МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Отчет по лабораторной работе № 1

По дисциплине «Компьютерная графика и геометрия»

Выполнил студент группы №M3102  
*Ларин Владислав Денисович*

Проверил:  
*Скаков Павел Сергеевич*

САНКТ-ПЕТЕРБУРГ  
2019

***Цель работы*:** изучить алгоритмы и реализовать программу, выполняющую простые преобразования серых и цветных изображений в формате PNM.

***Описание*:**

Программа должна поддерживать серые и цветные изображения (варианты PNM P5 и P6), самостоятельно определяя формат по содержимому. Аргументы программе передаются через командную строку:

*lab#.exe <имя\_входного\_файла> <имя\_выходного\_файла> <преобразование>*

где <преобразование>:

0 - инверсия,

1 - зеркальное отражение по горизонтали,

2 - зеркальное отражение по вертикали,

3 - поворот на 90 градусов по часовой стрелке,

4 - поворот на 90 градусов против часовой стрелки.

***Теоретическая часть*:**

Описание файла pnm:

Первые два байта файла кодируют сочетание символов P5 или P6, указывающих на формат файла (оттенки серого или RGB соответственно). После них идет перевод строки. Четвертый, пятый, шестой, восьмой, девятый и десятый байты кодируют ширину и высоту соответственно (в виде трехзначного числа). Далее снова следует переход на новую строку. В третьей строке содержится три байта, кодирующих число максимально доступного значения пикселя данного файла.

Далее следует массив чисел, кодирующих саму картинку.

P5: по одному байту на пиксель.

P6: по три байта на пиксель (Red, Green, Blue соответственно).

***Экспериментальная часть:***

Программа написана на языке С++ (стандарт С++17).

Данные считываются с помощью функции fread в массив pixel, память для которого выделяется с помощью функции malloc. Каждое преобразование реализовано с помощью отдельной функции. В конце программы (или в случае ошибки во входных данных) память, отведенная под массивы и выделенная с помощью функции malloc, очищается с помощью функции free, а открытые файлы закрываются.

***Вывод:***

Программа производит обработку цветных изображений и изображений в оттенках серого. Данные представлены с помощью последовательности байтов, характеризующих оттенок. Для RGB-изображений пиксель характеризуется тройкой байтов. В случае некорректно введенных входных данных или ошибки при выделении памяти, программа выводит соответствующий комментарий и завершает свою работу, предварительно закрыв открытые файлы и очистив выделенную память. Все полученные результаты совпадают с теоретическими выкладками.

***Листинг:***

main.cpp

#include **<iostream>**#include **<string>**#include **<stdlib.h>  
  
typedef unsigned char** uchar;  
  
**using namespace** std;  
  
  
**int** readFile(**int** argc, **char** \*argv[], **int** &type, **int** &w, **int** &h, **int** &dummy, uchar\*\* pixel)  
{  
 **if** (argc != 4)  
 {  
 cout << **"Invalid request"**;  
 **return** 2;  
 }  
  
 **char** \*fileName\_in = argv[1];  
 FILE \*f\_in = fopen(fileName\_in, **"rb"**);  
 **if** (f\_in == **NULL**)  
 {  
 cout << **"Invalid input file"**;  
 **return** 3;  
 }  
  
 fscanf(f\_in, **"P%i%i%i%i\n"**, &type, &w, &h, &dummy);  
  
 **if** ((type < 5 || type > 6) || (w <= 0 || h <= 0) || (dummy != 255))  
 {  
 cout << **"Incorrect parameters"**;  
 fclose(f\_in);  
 **return** 4;  
 }  
  
 **if** (type == 5)  
 {  
 \*pixel = (uchar \*) malloc(**sizeof**(uchar) \* h \* w);  
 }  
 **else if** (type == 6)  
 {  
 \*pixel = (uchar \*) malloc(**sizeof**(uchar) \* h \* w \* 3);  
 }  
 **if** (\*pixel == **NULL**)  
 {  
 cout << **"Allocation memory failed"**;  
 fclose(f\_in);  
 **return** 5;  
 }  
  
 **int** startPosition = ftell(f\_in);  
 fseek(f\_in, 0, **SEEK\_END**);  
 **int** countPixels = ftell(f\_in) - startPosition;  
 fseek(f\_in, startPosition, 0);  
 **if** (type == 5)  
 {  
 **if** (countPixels != w \* h)  
 {  
 cout << **"Not enough data"**;  
 fclose(f\_in);  
 **return** 6;  
 }  
 fread(\*pixel, **sizeof**(uchar), w \* h, f\_in);  
 }  
 **else if** (type == 6)  
 {  
 **if** (countPixels != w \* h \* 3)  
 {  
 cout << **"Not enough data"**;  
 fclose(f\_in);  
 **return** 6;  
 }  
 fread(\*pixel, **sizeof**(uchar), w \* h \* 3, f\_in);  
 }  
 fclose(f\_in);  
 **return** 0;  
}  
  
  
**void** inversion(**int** type, **int** w, **int** h, uchar\*\* pixel)  
{  
 **if** (type == 5)  
 {  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 (\*pixel)[i \* w + j] = 255 - (\*pixel)[i \* w + j];  
 }  
 }  
 }  
 **else if** (type == 6)  
 {  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 (\*pixel)[(i \* w + j) \* 3] = 255 - (\*pixel)[(i \* w + j) \* 3];  
 (\*pixel)[(i \* w + j) \* 3 + 1] = 255 - (\*pixel)[(i \* w + j) \* 3 + 1];  
 (\*pixel)[(i \* w + j) \* 3 + 2] = 255 - (\*pixel)[(i \* w + j) \* 3 + 2];  
 }  
 }  
 }  
}  
  
  
**void** horizontalShow(**int** type, **int** w, **int** h, uchar\*\* pixel)  
{  
 **if** (type == 5)  
 {  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w / 2; j++)  
 {  
 swap((\*pixel)[i \* w + j], (\*pixel)[(i + 1) \* w - j - 1]);  
 }  
 }  
 }  
 **else if** (type == 6)  
 {  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w / 2; j++)  
 {  
 swap((\*pixel)[(i \* w + j) \* 3], (\*pixel)[((i + 1) \* w - j - 1) \* 3]);  
 swap((\*pixel)[(i \* w + j) \* 3 + 1], (\*pixel)[((i + 1) \* w - j - 1) \* 3 + 1]);  
 swap((\*pixel)[(i \* w + j) \* 3 + 2], (\*pixel)[((i + 1) \* w - j - 1) \* 3 + 2]);  
 }  
 }  
 }  
}  
  
  
**void** verticalShow(**int** type, **int** w, **int** h, uchar\*\* pixel)  
{  
 **if** (type == 5)  
 {  
 **for** (**int** i = 0; i < h / 2; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 swap((\*pixel)[i \* w + j], (\*pixel)[(h - i - 1) \* w + j]);  
 }  
 }  
 }  
 **else if** (type == 6)  
 {  
 **for** (**int** i = 0; i < h / 2; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 swap((\*pixel)[(i \* w + j) \* 3], (\*pixel)[((h - i - 1) \* w + j) \* 3]);  
 swap((\*pixel)[(i \* w + j) \* 3 + 1], (\*pixel)[((h - i - 1) \* w + j) \* 3 + 1]);  
 swap((\*pixel)[(i \* w + j) \* 3 + 2], (\*pixel)[((h - i - 1) \* w + j) \* 3 + 2]);  
 }  
 }  
 }  
}  
  
  
**void** rightRotate(**int** type, **int** w, **int** h, uchar\*\* pixel)  
{  
 **if** (type == 5)  
 {  
 uchar \*temp;  
 temp = (uchar\*) malloc(**sizeof**(uchar) \* h \* w);  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 temp[j \* h + (h - i - 1)] = (\*pixel)[i \* w + j];  
 }  
 }  
 free(\*pixel);  
 \*pixel = temp;  
 }  
 **else if** (type == 6)  
 {  
 uchar \*temp;  
 temp = (uchar\*) malloc(**sizeof**(uchar) \* h \* w \* 3);  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 temp[(j \* h + (h - i - 1)) \* 3] = (\*pixel)[(i \* w + j) \* 3];  
 temp[(j \* h + (h - i - 1)) \* 3 + 1] = (\*pixel)[(i \* w + j) \* 3 + 1];  
 temp[(j \* h + (h - i - 1)) \* 3 + 2] = (\*pixel)[(i \* w + j) \* 3 + 2];  
 }  
 }  
 free(\*pixel);  
 \*pixel = temp;  
 }  
}  
  
  
**void** leftRotate(**int** type, **int** w, **int** h, uchar\*\* pixel)  
{  
 **if** (type == 5)  
 {  
 uchar \*temp;  
 temp = (uchar\*) malloc(**sizeof**(uchar) \* h \* w);  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 temp[(w - j - 1) \* h + i] = (\*pixel)[i \* w + j];  
 }  
 }  
 free(\*pixel);  
 \*pixel = temp;  
 }  
 **else if** (type == 6)  
 {  
 uchar \*temp;  
 temp = (uchar\*) malloc(**sizeof**(uchar) \* h \* w \* 3);  
 **for** (**int** i = 0; i < h; i++)  
 {  
 **for** (**int** j = 0; j < w; j++)  
 {  
 temp[((w - j - 1) \* h + i) \* 3] = (\*pixel)[(i \* w + j) \* 3];  
 temp[((w - j - 1) \* h + i) \* 3 + 1] = (\*pixel)[(i \* w + j) \* 3 + 1];  
 temp[((w - j - 1) \* h + i) \* 3 + 2] = (\*pixel)[(i \* w + j) \* 3 + 2];  
 }  
 }  
 free(\*pixel);  
 \*pixel = temp;  
 }  
}  
  
  
  
**int** main(**int** argc, **char** \*argv[])  
{  
 **int** type, w, h, dummy;  
 **char** task;  
 uchar \*pixel;  
  
 **int** result = readFile(argc, argv,type,w,h,dummy, &pixel);  
  
 **if** (result == 0)  
 {  
 **char** \*fileName\_out = argv[2];  
 **if** (fileName\_out == **NULL**)  
 {  
 cout << **"Invalid output file"**;  
 free(pixel);  
 **return** 7;  
 }  
  
 task = argv[3][0];  
 **if** ((task != **'0'** && task != **'1'** && task != **'2'** && task != **'3'** && task != **'4'**) || strlen(argv[3]) != 1)  
 {  
 cout << **"Invalid request"**;  
 free(pixel);  
 **return** 8;  
 }  
  
 FILE \*f\_out = fopen(fileName\_out,**"wb"**);  
 **if** (f\_out == **NULL**)  
 {  
 cout << **"Allocation memory failed"**;  
 free(pixel);  
 **return** 9;  
 }  
  
 **if** (task == **'0'**)  
 {  
 fprintf(f\_out, **"P%i\n%i %i\n%i\n"**, type, w, h, dummy);  
 inversion(type, w, h, &pixel);  
 }  
 **else if** (task == **'1'**)  
 {  
 fprintf(f\_out, **"P%i\n%i %i\n%i\n"**, type, w, h, dummy);  
 horizontalShow(type, w, h, &pixel);  
 }  
 **else if** (task == **'2'**)  
 {  
 fprintf(f\_out, **"P%i\n%i %i\n%i\n"**, type, w, h, dummy);  
 verticalShow(type, w, h, &pixel);  
 }  
 **else if** (task == **'3'**)  
 {  
 fprintf(f\_out, **"P%i\n%i %i\n%i\n"**, type, h, w, dummy);  
 rightRotate(type, w, h, &pixel);  
 }  
 **else if** (task == **'4'**)  
 {  
 fprintf(f\_out, **"P%i\n%i %i\n%i\n"**, type, h, w, dummy);  
 leftRotate(type, w, h, &pixel);  
 }  
  
 **int** size = 1;  
 **if** (type == 5)  
 {  
 size = 1;  
 }  
 **else if** (type == 6)  
 {  
 size = 3;  
 }  
 fwrite(pixel, **sizeof**(uchar), w \* h \* size, f\_out);  
 free(pixel);  
 fclose(f\_out);  
 cout << **"Successful"**;  
 **return** 0;  
 }  
 **else** {  
 **return** result;  
 }  
}