Go Programming Language

Data Types

Department of Computer Science and Engineering

Basic data types - integers

Integers

- Go provides both signed and unsigned integers.
- There are four distinct sizes of signed integers - 8, 16, 32, and 64 bits - represented by the types int8, int16, int32, and int64, and corresponding unsigned versions uint8, uint16, uint32, and uint64.
- Unsigned integer type **uintptr**, whose width is not specified but is sufficient to hold all the bits of a pointer value. The uintptr type is used only for low-level programming
- Regardless of their size, int, uint, and uintptr are different types from their explicitly sized siblings. Thus int is not the same type as int32, even if the natural size of integers is 32 bits, and an explicit conversion is required to use an int value where an int32 is needed, and vice versa.
- Float to integer conversion discards any fractional part, truncating toward zero.

Go Programming Language

Operations on integers

- Binary operators for arithmetic, logic, and comparison in order of decreasing precedence

```
*    /    %    <<    >>    &    &^
+    -    |    ^
==   !=   <    <=    >     >=
&&
||
```

- Comparison operators

==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

- Bitwise binary operators

&	bitwise AND
|	bitwise OR
^	bitwise XOR
&^	bit clear (AND NOT)
<<	left shift
>>	right shift

Basic data types – floating point numbers

Floating-point numbers

- Go provides two sizes of floating-point numbers, float32 and float64.
- A float32 provides approximately six decimal digits of precision, whereas a float64 provides about 15 digits
- Very small or very large numbers are written in scientific notation, with the letter e or E preceding the decimal exponent: **const Avogadro = 6.02214129e23**

Complex numbers

- Go provides two sizes of complex numbers, complex64 and complex128, whose components are float32 and float64 respectively.
- The built-in function complex creates a complex number from its real and imaginary components, and the built-in real and imag functions extract those components

Basic data types – boolean

Boolean

- A value of type bool, or boolean, has only two possible values, true and false.
- The conditions in if and for statements are booleans
- Comparison operators like == and < produce a boolean result.
- Boolean values can be combined with the && (AND) and || (OR) operators
- There is no implicit conversion from a boolean value to a numeric value like 0 or 1, or vice versa.

Basic data types – strings

Strings

- A string is an immutable sequence of bytes.
- Strings may contain arbitrary data, including bytes with value 0, but usually they contain human-readable text.
- Text strings are conventionally interpreted as UTF-8 encoded sequences
- The built-in **len** function returns the number of bytes in a string, and the index operation s[i] retrieves the i-th byte of string s, where 0 <= i < len(s)
- The substring operation s[i:j] yields a new string consisting of the bytes of the original string starting at index i and continuing up to, but not including, the byte at index j.
- The + operator makes a new string by concatenating two strings

Basic data types – strings

Conversion between strings and numbers

- To convert an integer to a string, use the function strconv.Itoa ()
- To convert a string to an integer, use the function strconv.Atoi ()
- strconv.ParseFloat converts the string s to a floating-point number with the precision specified by bitSize: 32 for float32, or 64 for float64

Complex data types – arrays

Arrays

- An array is a fixed-length sequence of zero or more elements of a particular type.
- Because of their fixed length, arrays are rarely used directly in Go.
- Slices, which can grow and shrink, are much more versatile
- Individual array elements are accessed with the conventional subscript notation, where subscripts run from zero to one less than the array length.
- The built-in function **len** returns the number of elements in the array.
- r := [...]int{99: -1} defines an array r with 100 elements, all zero except for the last, which has value −1.
- Two arrays can be directly compared using the == operator, which reports whether all the corresponding elements are equal.

Complex data types – slices

Slices

- Slices represent variable-length sequences whose elements all have the same type.
- A slice type is written []T, where the elements have type T; it looks like an array type without a size.
- A slice is a lightweight data structure that gives access to a subsequence (or perhaps all) of the elements of an array, which is known as the slice's underlying array.
- A slice has three components: a pointer, a length, and a capacity.
- The pointer points to the first element of the array that is reachable through the slice, which is not necessarily the array's first element.
- The length is the number of slice elements; it can't exceed the capacity, which is usually the number of elements bet ween the start of the slice and the end of the underlying array.
- The built-in functions len and cap return length and capacity values.

Complex data types – slices

Slices

- Multiple slices can share the same underlying array and may refer to overlapping parts of that array.
- Since a slice contains a pointer to an element of an array, passing a slice to a function permits the function to modify the underlying array elements
- Unlike arrays, slices are not comparable, so we cannot use == to test whether two slices contain the same elements.
- The standard library provides the highly optimized **bytes.Equal** function for comparing two slices of bytes
- The built-in append function appends items to slices
- The zero value of a slice type is nil. A nil slice has no underlying array. The nil slice has length and capacity zero

Complex data types – maps

Maps

- Hash table is one of the most ingenious and versatile of all data structures
- In Go, a map is a reference to a hash table, and a map type is written map[K]V, where K and V are the types of its keys and values.
- All of the keys in a given map are of the same type, and all of the values are of the same type, but the keys need not be of the same type as the values.
- Map elements are accessed through the usual subscript notation

Complex data types – Struct

Struct

- A struct is an aggregate data type that groups together zero or more named values of arbitrary types as a single entity. Each value is called a field.
- The classic example of a struct from data processing is the employee record, whose fields are a unique ID, the employee's name, address, date of birth, position, salary, manager, and the like.
- All of these fields are collected into a single entity that can be copied as a unit, passed to functions and returned by them, stored in arrays.
- The individual fields of a struct are accessed using the dot notation
- If all the fields of a struct are comparable, the struct its elf is comparable, so two expressions of that type may be compared using == or !=.

JSON

- Java Script Object Notation (JSON) is a standard notation for sending and receiving structured Information
- Go has excellent support for encoding and decoding these formats, provided by the standard library packages encoding/json
- Converting a Go data structure to JSON is called marshaling. Marshaling is done by json.Marshal

THANK YOU

Suresh Jamadagni

Department of Computer Science and Engineering