C-LIS

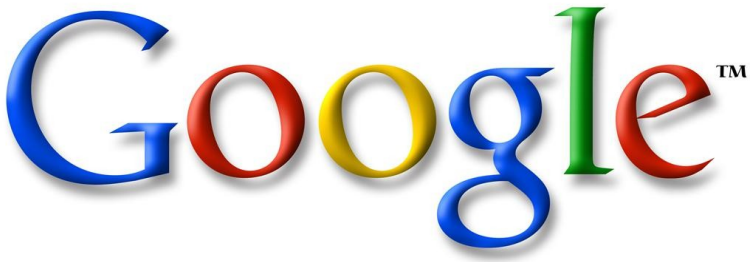
日本Androidの会神戸支部　　2011/09/24

devQuiz と Go言語

devQuizとは？

- Google主催のプログラミングコンテスト
- 出題は、Googleのテクノロジーに関係する問題
  - ウォームアップクイズ
  - 分野別クイズ

  そして、より純粋にプログラミングによる問題を解決力を見る「チャレンジクイズ」がある



Google Developer Day に参加するために、

devQuizで一定以上の点を獲得する必要がある

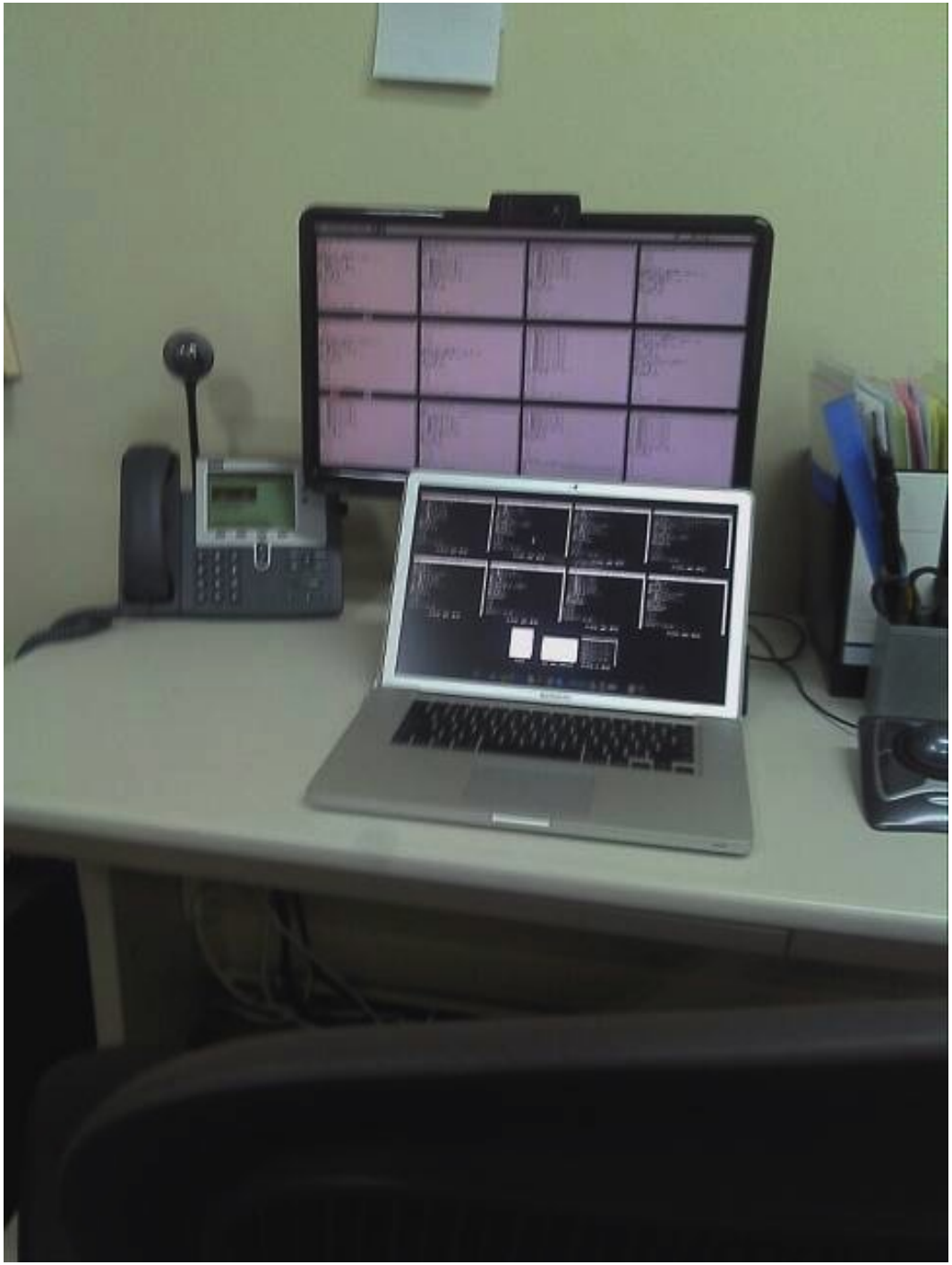
Google Developer Dayとは？

- Google Developer Day は、Google のプラットフォームやサービスに関する技術情報を紹介するイベント。
- 今年（2011年）は、１１月１日にパシフィコ横浜で、Google Developer Day 2011 Tokyo が、開催される。

問題

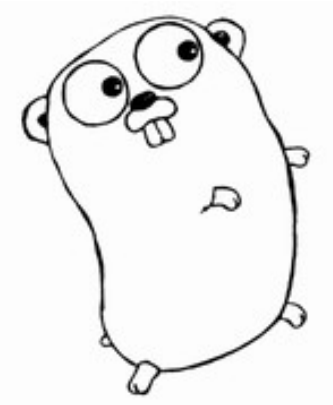
ウォームアップクイズ		
クイズ	40.0	/ 40
分野別クイズ (最大2問が得点に加算されます)		
Web Game	-	/ 30
Go!	30.0	/ 30
Android	30.0	/ 30
Google Apps Script	-	/ 30
一人ゲーム	-	/ 30
チャレンジクイズ		
スライドパズル	50.0	/ 50
合計	150.0	/ 150

みんなスライドパズルに夢中



けど、今回は、

Go言語



Go!

Go 言語で、PNG 画像を入力として受け取り、その画像が何色使っているかを返す関数

```
func CountColor(png io.Reader) int
```

を実装してください。PNG 画像は io.Reader 型で与えられます。
なお、入力の画像は R G B の各色の値が 0 から 255 までの 256 段階のいずれかであり、不透明（アルファチャンネルの値が常に 255）であることが保証されています。

サンプル画像

サンプルの答え

5

```
package main

import (
    "fmt"
    "io"
    "strings"
    "image"
    "image/png"
    "strconv"
)

func main() {
    png := GetPngBinary()
    cnt := CountColor(png)
    fmt.Println(cnt)
}

func GetPngBinary() io.Reader {
    // img_strの中身は提出するたびに変化します。
    // (内容は省略)
    img_str := ""
    return strings.NewReader(img_str)
}
```

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 16) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```

PNGの読込み

```
import (
    "fmt"
    "io"
    "strings"
    "image"
    "image/png"
    "strconv"
)
```

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 16) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```

ピクセルの各要素を取得

func (*ColorImage) RGBA

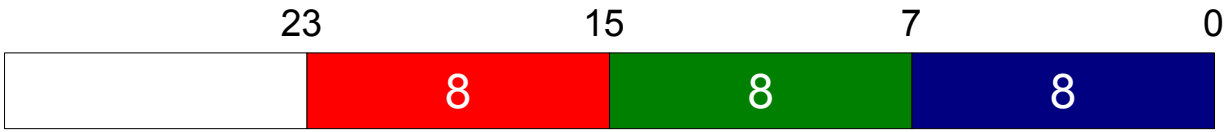
func (c *ColorImage) RGBA() (r, g, b, a uint32)

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 16) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```

1つに合成する



```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 16) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```

色数値を文字列に変換して
マップに追加

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 16) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```

マップに格納した個数を戻す

Length and capacity

The built-in functions len and cap take arguments of various types and return a result of type int. The implementation guarantees that the result always fits into an int.

Call	Argument type	Result
len(s)	string type	string length in bytes
	[n]T, *[n]T	array length (== n)
	[]T	slice length
	map[K]T	map length (number of defined keys)
	chan T	number of elements queued in channel buffer
cap(s)	[n]T, *[n]T	array length (== n)
	[]T	slice capacity
	chan T	channel buffer capacity

The capacity of a slice is the number of elements for which there is space allocated in the underlying array. At any time the following relationship holds:

```
0 <= len(s) <= cap(s)
```

http://golang.org/doc/go_spec.html#Length_and_capacity

前回の提出結果:

正解!

過去の提出結果はこちら.

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
            for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 26) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                    count[strconv.Uitoa(key)]++
                } else {
                    count[strconv.Uitoa(key)] = 1
                }
            }
        }
    }
```

しかし

提出した内容に

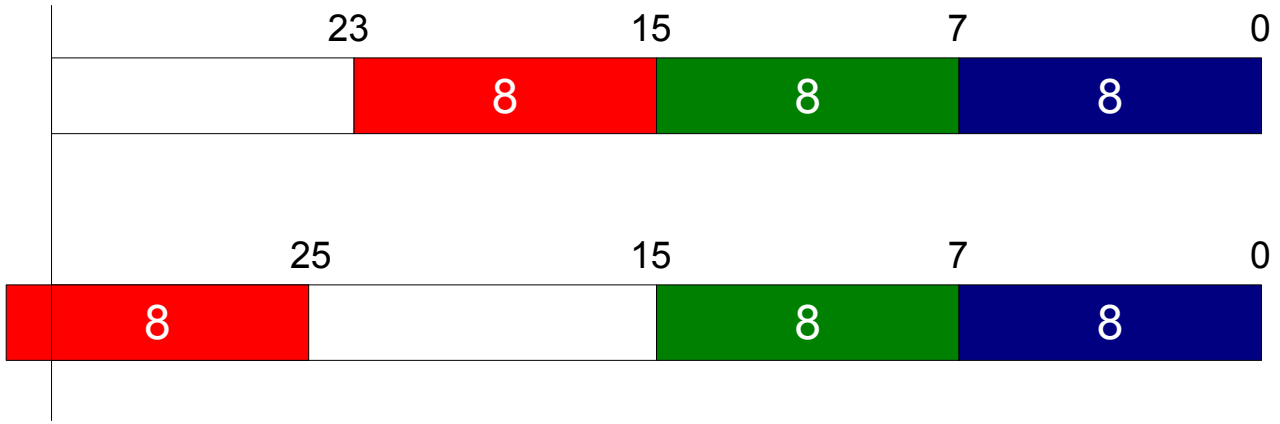
致命的なミスが。。。

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 26) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```

シフト桁数が間違っている



前回の提出結果:

正解!

過去の提出結果はこちら.

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
            for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 26) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                    count[strconv.Uitoa(key)]++
                } else {
                    count[strconv.Uitoa(key)] = 1
                }
            }
        }
    }
```

たまたま運が良かったのか！？

Numeric types

A *numeric type* represents sets of integer or floating-point values. The predeclared architecture-independent numeric types are:

```
uint8      the set of all unsigned  8-bit integers (0 to 255)
uint16     the set of all unsigned 16-bit integers (0 to 65535)
uint32     the set of all unsigned 32-bit integers (0 to 4294967295)
uint64     the set of all unsigned 64-bit integers (0 to 18446744073709551615)

int8       the set of all signed  8-bit integers (-128 to 127)
int16      the set of all signed 16-bit integers (-32768 to 32767)
int32      the set of all signed 32-bit integers (-2147483648 to 2147483647)
int64      the set of all signed 64-bit integers (-9223372036854775808 to 9223372036854775807)

float32    the set of all IEEE-754 32-bit floating-point numbers
float64    the set of all IEEE-754 64-bit floating-point numbers

complex64  the set of all complex numbers with float32 real and imaginary parts
complex128 the set of all complex numbers with float64 real and imaginary parts

byte       familiar alias for uint8
```

The value of an *n*-bit integer is *n* bits wide and represented using two's complement arithmetic.

There is also a set of predeclared numeric types with implementation-specific sizes:

```
uint     either 32 or 64 bits
int      same size as uint
uintptr  an unsigned integer large enough to store the uninterpreted bits of a pointer value
```

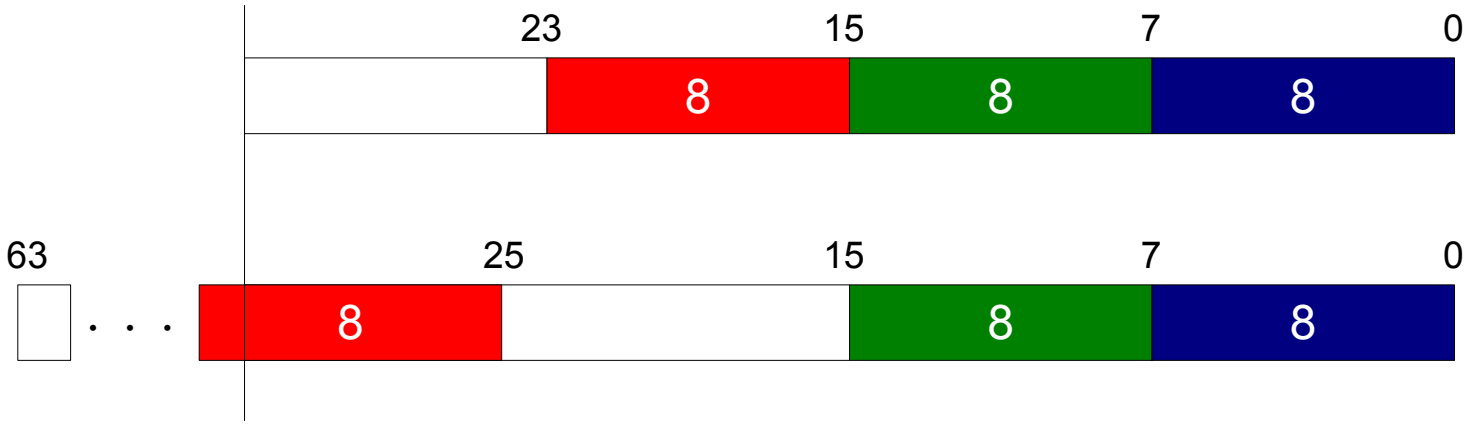
To avoid portability issues all numeric types are distinct except byte, which is an alias for uint8. Conversions are required when different numeric types are mixed in an expression or assignment. For instance, int32 and int are not the same type even though they may have the same size on a particular architecture.

http://golang.org/doc/go_spec.html

```
func CountColor(reader io.Reader) int {
    var count = map[string] int {}

    decodedPng, error := png.Decode(reader)
    if error == nil {
        var rect image.Rectangle = decodedPng.Bounds()
        for i := 0; i < rect.Size().X; i++ {
          for j := 0; j < rect.Size().Y; j++ {
                var pixel image.Color = decodedPng.At(i, j)
                var r,g,b,_ uint32 = pixel.RGBA()
                var key uint = (uint)((r << 26) + (g << 8) + (b))

                if _, ok := count[strconv.Uitoa(key)]; ok {
                        count[strconv.Uitoa(key)]++
                } else {
                        count[strconv.Uitoa(key)] = 1
                }
          }
        }
    }
    return len(count)
}
```



テスト環境 及び Googleの解答サーバーが

64bit 環境だったからセーフ！

ご清聴ありがとうございました。

ソースコードは github に公開しています。

https://github.com/keiji/devQuiz_GDD11JP