情 報 処 理

1　はじめに

機械の知能化にはコンピュータが不可欠である．機械に組み込む用途でコンピュータを扱うには，ワープロや表計算ソフトの利用では決して身につくことのない，もう一歩踏み込んだ IT スキルが要求される．そのようなスキルの 1 つに C/C++言語[1)]によるプログラミングがある．例えば，C/C++言語のスキルなしに，機械の知能化に携わるのは不可能であろう．

本実験では，C++言語を用いたプログラミングを学ぶ．開発環境には，C/C++ 言語と相性のよい UNIX 系のオペレーティング・システム (OS: 基本ソフト) を用いる[2)]．

1〜**4** 章で必要な基礎を習得し，**5** 章でレポート課題に取り組む．(本実験で体験する内容は，C++言語の機能のごく一部にしかすぎない．詳細は適当な入門書を参照のこと)

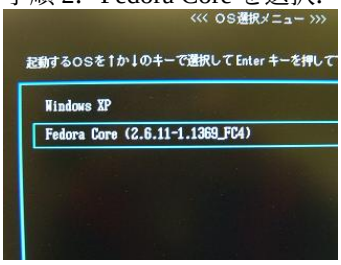
2　コンピュータの基本操作

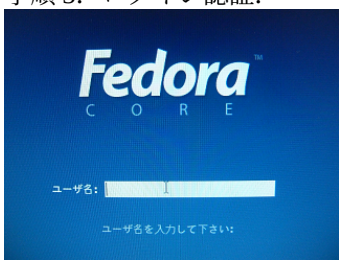
センターの端末は，起動時に，Windows と UNIX (Fedora Core[3)]) のどちらを利用するか選択できる．本単元では，科学技術計算に適した UNIX を利用する．

2.1　起動とログイン

起動方法は，左から順に，以下の通り．

手順 1．電源を投入．　手順 2．Fedora Core を選択．　手順 3．ログイン認証．

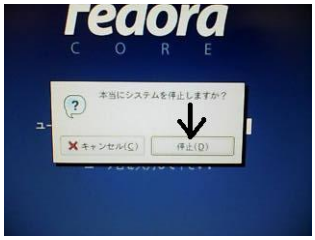




ログイン認証は，ユーザ名をタイプして(Enter)．パスワードをタイプして(Enter)の要領で行なう．

2.2　ログアウトと停止

停止するときは，左上メニューからログアウトしたあと，マウス操作で OS を停止させてから，ディスプレイを切る．

[1)] 歴史的にまず C 言語があり，その拡張版として C++ 言語が作られた．あわせて C/C++と表記した．

[2)] UNIX 自体が C 言語で開発されている．

[3)] UNIX は一社独占ではないため，いくつかの派生版が共存している．Fedora Core はその 1 つ．Fedora Core は GNU Project License(GPL) というライセンスに基づいて無償配布されている．科学技術計算用のフリーソフトが数多く動作する．

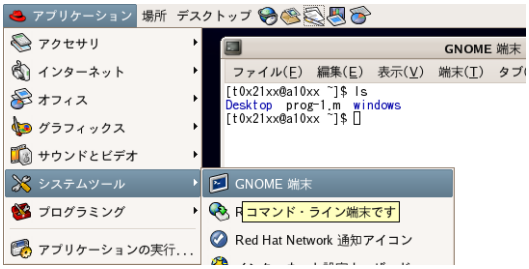
2.3 端末の起動

コンピュータが人間の役に立つためには，

1. 人間が，コンピュータに何らかの指示を与える．
2. その結果を，コンピュータが人間に返答する．

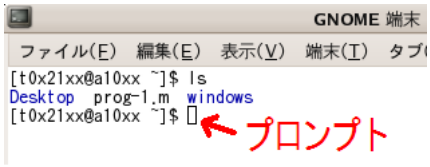
という一往復のやりとりが必ず必要である．このやりとりをマウス操作で行う方式を GUI (graphical user interface)，キーボード操作で行う方式を CUI (character user interface) という．

CUI を利用するには，まず「端末」と呼ばれるソフトを起動する．



2.4 コマンドの実行

具体例で示す．端末上の**プロンプト**と呼ばれる部分↓



に続けて，例えば試しに，(l)(s)(Enter)とキーボードを打つと，

```
[t0x21xxa10xx ~]$ ls(Enter) ←指示
Desktop    windows ←返答．最初はシステム用のフォルダしかない．
```

というような感じの出力を得る (各自の使用状況で異なる)．ちなみに，`ls` とは，現在のフォルダにあるファイルを一覧するためのコマンドである．代表的なコマンドを，表 1 にまとめた．

表 1: 端末上の基本コマンド

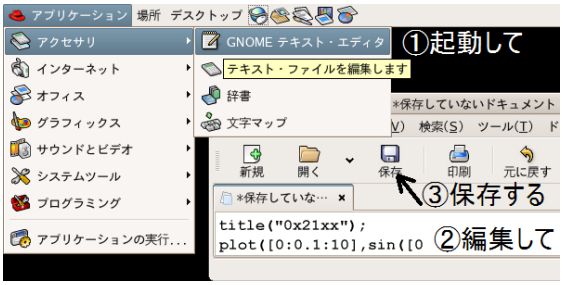
`exit`	利用を終了する．
`pwd`	現在どのフォルダ (UNIX 用語ではディレクトリ) に居るか表示する．
`ls`	フォルダにあるファイルを一覧表示する．
`cd`	フォルダを移動する． `cd` 自分のフォルダに戻る． `cd ..` １つ上のフォルダに移動する． `cd abc` abc という名前のフォルダに移動する．
`rm`	ファイルを削除する． `rm abc` abc という名前のファイルを削除する．
`cp`	ファイルをコピーする． `cp abc def` abc というファイルを def というファイルにコピーする．

2.5　ファイルの編集

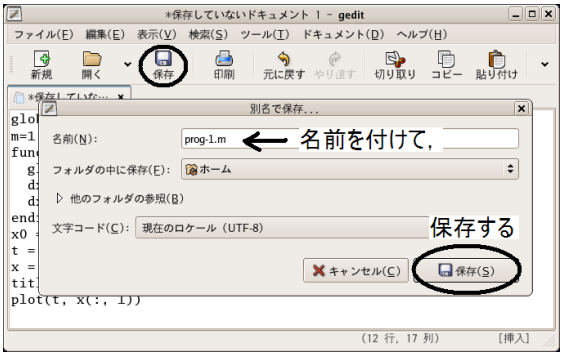
UNIX では，各種の情報を「テキスト・ファイル」という形式で保存する．テキスト・ファイルを作成したり，その内容を閲覧，編集するために「テキスト・エディタ」というアプリケーションを利用する．本単元では，gedit というテキスト・エディタを用いる．

※以下，プロンプトを `[... ~]$`と略す．

左上の「アプリケーション」メニューから，gedit を起動する．



試しに何か書き込み，適当な名前をつけて，保存しよう．



実際にファイルができたか，端末の上で確認してみる．

`[... ~]$ ls`Enter

`Desktop/ prog-1.m windows/` ※確かに，`prog-1.m` ができている！

ファイルを閲覧したり，修正するには，

`[... ~]$ gedit prog-1.m`Enter

とすればよい．

3　グラフの作成と印刷

Gnuplot というグラフ作成ツールを用いると，数式やデータのグラフを，簡単に作成できる．

3.1　グラフの作成 — gnuplot 入門

端末のプロンプトに，

```
[t0x21xxa10xx ~]$ gnuplotEnter
```

とタイプすることによって gnuplot が起動され，図 1 のように Gnuplot のプロンプトが入力待ちの状態になる．このプロンプトにコマンドをタイプすることによってグラフが作成される．

```
[t0x21xx@a10xx ~]$ gnuplot

        G N U P L O T
        Unix version 3.7
        patchlevel 1 (+1.2.0 2001/01/11)
        last modified Fri Oct 22 18:00:00 BST 1999

        Copyright(C) 1986 - 1993, 1998, 1999
        Thomas Williams, Colin Kelley and many others

        Type ‘help‘ to access the on-line reference manual
        The gnuplot FAQ is available from
        <http://www.ucc.ie/gnuplot/gnuplot-faq.html>

        Send comments and requests for help to <info-gnuplot@dartmouth.edu>
        Send bugs, suggestions and mods to <bug-gnuplot@dartmouth.edu>


Terminal type set to ’x11’
gnuplot>
```

図 1: Gnuplot の起動画面

簡単な使い方を例によって説明する．まず，以下のファイル（ファイル名はなんでも構わないが，ここでは説明のため「gnuplot.dat」とする) を gedit を使って作成せよ．

```
# ここはコメント．gnuplot は # 記号以降の文字を無視する．
# gnuplot.dat
#x       y
1.0    -1.0
2.0     1.0
3.0    -1.0
4.0     1.0
5.0    -1.0
```

そして，Gnuplot を起動し，Gnuplot のウインドウから，

```
gnuplot> plot "gnuplot.dat"Enter
```

とタイプすると図 2 のようなグラフが作成される．

図 2 は点と枠が重なって見づらい．gnuplot では，x と y 範囲を指定してプロットすることができる．範囲を指定するには，

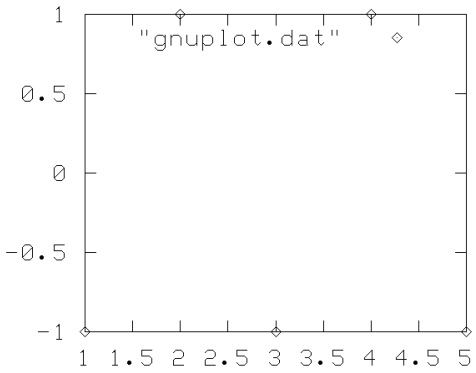


図 2: Gnuplot によるプロットの例（点と枠が重なっているので見づらい）

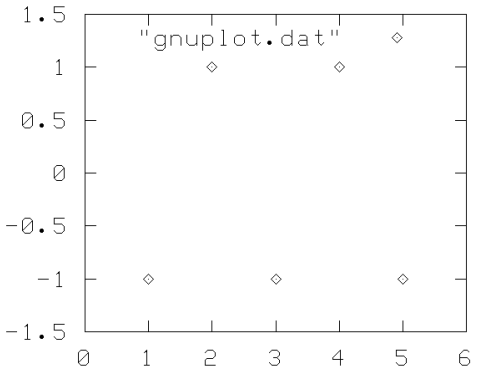


図 3: x と y の範囲を指定してプロット

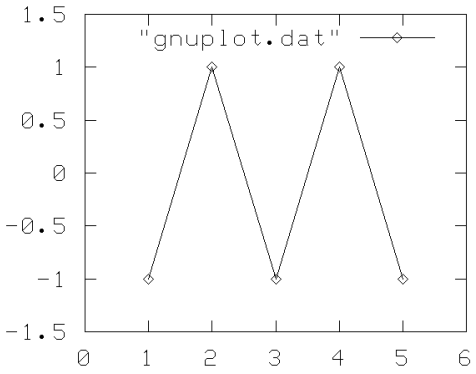


図 4: 点を線で結んでプロット

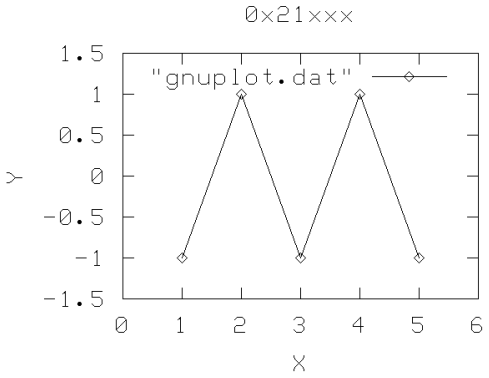


図 5: タイトルの表示

```
gnuplot> plot [0:6][-1.5:1.5] "gnuplot.dat"Enter
```

と入力する．これによって，図 3 のようなグラフが表示される．さらに，グラフを点と線でつなぎたい場合には，

```
gnuplot> plot [0:6][-1.5:1.5] "gnuplot.dat" w linespEnter
```

と入力する (`with` は `w` と短縮可)．すると，図 4 のようなグラフとなる．また，点には△や×など，線にはさまざまな破線が用意されており，次のように，

```
gnuplot> plot [0:6][-1.5:1.5] "gnuplot.dat" w linesp 4 5Enter
```

番号で指定する．ただしこの例で,「4」は線種,「5」とは点種の設定番号である．また，x 軸や y 軸のラベル，グラフのタイトルを付すには次のように行う．

```
gnuplot> set title "0x21xxx"Enter
```
※他人と区別するため学籍番号をタイトルにせよ!

```
gnuplot> set xlabel "X"Enter
gnuplot> set ylabel "Y"Enter
gnuplot> replotEnter
```

gnuplot では,「plot」や「replot」のコマンドを実行しないと設定が画面に反映されない．そのため,「set title」などのコマンドの後には，必ず「plot」か「replot」を実行する必要がある．また,「replot」を実行することによって前に描画してるデータに重ね書きすることもできる．図 5 に上記のコマンドを実行した結果を示す.

さらに gnuplot では，直接関数を書くこともできる．例えば,

```
gnuplot> plot sin(x)Enter
```

とすると，正弦波が描画される．

最後に，gnuplot を終了するときには,

```
gnuplot> quitEnter
```

とタイプする．

gnuplot の基本的なコマンドを，表 2 に示しておく．

表 2: Gnuplot の基本的なコマンド

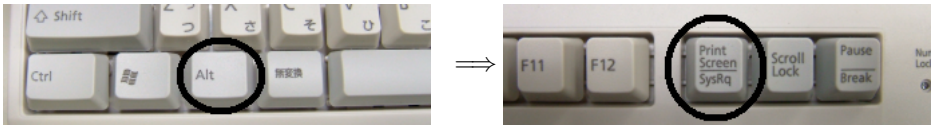
プロット	plot “ファイルを指定”	ファイルの数値をグラフにする．
	replot	再描画を行う．
	plot 計算式	計算式をグラフにする．
	[例] plot sin(x)	
	C 言語などの一般的な関数表記	
	** べき乗	/ 商
	* 積	% 乗余
	! 階乗	
表題	set title “名称”	画面の上部に名称を表示する．
	set xlabel “*x* 軸の名称”	*x* 軸の名称を表示する．
	set ylabel “*y* 軸の名称”	*y* 軸の名称を表示する．
プロットのスタイル	with lines [線番号]	線でつなぐ．
	with points [点番号]	点を打つ．
	with linespoints [線番号] [点番号]	点を打ち，線でつなぐ．
	with impulses	*x* 軸から垂線を引く．
線番号	1	実線
	2〜4	点線
	5〜6	一点破線
点番号	1 ＋	4 □
	2 ×	5 ■
	3 ＊	6 ○

3.2 グラフの印刷

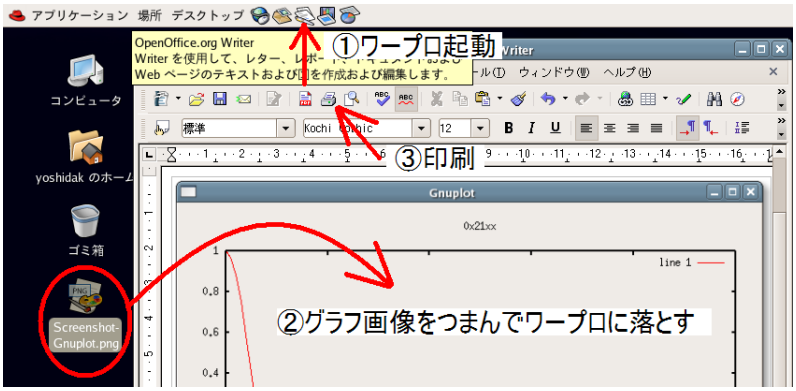
(a) 印刷したいグラフをクリックして「アクティブ」にする．

アクティブでない Window ⟹ アクティブな Window

(b) Alt を押したまま Print Screen を押すことで，グラフの画像をデスクトップに保存する．



(c) グラフの画像ファイルをつまんで，ワープロに落し，ワープロ文書として印刷する．



※ **他人のプリントアウトと区別するため「学籍番号」を記せ．**条件や考察を書き込んでおくと，そのままレポートに使える．

4 C++入門

プログラミング言語を使うと，コンピュータに次のような処理を実行させることができる．

- データの保持と加工 (変数宣言，代入，四則演算，etc)
- 繰り返し
- 条件分岐
- 入出力 (人間とのやりとり)

プログラマは，以上の処理の組み合せを文字情報として記述する．この文字情報を**プログラム**という．文字情報であるプログラムは，プログラミング言語によって，デジタル信号 (2 進数の列) に変換され，その指令どおりに CPU が動作する．デジタル信号への変換には 2 つの方式がある．

- インタプリタ： プログラムを一文単位で変換，実行する．
  例) UNIX のシェル (端末のプロンプト)，Octave.
- コンパイラ： プログラムを全て変換し終えてから，実行する．
  例) C，C++，JAVA.

一文ごとに処理を完結させるインタプリタは，コンパイラより実行速度が遅い[4]．したがって，例えば 3 次元 CG を動かすような用途には，コンパイラが適している[5]．以下では **C++** と呼ばれるコンパイラを用いて，プログラミング演習を進める．

[4] 例えるなら，辞書を引いては 1 回ずつ本棚に戻すような無駄が生じる．
[5] リアルタイム CG 用のインタプリタがあれば便利なのだが，決定的なものは現状見当たらないようである．

4.1 C++コンパイラの使い方

標準的な Unix 環境を前提に話を進める．まず，C++プログラムの一例として，次の内容をもつファイル sisoku.cpp を適当なエディタで作成せよ．

C++ プログラム：“sisoku.cpp”

```
//ここはコメント．"//"以降を C++は無視する．
int main( int argc, char* argv[] ) { //この行から最後の中括弧までを main 関数という
   double x=0.0, y=0.0;     //倍精度小数型の変数宣言
   x=5.0;                   //数値の代入
   y=1.0 + x - x*x / 4.0;   //四則演算と，その結果の代入
   return 0;                //shell に正常終了を通知する決まり文句
}
```

C++言語の基本ルールとして，C++プログラムは必ず **main 関数** を含まなければならない．

コンパイルと実行　このファイルを次の要領でコンパイルすると，CPU への命令からなるデジタル信号を収めた**実行ファイル** (executable file) ができあがる．実行ファイルのファイル名は “a.out” となる．

```
[... ~]$ ls
sisoku.cpp
[... ~]$ c++ sisoku.cpp
[... ~]$ ls
a.out  sisoku.cpp
 ↑コンパイラが作った実行ファイル
```

実行ファイル名の指定　実行ファイル名を指定したいときは，“-o 実行ファイル名” というオプションを付加する．

```
[... ~]$ rm a.out
[... ~]$ ls
sisoku.cpp
[... ~]$ c++ sisoku.cpp -o sisoku
[... ~]$ ls
sisoku.cpp  sisoku
              ↑コンパイラが作った実行ファイル
```

さっそく実行してみよう．

```
[... ~]$ ./sisoku
[... ~]$
```

当然，何も起らない．計算結果は確かに変数 y に保持されているのだが，”sisoku.cpp”には y の値を人間に知らせる処理が何も書かれていないから，人間には y の値を知るすべがない．

4.2 入出力

そこで，処理結果を人間に知らせるために 3 行書き足す．

C++ プログラム：“sisoku2.cpp”

```
#include <iostream>         //入出力機能の追加
using namespace std;        //std::cout という処理を cout と短縮表記する宣言
int main( int argc, char* argv[] ) {
   double x=0.0, y=0.0;
   x=5.0;
   y=1.0 + x - x*x / 4.0;
   cout << y << endl;       //端末に y の数値を出力
   return 0;
}
```

```
[... ~]$ c++ sisoku2.cpp -o sisoku2
[... ~]$ ./sisoku2
-0.25
[... ~]$
```

以上，計算結果が我々にも判明した．ちなみに，””で囲んだ文字はそのまま出力される．

C++ プログラム：“sisoku2b.cpp”

```
#include <iostream>
using namespace std;
int main( int argc, char* argv[] ) {
   double x=0.0, y=0.0;
   x=5.0;
   y=1.0 + x - x*x / 4.0;
   cout << "If x = " << x << ", ";
   cout << "then 1.0 + x - x*x / 4.0 = " << y << endl;
   return 0;
}
```

```
[... ~]$ c++ sisoku2b.cpp -o sisoku2b
[... ~]$ ./sisoku2b
If x = 5, then 1.0 + x - x*x / 4.0 = -0.25
[... ~]$
```

以上のプログラム例では，人間とのやりとりに文字を使ったが，このような文字を使う作法を CUI (character user interface) と呼んだ．文字の代りに何らかの画像やグラフを生成して，それを人間に提示すれば，GUI (graphical user interface) ということになる．

4.3 データの保持と加工

変数の型　すでに前節で用いたものを含め，C++では次のような型の変数を利用できる．

型	宣言方法	使い道
小数型	`float x;`	実数に相当 (有限精度)
倍精度小数型	`double x;`	同上 (float の倍精度)
整数型	`int i;`	整数
文字型	`char c;`	半角アルファベット単体
文字列型	`char s[];`	半角アルファベットからなる単語

型によって代入や四則演算の法則が変化するので注意を要する．宣言時と異なる型として演算したいときは，**キャスト** (cast) という文法で型を変換する．例えば，次のプログラム例を実行せよ．

C++ プログラム：“cast.cpp”

```
#include <iostream>
using namespace std;
int main( int argc, char* argv[] ) {
   double x=0.0, y=0.0, z=0.0;     // 小数型の変数宣言
   x = 10.0;
   y = 3.0;
   z = x/y;                  // 小数の除法: z=3.333...
   cout << "double/double: x/y = " << z << endl;
   int i=0, j=0;                   // 整数型の変数宣言
   i = 10;
   j = 3;
   z = i/j;                  // 整数の除法では小数部は切捨: z=3
   cout << "int/int: i/j = " << z << endl;
   z = (double)i/(double)j; // 整数 i,j を小数へキャスト: z=3.333...
   cout << "double/double: i/j = " << z << endl;
   return 0;
}
```

構造体　使用する変数が増えると命名に苦労するが，**構造体** (struct) という文法を使うと，変数に名字を付けるようにして整理できる．構造体に所属する変数を**メンバー変数**という．

C++ プログラム：“struct.cpp”

```
#include <iostream>
using namespace std;
struct Point { //メンバー構成を定義する (例えば 2 次元座標)
  double x, y; //メンバーは小数型の x と y
};
int main( int argc, char* argv[] ) {
   Point p, q; //個別の名字をつける
   p.x = 2.0;
   p.y = 3.0;
   q.x = 5.0;  //無理に不自然な命名をしなくて済む
   q.y = -1.5;
   cout << "p = (" << p.x << "," << p.y << ")" << endl;
   cout << "q = (" << q.x << "," << q.y << ")" << endl;
   return 0;
}
```

ユーザ関数の定義　定型処理を**関数** (function) として取り分けておくと，タイプ量を減らせる[6)]．

C++ プログラム：“function.cpp”

```
#include <iostream>
using namespace std;
struct Point {
   double x, y;
};
void print( Point p, char s[] ) {//関数の定義.
  //ここの p は代入された p. main 関数の p とは別物.
  cout << s << "(" << p.x << ", " << p.y << ")" << endl;
}
int main( int argc, char* argv[] ) {
  Point p, q; //この p は，print 関数の p とは別の p.
  p.x = 2.0;
  p.y = 3.0;
  q.x = 5.0;
  q.y = -1.5;
  print( p, "p = " );  //定義しておいた処理が実行される.
  print( q, "q = " );
  return 0;
}
```

C++に標準で搭載される関数を**標準ライブラリ関数**，自分で定義した関数を**ユーザ関数**という．

4.4　繰り返し (loop)

ソフトウェア的に数列 $\{x_1, x_2, \cdots, x_n\}$ を実現するのに必要な方法を述べる．

配列の使い方　数列 $\vec{x} = \{x_1, x_2, \cdots, x_n\}$ のデータを保持するために，**配列**という連番型の変数を使う．C/C++の配列は，デジタルコンピュータ (CPU) の構造をそのまま反映している．CPU は 2 次元の自然数ベクトル (a_i, d_i) を操作できるよう設計されており，第 1 の自然数 a_i を**アドレス**，第 2 の自然数 d_i を**データ** という[7)]．

例えば，次のように書くとメモリ上に 3 個のベクトル $(a_0, d_0), (a_1, d_1), (a_2, d_2)$ が確保される．

[6)] Fortran などではサブルーチンという．

[7)] 例えば，キーボードやディスプレイなどの各種デバイスの状態は，CPU から見ると，デバイス固有のアドレス $a_i, a_j, a_k, \cdots$ に格納された自然数 $d_i, d_j, d_k, \cdots$ に見える．

```
double *x = new double[3];
```

アドレス値 a_i を参照するときは (x+i)，データ値 d_i を参照するときは x[i] と書く．確保したベクトルをメモリ上から削除するには，

```
delete x;
```

と書く．以下に具体的なプログラム例を示す．

C++ プログラム：“pointer.cpp”

```
#include <iostream>
using namespace std;
int main( int argc, char* argv[] ) {
   double* x = new double[3]; // *を忘れずに！
   x[0] = 0.1;
   x[1] = 0.4;
   x[2] = 0.9;
//   x[3] = 0.9;   //未確保の連番は使用禁止．致命的なバグになる．
   cout << "(" << x << ", " << x[0] << ")" << endl;
   cout << "(" << x+1 << ", " << x[1] << ")" << endl;
   cout << "(" << x+2 << ", " << x[2] << ")" << endl;
   delete x;
   return 0;
}
```

```
[... ~]$ ./pointer
(0x4f0308, 0.1)
(0x4f0310, 0.4)
(0x4f0318, 0.9)
```

アドレス (第 1 の自然数) が 16 進数，データ (第 2 の自然数) が小数で表示されている[8)]．

for 文　ちょっと脱線して，例えば次のように書くと，0 から 9 までの数値が順に表示される．

```
for ( int i=0; i<10; i=i+1 ) { // i=i+1 は i+=1 または i++ と略記可能
   cout << i << endl;
}
```

あるいは，1 つおきに表示したければ，カウンタ i の増分を 2 に増して，

```
for ( int i=0; i<10; i=i+2 ) { // i=i+2 は i+=2 と略記可能
   cout << i << endl;
}
```

とすればよい．for 文と配列の組み合わせで数列を操作できる．等差数列を出力する典型例を示す．

C++ プログラム：“loop.cpp”

```
#include <iostream>
using namespace std;
int main( int argc, char* argv[] ) {
   int    xn=5;                    //項数
   double xmin=-5.0;               //最小値
   double xmax= 5.0;               //最大値
   double Dx=(xmax - xmin)/(double)(xn-1); //公差
   double* x = new double[xn];     //配列を作る
   for ( int i=0; i<xn; i++ ) {    //i=0 から i=xn-1 まで{ }の中身を繰り返す
      x[i] = xmin + Dx*(double)i; //等差数列の代入
   }
   for ( int i=0; i<xn; i++ ) {
      cout << x[i] << endl;        //配列の内容を表示
   }
   delete x;                       //配列の破棄
   return 0;
}
```

[8)] 小数といっても内部的にはデジタル信号 (2 進数) だが，C++ が x の型を見て倍精度小数として表示してくれている．

4.5 条件分岐

状況に合せて処理を変更する操作を，条件分岐という．プログラムに**知能**を持たせるために必須の機能であり，生物が有する論理思考を，機械が模倣するための文法ともいえる．

論理式　デジタルコンピュータであれば例外なく，数理論理学の計算を実行できる[9]．数理論理学では命題の真偽を議論するが，C++で扱える命題は次のような条件式である．

数学記号	$a=b$	$a\neq b$	$a>b$	$a<b$	$a\geq b$	$a\leq b$
C，C++	a==b	a!=b	a>b	a<b	a>=b	a<=b

これらの条件式は真偽に応じて必ず 0 か 1 の値をとる．これらの記号 ==, !=, >, ⋯ を**関係演算子**という．命題を組み合わせる論理演算を，C++では次のように書く．

論理積 (かつ)

P	Q	P && Q
1	1	1
1	0	0
0	1	0
0	0	0

論理和 (または)

P	Q	P || Q
1	1	1
1	0	1
0	1	1
0	0	0

否定 (でない)

P	!P
1	0
0	1

これらの記号 &&, ||, ! を**論理演算子**という．関係演算子と論理演算子の組み合わせを**論理式**という．例えば $0\leq x<1$ という条件は，C++で書くと，

```
( 0<=x && x<1 )
```

という論理式になる[10]．論理式は必ず 0 か 1 の値をとる．

C++ プログラム：“logic.cpp”

```
#include <iostream>
using namespace std;
int main( int argc, char* argv[] ) {
   double x=0.5;
   cout << ( 0<=x && x<1 ) << endl;
   x=5.0;
   cout << ( 0<=x && x<1 ) << endl;
   return 0;
}
```

```
[... ~]$ ./logic
1
0
```

if 文　**条件分岐**の文法を示す．

最も単純な省略形は，

```
if ( 論理式 ) {
  論理式が真 (1) のときの処理
}
```

省略しないで書くと，

```
if ( 論理式 ) {
  論理式が真 (1) のときの処理
}
else {
  論理式が偽 (0) のときの処理
}
```

複数の論理式で分岐させたいときは，

```
if ( 論理式１ ) {
  論理式１が真 (1) のときの処理
}
else if ( 論理式２ ) {
  論理式２が真 (1) のときの処理
}
else if ( 論理式３ ) {
  論理式３が真 (1) のときの処理
}
else {
  それ以外のときの処理
}
```

[9] そもそもデジタルコンピュータは数理論理学を自動計算するための機械である．そこから四則演算などを派生させた．
[10] これを 0 <= x < 1 と略記する文法は用意されてないので注意を要する．

というような書き方をする．条件の数に制限はない．条件分岐の応用例として，区分関数[11] を作る C++プログラムを示す.

C++ プログラム：“cond.cpp”

```
#include <iostream>
using namespace std;
int main( int argc, char* argv[] ) {
   int    xn=50;
   double xmin=-5.0;
   double xmax= 5.0;
   double Dx=(xmax - xmin)/(double)(xn-1);
   double* x = new double[xn];
   for ( int i=0; i<xn; i++ ) {
      x[i] = xmin + Dx*(double)i; //等差数列
   }
   double y=0.0, a=1.5;
   for ( int i=0; i<xn; i++ ) {
      if ( x[i] < -a ) {
         y = -1.0; // -a より小さいとき
      }
      else if ( -a <= x[i]  && x[i] < a ) {
         y =  0.0; // -a と a の間のとき
      }
      else {
         y =  1.0; // それ以外： a より大きいとき
      }
      cout << x[i] << " " << y << endl;
   }
   delete x;
   return 0;
}
```

単に実行するだけだと，グラフの座標を表わす数値が流れるだけで意味をなさない.

```
[... ~]$ ./cond
(中略)
4.59184 1
4.79592 1
5 1
[... ~]$
```

4.6 データ・ファイルの作成

そこで，数値データをファイルに保存するために，UNIX のシェルの機能を拝借しよう．**リダイレクト** “>” という機能を使うと，プログラムの文字出力を，そのままファイルに保存できる.

```
[... ~]$ ./cond > cond.dat
```

以上で，出力内容が cond.dat というファイルに保存される．適当なエディタで内容を確認せよ．このようなデータを保持したファイルを「データ・ファイル」という.

すでに学んだように，データ・ファイルの内容をグラフ化するには，gnuplot が利用できる[12]．次のようにすると階段状の区分関数のグラフがプロットされる.

```
[... ~]$ gnuplot
(中略)
gnuplot> plot "cond.dat" w l
```

[11] 例えば，脚式ロボットの知能を書き下す際，路面への衝突や摩擦への対処法は，区分関数なしには書けない.
[12] C++でも書ける．OpenGL などのグラフィックライブラリを併用してプログラムする.

5　C++による数値計算

機械の知能化をはかるとき，知能の実体として，我々が実際に開発すべきものは，数値計算プログラムである．センサは，外界の情報を測定値という数値に変換してコンピュータに送る．センサの数値に基いて機械の未来の動作を生成するためには，周到な数値計算を実行しなければならない．

機械の知能化の詳細は別の機会に譲るが，プログラムを開発するときの頭の使い方は，以下に学ぶ初等的な例題で十分に体験できる．

5.1　数学関数

数値計算に不可欠な機能として，C/C++言語には，標準で sin，cos などの数学関数が用意されている．代表的な数学関数と数学定数を表 3, 4 に示す．

表 3: 数学関数 (抜粋)

表記	内容	備考	表記	内容	備考
`sin(x)`	$\sin(x)$		`exp(x)`	$\exp(x)$	
`cos(x)`	$\cos(x)$		`log(x)`	$\log(x)$	
`tan(x)`	$\tan(x)$		`sqrt(x)`	$\sqrt{x}$	平方根
`asin(x)`	$\arcsin(x)$	逆三角関数	`fabs(x)`	$\lvert x\rvert$	絶対値
`acos(x)`	$\arccos(x)$	逆三角関数	`rand()`	0〜`RAND_MAX` の一様乱数	
`atan2(y, x)`	$\arctan(y/x)$	逆三角関数	`srand(i)`	サンプル番号を `i` に変更 *	

* 乱数は毎回同じ系列になる．`srand(整数)` とすれば整数に応じた系列に変更される．

表 4: 数学定数 (抜粋)

表記	内容	備考
`M_E`	e	自然対数の底
`M_PI`	π	円周率
`RAND_MAX`	乱数の最大値	

数学関数を応用して振動波形の数列を作り，これを乱数で乱すプログラム例を示す．

C++ プログラム：“rand.cpp”

```
#include <iostream>
#include <stdlib.h> //rand や srand を呼び出す
#include <math.h> //数学関数を呼び出す
using namespace std;
int main( int argc, char* argv[] ) {
   int    xn=200;
   double xmin= 0.0;
   double xmax=8.0;
   double Dx=(xmax - xmin)/(double)(xn-1);
   double* x = new double[xn];
   for ( int i=0; i<xn; i++ ) {
      x[i] = xmin + Dx*(double)i; //等差数列をつくる
   }
   double* y = new double[xn];
   for ( int i=0; i<xn; i++ ) {
      y[i] = exp( - x[i]) * cos(5.0 * x[i]); //振動波形をつくる
   }
   double w=0.0, a=0.1; //a は加える乱数の大きさ．0.0 ならきれいな波形
   srand(1); //1 を別の整数にすると乱数の系列が変わる
   for ( int i=0; i<xn; i++ ) {
      w = (double)rand()/RAND_MAX; //0〜1 の一様乱数
```

```
      y[i] = y[i] + a*w; //乱数を重ね合わせる
   }
   for ( int i=0; i<xn; i++ ) {
      cout << x[i] << " " << y[i] << endl; //結果の表示
   }
   delete x; delete y;
   return 0;
}
```

数学関数を使ったときは，`-lm` をつけてコンパイルする．

```
[... ~]$ gedit rand.cpp
[... ~]$ ls
rand.cpp
[... ~]$ c++ rand.cpp -o rand -lm
[... ~]$ ls
rand  rand.cpp
 ↑コンパイラが作った実行ファイル
[... ~]$ ./rand > rand.dat
[... ~]$ gnuplot
gnuplot> plot "rand.dat" w l
```

5.2 モンテカルロ法による円周率の計算

ある計算問題を解くときに，確定的な数値計算を使わずに，確率（乱数）を用いて解くことをモンテカルロ法と呼ぶ．本実験では，モンテカルロ法を用いて円周率の推定値を求める．

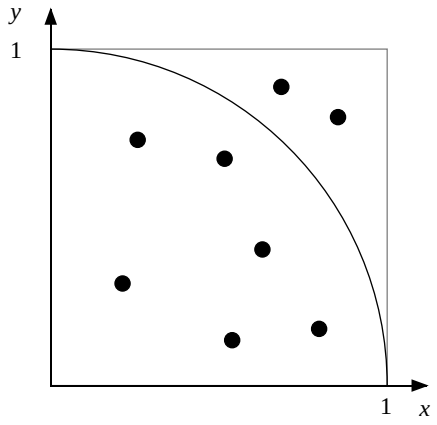


図 6: 乱数で作った座標点のプロット

まず，0〜1 の**一様乱数**を 2 つ発生させ，1 つめを x 座標，2 つめを y 座標とみなすと，座標点 (x,y) をランダムに生成できる．同様の操作を繰り返して，ランダムな座標点 (x_i,y_i) を n 個発生させると，各座標は一様乱数であるから，図 6 のように，座標点は 1×1 の正方形の中に均一にばらまかれる．このとき，正方形の面積と 1/4 円の面積の比は，それぞれにばらまかれた座標点数に比例すると期待できる．すなわち，1/4 円の中にばらまかれた座標点数を a，円外にばらまかれた座標点数を b とすると，次の関係から，円周率の近似値 $\bar{\pi}$ が推定できる．

$$\frac{\bar{\pi}}{4}:1=a:a+b \qquad \therefore \pi\approx\bar{\pi}=\frac{4a}{a+b}=\frac{4a}{n} \tag{1}$$

n は生成した座標点の総数である．

レポート課題

1. モンテカルロ法で円周率 π の近似値を求める C++プログラムを示せ.
2. プログラムを実行し，横軸を n，縦軸を円周率の近似値とするグラフを作成せよ.
3. モンテカルロ法の計算精度に影響を与える要因について考察せよ.

《ヒント》

底辺の長さが 1，高さが 1 の直角三角形の面積を推定するプログラムと，その実行例を以下に示す.

C++ プログラム：“trig.cpp”

```
#include <iostream>
#include <stdlib.h> //rand や srand を呼び出す
#include <math.h> //数学関数を呼び出す
using namespace std;
int main( int argc, char* argv[] ) {
   int    n=1000; //座標点の総数
   int    a=0;   //直角三角形の中の座標点数
   double x, y;  //座標
   double area;  //面積
   srand(1); //1 を別の整数にすると乱数の系列が変わる
   for ( int i=0; i<n; i++ ) {
      x = (double)rand()/RAND_MAX; //0〜1 の一様乱数
      y = (double)rand()/RAND_MAX; //0〜1 の一様乱数
      if ( x < y ) { //もし直角三角形の中なら
         a = a + 1;  //a を 1 増やす
      }
      area = (double)a/(double)i; //面積の推定値
      cout << i << " " << area << endl; //結果の表示
   }
   return 0;
}
```

```
[... ~]$ gedit trig.cpp
[... ~]$ c++ trig.cpp  -o trig -lm
[... ~]$ ./trig > trig.dat
[... ~]$ gnuplot
gnuplot> set xlabel "i"
gnuplot> set ylabel "area"
gnuplot> plot "trig.dat" w l
```

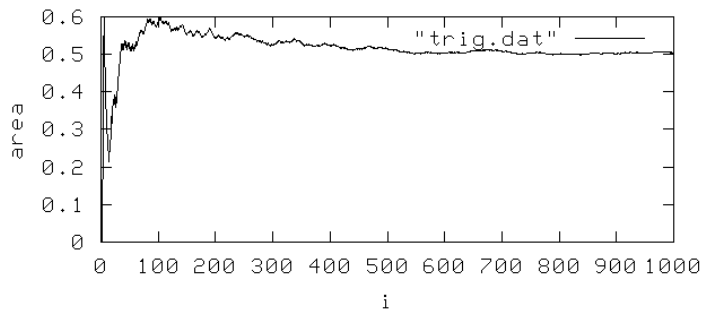


図 7: ヒントの実行例

（最終更新：平成 20 年 3 月 10 日 吉田）