プログラミング入門

プログラミング演習 1#15
平成 26 年 7 月 25 日

1　計算機 (computer) とプログラム (program)

1.1　プログラムとは?

プログラム：計算機がやるべき仕事を明確に記述したデータ.

プログラムの記述に使われる言語をプログラミング言語という.

機械に分かりやすい言語を低級言語 (low-level language), 人間に分かりやすい言語を高級言語 (high-level language) という. (「分かりやすい」というのは程度の問題なので,「高級」／「低級」というのも相対的な表現)

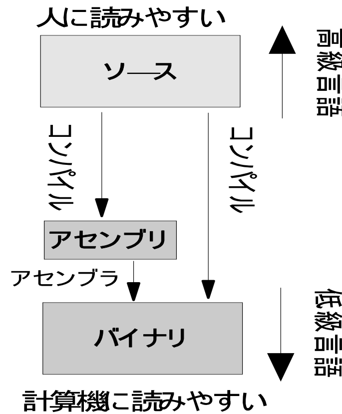

図 1: 高級言語と低級言語

1.2　高級言語によるプログラム: 仕様 (specification) としての側面

ここでちょっと復習：

定義 1 (階乗) n の階乗 *(factorial)* $n!$ は以下のように定義される.

$$n! = 1 \cdot ... \cdot n$$

定義 2 (階乗の厳密な定義) てかより厳密には, 次の定義が使われる

$$\begin{aligned} 0! &= 1 \\ n! &= n(n-1)!, \textit{ if } n \geq 1 \end{aligned}$$

例 1 階乗を計算する, *Haskell*（高級言語の一種）によるプログラム

```
factorial n = product [1..n]
      -- （「n の階乗の定義は, 1 から n までの積」という意味）
```

例 2 階乗を計算する，*Haskell* によるプログラムの別バージョン

```
factorial 0 = 1
factorial n = n * factorial (n-1)
```

ポイント：

- 一定の文法を持った「言語」である．
- （文法を知っている）人間に理解可能で，それでいて曖昧さがない
- 人間の持っている概念，計算機にやらせようとしている仕事（＝仕様）が，プログラムとして記述することで，明確になる

1.3 低級言語によるプログラム: 実装方法 (implementation) としての側面

機械語：CPU に直接理解できる言語．数の羅列．
アセンブリ言語：機械語を単純に文字列に置き換えたもの．
アセンブリ言語の各行は，1〜数バイトの機械語と 1 対 1 対応する．

例 3 階乗を計算する，アセンブリ言語によるプログラム[1]

```
        MOV (N), BX    ; メモリ上の N 番地に書いてある値を，CPU 内の BX というレジスタに読み込む
        MOV  1,  AX    ; CPU 内の AX というレジスタに 1 を代入
LABEL:  CMP  BX, 0     ; BX と 0 を比較して…
        JMP  Z,  EXIT  ; BX が 0 なら，EXIT に飛ぶ
        MUL  BX, AX    ; AX に BX を掛けて，AX に上書き
        DEC  BX        ; BX から 1 を引いて，BX に上書き
        JMP  LABEL     ; LABEL に飛ぶ.
EXIT:   MOV  AX, (M)   ; AX の内容をメモリ上の M 番地に書き込む
        RET            ; 終了
```

用語説明：

レジスタ CPU 内にある一時的な記憶場所．メモリよりも高速に読み書きできる．

番地 メモリ上の場所を表す．たとえば，メモリを $1GiB(=2^{30}B)$ 積んでいる場合，メモリには 0 番地から $2^{30}-1$ 番地までの計 2^{30} 個の番地がある．で，それぞれの番地に 0 から 255 までの値が格納されている．つまり言い換えると，メモリの状態は $\{0..2^{30}-1\} \to \{0..255\}$ な関数と等価.

ポイント：

- CPU が何をどういう順で実行するか（=実装方法）を記述．いわば人様に見せられない (?) 内部事情.

[1]とりあえずこのプログラムが何をやっているかは分からなくてよいです.「人間にとってはすごく分かりにくい」ということが分かれば OK．あと，このプログラムはかなりテキトーに書いてます．てか，書いたはいいけど本当に動作するかちゃんと確認してないですのであしからず.

- 基本的に頭から順番に実行．現在実行中の行を指し示すカウンタ（プログラムカウンタ，PC）がある．分岐命令（プログラム中 JMP と書いた所）以外では PC は順に増えていくが，分岐命令では PC に直接値を書き込む．
- CPU が直接理解できる（機械語の場合），
あるいは，CPU が理解しやすい（＝機械語に似ていて翻訳しやすい，細かい記述が可能・必要）
- 逆に人間には分かりにくい．理解するには，CPU やメモリの (抽象的な意味での) 構造を意識する必要がある．
- CPU が異なると言語も異なる．このため，たとえば ARM 用に書かれた機械語のプログラムを Intel Core で走らせるのは困難である．

1.4 コンパイラ (compiler)，インタープリタ (interpreter)

高級言語のプログラムは，人間には（比較的）分かりやすいものの，CPU が直接理解できるものではない．実行するためには，機械語に翻訳される必要がある．この翻訳の作業は，人間が手作業でやることも不可能ではないが，翻訳のための専用のプログラムを使った方が楽なので普通はそうする．

高級言語のプログラムを翻訳して機械語のプログラム (あるいはより低級な言語のプログラム) を生成するプログラムをコンパイラと呼び，この翻訳の作業をコンパイルという．高級 (とは限らない) 言語のプログラムを同時通訳的に解釈して実行するプログラムをインタープリタという.（これらの中間的存在として，プログラムが実行のため呼ばれるごとにコンパイルして実行する JIT コンパイラというのもある）

コンパイラとインタープリタの用途上の違いは以下のとおり:

- 何度も実行する場合，長時間実行する場合は，インタープリタで実行するよりも，コンパイルしてから実行した方が高速である．
- プログラムのバグ（=誤り）を見つけるのには，プログラムの修正と実行を繰り返すので，インタープリタが便利である．
- 機械語のプログラムの場合，特定の機種に依存してしまい，また悪意のあるコードがあるとそのまま実行してしまうことになる．このため，クライアント（閲覧者）側で実行されるウェブページ上のプログラムは,（速度があまり重要視されないこともあって）インタープリタ上で実行されることが多い．

同じプログラムに対して，コンパイラによって翻訳される前のプログラムをソースプログラム (source program),あるいは単にソースと呼ぶ．これに対し，翻訳された後のプログラムをバイナリ (binary) と呼ぶ．

1.5 オペレーティングシステム (Operating System, OS) と機械語プログラム

1.5.1 オペレーティングシステムとは?

計算機が動作するのに必要不可欠なプログラムで構成されるシステム．要は UNIX やら Windows やらのこと．

基本的に，計算機の電源が入っているときは必ず何らかの機械語プログラムが走っている.（何もしていないように見えるときは，分岐命令で同じ所をぐるぐる回っている）つまり，OS も一種のプログラムである．

1.5.2　機械語プログラムの起動

大まかに言って[2]，OS がおのおののプログラムを起動するときは，以下のような手順となる．

1. 起動するべきプログラムをメモリに読み込む．
2. プログラム実行後に戻ってくるべき場所を所定の場所（スタック）に書き込む．
3. プログラムカウンタの内容を，読み込んだプログラムの所定の場所（大抵は先頭）に書き換える．

終了するときは，戻るべき場所をスタックから読み込み，プログラムカウンタに代入することになる．

2　いろいろな計算機言語

計算機言語は，大まかに言って宣言型と命令型に分けられる．

2.1　宣言型言語 (declarative languages)

数学的あるいは論理的事実を並べていくことでプログラムを組み立てていく言語を宣言型言語という．宣言型言語はさらに，関数型と論理型に分けられる

2.1.1　関数型言語 (functional languages)

関数型言語は，数学的事実を積み重ねていくことでプログラムを構成する言語である．Haskell, O'Caml, ML, Scheme, Lisp, Agda などがこの範疇に属す．

2.1.2　論理型言語 (logic languages)

論理型言語は，論理的依存関係を論理式として積み重ねていくことでプログラムを構成する言語である．Prolog, Mercury などがこの範疇に属す．

2.2　命令型言語 (imperative languages)

これらに対し，機械語のように，何をどういう順番で実行していくかを記述する言語を命令型言語という．C, C++, Java, Ada, Pascal, Delphi, Fortran, Basic, Smalltalk, Perl, Ruby, などがこの範疇に属す．

プログラミング演習 2 では，基本的に C 言語を扱う[3]．ちなみに，階乗を計算する C 言語プログラムの例を以下に示す．

```
int factorial(int n) {
  int i, r = 1;         // i, r を整数とし，r を 1 とおく
  for (i=1; i<=n; i++)  // i を 1 から n まで動かして，
    r *= i;             // それぞれの i を順に r に掛け合わせていく
  return r;             // その結果の r が求める値として返される．
}
```

[2]厳密には，割り込みやらコールバックやらいろいろある．
[3]ほかの言語に興味のある方は個人的に聞いてください．

同じ変数がいろんな値を持つので，数学的な解釈が困難なのはアセンブリ言語や機械語と同様である．その一方で，アセンブリと比べると明らかに簡潔になっていることが分かる．

2.2.1 C 言語を学ぶ理由

- みんなが使っているから（ Windows を使う理由）
  - 複数の人間でプログラムを共同開発する場合，C 言語に統一すると便利（別に Java や C++で開発してもよいが，これらの言語は C 言語を発展させたものである）
  - C で書かれているオープンソースのプログラム（ソースが公開されているプログラム）は多いため，C を知っていると，これらのプログラムを自分で自由に修正，改良可能.
  - ライブラリ（使いまわしのできるプログラムの部品）が豊富
  - ほとんどの OS に対してコンパイラが存在し，また最初から入っているため，プログラムがコンパイルできなくて困る，ということがまずない.
  - 分からないときに人に聞ける
- 高級アセンブリだから.
  - いずれ計算機の内部構造を学ぶとき，アセンブリをある程度知っているとよいが，アセンブリは難しすぎる．
  - 機械語に近い記述ができるため,（労力を惜しまなければ）工夫次第で可能な限り効率的なコードが書ける
  - 単純なデータを単純な形で扱うため，他の高級言語と併用しやすい．たとえば，基本的には他の言語で書いておいて，特に高速化したい部分だけ C で書く，といったことが比較的容易である．

3 とりあえずプログラム書いて走らせてみよー

まず，入門にありがちな「“Hello, World!” と表示して終わるプログラム」というのを書いてみます．
手順:

1. emacs を起動する．
2. 拡張子が`.c`であるような適当なファイル名（たとえば hello.c）で，以下のプログラムを半角で入力して，セーブする．

   ```
   #include <stdio.h>
   int main () {
     printf("Hello, World!\n");
     return 0;
   }
   ```

3. コンパイルする．通常の UNIX マシンなら `gcc` という C コンパイラが使えるはずなので，コンソールから

   ```
   gcc hello.c
   ```

とタイプする．と，コンパイルされた結果が a.out というファイル名のコマンドとして，カレントディレクトリ（現在のディレクトリ）に保存される．

もし gcc が何の文句も言わないのにうまく行かない場合は，

```
gcc -Wall hello.c
```

のように-Wall を付けてコンパイルしなおしてみるべし．どこがおかしいかのヒントが得られることがある．

4. ./a.out とタイプすることで実行できる．ここで“./”とはカレントディレクトリのこと．

3.1 プログラムの解説

```
#include <stdio.h>
```

標準的な入出力関係のライブラリである stdio を使用するという意味．[4]ライブラリというのは，有用な関数（プログラムの部品）をいろいろ集めてまとめたもの．LaTeX でいうところのパッケージみたいなもん．関数については，次節を参照．

このプログラムでは，stdio の関数である printf が使われているので，この記述がある．

```
int main() {
  ..
}
```

main という関数を定義している．ここでは，main はパラメータを持たず，int（integer:整数）を返す．C 言語のプログラムにおいて，基本的に main という関数が実行される．

```
printf("Hello, World!\n");
```

printf という関数に"Hello, World!\n"という文字列をパラメータとして与えている．ここで，\n というのは改行を表す．printf というのはパラメータとして与えられた文字列を表示する関数である．

```
return 0;
```

return というのは，関数の実行を終了することを表す．return 0; というのはこの関数が 0 という値を返すことを意味する．[5]

3.2 関数とは?

C 言語における関数とは，パラメータ（引数ともいう）にもとづいて，なんかやって，なんかの値を返す（返さないこともある）もの．上の例で printf もなんか返しているのだが，その値は無視されている．

[4] stdioってのは多分 standard I/O の略だが，別に標準入出力に限らず一般のファイルを扱う関数とかも含まれている．

[5] main 関数が 0 を返すということは，プログラムが正常に (エラーなく) 終了したことを意味する．