Zadanie 0 – Funktory (1pkt)

Klasa Interpolator implementuje interpolację funkcji $\mathcal{R} \to \mathcal{R}$ metodą wielomianu Lagrange'a. W momencie tworzenia obiektu tej klasy zachowywane są znane wartości funkcji w pewnych punktach. Przeciąż dla tej klasy operator () tak, żeby można było wywołać jej obiekt dla punktu, w którym chcemy interpolować funkcję i wynikiem była liczba typu double (interpolowana wartość funkcji).

Przydatny wzór

$$L_f(x) = \sum_{i=0}^{n} f(x_i) \prod_{j=0, i\neq j}^{n} \frac{x - x_j}{x_i - x_j} \tag{1}$$

Szablon programu:

```
#include <iostream>
#include <vector>
using namespace std;

class Interpolator
{
private:
        vector<double> *x;
        vector<double>* y;
public:
        Interpolator()
        {
                x = nullptr;
                y = nullptr;
        }
        Interpolator(vector<double> &x, vector<double> &y)
        {
                if(x.size() != y.size())
                {
                        std::cout << "[ERR] X and Y dimension mismatch!";
                        exit(1);
                }
                this->x = new vector<double>(x);
                this->y = new vector<double>(y);
        }
        ~Interpolator()
        {
                if(x) delete x;
                if(y) delete y;
        }
};

int main()
{
        vector<double> X;
        vector<double> Y;
        for(int ii = -20; ii<20; ii++)
```

```
        {
                X.push_back(ii);
                Y.push_back( (ii-1)*(ii+2)*(ii+7) );
        }
        //tworzenie obiektu
        Interpolator interp(X,Y);
        for(int ii=-2; ii<7; ii++)
                cout << interp(ii+0.5) << " "
                << (ii+0.5-1)*(ii+0.5+2)*(ii+0.5+7)
                << std::endl;
        return 0;
}
```

Output:
-6.875 -6.875
-14.625 -14.625
-9.375 -9.375
14.875 14.875
64.125 64.125
144.375 144.375
261.625 261.625
421.875 421.875
631.125 631.125

Iteratory dostępu bezpośredniego

W zadaniach tej serii należy założyć, że iteratory przekazywane do funkcji są *iteratorami dostępu bezpośredniego*, tzn. są dla nich zdefiniowane operatory: $++, --, +, -, +=, -=, ==, !=, <, >, <=, >=$. Tego typu są iteratory dla obiektów klasy *std* :: *vector* <>, których używaliśmy wielokrotnie do tej pory.

Zadanie 1 – sortowanie szybkie (2pkt)

Napisz szablon funkcji

```
template <class RandomIt>
void quickSort(RandomIt begin, RandomIt end)
```

wykonującej szybkie sortowanie pomiędzy dwoma iteratorami wolnego dostępu. Algorytm rekurencyjny sortowania szybkiego:

1. Jeżeli sortowany ciąg nie jest jednoelementowy to wykonaj natępne kroki.
2. Wybierz jeden z elementów, np. pierwszy. Nazwijmy ten element PIVOT.
3. Przenieś wszystkie elementy mniejsze od PIVOT na lewo od niego, a wszystkie elementy większe od niego na prawo. W ten sposób PIVOT będzie na właściwym miejscu (posortowany).
4. Powtórz rekurencyjnie dla dwóch tablic: pierwszej składającej się z liczb stojących przed PIVOTem, drugiej dla liczb stojących za PIVOTem.

Przydatne będzie użycie pomocniczej funkcji dokonującej zamiany miejscami i zwracającej iterator na PIVOT

```
template <class RandomIt>
RandomIt partition(RandomIt begin, RandomIt end);
```

Szablon:

```
#include <algorithm>
#include <iostream>
#include <vector>

template <class RandomIt>
RandomIt partition(RandomIt begin, RandomIt end);

template <class RandomIt>
void quickSort(RandomIt begin, RandomIt end);

int main()
{
        std::vector<int> v = {11, -3, 2, 4, 0, 6, 4, -2, 2, 3, 1};
        quickSort(v.begin(), v.end());
        std::for_each(v.begin(), v.end(), [](auto x)
                { std::cout << x << " "; });
        std::cout<<std::endl;
        return 0;
}
```

Uwaga: Nie tworzyc nowych tablic ani kolekcji. Algorytm ma być zaimplementowany rekurencyjnie i ma być poprawnym sortowaniem szybkim, a nie innym rodzajem. Nie używać funkcji sortujących z bibliotek, tylko napisać własną implementację.
Podpowiedź: Przydatne będą funkcje, których używaliśmy do działań na iteratorach w zadaniach na poprzednich zajęciach.
Dla chętnych: Dodaj opcję, żeby przy wywołaniu szablonu można było zdecydować, czy sortowanie ma być rosnące czy malejące.

Zadanie 2 – sortowanie przez kopcowanie (2pkt)

Napisz szablon funkcji implementujący sortowanie przez kopcowanie (ang. heap sort)

```
template<typename I>
void heapSort(I begin, I end);
```

Algorytm jest następujący (dla sortowania malejącego):

1. Nadaj sortowanej tablicy strukturę kopca, tj. drzewa binarnego w którym każdy rodzic jest nie mniejszy od swoich dzieci.
2. Największym elementem w tablicy jest korzeń kopca. Weź go i umieść na końcu tablicy.
3. Przywróć strukturę kopca na wycinku tablicy nie zawierającym ostatniego elementu.
4. Powtarzaj procedurę zwiększając posortowaną część tablicy kosztem kopca.
5. Zakończ w momencie, gdy wszystkie elementy znajdą się w posortowanej części tablicy.

Będziesz potrzebować drugiego szablonu funkcji, którego zadaniem jest tworzenie struktury kopca w tablicy:

```
template<typename I>
void heapify(I begin, I end, std::size_t current);
```

Iteratory begin oraz end wskazują odpowiednio na początek i koniec tablicy. Zmienna current wskazuje na indeks rodzica (na raz rozważamy jedynie rodzica i jego nie więcej niż 2 dzieci). Uwaga: Funkcja powinna być rekurencyjna. Rozwiązania bez rekurencji będą mogły otrzymać max 70% dostępnych punktów. Należy pamiętać o poprawnym zakończeniu rekurencji.
Szablon:

```
#include <iostream>
#include <vector>
#include <algorithm>

template<typename I>
void heapify(I begin, I end, std::size_t current);

template<typename I>
void heapSort(I begin, I end);

int main()
{
        std::vector<int> v = {25, 1, 3, 13, 2, 8, 0, -2, 4,
         -2, 2, -3, -1, 1, 24};
        heapSort(v.begin(), v.end());
        std::for_each(v.begin(), v.end(), [](auto x)
         { std::cout << x << " "; });
        std::cout<<std::endl;
        return 0;

}
```

Uwaga: Nie tworzyć nowych tablic ani kolekcji.
Podpowiedź: Pomocne mogą być funkcje do operacji na iteratorach, które poznaliśmy w trakcie poprzednich zajęć. Może się również przydać $std :: distance(Iit1, Iit2)$, która liczy odległość między dwoma elementami wskazywanymi przez iteratory it1 i it2.
Dla chętnych: Dodaj opcję, żeby przy wywołaniu szablonu można było zdecydować, czy sortowanie ma być rosnące czy malejące.