Zadanie 1

Napisz i przetestuj następujący zestaw funkcji *rekurencyjnych* (w ich treści nie może być żadnych pętli!)

- `int gcdRec(int a, int b);` — zwraca największy wspólny dzielnik dwóch liczb naturalnych (całkowitych dodatnich);
- `int sumDigits(int n);` — zwraca sumę (dziesiętnych) cyfr podanej liczby naturalnej (e.g., 34→7);
- `int numDigits(int n);` — zwraca liczbę (dziesiętnych) cyfr podanej liczby naturalnej (e.g., 34→2);
- `void printOddEven(int n);` — drukuje w jednej linii
  - dla `n` parzystego – wszystkie liczby parzyste poczynając od 2 aż do `n`;
  - dla `n` nieparzystego – wszystkie liczby nieparzyste od 1 aż do `n`.
- `void hailstone(int n);` — drukuje w jednej linii wszystkie liczby ciągu Collatza (patrz niżej) od `n` aż do 1.

Ciąg Collatza, znany też jako ciąg Ulama lub „gradowy” (ang. *hailstone*), to ciąg dla którego pierwszy wyraz a_0 jest dodatnią liczbą całkowitą, a kolejne wyrazy są wyliczane ze wzoru

$$a_{n+1} = \begin{cases} a_n/2 & \text{jeśli } a_n \text{ jest parzyste,} \\ 3a_n + 1 & \text{jeśli } a_n \text{ jest nieparzyste.} \end{cases}$$

Hipoteza Collatza (wciąż nieudowodniona) mówi, że niezależnie od wartości pierwszego elementu a_0, po skończonej liczbie kroków ciąg osiągnie wartość 1 (i potem będzie już cyklicznie przybierał wartości 1→4→2→1...).
Następujacy program, po zdefiniowaniu wszystkich funkcji

download *SimpleRekur.cpp*

```
#include <iostream>

int gcdRec(int a, int b);
int sumDigits(int n);
int numDigits(int n);
void printOddEven(int n);
void hailstone(int n);

int main() {
    std::cout << "gcdRec(12, 42) = " <<
                 gcdRec(12, 42) << std::endl
              << "gcdRec(12, 25) = " <<
                 gcdRec(12, 25) << std::endl;
```

```
    std::cout << "sumDigits(123) = " <<
                 sumDigits(123) << std::endl
              << "sumDigits(971) = " <<
                 sumDigits(971) << std::endl;
    std::cout << "numDigits(12345) = " <<
                 numDigits(12345) << std::endl
              << "numDigits(971) = " <<
                 numDigits(971) << std::endl;
    std::cout << "printOddEven(15): ";
    printOddEven(15);
    std::cout << std::endl;
    std::cout << "printOddEven(14): ";
    printOddEven(14);
    std::cout << std::endl;
    std::cout << "hailstone(13): ";
    hailstone(13);
    std::cout << std::endl;
}
```

powinien wypisać

```
gcdRec(12, 42) = 6
gcdRec(12, 25) = 1
sumDigits(123) = 6
sumDigits(971) = 17
numDigits(12345) = 5
numDigits(971) = 3
printOddEven(15): 1 3 5 7 9 11 13 15
printOddEven(14): 2 4 6 8 10 12 14
hailstone(13): 13 40 20 10 5 16 8 4 2 1
```

Zadanie 2

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 3

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.
