2017年度 春学期・(火II)

情　報

第12回目「プログラミング」
29. プログラミング言語の概要
30. プログラミング

東京学芸大学 自然科学系
宇宙地球科学分野 講師
西浦 慎悟

● 最初に

講義資料は、http://astro.u-gakugei.ac.jp/~nishiura

→「西浦クンの講義室」に縮小版（PDFファイル）を置く予定

● 機械語

コンピュータが理解できる「0」「1」のみで記述されるプログラミング言語。最もコンピュータ寄りの低水準言語（低級言語ともいう、機械向きの言語）と言える。ただし、人間に対しては極めて難解で間違えやすい。

→「0」「1」の信号の機械語をどのように理解するかは、CPU内部の回路に焼き込まれている。

● アセンブリ言語

機械語を、人間にもわかるような形式で記述できるようにした低水準言語。CPUの命令に対応したニーモニックやデータ、結果の格納先であるメモリ・レジスタなどで記述する。アセンブリ言語で書かれたプログラムを機械語に変換することをアセンブル（assemble）、変換を行うプログラムをアセンブラ（assembler）という。

アセンブリ言語は、依然としてコンピュータ寄りの方式で記述され、必ずしも分り易いものでは無い。また、アセンブリ言語の形式は、CPUに依存し、汎用性も決して高いものでは無い。

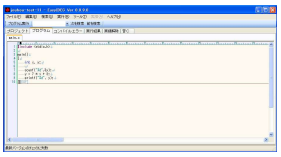
例）MOV　CL，61h　→ CLレジスタに61（16進数）を読み込む。

● アプリケーション/ソフトウェア

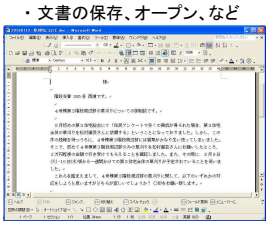
・「MS-Word」のアイコンをダブル左クリック → MS-Wordが起動

・文字入力、文字サイズ変更、文字フォント変更、などなど。

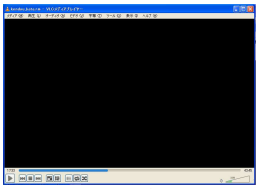
・文書の保存、オープン、など



↑学習用C言語開発環境の作業画面



↑MS-Wordの作業画面



↑VLCメディアプレイヤー

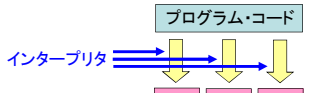
→ BIOSやOSを含めてコンピュータ・ソフトウェアは開発されたもの

● インタープリタ型プログラミング言語

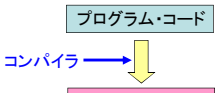
人間向きに記述されたプログラム・コードを、機械語に逐次変換し、実行させるソフトウェアの総称。実行形式の観点から、「N88BASIC」や「Perl」などはインタープリタ型のプログラミング言語に分類される。逐次変換するために、実行速度は遅いが、最近では、中間形式まで変換することで、実行速度を向上させるといった改善を施されたものもある。

● コンパイラ型プログラミング言語

人間向きに記述されたプログラム・コードを、全て機械語に変換して、実行型のプログラムを生成させるソフトウェアの総称。実行形式の観点から、「Fortran」や「C言語」などはコンパイル型のプログラミング言語に分類される。全て機械語に変換されているため、実行型プログラムの速度は速い。



↑インタープリタ型プログラム言語のイメージ

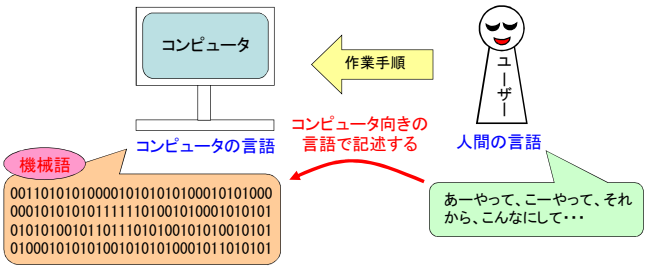


↑コンパイラ型プログラム言語のイメージ

アプリケーションなどのソフトウェアは、何らかの目的を実現するために、コンピュータに作業をさせるものである。

→ コンピュータに何らかの作業を行わせるためには、その手順をコンピュータに教える必要がある。

どのようにしてコンピュータに作業手順を教えるのか？



プログラミング言語は、記述思想により、以下のように分類されることもある。

・手続き型プログラミング（Procedural programming）言語

命令型プログラミング言語とも呼ばれ、内部状態を変化させる「手続き」（「命令」）によって記述されるプログラミング言語の総称。

1957年 FORTRAN、科学技術計算用プログラミング言語。
1960年 COBOL、事務処理用プログラミング言語。
1964年 BASIC、8ビットパソコン主流期の汎用性プログラミング言語。
1969年 Pascal、教育用プログラミング言語。
1972年 C言語、システム開発用プログラミング言語。
1987年 Perl
1995年 PHP

・関数型プログラミング（Functional programming）言語

「手続き」や「命令」では無く、「関数」を組み合わせることで記述されるプログラミング言語の総称。

1960年 LISP、初の関数型プログラミング言語。

29. プログラミング言語の概要　　第12回目

・論理型プログラミング（Logic programming）言語

「数理論理学」に基づき、命題の真偽を問うことを目指したプログラミング言語。人工知能の開発に大きな影響を与えたと言われている。

1972年 Prolog、初の論理型プログラミング言語。

・オブジェクト指向プログラミング（Object-oriented programming）言語

データとその処理を行う機能のひと纏まりを「オブジェクト」として捕え、この集合体として全体を構成させるプログラミング言語。

1962年 Simula、初めてオブジェクト指向の概念を取り込んだ言語。
1972年 SmallTalk、初めてオブジェクト間のメッセージを導入した言語。
1983年 C++
1990年 Python、スクリプト言語としては初めてのオブジェクト指向言語。
1993年 Ruby
1995年 JAVA

・スクリプト言語　比較的簡単なプログラミングを記述するための、簡易なプログラミング言語の総称。Awk、Perl、Python、Ruby、PHP、JavaScript などはこれに分類される。

第12回目

● C言語の書式

・基本構造：

```
#include <stdio.h>
必要があれば、ここにサブルーチンを置く
main()
{
    プログラム本体はここに書く
}
```

必要があれば、ここに扱う変数の形式を書くことがある。

プリプロセッサと呼ばれ、ソースのコンパイルに先立って行われる前処理について記述する。ヘッダーファイルの取り込みには「#include」、マクロの定義には「#define」を用いる（本授業では詳細は省略）。

main()関数中に置くと複雑になる部分は、main()の外に「サブルーチン」として置くことができる（本授業では省略）。

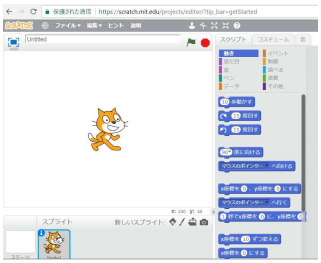
基本的には、main() 関数の中に書かれた順に、命令が実行される。

使用する変数の種類・形式は、最初に宣言しなければならない。

第12回目

● Scratch

2006年にMITメディアラボが開発・発表した初心者向きプログラミング言語学習環境。従来の教育用プログラミング言語と異なり、詳しいプログラミングの構文を記述することなく、プログラミングの結果が得られる仕様となっている。
MS-Windows、MacOS、Linux に対応しており、2013年5月のバージョンからはウェブ・アプリケーションとなった。様々なScratchの派生版も登場している。



↑（Scratch ウェブページのプログラミング画面）

GUI上で、様々なパラメータ（条件）やアクション（結果）のブロックを、ドラッグ・アンド・ドロップすることでシンプルなゲームやインタラクティブ・アニメーション、プレゼンテーションなどの作成が可能である。

子供から大学の初心者まで幅広い場所で使用されている。日本でも、小学校での情報やコンピュータ、プログラミングの本格導入に伴って、Scratchを用いたプログラミングや情報を教える塾が現れている。

Scratch の公式ウェブサイト
https://scratch.mit.edu/

30. プログラミング　　第12回目

・変数の宣言：

最初に、使用する変数の型と名称を明記し、変数のためのメモリ領域を確保する。なお、プログラムの途中における変数の宣言は禁止されている。

データ型名	変数型	バイト数	範囲
short	符号付整数型	2 bytes	−32768 ～ 32767
int	符号付整数型	2 bytes 4 bytes	−32768 ～ 32767 −2147483648 ～ 2147483647
long	符号付整数型	4 bytes	−2147483648 ～ 2147483647
float	単精度浮動小数点型	4 bytes	3.4×10^{-38} ～ 3.4×10^{38}
double	倍精度浮動小数点型	8 bytes	1.7×10^{-308} ～ 1.7×10^{308}
char	文字型	1 byte	−128 ～ 127

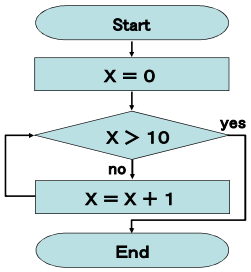
例）　int x; → 符号付整数型で変数"x"を用意。
int x, y; →符号付整数型で変数"x"と"y"を用意。
int x, y=10; → 符号付整数型で変数"x"と"y"を用意し、"y"に予め10を代入。

30. プログラミング　　第12回目

・アルゴリズム：コンピュータに行わせる作業の方法や手順を、一般化・定式化したもののことで、プログラムの基礎となる部分。

・フローチャート（flowchart、流れ図）：アルゴリズムやプログラムを、プロセス毎に表現した図のこと。

例）変数「X」に初期値「0」を入力し、それに順次「1」を加えて、10を超えたら終了。



→ プログラミング言語によって、その表記方法は異なる。

30. プログラミング　　第12回目

・配列変数の宣言：

以下のようにすることで、配列変数を用意できる。

例）　int x[3]; → 符号付整数型で変数"x[0]"、"x[1]"、"x[2]"を用意。
int x[3] = {12, 4, 71}; →符号付整数型で変数"x[0]"、"x[1]"、"x[2]"を用意し、それぞれに12、4、71を代入する。

実際に用意される配列は、0から始まり、（指定した数－1）までとなることに注意する。

・計算と変数への数値の代入：

演算	算術演算子	記述方式
加算	+	a + b
減算	−	a - b
乗算	*	a * b
除算	/	a / b
剰余算	%	a % b

代入演算子（＝）

"a = b + 10;" と記述することで、" b + 10" の結果を"a"に代入する。従って、現在の"a"から"b"を引いて、改めて"a"に代入する場合、" a = a - b ; " と記述する。

・論理演算子・比較演算子:

論理演算および比較演算は以下のように表記する。

演算	論理演算子 比較演算子	記述方式
and	&&	a * b
or	||	a / b
equal	==	a == b
not equal	!=	a != b
larger	>	a > b
larger equal	>=	a >= b
smaller	<	a < b
smaller equal	<=	a <= b

これを用いることで、プログラム中で、条件による制御や分岐が可能となる。

・条件によって処理を変える(その4)

```
switch( 変数 )
  {
    case 変数の値１:
      処理１
      break ;
    case 変数の値２:
      処理２
      break ;
    default :
      処理３
      break ;
  }
```

指定した「変数」が、「変数の値1」の時、「処理1」、「変数の値2」の時、「処理2」を実行する。変数の値に対する処理は、幾つも増やすことができる。

「変数」の値が、どれにも該当しない場合は、「default」に記された「処理3」を実行する。

注) 各条件の末尾には「:」(コロン)を記す。

注) 各条件分岐による処理の最後には「break;」を記す。

・文字を表示する。

```
printf (" 表示させる文字 ");
```

注) 各命令の最後に「;」(セミコロン)を付けることを忘れないこと。

注) 表示を改行したい場合は、その位置に「¥n」を入れる。

・変数に入力された数値を表示する。

```
printf (" 表示させる数字の書式 ", 表示させる変数名 );
```

特別な意味を持った文字列をエスケープ・シーケンスという。

・変数に数値を手動入力する。

```
scanf (" 入力したい数字の書式 ", &入力先の変数名 );
```

scanfの場合、変数名の直前の「&」を忘れないこと。

数字や文字の書式としては、以下のようなものがある。

%c	char型	1文字出力	%ld	long型	10進数出力
%d	int型	10進数出力	%f	float, double型	表示ケタ数を指定
%o	int型	8進数出力	%e	float, double型	指数形式出力
%x	int型	16進数出力	%s	Char型配列	文字列出力

・条件によって処理を繰り返す(その1)

```
for( 変数の初期値 ; 条件式 ; 増分 )
  {
    処理
  }
```

ある「変数」に対して初期値を与え、「条件式」を満たす(真である)場合、「処理」を実行する。この後、「増分」に従って、「変数」の値を加減し、その度に「条件式」が真ならば処理を繰り返す。

for文の中にfor文を記述することで、多重ループ(ネスト)処理を行うことができる。

```
例) for( i=0; i < 10; i=i+1)
      {
        printf( "%d¥n" , i);
      }
```

「i」を0から1ずつ増やして、10を超えるまで、「i」の値を表示し改行を続ける。

「i=i+1」の部分は、「++i」「i++」というインクリメントで記述してもよい。同様に1つずつ減らす場合はデクリメント「--i」「i--」を使用できる。

・条件によって処理を変える(その1)

```
if( 条件式 )
  { 処理１ }
```

「条件式」が真の場合、「処理1」を実行する。

・条件によって処理を変える(その2)

```
if( 条件式 )
  { 処理１ }
else
  { 処理２ }
```

「条件式」が真の場合、「処理1」を実行する。「条件式」が偽の場合、「処理2」を実行する。

・条件によって処理を変える(その3)

```
if( 条件式１ )
  { 処理１ }
else if( 条件式２ )
  { 処理２ }
else
  { 処理３ }
```

「条件式1」が真の場合、「処理1」を実行する。「条件式1」が偽で、「条件式2」が真の場合「処理2」を、偽の場合は「処理3」実行する。「else if」の条件分岐の部分は、幾つも増やすことができる。

・条件によって処理を繰り返す(その2)

```
while( 条件式 )
  {
    処理
  }
```

「条件式」を満たす(真である)場合、「処理」を実行し続ける。

while文では、条件判定の後、処理が行われる。

・条件によって処理を繰り返す(その3)

```
do
  {
    処理
  } while ( 条件式 ) ;
```

「条件式」を満たす(真である)場合、「処理」を実行し続ける。

do～while文では、処理を行った後に、条件判定が行われる。

do～while文の場合、「while(条件式)」の後に、「;」(セミコロン)を付ける。