プログラミング言語論
第二回

プログラミング言語の歴史

コンピュータ・コンピュータ言語の歴史を学ぶ目的

・ある機能・表現方法がそこにある理由が分かる
  – 必要なら、改善方法が分かる

・あって不思議ではない、ある機能・表現方法がない理由が分かる
  – かつて試みられ、そして受け入れられなかった理由が分かる
  – 改めて試みるとき、同じ失敗を繰り返さない可能性がある

・大きな、概念的な流れは繰り返す

歴史

- 歴史以前: 史上初のプログラマたち
- 1940年代: Von Neumann と Zuse
- 1950年代: 最初のプログラミング言語
- 1960年代: プログラミング言語の爆発
- 1970年代: 単純さ, 抽象性, 教育学習
- 1980年代: 統合と新しい方向性
- 1990年代: インターネットWWW
- 2000年代:
- 2010年代:
- 2020年代:

「プログラム言語」という視点から見ると、2000年代、2010年代はどう特徴付けられるか？

歴史以前: 最初のプログラマ

- Jacquard 織機. 1800年代初
  – カード上のパターンをデザインに翻訳
- Charles Babbage の解析エンジン (1830年代から40年代)
  プログラムはデータと命令が書かれたカード
- Ada Lovelace – 最初のプログラマ
  "The engine can arrange and combine its numerical quantities exactly as if they were letters or any other general symbols; And in fact might bring out its results in algebraic notation, were provision made."

ENIAC

Electronic Numerical Integrator And Computer

ENIAC was the first general-purpose electronic computer

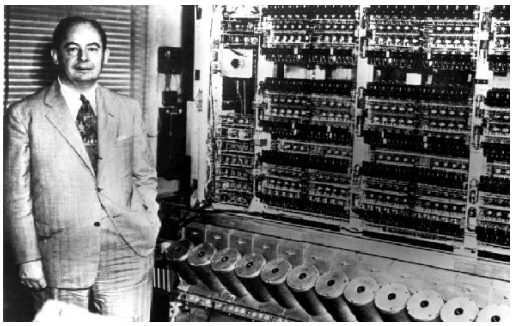
The ENIAC contained 17,468 vacuum tubes, along with 70,000 resistors, 10,000 capacitors, 1,500 relays, 6,000 manual switches and 5 million soldered joints. It covered 1800 square feet (167 square meters) of floor space, weighed 30 tons, consumed 160 kilowatts of electrical power.

In one second, the ENIAC (one thousand times faster than any other calculating machine to date) could perform 5,000 additions, 357 multiplications or 38 divisions.

2019年3月の物価/1946年の物価=254/18; $500,000=847MYen (1USD=120JPYとして)

1940年代: Von Neumann と Zuse

John Von Neumann が彼のチームと作ったコンピュータはプログラム記憶方式とCPUをもつEDSACであった。ただし、結線によってもプログラムできた



Von Neuman と EDSAC

Konrad Zuse と Plankalkul

Konrad Zuse が研究を行ったのは Plankalkul (plan calculus), 最初のアルゴリズム記述用プログラム言語であり、自然界が成立するための前提条件を理論的に求めるためであった.

その7年前に, Zuse は世界最初の2進電子計算機Z1を組み立てていた。1941年には. 動作するプログラム制御する電子機械的コンピュータZ3を作った.

Z4, 彼が作った最も緻密なもののみが第二次世界大戦を生き延びた.

1940年代: Von Neumann と Zuse

- Konrad Zuse (Plankalkul)
  - ドイツ – 戦争のために隔絶されていた
  - 1945年ごろ Plankalkul (program calculus) を定義したが実装はされなかった.
  - この言語を用いてアルゴリズムを書いた. その中にはチェスプログラムある.
  - 出版されたのは1972であった.
  - 先進的なデータタイプもある. 例えば
    - » 浮動小数点, 2の補数と隠れビットがある 説明必要ですか？
    - » 配列
    - » レコード（ネストすることもできる）

Plankalkul notation

A(7) := 5 * B(6)

```
    |  5  *  B  =>  A
V   |        6        7          (添え字)
S   |       1.n      1.n         (データ型)
```


Machine Code (1940年代)

- 最初期のコンピュータは機械語でプログラムされていた.
- すべて、数字！.
- 機械語で何が悪いか? まぁ、どれもこれも！
  - 読解性が低い
  - 変更が容易ではない
  - 式のプログラムが気が遠くなるほど大変
  - ハードウェアの欠点を引き継いでいる.
    例：インデックスがない、浮動小数点数がない

説明必要ですか？

擬似コード Pseudocodes (1949)

- Short Code または SHORTCODE - John Mauchly, 1949.
- 数学的問題に対する擬似コードの翻訳機, Eckert と Mauchly の BINAC, そして後に UNIVAC I と IIの上で.
- 恐らく, 高水準言語への最初の試み.
- 式は左から右にコード化された, e.g.:

```
X0 = sqrt(abs(Y0))
00 X0 03 20 06 Y0
```

- 演算子の例:

```
01 –      06 abs       1n (n+2)乗（べき乗）
02 )      07 +         2n (n+2)番目の根
03 =      08 pause     4n if <= n
04 /      09 (         58 print & tab
```


今では、別の意味で使用します

さらに Pseudocodes

Speedcoding; 1953-4

- 擬似コードのインタープリタ. 数学用. IBM 701, IBM 650.
- 開発者は John Backus
- 算術と数学的関数用の擬似命令
- 条件付と条件無しジャンプ 説明必要ですか？
- 配列アクセス用に自動増分のレジスタ 説明必要ですか？
- 遅い. しかし数学部分が遅いため
- インタプリタが大きくユーザエリアは 700 語

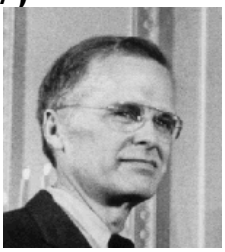
Laning and Zierler System - 1953

- the MIT Whirlwind に実装
- 最初の “algebraic” コンパイラシステム
- 添字つき変数, 関数呼び出し, 式の翻訳
- 他のハードウェアにはポートされず

1950年代:
最初のプログラミング言語

- Pseudocodes: アセンブラ語へのインタープリタ
- Fortran: 最初の高水準(higher level)プログラミング言語
- COBOL: 最初のビジネス向け言語
- Algol: これまで最も影響力が大きかったプログラミング言語
- LISP: 手続き的パラダイムから離れた最初のプログラミング言語
- APL: 革命的！

Fortran (1954-57)



- FORmula TRANslator
- 開発は IBM にて. John Backus が指導. 科学技術計算が主目的
- コンピュータの使い方を劇的かつ永久的に変革
- 継続的に進化、新機能・新概念の追加.
  - FORTRAN II, FORTRAN IV, FORTRAN 66, FORTRAN 77, FORTRAN 90
- これまで常に最も効率よく、高速度のコードを生成してきた
- いまだにポピュラー, e.g. スーパーコンピュータ用

Fortran 0 and 1

FORTRAN 0 – 1954 (実装せず)

FORTRAN I - 1957

新 IBM 704 用に設計, index registers と浮動小数点機構有

IBM 704 (Wikipedia)

開発時の状況:
コンピュータは小さく低信頼性
用途は科学技術計算
プログラミング方法論やツールはまだなし
機械が効率的に利用できることが最重要

状況が設計に与えた影響
- 動的記憶が不要
- 配列と数えながらのループのよい実装が必要
- 文字列, 十進数演算, 強力な入出力(ビジネス用途)は不要

Fortran I 特徴機能

- 識別名は最大6文字
- 最後にテストをする計数ループ (DO)
- 書式付 I/O
- プログラムが定義する副プログラム
- 3分岐文 (算術 IF)
  IF (ICOUNT-1) 100, 200, 300
- データ型定義文はなし
  識別名の最初の文字が
  i, j, k, l, m or n であれば整数型,
  その他はすべて浮動小数点数型
- 分割コンパイルはなし
- 400 行以上のプログラムが正しくコンパイルされることは稀であった, 原因は IBM 704 の低信頼性
- 出力されたコードの実行速度は速い
- ほどなく、広く使われるようになった


```
C AREA OF A TRIANGLE WITH A STANDARD SQUARE ROOT FUNCTION
C INPUT - CARD READER UNIT 5, INTEGER INPUT
C OUTPUT - LINE PRINTER UNIT 6, REAL OUTPUT
C INPUT ERROR DISPLAY ERROR OUTPUT CODE 1 IN JOB CONTROL LISTING
      READ INPUT TAPE 5, 501, IA, IB, IC
  501 FORMAT (3I5)
C IA, IB, AND IC MAY NOT BE NEGATIVE
C FURTHERMORE, THE SUM OF TWO SIDES OF A TRIANGLE
C IS GREATER THAN THE THIRD SIDE, SO WE CHECK FOR THAT, TOO
      IF (IA) 777, 777, 701
  701 IF (IB) 777, 777, 702
  702 IF (IC) 777, 777, 703
  703 IF (IA+IB-IC) 777,777,704
  704 IF (IA+IC-IB) 777,777,705
  705 IF (IB+IC-IA) 777,777,799
  777 STOP 1
C USING HERON'S FORMULA WE CALCULATE THE
C AREA OF THE TRIANGLE
  799 S = FLOATF (IA + IB + IC) / 2.0
      AREA = SQRT( S * (S - FLOATF(IA)) * (S - FLOATF(IB)) *
     +     (S - FLOATF(IC)))
      WRITE OUTPUT TAPE 6, 601, IA, IB, IC, AREA
  601 FORMAT (4H A= ,I5,5H  B= ,I5,5H  C= ,I5,8H  AREA= ,F10.2,
     +        13H SQUARE UNITS)
      STOP
      END
```


Fortran II, IV と 77

FORTRAN II - 1958
- 独立にコンパイル
- バグフィックス

FORTRAN IV - 1960-62
- 明示的な型宣言
- 論理的な選択 (IF) 文(分岐条件が論理式)
- 副プログラムの名称が実引数にできる
- ANSI 標準 1966

FORTRAN 77 - 1978
- 文字列の取り扱い
- 論理的なループ制御 (WHILE) 文
- IF-THEN-ELSE 文

Fortran 90 (1990)

多数の機能を付加. それらはより現代的なプログラミング言語のもの. 例えば

- ポインタ
- 再帰
- CASE 文
- 引数の型チェック
- 配列のポインタのさまざまな操作, DOTPRODUCT, MATMUL, TRANSPOSE, etc
- 配列の動的割付・解放
- レコードの一つのフォーム (導出型)
- Module 機能 (Ada のパッケージと類似)

COBOL

- COmmon Business Oriented Language
- Principal mentor: (Rear Admiral Dr.) Grace Murray Hopper (1906-1992).
  Rear Admiral: 海軍少将
- 基になったもの *FLOW-MATIC*. その特徴は:
  - 変数名は12文字まで, ハイフンが可能
  - 算術演算に、英語名が使用されている
    - add, multiply etc.
  - データとコードはまったく別
  - 動詞が、どの文についてもその先頭にくる
- CODASYL 委員会 (Conference on Data Systems Languages) が開発主体

COBOL

COBOL

第一回 CODASYL Design Meeting - May 1959

設計目標:

- 簡単な英語文に見えるべし
- 簡単に使えるべし, たとえその結果記述力が落ちても
- コンピュータユーザ基盤を広げるべし
- 現在のコンパイラ技術に独立であるべし

設計委員会メンバーは, コンピュータベンダと DoD から

設計上の課題: 数式をどうする? 添え字はどうする? ベンダ間の争い


```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. HELLO-WORLD.
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY 'Hello, world.'.
           STOP RUN.


       ADD YEARS TO AGE.
       MULTIPLY PRICE BY QUANTITY GIVING COST.
       SUBTRACT DISCOUNT FROM COST GIVING FINAL-COST.
```

```
000100 IDENTIFICATION          DIVISION.
000200 PROGRAM-ID.             HELLOWORLD.
000300 DATE-WRITTEN.           02/05/96  21:04.
000400*    AUTHOR              BRIAN COLLINS
000500 ENVIRONMENT             DIVISION.
000600 CONFIGURATION           SECTION.
000700 SOURCE-COMPUTER.        RM-COBOL.
000800 OBJECT-COMPUTER.        RM-COBOL.
000900
001000 DATA                    DIVISION.
001100 FILE                    SECTION.
001200
100000 PROCEDURE               DIVISION.
100100
100200 MAIN-LOGIC  SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "HELLO, WORLD." LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```


COBOL

貢献:

- マクロ機能. 高水準言語としては最初
- 階層的データ構造 (レコード)
- 選択文のネスト
- 識別名長が長い (上限 30 文字), ハイフン有り
- Data Division を設ける

補足:

- DoD が要求仕様を出した最初の言語; DoD なくして成功しなかった
- ビジネス分野で用いられる言語としては、現在でも非常に広く用いられている

COBOLの現状

- Micro Focus社によると
- With an estimated 220 billion lines of COBOL code in active use in business and finance applications today (2011)
- COBOL systems are responsible for transporting up to 72,000 shipping containers, caring for 60 million patients, processing 80% of point-of-sales transactions and connecting 500 million mobile phone users.
- It has been estimated that the average American relies on COBOL at least 13 times during the course of a routine day as they place phone calls, commute to and from work, and use credit cards.
- Around 5 billion lines of new COBOL code are added to live systems every year.
- There are over 200 times more transactions processed by COBOL applications than Google searches each day

BASIC (1964)

- Beginner's All purpose Symbolic Instruction Code
- 設計者 Kemeny & Kurtz at Dartmouth. コンピュータ: GE 225.
  目的:
  - 科学分野でない学生でも容易に学習・利用でき、Fortran とAlgol につながる
  - “pleasant and friendly” たるべし
  - 宿題回答用にターンアラウンドが早い
  - 自由かつ個人的に利用可能
  - ユーザの時間の方がコンピュータの時間より重要

パンチカード式の大型計算機(高価！)が普通であった

- 実装が適したものに最初の PC がある, e.g., Gates と Allen の 4K Basic インタープリタ. MITS Altair personal computer (1975前後) に実装
- 現在のポピュラーな方言に: Visual BASIC


```
10 INPUT "What is your name: ", U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want: ", N
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? ", A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

ASR-33 teletype (Wikipedia より)

LISP (1959)

- LISt Processing language (設計: MIT で McCarthyによる)
- AI研究が必要としていた言語は:
  - リスト中のデータを処理できる (配列よりはこちら)
  - 記号計算が扱える (数値計算よりこちら)
- 一つの汎用的な, 再帰的なデータ型: s-式
  - 一つの s-式は、アトムか、ゼロ個以上の s-式のリスト
- 構文はラムダ計算に基づいている
- 関数型プログラミング言語の先駆
  - 変数や代入という概念が不要
  - 制御ではなく再帰と条件式
- 現況
  - AI分野では未だ優勢
  - COMMON LISP と Scheme は現代の方言
  - ML, Miranda, Haskell は近い親戚


```
(defun fact (n)
   (cond ((= n 0) 1)
         ((< n 0) nil)
         (t       (* n (fact (1- n))))))


(defun insert (x y)
       (cond ((or (null y) (< x (car y))) (cons x y))
             (t (cons (car y) (insert x (cdr y))))))

(log (sin 1))

(length '(coffee milk (chocolate cake)))
(length '((A (B) (C (D)))) (E () F))

(car '(a b c d))
(cdr '(a b c d))
(cons (car '(a b c d)) (cdr '(a b c d)) )

(defun cadr (x) (car (cdr x)))
(cadr '(a b c d))
(defun caddr (x) (car (cdr (cdr x))))
(caddr '(a b c d))
```


Algol

開発時の状況:

1. FORTRAN が (ほぼ) できた. IBM 70x 用.
2. 数多くの言語が開発途上. どれも個別コンピュータ依存
3. 可搬型言語なし; どれも機械依存
4. アルゴリズムを記述し伝える共通言語なし

ACM と GAMM が設計のための4日間ミーティング
- 言語の目標:
  1. 数学的な式表現に近い
  2. アルゴリズムを記述するのに適している
  3. 機械語に翻訳可能

ACM: Association for Computing Machinery
GAMM: Gesellschaft für Angewandte Mathematik und Mechanik

Algol 58 特徴的機能

- 型概念が形式的に示された
- 識別名の長さは無制限
- 配列の次元は無制限
- 引数は mode (in と out) によって区分される
- 添え字は角括弧の中に
- 複合文 (begin ... end)
- セミコロンを用いて文を区切る
- 代入演算子は := （コロンとイコール）
- if には else-if 句あり

補足:

- 実装されることは意図しなかった, しかし多くの派生言語は実装された (MAD, JOVIAL)
- 当初 IBM は非常に熱心であったが, 1959年中ごろには支援は完全に打ちきられた

Algol 60

ALGOL 58 はパリで行われた 6日間ミーティングで改正され次のような機能が付け加えられた:

- ブロック構造 (識別名の有効範囲が局所的)
- 引数の受け渡し機構として2種類
- 副プログラムの再帰
- スタック - 動的配列
- もっとも、まだ I/O はなく、文字列も扱われなかった

成功:

- 20年以上、アルゴリズムを発表するときの標準的な表現方法となった
- その後のすべての命令型言語の基礎となった
- 最初の機械独立な言語
- 構文を形式的に定義した最初の言語 (BNFを用いた)


```
procedure Absmax(a) Size:(n, m) Result:(y) Subscripts:(i, k);
    value n, m; array a; integer n, m, i, k; real y;
comment The absolute greatest element of the matrix a, of size n by m
is transferred to y, and the subscripts of this element to i and k;
begin integer p, q;
    y := 0; i := k := 1;
    for p:=1 step 1 until n do
    for q:=1 step 1 until m do
        if abs(a[p, q]) > y then
            begin y := abs(a[p, q]);
            i := p; k := q
            end
end Absmax
```

```
FLOATING POINT ALGOL TEST'
BEGIN REAL A,B,C,D'

READ D'

FOR A:= 0.0 STEP D UNTIL 6.3 DO
BEGIN
  PRINT PUNCH(3),££L??'
  B := SIN(A)'
  C := COS(A)'
  PRINT PUNCH(3),SAMELINE,ALIGNED(1,6),A,B,C'
END'
END'
```


Algol 60 (1960)

失敗: 広く使われることはなかった, 特に米国では. その理由は

1. I/O の規定や文字集合の規定がなかったためプログラムが可搬ではない
2. あまりに柔軟—実装が難しい （当時としては）
3. FORTRANの基盤が堅固
4. 形式的な構文記述
5. IBMの支援がない

APL

```
life←{↑1 ⍵∨.∧3 4=+/,¯1 0 1∘.⊖¯1 0 1∘.⌽⊂⍵}
```

- A Programming Language
- 設計者 K.Iverson. Harvard 大 1950年代後期
- 数学的計算をプログラムする言語
  - 特に行列を用いる人のため
- 関数型スタイル、多数用意された配列全体への演算
- 欠点は、特殊なキーボードが必要なこと

参考: 日本APL協会 http://japla.sakura.ne.jp/

1960年代: プログラミング言語の爆発

- 数百のプログラミング言語が開発された
- PL/I の設計は 1963-4
  - 汎用を目指す
  - 各言語の特徴機能の統合: FORTRAN, COBOL, Algol 60 そして…
  - 翻訳機は遅く、大きく、信頼性に欠けた
  - ある人は、時代の先取りをしすぎたという.....
- Algol 68
- SNOBOL
- Simula
- BASIC

PL/I

- “計算” に関する状況. 1964年 (但しIBMの視点)
  科学技術計算
  - IBM 1620 と 7090 コンピュータ
  - FORTRAN
  - SHARE ユーザグループ

  ビジネス計算
  - IBM 1401 と 7080 コンピュータ
  - COBOL
  - GUIDE ユーザグループ
- IBM のゴール: 一つのコンピュータ (IBM 360) と一つの言語 (PL/I) を開発し、それによって科学技術計算もビジネス計算もカバーしよう.
- 次第に、その時代の実用的なプログラミング言語のすべてのアイデアを組み込もうとするようになる.


```
FINDSTRINGS: PROCEDURE OPTIONS(MAIN);
  /* READ A STRING, THEN PRINT EVERY */
  /* SUBSEQUENT LINE WITH A MATCH */

  DECLARE PAT VARYING CHARACTER(100),
          LINEBUF VARYING CHARACTER(100),
          (LINENO, NDFILE, IX) FIXED BINARY;

  NDFILE = 0;
  ON ENDFILE(SYSIN) NDFILE=1;
  GET EDIT(PAT) (A);
  LINENO = 1;
  DO WHILE (NDFILE=0);
    GET EDIT(LINEBUF) (A);
    IF LENGTH(LINEBUF) > 0 THEN DO;
      IX = INDEX(LINEBUF, PAT);
      IF IX > 0 THEN DO;
        PUT SKIP EDIT (LINENO,LINEBUF)(F(2),A);
      END;
    END;
    LINENO = LINENO + 1;
  END;
  END FINDSTRINGS;
```


PL/I

PL/I の貢献:

1. 最初の unit-level の並列性
2. 最初の例外処理
3. 選択可能な再帰
4. 最初のポインタデータ型
5. 最初の配列断面

補足:

- 設計のつめが甘い新規特徴機能が多かった
- あまりに大きくあまりに複雑
- 実際に科学技術計算にもビジネス計算にも用いられた(用いられている)
- サブセット (e.g. PL/C) が開発された. より扱いやすい

Simula (1962-67)

- 設計・実装 Ole-Johan Dahl と Kristen Nygaard. Norwegian Computing Centre (NCC). Oslo. 1962 ～ 1967.
- もともとの設計・実装は、離散事象のシミュレーション用言語
- 基盤: ALGOL 60

主な貢献:

- コルーチン – 一種の副プログラム
- クラス (データ + メソッド) と オブジェクト
- 継承
- 動的な結合

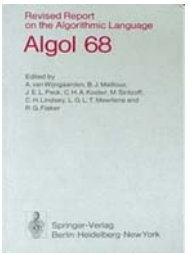
=> オブジェクト指向プログラミングに発展する基本概念を導入した.


```
Begin
   Class Glyph;
      Virtual: Procedure print Is Procedure print;;
   Begin
   End;

   Glyph Class Char (c);
      Character c;
   Begin
      Procedure print;
         OutChar(c);
   End;

   Glyph Class Line (elements);
      Ref (Glyph) Array elements;
   Begin
      Procedure print;
      Begin
         Integer i;
         For i:= 1 Step 1 Until UpperBound (elements, 1) Do
            elements (i).print;
         OutImage;
      End;
   End;

   Ref (Glyph) rg;
   Ref (Glyph) Array rgs (1 : 4);

   ! Main program;
   rgs (1):- New Char ('A');
   rgs (2):- New Char ('b');
   rgs (3):- New Char ('b');
   rgs (4):- New Char ('a');
   rg:- New Line (rgs);
   rg.print;
End;
```


Algol 68

ALGOL 60 の継続的な開発の成果. しかし言語としてはスーパーセットではない

- 設計は直交性(orthogonality)という概念に基づく
- 貢献:
  - ユーザ定義のデータ構造
  - 参照型
  - 動的配列 (flex arrays)
- 補足:
  - ALGOL60 よりも使用されなかった
  - しかし、後続の言語には多大な影響を与えた. 特に Pascal, C, と Ada



1970年代:
単純さ, 抽象化, 学習

- Algol-W - Nicklaus Wirth と C.A.R.Hoare
  - 1960年代への反発
  - 単純さ
- Pascal
  - 小, 単純, 効率的な構造
  - プログラムの教育のため
- C - 1972 - Dennis Ritchie
  - 単純さを目指す. 型システムの制約の軽減による
  - 基盤システムへのアクセスを可能とする
  - O/S へのインタフェイス. O/S は UNIX

Pascal (1971)

- 設計者 Wirth, ALGOL 68 委員会の元メンバー (委員会の方向が好きではなかった)
- 構造化プログラミングの教育用に設計
- 小さく, 単純
- 小規模な改善を導入. 例えば、case文
- 一時プログラミング教育に広く用いられる ~ 1980-1995.


```
program mine(output);

var i : integer;

procedure print(var j: integer);

  function next(k: integer): integer;
  begin
    next := k + 1
  end;

begin
  writeln('The total is: ', j);
  j := next(j)
end;

begin
  i := 1;
  while i <= 10 do print(i)
end.
```


C (1972-)

- 設計者 Dennis Ritchie と仲間. ベル研. システムプログラミング向けに設計.
- 主に B から発達, しかし ALGOL 68 からも
- 演算子を増強, しかし型チェックは非力
- 当初UNIXを通じて広がる. 高品質、コンパイラーがフリー, 特にgcc.

調べておこう: GNU and Richard Matthew Stallman

ALGOLの他の子孫

- **Modula-2** (1970年代中頃. Niklaus Wirth. ETH)
  - Pascal ＋ modules ＋ 低水準機能
    システムズプログラミング向けに設計
- **Modula-3** (1980年代後期. Digital社とOlivetti社)
  - Modula-2 ＋ クラス, 例外処理, ごみ集め, とっ並列性
- **Oberon** (1980年代後期. Niklaus Wirth. ETH)
  - OOP 支援を Modula-2 に追加
  - Modula-2 機能の多くを削除 (e.g., for文, 枚挙型, with 文, 非整数の配列添え字)

1980年代:
統合と新しいパラダイム

- Ada
  - US DoD 国防省
  - Jean Ichbiah 率いる欧州人チーム
- 関数型プログラミング
  - Scheme, ML, Haskell
- 論理プログラミング
  - Prolog
- オブジェクト指向プログラミング
  - Smalltalk, C++, Eiffel

Ada

- 1973-74 に行われた調査で, DoD はソフトウェアに毎年30億ドルに使用していることが明らかにされた. その半分以上は埋め込みシステム向けである.
- 高水準言語ワーキンググループ（The Higher Order Language Working Group）が作られ、1975-76にかけ、言語仕様の最初の版が作られ、改善され現存する言語の評価が行われた.
- 1997には, どの言語も不十分であると帰結された, ただし Pascal, ALGOL 68 と PL/I は良い出発点であろうとした.
- DoD-1 言語が開発された. 一連の競争的契約が用いられた.

```
with Ada.Text_IO;

procedure Hello is
begin
  Ada.Text_IO.Put_Line("Hello, world!");
end Hello;
```


Ada

- 1979年5月に名称変更: Ada 誕生.
- 参照マニュアル, Mil. Std. 1815 は1980年12月10日に承認された. (Ada Bryon の誕生日は1815年12月10日)
- DoD内での使用が義務付けられた. 1980年代後半から1990年代初期.
- Ada95, ISO と ANSI の共通標準, は1995年2月に承認された. 多くの新機能が含まれた.
- The Ada Joint Program Office (AJPO) は1998年10月に閉鎖された (ISO/IEC 14882:1998 (C++) が発表されたのとおなじ日)

Ada

貢献:

1. パッケージ – データ抽象の支援
2. 例外処理 – 精巧なものに
3. 総称的プログラム単位
4. 並列性 – タスクモデル

補足:

- 競争的設計
- ソフトウェア工学と言語設計に関し, 当時知られていたものすべてを組み込む
- 最初のコンパイラは困難を極めた; 最初の本当に使用可能なコンパイラは, 言語設計終了後5年後になった
- プログラミング技術を強制することは非常に難しい

論理プログラミング: Prolog

- 開発者 Colmerauer と Roussel. the University of Aix Marseille. Kowalski (the University of Edinburgh)も多少援助
- 形式論理に基づく
- 非手続き的
- ある解釈: 知的なデータベース. 与えられた質問に対する回答を、推論を用いて構成する


```
sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).



?- sibling(sally, erica).
Yes
```


関数型プログラミング

- **Common Lisp:** LISP 方言の統合でありLISPの実使用をLISP Machineと同様に加速した.
- **Scheme:** 単純な, 純LISP的言語. プログラミング教育に持ちられた.
- **Logo:** 子供たちにプログラムの仕方を教えた.
- **ML:** (MetaLanguage) 強い型付けの関数言語. 最初の開発は1970年代. Robin Milner.
- **Haskell:** 多態型（polymorphicly typed）, lazy, かつ純な関数型言語.

MLプログラム例

・リストの全ての要素に関数を適用 (apply) する

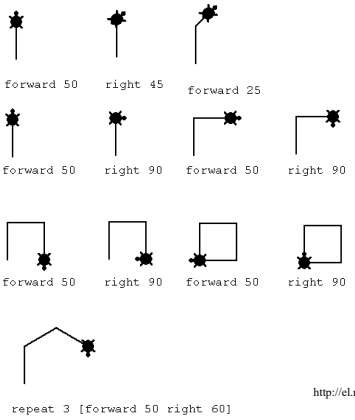
```
fun  map (f, nil) = nil
  |  map (f, x::xs) = f(x) :: map (f,xs);
```

map (fn x => x+1, [1,2,3]); ⇒ [2,3,4]

Lisp と比較

```
(define map
  (lambda (f  xs)
    (cond  ( (eq xs nil)  nil  )
           ( T  (cons (f  (car xs))  (map f  (cdr xs)))  )
    )))
```

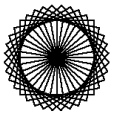

Logo プログラム例



http://www.wizforest.com/OldGood/logo/

http://el.media.mit.edu/logo-foundation/logo/turtle.html

Logo プログラム例



```
repeat 36 [right 10
square]
```

We can give this a name also.

```
to flower
repeat 36 [right
10 square]
end
```

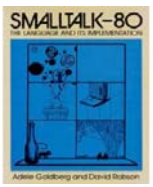
```
to square :size
repeat 4 [forward :size right 90]
end
```


```
set x 10
while x < 100 [
  do star x
  penup
  left 54
  forward 10
  right 54
  set x x + 20
  color red
]
home
do circle x
```

```
to star y
  pendown
  repeat 5 [
    forward y
    right 144
    forward y
    left 72
  ]
end

to circle x
  penup
  forward x
  color blue
  pendown
  right 90
  repeat 45 [
    forward 16
    right 8
  ]
end
```

http://www.cis.upenn.edu/~matuszek/cit594-2004/Assignments/5e-sample-program.html

Smalltalk (1972-80)



- 開発者 Alan Kay と仲間(特に Adele Goldberg). Xerox PARC. Simula 67 に啓発される
- 1971(2?)年に最初のメモ. メッセージ伝達に基づく言語は一ページのコードで書けるか？という賭けに勝つために書かれた.
- 1980年に, Smalltalk 80, どこもかしこもオブジェクト指向なプログラミング環境は、Smalltalk言語の最初の商用版リリースとなった
- 今では当たり前になってしまったGUI (graphical user interface) の先駆者である
- 現在でも使われている

C++ (1985)

- 開発者 Stroustrup. Bell Labs.
- C と SIMULA 67 の発展形
- オブジェクト指向プログラミング用の多くの機能が SIMULA 67 から持込まれ, C に付け加えられた
- 例外処理機能
- 大きく複雑な言語, 理由の一つは手続き的言語とオブジェクト指向プログラミングの両方をサポートしているから
- OOP オブジェクト指向プログラミングの広がりとともに、急速に広まる
- ANSI 標準として承認. 1997年11月

Eiffel

•Eiffel – OOPをサポートする言語

- 設計者 Bertrand Meyer. 1992年
- 直接の先祖言語はない
- C++より小さく単純だが, その強力な機能のほとんどを有する