Politechnika Wrocławska

Przetwarzanie obrazów I

Autor: **mgr. inż. Piotr Lampa**

Wstęp

Projekt polega na stworzeniu programu, który będzie wczytywał obrazy w formacie .PGM, tworzył dla nich strukturę danych i alokował pamięć. Obrazy będą poddawane przetwarzaniu, a przetworzony obraz ma mieć możliwość zapisu na dysku. Projekt przewidziany jest na 3 zajęcia.

Poniżej znajdują się zadania projektu oraz omówienie zagadnień koniecznych do ich zrealizowania. Zaznaczone zadania należy zrealizować na kolejne zajęcia.

Projekt: Przetwarzanie obrazów – część I

1. Menu użytkownika.
2. Funkcję wczytującą plik .PGM z zapytaniem o nazwę pliku i obsługą komentarzy.
3. Deklarację struktury na obraz oraz zmiennej strukturalnej na obraz i jej wypełnienie.
4. Dynamiczną alokację dwuwymiarowej pamięci na obraz wraz zabezpieczeniem przed wyciekiem pamięci.
5. Funkcję zapisującą plik .PGM z dowolnym obrazem z bazy z zapytaniem o nazwę pliku.
6. Dynamiczną tablice do przechowywania w pamięci i obsługi wielu obrazów równocześnie, pozwalającą na:
    - Dodanie (wczytanie) obrazu do tablicy;
    - Usunięcie obrazu z tablicy;
    - Wyświetlenie listy obrazów;
    - Wybranie „aktywnego” obrazu z tablicy, na którym mogą być wykonywane przetwarzania, zapis do pliku itd.
7. Funkcje przetwarzania obrazów:
    - Odbicie względem osi, obrót o 90·k stopni;
    - Histogram z zapisem do pliku .CSV;
    - Progowanie z podanym przez użytkownika progiem, negatyw;
    - Dodawanie szumu typu pieprz i sól, filtr Gaussa i filtr medianowy w oknie 3x3.
8. DLA AMBITNYCH:
    - Wyrównanie histogramu obrazu, umieszczenie histogramu na oryginalnym obrazie;
    - Powiększanie/pomniejszanie obrazu;
    - Splot obrazu dla dowolnego okna 3x3.

Omówienie zagadnień

Format .PGM

Plik .PGM, czyli *Portable Gray Map*, to plik graficzny, gdzie odcień szarości każdego piksela jest zapisany jako liczba, co czyni dobrym przykładem do nauki obsługi plików. Specyfikacja formatu .PGM znajduje się na http://netpbm.sourceforge.net/doc/pgm.html .

Istnieją dwa standardy formatu .PGM, *P5* – binarny, *P2* – ASCII. Bardziej intuicyjny dla człowieka jest ten drugi standard, dlatego będziemy skupiać się tylko na nim.

Każdy plik składa się z nagłówka, w którym znajdują się:

- Numer oznaczający standard (P2/P5);
- Biały znak (spacja, nowa linia, tabulator…);
- Szerokość obrazu – całkowita liczba dodatnia;
- Biały znak;
- Wysokość obrazu – całkowita liczba dodatnia;
- Biały znak;
- Maksymalna wartość szarości (głębia szarości) – od 1 do 65535;
- Biały znak (zwykle nowa linia);
- Piksele przyjmujące wartość od 0 do maksymalnej wartości szarości, rozdzielone znakami białymi, począwszy od pierwszego wiersza od góry obrazu.

Wartość piksela oznacza poziom szarości proporcjonalny do maksymalnej wartości, czyli na przykład: maksymalna wartość szarości to 100, piksele o wartości 0 będą wówczas czarne, o wartości 100 białe, pomiędzy tymi wartościami szare – im większa wartość tym „bliżej" białego.

Poniżej przykład pliku .PGM zawierający reprodukcję obrazu „Czarny kwadrat na białym tle" Kazimierza Malewicza, obok plik przestawiający gradient szarości:

Ćwiczenie: plik .PGM

Utwórz *Nowy dokument tekstowy*, następnie wpisz do niego podaną lub własną strukturę obrazu, zapisz z rozszerzeniem .PGM.

Otwórz zapisany plik w programie graficznym. **Uwaga!** Wbudowane narzędzia systemowe nie potrafią otwierać plików .PGM, w tym celu należy zainstalować dodatkowy program, np. IrfanView, Gimp, ImageJ itp.

```
P2
8 8
1
1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1
```

```
P2
11 11
20
0 2 4 6 8 10 12 14 16 18 20
2 4 6 8 10 12 14 16 18 20 18
4 6 8 10 12 14 16 18 20 18 16
6 8 10 12 14 16 18 20 18 16 14
8 10 12 14 16 18 20 18 16 14 12
10 12 14 16 18 20 18 16 14 12 10
12 14 16 18 20 18 16 14 12 10 8
14 16 18 20 18 16 14 12 10 8 6
16 18 20 18 16 14 12 10 8 6 4
18 20 18 16 14 12 10 8 6 4 2
20 18 16 14 12 10 8 6 4 2 0
```

Rys. 1. Przykładowe pliki .PGM

Pliki .PGM może zawierać komentarze rozpoczynające się znakiem #. Poza pierwszą linią, mogą się one pojawić w każdym miejscu pliku. Obowiązują zawsze do końca linii, podobnie jak // znany z C/C++. Program wczytujący plik .PGM powinien być odporny na komentarze umieszczone w dowolnym miejscu w pliku. Przy odczytywaniu, można je po prostu ignorować.

Baza przykładowych plików .PGM znajduje się na http://www.student.mvlab.pl/pliki/?dir=c/obrazy .

```
P2
8 20
#komentarz
255
191 188 188 190 192 189 190 191 #komentarz
187 188 190 196 194 191 193 195
192 186 191 195 196 192 201 199
189 188 192 199 200 202 201 195
183 188 189 196 197 201 194 198
#też komentarz
184 185 184 192 198 193 198 196
165 185 186 187 190 193 197 199
...
```

Rys. 2. Przykład pliku .PGM z komentarzami

Struktury – wspólnie przechowywane dane różnych typów

Nie ma najmniejszego problemu ze zgrupowaniem danych jednego typu (np. **int**) w postaci tablicy. W języku C mamy również możliwość tworzenia zgrupowań danych różnych typów, służą do tego **struktury**.

Standardowy przykład: księgarnia. Problemem jest utworzenie bazy danych książek. Najbardziej charakterystyczną cechą książki jest tytuł, można by więc utworzyć tablicę tytułów (**char[]**). Jednak to za mało informacji, na przykład jeśli poszukiwane są książki danego autora (**char[]**) albo konkretne wydanie jakiejś książki (**int**), ponadto książka musi mieć przypisaną cenę (**float**). Każdą z cech książki można zapisać w osobnej tablicy, jednak lepszym rozwiązaniem byłoby skorzystanie z jednej tablicy, w której każdy element zawiera wszystkie te informacje. Można to zrobić tworząc strukturę książki.

Korzystanie ze struktur

Struktura jest zatem czymś w rodzaju „worka” przechowującego różne elementy – pola. W praktyce korzystanie ze struktur składa się z:

- **deklaracji struktury**, czyli szablonu, opisu jakie pola ma struktura zawierać;
- **deklaracji zmiennych strukturalnych** według tego planu (w przykładzie byłaby to konkretna książka);
- uzyskiwania dostępu do poszczególnych pól tych zmiennych;

Prosty program korzystający ze struktur przedstawiono poniżej:

```
#include <stdio.h>

struct ksiazka              //słowo "struct" + nazwa
{
	char tytul[100];    //pola
	char autor[100];
	int rokWydania;
```

```
	float cena;
};			//obowiązkowy średnik na końcu

int main(void)
{
	struct ksiazka hp1;		//deklaracja zmiennej strukturalnej o nazwie hp1

	hp1.rokWydania = 2000;				//wpisanie danych do pol
	hp1.cena = 19.99;				//zmiennej strukturalnej
	strcpy(hp1.tytul, "Harry Potter i kamien filozoficzny");
	strcpy(hp1.autor, "J.K. Rowling");

	printf("tytul: %s \n", hp1.tytul);		//wyswietlenie
	printf("autor: %s \n", hp1.autor);		//danych
	printf("rok wydania: %d \n", hp1.rokWydania);
	printf("cena: %0.2f \n", hp1.cena);
}
```

Listing 1. Przykład użycia struktury

Zatem **deklaracja struktury** zawiera słowo **`struct`** oraz nazwę w nagłówku, a wewnątrz typy i nazwy jej pól. Jak widać na przykładzie struktura może zawierać pojedyncze zmienne, tablice, ponadto wskaźniki, a także inne struktury itd. Od umieszczenia deklaracji w programie zależy możliwość dostępu do niej – rozmieszczenie globalne jak w przykładzie, pozwala korzystać ze struktury w całym pliku. Umieszczenie struktury w bloku, np. w **`main()`**, ograniczy ten obszar tylko do tego bloku.

Deklaracja zmiennej strukturalnej składa się znów ze słowa **`struct`**, z nazwy zadeklarowanego wcześniej szablonu oraz nazwy zmiennej. W tym wypadku **`struct`** ma inne znaczenie. Pokazana w przykładzie deklaracja powołuje zmienną do życia, jednak nie wypełnia jej pól. Dokonano w kilku kolejnych liniach za pomocą operatora przynależności „ . ” (kropka). Chcąc wpisać lub odczytać dane z pola zmiennej strukturalnej używa się jej nazwy, kropki oraz nazwy pola (w przykładzie użyto funkcji **`strcpy()`**, która służy do kopiowania łańcuchów znaków).

Innym sposobem na inicjalizację pól zmiennej strukturalnej jest podanie ich w momencie deklaracji zmiennej, w sposób podobny jak robi się to w tablicach:

```
struct ksiazka jezykC = { "Szkola programowania C", "S. Prata", 2006, 99.99 };
```

Należy przy tym pamiętać, żeby wpisać w klamry wartości pól dokładnie w takiej kolejności, jak w szablonie struktury.

Jeszcze innym sposobem deklaracji zmiennej strukturalnej jest umieszczenie jej przy deklaracji szablonu, zmienna staje się przy tym globalna:

```
struct ksiazka
{
	char tytul[100];
	char autor[100];
	int rokWydania;
	float cena;
} nowaKsiazka;			//o, tutaj

//lub:

struct ksiazka
{
	char tytul[100];
	char autor[100];
	int rokWydania;
```

```
        float cena;
} jezykC = { "Szkola programowania C", "S. Prata", 2006, 99.99 };   // <----

//lub:

struct ksiazka
{
        char tytul[100];
        char autor[100];
        int rokWydania;
        float cena;
} ksiazka1, *ksiazka2, kilkaKsiazek[5];                              // <----
```

Listing 2. Przykłady deklaracji zmiennej strukturalnej przy deklaracji struktury

Zmienne strukturalne stosuje się analogicznie do zmiennych wbudowanych typów. Nic nie stoi na przeszkodzie, by deklarować tablicę struktur (to pozwalałoby na zbudowanie bazy danych księgarni). Zmienne strukturalne można też przekazywać jako argument funkcji a także tworzyć do nich wskaźniki.

Wskaźniki na struktury

Wskaźników do struktur również używa się podobnie jak przy „zwykłych" zmiennych, wykorzystuje się operatory pobrania adresu „&" i wyłuskania wartości „*" :

```
struct ksiazka podrecznik = { "Elementarz", "J. Nowak", 2016, 29.99 };
struct ksiazka *wskPodrecznik;    //powołanie wskaznika na strukturę ksiazka
wskPodrecznik = &podrecznik;      //przypisanie wskaznikowi zmiennej strukturalnej

printf( "tytul: %s \n", (*wskPodrecznik).tytul );   //wyswietlenie za pomocą wskaznika
```

Listing 3. Przykład użycia wskaźnika na struktury

Powyżej zastosowano zapis `(*wskPodrecznik).tytul` by odnieść się do pola struktury, na którą wskazuje wskaźnik. Zamiast tego można skorzystać z uproszczonego zapisu, gdzie operator „." jest zastąpiony „->". Oba zapisy są równoznaczne.

```
(*wskPodrecznik).tytul == wskPodrecznik->tytul
```

Reasumując: odwołując się do pól, przy korzystaniu ze zmiennej strukturalnej używaj „.", zaś przy korzystaniu ze wskaźnika na zmienną używaj „->".

Aby przekazać zmienną strukturalną do funkcji gdzie miałaby być zmodyfikowana, należy to zrobić przez wskaźnik (przypomnij sobie zarządzanie pamięcią przy przekazywaniu zmiennych i ich adresów do funkcji z instrukcji do lab. 2). Przykładowo funkcja, która zmienia cenę książki, mogłaby wówczas wyglądać następująco:

```
void zmienCene( float nowaCena, struct ksiazka* Ksiazka )
{
        Ksiazka->cena = nowaCena;
}
```

Listing 4. Funkcja zmieniająca pole struktury

Możliwa jest również dynamiczna alokacja pamięci dla struktury czy tablicy struktur, co zapewnia ich nieulotność. Wtedy rozmiar (ilość bajtów) struktury można wyrazić: `sizeof( struct ksiazka )`.

Jednym z zadań projektowych jest utworzenie struktury na obraz .PGM. Z czego powinna składać się taka struktura?

*Zadanie domowe (dla ambitnych)

Utwórz program ze strukturą zawierającą jedno pole typu `char` i jedno pole typu `int`. Za pomocą instrukcji `sizeof()` wyświetl rozmiar struktury, a obok sumę rozmiarów `char` i `int`.

- Przeanalizuj wynik.
- Dopisz do struktury pole typu `double` na końcu lub początku i powtórz eksperyment;
- Przenieś pole double pomiędzy `char` i `int` i powtórz eksperyment;
- Z czego wynikają różnice?
- Zastosuj na początku programu dyrektywę `#pragma pack(1)` i powtórz eksperymenty;

Wskazówki

- Sprawdź numer standardu przed odczytem.
- Na początek przyjmij stały rozmiar obrazu, który możesz odczytać np. otwierając plik w notatniku. Dynamiczna alokacja pamięci dla obrazu będzie omówiona później.
- Do wychwytywania komentarzy możesz wykorzystać fakt, że funkcja `fscanf()` zwraca 0, jeśli nie odnajdzie żadnych pozycji z listy.

Zagadnienia na następne zajęcia

- Dynamiczna alokacja pamięci dla dwuwymiarowej tablicy danych.
- Zabezpieczenie przed wyciekiem pamięci.
- Zapis pliku .PGM pod nazwą podaną przez użytkownika.

Literatura

Prata S (2006) Język C. Szkoła Programowania. Gliwice. Wydawnictwo Helion.

https://pl.wikibooks.org/wiki/C - Wikibooks, Kurs programowania w języku C.

http://www.cplusplus.com/reference/ - Standard C++ Library reference.

http://netpbm.sourceforge.net/doc/pgm.html - Poskanzer J, PGM Format Specification.