Poradnik jak zdać PP (100% legit no scam)

Ważne uwagi

Jeszcze zanim zacznę przerabiać te zadania informuję, że czasami wymagają rzeczy które wymagają takiego pierdolenia się z nimi a są warte tak mało punktów, że nawet nie będę podejmował się tłumaczenia tego bo po prostu nie ma sensu. Do takich rzeczy należy:

- sprawdzanie poprawności danych (pewien Karol za to stracił chyba 8 punktów ze 100 więc jebać)

- poprawna odmiana słów (giga sraka, trzeba olać)

Podstawy podstaw, czyli podstawowy template kodu do kolosa

Tak się zaczyna pisanie kodu i to tyle:

```
#include <iostream> // Importowanie biblioteki

using namespace std; // Wybranie namespace'a

int main()
{
    // Jakaś zawartość funkcji main
    return 0;
    // Kod wyjścia programu będzie 0 (0 oznacza, że wszystko było git)
}
```

Funkcje różnej maści

To jest funkcja zwracająca sumę wyrazów x i y. x oraz y są argumentami funkcji przyjmującymi liczby całkowite. Funkcja ta zwraca jakąś liczbę zmienną przecinkową. Proste i logiczne.

```
float suma(int x, int y)
{
    return x + y;
}
```

Jeśli jakaś funkcja nie zwraca nam żadnej wartości to przy deklarowaniu funkcji powinniśmy napisać, że zwraca void. Przykład funkcji suma, która nic nie zwraca:

```
void suma(int x, int y)
{
    cout << x + y << endl;
}
```

W przypadku funkcji return nie tylko może posłużyć nam do zwrócenia wartości przez funkcje, ale również do wyjścia z funkcji.

```
void jakas_funkcja(int x)
{
    int y = 0;
    while(1) // nieskończona pętla while
    {
        cout << y++ << endl;
        if (y == x) return;
    }
}
```

Funkcja ta wypisze wartości od 0 do (x-1) ponieważ, gdy y == x to funkcja się zakończy. Działa to trochę jak break dla pętli. W tym wypadku można by było dać break i by zadziałało tak samo ale wiadomo ocb.

Rekurencja i iteracja

Funkcja rekurencyjna to taka funkcja, która wykonuje sama siebie.

Funkcja iteracyjna to taka funkcja, która korzysta z pętli i nie wykonuje sama siebie.

Nie lubię funkcji rekurencyjnych bo o wiele ciężej się je piszę, ale spokojnie jest na to sposób. Przedstawię ten sposób rozumowania na podstawie funkcji liczącej silnie oraz ciąg Fibonacciego.

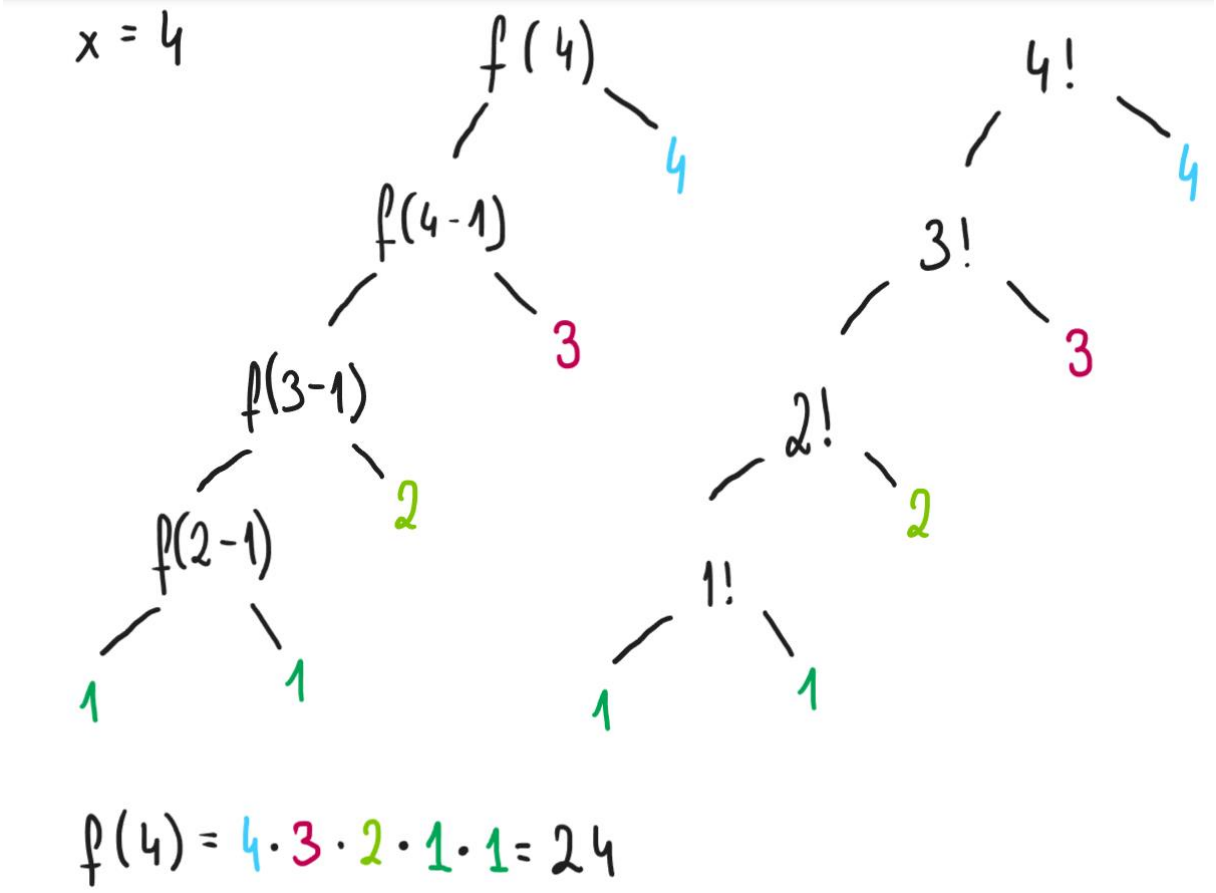
Silnia iteracyjna:

```
int silnia_iteracyjnie(int x)
{
    if (x == 0 || x == 1) return 1;
    int wynik = 1;
    for (int i = 2; i <= x; i++)
    {
        wynik *= i;
    }
    return wynik;
}
```

Silnia rekurencyjnie:

```
int silnia_rekurencyjnie(int x)
{
    if (x == 0 || x == 1) return 1;
    return (silnia_rekurencyjnie(x - 1) * x);
}
```

Skąd takie gówno, już tłumacze:
Zawsze przy rekurencji mamy jakąś wartość bazową do której musimy dążyć. W tym przypadku jest to x == 0 lub x == 1 dające nam 1. W każdej rekurencji musi być taka wartość bazowa bo jakbyśmy chcieli, żeby funkcja mogła zwracać tylko i wyłącznie samą siebie to nigdy by nic nie zwróciła.
Jako iż silnia to iloczyn liczb od 1 do x to po prostu mnożymy sobie wartość x przez silnie x-1. Szybkie rozpisanie tej kongowni (zmienię tą długą nazwę na po prostu f):



Ciąg Fibonacciego:

Iteracyjnie

```
int fibonacci_iteracyjnie(int x)
{
    if (x == 0) return 0;
    else if (x == 1) return 1;
    int a = 0;
    int b = 1;
    for (int i = 2; i <= x; i++)
    {
        b = a + b;
        a = b - a;
    }
    return b;
}
```

Rekurencyjnie:

```
int fibonacci_rekurencyjnie(int x)
{
if (x == 0) return 0;
else if (x == 1) return 1;
else
{
return fibonacci_rekurencyjnie(x - 1) + fibonacci_rekurencyjnie(x - 2);
}
}
```

Ciąg Fibonacciego (F_n), to ciąg (określony w zbiorze wszystkich liczb naturalnych), którego wyrazami są kolejne liczby Fibonacciego:

$$0, 1, 1, 2, 3, 5, 8, 13, \ldots$$

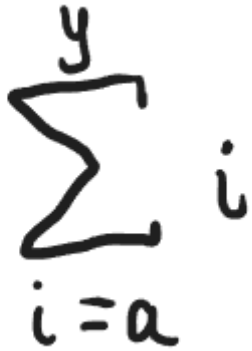
n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	...

$$5 + 8 = 13$$

Bazowe wartości do których dążymy to x == 0 zwracające zero oraz x == 1 zwracające jeden. Dla np. n = 2, wartość ciągu będzie po prostu sumą 2 poprzednich czyli wartości n-1 oraz n-2. Tak samo działa to dla każdego kolejnego. Nie będę tego rozrysowywał bo mi się nie chce.

Zadanie z sigmą

Nie jestem mistrzem pisania, ale jest takie przykładowe zadanie (proste w chuj)



To mniej więcej chodzi o to. Należy pamiętać, że przy sigmie „i” jest mniejsze bądź równe wartości na górze (w tym przypadku y).

```
int sigma_suma(int y)
{
    int suma;
    for(int i = a; i <= y; i++)
    {
        suma += i;
    }
    return suma;
}
```

Inicjowanie zmiennych

Tutaj nie ma zbytnio co się rozpisywać

```
int main()
{
    int x; // można nie przypisywać wartości
    int a = 0; // można przypisać wartość
    int b = 1, c, d = 5; // można zainicjować kilka zmiennych w tej samej linice
    string napis = "Ala ma kota"; // string to tablica charów ale taka lepsza
    char napis2[11] = "Kot ma Ale"; // og tablica charów (chujowe do używania ale trzeba)
    return 0;
}
```

Zmienne stałe (przyjmuje się, że nazwy stałych piszę się tylko wielkimi literami):

```
int main()
{
    const int STALA = 123;
    return 0;
}
```

Tablice (po prawo wyniki kodu):

```
int main()
{
    int tab[10];
    for (int i = 0; i < 10; i++)
    {
        cout << i << " : " <<  tab[i] << endl;
    }
    return 0;
}
```

```
0 : 8
1 : 0
2 : 4199705
3 : 0
4 : 8
5 : 0
6 : 37
7 : 0
8 : 7673584
9 : 0
```

Aby tablica została wyzerowana, trzeba ją zainicjować tak:

```
int main()
{
    int tab[10] = {0};
    for (int i = 0; i < 10; i++)
    {
        cout << i << " : " <<  tab[i] << endl;
    }
    return 0;
}
```

```
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 0
9 : 0
```

Tabelka z wyrównaniem do prawej strony

```cpp
#include <iostream>
#include <iomanip> // to ważne jest

using namespace std;

int main()
{
    int tab[100] = {0};
    // tworzenie dużej tablicy
    for (int i = 0; i < 100; i++)
    {
        tab[i] = i;
    }

    // Wypisanie tabelki
    for (int i = 0; i < 100; i++)
    {
        if (i % 10 == 0) cout << endl;
        cout << "|";
        cout << setw(8)<< tab[i] << "|";
    }

    return 0;
}
```

```
|       0||       1||       2||       3||       4||       5||       6||       7||       8||       9
|      10||      11||      12||      13||      14||      15||      16||      17||      18||      19
|      20||      21||      22||      23||      24||      25||      26||      27||      28||      29
|      30||      31||      32||      33||      34||      35||      36||      37||      38||      39
|      40||      41||      42||      43||      44||      45||      46||      47||      48||      49
|      50||      51||      52||      53||      54||      55||      56||      57||      58||      59
|      60||      61||      62||      63||      64||      65||      66||      67||      68||      69
|      70||      71||      72||      73||      74||      75||      76||      77||      78||      79
|      80||      81||      82||      83||      84||      85||      86||      87||      88||      89
|      90||      91||      92||      93||      94||      95||      96||      97||      98||      99
Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

Sortowanie tablicy:

```
void sortowanie(int *tab, int rozmiar)
{
    int pomocnicza;
    for (int i = 0; i < rozmiar; i++)
    {
        for (int j = 1; j < rozmiar - i; j++)
        {
            if (tab[j] < tab[j - 1])
            {
                pomocnicza = tab[j];
                tab[j] = tab[j - 1];
                tab[j - 1] = pomocnicza;
            }
        }
    }
}
```

Losowanie liczby z przedziału <p, k>

```
#include <iostream>
// ! To są 2 ważne biblioteki
#include <cstdlib>
#include <time.h>

using namespace std;

int losuj(int p, int k)
{
    int liczba = (rand() % (k - p + 1)) + p;
    return liczba;
}

int main()
{
    srand(time(0)); // to jest bardzo ważne
    for (int i = 0; i < 10; i++)
    {
        cout << losuj(1, 5) << endl;
    }
    return 0;
}
```

Zamiana wartości zmiennych funkcją

Żeby móc w przy pomocy funkcji zmienić wartość zmiennej jaką przekazujemy jako argument trzeba użyć tak zwanej referencji (znak &).

```
#include <iostream>

using namespace std;

void zamiana(int &a, int &b)
{
    int c;
    c = a;
    a = b;
    b = c;
}

int main()
{
    int a = 5;
    int b = 1;
    cout << "a : " << a << endl;
    cout << "b : " << b << endl;
    zamiana(a, b);
    cout << "a : " << a << endl;
    cout << "b : " << b << endl;
    return 0;
}
```

Dynamiczne alokowanie tablic oraz ich usuwanie

```
int main()
{
    int rozmiar;
    cin >> rozmiar; //przyjmowanie rozmiaru tablicy
    int *tab = new int(rozmiar); //dynamiczne alokowanie tablicy
    /*
    robienie czegoś
    pierdu pierdu
    */
   delete []tab; // usuwanie tablicy
    return 0;
}
```

Dynamicznego alokowania tablicy używamy jeśli rozmiar tablicy nie jest znany na samym początku działania programu lub jeśli robimy sobie tablice w funkcji innej niż main. Za każdym razem gdy talica, którą stworzyliśmy nie jest nam już potrzebna to trzeba ją usunąć.

Dynamiczne alokowanie tablic 2D

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

int main()
{
    int wiersze;
    int kolumny;

    cin >> wiersze;
    cin >> kolumny;

    // Dynamiczne tworzenie dwuwymiarowej tablicy
    int** tab = new int*[wiersze]; //Najpierw tablica przyjmująca wskaźniki
    for (int i = 0; i < wiersze; i++) {

        tab[i] = new int[kolumny]; //Do każdego wskaźnika przypisujemy miejsce na określoną liczne intów
    }

    // Zerowanie tablicy
    for (int i = 0; i < wiersze; i++) {
        for (int j = 0; j < kolumny; j++) {
            tab[i][j] = 0;
        }
    }


    // Usuwanie tablicy
    for (int i = 0; i < wiersze; i++)
    {
        // Najpierw usuwamy te mniejsze tablice intów
        delete[] tab[i];
    }
    // Po usunięciu tablic intów usuwamy tablicę wskaźników
    delete[] tab;

    return 0;
}
```

Przykład dla struktur wpisywanych z palca przez użytkownika

```
#include <iostream>

using namespace std;

struct Ksiazka
{
    char tytul[20];
    int rok_wydania;
};

Ksiazka SetKsiazka(char* tytul, int rok_wydania)
{
    Ksiazka x;
    // Czyszcenie tablicy charów z tytułem
    for (int i = 0; i < 20; i++)
    {
        x.tytul[i] = NULL;
    }
    // Przypisowanie wartości do tablicy charów
    for (int i = 0; tytul[i]; i++)
    {
        x.tytul[i] = tytul[i];
    }
    x.rok_wydania = rok_wydania;

    return x;
}

void GetKsiazka(Ksiazka x)
{
    cout << "tytul: " << x.tytul << " - rok wydania " << x.rok_wydania
<< endl;
}
```

Reszta na kolejnej stronie

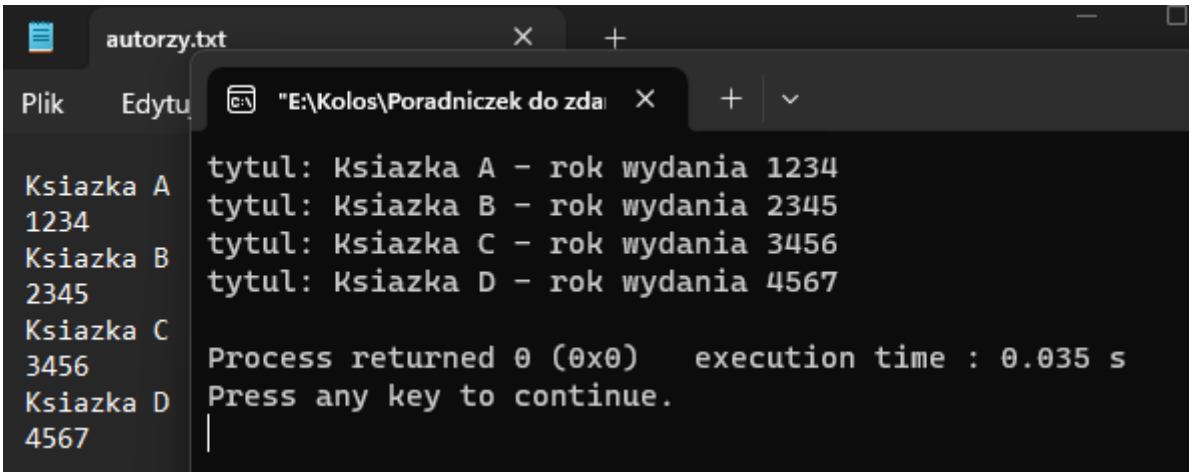
```cpp
int main()
{
    const int M = 4;
    Ksiazka Ksiazki[M];
    int rok;
    char tytul[20];

    for (int i = 0; i < M; i++)
    {
        cout << "Podaj dane " << i << " ksiazki: " << endl;
        cout << "Rok urodzenia: ";
        cin >> rok;
        cin.clear(); // te rzeczy daje na czuja troche
        cin.ignore(); // te rzeczy daje na czuja troche
        cout << "Tytul: ";
        cin.getline(tytul, 20);
        cin.clear(); // te rzeczy daje na czuja troche
        Ksiazki[i] = SetKsiazka(tytul, rok);
    }

    for (int i = 0; i < M; i++)
    {
        GetKsiazka(Ksiazki[i]);
    }

    return 0;
}
```

Struktury pobierane z pliku

```
#include <iostream>
#include <fstream>

// Dawid Gradowski 251524

using namespace std;

struct Ksiazka
{
    char tytul[20];
    int rok_wydania;
};

Ksiazka SetKsiazka(char* tytul, int rok_wydania)
{
    Ksiazka x;
    for (int i = 0; i < 20; i++)
    {
        x.tytul[i] = NULL;
    }
    for (int i = 0; tytul[i]; i++)
    {
        x.tytul[i] = tytul[i];
    }
    x.rok_wydania = rok_wydania;

    return x;
}

void GetKsiazka(Ksiazka x)
{
    cout << "tytul: " << x.tytul << " - rok wydania " << x.rok_wydania
<< endl;
}

int main()
{
    const int N = 3;
    const int M = 4;
    Ksiazka ksiazki[M];
    char numer[100];
    int rok;
    char tytul[20];
    ifstream plik;
    plik.open("autorzy.txt");
```

```
    // Sprawdza czy plik się otworzył czy nie
    // Jeśli nie to kończy program
    if(!plik.is_open())
    {
        cout << "Blad otwarcia pliku!!!" << endl;
        return 1;
    }

    for (int i = 0; i < M; i++)
    {
        // tak pobiera się tablice charow
        plik.getline(ksiazki[i].tytul, 20);
        // a tak jakies inty
        plik.getline(numer, 100);
        ksiazki[i].rok_wydania = atoi(numer);
    }

    for (int i = 0; i < M; i++)
    {
        GetKsiazka(ksiazki[i]);
    }

    // !! TRZEBA PAMIĘTAĆ O ZAMKNIĘCIU PLIKU !!
    plik.close();

    return 0;
}
```

Tak wyglądały dane w pliku i to co wypluł program:



Sortowanie alfabetyczne struktur na podstawie jakiegoś parametru

Cała podstawa skryptu działa tak jak to co było w poprzedniej sekcji. Sortowanie dla książek będzie wyglądało tak:

```
void Sortuj(Ksiazka * tab, int rozmiar)
{
    Ksiazka pomocnicza;
    for (int i = 0; i < rozmiar; i++)
    {
        for (int j = 1; j < rozmiar - i; j++)
        {
            if (strcmp(tab[j].tytul, tab[j - 1].tytul) < 0)
            {
                pomocnicza = tab[j];
                tab[j] = tab[j - 1];
                tab[j - 1] = pomocnicza;
            }
        }
    }
}
```

Funkcja strcmp jest z biblioteki string.h. Trzeba napisać #include <string.h> na początku.

Warto zauważyć, że nie zamieniamy tytułów miejscami a całe elementy, w tym przypadku całe książki. Aby odwrócić kolejność sortowania trzeba po prostu zamienić znak mniejszości na większości i jest gitara.

Operacje bitowe

Widziałem, że na kolokwiach były 2 bardzo podobne zadania które polegały praktycznie na tym samym. Jedno polegało na policzeniu ile jest 1 a drugie na policzeniu ile jest zarówno zer i jedynek. Jeśli dobrze interpretuje to coś to rozwiązanie tego jest praktycznie takie samo dla tych 2 zadań dlatego wytłumaczę to dla tego bardziej skomplikowanego.

5. Operacje bitowe

Napisz funkcję, która obliczy ilość bitów ustawionych na 0 i 1 w zmiennej typu `int`, przekazanej jako argument. Funkcja powinna spełniać następujące kryteria:

- przyjmuje jeden argument typu `int`,
- przyjmuje dwie referencje typu `int` (wartości wyjściowe), w których będzie zapisana ilość bitów ustawionych na 0 i 1,
- zlicza ilość bitów ustawionych na 1 za pomocą odpowiednich przesunięć i operacji bitowych z maską równą 1,
- zlicza ilość bitów ustawionych na 0 za pomocą odpowiednich przesunięć i operacji bitowych z maską równą 1.

```
#include <iostream>

using namespace std;

void licz_binarne(unsigned int x, int &w0, int&w1)
{
    // jeśli x nie będzie 0 to będzie się wykonywała pętla
    unsigned int maska = 1;
```

```
    while(x)
    {
        // Sprawdzanie czy na ostatnim miejscu w x jest jedynka
        if ((x & maska) == 1)
        {
            w1++;
        }
        // Sprawdzanie czy na ostatnim miejscu w x jest zero
        else if ((x & maska) == 0)
        {
            w0++;
        }
        // Przesunięcie bitowe o jeden w lewo
        x = (x >> 1);
    }
}

int main()
{
    unsigned int wartosc = 73;
    // w binarnym to 1001001
    int zera = 0, jedynki = 0;
    licz_binarne(wartosc, zera, jedynki);

    cout << "zera: " << zera << endl;
    cout << "jedynki: " << jedynki << endl;

    return 0;
}
```

Jak działa bitowe AND (&):

Liczba 73 jest w postaci binarnej zapisana jako:

0100 1001

Liczba jeden zaś jest zapisana jako:

0000 0001

Wszelkie zera po prawej stronie przed pierwszą jedynką od lewej nie mają znaczenia, piszę je dla czytelności.

Bitowe AND sprawdza w którym miejscu zarówno w pierwszej jak i drugiej porównywanej wartości jest jedynka. Jako iż w zadaniu mamy podaną maskę 1 to wynik może nam wyjść jeden lub zero.

0100 1001

0000 0001

Wynik

0000 0001

Czyli po prostu 1

Przesunięcia bitowe:

Przesunięcia bitowe to nic innego jak dodanie zera lub zabranie wartości z prawej strony.
Jeśli przesuwamy coś o jeden w prawo wystarczy usunąć ostatnią cyfrę

73 przed przesunięciem

0100 1001

73 po przesunięciu o jeden w prawo (będzie to równe 36)

0010 0100