Fundusze Europejskie
Wiedza Edukacja Rozwój
Politechnika Wrocławska
Unia Europejska
Europejski Fundusz Społeczny

„ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej”

Paradygmaty programowania - ćwiczenia
Lista 2

W Dodatku do wykładu 2 przypominam, że obliczenia w arytmetyce całkowitej są „zawijane”,
np. w Scali: fibTail(82) => -1230842041
Jeśli jednak użyjemy typu BigInt, to otrzymamy poprawny wynik:
fibTailBigInt(82) => 61305790721611591

Przypominam, że do przechowywania liczb rzeczywistych w komputerze jest stosowana [reprezentacja zmiennoprzecinkowa](), nazywana też zmiennopozycyjną (ang. floating point), która wykorzystuje ustaloną dla danej reprezentacji (skończoną) liczbę bitów. Wobec tego liczba rzeczywista najczęściej nie będzie mogła być zapamiętana dokładnie w pamięci komputera, gdyż możemy przechować jedynie **ograniczoną** liczbę cyfr cechy i mantysy.
Operacje na liczbach zmiennoprzecinkowych są wykonywane w arytmetyce zmiennoprzecinkowej (zmiennopozycyjnej) i ze względu na stosowaną reprezentację mogą być obarczone błędami zaokrągleń.
Z powyższego powodu dla liczb zmiennoprzecinkowych należy wystrzegać się testów równościowych w rodzaju:

```
if (x == 1.0) …
```

Na przykład

```
scala> 10.0 * (1.1 - 1.0) == 1.0
val res0: Boolean = false

# 10.0 *. (1.1 -. 1.0) = 1.0;;
- : bool = false
```

Gdy liczbę rzeczywistą x przybliżamy inną liczbą x^*, to *błąd bezwzględny* tego przybliżenia jest z definicji równy

$$|x - x^*|$$

a *błąd względny* jest (dla $x \neq 0$) równy

$$\left|\frac{x - x^*}{x}\right|$$

W pomiarach niemal zawsze istotny jest ten drugi błąd. Informacja o błędzie bezwzględnym jest rzadko użyteczna, gdy nic nie wiemy o rzędzie wielkości liczby.
D. Kincaid, W.Cheney, Analiza numeryczna, WNT 2006, str. 46-47

W przykładzie powyżej znamy rząd wielkości liczby, więc można zastosować dokładność bezwzględną, np. :

```
scala> math.abs(1.0 - 10.0*(1.1-1.0)) <= 1.0e-15
val res1: Boolean = true

# abs_float(1. -. 10.0 *. (1.1 -. 1.0)) <= 1e-15;;
- : bool = true
```

W zadaniu 3 jest wykorzystana dokładność względna.

Paradygmaty programowania - ćwiczenia
Lista 2

1. Jaka będzie głębokość stosu (i dlaczego) w Scali, a jaka w OCamlu dla wywołania evenR(3) (funkcja zdefiniowana na wykładzie)?

Poniższe funkcje należy napisać w obu językach: OCaml i Scala. W zadaniach 5 i 6 należy koniecznie wykorzystać mechanizm dopasowania do wzorca.

2. Liczby Fibonacciego są zdefiniowane następująco:

$$f(0) = 0$$
$$f(1) = 1$$
$$f(n+2) = f(n) + f(n+1)$$

Napisz dwie funkcje, które dla danego n znajdują n-tą liczbę Fibonacciego:

a) fib: int -> int (Scala: fib: Int => Int) opartą bezpośrednio na powyższej definicji,

b) fibTail: int -> int (Scala: fibTail: Int => Int) wykorzystującą rekursję ogonową.

Porównaj (bez mierzenia) ich szybkość wykonania, obliczając np. 42-gą liczbę Fibonacciego.

3. Dla zadanej liczby rzeczywistej a oraz dokładności ε można znaleźć pierwiastek trzeciego stopnia z a wyliczając kolejne przybliżenia x_i tego pierwiastka (metoda Newtona-Raphsona):

$x_0 = a/3$ dla $a > 1$

$x_0 = a$ dla $a \leq 1$

$x_{i+1} = x_i + (a/x_i^2 - x_i)/3$

Dokładność (względna) jest osiągnięta, jeśli $|x_i^3 - a| \leq \varepsilon * |a|$.

Napisz efektywną (wykorzystującą rekursję ogonową) funkcję root3: float -> float, która dla zadanej liczby a znajduje pierwiastek trzeciego stopnia z dokładnością 10^{-15}.

Uwaga. Pamiętaj, że OCaml nie wykonuje automatycznie żadnych koercji typów.

Scala: root3(a: Double): Double (metoda) i root3: Double => Double (funkcja)

W Scali napisz metodę i funkcję.

4. Zwiąż zmienną x z wartością 0 konstruując wzorce, do których mają się dopasować następujące wyrażenia (jest wiele takich wzorców):

a) [-2; -1; 0; 1; 2]

b) [(1,2); (0,1)]

Np. dla wyrażenia (true,"hello",0) wymaganym wzorcem jest (_ ,_ ,x).

W OCamlu zignoruj ostrzeżenie "this pattern-matching is not exhaustive".

5. Zdefiniuj funkcję initSegment : 'a list * 'a list -> bool sprawdzającą w czasie liniowym, czy pierwsza lista stanowi początkowy segment drugiej listy. Każda lista jest swoim początkowym segmentem, lista pusta jest początkowym segmentem każdej listy.

Scala: initSegment[A](xs: List[A], ys: List[A]): Boolean

6. a) Zdefiniuj funkcję replaceNth : 'a list * int* 'a -> 'a list, zastępującą n-ty element listy podaną wartością (pierwszy element ma numer 0), np.

replaceNth(['o';'l';'a'; 'm'; 'a'; 'k'; 'o'; 't'; 'a'], 1, 's') => (['o';'s';'a'; 'm'; 'a'; 'k'; 'o'; 't'; 'a']

Scala: replaceNth[A](xs: List[A], n: Int, x: A): List[A]

Nie wykorzystuj żadnej funkcji bibliotecznej!

b) Jaka jest złożoność obliczeniowa tej funkcji? Koniecznie zilustruj rysunkiem reprezentację wewnętrzną obu list (patrz wykład str. 41 - 44).