

Programowanie i projektowanie obiektowe

Podstawy programowania

Paweł Daniluk

Wydział Fizyki

Jesień 2011



[http://bioexploratorium.pl/wiki/
Programowanie_Objectowe_-_2011z](http://bioexploratorium.pl/wiki/Programowanie_Objectowe_-_2011z)

Podstawowe konstrukcje

Komentarze

```
/* This is a multi-line comment.  
It may occupy more than one line. */
```

```
// This is an end-of-line comment
```

Zmienne i przypisanie

```
int myInt;          /* Declaring an uninitialized variable  
                    called 'myInt', of type 'int' */  
myInt = 35;        // Initializing the variable  
int myInt = 35;    /* Declaring and initializing the variable  
                    at the same time */
```

Uwaga

= oznacza przypisanie
== jest operatorem porównania

Instrukcje warunkowe

Instrukcja warunkowa

```
if (i == 3) doSomething();
```

```
if (i == 2)
    doSomething();
else
    doSomethingElse();
```

```
if (i == 3) {
    doSomething();
} else if (i == 2) {
    doSomethingElse();
} else {
    doSomethingDifferent();
}
```

Instrukcje warunkowe

Instrukcja wyboru

```
switch (ch) {  
    case 'A':  
        doSomething(); // Triggered if ch == 'A'  
        break;  
  
    case 'B':  
    case 'C':  
        doSomethingElse(); // Triggered if ch == 'B'  
        break;           // or ch == 'C'  
  
    default:  
        doSomethingDifferent(); // Triggered in any other case  
        break;  
}
```

Pętle

while

```
while (i < 10) {  
    doSomething();  
}
```

```
// doSomething() is called at least once  
do {  
    doSomething();  
} while (i<10);
```

Pętle c.d.

for

```
for (int i = 0; i < 10; i++) {  
    doSomething();  
}
```

// A more complex loop using two variables

```
for (int i = 0, j = 9; i < 10; i++, j -= 3) {  
    doSomething();  
}
```

```
for (;;) {  
    doSomething();  
}
```

Instrukcje skoku

Etykiety

```
start:  
someMethod();
```

break

```
for (int i = 0; i < 10; i++) {  
    while (true) {  
        break;  
    }  
    // Will break to this point  
}
```



```
outer:  
for (int i = 0; i < 10; i++) {  
    while (true) {  
        break outer;  
    }  
}
```

Instrukcje skoku

continue

```
int ch;
while (ch = getChar()) {
    if (ch == ' ')
        continue; // Skips the rest of the while-loop

    // Rest of the while-loop, will not be reached if ch == ' '
    doSomething();
}
```

```
outer:
for (String str : stringsArr) {
    char[] strChars = str.toCharArray();
    for (char ch : strChars) {
        if (ch == ' ') {
            /* Continues the outer cycle and the next
            string is retrieved from stringsArr */
            continue outer;
        }
        doSomething(ch);
    }
}
```

Tablice

Tablice

```
int[] numbers = new int[5];  
numbers[0] = 2;  
numbers[1] = 5;  
int x = numbers[0];
```

Inicjalizacja

```
// Long syntax  
int[] numbers = new int[5] {20, 1, 42, 15, 34};  
// Short syntax  
int[] numbers2 = {20, 1, 42, 15, 34};
```

Tablice c.d.

Tablice wielowymiarowe

```
int[] [] numbers = new int[3][3];  
number[1][2] = 2;
```

```
int[] [] numbers2 = {{2, 3, 2}, {1, 2, 6}, {2, 4, 5}};
```

Wiersze mogą być różnej długości

```
//Initialization of the first dimension only  
int[] [] numbers = new int[2][];  
  
numbers[0] = new int[3];  
numbers[1] = new int[2];
```

Procedury, funkcje → metody

Metody

```
class Foo {  
    static int bar(int a, int b) {  
        return (a * 2) + b;  
    }  
  
    static int baz(int a) {  
        return bar(a, 0);  
        //return a * 2;  
    }  
}
```

Parametry są przekazywane przez wartość.

Procedury, funkcje → metody c.d.

Metoda main

```
class Foo {  
    public static void main(String[] args) {  
        doSomething();  
    }  
}
```

Projekty w NetBeans

Chwilowo będziemy traktować projekt jako zbiór programów.

- Klasa \longleftrightarrow program
- Projekt może się składać z wielu klas.
- Każda klasa może mieć metodę main.
- Klasę uruchamiamy opcją "Run file".

Zadanie 1 – Flaga polska

Zadanie

Tablica A typu $\text{int}[N]$ wypełniona zerami i jedynkami reprezentuje ciąg N urn w których znajdują się żetony białe (0) i czerwone (1). Podać algorytm, który zamieniając żetony miejscami doprowadzi do sytuacji, w której wszystkie żetony białe znajdują się na lewo od czerwonych.

Zadanie 1 – Flaga polska

Zadanie

Tablica A typu $\text{int}[N]$ wypełniona zerami i jedynkami reprezentuje ciąg N urn w których znajdują się żetony białe (0) i czerwone (1). Podać algorytm, który zamieniając żetony miejscami doprowadzi do sytuacji, w której wszystkie żetony białe znajdują się na lewo od czerwonych.

Wskazówka

Należy przesuwać się indeksem c od początku tablicy, zaś indeksem b od końca. Intencją jest utrzymywanie następującego niezmiennika: wszystkie elementy tablicy o indeksach mniejszych od c są czerwone, zaś większych od b są białe. Indeksy c i b będą się do siebie zbliżać i ostatecznie gdy c będzie równe b , to tablica będzie uporządkowana.

Szkielet rozwiązania

```
import java.util.Random;

public class FlagaPolska {

    public static void main(String[] args) {
        int N = 10;
        int[] T = new int[N];

        Random rGen = new Random();

        for (int i = 0; i < N; i++) {
            T[i] = rGen.nextInt(2);
        }

        System.out.println("Przed:");
        for (int i = 0; i < N; i++) {
            System.out.print(T[i]);
        }
        System.out.println();

        // Tu wstawić rozwiązanie.

        System.out.println("Po:");
        for (int i = 0; i < N; i++) {
            System.out.print(T[i]);
        }
        System.out.println();
    }
}
```

Zadanie 2 – Flaga trójkolorowa

Zadanie

Dana jest tablica A typu $\text{int}[N]$. Należy tak przestawić w niej elementy, żeby najpierw były elementy ujemne, potem równe zero, a na końcu dodatnie.

Zadanie 3 – Najdłuższe plateau

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, długość najdłuższego stałego segmentu (spójnego fragmentu tablicy).

Zadanie 3 – Najdłuższe plateau

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, długość najdłuższego stałego segmentu (spójnego fragmentu tablicy).

Wskazówki

- Zadanie to można rozwiązać używając dwóch pętli; zewnętrznej (po możliwych początkach segmentu) i wewnętrznej (w której szukamy końca segmentu stałego).

Zadanie 3 – Najdłuższe plateau

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, długość najdłuższego stałego segmentu (spójnego fragmentu tablicy).

Wskazówki

- Zadanie to można rozwiązać używając dwóch pętli; zewnętrznej (po możliwych początkach segmentu) i wewnętrznej (w której szukamy końca segmentu stałego).
- Dokładnie to samo rozwiązanie można zapisać używając jednej pętli.

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Wskazówki

- Najprostsze rozwiązanie to dla wszystkich możliwych segmentów policzyć ich sumę.

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Wskazówki

- Najprostsze rozwiązanie to dla wszystkich możliwych segmentów policzyć ich sumę.
- W powyższym rozwiązaniu sumę pewnych segmentów liczy się wiele razy. Lepiej dla danego początku l obliczać po kolei sumy coraz dłuższych segmentów zaczynających się w l .

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Wskazówki

- Najprostsze rozwiązanie to dla wszystkich możliwych segmentów policzyć ich sumę.
- W powyższym rozwiązaniu sumę pewnych segmentów liczy się wiele razy. Lepiej dla danego początku l obliczać po kolei sumy coraz dłuższych segmentów zaczynających się w l .
- Niech $\text{Pref}(i)$ oznacza sumę elementów tablicy od 1 do i włącznie. Suma segmentu od l do p to oczywiście $\text{Pref}(p) - \text{Pref}(l-1)$. Maksymalną sumę segmentu kończącego się w p uzyskamy odejmując od $\text{Pref}(p)$ minimalne $\text{Pref}(i)$ gdzie i przebiega $[1..p]$.

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Wskazówki

- Najprostsze rozwiązanie to dla wszystkich możliwych segmentów policzyć ich sumę.

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Wskazówki

- Najprostsze rozwiązanie to dla wszystkich możliwych segmentów policzyć ich sumę.
- W powyższym rozwiązaniu sumę pewnych segmentów liczy się wiele razy. Lepiej dla danego początku l obliczać po kolei sumy coraz dłuższych segmentów zaczynających się w l .

Zadanie 4 – Segment o maksymalnej sumie

Zadanie

Napisz program znajdujący w zadanej tablicy A typu $\text{int}[N]$, maksymalną sumę segmentu (spójnego fragmentu tablicy). Przyjmujemy, że segment pusty ma sumę 0.

Wskazówki

- Najprostsze rozwiązanie to dla wszystkich możliwych segmentów policzyć ich sumę.
- W powyższym rozwiązaniu sumę pewnych segmentów liczy się wiele razy. Lepiej dla danego początku l obliczać po kolei sumy coraz dłuższych segmentów zaczynających się w l .
- Niech $\text{Pref}(i)$ oznacza sumę elementów tablicy od 1 do i włącznie. Suma segmentu od l do p to oczywiście $\text{Pref}(p) - \text{Pref}(l-1)$. Maksymalną sumę segmentu kończącego się w p uzyskamy odejmując od $\text{Pref}(p)$ minimalne $\text{Pref}(i)$ gdzie i przebiega $[1..p]$.

Zadanie 5 – Podciąg

Zadanie

Dane są dwie tablice A typu $\text{int}[N]$ i B typu $\text{int}[M]$. Napisz program sprawdzający, czy A jest podciągiem B (tzn. czy istnieje funkcja f , rosnąca, z $1..N$ w $1..M$, t. że $A[i]=B[f(i)]$).

Zadanie 5 – Podciąg

Zadanie

Dane są dwie tablice A typu $\text{int}[N]$ i B typu $\text{int}[M]$. Napisz program sprawdzający, czy A jest podciągiem B (tzn. czy istnieje funkcja f , rosnąca, z $1..N$ w $1..M$, t. ze $A[i]=B[f(i)]$).

Wskazówka

Każdy element tablicy A musi odnaleźć swoją kopię w tablicy B. Przechodząc tablicę A od lewej do prawej i szukamy odpowiednika $A[i]$ w części B, która jeszcze nie została odwiedzona.

Sortowanie

Sortowanie to jeden z najbardziej podstawowych problemów algorytmicznych.

Klasyfikacja algorytmów sortowania

- Złożoność obliczeniowa $O(n^2)$, $O(n \log n)$.
- Złożoność pamięciowa (sortowanie w miejscu).
- Stabilność

Przykład stabilności

Sortujemy: (4, 1) (3, 7) (3, 1) (5, 6)

Kolejność zachowana: (3, 7) (3, 1) (4, 1) (5, 6) Kolejność zmieniona: (3, 1) (3, 7) (4, 1) (5, 6)

Zadanie 6 – Sortowanie bąbelkowe (*bubble sort*)

Algorytm

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Przykład

$$\begin{aligned} & \underbrace{[4, 2, 5, 1, 7]}_{4 > 2} \rightarrow \underbrace{[2, 4, 5, 1, 7]}_{4 < 5} \rightarrow \underbrace{[2, 4, 5, 1, 7]}_{5 > 1} \rightarrow \underbrace{[2, 4, 1, 5, 7]}_{5 < 7} \\ & \underbrace{[2, 4, 1, 5, 7]}_{2 < 4} \rightarrow \underbrace{[2, 4, 1, 5, 7]}_{4 > 1} \rightarrow \underbrace{[2, 1, 4, 5, 7]}_{4 < 5} \\ & \underbrace{[2, 1, 4, 5, 7]}_{2 > 1} \rightarrow \underbrace{[1, 2, 4, 5, 7]}_{2 < 4} \\ & \underbrace{[1, 2, 4, 5, 7]}_{1 < 2} \end{aligned}$$

Zadanie 7 – Sortowanie przez wstawianie (*insertion sort*)

Algorytm

- 1 Utwórz zbiór elementów posortowanych i przenieś do niego dowolny element ze zbioru nieposortowanego.
- 2 Weź dowolny element ze zbioru nieposortowanego.
- 3 Wyciągnięty element porównuj z kolejnymi elementami zbioru posortowanego póki nie napotkasz elementu równego lub elementu większego (jeśli chcemy otrzymać ciąg niemalejący) lub nie znajdziemy się na początku/końcu zbioru uporządkowanego.
- 4 Wyciągnięty element wstaw w miejsce gdzie skończyłeś porównywać.
- 5 Jeśli zbiór elementów nieuporządkowanych jest niepusty wróć do punkt 2.

Zadanie 8* – Sortowanie szybkie (*quick sort*)

Algorytm

Algorytm działa rekursywnie - wybierany jest pewien element tablicy, tzw. element osiowy, po czym na początek tablicy przenoszone są wszystkie elementy mniejsze od niego, na koniec wszystkie większe, a w powstałe między tymi obszarami puste miejsce trafia wybrany element. Potem sortuje się osobno początkową i końcową część tablicy. Rekursja kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa podtablica nie wymaga sortowania.

Niech l oznacza indeks pierwszego, r – ostatniego elementu sortowanego fragmentu tablicy, zaś i – indeks elementu, na którym tablica została podzielona.

Zadanie 8* – Sortowanie szybkie (*quick sort*) c.d.

Pseudokod

```
PROCEDURE Quicksort(l, r)
  BEGIN
    IF l < r THEN { jeśli fragment dłuższy niż 1 element }
      BEGIN
        i = PodzielTablice(l, r); { podziel i zapamiętaj punkt
        Quicksort(l, i-1);        { posortuj lewą część }
        Quicksort(i, r);          { posortuj prawą część }
      END
    END
  END
```