Ступникова Екатерина Петровна, 2 курс, направление ИТМ

Лабораторная работа № 2

Факс Цезаря

Цель работы

Произвести шифрование кодом Цезаря, сжатого при помощи кодирования длин серий изображения, передаваемого по факсу. Ознакомиться на примере кода Цезаря с поточными алгоритмами шифрования.

Задание

1. Смоделировать процесс передачи документа по факсу со сжатием.
2. Смоделировать процесс приема сжатого содержимого документа по факсу.
3. Произвести шифрование кодом Цезаря.
4. Реализовать обратную операцию расшифровки с известным ключом.
5. Произвести частотный анализ содержимого сжатого незашифрованного вектора.
6. Произвести анализ зашифрованного файла, воспользовавшись известными статистическими параметрами распределения значений в незашифрованных векторах.

Результаты выполнения задания

Программа была реализована с использованием языка программирования Java.

Полный код проекта доступен в репозитории: https://github.com/stupnikova-katya/ceasar

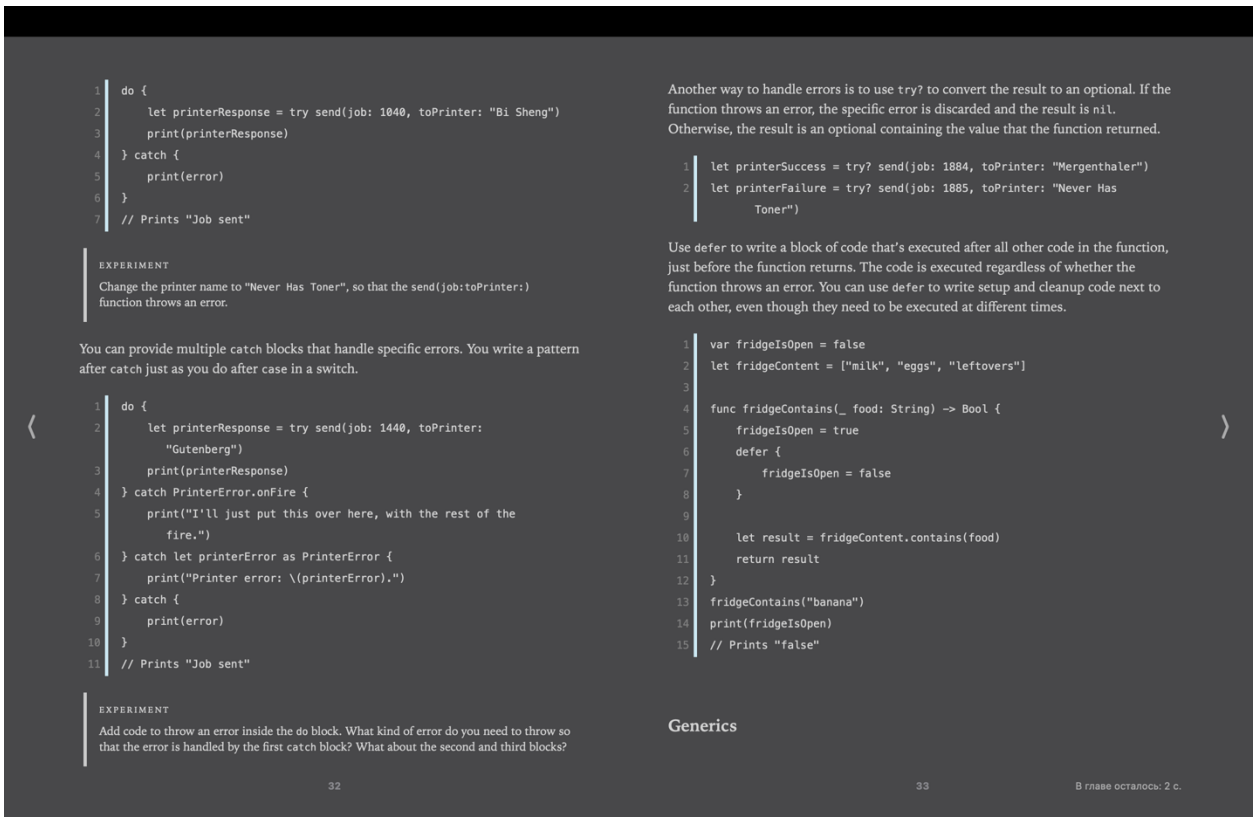
Для запуска программы необходимо выполнить ru.caesar.Main#main(String[] args).

1. Процесс передачи документа по факсу со сжатием смоделирован в классе ru.caesar.side.Sender и реализует следующий набор операций:
    a. открыть изображение (скриншот страницы книги или документа в формате A4)ru.caesar.side.Sender#readImageFile;
    b. преобразовать его в бинарное изображение ru.caesar.side.Sender#getBinaryImageVectorFromImage;
    c. представить результат в виде вектора ru.caesar.side.Sender#getBinaryImageVectorFromImage;

    d. обойти данный вектор и сформировать на его основе вектор в формате: [число белых пикселей, число черных пикселей, число белых пикселей, число черных пикселей...] ru.caesar.side.Sender.getBinaryImageVectorFromImage;
    e. сохранить содержимое вектора в бинарный файл ru.caesar.util.FileUtils.saveImageInBinaryVector.
2. Процесс приема сжатого содержимого документа по факсу смоделирован в классе ru.caesar.side.Receiver и реализует следующий набор операций:
    a. прочитать бинарный файл в вектор ru.caesar.util.FileUtils#readImageInBinaryVector;
    b. обратить операцию сжатия из пункта 1 ru.caesar.side.Receiver#getImageFromVector;
    c. представить полученный вектор в виде матрицы-бинарного изображения ru.caesar.side.Receiver#convertTo2D;
    d. сохранить полученное изображение в файл ru.caesar.util.FileUtils#saveImage.
3. Процесс шифрования кодом Цезаря (ru.caesar.domain.Ceaser#encode) смоделирован в классе ru.caesar.domain.Ceaser и реализует следующий набор операций:
    a. прочитать бинарный файл в вектор V ru.caesar.util.FileUtils#readImageInBinaryVector;
    b. для каждого значения Vi прибавлять к нему значение ключа k по модулю максимального числа для вашего типа данных max_val: (Vi + k) mod max_val ru.caesar.domain.Ceaser#createDecodedVector;
    c. сохранить полученный вектор в бинарный файл ru.caesar.util.FileUtils#saveImageInBinaryVector.
4. Процесс дешифрования кодом Цезаря (ru.caesar.domain.Ceaser#decode) смоделирован в классе ru.caesar.domain.Ceaser
5. Произведен частотный анализ содержимого сжатого незашифрованного вектора (ru.caesar.util.AnalysisUtil#writeImageStatisticsInFile).
    a. взяты 10 файлов-картинок с текстом с близкими свойствами (похожий размер шрифта и одинаковый формат страницы): src/resources/Swift1.png - src/resources/Swift10.png.
    b. Изображения были сжаты в соответствии с пунктом 1.

    c. Для их сжатого представления было посчитано, сколько раз встречается каждое значение.
    d. Таким образом была найдена первая десятка наиболее встречающихся значений в сжатых незашифрованных векторах. Все полученные статистики сохранены в файл src/resources/Statistics.txt и продублированы в ru.caesar.decoder.Decoder#SWIFT_STATS
6. Произведен анализ 3-х случайных зашифрованных файлов, с использованием известных статистических параметров распределения значений в незашифрованных (ru.caesar.decoder.Decoder#decodeFiles).
    a. Для создания трех случайных зашифрованных файлов был реализован метод ru.caesar.util.GeneratorUtil#generate3BinFiles
    b. Посчитаны для зашифрованного файла, сколько раз встречается там каждое значение.
    c. После перебора первые несколько наиболее встречающихся значений в сжатых незашифрованных векторах они были проверены на соответствие наиболее частому значению из зашифрованного.
    d. Ключ вычислен так, если бы это значение в зашифрованном файле соответствовало текущему из проверяемых.
    e. Попытаться расшифровать файл этим ключом. Если не выбрасывается исключение при попытке конвертировать вектор в изображение, то данный кандидат является успешным, и изображение сохраняется в виде файла.

Таким образом имелись следующие три пары шифруемых\дешифрованных изображений:



```
do {
    let printerResponse = try send(job: 1040, toPrinter: "Bi Sheng")
    print(printerResponse)
} catch {
    print(error)
}
// Prints "Job sent"
```

EXPERIMENT

Change the printer name to `"Never Has Toner"`, so that the `send(job:toPrinter:)` function throws an error.

You can provide multiple `catch` blocks that handle specific errors. You write a pattern after `catch` just as you do after `case` in a switch.

```
do {
    let printerResponse = try send(job: 1440, toPrinter:
       "Gutenberg")
    print(printerResponse)
} catch PrinterError.onFire {
    print("I'll just put this over here, with the rest of the
       fire.")
} catch let printerError as PrinterError {
    print("Printer error: \(printerError).")
} catch {
    print(error)
}
// Prints "Job sent"
```

EXPERIMENT

Add code to throw an error inside the `do` block. What kind of error do you need to throw so that the error is handled by the first `catch` block? What about the second and third blocks?

32

Another way to handle errors is to use `try?` to convert the result to an optional. If the function throws an error, the specific error is discarded and the result is `nil`. Otherwise, the result is an optional containing the value that the function returned.

```
let printerSuccess = try? send(job: 1884, toPrinter: "Mergenthaler")
let printerFailure = try? send(job: 1885, toPrinter: "Never Has
       Toner")
```

Use `defer` to write a block of code that's executed after all other code in the function, just before the function returns. The code is executed regardless of whether the function throws an error. You can use `defer` to write setup and cleanup code next to each other, even though they need to be executed at different times.

```
var fridgeIsOpen = false
let fridgeContent = ["milk", "eggs", "leftovers"]

func fridgeContains(_ food: String) -> Bool {
    fridgeIsOpen = true
    defer {
        fridgeIsOpen = false
    }

    let result = fridgeContent.contains(food)
    return result
}
fridgeContains("banana")
print(fridgeIsOpen)
// Prints "false"
```

Generics

33

В главе осталось: 2 с.

Рис. 1 – Шифруемое изображение


```
do {
    let printerResponse = try send(job: 1040, toPrinter: "Bi Sheng")
    print(printerResponse)
} catch {
    print(error)
}
// Prints "Job sent"
```

EXPERIMENT

Change the printer name to `"Never Has Toner"`, so that the `send(job:toPrinter:)` function throws an error.

You can provide multiple `catch` blocks that handle specific errors. You write a pattern after `catch` just as you do after `case` in a switch.

```
do {
    let printerResponse = try send(job: 1440, toPrinter:
       "Gutenberg")
    print(printerResponse)
} catch PrinterError.onFire {
    print("I'll just put this over here, with the rest of the
       fire.")
} catch let printerError as PrinterError {
    print("Printer error: \(printerError).")
} catch {
    print(error)
}
// Prints "Job sent"
```

EXPERIMENT

Add code to throw an error inside the `do` block. What kind of error do you need to throw so that the error is handled by the first `catch` block? What about the second and third blocks?

Another way to handle errors is to use `try?` to convert the result to an optional. If the function throws an error, the specific error is discarded and the result is `nil`. Otherwise, the result is an optional containing the value that the function returned.

```
let printerSuccess = try? send(job: 1884, toPrinter: "Mergenthaler")
let printerFailure = try? send(job: 1885, toPrinter: "Never Has
       Toner")
```

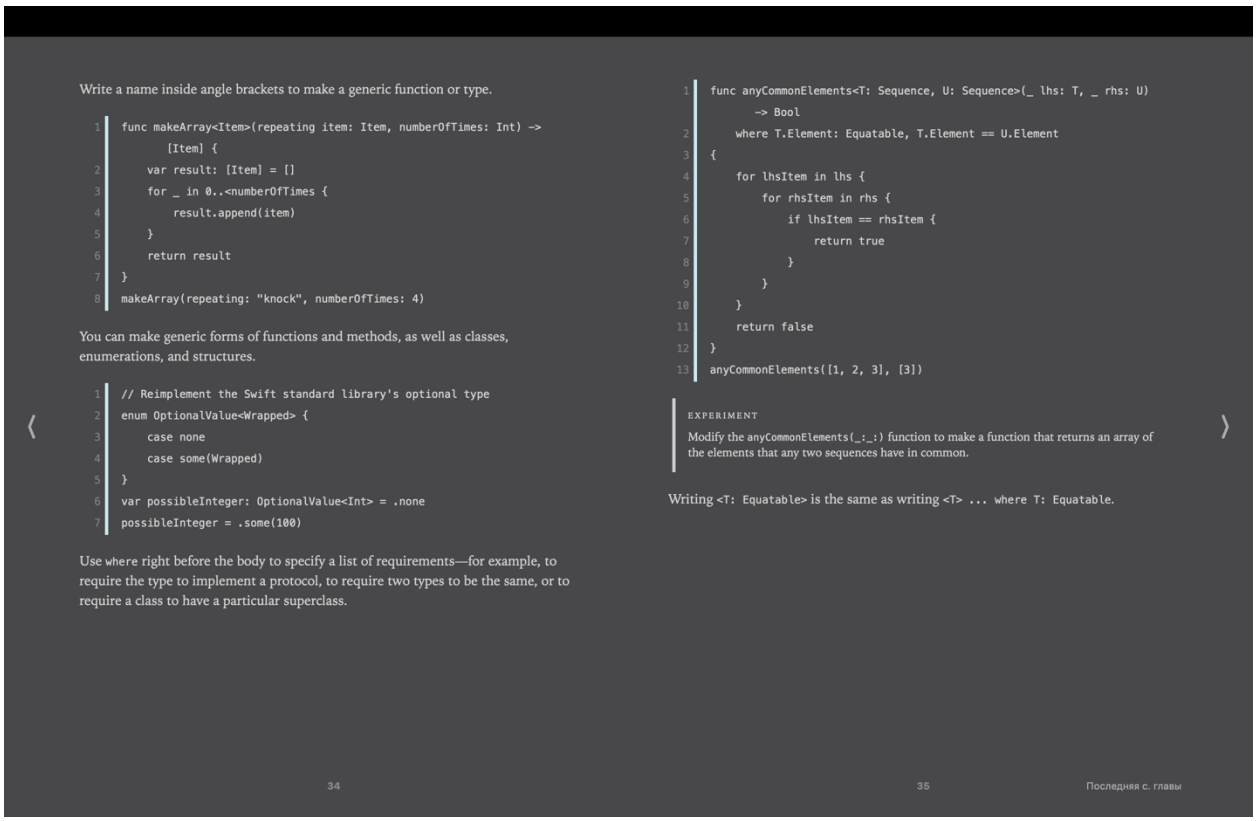
Use `defer` to write a block of code that's executed after all other code in the function, just before the function returns. The code is executed regardless of whether the function throws an error. You can use `defer` to write setup and cleanup code next to each other, even though they need to be executed at different times.

```
var fridgeIsOpen = false
let fridgeContent = ["milk", "eggs", "leftovers"]

func fridgeContains(_ food: String) -> Bool {
    fridgeIsOpen = true
    defer {
        fridgeIsOpen = false
    }

    let result = fridgeContent.contains(food)
    return result
}
fridgeContains("banana")
print(fridgeIsOpen)
// Prints "false"
```

Generics

В главе осталось: 2 с.

Рис. 2 – Дешифрованное изображение



Write a name inside angle brackets to make a generic function or type.

```
func makeArray<Item>(repeating item: Item, numberOfTimes: Int) ->
       [Item] {
    var result: [Item] = []
    for _ in 0..<numberOfTimes {
        result.append(item)
    }
    return result
}
makeArray(repeating: "knock", numberOfTimes: 4)
```

You can make generic forms of functions and methods, as well as classes, enumerations, and structures.

```
// Reimplement the Swift standard library's optional type
enum OptionalValue<Wrapped> {
    case none
    case some(Wrapped)
}
var possibleInteger: OptionalValue<Int> = .none
possibleInteger = .some(100)
```

Use `where` right before the body to specify a list of requirements—for example, to require the type to implement a protocol, to require two types to be the same, or to require a class to have a particular superclass.


```
func anyCommonElements<T: Sequence, U: Sequence>(_ lhs: T, _ rhs: U)
       -> Bool
    where T.Element: Equatable, T.Element == U.Element
{
    for lhsItem in lhs {
        for rhsItem in rhs {
            if lhsItem == rhsItem {
                return true
            }
        }
    }
    return false
}
anyCommonElements([1, 2, 3], [3])
```

EXPERIMENT

Modify the `anyCommonElements(_:_:)` function to make a function that returns an array of the elements that any two sequences have in common.

Writing `<T: Equatable>` is the same as writing `<T> ... where T: Equatable`.

Рис. 3 – Шифруемое изображение

Write a name inside angle brackets to make a generic function or type.

```
func makeArray<Item>(repeating item: Item, numberOfTimes: Int) ->
       [Item] {
    var result: [Item] = []
    for _ in 0..<numberOfTimes {
        result.append(item)
    }
    return result
}
makeArray(repeating: "knock", numberOfTimes: 4)
```

You can make generic forms of functions and methods, as well as classes, enumerations, and structures.

```
// Reimplement the Swift standard library's optional type
enum OptionalValue<Wrapped> {
    case none
    case some(Wrapped)
}
var possibleInteger: OptionalValue<Int> = .none
possibleInteger = .some(100)
```

Use `where` right before the body to specify a list of requirements—for example, to require the type to implement a protocol, to require two types to be the same, or to require a class to have a particular superclass.

```
func anyCommonElements<T: Sequence, U: Sequence>(_ lhs: T, _ rhs: U)
       -> Bool
    where T.Element: Equatable, T.Element == U.Element
{
    for lhsItem in lhs {
        for rhsItem in rhs {
            if lhsItem == rhsItem {
                return true
            }
        }
    }
    return false
}
anyCommonElements([1, 2, 3], [3])
```

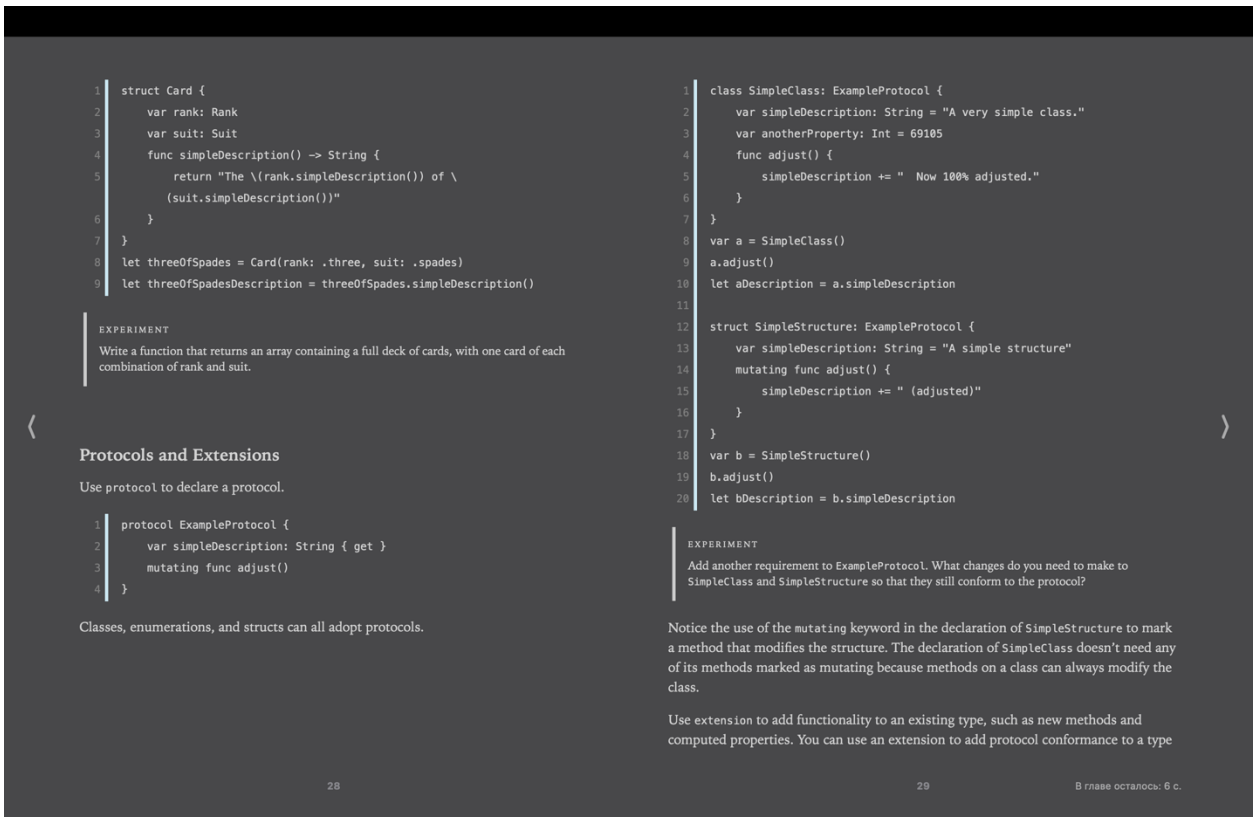
EXPERIMENT

Modify the `anyCommonElements(_:_:)` function to make a function that returns an array of the elements that any two sequences have in common.

Writing `<T: Equatable>` is the same as writing `<T> ... where T: Equatable`.

Последняя с. главы

Рис. 4 – Дешифрованное изображение



```
struct Card {
    var rank: Rank
    var suit: Suit
    func simpleDescription() -> String {
        return "The \(rank.simpleDescription()) of \
       (suit.simpleDescription())"
    }
}
let threeOfSpades = Card(rank: .three, suit: .spades)
let threeOfSpadesDescription = threeOfSpades.simpleDescription()
```

EXPERIMENT

Write a function that returns an array containing a full deck of cards, with one card of each combination of rank and suit.

Protocols and Extensions

Use `protocol` to declare a protocol.

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

Classes, enumerations, and structs can all adopt protocols.

28

```
class SimpleClass: ExampleProtocol {
    var simpleDescription: String = "A very simple class."
    var anotherProperty: Int = 69105
    func adjust() {
        simpleDescription += "  Now 100% adjusted."
    }
}
var a = SimpleClass()
a.adjust()
let aDescription = a.simpleDescription

struct SimpleStructure: ExampleProtocol {
    var simpleDescription: String = "A simple structure"
    mutating func adjust() {
        simpleDescription += " (adjusted)"
    }
}
var b = SimpleStructure()
b.adjust()
let bDescription = b.simpleDescription
```

EXPERIMENT

Add another requirement to `ExampleProtocol`. What changes do you need to make to `SimpleClass` and `SimpleStructure` so that they still conform to the protocol?

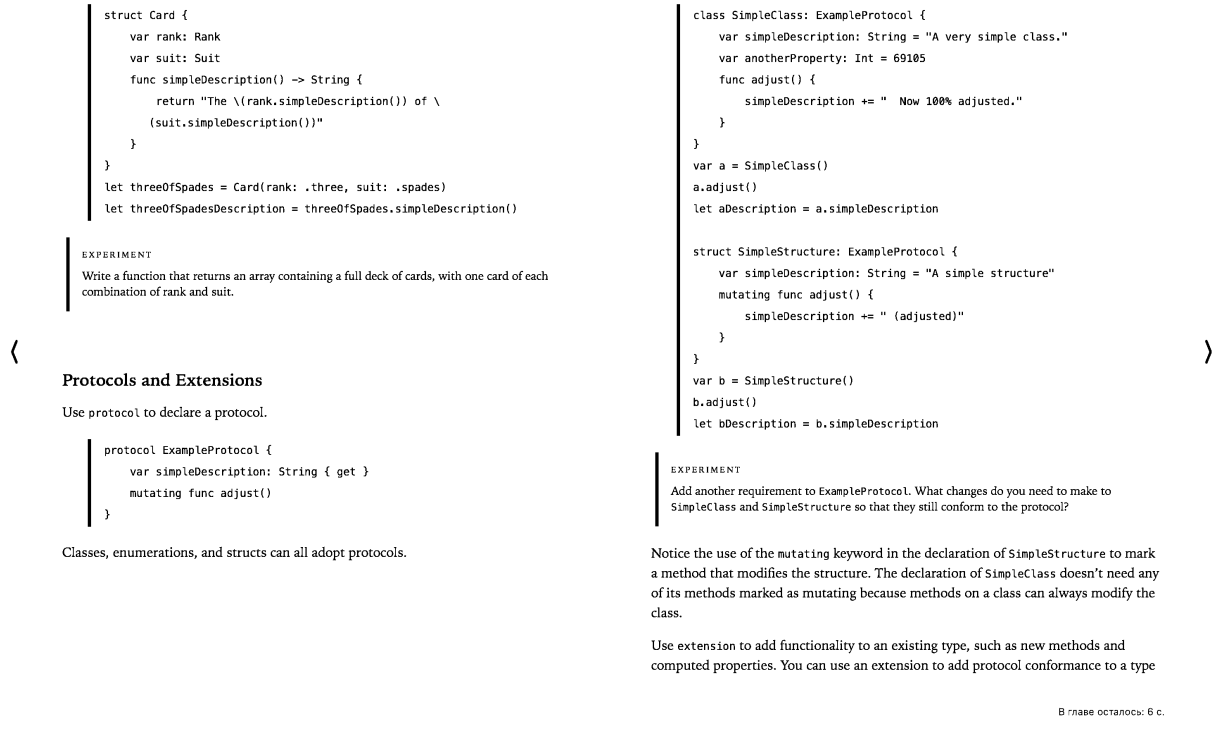
Notice the use of the `mutating` keyword in the declaration of `SimpleStructure` to mark a method that modifies the structure. The declaration of `SimpleClass` doesn't need any of its methods marked as mutating because methods on a class can always modify the class.

Use `extension` to add functionality to an existing type, such as new methods and computed properties. You can use an extension to add protocol conformance to a type

29

В главе осталось: 6 с.

Рис. 5 – Шифруемое изображение



```
struct Card {
    var rank: Rank
    var suit: Suit
    func simpleDescription() -> String {
        return "The \(rank.simpleDescription()) of \
       (suit.simpleDescription())"
    }
}
let threeOfSpades = Card(rank: .three, suit: .spades)
let threeOfSpadesDescription = threeOfSpades.simpleDescription()
```

EXPERIMENT

Write a function that returns an array containing a full deck of cards, with one card of each combination of rank and suit.

Protocols and Extensions

Use `protocol` to declare a protocol.

```
protocol ExampleProtocol {
    var simpleDescription: String { get }
    mutating func adjust()
}
```

Classes, enumerations, and structs can all adopt protocols.

```
class SimpleClass: ExampleProtocol {
    var simpleDescription: String = "A very simple class."
    var anotherProperty: Int = 69105
    func adjust() {
        simpleDescription += "  Now 100% adjusted."
    }
}
var a = SimpleClass()
a.adjust()
let aDescription = a.simpleDescription

struct SimpleStructure: ExampleProtocol {
    var simpleDescription: String = "A simple structure"
    mutating func adjust() {
        simpleDescription += " (adjusted)"
    }
}
var b = SimpleStructure()
b.adjust()
let bDescription = b.simpleDescription
```

EXPERIMENT

Add another requirement to `ExampleProtocol`. What changes do you need to make to `SimpleClass` and `SimpleStructure` so that they still conform to the protocol?

Notice the use of the `mutating` keyword in the declaration of `SimpleStructure` to mark a method that modifies the structure. The declaration of `SimpleClass` doesn't need any of its methods marked as mutating because methods on a class can always modify the class.

Use `extension` to add functionality to an existing type, such as new methods and computed properties. You can use an extension to add protocol conformance to a type

В главе осталось: 6 с.

Рис. 6 – Дешифрованное изображение

Сведения о частотном анализе для 10 исходных изображений:

Формат:

Наименованием файла

Значение: частота встречаемости

Swift1.png
2: 14845
3: 9802
4: 5549
5: 3625
1: 2981
6: 2399
32: 2312
7: 1779
11: 1746
12: 1639

Swift2.png
2: 16557
3: 8761
4: 5540
1: 3851
5: 3667
6: 2886
32: 2614
12: 2393
16: 2082
11: 2043

Swift3.png
2: 14845 3: 9802 4: 5549 5: 3625 1: 2981 6: 2399 32: 2312 7: 1779 11: 1746 12: 1639

Swift4.png
2: 14743 3: 10552 4: 5620 5: 3464 1: 2661 6: 2192 32: 1822 7: 1676 11: 1565 12: 1479

Swift5.png
2: 19321 3: 12214 4: 7393 5: 4579 1: 3770 32: 3178 6: 3073 7: 2292 33: 1981 11: 1960

Swift6.png
2: 11441 3: 9120 4: 4168 5: 2634 1: 1745 6: 1534 7: 1363 10: 1291 11: 1273 9: 1159

Swift7.png
2: 13360 3: 8792 4: 4495 5: 2890 1: 2571 6: 1993 32: 1691 11: 1653 12: 1614 7: 1582

Swift8.png
2: 12590 3: 9275 4: 4267 5: 2722 1: 2266 6: 1703 7: 1530 11: 1380 10: 1272 9: 1227

Swift9.png
2: 8425
3: 6604
4: 3035
5: 2126
1: 1437
6: 1133
7: 1023
11: 836
9: 707
10: 705

Swift10.png
2: 19368
3: 11578
4: 6837
5: 4581
1: 3955
6: 3173
32: 2543
7: 2282
12: 2180
11: 2160

Сводная таблица дешифрования файлов:

Название файла	Наиболее частое значение в зашифрованном виде	Наиболее частое значение в расшифрованном виде	Правильный вариант ключа	Номер подошедшего значения из таблицы с частотным анализом	Время, затраченное на перебор ключей и расшифровку
Swift8.png	1244	2	1242	1	2.03 с
Swift9.png	1159	2	1157	1	1.371 с
Swift6.png	445	2	443	1	1.371 с

```
Название файла: Swift8.png
Наиболее частое значение в зашифрованном виде: 1244
Наиболее частое значение в расшифрованном виде: 2
Правильный вариант ключа: 1242
Номер подошедшего значения из таблицы с  частотным анализом незашифрованных сжатых файлов: 1
Время, затраченное на перебор ключей и расшифровку: 2.03 с.
================================
Название файла: Swift9.png
Наиболее частое значение в зашифрованном виде: 1159
Наиболее частое значение в расшифрованном виде: 2
Правильный вариант ключа: 1157
Номер подошедшего значения из таблицы с  частотным анализом незашифрованных сжатых файлов: 1
Время, затраченное на перебор ключей и расшифровку: 1.371 с.
================================
Название файла: Swift6.png
Наиболее частое значение в зашифрованном виде: 445
Наиболее частое значение в расшифрованном виде: 2
Правильный вариант ключа: 443
Номер подошедшего значения из таблицы с  частотным анализом незашифрованных сжатых файлов: 1
Время, затраченное на перебор ключей и расшифровку: 1.371 с.
================================
```

Вывод: вычисление ключа с учетом статистических данных легко реализуется, так как шифрование Цезаря лишь заменяет одно значение другим, оставляя при этом статистические данные для зашифрованного и исходного файла одинаковыми.

Ответы на контрольные вопросы:

1. И отправителю, и получателю необходимо изначально знать ключ шифрования и размеры исходного изображения (ширину и высоту в пикселях).
2. Каждому пикселю будет соответствовать другой пиксель, такое изображение можно будет открыть, и скорее всего оно даже будет различимым, то есть шифрование не позволит скрыть исходную информацию. А еще данный способ накладывает очень узкое ограничение для уникальных значений ключа, так они будут принимать значения от 1 до 255 из-за того, что значение цвета RGB принимает значения из этого же диапазона.
3. В этом случае уже известны размеры изображения и можно просто перебрать все возможные ключи из [1, 255] пока изображение не станет читаемым.
4. Элементам для 2, 3 и 4 подряд идущих пикселей одного цвета соответствуют наиболее частые значения в сжатых файлах. Это связано с тем, что шифровались страницы с печатным текстом, и матрица разворачивалась в вектор (то есть страница считывалась построчно), а значит данные пиксели представляют из себя участки, кодирующие прямые линии, которые рисуют шрифт начертания букв.

Выводы

Я изучила алгоритм шифрования Цезаря, узнала основные принципы работы поточных алгоритмов шифрования, реализовала данный алгоритм в программе с использованием языка программирования Java и провела эксперимент с перебором возможных ключей и использованием статистических данных для дешифрования сжатых изображений для передачи по факсу.