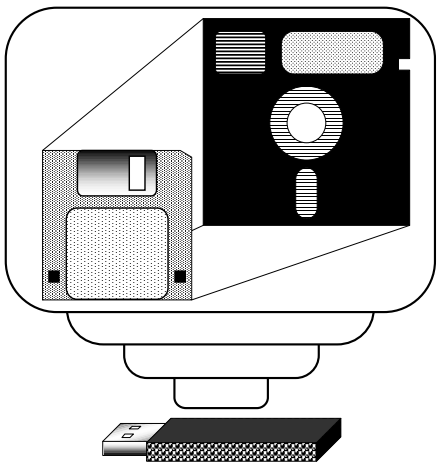
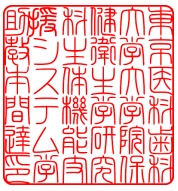
C 100



Contents



まえがき

医療情報学における情報科学実習

情報科学は他の分野と比較して，その全体像がつかみにくい学問である．文字通り受け止めるならば「情報」を「科学的に」取り扱う学問であり，そのための手法全般を含めると考えれば，自然科学分野に分類されるすべての学問が「情報科学」という学問の一分野に過ぎないと極論することもできる．

現実的には，自然科学の分野の細目として様々な学問が存在し，それぞれの領域での情報を取り扱う様々な手法に限定して取りまとめたものが「情報科学」として認識されている．したがって，主として統計学を代表とする数学全般と，そのための道具である電子計算機の原理(構造／ハードウェア)および論理(プログラム／ソフトウェア)についての知識，その応用についての学問が情報科学として認識される傾向にある．

▲医療情報学は情報科学を医療の分野で用いる場合の呼称の一つである．医療の分野は他の理系に分類される学問と異なり，業務と研究の境目がきわめて曖昧である．ゆえに医学分野における情報科学は特に，研究との関連を視野に入れるべきであろう．

■ところで研究や開発を前提とするとデータの測定は必須である．この測定時にリアルタイムでデータの確認を行い，同時に演算処理したデータ，すなわち情報の確認を行うことは，測定中に生じる異常の確認を容易にする利点がある．

研究開発が進行し，試料の採取からデータ測定・演算処理までがマニュアル化された場合には測定装置・ソフトウェアなどを包括したパッケージを利用すればよいが，進行中の研究では日々変更を要求されるものであり，これに直接関わる人間が自らの手で改良を加えなければ必ずしも現場の要求を反映したものにはなりえない．これらの点から実験装置・ソフトウェアの開発能力はあらゆる研究の場において必須のものである．

●一般的には情報科学実習の演習項目として，大別すると①プログラム技術の修得を目的としたプログラム作成，②データ処理法を主としたアプリケーションの使用演習，のいずれかが行われている．これらの演習はいずれもデータの処理に重点を置いたものであり，処理されるべきデータの測定が既に行われたことを前提としている．すなわち，研究の独自性の点から最重要と考えられるデータ実測のためのテクニックを無視した内容であることが多く，研究者の発想を狭めるのに一役買っているようである．

★現在，一般的に最も入手しやすい家庭用のパソコンはかつてのスパコン級の性能を有しており，研究・開発に利用しても遜色はない．しかしながら，これらに搭載されているOSはTSSによる擬似マルチタスクにより，常に複数のジョブを実行しつづけるタイプのものが主流であり，サンプリング周期を一定にした測定を行う上で必ずしも有効ではないが，使い方によって限りなく一定に近い条件で測定を行うことは不可能ではない．

▼本書では上記の点を考慮して，データを測定し，演算処理してグラフを描画して可視化するまでを実習の項目とした．プログラムや数学的演算処理はこのための一手段にすぎないが，最低限のスキルの修得は可能なように配慮した．しかしながら，これらについて詳解することは本来の目的ではないので，成書を参考にしていただくとした．

プログラム言語としてはC言語を採用した．コンパイラはフリーウェアとして定評のあるエル・エス・アイ・ジャパン社のLSI C-86 Ver.3.30 試食版を使用する．この試食版はメ

モリの使用について若干の制限がなされているが，その他の機能については制限されていない．また，C言語はグラフィックの描画についてANSI規格やJIS規格で定義していないのでLSI-C試食版ではサポートしていないが，小山佳孝氏製作のLSI-C試食版用簡易グラフィックライブラリと組み合わせるとグラフィックの使用が可能になる．

また，ソースファイルの記述に使用するテキストエディタは，うのしん氏が作成したPC9801版をたま吉 (川真田 光男)氏がDOS/V版として移植したNEED V1.60Aを使用することにした．

※なお，Windows VISTA以降のOSでは，上記のグラフィックライブラリおよびテキストエディタは利用できない．このため，該当するOSを利用する場合，テキストエディタとしてWindows付属のメモ帳(notepad)を利用し，8章以降の実習は行わないようにしていただきたい．

Ｃ言語的文法解説之書

何故Ｃ言語なのか

Ｃ言語は元々ＯＳ開発用に作られた言語である．したがって，機械の直接制御が可能である．さらに高級言語的要素もあり，人間にとって比較的理解しやすい関数が用意されている．これらの条件は医療機器の開発などに適していることが挙げられる．

課題

・上に挙げた以外にＣ言語の実習を行なう理由を考えること．
・もしＣ言語以外のプログラム言語を学ぶとしたら，どの言語を選択すべきか．理由とともに述べること（複数回答可）．

プログラムを作る

コンピュータがプログラムを動かすためには，コンピュータという機械にとって理解出来る形でプログラムが書かれている必要がある．この言語を機械語と呼ぶ．機械語で書かれた実行可能なプログラムを実行ファイルと呼ぶ．

ところで，機械語は人間にとって記述しにくい数字の羅列である．つまり，人間にはほとんどの場合，作成不可能である．そこで人間が記述しやすい言語でプログラムを書き，それを機械語に翻訳すれば良い．この翻訳する作業をコンパイルと呼び，コンパイル出来るプログラム言語をコンパイラ型言語と呼ぶ．

Ｃ言語はコンパイラ型言語であるので，まずソースファイルを作る必要がある．ソースファイルとは人間が理解しやすい言語で記述された，コンパイルされる元のファイルのことである．プログラムを作ると言うとき，ソースファイルを記述することと同義であることが多い．

課題

・プログラムを作る利点を3つ以上挙げ，説明すること．
・コンパイラ型言語としてはどのような言語が存在するか．幾つか挙げて，その特徴について記すこと．また，インタプリタ型言語にはどのような言語があるか．同様に記すこと．

実習で使用するコンパイラ

本実習書は，エル・エス・アイ　ジャパン（株）が提供するLSI C-86 Ver 3.30c 試食版の使用を前提として作成されている．このコンパイラは市販されているLSI C-86の評価版として配布されているフリーウェアで，Sモデルのみコンパイル可能である．極めて評価の高いコンパイラであり，これを拡張するための様々なライブラリがフリーウェアとして多くのプログラマから提供されている．現在同社による配布は行われていないが，インターネットで以下のサイトなどから入手可能である．

http://www.vector.co.jp/soft/dl/maker/lsi/se001169.html

実習で使用するエディタ

C言語のソースファイルはテキストエディタを用いて記述する．Windowsに標準で付属しているnotepadもテキストエディタの一つであり，市販されているものも多い．また，テキストエディタは比較的作成が容易であることから，多くのプログラマによって様々なものが作成されており，インターネットなどを通じて入手することができる．

本実習では，ソースファイル記述後，MS-DOSのコマンドラインによりコンパイルを実行することから，テキストエディタもMS-DOS上で動作すると便利である．そこで，インターネットを通じて配布されているNEED for DOS/Vを利用する．

このエディタはうのしん氏がNECのPC－98x1用に作成し，Nifty-Serveというネットワーク上で発表していたものを川真田光男氏がDOS/V用に移植したものである．ファイルのサイズが小さいのでフロッピーディスク上で動作するのに都合が良く，また動作も軽快で必要十分な機能を有している．本実習を目的とした場合，市販されているものと比較しても遜色のないものである．インターネットで以下のサイトなどから入手可能である．

http://www.vector.co.jp/soft/dos/writing/se025315.html

実習用フロッピーディスクの作成

本書では，フロッピーディスク，すなわち1.44MBでソースファイルの作成からコンパイルまで一切の作業を行うことを前提としている．このために，ダウンロードしたファイルから最小の実習用システムを構築する手法について紹介する．

まず，本書に付属のフロッピーディスクのコピーを作成する．これはMS-DOSコマンドのDISKCOPYコマンド(WINDOWSのMS-DOSプロンプトから実行可能)やWINDOWSのマイ コンピュータからFDのコピーなどにより可能である．また，フロッピー以外の書込み可能メディア(MO,Flash メモリなど：ＣＤやＤＶＤなどのメディアは不可)に全ての内容をコピーしても良い．

次に，LSI-C 3.30C試食版およびLSI-C試食版用簡易グラフィックライブラリ，DOS/V用テキストエディタNEED for DOS/Vをダウンロードする．また，ダウンロードするファイルはLHAやZIPなどの圧縮解凍ソフトで圧縮しているので，これらのソフトも別途用意する．

ダウンロードしたlsic330c.lzhをハードディスク上で解凍すると，いくつかのフォルダとファイルが作成される．コンパイルに必要なのはBin，Include，Libのフォルダなので，この3つのフォルダをフロッピーにコピーする．

次に，LSI-C試食版用簡易グラフィックライブラリを解凍ソフトウェアを使って解凍し，その中のGraphics.libをフロッピーのLibフォルダの下のSフォルダへ，Graph.h，Graphics.h，GraphSV.hをIncludeフォルダへそれぞれコピーする．

最後に，フロッピーにTOOLフォルダを作成し，ハードディスク上でNEED for DOS/Vを解凍して出来たファイルNEEDV.CFG，NEEDV.COM，NEEDV.HLP，NEEDV.DOCをTOOLフォルダにコピーする．

以上の手順によって，実習用フロッピーディスクが完成する．

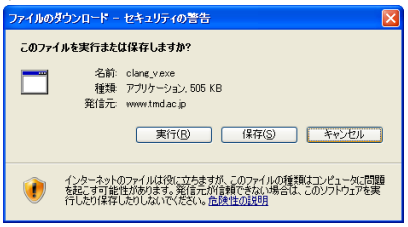
実習用USBメモリの作成

※東京医科歯科大学医学部保健衛生学科検査技術学専攻Only

近年のPCではフロッピードライブを搭載したものは少なくなっており，USBメモリを用いて実習を行えば，大学ばかりか家庭でもこれを用いて実習が可能になる．ここでは医用システム情報学(Ⅱ)実習で使用するためのUSBメモリコンパイラを作成する．

用意するもの　：　**中に何も記録されていないUSB**メモリ1本

手順①　以下のアドレスからファイルをダウンロードする．
http://www.tmd.ac.jp/med/mtec/clang_v.exe



手順②　ダウンロードしたファイル(clang_v.exe)をUSBメモリに移す．

手順③　USBメモリ上で移したファイル(clang_v.exe)を実行する．

上記の手順により，C言語コンパイラのディスクが出来たので，実習に進む．なお，このディスクは一度作成すれば，**毎回作成する必要はない**．
※　ファイルが破損した場合などを除く．

USBメモリを用いた実習の準備

手順④　Windowsのメニューから［スタート］→［すべてのプログラム］→［アクセサリ］→［コマンドプロンプト］の順に進めて，コマンドプロンプトを起動する．
※Windows XPの場合

手順⑤　コマンドプロンプトの真っ黒な画面からUSBドライブの**番号**を指定する．
※USBドライブがHドライブの場合，H: [Enter]とキー入力する．

手順⑥　コマンドプロンプトの真っ黒な画面からUSBドライブの番号と同じ**アルファベット**を入力する．
※H [Enter]とキー入力する．(⑤の場合と異なり　『:』(コロン)が不要)

手順⑦　以後，本実習書に従って実習をおこなう．

★　この手順は大学のPCや家庭のPCなどでドライブなどの条件が異なることから，おこなっている実習の手法である．
同一のPCの場合でも，USBメモリを挿したときの条件で，ドライブ番号が変わる場合がある．

Windows 7の場合，LSI-C試食版を使用するかぎり，コマンドプロンプトでは日本語が使えない．実習書ではDOS用テキストエディタを使うことになっているが，代わりにWindows付属のメモ帳(　プログラムーアクセサリ　に存在)を使用する．
また，プログラム中で，画面に表示される部分などには日本語の文字は使用しない．
なお，Windows VISTAでは同様にグラフィック関数が使用できないので，本書の8章以降は実践できない．8章以降を実践したい場合はWindowsXP以前のMicrosoft OSで行うこと．

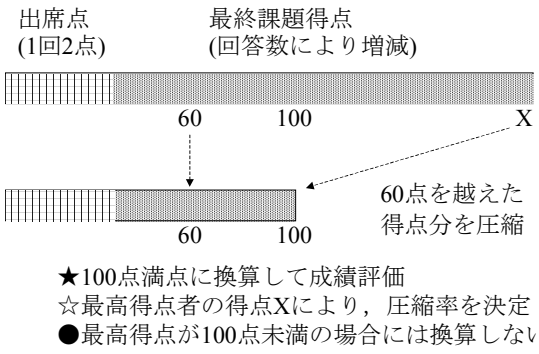
成績評価について

※東京医科歯科大学医学部保健衛生学科検査技術学専攻Only

本実習では出席点と最終課題による得点で評価を行う．

本実習書終章の後ろに記載した最終課題の内容に相当するプログラムを，本実習書に従ってC言語で作成し，指導教員による口答試問により理解度を確認する．具体的には①プログラムを実行して，最終課題に記載された内容を実現しているか．②ソースファイル内にC言語の文法上の誤りがないか．③そのアルゴリズムの内容およびそれを選択した理由を口答できるか．によって判断する．

口頭試問により合格した課題について，実習最終日までに1冊のレポートとして印刷し，提出することによって最終課題の得点が確定する．**レポートを提出しない限り得点とならない**ことに注意すること．



【注意点】

●実習書に記載されていない関数，表現などを用いても，文法上の誤りなどがなく，その意味を理解していると口頭試問で判断できれば問題なく評価する．

▲本書で使用しているコンパイラの作成された時期以降に，新たに定義されたC言語の規格を用いた場合，あるいは本書で使用しているコンパイラの作成された時期以前に定義された文法表記で，本コンパイラの作成された時期に定義から外された表現を用いた場合にはその旨を理解していなければならない．

◆プログラムソースのやり取り，もしくは相互相談によると考えられる内容であった場合，該当する最終課題の提出を棄却する．これは定期試験に際してカンニングが発見された場合に準ずると考えるからである．インターネットや成書に記載されている同一のプログラムを偶然参考にした場合でも，上記の可能性が示唆される場合には，同様に取り扱う．

C言語プログラムの作成の手順

準備段階

１．コンピュータの電源を入れてWindowsを起動する．
２．マウスポインタを画面左下にある［スタート］のところに動かして，マウスの左ボタンをクリックする．
３．カーソルを「プログラム(P)」→「アクセサリ」→「コマンド　プロンプト」の順に移動し，マウスの左ボタンをクリックする．(Windows98,Meなどの場合は，「プログラム(P)」→「MS-DOS　プロンプト」の順)
４．画面上に新しいWindowが開き，［C:¥> ］のような表示がでたらMS-DOSプロンプトである．ここでフロッピーディスクをコンピュータに入れる．キーボードを使い，【a:】もしくは【A:】と入力して〔Enter〕キーを押す．

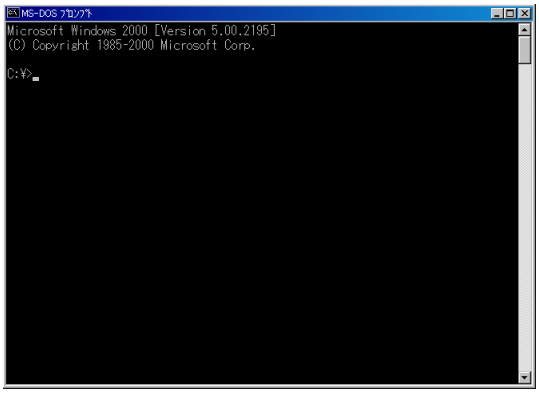


Fig.S-0 コマンドプロンプトの画面表示

５．表示が［A:¥>］のようになったのを確認したら，【c】もしくは【C】と入力して〔Enter〕キーを押す．これによってC.BATという名前のバッチファイルが実行され，C言語開発のための環境変数が設定される．
６．表示が［2004/04/６(THU)13:00:00 LSIC86 SYSTEM DISK A:¥>］のように変わったのを確認する．(表示はWindowsのバージョンによって若干異なる)
７．〔Alt〕+〔Enter〕キーでMS-DOSプロンプトのWindowが最大化されるので，場合によって切り換える．再度〔Alt〕+〔Enter〕で元の大きさに戻る．
　※グラフィック関数とPCを構成するハードウェアによって，Windowの大きさに動作が依存する場合があるので，状況に応じて切り換える．

ソースファイルの作成

８．【e】もしくは【E】と入力して〔Enter〕キーを押す．これによってE.BATという名前のバッチファイルが実行され，テキストエディタが起動する．
９．新規作成の場合は，まず〔F1〕キーを押してファイル名を変更できるようにする．A:¥*.*と表示されている部分の色が変わるので，「*.*」の部分をDeleteキーで削除し，残った「A:¥」に続けて，次ページの表を参考にプログラムの名前を半角８文字以内で決定して入力する．拡張子は必ず「.C」とする．以前作ったプログラムを修正する場合はカーソルを動かしてファイルを選択し，〔Enter〕キーを押す．

有効なプログラム名の例	無効なプログラム名の例
TEST.C	TE ST.C　　（間にスペースがある）
HASAMI6.C	HASAMI001.C　　（名前が８文字以上である）
EXAM-123.C	TMD*/.C　　（＊や／はファイル名に使用不可）

コンパイルの方法

10. エディタを終了する．NEEDVの場合，〔f･1〕キーを押してからカーソルキーでカーソルを動かして「ファイルのセーブと編集終了」を選択する．
11. 【CC ファイル名】〔Enter〕と入力する．「.C」はあってもなくても良い．CC.BATという名前のバッチファイルにより，LSI-C86のコンパイラであるLCC.EXEにソースファイルが渡され実行形式のファイルが作成される．

例）TEST.Cというソースファイルを作成した場合

【CC TEST】〔Enter〕
あるいは
【CC TEST.C】〔Enter〕
と入力する．

12. プログラムが正常に作成されていればコンパイルは正常に終了する．
→出てきたメッセージの中に「エラー」がなければ正常終了である．
→「警告」はプログラムとして不完全なところがあるが，動作可能なので気になった場合だけ直せば良い．

プログラムの実行

13. コンパイルが終了したら【ファイル名】〔Enter〕と入力する．「.C」はつけない．
Ｃ言語ではソースファイルTEST.Cをコンパイルすると実行ファイルTEST.EXEが作成される．
プログラムの実行のときはTEST.EXEを実行するので「.C」はつけない．

例）TEST.Cというソースファイルをコンパイルした場合

【TEST】〔Enter〕

サンプルプログラム

①万能カレンダー　　ファイル名：CALENDER.C

年号と月を入力すると自動的にカレンダーを作成・表示する．閏年対応．

計算ルール：
西暦1582年以降４で割り切れる年は閏年である．ただし100で割り切れる年は平年・400で割り切れる年は閏年である．年:year,月:month,日:dayが分かっているとして

1,2月はa1=year-1,a2=month+10
他の月はa1=year,a2=month-2
とおいて
a3=a1+a1/4-a1/100+a1/400+(13*a2-1)/5 +day
を計算し７で割った余り0,1,2,…が日，月，火…と対応する.

②Q極ピンボール　　ファイル名：PB.C

完全弾性衝突で７色のボールが画面上を跳ね回る．理論上は球体を無限に増やすことが可能である.

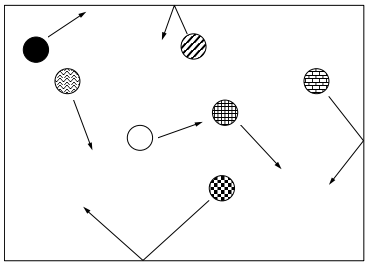


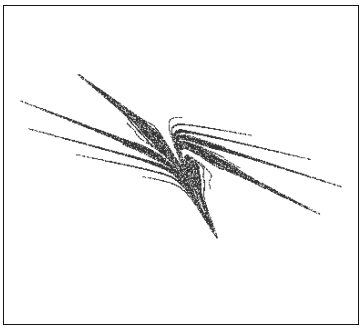
Fig.S-1 Pinball Action

③基本的ソート&データ　　ファイル名：SORT.C　　データ名：RANDOM.DAT

アルゴリズムに交換法を採用した場合の並べかえ.
交換法はもっとも入れ替えが少ない並べかえのアルゴリズムである.

④カオス・フラクタル　　ファイル名：CHAOS.C

規則的な方程式で記述できる系であっても，一見すると乱雑で不規則な挙動を示す場合がある．この挙動をカオスと呼ぶ．このカオスを図示したものがカオス・フラクタル図形である.
このプログラムでは実行するタイミングにより，Fig.S-3に示す2種類のいずれかが表示される.



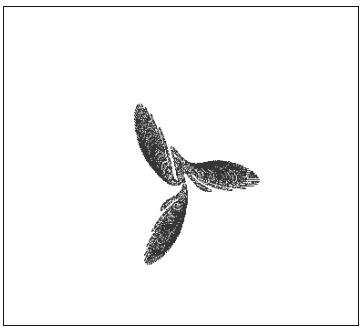


Fig.S-2 Chaos Fractal図形

⑤ライフゲームシミュレーション　　ファイル名：LIFE.C

人工生命と呼ばれるプログラムの中でもっとも古典的なものである.
ルール：自分を囲む8つのマス目に2ないし3の他者がいると生存であるが，0,1,4～8の場合死亡する．ただし，死亡した状態で周囲に3の他者がいると復活する.

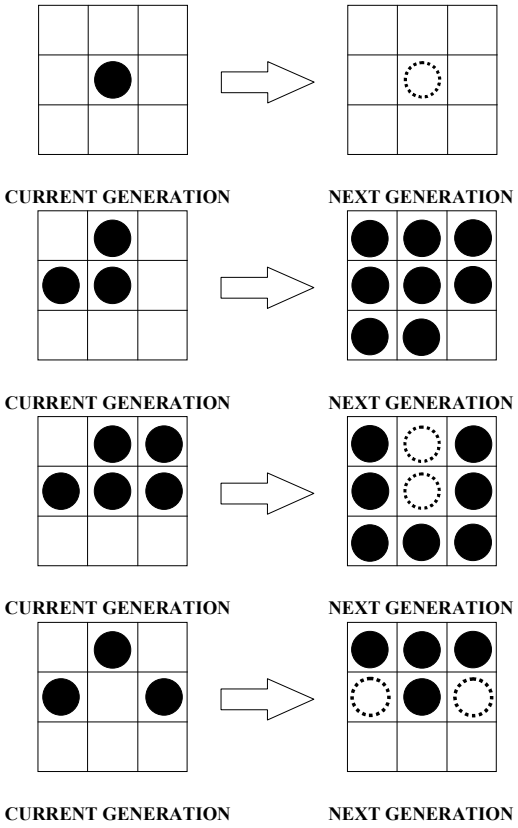


Fig.S-3 Algorhythm of Life game

0章　C言語のルール

まず，適当なエディタを準備して以下の例題の通りに入力してみることにする．

例題　　ファイル名：TEST.C

```
#include <stdio.h>

  main()
       {
       printf("Hello, world¥n");
       }
```

上の通り入力したら，ファイルをセーブするのを忘れないようにしてエディタを終了し，コンパイルしてみる．コンパイルの仕方は別紙の解説の通りである．

入力にミスがなく，コンパイルの仕方を間違えなければ実行形式のファイルTEST.EXEが出来ているはずである．次のように入力して確認する．

```
dir[Enter]
```

画面上に出てきた文字の羅列の中に，TEST.EXEが存在したらＯＫである．もし，何らかのエラーが出たり，TEST.EXEが存在しなかったらエディタを使ってTEST.Cの内容を確認すること．どこかが間違っている可能性大である．

ＯＫならば，続いて次のように入力する．

```
TEST[Enter]
```

このとき，画面上に

```
Hello, world
```

と出てきたらプログラムは完成である．

ここで入力してもらったTEST.Cをソースファイル（source file)といい，出来上がったTEST.EXEを実行ファイル(EXEcute file)という．Ｃ言語で作る実行ファイルは，MS-DOSの外部コマンドと呼ばれるものと同義である．

MS-DOSは命令語，すなわちコマンドをメモリの節約のために，必要最小限の内部コマンドと，その他あると便利な外部コマンドに分けている．これを利用してコンパイル型のプログラム言語を用いて自分にとって便利な外部コマンドを作成することがプログラムの基本である．

それでは順を追って見ていく．

```
#include <stdio.h>  ·····①

  main()            ·····②
```

```
{
printf("Hello, world¥n");  ·····························③
}
```

①これは標準入出力関数用ヘッダファイル(STanDard Input Output Header file)を含む(INCLUDE)という意味である．この実習中に学習するほとんどの関数はこれだけで十分なので，慣れるまではＣ言語のプログラムを作成するときの御約束と思っていて良い．

②Ｃ言語では関数という単位でプログラムを作成する．その中でもmain関数は特別な意味があり，どのような順番で記述されたプログラムでも必ずmain関数から実行するという規則がある．したがって，Ｃ言語のプログラムを作成するときには，絶対にmain関数を作らなければならない．

また，関数を作成するときは必ず{}でくくらなければならない．

③printfは文字や数字を表示する関数で，１章で学習する．このような基本的な関数は，Ｃ言語のコンパイラメーカーが標準で提供している．

また，Ｃ言語では空白(space)やタブ(TAB)は，プログラムを読みやすくするために，比較的自由に入れられることになっている．ただし，それによって本来一つであるべき関数名が分断されたりしてはならない．

さらに，文の区切りを表すために，必ず区切りごとに´;´をつけることになっている．

１章　表示する

Ｃ言語でコンピュータに何かを実行させてもその結果が目に見えなければ，何を処理しているのか人間には分からない．そこで，まず画面に表示させる関数を学習する．

1.1　printf

TEST.Cで簡単に説明したが，printfは文字や数字を表示する関数である．書式は以下の通りである．

```
printf ("書式",オブジェクトの並び);
```

printfの書式には以下のものを含むことが出来る．

Ⅰ．文字定数
Ⅱ．変換指示記号

変換指示記号については後述する．エディタを起動して次の２つのプログラムを入力し，実行結果を比較してみる．

ファイル名：EX11.C

```
#include <stdio.h>

main()
      {
       int a,b,c,d,e,f;

       a=5;
       b=10;
       c=a+b;
       d=a-b;
       e=a*b;
       f=a/b;

       printf("a=%d     ",a);
       printf("b=%d     ",b);
       printf("a+b=%d ¥n",c);
       printf("a-b=%d   ",d);
       printf("a*b=%d ¥n",e);
       printf("a/b=%d   ",f);

      }
```

【実行結果】

```
a=5      b=10      a+b=15
a-b=-5    a*b=50
a/b=0
```

ファイル名：EX12.C

```
#include <stdio.h>

main()
      {
       float a,b,c,d,e,f;

       a=5;
       b=10;
       c=a+b;
       d=a-b;
       e=a*b;
       f=a/b;

       printf("a=%f     ",a);
       printf("b=%f     ",b);
       printf("a+b=%f ¥n",c);
       printf("a-b=%f   ",d);
       printf("a*b=%f ¥n",e);
       printf("a/b=%f   ",f);

      }
```

【実行結果】

```
a=5.000000      b=10.000000      a+b=15.000000
a-b=-5.000000    a*b=50.000000
a/b=0.500000
```

自分で入力した結果と実行結果を比較し，異なっている場合はプログラムをチェックして，結果が一致するまで試行してみることが必要である．

ここでＣ言語では一般の数学と若干異なる演算記号を使うので以下にまとめておく．

Table1.1　C言語の演算記号の意味

＋	加算
－	減算
＊	乗算
／	除算
％	剰余

演算の意味を理解したところで，結果を比較してみる．すると，EX11.Cの方では除算の結果が明らかに間違っている．しかし，これはC言語では正しい結果なのである．これはデータの型に由来している．

詳しくは次項に記すが，intというのは，変数を整数型として定義する命令であり，floatというのは変数を浮動小数点型として定義する命令である．またprintf関数内の%dは整数型の変数もしくは定数を整数として表示するための変換指示記号であり，%fは浮動小数点型の変数もしくは定数を浮動小数として表示するための変換指示記号である．

EX11.CとEX12.Cは整数と浮動小数の違いのみであるので，EX11.Cについて順を追って説明する．

```
#include <stdio.h>     ·····①

main()                 ·····②
      {
       int a,b,c,d,e,f;      ·····③

       a=5;                  ·····④
       b=10;                 ·····④
       c=a+b;                ·····⑤
       d=a-b;                ·····⑤
       e=a*b;                ·····⑤
       f=a/b;                ·····⑤

       printf("a=%d     ",a);       ·····⑥
       printf("b=%d     ",b);       ·····⑥
       printf("a+b=%d ¥n",c);       ·····⑦
       printf("a-b=%d   ",d);       ·····⑥
       printf("a*b=%d ¥n",e);       ·····⑦
       printf("a/b=%d   ",f);       ·····⑥

      }
```

①C言語の御約束である．

②実行を開始する関数.

③整数型で変数a,b,c,d,e,fを定義する．

④C言語では＝の記号は代入演算子と呼ばれており，数学の等号とは意味が異なる．

⑤演算の結果をそれぞれの変数に代入している．

⑥printfの書式では文字変数と変換指示記号を混在させることが出来る．変換指示記号のところに変数の内容が代入されて表示される．

⑦￥の記号はエスケープシーケンスと呼ばれる特殊な記号を表している．詳しくは1.3の項で述べるが，¥nが書式に含まれていると，表示は改行されることだけ覚えておく．

1.2 変換指示記号

ここまで述べてきたようにC言語では変数を使う場合，変数型を指定しなければならない．また，これを表示する場合，変数型に対応した変換指示記号を使用しなければ正しい結果は得られない．変数の型と変換指示記号を以下の表にまとめておく．

Table 1.2 変数型と変換指示記号の対応

整数型	int	%d
倍精度整数型	long	%ld
整数型八進数		%o
整数型十六進数		%x
浮動小数型	float	%f
倍精度浮動小数型	double	%lf
文字型	char	%c
文字列		%s

これらの変数型の前にsigned，unsignedをあえて記述する場合がある．signedはそれぞれの変数型で正負の数を処理する場合で，特に記述していなければこれに該当する．一方，unsignedは負の記号を使用しない場合に記述する．この場合，それぞれの変数で取り扱える数値の範囲は変わらないが，正の最大値が2倍になる．つまり，LSI-C86の場合，int型では-32768～32767の範囲の整数を取り扱うが，unsigned int型では0～65536の範囲の整数を取り扱うようになる．

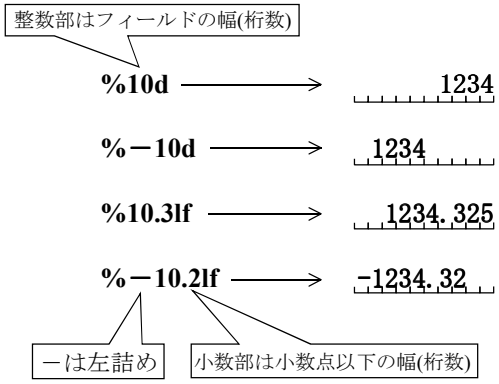


Fig.1.1 フィールド幅の指定

また，変換指示記号に，例えば%10dや%-10.2lfのように，フィールド幅を表す数値をつけることができる．数値の整数部がフィールドの桁数を表し，小数部が小数部の表示桁を表わす．数値が正であればフィールドに対して右詰めで表示し，負であれば左詰めで表示する．%10.5lfであれば，正負を表わす記号(正の場合は省略されるが桁数は確保)が1桁，整数部が3桁，小数点が1桁，小数部が5桁でトータル10桁になる．なお，本書で取り扱う例題では，全てフィールド幅を指定していない．必要と感じた場合には各自の判断で指定すると良い．

それでは，具体的なプログラムを元に解説する．

ファイル名:EX13.C

```
#include <stdio.h>

main()
      {
       char c;

       c='A';

       printf("%c",c);

      }
```

このプログラムを実行すると画面上にAという文字が表示される．これは文字変数cにAという文字を代入し，文字として表示させているからである．次にこのプログラムを一部だけ修正してEX14.Cを作成する．

ファイル名:EX14.C

```
#include <stdio.h>

main()
      {
       char c;

       c='A';

       printf("%d",c);

      }
```

このプログラムを実行すると画面上に65という数字が出る．EX13.Cとの違いは表示するときの変換指示記号だけである．

Ｃ言語では文字をASCIIコードもしくはJISコードで処理している．つまり，アルファベットや数字・記号などすべての文字には一対一で対応する数字が割り当てられており，文字Aに対応する数字が65である．これを変換指示記号%cで表示すればAと表示され，%dで表示すれば65と表示される．

つまり，char型とint型は基本的に整数を扱うという点で等価である．しかし，int型では0～65535まで扱えるのに対してASCIIコードは0～127，JISコードでも0～255までしか存在しないのでchar型ではそれ以上の変数を扱うことができない．

プログラムEX15.CおよびEX16.Cではどのように表示されるか，各自で考えて確認すると良い．

ファイル名:EX15.C

```
#include <stdio.h>

main()
      {
       char c;

       c='A';

       printf("%x",c);

      }
```

ファイル名:EX16.C

```
#include <stdio.h>

main()
      {
       char c;

       c='A';

       printf("%o",c);

      }
```

次のプログラムを入力して，実行し結果を確認する．

ファイル名:EX17.C

```
#include <stdio.h>

main()
      {
       char c[]="C Language";

       printf("%s",c);

      }
```

画面上にはC Languageと表示される．ここでc[]は配列変数と呼ぶ．詳しくは4.1節で解説する．

Ｃ言語では１つの変数で１つの文字を処理している．したがって，複数の文字で構成される単語を取り扱うためには，その単語の文字数分の変数が必要になる．

しかし，一々その文字ごとに変数を定義していては手間がかかり過ぎる．そこで変数に添え字をつけて異なる変数として扱い，表示するときに変換指示記号%sを用いて一括表示することが出来る．

文字変数に一文字だけ代入する場合には‘’（シングルクォーテーション）で文字を囲んだが，配列変数に文字列を代入するときは“”（ダブルクォーテーション）で囲む．

これは間違いやすいので注意しなければならない．

1.3　エスケープシークエンス

改行を行なう¥nのように￥に続けて記述されているものをエスケープシークエンスと呼ぶ．MS-DOSの流れを汲むWINDOWS9x系の場合，これを使用することで改行したり，文字に色をつけたりすることが可能になる．

以下に代表的なエスケープシークエンスをまとめておく．

Table 1.3　代表的なエスケープシークエンス

¥a	ベルを鳴らす
¥n	改行する
¥t	タブ機能を使う
¥¥	￥を表示する
¥'	´を表示する
¥"	”を表示する
¥?	？を表示する

1章の課題

●自分の肩書・氏名・住所・TEL・FAX・E-mail adress・性別・年齢などを名刺風に表示すること.

2章　入力する

身長と体重からBMI（Body Mass Index）を計算するとき，あらかじめ身長と体重が分かっていれば，１章で述べたような手法でプログラム中に組み込んでおくことが出来る．しかし，わざわざプログラムを作成して１回だけしか計算しないのではプログラム作成に時間を費やす分だけむしろ労力を必要とする．

身長と体重を後から入力することが出来れば，１回作成したプログラムで500人分のBMIを計算するのに使用することが出来る．

この章では，プログラムが実行された後で，数字や文字を入力する方法を学習する．

2.1　scanf

エディタを使用して次のプログラムを入力し，実行する．

ファイル名：EX21.C

```
#include <stdio.h>


main()
      {
       int a,b,c;

       printf ("a=");
       scanf ("%d",&a);
       printf ("b=");
       scanf ("%d",&b);

       c=a+b;

       printf ("%d+%d=%d",a,b,c);

      }
```

このプログラムを実行すると画面にａ＝と表示され，カーソルが点滅する状態になる．ここで，任意の整数を入力して[Enter]とする．

続いて画面にｂ＝と表示され，カーソルが点滅する状態になる．ここでも，任意の整数を入力して[Enter]とする．

すると画面上にそれら２つの数字を加え合わせた数字が数式として表示される．

scanfはキーボードから数字・文字を入力する関数である．書式は以下の通りである．

```
scanf ("変換指示記号",＆変数)
```

この関数でよくある間違いとして，**＆の忘れ**がある．

つづいて以下のプログラムを入力し，実行する．

ファイル名：EX22.C

```
#include <stdio.h>

main()
      {
       float a,b,c;

       printf ("a=");
       scanf ("%f",&a);
       printf ("b=");
       scanf ("%f",&b);

       c=a+b;

       printf ("%f+%f=%f",a,b,c);

      }
```

これはEX21.Cの変数と変換指示記号を浮動小数点型に変更したプログラムである．
結果が浮動小数点型になっていることを確認する．

ファイル名：EX23.C

```
#include <stdio.h>

main()
      {
       char c;

       printf ("Character:");
       scanf ("%c",&c);

       printf ("ASCII Code of Character¥'%c¥' is No.%d",c,c);

      }
```

このプログラムでは何文字文字を入力しても，最初の文字を一文字だけ入力し，処理している．文字として入力した変数を変換指示記号によって，文字と数字のいずれでも

表示することを確認する.

ファイル名：EX24.C

```
#include <stdio.h>


main()
      {
       char c[20];
       int a;

       printf ("Name:");
       scanf ("%s",&c);
       printf ("Age:");
       scanf ("%d",&a);

       printf ("%s is %d years old.",c,a);

      }
```

このプログラムでは配列変数を使用することで，複数の文字の入力が可能になっている．入力出来る最大の文字数は，最初の変数定義のc[20]の添え字で決定される．たとえば以下のように入力された文字列の1文字目，2文字目…がそれぞれ1文字ずつc[0],c[1],…に入力されていることを理解しておく.

I	N	F	O	R	M	A	T	I	O	N
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]	c[10]

2.2　getch

まず，次のプログラムを入力し，EX24.Cとの動作を比較してみる.

ファイル名：EX25.C

```
#include <stdio.h>


main()
      {
       char c[20];
       int a;
```

```
       printf ("Name:");
       scanf ("%s",&c);
       printf ("Age:");
       scanf ("%d",&a);

       printf ("OK! PUSH KEY!!!¥n");
       getch();

       printf ("%s is %d years old.",c,a);

      }
```

このプログラムを実行すると名前および年齢入力後，画面上にOK! PUSH KEY!!!と表示されて一旦プログラムが停止するはずである．ここで適当なキーを一回押すとEX24.Cと同様に進行する．getch関数は本来以下のように使用する．

```
変数＝getch();
```

この関数は一文字入力関数である．scanfのように多数の文字を入力することは出来ないが，変換指示記号を記述する必要がないというメリットがあり，また入力後[Enter]キーを押さなくても進行するという利点がある．
ここで，次のプログラムを入力し実行してみる．

ファイル名：EX26.C

```
#include <stdio.h>


main()
      {
       char c;

       printf ("PUSH ANY KEY!¥n");
       c=getch();

       printf ("ASCII Code of Character ¥' %c ¥' is %d.",c,c);

      }
```

このプログラムを実行すれば，キー入力後直ちに次の処理へ進行していることが理解出来るはずである．

２章の課題

●入力された数字のアスキーコードに対応する文字を表示すること.

◆入力された小文字のアルファベットを大文字のアルファベットに変換して表示すること.

★入力した数字をnとした場合の，n(1～26)番目のアルファベット(A～Z)を表示するプログラムを作成すること.

3章　条件分岐

人間の生活の仕方には様々なものがあるが，これと決めたらひたすらそれのみに邁進する「猪突猛進型」と，その時々の状況に応じて行動を変化させる「臨機応変型」に大別出来る．ほとんどの人間は，天候・予算・人間関係…etcの各種の制限により時々刻々対応を変化させている．

コンピュータで処理を行なうときも同様で，常に一定の処理を行なうことは少ない．むしろ，入力された情報に応じて処理を選択し実行していく方が，ある程度までは変動に対応出来るという点で優れている．

ここでは，条件分岐として代表的な２種類の手法を取り上げる．

3.1　if～else

まず，以下のプログラムを入力し，実行してみることにする．

ファイル名：EX31.C

```
#include <stdio.h>


main()
      {
       int a,b;

       printf ("a=");
       scanf ("%d",&a);
       printf ("b=");
       scanf ("%d",&b);

           if (a<b)
                   {
                    printf ("a<b");
                   }
      else if (a>b)
                   {
                    printf ("a>b");
                   }
      else if (a==b)
                   {
                    printf ("a=b");
                   }

      }
```

このプログラムを実行して，aとbにそれぞれ適当に数を入力する．するとaとbの大小関係を判断して，その関係が画面に表示される．

このように，if…elseは条件式を判定し，それが成立するときに実行式を実行する条件分岐である．書式は以下のようになる．

```
      if (条件式1)
                {
                 実行式1;
                }
else if (条件式2)
                {
                 実行式2;
                }
else if (条件式3)
                {
                 実行式3;
                }
                .
                .
                .
                .
```

条件によって細かく分類する場合，条件式は幾つ並べてもよい．また，実行式は１つだけではなく，幾つ並べてもよい．

条件式は大小の比較で表すが，数学的記述と若干異なるので注意する．特にＣ言語では＝が代入の意味で使用されるので，比較の場合＝＝になっていることに注意する．

Table2.1　条件式と数学的表現の対応

条件式	数学的意味
==	＝
>	＞
<	＜
!=	≠
>=	≧
<=	≦

ここで以下のプログラムを入力して実行してみることにする．

ファイル名：EX32.C

```
#include <stdio.h>


main()
      {
       int a,b,c;
```

```
        printf ("a=");
        scanf ("%d",&a);
        printf ("b=");
        scanf ("%d",&b);

            if (a<b)
                    {
                     c=b-a;
                     printf ("b-a=%d",c);
                    }
       else if (a>b)
                    {
                     c=a-b;
                     printf ("a-b=%d",c);
                    }
       else if (a==b)
                    {
                     c=a-b;
                     printf ("a-b=%d",c);
                    }

       }
```

これは，入力された２つの数の差を計算するプログラムである．入力した数字の順番にかかわらず，差の絶対値が表示されるように計算の方法と表示の仕方が選択されることを確認する．

ファイル名：EX33.C

```
#include <stdio.h>


main()
       {
        int a,b;

        printf ("sex:(male=1 female=2) ");
        scanf ("%d",&a);
        printf ("age:");
        scanf ("%d",&b);

              if (b<12)
                       {
                        printf ("You are only a child.");
```

```
                }
        else if (b<30)
                {
                        if (a==1)
                                {
                                 printf ("You are a gentleman.");
                                }
                    else if (a==2)
                                {
                                 printf ("You are a lady.");
                                }
                    else
                                {
                                 printf ("You are a new-half.");
                                }
                }
        else if (b>=30)
                {
                        if (a==1)
                                {
                                 printf ("You are an old man.");
                                }
                    else if (a==2)
                                {
                                 printf ("You are an old woman");
                                }
                    else
                                {
                                 printf ("You are a new-half.");
                                }
                }

    }
```

これは入力された性別と年齢から質問に対する回答者を階級化するプログラムである．年齢を12才，30才で区切ってあるのは単純化するためである．了承されたい．

このプログラムのようにifの実行文の中に，さらにif（条件分岐）が入るのを入れ子（ネスティングス）という．２つ以上の条件を重ねるときに使用される．

ところでEX33.Cと同様の処理を行なうのに，以下のEX34.Cのような記述も出来る．入力して，完全に同じであることを確認する．

ファイル名：EX34.C

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

```
main()
      {
       int a,b;

       printf ("sex:(male=1 female=2) ");
       scanf ("%d",&a);
       printf ("age:");
       scanf ("%d",&b);

             if (b<12)
                      {
                       printf ("You are only a child.");
                      }
        else if (b<30 && a==1)
                               {
                                printf ("You are a gentleman.");
                               }
        else if (b<30 && a==2)
                               {
                                printf ("You are a lady.");
                               }
        else if (b>=30 && a==1)
                               {
                                printf ("You are an old man.");
                               }
        else if (b>=30 && a==2)
                               {
                                printf ("You are an old woman");
                               }
        else
                              {
                               printf ("You are a new-half.");
                              }

     }
```

このように条件式を集合演算の∩(AND)と∪(OR)を組み合わせて，複数重ねることも出来る．ANDとORは以下のように記述する．

Table3.1　条件式の結合と集合演算記号の対応

記述	意味	記号
&&	AND	∩
||	OR	∪

２つ以上の条件を重ねた判定を行なうとき，ネスティングスを重ねるか，集合演算を用いるかはプログラムする者の自由である．実行されるプログラムが同じ結果を出すことが重要であり，記述の仕方に制限はない．

プログラムを行なうときに重要な事は，プログラムの実行速度，プログラムの汎用性，プログラムの読みやすさである．この中で，プログラムの読みやすさが近年では最も重要視されている．これは，結局，読みやすいプログラムは記述しやすく，また，将来の改変が容易であるなどの理由による．

プログラムを読みやすくするために，Ｃ言語では注釈を入れることが出来る．書式は以下の通りである．

```
/*  注釈文  */
```

このように，/*と*/で挟まれた間に記述されたことは全てコンパイル時に注釈として解釈され，実行されないようになる．本書で掲載しているプログラムは，解説を本文内で行なうことを前提としているので注釈文はつけていないが，将来，過去のプログラムを取り出して改良後に，再利用する場合のことを考慮すると，可能な限りつけるようにするべきである．

ところで，コンピュータでは，数字の大小だけを比較出来る訳ではない．文字にも大小がある．次のプログラムで考えていくことにする．

ファイル名：EX35.C

```
#include <stdio.h>


main()
      {
       unsigned char c1,c2;

       printf ("Character (1):");
       c1=getch();
       printf ("%c¥n",c1);

       printf ("Character (2):");
       c2=getch();
       printf ("%c¥n",c2);

            if (c1<c2)
                      {
                       printf ("¥'%c¥' is smaller than ¥'%c¥'.",c1,c2);
                      }
       else if (c1==c2)
                      {
                       printf ("¥'%c¥' is equal to ¥'%c¥'.",c1,c2);
```

```
                    }
        else if (c1>c2)
                    {
                     printf ("¥'%c¥' is larger than ¥'%c¥'.",c1,c2);
                    }

      }
```

プログラムを実行して，適当な文字を２つ順番に入力するとその大小を比較し，その結果に応じた回答が出力されるはずである．

ところで，本プログラムで変数型を表わすcharの前にunsignedを記述している．char型の場合，unsignedを記述すると扱える変数の範囲が0～255までのASCIIコードとなるので，日本語用MS-DOSで扱える1byteの文字全てが扱えるようになる．これを記述しない場合，0～127までのASCIIコードを扱うので，日本語特有の半角カナなどが処理できない．

コンピュータで文字を扱うために，ASCIIコードが定義されていることはすでに述べた．文字の大小を比較するということは，文字に割り当てられたASCIIコードの大小を比較することに相当する．これを応用すると次のようなプログラムが考えられる．

ファイル名：EX36.C

```
#include <stdio.h>


main()
       {
        unsigned char c1[30],c2[30];

        printf ("Name (1):");
        scanf ("%s",&c1);

        printf ("Name (2):");
        scanf ("%s",&c2);

             if (c1[0]<c2[0])
                      {
                       printf ("¥'%s¥' is smaller than ¥'%s¥'.",c1,c2);
                      }
        else if (c1[0]==c2[0])
                       {
                        printf ("¥'%s¥' is equal to ¥'%s¥'.",c1,c2);
                       }
        else if (c1[0]>c2[0])
                       {
```

```
                    printf ("¥'%s¥' is larger than ¥'%s¥'.",c1,c2);
                  }

    }
```

これは入力された２つの名前の頭文字の大小を比較している．このように比較して小さい方から順番に並べることで，例えば電話帳などのようなデータベースを整理して処理することが出来る．

ファイル名：EX37.C

```
#include <stdio.h>

main ()
      {
       int a,b,c;
       int x,y,z;

       printf ("a=");
       scanf ("%d",&a);
       printf ("b=");
       scanf ("%d",&b);
       printf ("c=");
       scanf ("%d",&c);

            if (a>b)
                  {
                        if (c>a)
                                {
                                 x=c;
                                 y=a;
                                 z=b;
                                }
                  else if (c>b)
                                {
                                 x=a;
                                 y=c;
                                 z=b;
                                }
                  else
                                {
                                 x=a;
                                 y=b;
                                 z=c;
```

```
                                }
            }
    else    {
                    if (c>b)
                                {
                                 x=c;
                                 y=b;
                                 z=a;
                                }
               else if (c>a)
                                {
                                 x=b;
                                 y=c;
                                 z=a;
                                }
               else
                                {
                                 x=b;
                                 y=a;
                                 z=c;
                                }

            }

     printf ("%d   %d   %d",x,y,z);

}
```

これは，3つの整数を入力すると大きい順に出力するプログラムである．このように並べ替えをするときには，一旦別の場所に格納するのが便利である．このようにして大小の判定を繰り返し，データベースを整理することが可能になる．

3.2 switch～case

条件分岐に際して，if～elseを用いることは前節で述べた．しかし，条件の取り方によってはネスティングスを多重に配置したり，集合演算を多用したりしなければならず，時として記述が複雑化する．さらに，if～elseでは条件判定を全て行なうので処理能力が低下するなどの問題点もある．

このように条件によって分岐する先が多数存在するような場合，条件ごとに行なうべき処理のみを行なう方が，処理の高速化につながり，かつプログラムが読みやすくなる．

詳しくは，次のプログラムの実行後に解説する．

ファイル名：EX38.C

```
#include <stdio.h>


main()
      {
       int a,b,c,d;

       printf ("1:Addition  2:Subtraction  3:Multiplication  4:Division¥n");
       printf ("Select Number: ");
       scanf ("%d",&d);
       printf ("a=");
       scanf ("%d",&a);
       printf ("b=");
       scanf ("%d",&b);

             switch (d)
                       {
                        case 1:
                               c=a+b;
                               printf ("%d+%d=%d¥n",a,b,c);
                               break;
                        case 2:
                               c=a-b;
                               printf ("%d-%d=%d¥n",a,b,c);
                               break;
                        case 3:
                               c=a*b;
                               printf ("%d*%d=%d¥n",a,b,c);
                               break;
                        case 4:
                               c=a/b;
                               printf ("%d/%d=%d¥n",a,b,c);
                               break;
                       default:
                               break;
                       }

      }
```

このプログラムは加算・減算・積算・除算をキーボードから入力し，続いて２つの整数を入力すると選択した演算を行い，演算結果を表示するプログラムである．

ここで条件分岐に使われているのがswitch～caseである．書式は以下のようになる．

```
switch (変数)
            {
             case 1:
                    実行文;
                    break;
             case 2:
                    実行文;
                    break;
             case 3:
                    実行文;
                    break;
             default:
                    実行文;
                    break;
            }
```

case　の後の数字は変数の内容と条件によって変化する．変数が70～73まで変わるとするならば，それぞれ70,71,72,73とすれば良い．また，変数がchar型である場合，'文字'として記述するか，その文字のASCIIコードで指定してもよい．

流れとしては以下のようになる．

switchによって変数の値に相当する条件を示すcaseのところに分岐し，以下のbreak;が記述されているところまで書かれているすべての実行文を実行し，その後，他の実行文は一切行なわずに{}の外へ出る．もし，相当する条件がなければdefault:のところにある実行文を実行する．

ここで，注意しなければならないのはcase文とcase文の間にある実行文を実行する訳ではなく，あくまでもcase文からbreak;までを実行するということである．もしcase文とcase文の間にbreak;がなければ，続けて次のcaseの実行文も実行してしまう．

次のプログラム EX39.Cによってこれを確認する．EX39.CはEX38.Cのbreak;を一部除いただけのプログラムなので，EX38.Cをコピーして修正すれば簡単に入力が終了する．

ファイル名：EX39.C

```
#include <stdio.h>


main()
      {
       int a,b,c,d;

       printf ("1:Addition  2:Subtraction  3:Multiplication  4:Division¥n");
       printf ("Select Number: ");
       scanf ("%d",&d);
       printf ("a=");
       scanf ("%d",&a);
       printf ("b=");
       scanf ("%d",&b);
```

```
        switch (d)
                {
                 case 1:
                        c=a+b;
                        printf ("%d+%d=%d¥n",a,b,c);
                 case 2:
                        c=a-b;
                        printf ("%d-%d=%d¥n",a,b,c);
                 case 3:
                        c=a*b;
                        printf ("%d*%d=%d¥n",a,b,c);
                 case 4:
                        c=a/b;
                        printf ("%d/%d=%d¥n",a,b,c);
                default:
                        break;
                }

    }
```

このように，EX38.Cと同様の結果を出すのは，分岐の最後に配置された除算のみである．これは，その後に実行するものが記述されていないからであり，本来はdefault文の後に実行文が存在すれば実行されているはずである．

ところで，条件は異なるが同一の処理を行ないたい場合はどのようにするかを考える．if～elseの場合には集合演算の||を用いて記述することが可能であった．しかし，switch～caseの場合はもっと簡単で，case文を複数行並べることで容易に行なえる．

また，case文の後の条件も順不同であり，数字の小さい順番であるとか，アルファベット順であるということは全くない．プログラマーの意思で自由に決定出来る．

これを次のプログラムで確認する．

ファイル名：EX3A.C

```
#include <stdio.h>


main()
      {
       int a,b;
       char c;

       printf ("M:Meiji  T:Taisho  S:Showa  H:Heisei¥n");
       printf ("Select Code: ");
       c=getch();
       printf ("%c¥n",c);
```

```
        printf ("year:");
        scanf ("%d",&a);

              switch (c)
                        {
                         case 'm':
                         case 'M':
                                  b=a+1867;
                                  printf ("M.%d is equal to A.D.%d.¥n",a,b);
                                  break;
                         case 't':
                         case 'T':
                                  b=a+1911;
                                  printf ("T.%d is equal to A.D.%d.¥n",a,b);
                                  break;
                         case 's':
                         case 'S':
                                  b=a+1925;
                                  printf ("S.%d is equal to A.D.%d.¥n",a,b);
                                  break;
                         case 'h':
                         case 'H':
                                  b=a+1988;
                                  printf ("H.%d is equal to A.D.%d.¥n",a,b);
                                  break;
                        default:
                                break;
                        }

      }
```

これは元号を選択し，年数を入力すると西暦に変換するプログラムである．
キー入力小文字の場合も，CAPS LOCKされて大文字で入力された場合もどちらも同様の処理を行なうのが特徴である．

３章の課題

★if～elseとswitch～caseの利点と欠点を挙げ，それらを比較出来るプログラムを作成すること．
●試験の点数を入力して，それに対する評価を優・良・可・不可で表示するプログラムを作成すること．

４章　繰り返しの処理

前章で条件分岐を学んだことで，プログラムのバリエーションが拡大した．しかし，まだ十分とは言えない．本来，プログラムとは同一の処理を繰り返して実行するときにその真価を発揮するものであり，あるいはプログラム言語というものはそれを前提として作成されている．

この章で繰り返し処理を学習することで，プログラムがそれらしくなるはずである．

4.1 配列変数

配列とは，例えば遺伝子配列と言う言葉から連想されるように，同一のカテゴリーのものが複数並んだ状態を考えれば良い．すなわち配列変数とは，同一の変数が複数並んだ状態と言い換えても良いだろう．

この章までに，「とりあえず」という条件付きではあるが，すでに配列変数を使用している．まず，配列変数の書式を以下に示す．

```
変数名［変数の個数］
```

これだけでは理解しにくいので以下にプログラムを示す．入力して実行してみる．

ファイル名：EX41.C

```
#include <stdio.h>

main()
      {
       int a[6];

       a[0]=10;
       a[1]=3;
       a[2]=a[0]+a[1];
       a[3]=a[0]-a[1];
       a[4]=a[0]*a[1];
       a[5]=a[0]/a[1];

       printf("a[0]=%d¥n",a[0]);
       printf("a[1]=%d¥n",a[1]);
       printf("a[0]+a[1]=%d¥n",a[2]);
       printf("a[0]-a[1]=%d¥n",a[3]);
       printf("a[0]*a[1]=%d¥n",a[4]);
       printf("a[0]/a[1]=%d¥n",a[5]);

      }
```

配列変数は変数の順番を示す［添え字］がついている以外の点では，通常の変数と同

じように処理することが出来る．プログラムを順を追って以下に解説する．

```
#include <stdio.h>                          ·····①

main()                                      ·····②
      {
       int a[6];                            ·····③

       a[0]=10;                             ·····④
       a[1]=3;                              ·····④
       a[2]=a[0]+a[1];                      ·····④
       a[3]=a[0]-a[1];                      ·····④
       a[4]=a[0]*a[1];                      ·····④
       a[5]=a[0]/a[1];                      ·····④

       printf("a[0]=%d¥n",a[0]);            ·····⑤
       printf("a[1]=%d¥n",a[1]);            ·····⑤
       printf("a[0]+a[1]=%d¥n",a[2]);       ·····⑤
       printf("a[0]-a[1]=%d¥n",a[3]);       ·····⑤
       printf("a[0]*a[1]=%d¥n",a[4]);       ·····⑤
       printf("a[0]/a[1]=%d¥n",a[5]);       ·····⑤

      }
```

①Ｃ言語の御約束である．

②Ｃ言語はmain関数から開始する．

③配列の宣言である．この場合，int型でaという名前の添字付変数（配列変数）を６つ用意している．

④コンピュータでは数字は0から始まる．すなわち，配列を6つ宣言した場合，添字は0～5の6つの数字が使用可である．

⑤ここでの配列変数はint型で宣言されているので，変換指示記号%dを使用している．

次に文字変数の配列について考えてみる．以下のプログラムを入力して実行する．

ファイル名：EX42.C

```
#include <stdio.h>

main()
      {
```

```
        char c[10];

        printf ("Name(MAX:10)  ");
        scanf ("%s",c);

        printf ("%c¥n",c[0]);
        printf ("%c¥n",c[1]);
        printf ("%c¥n",c[2]);
        printf ("%c¥n",c[3]);
        printf ("%c¥n",c[4]);
        printf ("%c¥n",c[5]);
        printf ("%c¥n",c[6]);
        printf ("%c¥n",c[7]);
        printf ("%c¥n",c[8]);
        printf ("%c¥n",c[9]);

      }
```

ここで，プログラムを実行すると横書きで入力した名前が，縦書きになる．もし，10文字以下で入力した場合，下の方に入力した覚えのない文字が出ることがある．これは，配列の初期化を省略したことによるゴミデータであるので無視して良い．

このプログラムでは入力の都合により配列を10個までしか宣言していない．配列変数として宣言出来る個数は使用しているコンピュータの設定によっても異なるが，通常1000～10000を目安にすると良い．

以下に詳しく解説する．

```
#include <stdio.h>                          ·····①

main()                                      ·····②
      {
        char c[10];                         ·····③

        printf ("Name(MAX:10)  ");          ·····④
        scanf ("%s",c);                     ·····⑤

        printf ("%c¥n",c[0]);               ·····⑥
        printf ("%c¥n",c[1]);               ·····⑥
        printf ("%c¥n",c[2]);               ·····⑥
        printf ("%c¥n",c[3]);               ·····⑥
        printf ("%c¥n",c[4]);               ·····⑥
        printf ("%c¥n",c[5]);               ·····⑥
        printf ("%c¥n",c[6]);               ·····⑥
        printf ("%c¥n",c[7]);               ·····⑥
        printf ("%c¥n",c[8]);               ·····⑥
```

```
       printf ("%c¥n",c[9]);                    ·····················⑥

      }
```

①C言語の御約束.

②C言語の御約束.

③文字変数型で10個の配列を宣言する.

④scanf文を使う場合，何を入力すべきかという「入力促進メッセージ」をあらかじめ表示するようにすると入力するとき，戸惑うことが少なくなる.

⑤複数の文字を一度に入力するとき変換指示記号は%sになる．このとき，入力された文字の1文字目がc[0],2文字目がc[1],3文字目がc[2],…と続く．詳しくは2章1節終行辺りを参照すること.

また，本プログラムでは10文字以上入力した場合，エラーにはならないが10文字までが正しく入力され，11文字以降は無視される．場合によっては他のデータを破壊していることもあるのでプログラムするときはむしろ余裕をもって配列を宣言するように注意する.

ところで，このscanfをエラーではないかと考えた人間は，3章までの内容を理解していると思われる．通常の変数の場合，scanfでは変数名に&をつけるのが規則であった.この&については6章で詳しく述べるが，配列変数の場合，&は必ずしも必要ではなく，したがってこの構文は正常である.

⑥1文字目から順番に表示する．配列の0番目に名前の1文字目が入力されたことがわかるはずである.

配列にあらかじめ文字を設定することも出来る.
以下のプログラムを入力し，実行する.

ファイル名：EX43.C

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

main()
      {
       char c[]="Science of Information.";
       int a;

       printf ("%s¥n",c);
       printf ("Exchange of upper message.Select No.(1-22)  ");
       scanf ("%d",&a);
```

```
    if (a<1 || a>22)
                    {
                     exit(0);
                    }
    else
                    {
                     printf ("Input any character: ¥n");
                     c[a-1]=getch();
                    }

    printf ("%s¥n",c);

   }
```

配列は必ずしも添字を記述しなければならないということはない．配列の適当な個数を数えるのが難しいとき，文字列を""でくくり，代入することで適当な数が設定される．
また，このプログラム中で使用したexit(0)はmain関数を強制的に終わらせる関数であり，process.hで定義されている．関数について詳しくは第５章で述べる．

4.2 for

繰り返し処理をする場合，一定回数繰り返す場合と，条件が満たされるまで繰り返す場合がある．この節では繰り返し回数が分かっている場合の繰り返しについて述べる．
まず以下の２つのプログラムを入力し，実行してみる．

ファイル名:EX44.C

```
#include <stdio.h>

main()
    {
     int i;

     for (i=1;i<10;i++)
                     {
                      printf ("%d times repeated.¥n",i);
                     }

    }
```

ファイル名:EX45.C

```
#include <stdio.h>

main()
      {
       int i,a;

       a=0;

       for (i=0;i<10;i++)
                          {
                           a+=i;
                           printf ("+%d",i);
                          }

            printf ("=%d",a);

      }
```

上の２つのプログラムはそれほど長くない．しかし，画面には９回以上表示した形跡があらわれる．これが繰り返し処理の効果である．このようにforを使うと処理を何回も繰り返すことが容易になる．以下に書式を示す．

```
for(初期条件;継続条件;演算処理)
                                    {
                                      実行文;
                                    }
```

初期条件として例えば変数iに0を代入し，継続条件としてi<10を指定したのがEX44.CおよびEX45.Cである．ここで演算処理i++というのはＣ言語特有の演算であり以下の表のような意味がある．

Table4.1　++と--の意味

表記	意味
i++	1加算する
i--	1減算する
++i	1加算する
--i	1減算する

i++と++iの区別は難しいが，次の2つのプログラムを実行すると理解しやすい．

ファイル名：EX46.C

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

```
main()
      {
       int i,j,a;

       a=0;
       j=0;

       for (i=0;i<10;i++)
                        {
                         a+=j++;
                         printf ("%d   %d    %d¥n",i,j,a);
                        }

            printf ("%d",a);

      }
```

ファイル名：EX47.C

```
#include <stdio.h>

main()
      {
       int i,j,a;

       a=0;
       j=0;

       for (i=0;i<10;i++)
                        {
                         a+=++j;
                         printf ("%d   %d    %d¥n",i,j,a);
                        }

            printf ("%d",a);

      }
```

C言語ではa=a+1;をa+=1;と記述出来る．演算は加算だけではなくすべての演算で有効である．

ここで，a+=j++;と記述するとa+=j; j++;の順番で2行に書くのと等価である．

しかし，a+=++j;と記述するとj++; a+=j;の順番で2行に書くのと等価になる．

このように，演算記号の位置で演算結果が大きく変わってくることがよくあるので注意する．
次に，以下のプログラムを入力し，実行する．

ファイル名：EX48.C

```
#include <stdio.h>

main()
      {
       char c[10];
       int i;

       printf ("Name(MAX:10)  ");
       scanf ("%s",c);

       for (i=0;i<10;i++)
                        {
                         printf ("%c¥n",c[i]);
                        }
      }
```

このプログラムではEX42.Cと同じ結果が得られる．10文字程度ならば順番に書いていってもなんとか記述出来るが，仮にこれが100文字以上となった場合，プログラムはとても書ききれず，どこかで入力を間違える可能性も出てくる．しかし，繰り返し処理を行なうことでプログラムも読みやすくなり，間違える可能性も少なくなる．同一の処理を行なうならば繰り返し処理をした方が，有利である．

ファイル名：EX49.C

```
#include <stdio.h>

main()
      {
       char c[]="Science of Information.";
       int i;

       printf ("%s¥n",c);

          for (i=0;i<23;i++)
                           {
                            if (c[i]=='o')
                                         {
```

```
                                    c[i]='O';
                                   }
                        else if (c[i]=='c')
                                   {
                                    c[i]='D';
                                   }
                            }

       printf ("%s¥n",c);

      }
```

ワープロではすでに常識となっているところの，長い文字列の中から特定の文字だけを探し出して置き換える「置換」作業は，くり返し処理をすることで意外と容易に行なえることがわかる．これを応用した暗号化の最も簡単な例を次のプログラムとして示す．

ファイル名：EX4A.C

```
#include <stdio.h>

main()
      {
       char c[]="Science of Information.";
       int i;

       printf ("%s¥n",c);

          for (i=0;i<23;i++)
                          {
                           c[i]+=1;
                          }

       printf ("%s¥n",c);

      }
```

文字のASCIIコードに1を加えただけで，全く意味のない文章が出来上がることが理解出来るはずである．このようにして文章を全く意味のないものに変換し，受渡後，逆の処理をすれば，元の文章が復活する．暗号化の手法を考えるのはそれだけで一大研究であり，ネットワーク時代の現在では，様々な手法が考案されている．

4.3 while（もしくはdo～while）

繰り返す回数が決まっているとき，forを使うのが最も便利である．しかし，実際に

繰り返しを行なうと，「～が終わるまで」とか，「～が入力されるまで」のような条件が成立するまで繰り返す方が便利なことも多数存在している．

このように条件が成立するまで繰り返す時に使われるのがwhileもしくは（do～while）である．基本的にwhileと（do～while）はほとんど同じであるので，以後はwhileのみを解説していく．

まず，次のプログラムを入力し，結果を確認する．

ファイル名：EX4B.C

```
#include <stdio.h>

main()
      {
       int i;

       i=0;
       while (i<10)
                  {
                   printf ("%d times repeated.¥n",i);
                   i++;
                  }

      }
```

このプログラムを実行すると，ここまでのプログラムを真面目に入力し，結果を確認した人間ならば見覚えのあるはずの結果（EX44.C参照）が得られる．

これはwhileを用いて，forと同じことをさせた場合のプログラムである．whileの書式は以下のようになる．

```
while（繰り返し条件）
                      {
                       実行文;
                      }
```

すなわち，whileでは繰り返し条件が満たされている限り，何度でも実行文を実行する．しかし，繰り返し条件を満たさない場合，一度も実行文を実行しない．

繰り返し条件についてコンパイラ内部では真偽のみを追求するので，例えば１だけを入力すると無限に繰り返しを行なうことも出来る．

続けて，whileを用いてforと同様の処理をしたプログラムを入力し，実行して結果を確認することにする．

ファイル名：EX4C.C

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

```
main()
      {
       int i,a;

       i=0;
       a=0;

       while (i<10)
                   {
                    a+=i++;
                    printf ("i=%d     a=%d¥n",i,a);
                   }

      }
```

次に，whileの本領であるところの「終了条件を満たした場合に終了する」プログラムを入力し結果を確認してみる．

ファイル名：EX4D.C

```
#include <stdio.h>

main()
      {
       char c[50];
       int i;

       printf ("Name(MAX:50)  ");
       scanf ("%s",c);

       i=0;

       while (c[i]!=0)
                       {
                        printf ("%c¥n",c[i]);
                        i++;
                       }
      }
```

このプログラムでの終了条件はc[i]==0である．c[]は文字列として定義されている．このとき，ASCIIコードでの0はNULLコードとして処理される．NULLとは「何もない」と

言う意味である.
　C言語では文字列の終末をNULLコードで規定しているので，文字列の最後は必ず0が入力されている．逆にいえば文字型配列変数では最初からNULLコードまでを一まとまりの文字列として取り扱っている．これを次のプログラムで確認することにする.

ファイル名：EX4E.C

```
#include <stdio.h>

main()
      {
       char c[]="Science of Information.";

       printf ("%s¥n",c);

          c[13]=0;

       printf ("%s¥n",c);

      }
```

　このプログラムを実行すれば，文字列が先のように定義されていることが一目瞭然である．次に，終了条件で処理出来ることの利点について考える.
　まず以下のプログラムを入力して実行する.

ファイル名：EX4F.C

```
#include <stdio.h>

main()
      {
       char c[]="Science of Information.";
       int i;

       printf ("%s¥n",c);

             i=0;

             while (c[i]!=0)
                           {
                            c[i]+=2;
                            i++;
                           }
```

```
       printf ("%s¥n",c);

      }
```

このプログラムをforを用いて記述すると，文字列の中に含まれる文字の数を予め数えておく必要がある．これを怠り，定義したすべての配列変数の個数について処理すると，時間の無駄と，予期しなかった結果を見ることになる．例えば，画面表示で考えた場合，初期化しなかった配列の予定外のところにASCIIコードで6が初めから入っていると，その文字を画面表示したときに，clear screenコードが送られて画面表示が全て消滅することもよく有る話である．これはEX4D.CとEX42.CおよびEX48.Cを比較すれば理解出来る．

scanfなどで不定に入力を行なう場合などの処理としては，whileを使う方が，特異的プログラムにならないので有利である．

ファイル名：EX4G.C

```
#include <stdio.h>

main()
      {
       int a,i;

             i=0;
             a=0;

             while (a<10000)
                           {
                            i++;
                            a=i*i*i;
                            printf ("i=%d   a=%d¥n",i,a);
                           }

      }
```

終了条件は，「何回繰り返し…」という予想のつくものだけではない．演算の結果や測定した結果がある一定の条件を満たした場合という条件設定も可能である．

forとwhileはそれぞれに長所があり，適材適所で使いこなす必要がある．

４章の課題

★'A'から'Z'までのアルファベットを１文字ずつ連続で表示すること
●1から100までの偶数の和をプログラムによって求めること
◆２乗した数が２桁である場合の元の数全ての和を求めること

５章　関数

Ｃ言語では関数をユーザーが自由に定義出来るのが特徴である．数学的表現でいう関数*f*(*x*)は変数*x*に対して一意的に決まる演算を行なうものであるが，Ｃ言語ではもう少し広い意味を持つ．

関数を作成するときのルールを以下に記す．

①関数は自由に作成出来る．
②関数には基本的に引数と戻り値がある．
③名前は比較的自由につけることが出来，名前の後に必ず『()』を必要とする．

具体的な関数の使用例を以下に記す．実行し意味を確認することにする．

ファイル名：EX51.C

```
#include <stdio.h>

int tri(int i)
              {
               int a;

               a=i*i*i;
               return (a);

              }
main()
      {
       int a,i;

             i=0;
             a=0;

             while (a<10000)
                           {
                            i++;
                            a=tri(i);
                            printf ("i=%d   a=%d¥n",i,a);
                           }

      }
```

この中でint tri(int i)が関数の名前である．intがついているのは，関数の終了時に返す戻り値をint型変数で戻すからである．戻り値とは関数の終了時にreturn()関数で戻される値のことをいう．また(int i)の部分を引数と呼び，関数を呼び出すときに，

前の関数の中で与えられるものである.

ところで，これまで使用してきたprintf,scanf,getch,mainなども関数である．基本的に使用される頻度が高い関数はANSI(JIS)-Cで規定されており，これらを標準関数と呼ぶ．標準関数は通常ヘッダファイルと呼ばれるものに定義されている．これらを利用するために記述するのがinclude文である．これまで御約束としていた#include <stdio.h>は標準関数を使用するための手続きである.

また，関数が自由に定義出来ると，どの関数から開始するべきか不明になる．そこで，Ｃ言語ではmain関数から開始することが定義されている．関数をどのような順番で記述するかは基本的に自由だが，main関数から始まることだけは変更出来ない.

次に戻り値のない関数を定義する．以下のプログラムを入力し，実行してみる.

ファイル名：EX52.C

```
#include <stdio.h>


void loop(int a)
                {
                 int i;

                 for (i=0;i<a;i++)
                                 {
                                  printf ("%d times repeated!¥n",i+1);
                                 }

                }

main()
      {
       int i;

       printf ("Input Number:");
       scanf ("%d",&i);

       loop(i);

      }
```

このプログラム中で定義されているloop関数は，表示のみを行い，前の関数に戻り値を戻さない．このように関数自体で完結する場合には必ずしも戻り値を必要としない．戻り値を持たない関数はvoid型として定義される．void型は省略が許可されており，記述しなくてもよい.

ファイル名：EX53.C

```
#include <stdio.h>


int sigma(int a,int b)
                      {
                       int c,i;

                       c=0;
                       for (i=a;i<=b;i++)
                                         {
                                          c+=i;
                                         }

                       return (c);

                      }

main()
      {
       int a,b,n;

       printf ("From :");
       scanf ("%d",&a);
       printf (" To  :");
       scanf ("%d",&b);

       n=sigma(a,b);

       printf ("Sum from %d to %d is %d",a,b,n);

      }
```

関数の戻り値は１つだけであり，２つ以上の戻り値を持つことはできない．しかし，関数の引数は一つとは限らず，必要に応じて幾つでもとることが出来る．

ところで，関数内で定義される変数はローカル変数と呼ばれ，その関数内でのみ有効である．したがって，他の関数で使用されている変数名でも使用することが出来る．

ファイル名：EX54.C

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

```
void vectr(int a)
                {
                 int i,j;

                 for (i=0;i<a;i++)
                                 {
                                  for (j=0;j<a;j++)
                                                        {
                 if (i==j)
                          {
                           printf ("1 ");
                          }
            else
                          {
                           printf ("0 ");
                          }
                                                        }
                           printf ("¥n");
                                 }

              }

main()
      {
       int n;

       printf ("Input No:");
       scanf ("%d",&n);

       vectr(n);

      }
```

これはｎ次の正方行列を表示するプログラムである.
関数を定義する目的は大別すると２つある.
一つは頻繁に使われる一連の処理を関数として定義し，同一の処理を重複して記述するのを避けることであり，もう一つは複雑な処理を一つの関数としてまとめることで，全体の流れを見やすくすることである．上の例で言えばvectr関数の中では複雑な処理を行なっているが，main関数の中はすっきりとしており，見やすくなっている.

ファイル名：EX55.C

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

```
int power(int a,int n)
                    {
                          if (n==0) return (1);
                     else           return (a*power (a,n-1));
                    }

main()
      {
       int a,b,i;

       a=2;

       for (i=1;i<=10;i++)
                         {
                          b=power (a,i);
                          printf ("%d **    %d = %d¥n",a,i,b);
                         }

      }
```

関数は自分自身も含めてどの関数からでも自由に呼び出すことが出来る．ただし，自分自身から呼び出す場合には，どこかで無限にループしないようなプログラムの構造になっていることが望ましい．

次に，標準関数の中で多用される基本的な関数を取り上げていく．

以下のプログラムを入力し，実行してみる．

ファイル名：EX56.C

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


void disp(float r)
                  {
                   char c[]="                    ";
                                          /*20 spaces of character*/
                   int x;

                   x=sin(r)*10+10;

                   c[x]='*';
```

```
                    printf ("%s¥n",c);

                   }

main()
      {
       int i;
       float pi=3.1415926;
       float r;

       for (i=1;i<=20;i++)
                         {
                          r=pi*2/20*i;
                          disp (r);
                         }

      }
```

sin()関数は三角関数のSIN関数と同義である．math.hに定義されており，初めにmath.hをincludeする必要がある．通常複数のヘッダファイルをincludeする場合，順不同で並べればよい．

ところで，Ｃ言語では角度の単位はdegreeではなくradianである．したがって角度を計算中で使用する場合，radianに変換しなければならない．

ファイル名：EX57.C

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


void disp(float r)
                  {
                   char c[]="                                    ";   /*20 spaces of
character*/
                   int x;

                   x=cos(r)*10+10;

                   c[x]='*';

                   printf ("%s¥n",c);

                  }
```

```
main()
      {
       int i;
       float pi=3.1415926;
       float r;

       for (i=1;i<=20;i++)
                          {
                           r=pi*2/20*i;
                           disp (r);
                          }

      }
```

これはcos()関数の使用例である. cos()関数は三角関数のCOS関数と同義である. 角度の単位がradianであることもsin()関数と同じである. 数学的関数はmath.hで定義されていることが多い.

ファイル名：EX58.C

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


void disp(float r)
                  {
                   char c[]="                    ";
                            /*20 spaces of character*/
                   int x;

                   x=log(r)*6;

                   c[x]='*';

                   printf ("%s¥n",c);

                  }

main()
      {
       int i;
       float r;
```

```
       for (i=1;i<=20;i++)
                          {
                           r=i;
                           disp (r);
                          }

      }
```

log()関数は自然対数を計算する関数である．常用対数の計算はlog10()関数を使用するので注意が必要である．

ファイル名：EX59.C

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

 main()
      {
       int i,d;
       float f;
       unsigned long time1;
       unsigned seed;

       time(&time1);
       seed=time1;
       srand(seed);

        for (i=0;i<50;i++)
                          {
                           f=rand();
                           d=f*10/32768.0;
                           printf ("%d ",d);
                          }

      }
```

rand()関数はstdlib.hで定義されている乱数発生関数で，0～32767(intの正の最大値)の範囲の乱数を発生する．乱数とは一切の法則無しに，ランダムに発生した数のことである．0～32767の範囲の乱数を32768.0で割ると，0～1未満になるので，*n*倍すれば0～*n*未満となる．この演算結果をint型の変数dに代入すると，整数化されて0～*n*-1の乱

数を得ることができる．例えば，電子サイコロのように1～6としたいならば，*n*=6とし，演算結果に1加えれば良い．

コンピュータで乱数を発生させると一様分布乱数となる．実際には発生パターンに法則が認められ，厳密な意味での乱数ではない．しかし，ゲームのイベント発生条件などのようにサイコロで決定されるような判定や，非線形最小二乗法における初期条件の偏差決定などには有効である．

通常乱数を使用する場合は，このプログラムのようにシステム時間呼出関数time()と乱数発生条件初期化関数srand(seed)により，乱数使用ごとに乱数を初期化するように設定し，乱数に法則性が見いだせない程度に攪乱する．seedは0～65535で任意の整数を与える．

ファイル名：EX5A.C

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

 main()
      {
       char c1[]="Australia";
       char c2[20];

       printf ("Before Copy¥n¥n");
       printf ("c1:%s ¥n",c1);
       printf ("c2:%s ¥n",c2);

       strcpy (c2,c1);

       strcpy(c1,"America");

       printf ("¥nAfter Copy¥n¥n");
       printf ("c1:%s ¥n",c1);
       printf ("c2:%s ¥n",c2);

      }
```

strcpy()関数は文字列を複写する関数で，strcpy(a,b)のように記述し，aの文字列にbの文字列を複写する．このとき，複写先の文字列の配列が，複写元の文字列の長さより短い場合，他の変数のデータ領域を破壊するので複写先の配列長を長めに取るなどの注意が必要である．

ファイル名：EX5B.C

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

```
#include <stdlib.h>
#include <string.h>
#include <time.h>

 main()
      {
       int i,j,d,cnt,n;
       float f;
       unsigned long time1;
       unsigned seed;
       char scodon[4],codon[4],dna[200];

       time(&time1);
       seed=time1;
       srand(seed);

       strcpy (scodon,"AUG");
                                              /* scodon ="AUG"  Start Codon */
       cnt=200;

        for (i=0;i<cnt;i++)
                          {
                           f=rand();
                           d=f*4/32768.0;
                           switch (d)
                                     {
                                     case 0:
                                            dna[i]='A';
                                            break;
                                     case 1:
                                            dna[i]='U';
                                            break;
                                     case 2:
                                            dna[i]='G';
                                            break;
                                     case 3:
                                            dna[i]='C';
                                            break;
                                     default:
                                            break;
                                     }
                          }
                          dna[199]='¥0';
                          codon[3]='¥0';

                          printf ("%s ¥n¥n",dna);
```

```
        for (i=0;i<cnt-2;i++)
                          {
                for (j=0;j<3;j++)
                                  {
                                   codon[j]=dna[i+j];
                                  }
                              n=strcmp(scodon,codon);
                if (n==0)
                         {
                          printf (" ");
                         }
                          printf ("%c",dna[i]);

                             }

}
```

strcmp()関数は2つの文字列を比較し，その大小を判定する．具体的には各文字列の1文字目から順にキャラクターコードを比較し，キャラクターコードの異なる文字が出た時点でその大小を判別し，文字列の大小とする．strcmp(a,b)のように記述し，a>bの場合に正の値，a=bの場合0，a<bの場合負の値を戻り値として返す．

このプログラムでは，まずA(adenine)，U(uracil)，G(guanine)，C(cytosine)の4種類の塩基のいずれか200個で構成されるmRNAを乱数で擬似的に作成する．mRNAに書き込まれた塩基は順に3つずつの組み合わせ，すなわちトリプレット(triplet)でアミノ酸の配列順序を決めるための暗号を現しており，これをコドン(codon)という．遺伝子解析するためには，まずどこから蛋白質の合成が開始されるかを見つけ出す必要があり，この解析はコンピュータで自動化される．すなわち，メチオニン(methionine)のコードであると同時に蛋白質合成開始部位を示す開始コドン(initiation codon)はAUGの配列であることが分かっているので，これを見つけ出し，その前に空白を挿入して開始コドンであることを示す．

５章の課題

●関数（もしくは関数の組）を作成して，入力された数値（総数不明）の平均値を求めること．

◆任意のｎに対するｎ！を求めること．

★任意のｎの倍数のうちの２桁の数の平均値を求めること．

▲EX5B.Cを改良して開始コドンから停止コドンまでを明示するプログラムを作成せよ．このとき，コドンごとの単位で考え，遺伝子の欠損があれば指摘するようにせよ．

6章　ポインタ

この章ではC言語を学ぶ上で最も難関であるポインタについて解説する.

6.1　ポインタの定義

ポインタを使わずに実行可能なプログラムを作成することは，ほとんどの場合不可能ではない．しかし，ポインタを使用することでプログラムの構造が単純になり，効率的なプログラミングが行なえる.
まず，ポインタ変数を使用したプログラムを実行し，その意味を考える.

ファイル名：EX61.C

```
#include <stdio.h>

main()
      {
       int a;
       int *point;

       a=5;
       *point=a;

       printf("a=%d     ",a);
       printf("*point=%d     ",*point);
       printf("point=%x     ",point);

      }
```

ポインタは変数などがメモリ上のどこに存在するかを示すアドレスを取り扱う.
書式は以下のようになる.

```
変数型　*変数名;
```

*が頭についているときは，通常の変数と同様に扱い，数値や文字の代入，演算などを行なうことが出来る．ところが*がついていないとき，その変数はアドレスを表しており，その変数の内容が格納されているメモリ上のアドレスを扱っている.
先のプログラムEX61.Cを例にして詳しく解説していく.

```
#include <stdio.h>          ……①

main()                      ……②
      {
       int a;               ……③
```

```
    int *point;              ·····④

    a=5;                     ·····⑤
    *point=a;                ·····⑥

    printf("a=%d     ",a);              ·····⑦
    printf("*point=%d     ",*point);     ·····⑧
    printf("point=%x     ",point);       ·····⑨

  }
```

①標準入出力関数用ヘッダファイルをincludeしている．標準入出力関数を使用するときには必ずincludeする必要があり，標準入出力関数を使用しないプログラムはほとんどない（作れないことはないが，それでいて意味のあるプログラムを作成するのはかなりの高等技術である）のでＣ言語の御約束となる．

②Ｃ言語ではこの関数がないと，どこから始めるか分からなくなるのでＣ言語の御約束となる．

③通常の変数宣言．

④ポインタ変数の宣言．

⑤通常の変数の代入．

⑥ポインタ変数*pointに変数aの内容を代入している．

⑦通常の変数の内容の表示．

⑧ポインタ変数の内容の表示．

⑨ポインタ変数の内容を記憶しているメモリ領域のアドレスを表示．アドレスは通常16進数で表現するので，変換指示記号も%xを使用する．

このようにポインタ変数を使用することで，メモリ領域内の特定のアドレスの内容を表示することが出来る．次に，アドレスを用いた変数の代入を例としてあげる．入力して実行し，意味を考える．

ファイル名：EX62.C

```
#include <stdio.h>

main()
    {
```

```
        int a;
        int *point;

        a=5;
        printf("a=%d     ",a);
        printf("&a=%x    ",&a);
        printf("*point=%d     ",*point);
        printf("point=%x     ¥n",point);

        *point=a;
        a=9;

        printf("a=%d     ",a);
        printf("&a=%x    ",&a);
        printf("*point=%d     ",*point);
        printf("point=%x    ¥n",point);

        point=&a;
        a=1;

        printf("a=%d     ",a);
        printf("&a=%x    ",&a);
        printf("*point=%d     ",*point);
        printf("point=%x     ¥n",point);

      }
```

これまでの章で変数名に&をつけて表した例は一つしかない．すなわち，scanf関数である．すべての変数は&を頭につけることで，その変数のアドレスを取り出すことが出来る．しかし，そのアドレスに対して直接書き込むことは困難である（不可能ではないが安易に行なって失敗するとシステムの暴走を招きかねない高等技術である）．ところが，ポインタ変数を用いると比較的容易になる．

先のプログラムEX62.Cを順を追って解説していく．

```
#include <stdio.h>                                  ……①

main()                                              ……②
      {
        int a;                                      ……③
        int *point;                                 ……④

        a=5;                                        ……⑤
        printf("a=%d     ",a);                      ……⑥
        printf("&a=%x    ",&a);                     ……⑦
        printf("*point=%d     ",*point);            ……⑧
```

```
    printf("point=%x     ¥n",point);        ·····⑨

    *point=a;                               ·····⑩
    a=9;                                    ·····⑪

    printf("a=%d     ",a);                  ·····⑥
    printf("&a=%x    ",&a);                 ·····⑦
    printf("*point=%d     ",*point);        ·····⑧
    printf("point=%x    ¥n",point);         ·····⑨

    point=&a;                               ·····⑫
    a=1;                                    ·····⑪

    printf("a=%d     ",a);                  ·····⑥
    printf("&a=%x    ",&a);                 ·····⑦
    printf("*point=%d     ",*point);        ·····⑧
    printf("point=%x     ¥n",point);        ·····⑨

}
```

①Ｃ言語の御約束.

②Ｃ言語の御約束.

③通常の変数宣言.

④ポインタ変数の宣言.

⑤通常の変数の代入.

⑥通常の変数の内容の表示.

⑦通常の変数を格納しているメモリ領域のアドレスの表示.

⑧ポインタ変数の内容の表示.

⑨ポインタ変数を格納しているメモリ領域のアドレスの表示.

⑩ポインタ変数に通常変数の内容を代入.

⑪通常変数の内容を変更.

⑫ポインタ変数の取り扱うアドレスを，通常変数のアドレスに変更.

このプログラムと実行結果を比較すると，ポインタの意味が比較的分かりやすい.

まず最初の表示の時点ではポインタ変数の内容は不定であり，実行したコンピュータや，時刻などによって変動する．

次に，⑩でポインタ変数に通常変数を代入しているので，ポインタ変数*pointの内容は，⑩の代入した時点での変数aの値5となっており，代入直後にaの値を変更してもポインタ変数*pointの内容は代入された時点のものである．

最後に⑫ではポインタ変数の扱うアドレスを変数aの格納されているメモリ領域のアドレスに書き換えている．これによりポインタ変数*pointは変数aと同一の内容を指すようになる．したがって，変数aの内容を変更すると同時にポインタ変数*pointの内容も変更されてしまい，前の変数aの内容である9にはならない．

それでは次に，ポインタ変数によって配列変数を取り扱う．以下のプログラムを代入して実行してみることにする．

ファイル名：EX63.C

```
#include <stdio.h>

main()
      {
       int a[5],i;
       int *point;

       printf ("First¥n¥n");

         for (i=0;i<5;i++)
                        {
                         a[i]=i*2;

               printf ("i=%d   a[%d]=%d  *point=%d¥n",i,i,a[i],*(point+i));
                        }

         point=&a[0];

       printf ("¥nAfter  copy¥n¥n");

         for (i=0;i<5;i++)
                        {
               printf ("i=%d   a[%d]=%d  *point=%d¥n",i,i,a[i],*(point+i));
                        }

       printf ("¥nChange value¥n¥n");

         for (i=0;i<5;i++)
                        {
                         *(point+i)=i*3;
```

```
                printf ("i=%d    a[%d]=%d  *point=%d¥n",i,i,a[i],*(point+i));
                        }

      }
```

配列変数はメモリ領域の中に連続して領域を確保している．したがって，ポインタ変数pointに配列の最初の変数a[0]のアドレスを代入することで，以降の配列もポインタ変数を使って取り出すことが出来る．さらに，ポインタ変数を書き換えることにより，配列変数で取り出される変数も変更出来る．これはポインタ変数で取り扱うアドレスを配列の最初の変数のアドレスと同一にしたことで，配列変数の格納されているメモリ領域の内容，すなわち配列変数を直接書き換えることになるからである．

6.2 call by reference

ポインタの基本的概念は前節で述べた通りである．しかし，通常の変数と同様の使い方をしている範囲では，むしろ書式や使用法が複雑であり，プログラムの記述と理解を難解にするだけである．にもかかわらずポインタは必要な概念である．この節ではポインタの応用的な使用法を述べる．

まず，以下のプログラムを入力し，実行してみることにする．

ファイル名：EX64.C

```
#include <stdio.h>

void set_value(int *point)
                {

                 *point=5;

                }

main()
      {
       int a;

       a=2;

       printf ("First a=%d¥n",a);

       set_value(&a);

       printf ("After a=%d¥n",a);

      }
```

プログラム中で定義されているset_value()関数へはポインタ変数を用いて，main()関数内で定義された通常変数aのアドレスを受け渡している．
set_value()関数では受け渡されたアドレスの変数の内容，すなわち通常変数aの内容を書き換えている．
ポインタ変数を受け渡すとき，受け取り先の関数内での変数型と呼出し関数内で受け渡す変数の変数型とが一致していなければならない．
ポインタ変数はアドレスを渡されるごとに変更されるので，変数型が同じならばどの変数にも対応出来る．これを次のプログラムで確認する．

ファイル名：EX65.C

```
#include <stdio.h>

void set_value(int *point)
                {

                 *point=5;

                }

main()
      {
       int a,b,c,d,e,f,g;

       a=1;
       b=2;
       c=3;
       d=4;
       e=6;
       f=7;
       g=8;

       printf ("First a=%d¥n",a);
       printf ("First b=%d¥n",b);
       printf ("First c=%d¥n",c);
       printf ("First d=%d¥n",d);
       printf ("First e=%d¥n",e);
       printf ("First f=%d¥n",f);
       printf ("First g=%d¥n",g);

       set_value(&a);
       set_value(&c);
       set_value(&e);
```

```
        set_value(&g);

        printf ("¥n");

        printf ("After a=%d¥n",a);
        printf ("After b=%d¥n",b);
        printf ("After c=%d¥n",c);
        printf ("After d=%d¥n",d);
        printf ("After e=%d¥n",e);
        printf ("After f=%d¥n",f);
        printf ("After g=%d¥n",g);

      }
```

このプログラムを実行すると，set_value()関数に受け渡した変数のみ変更されることが確認出来る．

EX64.C，EX65.Cのように，関数への変数の受け渡しをポインタ変数すなわち変数のアドレスによって行なうことを『call by reference』という．これに対して関数に直接変数を受け渡すことを『call by value』という．

call by referenceとcall by valueの最大の違いは，戻り値を必要としないかするかという点である．call by referenceではアドレスを受け渡しているので，直接アドレスに書き込むことが出来る．従って，元の関数で定義されている変数を直接書き換えることが可能なので，戻り値をreturn()関数で戻す必要はない（戻しても間違いではない）．

これに対してcall by valueの場合，値を返す場合は必ずreturn()関数で戻さなければならない．しかもreturn()関数で返せる戻り値は１つのみであるので，演算の結果出てきた２つ以上の数値を戻すと言うことはできない．

これを次のプログラムで確認する．

ファイル名：EX66.C

```
#include <stdio.h>

void set_value(int *point)
                {
                 int i;

                 for (i=0;i<10;i++)
                                        {
                                         *(point+i)=i;
                                        }

                }
```

```
main()
      {
       int a[10],i;

                for (i=0;i<10;i++)
                                {
                                 printf ("First a[%d]=%d¥n",i,a[i]);
                                }

       set_value(&a[0]);
                                 printf ("¥n");

                for (i=0;i<10;i++)
                                {
                                 printf ("After a[%d]=%d¥n",i,a[i]);
                                }

      }
```

ここでは配列変数を受け渡している．配列変数は配列の最初の変数（添字が0の変数）のアドレスを受け渡すことで，このプログラムのように配列変数全てを処理することが出来る．ここでは配列の個数が分かっているのでforのループを利用したが，条件を設定することでwhileのループも利用出来る．

ファイル名：EX67.C

```
#include <stdio.h>

void calc(float *point)
               {
                int i;
                float total=0.0;

                for (i=0;i<3;i++)
                                {
                                 total+=*(point+i*2);
                                }

                printf ("Total  %f[L]¥n",total);

                for (i=0;i<3;i++)
                                {
                                 *(point+i*2+1)=*(point+i*2)/total*100.0;
                                }
```

```
            }

main()
      {
       int i;
       float a[6];

                for (i=0;i<3;i++)
                                {
                                 printf ("Volume %d [L]:",i);
                                 scanf ("%f",&a[i*2]);
                                }

                                 printf ("¥n");

       calc(&a[0]);

             for (i=0;i<3;i++)
                       {
                        printf ("Volume %d    %f[L] ",i,a[i*2],a[i*2+1]);
                        printf ("    %f[Percent]¥n",a[i*2+1]);
                       }

      }
```

実用的なプログラムでは１つの関数である程度のまとまった処理をさせることが多い．目的ごとに，関数を作成すると，変数の内容を渡して，幾つかの演算の結果をまとめて返すという形で利用出来るので，プログラムが理解しやすくなる．

しかしながら，関数の下に関数を作成し，そのたびにすべての変数を受け渡すような処理を重ねるとかえってプログラムが難解になることもあるので注意した方が良い．

6.3 ファイル操作

全てのＯＳはファイルを取り扱う．ファイルは大きく分けると２種類に分類される．すなわち，**テキストファイル**と**バイナリファイル**である．

テキストファイルはASCIIコードもしくはJISコード，Shift JISコードなどで構成されており，エディタなどで編集することが出来る．ここまで記述してきたＣ言語のソースファイルはこれに分類される．

バイナリファイルはほとんどの場合バイナリコードというもので構成されており，特殊な方法でしか編集することができない．ソースファイルをコンパイルして出来る実行ファイルはこれに分類される．

プログラムを作成して実験データを処理する場合など，データを記録しておいたり，あるいは記録されたデータを取り出して処理することが出来れば，同じデータを再度入力する手間が省け，非常に便利である．

この節では主に汎用性の高いテキストファイルの操作に限定して，解説する．
まず，以下のプログラムを入力して実行してみる．

ファイル名：EX68.C

```
#include <stdio.h>

main()
      {
       int i,a[10];
       FILE *fp;

       fp=fopen ("EX68.DAT","w");

       for (i=0;i<10;i++)
                        {
                         a[i]=i*3+1;
                         printf ("%d -------->%d¥n",i,a[i]);
                        }

       for (i=0;i<10;i++)
                        {
                         fprintf (fp,"%d    %d¥n",i,a[i]);
                        }
       fclose(fp);

      }
```

このプログラムを実行すると，コンピュータのディスクアクセスランプが通常より長く点灯する．これはディスク上にファイルを作成しているからである．
実際にファイルEX68.DATが作成されていることはDOSコマンドのDIR[Enter]と入力し，ディレクトリの中を確認すれば良い．
ファイルに書き込んだり，読み出したりするためには，ファイルを開かなければならない．そのための関数はfopenであり，基本的な書式は以下のようになる．

ファイルポインタ=fopen("ファイル名","オープンモード")

オープンモードは表７のように定義されている．
オープンモードはこの他にもあるが，ここでは最も使用頻度の高い３つに限定し，テキストファイルとバイナリファイルのそれぞれの場合について記した．実際の使用では上から２つのオープンモードだけで十分である．
ファイルポインタはFILE型で定義されるポインタ変数である．ファイルを処理するときに，複数のファイルを取り扱うとどのファイルを処理しているのか分からなくなる恐れがある．これを回避するために，ファイルを開くたびにつける記号だと思えば良い．

Table6.1　fopenのオープンモード

w	**書き込み専用でファイルをオープン・テキストファイル**
r	**読み込み専用でファイルをオープン・テキストファイル**
a	**追加書き込み用にファイルをオープン・テキストファイル**
wb	**書き込み専用でファイルをオープン・バイナリファイル**
rb	**読み込み専用でファイルをオープン・バイナリファイル**
ab	**追加書き込み用にファイルをオープン・バイナリファイル**

一旦，ファイルを開いたら，以後の処理は全てファイルポインタで区別する．

このようにしてファイルをオープンしたら，ファイルの内容を書き込む，あるいは読み出すことになる．書き込むための関数は幾つかあるが，画面に文字を表示する関数であるprintfとほとんど同様の書式を適用出来るfprintf()関数が理解しやすい．書式は以下のようになる．

```
fprintf(ファイルポインタ,"変換指示記号",変数);
```

書き込まれたデータを読み出すときには，キーボードからの入力関数であるscanfとほとんど同様の書式を適用出来るfscanf()関数が理解しやすい．書式は以下のようになる．

```
fscanf(ファイルポインタ,"変換指示記号",&変数);
```

そして，ファイルの書き込みもしくは読み出しが終了したらfclose()関数でファイルを閉じなければならない．書式は以下のようになる．

```
fclose(ファイルポインタ);
```

ところで，ファイルは常に開くことが出来るとは限らない．ディレクトリエントリの個数の関係上，新しいファイルを作成出来なかったり，あるいは読み出そうとしたファイル名のファイルが存在しなかったりするとエラーが生じる．この類のエラーはその後の処理が不定であるので，プログラム作成時には対策を講じておかなければならない．プログラムEX68.Cに対策を講じた次のプログラムを入力し，動作を確認することにする．

ファイル名：EX69.C

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

main()
      {
       int i,a[10];
       FILE *fp;

       if ((fp=fopen ("EX69.DAT","w"))==NULL)
                                       {
```

```
                                        printf ("File cannot open!!");
                                        exit(0);
                                       }

       for (i=0;i<10;i++)
                          {
                           a[i]=i*3+1;
                           printf ("%d -------->%d¥n",i,a[i]);
                          }

       for (i=0;i<10;i++)
                          {
                           fprintf (fp,"%d    %d¥n",i,a[i]);
                          }
       fclose(fp);

      }
```

NULLはファイルを開けなかった場合などのエラーコードを定義した変数であり，標準的に使用出来るものである．この他，ファイルの終了を知らせるエラーコードEOFなども定義されている．エラーコードEOFの使用例はプログラムEX6C.Cを参照するとよい．

今度は，EX69.Cによって作成されたファイルをオープンして，データを読み出す次のプログラムを入力し，実行する．

ファイル名：EX6A.C

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

main()
      {
       int i,a[10],b[10];
       FILE *fp;

       if ((fp=fopen ("EX69.DAT","r"))==NULL)
                                        {
                                         printf ("File cannot open!!");
                                         exit(0);
                                        }

       for (i=0;i<10;i++)
                          {
                           fscanf (fp,"%d    %d",&a[i],&b[i]);
                          }
```

```
        for (i=0;i<10;i++)
                        {
                         a[i]=i*4+2;
                         printf ("%d   :%d -------->%d¥n",i,a[i],b[i]);
                        }

        fclose(fp);

       }
```

EX69.Cによってファイルが作成されていない場合との動作の違いを比較するために，ファイル名：EX69.DATを消去すると良い．間違ってEX69.Cを消去しないように気をつけるようにする．

今度はファイルを複数同時に開く場合のプログラムを考える．次のプログラムを入力して実行してみる．

ファイル名：EX6B.C

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

main()
       {
        int i,a[10],b[10];
        FILE *fi,*fo;

        if ((fi=fopen ("EX69.DAT","r"))==NULL)
                                        {
                                         printf ("File cannot open!!");
                                         exit(0);
                                        }
        else
            {

        for (i=0;i<10;i++)
                        {
                         fscanf (fi,"%d    %d",&a[i],&b[i]);
                        }

            }

        if ((fo=fopen ("EX6B.DAT","w"))==NULL)
                                        {
                                         printf ("File cannot open!!");
```

```
                                        exit(0);
                                       }
      else
          {

                          printf ("No.  decimal   hexadecimal¥n");
      for (i=0;i<10;i++)
                         {
                        printf ("%x     %d          %x¥n",a[i],b[i],b[i]);
                        fprintf (fo,"%x   %d     %x¥n",a[i],b[i],b[i]);
                         }

          }

      fclose(fi);
      fclose(fo);

     }
```

ファイルを複数開く場合は，開くファイルの分だけファイルポインタを設定すれば良い．ところで，読み出すデータの個数が分かっているときは，forのループを設定すれば良いが，現実のデータ処理ではデータの個数が決まっていないこともよくある．このような場合の処理の方法を次のプログラムを実行して確認する．

ファイル名：EX6C.C

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

main()
     {
      int i,k,a[10],b[10];
      FILE *fi;

      if ((fi=fopen ("EX69.DAT","r"))==NULL)
                                       {
                                        printf ("File cannot open!!");
                                        exit(0);
                                       }
      else
          {

           k=0;
```

```
        while ((fscanf (fi,"%d    %d",&a[k],&b[k]))!=EOF)
                        {
                         k++;
                        }

    fclose(fi);

        }

                         printf ("No.  data¥n");
    for (i=0;i<k;i++)
                        {
                         printf ("%x     %d¥n",a[i],b[i]);
                        }

  }
```

データの個数が分からないときは，データを読み出すと同時にデータの個数をカウントすることにより，以後の処理をforのループで処理することが出来るようになる．

６章の課題

★１つの関数で同時に球の表面積と体積を求める関数を作成し，入力された半径に対してこれらを求め表示せよ．

●テキストファイルを別の名前のテキストファイルとしてコピーするプログラムを作成せよ

▼身長・性別・年齢から肺活量予測値ならびに努力性肺活量予測値を同時に求める関数を作成し，自分の予測値を確認せよ．ただし，肺活量予測値ならびに努力性肺活量予測値は以下の式で求められる．

肺活量予測値（男）＝[27.63-(0.112*年齢)]*身長(cm)
肺活量予測値（女）＝[21.78-(0.101*年齢)]*身長(cm)
努力性肺活量予測値（男）＝51.0*身長-25.0*年齢-3,550±590
努力性肺活量予測値（女）＝33.0*身長-23.0*年齢-1,400±460

7章　構造体

これまでデータを扱うとき，扱うデータの型に応じて変数を定義し，利用してきた．

しかし，住所録のように異なるデータ形式の変数を組み合わせて利用し，しかも，このフォーマットを多数の人間について共通して利用したい場合がある．

このようなときに，異なる変数型の変数群を，構造体という一つの型として定義し，これを利用することによって，共通するフォーマットを容易に利用可能にすることが出来る．

構造体の定義は以下のように行なう．

```
struct 構造体タグ
                {
                 型  メンバ名;
                 型  メンバ名;
                 型  メンバ名;
                 型  メンバ名;
                  ・    ・
                  ・    ・
                  ・    ・
                  ・    ・
                };
```

続いて，構造体タグを用いて変数の定義を行なう．

```
struct 構造体タグ 変数名
```

具体例をもとに解説を行なうことにする．

まず，以下のプログラムを入力し，実行してみる．

ファイル名：EX71.C

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

struct data
          {
           char  name[40];
           char  address[100];
           int   age;
           float height;
           float weight;
          }

main ()
```

```
       {
        struct data a;

        strcpy(a.name,"Mr.X");
        strcpy(a.address,"Bunkyo Yushima 1-5-45");
        a.age=20;
        a.height=175.0;
        a.weight=60.5;

        printf ("My name is %s.¥n",a.name);
        printf ("I live at %s.¥n",a.address);
        printf ("I am %d years old.¥n",a.age);
        printf ("My height is %f cm.¥n",a.height);
        printf ("My weight is %f kg.¥n",a.weight);

        printf ("Let's calculate my BMI.¥n");

       }
```

このプログラムは構造体を用いたプログラムとして最も単純なものの中の一つである.
以下に順を追って解説していく.

```
#include <stdio.h>    ·····①
#include <string.h>   ·····②

struct data           ·····③
          {
           char  name[40];          ·····④
           char  address[100];      ·····④
           int   age;               ·····④
           float height;            ·····④
           float weight;            ·····④
          }

main ()               ·····⑤
       {
        struct data a;    ·····⑥

        strcpy(a.name,"Mr.X");     ·····⑦
        strcpy(a.address,"Bunkyo Yushima 1-5-45");     ·····⑦
        a.age=20;                  ·····⑦
        a.height=175.0;            ·····⑦
        a.weight=60.5;             ·····⑦

        printf ("My name is %s.¥n",a.name);            ·····⑧
```

```
        printf ("I live at %s.¥n",a.address);          ……⑧
        printf ("I am %d years old.¥n",a.age);         ……⑧
        printf ("My height is %f cm.¥n",a.height);     ……⑧
        printf ("My weight is %f kg.¥n",a.weight);     ……⑧

        printf ("Let's calculate my BMI.¥n");

       }
```

①C言語の御約束である.

②strcpy()を使うためのヘッダファイル.

③構造体の宣言.

④構造体メンバの定義.

⑤実行を開始する関数.

⑥宣言した構造体型の変数を定義.

⑦定義した構造体型の変数のメンバへの代入．代入の方法としては主に，このプログラムのように一つ一つメンバに代入していく方法と，データを構造体変数ごとに一括して代入する方法がある．一括して入力する方法はEX73.Cにて解説する.

⑧構造体メンバの内容確認.

構造体メンバの内容を利用するとき，構造体変数名とメンバをドット（.）で連結し，どの変数のどのメンバであるかを明確に示さなければならない.
この例では構造体変数を一つしか定義していないので，実感としてとらえにくいが，構造体は本来，多数の変数を定義する場合に有効となるので，上記の点に注意しなければならない.
次に，構造体変数としてポインタ変数を定義する場合について考える.
以下のプログラムを入力し，実行する.

ファイル名：EX72.C

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

struct data
          {
           char  name[40];
```

```
            char  address[100];
            int   age;
            float height;
            float weight;
           }

main ()
       {
        struct data a;
        struct data *p1;

        strcpy(a.name,"Mr.X");
        strcpy(a.address,"Bunkyo Yushima 1-5-45");
        a.age=20;
        a.height=175.0;
        a.weight=60.5;

        p1=&a;

        printf ("My name is %s.¥n",a.name);
        printf ("I live at %s.¥n",a.address);
        printf ("I am %d years old.¥n",a.age);
        printf ("My height is %f cm.¥n",a.height);
        printf ("My weight is %f kg.¥n",a.weight);

        printf ("¥n");
        printf ("My name is %s.¥n",p1->name);
        printf ("I live at %s.¥n",p1->address);
        printf ("I am %d years old.¥n",p1->age);
        printf ("My height is %f cm.¥n",p1->height);
        printf ("My weight is %f kg.¥n",p1->weight);

        p1->age=30;

        printf ("¥n");
        getch();

        printf ("My name is %s.¥n",a.name);
        printf ("I live at %s.¥n",a.address);
        printf ("I am %d years old.¥n",a.age);
        printf ("My height is %f cm.¥n",a.height);
        printf ("My weight is %f kg.¥n",a.weight);

        printf ("¥n");
        printf ("My name is %s.¥n",p1->name);
        printf ("I live at %s.¥n",p1->address);
```

```
        printf ("I am %d years old.¥n",p1->age);
        printf ("My height is %f cm.¥n",p1->height);
        printf ("My weight is %f kg.¥n",p1->weight);

        printf ("¥n");

        printf ("Let's calculate my BMI.¥n");

       }
```

ポインタ変数を利用する場合，変数名とメンバとの連結をアロー（->）で行なう点に注意する．

最後に，構造体の変数に配列変数を利用する場合について考える．

以下のプログラムを入力し，実行してみる．

ファイル名：EX73.C

```
#include <stdio.h>

struct data
           {
            char  name[40];
            char  address[100];
            int   age;
            float height;
            float weight;
           }

main ()
       {
        int i;
        struct data a[4]={
                          {"Mr.X","Bunkyo Yushima 1-5-45",20,175.0,60.5},
                          {"Dr.W","Chiyoda Kanda 1-1-1",50,163.4,70.5},
                          {"Dr.Z","Bunkyo Hongo 3-2-1",36,143.0,46.3},
                          {"Mrs.X","Bunkyo Yushima 1-5-45",29,120.0,40.5}
                         };

        for (i=0;i<4;i++)
                         {
                          printf ("My name is %s.¥n",a[i].name);
                          printf ("I live at %s.¥n",a[i].address);
                          printf ("I am %d years old.¥n",a[i].age);
```

```
                        printf ("My height is %f cm.¥n",a[i].height);
                        printf ("My weight is %f kg.¥n",a[i].weight);
                        printf ("¥n");
                        getch();
                       }

       printf ("Let's calculate our BMI.¥n");

      }
```

以下に順を追って解説していく.

```
#include <stdio.h>    ·····①

struct data           ·····②
          {
           char  name[40];         ·····③
           char  address[100];     ·····③
           int   age;              ·····③
           float height;           ·····③
           float weight;           ·····③
          }

main ()           ·····④
       {
        int i;
        struct data a[4]={    ·····⑤
                         {"Mr.X","Bunkyo Yushima 1-5-45",20,175.0,60.5},
                         {"Dr.W","Chiyoda Kanda 1-1-1",50,163.4,70.5},
                         {"Dr.Z","Bunkyo Hongo 3-2-1",36,143.0,46.3},
                         {"Mrs.X","Bunkyo Yushima 1-5-45",29,120.0,40.5}
                        };

        for (i=0;i<4;i++)
                        {
                         printf ("My name is %s.¥n",a[i].name);
                         printf ("I live at %s.¥n",a[i].address);
                         printf ("I am %d years old.¥n",a[i].age);
                         printf ("My height is %f cm.¥n",a[i].height);
                         printf ("My weight is %f kg.¥n",a[i].weight);
                         printf ("¥n");
                         getch();
                        }
```

```
    printf ("Let's calculate our BMI.¥n");

  }
```

①C言語の御約束である.

②構造体の宣言.

③構造体メンバの定義.

④実行を開始する関数.

⑤宣言した構造体型の配列変数を定義し，同時にメンバへのデータ入力を行っている.
　データ入力は，一つの構造体変数に含まれるデータを過不足なく{}でくくり，かつそれらのデータ集合を{}でくくっている．データの変数への入力であり，変数定義でもあるので，文末に;（セミコロン）を忘れてはいけない.

　構造体はC++言語のクラスの基本となる概念であり，これを理解せずにC++言語に取り組むことは非常に困難である．可能な限り，独習にて補っておくことが望ましい.

７章の課題

　★構造体を用いて成績表を表示するプログラムを作成せよ．ただし，成績表のデータは，学籍番号・名前・点数・評価(A～D)であるとする.
　　プログラムの完成を確認するために，十人分のダミーデータを設定すること.

　●構造体を用いて住所録を作成せよ．ただし，住所録のデータとして必要なデータは，学籍番号・名前・住所・電話番号・血液型・誕生日であるとする

8章　グラフィック関数

Ｃ言語の特徴の一つとして，関数表記の統一が挙げられる．Ｃ言語以前の言語ではコンパイラもしくはインタプリタを開発するメーカごとに命令語(関数)などの表記が完全に統一されておらず，プログラムの移植に不便であったことを考慮して，Ｃ言語開発時にANSI規格として統一したことによる．しかし，コンピュータのハードウェアを直接操作する場合には，機種ごとの特性を考慮しなければならない．その一つがグラフィック機能である．画面上に点を描くことはビデオRAMを1bit単位で操作することと同義であり，この延長にある描画機能はきわめてハードウェアとの関連性が高い．

以上の理由により，Ｃ言語ではグラフィック関係の関数を規格として定義していない．したがって製品化されている多くのＣ言語コンパイラではメーカ独自のグラフィック関数を定義し，利用できるようにしている．

本実習書で使用しているLSI-C86 Ver.3.30C試食版では，機種依存性をなくすためにグラフィック関数をサポートしていない．それにもかかわらずフリーウェアである利点からＣ言語入門用として評価が高く，各機種に対応したグラフィックライブラリが一部のプログラマの手によりフリーウェアとして提供されている．

そこで，本章以降では小山佳孝氏が作成し，フリーウェアとしてインターネットで配布している「LSI-C試食版用 簡易グラフィックライブラリ 1.093」を使用してグラフィック関数を学ぶ．

上述のグラフィックライブラリはIBM-PC，すなわちDOS/V機と呼ばれるパーソナルコンピュータ向けに作成されており，インターネットを通じて以下のサイトから入手可能である．

http://www.vector.co.jp/soft/dl/dos/prog/se243629.html

8.1　VGAモード

DOS/V機は複数の画面モードを持つ．この中で，モード12hのVGAモードはMS-DOS時代には最も多用された画面モードである．このモードでは文字の場合80文字×30行，グラフィック画面の解像度が横640pixel×縦480pixelとなっており，文字・グラフィック共に16色が使用可能である．モード12hの場合，文字・グラフィックのカラーパレット(色指定番号)が共通であり，以下の表のようになっている．

カラーパレットの1,2,4が光の三原色である青，緑，赤に対応しており，光の混合色に対応したカラーパレットになっている．例えば青と赤の混合色は紫であるが，紫のカラーパレットは青と赤のカラーパレットを加えた5に設定されている．

Table 8.1　カラーパレットと表示色の対応

カラーパレット	表示色	カラーパレット	表示色
0	黒	8	灰色
1	青	9	薄い青
2	緑	10	薄い緑
3	水色	11	薄い水色
4	赤	12	薄い赤
5	紫	13	薄い紫
6	茶(黄)	14	薄い黄
7	白	15	明るい白

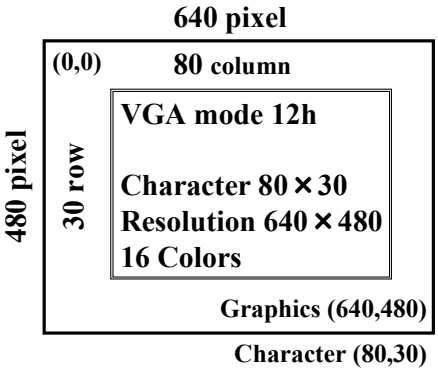


Fig.8.1　モード12hのVGAモード

コンピュータの画面上での座標軸は数学と異なり，左上の隅が原点となる．右下に行くほど*x*座標，*y*座標のそれぞれが増加する．モード12hのVGAモードの場合，右下隅の座標はグラフィック画面で(639,479)，キャラクター画面で(79,29)となる．

8.2　グラフィック関数のコンパイルの方法

グラフィック関数を使用する場合，ソースファイルでGraph.hをincludeし，さらにコンパイル時にGraphics.libをリンクしなければならない．リンクするためには以下のようにソースファイルの後ろにライブラリのファイル名を記述する．

lcc［ソースファイル名］Graphics.lib　　〔Enter〕

しかしながら，コンパイルの都度，Graphics.libと記述するのは面倒であるのでバッチファイル【cc.bat】にはあらかじめこれらの手順が記述してある．このため，ここまでのコンパイルと同様に以下のように記述すればよい．

cc［ソースファイル名］　　〔Enter〕

8.3　グラフィック関数の使用

グラフィック関数を使用するにはいくつかの手続きが必要である．

①Graph.hをincludeする．
②SetGraphicsMode()関数でグラフィックモードを12hに切り替える．
③プログラム終了後，RestoreMode() 関数でグラフィックモードを元の状態に復帰する．

まず以下のプログラムで，この流れを確認する．

ファイル名：EX81.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                                /* Restore Mode */
      }
```

ClearScreen(int col)関数は画面を消去する関数である．colで指定された色を背景色として画面上の文字とグラフィックスを消去する．
グラフィック関数と従来のキャラクター表示関数printf()も併用可能である．
RestoreMode()関数を実行すると，表示画面が消えて元のMS-DOS画面に戻ってしまうので，その直前でgetch()関数を実行し，キー入力待ちとする．

ファイル名：EX82.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       int i;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       for(i=0;i<10;i++)
                      {
                       SetCursorPos(10+i*2,5+i);
                       printf("*");
                      }
```

```
        SetCursorPos(0,23);
        printf ("Push ANYKEY!!");
        getch();
        RestoreMode();                                  /* Restore Mode */

      }
```

SetCursorPos(int x,int y)は文字列を表示する場合の書き出し位置を指定する関数である．横x文字目，縦y文字目，すなわち座標(x,y)にカーソルを移動する．座標指定はキャラクター画面の指定である．この関数はNT系列のWINDOWSでも使用可能である．

ファイル名：EX83.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
        int i;
        char Buf[7]="Color#";

        SetGraphicsMode();                          /* Change Mode 12h */

        ClearScreen(0);

        for(i=0;i<8;i++)
                        {
                          Buf[5]='0'+i;
                          ShowStringC(Buf,10+i*2,5+i,i);
                        }

        SetCursorPos(0,23);
        printf ("Push ANYKEY!!");
        getch();

        RestoreMode();                                  /* Restore Mode */
      }
```

ShowStringC(char Buf,int x, int y,int col)関数はcolで色を指定し，Bufに格納された文字列を座標(x,y)に表示する関数である．座標指定はキャラクター画面の指定で

ある．NT系列のWINDOWSではMS-DOSのエスケープシーケンスを利用する色指定関数が使用出来ない．そこで，必要に応じてこの関数を用いて任意の色で文字を表示する．

ファイル名：EX84.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       int i;
       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       for(i=0;i<400;i++)
                      {
                       PutPoint(i+100,i+20,i%7+1);
                      }

       SetCursorPos(0,23);
       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                                /* Restore Mode */
      }
```

PutPoint(int x, int y,int col)関数はグラフィック画面の任意の座標(x,y)にcolで指定した色の点を表示する．このプログラムでは連続して色を変化しながら点を描画しているので線を描画したように見える．

グラフィック画面で座標を指定する際，x座標は画面に対して左から右に増加し，y座標は上から下に増加する．これは文字を表示する場合の座標指定と同様に考えればよい．

ファイル名：EX85.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
```

```
{
 int i;
 SetGraphicsMode();                          /* Change Mode 12h */

 ClearScreen(0);

 for(i=0;i<400;i+=20)
                    {
                     Line(i,0,0,400-i,i%7+1);
                    }

 SetCursorPos(0,23);
 printf ("Push ANYKEY!!");
 getch();

 RestoreMode();                                /* Restore Mode */
}
```

Line(int x_1,int y_1,int x_2,int y_2,int col)関数は任意の座標(x_1,y_1)から任意の座標(x_2,y_2)までcolで指定した色の線分を描画する関数である．(x_1,y_1)と(x_2,y_2)の大小関係が逆になっても問題なく使用できる．

ファイル名：EX86.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>

main()
{
 int i;
 int x,y;
 double pi=3.1415926;

 SetGraphicsMode();                          /* Change Mode 12h */

 ClearScreen(0);

 for(i=0;i<8;i++)
                {
                 x=320+150*cos(pi/8.0*(i+1)*2);
                 y=240+150*sin(pi/8.0*(i+1)*2);
```

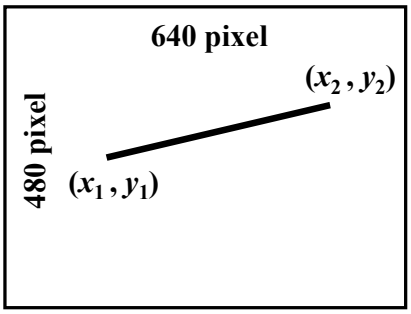
```
                    Circle(x,y,20,i%7+1);
                  }

        SetCursorPos(0,23);
        printf ("Push ANYKEY!!");
        getch();

        RestoreMode();                              /* Restore Mode */
      }
```

Circle(int x, int y,int r,int col)関数は任意の座標(x,y)を中心として，colで指定した色で，半径r (pixel)の真円を描く関数である．内部の塗りつぶしは行わない．

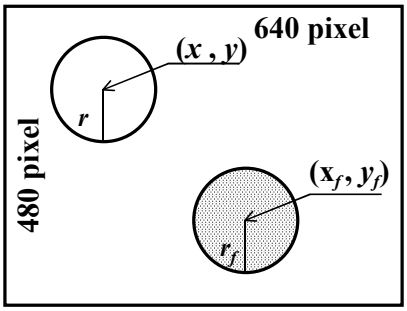
ファイル名：EX87.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>

main()
      {
        int i;
        int x,y;
        double pi=3.1415926;

        SetGraphicsMode();                          /* Change Mode 12h */

        ClearScreen(0);

        for(i=0;i<20;i++)
                    {
                     x=320+10*(i+2)*cos(pi/20.0*(i+1)*2);
                     y=240+10*(i+2)*sin(pi/20.0*(i+1)*2);
                     FillCircle(x,y,20,i%7+1);
                    }

        SetCursorPos(0,23);
        printf ("Push ANYKEY!!");
        getch();

        RestoreMode();                              /* Restore Mode */
      }
```

FillCircle(int x_f, int y_f, int r_f, int col_f)関数は任意の座標(x_f, y_f)を中心として，col_fで指定した色で，半径r_f (pixel)の真円を描く関数である．内部もcol_fで指定した色で塗りつぶしを行なう．



Line(int x_1, int y_1, int x_2, int y_2, int *col*)



Circle(int *x*, int *y*, int *r*,int *col*)
FillCircle(int x_f, int y_f, int r_f, int col_f)

Fig.8.2　Line関数とCircle，FillCircle関数

ファイル名：EX88.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       int i;
       int x1,y1,x2,y2;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       for(i=0;i<20;i++)
                        {
                         x1=320-10*(i+1);
                         y1=240-10*(i+1);
                         x2=320+10*(i+1);
                         y2=240+10*(i+1);
                         Rectangle(x1,y1,x2,y2,i%7+1);
                        }
```

```
        SetCursorPos(0,23);
        printf ("Push ANYKEY!!");
        getch();
        RestoreMode();                                  /* Restore Mode */

      }
```

Rectangle(int x_1,int y_1,int x_2,int y_2,int col)関数は左上角の点を任意の座標(x_1,y_1)で，右下角の点を任意の座標(x_2,y_2)で指定し，colで指定した色で長方形を描く．内部の塗りつぶしは行わない．

ファイル名：EX89.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
        int i;
        int x1,y1,x2,y2;

        SetGraphicsMode();                          /* Change Mode 12h */

        ClearScreen(0);

        for(i=0;i<20;i++)
                        {
                         x1=i*5;
                         y1=i*10;
                         x2=640-30*(i+1);
                         y2=480-20*(i+1);
                         FillRectangle(x1,y1,x2,y2,i%7+1);
                        }

        SetCursorPos(0,23);
        printf ("Push ANYKEY!!");
        getch();

        RestoreMode();                                  /* Restore Mode */
      }
```

FillRectangle(int x_{f1}, int y_{f1}, int x_{f2}, int y_{f2}, int col_f)関数は左上角の点を任意の座標(x_{f1}, y_{f1})で，右下角の点を任意の座標(x_{f2}, y_{f2})で指定し，col_fで指定した色で長方形を描く．内部もcol_fで指定した色で塗りつぶしを行なう．

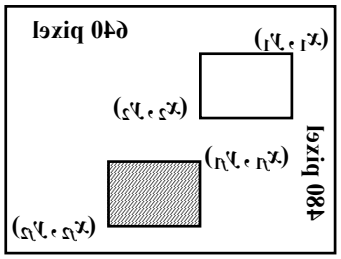
ファイル名：EX8A.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       int i;
       int x1,y1,x2,y2,x3,y3;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       for(i=0;i<14;i++)
                      {
                       x1=i*5;
                       y1=i*10;
                       x2=640-40*(i+1);
                       y2=480-20*(i+1);
                       x3=x1;
                       y3=y2;
                       Triangle(x1,y1,x2,y2,x3,y3,i%7+1);
                      }

       SetCursorPos(0,23);
       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                                /* Restore Mode */
      }
```

Triangle(int x_1, int y_1, int x_2, int y_2, int x_3, int y_3, int col)関数は3点(x_1, y_1), (x_2, y_2), (x_3, y_3)を頂点とする三角形を，colで指定した色で描く関数である．内部の塗りつぶしは行わない．

ファイル名：EX8B.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       int i;
       int x1,y1,x2,y2,x3,y3;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       for(i=0;i<14;i++)
                        {
                         x1=320;
                         y1=i*10;
                         x2=640-20*i;
                         y2=480-5*i;
                         x3=20*i;
                         y3=y2;
                         FillTriangle(x1,y1,x2,y2,x3,y3,i%7+1);
                        }

       SetCursorPos(0,23);
       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                                /* Restore Mode */
      }
```



Rectangle(int x_1, int y_1, int x_2, int y_2, int col)
FillRectangle(int x_1, int y_1, int x_2, int y_2, int col)

Triangle(int x_1, int y_1, int x_2, int y_2, int x_3, int y_3, int *col*)
FillTriangle(int x_{f1}, int y_{f1}, int x_{f2}, int y_{f2}, int x_{f3}, int y_{f3}, int col_f)

Fig.8.3 Rectangle, FillRectangle関数とTriangle, FillTriangle関数

FillTriangle(int x_{f1}, int y_{f1}, int x_{f2}, int y_{f2}, int x_{f3}, int y_{f3}, int col_f)関数は3点(x_{f1}, y_{f1}), (x_{f2}, y_{f2}), (x_{f3}, y_{f3})を頂点とする三角形を, col_fで指定した色で描く関数であ

る．内部もcol_fで指定した色で塗りつぶしを行なう．

ファイル名：EX8C.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       Triangle(320,50,200,290,360,70,1);
       Circle(320,240,180,4);
       Rectangle (120,60,500,400,2);
       Line(0,20,600,380,6);
       Fill (320,240,7);

       SetCursorPos(0,23);
       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                              /* Restore Mode */
      }
```

Fill (int x,int y,int col)関数は座標(x,y)で指定される点の色以外を境界色として，colで指定した色で境界の内部を塗りつぶす関数である．

以上の関数がグラフィック関数の基本的なものである．他にもグラフィック関数は用意されているが，それらを使わなくとも事足りることが多いので，本書では取り上げない．

これらの関数を用いたコンピュータグラフィックスの応用例を以下にとりあげる．単純なグラフィック関数も使い方しだいであることが理解できると思う．

ファイル名：EX8D.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>
```

```
#include <time.h>


#define CX   320
#define CY   200

main()
      {
       int i,j,r,col;
       int x[65],y[65];
       long nowtime,n;
       double rd;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

                                   time (&nowtime);
                                   n=nowtime%30+5;
                                   rd=2*PI/n;
                                   r=200;
                                   col=n%7+1;

        for (i=0;i<n;i++)
                        {
                         x[i]=CX+r*cos (rd*i);
                         y[i]=CY+r*sin (rd*i);
                        }
        for (i=0;i<n;i++)
                        {
                         time (&nowtime);
                         col=nowtime%7+1;
                         for (j=i;j<n;j++)
                                        {
                                      Line (x[i],y[i],x[j],y[j],col);
                                        }
                        }

       SetCursorPos(0,23);
       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                              /* Restore Mode */
      }
```

このプログラムは任意の多角形を描き，その対角線を全て描くものである．コンピュータの内部時計によって，5～34角形まで任意に決定し，色も7色の中から任意に選択されるので，様々なパターンの多角形が描かれる．

なお，defineはプログラム中で不変の定数などを定義するときに用いる．

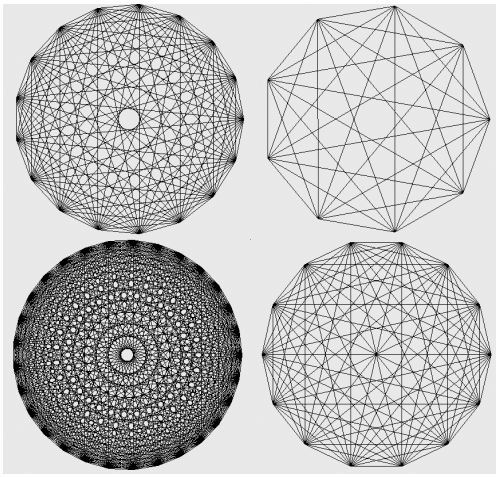


Fig.8.4　対角線を描いた多角形

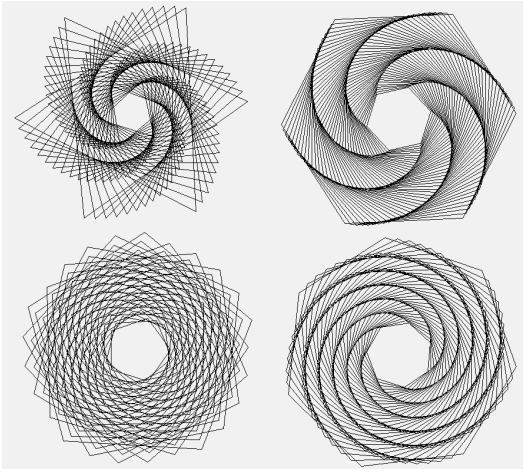


Fig.8.5　回転しながら縮小する多角形

ファイル名：EX8E.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>


#define CX   320
#define CY   200

main()
      {
       int i,r,col,count,cnt;
       int x[40],y[40];
       long nowtime,n;
       double rd;

       cnt=10;

       SetGraphicsMode();                    /* Change Mode 12h */
```

```
                        while (cnt)
                                {
                                ClearScreen(0);
                                time (&nowtime);
                                n=nowtime%5+3;
                                rd=2*PI/n;
                                r=203;
                                col=n%7+1;
for (count=1;count<50;count++)
                                {
                                r-=3;

    for (i=0;i<n+1;i++)
                                {
                                x[i]=CX+r*cos (rd*i+count);
                                y[i]=CY+r*sin (rd*i+count);
                                }
    for (i=0;i<n;i++)
                                {
                                time (&nowtime);
                                col=nowtime%7+1;
                                Line (x[i],y[i],x[i+1],y[i+1],col);
                                }

                                }
     cnt--;
     getch();
                                }

    SetCursorPos(0,23);
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
  }
```

これは任意の多角形を，大きさを一定の割合で縮小しながら，一定の割合で回転させて連続的に描くプログラムである．10パターン描いてプログラムは終了する．多角形の形と色はパソコンの内部時刻によって決定するので，どのようなパターンが描画されるかは実行しないとわからない．

なお，プログラム中の囲った部分はEX8D.Cと異なる部分である．

ファイル名：EX8F.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

main()
      {
       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       FillRectangle(210,396,430,426,4);
       FillCircle(210,411,15,4);
       FillCircle(430,411,15,4);
       getch();

       FillCircle(320,200,200,1);
       FillCircle(320,240,160,7);
       FillCircle(320,250,100,4);
       FillRectangle(210,140,430,250,7);
       getch();

       FillCircle(290,80,30,7);
       FillCircle(293,80,7,0);
       Circle(290,80,30,0);
       FillCircle(350,80,30,7);
       FillCircle(347,80,7,0);
       Circle(350,80,30,0);
       FillCircle(320,120,20,4);
       getch();

       Line (320,140,320,250,0);
       Line (260,160,100,130,5);
       Line (260,180, 90,180,5);
       Line (260,200,100,230,5);
       Line (380,160,540,130,5);
       Line (380,180,550,180,5);
       Line (380,200,540,230,5);
       getch();

       FillCircle(320,410,10,2);
       Line(310,410,330,410,0);
       Line(320,410,320,420,0);
       Circle(320,410,10,6);
```

```
    SetCursorPos(60,2);
    printf ("Do You Know Me?");
    getch();

    SetCursorPos(0,23);
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
}
```

これは世界的に有名な和製キャラクターを描くプログラムである．キーを押すごとに各部分が描かれていく．

プログラムを実行しながら，どの関数がどの部分を描いているかを確認する．

ところで，本書ではVGAモードのみに限定してグラフィックライブラリを使用したが，提供されているライブラリは様々なモードに対応しており，さらに高度なグラフィック処理関数も用意されている．

解説などを参考に，ライブラリの利用法について研究し使いこなせは，市販のライブラリと比較して遜色のないプログラムが可能である．

８章の課題

▲EX56.C，EX57.C，EX58.Cのプログラムは縦描きであるが，SetCursorPos関数を用いて，これらを横描きせよ．

★EX8F.Cで描いたキャラクターの全身像を描け．

●EX8F.Cで描いたキャラクターの表情を変化させよ．

9章　グラフ描画

一般に演算や計測の結果を数字で表示しても，それが正確なものであるかを判断するのは困難であることが多い．しかし，これらのデータ間に存在する相関関係を図示すると，その傾向などから正否が直感的に判断できるようになる．したがって実験によって得られた結果や演算処理の結果をグラフ化することは古くから行われてきた．

データがすでに得られている場合，これをグラフ化するためのソフトウェアは様々なものが存在しており，その利用によって報告書などの作成は容易に行える．しかしながら，実験やシミュレーションの演算などの成否を確認したい場合，これらのソフトウェアの利用は，無用の時間遅れを生じる．すなわち，進行中の実験が終了するまでその成否が確認できないので，長時間のプロセスの場合，リアルタイムでの状態を表現するものとはなりえない．

したがって，コンピュータを利用した計測・演算と同時進行でグラフを描画することは極めて有効であり，本章では８章で学習したグラフィック関数を用いたグラフ化の手法について解説する．

9.1　描画範囲の設定と座標軸変換

一般にグラフを描画する場合，その領域を設定しなければならない．コンピュータによってリアルタイムな描画を行う場合，同時にその値の妥当性を検討しなければならないことが多いので，領域外に数値を表示することを前提として描画範囲を設定する．

ところで，コンピュータでグラフを描画するときに気をつけなければいけないのは原点の移動，座標軸の倍率，*y*軸の向きの3点である．特に，コンピュータの*y*座標は下向きに増加として考えるが，グラフは上向きに増加するのが一般的である．

これらの点を解決するために座標変換式を構築し，プログラムすると便利である．詳しくは以下の例題を参考に述べる．

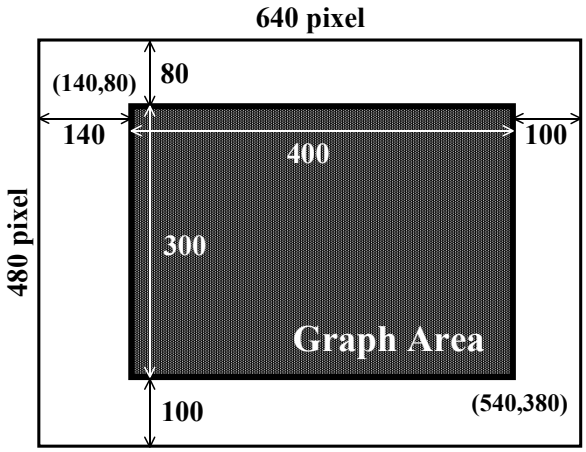


Fig.9.1　グラフ領域の設定

9.2　数学関数のグラフ化

ファイル名：EX91.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

trans_axis(double x, double y)
                              {
                               int tx,ty;

                               tx=x*20+340;
                               ty=y*(-10)+230;

                  if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                             {
                               PutPoint(tx,ty,4);
                             }

                               }

double func(double x)
                     {
                      double x1,x2,x3,y;

                      x1=x-1;
                      x2=x+1;
                      x3=x-2;
                      y=x1*x2*x3;

                      return(y);

                     }

main()
      {
       double x,y;
       int i;

       SetGraphicsMode();                              /* Change Mode 12h */

       ClearScreen(0);
       FillRectangle(140,80,540,380,7);
       Line(140,230,540,230,1);
       Line(340,80,340,380,1);

       for (i=-1000;i<1100;i++)
                               {
```

```
                x=i/100.0;
                y=func(x);
                trans_axis(x,y);
               }


    SetCursorPos(0,28);
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                              /* Restore Mode */
   }
```

このプログラムは3次関数を例として描画するプログラムである．*x*を-10.0から11.0まで0.01刻みで変化し，これに対応する*y*を関数func(double x)で算出している．

グラフ領域の中心を原点とし，関数trans_axis(double x, double y)で*x*軸および*y*軸ごとに適切な倍率を設定して，対応する座標に点を描画している．関数内で*x*, *y*に倍率を掛けると同時に，バイアスを加えて原点移動も同時に行っている．*y*の倍率を負の数にして，描画する際の座標軸の向きを実際のグラフと合わせている．

また，グラフ領域からはずれた場合には描画を行わないように条件判断を行う．

これが座標変換によるグラフ描画の基本である．

ファイル名：EX92.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>

trans_axis(double x, double y, int col)
                {
                 int tx,ty;

                 tx=x*2+340;
                 ty=y*(-80)+230;

          if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                    {
                     Circle(tx,ty,2,col);
                    }

                 }
```

```
main()
      {
       double x,y1,y2,y3,pi;
       int i;

       pi=3.1415926;

       SetGraphicsMode();                              /* Change Mode 12h */

       ClearScreen(0);
       FillRectangle(140,80,540,380,7);
       Line(140,230,540,230,1);
       Line(340,80,340,380,1);

       for (i=-100;i<100;i++)
                              {
                               x=pi*2*i/100.0;
                               y1=sin(x);
                               y2=cos(x);
                               y3=0.0;
                  if (i%25!=0)
                               {
                                y3=tan(x);
                               }

                               trans_axis(i,y3,0);
                               trans_axis(i,y2,4);
                               trans_axis(i,y1,2);
                              }


       SetCursorPos(0,28);
       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                                    /* Restore Mode */
      }
```

このプログラムではxが-2πから2πまで変化した場合のsin x, cos x, tan xを同時に描画する．前のプログラムでは点を描画していたのに対して，このプログラムでは円を描いている．点の連続により作成した線の表示は連続的に変化するグラフを描く場合に適している．ところでこのプログラムの場合，x軸が400pixelに対して，描画するポイントが200であり，最大でも1pixelおきに描画することになる．半径をもつ円で表示すると，描画されたのがはっきりと認識できるので，変化の程度が連続と言えない場合などに適している．

ファイル名：EX93.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>

trans_axis(double x, double y, int col)
                              {
                               int tx,ty;

                               tx=x+340;
                               ty=y*(-20)+230;

                    if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                              {
                                Circle(tx,ty,2,col);
                              }

                               }

main()
      {
       double x,y;
       int i;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);
       FillRectangle(140,80,540,380,7);
       Line(140,230,540,230,1);
       Line(340,80,340,380,1);

       for (i=1;i<200;i++)
                               {
                                x=i;
                                y=log(x);
                                trans_axis(x,y,2);
                               }


       SetCursorPos(0,28);
       printf ("Push ANYKEY!!");
       getch();
```

```
       RestoreMode();                                    /* Restore Mode */
      }
```

このプログラムでは*x*座標を1刻みで1から199までの199個の点を与えて計算している．これに対して画面上の対応する座標も199pixelであるので連続的に表示できる．また円で描画しているので太い線で描いたように見える．

ファイル名：EX94.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>

trans_axis(double x, double y, int col)
                              {
                               int tx,ty;

                               tx=x*100+340;
                               ty=y*(-20)+230;

                  if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                             {
                               Circle(tx,ty,2,col);
                             }

                               }

main()
      {
       double x,y;
       int i;

       SetGraphicsMode();                              /* Change Mode 12h */

       ClearScreen(0);
       FillRectangle(140,80,540,380,7);
       Line(140,230,540,230,1);
       Line(340,80,340,380,1);

       for (i=-200;i<200;i++)
                               {
```

```
            x=i/100.0;
            y=exp(x);
            trans_axis(x,y,2);
          }


    SetCursorPos(0,28);
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
  }
```

このプログラムは*x*座標を0.01刻みで-2から2まで与えて計算している．このようにあらかじめ*x*の範囲がわかっている場合，刻み幅を調節してグラフの*x*軸のpixel数と合わせると適切なグラフの描画が出来る．しかし，*y*座標は演算の結果であるので常に分かっているとは限らない．したがって，適切と思われる範囲を設定して描画するのでグラフ領域を有効に使うかは実行するまで分からない．

9.3　棒グラフ・帯グラフ・円グラフ

上記4つのプログラムのように関数が分かっている場合には，あらかじめ計算しておくことで適切な描画範囲を与えることもできる．しかし，演算結果が分からない，もしくは複雑な計算を必要とするので演算結果を予測できないものをプログラムする場合も少なくない．このような場合，グラフの描画範囲をあらかじめ決めてしまうと予想外の数値を得た場合に不適切な描画を行う可能性がある．

グラフを描く目的の一つとして，このような予想外の点が得られた場合，それが測定ミスなどから生じたものか，適切なものであるのか判定することが挙げられるので，予想外の点が描画されないのは好ましくない．

そこで，以下では自動的に描画範囲を調節してグラフを描画する．

以下の2つのプログラムでは，2個のサイコロを投げて出た目の和の回数とその確率についてシミュレーションし，試行回数が増加するほど理論値に近づいていくことを確認する．

Table 9.1　２つのサイコロの目の和と起こりうる確率

目の和	組合せの数	確率	目の和	組合せの数	確率
2	1	1/36	8	5	5/36
3	2	2/36	9	4	4/36
4	3	3/36	10	3	3/36
5	4	4/36	11	2	2/36
6	5	5/36	12	1	1/36
7	6	6/36			

ファイル名：EX95.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


gra_draw()
          {
           int i,dx;

           ClearScreen(0);
           Rectangle(140,80,540,380,7);

           for(i=2;i<13;i++)
                          {
                           dx=i*400/13/8+18;
                           SetCursorPos(dx,24);
                           printf("%d",i);
                          }

           }

draw_point(double rymax,int x[],double rate[])
          {
           int gx,gy,i;
           int col[]={0,1,2,3,4,5,6,8,9,10,11,12,13};
           double rx1,rx2;

           SetCursorPos(10,5);
           printf("%5.1lf",rymax);
           rx1=rx2=140.0;

           for (i=2;i<13;i++)
                          {
                           gx=i/13.0*400+140;
                           gy=x[i]/rymax*(-300)+380;
                           FillRectangle(gx,gy,gx+10,380,col[i]);

                           rx2=400.0*rate[i]+rx1;
                           FillRectangle(rx1,30,rx2,40,col[i]);
                           rx1=rx2;
```

```
                        }
    }

main()
    {
     int    x[13],i,d1,d2,sum,cnt;
     float  f;
     unsigned long time1;
     unsigned seed;
     double rymax;
     double rate[13];

     SetGraphicsMode();                          /* Change Mode 12h */

     ClearScreen(0);

     rymax=3.0;
     cnt=0;

     time(&time1);
     seed=time1;
     srand(seed);

     for(i=0;i<13;i++)
                 {
                  x[i]=0;
                  rate[i]=0.0;
                 }

     while (cnt<10000)
           {
            f=rand();
            d1=f*6/32768.0+1;
            f=rand();
            d2=f*6/32768.0+1;
            sum=d1+d2;

            x[0]++;
            x[sum]++;
            rate[0]=0.0;

            if (x[sum]>(rymax-1))
                        {
                         rymax*=1.2;
                        }
```

```
                for (i=1;i<13;i++)
                                {
                                 rate[i]=(double)x[i]/x[0];
                                 rate[0]+=rate[i];
                                }

                 gra_draw();
                 draw_point(rymax,x,rate);
                 cnt++;
                 SetCursorPos(5,28);
                 printf("COUNT:%d",cnt);

                 getch();

                }

      printf ("Push ANYKEY!!");
      getch();

      RestoreMode();                                    /* Restore Mode */
     }
```

コンピュータでサイコロをシミュレーションする場合，乱数を用いる．サイコロに何の問題もない場合，出る目の確率は一様に等しいので1から6までの乱数を発生すればよい．変数d1，d2がそれぞれ出た目の数字であり，これらの合計が変数sumである．

合計値のカウントを1増加し，その回数が(現在設定されている最大値-1)をこえたら，現在設定されている最大値を1.2倍する．

適当なキーを押すたびに試行を行う．試行回数が多くなると正規分布に近づいてくる．棒グラフの上に表示されるのが，出た目の和の確率を表す帯グラフである．帯グラフは，各確率に帯グラフの長さを掛け，継ぎ足していくように描いている．

ところで，本プログラム中でprintf関数の変換指示記号%lfを%5.1lfとして使用している部分がある．変換指示記号に含まれる5.1のうち実数部は最少フィールド幅を表す．この場合，5桁分の表示幅を確保し，表示する文字数が少ない場合，左からスペースを挿入して右詰め表示をする．桁数が多い場合は，その桁数に合わせる．小数部は，表示する数字の小数点以下の桁数で，桁数が不足する場合0を補う．

以下のプログラムではEX95.Cとの差異を囲いで示す．

ファイル名：EX96.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
```

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


gra_draw()
          {
           int i,dx;

           ClearScreen(0);
           Rectangle(140,80,540,380,7);

           for(i=2;i<13;i++)
                           {
                            dx=i*400/13/8+18;
                            SetCursorPos(dx,24);
                            printf("%d",i);
                           }

           }

draw_point(double rymax,int x[],double rate[])
          {
           int gx,gy,i,rx,ry;
           int col[]={0,1,2,3,4,5,6,8,9,10,11,12,13};
           double pi=3.1415926;
           double th1,th2,theta;

           SetCursorPos(10,5);
           printf("%5.1lf",rymax);
           th1=th2=0;

           for (i=2;i<13;i++)
                           {
                            gx=i/13.0*400+140;
                            gy=x[i]/rymax*(-300)+380;
                            FillRectangle(gx,gy,gx+10,380,col[i]);

                            th2+=360.0*rate[i];
                            if (rate[i]!=0.0)
                                           {
                            for (theta=th1;theta<th2;theta++)
                                                          {
                            rx=sin(theta*2.0*pi/360.0)*40+590;
                            ry=cos(theta*2.0*pi/360.0)*40+230;
                            Line(590,230,rx,ry,col[i]);
```

```
                                                    }
                                                }
                            th1=th2;
                        }
          Circle(590,230,40,15);
        }


main()
      {
       int    x[13],i,d1,d2,sum,cnt;
       float  f;
       unsigned long time1;
       unsigned seed;
       double rymax;
       double rate[13];

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       rymax=3.0;
       cnt=0;

       time(&time1);
       seed=time1;
       srand(seed);

       for(i=0;i<13;i++)
                       {
                        x[i]=0;
                        rate[i]=0.0;
                       }

       while (cnt<10000)
                 {
                  f=rand();
                  d1=f*6/32768.0+1;
                  f=rand();
                  d2=f*6/32768.0+1;
                  sum=d1+d2;

                  x[0]++;
                  x[sum]++;
                  rate[0]=0.0;
```

```
            if (x[sum]>(rymax-1))
                            {
                             rymax*=1.2;
                            }

            for (i=1;i<13;i++)
                             {
                              rate[i]=(double)x[i]/x[0];
                              rate[0]+=rate[i];
                             }

             gra_draw();
             draw_point(rymax,x,rate);
             cnt++;
             SetCursorPos(5,28);
             printf("COUNT:%d",cnt);

             getch();

            }

     printf ("Push ANYKEY!!");
     getch();

     RestoreMode();                                  /* Restore Mode */
    }
```

このプログラムは前のプログラムの帯グラフを円グラフとしたものである．扇形を描く関数がないので，円の中心から円周に対して線を連続的に描く手法で円グラフを描いている．

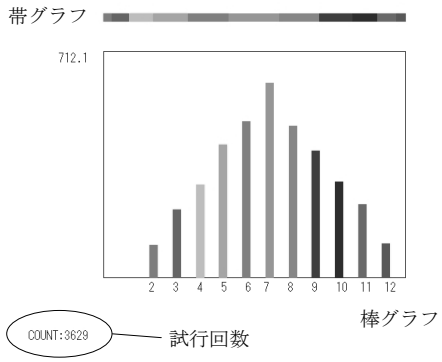


Fig.9.2　棒グラフと帯グラフ

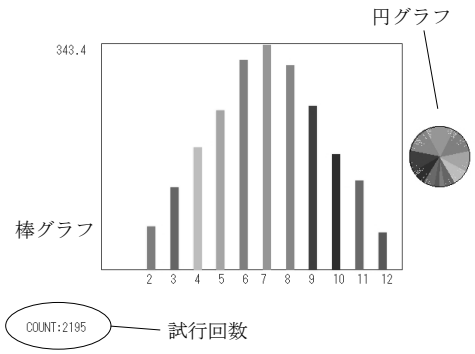


Fig.9.3　棒グラフと円グラフ

9.4　折れ線グラフ

時系列変化を見る時多用される折れ線グラフは，上述してきたグラフと異なり2点を線で接続する必要がある．BLOOD.DATに記録されている血液検査データを折れ線グラフで表示するプログラムを作成する．

ファイル名：EX97.C

```
#include "Graph.h"
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <math.h>


gra_draw()
          {
           ClearScreen(0);
           FillRectangle(140,80,540,380,15);
          }

draw_point(char sn[10],int cnt,double x[100][7])
          {
           int    i,j,gx,gy,bx,by,sl,n,dx,dy,y1,y2,y3,x1;
           int    sx,ex,sy,ey;
           double hy,yr,xr,xmax,xmin,ymax,ymin;
           double rxmax,rxmin,rymax,rymin;
           int col[]={0,1,5,6,8,9,12,13};

           gra_draw();

           xmin=rxmin=-1.0;
           xmax=rxmax=cnt+1;
           ymin=1000.0;
           ymax=0.0;

           sl=strlen(sn);

           for (i=0;i<sl;i++)
                          {
                n=sn[i]-'0';
                if (n>=1 && n<=6)
                               {
                           for (j=0;j<cnt;j++)
                                          {
                               if (ymax<x[j][n])
```

```
                                        {
                                         ymax=x[j][n];
                                        }
                        if (ymin>x[j][n])
                                        {
                                         ymin=x[j][n];
                                        }
                                      }
                    }
                }

yr=ymax-ymin;
dy=log10(yr);

     hy=yr*0.1;
     y1=hy/(pow(10,dy-1));
     y2=ymax*10.0/(pow(10,dy));
     y3=ymin*10.0/(pow(10,dy));
     rymax=(y2+y1)*pow(10,dy-1);
     rymin=(y3-y1)*pow(10,dy-1);
     xr=rxmax-rxmin;
     yr=rymax-rymin;

SetCursorPos(16,24);
printf("%5.1lf",rxmin);
SetCursorPos(63,24);
printf("%5.1lf",rxmax);
SetCursorPos(10,23);
printf("%5.1lf",rymin);
SetCursorPos(10,5);
printf("%5.1lf",rymax);

dx=log10(xr);
dy=log10(yr);
sx=rxmin/(pow(10,dx));
ex=rxmax/(pow(10,dx));
sy=rymin/(pow(10,dy));
ey=rymax/(pow(10,dy));

for (i=sx;i<ex+1;i++)
                    {
                     x1=(i*pow(10,dx)-rxmin)/xr*400+140;
                     if (x1>140 && x1<540)
                                        {
                              Line(x1,80,x1,380,3);
                                        }
```

```
                        }
        for (i=sy;i<ey+1;i++)
                        {
                         y1=(i*pow(10,dy)-rymin)/yr*(-300)+380;
                         if (y1>80 && y1<380)
                                          {
                                  Line(140,y1,540,y1,3);
                                          }
                        }

        for (i=0;i<sl;i++)
                       {
             n=sn[i]-'0';
             if (n>=1 && n<=6)
                           {
                  Line (0,(i*2+10)*16-8,80,(i*2+10)*16-8,col[n]);

                        gx=(0-rxmin)/xr*400+140;
                        gy=(x[0][n]-rymin)/yr*(-300)+380;

                        Circle(gx,gy,3,col[n]);

                         for (j=1;j<cnt;j++)
                                          {
                        bx=gx;
                        by=gy;

                        gx=(j-rxmin)/xr*400+140;
                        gy=(x[j][n]-rymin)/yr*(-300)+380;

                        Circle(gx,gy,3,col[n]);
                        Line(bx,by,gx,gy,col[n]);

                                          }
                            }
                        }
         }

main()
      {
       int cnt,i,sl,n;
       char c1[100],c2[100],c3[100],c4[100],c5[100],c6[100],c7[100];
       char day[100][10],d[10];
       char sn[10];
       double x[100][7],x1,x2,x3,x4,x5,x6;
       FILE *fp;
```

```
    SetGraphicsMode();                          /* Change Mode 12h */

    ClearScreen(0);

    x1=x2=x3=x4=x5=x6=0.0;
    cnt=0;

    if ((fp=fopen("BLOOD.DAT","r"))==NULL)
                                        {
                                         printf("Cannot Open File!");
                                        }
    else
        {
         fscanf(fp,"%s%s%s%s%s%s%s",c1,c2,c3,c4,c5,c6,c7);
         printf("%6s  %6s %6s %6s %6s %6s %6s¥n",c1,c2,c3,c4,c5,c6,c7);

  while(fscanf(fp,"%s%lf%lf%lf%lf%lf%lf",d,&x1,&x2,&x3,&x4,&x5,&x6)!=EOF)
                                                    {
                        for (i=0;i<7;i++)
                                        {
                               day[cnt][i]=d[i];
                                        }

                        x[cnt][1]=x1;
                        x[cnt][2]=x2;
                        x[cnt][3]=x3;
                        x[cnt][4]=x4;
                        x[cnt][5]=x5;
                        x[cnt][6]=x6;


printf("%s   %5.1lf   %5.1lf   %5.1lf   %5.1lf   %5.1lf   %5.1lf¥n",day[cnt],x
[cnt][1],x[cnt][2],x[cnt][3],x[cnt][4],x[cnt][5],x[cnt][6]);

                        cnt++;
                                                  }

          SetCursorPos(0,28);
          printf("Select any Number(1-6):");
          scanf("%s",sn);
          draw_point(sn,cnt,x);

          sl=strlen(sn);

        for (i=0;i<sl;i++)
```

```
                    {
            n=sn[i]-'0';
            if (n>=1 && n<=6)
                        {
                SetCursorPos(0,i*2+10);
                        switch (n)
                                {
                    case 1:
                        printf("%s¥n",c2);
                        break;
                    case 2:
                        printf("%s¥n",c3);
                        break;
                    case 3:
                        printf("%s¥n",c4);
                        break;
                    case 4:
                        printf("%s¥n",c5);
                        break;
                    case 5:
                        printf("%s¥n",c6);
                        break;
                    case 6:
                        printf("%s¥n",c7);
                        break;
                    default:
                        break;
                                }
                    }
                }

    }

  SetCursorPos(0,28);
  printf ("Push ANYKEY!!");
  getch();

  RestoreMode();                                    /* Restore Mode */
}
```

グラフを描く時は繰り返しによって同じ処理を施すのが一般的である．*n*個のポイントを接続して折れ線グラフを描く場合，まず1番目のポイントを描画し，その座標を別の変数に移して保存する．2番目以降のポイントを描く時，同時に直前のポイントと接続する直線を描画する．

プログラムを実行するとBLOOD.DATの内容が画面に表示されるので，1から6までの任

意の数字を入力する．一番上に表示される項目の年月日を除いて，左から順に対応している．このとき，同時に表示する項目を連続して入力する．例えば，1番目の項目を表示したい場合，1[Enter]と入力する．1・4・5番目の項目を表示したい場合，145[Enter] と入力する．1から6以外の数字を入力しても無視されるが，グラフ左側に表示される項目名に空白が出来る．

このプログラムでは*y*座標の範囲を，描画するデータの最大値と最小値から決定している．また，グラフ中に描いた座標の目盛軸の刻み幅もデータの最大値と最小値から決定する．例えば，最大値と最小値の差が2桁であれば，刻み幅は10として目盛軸を描く．

9.5　散布図

x-*y*座標を設定したグラフ領域の指定座標に，ポイントのみ描画したものを一般には散布図と言う．実験などでデータを採取後，ただちに散布図を描くと測定ミスか否かを判断する一つの材料になる．しかし，測定値の範囲が予測できない場合など一通りの測定が終わるまでグラフ化できないことが多い．

そこで，データ入力ごとに描画範囲の自動修正を行いながらグラフ化するプログラムを作成する．

ファイル名：EX98.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>


gra_draw(double rxmax,double rxmin,double rymax, double rymin)
          {
           int i,sx,ex,sy,ey,dx,dy,x1,y1;
           double rx,ry;

           ClearScreen(0);
           FillRectangle(140,80,540,380,15);

           rx=rxmax-rxmin;
           ry=rymax-rymin;

           dx=log10(rx);
           dy=log10(ry);

           sx=rxmin/(pow(10,dx));
           ex=rxmax/(pow(10,dx));
           sy=rymin/(pow(10,dy));
```

```
            ey=rymax/(pow(10,dy));

            for (i=sx;i<ex+1;i++)
                                {
                                 x1=(i*pow(10,dx)-rxmin)/rx*400+140;
                                 if (x1>140 && x1<540)
                                                        {
                                          Line(x1,80,x1,380,3);
                                                        }
                                }
            for (i=sy;i<ey+1;i++)
                                {
                                 y1=(i*pow(10,dy)-rymin)/ry*(-300)+380;
                                 if (y1>80 && y1<380)
                                                        {
                                          Line(140,y1,540,y1,3);
                                                        }
                                }
          }

draw_point(double rxmax,double rxmin,double rymax,double rymin,int cnt,double
x[],double y[])
          {
            int    i,gx,gy;
            double xr,yr;
            int col[]={0,1,4,5,6};

            xr=rxmax-rxmin;
            yr=rymax-rymin;

            SetCursorPos(16,24);
            printf("%5.1lf",rxmin);
            SetCursorPos(63,24);
            printf("%5.1lf",rxmax);
            SetCursorPos(10,23);
            printf("%5.1lf",rymin);
            SetCursorPos(10,5);
            printf("%5.1lf",rymax);

            for (i=0;i<cnt;i++)
                               {
                                gx=(x[i]-rxmin)/xr*400+140;
                                gy=(y[i]-rymin)/yr*(-300)+380;

                                Circle(gx,gy,3,col[i%5]);
                               }
```

```
        }


main()
      {
       int n,cnt,dx,dy,x1,y1,x2,y2,x3,y3;
       double xmax,xmin,ymax,ymin;
       double rxmax,rxmin,rymax,rymin,rx,ry;
       double hx,hy;
       double x[100],y[100];

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       xmax=xmin=ymax=ymin=0.0;
       n=0;
       cnt=n+1;

       while (-1)
              {
               SetCursorPos(0,26);
               printf ("x[%2d]=",n);
               scanf("%lf",&x[n]);
               printf ("y[%2d]=",n);
               scanf("%lf",&y[n]);
               printf ("¥n");

               if (n==0)
                        {
                         xmax=xmin=x[n];
                         ymax=ymin=y[n];
                         rxmax=(int)xmax+1.0;
                         rxmin=(int)xmin-1.0;
                         rymax=(int)ymax+1.0;
                         rymin=(int)ymin-1.0;
                        }
               else
                        {
                         if (xmax<x[n])
                                       {
                                        xmax=x[n];
                                       }
                         if (xmin>x[n])
                                       {
                                        xmin=x[n];
```

```
                    }
            if (ymax<y[n])
                    {
                     ymax=y[n];
                    }
            if (ymin>y[n])
                    {
                     ymin=y[n];
                    }

    if (xmax==xmin)
                  {
                   xmax+=1.0;
                   xmin-=1.0;
                  }
    if (ymax==ymin)
                  {
                   ymax+=1.0;
                   ymin-=1.0;
                  }

    rx=xmax-xmin;
    ry=ymax-ymin;

    dx=log10(rx);
    dy=log10(ry);

    hx=rx*0.1;
    x1=hx/(pow(10,dx-1));
    x2=xmax*10.0/(pow(10,dx));
    x3=xmin*10.0/(pow(10,dx));
    rxmax=(x2+x1)*pow(10,dx-1);
    rxmin=(x3-x1)*pow(10,dx-1);

    hy=ry*0.1;
    y1=hy/(pow(10,dy-1));
    y2=ymax*10.0/(pow(10,dy));
    y3=ymin*10.0/(pow(10,dy));
    rymax=(y2+y1)*pow(10,dy-1);
    rymin=(y3-y1)*pow(10,dy-1);

          }

    gra_draw(rxmax,rxmin,rymax,rymin);
    draw_point(rxmax,rxmin,rymax,rymin,cnt,x,y);
```

```
            n++;
            cnt++;

          }

    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
  }
```

x, *y*のいずれの場合も最大値と最小値の差を求め，差の常用対数を算出する．常用対数を切り捨てにより整数化すれば，刻み幅の桁数が算出できる．これにより，前プログラム同様，グラフの軸の設定範囲を自動的に決定している．

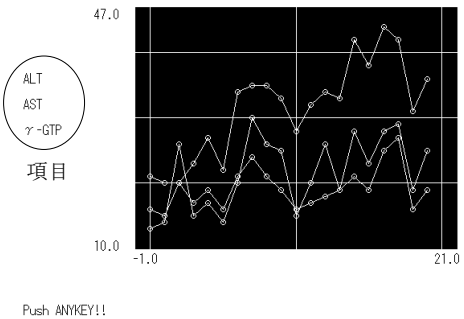


Fig.9.4　折れ線グラフで表した血中成分濃度

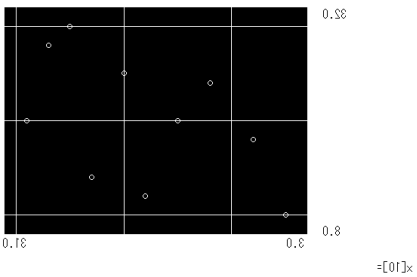


Fig.9.5　自動で描画範囲調整する散布図

９章の課題

▲EX97.Cで描画したBLOOD.DATの内容・グラフからこの血液検査・検体について気付いたことを述べよ．

●EX98.Cは数値以外の文字を入力した場合の処理が設定されていない．これを設定して正常な動作をするようにせよ．

★EX98.Cは中空円のみ描画するので，異なる測定対象のデータを重ね合わせるのに適切であるといえない．異なる図形を描き，異なる種類の測定データがポイントできるように改良せよ．

１０章　演算処理

測定データは多くの場合，雑音や誤差などの無関係なデータを含んでいる．あるいは，そのままの形ではそこに含まれている情報が読み取りにくく，意味を持たない場合が多い．そこで，通常は測定データを演算処理して情報を抽出し，その意味を判断するフィルタリングを行う．

本章では統計学的処理と，その基本となる基礎的な数学アルゴリズムについて学習する．

10.1　χ^2検定

前章で紹介したサイコロ・シミュレーションプログラム(EX95.C,EX96.C)は1000～10000回以上の試行回数でそのヒストグラムが正規分布曲線に相似する．コンピュータでは10000回以上の試行も瞬時に完了するので，2つのサイコロの目の和が確率論的に正しいことを検討する際に不都合を生じない．

しかし，ほとんどの研究では，得られる標本数(検体)は母集団と比較して少なく，統計学的手法に基づいた検定に頼らざるを得ない．

ここではサイコロ・シミュレーションプログラム(EX95.C)を例に，少ない試行回数で2つのサイコロの目の和を検定する．まず，すべての「目の和」の出現確率について，試行1回ごとにχ^2検定を行うプログラムを作成する．

以下のプログラムではEX95.Cとの差異を囲いで示す．

ファイル名：EXa1.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


gra_draw()
          {
           int i,dx;

           ClearScreen(0);
           Rectangle(140,80,540,380,7);

           for(i=2;i<13;i++)
                           {
                            dx=i*400/13/8+18;
                            SetCursorPos(dx,24);
                            printf("%d",i);
```

```
                        }
        }

draw_point(double rymax,int x[],double rate[])
        {
         int gx,gy,i;
         int col[]={0,1,2,3,4,5,6,8,9,10,11,12,13};
         double rx1,rx2;

         SetCursorPos(10,5);
         printf("%5.1lf",rymax);
         rx1=rx2=140.0;

         for (i=2;i<13;i++)
                        {
                         gx=i/13.0*400+140;
                         gy=x[i]/rymax*(-300)+380;
                         FillRectangle(gx,gy,gx+10,380,col[i]);

                         rx2=400.0*rate[i]+rx1;
                         FillRectangle(rx1,30,rx2,40,col[i]);
                         rx1=rx2;

                        }
        }

int judge(int n,int x[],double exrate[])
        {
         int i;
         double  kai,kai2,d,ex[13];

         kai=23.2093;              /*p=0.01,degree of freedom=10*/
         kai2=0.0;

         for (i=2;i<13;i++)
                        {
                         ex[i]=n*exrate[i];
                         d=x[i]-ex[i];
                         kai2+=d*d/ex[i];
                        }

                         printf ("%lf %lf   :",kai,kai2);

         if (kai2<kai)
                        {
                         return (1);
```

```
                    }
                else
                    {
                     return (0);
                    }
                                                        /*1:OK 0:False*/
        }

main()
      {
       int    x[13],i,d1,d2,sum,cnt;
       int    h;
       float  f;
       unsigned long time1;
       unsigned seed;
       double rymax;
       double rate[13],exrate[13];

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       rymax=3.0;
       cnt=0;

       time(&time1);
       seed=time1;
       srand(seed);

       for(i=0;i<13;i++)
                       {
                        x[i]=0;
                        rate[i]=0.0;
                       }

       exrate[0]=0.0;
       exrate[1]=0.0/36.0;
       exrate[2]=1.0/36.0;
       exrate[3]=2.0/36.0;
       exrate[4]=3.0/36.0;
       exrate[5]=4.0/36.0;
       exrate[6]=5.0/36.0;
       exrate[7]=6.0/36.0;
       exrate[8]=5.0/36.0;
       exrate[9]=4.0/36.0;
       exrate[10]=3.0/36.0;
```

```
exrate[11]=2.0/36.0;
exrate[12]=1.0/36.0;

while (cnt<10000)
          {
           f=rand();
           d1=f*6/32768.0+1;
           f=rand();
           d2=f*6/32768.0+1;
           sum=d1+d2;

           x[0]++;
           x[sum]++;
           rate[0]=0.0;

           if (x[sum]>(rymax-1))
                          {
                           rymax*=1.2;
                          }

           for (i=1;i<13;i++)
                           {
                            rate[i]=(double)x[i]/x[0];
                            rate[0]+=rate[i];
                           }

            gra_draw();
            draw_point(rymax,x,rate);
            cnt++;
            SetCursorPos(5,28);
            printf("COUNT:%d     ",cnt);
            h=judge(cnt,x,exrate);

            if (h==1)
                    {
                     printf ("OK");
                    }
                else
                    {
                     printf ("False");
                    }

            getch();

           }
```

```
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
}
```

2つのサイコロの目の和の出現確率の理論値はTable 9.1で示した通りである．本プログラムでは，この理論値から試行回数に対する各「目の和」ごとの出現回数を算出し，シミュレーションの結果を用いて，χ^2検定を行う．

χ^2検定は式(10.1)でχ^2値を求め，この値がχ^2分布表から得られた値より小さければ，統計学上，棄却される確率がp以下であると考える．このとき，「目の和」の数から自由度νが決定する．この場合は，「目の和」は11通りなので，自由度は10である．また，棄却域はα<0.01とした．

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)}{e_i} \qquad \cdots\cdots (10.1)$$

ただしo_iは観測度数であり，e_iは期待度数，nは試行回数である．

本プログラムでは，試行1回ごとにχ^2検定を行っているが，統計学的にはすべての期待度数が5以上である場合に適応するとされている．したがって，試行回数が180回以上の場合に有効である．

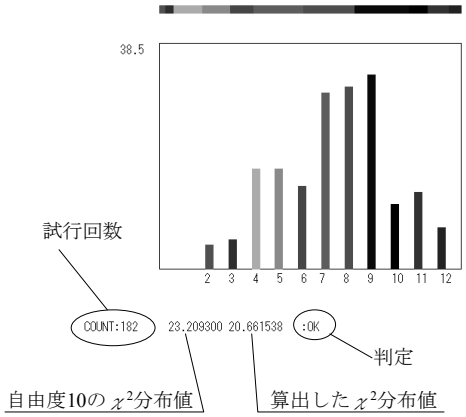


Fig.10.1　サイコロシミュレーションのχ^2検定による判定

10.2　正規分布曲線に基づいた大標本法による検定

サイコロ・シミュレーションは二項分布であるので，標本数すなわち試行回数が多い場合，その確率が正規分布曲線にしたがう．したがって，ある「目の和」の出現確率$\hat{p}$を基に，理論上の割合pについての検定を行う．棄却域をα<0.01とすると，正規分布表から式(10.2)が成立する場合，検定上，シミュレーションの結果が正しいと言える．

$$p - 2.58\sqrt{\frac{pq}{n}} < \hat{p} < p + 2.58\sqrt{\frac{pq}{n}} \qquad \cdots\cdots (10.2)$$

ただし$q = 1 - p$である．

以下のプログラムではEX95.Cとの差異を囲いで示す．

ファイル名：EXa2.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


gra_draw()
          {
           int i,dx;

           ClearScreen(0);
           Rectangle(140,80,540,380,7);

           for(i=2;i<13;i++)
                           {
                            dx=i*400/13/8+18;
                            SetCursorPos(dx,24);
                            printf("%d",i);
                           }
           }

draw_point(double rymax,int x[],double rate[])
          {
           int gx,gy,i;
           int col[]={0,1,2,3,4,5,6,8,9,10,11,12,13};
           double rx1,rx2;

           SetCursorPos(10,5);
           printf("%5.1lf",rymax);
           rx1=rx2=140.0;

           for (i=2;i<13;i++)
                           {
                            gx=i/13.0*400+140;
                            gy=x[i]/rymax*(-300)+380;
                            FillRectangle(gx,gy,gx+10,380,col[i]);

                            rx2=400.0*rate[i]+rx1;
                            FillRectangle(rx1,30,rx2,40,col[i]);
```

```
                        rx1=rx2;

                       }
        }

main()
      {
       int    x[13],i,d1,d2,sum,cnt,dx;
       int    h[13];
       float  f;
       unsigned long time1;
       unsigned seed;
       double rymax,pmax,pmin,p,p0;
       double rate[13],exrate[13];

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       rymax=3.0;
       cnt=0;

       time(&time1);
       seed=time1;
       srand(seed);

       for(i=0;i<13;i++)
                  {
                   x[i]=0;
                   rate[i]=0.0;
                  }

       exrate[0]=0.0;
       exrate[1]=0.0/36.0;
       exrate[2]=1.0/36.0;
       exrate[3]=2.0/36.0;
       exrate[4]=3.0/36.0;
       exrate[5]=4.0/36.0;
       exrate[6]=5.0/36.0;
       exrate[7]=6.0/36.0;
       exrate[8]=5.0/36.0;
       exrate[9]=4.0/36.0;
       exrate[10]=3.0/36.0;
       exrate[11]=2.0/36.0;
       exrate[12]=1.0/36.0;
```

```
while (cnt<10000)
          {
           f=rand();
           d1=f*6/32768.0+1;
           f=rand();
           d2=f*6/32768.0+1;
           sum=d1+d2;

           x[0]++;
           x[sum]++;
           rate[0]=0.0;

           if (x[sum]>(rymax-1))
                          {
                           rymax*=1.2;
                          }

           for (i=1;i<13;i++)
                           {
                            rate[i]=(double)x[i]/x[0];
                            rate[0]+=rate[i];
                            p=exrate[i];
                            p0=sqrt(p*(1-p)/(cnt+1));
                            pmax=p+2.58*p0;
                            pmin=p-2.58*p0;
                                    /* In case of alpha=0.01 */
                            if (rate[i]<=pmax && rate[i]>=pmin)
                                                        {
                                        h[i]=1;
                                                        }
                                                    else
                                                        {
                                        h[i]=0;
                                                        }
                           }

            gra_draw();
            draw_point(rymax,x,rate);
            cnt++;
            SetCursorPos(5,28);
            printf("COUNT:%d     ",cnt);

    for(i=2;i<13;i++)
                        {
                        dx=i*400/13/8+18;
                        SetCursorPos(dx,25);
```

```
                    if (h[i]==0)
                                {
                                 printf ("X");
                                }
                            else
                                {
                                 printf ("O");
                                }
                    }

              getch();

             }

     printf ("Push ANYKEY!!");
     getch();

     RestoreMode();                                    /* Restore Mode */
    }
```

このプログラムではそれぞれの「目の和」の出現確率を検定している．試行ごとに検定を行い，検定上正しいと判断した場合には「O」を，棄却と判断される場合には「X」を表示している．

プログラム上，試行回数1回から検定を行うが，大標本法であるので少なくとも30回以上の場合について有効であると考える．

なお，sqrt関数は平方根を求める関数である．

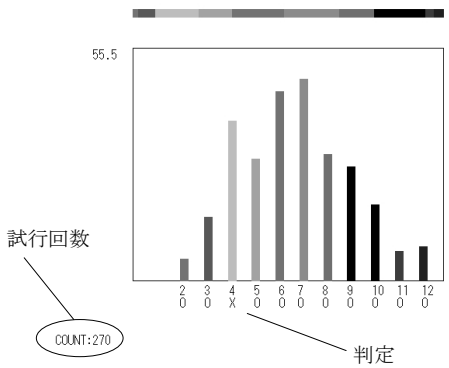


Fig.10.2　サイコロシミュレーションの大標本法による判定

10.3　Studentの*t*ー分布曲線に基づいた小標本法による検定

標本数，すなわち試行回数が少ない場合，大標本法では適切な検定ができない．そこで，正規分布曲線の代わりにStudentの*t*ー分布曲線を用いると，少ない標本数でも信頼しうる検定が可能であるとされている．

***t*－検定では棄却域 α と，標本数*n*から決定する自由度 ν により*t*値を決定し，大標本で用いた式(10.2)の代わりに，式(10.3)によって検定を行う．

$$p-t\sqrt{\frac{pq}{n}}<\hat{p}<p+t\sqrt{\frac{pq}{n}} \qquad \cdots\cdots(10.3)$$

ただし $q=1-p$ である．

以下のプログラムではEXa2.Cとの差異を囲いで示す．

ファイル名：EXa3.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


gra_draw()
          {
           int i,dx;

           ClearScreen(0);
           Rectangle(140,80,540,380,7);

           for(i=2;i<13;i++)
                          {
                           dx=i*400/13/8+18;
                           SetCursorPos(dx,24);
                           printf("%d",i);
                          }
           }

draw_point(double rymax,int x[],double rate[])
          {
           int gx,gy,i;
           int col[]={0,1,2,3,4,5,6,8,9,10,11,12,13};
           double rx1,rx2;

           SetCursorPos(10,5);
           printf("%5.1lf",rymax);
           rx1=rx2=140.0;

           for (i=2;i<13;i++)
                          {
```

```
                                gx=i/13.0*400+140;
                                gy=x[i]/rymax*(-300)+380;
                                FillRectangle(gx,gy,gx+10,380,col[i]);

                                rx2=400.0*rate[i]+rx1;
                                FillRectangle(rx1,30,rx2,40,col[i]);
                                rx1=rx2;

                               }
          }

main()
      {
       int    x[13],i,d1,d2,sum,cnt,dx;
       int    h[13];
       float  f;
       unsigned long time1;
       unsigned seed;
       double rymax,pmax,pmin,p,p0,t1;
       double rate[13],exrate[13];
       double t[]={63.557,9.925,5.841,4.604,4.032,
                   3.707,3.499,3.355,3.250,3.169,
                   3.106,3.055,3.012,2.977,2.947,
                   2.921,2.898,2.878,2.861,2.845,
                   2.831,2.819,2.807,2.797,2.787,
                   2.779,2.771,2.763,2.756,2.750
                  };

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       rymax=3.0;
       cnt=0;

       time(&time1);
       seed=time1;
       srand(seed);

       for(i=0;i<13;i++)
                        {
                         x[i]=0;
                         rate[i]=0.0;
                        }

       exrate[0]=0.0;
```

```
exrate[1]=0.0/36.0;
exrate[2]=1.0/36.0;
exrate[3]=2.0/36.0;
exrate[4]=3.0/36.0;
exrate[5]=4.0/36.0;
exrate[6]=5.0/36.0;
exrate[7]=6.0/36.0;
exrate[8]=5.0/36.0;
exrate[9]=4.0/36.0;
exrate[10]=3.0/36.0;
exrate[11]=2.0/36.0;
exrate[12]=1.0/36.0;

while (cnt<10000)
          {
           f=rand();
           d1=f*6/32768.0+1;
           f=rand();
           d2=f*6/32768.0+1;
           sum=d1+d2;

           x[0]++;
           x[sum]++;
           rate[0]=0.0;

           if (x[sum]>(rymax-1))
                          {
                           rymax*=1.2;
                          }
           t1=2.58;

           if (cnt>=1 && cnt<=30)
                                {
                                 t1=t[cnt-1];
                                }

           for (i=1;i<13;i++)
                           {
                            rate[i]=(double)x[i]/x[0];
                            rate[0]+=rate[i];
                            p=exrate[i];
                            p0=sqrt(p*(1-p)/(cnt+1));
                            pmax=p+t1*p0;
                            pmin=p-t1*p0;
                                        /* In case of p=0.01 */
                            if (rate[i]<=pmax && rate[i]>=pmin)
```

```
                                                        {
                                                h[i]=1;
                                                        }
                                                    else
                                                        {
                                                h[i]=0;
                                                        }
                                    }

                gra_draw();
                draw_point(rymax,x,rate);

               if (cnt>=1 && cnt<=30)
                                    {
                                     SetCursorPos(5,26);
                                     printf("t-distribution");
                                    }

                cnt++;
                SetCursorPos(5,28);
                printf("COUNT:%d     ",cnt);

        for(i=2;i<13;i++)
                          {
                           dx=i*400/13/8+18;
                           SetCursorPos(dx,25);
                           if (h[i]==0)
                                       {
                                        printf ("X");
                                       }
                                  else
                                       {
                                        printf ("O");
                                       }
                          }


                getch();

               }

 printf ("Push ANYKEY!!");
 getch();

 RestoreMode();                                  /* Restore Mode */
}
```

***t*－検定では試行回数*n*から自由度 ν が決定するので，プログラム中で一意的に式を与えることができない．そこで，棄却域 α<0.01の場合の*t*値を配列変数として与え，これにより試行回数ごとの*t*－検定を行う．**

プログラム中で与えた*t*値は30個であるので試行回数2～31回までは*t*－検定を行っており，32回以上は前節の大標本法で検定を行った．

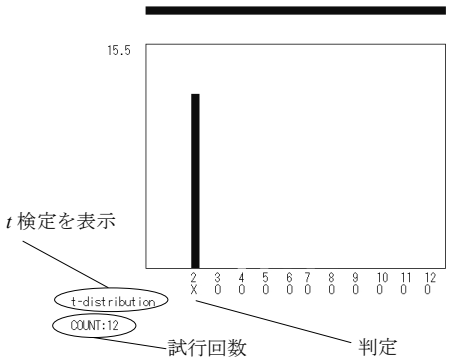


Fig. 10.3　サイコロシミュレーションの*t*検定による判定

10.4　線形最小二乗法

科学的に測定された事象から，測定されていない事象の予測を行う場合，説明変量*x*に対する目的変量*y*を予測するためのモデルを構築する．回帰法はこのための手法の一つで，2つの変数*x*，*y*を用いた線形式，すなわち一次方程式の係数を算出する．このとき，予測されるモデルにより得られる理論値と実測値の誤差(残差)の二乗の和(残差二乗和)が最小となるようにすることから，線形最小二乗法(method of least squares)**として知られている．**

一次方程式 $y = a_0 + a_1 x$ **について，*n*組の実測値(x_1，y_1)，(x_2，y_2)，(x_3，y_3)…(x_n，y_n)が得られたとき，式(10.4)で表される関係式が成立する．**

$$\begin{cases} a_0 \sum_{i=1}^{n} 1 + a_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i \\ a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^{\,2} = \sum_{i=1}^{n} x_i y_i \end{cases} \qquad \cdots\cdots (10.4)$$

ここで $S_x = \sum_{i=1}^{n} x_i, S_{x2} = \sum_{i=1}^{n} x_i^{\,2}, S_y = \sum_{i=1}^{n} y_i, S_{xy} = \sum_{i=1}^{n} x_i y_i$ **とおくと**

$a_0 = \dfrac{-S_y S_{x2} + S_x S_{xy}}{S_x^{\,2} - nS_{x2}}$，　$a_1 = \dfrac{S_y S_x + nS_{xy}}{S_x^{\,2} - nS_{x2}}$ **が成立する．**

ここでは，入力したデータに対する相関式が得られることを，散布図を描くプログラム(EX98.C)を基礎とした以下のプログラムで確認する．

ファイル名：EXa4.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>


gra_draw(double rxmax,double rxmin,double rymax, double rymin)
          {
           int i,sx,ex,sy,ey,dx,dy,x1,y1;
           double rx,ry;

           ClearScreen(0);
           FillRectangle(140,80,540,380,15);

           rx=rxmax-rxmin;
           ry=rymax-rymin;

           dx=log10(rx);
           dy=log10(ry);

           sx=rxmin/(pow(10,dx));
           ex=rxmax/(pow(10,dx));
           sy=rymin/(pow(10,dy));
           ey=rymax/(pow(10,dy));

           for (i=sx;i<ex+1;i++)
                              {
                               x1=(i*pow(10,dx)-rxmin)/rx*400+140;
                               if (x1>140 && x1<540)
                                                    {
                                          Line(x1,80,x1,380,3);
                                                    }
                              }
           for (i=sy;i<ey+1;i++)
                              {
                               y1=(i*pow(10,dy)-rymin)/ry*(-300)+380;
                               if (y1>80 && y1<380)
                                                    {
                                          Line(140,y1,540,y1,3);
                                                    }
                              }
          }

draw_point(double rxmax,double rxmin,double rymax,double rymin,int cnt,double
x[],double y[])
          {
```

```
        int    i,gx,gy;
        double xr,yr;
        int col[]={0,1,4,5,6};

        xr=rxmax-rxmin;
        yr=rymax-rymin;

        SetCursorPos(16,24);
        printf("%5.1lf",rxmin);
        SetCursorPos(63,24);
        printf("%5.1lf",rxmax);
        SetCursorPos(10,23);
        printf("%5.1lf",rymin);
        SetCursorPos(10,5);
        printf("%5.1lf",rymax);

        for (i=0;i<cnt;i++)
                        {
                         gx=(x[i]-rxmin)/xr*400+140;
                         gy=(y[i]-rymin)/yr*(-300)+380;

                         Circle(gx,gy,3,col[i%5]);
                        }
    }
draw_eq(double  rxmax,double  rxmin,double  rymax,double  rymin,double  a0,double
a1)
    {
        int i,gx,gy;
        double xdif,x,y,xr,yr;

        xdif=(rxmax-rxmin)/1000.0;
        xr=rxmax-rxmin;
        yr=rymax-rymin;

        SetCursorPos(5,3);
        printf ("y=%5.4lf x + %5.4lf",a1,a0);

        for (i=0;i<1000;i++)
                        {
                         x=rxmin+xdif*i;
                         y=a0+a1*x;
                         gx=(x-rxmin)/xr*400+140;
                         gy=(y-rymin)/yr*(-300)+380;

                         if (gx>140 && gx<540 && gy>80 && gy<380)
                            {
```

```
                                PutPoint(gx,gy,0);
                            }

                        }

            }


main()
      {
       int n,cnt,dx,dy,x1,y1,x2,y2,x3,y3;
       double xmax,xmin,ymax,ymin;
       double rxmax,rxmin,rymax,rymin,rx,ry;
       double hx,hy;
       double x[100],y[100];
       double xsum,ysum,x2sum,xysum;
       double a0,a1,div;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       xmax=xmin=ymax=ymin=0.0;
       n=0;
       cnt=n+1;
       xsum=ysum=x2sum=xysum=0.0;
       a0=a1=0.0;

       while (-1)
              {
               SetCursorPos(0,26);
               printf ("x[%2d]=",n);
               scanf("%lf",&x[n]);
               printf ("y[%2d]=",n);
               scanf("%lf",&y[n]);
               printf ("¥n");

               xsum+=x[n];
               ysum+=y[n];
               x2sum+=x[n]*x[n];
               xysum+=x[n]*y[n];
               div=xsum*xsum-x2sum*cnt;

               if (div!=0.0)
                        {
                         a0=(-ysum*x2sum+xysum*xsum)/div;
```

```
            a1=(ysum*xsum-xysum*cnt)/div;
           }

if (n==0)
        {
         xmax=xmin=x[n];
         ymax=ymin=y[n];
         rxmax=(int)xmax+1.0;
         rxmin=(int)xmin-1.0;
         rymax=(int)ymax+1.0;
         rymin=(int)ymin-1.0;
        }
else
        {
         if (xmax<x[n])
                        {
                         xmax=x[n];
                        }
         if (xmin>x[n])
                        {
                         xmin=x[n];
                        }
         if (ymax<y[n])
                        {
                         ymax=y[n];
                        }
         if (ymin>y[n])
                        {
                         ymin=y[n];
                        }

if (xmax==xmin)
              {
               xmax+=1.0;
               xmin-=1.0;
              }
if (ymax==ymin)
              {
               ymax+=1.0;
               ymin-=1.0;
              }

rx=xmax-xmin;
ry=ymax-ymin;

dx=log10(rx);
```

```
                dy=log10(ry);

                hx=rx*0.1;
                x1=hx/(pow(10,dx-1));
                x2=xmax*10.0/(pow(10,dx));
                x3=xmin*10.0/(pow(10,dx));
                rxmax=(x2+x1)*pow(10,dx-1);
                rxmin=(x3-x1)*pow(10,dx-1);

                hy=ry*0.1;
                y1=hy/(pow(10,dy-1));
                y2=ymax*10.0/(pow(10,dy));
                y3=ymin*10.0/(pow(10,dy));
                rymax=(y2+y1)*pow(10,dy-1);
                rymin=(y3-y1)*pow(10,dy-1);

                        }

                gra_draw(rxmax,rxmin,rymax,rymin);
                draw_point(rxmax,rxmin,rymax,rymin,cnt,x,y);
                if (cnt>1 && div!=0.0)
                                        {
                                         draw_eq(rxmax,rxmin,rymax,rymin,a0,a1);
                                        }
                n++;
                cnt++;

               }

        printf ("Push ANYKEY!!");
        getch();

        RestoreMode();                                  /* Restore Mode */
      }
```

一般に，科学実験で得られる結果は線形一次方程式で表されるか，限定された範囲内について考えることから線形一次方程式であると見なすことが多い．

例えば，酵素反応速度と基質濃度の関係を表したミカエリスーメンテン(Michaeris-Menten)の式は非線形式であるが，これを変形して得られるラインウィーバー・バーク(Lineweaver-Burk)の式は基質濃度の逆数と反応速度の逆数の間に，線形一次方程式で表されることが知られている．

本プログラムのデータ入力部を変更することで，上記のような事例にも対応できる．

ところで，本プログラム中で使用したdouble型で定義されるpow(double x,double y)関数はmath.hで定義されており，x^yを戻り値として返す．

10.5　加算平均による雑音除去

医学検査の一つに事象関連電位(ERP,Event-related potential)の測定がある．これは，人間の感覚器もしくは神経にある種の刺激を与え，これによって引き起こされる誘発電位(Evoked Potential)を測定するものである．

これは神経が信号伝達のために発する電位のうち，刺激入力後特定の時刻(数～数十[msec])に出現するパルス状の電位で，これを測定することで神経の伝達異常などを判断できる．この電位は数～数十[μV]なので，様々な雑音の影響により波形としてとらえにくい．

一般に雑音は刺激と無関係に乱雑に生じるので，測定した信号データを繰り返し加算することにより，互いに打ち消しあい，刺激に同期して出現する誘発電位は加算によって振幅が増大する．

このように加算の繰り返しによって雑音を打ち消し，目的とする信号を増幅した後，加算回数で平均して測定値を得る手法を加算平均法(averaging)という．

ファイル名：EXa5.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

draw_gra1(double y , double by , int x)
                                {
                                 int gx1,gx2,gy1,gy2;

                                 gx1=x*2+140;
                                 gx2=(x-1)*2+140;
                                 gy1=-y*50+155;
                                 gy2=-by*50+155;

                                 Line(gx1,gy1,gx2,gy2,3);
                                }
draw_gra2(double y , double by , int x , int cnt)
                                {
                                 int gx1,gx2,gy1,gy2,c;
                                 int col[]={2,3,4,7,9,10,13};

                                 c=cnt%6;

                                 gx1=x*2+140;
                                 gx2=(x-1)*2+140;
```

```
                                gy1=-y*50+325;
                                gy2=-by*50+325;

                                Line(gx1,gy1,gx2,gy2,col[c]);
                               }

main()
      {
       int i,cnt;
       float f,wn;
       double y,by,pi;
       double sy[200];
       unsigned long time1;
       unsigned seed;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       time(&time1);
       seed=time1;
       srand(seed);

       pi=3.1415926;
       cnt=1;
       by=0.0;

       for (i=0;i<200;i++)
                        {
                         sy[i]=0.0;
                        }

       Rectangle(140,250,540,400,7);

       while (cnt<1000)
                 {
                  FillRectangle(140,80,540,230,0);
                  Rectangle(140,80,540,230,15);

                  for (i=1;i<200;i++)
                                       {
                                        f=rand();
                                        wn=f/32768.0-0.5;

                       if (i>36 && i<72)
                                            {
```

```
                y=-sin(i*5/360.0*2*pi);
              }
            else
              {
                y=0.0;
              }

            y+=wn;
            sy[i]+=y;

            draw_gra1(sy[i]/cnt,sy[i-1]/cnt,i);
            draw_gra2(y,by,i,cnt);

            by=y;

          }

        SetCursorPos(5,26);
        printf("%d times¥n",cnt);

        cnt++;
        getch();
      }

  printf ("Push ANYKEY!!");
  getch();

  RestoreMode();                              /* Restore Mode */
}
```

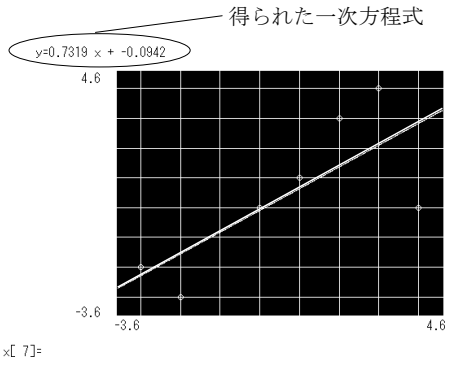


Fig.10.4　最小二乗法によるグラフ描画

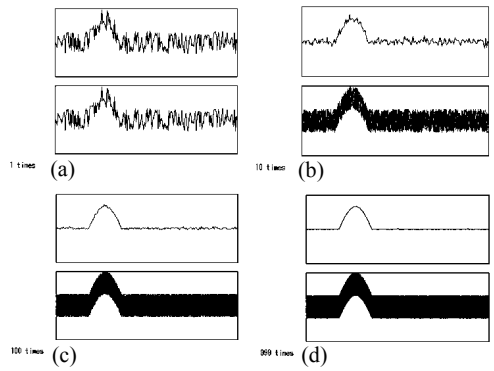


Fig.10.5　加算平均法のシミュレーション

このプログラムでは信号として正弦波に信号振幅の半分以下の雑音を混合したデータを与えて，加算平均のシミュレーションを行った．

加算回数が増加するほど，信号波形が鮮明に得られることが確認できる．

10.6　数値積分

ガス・クロマトグラフや液体クロマトグラフは出力された曲線と基線によって構成される面積の比により成分濃度比を決定する．

一般に，面積は積分によって算出可能であるが，検出器(detector)によって測定した曲線を表現する方程式を得ることは必ずしも容易ではない．したがって，公式に基づいた定積分を行うのは困難である．

一方，積分の原理に基づくと，台形や長方形のような面積の導出が容易な小領域に分割したものの総計として得ることが出来る．定式化したものはこの極限を考えるが，コンピュータで処理する場合，演算によって生じる丸め誤差などを考慮すると，分割幅を極めて小さくすれば十分な精度の結果を得ることができる．

この考え方として，Fig.10.6に示したようなものが主流である．EX91.Cをベースとした以下のプログラムで精度を確認する．

ファイル名：EXa6.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

trans_axis(double x, double y)
                              {
                               int tx,ty;

                               tx=x*60+340;
                               ty=y*(-10)+230;

                  if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                             {
                               PutPoint(tx,ty,4);
                             }

                               }

section(double x, double y)
                            {
                             int tx,ty1,ty2;

                             tx=x*60+340;
                             ty1=y*(-10)+230;
                             ty2=230;
```

```
                if (tx>=140 && tx<=540 && ty1>=80 && ty1<=380)
                        {
                          Line(tx,ty1,tx,ty2,10);
                        }

                         }

double func(double x)
                       {
                        double x1,x2,x3,y;

                        x1=x-1;
                        x2=x+1;
                        x3=x-2;
                        y=x1*x2*x3;

                        return(y);

                       }

main()
      {
       double x,y,sx,ex,tx,div,s,ty1,ty2;
       int i;

       SetGraphicsMode();                              /* Change Mode 12h */

       ClearScreen(0);
       FillRectangle(140,80,540,380,7);
       Line(140,230,540,230,1);
       Line(340,80,340,380,1);

       for (i=-1000;i<1500;i++)                                /*Draw Graph*/
                              {
                               x=i/400.0;
                               y=func(x);
                               trans_axis(x,y);
                              }

      sx=-1.0;
      ex= 1.0;
      div=0.001;
      s=0.0;
      ty1=func(sx);

      for (tx=sx+div;tx<ex;tx+=div)
```

```
{
 ty2=func(tx);
 s+=(ty1+ty2)*div/2.0;
 ty1=ty2;
 SetCursorPos(5,21);
 printf("%lf",s);
 section (tx,ty1);
 getch();
}
```

```
 SetCursorPos(0,28);
 printf ("Push ANYKEY!!");
 getch();

 RestoreMode();                                    /* Restore Mode */
}
```

プログラムを実行すると，任意のキーを一回押すごとに，分割した台形領域の面積を加算し，最終的な結果を得る．刻み幅Δx=0.001である．

このプログラムは，関数$f(x)=(x+1)(x-1)(x-2)$を例としている．区間[-1,1]で積分すると解として8/3を得る．この結果と，プログラムの結果は一致していることを確認する．

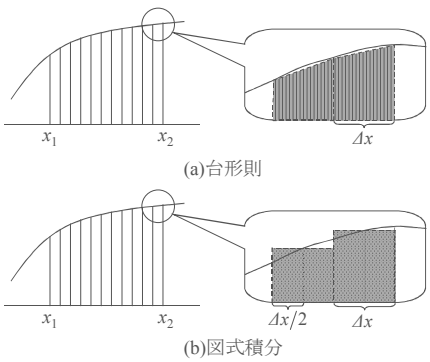


Fig.10.6　数値演算による積分法

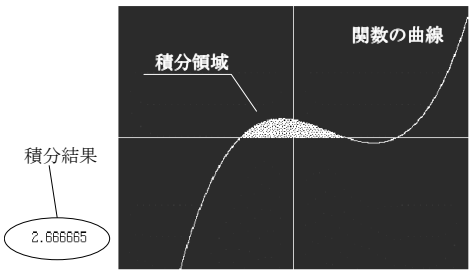


Fig.10.7　台形則による積分演算の結果

10.7　数値微分

微分は数値演算上重要な基礎の技術である．積分同様に，方程式の微分という形では計算できないが，原理に基づいた数値演算は可能である．微分する点を中心として前方差分商近似，後方差分商近似，中央差分商近似がある．詳しくは，Fig.10.8を参照してほしい．

EX91.Cをベースとした以下のプログラムでは，もっとも多用される前方差分商近似を用いて数値微分を行う．

ファイル名：EXa7.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

trans_axis(double x, double y)
                                {
                                 int tx,ty;

                                 tx=x*60+340;
                                 ty=y*(-10)+230;

                    if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                              {
                                PutPoint(tx,ty,4);
                              }

                                 }

double func(double x)
                       {
                        double x1,x2,x3,y;

                        x1=x-1;
                        x2=x+1;
                        x3=x-2;
                        y=x1*x2*x3;

                        return(y);

                       }

linear(double a, double b)
                             {
                              int tx,ty,i;
                              double x,y;

        for (i=-1500;i<1500;i++)
                                  {
                                   x=i/400.0;
                                   y=a*x+b;

                              tx=x*60+340;
                              ty=y*(-10)+230;
```

```
                    if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                                        {
                                         PutPoint(tx,ty,0);
                                        }

                                }
                        }

main()
       {
        double x,y,dx,dy,tx,ty,a,b,div;
        int i,px,py;

        SetGraphicsMode();                              /* Change Mode 12h */

        ClearScreen(0);
        FillRectangle(140,80,540,380,7);
        Line(140,230,540,230,1);
        Line(340,80,340,380,1);

        for (i=-1000;i<1500;i++)                                /*Draw Graph*/
                                {
                                 x=i/400.0;
                                 y=func(x);
                                 trans_axis(x,y);
                                }

        SetCursorPos(0,24);
        printf ("x=");
        scanf("%lf",&dx);
        dy=func(dx);

        px=dx*60+340;
        py=dy*(-10)+230;
        Circle (px,py,4,0);

        div=0.001;
        tx=dx+div;
        ty=func(tx);
        a=(ty-dy)/div;
        b=ty-a*dx;

        SetCursorPos(0,25);
        printf ("y=(%lf )x+(%lf)",a,b);
        linear(a,b);
```

```
    SetCursorPos(0,28);
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
}
```

プログラムを実行すると，任意のxを入力するように求めてくる．この点における微分を行い，接線の方程式を求め，グラフ中に接線を描画する．

この関数の導関数は$f'(x)=3x^2-4x-1$であるので，実行結果を手計算で求め，比較してみる．実際には，演算誤差による若干のズレを生じる場合がある．

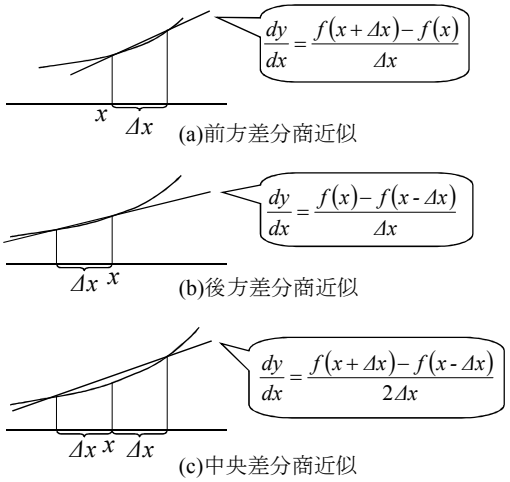


Fig.10.8　数値演算による微分法

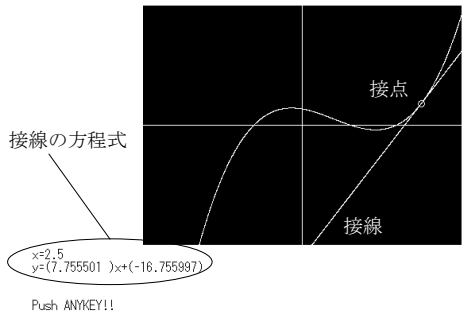


Fig.10.9　微分演算による接線の描画

10.8　数値解の導出　その１～ニュートン法～

ニュートン法は，代数方程式$f(x)=0$を解く時，初期値x_1を与え，x_1における関数$y=f(x)$の接線を考え，この接線とx軸との交点をx_2として同様の操作を繰り返す．n番目のx_nとn+1番目のx_{n+1}の差がきわめて小さい誤差ε以下になったとき収束し，解が求まったとする数値解の導出法である．ここで誤差εを収束誤差という．詳しくはFig.10.10を参照してほしい．

具体的にはEXa7.Cをベースとした以下のプログラムによって解説する．

ファイル名：EXa8.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
```

```
#include <math.h>

trans_axis(double x, double y)
                                {
                                 int tx,ty;

                                 tx=x*60+340;
                                 ty=y*(-10)+230;

                    if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                              {
                                PutPoint(tx,ty,4);
                              }

                                 }

double func(double x)
                       {
                        double x1,x2,x3,y;

                        x1=x-1;
                        x2=x+1;
                        x3=x-2;
                        y=x1*x2*x3;

                        return(y);

                       }

linear(double a, double b ,double x1, double x2)
                              {
                               int tx,ty,ty2,i;
                               double x,y,div;

                               div=(x2-x1)/400.0;

       for (i=0;i<400;i++)
                                {
                                 x=x1+div*i;
                                 y=a*x+b;

                                 tx=x*60+340;
                                 ty=y*(-10)+230;

                    if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                                          {
```

```
                                        PutPoint(tx,ty,0);
                                      }
                                   }

                                  y=func(x2);
                                  ty2=y*(-10)+230;
                                  Line(tx,ty,tx,ty2,0);

                              }

main()
      {
       double x,y,dx,dy,tx,ty,a,b,div,x1,x2,err;
       int i,px,py;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);
       FillRectangle(140,80,540,380,7);
       Line(140,230,540,230,1);
       Line(340,80,340,380,1);

       for (i=-1000;i<1500;i++)                          /*Draw Graph*/
                              {
                               x=i/400.0;
                               y=func(x);
                               trans_axis(x,y);
                              }

       err=100.0;

       SetCursorPos(1,24);
       printf ("x=");
       scanf("%lf",&dx);

       while (err>0.0001)                                /*Error value*/
                         {
       dy=func(dx);

       px=dx*60+340;
       py=dy*(-10)+230;
       Circle (px,py,4,0);

       div=0.001;
       tx=dx+div;
       ty=func(tx);
```

```
    a=(ty-dy)/div;
    b=ty-a*dx;
    x1=dx;
    x2=-b/a;
    err=fabs(x1-x2);

    linear(a,b,x1,x2);

    dx=x2;

                    }

    SetCursorPos(1,25);
    printf ("Solution:x=%lf¥n",x2);

    SetCursorPos(0,28);
    printf ("Push ANYKEY!!");
    getch();

    RestoreMode();                                  /* Restore Mode */
}
```

プログラムを実行すると，任意のxを入力するよう求めてくる．入力したxを始点として，解の探索を開始する．この関数では，3つの解が存在するが，そのうちの1つの点にたどり着く．どの点にたどり着くかは始点によって異なる．

グラフ表示の都合上，-3～3でxを選択しないと，表示がおかしくなる場合があるが，演算の解としては誤差の範囲内で正しい解を得る．

なお，このプログラム中で使用しているfabs()関数はmath.hに定義されているdouble型の関数で，実数の絶対値を戻り値として返すものである．

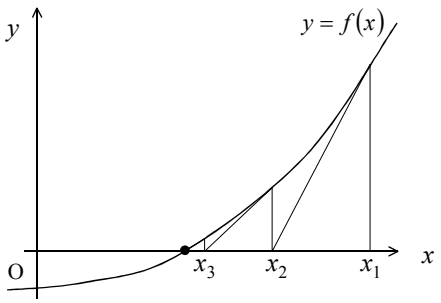


Fig.10.10　ニュートン法の原理

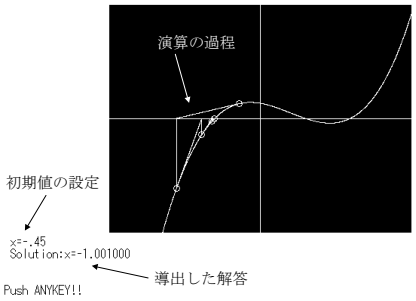


Fig.10.11　ニュートン法による数値解の算出

10.9　数値解の導出　その２～二分割法～

二分割法は，代数方程式$f(x)=0$を解く時，解の近似値として与えたx_1，x_2について$f(x_1)\bullet f(x_2)<0$が成立するとき，x_1，x_2の間に少なくとも一つの実数解が存在することを

利用して数値解を求める手法である．この手順は以下の通りである．

①初期値x_1，x_2について$f(x_1)$，$f(x_2)$を求める．

②x_1，x_2の中点x_3を考え，$f(x_3)$の符号を考え，$f(x_1)$，$f(x_2)$のうち$f(x_3)$と異符号のものを選び，それを与えるx_1もしくはx_2とx_3の中点x_4を考える．

③以下，同様の手順により計算を行い，n番目のx_nとn+1番目のx_{n+1}の差が収束誤差ε以下になったとき収束し，解が求まったとする．

具体的にはEXa8.Cをベースとした以下のプログラムによって解説する．

ファイル名：EXa9.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>
#include <math.h>

trans_axis(double x, double y)
                              {
                               int tx,ty;

                               tx=x*60+340;
                               ty=y*(-10)+230;

                  if (tx>=140 && tx<=540 && ty>=80 && ty<=380)
                            {
                              PutPoint(tx,ty,4);
                            }

                               }

double func(double x)
                     {
                      double x1,x2,x3,y;

                      x1=x-1;
                      x2=x+1;
                      x3=x-2;
                      y=x1*x2*x3;

                      return(y);

                     }

main()
     {
```

```
double x,y,dx1,dy1,dx2,dy2,sx,sy,err;
int i,px1,py1,px2,py2;

SetGraphicsMode();                              /* Change Mode 12h */

ClearScreen(0);
FillRectangle(140,80,540,380,7);
Line(140,230,540,230,1);
Line(340,80,340,380,1);

for (i=-1000;i<1500;i++)                                /*Draw Graph*/
                        {
                         x=i/400.0;
                         y=func(x);
                         trans_axis(x,y);
                        }

dx1=dx2=dy1=dy2=0.0;

SetCursorPos(1,24);
printf ("x1=");
scanf("%lf",&dx1);
dy1=func(dx1);
px1=dx1*60+340;
py1=dy1*(-10)+230;
Circle (px1,py1,4,0);

while (dy1*dy2>=0)
                   {
                    SetCursorPos(1,25);
                    printf ("x2=");
                    scanf("%lf",&dx2);
                    dy2=func(dx2);
                   }

err=fabs(dx1-dx2);

while (err>0.0001)                                      /*Error value*/
                   {
                    px1=dx1*60+340;
                    py1=dy1*(-10)+230;
                    Circle (px1,py1,4,0);
                    px2=dx2*60+340;
                    py2=dy2*(-10)+230;
                    FillCircle (px2,py2,4,0);
```

```
            sx=(dx1+dx2)/2.0;
            sy=func(sx);

            if (sy*dy1>0)
                        {
                         dx1=sx;
                         dy1=sy;
                        }
            if (sy*dy2>0)
                        {
                         dx2=sx;
                         dy2=sy;
                        }
            if (sy==0)
                       {
                        dx1=dx2=sx;
                       }

            err=fabs(dx1-dx2);
          }

 SetCursorPos(1,27);
 printf ("Solution:x=%lf¥n",sx);
 px2=sx*60+340;
 py2=sy*(-10)+230;
 FillCircle (px2,py2,4,5);

 SetCursorPos(1,28);
 printf ("Push ANYKEY!!");
 getch();

 RestoreMode();                                  /* Restore Mode */
}
```

プログラムを実行すると任意の2つのxを入力するよう求めてくる．このとき，2つ目のx_2による関数の値 $f(x_2)$は1つ目のx_1による関数の値 $f(x_1)$の符号と異なるように選択しなければならない．もし，同符号となる場合，もしくは偶然，解となる値を選択した場合は，条件を満たした値を入力するまでx_2を求めつづける．

条件を満たした場合，選択した2つのxの間に必ず解が存在するので，1つの解が必ず得られる．ニュートン法と同様に，3つの解のうちどれが得られるかは，入力した2つのxに依存する．

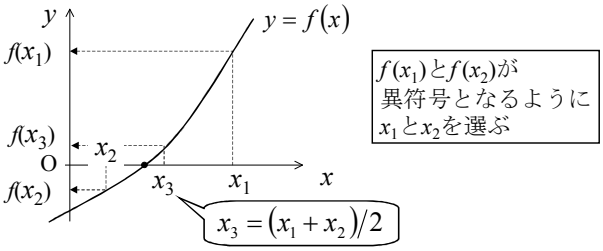


Fig.10.12　二分割法の原理

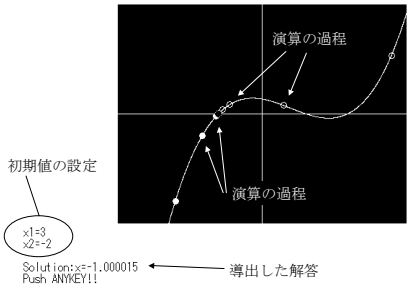


Fig.10.13　二分割法による数値解の算出

10章の課題

▼EXA6.Cは台形則に基づいたプログラムになっている．図式積分によるプログラムに変更し，比較検討せよ．

●EXA7.Cは前方差分商近似に基づいたプログラムになっている．後方差分商近似，中央差分商近似を用いてそれぞれプログラムを記述し，比較検討せよ．

▲ニュートン法では初期値の取り方によって得られる解が異なる．EXA8.Cにおいて初期値を変更して他の解が得られることを確認せよ．

◆二分割法では初期値の取り方によって得られる解が異なる．EXA9.Cにおいて初期値を変更して他の解が得られることを確認せよ．

１１章　外部装置制御の基礎～インターフェースの操作

コンピュータで実験データを測定する場合，各種センサに接続した様々なインターフェースを介してデータを数値化する．したがって，インターフェースにアクセスすればプログラムでデータの測定が可能になる．インターフェースはポートアドレスを設定したハードウェア(電子回路)であり，設定したアドレスのポートにアクセスして，インターフェースとデータを送受信する．

本章では，最も汎用性の高いアナログ入出力機能とデジタル入出力機能について述べる．このために，CONTEC社の非絶縁型アナログ入出力カードAD12-8(PM)を使用することを想定する．外観をFig.11.1に示す．

このカードはアナログ入出力，デジタル入出力を行うPCMCIA Rel.2.0 / JEIDA Ver.4.1以降に対応したTYPEⅡサイズのPCカードであり機種依存性が少ない利点がある．このカードの主な特長は以下の通りである．

①非絶縁シングルエンド8チャンネルのアナログ入力機能(-10～10[V]).
②非絶縁2チャンネルのアナログ出力機能(0～4.095[V]).
③TTLレベルの非絶縁4点デジタル出力機能.
④TTLレベルの非絶縁4点デジタル入力機能.

11.1　アナログ出力

コンピュータは2進数で内部演算処理を行っている．すなわちデジタルで処理を行っている．このデジタル演算の結果を電圧(アナログ量)として出力するためのインターフェースがD/A変換器(Digital / Analogue Converter)である．この原理をFig.11.2に示す．

分解能はbit数で表現する．したがって最小出力電圧幅はD/A変換器の最大出力可能電圧を，分解能で表現できる最大の数で割った値になる．

AD12-8(PM)の場合，分解能は12bitであるので，出力電圧0～4.095[V]とカードに送信するデータの0～4095(=2^{12}-1)が対応する．

モータドライバや流量制御電磁弁などの外部機器を制御する場合，目標値と対応した電圧を送信することがあり，多用されている．

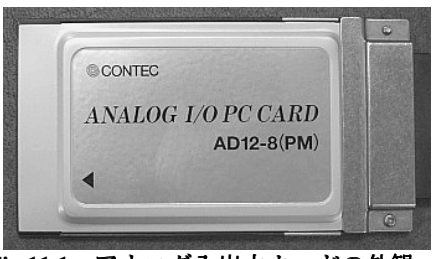


Fig.11.1　アナログ入出力カードの外観

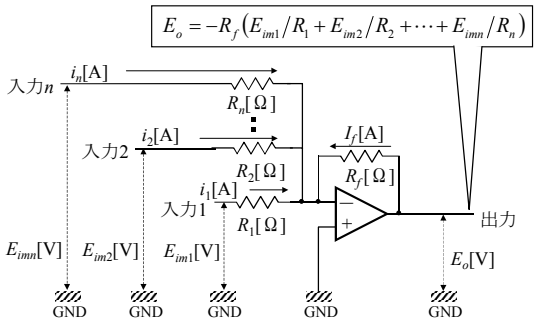


Fig.11.2　D/A変換器の回路構成

11.2　アナログ入力

近年，各種センサや測定装置の出力は電圧に変換されて出力されることが多い．例えば，酸素濃度の0～100[%]が0～10[V]に対応させる場合，濃度21[%]は2.1[V]として出力される．あるいは，圧力センサで100[kPa]=1[V]という設定の場合，1.013[V]の出力を観

測すれば101.3[kPa]の圧力を検出したと分かる．

このように，外部デバイスの電圧による出力データ(アナログ量)をコンピュータで演算処理可能な数値(デジタル量)として計測するためのインターフェースが，A/D変換器(Analogue / Digital Converter)である．

アナログ量は常に変動していると考えられ，変化量も一定ではない．これに対して，アナログ量をデジタル量に変換する符号化のためには若干の時間が必要である．そこで，A/D変換器はサンプリング開始時における測定電圧をサンプルホールド回路で捕らえ，上述した符号化の時間，ホールドした電圧を維持する．これを標本化という．

標本化した電圧をFig.11.4に示すような回路により符号化し，プログラムの変数で取り扱える状態にする．このときの分解能はbit数で表現する．すなわち最小入力電圧幅はA/D変換器の入力可能電圧の範囲を，分解能で表現できる最大の数で割った値になる．

AD12-8(PM)の場合，分解能は12bitであるので，入力電圧-10～10[V]とカードに送信するデータの0～4095(=2^{12}-1)が対応する．

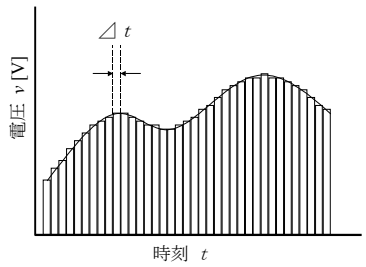


Fig. 11. 3　標本化と符号化

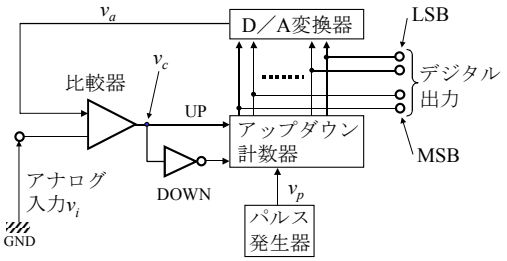


Fig. 11. 4　A/D変換器の回路構成

11.3　デジタル入出力

論理回路によって構成され演算・記憶を行うコンピュータは，基本的にはTTLレベルのON・OFF切替処理による2進数処理を行っているので，電圧を変換を行うアナログ入出力より単純な回路構成でデジタル入出力を行うことができる．

この機能を利用して，計測機器から出力されるデジタル信号を入力したり，複数のPC間の信号伝達などに利用する．また，トランジスタ回路のON・OFFをプログラムによって制御するのに利用する．

11.4　カード型インターフェースの利用

上述のインターフェースはPCカードなので，利用するためにはコンピュータの設定を行わなければならない．

最も重要なのはコンピュータのポートアドレスの割り当てである．ポートアドレスとはコンピュータ内部に割り当てられた入出力用の番地のことで，ハードウェアを操作する場合，この番地に対してデータの書き込み，もしくは読み込みを行う．割り当て可能なポートは機種によって決まっており，また接続しているハードウェアがすでに使用しているポートによっても変更しなければならない．

基本的には，Plug and Playに対応しているので，Windows環境で使用する場合には，自動的にポートアドレスが割り当てられる．この番号を，「マイ　コンピュータ」の中にある「システム」で確認する．詳しくは，インターフェースカードの解説書を参照し，各自のプログラム環境に応じた設定を行う．

本書ではポートアドレスを0x0280に指定した．

以下に，インターフェースカードを利用する際のプログラム例を示す.

最初に，データの測定で最も重要なA/D変換機能の使用プログラムである．入力して動作を確認してみる．なお，ADR0として与えた値はコンピュータごとの設定によって変わる可能性があり，注意を要する.

ファイル名：EXb1.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

#define     ADR0   0x0280

double get_ad(int ch)
      {
       int i,adat;
       double addat[8];

       outp(ADR0+6,0);                  /* Initialize card for A/D convert */
       outp(ADR0+6,1);                              /* Set sampling mode */

       outp(ADR0+7,5);                        /* Set multi channel mode */
                                             /* with internal clock mode */

       outp(ADR0+6,2);                    /* Set sampling clock as 10μsec */
       outp(ADR0+7,99);
       outp(ADR0+7,0);
       outp(ADR0+7,0);

       outp(ADR0+2,7);                        /* Sampling start for 0～7ch */

       while ( inp(ADR0+2) & 3 )
                          {
             if ( inp(ADR0+2) & 2 )
                               {
                    for (i=0;i<8;i++)
                                    {
                        adat=inpw(ADR0);
                        addat[i]=(double)adat*20.0/4096-10.0;
                                    }
                               }
                          }

       outp(ADR0+6,4);                                  /* Stop sampling */
```

```
        return (addat[ch]);

      }

main()
      {
       double volt;

       SetGraphicsMode();                              /* Change Mode 12h */

       ClearScreen(0);

       volt=get_ad(0);

       SetCursorPos (10,10);
       printf ("%lf",volt);


       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                                  /* Restore Mode */
      }
```

多くのA/D変換器は，データのサンプリングの際にプログラム中での設定を必要とする．このインターフェースカードの場合，

①カードの初期化
②サンプリングモードの選択
③クロックモードの設定
④サンプリング間隔の設定

という手順を経て，データのサンプリングを行い，サンプリング終了後はサンプリング停止処理を必要とする．

サンプリングは12bitのデータを読み込む．これは，inpw()関数を用いて，2byte分同時に読み込んでいる．読み込んだデータadatに対して，電圧voltage[V]は以下の式で与えられる．

$$\mathrm{voltage} = \frac{\mathrm{adat} \times 20}{4096} - 10 \qquad \cdots\cdots\cdots\cdots \textbf{(11.1)}$$

カードの初期化とサンプリング停止はプログラムの始めと終わりで行えばよいが，本プログラムでは，これらの手順を関数化し，関数呼出ごとに全8チャンネルでサンプリングを行う．この関数では，呼び出す際にチャンネルを引数として与え，そのチャンネルから読み取った電圧データを戻り値として返す仕様としたが，ポインタの受け渡しにより全チャンネルをサンプリングして，複数の測定を同時に行うように変更することもできる．詳しくはカードの解説書を参照すると良い．

なおoutp()，inp()，inpw()はconio.hで定義された関数であり，それぞれ引数として

与えたポートアドレスに1byteのデータを出力，引数として与えたポートアドレスから1byteのデータを入力，引数として与えたポートアドレスから2byteのデータを入力する関数である．ここで，ポートアドレスから2byteのデータを入力というのは，ポートアドレスとポートアドレス+1のアドレスから1byteずつのデータを読み込み，ポートアドレス+1のデータを上位とする2byteのデータとして取り扱うことである．

次に，D/A変換の実例を示す．

ファイル名：EXb2.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

#define     ADR0   0x0280

void put_da(int ch,double volt)
      {
       unsigned int voldata;
       int j,rt;

       voldata=volt*1000;

       outpw(ADR0+8,voldata);
       outp (ADR0+10,ch+1);

                 for (j=0;j<4;j++)                        /* wait 2 μsec */
                              {
                               rt=inp(0x2ef);
                              }

      }

main()
      {
       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       put_da(0,1.0);                              /* ch0 , 1.0[V] */

       printf ("Push ANYKEY!!");
       getch();

       RestoreMode();                              /* Restore Mode */
```

```
}
```

D/A変換器はA/D変換器と比べて手続きが単純であるものが多い．指定のポートアドレスに出力データを出力することで，実際の電圧が出力される．出力された電圧は次に新たなデータが出力されるまで，その値を維持する．このインターフェースカードでは以下の式により電圧voltage[V]から出力データaoutにデータ変換する．

$$\mathrm{aout} = \frac{\mathrm{voltage} \times 4096}{4.096} \qquad \cdots\cdots\cdots (11.2)$$

出力データを送信手続き後，実際にインターフェースカードにデータが送信され，電圧を出力するまでに2[μsec]必要である．解説書では，DOS/V機の場合，ポートアドレス0x2efから1回データを読み込むことで0.5[μsec]かかることを利用して，時間を調節することを推奨している．

なお，outpw()はconio.hで定義された関数であり，引数として与えたポートアドレスに2byteのデータを出力する．このとき，与えたポートアドレスに下位1byteが，ポートアドレス+1に上位1byteが出力される．

最後に，デジタル入出力機能を用いて任意のbitのTTL出力をON・OFF切替するプログラムを示す．

ファイル名：EXb3.C

```
#include "Graph.h"
#include <stdio.h>
#include <conio.h>

#define     ADR0   0x0280

put_pio(int pout0)
             {
              outp(ADR0+3,pout0);
             }

main()
      {
       int pout0,ch;

       pout0=0;

       SetGraphicsMode();                          /* Change Mode 12h */

       ClearScreen(0);

       printf ("Channel:");
       scanf ("%d",&ch);
```

```
            switch (ch)
                    {
                     case 1:
                              pout0=pout0 ^ 0x01;
                              break;
                     case 2:
                              pout0=pout0 ^ 0x02;
                              break;
                     case 3:
                              pout0=pout0 ^ 0x04;
                              break;
                     case 4:
                              pout0=pout0 ^ 0x08;
                              break;
                    default:
                              break;
                    }

                    put_pio(pout0);

     printf ("Push ANYKEY!!");
     getch();

     RestoreMode();                                  /* Restore Mode */
    }
```

デジタル出力機能も出力時の状態を維持しつづける．この状態を記憶するため，本プログラムでは変数pout0を使用している．0～3の任意のbitを指定すると，そのbitを反転し出力する．

なおC言語で^はExOR演算である．

なお，一般的にデジタル出力機能の方がアナログ出力よりも駆動電流が大きいので，トランジスタのスイッチングなどに利用しやすい．

11章の課題

▼A/D変換によって，任意のセンサ出力を測定し，グラフをリアルタイムで描け．

◆デジタル出力機能を利用して4個のLEDを任意に点滅するプログラムを作成せよ．このとき，LEDをスイッチングするためのトランジスタ回路も考案せよ．

終章　医学系研究者への道

本書は基本的には医学系の研究における機器開発・実験データの測定・演算処理等に必要なソフトウェア開発を行うための知識の習得を念頭に置いた情報科学の実習として作成した．このためにC言語によるプログラムの作成を基礎として，実用上必要と思われる最低限の技術に限定した．したがって，プログラム作成はあくまでも研究の一手段であり，目的ではないと考えた．故にシステムエンジニアやプログラマなど，プログラム作成を生業とすることを目的とする場合に必要な，本格的なC言語の習得を望む場合には他の解説書を参照し，独自の学習を継続することが望ましい．

元々はMS-DOS(あるいはUNIX)上での使用を前提としたC言語も，現在ではWINDOWS上でのプログラムが主流となり，C++へと移り変わっている．さらに，インターネットの普及によりJAVAやPerl,PHPなども開発されている．しかし，これらはいずれもC言語を基本としており，C言語習得者にとっては難しい言語ではない．

いずれの方向を選択するかは各自の環境と，必要性に依るものであり，今後の一層の努力を期待する．

C言語の関数の理解が出来ればプログラムが作れると考えるのは明らかな間違いである．すなわち，日本語が出来れば小説や俳句・短歌などが作れると考えることと同様の間違いであることを指摘しておく．小説を書くときに重要なのが筋立てと演出であるように，プログラムを作るときにはアルゴリズムこそが最も重要なのである．アルゴリズムを考えることこそがプログラマーにとって最も基本であり，同時にあらゆる分野のあらゆる仕事において通用する概念であることを記しておく．

最終課題

【1】並べ変えプログラムを作成せよ.　[30]

フロッピーディスクに入っているRANDOM.DATというファイルで与えられた100個のデータをORDER.DATというファイルに出力する．同時に出力の是非を問い，是の場合，画面の上の方に元のデータを表示し，画面の下の方に並べ変えたデータを出力する．ただし，データは0～999までの整数とし，表示の際にはスペース1個を入れてデータの区切りとする．さらに，分かりやすくするために，各データにはタイトル（元データ，改データ）をつけるものとする.

※この課題の回答例はサンプルプログラムのSORT.Cである．よって，このファイルをコピーした回答例は不許可である.
並べかえのアルゴリズムは，クイックソート／バブルソート／交換ソート／単純挿入法／マージソート／バケットソート…etc.と様々なものが考え出されている.
全員が同じプログラムを考える可能性はかなり低いことを付記しておく.

【2】MS-DOSのTYPEコマンドを作成せよ.　[30]

ただし，画面に表示できるのはテキスト形式のファイルのみとし，出力の際には20行分出力したら一旦停止するものとする.

【3】ある桁について1ずつ加算し，その桁の数字が9になったら，一桁ずつずらしながら同様にして，演算精度の違いを確認せよ．すなわち，1→9の後，9.1→9.9，9.91→9.99，9.991→9.999　…　となる様に続けて計算し，float型とdouble型の違いが確認できるように同時に表示すること　[20]

【4】円周率を求めるプログラムを作成せよ.　[20＋α]

アルゴリズムは任意のものとするが，組み込みの三角関数，もしくはテイラー展開あるいはマクローリン展開により得られた三角関数の等価式を用いる手法は，課題【5】との重複を避けるために，<u>選択しないこと.</u>

【5】sin,cos,tan,log,exp等の関数をマクローリン展開によって近似的に算出せよ.　[各5]

いずれの関数もxの定義域は$-\infty \sim \infty$で成立すること.
提出は1回でまとめて行うこと.（後から追加しないこと）

<u>※以下の【6】～【11】はいずれか一つのみ選択せよ.</u>

【6】素数を見つけるプログラムを作成せよ.　[20]

最低でも10000個の素数を見つけ出すこと.

【7】任意の整数に対して素因数分解するプログラムを作成せよ．　[20]

【8】任意の整数の約数をすべて表示するプログラムを作成せよ．　[20]

【9】5つ以上の完全数を見つけ出すプログラムを作成せよ．　[20]
※完全数とはその数自身を除く約数の和が、その数自身と等しい自然数のことである．

【10】5組以上の婚約数を見つけ出すプログラムを作成せよ．　[20]
※婚約数とは異なる2つの自然数の組で，1と自分自身を除いた約数の和が，互いに他方と等しくなるような数のことである．

【11】5組以上の友愛数を見つけ出すプログラムを作成せよ．　[20]
※友愛数とは異なる2つの自然数の組で，自分自身を除いた約数の和が，互いに他方と等しくなるような数のことである．

【12】社交数を見つけ出すプログラムを作成せよ．　[30＋α]
※社交数とは，ある数(A)の自分自身を除いた約数の和がほかの数(B)になり，(B)の自分自身を除いた約数の和が(C)になる，というように続けた結果，元の数(A)になるような数の組のことである．

【13】任意の入金額で自動販売機で品物を購入したときの，硬貨の種類ごとのおつりの枚数を計算するプログラムを作成せよ．ただし初期状態で販売機内に入っているおつり用硬貨の枚数は乱数で与え，つり銭切れなども考慮せよ．　[20]

【14】ゲームを作る　其の弐　[30＋α]

MASTER MINDを作ること．
MASTER MINDのルールは以下に示す通りとする．

０．**MASTER MIND**とは

別名を**Hit&Blow**とも呼ばれるこのゲームは，一言で言えば４桁の数当てである．ヒントを便りに相手の設定した数字を相手より速く当てることを目標とするこのゲームは，古くからコンピュータを相手とする一人用暇潰しゲームとして，初級プログラマーが一度は作るゲームとなった．

１．ルール

全ての桁の数字が異なる４桁の数をヒントを頼りとして当てるゲームである．
プレイヤーからの回答の正解度を，コンピュータがヒントとして返し，プレイヤーはそのヒントを基にして４桁の数を推理する．
ヒントは以下のルールに従う．

４桁の数のうち，数字とその位置が一致した場合をHitとする．
４桁の数のうち，数字だけが一致した場合をBlowとする．

例えば，０４９２という数を当てるものとしたとき回答とヒントは以下のように対応する．

回答	ヒント
０１２３	１Ｈ１Ｂ
４５６７	０Ｈ１Ｂ
８１３５	０Ｈ０Ｂ
９２０４	０Ｈ４Ｂ

２．プログラムの製作

2.1 解答の作成

乱数の発生はrand()関数を使用する．rand()関数はfloat型の変数fとint型の変数d(変数名は自由に設定してよい)を用いて以下のように記述することで，0～9の乱数を発生させることが出来る．EX59.C参照

```
unsigned long time1;
unsigned seed;

time(&time1);
seed=time1;
srand(seed);
```

変数の型宣言の下に入れる．

```
f=rand();
d=f*10/32768.0;
```

使いたい場所で使用する．
(事前にfloat型のfとint型のdを宣言しておく必要あり)

これを4回繰返し，4桁の数を作成する．作成された数は変数に記録する．このとき，各桁のそれぞれが異なっているという条件を満たす必要がある．

2.2 回答の入力

解答が決定されたら，今度はプレイヤーからの回答を入力する．このとき，プレイヤーがズルをしないように判定する．つまり4桁の数字がすべて異なるように入力されなければならない．
入力された4桁の数字を各桁ごとに判定し，Hit&Blowを数え，表示する．
4Hitが出たら，そこまでの回数を表示する．

【15】ゲームを作る　　其の弐　　[30＋α]

ルールが明確であり，多くの人間が楽しめる任意のゲームを作ること．

ジャンケンのような単純なルールのものは不許可とする．
MASTER MINDから派生する数当て・文字当てゲームは不許可とする．

【16】精度管理で用いる$\bar{x}-R$管理図を描画せよ．　［30＋α］

管理用試料は2回測定し，その平均値を$\bar{x}$とする．$R=|x_1-x_2|$で算出する．
それぞれの管理図において，$\bar{\bar{x}}$および$\bar{R}$を表す中央線と±3*SD*の線を描線せよ．
ただし，$\bar{\bar{x}}$，$\bar{R}$およびそれぞれの±3*SD*はデータ入力ごとに算出して，再描画すること．
トレンドやシフトを自動判定して通知するなどの機能を充実したものは点数を＋αする．

【17】銀行のATMに並んだときの待ち時間をシミュレートせよ．　［40＋α］

結果はリアルタイムでアニメーション表示により視覚化して表すこと．

【18】床面に落ちるボールをアニメーション表示するプログラムを作成せよ．　［30＋α］

ただし，重力加速度は9.8[m/sec^2]とし，ボールと床面との反発係数*e*を0.8とする．
またボールは初速度10.0[m/sec]で高さ20[m]の点から水平方向に射出されるとする．

【19】底に穴の空いたタンクに一定流量で水を流入した場合に変化する水位を計算し，時刻を*x*軸，水位を*y*軸にとってグラフ表示せよ．ただし，タンクの穴から流出する水の量は水位に比例するものとする．　［20］

【20】底に穴の空いた2つのタンクを考え，1つ目のタンクには水道管から水が流入し，2つ目のタンクには1つ目のタンクから流出した水が流れ込むとする．このとき，2つのタンクそれぞれの変化する水位を計算し，時刻を*x*軸，水位を*y*軸にとってグラフ表示せよ．ただし，タンクの穴から流出する水の量は水位に比例するものとする．　［30］

【21】底に穴の空いたタンクに流入する水の量を調節し，任意の水位を維持する場合の流入水量の変化と変化する水位を計算し，時刻を*x*軸，水位を*y*軸にとって1つのグラフに重ねて表示せよ．ただし，タンクの穴から流出する水の量は水位に比例するものとする．　［40＋α］

【22】底に穴の空いた2つのタンクを考え，1つ目のタンクには水道管から水が流入し，2つ目のタンクには1つ目のタンクから流出した水が流れ込むとする．このとき，2つ目のタンクについて，任意の水位を維持する場合の水道管からの流入水量の変化と2つのタンクそれぞれの変化する水位を計算し，時刻を*x*軸，水位を*y*軸にとって1つのグラフに重ねて表示せよ．ただし，タンクの穴から流出する水の量は水位に比例するものとする．　［50＋α］

【23】画面上を自由に動き回る２種類のキャラクターが追いかけっこするシミュレーションプログラムを作成せよ. [30]

パターンⅠ　「カルガモの親子」

親の後をついて回る子供をシミュレートせよ.
親は一定のポイントのまわりをうろついて餌を探す．適当な時間経過後，別のポイントに移動してまたそのまわりで餌を探す.
親が移動すると子供はその後をついていく．親がポイントに到達し，うろつきはじめると子供も適当にそのまわりをうろついて餌を探す.
親の後をついていく順番は不定であり，親に近い順から後をついていく.
子供は5～8羽程度を想定し，起動ごとに異なる結果が得られるようにせよ.

パターンⅡ　「猫と鼠」

獲物を待ち構える猫と適当にうろつき回る鼠をシミュレートし，猫の射程圏内に鼠が入った場合に，猫がダッシュして鼠をつかまえる．鼠は追いかけられた瞬間に猫から逃げ始める．鼠の移動速度を猫の2倍に設定せよ.

【24】球体が回転している様子を表示せよ. [40＋α]

結果はリアルタイムでアニメーション表示により視覚化して表すこと.
球体に光が当たった場合の陰影と特定のポイントとの位置関係によって動きを表現すること.
光源が動いた場合の．影の移動をアニメーション表示すること.

【25】五択もしくは五選択二方式で問題演習を行なうシステムを開発せよ. [40＋α]

問題文の入れ換えを考慮し，問題解答のデータはテキスト形式で保存された別データファイルから読み込んで利用すること.
問題番号をシャッフルして，ランダムな順番で，かつ重複しないように出題されるシステムとすること.
選択肢もシャッフルし，同一の問題であった場合も前回と異なる番号が回答となるように並べ替えること.
入力された回答に対し，正解・不正解とその解説を表示すること.
グラフィック表示などにより，図表問題も取り扱えるのが望ましい.

MS-DOS 必修 拾弐ヶ条

緒言

世の中のOSがWINDOWS一色に染められつつある現在，MS-DOSなどは一部の研究者を除いてほとんど使う者がないといっても良いように思われている．

しかし，MS-DOS無しには語れないのがWINDOWSであり，そのシステムがMS-DOSを内包していることは周知の事実である．

WINDOWSはその性質上，トラブルは避けられないといわれており，実際，最悪の事態（＝システムの崩壊）が起こった場合，システムの再インストール以外に復活させる手段がないこともしばしばである．

このような事態に陥った時，もっとも必要とされるのがパーソナルデータ，すなわち個人的に作成した文書ファイル等のバックアップである．このために，崩壊したシステム上で，最低限のMS-DOSを起動し，MS-DOSモードでのコマンドオペレーションによってバックアップをとる必要がある．不思議なことに，常時バックアップを保存し，崩壊しても全く問題のない人間の使用するシステムは滅多に崩壊しない．

MS-DOS上で複雑な処理をするためには数々の呪文（＝**コマンド**）を覚え，キーボードアレルギーを押さえこみ，立ち向かわなければならない．しかし，トラブルの原因を突き止めシステムを復旧させるために必要なコマンドに限れば，その数は決して多くはない．

ここでは，WINDOWSがトラブルを起こした時，最低限備えておきたいコマンドについて記し，それ以上の知識については専門書を参照していただくこととした．

これらのコマンドは少々難解なイメージを受けるかもしれないが，自らの手で実行してみれば決して難しくないことが分かるはずである．恐れずに経験することで，一つ一つ身に付けていただくことを切に希望するものである．

DIR

ディレクトリ中のファイルとサブディレクトリを一覧表示する.

DIR [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ][ﾌｧｲﾙ名] [/P] [/W] [/A[[:]属性]] [/O[[:]並べ順]]
[/S] [/B] [/L] [/C[H]]

[ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ][ﾌｧｲﾙ名]	一覧表示させるドライブ, ディレクトリ, ファイルを指定する.
/P	一画面ごとに停止して表示する.
/W	ワイド一覧形式で表示する.
/A	指定した属性のファイルを表示する. 属性: D ディレクトリ R 書き込み禁止　H 隠しファイル S システムファイル　A アーカイブ - その属性以外
/O .	ファイルを並べ替えて表示する 並べ順: N 名前順　S サイズ順 E 拡張子順　D 日付順 G ディレクトリ優先　C 圧縮率順 - 逆順
/S	指定されたディレクトリのサブディレクトリ中のファイルも すべて表示する.
/B	ディレクトリ名とファイル名だけを表示する.
/L	小文字で表示する.
/C[H]	圧縮率を表示する. /CH はホストドライブのクラスタサイズを 使う. 環境変数 DIRCMD にスイッチを設定することもできる. たとえば /-W のように- (ハイフン)を前につけると, そのスイッチは無効になる.

CD

現在のディレクトリを表示したり,変更する.

CHDIR [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]
CHDIR[..]
CD [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]
CD[..]

.. 親ディレクトリに変えたいときに指定する.

CD ﾄﾞﾗｲﾌﾞ: と入力すると指定したドライブの現在のディレクトリが表示される.
パラメータの指定がなければ, 現在のドライブとディレクトリが表示される.

COPY

ファイル(複数可)を別の場所にコピーする.

COPY [/A | /B] 送り側 [/A | /B] [+ 送り側 [/A | /B] [+ ...]]
[受け側 [/A | /B]] [/V] [/Y | /-Y]

送り側	コピーするファイル(複数可)を指定する.
/A	アスキーテキストファイルとして扱う
/B	バイナリファイルとして扱います.
受け側	新しいファイルのディレクトリまたはファイル名(複数可)を指定する.
/V	正しくコピーされたかどうか照合する.
/Y	受け側のファイルを上書きするか確認するためのプロンプトを 表示しない.
/-Y	受け側のファイルを上書きするか確認するためのプロンプトを表示する.

環境変数 COPYCMD に /Y スイッチを設定することもできる.
これは, コマンドラインで /-Y スイッチを指定すると無効になる.

複数のファイルを追加するには, 受け側に 1個のファイルを指定し, 送り側に複数の
ファイルを指定(ワイルドカードを使うか, ﾌｧｲﾙ1+ﾌｧｲﾙ2+ﾌｧｲﾙ3+... と指定)する.

FORMAT

指定されたドライブのディスクをＭＳ－ＤＯＳで使えるように初期化する.

FORMAT [d:] [/S|/B] [/V] [/P] [/M|/6|/9|/4] [/U] [/Q]
FORMAT /F|/H|/E

d:	フロッピィディスクまたは３.５インチ光ディスクのドライブ名を指定する.
/S	システムを登録する.
/B	ブランクディスクを作成する.
/V	ボリュームラベルをつける.
/M	１Mバイトのフロッピィディスクを初期化する.
/6	６４０Kバイトのフロッピィディスクを初期化する.
/9	６４０Kバイトのフロッピィディスクを, １トラックあたり９セクタで 初期化する.
/4	1.44Mバイトのフロッピィディスクを初期化する.
/P	キー入力要求メッセージを表示しない.
/U	無条件フォーマットを実行する.無条件フォーマットを実行すると, UNFORMATコマンドによる,データの復旧ができなくなる.
/Q	クイックフォーマットを実行する.
/F	フロッピィディスクのフォーマットのメニューを表示する.
/H	固定ディスクまたは光ディスクの初期化を行う.
/E	固定ディスクまたは光ディスクの初期化を,簡単な操作で実行する.

コマンド起動時にスイッチを省略すると, 装置選択のメニューを表示する.

TYPE

テキスト ファイルの内容を表示する.

TYPE [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]ﾌｧｲﾙ名

DISKCOPY

指定されたドライブ上で, フロッピィディスクまたは３.５インチ光ディスクのバックアップコピーまたは, 照合を行う.

DISKCOPY [[<送り側ドライブ:>][<受け側ドライブ:>][/P][/Q][/V]]

[送り側ドライブ:]	コピー元のディスクをいれるドライブを指定する.
[受け側ドライブ:]	コピー先のディスクをいれるドライブを指定する.
/P	DISKCOPYの次のキー入力を行わなくするスイッチである. ①ドライブへのディスク挿入と確認の要求 ②処理終了時の再実行確認メッセージ
/Q	コピー時に照合を省略して, 高速にコピーする.
/V	照合のみを行う場合に指定する.

送り側と受け側のディスクは同じタイプでなければならない.
フロッピィディスクの場合は送り側と受け側のドライブは同一ドライブを指定することもできる.また,フォーマットしながらコピーすることもできる.

CHKDSK

ディスクをチェックして, 現在の状態を表示する.

CHKDSK [ﾄﾞﾗｲﾌﾞ:][[ﾊﾟｽ]ﾌｧｲﾙ名] [/F] [/V]

[ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]	チェックするドライブとディレクトリを指定する.
ﾌｧｲﾙ名	チェックするファイルを指定する.
/F	ディスクのエラーを修復する
/V	ディスクの全ファイルのフルパスと名前を表示する.

パラメータの指定がなければ, 現在のディスクをチェックする.

このコマンドを使用するとMS-DOS version 6.2では以下のようなメッセージが出る.

『CHKDSK を実行する代わりに, SCANDISK を使ってみてください. SCANDISK の方が, より確実に問題を見つけ, 修正できる問題の範囲も広くなります. 詳しくは, コマンドプロンプトで HELP SCANDISK と入力してください.』

SCANDISK

ディスク修復プログラムを実行する.

ドライブのチェックと修復を行うには, 以下の書式を使うこと:
SCANDISK [ﾄﾞﾗｲﾌﾞ: | /ALL] [/CHECKONLY | /AUTOFIX [/NOSAVE]] [/SURFACE]

ファイルが断片化していないか調べるには:
SCANDISK /FRAGMENT [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]ﾌｧｲﾙ名

以前行った修復を取り消すには:
SCANDISK /UNDO [ﾄﾞﾗｲﾌﾞ:]
[ﾄﾞﾗｲﾌﾞ:]には, Undo ディスクの含まれているドライブを指定してすること.

/ALL	全ローカルドライブのチェックと修復を行う
/AUTOFIX	プロンプトを表示せずに損傷を修復する
/CHECKONLY	ドライブのチェックだけ行い,損傷は修復しない.
/CUSTOM	SCANDISK.INIファイルの設定で ScanDisk を実行する.
/NOSAVE	/AUTOFIXと一緒に使い, 破損クラスタを保存せず削除する.
/NOSUMMARY	/CHECKONLYか/AUTOFIX と一緒に使い,要約の画面で停止しないようにする.
/SURFACE	他のチェックの後にクラスタスキャンを実行する.
/MONO	ScanDiskをモノクロディスプレイで使うために設定する.

現在のドライブのチェックと修復を行うには, パラメータを指定せずに SCANDISK と入力する.

SET

MS-DOS 環境変数の表示,設定または削除をする.

SET [変数名=[文字列]]

変数名　　環境変数の名前を指定する.
文字列　　変数に割り当てる文字列を指定する.

パラメータの指定がなければ, 現在の環境変数が表示される.

MD(MKDIR)

ディレクトリを作る.

MKDIR [ﾄﾞﾗｲﾌﾞ:]ﾊﾟｽ
MD [ﾄﾞﾗｲﾌﾞ:]ﾊﾟｽ

RD(RMDIR)

ディレクトリを削除する.

RMDIR [ﾄﾞﾗｲﾌﾞ:]ﾊﾟｽ
RD [ﾄﾞﾗｲﾌﾞ:]ﾊﾟｽ

DEL

ファイル(複数可)を削除する.

DEL [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]ﾌｧｲﾙ名 [/P]
ERASE [ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]ﾌｧｲﾙ名 [/P]

[ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]ﾌｧｲﾙ名	削除するファイルを指定する. 複数のファイルを指定するには, ワイルドカードを使うこと.
/P	削除する前に確認のメッセージを表示する.

UNDELETE

DEL コマンドによって削除されたファイルを復元する

UNDELETE [[ﾄﾞﾗｲﾌﾞ:][ﾊﾟｽ]ﾌｧｲﾙ名] [/DT | /DS | /DOS]
UNDELETE [/LIST | /ALL | /PURGE[ﾄﾞﾗｲﾌﾞ] | /STATUS | /LOAD | /UNLOAD
/S[ﾄﾞﾗｲﾌﾞ] | /T[ﾄﾞﾗｲﾌﾞ]-[ｴﾝﾄﾘ]]

/LIST	復元可能なファイルを一覧表示する.
/ALL	確認するためのプロンプトを表示せずに復元する.
/DOS	MS-DOS が削除したと記録されているファイルだけを復元する.
/DT	削除追跡に保護されているファイルを復元する.
/DS	削除セントリに保護されているファイルを復元する.
/LOAD	Undelete プログラムを削除保護のためにメモリに読み込む.
/UNLOAD	Undelete プログラム常駐部分をメモリから解放する.
/PURGE[ﾄﾞﾗｲﾌﾞ]	SENTRY ディレクトリ内のファイルをすべて消去する.
/STATUS	各ドライブで有効な保護手段を表示する.
/S[ﾄﾞﾗｲﾌﾞ]	削除セントリによるファイル保護を有効にする.
/T[ﾄﾞﾗｲﾌﾞ][-ｴﾝﾄﾘ]	削除追跡によるファイル保護を有効にする.

REN

ファイルまたはディレクトリ名(複数可)の変更する.

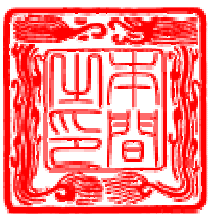
RENAME [ﾄﾞﾗｲﾌﾞ:]ﾊﾟｽ][ﾃﾞｨﾚｸﾄﾘ名1 | ﾌｧｲﾙ名1] [ﾃﾞｨﾚｸﾄﾘ名2 | ﾌｧｲﾙ名2]
REN [ﾄﾞﾗｲﾌﾞ:]ﾊﾟｽ][ﾃﾞｨﾚｸﾄﾘ名1 | ﾌｧｲﾙ名1] [ﾃﾞｨﾚｸﾄﾘ名2 | ﾌｧｲﾙ名2]

受け側用には新しいドライブもパスも指定できないことに注意する.

参考文献

技術評論社	高田美樹著：C言語の総合研究第3版，1993年
西東社	小山清一著：はじめてのCプログラミング，1993年
朝倉書店	平田光穂，小島和夫，栃木勝己著：化学系のための実用数学，1992年
近代科学社	千葉則茂，村岡一信，小沢一文，海野啓明：Cアルゴリズム全科　基礎からグラフィックスまで，1995年
共立出版	谷口慶治，若松秀俊：医用電子・生体情報，1996年
共立出版	若松秀俊，本間達：医用工学―医療技術者のための電気電子工学―，2003年
CONTEC社	AD12-8(PM)解説書，2002年
	小山佳孝　　LSI-C試食版用簡易グラフィックライブラリ説明書，2005年
エル・エス・アイジャパン社	LSI C-86 Ver 3.30 試食版ユーザーズマニュアル，1993年
南江堂	林典夫，廣野治子編集:シンプル生化学2版，1993年

索 引



1998年　7月20日　初版第弐刷発行
2011年　7月31日　拾参版第弐刷発行