情報処理実習３「時系列データ処理とツール」
教材２「C 言語入門」

纐纈 一起（こうけつ かずき）

第 1 版（2003 年 6 月 16 日）
第 2 版（2004 年 6 月 14 日）

1. コンパイル

コンピュータに対して，どのような手順で動作をすべきかを指示するものを**プログラム** *program*，または**コード** *code* という．プログラムは人間が書くものであるから**人間可読** *human readable* でなければならない．これをコンピュータが理解可能な（**機械可読** *machine-readable* な）**機械語** *machine language* に変換する作業を**コンパイル** *compile* という．

プログラムを記述する人間可読な**プログラミング言語** *programming language* には，まず機械語と一対一に対応する**アセンブラ** *assembler* があるが，より自然言語に近い高級言語を用いる方がプログラミングは容易である．高級言語にはここで紹介する C やその拡張である C++をはじめ，FORTRAN, Pascal, BASIC などがある．

C は Linux など UNIX を起源とする**オペレーティングシステム** *operating system*（**OS**）が標準で備えているだけでなく，OS 自体も C で記述されているものが多い．つまり，C はシステムの記述も可能という意味でアセンブラに近い，比較的低水準なプログラミング言語である．しかし，数値計算やテキスト処理も可能な言語仕様も併せ持ち，幅広い適用範囲があるので現状では事実上の標準プログラミング言語と言える．

プログラミングはまず，vi, emacs などのエディタを使ってプログラムを記述した**ソースファイル** *source file* 作成することから始まる．C のソースファイルはその名前の最後が「`.c`」で終らなければならない．たとえば xyz.c というソースファイルのコンパイルして**実行可能な** *executable* ファイルを作るには

```
cc xyz.c
```

と cc コマンドを実行する．この場合，実行可能ファイルの名前は `a.out` となるが

```
cc -o abc xyz.c
```

とすれば `abc` という名前にすることができる．上の `cc` は実はコンパイルするだけでなく，コンパイル結果を実行可能ファイルに変換する**リンク** *link* の作業も同時に行う．コンパイルだけ行うには

```
cc -c xyz.c
```

とすればよい．コンパイル結果は `xyz.o` というファイルに書かれ，これを**オブジェクトファイル** *object file* と呼ぶ．オブジェクトファイルをリンクして実行可能ファイル `abc` に変換するには

```
cc -o abc xyz.o
```

とする．もしプログラムの中で数学関数を使っていたら，あらかじめ用意されたそれらのオブジェクトファイルの集合体（**ライブラリ** *library* という）も読み込んでリンクするために

```
cc -o abc xyz.c -lm    または    cc -o abc xyz.o -lm
```

としなければならない．

2. 文

プログラムはいくつかの**文** *statement* で構成される．ひとつの文は 1 行以上にまたがってよい．ふたつ以上の文を 1 行に書いてもよい．文の終りはセミコロン「;」で示す．複数の文を大括弧「{ }」でくくって**ブロック** *block* を作り，それを文と同様に扱うことができる．ブロック末尾の「}」の後に「;」は付けない．文には表 1 の種類がある．

種類	例
式からなる文	x = 3;
宣言文	int x;
制御フロー文	if(x < 1) y = 2;
関数定義文	int func(a) int a; { return(a); }
関数呼出し文	func(4);

表 1. 文の種類.

文の要素の区切りはスペース，タブ，改行が使われ，これらをいくつ使ってもかわないので，以下のようにプログラムを読みやすく整形するために活用すべきである．コメントは「/*」で始まり「*/」で終わる．スペースが許されるところならばどこにおいてもよい．

```
/* for loop */
for(i=0; i<n; i++) {
        a[i] = b[i]; /* 代入     */
        k++;         /* カウンタ */
}
```

3. 式

式 *expression* は**演算子** *operator* とその**オペランド** *operand* からなっている．主な演算子を表 2 に示した．

4. 識別子とデータ型

変数や関数の名前を**識別子** *identifier* という．識別子に許される文字は英数字と下線「_」であり，先頭文字は数字であってはならない．また，すべての変数，関数はその**データ型** *data type* を，**宣言文** *declaration statement* または関数定義文で宣言されなければならない．データ型には表 3 に示すような種類がある．なお，1 **バイト** *byte* は 8 **ビット** *bit* を表し，データ型のバイト数は CPU により異なる．表は SPARC の場合を示すが，Pentium でもバイト順を除いて同じである．SPARC

種類	演算
算術式 + − ∗ / % ++ −−	 足し算，引き算，掛け算，割り算，余り演算． 1 オペランドのときの − は負号を意味する． インクリメント（1 増やす），デクリメント（1 減らす）． オペランドが前（後）に付くとき式の値は増減の前（後）の値．
代入式 = += −= ∗= /= %=	 単純代入． 足し算，引き算，掛け算，割り算，余り演算の後，代入．
関係式 == != > < >= <=	 オペランドが等しい，等しくないとき真． オペランドが大きい，小さい，以上，以下のとき真．
論理式 ! || &&	 否定，論理オア，論理アンド．

表 2. 主な演算子の種類．

種類	宣言	バイト数
文字型	char	1
整数型	int	4
短整数型	short	2
長整数型	long	4
浮動小数点型	float	4
倍長浮動小数点型	double	8

表 3. データ型の種類．

は 4 バイトのデータを低いアドレスから 3 2 1 0 の順に並べるのに対して（big endian），Pentium は 0 1 2 3 の順に並べる（little endian）．

整数と扱うことができる文字型，整数型，短整数型，長整数型には符号なしの修飾語 `unsigned` を付けることができる．また，**配列** *array* の変数は［ ］を付け，要素数を与えて以下のように宣言する．配列の最初の要素は［`0`］である．

```
unsigned int pqr[10];
```

整数型の定数は通常，10 進数で表すが，先頭が 0 の場合は 8 進数と解釈されるので，076 は 10 進数の 62 を意味する．16 進数表現にしたいときは先頭に 0x を付ける．また，`long` であることを明示するには末尾に L を付ける．一方，浮動小数点型の定数は小数点のある小数表現で表し，43.2e-5 などと指数部を末尾に付けてもよい．すべての浮動小数点型定数は `double` として扱われる．

文字型の定数は単引用符で囲んで ’A’ などと表すが，改行のような特殊文字は ’＼n’ と表す．文字型の配列は文字列となる．文字列の定数は複引用符で囲んで ”abcd” などと表し，文字列を変数の途中で終結させるにはその部分にヌル文字 ’＼0’ を与える．

5. 制御フロー

文の実行順序は表 4 の制御フロー文により制御される．

種類	演算
if(式) 文;	式が真なら文を実行，偽なら何もしない．
if(式) 文 1; else 文 2;	式が真なら文 1 を実行，偽なら文 2 を実行．
switch(式) { case 定数 1: 文 1; case 定数 2: 文 2; ⋮ default: 文 n; }	式と一致する定数の文にジャンプする． どの定数とも一致しないときは文 n にジャンプ．
while(式) 文;	式が真なら式が偽になるまで文を実行，偽なら何もしない．
do 文; while(式);	まず文が実行される．その後は while 文に同じ．
for(式 1; 式 2; 式 3) 文;	まず式 1 を実行．式 2 が真なら文と式 3 を実行，偽なら何もしない．
break;	switch, while, do, for 文の一番内側のループを抜け出す．
continue;	while, do, for 文の一番内側のループの先頭に戻り実行を継続する．
return 式;	関数から抜け出て式の値を返す．値なしの関数なら式を付けない．
goto ラベル;	ラベル: が付いた文にジャンプする．

表 4. 制御フロー文（文 i はブロックでもよい）．

制御フロー文は入れ子にすることが可能で，たとえば else 文の文 2 が if 文を含めば，以下のように三分以上の分岐制御ができる．

```
if(式1) 文1;
else if(式2) 文2;
else if(式3) 文3;
else 文4;
```

6. 関数

C のプログラムは**関数** *function* を単位として構成されており，識別子の最後に（）が付いているとき，その識別子は関数として扱われる．特にプログラムの入口は `main()` という関数に固定されているので，あらゆるプログラムは関数 `main()` を持たなければならない．

関数の定義文は以下のような形をしている．そのパラメータと返す値のデータ型を宣言し，関数が何を行うかを指定するブロックを書くことによって定義する．値を返さない場合には `void pqr()` などと void 型を宣言する．

```
double func(x, a, b)
double x;
int    a, b;
{
    int  j;
    j = a + b;
```

```
    return( j*x );
}
```

一方，関数を呼び出すには単に `func(x, a, b)` とすればよく，演算子のオペランドとすることもできる．パラメータは定数でもかまわない．C の関数のパラメータが変数で与えられていたとき，関数の中で値が変化しても，呼び出し元にその変化が反映されない．反映させるには以下のように変数のアドレスをパラメータとして与えるとともに，関数の定義をそれに応じて書き換える．なお，配列は特に指示しなくてもアドレス渡しとなる．

```
main()
{
    double  y;
    func(&y, 1, 2);
}
void func(x, a, b)
double *x;
int    a, b;
{
    *x = a + b;
    return;
}
```

7. 数学関数

主要な数学関数は用意されているが，利用する場合にはそれらを宣言するため，プログラム冒頭に

```
#include <math.h>
```

と書くだけでなく，コンパイル時に `-lm` オプションを付けなければならない．

なお，数学関数の値やパラメータはすべて double で与えられる．たとえば `sin(x)` ならば `x` も `sin(x)` も double である．どのような数学関数があるかは `/usr/include/math.h` の中味を見るとわかる．

8. 入出力

標準出力 *standard output* に変数の値を書き出したり，**標準入力** *standard input* から変数に値を読み込むには以下のように printf(), scanf() 関数を用いる．冒頭の include 文は上の場合と同じく，これら関数を宣言するために必要である．

```
#include <stdio.h>
main()
{
    double  y;
    scanf("%lf", &y);
```

```
        y = y + 1.;
        printf("%lf\n", y);
    }
```

ここで printf, scanf の最初の文字列パラメータは書式を表し，%d が整数型，%f が浮動小数点型，「l」を付けてそれぞれ long あるいは double を意味する．printf の書式の最後に＼n があるので，ここでは y の値を出力した後，改行が行われる．scanf() 関数により変数 y が変化しなければならないから，y はそのアドレスをパラメータとして与えなければならない．なお，これら関数についてはコンパイル時に特別なライブラリを指定する必要はない．

9. 演習問題

標準入力から浮動小数点の値を読み込み，その値の余弦を計算して標準出力に書き出すプログラムを，onaga 上の C で作成せよ．なお，読み込み・計算・書出しは無限に繰り返すものとし，負の値が読み込まれたときにプログラムが終了するようにせよ．

次回，プログラムに説明文を付けたレポートを提出すること．