TECHNIKI PROGRAMOWANIA

Wykład 1:

Języki programowania

Ryszard Myhan

Programowanie

to proces projektowania, tworzenia, testowania i utrzymywania kodu źródłowego programów komputerowych lub urządzeń mikroprocesorowych.

Programowanie

- wymaga dużej wiedzy i doświadczenia w wielu różnych dziedzinach, takich jak projektowanie aplikacji, algorytmika, struktury danych,
- znajomości języków programowania i *narzędzi programistycznych*,
- wiedzy nt. kompilatorów, czy sposób działania podzespołów komputera.

Kod źródłowy

Kod źródłowy

(ang. *source code*) – ciąg instrukcji i deklaracji opisujący operacje, jakie powinien wykonać komputer przy pomocy skończonej liczby ściśle zdefiniowanych rozkazów.

Kod źródłowy

jest napisany w języku programowania, z użyciem określonych reguł, może on być modyfikacją istniejącego programu lub czymś zupełnie nowym.

Kod źródłowy

pozwala wyrazić w czytelnej dla człowieka formie strukturę i działanie programu komputerowego, biblioteki czy *skryptu*.

Kod źródłowy

Kod źródłowy

w zależności od wielkości projektu, może składać się z jednego lub większej liczby plików tekstowych, niekiedy pogrupowanych w katalogi, może też występować w postaci procedur składowanych w bazach danych.

Kod źródłowy

musi zostać poddany translacji na kod wynikowy, np. skompilowany (przetłumaczony) do postaci *kodu maszynowego* lub kodu pośredniego, możliwe jest także wykonywanie go w locie za pośrednictwem dodatkowego programu zwanego interpreterem.

Translacja kodu źródłowego

Translator

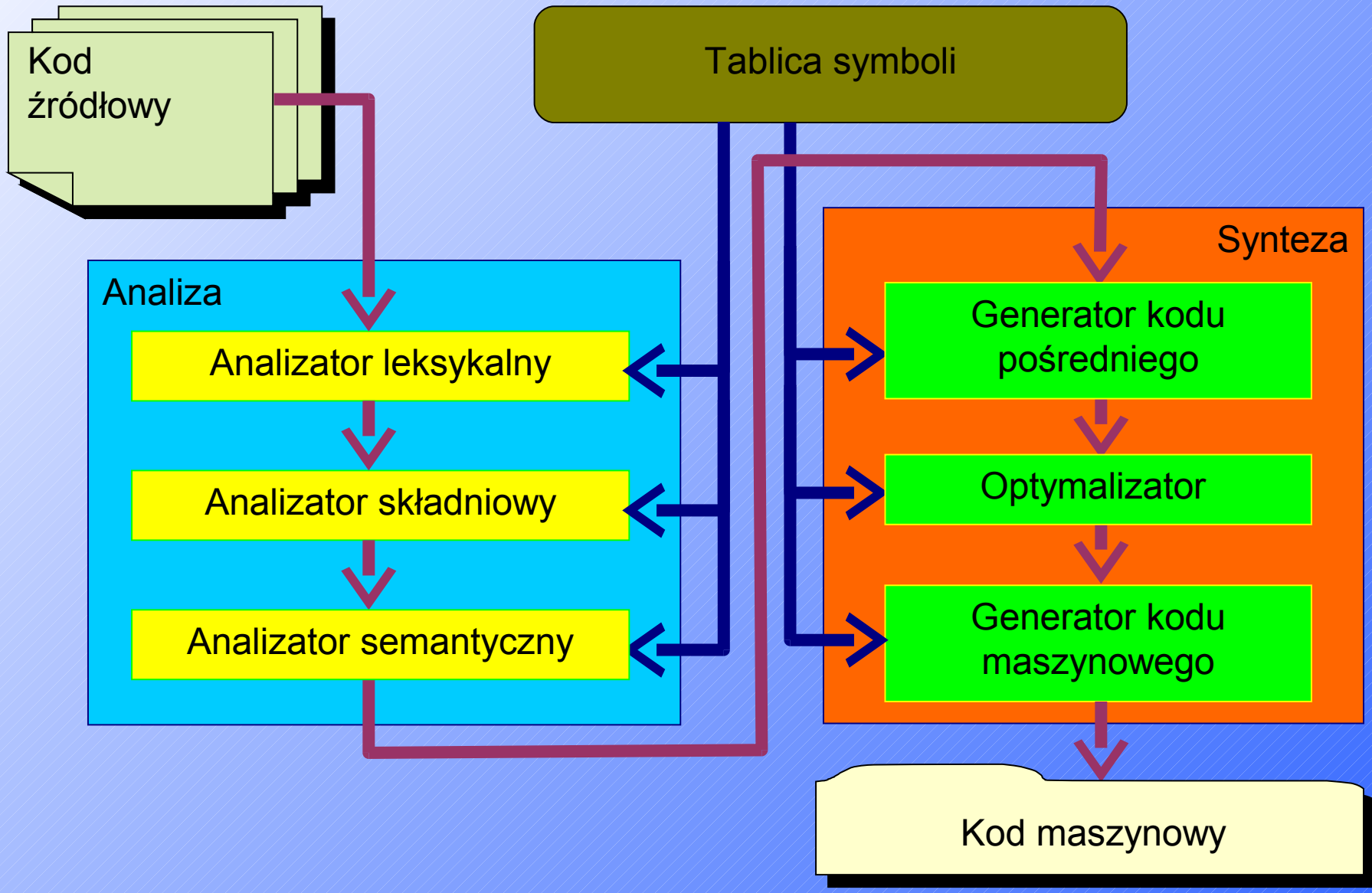
program dokonujący przekładu tekstu wyrażonego w języku programowania na kod wynikowy.

★ Zadaniem translatora jest wykonanie rozbioru gramatycznego przedłożonego tekstu, aby rozstrzygnąć, czy tekst jest poprawnym programem.

★ W skład każdego translatora wchodzi:

- analizator leksykalny,
- analizator składniowy,
- analizator semantyczny,
- generator kodu pośredniego,
- optymalizator kodu,
- generator kodu wynikowego.

Translator - struktura



Analiza leksykalna

ma miejsce wszędzie tam, gdzie wczytuje się dane o określonej składni.

★ Wczytując takie dane, zanim będziemy mogli je przetwarzać, musimy rozpoznać ich składnię.

★ Pomysł polega na tym, aby najpierw wczytywany ciąg znaków podzielić na elementarne cegiełki składniowe - nazywane *leksemami* , a dopiero dalej analizować ciąg leksemów.

★ Analizator leksykalny (nazywany też skanerem) to wyodrębniony moduł zajmujący się tym zadaniem.

Analizator leksykalny

Zadania

- Wczytuje strumień znaków składający się na program wejściowy – strumień jest wczytywany od lewej do prawej;
- Grupuje znaki ze strumienia w symbole leksykalne (tokeny);
- Buduje i wypełnia tablicę symboli leksykalnych;
- Usuwa z pliku wejściowego komentarze, białe znaki (spacje, tabulacje), znaki nowych wierszy;
- Dopasowuje komunikaty o błędach w kodzie źródłowym (śledzenie numerów wierszy);
- Rozwija makra preprocesora, jeśli język je udostępnia.

Analiza leksykalna

Większość symboli leksykalnych należy do jednej z grup:

- → nazwy (identyfikatory);
- → słowa zarezerwowane (podzbiór zbioru nazw);
- → liczby całkowite;
- → liczby rzeczywiste;
- → łańcuchy znakowe;
- → operatory: addytywne (+, -), unarne (+, -),
  multiplikatywne (*, /),
  relacyjne (<, >, =, <=, >=, <>),
  logiczne (and, or, not), przypisania (:=);
- → ograniczniki jednoznakowe: ; , [ ] ( ) ;
- → ograniczniki dwuznakowe: (* , *), +=, ++ itd.

Analiza leksykalna

Leksemy:

pozycja := poczatek + tempo * 60

Symbole leksykalne:

identyfikator op_przypisania identyfikator op_add identyfikator op_mul liczba

identyfikator:
zbiór zaczynający się od litery lub podkreślenia, po którym następuje dowolnie długi ciąg liter, cyfr, podkreśleń

op_add:
operator dodawania lub odejmowania

op_mul:
operatory mnożenia i dzielenia

op_przypisania:
operator składający się ze znaku „:” po którym następuje znak „=„

Identyfikator	typ	adres
pozycja	real	0
poczatek	real	8
tempo	real	16

Analiza składniowa

ma miejsce wówczas, gdy wczytujemy dane o pewnej określonej składni.

★ Analiza składniowa stanowi kolejną fazę przetwarzania wczytywanych danych po analizie leksykalnej.

★ Zadaniem analizatora składniowego (tzw. *parsera*)

jest weryfikacja poprawności składniowej programu źródłowego, wykrycie i neutralizacja błędów składniowych oraz transformacja struktury programu do postaci drzewa wywodu.

★ Efektem pracy parsera jest odtworzenie drzewa wyprowadzenia (a ściślej mówiąc jego obejścia) dla

Analiza składniowa

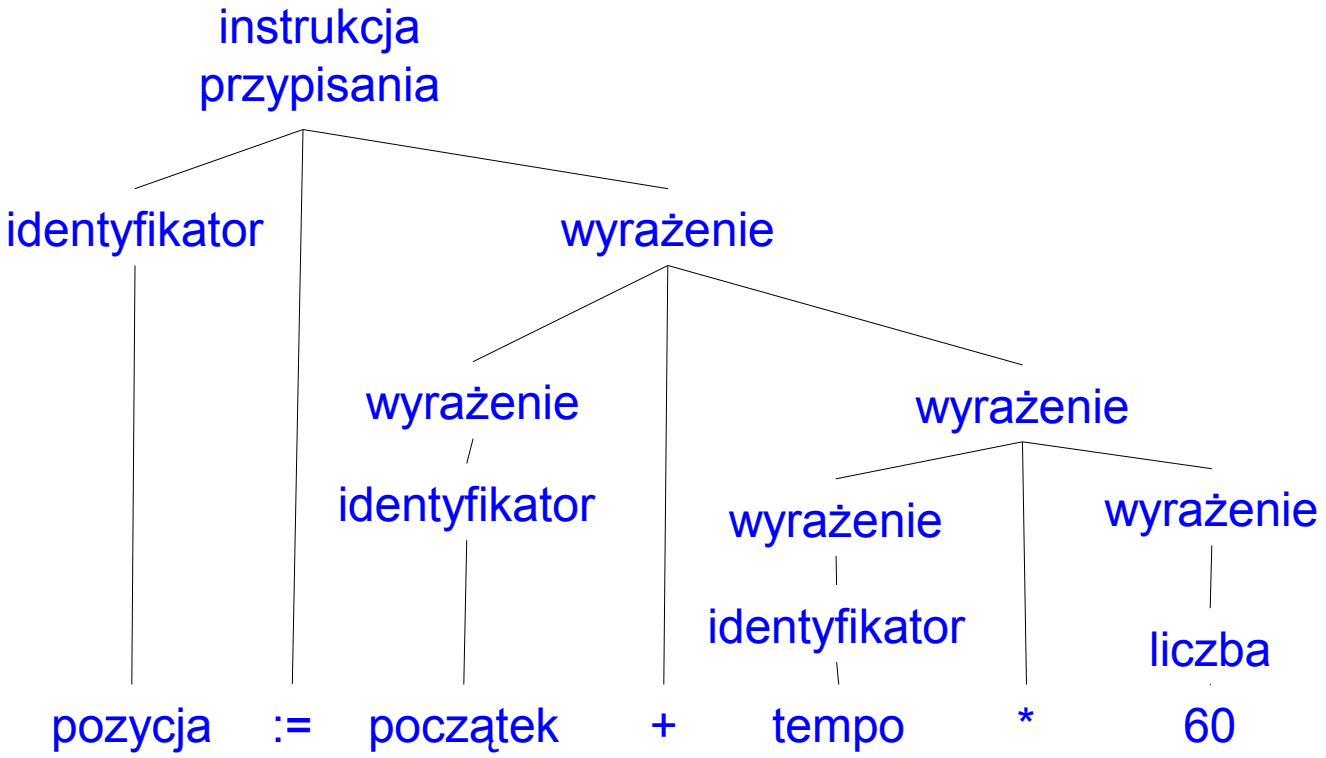
istrukcja_przypisania -> **identyfikator** **':='** wyrażenie

wyrażenie -> **liczba**

wyrażenie -> **identyfikator**

wyrażenie -> wyrażenie **'+'** wyrażenie

wyrażenie -> wyrażenie **'*'** wyrażenie



Analiza semantyczna

to faza procesu kompilacji, wykonywana po analizie syntaktycznej, a przed generowaniem kodu, w której sprawdzana jest poprawność programu na poziomie znaczenia poszczególnych instrukcji oraz programu jako całości.

★ Główne zadania analizy semantycznej to:

- weryfikacja poprawności semantycznej programu źródłowego;
- sprawdzenie czy zmienne są zadeklarowane;
- sprawdzenie zgodności pomiędzy parametrami aktualnymi i formalnymi procedur i funkcji;
- sprawdzenie czy obiekty nie są deklarowane wielokrotnie.

Analiza semantyczna - cele

★ Analiza semantyczna najczęściej operuje na drzewie składni, do którego dodaje dodatkowe informacje.

★ W analizie semantycznej można wyróżnić trzy obszary działania:

- kontrola typów, czyli sprawdzanie poprawności typów w każdym węźle drzewa składni programu (w tym także sprawdzanie, czy *identyfikatory* zostały zadeklarowane);
- kontrola poprawności instrukcji, czyli sprawdzenie, czy instrukcje i wyrażenia mają sens w kontekście, w którym zostały użyte;
- kontrola nazw, czyli sprawdzenie, czy nazwy jednoznacznie identyfikują *funkcje*, *etykiety* i inne konstrukcje języka programowania.

Analiza semantyczna – kontrola typów

Kontrola typów

ma na celu sprawdzenie poprawności typów w takich konstrukcjach językowych jak:

- **przypisania** - typ wartości przypisywanej musi być zgodny z typem elementu do którego przypisujemy ,
- **operacje arytmetyczne** - wartości, do których używany jest operator, muszą zgadać się z rodzajem operatora,
- **wywołania funkcji** - typy parametrów funkcji przy jej wywołaniu muszą zgadzać się z zadeklarowanymi,
- **odwołania do pól rekordu** - rekord, do którego się odwołujemy, musi mieć pole o podanej nazwie,
- **wywołania metod obiektu** - obiekt musi być instancją klasy, która zawiera wywoływaną metodę.

Analiza semantyczna – kontrola poprawności instrukcji

Kontrola poprawności instrukcji

obejmuje wszelką inną poprawność instrukcji poza sprawdzaniem typów i identyfikatorów.

★ Sprawdzanie obejmuje:

- **kontrolę L-wartości** - pojęcie to odnosi się do wartości, które istnieją dłużej niż przez jedno wyrażenie i można pobrać ich adres;
- **kontrolę przepływu sterowania** - czyli co, kiedy i w jakiej kolejności się wykonuje;
- **kontrolę dostępu do obiektów i klas** - zapobiega użyciu klasy w sposób niepożądany.

Analiza semantyczna – kontrola nazw

Kontrola nazw

polega na sprawdzeniu, czy nazwy identyfikatorów i etykiet w programie źródłowym są poprawne.

Tablica symboli

przechowuje informacje na temat stałych, zmiennych, funkcji, procedur zadeklarowanych w programie.

Informacje w tablicy symboli są zbierane podczas analizy leksykalnej i składniowej, natomiast są wykorzystywane podczas analizy syntaktycznej, optymalizacji i generacji kodu

Analiza semantyczna – kontrola nazw

Za niepoprawne zwykle uważa się:

- deklaracje dwóch zmiennych o tej samej nazwie w tym samym *zakresie widoczności*,
- deklaracje dwóch funkcji o tej samej nazwie i tych samych parametrach w tym samym zakresie widoczności,
- użycie niezadeklarowanej zmiennej,
- użycie tej samej nazwy w dwóch kontekstach, np. jako nazwa typu rekordu i nazwa zmiennej,
- użycie słowa zastrzeżonego jako identyfikatora,
- deklaracja dwóch etykiet (np. dla goto lub case) o tej samej nazwie, lub brak deklaracji użytej etykiety.

Generator kodu pośredniego

Generator kodu pośredniego

jest programem, który generuje kod w pewnym niskopoziomowym języku, którego przekształcenie na kod bajtowy powinno być już proste.

Generacja kodu pośredniego

niezależnego od sprzętu oraz wejściowego języka programowania umożliwia przenośność kompilatora pomiędzy różnymi architekturami oraz językami.

Generacja kodu pośredniego

umożliwia implementację optymalizacji niezależnych d docelowej architektury oraz wejściowego języka programowania

Optymalizator kodu

Optymalizacja kodu pośredniego

(ulepszanie kodu) ma na celu poprawę jego efektywności (miarami efektywności są: przyspieszenie kodu i redukcja rozmiaru kodu).

- Optymalizacja może być wykonywana na poziomie kodu źródłowego, kodu pośredniego i kodu wynikowego.
- Do optymalizacji stosowane są metody analizy przepływu danych i sterowania oraz metody heurystyczne.

Optymalizator kodu

- Problem wygenerowania optymalnego kodu zależnego sprzętowo jest praktycznie nierozwiązywalny.
- Przykładowe algorytmy optymalizacji:
  - usuwanie kodu martwego;
  - eliminacja podwyrażeń wspólnych;
  - propagacja kopii;
  - zmienne indukcyjne;
  - redukcja mocy;
  - użycie tożsamości algebraicznych.

Generator kodu wynikowego

Generator kodu wynikowego

na podstawie pośredniej reprezentacji programu źródłowego wytwarza równoważny program docelowy.

Kod wynikowy

to rezultat pracy translatora (np. kompilatora), nadający się do bezpośredniego wykonywania przez procesor albo wymagający dalszej obróbki (np. konsolidacji).

Generator kodu wynikowego

Główne zadania generatora kodu wynikowego:

- Wygenerowanie kodu wynikowego w postaci:
  - kodu asemblerowego
  - przemieszczalnego kodu maszynowego
  - gotowego programu wykonywalnego
- Wybór rozkazów
- Przydział rejestrów

Generator kodu wynikowego – kod maszynowy

- Program w kodzie maszynowym składa się z ciągu wartości binarnych, które oznaczają zarówno instrukcje jak i dane.
- Postać kodu maszynowego zależy od architektury procesora, na który dany program jest przeznaczony.
- Program musi zostać skompilowany na konkretnej maszynie, ewentualnie na systemie kompatybilnym z systemem docelowym.

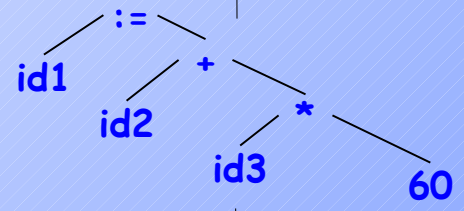
Translacja pojedynczej instrukcji - przykład

pozycja := poczatek + tempo * 60

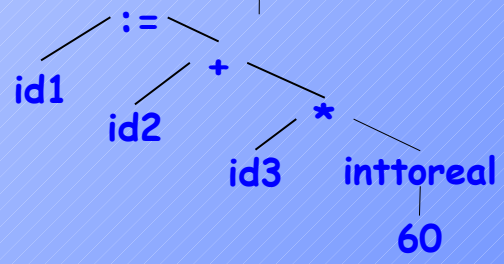
Analizator leksykalny

id1 := id2 + id3 * 60

Analizator składniowy



Analizator semantyczny



Generacja kodu pośredniego

```
temp1 := inttoreal (60)
temp2 := id1 * temp1
temp3 := id2 + temp2
id3 := temp3
```

Optymalizator kodu

```
temp1 := tempo * 60.0
pozycja := poczatek + temp1
```

Generator kodu

```
MOVF 16, R2
MULF #60.0, R2
MOVF 8, R1
ADDF R2, R1
MOVF R1,0
```

Translacja - sposoby

KOMPILATOR

translator języka wysokiego poziomu, którego programy wynikowe mogą być wprowadzone do pamięci i wykonane dopiero po zakończeniu tłumaczenia. Programy wynikowe kompilatora mogą być przechowywane, łączone z innymi programami i wielokrotnie wykonywane.

INTERPRETER

translator przekładający instrukcje programu na kod pośredni, który jest następnie interpretowany przy każdym wykonaniu. Ponieważ interpreter nie tworzy przekładu w kodzie maszynowym, lecz wykonuje instrukcje, tłumacząc je na bieżąco za każdym razem, wykonanie programu znacznie się wydłuża.

Konsolidacja

LINKER STATYCZNY

program łączący (konsolidujący) biblioteki lub moduły biblioteczne z modułami programu wynikowego wyprodukowanymi przez kompilator; efektem działania konsolidatora jest kod w postaci gotowej do ładowania.

LINKER DYNAMICZNY

to część systemu operacyjnego, która wywoływana jest w chwili uruchomienia programu i odpowiada za załadowanie do przestrzeni adresowej procesu niezbędnych dla niego *bibliotek programistycznych*.

Narzędzia wspomagające pisanie kompilatorów

Istnieje kilka narzędzi służących do konstrukcji pewnych modułów kompilatora, które wspomagają i znacznie przyspieszają cały proces pisania kompilatorów.

- Są to przede wszystkim:
  - generatory analizatorów leksykalnych – **LEX**, **FLEX**
  - generatory analizatorów składniowych – **YACC**, **Bison**.

❖ **LEX** i **FLEX** na podstawie specyfikacji symboli leksykalnych zadanych w postaci wyrażeń regularnych generują funkcję analizatora leksykalnego w języku C/C++.

❖ **YACC** i **Bison** dostając na wejściu specyfikację języka źródłowego (gramatykę i akcje semantyczne związane z produkcjami gramatyki) generują odpowiednią funkcję analizatora składniowego wraz z analizatorem semantycznym również w języku C/C++.

Język programowania - definicja

- **Język programowania** – ogólne określenie sztucznego języka, pozwalającego na zapis zadania i sposobu jego wykonania przez komputer.
- Podobnie jak języki naturalne, język programowania składa się ze zbiorów reguł **syntaktycznych** oraz **semantycznych**, które opisują, jak należy budować poprawne wyrażenia oraz jak komputer ma je rozumieć.
- Język programowania określony jest przez podanie jego składni i semantyki - ścisłych zasad tworzenia - **instrukcji**, określających elementarne (w danym języku) operacje.

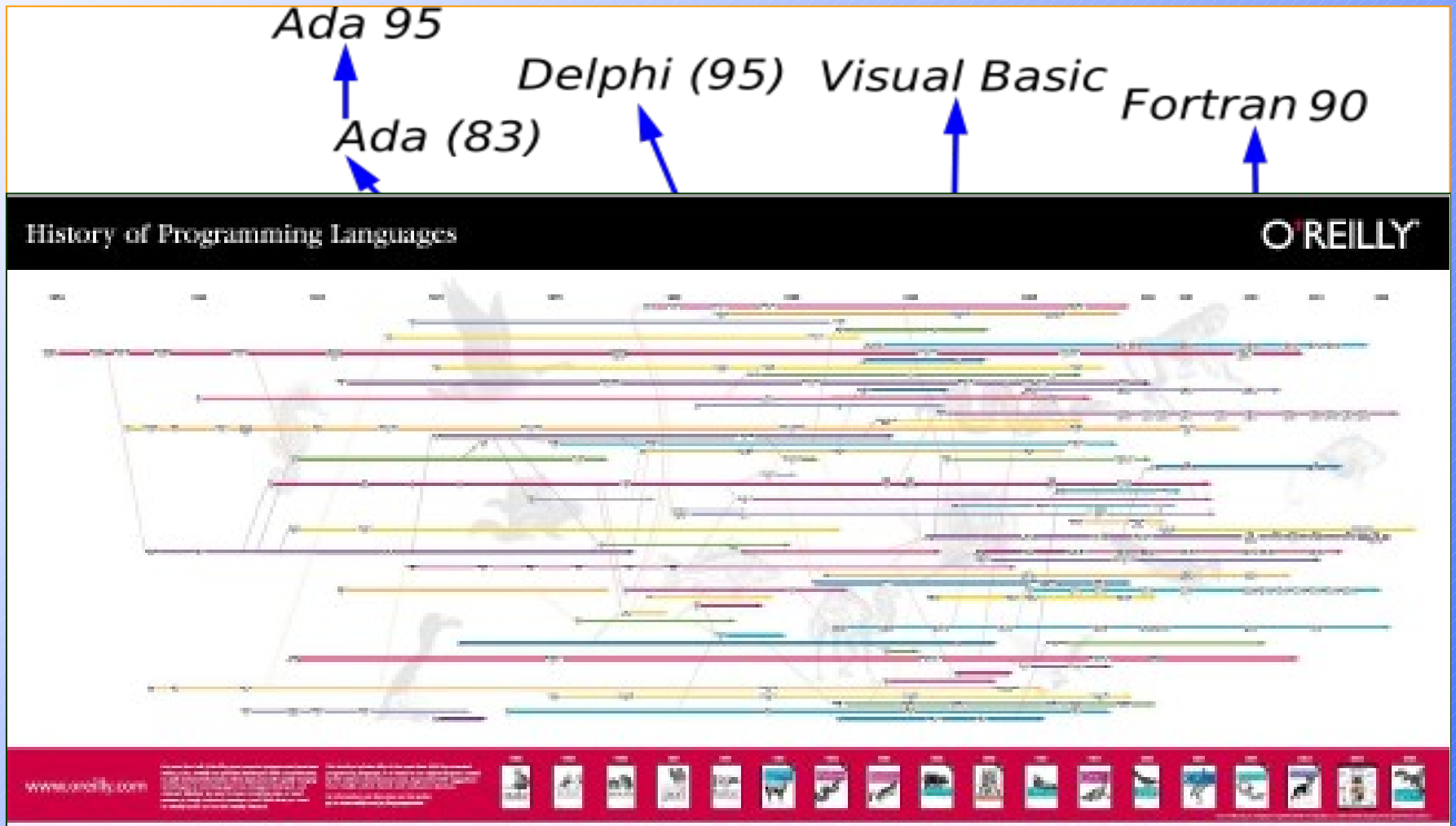
Języki programowania

- **Języki programowania** od języków naturalnych odróżniają się **wysoką precyzją** oraz **jednoznacznością**.

- Ludzie komunikując się między sobą stale popełniają niewielkie błędy lub pozostawiają niedomówienia wiedząc, że drugi rozmówca najczęściej go zrozumie.

- **Maszyny wykonują zadania dokładnie, dlatego każdą czynność trzeba opisać ściśle krok po kroku, ponieważ komputer nie potrafi domyślić się, co programista miał na myśli.**

Języki programowania



Elementy języki programowania

- Na język programowania składa się kilka elementów:
  - składnia
  - semantyka
  - typy danych
  - biblioteki standardowe
- **Składnia** opisuje rodzaje dostępnych symboli oraz zasady, według których symbole mogą być łączone w większe struktury.
- **Semantyka** definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję w programie.

Elementy języki programowania

Typy danych

- Każdy język operuje na jakimś zestawie danych, dlatego niezbędne jest podzielenie danych na odpowiednie typy, zdefiniowane ich właściwości oraz operacji, jakie można na nich realizować.

- Większość języków posiada typy danych do reprezentowania:
  - liczb całkowitych w różnych zakresach.
  - liczb zmiennoprzecinkowych (reprezentacje liczb rzeczywistych o różnym stopniu dokładności)
  - ciągów tekstowych.

Elementy języki programowania

Biblioteki standardowe

- Dla większości języków zdefiniowana jest biblioteka standardowa zawierająca podstawowy zestaw funkcji pozwalających realizować wszystkie najważniejsze operacje, takie jak:
  - obsługa wejścia-wyjścia;
  - obsługa plików;
  - obsługa wielowątkowości;
  - zarządzanie pamięcią.
- Użytkownicy traktują bibliotekę standardową często jako część języka, lecz od strony twórców są to osobne twory.

Podział języków programowania

- Języki programowania mogą być podzielone ze względu na:
  - Paradygmat programowania.
  - Generację języka programowania.
  - Sposób kontroli typów.
  - Sposób wykonywania (kompilacja, interpretacja).
  - Poziom (języki niskopoziomowe są bardziej zbliżone pod względem budowy do działania sprzętu).
  - Przeznaczenie.

Paradygmat programowania

- Definiuje sposób patrzenia programisty na przepływ sterowania i wykonywanie programu komputerowego.
  - ⇨ Programowanie proceduralne
  - ⇨ Programowanie strukturalne
  - ⇨ Programowanie funkcyjne
  - ⇨ Programowanie imperatywne
  - ⇨ Programowanie obiektowe
  - ⇨ Programowanie uogólnione
  - ⇨ Programowanie zdarzeniowe
  - ⇨ Programowanie logiczne
  - ⇨ Programowanie aspektowe
  - ⇨ Programowanie deklaratywne
  - ⇨ Programowanie modularne

Generacje języków programowania

Generacje języków opisują zaawansowanie struktury języka, co jest równocześnie związane z łatwością posługiwania się nimi

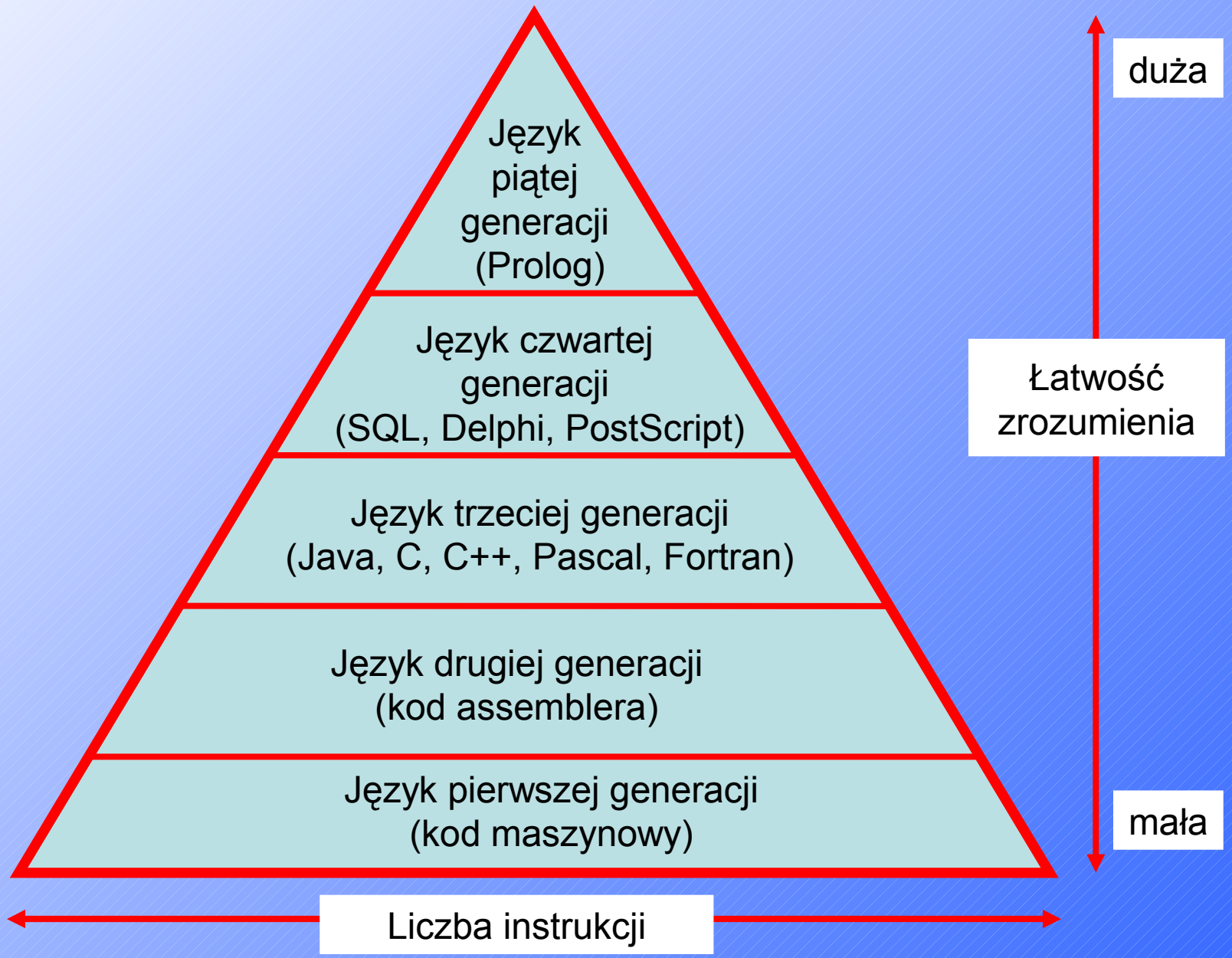
- im mniejsza liczba oznaczająca generację języka tym bardziej jest on zbliżony do sprzętu;
- im większa generacja języka tym jest on bardziej intuicyjny i niezależny od sprzętu.

- **Generacja I - język maszynowy** – lista instrukcji, z których każda jest serią binarnych liczb.
- **Generacja II - język asemblerowy** – skojarzenie kolejnych komponentów instrukcji z symbolami i nazwami, wymaga tzw. **asemblera** – programu tłumaczącego kod symboliczny na maszynowy.

Generacje języków programowania (cd.)

- **Generacja III –** instrukcje podobne do fraz i zdań języka naturalnego (angielskiego), wymaga tzw. **kompilatora** - tłumaczącego kod źródłowy na maszynowy, lub **interpretera** - iteracyjnie pobierającego kolejne instrukcje, tłumaczącego na kod wykonywalny i natychmiast go wykonującego.
- **Generacja IV** - systemy oprogramowania pozwalające na interaktywne tworzenie oprogramowania, systemy CASE.
- **Generacja V –** języki sztucznej inteligencji (LISP, PROLOG) z wbudowanymi mechanizmami wnioskowania.

Generacje języków programowania (cd.)



Sposoby kontroli typów

 **Statyczne typowanie** - wymagane zadeklarowanie typu zmiennej w kodzie źródłowym, a kontrola typów odbywa się w czasie kompilacji programu.

 **Dynamiczne typowanie –** zmienne nie mają przypisanych na sztywno typów.

- W takiej sytuacji typ zmiennej wynika z wartości jaką dana zmienna przechowuje.
- Ułatwia to operacje na zmiennych, utrudnia natomiast kontrolę integralności programu.
- Do języków stosujących dynamiczne typowanie należą między innymi: Lisp, Perl, Python, Matlab, PHP, Ruby, Erlang, Tcl, JavaScript.

Sposób wykonania

- **Kompilacja** – kod źródłowy jest tłumaczony do postaci kodu maszynowego, czyli sekwencji elementarnych operacji gotowych do bezpośredniego przetworzenia przez procesor komputera. Jeżeli dany język programowania podlega kompilacji, określany jest mianem ***kompilowanego języka programowania***.

- **Interpretacja** – kod źródłowy jest na bieżąco tłumaczony i wykonywany przez dodatkowy program zwany interpreterem. Jeżeli język podlega interpretacji, nazywany jest ***interpretowanym językiem programowania***.

Poziom

- **Język niskiego poziomu** - typ języka programowania, który w małym stopniu abstrahuje od konstrukcji jednostki centralnej komputera. Język ten wykazuje duże podobieństwo do kodu maszynowego, zaś kompilacja jest w miarę nieskomplikowana.
- **Język wysokiego poziomu** - typ języka, którego składnia i słowa kluczowe mają maksymalnie ułatwić rozumienie kodu programu dla człowieka, Kod napisany w języku wysokiego poziomu nie jest bezpośrednio „zrozumiały” dla komputera – aby umożliwić wykonanie programu napisanego w tym języku należy dokonać procesu kompilacji.

Przeznaczenie

Języki programowania ze względu na szeroko rozumiane przeznaczenie można podzielić na następujące kategorie:

- **Języki ogólnego przeznaczenia** (np.: C++, C#, Delphi, Java, Assembler)
- **Języki dziedzinowe** – języki specjalizowane :
  - skryptowe (np.; skrypty powłoki systemu operacyjnego)
  - obsługi stron internetowych - po stronie serwera (np.: PHP, JSP);
  - obsługi stron internetowych - po stronie klienta (np.: JavaScript, JScript, VBScript, ActionScript);
  - opisu dokumentów (np.: HTML, TeX);
  - opisu danych (np.: XML);
  - przetwarzania tekstu i danych (np.: AWK, Perl);
  - programowania baz danych (np.: PL/SQL).