## Zadanie 1

Napisać efektywną funkcję zwracającą ilość liczb naturalnych mniejszych od n, ] posiadających w swoim rozkładzie wyłącznie czynniki 2,3 i 5.
Przykład: dla n=10 f(10)=7, są to liczby 2,3,4,5,6,8,9.
Wskazówka: rozwiązanie efektywne nie wymaga operacji div ani mod.

## Rozwiązanie zadania 1 (wersja nieefektywna ale za 2.5 ptk)

```
{ czy w rozkladzie liczby n sa tylko 2,3,5 }
function czyn235(n:longint):boolean;
begin
while n mod 2=0 do n:=n div 2;
while n mod 3=0 do n:=n div 3;
while n mod 5=0 do n:=n div 5;
czyn235:=n=1
end;

{ ile liczb mniejszych od n }
function ile(n:longint):longint;
var
i,l : longint;
begin
l:=0;
for i:=2 to n-1 do
if czyn235(i) then inc(l);
ile:=l
end;

begin
writeln(ile(100000000)) { przykładowe wywolanie }
end.
```

## Rozwiązanie zadania 1 (wersja efektywna)

```
{ zlicza wielokrotnosci liczby l ograniczone przez n }
function ala(l,p,n:longint):longint;
begin
p:=p*l;
if p>=n then ala:=0 else
if l=5 then ala:=1+ala(5,p,n) else
if l=3 then ala:=1+ala(3,p,n)+ala(5,p,n) else
if l=2 then ala:=1+ala(2,p,n)+ala(3,p,n)+ala(5,p,n)
end;

{ ile liczb mniejszych od n }
function ile(n:longint):longint;
begin
ile:=ala(2,1,n)+ala(3,1,n)+ala(5,1,n)
end;

begin
writeln(ile(100000000)) { przykladowe wywolanie }
end.
```

Proszę zauważyć, że w rozwiązaniu, pomiędzy instrukcjami występuje zaledwie JEDEN średnik!

Aby zrozumieć co to znaczy rozwiązanie efektywne proszę uruchmić oba programy.

## Zadanie 2

Napisać procedurę, która dla danego zbioru znaków wypisuje wszystkie jego podzbiory zawierające co najmniej jedną literę. Zbiór znaków przekazywany jest do procedury jako napis. Można założyć, że napis zawiera niepowtarzające

znaki będące literami z zakresu a-z (bez polskich znaków) bądź cyframi 0-9 oraz w napisie występuje co najmniej jedna litera.
Przykład: dla danych 'a2b' program powinien wypisać zbiory: a,b,ab,a2,b2,ab2.

## Rozwiązanie zadania 2

```
{ parametry procedury wypisz:
s - wejsciowy zbior znakow
w - wynikowy zbior znakow
n - kolejny znak ze zbioru
b - czy w zbiorze wynikowym jest litera }


procedure wypisz(s,w:string; n:integer; b:boolean);
begin
if n<=length(s) then begin
wypisz(s,w,n+1,b); { bez n-tego znaku }
wypisz(s,w+s[n],n+1,b or (s[n]>='a')) { z n-tym znakiem }
end else if b then writeln(w)
end;

begin
wypisz('a12b','',1,false) { przykladowe wywolanie }
end.
```

Tu także w rozwiązaniu, pomiędzy instrukcjami występuje zaledwie JEDEN średnik!

## Zadanie 3

---------

Dane są dwie uporządkowane listy zawierające niepowtarzające się liczby naturalne. Proszę napisać funkcję scalającą dwie listy tak, aby scalona lista zawierała niepowtarzające się elementy występujące w jednej lub drugiej liście. Funkcja powinna zwrócić wskaźnik do scalonej listy.

Rozwiązanie zadania 3
---------------------

```
type
PNode = ^TNode;
TNode =
record
Val: Integer;
Next: PNode
end;

function Merge( var p1, p2: PNode ): PNode;
var
Tail, q: PNode;
begin
Tail := nil;
while (p1 <> nil) and (p2 <> nil) do
begin
if p1^.Val < p2^.Val then
begin
q := p1;
p1 := p1^.Next
end
else if p1^.Val > p2^.Val then
begin
q := p2;
p2 := p2^.Next
end
else { p1.Val = p2.Val }
begin
q := p1;
p1 := p1^.Next;
Dispose( q );
q := p2;
p2 := p2^.Next
end;
if Tail = nil then { pierwszy element }
Merge := q
else
Tail^.Next := q;
Tail := q;
Tail^.Next := nil
end;
{ (p1 = nil) or (p2 = nil) }
if p1 <> nil then { p2 = nil }
begin
if Tail = nil then
Merge := p1
else
Tail^.Next := p1
end
else if p2 <> nil then { p1 = nil }
begin
if Tail = nil then
Merge := p2
else
Tail^.Next := p2
end
else { (p1 = nil) and (p2 = nil) }
if( Tail = nil ) then
Merge := nil
end;
```

Zadanie 4
---------

Tablicę typu tab=array[1..100,1..100] of Integer, zawierającą liczby naturalne podzielono na 100 pól o rozmiarze 10 na 10. Proszę napisać funkcję, która dla tablicy typu tab zwraca liczbę pól, w których większość stanowią liczby pierwsze.

Rozwiązanie zadania 4
---------------------

```
program Zadanie4;

type
Index = 1..100;
Tab = array[Index, Index] of Integer;

function IsPrime( x: Integer ): Boolean;
var
q: Boolean;
i, xsq: Integer;
begin
q := (x = 2) or ((x > 2) and Odd(x));
if q then
begin
i := 3;
xsq := Trunc(Sqrt(x));
while( q and (i <= xsq) ) do
if x mod i = 0 then q := False else i := i + 2
end;
IsPrime := q
end;

function IsFieldOverPrimed( var a: Tab; x, y: Index ): Boolean;
var
i, j: Index;
Count: 0..100;
Remaining: 0..100;
begin
Count := 0;
Remaining := 100;
i := x;
while (i <= x+10) and (Count+Remaining > 50) do
begin
j := y;
while (j <= y+10) and (Count+Remaining > 50) do
```

```
begin
if IsPrime( a[i,j] ) then Count := Count
+ 1;
Remaining := Remaining - 1;
j := j+1;
end;
i := i+1
end;
IsFieldOverPrimed := Count > 50
end;

function GetOverPrimedFieldCount( var a:
Tab ): Integer;
var
i, j: 0..9;
Count: Integer;
begin
Count := 0;
for i := 0 to 9 do
for j := 0 to 9 do
if IsFieldOverPrimed( a, 10*i+1,
10*j+1 ) then
Count := Count + 1;
GetOverPrimedFieldCount := Count
end;

var i, j: Index;

var a: Tab;

begin
Randomize;
for i:=1 to 100 do
for j:=1 to 100 do
a[i,j] := 1+Random(4);

Writeln( GetOverPrimedFieldCount( a ) )
end.
```

Zadanie 5
---------

Proszę napisać definicje i deklaracje odpowiednich typów, zmiennych, funkcji lub procedur tak, aby poniższe odwołania, jeżeli to możliwe, były poprawne
w Pascalu. Jeśli odwołania są niepoprawne, proszę uzasadnić, dlaczego.

```
(a) a[pred('x')][ord(false)]-a[b][c]
(b) a[b.c.d]=a[c.d].b
(c) a^.b^.c^.d^.e^.f^.g^.h^
(d) a^^['b'][c-d]
(e) a := b['w',b['x','y']]-succ('z')
(f) a^[b(c^)^]^
```

Rozwiązanie zadania 5
---------------------

```
var
Foo: Byte;

{a}
var
a1: array[Char, 0..1] of Byte;
b1: Char;
c1: 0..1;

{b --- a[b.c..d] = a[c.d].b }
{
Niemozliwe, poniewaz rekord nie moze miec
pola tego samego typu, co on sam.
}

{c}
type
PRec1 = ^TRec1;
TRec1 =
record
b, c, d, e, f, g, h: PRec1
end;

var
a2: PRec1;

{d}
type
TTab4 = array['b'..'b',Byte] of Byte;
PTab4 = ^TTab4;

var
a4: ^PTab4;
c4, d4: Byte;

{e --- a := b['w',b['x','y']]-Succ('z')]}

{
Niemozliwe, bo w Pascalu nie mozna
odejmowac wartosci typu Char,
a typem elementu b musi byc Char, gdyz
jest uzyty do indeksowania b,
a oba indeksy b musza byc typu Char, co
wynika z fragmentu wyrazenia
b['x','y]
}

{f}
type
TTab6 = array[Byte] of ^Byte;
PByte = ^Byte;

var
c6: PByte;
a6: ^TTab6;

function b6( Bar: Byte ): PByte;
begin
end;

begin
{a} Foo := a1[Pred('x')][Ord(False)] -
a1[b1][c1];
{b --- a[b.c..d] = a[c.d].b }
{c} a2^ := a2^.b^.c^.d^.e^.f^.g^.h^;
{d} Foo := a4^^['b'][c4-d4];
{e --- a := b['w',b['x','y']]-Succ('z')]}

{f} Foo := a6^[b6(c6^)^]^;
end.
```

----------------------------------------
---------------------------------------