dr inż. Piotr Matyasik
Piotr Klimiec

Erlang

Skrypt dla studentów 3 roku informatyki stosowanej
wydziału Elektrotechniki, Automatyki Informatyki
i Inżynierii Biomedycznej

Wstęp

Skrypt stanowi uzupełnienie laboratoriów z przedmiotu PWiR z zakresu programowania w języku Erlang. Składa się z 35 zadań o rosnącym poziomie trudności: od prostego operowania na listach poprzez implementację konkretnych problemów programistycznych. Rozwiązania do każdego zadania znajdują się w oddzielnym pliku *zadania.erl*.

Celem pracy było zebranie i napisanie jak największej możliwej liczby rozwiązań poszczególnych zadań. Takie podejście umożliwi studentom zapoznanie się z różnymi sposobami rozwiązywania typowych problemów.

Każde zadanie zostało opatrzone komentarzem o bieżącej liczbie rozwiązań. Zaleca się próbowanie znalezienia wszystkich rozwiązań przed analizą gotowych już rozwiązań.

Powodzenia! ☺

1. Programowanie sekwencyjne: działania na listach

Zad. 1. Napisz funkcję `sum/1`, która dla podanej liczby naturalnej N zwróci sumę wszystkich liczb naturalnych od 1 do N.

Przykład: IN: N = 3; OUT: 6

Liczba rozwiązań: 4

Zad. 2. Napisz funkcję `sum/2`, która dla danych liczb N i M, gdzie $N \leq M$, zwróci sumę liczb pomiędzy N i M. Jeżeli $N > M$, to zakończ proces.

Przykład: IN: N = 2, M = 5; OUT: 14

Liczba rozwiązań: 4

Zad. 3. Napisz funkcję, która dla danego N zwróci listę postaci `[1,2,...,N-1,N]`.

Przykład: IN: `create(3)`; OUT: `[1,2,3]`

Liczba rozwiązań: 3

Zad. 4. Napisz funkcję, która dla danego N zwróci listę formatu `[N,N-1,...,2,1]`.

Przykład: IN: `reverse_create(3)`; OUT: `[3,2,1]`

Liczba rozwiązań: 3

Zad. 5. Napisz funkcję, która wyświetli liczby naturalne pomiędzy 1 a N. Każda liczba ma zostać wyświetlona w nowym wierszu.

Liczba rozwiązań: 3

Zad. 6. Napisz funkcję która wyświetli wszystkie liczby parzyste pomiędzy 1 a N. Każda liczba ma zostać wyświetlona w nowym wierszu.

Liczba rozwiązań: 3

Zad. 7. Napisz funkcję, która dla podanej listy L oraz liczby całkowitej N zwróci listę wszystkich liczb z listy L, które są mniejsze bądź równe liczbie N.

Przykład: IN: `filter([1,2,3,4,5], 3)`; OUT: `[1,2,3]`

Liczba rozwiązań: 3

Zad. 8. Napisz funkcję, która odwróci porządek wszystkich elementów w tablicy.

Przykład: IN: `reverse([1,2,3])`; OUT: `[3,2,1]`

Liczba rozwiązań: 2

Zad. 9. Napisz funkcję, która dla danej listy list scali wszystkie liczby.

Przykład: IN: `concatenate([[1,2,3],[],[4, five]])`; OUT: `[1,2,3,4,five]`

Liczba rozwiązań: 2

Zad. 10. Napisz program liczący długość listy.

Przykład: IN: `[1,2,3,4,5]`; OUT: `5`

Liczba rozwiązań: 4

Zad. 11. Napisz program podający najmniejszy element listy (`min/1`).

Przykład: IN: `[1,2,6,3,-4,8]`; OUT: `-4`

Liczba rozwiązań: 4

Zad. 12. Napisz program zwracający krotkę 2-elementową z najmniejszym i największym elementem listy (`min_max/1`).

Przykład: IN: `[1,2,3,4,5]`; OUT: `{1,5}`

Liczba rozwiązań: 3

Zad. 13. Napisz funkcję `factorial(N)`, która obliczy silnię z liczby N.

Przykład: IN: `factorial(5)`; OUT: `120`

Liczba rozwiązań: 3

Zad. 14. Wypisz N kolejnych liczb trójkątnych.
Wejście: N — liczba określa, ile liczb trójkątnych zwrócić
Wyjście: Lista liczb trójkątnych

Przykład: IN: `5`; OUT: `[1,3,6,10,15]`

Liczba rozwiązań: 1

Zad. 15. Znajdź wszystkie trójki pitagorejskie dla $L =$ `[1,2,3,4,5,6,7,8,9,10]`. Wynik wypisz w postaci listy list.

Przykład: `[[a1,b1,c1],[a2, b2, c2],...]`

Liczba rozwiązań: 1

Zad. 16. Zbadaj, czy elementy danej listy składającej się cyfr tworzą liczbę palindromiczną.

Przykład: IN: `[1,2,3,2,1]`; OUT: `TAK`
IN: `[1,2,3,3,1]`; OUT: `NIE`

Liczba rozwiązań: 1

Zad. 17. Napisz program, który dla zadanej listy L zwróci listę zawierającą tylko elementy parzyste z listy L.

Przykład: IN: `[1,2,3,4,5,6]`; OUT: `[2,4,6]`

Liczba rozwiązań: 2

Zad. 18. Na podstawie danej listy L zrób nową listę niezawierającą liczb całkowitych.

Przykład: IN: `[jamnik, 1, {ja}, 3]`; OUT: `[jamnik, {ja}]`

Liczba rozwiązań: 3

Zad. 19. Napisz nieskończoną pętlę, która dla podanej listy L będzie wyświetlała każdy element w osobnym wierszu.

Liczba rozwiązań: 1

Zad. 20. Napisz funkcję, która dla podanej listy L i indeksu `Index` zwróci nową listę, gdzie element pod wskazanym indeksem podwoi swoją wartość.

Przykład: IN: L `= [1,2,3,4,5,6]`, `Index = 4`; OUT: `[1,2,3,8,5,6]`

Liczba rozwiązań: 2

Zad. 21. Napisz funkcję, która dla dwóch podanych List $L1$ i $L2$ (tej samej długości) połączy je tworząc nową listę. Każdy element w nowej liście ma być maksimum z wartości lokalnej w liście $L1$ i $L2$.

Przykład: IN: `L1 = [1,5,3]`, `L2 = [7,2,6]`; OUT: `L3 = [7,5,6]`

Liczba rozwiązań: 3

Zad. 22. Napisz funkcję, która usypia dany proces na X milisekund. Ma działać jak `timer:sleep()`.

Liczba rozwiązań: 1

Zad. 23. Napisz funkcję, która dla danej wartości N oraz listy L zwróci krotkę zawierającą dwie listy postaci `{Lmin, Lmax}`. `Lmin` ma być listą zawierającą elementy mniejsze od N. `Lmax` ma być listą zawierającą elementy większe bądź równe N.

Liczba rozwiązań: 3

Zad. 24. Zaimplementuj algorytm sortowania QuickSort.
Wskazówka: spróbuj różnych metod wyboru pivota!

Liczba rozwiązań: 2

---

2. Programowanie współbieżne: tworzenie procesów

Zad. 25. Napisz program, który zwróci listę N procesów.

Liczba rozwiązań: 3

Zad. 26. Napisz program, który dla podanej listy procesów L roześle do każdego procesu z listy wiadomość „hello”.

Liczba rozwiązań: 3

Zad. 27. Napisz prostą bazę danych. Ma istnieć możliwość zapisu, usunięcia oraz odczytu wszystkich elementów. Baza ma działać jako oddzielny proces i ma zapisywać, usuwać elementy z wewnętrznej listy. Do komunikacji z bazą używaj funkcji `add/1`, `delete/1`, oraz `show/0`.

Liczba rozwiązań: 1

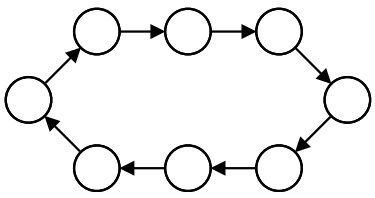
Zad. 28. Napisz prosty serwer, który będzie zwracał wiadomość, jaką otrzymał od klienta (usługa „echo”). Serwer ma przyjmować komunikaty od dowolnego procesu. Pid serwera przekazywany jest jako parametr, nie programuj na sztywno!

Liczba rozwiązań: 1

Zad. 29. Napisz funkcję, która odpali dwa procesy i prześle N razy wiadomość pomiędzy nimi tam i z powrotem. Po przesłaniu wszystkich komunikatów zakończ pracę procesów.

Liczba rozwiązań: 1

Zad. 30. Napisz funkcję, która odpali N procesów połączonych w pierścień (rys. 1). Następnie prześlij dowolną wiadomość pomiędzy nimi M razy. Kiedy przesyłanie wiadomości się zakończy, zabij wszystkie procesy, które tworzyły pierścień.



Rysunek 1. Procesy połączone w pierścień

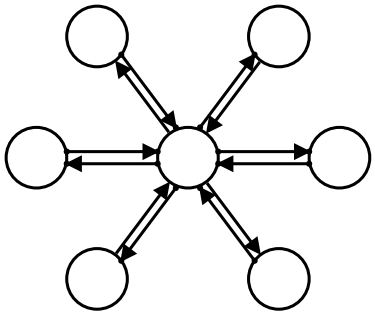
Liczba rozwiązań: 1

Zad. 31. Napisz funkcję, która odpali 10 procesów i prześle wiadomość do pierwszego z nich. Każdy proces przekazuje wiadomość dalej, po czym kończy swoje życie. Ostatni proces na liście wysyła wiadomość do funkcji startowej, która utworzyła procesy.

Przykład: `funkcja_startowa()` → `proces1` → `proces2` → `...` → `proces10` → `funkcja_startowa(): koniec`

Liczba rozwiązań: 1

Zad. 32. Napisz funkcję, którą uruchomi N procesów w połączeniu gwiazdy (rys. 2), a następnie prześle wiadomość do każdego z nich M razy. Proces po otrzymaniu wiadomości ma ją zwrócić. Otrzymana wiadomość powinna zostać wyświetlona na ekranie.



Rysunek 2. Procesy w konfiguracji gwiazdy

Liczba rozwiązań: 1

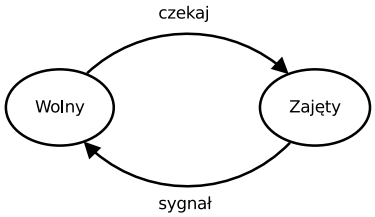
Zad. 33. Narysuj zaimplementowaną FST (skończona maszyna stanowa).

Wskazówka: *State*: każdy stan reprezentowany jest poprzez funkcję. *Event*: zdarzenie reprezentowane jest poprzez wiadomość.

Liczba rozwiązań: 1

Zad. 34. Zaimplementuj mechanizm MUTEXu.

Wskazówka: implementację oprzyj o schemat maszyny dwustanowej podany na rysunku 3.



Rysunek 3. Schemat maszyny dwustanowej

Liczba rozwiązań: 1

Zad. 35. Zaimplementuj „Consumer/Producer pattern”. Wykorzystaj mutex z poprzedniego zadania do ochrony danych danych współdzielonych. Producent produkuje dane i umieszcza je we wspólnym buforze, który jest listą 5-elementową. Liczba Konsumentów jest dowolna i ustala ją się przy wywołaniu programu. Producent jest tylko jeden. Chodzi o to, żeby zaobserwować przepływ sterowania w programie.

Liczba rozwiązań: 1
