Zadanie 1

Napisz *przeciążone* funkcje

```
void ord3(double& a, double& b, double& c);
void ord3(double* a, double* b, double* c);
```

które pobierają trzy liczby typu **double** odpowiednio przez referencje i przez wskaźniki i porządkują je w kolejności wzrastającej (tak, że *po wyjściu* z funkcji ich wartości są zmienione).

Napisz *przeciążone* funkcje

```
void getMinMax(double &a, double& b, double& c,
               double*& ptrMin, double*& ptrMax);
void getMinMax(double *a, double* b, double* c,
               double** ptrMin, double** ptrMax);
```

które pobierają trzy liczby typu **double** odpowiednio przez referencje i przez wskaźniki i do wskaźników ptrMin i ptrMax przekazanych przez, odpowiednio, referencje i wskaźniki, wstawiają adresy zmiennych odpowiadających najmniejszej i największej z przekazanych liczb. Wartości przekazanych liczb nie ulegają zmianie.

Uwaga: Nie używaj tablic, napisów ani żadnych kolekcji. Funkcje nie mogą niczego pisać na ekran — wszystkie wyniki są drukowane w funkcji **main**.

Następujący program, beż żadnych zmian w funkcji **main**,

download *RefsPointers.cpp*

```
#include <iostream>
#include <utility>  // you can use std::swap

void getMinMax(double &a, double& b, double& c,
               double*& ptrMin, double*& ptrMax) {
    // ...
}

void getMinMax(double *a, double* b, double* c,
               double** ptrMin, double** ptrMax) {
    // ...
}

void ord3(double& a, double& b, double& c) {
    // ...
}

void ord3(double* a, double* b, double* c) {
```

```
    // ...
}

int main() {
    using std::cout; using std::endl;
    double a, b, c, *ptrMin, *ptrMax;

    a = 2; b = 1; c = 3;
    ord3(a, b, c);
    cout << a << " " << b << " " << c << endl;

    a = 3; b = 2; c = 1;
    ord3(&a, &b, &c);
    cout << a << " " << b << " " << c << endl;

    a = 2; b = 3; c = 1; ptrMin = ptrMax = nullptr;
    getMinMax(a, b, c, ptrMin, ptrMax);
    std:: cout << "Min = " << *ptrMin << "; "
               << "Max = " << *ptrMax << std::endl;

    a = 3; b = 1; c = 2; ptrMin = ptrMax = nullptr;
    getMinMax(&a, &b, &c, &ptrMin, &ptrMax);
    std:: cout << "Min = " << *ptrMin << "; "
               << "Max = " << *ptrMax << std::endl;
}
```

powinien wypisać

```
1 2 3
1 2 3
Min = 1; Max = 3
Min = 1; Max = 3
```

Sprawdź program również dla innych danych; w szczególności dla sytuacji, gdy niektóre lub wszystkie wartości są równe.

Zadanie 2

Napisz funkcję

```
void merge(const int* a, size_t dima,
           const int* b, size_t dimb, int* c);
```

która pobiera dwie *posortowane niemalejąco* tablice a i b o wymiarach, odpowiednio, dima i dimb oraz tablicę c o wymiarze (z założenia) dima+dimb. Zadaniem funkcji jest wpisanie do tablicy c wszystkich elementów z a i b tak, żeby były one również posortowane niemalejąco. Nie wolno przy tym korzystać z żadnych pomocniczych tablic ani kolekcji, a algorytm musi być liniowy, czyli liczba operacji musi być proporcjonalna do dima+dimb (a nie do iloczynu wymiarów czy ich kwadratów).

Na przykład poniższy program (w którym funkcja **printArr** jest tylko pomocniczą funkcją wypisującą zawartość tablicy)

download *ArrMerge.cpp*

```
#include <iostream>

void merge(const int* a, size_t dima,
    // ...
}

void printArr(const int* a, size_t dima, const char* m) {
    std::cout << m << " [ ";
    for (size_t i = 0; i < dima; ++i)
        std::cout << a[i] << " ";
    std::cout << "]\n";
}

int main() {
    int a[] = {1,4,4,5,8};
    int b[] = {1,2,2,4,6,6,9};
    constexpr size_t dima = std::size(a);
    constexpr size_t dimb = std::size(b);
    constexpr size_t dimc = dima + dimb;
    int c[dimc];

    merge(a,dima,b,dimb,c);

    printArr(a,dima,"a");
    printArr(b,dimb,"b");
    printArr(c,dimc,"c");
}
```

powinien wypisać

```
a [ 1 4 4 5 8 ]
b [ 1 2 2 4 6 6 9 ]
c [ 1 1 2 2 4 4 4 5 6 6 8 9 ]
```

Nie włączaj żadnych innych plików nagłówkowych.

Zadanie 3

Sekwencje nukleotydów w cząsteczkach DNA oznaczane są ciągami liter A, C, G i T o dowolnej długości (skróty pochodzą od *adenine*, *cytosine*, *guanine* i *thymine*). Bardzo uproszczona metoda mierzenia podobieństwa między dwoma sekwencjami jest następująca:

- Dla ciągów *o tej samej długości* za każdy identyczny odcinek kodów o długości d doliczamy d^2 punktów; tak obliczona suma jest współczynnikiem podobieństwa

obu sekwencji. Na przykład dla sekwencji `ACGTC` i `AGGTG` będzie to 5: jeden punkt za `A` na pierwszej pozycji i cztery za dwuznakowy ciąg `GT` na pozycjach 3-4.

- Dla sekwencji o niejednakowej długości obliczamy powyższym sposobem współczynniki podobieństwa między ciągiem krótszym, o długości, powiedzmy, d, a każdym podciągiem o tejże długości kolejnych elementów ciągu dłuższego. Obliczonym współczynnikiem jest wtedy największy z tak uzyskanych współczynników częściowych. Na przykład dla ciągów `AGG` i `CGGAT` obliczamy podobieństwa między trójznakowym ciągiem `AGG` a, kolejno, `CGG` (4 punkty), `GGA` (1 punkt) i `GAT` (0 punktów). Tak więc współczynnikiem podobieństwa będzie największa z tych wartości, czyli 4.

Napisz funkcję

```
int simil(const char a[], const char b[]);
```

która pobiera dwie sekwencje i zwraca ich podobieństwo (możesz też, jeśli uznasz to za wygodne, zdefiniować jakąś funkcję pomocniczą). Nie twórz żadnych dodatkowych tablic czy kolekcji i nie włączaj żadnych dodatkowych plików nagłówkowych. Na przykład program

download *DNA.cpp*

```
#include <iostream>

int simil(const char a[], const char b[]) {
    // ...
}

int main() {
    char a[] = "AACTACGTC",
         b[] = "ACGTA";
    std::cout << a << " and " << b << " -> "
              << simil(a,b) << std::endl;
    char c[] = "GCGC",
         d[] = "AGGGCA";
    std::cout << c << " and " << d << " -> "
              << simil(c,d) << std::endl;
}
```

powinien wydrukować

```
AACTACGTC and ACGTA -> 16
GCGC and AGGGCA -> 5
```
