ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра прикладной математики и кибернетики

Лабораторная работа №3

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 3](#_Toc160977429)

[Листинг программы 3](#_Toc160977430)

[Результат работы программы 10](#_Toc160977431)

# Задание

Преобразовать TrueColor BMP файл, развернув растр на 90 градусов. Длина строки BMP файла выравнивается по 32-битовой границе, (4-м байт), при необходимости к каждой строке в файле добавляются выравнивающие байты! Изменить соответствующие поля в заголовке и сохранить файл под новым именем.

# 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 Rotate90(self):

with open(self.name, 'rb') as originalFile:

header = originalFile.read(BMP\_HEADER\_BSIZE + BMP\_INFO\_HEADER\_BSIZE)

if self.depthColor <= 8:

palette = originalFile.read(self.paletteSize)

originalPixels = originalFile.read()

newWidth = self.height

newHeight = self.width

with open('rotated90\_' + self.name, 'wb') as newFile:

newHeader = bytearray(header)

newHeader[18:22] = newWidth.to\_bytes(4, 'little')

newHeader[22:26] = newHeight.to\_bytes(4, 'little')

newFile.write(newHeader)

if self.depthColor <= 8:

newFile.write(palette)

padding = b'\x00' \* ((4 - (newWidth \* self.bpp) % 4) % 4)

newPixels = bytearray()

for x in range(self.width):

for y in range(self.height):

pixelPos = y \* self.width \* self.bpp + x \* self.bpp

newPixels.extend(originalPixels[pixelPos : (pixelPos + self.bpp)])

newPixels.extend(padding)

newFile.write(newPixels)

def RotateScript():

bmpReader = BmpFileReader('fish.BMP')

bmpReader.Read()

bmpReader.bmpObj.PrintInfo()

bmpReader.bmpObj.Rotate90()

if name == 'main':

RotateScript()

# Результат работы программы



