Why go?

О себе:

- Фанатик технологий
- Есть опыт в embedded и backend разработке

I use Arch btw

План:

- Плюсы
- Ни туда ни сюда
- Минусы

Плюсы:

> Инициализация проекта

```
// Для простых локальных проектов
go mod init test

// Для production ready проектов
go mod init github.com/HardDie/cool_project
```

> Компиляция

```c
#include <stdio.h>

int main() {
    printf("Hello world!\n");
}
```

```go
package main

import "fmt"

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

```
gcc -o bin main.c
./bin
```

```
go run main.go
```

```
ifndef V
    QUIET_CC       = @echo '   ' CC '       ' $@;
    QUIET_BUILT_IN = @echo '   ' BUILTIN '  ' $@;
    QUIET_CLEAN    = @echo '   ' CLEAN '    ' $<;
endif

CC = cc
CFLAGS = -std=c99
RM = rm -f

lib = -lncurses
path = bin
obj = \
        $(path)/main.o   \
        $(path)/game.o   \
        $(path)/global.o \
        $(path)/draw.o   \
        $(path)/net.o
all : check_path $(path)/BattleShip

$(path)/BattleShip : $(obj)
    $(QUIET_BUILT_IN)$(CC) $(CFLAGS) $(obj) -o $(path)/BattleShip $(lib)
$(path)/%.o : %.c
    $(QUIET_CC)$(CC) $(CFLAGS) -c $< -o $@

clean : $(path)
    $(QUIET_CLEAN)$(RM) -r $<

check_path :
    @ if [ ! -d $(path) ]; then mkdir $(path); fi
```

```
// Если все файлы в текущей директории
go run .

// Если проект лежит по разным пакетам
go run cmd/main.go
```

> Кросскомпиляция

C

- Найти компилятор под другую платформу
- Возможно еще и скомпилировать его придется
- Только после этого собрать приложение
- А еще собрать все зависимости под платформу

go

- GOOS=windows GOARCH=amd64 go build -o bin main.go
- GOOS=darwing GOARCH=arm64 go build -o bin main.go

{"aix", "ppc64"},
{"android", "386"}, {"android", "amd64"}, {"android", "arm"}, {"android", "arm64"},
{"darwin", "amd64"}, {"darwin", "arm64"},
{"dragonfly", "amd64"},
{"freebsd", "386"}, {"freebsd", "amd64"}, {"freebsd", "arm"}, {"freebsd", "arm64"}, {"freebsd", "riscv64"},
{"illumos", "amd64"},
{"ios", "amd64"}, {"ios", "arm64"},
{"js", "wasm"},
{"linux", "386"}, {"linux", "amd64"}, {"linux", "arm"}, {"linux", "arm64"}, {"linux", "loong64"},
{"linux", "mips"}, {"linux", "mips64"}, {"linux", "mips64le"}, {"linux", "mipsle"},
{"linux", "ppc64"}, {"linux", "ppc64le"}, {"linux", "riscv64"}, {"linux", "s390x"}, {"linux", "sparc64"},
{"netbsd", "386"}, {"netbsd", "amd64"}, {"netbsd", "arm"}, {"netbsd", "arm64"},
{"openbsd", "386"}, {"openbsd", "amd64"}, {"openbsd", "arm"}, {"openbsd", "arm64"}, {"openbsd", "mips64"},
{"openbsd", "ppc64"}, {"openbsd", "riscv64"},
{"plan9", "386"}, {"plan9", "amd64"}, {"plan9", "arm"},
{"solaris", "amd64"},
{"wasip1", "wasm"},
{"windows", "386"}, {"windows", "amd64"}, {"windows", "arm"}, {"windows", "arm64"},

> Строгая типизация

```
var a int32
var b int64

// wrong: b = a
b = int64(a)
```

> Возврат нескольких значений

```c
#include <stdio.h>

int some(int arg1, int arg2, int *res) {
    *res = 7
    return -1;
}

int main() {
    int ret;
    int result;

    ret = some(1, 2, &result);
    if (ret < 0) {
        printf("err result: %d\n", result);
    }
}
```

```go
package main

import (
    "fmt"
)

func some(arg1, arg2 int) (int, error) {
    return 0, fmt.Errorf("7")
}

func main() {
    res, err := some(1, 2)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(res)
}
```

> Статическая линковка

```
└─$ ldd bin.c
        linux-vdso.so.1 (0x00007ebfa269d000)
        libc.so.6 => /usr/lib/libc.so.6 (0x00007ebfa2482000)
        /lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2 (0x00007ebfa269f000)

└─$ ldd bin.go
        not a dynamic executable
```

```
CGO_ENABLED=0 go build -o bin.go cmd/main.go
```

> Embed (1.16)

(!= embedding)

```go
import (
    "embed"
)

var (
    //go:embed dir
    res embed.FS

    //go:embed file.png
    swagger []byte
)
```

```
└─$ tree
.
├── dir
├── file.go
└── file.png
```

> Unit тесты

```go
// main.go
package main

func sum(a, b int) int {
    return a + b
}
```

```go
// main_test.go
package main

import (
    "testing"
)


func TestSum(t *testing.T) {
    tests := []struct{
        a int
        b int
        wait int
    }{
        {a: 1, b: 2, wait: 3},
        {a: 2, b: 7, wait: 9},
    }

    for _, tc := range tests {
        got := sum(tc.a, tc.b)
        if got != tc.wait {
            t.Fatalf("sum(%d, %d) got: %d, wait: %d", tc.a, tc.b, got, tc.wait)
        }
    }
}
```

```
# Если тесты в текущей директории
└─$ go test -v .
=== RUN   TestSum
--- PASS: TestSum (0.00s)
PASS
ok      check   0.001s


# Если проект большой и хотим запустить все тесты
go test -v ./...
```

> Benchmark

```go
// main.go
package main

func sum(a, b int) int {
    return a + b
}

func stupidSum(a, b int) int {
    var res int
    for i := 0; i < a; i++ {
        res = res + 1
    }
    for i := 0; i < b; i++ {
        res = res + 1
    }
    return res
}
```

```go
// main_bench_test.go
package main

import (
    "testing"
)

func BenchmarkSum(b *testing.B) {
    for i := 0; i < b.N; i++ {
        sum(100, 100)
    }
}

func BenchmarkStupidSum(b *testing.B) {
    for i := 0; i < b.N; i++ {
        stupidSum(100, 100)
    }
}
```

```
└─$ go test -bench=.
goos: linux
goarch: amd64
pkg: check
cpu: 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
BenchmarkSum-8           1000000000               0.2423 ns/op
BenchmarkStupidSum-8     17587222                66.08 ns/op
PASS
ok      check   1.503s
```

> Race checker

> Fuzzing тестирование (1.18)

```go
// main_fuzz_test.go
package main

import (
    "testing"
)

func FuzzSum(f *testing.F) {
    f.Fuzz(func (t *testing.T, a int, b int) {
        sum(a, b)
    })
}
```

```
└─$ go test -fuzz=.
warning: starting with empty corpus
fuzz: elapsed: 0s, execs: 0 (0/sec), new interesting: 0 (total: 0)
fuzz: elapsed: 3s, execs: 604985 (201600/sec), new interesting: 1 (total: 1)
fuzz: elapsed: 6s, execs: 1233432 (209325/sec), new interesting: 1 (total: 1)
```

> Менеджер пакетов

> Подключение новых библиотек

> Кодстайл

```
# Если все файлы в одном месте
gofmt -w *.go

# Если нужно рекурсивно
find -name '*.go' -exec gofmt -w {} \;
```

Ни туда ни сюда:

> heap escape

```c
int a;

int *b = malloc(sizeof(int) * 10);
```

```go
var a int

b := make([]int, 10)
```

```go
package main

import (
    "fmt"
)

func main() {
    var a int
    var b int

    c := make([]int, 0, 10)
    d := make([]int, 0, 10)

    println(a)
    println(c)

    fmt.Println(b)
    fmt.Println(d)
}

// └─$ go build -o bin -gcflags "-m" main.go
// ./main.go:11:11: make([]int, 0, 10) does not escape
// ./main.go:12:11: make([]int, 0, 10) escapes to heap
// ./main.go:17:14: b escapes to heap
// ./main.go:18:14: d escapes to heap
```

> Garbage collector

> Нельзя средствами языка написать map

> Нельзя писать под embedded устройства

Минусы:

> __attribute__((__packed__))

> Нет прямого маппинга структуры в бинарные данные

```
typedef struct {
    uint8_t  len;
    uint16_t type;
    uint8_t  num;
} __attribute__((__packed__)) some_t;


uint8_t pBuff = { 0x11 , 0x22, ..., 0x18, 0x19 };
some_t some = (some_t)pBuff;
```

> Нет const в прототипах функции

```c
#include <stdio.h>

typedef struct test {
    int a;
    int b;
} test_t;

void print(const test_t * val) {
    printf("a: %d, b: %d\n", val->a, val->b);
}

int main() {
    test_t val;
    val.a = 7;
    val.b = 8;

    print(&val);
}
```

```
void print(const test_t const * val) {
    printf("a: %d, b: %d\n", val->a, val->b);
}
```

> Кросскомпиляция

https://github.com/HardDie/meetup_why_go_slides