

Podstawy programowania
Semestr letni 2019/20
Materiały z laboratorium i zadania domowe

Przemysław Olbratowski

1 kwietnia 2020

Slajdy z wykładu są dostępne w serwisie UBI. Informacje organizacyjne oraz formularz do uploadu zadań domowych znajdują się na stronie info.wsisiz.edu.pl/~olbratow. Przy zadaniach domowych w nawiasach są podane terminy sprawdzeń.

Spis treści

1 Podstawy: 25 i 26 lutego

Wejście-wyjście, Zmienne, Warunki, Losowość, Logika	6
1.1 Przykłady laboratoryjne z działu Podstawy	6
1.1.1 Area: Pola figur płaskich	6
1.1.2 Signum: Gra w znaki	6
1.2 Zadania domowe z działu Podstawy (5, 12, 19 marca)	8
1.2.1 Temperature: Skale temperatury	8
1.2.2 Barometric: Wzór barometryczny	8
1.2.3 Dice: Rzut dwiema kostkami	8
1.2.4 Leap: Lata przestępne	8
1.2.5 Triangle: Warunek trójkąta	9
1.2.6 Shop: Godziny otwarcia sklepu - bitcoin	9
1.2.7 BMI: Indeks Masy Ciała	9
1.2.8 History: Test z historii	9
1.2.9 Holidays: Dni wolne od pracy	10
1.2.10 Countdown: Odliczanie	10
1.2.11 Calculator: Kalkulator pięciodziałaniowy - bitcoin	10

2 Pętle: 3 i 4 marca

Pętle, Format wydruku	12
2.1 Przykłady laboratoryjne z działu Pętle	12
2.1.1 Quiz: Test z tabliczki mnożenia	12
2.1.2 Factor: Rozkład na czynniki pierwsze	12
2.1.3 Multi: Drukowanie tabliczki mnożenia	13
2.2 Zadania domowe z działu Pętle (12, 19, 26 marca)	14
2.2.1 Bifactorial: Podwójna silnia	14
2.2.2 Fibonacci: Ciąg Fibonacciego - bitcoin	14
2.2.3 Section: Losowy podział liczby	14
2.2.4 Pi: Oszacowanie liczby pi metodą Monte-Carlo	14
2.2.5 Sum: Suma skończona	15
2.2.6 Nominals: Nominały	15
2.2.7 Collatz: Hipoteza Collatza	15
2.2.8 Precision: Dokładność maszynowa	15
2.2.9 Guess: Odgadywanie liczby	16
2.2.10 Stars: Wzorki z gwiazdek	16
2.2.11 Tri: Liczby trzycyfrowe - bitcoin	16
2.2.12 Polyhedron: Wielościany foremne	16
2.2.13 Trigonometry: Tabele trygonometryczne	17
2.2.14 Logic: Tabela prawdy	17
2.2.15 Clock: Zegar cyfrowy	17

3 EOF: 10 i 11 marca

Koniec pliku, Przekierowanie	19
3.1 Przykłady laboratoryjne z działu EOF	19
3.1.1 Means: Średnia arytmetyczna i geometryczna	19
3.1.2 XOR: Różnica symetryczna	19
3.2 Zadania domowe z działu EOF (19, 26 marca, 2 kwietnia)	20
3.2.1 Desert: Brakująca liczba	20
3.2.2 Statistics: Średnia i błąd	20
3.2.3 Minimum: Najmniejsza liczba - bitcoin	20
3.2.4 Pass: Zagadnienie mijania	20
3.2.5 EKG: Pulsometr	21
3.2.6 Neighbors: Najbliżsi sąsiedzi	21
3.2.7 Polygonal: Długość łamanej - bitcoin	21

4	Funkcje: 17 i 18 marca	
	Funkcje, Referencje	22
4.1	Przykłady laboratoryjne z działu Funkcje	22
4.1.1	Euclid: Algorytm Euklidesa	22
4.1.2	Quadratic: Równanie kwadratowe	22
4.1.3	Clock: Zegar analogowy	23
4.2	Zadania domowe z działu Funkcje (26 marca, 2 i 9 kwietnia)	24
4.2.1	Sign: Znak liczby całkowitej	24
4.2.2	Geometric: Ciąg geometryczny	24
4.2.3	Round: Zaokrąglanie z zadaną dokładnością	24
4.2.4	Implication: Implikacja	24
4.2.5	Lesser: Mniejsza z dwóch liczb	25
4.2.6	Factorial: Silnia	25
4.2.7	Bifactorial: Podwójna silnia	25
4.2.8	Digits: Cyfry dziesiętne	26
4.2.9	Uniform: Płaski rozkład prawdopodobieństwa	26
4.2.10	Prime: Czy liczba jest pierwsza - bitcoin	26
4.2.11	Power: Potęga całkowita	27
4.2.12	Coin: Rzut oszukaną monetą	27
4.2.13	Root: Pierwiastek kwadratowy	27
4.2.14	Pitagoras: Trójki pitagorejskie	28
4.2.15	RNG: Generator liczb losowych	28
4.2.16	DMS: Stopnie, minuty, sekundy	28
4.2.17	Fraction: Część całkowita i ułamkowa - bitcoin	29
4.2.18	Exchange: Zamiana wartości zmiennej	29
4.2.19	RNG: Liczby pseudolosowe	29
4.2.20	Choose: Wybór zmiennej	30
5	Wektory: 24 i 25 marca	
	Wektory	31
5.1	Przykłady laboratoryjne z działu Wektory	31
5.1.1	Bubble Sort: Sortowanie bąbelkowe	31
5.1.2	Select: Wybór elementów	31
5.1.3	Reverse: Odwracanie kolejności elementów	32
5.2	Zadania domowe z działu Wektory (2, 9, 23 kwietnia)	33
5.2.1	Accumulate: Akumulacja	33
5.2.2	Shuffle: Tasowanie elementów	33
5.2.3	Rotate: Cykliczne przesunięcie wektora	33
5.2.4	Heap Sort: Sortowanie kopcowe	34
5.2.5	Selection Sort: Sortowanie przez wybór - bitcoin	35
5.2.6	Find: Wyszukiwanie elementu - bitcoin	35
5.2.7	Count: Zliczanie elementów	35
5.2.8	Minimum: Element najmniejszy	36
5.2.9	Binary Search: Wyszukiwanie binarne	36
5.2.10	Count Sort: Sortowanie przez zliczanie	36
5.2.11	Back: Wydruk w odwrotnej kolejności	37
5.2.12	Maximum Sum: Przedział o największej sumie elementów	37
5.2.13	Longest Slope: Najdłuższy przedział niemalejący	37
5.2.14	Triangles: Liczba trójkątów	37
5.2.15	Relay: Przesiadka	38
5.2.16	Intersection: Część wspólna zbiorów	38

6	Argumenty: 31 marca i 1 kwietnia	
	Argumenty wywołania programu	39
6.1	Przykłady laboratoryjne z działu Argumenty	39
6.1.1	Eratostenes: Sito Eratostenesa	39
6.1.2	Beaufort: Skala Beauforta	40
6.2	Zadania domowe z działu Argumenty (9, 23, 30 kwietnia)	41
6.2.1	Sum: Suma argumentów - bitcoin	41
6.2.2	Permutation: Losowa permutacja	41
6.2.3	Permutations: Wszystkie permutacje	41
6.2.4	Pascal: Trójkąt Pascala	41
6.2.5	Days: Długości miesięcy	42
6.2.6	Nominals: Odliczanie kwoty w nominalach - bitcoin	42
6.2.7	Exact Sum: Przedział o zadanej sumie elementów	42
6.2.8	Horner: Schemat Hornera	42
7	Pliki: 7 i 8 kwietnia	
	Znaki, Pliki tekstowe	44
7.1	Przykłady laboratoryjne z działu Pliki	44
7.1.1	Capital: Zamiana liter na wielkie	44
7.1.2	WC: Zliczanie znaków, wyrazów i linii	44
7.2	Zadania domowe z działu Pliki (23, 30 kwietnia, 7 maja)	45
7.2.1	Char: Znak o zadanym kodzie	45
7.2.2	ASCII: Tabela kodów ASCII	45
7.2.3	Password: Losowe hasło	45
7.2.4	Is: Znaki białe, cyfry, małe i duże litery	45
7.2.5	Letters: Zliczanie liter	45
7.2.6	Phones: Numery telefonów	46
7.2.7	Cat: Łączenie plików tekstowych	46
7.2.8	Departures: Godziny odjazdów	46
7.2.9	Numerator: Numerowanie linii - bitcoin	47
7.2.10	Separator: Odstępy między wyrazami	47
7.2.11	Caesar: Szyfr Cezara - bitcoin	48
7.2.12	LF: Konwersja znaków końca linii	48
7.2.13	XOR: Szyfrowanie bitową różnicą symetryczną	48
7.2.14	Spacer: Zamiana tabulatorów na spacje	49
7.2.15	Tabulator: Zamiana spacji na tabulatory	49
7.2.16	Camel: Kamelizacja wyrazów	49
8	Łańcuchy: 21 i 22 kwietnia	
	Łańcuchy tekstowe, Strumienie łańcuchowe	50
9	Gry: 28 i 29 kwietnia	
	Prosta gra w trybie tekstowym	51
10	Sprawdzian: 5 i 6 maja	52
11	Iteratory: 12 i 13 maja	
	Iteratory wektora	53
12	Lambdy: 19 i 20 maja	
	Funkcje wyższego rzędu, Wyrażenia lambda	54
13	Algorytmy: 26 i 27 maja	
	Algorytmy biblioteki standardowej	55
14	Pojemniki: 2 i 3 czerwca	
	Pojemniki biblioteki standardowej	56

1 Podstawy: 25 i 26 lutego

Wejście-wyjście, Zmienne, Warunki, Losowość, Logika

1.1 Przykłady laboratoryjne z działu Podstawy

1.1.1 Area: Pola figur płaskich

Napisz program `area`, który wczytuje ze standardowego wejścia liczbę 1 lub 2. Jeżeli wpisano 1, program wczytuje promień koła i wypisuje na standardowe wyjście jego pole. Jeżeli wpisano 2, wczytuje długości trzech boków trójkąta i wypisuje jego pole. Jeżeli podano niepoprawne dane, program wypisuje komunikat o błędzie.

Przykładowe wykonanie

```
In: 2
In: 3 4 5
Out: 6
```

Rozwiązanie

```
#include <cmath>
#include <iostream>

constexpr double pi = 4. * std::atan(1.);

int main() {
    int figure;
    std::cin >> figure;
    if (figure == 1) {
        double r;
        std::cin >> r;
        if (r >= 0.) {
            double area = pi * std::pow(r, 2);
            std::cout << area << std::endl; }
        else {
            std::cout << "error" << std::endl; }}
    else if (figure == 2) {
        double a, b, c;
        std::cin >> a >> b >> c;
        if (a > 0. && b > 0. && c > 0. && a > b + c && b > c + a && c > a + b) {
            double s = (a + b + c) / 2;
            double area = std::sqrt(s * (s - a) * (s - b) * (s - c));
            std::cout << area << std::endl; }
        else {
            std::cout << "error" << std::endl; }}
    else {
        std::cout << "error" << std::endl; }}
```

1.1.2 Signum: Gra w znaki

Signum to gra dla dwóch osób, z których jedna przyjmuje rolę pozytywnej, a druga negatywnej. Każdy gracz zapisuje w ukryciu liczbę jeden lub minus jeden. Następnie osoby odkrywają swoje liczby i jeżeli ich iloczyn jest dodatni, to wygrywa gracz pozytywny, zaś w przeciwnym razie - negatywny. Napisz program `signum` grający z użytkownikiem. Po uruchomieniu program losowo wybiera swoją liczbę, ale jej nie wyświetla. Następnie wczytuje ze standardowego wejścia liczbę użytkownika. Potem wypisuje na standardowe wyjście swoją liczbę oraz `true` jeśli wygrywa gracz pozytywny albo `false` jeśli negatywny.

Przykładowe wykonanie

In: -1
Out: -1
Out: true

Rozwiązanie

```
#include <cstdlib>
#include <ctime>
#include <iostream>

int main() {
    std::srand(std::time(nullptr));
    int computer = 2 * (std::rand() % 2) - 1, player;
    std::cin >> player;
    std::cout << computer << std::endl;
    std::cout << std::boolalpha << (0 < computer * player) << std::endl; }
```

1.2 Zadania domowe z działu Podstawy (5, 12, 19 marca)

1.2.1 Temperature: Skale temperatury

Temperatura w stopniach Celsjusza jest o 273.15 niższa niż w Kelwinach. Temperatura w stopniach Rankina jest dziewięć piątych razy większa niż w Kelwinach. Temperatura w stopniach Réaumura to temperatura w stopniach Celsjusza pomnożona przez cztery piąte. Napisz program `temperature`, który wczytuje ze standardowego wejścia temperaturę w Kelwinach i wypisuje na standardowe wyjście odpowiadające jej temperatury w stopniach Celsjusza, Rankina i Réaumura. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

```
In: 200
Out: -73.15 360 -58.52
```

1.2.2 Barometric: Wzór barometryczny

Wyrażoną w metrach wysokość h nad poziomem morza można obliczyć ze wzoru barometrycznego

$$h = -\frac{RT}{\mu g} \log\left(\frac{p}{p_0}\right)$$

gdzie $R = 8.3144598$, $\mu = 0.0289644$, $g = 9.80665$, $p_0 = 1013.25$, zaś p oraz T są odpowiednio ciśnieniem atmosferycznym w hektopaskalach i temperaturą powietrza w kelwinach. Napisz program `barometric`, który wczytuje ze standardowego wejścia ciśnienie w hektopaskalach oraz temperaturę w Kelwinach i wypisuje na standardowe wyjście wysokość w metrach. Program załącza tylko pliki nagłówkowe `cmath` i `iostream`.

Przykładowe wykonanie

```
In: 955 290
Out: 502.596
```

1.2.3 Dice: Rzut dwiema kostkami

Napisz program `dice` symulujący rzut dwiema sześciennymi kostkami do gry. Program wypisuje na standardowe wyjście liczby oczek na obu kostkach oraz ich sumę. Przy każdym uruchomieniu wyniki powinny być inne. Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime` i `iostream`.

Przykładowe wykonanie

```
Out: 5 2 7
```

1.2.4 Leap: Lata przestępne

Według kalendarza gregoriańskiego przestępne są lata podzielne przez 4 z wyjątkiem lat podzielnych przez 100 ale niepodzielnych przez 400. Napisz program `leap`, który wczytuje ze standardowego wejścia rok i wypisuje na standardowe wyjście `true` jeśli jest on przestępny albo `false` w przeciwnym razie. Program korzysta tylko z pliku nagłówkowego `iostream`.

Przykładowe wykonanie

```
In: 2000
Out: true
```

Uwaga Spróbuj napisać program bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

1.2.5 Triangle: Warunek trójkąta

Napisz program `triangle`, który wczytuje ze standardowego wejścia długości trzech odcinków i wypisuje na standardowe wyjście `true` jeśli można z nich zbudować trójkąt, albo `false` w przeciwnym razie. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 2 7 4
Out: false

Uwaga Z trzech odcinków można zbudować trójkąt jeśli długość każdego z nich jest mniejsza od sumy długości dwóch pozostałych. Spróbuj napisać program bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

1.2.6 Shop: Godziny otwarcia sklepu - bitcoin

Pewien sklep jest czynny od 10:30 włącznie do 18:30 wyłącznie. Napisz program `shop`, który wczytuje ze standardowego wejścia godzinę i minuty, na przykład 11 45 dla 11:45, i wypisuje na standardowe wyjście `true`, jeśli sklep jest wtedy otwarty, albo `false` jeśli jest zamknięty. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 11 45
Out: true

Uwaga Spróbuj napisać program bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

1.2.7 BMI: Indeks Masy Ciała

Indeks Masy Ciała BMI, z angielskiego *Body-Mass Index*, to masa wyrażona w kilogramach dzielona przez kwadrat wzrostu wyrażonego w metrach. Wagę człowieka można ocenić według następującej tabelki:

BMI	Waga
Poniżej 18.5	Niedowaga
18.5 - 25	Norma
25 - 30	Nadwaga
Powyżej 30	Otyłość

Napisz program `bmi`, który wczytuje ze standardowego wejścia masę w kilogramach oraz wzrost w centymetrach i wypisuje na standardowe wyjście komunikat `underweight`, `normal`, `overweight`, lub `obese`. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 80 178
Out: overweight

1.2.8 History: Test z historii

Napisz program `history` przeprowadzający test z historii świata. Program wypisuje na standardowe wyjście trzy pytania o rok jakiegoś wydarzenia i po każdym wczytuje ze standardowego wejścia odpowiedź. Po poprawnej odpowiedzi wypisuje `true`, a po niepoprawnej `false`. Na końcu wypisuje liczbę poprawnych odpowiedzi. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

```
Out: foundation of rome   In: -753
Out: true
Out: discovery of america In: 1975
Out: false
Out: first airplane flight In: 1903
Out: true
Out: 2
```

1.2.9 Holidays: Dni wolne od pracy

W Polsce obowiązują następujące święta stałe wolne od pracy:

1 stycznia	Nowy Rok
6 stycznia	Trzech Króli
1 maja	Święto Pracy
3 Maja	Święto Konstytucji 3 Maja
15 sierpnia	Święto Wojska Polskiego
1 listopada	Wszystkich Świętych
11 listopada	Święto Niepodległości
25 grudnia	Boże Narodzenie
26 grudnia	Boże Narodzenie

Napisz program `holidays`, który wczytuje ze standardowego wejścia numer miesiąca i dnia, na przykład 8 i 15 dla piętnastego sierpnia. Jeżeli tego dnia wypada święto stałe wolne od pracy, program wypisuje na standardowe wyjście angielską nazwę święta. W przeciwnym razie wypisuje słowa `ordinary day`. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

```
In: 8 15
Out: armed forces day
```

Uwaga Użyj instrukcji wyboru.

1.2.10 Countdown: Odliczanie

Napisz program `countdown` przeprowadzający odliczanie przed startem. Program wczytuje ze standardowego wejścia liczbę całkowitą. Jeżeli należy ona do przedziału od zera do dziesięciu włącznie, to wypisuje na standardowe wyjście angielskie nazwy liczb od podanej w dół do zera, a na końcu słowo `start`. W przeciwnym razie wypisuje słowo `stop`. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

```
In: 7
Out: seven six five four three two one zero start
```

Uwaga Użyj instrukcji wyboru. Spróbuj napisać program tak, aby nazwa każdej liczby wystąpiła w nim tylko raz.

1.2.11 Calculator: Kalkulator pięciodziałaniowy - bitcoin

Napisz program `calculator` wykonujący cztery podstawowe działania arytmetyczne oraz wyciągający pierwiastek kwadratowy. Po uruchomieniu program wczytuje ze standardowego wejścia liczbę 1, 2, 3, 4, lub 5, co odpowiada dodawaniu, odejmowaniu, mnożeniu, dzieleniu i pierwiastkowaniu. Następnie wczytuje argumenty wybranego działania i wypisuje na standardowe wyjście jego wynik. Program załącza tylko pliki nagłówkowe `cmath` i `iostream`.

Przykładowe wykonanie

In: 2

In: 7.5 5.2

Out: 2.3

Uwaga Użyj instrukcji wyboru.

2 Pętle: 3 i 4 marca

Pętle, Format wydruku

2.1 Przykłady laboratoryjne z działu Pętle

2.1.1 Quiz: Test z tabliczki mnożenia

Napisz program quiz, który wypisuje na standardowe wyjście dziesięć losowych pytań z tabliczki mnożenia liczb od jednego do dziesięciu i po każdym pytaniu wczytuje ze standardowego wejścia odpowiedź. Każde pytanie zadaje aż do uzyskania poprawnej odpowiedzi.

Przykładowe wykonanie

```
Out: 2 8 In: 16
Out: 5 1 In: 3
Out: 5 1 In: 5
Out: 9 5 In: 45
Out: 9 9 In: 81
Out: 3 5 In: 15
Out: 6 6 In: 36
Out: 2 8 In: 16
Out: 2 2 In: 4
Out: 6 3 In: 18
Out: 8 7 In: 56
```

Rozwiązanie

```
#include <cstdlib>
#include <ctime>
#include <iostream>

int main() {
    std::srand(std::time(nullptr));
    for (int index = 0; index < 10; ++index) {
        int a = 1 + std::rand() % 10;
        int b = 1 + std::rand() % 10;
        int answer;
        do {
            std::cout << a << " " << b << " ";
            std::cin >> answer; }
        while (answer != a * b); }}
```

2.1.2 Factor: Rozkład na czynniki pierwsze

Napisz program factor, który wczytuje ze standardowego wejścia liczbę całkowitą większą od jednego i wypisuje na standardowe wyjście jej rozkład na czynniki pierwsze.

Przykładowe wykonanie

```
In: 517
Out: 11 47
```

Uwaga Liczbę naturalną można rozłożyć na czynniki pierwsze w następujący sposób. Sprawdzamy, czy liczba jest podzielna przez 2. Jeżeli tak, to 2 dopisujemy do rozkładu, a samą liczbę dzielimy przez 2. Czynność tę powtarzamy aż liczba przestanie być podzielna przez 2. Następnie w taki sam sposób badamy podzielność przez 3, 4 i tak dalej, aż rozważana liczba stanie się równa 1.

Rozwiązanie

```
#include <iostream>

int main() {
    int number;
    std::cin >> number;
    for (int divisor = 2; number != 1; ++divisor) {
        while (number % divisor == 0) {
            std::cout << divisor << " ";
            number /= divisor; }
        std::cout << std::endl; }
}
```

2.1.3 Multi: Drukowanie tabliczki mnożenia

Napisz program `multi`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n i drukuje na standardowe wyjście tabliczkę mnożenia liczb od 1 do n sformatowaną jak w poniższym przykładzie. Szerokości wszystkich kolumn są jednakowe i dobierane automatycznie na podstawie wartości n .

Przykładowe wykonanie

```
In: 10
Out:  1  2  3  4  5  6  7  8  9 10
Out:  2  4  6  8 10 12 14 16 18 20
Out:  3  6  9 12 15 18 21 24 27 30
Out:  4  8 12 16 20 24 28 32 36 40
Out:  5 10 15 20 25 30 35 40 45 50
Out:  6 12 18 24 30 36 42 48 54 60
Out:  7 14 21 28 35 42 49 56 63 70
Out:  8 16 24 32 40 48 56 64 72 80
Out:  9 18 27 36 45 54 63 72 81 90
Out: 10 20 30 40 50 60 70 80 90 100
```

Rozwiązanie

```
#include <iomanip>
#include <iostream>

int main () {
    int number;
    std::cin >> number;
    int width = 0;
    for (int maximum = number * number; maximum; maximum /= 10) {
        ++width; }
    for (int row = 1; row <= number; ++row) {
        for (int column = 1; column <= number; ++column) {
            std::cout << " " << std::setw(width) << row * column; }
        std::cout << std::endl; }
}
```

2.2 Zadania domowe z działu Pętle (12, 19, 26 marca)

2.2.1 Bifactorial: Podwójna silnia

Podwójna silnia nieujemnej liczby całkowitej n , oznaczana jako $n!!$, to iloczyn wszystkich dodatnich liczb całkowitych mniejszych lub równych n , o takiej samej parzystości jak n , przy czym $0!! = 1$. Napisz program `bifactorial`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą i wypisuje na standardowe wyjście jej podwójną silnię. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 6
Out: 48

2.2.2 Fibonaccii: Ciąg Fibonacciego - bitcoin

Ciąg Fibonacciego zaczyna się od wyrazów 0 i 1, a każdy następny jest sumą dwóch poprzednich. Napisz program `fibonacci`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i drukuje na standardowe wyjście n pierwszych wyrazów ciągu Fibonacciego. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 10
Out: 0 1 1 2 3 5 8 13 21 34

2.2.3 Section: Losowy podział liczby

Napisz program `section`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n , po czym wypisuje na standardowe wyjście n losowych liczb nieujemnych, które dają w sumie jeden. Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime` oraz `iostream`.

Przykładowe wykonanie

In: 3
Out: 0.54095 0.345354 0.113696

2.2.4 Pi: Oszacowanie liczby pi metodą Monte-Carlo

Przybliżoną wartość liczby π można wyznaczyć następująco. Rozważmy kwadrat o boku 1 oraz zawartą w nim ćwiartkę koła o promieniu 1. Pola kwadratu i ćwiartki koła wynoszą odpowiednio $A_s = 1$ i $A_c = \pi/4$. Wybierzmy losowo dużo punktów tak, aby równomiernie wypełniały cały kwadrat. Stosunek liczby punktów wewnątrz ćwiartki koła do liczby wszystkich punktów w kwadracie jest w przybliżeniu równy stosunkowi pól tych figur, $N_c/N_s \approx A_c/A_s$. Dostajemy stąd

$$\pi \approx 4N_c/N_s$$

Napisz program `pi`, który wczytuje ze standardowego wejścia liczbę punktów do wylosowania i wypisuje na standardowe wyjście otrzymane przybliżenie liczby π . Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime` i `iostream`.

Przykładowe wykonanie

In: 10000
Out: 3.144

2.2.5 Sum: Suma skończona

Napisz program `sum`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą n i wypisuje na standardowe wyjście sumę

$$4 \sum_{k=1}^n \frac{(-1)^{k+1}}{2k-1}$$

Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 1000
Out: 3.14059

2.2.6 Nominals: Nominały

Napisz program `nominals`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą i wypisuje na standardowe wyjście wszystkie mniejsze od niej liczby postaci 1, 2, 5, 10, 20, 50,... Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 1000
Out: 1 2 5 10 20 50 100 200 500

2.2.7 Collatz: Hipoteza Collatza

Jeżeli liczba jest parzysta, to dzielimy ją przez dwa, a jeśli jest nieparzysta, to mnożymy przez trzy i dodajemy jeden. Z otrzymanym wynikiem postępujemy tak samo. Hipoteza Collatza mówi, że w końcu zawsze otrzymamy jeden. Do tej pory nie wiadomo, czy to prawda. Napisz program `collatz`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą i wypisuje na standardowe wyjście kolejne liczby otrzymane w opisany sposób, aż do jedynki. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 9
Out: 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

2.2.8 Precision: Dokładność maszynowa

Dokładnością maszynową nazywamy najmniejszą potęgę dwójki, która dodana do jedynki daje wynik numerycznie różny od jednego. Pojęcie to odnosi się tylko do liczb zmiennoprzecinkowych i chodzi tu o potęgę z wykładnikiem ujemnym. Dokładność maszynowa jest różna dla zmiennych różnego typu. Napisz program `precision`, który doświadczalnie wyznacza i wypisuje na standardowe wyjście dokładności maszynowe typów `float`, `double`, oraz `long double`. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

Out: 5.960464e-08 1.110223e-16 5.421011e-20

Uwaga Dokładność maszynową można wyznaczyć doświadczalnie w następujący sposób. Zaczynamy od zerowej potęgi dwójki równej jeden, a następnie rozpatrujemy coraz mniejsze potęgi. Każdą kolejną potęgę dodajemy do jedynki i sprawdzamy, czy wynik jest numerycznie równy jedności. Trzeba jednak pamiętać, że procesor nie wykonuje operacji zmiennoprzecinkowych bezpośrednio na zmiennych, lecz w swoich wewnętrznych rejestrach. Może się więc zdarzyć, że zamiast wyznaczyć dokładność żadanego typu znajdziemy dokładność rejestrów procesora, która jest na ogół większa. Aby temu zapobiec, wynik każdego dodawania należy najpierw wpisać do zmiennej żadanego typu i dopiero tę zmienną porównać z jedynką. Każdą kolejną potęgę dwójki najprościej wyznaczyć dzieląc poprzednią potęgę przez dwa.

2.2.9 Guess: Odgadywanie liczby

Napisz program `guess` odgadyujący pomyślaną przez użytkownika liczbę. Przed uruchomieniem użytkownik wybiera losowo liczbę całkowitą z przedziału od zera włącznie do stu wyłącznie. Po uruchomieniu program wypisuje na standardowe wyjście pewną liczbę z tego przedziału i wczytuje ze standardowego wejścia odpowiedź użytkownika równą `-1`, `0`, lub `+1`. Odpowiedzi te oznaczają odpowiednio, że pomyślana liczba jest mniejsza, równa, lub większa od liczby wyświetlonej przez program. Program kontynuuje zgadywanie aż do odgadnięcia właściwej liczby. Program powinien odgadnąć tę liczbę w nie więcej niż siedmiu próbach. Program załącza tylko plik nagłówkowy `iostream`.

Uwaga Zastosuj tak zwane wyszukiwanie binarne. Pomyślana przez użytkownika liczba leży w przedziale od 0 do 100. Na początku program wyświetla liczbę leżącą w połowie tego przedziału, czyli 50. Jeżeli użytkownik odpowie `-1`, to jego liczba leży w przedziale od 0 do 50, i w następnej próbie program wyświetla liczbę w połowie tego przedziału, czyli 25. Jeżeli natomiast użytkownik odpowie `+1`, to jego liczba leży w przedziale od 50 do 100, i w następnej próbie program wyświetla liczbę w połowie tego przedziału, czyli 75. W ten sposób w każdej próbie program dwukrotnie zawęża przedział, w którym może leżeć pomyślana przez użytkownika liczba. Zapewnia to zgadnięcie liczby w nie więcej niż siedmiu próbach.

2.2.10 Stars: Wzorki z gwiazdek

Napisz program `stars`, który wczytuje ze standardowego wejścia nieujemną liczbę całkowitą n i wypisuje na standardowe wyjście wybrany wzorek z gwiazdek złożony z $2n + 1$ wierszy i kolumn. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wzorki

```

*      *      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
***** *      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
*      *      *      *      *      *      *
```

2.2.11 Tri: Liczby trzycyfrowe - bitcoin

Napisz program `tri` wypisujący na standardowe wyjście w kolejności rosnącej wszystkie liczby trzycyfrowe, których cyfra setek to 1, 2, 5, 6, 7, lub 9, cyfra dziesiątek jest potęgą dwójki, cyfra jedności jest parzysta, a suma cyfr dzieli się przez 7. Program wypisuje te liczby przy użyciu odpowiednich pętli i załącza tylko plik nagłówkowy `iostream`.

Wykonanie

Out: 124 142 214 248 284 518 520 588 610 626 644 680 716 786 914 948 984

2.2.12 Polyhedron: Wielościany foremne

W każdym wielościanie foremnym całkowita liczba krawędzi, e_t , liczba krawędzi jednej ściany, e_f , oraz liczba krawędzi wychodzących z jednego wierzchołka, e_v , spełniają zależność

$$2e_t(e_f + e_v) = (2 + e_t)e_f e_v$$

Napisz program `polyhedron`, który wypisuje na standardowe wyjście wszystkie możliwe kombinacje całkowitej liczby ścian, $f_t = 2e_t/e_f$, i liczby krawędzi jednej ściany, e_f . Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

```
Out: 4 3
Out: 6 4
Out: 8 3
Out: 12 5
Out: 20 3
```

2.2.13 Trigonometry: Tabele trygonometryczne

Napisz program `trigonometry`, który drukuje na standardowe wyjście w kolejnych kolumnach kąt w stopniach od 0° do 180° z krokiem 10° oraz sinus i kosinus tego kąta w formacie jak poniżej. Program załącza tylko pliki nagłówkowe `cmath`, `iomanip`, oraz `iostream`.

Wykonanie

```
Out:  0  0.000  1.000
Out: 10  0.174  0.985
Out: 20  0.342  0.940
Out: 30  0.500  0.866
Out: 40  0.643  0.766
Out: 50  0.766  0.643
Out: 60  0.866  0.500
Out: 70  0.940  0.342
Out: 80  0.985  0.174
Out: 90  1.000  0.000
Out: 100 0.985 -0.174
Out: 110 0.940 -0.342
Out: 120 0.866 -0.500
Out: 130 0.766 -0.643
Out: 140 0.643 -0.766
Out: 150 0.500 -0.866
Out: 160 0.342 -0.940
Out: 170 0.174 -0.985
Out: 180 0.000 -1.000
```

2.2.14 Logic: Tabela prawdy

Napisz program `logic` wypisujący na standardowe wyjście tabelę prawdy operacji $(p \vee q) \wedge r$, jak pokazano poniżej. Kolejne kolumny zawierają odpowiednio wartości p , q , r , oraz $(p \vee q) \wedge r$. Program wypisuje tabelę przy pomocy odpowiednich pętli i załącza tylko pliki nagłówkowe `iomanip` oraz `iostream`.

Wykonanie

```
Out: true  true  true  true
Out: true  true false false
Out: true false  true  true
Out: true false false false
Out: false true  true  true
Out: false true false false
Out: false false  true false
Out: false false false false
```

2.2.15 Clock: Zegar cyfrowy

Napisz program `clock`, który w przybliżeniu co sekundę wypisuje na standardowe wyjście godzinę jak w poniższym przykładzie, poczynając od północy. Program załącza tylko pliki nagłówkowe `iomanip` i `iostream`.

Początek przykładowego wykonania

Out: 00:00:00

Out: 00:00:01

Out: 00:00:02

Wskazówka Do odczekania sekundy użyj pustej pętli powtarzającej się dostatecznie dużo razy.

3 EOF: 10 i 11 marca

Koniec pliku, Przekierowanie

3.1 Przykłady laboratoryjne z działu EOF

3.1.1 Means: Średnia arytmetyczna i geometryczna

Średnie arytmetyczna i geometryczna nieujemnych liczb rzeczywistych x_1, x_2, \dots, x_n , dane są wzorami:

$$A = \frac{x_1 + x_2 + \dots + x_n}{n} \quad G = \sqrt[n]{x_1 \times x_2 \times \dots \times x_n}$$

Napisz program `means`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście ich średnią arytmetyczną oraz geometryczną.

Przykładowe wykonanie

In: 2.5 0.3 1.7 0.25 3.75
Out: 1.7 1.03633

Rozwiązanie

```
#include <cmath>
#include <iostream>

int main() {
    double sum = 0., product = 1.;
    int count = 0;
    for (double value; std::cin >> value;) {
        sum += value;
        product *= value;
        ++count; }
    std::cout << sum / count << std::endl;
    std::cout << std::pow(product, 1. / count) << std::endl; }
```

3.1.2 XOR: Różnica symetryczna

Napisz program `xor`, który czyta ze standardowego wejścia wartości logiczne do napotkania końca pliku i wypisuje na standardowe wyjście ich różnicę symetryczną `xor`. Zarówno na wejściu, jak i na wyjściu, wartości logiczne są zapisane jako słowa `true` albo `false`.

Przykładowe wykonanie

In: false true true false true
Out: true

Rozwiązanie

```
#include <iostream>

int main() {
    bool result = false;
    std::cin >> std::boolalpha;
    for (bool value; std::cin >> value;) {
        result = (result != value); }
    std::cout << std::boolalpha << result << std::endl; }
```

3.2 Zadania domowe z działu EOF (19, 26 marca, 2 kwietnia)

3.2.1 Deserter: Brakująca liczba

Spośród liczb całkowitych od 1 do n włącznie usuwamy losowo jedną, a resztę zapisujemy w przypadkowej kolejności. Napisz program `deserter`, który czyta zapisane liczby ze standardowego wejścia do napotkania końca pliku i wypisuje na standardowe wyjście tę brakującą. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 2 5 1 4
Out: 3

3.2.2 Statistics: Średnia i błąd

Średnia arytmetyczna x liczb rzeczywistych x_1, \dots, x_n oraz jej błąd σ dane są wzorami

$$x = \frac{1}{n} \sum_{k=1}^n x_k \quad \sigma = \sqrt{\frac{1}{n(n-1)} \sum_{k=1}^n (x_k - x)^2}$$

Napisz program `statistics`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście ich średnią arytmetyczną oraz jej błąd. Program załącza tylko pliki nagłówkowe `cmath` i `iostream`.

Przykładowe wykonanie

In: 1 2 3 4 5 6 7 8 9 10
Out: 5.5 0.957427

3.2.3 Minimum: Najmniejsza liczba - bitcoin

Napisz program `minimum`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście najmniejszą z nich. Jeżeli nie wprowadzono żadnej liczby, program nic nie wypisuje. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 23.5 7.16 2 -1.3 -7 0.13 -1.3 28 -7 23.5
Out: -7

3.2.4 Pass: Zagadnienie mijania

Zapisujemy w jednej linii losowo zera i jedynki oddzielając je spacjami. Wyobraźmy sobie, że w pewnej chwili wszystkie zera przesuwają się na początek linii, a wszystkie jedynki na koniec. Napisz program `pass`, który czyta ze standardowego wejścia zapisane liczby do napotkania końca pliku i wypisuje na standardowe wyjście, ile razy zera miną się z jedynkami. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 0 1 1 0 1 0 1
Out: 5

3.2.5 EKG: Pulsometr

Plik tekstowy `ekg.txt` zawiera sygnał EKG próbkowany z częstotliwością 128Hz i przyjmujący wartości mniej więcej od 0 do 6000mV. Napisz program `ekg`, który wczytuje ten plik ze standardowego wejścia i wypisuje na standardowe wyjście średnią liczbę uderzeń serca na minutę. Program załącza tylko plik nagłówkowy `iostream`.

Wskazówka Obejrzyj sygnał w dowolnym programie do sporządzania wykresów. Widać tam wyraźne piki o wysokości około 6000mV. Ich położenia można znaleźć sprawdzając, kiedy sygnał przekracza progową wartość równą przykładowo 3000mV. Dzieje się tak gdy z dwóch kolejnych wartości sygnału pierwsza jest mniejsza, a druga większa od 3000mV. Analizując sygnał wystarczy więc pamiętać dwie ostatnio wczytane wartości. Do wyznaczenia tętna wystarczy znaleźć położenia pierwszego i ostatniego piku oraz liczbę pików między nimi.

3.2.6 Neighbors: Najbliżsi sąsiedzi

Napisz program `neighbors`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje na standardowe wyjście parę tych kolejnych liczb, które się najmniej różnią. Program załącza tylko pliki nagłówkowe `cmath` i `iostream`.

Przykładowe wykonanie

```
In: 0.3 1.5 8.9 8.7 1.6 0.2
Out: 8.9 8.7
```

3.2.7 Polygonal: Długość łamanej - bitcoin

Napisz program `polygonal` obliczający długość łamanej. Program czyta ze standardowego wejścia pary liczb rzeczywistych określające kartezjańskie współrzędne punktów na płaszczyźnie. Czyta je do napotkania końca pliku i wypisuje na standardowe wyjście długość łamanej otwartej łączącej te punkty, od pierwszego do ostatniego. Jeżeli podano tylko jedną parę liczb, program wypisuje zero. Jeżeli nie podano żadnej pary, program nic nie wypisuje. Program załącza tylko pliki nagłówkowe `cmath` i `iostream`.

Przykładowe wykonanie

```
In: -0.3 7.5
In: 9.5 -3.7
In: 5.0 0.4
Out: 20.9699
```

4 Funkcje: 17 i 18 marca

Funkcje, Referencje

4.1 Przykłady laboratoryjne z działu Funkcje

4.1.1 Euclid: Algorytm Euklidesa

Największy wspólny dzielnik dwóch dodatnich liczb całkowitych można znaleźć następującą metodą przypisywaną Euklidesowi. Pierwszą liczbę zastępujemy resztą z dzielenia przez drugą i zamieniamy liczby miejscami. Czynność tę powtarzamy aż druga liczba stanie się równa zero. Wtedy pierwsza jest poszukiwanym największym wspólnym dzielnikiem. Napisz funkcję `euclid`, która przyjmuje dwie dodatnie liczby całkowite i zwraca ich największy wspólny dzielnik. Korzystając z tej funkcji napisz program, który czyta ze standardowego wejścia dodatnie liczby całkowite do napotkania końca pliku i wypisuje na standardowe wyjście ich największy wspólny dzielnik.

Przykładowe wykonanie

```
In: 12 42 18 66 30
Out: 6
```

Rozwiązanie

```
#include <iostream>

int euclid(int m, int n) {
    while (n) {
        int k = m % n;
        m = n;
        n = k; }
    return m; }

int main() {
    int divisor = 0;
    for (int number; std::cin >> number;) {
        divisor = euclid(divisor, number); }
    std::cout << divisor << std::endl; }
```

4.1.2 Quadratic: Równanie kwadratowe

Napisz funkcję `quadratic` rozwiązującą równanie kwadratowe $ax^2 + bx + c = 0$. Funkcja przyjmuje wartości parametrów a , b , c oraz referencje dwóch zmiennych rzeczywistych i zwraca wyróżnik równania. Jeżeli równanie posiada rozwiązania, funkcja wpisuje je do dwóch zmiennych wskazywanych otrzymanymi referencjami. Korzystając z tej funkcji napisz program, który wczytuje ze standardowego wejścia współczynniki a , b , c i wypisuje na standardowe wejście wyróżnik równania oraz jego rozwiązania, o ile istnieją.

Przykładowe wykonanie

```
In: 2 -7 -15
Out: 169
Out: -1.5 5
```

Rozwiązanie

```
#include <cmath>
#include <iostream>

double quadratic(double &x1, double &x2, double a, double b, double c) {
    double delta = b * b - 4 * a * c;
    if (delta >= 0) {
        x1 = (-b - std::sqrt(delta)) / a / 2;
        x2 = (-b + std::sqrt(delta)) / a / 2; }
    return delta; }

int main() {
    double a, b, c;
    std::cin >> a >> b >> c;
    double x1, x2;
    double delta = quadratic(x1, x2, a, b, c);
    std::cout << delta << std::endl;
    if (delta >= 0) {
        std::cout << x1 << " " << x2 << std::endl; }}}
```

4.1.3 Clock: Zegar analogowy

Napisz funkcję clock, która przyjmuje całkowitą godzinę i minuty oraz referencje dwóch zmiennych rzeczywistych i wpisuje do tych zmiennych kąty wychylenia małej oraz dużej wskazówki zegara wyrażone w stopniach i liczone zgodnie z ruchem wskazówek od położenia pionowo w górę. Korzystając z tej funkcji napisz program, który wczytuje ze standardowego wejścia godzinę oraz minuty i wypisuje na standardowe wyjście obliczone kąty.

Przykładowe wykonanie

```
In: 13 23
Out: 41.5 138
```

Rozwiązanie

```
#include <iostream>

void clock(double &small, double &large, int hour, int minute) {
    small = (hour % 12) * 30. + minute / 2.;
    large = minute * 6.; }

int main() {
    int hour, minute;
    std::cin >> hour >> minute;
    double small, large;
    clock(small, large, hour, minute);
    std::cout << small << " " << large << std::endl; }
```

4.2 Zadania domowe z działu Funkcje (26 marca, 2 i 9 kwietnia)

4.2.1 Sign: Znak liczby całkowitej

Napisz funkcję `sign`, która przyjmuje liczbę całkowitą i zwraca -1, 0 lub 1 jeżeli liczba ta jest odpowiednio ujemna, równa zero lub dodatnia. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {  
    std::cout << sign(-15) << std::endl; }
```

Wykonanie

Out: -1

Uwaga Spróbuj napisać funkcję bez użycia instrukcji warunkowej ani operatora warunkowego.

4.2.2 Geometric: Ciąg geometryczny

n -ty wyraz ciągu geometrycznego o wyrazie początkowym a_0 i ilorazie q jest równy $a_n = a_0 q^n$. Napisz funkcję `geometric`, która przyjmuje a_0 , q oraz n i zwraca a_n . Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cmath`.

Przykładowy program

```
int main() {  
    std::cout << geometric(8., -0.5, 3) << std::endl; }
```

Wykonanie

Out: -1

4.2.3 Round: Zaokrąglanie z zadaną dokładnością

Zdefiniowana w nagłówku `cmath` funkcja `std::round` zaokrągla podaną liczbę rzeczywistą do najbliższej liczby całkowitej, na przykład `std::round(3.14159)` daje w wyniku 3. Korzystając z tej funkcji napisz własną funkcję `round`, która przyjmuje liczbę rzeczywistą oraz liczbę cyfr po przecinku i zwraca podaną liczbę rzeczywistą zaokrągloną do podanej liczby cyfr po przecinku. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cmath`.

Przykładowy program

```
int main() {  
    std::cout << round(3.14159, 3) << std::endl; }
```

Wykonanie

Out: 3.142

4.2.4 Implication: Implikacja

Implikacja $p \Rightarrow q$ wartości logicznych p i q jest określona tabelą:

p	q	$p \Rightarrow q$
false	false	true
false	true	true
true	false	false
true	true	true

Napisz funkcję `implication`, która przyjmuje p oraz q i zwraca $p \Rightarrow q$. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    std::cout << std::boolalpha << implication(true, false) << std::endl; }
```

Wykonanie

Out: false

Uwaga Spróbuj napisać funkcję bez użycia instrukcji wyboru, instrukcji warunkowej, ani operatora warunkowego.

4.2.5 Lesser: Mniejsza z dwóch liczb

Napisz funkcję `lesser`, która przyjmuje dwie liczby rzeczywiste i zwraca mniejszą z nich. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    std::cout << lesser(3.12, 2.13) << std::endl; }
```

Wykonanie

Out: 2.13

Uwaga Spróbuj napisać funkcję bez użycia instrukcji warunkowej.

4.2.6 Factorial: Silnia

Napisz funkcję `factorial`, która przyjmuje nieujemną liczbę całkowitą i zwraca jej silnię. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    std::cout << factorial(6) << std::endl; }
```

Wykonanie

Out: 720

4.2.7 Bifactorial: Podwójna silnia

Podwójna silnia nieujemnej liczby całkowitej n , oznaczana jako $n!!$, to iloczyn wszystkich dodatnich liczb całkowitych mniejszych lub równych n , o takiej samej parzystości jak n , przy czym $0!! = 1$. Napisz funkcję `bifactorial`, która przyjmuje nieujemną liczbę całkowitą i zwraca jej podwójną silnię. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    std::cout << bifactorial(6) << std::endl; }
```

Wykonanie

Out: 48

4.2.8 Digits: Cyfry dziesiętne

Napisz funkcję `digits`, która przyjmuje nieujemną liczbę całkowitą i zwraca liczbę cyfr w jej zapisie dziesiętnym. Przyjmij, że liczba zero ma zero cyfr. Napisz funkcję `digit`, która przyjmuje nieujemną liczbę całkowitą oraz numer cyfry w jej zapisie dziesiętnym i zwraca tę cyfrę. Cyfry numerujemy od zera dla cyfry jedności. Funkcje powinny być przystosowane do użycia w przykładowym programie poniżej. Funkcje nie korzystają z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {  
    std::cout << digits(2018) << " " << digit(2018, 3) << std::endl; }
```

Wykonanie

Out: 4 2

4.2.9 Uniform: Płaski rozkład prawdopodobieństwa

Napisz funkcję `uniform`, która przyjmuje liczby rzeczywiste a oraz b i zwraca losową liczbę rzeczywistą z przedziału domkniętego od a do b . Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cstdlib`.

Przykładowy program

```
int main() {  
    std::srand(std::time(nullptr));  
    for (int counter = 0; counter < 10; ++counter) {  
        std::cout << uniform(-5., 10.) << " "; }  
    std::cout << std::endl; }
```

Przykładowe wykonanie

Out: -3.39 9.39528 -4.18653 -4.95605 0.977203 -2.45064 1.47938 -4.03088 9.50011 -2.12058

4.2.10 Prime: Czy liczba jest pierwsza - bitcoin

Napisz funkcję `prime`, która przyjmuje nieujemną liczbę całkowitą i zwraca prawdę jeśli jest ona pierwsza oraz fałsz w przeciwnym razie. Przyjmujemy, że jeden nie jest liczbą pierwszą. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {  
    std::cout << std::boolalpha << prime(97) << std::endl; }
```

Wykonanie

Out: true

Uwaga Pierwszość liczby najprościej sprawdzić badając jej podzielność przez wszystkie liczby większe od jednego i mniejsze od niej samej. W ulepszonej wersji wystarczy rozpatrzyć liczby, których kwadrat nie przekracza liczby badanej. Można też opuścić badanie podzielności przez liczby parzyste większe od dwóch. Spróbuj napisać funkcję bez użycia instrukcji warunkowej.

4.2.11 Power: Potęga całkowita

Napisz funkcję `power`, która przyjmuje niezerową liczbę rzeczywistą x oraz dowolną liczbę całkowitą n i zwraca n -tą potęgę liczby x . Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    std::cout << power(-0.5, -3) << std::endl; }
```

Wykonanie

Out: -8

Uwaga Spróbuj napisać funkcję bez użycia instrukcji warunkowej.

4.2.12 Coin: Rzut oszukaną monetą

Napisz funkcję `coin` symulującą rzut oszukaną monetą. Funkcja przyjmuje prawdopodobieństwo wyrzucenia orła i zwraca prawdę jeśli wypadł orzeł albo fałsz jeśli reszka. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cstdlib`.

Przykładowy program

```
int main() {
    std::srand(std::time(nullptr));
    for (int counter = 0; counter < 10; ++counter) {
        std::cout << (coin(0.2) ? "heads" : "tails") << " "; }
    std::cout << std::endl; }
```

Przykładowe wykonanie

Out: heads heads tails tails heads tails tails tails tails tails

4.2.13 Root: Pierwiastek kwadratowy

Pierwiastek kwadratowy z liczby rzeczywistej x można obliczyć następująco. Jeżeli $x < 1$ to pierwiastek leży między 0 a 1, zaś w przeciwnym razie między 1 a x . Bierzemy środek r odpowiedniego z tych przedziałów. Jeżeli $x < r^2$, to poszukiwany pierwiastek leży w lewej połowie przedziału, zaś w przeciwnym razie leży w prawej połowie. Do dalszych rozważań bierzemy więc odpowiednią połowę, dzielimy ją na pół i tak dalej. Dzięki temu w każdym kroku dwukrotnie zawężamy przedział, w którym leży pierwiastek. Ze względu na skończoną dokładność obliczeń środek któregoś kolejnego przedziału okaże się numerycznie równy jednemu z jego krańców. Oznacza to, że znaleźliśmy wynik z dokładnością maszynową. Napisz funkcję `root`, która przyjmuje nieujemną liczbę rzeczywistą i zwraca jej pierwiastek obliczony opisaną metodą. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    std::cout << root(7) << " " << std::sqrt(7) << std::endl; }
```

Wykonanie

Out: 2.64575 2.64575

4.2.14 Pitagoras: Trójki pitagorejskie

Trójką pitagorejską nazywamy każde trzy dodatnie liczby całkowite a , b , c , takie że $a^2 + b^2 = c^2$. Trójkę nazywamy pierwotną, jeżeli liczby a , b , c są względnie pierwsze. Napisz program `pitagoras`, który wczytuje ze standardowego wejścia dodatnią liczbę całkowitą d i wypisuje na standardowe wyjście wszystkie pierwotne trójki pitagorejskie, dla których $c < d$. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

```
In: 30
Out: 3 4 5
Out: 5 12 13
Out: 8 15 17
Out: 7 24 25
Out: 20 21 29
```

4.2.15 RNG: Generator liczb losowych

Napisz bezargumentową funkcję `rng`, która zwraca pseudolosową liczbę całkowitą. Funkcja oblicza kolejną liczbę x_{next} na podstawie poprzedniej liczby x_{previous} ze wzoru

$$x_{\text{next}} = (33 * x_{\text{previous}} + 1) \bmod 1024$$

Jest to prosta wersja liniowego generatora kongruentnego. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta ze zmiennych globalnych ani z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    for (int counter = 0; counter < 10; ++counter) {
        std::cout << rng() << " ";
    }
    std::cout << std::endl;
}
```

Wykonanie

```
Out: 1 34 99 196 325 486 679 904 137 426
```

4.2.16 DMS: Stopnie, minuty, sekundy

W geografii kąty często wyraża się podając całkowite liczby stopni, minut i sekund. Napisz funkcję `dms`, która przyjmuje rzeczywistą liczbę stopni oraz referencje trzech zmiennych całkowitych i wpisuje do tych zmiennych całkowite liczby stopni, minut oraz sekund. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    int degrees, minutes, seconds;
    dms(degrees, minutes, seconds, 123.37);
    std::cout << degrees << " " << minutes << " " << seconds << std::endl;
}
```

Wykonanie

```
Out: 123 22 12
```

4.2.17 Fraction: Część całkowita i ułamkowa - bitcoin

Napisz funkcję `fraction`, która przyjmuje nieujemną liczbę rzeczywistą oraz referencje dwóch zmiennych rzeczywistych i wpisuje do nich całkowitą oraz ułamkową część podanej liczby. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `cmath`.

Przykładowy program

```
int main() {
    double integral, fractional;
    fraction(integral, fractional, 3.14159);
    std::cout << integral << " " << fractional << std::endl; }
```

Wykonanie

Out: 3 0.14159

4.2.18 Exchange: Zamiana wartości zmiennej

Napisz funkcję `exchange`, która przyjmuje referencję zmiennej rzeczywistej oraz liczbę rzeczywistą, wpisuje liczbę do zmiennej i zwraca poprzednią wartość zmiennej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    double a = 2.71828, b = exchange(a, 3.14159);
    std::cout << a << " " << b << std::endl; }
```

Wykonanie

Out: 3.14159 2.71828

4.2.19 RNG: Liczby pseudolosowe

Napisz funkcję `rng` generującą pseudolosowe liczby całkowite. Funkcja oblicza kolejną liczbę x_{next} na podstawie poprzedniej x_{previous} według wzoru

$$x_{\text{next}} = (33 * x_{\text{previous}} + 1) \bmod 1024$$

Funkcja nie przyjmuje żadnych argumentów i zwraca referencję statycznej zmiennej zadeklarowanej wewnątrz funkcji. Każde wywołanie funkcji traktuje wartość tej zmiennej jako x_{previous} i nadaje jej wartość x_{next} . Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {
    rng() = 7;
    for (int counter = 0; counter < 10; ++counter) {
        std::cout << rng() << " ";
    }
    std::cout << std::endl; }
```

Wykonanie

Out: 232 489 778 75 428 813 206 655 112 625

4.2.20 Choose: Wybór zmiennej

Napisz funkcję `choose`, która przyjmuje wartość logiczną oraz referencje dwóch zmiennych rzeczywistych i zwraca referencję pierwszej albo drugiej z nich jeżeli przekazana wartość logiczna jest odpowiednio prawdziwa albo fałszywa. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja nie korzysta z żadnych plików nagłówkowych.

Przykładowy program

```
int main() {  
    double a = 2.3, b = 3.2;  
    choose(a, b, false) = 10.9;  
    std::cout << a << " " << b << std::endl; }
```

Wykonanie

Out: 2.3 10.9

5 Wektory: 24 i 25 marca

Wektory

5.1 Przykłady laboratoryjne z działu Wektory

5.1.1 Bubble Sort: Sortowanie bąbelkowe

Bąbelkowe sortowanie wektora przebiega następująco. Porównujemy pierwszy element z drugim i jeśli są w niewłaściwej kolejności, to zamieniamy je miejscami. Następnie porównujemy drugi element z trzecim i tak dalej, do końca wektora. Jeżeli w takim pojedynczym przebiegu musieliśmy wykonać choćby jedną zamianę, to powtarzamy wszystko od początku. W przeciwnym razie wektor jest już posortowany. Napisz funkcję `bubble_sort`, która przyjmuje referencję wektora liczb całkowitych i sortuje go bąbelkowo w kolejności niemalejącej. Korzystając z tej funkcji napisz program, który czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku i wypisuje je na standardowe wyjście w kolejności niemalejącej.

Przykładowe wykonanie

```
In: 5 7 3 6 3 5 1 9 3 1
Out: 1 1 3 3 3 5 5 6 7 9
```

Rozwiązanie

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

void bubble_sort(std::vector<int> &vector) {
    bool unordered = vector.size() > 1;
    while (unordered) {
        unordered = false;
        for (int index = 0; index + 1 < vector.size(); ++index) {
            if (vector[index] > vector[index + 1]) {
                std::swap(vector[index], vector[index + 1]);
                unordered = true;
            }
        }
    }
}

int main() {
    std::vector<int> vector;
    for (int element; std::cin >> element;) {
        vector.push_back(element);
    }
    bubble_sort(vector);
    for (int index = 0; index < vector.size(); index++) {
        std::cout << vector[index] << " ";
    }
    std::cout << std::endl;
}
```

5.1.2 Select: Wybór elementów

Napisz funkcję `select`, która przyjmuje stałą referencję wektora liczb całkowitych i zwraca wektor zawierający tylko jego ujemne elementy, w niezmienionej kolejności. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    std::vector<int> result = select(std::vector<int> {3, -1, 12, -5, 7, -10});
    for (int index = 0; index < result.size(); ++index) {
        std::cout << result[index] << " ";
    }
    std::cout << std::endl;
}
```

Wykonanie

Out: -1 -5 -10

Rozwiązanie

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

std::vector<int> select(const std::vector<int> &vector) {
    std::vector<int> result;
    for (int index = 0; index < vector.size(); ++index) {
        if (vector[index] < 0) {
            result.push_back(vector[index]);
        }
    }
    return result;
}

int main() {
    std::vector<int> result = select(std::vector<int> {3, -1, 12, -5, 7, -10});
    for (int index = 0; index < result.size(); ++index) {
        std::cout << result[index] << " ";
    }
    std::cout << std::endl;
}
```

5.1.3 Reverse: Odwracanie kolejności elementów

Napisz funkcję `reverse`, która przyjmuje referencję wektora liczb całkowitych i odwraca kolejność jego elementów. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej.

Przykładowy program

```
int main() {
    std::vector<int> vector {7, 3, -1, 9, 2};
    reverse(vector);
    for (int index = 0; index < vector.size(); ++index) {
        std::cout << vector[index] << " ";
    }
    std::cout << std::endl;
}
```

Wykonanie

Out: 2 9 -1 3 7

Rozwiązanie

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

void reverse(std::vector<int> &vector) {
    for (int begin = 0, end = vector.size(); begin < end;) {
        int element = vector[--end];
        vector[end] = vector[begin];
        vector[begin++] = element;
    }
}

int main() {
    std::vector<int> vector {7, 3, -1, 9, 2};
    reverse(vector);
    for (int index = 0; index < vector.size(); ++index) {
        std::cout << vector[index] << " ";
    }
    std::cout << std::endl;
}
```


5.2 Zadania domowe z działu Wektory (2, 9, 23 kwietnia)

5.2.1 Accumulate: Akumulacja

Napisz funkcję `accumulate`, która przyjmuje referencję wektora liczb rzeczywistych i do każdego elementu dodaje sumę wszystkich go poprzedzających. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<double> vector {3.1, 2.7, -0.5, 0.1, 4.3};
    accumulate(vector);
    for (double element: vector) {
        std::cout << element << " ";
    }
    std::cout << std::endl; }
```

Wykonanie

Out: 3.1 5.8 5.3 5.4 9.7

5.2.2 Shuffle: Tasowanie elementów

Napisz funkcję `shuffle`, która przyjmuje referencję wektora liczb rzeczywistych i losowo przestawia jego elementy. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `cstdlib`, `utility` i `vector`.

Przykładowy program

```
int main() {
    std::srand(std::time(nullptr));
    std::vector<double> vector {7.2, -1, 12.3, 3, 10.5};
    shuffle(vector);
    for (double element: vector) {
        std::cout << element << " ";
    }
    std::cout << std::endl; }
```

Wykonanie

Out: 10.5 12.3 7.2 3 -1

5.2.3 Rotate: Cykliczne przesunięcie wektora

Cykliczne przesunięcie wektora o jedną pozycję w lewo polega na przesunięciu elementów o indeksach większych od zera o jedną pozycję w lewo i przestawieniu oryginalnego elementu o indeksie zero na koniec. Napisz funkcję `rotate`, która przyjmuje referencję wektora liczb całkowitych oraz liczbę naturalną n i przesuwa elementy wektora cyklicznie o n pozycji w lewo. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `utility` i `vector`.

Przykładowy program

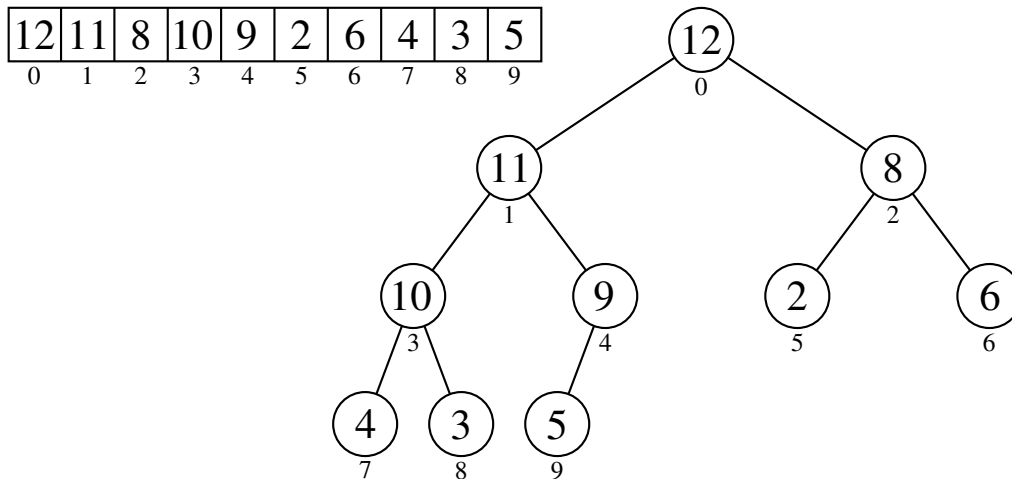
```
int main() {
    std::vector<double> vector {2.5, 0, 12, 3, 7, -2.8, 0.1};
    rotate(vector, 2);
    for (double element: vector) {
        std::cout << element << " ";
    }
    std::cout << std::endl; }
```

Wykonanie

Out: 12 3 7 -2.8 0.1 2.5 0

5.2.4 Heap Sort: Sortowanie kopcowe

Zupełny kopiec binarny to rodzaj drzewa przedstawiony na rysunku. Każdy węzeł jest mniejszy lub równy swojemu rodzicowi. Węzły dodajemy do kopca w kolejności wyznaczonej małymi cyframi, które są jednocześnie numerami węzłów. Kopiec wygodnie jest przechowywać w wektorze, jak zaznaczono na rysunku. Wartość każdego węzła przechowujemy w komórce wektora o indeksie równym numerowi węzła.



Kopcowe sortowanie wektora w kolejności niemalejącej odbywa się w dwóch etapach. W pierwszym wyjmujemy z wektora kolejne liczby od lewej do prawej i budujemy z nich kopiec. Każdą liczbę dodajemy początkowo jako ostatni węzeł kopca według opisanej kolejności. Może się jednak zdarzyć, że tak dodany węzeł jest większy od swojego rodzica. W takiej sytuacji zamieniamy go z rodzicem miejscami. Czynność tę powtarzamy aż przestawiany węzeł okaże się mniejszy lub równy nowemu rodzicowi bądź zawędruje na samą górę kopca stając się jego nowym korzeniem.

W drugim etapie wyjmujemy z kopca kolejne liczby od największej do najmniejszej i wpisujemy je do wektora od końca. Największym węzłem kopca jest oczywiście jego korzeń, więc jego wyjmujemy najpierw. Następnie odbudowujemy naruszoną w ten sposób strukturę kopca. Na miejsce korzenia wstawiamy ostatni węzeł kopca. Prawdopodobnie okaże się on mniejszy od swoich nowych dzieci. W takiej sytuacji zamieniamy go miejscami z większym dzieckiem. Czynność tę powtarzamy aż przestawiany węzeł okaże się mniejszy lub równy nowemu rodzicowi.

Wszystkie te operacje wykonujemy bezpośrednio w sortowanym wektorze, który jednocześnie przechowuje kopiec. W pierwszym etapie kopiec rozbudowuje się od lewej przejmując kolejne elementy wektora, zaś w drugim kurczy się pozostawiając po prawej elementy posortowane.

Napisz funkcję `heap_sort`, która przyjmuje referencję wektora liczb całkowitych i sortuje go kopcowo w kolejności niemalejącej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z plików nagłówkowych `utility` i `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {3, 7, -1, 12, -5, 7, 10};
    heap_sort(vector);
    for (int element: vector) {
        std::cout << element << " ";
    }
    std::cout << std::endl;
}
```

Wykonanie

Out: -5 -1 3 7 7 10 12

5.2.5 Selection Sort: Sortowanie przez wybór - bitcoin

Sortowanie wektora w kolejności niemalejącej przez wybór przebiega następująco. Znajdujemy w wektorze pierwszy element najmniejszy i zamieniamy go z pierwszym elementem wektora. Następnie powtarzamy te czynności dla wektora bez pierwszego elementu i tak dalej. Napisz program `selection_sort`, który czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku i sortuje je niemalejąco przez wybór. Program wypisuje na standardowe wyjście w kolejnych liniach sortowane liczby po każdej zamianie. Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

```
In: 3 7 -1 12 -5 7 10
Out: -5 7 -1 12 3 7 10
Out: -5 -1 7 12 3 7 10
Out: -5 -1 3 12 7 7 10
Out: -5 -1 3 7 12 7 10
Out: -5 -1 3 7 7 12 10
Out: -5 -1 3 7 7 10 12
Out: -5 -1 3 7 7 10 12
```

5.2.6 Find: Wyszukiwanie elementu - bitcoin

Napisz funkcję `find`, która przyjmuje stałą referencję wektora liczb całkowitych oraz pojedynczą wartość całkowitą i zwraca indeks pierwszego wystąpienia tej wartości w wektorze albo długość wektora jeżeli ta wartość w nim nie występuje. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    int result = find(std::vector<int> {3, -1, 7, 12, -5, 7, 10}, 7);
    std::cout << result << std::endl; }
```

Wykonanie

Out: 2

5.2.7 Count: Zliczanie elementów

Napisz funkcję `count`, która przyjmuje stałą referencję wektora liczb całkowitych oraz pojedynczą wartość całkowitą i zwraca liczbę wystąpień tej wartości w zadanym wektorze. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    int result = count(std::vector<int> {3, 7, -1, 12, -5, 7, 10}, 7);
    std::cout << result << std::endl; }
```

Wykonanie

Out: 2

5.2.8 Minimum: Element najmniejszy

Napisz funkcję `minimum`, która przyjmuje stałą referencję wektora liczb całkowitych i zwraca indeks najmniejszego elementu. Jeżeli takich elementów jest kilka, funkcja zwraca indeks pierwszego z nich. Jeżeli taki element nie istnieje, funkcja zwraca długość wektora. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    int result = minimum(std::vector<int> {9, -7, 5, 1, 12, 3, -7});
    std::cout << result << std::endl; }
```

Wykonanie

Out: 1

5.2.9 Binary Search: Wyszukiwanie binarne

Chcąc znaleźć daną wartość w wektorze posortowanym niemalejąco można zastosować wyszukiwanie binarne. Rozważamy najpierw element leżący w połowie wektora. Jeżeli jest on większy od poszukiwanej wartości, to na pewno leży ona w lewej połowie wektora, więc do niej ograniczamy dalsze poszukiwania. W przeciwnym razie prowadzimy je tylko w prawej połowie. Wybraną połowę znów dzielimy na pół i tak dalej. Napisz funkcję `binary_search`, która przyjmuje stałą referencję posortowanego niemalejąco wektora liczb rzeczywistych oraz pojedynczą wartość rzeczywistą i zwraca indeks pierwszego elementu większego od podanej wartości. Jeżeli taki element nie istnieje, funkcja zwraca długość wektora. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    int result = binary_search(std::vector<double> {-4.3, -1.2, 0.1, 8.2, 11.8}, 7.6);
    std::cout << result << std::endl; }
```

Wykonanie

Out: 3

5.2.10 Count Sort: Sortowanie przez zliczanie

Wektor zawierający tylko liczby całkowite od zera włącznie do k wyłącznie najszybciej jest posortować metodą zliczania. Zliczamy, ile w sortowanym wektorze jest zer, jedynek i tak dalej. Następnie, sortując w kolejności niemalejącej, zaczynamy od początku wektora i wpisujemy do kolejnych komórek tyle zer, ile zliczyliśmy, potem tyle jedynek, ile zliczyliśmy, i tak dalej. Napisz funkcję `count_sort`, która przyjmuje referencję wektora liczb całkowitych oraz liczbę k i sortuje ten wektor metodą zliczania w kolejności niemalejącej. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main() {
    std::vector<int> vector {4, 2, 7, 5, 2, 1, 5};
    count_sort(vector, 10);
    for (int element: vector) {
        std::cout << element << " "; }
    std::cout << std::endl; }
```

Wykonanie

Out: 1 2 2 4 5 5 7

5.2.11 Back: Wydruk w odwrotnej kolejności

Napisz program `back`, który czyta ze standardowego wejścia liczby rzeczywiste do napotkania końca pliku i wypisuje je na standardowe wyjście w odwrotnej kolejności. Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

In: 3.1 2.7 -0.5 0.1 4.3
Out: 4.3 0.1 -0.5 2.7 3.1

5.2.12 Maximum Sum: Przedział o największej sumie elementów

Napisz program `maximum_sum`, który czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku i wypisuje na standardowe wyjście podciąg tych liczb o największej dodatniej sumie elementów. Jeżeli takich podciągów jest kilka, program wypisuje najkrótszy i najwcześniej się zaczynający. Jeżeli taki podciąg nie istnieje, program nic nie wypisuje. Zaproponuj algorytm o złożoności liniowej. Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

In: -5 0 -3 -2 -3 -3 -3 0 0 -2 -2 3 4 -1 4 5 -4 1 3 -2
Out: 3 4 -1 4 5

5.2.13 Longest Slope: Najdłuższy przedział niemalejący

Napisz program `longest_slope`, który czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku i wypisuje na standardowe wyjście najdłuższy niemalejący podciąg tych liczb. Jeżeli takich podciągów jest kilka, program wypisuje najwcześniej się zaczynający. Zaproponuj algorytm o złożoności liniowej. Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

In: 10 10 2 4 7 13 8 2 0 17 7 0 1 12 18 18 17 6 5 19
Out: 0 1 12 18 18

5.2.14 Triangles: Liczba trójkątów

Napisz program `triangles`, który czyta ze standardowego wejścia podane w kolejności niemalejącej długości odcinków aż do napotkania końca pliku i wypisuje na standardowe wyjście liczbę trójkątów, które można z nich zbudować. Każdego odcinka można użyć w jednym trójkącie tylko raz a dwa odcinki o jednakowych długościach uważamy za różne. Zaproponuj algorytm o kwadratowej złożoności obliczeniowej. Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

In: 1 2.5 3 3.7
Out: 3

5.2.15 Relay: Przesiadka

Kierowca planuje podróż z miasta początkowego przez kilka pośrednich do miasta końcowego. Trasa jest na tyle długa, że wymaga po drodze jednego noclegu. Kierowca chce tak wybrać miasto na nocleg, aby kilometraż pierwszego i drugiego dnia podróży jak najmniej się różniły. Napisz program `relay`, który wczytuje ze standardowego wejścia odległości między kolejnymi miastami do napotkania końca pliku i wypisuje na standardowe wyjście numer miasta, w którym wypada nocleg, przy czym miasto początkowe ma numer zero. Program załącza tylko pliki nagłówkowe `cmath`, `iostream` i `vector`.

Przykładowe wykonanie

In: 15.2 23.1 2.5 7.3 11 5.3
Out: 2

5.2.16 Intersection: Część wspólna zbiorów

Napisz funkcję `intersection`, która przyjmuje stałe referencje dwóch uporządkowanych rosnąco wektorów liczb całkowitych i zwraca uporządkowany rosnąco wektor liczb zawartych w obu tych wektorach jednocześnie. Funkcja powinna być przystosowana do użycia w przykładowym programie poniżej. Funkcja korzysta tylko z pliku nagłówkowego `vector`.

Przykładowy program

```
int main () {
    for (int element: intersection(std::vector<int> {-7, 2, 3, 7, 15, 18, 23},
                                   std::vector<int> {-8, 3, 5, 8, 15, 23, 30})) {
        std::cout << element << " "; }
    std::cout << std::endl; }
```

Wykonanie

Out: 3 15 23

6 Argumenty: 31 marca i 1 kwietnia

Argumenty wywołania programu

6.1 Przykłady laboratoryjne z działu Argumenty

6.1.1 Eratostenes: Sito Eratostenesa

Wszystkie liczby pierwsze mniejsze od dodatniej liczby całkowitej n można wyznaczyć następującą metodą zwaną sitem Eratostenesa. Spośród liczb większych od 1 i mniejszych od n wykreślamy wszystkie wielokrotności 2 poczynając od 4, następnie wszystkie wielokrotności 3 poczynając od 6 i tak dalej. Pozostałe na koniec niewykreślone liczby to wszystkie liczby pierwsze mniejsze od n . Napisz funkcję `eratostenes`, która przyjmuje dodatnią liczbę całkowitą i zwraca wektor zawierający wszystkie mniejsze od niej liczby pierwsze. Korzystając z tej funkcji napisz program, który przyjmuje jako argument wywołania dodatnią liczbę całkowitą i wypisuje na standardowe wyjście wszystkie mniejsze od niej liczby pierwsze.

Przykładowe wykonanie

```
Linux: ./eratostenes 100
Windows: eratostenes.exe 100
Out: 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Uwaga Korzystając z następujących obserwacji można przyspieszyć działanie programu i zmniejszyć zużycie pamięci:

1. Wielokrotności każdej liczby złożonej zostały już wykreślone wcześniej jako wielokrotności jej dzielników. Można więc pominąć wykreślanie wielokrotności liczb już wykreślonych.
2. Wielokrotności liczby k od drugiej do $(k - 1)$ -szej zostały już wykreślone wcześniej jako wielokrotności liczb $2, \dots, (k - 1)$. Wykreślanie wielokrotności każdej liczby można więc zacząć od jej kwadratu.
3. Jeżeli kwadrat danej liczby jest większy lub równy n , to kwadraty wszystkich kolejnych będą większe od n . Oznacza to, że wszystkie liczby złożone mniejsze od n zostały już wykreślone i całą procedurę można zakończyć.

Rozwiązanie

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

std::vector<int> eratostenes(int limit) {
    std::vector<bool> sieve(limit, true);
    for (int number = 2; number < limit; ++number) {
        for (int multiple = 2 * number; multiple < limit; multiple += number) {
            sieve[multiple] = false;
        }
    }
    std::vector<int> primes;
    for (int number = 2; number < limit; ++number) {
        if (sieve[number]) {
            primes.push_back(number);
        }
    }
    return primes;
}

int main(int argc, char *argv[]) {
    int limit = std::atoi(argv[1]);
    std::vector<int> primes = eratostenes(limit);
    for (int index = 0; index < primes.size(); ++index) {
        std::cout << primes[index] << " ";
    }
    std::cout << std::endl;
}
```

6.1.2 Beaufort: Skala Beauforta

Skala Beauforta określona jest następującą tabelą:

Beaufort	km/h
0	0 - 0.5
1	0.5 - 6.5
2	6.5 - 11.5
3	11.5 - 19.5
4	19.5 - 29.5
5	29.5 - 39.5
6	39.5 - 50.5
7	50.5 - 62.5
8	62.5 - 75.5
9	75.5 - 87.5
10	87.5 - 102.5
11	102.5 - 117.5
12	117.5 - ∞

Napisz program beaufort, który przyjmuje jako argument wywołania prędkość wiatru w kilometrach na godzinę i wypisuje na standardowe wyjście siłę wiatru w skali Beauforta.

Przykładowe wykonanie

```
Linux: ./beaufort 29.2
Windows: beaufort.exe 29.2
Out: 4
```

Rozwiązanie

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

int main(int argc, char *argv[]) {
    double speed = std::atof(argv[1]);
    const std::vector<double> limits {
        0.5, 6.5, 11.5, 19.5, 29.5, 39.5, 50.5, 62.5, 75.5, 87.5, 102.5, 117.5};
    int degree = 0;
    for (; degree < limits.size() && limits[degree] < speed; ++degree);
    std::cout << degree << std::endl; }
```


6.2 Zadania domowe z działu Argumenty (9, 23, 30 kwietnia)

6.2.1 Sum: Suma argumentów - bitcoin

Napisz program `sum`, który przyjmuje jako argumenty wywołania nieznana z góry liczbę wartości rzeczywistych i wypisuje na standardowe wyjście ich sumę. Program załącza tylko pliki nagłówkowe `cstdlib` i `iostream`.

Przykładowe wykonanie

```
Linux: ./sum 0.6 -0.2 1.3
Windows: sum.exe 0.6 -0.2 1.3
Out: 1.7
```

6.2.2 Permutation: Losowa permutacja

Napisz program `permutation`, który przyjmuje jako argument wywołania nieujemną liczbę całkowitą n i wypisuje na standardowe wyjście losową permutację wszystkich liczb całkowitych od 1 do n . Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime`, `iostream` i `vector`.

Przykładowe wykonanie

```
Linux: ./permutation 10
Windows: permutation.exe 10
Out: 8 1 10 3 4 6 2 9 7 5
```

6.2.3 Permutations: Wszystkie permutacje

Napisz program `permutations`, który przyjmuje jako argument wywołania nieujemną liczbę całkowitą n i wypisuje na standardowe wyjście wszystkie permutacje liczb od 1 do n . Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
Latex: ./permutations 3
Windows: permutations.exe 3
Out: 1 2 3
Out: 1 3 2
Out: 2 1 3
Out: 2 3 1
Out: 3 1 2
Out: 3 2 1
```

6.2.4 Pascal: Trójkąt Pascala

Napisz program `pascal`, który przyjmuje jako argument wywołania liczbę naturalną n i drukuje na standardowe wyjście n pierwszych wierszy trójkąta Pascala. Wydruk jest automatycznie formatowany zależnie od liczby wierszy. Program załącza tylko pliki nagłówkowe `iomanip`, `iostream` i `vector`.

Przykładowe wykonanie

```
Linux: ./pascal 6
Windows: pascal.exe 6
Out:
      1
Out:    1 1
Out:   1 2 1
Out:  1 3 3 1
Out: 1 4 6 4 1
Out: 1 5 10 10 5 1
```

6.2.5 Days: Długości miesięcy

Napisz program `days`, który przyjmuje jako argumenty wywołania rok oraz numer miesiąca i wypisuje na standardowe wyjście liczbę dni w tym miesiącu z uwzględnieniem lat przestępnych. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
Linux: ./days 2000 2
Windows: days.exe 2000 2
Out: 29
```

6.2.6 Nominals: Odliczanie kwoty w nominałach - bitcoin

W pewnym państwie emitowane są nominały 1, 2, 5, 10, 20, 50, 100, 200. Napisz program `nominals`, który przyjmuje jako argument wywołania kwotę bez groszy i wypisuje na standardowe wyjście dającą w sumie nominały, od największego do najmniejszego. Program odlicza zadaną kwotę w możliwie najmniejszej liczbie nominałów dysponując nieograniczoną liczbą monet lub banknotów każdego nominału. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
Linux: ./nominals 739
Windows: nominals.exe 739
Out: 200 200 200 100 20 10 5 2 2
```

6.2.7 Exact Sum: Przedział o zadanej sumie elementów

Napisz program `exact_sum`, który przyjmuje jako argument wywołania pojedynczą liczbę całkowitą, czyta ze standardowego wejścia liczby całkowite do napotkania końca pliku i wypisuje na standardowe wyjście pierwszy napotkany podciąg tych liczb o sumie zadanej argumentem wywołania. Jeżeli taki podciąg nie istnieje, program nic nie wypisuje. Zaproponuj algorytm o złożoności liniowej. Program załącza tylko pliki nagłówkowe `cstdlib`, `iostream` i `vector`.

Przykładowe wykonanie

```
Linux: ./exact_sum 18
Windows: exact_sum.exe 18
In: 3 7 -1 12 -5 7 10
Out: 7 -1 12
```

6.2.8 Horner: Schemat Hornera

Rozważmy wielomian a stopnia n o współczynnikach rzeczywistych. Niech wielomian b oraz liczba r będą odpowiednio ilorazem oraz resztą z dzielenia wielomianu a przez dwumian $x - c$. Wielomian b oraz resztę r można obliczyć stosując kolejno wzory:

$$\begin{aligned} b_{n-1} &= a_n \\ b_{n-2} &= a_{n-1} + c \cdot b_{n-1} \\ &\dots \\ b_0 &= a_1 + c \cdot b_1 \\ r &= a_0 + c \cdot b_0 \end{aligned}$$

Taki sposób dzielenia wielomianu przez jednomian nazywa się schematem Hornera. Napisz program `horner`, który przyjmuje jako argument wywołania współczynnik c , następnie czyta ze standardowego wejścia współczynniki wielomianu a od zerowego do napotkania końca pliku i wypisuje na standardowe wyjście w kolejnych liniach resztę r oraz współczynniki wielomianu b . Program załącza tylko pliki nagłówkowe `iostream` i `vector`.

Przykładowe wykonanie

```
Linux: ./horner 0.8
Windows: horner.exe 0.8
In: 0.5 -1.2 2 3.8
Out: 2.7656
Out: 2.832 5.04 3.8
```

7 Pliki: 7 i 8 kwietnia

Znaki, Pliki tekstowe

7.1 Przykłady laboratoryjne z działu Pliki

7.1.1 Capital: Zamiana liter na wielkie

Napisz program `capital`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zamieniając małe litery na wielkie.

Przykładowy plik wejściowy `inut.txt`

```
Ala ma 15 lat!  
Czy Ala ma 17 lat?
```

Przykładowe wywołanie

```
Linux: ./capital input.txt output.txt  
Windows: capital.exe input.txt output.txt
```

Przykładowy plik wyjściowy `output.txt`

```
ALA MA 15 LAT!  
CZY ALA MA 17 LAT?
```

7.1.2 WC: Zliczanie znaków, wyrazów i linii

Napisz program `wc`, który przyjmuje jako argument wywołania nazwę pliku tekstowego i wypisuje na standardowe wyjście liczbę zawartych w tym pliku linii, słów, oraz znaków łącznie ze znakami białymi. Linia to każdy ciąg znaków zakończony znakiem końca wiersza lub niepusty ciąg znaków poprzedzający koniec pliku.

Przykładowy plik `input.txt`

```
ala ma kota[LF]  
lorem ipsum dolor sit amet[LF]  
[LF]  
litwo ojczyzna moja[EOF]
```

Przykładowe wykonanie

```
Linux: ./wc input.txt  
Windows: wc.exe input.txt  
Out: 4 11 59
```

7.2 Zadania domowe z działu Pliki (23, 30 kwietnia, 7 maja)

7.2.1 Char: Znak o zadanym kodzie

Napisz program `char`, który wczytuje ze standardowego wejścia kod znaku i wypisuje ten znak na standardowe wyjście. Program załącza tylko plik nagłówkowy `iostream`.

Przykładowe wykonanie

In: 82
Out: R

7.2.2 ASCII: Tabela kodów ASCII

Napisz program `ascii`, który wypisuje na standardowe wyjście wszystkie znaki od wykrzyknika do litery zet wraz z ich kodami. Program załącza tylko plik nagłówkowy `iostream`.

Wykonanie

Out: 33 !	43 +	53 5	63 ?	73 I	83 S	93]	103 g	113 q
Out: 34 "	44 ' ,	54 6	64 @	74 J	84 T	94 ^	104 h	114 r
Out: 35 #	45 -	55 7	65 A	75 K	85 U	95 _	105 i	115 s
Out: 36 \$	46 .	56 8	66 B	76 L	86 V	96 ' ,	106 j	116 t
Out: 37 %	47 /	57 9	67 C	77 M	87 W	97 a	107 k	117 u
Out: 38 &	48 0	58 :	68 D	78 N	88 X	98 b	108 l	118 v
Out: 39 ' ,	49 1	59 ;	69 E	79 O	89 Y	99 c	109 m	119 w
Out: 40 (50 2	60 <	70 F	80 P	90 Z	100 d	110 n	120 x
Out: 41)	51 3	61 =	71 G	81 Q	91 [101 e	111 o	121 y
Out: 42 *	52 4	62 >	72 H	82 R	92 \	102 f	112 p	122 z

7.2.3 Password: Losowe hasło

Napisz program `password`, który przyjmuje jako argument wywołania długość hasła i wypisuje na standardowe wyjście tej długości hasło złożone z losowych cyfr i małych liter. Program załącza tylko pliki nagłówkowe `cstdlib`, `ctime` i `iostream`.

Przykładowe wykonanie

Linux: `./password 7`
Windows: `password.exe 7`
Out: 5zy4hsu

7.2.4 Is: Znaki białe, cyfry, małe i duże litery

Korzystając z funkcji z pliku nagłówkowego `cctype` napisz program `is`, który wczytuje ze standardowego wejścia jeden znak bez opuszczania znaków białych i wypisuje na standardowe wyjście `digit`, `upper`, `lower` albo `other` jeśli jest on odpowiednio cyfrą, wielką literą, małą literą albo innym znakiem. Program załącza tylko pliki nagłówkowe `cctype` i `iostream`.

Przykładowe wykonanie

In: a
Out: lower

7.2.5 Letters: Zliczanie liter

Napisz program `letters`, który czyta ze standardowego wejścia znaki do napotkania końca pliku, a następnie wypisuje na standardowe wyjście liczby wystąpień kolejnych liter pośród wpisanych znaków. Program nie rozróżnia wielkich i małych liter oraz pomija wszelkie inne znaki. Program załącza tylko pliki nagłówkowe `cctype`, `iostream` i `vector`.

Przykładowe wykonanie

```
In: Lorem ipsum dolor sit amet!  
Out: 1 0 0 1 2 0 0 0 2 0 0 2 3 0 3 1 0 2 2 2 1 0 0 0 0 0
```

7.2.6 Phones: Numery telefonów

Pewien plik tekstowy zawiera numery telefonów zapisane jak poniżej. Napisz program `phones`, który przyjmuje nazwę tego pliku jako argument wywołania, po uruchomieniu wczytuje ze standardowego wejścia sekwencję cyfr i wypisuje na standardowe wyjście wszystkie numery telefonów zawierające tę sekwencję. Następnie wczytuje kolejną liczbę i tak do napotkania końca pliku. Program załącza tylko pliki nagłówkowe `fstream`, `iostream` i `vector`.

Przykładowy plik wejściowy `input.txt`

```
72648120 5812419 4285725 3897124 159004621 8887133
```

Przykładowe wykonanie

```
Linux: ./phones input.txt  
Windows: phones.exe input.txt  
In: 12  
Out: 72648120 5812419 3897124  
In: 124  
Out: 5812419 3897124
```

7.2.7 Cat: Łączenie plików tekstowych

Napisz program `cat`, który przyjmuje jako argumenty wywołania nazwy dowolnej liczby plików tekstowych i wypisuje zawartości kolejnych plików na standardowe wyjście nie wstawiając między nimi żadnych dodatkowych znaków. Program załącza tylko pliki nagłówkowe `fstream` i `iostream`.

Przykładowy plik wejściowy `input1.txt`

```
to samo
```

Przykładowy plik wejściowy `input2.txt`

```
lot do Londynu
```

Przykładowe wykonanie

```
Linux: ./cat input1.txt input2.txt  
Windows: cat.exe input1.txt input2.txt  
Out: to samolot do Londynu
```

7.2.8 Departures: Godziny odjazdów

Pewien plik tekstowy zawiera godziny odjazdu pociągu zapisane w kolejności rosnącej jak poniżej. Napisz program `departures`, który przyjmuje nazwę takiego pliku jako argument wywołania. Po uruchomieniu wczytuje ze standardowego wejścia godzinę, na przykład `10:45`, wypisuje na standardowe wyjście godzinę odjazdu najbliższego pociągu, ponownie wczytuje godzinę i tak dalej, do napotkania końca pliku. Jeżeli tego samego dnia nie ma już żadnego połączenia, program wypisuje godzinę odjazdu pierwszego pociągu następnego dnia. Program załącza tylko pliki nagłówkowe `fstream`, `iomanip`, `iostream` i `vector`.

Przykładowy plik wejściowy `input.txt`

```
6:50 8:27 10:30 12:05 13:48 16:07 18:00 21:30
```

Przykładowe wykonanie

```
Linux: ./departures input.txt
Windows: departures.exe input.txt
In: 10:45 Out: 12:05
In: 13:48 Out: 13:48
In: 22:03 Out: 6:50
```

Uwaga Godzinę z minutami pamiętaj jako jedną liczbę całkowitą, której dwie ostatnie cyfry to minuty, a wcześniejsze to godzina, na przykład 8:05 to 805.

7.2.9 Numerator: Numerowanie linii - bitcoin

Napisz program `numerator`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego dopisując na początku każdej linii jej kolejny numer poczynając od jednego. Program załącza tylko plik nagłówkowy `fstream`.

Przykładowy plik wejściowy `input.txt`

```
Ala ma kota.

To kot Ali.
```

Przykładowe wywołanie

```
Linux: ./numerator input.txt output.txt
Windows: numerator.exe input.txt output.txt
```

Przykładowy plik wyjściowy `output.txt`

```
1 Ala ma kota.
2
3 To kot Ali.
```

7.2.10 Separator: Odstępy między wyrazami

Napisz program `separator`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zamieniając każdą sekwencję spacji i tabulatorów na pojedynczą spację. Program załącza tylko plik nagłówkowy `fstream`.

Przykładowy plik wejściowy `input.txt`

```
Ala      ma      kota.
To kot      Ali.
```

Przykładowe wywołanie

```
Linux: ./separator input.txt output.txt
Windows: separator.exe input.txt output.txt
```

Przykładowy plik wyjściowy `output.txt`

```
Ala ma kota.
To kot Ali.
```

7.2.11 Caesar: Szyfr Cezara - bitcoin

Kodowanie wiadomości tekstowej szyfrem Cezara z przesunięciem n polega na zastąpieniu każdej litery inną, znajdującą się w alfabecie n pozycji dalej. Przesunięcie n może być dodatnie lub ujemne, przy czym przyjmujemy, że za literą z wypada litera a , zaś przed literą a wypada litera z . Małe litery są kodowane jako małe, a duże jako duże. Inne znaki nie są szyfrowane. Napisz program `caesar`, który przyjmuje jako argumenty wywołania przesunięcie oraz nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego kodując ją szyfrem Cezara z zadaniem przesunięciem. Program załącza tylko pliki nagłówkowe `cstdlib` i `fstream`.

Przykładowy plik wejściowy `input.txt`

```
Bwf Dbftbs!  
Wfoj, wjej, wjdj!
```

Przykładowe wywołanie

```
Linux: ./caesar -1 input.txt output.txt  
Windows: caesar.exe -1 input.txt output.txt
```

Przykładowy plik wyjściowy `output.txt`

```
Ave Caesar!  
Veni, vidi, vici!
```

7.2.12 LF: Konwersja znaków końca linii

W systemie Linux koniec linii oznacza się pojedynczym znakiem *line feed*, czyli `\n`, zaś w systemie Windows używa się pary *carriage return - line feed*, czyli `\r\n`. Napisz program `lf`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zastępując pojedyncze znaki `\n` parami `\r\n` i na odwrót. Program załącza tylko plik nagłówkowy `fstream`.

Przykładowy plik wejściowy `input.txt`

```
Ala ma kota.[CR][LF]  
To kot Ali.[LF]
```

Przykładowe wykonanie

```
Linux: ./lf input.txt output.txt  
Windows: lf.exe input.txt output.txt
```

Przykładowy plik wyjściowy `output.txt`

```
Ala ma kota.[LF]  
To kot Ali.[CR][LF]
```

7.2.13 XOR: Szyfrowanie bitową różnicą symetryczną

Znak można zaszyfrować biorąc bitową różnicę symetryczną jego kodu zadaną liczbą całkowitą. Biorąc różnicę symetryczną uzyskanej wartości z tą samą liczbą odzyskuje się wyjściowy znak. Napisz program `xor` szyfrujący i odczytujący w ten sposób wiadomość tekstową. Program przyjmuje jako argumenty wywołania liczbę całkowitą oraz nazwy dwóch plików tekstowych. Jeżeli podana liczba jest dodatnia, program czyta z pierwszego pliku znaki, bierze ich różnicę symetryczną z podaną liczbą i uzyskane wartości zapisuje do drugiego pliku jako liczby, jak w przykładzie poniżej. W przeciwnym razie program czyta z pierwszego pliku liczby, bierze ich różnicę symetryczną z liczbą przeciwną do podanej i uzyskane wartości zapisuje do drugiego pliku jako znaki. Program załącza tylko pliki nagłówkowe `cstdlib` i `fstream`.

Przykładowy plik wejściowy source.txt

October 2018

Przykładowe wywołanie

Linux: `./xor 157 source.txt target.txt`
Windows: `xor.exe 157 source.txt target.txt`

Przykładowy plik wyjściowy target.txt

210 254 233 242 255 248 239 189 175 173 172 165

7.2.14 Spacer: Zamiana tabulatorów na spacje

Napisz program `spacer`, który przyjmuje jako argumenty wywołania liczbę spacji w tabulatorze oraz nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zastępując wszystkie tabulatory spacjami tak, aby w edytorze tekstu oba pliki wyglądały tak samo. Program łączy tylko pliki nagłówkowe `cstdlib` i `fstream`.

7.2.15 Tabulator: Zamiana spacji na tabulatory

Napisz program `tabulator`, który przyjmuje jako argumenty wywołania liczbę spacji w tabulatorze oraz nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zastępując jak najwięcej spacji tabulatorami tak, aby w edytorze tekstu oba pliki wyglądały tak samo. Program łączy tylko pliki nagłówkowe `cstdlib` i `fstream`.

7.2.16 Camel: Kamelizacja wyrazów

Napisz program `camel`, który przyjmuje jako argumenty wywołania nazwy dwóch plików tekstowych i przepisuje zawartość pierwszego pliku do drugiego zamieniając pierwszą literę każdego słowa na wielką, a następne litery na małe. Pozostałe znaki program przepisuje bez zmian. Program łączy tylko pliki nagłówkowe `cctype` i `fstream`.

Przykładowy plik wejściowy input.txt

AlA mA 15 LAT!
czy ala ma 17 lat?

Przykładowe wywołanie

Linux: `./camel input.txt output.txt`
Windows: `camel.exe input.txt output.txt`

Przykładowy plik wyjściowy output.txt

Ala Ma 15 Lat!
Czy Ala Ma 17 Lat?

8 Łącuchy: 21 i 22 kwietnia
Łącuchy tekstowe, Strumienie łańcuchowe

- 9 Gry: 28 i 29 kwietnia
Prosta gra w trybie tekstowym

10 Sprawdzian: 5 i 6 maja

11 Iteratory: 12 i 13 maja
Iteratory wektora

12 Lambdy: 19 i 20 maja
Funkcje wyższego rzędu, Wyrażenia lambda

13 Algorytmy: 26 i 27 maja
Algorytmy biblioteki standardowej

14 Pojemniki: 2 i 3 czerwca
Pojemniki biblioteki standardowej

15 Kolokwium: 9 i 10 czerwca