Functional Programming in C++ and Go

Agenda

- The Programming Language C++
- Lambda Function Objects
- Lambda Calculus
- Functional Composition
- High-Order Functions
  - Map
  - Fold / Reduce

The Programming Language C++

- 1979 developed by Bjarne Stroustrup as “C with Classes“
  - Main Goal: integrate object oriented concepts into C
  - Classes, Inheritance, etc.
- 1983 renamed to C++
- 1985 first published
- 1998 first standardized by ISO / IEC

- 2011 Lambda Function Objects and “auto“ keyword added

Lambda Function Objects

- Added 2011 in C++11 Standard
- Anonymous functions objects
- Can be used as functions
- Can be used as arguments or values

- auto f = [](){}

- []: usage of surrounding values; copy, reference, none
  - also: initialize, only certain values, all from class, etc.
- (): function arguments
- {}: functional body

Lambda Calculus

Golang:

```
type fnf func(fnf) fnf

True := func(x fnf) fnf { return func(y fnf) fnf { return x } }
False := func(x fnf) fnf { return func(y fnf) fnf { return y } }
Not := func(x fnf) fnf { return x(False)(True) }
```

C++:

```
auto True = [](auto x) { return [=](auto y) { return x; }; };
auto False = [](auto x) { return [=](auto y) { return y; }; };
auto Not = [](auto x) { return x(False)(True); };
```

Call:
Not(False)(f)(g)(ID)

Functional Composition

- Compose two functions into one function
  - Using the return values
  - Using the functions as arguments

Functional Composition - Golang

```
square := func(x any) any { return x.(int) * x.(int) }
plus := func(x int) int { return x + 1 }

// two functions composed to one function
gf := func(x int) int { return plus(square(x)) }

type any interface{}
type function func(any) any

compose := func(g, f function) function { return func(a any) any { return g(f(x)) } }

fmt.Printf("%v\n", gf(2)) // --> 5
fmt.Printf("%v\n", compose(square, square)(2)) // --> 16
fmt.Printf("%v\n", compose(compose(square, square), square)(2))
```

Functional Composition - C++

```
// functions to be composed together
auto square(int x){ return x*x; }
auto plus(int x){ return x+1; }

// two functions composed to one function
auto gf(int x){ return plus(square(x)); }

// lambda function object for composing two functions f and g
auto compose = [](auto f, auto g) { return [=](auto x) { return f(g(x)); }; };

std::cout << gf(2) << "\n"; // --> 5
std::cout << compose(square, square)(2) << "\n"; // --> 16
std::cout << compose(compose(square, square), square)(2) << "\n"; // --> 256
```

High-Order Functions

- Functions that take other functions as arguments
- Two most known function families:
- Map
  - takes function and list of data as arguments
  - maps function argument on every element of a list of data
  - can also be used as filter
- Fold / Reduce
  - takes function and list of data as arguments
  - reduces data into a single return value, depending on function argument

High-Order Functions – Map in Golang

- In strings

```
func Map(mapping func(rune) rune, s string) strings
s := "Hello, world!"
s = strings.Map(func(r rune) rune { return r + 1 }, s)

fmt.Println(s) // --> Ifmmp-!xpsme"
```

High-Order Functions – Map in C++

- In algorithm header

```
std::string v = "Hello, world!";
std::transform(v.begin(), v.end(), v.begin(), [](int x) { return x + 1; } );

std::cout << v << "\n"; // --> Ifmmp-!xpsme"
```

High-Order Functions – Fold in Golang

- needs to be self implemented

```
func (s *SliceStream) Reduce(accumulate Accumulator) Any {
	var result interface{}
	for i, e := range s.data {
		if i == 0 {
			result = e
		} else {
			result = accumulate(result, s.data[i])
		}
	}
return result
}
```

High-Order Functions – Fold in C++

- in numeric header

```
std::vector<int> v{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
std::vector<std::string> s {"a", "b", "asdf", "Hallo", "World"};

//sum ist 55
auto sum = std::accumulate(v.begin(), v.end(), 0);

// s2 = "abasdfHalloWelt"
auto s2 = std::accumulate(s.begin(), s.end(), std::string(""));

//s3 = "12345678910"
auto s3 = std::accumulate(z.begin(), z.end(), std::string(""),
    [](std::string ret, int i) { return ret + std::to_string(i); }
);
```

Any Question?

Sources

- [HoC] 'History of C\++'. http://www.cplusplus.com/info/history/. Last visited: 27.12.2018
- [FP11] Grimm Rainer. 'Functional Programming in C++ 11'. science + computing ag. https://www.grimm-jaud.de/images/stories/pdfs/FunctionalProgrammingInC++11.pdf. Last visited: 31.12.2018
- [FPGN] Gigante Nicola. 'Functional Programming in C++ - Nicola Gigante - Meeting C++ 2015'. https://www.youtube.com/watch?v=SCC23W3CQc8. Last visited: 31.12.2018
- [CoPL] Weigend Johannes. 'Master Course: Concepts of Programming Languages - University of Applied Sciences Rosenheim (TH Rosenheim)'. https://github.com/jweigend/concepts-ofprogramming-languages. Last visited: 27.12.2018
- [HFP] 'Functional programming'. https://wiki.haskell.org/Functional_programming. Last visited 03.01.2019
- [Go] 'The Go Programming Language'. https://golang.org. last visited 03.01.2019