Współprogramy

Andrzej Salwicki
2 październik 2010

Współprogramy I

Plan:

- Motywacja
- Składnia
- Scenariusz obiektu współprogramu
- Przykłady
  - Producent – konsument (instrukcja attach)
  - Czytelnik -pisarze (instukcja detach)
  - Łączenie drzew binarnych poszukiwań (pojęcie łańcucha dynamicznego)

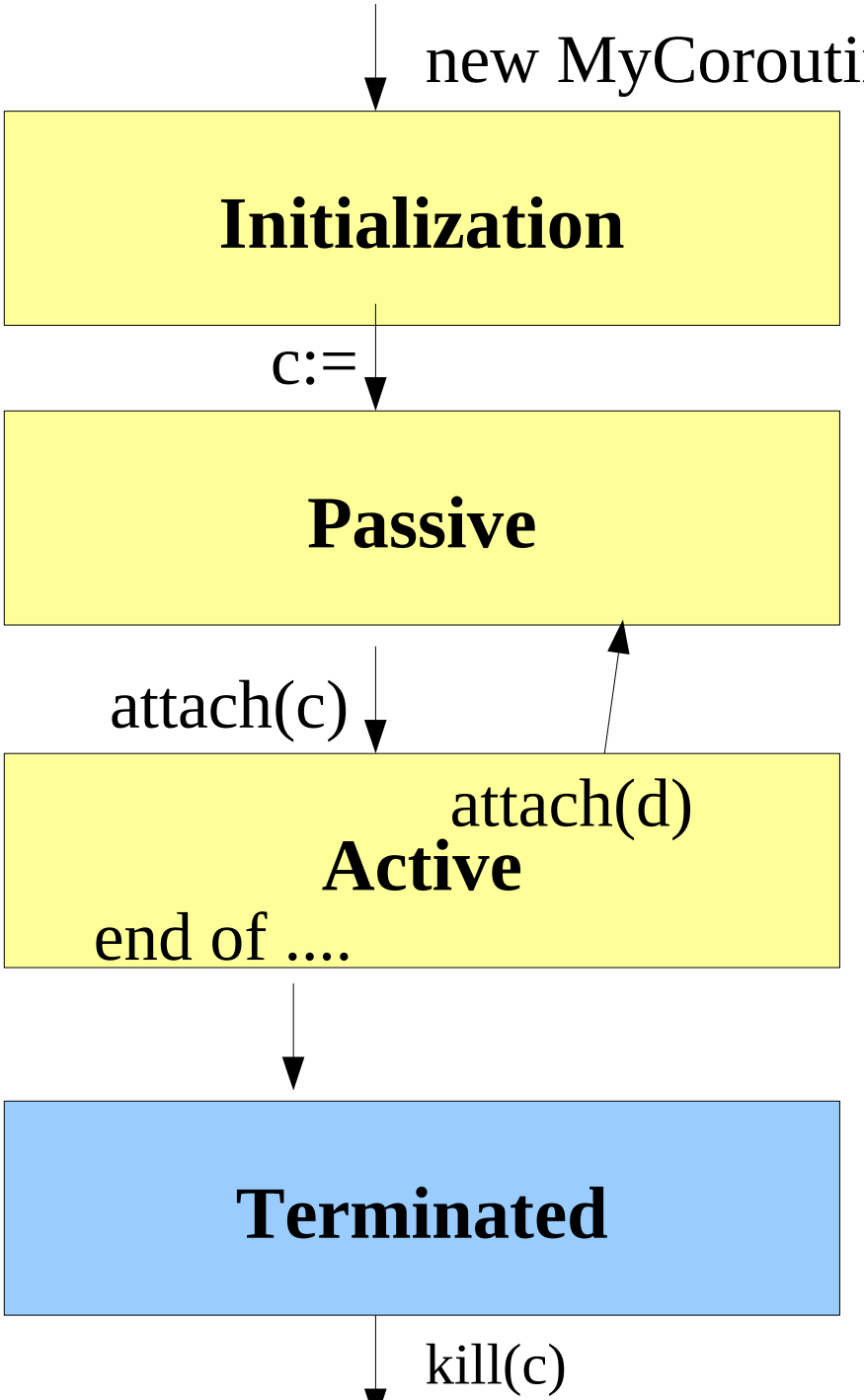
Składnia

unit <*nazwa*>: {<prefix>} **coroutine**(<*parametry formalne*>);
<*deklaracje lokalne: zmiennych, metod, klas i in.*>
begin
<*instrukcje inicjalizacji*>**;**
return;
<*instrukcje współprogramu*>
end <*nazwa*>;

W instrukcjach współprogramu można używać instrukcji przełączania: **attach**(*obCoroutiny*) i **detach**.

Semantyka – scenariusz

Semantyka – scenariusz współprogramu



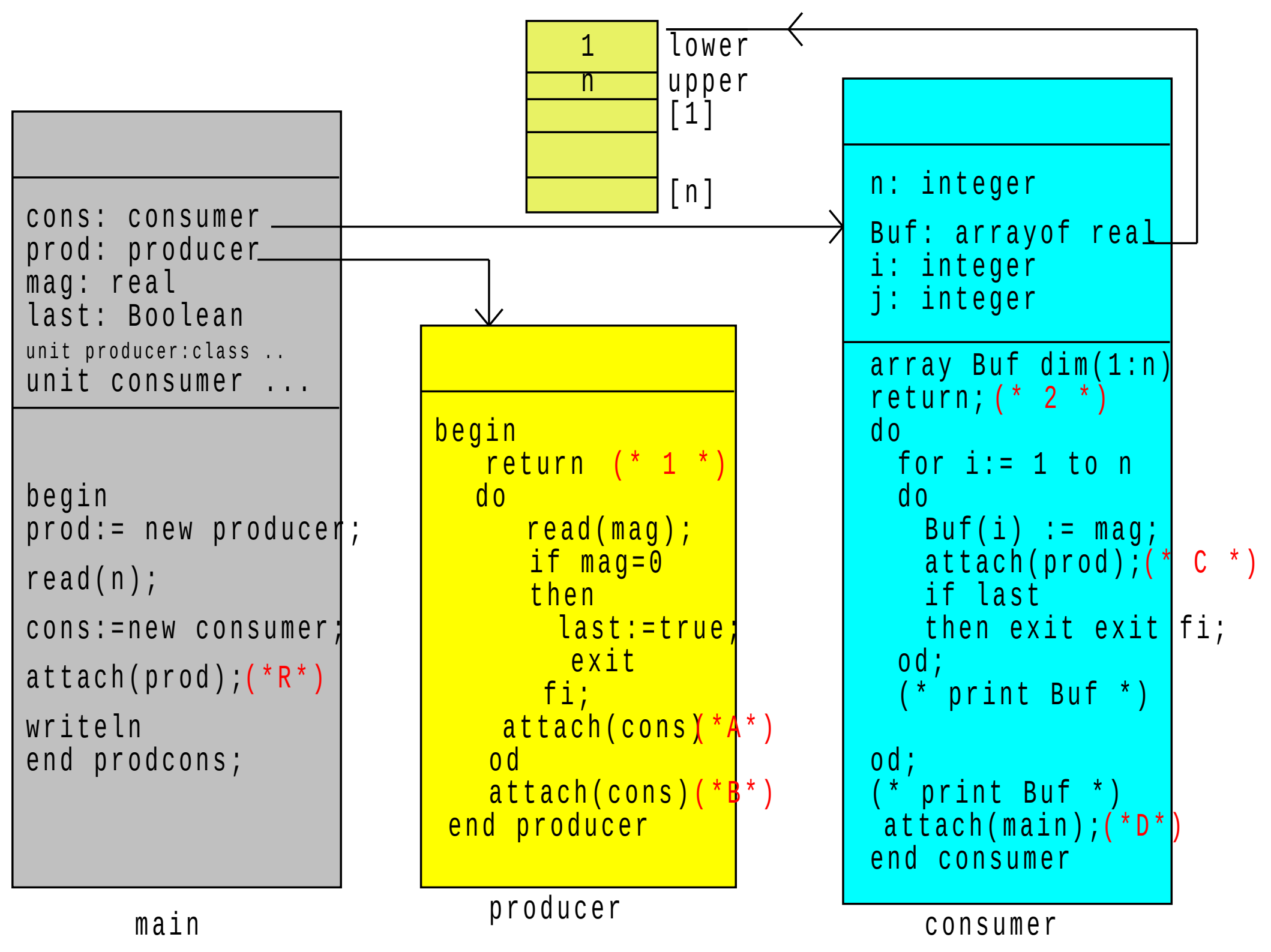
Axiom of coroutines

card(Active)=1

```
program prodcons;
  var prod:producer,cons:consumer,n:integer,mag:real,last:bool;

  unit producer: coroutine; ...
  end producer;

  unit consumer: coroutine(n:integer); ...
  end consumer;

 begin    (*  MAIN program  *)
    prod:=new producer;
    read(n);
    cons:=new consumer(n);
    attach(prod);    (* R *)
    writeln;
 end prodcons;
```

```
unit producer: coroutine;
begin
  return;   (* 1 *)
  do
    read(mag);       (* mag-nonlocal variable, common store*)
    if mag=0
    then             (* end of data *)
      last:=true;
      exit
    fi;
    attach(cons);   (* A *)
  od;
  attach(cons)       (* B *)
end producer;
```

```
unit consumer: coroutine(n:integer);
var Buf:arrayof real; var i,j:integer;
begin
  new_array Buf dim(1:n);
  return;    (* 2*)
  do
    for i:=1 to n
    do
      Buf(i):=mag;
      attach(prod);    (* C *)
      if last then exit exit fi;
    od;
    for i:=1 to n
    do     (* print Buf *)
      write(' ',Buf(i):10:2)
    od;
    writeln;
  od;
  for j:=1 to i do write(' ',Buf(j):10:2) od;  writeln;      (* print the rest of Buf *)
  attach(main);      (* D *)
end consumer;
```

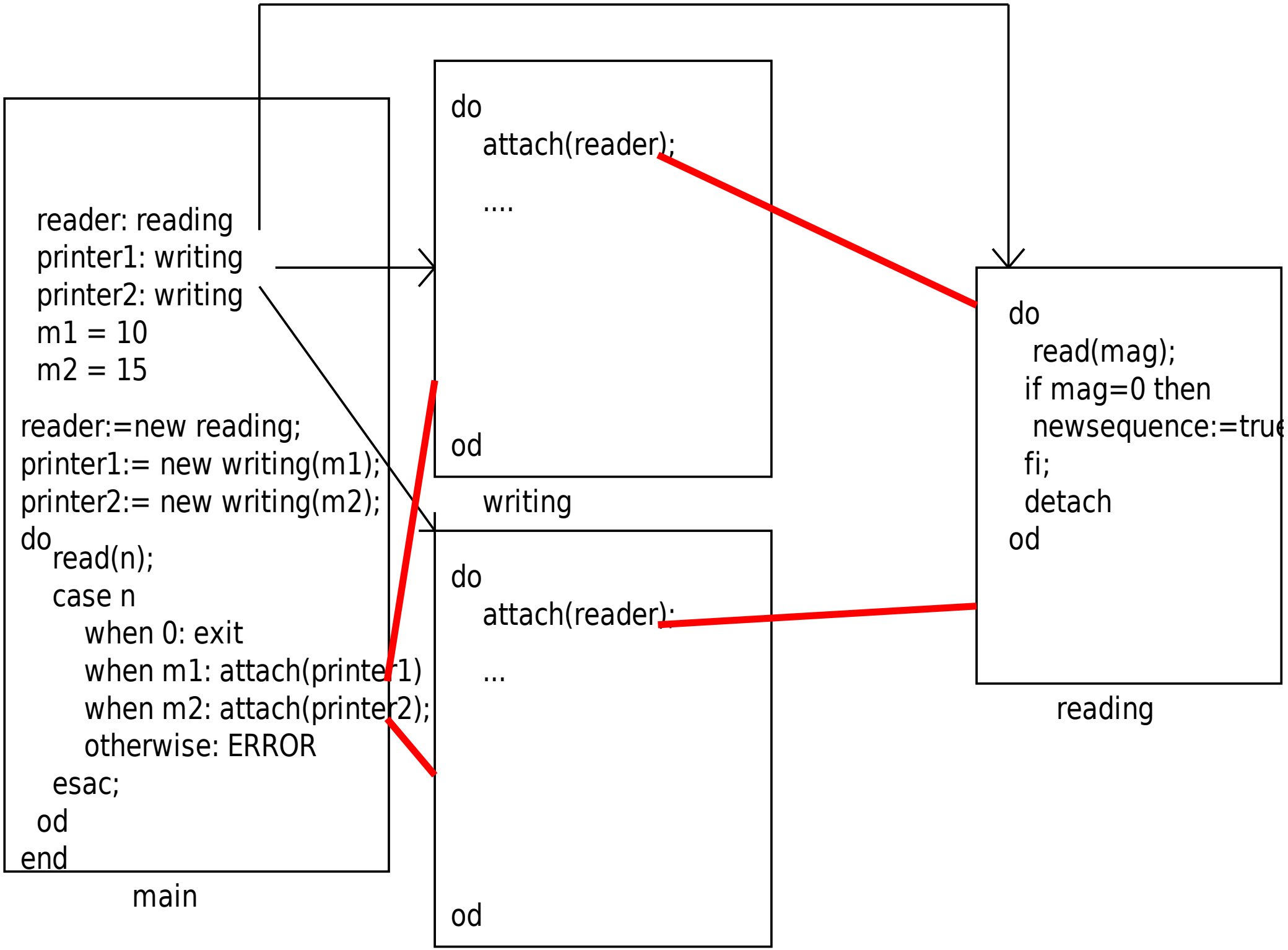

1
n
lower
upper
[1]
[n]
cons: consumer
prod: producer
mag: real
last: Boolean
unit producer:class ..
unit consumer ...
begin
prod:= new producer;
read(n);
cons:=new consumer;
attach(prod);(*R*)
writeln
end prodcons;
main
begin
return (* 1 *)
do
read(mag);
if mag=0
then
last:=true;
exit
fi;
attach(cons)(*A*)
od
attach(cons)(*B*)
end producer
producer
n: integer
Buf: arrayof real
i: integer
j: integer
array Buf dim(1:n)
return;(* 2 *)
do
for i:= 1 to n
do
Buf(i) := mag;
attach(prod);(* C *)
if last
then exit exit fi;
od;
(* print Buf *)
od;
(* print Buf *)
attach(main);(*D*)
end consumer
consumer

```
program reader_writers;
   (* In this example a single input stream consisting of blocks of numbers,
     each ending with 0, is printed on two printers of different width.
    The choice of the printer is determined by the block header
    which indicates the desired number of print columns.
   The input stream ends with a double 0.  m1 = the width of printer_1,
    m2 = the width of printer_2 *)

   end;
```

```
program reader_writers;
   const m1=10,m2=20;    var reader:reading,printer_1,printer_2:writing;
   var n:integer,new_sequence:boolean,mag:real;
  unit writing:coroutine(n:integer);   ...
  end writing;
  unit reading:coroutine ;    ...
  end reading;
  begin
   reader:=new reading;
   printer_1:=new writing(m1);     printer_2:=new writing(m2);
   do
    read(n);
    case n
     when 0:  exit
     when m1: attach(printer_1)
     when m2: attach(printer_2)
     otherwise  write(" wrong data"); exit
    esac
   od
  end;
```

```
unit writing:coroutine(n:integer);
var Buf: arrayof real, i,j:integer;
begin
  new_array Buf dim (1:n);      (* array  generation *)
  return;          (* return terminates coroutine  initialization *)
  do
    attach(reader);       (* reactivates coroutine reader *)
    if new_sequence
    then
     (* a new sequence causes buffer Buf to be cleared up *)
      for j:=1 to i do write(' ',Buf(j):10:2) od;  writeln;
      i:=0; new_sequence:=false;  attach(main)
    else
      i:=i+1;   Buf(i):=mag;
      if i=n
      then
        for j:=1 to n do write(' ',Buf(j):10:2) od;    writeln;     i:=0;
      fi
    fi
  od
end writing;
```

```
unit reading: coroutine;
  begin
    return;
    do
      read(mag);
      if mag=0  then  new_sequence:=true;   fi;
      detach;
      (* detach returns control to printer_1 or  printer_2
         depending which one reactivated reader *)
    od
  end reading;
```


reader: reading
printer1: writing
printer2: writing
m1 = 10
m2 = 15
reader:=new reading;
printer1:= new writing(m1);
printer2:= new writing(m2);
do
read(n);
case n
when 0: exit
when m1: attach(printer1)
when m2: attach(printer2);
otherwise: ERROR
esac;
od
end
main
do
attach(reader);
....
od
writing
do
attach(reader);
...
od
do
read(mag);
if mag=0 then
newsequence:=true
fi;
detach
od
reading

Zadania

- Napisz klasę **coroutine** w Javie, jako rozszerzenie klasy threads.
- Napisz klasę **coroutine** w C++.
- Napiszcie z kolegą grę strategiczną (tzn. nie strzelankę, nie karcianą) np. *“kółko i krzyżyk”*, *warcaby*,

Współprogramy II

W tym wykładzie pogłębimy naszą znajomość z współprogramami. Omówimy współpracę procedur rekurencyjnych i współprogramów, wprowadzimy pojęcie łańcucha dynamicznego.

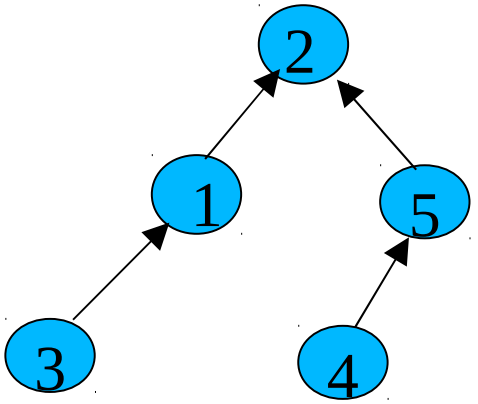
Przykład

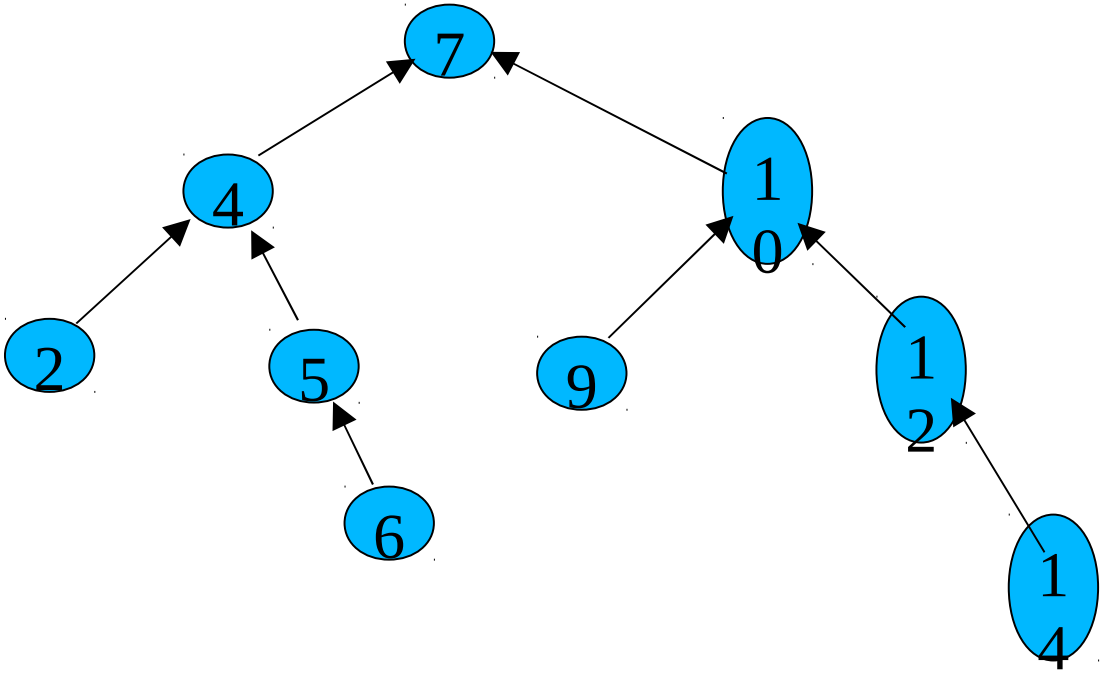
- Rozpatrzmy następujące zadanie:
  Dany jest zbiór drzew binarnych poszukiwań. Należy scalić dane zapisane w tym zbiorze i wydrukować je w porządku rosnącym.
- Przypomnijmy sobie najpierw: czym są drzewa binarnych poszukiwań?

Przykład

- Rozpatrzmy następujące zadanie:
  Dany jest zbiór drzew binarnych poszukiwań. Należy scalić dane zapisane w tym zbiorze i wydrukować je w porządku rosnącym.
- Przypomnijmy sobie najpierw: czym są drzewa binarnych poszukiwań BST?
- **Definicja.** Drzewem binarnych poszukiwań (BST) jest
  - drzewo binarne, skończone, zawierające elementy z pewnego zbioru E, uporządkowanego przez relację *less* (<), i takie, że,
  - dla każdego poddrzewa *pd* dowolnego drzewa $d \in$ BST ma miejsce następująca własność: *każdy element zapisany w lewym poddrzewie drzewa pd jest niewiększy od elementu zapisanego w korzeniu drzewa pd, a każdy element zapisany w prawym pod-drzewie drzewa pd jest niemniejszy od elementu z korzenia.*

czy to jest drzewo BST?





Zadanie: scalić drzewa BST w ciąg

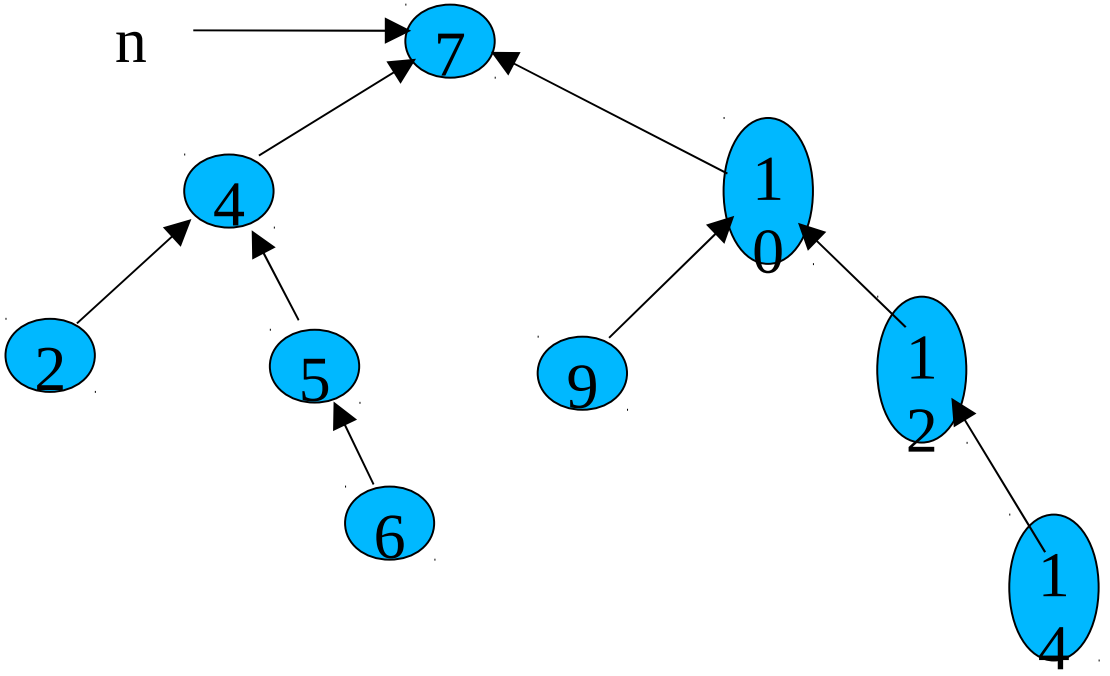
- Rozpatrzmy następujące zadanie:
  Dany jest zbiór drzew binarnych poszukiwań. Należy scalić dane zapisane w tym zbiorze i wydrukować je w porządku rosnącym.

Zadanie: scalić drzewa BST w ciąg

- Rozpatrzmy następujące zadanie:
  Dany jest zbiór drzew binarnych poszukiwań. Należy scalić dane zapisane w tym zbiorze i wydrukować je w porządku rosnącym.
- Niech *node* będzie klasą

```
unit node: class(val: integer);  (* klucz poszukiwań*)
   var  left, right: node;
   unit  insert: procedure(e: integer);  ...     end insert;
   unit T: procedure(y: node);
   begin
      if y<>none  then
         call T(y.left);  write(y.val); call T(y.right)
      fi
   end T;
end node;
```

rozpatrzmy drzewo

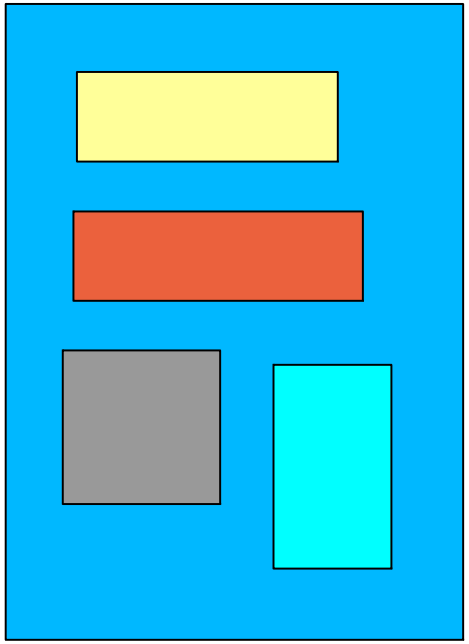


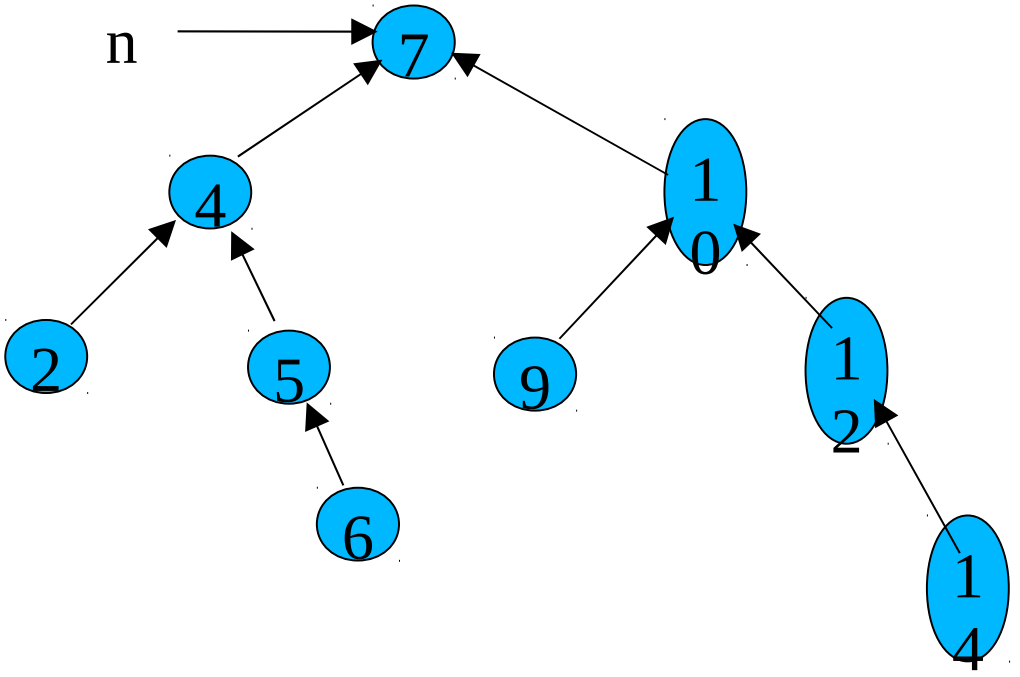
przykład c.d.

- Co się stanie gdy zmienna n typu node `var n: node` wskazuje na korzeń drzewa binarnych poszukiwań i gdy wykonamy instrukcję?

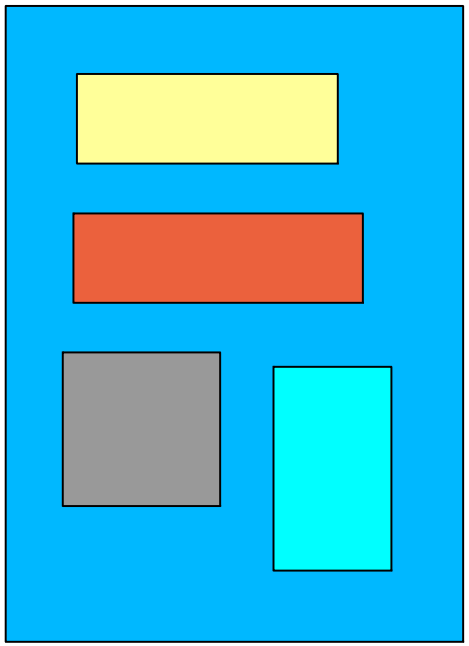
```
call n.T(n)
```

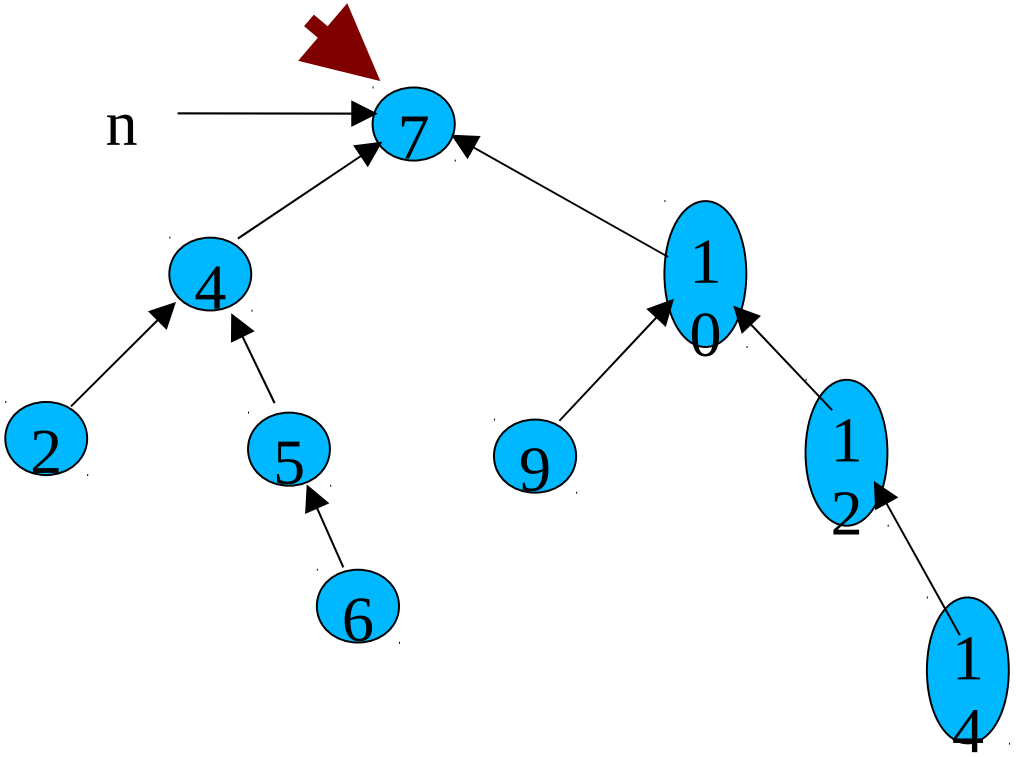
- Tak, masz rację. Zostanie wydrukowany ciąg wartości val zapisanych w wierzchołkach drzewa o korzeniu n, liczby te będą wypisane od najmniejszej do największej. Dlaczego? Udowodnij odpowiedni lemat.



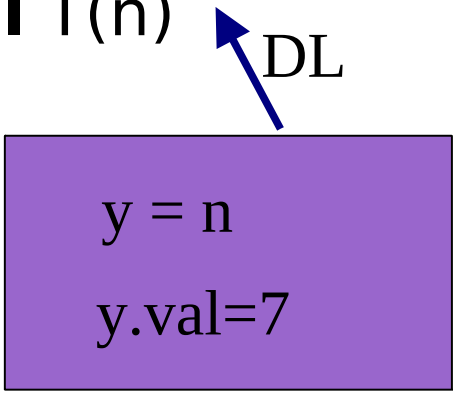


jednostki dynamiczne
utworzone dotychczas

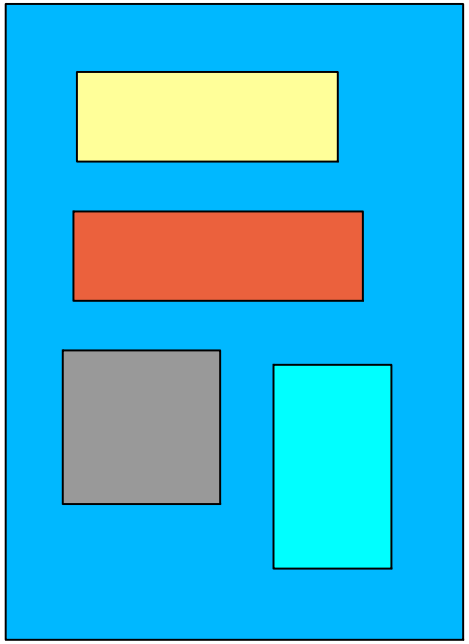


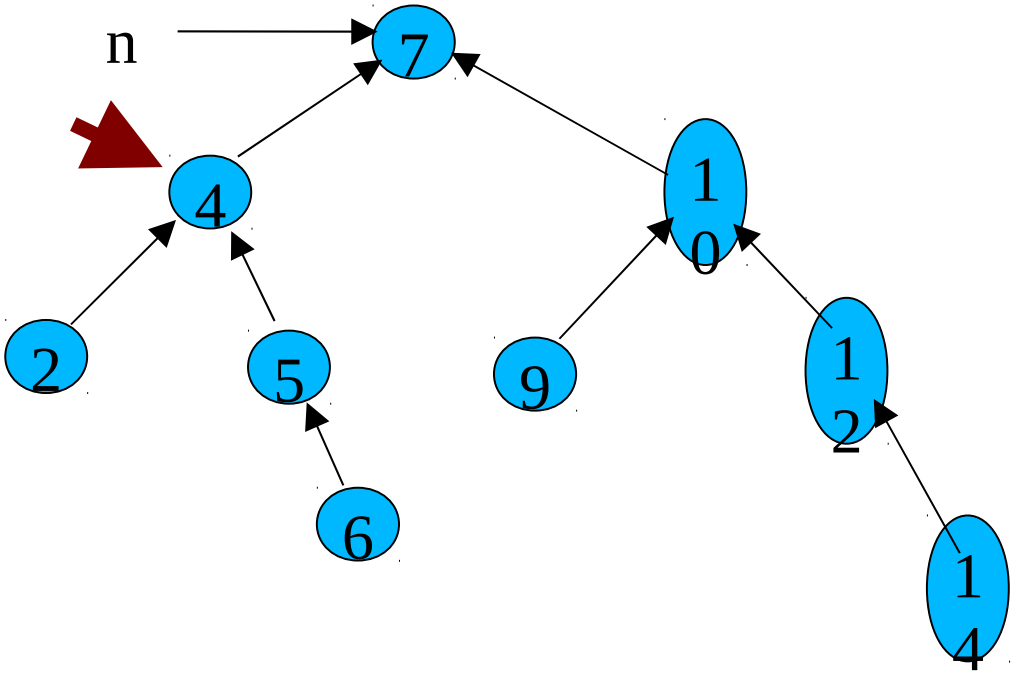


wykonujemy instr.
call T(n)

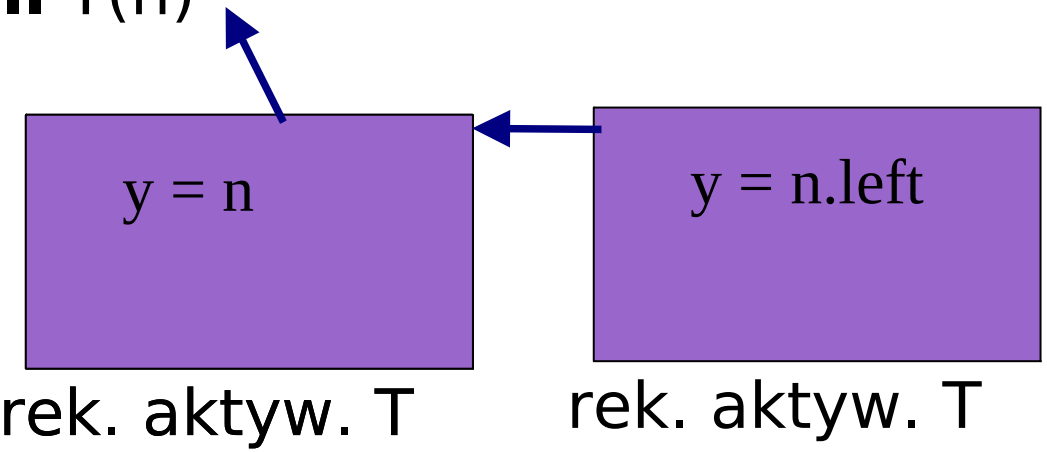


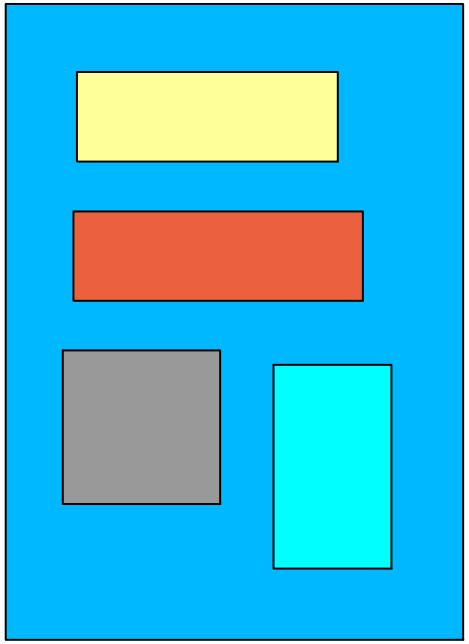
rek. aktyw. T

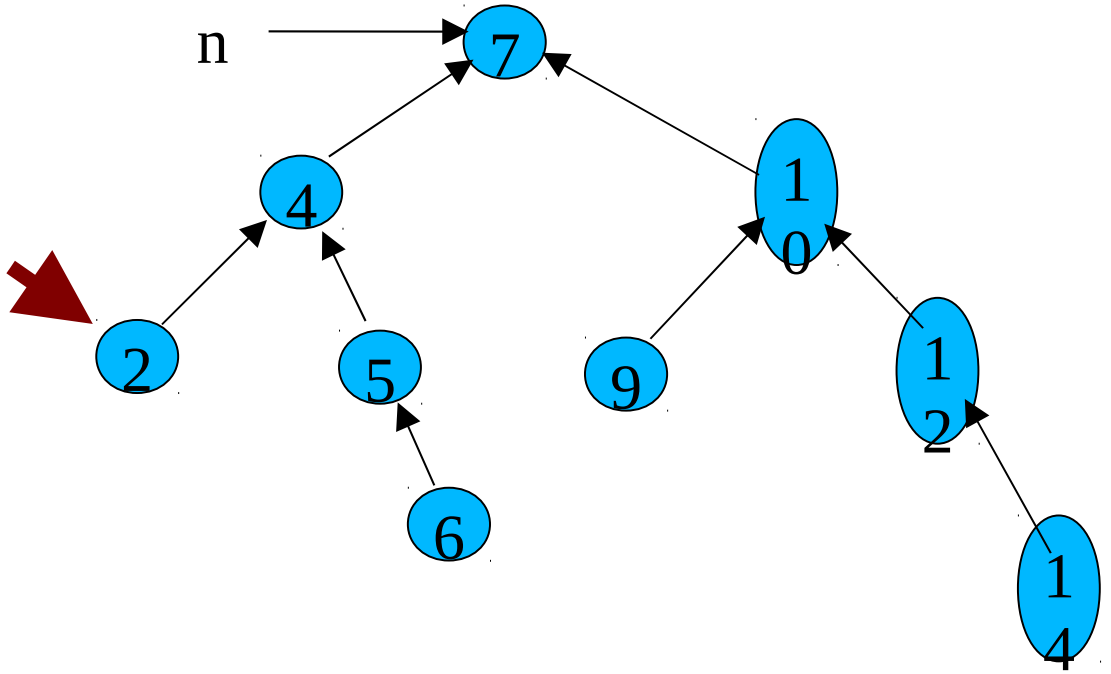




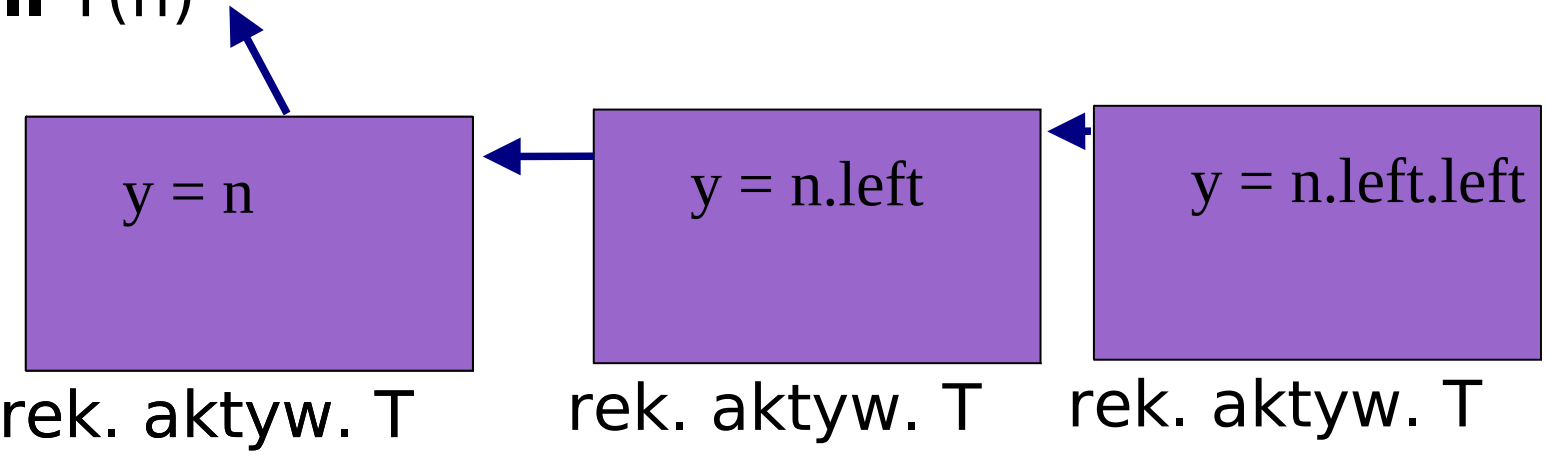
wykonujemy instr.
call T(n)

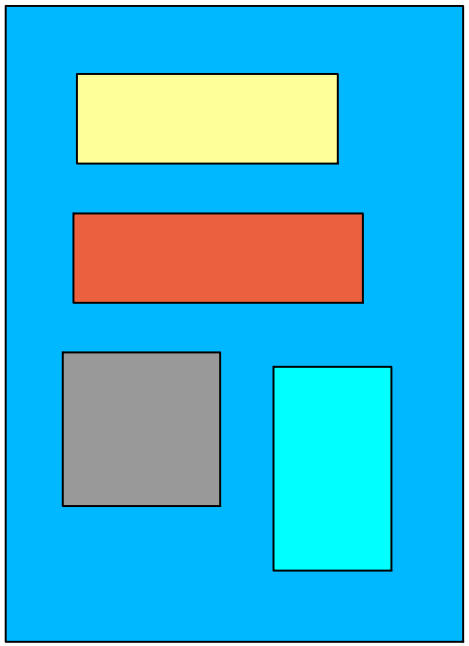


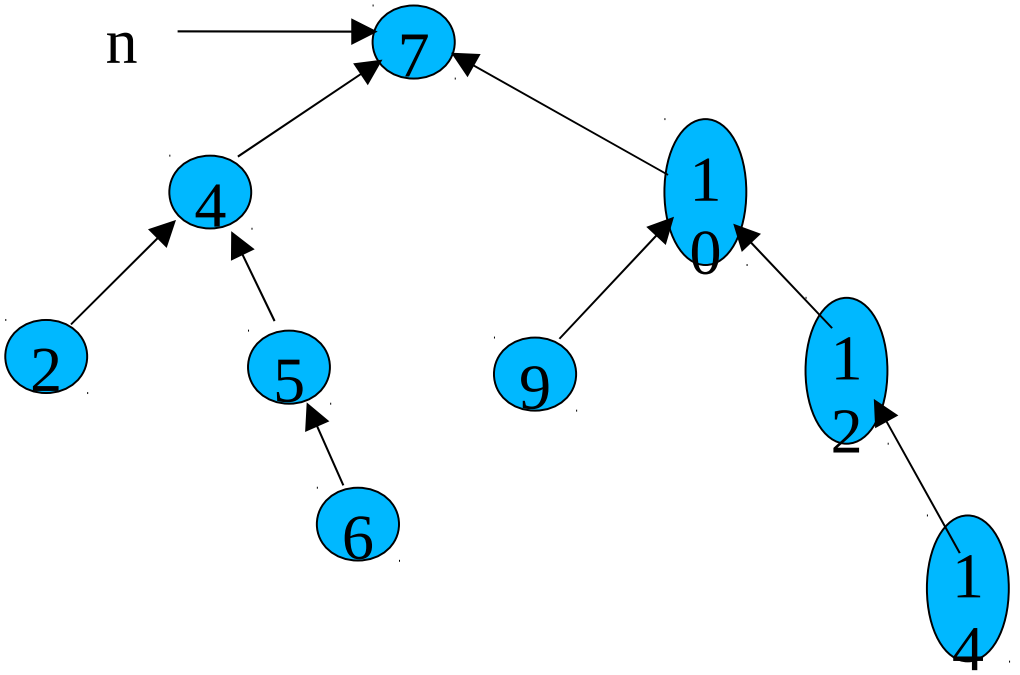




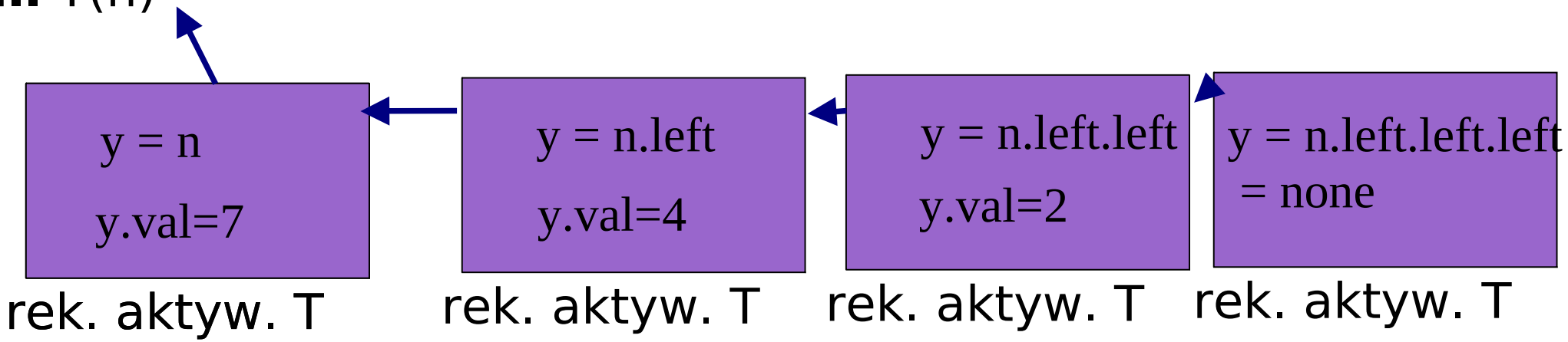
wykonujemy instr.
call T(n)

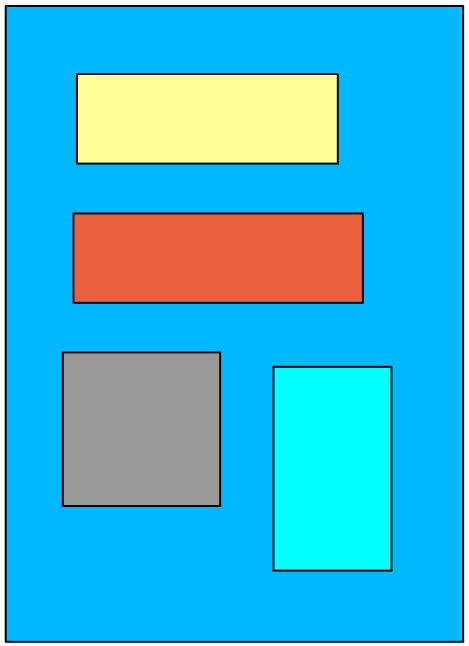


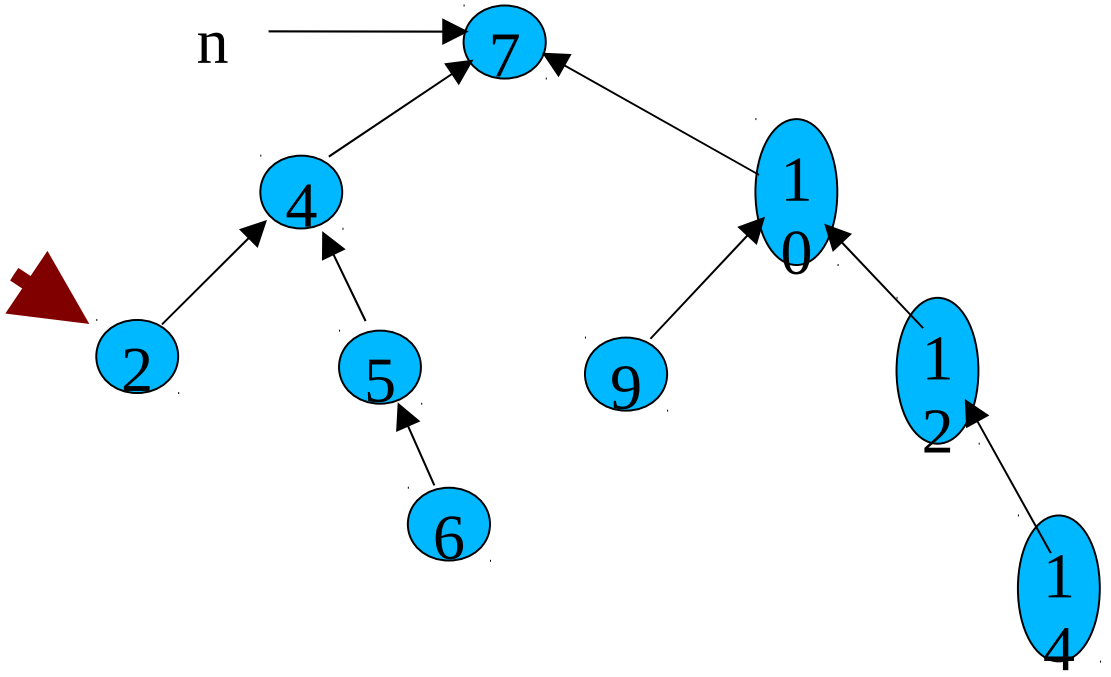




wykonujemy instr.
call T(n)

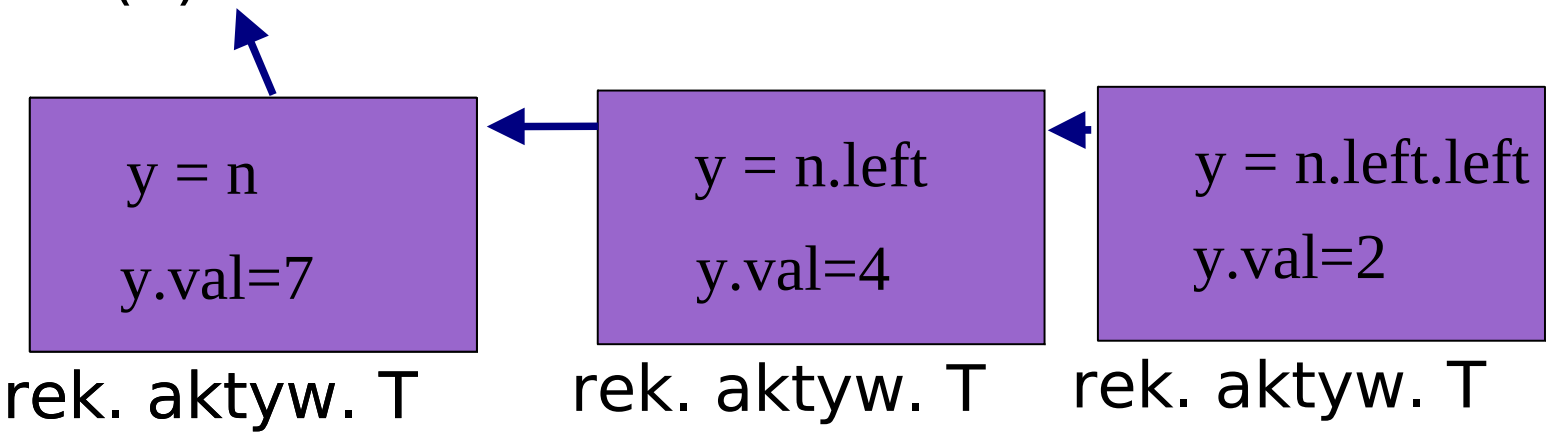


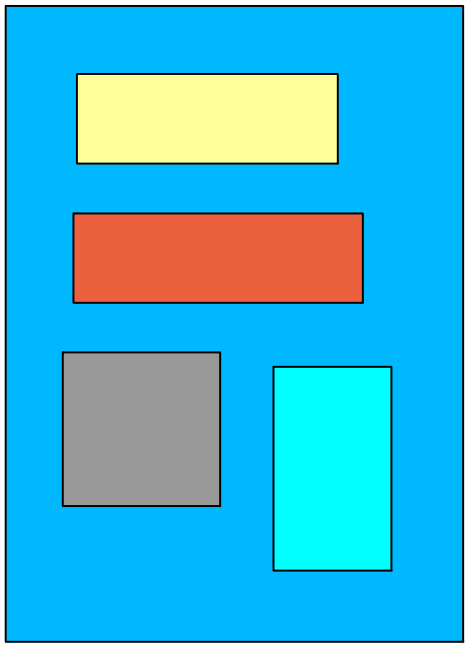


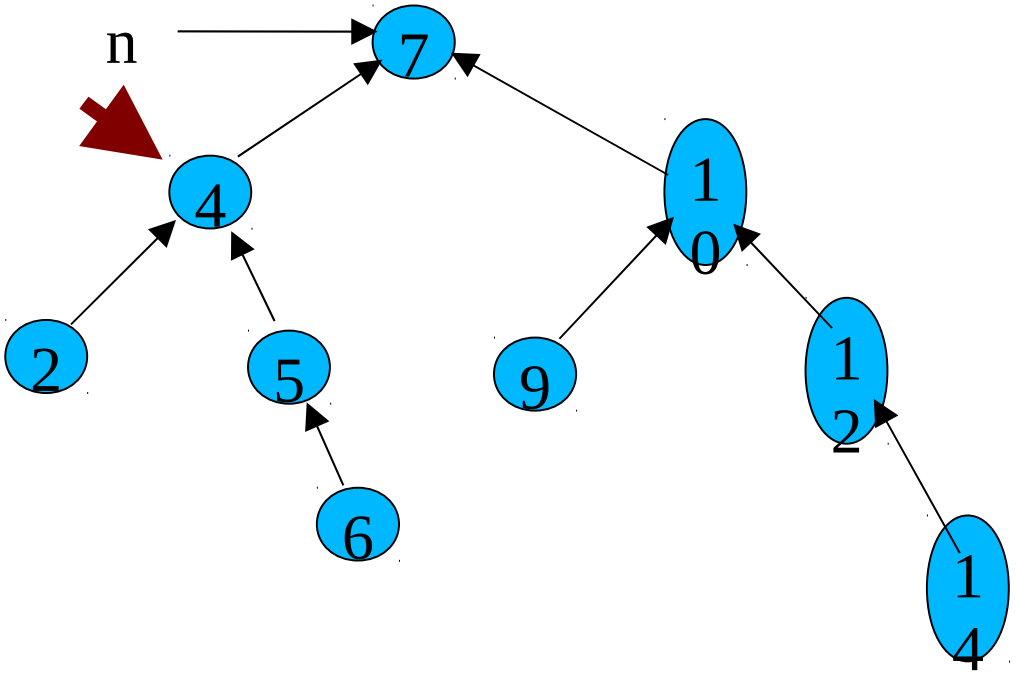


wykonujemy instr.
call T(n)

write(y.val) 2



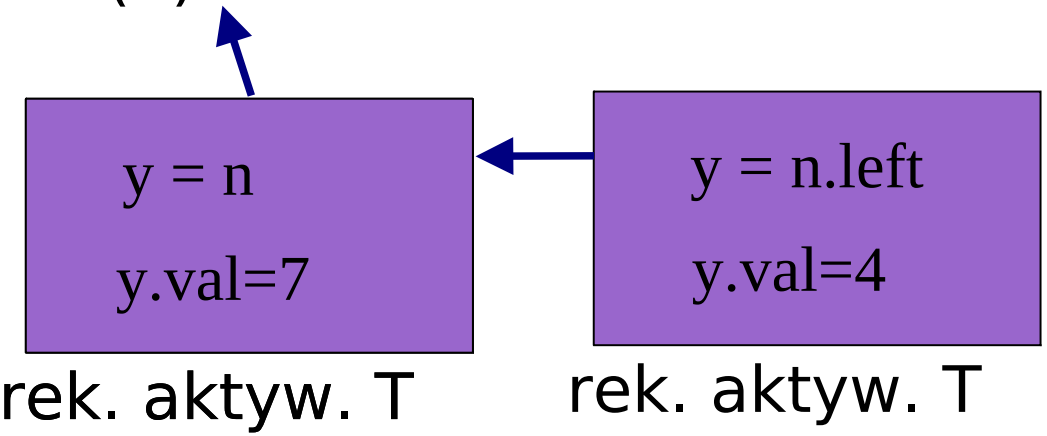


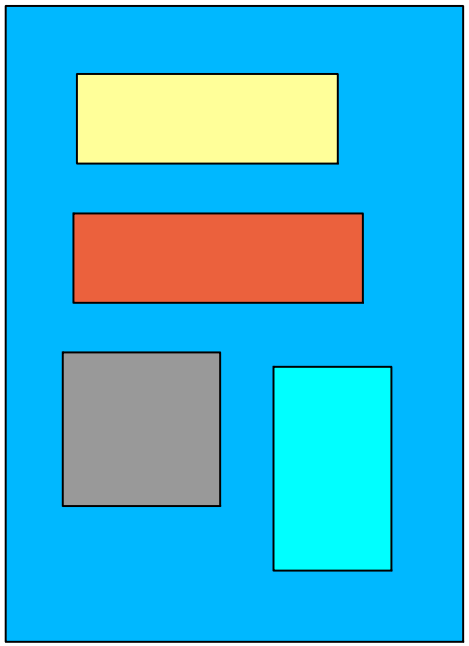


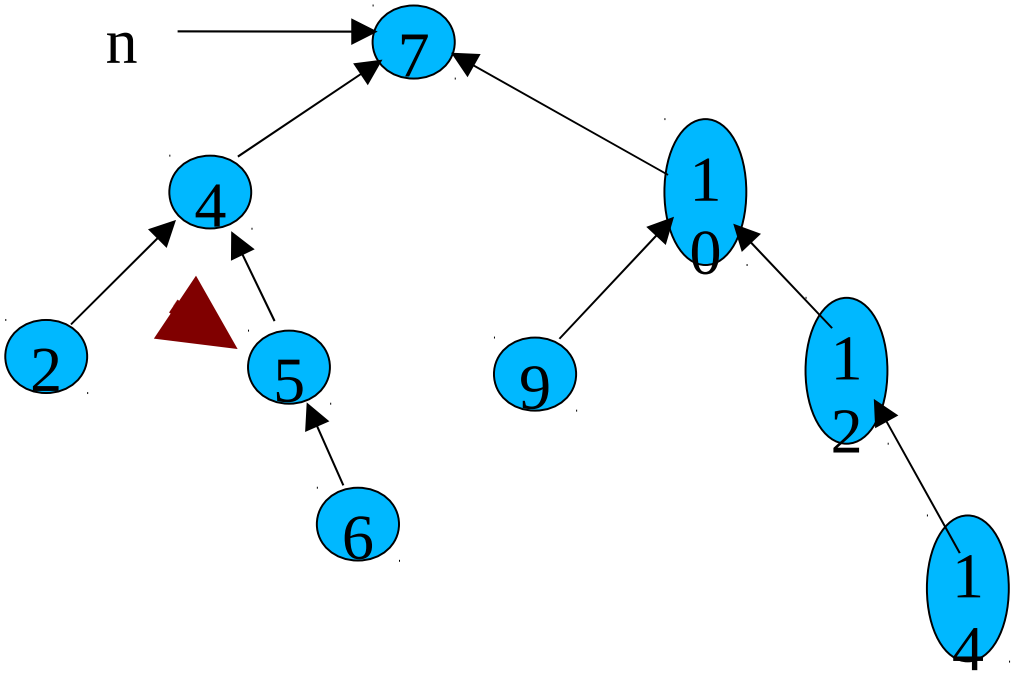
wykonujemy instr.
call T(n)

write(y.val) 2

write(y.val) 4







wykonujemy instr.
call T(n)

write(y.val) 2

write(y.val) 4

y = n
y.val=7

y = n.left
y.val=4

y = n.left.right
y.val=5

rek. aktyw. T

rek. aktyw. T

rek. aktyw. T

Lemat
Wykonanie instrukcji **call** T(n) gdzie n wskazuje na korzeń drzewa BST spowoduje wydrukowanie wszystkich elementów zapisaych w drzewie w kolejności od najmniejszego do największego elementu
Dowód.
(*przez indukcję ze względu na wysokość drzewa* **)**
Dla drzew o wysokości jeden, lemat zachodzi, widać to z treści procedury T.
Załóżmy więc, że lemat zachodzi dla wszystkich drzew BST o wysokości nie większej niż *k*.
Rozpatrzmy drzewo o wysokości *k+1*. Z treści procedury T wynika, że najpierw wykonana zostanie instrukcja **call** T(n.left). To poddrzewo ma wyskość nie większą od *k*. Z założenia indukcyjnego wydrukowane zostaną elementy zapisane w tym poddrzewie w kolejności rosnącej.
Potem wykonana zostanie instrukcja write(n.val), wydrukowana zostanie wartość większa od dotychczas wydrukowanych, bo to jest drzewo BST.
Z tego samego powodu i z założenia indukcyjnego instrukcja **call** T(n.right) spowoduje wydrukowanie ciągu rosnącego elementów zawartych w prawym poddrzewie. Wszystkie one są większe od wartości n.val.

A teraz wyobraźmy sobie dwa współprogramy A i B: w jednym z nich, w A, wywoływana jest procedura T **`call T(n)`**, a w drugim powtarzamy instrukcje **attach**(A).

```
program test;
   var A: CA, B: CB, kolejny, n: integer;
```

```
unit CA: coroutine(n: node);
  unit T: procedure(y: node);
  begin
    if y<>none  then
       call T(y.left);
       kolejny := y.val; detach;
      call T(y.right)
    fi
  end T;
begin return;call T(n);
end CA;
```

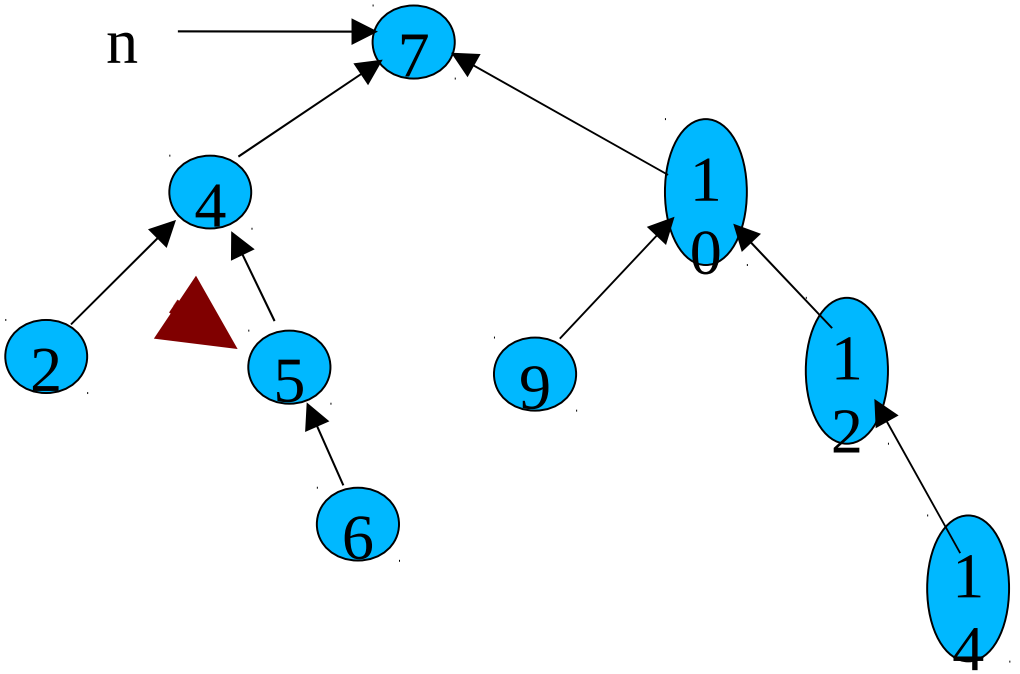
```
unit CB: coroutine;
begin
    return;
   do
      write(kolejny);
      attach(A)
    od
end CB
```

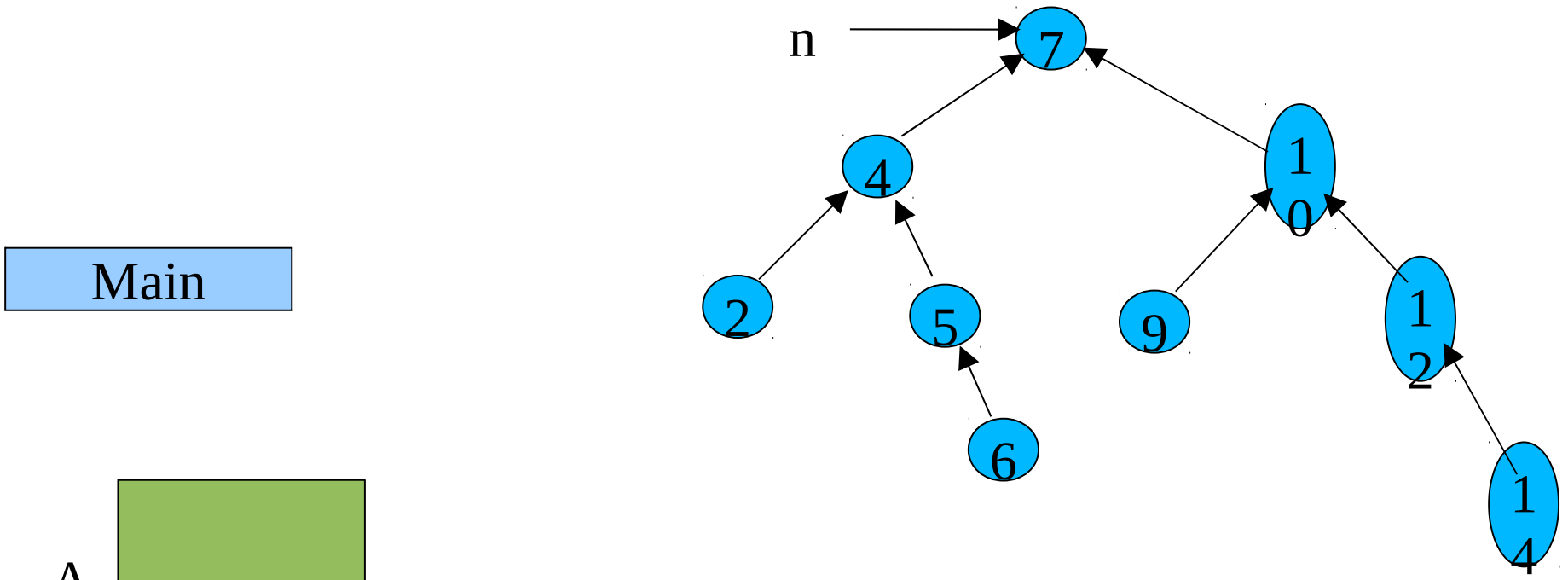
```
begin
    A:= new CA; B:= new CB; attach(B)
end test
```

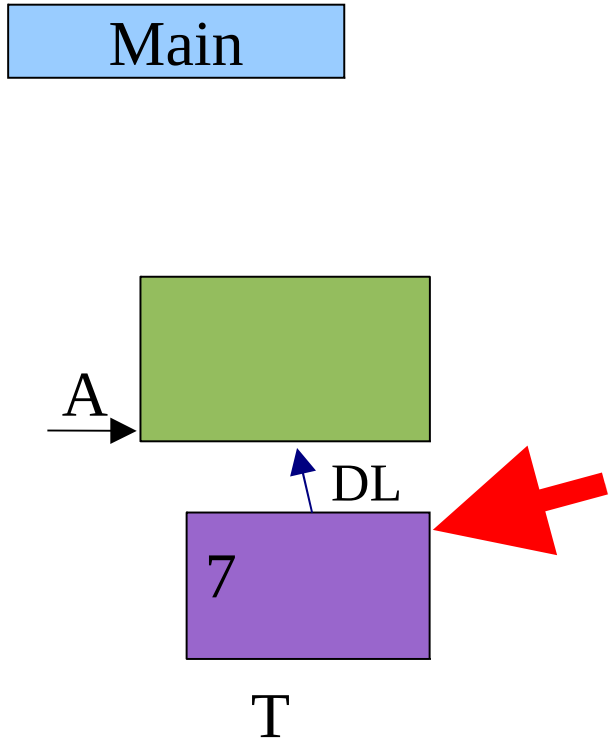
Na tym rysunku narysować: rekord aktywacji Main, drzewo BST, obiekty współprogramów A i B, rekordy aktywacji procedury T

Main

Main

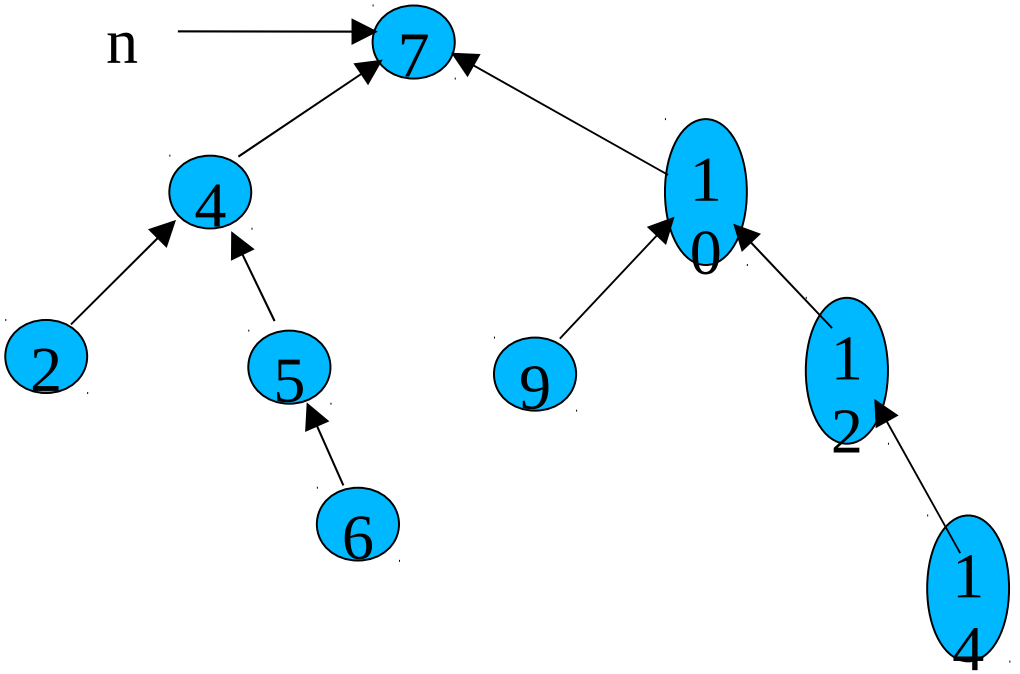


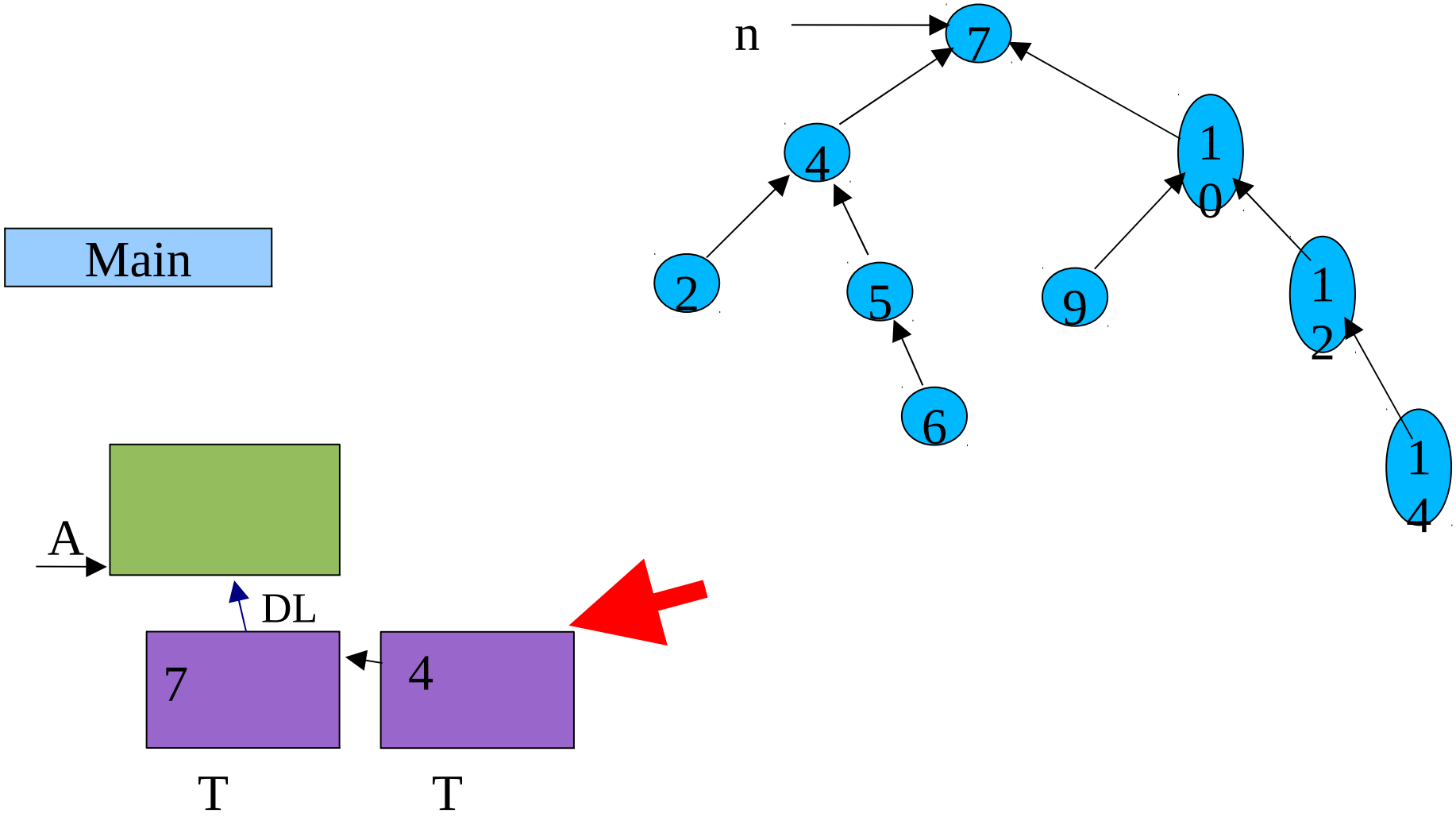

n
7
4
10
2
5
9
12
6
14
Main
A
B
do
write(kolejny);
attach(A)
od

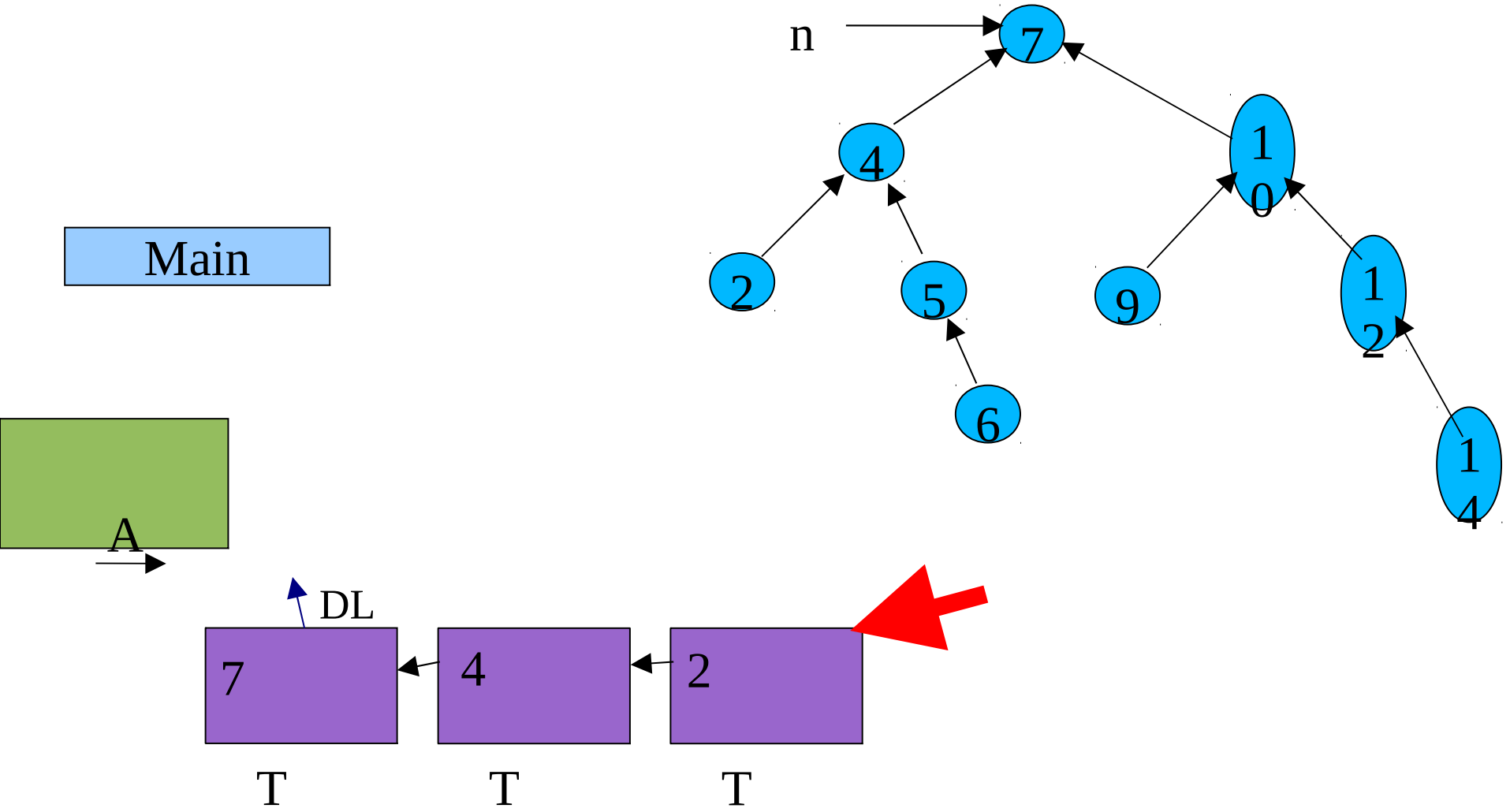


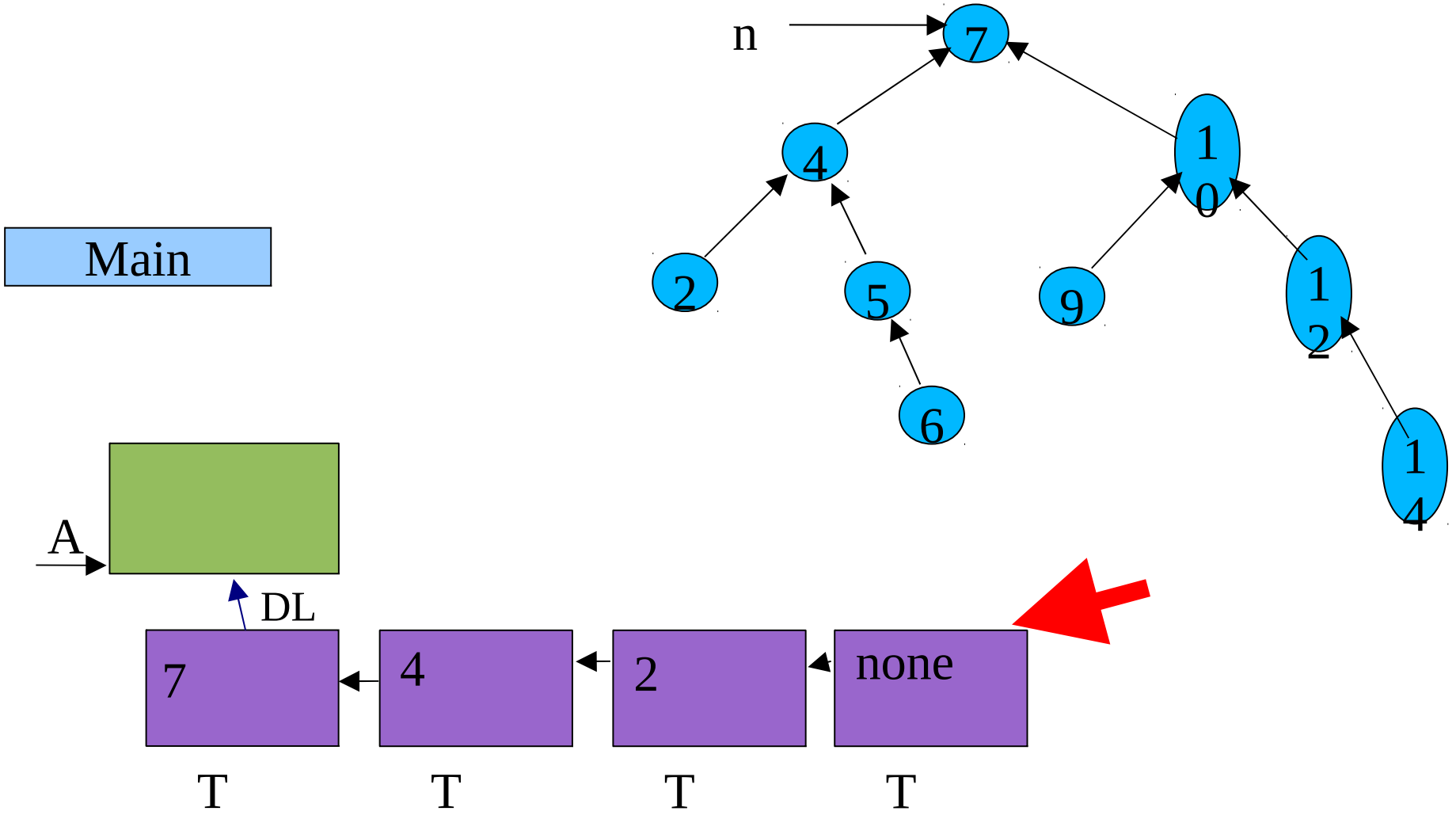
B

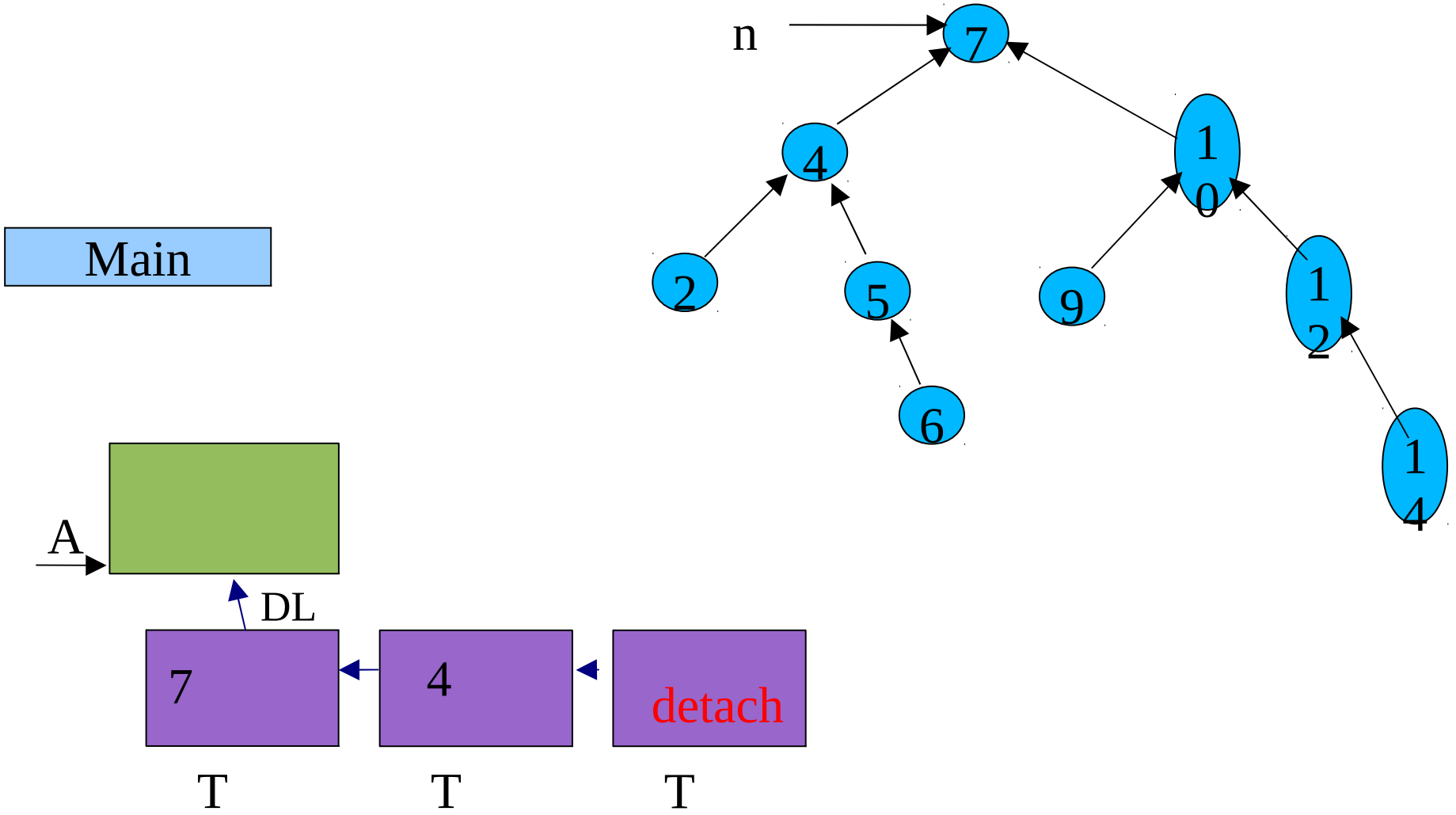
```
do
  write(kolejny);
  attach(A)
od
```

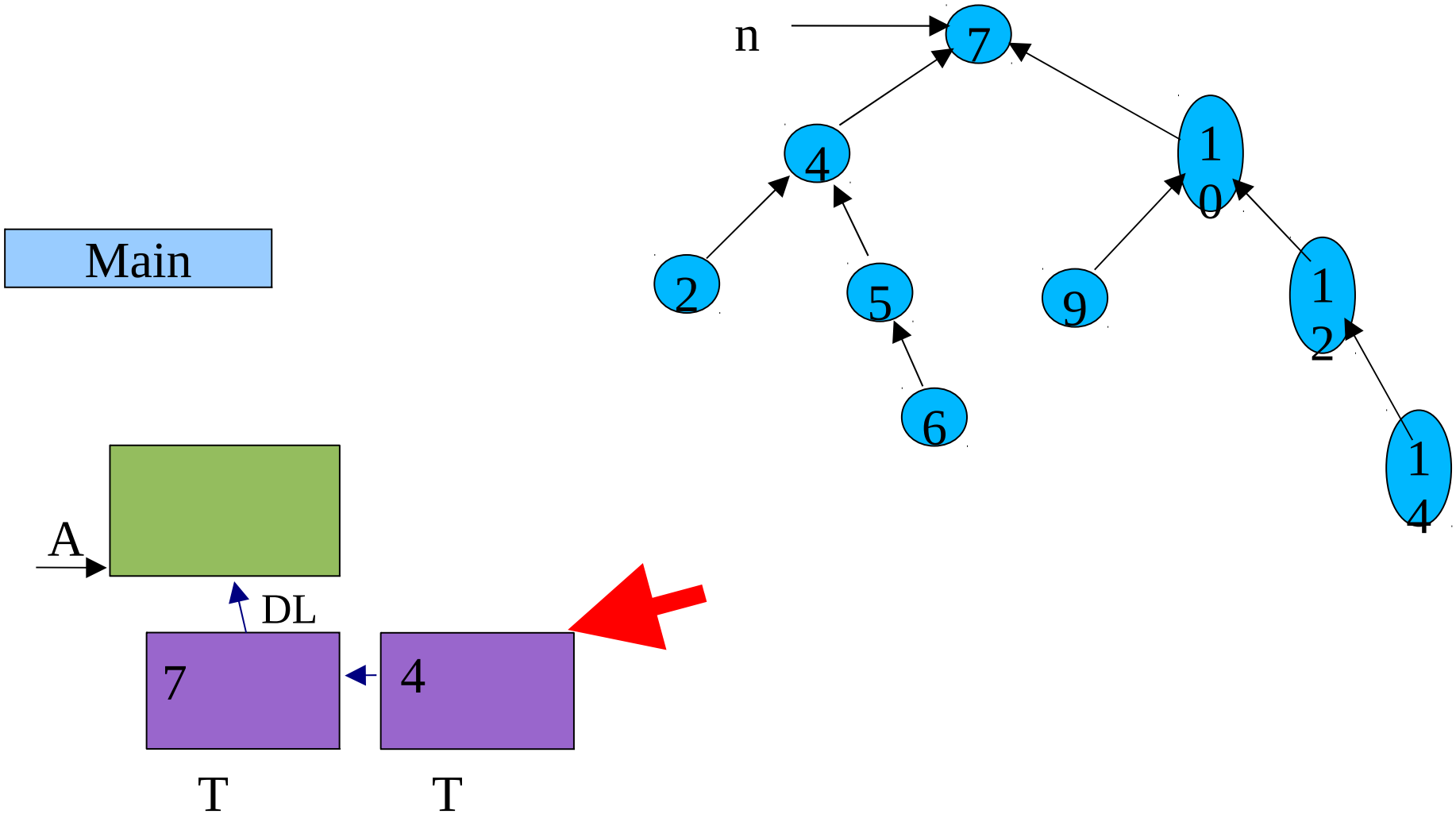


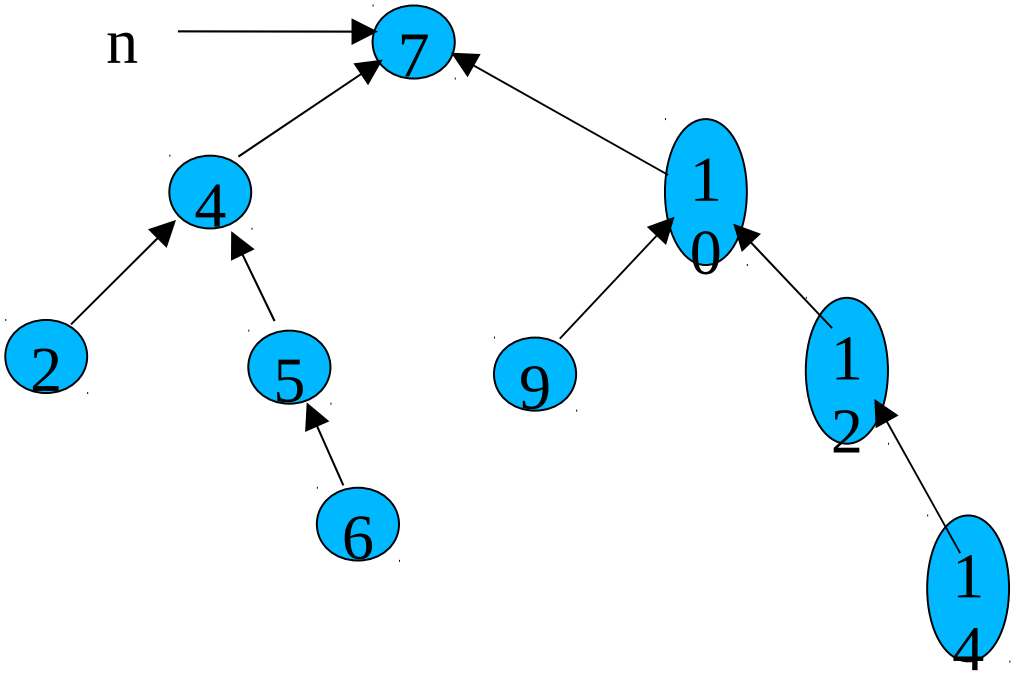

n
7
4
10
2
5
9
12
6
14
Main
A
DL
7
4
T
T
B
do
write(kolejny);
attach(A)
od


n
7
4
10
2
5
9
12
6
14
Main
A
DL
7
4
2
T
T
T
B
do
write(kolejny);
attach(A)
od

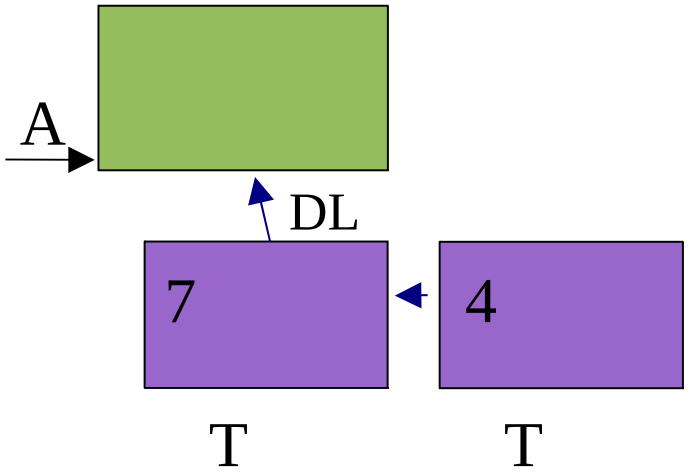

n
7
4
10
2
5
9
12
6
14
Main
A
DL
7
4
2
none
T
T
T
T
B
do
write(kolejny);
attach(A)
od


n
7
4
10
2
5
9
12
6
14
Main
A
DL
7
4
detach
T
T
T
B
do
write(kolejny);
attach(A)
od


n
7
4
10
2
5
9
12
6
14
Main
A
DL
7
4
T
T
B
do
write(kolejny);
attach(A)
od

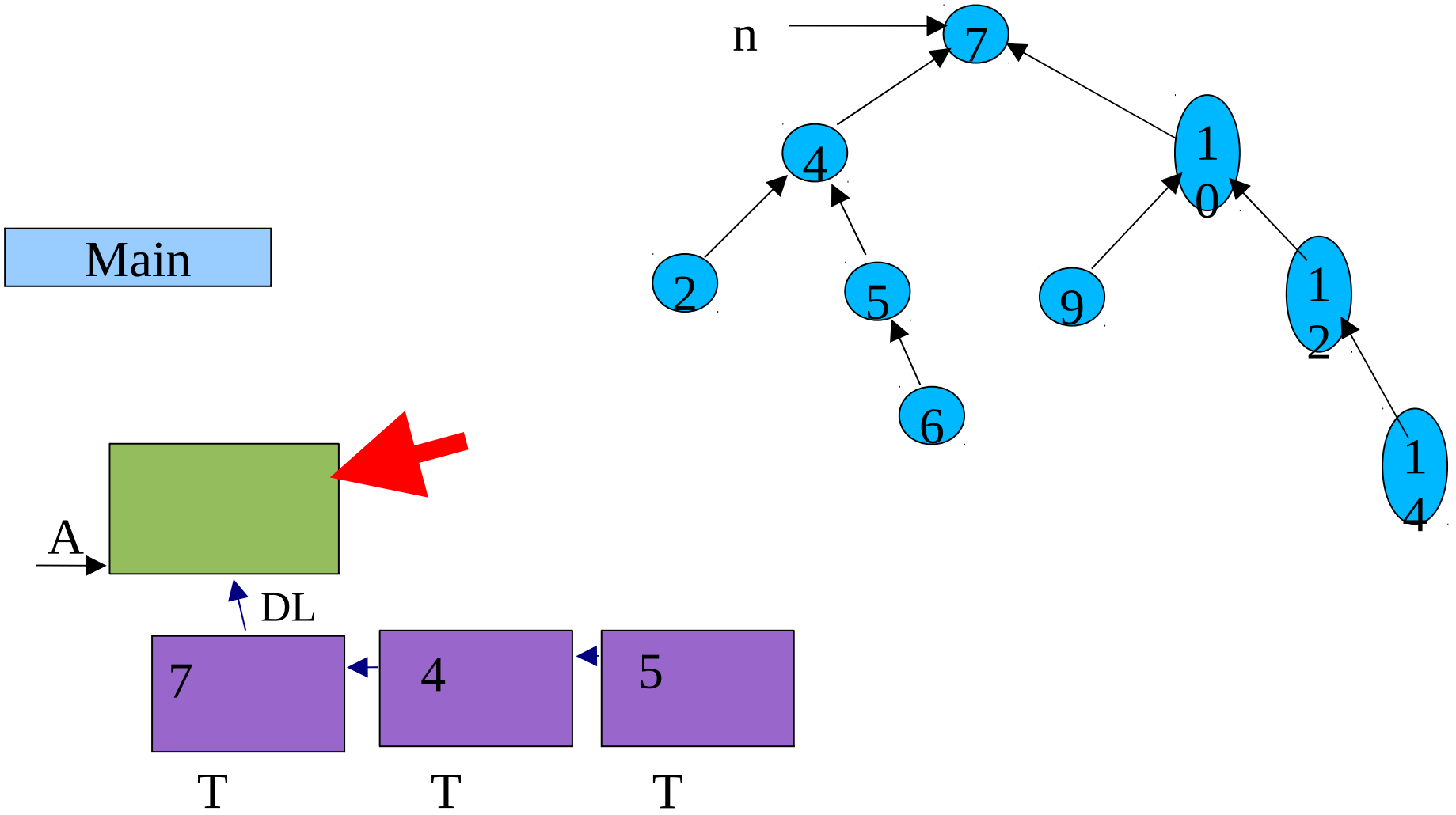


Main



B

```
do
  write(kolejny);
  attach(A)
od
```


n
7
4
10
2
5
9
12
6
14
Main
A
DL
7
4
5
T
T
T
B
do
write(kolejny);
attach(A)
od

Lemat 2

Dla kazdego drzewa binarnych poszukiwań d współpraca dwu współprogramów A i B spowoduje wydrukowanie wszystkich elementów zapisanych w drzewie d w kolejności od najmniejszego do największego.
Dowód. Wynika wprost z poprzedniego lematu i z faktu, ze instrukcja detach w nowej wersji procedury T prowadzi wprost do wykonania instrukcji write(kolejny), po niej zaś instrukcja attach(A) spowoduje wznowienie obliczeń w najnowszym rekordzie aktywacji procedury T.

Po co tak dziwacznie programowac? Lemat powyzszy pozwoli nam rozwiązać zadanie scalania drzew BST.

zbiór drzew? proszę bardzo

- rozważmy tablicę
  var Las: **arrayof** node;
- utworzenie Lasu
  array Las **dim**(1:p);
  for i:= 1 **to** n **do** Las(i) := **new** node(...) **od**;
  zmienna Las(i) wskazuje na korzeń i-tego drzewa w zbiorze, należy zadbać o podanie wartości val zapisanej w korzeniu,
- napełnianie drzewa o korzeniu n wartościami $e_1, e_2, \ldots, e_n$ polega na wykonywaniu instrukcji
  call n.insert(e_j)

Szkic programu

- Wczytaj liczbę *n* drzew BST,
- Utwórz *n* drzew i napełnij je danymi,
- Utwórz *n* wspólprogramów,
- **powtarzaj** aż do końca
  wybierz najmniejszy spośród *n* elementów i wydrukuj,
  uaktywnij odpowiedni współprogram by dostarczył kolejny większy element w tym drzewie,
- zakończ gdy wydrukowałeś wszystkie elementy

```
PROGRAM Merge;(* COROUTINES MERGE  BINARY SEARCH TREES *)
UNIT NODE : CLASS;(* NODE OF BINARY TREE *) ...
UNIT TRAVERS : COROUTINE (X :NODE); ...
VAR N,I,J,MIN,M,K : INTEGER,
     D :  ARRAYOF NODE,  TR: ARRAYOF TRAVERS;
BEGIN
 (* czytaj N ) ARRAY D DIM(1:N); M:= maxInt;
FOR I := 1 TO N
DO READ(J);D(I) := NEW NODE; D(I).VAL := J;
   DO READ(J); CALL D(I).INS(J);
         (* exit gdy j=-1 *)   OD;
OD I;            ARRAY TR DIM(1:N);  MIN := 0;
FOR I:= 1 TO N DO
  TR(I) := NEW TRAVERS(D(I));ATTACH(TR(I));
OD;  K:=0;
DO
  IF MIN = M THEN EXIT FI;  (* KONIEC exit*)
  MIN := TR(1).VAL;  J :=1;
  FOR I:= 2 TO N
  DO
   IF MIN>TR(I).VAL THEN MIN:= TR(I).VAL; J := I FI;
  OD;
  IF MIN < M THEN WRITE(MIN); ATTACH(TR(J));K:=K+1;  FI
OD;
END merge
```

Podsumowanie

- pojęcie *łańcucha dynamicznego*,
- instrukcje **attach** i **detach** przełączają pomiędzy łańcuchami dynamicznymi współprogramów,
- współpraca:
  - struktury danych,
  - wspólprogramów,
  - procedur rekurencyjnych

  może dać bardzo wiele.
- program *merge.log*

Współprogramy III

Ten wykład ma na celu pokazanie kolejnej ciekawej możliwości, którą oferują współprogramy.

Wspólprogramy reprezentujące wyrażenia regularne kooperują w celu wypisania języka regularnego.

Drugim naszym celem jest opanowanie techniki dowodzenia przez indukcję ze względu na hierarchię klas (w tym przypadku hierarchię coroutin).

Wyrażenia regularne

Niech A będzie ustalonym zbiorem. Jego elementy nazywać będziemy znakami alfabetu, a on sam nazywany będzie *alfabetem*. Niech A^* oznacza zbiór wszystkich skończonych ciągów znaków alfabetu A. Zbiór A^* zawiera ciąg pusty, oznaczany $\varnothing$.

Zakładamy, że znaki: $\cup$, $\bullet$, $*$, $($, $)$ nie należą do alfabetu A.

Definicja. Zbiorem *wyrażeń regularnych* WR nad alfabetem A jest najmniejszy zbiór wyrażeń taki, że:
wr1) każdy element zbioru A jest elementem zbioru WR,
wr2) jeśli dwa wyrażenia α i β należą do zbioru WR, to do zbioru WR należą też wyrażenia

$(\alpha \cup \beta)$, $(\alpha \bullet \beta)$, α^*

Przykłady.

Niech $A = \{x, y, z\}$. Napisy x, z, $((x)\bullet(y \cup z))$, $(x \cup y)^*$ są wyrażeniami regularnymi, napis $(x \cup t)^*$ nie jest wyrażeniem regularnym nad alfabetem A.

Języki regularne

Z każdym wyrażeniem regularnym α można związać zbiór $|\alpha|$ ciągów skończonych znaków z alfabetu A w następujący sposób.

Definicja.

Jeżeli $\alpha=a$ jest atomowym wyrażeniem regularnym $\alpha \in A$ to $|\alpha| = \{a\}$,

Jeżeli α jest wyrażeniem regularnym postaci $(\beta \cup \gamma)$ to $|\alpha| = |\beta| \cup |\gamma|$,

jeżeli α jest wyrażeniem regularnym postaci $(\beta \bullet \gamma)$ to

$|\alpha| = \{uw: u \in |\beta| \text{ i } w \in |\gamma|\}$,

jeżeli α jest wyrażeniem regularnym postaci β^*, to

$|\alpha| = |\beta^*| = \{\varnothing \cup |\beta| \cup |\beta \bullet \beta| \cup |(\beta \bullet \beta) \bullet \beta| \cup \ldots \cup |\beta \bullet \beta \bullet \ldots \bullet \beta| \cup \ldots\}$

Elementy zbioru $|\alpha|$ nazywamy słowami języka regularnego $|a|$.

Przykład.

$|(x \bullet (y \cup z)| = \{xy, xz\}$

$|(x \cup y)^*| = \{\varnothing, x, y, xx, xy, yx, yy, xxy, xyx, ...\}$ ten język regularny zawiera wszystkie słowa skończone zawierające znaki x i y.

Zastosowania wyrażeń regularnych

Wyrażenia i języki regularne znalazły wiele zastosowań:

- w kompilatorach,
- w opisie procesów,
- w lingwistyce formalnej,
- w teorii złożoności,
- w aplikacjach dotyczących sterowania.

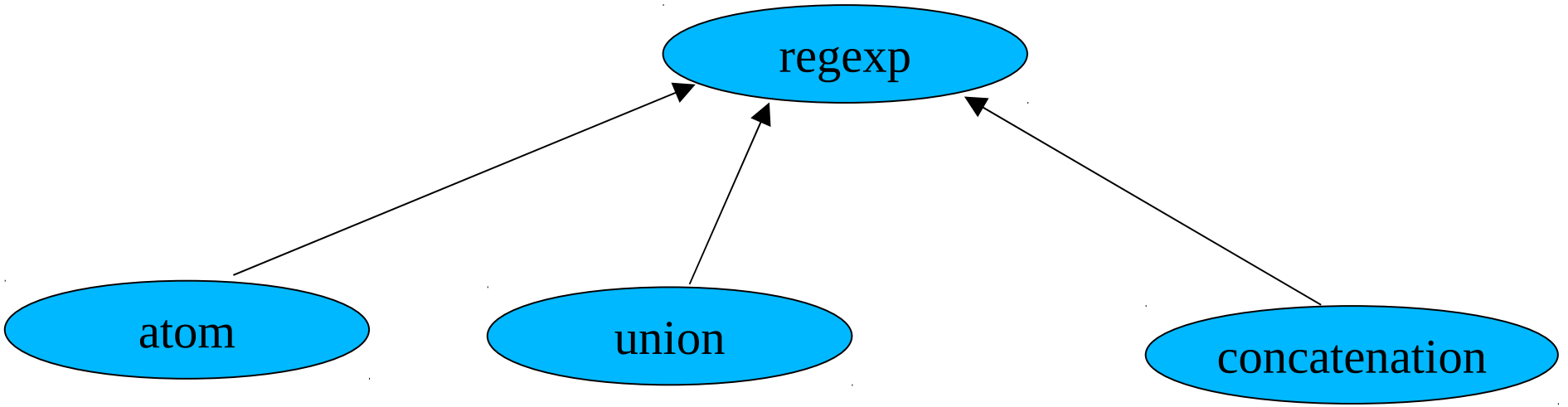
Obliczanie wartości wyrażeń regularnych

Na ogól chcemy znaleźć odpowiedź na pytanie czy dane słowo należy do określonego języka regularnego.
Czasami jednak chcemy wypisać wszystkie słowa należące do języka regularnego opisanego wyrażeniem regularnym α.

Jest oczywiste, że nie da się tego zrobić dla języków opisanych wyrażeniami zawierającymi znak *.

Wobec tego w dalszym ciągu tego wykładu *ograniczamy* się do wyrażeń regularnych, w których nie występuje operator *.

Hierarchia coroutin: regexp



Klasa bazowa hierarchii: regexp

```
unit REGEXP:coroutine;
    var B:BOOL; (* B all the words of the language were shown *)
begin
    return
    inner;
    B := true
end REGEXP;
```

klasa Atom

```
unit ATOM: REGEXP class(C:CHAR);
begin
   do
        I:=I+1;    (* update the position *)
        Word(I):=C;
        B:=TRUE;
        detach
   od
end ATOM;
```

klasa Union

```
unit UNION: REGEXP
                    class(L,R:REGEXP);
   var M: INTEGER;
begin
    do
      M:=I;
      do
          attach(L);
          if L.B then exit fi;
          detach;
          I:=M
      od;
      L.B:=FALSE;
      do
         detach;
         I:=M;
         attach(R);
         if R.B then exit fi;
      od;
      R.B:=FALSE;
      B:=TRUE;
      detach;
    od;
end UNION;
```

klasa Concatenation

unit CONCATENATION: regexp class(L, R: regexp);

```
    var N,M:INTEGER;
begin
  do
     M:=I;
     do
        attach(L);
        N:=I;
        do
          attach(R);
          if R.B then if L.B
          then exit exit
          else exit fi fi;
         detach;
         I:=N
     od;
        R.B:=FALSE;
        detach;
        I:=M
   od;
   R.B,L.B:=FALSE;
   B:=TRUE;
   detach
od;
end CONCATENATION;
```

Twierdzenie

Dla każdego obiektu o spełniającego relację o in regexp, następujący program Pr wydrukuje wszystkie słowa języka regularnego reprezentowanego przez ten obiekt o i zatrzyma się.

```
Pr:  I:=0;
     do
        attach(o);
        drukuj zawartość tablicy Word
                        for J:=1 to I
                        do
                            write(Word(J))
                        od;       writeln;
        if W.B then exit fi
     od
```

Lemat

Załóżmy, że miejsca tablicy Word są wypełnione aż do pozycji I. Załóżmy ponadto, że pewne słowa z języka L(o) zostały już wcześniej wytworzone i wydrukowane przez wcześniejsze aktywowanie współprogramu o. (W tablicy Word nie ma więc już po nich śladu.)

Wykonanie instrukcji **attach(o)** ma następujący efekt: kolejne słowo języka L(o) zostanie dopisane do tablicy Word poczynając od pozycji Word(I+1). Atrybut B przyjmie wartość true wtedy i tylko wtedy gdy wszystkie słowa języka L(o) zostaną wydrukowane.

dowód lematu

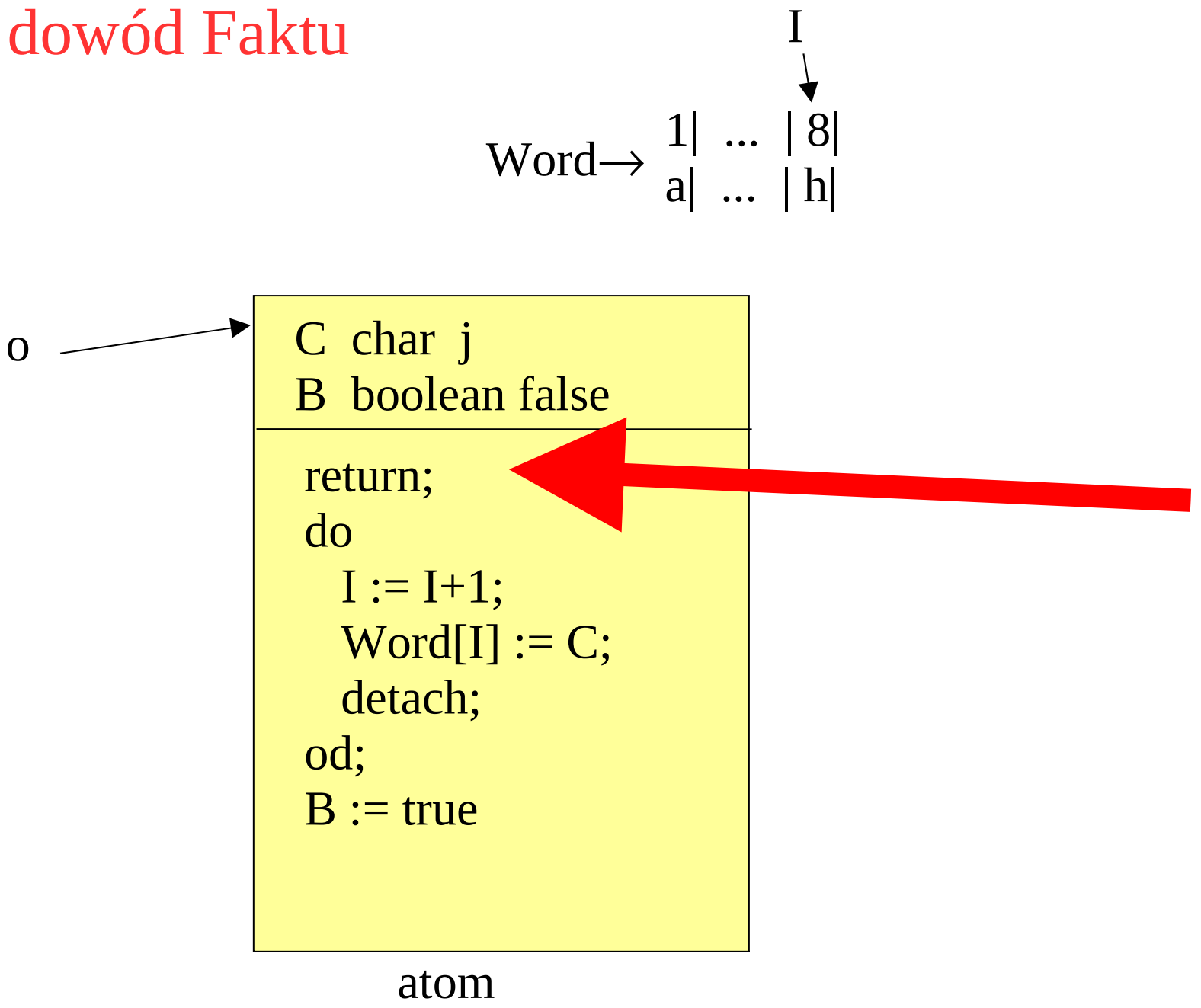
Obiekt o należy do klasy Atom, lub do klasy Union lub do klasy Concatenation.
W każdym z tych przypadków jest on korzeniem pewnego drzewa obiektów.

Dowód będzie przebiegać ze względu na wysokość tego drzewa.

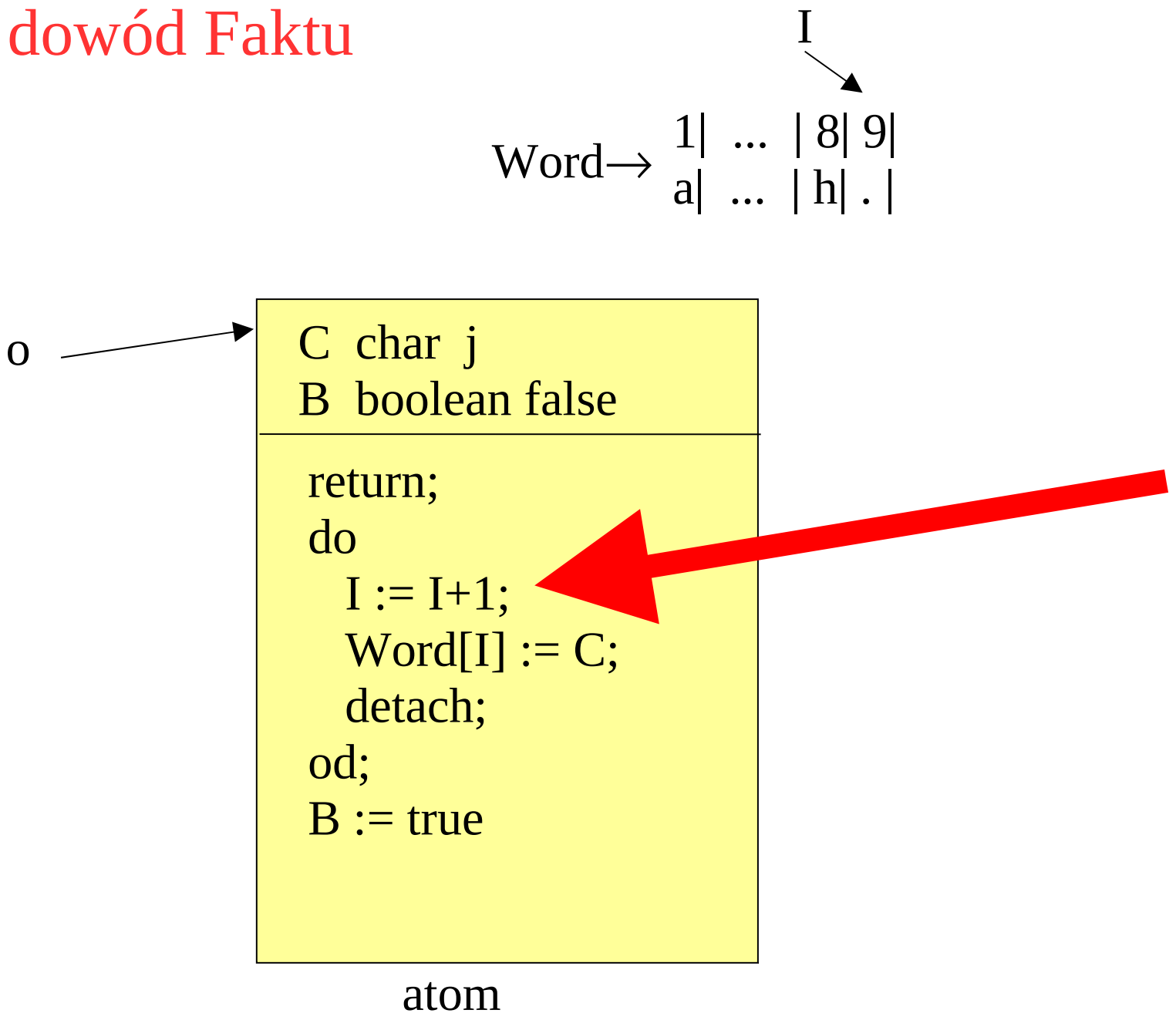
W przypadku gdy obiekt o jest atomem, drzewo ma wysokość zero, i zachodzi następujący

Fakt.
Wykonanie instrukcji **attach(o)** zastosowanej do obiektu o takiego, że o **in** Atom spowoduje umieszczenie litery C na I+1 miejscu tablicy Word i atrybut B w tym obiekcie przyjmie wartość true.

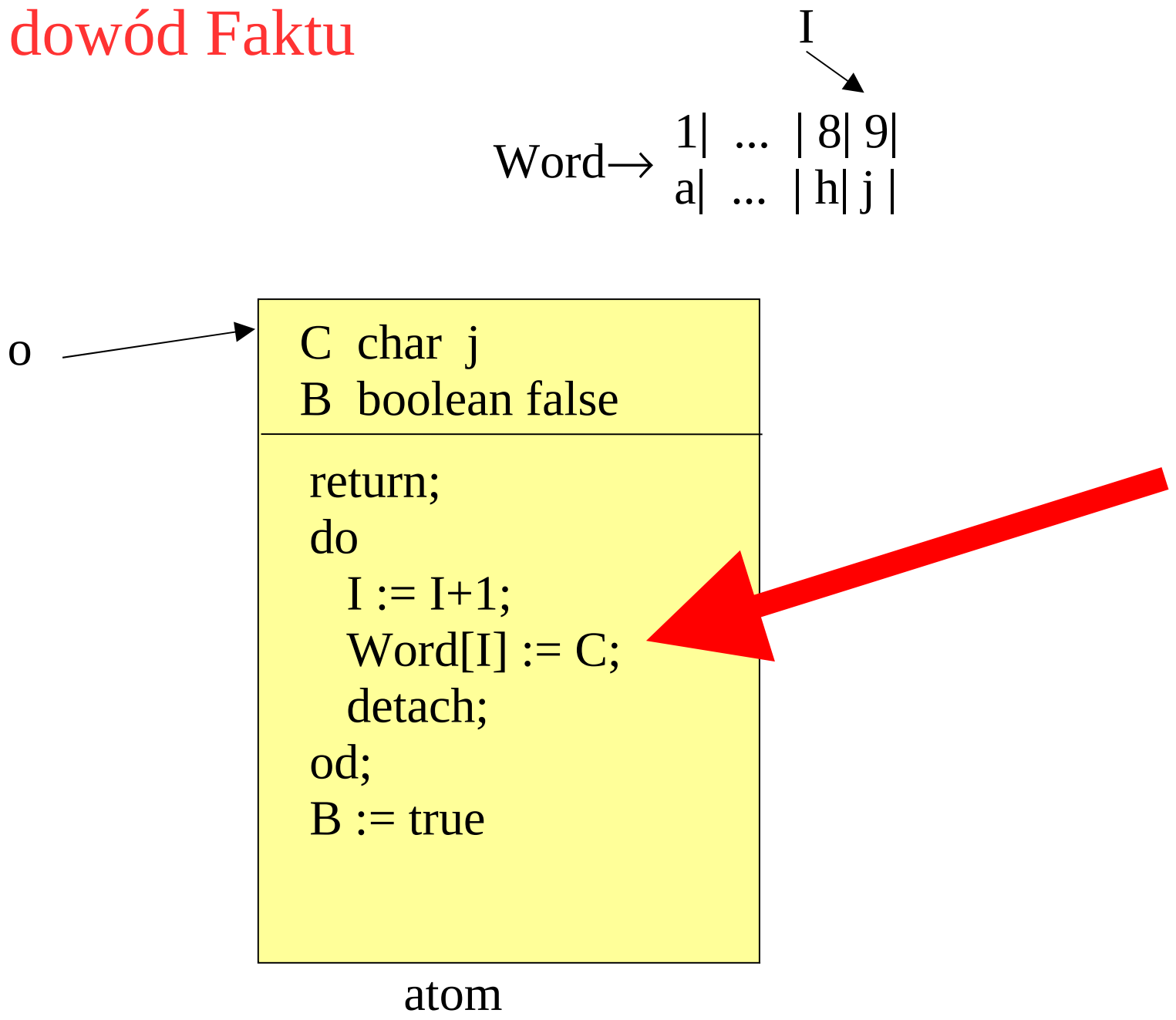
dowód Faktu


I
Word→
1| ... | 8|
a| ... | h|
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

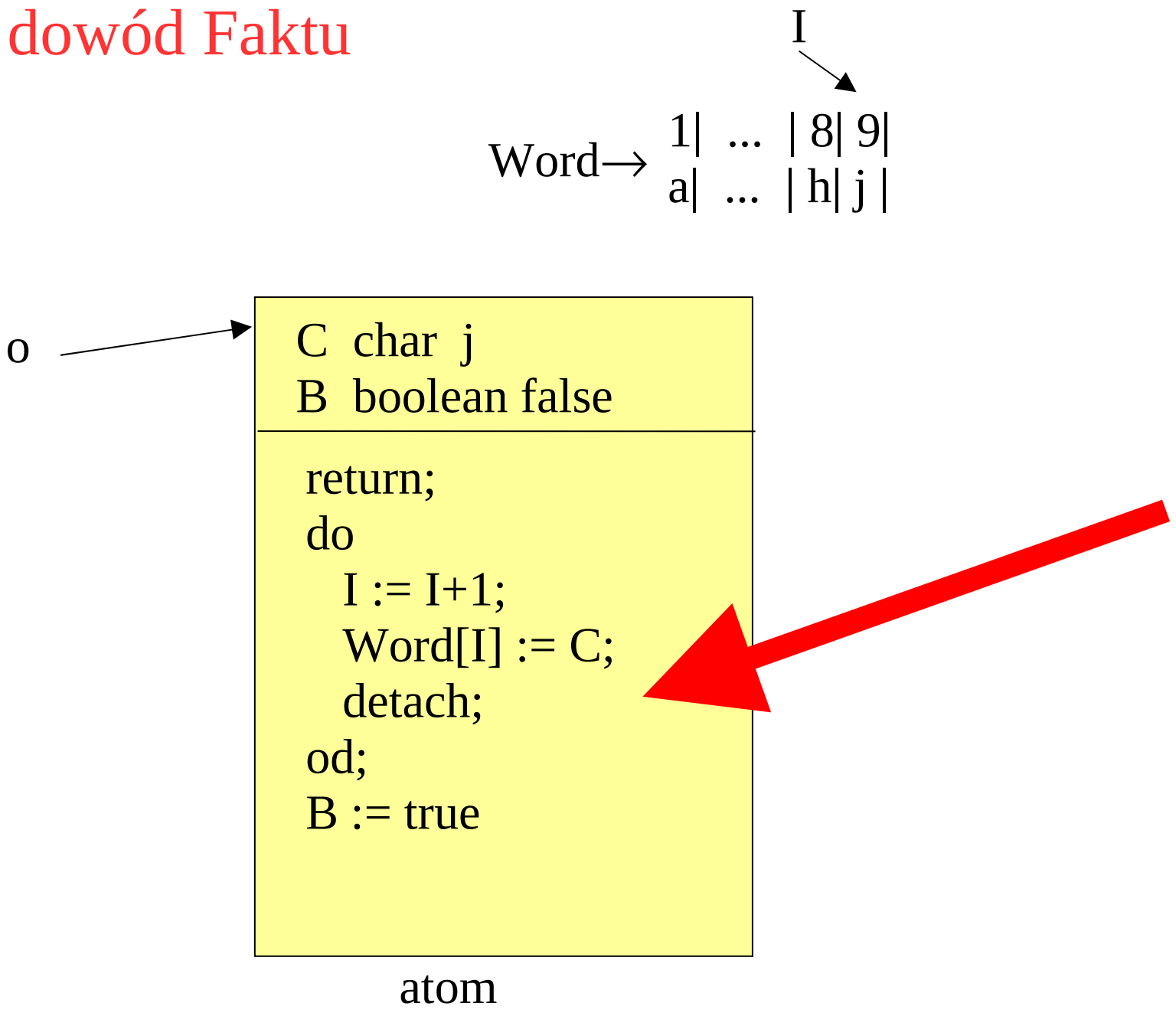
dowód Faktu


I
Word→
1| ... | 8| 9|
a| ... | h| . |
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

dowód Faktu


I
Word→
1| ... | 8| 9|
a| ... | h| j |
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

dowód Faktu


I
Word→
1| ... | 8| 9|
a| ... | h| j |
o
C char j
B boolean false
return;
do
I := I+1;
Word[I] := C;
detach;
od;
B := true
atom

dowód lematu c.d.

Pozostaje nam do wykazania, że jeśli lemat jest spełniony dla obiektów L i R będących korzeniami lewego i prawego poddrzewa drzewa o korzeniu o, to lemat jest spełniony dla samego obiektu o.

Przypadek A) o in Union

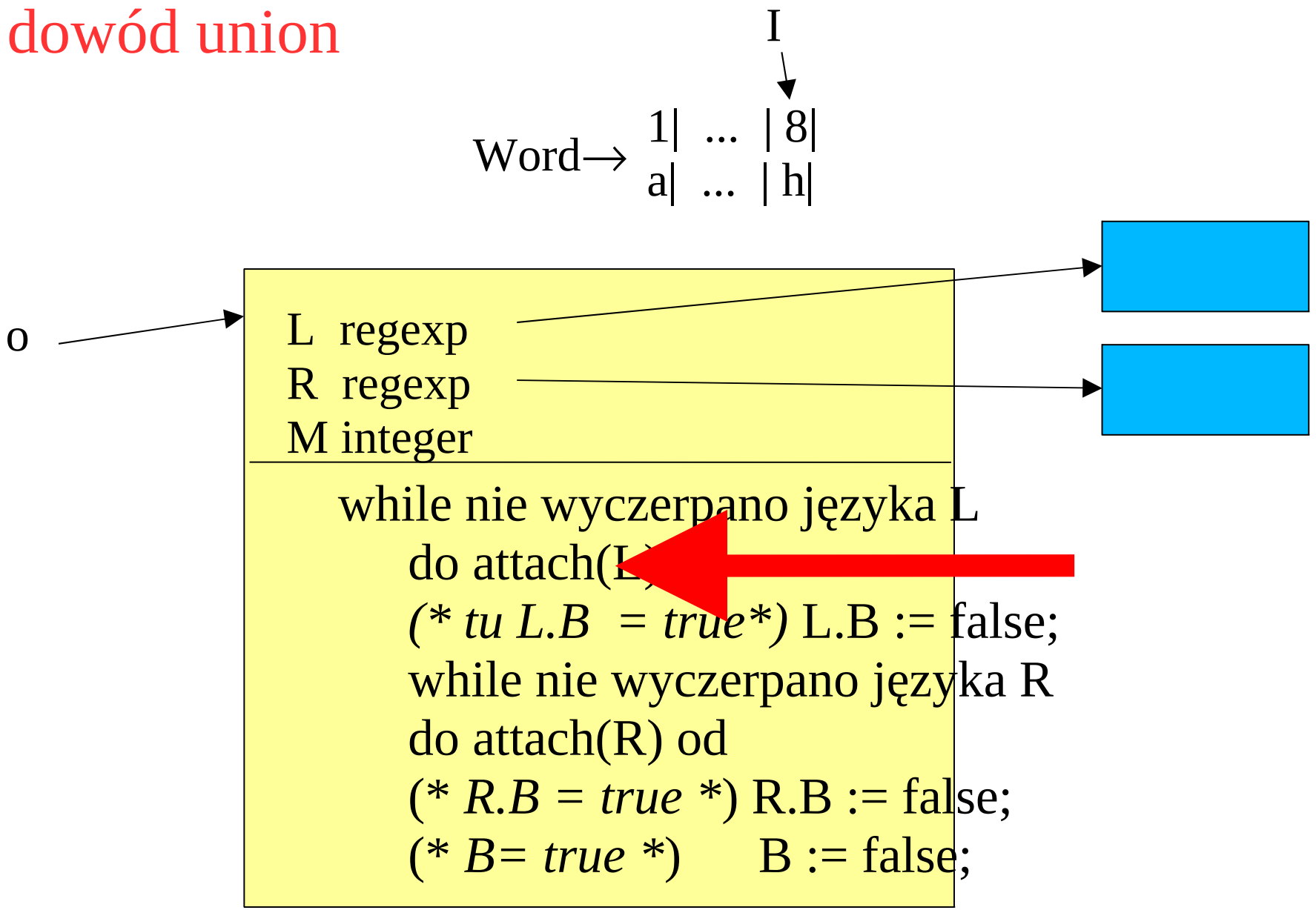
Struktura instrukcji we współprogramie o jest następująca

```
while nie wyczerpano języka L
do attach(L) od        -- z założenia indukcyjnego
(* tu L.B  = true*) L.B := false;
while nie wyczerpano języka R
do attach(R) od
(* R.B = true *) R.B := false;
(* B= true *)      B := false;
```

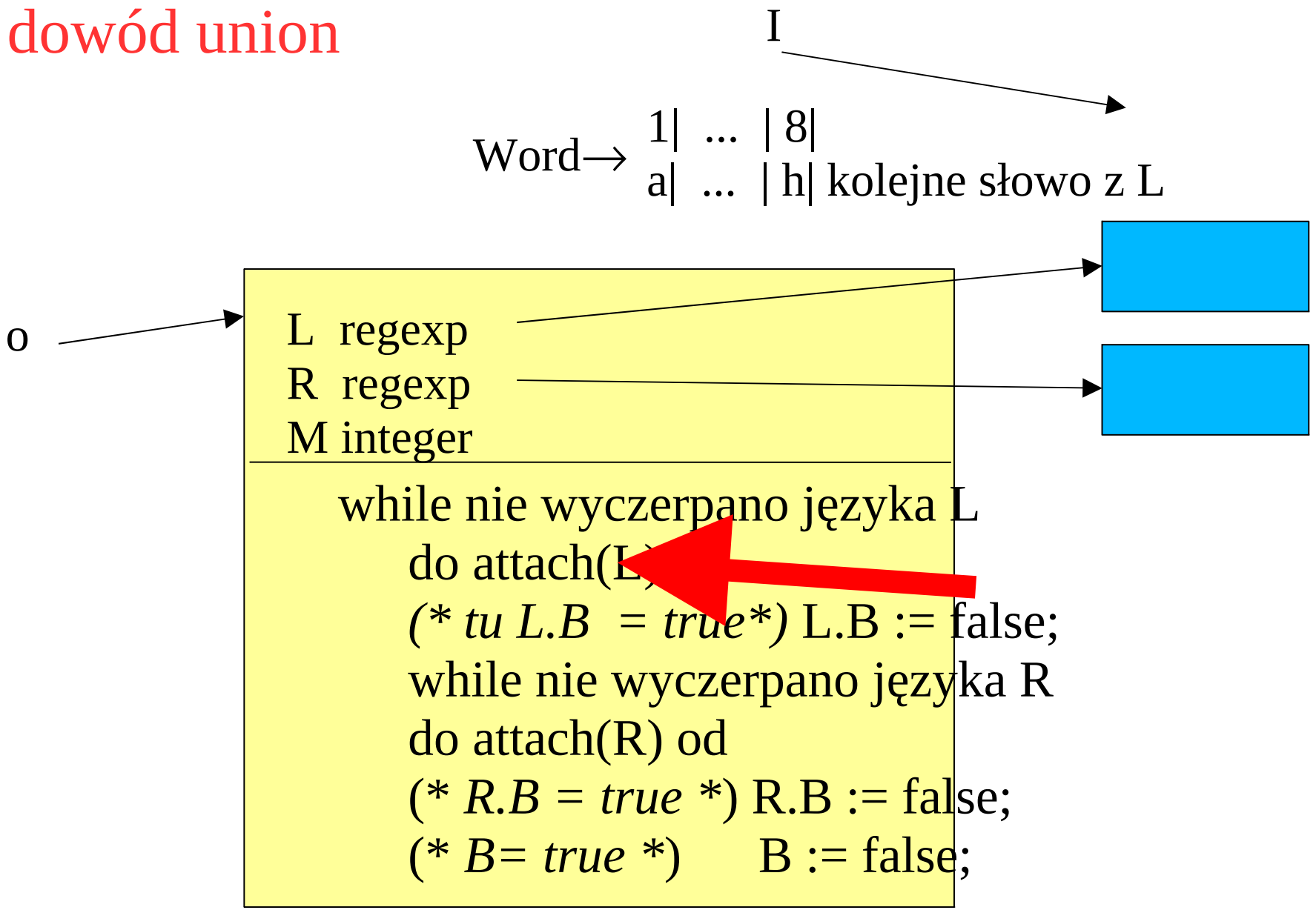
dowód lematu c.d.

```
unit Union: regexp  class(L, R: regexp);
     var M: integer;
begin
   do
     M:=I; (* I is the position of the lastly generated letter.  *)
     do
       attach(L)   (* by the inductive assumption this statement causes that one word will be generated of
                      the language L and it will be concatenated to the content of WORD(1) , ... , WORD(I) *)
       if L.B then exit fi;
       detach;
       I:=M          (* reestablish the position in the table WORD for the next word  *)
     od;
     L.B:=FALSE;  (* restart language L *)
     do
       detach;
       I:=M;          (* reestablish the position in the table WORD for the next word  *)
       attach(R);   (* by  the inductive assumption this statement causes that one word will be generated of
                      the language R and it will be concatenated to the content of WORD(1) , ... , WORD(I) *)
       if R.B then exit fi;
     od;
     R.B:=FALSE;  (* restart language R *)
     B:=TRUE;
     detach;
   od;
 end Union;
```

dowód union


I
Word→
1| ... | 8|
a| ... | h|
o
L regexp
R regexp
M integer
while nie wyczerpano języka L
do attach(L)
(* tu L.B = true*) L.B := false;
while nie wyczerpano języka R
do attach(R) od
(* R.B = true *) R.B := false;
(* B= true *) B := false;
union

dowód union


I
Word→
1| ... | 8|
a| ... | h| kolejne słowo z L
o
L regexp
R regexp
M integer
while nie wyczerpano języka L
do attach(L)
(* tu L.B = true*) L.B := false;
while nie wyczerpano języka R
do attach(R) od
(* R.B = true *) R.B := false;
(* B= true *) B := false;
union

dowód lematu c.d.

Przypadek B) o in Concatenation

Struktura instrukcji we współprogramie o jest następująca

```
while nie wyczerpano języka L
do
    zapamietaj I;
    attach(L);        -- słowo z języka L
    attach(R);        --  a po nim słowo z języka R
    detach;           --  a więc słowo z języka (L • R)
    odtwórz I;
 od;
```

dowód lematu c.d.

```
unit CONCATENATION: REGEXP class(L,R:REGEXP);
      var N,M:INTEGER;
 begin
   do
     M:=I; (*begin of first language word position *)
     do
       attach(L);
       N:=I; (* begin of the second language word position *)
       do
         attach(R);
         if R.B then if L.B then exit exit else exit fi fi;
         detach; I:=N (* restart language R word generation position *)
       od;
       R.B:=FALSE; (* restart language R *)
       detach; I:=M (* restart language L word generation position *)
     od;
     R.B,L.B:=FALSE; B:=TRUE; detach
   od;
 end CONCATENATION;
```

Program główny

```
const N=50;   (* DIMENSION FOR ARRAY WORD *)
     var          A,B,C,D,E,W,V,L,O,G,II,NN:REGEXP, I,J,N,M:INTEGER;
                    (* I = GLOBAL POSITION POINTER FOR ARRAY WORD *)
     var WORD: arrayof CHAR; (* BUFFER FOR WORDS GENERATION *)
 begin
    A:=new ATOM('A'); B:=new ATOM('B'); C:=new ATOM('C');    D:=new ATOM('D');
     E:=new ATOM('E');  L:=new ATOM('L'); G:=new ATOM('G');    II:=new ATOM('I');
     NN:=new ATOM('N');  O:=new ATOM('O');  W:=new UNION(A,L);
     W:=new CONCATENATION(W,new UNION(D,O));
     V:=new CONCATENATION(II,C);
     V:=new UNION(V,new CONCATENATION(L,new CONCATENATION(A,NN)));
     V:=new CONCATENATION(G,V);  V:=new UNION(A,V);
     W:=new CONCATENATION(W,V);
     writeln(" WE HAVE LANGUAGE DEFINED BY THE FOLLOWING
EXPRESSION");
     writeln(" (A∪L)•(D∪O)•(A∪G•(I•C∪L•A•N))");
     array WORD dim(1:N);
    do
      attach(W);
      write(" ");      for J:=1 to I  do   write(WORD(J))   od;
      if W.B then exit fi
   od
 end
```

Czy wiemy co wydrukuje ten program?

Czy potrzebne jest nam do tego wykonanie programu?

==========

przeczytaj i uruchom program z pliku:
treegen.log

dodaj klasę iteration pochodną klasy regexp

Współprogramy IV

Współprogram może pozwolić na szybsze wykonanie

- Przygotujmy obiekty coroutin odpowiadające rekordom aktywać procedur
- Przykład – wieże Hanoi

Procedura

```
unit Hanoi: procedure(ile, z , na: integer);
   var k: integer;
begin
   k:=6 -(z+na) ;    (* z + na +k = 6 *)
   if ile>1 then call Hanoi(ile-1, z, k) fi;
   przenieś krążek z wieży z na wieżę na;
   if ile>1 then call Hanoi(ile-1, k, na) fi
end Hanoi;
```

Tablica współprogramów I

Utwórzmy tablicę współprogramów – każdemu możliwemu rekordowi aktywacji procedury będzie odpowiadać jeden współprogram

var obiekty: arrayof arrayof arrayof CHanoi;

Wartość ile zmienia się od 1 do n= liczba krążków
wartości z oraz na zmieniają się od 1 do 3

Tablica współprogramów II

Wypełniamy tablicę obiekty:

```
for ile := 1 to n do
   newarray obiekty(ile) dim (1:3);
   for z := 1 to 3 do
      newarray obiekty(ile, z) dim (1:3)
      for na := 1 to 3 do
         if z =/= na then obiekty(ile,z,na):=new CHanoi(ile,z,na)
      od
   od
od
```

Współprogram CHanoi

```
unit CHanoi: coroutine(ile, z, na : integer);
    var k: integer;
begin
    k:= 6-(z + na);
    return;
    do
       if ile>1 then attach(obiekty[ile-1, z, k]) fi;
       przenieś krążek z wieży z na wieżę na;
       if ile>1 then attach(obiekty[ile-1, k, na]) fi;
    od
end CHanoi;
```

Zmontuj to razem i … wykonaj

```
program WieżeHanoi;
    unit Chanoi: coroutine…

    var obiekty: arrayof arrayof arrayof Chanoi,
          i,j,m: integer;
    const n=64;
begin
    (* utwórz tablicę obiekty *)
     ...
    attach(obiekty[n, 1, 3])
end WieżeHanoi;
```

Spostrzeżenia

- Obie wersje programu WieżeHanoi: *rekurencyjna* i *ze współprogramami* są równoważne!
- Druga wersja będzie działać znacznie szybciej – ponieważ zaoszczędzimy na tworzeniu rekordów aktywacji.
- Oczywiście sam problem ma złożoność wykładniczą i nic na to nie poradzimy.
  Ale w wielu sytuacjach można sporo zyskać, np. zastępując procedury modyfikujące strukturę danych np. DRZEWO przez odpowiednie współprogramy.

Drzewa BST z coroutinami

Zamiast procedur

- Insert
- Delete
- IsMember

Stwórz 3 współprogramy o podobnych nazwach.

Wywołanie

argI:= elem; attach(insrt);	argD:=elem; attach(delete);	ArgM := elem; attach(isMember);

Symulacja

- Bardzo ważne zastosowanie współprogramów to narzędzie do symulacji procesów z czasem dyskretnym.

  Postaramy się rozwinąć ten temat przy innej okazji.

Problems

- Is it possible to implement coroutines in Java?
  Attention. We exclude a possibility of using threads as coroutines!
  Prize: 50 €

- Give a complete and concise specification of coroutines.

-

Literatura

- Conway
- D. Knuth
- Marlin *Coroutines,* LNCS xxx SpringerVlg
- O.-J. Dahl, K. Nygaard, *Simula67,* NCC, Oslo
- O.-J.Dahl, A. Wang, *Coroutine sequencing in a block structured environment,* BIT, 1971, pp. 425-
- *Report on Loglan'82*, PWN 1983
- W.M. Ratajczak-Bartol *Opis współprogramów w Loglanie,*PhD I Inf UW 1982
- B.B. Kristenssen*, Coroutine sequencing in BETA,* report 235 CS Dept. Aarhus Univ. 1989