

# Wstęp do programowania

## 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 na temat modelowanej dziedziny

# Architektura komputerów a programowanie

## Architektura komputerów w 3 minuty

- binarna reprezentacja liczb, ciągów, słów, tablic, rekordów
- binarna reprezentacja programów
- ograniczoność reprezentacji, a **nieograniczoność 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 (Schema, 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

# Struktury danych — mechanizmy kontroli działania

Wirth: Algorytmy + struktury danych = programy

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

# Pojęcie algorytmu

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

# Specyfikowanie programów imperatywnych

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

$\Pi$  : **program** — relacja opisująca zmianę **stanu**. Notacja:

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

$\Phi, \Psi$  : **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  $\Pi$  względem prewarunku  $\Phi$  i postwarunku  $\Psi$ .

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

**Własność stopu** programu  $\Pi$  względem prewarunku  $\Phi$ .

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

# Algorytm Euklidesa — specyfikacja

## Stan

Wartościowanie zmiennych **M**, **N** i **wynik**

## Prewarunek

**M** > 0, **N** > 0

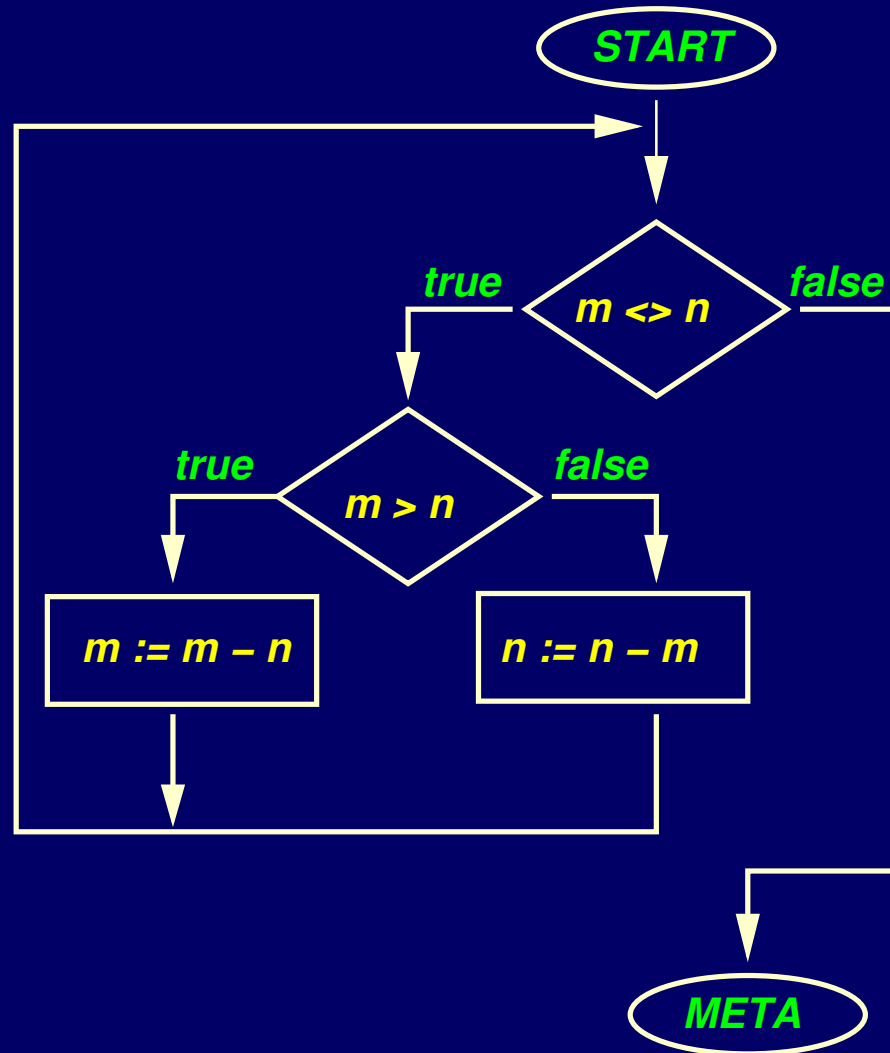
## Postwarunek

**wynik** =  $NWD(\mathbf{M}, \mathbf{N})$

## Prosta realizacja

**wynik** :=  $NWD(\mathbf{M}, \mathbf{N})$

# Algorytm Euklidesa — 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

$\Phi : \mathbf{M} > 0 \wedge \mathbf{N} > 0$  oraz  $\Psi : \mathbf{wynik} = NWD(\mathbf{M}, \mathbf{N})$

**Tw 2** Alg. Euklidesa ma własność stopu wzgl.  $\Phi$ .

**Wniosek** Alg. Euklidesa jest całkowicie poprawny wzgl.  $\Phi$  i  $\Psi$ .

# 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](http://www.freepascal.org)

dokumentacja on-line: [www.freepascal.org/docs.html](http://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')
```