

# ZAAWANSOWANE PROGRAMOWANIE KOMPUTEROWE WNE (2022)

KRZYSZTOF ZIEMIAŃSKI

## PRZYKŁADOWE ZADANIA NA KOŁOKWIUM

### Funkcje.

- A1. Napisać funkcję, która zwraca wartość `true` wtedy i tylko wtedy, gdy tablica `t` o długości `n` jest uporządkowana (rosnąco lub malejąco).

```
bool f(int* t, int n);
```

- A2. Napisać funkcję, która zwraca medianę elementów z tablicy `t` o rozmiarze `n`.

```
int f(int* t, int n);
```

- A3. Napisać funkcję

```
int f(int a, int b, int n);
```

która zlicza liczbę sposobów, na jakie można zapisać liczbę `n` w postaci sumy liczb `a` i `b`. Utożsamiamy przedstawienia, które różnią się tylko kolejnością składników, np.  $f(2,3,8)=2$ , bo

$$8 = 2 + 2 + 2 + 2 = 2 + 3 + 3.$$

Oszacować złożoność napisanej funkcji.

- A4. Napisać funkcję

```
int f(int n, int a, int b);
```

która zlicza liczbę sposobów, na jakie można zapisać liczbę `n` w postaci sumy liczb `a` i `b`. Rozróżniamy przedstawienia, które różnią się kolejnością składników, np.  $f(2,3,8)=4$ , bo

$$8 = 2 + 2 + 2 + 2 = 2 + 3 + 3 = 3 + 2 + 3 = 3 + 3 + 2.$$

Oszacować złożoność napisanej funkcji.

- A5. Napisać funkcję

```
bool f(int n, int a, int b, int c);
```

która zwraca `true` wtedy i tylko wtedy, gdy `n` można przedstawić w postaci sumy, której wszystkie składniki są równe jednej z liczb `a`, `b`, `c`. Oszacować złożoność napisanej funkcji.

- A6. Napisać funkcję

```
void f(int n);
```

która wypisuje na ekran wszystkie możliwe przedstawienia liczby `n` w postaci nierosnącej sumy liczb całkowitych dodatnich. Oszacować złożoność tej funkcji.

A7. Napisać funkcję

```
bool f(char* a, char* b);
```

która zwraca `true` wtedy i tylko wtedy, gdy jeden napis jest przesunięciem drugiego. Np. przesunięciami napisu "ABCD" są napisy "BCDA", "CDAB", "DABC" i oczywiście "ABCD".

A8. Napisać funkcję

```
int f(vector<int>& t);
```

która zwraca długość najdłuższego ciągu kolejnych elementów wektora `t` tworzących ciąg arytmetyczny.

A9. Napisać funkcję

```
int g(vector<int>& t);
```

która zwraca długość najdłuższego ciągu kolejnych elementów wektora `t` o dodatniej sumie. Jeśli żaden z elementów tablicy nie jest dodatni, funkcja powinna zwrócić 0.

A10. Napisać funkcję

```
void g(double* t1, double* t2, double* t3, double* t, int n);
```

Parametry `t1`, `t2`, `t3` to posortowane rosnąco tablice o rozmiarze `n`, a tablica `t` ma rozmiar `3*n`. Funkcja powinna przepisać elementy z tablic `t1`, `t2`, `t3` do tablicy `t` tak, aby zachować porządek rosnący. Należy zadbać o to, aby funkcja działała jak najszybciej.

A11. Ciąg  $(a_1, \dots, a_n)$  jest szybko rosnący jeśli wszystkie jego elementy są dodatnie i każdy jest co najmniej dwukrotnie większy od poprzedniego. Napisać funkcję

```
int a(int m);
```

która zwraca liczbę różnych ciągów szybko rosnących kończących się na `m`. Np.  $a(5)$  powinno zwrócić 4 (są cztery ciągi: (5), (1, 5), (2, 5), (1, 2, 5)). Ocenić złożoność napisanej funkcji.

**Rekurencja.** Dla każdej z poniższych funkcji obliczyć złożoność ze względu na jej parametry. Napisać równoważną funkcję, która nie używa rekurencji.

```
B1. int f(int n) {
    if(n<3) return 1;
    return f(n-1)+f(n-2)+f(n-3);
}
```

```
B2. int f(int a, int b) {
    if(a<=0 || b<=0) return 1;
    return f(a-1, b)+f(a,b-1)+a*b;
}
```

```
B3. int f(int k) {
    if(k) return 1+k%10+f(k/10);
    return 0;
}
```

```

    }
B4.   int f(int n, int k) {
        if(n==1) return 2;
        if(n%k==0)
            return k*k*f(n/k, k);
        else
            return f(n, k+1);
    }
B5.   int g(int n) {
        if(n<3) return n;
        return f(n-1)+f(n-2)+f(n-2);
    }

```

### Klasy.

C1. Stworzyć klasę Tablica przechowującą elementy typu double.

```

class Tablica {
public:
    Tablica(int n, double x);
    // tworzy nową tablicę wypełnioną elementami x
    void ustaw(int i, double v); // ustawia wartość o wskazanym indeksie
    double wartosc(int i); // zwraca wartość o podanym indeksie
    Tablica& dolacz(Tablica& t);
    // dołącza do tablicy elementy z tablicy będącej parametrem
    int rozmiar(); // zwraca rozmiar tablicy
};

```

C2. Stworzyć klasę Czas, która przechowuje aktualny czas (godzinę i minutę)

```

class Czas {
public:
    Czas();
    Czas(int h, int m);
    Czas& dodajGodziny(int ileGodzin);
    Czas& dodajMinuty(int ileMinut);
    void drukuj();
    // drukuje godzinę w formacie 12-godzinnym (np. 7:30AM lub 1:14PM)
    int zaIleMinut(Czas t); // podaje, za ile minut nastąpi podany czas
};

```

C3. Napisać klasę ZbiorNapisow, która przechowuje zbiory napisów

```

class ZbiorNapisow {
public:
    ZbiorNapisow(); // tworzy pusty zbiór
    void dodaj(char* napis); // dodaje napis
    void usun(char* napis); // usuwa napis
    bool nalezy(char* napis); // sprawdza, czy podany napis należy do zbioru
    int ile(); // zwraca liczbę napisów
};

```

```

    double sredniaDlugosc(); // zwraca średnią długość napisu ze zbioru
};

```

- C4. Labirynt to prostokątny budynek podzielony na kwadratowe pomieszczenia jednakowej wielkości. Pomieszczenia są ponumerowane parami liczb całkowitych  $(a, b)$ ,  $0 \leq a < s$ ,  $0 \leq b < w$ , gdzie  $s$  i  $w$  są rozmiarami labiryntu. Pomiedzy sąsiednimi pomieszczeniami może znajdować się przejście, pomieszczenia skrajne nie mogą mieć wyjść na zewnątrz. Napisać klasę Labirynt, której obiekty reprezentują labirynty opisane powyżej.

```

class Labirynt {
public:
    Labirynt(int s, int w);
    // tworzy nowy Labirynt o rozmiarach s,w bez przejść pomiędzy polami
    void wstawPrzejscie(int x, int y, char k);
    // wstawia przejście z pomieszczenia (x,y) w kierunku k
    void usunPrzejscie(int x, int y, char k);
    // usuwa przejście z pomieszczenia (x,y) w kierunku k
    bool jestPrzejscie(int x, int y, char k);
    // zwraca informację czy jest przejście
};

```

- C5. Napisać klasę T której obiekty reprezentują uporządkowane ciągi liczb typu double o długości nie większej niż 100.

```

class T {
public:
    T(); // tworzy nowy pusty ciąg
    void dodaj(double x); // dodaje nową liczbę x
    bool nalezy(double x); // zwraca przynależność
    void usun(double x); // usuwa podaną liczbę (tylko jedno wystąpienie)
    void usunWszystkie(double x); // usuwa wszystkie wystąpienia
    double& operator[](int i);
    // zwraca i-ty najmniejszy element lub 0 jeśli indeks jest niepoprawny
};

```

- C6. Napisać klasę Drzewa, której obiekty reprezentują zbiory drzew w lesie. Każde drzewo jest reprezentowane przez dwie współrzędne i literę kodującą gatunek drzewa. Zakładamy, że drzew jest nie więcej niż 100.

```

class Drzewa {
public:
    Drzewa(); // tworzy pusty las
    void dodaj(double x, double y, char gatunek); // dodaje nowe drzewo
    void usun(char gatunek); // usuwa wszystkie drzewa o podanym gatunku
    int ile(char gatunek); // zwraca liczbę drzew o podanym gatunku
    char najblizszy(double x, double y);
    // zwraca gatunek drzewa rosnącego najbliżej zadanego punktu
    // ('?' jeśli nie ma żadnego drzewa)
};

```

**Listy.** Dana jest struktura

```
class ElListy{  
public:  
    int wartosc;  
    ElListy* nast;  
};
```

której będziemy używać do tworzenia list jednokierunkowych.

D1. Napisać funkcję

```
ElListy* sklej(ElListy* a, ElListy* b);
```

która zwraca listę, w której występują najpierw elementy listy **a** (kolejność elementów powinna być zachowana), a następnie elementy listy **b**. Należy użyć istniejących już elementów z list **a** i **b**.

D2. Napisać funkcję

```
bool petla(ElListy* a);
```

która zwraca **true** wtedy i tylko wtedy, gdy lista wskazywana przez **a** zawiera pętlę, tzn. jeśli przechodząc do kolejnych elementów listy nigdy nie dojdziemy do końca.

D3. Napisać funkcję

```
ElListy* usun(ElListy* a, int w);
```

która usuwa z listy **a** wszystkie wartości równe **w** i zwraca tą listę.