POLITECHNIKA
RZESZOWSKA
im. IGNACEGO ŁUKASIEWICZA

Karolina Magdoń

Inżynieria i analiza danych
I rok

Sprawozdanie z projektu przesuwania elementów tablicy (C++)

Praca przygotowana na zajęcia
„Algorytmy i struktury danych”
w roku akademickim 2022/2023

Rzeszów, listopad 2022

Spis treści

1. Wstęp i opis zagadnień projektu 3
2. Podstawy teoretyczne zagadnienia 3
3. Cechy programu 4
4.1 Złożoność obliczeniowa i czasowa 5
4.2 Schemat blokowy 6
4.3 Dane we/wy i zmienne pomocnicze 7
4.4 Rezultaty testów 8
5. Podsumowanie i wnioski 10
6. Appendix: kod programu 11

Spis ilustracji

Rys. 1 Schemat blokowy algorytmu 6
Rys. 2 Test 1. (wcześniejsza wersja programu) 8
Rys. 3 Test dla tablicy pobieranej z pliku tekstowego (dane z przykładu dołączonego do zadania) 8
Rys. 4 Test 2. dla tablicy długości 10 liczb z przedziału [-100,100] 8
Rys. 5 Test 3. dla tablicy długości 100 liczb z przedziału [-200,200] 8
Rys. 6 Test 4. dla tablicy długości 1000 liczb z przedziału [-5000,5000] 9
Rys. 7 Wykres długości czasu działania przy zwiększającej się długości tablicy 9

1. Wstęp i opis zagadnień projektu

Moim zadaniem projektowym było utworzenie programu, który dla zadanej tablicy liczb całkowitych przesunie wszystkie elementy mniejsze od 0 na jej koniec. Dodatkowym warunkiem było zachowanie kolejności występowania elementów ujemnych. Do kolejnych zaleceń należało: przedstawienie podstaw teoretycznych zagadnienia oraz schemat blokowy algorytmu (również pseudokod). Zobrazować rezultaty działania programu. Ukazać złożoność czasową, obliczeniową.

2. Podstawy teoretyczne zagadnienia

W algorytmie będę używać Tablic i wskaźników, ważne jest, by rozumieć, co znaczą te pojęcia.

Tablica - Tablica to zbiór danych określonego typu. Dostęp do danych możliwy jest za pomocą kluczy, będących ich indeksami. Wykorzystanie tablic umożliwia operowanie na dużych ilościach danych tego samego typu. Deklaracji dokonujemy w sposób następujący:

typ_danych nazwa_tablicy[ilość_elementów];

Wskaźnik - Wskaźnik jest zmienną przechowującą adres innej zmiennej. Wskaźnik przechowuje adres zmiennej znajdującej się w pamięci komputera. Nie ma konieczności odwoływania się do konkretnej zmiennej. Deklaracja wskaźnika wygląda prawie identycznie jak deklaracja typowej zmiennej:

int * wskaznik;

Dla nas wskaźniki będą o tyle przydatne, że w przeciwieństwie do Tabeli, gdzie zawsze trzeba podać indeks, do którego się odwołujemy, wskaźnik będzie można w całości przekazać jako to co zwraca funkcja. Bez tego nie mogłabym działać większą ilością funkcji na jednej tabeli.

3. Cechy programu

1. Możliwość odczytywania danych wejściowych z pliku tekstowego i zapisu posortowanego ciągu do pliku tekstowego z danymi wyjściowymi.
2. Na potrzeby testów zaimplementowanie funkcji generującej „losowe” ciągi elementów (o zadanej długości)
3. Główna funkcja przesuwająca elementy w tablicy tak, by spełniały wymaganie zadania.
4. Mierzenie czasu działania głównego algorytmu i zapis do pliku czasu działania dla serii zmierzonych czasów z zwiększającą się ilością elementów tablic.

Do wykonania tego projektu stworzyłam kod w języku C++ zawierający:

- funkcję wczytującą z pliku serię liczb,
- funkcję generującą losowy zbiór liczb (różną ich ilość oraz liczby z różnych przedziałów) służącą do testowania algorytmu przesuwającego,
- właściwy algorytm przesuwający,
- zaimplementowany do algorytmu przesuwającego miernik czasu działania,
- segmenty zapisujące do plików otrzymane dane.

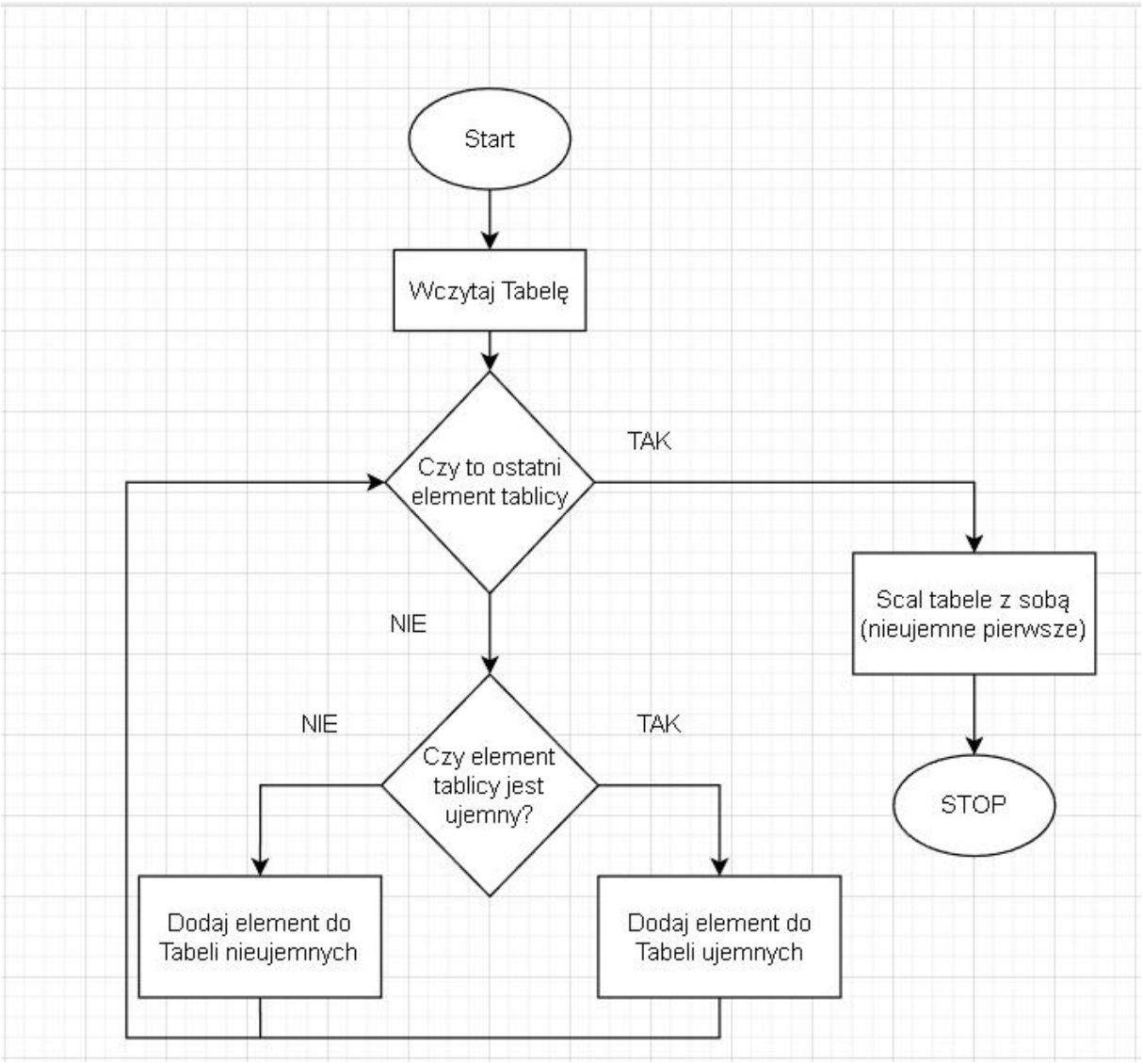
4. Opis algorytmu przesuwającego

Algorytm bierze Tablicę liczb naturalnych i dla każdego elementu sprawdza, czy jest ujemna czy nieujemna. Wstawia go do odpowiedniej tablicy tymczasowej, po czym scala te tablice, zaczynając od tablicy elementów nieujemnych.

4.1 Złożoność obliczeniowa i czasowa

Złożoność obliczeniowa algorytmu jest bardzo wydajna, w każdym wypadku wynosi O(n), gdyż wystarczy, że tylko 2 razy przeanalizujemy wszystkie liczby z tabeli, aby wyznaczyć wynik. Złożoność czasowa jest znikoma, miałam problemy ze sprawieniem, by miernik czasu nawet zmierzył czas wykonywania się algorytmu. W najgorszym wypadku będzie to kilka tysięcznych sekundy, przy naszej skali działania jest to prawie nic.

4.2 Schemat blokowy



Rys. 1 Schemat blokowy algorytmu

4.3 Pseudokod

K01: Wczytaj Tabelę

K02: Stwórz 2 Tabele Pomocnicze:

1. Tabela która będzie przetrzymywać liczby nieujemne
2. Tabela która będzie przetrzymywać liczby ujemne

K03: Stwórz 2 Zmienne Pomocnicze:

1. Zmienna która będzie wskaźnikiem tabeli z liczbami nieujemnymi
2. Zmienna która będzie wskaźnikiem tabeli z liczbami ujemnymi

K04: Patrzymy na każdy element po kolei i sprawdzamy dla niego, czy jest mniejszy od 0, jeśli nie, przechodzimy do K04.1, jeśli tak, do K04.2

K04.1 Dopisz element do Tabeli dla elementów nieujemnych na miejsce, które wskazuje wskaźnik i zwiększ wartość wskaźnika tabeli z liczbami nieujemnymi

K04.2 Dopisz element do Tabeli dla elementów ujemnych na miejsce, które wskazuje wskaźnik i zwiększ wartość wskaźnika tabeli z liczbami ujemnymi

K05: Scalamy Tabele ze sobą (nieujemne jako pierwsza)

K06: Wypisz Tabelę.

4.3 Dane we/wy i zmienne pomocnicze

- Dane wejściowe: Tablica nieposortowana;
- Dane wyjściowe: Tablica posortowana;
- Zmienne pomocnicze:
    - Tablice Ujemnych/Nieujemnych – służą do przechowywania rozdzielonych liczb;
    - Iteratory Ujemnych/Nieujemnych - do zapisywania do tablic i do liczenia ich długości;
    - Przesunięta tablica – tablica, do której scalamy tablice nieujemnych i ujemnych;
    - Iterator przesuniętej tablicy - do zapisywania do tablicy i do liczenia jej długości.

4.4 Rezultaty testów

Algorytm w każdym przypadku poprawnie wykonał przesuwanie na Tabeli. Oto parę przykładów:

```
Wygenerowana Tabica = [ 6 -7 8 4 -1 -7 9 8 -3 9 ]
Tabica po przesunieciu = [ 6 8 4 9 8 9 -7 -1 -7 -3 ]
```

Rys. 2 Test 1. (wcześniejsza wersja programu)

```
Wczytana Tablica = [ -10 5 8 -4 1 3 0 -7 ]
Tablica po przesunieciu = [ 5 8 1 3 0 -10 -4 -7 ]
```

Rys. 3 Test dla tablicy pobieranej z pliku tekstowego (dane z przykładu dołączonego do zadania)

```
Dlugosc: 10 Granica: 100
Wygenerowana Tablica = [ -48 14 -40 62 40 98 88 -86 -52 -82 ]
Posortowana Tablica = [ 14 62 40 98 88 -48 -40 -86 -52 -82 ]
```

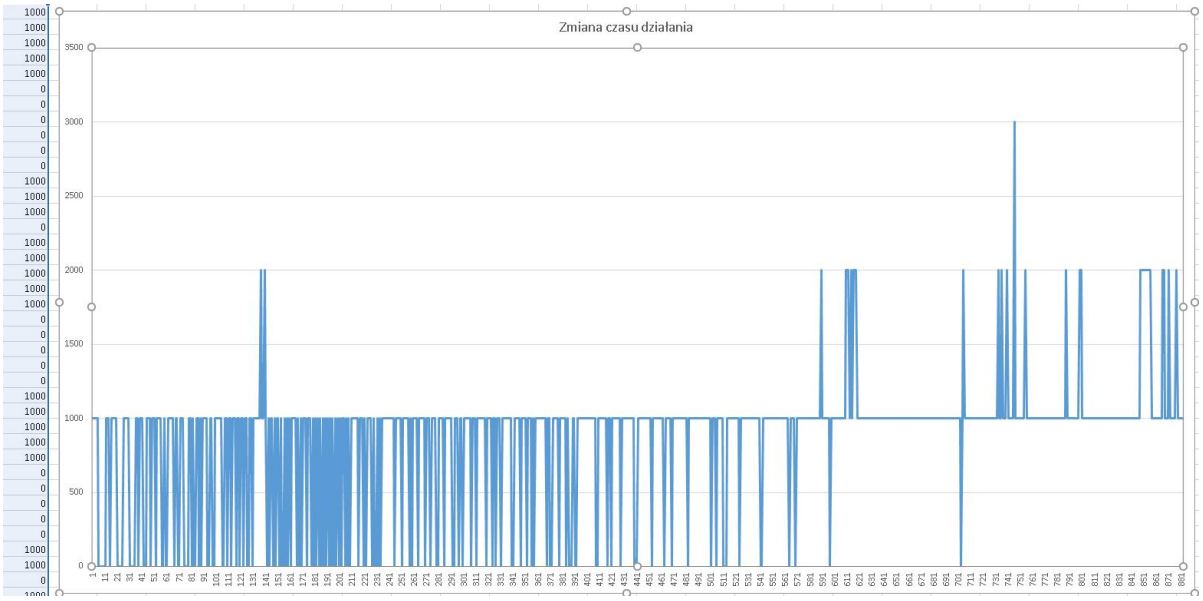
Rys. 4 Test 2. dla tablicy długości 10 liczb z przedziału [-100,100]

```
Dlugosc: 100 Granica: 200
Wygenerowana Tablica = [ 52 114 -140 162 140 -2 -12 -186 48 -182 -164 104 24 120 116 112 -194 -176 178 -74 2 -80 -54 120 -174
 -44 -4 102 -2 -168 -86 -24 16 10 -142 24 154 122 -60 -10 92 100 -72 182 -150 -40 -22 128 154 -168 68 -142 26 194 -130 108 -1
94 20 110 68 62 54 22 70 64 68 186 -80 0 2 160 -124 -98 50 -64 -132 -56 72 -16 -62 -110 -56 8 -70 -64 -154 198 -152 -4 -28 -9
2 180 -194 28 -70 -82 196 14 -84 -118 ]
Posortowana Tablica = [ 52 114 162 140 48 104 24 120 116 112 178 2 120 102 16 10 24 154 122 92 100 182 128 154 68 26 194 108
20 110 68 62 54 22 70 64 68 186 0 2 160 50 72 8 198 180 28 196 14 -140 -2 -12 -186 -182 -164 -194 -176 -74 -80 -54 -174 -44 -
4 -2 -168 -86 -24 -142 -60 -10 -72 -150 -40 -22 -168 -142 -130 -194 -80 -124 -98 -64 -132 -56 -16 -62 -110 -56 -70 -64 -154 -
152 -4 -28 -92 -194 -70 -82 -84 -118 ]
```

Rys. 5 Test 3. dla tablicy długości 100 liczb z przedziału [-200,200]

Rys. 6 Test 4. dla tablicy długości 1000 liczb z przedziału [-5000,5000]

Dalej wykonałam testy sprawności zależne od ilości elementów. Ich liczba zwiększała się od 100000 do 250000, krok wynosił 100. Co dopiero, takie liczby sprawiły, że biblioteki do mierzenia czasu wykryły jakąkolwiek zmianę czasu. Wyniki wypisane zostały w milisekundach.



Rys. 7 Wykres długości czasu działania przy zwiększającej się długości tablicy

Jak widać wykres nie jest liniowy, ponieważ zależnie od losowo wygenerowanych liczb, czasami czas był mierzony, jako tysięczne sekundy, a czasami po prostu 0. Widać jednak lekką tendencję wzrostu czasu działania funkcji, być może przy ogromnych ilościach elementów czas działania byłby znaczny. (Mi niestety przy większych liczbach wywala CodeBlocks.)

5. Podsumowanie i wnioski

Założenia projektu udało się zrealizować relatywnie nieskomplikowanym obliczeniowo algorytmem, ponadto poradził on sobie bardzo dobrze, wykonanie algorytmu wręcz nie zajmuje żadnego czasu i co dopiero przy ogromnych długościach tablic mogliśmy wykryć czas działania, który i tak był liczony w tysięcznych sekundach. Przez specyfikę zagadnienia użycie algorytmu sortującego nie dawało nam pożądanego wyniku, więc użyłam w moim algorytmie bardziej codziennego znaczenia słowa sortowanie i podzieliłam liczby na ujemne i nieujemne. Ponieważ przechodziliśmy po tabelach w kolejności od lewej do prawej, tabele po „sortowaniu” mogą zostać ze sobą złożone bez dodatkowych operacji.

Podsumowując:

1. Program odczytuje dane liczbowe z pliku i zapisuje do tablicy.
2. Program zapisuje wynik do pliku.
3. Wykonano testy sprawdzające działanie algorytmu.
4. Program mierzy czas wykonania się algorytmu.
5. Kod opatrzono komentarzami.
6. Sporządzono schemat blokowy oraz pseudokod.

6. Appendix: kod programu

```
#include <iostream>
#include <ctime> //biblioteka potrzebna to losowania
#include <fstream> //biblioteka do wczytywania i zapisywania do txt
#include <chrono> //biblioteka do mierzenia czasu
using namespace std::chrono; //potrzebne do mierzenia czasu
using namespace std;

//Pierwsza funkcja służy do wczytania tabeli z pliku txt
int * WczytanieTablicy (){ //Funkcja zdefiniowana tak żeby nie przyjmowała żadnych argumentów i tak żeby zwracała wczytaną tablicy

   fstream wczytywanie; //Tworzenie zmiennej, do której zapisany będzie plik, typu do tego przeznaczonego
   wczytywanie.open("Liczby.txt", ios::in); //Otwieranie wcześniej przygotowanego pliku tekstowego

   int i=0; //Deklaracja iteratora który będzie służył do zapisywania danych i równocześnie do zmierzenia długości tablicy
   int * WczytanaTabela = new int[i]; //Tworzenie tablicy dynamicznej do której zapisane będą liczby
   while(!wczytywanie.eof()){ //While wykonuje się aż skończy się plik
      wczytywanie >> WczytanaTabela[i]; //Po kolei wczytywane są liczby na kolejne miejsca w tablicy
      i++; //Iterator jest zwiększany przy każdym wczytaniu liczby, co na raz liczy ile liczb jest wczytane i służy do wskazania kolejnego miejsca na które wpisywana jest liczba
   }
   wczytywanie.close(); //Zamykanie pliku

   //Wypisywanie do konsoli wczytanej tablicy
   cout<<"Wczytana Tablica = [ ";
   for (int j=0; j<i; j++){ //For idący od 0 do i (które jest jest ilością liczb)
```

```
        cout <<WczytanaTabela[j]<<" ";
    }
    cout<<"]"<<endl;

return WczytanaTabela; //Funkcja zwraca wczytaną tablicy
}

//Druga funkcja generuje tablicę liczb naturalnych od -100 do 100
int * GenerowanieTablic (int N) { //Przyjmuje argument długości tabicy do wygenerowania i zwraca tablicę

srand((unsigned) time(0)); //Potrzebne do generowania liczb losowych
int * GenerowanaTablica = new int[N]; //Deklaracja tablicy dynamicznej na liczby

for (int j=0; j<N; j++){ //for działający N (przesłana do funkcji porządana długość tablicy) razy
    GenerowanaTablica[j]=((rand()%400)-200); //Wykonywanie na rand modulo 200 daje liczby od 0 do 200, a potem odjęcie 100, daje liczby od -100 do 100
}

//Kolejna sekcja do wyświetlania, przeszkadza przy generowaniu dużej ilości tablic
//cout<<"Wygenerowana Tabica = [ ";
//for (int j=0; j<N; j++){
//    cout <<GenerowanaTablica[j]<<" ";
//}
//cout<<"]"<<endl;

return GenerowanaTablica; //Funkcja zwraca wygenerowaną tablicę
}

//Główana funkcja projektu, bierze tablicę, znajduje w niej liczby ujemne i przesuwa je na koniec
```

```
int * GlownaFunkcjaPrzesuwania (int PrzesuwanaTablica[]){ //Funkcja przyjmuje jeden argument, tablicę do przesortowania, i zwraca przesortowaną tablicę


int N=sizeof(PrzesuwanaTablica); //Mierzenie długości tablicy

int TablicaUjemnych[N]; //Tworzenie pustej tablicy gdzie zapisane będą liczby ujemne

int IteratorUjemnych = 0; //Tworzenie iteratora który będzie służył do wstawiania do tablicy ujemnych i do liczenia ich ilości

int TablicaNieujemnych[N]; //Tworzenie pustej tablicy gdzie zapisane będą liczby nieujemne

int IteratorNieujemnych = 0; //Tworzenie iteratora który będzie służył do wstawiania do tablicy nieujemnych i do liczenia ich ilości

int * PrzesunietaTablica = new int[N]; //Tworzenie tablicy dynamicznej na finalny wynik

int IteratorPrzesunietejtablicy = 0; //Tworzenie iteratora który będzie służył do wstawiania do tablicy finalnej


for (int j=0; j<N; j++){ //For wykonuje się N (długość tablicy) razy

  if(PrzesuwanaTablica[j]<0){ //Jeśli element tablicy jest mniejszy od zera to przejście do pierwszej sekcji

    TablicaUjemnych[IteratorUjemnych]=PrzesuwanaTablica[j]; //Zapisywanie liczby do tablicy z ujemnymi

    IteratorUjemnych++; //Zwiększanie o 1 liczby ujmnych elementów i zarazem liczenie ich

  }else{ //Jeśli nie jest mniejszy od zera to do drugiej sekcji

    TablicaNieujemnych[IteratorNieujemnych]=PrzesuwanaTablica[j]; //Zapisywanie liczby do tablicy z nieujemnymi

    IteratorNieujemnych++; //Zwiększanie o 1 liczby nieujmnych elementów i zarazem liczenie ich

  }

}

  for (int j=0; j<IteratorNieujemnych; j++){ //For wykonuje się tyle razy, ile znaleziono liczb nieujemnych

    PrzesunietaTablica[IteratorPrzesunietejtablicy]=TablicaNieujemnych[j]; //Dopisywanie do Finalnej tablicy liczb nieujemnych

    IteratorPrzesunietejtablicy++; //Zwiększanie iteratora elementów w finalnej tabeli i zarazem liczenie ich

  }
```

```
   for (int j=0; j<IteratorUjemnych; j++){//For wykonuje się tyle razy, ile znaleziono liczb ujemnych

      PrzesunietaTablica[IteratorPrzesunietejtablicy]=TablicaUjemnych[j]; //Dopisywanie do Finalnej tablicy liczb ujemnych (to dalej ten sama tablica z poprzedniego fora)

      IteratorPrzesunietejtablicy++; //Zwiększanie iteratora elementów w finalnej tabeli i zarazem liczenie ich (to dalej ten sam iterator z poprzedniego fora)

   }

return PrzesunietaTablica; //Funkcja zwraca przesuniętą tablicę

}


//Funkcja main, tu odpalane są inne fuknce i ich obsługa

int main()

{

   int * Tablica; //Tworzenie tablicy dynamicznej do której będą przypisywane wyniki


   Tablica = WczytanieTablicy (); //Wywołanie funkcji wczytywania tablicy i przypisanie wyjścia do zmiennej "Tablica" (czyli tablicy z pliku)


   Tablica = GlownaFunkcjaPrzesuwania(Tablica); //Wywołanie głównej funkcji gdzie argumentem jest tablica wczytana z pliku


   //Wypisanie wyniku przesuwania

   cout<<"Tabica po przesunieciu = [ ";

   for (int j=0; j<sizeof(Tablica); j++){

      cout<<Tablica[j]<<" ";

   }

   cout<<"]"<<endl;


   fstream zapisywanie; //Tworzenie zmiennej, która będzie przekazywać dane do pliku, typu do tego przeznaczonego

   zapisywanie.open("Wynik.txt", ios::out); //Otwieranie nowego pliku tekstowego do którego zapisywany będzie wynik
```

```
   zapisywanie<<"Tabica po przesunieciu = [ "; //Zamiast "cout" mamy zmienną "zapisywanie" więc wszytko idzie do pliku txt
   for (int j=0; j<sizeof(Tablica); j++){
     zapisywanie<<Tablica[j]<<" ";
   }
   zapisywanie<<"]"<<endl;
   zapisywanie.close(); //Zamknięcie pliku


   fstream zapisywanie2; //Tworzenie zmiennej, która będzie przekazywać dane do pliku, typu do tego przeznaczonego
   zapisywanie2.open("Czasy.txt", ios::out); //Otwieranie nowego pliku tekstowego do którego zapisywane będą wyniki


   for (int ile=100000; ile<250000; ile+=100){ //For który wykonuje się 1500 razy, wartość iteratora jest na raz też długością tablicy którą każemy wygenerować programowi. Jest ona tak duża bo co dopiero taka długość daje jakikolwiek czas działania


      Tablica = GenerowanieTablic (ile); //Wywołanie generowania tablic o długości zgodnej z iteratorem for'a


      auto start = high_resolution_clock::now(); //Start mierzenia czasu


      GlownaFunkcjaPrzesuwania(Tablica); //Wywołanie głównej funkcji z wygnenrowaną tablicą


      auto koniec = high_resolution_clock::now(); //Koniec mierzenia czasu
      auto duration = duration_cast<microseconds>(koniec - start); //Odejmowanie czasu początkowego od końcowego i konwersja na microsekundy
      zapisywanie2<<duration.count()<<endl; //zapisywanie każdego czasu do plik
   }
   zapisywanie.close(); //zamknięcie pliku

return 0;
}
```