Języki i paradygmaty programowania
Wprowadzenie

Marcin Szpyrka

Katedra Informatyki Stosowanej AGH
Interdyscyplinarne Centrum Modelowania Komputerowego UR

2013/14

Paradygmat programowania

- *Słownik języka polskiego*: Paradygmat to przyjęty sposób widzenia rzeczywistości w danej dziedzinie, doktrynie itp.
- Paradygmat programowania jest to zbiór koncepcji reprezentujących podejście do implementacji algorytmów. Definiuje on sposób patrzenia programisty na przepływ sterowania i wykonywanie programu komputerowego.
- Różne języki programowania mogą wspierać różne paradygmaty programowania, na przykład Java jest językiem programowania obiektowego, Haskell jest językiem funkcyjnym, Prolog jest językiem programowania w logice itd.
- Istnieją języki programowania wspierające kilka paradygmatów, na przykład, C++ posiada elementy programowania proceduralnego, obiektowego oraz uogólnionego.
- Ten sam algorytm można zaimplementować na różne sposoby, zgodnie z różnymi paradygmatami. Wybór paradygmatu ma często kluczowy wpływ na łatwość implementacji algorytmów.

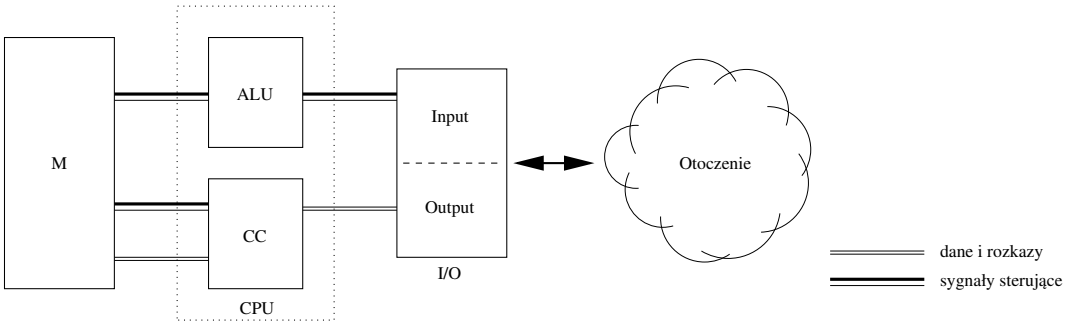
Programowanie imperatywne

First **do this** and next **do that**

W programowaniu imperatywnym program postrzegany jest jako ciąg poleceń dla komputera. Obliczenia są tu rozumiane jako sekwencja poleceń zmieniających krok po kroku stan maszyny (zawartość pamięci oraz rejestrów i znaczników procesora), aż do uzyskania oczekiwanego wyniku.

- Paradygmat ten jest odzwierciedleniem sposobu działania maszyny Von Neumana.
- O kolejności wykonywania poszczególnych instrukcji programu decydują instrukcje sterujące.
- Naturalną abstrakcją programowania imperatywnego jest programowanie proceduralne – ciąg instrukcji jest przekształcany w procedurę, która może być wywołana za pomocą pojedynczego polecenia.
- Przykłady języków wspierających programowanie imperatywne: Fortran, Algol, Pascal, Basic, C.

Model von Neumanna – przypomnienie



- Bloki funkcjonalne: pamięć (M), jednostka arytmetyczno-logiczna (ALU), centralne sterowanie (CC) oraz wejście i wyjście. Zespół przetwarzający złożony z bloków ALU i CC stanowi procesor centralny (CPU). Model ten jest swoistą implementacją maszyny Turinga.
- Dane (argumenty operacji i ich wyniki) i rozkazy są przechowywane we wspólnej pamięci.
- Interpretacja każdego rozkazu przebiega w dwóch fazach – najpierw odpowiednie słowo rozkazowe jest pobierane z pamięci do procesora, a następnie zdekodowana operacja jest wykonywana. Po wykonaniu zostaje zaadresowany następny rozkaz programu i cykl się powtarza.

Programowanie imperatywne – przykład

```c
#include <stdio.h>

int main()
{
  int s, n, i;

  printf("Podaj liczbę naturalną n: ");
  scanf("%d", &n);
  i = 1;
  s = 1;

  while (i < n)
  {
    ++i;
    s *= i;
  }

  printf("Silnia n: %d\n", s);
  return 0;
}
```


Programowanie funkcyjne

Evaluate an expression and use the resulting value for something

Programowanie funkcyjne wywodzi się dziedziny matematyki zwanej teorią funkcji. Wykonanie programu jest obliczeniem wartości funkcji, która jako listę argumentów otrzymuje kolejne funkcje, których wartości są obliczane itd.

- W programowaniu funkcyjnym stany i wartości pośrednie programu nie są zapamiętywane, a jedynie używane jako argumenty kolejnych wyrażeń – brak jest zmiennych w klasycznym rozumieniu.
- Brak jest tradycyjnie rozumianych pętli, które są instrukcjami typowo imperatywnymi.
- Konstruowanie programów polega na składaniu funkcji, często z wykorzystaniem rekurencji.
- Przykłady języków wspierających programowanie funkcyjne: Haskell, LISP, ML.

```haskell
silnia :: Int -> Int
silnia 0 = 1
silnia n = n * silnia(n - 1)
```

Programowanie w logice

Answer a question via search for a solution

W programowaniu w logice program składa się z opisu faktów, reguł wnioskowania oraz celu do którego zmierzamy, a rola komputera polega na odpowiednim zastosowaniu reguł aby znaleźć rozwiązanie.

- Wykonanie programu to próba udowodnienia celu w oparciu o podane przesłanki.
- Konstruowanie programów polega na opisywaniu co wiemy i co chcemy uzyskać.
- Paradygmat ten ma zastosowanie głównie przy programowaniu sztucznej inteligencji, aplikacjach zorientowanych na wiedzę itp.
- Przykłady języków wspierających programowanie funkcyjne: Prolog (PROgrammation en LOGique, PROgramming in LOGic).

```
1 lubi(jan, tatry).
2 lubi(jan, beskidy).
3 lubi(jerzy, beskidy).
4 lubi(jerzy, bieszczady).
5 lubi(józef, sudety).
6 bratniadusza(X, Y) :- lubi(X, S), lubi(Y, S), X \= Y.
7
8 ?- bratniadusza(jan,X).
```


Programowanie obiektowe

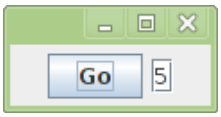
Send messages between objects to simulate the temporal evolution of a set of real world phenomena

W programowaniu obiektowym program składa się ze zbioru obiektów, komunikujących się między sobą w celu wykonywania określonych zadań. Podstawą tego paradygmatu jest obiekt, czyli kombinacja danych określających jego stan oraz operacji, które na nim można wykonywać.

- Podstawowe cechy programowania obiektowego to: abstrakcja danych, enkapsulacja, dziedziczenie i polimorfizm.
- Paradygmat obiektowy ma zastosowanie nie tylko na etapie implementacji kodu, lecz także na etapie analizy problemu i projektowania programu.
- Programowanie obiektowe jest obecnie dominującym paradygmatem programowania.
- Przykłady języków wspierających programowanie obiektowe: Smalltalk, C++, Java, Python, Ruby.

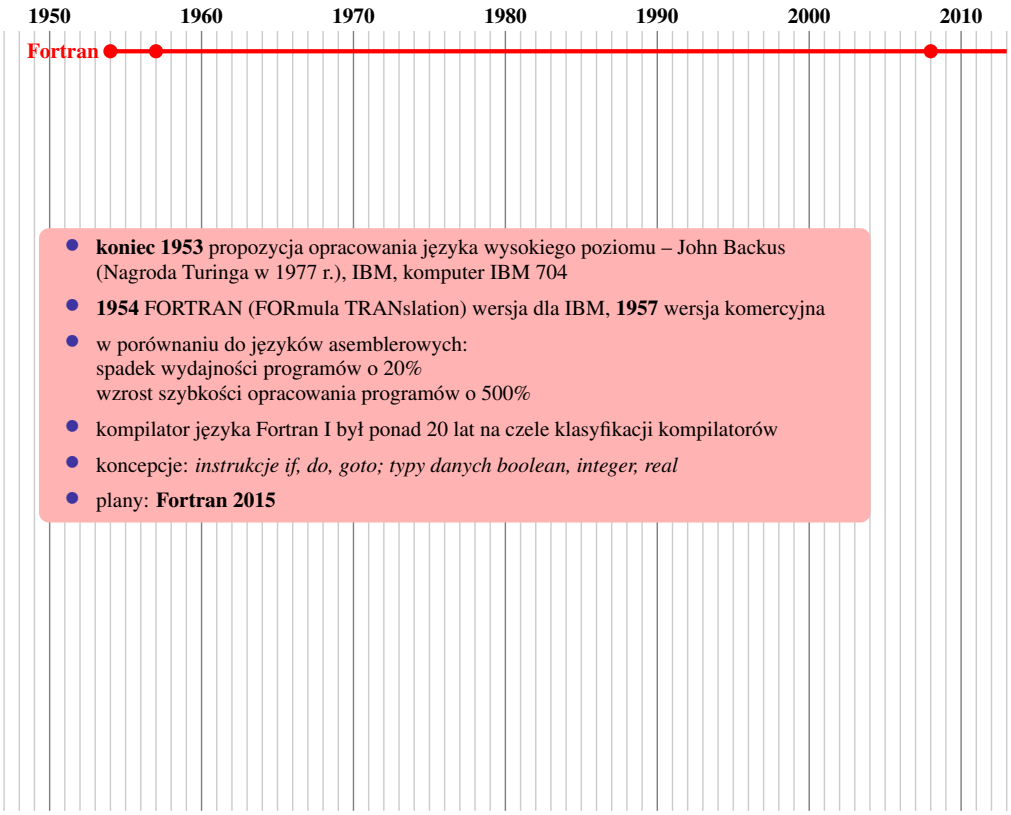
Programowanie obiektowe – przykład

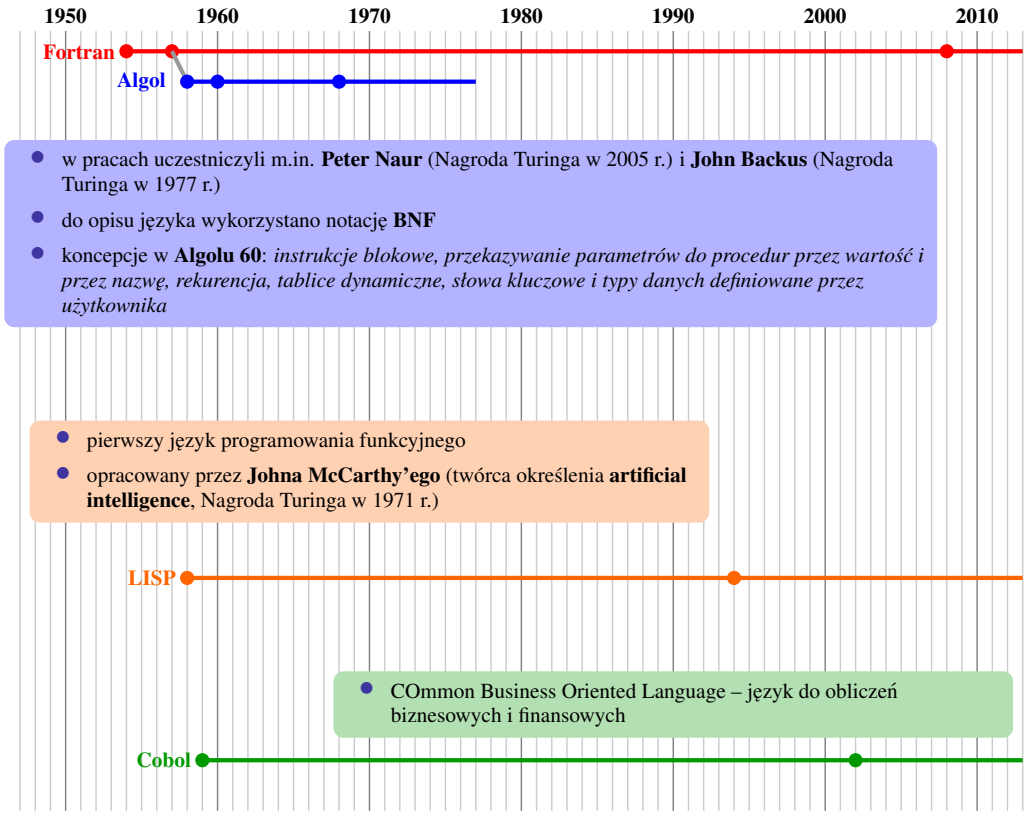
```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4  import java.util.*;
5
6  public class Dice extends JFrame {
7    private JTextField textField = new JTextField("1");
8    private JButton button = new JButton("Go");
9    private Random r = new Random();
10
11   public Dice() {
12     ActionListener al = new ActionListener() {
13       public void actionPerformed(ActionEvent e) {
14         if(e.getSource() == button)
15           textField.setText(String.valueOf(r.nextInt(6) + 1)); }
16     };
17     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
18     setLayout(new FlowLayout());
19     button.addActionListener(al);
20     add(button);
21     add(textField);
22     setSize(120,60);
23     setVisible(true);
24   }
25   static public void main(String args[]) {
26     SwingUtilities.invokeLater(new Runnable() {
27       public void run() { new Dice(); } } );
28   }
29 }
```

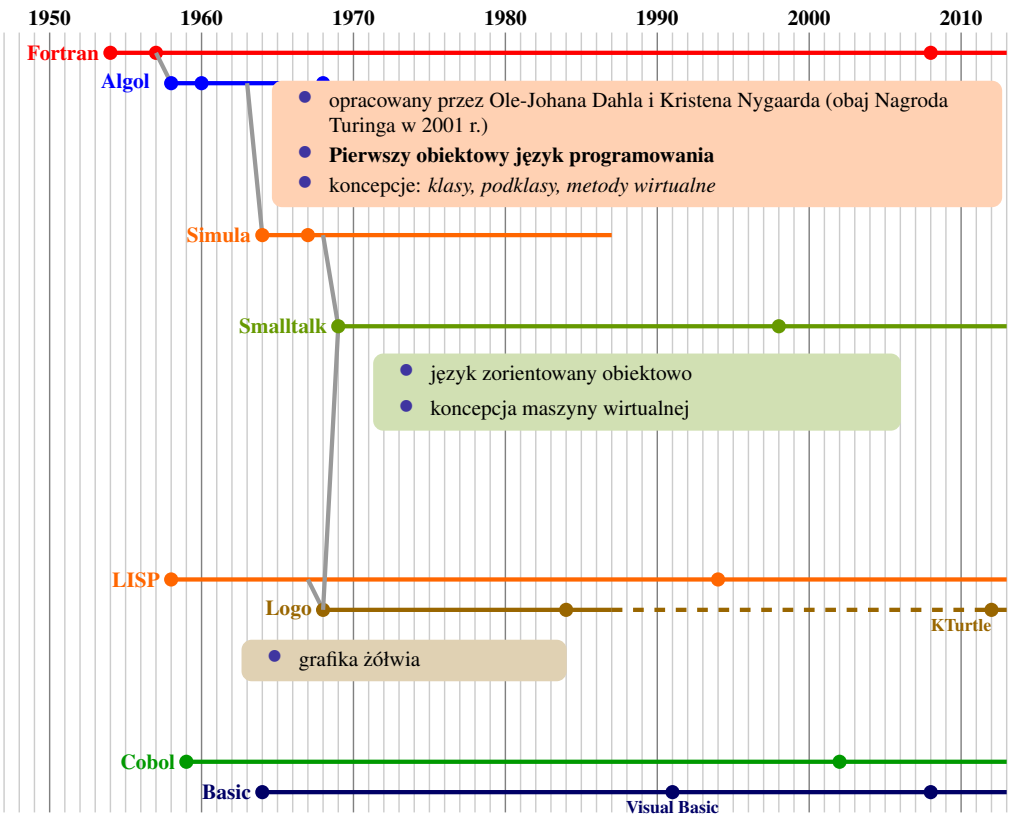


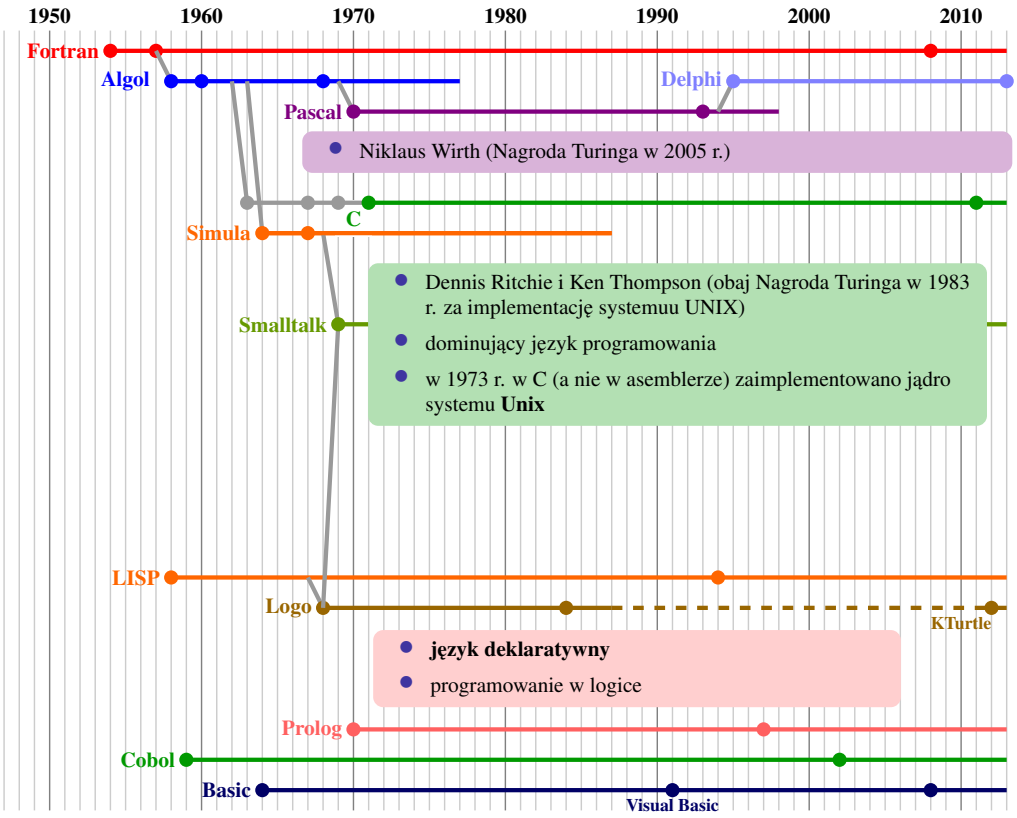
Historia języków programowania

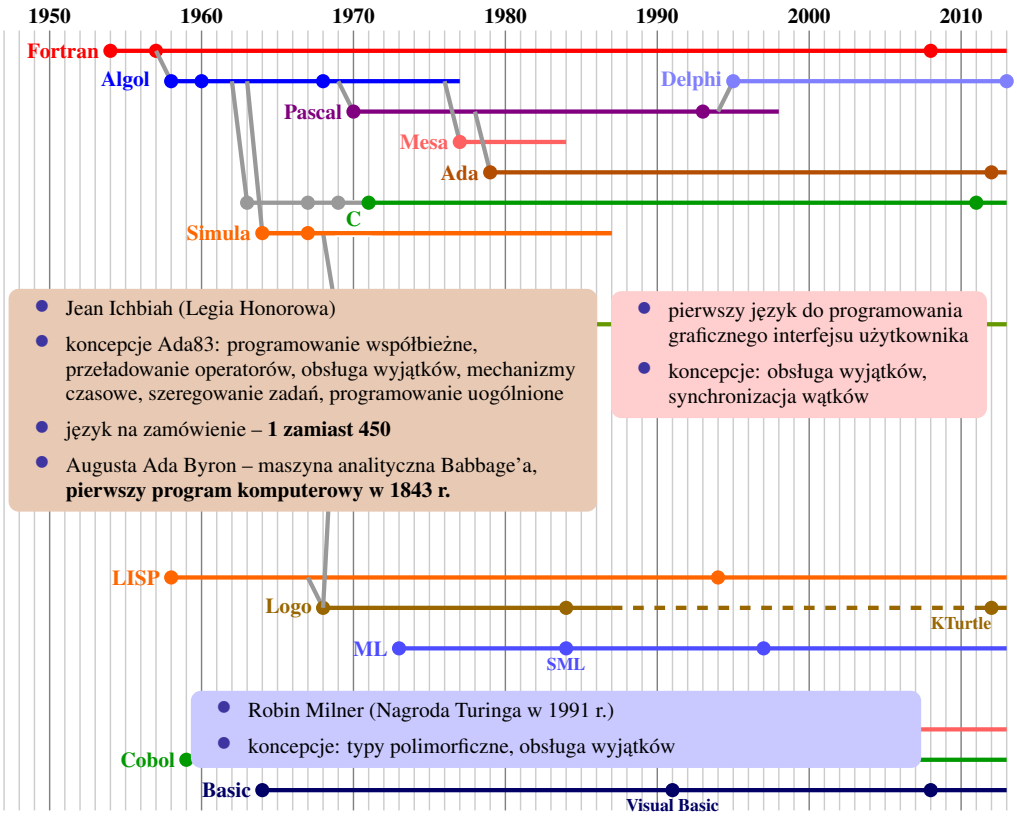
- **Pierwszy język wysokiego poziomu** powstał w 1954 r., a decyzję o jego opracowaniu podjęto jesienią 1953 r.
- Najpopularniejszy od kilku dekad język programowania powstał **ponad 40 lat temu**.
- W wyniku przeprowadzonego przez Departament Obrony Stanów Zjednoczonych studium kosztów w latach 1973-1974 stwierdzono, że używano w tym czasie **ponad 450 języków programowania**, z których żaden nie miał oficjalnego standardu.
- Podstawowe paradygmaty programowania zostały określone **przed rokiem 1980**.
- Strona **The Language List** zawiera obecnie informacje o ok. 2500 języków programowania.

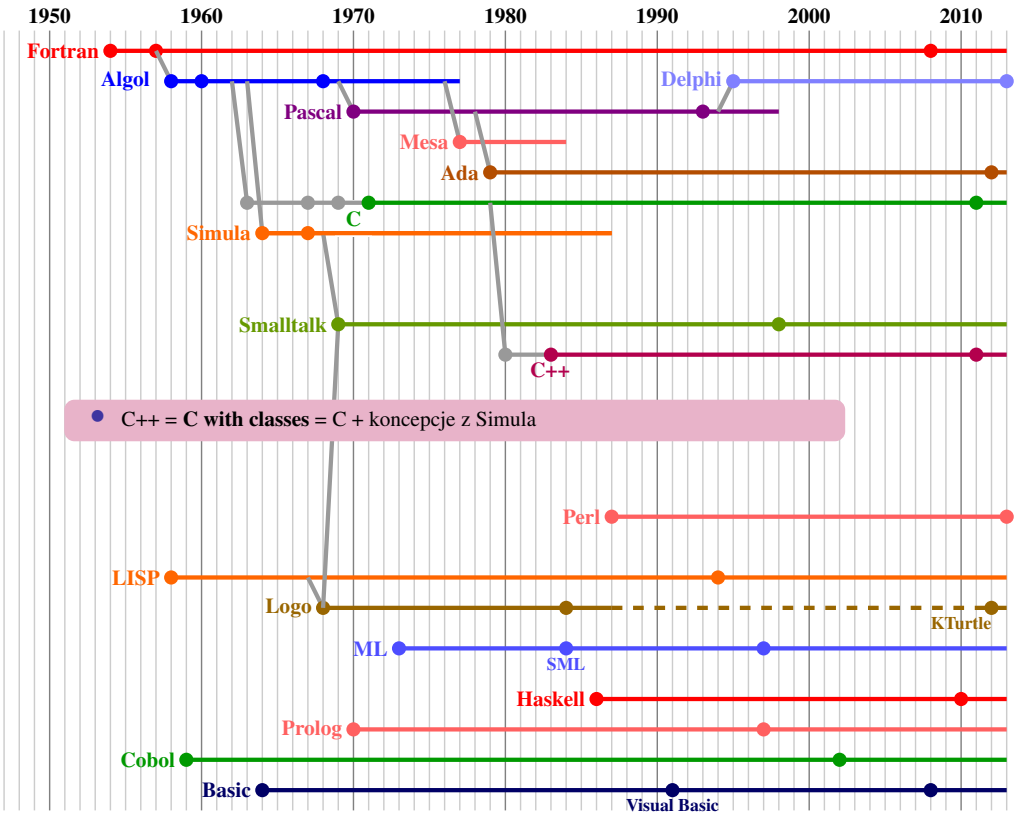

1950
1960
1970
1980
1990
2000
2010
Fortran
• **koniec 1953** propozycja opracowania języka wysokiego poziomu – John Backus (Nagroda Turinga w 1977 r.), IBM, komputer IBM 704
• **1954** FORTRAN (FORmula TRANslation) wersja dla IBM, **1957** wersja komercyjna
• w porównaniu do języków asemblerowych:
spadek wydajności programów o 20%
wzrost szybkości opracowania programów o 500%
• kompilator języka Fortran I był ponad 20 lat na czele klasyfikacji kompilatorów
• koncepcje: *instrukcje if, do, goto; typy danych boolean, integer, real*
• plany: **Fortran 2015**

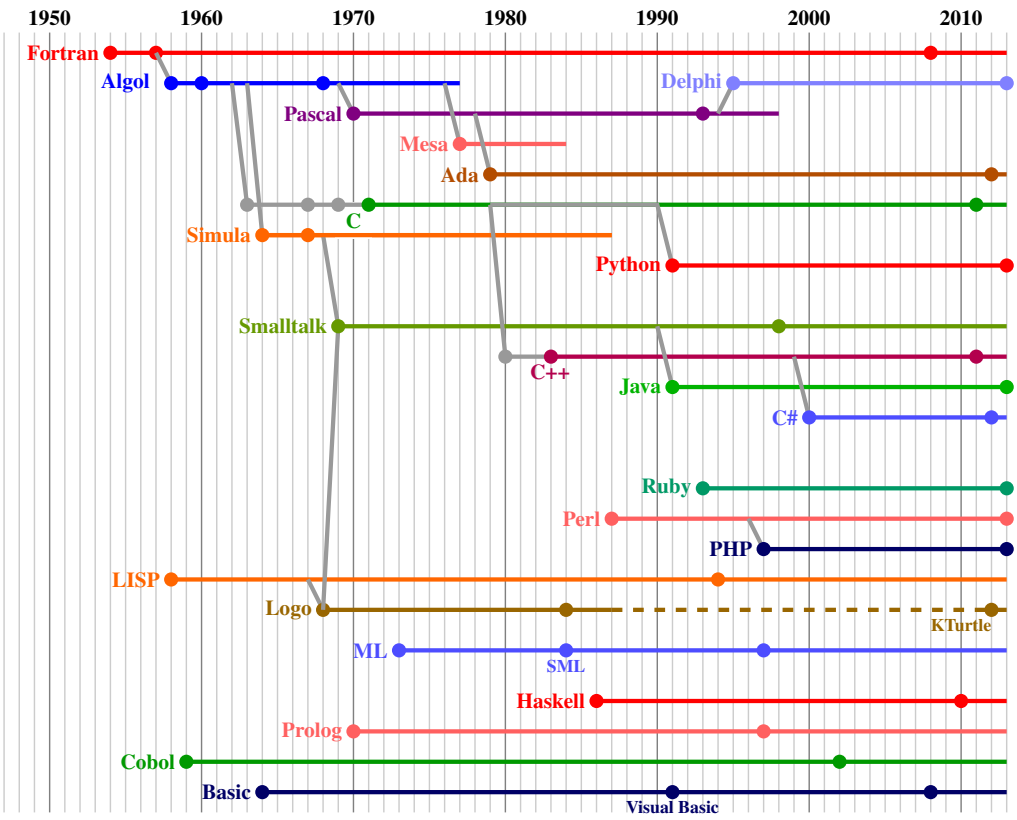

1950
1960
1970
1980
1990
2000
2010
Fortran
Algol
• w pracach uczestniczyli m.in. **Peter Naur** (Nagroda Turinga w 2005 r.) i **John Backus** (Nagroda Turinga w 1977 r.)
• do opisu języka wykorzystano notację **BNF**
• koncepcje w **Algolu 60**: *instrukcje blokowe, przekazywanie parametrów do procedur przez wartość i przez nazwę, rekurencja, tablice dynamiczne, słowa kluczowe i typy danych definiowane przez użytkownika*
• pierwszy język programowania funkcyjnego
• opracowany przez **Johna McCarthy'ego** (twórca określenia **artificial intelligence**, Nagroda Turinga w 1971 r.)
LISP
• COmmon Business Oriented Language – język do obliczeń biznesowych i finansowych
Cobol


1950
1960
1970
1980
1990
2000
2010
Fortran
Algol
opracowany przez Ole-Johana Dahla i Kristena Nygaarda (obaj Nagroda Turinga w 2001 r.)
Pierwszy obiektowy język programowania
koncepcje: klasy, podklasy, metody wirtualne
Simula
Smalltalk
język zorientowany obiektowo
koncepcja maszyny wirtualnej
LISP
Logo
KTurtle
grafika żółwia
Cobol
Basic
Visual Basic


1950
1960
1970
1980
1990
2000
2010
Fortran
Algol
Delphi
Pascal
Niklaus Wirth (Nagroda Turinga w 2005 r.)
C
Simula
Smalltalk
Dennis Ritchie i Ken Thompson (obaj Nagroda Turinga w 1983 r. za implementację systemuu UNIX)
dominujący język programowania
w 1973 r. w C (a nie w asemblerze) zaimplementowano jądro systemu Unix
LISP
Logo
KTurtle
język deklaratywny
programowanie w logice
Prolog
Cobol
Basic
Visual Basic


1950
1960
1970
1980
1990
2000
2010
Fortran
Algol
Delphi
Pascal
Mesa
Ada
C
Simula
Jean Ichbiah (Legia Honorowa)
koncepcje Ada83: programowanie współbieżne, przeładowanie operatorów, obsługa wyjątków, mechanizmy czasowe, szeregowanie zadań, programowanie uogólnione
język na zamówienie – 1 zamiast 450
Augusta Ada Byron – maszyna analityczna Babbage'a, pierwszy program komputerowy w 1843 r.
pierwszy język do programowania graficznego interfejsu użytkownika
koncepcje: obsługa wyjątków, synchronizacja wątków
LISP
Logo
KTurtle
ML
SML
Robin Milner (Nagroda Turinga w 1991 r.)
koncepcje: typy polimorficzne, obsługa wyjątków
Cobol
Basic
Visual Basic

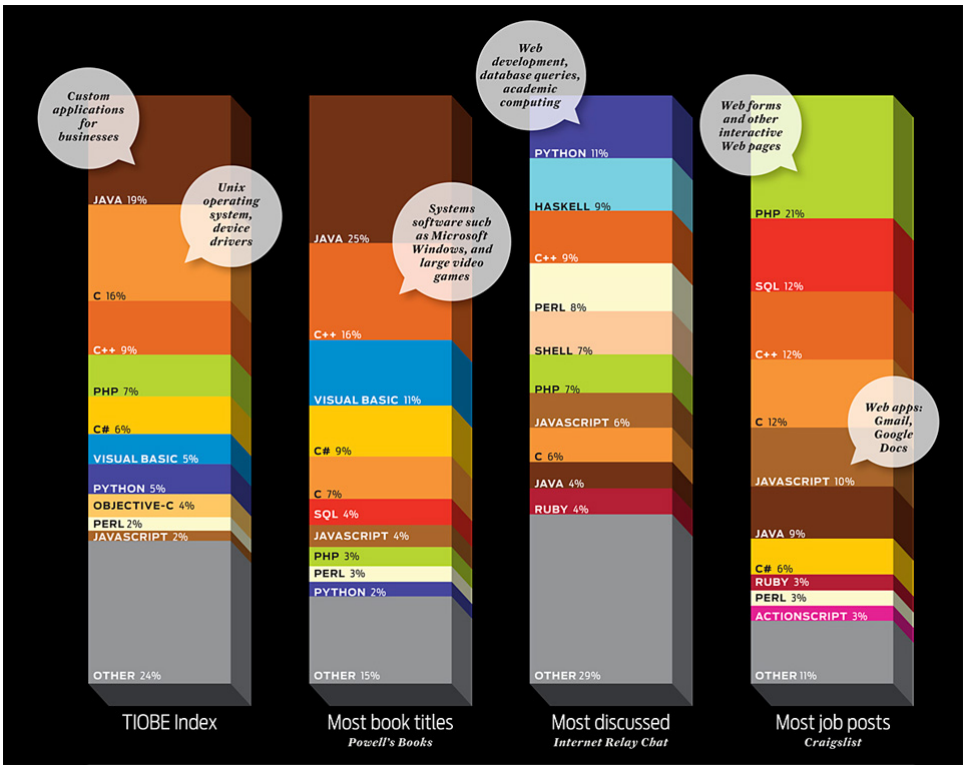

1950
1960
1970
1980
1990
2000
2010
Fortran
Algol
Delphi
Pascal
Mesa
Ada
C
Simula
Smalltalk
C++
C++ = C with classes = C + koncepcje z Simula
Perl
LISP
Logo
KTurtle
ML
SML
Haskell
Prolog
Cobol
Basic
Visual Basic



Najważniejsze języki programowania

Programming Language	Position Sep 2013	Position Sep 2008	Position Sep 1998	Position Sep 1988
C	1	2	1	1
Java	2	1	4	-
C++	3	3	2	3
Objective-C	4	42	-	-
PHP	5	5	-	-
C#	6	8	-	-
(Visual) Basic	7	4	3	7
Python	8	6	28	-
JavaScript	9	9	32	-
Transact-SQL	10	29	-	-
Lisp	16	16	13	2

Popularność języków programowania



Translatory

- Translator jest to program, który umożliwia wykonanie programów napisanych w języku różnym od języka komputera. Rozróżnia się dwa rodzaje translatorów: kompilatory i interpretery.
- Kompilator jest to program, który tłumaczy program źródłowy na równoważny program wynikowy. Program po przetłumaczeniu nie da się zmienić, jest statyczny. Kompilator jako dane wejściowe otrzymuje cały program źródłowy i przekształca go na postać wynikową.
- Interpreter jest dynamicznym translatorem. Tłumaczy on na bieżąco i wykonuje program źródłowy. Działanie interpretera polega na wyodrębnieniu niewielkich jednostek programu źródłowego, tłumaczeniu ich na pewną postać wynikową i natychmiastowym ich wykonywaniu. Proces jest cykliczny. W czasie interpretacji przechowywany jest program źródłowy.

W przypadku kompilatora program wynikowy wykonuje się szybciej i do wykonania programu wynikowego kompilator nie jest potrzebny. W przypadku interpretera mamy łatwość zmian programu, mniejszą zajętość pamięci zewnętrznej (tylko tekst źródłowy), możliwość pracy konwersacyjnej (zatrzymanie wykonania, zmiana wartości zmiennych, kontynuacja wykonania) oraz przenośność (tekst źródłowy nie jest uzależniony od architektury procesora).