

Wstęp do programowania

Marek A. Bednarczyk, www.ipipan.gda.pl

Literatura

David Harel. **Rzecz o istocie informatyki. Algorytmika.**

Wydawnictwa Naukowo-Techniczne. Wydanie trzecie. Seria: Klasyka informatyki. Warszawa 2000.

Niklaus Wirth. **Algorytmy + Struktury Danych = Programy.**

WNT. Seria: Klasyka informatyki. Warszawa 2000. (Istnieje wiele wydań wcześniejszych, pierwsze: WNT, 1980)

Programowanie = sztuka czy inżynieria?

- Czym jest programowanie – sztuka czy inżynieria ?
- Specyfikowanie/prezentowanie algorytmów - diagramy blokowe.
- Poprawność, dokumentowanie i uaktualnienie oprogramowania
- Spektakularne wpadki, ciche sukcesy - klęska informatyki teoretycznej?

Programowanie kontra kodowanie

Cykl programistyczny:

begin

pocz : analiza zjawiska/procesu

projekt modelu abstrakcyjnego zjawiska/procesu

weryfikacja modelu if błąd go to pocz

prog : model programistyczny zjawiska/procesu

kodowanie

testowanie kodu if błąd go to prog

go to pocz

end

Wykład : model programistyczny + kodowanie

- prezentacja algorytmu w postaci schematu blokowego
- specyfikowanie algorytmu
- poprawność algorytmu
- poprawność kodowania
- dokumentowanie
- bieżąca aktualizacja

Cechy dobrego informatyka

Idealny **programista/projektant/analitik** powinien

- znać język angielski — przychodzi samo
- umieć myśleć analitycznie
- nieustannie uczyć się nowych rzeczy
- podpatrywać lepszych programistów

Idealny **projektant/analitik** powinien winien dodatkowo

- poszerzać wiedzę z informatyzowanej dziedziny

Architektura komputerów w 3 minuty

- binarna reprezentacja liczb: *10110000 01100001*
- binarna reprezentacja ciągów
- binarna reprezentacja tablic
- binarna reprezentacja rekordów
- binarna reprezentacja programów *10110000 01100001*
czyli *mov al, 0x61*

Konsekwencje binarnej reprezentacji

PROBLEM skończoności

Konflikt między ograniczonością reprezentacji a nieograniczonnością świata

PROBLEM Binarnej Bariery

Rozdzwięk między ludzkim a maszynowym sposobem reprezentacji danych i programów

Ewolucja — uniwersalne języki programowania

Rozwój języków programowania próbą pokonania **binarnej bariery**

- języki maszynowe: IA32, AMD64, IA64
- asemblery
- fortran — **skoki, tablice i pętla for**
- COBOL — język dla biznesu
- Algol60 — **typy obietów, procedury i funkcje**
- Pascal /Modula2 — rekordy, typy definiowalne
- java — **programowanie obiektowe, zdarzenia i wyjątki**

Ewolucja — specjalizowane języki programowania

- Języki przetwarzania symbolicznego (AI, logika)
 - języki manipulowania tekstami (SNOBOL)
 - LISP — języki funkcyjne (Scheme, ML, Haskell)
 - Prolog — programowanie logiczne
- Języki sterowania systemami czasu rzeczywistego
- Języki projektowania systemów współbieżnych

Skutki ewolucji

- Mniejszy dystans między opisem a kodem
- Łatwiejsze poprawianie, konserwacja i aktualizacja
- Zwiększenie wydajności programowania (**reusability**)
- Języki specjalizowane do specyficznych dziedzin zastosowań

Składnia a semantyka, czyli przepis kontra funkcja

Precyzyjna składnia = łatwo o błędy!

Kłopoty z semantyką

- niejednoznaczność i/lub braki w definicji języka
- różnice między implementacjami tego samego języka w różnych środowiskach (np.: Linux i win XP)
- problemy fundamentalnej natury.

Semantyka operacyjna kontra semantyka denotacyjna.

Problemy z częściowością obliczeń.

Kompilacja a interpretacja

Jak wykonać **program** napisany w języku programowania na komputerze?

Kompilacja

- Uruchamiamy na **kompilator** podając **program** jako dane.
- Otrzymujemy **program wykonywalny**
- Uruchamiamy **program wykonywalny** celem wykonania.

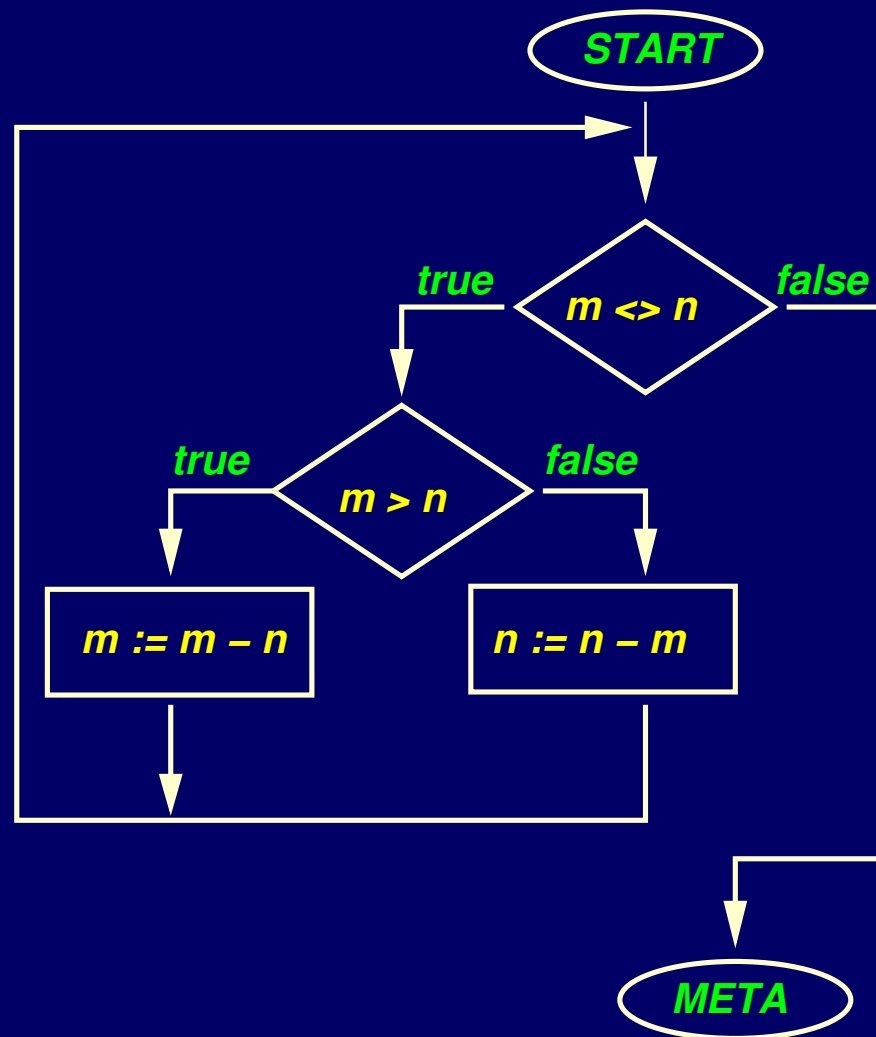
Iterpretacja

- Uruchamiamy na **interpreter** podając **program** jako dane.
- **Interpreter** wykonuje instrukcje **programu** i kończy działanie

Pojęcie algorytmu

- Specyfikacja **Spec**
- Prezentacja **prog**, np. w postaci **schematu blokowego**
- **Poprawność** — czy program **prog** spełnia specyfikację **Spec** ?
- **Poprawność częściowa**
- **Własność stopu**
- **Poprawność całkowita**
 $\text{całkowita} = \text{częściowa} + \text{Własność stopu}$

Schemat blokowy



Pascal — Składnia i semantyka

Składnia przedstawiona w postaci reguł gramatycznych.

Semantyka operacyjna instrukcji podana w postaci schematów blokowych.

Składnia – najprostszy program w Pascalu

```
program Euklides ;
```

```
begin
```

```
end.
```

Instrukcje

$\langle \text{Inst} \rangle ::=$ $\langle \text{Inst_przyp} \rangle$ |
 $\langle \text{Inst_bloku} \rangle$ |
 $\langle \text{Inst_if_then_else} \rangle$ |
 $\langle \text{Inst_if_then} \rangle$ |
 $\langle \text{Inst_while} \rangle$ |
 $\langle \text{Inst_repeat} \rangle$ |
 $\langle \text{Inst_for} \rangle$ |
...

Instrukcja przypisania

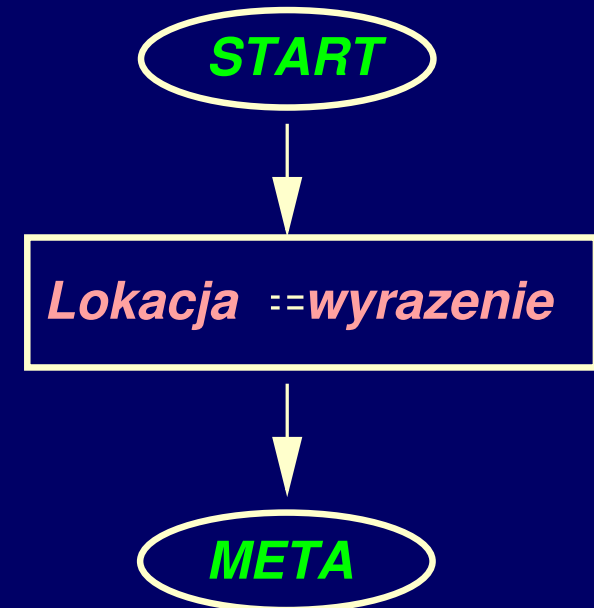
$\langle \text{Inst_przyp} \rangle ::=$
 $\langle \text{Lokacja} \rangle := \langle \text{Wyrażenie} \rangle$

Na przykład:

$\text{pom} := \text{pom} - 15$

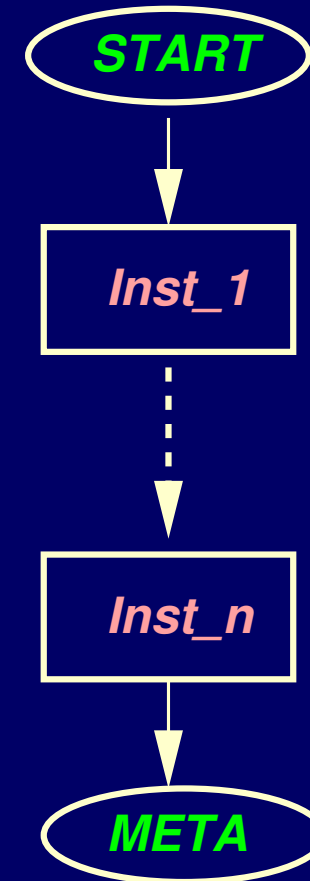
$\text{tab}[i + 3] := \text{tab}[i]$

$\text{rekord.pole} := 3.1459$



Ciąg instrukcji a instrukcja bloku

$\langle \text{Ciąg_Inst} \rangle ::=$
 $\langle \rangle$ |
 $\langle \text{Inst} \rangle$ |
 $\langle \text{Inst} \rangle ; \langle \text{Ciąg_Inst} \rangle$



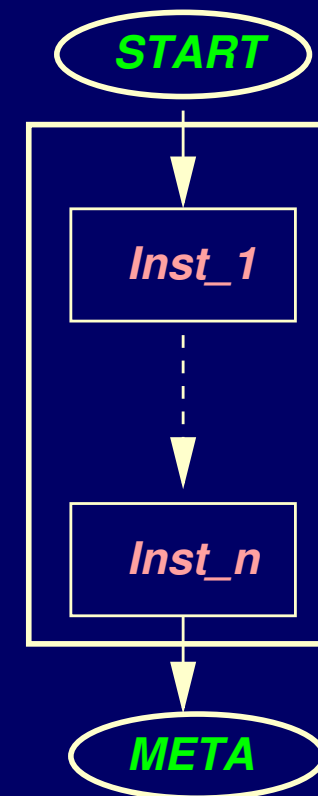
Ciąg instrukcji a instrukcja bloku

$\langle \text{Inst_bloku} \rangle ::=$

begin

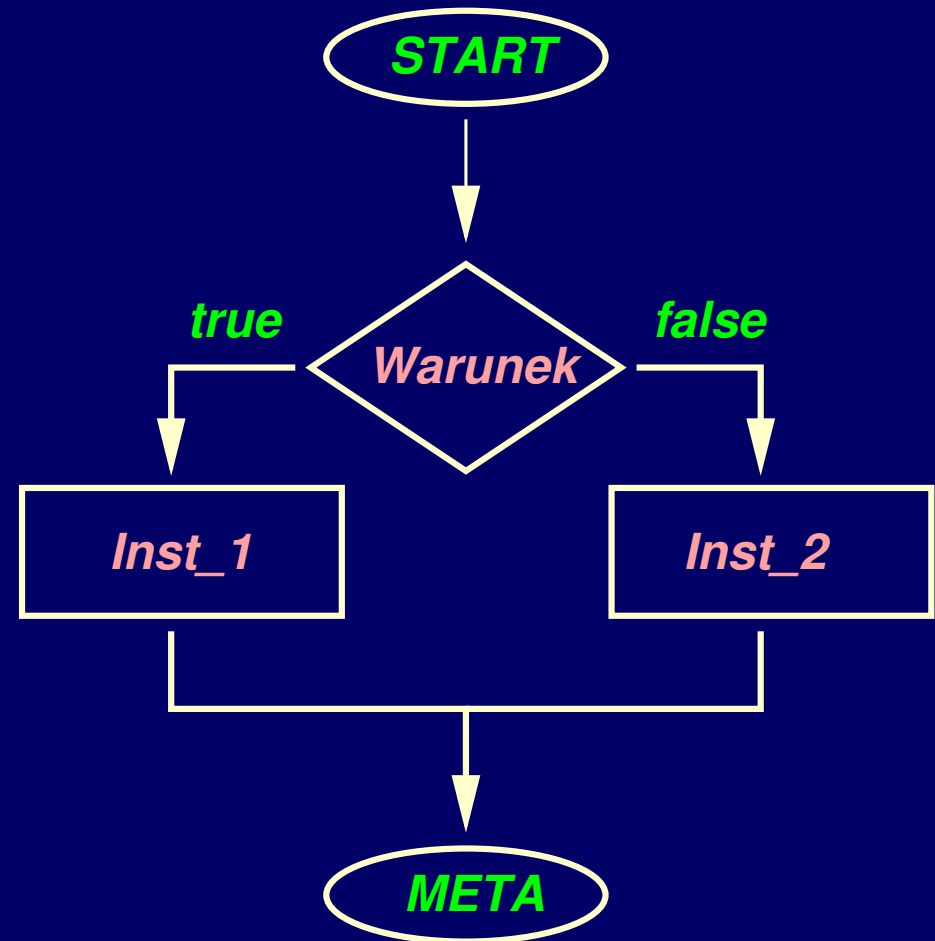
$\langle \text{Ciąg_Inst} \rangle$

end



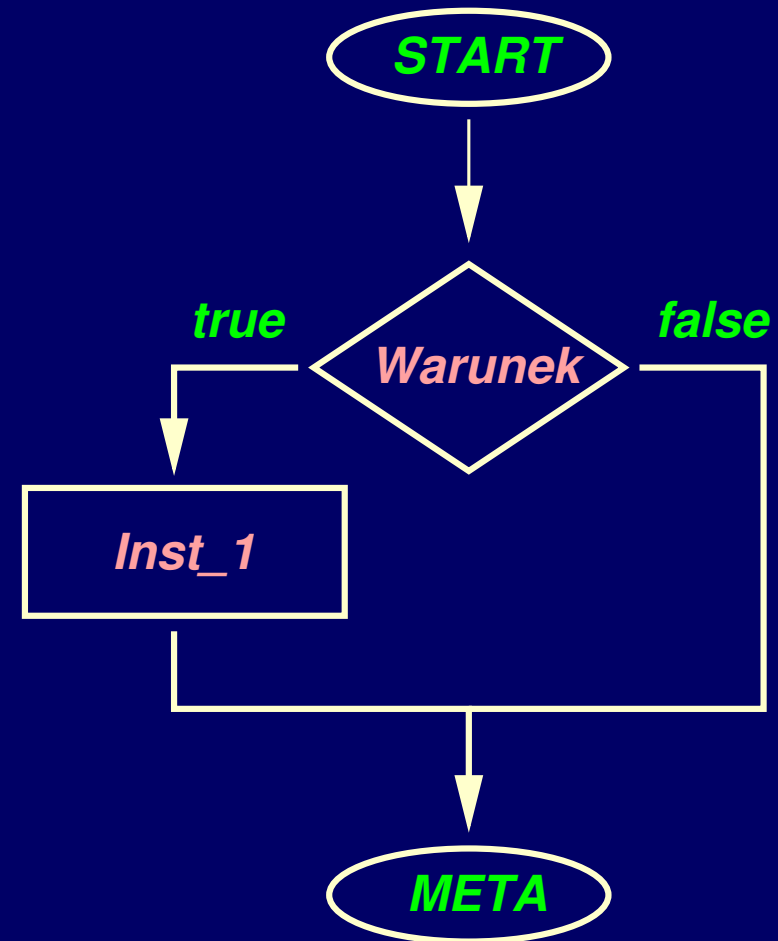
Instrukcja warunkowa if-then-else

<Inst_if_then_else> ::=
if <Warunek> then
 <Inst_1>
else <Inst_2>



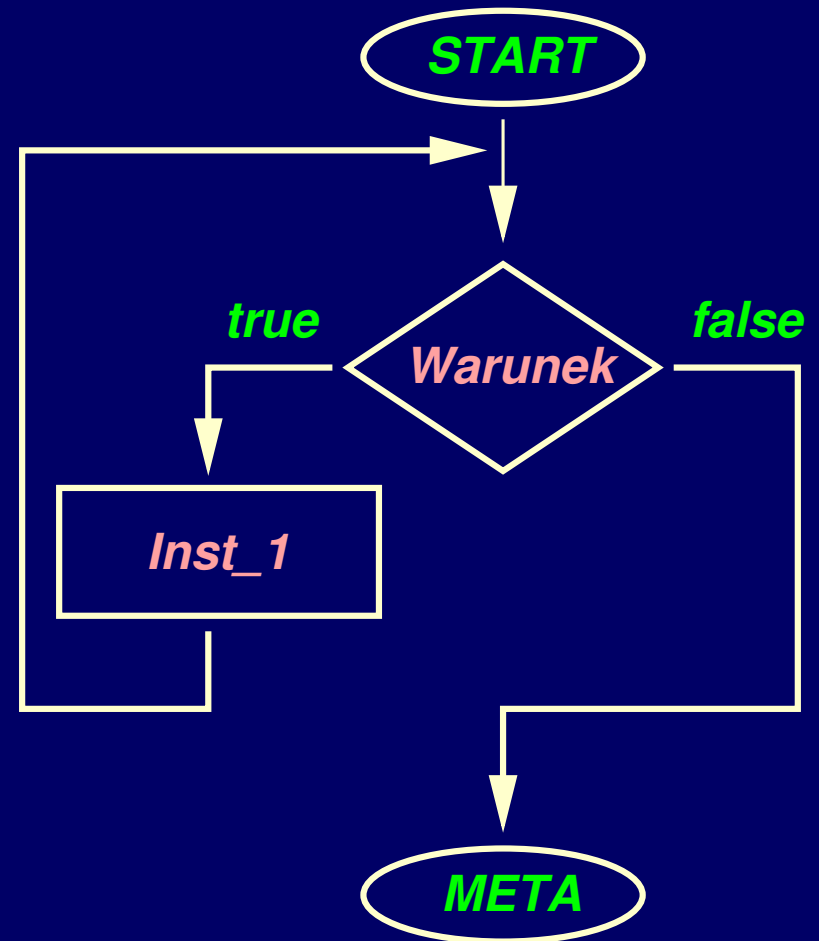
Instrukcja warunkowa if-then

<Inst_if_then> ::=
if <Warunek>
then <Inst_1>



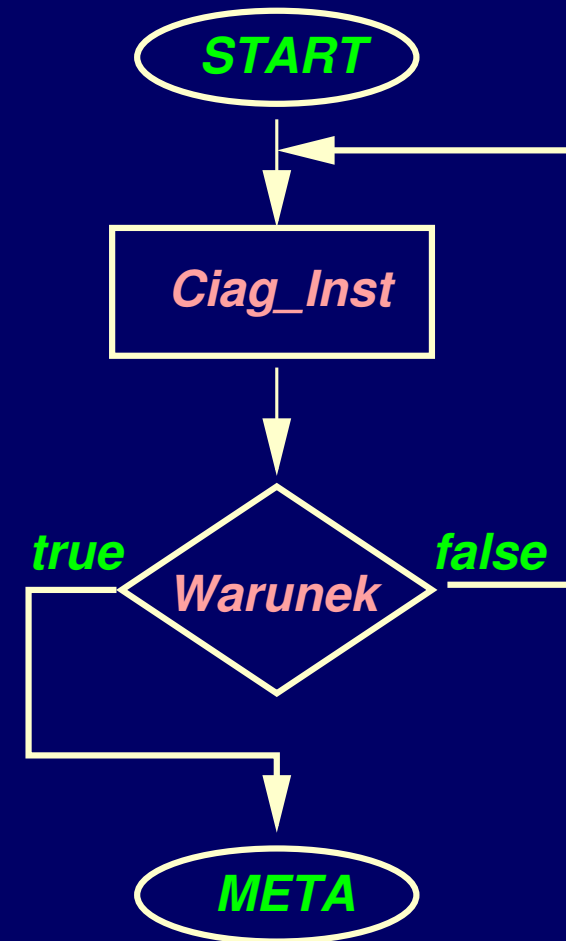
Instrukcja pętli **while-do**

<Inst_while> ::=
while <Warunek> do
<Inst_1>

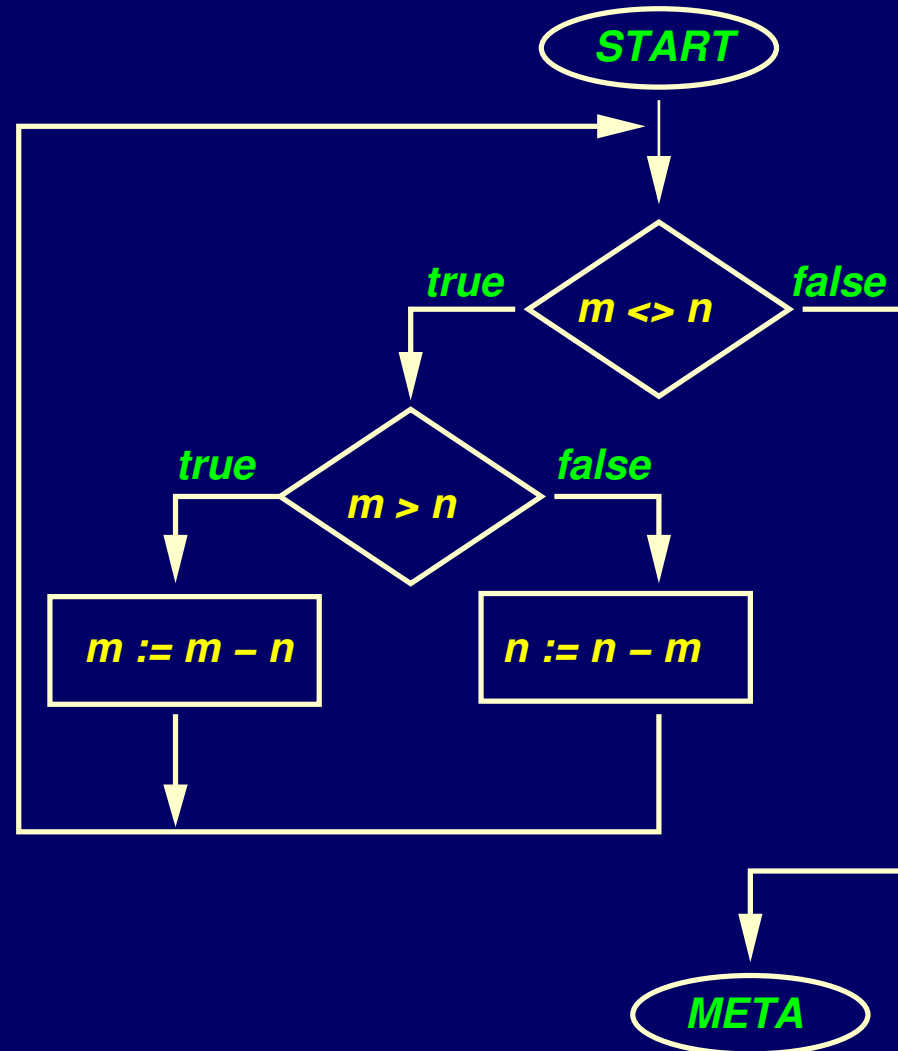


Instrukcja pętli **repeat-until**

<Inst_repeat> ::=
repeat
 <Ciag_Inst>
until <Warunek> do

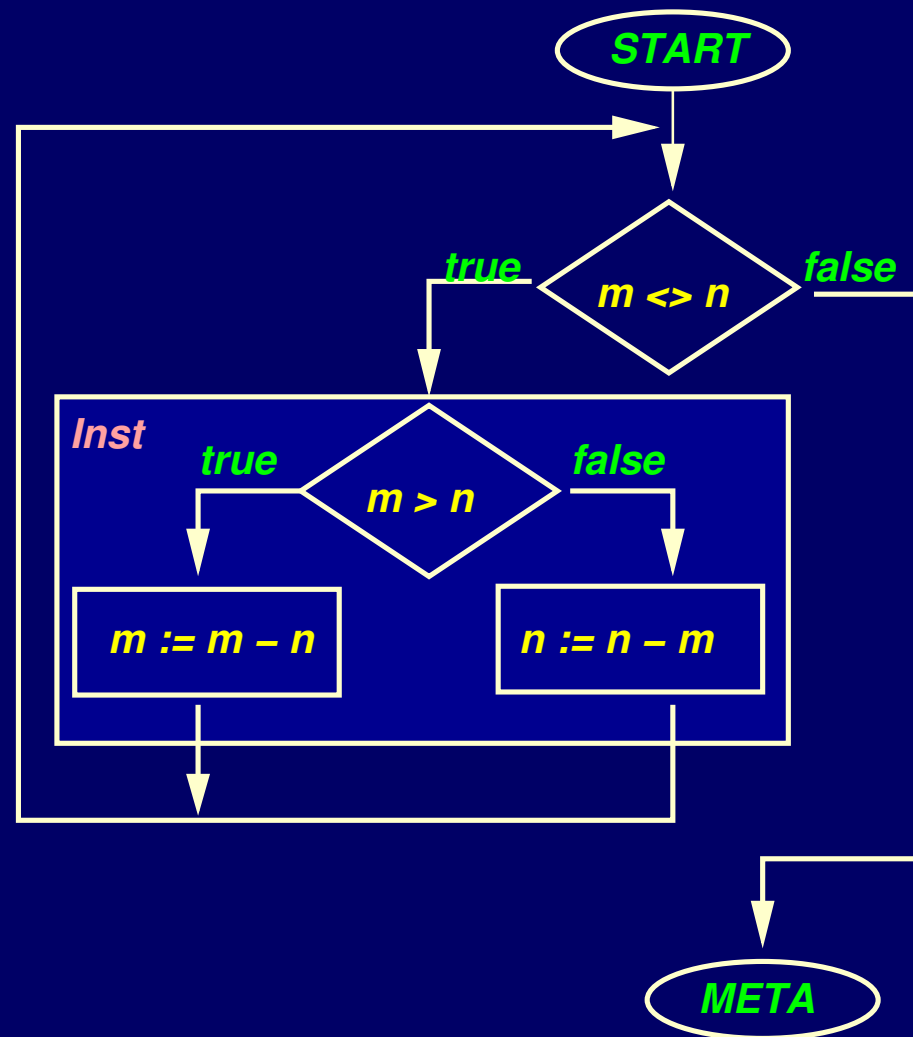


Algorytm Euklidesa



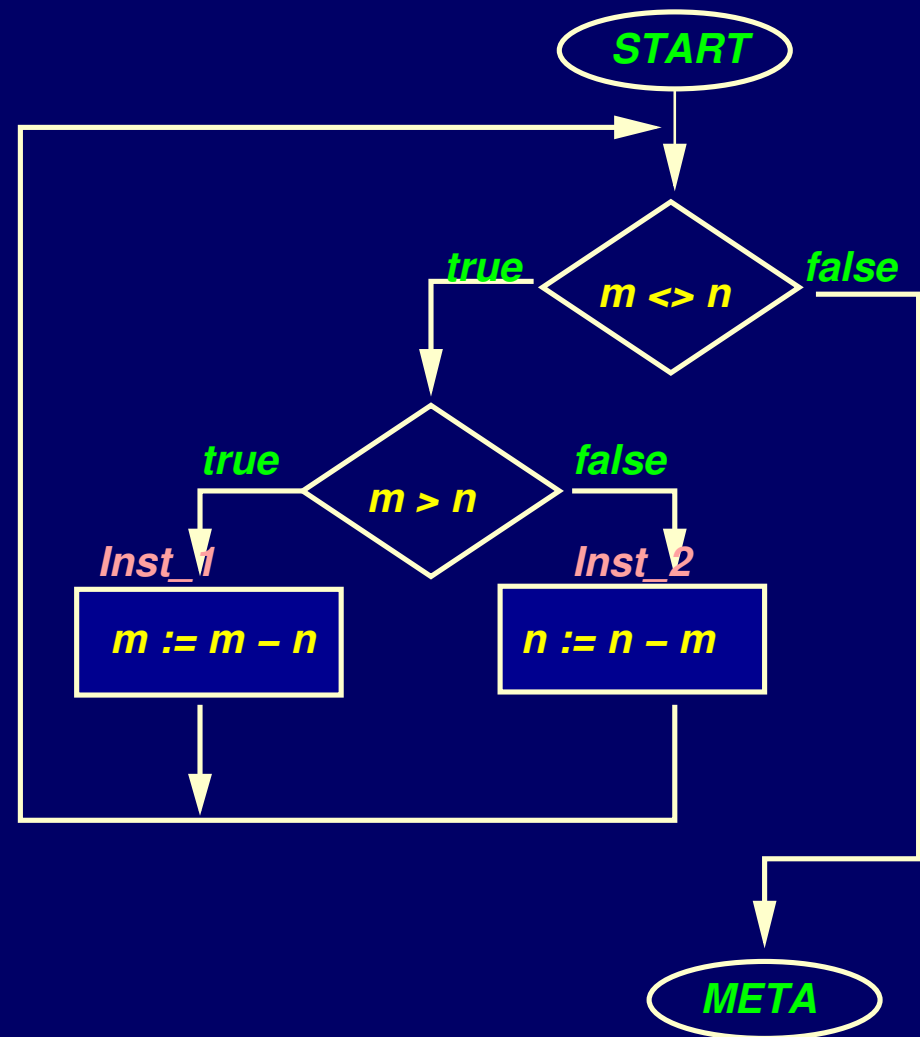
Algorytm Euklidesa

while $m \neq n$ do
 $\langle Inst \rangle$



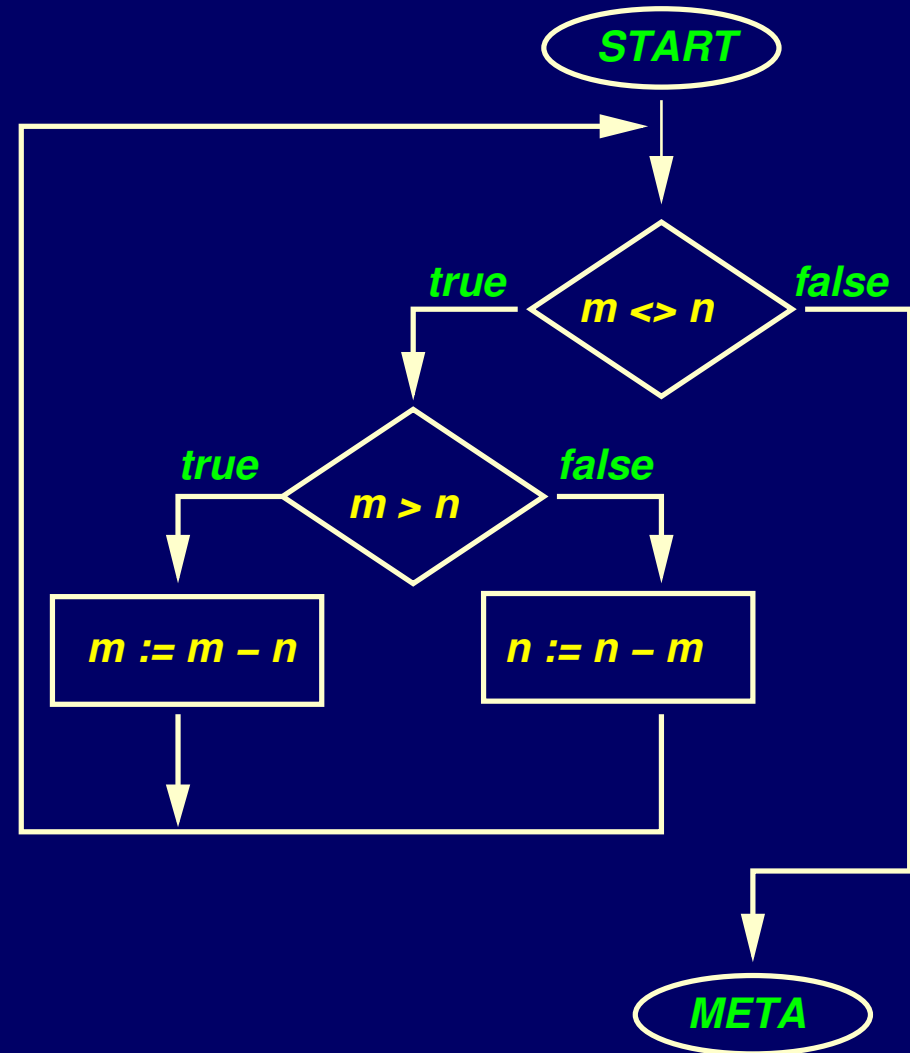
Algorytm Euklidesa

```
while  $m \neq n$  do  
  if  $m > n$  then  
     $\langle \text{Inst}_1 \rangle$   
  else  $\langle \text{Inst}_2 \rangle$ 
```

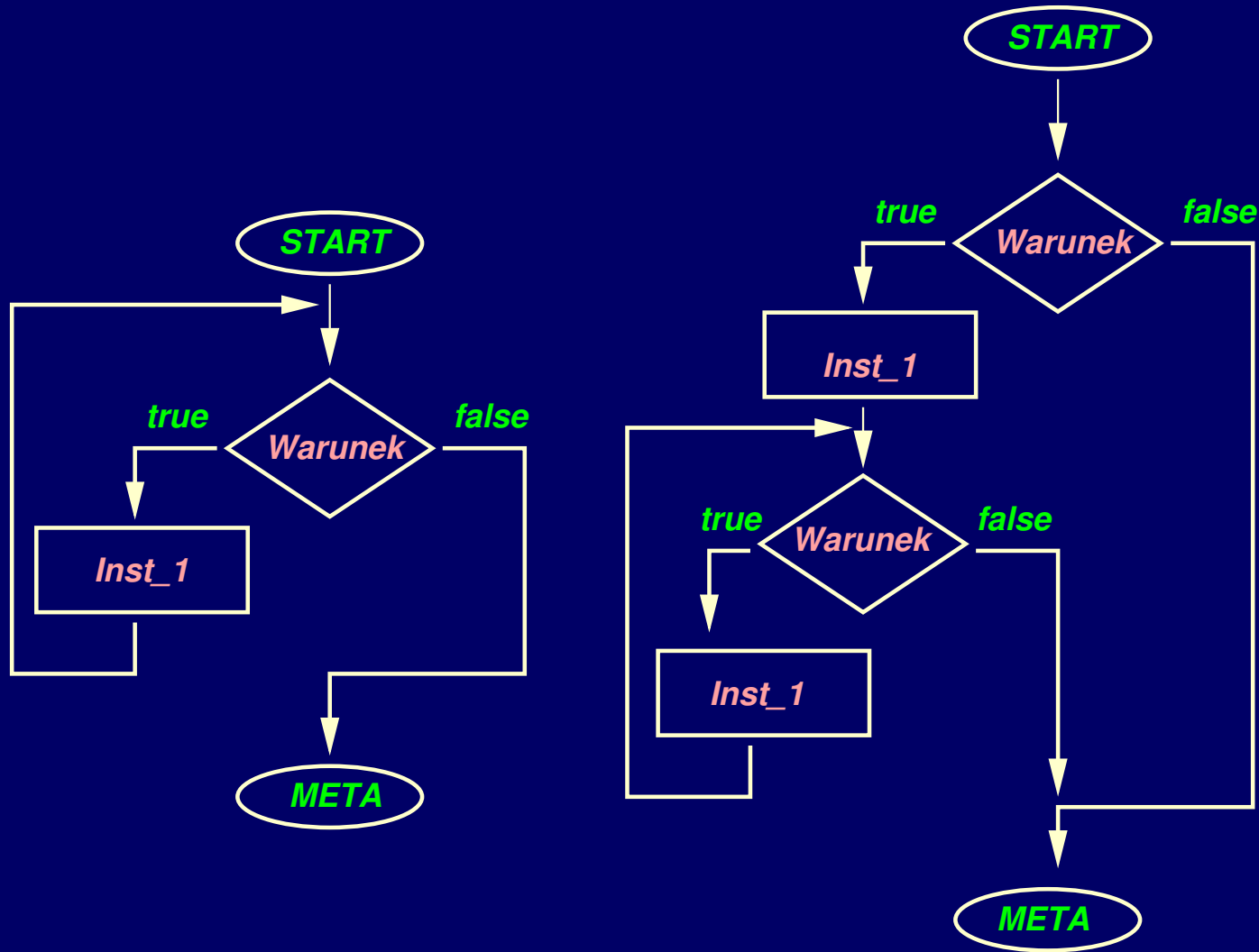


Algorytm Euklidesa

```
while  $m \neq n$  do  
  if  $m > n$  then  
     $m := m - n$   
  else  $n := n - m$ 
```



Pętla while-do a if



Pętla while-do a if

```
while <warunek> do  
  <Inst_1>
```

jest równoważne

```
if <warunek> then  
  begin  
    <Inst_1> ;  
    while <warunek> do  
      <Inst_1>  
  end
```

Specyfikowanie programów imperatywnych

S, S' : **Stan** — wartościowanie **zmiennych**.

Π : **program** — relacja opisująca zmianę **stanu**. Notacja:

$S - \Pi \rightarrow S'$ znaczy Π uruchomiony w S skończył w stanie S'

Φ, Ψ : **asercja** — własność logiczna opisująca stan.

Prewarunek — własność opisująca stany dla których chcemy by program działał porawnie.

Postwarunek — własność stanów uzyskanych po zakończeniu działania programu.

Specyfikowanie programów imperatywnych (cd)

Częściowa poprawność programu Π względem prewarunku Φ i postwarunku Ψ .

$S \models \Phi$ i $S - \Pi \rightarrow S'$ implikuje $S' \models \Psi$

Własność stopu programu Π względem prewarunku Φ .

$S \models \Phi$ implikuje Ψ uruchomiony w S kończy działanie

Algorytm Euklidesa — specyfikacja

Prewarunek — $M > 0, N > 0$

Postwarunek — *wynik* = $NWD(M, N)$

Prosta realizacja

wynik:=NWD(M,N)

Schemat blokowy

Algorytm Euklidesa — poprawność

Lemat 1 Jeśli $p = d > 0$ to $NWD(p, d) = p = d$

Lemat 2 Jeśli $p > d > 0$ to $NWD(p, d) = NWD(p - d, d)$

Lemat 3 Jeśli $0 < p < d$ to $NWD(p, d) = NWD(p, d - p)$

Tw 1 Algorytm Euklidesa jest częściowo poprawny względem $\Phi : M > 0 \wedge N > 0$, oraz $\Psi : \text{wynik} = NWD(M, N)$.

Tw 2 Alg. Euklidesa ma własność stopu wzgl. Φ .

Wniosek Alg. Euklidesa jest całkowicie poprawny wzgl. Φ i Ψ .

Pascal — uniwersalny język programowania

Dlaczego **Free Pascal** a nie na przykład **java**, **C** czy **C++**?

- elegancki
- prosty koncepcyjnie, w użyciu, w opanowaniu,
- Free Pascal dostępny za darmo,
- Dostępny na wiele platform: MacOS, BeOS, FreeBSD, OS/2, Linux (x86 i Motorola 68k), Amiga, Solaris, netBSD, QNX, Win32 (od 95 do NT)
- Rozwijany — wyjątki, obiekty i klasy, ...

strona domowa: www.freepascal.org

dokumentacja on-line: www.freepascal.org/docs.html

Algorytm Euklidesa w Pascalu

program **Euklides** ;

{ wczytuje liczby naturalne m i n. Jesli dodatnie, liczy
NWD(m,n). Działa poprawnie dla liczb w dziedzinie typu
INTEGER. Np. daje wynik 1 dla m=66666 i n = 66777 gdy
NWD(m,n)=3 }

var **m, pie, n, dru, wynik** : integer;

begin

<ciało programu>

end.

Uwaga!

Dokumentacja (komentarze) i ładna edycja ważne dla
programisty!

```
writeln('Podaj dwie liczby naturalne');
readln (m,n);
if (m>0) and (n>0) then
  begin
    pie := m; dru := n;
    while pie <> dru do
      if pie > dru then
        pie := pie - dru
      else dru := dru - pie ;
    wynik := pie;
    writeln('NWD(',m,',',n,')= ',wynik)
  end
else  writeln('niewłaściwe dane')
```

Struktury danych a mechanizmy kontroli działania

Wirth: **Algorytmy** + **struktury danych** = **programy**

- **pęta for** — **tablice**
- **rekursja** — **drzewa**
- **wyjątki** — **zdarzenia**
- **komunikaty/metody** — **obiekty**

Drukowanie prostokąta znaków '*'

Zadanie:

Napisz program, który wczytuje dwie liczby: **k** i **w**, a następnie wypisuje **w**-wierszy, po **k**-znaków '*' w każdym wierszu.

Zadanie pomocnicze:

Napisz program, który wczytuje liczbę: **k**, a następnie wypisuje **k**-znaków '*' w jednym wierszu.

Drukowanie wiersza znaków '*'

```
program druk_wiersza ;  
  { Wczytuje liczbę k i drukuje wiersz, złożony z k-znaków '*' }  
  var k, ind_k : integer;  
begin  
  write('Kolumn:'); readln(k);  
  for ind_k := 1 to k do  
    write('*');  
  writeln  
end.
```

Drukowanie prostokąta znaków '*'

```
program druk_prostokata ;
  { Wczytuje liczby k i w, drukuje w-wierszy, po k-znaków '*' }
  var k, ind_k : integer;    w, ind_w : integer;
begin
  write('Wierszy:'); readln(w); write('Kolumn:'); readln(k);
  for ind_w := 1 to w do
    begin
      for ind_k := 1 to k do
        write('*');
      writeln
    end
end.
```

Drukowanie prostokąta znaków '*' — z procedurą

```
program druk_prostokata ;  
  { Wczytuje liczby k i w, drukuje w-wierszy, po k-znaków '*' }  
  var k : integer;  
      w, ind_w : integer;  
begin  
  write('Wierszy:'); readln(w);  
  write('Kolumn:'); readln(k);  
  for ind_w := 1 to w do  
    druk_wiersza(k)  
end.
```

Procedura drukowanie wiersza k-znaków '*'

```
procedure druk_wiersza (k : integer);  
  { Drukuje wiersz złożony z k-znaków '*' }  
  var ind_k : integer;  
begin  
  for ind_k := 1 to k do  
    write('*');  
  writeln  
end;
```

Procedura drukowanie wiersza **k**-znaków **ch**

```
procedure druk_wiersza (k : integer, ch : char );  
  { Drukuje wiersz, złożony z k-znaków ch }  
  var ind_k : integer;  
begin  
  for ind_k := 1 to k do  
    write(ch);  
  writeln  
end;
```

Drukowanie prostokąta dowolnych znaków — z procedurą

```
program druk_prostokata ;  
  { Wczytuje liczby k i w oraz znak z i drukuje w-wierszy po  
  k-znaków z}  
  var k : integer;    w, ind_w : integer; var z : char;  
  procedure druk_wiersza (k:integer , ch:char);  
begin  
  write('Wierszy:'); readln(w); write('Kolumn:'); readln(k);  
  write('Znak:'); readln(z);  
  for ind_w := 1 to w do  
    druk_wiersza(k, znak)  
end.
```

3 wieże — przykład **trudnego** problemu

Zadanie algorytmiczne — poszukiwanie przepisu na osiągnięcie celu

Przykład celu: Wieże z Hanoi.

Dane są trzy pale: **X**, **Y** i **Z**. Na jeden z nich, np. na **X** nałożono krążki o różnych średnicach, tak że krążek mniejszy nie leży pod większym. Pale **Y** i **Z** są puste. Należy przenieść wszystkie krążki z pala **X** na **Z**. Można używać **Y** jako pomocnicze miejsce przechowywania. Nie można kłaść krążków większych na mniejsze.

Wieże z Hanoi

Stan wyjściowy

oo X oo	Y	Z
ooo X ooo	Y	Z
oooo X oooo	Y	Z

Stan docelowy

X	Y	oo Z oo
X	Y	ooo Z ooo
X	Y	oooo Z oooo

Wieże z Hanoi

Stan wyjściowy

oo X oo	Y	Z
ooo X ooo	Y	Z
oooo X oooo	Y	Z

Pierwszy ruch

X	Y	Z
ooo X ooo	Y	Z
oooo X oooo	Y	oo Z oo

Wieże z Hanoi

Pierwszy ruch

	X		Y		Z
	ooo X ooo		Y		Z
oooo	X	oooo	Y	oo	Z oo

Drugi ruch

	X		Y		Z
	X		Y		Z
oooo	X	oooo	ooo Y ooo	oo	Z oo

Wieże z Hanoi

Drugi ruch

	X		Y		Z
	X		Y		Z
oooo	X	oooo	ooo Y ooo		oo Z oo

Trzeci ruch

	X		Y		Z
	X		oo Y oo		Z
oooo	X	oooo	ooo Y ooo		Z

Wieże z Hanoi

Trzeci ruch

	X		Y		Z
	X		oo Y oo		Z
oooo	X	oooo	ooo Y ooo		Z

Czwarty ruch

X		Y		Z
X		oo Y oo		Z
X		ooo Y ooo	oooo	Z oooo

Wieże z Hanoi

Czwarty ruch

```

X           Y           Z
X           oo Y oo     Z
X           ooo Y ooo   ooooo Z ooooo

```

Piąty ruch

```

X           Y           Z
X           Y           Z
oo X oo     ooo Y ooo   ooooo Z ooooo

```

Wieże z Hanoi

Piąty ruch

	X		Y		Z
	X		Y		Z
oo	X	oo	ooo Y ooo	oooo	Z ooooo

Szósty ruch

	X		Y		Z
	X		Y	ooo	Z ooo
oo	X	oo	Y	oooo	Z ooooo

Wieże z Hanoi

Szósty ruch

X	Y	Z
X	Y	000 Z 000
00 X 00	Y	0000 Z 0000

Siódmy ruch — stan docelowy

X	Y	00 Z 00
X	Y	000 Z 000
X	Y	0000 Z 0000

Wieże z Hanoi — algorytm rekursywny

```
procedure przenies (ile : integer; var X, Y, Z : kolek );  
  
begin  
  
if ile = 1 then  
    przenies_1(X, Z)  
        begin  
            przenies(ile-1, X, Z, Y);  
else  
    przenies_1(X, Z);  
    przenies(ile-1, Y, X, Z)  
end
```

end;

Wieże z Hanoi — algorytm rekursywny

```
procedure przenies (ile : integer; var X, Y, Z : kolek );  
  
begin  
  
if ile > 0 then  
    begin  
        przenies(ile-1, X, Z, Y);  
        przenies_1(X, Z);  
        przenies(ile-1, Y, X, Z)  
    end  
  
end;
```

Wieże z Hanoi — algorytm rekursywny

```
program Hanoi;  
  
const n      = 9;  
  
type zakres = 1..n;  
type kolek  = record  
    stan : array [zakres] of zakres;  
    top  : 0..n;  
    name : char  
end;  
  
var X, Y, Z : kolek;
```

```

procedure druk;
{ drukuje aktualne rozmieszczenie kregow na kolkach }
var i : zakres;

    procedure druk1(linia: zakres; var k : kolek);
    { drukuje krag na miejscu linia kolka k }
    var i : zakres;
    begin
        if k.top < linia then
            begin
                for i:= 1 to n do write(' ');
                write(' ',k.name,' ');
                for i:= 1 to n do write(' ')
            end
        end
    end

```

```

else
begin
    for i:= n downto 1 do
        if k.stan[linia] < i then write(' ')
        else write('o');
    write(' ',k.name,' ');
    for i := n downto 1 do
        if k.stan[linia] < n-i+1 then write(' ')
        else write('o')
    end
end;
begin
    for i:= n downto 1 do
    begin
        druk1(i, x); write(' ');

```

```

        druk1(i, y); write('      ');
        druk1(i, z); writeln
    end;
    writeln
end;

procedure genpelny(var K : kolek; name : char);
var i : zakres;
begin
    k.top := n;
    for i:= 1 to n do
        k.stan[i] := n+1-i;
    k.name:=name
end;

```

```
procedure genpusty(var K:kolek;name:char);
begin
    k.top:=0;
    k.name:=name
end;
```

```
procedure przenies1(var z, na : kolek);
begin
    na.top := na.top+1;
    na.stan[na.top] := z.stan[z.top];
    z.top:= z.top-1;
    druk
end;
```

```
procedure przenies (ile:zakres; var z, na, przez:kolek);
begin
  if ile > 0 then
  begin
    przenies(ile-1, z,przez,na);
    przenies1(z,na);
    przenies(ile-1,przez,na,z);
  end
end;
begin
  writeln;
  genpelny(x,'X');   genpusty(y,'Y');   genpusty(z,'Z');
  druk;
  przenies(n,x,z,y)
end.
```

Wydruk z programu

```
  o X o           Y           Z
 oo X oo         Y           Z
ooo X ooo       Y           Z

      X           Y           Z
 oo X oo        Y           Z
ooo X ooo      Y           o Z o

      X           Y           Z
      X           Y           Z
ooo X ooo     oo Y oo       o Z o
```

	X		Y		Z
	X		o Y o		Z
ooo	X	ooo	oo Y oo		Z
	X		Y		Z
	X		o Y o		Z
	X		oo Y oo	ooo	Z ooo
	X		Y		Z
	X		Y		Z
o	X	o	oo Y oo	ooo	Z ooo
	X		Y		Z
	X		Y	oo	Z oo

o X o

Y

ooo Z ooo

X

Y

o Z o

X

Y

oo Z oo

X

Y

ooo Z ooo

Algorytm Euklidesa — rekursja

```
program Euklides ;  
  var m, n : integer;  
  function NWD (m,n : integer) : integer;  
  begin  
    if m <> n then  
      if m > n then NWD := NWD(m - n, n)  
      else NWD := NWD(m, n - m)  
    else NWD := m  
  end;  
begin  
  <ciało programu>  
end.
```

Program — wywołanie funkcji NWD

```
writeln('Podaj dwie liczby naturalne');  
readln (m,n);  
if (m>0) and (n>0) then  
    writeln('NWD(',m,',',n,')= ',NWD(m, n))  
else    writeln('niewłaściwe dane')
```

TEX — jak produkuje się taki slajd?

```
\begin{slide}
\heading{ Program --- wywołanie funkcji \pz{NWD} }
\vfil
  \pz{writeln}(\pciag{'Podaj dwie liczby naturalne'})\Km;
  \pz{readln} (\pz m, \pz n)\Km;  \\\
  \pifelse{(\pz m$>$0) and (\pz n$>$0)}
    {\pz{writeln}(\pciag{'NWD('}, \pz m, \pciag
      {','}, \pz n, \pciag{'})= '}, \pz{NWD}(\pz m,
        \pz n))}%
    {\pz{writeln}(\pciag{'niewłaściwe dane'})}
\vfil
\end{slide}
```

TEX — składanie tekstów matematycznych

$$2^2 \cdots 2^2 \} n$$

$$\sum_{i=1}^n i = n * (n - 1) \quad (1)$$

$$\left| \begin{array}{l} n \\ k \\ 1 \end{array} \right| \quad \left| \begin{array}{l} \sum_{i=1}^n i \\ x^2 + \frac{1}{2} \end{array} \right|$$