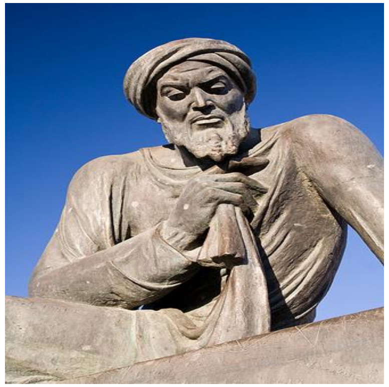
Programming Languages

Design and Implementation

What is a programming language?

Algorithm

Abu Ja’far Muhammad ibn Musa al-Khorezmi (“from Khorezm”)



- Lived in Baghdad around 780 – 850 AD
- Chief mathematician in Khalif Al Mamun’s “House of Wisdom”
- Author of “A Compact Introduction To Calculation Using Rules Of Completion And Reduction”

كتاب الخوارزمي

مما الفه الشيخ الاجل ابو عبد الله

محمد بن موسى الخوارزمي رضي الله عنه واثابه ورحمه

ملكه لمستر ذنوبه وخطاياه العبد الفقير

الى الله الغني به خطاب بن محمد بن علي

ابن حسن بن علي بن محمد بن علي بن الحسن بن

جعفر بن الحسن بن يحيى بن ابراهيم بن محمد بن

ابراهيم بن احمد بن المغيرة بن عمران بن عاصم بن

الوليد بن عتبة بن ربيعة بن عبد شمس بن

عبد مناف

نفعه الله بالعلم والعمل

الصالحين

وحسبنا الله ونعم الوكيل

[illegible]

Calculus of Thought

Gottfried Wilhelm Leibniz

- 1646 – 1716
- Inventor of calculus and binary system
- “**Calculus ratiocinator**”: Human reasoning can be reduced to a formal symbolic language
- Invented a mechanical calculator

Formalisms for Computation

Predicate logic

- Logic programming
  - Computation as logical deduction

Turing machines

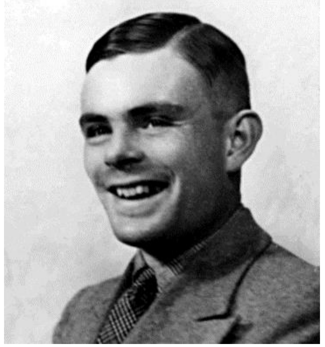
- Imperative programming
  - Sequences of commands, explicit state transitions, update via assignment

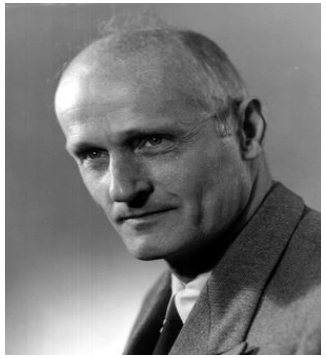
Lambda calculus

- Functional programming
  - Pure expression evaluation, no assignment operator

Recursive functions & automata

- Regular expressions, finite-state machines





Church's Thesis

All these different syntactic formalisms describe the same class of mathematical objects

- Church's Thesis: "Every effectively calculable function (effectively decidable predicate) is general recursive"

- Turing's Thesis: "Every function which would be naturally regarded as computable is computable by a Turing machine"

Recursion, Lambda-calculus and Turing machines are equivalent in their expressive power

Why is this a "thesis" and not a "theorem"?

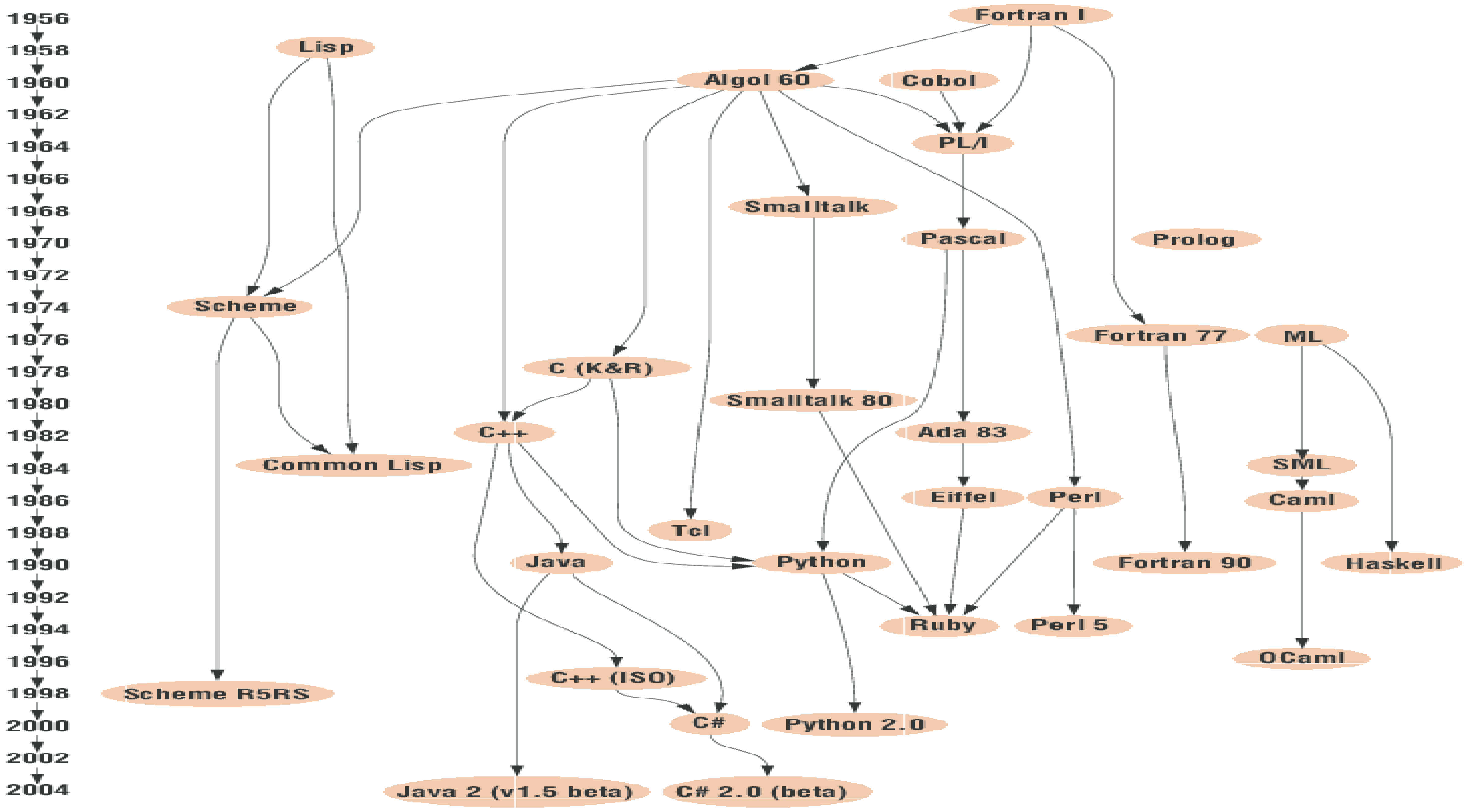
Programming Languages

Formal notation for specifying computations

- Designed by
    - a single person :C++
    - small groups : C and Java
    - large groups: ADA
- Syntax (usually specified by a context-free grammar)
- Semantics for each syntactic construct
- Practical implementation on a real or virtual machine
    - Translation vs. compilation vs. interpretation
        - C++ was originally translated into C by Stroustrup’s Cfront
        - Java originally used a bytecode interpreter, now native code compilers are commonly used for greater efficiency
        - Lisp, Scheme and most other functional languages are interpreted by a virtual machine, but code is often precompiled to an internal executable for efficiency

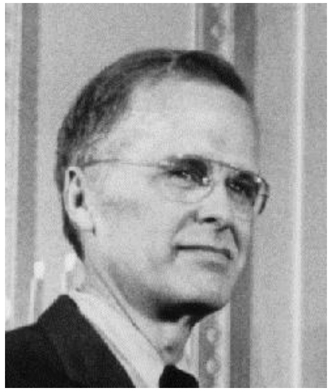
High-level languages

- (More or less) Independent of the machines on which programs are executed
- Implemented by
  - compiling programs into machine language
  - interpreting them directly
  - some combination of compilation and interpretation


1956
1958
1960
1962
1964
1966
1968
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
Fortran I
Lisp
Algol 60
Cobol
PL/I
Smalltalk
Pascal
Prolog
Scheme
Fortran 77
ML
C (K&R)
Smalltalk 80
C++
Ada 83
Common Lisp
SML
Eiffel
Perl
Caml
Tcl
Java
Python
Fortran 90
Haskell
Ruby
Perl 5
OCaml
C++ (ISO)
Scheme R5RS
C#
Python 2.0
Java 2 (v1.5 beta)
C# 2.0 (beta)

FORTRAN

- FORmula TRANslator
- 1954 – 1957
- Designed for numerical computations
  - It is still used for some numerical applications
- Developed at IBM by a team lead by John Backus



(well known for the Backus normal form, BNF, for defining context-free grammars)

برای نمایش گرامر های مستقل از متن

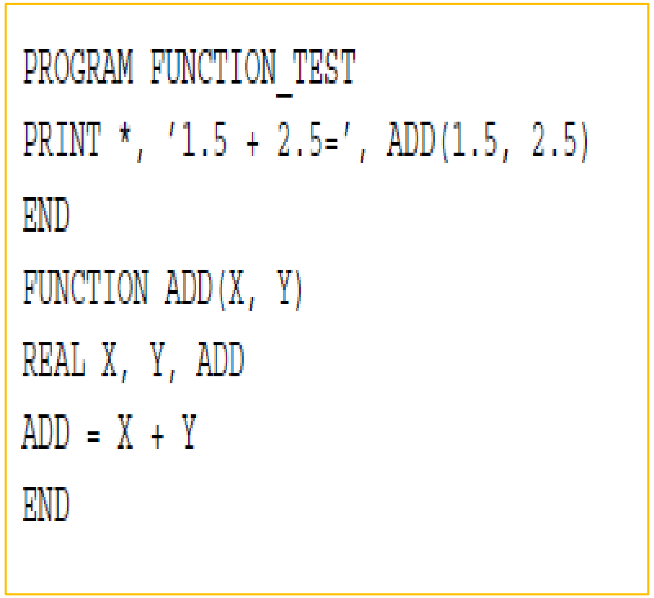
- A procedural, imperative language

FORTRAN

- It became possible to use ordinary mathematical notation in expressions:

  i+2*j

- It introduced arrays and procedures (“subroutines”) with parameters
- Many early Fortran compilers stored numbers 1, 2, 3 . . . in memory locations
  - Programmers could change the values of numbers if they were not careful!
- It was not possible for a Fortran subroutine to call itself
  - This required memory management techniques that had not been invented



```
PROGRAM FUNCTION_TEST
PRINT *, '1.5 + 2.5=', ADD(1.5, 2.5)
END
FUNCTION ADD(X, Y)
REAL X, Y, ADD
ADD = X + Y
END
```

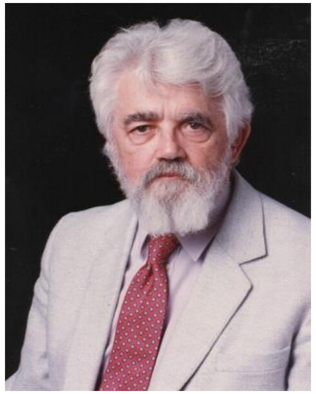
COBOL

- COmmon Business-Oriented Language (1959)
- Developed by Grace Murray Hopper
- Designed for business applications
- The syntax of Cobol was intended to resemble that of common English
  - Like Fortran, many Cobol programs are still in use today

- A procedural, imperative language and since 2002, an object-oriented language
- Memory management is entirely static

LISP

- LISt Processor (1960)
- Designed for non-numeric and AI applications
- Developed by a group led by John McCarthy at MIT
- The Scheme language was born from a variant of LISP
- A functional language
- Pioneered many PL concepts
  - Automated memory management (garbage collection)
  - Dynamic typing
- Using stack memory management and recursive functions or procedures
- The first concrete implementation of lambda calculus as a programming language



ALGOL 60

یک متغیر را برای یک procedure پاس دهیم
دقیقا همان متنی که به آن پاس دادیم
داخل کد تابع یا procedure کپی می شود
و بنابراین مثل این است که اصل obj در اختیار procedure قرار میگیرد

- Designed in 1958-1960
- Algol is a family of imperative languages
  - Predominant in the academic world in the 1960s
- Great influence on modern languages
- Three fundamental contributions
  - Parameter passing by name
  - Lexical scoping: begin … end or {…}
  - Formally specified syntax (BNF)
- Recursive procedures
- Dynamic memory management

تفاوتcall by name باcall by reference :
با سطحی از پوینتر ها کار می کنیم که به ما اجازه
داشتن متغیرهای هم نام در scopeهای متفاوت را می دهد

Algol 60 Sample

```
real procedure average(A,n);
  real array A; integer n;
  begin
     real sum; sum := 0;
     for i = 1 step 1 until n do
           sum := sum + A[i];
     average := sum/n
  end;
```

no array bounds

no ; here

set procedure return value by assignment

Algol Oddity

- Question
  - Is x := x equivalent to doing nothing?
- Interesting answer in Algol

```
integer procedure p;
begin
        …
        p := p
        …
 end;
```

  - Assignment here is actually a recursive call

Algol 60 Pass-by-Name

- Substitute text of actual parameter
  - Unpredictable with side effects!
- Example

```
procedure inc2(i, j);
   integer i, j;
   begin
      i := i+1;
      j := j+1
   end;
inc2 (k, A[k]);
```

→

```
begin
   k := k+1;
   A[k] := A[k] +1
end;
```

Is this what you expected?

ALGOL 68

- New terminologies
  - Types were called “modes”
  - Arrays were called “multiple values”

vW

نحوه نمایش گرامرهای حساس به متن

- vW grammars instead of BNF
  - Context-sensitive grammar invented by A. van Wijngaarden
- Parameter passing
  - Eliminated pass-by-name (introduced pass-by-reference)
  - Pass-by-value and pass-by-reference using pointers
- Considered difficult to understand

Pascal

در زبان های functional ورودی یک تابع زمانی که
تابع یا procedure را تعریف می کنیم میتواند یک procedure
دیگر باشد(قرار است یک بخشی از کار بدنه را انجام دهد)
محدودیت: اگر به صورت بالا تعریف کردیم تابعی که به عنوان
ورودی تعریف می شود ، نمی تواند خودش پارامتر از جنس procedure
داشته باشد

- Designed by Niklaus Wirth (1968–69)
  - 1984 Turing Award

خط به خط اجرا میشود و pc زیاد می شود تا به دستور بعدی برسد

- A procedural-imperative language
- Evolved from Algol W
- Revised type system of Algol
  - Good data-structuring concepts : records, ...
  - More restrictive than Algol 60/68 –
    - Procedure parameters cannot have procedure parameters
- Popular teaching language
- Simple one-pass compiler

اگر کامپایلر با یک بار خواندن فایل ورودی خروجی معادل را تولید کند
one-pass
گفته می شود در غیر این صورت multipass گفته می شود (مثل جاوا)

Procedure parameters in Pascal

- Allowed procedures

```
Proc1(i,j: integer);

procedure Proc2(procedure P(i:integer); i,j: integer);
```

- Not allowed procedure

```
NotA(procedure Proc3(procedure P(i:integer)));
```

SIMULA 67

- Developed in the 1960s by Ole-Johan Dahl and Kristen Nygaard

- Designed for doing simulations

اجرای کند

- First object-oriented language
  - Objects and classes
  - Subclasses and inheritance

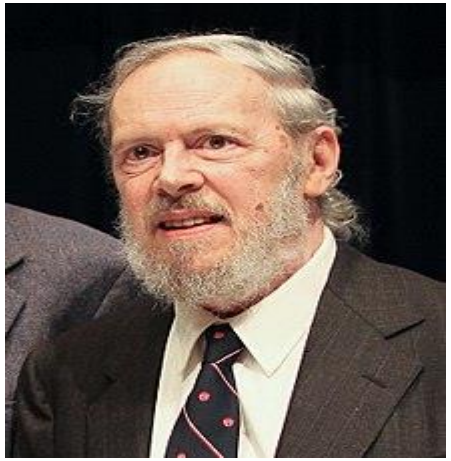
C

- Designed for writing Unix by Dennis Ritchie (1969 - 1973)
- Evolved from B, which was based on BCPL
  - B was an untyped language
  - C adds some checking

هدف: زبانی داشته باشند برای طراحی سیستم عامل یونیکس

کارکردن با پوینترها به صورت صریح امکان پذیر است

- Relation between arrays and pointers
  - An array is treated as a pointer to first element
  - E1[E2] is equivalent to pointer dereference *((E1)+(E2))
  - Pointer arithmetic is not common in other languages
- A procedural-imperative language



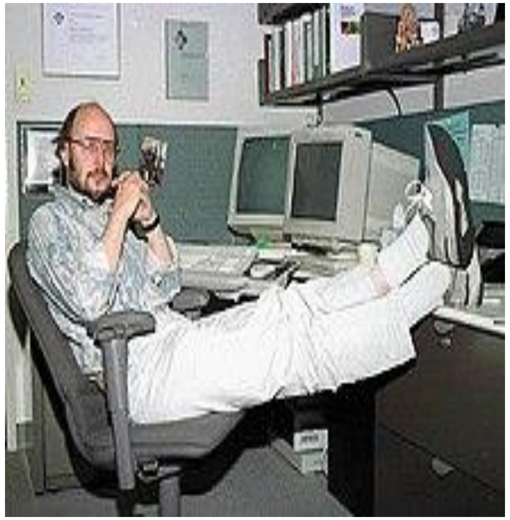
C++

- Developed by Bjarne Stroustrup (1979)
- Influenced by Simula
- Originally translated into C using Cfront, then native compilers
  - GNU g++
- An imperative, object-oriented language
- Several PL concepts
  - Multiple inheritance
  - Templates / generics
  - Exception handling

template resolve in compile time

generic resolve in runtime

کدی بنویسیم که وابسته به یک type خاص نباشد



JAVA

یک سخت افزار که پردازش های دیجیتال را به جای آنالوگ بودن سیگنال های انجام دهد

- 1991-1995 (James Gosling)
  - Originally called Oak,
  - intended for set top boxes
- A concurrent, object-oriented
- Mixture of C and Modula-3
  - Unlike C++
    - No templates (generics), no multiple inheritance, no operator overloading
  - Like Modula-3 (developed at DEC SRC)
    - Explicit interfaces, single inheritance, exception handling, built-in threading model
    - automatic garbage collection (no explicit pointers!)
- “Generics” added later

تعریف تابع برای یک type خاص

مثال عمل جمع

Other Important Languages

- Algol-like
  - Modula, Oberon, Ada
- Functional
  - ISWIM, FP, SASL, Miranda, Haskell, LCF, ML, Caml, Ocaml, Scheme, Common LISP
- Object-oriented
  - Smalltalk, Objective-C, Eiffel, Modula-3, Self, C#, CLOS
- Logic programming
  - Prolog, Gödel, LDL, ACL2, Isabelle, HOL

… And More

- Data processing and databases
  - Cobol, SQL, 4GLs, XQuery
- Systems programming
  - PL/I, PL/M, BLISS
- Specialized applications
  - APL, Forth, Icon, Logo, SNOBOL4, GPSS, Visual Basic
- Concurrent, parallel, distributed
  - Concurrent Pascal, Concurrent C, C*, SR, Occam, Erlang, Obliq

What’s Driving PLs Evolution?

- Constant search for better ways to build software tools for solving computational problems
  - Many PLs are general purpose tools
  - Others are targeted at specific kinds of problems
    - For example, massively parallel computations or graphics

کد نویسی بهینه تر و راحتر

نیازهای جدید

- Useful ideas evolve into language designs
  - Algol → Simula → Smalltalk → C with Classes → C++
- Often design is driven by expediency
  - Scripting languages: Perl, Tcl, Python, PHP, etc.

What Do They Have in Common?

- Lexical structure and analysis
  - Tokens: keywords, operators, symbols, variables
  - Regular expressions and finite automata
- Syntactic structure and analysis
  - Parsing, context-free grammars
- Pragmatic issues
  - Scoping, block structure, local variables
  - Procedures, parameter passing, iteration, recursion
  - Type checking, data structures
- Semantics
  - What do programs mean and are they correct

References

- http://www.cs.utexas.edu/~shmat/courses/cs345/
- Gabbrielli, Maurizio, and Simone Martini. *Programming languages: principles and paradigms.* Springer Science & Business Media.