Zadanie 0 – Sortowanie przez scalanie (1*pkt)

Napisz szablon funkcji

```
template <class RandomIter>
void mergeSort(RandomIter begin, RandomIter end)
```

wykonującej sortowanie przez scalanie pomiędzy dwoma iteratorami swobodnego dostępu.
Algorytm **rekurencyjny** sortowania przez scalanie:

1. Jeżeli sortowany ciąg nie jest jednoelementowy wykonaj następne kroki.
2. Znajdź środkowy element ciągu. Podziel ciąg na dwa podciągi: pierwszy od elementu pierwszego do środkowego, drugi od elementu środkowego do ostatniego elementu.
3. Wykonaj sortowanie przez scalanie dla obu podciągów.
4. Scal posortowane podciągi.

Najtrudniejszym elementem sortowania jest procedura scalania posortowanych podciągów. Zamiast pisać własną możesz wykorzystać algorytm biblioteki standardowej. Do scalania posortowanych podciągów (pierwszy od iteratora first do middlle, drugi od middle do last) możesz użyć $std :: inplace_merge(first, middle, last)$.
Zadanie z użyciem tej funkcji staje się bardzo proste.
Jeżeli napiszesz własną procedurę scalania otrzymasz dodatkowy punkt! Warunkiem jest, żeby nie używała żadnych wysokopoziomowych funkcji scalających oraz, oczywiście, musi działać poprawnie dla różnych danych, nie tylko dla przykładowych. Ponieważ jest to zadanie dodatkowe, nie będę w nim pomagał i trzeba poradzić sobie samemu.
Do znalezienia iteratora na środkowy element możesz wykorzystać arytmetykę iteratorów (analogicznie jak dla wskaźników), operator $\pm int$ daje oterator przesunięty o int w prawo dla + i w lewo dla -. Odległość między dwoma iteratorami można znaleźć odejmując je od siebie i rzutując na typ int, albo z użyciem funkcji $std :: distance(first, second)$.
Szablon:

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


template <class RandomIter>
void mergeSort(RandomIter begin, RandomIter end);

int main() {
std::vector<int> v = {6, 4, 2, 2, 3, 1, 24, -12, 0, 3, 1};
mergeSort(v.begin(), v.end());
std::for_each(v.begin(), v.end(), [](auto x) { std::cout << x << " "; });
return 0;
}
```

Output: Posortowana rosnąco tablica.
Uwaga: Nie korzystać z gotowych implementacji sortowania przez scalanie. Trzymać się algorytmu i nie łączyć różnych metod sortowania. Nie tworzyć nowych tablic czy kolekcji – posortować istniejący wektor.

1 Zadanie 1 – lambdy (2pkt)

Celem tego zadania jest oswojenie się ze składnią funkcji lambda. W tym celu trzeba napisać 10 trywialnych funkcji, każda będzie warta 0.1+0.1 punktów (poprawność i terminowość).
Szbalon

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

using namespace std;

int main()
{
        vector<double> v =
        {12.54, -7.2, 9.09, 3.561, 0.0, -4.2135, 6.0009, 12.1, 2.45, -3.1};

        auto wypisz = [&v]()->void
        {
                for(auto em : v)
                        cout << em << " ";
                cout << endl;
        };

        auto sum = ...
        auto av = ...
        auto std = ...
        auto max = ...
        auto min = ...
        auto square = ...
        auto inverse = ...
        auto multKtimes = ...
        auto toInt = ...
        auto modify = ...

        wypisz();

        cout << setprecision(3) << "SUMA=" << sum()
        << " SREDNIA=" << av()
        << " ODCHYLENIE=" << std()
        << " MAX=" << max()
        << " MIN=" << min()
        << endl;

    square();
        wypisz();
        inverse();
        wypisz();
        multKtimes(1241.78);
```

```
        wypisz();
        toInt();
        wypisz();
        modify(-13, 50);
        wypisz();
        return 0;
}
```

Funkcje sum, av, std, max, min zwracają kolejno sumę, średnią, odchylenie standardowe, element największy, element najmniejszy z wektora v. **Te funkcje mają być napisane tak, żeby programistycznie nie były w stanie zmienić wektora v.**
Teraz funkcje, które **modyfikują** wektor v: Funkcja square zamienia wszystkie liczby w wektorze v na ich kwadraty. Funkcja inverse zamienia liczby na ich odwrotności, chyba, że napotka 0, wtedy zostawia 0 (a nie np. inf). Funkcja multKtimes mnoży wszystkie elementy wektora v przez liczbę podaną jako argument. Funkcja toInt pozbywa się części ułamkowej liczb z wektora (typ wektora się ostatecznie nie zmienia).
Funkcja modify przyjmuje dwa argumenty typu double, pierwszy określa nową liczbę, którą dodamy do wektora v, drugi ustala próg, po przekroczeniu którego mamy dodać nowy element. Działanie jest następujące: iterujemy po wszystkich elementach wektora v, jeżeli napotkamy element większy od drugiego argumentu to zmieniamy mu znak na przeciwny i dodajemy **za nim** nowy element wektora równy pierwszemu argumentowi (patrz przykładowy output). Efektem działania jest dodanie nowych elementów do wektora oraz zmiana znaku niektórych starych elementów. Aby dodać element VAL **przed** elementem wskazywanym przez iterator ITER, można użyć funkcji $v.insert(ITER, VAL)$.
Output:
12.54 -7.2 9.09 3.561 0 -4.2135 6.0009 12.1 2.45 -3.1
SUMA=31.2 SREDNIA=3.12 ODCHYLENIE=6.5 MAX=12.5 MIN=-7.2
157 51.8 82.6 12.7 0 17.8 36 146 6 9.61
0.00636 0.0193 0.0121 0.0789 0 0.0563 0.0278 0.00683 0.167 0.104
7.9 24 15 97.9 0 69.9 34.5 8.48 207 129
7 23 15 97 0 69 34 8 206 129
7 23 15 -97 -13 0 -69 -13 34 8 -206 -13 -129 -13

Uwaga: W tym zadaniu nie piszemy standardowych funkcji tylko funkcje lambda. W outpucie można zauważyć, że konwersja na liczbę całkowitą zmieniła 207 na 206. Nie ma tutaj błędu, bo tak na prawdę nie ma lcizby 207 tylko 206.X gdzie $X >= 5$. Z powodu ustawionej precyzji nie wyświetliły się cyfry po przecinku.
Podpowiedź: Nie trzeba tutaj pisać ogólnego kodu, w szczególności nie trzeba przekazywać wektora jako argumentu, tylko dać funkcji lambda dostęp do zmiennej v.

2 Zadanie 2 – składowe statyczne (1pkt)

Klasa Box opisuje prostopadłościenne pudełko:

```
#include <iostream>

class Box
{
public:
        double a;
        double b;
        double c;
```

```
        Box(double x=0, double y=0, double z=0)
        : a(x), b(y), c(z){}
        double volume(){return a*b*c;}
};


int main()
{
        Box(5.2, 1., 9.);
        Box(11, 4.35, 5.5);
        Box(8., 2., 1.);
        Box(4., 4., 4.);
        Box(1., 2., 3.);

        std::cout << "No of boxes: "
        << Box::getCounter()
        << std::endl;
        return 0;
}
```

Rozbuduj klasę Box o dwie składowe statyczne: pole typu int przechowujące informację o ilości stworzonych obiektów typu Box (pole trzeba zainicjalizować zerem) oraz metodę statyczną *intgetCounter*(), która zwraca wartość pola statycznego.
Wywołanie kodu z przykładu po uzupełnieniu klasy Box o składowe statyczne powinno spowodować wyświetlenie
No of boxes: 5
Uwaga: Proszę jawnie zainicjalizować pole statyczne.
Podpowiedź: Składowe statyczne są opisane w teoretycznym minimum, jest na nie przykład w ćwiczeniach. Zmienną statyczną w zadaniu trzeba zainicjalizować poza ciałem klasy, używając operatora dostępu do przestrzeni nazw, czyli ::.

3 Zadanie 3 – set i map (1pkt)

Paulina należy do ESN i zebrała informacje o grupie studentów z programu Erasmus+ i wpisała je do funckji main:

```
#include <vector>
#include <map>
#include <set>
#include <iostream>
#include <string>

int main()
{
        std::vector<std::string> imiona =
        {"Jose", "Mark", "Maria", "Antonio", "Lorenzo", "Anna",
         "Mark", "Anna", "Carol", "Jose", "Maria", "Anna", "Hans"};

        std::vector<std::string> kraje =
         {"Hiszpania", "Holandia", "Portugalia", "Wlochy", "Wlochy",
         "Czechy", "Szwecja", "Hiszpania", "Niemcy", "Hiszpania",
```

```
          "Portugalia", "Niemcy", "Norwegia"};

        std::vector<std::string> kierunki =
        {"Socjologia", "Ekonomia", "Psychologia", "Ekonomia",
        "Fizyka", "Biotechnologia", "Psychologia", "Socjologia",
        "Psychologia", "Medycyna", "Ekonomia", "Psychologia", "Fizyka"};

        std::vector<std::string> urodziny =
        {"Maj", "Kwiecien", "Styczen", "Marzec", "Listopad",
        "Grudzien", "Maj", "Kwiecien", "Wrzesien", "Grudzien",
        "Styczen", "Marzec", "Sierpien"};

        /* Tutaj stworz mape, dodaj set stworzone z wektorow i wypisz */
        return 0;
}
```

Uzupełnij funkcję main, wg poniższych wytycznych:

1. Stwórz obiekt typu $std :: map$, gdzie klucze będą typu $std :: string$ a wartości typu $std :: set < std :: string >$. Typ $std :: string$ to zawarty w pliku $< string >$ kontener na napisy, ułatwiający ich manipulowaniem, np. pozwalający przepisywać napisy tak jak zwykłe zmienne a nie jak tablice znaków. Nie musisz znać jego szczegółów, wiedz tylko, że często da się zrobić niejawną konwersję z łańczucha znaków, np. "Napis", na obiekt std::string.

2. Dodaj do mapy dane zebrane przez Paulinę. W tym celu trzeba stworzyć obiekty typu set z istniejących wektorów i dodać je do mapy z odpowiednimi kluczami. Klucze mogą być takie jak nazwy zmiennych (ale musisz je wpisać ręcznie).

3. Wypisz na standardowe wyjście klucze i wartości z mapy w formacie
   KLUCZ : WART WART WART

Output:
Imiona : Anna Antonio Carol Hans Jose Lorenzo Maria Mark
Kierunki : Biotechnologia Ekonomia Fizyka Medycyna Psychologia Socjologia
Kraje : Czechy Hiszpania Holandia Niemcy Norwegia Portugalia Szwecja Włochy
Urodziny : Grudzien Kwiecien Listopad Maj Marzec Sierpien Styczen Wrzesien

Podpowiedź: Mapy i zbiory są opisane w teoretycznym minimum, są na nie przykłady w ćwiczeniach. Obiekty typu set można szybko stworzyć z wektorów za pomocą iteratorów. Aby wypisać elementy z mapy również należy użyć iteratorów, podobnie jak przy wypisywaniu elementów wektora. Różnica polega na tym, że iterator w mapie pokazuje do dwóch obiektów, do klucza i do wartości. Jeżeli iterator nazywa się IT to dostęp do klucza dostajemy przez $IT- > left$ a do wartości przez $IT- > right$. Ponieważ wartościami w mapie są kolekcje set, to przy wypisywaniu trzeba będzie zagnieździć pętle for, żeby wypisać wszystko.