Programowanie i projektowanie obiektowe

Wstęp

Paweł Daniluk

Wydział Fizyki

Jesień 2011

Plan wykładu

1. Wstęp – komputery, algorytmy i programy
2. Podstawowe konstrukcje programistyczne
3. Procedury i funckje
4. Obiekty i metody
5. Struktury danych
6. Zarządzanie pamięcią
7. Rekurencja
8. Programowanie dynamiczne
9. Modelowanie dziedziny
10. Dziedziczenie
11. Klasy w Javie, kapsułkowanie, konstruktory
12. Wyjątki
13. Typy uogólnione
14. Kolekcje
15. Graficzny interfejs użytkownika

Algorytm

Sposób postępowania prowadzący do pożądanego efektu.
Przykłady:

1. Przepisy kucharskie
2. Zapis nutowy muzyki
3. Instrukcje montażu
4. Opis dojazdu

Instrukcja szamponu

1. Nałożyć szampon
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć

Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć

Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć – skok do 1

Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć – skok do 1
    1. Nałożyć szampon
    2. Umyć włosy
    3. Spłukać
    4. Czynność powtórzyć

Instrukcja szamponu

1. Nałożyć szampon – ile?, na co?
2. Umyć włosy
3. Spłukać
4. Czynność powtórzyć – skok do 1
    1. Nałożyć szampon
    2. Umyć włosy
    3. Spłukać
    4. Czynność powtórzyć
        1. Nałożyć szampon
        2. Umyć włosy
        3. Spłukać
        4. Czynność powtórzyć

Największy wspólny dzielnik

Dzielenie z resztą

$$b = ax + r$$

$$b \div a = x$$

$$b \mod a = r$$

Podzielność

Mówimy a dzieli b $(a|b)$ jeżeli istnieje liczba całkowita x, taka że $b = ax$.

$$a|b \iff b \mod a = 0$$

Największy wspólny dzielnik

Największym wspólnym dzielnikiem a i b jest największa liczba która dzieli a i b.

$$NWD(15, 18) = 3 \qquad NWD(7, 13) = 1$$

Algorytm Euklidesa

Pomysł

$$NWD(a,b) = \begin{cases} a & \text{jeśli } b = 0 \\ NWD(b,a) & \text{jeśli } b > a \\ NWD(a-b,b) & \text{jeśli } a \geq b > 0 \end{cases}$$

Naiwna implementacja

Pseudokod

NWD(a, b)

▷ zakładamy, że $a > b$ i $a + b > 0$

1 **while** $b > 0$
2 **do** $a \leftarrow a - b$
3 **if** $b > a$
4 **then** swap(a, b)
5 **return** a

Niska wydajność

Dla $a = 10^10$ i $b = 1$ trzeba wykonać 10^10 odejmowań.

Poprawna implementacja

Pseudokod

NWD(a, b)

▷ zakładamy, że $a > b$ i $a + b > 0$

1 **while** $b > 0$

2 **do** $a \leftarrow a \mod b$

3 swap(a, b)

4 **return** a

Poprawność algorytmu

Poprawność częściowa

Jeżeli obliczenie się kończy, to dostajemy poprawny wynik.

Przykład

isEven(a)

```
1  while a ≠ 0
2        do a ← a − 2
3  return true
```

Poprawność algorytmu

Poprawność częściowa

Jeżeli obliczenie się kończy, to dostajemy poprawny wynik.

Przykład

isEven(a)

1 **while** $a \neq 0$
2 **do** $a \leftarrow a - 2$
3 **return** *true*

Poprawność pełna

Dla dowolnych danych wejściowych obliczenie się kończy i dostajemy poprawny wynik.

Nie wszystko da się zaprogramować

Problem STOPu

Rozstrzygnąć czy program P zatrzymuje się dla dowolnych danych wejściowych.

Nie wszystko da się zaprogramować

Problem STOPu

Rozstrzygnąć czy program P zatrzymuje się dla dowolnych danych wejściowych.

Nie istnieje algorytm rozwiązujący problem stopu.

Nie wszystko da się zaprogramować c.d.

Dowód

Załóżmy, że istnieje w pełni poprawny program S, który dla dowolnego programu P i danych D rozstrzyga, czy P uruchomiony na danych D zatrzymuje się.
Weźmy program T:

T(P)

```
1  if S(P, P) = true
2      then loop
3      else stop
```

Jaki jest rezultat uruchomienia $T(T)$?

Jak działa komputer

- Procesor – wczytuje rozkazy z pamięci i je wykonuje

Jak działa komputer

- Procesor – wczytuje rozkazy z pamięci i je wykonuje
- Pamięć – przechowuje programy i dane

Jak działa komputer

- Procesor – wczytuje rozkazy z pamięci i je wykonuje
- Pamięć – przechowuje programy i dane
- UWAGA: Programy i dane są traktowane tak samo

Jak działa komputer

- Procesor – wczytuje rozkazy z pamięci i je wykonuje
- Pamięć – przechowuje programy i dane
- UWAGA: Programy i dane są traktowane tak samo
- Szyna danych – Łączy procesor, pamięć i inne elementy komputera

Jak działa komputer

- Procesor – wczytuje rozkazy z pamięci i je wykonuje
- Pamięć – przechowuje programy i dane
- UWAGA: Programy i dane są traktowane tak samo
- Szyna danych – Łączy procesor, pamięć i inne elementy komputera
- Urządzenia wejścia/wyjścia – procesor komunikuje się z nimi za pośrednictwem szyny danych i dodatkowych interfejsów

Kod maszynowy

Postać binarna

```
10110000 01100001
```

Kod maszynowy

Postać binarna

```
10110000 01100001
```

Postać szestnastkowa

```
B0 61
```

Kod maszynowy

Postać binarna

```
10110000 01100001
```

Postać szestnastkowa

```
B0 61
```

Postać mnemoniczna

```
MOV AL, 61h ; Load AL with 97 decimal (61 hex)
```

Kod maszynowy c.d.

Fragment programu

```
mov ax, 0D625h
mov es, ax    ; wprowadź do rejestru segmentowego ES wartość
              ; z AX wynoszącą D625 szesnastkowo
              ; (54821 dziesiętnie)
mov al, 24
mov ah, 0     ; załaduj do rejestru AX wartość 24
              ; (wyzeruj AH - starszą połówkę rejestru AX
              ; i zapisz wartość 24 w młodszej AL)
int 21h       ; wywołaj przerwanie nr 33 (21 szesnastkowo)
```

Paradygmaty programowania

1. Programowanie imperatywne – Program jest sekwencją instrukcji powodujących zmiany stanu (np. wartości zmiennych).

Paradygmaty programowania

1. Programowanie imperatywne – Program jest sekwencją instrukcji powodujących zmiany stanu (np. wartości zmiennych).
2. Programowanie funkcyjne – Brak skutków ubocznych, stan się nie zmienia.

Paradygmaty programowania

1. Programowanie imperatywne – Program jest sekwencją instrukcji powodujących zmiany stanu (np. wartości zmiennych).
2. Programowanie funkcyjne – Brak skutków ubocznych, stan się nie zmienia.
3. Programowanie deklaratywne – Programista opisuje pożądany wynik bez precyzowania algorytmu.

Paradygmaty programowania

1. Programowanie imperatywne – Program jest sekwencją instrukcji powodujących zmiany stanu (np. wartości zmiennych).
2. Programowanie funkcyjne – Brak skutków ubocznych, stan się nie zmienia.
3. Programowanie deklaratywne – Programista opisuje pożądany wynik bez precyzowania algorytmu.
4. Programowanie obiektowe – Obiekty łączą w sobie dane (stan) i zachowanie (metody).

Język programowania

1. Składnia – słowa i gramatyka
2. Semantyka – znaczenie
3. Typy danych
4. Biblioteki standardowe

Kompilować, czy interpretować

Języki kompilowane

Kod źródłowy jest kompilowany do maszynowego.

- szybkie
- podatne na błędy
- trudno poprawiać
- zależne od platformy

Kompilować, czy interpretować

Języki kompilowane

Kod źródłowy jest kompilowany do maszynowego.

- szybkie
- podatne na błędy
- trudno poprawiać
- zależne od platformy

Języki interpretowane

Interpreter analizuje kod źródłowy w miarę wykonywania.

- mało wydajne
- wygodne
- niezależne od platformy

Kompilować, czy interpretować c.d.

Java – rozwiązanie pośrednie

Kod źródłowy kompilowany do kodu bajtowego (*bytecode*) i wykonywany przez maszynę wirtualną.

- wydajne (niby)
- niezależne od platformy

Java - założenia

1. It should be “simple, object-oriented and familiar”.
2. It should be “robust and secure”.
3. It should be “architecture-neutral and portable”.
4. It should execute with “high performance”.
5. It should be “interpreted, threaded, and dynamic”.

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 przypisania

```
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

Podstawowe konstrukcje c.d.

Instrukcja warunkowa

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

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

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

Podstawowe konstrukcje c.d.

Pętla

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

```
// doSomething() is called at least once
do {
    doSomething();
}
```

Podstawowe konstrukcje c.d.

Wypisywanie wyniku

```
System.out.println(s); // Display the string.
```

Najprostszy program

```
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Integrated Development Environment (IDE)

Zmiana hasła

```
$ passwd
```

NetBeans

```
$ netbeans
```

Algorytm Euklidesa w Javie

Szkielet programu do rozbudowania

```
import java.util.Scanner;

public class Euclid {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Podaj pierwsza liczbe:");
        int a = sc.nextInt();

        System.out.println("Podaj druga liczbe:");
        int b = sc.nextInt();

        System.out.println("Wynik:"+Integer.toString(a) +"+"+Ir
    }
}
```

Algorytm Euklidesa w Javie c.d.

Operatory porównania

==	$=$
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$
!=	$\neq$

Operatory arytmetyczne

+	$+$
-	$-$
*	$\cdot$
/	$\div$
%	mod

+ oznacza również konkatenację łańcuchów znakowych.