

Sprawozdanie

Adam Stypczyc, 259273

11 stycznia 2021

Zadanie 5 przetwarzanie obrazów 2

Spis treści

1	Wstęp	3
2	Ważne fragmenty kodu programu	5
2.1	Struktura	5
2.2	Alokacja pamięci	6
2.3	Funkcja progowania czerni	7
2.4	Funkcja progowania bieli	8
2.5	Funkcja zamiany poziomów	9
2.6	Funkcja konwersji do szarości	11
2.7	Funkcja rozmazywania poziomego o promieniu $r = 1$	12
2.8	Funkcja rozmazywania poziomego o promieniu $r = 2$	13
2.9	Funkcja zamiany koloru	15
3	Testy programu	16
3.1	Co dane testy ma na celu	16
3.1.1	Test 1	16
3.1.2	Test 2	16
3.1.3	Test 3	16
3.1.4	Test 4	16
3.1.5	Test 5	16
3.1.6	Test 6	17
3.1.7	Test 7	17
3.1.8	Test 8	17

3.1.9	Test 9	17
3.1.10	Test 10	17
3.1.11	Test 11	17
3.2	Metoda testowania	18
3.3	Dane wyjściowe zwrócone w teście przez program	18
3.3.1	Test 1	18
3.3.2	Test 2	19
3.3.3	Test 3	19
3.3.4	Test 4	20
3.3.5	Test 5	20
3.3.6	Test 6	21
3.3.7	Test 7	21
3.3.8	Test 8	22
3.3.9	Test 9	23
3.3.10	Test 10	23
3.3.11	Test 11	24
4	Wnioski końcowe	24

Listings

1	Struktura	5
2	Alokacja pamięci	6
3	Progowanie czerni	7
4	Progowanie bieli	8
5	Zamiana poziomów	9
6	Konwersja do szarości	11
7	1. Rozmazywanie poziome	12
8	2. Rozmazywanie poziome	13
9	Zamiana koloru	15

1 Wstęp

Zadaniem, które musiałem wykonać, było napisanie kolejnej części przetwarzania obrazów. Od zadania, które musiałem wykonać poprzednio, różni się ono przede wszystkim tym, że przetwarza dodatkowo obrazy w formacie PPM, dynamicznie alokuje pamięć, przechowuje informacje o obrazie w strukturze danych oraz ma dużo więcej opcji przetwarzania. Ponadto nie posiada menu, zamiast tego wszystkie informacje, które są potrzebne do przetworzenia obrazu, podajemy w tym momencie, kiedy wywołujemy funkcję. Problemów jakie napotkałem podczas tworzenia tego programu było wiele, zaczynając chociażby od błędu w alokacji pamięci. Na szczęście błąd nie okazał się trudny do naprawienia, jednakże jego znalezienie zajęło około 2 godziny. Okazało się, że omyłkowo przy alokowaniu pamięci dla obrazów PPM wpisałem zmienną „obraz_pgm” zamiast „obraz_ppm”. Prawie zapomniałem o pierwszym problemie jaki napotkałem, a mianowicie było nim wymyślenie jak zdefiniować obraz w formacie PPM, było to problematyczne, ponieważ posiada on kolory i jednemu pikselowi obrazu odpowiadają trzy wartości, a nie jedna tak jak w obrazie PGM. Problem ten rozwiązałem definiując obrazy PPM w tablicy trójwymiarowej o głębi odpowiadającej trzem elementom. Elementy o głębi równej 0 odpowiadają odcieniowi czerwonego, 1 odpowiadającej odcieniom zieleni, 2 odpowiadającej odcieniom koloru niebieskiego. Utworzenie kolejnych funkcji przetwarzania obrazu nie było problematyczne, ponieważ napisałem je w poprzedniej części tego zadania, a następnie udoskonaliłem je tak, aby pracowały na dynamicznych tablicach, strukturze oraz na obrazach PPM. Warto również wspomnieć, że podczas usprawniania funkcji histogramu, zauważyłem błąd, zamiast maksymalnej wartości „szarosci” obrazu wczytywałem MAX, a to odpowiadało maksymalnemu rozmiarowi obrazu w poprzedniej części tego zadania. Konwersja obrazu PPM do PGM była w mojej opinii najbardziej problematyczna, bo trzeba było zastanowić się nad poprawnym zwalnianiem pamięci. Plik main.c oparty jest na pliku opcje.c dostarczonym przez doktora Muszyńskiego. Zmodyfikowałem go w taki sposób, że dodałem do niego funkcje, których nie było wcześniej w tym pliku. Dodałem również odpowiednie flagi do funkcji. Zgadza się one w większości z tym co dostaliśmy, różnią się one w progowaniu czerni(flaga -c), progowaniu bieli(flaga -b), rozmywaniu pionowym (flaga -y) i poziomym (flaga -x) o promieniu równym 1. Od siebie również dodałem roz-

mywanie o promieniu $r=2$ (pionowe flaga -Y, poziome flaga -X) oraz zamianę obrazu PPM (-K) na kolor czerwony, zielony bądź niebieski, aby ta zamiana działała poprawnie musimy podać r (red), g (green) lub b (blue), zależy to od koloru na jaki chcemy zmienić obraz PPM, po flagie -m. Musimy również pamiętać aby przy przetwarzaniu obrazów PPM podawać po flagie -m kolor, bez tego otrzymamy błąd ochrony pamięci. Problem również miałem z funkcją zamiany poziomów, bo początkowo nie czytało drugiej wartości (poziom bieli), jednakże po krótkiej konsultacji okazało się, że wystarczy dodać po skanowaniu czerni jedną instrukcję warunkową `if(++i<argc)`. Potem wszystko działało poprawnie. Jedną z większych komplikacji jaką napotkałem było poprawne wywołanie funkcji „wyswietl”, która znajdowała się w funkcji main, a nazwa pliku wyjściowego, który miał być wyświetlany znajdowała się w funkcji „przetwarzaj_opcje”. Rozwiązałem ten problem dodając do struktury „t_opcje” zmienne „nazwa_pliku_we” oraz „nazwa_pliku_wy”. Potem bez problemu mogłem wyświetlić nowo powstały plik. W kodach programu podanych w sprawozdaniu zamieszczę nowe funkcje, które będą w stanie przetwarzać obrazy PPM. Funkcje rozmazywania zamieszczę tylko dla poziomu, ponieważ funkcje rozmazywania pionu są analogiczne.

2 Ważne fragmenty kodu programu

2.1 Struktura

```
1 #ifndef STRUKTURA_H
2 #define STRUKTURA_H
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdlib.h>
6 #include <math.h>
7 #define MAX 512
8 struct Obraz
9 {
10     char magiczna_liczba;
11     int ***obraz_ppm;
12     int **obraz_pgm;
13     int wymx;
14     int wymy;
15     int szarosci;
16 };
17 typedef struct Obraz Obraz;
18 #endif
```

Listing 1: Struktura

2.2 Alokacja pamięci

```
1  if(Obraz->magiczna_liczba=='2'){
2      Obraz->obraz_pgm=malloc(Obraz->wymy*sizeof(int*));
3      for(i=0; i<Obraz->wymy; i++){
4          Obraz->obraz_pgm[i]=malloc(Obraz->wymx*sizeof(int));
5      }
6      for (i=0;i<Obraz->wymy;i++){
7          for (j=0;j<Obraz->wymx;j++){
8              if (fscanf(plik_we,"%d",&(Obraz->obraz_pgm[i][j]))!=1){
9                  fprintf(stderr,"Blad: Niewlasciwe wymiary obrazu\n");
10                 return(0);
11             }
12         }
13     }
14 }
15 else if(Obraz->magiczna_liczba=='3'){
16     Obraz->obraz_ppm=malloc(Obraz->wymy*sizeof(int**));
17     for(i=0; i<Obraz->wymy; i++){
18         Obraz->obraz_ppm[i]=malloc(Obraz->wymx*sizeof(int*));
19         for(j=0; j<Obraz->wymx; j++){
20             Obraz->obraz_ppm[i][j]=malloc(3*sizeof(int));
21         }
22     }
23     for (i=0;i<Obraz->wymy;i++){
24         for (j=0;j<Obraz->wymx;j++){
25             for(k=0; k<3; k++){
26                 if (fscanf(plik_we,"%d",&(Obraz->obraz_ppm[i][j][k]))
27                 !=1){
28                     fprintf(stderr,"Blad: Niewlasciwe wymiary obrazu\n"
29                     );
30                     return(0);
31                 }
32             }
33         }
34     }
35 }
```

Listing 2: Alokacja pamięci

2.3 Funkcja progowania czerni

```
1 void progowanie_czerni_PPM(float proc_pc, Obraz *Obraz, char
   kolor)
2 {
3     int i,j,k;
4     if(kolor=='r'){
5         k=0;
6     }
7     else if(kolor=='g'){
8         k=1;
9     }
10    else if(kolor=='b'){
11        k=2;
12    }
13    int prog;
14    prog=Obraz->szarosci*proc_pc/100;
15    if((k==0)|(k==1)|(k==2)){
16        for(i=0;i<Obraz->wymy;i++){
17            for(j=0;j<Obraz->wymx;j++){
18                if(Obraz->obraz_ppm[i][j][k]<=prog){
19                    Obraz->obraz_ppm[i][j][k]=0;
20                }
21                else{
22                    Obraz->obraz_ppm[i][j][k]=Obraz->obraz_ppm[i][j][k];
23                }
24            }
25        }
26    }
27    else{
28        for(k=0; k<3; k++){
29            w kodzie jest tu taki sam kod jak w if((k==0)|(k==1)|(k
   ==2)){}
30        }
31    }
32 }
```

Listing 3: Progowanie czerni

2.4 Funkcja progowania bieli

```
1 void progowanie_bieli_PPM(float proc_pb, Obraz *Obraz, char
   kolor)
2 {
3     int i,j,k;
4     if(kolor=='r'){
5         k=0;
6     }
7     else if(kolor=='g'){
8         k=1;
9     }
10    else if(kolor=='b'){
11        k=2;
12    }
13    int prog;
14    /
15    prog=Obraz->szarosci*proc_pb/100;
16    if((k==0)|(k==1)|(k==2)){
17        for(i=0;i<Obraz->wymy;i++){
18            for(j=0;j<Obraz->wymx;j++){
19                if(Obraz->obraz_ppm[i][j][k]<=prog){
20                    Obraz->obraz_ppm[i][j][k]=Obraz->obraz_ppm[i][j][k];
21                }
22                else{
23                    Obraz->obraz_ppm[i][j][k]=Obraz->szarosci;
24                }
25            }
26        }
27    }
28    else{
29        for(k=0; k<3; k++){
30            w kodzie jest tu taki sam kod jak w if((k==0)|(k==1)|(k
31            ==2)){}
32        }
33    }
```

Listing 4: Progowanie bieli

2.5 Funkcja zamiany poziomów

```
1 void zamiana_poziomow_PPM(float proc_czern, float proc_biel,
   Obraz *Obraz, char kolor)
2 {
3     int i,j,k;
4     if(kolor=='r')
5     {
6         k=0;
7     }
8     else if(kolor=='g')
9     {
10        k=1;
11    }
12    else if(kolor=='b')
13    {
14        k=2;
15    }
16    float czern;
17    float biel;
18    czern=Obraz->szarosci*proc_czern/100;
19    biel=Obraz->szarosci*proc_biel/100;
20    if((k==0)|(k==1)|(k==2))
21    {
22        for(i=0;i<Obraz->wymy;i++)
23        {
24            for(j=0;j<Obraz->wymx;j++)
25            {
26                if(Obraz->obraz_ppm[i][j][k]<=czern)
27                {
28                    Obraz->obraz_ppm[i][j][k]=0;
29                }
30                else if(Obraz->obraz_ppm[i][j][k]>=biel)
31                {
32                    Obraz->obraz_ppm[i][j][k]=Obraz->szarosci;
33                }
34                else if(czern<Obraz->obraz_ppm[i][j][k] && Obraz->
   obraz_ppm[i][j][k]<biel)
35                {
36                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j][k]-
   czern)*Obraz->szarosci/abs(biel-czern);
37                }
38                else if(Obraz->obraz_ppm[i][j][k]>biel && czern>Obraz->
   obraz_ppm[i][j][k])
```

```

39         {
40             Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j][k]-
41             czern)*Obraz->szarosci/abs(biel-czern);
42         }
43     }
44 }
45 else
46 {
47     for(k=0; k<3; k++)
48     {
49         w kodzie jest tu taki sam kod jak w if((k==0)|(k==1)|(k
50         ==2)){
51     }
52 }

```

Listing 5: Zamiana poziomów

2.6 Funkcja konwersji do szarości

```
1 void konwersja_do_szarosci(Obraz *Obraz)
2 {
3     int i,j,k;
4
5     Obraz->obraz_pgm=malloc(Obraz->wymy*sizeof(int*));
6     for(i=0;i<Obraz->wymy;i++)
7     {
8         Obraz->obraz_pgm[i]=malloc(Obraz->wymx*sizeof(int));
9         Obraz->magiczna_liczba='2';
10    }
11    for(i=0; i<Obraz->wymy;i++)
12    {
13        for(j=0; j<Obraz->wymx; j++)
14        {
15            Obraz->obraz_pgm[i][j]=(Obraz->obraz_ppm[i][j][0]+Obraz->
16            obraz_ppm[i][j][1]+Obraz->obraz_ppm[i][j][2])/3;
17        }
18    }
19    for(i=0;i<Obraz->wymy;i++)
20    {
21        for(j=0;j<Obraz->wymx;j++)
22        {
23            free(Obraz->obraz_ppm[i][j]);
24        }
25    }
26    for(i=0;i<Obraz->wymy;i++)
27    {
28        free(Obraz->obraz_ppm[i]);
29    }
30    free(Obraz->obraz_ppm);
31 }
```

Listing 6: Konwersja do szarości

2.7 Funkcja rozmazywania poziomego o promieniu $r = 1$

```
1 void roz_poziome1_PPM(Obraz *Obraz, char kolor)
2 {
3     int i,j,k;
4     if(kolor=='r'){
5         k=0;
6     }
7     else if(kolor=='g'){
8         k=1;
9     }
10    else if(kolor=='b'){
11        k=2;
12    }
13    if((k==0)|(k==1)|(k==2)){
14        for(i=0;i<Obraz->wymy;i++){
15            for(j=0;j<Obraz->wymx;j++){
16                if(j==0){
17                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][Obraz
18                    ->wymx-1][k]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][
19                    j+1][k])/3;
20                }
21                else if(j==Obraz->wymx-1){
22                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j-1][k
23                    ]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][0][k])/3;
24                }
25                else{
26                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j-1][k
27                    ]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][j+1][k])/3;
28                }
29            }
30        }
31    }
32    else{
33        for(k=0; k<3; k++){
34            w kodzie jest tu taki sam kod jak w if((k==0)|(k==1)|(k
35            ==2)){}
36        }
37    }
38 }
```

Listing 7: 1. Rozmazywanie poziome

2.8 Funkcja rozmazywania poziomego o promieniu $r = 2$

```
1 void roz_poziome2_PPM(Obraz *Obraz, char kolor){
2     int i,j,k;
3     if(kolor=='r'){
4         k=0;
5     }
6     else if(kolor=='g'){
7         k=1;
8     }
9     else if(kolor=='b'){
10        k=2;
11    }
12    if((k==0) | (k==1) | (k==2)){
13        for(i=0; i<Obraz->wymy; i++){
14            for(j=0; j<Obraz->wymx; j++){
15                if(j==0){
16                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][Obraz->wymx-2][k]+Obraz->obraz_ppm[i][Obraz->wymx-1][k]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][j+1][k]+Obraz->obraz_ppm[i][j+2][k])/5;
17                }
18                else if(j==1){
19                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][Obraz->wymx-1][k]+Obraz->obraz_ppm[i][j-1][k]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][j+1][k]+Obraz->obraz_ppm[i][j+2][k])/5;
20                }
21                else if(j==Obraz->wymx-2){
22                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j-2][k]+Obraz->obraz_ppm[i][j-1][k]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][j+1][k]+Obraz->obraz_ppm[i][0][k])/5;
23                }
24                else if(j==Obraz->wymx-1){
25                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j-2][k]+Obraz->obraz_ppm[i][j-1][k]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][0][k]+Obraz->obraz_ppm[i][1][k])/5;
26                }
27                else{
28                    Obraz->obraz_ppm[i][j][k]=(Obraz->obraz_ppm[i][j-2][k]+Obraz->obraz_ppm[i][j-1][k]+Obraz->obraz_ppm[i][j][k]+Obraz->obraz_ppm[i][j+1][k]+Obraz->obraz_ppm[i][j+2][k])/5;
29                }
30            }
31        }
32    }
```

```

31     }
32 }
33 else{
34     for(k=0; k<3; k++){
35         w kodzie jest tu taki sam kod jak w if((k==0)|(k==1)|(k
36         ==2)){}
37     }
38 }

```

Listing 8: 2. Rozmazywanie poziome

2.9 Funkcja zamiany koloru

```
1 void zmiana_koloru(Obraz *Obraz, char kolor)
2 {
3     int i,j,k;
4     if(kolor=='r'){
5         k=0;
6     }
7     else if(kolor=='g'){
8         k=1;
9     }
10    else if(kolor=='b'){
11        k=2;
12    }
13    if(k==0){
14        for(i=0;i<Obraz->wmy;i++){
15            for(j=0;j<Obraz->wymx;j++){
16                Obraz->obraz_ppm[i][j][1]=0;
17                Obraz->obraz_ppm[i][j][2]=0;
18            }
19        }
20    }
21    else if(k==1){
22        for(i=0;i<Obraz->wmy;i++){
23            for(j=0;j<Obraz->wymx;j++){
24                Obraz->obraz_ppm[i][j][0]=0;
25                Obraz->obraz_ppm[i][j][2]=0;
26            }
27        }
28    }
29    else if(k==2){
30        for(i=0;i<Obraz->wmy;i++){
31            for(j=0;j<Obraz->wymx;j++){
32                Obraz->obraz_ppm[i][j][0]=0;
33                Obraz->obraz_ppm[i][j][1]=0;
34            }
35        }
36    }
37 }
```

Listing 9: Zamiana koloru

3 Testy programu

3.1 Co dane testy ma na celu

Testy mają na celu weryfikację poprawnego działania programu. Każdy test weryfikuje również poprawność alokacji pamięci.

3.1.1 Test 1

Test 1. sprawdza działanie funkcji, które były już wcześniej zawarte w przetwarzaniu obrazów 1. Są to funkcje: odczytu, zapisu, wyświetlania, negatywu, progowania, korekcji gamma, konturowania i rozciągania histogramu. Test ten jest wykonywany dla obrazów w formacie PGM i PPM. Obrazy PPM są przetwarzane odpowiednio dla koloru czerwonego, zielonego i niebieskiego. Obrazy zostały przetworzone przez negatyw, progowanie o progu równym 45%, korekcji gamma o gammie równej 2.5, konturowaniu i rozciągnięciu histogramu. Osobno również przeprowadziłem negatyw dla tych obrazów.

3.1.2 Test 2

Test 2. kontroluje poprawność funkcji progowania czerni dla progów 40% i 60%, sprawdza również niektóre z funkcji testu 1., ponieważ plik zostaje wczytany, przetworzony przez progowanie czerni, zapisany oraz wyświetlony.

3.1.3 Test 3

Test 3. analogicznie do testu 2. bada funkcję progowania bieli.

3.1.4 Test 4

Test 4. przygląda się działaniu programu, gdy chcemy wykonać zamianę poziomów. Funkcja ta została przetestowana dla wartości czerni równej 35 i bieli równej 55 i na odwrót. Prównanie zostało zamieszczone poniżej.

3.1.5 Test 5

Test 5. ustala, czy funkcja konwersji do szarości działa poprawnie. Funkcja ta może działa tylko dla obrazów w formacie PPM i konwertuje je do PGM. Z tego powodu nie testowałem funkcji dla obrazu PGM.

3.1.6 Test 6

Test 6. poddaje próbie funkcję rozmywania poziomego i pionowego o promieniu 2. Testów pozostałych rozmyć nie umieściłem w sprawozdaniu, ponieważ uznałem to za niepotrzebne, ponieważ są one do siebie bardzo analogiczne.

3.1.7 Test 7

Test 7. sprawdza co się stanie gdy będziemy chcieli zmienić kolor Kubusia na czerwony, zielony lub niebieski. Funkcję tą dodałem sam od siebie.

3.1.8 Test 8

Test 8. sprawdza co się stanie, gdy przy wczytywaniu pliku wpisujemy „-”. Zgodnie z tym co napisał doktor Muszyński obraz powinien zostać wczytany ze standardowego wejścia.

3.1.9 Test 9

Test 9. bada program, gdy przy wywoływaniu po odpowiednich flagach nie podamy pewnych informacji. Sprawdzone również zostało, czy nie zapisany plik może być wyświetlony, jak i również to co się stanie, gdy nie podamy flagi odpowiadającej za zapisanie.

3.1.10 Test 10

Test 10. sprawdza co się stanie, gdy przy wywoływaniu obrazu PPM nie podamy informacji w jakim kolorze ma być podany obraz.

3.1.11 Test 11

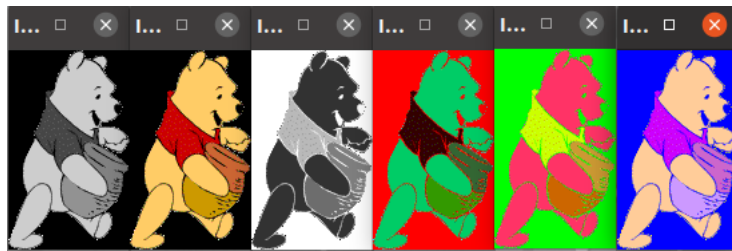
Test 11. bada program czy plik w innym formacie niż PPM lub PGM może być przetworzony.

3.2 Metoda testowania

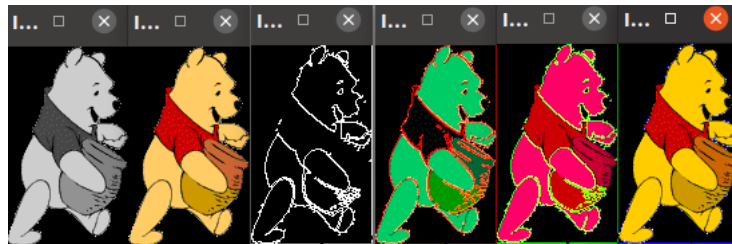
Program został przetestowany ręcznie przez wpisywanie odpowiednich flag przy wywołaniu programu.

3.3 Dane wyjściowe zwrócone w teście przez program

3.3.1 Test 1

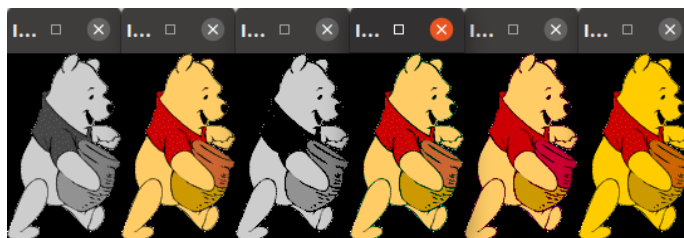


Rysunek 1: Porównanie oryginalnych obrazków z ich negatywami. Odpowiednio oryginał PGM, oryginał PPM, przetworzony PGM, przetworzony czerwony PPM, przetworzony zielony PPM, przetworzony niebieski PPM. W następnych porównaniach też będą w takiej kolejności.

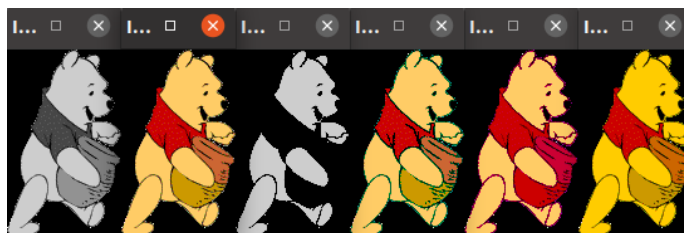


Rysunek 2: Porównanie oryginalnych obrazków z ich mocno przetworzonymi wersjami.

3.3.2 Test 2

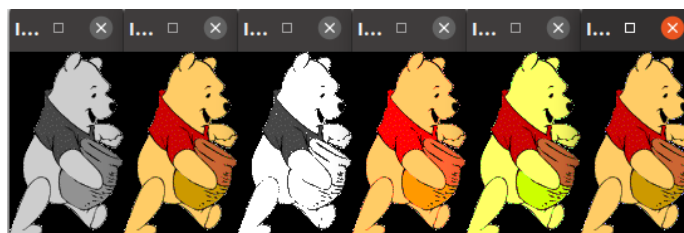


Rysunek 3: Porównanie progowania czerni o progu 40.

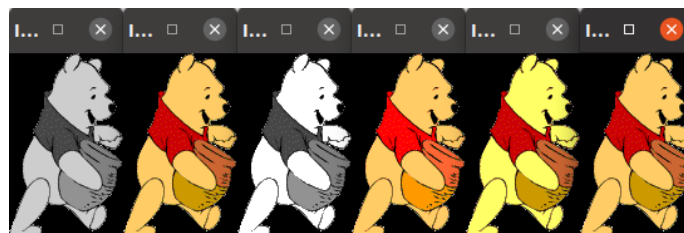


Rysunek 4: Porównanie progowania czerni o progu 60.

3.3.3 Test 3

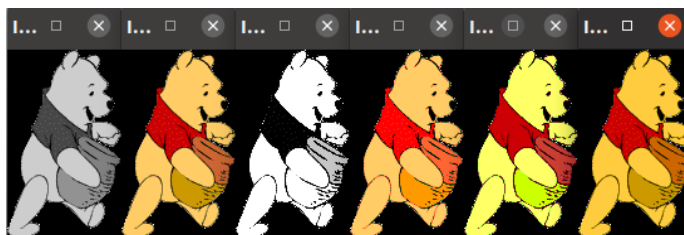


Rysunek 5: Porównanie progowania bieli o progu 40.

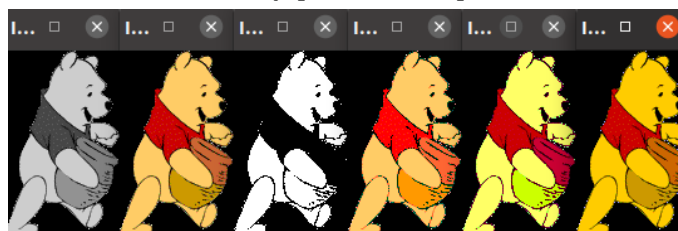


Rysunek 6: Porównanie progowania bieli o progu 60.

3.3.4 Test 4

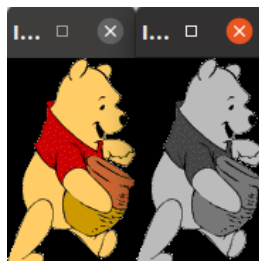


Rysunek 7: Porównanie zamiany poziomów o poziomie czerni 35 i bieli 55.



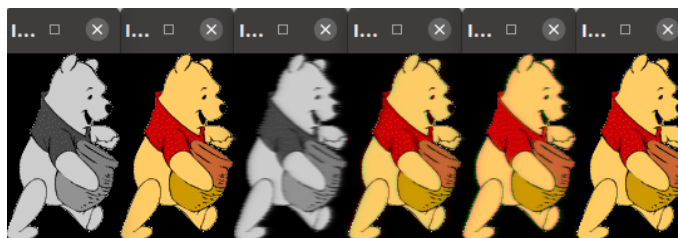
Rysunek 8: Porównanie zamiany poziomów o poziomie czerni 55 i bieli 35.

3.3.5 Test 5

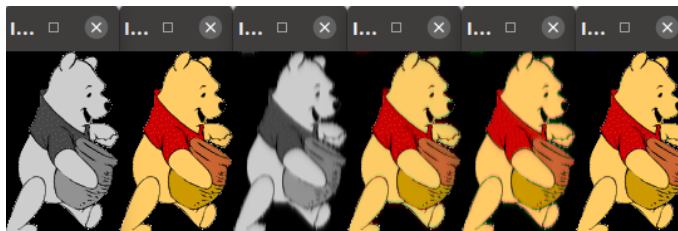


Rysunek 9: Porównanie oryginału Kubusia z Kubusiem skonwertowanym do szarości.

3.3.6 Test 6

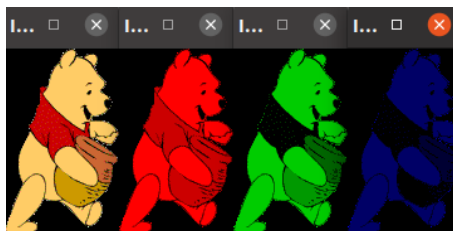


Rysunek 10: Porównanie rozmywania poziomego o promieniu równym 2.



Rysunek 11: Porównanie rozmywania pionowego o promieniu równym 2.

3.3.7 Test 7



Rysunek 12: Porównanie oryginalnego Kubusia z Kubusiami ze zmienionym kolorem.

3.3.8 Test 8

[illegible]

Rysunek 13: Zdjęcie potwierdzające to, że obraz został wczytany ze standardowego wejścia i obraz który otrzymaliśmy.

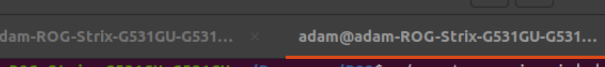
3.3.9 Test 9

```
adam@adam-ROG-Strix-G531GU-G531GU: ~/Programy/PO2
adam@adam-ROG-Strix-G531GU-G531GU:~/Programy/PO2$ ./przetwarzanie -i kubus.ppm
n -d -p 20 -o
Blad nr -2
adam@adam-ROG-Strix-G531GU-G531GU:~/Programy/PO2$ ./przetwarzanie -i -n -d -p 20
-o kubus---.ppm
Blad nr -4
adam@adam-ROG-Strix-G531GU-G531GU:~/Programy/PO2$ ./przetwarzanie -i kubus.ppm
n -d -p -o kubus---.ppm
Blad nr -3
adam@adam-ROG-Strix-G531GU-G531GU:~/Programy/PO2$
```

Rysunek 14: Efekt działania bez informacji po flagach

Przy okazji zostało sprawdzone czy funkcja wyświetlania obrazu działa gdy plik nie zostanie zapisany. Funkcja nie działała poprawnie.

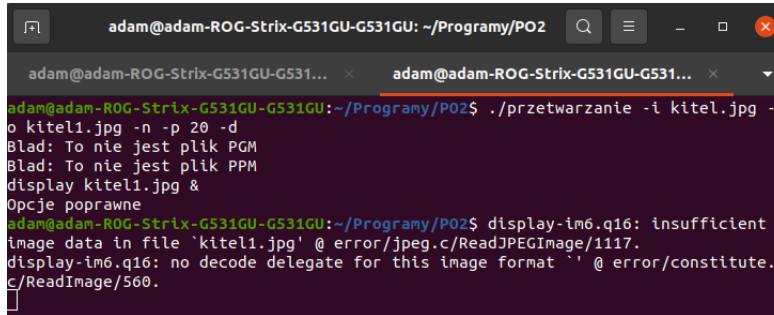
3.3.10 Test 10



```
adam@adam-ROG-Strix-G531GU-G531GU: ~/Programy/PO2
adam@adam-ROG-Strix-G531GU-G531GU: ~/Programy/PO2$ ./przetwarzanie -i kubus.ppm -o kubus.bezm.ppm -n -p 20 -d
display kubus.bezm.ppm &
Opcje poprawne
adam@adam-ROG-Strix-G531GU-G531GU: ~/Programy/PO2$
```

Rysunek 15: Efekt działania bez informacji o kolorze.

3.3.11 Test 11



```
adam@adam-ROG-Strix-G531GU-G531GU: ~/Programy/PO2
adam@adam-ROG-Strix-G531GU-G531GU:~/Programy/PO2$ ./przetwarzanie -i kitel1.jpg -o kitel1.jpg -n -p 20 -d
Bład: To nie jest plik PGM
Bład: To nie jest plik PPM
display kitel1.jpg &
Opcje poprawne
adam@adam-ROG-Strix-G531GU-G531GU:~/Programy/PO2$ display-im6.q16: insufficient image data in file 'kitel1.jpg' @ error/jpeg.c/ReadJPEGImage/1117.
display-im6.q16: no decode delegate for this image format '' @ error/constitute.c/ReadImage/560.
```

Rysunek 16: Próba przetworzenia obrazu w formacie JPG.

4 Wnioski końcowe

Przetestowanie programu pozwoliło na przetworzenie obrazów kubus.pgm i kubus.ppm. Test 1. pokazał, że funkcje, które były zawarte w 1. części przetwarzania obrazów działają poprawnie dla obrazów w formacie PGM i PPM we wszystkich kolorach. Test 2. i 3. również pokazały, że progowania czerni i bieli działają bez problemu. Test 4. pokazał, że funkcja zamiany poziomów działa poprawnie i zamienia poziomy. Również test 5., który testował funkcję konwersji, pokazał, że zamienia pliki PPM na pliki PGM. Test funkcji rozmywania również pokazał, że funkcje te działają poprawnie. Również moja dodatkowa funkcja zamiany koloru działa bez zarzutów. Test 8. pokazał, że program poprawnie oczekuje obrazu na standardowym wejściu. Test 9. pokazał, że jeżeli nie podamy odpowiednich informacji przy flagach to wyskoczą nam błędy. Ciekawe jest to, że gdy damy flagę „-o” w środku wywołania, a po tej fladze kolejne flagi, to flaga będąca po fladze „-o” nie wykona przetwarzania, ale plik zostanie zapisany pod nazwą tej flagi. Podobna sprawa istnieje gdy po fladze „-i” nie damy „-”. Flaga ta uzna wtedy, że ma odczytać plik zapisany pod nazwą następnej flagi. Również okazało się, że funkcja wyświetlania nie działa przy braku zapisania pliku. Dzieje się tak, ponieważ wyświetla ona plik nowo powstały, gdy go nie zapiszemy nie wyświetli ona nam go. Efekt działania funkcji testowanej w teście 10. również uważam za zadowalający, ponieważ zgodnie z założeniami, gdy nie podamy informacji o

kolorze, operacje są wykonywane na każdym kanale. Jednakże trzeba pamiętać, że gdy użyjemy flagi „-m” bez odpowiedniej informacji po niej, odczyta ona następną flagę jako swój argument, przez to ta „zjedzona” flaga nie wykona przetwarzania, a pozostałe przetwarzania wykonają się na wszystkich kanałach. Test 11. pokazał, że plik w innym formacie niż PPM lub PGM nie zostanie przetworzony, ani nawet wyświetlony, ponieważ nie zostanie poprawnie zapisany. Po tych wszystkich testach mogę powiedzieć, że program działa poprawnie.