1. Pytania zamknięte

Zaznacz prawdziwe zdanie dotyczące obiektów chronionych w Adzie:

a) ochronę dostępu do ob. chronionego wykonuje dedykowane zadanie
b) wywołanie procedury ob. chronionego powoduje reewaluację dozorów dla wejść
c) tylko wejścia umożliwiają zmianę stanu ob.chronionego
d) funkcje służą do modyfikacji stanu tylko wybranych ob. chronionych w Adzie

Spotkania w Adzie są:

a) leniwe
b) symetryczne
c) niesymetryczne
d) asynchroniczne

Symbol '&' w Adzie służy do:

a) konkatenacji łańcuchów znaków, tablic
b) oznacza liczby szesnastkowe
c) nie występuje w Adzie
d) wyłuskania adresu zmiennej

Wywołanie pm:d(). zwróci:

```
-module (pm).
-export ( [ pm/2, a/0, b/0, c/0, d/0 ]).
pm(list1, list2) -> 1;
pm(n, m) -> 2;
pm(X, X) -> 3;
pm( [ elem|elems ] , [ x|xs ]) -> 4;
pm( [ Elem|Elems ] , [ Elem|Elems ]) -> 5;
pm( [ _A|_As ] , [ _B|_Bs ]) -> 6;
pm( { _X } , _Y) -> 7;
pm( { A,B } , { B,A }) -> 8;
pm( { _A,_B } , { _C,_D }) -> 9;
pm(_X, _Y) -> 10.

a() -> pm( { a_atom } , another).
b() -> pm( [ elem,mints ] , [ candy,crush ]).
c() -> pm( { varied,flowers } , { flowers,varied }).
d() -> pm( [ cross, bow, service ] , [ cross, bow, service ]).
```

a) 3
b) 8

c) 7
d) 6

Która odpowiedź jest prawidłowa <<A:4,B:4,C:8>> = <<56,66>>.

a) A=5,B=6,C=66
b) A=3,B=8,C=8#102
c) A=4,B=6,C=2#001000010
d) A=1,B=6,C=65

Która odpowiedź jest prawidłowa <<A:3, B:4, C:8>> == <<56, 66>>

a) obie wymienione wartości są prawidłowe
b) A = 3, B = 8, C = 8#102
c) żadne z wymienionych
d) A = 3, B = 8, C = 66

[X+Y || X <- [1,2], Y <- [2,3]].
Wynikiem będzie:

a) [{1,2},{1,3},{2,2},{2,3}]
b) [[3],[4],[4],[5]]
c) [[1,2],[1,3],[2,2],[2,3]]
d) [3,4,4,5]

Operacje bitowe możemy wykonywać na:

a) typie resztowym
b) nie są dostępne
c) typie wyliczeniowym
d) dowolnym typie całkowitym

Weather = [{toronto, rain}, {montreal, storms}, {london, fog}, {paris, sun}, {boston, fog}, {vancouver, snow}].

[X || {X, Y} <- Weather, (Y == fog) or (Y == snow)].

Wynikiem będzie:

a) [boston]
b) [london,boston,boston ,vancouver]
c) []
d) [london,boston,vancouver]

Operator przypisania w j. Ada można przeładować:

a) tylko w wersji komercyjnej kompilatora
b) nie można tego zrobić
c) można ale tylko w wersji Ada95
d) w typie pochodnym

Typy kontrolowane w Adzie (wybierz prawdę)

a) muszą być zdefiniowane dla typów znakowych (pochodnych Character)
b) pozwalają programiście na zdefiniowanie metod wykonywanych przy: inicjalizacji, finalizacji i poprawianiu po przypisaniu
c) wymuszają zdefiniowanie metod wykonywanych przy: inicjalizacji, finalizacji i poprawianiu po przypisaniu
d) muszą być zdefiniowane dla typów znakowanych bo inaczej będą one źle działać

Dla typu ograniczonego w Adzie (wybierz prawdziwe zdanie):

a) nie można zdefiniować operatorów :=, =, /=
b) nie są automatycznie generowane operatory :=, =, /=
c) operatory :=, =, /= automatycznie trafiają do sekcji prywatnej
d) wartości są ograniczone tylko do podanego podzakresu

[{(X), Y} || X <- [1,2,3], Y <- [a,b]].
Wynikiem będzie:

a) [{1,a},{1,b},{2,a},{2,b},{3,a},{3,b}]
b) [[{1,a}],[{1,b}],[{2,a}],[{2,b}],[{3,a}],[{3,b}]]
c) {(1),a},{(1),b},{(2),a},{(2),b},{(3),a},{(3),b}]
d) [[1,a],[1,b],[2,a],[2,b],[3,a],[3,b]]

lists:zipwith(fun(X,Y) -> {X, Y} end,[1,2,3],[a,b,c]).
Wynikiem będzie:

a) [{1,a},{b,2},{c,3}]
b) {{1,a},{2,b},{3,c}}
c) [{1,a},{2,b},{3,c}]
d) [{a,b,c},{1,2,3}]

Wywołanie pm:a(). zwróci:

```
-module (pm).
-export ( [ pm/2, a/0, b/0, c/0, d/0 ]).
pm(list1, list2) -> 1;
pm(n, m) -> 2;
pm(X, X) -> 3;
pm( [ elem|elems ] , [ x|xs ]) -> 4;
```

```
pm( [ Elem|Elems ] , [ Elem|Elems ]) -> 5;
pm( [ _A|_As ] , [ _B|_Bs ]) -> 6;
pm( { _X } , _Y) -> 7;
pm( { A,B } , { B,A }) -> 8;
pm( { _A,_B } , { _C,_D }) -> 9;
pm(_X, _Y) -> 10.

a() -> pm( { a_atom } , another).
b() -> pm( [ elem,mints ] , [ candy,crush ]).
c() -> pm( { varied,flowers } , { flowers,varied }).
d() -> pm( [ cross, bow, service ] , [ cross, bow, service ]).
```

a) 9
b) 7
c) 8
d) 10

Wybierz prawdziwe wyrażenie (Ada):

a) (2+3)*4 = 2+(3*4)
b) 6/4 = 1.5
c) 6/2 > 21 - 1
d) 2+8 /= 28

lists:foldl(fun(X, Sum) -> X - Sum end, 0, [1,2 ,3,4,5]).
Wynikiem będzie:

a) 15
b) 3
c) -13
d) 2

Bariery nie są wartościowanie w obiekcie chronionym:

a) po wykonaniu funkcji
b) przy wywołaniu wejścia
c) po opuszczeniu sekcji krytycznej przez zadanie będące wcześniej w kolejce wywołań
d) po wykonaniu procedury operującej na wartościach argumentów będących elementami bariery

Wywołanie pm:c(). zwróci:

```
-module (pm).
-export ( [ pm/2, a/0, b/0, c/0, d/0 ]).
pm(list1, list2) -> 1;
pm(n, m) -> 2;
```

```
pm(X, X) -> 3;
pm( [ elem|elems ] , [ x|xs ]) -> 4;
pm( [ Elem|Elems ] , [ Elem|Elems ]) -> 5;
pm( [ _A|_As ] , [ _B|_Bs ]) -> 6;
pm( { _X } , _Y) -> 7;
pm( { A,B } , { B,A }) -> 8;
pm( { _A,_B } , { _C,_D }) -> 9;
pm(_X, _Y) -> 10.

a() -> pm( { a_atom } , another).
b() -> pm( [ elem,mints ] , [ candy,crush ]).
c() -> pm( { varied,flowers } , { flowers,varied }).
d() -> pm( [ cross, bow, service ] , [ cross, bow, service ]).
```

a) 6
b) 9
c) 8
d) 7

Typ Character w Adzie należy do typów (wybierz prawdę)

a) wyliczeniowych
b) stałoprzecinkowych
c) żaden z wymienionych
d) rzeczywistych

Która deklaracja procedury jest nieprawidłowa:

a) procedure Delete_File(Integer : in A);
b) procedure Delete_File(FOR : in A);
c) procedure Delete_File(X : in Integer);
d) procedure Delete_File(Y : in Float);

Wywołanie pm:b(). zwróci:

```
-module (pm).
-export ( [ pm/2, a/0, b/0, c/0, d/0 ]).
pm(list1, list2) -> 1;
pm(n, m) -> 2;
pm(X, X) -> 3;
pm( [ elem|elems ] , [ x|xs ]) -> 4;
pm( [ Elem|Elems ] , [ Elem|Elems ]) -> 5;
pm( [ _A|_As ] , [ _B|_Bs ]) -> 6;
pm( { _X } , _Y) -> 7;
pm( { A,B } , { B,A }) -> 8;
pm( { _A,_B } , { _C,_D }) -> 9;
```

```
pm(_X, _Y) -> 10.

a() -> pm( { a_atom } , another).
b() -> pm( [ elem,mints ] , [ candy,crush ]).
c() -> pm( { varied,flowers } , { flowers,varied }).
d() -> pm( [ cross, bow, service ] , [ cross, bow, service ]).
```

a) 8
b) 9
c) 6
d) 7

Wyróżnik pozwala na:

a) jest wskaźnikiem do funkcji
b) parametryzację wybranych typów
c) dostęp do zadania przez wskaźnik
d) dostęp do elementów tablicy

Który z elementów nie jest atomem dla j. Erlang

a) Elka
b) a
c) 'Elka'
d) zOO

Przy deklaracji: type Day is (Mon, Tue, Wed, Thu, Fri, Sat, Sun); wartość wyrażenia Day'Pred(Thu) to:

a) 2
b) Wed
c) Day
d) Mon

Wskaż niepoprawną deklarację:

a) procedure DFile(Int: in A)
b) function DFile(X: in out A) return D //funkcja musi mieć wszystkie parametry typu in
c) procedure DFile(X: out Integer)
d) function DFile(Y: in A) return D

Wynikiem lists:zipwith(fun(X,Y) - > [X,[X|[Y]]] end, [1,2,3] , [a,b,c])

a) [[1,[1,a]],[2,[2,b]],[3,[3,c]]]
b) [[1,[1|a]],[2,[2|b]],[3,[3|c]]]
c) [{1,1,a},{2,2,b},{3,3,c}

d) {[1,a,1],[2,b,2],[3,c,3]}

Wynikiem lists:foldl(fun(X,Sum) -> X - Sum end, 3, [X-1 || X <- lists:seq(1,5), X > 2]).

a) wtf xd
b) 0
c) 3
d) 5

Instrukcja abort powoduje (zaznacz prawdę)

a) natychmiastowe, bezwarunkowe zakończenie zadania
b) usunięcie zadania z kolejki oczekujących na wejście do ob. chronionego
c) ustawienie statusu Va_t'em
d) wywołanie procedury stop() zadania

Parametr klasowy T'Class w Adzie (zaznacz fałsz)

a) ma zawsze stały rozmiar i dlatego pozwala na wskazywanie dowolnego typu pochodnego
b) obejmuje wszystkie typy wyprowadzone z T
c) pozwala na przechowywanie wartości dowolnego typu wyprowadzonego z T
d) wchodząc w skład parametrów podprogramów wyklucza je z operacji podstawowych

Zakreśl poprawne zdanie odnoszące się do wykonania w przeplocie:

a) nowe zadanie po zakończeniu poprzedniego
b) kilka zadao wykonywanych jednocześnie
c) jedno zadanie wykonywane w danym momencie, kilka rozpoczętych
d) następne zadanie rozpoczynane tylko po zakończeniu poprzedniego

CCS dopuszcza, aby definicje agentów były wzajemnie rekurencyjne (wybierz prawdę)

a) tylko dla agentów niewykorzystujących później rekurencji
b) tylko dla agenta pustego
c) tylko jeśli zostanie użyty operator złożenia równoległego
d) zawsze – nie znalazłem żadnych obostrzeń

Zakreśl poprawne zdanie odnoszące się do wykonania równoległego:

a) nowe zadanie po zakończeniu poprzedniego
b) kilka zadań wykonywanych jednocześnie
c) jedno zadanie wykonywane w danym momencie, kilka rozpoczętych
d) następne zadanie rozpoczynane tylko po zakończeniu poprzedniego

Zaznacz odpowiedź zawierającą tylko poprawne identyfikatory w j. Ada

a) Hello, 2Run, QuitNow – musi się zaczynać od litery
b) Quit Now, RemoveElEM – nie może zawierać spacji
c) Add_Delta, XXX
d) 2-gi_Rocznik, TestQuest – nie może zaczynać się od cyfry, ani zawierać znaku specjalnego innego niż _

2. Pytania otwarte

Napisz program, który wyliczy trójki pitagorejskie w zakresie 1 do n. Wynik ma być listą krotek.

```
fun(X) ->
   N = lists:seq(1, X),
   [ {A,B,C} ||
      A <- N,
      B <- N,
      C <- N,
      A < B,
      A*A+B*B == C*C
   ].
```

INNY przykład z dokumentacji:

```
pyth(N) ->
   [ {A,B,C} ||
      A <- lists:seq(1,N),
      B <- lists:seq(1,N),
      C <- lists:seq(1,N),
      A+B+C =< N,
      A*A+B*B == C*C
   ].
```

Napisz fragment kodu w języku Ada, który będzie uruchamiał procedurę `blabla` co 0,5 sekundy.Kod ma niwelować wpływ długości wykonania procedury `blabla` na czas pomiędzy uruchomieniami. Procedura `blabla` wykonuje się zawsze w mniej niż 0,5 sek.

```
procedure Pol_Sec is
        procedure blabla is
        begin
                //delay 0.3;
                Put_Line("Blabla");
        end blabla;
```

```
        Loop_Time : Time_Span := To_Time_Span(0.5);
        Next_Loop_Time : Time;

begin
        Next_Loop_Time := Clock;
        loop
                blabla;
                Next_Loop_Time := Next_Loop_Time + Loop_Time;
                delay until Next_Loop_Time;
        end loop;
end Pol_Sec;
```

Napisz funkcję w j. erlang, która podaną listę skróci o połowę uśredniając sąsiednie elementy. Funkcja ma obsługiwać też listy nieparzyste. Napisz wersje gdzie ostatni element jest pomijany i doklejany.
dest(0) = (src(0)+src(1))/2

```
short(X) -> short2(X, []).
short2([], Res) -> lists:reverse(Res);

short2([A], Res) -> short2([], [A | Res]);   //tutaj wersja “doklejany”
short2([A], Res) -> short2([], Res);         //tutaj wersja pomijany

short2([A,B|T], Res) ->
     Md = (A+B)/2,
     short2(T, [Md|Res])].
```

INNE rozwiązanie:

```
start(Lista) ->
  D = lists:length(Lista),
  D2 = math:round(D),
  process(Lista, D2).

process([H, H2 | T], 0) ->
  [H]; // doklejany
  []; // pomijany
process([H, H2 | T], D) ->
  [srednia(H, H2) | process([H2 | T], D-1)].

srednia(A, B) ->
  (A + B)/2
```

JESZCZE INNE rozwiązanie:

```
doAverage([]) -> [];
doAverage([H1]) -> [H1];
```

```
doAverage([H1,H2|T]) -> [(H1 + H2) / 2] ++ doAverage(T).
```

Napisz równoległą implementację funkcji map w Erlangu pmap(Fun,[Items]). Funkcja ma uwzględniać błędne działanie funkcji Fun na elementach listy.

```
map_parallel(Fun, List) ->
   Pids = map_elements(Fun, List),
   Results = gather_elements(Pids).

gather_elements([]) -> [];
gather_elements([Pid | Rest_pids]) ->
   Result = receive
      {Pid, ok, Res} -> Res;
      {Pid, error} -> error
   end,
   Rest_result =  gather_elements(Rest_pids),
   if
      (Result == error) or (Rest_result == error) ->
         error;
      true -> [Result | Rest_result]
   end.

map_element_thread(Parent_pid, Fun, El) ->
   try Fun(El) of
      Result -> Parent_pid ! {self(), ok, Result}
   catch
      _:_ -> Parent_pid ! {self(), error}
   end.

map_elements(_, []) -> [];
map_elements(Fun, [H | T]) ->
   Pid = spawn(?MODULE, map_element_thread, [self(), Fun, H]),
   Rest_pids = map_elements(Fun, T),
   [Pid | Rest_pids].

%%%% Zgrabniejsza wersja:

pmap(Function, List) ->
  % spawn process for each element, and gather their pids into list
  Pids = [spawn(?MODULE,execute, [self(), Function, El]) || El <- List],
  %gather the results of the processes (in order) into a list
  gather(Pids).

% Execute the function and send the result back to the receiver
execute(Recv, Function, Element) ->
  Recv ! {self(), catch(Function(Element))}.
```

```
gather([]) ->
  [];
gather([H | T]) ->
  receive
    {H, Ret} ->
      [Ret | gather(T)]
  end.

% example usage
% MODULE_NAME:pmap((fun(X)-> 10/X end),[1,2,0,3,4,5,0,-1]).

///inna wersja
pmap(_,[])->[];
pmap(FUN,L)->
    [begin
       spawn(?MODULE,usingFun,[self(),FUN,A]),
       receive    X->X end
    end || A<-L].

usingFun(Pid,FUN,A)->
    case catch(FUN(A)) of
       B -> Pid!B
    end.
```

Napisz program, który stworzy 10 zadań i wywoła ich wejścia. Po otrzymaniu wywołaniu wejścia zadania mają się zakończyć (Ada). Uwaga!! Program najpierw stworzy wszystkie zadania a później rozpocznie komunikację.

```
procedure zad1 is

task type A is
  entry E;
end;

task body A is
begin
  accept E;
end;

procesy: array(Integer range 1..10) of A;
begin
  for i in 1..10 loop
    procesy(i).E;
  end loop;
end zad1;
```

Napisz w Erlangu „fork bomb”. Funkcja start() ma utworzyć 10 procesów, każdy z nich ma utworzyć kolejnych 10 procesów itd. Po stworzeniu wspomnianych 10 procesów każdy proces „ojciec” ma zacząć wysyłać do nich wiadomości w nieskończonej pętli.

```
start() ->
Pids = [ spawn(fun start/0) || _ <- lists:seq(1,10)],
loop(Pids).

loop(Pids) ->
lists:foreach(fun(Pid) -> Pid ! {starting_pids, Pids} end, Pids),
loop(Pids).
```

Napisz program w j. Erlang, który stworzy 100 procesów. Po stworzeniu wszystkich procesów program ma odczekać 2 sek i wysyłać do stworzonych procesów wiadomość 'pa'. Po otrzymaniu wiadomości procesy potomne mają wypisać swój pid i zakończyć się.

```
start() ->
Pids = [ spawn(fun dziecko/0) || _ <- lists:seq(1,100)],
timer:sleep(2000),
lists:foreach(fun(Pid) -> Pid ! pa end, Pids).

dziecko() ->
receive
pa ->
io:fwrite("~p~n",[self()])
end.
```

Napisz fragment kodu w języku Ada, który pozwoli max przez 5sek oczekiwać na zakończenie wykonania procedury "FixWorld(W:World)". Jeśli procedura się nie zakończy w tym czasie to ma zostać wyświetlony komunikat.

```
with ada.text_io; use ada.text_io;
procedure fixworld is

procedure xx is
begin
   delay 3.0;
   put_line("xx");
end xx;

begin
   select
```

```
        delay 2.0;
        put_line("nie wykonano xx");
    then abort
        xx;
    end select;

end fixworld;
```

Napisz w Erlangu sam kod procesu, który będzie odsyłał do nadawcy jego własną wiadomość. Przychodząca wiadomość ma format {Pid,Data}. Gdzie Pid to identyfikator procesu nadawcy a Data to wiadomość do odesłania. Jeśli proces otrzyma wiadomości, które nie pasują do podanego wzorca to mają one zostać usunięte z kolejki i zignorowane.

```
loop() ->
receive
{Pid,Data} -> Pid ! {Pid,Data};
_ -> ok
end,
loop().
```

Napisz kod implementujący zegar Lamporta. (Pojedyncza funkcja procesu.)

```
lamport(N) ->

receive
{msg,T} when  T> N ->  lamport(T+1);
{msg,T} when  T =< N ->  lamport(N+1)
end.
```

Napisz w Adzie procedurę uruchamiającą równoległą inwersję obrazka w 256 odcieniach szarości na n zadaniach. Dane obrazka zdefiniowane są następująco: type PIC is array (POSITIVE range <>, POSITIVE range <>) of INTEGER;

```
with Ada.Text_IO;
use Ada.Text_IO;

type PIC is array (POSITIVE range <>, POSITIVE range <>) of INTEGER;
type PIC_Acc is access PIC;

task type wykonaj is
    entry start(Picture_Acc: PIC_Acc; poczatek, koniec, rozmiar_kolumny : Integer);
    entry koniec;
```

```
end wykonaj;

task body wykonaj is
   local_Img_Acc : PIC_Acc;
   local_poczatek, local_koniec, local_rozmiar : Integer;
begin
   accept start(Picture_Acc: PIC_Acc; poczatek, koniec, rozmiar_kolumny : Integer) do
      local_Img_Acc := Picture_Acc;
      local_poczatek := poczatek;
      local_koniec := koniec;
      local_rozmiar := rozmiar_Kolumny;
   end start;

   for i in Integer range local_poczatek..local_koniec loop
      for j in Integer range 1..local_rozmiar loop
         local_Img_Acc(i, j) := 255 - local_Img_Acc(i, j);
      end loop;
   end loop;

   accept koniec;
end;

procedure inwersja(Picture: PIC_Acc; X, Y, Count: in Integer) is
   Step: Integer;
   Taski : Array(1..Count) of wykonaj;
begin
   Step := X / Count;
   for i in 1..Count loop
      if (i = Count) then
         Taski(i).start(Picture,step*(i-1),X,Y);
      else
         Taski(i).start(Picture,step*(i-1),step*i,Y);
      end if;
   end loop;

   for i in 1..Count loop
      Taski(i).koniec;
   end loop;
end;
```

Napisz kod w języku Erlang, który będzie implementacją następującej maszyny stanowej:

$(A) --a-->A ; A--b-->B; A--c-->C; B--f-->F; B--c-->C; F--r-->A; C--d-->D ; D--g-->A ; D--q-->F;

Oznaczenia: X--y-->Z automat ze stanu X po wykonaniu akcji y przechodzi do stanu Z, () to stan początkowy. Akcje mają być przesyłane jako wiadomości do procesu realizującego automat. Akcje niepożądane (niezdefiniowane dla danego stanu) mają być ignorowane.

```
-module(maszyna_stanowa).
-compile([export_all]).

start() ->
   st = spawn(?MODULE,state,['A']),
   register(automat,st).

changeState(A) ->
   automat ! A.


state('A') ->
   receive
      a -> state('A');
      b -> state('B');
      c -> state('C');
      _ -> state('A')
   end;

state('B') ->
   receive
      f -> state('F');
      c -> state('C');
      _ -> state('B')
   end;

state('C') ->
   receive
      d -> state('D');
      _ -> state('C')
   end;

state('D') ->
   receive
      g -> state('A');
      q -> state('F');
      _ -> state('D')
   end;

state('F') ->
```

```
    receive
      r -> state('A');
      _ -> state('F')
    end.
```

Napisz kod w języku Erlang, który uruchomi 3 procesy A,B i C. Proces A będzie wysyłał do B wiadomości co 1sek, B po otrzymaniu wiadomości od A będzie ją przesyłał natychmiast do C, C będzie ją natychmiast przesyłał do A. Jeśli A nie otrzyma odpowiedzi od C w max 500ms to wyświetli komunikat i zakończy działanie wszystkich procesów.

```
start() ->
register(a_pid, spawn(procA)),  %register(apid,spawn(?MODULE, procA, [])),
register(b_pid, spawn(procB)),
register(c_pid, spawn(procC)).

procA() ->
whereis(b_pid) ! {msg},
receive
{msg} -> timer:sleep(1000),
procA()
after 500 ->
exit(whereis(b_pid), kill),
exit(whereis(c_pid), kill),
io:format("za dlugo")
end.

procB() ->
receive
{msg} -> whereis(c_pid) ! {msg}
end,
procB().

procC() ->
receive
{msg} -> whereis(a_pid) ! {msg}
end,
procC().
```

Napisz w Erlangu funkcję generującą liczby Fibonacciego na dwa sposoby: rekurencyjnie i iteracyjnie.

```
fibo_reku(0) -> 0;
fibo_reku(1) -> 1;
```

```
fibo_reku(N) -> fibo_reku(N-1) + fibo_reku(N-2).

fibo_iter(N) -> fibo_iter(N, 0, 1).
fibo_iter(0, Wynik, _) -> Wynik;
fibo_iter(Iter, Wynik, Next) -> fibo_iter(Iter-1, Next, Wynik+Next).
```

Napisz w Adzie procedurę normalizującą próbkę dźwięku na n zadaniach. Dane zdefiniowane są następująco:

type Sample is new Float range 0.0..1.0;
type SND is array (POSITIVE range <>) of Sample;

(Znajdź maksymalną wartość we wszystkich próbkach i pomóż każdą z nich przez 1/max)

```
procedure normalize(sound: in out SND, n: in POSITIVE) is
  task type worker is
    entry set(sound: in SND, max in Sample);
    entry get(sound: out SND);
  end;
  task body worker is
    samples: SND;
    maxSample: Sample;
  begin
    accept set(sound: in SND, max in Sample)
      samples := sound;
      maxSample := max;
    end;
    for i in samples'Range
      samples(i) := samples(i) * 1/maxSample;
    end
    accept get(sound: out SND)
      sound := samples
    end
  end;

  tasks: array(1..n) of worker
  max: Sample;
  spacing: Positive;
  partialResult: SND;
begin
  max := sound(sound'First);
  for i in sound'Range do
    if max < sound(i) then
      max := sound(i);
```

```
   end;
  end;

  spacing := sound'length mod n;

  for i in 1..(n-1) do
    tasks(i).set(sound((sound'first + (i-1) * spacing)..(sound'first + i * spacing - 1)), max);
  end;
  tasks(n).set(sound( (sound'first + (n-1) * spacing)..sound'last ), max);

  for i in 1..(n-1) do
    tasks(i).get(partialResult);
    sound((sound'first + (i-1) * spacing)..(sound'first + i * spacing - 1)) := partialResult;
  end;
  tasks(n).get(partialResult);
  sound( (sound'first + (n-1) * spacing)..sound'last) := partialResult;

end
```

Napisz w Erlangu program, który stworzy 100 procesów. Po stworzeniu wszystkich procesów program ma wysłać do pierwszego wiadomość, pierwszy ma do drugiego itd. Ostatni ma wysłać wiadomość do funkcji startującej. Po otrzymaniu wiadomości i wykonaniu akcji procesy mają się zakończyć.

```
start() ->
        Pids = [spawn(resend) || _ <- lists:seq(1,100)],
        register('A', self()).
        wyslijListe(Pids),
        receive
                msg -> io:format("koniec")
        end.

wyslijListe([H|T]) -> H ! T.

resend() ->
        receive
                [H,T] -> H ! T;
                [] -> whereis('A') ! msg;
        end.
```

Function bla(X:out Float) return Integer. Czy ta deklaracja jest poprawna - odpowiedź uzasadnij.

Nie jest, ponieważ funkcja może przyjmować tylko parametry typu ‘in’.

Jeśli w Adzie zadeklarujemy typ jako typ kontrolowany to…

Możemy zaimplementować własne wersje metod Initialize, Adjust i Finalize.

Initialize wywoływane jest przy tworzeniu obiektu (new MojTypKontrolowany). Adjust wywoływane przy przypisywaniu do typu MojTypKontrolowany. Finalize wywoływane przy usuwaniu obiektu (gdy osiągnięto koniec bloku w którym stworzony był obiekt typu MojTypKontrolowany, albo przypisano obiekt tego typu do jakiejś nowej zmiennej).

Opisz krótko (w punktach, diagram) jak działa instrukcja receive w Erlangu.

Kiedy wiadomości są wysyłane do procesu, są one przechowywane w takiej jakby “skrzynce odbiorczej”, do czasu kiedy proces ich nie odczyta. Są one tam przechowywane w kolejności w jakiej przyszły. To znaczy, że za każdym razem jak się pobiera wiadomość, zaczyna się od tej najstarszej.

Najstarsza wiadomość jest przyporządkowywana do wzorca, który przyszedł, aż do momentu jak się będzie zgadzał. Kiedy to się stanie, wiadomość jest usuwana ze “skrzynki odbiorczej” i dalszy kod jest wykonywany, aż do kolejnego wywołania receive.

Kiedy wiadomość nie zgadza się ze wzorcem, jest ona zachowana w kolejce (save queue) i jest próbowana następna wiadomość, z tego naszego jakby stosu wiadomości. Kiedy ta kolejna wiadomość “wejdzie”, ta pierwsza jest przywracana do “skrzynki odbiorczej” na swoje miejsce, żeby mogła być odebrana później.

Czy deklaracja jest poprawna, uzasadnij odpowiedź: function A(B : in C) return D;

Tak. W funkcjach wszystkie parametry są wejściowe.

Co to jest typ limited w Adzie?

Jeśli typ jest limited to obiektowi tego typu nie można przypisać wartości tego samego typu: To znaczy, jest to zabezpieczenie przed wykonywaniem płytkich kopii obiektów.

Jeżeli definiujemy pakiet implementujący jakiś typ chcemy ograniczyć operacje na tym typie do operacji zdefiniowanych w pakiecie, to typ można zdefiniować jako ograniczony. W takim przypadku, dla typu nie są generowane nawet operatory :=, = i /=. Ten sam efekt można osiągnąć deklarując zabronione operatory w części prywatnej pakietu. Typy ograniczone są pozbawione operatora przypisania. W języku Ada nie można przeładować operatora przypisania.

Różnice między instrukcjami abort i terminate w Adzie:

Terminate: Instrukcja terminate pozwala na zakończenie zadania (jest to normalne `zakończenie)´ w sytuacji, gdy jednostka macierzysta zadania jest zakończona (wykonana jest jej ostatnia instrukcja) oraz wszystkie zadania siostrzane i zależne od tego zadania sa albo zakończone, albo równiez moga wykonać instrukcje terminate.

Abort: Instrukcja abort powoduje przerwanie wykonywania zadania. Może być wykonana wyłącznie w innym zadaniu lub przez jednostke macierzysta. – Jeżeli zadanie zostaje przerwane, to równiez wszystkie zadania zalezne od niego, a takze podprogramy przez nie wywołane zostaja przerwane.

Realizacja funkcji map(F, Lista) współbieżnie. [Erlang]

```
parallelMap(F, List) ->
  Processes = spawnProcesses(F, List),
  askForOutput(Processes).

spawnProcesses(_, []) -> [];
spawnProcesses(F, [Head | Tail]) ->
  [spawn(?MODULE, singleMap, [F, Head]) | spawnProcesses(F, Tail)].

singleMap(Fun, Element) ->
  Output = Fun(Element),
  receive
    {get, Pid} ->
      Pid ! Output
  end.

askForOutput([]) -> [];
askForOutput([Process | Tail]) ->
  Process ! {get, self()},
  receive
    Output -> [Output | askForOutput(Tail)]
  end.
```

Opisz semantykę spotkań. Jaki to rodzaj komunikacji? Ada

Synchroniczny. Jedno zadanie wykonuje wejście przez konstrukcje NazwaZadania.WejscieZadania, a drugie zadanie musi posiadać w deklaracji blok ‘entry WejscieZadania’ a w ciele ‘accept WejscieZadania’ aby doszło do spotkania.

Zrobić funkcję w erlangu która z listy usuwa liczby nieparzyste [EDIT: coś było jeszcze, że funkcja ma wykrywać elementy dla których nie działa pojęcie parzysty - i automatycznie to wykrywać, a jeśli wyrzuci wyjątek to powinna iść do warunku true]

bez dodatkowego sprawdzania:

```
onlyEven([]) -> [];=
onlyEven([Head | Tail]) ->
  if
    Head rem 2 == 0 -> [Head | onlyEven(Tail)];
    true -> onlyEven(Tail)
  end.

dareFun(X) -> [Z || Z <-X , Z rem 2 == 0].
```

Różnice między Adą a Erlangiem

Ada:

- typowanie statyczne (typy określane na etapie kompilacji)
- proceduralna
- systemy twarde czasu rzeczywistego (restrykcyjne przestrzeganie ograniczeń czasowych)

Erlang:

- typowanie dynamiczne
- język funkcyjny
- systemy miękkiego czasu rzeczywistego (istnieją ograniczenia czasowe dla procesów, ale podejście do nich jest luźne)

Erlang suma elementów z listy z użyciem funkcji wbudowanych i bez

```
lists:sum(L).                                  // rozwiązanie 1 wbud.
lists:foldl(fun(X, Sum)-> X+Sum end, 0, L).    // rozwiązanie 2 wbud.

s(L) -> s(L,0).
s([], L) -> L;
s([H|T], L) -> s(T, L+H).
```

Opisz semantykę funkcji obiektu chronionego w Adzie

Obiekty chronione, podobnie jak zadania i pakiety, składają się z dwóch części: specyfikacyjnej i z definicyjnej (implementacji). W części specyfikacyjnej definiuje się dane oraz operacje, które można wykonywać na tych danych, natomiast w części definicyjnej umieszcza się treść tych operacji. W obiektach chronionych jako operacje można umieszczać zarówno wejścia, jak i procedury i funkcje.

Zasadniczą różnicą pomiędzy procedurą a funkcją w obiekcie chronionym jest to, że procedura może zmieniać wartości danych chronionych, natomiast funkcja może tylko odczytywać te wartości. Dodatkowo funkcje odczytujące wartości danych chronionych mogą w tym samym czasie zwracać wyniki dla różnych zadań, natomiast procedura może być w danym momencie wywołana tylko przez jedno zadanie. Odwołanie się do dowolnego wejścia (procedury, funkcji) obiektu chronionego zapisuje się w notacji kropkowej, np.:
X := Monitor.Czytaj;

Jaki rodzaj typowania posiada j. Ada

Typowanie statyczne (silne) – nadawanie typów zmiennym w czasie kompilacji programu.

W porównaniu do typowania dynamicznego, zaletami są możliwość większej optymalizacji oraz możliwość wykrycia większej liczby błędów w czasie kompilacji. Wadą jest natomiast konieczność pisania dużej ilości informacji o typach.

Przypisz do identyfikatora X1 funkcję dwuargumentową zawierającą sumę wartości argumentów (Erlang)

X1 = fun(A,B) -> A+B end.

Co to jest inwersja priorytetów?

Zadanie z sekcji krytycznej uzyskuje dostęp do zasobu kosztem innego zadania posiadającego wyższy priorytet.

Dlaczego stosuje się programowanie wielowątkowe/współbieżne?

- Lepsza eksploatacja zasobów
- Prostszy projekt programu w niektórych sytuacjach
- Bardziej elastyczne programy

Czym różni się przeplot od współbieżności w wykonywaniu zadań?

1. *Wykonanie sekwencyjne*. Poszczególne akcje procesu są wykonywane jedna po drugiej. Dokładniej: kolejna akcja rozpoczyna się po całkowitym zakończeniu poprzedniej.
2. *Wykonanie równoległe*. Kilka akcji jest wykonywanych w tym samym czasie. Jest to "prawdziwa" współbieżność, możliwa do uzyskania na komputerze z wieloma procesorami.

3. *Wykonanie w przeplocie*. Choć jednocześnie odbywa się wykonanie tylko jednej akcji, to jednak jest wiele czynności rozpoczętych i wykonywanych na zmianę krótkimi fragmentami.
4. *Wykonanie współbieżne*. Kolejna akcja rozpoczyna się przed zakończeniem poprzedniej. Zauważmy, że nie mówimy nic na temat tego, czy akcje te są wykonywane w tym samym czasie czy też w przeplocie. Tak naprawdę wykonanie współbieżne jest abstrakcją równoległości i zawiera w sobie zarówno wykonania równoległe jak i wykonania w przeplocie.

Przypisz do identyfikatora Fun funkcję zwracającą sumę wartości elementów listy będącej parametrem wywołania (Erlang).

```
F2 = fun(L) -> lists:sum(L) end.
```

Co to jest własność bezpieczeństwa w systemach komputerowych?

“Nigdy nie dojdzie do niepożądanej sytuacji”. Własność zapewniania, musi ją mieć i spełniać program współbieżny. Oznacza, że nigdy nie dojdzie do sytuacji niepożądanej (utrata/modyfikacja danych, błąd oprogramowania). Podawana zazwyczaj w specyfikacji zadania.

Napisz, kiedy wartościowane są bariery w obiekcie chronionym?

Wartościowanie barier odbywa się gdy:

1. Zadanie wywołuje jedno z wejść i skojarzona z tym wejściem bariera odwołuje się do zmiennej lub atrybutu, który mógł zostać zmieniony od czasu ostatniego wartościowania.
2. Zadanie wykonuje swoje operacje i opuszcza wejście lub procedure obiektu chronionego i ´ istnieją inne zadania (zadanie) zawieszone na wejściach z barierami, które odwołują się do zmiennej lub atrybutu, który mógł zostać zmieniony od czasu ostatniego wartościowania. ´ Bariery nie są wartościowane w wyniku wykonywania funkcji obiektu chronionego.

Ada. Znajdź element maksymalny w tablicy i podziel wszystkie elementy przez niego.

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Max_I_Podziel is
        Arr : Array (1..10) of Float;
        Max : Float;
```

```
begin
	Arr := ( 1..2 => 2.0, 5..8 => 4.0, others => 1.0);

	Max :=  Arr(Arr'First);
	for I in Arr'Range loop
		if Max < Arr(I) then
			Max := Arr(I);
		end if;
	end loop;

	Put_Line("Max: " & Max'Img);

	for I in Arr'Range loop
		Arr(I) := Arr(I) / Max;
		Put_Line(I'Img & ":" & Arr(I)'Img);
	end loop;
end Max_I_Podziel;
```

Co to jest sekcja krytyczna?

Fragment kodu, gdzie odbywa się dostęp procesu do zasobu współdzielonego.

Różnice pomiędzy Adą a C?

C:
• Trust the programmer.
• Don’t prevent the programmer from doing what needs to be done.
• Keep the language small and simple.
• Provide only one way to do an operation.
• Make it fast, even if it’s not guaranteed to be portable.

ADA:
• Trust the programmer, but verify through appropriate checking since programmers are human and make mistakes.
• Prevent the programmer from doing what shouldn’t be done.
• Keep the language kernel small and simple, but provide extension mechanisms to increase expressiveness.
• Provide one principal and intuitive way to do an operation.
• Make it reliable and portable, and depend on the compiler to produce efficient code.

Napisz w Adzie szkielet programu do obsługi przerwań (sygnałów).

protected Sensor is

```
        procedure Int_Handler;
        pragma Attach_Handler(Int_Handler, InterruptID);
private
        -- ...
end;

protected body Sensor is
        procedure Int_Handler is
        begin
                -- ...
        end;
        -- ...
end;
```

Zaimplementuj algorytm tyrana w Erlangu.

```
-module(tyrant).
-export([start/0, init/0, send/1, elect/1, wait/1, muted/0]).
start() ->
        Pids = gen(5) ++ [{spawn(?MODULE, muted, []), 10}],
        spawn(?MODULE, send, [Pids]),
        Pids.

gen(N) -> gen(N, []).
gen(0, Acc) -> Acc;
gen(N, Acc) -> gen(N - 1, [{spawn(?MODULE, init, []), N} | Acc]).

send(L) ->
        timer:sleep(500),
        send(L, L).
send([], _) ->
        done;
send([H|T], C) ->
        {Pid, HP} = H,
        Pid ! lists:partition(fun({_, P}) -> P > HP end, C -- [H]),
        send(T, C).

muted() ->
        receive
                {_, 'ELECTION'} ->
                        muted()
end.
init() ->
        receive
                {Higher, Lower} ->
```

```
                        ?MODULE:elect({Higher, Lower})
        end.
elect({Higher, Lower}) ->
        lists:foreach(fun({Pid, _}) -> Pid ! {self(), 'ELECTION'} end, Higher),
        receive
                'ANSWER' ->
                ?MODU          LE:wait({Higher, Lower});
                {Pid, 'ELECTION'} ->
                        io:format("~p: Answering ~p.~n", [self(), Pid]),
                        Pid ! 'ANSWER',
                        elect({Higher, Lower})
                after 1000 ->
                        io:format("~p: Nobody answered, I'm winner.~n", [self()]),
                        lists:foreach(fun({Pid, _}) -> Pid ! 'COORDINATOR' end, Lower)
        end.
wait({Higher, Lower}) ->
        io:format("~p: Waiting for coordinator msg~n", [self()]),
        receive
                'COORDINATOR' ->
                        io:format("~p: Received coordinator msg~n", [self()]),
                        done
                after 1500 ->
                        io:format("~p: No coordinator, reelecting~n", [self()]),
                        ?MODULE:elect({Higher, Lower})
end.
```

Odwracanie obrazu w Adzie.

```
with Ada.Text_IO;
use Ada.Text_IO;

procedure Main is
        type PIC is array(Positive range <>, Positive range <>) of Integer;
        tab: PIC(1..25,1..25);
        N: Integer := 20;

        task type Inwersja;

        task Koordynator is
                entry czyjeszcze(jeszcze: out Boolean; wiersz: out Integer);
                entry start;
                entry wynik;
        end Koordynator;

        task body Inwersja is
```

```
                w: Integer;
                jeszcze: Boolean;
        begin
                loop
                        Koordynator.czyjeszcze(jeszcze,w);
                        if jeszcze then
                                for i in tab'range(2) loop
                                        tab(w,i) := 255 - tab(w,i);
                                end loop;
                                else
                                        exit;
                        end if;
                end loop;
                Put_Line("ubity");
        end Inwersja;

        task body Koordynator is
                zadania: array(1..N) of Inwersja;
        begin
                accept start;
                        for i in tab'Range(1) loop
                                accept czyjeszcze(jeszcze: out Boolean; wiersz: out Integer)
do
                                        jeszcze := true;
                                        wiersz := i;
                                end czyjeszcze;
                        end loop;
                --konczenie zadan
                for i in 1..N loop
                        accept czyjeszcze(jeszcze: out Boolean; wiersz: out Integer) do
                                jeszcze := false;
                        end czyjeszcze;
                end loop;
                accept wynik;
        end Koordynator;

        begin
        for i in tab'Range(1) loop
                for j in tab'Range(2) loop
                        tab(i,j) := (255 + i + j) mod 256;
                end loop;
        end loop;
        Koordynator.start;
        Koordynator.wynik;
end Main;
```

Średnia z obrazu w Adzie.

```
with ADA.Text_IO;
use ADA.Text_IO;

procedure Srednia is
	type PIC is array(Positive range <>, Positive range <>) of Integer;
	tab: PIC(1..25,1..25);
	N: Integer := 20;
	Suma: Integer := 0;
	SumaTest: Integer := 0;
	tmp: Float;

	task type Sumator;

	task Koordynator is
		entry czyjeszcze(jeszcze: out boolean; wiersz: out Integer);
		entry wynik;
		entry start;
	end Koordynator;

	task body Koordynator is
		zadania: array(1..N) of Sumator;
	begin
		accept start;
			for i in tab'range(1) loop
				accept czyjeszcze(jeszcze: out boolean; wiersz: out Integer)
do
					jeszcze := true;
					wiersz := i;
				end czyjeszcze;
			end loop;
		for i in 1..N loop
			accept czyjeszcze(jeszcze: out boolean; wiersz: out Integer) do
				jeszcze := false;
				wiersz := 0;
			end czyjeszcze;
		end loop;
			accept wynik;
	end;

	task body Sumator is
		jeszcze: Boolean;
		wiersz: Integer;
	begin
```

```
            loop
                Koordynator.czyjeszcze(jeszcze, wiersz);
                if jeszcze then
                    for j in tab'range(2) loop
                        Suma := Suma + tab(wiersz, j);
                    end loop;
                else
                    exit;
                end if;
            end loop;
        end Sumator;

        begin
        for i in tab'Range(1) loop
            for j in tab'Range(2) loop
                tab(i,j) := (255 + i + j) mod 256;
                SumaTest := SumaTest + tab(i, j);
            end loop;
        end loop;
        Koordynator.start;
        Koordynator.wynik;
        tmp := Float(SumaTest)/Float(tab'length(1)*tab'length(2));
        tmp := Float(Suma)/Float(tab'length(1)*tab'length(2));
end;
```

Spotkanie w Adzie

Aby w ogóle do niego doszło, proces aktywny musi wywołać pewne wejście udostępniane przez proces pasywny. Wywołanie wejścia zadania biernego powoduje wstrzymanie procesu wywołującego aż do chwili, gdy sterowanie w procesie biernym dotrze do bloku **accept**. Proces wywołujący jest przy tym wstrzymywany w kolejce związanej z danym wejściem. Gdy sterowanie w zadaniu biernym dochodzi do **accept** dochodzi do randki między zadaniem biernym a zadaniem aktywnym znajdującym się na początku kolejki zadań oczekujących na danym wejściu.

1. Zadanie aktywne przekazuje do zadania biernego wartości wszystkich parametrów **in** i **inout**.
2. Zadanie aktywne zostaje wstrzymane do czasu zakończenia randki.
3. Zadanie bierne wykonuje instrukcje stanowiące treść bloku accept.
4. Po zakończeniu **accept** zadanie bierne przekazuje do zadania aktywnego wartości wszystkich parametrów **out** oraz **inout** i przechodzi do kolejnej instrukcji za instrukcją akceptującą.
5. Po odebraniu wartości parametrów wyjściowych zadanie aktywne wznawia swoje działanie.

Własność żywotności

Jeśli proces chce coś zrobić, to w końcu mu się uda.

Zakleszczenie

Globalny brak żywotności. Nic się nie dzieje, system nie pracuje, oczekując na zajście zdarzenia, które nigdy nie zajdzie.

Zagłodzenie

Lokalny brak żywotności. Proces lub grupa procesów nie może uzyskać dostępu do zasobu współdzielonego, bo ten ciągle jest zajęty i nigdy się nie zwolni.

Sprawiedliwość

Każdy proces dostanie swój ,,czas antenowy”

Asynchroniczna zmiana wątku sterowania

Umożliwia przerwanie wykonywania programu po zakończeniu instrukcji wyzwalającej (po ustalonym czasie lub zakończeniu wywołania wejścia itp.):

```
select
        delay 5.0;
        Put_line (”Czas minął”);
then abort
        obliczaj(...);
end select;
```

lists:zipwith(fun({X},Y) -> [X,[[X] -- [Y]]] end, [{1},{2},{3}], [a,b,c])

[[1, [[1]]], [2, [[2]]], [3, [[3]]]]

lists:zipwith(fun({X},Y) -> [X,[[X] ++ [Y]]] end, [{1},{2},{3}], [a,b,c])

[[1, [[1, a]]], [2, [[2, b]]], [3, [[3, c]]]]

W jaki sposób tworzy się proces w języku Erlang wykonujący funkcję F1?

Używając funkcji spawn w poniższy sposób:
Pid = spawn(F1)

Jak w języku Erlang przesyła się wiadomości (Mesg) do procesu posiadając jego identyfikator (Pid)?

Pid ! Mesg

Jak zrealizowana jest komunikacja między procesami w języku Erlang?

Jedynym sposobem interakcji są wiadomości.

Jaki będzie wynik wykonania następującej instrukcji w języku Erlang:
lists:map(fun(X) -> {X, X+1} end,[1,2,3]).

Funkcja map mapuje dane wejściowe zgodnie z podaną dla niej funkcją.
[{1, 2}, {2, 3}, {3, 4}]

Jakie operacje są możliwe do zdefiniowania dla typu kontrolowanego w Adzie?

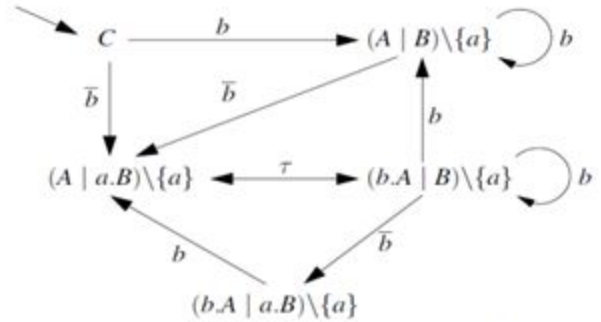
- inicjalizacja po stworzeniu
- finalizacja przed unicestwieniem
- poprawka po przypisaniu

Jak wygląda szkielet obsługi wyjątków w języku Ada?

```
begin
-- . . .
exception
when Constraint_Error | Program_Error =>
 --. . .
when Storage_Error
--. . .
when others =>
--. . .
end ;
```

Narysuj graf stanów osiągalnych dla następującego systemu zapisanego w CCS:
A = 'a.b.A + a.b.0
B = 'b.a.B + b.B

C = (A | B) \ {a}



[analiza rozwiązania w pliku ccs-rozwiazanie.pdf na FTP]

3. Pytania z GitHuba

3.1 Działania na listach

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

```
-module(mod).
-compile(export_all).

sum(1) -> 1;
sum(N) -> N + sum(N-1).
```

2. Napisz funkcję sum/2, która dla danych liczb N i M, gdzie N<=M, zwróci sumę liczb pomiędzy N i M. Jeżeli N>M, to zakończ proces.

```
-module(mod).
-compile(export_all).

sum(N, M) when N > M -> exit(self(), kill);
sum(M, M) -> M;
sum(N, M) -> N + sum(N+1, M).
```

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

```
-module(mod).
-compile(export_all).

create(N) -> create(1, N).
```

```
create(N, N) -> [N];
create(A, N) -> [A | create(A+1, N)].
```

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

```
-module(mod).
-compile(export_all).

reverse_create(0) -> [];
reverse_create(N) -> [N | reverse_create(N-1)].
```

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

```
-module(mod).
-compile(export_all).

func(N) -> func(1, N).

func(N, N) ->
   io:format("Number:~p~n",[N]);
func(A, N) ->
   io:format("Number:~p~n",[A]),
   func(A+1, N).
```

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

```
-module(mod).
-compile(export_all).

func(N)  when N > 1 -> func(2, N);
func(_) -> {brak}.

func(A, N) when A =< N ->
   io:format("Number:~p~n",[A]),
   func(A+2, N);
func(_, _) ->
   {koniec}.
```

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

```
-module(mod).
-compile(export_all).

filter([H|T], N) when H =< N ->
  [H | filter(T, N)];
filter(_, _) ->
  [].
```

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

```
-module(mod).
-compile(export_all).

reverse([H]) -> [H];
reverse([H|T]) -> reverse(T) ++ [H].
```

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

```
-module(mod).
-compile(export_all).

concatenate([]) -> [];
concatenate([H|T]) -> H ++ concatenate(T).
```

10. Napisz program liczący długość listy.

```
-module(mod).
-compile(export_all).

listLength([]) -> 0;
listLength([_|T]) -> 1 + listLength(T).
```

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

```
-module(mod).
-compile(export_all).

minL([H|T]) -> minL(T, H).

minL([], Min) -> Min;
minL([H|T], Min) when H < Min -> minL(T, H);
```

```
minL([_|T], Min) -> minL(T, Min).
```

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

```
-module(mod).
-compile(export_all).


func([H|T]) -> func(T, H, H).

func([], Min, Max) -> {Min, Max};
func([H|T], Min, Max) when H < Min -> func(T, H, Max);
func([H|T], Min, Max) when H > Max -> func(T, Min, H);
func([_|T], Min, Max) -> func(T, Min, Max).
```

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

```
-module(mod).
-compile(export_all).

factorial(0) -> 1;
factorial(N) -> N * factorial(N-1).
```

14. Wypisz N kolejnych liczb trójkątnych.

```
-module(mod).
-compile(export_all).

func(1) -> [1];
func(N) -> func(N-1) ++ [lists:foldl(fun(X, Sum)-> X+Sum end, 0, lists:seq(1,N))].
```

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

```
-module(mod).
-compile(export_all).

func(L) -> [[X, Y, Z] || X <-L, Y <-L, Z <- L, X*X + Y*Y == Z*Z].
```

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

```
-module(mod).
-compile(export_all).

func(L) -> func(L, lists:reverse(L)).

func([],[]) -> {tak};
func([H1|T1], [H2|T2]) when H1 == H2 -> func(T1, T2);
func(_, _) -> {nie}.
```

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

```
-module(mod).
-compile(export_all).

func([]) -> [];
func([H|T]) when H rem 2 == 0 -> [H | func(T)];
func([_|T]) -> func(T).
```

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

```
-module(mod).
-compile(export_all).

func([]) -> [];
func([H|T]) when not is_integer(H) -> [H | func(T)];
func([_|T]) -> func(T).
```

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

```
-module(mod).
-compile(export_all).

func(L) -> func(L, L).

func([], L) ->
    func(L, L);
func([H|T], L) ->
    io:format("~p ~n",[H]),
```

```
    timer:sleep(250),
    func(T, L).
```

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

```
-module(mod).
-compile(export_all).

func(L, Index) ->
  NewValue = 2 * lists:nth(Index, L),
  lists:sublist(L, Index-1) ++ [NewValue] ++ lists:sublist(L, Index+1, length(L)).
```

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.

```
-module(mod).
-compile(export_all).

func([],[]) -> [];
func([H1|T1], [H2|T2]) ->
  if
    H1 > H2  -> [H1 | func(T1, T2)];
    H1 =< H2 -> [H2 | func(T1, T2)]
  end.
```

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

```
-module(mod).
-compile(export_all).

sleep(N) ->
  receive
     after N ->
        ok
  end.
```

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.

```
-module(mod).
-compile(export_all).

func(N, L) -> func(N, L, [], []).

func(_N, [], Lmin, Lmax) -> {Lmin, Lmax};
func(N, [H|T], Lmin, Lmax) when H < N -> func(N, T, [H|Lmin], Lmax);
func(N, [H|T], Lmin, Lmax) -> func(N, T, Lmin, [H|Lmax]).
```

24. Zaimplementuj algorytm sortowania QuickSort.

```
-module(mod).
-compile(export_all).

% tablica zero elementowa juz jest posorotwana
qsort([]) -> [];
% tablica jedno elementowa już jest posortowana
qsort([H]) -> [H];
qsort(L)  when length(L) > 1 ->
  % wybor elementu który będzie pivotem
  PivotIndex = length(L) div 2,
  % pbranie wartości pivotu
  PivotElement = lists:nth(PivotIndex, L),
  % usunięcie wartości Pivota z listy
  NewList = lists:delete(PivotElement, L),
  % posegreguj elementy na liscie
  {Lmin, Lmax} = func(PivotElement, NewList),
  % wywolaj rekurencyjnie sortowanie na dwóch podtablicach.
  qsort(Lmin) ++ [PivotElement] ++ qsort(Lmax).

% funkcja pomocnicza : dzieli tablicę na dwie podtablice
func(N, L) -> func(N, L, [], []).

func(_N, [], Lmin, Lmax) -> {Lmin, Lmax};
func(N, [H|T], Lmin, Lmax) when H < N -> func(N, T, [H|Lmin], Lmax);
func(N, [H|T], Lmin, Lmax) -> func(N, T, Lmin, [H|Lmax]).
```

3.2 Tworzenie procesów

25. Napisz program, który zwróci listę N procesów.

```
-module(mod).
```

```
-compile(export_all).


%---- tworzenie listy procesów -----
func(N) -> [spawn(mod, loop, []) || _<-lists:seq(1,N)].

%------ funkcja procesu -----
loop() ->
   io:format("Pid ~p ~n",[self()]).
```

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

```
func([]) ->
   {ok};
func([H|T]) ->
   H!hello,
   func(T).
```

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.

```
-module(mod).
-compile(export_all).


% --- tworzy bazę + rejestruje nazwę celem łatwego dostępu do niej z innych funkcji
start() ->
  register(database, spawn(mod, loop, [[]])).


% --- proces bazy danych ----
% L : jest to lista do której będę dodawane/odejmowane elementy

loop(L) ->
   receive
     {add, Data} ->
        loop(L++[Data]);
     {delete, Data} ->
          loop(lists:delete(Data, L));
     {show} ->
        io:format("~p ~n",[L]),
```

```
        loop(L)
    end.


%------ Obsługa bazy danych ---------

add(Data) -> database!{add, Data}.
delete(Data) -> database!{delete, Data}.
show() -> database!{show}.
```

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.

```
-module(mod).
-compile(export_all).



% -----  proces server  -------

loop() ->
    receive
      {req, From, Msg} ->
          From ! {response, self(), Msg}
    end,
    loop().


% -------  funkcja klienta  --------
% ServerPid :  pid procesu który odapala funkcję loop() - czyli naszego servera
% Msg : treść wiadomości

client(ServerPid, Msg) ->
    ServerPid ! {req, self(), Msg},
    receive
      {response, ServerPid, Msg} ->
          io:format("Wiadomosc otrzymana z server: ~p ~n",[Msg])
      after 1000 -> {error}
    end.


% ------  funkcja tworząca proces servera  ------

start() ->
```

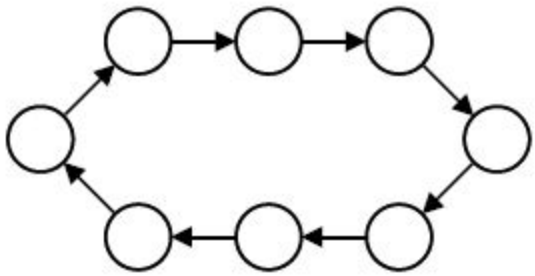
```
    spawn(mod, loop, []).
```

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

```
%---------- funkcja procesu A ----------
% N : licznik przesłanych wiadomości. Z każdą nową wiadomością jest decrementowany, aż do zera.
% PidB :

loopA(0, _PidB) ->
    io:format("proces A koniec ~n"),
    exit(normal);                   % koniec pracy, ta wiadomośc zostanie przesłana do procesu B i on też się zakończy.
loopA(N, PidB) ->
    io:format("A nr ~p ~n",[N]),
    PidB!{msg, self(), N},
    receive
      {response, PidB}->
            loopA(N-1, PidB)
    end.

% ---------- funkcja tworząca proces A ----------
% Chodzi o to, żeby nie wrzucać funkcji link() do funkcji procesu.

makeProcessA(N, PidB) ->
     link(PidB),
     loopA(N, PidB) .
%#########################################################################
#########################################################################


%#########################################################################
#########################################################################
%---------- funkcja procesu B ----------
loopB() ->
    receive
      {msg, From, N} ->
          io:format("B nr ~p ~n",[N]),
          From ! {response, self()},
          loopB();
      {'EXIT', _From, _Reason} ->
          io:format("proces B koniec~n")
    end.
```

```
% ------------ funkcja tworząca proces B ------------
% Chodzi o to, żeby nie wrzucać funkcji process_flag(trap_exit, true) do funkcji procesu.

makeProcessB() ->
    process_flag(trap_exit, true),
    loopB().

%#########################################################################
#############################################################

%---------- Funkcja startowa ----------------------

start(N) ->
   PidB = spawn(mod, makeProcessB, []),
   spawn(mod, makeProcessA, [N, PidB]).
```

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



```
% Opis algorytmu:
%   1. Każdy proces jako parametr posiada: Unikalny numer, ilość razy jaką ma przekazać wiadomość oraz pid sąsiada do któego wysąłć wiadomość.
%   2. Tworzymy listę N procesów. Na tym etapie jako NextPid wpisz dowolną wartość - nie ma ona znaczenia
%   3. Stwórz nową  listę procesów tak aby pid pierwszego procesu na liście stał się ostatnim.
%   4. Mając orginalną listę procesów i listę z punkt 3 możemy każdemu procesowi przesłac PID kolejnego procesu w pierścieniu.
%   5. Rozpocznij przesyłanie wiadomości

-module(mod).
-compile(export_all).


% ------ funkcja startowa  ---------
```

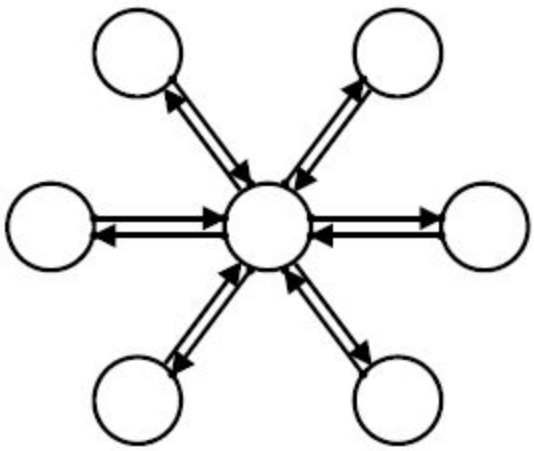
```
% N : ilość procesów w pierścieniu
% M : ilość wiadomości do przesłania w pierscieniu.

start(N, M) ->
   L = [spawn(mod, loop, [X, M, 0]) || X <- lists:seq(1,N)],  % utworzenie listy procesów
składajacych się na pierścień.
   [H|T] = L,
   PidNextList = T ++ [H],
   lists:zipwith(fun(X,Y) -> X!{set, Y} end, L, PidNextList), %  ustawienie parametru NextPid
dla każdego procesu.
   H!{req, tresc}.                                  %  wysłanie wiadomości do pierwszego procesu
w pierścieniu


% ------ funkcja procesu  ---------
%
%   Nr: unikalny numer procesu - w celu rozrównienia ich przy analizie działania programu
%   M : ilość razy jaką proces może przesłac wiadomość. Kiedy M osiąga wartośc 1 przesyła
%       wiadomość dalej i kończy działanie.
%   NextPid : pid sąsiada do którego będą wysyłane wiadomości.

loop(Nr,M, NextPid) ->
   receive
     {set, Pid} ->          % ustawia Pid następnego procesu w pierścieniu.
        loop(Nr, M, Pid);
     {req, Msg} when M > 1->
        io:format("Nr procesu: ~p | Nr wiadomosci: ~p  | Tresc wiadomosci: ~p ~n",[Nr, M,
Msg]),
        NextPid ! {req, Msg},
        loop(Nr, M-1, NextPid);
     {req, Msg} when M == 1->
         io:format("Nr procesu: ~p | Nr wiadomosci: ~p  | Tresc wiadomosci: ~p | KONIEC ~n
",[Nr, M, Msg]),
        NextPid ! {req, Msg},
        exit(normal)
   end.
```

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.

```
-module(mod).
-compile(export_all).
```

```
start() ->
  L = [spawn(mod, loop, [0]) || _ <- lists:seq(1,10)],
  [H|T] = L,
  PidList = T ++ [self()],
  lists:zipwith(fun(X, Y) -> X ! {set, Y} end, L, PidList),
  H ! {req, groundhogDay},
  receive
     {req, Msg} ->
        io:format("----- THE END------- | pid: ~p ~n",[self()])
  end.



loop(NextPid) ->
  receive
     {set, Pid} ->
       io:format("self(): ~p | nextPid() :  ~p ~n",[self(), Pid]),
       loop(Pid);
     {req, Msg} ->
       io:format("pid: ~p   wysyla do pidNext:  ~p ~n",[self(), NextPid]),
       NextPid ! {req, Msg}
  end.
```

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



```
-module(mod).
-compile(export_all).


%---------------------- Funkcja procesu bocznego (jak by płatka gwiazdy :P )-----------------
```

```
loop() ->
   receive
      {req, From, N} ->
         io:format("Pid ~p otrzymal wiadomosc Nr ~p od procesu: ~p ~n",[self(), N, From]),
         From!{response, self(), N},
         loop();
      {stop, From} ->
         io:format("Pid ~p TERMINATE... ~n",[self()]),
         From!{response, self()},
         exit(normal)
   end.


%---------- Funkcja procesu centralnego  ----------

loopCentral() ->
    receive
      {response, From, N} ->
         io:format("<---> | Pid ~p otrzymal wiadomosc Nr ~p od Pid: ~p ~n",[self(), N, From]),
         loopCentral()
  end.


%-------------------- Funkcja startowa ------------------------------

start(N, M) ->
  L = [spawn(mod, loop, []) || _ <-lists:seq(1,N)],
  PidCentral = spawn(mod, loopCentral, []),
  sendMsg(L, M, PidCentral).


%---------- Funkcja pomocniczna sendMsg() ----------------------

sendMsg(L, 0, MainPid) ->
   [X!{stop, MainPid} || X <-L],
   {done};
sendMsg(L,Ile, MainPid) ->
  [X!{req, MainPid, Ile} || X <-L],
  sendMsg(L, Ile -1, MainPid).
```

33. Narysuj zaimplementowaną FST (skończona maszyna stanowa).

```
s1() ->
  receive
  {a} ->
```

```
    s2();
  {b} ->
    s3()
  end.


s2() ->
  receive
  {c} ->
    s3();
  {d} ->
    s4()
  end.


s3() ->
  receive
  {e} ->
    s1();
  {f} ->
    s2()
  end.


s4() ->
  receive
  {g} ->
    s3()
  end.
```

34. Zaimplementuj mechanizm MUTEXu.

```
blokuj() ->
   mutex ! {blokuj, self()},   % wysłanie żądania do mutexu
   receive                     % do momentu aż nie nadejdzie wiadomośc od mutexu proces
będzie blokowany w klauzuli receive ... end.
      {mutex, mozna_wchodz} ->
         wchodze
   end.                        % tutaj następuje wejście do współdzielonego zasobu



%  odblokuj() : jeżeli proces skończył operacje w sekcji krytycznej to powinien zwolnić
mutexa.
```

% w tym celu wywołuje metodę odblokuj(). Proces Mutexa może odblokować tylko proces który do blokował!

```
odblokuj() ->
  mutex ! {odblokuj, self()}.
```

```
%  Mutex będzie implementowany jako maszyna 2 stanowa.
%  Stany implementowane będą jako funkcje.
%  Uwaga: na proces może składać się wiele funkcji, ale każdy proces ma tylko jedną skrzynkę odbiorczą!

wolny() ->
  receive
     {blokuj, Pid} ->
        Pid!{mutex, mozna_wchodz},
        zajety(Pid);
     {stop} ->  % zatrzymanie procesu mutexa
        true
  end.

zajety(Pid) ->
  receive
    {odblokuj, Pid} ->
      wolny()
  end.
```

3.3 Pytania inne

1. Zaimplementuj zegar wektorowy, vector clock.

- ***Początkowo wszystkie zegary ustawione są na 0***
- ***proces zwiększa swój zegar przed wystąpieniem każdego swojego zdarzenia***
- ***każdorazowo, przy wysyłaniu wiadomości proces zwiększa swój zegar o jeden i przesyła cały swój wektor wraz z wiadomością***
- ***każdorazowo, przy odbiorze wiadomości proces zwiększa swój zegar o jeden i aktualizuje wszystkie elementy w swojej kopii zegara biorąc maksimum z wartości lokalnej oraz z otrzymanego wektora***

```
-module(mod).
-compile(export_all).

loop(Numer, ClockList, ProcessList) ->
```

```
    NewClockList1 = internatEvent(Numer, ClockList),
    NewClockList2 = incListaZegarow(NewClockList1, Numer),
    losujProces(ProcessList) ! {msg, NewClockList2},
    receive
        {msg, MsgList} ->
            NewClockList3 = func(NewClockList2, MsgList),
            timer:sleep(2000),
            io:format("Proces nr: ~p  pid: ~p | innerClock: ~w  | receivedClock: ~w |
combinedClock: ~w ~n",[Numer, self(), NewClockList2, MsgList, NewClockList3]),
            loop(Numer, NewClockList3, ProcessList)
    end.

makeLoop(Numer, ClockList) ->
   receive
      {set, Lista} ->
            NewProcessList = Lista
   end,
   loop(Numer, ClockList, NewProcessList).

internatEvent(Numer,ClockList) ->
    case random:uniform(100) rem 30 == 0 of
        true ->
            io:format("wystapil 'inner event' w procesie pid: ~p ~n",[self()]),
            incListaZegarow(ClockList, Numer);  % wysąpił internal event, zwiększ zegar o 1.
        false -> ClockList
    end.

incListaZegarow(ListaZegarow, Index) ->
    InkrementowanyZegar = lists:nth(Index, ListaZegarow),
    Temp = InkrementowanyZegar + 1,
    lists:sublist(ListaZegarow, Index-1) ++ [Temp] ++ lists:sublist(ListaZegarow, Index+1,
length(ListaZegarow)).

func([], []) -> [];
func([H1|T1], [H2|T2]) ->
   [max(H1,H2) | func(T1, T2)].

losujProces(List) ->
    lists:nth(random:uniform(length(List)),List).

start(N) ->
   ClockList = [0 || _<-lists:seq(1,N)],
   ListaProcesow = [spawn(mod, makeLoop, [X, ClockList]) || X <- lists:seq(1,N)],
   [X ! {set, ListaProcesow} || X <-ListaProcesow],
   {koniec_funkcji_startowej}.
```