Програмирање 1

Групно спремање – питалице

Увод у Паскал

Општи пример програма

```
PROGRAM ime(input,output);

CONST pi = 3.14;
      max = 100;
TYPE niz = array [0..max] of integer;
     ceo = integer;
VAR i:integer;
    x,y,z:ceo;

PROCEDURE p();
BEGIN
  ...
END;

FUNCTION f();
BEGIN
  ...
END;

BEGIN
  ... {komentar}
END.
```

Типови података (прости)

Основни (предефинисани):

`integer` – цео број (2, `350`, `-17`)

`real` – реални број (`3.125`, `18e10`, `1.2e-7`)

`char` – знак (`'a'`, `'R'`, `'0'`, `'?'`)

`boolean` – логички тип (`false`, `true`)

Типови `integer`, `char` и `boolean` су набројиви (могуће их је пребројати).

Опсег (део неког од набројивих типова):

`0..100` – цели бројеви од 0 до 100

`'a'..'z'` – знаци од 'a' до 'z'

Типови података (прости)

Набројани тип:

(*ident1*, *ident2*, *ident3*,… , *identN*)

Набраја се низ идентификатора (имена) који представљају могуће вредности овог типа.

Пример:

```
VAR dan:(pon,uto,sre,cet,pet,sub,ned);
```

Операције:

```
dan:=uto;
dan:=succ(sre); {sledbenik -> cet}
dan:=pred(sre); {prethodnik -> uto}
ord(sre); {redni broj -> 2 (počinje od 0)}
```

не могу се читати нити писати помоћу `read` и `write`

Типови података (сложени)

`string` – низ знакова, текст (`'ovo je string'`)

Пример:

```
VAR s,s1,s2:string[30];
```

Операције:

```
s1:='string';
s2:=''; {prazan string}
s:=s1 + s2; {nadovezivanje stringova}
read(s); {citanje}
write(s); {pisanje}
s[3]; {treći znak u stringu}
```

Типови података (сложени)

Низови:

`array [opseg] of tip` – дефинише низ задатог типа

Пример:

```
VAR niz: array [1..100] of integer;
```

Операције:

```
niz[3]:=123; {pristup elementu niza}
```

морају се читати и писати елемент по елемент
(не могу одједном)

Типови података (сложени)

Скупови:

`set of` *opseg* – скуп елемената из задатог опсега (опсег не сме бити већи од 255 елемената)

Пример:

```
VAR skup,s1,s2: set of 1..100;
```

Operacije:

```
s1:=[1,2,4..7,9]; {skup sadrži:1,2,4,5,6,7,9}
s2:=[]; {prazan skup}
skup:=s1+s2; {unija}
skup:=s1-s2; {razlika}
skup:=s1*s2; {presek}
broj IN skup; {da li je broj u skupu}
skup:=skup+[broj]; {dodavanje broja u skup}
```

Типови података (сложени)

Записи (Record):

`RECORD deklaracije_polja END;` – сложени тип који се састоји од више „поља“; поља су променљиве које су груписане унутар податка овог типа

Пример:

```
VAR komplBr: RECORD
       r, i: real;
     END;
```

Операције:

```
komplBr.r:=10; {pristup polju zapisa}
komplBr.i:=-3; {komplBr je 10-3i}
```

Дефинисање типа

Могуће је увести нова имена за било који тип података.
Имена се уводе у TYPE делу.

```
TYPE novoIme = opisTipa;
```

Пример:

```
TYPE ceo = integer;
     dan = (pon,uto,sre,cet,pet,sub,ned);
     ocena = 5..10;

VAR o:ocena; {umesto VAR o:5..10}
    d:dan;
    i:ceo;
```

Фебруар 2012. (2. питалица)

```
program pitanje(input, output);
  type st = (prvo, drugo, trece);
  var a: array [1..100] of integer;
      n, i: integer;
      s: st;
begin
  n:= 4;
  for i:= n downto 1 do
  read(input, a[i]);
  s:= prvo;
  i:= 1;
    repeat
      case s of
        prvo: begin a[i]:= a[i+1] - a[i]; i:= i+1 end;
        drugo: begin a[i]:= a[i-1] + a[i] end;
        trece: begin i:=i+1; a[i]:= a[i] div 10 end;
      end;
      if (ord(s) <= ord(drugo)) then s:=succ(s);
      write(output,a[i],' ');
    until (i mod 4) = 0;
end.
```

A) 20 10 1 B) 30 20 2 1 C) 30 10 3 4

Процедуре и функције

Дефинисање процедуре и функције

```
PROCEDURE ime(a,b:integer; VAR c:integer);
VAR ...
BEGIN
   ...
END;
```

```
FUNCTION ime(a,b:integer; VAR c:integer):povratniTip;
VAR ...
BEGIN
   ...
   ime:=povratnaVrednost;
   ...
END;
```

Позив процедуре и функције

```
PROGRAM prog(output);
PROCEDURE p(x:integer); {ispisuje prosleđenu vrednost}
BEGIN
  writeln('x=', x);
END;

FUNCTION f(n:integer):integer; {računa faktorijel}
VAR i:integer;
BEGIN
  f:=1;
  for i:=2 to n do f:=f*i;
END;

VAR a,b:integer;
BEGIN
  a:=3; b:=3;
  p(a); {ispisuje „x=3“}
  p(7); {ispisuje „x=7“}
  b:=p(7); {GREŠKA: p je procedura, nema povratnu vrednost}
  b:=f(4); {upisuje 4!=24 u promenljivu b}
  writeln(f(a)); {ispisuje „6“}
  f(5); {računa vrednost 5!, ali se sa tom vrednošću ništa ne radi}
END.
```

Прослеђивање параметара

По вредности (без VAR) – приликом прослеђивања, прослеђена вредност се копира и прави се локална копија прослеђеног податка. Мењање копије не утиче на оригинал.

По референци (са VAR) – приликом прослеђивања додаје се ново име већ прослеђеној променљивој (референца), помоћу кога јој се може приступати из процедуре. Измена параметра повлачи и измену прослеђене променљиве.

Прослеђивање параметара

```
PROGRAM prog(output);

PROCEDURE p(x:integer; VAR y:integer);
BEGIN
  x:=10;
  y:=10;
  writeln(x,y);
END;

VAR a,b:integer;
BEGIN
  a:=3;
  b:=3;
  p(a,b);
  writeln(a,b);
END.
```

ispisuje: 10 10

ispisuje: 3 10

Живот променљивих и опсег важења

```
PROGRAM prog;
VAR x,y,z:integer;

  PROCEDURE p();
  VAR x,a,b,c:integer;
    PROCEDURE pp();
    VAR y,a,t:integer;
    BEGIN ... END;
  BEGIN ... END;

  PROCEDURE q();
  VAR x,y,a,s:integer;
  BEGIN ... END;

BEGIN ... END.
```

Опсези важења
(видљивост променљивих):

prog: **x**, **y**, **z**

p(): **x**, **a**, **b**, **c**, ~~**x**~~, **y**, **z**

pp(): **y**, **a**, **t**, **x**, ~~**a**~~, **b**, **c**, ~~**x**~~, ~~**y**~~, **z**

q(): **x**, **y**, **a**, **s**, ~~**x**~~, ~~**y**~~, **z**

Живот променљивих:

Променљиве живе од тренутка када њихова процедура почне са радом, до њеног краја. Редослед приступа променљивој **увек** се одређује на основу права опсега важења, а не на основу тога која је променљива најмлађа (најкасније створена).

Октобар 2014. (4. питалица)

```
PROGRAM xy (output);

VAR a, b, c, d, e, f:integer;

PROCEDURE P(VAR c:integer; d,e:integer; VAR f:integer);
  VAR b: integer;
BEGIN
  a:=a+1; b:=1; c:=c+1; d:=d+1; e:=e+1; f:=f+1;
END;

BEGIN
  a:= 1; b:= 2; c:= 3; d:= 4; e:= 5; f:= 6;
  P (b, d, b, d);
  writeln (a:2, b:2, c:2, d:2, e:2, f:2)
END.
```

A) 2 4 3 5 5 6 B) 2 3 3 5 5 6 C) 2 2 3 4 5 6

Фебруар 2009. (3. питалица)

```
program pq(output);
   var a, b, c, d, e, f : integer;

procedure p(a:integer; var b:integer; c:integer; var d:integer);
   procedure q(e:integer; var f:integer);
   begin
       e:=e*2; f:=f*3
   end;
begin
   a:=a+1; b:=b+1; c:=c+b; d:=d+c; q(d, e); e:=e+d; f:=f+1
end;

begin
   a:=1; b:=1; c:=1; d:=1; e:=1; f:=1;
   p(a, c, b, f);
   writeln(output, a,b,c,d,e,f)
end.
```

A) 112155 B) 112175 C) 212173

Фебруар 2012. (3. питалица)

```
program zad(input, output);
var a, b, c, x, y:integer;

function f(var a:integer; b, c: integer):integer;
begin
  x:= a + b; a:= b - c; f:= x + c
end;

procedure p(var x, y: integer);
begin
  a:= b - c;
  x:= x + f(y, x, y);
  y:= y - x
end;

begin
  a:= 1; b:= 2; c:= 3; x:= 4; y:= 5;
  p(a, c);
  write(a,' ',c)
end.
```

A) 4 -8 B) -3 2 C) 0 0

Септембар 2010. (4. питалица)

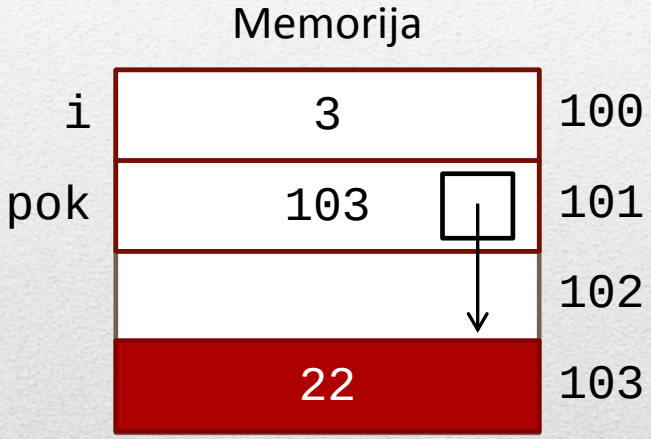
```
program rekurzija(output);
  var a, b, c: integer;
function f(var c, d: integer):integer;
begin
  if c>1 then begin
    a := a-3;
    c := c - 4;
    f := f(a, c) + 3
  end
  else if d<5 then begin
   d := d - 3;
   c := c + 8;
   f := f(d,a) + 2
  end
  else f := 1
end;

begin
   a := 4;
   b := 7;
   c := 9;
   writeln(output, f(b, c))
end.
```

A) 7 B) 6 C) 5

Показивачи

Дефинисање показивача

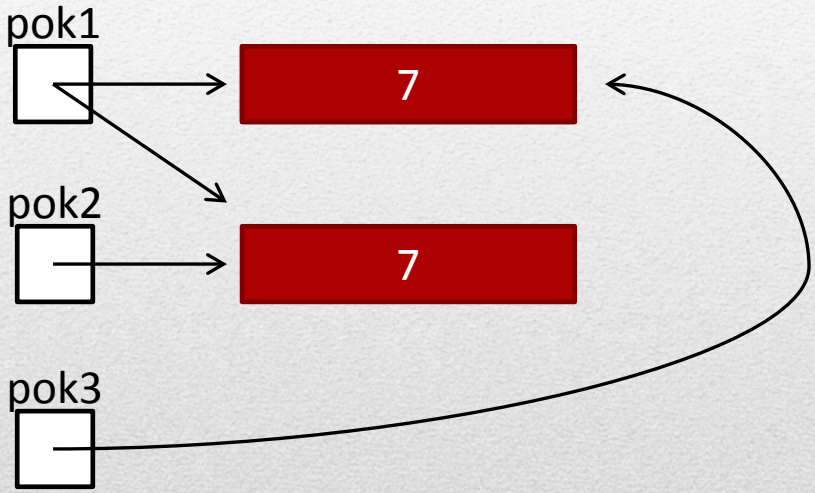
```
VAR i:integer;
    pok:^integer; {pokazivač}
```




```
i:=3;
pok:=3; {GREŠKA: 3 nije pokazivač}
new(pok); {kreira dinamičku promenljivu}
pok^:=22; {pok^ - ono na šta pokazuje pok}
dispose(pok); {briše dinamičku promenljivu}
```

Промена вредности показивача

```
VAR pok1, pok2, pok3: ^integer;

new(pok1);
new(pok2);
pok1^:=3;
pok2^:=7;
pok3:=pok1;
pok1:=pok2;
pok3^:=pok1^;
dispose(pok1);
dispose(pok3);
```

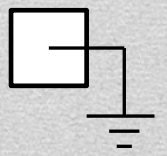


Празан показивач

Ни једна променљива никада не може бити празна. Увек има неку вредност (која понекада није позната).

Када креирамо неки показивач, он није празан него показује на неку недефинисану локацију.

Да бисмо нагласили да показивач не показује нигде стављамо га да буде једнак `NIL` – нула показивач.



Октобар 2012. (5. питалица)

```
program xzy (output);
  type pok = ^integer;
  var x: integer;

procedure p1(x: pok);
begin
  x^ := 13;
end;

procedure p2(var y: integer);
  var p: pok;
begin
  y := 24; new(p); p^ := x + y; p1(p);
  y := p^ + x; dispose(p)
end;

begin
 x := 34; p2(x); writeln(x)
end.
```

A) 37 B) 47 C) 57

Уланчане листе

Шта је листа

Имамо само једну статичку променљиву (показивач на почетак) помоћу које можемо да приступимо неограниченом броју динамичких променљивих.

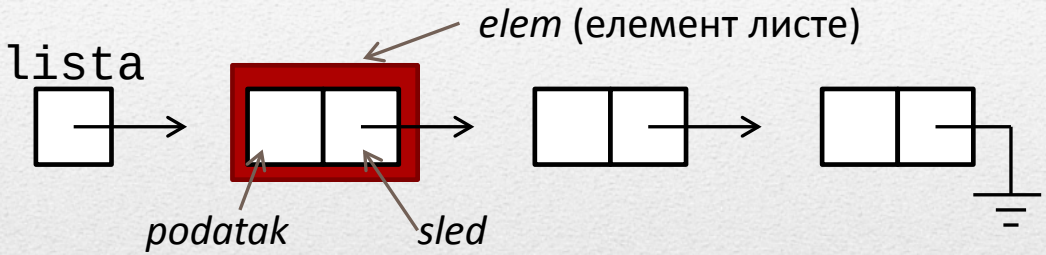
Листа је само „омотач“ за податке. Начин решења проблема не зависи од тога да ли за проблем користимо листу или низ.

За имплементацију листе користимо записе (RECORD). Сваки елемент листе поред податка садржи и показивач на следећи елемент.

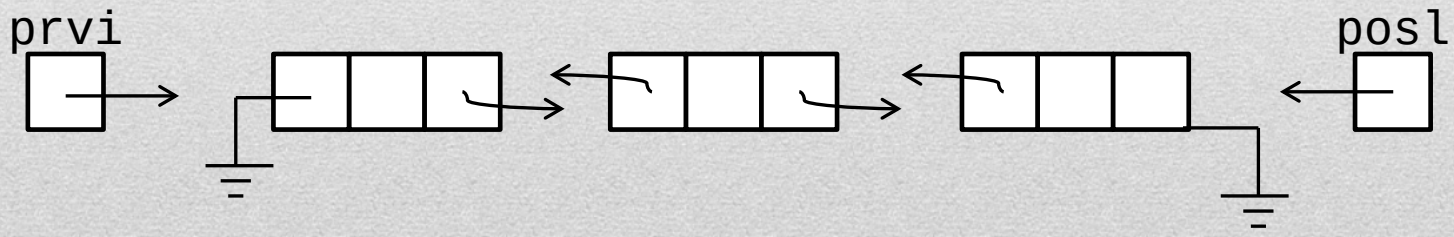
Ефикасније од низа за операције избацивања и уметања, и нема ограничење дужине. За непознат број података боље искоришћење меморије код листе него код низа.

Како листа изгледа

Једноструко уланчана



Двоструко уланчана



Како се дефинише

Једноструко уланчана:

```
TYPE pelem = ^elem;
     elem = RECORD
       podatak: TIP;
       sled: pelem;
     END;
VAR lista: pelem;
```

Двоструко уланчана:

```
TYPE pelem = ^elem;
     elem = RECORD
       podatak: TIP;
       pret, sled: pelem;
     END;
VAR prvi, posl: pelem;
```

Фебруар 2013. (1. питалица)

Која од следеће три процедуре на исправан начин обрже елементе једноструко уланаче листе целих бројева, ако јој се проследи показивач на почетак листе?

A)
```
procedure obrni1(var prvi: pok); var preth, sled, tek: pok;
begin
  tek:= prvi; preth:= nil;
  while (tek <> nil) do begin
    sled:= tek^.sled; tek^.sled:= preth; preth:= tek; tek:= sled;
  end; prvi:= preth;
end;
```

B)
```
procedure obrni2(var prvi: pok); var preth, sled, tek: pok;
begin
  tek:= prvi;  sled:= prvi;
  while (tek^.sled <> nil) do begin
    sled:= tek^.sled; preth^.sled:= tek; preth:= tek; tek:= sled;
  end; prvi:= tek;
end;
```

C)
```
procedure obrni3(var prvi: pok); var preth, tek: pok;
begin
  tek:= prvi; preth:= nil;
  while (tek <> nil) do begin
    tek^.sled:= preth; preth:= tek; tek := tek^.sled;
  end; prvi:= preth;
end;
```

Јануар 2012. (1. питалица)

Шта исписује следећи програм на програмском језику Паскал уколико једноструко уланчана листа садржи редом бројеве 7 8 2 5 4 6? Сматрати да функција `ucitaj` исправно формира листу и враћа показивач на њен почетак, а процедура `pisi` исправно исписује садржај једноструко уланчане листе, на чији почетак показује показивач `glava`.

```
program lista(input, output);
type pok = ^elem;
     elem = record
       broj: integer; sled: pok
     end;
var glava: pok;
procedure obrada (var glava: pok);
var tek : pok;
begin
  tek := glava;
  while (tek^.sled <> nil) do
    tek := tek^.sled
  while (tek^.broj < glava^.broj) do
  begin
    tek^.sled := glava;
    glava := glava^.sled;
    tek := tek^.sled;
    tek^.sled := nil
  end
end;
begin
glava := ucitaj;
obrada(glava);
pisi(glava)
end.
```

A) 2 5 4 6 7 8 B) 6 4 5 2 8 7 C) 6 7 8 2 5 4

Октобар 2011. (3. питалица)

Шта ради процедура `obrada()`, ако јој се проследи показивач на почетак листе целих бројева? Елемент листе садржи цео број у пољу `v` и показивач на следећи елемент у пољу `sled`.

A) Сваки пут када наиђе на секвенцу истих елемената у листи, из секвенце избаци само први елемент.

B) Сваки пут када наиђе на секвенцу истих елемената у листи, из секвенце избаци све елементе осим првог.

C) Сваки пут када наиђе на секвенцу истих елемената у листи, из секвенце избаци само последњи елемент.

```
procedure obrada(var lst: PElement);
var tek, sl: PElement;
begin
  tek := lst;
  if lst <> nil then begin
    sl := lst^.sled;
    while sl <> nil do begin
      if tek^.v = sl^.v then begin
        tek^.sled := sl^.sled;
        dispose(sl);
        sl := tek^.sled
      end
      else begin
        tek := sl;
        sl := tek^.sled
      end
    end
  end
end;
```

Синтаксне нотације

Увод

Синтаксне нотације користе се за описивање синтаксе („правописа“) неког језика (најчешће програмских).

Пример:

```
<indeks>::= <godina>/<broj>
<godina>::= <cifra><cifra>
<broj>::= <cifra><cifra><cifra><cifra>
<cifra>::= 0|1|2|3|4|5|6|7|8|9
```

БНФ нотација

`<neterminal>`

објекат који се описује, служи да сложене описе поделимо на више простих

`terminal`

симбол који се очекује на излазу – азбука синтаксе

оператор `|` значи избор између више опција, на пример `<x>::=a|b` znači da *x* može biti „a“ ili „b“

`{x}` значи да се „*x*“ понавља од 1 до ∞ пута, број понављања се може ограничити на овај начин: $\{x\}_3^{10}$

ЕБНФ нотација

`neterminal`
симбол који се описује

`"terminal"`
симбол који се појављује на излазу (у језику)

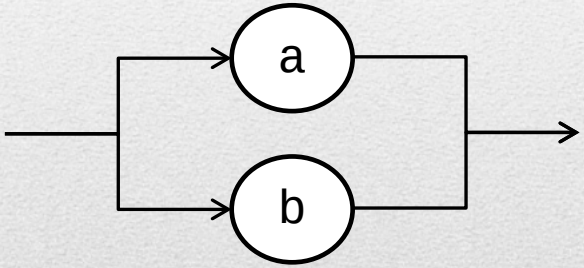
`(a|b)`
избор између *a* и *b*

`{x}`
понављање од 0 до ∞ пута (или се може ограничити: $\{x\}_3^{10}$)

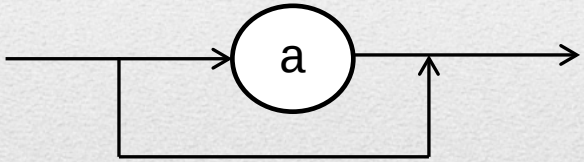
`[x]`
опција – може се појавити али не мора, понављање од 0 до 1

Синтаксни дијаграм

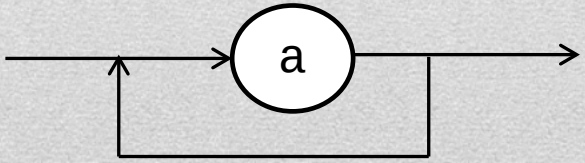
избор



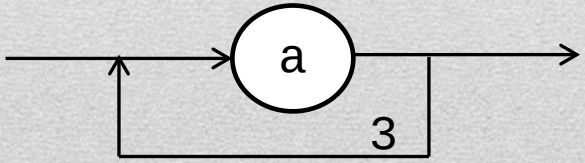
опција



понављање



ограничено понављање



Пример (број мобилног телефона)

БНФ

<telefon>::=+381<pozivni><broj>

<pozivni>::=60|61|62|63|64|65|66

<broj>::={<cifra>}$_6^7$

<cifra>::=0|1|2|3|4|5|6|7|8|9

ЕБНФ

telefon = "+381" pozivni broj.

pozivni = "6"("0"|"1"|"2"|"3"|"4"|"5"|"6").

broj = {cifra}$_6^7$.

cifra = ("0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9").

Октобар 2013. (2. питалица)

Који од понуђених бројева рачуна је формиран у складу са синтаксним правилима за рацун записаним

у ЕБНФ нотацији?

```
racun = cifra [cifra | 0] “-” cifra {cifra | 0} “-” cifra [cifra].
cifra = 1|2|3|4|5|6|7|8|9.
```

A) 200-150000001-40

B) 22-111111120-30

C) 20-140-134

Јануар 2013. (3. питалица)

Који од понуђених израза одговарају синтаксној дефиницији <S>?

<S> ::= x<A>y
<A> ::= <A><B>|xx
<B> ::= y<A>x|y<B>x

A) xxxyxxxyyxxxy
B) xxxyxxyyxxxxxy
C) xxxyxxyxxyxxxy

Сложеност алгоритма

Увод

- Сложеност алгоритма описује колико је времена потребно програму да одради посао у зависности од (броја) података са којима ради.
- Функцију сложености добијамо када пребројимо колико се наредби изврши у програму – веома тешко.
- Ред функције сложености је највећи члан те функције (нпр. ако је функција: $f(n) = 3n^2 - 2n + 3$, онда је ред сложености $O(n^2)$ – квадратна сложеност).
- Не морамо да бројимо све наредбе, интересује нас најсложенији део кода – гледамо петље.

Константна – *O(1)*

```
readln(a,b);
c:=a+b;
writeln(c);
```

Функција сложености:

f(n) = 3

Ред сложености :

O(1)

Линеарна – *O(n)*

```
for i:= 1 to n do
  writeln(i);
```

Функција сложености :

$f(n) = 2n + 1$

Ред сложености :

$O(n)$

Логоритамска – $O(log_2 n)$

```
k:=n;
while k>0 do begin
  writeln(k);
  k:=k div 2;
end;
```

Функција сложености :

$f(n) = 3log_2 n + 5$

Ред сложености :

$O(log_2 n)$

Квадратна – $O(n^2)$

```
for i:= 1 to n do
  for j:= 1 to n do
    writeln(i,j);
```

Функција сложености :

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

Ред сложености:

$O(n^2)$

Одређивање сложености

```
for i:= 1 to n do begin
  k:=n;
  while k>0 do begin
    writeln(k);
    k:=k div 2;
  end;
end;
```

O(n·log(n))

Јануар 2014. (3. питалица)

```
i := 1; k := 1
while i <= n do
begin
   i := i * 2;
   k := k + 1;
end;
r := 1; s := 0;
for i:= 0 to n do
begin
   for j:= 1 to r do
      s := s + j * i;
   r := r * k;
end;
```

A) $n \cdot \log(n)$ B) $n^{\log(n)}$ C) $(\log(n))^n$

Срећно на испиту!

Аутор: Игор Лукић – Змајчики
Приредила: Студентска унија Електротехничког факултета
фебруар 2014.