Zadania na plusy

Spis treści

Zasady zdobywania i liczenia się plusów

Plusa (ew. plusy) za zrobione zadanie może dostać **tylko jedna osoba**, ta, które pierwsza prześle prawidłowe rozwiązanie na adres mailowy: rperlinski.pcz@gmail.com.
Tytuł maila: PP_<nazwisko>_ZDod_<numer_zadania>, np. PP_Kowalski_ZDod_01.
Najlepiej przesłać kod źródłowy w załączniku.

Przesłane zadanie, jeśli będzie poprawnie działać i będzie nadesłane jako pierwsze, będzie trzeba mi objaśnić po wtorkowych zajęciach, dla sprawdzenia autorstwa i/lub dokładnego zrozumienia kodu mi przesłanego.

Dwa plusy pozwalają na jedno nieprzygotowanie więcej do zajęć. Jeśli ktoś nie ma żadnych nieprzygotowań do zajęć, to plusy przechodzą na dodatek do oceny na koniec z przedmiotu. Każdy plus to +0.1 do oceny na koniec. Jedna osoba **może zdobyć maksymalnie 3 plusy**. Trudne zadania (przy czym „trudne” to pojęcie względne) mogą być od razu na dwa plusy.

Zad01 – 1 plus (już ZREALIZOWANE)

Proszę napisać zadanie 2 z laboratorium 2 (odwrócona pół-choinka) **z wykorzystaniem tylko jednej pętli, bez wykorzystania rekurencji**. Całość rozwiązania powinna mieścić się w jednej funkcji, w której jest tylko jedna pętla i nie ma żadnych wywołań innych funcji. Da się to dość łatwo zrobić. Wykorzystanie instrukcji warunkowych czy różnych operatorów matematycznych bez ograniczeń.

Zad02 – 2 plusy (już ZREALIZOWANE)

Dane jest rozwiązanie łamigłówki sudoku, tablica 9x9. Należy napisać funkcję, które sprawdzi poprawność jej rozwiązania (każdy wiersz, kolumna i mały kwadrat 3x3 powinien zawierać wszystkie liczby od 1 do 9, czyli żadna z liczb nie powinna się powtarzać). Funkcja powinna zwracać wartość `true` jeśli rozwiązanie jest poprawne albo wartość `false` jeśli jest niepoprawne. W tym drugim przypadku powinna wyświetlać informację o położeniu znalezionej powtórzonej liczby.

Napisaną funkcję proszę sobie przetestować dla takich poniższych trzech tablic:

```
int t1[9][9] = {
  {4,8,9,  5,3,1,  6,2,7},
  {7,5,2,  6,4,9,  8,1,3},
  {3,1,6,  8,2,7,  5,9,4},

  {2,3,8,  4,9,6,  1,7,5},
  {5,4,7,  2,1,8,  3,6,9},
  {6,9,1,  7,5,3,  2,4,8},

  {1,6,5,  3,7,4,  9,8,2},
  {9,2,4,  1,8,5,  7,3,6},
  {8,7,3,  9,6,2,  4,5,1},
};
```

```
int t2[9][9] = {
  {4,8,9,  5,3,1,  6,2,7},
  {7,5,2,  6,4,9,  8,1,3},
  {3,1,6,  8,2,7,  5,9,4},

  {2,3,8,  4,9,6,  1,7,5},
  {5,4,7,  2,1,8,  3,6,9},
  {6,9,1,  7,5,3,  2,4,8},

  {1,6,5,  3,7,4,  9,1,2},
  {9,2,4,  1,8,5,  7,3,6},
  {8,7,3,  9,6,2,  4,5,1},
};
```

```
int t3[9][9] = {
  {4,8,5,  9,3,1,  6,2,7},
  {7,5,2,  6,4,9,  8,1,3},
  {3,1,6,  8,2,7,  9,5,4},

  {2,3,8,  4,9,6,  1,7,5},
  {5,4,7,  2,1,8,  3,6,9},
  {6,9,1,  7,5,3,  2,4,8},

  {1,6,9,  3,7,4,  5,8,2},
  {9,2,4,  1,8,5,  7,3,6},
  {8,7,3,  5,6,2,  4,9,1},
};
```

Kod testujący i przykładowy wyświetlany wynik:

```
cout << "################ Sprawdzanie t1" << endl;
cout << sprawdz_sudoku(t1) << endl;
cout << "################ Sprawdzanie t2" << endl;
cout << sprawdz_sudoku(t2) << endl;
cout << "################ Sprawdzanie t3" << endl;
cout << sprawdz_sudoku(t3) << endl;
```

```
############### Sprawdzanie t1
### sprawdzanie wierszy
### sprawdzanie kolumn
### sprawdzanie małych kwadratów
1
############### Sprawdzanie t2
### sprawdzanie wierszy
Powtórzona liczba na pozycji 7x8.
### sprawdzanie kolumn
Powtórzona liczba na pozycji 7x8.
### sprawdzanie małych kwadratów
Powtórzona liczba (kwadrat 3x3) na pozycji 9x9.
0
############### Sprawdzanie t3
### sprawdzanie wierszy
### sprawdzanie kolumn
### sprawdzanie małych kwadratów
Powtórzona liczba (kwadrat 1x1) na pozycji 2x2.
Powtórzona liczba (kwadrat 1x2) na pozycji 2x6.
Powtórzona liczba (kwadrat 3x1) na pozycji 8x1.
Powtórzona liczba (kwadrat 3x2) na pozycji 9x4.
0
```

Zad03 – 1 plus

Należy zaimplementować funkcję, która przyjmuje jeden parametr, liczbę zmiennopozycyjną pojedynczej precyzji (typ float) i wyświetla binarną postać tej liczby z odpowiednim opisem (z – bit znaku, c – bity cechy, m – bity mantysy liczby). Program powinien po prostu wyświetlić binarną postać liczby zmiennopozycyjnej. Zakładam tutaj, że typ float ma 4 bajty (32 bity). Warunek ten powinien być spełniony dla większości dzisiejszych komputerów. Zob. https://pl.wikipedia.org/wiki/IEEE_754. Przykładowy kod testujący i przykład działania poniżej.

```
cout << " ujemne, malejący wykładnik" << endl;
cout << "========================================" << endl;
float_binarnie(-8.0);
float_binarnie(-4.0);
float_binarnie(-2.0);
float_binarnie(-1.0);
float_binarnie(-0.0);
float_binarnie(0.0);

cout << " dodatnie, rosnący wykładnik, mantysa bez zmian" << endl;
cout << "========================================" << endl;
float_binarnie(0.125);
float_binarnie(0.25);
float_binarnie(0.5);

cout << " dodatnie, stały wykładnik, mantysa się zwiększa" << endl;
cout << "========================================" << endl;
float_binarnie(0.75);
float_binarnie(0.875);
float_binarnie(0.9375);

cout << " dodatnie, rosnący wykładnik" << endl;
cout << "========================================" << endl;
float_binarnie(1.0);
float_binarnie(2.0);
float_binarnie(4.0);
float_binarnie(8.0);
float_binarnie(16.0);
```

```
 ujemne, malejący wykładnik
========================================
        wartość float: -8.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        1 10000010 00000000000000000000000

        wartość float: -4.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        1 10000001 00000000000000000000000

        wartość float: -2.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        1 10000000 00000000000000000000000

        wartość float: -1.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        1 01111111 00000000000000000000000

        wartość float: -0.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        1 00000000 00000000000000000000000

        wartość float: 0.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 00000000 00000000000000000000000

 dodatnie, rosnący wykładnik, mantysa bez zmian
========================================
        wartość float: 0.125000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111100 00000000000000000000000

        wartość float: 0.250000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111101 00000000000000000000000

        wartość float: 0.500000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111110 00000000000000000000000

 dodatnie, stały wykładnik, mantysa się zwiększa
========================================
        wartość float: 0.750000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111110 10000000000000000000000

        wartość float: 0.875000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111110 11000000000000000000000

        wartość float: 0.937500
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111110 11100000000000000000000

 dodatnie, rosnący wykładnik
========================================
        wartość float: 1.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 01111111 00000000000000000000000

        wartość float: 2.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 10000000 00000000000000000000000

        wartość float: 4.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 10000001 00000000000000000000000

        wartość float: 8.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 10000010 00000000000000000000000

        wartość float: 16.000000
        z cccccccc mmmmmmmmmmmmmmmmmmmmmmm
        0 10000011 00000000000000000000000
```

Zad04 – 1 plus

Należy rozwinąć zadanie 9 z laboratorium 3. Proszę napisać funkcję przyjmującą dwa argumenty:

- nieujemną liczbę całkowitą zapisaną w systemie dziesiętnym,
- nieujemną liczbę całkowitą oznaczającą podstawę systemu liczbowego,

która zamieni podaną liczbę z systemu dziesiętnego na liczbę w podanym innym systemie liczbowym. Funkcja powinna obsługiwać systemy z zakresu [2,36] czyli od systemu binarnego, poprzez trójkowy, …, ósemkowy, dziesiętny, szesnastkowy, … aż do systemu trzydziestoszóstkowego (jakoś tak). Cyfry od 10 do 36 oznaczamy kolejnymi literami alfabetu łacińskiego. Można wykorzystać poniższą tablicę cyfry:

```
char cyfry[37] = "0123456789abcdefghijklmnopqrstuvwxyz";
for(int i =0; i<36; ++i) {
   cout << setw(2) << i << ": " << cyfry[i] << "  ";
   cout << ((i%6==5)?"\n":"");
}
```

```
 0: 0   1: 1   2: 2   3: 3   4: 4   5: 5
 6: 6   7: 7   8: 8   9: 9  10: a  11: b
12: c  13: d  14: e  15: f  16: g  17: h
18: i  19: j  20: k  21: l  22: m  23: n
24: o  25: p  26: q  27: r  28: s  29: t
30: u  31: v  32: w  33: x  34: y  35: z
```

Kod testujący i wynik działania (liczba 63 w kolejnych systemach liczbowych):

```
for(int sys_licz=2; sys_licz<=36; ++sys_licz) {
  systemowo(63,sys_licz);
}
```

```
    63 (10)
111111 (2)

  63 (10)
2100 (3)

 63 (10)
333 (4)

 63 (10)
223 (5)

 63 (10)
143 (6)

 63 (10)
120 (7)

63 (10)
77 (8)

63 (10)
70 (9)
```

```
63 (10)
63 (10)

63 (10)
58 (11)

63 (10)
53 (12)

63 (10)
4b (13)

63 (10)
47 (14)

63 (10)
43 (15)

63 (10)
3f (16)

63 (10)
3c (17)

63 (10)
39 (18)
```

```
63 (10)
36 (19)

63 (10)
33 (20)

63 (10)
30 (21)

63 (10)
2j (22)

63 (10)
2h (23)

63 (10)
2f (24)

63 (10)
2d (25)

63 (10)
2b (26)

63 (10)
29 (27)
```

```
63 (10)
27 (28)

63 (10)
25 (29)

63 (10)
23 (30)

63 (10)
21 (31)

63 (10)
1v (32)

63 (10)
1u (33)

63 (10)
1t (34)

63 (10)
1s (35)

63 (10)
1r (36)
```

Zad05 – 3 plusy (wydaje mi się dość trudne dlatego 3)

Dana jest łatwa krzyżówka sudoku zapisana w postaci dwuwymiarowej tablicy zmiennych typu int – tablica t1. Należy napisać funkcję rozwiaz_sudoku(), która rozwiąże sudoku zastępując w tabeli t1 wszystkie zera odpowiednimi liczbami. Kod testujący i przykład działania:

```
int t1[9][9] = {
  {0,0,4,  0,0,9,  2,6,0},
  {0,0,0,  0,0,3,  0,0,1},
  {0,3,0,  0,8,0,  0,9,4},

  {0,0,0,  0,0,0,  7,0,2},
  {2,0,5,  0,3,0,  9,0,8},
  {7,0,8,  0,0,0,  0,0,0},

  {8,7,0,  0,2,0,  0,5,0},
  {9,0,0,  7,0,0,  0,0,0},
  {0,5,3,  8,0,0,  1,0,0},
};

wypisz_sudoku(t1);
rozwiaz_sudoku(t1);
wypisz_sudoku(t1);
```

```
+-------+-------+-------+
| . . 4 | . . 9 | 2 6 . |
| . . . | . . 3 | . . 1 |
| . 3 . | . 8 . | . 9 4 |
+-------+-------+-------+
| . . . | . . . | 7 . 2 |
| 2 . 5 | . 3 . | 9 . 8 |
| 7 . 8 | . . . | . . . |
+-------+-------+-------+
| 8 7 . | . 2 . | . 5 . |
| 9 . . | 7 . . | . . . |
| . 5 3 | 8 . . | 1 . . |
+-------+-------+-------+
Zmian w sumie: 52
+-------+-------+-------+
| 1 8 4 | 5 7 9 | 2 6 3 |
| 5 2 9 | 4 6 3 | 8 7 1 |
| 6 3 7 | 1 8 2 | 5 9 4 |
+-------+-------+-------+
| 3 6 1 | 9 5 8 | 7 4 2 |
| 2 4 5 | 6 3 7 | 9 1 8 |
| 7 9 8 | 2 1 4 | 6 3 5 |
+-------+-------+-------+
| 8 7 6 | 3 2 1 | 4 5 9 |
| 9 1 2 | 7 4 5 | 3 8 6 |
| 4 5 3 | 8 9 6 | 1 2 7 |
+-------+-------+-------+
```

Przykład sudoku do rozwiązania wziąłem ze strony: http://pl.sudokuonline.eu/.
W moim rozwiązaniu napisałem sobie trzy funkcje pomocnicze (do rozwiązania często wystarczą dwie):

- rozwiaz_sudoku_wiersze() – sprawdza możliwe do wpisania liczby z punktu widzenia wiersza
- rozwiaz_sudoku_kolumny() – sprawdza możliwe do wpisania liczby z punktu widzenia kolumny
- rozwiaz_sudoku_pola() – sprawdza możliwe liczby dla każdego pola osobno, jeśli jest tylko jedna możliwa to ją dopisuje, np. liczba 5 w wierszu 3, kolumnie 7.

Każda z nich inaczej analizuje układ liczb w krzyżówce i dopisuje jednoznacznie określone liczby w odpowiednie miejsca. I tak aż do rozwiązania. Ktoś z Państwa może to zrobić zapewne trochę inaczej, liczy się poprawne działanie. Można sobie też testować na innych układach liczb, na innych krzyżówkach sudoku.

Zad06 – 1 plus (już ZREALIZOWANE)

Poniższy kod obrazuje wykorzystanie dwuwymiarowej dynamicznej tablicy liczb całkowitych. Trzeba napisać funkcję, która wyświetli na ekranie ładnie zobrazowane:

- elementy umieszczone w tej tablicy,
- adresy w pamięci:
  - w którym jest zmienna przechowująca tę tablicę,
  - w których są umieszczone elementy tej tablicy,
  - w których jest sama tablica.

Dla poniższego przykładu dla dwuwymiarowej tablicy dynamicznej mamy zarezerwowane:

- 1 wskaźnik na tablicę wskaźników,
- 4-elementowa tablica wskaźników (4 wskaźniki typu int*),
- cztery 7-elementowe tablice typu int.

W sumie daje to 28 komórek pamięci typu int i 5 komórek pamięci typu int*.

```
#include <iostream>
#include <iomanip>
using namespace std;

// Mamy dwuwymiarową tablicę dynamiczną o rozmiarze 4x7 (4 wiersze, 7 kolumn)

int main(int argc, char const *argv[]) {
  int lw = 4;
  int lk = 7;

  // REZERWOWANIE PAMIĘCI
  // alokacja pamięci: jednowymiarowa tablica zawierająca wskaźniki
  int **tab = new int*[lw];
  // alokacja pamięci: lw jednowymiarowych tablic z liczbami całkowitymi
  for(int i=0; i<lw; ++i) {
    tab[i] = new int[lk];
  }

  // UŻYCIE TABLICY
  // robimy sobie prostą tabliczkę mnożenia
  for(int i=0; i<lw; ++i) {
    for(int j=0; j<lk; ++j) {
      tab[i][j] = (i+1)*(j+1);
    }
  }
  // wyświetlamy naszą tabliczkę mnożenia
  for(int i=0; i<lw; ++i) {
    for(int j=0; j<lk; ++j) {
      cout << setw(4) << tab[i][j];
    }
    cout << endl;
  }

  // ZWALNIANIE PAMIĘCI
  // Zwalnianie kolejnych wierszy tablicy dwuwymiarowej
  for(int i=0; i<lw; ++i) {
    delete [] tab[i];
  }
  // Zwalnianie tablicy ze wskaźnikami na kolejne wiersze,
  // to trochę jakby pojedyncza kolumna, w której każdy element wskazuje na cały jeden wiersz
  delete [] tab;
  tab = NULL;
  return 0;
}
```

Kod testujący:

```
  cout << "Trochę informacji o zajmowanej pamięci przez typ ini i int*" << endl;
  cout << "sizeof(int): " << sizeof(int) << endl;
  cout << "sizeof(int*): " << sizeof(int*) << endl;
  cout << "sizeof(int**): " << sizeof(int**) << endl;
  cout << "sizeof(void*): " << sizeof(void*) << endl;
  cout << "sizeof(void**): " << sizeof(void**) << endl;
  cout << "Zmienna typu int zajmuje " << sizeof(int) << " bajtów pamięci." << endl;
  cout << "Każdy wskaźnik zajmuje " << sizeof(int*) << " bajtów pamięci." << endl << endl;

  wyswietl_tablice(tab, lw, lk);
```

Przykład działania:

```
Trochę informacji o zajmowanej pamięci przez typ ini i int*
sizeof(int): 4
sizeof(int*): 8
sizeof(int**): 8
sizeof(void*): 8
sizeof(void**): 8
Zmienna typu int zajmuje 4 bajtów pamięci.
Każdy wskaźnik zajmuje 8 bajtów pamięci.

Tablica dwuwymiarowa ma 4 wierszy i 7 kolumn.
Cała tablica (tab) jest pod adresem 0x7ffc99345df8 i przechowuje wskaźnik o wartości 0x20d3c20

Tablica (tab[i]) przechwująca wskaźniki na poszczególne wiersze tablicy dwuwymiarowej:
   Adresy:    0x20d3c20   0x20d3c28   0x20d3c30   0x20d3c38
Zawartość:    0x20d3c50   0x20d3c80   0x20d3cb0   0x20d3ce0

Adresy i zawartości poszczególnych wierszy tabeli dwuwymiarowej (tab[i][j]):
    Adres:    0x20d3c50    0x20d3c54    0x20d3c58    0x20d3c5c    0x20d3c60    0x20d3c64    0x20d3c68
Zawartość:            1            2            3            4            5            6            7
    Adres:    0x20d3c80    0x20d3c84    0x20d3c88    0x20d3c8c    0x20d3c90    0x20d3c94    0x20d3c98
Zawartość:            2            4            6            8           10           12           14
    Adres:    0x20d3cb0    0x20d3cb4    0x20d3cb8    0x20d3cbc    0x20d3cc0    0x20d3cc4    0x20d3cc8
Zawartość:            3            6            9           12           15           18           21
    Adres:    0x20d3ce0    0x20d3ce4    0x20d3ce8    0x20d3cec    0x20d3cf0    0x20d3cf4    0x20d3cf8
Zawartość:            4            8           12           16           20           24           28
```

Widać powyżej jak poszczególne wskaźniki (ten główny w zmiennej tab, i te w tablicy ze wskaźnikami tab[i]) wskazują na poszczególne komórki pamięci. tab wskazuje na adres ***c20, pod którym jest tablica 4 wskaźników, wskazujących na 4 tablice liczb typu int.

Zad07 – 1 plus

Jest to rozwinięcie funkcji `wstaw()` z zadania 3 laboratorium 4/5. Należy wstawić napis podany jako drugi parametr do napisu podanego jako pierwszy parametr w miejsce podane jako trzeci parametr. W przypadku podania przy uruchamianiu mniej niż trzech parametrów, należy zwrócić informację o ich niedostatecznej liczbie. Wersja z tego zadania ma działać z określaniem miejsca wstawiania za pomocą:

- liczby dodatniej, oznacza liczenie pozycji od początku, 0 oznacza wstawianie przed napisem
- liczby ujemnej, oznacza wstawianie od końca, wartość -1 oznacza że wstawiamy za ostatnim znakiem, wstawianą pozycję odliczamy od tyłu napisu.

Jeśli podana pozycja wstawiania wyjdzie już poza zakres (długość napisu, do którego wstawiamy), to nic nie robimy, najwyżej zgłaszamy, że pozycja jest niewłaściwa. Jeśli program rezerwuje jakąś pamięć to należy ją zwolnić przed zakończeniem programu. Przykładowe działanie dla różnych parametrów uruchamiania obok:

```
$ ./wstaw "Podstawy programowania" -ABC- 0
-ABC-Podstawy programowania

$ ./wstaw "Podstawy programowania" -ABC- 1
P-ABC-odstawy programowania

$ ./wstaw "Podstawy programowania" -ABC- 4
Pods-ABC-tawy programowania

$ ./wstaw "Podstawy programowania" -ABC- 12
Podstawy pro-ABC-gramowania

$ ./wstaw "Podstawy programowania" -ABC- 18
Podstawy programow-ABC-ania

$ ./wstaw "Podstawy programowania" -ABC- 22
Podstawy programowania-ABC-

$ ./wstaw "Podstawy programowania" -ABC- 23
Poza zakresem
Podstawy programowania

$ ./wstaw "Podstawy programowania" -ABC- -1
Podstawy programowania-ABC-

$ ./wstaw "Podstawy programowania" -ABC- -3
Podstawy programowan-ABC-ia

$ ./wstaw "Podstawy programowania" -ABC- -9
Podstawy progr-ABC-amowania

$ ./wstaw "Podstawy programowania" -ABC- -22
P-ABC-odstawy programowania

$ ./wstaw "Podstawy programowania" -ABC- -23
-ABC-Podstawy programowania

$ ./wstaw "Podstawy programowania" -ABC- -24
Od ko?ca napisu ale poza zakresem
Podstawy programowania

$ ./wstaw "Podstawy programowania" -ABC-
Za mało parametrów - ko?cz? program

$ ./wstaw "Podstawy programowania"
Za mało parametrów - ko?cz? program
```

Zad08 – 2 plusy

Mamy tutaj taką małą analizę sytuacji w grze kółko i krzyżyk. Danych jest kilka sytuacji z gry i należy napisać odpowiednią funkcję, która określi, czy dana sytuacja jest zgodna z zasadami gry. Aby sytuacja na planszy była zgodna z zasadami gry muszą być spełnione dwa warunki:

- na planszy musi być równomierna liczba symboli obu graczy:
  - albo jest tyle samo O i X,
  - albo jeden z graczy (ten, który zaczynał) ma o jeden symbol więcej, ale tylko o jeden,
- na planszy nie może wystąpić więcej niż jedna linia prosta z trzech takich samych symboli, tzn. po wygraniu jednego z graczy drugi już nie rysuje swojego O czy X.

Zakładam tutaj, że obaj gracze muszą wykonać ruch w swojej turze. Będzie trzeba sobie tutaj napisać jakieś funkcje pomocnicze. Przykładowy kod testujący obok, wynik działania poniżej. Zero to puste pole, jeden to gracz O, dwa to gracz X.

```
Pozycja zgodna z zasadami gry
 . . .
 . . .
 . . .
Pozycja zgodna z zasadami gry
 o . x
 o x .
 o x o
Nierównomierna liczba X i O na planszy, pozycja niezgodna z zasadami
 . . x
 x o .
 x o x
Więcej niż jedna linia wygrywająca na planszy, pozycja niezgodna z zasadami
 x o .
 x o .
 x o x
Więcej niż jedna linia wygrywająca na planszy, pozycja niezgodna z zasadami
 x o o
 x x o
 x o x
Nierównomierna liczba X i O na planszy, pozycja niezgodna z zasadami
Więcej niż jedna linia wygrywająca na planszy, pozycja niezgodna z zasadami
 x o o
 x . o
 x o o
```

```
int p1[3][3] = {
  {0,0,0},
  {0,0,0},
  {0,0,0}
};
int p2[3][3] = {
  {1,0,2},
  {1,2,0},
  {1,2,1}
};
int p3[3][3] = {
  {0,0,2},
  {2,1,0},
  {2,1,2}
};
int p4[3][3] = {
  {2,1,0},
  {2,1,0},
  {2,1,2}
};
int p5[3][3] = {
  {2,1,1},
  {2,2,1},
  {2,1,2}
};
int p6[3][3] = {
  {2,1,1},
  {2,0,1},
  {2,1,1}
};

zgodna_z_zasadami(p1);
plansza_xo(p1);
zgodna_z_zasadami(p2);
plansza_xo(p2);
zgodna_z_zasadami(p3);
plansza_xo(p3);
zgodna_z_zasadami(p4);
plansza_xo(p4);
zgodna_z_zasadami(p5);
plansza_xo(p5);
zgodna_z_zasadami(p6);
plansza_xo(p6);
```