

**Министерство образования и науки Российской Федерации**  
федеральное государственное автономное образовательное  
учреждение высшего образования  
**Санкт-Петербургский исследовательский университет**  
**Информационных технологий, механики и оптики**  
Факультет информационных технологий и программирования

Дисциплина: компьютерная геометрия и графика

# **Отчет**

по лабораторной работе № 4  
*Изучение цветовых пространств*

Выполнила: студент гр. М3102  
Карпов Арсений Викторович  
Преподаватель: Скаков П.С.

Санкт-Петербург  
2020

**Цель работы:** реализовать программу, которая позволяет проводить преобразования между цветовыми пространствами.

## Описание работы

Программа должна быть написана на C/C++ и не использовать внешние библиотеки.

Аргументы передаются через командную строку:

**lab4.exe -f <from\_color\_space> -t <to\_color\_space> -i <count> <input\_file\_name> -o <count> <output\_file\_name>**,

где

- <color\_space> - RGB / HSL / HSV / YCbCr.601 / YCbCr.709 / YCoCg / CMY
- <count> - 1 или 3
- <file\_name>:
  - для count=1 просто имя файла; формат ppm
  - для count=3 шаблон имени вида <name.ext>, что соответствует файлам <name\_1.ext>, <name\_2.ext> и <name\_3.ext> для каждого канала соответственно; формат pgm

Порядок аргументов (-f, -t, -i, -o) может быть произвольным.

Везде 8-битные данные и полный диапазон (**0..255, PC range**).

**Полное решение:** всё работает + корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок.

/\* да, частичного решения нет \*/

Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.

Коды возврата:

0 - ошибок нет

1 - произошла ошибка

В поток вывода ничего не выводится (printf, cout).

Сообщения об ошибках выводятся в поток вывода ошибок:

C: fprintf(stderr, "Error\n");

C++: std::cerr

Следующие параметры гарантировано не будут выходить за обусловленные значения:

- <count> = 1 или 3;
- width и height в файле - положительные целые значения;
- яркостных данных в файле ровно width \* height;

## Теоретическая часть

Цветовые пространства соответствуют различным системам представления информации о цвете.

Так как в соответствии с моделью зрения человека мы имеем 3 вида рецепторов, отвечающих за цветное зрение, то и для кодирования информации о цвете разумно использовать трёхмерное цветовое пространство.

Переход от одного цветового пространства к другому можно представить себе как изменение базиса системы координат: значения меняются, но информация остаётся.

В данной лабораторной, производится работа с 6 различными цветовыми пространствами:

1. **RGB** (англ. *Red, Green, Blue*) - цветовая модель, описывающая способ кодирования цвета с помощью трёх основных цветов.
2. **HSV** (англ. *Hue, Saturation, Value*) - цветовая модель, в которой координатами цвета являются тон, насыщенность и значение цвета.

### RGB → HSV [\[ править \]](#) [\[ править код \]](#)

Считаем, что:

$$H \in [0, 360]$$

$$S, V, R, G, B \in [0, 1]$$

Пусть  $MAX$  — максимальное значение из  $R, G$  и  $B$ , а  $MIN$  — минимальное из них.

$$H = \begin{cases} 0, & \text{если } MAX = MIN \\ 60 \times \frac{G - B}{MAX - MIN} + 0, & \text{если } MAX = R \text{ и } G \geq B \\ 60 \times \frac{G - B}{MAX - MIN} + 360, & \text{если } MAX = R \text{ и } G < B \\ 60 \times \frac{B - R}{MAX - MIN} + 120, & \text{если } MAX = G \\ 60 \times \frac{R - G}{MAX - MIN} + 240, & \text{если } MAX = B \end{cases}$$

$$S = \begin{cases} 0, & \text{если } MAX = 0; \\ 1 - \frac{MIN}{MAX}, & \text{иначе} \end{cases}$$

$$V = MAX$$

### HSV → RGB [\[ править \]](#) [\[ править код \]](#)

Для любых оттенков  $H \in [0, 360]$ , насыщенности  $S \in [0, 100]$  и яркости  $V \in [0, 100]$ :

$$H_i = \left\lfloor \frac{H}{60} \right\rfloor \bmod 6$$

$$V_{min} = \frac{(100 - S) * V}{100}$$

$$a = (V - V_{min}) * \frac{H \bmod 60}{60}$$

$$V_{inc} = V_{min} + a$$

$$V_{dec} = V - a$$

$H_i$	R	G	B
0	$V$	$V_{inc}$	$V_{min}$
1	$V_{dec}$	$V$	$V_{min}$
2	$V_{min}$	$V$	$V_{inc}$
3	$V_{min}$	$V_{dec}$	$V$
4	$V_{inc}$	$V_{min}$	$V$
5	$V$	$V_{min}$	$V_{dec}$

### 3. HSL - цветовая модель, в которой цветовыми координатами являются тон, насыщенность и светлота

Конвертация из RGB [\[править\]](#) [\[править код\]](#)

$$H = \begin{cases} \text{undefined} & \text{if } MAX = MIN \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 0^\circ, & \text{if } MAX = R \\ & \text{and } G \geq B \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ, & \text{if } MAX = R \\ & \text{and } G < B \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ, & \text{if } MAX = G \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ, & \text{if } MAX = B \end{cases}, S = \begin{cases} 0 & \text{if } L = 0 \text{ or } MAX = MIN \\ \frac{MAX-MIN}{MAX+MIN} = \frac{MAX-MIN}{2L}, & \text{if } 0 < L \leq \frac{1}{2} \\ \frac{MAX-MIN}{2-(MAX+MIN)} = \frac{MAX-MIN}{2-2L}, & \text{if } \frac{1}{2} < L < 1 \end{cases}, \text{ или, в общем случае, } S = \frac{MAX-MIN}{1-|1-(MAX+MIN)|}$$

$L = \frac{1}{2} (MAX + MIN)$ , где:

- R, G, B — значения цвета в цветовой модели RGB, значения в диапазоне [0; 1] (R - красный, G - зелёный, B - синий).
- MAX — максимум из трёх значений (R, G, B)
- MIN — минимум из трёх значений (R, G, B)
- H — тон [0; 360]
- S — насыщенность [0; 1]

Конвертация в RGB [\[править\]](#) [\[править код\]](#)

$$Q = \begin{cases} L \times (1.0 + S), & \text{if } L < 0.5 \\ L + S - (L \times S), & \text{if } L \geq 0.5 \end{cases}$$

$$P = 2.0 \times L - Q$$

$$H_k = \frac{H}{360} \text{ (приведение к интервалу [0,1])}$$

$$T_R = H_k + \frac{1}{3}$$

$$T_G = H_k$$

$$T_B = H_k - \frac{1}{3}$$

$$\text{if } T_c < 0 \rightarrow T_c = T_c + 1.0 \quad \text{for each } c = R, G, B$$

$$\text{if } T_c > 1 \rightarrow T_c = T_c - 1.0 \quad \text{for each } c = R, G, B$$

Для каждого цвета  $c = R, G, B$ :

$$\text{color}_c = \begin{cases} P + ((Q - P) \times 6.0 \times T_c), & \text{if } T_c < \frac{1}{6} \\ Q, & \text{if } \frac{1}{6} \leq T_c < \frac{1}{2} \\ P + ((Q - P) \times (\frac{2}{3} - T_c) \times 6.0), & \text{if } \frac{1}{2} \leq T_c < \frac{2}{3} \\ P, & \text{otherwise} \end{cases}$$

### 4. YCbCr - семейство цветовых пространств, которые используются для передачи цветных изображений в компонентном видео и цифровой фотографии. Y — компонента яркости, Cb и Cr являются синей и красной цветоразностными компонентами.

**Значения коэффициентов для разных стандартов:**

	<b>Kr</b>	<b>Kb</b>
<b>601</b>	0.299	0.114
<b>709</b>	0.2126	0.0722

$$K_g = 1 - K_r - K_b$$

$$Y' = K_R \cdot R' + (1 - K_R - K_B) \cdot G' + K_B \cdot B'$$

$$P_B = \frac{1}{2} \cdot \frac{B' - Y'}{1 - K_B}$$

$$P_R = \frac{1}{2} \cdot \frac{R' - Y'}{1 - K_R}$$

5. **YCoCg** - цветовое пространство, образованное из простого преобразования цветового пространства RGB в значение яркости (Y) и два значения цветности, называемых цветность зеленого (Cg) и цветность оранжевого (Co).

**Формулы для перевода в RGB:**

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & 0 \\ 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} Y \\ Cg \\ Co \end{bmatrix}$$

**Формулы для перевода из RGB:**

$$\begin{bmatrix} Y \\ Cg \\ Co \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

**Диапазон значений, используемый в преобразованиях:**

$$0 \leq Y, R, G, B \leq 1$$

$$-0.5 \leq Co, Cg \leq 0.5$$

6. **СМУ** (англ. Cyan, Magenta, Yellow) - схема формирования цвета, использующая голубой, пурпурный и жёлтый цвета в роли основных.

**Формулы для перевода из RGB:**

$$C = 1 - R$$

$$M = 1 - G$$

$$Y = 1 - B$$

**Формулы для перевода в RGB:**

$$R = 1 - C$$

$$G = 1 - M$$

$$B = 1 - Y$$

**Диапазон значений, используемый в преобразованиях:**

$$0 \leq R, G, B \leq 1$$

$$0 \leq C, M, Y \leq 1$$

### Экспериментальная часть

Язык программирования: C++17.

Для начала, каждую цветовую компоненту записываем в отдельный массив, чтобы было удобнее преобразовывать.

После этого, переводим данные из указанного пространства в RGB с помощью одной из функций `*something*_RGB`. И затем, переводим из RGB в заданное пространство с помощью одной из функций `RGB_*something*`.

Если компонента может принимать отрицательные значения, т.е. ее диапазон  $[-a, a]$ , то перед записью в массив нужно привести диапазон к виду  $[0, 2*a]$ , прибавив к ее значению  $a$ , после чего использовать вышеуказанную формулу, а перед началом работы после применения формулы «перед работой», привести ее в первоначальный диапазон, отняв от значения  $a$  (в обеих формулах значение  $\text{max\_value} = 2 * a$ ).

После проведения преобразований между пространствами, записываем полученные результаты в выходной файл. Если указан один файл, то побайтово записываем первый, второй и третий массив по очереди. Если же нужно представить выходной результат в трех файлах, то каждый массив записываем в отдельный файл.

### Выводы

В ходе проделанной работы были изучены некоторые характеристики различных цветовых пространств, а также найдены формулы связи между RGB и другими цветовыми пространствами. В результате работы была реализована программа, выполняющая преобразования между заданными цветовыми пространствами.

Полученные результаты совпадают с теоретическими, т.к. после преобразования изображения из RGB в заданное цветовое пространство и последующего преобразования результата из этого же пространства обратно в RGB полученное в

результате изображение совпадает с исходным => в процессе преобразований не была нарушена целостность информации.

## Листинг

### transfer.cpp

```
//
// Created by Арсений Карпов on 30.05.2020.
//

#include "transfer.h"

transfer::transfer(int width, int height, double* first, double* second, double* third)
:_width(width), _height(height), _first(first), _second(second), _third(third) {
    base.resize(7);
    base[0] = "RGB";
    base[1] = "HSL";
    base[2] = "HSV";
    base[3] = "YCbCr.601";
    base[4] = "YCbCr.709";
    base[5] = "YCoCg";
    base[6] = "CMY";
}

double transfer::for_HSL_RGB(double T, double q, double p)
{
    T = (T < 0) ? T + 1 : (T > 1) ? T - 1 : T;
    if (T < double(1) / 6)
        return p + (q - p) * 6 * T;
    if (T < double(1) / 2)
        return q;
    if (T < double(2) / 3)
        return p + ((q - p) * (double(2) / 3 - T) * 6);
    return p;
}

void transfer::HSL_RGB()
{
    double Q, P, L, S, H;
```

```

for (int i = 0; i < _width * _height; i++)
{
    H = _first[i] / 255.0;
    S = _second[i] / 255.0;
    L = _third[i] / 255.0;
    Q = (L < 0.5) ? L * (1 + S) : (L + S - L * S);
    P = 2 * L - Q;
    _first[i] = for_HSL_RGB(H + 1.0 / 3.0, Q, P) * 255;
    _second[i] = for_HSL_RGB(H, Q, P) * 255;
    _third[i] = for_HSL_RGB(H - 1.0 / 3.0, Q, P) * 255;
}
}

```

```

void transfer::RGB_HSL()
{
    double _max, _min, H, S, L, r, g, b;
    for (int i = 0; i < _width * _height; i++)
    {
        r = _first[i] / 255.0;
        g = _second[i] / 255.0;
        b = _third[i] / 255.0;
        _max = std::max(r, std::max(g, b));
        _min = std::min(r, std::min(g, b));
        L = (_max + _min) / 2.0;
        if ((_max == _min) || (L == 0))
            S = 0;
        else
        {
            if (L <= 0.5)
                S = (_max - _min) / (2.0 * L);
            else
                S = (_max - _min) / (2.0 - 2.0 * L);
        }
        if (_max == _min)
            H = 0;
        else
        {
            if (_max == r)

```



```

{
    H = 60.0 * (g - b) / (_max - _min);
    if (g < b)
        H += 360;
}
if (_max == g)
    H = 60.0 * (b - r) / (_max - _min) + 120;
if (_max == b)
    H = 60.0 * (r - g) / (_max - _min) + 240;
}
_first[i] = H * 255 / 360.0;
_second[i] = S * 255;
_third[i] = L * 255;
}
}

```

```

void transfer::HSV_RGB()

```

```

{
    double H, S, V, Hi, Vmin, Vinc, Vdec, a;
    int H1;
    for (int i = 0; i < _width * _height; i++)
    {
        H = _first[i] / 255.0 * 360;
        S = _second[i] / 255.0;
        V = _third[i] / 255.0;
        Vmin = (1 - S) * V;
        Hi = (H / 60.0);
        H1 = int(Hi);
        a = (V - Vmin) * (Hi - H1);
        H1 = H1 % 6;
        Vinc = Vmin + a;
        Vdec = V - a;
        switch (H1)
        {
            case 0:
            {
                _first[i] = V * 255;
                _second[i] = Vinc * 255;

```

```
    _third[i] = Vmin * 255;
    break;
}
case 1:
{
    _first[i] = Vdec * 255;
    _second[i] = V * 255;
    _third[i] = Vmin * 255;
    break;
}
case 2:
{
    _first[i] = Vmin * 255;
    _second[i] = V * 255;
    _third[i] = Vinc * 255;
    break;
}
case 3:
{
    _first[i] = Vmin * 255;
    _second[i] = Vdec * 255;
    _third[i] = V * 255;
    break;
}
case 4:
{
    _first[i] = Vinc * 255;
    _second[i] = Vmin * 255;
    _third[i] = V * 255;
    break;
}
default:
{
    _first[i] = V * 255;
    _second[i] = Vmin * 255;
    _third[i] = Vdec * 255;
    break;
}
```

```

    }
}

void transfer::RGB_HSV()
{
    double S, V, r, g, b, H, _max, _min;
    for (int i = 0; i < _width*_height; i++)
    {
        r = _first[i] / 255.0;
        g = _second[i] / 255.0;
        b = _third[i] / 255.0;
        _max = std::max(r, std::max(g, b));
        _min = std::min(r, std::min(g, b));
        V = _max;
        if (_max == 0)
            S = 0;
        else
            S = 1 - double(_min) / _max;
        if (_max == _min)
            H = 0;
        else
        {
            if (_max == r)
            {
                H = 60.0 * (g - b) / (_max - _min);
                if (g < b)
                    H += 360;
            }
            if (_max == g)
                H = 60.0 * (b - r) / (_max - _min) + 120;
            if (_max == b)
                H = 60.0 * (r - g) / (_max - _min) + 240;
        }
        _first[i] = H * 255 / 360.0;
        _second[i] = S * 255;
        _third[i] = V * 255;
    }
}

```

```

}

void transfer::YCbCr_RGB(double Kb, double Kr)
{
    double Y, Cb, Cr;
    for (int i = 0; i < _width * _height; i++)
    {
        Y = _first[i] / 255.0;
        Cb = _second[i] / 255.0 * 2 - 1;
        Cr = _third[i] / 255.0 * 2 - 1;

        _first[i] = std::max(std::min((Y + Cr * (1 - Kr)) * 255, 255.0), 0.0);
        _second[i] = std::max(std::min((Y - double((Kb * (1 - Kb) * Cb + Cr * (1 - Kr) * Kr)) / (1 - Kr - Kb)) *
255, 255.0), 0.0);
        _third[i] = std::max(std::min((Y + Cb * (1 - Kb)) * 255, 255.0), 0.0);
    }
}

void transfer::RGB_YCbCr(double Kb, double Kr)
{
    double r, g, b;
    for (int i = 0; i < _width * _height; i++)
    {
        r = _first[i] / 255.0;
        g = _second[i] / 255.0;
        b = _third[i] / 255.0;

        _first[i] = Kr * r + (1 - Kr - Kb) * g + Kb * b;
        _second[i] = double(b - _first[i]) / (1 - Kb) + 1;
        _third[i] = double (r - _first[i]) / (1 - Kr) + 1;

        _first[i] = std::max(std::min(_first[i] * 255, 255.0), 0.0);
        _second[i] = std::max(std::min(_second[i] * 255 / 2, 255.0), 0.0);
        _third[i] = std::max(std::min(_third[i] * 255 / 2, 255.0), 0.0);
    }
}

void transfer::YCoCg_RGB()
{
    double Y, Co, Cg;

    for (int i = 0; i < _width * _height; i++)

```

```

{
    Y = _first[i] / 255.0;
    Co = _second[i] / 255.0 - 0.5;
    Cg = _third[i] / 255.0 - 0.5;
    _first[i] = std::max(std::min((Y + Co - Cg) * 255, 255.0), 0.0);
    _second[i] = std::max(std::min((Y + Cg) * 255, 255.0), 0.0);
    _third[i] = std::max(std::min((Y - Co - Cg) * 255, 255.0), 0.0);
}
}

void transfer::RGB_YCoCg()
{
    double r, g, b;
    for (int i = 0; i < _width * _height; i++)
    {
        r = _first[i] / 255.0;
        g = _second[i] / 255.0;
        b = _third[i] / 255.0;
        _first[i] = std::max(std::min((r / 4.0 + g / 2.0 + b / 4.0) * 255, 255.0), 0.0);
        _second[i] = std::max(std::min((0.5 * r - 0.5 * b + 0.5) * 255, 255.0), 0.0);
        _third[i] = std::max(std::min((- r / 4.0 + g / 2.0 - b / 4.0 + 0.5) * 255, 255.0), 0.0);
    }
}

void transfer::CMY_RGB()
{
    for (int i = 0; i < _width * _height; i++)
    {
        _first[i] = 255 - _first[i];
        _second[i] = 255 - _second[i];
        _third[i] = 255 - _third[i];
    }
}

int transfer::convert(std::string from, std::string to)
{
    int i = 0;
    while ((from != base[i]) && (i < 7))

```

```
    i++;  
    switch (i) {  
    case 0:  
        break;  
    case 1:  
    {  
        HSL_RGB();  
        break;  
    }  
    case 2:  
    {  
        HSV_RGB();  
        break;  
    }  
    case 3:  
    {  
        YCbCr_RGB(0.114, 0.299);  
        break;  
    }  
    case 4:  
    {  
        YCbCr_RGB(0.0722, 0.2126);  
        break;  
    }  
    case 5:  
    {  
        YCoCg_RGB();  
        break;  
    }  
    case 6:  
    {  
        CMY_RGB();  
        break;  
    }  
    default:  
    {  
        return 1;  
    }  
    }
```

```
}  
  
i = 0;  
while ((to != base[i]) && (i < 7))  
    i++;  
switch (i) {  
case 0:  
    break;  
case 1:  
{  
    RGB_HSL();  
    break;  
}  
case 2:  
{  
    RGB_HSV();  
    break;  
}  
case 3:  
{  
    RGB_YCbCr(0.114, 0.299);  
    break;  
}  
case 4:  
{  
    RGB_YCbCr(0.0722, 0.2126);  
    break;  
}  
case 5:  
{  
    RGB_YCoCg();  
    break;  
}  
case 6:  
{  
    CMY_RGB();  
    break;  
}  
default:
```

```

{
    return 1;
}
}
return 0;
}

```

## transfer.h

```

//
// Created by Арсений Карпов on 30.05.2020.
//

#ifndef KGIG4_TRANSFER_H
#define KGIG4_TRANSFER_H

#include <string>
#include <vector>
#include <algorithm>

class transfer {
private:
    double* _first{};
    double* _second{};
    double* _third{};
    int _width;
    int _height;
    std::vector<std::string> base;
    double for_HSL_RGB(double T, double q, double p);
    void HSL_RGB();
    void RGB_HSL();
    void HSV_RGB();
    void RGB_HSV();
    void YCbCr_RGB(double Kb, double Kr);
    void RGB_YCbCr(double Kb, double Kr);
    void YCoCg_RGB();
    void RGB_YCoCg();
    void CMY_RGB();
public:
    transfer(int width, int height, double* first, double* second, double* third);

```



```

        int convert(std::string from, std::string to);
};

#endif //KGIG4_TRANSFER_H

```

## main.cpp

```

#include <iostream>
#include "transfer.h"
#include <cmath>
#include <string.h>
int count_i, count_o, num, numo;
double* first, * second, * third;
typedef unsigned char uchar;
class Exception
{
public:
    Exception(std::string error)
    {
        error_ = error;
    }
    std::string out_error()
    {
        return error_;
    }
private:
    std::string error_;
};

char* create_name(char* name,int num)
{
    char* new_name;
    int j = strlen(name);
    while ((name[j] != '.') && (j > -1))
        j--;
    new_name = new char[100];
    for (int i = 0;i < 100; i++)
        new_name[i] = '\0';
}

```

```

    strncat(new_name, name, j);

    switch (num)
    {
    case 1: {
        strcat(new_name, "_1.pgm");
        break;
    }
    case 2:
    {
        strcat(new_name, "_2.pgm");
        break;
    }
    default:
    {
        strcat(new_name, "_3.pgm");
        break;
    }
    }
    strcat(new_name, "\0");
    return new_name;
}

int input_files(char* in, int &width, int &height, int &bpp)
{
    if (count_i==1) {
        FILE* fin = fopen(in, "rb");
        if (fin==NULL)
            return 1;

        int type;
        int i = fscanf(fin, "P%d", &type);
        if ((type!=6) || (i<1)) {
            fclose(fin);
            return 2;
        }
        i = fscanf(fin, "\n%d %d\n%d\n", &width, &height, &bpp);
        if ((i<3) || (bpp<0) || (bpp>255)) {
            fclose(fin);
            return 2;
        }
    }
}

```

```

}

uchar* p = (uchar*) malloc(3*width*height*sizeof(uchar));

int k = fread(p, 1, 3*width*height, fin);

if (k < 1) {
    fclose(fin);
    return 2;
}

first = new double[width*height];
second = new double[width*height];
third = new double[width*height];

for (i = 0; i < width*height; i++) {
    first[i] = (double) *(p + i * 3);
    second[i] = (double) *(p + i * 3 + 1);
    third[i] = (double) *(p + i * 3 + 2);
}

free(p);
fclose(fin);
return 0;
}

int width1, height1;

for (int i = 1; i < 4; i++)
{
    char* new_name;

    new_name = create_name(in, i);
    FILE* fin = fopen(new_name, "rb");
    if (fin == NULL)
        return 1;

    int type;
    int it = fscanf(fin, "P%d", &type);
    if ((type != 5) || (it < 1)) {
        fclose(fin);
        return 2;
    }

    it = fscanf(fin, "\n%d %d\n%d\n", &width, &height, &bpp);
    if (i == 1) {
        width1 = width;
        height1 = height;
    }
}

```

```

if ((it < 3) || (bpp < 0) || (bpp > 255) || (width1 != width) || (height1 != height)) {
    fclose(fin);
    return 2;
}

switch (i)
{
case 1:
{
    first = new double[width * height];
    int k = 0;
    uchar* p = (uchar*) malloc(3*width*height*sizeof(uchar));
    k = fread(p, 1, width * height, fin);
    for (int i = 0; i < width * height; i++)
        first[i] = (double)(*(p + i));
    free(p);
    break;
}

case 2:
{
    second = new double[width * height];
    uchar* p = (uchar*) malloc(3*width*height*sizeof(uchar));
    int k = fread(p, 1, width * height, fin);
    for (int i = 0; i < width * height; i++)
        second[i] = (double)(*(p + i));
    free(p);
    break;
}

default:
{
    third = new double[width * height];
    uchar* p = (uchar*) malloc(3*width*height*sizeof(uchar));
    int k = fread(p, 1, width * height, fin);
    for (int i = 0; i < width * height; i++)
        third[i] = (double)(*(p + i));
    free(p);
    break;
}
}

```

```

    }

    fclose(fin);

    width1 = width;
    height1 = height;
}

return 0;
}

int output_files(char* out, int width, int height, int bpp)
{
    if (count_o == 1)
    {
        FILE* fout = fopen(out, "wb");
        if (fout == NULL)
            return 1;

        uchar u;
        fprintf(fout, "P6\n%d %d\n%d\n", width, height, bpp);
        for (int i = 0; i < width*height; i++) {
            u = (uchar) first[i];
            fwrite(&u, 1, 1, fout);
            u = (uchar) second[i];
            fwrite(&u, 1, 1, fout);
            u = (uchar) third[i];
            fwrite(&u, 1, 1, fout);
        }
        delete(first);
        delete(second);
        delete(third);
        fclose(fout);
        return 0;
    }

    for (int i = 1; i < 4; i++)
    {
        char * new_name = create_name(out, i);
        FILE* fout = fopen(new_name, "wb");
        if (fout == NULL)
            return 1;

        uchar u;
        fprintf(fout, "P5\n%d %d\n%d\n", width, height, bpp);
    }
}

```

```

switch (i)
{
case 1:
{
for (int j = 0; j<width*height; j++)
{
u = (uchar) first[j];
fwrite(&u, 1, 1, fout);
}
delete(first);
break;
}
case 2:
{
for (int j = 0; j<width*height; j++)
{
u = (uchar) second[j];
fwrite(&u, 1, 1, fout);
}
delete(second);
break;
}
default:
{
for (int j = 0; j<width*height; j++)
{
u = (uchar) third[j];
fwrite(&u, 1, 1, fout);
}
delete(third);
break;
}
}
fclose(fout);
}
return 0;
}
int main(int argc, char* argv[])

```

```

{
    try {
        if (argc != 11)
            throw Exception("Incorrect nubmer of arguments");

        std::vector<bool> used(4, false);
        std::vector<std::string> commands(4);

        commands[0] = "-f";
        commands[1] = "-t";
        commands[2] = "-i";
        commands[3] = "-o";

        std::string from, to;
        int i = 1;
        while (i < 10)
        {
            int j = 0;
            while ((j < 4) && (commands[j] != argv[i]))
                j++;

            if ((used[j]) || (j == 4))
                throw Exception("Incorrect command");

            i++;

            switch (j)
            {
            {
            case 0:
            {
                from = argv[i];
                break;
            }
            case 1:
            {
                to = argv[i];
                break;
            }
            case 2:
            {
                count_i = atoi(argv[i]);
                i++;

                num = i;

                break;
            }
            }
            }

```

```

    }
    default:
    {
        count_o = atoi(argv[i]);
        i++;
        numo = i;
        break;
    }
}
i++;
}
int width, height, bpp;
if (input_files(argv[num], width, height, bpp) != 0)
    throw Exception("Problems with input file");
uchar u;
transfer clas(width, height, first, second, third);
if (clas.convert(from, to) != 0)
    throw Exception("Incorrect colorspace");
if (output_files(argv[numo], width, height, bpp) != 0)
    throw Exception("Problems with output file");
}
catch (Exception &ex)
{
    std::cerr<<ex.out_error();
    delete(first);
    delete(second);
    delete(third);
    return 1;
}
return 0;
}

```