Execution mode in Golang 1.8

Umbo CV - David Chou

We are Umbo CV

We build smart cloud surveillance

Golang is a nice language, but...

Huge binary size

- 1.5 MB for HelloWorld
- Static link with everything
- Not friendly for embedded system

```go
package main

import "fmt"

func main() {
	fmt.Println("Hello World")
}
```

Build library with Go

- We could invoke C-API through cgo
- Could we write library in Go but run from other languages ?

```
package print

// #include <stdio.h>
// #include <stdlib.h>
import "C"
import "unsafe"

func Print(s string) {
    cs := C.CString(s)
    C.fputs(cs, (*C.FILE)(C.stdout))
    C.free(unsafe.Pointer(cs))
}
```

Golang Execution Mode

- -buildmode=exe
- -buildmode=archive
- -buildmode=shared
- -buildmode=c-archive
- -buildmode=c-shared
- -buildmode=pie
- -buildmode=plugin

buildmode=exe

- Default build mode for ***main*** package
- Build main and everything that imported into executable
- Support for all targets

buildmode=archive

- Default build mode for ***non-main*** package
- Build that package into a .a file
- Support for all targets

buildmode=shared

- Combine all the listed packages into a single shared library
- That shared library will be used when building with the ***-linkshared*** option
- Dynamic linking: All shared libraries loaded when process starts
- Currently only support for Linux

Reduce binary size with shared mode

```go
package calc

import "fmt"

func SayHello(name string) {
    fmt.Printf("Go says: Hello, %s!\n", name)
}

func Add(num0, num1 int) int {
    return num0 + num1
}
```

```go
package main

import (
    "fmt"
    "github.com/david7482/go-plugin-playground/calc"
)

func main()  {
    fmt.Print("This is a Go Application.\n")
    calc.SayHello("World")
    fmt.Printf("Add(3, 5): %d\n", calc.Add(3, 5))
}
```

Original binary size

```
# go build main.go
# ./main
This is a Go Application.
Go says: Hello, World!
Add(3, 5): 8

# ls main -lh
-rwxr-xr-x 1 root root 1.5M Feb 15 17:25 main
```

Build in shared mode

```
# go install -buildmode=shared std
# go install -buildmode=shared -linkshared github.com/david7482/go-plugin/calc
# go build -buildmode=shared -linkshared main.go
# ls -lh main
-rwxr-xr-x 1 root root 20K Feb 15 17:43 main
```

```
# ldd main
	linux-vdso.so.1 (0x00007ffea0bc5000)
	libstd.so => /usr/local/go/pkg/linux_amd64_dynlink/libstd.so (0x00007fc62fab6000)
	libgithub.com-david7482-go-plugin-calc.so =>
/go/pkg/linux_amd64_dynlink/libgithub.com-david7482-go-plugin-calc.so (0x00007fc62f8b1000)
	libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc62f502000)
	libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fc62f2fe000)
	libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fc62f0e0000)
	/lib64/ld-linux-x86-64.so.2 (0x000055cc668b3000)


# ls -gGh /usr/local/go/pkg/linux_amd64_dynlink/libstd.so
-rw-r--r-- 1 38M Feb 15 11:49 /usr/local/go/pkg/linux_amd64_dynlink/libstd.so
# ls -gGh /go/pkg/linux_amd64_dynlink/libgithub.com-david7482-go-plugin-calc.so
-rw-r--r-- 1 21K Feb 15 17:39 /go/pkg/linux_amd64_dynlink/libgithub.com-david7482-go-plugin-calc.so
```

Binary size comparision

default: 1.5MB

shared: libstd.so(38MB)
libcalc.so(21KB)
main(20KB)

buildmode=c-archive

- Build the main package, plus all imported packages, into a single C archive file
- Requires main package, but the main function is ignored
- Need to mark callable symbol as ***exported***
- Support for Linux, macOS and Windows

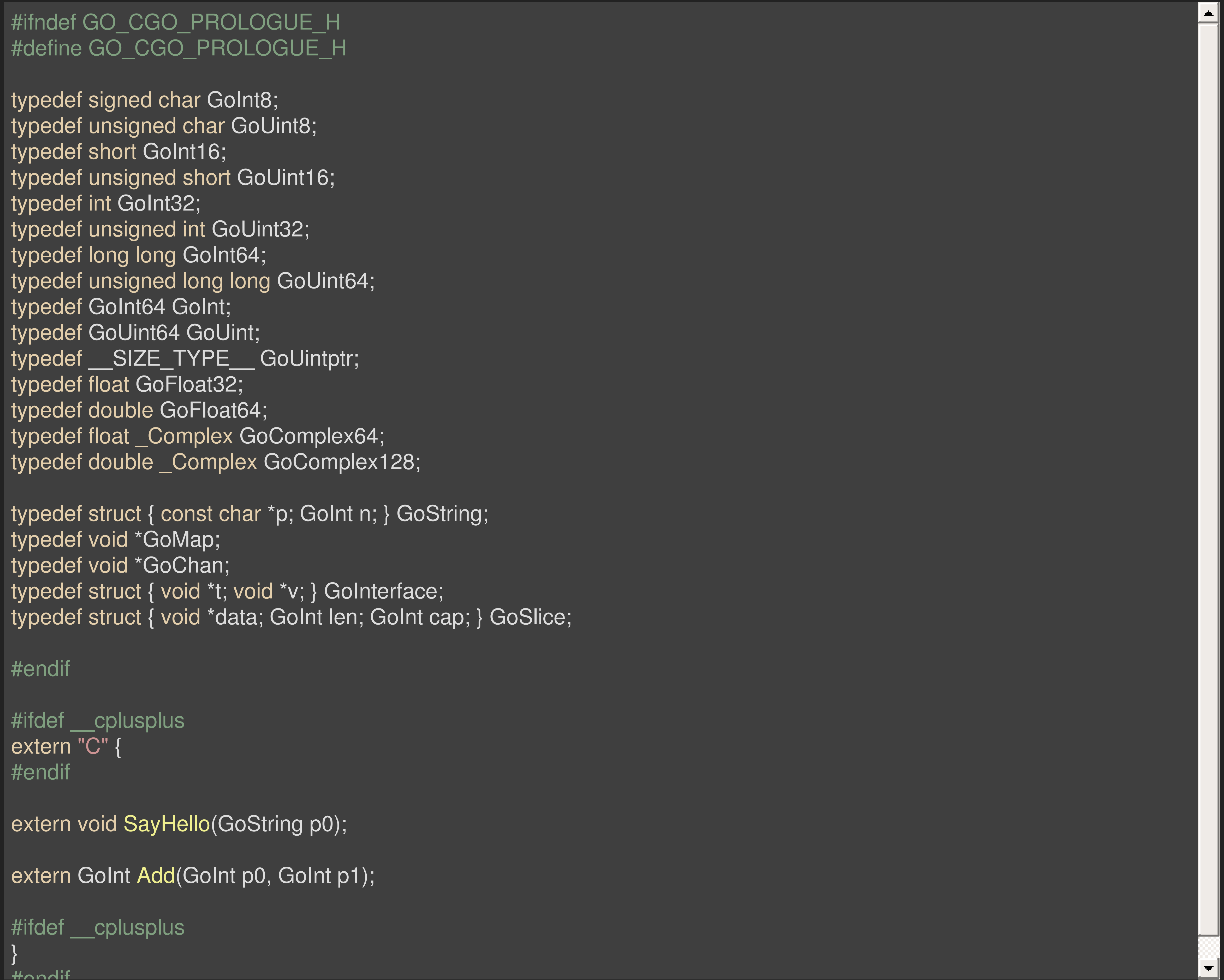
buildmode=c-shared

- Like c-archive, but build into a single C shared file
- Support for Linux, macOS and ~~Windows~~

Same example but from C++

```go
package main

import "C"

func main() {}

//export SayHello
func SayHello(name string) {
	fmt.Printf("Go says: Hello, %s!\n", name)
}

//export Add
func Add(num0, num1 int) int {
	return num0 + num1
}
```

```
# go build -buildmode=c-shared -o calc.so calc.go
# go build -buildmode=c-archive -o calc.a calc.go
# ls calc.*
calc.a  calc.go  calc.h  calc.so
```



```c
#ifndef GO_CGO_PROLOGUE_H
#define GO_CGO_PROLOGUE_H

typedef signed char GoInt8;
typedef unsigned char GoUint8;
typedef short GoInt16;
typedef unsigned short GoUint16;
typedef int GoInt32;
typedef unsigned int GoUint32;
typedef long long GoInt64;
typedef unsigned long long GoUint64;
typedef GoInt64 GoInt;
typedef GoUint64 GoUint;
typedef __SIZE_TYPE__ GoUintptr;
typedef float GoFloat32;
typedef double GoFloat64;
typedef float _Complex GoComplex64;
typedef double _Complex GoComplex128;

typedef struct { const char *p; GoInt n; } GoString;
typedef void *GoMap;
typedef void *GoChan;
typedef struct { void *t; void *v; } GoInterface;
typedef struct { void *data; GoInt len; GoInt cap; } GoSlice;

#endif

#ifdef __cplusplus
extern "C" {
#endif

extern void SayHello(GoString p0);

extern GoInt Add(GoInt p0, GoInt p1);

#ifdef __cplusplus
}
#endif
```

```c
#include "calc.h"
#include <stdio.h>
#include <string.h>

int main() {
    printf("This is a C Application.\n");

    GoString name;
    name.p = "World";
    name.n = strlen(name.p);
    SayHello(name);

    printf("Add(3, 5): %d\n", Add(3, 5));
    return 0;
}
```

```
# gcc -o main main.cpp calc.a -lpthread
# gcc -o main main.cpp calc.so -lpthread
# ./main
This is a C Application.
Go says: Hello, World!
Add(3, 5): 8


# readelf -d main | grep NEEDED
 0x0000000000000001 (NEEDED)          Shared library: [calc.so]
 0x0000000000000001 (NEEDED)          Shared library: [libc.so.6]


# readelf -d calc.so | grep NEEDED
 0x0000000000000001 (NEEDED)          Shared library: [libpthread.so.0]
 0x0000000000000001 (NEEDED)          Shared library: [libc.so.6]
```

buildmode=pie

- Like exe mode, but build into a Position Independent Executable
- Avoid security attack relys on knowing the offset of executable code
- Only support for Linux

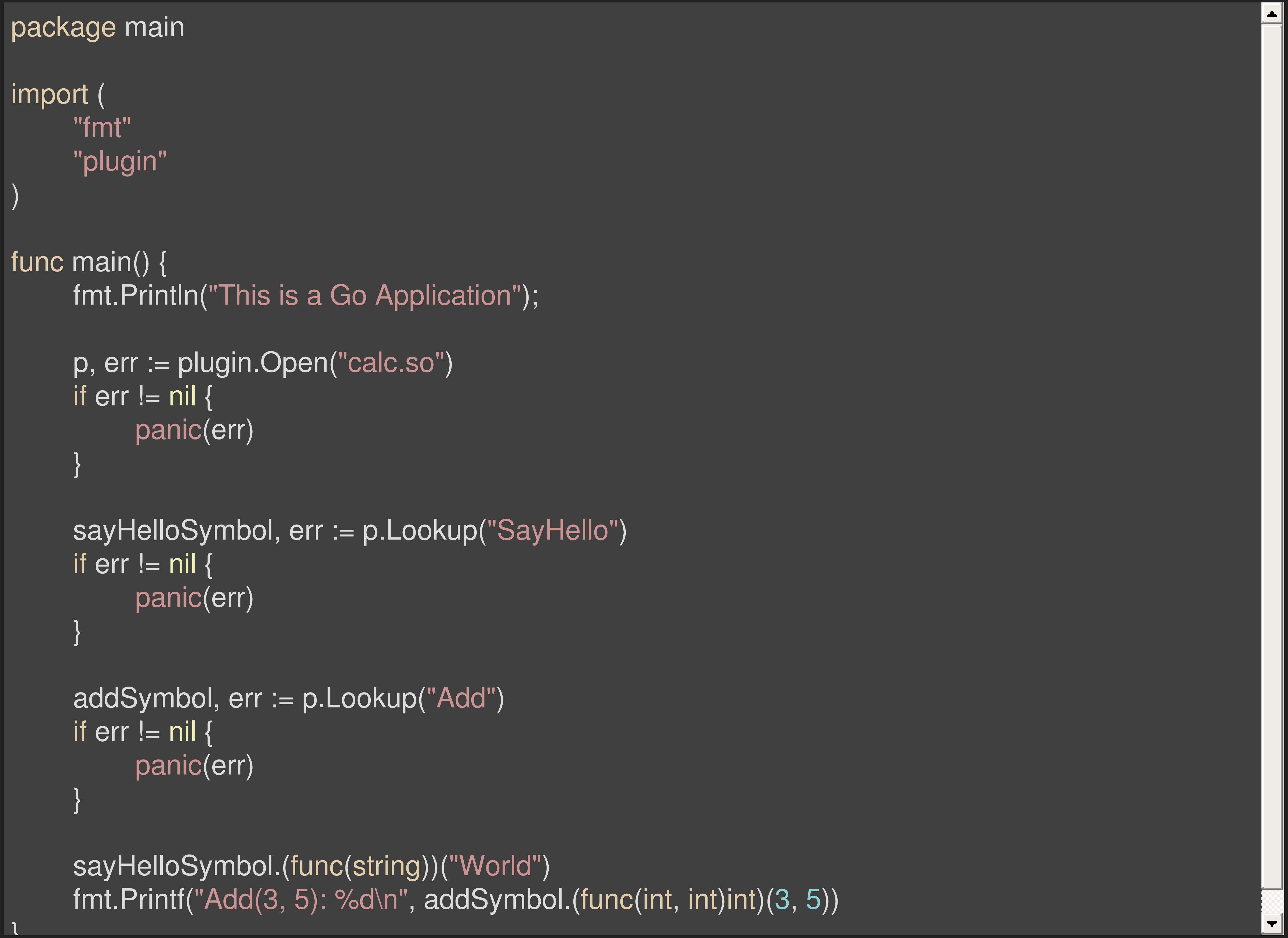
buildmode=plugin

- Build the main package, plus all imported packages, into a single shared library
- Dynamic loading: plugin.Open(), plugin.Lookup()
- Currently only support for Linux
- **Support from Go 1.8**

Same example but in plugin mode

```go
package main

import "C"

func SayHello(name string) {
	fmt.Printf("Go says: Hello, %s!\n", name)
}

func Add(num0, num1 int) int {
	return num0 + num1
}
```

```
# go build -buildmode=plugin -o calc.so calc.go
# ls calc.*
calc.go  calc.so
```



```go
package main

import (
	"fmt"
	"plugin"
)

func main() {
	fmt.Println("This is a Go Application");

	p, err := plugin.Open("calc.so")
	if err != nil {
		panic(err)
	}

	sayHelloSymbol, err := p.Lookup("SayHello")
	if err != nil {
		panic(err)
	}

	addSymbol, err := p.Lookup("Add")
	if err != nil {
		panic(err)
	}

	sayHelloSymbol.(func(string))("World")
	fmt.Printf("Add(3, 5): %d\n", addSymbol.(func(int, int)int)(3, 5))
}
```

```
# go build main.go
# ./main
This is a Go Application
Go says: Hello, World!
Add(3, 5): 8

# readelf -d main | grep NEEDED
 0x0000000000000001 (NEEDED)             Shared library: [libdl.so.2]
 0x0000000000000001 (NEEDED)             Shared library: [libpthread.so.0]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
```

How to use Plugin mode

- Load the plugin
- Lookup a symbol by name
- Type-assert the symbol to the type that you expect
- Use the loaded code

Caveats

You cannot dlclose() a Go c-shared library

#11100

There is no plugin.Close()

reddit thread

All Go code must be built with the same version of Go toolchain

Questions ?