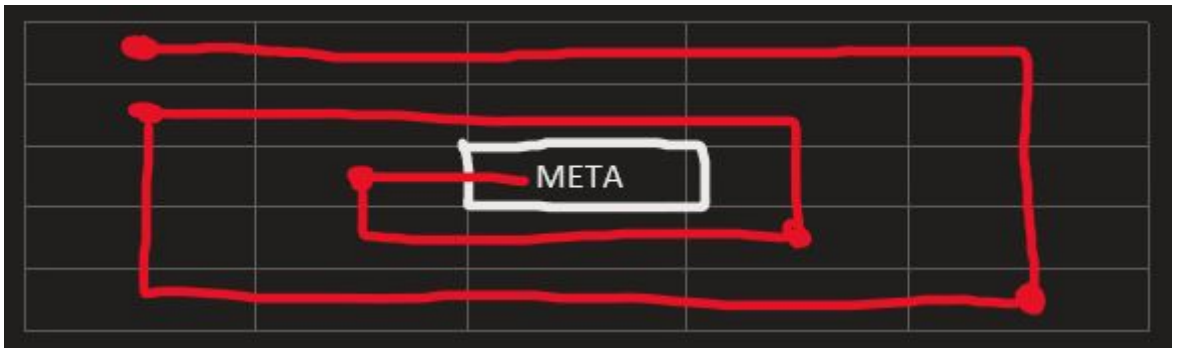
Sprawozdanie
Projekt 1

Albert Gawin

Wstęp

Algorytm polega na wypisaniu macierzy po spirali. Macierz przyjmuje wymiary MxN podane przez użytkownika, a wypisanie jej zawartości zaczyna się od lewego górnego idąc po zewnętrznych stronach tablicy zmierzając do jej wnętrza.

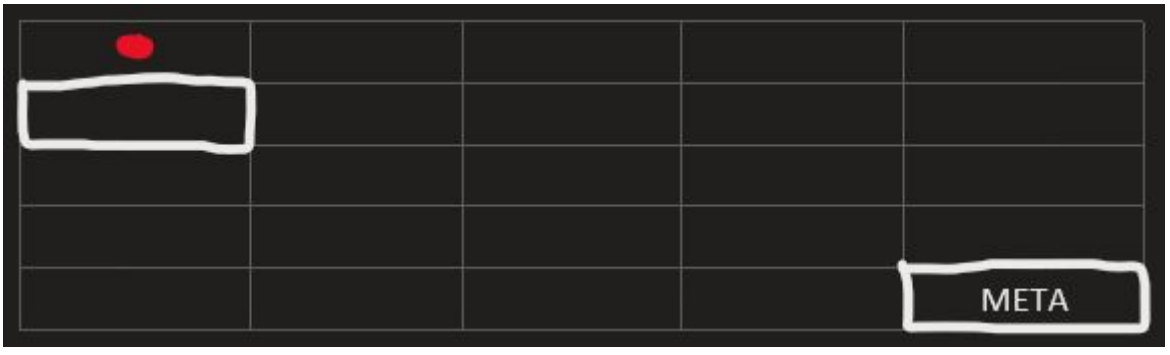


Opis algorytmu

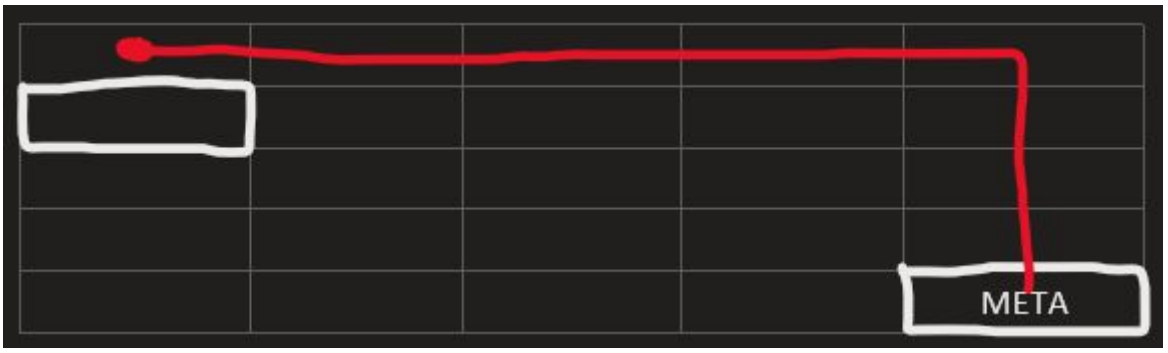
Algorytm polega na wyświetleniu wszystkich elementów tablicy dwuwymiarowej po spirali. Aby ułatwić wyjaśnienie działania algorytmu nadajemy nazwę naszemu znacznikowi "**biegacz**" (na rysunkach kolor czerwony). Natomiast dwa białe bloki służą nam jako narożniki tablicy. Pomogą nam one w zmianie położenia "**mety**" do której dobiec ma biegacz.

Ustawiamy pozycje wszystkich zmiennych na:

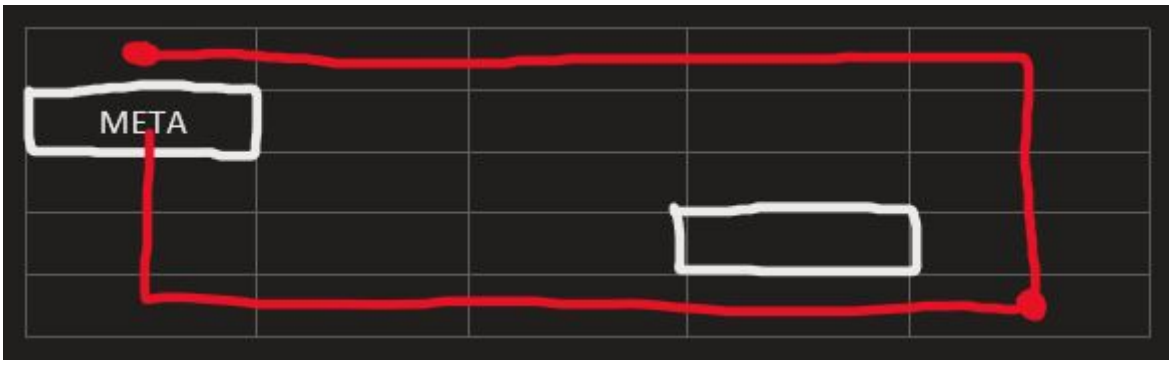
- biegacz **[0][0]**
- górny róg **[1][0]**
- dolny róg **[M][N]**



Jako biegacz dążymy do mety **[M][N].** Biegacz najpierw biegnie do kolumny **[N]**, następnie do wiersza **[M]**.

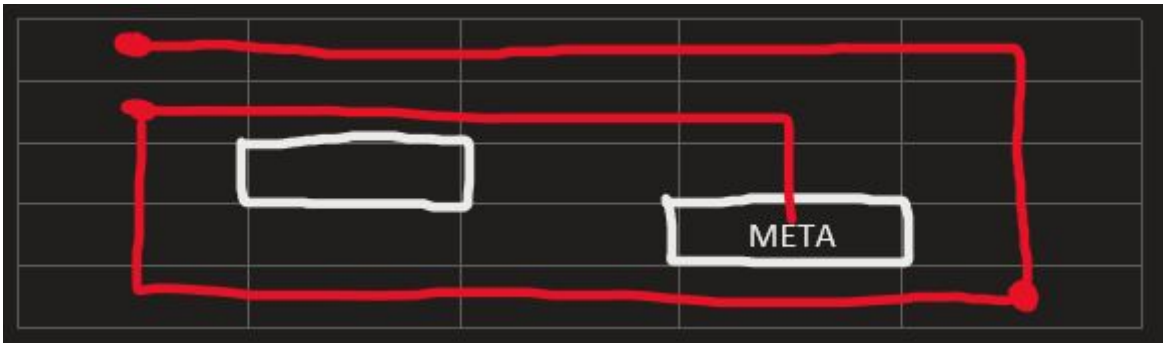


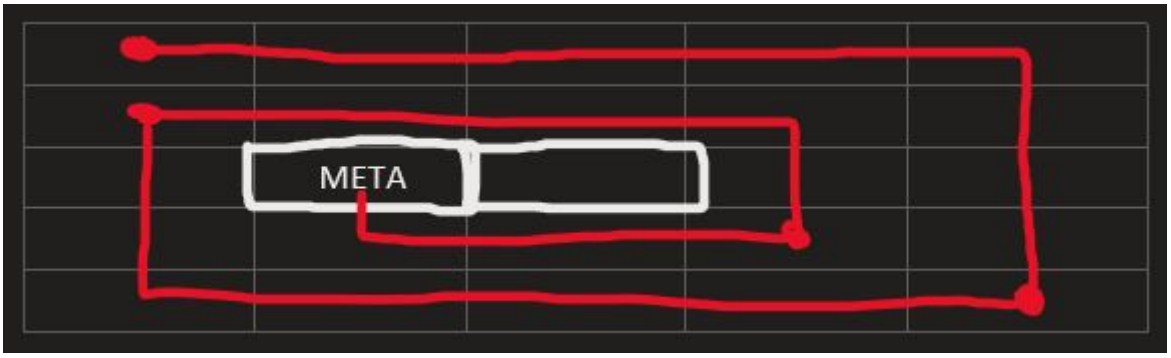
Kiedy biegacz dotrze do **mety** zmieniamy jej miejsce na drugi róg. Następnie zmieniamy położenie rogu dolnego na [M-1][N-1]. Biegacz będzie tak samo biegł do mety.

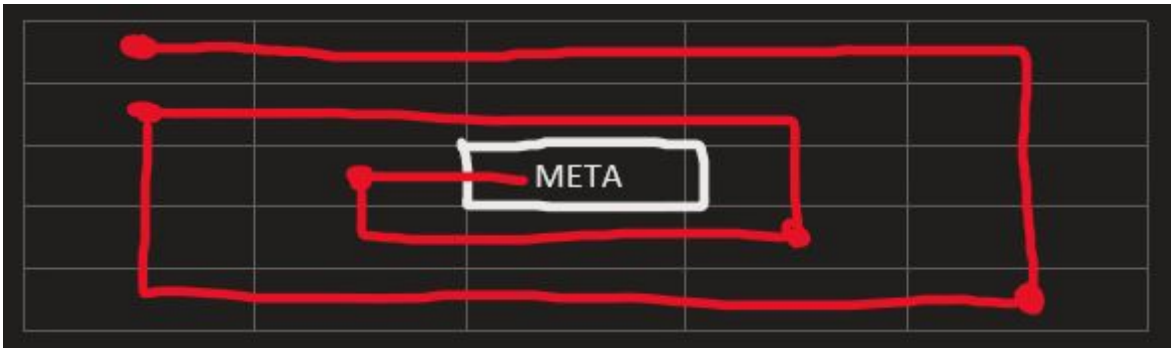


Cały czas powtarzamy te same kroki aż do wypisania tylu wyników ile miejsc w tablicy (M*N). Tym razem jako biegacz dążymy do mety **[M-1][N-1].** Jednocześnie swoje miejsce zmienia górny róg na wcześniejsze położenie + 1.

Następnych kroków nie będę opisywał, ponieważ działają na tej samej zasadzie co poprzednie.







Przeprowadzane testy

Program pilnuje czy plik nie jest pusty lub nie istnieje. Jeżeli jeden z tych przypadków wystąpi program pominie funkcję odpowiedzialną za spiralę oraz zapisanie macierzy do pliku. Dzięki takiemu zabiegowi program nie zwróci błędu w razie braku pliku lub braku danych w pliku.

Prosty warunek sprawdzający czy plik nie zawiera błędów oraz czy nie jest pusty.

```
bool isEmpty = file.peek() == std::ifstream::traits_type::eof();
if(file.good() && !isEmpty)
```

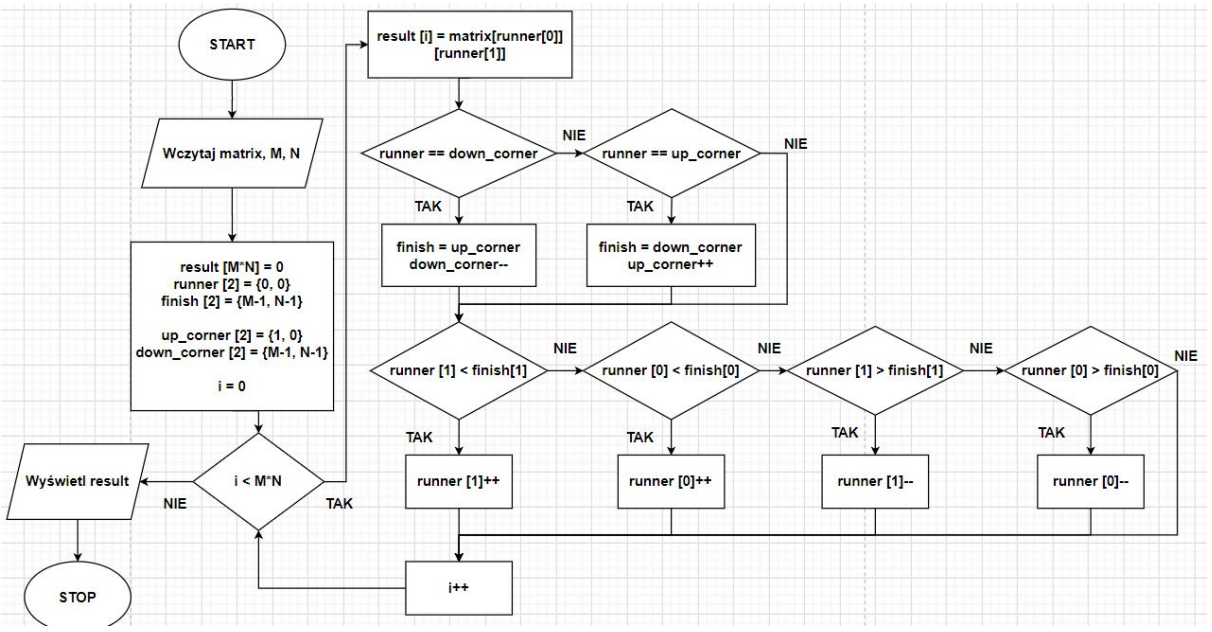
Jeżeli warunek sprawdzający nie spełni się to funkcja zwraca 0. **Continue** pomija dalsze wywołania funkcji i przechodzi do kolejnego pliku.

```
if (matrix == 0) continue;
```

Oto wynik, kiedy plik 8.txt nie istnieje a plik3.txt jest pusty.

```
WYNIK [1.txt]: 1

WYNIK [2.txt]: 1 2 3 4

WYNIK [4.txt]: 1 2 3 4 5 6 7 8 9 10 11

WYNIK [5.txt]: 1 2 3 4 5 6 7 8 9 10 11

WYNIK [6.txt]: 9 9 9 9 3 2 2 2 2 3 3 3

WYNIK [7.txt]: 1 2 3 4 5 6 7 8 9

WYNIK [9.txt]: 1 2 3 4

WYNIK [10.txt]: 1 2 3 4
```

Schemat blokowy



Pseudokod

1. **Wczytaj** macierz, M, N

2. wynik [M*N] **<-** 0
   biegacz [2] **<-** {0, 0}
   meta [2] **<-** {M-1, N-1}
   gorny_rog [2] **<-** {1,0}
   dolny_rog [2] **<-** {M-1, N-1}
   i **<-** 0

3. **Dopóki** i **mniejsze od** M*N
4. wynik [i] **<-** macierz[biegacz[0]] [biegacz[1]]
5. **Jeżeli** biegacz **równa się** dolny_rog **to**
6. meta <- gorny_rog
7. dolny_rog **zmniejsz o** 1
8. **albo jeżeli** biegacz **równa się** gorny_rog **to**
9. meta <- dolny_rog
10. gorny_rog **zwiększ o** 1

11. **Jeżeli** biegacz [1] **mniejsze od** meta [1] **to**
12. biegacz [1] **zwiększ o** 1
13. **albo jeżeli** biegacz [0] **mniejsze od** meta [0] **to**
14. biegacz [0] **zwiększ o** 1
15. **albo jeżeli** biegacz [1] **większe od** meta [1] **to**
16. biegacz [1] **zmniejsz o** 1
17. **albo jeżeli** biegacz [0] **większe od** meta [0] **to**
18. biegacz [0] **zmniejsz o** 1

19. i **zwiększ o** 1

20. **Wyświetl** wynik

Doświadczenia

Większość czasu spędziłem na wymyśleniu sposobu implementacji algorytmu. Największy kłopot sprawiło mi poprawne działanie “**biegacza**”. Natomiast projekt ten głównie nauczył mnie poprawnej pracy na tablicach jedno i dwuwymiarowych. Jak się okazało wysyłanie tablic do funkcji nie musi być tak ciężkie.

Wnioski

1. Najlepszym sposobem pracy na tablicach jest używanie wskaźników. Poniżej prosty przykład inicjalizacji tablicy dwuwymiarowej. Ułatwia to przesyłanie/zwracanie tablic wielowymiarowych do/z funkcji.

```
int** matrix;
matrix = new int*[M];

for(int i=0; i<M; i++) {
    matrix[i] = new int[N];
    for(int j=0; j<N; j++) {
        file >> matrix[i][j];
    }
}
```

2. Nazewnictwo funkcji jest bardzo ważne w programowaniu. Musimy pilnować nazw zmiennych oraz funkcji, ponieważ może to doprowadzić do błędnego zrozumienia naszego kodu przez innych programistów. Dla przykładu, jeśli istnieje funkcja o nazwie “saveMatrixToFIle” możemy domyśleć się, że zapisuje ona macierz do pliku.

Niestety jeżeli okaże się, że funkcja ta nie tylko zapisuje matrycę do pliku ale również wywołuje inne funkcje jak “getMatrixFromFile” oraz “getResultOfSpiral” to może to wywołać dalsze błędy w kodzie.