Historia języków programowania

Wstęp do Informatyki i Programowania

Maciek Gębala

23 stycznia 2025

Notes

Historia języków programowania

- 1949 - pierwszy assembler.
- 1957 - FORTRAN (FORmula TRANslation) - pierwszy język strukturalny
- 1958 - ALGOL (ALGOrithmic Language)
- 1958 - LISP (LISt Processor) - pierwszy język funkcyjny
- 1959 - COBOL (COmmon Business Oriented Language)
- 1964 - BASIC (Beginner's All-purpose Symbolic Instruction Code)
- 1966 - LOGO

Notes

Historia języków programowania

- 1970 - PASCAL
- 1970 - Prolog
- 1972 - Smalltalk - wprowadzenie obiektów
- 1972 - C - język do pisania systemów operacyjnych
- 1972 - SQL - deklaratywny język zapytań do baz danych
- 1980 - Ada
- 1983 - C++, Objective C
- 1987 - Perl
- 1990 - Haskell

Notes

Historia języków programowania

- 1991 - Python
- 1991 - Visual Basic
- 1995 - Java
- 1995 - PHP (Personal Home Page)
- 1995 - JavaScript
- 2000 - C#
- 2003 - Scala
- 2009 - Go
- 2010 - Rust

Notes

Programowanie imperatywne

Programowanie imperatywne to najbardziej pierwotny sposób programowania, w którym program postrzegany jest jako ciąg poleceń dla komputera.

Obliczenia rozumiemy tu jako sekwencję poleceń zmieniających krok po kroku stan maszyny, aż do uzyskania oczekiwanego wyniku.

Stan maszyny to zawartość całej pamięci oraz rejestrów i znaczników procesora.

Sposób patrzenia związany ściśle z budową sprzętu komputerowego o architekturze von Neumanna, w którym poszczególne instrukcje (w kodzie maszynowym) to właśnie polecenia zmieniające ów globalny stan.

Notes

Paradygmaty programowania imperatywnego

Programowanie strukturalne

Program posiada struktury: instrukcje warunkowe, pętle, ... (nie używa skoków). [Fortran]

Programowanie proceduralne

Programowanie strukturalne z dodaniem możliwości podziału na podprogramy (procedury i funkcje). [Algol, C, Pascal, Ada, ...]

Programowanie modularne

Programowanie proceduralne z dodaniem możliwości tworzenia modułów (związanie ze sobą grupy funkcji/procedur oraz pewnych niezbędnych dla nich danych). [Pascal, Ada, ...]

Programowanie obiektowe

Programowanie modularne w którym łączymy ściśle ce sobą dane i metody operujące na nich, a program traktujemy jako zbiór porozumiewających się ze sobą obiektów. [Smalltalk,Java, C++, ...]

Notes

Paradygmaty programowania imperatywnego

Programowanie imperatywne jest tak stare jak komputery, a nawet starsze. Przepisy kuchenne czy sformalizowane procedury urzędowe można uznać za przejaw programowania imperatywnego. Oczywiście używane dzisiaj języki imperatywne są znacznie bogatsze niż te z czasów pionierskich.

Notes

Programowanie obiektowe

Powiązanie danych (czyli stanu) z operacjami na nich (czyli poleceniami) w całość, stanowiącą odrębną jednostkę — obiekt. Dodatkowo kontrola dostępu do danych i metod.

Mechanizm dziedziczenia, czyli możliwość definiowania nowych, bardziej złożonych obiektów, na bazie obiektów już istniejących.

Polimorfizm - możliwość używania w jednakowy sposób obiektów mających wspólnego przodka.

Notes

Programowanie deklaratywne

Program opisuje warunki jakie muszą być spełnione (jaki cel chcemy osiągnąć) a nie sekwencję kroków które chcemy osiągnąć. [SQL, Prolog, Lisp, Haskell, ...]

W większości wywodzą się z matematycznych modeli obliczeń (λ-rachunek, rezolucja, unifikacja, algebra relacji,...).

Programowanie funkcyjne sięga korzeniami okresu międzywojennego, kiedy to stworzony został rachunek lambda. Pierwsze funkcyjne języki programowania powstały w połowie lat pięćdziesiątych XX wieku. Przełom lat pięćdziesiątych i sześćdziesiątych to pierwsze propozycje programowania w logice.

Notes

Paradygmaty programowania deklaratywnego

Programowanie logiczne

Opierają się na wnioskowaniu logicznym i unifikacji. [Prolog]

Programowanie ograniczeń

Szukanie rozwiązania przez podanie zbioru (logicznych) ograniczeń na rozwiązanie. [Programowanie liniowe, programowanie całkowitoliczbowe, ...]

Programowanie funkcyjne

Programujemy funkcje, brak stanu maszyny (nie ma zmiennych i efektów ubocznych, stąd nie ma pętli i wykorzystuje się rekurencję). Wywodzi się z λ-rachunku. [Scheme, Haskell, ML, ...]

Notes

Common Lisp - przykład

Funkcja Fibonacciego

```
1  (defun fibonacci (n)
2    (if (< n 2) n (+ (fibonacci (- n 1)) (fibonacci (- n 2))))
3  )
4  ;;-----------------------------------------------------------------------
5  (defun fibonacci2 (n)
6    (labels (
7      (_fib2 (n x y)
8        (if (= n 0) x (_fib2 (- n 1) (+ x y) x))
9      ))
10     (_fib2 n 0 1)
11   )
12 )
```


Notes

Common Lisp - QuickSort

```
1  (defun quicksort (L)
2    (labels (
3      (_filter_lt (x L)
4        (if (null L) L (if (< (car L) x)
5                           (cons (car L) (_filter_lt x (cdr L)))
6                           (_filter_lt x (cdr L))))
7      )
8      (_filter_eq (x L)
9        (if (null L) L (if (= (car L) x)
10                          (cons (car L) (_filter_eq x (cdr L)))
11                          (_filter_eq x (cdr L))))
12     )
13     (_filter_gt (x L)
14       (if (null L) L (if (> (car L) x)
15                          (cons (car L) (_filter_gt x (cdr L)))
16                          (_filter_gt x (cdr L))))
17     ))
18     (if (< (length L) 2) L (append
19                              (quicksort (_filter_lt (car L) L))
20                              (_filter_eq (car L) L)
21                              (quicksort (_filter_gt (car L) L))))
22   )
23 )
```


Notes

Standard ML - przykłady

```
fun fibonacci 0 = 0
 |  fibonacci 1 = 1
 |  fibonacci n = fibonacci (n-1) + fibonacci (n-2);

fun fibonacci2 n = let
    fun fib 0 a b = a
     |  fib n a b = fib (n-1) (a+b) a
  in
    fib n 0 1
  end;
```


Notes

Standard ML - przykłady

```
local
  fun partition   []    x l e r = (l,e,r)
   |  partition (y::ys) x l e r = if y<x then
                                    partition ys x (y::l) e r
                                  else if y=x then
                                    partition ys x l (y::e) r
                                  else
                                    partition ys x l e (y::r);
in
  fun quicksort []      = []
   |  quicksort [x]     = [x]
   |  quicksort (x::xs) = let
                          val (l,e,r) = partition (x::xs) x [] [] []
                        in
                          (quicksort l) @ e @ (quicksort r)
                        end;
end;
```


Notes

Haskell - przykłady

```
fibonacci n = if n<2 then n
              else fibonacci (n-1) + fibonacci (n-2)

fibonacci2 n = let
    fib n x y
      | n==0      = x
      | otherwise = fib (n-1) (x+y) x
  in
    fib n 0 1
```


Notes

Haskell - przykłady

```
quicksort []     = []
quicksort [x]    = [x]
quicksort (x:xs) = let
    partition k [] = ([],[],[])
    partition k (x:xs) = let
        (as,bs,cs) = partition k xs
      in
        if x < k then (x:as,bs,cs)
        else if x == k then (as,x:bs,cs)
        else (as,bs,x:cs)
   in
     let (as,bs,cs) = partition x (x:xs) in
       (quicksort as) ++ bs ++ (quicksort cs)

quicksort2 [] = []
quicksort2 (x:xs) = quicksort2 [y | y <- xs, y < x] ++
                    [x] ++
                    quicksort2 [y | y <- xs, y >= x]
```


Notes

Prolog - przykłady

```
fibonacci(0,0).
fibonacci(1,1).
fibonacci(X,Y):-X>1,A is X-1,B is X-2,
                fibonacci(A,C),fibonacci(B,D),Y is C+D.

fib(0,X,_,X).
fib(N,X,Y,Z):-N>0,N1 is N-1,A is X+Y,fib(N1,A,X,Z).

fibonacci2(N,Z):-fib(N,0,1,Z).

gcd(X,Y,X):-Y=:=0,!.
gcd(X,Y,Z):-Y>0,T is X mod Y,gcd(Y,T,Z).
```


Notes

Prolog - przykłady

```
partition(_,[],[],[]).
partition(K,[A|AS],[A|X],Y):-A=<K,partition(K,AS,X,Y).
partition(K,[A|AS],X,[A|Y]):-A>K,partition(K,AS,X,Y).

quicksort([],[]).
quicksort([X|XS],Y):-partition(X,XS,A,B),
                     quicksort(A,C),quicksort(B,D),
                     append(C,[X|D],Y).
```


Notes

Notes

Notes