1. Obraz w sepii

Zadanie polega na zastąpieniu kolorów w obrazie RGB przez odcienie sepii. Zamiana jest wykonywana dla pikseli, które spełniają nierówność:

$$dist \geq \sqrt{(R - R_{sel})^2 - (G - G_{sel})^2 - (B - B_{sel})^2}$$

gdzie *(R, G, B)* są składowymi czerwoną, niebieską i zieloną piksela obrazu, *(R_{sel}, G_{sel}, B_{sel})* są składowymi koloru odniesienia (jednego dla całego obrazu), natomiast *dist* jest progiem odległości od koloru odniesienia.

Składowe odcienia sepii wyznacza się z równań:

outputRed = (inputRed * .393) + (inputGreen *.769) + (inputBlue * .189)

outputGreen = (inputRed * .349) + (inputGreen *.686) + (inputBlue * .168)

outputBlue = (inputRed * .272) + (inputGreen *.534) + (inputBlue * .131)

Jeśli wartość składowej przekracza 255, należy ją zastąpić przez 255.

Wejście:

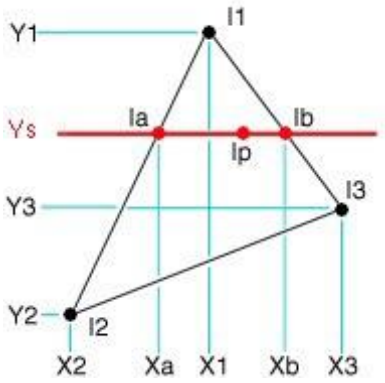
- Plik BMP zawierający obraz źródłowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 320x240 pikseli (istotne dla rozmiaru bufora),
  - Nazwa pliku: `source.bmp`
- *(R_{sel}, G_{sel}, B_{sel})* są składowymi koloru odniesienia (mogą być zmiennymi w programie, zakres wartości 0-255),
- dist – wartość progu odległości koloru (może być zmienną w programie, zakres wartości 0-442)

Wyjście:

- Plik BMP zawierający zmodyfikowany obraz:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar taki, jak pliku wejściowego (nie większy niż 320x240 pikseli),
  - Nazwa pliku: `result.bmp`

2. Cieniowanie

Zadanie polega na napisaniu programu gładko cieniującego trójkąt. Cieniowanie jest procesem wypełniania wielokąta, który uwzględnia kolory wierzchołków wielokąta. Do wyznaczenia składowych koloru pikseli na krawędziach i we wnętrzu jest używana interpolacja liniowa:



Ia = (Ys - Y2) / (Y1 - Y2) * I1 + (Y1 - Ys) / (Y1 - Y2) * I2

Ib = (Ys - Y3) / (Y1 - Y3) * I1 + (Y1 - Ys) / (Y1 - Y3) * I3

Ip = (Xb - Xp) / (Xb - Xa) * Ia + (Xp - Xa) / (Xb - Xa) * Ib

Wejście:

- (X1, Y1), (X2, Y2), (X3, Y3) – współrzędne wierzchołków trójkąta, mogą być zadane jako wartości zmiennych w programie,
- (R_1, B_1, G_1), (R_2, B_2, G_2), (R_3, B_3, G_3) – składowe koloru wierzchołków trójkąta; mogą być zadane jako wartości zmiennych w programie,

Wyjście:

- Plik BMP zawierający cieniowany trójkąt:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar nie większy niż 320x240 pikseli,
  - Nazwa pliku: `result.bmp`

3. Obraz czarno-biały

Zadanie polega na zamianie prostokątnego fragmentu obrazu RGB (24-bitowego) na obraz czarno-biały z wykorzystaniem progowania. Piksel podlegający konwersji jest biały kiedy spełniona jest nierówność:

$$thresh \geq 0.21R + 0.72G + 0.07B$$

gdzie (R, G, B) są składowymi koloru piksela, a *thresh* jest wartością progu (0-255).

Wejście:

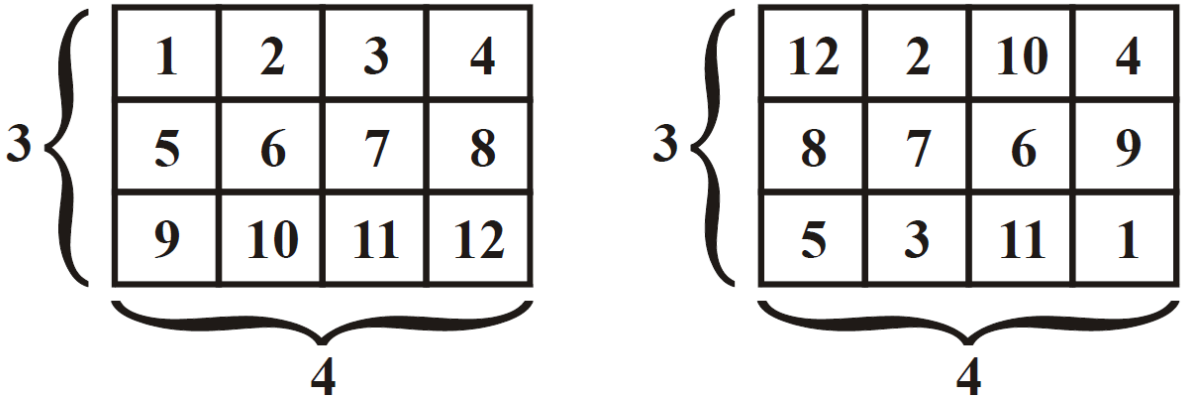
- Plik BMP zawierający obraz źródłowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 320x240 pikseli (istotne dla rozmiaru bufora),
  - Nazwa pliku: `source.bmp`
- *(x_1, y_1)* współrzędne lewego górnego rogu prostokątnego obszaru do modyfikacji (mogą to być wartości zmiennych w programie),
- *(x_2, y_2)* współrzędne prawego dolnego rogu prostokątnego obszaru do modyfikacji (mogą to być wartości zmiennych w programie),
- *thresh* – wartość progu jasności (może być zmienną w programie, zakres wartości 0-255)

Wyjście:

- Plik BMP zawierający zmodyfikowany obraz:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar taki, jak pliku wejściowego (nie większy niż 320x240 pikseli),
  - Nazwa pliku: `result.bmp`

4. Puzzle

Zadanie polega na podziale obrazu źródłowego na n x m prostokątnych fragmentów i ustawienie ich w losowym porządku. Na przykład:



Wejście:

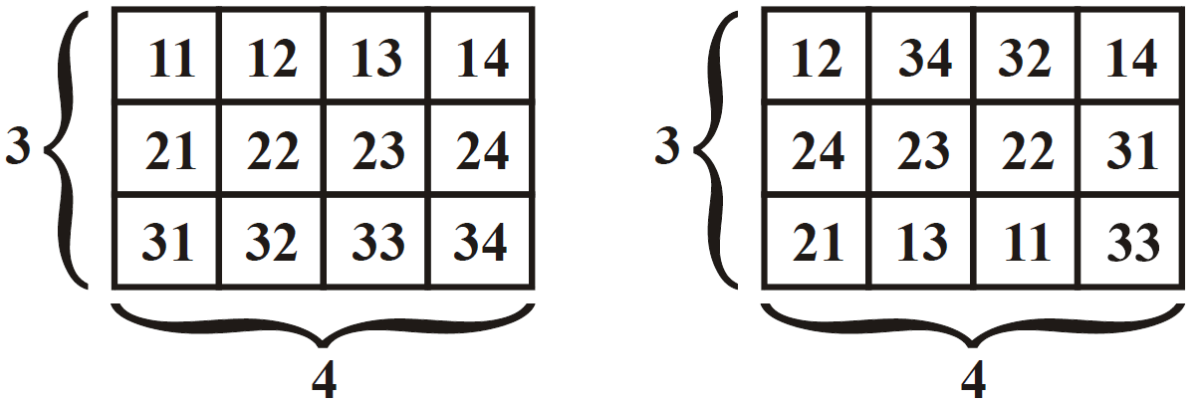
- Plik BMP zawierający obraz źródłowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 320x240 pikseli (istotne dla rozmiaru bufora),
  - Nazwa pliku: `source.bmp`
- *n* liczba wierszy w podzielonym obrazku (może to być wartość zmiennej w programie),
- *m* liczba kolumn w podzielonym obrazku (może to być wartość zmiennej w programie),

Wyjście:

- Plik BMP zawierający zmodyfikowany obraz:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar taki, jak pliku wejściowego (nie większy niż 320x240 pikseli),
  - Nazwa pliku: `result.bmp`

5. Puzzle 2

Zadanie polega na utworzeniu obrazu w formacie BMP, który zawiera prostokątne fragmenty obrazu źródłowego ułożone w zadany sposób. Obraz źródłowy jest podzielony na trzy wiersze i cztery kolumny (czyli mamy 12 prostokątnych fragmentów), które są identyfikowane dwoma cyframi: numerem wiersza (1-3) i numerem kolumny (1-4).



Widoczny po prawej stronie obraz wynikowy jest wynikiem wykonania zadania dla sekwencji:

`12,34,32,14,24,23,22,31,21,13,11,33`

Wejście:

- Plik BMP zawierający obraz źródłowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 320x240 pikseli (istotne dla rozmiaru bufora); proszę zwrócić uwagę, żeby szerokość obrazu źródłowego była podzielna przez 3, a wysokość podzielna przez 4),
  - Nazwa pliku: `source.bmp`
- *s* sekwencja dwucyfrowych identyfikatorów fragmentów oddzielonych spacjami lub przecinkami opisująca położenie fragmentów z obrazu źródłowego w porządku wierszowym.

Wyjście:

- Plik BMP zawierający zmodyfikowany obraz:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar taki, jak pliku wejściowego (nie większy niż 320x240 pikseli),
  - Nazwa pliku: `result.bmp`

6. Największa jasność

Zadanie polega na wykryciu w kolorowym obrazie źródłowym w formacie BMP prostokątnego obszaru o największej jasności. Do wyliczenia jasności piksela *I* proszę użyć wzoru:

$$I = 0.21R + 0.72G + 0.07B$$

gdzie (R, G, B) są składowymi koloru piksela.

Średnia jasność należy policzyć dla fragmentu obrazu m x n pikseli (m oznacza liczbę wierszy, a n liczbę kolumn prostokątnego fragmentu okna). Fragment o największej jasności trzeba zaznaczyć na obrazie wyjściowym czerwoną ramką. Jeśli jest więcej fragmentów o takiej samej, maksymalnej jasności należy zaznaczyć ramką tylko pierwszy fragment.

Wejście:

- Plik BMP zawierający obraz źródłowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 320x240 pikseli (istotne dla rozmiaru bufora),
  - Nazwa pliku: `source.bmp`
- *n* liczba wierszy fragmentu obrazu (może to być wartość zmiennej w programie),
- *m* liczba kolumn fragmentu obrazu (może to być wartość zmiennej w programie),

Wyjście:

- Plik BMP zawierający zmodyfikowany obraz:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar taki, jak pliku wejściowego (nie większy niż 320x240 pikseli),
  - Nazwa pliku: `result.bmp`

7. Dodawanie tekstu

Zadanie polega na dodaniu do kolorowego obrazu w formacie BMP jednej linii znaków; dopuszczalne znaki to cyfry i kropka. Znaki należy zdefiniować jako tablice 8x8 pikseli, np. cyfra 1 może wyglądać tak:

Wejście:

- Plik BMP zawierający obraz źródłowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 320x240 pikseli (istotne dla rozmiaru bufora),
  - Nazwa pliku: `source.bmp`
- *txt* sekwencja znaków zawierająca tylko cyfry i kropki (może to być zmienna w programie),
- *(x,y)* współrzędne górnego lewego rogu prostokątnego obszaru, w którym należy umieścić tekst (może to być zmienna w programie).

Wyjście:

- Plik BMP zawierający zmodyfikowany obraz:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar taki, jak pliku wejściowego (nie większy niż 320x240 pikseli),
  - Nazwa pliku: `result.bmp`

8. Odczyt kodu paskowego Code 128 (wersje A, B, C)

Zadanie polega na napisaniu programu, który odczytuje z kolorowego obrazu w formacie BMP informacje zakodowane przy użyciu kodu paskowego Code 128.

Code 128 umożliwia zapisanie 128 symboli z jednego z trzech zestawów znaków (Code Set A, Code Set B, Code Set C). O wyborze konkretnego zestawu decyduje znak startu. Każdy znak składa się z trzech pasków i trzech spacji (wyjątek stanowi znak Stop, który składa się z czterech pasków i trzech spacji). Szerokości pasków i spacji są całkowitymi (1/2/3/4) wielokrotnościami podstawowej szerokości paska (spacji).

Code 128 składa się z:

- strefy ciszy (minimum 10-krotność podstawowej szerokości paska),
- znaku startu
- zakodowanych danych
- znaku kontrolnego,
- znaku stop,
- strefy ciszy.

Więcej informacji o kodowaniu można znaleźć tutaj: https://pl.wikipedia.org/wiki/Kod_128

Wejście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 768x64 pikseli (istotne dla rozmiaru bufora),

Wyjście:

- Zdekodowane dane (lub informacja o błędzie) wyświetlona na konsoli

Wersje projektu:

- Dekodowanie Code Set A
- Dekodowanie Code Set B
- Dekodowanie Code Set C

Uwagi:

1. Wzorów pasków i odstępów nie należy przechowywać jako ciągów znaków.
2. Można analizować tylko jedną linię obrazu (w połowie wysokości obrazu).

9. Generowanie kodu paskowego Code 128 (wersje A, B, C)

Zadanie polega na napisaniu programu, który zapisuje w kolorowym obrazie w formacie BMP informacje zakodowane przy użyciu kodu paskowego Code 128.

Code 128 umożliwia zapisanie 128 symboli z jednego z trzech zestawów znaków (Code Set A, Code Set B, Code Set C). O wyborze konkretnego zestawu decyduje znak startu. Każdy znak składa się z trzech pasków i trzech spacji (wyjątek stanowi znak Stop, który składa się z czterech pasków i trzech spacji). Szerokości pasków i spacji są całkowitymi (1/2/3/4) wielokrotnościami podstawowej szerokości paska (spacji).

Code 128 składa się z:

- strefy ciszy (minimum 10-krotność podstawowej szerokości paska),
- znaku startu
- zakodowanych danych
- znaku kontrolnego,
- znaku stop,
- strefy ciszy.

Więcej informacji o kodowaniu można znaleźć tutaj: https://pl.wikipedia.org/wiki/Kod_128

Wejście:

- Podstawowa szerokość paska w pikselach (może to być zmienna w programie),
- Tekst do zakodowania (może to być zmienna w programie).

Wyjście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku 768x64 pikseli,
- Nazwa pliku: `code128.bmp`

Wersje projektu:

- Dekodowanie Code Set A
- Dekodowanie Code Set B
- Dekodowanie Code Set C

Uwagi:

1. Wzorów pasków i odstępów nie należy przechowywać jako ciągów znaków.
2. Proszę wykryć sytuację, w której przy zadanej szerokości paska i tekście do zakodowania szerokość pliku wyjściowego jest zbyt mała.

10. Odczyt kodu paskowego Code 39

Zadanie polega na napisaniu programu, który odczytuje z kolorowego obrazu w formacie BMP informacje zakodowane przy użyciu kodu paskowego Code 39.

Code 39 jest kodem o stałej szerokości znaku i umożliwia zapisanie 43 znaków alfanumerycznych (wielkie litery, cyfry i kilka znaków specjalnych). Jako znak startu i stopu jest używana '*', która w związku z tym nie może znajdować się w kodowanych danych. Każdy znak jest reprezentowany przez dziewięć pasków (pięć pasków czarnych i 4 cztery paski białe). Trzy z tych pasków mają większą szerokość, przy czym stosunek pasków szerokich do wąskich mieści się w przedziale od 2:1 do 3:1

Code 39 składa się z:

- znaku startu ('*')
- zakodowanych danych
- znaku kontrolnego,
- znaku stop ('*').

Więcej informacji o kodowaniu można znaleźć tutaj: https://pl.wikipedia.org/wiki/Code_39

Wejście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 768x64 pikseli (istotne dla rozmiaru bufora),

Wyjście:

- Zdekodowane dane (lub informacja o błędzie) wyświetlona na konsoli

Uwagi:

1. Wzorów pasków i odstępów nie należy przechowywać jako ciągów znaków.
2. Można analizować tylko jedną linię obrazu (w połowie wysokości obrazu).

11. Generowanie kodu paskowego Code 39

Zadanie polega na napisaniu programu, który zapisuje w kolorowym obrazie w formacie BMP informacje zakodowane przy użyciu kodu paskowego Code 39.

Code 39 jest kodem o stałej szerokości znaku i umożliwia zapisanie 43 znaków alfanumerycznych (wielkie litery, cyfry i kilka znaków specjalnych). Jako znak startu i stopu jest używana '*', która w związku z tym nie może znajdować się w kodowanych danych. Każdy znak jest reprezentowany przez dziewięć pasków (pięć pasków czarnych i 4 cztery paski białe). Trzy z tych pasków mają większą szerokość, przy czym stosunek pasków szerokich do wąskich mieści się w przedziale od 2:1 do 3:1

Code 39 składa się z:

- znaku startu ('*')
- zakodowanych danych
- znaku kontrolnego,
- znaku stop ('*').

Więcej informacji o kodowaniu można znaleźć tutaj: https://pl.wikipedia.org/wiki/Code_39

Wejście:

- Podstawowa szerokość paska w pikselach (może to być zmienna w programie),
- Tekst do zakodowania (może to być zmienna w programie).

Wyjście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku 768x64 pikseli,
- Nazwa pliku: `code39.bmp`

Uwagi:

1. Wzorów pasków i odstępów nie należy przechowywać jako ciągów znaków.
2. Proszę wykryć sytuację, w której przy zadanej szerokości paska i tekście do zakodowania szerokość pliku wyjściowego jest zbyt mała.

12. Odczyt kodu paskowego RM4SCC

Zadanie polega na napisaniu programu, który odczytuje z kolorowego obrazu w formacie BMP informacje zakodowane przy użyciu kodu paskowego RM4SCC.

W kodzie RM4SCC kodowane znaki składają się z czterech pasków (za wyjątkiem znaków Start i Stop), które mogą być rozszerzone w górę oraz w dół względem części środkowej (ta jest obecna zawsze).

Kod RM4SCC składa się z:

- znaku startu (pojedynczy pasek rozszerzony w górę)
- zakodowanych danych
- znaku kontrolnego,
- znaku stop (pojedynczy pasek rozszerzony w górę i w dół).

Więcej informacji o kodowaniu można znaleźć tutaj: https://en.wikipedia.org/wiki/RM4SCC

Wejście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku nie większy niż 768x64 pikseli (istotne dla rozmiaru bufora),

Wyjście:

- Zdekodowane dane (lub informacja o błędzie) wyświetlona na konsoli

Uwagi:

1. Wzorów pasków i odstępów nie należy przechowywać jako ciągów znaków.
2. Można analizować tylko trzy linie obrazu (w połowie wysokości obrazu, w ¼ wysokości – linia górna i ¾ wysokości – linia dolna).

13. Generowanie kodu paskowego RM4SCC

Zadanie polega na napisaniu programu, który zapisze w kolorowym obrazie w formacie BMP informacje zakodowane przy użyciu kodu paskowego RM4SCC.

W kodzie RM4SCC kodowane znaki składają się z czterech pasków (za wyjątkiem znaków Start i Stop), które mogą być rozszerzone w górę oraz w dół względem części środkowej (ta jest obecna zawsze).

Kod RM4SCC składa się z:

- znaku startu (pojedynczy pasek rozszerzony w górę)
- zakodowanych danych
- znaku kontrolnego,
- znaku stop (pojedynczy pasek rozszerzony w górę i w dół).

Więcej informacji o kodowaniu można znaleźć tutaj: https://en.wikipedia.org/wiki/RM4SCC

Wejście:

- Szerokość paska w pikselach (może to być zmienna w programie),
- Tekst do zakodowania (może to być zmienna w programie).

Wyjście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku 768x64 pikseli,
- Nazwa pliku: `coderm4scc.bmp`

Uwagi:

1. Wzorów pasków i odstępów nie należy przechowywać jako ciągów znaków.
2. Proszę wykryć sytuację, w której przy zadanej szerokości paska i tekście do zakodowania szerokość pliku wyjściowego jest zbyt mała.

14. Grafika żółwia – wersja 1

Grafika żółwia to grafika wektorowa wykorzystująca względny kursor („żółw”) na dyskretnej płaszczyźnie kartezjańskiej. Żółw ma trzy atrybuty: lokalizację, orientację (kierunek) i pióro. Pióro również ma atrybuty: kolor, stan włącz/wyłącz (opuszczone/podniesione).

Żółw porusza się za pomocą poleceń odnoszących się do jego własnej pozycji, takich jak „przesuń się o 10 kroków do przodu” i „skręć w lewo o 90 stopni”. Pióro trzymane przez żółwia można również kontrolować, włączając go lub ustawiając jego kolor.

Zadanie polega na napisaniu programu, który tłumaczy zakodowane binarnie polecenia żółwia na obraz rastrowy w pliku BMP.

Polecenia żółwia

Długość wszystkich poleceń żółwia wynosi 16 lub 32 bity. Dwa najmłodsze bity definiują jedno z czterech poleceń (ustaw pozycję, ustaw kierunek, ruch, ustaw stan). Niewykorzystane bity we wszystkich poleceniach są oznaczone znakiem –. Nie należy ich brać pod uwagę podczas dekodowania polecenia.

Polecenie ustawienia pozycji (*set position*)

Polecenie *set position* ustawia nowe współrzędne żółwia. Składa się z dwóch słów. Pierwsze słowo określa polecenie (bity 1-0) i współrzędne Y (bity y5-y0) nowej pozycji. Drugie słowo zawiera współrzędną X (bity x9-x0) nowej pozycji. Punkt (0,0) znajduje się w lewym dolnym rogu obrazu.

Tabela 1 Pierwsze słowo polecenia set position

bit no. 15	14	13	12	11	10	9	8
-	-	-	-	-	-	-	-
bit no. 7	6	5	4	3	2	1	0
y5	y4	y3	y2	y1	y0	1	1

Tabela 2 Drugie słowo polecenia set position

bit no. 15	14	13	12	11	10	9	8
-	-	-	-	-	-	x9	x8
bit no. 7	6	5	4	3	2	1	0
x7	x6	x5	x4	x3	x2	x1	x0

Polecenie ustaw kierunek (*set direction*)

Polecenie *set direction* określa kierunek, w którym żółw będzie się poruszał po wydaniu polecenia ruchu. Kierunek jest określony przez bity d1, d0.

Tabela 3 Polecenie set direction

bit no. 15	14	13	12	11	10	9	8
d1	d0	-	-	-	-	-	-
bit no. 7	6	5	4	3	2	1	0
-	-	-	-	-	-	1	0

Wartości bitów d1, d0 przekładają się na kierunek następująco:

00 Prawo 01 Góra 10 Lewo 11 Dół

Polecenie przesuń (*move*)

Polecenie *move* przesuwa żółwia w kierunku określonym przez bity d1-d0. Odległość ruchu jest określona przez bity m9-m0. Jeśli punkt docelowy znajduje się poza obszarem rysowania, żółw powinien zatrzymać się na krawędzi rysunku (czyli nie może opuścić obszaru rysowania). Żółw pozostawia widoczny ślad, gdy pióro jest opuszczone (patrz stan pióra niżej).

Tabela 4 Polecenie move

bit no. 15	14	13	12	11	10	9	8
m9	m8	m7	m6	m5	m4	m3	m2
bit no. 7	6	5	4	3	2	1	0
m1	m0	-	-	-	-	0	1

Polecenie ustaw stan pióra (*set pen state*)

Polecenie *set pen state* określa, czy pióro jest podniesione, czy opuszczone (bit ud) oraz kolor śladu. Bity r3-r0 są najbardziej znaczącymi bitami 8-bitowej czerwonej składowej koloru (pozostałe bity są ustawione na zero). Bity g3-g0 są najbardziej znaczącymi bitami 8-bitowej zielonej składowej koloru (pozostałe bity są ustawione na zero). Bity b3-b0 są najbardziej znaczącymi bitami 8-bitowej niebieskiej składowej koloru (pozostałe bity są ustawione na zero).

Tabela 5 Polecenie set pen state

bit no. 15	14	13	12	11	10	9	8
r3	r2	r1	r0	g3	g2	g1	g0
bit no. 7	6	5	4	3	2	1	0
b3	b2	b1	b0	ud	-	0	0

Interpretacja bitu ud jest następująca: 0 – pióro podniesione, 1 – pióro opuszczone.

Wejście:

- Plik binarny zawierający polecenia żółwia,
- Nazwa pliku: `turtle1.bin` (może to być zmienna w programie).

Wyjście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku 768x64 pikseli,
- Nazwa pliku: `turtle1.bmp`

15. Grafika żółwia – wersja 2

Grafika żółwia to grafika wektorowa wykorzystująca względny kursor („żółw”) na dyskretnej płaszczyźnie kartezjańskiej. Żółw ma trzy atrybuty: lokalizację, orientację (kierunek) i pióro. Pióro również ma atrybuty: kolor, stan włącz/wyłącz (opuszczone/podniesione).

Żółw porusza się za pomocą poleceń odnoszących się do jego własnej pozycji, takich jak „przesuń się o 10 kroków do przodu” i „skręć w lewo o 90 stopni”. Pióro trzymane przez żółwia można również kontrolować, włączając go lub ustawiając jego kolor.

Zadanie polega na napisaniu programu, który tłumaczy zakodowane binarnie polecenia żółwia na obraz rastrowy w pliku BMP.

Polecenia żółwia

Długość wszystkich poleceń żółwia wynosi 16 lub 32 bity. Dwa najstarsze bity definiują jedno z czterech poleceń (ustaw pozycję, ustaw kierunek, ruch, ustaw stan). Niewykorzystane bity we wszystkich poleceniach są oznaczone znakiem –. Nie należy ich brać pod uwagę podczas dekodowania polecenia.

Polecenie ustawienia pozycji (*set position*)

Polecenie *set position* ustawia nowe współrzędne żółwia. Składa się z dwóch słów. Pierwsze słowo określa polecenie (bity 1-0) i współrzędne Y (bity y5-y0) nowej pozycji. Drugie słowo zawiera współrzędną X (bity x9-x0) nowej pozycji. Punkt (0,0) znajduje się w lewym dolnym rogu obrazu.

Tabela 6 Pierwsze słowo polecenia set position

bit no. 15	14	13	12	11	10	9	8
0	0	-	-	-	-	-	-
bit no. 7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-

Tabela 7 Drugie słowo polecenia set position

bit no. 15	14	13	12	11	10	9	8
y5	y4	y3	y2	y1	y0	x9	x8
bit no. 7	6	5	4	3	2	1	0
x7	x6	x5	x4	x3	x2	x1	x0

Polecenie ustaw kierunek (*set direction*)

Polecenie *set direction* określa kierunek, w którym żółw będzie się poruszał po wydaniu polecenia ruchu. Kierunek jest określony przez bity d1, d0.

Tabela 8 Polecenie set direction

bit no. 15	14	13	12	11	10	9	8
0	1	-	-	-	-	-	-
bit no. 7	6	5	4	3	2	1	0
-	-	-	-	-	-	d1	d0

Wartości bitów d1, d0 przekładają się na kierunek następująco:

00 Prawo 01 Góra 10 Lewo 11 Dół

Polecenie przesuń (*move*)

Polecenie *move* przesuwa żółwia w kierunku określonym przez bity d1-d0. Odległość ruchu jest określona przez bity m9-m0. Jeśli punkt docelowy znajduje się poza obszarem rysowania, żółw powinien zatrzymać się na krawędzi rysunku (czyli nie może opuścić obszaru rysowania). Żółw pozostawia widoczny ślad, gdy pióro jest opuszczone (patrz stan pióra niżej).

Tabela 9 Polecenie move

bit no. 15	14	13	12	11	10	9	8
1	0	-	-	-	-	m9	m8
bit no. 7	6	5	4	3	2	1	0
m7	m6	m5	m4	m3	m2	m1	m0

Polecenie ustaw stan pióra (*set pen state*)

Polecenie *set pen state* określa, czy pióro jest podniesione, czy opuszczone (bit ud) oraz kolor śladu. Bity r3-r0 są najbardziej znaczącymi bitami 8-bitowej czerwonej składowej koloru (pozostałe bity są ustawione na zero). Bity g3-g0 są najbardziej znaczącymi bitami 8-bitowej zielonej składowej koloru (pozostałe bity są ustawione na zero). Bity b3-b0 są najbardziej znaczącymi bitami 8-bitowej niebieskiej składowej koloru (pozostałe bity są ustawione na zero).

Tabela 10 Polecenie set pen state

bit no. 15	14	13	12	11	10	9	8
1	1	ud	-	b3	b2	b1	b0
bit no. 7	6	5	4	3	2	1	0
g3	g2	g1	g0	r3	r2	r1	r0

Interpretacja bitu ud jest następująca: 0 – pióro podniesione, 1 – pióro opuszczone.

Wejście:

- Plik binarny zawierający polecenia żółwia,
- Nazwa pliku: `turtle2.bin` (może to być zmienna w programie).

Wyjście:

- Plik BMP zawierający kod paskowy:
  - Format RGB (24 bity) bez kompresji,
  - Rozmiar pliku 768x64 pikseli,
- Nazwa pliku: `turtle2.bmp`