14 Wektor-projekt

Autor: Tomasz Szewczyk

Prowadzący: prof. Jerzy Respondek

Rok akademicki:2020/2021

Kierunek: informatyka

Rodzaj studiów: SSI

Semestr: 3

Termin labolatorium: poniedziałek i piątek każdego tygodnia

Sekcja: 2

Termin oddania sprawozdania 10.11.2020

1.Treść zadania

Klasa implementująca 1-wymiarowe wektory liczbowe. Klasa powinna umożliwiać i zawierać:
- Konstruktor z parametrem umożliwiającym określenie rozmiaru
- Metodę umożliwiającą dostęp do elementu wektora, zarówno do odczytu jak i zapisu.
- Przeciążone operatory umożliwiające dodawanie, odejmowanie i iloczyn skalarny wektorów.
- Operator mnożenia powinien umożliwiać przekazania jako argument zarówno liczby jak i wektora.
- Przeciążone operatory +=, -=, *=, zakodowane w postaci inline za pomocą wcześniej zaimplementowanych operatorów +, -, *.
- Metody obliczania normy (suma kwadratów elementów), średniej i ekstremów (max, min).
- Sortowanie wektora.
- Wyszukiwanie elementu liniowe i połówkowe.

Czym jest wektor?

Wektor – obiekt matematyczny opisywany za pomocą wielkości: modułu (nazywanego też – zdaniem niektórych niepoprawnie – długością lub wartością, kierunku wraz ze zwrotem (określającym orientację wzdłuż danego kierunku); istotny przede wszystkim w matematyce elementarnej, inżynierii i fizyce.

Reprezentacja wektorów za pomocą współrzędnych umożliwia wyrażenie cech algebraicznych wektorów w dogodny liczbowy sposób.

Struktura danych:

Jako strukturę danych przechowujących zawartość kontenera użyję **tablicę przydzielaną dynamicznie o stałym rozmiarze.** Struktura ta będzie przechowywać wartości wektora typu **int.**

```
int* tab;
```

Konstruktory:

```
Wektor(int rozmiar) : tab(new int[rozmiar]), pojemnosc(0), rozmiar_wektora(rozmiar)
  {
  }
```

Konstruktor będzie wywołany poprzez parametr **rozmiar**.

Będzie on określał rozmiar wektora oraz zaaloluje odpowiednią ilość w pamięci.

Domyślna **pojemność wektora** logicznie wynosi 0.

Będzie także przechowywać informacje o tym jaki rozmiar ma wektor w zmiennej **rozmiar_wektora.**

Konstruktor kopiujący

```
Wektor(const Wektor& wektor):rozmiar_wektora(100),tab(new int [100]), pojemnosc(wektor.pojemnosc)//konstruktor kopiujacy
  {
    memcpy(tab, wektor.tab, sizeof(int) * rozmiar_wektora);

  }
```

Konstruktor ten kopiuje nasz wektor. Dzięki temu dużo kodu zostaje zaoszczędzone. Przykładowo tworząc przeciążony operator, którego zadaniem jest dodanie dwóch wektorów do siebie i utworzenie trzeciego zamiast przepisywać na nowo wektor poprostu go kopiuję za pomocą teog wektora.

Destruktor

```
~Wektor()
  {
  delete[]tab;
  }
```

Przeciążone operatory umożliwią dodawanie, odejmowanie i iloczyn skalarny wektorów. Przykład operacji dodawania wektorów:

$$(1,2,3)+(-2,0,4)=(1-2,2+0,3+4)=(-1,2,7).$$

Kod:

```
friend Wektor operator + (const Wektor& lewy,const Wektor& prawy);
```

To deklaracja operatora w klasie

Definicja powyższego operatora:

```
Wektor operator+(const Wektor& lewy,const  Wektor& prawy)
{
  Wektor suma(lewy);
    for (int i = 0; i < lewy.pojemnosc; i++)
    {
      suma.tab[i] = suma.tab[i] + prawy.tab[i];
    }

    return suma;

}
```

Odejmowanie wektorów:

$$\vec{a} = [4, 2] \quad \vec{b} = [5, 0]$$

$$\vec{c} = \vec{a} - \vec{b} = [4, 2] - [5, 0] = [4 - 5, 2 - 0] = [-1, 2]$$

Definicja:

```
Wektor operator-(const Wektor& left, const Wektor& right)
 {
   Wektor roznica(left);
   for (int i=0;i<left.pojemnosc;i++)
   {
     roznica.tab[i] = roznica.tab[i] - right.tab[i];
   }
   return roznica;
 }
```

Iloczyn skalarny wektorów:

$$\vec{a} \circ \vec{b} = [a_1, a_2] \circ [b_1, b_2] = a_1 b_1 + a_2 b_2$$

Iloczyn skalarny wektorów został zaimplementowany na dwa sposoby. Jeden, w którym robimy iloczyn skalarny dwóch wektorów, drugi zaś to iloczyn skalarny wektora najpierw przemnożonego przez skalar.

Nagłówki:

```
friend int operator *(const Wektor&left, int x);
friend int operator*(const Wektor& left, const Wektor& right);
```

Kod:

```
int operator*(const Wektor& left, const Wektor& right)
 {
  Wektor iloczyn_skalarny(left);
  int ilocz_skal = 0;
```

```
  for (int i = 0; i < iloczyn_skalarny.pojemnosc; i++)
  {
    ilocz_skal = ilocz_skal + (iloczyn_skalarny.tab[i] * right.tab[i]);


  }
  cout  <<endl<<"Iloczyn skalarny wynosi:" << ilocz_skal << endl;
  return ilocz_skal;
 }



int operator*(const Wektor& left, int x)
{
  Wektor mnoz_przez_skalar(left);
  int mnoz_wektora_przez_skalar = 0;
  for (int i = 0; i < mnoz_przez_skalar.pojemnosc; i++)
  {
    mnoz_wektora_przez_skalar = mnoz_wektora_przez_skalar + (mnoz_przez_skalar.tab[i] * x);
  }
  cout << "Iloczyn skalarny po uwczesnym przemnozenie przez skalar: " << mnoz_wektora_przez_skalar <<endl;
  return mnoz_wektora_przez_skalar;
}
```

Przeciążone operatory +=, -=, *=:

Zakodowane są w postaci inline. Wykorzystują wcześniej zaimplementowane operatory +,-,*.

Nagłówek:

```
friend inline Wektor& operator-=( Wektor& lewy, const Wektor& prawy);
friend inline Wektor& operator+=(Wektor& lewy, const Wektor& prawy);
friend inline int operator*=(const Wektor& left, const Wektor& right);
friend inline int operator*=(const Wektor& left, int x);
```

,

Definicje:

```
inline Wektor& operator+=(Wektor& lewy,  Wektor& prawy)
{
  Wektor suma(lewy);
  suma = lewy + prawy;
  lewy = suma;
  return lewy;
}

inline int operator*=(const Wektor& left, const Wektor& right)
{
  int ilocz_skal;
  Wektor iloczyn_skal(left);

  ilocz_skal = left *right;
  return ilocz_skal;
}

inline int operator*=(const Wektor& left, int x)
{
  int mnoz_przez_skalar;
  Wektor mnozenie(left);
  mnoz_przez_skalar = left * x;
  return mnoz_przez_skalar;
}
```

Wszystkie te operatory **inline** są umieszczone w plikach nagłówkowych pod klasą.

METODY:

Obliczenie normy wzór:

$$|\vec{a}| = \sqrt{\sum_{i=1}^{n} a_i{}^2}$$

Kod:

```
void Wektor::norma(const Wektor& wektor)
{
  float norma = 0;
  for (int i = 0; i < wektor.pojemnosc; i++)
  {
    norma = norma + (wektor.tab[i] * wektor.tab[i]);
  }
  cout << "Norma wektora: " << norma <<endl;

}
```

Wzór na średnią- w tym przypadku zastosuję **średnią arytmetyczną**:

$$\bar{x} = \frac{x_1 + x_2 + x_n}{N}$$

Kod:

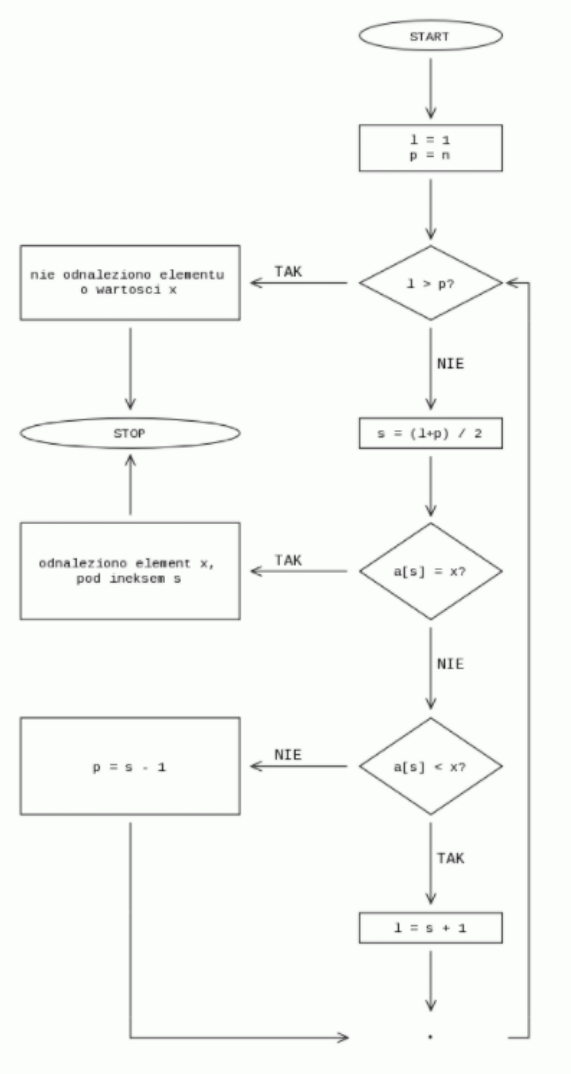
```
void Wektor::srednia_arytmetyczna(const Wektor& wektor)
{

  float wynik=0;
  int licznik=0;
  for (int i = 0; i < wektor.pojemnosc; i++)
  {
    wynik = wynik + wektor.tab[i];
    licznik++;
  }
  cout << "Srednia arytmetyczna wynosi: " << (wynik / licznik);

}
```

Wyznaczenie ekstremów to wyznaczenie elementu minimalnego wektora i maksymalnego.

```
void Wektor::wyszukaj_max(const Wektor& wektor)
{
  int max = wektor.tab[0];

  for (int i = 0; i < wektor.pojemnosc - 1; i++)
  {
    if (tab[i] < tab[i + 1])
    {
      max = tab[i + 1];
    }
  }
  cout << "Wartosc maksymalna: " << max <<endl;

}

void Wektor::wyszukaj_min(const Wektor& wektor)
{
  int min = wektor.tab[0];

  for (int i = 0; i < wektor.pojemnosc-1; i++)
  {
    if (tab[i] > tab[i + 1])
    {
      min = tab[i + 1];
    }
  }
  cout << "Wartosc minimalna: " << min << endl;

}
```

Wyszukiwanie elementu połówkowe

Algorytm ten ma bardzo dobrą złożoność obliczeniową wynoszącą O(log n), co oznacza, że czas potrzebny na wyszukanie elementu tą metodą rośnie w sposób logarytmiczny wraz z liniowym wzrostem liczby elementów w przeszukiwanej tablicy. Schemat algorytmu:



Kod:

```
void Wektor::wyszukiwanie_polowkowe(const Wektor& wektor, int x)
{
  int licznik = 0;
  int p;
  int s;

  for (int i = 0; i < wektor.pojemnosc; i++)
  {
    licznik++;
  }
  int l = 0;
  p = licznik - 1;
  while (true)
  {
    if (l > p)
    {
      cout << "Nie odnaleziono szukanego elementu" << endl;
      break;
    }
    s = (l + p) / 2;
```

```
    if (wektor.tab[s] == x)
    {
      cout << "Odnaleziono liczbe " << x << " pod indeksem " << s  << endl;
      break;
    }
    if (wektor.tab[s] < x)
      l = s + 1;
    else
      p = s - 1;
  }
}
```

Liniowe wyszukiwanie elementu:

Wyszukiwanie liniowe (ang. linear search), zwane również sekwencyjnym (ang. sequential search) polega na przeglądaniu kolejnych elementów zbioru Z. Jeśli przeglądany element posiada odpowiednie własności (np. jest liczbą o poszukiwanej wartości), to zwracamy jego pozycję w zbiorze i kończymy. W przeciwnym razie kontynuujemy poszukiwania aż do przejrzenia wszystkich pozostałych elementów zbioru Z.

Kod:

```
void Wektor::wyszukiwanie_liniowe(const Wektor& wektor, int x)
{
  for (int i = 0; i < wektor.pojemnosc; i++)
  {
    if (wektor.tab[i] == x)
    {
      cout << "Znaleziono liczbe " << x << " pod indeksem " << i <<endl;
    }
  }
}
```

INTERFEJS KLAS:

```
class Wektor
{
private:
  int pojemnosc;
  int* tab;
  int rozmiar_wektora;

public:
  Wektor& operator=(const Wektor& right) {
    if (&right != this) {
      tab = new int[right.pojemnosc];

      for (int i = 0; i < pojemnosc; i++)
      {
        tab[i] = right.tab[i];
      }
      rozmiar_wektora = right.rozmiar_wektora;

    }
    return *this;
  }

  friend ostream& operator<<(ostream& output, Wektor& wektor);
  friend Wektor operator + (const Wektor& lewy,const Wektor& prawy);
  friend inline Wektor& operator+=(Wektor& lewy, const Wektor& prawy);
  friend Wektor operator-(const Wektor& left, const Wektor& right);
  friend inline Wektor& operator-=( Wektor& lewy, const Wektor& prawy);
  friend int operator*(const Wektor& left, const Wektor& right);
  friend inline int operator*=(const Wektor& left, const Wektor& right);
  friend int operator *(const Wektor&left, int x);
  friend inline int operator*=(const Wektor& left, int x);
  void dodaj_do_wektora(int x);
```

```
  void sortuj(const Wektor& wektor);
  void wyszukaj_min(const Wektor & wektor);
  void wyszukaj_max(const Wektor & wektor );
  void wyszukiwanie_polowkowe(const Wektor& wektor,int x);
  void wyszukiwanie_liniowe(const Wektor& wektor, int x);
  void srednia_arytmetyczna(const Wektor& wektor);
  void norma(const Wektor& wektor);
  Wektor(int rozmiar) : tab(new int[rozmiar]), pojemnosc(0), rozmiar_wektora(rozmiar)
  {
  }
  Wektor(const Wektor& wektor):rozmiar_wektora(100),tab(new int [100]), pojemnosc(wektor.pojemnosc)//konstruktor kopiujacy
  {
    memcpy(tab, wektor.tab, sizeof(int) * rozmiar_wektora);

  }
  ~Wektor()
  {
  delete[]tab;
  }
};
```

PROGRAM TESTUJĄCY:

```
int main()
 {
  cout << "Wektor1: ";
  Wektor x1(100);
  x1.dodaj_do_wektora(1);
  x1.dodaj_do_wektora(1);
  x1.dodaj_do_wektora(-1);


  cout << x1 << endl;
  Wektor x2(100);
  x2.dodaj_do_wektora(11);
  x2.dodaj_do_wektora(2);
  x2.dodaj_do_wektora(2);
  cout <<"Wektor2: "<< x2 << endl;
  Wektor x100(199);
  x100.dodaj_do_wektora(2);
  x100.dodaj_do_wektora(1);
  x100.dodaj_do_wektora(-100);
  x100.dodaj_do_wektora(250);
  cout << "Wektor 100:" << x100<<endl;
  Wektor x5(100);
  x5.dodaj_do_wektora(10);
  x5.dodaj_do_wektora(11);

  cout << "Wektor 5: "  << x5<<endl;
  //dodawanie
  Wektor x3 = x1 + x2;

  cout << "Dodawanie-wektor 1+2: " << x3;

  //iloczyn skalarny
  Wektor x6 = x1 * x2;

  //mnozenie przez skalar
  Wektor x7 = x1 * 2;
  x1 -=x2;
  cout << "Skrócony operator odejmowania x1,z2: " << x1;
  cout << endl;
  x1=x1 += x2;
  cout << "Skrócony operator += x1 (po operacji odejmowania),x2: " << x1;
  x1 *= x2;
  cout << endl;
  x1 *= 2;
  //SORTOWANIE
  x100.sortuj(x100);
```

```
  cout  << x100 << endl;
  //SZUKANIE MINIMUM
  x100.wyszukaj_min(x100);
  //SZUKANIE MAXIMUM
  x100.wyszukaj_max(x100);
  cout << x100 << endl;
  //WYSZUKIWANIE POLOWKOWE
  x100.wyszukiwanie_polowkowe(x100, 2);
  //WYSZUKIWANIE LINIOWE
  x100.wyszukiwanie_liniowe(x100, 2);
  //SREDNIA ARYTMETYCZNA
  x100.srednia_arytmetyczna(x100);
  x2.norma(x2);
  //
  //
}
```

WNIOSKI:

Stworzenie klasy Wektor okazało się być wymagającym zadaniem oraz pokazuje wiele zagadnień dotyczących programowania obiektowego. Największym wyzwaniem było pozbycie się błędów związanych z destruktorem. Projekt był dla mnie wyzwaniem, ponieważ czułem presję związaną z czasem lecz podołałem zadaniu. Projekt pozwolił mi na rozwinięcie swoich umiejętności twardych.