Informatyka

MPDI 3 semestr

Wykład 5

Podstawy programowania

Algorytmy

Wstęp do języka JavaScript

- instrukcja przypisania

- wyrażenia arytmetyczne i logiczne

ALGORYTMY

Algorytm jest to sformalizowany ciąg logicznie powiązanych instrukcji (poleceń, rozkazów), których wykonanie pozwoli na przetworzenie informacji wejściowych (danych) w informacje wyjściowe (wyniki).

Algorytm – przepis na rozwiązywanie "krok po kroku" dowolnego problemu.

Algorytm ma przeprowadzić system z pewnego stanu początkowego do pożądanego stanu końcowego.

Badaniem algorytmów zajmuje się algorytmika.

Każdy algorytm komputerowy musi być wprowadzony do komputera w bardzo rygorystycznie zdefiniowanym języku - jednoznaczne instrukcje.

Jeżeli dany algorytm da się wykonać na maszynie o dostępnej mocy obliczeniowej i pamięci oraz akceptowalnym czasie, to mówi się że jest to algorytm obliczalny.

Algorytm – przepis - niezależny od implementacji

Program – zastosowanie algorytmu w języku zrozumiałym przez komputer

Czynności służące do rozwiązania zadania:

- analiza treści zadania
- wykaz danych wejściowych; wiadomych i niewiadomych oraz relacji między nimi
- sprawdzenie czy zadanie posiada jednoznaczne rozwiązanie
- wybór metody rozwiązania zadania
- opis czynności dla wybranej metody rozwiązania
- sporządzenie i przedstawienie wyników rozwiązania zadania

Sposoby zapisu algorytmów

- Opis słowny (pseudokod) - przedstawienie kolejnych czynności (akcji) na elementach (danych). Przykład:przepis kulinarny
- Schemat blokowy – operacje na danych przedstawione graficznie w postaci elementarnych bloków.

ZASADY BUDOWY SCHEMATU BLOKOWEGO

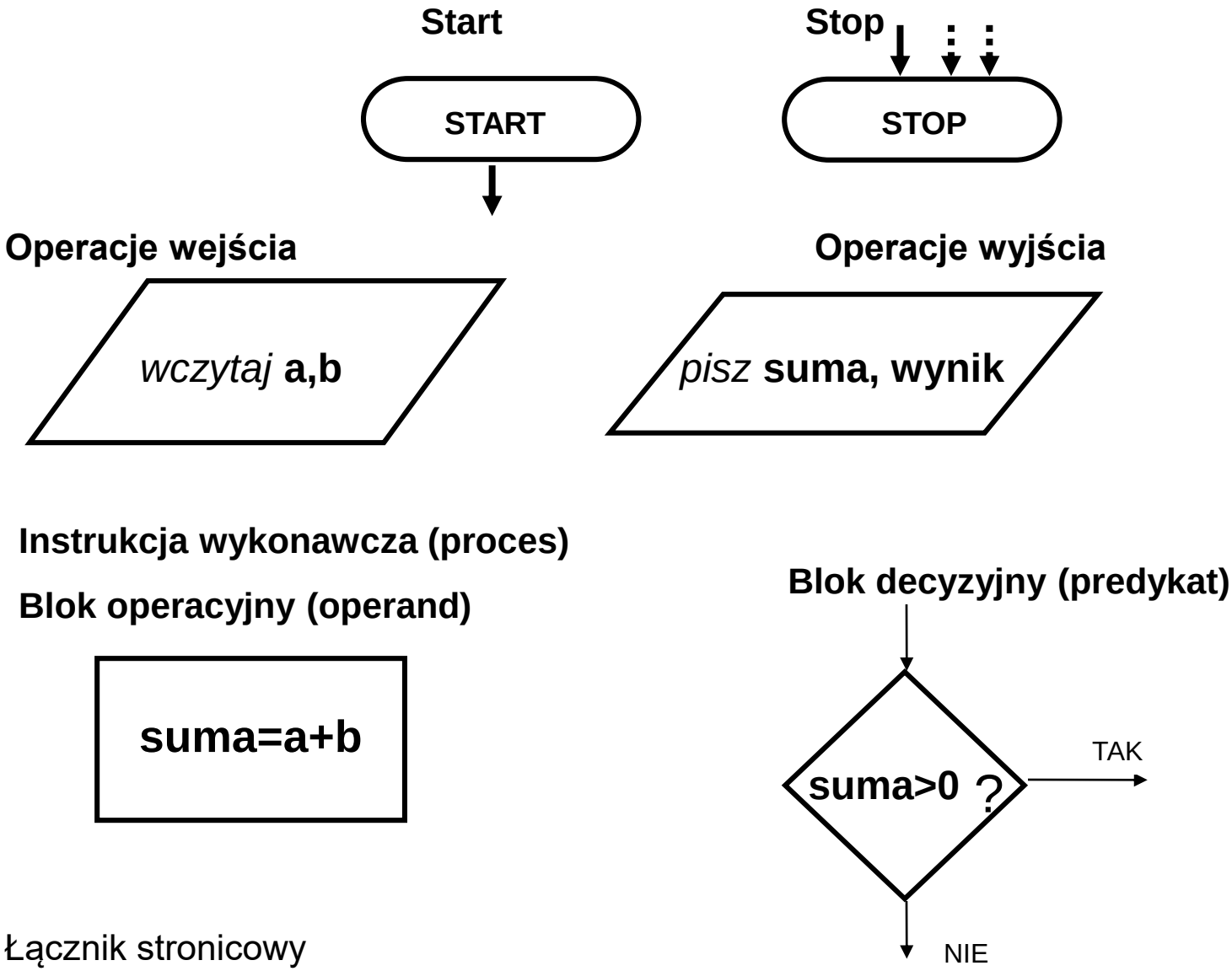
1) Każda operacja jest umieszczona w bloku
2) Schemat ma tylko jeden blok "START" i przynajmniej jeden blok "STOP"
3) Bloki mają połączenia **ukierunkowane**
4) Z bloku wychodzi **jedno** połączenie;

   wyjątki:

   - "STOP" (nie wychodzą żadne połączenia)
   - blok "warunkowy" (wychodzą dwa połączenia opisane TAK i NIE)

5) W bloku "operacyjnym" odbywa się nadanie wartości (przypisanie) znak :=

Reguły graficzne tworzenia schematów blokowych



Klasy problemów

- algorytmy obliczeniowe (np. całkowanie numeryczne, obliczanie funkcji z sumy wyrazów ciągu, wyznaczanie liczb pierwszych, zamiana systemów liczbowych itp.)
- algorytmy sortujące
- algorytmy wyszukujące
- algorytmy kompresji

Metodyka

- algorytm "krok po kroku" (sekwencyjny)
- algorytmy iteracyjne
- algorytmy rekurencyjne
- inne (np. genetyczne, sztucznej inteligencji)

Dla utworzenia algorytmu konieczne są:

- opis **elementów** do przechowywania w pamięci danych wejściowych, danych pośrednich i wyników
- opis **czynności** jakie należy wykonać z elementami przechowującymi dane, co realizujemy przy pomocy instrukcji, które opisują:
  - ✓ sposób działania
  - ✓ kolejność ich wykonywania
  - ✓ ewentualne sprawdzanie warunków jakie muszą być spełnione.
- opis **wyników** - zawiera sposób udostępnienia wyników rozwiązanego zadania (dane, wykresy, zapis do plików)

Rodzaje sieci działań:

Proste (sekwencyjne) - kolejność realizacji poszczególnych operacji jest ściśle określona i żadna z nich nie może być pominięta ani powtórzona - nie używa się bloków warunkowych.

Z **rozwidleniem** - zawiera w sobie wybór jednej z kilku możliwych dróg realizacji danego zadania - istnieje przynajmniej jeden **blok warunkowy**.

Z **pętlą**, często w trakcie realizacji zadania konieczne jest powtórzenie niektórych operacji różniących się zestawem danych. Pętla obejmuje tę część bloków, która ma być powtarzana.

Złożone - będące kombinacją powyższych sieci.

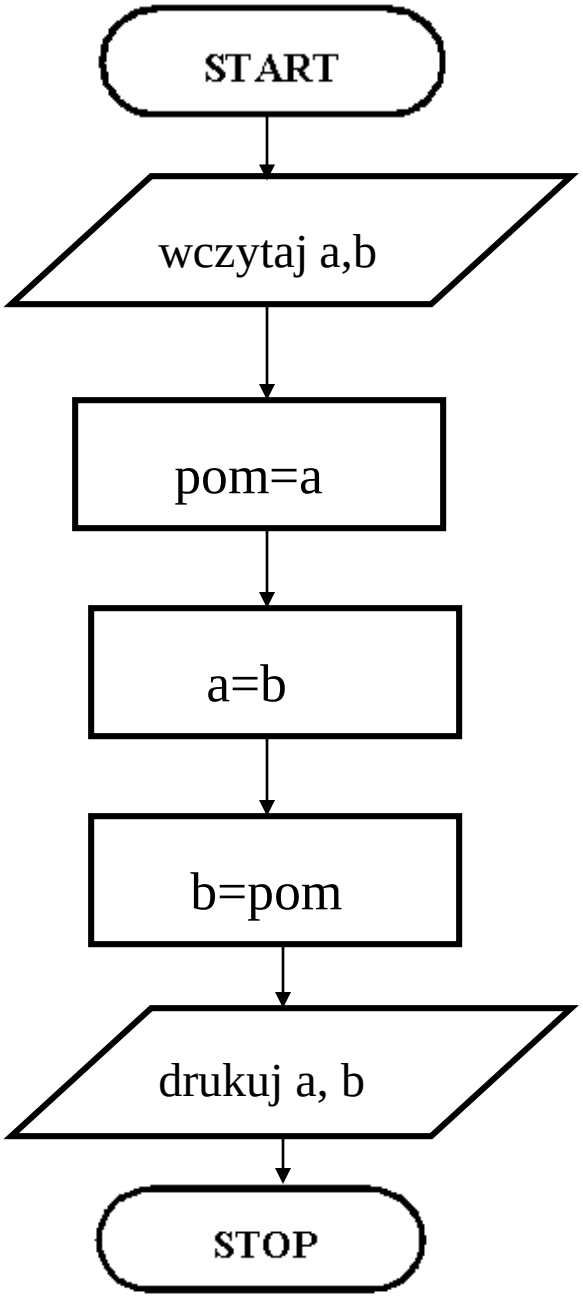
Przykład

W algorytmach sortujących potrzebny jest mechanizm zamiany wartości umieszczonych w dwóch zmiennych, jeśli są w niewłaściwej kolejności

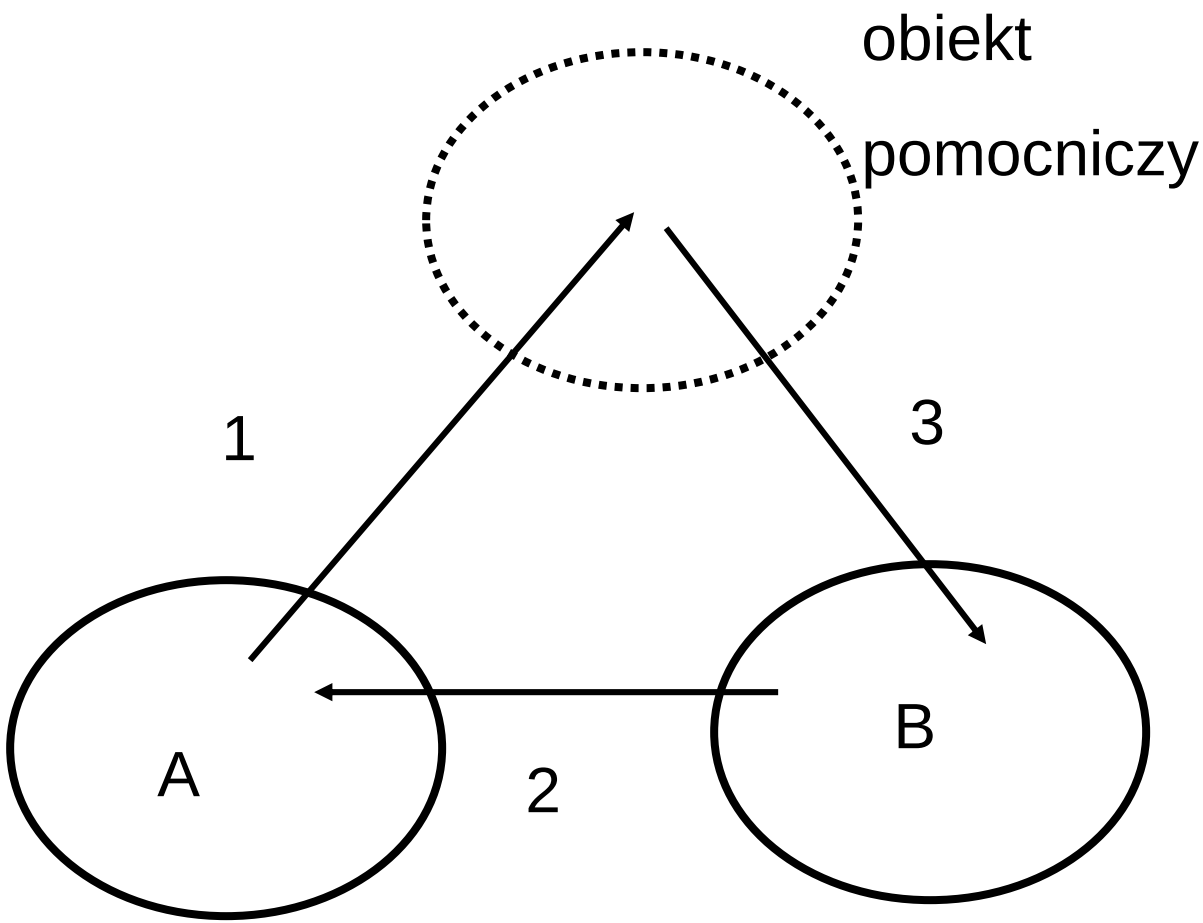
Algorytm wymaga zmiennej pomocniczej (jak do zamiany zawartości dwóch szklanek potrzebna jest trzecia szklanka)

Opis słowny:

1. Wczytaj dane do obiektów 1 i 2
2. Przepisz zawartość obiektu 1 do obiektu pomocniczego
3. Przepisz zawartość obiektu 2 do obiektu 1
4. Przepisz zawartość obiektu pomocniczego do obiektu 2
5. Wyprowadź wartości obiektów 1 i 2



schemat blokowy zamiany wartości w dwóch zmiennych

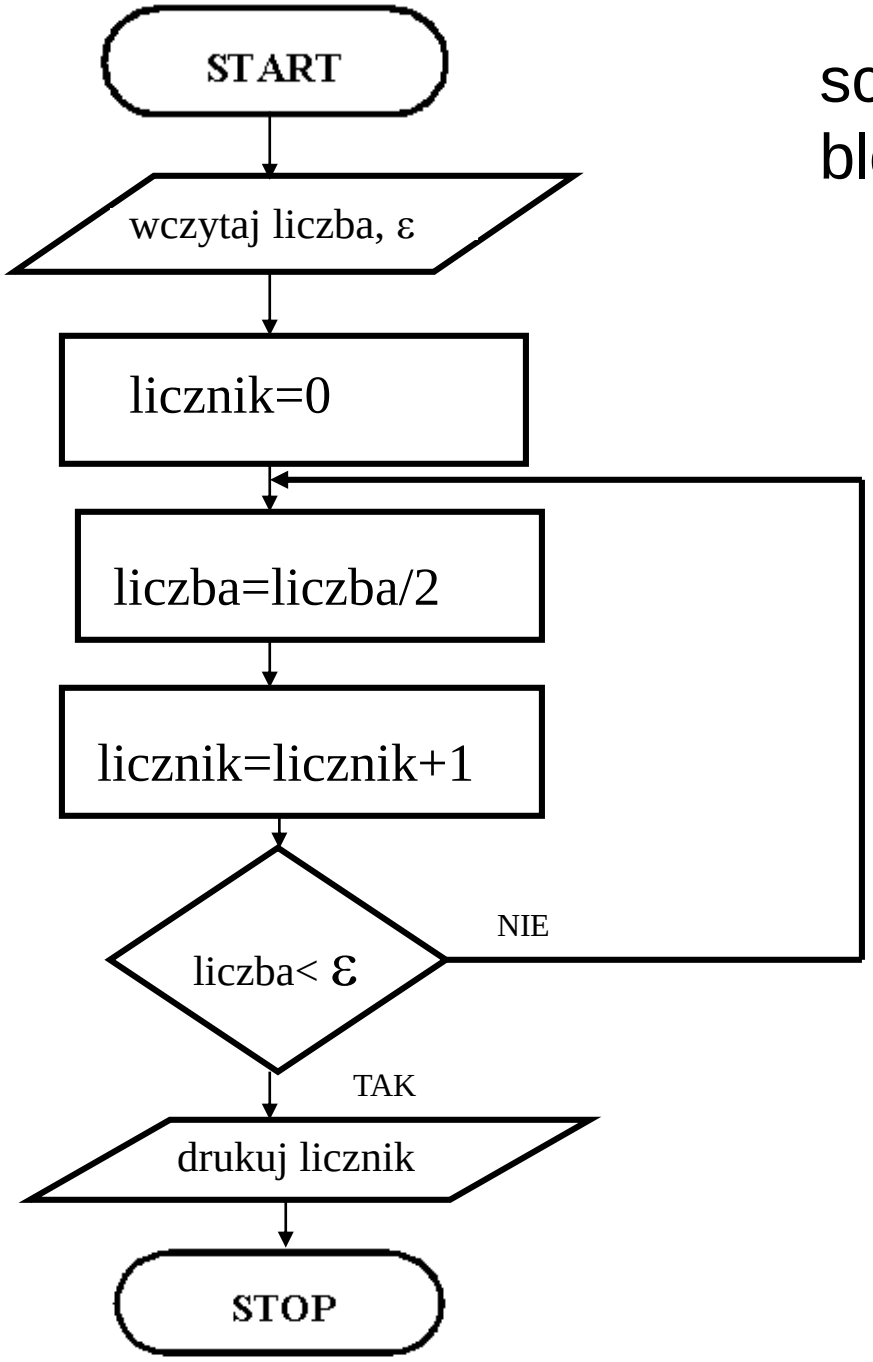

obiekt
pomocniczy
1
3
A
2
B

Przykład 2 – **z blokiem decyzyjnym i pętlą**

Problem: Ile razy trzeba podzielić na pół daną liczbę, aby uzyskać wartość mniejszą od ε

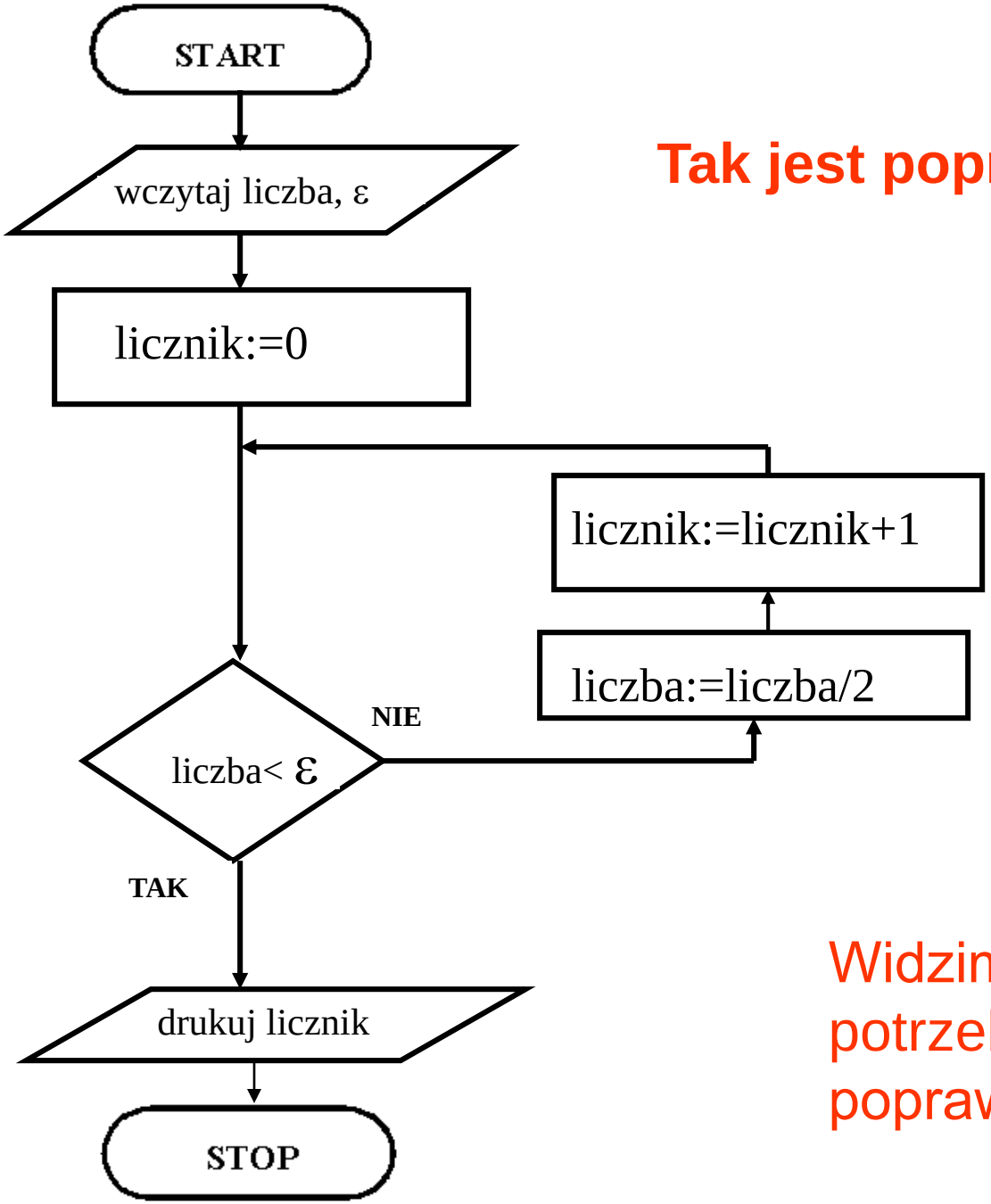
Opis słowny:

1. Wczytaj liczbę i ε
2. Ustal wartość licznika równą 0
3. Podziel liczbę przez 2
4. Zwiększ licznik o 1
5. Sprawdź czy wynik jest mniejszy od ε, jeśli tak, to przejdź do punktu 6, jeśli nie to wróć do punktu 3
6. Wyprowadź wartość licznika



schemat blokowy

zauważamy niepoprawność tego algorytmu w przypadku gdy liczba na wstępie jest już mniejsza od ε, sprawdzenie warunku trzeba wykonać wcześniej!



Tak jest poprawnie

Widzimy, że jest potrzebna analiza poprawności

A więc, aby tworzyć efektywne algorytmy i móc je aplikować programowo, potrzebne są następujące elementy:

Spektrum elementów programowych, możliwość ich odróżniania, klasyfikacja do typu, umieszczanie ich identyfikatorów (nazw) i wartości w pamięci

Spektrum operacji (operatorów nadawania wartości, arytmetycznych, porównania) – operacje zgodne z typem danej

Mechanizm badania warunków (rozwidlenia)

Mechanizm powtarzania (iteracje - "pętle")

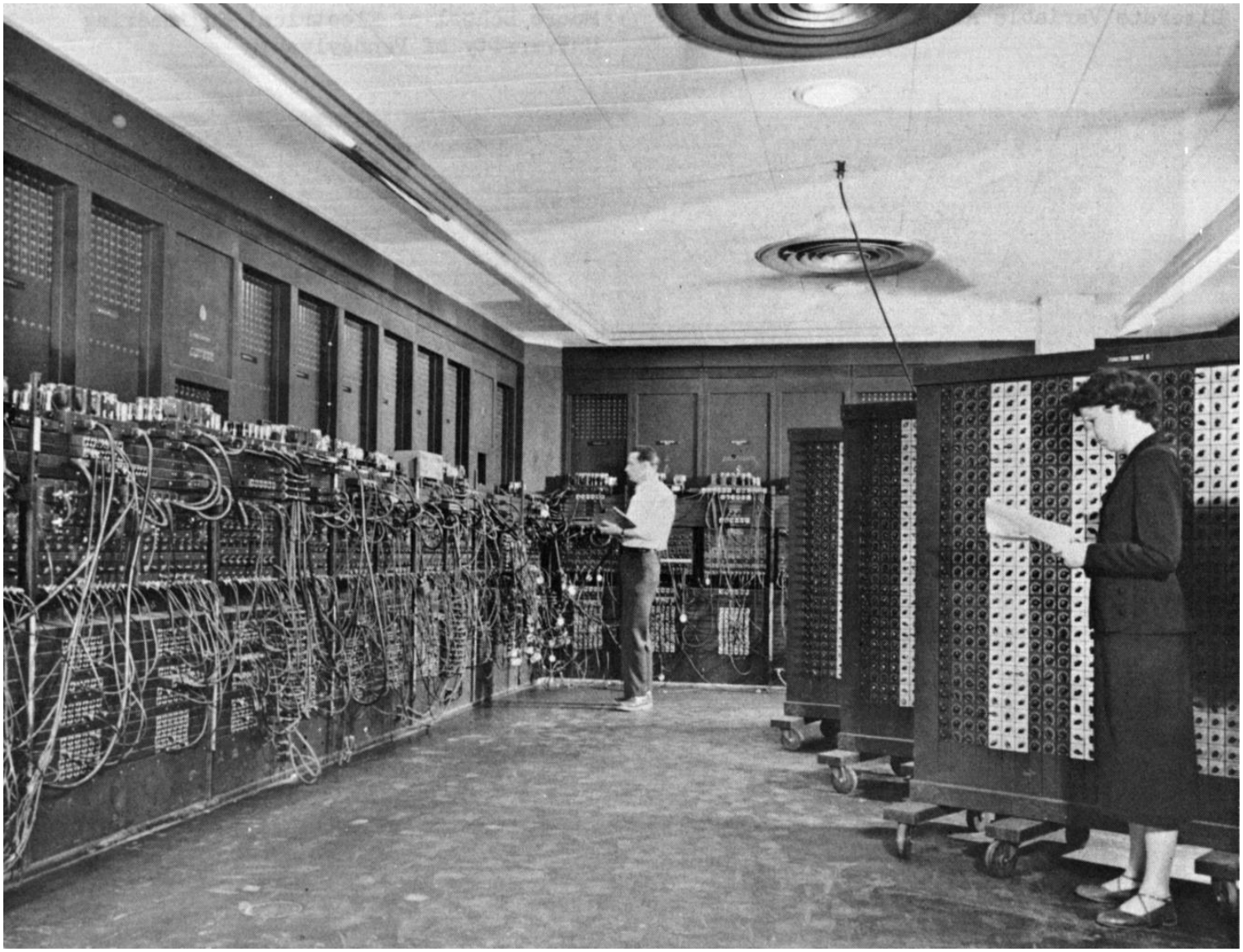
Mechanizm sterowania – co wykonać następnie po aktualnej operacji

Historia programowania

Do 1946-54 roku – język maszynowy (Eniac, Mark I- USA) – niskiego poziomu - binarny

Kodowanie instrukcji procesora w postaci ciągu liczb dwójkowych (elementarny rozkaz i dane)

- pracochłonność (same liczby, długie algorytmy działań takich jak dodawanie i mnożenie) – np. rzymskie
- trudność śledzenia błędów



lata 50-te - zastąpienie kodów liczbowych operacji nazwami mnemonicznymi np. mnożenie MPY (multiply)

język symboliczny - assembler

ASSEMBLER – język, a także program tłumaczący na kod maszynowy (wewnętrzny)

DISSASSEMBLER – tłumaczenie odwrotne

Stosowany do dziś – np. oprogramowanie interfejsów pomiarowych, wprowadzanie danych z pomiarów i ich obróbka komputerowa

Cechy *assemblera*

- zmniejszona ale nadal duża pracochłonność
- błędy – prawie każda kombinacja znaków mogła być wykonana
- ukierunkowanie na konkretny komputer (procesor) dla którego program był pisany
- wąski zbiór rozkazów, prymitywna architektura ówczesnych komputerów,
- brak rejestrów (pamięci pomocniczych),
- brak operacji zmiennoprzecinkowych

Etap przejściowy – tzw. systemy automatycznego programowania – „sztuczki” programistyczne pozwalające na prostsze wykonywanie operacji zmiennopozycyjnych na liczbach

1954 + języki tzw. **wysokiego poziomu**

(kod maszynowy i assembler to niski poziom – bliższy procesorowi)

zapoczątkował język **Fortran** (Formula Translator), standard opisany później - w 1966 roku.

Zapis operacji w sposób łatwiejszy, zrozumiały i dobrze kontrolowany przez programistę

Po stworzeniu kodu programu w języku następuje proces tłumaczenia (translacji lub kompilacji) na język wewnętrzny komputera.

Problemy:

- **zrozumiałość**
- **jednoznaczność**
- **skuteczność tłumaczenia**

Języki wysokiego poziomu po etapie początkowym stały się maszynowo niezależne z powodu wielu wersji programów kompilujących

Kompilator (ang. compiler) to program służący do tłumaczenia kodu napisanego w jednym języku (tzw. **języku źródłowym**) na równoważny kod w innym języku (języku wynikowym). Najczęściej jest to tłumaczenie z języka wysokiego poziomu (znacznie łatwiejszego dla programisty) ma język wewnętrzny procesora.

Proces ten nazywany jest kompilacją.

Języki wysokiego poziomu

COBOL – dla przedsiębiorstw, język prawie naturalny (ang)

Do dziś specjaliści potrzebni.

W kolejnych latach - obfitość języków programowania

BASIC, LOGO – prostota, interpretacja w odróżnieniu od kompilacji, tłumaczenie „na bieżąco” każdej instrukcji a nie programu w całości

PASCAL, C, C+, C++ - języki strukturalne z elementami tzw. programowania obiektowego

OOP (Object Oriented Programming) – języki zorientowane obiektowo - **PROLOG, Visual Basic, Turbo Vision dla Pascala,**

Java, C# *(fonet. "si-szarp"*), **Python**

Lista 20 najpopularniejszych języków programowania wg Stackoverflow (stan na 2023):

1. **JavaScript** (67%)
2. **Python**(47%)
3. **Java** (41%)
4. C# (31%)
5. PHP(26%)
6. C++(23%)

Język Asemblera
Swift
Objective-C
Ruby
Groovy
Go
Perl
Delphi/Object Pascal
MATLAB
Visual Basic
PL/SQL

Klasyfikacja języków programowania

- do **przetwarzania** danych: COBOL, Dbase
- do **obliczeń** naukowych : Fortran, Pascal (w zasadzie uniwersalne)
- do programowania **systemowego** – tworzenie systemów operacyjnych: C, BCPL, BLISS
- opisu **poleceń** - DOS, MVS
- **konwersacyjne**: LISP, zmiany z terminala w trakcie konwersacji
- **symulacyjne** – symulacja procesów (najczęściej w czasie rzeczywistym): mechanika, elektronika, termodynamika itp. – SIMULA, CSSL
- **algorytmiczne** (większość): programowanie to zapis sposobu wykonania
- **niealgorytmiczne**: nie jak ale co ma być zrobione – sterowanie nie algorytmem lecz danymi – problematyka sztucznej inteligencji
- **obiektowe** – pewien zbiór danych (obiekty) są aktywne i mają swoje cechy (własności); poprzez zdarzenia (zwykle wymuszone przez użytkownika ale nie tylko) mogą te własności zmieniać lub wywoływać akcje obliczeniowe
- **specjalizowane** – ukierunkowane na pewną klasę problemów, np. obróbkę baz danych, analizę układów elektronicznych itp.

CO TO JEST JĘZYK PROGRAMOWANIA ?

DANE – reprezentują świat rzeczywisty lub abstrakcyjny, przechowywane są w nazwanych zmiennych określonego typu, mają przypisany zbiór operacji – działań na tych obiektach

Alfabet języka - skończony zbiór znaków.

Słowo - skończony ciąg znaków alfabetu

Język formalny- zbiór słów nad określonym alfabetem

Język programowania – zbiór konwencji i umów umożliwiających komunikatywność (zrozumiałość) dla kompilatora.

Określa się przy tym:

- syntaktykę (składnię) języka - zbiór reguł opisujących poprawne konstrukcje językowe
- semantykę - zasady interpretacji tych konstrukcji

Składnia języka – określa zasady kolejności słów kluczowych, operatorów, argumentów.
Przykładowe błędy to:

- brak słowa kluczowego instrukcji (for if end itp.)
- błąd literowy w słowie kluczowym
- brak operatora mnożenia pomiędzy argumentami
- brak domknięcia nawiasu

Semantyka – definiowane opisowo znaczenie symboli i słów, oraz ich znaczenie - błędy trudne do wychwycenia, czasem pojawiają się w trakcie wykonania,
np.

- brak pliku z danymi,
- zły typ danych,
- odwołanie do nieistniejącego argumentu lub niezdefiniowanej funkcji,
- j=0; k=1/j (dzielenie przez zero)
- nieprawidłowy obszar określoności funkcji (np. sqrt – pierwiastek arytmetyczny z liczby ujemnej))

Jeżeli ponadto:

- syntaktyka pozwala na analizę poprawności konstrukcji językowych
- opis języka obejmuje pragmatykę (zalecenia do używania poprawnych struktur)

to język formalny może być językiem programowania.

Zmienna przechowuje dane

dana $\Rightarrow$(**nazwa** danej, **typ** danej,**wartość** danej)

Np.

x	typ liczbowy	100
y	typ tekstowy	'Kowalski'
z	typ logiczny	prawda

deklaracje, definicje - opisy zmiennych i ich typów

instrukcje - operacje na zmiennych

Program - algorytm zapisany w języku programowania

Podprogram - wyodrębniona część programu (ze względu na czytelność lub wielokrotne użycie) posiadająca wyodrębnioną nazwę i sposób wymiany informacji z pozostałymi jego częściami

Zazwyczaj blok rozpoczynający się słowem kluczowym function

W innych językach: subroutine, procedure

Definicja podprogramu - opis działania podprogramu – jakie są argumenty i jakie na nich wykonywać operacje, co jest rezultatem podprogramu

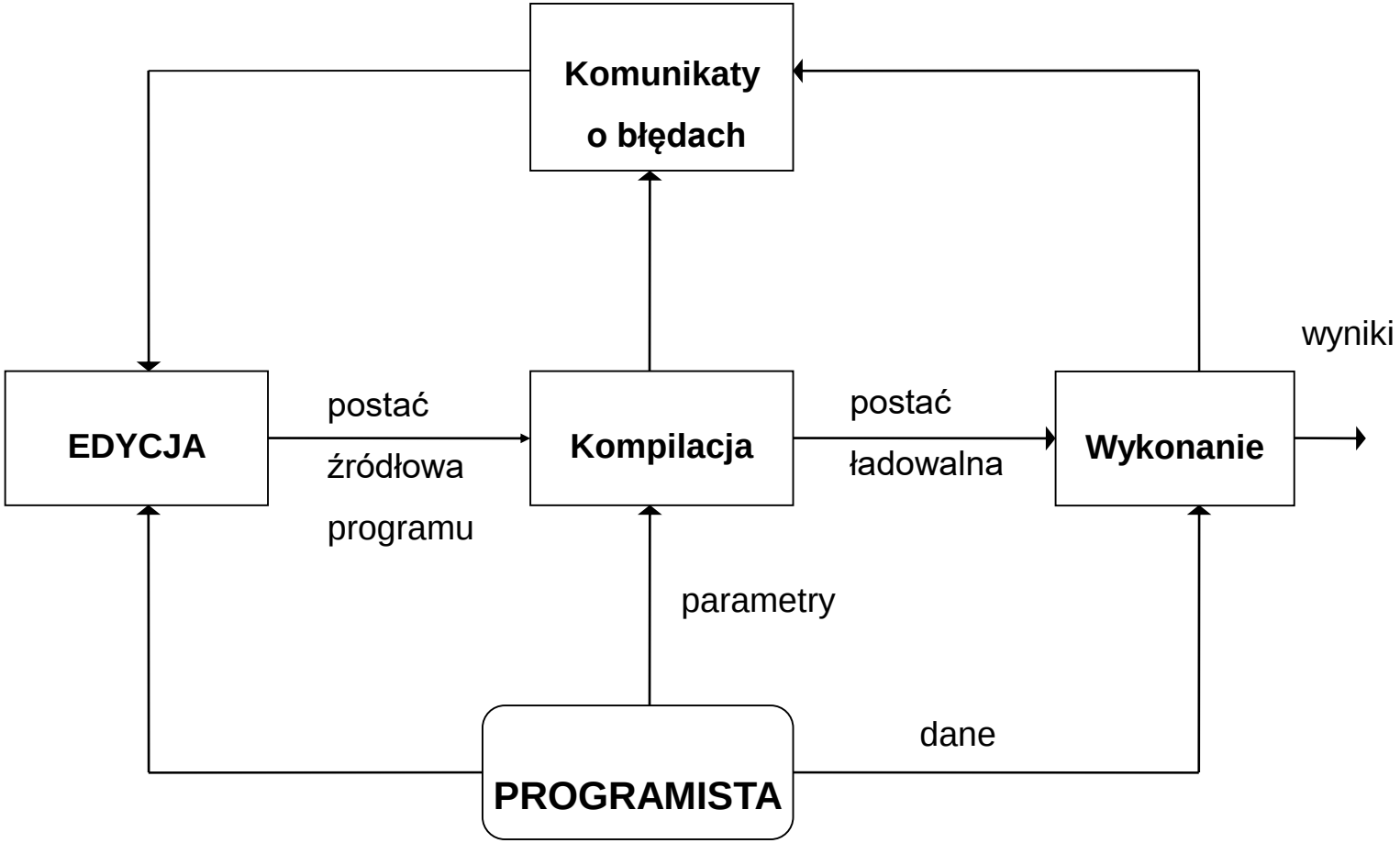
Instrukcja wywołania (wykonania) podprogramu – wykonanie, zastosowanie podprogramu wewnątrz programu głównego lub innego podprogramu

Kod źródłowy - tekst programu w języku programowania (plik tekstowy ***html pas m***)

Kompilacja - tłumaczenie (w całości) kodu źródłowego na wykonywalną postać binarną, ładowalną (plik *exe*) – odrębny proces - wyszukuje wszystkie błędy składniowe

Interpretacja – tłumaczenie kolejnych instrukcji w trakcie procesu wykonywania – wykonane są instrukcje prawidłowe, zatrzymanie na pierwszym błędzie

Przetwarzanie programu użytkowego



W przypadku przetwarzania programu, wymagającego wykorzystania programów bibliotecznych, uzyskany po procesie kompilacji kod wynikowy należy poddać procesowi łączenia, konsolidacji (link) z kodami dołączanymi a następnie wykonać kod ładowalny.

Sposoby pisania programu:

1. Napisanie programu źródłowego dowolnym edytorem tekstu
2. kompilacja
3. konsolidacja (łączenie z podprogramami bibliotecznymi – gotowymi modułami)
4. wykonanie

lub

Korzystanie ze zintegrowanych pakietów programistycznych (wewnętrzne narzędzia edycji, kompilacji i konsolidacji, weryfikacji błędów itp.)

Co świadczy o przydatności i skuteczności języka programowania

- **Dobra notacja, przejrzystość, prostota (łatwość przyswojenia zasad)**
- **Jednoznaczność zapisu**
- **Funkcjonalność**
- **Weryfikacyjność – test błędów (czasem te cechy w sprzeczności, bo deklaracje poszerzają strukturę programu ale ułatwiają kontrolę zmiennych, np. błąd literowy w nazwie obiektu bez deklaracji to nowa zmienna, a z deklaracjami błąd braku deklaracji)**
- **Elastyczna opisywalność obiektów: np. data to liczby, operacje liczbowe dopuszczalne ale dzielenie jest bezsensowne**
- **Czytelność struktury programu**
- **Dostępność**
- **Cena**
- **Efektywność – skuteczność kompilacji, szybkość programu wynikowego**
- **Dobra dokumentacja**

Metodyka programowania

Programowanie strukturalne – opiera się na podziale kodu programu na procedury (**podprogramy**) i hierarchiczne bloki; wykorzystuje struktury kontrolne: instrukcje wyboru (**warunkowe**) i iteracje ("**pętle**"). Zarzucono wykorzystane poprzednio proste instrukcje warunkowe i skoki (*goto*).

Programowanie obiektowe (ang. *object-oriented programming*, OOP) –definiuje się **obiekty** – elementy łączące dane (nazywane najczęściej polami) i *zachowanie* (czyli procedury, nazywane metodami). Zbiór obiektów komunikuje się pomiędzy sobą w celu wykonywania zadań.

Podejście to różni się od tradycyjnego programowania, gdzie dane i procedury nie są ze sobą bezpośrednio związane.

Javascript

JavaScript jest to interpretowany, **zorientowany obiektowo**, skryptowy **język programowania**.

JavaScript jest oddzielnym językiem (nie jest uproszczoną wersją Javy).

Javascript stosunkowo jest łatwy w nauce i pozwala na pewne „zdynamizowanie” stron internetowych.

JavaScript nie wymaga licencji

- kod źródłowy można tworzyćw zwykłym edytorze tekstu nieformatowanego,
- każda przeglądarka jest platformą translacji, wykonania i także **debuggowania** (kontroli błędów)

Cechy JavaScript

Może być osadzany jako składnik pliku HTML,

Przeglądarki internetowe "potrafią" zinterpretować instrukcje języka JavaScript (client-side)

Przeglądarki

- powinny mieć włączoną obsługę JavaScript
- mogą pytać o zezwolenie na wykonanie skryptu.

… a zatem

HTML definiuje zawartość stron internetowych

CSS specyfikuje ich formę

zaś **JavaScript**

- daje szerokie możliwości obliczeń i analiz numerycznych oraz prezentacji wyników,

- może decydować o funkcjonowaniu elementów strony, mając dostęp do obiektów modelu obiektowego znaczników na stronie internetowej (tzw. – DOM, *document object model*), można sprawować nad tym otoczeniem kontrolę

Wstawienie skryptu do dokumentu HTML

Skrypty JavaScript są zagnieżdżane w dokumentach HTML. Skrypt JavaScript umieszczane są w znaczniku <SCRIPT>

```
<SCRIPT LANGUAGE="JavaScript">

 treść skryptu

</SCRIPT>
```

Albo prościej – język JavaScript jest domyślny

```
<SCRIPT>

 treść skryptu

</SCRIPT>
```

```
<HTML><HEAD></HEAD>
<BODY>
<P> To jest tekst 1</P>
<SCRIPT LANGUAGE="JavaScript">
 treść skryptu 1
</SCRIPT>
<P> To jest akapit HTML</P>
<SCRIPT LANGUAGE="JavaScript">
 treść skryptu 2
</SCRIPT>
</BODY></HTML>
```

Może istnieć wiele skryptów naprzemiennie z pozostałym kodem HTML

Jak w innych językach, aby tworzyć działające skrypty JavaScript wymagane są:

- jakaś metoda **wprowadzania** i **przechowania danych** (liczb, tekstów itp.)
- jakieś **instrukcje**, które umożliwią obliczenia, sprawdzanie warunków, powtórzenia, wywołanie podprogramów itp.
- jakieś **metody prezentacji wyników**

Jak widzimy skrypt składa się z wielu instrukcji

```
<SCRIPT LANGUAGE="JavaScript">
instrukcja1;
instrukcja2
instrukcja3; instrukcja4
itd
</SCRIPT>
```

Instrukcje wykonywane są kolejno od pierwszej do ostatniej (a zatem istotna jest ich kolejność)

Mogą być zakończone znakiem średnika **; (lub też pisane w osobnych wierszach)**

Średnik konieczny do oddzielenia instrukcji pisanych w tym samym wierszu

Kilka zasad przy pisaniu skryptów

Komentarze

Jednowierszowy komentarz od znaków // do końca wiersza

// tekst

Komentarz o kilku wierszach, pomiędzy dwuznakami ***/**** i ****/***

/*

treść komentarza

****/***

Wykorzystywane są w celu:

- opisy wyjaśniające

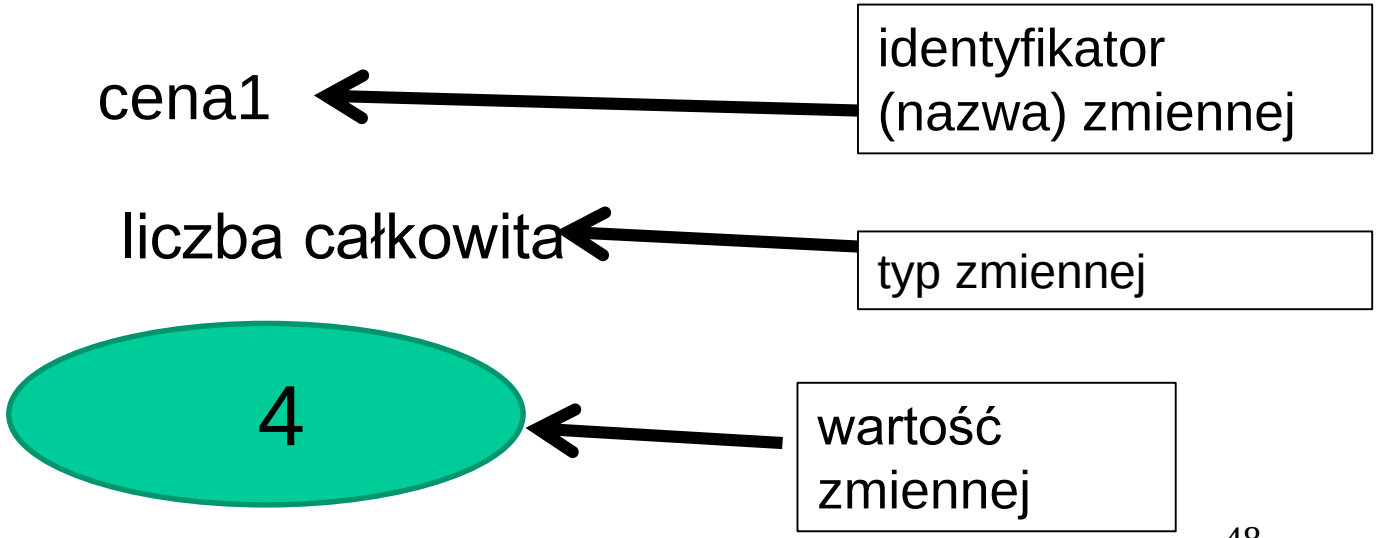
- dezaktywowanie instrukcji (testy, błędy)

Zmienne przechowują dane !!!

Zmienne nie **muszą być deklarowane! (opisane przed użyciem)**

Zmienna ma:

- **nazwę (identyfikator)**
- **typ**
- **wartość**



Identyfikatory

Są to nazwy elementów (zmiennych, obiektów, funkcji)

Zmienne służą do **przechowania wartości** określonego typu

Jednym z podstawowych sposobów nadania wartości zmiennej jest instrukcja przypisania (nadania wartości)

Ciąg liter, cyfr, znaków podkreślenia (nie wolno spacji!)

Musi się zaczynać od litery

Ważne duże i małe litery ! (w odróżnieniu od innych języków)

Przykłady identyfikatorów dla zmiennych:

x **alfa** **Alfa** mojaZmienna B11 c_33

to dwie różne zmienne

styl "wielbłądzi"

Instrukcje JavaScript – wykonanie akcji!

Czasem blok kilku instrukcji otaczamy klamrami { } zazwyczaj we wnętrzu instrukcji warunkowych i iteracyjnych (np**. if, for, while**) – o nich za chwilę

```
<SCRIPT LANGUAGE="JavaScript">
instrukcja1
{
instrukcja2
instrukcja3
instrukcja4
}
</SCRIPT>
```

wyodrębniony blok instrukcji

Typy instrukcji

- **przypisania** (nadania wartości)

- **warunkowe**

- **iteracje (pętle)**

- **wykonania funkcji** (metody obiektowej lub własnej użytkownika)

Instrukcja przypisania

zmienna **operator_przypisania** [*wyrażenie*];

[ ] oznaczają, że wyrażenie jest opcjonalne (pomijamy wyrażenie w przypadku operatorów ++ i --)

Po **lewej** stronie operatora przypisania tylko nazwa zmiennej!!! inicjowanej (pierwszy raz definiowanej) lub istniejącej i przedefiniowywanej.

Po **prawej** stronie operatora przypisania piszemy wyrażenie (**bardzo podobne jak w Excelu**), zawierające liczby, operatory, nawiasy okrągłe, funkcje – metody obiektów, oraz zmienne o ZNANYCH WARTOŚCIACH

Wyrażenie musi być "obliczalne" – wszystkie jego elementy muszą być znane, przede wszystkim użyte w nim zmienne. Obliczona wartość wyrażenia jest przechowana w zmiennej.

Operatory przypisania

Operator inicjacji wartości (lub nowej wartości)

= nadaj zmiennej wartość obliczonego wyrażenia

Operatory składania (zmiana wartości – zmienna wcześniej musi mieć ewartość)

+= zwiększ o wartość wyrażenia

–= zmniejsz o wartość wyrażenia

*= pomnóż przez wartość wyrażenia

/= podziel przez wyrażenie

%= reszta z dzielenia przez wyrażenie

++ inkrementacja – zwiększenie o 1

-- dekrementacja – zmniejszenie o 1

Po operatorach ++ i - - brak wyrażenia

Przykłady instrukcji przypisania

```
a=3     //nadaj zmiennej a wartość 3
a++     //zwiększ tę wartość o 1  (a=4)
x=7     //nadaj x wartość 7
x--     ///zmniejsz tę wartość o 1 (x=6)
x+=2.5 //zwiększ wartość x o 2.5  (x=8)
x–=a   //zmniejsz x o wartość a  (x=4)
a*=4   //pomnóż a przez 4, wynik w a  (a=16)
x/=2   //x podziel przez 2, wynik w x  (x=2)
a%=x  //reszta z dzielenia x przez a, wynik w a (a=0)
```

++ inkrementacja – zwiększenie o 1 (brak wyrażenia)

-- dekrementacja – zmniejszenie o 1 (brak wyrażenia)

Typ zmiennej

Typ zmiennej zależy od typu wartości wyrażenia przypisywanego zmiennej

x = 4	całkowitoliczbowy
y= 2.3	dziesiętny - stałoprzecinkowy
y2= 3.4e8	dziesiętny - zmiennoprzecinkowy
z = true	logiczny
v = "Kowalski"	tekstowy (łańcuchowy)

Postacie wyrażeń

stała

zmienna

funkcja(*wyrażenie*)

wyrażenie operator *wyrażenie*

definicja rekurencyjna – z niej wynika, że funkcje można zagnieżdżać praz może być długi ciąg wyrażeń łączonych operatorami

dodatkowo stosujemy nawiasy (okrągłe) dla zmiany domyślnego priorytetu operatorów

Operatory arytmetyczne

+	dodawanie
-	odejmowanie (także zmiana znaku)
*	mnożenie
/	dzielenie
%	reszta z dzielenia

addytywne

multipikatywne

Oczywiście operatory * / maja wyższy priorytet od + i –

W języku JavaScript operator ^ nie służy do potęgowania! (jest to tzw. bitowy XOR)

operator + służy też do konkatenacji (łączenia tekstów)

"Mateusz " + 'Kowalski'

para cudzysłowów lub para apostrofów

Przykład

```
x=7
```

Wykonanie takiej instrukcji:

- inicjuje zmienną o identyfikatorze x
- nadaje jej wartość 7 (typ liczbowy)
- zmienną można wykorzystać w następnych instrukcjach

Przykłady instrukcji przypisania wyrażeń do zmiennej:

```
z1 = 4.67
z2 = z1
z3 = z1/2
```

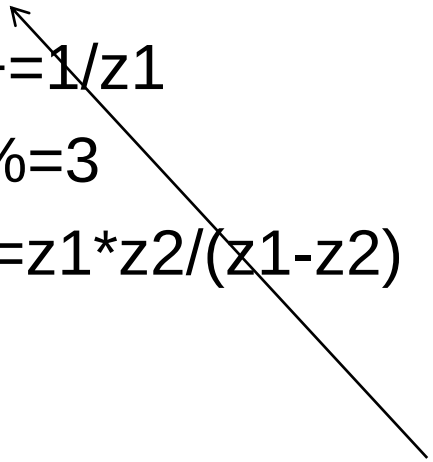
inicjacja (nowe zmienne)

```
z1 = -6.7
```

inna wartość zmiennej z1

```
z3++
z3+=1/z1
z3%=3
z3/=z1*z2/(z1-z2)
```

zmiana wartości (wcześniej zainicjowanej zmiennej)



Można też: z3=z3+1

Inne przykłady:

```
x=5
alfa= 3*x
b15=(3-x)+2.7/alfa
b15++
x+=4.5
```

stałe liczbowe:

liczby całkowite

liczby dziesiętne z KROPKĄ!!!