Zadanie 0 – To co ostatnio

Napisz funkcję

```
void minMaxRep(int a[], size_t size, int& mn, size_t& in,
                                              int& mx, size_t& ix);
```

która pobiera tablicę intów a, jej wymiar size oraz, przez referencje, cztery zmienne, do których wpisany ma być wynik działania funkcji: mn, in, mx i ix. Funkcja znajduje wartości najmniejszego i największego elementu tablicy i wpisuje je do zmiennych mn i mx, a do in i ix wpisuje, odpowiednio, liczbę wystąpień tej najmniejszej i największej wartości w całej tablicy.
Na przykład program:

```
#include <iostream>

void minMaxRep(int a[], size_t size,
int& mn, size_t& in, int&mx, size_t& ix)
{
        // ...
}

int main()
{
        using std::cout;
        int a[] = {2,3,4,2,7,4,7,2};
        size_t size = sizeof(a)/sizeof(*a);
        int mn, mx;
        size_t in, ix;
        minMaxRep(a,size,mn,in,mx,ix);
        cout << "Array: [ ";
        for (size_t i = 0; i < size; ++i)
                cout << a[i] << " ";
        cout << "]\n";
        cout << "Min = " << mn << " " << in << " times\n";
        cout << "Max = " << mx << " " << ix << " times\n";
}
```

powinien wydrukować:
Array: [2,3,4,2,7,4,7,2]
Min = 2 3 times
Max = 7 2 times
Uwaga: nie wolno stosować żadnych dodatkowych tablic ani kolekcji. Funkcja może przebiec w pętli po elementach tablicy tylko raz. Liczby w tablicy są zarówno ujemne jak i dodatnie.
Podpowiedź: Można wykorzystać kody z poprzednich zadań. Można założyć, że liczby w tablicy są z jakiegoś rozsądnego przedziału, np. $(-10^5, 10^5)$.

Zadanie 1 – Odwzorowanie logistyczne

Napisz program, który przeprowadzi symulację rozwoju populacji zwierząt wg modelu logistycznego. Model jest zdefiniowany poprzez ciąg rekurencyjny:

$$x_{n+1} = ax_n(1 - x_n), \tag{1}$$

gdzie $0 < a < 4$ jest parametrem wzrostu, a $0 \leqslant x_k \leqslant 1$ populacją unormowaną do jedynki ($x = 1$ odpowiada zużyciu całych zasobów i maksymalnej dopuszczalnej wielkości populacji).
Program powinien pobierać wartości a i x_0 jako parametry programu oraz sprawdzić ich zakresy. Jako trzeci parametr powinien pobierać liczbę iteracji ($\geqslant 0$). Do liczenia wyrazów ciągu użyj rekurencji. Wynikiem działania programu powinna być informacja o numerze iteracji i wielkości populacji (każda iteracja w nowej linii). W przypadku podania niepoprawnych argumentów program powinien kończyć pracę z wyświetleniem komunikatu o błędzie.

Przykład działania:
./zad1 0.7 0.2
Użyj: ./zad1 a x0 n

./zad1 0.7 0.2 10
ii x
0 0.2
1 0.112
2 0.0696192
3 0.0453407
4 0.0302994
5 0.020567
6 0.0141008
7 0.00973136
8 0.00674566
9 0.00469011
10 0.00326768

Zaobserwuj następujące zjawiska:

1. Dla $0 < a < 1$ populacja wyginie. Oznacza to, że ciąg będzie zbieżny do zera. Fachowo mówi się, że zero jest *atraktorem*.

2. Dla $1 < a < 2$ populacja zbiega do $\frac{a-1}{a}$ niezależnie od początkowej populacji.

3. Dla $3 < a < 3.44949$ populacja będzie oscylować wokół dwóch wartości zależnych od r. Rozszczepienie atraktora na dwa nazywa się *bifurkacją*.

4. Dla $3.44949 < a < 3.54409$ (*approx*) populacja będzie oscylować wokół czterech wartości zależnych od r. Kolejna bifurkacja. Dla większych wartości będzie 8, 16, 32 punkty oscylacji itd.

5. Dla $a > 3.56995$ większość[1] wartości a i $x0$ prowadzi do zjawiska *chaosu deterministycznego*

[1]Dla niektórych wartości parametrów istnieją tzw. okna stabilności, gdzie nie obserwuje się chaosu. Jeżeli trafiłeś na takie wartości to gratulacje! Zainteresowanych odsyłam np do Wikipedii. Jest też wiele fajnych książek o chaosie na różnym poziomie zaawansowania matematycznego. Sam chaos deterministyczny wywodzi się z badań nad prognozowaniem pogody, ale pojawia się w wielu dziedzinach fizyki, a jak pokazuje przykład również w innych naukach.

(atraktor nazywamy chaotycznym). Oznacza to, że chociaż kolejne wyrazy ciągu są ściśle zdefiniowane, to ciąg wydaje się być losowy, nie wystepują w nim oscylacje (można powiedzieć, że są oscylacje o okresie nieskończonym).

Wartość x_0 ma mniejsze znaczenie dla zachowania ciągu niż wartość a.

Uwaga:Nie wolno używać tablic ani kolekcji. Sprawdź ilość i poprawność argumentów w mainie. Funkcja licząca wyrazy ciągu ma być rekurencyjna (żadnych pętli!).
Podpowiedź: Przy sprawdzaniu unikaj podawania brzegowych wartości a. Zacznij od małego n, powiedzmy 30. Jeżeli nie widzisz zbieżności/oscylacji to zwiększ n kilka razy. **To zadanie jest podobne do zadania na rekurencję z poprzednich zajęć.**

Zadanie 2 – Sortowanie bąbelkowe

Korzystając z szablonów funkcji zaimplementuj algorytm sortowania bąbelkowego (wersja dla sortowania rosnącego):

Na starcie dostajesz nieposortowaną tablicę jednowymiarową o długości n.

1. Niech indeks ii=0.
2. Sprawdz czy element tablicy o indeksie ii jest większy od elementu ii+1.
   (a) Jeśli tak, to zamień obydwa elementy.
   (b) Jeśli nie, to nic nie rób.
3. Zwiększ ii o jeden.
   (a) Jeśli ii jest większe niż potrzeba[2] to idź do 1).
   (b) Jeśli ii nie jest większe niż potrzeba (patrz stopka) to idź do 2).

Wykorzystaj szablony funkcji, żeby napisać funkcję przyjmującą tablicę jednowymiarową dowolnego typu i długości i sortującą ją z użyciem powyższego algorytmu. Użyj poniższego szablonu i przeciąż funkcję **void wypisz(int tab[10])**, żeby działała również dla charów.

```
#include<iostream>

\\tutaj napisz sortowanie babelkowe
\\dzialajace dla tablicy dowolnego typu i rozmiaru

void wypisz(int tab[10])
{
        std::cout << "[";
        for(int ii=0; ii<10; ii++)
        {
                std::cout << tab[ii] <<" ";
```

[2]Po pierwszym przejściu, gdy ii==n, ostatni element w tablicy tj. element o indeksie n-1, jest już posortowany. Oznacza to, że w następnym przejściu masz jeden element do sprawdzania mniej i petla powinna wykonać jedną iterację mniej.

```
        }
        std::cout << "]";
        std::cout << std::endl;

}

\\ tutaj przeciaz funkcje wypisz(),
\\zeby dzialala dla tablicy 10 charow

int main()
{
        int tabi[10]={9, 4, 2, 1, 3, 6, 7, 2, 9, 0};
        char tabc[10]={'b','g','z','x','a','y','r','y','y','e'};

        std::cout << "Tablica intow przed:" << std::endl;
        wypisz(tabi);
        std::cout << "Tablica intow po:" << std::endl;
        sort(10, tabi);
        wypisz(tabi);

        std::cout << "Tablica charow przed:" << std::endl;
        wypisz(tabc);
        std::cout << "Tablica charow po:" << std::endl;
        sort(10, tabc);
        wypisz(tabc);

        return 0;
}
```

Output:
Tablica intów przed:
[9 4 2 1 3 6 7 2 9 0]
Tablica intów po:
[0 1 2 2 3 4 6 7 9 9]
Tablica charów przed:
[b g z x a y r y y e]
Tablica charów po:
[a b e g r x y y y z]

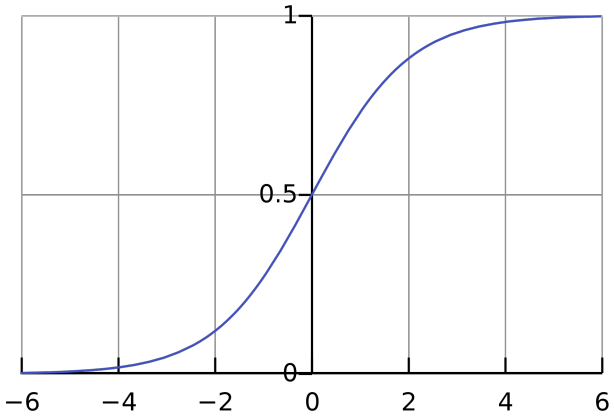
Uwaga: Nie twórz nowych tablic czy kolekcji, posortuj istniejącą tablicę. Nie używać gotowych algorytmów. Sortowanie nie powinno iterować po tablicy więcej razy niż to konieczne. Nie trzeba sprawdzać, czy wielkość tablicy podana do funkcji jest poprawna ani czy typ tablicy ma zdefiniowane operatory porównania ><.
Podpowiedź: Przydatne będzie napisanie osobnej funkcji do zamiany dwóch elementów tablicy.

Zadanie 3 – interpolacja

Namnażanie się prostych organizmów żywych, np. wirusów czy bakterii, symuluje się często z wykorzystaniem *funkcji logistycznej*, zdefiniowanej:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0))}}, \tag{2}$$

gdzie L jest maksymalną wartością funkcji (normalizacja), k jest pochyłością krzywej, a x_0 jest położeniem środka krzywej.



Rysunek 1: Krzywa logistyczna dla $L = 1, k = 1, x_0 = 0$

Napisz funkcję **lagrint** obliczającą wartość wielomianu Lagrange'a:

$$L_f(x) = \sum_{i=0}^{n} f(x_i) \prod_{j=0 \wedge j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{3}$$

Wielomian ten interpoluje dowolną funkcję f. Funkcja lagrint powinna przyjmować jako argumenty:

1. wartość x
2. tablicę wartości x_i w podanych punktach (trzeba wcześniej taką wygenerować!)
3. tablicę odpowiadających im wartości $f(x_i)$ (również wygenerowana wcześniej)
4. liczbę punktów n (rozmiar obu tablic).
5. może również przyjmować argumenty do podania dalej do funkcji logistycznej

Następnie przetestuj funkcję lagrint dla funkcji logistycznej z parametrami: $L = 100, x_0 = 14, k = 2$. wypisując na ekran wartości f w równoodległych punktach x_i wraz z wartościami interpolowanymi za pomocą funkcji lagrint (wartości powinny być jednakowe).

Przykład działania:
x y interp
8 0.000614417 0.000614417

8.12 0.000781076 0.000781076
8.24 0.000992941 0.000992941
8.36 0.00126227 0.00126227
8.48 0.00160466 0.00160466
8.6 0.00203991 0.00203991
. . .

Uwaga: Nie używać żadnych algorytmów z bilbiotek.
Podpowiedź: Funkcję eksponencjalną można znaleźć w bibliotece $< cmath >$, nazywa się *exp*. Będziesz potrzebował/a w sumie 5 tablic: znanych argumentów funkcji, znanych wartości funkcji, punktów do interpolowania, prawdziwych wartości funkcji dla interpolowanych argumentów, interpolowanych wartości funkcji dla interpolowanych argumentów. Dla chętnych (nie punktowane): Napisz funkcję **lagrint** z użyciem szablonów funkcji. Po skończeniu narysuj wykres z punktami znanymi w jednym kolorze i interpolowanymi w innym. Jeżeli nie znasz żadnych narzędzi do rysowania to możesz wyeksportować dane do formatu .csv. Aby to zrobić, ustaw wypisywanie interesujących Cię danych oddzielonych przecinkiem, nowa linia ma oznaczać nowy zestaw danych (nowy wiersz w tablicy). Następnie odpal program i przekieruj jego wynik do pliku: **./zad3 ¿ out.csv**

Format csv to Comma Separated Value, a więc wartości oddzielone przecinkiem. To format tekstowy do przechowywania tabel, każdy wiersz w pliku to wiersz w tabeli, wartości odzielone przecinkami to wartości kolejnych kolumn. Większość arkuszy kalkulacyjnych (np. MS Excell) potrafi wczytywać tego rodzaju pliki jako tabele. Możesz użyć arkusza, żeby narysować wykres.
Krzywa logistyczna z tego zadania pojawia się ostatnio często w mediach w związku z koronawirusem. Spróbuj pobawić się z różnymi parametrami i zasymulować różne scenariusze (nie podam Ci realistycznych parametrów, gdyż nikt ich obecnie nie zna, choć każdy by chciał).

Zadanie 4 – ułamki egipskie

Napisz funkcje, który wypisuje ułamki w postaci egipskiej, tj. jako sumę możliwie najmniejszej liczby ułamków o liczniku równym 1. Wykorzystaj poniższy szablon:

```
#include <iostream>

int main()
{
        int num[] = {3, 7, 0, 7, 123};
        int den[] = {4, 9, 8, 0, 43};
        for (int ii=0; ii<5; ii++)
        {
                int n = num[ii];
                int d = den[ii];
                std::cout << n << '/' << d << " -> ";
                egyptian(n, d);
                std::cout << std::endl;
        }
}
```

Output:
3/4 -¿ 1/2 + 1/4
7/9 -¿ 1/2 + 1/4 + 1/36
0/8 -¿ 0
7/0 -¿ Denominator is 0!!!
123/43 -¿ 2 +1/2 + 1/3 + 1/37 + 1/9546
Uwaga: Nie trzeba sprawdzać czy liczby są ujemne, zakładamy że są nieujemne.

Podpowiedź: Można to zrobić rekurencyjnie. Zastanów się jak można wykorzystać operator modulo, żeby znaleźć największy ułamek egipski nie mniejszy niż dany ułamek. Pamiętaj, żeby rozważyć różne przypadki liczb podawanych do funkcji, żeby rekurencja zawsze dobrze się kończyła.