Московский государственный технический университет имени Н. Э. Баумана.

Факультет “Информатика и системы управления”.

Кафедра “Системы обработки информации и управления”.

|  |  |
| --- | --- |
| Утверждаю: |  |
| Галкин В.А. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2021г. |

Курсовая работа

по курсу

Сетевые технологии в АСОИУ

«Программа пересылки файлов»

Описание программы

(вид документа)

бумага А4

(вид носителя)

17

(количество листов)

Вариант 24.

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛИ: |  |
| студенты группы ИУ5-64 |  |
| Сысойкин Е.М. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Кан А.Д. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Шпак И.Д. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г. |

Москва 2021

Оглавление

[1. Введение 3](#_Toc67089827)

[2. Класс ConnectionDetailsView. 3](#_Toc67089828)

[2.1. Переменные 3](#_Toc67089829)

[2.2. События 3](#_Toc67089830)

[2.3. Методы 3](#_Toc67089831)

[3. Класс MenuBar 3](#_Toc67089832)

[3.1. Переменные 3](#_Toc67089833)

[3.2. События 4](#_Toc67089834)

[3.3. Методы 4](#_Toc67089835)

[4. Класс TransferSettingView 4](#_Toc67089836)

[4.1. Переменные 4](#_Toc67089837)

[4.2. События 4](#_Toc67089838)

[4.3. Методы 4](#_Toc67089839)

[5. Класс MainWindowView 4](#_Toc67089840)

[5.1. Переменные 4](#_Toc67089841)

[5.2. Методы 4](#_Toc67089842)

[6. Класс Connection 5](#_Toc67089843)

[6.1. Переменные 5](#_Toc67089844)

[6.2. Методы 5](#_Toc67089845)

[7. Класс Coder 5](#_Toc67089846)

[7.1. Переменные 5](#_Toc67089847)

[7.2. Методы 5](#_Toc67089848)

[8. Листинг 6](#_Toc67089849)

[8.1. Класс ConnectionDetailsView. 6](#_Toc67089850)

[8.2. Класс MenuBar 7](#_Toc67089851)

[8.3. Класс TransferSettingView 10](#_Toc67089852)

[8.4. Класс MainWindowView 13](#_Toc67089853)

[8.5. Класс Connection 13](#_Toc67089854)

[8.6. Класс Coder 17](#_Toc67089855)

# Введение

Программный продукт написан на языке программирования Kotlin.

Для создания графического интерфейса и взаимодействия с COM-портом использовались стандартные библиотеки и элементы управления. Дополнительные функции, не относящиеся к стандартным, приведены ниже.

# Класс ConnectionDetailsView.

Класс, описывающий соединение и разъединение.

## Переменные

private lateinit var masterButtons: HBox – объект, содержащий в себе кнопки «Connect» и «Disconnect»

private lateinit var connectionStatusLabel: Label – надпись статуса соединения

override val root: Parent – нужно для описания событий

## События

checkbox("Master") – указывает, кто является отправителем файла

button("Connect") – появляется, если вы отправитель

button("Disconnect") - появляется, если вы отправитель

## Методы

override fun onCurrentSpeedChanged(speed: Int) – при изменении скорости.

override fun onConnectionUp() – при соединении.

override fun onConnectionDown() – при разъединении.

# Класс MenuBar

Класс, описывающий меню программы, в котором можно выбрать параметры порта и сам порт

## Переменные

private lateinit var devicesMenu: Menu – список устройств

private lateinit var speedsMenu: Menu – список скоростей

override val root: Parent – нужно для описания событий

## События

menu("Settings") – меню открытия настроек

item("Refresh") – кнопка обновления

menu("Device") – меню доступных устройств

menu("Port speed") – меню выбора скорости порта

## Методы

private fun updateDevices(menu: Menu) – обновление доступных устройств

private fun updateSpeeds(menu: Menu) – обновление скоростей портов.

# Класс TransferSettingView

Класс, описывающий главное окно программы.

## Переменные

Надписи:

private lateinit var selectedFileLabel: Label

private lateinit var downloadsFolderLabel: Label

private lateinit var progressBar: ProgressBar

private lateinit var downloadingFileLabel: Label

private lateinit var downloadedPercentageLabel: Label

override val root: Parent – нужно для описания событий

## События

button("Select file to transfer") – кнопка выбора файла

button("SEND") – кнопка отправки

button("Select Downloads folder") – кнопка выбора папки назначения

## Методы

override fun onStartDownload(file: File) – при старте загрузки

override fun onEndDownload(file: File) – при завершении загрузки

override fun onProgressUpdate(progress: Double) – при обновлении процесса отправки

# Класс MainWindowView

Класс, объединяющий все вышеперечисленные классы.

## Переменные

override val root – нужен для объединения всех вышеперечисленных классов.

## Методы

Add – добавляет элементы View.

# Класс Connection

Класс, описывающий соединение и разъединение.

## Переменные

private var device = SerialPort(deviceName)

private val listeners = mutableListOf<ConnectionListener>()

var isOpened: Boolean = false

private var uploadListener: BinaryUploadListener? = null

private var downloadListener: BinaryDownloadListener? = null

## Методы

fun openConnection(): Boolean

fun connect(): Boolean

fun changeMasterSpeed(speed: Int): Boolean

fun changeDevice(deviceName: String): Boolean

fun disconnect(): Boolean

fun writeBinaryData(data: ByteArray): Boolean

fun writeFileHeader(file: File): Boolean

fun writeAck(): Boolean

fun writeError(): Boolean

fun closeConnection(): Boolean

# Класс Coder

Класс, описывающий кодирование и декодирование.

## Переменные

val c1 – проверочный бит 1

val c2 – проверочный бит 2

val c3 – проверочный бит 3

val c4 – проверочный бит 4

val h1 – бит ошибки 1

val h2 – бит ошибки 2

val h3 – бит ошибки 3

val h4 – бит ошибки 4

## Методы

private fun code(input: Short): Short - закодировать

private fun decode(input: Short): Short – декодировать

private fun codeShortByIndex(index: Int, bytes: ByteArray): Short – закодировать short

private fun writeShortByIndex(index: Int, bytes: ByteArray, short: Short, size: Int): ByteArray – написать short

private fun decodeShortFromBytes(bytes: ByteArray): Pair<Short, Int> - декодировать short

fun codeByteArray(bytes: ByteArray): ByteArray – закодировать bytearray

fun decodeByteArray(bytes: ByteArray): ByteArray – декодировать bytearray

# Листинг

## Класс ConnectionDetailsView.

package views

import FileTransferApp.Companion.myApp

import core.ConnectionListener

import javafx.geometry.Insets

import javafx.geometry.Pos

import javafx.scene.Parent

import javafx.scene.control.Label

import javafx.scene.layout.HBox

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.GlobalScope

import kotlinx.coroutines.javafx.JavaFx

import kotlinx.coroutines.launch

import tornadofx.\*

class ConnectionDetailsView: View(), ConnectionListener {

private lateinit var masterButtons: HBox

private lateinit var connectionStatusLabel: Label

init {

myApp.subscribeOnDevice(this)

}

override val root: Parent = vbox(alignment = Pos.TOP\_CENTER) {

checkbox("Master") {

isSelected = myApp.isMaster

action {

myApp.isMaster = isSelected

masterButtons.isVisible = myApp.isMaster

}

vboxConstraints {

margin = Insets(10.0, 0.0, 10.0, 0.0)

}

}

label("Disconnected") { connectionStatusLabel = this }

hbox(alignment = Pos.TOP\_CENTER) {

masterButtons = this

isVisible = myApp.isMaster

button("Connect") {

hboxConstraints {

margin = Insets(5.0, 5.0, 5.0, 5.0)

}

}.action { myApp.connect() }

button("Disconnect") {

hboxConstraints {

margin = Insets(5.0, 5.0, 5.0, 5.0)

}

}.action { myApp.disconnect() }

}

}

override fun onCurrentDeviceChanged() {}

override fun onCurrentSpeedChanged(speed: Int) {

GlobalScope.launch(Dispatchers.JavaFx) {

connectionStatusLabel.text = "Connected via ${myApp.currentDeviceName} with $speed"

}

}

override fun onConnectionUp() {

GlobalScope.launch(Dispatchers.JavaFx) {

connectionStatusLabel.text = "Connected via ${myApp.currentDeviceName} with ${myApp.currentMasterSpeed}"

}

}

override fun onConnectionDown() {

GlobalScope.launch(Dispatchers.JavaFx) {

connectionStatusLabel.text = "Disconnected"

}

}

}

## Класс MenuBar

package views

import FileTransferApp.Companion.myApp

import javafx.scene.Parent

import javafx.scene.control.CheckMenuItem

import javafx.scene.control.Menu

import jssc.SerialPort

import jssc.SerialPortList

import tornadofx.\*

import views.css.Styles

import java.util.regex.Pattern

class MenuBar : View() {

companion object {

private val speeds = arrayListOf(

SerialPort.BAUDRATE\_110,

SerialPort.BAUDRATE\_300,

SerialPort.BAUDRATE\_600,

SerialPort.BAUDRATE\_1200,

SerialPort.BAUDRATE\_4800,

SerialPort.BAUDRATE\_9600,

SerialPort.BAUDRATE\_14400,

SerialPort.BAUDRATE\_19200,

SerialPort.BAUDRATE\_38400,

SerialPort.BAUDRATE\_57600,

SerialPort.BAUDRATE\_115200,

SerialPort.BAUDRATE\_128000,

SerialPort.BAUDRATE\_256000

)

}

private lateinit var devicesMenu: Menu

private lateinit var speedsMenu: Menu

override val root: Parent = menubar {

menu("Settings") {

item("Refresh").action {

updateDevices(devicesMenu)

}

separator()

menu("Device") {

devicesMenu = this

updateDevices(devicesMenu)

}

menu("Port speed") {

speedsMenu = this

updateSpeeds(speedsMenu)

}

}

}

private fun updateDevices(menu: Menu) {

val ports = SerialPortList.getPortNames(Pattern.compile("(ttyS|ttyUSB|ttyACM|ttyAMA|rfcomm|tnt)[0-9]{1,3}"))

if (ports.isEmpty()) {

myApp.currentDeviceName = ""

}

menu.items.clear()

ports.forEach {

val item = CheckMenuItem(it)

item.addClass(Styles.checkMenuItem)

item.isSelected = myApp.currentDeviceName == it

item.action {

myApp.currentDeviceName = it

updateDevices(menu)

}

menu += item

}

}

private fun updateSpeeds(menu: Menu) {

menu.items.clear()

speeds.forEach {

val item = CheckMenuItem(it.toString())

item.addClass(Styles.checkMenuItem)

item.isSelected = myApp.currentMasterSpeed == it

item.action {

myApp.currentMasterSpeed = it

updateSpeeds(menu)

}

menu += item

}

}

}

## Класс TransferSettingView

package views

import FileTransferApp.Companion.myApp

import ProgressListener

import javafx.geometry.Insets

import javafx.geometry.Pos

import javafx.scene.control.Alert

import javafx.scene.control.ButtonType

import javafx.scene.control.Label

import javafx.scene.control.ProgressBar

import tornadofx.\*

import javafx.stage.FileChooser

import kotlinx.coroutines.Dispatchers

import kotlinx.coroutines.GlobalScope

import kotlinx.coroutines.javafx.JavaFx

import kotlinx.coroutines.launch

import java.io.File

class TransferSettingView : View(), ProgressListener {

private lateinit var selectedFileLabel: Label

private lateinit var downloadsFolderLabel: Label

private lateinit var progressBar: ProgressBar

private lateinit var downloadingFileLabel: Label

private lateinit var downloadedPercentageLabel: Label

init {

myApp.subscribeOnProgressListener(this)

}

override val root = vbox(alignment = Pos.TOP\_CENTER) {

hbox(alignment = Pos.TOP\_CENTER) {

vboxConstraints {

margin = Insets(10.0, 5.0, 0.0, 5.0)

}

button("Select file to transfer") {

action {

val filter = arrayOf(FileChooser.ExtensionFilter("Empty filter", "\*"))

val fileList = chooseFile("Select file to transfer", filter, mode = FileChooserMode.Single)

if (fileList.isNotEmpty()) {

myApp.transferFile = fileList[0]

selectedFileLabel.text = "Selected files:${myApp.transferFile.name}"

} else {

selectedFileLabel.text = "Not selected file"

}

}

}

button("SEND") {

action { myApp.sendSelectedFile() }

}

}

label("Not selected file") { selectedFileLabel = this }

button("Select Downloads folder") {

vboxConstraints {

margin = Insets(10.0, 5.0, 0.0, 5.0)

}

action {

val folder = chooseDirectory()

if (folder != null) {

downloadsFolderLabel.text = "${folder.absolutePath}${File.separator}"

myApp.downloadsFolder = "${folder.absolutePath}${File.separator}"

}

}

}

label(myApp.downloadsFolder) { downloadsFolderLabel = this }

borderpane {

vboxConstraints {

margin = Insets(10.0, 20.0, 0.0, 20.0)

}

top = label {

downloadingFileLabel = this

isVisible = false

useMaxWidth = true

}

center = progressbar {

progressBar = this

useMaxWidth = true

isVisible = false

}

right = label {

downloadedPercentageLabel = this

isVisible = false

useMaxWidth = true

borderpaneConstraints {

margin = Insets(0.0, 0.0, 0.0, 5.0)

}

}

}

}

override fun onStartDownload(file: File) {

GlobalScope.launch(Dispatchers.JavaFx) {

progressBar.progress = 0.0

progressBar.isVisible = true

downloadingFileLabel.text = file.name

downloadingFileLabel.isVisible = true

downloadedPercentageLabel.text = "0 %"

downloadedPercentageLabel.isVisible = true

}

}

override fun onProgressUpdate(progress: Double) {

GlobalScope.launch(Dispatchers.JavaFx) {

progressBar.progress = progress

downloadedPercentageLabel.text = "${(progress \* 100).toInt()} %"

}

}

override fun onEndDownload(file: File) {

GlobalScope.launch(Dispatchers.JavaFx) {

progressBar.isVisible = false

downloadingFileLabel.isVisible = false

downloadedPercentageLabel.isVisible = false

val alert = Alert(Alert.AlertType.INFORMATION, "File \"${file.name}\" with ${file.length()} bytes", ButtonType.OK)

alert.headerText = "Downloaded successfully!"

alert.isResizable = true

alert.initOwner(currentWindow)

alert.show()

}

}

}

## Класс MainWindowView

package views

import tornadofx.\*

import views.css.Styles

class MainWindowView : View() {

override val root = vbox {

addClass(Styles.base)

add(MenuBar())

add(ConnectionDetailsView())

add(TransferSettingView())

}

## Класс Connection

package core

import jssc.SerialPort

import jssc.SerialPortEvent

import jssc.SerialPortEventListener

import jssc.SerialPortException

import utils.DataUtils.Companion.readFrames

import utils.DataUtils.Companion.toByteArray

import utils.DataUtils.Companion.writeFrame

import java.io.File

class Connection(deviceName: String, private var currentSpeed: Int, var isMaster: Boolean) {

companion object {

private const val DEBUG = false

}

private var device = SerialPort(deviceName)

private val listeners = mutableListOf<ConnectionListener>()

var isOpened: Boolean = false

private set(value) {

field = value

if (value) {

listeners.forEach { it.onConnectionUp() }

} else {

listeners.forEach { it.onConnectionDown() }

}

}

private var uploadListener: BinaryUploadListener? = null

private var downloadListener: BinaryDownloadListener? = null

fun openConnection(): Boolean {

return try {

val isOpened = if (!device.isOpened) device.openPort() else true

// Поднимаем изначально соединение на скорости по-умолчанию

val isValid = device.setParams(

SerialPort.BAUDRATE\_110,

SerialPort.DATABITS\_8,

SerialPort.STOPBITS\_1,

SerialPort.PARITY\_NONE

)

device.addEventListener(PortListener())

device.isOpened && isOpened && isValid

} catch (e: SerialPortException) {

println("${e.methodName}: ${e.portName}")

false

}

}

fun connect(): Boolean {

if (!isMaster) return false

return try {

device.writeFrame(Frame(Frame.Type.LINK))

&& device.writeFrame(Frame(Frame.Type.SYNC, syncSpeed = currentSpeed))

} catch (e: SerialPortException) {

false

}

}

fun changeMasterSpeed(speed: Int): Boolean {

if (!isMaster) return false

currentSpeed = speed

return try {

val syncFrameResult = device.writeFrame(Frame(Frame.Type.SYNC, syncSpeed = currentSpeed))

val isValid =

device.setParams(currentSpeed, SerialPort.DATABITS\_8, SerialPort.STOPBITS\_1, SerialPort.PARITY\_NONE)

listeners.forEach { it.onCurrentSpeedChanged(currentSpeed) }

isValid && syncFrameResult

} catch (e: SerialPortException) {

false

}

}

fun changeDevice(deviceName: String): Boolean {

closeConnection()

device = SerialPort(deviceName)

return deviceName.isNotEmpty() && !openConnection()

}

fun disconnect(): Boolean {

return try {

device.writeFrame(Frame(Frame.Type.DOWN\_LINK))

} catch (e: SerialPortException) {

false

}

}

fun writeBinaryData(data: ByteArray): Boolean {

return try {

device.writeFrame(Frame(Frame.Type.BINARY\_DATA, data))

} catch (e: SerialPortException) {

false

}

}

fun writeFileHeader(file: File): Boolean {

return try {

device.writeFrame(Frame(Frame.Type.FILE\_HEADER, file.length().toByteArray() + file.name.toByteArray()))

} catch (e: SerialPortException) {

false

}

}

fun writeAck(): Boolean {

return try {

device.writeFrame(Frame(Frame.Type.ACK))

} catch (e: SerialPortException) {

false

}

}

fun writeError(): Boolean {

return try {

device.writeFrame(Frame(Frame.Type.ERROR))

} catch (e: SerialPortException) {

false

}

}

fun closeConnection(): Boolean {

return try {

device.closePort()

} catch (e: SerialPortException) {

false

}

}

fun addListener(listener: ConnectionListener) {

listeners.add(listener)

}

fun removeListener(listener: ConnectionListener) {

listeners.remove(listener)

}

fun setDataListener(listener: BinaryUploadListener) {

uploadListener = listener

}

fun setDataListener(listener: BinaryDownloadListener) {

downloadListener = listener

}

private inner class PortListener : SerialPortEventListener {

override fun serialEvent(event: SerialPortEvent?) {

if (event == null) {

return

}

if (DEBUG) {

println(String.format("EVENT(%s), %d %d", event.portName, event.eventType, event.eventValue))

}

when (event.eventType) {

SerialPortEvent.RXCHAR -> {

device.readFrames().forEach { frame ->

if (DEBUG) {

println(String.format("FRAME(%s): %s %d", event.portName, frame.type, frame.data.size))

}

when (frame.type) {

Frame.Type.LINK -> {

if (!isMaster)

device.writeFrame(Frame(Frame.Type.LINK))

isOpened = true

}

Frame.Type.ACK -> {

uploadListener?.onAckReceived()

}

Frame.Type.SYNC -> {

if (frame.syncSpeed < 0 || isMaster)

return

currentSpeed = frame.syncSpeed

val isValid = device.setParams(

currentSpeed,

SerialPort.DATABITS\_8,

SerialPort.STOPBITS\_1,

SerialPort.PARITY\_NONE

)

if (isValid) {

device.writeFrame(Frame(Frame.Type.SYNC, syncSpeed = currentSpeed))

listeners.forEach { it.onCurrentSpeedChanged(currentSpeed) }

}

}

Frame.Type.FILE\_HEADER -> {

downloadListener?.onFileHeaderReceived(frame.data)

}

Frame.Type.BINARY\_DATA -> {

downloadListener?.onBinaryDataReceived(frame.data)

}

Frame.Type.ERROR -> {

uploadListener?.onErrorReceived()

}

Frame.Type.DOWN\_LINK -> {

if (!isMaster)

device.writeFrame(Frame(Frame.Type.DOWN\_LINK))

isOpened = false

}

Frame.Type.UNKNOWN -> {

}

}

}

}

SerialPortEvent.TXEMPTY -> {

}

}

}

}

}

## Класс Coder

package core

import java.nio.ByteBuffer

import java.nio.ByteOrder

import kotlin.experimental.and

import kotlin.experimental.or

import kotlin.experimental.xor

object Coder {

private fun code(input: Short): Short {

val c1 = (input and 1) xor ((input and 2).toInt() shr 1).toShort() xor

((input and 8).toInt() shr 3).toShort() xor ((input and 16).toInt() shr 4).toShort() xor

((input and 64).toInt() shr 6).toShort() xor ((input and 256).toInt() shr 8).toShort() xor

((input and 1024).toInt() shr 10).toShort()

val c2 = (input and 1) xor ((input and 4).toInt() shr 2).toShort() xor

((input and 8).toInt() shr 3).toShort() xor ((input and 32).toInt() shr 5).toShort() xor

((input and 64).toInt() shr 6).toShort() xor ((input and 512).toInt() shr 9).toShort() xor

((input and 1024).toInt() shr 10).toShort()

val c3 = ((input and 2).toInt() shr 1).toShort() xor ((input and 4).toInt() shr 2).toShort() xor

((input and 8).toInt() shr 3).toShort() xor ((input and 128).toInt() shr 7).toShort() xor

((input and 256).toInt() shr 8).toShort() xor ((input and 512).toInt() shr 9).toShort() xor

((input and 1024).toInt() shr 10).toShort()

val c4 = ((input and 16).toInt() shr 4).toShort() xor ((input and 32).toInt() shr 5).toShort() xor

((input and 64).toInt() shr 6).toShort() xor ((input and 128).toInt() shr 7).toShort() xor

((input and 256).toInt() shr 8).toShort() xor ((input and 512).toInt() shr 9).toShort() xor

((input and 1024).toInt() shr 10).toShort()

return ((input.toInt() shl 4) xor ((c4.toInt() shl 3)) xor ((c3.toInt() shl 2)) xor

((c2.toInt() shl 1)) xor ((c1.toInt() shl 0))).toShort()

}

private fun decode(input: Short): Short? {

val h1 = ((input and 1 \* 16).toInt() shr 4).toShort() xor ((input and 2 \* 16).toInt() shr 1 + 4).toShort() xor

((input and 8 \* 16).toInt() shr 3 + 4).toShort() xor ((input and 16 \* 16).toInt() shr 4 + 4).toShort() xor

((input and 64 \* 16).toInt() shr 6 + 4).toShort() xor ((input and 256 \* 16).toInt() shr 8 + 4).toShort() xor

((input and 1024 \* 16).toInt() shr 10 + 4).toShort() xor ((input and 1).toInt() shr 0).toShort()

if (h1 != 0.toShort()) return null

val h2 = ((input and 1 \* 16).toInt() shr 4).toShort() xor ((input and 4 \* 16).toInt() shr 2 + 4).toShort() xor

((input and 8 \* 16).toInt() shr 3 + 4).toShort() xor ((input and 32 \* 16).toInt() shr 5 + 4).toShort() xor

((input and 64 \* 16).toInt() shr 6 + 4).toShort() xor ((input and 512 \* 16).toInt() shr 9 + 4).toShort() xor

((input and 1024 \* 16).toInt() shr 10 + 4).toShort() xor ((input and 2).toInt() shr 1).toShort()

if (h2 != 0.toShort()) return null

val h3 =

((input and 2 \* 16).toInt() shr 1 + 4).toShort() xor ((input and 4 \* 16).toInt() shr 2 + 4).toShort() xor

((input and 8 \* 16).toInt() shr 3 + 4).toShort() xor ((input and 128 \* 16).toInt() shr 7 + 4).toShort() xor

((input and 256 \* 16).toInt() shr 8 + 4).toShort() xor ((input and 512 \* 16).toInt() shr 9 + 4).toShort() xor

((input and 1024 \* 16).toInt() shr 10 + 4).toShort() xor ((input and 4).toInt() shr 2).toShort()

if (h3 != 0.toShort()) return null

val h4 =

((input and 16 \* 16).toInt() shr 4 + 4).toShort() xor ((input and 32 \* 16).toInt() shr 5 + 4).toShort() xor

((input and 64 \* 16).toInt() shr 6 + 4).toShort() xor ((input and 128 \* 16).toInt() shr 7 + 4).toShort() xor

((input and 256 \* 16).toInt() shr 8 + 4).toShort() xor ((input and 512 \* 16).toInt() shr 9 + 4).toShort() xor

((input and 1024 \* 16).toInt() shr 10 + 4).toShort() xor ((input.toInt() and 8) shr 3).toShort()

if (h4 != 0.toShort()) return null

return ((input xor (input and 1) xor (input and 2) xor (input and 4) xor (input and 8)).toInt() shr 4).toShort()

}

private fun codeShortByIndex(index: Int, bytes: ByteArray): Short {

var indexToRead = index

var byteToRead = index / 8

var a: Short = 0

var bitsToRead = 11

if (byteToRead > bytes.lastIndex)

return -1

if (indexToRead + 10 > (bytes.size \* 8) - 1) {

while (indexToRead < bytes.size \* 8) {

if (indexToRead / 8 > byteToRead) byteToRead += 1

a = ((a.toInt() shl 1) or (bytes[byteToRead].toInt() shr (7 - (indexToRead % 8)) and 1)).toShort()

indexToRead += 1

bitsToRead -= 1

}

} else {

while (bitsToRead > 0) {

if (indexToRead / 8 > byteToRead) byteToRead += 1

a = ((a.toInt() shl 1) or (bytes[byteToRead].toInt() shr (7 - (indexToRead % 8)) and 1)).toShort()

indexToRead += 1

bitsToRead -= 1

}

}

a = code(a)

a = a xor ((1 shl (15 - bitsToRead)).toShort())

return a

}

private fun writeShortByIndex(index: Int, bytes: ByteArray, short: Short, size: Int): ByteArray {

var byteToWriteAt = index / 8

val a: Short = short

for (i in index until index + size) {

if (i / 8 > byteToWriteAt) {

byteToWriteAt += 1

}

if (size - 1 - (i - index) > 7 - i % 8) {

bytes[byteToWriteAt] =

bytes[byteToWriteAt] or

((a.toInt() and (1 shl (size - 1 - (i - index)))) shr (size - 1 - (i - index) - (7 - i % 8))).toByte()

} else {

bytes[byteToWriteAt] =

bytes[byteToWriteAt] or

((a.toInt() and (1 shl (size - 1 - (i - index)))) shl ((7 - i % 8) - (size - 1 - (i - index)))).toByte()

}

}

return bytes

}

private fun decodeShortFromBytes(bytes: ByteArray): Pair<Short, Int> {

var a: Short = ByteBuffer.wrap(bytes).order(ByteOrder.LITTLE\_ENDIAN).short

var index = 0

for (i in 4..15) {

if (((bytes[i / 8].toInt() shr i % 8) and 1) == 1) {

index = i

}

}

a = a xor (1 shl index).toShort()

index -= 4

a = decode(a) ?: return Pair(0, -1)

return Pair(a, index)

}

fun codeByteArray(bytes: ByteArray): ByteArray {

var toIndex = 0

var fromIndex = 0

var shortByte: Short

val codedBytes: ByteArray = if ((bytes.size \* 8) % 11 == 0) {

ByteArray((bytes.size \* 8 / 11) \* 2)

} else {

ByteArray(((bytes.size \* 8 / 11) + 1) \* 2)

}

while (toIndex < codedBytes.size - 1) {

shortByte = codeShortByIndex(bytes = bytes, index = fromIndex)

codedBytes[toIndex] = (shortByte and 0xff).toByte()

codedBytes[toIndex + 1] = ((shortByte.toInt() shr 8).toShort() and 0xff).toByte()

fromIndex += 11

toIndex += 2

}

return codedBytes

}

fun decodeByteArray(bytes: ByteArray): ByteArray? {

var decodedBytes = ByteArray(bytes.size / 2 \* 11 / 8)

var toIndex = 0

var fromIndex = 0

while (fromIndex <= bytes.lastIndex - 1) {

val bytesToDecode = bytes.slice(fromIndex..fromIndex + 1).toByteArray()

val (short, size) = decodeShortFromBytes(bytesToDecode)

if (size < 0) return null

decodedBytes = writeShortByIndex(toIndex, decodedBytes, short, size)

fromIndex += 2

toIndex += size

}

return decodedBytes

}

}