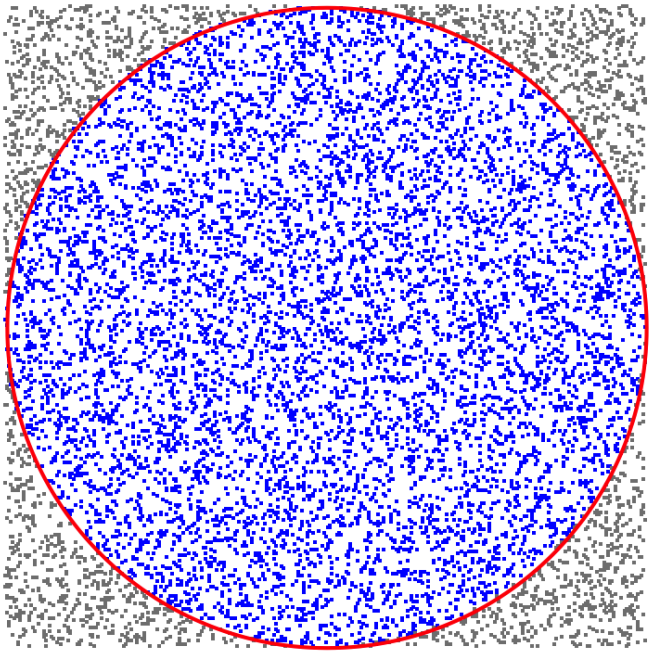
Zadanie 0 – całkowanie Monte Carlo

Chcemy znaleźć przybliżoną wartość liczby π. Rozważmy koło jednostkowe (o promieniu $r = 1$) wpisane w kwadrat o boku 2 (patrz obrazek poniżej). Zastanówmy się, jakie jest prawdopodobieństwo, że jeśli wylosujemy punkt na kwadracie to będzie on leżał wewnątrz koła? To prawdopodobieństwo wynosi tyle co stosunek pól koła do kwadratu, czyli $\pi/4$. Jeżeli oszacujemy empirycznie to prawdopodobieństwo to będziemy mogli wyznaczyć przybliżoną wartość liczby π.

Procedura jest następująca:

1. Wygeneruj losowo n punktów z rozkładu jednostajnego na kwadracie.
2. Policz ile punktów znajduje się wewnątrz koła, powiedzmy, że jest to k punktów.
3. Stosunek k/n jest estymatą prawdopodobieństwa. Ale wiemy, że prawdopodobieństwo wynosi ściśle $\pi/4$, więc mnożąc estymatę przez 4 dostajemy przybliżoną wartość liczby pi.



Rysunek 1: Źródło: Google Image

Napisz program, który pobiera jeden parametr w momencie uruchomienia będący liczbą punktów do wylosowania. Nazwijmy ten parametr N. Następnie wygeneruj 2N liczb pseudolosowych z rozkładu jednostajnego, w ten sposób dostaniesz N punktów na kwadracie, każdy o 2 współrzędnych. Policz ile punktów jest wewnątrz koła i podziel to przez liczbę wszystkich punktów N, żeby dostać estymatę prawdopodobieństwa. Na koniec wypisz wynik na standardowe wyjście. Proszę przetestować program dla co najmniej 3 różnych wartości N, każda innego rzędu wielkości.

Przykładowy output:
./zadanie0 10
pi = 2.8

./zadanie0 100
pi = 3.28

./zadanie0 1000
pi = 3.16

./zadanie0 10000
pi = 3.1556

./zadanie0 100000
pi = 3.14592

./zadanie0 1000000
pi = 3.14231

./zadanie0 10000000
pi = 3.14091

./zadanie0 100000000
pi = 3.14151

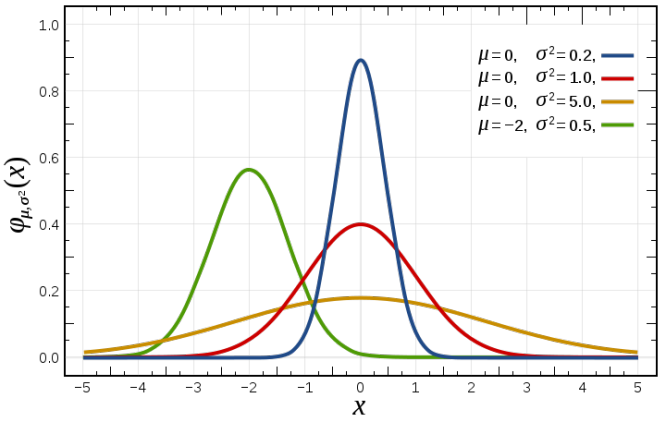
Uwaga: Nie używać tablic ani kolekcji. Nie używać std::cin. Sprawdzić, że argument N jest podany (jeśli nie to komunikat i koniec programu) oraz, że jest dodatni (jeśli nie to komunikat i koniec programu). Uważać z podawaniem dużych N, bo program może się przywiesić na chwilę.
Podpowiedź: Generowanie liczb pseudolosowych jest opisane krótko w teoretycznym minimum oraz w ćwiczeniach do tych zajęć. Zamiast porównywać pole koła jednostkowego i kwadratu o boku 2, można np. porównać ćwiartkę koła jednostkowego na kwadracie o boku 1 (stosunek pól jest taki sam). Jeżeli dostajesz jako wynik ciągle 0, to prawdopodobnie nie zrobiłeś konwersji z int na double

Zadanie 1 – Monte Carlo kontratakuje

Rozkład Gaussa dany jest wzorem:

$$\rho(\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{1}$$

gdzie μ jest średnią, a σ odchyleniem standardowym.



Rysunek 2: Źródło: `https://en.wikipedia.org/wiki/Gaussian_function#/media/File:Normal_Distribution_PDF.svg`

Jak policzyć pole pod krzywą Gaussa dla pewnego przedziału? Moglibyśmy postąpić podobnie jak w Zadaniu nr 0, to jest wygenerować z rozkładu normalnego punkty w odpowiednio dużym prostokącie, np. 100σ na 1. Następnie dla każdego x z zadanego przedziału policzylibyśmy wartość funkcji Gaussa i porównali z wylosowanym y. Jeżeli wylosowany y bybły mniejszy, oznaczałoby to, że punkt jest pod krzywą i zaakceptowalibyśmy go, w przeciwnym razie odrzucilibyśmy. Dzieląc przez liczbę wszystkich wygenerowanych punktów dostalibyśmy pole pod krzywą, a więc całkę oznaczoną.
Ale jest lepsza i prostsza metoda, która wymaga znacznie mniej operacji i nie narzuca ograniczenia się do jakiegoś przedziału (np. od -50σ do 50σ). Możemy wygenerować liczby pseudolosowe z rozkładu normalnego i policzyć ile z nich wpada do określonego przedziału całkowania. Im większa wartość ρ w jakimś przedziale tym więcej będzie tam liczb. Na koniec wystarczy podzielić przez liczbę wszystkich wygenerowanych liczb.
Napisz program realizujący następujące kroki:

1. Niech program przyjmuje 3 parametry w momencie uruchomienia: średnią μ, odchylenie standardowe σ i liczbę liczb do wylosowania N.

2. Użyj tych parametrów, żeby wygenerować N liczb z rokładu normalnego.

3. Policz, ile spośród tych liczb trafiło do przedziału $(-\sigma, \sigma)$ a ile do $(-2\sigma, 2\sigma)$.

4. Podziel te liczby przez N i wypisz **czytelny** komunikat.

5. Przetestuj porgram. Niezależnie od μ i σ, dla dostatecznie dużej liczby N powinieneś dostać odpowiednio 0.68 i 0.95 (około).

Przykładowy output:
./zadanie1 3.1224 5.223 0
Liczba punktow do wygenerowania musi być większa niż 0!

./zadanie1 3.1224 5.223 100
Całka na przedziale (-2.1006, 8.3454) = 0.63
Całka na przedziale (-7.3236, 13.5684) = 0.98

./zadanie1 -3.1224 10.223 10000
Całka na przedziale (-13.3454, 7.1006) = 0.6843
Całka na przedziale (-23.5684, 17.3236) = 0.9518

Uwaga: Nie używać tablic ani kolekcji. Nie używać std::cin. Sprawdzić liczbę i poprawność argumentów. Uważać z podawaniem dużych N, bo program może się przywiesić na chwilę.
Podpowiedź: Generowanie liczb pseudolosowych jest opisane krótko w teoretycznym minimum oraz w ćwiczeniach do tych zajęć. Jeżeli dostajesz jako wynik ciągle 0, to prawdopodobnie nie zrobiłeś konwersji z int na double

Zadanie 2 – Powrót MinMaxa

Zamień funkcję:

```
void print(const std::vector<int> &v) {
        std::cout << "[ ";
        for ( const auto &i : v) {
                std::cout << i << " ";
        }
        std::cout << "]" << std::endl;
}
```

w szablon funkcji, tak żeby print mogła też przyjmować wektory innych typów, takich jak double, char, itd. Zadziw się nad użytecznością szablonów.

Następnie napisz szablon funkcji:

```
template < ... >
void minMaxRep (...);
```

która pobiera przez const[1] referencję wektor i 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 wektorze. Przetestuj dla wektorów typu int, uint, char, double.

Rowiązania z przeciążaniem funkcji, tzn:

```
void minMaxRep(const std::vector<int>& v, double& mn, size_t& in,
int& mx, size_t& ix);
void minMaxRep(const std::vector<double>& v, double& mn, size_t& in,
double& mx, size_t& ix);
```

nie będą podlegały ocenie!

Przykład:

[1]Const referencja to taka, która zabrania zmiany wartości zmiennej przekazywanej. Const referencje deklarujemy poprzez użycie słowa const przed typem, np. **const int&**.

```cpp
#include <iostream>
#include <vector>

void print(const std::vector<int> &v)
{
        std::cout << "[ ";
        for (const auto &i : v)
        {
                std::cout << i << " ";
        }
        std::cout << "]" << std::endl;
}

template <...> void minMaxRep(...) { ... }

int main()
{
        std::cout << "Int: ";
        print(std::vector<int>{-1, 2, -2, 5, 3});
        std::cout << "Double: ";
        print(std::vector<double>{-1, 2, -2, 5, 3});
        // jezeli argument szablonu nie jest jasny
        // trzeba go podac
        // w tym przypadku nie jest to konieczne
        // jawny argumeny szablonu:
        print<double>(std::vector<double>{-1, 2, -2, 5, 3});
        std::cout << "Char: ";
        print(std::vector<char>{75, 119, 73, 107, 63});
        print(std::vector<char>{'K', 'W', 'I', 'K', '!'});
        std::cout << "Const char* : ";
        print(std::vector<const char *>{"One", "does", "not",
        "simply", "learn", "template", "meta-programing"});

        int ix=0, in=0;
        {
        int mn, mx;
        std::vector<int> v{-1, 1, 2, -1, 5, 2, 2, -1};
        print(v);
        minMaxRep(v, mn, in, mx, ix);
        std::cout << "Min = " << mn << " " << in << " times\n";
        std::cout << "Max = " << mx << " " << ix << " times\n";
        }

        {
        double mn, mx;
        // dla leniwych, decltype(i) jest tym samym typem co i
        std::vector<decltype(mn)> v{-1.23, 5.43, -2.33, 8.66, -1.23, 8.66};
```

```
        print(v);
        minMaxRep(v, mn, in, mx, ix);
        std::cout << "Min = " << mn << " " << in << " times\n";
        std::cout << "Max = " << mx << " " << ix << " times\n";
        }
}
```

Output:
Int: [-1 2 -2 5 3]
Double:[-1 2 -2 5 3]
[-1 2 -2 5 3]
Char: [K w I k ?]
[K W I K !]
Const char* : [One does not simply learn template meta-programing]
[-1 1 2 -1 5 2 2 -1]
Min = -1 3 times
Max = 5 1 times
[-1.23 5.43 -2.33 8.66 -1.23 8.66]
Min = -2.23 1 times
Max = 8.66 2 times

Zadanie 3 – szyfr Cezara

Szyfr Cezara to bardzo prosta metoda szyfrowania wiadomości. Szyfrowanie polega na zamianie każdej litery na inną, oddaloną od oryginalnej o pewną stałą odległość w alfabecie. Dla wszystkich liter szyfrowanej wiadomości stosuje się identyczne przesunięcie w tę samą stronę, np. jeżeli ograniczymy się do alfabetu angielskiego i przesuwamy do przodu o dwa, to zachodzą nastepujące podmiany:

1. $a \rightarrow c$
2. $b \rightarrow d$
3. $c \rightarrow e$
4. ...
5. $y \rightarrow a$
6. $z \rightarrow b$

Zauważmy, że alfabet jest zapętlony, zatem ostatnie litery przechodzą na pierwsze.
Napisz program, który przyjmuje dwa parametry wywołania: przesunięcie oraz napis do zaszyfrowania. Niech rezultatem działania programu będzie wypisanie zaszyfrowanej wiadomości na standardowe wyjście. Przesunięcie może być zarówno ujemne, zero, lub dodatnie.

Przykładowy output:
./zadanie3 -3 ”abc-def-ghi”
xyz-abc-def

./zadanie3 13 "ala ma kota, a jacek ma 2 psy: azora i puszka."
nyn zn xbgn, n wnprx zn 2 cfl: nmben v chfmxn.

Uwaga: Nie tworzyć nowych tablic ani kolekcji. Pamiętaj, żeby sprawdzić liczbę argumentów. Zazwyczaj szyfr Cezara ignoruje wielkość liter, dlatego możesz założyć, że wiadomość nie zawiera wielkich liter, tylko małe od a do z. Wiadomość może zawierać inne znaki, np.: kropkę, przecinek, myślnik, cyfry – te znaki powinny pozostać bez zmian!
Podpowiedź: Ciągi znaków są traktowane jakby były tablicami, dlatego możemy myśleć o argv jak o dwuwymiarowej tablicy: pierwszy wymiar określa argument będący ciągiem znaków, a drugi konkretne znaki. Np. **argv[1][3]** zwróci 3 znak pierwszego argumentu podanego podczas uruchamiania.
Dla chętnych: Zrób, żeby działało też dla wielkich liter, tzn. żeby wielkie litery przechodziły w wielkie, a małe litery w małe.

Zadanie 4 – Macierze

Użyj

```
std::vector< std::vector<double> >
```

do zaimplementowania macierzy i uzupełnij brakujące funkcje w poniższym szablonie:

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

using namespace std;

typedef vector<double> vec;
typedef vector<vec> matrix;

//przydatna funkcja sprawdzajaca czy macierz jest poprawnie zdefiniowana
//nalezy uzyc jej do sprawdzenia poprawnosci danych
//we wszystkich pozostalych funkcjach
bool wymiary(matrix M, int &n, int &m)
{
        n = M.size();
        if(n == 0)
                return false; //pierwszy wymiar jest zero
        m = M[0].size();
        for(int ii=0; ii<n; ii++)
        {
                if(M[ii].size() != m)
                {
                        return false; //rozna liczba kolumn w wierszach
                }
        }
```

```
        return true;
}

void skalar(matrix& M, double s)
{
        //mnozenie macierzy przez skalar
}

//const referencje zapobiegaja modyfikacji zmiennych
void odejmowanie(const matrix& A, const matrix& B, matrix& wynik)
{
        //odejmowanie 2 macierzy i zapisanie do
        //poczatkowo pustej macierzy wynik
}

void dodawanie(const matrix& A, const matrix& B, matrix& wynik)
{
        //dodawanie 2 macierzy i zapisanie do
        //poczatkowo pustej macierzy wynik
}

void mnozenie(const matrix& A, const matrix& B, matrix& wynik)
{
        //mnozenie dwoch macierzy
}

void potegowanie(matrix& M, unsigned n)
{
        //podniesienie do potegi naturalnej
        //i zapisanie wyniku do macierzy wynik
        //MA BYC REKURENCYJNIE!
}

void wypisz(const matrix& M)
{
        //wypisz macierz rzedami
}


int main()
{
        vec v1({1., 2., 3., 4.});
        vec v2({5., 6., 7., 8.});
        vec v3({9., 10., 11., 12.});
        matrix M({v1, v2, v3});
        matrix N = M;
        matrix O,P,Q, R;
```

```
        P = matrix( {vec({2.0, 0.0, 0.0}), vec({1.0, 0.0, 1.0}), vec({0.0,
        cout << "M:" << endl;
        wypisz(M);
        cout << "P:" << endl;
        wypisz(P);
        skalar(M, 0.5);
        cout << "M=0.5M:" << endl;
        wypisz(M);
        odejmowanie(M, N, O);
        cout << "O=M-N:" << endl;
        wypisz(O);
        dodawanie(M, N, Q);
        cout << "Q=M+N:" << endl;
        wypisz(Q);
        potegowanie(Q, 2);
        cout << "Q=Q*Q:  (nieudane)" << endl;
        wypisz(Q);
        mnozenie(O, P, R);
        cout << "R=O*P:" << endl;
        wypisz(R);
        potegowanie(R, 2);
        cout << "R=R*R:" << endl;
        wypisz(R);
        potegowanie(R, 3);
        cout << "R=R*R*R:" << endl;
        wypisz(R);
        return 0;
}
```

Output:
M:
1 2 3 4
5 6 7 8
9 10 11 12

P:
2 0 0
1 0 1
0 0 -1
0 0 0

M=0.5M:
0.5 1 1.5 2
2.5 3 3.5 4
4.5 5 5.5 6

O=M-N:

-0.5 -1 -1.5 -2
-2.5 -3 -3.5 -4
-4.5 -5 -5.5 -6

Q=M+N:
1.5 3 4.5 6
7.5 9 10.5 12
13.5 15 16.5 18

Niepoprawny wymiar macierzy!
Q=Q*Q: (nieudane)
1.5 3 4.5 6
7.5 9 10.5 12
13.5 15 16.5 18

R=O*P:
-2 0 0.5
-8 0 0.5
-14 0 0.5

R=R*R:
-3 0 -0.75
9 0 -3.75
21 0 -6.75

R=R*R*R:
173.812 0 -44.2969
707.062 0 -45.9844
1240.31 0 -47.6719

Uwaga: Nie używać operatora new. Nie używać żadnych wbudowanych funkcji do operowania na macierzach. Funkcja potegowanie ma być rekurencyjna. Każda z funkcji ma sprawdzać poprawność wymiarów macierzy. Jeżeli wymiary są niepoprawne to funkcja ma nic nie zrobić i wyświetlić komunikat (patrz output).
Podpowiedź: Pamiętaj, że to co tworzysz wewnątrz funkcji ginie z chwilą gdy program wychodzi z funkcji. Dlatego nie możesz utworzyć macierzy wewnątrz funkcji i przypisać jej do referencji (chyba, że użyłbyś new, ale tutaj jest zabronione).
Dla chętnych: Zrób to na szablonach funkcji tak, żeby macierz mogła mieć wartości typów, int, unsigned, long itp.