Przetwarzanie obrazów 2

Kacper Wiącek

9 Styczeń 2020

Spis treści

Listings

Spis rysunków

1 Program i jego działanie

Program działa na podobnej zasadzie jak poprzedni "Przetwarzanie obrazów 1" lecz posiada znaczące różnice działania. Na wstępie należy wywołać program wraz z określonymi opcjami oraz argumentami (jeśli jest to konieczne). Kolejne opcje to jedno literowe znaki zaczynające sie znakiem "-". Program na ich podstawie wie, które operacje ma wykonać spośród wszystkich możliwych. Wymagane jest podanie opcji "-i" z argumentem "-" badz nazwa pliku wejsciowego, na ktorym mamy wykonywac rozne operacje. W programie "Przetwarzanie obrazów 2" dodatkowo możliwe jest wczytanie pliku formatu PPM. Program po wczytaniu wie z jakim plikiem mamy do czynienia(PGM lub PPM). Możliwe jest wybranie funckji do przetworzenia obrazów spośrod wszytskich poniżej:negatyw(opcja-n),progowanie(opcja -p)(wymagane podanie argumentu progu), progowanie bieli(opcja -pb)(wymagane podanie argumentu progu bieli),progowanie czerni(opcja -pc)(z podaniem wartosci progu czerni),korekcja gamma(opcja -g)(z podaniem wspolczynnika gamma),zmiana poziomów(opcja -z)(wymagne argumenty czern biel w podanej kolejnosci,konturowanie(opcja -k), rozmywanie poziome(opcja -rx), rozmywanie pionowe(opcja -ry), rozciaganie histogramu(opcja -h). Oraz opcje dodatkowe dzialajace tylko na obrazach PPM: konwersja do stopnia szarosci(opcja -m z argumentem s) obraz PPM na wyjsciu ma format PGM, oraz funckja przetwarzaj kolor(-m (podajemy kolor z trzech -> czerwony('r'), zielony('g', niebieski('b')). DO funckji przetwarzaj kolor wymagane jest podanie przynjamniej jednej opcji przetwarzania obrazow, aby to mialo sens. Uwaga gdy wczytamy obraz formatu PPM nalezy go najpierw skonwertowac badz wybrac tylko jeden kolor do przetwarzania!!. Funkcja przetwarzania obrazow działają tylko na obrazach PPM badz na jednym kolorze z trzech w obrazie PPM.!!Po operacjach nastepuje zapis pliku. W przypadku braku opcji (-o) program wypisze plik na standardowe wyjscie. PO zapisaniu Możliwe jest również wyświetlenie przetworzonego obrazu opcja (-d). Ta operacja wykonuje się zawsze na końcu jesli zosatnie wyywołana(wyswietlamy tylko przetworzony obraz).Gdy podczas wywolania programu nastapi blad jakikolwiek to dostaniemy odpowiedni komunikat, program się zakończy.

2 Różnice, nowości, porównanie programów PO1 oraz PO2

1. Program PO2 został podzielony na moduł (plik z nagłowkami funckji oraz z strukturami -> modul.h, plik z deklaracjami funkcji -> modul.c oraz funckja główna -> main.c) pozwala to na sprawne i wydajne poruszanie się po całym programie. Program jest uporządkowany i czytelny dla oka. Dodatkowo mamy plik Makefile, który kompiluje za nas te pliki w jedna calosc, pozwala to nam zaoszczedzić czas i cierpliwść.

2. Kolejna zmiana bardzo użyteczna to dynamiczne alokowanie pamięci i posługiwanie się zmienną typu struktura. Nasz program działa dynamicznie, Ilość pamięci przydzilonej zostaje tak dużo ile rzeczywiście potrzeba na rozmiar obrazu.Struktura typu t obraz przechowuje wszystkie informacje o obrazie w jednym miejscu, co naprawdę nie powoduje zagubienia wśród całego kodu programu. DOdatkowo struktura t opcje zawiera wszystkie wywolane opcje i argumenty rowniez w jednym miejscu. Bardzo rozsądne i umożliwiające dokonywać zmian szybciej i skuteczniej. Ważne jest aby zwalniać pamięć kiedy tylko to konieczne(gdy mamy jej nadmiar, badz już jej nie potrzebujemy), aby program byl wydajny i na miare swoich możliwości jak najbardziej elastyczny.

3. Równie ważnym jak i nie najważniejszym są asercje. SPrawdzaja poprawnosc wywolania programu, czy prawidłowo przydzielono pamieć, oraz przypadki szczególne funkcji. Złe arguemnty badz opcje nie pozwolą na poprawne działanie.

4. Dodatkowa możliwość to obsługa obrazów kolorowych foramtu PPM. Każdy piksel składa się z trzech składowych kolorowych : czerwony, niebieski oraz zielony. Magiczne słówko "P3" oraz wysokość i szerokość a także wartość maksymalnych składowych kolorowych obrazu jest niemal identyczna jak w pliku PGM. Ten wczytany obraz można skonwertować (dodatkowa funckja w programie) oraz przetworzyć go ale uwaga tylko jeden kolor z pośród trzech wybranych.

5. zamiast menu uztkownika wprowadzono jak wczesniej bylo napisane możliwość wywyołania programy z opcjami, arguemnta-

mi (w razie koniecznosci). Kolejnosc opcji w jakiej mogą wystąpić opcje jest dowolna, Program wykonuje każdą opcję, wielkokrotnie jeśli jest taka potrzeba. Kolejność w jakiej wykonują się operacje nie jest losowa.Obraz zostanie przetworzony dzięki podanej kolejności wywołania opcji w programie. Jedynie zapis,wczytywanie, wyświetlanie oraz konwersja działają poza regułą kolejności.Najpierw zawsze obraz jest wczytywany, jeśli jest to obraz PPM dochodzi do konwersji bądź przetwarzania koloru.Następnie wykonują się operacje przetwarzania w odpowiedniej kolejności. Na samym końcu jest zapis i wyświetlanie.

3 Kompilacja

Kompilacja pliku modul.c oraz main.c można wykonać w podany sposób:

```
/*plik wykonawczy main.o sk ada si  z nag owkowego modul.h oraz g ownego main.c*/
main.o: main.c modul.h
  gcc -c main.c
/*plik wykonawczy modul.o sk ada si  z nag  wkowego modul.h oraz modul.c */
funkcje.o: modul.c modul.h
  gcc -c modul.c
/* Na samym koncu laczymy to w calosc*/
main: main.o modul.o
  gcc -std=c99  -Wall -pedantic main.o modul.o -lm -o prog
```

Listing 1: Kompilacja

Można również ułatwić sprawę i skorzystać z dodanego pliku Makefile, który wykona to za nas i oczywiscie dodatkowo opcja "make clean" posrząta za nas wszystkie pliki wykonawcze z formatem .o

```
#Makefile
#Wywo anie komend  "make"
CFLAGS = -std=c99 -pedantic -Wall  #Inne flagi kompilatora
LIBS = -lm #DOdawanie bibliotek
#Z podanych wcze niej wygenerowanych plik w  ca kowita kompilacja
main: main.o modul.o
  gcc $(CFLAGS) main.o modul.o $(LIBS) -o prog

#Aby otrzyma  plik main.o podajemy z czego: main.c modul.h oraz jak: gcc -c main.c
main.o: main.c modul.h
  gcc -c main.c

#Aby otrzyma  plik "modul.o" podajemy z czego: modul.c, modul.h oraz jak: gcc -c modul.c
funkcje.o: modul.c modul.h
```

```
15   gcc -c modul.c
16 # KOmenda "make clean" czy ci pliki z rozszerzeniem .o oraz
       utworzony plik prog
17 clean:
18   rm *.o prog
```

Listing 2: Makefile

4 Szczególne rozwiązania określonych problemów w kodzie

4.1 Dodane biblioteki, stałe symboliczne, zmienne

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>               /*biblioteka u yta do operacji
       matematycznych*/
5  #include "modul.h"              /*przydzielenie nag   wk w funckji
        oraz struktur danych*/
6
7  #define DL_LINII 1024             /* Dlugosc buforow pomocniczych
       */
8  #define W_OK 0                    /* wartosc oznaczajaca brak bledow
        */
9  #define B_NIEPOPRAWNAOPCJA -1     /* kody bledow rozpoznawania opcji
        */
10 #define B_BRAKNAZWY   -2
11 #define B_BRAKWARTOSCI  -3
12 #define B_BRAKPLIKU   -4
13 #define B_BLEDNE_DANE -5
14 #define B_NIEPOPRAWNY_PLIK -6
15 #define B_BRAK_PAMIECI -7
16 /*dodatkowo definiowanie pliku naglowkowego*/
17 #ifndef MODUL_H
18 #define MODUL_H
19 #endif
```

Listing 3: Biblioteki i stałe symboliczne

Skorzystaliśmy z biblioteki <math.h>, która posiadała funkcje "pow" oraz "abs" potrzebne do wykonywania operacji i obliczeń.Stała "DL LINII" użyta została w celu zdefiniowania maksymalnego rozmiaru bufora pomocniczego do odczytu plików.Mamy stałe symboliczne z wartościami błedów oraz stałą W OK gdy nie ma błędów.Mamy PLik nagłówkowy MODUL.H, który zawiera nagłówki oraz struktury programu. Ważne i niezwykle pożyeczne struktury są jednym workiem na dane gdzie możemy przechowywać wszystkie dane o obrazie i wybranych opcjach wywyołania programu. W strukturze

opcje mamy uchwyty do plików, wybrane opcje z argumentami i nazwę pliku, jakby trzeba było go wyświetlić. Struktura obraz przechowuje wszytsko o obrazie -> format, wysokość, szerokość stopień odcieni lub składowych obrazu, oraz wszystkie piskele w dynamicznej tablicy dwuwymiarowej. Struktura przechowuje wskaznik na taka tablice typu void gdyż nie jesteśmy w stanie określić jakiego typu ma być zaalokowana tablica.

```
/* strukura do zapamietywania opcji podanych w wywolaniu programu */
typedef struct {
  FILE *plik_we, *plik_wy;     /* uchwyty do pliku wej. i wyj. */
  char *nazwa_pliku;           /*nazwa pliku wyj. do wy wietlania przetworzonego obrazu*/
  int negatyw,progowanie,konturowanie,wyswietlenie,korekcja_gamma, roz_histogramu;   /* opcje */
  int progowanie_czerni,progowanie_bieli,zmiana_poziomow, rozmywanie_poziome, rozmywanie_pionowe;/*dodatkowe opcje*/
  int przetwarzaj_kolor, konwersja_do_szarosci; /*opcje do obrazow ppm*/
  int w_progu;            /* wartosc progu dla opcji progowanie*/
  float w_gamma;            /*wartosc wspo czynnika dla opcji korekcji gamma*/
  int w_progu_bieli;     /*wartosc progu dla opcji progowanie bieli */
  int w_progu_czerni;    /*wartosc progu dla opcji progowanie czerni*/
  int biel,czern;        /*dwie wartosci dla opcji zmiany poziom w */
  char kolor;            /*kolor do przetwarzania funckji przetwarzaj_kolor*/
  } t_opcje;             /*zmienna typu t_opcje*/

/*struktura do zapisywania danych o wczytanym obrazie pgm lub ppm*/
typedef struct {
  int format; /*przechowuje informacje o formacie wczytanego obrazu */
  int wym_x, wym_y, odcieni;   /*szeroko    obrazu,wysoko obrazu,odcieni->warto    odcieni szarosci lub sk adowych kolorowych obrazu*/
  void *piksele;               /*wskaznik piksele wskazuje na tablice typu "pustego"*/
} t_obraz;/*zmienna typu t_obraz*/       /*dynamiczna tablica dwuwymiarowa do przechowywania pikseli obrazu*/
```

Listing 4: Zdefiniowane struktury

4.2 Asercje

W naszym programie mogą występować błędy. Zadaniem asercji jest je wykryć i zakomunikować o jakim błędzie mamy do czynienia. W programie możemy napotkać się na:

1. NIEPOPRWNĄ OPCJĘ przy wywołaniu programu
2. BRAK NAZWY pliku wejściowego lub wyjściowego
3. BRAK WARTOŚCI arguemntu koniecznego do wywołania funckji jak progowanie czy korekcji gamma
4. BRAK PLIKU nie otwarcie pliku wejściowego
5. BŁĘDNE DANE ten rodzaj błędów można napotkać w przypadkach szczególncyh jak: zerowanie mianownika, nie podania operacji (do przetwarzania kolru wymagane jest podanie conajmniej jednej), wywołaniu funkcji przetwarzania na obrazie PPM bez wcześniej konwersacji itd.
6. NIEPOPRAWNY PLIK W funkcji czytaj obraz musi być formatu PPM lub PGM lub złe wymiary obrazu, brak wartości szerokości itd.
7. BRAK PAMIĘCI błąd przy alkokowaniu pamięci, gdy wskaznik zwraca NULL, nie przydzielono pamięci.

4.3 Opcje wywoływania programu

Program rozpoznaje opcje jakie ma wykonać na podstawie znaku jednego (badz dwoch) znaków. Ten zank powinien być poprzedzony znakiem "-". DO opcji -g, -p, -pb. pc, -z, należy podać dodatkowo arguemnt (badz dwa). Opcja -i odpowiada za wczytanie pliku (wymaga argumentu "-" bądź nazwy pliku) Opcja "-" wywołana bez argumentu obraz wczytany powinien być ze standardowego wejścia. W przypadku braku opcji (-o) obraz zostaje wypisany na standardowe wyjście. W przypadku błędnych opcji lub braku argumentów zostaje wyświetlony komunikat z błędem. Kolejnoscść wywołania opcji nie ma zanczenia.

```
1 int przetwarzaj_opcje(int argc, char **argv, t_opcje *wybor) {
2     wyzeruj_opcje(wybor);wybor->plik_wy=stdout;        /* na
      wypadek gdy nie podano opcji "-o" */
3     for (i=1; i<argc; i++) {
4       if (argv[i][0] != '-'){  /* blad: to nie jest opcja - brak
      znaku "-" */
5           fprintf(stderr,"\nBlad! Niepoprawna opcja!\n\n"); return
      B_NIEPOPRAWNAOPCJA;}
6     switch (argv[i][1]) {
7     case 'i': { /* opcja z nazwa pliku wejsciowego */
8         if (++i<argc) {   /* wczytujemy kolejny argument jako nazwe
       pliku */ nazwa_pliku_we=argv[i];
```

```
9         if (strcmp(nazwa_pliku_we,"-")==0) /* gdy nazwa jest "-"
           */
10           wybor->plik_we=stdin;/* ustwiamy wejscie na stdin*/else
     /* otwieramy wskazany plik*/
11        wybor->plik_we=fopen(nazwa_pliku_we,"r");
12     } else { fprintf(stderr,"\nBlad brak nazwy pliku\n\n");
13        return B_BRAKNAZWY;}/* blad: brak nazwy pliku */break;}
14     case 'o': {  /* opcja z nazwa pliku wyjsciowego */
15       if (++i<argc) {   /* wczytujemy kolejny argument jako nazwe
     pliku */
16   nazwa_pliku_wy=argv[i];
17   if (strcmp(nazwa_pliku_wy,"-")==0)/* gdy nazwa jest "-"
     */
18       wybor->plik_wy=stdout;          /* ustwiamy wyjscie na stdout
      */
19   else  { /* otwieramy wskazany plik*/ if(argv[i][0] != '-'){    /*
     Na wszelki wypadek*/
20     wybor->plik_wy=fopen(nazwa_pliku_wy,"w"); /*zapisujemy nazwe
     pliku*/
21       wybor->nazwa_pliku=nazwa_pliku_wy;}       /*aby go wyswietlic
      w razie potrzeby*/
22     else{ fprintf(stderr,"\nBlad, Brak nazwy pliku\n\n"); return
     B_BRAKNAZWY;}}
23     }else{ fprintf(stderr,"\nBlad, Brak nazwy pliku\n\n");
24   return B_BRAKNAZWY;}/* blad: brak nazwy pliku */ break;}
25      case 'n': {/* mamy wykonac negatyw */ wybor->negatyw=1; break
     ;}
26     case 'k': { /* mamy wykonac konturowanie */ wybor->konturowanie
     =1; break;}
27     case 'd': { /* mamy wyswietlic obraz */ wybor->wyswietlenie=1;
     break;}
28     case 'h' : {/*mamy wykonac roz_histogramu*/ wybor->
     roz_histogramu=1; break;}
29     case 'g': {              /*mamy wykonac korekcje_gamma*/
30       if((++i<argc)&&(argv[i][0]!='-')) {  /* wczytujemy kolejny
     argument jako wspolczynnik gamma*/
31   if(sscanf(argv[i],"%f",&gamma)==1) {
32     wybor->korekcja_gamma=1; wybor->w_gamma=gamma;} else { /*blad
     bledna wartosc wspolczynnika gamma*/
33     fprintf(stderr,"\nBlad, Podano nieprawidlowa wartosc
     wspolczynnika gamma\n\n");
34     return B_BRAKWARTOSCI;} } else{ /*blad brak wspolczynnika gamma
     */
35   fprintf(stderr,"\nBlad, Nie podano wartosci wspolczynnika gamma\n
     \n");
36   return B_BRAKWARTOSCI;} break;}               /*mamy wykonac
     zmiane_poziomow*/
37     case 'z' : { if((++i<argc)&&(argv[i][0]!='-')) {  /*wczytujemy
     kolejny argument czerni*/
38   if(sscanf(argv[i],"%d", &czarny)==1) {wybor->czern=czarny;
39   } else { /*blad nieprawidlowy argument czerni*/
40     fprintf(stderr,"\nBlad, Podano nieprawidlow  warto c czerni\n
     \n"); return B_BRAKWARTOSCI;}
41      } else { /*blad brak argumentow*/
42  fprintf(stderr,"\nBlad, Nie Podano argumentu czerni oraz bieli\n\n
     "); return B_BRAKWARTOSCI;}
```

```
43      if((++i<argc)&&(argv[i][0]!='-')) {     /*Wczytujemy argument
        bieli*/
44    if(sscanf(argv[i],"%d", &bialy)==1) { wybor->biel=bialy; wybor->
        zmiana_poziomow=1;
45    } else { /*blad bledny parametr*/
46      fprintf(stderr,"\nBlad, Podano nieprawidlow  warto   bieli\n
        \n"); return B_BRAKWARTOSCI;}
47    } else { /*blad brak parametru*/
48    fprintf(stderr,"\nBlad, Nie Podano drugiego argumentu bieli\n\n")
        ; return B_BRAKWARTOSCI;}
49      break;}             /*mamy wykonac operacje
        konwersji_do_szarosci*/
50      case 'm' : { if((++i<argc)&&(argv[i][0]!='-')){ /* Wczytujemy
        kolejny arguemnt*/
51    if(sscanf(argv[i],"%c", &kolorek)==1) {
52        if((kolorek=='r') || (kolorek=='g') || (kolorek=='b')) {
                    wybor->kolor=kolorek; wybor->przetwarzaj_kolor
        =1;
53        } /*argument ma wartosc 's' wykonujemy
        konwersje_do_szarosci*/
54        else if(kolorek=='s') wybor->konwersja_do_szarosci=1; else{
         /*blad bledna opcja */
55         fprintf(stderr,"\nBlad! Niepoprawny argument do opcji -m!\
        n\n"); return B_NIEPOPRAWNAOPCJA;}
56    } else {/*blad nieprawidlowy parametr*/ fprintf(stderr,"\nBlad,
        Podano nieprawidlowy parametr\n\n");
57      return B_BRAKWARTOSCI;} } else { /*blad brak parametru*/
58    fprintf(stderr,"\nBlad,Nie podano koloru do przetworzenia bad
        parametru 's'!\n\n");
59    return B_BRAKWARTOSCI;}
60      break;} case 'p': { switch(argv[i][2]) {     /*mamy wykonac
        progowanie_czerni*/
61        case 'c' : { if((++i<argc)&&(argv[i][0]!='-')){/*wczytujemy
         kolejny argument progu czerni*/
62      if(sscanf(argv[i],"%d", &prog_cz)==1){ wybor->progowanie_czerni
        =1; wybor->w_progu_czerni=prog_cz;
63      } else { /*blad bledny argument*/
64       fprintf(stderr,"\nBlad, Podano nieprawidlowa wartosc progu
        czerni\n\n");
65        return B_BRAKWARTOSCI;} } else { /*blad brak argumentu*/
66        fprintf(stderr,"\nBlad,Nie podano wartosci progu czerni\n\n")
        ; return B_BRAKWARTOSCI;}
67        break;}  /*mamy wykonac progowanie_bieli*/ case 'b' : {
68          if((++i<argc)&&(argv[i][0]!='-')) {    /*wczytujemy
        kolejny argument progu bieli*/
69      if(sscanf(argv[i],"%d", &prog_b)==1) { wybor->progowanie_bieli
        =1; wybor->w_progu_bieli=prog_b;
70      } else { /*blad bledny argument*/
71      fprintf(stderr,"\nBlad, Podano nieprawidlowa wartosc progu
        bieli\n\n");
72      return B_BRAKWARTOSCI;} } else { /*blad brak argumentu*/
73      fprintf(stderr,"\nBlad,Nie podano wartosci progu bieli\n\n");
        return B_BRAKWARTOSCI;}
74        break;}  /*mamy wykonac progowanie*/ case '\0' : {
75          if ((++i<argc)&&(argv[i][0]!='-')) { /* wczytujemy
        kolejny argument jako wartosc progu */
```

```
76      if (sscanf(argv[i],"%d",&prog)==1) { wybor->progowanie=1; wybor
        ->w_progu=prog;
77      } else { fprintf(stderr,"\nBlad, Podano nieprawidlowa wartosc
        progu\n\n");
78      return B_BRAKWARTOSCI;}     /* blad: niepoprawna wartosc progu
        */
79       } else { fprintf(stderr,"\nBlad,Nie podano wartosci progu\n\n"
        );
80      return B_BRAKWARTOSCI;} /* blad: brak wartosci progu */ break;}
81        default: { /*blad niepoprawna opcja*/ fprintf(stderr,"\nBlad!
         Niepoprawna opcja!\n\n");
82           return B_NIEPOPRAWNAOPCJA;}}break;}
83      case 'r' : { switch(argv[i][2]) { /*mamy wykonac
        rozmywanie_poziome*/
84          case 'x' : { wybor->rozmywanie_poziome=1; break;}
85          case 'y' : {/*mamy wykonac rozmywanie_pinowe*/ wybor->
        rozmywanie_pionowe=1; break;}
86          default: { /*blad niepoprawna opcja*/ fprintf(stderr,"\
        nBlad! Niepoprawna opcja!\n\n");
87            return B_NIEPOPRAWNAOPCJA;}}break;} default: {
88        fprintf(stderr,"\nBlad! Niepoprawna opcja!\n\n");      /*
        nierozpoznana opcja */
89        return B_NIEPOPRAWNAOPCJA;}} /*koniec switch */} /* koniec
        for */
90    if (wybor->plik_we!=NULL)     /* ok: wej. strumien danych
        zainicjowany */ return W_OK;else {
91      fprintf(stderr,"\nBlad, Nie otwarto pliku wejsciowego\n\n");
92      return B_BRAKPLIKU;}         /* blad:  nie otwarto pliku
        wejsciowego  */}
```

Listing 5: Ffragment funkcji Przetwarzaj opcje

4.4 Wykonywanie opcji

Opcja wczytaj i zapisz wykonuje się zawsze. Najapierw program wczytuje plik. Następnie plik formatu PPM ma pierwszeństwo i wykonujemy operacje konwersji bądź przetwarzania wybranego koloru jeśli te opcje zostaną wywołane. Opcje przetwarzania obrazów wykonują się w podanej kolejności wywoływania programu. Jeśli przetworzono obraz oraz podano nazwę pliku wyjsciowego, wybrano opcje wyświetlanie to obraz się wyświetla. (opcja wyswietlanie wykonuje się jako osatnia).Funckja korzysta z tablicy argv, która zawiera argumenty wywołania.Za pomocą pętli będziemy w stanie wykonywać opcje w kolejności i wielokrotnie.Gdy wszystko zostało wykonane pomyślnie funckja zwraca wartość 0.

```
int wykonaj_opcje(t_opcje *opcje, t_obraz *obraz, int argc, char **
    argv) {
  /*Najpierw nastepuje konieczne wczytanie pliku*/
  if(wczytaj(opcje->plik_we,obraz)!=0) exit(-1);
  /*W zaleznosci od wybranej opcji wykona sie dana operacja*/
  /*Do kazdej funkcji dostarczany jest parametr wskaznik na
    strukture obrazu*/
  /*oraz parametry potrzebne do przetworzenia w okreslonych
    funkcjach*/
  if(opcje->przetwarzaj_kolor==1) if(przetwarzaj_kolor(opcje,obraz,
    opcje->kolor,argc,argv)!=0) exit(-1);
  if(opcje->konwersja_do_szarosci==1) if(konwersja_do_szarosci(
    obraz)!=0) exit(-1);
  if(opcje->przetwarzaj_kolor!=1){ /*na wypadek dwukrotnego
    powtorzenia tej samej operacji*/
  for(int i=0; i<argc; i++){ /*W tablicy argv zapisane s  opcje w
    kolejno ci wywo ania*/
      switch(argv[i][1]){ /*W zale no ci aktualnej opcji argv[i
    ][1] wywo ujemy okre lon  funkcje*/
          case 'n':{
               if(negatyw(obraz)!=0) exit(-1);
               break;}
          case 'p':{
              switch(argv[i][2]){
                  case '\0':{
                      if(progowanie(obraz,opcje->w_progu)!=0) exit
    (-1);
                      break;}
                  case 'c':{
                      if(progowanie_bieli(obraz,opcje->
    w_progu_czerni)!=0) exit(-1);
                      break;}
                  case 'b':{
                      if(progowanie_czerni(obraz,opcje->
    w_progu_bieli)!=0) exit(-1);
                      break;}}
              break;}
          case 'g':{
              if(korekcja_gamma(obraz,opcje->w_gamma)!=0) exit(-1);
```

```
29              break;}
30          case 'z':{
31              if(zmiana_poziomow(obraz,opcje->czern,opcje->biel)
    !=0) exit(-1);
32              break;}
33          case 'k':{
34              if(konturowanie(obraz)!=0) exit(-1);
35              break;}
36          case 'r':{
37              switch(argv[i][2]){
38                  case 'x':{
39                      if(rozmywanie_poziome(obraz)!=0) exit(-1);
40                      break;}
41                  case 'y':{
42                      if(rozmywanie_pionowe(obraz)!=0) exit(-1);
43                      break;} }
44              break; }
45          case 'h':{
46              if(roz_histogramu(obraz)!=0) exit(-1);
47              break;}/*pomijamy zapis, wczytywanie i ewentualnie
    wy wietlanie*/
48          default: break;}}}
49  zapisz(opcje->plik_wy, obraz); /*Zapisanie zmian w pliku
    wyjsciowym*/
50  if(opcje->wyswietlenie==1){ /*Nie wyswietlimy obraz ze
    standardowego wyjscia*/
51      if(opcje->plik_wy==stdout) printf("\n\nNie mozna wyswietlic
    obrazu ze standardowego wyjscia!!\n\n");
52      else wyswietlanie(opcje->nazwa_pliku);} /*Na samym koncu
    wyswietlamy dokonane zmiany*/
53  return W_OK; /*Udalo sie wykonac wszytsko bez bledow? ->  zwroc 0
    */}
```

Listing 6: Wykonaj opcje

4.5 Wczytanie pliku

WYbranie opcji -i jest konieczne z argumentem. Funckja wczytaj rozpoznaje format pliku, alokuje potrzebną pamięć na tablice do pikseli obrazu. Zapisuje wszystkie dane do struktury obraz. Sprawdza zgodność danych, występowanie wszytskich arguemntów (wysokość, szerokość, "P2" LUB "P3" itd.) W zależności od magicznego P2 LUB P3 alkokuje inną pamięć. FUnkcja pomija zbędne komentarze i ich nie zapisuje. Na sam koniec zwraca wartość zero gdy nie wystąpił żaden błąd.

```
1  int wczytaj(FILE *plik_we, t_obraz *obraz) {
2    char buf[DL_LINII];      /* bufor pomocniczy do czytania naglowka
       i komentarzy */
3    int znak;                /* zmienna pomocnicza do czytania
       komentarzy */
4    int koniec=0;            /* czy napotkano koniec danych w pliku
       */
```

```
5  /* Sprawdzenie "numeru magicznego" - powinien by  P2 lub P3 */
6  if (fgets(buf,DL_LINII,plik_we)==NULL)  /* Wczytanie pierwszej
     linii pliku do bufora */
7      koniec=1;                          /* Nie udalo sie? Koniec
      danych! */
8  if ( (buf[0]!='P') || koniec) {     /*Brak litery "P"-> z y
     format pliku*/
9    fprintf(stderr,"\n\t\tBlad: To nie jest plik PGM ani PPM!\n\n")
     ; return B_NIEPOPRAWNY_PLIK;}
10 if (buf[1]!='2') {   /* Czy jest magiczne "P2" lub "P3? */
11     if (buf[1]!='3') { fprintf(stderr,"\n\t\tBlad: To nie jest
     plik PGM ani PPM!\n\n");
12           return B_NIEPOPRAWNY_PLIK;}} /*Mamy "P3" ?->wczytujemy
      obraz PPM*/
13   do { /* Pominiecie komentarzy */
14       if ((znak=fgetc(plik_we))=='#') {        /* Czy linia
     rozpoczyna sie od znaku '#'? */
15           if (fgets(buf,DL_LINII,plik_we)==NULL)    /* Przeczytaj
      ja do bufora*/
16           koniec=1;                                 /* Zapamietaj
     ewentualny koniec danych */
17       }
18       else{                                     /* Gdy przeczytany
      znak z poczatku linii */
19           ungetc(znak,plik_we);/* nie jest '#' zwroc go*/
20       }                                             /* Powtarzaj
     dopoki sa linie komentarza */
21   }while (znak=='#' && !koniec);                    /* i nie
     nastapil koniec danych          */
22   /* Pobranie wymiarow obrazu i liczby sk adowych kolorowych
     obrazu*/
23   if(fscanf(plik_we,"%d %d %d", &(obraz->wym_y), &(obraz->wym_x),
      &(obraz->odcieni))!=3){
24   fprintf(stderr,"\n\t\tBlad: Brak wymiarow obrazu lub
     maksymalnej liczby sk adowych kolorowych obrazu\n\n"); return
     B_NIEPOPRAWNY_PLIK;}
25   obraz->piksele = malloc(obraz->wym_x*3*obraz->wym_y*sizeof(int)
     );
26   if(obraz->piksele==NULL){              /*Nie uda o si
     przydzieli  pami ci? -> zwroc b ad*/
27     fprintf(stderr,"\nBlad, Brak przydzialu pamieci\n\n"); return
      B_BRAK_PAMIECI;}
28   int (*piksele)[obraz->wym_x*3]; piksele=(int(*)[obraz->wym_x
     *3]) obraz->piksele;
29   /* Pobranie obrazu i zapisanie w strukturze t_obraz obraz*/
                                    for(int i=0; i<obraz->wym_y; i
     ++) {
30       for(int j=0; j<obraz->wym_x*3; j++) {
31           if(fscanf(plik_we,"%d", &(piksele[i][j]))!=1){   /*blad
      zle wymiary? ->zakomunikuj o bledzie i zakoncz*/
32               fprintf(stderr,"\n\t\tBlad: Niewlasciwe wymiary
     obrazu\n\n");
33               return B_NIEPOPRAWNY_PLIK;}}}
34   obraz->format=3; /*Zapisanie format=3 -> wczytano obraz PPM*/
     return W_OK;}
```

Listing 7: Wczytanie obrazu PPM - fragment kodu

4.6 Zapis obrazu do pliku

Opcja -o musi się pokazać lub nie. POdajemy arguemnt nazwy pliku do ktorego zapisać przetworzony obraz. Bez argumentu badź podanie argumentu "-" -> wypisujemy obraz na standardowe wyjście. Funckja zapisz najpierw sprawdza format pliku, na jego podstawie wie jaki obraz ma zapisać (PPM LUB PGM).Na samym końcu następuje zwalnianie pamięci. Gdy wszystko zostanie wykonane zwracamy wartość 0 i kończymy zapis obrazu do pliku.

```
int zapisz(FILE *plik_wy, t_obraz *obraz){
    if(obraz->format==2){ /*format=2 -> zapisujemy obraz PGM*/
        fprintf(plik_wy,"P2\n");  /*Magiczne "P2"*/
        fprintf(plik_wy,"%d %d\n%d\n",obraz->wym_y,obraz->wym_x,
    obraz->odcieni); /*zapisanie wysokosci*/

                    /*szerokosci, odcieni szarosci*/
        /*pomocniczy wskaznik na tablice dynamiczna typu int(*)[
    obraz->wym_x]*/
        int (*piksele)[obraz->wym_x]=(int(*)[obraz->wym_x]) obraz->
    piksele;
        /*zapisuje kazdy piksel oddzielony spacja w pliku*/
        for(int i=0; i<obraz->wym_y; i++){
            for(int j=0; j<obraz->wym_x; j++){
                fprintf(plik_wy,"%d ",piksele[i][j]);
            }
        }
    } /*format=3 -> zapisujemy obraz PPM*/
    else {
        fprintf(plik_wy,"P3\n");   /*Magiczne "P3"*/
        fprintf(plik_wy,"%d %d\n%d\n",obraz->wym_y,obraz->wym_x,
    obraz->odcieni); /*zapis wysokosci,szerokosci*/

                    /*wartosci skladowych kolorowych obrazu*/
        /*pomocniczy wskaznik na tablice dynamiczna typu int(*)[
    obraz->wym_x]*/
        int (*piksele)[obraz->wym_x*3]  =(int(*)[obraz->wym_x*3])
    obraz->piksele;
         /*zapisuje kazdy piksel oddzielony spacja w pliku*/
        for(int i=0; i<obraz->wym_y; i++){
            for(int j=0; j<obraz->wym_x*3; j++){
                fprintf(plik_wy,"%d ",piksele[i][j]);
            }
        }
        /*Zwalnianie zaalokowanej pamieci tu  przed zako czeniem
    programu(funckja zapisz wykonuje si  przedostatnia)*/
        /*po zapisaniu pikseli w pliku wyjsciowym, nalezy zwolnic
    pamiec*/
        free(obraz->piksele);
        obraz->piksele=NULL;
    }/*wszystko ok? -> brak bledow zwroc 0*/
    return W_OK;
}
```

Listing 8: Zapis obrazu do pliku

4.7 Wyswietlanie pliku

Wyświetlanie pliku wyjściowego następuje za pomocą programu display. Do funkcji jako argument wymagana jest tylko nazwa pliku jaki wyświetlić. Opcja odpowiedzialna za wyświetlanie to (-d). Wyświetlanie wykonuje się na samym końcu, o ile zostanie wywołane. Wyświetlamy tylko obraz przetworzony z pliku!!

```
1 /* Wyswietlenie obrazu o zadanej nazwie za pomoca programu "display
      "   */
2 void wyswietl(char *n_pliku){
3     char polecenie[DL_LINII];       /* bufor pomocniczy do
      zestawienia polecenia */
4
5     strcpy(polecenie,"display ");  /* konstrukcja polecenia postaci
       */
6     strcat(polecenie,n_pliku);     /* display "nazwa_pliku" &
       */
7     strcat(polecenie," &");
8     printf("%s\n",polecenie);        /* wydruk kontrolny polecenia
      */
9     system(polecenie);               /* wykonanie polecenia
      */}
```

Listing 9: Wyswietlanie obrazu

4.8 Operacje przetwarzania

Każda funkcja jest wywoływana ze wskaznikiem na strukture obraz, która zawiera wszystkie informacje o obrazie.Oczywiście w niektórych funckjach potrzebny jest jeden argument dodatkowy lub dwa (zmiana poziomow)).W każdej funkcji zostaje zainiciowana zmienna pomocnicza typu int(*)[obraz->wym x] lub int(*)[obraz wym x *3](dla obrazów PPM), która pozwala poruszać się po tablicy pikseli obrazu. W każdej fukcnji występuje sprawdzenie fomatu pliku,(rzecz oczywista jest, że całego obrazu formatu PPM nie da się przetworzyć, tylko jego jeden z kolorów). Gdy nie ma żadnych błędów każda funkcja zwraca wartość 0.

4.8.1 Negatyw

Funkcja działa według wzoru $G(x, y) = MAX - L(x, y)$, gdzie MAX=szarosci; G=piksel wynikowy,L=piksel wejściowy. W tej operacji nie ma żadnych szczególnych przypadków. Wywołanie opcji (-n) bez żadnych argumentów.

```
    /*operacja negatywu na ka dym z pikseli*/
    for(int i=0; i<obraz->wym_y; i++){
        for(int j=0; j<obraz->wym_x; j++){
            piksele[i][j]=obraz->odcieni-piksele[i][j];} }
```

Listing 10: Negatyw - Algorytm

4.8.2 Progowanie

Funkcja działa według wzoru:

$$G(x,y) = \begin{cases} 0 & dlaL(x,y) <= PROG \\ MAX & dlaL(x,y) > PROG \end{cases} \tag{1}$$

gdzie PROG=wartość progu, MAX=szarosci, G=piksel wynikowy,L=piksel wejściowy. W momencie wywołania opcji progowanie (-p) wymagany jest argument poziomu progu z zakresu od 0 do 100.Następnie próg zostaje obliczony i ustanowiony dla danego obrazu.Brak przypadków szczególnych.

```
    int prog=(w_progu*obraz->odcieni)/100;   /*przygotowanie progu
    do obliczen*/
    /*operacja progowanie na ka dym z pikseli*/
    for(int i=0; i<obraz->wym_y; i++){
        for(int j=0; j<obraz->wym_x; j++){
            if(piksele[i][j]<=prog){
                piksele[i][j]=0;
            }
            else piksele[i][j]=obraz->odcieni;}}
```

Listing 11: Progowanie - Algorytm

4.8.3 Korekcja gamma

Funkcja działa zgodnie ze wzorem:

$$G(x,y) = \left(\frac{L(x,y)}{MAX}\right)^{\frac{1}{\gamma}} \cdot MAX \tag{2}$$

gdzie G=piksel wynikowy, L=piksel wejściowy, MAX=szarosci, γ=wspolczynnik gamma Nasza funkcja zmienia wartość pikseli zgodnie z powyższym wzorem. Oczywiście mianownik nie ma możliwości się zerować.Wartość współczynnika gamma nie może sie zerować. Jeśli jest równy zero zostaniemy poinformowani błędem.Została użyta funkcja pow(), a co za tym idzie potrzebna biblioteka <math.h>. Wywołanie opcji (-g) z argumentem współczynnika gamma, który jest tyu float.

```
    if(w_gamma==0){ /*wspolczynnik gamma nie moze sie zerowac ->
    sprawdz*/
        fprintf(stderr,"\nBlad! WSpolczynnik gamma nie moze sie
    zerowac! Operacja sie nie powiedzie!\n\n");
        return B_BLEDNE_DANE;}
    float w=1/w_gamma;   /*zmienna pomocnicza wykladnik potegi */
    /*do ulatwienia obliczen*/
    float w2=(1-w_gamma)/w_gamma; /*zmienna pomocnicza wykladnik
    drugiej potefi*/
    /*operacja korekcji_gamma na kazdym pikselu*/
    for(int i=0; i<obraz->wym_y; i++){
        for(int j=0; j<obraz->wym_x; j++){
            piksele[i][j]=(pow(piksele[i][j],w))/(pow(obraz->
    odcieni,w2));}}
```

Listing 12: Korekcja gamma - Algorytm

4.8.4 Konturowanie

Funkcja działa zgodnie ze wzorem:

$$G(x,y) = |L(x+1,y) - (L(x,y)| + |L(x,y+1) - L(x,y)| \quad (3)$$

gdzie G=piksel wynikowy,L=piksel wejściowy. Użyto funckji abs(), więc również przydatna jest biblioteka <math.h> Mamy skłonność do napotkania granicznych przypadków gdy piksel[i][j] znajduje się na skraju tablicy(i=wymy-1).Wtedy piksel[i+1][j] nie istnieje.Dla tego przypadku wykonuje się tylko druga część operacji.Natomiast jeżeli piksel[i][j] znajduje się na drugim skraju tablicy(j=wymx-1) wtedy piksel[i][j+1] nie istnieje.Wkonuje się tylko pierwsza część operacji. A także gdy piksel[i][j] znajduje się na skraju krawędzi tablicy->(j=wymx-1)(i=wymy-1) to piksel po prostu się nie zmienia. Wywołanie opcji (-k) bez zbędnych argumentów.

```
    /*Operacja konturowania na ka dym pikselu*/
    for(int i=0; i<obraz->wym_y; i++){ /*Wazne jest sprawdzenie
    przypadkow szczegolnych*/
       for(int j=0; j<obraz->wym_x; j++){  /*aby nie wyjsc poza
    zakres tablicy*/
       if((i==obraz->wym_y-1)&&(j==obraz->wym_x-1)) piksele[i][j]=
    piksele[i][j];
       else if(i==obraz->wym_y-1) piksele[i][j]=abs(piksele[i][j
    +1]-piksele[i][j]);
       else if(j==obraz->wym_x-1) piksele[i][j]=abs(piksele[i+1][j
    ]-piksele[i][j]);
       else piksele[i][j]= abs(piksele[i+1][j]-piksele[i][j])+ abs(
    piksele[i][j+1]-piksele[i][j]);} }
```

Listing 13: Konturowanie - Algorytm

4.8.5 Rozciąganie histogramu*

Funkcja działa według wzoru:

$$G(x,y) = (L(x,y) - L_{MIN}) \cdot \frac{MAX}{L_{MAX} - L_{MIN}} \tag{4}$$

gdzie G=piksel wynikowy,L=piksel wejściowy,L_{MIN}=najmniejsza jasność występująca w obrazie wejściowym,L_{MAX}=największa jasność występująca w obrazie wejściowym,MAX=szarosci. Operacja nie zmieni obrazu jak widać gdy L_{MIN}=0 oraz L_{MAX}=szarosci, jeśli do tego nastąpi ukaże się komunikat.Wartość przedziału pikseli powinien znajdować się w przedziale L_{MIN}<L(x,y)<L_{MAX}.Oczywiście nie ma prawa nastąpić sytuacja gdy $L_{MAX} = L_{MIN}$->spowoduje to wyzerowanie mianownika.Funkcja w tym przypadku wyświetli ostrzegawczy komunikat.Najpierw wykonuje się pętla, która znajduje L_{MIN} oraz L_{MAX} w całej tablicy pikseli. Wywołanie opcji (-h) bez zbędnych argumentów.

```
1     int min=obraz->odcieni; /*pomocnicze parametry min and max*/
2     int max=0;          /*do wyszukania najmniejszj wartosci
      odcieni szarosci*/
3     /*Operacja wyszukiwania min and max w calej tablicy pikseli*/
4     for(int i=0; i<obraz->wym_y; i++){
5         for(int j=0; j<obraz->wym_x; j++){
6             if(piksele[i][j]>max)
7                 max=piksele[i][j];
8             if(piksele[i][j]<min)
9                 min=piksele[i][j];}
10    } /*max==min -> obraz sie nie zieni , wypisywanie komunikatu*/
11    if((max==obraz->odcieni)&&(min==0)) printf("\nPiksele pokrywaja
       caly zakres jasnosci obrazu!,Obraz sie nie zmienil!\n\n");
12    if(min!=max){ /*Operacja roz_histogramu wykona sie tylko gdy
      min!=max*/
13        for(int i=0; i<obraz->wym_y; i++){
14            for(int j=0; j<obraz->wym_x; j++){
15                piksele[i][j]=((piksele[i][j]-min)*obraz->odcieni)
      /(max-min);}}
16    } /*min==max? -> zwroc blad*/
17    else{
18        printf("\nBlad: Najmniejsza jasnosc = maksymalnej jasnosci!
       Operacja sie nie powiedzie!\n\n");
19        return B_BLEDNE_DANE;}
```

Listing 14: Rozciąganie histogramu - ALgorytm

4.8.6 Progowanie bieli

Funkcja działa według wzoru:

$$G(x,y) = \begin{cases} L(x,y) & dlaL(x,y) <= PROG \\ MAX & dlaL(x,y) > PROG \end{cases} \tag{5}$$

gdzie PROG jest wartością progu bieli, MAX makymalnym odcieniem szarości, G(x,y) nowym pikselem, a L(x,y) to pierwotny piksel. Brak przypadków szczególnych. Wywołanie funkcji (-pb) z argumentem wartości progu bieli.

```
int prog=(w_progu_bieli*obraz->odcieni)/100; /*przygotowanie
 prog bieli do obliczen*/
/*Operacja progowania_bieli na kazdym pikselu*/
for(int i=0; i<obraz->wym_y; i++){
     for(int j=0; j<obraz->wym_x; j++){
         if(piksele[i][j]>prog)
             piksele[i][j]=obraz->odcieni;}}
```

Listing 15: Progowanie bieli - ALgorytm

4.8.7 Progowanie czerni

FUnkcja działa według wzoru:

$$G(x,y) = \begin{cases} 0 & dlaL(x,y) <= PROG \\ L(x,y) & dlaL(x,y) > PROG \end{cases} \tag{6}$$

gdzie P ROG jest wartością progu czerni, MAX makymalnym odcieniem szarości, G(x,y) nowym pikselem, a L(x,y) to pierwotny piksel.Brak przypadków szczególnych. Wywołanie opcji (-pc) z arguemntem wartości progu czerni.

```
int prog=(w_progu_czerni*obraz->odcieni)/100; /*Przygotowanie
 progu czerni do obliczen*/
for(int i=0; i<obraz->wym_y; i++){
     for(int j=0; j<obraz->wym_x; j++){
         if(piksele[i][j]<=prog)
             piksele[i][j]=0;}}
```

Listing 16: Progowanie czerni - Algorytm

4.8.8 Rozmywanie poziome

Funckja działa według wzoru:

$$G(x,y) = (L(x-1,y) + L(x,y) + L(x+1,y)) \cdot \frac{1}{3} \tag{7}$$

gdzie G(x,y) jest nowym pikselem, L(x,y) pierwotnym pikselem, a pozostałe dwa znajdują się obok aktualnie przetwarzanego piksela.W tej funkcji została zaalkowana kolejna tablica, przechowująca kopie pikseli obrazu. ZOstała wykonana w celu poprawnych obliczeń. Każdy piksel powstaje z z sumy trzech pikseli(aktualnie przetwarzanego poprzedniego oraz następnego), więc poprzedni piksel jeśli został przetworzony, a przetwarzanie wykonuje się na kolejnym pikselu, który potrzebuje wartości poprzedniego piksela, a on został zmieniony, tak więc tablica przechowuje kopie, aby nie było niejasności z obliczeniami.Wywołanie funkcji (-rx) bez zbędnych argumentów.Po zakończeniu funkcji zwalniana jest dodatkowa pamięć, gdyż nie jest już potrzebna.

```
1  /*kopia pierwotnej tablicy wymagana do obliczen pikseli*/
2  /*(po przetworzeniu piksela w nastepnym obok pikselu w
     obliczeniach wymagany jest piksel poprzedni pierwotny)*/
3  int (*pomocna_tablica)[obraz->wym_x]=(int (*)[obraz->wym_x])
     malloc(obraz->wym_x*obraz->wym_y*sizeof(int));
4  if(pomocna_tablica==NULL){ /*nie udalo sie przydzielic pamiec? ->
      zwroc blad*/
5       free(pomocna_tablica);
6       fprintf(stderr,"\nBlad, Brak przydzialu pamieci\n\n");
7       return B_BRAK_PAMIECI;
8  }
9  for(int i=0; i<obraz->wym_y; i++){
10      for(int j=0; j<obraz->wym_x; j++){
11          pomocna_tablica[i][j]=piksele[i][j];}}
12 /*Operacja rozmywanie_poziome na kazdym pikselu*/
13 for(int i=0; i<obraz->wym_y; i++){ /*Wazne jest sprawdzenie
     przypadkow szczegolnych*/
14      for(int j=0; j<obraz->wym_x; j++){ /*aby nie wyjsc poza
     zakres tablicy*/
15          if(j==0) piksele[i][j]=(piksele[i][j]+piksele[i][j+1])/2;
16          else if(j==obraz->wym_x-1) piksele[i][j]=(piksele[i][j]+
     pomocna_tablica[i][j-1])/2;
17          else piksele[i][j]=(pomocna_tablica[i][j-1]+piksele[i][j
     ]+piksele[i][j+1])/3;
18      }
19 }/*zwalnianie dodatkowej pamieci na potrzeby funckji*/
20 free(pomocna_tablica);
21 pomocna_tablica=NULL;
```

Listing 17: Rozmywanie poziome - Algorytm

4.8.9 Rozmywanie pionowe

FUnckja działa według wzoru:

$$G(x,y) = (L(x,y-1) + L(x,y) + L(x,y+1)) \cdot \frac{1}{3} \qquad (8)$$

gdzie G(x,y) jest nowym pikselem, L(x,y) pierwotnym pikselem, a pozostałe dwa znajdują się obok (pod i nad)aktualnie przetwarzanego piksela.W tej funkcji została zaalkowana kolejna tablica, przechowująca kopie pikseli obrazu. ZOstała wykonana w celu poprawnych obliczeń. Każdy piksel powstaje z z sumy trzech pikseli(aktualnie przetwarzanego poprzedniego(z dołu) oraz następnego(z góry)), więc poprzedni piksel jeśli został przetworzony, a przetwarzanie wykonuje się na kolejnym pikselu, który potrzebuje wartości poprzedniego piksela, a on został zmieniony, tak więc tablica przechowuje kopie, aby nie było niejasności z obliczeniami.Wywołanie funkcji (-ry) bez zbędnych argumentów.Po zakończeniu funkcji zwalniana jest dodatkowa pamięć, gdyż nie jest już potrzebna.

```
/*kopia pierwotnej tablicy wymagana do obliczen pikseli*/
/*(po przetworzeniu piksela w nastepnym obok pikselu w
  obliczeniach wymagany jest piksel poprzedni pierwotny)*/
int (*pomocna_tablica)[obraz->wym_x]=(int (*)[obraz->wym_x])
  malloc(obraz->wym_x*obraz->wym_y*sizeof(int));
if(pomocna_tablica==NULL){ /*nie udalo sie przydzielic pamiec? ->
   zwroc blad*/
    free(pomocna_tablica);
    fprintf(stderr,"\nBlad, Brak przydzialu pamieci\n\n");
    return B_BRAK_PAMIECI;}
for(int i=0; i<obraz->wym_y; i++){
    for(int j=0; j<obraz->wym_x; j++){
        pomocna_tablica[i][j]=piksele[i][j];}}
/*Operacja rozmywanie_pionowe na kazdym pikselu*/
for(int i=0; i<obraz->wym_y; i++){ /*Wazne jest sprawdzenie
  przypadkow szczegolnych*/
    for(int j=0; j<obraz->wym_x; j++){ /*aby nie wyjsc poza
  zakres tablicy*/
        if(i==0) piksele[i][j]=(piksele[i][j]+piksele[i+1][j])/2;
        if(i==obraz->wym_y-1) piksele[i][j]=(piksele[i][j]+
  pomocna_tablica[i-1][j])/2;
        else piksele[i][j]=(piksele[i+1][j]+piksele[i][j]+
  pomocna_tablica[i-1][j])/3;}
}/*zwalnianie dodatkowej pamieci na potrzeby funckji*/
free(pomocna_tablica);
pomocna_tablica=NULL;

```

Listing 18: Progowanie pionowe - Algorytm

4.8.10 zmiana poziomów

Funckja działa według wzoru:

$$G(x,y) = \begin{cases} 0 & dlaL(x,y) <= CZERN \\ (L(x,y) - CZERN) * \frac{MAX}{BIEL-CZERN} & dlaCZERN < L(x,y) < BIEL \\ MAX & dlaL(x,y) >= BIEL \end{cases} \tag{9}$$

gdzie BIEL jest nowym poziomem bieli, CZERN nowym poziomem czerni, zaś MAX maksymalną szarością pikseli obrazu. Zgodnie ze wzorem funkcji BIEL nie może się równać CZERNI. Gdy jednak BIEL = CZERN mamy błąd i komunikat.Wywoływanie funkcji (-z) i koniecznie dwa argumenty CZERN I BIEL w zadanej kolejności.

```
1  int biel=(bialy*obraz->odcieni)/100; /*Przygotwanie bieli i
     czerni do obliczen*/
2  int czern=(czarny*obraz->odcieni)/100;
3  if(biel==czern){  /*Biel nie moze sie rownac czerni (biel=czern
     -> sprawdz)*/
4      fprintf(stderr,"\nBlad!! czern nie moze byc rowna bieli!,
     Przetwarzanie si  nie wykona.\n\n");
5      return B_BLEDNE_DANE;}
6  /*Operacja zmiany_poziomow na ka dym pikselu*/
7  for(int i=0; i<obraz->wym_y; i++){
8        for(int j=0; j<obraz->wym_x; j++){
9          if(piksele[i][j]<=czern)
10             piksele[i][j]=0;
11         else if(piksele[i][j]>=biel)
12             piksele[i][j]=obraz->odcieni;
13         else piksele[i][j]=(piksele[i][j]-czern)*obraz->odcieni/(
     biel-czern);}}
```

Listing 19: Zmiana poziomów - ALgorytm

4.8.11 Konwersja do szarości

Funckja wywoływana opcja -m z argumentem (s). Należy zaznaczyć ,że wczytany plik musi być formatu PPM. FUnkcja z 3 składowych pikseli robi średnią arytmetyczną i w ten sposób powstaje obraz odcieni szarości, zmienia format na PPM, więc może zostać dalej przetworzony.Gdy wywołamy funkcje konwersji dla obrazu PGM zostanie wyrzucony błąd.W tej funkcji po przetworzeniu korzytsamy z funckji realloc, która znajduję dla nas nową pamięć (mniejszą) , robi oryginalną kopię,i zwalnia nadmiarową pamięć z użytku. Funckja kończy się zwracając wartość 0 -> brak błędów.

```
1  int konwersja_do_szarosci(t_obraz *obraz){
2    /*Jezeli obraz jest formatu PGM -> nie mozna skonwertowac obrazu
       */
```

```
3   if(obraz->format!=3){ /*Mamy obraz PGM? ->zwroc blad*/
4       fprintf(stderr,"\nBlad Nie wczytano kolorowego obrazu, Nie
      mozna wykonac operacji konwersji do szarosci!\n\n");
5       return B_BLEDNE_DANE;
6   }
7   /*pomocniczy wskaznik na tablice dynamiczna typu int(*)[obraz->
      wym_x*3]*/
8   int (*piksele)[obraz->wym_x*3]=(int(*)[obraz->wym_x*3]) obraz->
      piksele;
9   int k=0; /*Zmienne pomocnicze do nadpisu tablicy*/
10  int w=0;
11  /*Operacja konwersji pikseli*/
12  /*Nowy piksel formatu PGM jest srednia arytmetyczna 3 skladowych
      piksela formatu PPM*/
13  for(int i=0; i<obraz->wym_y; i++){
14      for(int j=0; j<obraz->wym_x*3; j+=3){
15          piksele[k][w]=(piksele[i][j]+piksele[i][j+1]+piksele[i][j
      +2])/3;
16          w++;
17          if(w==obraz->wym_x*3){
18            k++;
19            w=0;
20          }
21      }
22  }/*Po dokonaniu zmiany pliku z PPM na PGM format=2 -> plik PGM*/
23  obraz->format=2;
24  /*zwalnianie pamieci nadmiarowej-> funkcja realloc znajduje nowy
      obszar pamieci, nastepnie kolejno kopiuje dane oryginalne*/
25  /*w nowe miejsce, zwalnia stary obszar pamieci i zwraca wskaznik
      do nowego obszaru lub NULL*/
26  obraz->piksele=realloc(obraz->piksele,obraz->wym_x*obraz->wym_y*
      sizeof(int));
27  if(obraz->piksele==NULL){        /*Nie uda o si  przydzieli
      pami ci? -> zwroc b ad*/
28    free(obraz->piksele);
29    fprintf(stderr,"\nBlad przydzialu pamieci\n\n");
30    return B_BRAK_PAMIECI;
31  }
32  /*wszystko ok? -> brak bledow zwroc 0*/
33  return W_OK;
34 }
```

Listing 20: Konwersja do szarości

4.8.12 Przetwarzaj kolor

Funckja jest wywoływana opcją -m i należy podać jeden z trzech kolorów, jaki chcemy przetworzyć: czerwony(r), zielony(g) lub niebieskie(b).Funckja sprawdza czy format pliku to PPM, następnie inicjujemy dodatkową strukturę, do której zapisujemy wskaznik na pamięć tylko dla jednego koloru.Mamy dwa pomocne wskazniki pomocna tablica -> wskazuje na pamięć z jednym kolorem, oraz wskaznik na pierwotną tablicę(piksele).W zależności od koloru pętla zapisuje do

nowej pamięci składowe piksela jednego koloru, aby moć potem wywołać funckje przetwarzającą tylko ten jeden kolor.Gdy nie podano żadnej operacji do przetworzenia tego koloru to nie ma sensu i wyrzucany jest błąd. Na samym końcu te przetworzone piksele nadpisują pierwotne piksele tylko w tych określonych miejscach.Oczywiście nie zabrakło zwolnienia dodatkowej pamięci.Operacje na kolorze są wykonywane w kolejności wywyołania(posługujemy się tablicą argv).Program powinien się zakończyć z kodem 0 -> brak błędow.

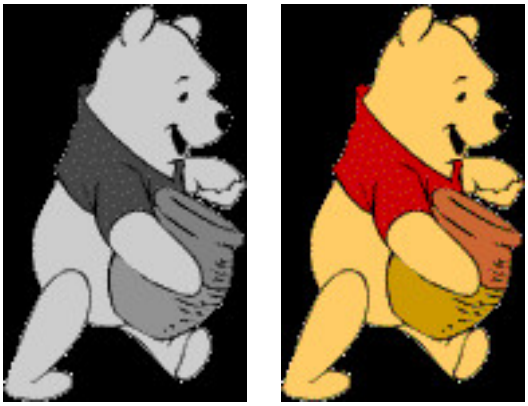
```
1  int przetwarzaj_kolor(t_opcje *opcje,t_obraz *obraz, char kolor,
       int argc, char **argv){
2    t_obraz pojedynczy_kolor;
3    pojedynczy_kolor.wym_x=obraz->wym_x;
4    pojedynczy_kolor.wym_y=obraz->wym_y;
5    pojedynczy_kolor.odcieni=obraz->odcieni;
6    pojedynczy_kolor.format=2;
7    /*alokowanie pamieci do przechowywania tylko skladowych jedngo
       koloru kazdego piksela*/
8    pojedynczy_kolor.piksele=malloc(obraz->wym_x*obraz->wym_y*sizeof(
       int));
9    if(pojedynczy_kolor.piksele==NULL){ /*nie udalo sie przydzielic
       pamiec? -> zwroc blad*/
10     free(pojedynczy_kolor.piksele);
11     fprintf(stderr,"\nBlad, Brak przydzialu pamieci\n\n");
12     return B_BRAK_PAMIECI;}
13   int (*piksele)[obraz->wym_x*3]=(int(*)[obraz->wym_x*3]) obraz->
       piksele;
14   int (*pomocna_tablica)[obraz->wym_x]=(int(*)[obraz->wym_x])
       pojedynczy_kolor.piksele;
15   int k=0,w=0; /*zmienne iteracyjne*/
16   if(kolor=='r'){ /*wybrano kolor czerwony*/
17       for(int i=0; i<obraz->wym_y; i++){ /*zapisywanie czerwonych
       skladowych pikseli obrazu do struktury pojedynczy.kolor*/
18           for(int j=0; j<obraz->wym_x*3; j+=3){
19               pomocna_tablica[k][w]=piksele[i][j];
20               w++;
21               if(w==obraz->wym_x){
22                 k++;
23                 w=0;}}}
24   /*wykonywanie operacji przetwarzania na tej jednej skladowej*/
25      for(int i=0; i<argc; i++){ /*pos ugiwanie si  tablic  a
       argumentami argv*/
26       switch(argv[i][1]){       /*Pozwala to na wykonywanie
       operacji w zadanej kolejno ci*/
27           case 'n':{
28                negatyw(&pojedynczy_kolor);
29                break;}
30           case 'p':{
31               switch(argv[i][2]){
32                   case '\0':{
33                       progowanie(&pojedynczy_kolor,opcje->w_progu);
34                       break;}
35                   case 'c':{
```

```
36                      progowanie_bieli(&pojedynczy_kolor,opcje->
    w_progu_czerni);
37                      break;}
38                  case 'b':{
39                      progowanie_czerni(&pojedynczy_kolor,opcje->
    w_progu_bieli);
40                      break;}}
41              break;}
42          case 'g':{
43              korekcja_gamma(&pojedynczy_kolor,opcje->w_gamma);
44              break;}
45          case 'z':{
46              zmiana_poziomow(&pojedynczy_kolor,opcje->czern,opcje
    ->biel);
47              break;}
48          case 'k':{
49              konturowanie(&pojedynczy_kolor);
50              break;}
51          case 'r':{
52              switch(argv[i][2]){
53                  case 'x':{
54                      rozmywanie_poziome(&pojedynczy_kolor);
55                      break;}
56                  case 'y':{
57                      rozmywanie_pionowe(&pojedynczy_kolor);
58                      break;} }
59              break; }
60          case 'h':{
61              roz_histogramu(&pojedynczy_kolor);
62              break;}
63          default: break;}}
64        w=0; k=0; /*na samym koncu zmiany zostaja zapisane do
    pierwotnej tablicy zawierajacej wszytskie skladowe pikseli*/
65        for(int i=0; i<obraz->wym_y; i++){
66          for(int j=0; j<obraz->wym_x*3; j+=3){
67              piksele[i][j]=pomocna_tablica[k][w];
68              w++;
69              if(w==obraz->wym_x){
70                k++;
71                w=0;}}}}
72  free(pojedynczy_kolor.piksele);
73  pojedynczy_kolor.piksele=NULL;
74  return W_OK;}
```

Listing 21: Fragment funkcji przetwarzaj kolor

5 Testy

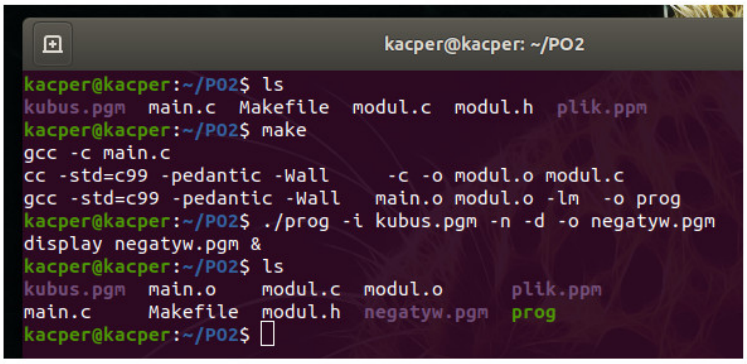
Testy działania funkcji przetwarzających zostały wykonane na jednym pliku PGM "kubus.pgm" oraz na drugim pliku PPM "kubus.ppm".



Rysunek 1: Plik PGM kubus.pgm oraz PLik PPM kubus.ppm

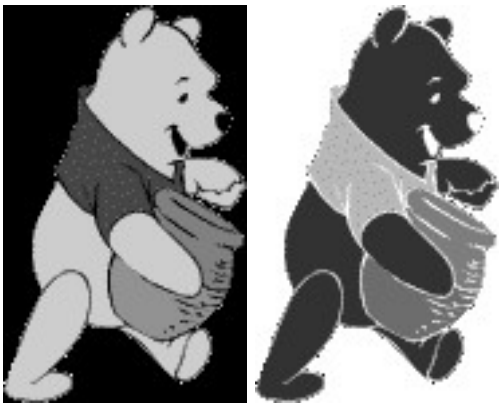
5.1 TEST 1

Pierwszy test bardzo podstawowy sprawdza zgodność działania funckji negatyw, wczytaj, zapisz oraz wyświetlanie pliku. Na wejściu zostały wykonane następujące czynności:



Rysunek 2: Wywołanie programu(pierwszy test

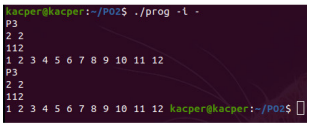
Przedstawiono działanie Makefila, który poprawnie skompilował wszystkie pliki w jedną całość. Wywołano opcje wczytania pliku "kubus.pgm", wyświetlono go po przetworzeniu, został wykonany negatyw oraz zapisano plik pod inną nazwą.Na samym końcu widać pliki wykonawcze i plik zapisany pod nazwą negatyw.pgm.Program nie zwrócił żadnych błędów.Wyniki nie są dla nas zaskakujące, negatyw, zapis, wczyt oraz wyświetlenie pliku można uważać,że działają poprawnie.Oto wynik operacji negatyw:



Rysunek 3: Negatyw

5.2 TEST 2

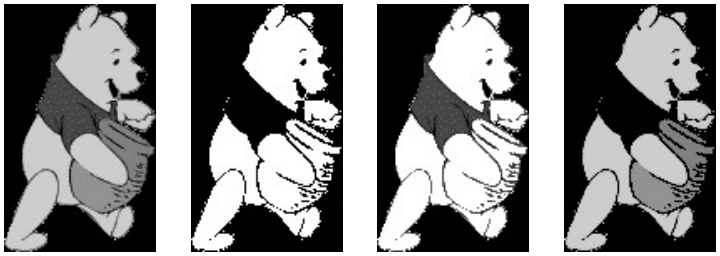
Drugi test wywołuje program bez podawania pliku do wczytania tylko znak "-" (wczytujemy ze standardowego wejścia,a obraz zostanie wyrzucony na standardowe wejście (nie podajemy opcji -o) Jak widać po wczytaniu przypadkowego obrazu przez nas z standardowego wejścia, został przekierowany na standardowe wyjscie. Rezultaty są poprawne i oczywiste. Wynik jest zadowalający, w przypadku braku opcji -i następuję błedny komunikat: Blad, Nie otwarto pliku wejsciowego.



Rysunek 4: wywołanie programu(test2

5.3 TEST 3

W tym teście skupimy się na trzech operacjach(wykonijemy je osobno jedna za drugą), sprawdzimy działanie operacji progowania, progowania bieli oraz progowania czerni, za każdym razem wyświetlimy obraz po przetworzeniu. Wczytamy plik "kubus.pgm" opcją -i, następnie -d, -p 50 -o progowanie.pgm i tak rownież dla dwóch pozostałych funckji (-pb i -pc). Sprawdzimy wywoływanie fukncji z argumentem progu, w każdym przypadku równy 50. argument musi być liczbą całkowitą, dopiero w funkcji zostaje zamieniony na te 50 procent do obliczeń. Gdy nie pdoamy argumentu, bądź opcja jest błedna, zostaje wyrzucony błąd. Na wyjściu dostaliśmy trzy pliki zapisane każdy pod inną nazwą. Obrazy zostały prawidłowo przetworzone.Brak zwracanych błędów, wynik jest zadowalający.Oto wyniki:

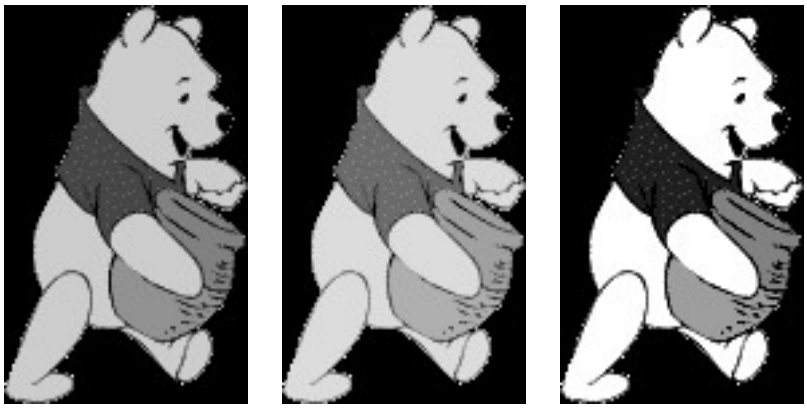


Rysunek 5: Progowanie,progowania bieli i czerni dla 50

5.4 TEST 4

W tym przypadku również zbadamy działanie dwóch innych funkcji wymagających podania argumentu przy wywołaniu: są to kolejno korekcja gamma oraz zmiana poziomów. Wczytujemy zawsze ten sam plik "kubus.pgm", będziemy mieli zamiar go wyświeltlić i zmienić najpierw operacją gamma, następnie tą drugą. Korekcja gamma wymaga argumentu typu float, zaś funkcja zmiana poziomów wymaga aż dwa argumenty: prog czerni i bieli(pierwszy argument jest progiem czerni)Przy braku argumentów zostają wyrzucane błędy jak wiadomo, wywołujemy program komendą: ./prog -i kubus.pgm -d -g 1.5 -o korekcjagamma1.5.pgm, w przypadku zmiany poziomow -> -z 20 80 (dla wartości 20 i 80). Należy pamiętać, że współczynnik gamma nie może sie zerować oraz BIEL nie moze sie równać CZER-

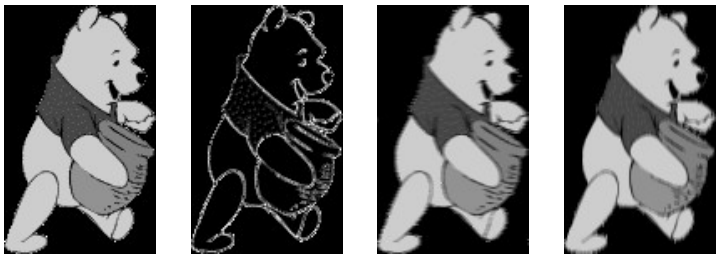
NI.Będą zwracane błedy w przeciwnym wypadku.Wyniki są również zadowalające, nie zwrócono błędów, i pomyślnie zapisano dwa pliki. Oto rezultaty przetworzonych obrazów:



Rysunek 6: Korekcja gamma(1.5),zmiana poziomów (20 80)

5.5 TEST 5

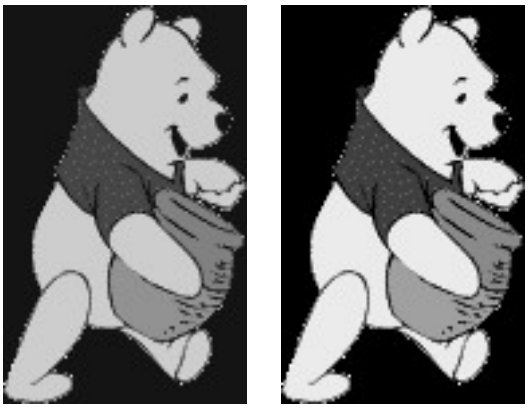
Znów się skupimy na jak ważnych dla nas kolejnych trzech funkcjach konturowanie, rozmywanie w poziomie oraz w pionie. Wszystkie opcje nie wymagają zbędnych argumentów. Każda operacja jak zwykle po wykonaniu zostanie wyświetlona i zapisana. Wczytujemy plik "kubus.pgm", oto opcje wywołania: ./prog -i kubus.pgm -d -k -o konturowanie.pgm, podobnie pozostałe dwie funckje ale z zamaist opcji -k mamy -rx lub -ry. i inne nazwy plików wyjściowych.PO wszystkim nasz program wyświetlił 3 przetworzone obrazy osobno jeden po drugim, zapisał je i nie zwrócił ani jednego błędu. Te wyniki pozwalają przypuszczać, że każda z tych funkcji działa poprawnie.Nasze przetworzone obrazy można zobaczyć niżej.



Rysunek 7: Konturowanie,rozmywanie poziome i pionowe

5.6 TEST 6

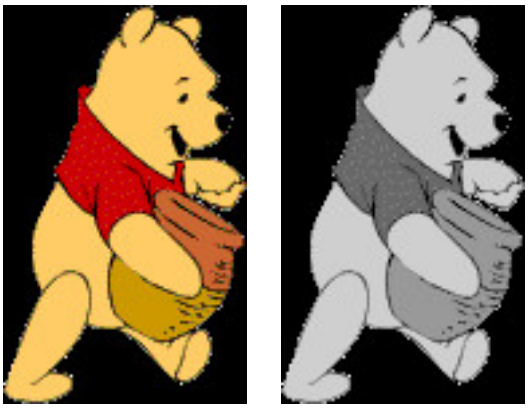
Nasza wyróżniająca się operacja rozciąganie histogramu została wykonana kilkakrotnie, ze względu na jej specyfikę. Jak pamiętamy gdy piksel o minimalnej odcieni = 0 oaz piksel o maksymalnej odcieni = odcieni szarośći maksymalnej obraz się nie zmieni, więc specjalnie przerobiliśy plik "kubus.pgm" na potrzebę tej funkcji, aby piksele jego zawierały się od 20 do 220 wtedy obraz się może zmienić. wywołanie programu było następujące: -i kubushistogram.pgm -d -h -o histogram.pgm. Opcja nie wymaga zbędnego parametru, wyświetlić obraz zawsze warto i sprawdzić zmiany. Gdyby zdarzyło się, że minimalny piksel obrazu równa się makymalnemu to mamy błąd, który zosatnie wyrzucony z komunikatem. Na wyjściu funckja wyświetliła obraz za pomocą funkcji display, zapisała plik pod zadaną nazwę oraz poprawnie dokonała rozciągania histogramu obrazu. Wyniki mogą dać do zrozmienia, że rozciąganie histogramu działa nie najgorzej.Oto obraz przed i po operacji:



Rysunek 8: Rozciąganie histogramu

5.7 TEST 7

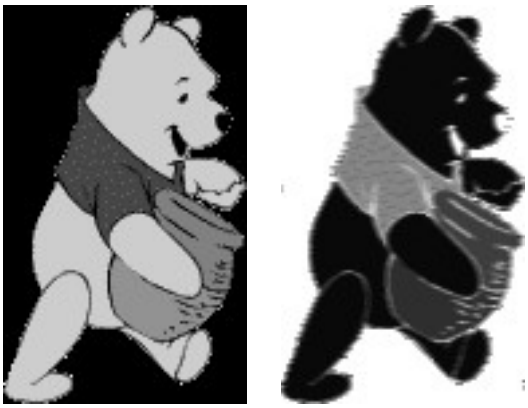
W tym teście zajmiemy się operacjami na obrazie PPM. Wczytujemy podstawowy plik wcześniej przygotowany formatu ppm. Mamy zamiar go skonwertować do szarości i następnie wykonać na nim dowolną operacje przetwarzania, w tym przypdaku jest to korekcja gamma, która w wyniku wywyołania opcją -g wymaga argumentu, podany został arguemnt 1.5.Oto wywołanie programu: ./prog -i kubus.ppm -d -m s -g 1.5 -o kubus.pgm. Jak zawsze plik wyświetlimy oraz zapiszemy pod zadaną nazwę. Na wyjściu program display wyświetlił nam nasz przetworzony obraz, plik został zapisany, a obraz z kolorowego zamienił się na odcieni szarości. Wykonaliśmy kilka operacji na jednym obrazie. Wynik testu jest na plus. Program nie zwrócił ani jednego błędu.Obraz przed konwersją i po przedstawiono poniżej:



Rysunek 9: Konwersja oraz korekcji gamma o współczynniku 1.5

5.8 TEST 8

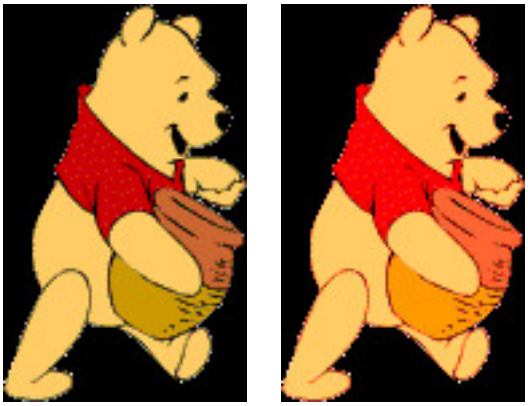
Test ma na celu sprawdzić czy możliwe jest wykonanie 3 różnych operacji przetwarzania na jednym obrazie kubus.pgm. za pomoć jednego wywołania programu: -i kubus.pgm -d -n -rx -g 0.5 -o kubus3.pgm. Jak widać wykonamy operacje negatywu, rozmywania poziomegooraz korekcji gamma 0.5. DOdatkowo wyświetlimy obraz na samym końcu, przetworzony plik zapiszemy pod nazwą kubus5.pgm. Ważne jest, że operacje negatywu, progowania oraz zmiany poziomów wykonują się w kolejności wywołania.Program zwrócił nam obraz zapisany w pliku kubus5.pgm, plik został zmieniony, błędów nie wykryto.Nasz program może wykonywać wiele operacji na jedynm pliku. Oto wyniki:



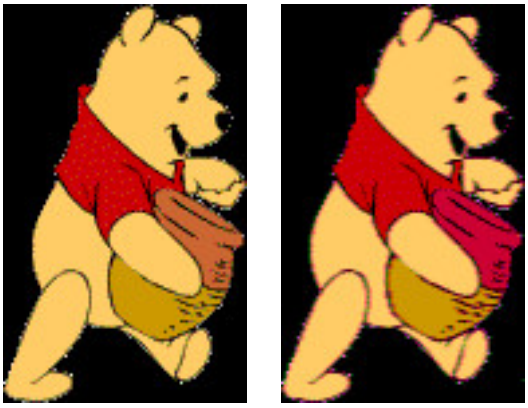
Rysunek 10: Negatyw, rozmywanie poziome, korekcja gamma (0.5)

5.9 TEST 9

W tym teście przetworzymy kolor czerwony i niebieski pliku kubus.ppm. Dokonamy operacji progowanie czerni oraz korekcji gamma za jednym razem, nie zapomijmy o wyświetlenie wyników. Oto wywołanie programu: -i kubus.ppm -m r -pc 50 -g 2.5 -d -o kubus3.ppm.Argumenty funkcji progowania czerni oraz korekcji gamma to następująco: 50 procent, 2.5.Wykonujemy tylko na jednym kolorze czerwonym dwie operacje.Następnie na kolorze niebieskim wywołamy program dla dwóch innych opcji: -i kubus.ppm -o kubus4.ppm -m g -pc 50 -rx -d. Progowanie bieli dla 50 procent oraz rozmywanie poziome.Program zwrócił nam plik kubus3.ppm -> jest to kubuś przetworzony oraz plik kubus4.ppm. Obraz się wyświetił za jednym i drugim razem, program nie zwrócił błędów, Nasze wyniki, jaki i cały test pokazuje nam ogromną funkcjonalność programu.Działa on nie najgorzej.Oto obrazy przed i po:



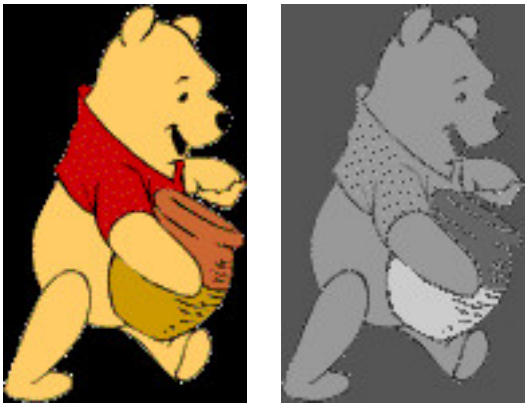
Rysunek 11: Przetwarzanie koloru czerwonego za pomcoą operacji progowania czerni (50) oraz korekcji gamma (2.5)



Rysunek 12: Przetwarzanie koloru zielonego za pomcoą operacji progowania bieli (50) oraz rozmywania poziomego

5.10 TEST 10

Nasz ostatni test zmieni obraz PPM w każdy możliwy sposób -> dokonamy przetworzenia koloru czerwonego za pomocą funkcji negatywu oraz progowania bieli dla 80 procent, następnie zmienimy obraz PPM na PGM czyli dokonamy konwersji. WYświetlimy wyniki i zapisujemy plik.Wywołanie jest następujące: -i kubus.ppm -o kubus2.ppm -d -n -p 80 -m b -m s. Jak wiadomo najpierw zostanie wczytany plik, zapis i wyświetlenie wykona się na końcu. Pierwszeństwo ma zawsze przetwarzanie jedngo koloru od konwersji.Oczekujemy, że zostanie dokonany negatyw oraz progowanie bieli 80 procent dla niebieskich składowych pikseli. następnie następuje konwersja i plik zmienia format na PGM. Obraz się wyświetił na końcu, program nie zwrócił błędów, wywołanie było prawidłowe. Obraz został przetworzony i zmieniony i oczywiście zapisany. Ostatni test jest decydujący i daje pozytywny wynik operacji na pliakch PPM. Oto rezultaty przetworzenia obrazu kubus.ppm:



Rysunek 13: Przetwarzanie koloru niebieskiego i komwersja na PGM

6 Wnioski

Program Przetwarzanie obrazów 2 jest wydajniejszy, bardziej uporządkowany oraz elastyczniejszy niż program Przetwarzanie obrazów 1. Jego wielofunkyjne działanie pozwala na więcej działań na obrazach formatu PGM oraz na naszych nowych obrazach PPM. Jak wcześniej było wspomniane program pozwala alokować dynamicznie pamięć oraz korzysta ze zmiennej typu stuktura, która umożliwia nam przechowywania wszystkich zmiennych różnego typu, ktore są ze sobą powiązane w jednym miejscu. Asercje jako nowy element, który się również pojawił sprawdza różne niedociągnięcia i niedopatrzenia.Wyrzuca błąd z komunikatem jaki błąd nastąpił. Daje to użytkownikowi możliwość zastanowienia się i wywołania programu z poprawnymi opcjami. Opcje jak i ich wywoływanie dodane do programu są pewnego rodzaju nakierunkowaniem jaka funkcja ma zostać wywołana i użyta, aby dany obraz zmienić, przetworzyć.Ważne jest podanie opcji -i z argumentem '-' bądź nazwą pliku wejściowego. Bez tego program nie zadziała poprawnie. Zapis jest wykonywany niemal za każdym razem albo do wskazanego pliku albo jest wypisywany na standardowe wyjście.W zależności od wywołania możemy przetworzyć obraz za pomocą 10 funkcji plus konwersja dla obrazów PPM. Obrazy PPM nie można zmienić odrazu.Funckje wykonywane są w kolejnościc wywołania programu, jedynie wyświetlanie wykonuje się na końcu(wyświetlamy tylko przetworzony obraz).Program pozwala wykonywać wielokrotnie jedną operację bądź kilka na raz.Stara się zwalniać bądź redukować pamięć jak tylko to możliwe. Co ważne plik z kodem został podzielony na moduły. Mamy trzy pliki kodu plus jeden Makefile, który ułatwia nam kompilacje i zmniejsza wykorzystywany na to czas.Testy zostały wykonane w celu sprawdzenia każdej możliwej funkcji programu. Wczytywanie błędnych danych odpuszczono, bo są to przypadki dość oczywiste, że wyrzucony zostanie błąd wraz z nim komunikat.Program jest w stanie wykonywać przetwarzanie tylko na jednym wczytanym obrazie.Należy uważnie i rozważnie wczytywać argumenty funkcji, aby nie wykryto niechcianego błędu.Po dokonaniu koniecznych testów i operacji na obrazach kubus.pgm oraz kubus.ppm można śmiało wnioskować,że progra Przetwarzanie Obrazów 2 jest ulepszoną wersją PO1 oraz jego działanie jest uporządkowane i elastyczne.Program spełnia swoje zadanie i wykonuje się poprawnie.