + b
}

// C++
int add(int a, int b) {
    return a + b;
}
```

Summary

- Go offers a range of built-in data types.
- Understanding the differences between numeric, boolean, string, and composite types is crucial.
- Pointers and functions are special data types.
- Go and C++ have similarities and differences in their type declarations and usage.

Questions

Any questions on Go data types or comparisons with C++?