Laboratorium XII

Erlang – część pierwsza

4 stycznia 2022 r.

Erlang

- Erlang = **Er**icsson **Lang**uage (choć tak naprawdę nazwa została nadana na cześć A.K. Erlanga)
- Zaprojektowany z myślą o zastosowaniach współbieżnych.
- Zaprojektowany także pod kątem tworzenia rozproszonych systemów.
- Umożliwia aktualizacje kodu bez zatrzymywania aplikacji.
- W odróżnieniu od Haskella jest językiem dynamicznie typowanym.

(źródło: Wikipedia:Erlang (język programowania)).

Erlang – pierwszy program

Spójrzmy na pierwszy przykład programu w Erlangu

```
-module (prog1).
-export ([binom/2]).
% Komentarz: funkcja obliczajaca silnie
silnia(0) -> 1;
silnia(N) -> N * silnia (N-1).

binom(N,K) -> silnia(N)/(silnia(K) * silnia(N-K)).
```

Pierwszy program – komentarz

- Nazwa **modułu** musi być taka sama jak nazwa pliku. W naszym przypadku zatem całość zapisujemy w pliku `prog1.erl`.
- W drugiej linii **eksportujemy funkcje** – tylko wyeksportowane funkcje dostępne są na zewnątrz. W naszym przypadku wyeksportowana zostanie jedna funkcja `binom` o dwóch argumentach.
- W liniach 4-5 definiujemy funkcję za pomocą ***dopasowania do wzorca*** (działa on analogicznie jak w Haskellu). Średniki oddzielają kolejne wzorce, natomiast kropka kończy definicję.
- Zauważmy, że w odróżnieniu od Haskella, **argumenty funkcji** są oznaczane dużymi literami.

Środowisko

- Aby wykonać powyższy program należy go zapisać (jak już o tym mówiliśmy) w pliku `prog1.erl`.
- Następnie wykonujemy polecenie `erl` w tym samym katalogu, co plik. Jeśli nie zrobimy tego, będziemy musieli przejść do tego katalogu za pomocą funkcji `cd/1`.
- Poniższej przedstawiamy schemat interakcji z Erlangiem:

```
erl
1>c(prog1).
{ok,prog1}
2>prog1:binom(20,4).
4845.0
3>prog1:binom(20,10).
184756.0
```

Zadanie

Proszę teraz skompilować ten kod i za jego pomocą obliczyć $\binom{20}{15}$.
Co się stanie, jeśli będziemy chcieli obliczyć `silnia(10)`?
Dlaczego?

Środowisko – podstawowe komendy

- **c(moduł).** – kompilacja modułu `moduł`.
- **nazwa modułu:funkcja(...).** – wywołanie funkcji.
- **pwd().** – wyświetlenie aktualnego katalogu.
- **cd(...).** – zmiana katalogu.
- **q().** – opuszczenie.

Uwaga na kropki na końcach!

Zadanie 1.

Interpunkcja

W tej części zbierzemy zasady znaków interpunkcyjnych.

- **Przecinek** – służy do oddzielania linii kodu w ramach jednego bloku.
- **Średnik** – służy do oddzielania kolejnych bloków.
- **Kropka** – służy do kończenia definicji.

Atomy

Atomy to po prostu literały. Nazwy atomów piszemy (prawie [a]) zawsze z małej litery

```
-module (zeros).
-export ([isZero/1]).

isZero(0) -> yes;
isZero(_) -> no.
```

Funkcja `isZero/1` zwraca atom `yes`, jeśli jej argument jest zerem. W przeciwnym wypadku zwraca atom `no`.

[a] Dla zainteresowanych: można też tego nie robić, ale wtedy nazwa atomu musi być umieszczona w apostrofach.

Krotki

Krotki zapisujemy w nawiasach wąsatych – na przykład

- `{10,20}` – para liczb;
- `{"ala",ma,3,psy}` – czwórka złożona z **łańcucha znaków,atomu, liczby całkowitej i atomu**.

Zmienne

To za dużo powiedziane. "Zmienna" raz zainicjowana nie może być już później zmieniona. Nazwy zmiennych muszą rozpoczynać się od dużej litery.

Listy

- W odróżnieniu od Haskella, listy mogą zawierać elementy różnych typów.
- Listy zapisujemy za pomocą nawiasów kwadratowych. Na przykład `List = ["ala","ola","pies",3,koty]`.
- Dopasowanie do wzorca dla listy `List`:
  - `[A|Reszta]` $\rightarrow$ $A =$ "*ala*", Reszta=["*ola*", "*pies*", 3, *koty*].
  - `[A,B|Reszta]` $\rightarrow$ $A =$ "*ala*", $B =$ "*ola*", Reszta=[3, *koty*].
  - itd.
- Tak jak w Haskellu, do konkatenacji list używamy operatora ++.

Zwróćmy uwagę na **duże litery** we wzorcach.

Listy

Podamy teraz przykład obliczający iloczyn wszystkich elementów na w liście.

```
-module (list).
-export ([product/1]).

product([]) -> 1;
product([A|Rest]) -> A * product(Rest).
```

Zadanie 2.

Strażnicy

Do konstrukcji **strażników** używamy słowa kluczowego **when.** Zobaczmy przykład:

```
-module (silnia2).
-export ([fact/1]).

fact(0)-> 1;
fact(N) when N>0 -> N * fact(N-1).
```

Więcej o tym, czego możemy użyć w strażnikach: Valid guard expressions.

Uwaga: W dozorach (w szczególności) nie można stosować własnych funkcji (warunków).

If/Then/Else

Ogólna struktura `if/then/else` podana jest poniżej. Zwracam uwagę na **średniki** oraz jego **brak na końcu**.

```
if warunek1->
  kod1;
   warunek2->
  kod2;
  ...
   warunekN->
  kodN
end
```

If/Then/Else

Else osiągamy dodając na końcu `true->...`.

```
if X /=0 ->
            kod1;
true -> kod2;
```

Jeśli $X \neq 0$, to wykona się `kod1`; *w przeciwnym przypadku* wykona się `kod2`.

Zadanie 3,4,5.

Funkcje anonimowe

Funkcje anonimowe to funkcję definiowane za pomocą słowa kluczowego **fun.** Można je również "przypisywać do zmiennych".

```
Xf=fun(X)-> X*X+1 end.
```

Zadanie 6.