コンピュータ概論B
ー ソフトウェアを中心に ー

#11 さまざまなプログラミング言語

京都産業大学
安田豊

プログラミング言語

- 低水準プログラミング言語 (教科書 pp.112)
  - 例えばアセンブリ言語
  - CPU依存 = CPUの構造を理解して指示する
  - 実行時の効率の良さ（速度とメモリ占有量）
- 高水準(高級)プログラミング言語
  - 例えば C など
  - CPU 依存せず（大抵 OS にも依存しない）
  - ニーモニックではなく、より自然言語に近い記述
  - 等価な機械語を導出して実行する

アセンブリ言語の使い道

- 教科書 pp.116-
- 低水準言語はもう使われないか？
  - NO : 減っては来ているが無くなっていない
- 極端に高い効率が要求される場合
  - OS やドライバなどシステムソフトウェア
  - ハードウェア性能を低く抑えたい場合
- 組み込み機器での利用
  - コスト第一
  - ハード性能の低さ＝極限までの高効率
  - 機能の単純さ＝OS不要

組み込み機器

- コスト至上の世界
  - システムの部品でしかない = 数十円レベルでの攻防
  - ハードウェア性能を極限まで抑える
  - アセンブリ言語でしか組めない
    （話が逆？つまり組める限界までハード能力を下げたい）
- 単機能かつ単純機能
  - OS も不要である場合が多々
- リアルタイム性
  - 最悪条件でも何ms以内に終了するか明確にする
  - 例：自動改札機：複雑な組み合わせの切符だった場合にいつ終わるか分からない、では？
- アセンブリ言語が最適

高水準言語の種類

- 教科書 pp.117-
- 手続き型言語と非手続き型言語
  - 処理手順を書き下すタイプ
  - 処理手順を意識しないタイプ
- 分野や対象を絞って開発・使用される
  - FORTRAN : 科学技術計算用
  - COBOL : 事務処理用
  - C : OS記述用
  - 教科書 pp.117 の図が参考になる
  - つまり言語開発には流れもある

手続き型言語

- 処理手順を書き下す
  - 1からNの和を計算する

```
K=0
FOR I=1 TO N
  K=K+I
NEXT I
```

- ループ（繰り返し）
  - 代表的な自動処理
- 変数という概念
  - 時間と共に移り変わる値の入れ物
  - メモリそのもの
    ＝ノイマン型コンピュータモデルの本質

- 変数K (ある位置のメモリ) の内容をゼロに
- 変数 I を 1 から N までひとつずつ変化させながら繰り返し
- K の内容と I の内容を足したもので K を上書き

アルゴリズム (Algorithm)

- 人間向け問題
  「1からNの和を計算する」
- 計算機向け問題 →
  - この「導出を可能にする解法」をアルゴリズムと呼ぶ
  - 手続き型プログラムとはアルゴリズムを「手順重視」で、文法に従い記述したもの
- プログラミング
  - 人間向け問題から計算機向け問題への変換
  - アルゴリズムの発見と記述

```
K=0
FOR I=1 TO N
  K=K+I
NEXT I
```

- 変数K (ある位置のメモリ) の内容をゼロに
- 変数 I を 1 から N までひとつずつ変化させながら繰り返し
- K の内容と I の内容を足したもので K を上書き

FORTRAN

- 科学技術計算用
  - 数値演算精度を重視したシステム（但し文法上にはほとんど現れず）
  - 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
- ライブラリ
  - ユーザが作成するプログラムから再利用可能なプログラム集
  - プログラミング言語やソフトウェアに蓄積が重要である一つの理由

FORTRAN

- 1-Nまでの総和を求める
- 変数K (ある位置のメモリ) の内容をゼロに
- ラベル10までを変数 I を 1 から N までひとつずつ変化させながら繰り返し
- K に K の内容と I の内容を足したものを記入

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

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
  - 後に導入
  - Visual Basicなどの本格的アプリケーション開発にも使われている
- 8bit マイクロコンピュータと簡易なインタプリタ（後述）とともに 1980 頃大きく普及
  - コンパイラ（後述）もある

BASIC

- 1-Nまでの総和を求める
- 変数K (ある位置のメモリ) の内容をゼロに
- ラベル10までを変数 I を 1 から N までひとつずつ変化させながら繰り返し
- K に K の内容と I の内容を足したものを記入
- FORTRAN の例とほぼ同じ
- 典型的手続き型記述

```
K=0
FOR I=1 TO N
   K=K+I
NEXT I
```

COBOL

- COmmon Business Oriented Language
  共通事務指向言語(?)
- 1960年以来長く使われている
- 大型汎用機生まれ
  - 現在では小型機やPCでも
- レコード処理に特化した機能が多い
- 整数多倍長演算などが容易
  - 事務向け
  - 全体の精度より最後の一桁までの正確さを重視
  - 結果的に(多倍長処理部分などは)低速

COBOL

- 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.
    MOVE ZERO TO I.
    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 FILLER      PICTURE 9(3)V99.
   10 FILLER      PICTURE X(20).
....
   MOVE ID TO WORKID.
   ADD 1 TO SEQ OF WORKID.
   WRITE OUTPUT-RECORD.
```

C

- 1972年に開発（これでも後発組みにあたる）
  - 構造化プログラミングスタイル
  - 構造化された記述、モジュール化、再帰呼出し
- 元々システム記述用言語として開発
  - メモリ構造を意識した記述が可能
  - CPU を強く意識した記述 (i++ 等) もある
  - 過去にはアセンブラで記述したような処理も可
- 現在では汎用
  - 殆どどこででも使われている
  - それほど高級でもなく、バグも入りやすいが、、

C

- 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=1;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);
};
```

非手続き型言語

- データの処理手順を書くのではなく
- データ間の関係を記述する（など）
  - 計算機が自動処理する内容はそこから導き出す
  - アルゴリズムの別の視点からの記述でもある
- 例：論理型言語 Prolog
  - 論理(二者間の関係)を記述すれば
  - 自動的に解を導出することができる
- 例：データフロー
  - 処理するべきデータ間の演算方法を書けば
  - データが揃い次第先へ進み解が出る

Prologによるプログラム例

```
grand(X,Y) :- parent(X,Z), parent(Z,Y).
parent(amy,bob).
parent(amy,jane).
parent(bob,carol).
parent(bob,will).
```

- 手続きを書くのではなくデータの関係を記述
- そこからシステムが解を導出する
- 実際は文頭から手続き的に解く、、、
- 例：
  - amyの親はbobとjane
  - bob の親はcarolとwill
  - 事実を入力すると yes と答え
  - 変数を与えると、あり得る解を返す
  - amyの祖父母は？willが祖父母なのは？

```
% sbprolog
SB-Prolog Version 3.1
| ?- consult('a.pro').
yes
| ?- parent(amy,bob).
yes
| ?- parent(amy,X).
X = bob
X = jane
no
| ?- grand(amy,X).
X = carol
X = will
no
| ?- grand(X,will).
X = amy
no
```

Prolog による1-Nまでの和

```
kei(0,0).
kei(N,K) :- N > 0, N1 is N - 1,
    kei(N1,Kn), K is Kn + N.
```

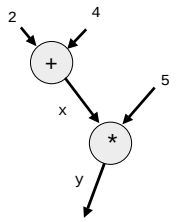
```
| ?-kei(0,0).
yes
| ?-kei(0,100).
no
| ?-kei(0,N).
N = 0
yes
| ?-kei(1,N).
N = 1
yes
| ?-kei(4,N).
N = 10
yes
| ?-kei(1000,N).
N = 500500
yes
```

- 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
- 再帰的定義
- Nを指定し、kei(N,K)を評価することでKを導出
- 再帰的手順

データフロー

- 関数型言語の一つ
  - 処理をデータ間の関係として定義
  - データが揃った関数から実行
  - ループは再帰的実行で処理
- 実行順序は指示されない
  - 値が上書きされるようなメモリの使い方ができない
  - ノイマン型である必要がなくなるポイント

{x=2*4, y=x+5}
という計算をフローグラフで表現



データフロー

- 並列処理に有利
  - 順序による制約はノイマン型の致命的弱点
  - 処理の並列度が上がらない
- ハードウェアとの親和性
  - そのままプログラムを回路化できる可能性
  - 演算素子を並列に用意できる
- 並列動作可能な構成
  - ノイマン型では複数用意しても同時に使えない
  - CPU （ノイマン型処理装置）は一点注視タイプ

まとめ

- 低水準言語と高水準言語
  - CPU 依存と非依存
  - 自然言語との距離
- 手続き型言語と非手続き型言語
  - 手続き型：ノイマン型そのもの
  - 非手続き型＝非ノイマン型システムの可能性
    - 非手続き型言語のノイマン型システムによる処理は効率が上がらない場合が多い
  - いずれにしてもアルゴリズムの記述である
- 目的に合わせて多様な言語が存在する
  - データフローチップも画像処理などで実用化

組み込み機器再び

- Javaと組み込み機器
  - J-TRON
  - ハードウェア性能の向上
  - かなりのシステム資源を割けるように
  - 組み込み機器にも高級言語システムが載せられるようになった
  - 開発工数の短縮のために
  - 組み込み機器と Linux

コンパイラとインタプリタ

- FORTRANとC
  - 再帰呼び出し、pure code 、再入可能性
- CとBASIC
  - コンパイラとインタプリタ
- Java、SmallTalk
  - バイトコード・コンパイルとインタプリタ
  - Virtual Machine の考え方