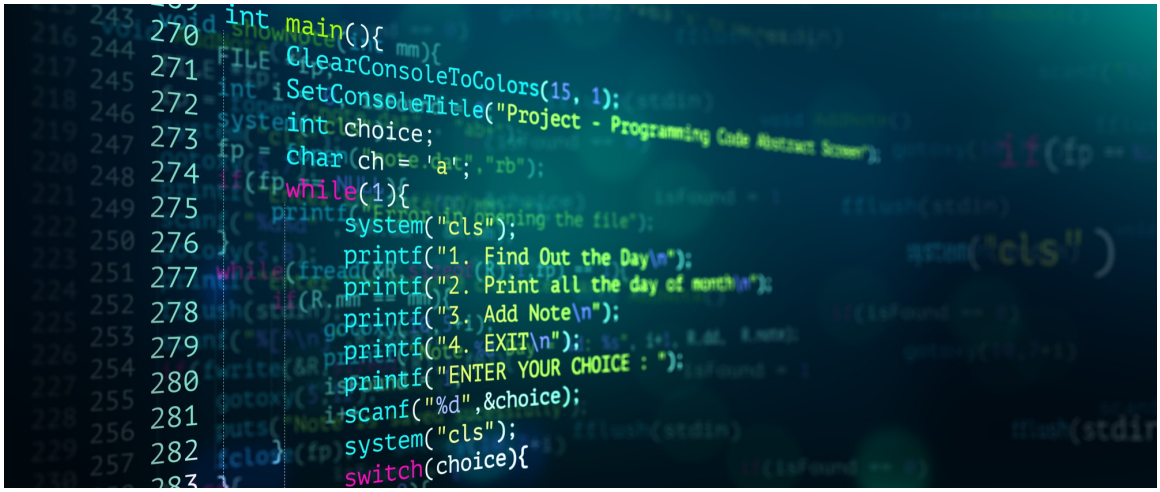

Zbiór zadań - C

Piotr Jastrzębski

2025-04-28

Spis treści

Zbiór zadań - C

Tu będzie zbiór zadań z programowania w języku C. Inspiracją było zebranie zadań powstałych w trakcie prowadzenia zajęć dydaktycznych realizowanych na Wydziale Matematyki i Informatyki Uniwersytetu Warmińsko-Mazurskiego w Olsztynie.

Rozwiązania wybranych zadań dostępne są tutaj: https://github.com/pjastr/zbior-zadan-c-rozw.

Część I

Podstawy języka C

1 Operacje wejścia, wyjścia.

1. Napisz program, który prosi użytkownika o wprowadzenie jednej liczby całkowitej, a następnie wyświetla ją na ekranie.
2. Stwórz program, który wczytuje od użytkownika dwie liczby zmiennoprzecinkowe i wypisuje ich różnicę.
3. Zaprojektuj aplikację, która pyta użytkownika o jego rok urodzenia, a następnie wypisuje rok poprzedni (o jeden mniejszy).
4. Napisz program, który wczytuje od użytkownika trzy liczby całkowite i wypisuje ich średnią jako wartość zmiennoprzecinkową.
5. Utwórz program, który prosi użytkownika o wprowadzenie dwóch liter (znaków), a następnie wypisuje je w odwrotnej kolejności.
6. Napisz program, który prosi użytkownika o wprowadzenie jednej liczby zmiennoprzecinkowej, a następnie podwaja jej wartość i wypisuje wynik.
7. Utwórz program, który wczytuje od użytkownika liczbę zmiennoprzecinkową reprezentującą kwotę w dolarach, a następnie wypisuje, ile to jest euro, przyjmując stały kurs wymiany (np. 1 dolar = 0.85 euro).
8. Napisz program, który wyświetla na ekranie tekst: `To jest cytat: "Często używam języka C."`. Upewnij się, że znaki cudzysłowu są poprawnie wyświetlane jako część napisu.
9. Stwórz program, który wypisuje na standardowe wyjście ścieżkę do folderu w systemie Windows, np. `C:\Program Files\MojaAplikacja\` (na sztywno, bez pobierania czegoś z systemu) lub `C:\\Program Files\\MojaAplikacja\\`.
10. Zaprojektuj program, który pokazuje, jak wypisać na ekranie następujący tekst: `Specjalne znaki: \t (tabulacja), \n (nowa linia), % (procent), \\ (ukośnik wsteczny).`.
11. Napisz program, który wczytuje ze standardowego wejścia dwie liczby wymierne reprezentujące długości boków trójkąta prostokątnego. Następnie oblicz i wyświetl długość przeciwprostokątnej.
12. Napisz program, który wczytuje ze standardowego wejścia całkowitą i wypisuje na standardowym wyjściu jej wartość bezwzględną.
13. Napisz program, który wczytuje ze standardowego wejścia zmiennoprzecinkową i wypisuje na standardowym wyjściu jej wartość bezwzględną.
14. Znajdź przykład i wyświetl na standardowym wyjściu, kiedy dodawanie liczb zmiennoprzecinowych nie jest łączne.

2 Operatory arytmetyczne

1. Napisz program, który oblicza resztę z dzielenia sumy dwóch liczb całkowitych przez trzecią liczbę całkowitą.
2. Stwórz program, który oblicza różnicę kwadratów dwóch podanych liczb całkowitych.
3. Opracuj program, który oblicza iloczyn różnicy dwóch liczb całkowitych i trzeciej liczby całkowitej.
4. Zaprojektuj program, który oblicza średnią geometryczną bezwzględnych wartości trzech podanych liczb całkowitych.
5. Napisz program, który oblicza kwadrat sumy dwóch podanych liczb całkowitych.
6. Stwórz program, który oblicza sumę kwadratów trzech podanych liczb całkowitych.
7. Opracuj program, który oblicza, ile razy jedna podana liczba całkowita mieści się w drugiej podanej liczbie całkowitej.
8. Zaprojektuj program, który oblicza kwadrat różnicy dwóch podanych liczb całkowitych.
9. Napisz program, który oblicza iloraz sumy dwóch liczb całkowitych przez ich różnicę.
10. Stwórz program, który oblicza sumę trzech kolejnych liczb całkowitych, zaczynając od podanej liczby całkowitej.
11. Znajdź średnią arytmetyczną trzech liczb zmiennoprzecinkowych.
12. Oblicz wartość wyrażenia $\frac{1}{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}$, gdzie a, b, c są różne od zera.
13. Wylicz wartość e^x dla małych wartości x np. $\|x\| < 0.01$ przybliżając e^x jako $1 + x$, bez użycia funkcji eksponencjalnych.
14. Oblicz pole trójkąta o bokach a, b i kącie między nimi C w stopniach, używając wzoru $0.5 \cdot a \cdot b \cdot \sin C$, przyjmując $\sin C \approx C$ dla małych kątów w radianach.
15. Oblicz $\sqrt[3]{x}$ dla małych wartości x np. $\|x\| < 0.01$ przybliżając $\sqrt[3]{x} \approx 1 + \frac{1}{3}x$, bez użycia funkcji pierwiastkowania.

3 Instrukcje warunkowe, operator warunkowy

W poniższych zadaniach jako instrukcję warunkową należy wykorzystać `if`, `if else`, `switch case`. Przez operator warunkowy rozumie się `?`.

1. Napisz program, który prosi użytkownika o wprowadzenie liczby całkowitej. Program powinien wyświetlić informację, czy wprowadzona liczba jest dodatnia, ujemna czy równa zero.
2. Napisz program, który przyjmuje od użytkownika dwie liczby całkowite i wyświetla większą z nich.
3. Napisz program, który prosi o wprowadzenie oceny w skali od 1 do 5. Program powinien wyświetlić opis oceny: niedostateczny (1), dopuszczający (2), dostateczny (3), dobry (4), bardzo dobry (5). Dla liczby spoza zakresu, program powinien wyświetlić komunikat o błędzie.
4. Napisz program, który prosi użytkownika o wprowadzenie trzech różnych liczb całkowitych i wyświetla najmniejszą z nich.
5. Napisz program, który pyta użytkownika o rok i sprawdza, czy podany rok jest rokiem przestępnym. Rok przestępny to taki, który jest podzielny przez 4, ale nie jest podzielny przez 100, chyba że jest też podzielny przez 400.
6. Napisz program, który przyjmuje od użytkownika dwie liczby całkowite i wyświetla informację, czy suma obu liczb jest parzysta czy nieparzysta.
7. Napisz program, który przyjmuje od użytkownika trzy liczby zmiennoprzecinkowe `a, b, c`. Potrakuj je jako współczynniki równania kwadratowego $ax^2 + bx + c = 0$. Na standardowym wyjściu wypisz wszystkie warianty rozwiązań tego równania.
8. Napisz program, który przyjmuje dwie liczby całkowite jako wejście od użytkownika i używa operatora warunkowego, aby znaleźć i wyświetlić największą z nich.
9. Napisz program, który przyjmuje trzy liczby całkowite jako wejście od użytkownika i używa operatora warunkowego, aby znaleźć i wyświetlić najmniejszą z nich.
10. Używając operatora warunkowego `?`, napisz program, który przyjmuje od użytkownika jedną liczbę całkowitą i wyświetla "parzysta" lub "nieparzysta" w zależności od wartości liczby.

11. Stwórz program, który prosi o wprowadzenie dwóch liczb zmiennoprzecinkowych i używa operatora warunkowego, aby wyświetlić, która z nich jest większa, lub czy są równe z dokładnością do dwóch miejsc po przecinku.

12. Używając operatora warunkowego, napisz program, który prosi użytkownika o wprowadzenie oceny w skali od 0 do 100 i wyświetla "Zdane", jeśli ocena jest większa lub równa 51, lub "Nie zdane" w przeciwnym przypadku.

13. Napisz program, który przyjmuje rok jako wejście od użytkownika i za pomocą operatora warunkowego sprawdza, czy jest to rok przestępny. Program powinien wyświetlać "Rok przestępny" lub "Rok nieprzestępny" w zależności od wyniku.

4 Operatory bitowe

1. Zamień wartości dwóch zmiennych całkowitych bez użycia dodatkowej zmiennej.
2. Sprawdź, czy liczba całkowita jest parzysta czy nieparzysta za pomocą operatorów bitowych.
3. Wyznacz wartość bitu na określonej pozycji w liczbie całkowitej.
4. Zeruj wartość bitu na określonej pozycji w liczbie całkowitej.
5. Odwróć wartość wszystkich bitów w liczbie całkowitej.
6. Przesuń bity liczby całkowitej o określoną liczbę pozycji w lewo.
7. Przesuń bity liczby całkowitej o określoną liczbę pozycji w prawo.
8. Wyznacz "XOR" dwóch liczb całkowitych.

5 Debugowanie - podstawy

1. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>

int main() {
    int a = 10; // a = , b =
    int b = 5;  // a = , b =
    a = a + b;  // a = , b =
    b = a - b;  // a = , b =
    a = a - b;  // a = , b =
    b = a * b;  // a = , b =
    a = b / a;  // a = , b =
    b = a << 2; // a = , b =
    a = b >> 1; // a = , b =
    b = a & b;  // a = , b =
    a = a ^ b;  // a = , b =
    b = ~a;     // a = , b =
    return 0;
}
```

2. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>

int main() {
    int a = 10; // a = , b = , c =
    int b = 5;  // a = , b = , c =
    int c = 0;  // a = , b = , c =
    a = a + b;  // a = , b = , c =
```

```
    b = a - b;  // a = , b = , c =
    a = a - b;  // a = , b = , c =
    c = a;      // a = , b = , c =
    a = b * c;  // a = , b = , c =
    b = a / c;  // a = , b = , c =
    c = b << 2; // a = , b = , c =
    b = c >> 1; // a = , b = , c =
    a = b & c;  // a = , b = , c =
    c = a ^ b;  // a = , b = , c =
    b = ~c;     // a = , b = , c =
    return 0;
}
```

3. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```
#include <stdio.h>
int main() {
    int x = 7;    // x = , y = , z =
    int y = 3;    // x = , y = , z =
    int z = 12;   // x = , y = , z =
    x = x * 2;    // x = , y = , z =
    y = y + x;    // x = , y = , z =
    z = z - y;    // x = , y = , z =
    x = z / 2;    // x = , y = , z =
    y = x % 3;    // x = , y = , z =
    z = x + y;    // x = , y = , z =
    x = z++;      // x = , y = , z =
    y = ++x;      // x = , y = , z =
    z = y--;      // x = , y = , z =
    return 0;
}
```

4. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>
int main() {
    int p = 15;   // p = , q = , r =
    int q = 6;    // p = , q = , r =
    int r = 1;    // p = , q = , r =
    p = p | q;    // p = , q = , r =
    q = p & 10;   // p = , q = , r =
    r = r << 3;   // p = , q = , r =
    p = p ^ r;    // p = , q = , r =
    q = q >> 1;   // p = , q = , r =
    r = ~p;       // p = , q = , r =
    p = q | r;    // p = , q = , r =
    q = p & r;    // p = , q = , r =
    r = r ^ q;    // p = , q = , r =
    return 0;
}
```

5. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>
int main() {
    float a = 10.5;  // a = , b = , c =
    float b = 2.0;   // a = , b = , c =
    float c = 0.5;   // a = , b = , c =
    a = a / b;       // a = , b = , c =
    b = b * 3;       // a = , b = , c =
    c = a + b;       // a = , b = , c =
    a = c - a;       // a = , b = , c =
    b = b / 2;       // a = , b = , c =
    c = a * b;       // a = , b = , c =
    a = c + 0.5;     // a = , b = , c =
    b = a - c;       // a = , b = , c =
    c = a / b;       // a = , b = , c =
    return 0;
}
```

6. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>
int main() {
    int m = 20;    // m = , n = , k =
    int n = 4;     // m = , n = , k =
    int k = 2;     // m = , n = , k =
    m += n;        // m = , n = , k =
    n -= k;        // m = , n = , k =
    k *= 3;        // m = , n = , k =
    m /= 2;        // m = , n = , k =
    n += m;        // m = , n = , k =
    k %= n;        // m = , n = , k =
    m <<= 1;       // m = , n = , k =
    n >>= 2;       // m = , n = , k =
    k ^= m;        // m = , n = , k =
    return 0;
}
```

6 Pętle

Zadania należy rozwiązać bez tablic, napisów, wskaźników, wbudowanych funkcji matematycznych. Nie twórz samodzielnie też własnych funkcji.

1. Napisz program, który wyświetla wszystkie liczby całkowite od 1 do 100, używając pętli `for`.
2. Utwórz program, który prosi użytkownika o wprowadzenie liczby całkowitej `n`, a następnie wyświetla sumę wszystkich liczb całkowitych od 1 do `n` używając pętli `while`.
3. Napisz program, który czyta od użytkownika liczby całkowite do momentu wprowadzenia zera i następnie wyświetla sumę wszystkich wprowadzonych liczb pozytywnych oraz sumę wszystkich liczb negatywnych, używając pętli `do-while`.
4. Stwórz program, który oblicza i wyświetla silnię podanej przez użytkownika nieujemnej liczby całkowitej, używając pętli `for`.
5. Napisz program, który wyświetla pierwszych 10 liczb ciągu Fibonacciego, używając pętli `while`.
6. Napisz program, który prosi użytkownika o wprowadzenie dodatniej liczby całkowitej `n`, a następnie oblicza i wyświetla $\lfloor \sqrt{n} \rfloor$ (część całkowita/podłoga pierwiastka kwadratowego).
7. Napisz program, który prosi użytkownika o wprowadzenie dodatniej liczby całkowitej `n`, a następnie oblicza i wyświetla $\lceil \sqrt{n} \rceil$ (sufit pierwiastka kwadratowego).
8. Napisz program, który prosi użytkownika o wprowadzenie 10 dodatnich liczb całkowitych i ustawia je w ciąg $a_1, \ldots, a_{10}$. Oblicz i wyświetl ile elementów ciągu spełnia nierówność $a_k < \frac{a_{k-1}+a_{k+1}}{2}$ dla $1 < k < 10$.
9. Napisz program, który prosi użytkownika o wprowadzenie dodatniej liczby całkowitej n, a następnie n dodatnich liczb całkowitych i ustawia je w ciąg $a_1, \ldots, a_n$. Oblicz i wyświetl ile elementów ciągu spełnia nierówność $a_k < \frac{a_{k-1}+a_{k+1}}{2}$ dla $1 < k < n$.
10. Napisz program, który sprawdza podzielność liczby n przez wszystkie liczby od 2 do n/2. Program powinien wypisać wszystkie dzielniki tej liczby. Wczytaj wartość n od użytkownika.
11. Napisz program, który wczytuje liczbę całkowitą n i wypisuje wszystkie jej cyfry od końca (od cyfry jedności) oraz oblicza ich sumę.

12. Napisz program realizujący algorytm Euklidesa w wersji iteracyjnej z odejmowaniem. Program powinien wczytać dwie liczby naturalne a i b, a następnie wypisać ich największy wspólny dzielnik.

13. Napisz program realizujący algorytm Euklidesa w wersji iteracyjnej z dzieleniem. Program powinien wczytać dwie liczby naturalne a i b, a następnie wypisać ich największy wspólny dzielnik.

14. Napisz program, który sprawdza, czy dana liczba n jest liczbą pierwszą. Program powinien sprawdzić wszystkie potencjalne dzielniki od 2 do pierwiastka z n (wykorzystaj pętlę do obliczenia pierwiastka). Wypisz odpowiedni komunikat.

15. Napisz program symulujący wydawanie reszty. Program powinien wczytać kwotę do wydania i wypisać, ile monet o nominałach 5, 2 i 1 należy wydać, aby użyć jak najmniejszej liczby monet. Wypisz liczbę monet każdego typu.

16. Napisz program, który wyznacza przybliżoną wartość pierwiastka kwadratowego z liczby a metodą Newtona (iteracyjną). Początkowe przybliżenie `x = a/2`, a kolejne obliczamy ze wzoru `x_{i+1} = (x_i + a/x_i)/2`. Wykonaj 10 iteracji algorytmu i wypisz końcowy wynik.

17. Napisz program, który wczytuje od użytkownika stopień wielomianu, współczynniki (od najwyższej potęgi do wyrazu wolnego) oraz wartość x. Oblicz wartość wielomianu za pomocą schematu Hornera, wykorzystując pętlę. Wyświetl wynik obliczeń.

7 Funkcje

Zadania należy rozwiązać bez tablic, napisów, wskaźników, wbudowanych funkcji matematycznych. W zadaniach można stworzyć kilka funkcji pomocniczych. Do każdej funkcji z polecenia należy wywołać przypadek testowy (wywołać co najmniej jeden raz na poprawnym argumencie). W poleceniach używana jest konwencja `camelCase`. Zwróć uwagę, czy funkcja ma coś zwrócić czy wyświetlić.

1. Napisz funkcję `sumTwoNumbers`, której argumentami są dwie liczby całkowite. Funkcja ma wyświetlać sumę liczb przekazany jako argument funkcji. Stwórz przypadek testowy.
2. Napisz funkcję `calculateAbsoluteValue`, której argumentem jest liczba zmiennoprzecinkowa. Funkcja ma zwracać wartość bezwzględną liczby przekazanej jako argument funkcji. Stwórz przypadek testowy.
3. Napisz funkcję `calculateFactorial`, której argumentem jest liczba całkowita nieujemna. Funkcja ma zwracać wartość silni liczby przekazanej jako argument funkcji, obliczoną metodą nierekurencyjną. Stwórz przypadek testowy.
4. Napisz funkcję `sumNumbers`, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwracać sumę liczb od 1 do n włącznie. Stwórz przypadek testowy.
5. Napisz funkcję `sumSquares`, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwracać sumę kwadratów liczb od 1 do n włącznie. Stwórz przypadek testowy.
6. Napisz funkcję `calculatePowerOfTwo`, której argumentem jest liczba całkowita n. Funkcja ma zwracać wartość 2^n. Stwórz przypadek testowy.
7. Napisz funkcję `calculateSquareRootFloor`, której argumentem jest nieujemna liczba całkowita n. Funkcja ma zwracać część całkowitą pierwiastka kwadratowego z n. Stwórz przypadek testowy.
8. Napisz funkcję `countFunctionCalls`, która nie przyjmuje żadnych argumentów. Funkcja ma zliczać i wypisywać na standardowym wyjściu liczbę swoich wywołań od momentu uruchomienia programu. Stwórz przypadek testowy. Wykorzystaj zmienne statyczne.
9. Napisz funkcję `calculateFactorialRecursively`, której argumentem jest liczba całkowita nieujemna n. Funkcja ma zwracać wartość silni liczby n, obliczoną metodą rekurencyjną. Stwórz przypadek testowy.

10. Napisz funkcję `calculateFibonacciRecursively`, której argumentem jest liczba całkowita nieujemna n. Funkcja ma zwracać n-ty wyraz ciągu Fibonacciego, obliczony metodą rekurencyjną. Stwórz przypadek testowy.

11. Napisz funkcję `calculateArithmeticSequenceRecursively`, której argumentami są dwie liczby całkowite: dodatnie n (numer wyrazu ciągu do obliczenia) oraz d (różnica ciągu arytmetycznego), przy założeniu, że wyraz początkowy ciągu a_1 wynosi 1. Funkcja ma zwracać n-ty wyraz ciągu arytmetycznego, obliczony metodą rekurencyjną. Stwórz przypadek testowy.

12. Napisz funkcję `calculateGeometricSequenceRecursively`, której argumentami są dwie liczby całkowite: dodatnie n (numer wyrazu ciągu do obliczenia) oraz d (iloraz ciągu geometrycznego), przy założeniu, że wyraz początkowy ciągu a_1 wynosi 1. Funkcja ma zwracać n-ty wyraz ciągu geometrycznego, obliczony metodą rekurencyjną. Stwórz przypadek testowy.

13. Napisz funkcję `calculate13`, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwracać wartość wyrażoną wzorem $f(n) = 2f(n-1) + 3$, gdzie $f(1) = 1$. Stwórz przypadek testowy.

14. Napisz funkcję `calculate14`, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwracać wartość wyrażoną wzorem $f(n) = 3f(n-1) - 1$, gdzie $f(1) = 2$. Stwórz przypadek testowy.

15. Napisz funkcję `calculate15`, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwracać wartość wyrażoną wzorem $f(n) = f(n-1) + 2f(n-2)$, gdzie $f(1) = 1$ i $f(2) = 2$. Stwórz przypadek testowy.

16. Napisz funkcję `calculate16`, której argumentem jest dodania liczba całkowita n. Funkcja ma zwracać wartość wyrażoną wzorem $f(n) = 2f(n-1) + 3f(n-2)$, gdzie $f(1) = 2$ i $f(2) = 3$. Stwórz przypadek testowy.

17. Napisz rekurencyjną funkcję `calculate17`, której argumentem jest nieujemna liczba całkowita n. Funkcja ma zwracać wartość wyrażoną wzorem $f(n) = f(0)+f(1)+...+f(n-1)$, gdzie $f(0) = f(1) = 1$. Stwórz przypadek testowy.

18. Napisz rekrurencyjną funkcję `calculateGCD`, której argumentami są dwie dodatnie liczby całkowite n i m. Funkcja ma zwracać największy wspólny dzielnik (NWD) tych liczb algorytmem Euklidesa. Stwórz przypadek testowy.

8 Wskaźniki

Zadania należy rozwiązać bez interpretacji wskaźników jako tablic.

1. Skopiuj lub przepisz kod i sprawdź wyniki na standardowym wyjściu:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("%Iu\n", sizeof(int));
    printf("%Iu\n", sizeof(int*));
    return 0;
}
```

2. W jednym pliku wykonaj czynności:

- Napisz funkcję `sum` z dwoma argumentami typu `int`. Funkcja ma zwracać sumę wartości przekazanych przez argumenty. Stwórz przypadek testowy.
- Napisz funkcję `sumVals` z dwoma argumentami, które są wskaźnikami do zmiennych typu `int`. Funkcja powinna zwracać sumę wartości, na które wskazują te argumenty. Utwórz przypadek testowy.
- Napisz funkcję `addPtr` z trzema argumentami, które są wskaźnikami do zmiennych typu `int`. Funkcja ma być procedurą (typ zwracany `void`). Funkcja ma ustawić wartość zmiennej wskazywanej przez trzeci argument funkcji jako sumę wartości wskazywanych przez dwa wcześniejsze argumenty. Utwórz przypadek testowy.

3. Napisz funkcję `copyInt` z argumentami: `x` typu `int` oraz `w`, który jest wskaźnikiem do `int`. Funkcja ma przepisać wartość `x` do zmiennej wskazywanej przez `w`. Stwórz przypadek testowy.
4. Napisz funkcję `findMax` z dwoma argumentami: wskaźnikiem `num1` na stałą wartość typu `double` i stałym wskaźnikiem `num2` na zmienną typu `double`. Funkcja ma zwracać większą wartość spośród tych, na które wskazują `num1` i `num2`. Utwórz przypadek testowy.

5. Napisz funkcję `initInts`, która nie przyjmuje argumentów i rezerwuje blok trzech zmiennych typu `int`, ustawiając ich wartości kolejno na 5, -12, 33. Funkcja zwraca wskaźnik na środkową zmienną. Utwórz przypadek testowy w funkcji `main`, który wyświetla wartości z bloku stworzonego przez funkcję.

6. Napisz funkcję `initFloats`, która nie przyjmuje argumentów i rezerwuje blok trzech zmiennych typu `float`, ustawiając ich wartości kolejno na 4.5, 2.3, -4.2. Funkcja zwraca wskaźnik na początkową ze zmiennych w bloku. Utwórz przypadek testowy w funkcji `main`, który wyświetla wartości z bloku stworzonego przez funkcję.

7. Napisz funkcję `initFlts`, która nie przyjmuje argumentów i rezerwuje blok czterech zmiennych typu `float`, ustawiając ich wartości kolejno na 0.5, 1.5, 2.5, i 3.5. Funkcja zwraca wskaźnik na ostatnią zmienną w bloku. Utwórz przypadek testowy w `main`, aby wyświetlić wartości z bloku stworzonego przez funkcję.

8. Napisz funkcję `sumToPtr` z trzema argumentami: dwoma wskaźnikami na stałe typu `int` i wskaźnikiem na zmienną typu `int`. Funkcja ma przepisać do zmiennej wskazywanej przez trzeci argument sumę wartości stałych wskazywanych przez pierwszy i drugi argument. Utwórz przypadek testowy.

9. Napisz funkcję `sqrCopy` z dwoma argumentami: wskaźnikiem na stałą typu `int` i wskaźnikiem na zmienną typu `int`. Funkcja ma przepisać kwadrat wartości stałej do zmiennej wskazywanej przez drugi argument. Utwórz przypadek testowy.

10. Napisz funkcję `subPtrs` z dwoma argumentami: wskaźnikiem `num1` na stałą wartość typu `double` i stałym wskaźnikiem `num2` na zmienną typu `double`. Funkcja ma zwracać różnicę wartości, na które wskazują `num1` i `num2`. Utwórz przypadek testowy.

11. Napisz funkcję `sumSqrs` z dwoma argumentami: wskaźnikiem `num1` na stałą wartość typu `int` i stałym wskaźnikiem `num2` na zmienną typu `int`. Funkcja ma zwracać sumę kwadratów wartości wskazywanych przez `num1` i `num2`. Utwórz przypadek testowy.

12. Napisz funkcję `linFuncVal` z trzema argumentami: wskaźnikiem `a` na stałą wartość typu `float`, stałym wskaźnikiem `b` na zmienną typu `float`, i wskaźnikiem `x` na stałą wartość typu `float`. Funkcja ma obliczać i zwracać wartość funkcji liniowej `y=ax+b` dla argumentu `x`, gdzie `a` i `b` są wskazywane przez odpowiednie wskaźniki. Utwórz przypadek testowy.

13. Napisz funkcję `minPtr` z trzema argumentami, które są wskaźnikami na zmienne typu `int`. Funkcja zwraca wskaźnik na zmienną o najmniejszej wartości spośród tych, na które wskazują argumenty. Utwórz przypadek testowy w `main`, aby wyświetlić najmniejszą wartość spośród trzech zmiennych.

14. Napisz funkcję `multPtrs` z dwoma argumentami: wskaźnikiem `num1` na stałą wartość typu `double` i stałym wskaźnikiem `num2` na zmienną typu `double`. Funkcja zwraca iloczyn wartości wskazywanych przez te wskaźniki. Utwórz przypadek testowy.

15. Napisz funkcję `absVal` z jednym argumentem, którym jest wskaźnik na zmienną typu `int`. Funkcja oblicza wartość bezwzględną zmiennej wskazywanej przez wskaźnik i aktualizuje tę zmienną. Utwórz przypadek testowy w `main`, aby wyświetlić wartość zaktualizowanej zmiennej.

16. Napisz funkcję `swap` z dwoma argumentami: wskaźnikiem `ptr1` na zmienną typu `int` i wskaźnikiem `ptr2` na inną zmienną tego samego typu. Funkcja zamienia miejscami wartości wskazywane przez wskaźniki. Utwórz przypadek testowy.

17. Napisz funkcję `swapSign` z dwoma argumentami: wskaźnikiem `ptr1` na zmienną typu `double` i wskaźnikiem `ptr2` na inną zmienną tego samego typu. Funkcja zamienia miejscami wartości wskazywane przez wskaźniki, jeśli mają one różne znaki. W przeciwnym razie nie robi nic. Utwórz przypadek testowy.

9 Wskaźniki na funkcję

1. Napisz funkcję `calculate`, która przyjmuje dwa argumenty: wskaźnik na funkcję `operation` oraz liczbę całkowitą `number`. Funkcja `operation` ma przyjmować jeden argument typu `int` i zwracać wartość typu `int`. Funkcja `calculate` powinna wywołać funkcję `operation` z argumentem `number` i zwrócić jej wynik. Stwórz przypadek testowy.

2. Napisz funkcję o nazwie `applyFunction`, która przyjmuje trzy argumenty: wskaźnik na funkcję `func`, która przyjmuje jeden argument typu `int` i zwraca `int`, oraz dwie liczby całkowite: `start` i `end`. Funkcja `applyFunction` powinna wywołać funkcję `func` dla każdej liczby w zakresie od `start` do `end` (włącznie) i wydrukować wyniki na standardowe wyjście. Stwórz przypadek testowy.

3. Napisz funkcję, która otrzymuje trzy argumenty:

- dwa wskaźniki na funkcje z jednym argumentem typu `int` zwracające wartość typu `int`,
- dodatnią wartość `n` typu `int`,

i zwraca 1, jeżeli otrzymane w argumentach funkcje mają ten sam znak dla wartości dla liczb całkowitych od 0 do n, a zwraca 0 w przeciwnym wypadku. Stwórz przypadek testowy.

4. Napisz funkcję `calculateOperation`, która przyjmuje jako argumenty: wskaźnik na funkcję `operation`, która przyjmuje dwa argumenty typu `double` i zwraca `double`, oraz dwa argumenty typu `double` - `number1` i `number2`. Funkcja `calculateOperation` ma zwracać wynik wywołania funkcji `operation` na argumentach `number1` i `number2`. Stwórz przypadek testowy.

5. Napisz funkcję `modifyAndSum`, która ma przyjmować jako argument wskaźnik na funkcję `modifier`, która przyjmuje jeden argument typu `int` i zwraca `int`, oraz dwa argumenty typu `int`: `number1` i `number2`. Funkcja `modifyAndSum` powinna modyfikować obie liczby za pomocą funkcji `modifier` i zwracać ich sumę. Przykładem funkcji `modifier` może być funkcja, która zwiększa liczbę o 1 lub zmienia znak liczby. Stwórz przypadek testowy.

6. Stwórz funkcję `applyCondition`, która przyjmuje trzy argumenty: wskaźnik na funkcję `condition` zwracającą wartość typu `int` i przyjmującą `int`, wskaźnik na funkcję `action` również przyjmującą i zwracającą `int`, oraz wartość całkowitą `value`. Funkcja `applyCondition` powinna najpierw wywołać `condition` z `value` jako argumentem. Jeśli wynik to 1, `applyCondition` powinna następnie wywołać `action` na `value` i zwrócić wynik. W przeciwnym wypadku powinna zwrócić `value` bez zmian. Stwórz przypadek testowy.

7. Stwórz funkcję `executeSequence`, która przyjmuje dwa argumenty: wskaźnik na funkcję `operation` zwracającą `void` i przyjmującą `int` oraz liczbę całkowitą `count`. Funkcja `executeSequence` powinna wywołać funkcję `operation` dokładnie `count` razy, przekazując jej jako argument kolejne liczby od 1 do `count`. Stwórz przypadek testowy.

8. Stwórz funkcję `modifyValue`, która przyjmuje argument typu `int` oraz wskaźnik na funkcję `modifier` nie przyjmującą żadnych argumentów, ale zwracającą wartość typu `int`. Funkcja `modifyValue` powinna dodać do przekazanej wartości wynik wywołania funkcji `modifier` i zwrócić otrzymaną sumę. Stwórz przypadek testowy.

9. Stwórz funkcję `processPair` przyjmującą dwie liczby całkowite oraz wskaźnik na funkcję `processor`, która zwraca `double` i przyjmuje dwa argumenty typu `int`. Funkcja `processPair` powinna wywołać `processor` z przekazanymi liczbami, zaokrąglić wynik w dół do najbliższej liczby całkowitej i zwrócić tę wartość jako `int`. Stwórz przypadek testowy.

10. Stwórz funkcję `transformLoop`, która przyjmuje wskaźnik na funkcję `transform` zwracającą `int` i przyjmującą `int`, wartość początkową `start` oraz liczbę iteracji `iterations`. Funkcja powinna wykonać określoną liczbę iteracji, w każdej iteracji przypisując zmiennej wynikowej wartość zwróconą przez funkcję `transform` wywołaną na wyniku z poprzedniej iteracji (w pierwszej iteracji na wartości `start`). Na koniec funkcja powinna zwrócić finalną wartość. Stwórz przypadek testowy.

11. Stwórz funkcję `computeWithFallback`, która przyjmuje trzy argumenty: wskaźnik na funkcję `primary` przyjmującą dwa parametry typu `int` i zwracającą `int`, wskaźnik na funkcję `backup` o takiej samej sygnaturze oraz dwie wartości całkowite `a` i `b`. Funkcja `computeWithFallback` powinna wywołać funkcję `primary` z argumentami `a` i `b`. Jeśli wynik funkcji `primary` jest mniejszy od zera, funkcja `computeWithFallback` powinna wywołać `backup` z tymi samymi argumentami `a` i `b` i zwrócić wynik tej funkcji. W przeciwnym przypadku (gdy wynik funkcji `primary` jest większy lub równy zero), funkcja `computeWithFallback` powinna zwrócić wynik funkcji `primary`. Funkcja `computeWithFallback` powinna zwracać wartość typu `int`. Stwórz przypadek testowy.

10 Tablice jednowymiarowe

Zadania należy rozwiązać bez konwersji liczb na napisy. W zadaniach nie należy używać podwójnych wskaźników i odpowiednik tablic wielowymiarowych - składnia ma być wykonana na jednym wymiarze. W zadaniach nie należy uwzględniać tablicy o rozmiarze zero, bo argumenty związane z rozmiarem są dodatnie.

1. Napisz funkcję `findMaxValue`, która przyjmuje jako argumenty tablicę liczb całkowitych `numbers` oraz jej rozmiar `size` (dodatnia liczba całkowita). Funkcja powinna przeszukać tablicę i wyświetlić maksymalną wartość znalezioną w tej tablicy. Stwórz przypadek testowy.
2. Napisz funkcję `average`, której argumentem jest dodatnia liczba całkowita n oraz n-elementowa tablica `tab` o elementach typu `int` oraz zwraca średnią arytmetyczną jako liczbę zmiennoprzecinkową. Stwórz przypadek testowy.
3. Napisz funkcję `sumSquares`, której argumentem jest dodatnia liczba całkowita n oraz n-elementowa tablica `tab` o elementach typu `int` oraz zwraca sumę kwadratów elementów tablicy. Stwórz przypadek testowy.
4. Napisz funkcję `copyArr`, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą `n` oraz dwie `n`-elementowe tablice `tab1`, `tab2` o elementach typu int i przepisuje kolejno elementy tablicy `tab1` do tablicy `tab2`. Stwórz przypadek testowy.
5. Napisz funkcję `revCopy`, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą `n` oraz dwie `n`-elementowe tablice `tab1`, `tab2` o elementach typu int i przepisuje elementy tablicy `tab1` do tablicy `tab2` w odwrotnej kolejności. Stwórz przypadek testowy.
6. Napisz funkcję `reverseArr`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ta ma odwrócić kolejność elementów w tablicy `tab` bezpośrednio w niej (w miejscu), nie używając dodatkowej tablicy do przechowywania wyników. Stwórz przypadek testowy.
7. Napisz funkcję `maxValue`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ta ma przeszukać tablicę i zwrócić największą znalezioną w niej wartość. Zakładamy, że tablica nie jest pusta. Stwórz przypadek testowy.

8. Napisz funkcję `minValue`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ta ma przeszukać tablicę i zwrócić najmniejszą znalezioną w niej wartość. Zakładamy, że tablica nie jest pusta. Stwórz przypadek testowy.

9. Napisz funkcję `maxIdx`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ta ma znaleźć i zwrócić indeks największego elementu w tablicy. W przypadku wystąpienia więcej niż jednego elementu o maksymalnej wartości, funkcja powinna zwrócić indeks pierwszego z nich (czyli najmniejszy możliwy indeks). Zakładamy, że tablica nie jest pusta. Stwórz przypadek testowy.

10. Napisz funkcję `minIdxMax`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ta ma przeszukać tablicę i zwrócić indeks najmniejszego elementu w tablicy. W przypadku wystąpienia kilku takich samych najmniejszych elementów, zwróć największy z możliwych indeksów tych elementów. Zakładamy, że tablica nie jest pusta. Stwórz przypadek testowy.

11. Napisz funkcję `shiftLeft`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `float`. Funkcja ta ma przesunąć wszystkie elementy tablicy o jedną pozycję w lewo, a ostatni element w tablicy powinien zostać zastąpiony przez pierwszy element oryginalnej tablicy. Zakładamy, że tablica nie jest pusta. Stwórz przypadek testowy.

12. Napisz funkcję `shiftRight2`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ta ma przesunąć wszystkie elementy tablicy o dwa miejsca w prawo. Elementy, które "wypadną" poza tablicę, powinny pojawić się na jej początku w tej samej kolejności. Stwórz przypadek testowy.

13. Napisz funkcję `shiftLeft`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ma zwrócić wskaźnik do nowo utworzonej dynamicznej tablicy, w której wszystkie elementy zostały przesunięte o jedną pozycję w lewo w stosunku do oryginalnej tablicy, a ostatni element nowej tablicy jest równy początkowemu elementowi oryginalnej tablicy. Stwórz przypadek testowy.

14. Napisz funkcję `shiftRight`, której argumentami są dodatnia liczba całkowita `n` reprezentująca rozmiar tablicy oraz `n`-elementowa tablica `tab` o elementach typu `int`. Funkcja ma zwrócić wskaźnik do nowo utworzonej dynamicznej tablicy, w której wszystkie elementy zostały przesunięte o jedną pozycję w prawo w stosunku do oryginalnej tablicy, a pierwszy element nowej tablicy jest równy ostatniemu elementowi oryginalnej tablicy. Stwórz przypadek testowy.

15. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>
#include <stdlib.h>

int foo(int*a,int*b)
{
    return *a-*b;
}

int main()
{
    int tab[] = {3,3,0,0,3,3,-4,-4,0,1};
    int *wsk=tab-1;
    int b = *(wsk+=4); //b=
    int c = b+3; // b=  , c=
    int d = foo(&b,&c); // b=  , c=  , d=
    int e = (wsk+=-1)[4]; // b=  , c=  , d=  , e=
    e = (d *= 2) + (c += 2); // b=  , c=  , d=  , e=
    c = d + (b+=4); // b=  , c=  , d=  , e=
    e= (--c)-(d++);  // b=  , c=  , d=  , e=
    b = *wsk + e; // b=  , c=  , d=  , e=
    return 0;
}
```

16. Poniżej znajduje się kod w języku C. W niektórych linijkach są komentarze. Twoim zadaniem jest wpisanie wartości odpowiednich zmiennych po wykonaniu konkretnej linii kodu. Dopisanie nowych linijek czy zaburzenie struktury kodu oznacza złe wykonanie polecenia.

```c
#include <stdio.h>
#include <stdlib.h>

int foo(int*a,int*b)
{
    return *a-*b;
}

int main()
{
```

```
    int tab[] = {1,5,5,2,3,3,-4,-4,0,1};
    int *wsk=tab-1;
    int b = *(wsk+=4); //b=
    int c = b+3; // b=  , c=
    int d = foo(&b,&c); // b=  , c=  , d=
    int e = (wsk+=-1)[4]; // b=  , c=  , d=  , e=
    e = (d -= 2) - (c += 2); // b=  , c=  , d=  , e=
    c = d + (b+=4); // b=  , c=  , d=  , e=
    e= (--c)-(d++);  // b=  , c=  , d=  , e=
    b = *wsk + e; // b=  , c=  , d=  , e=
    return 0;
}
```

17. Napisz funkcję o nazwie `findElement`, która przyjmuje jako argumenty tablicę liczb całkowitych, rozmiar tablicy, liczbę całkowitą `val` do znalezienia oraz wskaźnik na funkcję `isEqual`. Funkcja `isEqual` przyjmuje jako argumenty dwie liczby całkowite i zwraca wartość typu 0 lub 1, oznaczającą odpowiednio, czy liczby są równe. Funkcja `findElement` powinna przeszukiwać tablicę w celu znalezienia wartości `val` zgodnie z zasadami określonymi przez funkcję `isEqual`, a następnie zwrócić indeks tego elementu w tablicy (pierwszego napotkanego) lub `-1`, jeśli element nie istnieje. Stwórz przypadek testowy dla funkcji `findElement`.

18. Napisz funkcję `findWithCondition`, która przyjmuje jako argumenty tablicę liczb całkowitych, rozmiar tablicy oraz wskaźnik na funkcję `condition`. Funkcja `condition` przyjmuje jako argument liczbę całkowitą i zwraca wartość typu 0 lub 1, oznaczającą odpowiednio, czy liczba spełnia określony warunek. Funkcja `findWithCondition` powinna przeszukać tablicę w celu znalezienia ostatniej liczby spełniającej warunek określony przez funkcję `condition`, a następnie zwrócić indeks tego elementu w tablicy lub `-1`, jeśli taki element nie istnieje. Stwórz przypadek testowy dla funkcji `findWithCondition`.

19. Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n-elementową tablicę `tab` o elementach typu `int`. Funkcja powinna zwrócić wartość `1`, gdy tablica `tab` jest symetryczna (tzn. czy początkowy element jest równy ostatniemu, kolejny przedostatniemu itd.) oraz zwróćić zero w pozostałych przypadkach. Stwórz przypadek testowy dla funkcji.

20. Napisz funkcję, która otrzymuje trzy argumenty: dodatnią liczbę całkowitą n oraz dwie n-elementowe tablice `tab1` i `tab2` o elementach typu `int`. Funkcja powinna sprawdzić, czy obie tablice są identyczne. Jeśli tak, funkcja powinna zwrócić 1, w przeciwnym razie zwrócić 0. Stwórz przypadek testowy dla funkcji.

21. Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n-elementową tablicę `tab` o elementach typu `int` i zwraca indeks pierwszego wystąpienia

największej wartości bezwzględnej elementów przechowywanych w tablicy `tab`. Stwórz przypadek testowy dla funkcji.

22. Napisz funkcję, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą `n` ($n > 3$) oraz `n`-elementową tablicę `tab` o elementach typu `double` i przesuwa o dwa w lewo wszystkie elementy tablicy (tak, żeby wartość elementu o indeksie `n-1` znalazła się w elemencie o indeksie `n-3`, wartość elementu o indeksie `n-2` znalazła się w elemencie o indeksie `n-4`, zaś wartość elementu o indeksie `0` w elemencie o indeksie `n-2`, a wartość elementu o indeksie `1` w elemencie o indeksie `n-1`). Stwórz przypadek testowy dla funkcji.

23. Napisz funkcję, która otrzymuje cztery argumenty: dodatnią liczbę całkowitą `n`, `n`-elementowe tablice `tab1` i `tab2` oraz `n`-elementową tablicę `tab3` o elementach typu `double`. Funkcja powinna obliczać iloczyn elementów tablic `tab1` i `tab2` o tych samych indeksach i zapisywać wyniki do tablicy `tab3`. Stwórz przypadek testowy dla funkcji.

24. Napisz funkcję `sum_odd_indices`, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n-elementową tablicę `tab` o elementach typu `int`. Funkcja ma zwrócić sumę elementów znajdujących się na nieparzystych indeksach. Stwórz przypadek testowy.

25. Napisz funkcję `count_positive_elements`, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n-elementową tablicę `tab` o elementach typu `int`. Funkcja ma zwrócić liczbę dodatnich elementów w tablicy. Stwórz przypadek testowy.

Przykład. Dla tablicy zawierającej elementy 3,-4,5 powinno być zwrócone 2.

26. Napisz funkcję `double_odd_elements`, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n-elementową tablicę `tab` o elementach typu `int`. Funkcja ma podwoić elementy nieparzyste znajdujące się w tablicy. Stwórz przypadek testowy.

27. Napisz funkcję `increase_by_index`, która otrzymuje dwa argumenty: dodatnią liczbę całkowitą n oraz n-elementową tablicę `tab` o elementach typu `int`. Funkcja ma zwiększyć wartość każdego elementu tablicy o jego indeks. Stwórz przypadek testowy.

11 Napisy

1. Napisz funkcję `length`, która jako argument otrzymuje napis i zwraca jako wartość jego długość. Użyj typu `char` i nie korzystaj ze wbudowanych funkcji poza operacjami wejścia/wyjścia. Stwórz przypadek testowy.
2. Napisz funkcję `countNums`, która przyjmuje jako argument tablicę znaków typu `char` i zwraca liczbę znaków w tej tablicy, które są cyframi. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla swojej funkcji.
3. Napisz funkcję `countVow` przyjmującą jako argument tablicę znaków typu `char` i zwracającą liczbę znaków w tej tablicy, które są samogłoskami z alfabetu łacińskiego. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.
4. Napisz funkcję `cmpStrNew` przyjmującą dwa argumenty typu `char[]` (tablice znaków) i zwracającą 1, jeśli napisy są identyczne, oraz 0 w pozostałych przypadkach. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.
5. Napisz funkcję `lexComp` przyjmującą dwa argumenty typu `char[]` (tablice znaków) i zwracającą 1, jeśli pierwszy napis jest wcześniej w porządku leksykograficznym niż drugi, oraz 0 w pozostałych przypadkach. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.
6. Napisz funkcję `toLowerNew`, która przyjmuje jako argument tablicę znaków typu `char` i zamienia w niej wszystkie duże litery na małe. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.
7. Napisz funkcję `toUpperNew`, która przyjmuje jako argument tablicę znaków typu `char` i zamienia w niej wszystkie małe litery na duże. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.
8. Napisz funkcję `strCopyNew`, która otrzymuje dwa argumenty typu `char[]` (tablice znaków): źródłową i docelową. Funkcja przepisuje napis znajdujący się w tablicy źródłowej do tablicy docelowej. Zakładamy, że w tablicy docelowej jest wystarczająco dużo miejsca. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

9. Napisz funkcję `strNcopy`, która otrzymuje trzy argumenty: dwie tablice znaków `nap1`, `nap2` oraz dodatnią liczbę całkowitą `n`. Funkcja przekopiuje n pierwszych znaków z napisu przechowywanego w tablicy `nap1` do tablicy `nap2`. Jeśli napis w tablicy `nap1` jest krótszy niż `n` znaków, funkcja przepisuje cały napis. Funkcja gwarantuje, że na końcu napisu w tablicy `nap2` znajdzie się znak null-terminujący (`'\0'`). Zakładamy, że w tablicy `nap2` jest wystarczająco dużo miejsca. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

10. Napisz funkcję `strConcat`, która otrzymuje jako argumenty trzy tablice znaków: `nap1`, `nap2` oraz `nap3`. Funkcja zapisuje do tablicy `nap3` konkatenację (połączenie) napisów znajdujących się w tablicach `nap1` i `nap2`. Zakładamy, że w tablicy `nap3` jest wystarczająco dużo miejsca. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

11. Napisz funkcję `rmLower`, która przyjmuje jako argument tablicę znaków typu `char` i usuwa z niej wszystkie znaki będące małymi literami. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

12. Napisz funkcję `rmDigits`, która przyjmuje jako argument tablicę znaków typu `char` i usuwa z niej wszystkie znaki będące cyframi. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

13. Napisz funkcję `cutStr`, która otrzymuje jako argumenty tablicę znaków typu `char` oraz dwie liczby całkowite `n` i `m`, i wycina z otrzymanego napisu znaki o indeksach od `n` do `m` (przy założeniu, że `n<m`). Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

14. Napisz funkcję `wStrCopyNew`, która otrzymuje dwa argumenty typu `wchar_t[]` (tablice znaków): źródłową i docelową. Funkcja przepisuje napis znajdujący się w tablicy źródłowej do tablicy docelowej. Zakładamy, że w tablicy docelowej jest wystarczająco dużo miejsca. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

15. Napisz funkcję `wStrConcat`, która otrzymuje jako argumenty trzy tablice znaków: `nap1`, `nap2` oraz `nap3` (użyj typu znakowego `wchar_t`). Funkcja zapisuje do tablicy `nap3` konkatenację (połączenie) napisów znajdujących się w tablicach `nap1` i `nap2`. Zakładamy, że w tablicy `nap3` jest wystarczająco dużo miejsca. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia. Stwórz przypadek testowy dla tej funkcji.

16. Napisz funkcję `w16`, której argumentem jest napis. Jeśli napis zawiera inne znaki niż cyfr, to funkcja ma zwracać zero. Jeśli napis zawiera tylko cyfry, funkcja ma zwrócić liczbę całkowitą powstałą z przepisania kolejno znaków cyfr. Załóż, że napis jest długości dokładnie 3. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy. Wykorzystaj w rozwiązaniu typ `wchar_t`.

17. Napisz funkcję `rmOdd`, której argumentem jest napis. Funkcja ma za zadanie usunąć znaki z napisu o nieparzystych indeksach. Stwórz przypadek testowy.

18. Napisz funkcję `copyEveryThird`, której argumentem jest napis. Funkcja ma za zadanie zwrócić nowy napis, który zawiera tylko co trzeci znak z oryginalnego napisu. Stwórz przypadek testowy.

Przykład: Dla napisu `"ABCXYZ"` funkcja powinna zwrócić `"AX"`.

19. Napisz funkcję `indexLower`, której argumentem jest napis. Funkcja zwraca numer indeksu, na którym występuje pierwsza od lewej mała litera. W przypadku pustego napisu lub braku małych liter, funkcja powinna zwracać zero. W zadaniu nie korzystaj z funkcji bibliotecznych poza instrukcjami wejścia/wyjścia. Stwórz przypadek testowy.

20. Napisz funkcję `countOdds`, której argumentem jest napis. Funkcja ma zwrócić liczbę znaków cyfr nieparzystych występujących w napisie. Stwórz przypadek testowy.

21. Napisz funkcję `toLowerNew`, która przyjmuje jako argument wskaźnik do napisu typu `const char*` i zwraca wskaźnik do nowego napisu, w którym wszystkie duże litery zostały zamienione na małe. Oryginalna tablica znaków pozostaje niezmieniona. Pamiętaj o alokacji pamięci dla nowego napisu. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia i funkcji alokacji pamięci. Stwórz przypadek testowy dla tej funkcji.

22. Napisz funkcję `toUpperNew`, która przyjmuje jako argument wskaźnik do napisu typu `const char*` i zwraca wskaźnik do nowego napisu, w którym wszystkie małe litery zostały zamienione na duże. Oryginalna tablica znaków pozostaje niezmieniona. Pamiętaj o alokacji pamięci dla nowego napisu. Nie korzystaj ze wbudowanych funkcji języka oprócz operacji wejścia/wyjścia i funkcji alokacji pamięci. Stwórz przypadek testowy dla tej funkcji.

12 Tablice wielowymiarowe

1. Napisz funkcję `sumMatrix`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę o elementach typu `int` o wymiarach $n \times m$ i zwraca jako wartość sumę wartości elementów tablicy. Stwórz przypadek testowy dla funkcji.
2. Napisz funkcję `sumArray`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$ i zwraca jako wartość sumę wartości elementów tablicy. Stwórz przypadek testowy dla funkcji.
3. Napisz funkcję `maxRowIdx`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę o elementach typu `int` o wymiarach $n \times m$. Funkcja ma zwrócić indeks wiersza, w którym znajduje się największy element (w przypadku kilku możliwych indeksów, zwróć najmniejszy z możliwych). Stwórz przypadek testowy dla funkcji.
4. Napisz funkcję `minColIdx`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma zwrócić indeks kolumny, w którym znajduje się najmniejszy element (w przypadku kilku możliwych indeksów, zwróć najmniejszy z możliwych). Stwórz przypadek testowy dla funkcji.
5. Napisz funkcję `copyMat`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwie dwumiarowe tablice o elementach typu `int` o wymiarach $n \times m$. Funkcja ma przepisać zawartość drugiej tablicy do pierwszej tablicy. Stwórz przypadek testowy dla funkcji.
6. Napisz funkcję `copyArr2D`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwie dwumiarowe tablice tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma przepisać zawartość drugiej tablicy do pierwszej tablicy. Stwórz przypadek testowy dla funkcji.
7. Napisz funkcję `swapElems`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwie dwumiarowe tablice o elementach typu `int` o wymiarach $n \times m$. Funkcja ma zamienić odpowiednie elementy obu tablic między sobą. Stwórz przypadek testowy dla funkcji.
8. Napisz funkcję `swapItems`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwie dwumiarowe tablice tablic o elementach typu `int` o wymiarach $n \times m$.

Funkcja ma zamienić odpowiednie elementy obu tablic między sobą. Stwórz przypadek testowy dla funkcji.

9. Napisz funkcję `revRows`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę o elementach typu `int` o wymiarach $n \times m$. Funkcja ma odwrócić kolejność elementów w każdym wierszu. Stwórz przypadek testowy dla funkcji.

10. Napisz funkcję `revRowArr`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma odwrócić kolejność elementów w każdym wierszu. Stwórz przypadek testowy dla funkcji.

11. Napisz funkcję `revCols`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę o elementach typu `int` o wymiarach $n \times m$. Funkcja ma odwrócić kolejność elementów w każdej kolumnie. Stwórz przypadek testowy dla funkcji.

12. Napisz funkcję `revColArr`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma odwrócić kolejność elementów w każdej kolumnie. Stwórz przypadek testowy dla funkcji.

13. Napisz funkcję `transpose`, która dostaje jako argumenty dodatnią liczbę całkowitą `n` i dwuwymiarową kwadratową tablicę o elementach typu `int` o wymiarach $n \times n$. Funkcja ma transponować elementy tablicy (zamienić wiersze na kolumny). Stwórz przypadek testowy dla funkcji.

14. Napisz funkcję `transArr`, która dostaje jako argumenty dodatnią liczbę całkowitą `n` i dwuwymiarową kwadratową tablicę tablic o elementach typu `int` o wymiarach $n \times n$. Funkcja ma transponować elementy tablicy (zamienić wiersze na kolumny). Stwórz przypadek testowy dla funkcji.

15. Napisz funkcję `oddAvg`, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu `int`) oraz jej wymiary $n, n > 1$ i $m, m > 1$. Funkcja ma zwrócić średnią elementów stojących na nieparzystych indeksach (oba mają być jednocześnie nieparzyste). Stwórz przypadek testowy.

16. Napisz funkcję `evenSum`, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu `int`) oraz jej wymiary $n, n > 1$ i $m, m > 1$. Funkcja powinna zwrócić sumę elementów stojących na parzystych indeksach (zarówno indeksy wierszy jak i kolumn są parzyste). Stwórz przypadek testowy.

17. Napisz funkcję `swap2nd`, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu `int`) oraz jej wymiary n i m. Funkcja ma zamienić miejscami wiersz o indeksie 1 z przedostatnim. Jeśli tablica ma mniej niż cztery wiersze, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 2 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \\ -2 & 8 & 23 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 3 & -3 \\ -3 & -6 & 11 \\ 1 & 4 & 7 \\ -2 & 8 & 23 \end{bmatrix}$$

18. Napisz funkcję `flipDiag`, której argumentami są dwie dodatnie liczby całkowite n i m oraz dwuwymiarowa tablica elementów (zawierająca zmienne typu int) wymiaru $n \times m$. Funkcja ma zamienić kolejność elementów leżących na głównej przekątnej. Funkcja powinna działać tylko dla macierzy kwadratowych, w przeciwnym przypadku nie powinna nic robić. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 2 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 2 \end{bmatrix}$$

19. Napisz funkcję `swapColumns`, która przyjmuje jako argumenty dwuwymiarową tablicę tablic liczb całkowitych, jej wymiary oraz dwa indeksy kolumn do zamiany miejscami. Funkcja powinna przestawić wskazane kolumny i zwrócić zmodyfikowaną tablicę. Uwzględnij sytuację, jeśli podane indeksy są nieprawidłowe - wtedy funkcja ma nic nie robić. Stwórz przypadek testowy dla funkcji.

20. Napisz funkcję `revOddCol`, której argumentami są dwie dodatnie liczby całkowite n i m oraz dwuwymiarowa tablica elementów (zawierająca zmienne typu int) wymiaru $n \times m$. Funkcja ma odwrócić kolejność elementów w kolumnach o nieparzystych indeksach. Stwórz przypadek testowy.

Przykład.

$$\begin{bmatrix} 2 & 3 & -3 \\ 1 & 4 & 7 \\ -3 & -6 & 11 \\ -2 & 8 & 23 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 8 & -3 \\ 1 & -6 & 7 \\ -3 & 4 & 11 \\ -2 & 3 & 23 \end{bmatrix}$$

21. Stwórz funkcję `idxDiff`, której argumentem jest dwuwymiarowa kwadratowa tablica tablic (zawierająca elementy typu `int`) oraz jej wymiar $n, n > 0$. Funkcja powinna zwracać różnicę między sumą indeksów najmniejszego a sumą indeksów największego elementu w tablicy. W przypadku kilku elementów o najmniejszej lub największej wartości, powinny to być najmniejsze możliwe sumy indeksów. Stwórz przypadek testowy.

22. Napisz funkcję `sqEndCols`, której argumentem jest dwuwymiarowa tablica tablic (zawierająca zmienne typu `int`) oraz jej wymiary n i m ($n > 0, m > 1$). Funkcja ma podnieść do kwadratu element z pierwszej i ostatniej kolumny. Stwórz przypadek testowy.

Przykład:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 5 & 8 \\ 9 & 8 & -2 \\ -2 & 3 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 9 \\ 0 & 5 & 64 \\ 81 & 8 & 4 \\ 4 & 3 & 16 \end{bmatrix}$$

23. Napisz funkcję `transCopy`, która dostaje jako argumenty dodatnią liczbę całkowitą `n` i dwuwymiarową kwadratową tablicę tablic o elementach typu `int` o wymiarach $n \times n$. Funkcja ma zwrócić wskaźnik na nowo-utworzoną dynamiczną dwuwymiarową tablicę powstałą z przekazanego argumentu poprzez transponowanie jej elementów. Stwórz przypadek testowy dla funkcji.

24. Napisz funkcję `newArrOneD`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma zwrócić wskaźnik nowo-utworzoną dynamiczną jednowymiarową tablicę powstało z argumentu poprzez przepisanie elementów kolejno wierszami. Stwórz przypadek testowy dla funkcji.

25. Napisz funkcję `sortRows`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma posortować elementy osobno każdego wiersza od najmniejszego do największego. Stwórz przypadek testowy dla funkcji.

Przykład:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 8 \\ 9 & 8 & -2 \\ -3 & 5 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ -5 & 0 & 8 \\ -2 & 8 & 9 \\ -3 & 4 & 5 \end{bmatrix}$$

26. Napisz funkcję `sortCols`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma posortować elementy osobno każdej kolumny od największej do najmniejszej. Stwórz przypadek testowy dla funkcji.

Przykład:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 8 \\ 9 & 8 & -2 \\ -3 & 5 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 9 & 8 & 8 \\ 1 & 5 & 4 \\ 0 & 2 & 3 \\ -3 & -5 & -2 \end{bmatrix}$$

27. Napisz funkcję `showRows`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma wyświetlić elementy tablicy w następujący sposób: elementy każdego wiersza mają być rozdzielone przecinkiem, a kolejne wiersze między osobą rozdzielone znakiem końca linii, na końcu każdej linii nie powinno być przecinka. Stwórz przypadek testowy dla funkcji.

Przykład: Dla tablicy

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 8 \\ 9 & 8 & -2 \\ -3 & 5 & 4 \end{bmatrix}$$

ma być wyświetlone dokładnie:

```
1,2,3
0,-5,8
9,8,-2
-3,5,4
```

28. Napisz funkcję `showCols`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma wyświetlić elementy tablicy w następujący sposób: elementy każdej kolumny mają być rozdzielone przecinkiem w jednej linii, a kolejne kolumny między osobą rozdzielone znakiem końca linii, na końcu każdej linii nie powinno być przecinka. Stwórz przypadek testowy dla funkcji.

Przykład: Dla tablicy

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 8 \\ 9 & 8 & -2 \\ -3 & 5 & 4 \end{bmatrix}$$

ma być wyświetlone dokładnie:

```
1,0,9,-3
2,-5,8,5
3,8,-2,4
```

29. Napisz funkcję `snkPrint`, która dostaje jako argumenty dwie dodatnie liczby całkowite `n` i `m` oraz dwuwymiarową tablicę tablic o elementach typu `int` o wymiarach $n \times m$. Funkcja ma wyświetlić elementy tablicy w jednym wierszu rozdzielone spacją na zasadzie "ślimaka,węża" (spójrz na przykład). Stwórz przypadek testowy dla funkcji.

Przykład: Dla tablicy

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -5 & 8 \\ 9 & 8 & -2 \\ -3 & 5 & 4 \end{bmatrix}$$

ma być wyświetlone dokładnie:

```
1 2 3 8 -2 4 5 -3 9 0 -5 8
```

13 Złożone typy danych

W poniższych zadaniach o ile nie zaznaczono inaczej pojęcia nazwa i alias struktury można używać zamiennie.

1. Zdefiniuj strukturę `Car` z polami: `brand` (napis), `model` (tablica znaków o rozmiarze 20), `year` (liczba całkowita) i `isAvailable` (wartość 0 lub 1), która reprezentuje informację o dostępności samochodu na sprzedaż. W `main` stwórz kilka zmiennych w typie `Car`.
2. Utwórz strukturę `Student`, która zawiera: `firstName` (łańcuch znaków), `lastName` (łańcuch znaków), `id` (liczba całkowita) oraz `gpa` (liczba zmiennoprzecinkowa), reprezentującą średnią ocen studenta. W `main` stwórz trzy zmienne w typie `Student`. Następnie z tych trzech zmiennych stwórz tablicę.
3. Zaplanuj strukturę `Rectangle`, mającą pola: `width` (liczba zmiennoprzecinkowa), `height` (liczba zmiennoprzecinkowa) i `color` (łańcuch znaków), która ma służyć do przechowywania informacji o wymiarach i kolorze prostokąta. W `main` stwórz 4-elementową tablicę zmiennych typu `Rectangle`.
4. Stwórz strukturę `WeatherInfo` zawierającą: `temperature` (liczba zmiennoprzecinkowa), `humidity` (liczba zmiennoprzecinkowa) i `windSpeed` (liczba zmiennoprzecinkowa), która będzie używana do przechowywania danych meteorologicznych. W `main` stwórz kilka zmiennych w typie `WeatherInfo`.
5. Zaprojektuj strukturę `Book` z polami: `title` (łańcuch znaków), `author` (łańcuch znaków), `publishedYear` (liczba całkowita) i `pages` (liczba całkowita), która ma opisywać podstawowe informacje o książce. W `main` stwórz 5-elementową tablicę zmiennych typu `Book`.
6. Napisz strukturę `Person` z polami `name` (tablica znaków długości 20) oraz `age` (typu `int`). Następnie napisz funkcje i wywołają każdą z nich co najmniej jeden raz:

a) `initPerson` - funkcja przyjmuje dwa argumentem imię i wiek i zwraca wskaźnik na nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów.
b) `showPerson` - funkcja, której argumentem jest zmienna w typie `Person`. Funkcja ma wypisać opis przekazanego argumentu (wpisać wiek i imię na standardowym wyjściu).
c) `birthday` - funkcja, której argumentem jest wskaźnik do struktury typu `Person`. Funkcja ma powiększyć wiek o 1 w przekazanym argumencie.

Stwórz przypadek testowy dla każdej z funkcji.

7. Napisz strukturę `Car` z polami `brand` (tablica znaków długości 20) oraz `mileage` (typu `int`). Następnie napisz funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initCar` - funkcja przyjmuje dwa argumenty: markę i przebieg, i zwraca nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów.
b) `showCar` - funkcja, której argumentem jest zmienna w typie `Car`. Funkcja ma wypisać opis przekazanego argumentu (wypisać markę i przebieg na standardowym wyjściu).
c) `mileageService` - funkcja, której argumentem jest wskaźnik do struktury typu `Car`. Funkcja ma dodać 10000 do przebiegu w przekazanym argumencie.

Stwórz przypadek testowy dla każdej z funkcji.

8. Napisz strukturę `Book` z polami `title` (tablica znaków długości 50) oraz `page_count` (typu `int`). Następnie napisz funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initBook` - funkcja przyjmuje dwa argumenty: tytuł i liczbę stron, i zwraca nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów.
b) `showBook` - funkcja, której argumentem jest zmienna w typie `Book`. Funkcja ma wypisać opis przekazanego argumentu (wypisać tytuł i liczbę stron na standardowym wyjściu).
c) `addPages` - funkcja, której argumentem jest wskaźnik do struktury typu `Book`. Funkcja ma dodać 10 do liczby stron w przekazanym argumencie.

Stwórz przypadek testowy dla każdej z funkcji.

9. Napisz strukturę `Laptop` z polami `model` (tablica znaków długości 30) oraz `price` (typu `float`). Następnie napisz funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initLaptop` - funkcja przyjmuje dwa argumenty: model i cenę, i zwraca nowo-utworzoną strukturę ustawiającą składowe z przekazanych argumentów.
b) `showLaptop` - funkcja, której argumentem jest zmienna w typie `Laptop`. Funkcja ma wypisać opis przekazanego argumentu (wypisać model i cenę na standardowym wyjściu).
c) `reducePrice` - funkcja, której argumentem jest wskaźnik do struktury typu `Laptop`. Funkcja ma obniżyć cenę o 5% w przekazanym argumencie.

Stwórz przypadek testowy dla każdej z funkcji.

10. Stwórz strukturę `Airplane` o dwóch polach `model` (napis) oraz `number_of_engines` (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur `Airplane` oraz rozmiar tablicy. Funkcja ma zwrócić najmniejszą liczbę silników. Stwórz przypadek testowy.

11. Stwórz strukturę `Computer` o dwóch polach `brand` (napis) oraz `number_of_cores` (dowolny typ całkowity). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur `Computer` oraz rozmiar tablicy. Funkcja ma zwrócić największą liczbę rdzeni. Stwórz przypadek testowy.

12. Stwórz strukturę `Book` o dwóch polach: `title` (napis) i `page_count` (`int`). Następnie stwórz funkcję, której argumentami jest niepusta tablica struktur `Book` oraz rozmiar tablicy. Funkcja ma zwrócić książkę (jako strukturę) z największą liczbę stron. W przypadku kilku książek w tablicy z największą liczbą stron, to zwróć ostatnią z możliwych. Stwórz przypadek testowy.

13. Napisz strukturę `Car` z polami `brand` (tablica znaków długości 50) oraz `mileage` (typu `int`). Następnie napisz dwie funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initCar` - funkcja przyjmuje dwa argumenty: markę i przebieg, i zwraca wskaźnik na nowo utworzoną strukturę, ustawiając składowe z przekazanych argumentów. Dodatkowo funkcja powinna sprawdzić, aby marka była napisem długości co najmniej 2 i przebieg był większy niż 0. W przypadku nie spełnienia co najmniej jednego z warunków, funkcja powinna zwracać NULL.
b) `increaseMileage` - funkcja, której argumentem jest wskaźnik do struktury typu `Car`. Funkcja ma dodać 1000 do przebiegu w przekazanym argumencie.

Upewnij się, że drugą funkcję możesz wywołać w `main`.

14. Napisz strukturę `Bike` z polami `model` (tablica znaków długości 40) oraz `distance_travelled` (typu `int`). Następnie napisz dwie funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initBike` - funkcja przyjmuje dwa argumenty: model roweru i odległość przejechaną, i zwraca wskaźnik na nowo utworzoną strukturę, ustawiając wartości zgodnie z przekazanymi argumentami. Funkcja powinna dodatkowo sprawdzać, czy model jest napisem o długości co najmniej 3 i odległość przejechana jest większa od 0. W przypadku nie spełnienia co najmniej jednego z warunków, funkcja powinna zwracać NULL.
b) `increaseDistance` - funkcja, której argumentem jest wskaźnik do struktury `Bike`. Funkcja powinna dodać 500 do przejechanej odległości.

Upewnij się, że drugą funkcję możesz wywołać w `main`.

15. Napisz strukturę `Computer` z polami `manufacturer` (tablica znaków długości 30) oraz `usage_hours` (typu `int`). Następnie napisz dwie funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initComputer` - funkcja przyjmuje dwa argumenty: producenta i liczbę godzin używania, i zwraca wskaźnik na nowo utworzoną strukturę, ustawiając wartości zgodnie z przekazanymi argumentami. Funkcja powinna także sprawdzać, czy nazwa producenta ma co najmniej 2 znaki i liczba godzin jest większa od 0. Jeśli oba warunki nie są spełnione łącznie, funkcja powinna zwracać NULL.
b) `increaseUsage` - funkcja, której argumentem jest wskaźnik do struktury `Computer`. Funkcja powinna dodać 100 do liczby godzin użytkowania.

Upewnij się, że drugą funkcję możesz wywołać w `main`.

16. Napisz strukturę `Apartment` z polami `address` (tablica znaków długości 100) oraz `number_of_residents` (typu `int`). Następnie napisz dwie funkcje i wywołaj każdą z nich co najmniej jeden raz:

a) `initApartment` - funkcja przyjmuje dwa argumenty: adres i liczbę mieszkańców, i zwraca wskaźnik na nowo utworzoną strukturę, ustawiając wartości zgodnie z przekazanymi argumentami. Funkcja powinna sprawdzać, czy adres ma co najmniej 5 znaków i liczba mieszkańców jest większa niż 0. W przypadku nie spełnienia obu warunków łącznie, funkcja powinna zwracać NULL.
b) `increaseResidents` - funkcja, której argumentem jest wskaźnik do struktury `Apartment`. Funkcja powinna dodać 2 do liczby mieszkańców.

Upewnij się, że drugą funkcję możesz wywołać w `main`.

17. Utwórz typ wyliczeniowy `Month` reprezentujący miesiące. Napisz funkcję `days_in_month()`, która przyjmuje jako argument wartość typu `Month` i zwraca liczbę dni w danym miesiącu. Pomiń problem roku przestępnego w tym zadaniu. Stwórz przypadek testowy dla funkcji.

18. Utwórz typ wyliczeniowy `Day` reprezentujący dni tygodnia. Napisz funkcję `print_days()`, która przyjmuje jako argumenty wartość typu `Day` i liczbę `n`. Funkcja powinna wydrukować bieżący dzień tygodnia, a następnie rekurencyjnie wywołać siebie z następnym dniem tygodnia, dekrementując `n` przy każdym wywołaniu, aż `n` spadnie do 0. Stwórz przypadek testowy dla funkcji.

19. Stwórz unię `Number`, która umożliwia jednoczesne przechowywanie liczby całkowitej i wymiernej. Następnie "opakuj" unię w strukturę i stwórz funkcję, które umożliwiają zainicjowanie pól unii ze standardowego wejścia. Stwórz przypadek testowy dla funkcji.

20. Stwórz strukturę `element` o dwóch polach: `i` typu `int` oraz `next` będące wskaźnikiem na zdefiniowaną strukturę. Stwórz w `main` kilka zmiennych strukturalnych.

14 Listy jednokierunkowe

Wykorzystaj z zadaniach konwencją omawianą na wykładzie.

14.1 Bez funkcji

1. W `main` stwórz listę bez głowy o trzech elementach 4,5,-12. Następnie w kolejnych wierszach wyświetl elementy znajdujące się na liście. Wykorzystaj strukturę:

```
struct element {
    int x;
    struct element * next;
};
```

2. W `main` stwórz listę z głową o trzech elementach 4,5,-12. Następnie w kolejnych wierszach wyświetl elementy znajdujące się na liście. Wykorzystaj strukturę:

```
struct element {
    int x;
    struct element * next;
};
```

14.2 Wyświetlanie

3. Napisz funkcję `printListWithoutHead`, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma wyświetlić kolejne elementy listy w kolejnych wierszach lub wyświetlić komunikat, że lista jest pusta. Stwórz przypadek testowy.

4. Napisz funkcję `printListWithHead`, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma wyświetlić kolejne elementy listy w kolejnych wierszach lub wyświetlić komunikat, że lista jest pusta. Stwórz przypadek testowy.

5. Napisz funkcję `printPos`, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma wyświetlić te elementy listy, które są dodatnie. Stwórz przypadek testowy.

6. Napisz funkcję `printOdd`, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma wyświetlić te elementy listy, które są nieparzyste. Stwórz przypadek testowy.

14.3 Tworzenie listy

7. Napisz bez-argumentową funkcję `createNoHead`, która zwraca wskaźnik na "nowo-utworzoną" pustą listę bez głowy. Stwórz przypadek testowy. Wykorzystaj strukturę:

```
struct element {
    int x;
    struct element * next;
};
```

8. Napisz bez-argumentową funkcję `createWithHead`, która zwraca wskaźnik na "nowo-utworzoną" pustą listę z głową. Stwórz przypadek testowy. Wykorzystaj strukturę:

```
struct element {
    int x;
    struct element * next;
};
```

14.4 Dodawanie na początek

9. Napisz funkcję `addFirst`, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

oraz liczbę całkowitą `a`. Funkcja ma dodać na początek listy nowy element o wartości `a`. Stwórz przypadek testowy.

10. Napisz funkcję `addFirst`, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

oraz liczbę całkowitą `a`. Funkcja ma dodać na początek listy nowy element o wartości `a`. Stwórz przypadek testowy.

14.5 Dodawanie na koniec

11. Napisz funkcję `addLast`, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

oraz liczbę całkowitą `a`. Funkcja ma dodać na koniec listy nowy element o wartości `a`. Stwórz przypadek testowy.

12. Napisz funkcję `addLast`, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

oraz liczbę całkowitą `a`. Funkcja ma dodać na koniec listy nowy element o wartości `a`. Stwórz przypadek testowy.

14.6 Wyszukiwanie/liczniki

13. Napisz funkcję `find`, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

oraz liczbę całkowitą `a`. Funkcja ma zwrócić 1, jeśli co najmniej jeden element listy jest równy `a` oraz ma zwrócić 0 w pozostałych wypadkach. Stwórz przypadek testowy.

14. Napisz funkcję `find`, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

oraz liczbę całkowitą `a`. Funkcja ma zwrócić 1, jeśli co najmniej jeden element listy jest równy `a` oraz ma zwrócić 0 w pozostałych wypadkach. Stwórz przypadek testowy.

15. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma zwrócić sumę elementów nieparzystych znajdujących się na liście. Stwórz przypadek testowy.

16. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma zwrócić sumę elementów ujemnych znajdujących się na liście. Stwórz przypadek testowy.

17. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma zwrócić adres ostatniego elementu parzystego lub NULL gdy takie elementu nie ma lub lista jest pusta. Stwórz przypadek testowy.

18. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma zwrócić adres ostatniego elementu dodatniego lub NULL gdy takie elementu nie ma lub lista jest pusta. Stwórz przypadek testowy.

14.7 Usuwanie pojedynczych elementów

19. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć początkowy element na liście (o ile lista nie jest pusta). Stwórz przypadek testowy.

20. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć początkowy element na liście (o ile lista nie jest pusta). Stwórz przypadek testowy.

21. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć ostatni element na liście (o ile lista nie jest pusta). Stwórz przypadek testowy.

22. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć ostatni element na liście (o ile lista nie jest pusta). Stwórz przypadek testowy.

23. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć przedostatni element na liście (o ile lista ma co najmniej dwa elementy). Stwórz przypadek testowy.

24. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć przedostatni element na liście (o ile lista ma co najmniej dwa elementy). Stwórz przypadek testowy.

14.8 Usuwanie całej listy

25. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć całą listę (ręcznie element za elementem). Po usunięciu upewnij się, że lista jest pusta zgodnie z konwencją. Stwórz przypadek testowy.

26. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct element {
    int x;
    struct element * next;
};
```

Funkcja ma usunąć całą listę (ręcznie element za elementem). Po usunięciu upewnij się, że lista jest pusta zgodnie z konwencją. Stwórz przypadek testowy.

14.9 Zadania różne

27. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:

```
struct node {
    int i;
    struct node * next;
};
```

Funkcja ma wyświetlić na konsoli w kolejnych wierszach wartości elementów na liście będących kwadratami liczb całkowitych. Stwórz przypadek testowy.

Przykład. Jeśli lista składa się z elementów 4,5,6,-34,0,25,11, to ma być wyświetlone w kolejnych wierszach: 4,0,25.

28. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct node {
    double x;
    struct node * next;
};
```

i zwraca najmniejszą z liczb znajdujących się na liście. W przypadku pustej listy, funkcja ma zwrócić zero. Stwórz jeden przypadek testowy.

29. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:

```
struct node {
    int i;
    struct node * next;
};
```

Funkcja ma zdublować wartość ostatniego elementu, o ile lista jest nie pusta. W przypadku pustej listy, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład: Dla listy 3,4,5 ma być zdublowana 5, więc po modyfikacji lista ma być postaci 3,4,5,5.

30. Napisz funkcję, która otrzymuje jako argument listę z głową o elementach typu:

```
struct elem {
    int a;
    struct elem * next;
};
```

Funkcja ma zdublować wartość ostatniego elementu, o ile lista jest nie pusta. W przypadku pustej listy, funkcja ma nic nie robić. Stwórz przypadek testowy.

Przykład: Dla listy 3,4,5 ma być zdublowana 5, więc po modyfikacji lista ma być postaci 3,4,5,5.

31. Napisz funkcję, która porównuje dwie listy z głową o elementach typu:

```
struct node {
    int i;
    struct node * next;
};
```

i zwraca 1 jeśli ostatnie elementy na liście są równe oraz 0 w pozostałych przypadkach (także wtedy gdy któraś z list lub obie są puste). Stwórz jeden przypadek testowy.

32. Napisz funkcję, która porównuje dwie listy bez głowy o elementach typu:

```
struct node {
    int a;
    struct node * next;
};
```

i zwraca 1 jeśli listy są takiej samej długości oraz 0 w pozostałych przypadkach (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

33. Napisz funkcję, która przyjmuje dwie listy z głową o elementach typu:

```
struct node {
    int value;
    struct node * next;
};
```

i zwraca 1 jeśli suma wszystkich elementów nieparzystych w obu listach jest taka sama oraz 0 w przeciwnym przypadku (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

34. Napisz funkcję, która przyjmuje dwie listy bez głowy o elementach typu:

```
struct node {
    int w;
    struct node * next;
};
```

i zwraca 1 jeśli suma wszystkich elementów parzystych w obu listach jest taka sama oraz 0 w przeciwnym przypadku (także wtedy, gdy któraś z list lub obie są puste). Stwórz przypadek testowy.

35. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    int x;
    struct node * next;
};
```

Funkcja ma zwrócić sumę elementów nieparzystych z listy. Stwórz jeden przypadek testowy.

36. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct elem {
    int x;
    struct elem * next;
};
```

Funkcja ma podwoić wszystkie elementy dodatnie na liście (o ile istnieją). Stwórz jeden przypadek testowy.

37. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    int y;
    struct node * next;
};
```

oraz dwie liczby całkowite `a` i `b`. Funkcja ma dodać na początek listy dwa nowe elementy i ich wartości ustawić odpowiednio z podanych argumentów. Stwórz jeden przypadek testowy.

38. Napisz funkcję, która przyjmuje jako argument listę bez głowy o elementach typu:

```
struct node {
    int a;
    struct node * next;
};
```

oraz dwie liczby całkowite `a` i `b`. Funkcja ma dodać na początek listy dwa nowe elementy i ich wartości ustawić odpowiednio z podanych argumentów. Stwórz jeden przypadek testowy.

39. Napisz funkcję, która otrzymuje jako argument listę bez głowy o elementach typu:

```
struct element {
    float value;
    struct element * next;
};
```

Funkcja ma zamienić wartości każdego elementu na jego wartość bezwzględną. Dla listy pustej funkcja ma nic nie robić. Stwórz przypadek testowy.

40. Napisz funkcję, która otrzymuje jako argument listę z głową o elementach typu:

```
struct node {
    int val;
    struct node * next;
};
```

Funkcja ma zamienić wartości każdego elementu na jego wartość bezwzględną. Dla listy pustej funkcja ma nic nie robić. Stwórz przypadek testowy.

41. Napisz funkcję, która przyjmuje jako argument listę z głową o elementach typu:

```
struct node {
    int x;
    struct node * next;
};
```

oraz liczbę całkowitą `d`. Funkcja ma zwrócić ile na liście jest elementów równych `d`. Stwórz jeden przypadek testowy.

Część II

Dodatki

15 Formatowanie zmiennych liczbowych w `printf`

1. Napisz program, który wyświetla liczbę całkowitą w formacie dziesiętnym.
2. Napisz program, który wyświetla liczbę całkowitą w formacie szesnastkowym z małymi literami.
3. Napisz program, który wyświetla liczbę całkowitą w formacie szesnastkowym z dużymi literami.
4. Napisz program, który wyświetla liczbę całkowitą w formacie ósemkowym.
5. Napisz program, który wyświetla liczbę całkowitą z wiodącymi zerami, zakładając, że szerokość pola wynosi 8 znaków.
6. Napisz program, który wyświetla liczbę całkowitą z wyrównaniem do prawej w polu o szerokości 10 znaków.
7. Napisz program, który wyświetla liczbę całkowitą z wyrównaniem do lewej w polu o szerokości 10 znaków.
8. Napisz program, który wyświetla dodatnią liczbę całkowitą z dodanym znakiem plus na początku.
9. Napisz program, który wyświetla liczbę całkowitą z użyciem notacji naukowej (z małymi literami).
10. Napisz program, który wyświetla liczbę całkowitą, dodając separator tysięcy.
11. Napisz program, który wyświetla liczbę zmiennoprzecinkową 123.456789 z dokładnością do dwóch miejsc po przecinku.
12. Stwórz program, który formatuje i wyświetla liczbę zmiennoprzecinkową -9876.54321, tak aby była przedstawiona z co najmniej 10 miejscami przed przecinkiem, dopełniając brakujące miejsca spacjami.
13. Utwórz program, który prezentuje liczbę zmiennoprzecinkową 0.000789 w notacji naukowej z dokładnością do czterech miejsc po przecinku.
14. Zaprojektuj program, który wyświetla liczbę zmiennoprzecinkową 12345.6789 z użyciem notacji naukowej i dokładnością do jednego miejsca po przecinku.
15. Stwórz program, który wyświetla liczbę zmiennoprzecinkową 3.14159 w formacie, gdzie przed liczbą znajduje się znak plus, z dokładnością do trzech miejsc po przecinku.
16. Napisz program, który formatuje i wyświetla liczbę zmiennoprzecinkową 123456.789 z dokładnością do pięciu miejsc po przecinku, zapewniając, że przed liczbą pojawi się miejsce na znak.
17. Utwórz program, który wyświetla liczbę zmiennoprzecinkową -0.0025 z dokładnością do sześciu miejsc po przecinku, zawsze zaczynając od znaku minus.

18. Zaprojektuj program, który przedstawia liczbę zmiennoprzecinkową 0.00123456789 w notacji naukowej z dokładnością do dwóch miejsc po przecinku i zawsze z przedrostkiem plus dla dodatnich wartości.
19. Stwórz program, który wyświetla liczbę zmiennoprzecinkową 9999999.99999 z dokładnością do trzech miejsc po przecinku, używając notacji naukowej, gdy jest to konieczne.
20. Napisz program, który formatuje i wyświetla liczbę zmiennoprzecinkową -123.456 w taki sposób, że zawsze zajmuje ona dokładnie 12 miejsc, w tym znak, liczby przed i po przecinku, dopełniając niewykorzystane miejsca spacjami.

16 Zadanie różne cz.1

1. Napisz funkcję, która ma dwa argumenty: dodatnią liczbę całkowitą n oraz liczbę wymierną x. Funkcja ma zwrócić jako liczbę wartość wyrażenia będącego sumą szeregu:

$$\frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!}.$$

W zadaniu nie korzystaj z wbudowanych funkcji matematycznych. Stwórz przypadek testowy dla funkcji.

2. Napisz funkcję, która ma dwa argumenty: dodatnią liczbę całkowitą n oraz liczbę wymierną x. Funkcja ma zwrócić jako liczbę wartość wyrażenia będącego sumą szeregu:

$$(x + 1) + (x^2 + 2) + ... + (x^n + n).$$

W zadaniu nie korzystaj z funkcji matematycznych. Stwórz przypadek testowy dla funkcji.

3. Napisz rekurencyjną funkcję, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwracać sumę:

$$5 + 55 + 555 + ... + \underbrace{5...5}_{n\ razy}.$$

Stwórz przypadek testowy.

4. Napisz funkcję rekurencyjną, która wypisze wszystkie liczby naturalne od n do $2n$ (włącznie) dla pewnej dodatniej liczby całkowitej n. Możesz samodzielnie ustalić liczbę i typ argumentów pamiętając, że funkcja ma być rekurencyjna. Stwórz przypadek testowy.

5. Napisz funkcję, która ma dwa argumenty: dodatnią liczbę całkowitą n oraz dodatnią liczbę wymierną x. Funkcja ma zwrócić obliczoną wartość wyrażenia:

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

W zadaniu nie korzystaj ze wbudowanych funkcji matematycznych. Stwórz przypadek testowy.

6. Napisz funkcję, której argumentem jest jest dodatnia liczba całkowita n. Funkcja ma wyświetlać wszystkie możliwe liczby Nivena mniejsze lub równe n (bez rozkładów). Stwórz przypadek testowy dla funkcji. W zadaniu nie korzystaj ze wbudowanych funkcji matematycznych.

Liczby Nivena – liczby naturalne, które są podzielne przez sumę tworzących je cyfr. Początkowe liczby Nivena: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 18, 20, 21, 24, 27, 30, 36, 40.

7. Napisz funkcję, której argumentem są dodatnia liczba całkowita n. Funkcja ma zwrócić odpowiednią wartość logiczną (zero lub jeden) z informacją czy liczby n jest automorficzna. Nie korzystaj ze wbudowanych funkcji poza instrukcjami wejścia/wyjścia oraz nie korzystaj z konwersji na string/wektor. Stwórz przypadek testowy.

Liczba automorficzna – liczba, która podniesiona do kwadratu zawierają w końcówce same siebie. Np. $76 \cdot 76 = 5776$.

Przykłady: 0, 1, 5, 6, 25, 76, 376, 625.

8. Napisz funkcję, której argumentem są dodatnia liczba całkowita n. Funkcja ma zwrócić odpowiednią wartość logiczną (zero lub jeden) z informacją czy liczby n jest wesoła. Stwórz przypadek testowy.

Liczba wesoła – liczba naturalna zdefiniowana jako obliczanie sumy kwadratów cyfr składających się na liczbę. Powtarzamy tę operację dla kolejnych wyników tak długo, aż uzyskamy liczbę 1 lub wyniki zaczną się powtarzać. Jeżeli w wyniku procesu otrzymaliśmy 1, pierwotna liczba jest liczbą wesołą. W przeciwnym przypadku jest liczbą niewesołą.

Przykładowo 7 jest liczbą wesołą:

$$7^2 = 49, \quad 4^2 + 9^2 = 97, \quad 9^2 + 7^2 = 130$$
$$1^2 + 3^2 + 0^2 = 10, \quad 1^2 + 0^2 = 1.$$

Przykładowo 85 jest liczbą niewesołą:

$$8^2 + 5^2 = 89, \quad 8^2 + 9^2 = 145, \quad 1^2 + 4^2 + 5^2 = 42, \quad 4^2 + 2^2 = 20$$
$$2^2 + 0^2 = 4, \quad 4^2 = 16, \quad 1^2 + 6^2 = 37, \quad 3^2 + 7^2 = 58, \quad 5^2 + 8^2 = 89$$

9. Napisz funkcję, której argumentem są dodatnia liczba całkowita a. Funkcja ma zwrócić odpowiednią wartość logiczną (zero lub jeden) z informacją czy liczby a jest narcystyczna. Nie korzystaj ze wbudowanych funkcji poza instrukcjami wejścia/wyjścia oraz nie korzystaj z konwersji na string/wektor. Stwórz przypadek testowy.

Liczba narcystyczna - n-cyfrowa liczba naturalna, która jest sumą swoich cyfr podniesionych do potęgi n.

Przykład: 153 jest liczbą narcystyczną. 3 to liczba cyfr oraz $153 = 1^3 + 5^3 + 3^3$.

10. Napisz funkcję, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwrócić ile cyfr 1 występuje w zapisie dziesiętnym tej liczby. W zadaniu nie korzystaj z konwersji liczby na napis. Stwórz przypadek testowy dla funkcji

11. Napisz funkcję, która ma dwa argumenty: dwie dodatnie liczby całkowite m i n ($n > 1$). Funkcja ma zwrócić wartość wyrażenia:

$f(m, n) = \lfloor \sqrt[n]{m} \rfloor$

Symbol $\lfloor x \rfloor$ - oznacza część całkowitą z x. Stwórz przypadek testowy dla funkcji. W zadaniu nie korzystaj ze wbudowanych funkcji matematycznych.

12. Napisz funkcję, której argumentem są dwie dodatnie liczby całkowite a i b. Funkcja ma zwrócić odpowiednią wartość logiczną (zero lub jeden) z informacją czy liczby a i b są swoim lustrzanym odbiciem. Nie korzystaj ze wbudowanych funkcji poza instrukcjami wejścia/wyjścia oraz nie korzystaj z konwersji na napis. Stwórz przypadek testowy.

Dwie liczby a i b są swoim lustrzanym odbicie, jeśli jedna z liczb powstaje poprzez odwrócenie kolejności cyfr z drugiej liczby. Przykład 345 i 543 są swoim lustrzanym odbicie.

13. Napisz program dokonujący rozkładu danej dodatniej liczby naturalnej n na czynniki pierwsze. Na przykład dla $n = 24$ sa to czynniki 2 2 2 3. Liczba n ma być pobrana ze standardowego wejścia, rozkład wyświetlony na standardowym wyjściu.

14. Napisz program obliczający tzw. jednocyfrową sumę cyfr liczby n. Na początku oblicza się sumę cyfr liczby n; jeśli wynikiem jest liczba wielocyfrowa, to znowu oblicza się sumę cyfr tej poprzedniej sumy i tak powtarza, aż do uzyskania liczby jednocyfrowej, która jest wynikiem końcowym. Na przykład dla $n = 48$ suma jednocyfrowa wynosi 3. Liczba n ma być pobrana ze standardowego wejścia, wynik końcowy wyświetlony na standardowym wyjściu.

15. Napisz funkcję, której argumentem jest jest dodatnia liczba całkowita n. Funkcja ma wyświetlać wszystkie możliwe liczby Armstronga mniejsze lub równe n (bez rozkładów). Stwórz przypadek testowy dla funkcji. W zadaniu nie korzystaj ze wbudowanych funkcji matematycznych.

Liczba Armstronga to dodatnia liczba naturalna, której suma sześcianów poszczególnych cyfr jest równa tej liczbie. Przykładowo: $153 = 1^3 + 5^3 + 3^3$.

16. Napisz funkcję, której argumentem jest dodatnia liczba całkowita n. Funkcja ma zwrócić ile liczb całkowitych dodatnich mniejszych lub równych n ma cyfry, które stanowią palindrom. Stwórz przypadek testowy. W zadaniu nie korzystaj z konwersji na napis.

Przykłady:

- liczba 12321 jest palindromem
- liczby jednocyfrowe są palindromami
- liczba 4556 nie jest palindromem
- liczba 44 jest palindromem

17. Napisz funkcję, która ma dwa argumenty: dwie dodatnie liczby całkowite m i n ($m < n$). Funkcja ma zwrócić ile liczb pierwszych jest w przedziale $[2m, 3n]$. Stwórz przypadek testowy dla funkcji.

18. Napisz funkcję `czyBliskie`, której argumentami są trzy liczby wymierne x, y, ε. Funkcja ma zwrócić odpowiednią wartość logiczną (zero lub jeden) po sprawdzeniu czy wartość bezwzględna różnicy x i y jest mniejsza od ε. Następnie pobierz od użytkownika 5 liczb wymiernych i wyświetl informację, ile z nich jest bliskich spośród wprowadzonych wzajemnie między sobą dla $\varepsilon = 2$. Przykładowy komunikat na koniec:

```
Liczba 1: ... Ile liczb bliskich: ..
Liczba 2: ... Ile liczb bliskich: ..
Liczba 3: ... Ile liczb bliskich: ..
Liczba 4: ... Ile liczb bliskich: ..
Liczba 5: ... Ile liczb bliskich: ..
```

Bibliografia i inne zbiory zadań

Krzaczkowski, Jacek. 2011. *Zadania z programowania w języku C/C++, cz. I.* Instytut Informatyki UMCS Lublin.

Historia zmian

- 17.02.2024 - dodano część: Formatowanie zmiennych liczbowych w `printf`, Operatory arytmetyczne, Operatory bitowe
- 25.02.2024 - dodano część: Instrukcje warunkowe, operator warunkowy, Pętle
- 3.03.2024 - dodano część: Funkcje
- 10.03.2024 - dodano część: Wskaźniki
- 11.03.2024 - dodano link do repozytorium na wybrane rozwiązania
- 17.03.2024 - dodano część: Wskaźniki na funkcję, Debugowanie
- 24.03.2024 - dodano część: Tablice
- 4.04.2024 - dodano część: Napisy
- 25.04.2024- dodano część: Tablice wielowymiarowe
- 5.05.2024- dodano część: Złożone typy danych
- 18.05.2024 - dodano część: Listy jednokierunkowe
- 23.02.2025 - odświeżenie styli
- 2.03.2025 - drobne poprawki
- 6.03.2025 - drobne poprawki
- 9.03.2025 - dodanie nowych zadań w podstaw debugowania
- 16.03.2025 - dodane nowe zadania dot. pętli i zastosowania wybranych algorytmów liczbowych
- 3.04.2025 - dodanie nowych zadań dot. wskaźników na funkcję
- 6.04.2025 - dobre poprawki w zadaniach z tablicami
- 28.04.2025 - dodano dwa nowe zadania dot. napisów