FORTRAN

- 科学技術計算用
- 数値演算精度を重視したシステム
- 事務処理計算に(文法上も)特化したCOBOLの対極
- 1955以来長く使われ、多数の数値演算ライブラリが用意されている

```
      K=0
      DO 10 I=0,N,1
        K=K+I
10 CONTINUE
```

COBOL

- 事務処理向け
- 自然言語に近く、冗長な記述
  - ビジネスロジックをそのまま記述しバグを減らす
- レコード定義が自然に可能
- 1-Nまでの加算も冗長だが普通に書ける

```
    MOVE ZERO TO K.
    PERFORM LOOP-ADD UNTIL
                  I EQUAL TO N.
....
LOOP-ADD.
    ADD I TO K.
    ADD 1 TO I.
```

```
01 INPUT-RECORD.
   10 ID          PICTURE X(6).
     20 YEAR      PICTURE 9(2).
     20 SEQ       PICTURE 9(4).
   10 LENGTH      PICTURE 9(3)V99.
   10 NAME        PICTURE X(20).

01 OUTPUT-RECORD.
   10 ID          PICTURE X(6).
     20 YEAR      PICTURE 9(2).
     20 SEQ       PICTURE 9(4).
   10 PRICE       PICTURE 9(3)V99.
   10 FILLER      PICTURE X(20).
....
   MOVE ID TO WORKID.
   ADD 1 TO SEQ OF WORKID.
   WRITE OUTPUT-RECORD.
```

BASIC

- Beginner's All porpose Symbolic Instruction Code
  初心者向け汎用記号命令列(?)
- 1964年から
- 学習が容易
- 簡易な記述
- 長いプログラムをわかりやすく書く仕組みがない
- 8bit マイクロコンピュータと簡易なインタプリタとともに 1980 頃大きく普及

```
K=0
FOR I=0 TO N
    K=K+I
NEXT I
```

Pascal

- 1968 年に開発
- 構造化プログラミングの思想に沿った最初の言語
- プログラミング教育用
- 商用でも利用される
  - Macintosh OS
  - Borland の Delphi など

```
program sample;
 var k, i : integer;
 begin
  k := 0;
  for i := 1 to 10 do
   begin
     k := k + i;
   end;
  writeln(k);
 end.
```

C

- システム記述向け
- 1972年に開発（これでも後発）
- 下例が一般的手法
- 右例が再帰呼び出しを利用した記述
  - 0の累計は0と定義
  - 0以上のNの累計はN-1の累計にNを足したものと定義

```
main() {
  int i,k,n;
  n=10;
  k=0;
  for(i=0;i<=n;i++) {
    k+=i;
  };
  printf("%d¥n",k);
};
```

```
int sub(i) {
  if(i==0) {
    return(0);
  } else {
    return(sub(i-1)+i);
  };
}

main() {
  int k,n;
  n=10;
  k=sub(n);
  printf("%d¥n",k);
};
```

pp.133-

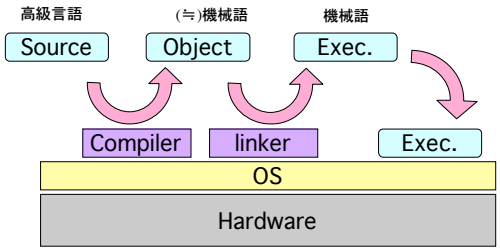
コンパイラとインタプリタ

- プログラミング言語から機械語へ
  - CPUは機械語しか実行できない
  - 機械語によって等価な振舞いをさせなければ
  - どうやって？
- 二つの方法
- コンパイラ
  - 機械語に変換する
- インタプリタ
  - 変換せず真似て動作させる

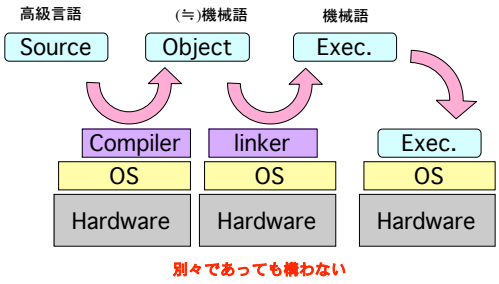
コンパイラ

- Compiler = 編集、編纂
- 機械語に変換して実行
  - 変換前：原始プログラム (Source program)
  - 変換後：目的プログラム (Object program)
  - 多くの OS では目的プログラムをリンク (link) と呼ばれる処理を通してライブラリと結合し、実行可能プログラム (executable program) とし、これを実行する

コンパイラ



コンパイラ



別々であっても構わない

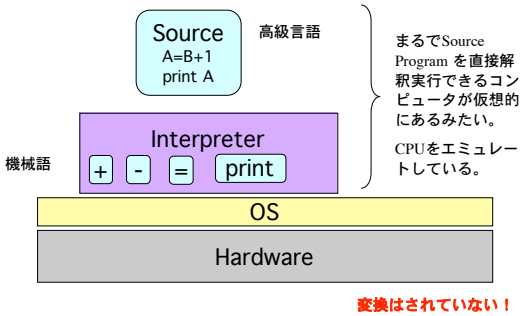
コンパイラ

- 最適化の可能性
  - ループの中の無駄な演算をループの外に
  - 最近の CPU では並列度を上げるためにも使われる（Itanium などでは CPU 中の並列動作可能な演算命令を同時に投入することが可能）
- 変換一回、実行多数回、では高効率
- 秘密保持
  - ソースが得られない場合が多い（逆変換不可）

インタプリタ

- プログラムの機能ごとに対応し、部分的に実行可能な機械語コードを予め用意しておく
- 実行したいプログラムを部分的に読み出し、対応する機械語コードを選択して実行
- Interpreter = 通訳、ではあるが変換はしていない
  - 逐語的に処理しているため、と理解するべき
- まるでシミュレーション
  (simulate : 装う / emulation : 真似る)
  - その言語を直接実行できる CPU をソフトウェアで作ったようなもの

インタプリタ



変換はされていない！

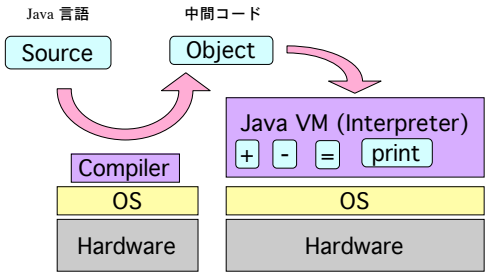
インタプリタ

- 部分的実行が可能
  - 一行ずつテストしながら開発するような作業が可能
- 実行速度が遅い
  - 真似るために必要な機械語は等価変換した機械語より処理量が多い
- 機械語が異なる環境でも動作する
  - Java / iアプリ

Java

- 教科書 pp.131
  - Sun Microsystems が 1995 年に開発
  - C に似た文法
- まず中間プログラムにコンパイル
  - 実行速度を向上させるため
  - バイトコード (Byte Code) と呼ばれる
- インタプリタで実行
  - CPU やハードウェアの差を吸収して実行
  - Java VM (Virtual Machine 仮想計算機) の存在
  - iAPPLや Web で使われる

Java



コンパイラとインタプリタ

- コンパイラ：一括変換して実行
- インタプリタ：逐次真似して実行
- 様々な方式がある
  - Java のような組み合わせ
- Java 自身、VM 高速化のためにいろいろやっている
  - Just In Time コンパイラ：バイトコード実行前に一括して機械語に変換、実行
  - Hot Spot ：バイトコードの一部分（ループの中など）をコンパイルしながら実行
- 工夫の産物である
  - 無意味な分類にとらわれないように