ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра прикладной математики и кибернетики

Лабораторная работа №8

Выполнили:
Студенты 4 курса ИВТ,
группы ИП-013
Копытина Татьяна, Семилетко Максим

Работу проверил: доцент кафедры ПМиК
Перцев И.В.

Новосибирск 2024 г.

Оглавление

Задание

Написать программу для декодирования и вывода на экран PCX файла.

Листинг программы

```
import random
import math
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

BMP_HEADER_BSIZE = 14
BMP_INFO_HEADER_BSIZE = 40

class BmpFile:
    def init(self, name):
        self.name = name
        self.fileObj = None
        self.header = None
        self.infoHeader = None
        self.palette = None
        self.paletteSize = None
        self.colorCount = None
        self.bpp = None
        self.padding = None

        self.type = None
        self.size = None
        self.reserved = None
        self.offset = None

        self.infoHeaderSize = None
```

```
        self.width = None
        self.height = None
        self.planes = None
        self.depthColor = None
        self.compression = None
        self.compressedSize = None
        self.xPixPM = None
        self.yPixPM = None
        self.usedColors = None
        self.importantColors = None

    def PrintInfo(self):
        print("---------HEADER----------")
        print(f"TYPE: {self.type}")
        print(f"FILE SIZE: {self.size}")
        print(f"RESERVED: {self.reserved}")
        print(f"DATA OFFSET: {self.offset}")
        print("--------INFO HEADER--------")
        print(f"HEADER SIZE: {self.infoHeaderSize}")
        print(f"WIDTH: {self.width}")
        print(f"HEIGHT: {self.height}")
        print(f"PLANES: {self.planes}")
        print(f"DEPTH: {self.depthColor}")
        print(f"COMPRESSION: {self.compression}")
        print(f"COMPRESSED SIZE: {self.compressedSize}")
        print(f"X RESOLUTION: {self.xPixPM}")
        print(f"Y RESOLUTION: {self.yPixPM}")
        print(f"USED COLORS: {self.usedColors}")
        print(f"IMPORTANT COLORS: {self.importantColors}")
        print()
```

```
class BmpFileReader:

    def init(self, fileName):

        self.bmpObj = BmpFile(fileName)


    def Read(self):

        self.bmpObj.fileObj = open(self.bmpObj.name, 'rb')

        self.bmpObj.header =
self.bmpObj.fileObj.read(BMP_HEADER_BSIZE)

        # HEADER

        self.bmpObj.type = self.bmpObj.header[:2].decode('utf-8')

        self.bmpObj.size = int.from_bytes(self.bmpObj.header[2:6],
'little')

        self.bmpObj.reserved =
int.from_bytes(self.bmpObj.header[6:10], 'little')

        self.bmpObj.offset = int.from_bytes(self.bmpObj.header[10:14],
'little')

        # HEADER #


        self.bmpObj.infoHeader =
self.bmpObj.fileObj.read(BMP_INFO_HEADER_BSIZE)

        # INFO HEADER

        self.bmpObj.infoHeaderSize =
int.from_bytes(self.bmpObj.infoHeader[:4], 'little')

        self.bmpObj.width =
int.from_bytes(self.bmpObj.infoHeader[4:8], 'little')

        self.bmpObj.height =
int.from_bytes(self.bmpObj.infoHeader[8:12], 'little')

        self.bmpObj.planes =
int.from_bytes(self.bmpObj.infoHeader[12:14], 'little')

        self.bmpObj.depthColor =
int.from_bytes(self.bmpObj.infoHeader[14:16], 'little')

        self.bmpObj.compression =
int.from_bytes(self.bmpObj.infoHeader[16:20], 'little')
```

```
        self.bmpObj.compressedSize =
int.from_bytes(self.bmpObj.infoHeader[20:24], 'little')

        self.bmpObj.xPixPM =
int.from_bytes(self.bmpObj.infoHeader[24:28], 'little')

        self.bmpObj.yPixPM =
int.from_bytes(self.bmpObj.infoHeader[28:32], 'little')

        self.bmpObj.usedColors =
int.from_bytes(self.bmpObj.infoHeader[32:36], 'little')

        self.bmpObj.importantColors =
int.from_bytes(self.bmpObj.infoHeader[36:40], 'little')

        # INFO HEADER #



        self.bmpObj.colorCount = pow(2, self.bmpObj.depthColor)

        self.bmpObj.paletteSize = self.bmpObj.colorCount * 4

        #self.bmpObj.palette =
self.bmpObj.fileObj.read(self.bmpObj.paletteSize)



        self.bmpObj.bpp = self.bmpObj.depthColor // 8

        self.bmpObj.padding = (4 - (self.bmpObj.width *
self.bmpObj.bpp) % 4) % 4



        return self.bmpObj.fileObj

def GenerateNewPalette(self, pixels, width, height):

        colors = {}

        for y in range(height):

            for x in range(width):

                flattenColor = (pixels[y, x][0] >> 4 << 4, pixels[y,
x][1] >> 4 << 4, pixels[y, x][2] >> 4 << 4)

                colors[flattenColor] = colors[flattenColor] + 1 if
flattenColor in colors else 1



        colors = list(colors.items())

        colors.sort(key=lambda x: x[1], reverse=False)
```

```
        newPalette = []
        newPalette.append(colors.pop()[0])
        newColorCount = 1

        while newColorCount < self.outputColorNum:
            newColor = colors.pop()[0]
            for color in newPalette:
                if 128*128*3 < self.CountDelta(color, newColor):
                    newPalette.append(newColor)
                    newColorCount += 1
                    break
        return newPalette
    row = originalFile.read((self.width + self.padding) * self.bpp)
                    newFile.write(row)

                    for _ in range(borderWidth):
                        newFile.write(random.randint(0,
colorNum).to_bytes(self.bpp, 'little'))

                    newFile.write(b'\x00' * (padding - self.padding))

                for _ in range(borderWidth):
                    for _ in range(newWidth):
                        newFile.write(random.randint(0,
colorNum).to_bytes(self.bpp, 'little'))
                    newFile.write(b'\x00' * padding)
def EncodeText(self, sizePercent):
        self.encodeOffset = math.ceil(8 * sizePercent)
        with open('text.txt', 'r') as textFile:
            textBits = ""
            readBitsCount = int(self.compressedSize * sizePercent)
```

```
            readBitsCount = (readBitsCount - readBitsCount % 24) * 8
            text = textFile.read()
            for i in text:
                textBits = textBits + bin(ord(i))[2:].zfill(8)
                if len(textBits) >= readBitsCount:
                    break

with open(self.name, 'rb') as originalFile:
            header = originalFile.read(self.offset)
            graphImg = np.zeros((self.height, self.width, 3),
dtype=np.uint8)

            for y in range(self.height - 1, -1, -1):
                for x in range(self.width):
                    blue = int.from_bytes(originalFile.read(1),
'little')
                    green = int.from_bytes(originalFile.read(1),
'little')
                    red = int.from_bytes(originalFile.read(1),
'little')
                    graphImg[y, x] = [red, green, blue]

                originalFile.read(self.padding)

            bitCounter = 0
            textBitsCount = len(textBits)
            self.textBitsCount = textBitsCount

            for y in range(self.height - 1, -1, -1):
                for x in range(self.width):
                    if bitCounter < textBitsCount:
                        for i in range(3):
```

```
                            botRowOffset = bitCounter +
(self.encodeOffset * i)
                            graphImg[y, x][i] = ((graphImg[y, x][i] >>
self.encodeOffset) << self.encodeOffset) |
int(textBits[botRowOffset:botRowOffset+self.encodeOffset], 2)


                        bitCounter += self.encodeOffset * 3


            with open('encoded_' + self.name, 'wb') as newFile:
                newHeader = bytearray(header)
                newFile.write(newHeader)


                newPixels = bytearray()
                for y in range(self.height - 1, -1, -1):
                    for x in range(self.width):
                        blue = int(graphImg[y, x][2]).to_bytes(1,
'little')
                        green = int(graphImg[y, x][1]).to_bytes(1,
'little')
                        red = int(graphImg[y, x][0]).to_bytes(1,
'little')
                        newPixels.extend(blue)
                        newPixels.extend(green)
                        newPixels.extend(red)


                    newPixels.extend(b'\x00' * self.padding)


                newFile.write(newPixels)


            plt.figure()
            plt.imshow(graphImg)
            plt.axis('off')
            return graphImg
```

```
    def DecodeText(self):
        bits = ""
        decodedText = ""
        bitsStr = ""
        bitCounter = 0

        with open('encoded_' + self.name, 'rb') as originalFile:
            header = originalFile.read(self.offset)

            graphImg = np.zeros((self.height, self.width, 3),
dtype=np.uint8)

            for y in range(self.height - 1, -1, -1):
                for x in range(self.width):
                    blue = int.from_bytes(originalFile.read(1),
'little')
                    green = int.from_bytes(originalFile.read(1),
'little')
                    red = int.from_bytes(originalFile.read(1),
'little')
                    graphImg[y, x] = [red, green, blue]

                originalFile.read(self.padding)

            textBitsCount = self.textBitsCount

            for y in range(self.height - 1, -1, -1):
                for x in range(self.width):
                    if bitCounter < textBitsCount:
                        binPowOffset = (pow(2, self.encodeOffset) - 1)
```

```
                        bits = bits + bin(graphImg[y, x][0] &
binPowOffset)[2:].zfill(self.encodeOffset)
                        bits = bits + bin(graphImg[y, x][1] &
binPowOffset)[2:].zfill(self.encodeOffset)
                        bits = bits + bin(graphImg[y, x][2] &
binPowOffset)[2:].zfill(self.encodeOffset)

                        bitCounter += self.encodeOffset * 3

            with open('decoded_text.txt', 'w') as decodedTxt:
                for i in range(0, len(bits), 8):
                    decodedText += chr(int(bits[i:i+8], 2))
                    a = 0
                decodedTxt.write(decodedText)
def Rewrite(self, newName):
        with open(newName, 'wb') as newFileObj:
            newFileObj.write(self.bmpObj.header)
            newFileObj.write(self.bmpObj.infoHeader)
            newFileObj.write(self.bmpObj.palette)
            newFileObj.write(self.bmpObj.fileObj.read())
            newFileObj.close()


class PcxFile:
    def init(self, outputColorNum, filename):
        self.outputColorNum = outputColorNum
        with open(filename, 'rb') as file:
            self.header = file.read(128)

            self.depth = int.from_bytes(self.header[3:4], 'little')

            self.width = self.header[8] + (self.header[9] << 8) -
self.header[4] - (self.header[5] << 8) + 1
```

```
            self.height = self.header[10] + (self.header[11] << 8) -
self.header[6] - (self.header[7] << 8) + 1


            file.seek(-768, 2)

            self.palette = np.frombuffer(file.read(),
dtype=np.uint8).reshape((256, 3))

            self.graphImg = np.zeros((self.height, self.width, 3),
dtype=np.uint8)


            file.seek(128)

            x, y = 0, 0

            while y < self.height:

                x = 0

                while x < self.width:

                    binByte = file.read(1)

                    byte = int.from_bytes(binByte, 'little')

                    if byte < 192:

                        self.graphImg[y, x] = self.palette[byte]

                        x += 1

                    else:

                        count = byte - 192

                        binRepeatedByte = file.read(1)

                        repeatedByte = int.from_bytes(binRepeatedByte,
'little')

                        if count == 1 and repeatedByte == 0: continue


                        for _ in range(count):

                            if x >= self.width: break

                            self.graphImg[y, x] =
self.palette[repeatedByte]

                            x += 1

                y += 1
```

```
def Convert(self):
        self.newPalette = self.GenerateNewPalette(self.graphImg,
self.width, self.height)
        dictPalette = {self.newPalette[i]: i for i in
range(len(self.newPalette))}
        imgBytes = []

        for y in range(self.height - 1, -1, -1):
            for x in range(self.width):
                self.graphImg[y, x] =
self.GetSimilarColor(self.newPalette, self.graphImg[y, x])
                colorInd = dictPalette.get(tuple(self.graphImg[y, x]))
                if colorInd == None:
                    colorInd = 0
                imgBytes.append(colorInd)

        self.imgZipBytes = []
        for i in range(0, len(imgBytes), 2):
            colorByte = 0
            pixel1 = (colorByte | imgBytes[i]) << 4
            #pixel2 = pixel1 | imgBytes[i]
            pixel2 = pixel1 | imgBytes[i + 1]
            self.imgZipBytes.append(pixel2)

        self.WriteOutputFile()

    def WriteOutputFile(self):
        bmpHeader = bytearray()

        headerSize = 54
```

```
        paletteSize = 4 * 16


        bmpHeader.extend(b'BM')
        bmpHeader.extend(int(self.width * self.height // 2 +
paletteSize + headerSize).to_bytes(4, 'little'))
        bmpHeader.extend(b'\x00\x00\x00\x00')
        bmpHeader.extend(int(paletteSize + headerSize).to_bytes(4,
'little'))


        bmpHeader.extend(int(40).to_bytes(4, 'little'))
        bmpHeader.extend(int(self.width).to_bytes(4, 'little'))
        bmpHeader.extend(int(self.height).to_bytes(4, 'little'))
        # Planes
        bmpHeader.extend(int(1).to_bytes(2, 'little'))
        # bit count
        bmpHeader.extend(int(4).to_bytes(2, 'little'))
        # compression
        bmpHeader.extend(int(0).to_bytes(4, 'little'))
        # compressed size
        bmpHeader.extend(int(0).to_bytes(4, 'little'))
        # pix per m X
        bmpHeader.extend(int(0).to_bytes(4, 'little'))
        # pix per m Y
        bmpHeader.extend(int(0).to_bytes(4, 'little'))
        # used colors
        bmpHeader.extend(int(0).to_bytes(4, 'little'))
        # important colors
        bmpHeader.extend(int(0).to_bytes(4, 'little'))


        paletteBytes = []
        for color in self.newPalette:
```

```
            r, g, b = color
            paletteBytes.extend([b, g, r, 0])

        with open('converted_pcx.bmp', 'wb') as f:
            f.write(bytes(bmpHeader))
            f.write(bytes(paletteBytes))
            f.write(bytes(self.imgZipBytes))

    def Show(self):
        plt.imshow(self.graphImg)
        plt.axis('off')
        plt.show()

    def CountDelta(self, left, right):
        return sum([(x - y) ** 2 for x, y in zip(left, right)])
def DecodePcxScript():
    pcxFile = PcxFile(16, "200001.pcx")
    #pcxFile = PcxFile(16, "CAT256.pcx")
    pcxFile.Show()

def ConvertScript():
    pcxFile = PcxFile(16, "200001.pcx")
    #pcxFile = PcxFile(16, "CAT256.pcx")
    pcxFile.Convert()
    pcxFile.Show()
    if name == 'main':
     DecodePcxScript()
     ConvertScript()
```

Результат работы программы