基礎だけのＣ言語

三木　邦弘

平成7年 1月 7日

目 次

1　はじめに

平成６年度の三木ゼミではＣ言語だけでなく、ワークステーションを利用した演習も予定しています。平成５度はＣ言語だけで１年間使いましたので、昨年度と同じペースでやっていたのでは間に合いません。内容のレベルを維持しながらスピードアップをするために、昨年度のプリントを元にこの冊誌を作成してみました。プリントでの誤りは直したつもりですが、大幅に説明を増やしているので、新たな誤りが混入しているかもしれません。

本来ならば何かＣ言語の解説の本を買って頂いてそれを元にゼミを進めたいのですが、なかなか適当な本がありません。来年度のプログラミング演習との重なりもあるので、このゼミでは極めて基本的な部分だけにとどめたいのですが、通常の本では一通りＣ言語について述べなければならないためか、余分な説明が多過ぎます。と言うわけで基本的な部分にしか触れられませんので、さらに勉強されたい方は、Ｃ言語に関する本は非常に多く出版されていますので、本屋で適当な本を購入して勉強して下さい。

2　Ｃ言語について

2.1　プログラミング言語とは

コンピュータはプログラムと言う形で与えられた指示を非常に高速に忠実に実行する事ができます。コンピュータが基本的にできることは数値の演算に限られるのですが、それらを高速に多様に組合わすことによって様々な仕事ができます。我々は既に何らかの用途に合わせたプログラムを内蔵したコンピュータをよく利用します。ワープロは文書編集用のプログラムが組込まれたコンピュータで、ファミコンはゲームを実行するためのプログラムが組込まれたコンピュータです。このようにプログラムが組込まれたコンピュータは電源スイッチをオンにするだけで使うことができますが、それ以外の用途には使えません。逆にパソコンを初め通常コンピュータと呼ばれているものは、何か仕事をするためのプログラムは持っていませんが、プログラムを実行するための用意はできているので、仕事をするプログラムを入れてやればその仕事をするようになります。

コンピュータに何か仕事をさせたい時には、それをするためのプログラムが必要です。既に多くのプログラムが市販されていますので、大抵の仕事はそれを購入することにより済ますことができます。しかし仕事の内容によっては、一部だけ市販のプログラムでは合わない部分がある事はよく生じます。また全く対応できるプログラムがないとか、貧乏なのでプログラムを購入できない事もあります。そう言った場合の対処の仕方として自分でプログラムを作成する手があります。

プログラムはコンピュータに何をやらせるか、とか仕事をどのように処理するかを記述したものです。具体的にかつ詳細に論理的に記述するために様々なプログラミング言語が考えられました。各言語はそれぞれが記述しようとするプログラムの対象をある程度想定し、対象が合えば楽に記述ができるようになっています。数値計算ならばFORTRAN、事務用ならばCobol、知識情報処理ならばPrologなどが有名です。もちろんFORTRANで全てのプログラムを記述するのも不可能ではありませんが、向いていない分野のプログラムを書くのは、記述が長くなったり複雑な手法が必要になります。

2.2　Ｃ言語の生立ち

Ｃ言語はUnixと言うオペレーティングシステム（ＯＳ）を記述するために生まれました。ＯＳは基本ソフトとも訳されますが、他の実際に仕事をするプログラムと異なり、コンピュータシステムを管理・運用するためのプログラムで、通常のプログラムはこれの助け無しでは働くことができないという重要なものです。ＯＳを記述するためにはコンピュータの非常に基本的な動作まで記述できる必要もあります。その

ためにＣ言語は他のプログラミング言語では扱うことができないような、コンピュータの生の情報を扱えるようになっています。

Ｃ言語は1972年にアメリカのベル研究所のDennis Ritchieによって開発されました。ただし単独で全く無の状態からＣ言語が作られた訳ではなく、Algol60、CPL、BCPL、Ｂと言う名前のプログラミング言語の系列の最後にできました。当時のベル研究所ではＢ言語を使用してUnixを開発していましたが、幾つかの点で問題があったため、Ｂ言語を拡張する形でＣ言語が誕生しました。それ以降Unixの大部分はＣ言語で記述され、開発されたUnixは教育機関には無料で、他の企業にも極めて安価にて供給されたので急速に普及しました。

この当時のコンピュータは大型汎用計算機と呼ばれるものと、ミニコンと呼ばれる小型計算機の２種類があり、Unixは後者のミニコン用のＯＳとして使われました。これは様々な会社のミニコンがありましたが、UnixがＣ言語で大部分記述されているために、簡単に別のミニコンでもUnixが動くようにできたためでした。

80年代になると、パソコンがより強力になりかつ普及してきました。当初パソコンにおいてはBASIC言語がよく使われました。これは大抵のパソコンに無料で添付されていたためと、初心者向きの簡単な構造を持っていたためと思われます。また業務用等のパソコンの性能を限界まで引き出さなければならない分野ではコンピュータ本来の命令に近いアセンブリ言語が使われて来ました。最近はまず後者のような分野でＣ言語が使われるようになってきました。これはアセンブリ言語では大規模なプログラムの開発が困難である事やＯＳのようなプログラムでも記述できるＣ言語の良さが認められて来たからだと思います。そしてそれにつられるような形で普通のプログラムもＣ言語で書かれる事が多くなりました。

3　学園センターでＣ言語を利用するには

学園センターでＣ言語を利用するには、次のような手順をふむ必要があります。また現在は実習室１でのみＣ言語は利用可能です。以下画面及びキーボードでは￥のところが印刷の都合で全て\になっています。また改行のキーを押すところは△の記号を使っていますので間違えないようにしてください。

1. Ｃの利用環境の設定
   - 電源スイッチを押す。
   - メニューが出たら ESC を押す。プロンプト(A:\>)が表示される。
   - フロッピーを左側のドライブに入れる。
   - a:makeup△ と入力する。ちょっとすると再びプロンプトが表示される。
2. 終了の仕方
   - bye△ と入力する。
   - 電気が切れたらフロッピーを取り出す。
3. プログラムの編集の仕方
   - red ファイル名.c△ と入力する。ファイル名は、英字、数字と一部の記号よりなる８文字以内のもの。
   - 矢印キーで修正したい所へカーソルを移動し、入力や削除を行う。
   - 終了するには、PF １を押し、1(save)を押す。
   - 前回と同じファイルを編集する場合には、red -q△ と入力する。

4. プログラムのコンパイルの仕方

   - lcc ファイル名△ と入力する。

5. プログラムの実行の仕方

   - ファイル名△ と入力する。

［演習］

以下の手順で簡単なC言語で書かれたプログラムをコンパイルして実行してみて下さい。

```
A:\>red test1.c△

  入力するプログラム
  #include <stdio.h>
  main(){
     printf("This is a program.\n");
  }

A:\>lcc test1△
cpp -DLSI_C -IE:\LSIC\INCLUDE -j -o E:1.$$$ test1.c
cf -chut j E:1.$$$ E:2.$$$ E:3.$$$
cg86 -v E:2.$$$ E:3.$$$
main_ _...:;
r86 -o E:test1.obj -m test1 E:3.$$$
lld @link.i

A:\>test1△
This is a program.
```

4　タイプの練習ソフトについて

パソコンの標準的な入力装置はキーボードです。プログラムやデータの入力は基本的にこれを用います。そのためにパソコンを使いこなすには、キーボードで早く入力ができることが必須の技能となります。欧米ではコンピュータができる以前よりタイプライターが存在し、タイピストと呼ばれる専門家以外でも多くの人がこれを利用してきました。このことが欧米でのパソコンの普及を容易にしている一因と言われます。日本では最近こそワープロが普及していますが、ほとんどの利用者がキーボードのキーを探しながらキーを押しているような状況です。

理想的にはキーボードを全く見ないで打てるのが良いのですが、キーの押しにくい周辺のキーは確認して押すレベルでも、全く練習していない状況よりはましです。キーボードを全く見ない状況とは、キーの位置を全て憶えている状態ですが、いちいちキーの位置を意識の上で考えるのではありません。条件反射でキーを思うだけで反射的に指が動くようにするのです。

4.1　タイプ練習ソフトの起動方法

タイプの練習ソフトは各自のフロッピーに入っています。起動する際には、前節の「Cの利用環境の設定」と同じ事をします。その後で、

```
A:\>mikatype△
```

と入力します。するとプログラムが起動されて表題の画面になりますので、もう一回改行キーでも押しますと、メニューが出てきます。これ以降はやりたい項目の先頭に書いてある番号を入力するだけです。

練習が終わりましたら、前節の「終了の仕方」と同じ事をします。その前に続けてＣ言語の入力や実行などをしてもかまいません。

タイプ練習用のプログラムはフロッピーに入っているので、実習室１以外のパソコンでも実行可能です。

4.2　タイプの練習の進め方

やはりピアノが上手とか、運動神経に自信のある方は上達が早いようです。しかし普通のひとならば誰でも練習に応じて上達します。まだ20歳にもならない若さを信じて頑張ってください。

進め方は基本的に最初に表示されるメニューの順番に従って下さい。まず１番の「ポジション練習」でどの指でどのキーを押すのか憶えます。「ポジション練習」のメニューでは、やはり１番から順番にやってください。画面上部に押すべきキーが表示されますので、最初のうちはその下のキーボードの図や、その下のそれを押すべき指の表示を確認しながら、その通り押してください。両手にそれぞれ５本の指が無い人以外は必ずそれに従って下さい。

「ポジション練習」は１回につき60個のキーの練習ができます。終わったところで、まだ足りない人は改行キーをおせば続けて練習ができます。終わりたい人はESCキーを押せばメニューに戻ります。練習を続ける際は、１回終わる度に少しパソコンのディスプレイから目を離して休憩を取りましょう。

「ポジション練習」で１つマスターできた様ならば、「ランダム練習」に同じメニューがありますので、そちらに挑戦して見てください。とりあえず目標を50文字／分以上としてください。目標をクリアーできたら次のキーの位置を「ポジション練習」で憶えて下さい。

「ランダム練習」の全てのメニューで目標をクリアーできた方は、「英単語練習」のメニューに進んで下さい。ここでは「基本英単語練習」から「８０８６アセンブラ練習」まであります。大文字ばかり出てくるものもありますが、そのままキーを押せば済みます。これらに関しては100文字／分以上を目標にして、全てのメニューをクリアーしてください。

さらに「ローマ字練習」もありますので、余裕のある方は挑戦してみて下さい。間違って入力した場合、最初の子韻の部分なのか母韻の部分なのかわかりにくいので苦労するでしょう。また１文字当たり平均２つのキーを押さなければならないので、記録は「ランダム練習」の約半分になるでしょう。

ここでは目標を速度にします。もちろん「ポジション練習」以外でキーボードを見ながらやってはいけません。反射神経育成のために少々間違えてもどんどんキーを押してください。

メニューの「成績」のところでこれまでの記録や練習時間を見ることができます。と言うことは、ちゃんと練習をしたけれども遅い人と、さぼっているので遅い人との区別ができると言うことです。

4.3　練習ソフトの問題点

このタイプ練習プログラムはフリーソフトです。作者の指定した条件を守れば、このように無料で配布しても著作権法等に触れる心配はありません。このプログラムは様々な機種のパソコンでも動作可能なように配慮して作ってありますので、本来の対象機種はＮＥＣのＰＣ９８シリーズですが、学園センターのＦＭＲでも利用できます。画面に若干の乱れが見られますが大きな問題ではありません。

そのような利点の反面、この練習ソフトでは文字と数字キーのみを練習の対象としているので、プログラムの入力の際に必要な記号の入力の練習ができない欠点があります。英文にしろ和文にしろ入力の際には最低句読点の入力は必要ですので、改善が望まれます。

5　基礎だけのＣ言語

5.1　Ｃ言語のプログラムの形

今の段階ではＣ言語のプログラムは必ず次のような形をしているものと憶えてください。

```
#include <stdio.h>

main(){
  文がいくつか
}
```

最初の#includeはコンパイラが持っているstdio.hと言うファイルをここで読み込む事を指示しています。このファイルの中には通常のプログラムに必要な様々な関数や定数の定義が入っています。

main(){はここからmainと言う名前の関数の定義が始まることを示しています。関数についてはそのうち詳しく説明します。Ｃ言語のプログラムには必ずmain と言う名前の関数の定義が存在し、プログラムを実行するとこの関数が最初に実行されます。

その次の「文がいくつか」の部分には様々なＣ言語で言う文が;で区切られて並びます。原則的に書いた順番に実行されます。どのような文があり、どのような働きをするのかを憶えないとプログラムは書けません。しかしそれほど種類はありませんから、憶えることには問題は生じません。いかに組合わせれば目的とする動作をするのか考えるのが難しいのです。

最後の}を忘れてはいけません。必ず{ }や( )は対になっています。この} はmainの定義の終わりを意味します。

5.2　printfの使い方

コンピュータは別名電子計算機とも呼びますので、計算の仕方からやっても良いのですが、計算の結果を画面に出す方法が判らなくては正しく計算できたのかどうかも判りません。Ｃ言語ではprintf()と言う関数を文字列や計算結果の出力に用います。printf()の全ての機能を一度に憶えるのは大変ですので、少しずつ分けて説明します。

まず一番簡単な形としてprintf(”何でもＯＫ”)があります。( )の中に” ”で囲った文字列を入れるとその入れたものだけが、ほぼそのまま画面に表示されます。ここで「ほぼそのまま」と言ったのは、\や%については特殊な意味があるので、そのままの形ではでないからです。とりあえずここでは\nだけ憶えてください。\nがあると画面上のカーソルが次の行の先頭に動き、以後の文字列は次の行の先頭から表示されるようになります。（通常これを改行と呼びます。） ３章の演習では、printf(”This is a program.\n”)となっていたので、This is a program.と表示された後で改行がなされています。

［演習問題］

画面に次のような物を表示するプログラムを作りなさい。(hello.c)

以後の演習問題にも( )の中にプログラムを入れるファイル名を指定しますので、できるだけそれに従って下さい。

```
A>hello△
*********
* HELLO *
*********
```

5.3 整数型変数

コンピュータの基本的な機能は数値の計算です。複数の値の計算や複雑な計算式の実行には、途中の計算値を記憶する機能が必要です。そのためにどのようなプログラミング言語でも変数と呼ばれるものが利用できるようなっています。

整数型変数は整数を記憶することができます。パソコンのC言語では通常-32768から+32767の範囲の値が記憶可能です。複数の変数を使い分けるために変数には全て名前（変数名）が付いています。変数名は英字、数字、下線(_)、で８文字程度以下にします。変数名の先頭は英字でなければばりません。

一部の言語を除き通常のプログラミング言語では、変数を利用する前に宣言をしなければなりません。C言語の場合、int a,b,c; と宣言するとa, b, cと言う名前の整数型変数が利用可能になります。

実際の変数の利用は次のような感じになります。a=3; 変数aに３を入れる。a=3+5; 加算、a=5-3; 減算、a=5*3; 乗算、a=15/3; 除算、a=16%3;剰余（余り）、b=3; a=b+3; aに６が入る、a=3; a=a+3; aに６が入ります。最後の例では=の両側にaがあります。同じaですが左側のは結果を入れる場所、右側のは変数の値を意味しています。

こうして変数に入れた値を表示するにはprintfの次のような機能を使います。

```
printf("\%d",a);  aに入っている値が表示される。
```

%はこの後に出力形式の指示があることを示しています。dは10進数を示します。さらに、printfの出力形式の指定の際にきます。%4dとなっていれば、４桁以下の数値の出力の際には数字の左側に空白が補われます。

［演習］

次のプログラムを入力し、実行してみよ。(printf.c)

```
#include <stdio.h>

main(){
   int i,j;
   i=3;j=5;
   printf("i+j=%d  i-j=%d \n",i+j,i-j);
}
```

上記のプログラムを実行すると、

```
i+j=8  i-j=-2
```

のように表示されます。このように変数の代わりに式を書いても良いし、複数の値を一度に表示する事もできるし、%d以外の部分はそのまま表示されます。

5.4 for文

for文はC言語で非常によく使われるもので、処理のくり返しを表現する代表的なものです。その形式と動作は次のようになります。

```
for ( A ; B ; C ) D ;
```

まずAを実行する。Bの条件を調べて、だめならば、for文の実行を終えて次の文へ。OKならばD、そしてCを実行して、ふたたび条件判定のところへもどる。このような動作をします。具体例は次のようになります。

```
for (i=0;i<100;i++) printf("%d\n",i);
```

ここでi++と言う表記が出てきましたが、これは変数iの内容を１増やすと言うC独特の式です。まず変数iの値がゼロになります。条件判定は変数iの値が100未満か？となっているので、変数iの値が100以上になったらおしまいです。100未満ならば、iの値を表示して改行する。そしてiの値を１増やして条件判定にもどる。この繰り返しになります。

注意点として、Dの部分が複数の文からなる場合には{ }で囲う必要があります。次の例で左側はHaHaHaとCyaCyaCyaがそれぞれ交互に10個出ますが、右側ではHaHaHaが10個出た後に１回だけCyaCyaCyaが出ます。

```
for (i=0;i<10;i++) {
   printf("HaHaHa\n");
   printf("CyaCyaCya\n");
}
```

```
for (i=0;i<10;i++)
   printf("HaHaHa\n");
printf("CyaCyaCya\n");
```

［演習］

次のプログラムを入力して実行してみよ。ただし実行する前にどのような結果が得られるかよく考えてみること。例えば最初のfor文では何が得られるか？次のfor文ではcatを出している様だがどのような形に並ぶか？予想と異なる結果が得られた場合にはよくその原因を考えること。(for.c)

```
#include <stdio.h>

main(){
   int i;
   for (i=1;i<10;i++) printf("%d %d %d\n",i,i*i,i*i*i);
   for (i=0;i<400;i++) printf("cat ");
}
```

［演習問題］

１．以下の図形を縦横５匹？ずつ計２５匹表示するプログラムを作れ。(tanuki.c)

```
 0 0
(o.o)
=( x )=
 U U
```

２．次のような九九の表を出力するプログラムを作れ。(kuku.c)

```
A:\> kuku△

   ***  The Multiplication Table  ***

   1   2   3   4   5   6   7   8   9
   2   4   6   8  10  12  14  16  18
       中略
   9  18  27  36  45  54  63  72  81
```

5.5 MS–DOS (1)

通常のパソコンは、基本ソフトウェアとしてMS–DOSを使用しています。これは、キーボードやディスプレイの管理の他に、フロッピーやハードディスクのファイルの管理を行っています。通常のプログラムはこのMS–DOSによってディスクからメモリーに読みだされ、MS–DOSの助けを借りながら実行されます。

以下の表はそのコマンド（命令）の幾つかの例です。

A> dir	Aドライブにあるファイル名の表示。
A> type test1.c	ファイルtest1.cの内容を表示する。
A> copy test1.c abc	ファイルtest1.cをabcと言う名前のファイルにコピーする。
A> ren abc def	ファイルabcをdefと言う名前に変更する。
A> del def	ファイルdefを消去する。

［演習］

MS–DOSのコマンド例を実行してみよう。コピー、名前の変更、削除が確実に行われたかどうかdirコマンドで確認すること。

5.6 条件判断(if)

コンピュータは計算だけでなく判断することが可能です。もちろん人間のように玉虫色なファジーな判断は無理ですが、yesまたはnoの論理的な判断をし、その結果によって異なる文を実行することができます。

```
if（条件）A;
```

条件が成立したときのみAが実行されます。

```
if（条件） A; else B;
```

条件が成立するとAが実行されて、そうでない時にはBが実行されます。AとBともに複数の文からなる場合には で囲みます。条件としては、

x<5	xの値が５より小さい。	x==5	xの値が５と等しい。
x!=5	xの値が５と等しくない。	x<=5	xの値が５と等しいか少ない。
(x==3) || (x==5)	xは３に等しいか５に等しい。		
(x>5) && (y<3)	xは５より大きく、かつyは３より小さい。		

２つの値を比較するのが基本で３つの数が等しいかどうかを判断する際には、２つづつ比較するようにしなければなりません。つまりx、y、zがみな等しい条件は、x==y==zではなく、必ず(x==y) && (y==z)と書かねばなりません。

5.7 入力用関数(scanf)

キーボードから入力した数値などを変数に入れるためには、scanfと言う関数が使えます。

```
int i;
scanf("%d",&i);
```

&は、アドレスを求める演算子です。&iで、変数iが実際にメモリーのどこにあるかを示します。scanfは入力されたものを10進の数値とみなして解釈し、与えられた変数の位置（アドレス）にそれを書き込みます。

［演習］

以下のプログラムを入力し、実行してみよ。(hantei.c)

```
#include <stdio.h>

main(){
   int h,w;

   printf("Height (cm) = ");scanf("%d",&h);
   printf("Weight (Kg) = ");scanf("%d",&w);
   if ((h-100)*0.9>w) printf("You are smart!\n");
    else              printf("You are too heavy!\n");
}
```

scanf関数自体は何も画面に出力を行わないので、この例のようにscanfの前にprintfを使用して、何を入力しようとしているのか示すのが普通です。

［演習問題］

1. 身長と収入を尋ねて、それぞれ170と1000以上ならば、I love you.と答えるプログラム。条件に合わない場合には適当なメッセージを出す事。(2kou.c)

```
A>2kou△          A>2kou△                        A>2kou△
Height=177       Height=150                     Height=190
Income=1500      Income=2000                    Income=600
I love you.      Bye bye, you are too small.    Bye bye, you must work hard.
```

2. ３つの数値を入れたら、その中で最大のものを答えるプログラム。(max3.c)

```
A>max3△          A>max3△          A>max3△
A=5              A=7              A=7
B=6              B=3              B=12
C=7              C=4              C=8
Max=7            Max=7            Max=12
```

3. ｘの字で直角三角形を表示するプログラムを作れ。その大きさは入力して指定する。(3kaku.c)

```
A>3kaku△
Size=8
x
xx
xxx
xxxx
xxxxx
xxxxxx
xxxxxxx
xxxxxxxx
```

5.8 MS–DOS (2)

MS–DOSで使用できるファイル名の形式は、ファイル名の本体として半角で８文字までと、ファイル名の拡張子として半角で３文字までが使用できます。拡張子は省略する事が可能です。拡張子のうちのいくつかはMS–DOSによって特別な意味を持っています。またプログラムによってはある特定の拡張子を持ったファイルのみを扱うものがあります。

MS–DOSにとって特別な意味を持つ拡張子は、exe、com、batです。また通常C言語で書かれたプログラムはcと言う拡張子の付いたファイルに保存する事になっています。

通常は何かファイルに対して処理を行う場合には、それぞれのファイル名を個別に指定するのですが、ワイルドカードと呼ばれる複数のファイルを一度に指定する方法があります。

```
A>copy *.c b:
```

これは拡張子がcである全てのファイルをBドライブにコピーします。

```
A>del test1.*
```

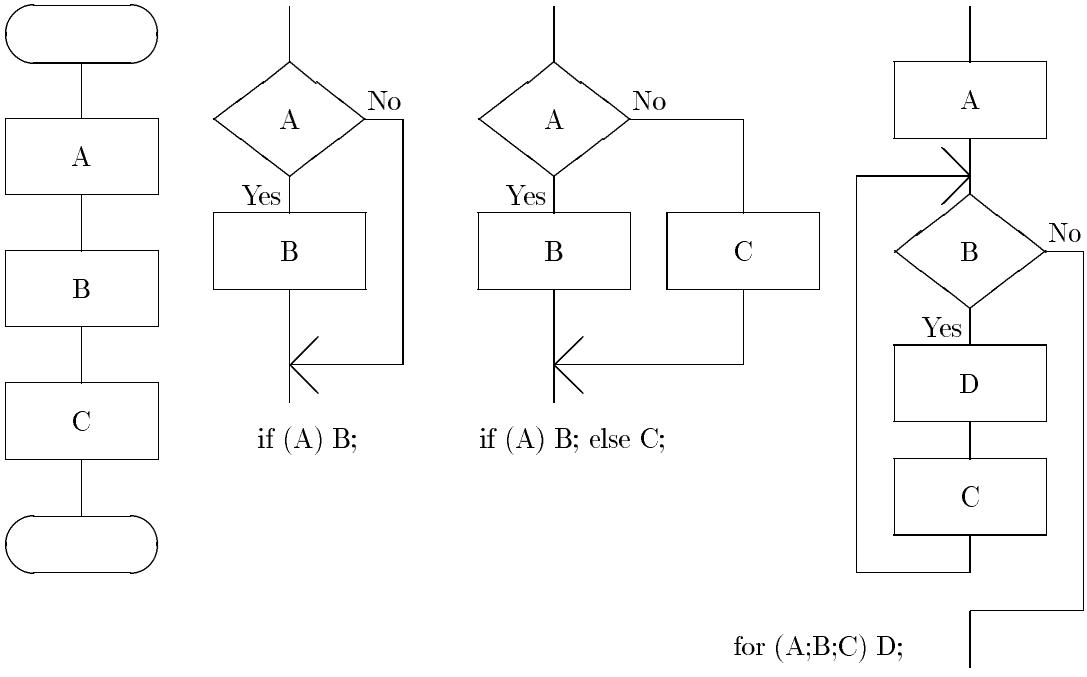
これでファイル名の本体がtest1であるファイルは全て消去されます。

［演習］

自分のフロッピーにある拡張子がbakであるファイルを全て消去して、その確認をせよ。通常拡張子がbakであるファイルは、エディタで編集する前の古いファイルの内容を保存しているファイルにつきます。

5.9 フローチャート（流れ図）

フローチャートはプログラムの構造を図示するのに用いられます。プログラムの最初と終わりを長丸、処理を長方形、判断をひし形で表します。



必ずしもC言語の１つの文を１つの箱に対応させる必要はありません。通常はプログラム全体の大きな流れを数個の箱で書き、次にその箱の１つ１つを別のフローチャートでより詳しく記述するような事が行われます。

［演習問題］

前回の課題のmax3.cのプログラムをフローチャートに直せ。

5.10 ループ (1)

コンピュータの処理能力を生かすにはできるだけ大量のデータを処理してもらうことが重要です。数値の計算にしても僅か数個の数値を計算するのならば、プログラムを書くよりも電卓を利用した方が速いでしょう。しかし数百個のデータを加えて平均を求めるような場合に、プログラムに数百回加算する文を書くのでは大変です。そのような場合には、その規則性に注目して少ない文で同様な働きをするプログラムを作成するのが普通です。

くり返しの回数があらかじめ決まっているような場合には、既に学んだfor文を使用するのが普通ですが、さらにちょっとした手法が必要になります。以下のプログラムは１から与えられた数までを全て加えるものです。

```
#include <stdio.h>

main(){
   int i,n,s;

   printf("N = ");scanf("%d",&n);
   s=0;
   for (i=1;i<=n;i++) s=s+i;
   printf("Sum = %d\n",s);
}
```

［演習問題］

1. ｎの階乗(n!)を計算するプログラムを作れ。なおn!=1*2*3*...*(n-1)*n である。(kaijo.c)

   ```
   A>kaijo△
   N=5
   N!=120
   ```

2. １からｎまでの奇数の総和を求めるプログラムを作れ。(kisuwa.c)

   ```
   A>kisuwa△
   N=15
   1+3+5+7+9+11+13+15=64
   ```

5.11 while文

forと同様に繰り返しを行うもので、先に条件判定するものと、後で行うものがあります。

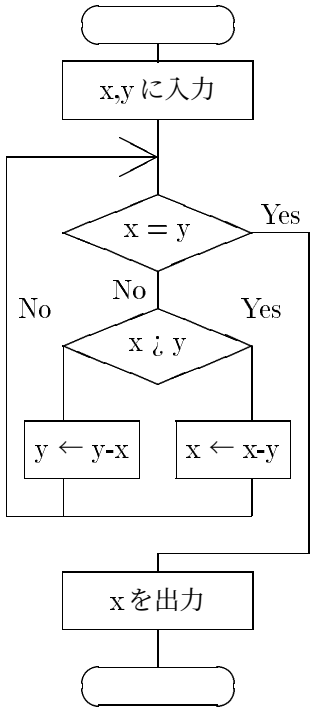
```
while (A) B;                do A; while (B);
```

左側のwhileではまずＡの条件を調べます。ＯＫならばＢを実行します。そして再びＡの条件を調べます。Ａの条件がＯＫの間Ｂを繰り返し実行します。実はこれはfor(; A;) B;と同じ働きになります。

右側のwhileはまずＡを実行します。それからＢの条件を調べます。条件がＯＫならば再びＡを実行します。Ｂの条件がＯＫの間Ａを繰り返し実行します。先程のwhileと異なるのは、条件に係わらずこちらのwhileは最低１回の実行がなされることです。左側のwhileでは条件が最初から駄目な場合には１回も実行されません。

［演習問題］

1．フローチャートに基づいて最大公約数を求めるプログラムを作れ。(gcd.c)




```
A>gcd△
A=12
B=15
G.C.D.=3
```

2．任意の自然数に対して、偶数ならば 2 で割る、奇数ならば 3 倍して 1 を足すと言う操作を繰り返すと、必ず 1 になることが知られています。実際に入力した数に対して同様な操作を行うプログラムを作れ。(test231.c)

```
A>test231△
x = 5△
x = 16
x = 8
x = 4
x = 2
x = 1
```

5.12 ループ (2)

以下は任意の数のデータの最大値を求めるプログラムです。データの個数が予め判っている場合には for を使って繰り返すほうが簡単です。ここでは正のデータを入力して行って、最後に数値のゼロを入力してプログラムにデータの終わりを知らせるようにしています。

```
#include <stdio.h>

main(){
   int i,m;

   m=0;
   do {
      printf("Data=");scanf("%d",&i);
      if (i>m) m=i;
   } while (i>0);
   printf("Max = %d\n",m);
}
```

［演習問題］

1. 上記のプログラムをフローチャートに直せ。
2. 平均値を求めるプログラムを作れ。ただしデータは全て正の値でデータの終わりを示すのに 0 を用いるものとする。(heikin.c)

```
A>heikin△
Data=20
Data=30
Data=10
Data=40
Data=0      (データの終わりを示す)
Mean=25     ((20+30+10+40)/4)
```

5.13 コメント

プログラムの説明等をプログラム中に埋め込むことができます。/* と */で囲みます。例は次の break 文の例を見てください。

5.14 break 文

break を実行すると一番内側のループ (for でも while でも) から抜け出る事ができます。これを利用した際には、ループの次に実行が移るのが、正常にループを終了した場合と、途中で break で抜けてきた場合の 2 通りになるので注意が必要です。

```
/*     素数の判定プログラム     */

#include <stdio.h>

main(){
   int i,s;                  /*  iには割ってみる数、sには素数かどうか調べる数が入る。 */

   printf("?=");scanf("%d",&s);
   for (i=2;i<s;i++)         /*  2からs-1までの数で割ってみる。 */
      if ((s%i)==0) {        /*  もしsがiで割れたとすると。。。 */
         printf("%dは素数ではない。%dで割れる。\n",s,i);
         break;              /*  forを中断する。  */
      }
   if (i==s) printf("%dは素数である。\n",s); /* forを中断して出てきたときはi<sである。*/
}
```

［演習問題］

1000以下の素数を表示するプログラムを作れ。(s1000.c)

```
A>s1000△
   2   3   5   7  11  13  17  ...
  73  79  83  89  97 101 103  ...
 179 181 191 193 197 199 211  ...
 ...
```

5.15 配列

同じ型の変数を複数個、配列と言う形で扱う事ができます。その場合宣言の際に必要な個数を要求しなければなりません。例えば、int a[10]; とすると、a[0], a[1], a[2], ... a[9] と言う変数が計 10 個使えるようになります。

次の例は配列を使った入力されたデータを入力した逆順に出すプログラムです。データの入力の部分を見ると 1 つの scanf() で全てのデータの入力が可能になっています。

```
#include <stdio.h>

main(){
   int i,n,data[100];

   printf("How many data? = ");scanf("%d",&n); /*  データの個数を入力する */
   for (i=0;i<n;i++) {                         /*  データを配列に読み込む */
      printf("data=");scanf("%d",&data[i]);
   }
   for (i=n-1;i>=0;i--)                        /*  データを逆順に出力する */
      printf(" %d",data[i]);
   printf("\n");
}
```

[演習問題]

上記のプログラムを改良し、データの終わりをゼロで示すことにし、最初にデータの個数を入れなくても良いようにせよ。(reverse.c)

```
A>reverse△
data=5
data=8
data=9
data=0
 9 8 5
```

5.16 文字型

文字データを扱うために文字型の変数があります。宣言は int の代わりに char を用います。普通の文字型変数 1 つには 1 文字しか入りません。複数の文字からなる文字列を入れるためには文字型の配列を用いる必要があります。文字列の最後には必ず'\0'（ヌル文字）が入ります。例えば word という文字型の配列に"bat"と言う文字列を入れると、word[0] には'b'、word[1] には'a'、word[2] には't'、word[3] には'\0' が入ります。

```
#include <stdio.h>

main(){
   int i,l;

   char word[30];                          /*  最大30文字入れる事ができる。          */
   printf("Word = ");scanf("%s",word);
                        /*  文字列の入力の時は%sにして、配列の [ ] の前の部分だけ書く */
   for (l=0;word[l]!='\0';l++) ;           /*  変数lの値を文字列の終わりまで増やす    */
   printf("Word Length=%d\n",l);
   for (i=l-1;i>=0;i--)                    /*  単語を逆順に出力する                  */
      printf("%c",line[i]);                /*  文字を出すときには%cを用いる          */
   printf("\n");
}
```

［演習問題］

1. 入力した単語の中に母音(a,e,i,o,u)が幾つ含まれるか計算するプログラムを作れ。(vowels.c)

```
A>vowels△
Word=computer△
There are 3 vowels in 'computer'.

A>vowels△
word=big△
There is a vowel in 'big'.
```

2. 入力した単語の中でもっとも短いものの長さを表示するプログラムを作れ。(shortest.c)

```
A>shortest△
Word=book△
Word=are△
Word=personal△
Word=today△
Word=Z△          ← 一番最後はZを入れるが、これは考えない。
The shortest word length is 3.
```

5.17 文字列を扱う関数

C言語にはいくつかの文字列を扱うための関数が標準で用意されています。今sとtを文字列変数（char s[100],t[100];のように宣言した物）とします。

```
gets(s);      sに1行丸ごと取り込みます。
puts(s);      sの内容を表示して改行します。
strcat(s,t);  sに入っている文字列の後にtに入っている文字列を追加します。
strcmp(s,t);  sに入っている文字列とtに入っている文字列を比較します。結果は数値です。
    strcmp("abc","def") → 負の値    strcmp("abc","abc") → ゼロ
    strcmp("def","abc") → 正の値
strcpy(s,t);  tに入っている文字列をsにコピーします。
strlen(s);    sに入っている文字列の長さを求めます。
```

C言語では文字列を代入する事はできません。通常s="abc";はできないのでかわりにstrcpy(s,"abc");を使います。

注意：strcat, strcmp, strcpy, strlenを使用する際には、プログラムの先頭に `#include <string.h>`が必要です。

以下は上記の関数を使用したプログラムの例です。wordと言う文字列の配列を利用しています。この例では最大20文字が入るword[0]からword[29]まで30個の文字型変数が使えるようになります。

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

main(){
   int i;
   char word[30][20],sentence[80];

   printf("Sentence = ");gets(sentence);
   printf("Sentence Length = %d\n",strlen(sentence));
   strcat(sentence,"!!!");
   puts(sentence);
   i=-1;
   do {
```

```
      i++;
      printf("Word = ");gets(word[i]);
   } while (strcmp(word[i],"end")!=0);
   for (i--;i>=0;i--) printf("%s ",word[i]);
   printf("\n");
}
```

［演習問題］

1. 入力した文から単語を取り出すプログラムを作れ。（単語間にはスペースが必ず１つだけあり、行末には何もつかないものとする。）(getword.c)

```
A>getword△
Sentence = I have a book△
Word = I
Word = have
Word = a
Word = book
```

2. 文中に指定した文字列が含まれているときにyesと答えるプログラムを作れ。(search.c)

```
A>search△
 Sentence = I have a book.△
 String = oo△
 yes
```

5.18　switch文

Ｘの値がＡだったらこうして、Ｂだったらああして。。。と言った事をしたい時には、ifを何個も書く必用がありましたが、このような場合にはswitch文と言うものが使えます。

```
switch（X）{
   case A : こうして; break;
   case B : ああして; break;
   default : その他する;
}
```

Ｘの部分には任意の式が書けます。ＡやＢの部分には、数値とか文字を書きます。ここには変数を含む式は書けません。「こうして」の部分には複数の文が{ }で囲わなくても書けます。その後のbreakは通常必要ですが、無くともエラーにはならず、「こうして」をやった後で「ああして」もするような動作になります。Ｘの値がＡでもＢでもなかった場合にはdefault以降の「その他する」の部分が実行されます。そのような物が必要でない場合にはdefaultの行は省略可能です。

5.19　関数の定義のやり方(1)

Ｃ言語では自分で好きな関数を定義して利用することができます。と言うか数十行を越えるプログラムは、１画面に収まる程度の長さの関数に分割して作成するのが普通です。

これまでどのプログラムもmain()...と言う形をしていました。実はこれはmain()と言う関数を定義していた事になります。ですからtekitou()と言う関数を定義したい時にはmain()同様に行えば良いのです。

```
tekitou(){
   tekitou()の定義
}
```

```
main(){
   main() の定義
}
```

main() の内部で tekitou() を利用したいときには、tekitou(); と言う行を書けばできます。何度利用してもかまいませんからプログラム中に繰り返し用いられる部分を関数として定義すると、プログラムの長さを短く、見易くすることができます。さらに tekitou() の中で別の関数を利用することもできます。この場合利用される関数の定義が利用する関数の前に書くことを薦めます。

tekitou() の中で変数の宣言をしてそれを使うことができます。ただしそこでどのような事を行っても他の関数で宣言された変数の値は変化しません。例えば、

```
tekitou(){
   int i;
   i=100;
}

main(){
   int i;
   i=3;
   tekitou();
   printf("i=%d\n",i);
}
```

のような場合、main() で最初に 3 が変数 i に入ります。そして tekitou() の中では変数 i に 100 を入れてますが、main() にもどるとそれとは関係無く、画面には i=3 と表示されます。このような関数の中でしか通用しない変数の事を局所変数（ローカル変数）と呼びます。逆に大域変数（グローバル変数）と呼ばれるものもあります。関数の定義の前に宣言した変数はそれ以降の関数の中で利用することができますが、どこかの関数でその値を変更するとその変更は他の関数に対しても有効になります。

```
int i;     /*  関数の定義の外にあるので、大域変数  */

tekitou(){
   i=100;  /* 変数 i は既に宣言されているので、すぐ使える  */
}

main(){
   i=3;
   tekitou();
   printf("i=%d\n",i);
}
```

この場合には i=100 と表示されます。各関数の定義の際に int i; が無いことに注意して下さい。もし int i; をそのまま残していると、ローカル変数の宣言の方が優先されます。

関数を利用する側から関数に値を渡すことができます。例えば、

```
tekitou(int height){
   if (height>170) printf("You are tall!\n");
}

main(){
   tekitou(155);
   tekitou(165);
   tekitou(175);
   tekitou(185);
}
```

のような形になります。最初の行の( )の中のintが変数の型を示し、その次が変数名です。tekitou()の中ではここで指定した変数がローカル変数と同様に使えます。複数の値を渡したいときには、型と変数名の後にカンマを入れて型と変数名を繰り返します。上のプログラムでは最初の呼び出しで155がheight に入り、次の呼び出しで165がheightに入ります。よって画面には2回You are tall!が表示される事になります。

［演習問題］

数回にわけて3目並べのプログラムを作成します。今回はまず丸ペケを画面に表示する部分を作成します。毎回少しずつ作っていくのが難しくなるので、初回は関数に分割しない例を出して、これを関数を使って短く完成させてもらいます。（と言っても50行ぐ らいはかかるよ）

```
#include <stdio.h>

main(){
   int i,j,k,ban[10];

   printf("\n  ***  Tick-Tack-Toe Game  ***\n");
   printf("        ver.1 Man - Man\n\n");
   for (i=1;i<=9;i++) ban[i]=0; /* ban[i]がゼロならばiマスは空         */

   j=1;                          /* jが１ならばoの番、－１ならばxの番 */
   for (i=0;i<9;i++) {
      do {
         printf("==> ");scanf("%d",&k);
      } while ((k<1) || (k>9) || (ban[k]!=0));
      ban[k]=j;
      j=-j;
      printf("\n      |   |\n");
      switch(ban[1]) {
         case -1 : printf("    X |"); break;
         case  0 : printf("    1 |"); break;
         case  1 : printf("    0 |");
      }
      switch(ban[2]) {
         case -1 : printf(" X |"); break;
         case  0 : printf(" 2 |"); break;
         case  1 : printf(" 0 |");
      }
      switch(ban[3]) {
         case -1 : printf(" X\n"); break;
         case  0 : printf(" 3\n"); break;
         case  1 : printf(" 0\n");
      }
      printf("      |   |\n");
      printf("   ---+---+---\n");
      printf("      |   |\n");
      switch(ban[4]) {
         case -1 : printf("    X |"); break;
         case  0 : printf("    4 |"); break;
         case  1 : printf("    0 |");
      }
/*   途中省略   */
      printf("      |   |\n\n");
   }
}
```

関数を使わないと大変長くて何をしているか判らない物になります。ここでintialize()、input()、display()と言う関数の定義を適当に作れば、main() の部分は次のようになります。

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

   initialize();      /*  メッセージの表示とban[ ]の初期化  */
   j=1;

   for (i=0;i<9;i++) {
      input(j);       /*  空いているところに入力  */
      j=-j;
      display();      /*  ban[ ]の表示   */
   }
}
```

となるようにします。かなりわかりやすいでしょう？以下はその実行例です。いずれコンピュータが人間の相手をするように直しますが、今回のは人間対人間です。(ox1.c)

```
A>ox1△

  ***  Tick-Tack-Toe Game  ***
       ver.1 Man - Man

==> 1△
       |   |
    0  | 2 | 3
       |   |
   ----+---+----
       |   |
    4  | 5 | 6
       |   |
   ----+---+----
       |   |
    7  | 8 | 9
       |   |

==> 5△
       |   |
    0  | 2 | 3
       |   |
   ----+---+----
       |   |
    4  | X | 6
       |   |
   ----+---+----
       |   |
    7  | 8 | 9
       |   |
```

以下省略。

5.20　関数の定義のやり方 (2)

関数本来の働きと言えば、与えられた値に対して何か値を返すことです。C言語の関数は数値だけでなく、文字なども返すことができます。そのやり方を以下に説明します。まず関数定義の先頭に返す値の型を書きます。何も返さない場合にはvoidと書きます。(前回の課題のように何も型を書かない場合には、整数が返されるものと解釈されます。)

関数から値を返す場合にはreturn(...);を使います。この( )の中に入っている値が関数の値になります。なおこのreturnが実行されると、関数の実行は終わってこれを呼び出した方へ実行が戻ります。

```
int abs(int i){              /*   abs() は整数の値を返す関数    */
   if (i>=0) return (i);     /*   i が正ならばそのまま返す      */
     else    return (-i);    /*   i が負ならば符号を変えて返す  */
}

main(){
   printf("abs(3)=%d\n",abs(3));
   printf("abs(-3)=%d\n",abs(-3));
}
```

関数は他の関数を呼び出せるだけでなく、自分自身を呼び出すことも可能です。例えばnの階乗（n!=n*(n-1)*(n-2)*(n-3)*...3*2*1）と呼ばれる値は以前 for を利用して計算しましたが、以下のように関数を使っても計算可能です。

```
int fact(int n) {
   if (n>1) return (n*fact(n-1));
      else  return (1);
}

main(){
   printf("6!=%d\n",fact(6));
}
```

このように関数が自分自身を呼び出すことを再帰呼び出しと言います。

5.21 プログラムの挿入

次の演習問題は前の演習問題で作成した関数をそのまま利用します。その場合２つのやり方があります。includeを使用する方法と別々にコンパイルしたものを結合する方法です。後者はまた後に説明します。前回の課題の関数の部分がox1.cに入っているとすると、今日の課題のプログラムの中に、

```
#include "ox1.c"
```

と言う指示を入れておくと、入れた所にox1.cの内容が挿入されます。これまで通常のプログラムの先頭には#include <stdio.h>と言う行がありましたが、基本的には全く同じ動作になります。違いは利用者が独自に作ったファイルを挿入する場合にはファイル名を” ”で囲み、C言語のシステムが持っているファイルを挿入する場合には< >で囲む所だけです。ちなみにここで使っているstdio.hファイルは177行もあります。

［演習問題］

1. まず前回の課題の関数の定義の部分だけ取り出して、ox1.cと言う名前で保存します。この時、先頭の#include <stdio.h>、大域変数の宣言、main()の定義の部分は含めないようにします。

2. 念のために前回の関数の動作確認をします。ox2.cと言うファイルに以下の内容を入力し前回同様に動作することを確認してください。

```
#include <stdio.h>
大域変数の宣言
#include "ox1.c"
前回のmain()の定義
```

3. 次のような改良を行います。まずinitialize()の後で盤面を表示させる様にします。それからowari()と言う関数を定義します。これはOかXがどこかに3つ並んでいないかどうか調べてもしOが3つ並んでいれば、「あなたの勝ち」と表示して1を返します。もしXが3つ並んでいれば「あなたの負け」と表示して1を返します。どちらも3つ並んでいない場合には、0を返すようにします。main()に手を加えて盤面を表示してからこの関数を呼び、勝負がついた場合はそこでプログラムが終了するようにします。ox2.cにはowari()の定義部分とmain()の定義が入ります。

5.22 式の値

C言語においては、関数だけでなくプログラムを構成するほとんどの要素が値を持ちます。例えばj=1は1と言う値を持ちます。それをさらに利用することも可能でj=k=1とすると、変数kに1を代入した結果の値1を変数jにも入れることになります。

forの中によく使われるi++に似た物として++iと言うものがあります。どちらも変数iの内容を1増やす働きがありますが、前者の値は1増やす前の変数iの値であり、後者の値は1増やした後の変数iの値になります。　owari()==1の結果は0又は0以外の数になります。条件が成立した場合に0以外になります。ifはこの値によって判断するので実は if (owari()==1) ... はif (owari()) ... と同じ事になります。

逆にこのあたりが災いしてif (ban[1]=1) ... なんて間違えると、ban[1]=1はban[1]の値にかかわらず1になりますので、... の部分が必ず実行されることになります。さらにban[1]の値まで変化するので大きな被害が出ることがありますが、C言語では文法に従った正しい記述とみなされます。

5.23 乱数

コンピュータのプログラムは同じデータを与えると同じ結果が得られるのが普通です。しかしゲーム等では、コンピュータ側がいつも同じ手順で仕掛けてくるのでは面白くありません。大抵のプログラミング言語では、呼び出すたびに毎回異なる値を返す関数を用意しています。

C言語ではrand()と言う関数がそれにあたります。rand()は呼び出す度に0から32767迄の適当な値を返します。通常はもう少し狭い範囲の数が必要となります。例えばさいころの目の代わりをさせるには、1から6までで十分です。以下はさいころを20回振るプログラムです。

```
#include <stdio.h>
#include <stdlib.h>       /*  rand() を利用する際には必要  */

main(){
   int  i;

   for (i=0;i<20;i++)
      printf("%d ",(rand()%6)+1);  /* a % b でaをbで割った余りが求まる  */
   printf("\n");
}
```

このようにrand()の値を6で割って余りを求めると0から5までの数になるので、1を加えると1から6までの値になります。本当にでたらめな数列の事を乱数と呼びますが、このrand()の返す値は内部でそれらしくなるように計算した値なので疑似乱数と呼ばれます。実際上記のプログラムを実行すると、それらしき1から6までの数が20個出ますが、もう一回プログラムを実行するとまた同じ物が出てきます。

```
A>ran△
3 5 4 2 2 6 1 4 1 1 2 6 1 1 6 6 4 4 4 1
A>ran△
3 5 4 2 2 6 1 4 1 1 2 6 1 1 6 6 4 4 4 1
```

これでは困ることが多いので、通常は、時刻を得る関数とrand()の計算の元になる数を設定する関数srand()を利用して、プログラムの起動時刻によって異なる系列の乱数が得られるようにします。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>     /*  clock()を利用する際には必要。   */

main(){
   int  i;

   srand(clock());  /*  現在の時刻の値をsrand()に与える。   */
   for (i=0;i<20;i++) printf("%d ",(rand()%6)+1);
   printf("\n");
}
```

```
A>ran△
5 2 2 6 3 6 2 2 2 3 2 2 6 4 4 4 5 1 4 6
A>ran△
4 5 6 4 2 5 3 3 3 5 6 6 2 1 4 3 3 2 6 6
A>ran△
4 3 5 6 5 3 6 1 1 2 6 3 2 4 4 3 2 2 5 4
```

［演習問題］

1. まず前回作成したox2.cをox3.cにコピーします。（MS–DOSのコマンドを憶えていますか？）それからox2.cの関数の定義の部分だけ残して保存します。逆にox3.cから関数の定義部分を消去します。念のためにここでプログラムのの動作確認をします。

2. まずXの順番の時に、適当に空いた所にXを入れる関数umeru()を作ります。これはrand()を利用して１から９の値を求めて、対応する位置が空いていればそこにXを入れるものです。以前作成したinitialize()の中にsrand(clock());の行を追加しておきます。またmain()を少し直して、Oの番ならばinput(j);を呼び、Xの番ならばumeru()を呼ぶようにします。

3. OやXが２つ並んでいる時には、空いた所にXを入れるtomeru()を作ります。tomeru()はXを入れた場合には０を返し、何もしなかった場合には１を返すようにします。main()もまた少し直して、tomeru()をやってみて、何もなければ先程のumeru()を実行するようにします。

4. umeru()やtomeru()の定義部分はox3.cに入れてox3.cの内容を提出してください。

注意事項：関数が返すものに合わせて、void、int等を関数の定義の先頭に必ず入れるようにして下さい。ifの（）の中の条件に誤ってj=1なんかを書く例が非常に多いようです。条件の中に＝が単独で表れることは普通にはありませんので御注意ください。

5.24　プリプロセッサ

Ｃ言語のコンパイル（Ｃ言語で書かれたプログラムを実際の機械で動く形に変換すること）において、特徴的なものとしてプリプロセッサによる処理がまずなされる事があげられます。これはlcc ox3と入力した際に

```
cpp -DLSI_C -IE:\LSIC\INCLUDE -j -o E:1.$$$ ox3.c
```

とまず表示されますが、最初のcppが今利用しているＣコンパイラーのプリプロセッサのコマンドです。オプションの説明は省略しますが、ox3.cを処理してE:1.$$$へその結果を保存するような指示になっています。

プリプロセッサは２つの仕事をします。１つは#includeで指定したファイルを読み込んで、合体することです。もう１つはマクロの展開です。Ｃ言語で言うマクロは文字列の置き換えですが、置き換えが定義されているかどうかで判断する機能もあり、結構複雑な働きをします。ここではその基本的な所だけ説明します。

まずマクロの定義の仕方ですが、

```
#define MACRO_NAME MACRO_BODY
```

のような形をしています。これより後のプログラム中にMACRO_NAMEが表れると、全てMACRO_BODYに置き換えられます。例えば、

```
前略
#define WEIGHT_LIMIT   100
中略
if (w>WEIGHT_LIMIT) printf("やーい、デブ\n");
後略
```

ですと、wの値が100以上だとメッセージが出ます。通常複数の箇所で参照される値で、あまり変更されることがないものをこのようにマクロで記述します。そして#defineの部分はプログラムの先頭に置かれるので、変更も容易です。通常の変数と異なることを示すために通常マクロ名は大文字で書かれます。

前回の解答でa(1,2,3)とするとban[1]+ban[2]+ban[3]を返す関数がありました。このような単純な関数は次のようなマクロでも記述できます。

```
#define a(x,y,z)  ban[x]+ban[y]+ban[z]
```

関数として定義する場合とマクロとして定義する場合の得失ですが、関数にするとマクロにしたときよりコンパイルして得られる物が小さくなります。しかし若干の実行速度の低下を招きます。

［演習問題］

1. まず前回作成したox3.cをox4.cにコピーします。それから次の点に関しての変更をox4.cに対して行います。ox4.cの内容とox1.cやox2.cの変更した部分のみ提出して下さい。

2. 前回のプログラムでは必ず人が先行でしたが、最初に「先行ですか？(y/n)」とメッセージを出して、人がyと答えた場合のみこれまで通りとし、そうでない時にはコンピュータ側が先行になるようにせよ。

3. 関数umeru()を改良し、中心が空いていればかならず中心を埋め、そうでなく四隅のどれかが空いていれば、その中からランダムに選んで埋め、そうでもなければ残りの中からランダムに選んで埋めるようにせよ。

4. #defineを使って、プログラム起動時のメッセージの２行目が変更できるようにせよ。

5.25　コンパイルの手順

Ｃ言語のコンパイル（Ｃ言語で書かれたプログラムを実際の機械で動く形（機械語）に変換すること）では、まずプリプロセッサによる処理が行われます。その後の処理はだいたいどのプログラミング言語でも同じ様なステップを踏みます。１．構文解析、２．アセンブリプログラムへの変換、３．アセンブル、４．ライブラリーとの結合編集です。演習で用いているLSI-Cはこのような方式でやっていますが、コンパイラーによっては２．と３．を合体してアセンブリプログラムを生成しないものもあります。

アセンブリプログラムは、機械の個々の命令に対応するコマンドを持ったプログラミング言語です。これを機械語に直すプログラムをアセンブラーと呼び、アセンブラーを実行することをアセンブルと言います。

ライブラリーには既に機械語に変換されたC言語が標準で持っている関数（例えばprintfやscanf）が入っています。４．でこれらの関数を利用しているプログラムはライブラリーから必要な部分を取り出して３．の出力と結合して１つのプログラムにします。

演習で用いているLSI-Cでは１．はcfコマンド、２．はcg86コマンド、３．はr86コマンド、４．はlldコマンドで実行されます。先週説明したcppコマンドを含めて５つのコマンドを自動的に呼び出してくれるのが、lccコマンドです。

長くて多くの関数の定義からなるプログラムのほんの一部だけ直した場合にも毎回５つのコマンドを呼ぶのは時間がかかります。そこでプログラムを関数ごとに分割して３．の出力の段階のものを保存しておき、手直しした所だけ１．から３．を行い４．で全体を結合して作る事によりコンパイルの時間を短縮することができます。

5.26　キーボード入力関数

scanf()を使用して文字、文字列、数値をキーボードから入力する事ができました。しかし、毎回入力の最後にはリターンキーを押さなければなりません。また一度入力待ちになると、先に進むことができなくなります。特にゲーム等ではこれでは困ることがあります。

`#include <conio.h>`にはkbhit()と言う関数とgetch()と言う関数が定義されています。kbhit()は数値を返す関数で、現在キーボードで何かのキーが押されていると０以外を、何も押されていないと０を返します。またgetch()はキーボードから１文字だけ文字を入力してその文字を返します。（もしまだ何もキーが押されていない場合には押される迄待ちます。）

［演習問題］

プログラムを起動すると”Hit any key!”と表示され、何かキーを押すとサイコロの目がランダムに表示されるものをrand()を使用せずに作れ。(dice.c)

1. まず指定した目を出す関数disp()を作る。（disp(1)とすると・が表示される）
2. 変数の値が１→２→… →６→１… と変るようにし、変える度にキー入力を調べて何か押されていたらdisp()を呼ぶようにする。

注意事項：関数の定義の際にはまず呼び出し元に返す値の型を書きます。disp()の場合、何も返すものが無いのでvoidになります。disp()には、サイコロのどの目を出すか呼び出し側が指定する必要があります。これが関数に渡す値なので、関数定義の際には( )の中にその型と渡された値を入れる変数を指定します。disp()の場合、数値ですのでintで例えばiと言う変数に入れるようにします。

呼び出し元からもらった１から６の値にしたがって異なる目を書かねばなりません。ifを６つ並べるより、switchを使うのが普通でしょう。disp()の中身はこれだけで済みます。

kbhit()やgetch()を使う際には`#include <conio.h>`をお忘れなく。

5.27　間違い探し

以下は昨年度の学生が犯した典型的な誤りです。ちゃんとわかりますか？

1. 演習問題の１番、最初に「先行ですか？(y/n)」とメッセージを出して、人がyと答えた場合のみこれまで通りとし、そうでない時にはコンピュータ側が先行になるようにせよ。と言う問題に対してケーコさんは以下のようなプログラムを書いた。誤りや直した方が良いところを２つ指摘せよ。

```
main(){
   int  i,j;
   char a;
... 中略...
   if (a='y') j=1;
   if (a='n') j=-1;
... 後略...
```

2. 演習問題の２番、関数umeru()を改良し、中心が空いていればかならず中心を埋め、そうでなく...の部分をクマさんは以下のようなプログラムを書いた。誤りを２つ指摘せよ。

```
void umeru(){
   int i;
   if (ban[5]==0) ban[5]=-1; break;
... 後略...
```

3. 演習問題の１番をサッチャンは以下のようにした。誤りを３つ指摘せよ。

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

   initialize();
   display();

   printf("先行ですか？(y/n)");scanf("%c",word);
   if (word="y") j=1;
      else       j=-1;
... 後略...
```

5.28 分割コンパイル

C言語は関数毎にコンパイルして後で結合して１つのプログラムにすることが可能です。これによって修正した関数のみコンパイルすればよくなり、大規模なプログラム開発に対応できるようになります。

今回の演習で利用しているLSI-Cでは次のように行います。まず、結合前までコンパイルする方法ですが、例えばscreen.cのオブジェクトファイルscreen.obj（結合する前段階のファイル）を作成するには、

```
A>lcc -c screen△
```

と入力します。複数のファイルをコンパイルする場合にはscreenの後に空白で区切って複数のファイル名を一度に指定できます。こうして作成したオブジェクトファイルdice2.objとscreen.objを結合させるには、

```
A>lcc dice2.obj screen.obj△
```

と入力します。この時の指定の順番は自由ですが、一番最初に指定したものの名前が結合してできるプログラムの名前になります。（この場合dice2と入力するとプログラムが実行されるようになる。）

5.29 MS–DOSのエスケープシーケンス

画面の任意の位置に文字を出したり、色を付けたりする方法の一つにMS–DOS のエスケープシーケンスと呼ばれるものを利用する手があります。これはエスケープコードを出した後に決められた文字列を続けて出力すると、カーソルを移動したり文字に色を付けることができます。C言語の場合プログラムにこのエスケープコードを書くことができないので\033のように書きます。033は８進数で書いた文字のコードで10進数ならば27になります。これらの機能を使い易いように関数としてまとめると以下のようになります。

```
#include <stdio.h>

#define black   30    /*  color(black);  とすると以後黒い字になる。    */
#define red     31    /*  color(red);    とすると以後赤い字になる。    */
#define green   32    /*  color(green);  とすると以後緑の字になる。    */
#define yellow  33    /*  color(yellow); とすると以後黄色い字になる。  */
#define blue    34    /*  color(blue);   とすると以後青い字になる。    */
#define purple  35    /*  color(purple); とすると以後紫の字になる。    */
#define cyan    36    /*  color(cyan);   とすると以後水色の字になる。  */
#define white   37    /*  color(white);  とすると以後白い字になる。    */

#define ON      0     /*  cursol(ON);  とするとカーソルが表示される。   */
#define OFF     1     /*  cursol(OFF); とするとカーソルは見えなくなる。*/

void locate(int x, int y){    /* x は１から８０桁、y は１から２４行で  */
   printf("\033[%d;%dH",y,x); /* 指定した位置にカーソルが移動する。     */
}

void color(int n){            /*  以後指定した色になる。                */
   printf("\033[%dm",n);
}

void cls(){                   /*  画面を消去し、色ももどす。            */
   printf("\033[2J\033[0m");
}

void cursol(int f){           /*  カーソルを表示させたり、消したりする */
   printf("\033[%dv",f);
}
```

［演習問題］

1. 上の関数の定義をscreen.cと言うファイルに入力し、この後にmain()の定義を追加して関数のテストをします。テストの内容としては、画面消去をし、中央付近にカーソルを移動し、自分の好きな色（白と黒を除く）を指定し、適当なメッセージを出すものにします。

2. サイコロの目によるスロットゲームを作ります。プログラムをスタートすると画面が消去され中央付近にくるくる変化するサイコロの目が３つ表示されます。何かキーが押されると左端のサイコロの目が停止し、もう一回押すと真ん中のサイコロの目が停止し、さらにもう一回押すと右端のサイコロの目も停止してプログラムも終了するものとします。(dice2.c)

   (a) １．で作成したscreen.hの中のmain()...の部分を削除する。そしてdice2.cの先頭でこれを#includeする。

   (b) 前回作成したdisp()を改良する。disp(1,3,10)とすると、１の目を３行目の10桁目から表示するようにする。

   (c) 前回の課題では最後に１回だけ呼んでいたdisp()を今回は何度も呼ぶ事になる。停止したサイコロは書き直さないようにすること。

注意事項：動くプログラムができたら、もう少し短くならないか、簡単にならないか考えるようにしましょう。（locateを18個も書かないように。）

5.30 分割コンパイル(2)

分割コンパイルを利用する際に問題になるのは、分割コンパイルする各部分でコンパイラーが必要な情報を得られるようにする必要があることです。たとえば、先週の課題をscreen.cの部分とdice2.cの部分に分けたとします。(この時dice2.cの先頭にある`#include "screen.c"`を除きます。)これで、

```
A>lcc -c screen△
```

とするとscreen.objはできますが、同様にdice2.objを作ろうとすると、2種類のエラーが出ます。1つめはlocate()やcolor()などの関数が未定義と言うものと、blackやONなどのdefineで定義したものが未定義と言うもので、後者のためにコンパイルは途中で終わってしまいます。

これを避けるために通常はscreen.cをscreen.hとscreen.cに分割します。前者には#defineや関数の形の宣言を入れます。後者には実際の関数の定義部分が入り、その先頭で前者を#includeします。関数の形の宣言は以下のような形でします。

```
void locate(int x, int y);
void color(int n);
void cls();
void cursol(int f);
```

ちょうど関数の頭の部分だけのものです。こうしてできたscreen.hをdice2.cの中に#includeするとdice2.objを作ることができるようになります。

[演習問題]

1. 上の説明に従ってscreen.cやdice2.cを修正し、screen.hを作成し分割コンパイルを試みよ。
2. 先週のdice2.cのmain()の定義を直してgame()とします。これは5ゲームして、その総得点を返す関数とします。得点は3つのサイコロの目がそろったら10点、2つのサイコロの目がそろったら3点とします。(dice3.c)

   ヒント：実際は先週のmain()の内容はonegame()と言う別の関数にして、game()の中からこれを5回呼ぶ形にした方が良いでしょう。また、コンパイルするときには、

```
A>lcc -c screen△              /*  screen.cを変更したときに必要 */
A>lcc -c dice3△               /*  dice3.cを変更したときに必要  */
A>lcc dice3.obj screen.obj△   /*  何か変更があれば必要         */
```

注意事項：よく見られるまちがいは、i==j==kと言う条件です。気持ちは解りますが、通常のプログラミング言語では3つの数を一度に比較することはできません。この場合は、(i==j) && (j==k)と書きます。

それから関数から値を返すときにはreturnを用いる事は理解しているようですが、今回の問題においてサイコロの目が何も得点をもたらさなかった時にゼロを返すのを忘れないで下さい。

5.31 分割コンパイル(3)

プログラムを分割して分割コンパイルを利用すると、コンパイル時間を節約できますが、いちいち修正したファイルのみコンパイルして、結合を行うのは面倒です。特に分割したファイルの数が何十に及ぶ大規模なプログラムや、includeしているファイルが多数存在すると面倒では済まなくなります。そこで通常はmakeと言うプログラムを使います。これはmakefileと言うファイルに、ファイル間の関係やコンパイルのコマンドを記述しておけば、自動的にファイルの修正日付を調べて必要なコンパイルのみを実行してくれます。

例えば前の演習問題のように、screen.h, screen.c, dice3.cの３つのファイルがあり、これからscreen.obj, dice3.objを生成し、さらにこの２つからdice3.exeを作成する場合には次のような内容のmakefileを作ります。

```
dice3.exe : dice3.obj screen.obj
        lcc dice3.obj screen.obj    # 先頭の空白の部分はTABキーで入れること
dice3.obj : dice3.c screen.h
        lcc -c dice3                # 先頭の空白の部分はTABキーで入れること
screen.obj : screen.c screen.h
        lcc -c screen               # 先頭の空白の部分はTABキーで入れること
```

最初の行の意味はdice3.exeを作るにはdice3.objとscreen.objが必要と言うことです。もしdice3.exeがなければこの２つが存在するかどうかmakeが探します。もしdice3.exeがあれば、これが作られた日時とdice3.objやscreen.objの作られた日時を比較します。dice3.exeの方が新しければ何もしませんが、そうでない時やもともと存在しない時は２行目のコマンドを実行します。ところがその時dice3.obj等が存在しなかったり、これがdice3.cやscreen.h等よりも古かった場合には４行目のコマンドが先に実行されます。このような判断をmakeが全て行ってくれますので、利用者はmakefileさえ作っておけば、後はmakeを実行する（単にA>make△と入力する）だけになります。

5.32 ファイルの読み書き (1)

ゲームプログラムなどを除き、有用なプログラムはファイルの読み書きができることが最低必要です。例えば保存できないワープロソフトでは困ります。ここではファイルの読み書きの基本を説明します。

ファイルからデータを読み込むときには、まずfopen()を実行します。この時にファイルを読むのか書くのかの指定もします。正常にファイルを開くことができるとこの関数はファイルディスクリプタへのポインターを返します。（ファイルディスクリプターはファイルに関する様々なパラメタを記憶している所の事です。ポインターについては後で詳しく説明を行いますが、この場合記憶している場所を示すものの事です。ファイルを操作するにはここに記憶しているパラメタが必ず必要となります。）その後はこの返された値を元にファイルを操作します。ファイルからデータを読む場合にはfscanf()等を使います。これはscanf()とほぼ同じ動作ですが、キーボードから読み込む代わりにファイルから読み込んでくれます。全てのファイルを読み込んだ後は、fclose()を必ず実行します。次は数値を１つdataと言う名前のファイルから読み込むプログラムの例です。

```
#include <stdio.h>

main(){
   FILE *fp;     /*   ファイルディスクリプターへのポインターを入れる変数の宣言    */
   int i;

   fp=fopen("data","r");           /*   ファイル名(data)と読みだし(r)          */
   fscanf(fp,"%d",&i);             /*   最初にfpが入るだけで後はscanf()と同じ  */
   printf("Read Data is %d.\n",i);
   fclose(fp);                     /*   使い終わったら必ずこれを呼ぶこと       */
}
```

逆にファイルへデータを書き込むときには、fopen()でファイル名と書き込みを指定して、fprintf()等でデータを書き込み、最後にfclose()を行います。以下はキーボードから入力した数値をdataと言うファイルに書き込むプログラムです。

```
#include <stdio.h>

main(){
```

```
   FILE *fp;
   int i;

   fp=fopen("data","w");     /*   ファイル名(data)と書き込み(w)          */
   printf("Number = ");scanf("%d",&i);
   fprintf(fp,"%d\n",i);     /*   最初にfpが入るだけで後はprintf()と同じ  */
   fclose(fp);
}
```

［演習問題］

1. dice3.exe作成用のmakefileを作成し、正しくコンパイルされるかどうか確認せよ。

2. 前回のdice3.cのmain()の定義の部分を消します。そしてdice4.cに新しいmain()として、何回もゲームができるようなものを入れます。（ゲーム終了時にまだやるかどうか確認をする。）もし総得点がこれまでの最高点だった場合には、名前が登録できるようにします。（makefileも直してやろうね。）

注意事項：１回ゲームをして得られた得点は、表示しなければならないし、これまでの最高点と比較もしなければなりません。複数回参照しなければならない値は、変数に保存するのが当然でしょう。得点を得るために何回もgame()を呼んでは、その度にゲームをしなければなりません。

if文はif (条件) 実行部; で１つの単位になります。条件が成立したら実行部を行う働きをします。この時if (条件) ; 実行部; とすると実はifは条件の後の;で終わっており、実行部にあたる部分は条件にかかわらず行われます。ご注意下さい。

もう一回ゲームをするかどうか問い合わせてその答えを得る方法は、３つあります。scanf(”の様に文字列として読み込む方法とgetch()を用いる方法です。問題を生じるのは一番最初の方法で、１回目はうまく行くのに２回目がうまくいきません。これは１文字だけ読みたいのに、y△と入力しないと受け付けてくれないので、２回目は１回目に入れた改行コードを読んでしまうからです。

main()の中でreturnは普通用いません。main()の中でreturnを実行するとプログラムの実行を終了することになるからです。

while (条件) の時の条件は繰り返すための条件で、この条件が成立したら繰り返すのをやめるのではありません。

5.33　ファイルの読み書き(2)

ファイルへの出力に関しては、プログラムが出力したいだけデータを出力するで済みますが、入力の場合で特にデータの量が既知でない場合は、まだファイルにデータが残っているかどうか調べながら読むことになります。例えばfscanf()は正しく読めなかった場合-1を返すので、そうでなければファイルからデータが読めたことになります。

```
#include <stdio.h>

main(){
  FILE *fp;
  char line[80];

  fp=fopen("test.c","r");
  while (fscanf(fp,"%s",line)!=-1) puts(line);
  fclose(fp);
}
```

ファイルにデータをどんどん追加して行きたい場合には、fopen() の際に”w”のかわりに”a”を指定すると、以前出力した結果の後にこれから出力する結果を追加することができます。逆に言えば、”w”を指定した場合には以前ファイルに出力したデータは全て消去されます。

［演習問題］

1. dice.max と言うファイルを作成し、最初の行に適当な名前、次の行に 0 を入れておきます。read_score() と言う関数を作成し、このファイルから名前と点数を読み込むようにします。また write_score() と言う関数を作成し、このファイルに最高点を獲得した人の名前とその得点を書き込むようにします。この 2 つの関数の定義を dice4.c に追加し、main() の最初と終わりにこれらを呼ぶようにします。

2. 最高点を記録した歴代の名前とその得点をファイルに保存するように、改良します。ファイルには点と名前が交互に並んだ形にします。read_score() を修正して、ファイルにある得点と名前を読みながら画面に表示します。そして最後の点（歴代の中の最高点）のみ記憶します。write_score() も修正して名前と点をファイルに追加する様にします。そして、write_score() は最高点が出た後で実行する様に main() を直します。

5.34　リダイレクト とパイプ

MS–DOS の機能には、コマンド の入力や出力をファイルから行ったり、ファイルへ行ったりするものがあります。通常のコマンド はキーボード から入力し、画面に出力します。例えば、

```
A>dir△
```

とすれば、画面にディレクトリーの内容が表示されます。この時次のようにすると、画面に表示される代わりに DATA と言うファイルにディレクトリーの内容が入ります。

```
A>dir >DATA△
```

実際に入ったかどうかは、

```
A>type DATA△
```

とやれば確認ができます。要するに>の後にファイル名を書けば画面に出る内容が指定したファイルに入るのです。逆向きの<を使うとキーボード の代わりにファイルからデータを与える事ができます。

```
A>sort <DATA△
```

sort はデータをアルファベット順に並び換えるコマンド です。以上のような入出力を切り替えることを「リダイレクト」と呼びます。さらにあるコマンド の出力を別のコマンド の入力とすることができます。

```
A>dir | sort△
```

上記のようにすると、dir の出力を sort の入力とすることができて先程と同じ結果が得られます。これを「パイプ」と呼んでいます。

［演習］

上記の説明通りに実際に入力し、その結果を確認せよ。

5.35　ファイルの読み書き (3)

プログラムによっては、読み込みは１つのファイルからだけ、書き込みも１つのファイルだけで特にキーボードからの入力や画面への出力を必要としない物もあります。そのような場合には、キーボードから読み込んで画面に出力するプログラムとして作り、実行する段階ではリダイレクトでファイルを指定して使う方法があります。そうするとfopen()やfclose()を使わなくても済みます。たとえば、ファイルから先頭の５行だけ取り出すプログラムは次のようになります。

```
#include <stdio.h>

main(){
  char line[300];
  int i;

  for (i=0;(i<5) && (gets(line)!=NULL);i++) puts(line);
}
```

gets()はファイルから読み込めなかった時にはNULLと言う特殊な値を返す事になっています。ですからforの中の最初の条件は５回繰り返すこと、２つ目の条件がファイルが読める間を意味しています。両方の条件が成立する間、ファイルから１行読み込んでは表示する事を繰り返しますので、内容が５行未満のファイルでも正しく処理する事ができます。このプログラムをコンパイルした結果がhead.exeだとすると、

```
A>head <dice3.c△
```

とするとdice3.cの先頭５行が画面に表示されます。

［演習問題］

1. ファイルを修正すると自動的に.BAKと言う拡張子の付いたファイルが作られます。また、.OBJと言う拡張子のファイルもひとまず不要になったので.BAKのものと合わせて消去しましょう。

   ```
   A>del *.bak△
   A>del *.obj△
   ```

2. ファイルの行数を数えるプログラムを作れ。(wc.c)

   ```
   A>wc <head.c△
    There are 9 lines.
   ```

3. ファイルの終わりの５行を取り出すプログラムを作れ。(tail.c)

   ```
   A>tail <head.c△
      char line[300];
      int i;

      for (i=0;(i<5) && (gets(line)!=NULL);i++) puts(line);
   }
   ```

注意事項：２番の答えは、forでもwhileでもdo whileでも可能です。forを使うのが一番Ｃらしいプログラムでしょう。通常のプログラミングの発想で書くとwhileになります。行数がゼロのファイルにも対応するとなるとdo whileは適切ではないようです。

5.36 ポインター(1)

C言語を理解する上で一番解りにくいのがポインターと呼ばれるものと言われますが、これを理解し、使用しないと書けないプログラムが沢山あります。

これまでのプログラムでは必ず変数を利用してきました。変数は数値や文字を記憶することができますが、その実態はコンピュータ内部のメモリーの特定の領域です。プログラムにint i;と書くとメモリーに2バイトの領域が確保されて以後iと言う名前でそこに値を入れたり、参照したりできるようになります。一方メモリーの全ての領域には通常バイト単位でアドレスが決まっています。コンピュータはメモリーの値を操作するときにはこのアドレスを用います。ですからi=10;はCコンパイラーが変数i用に確保した領域のアドレスに10を入れるような命令に変換されます。

scanf()の中で用いられる&はこのアドレスを得るための演算子です。そしてこのアドレスを入れるための変数をポインター変数と呼びます。コンピュータにとっては整数型の変数のアドレスも文字型の変数のアドレスも同じ事ですが、ポインター変数はそれぞれ対応する別の型のものになります。そして&と逆の働きをするのが*です。この記号は乗算の印にも使われていますが、アドレスから元の変数に戻す時にも使われます。

ポインター変数の宣言はint *p;のようになります。しかし、実際使えるポインター変数の名前はpです。そして次のような事ができます。

```
int i,*p;

p=&i;          /*  ポインター変数pに変数iのアドレスを入れる。   */
i=10; *p=10;   /*  この2つは同じ結果をもたらす。               */
```

5.37 ポインター(2)

ポインターをよく用いるのは、文字列の処理と関数から変数に結果を渡す場合です。後者はscanf()でキーボードから得た値を指定された変数に入れる際に使われています。前者の例として、strcpy()は次のように定義できます。

```
void strcpy(char *p, char *q) {
   while ((*p++=*q++)!='\0');
}
```

これをstrcpy(str1,str2);のように呼び出すと、ポインターpはstr1[0]を指すようになり、qはstr2[0]を指します。whileに入って1回目では*pはstr1[0]になります。*qはstr2[0]に相当するのでstr1[0]にはstr2[0]の内容が入ります。pとqにそれぞれ++が付いているので、次はそれぞれstr1[1]とstr2[1]を指すようになります。そして文字列の最後にある'\0'を代入するまで繰り返されます。これまでの文字変数の配列を使うやり方ではこのように記述することはできません。

また、strstr()のようにポインターで結果を返す関数があります。これは文字列の中に指定した文字列があるかどうか調べる関数ですが、strstr("abcdefg","cde")とすると"cde"が"abcdefg"の3文字目以降に含まれているので"abcdefg"の3文字目のアドレスを返します。もし見つからなければ、NULLを返します。

［演習問題］

ファイルの行数および文字数（空白を含む）を数えるプログラムを作れ。ただし、string.hを利用せずにポインターを用いること。(wc2.c)

```
A>wc2 <head.c△
 There are 107 charcters in 9 lines.
```

注意事項：いい方法が思いつかない人はまず既存のstrlen()を使用してプログラムを作りましょう。それが上手く動いたら、自分でstrlen()を定義すれば完成です。さらにこの定義をmain()に埋め込むとより短いプログラムになります。

5.38 ポインター(3)

関数から値を１つ返すにはreturn()を利用します。もし複数の値を返したいときにはポインターを利用します。たとえば与えられた２つの値の四則の値を返す関数shisoku()は次のように定義できます。

```
#include <stdio.h>

void shisoku(int x, int y, int *add, int *sub, int *mul, int *div) {
   *add=x+y; *sub=x-y; *mul=x*y; *div=x/y;
}

main(){
   int i,j,k,l,m,n;

   printf("X = ");scanf("%d",&i);
   printf("Y = ");scanf("%d",&j);
   shisoku(i,j,&k,&l,&m,&n);
   printf("X+Y = %d, X-Y = %d, X*Y = %d, X/Y = %d\n",k,l,m,n);
}
```

shisoku()の最初の２つの変数はこれまでやったものと同じです。main()で指定した値が変数xとyに渡されます。その後の４つの変数は整数変数を指すポインターとして宣言されているので、main()で指定する値も&の付いた変数のアドレスを渡さなければなりません。普通の呼び出し方は、呼び出した側が関数にこの値について計算して下さいとお願するものですが、このポインターを使用した呼び出し方は、呼び出した側が関数にこの変数の場所に値を入れて下さいとお願するものになります。

厳密に言えば変数のアドレスでもらった関数側は、この例のように式の左辺で用いる他に右辺でも用いる事が可能なので、単に値を参照する事もできます。

［演習問題］

1994年の指定した月の日数と１日の曜日を数値で返す関数calmonth()を定義せよ。なお曜日と数値の対応は日曜日が０、月曜日が１、...、土曜日が６になるようにせよ。この関数を使用する簡単なmain()を追加して動くプログラムを書け。(cal1.c)

```
A>cal1△
Month = 1△
Day = 31, Start from = 6.
```

5.39 いかにプログラムを作っていくのか？

与えられた課題のプログラムを考えて行くにはどうすれば良いのか？と言うのは難しい問題です。例えばこの章末の問題はどうすれば良いでしょうか？まず、タイプの練習をする部分とtype.txtから例文を読み込む部分の２つには分けられそうです。C言語ではこのような場合、それぞれを別の関数として定義して使うのが普通ですので、それぞれtype_test()とread_text()と言う名前にしましょう。読み込むテキストのファイル名を関数に与える事にします。また２つの関数の間で共通な変数として少なくとも例文を入れておく変数が必要です。以上をプログラムの形にすると次の様なものになります。

```
#include <stdio.h>

char text[5][80];    /*  とりあえず５行分  */

main(){
   read_text("type.txt");   /*  例文を読み込む  */
   type_test();             /*  例文でテストをする  */
}
```

次にこの関数の定義を考えていきます。read_text()は何も値を返しません。そのかわりファイルの名前をmain()から受け取ります。するとこの関数の定義の先頭は、

```
void read_text(char *filename){
```

となり、同様にしてもう１つの関数は、

```
void type_test(){
```

になります。ファイルから何かデータを読む例はなかったか？これは5.32に例がありましたね。例ではデータを一つだけ読み込んでいますが、ここではtext[]に５行分読み込む事になります。（ファイルに５行分データがなかったらどうする？）

さて問題はtype_test()の方になります。まだかなり複雑そうな動作なので探しても似たような例は見つからないでしょう。そうすると自分で考えないといけません。例文が５行ありますから、５回繰り返す事になります。するとforを使うのでしょう。それから全文字数と誤って押したキーの数を数えるための変数が必要です。その初期化と最後の結果の計算も考えると、

```
void type_test(){
   int i, miss, all;

   miss=all=0;               /*  やる前はミスも文字数もゼロ。  */
   for (i=0;i<5;i++){
      .....                  /*  ここを考えなくてはならない。  */
   }
   printf("Your Score is %d %%\n",miss*100/all);  /*  miss/all*100ではうまく行かない  */
}
```

文字数を数えるのは、5.37の演習問題にありました。この時はなんか制限がありましたが、今回は何もないので一番楽そうな手を使います。そしていよいよ課題の核心の部分となります。１行表示してそれと同じ物をタイプしてもらう所です。これも明らかに表示する部分とタイプしてもらう部分に分けられます。でも表示はprintf()一発ですからわざわざ関数としてわける事もないでしょう。ではタイプしてもらう所をtype_line()とでもしましょう。今何行目を表示してタイプしてもらうつもりかを知らせてやる必要がありますし、逆にタイプした際の誤りの数を教えてもらわないといけません。するとこの関数の定義の先頭は、

```
int type_in(int i){
```

と言った形になるでしょう。さて通常の文字列の入力ではリターンキーを押すまで自由に変更ができます。ですから通常の入力方式ではタイプのテストになりませんので、5.26でやったgetch()を使って１文字ずつ取り込む事にします。タイプされた文字が例文と同じ文字ならば当たりですから次の文字と比較するようにします。外れならば正しい字を打ってくれるまで待たなければなりません。何回間違えるかなんてやってみなければわかりませんから、forでないループになります。キー入力があってから判定をしますから、do { } while ( );が使えます。

だいたいこれでできてしまいますが、実際これでやってみるとgetch()は押したキーの文字を画面に出してくれないので、やっている人はどこまで自分が入力したのか判らなくなります。そこで正しく入力できた文字は、画面に出してやるようにします。

［演習問題］

英語のテキストを１行ずつ表示して、それと同じ物をキー入力してもらい、その際にどのくらい正しくキーを押せたかどうか（つまり全文字数に対する誤って押したキーの数の割合）を調べるプログラムを作れ。ただし表示するテキストはtype.txtと言う名前のファイルに入っているものとするが、ファイルの内容は適当に英語の講義のテキスト等を参考に各自入力すること。(ttest.c)

注意事項：read_text()は簡単です。filenameの与え方もちょっとやってみれば判ると思います。１行丸ごとファイルから読み込むのにはfgets()関数を用います。これは、

```
fgets(文字列変数，読み込む文字数，ファイルポインター)
```

のような形で使います。ファイルから読もうとした行が２つ目に指定した文字数よりも大きい場合は、指定した文字数だけ読み込まれます。またgets()と異なり読み込んだ行の最後にある'\n'も文字列変数に入ります。ファイルから全く読み込めなかった場合（ファイルの終わりに達した場合）にNULLを返すのはgets()と同じです。

type_test()で考えるのはmissやallを数える部分だけです。

type_line()の部分は、１文字ずつgetch()で取ってきて比較します。文字の比較ですから文字列の比較と混同しないように。不用意にgetch()を呼ぶとその度にキーを押すことになりますので、getch()が複数出てきた人は注意して下さい。

5.40　実行時のパラメタの取得

C言語で書いたプログラムをコンパイルすると、その結果はMS–DOSのコマンドになって、その名前を入力するだけで実行することができます。通常のコマンドはパラメタを幾つか必要とします。例えば、ファイルを消去するコマンド delは消去するファイルの名前をパラメタとして与えなければなりません。

```
A>del aaa△
```

上の例では、delがコマンド名でaaaがパラメタになります。C言語でこのようなパラメタを使用したプログラムを書くにはこれまでいつもmain()だった所をmain(int argc, char *argv[])のように直します。argcやargvは慣習上これを使うだけで任意の名前にしてもかまいません。このような形のmain()にすると、argcにパラメタの数＋１が入り、argv[1]に１つ目のパラメタ、argv[2]に２つ目のパラメタ、と言った感じでパラメタが文字列になって入ります。

```
#include <stdio.h>

main(int argc, char *argv[]){
   int i;

   for (i=1;i<argc;i++) printf("パラメタ = %s\n",argv[i]);
}
```

これをコンパイルしてtestと言うコマンドにして次のように実行すると、

```
A>test a b c△
パラメタ = a
パラメタ = b
パラメタ = c
```

となります。

［演習］

argv[0] には何が入るかを、上記のプログラムちょっと直した物を実行して確認して下さい。

［演習問題］

前章の演習問題のプログラムを少し直します。パラメタとして問題の入ったファイル名を指定した場合はそれを使用し、パラメタが指定されなければ、type.txt を使用するようにせよ。(ntype.c)

5.41　データ構造

プログラムを作成する際には、どのようなやり方でやるかと言うアルゴリズムも大切ですが、データをどのように記憶するかも重要です。これまでの例では３目並べのゲームの状態を配列で記憶するぐらいで特に複雑なものはなかったと思います。ここでは少し複雑な例を考えてみます。

簡単なCAIでは、問題文を提示した後にその答えを選択させると言うものがあります。雑誌などに性格診断とか言ってスタートの矢印から順番にyes/noでたどっていくと、最後に診断が書いてあるものがあります。ファミコンなどのゲームの中には運動神経で勝負がつくものの他に、物語の中の世界に入っていって様々な探検をするようなものがあります。これは各場面で幾つかの可能な行動を選択することによって話しが進んで行きます。このようなものの共通部分を取り出して、データさえ交換すればCAIにもゲームにもなるものを考えてみましょう。

データを簡単に交換できるようにするのならば、プログラムの中には書き込めません。ファイルから読み込むのが良いでしょう。データを作成するプログラムを別に作成するのも大変ですから、REDなんかで作成や修正のできるテキストファイルの形式が良いでしょう。

さてそのデータの中身ですが、CAIを例に取れば、問題文がなければ困ります。本当は図などもできれば良いのですが文字だけにします。複数行に渡る物を許すとプログラムでは配列変数で各問題文ごとにその行数を管理しなければなりません。またファイルの中身を解釈するときにも面倒なので、問題文は１行にしましょう。

ただし文中に/があれば、プログラムの方で画面に/を出す代わりにそこで改行する様にすれば、複数行の問題文も可能になります。

次に選択子の内容とそれを選択したときの行き先があります。ここでは手抜きのために、選択子の内容も問題文のデータの中に入れてしまう事にします。

利用者が問題と選択子の内容を見て選択します。選択子も色々考えられますが、１、２、３のように必ず１から始まる連続した数と決めてしまえば、扱いも簡単でしょう。

そして選択によって次の問題が決まるのですが、これは問題ごとに番号を付けておいて、１を選んだら問題の３番、２を選んだら問題の４番のような対応関係をファイルに書いておく事にします。

最後はやはり終わってくれないと困ります。問題の０番と言うのを空けておいて、次は問題０番と言う指定があったらそこで終わるようにします。以上に従って実際に作ってみたデータの例を以下に示します。

```
1
まじめな問題です。フランス語ではコンピュータを示す名詞の性は何でしょうか？//
１．男性である。/２．女性である。/３．中性である。/４．最近の物には性が無い。
4 2 3 4 5
2
正解です。通常男性不定冠詞をつけてun ordinatureと呼びます。/残念ながらもう時
間がありませんので、これでおしまいです。/１を押して下さい。
1 0
3
残念ながらコンピューターは女性名詞ではありません。/１．を押して下さい。
1 1
```

```
4
フランス語には中性名詞はありません。/１．を押して下さい。
1 1
5
フランスにはちゃんと名詞の性を決定する由緒正しい機関があって、全ての名詞に性
別を割り振っています。/１．を押して下さい。
1 1
```

最初の数字が問題番号です。次の問題文は印刷の都合で２行になっているものもありますが、実際は１行です。次に並んでいる数字の一番最初の数字は選択子の数です。その後に並んでいるのは、選択した番号に対応する次の問題の番号です。つまり問題１番の４２３４５は、4が選択子が１から４まであることを示し、１を選択したら問題２へ、２を選択したら問題３へ、３を選択したら問題４へ、４を選択したら問題５へ行くことを示しています。

［演習問題］

上記の説明のプログラムを作成せよ。なお実行データの入ったファイルは、パラメタとして与えるものとする。(play.c)

```
A>play maze.dt△

A>play cai.dt△
```

まずは、ファイルから読み込む関数と読み込んだデータを使用して実行する関数に分かれるでしょう。２つの関数でデータを共有するので大域変数を使いましょう。問題数や選択子の最大数がわからないと変数の宣言に困りますが、100と10位で足りるでしょう。個々の問題文の長さも判りませんが、400位あれば５行入りますので十分でしょう。

割と簡単なプログラムですが、データを変更するだけでお勉強のプログラムにもゲームのプログラムにもなります。簡単な動きのプログラムは別として、少し複雑なプログラムは大抵別にデータファイルを持って、その内容で動作を修正できる形になっています。そのときにプログラムから見ると扱い易い形が望ましいし、利用者から見れば、自由度の大きい、書き易いものが望ましい事になります。

またプログラム内部へデータを読み込む場合に、どのような形で記憶するかによって処理速度に大きな差が生じる事があります。このあたりの詳しい話は、「プログラミング論」などで扱われるでしょう。

A　演習問題の解答

A.1　hello.c

次が一番普通の答えでしょう。

```
#include <stdio.h>

main(){
   printf("*********\n");
   printf("* HELLO *\n");
   printf("*********\n");
}
```

実はこれでも同じ結果が得られます。

```
#include <stdio.h>

main(){
  printf("*********\n* HELLO *\n*********\n");
}
```

A.2　tanuki.c

根性のある方は次の答えでも構いませんが、考えるだけでも疲れませんか？

```
#include <stdio.h>

main(){
   printf("  O O      O O      O O      O O      O O     \n");
   printf(" (o.o)    (o.o)    (o.o)    (o.o)    (o.o)    \n");
   printf("=( x )=  =( x )=  =( x )=  =( x )=  =( x )=   \n");
   printf("  U U      U U      U U      U U      U U     \n");
   printf("  O O      O O      O O      O O      O O     \n");
   printf(" (o.o)    (o.o)    (o.o)    (o.o)    (o.o)    \n");
   printf("=( x )=  =( x )=  =( x )=  =( x )=  =( x )=   \n");
   printf("  U U      U U      U U      U U      U U     \n");
   printf("  O O      O O      O O      O O      O O     \n");
   printf(" (o.o)    (o.o)    (o.o)    (o.o)    (o.o)    \n");
   printf("=( x )=  =( x )=  =( x )=  =( x )=  =( x )=   \n");
   printf("  U U      U U      U U      U U      U U     \n");
   printf("  O O      O O      O O      O O      O O     \n");
   printf(" (o.o)    (o.o)    (o.o)    (o.o)    (o.o)    \n");
   printf("=( x )=  =( x )=  =( x )=  =( x )=  =( x )=   \n");
   printf("  U U      U U      U U      U U      U U     \n");
   printf("  O O      O O      O O      O O      O O     \n");
   printf(" (o.o)    (o.o)    (o.o)    (o.o)    (o.o)    \n");
   printf("=( x )=  =( x )=  =( x )=  =( x )=  =( x )=   \n");
   printf("  U U      U U      U U      U U      U U     \n");
}
```

通常は次のようにします。

```
#include <stdio.h>

main(){
     int i;

     for (i=0;i<5,i++) {
```

```
        printf("  O O      O O      O O      O O      O O    \n");
        printf(" (o.o)    (o.o)    (o.o)    (o.o)    (o.o)   \n");
        printf("=( x )=  =( x )=  =( x )=  =( x )=  =( x )=  \n");
        printf("  U U      U U      U U      U U      U U    \n");
    }
}
```

さらに横方向にも繰り返しがあるので、forを重ねるとこの場合はかえって長いプログラムになります。

```
#include <stdio.h>

main(){
  int i,j;
  for (i=0;i<5;i++) {
      for (j=0;j<5;j++) printf("  O O    ");
      printf("\n");
      for (j=0;j<5;j++) printf(" (o.o)   ");
      printf("\n");
      for (j=0;j<5;j++) printf("=( X )=  ");
      printf("\n");
      for (j=0;j<5;j++) printf("  U U    ");
      printf("\n");
  }
}
```

A.3 kuku.c

printf()の中で"%4d"の４がないと、表示する数値によって桁数が変わるので四角い表になりません。

```
#include <stdio.h>

main(){
   int i,j;

   printf("\n  ***  The Multiplication Table  ***\n\n");
   for (i=1;i<10;i++) {
      for (j=1;j<10;j++) printf("%4d",i*j);
      printf("\n");
   }
}
```

A.4 2kou.c

本当は高学歴も入れたかったのですが、学歴を数値で入れるにはどうしたら良いと思いますか？昔ジャンケンのプログラムで０、２、５と入力するものがありました。

```
#include <stdio.h>

main(){
   int h,i;

   printf("Height=");scanf("%d",&h);
   printf("Income=");scanf("%d",&i);
   if (h>=170)
      if (i>=1000) printf("I love you.\n");
            else   printf("Bye bye, you must work hard.\n");
     else
      if (i>=1000) printf("Bye bye, you are too small.\n");
            else   printf("Bye bye.\n");
}
```

A.5 max3.c

３つの数の中の最大値ですから、まず２つを比較してその大きいほうと残りの数値を比較するやり方をとると次のようなプログラムになります。

```
#include <stdio.h>

main(){
   int a,b,c;

   printf("A=");scanf("%d",&a);
   printf("B=");scanf("%d",&b);
   printf("C=");scanf("%d",&c);
   if (a>b)
      if (c>a) printf("Max=%d\n",c);
         else  printf("Max=%d\n",a);
    else
      if (c>b) printf("Max=%d\n",c);
         else  printf("Max=%d\n",b);
}
```

一方、変数aに入っている値を最大値として、他の値と比較してもし負けるようだったらその値を新しい最大値として変数aに入れる方法もあります。この方が短いプログラムになります。

```
#include <stdio.h>

main(){
   int a,b,c;

   printf("A=");scanf("%d",&a);
   printf("B=");scanf("%d",&b);
   printf("C=");scanf("%d",&c);
   if (b>a) a=b;
   if (c>a) a=c;
   printf("Max=%d\n",a);
}
```

３つの数の最大値を求める場合、入力される値としては3つとも異なる、２つが等しい、3つとも等しいの大きく分けて３通りがあります。これらの全てに対して正しく結果が得られるものでないといけません。

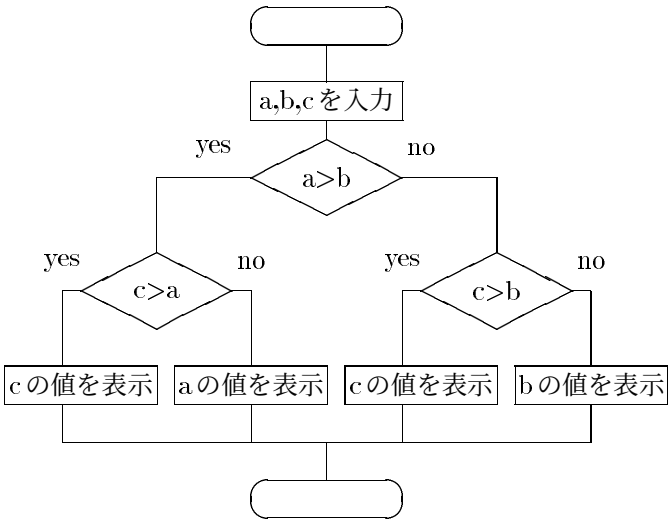
A.6 3kaku.c

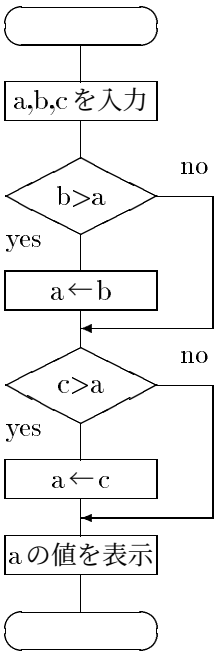
i行目にはi個のxを出す。そのiを１からSizeまで順番に変えていくと言った感じのプログラムです。

```
#include <stdio.h>

main(){
   int i,j,Size;

   printf("Size=");scanf("%d",&Size);
   for (i=1;i<=Size;i++) {
      for (j=1;j<=i;j++) printf("x")
      printf("\n");
   }
}
```

A.7 max3.c のフローチャート

最初のプログラムのフローチャートが左側のものです。





A.8 kaijo.c

0 に何を掛けても 0 なんて小学生でも知っている所に引っ掛かっていませんでしたか？

```
#include <stdio.h>

main(){
   int i,n,s;

   printf("N=");scanf("%d",&n);
   s=0;
   for (i=1;i<=n;i++) s=s*i;
   printf("N!=%d\n",s);
}
```

A.9 kisuwa.c

この問題のポイントは 2 つありました。1 つは for で 3 つ目に指定するところは必ずしも i++でなくとも良いこと。2 つ目は表示の最後が違うので別の方法で書き分ける必要があることでした。

```
#include <stdio.h>

main(){
   int i,n,s;

   printf("N=");scanf("%d",&n);
   s=0;
   for (i=1;i<n;i=i+2) {
      s=s+i;
      printf("%d+",i);
   }
```

```
   printf("%d=%d\n",n,n+s);
}
```

別解

```
#include <stdio.h>

main(){
   int i,n,s;

   printf("N=");scanf("%d",&n);
   s=0;
   for (i=1;i<=n;i=i+2){
      s=s+i;
      if (i==n) printf("%d=%d\n",i,s);
      else      printf("%d+",i);
   }
}
```

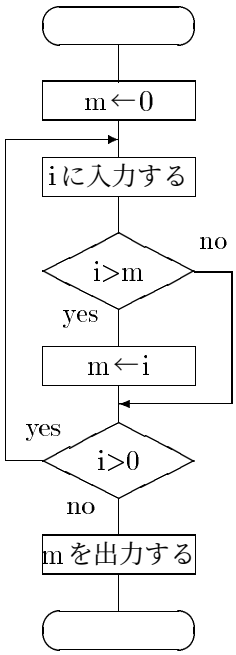
A.10 gcd.c

```
#include <stdio.h>

main(){
   int x,y;

   printf("X=");scanf("%d",&x);
   printf("Y=");scanf("%d",&y);
   while (x!=y)
      if (x>y) x=x-y;
      else     y=y-x;
   printf("G.C.D.=%d\n",x);
}
```

A.11 test231.c

```
#include <stdio.h>

main(){
   int i,x,m,n;
   for (i=1;i<=300;i++){
      n=0;m=i;x=i;
      while (x>1) {
         n++;
         if ((x%2)==0) x=x/2;
            else       x=x*3+1;
         if (x>m) m=x;
      }
      printf("x=%3d  max=%d step=%d\n",i,m,n);
   }
}
```

A.12 ループ (2) のフローチャート



A.13 heikin.c

データの個数を数える際に、最後の 1 個は単なるデータの終わりの印なので除かないといけません。次の例のように割り算をする際に 1 を引く方法の他に予め n に-1 を入れておく方法があります。

```
#include <stdio.h>
main(){
   int d,n,s;
   n=0;s=0;
   do {
      printf("Data=");scanf("%d",&d);
      s=s+d;
      n++;
   } while (d>0);
   printf("Mean=%d\n",s/(n-1));
}
```

A.14 s1000.c

ここでは素数かどうか調べたい数より 1 少ない数まで割って調べていますが、そんなに大きな数までで割って調べる必要はありません。普通は$\sqrt{n}$ までで十分で、もし n=10000 ならば 100 倍ぐ らい速くなります。

```
#include <stdio.h>

main(){
   int i,s;

   for (s=2;s<=1000;s++) {
      for (i=2;i<s;i++)
         if ((s%i)==0) break;
      if (s==i) printf("%4d",s);
   }
}
```

A.15 reverse.c

data[i] に入力した時には data[i] を調べて終わりかどうか判断しなければなりません。この入力と調べるの間に i++ が入っていると見た目は同じですが、正しく動きません。

```
#include <stdio.h>

main(){
   int i,data[100];

   i=-1;
   do {
      i++;
      printf("data=");scanf("%d",&data[i]);
   } while (data[i]>0);
   for (i=i-1;i>=0;i--)
      printf(" %d",data[i]);
   printf("\n");
}
```

A.16 vowels.c

```
#include <stdio.h>

main(){
   int i,n;
   char word[30];

   n=0;
   printf("Word=");scanf("%s",word);
   for (i=0;word[i]!='\0';i++)
      if ((word[i]=='a') || (word[i]=='e') || (word[i]=='i') ||
          (word[i]=='o') || (word[i]=='u')) n++;
   if (n==1) printf("There is a vowel in '%s'.\n",word);
   else      printf("There are %d vowels in '%s'.\n",n,word);
}
```

A.17 shortest.c

```
#include <stdio.h>

main(){
   int i,l;
   char word[30];

   l=30;i=30;
   do {
      if (i<l) l=i;
      printf("Word=");scanf("%s",word);
      for (i=0;word[i]!='\0';i++);
   } while ((i>1) || (word[0]!='Z'));
   printf("The shortest word length is %d.\n",l);
}
```

A.18 getword.c

頭から 1 文字ずつ見て行って、普通の文字はそのまま出して、空白が来たら改行して、Word=を出しておくだけです。他には、前の空白の位置を憶えて置いて、一気に書く方法もあります。むしろこの方が普

通かもしれません。

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

main(){
   int i;
   char sentence[80];

   printf("Sentence=");gets(sentence);
   printf("Word = ");
   for (i=0;sentence[i]!='\0';i++)
      if (sentence[i]==' ') printf("\nWord = ");
      else                  printf("%c",sentence[i]);
   printf("\n");
}
```

A.19 search.c

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

main(){
   int i,j;
   char sentence[80],string[20];

   printf("Sentence = ");gets(sentence);
   printf("String = ");gets(string);
   for (i=0;sentence[i]!='\0';i++){
      for (j=0;string[j]!='\0';j++)
         if (sentence[i+j]!=string[j]) break;
      if (string[j]=='\0'){
         printf("yes\n");
         break;
      }
   }
}
```

A.20 ox1.c

１つの関数の中でのみ使う変数（同じ名前でも全く関係ない変数も同様）はローカル変数とし、他の関数でも使用する変数のみ大域変数にするのが普通です。よってこの場合ban[ ]だけが大域変数となります。

```
#include <stdio.h>

int ban[10];

initialize(){
   int i;

   printf("\n  ***  Tick-Tack-Toe Game  ***\n");
   printf("        ver.1 Man - Man\n\n");
   for (i=1;i<=9;i++) ban[i]=0;
}

input(int j){
   int k;

   do {
```

```
      printf("==> ");scanf("%d",&k);
   } while ((k<1) || (k>9) || (ban[k]!=0));
   ban[k]=j;
}

suuji(int j){
   switch(ban[j]) {
      case -1 : printf(" X "); break;
      case  0 : printf(" %d ",j); break;
      case  1 : printf(" O ");
   }
}

yoko(int i){
   printf("      |   |\n");
   printf("   ");suuji(i);printf("|");suuji(i+1);printf("|");suuji(i+2);printf("\n");
   printf("      |   |\n");
}

display(){
   yoko(1);
   printf("   ---+---+---\n");
   yoko(4);
   printf("   ---+---+---\n");
   yoko(7);
}
```

main() の定義は演習問題文にあったものと同じ。

A.21 ox2.c

```
#include <stdio.h>

int ban[10];

#include "ox1.c"

int a(int i, int j, int k) {  /* 関数aは３つの引き数i, j, kを取り整数値を返す */
   return (ban[i]+ban[j]+ban[k]);
}

int test(int f){
   if (a(1,2,3)==f || a(4,5,6)==f || a(7,8,9)==f || a(1,4,7)==f ||
       a(2,5,8)==f || a(3,6,9)==f || a(1,5,9)==f || a(3,5,7)==f)
      return (1);
    else
      return (0);
}

int owari(){
   if (test(3)==1) {
      printf("あなたの勝ち\n");
      return (1);
   }
   if (test(-3)==1) {
      printf("あなたの負け\n");
      return (1);
   }
   return (0);
}
```

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

   initialize();              /*  メッセージの表示とban[ ]の初期化  */
   display();                 /*  ban[ ]の表示   */

   j=1;
   for (i=0;i<9;i++) {
      input(j);               /*  空いているところに入力  */
      j=-j;
      display();              /*  ban[ ]の表示   */
      if (owari()==1) break;  /*  ゲーム終了ならば for から抜ける  */
   }
}
```

A.22 ox3.c

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

int ban[10];

#include "ox1.c"
#include "ox2.c"

void umeru(){
   int i;
   while (ban[i=(rand()%9)+1]!=0) ;
   ban[i]=-1;
}

int t(int i, int j, int k){
   switch (ban[i]+ban[j]+ban[k]) {
      case   2 : ;
      case  -2 : if (ban[i]==0) ban[i]=-1;
                 else if (ban[j]==0) ban[j]=-1;
                 else ban[k]=-1;
                 return (1);
      default  : return (0);
    }
}

int tomeru(){
   if (t(1,2,3) || t(4,5,6) || t(7,8,9) || t(1,4,7) ||
       t(2,5,8) || t(3,6,9) || t(1,5,9) || t(3,5,7))
      return(0);
    else
      return(1);
}
```

main() の定義はox2.c のものと同じ。

A.23 ox4.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define  MESSAGE  "      ver. 4 : Man - Machine\n\n"
```

```
int ban[10];

#include "ox1.c"
#include "ox2.c"

void umeru(){
   int i;

   if (ban[5]==0)
      i=5;
   else
      if (ban[1]*ban[3]*ban[7]*ban[9]==0) /*  どれかがゼロならばゼロ   */
         do {
            while (ban[i=(rand()%9)+1]!=0) ;
         } while ((i!=1) && (i!=3) && (i!=7) && (i!=9));
      else
         do {
            while (ban[i=(rand()%9)+1]!=0) ;
         } while ((i!=2) && (i!=4) && (i!=6) && (i!=8));
   ban[i]=-1;
}

... 中略...

main(){
   int i,j;
   char ans;

   initialize();              /*  メッセージの表示と ban[ ] の初期化  */

   printf("先行ですか (y/n) ");scanf("%c",&ans);
   if (ans=='y') {
      j=1;
      display();              /*  人が先行するときのみ盤を表示する。 */
   } else j=-1;

   for (i=0;i<9;i++) {
      if (j==1) input(j);
      else      if (tomeru()) umeru();
      j=-j;
      display();              /*  ban[ ] の表示   */
      if (owari()) break;     /*  ゲーム終了ならば for から抜ける  */
   }
}
```

ox1.c の修正部分

```
void initialize(){
   int i;

   printf("\n  ***  Tick-Tack-Toe Game  ***\n");
   printf(MESSAGE);           /*   この行のみ変更    */
   for (i=1;i<=9;i++) ban[i]=0;
   srand(clock());
}
```

A.24 dice1.c

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

```
#include <conio.h>

void disp(int i){
   switch (i) {
      case 1 : printf("      \n  ●  \n      \n");break;
      case 2 : printf("    ●\n      \n●    \n");break;
      case 3 : printf("    ●\n  ●  \n●    \n");break;
      case 4 : printf("●  ●\n      \n●  ●\n");break;
      case 5 : printf("●  ●\n  ●  \n●  ●\n");break;
      case 6 : printf("●  ●\n●  ●\n●  ●\n");
   }
}

main(){
   int i;
   printf("Hit any Key!\n\n");
   i=1;
   do {
       if (++i>6) i=1;      /*  iの値が6を越えたら1に戻す      */
   } while (kbhit()==0);    /*  キーが押されていない間繰り返す  */
   getch();                 /*  押された内容を読み捨てる        */
   disp(i);
}
```

A.25 dice2.c

```
#include "screen.c"
#include <conio.h>

void disp(int i, int y, int x){
   color(black+i);       /* 表示する目の数によって異なる色にする */
   locate(x,y);
   switch (i) {
      case 1:         printf("      \n");break;
      case 2: case 3: printf("    ●\n");break;
      default:        printf("●  ●\n");
   }
   locate(x,y+1);
   switch (i) {
      case 1: case 3: case 5:
                      printf("  ●  \n");break;
      case 6:         printf("●  ●\n");break;
      default:        printf("      \n");
   }
   locate(x,y+2);
   switch (i) {
      case 1:         printf("      \n");break;
      case 2: case 3: printf("●    \n");break;
      default:        printf("●  ●\n");
   }
}

main(){
   int i;

   cls();cursol(OFF);
   locate(20,10);printf("Hit any Key!\n\n");
   i=1;
   do {          /* 3つのさいころが同時に動いている状態  */
      if (++i>6) i=1;
```

```
      disp(i,15,10);disp(i,15,20);disp(i,15,30);
   } while (kbhit()==0);
   getch();
   do {          /*  2つのさいころが同時に動いている状態  */
      if (++i>6) i=1;
      disp(i,15,20);disp(i,15,30);
   } while (kbhit()==0);
   getch();
   do {          /*  1つだけさいころが動いている状態  */
      if (++i>6) i=1;
      disp(i,15,30);
   } while (kbhit()==0);
   getch();
   color(white);
   cursol(ON);
}
```

さいころがいくつ動いているかをmと言う変数で表すとすると、2重ループ1つでもできます。以下はmain()の定義だけで他の部分は上記のプログラムと同じです。

```
main(){
   int i,j,k,m;

   cls();cursol(OFF);
   locate(20,10);printf("Hit any Key!\n\n");
   i=j=k=1;
   for (m=0;m<3;) {
      if (kbhit()) {  /*  キーを押すとmが増える。→動くさいころの数が減る。  */
         m++;
         getch();
      }
      switch (m) {
         case 0 : if (++i>6) i=1;disp(i,15,10);  /* breakが無いことに注意 */
         case 1 : if (++j>6) j=1;disp(j,15,20);
         case 2 : if (++k>6) k=1;disp(k,15,30);
      }
   }
   color(white);
   cursol(ON);
}
```

A.26 dice3.c

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

void disp(int i, int y, int x){
   この関数の定義はdice2.cのものと同じなので省略。
}

int onegame(){                          /*  数値を返すので int が付く  */
   int i,j,k,m;

   cls();cursol(OFF);
   locate(20,10);printf("Hit any Key!");
   i=j=k=1;
   for (m=0;m<3;) {
     if (kbhit()) {
        m++;
```

```
      getch();
    }
    switch (m) {
      case 0 : if (++i>6) i=1;disp(i,15,10);
      case 1 : if (++j>6) j=1;disp(j,15,20);
      case 2 : if (++k>6) k=1;disp(k,15,30);
    }
  }
  color(white);
  cursol(ON);
  if ((i==j) && (j==k)) return (10);            /*  ここから３行が追加分  */
  if ((i==j) || (j==k) || (i==k)) return (3);
  return (0);
}

int game(){
  int i,s;

  s=0;
  for (i=0;i<5;i++) s=s+onegame();
  return (s);
}

main(){
  printf("総得点 = %d\n",game());
}
```

A.27 dice4.c (1)

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

int game();         /*  コンパイル時にエラーがでないようにする。  */

main(){
  int p,top;
  char name[16];

  top=-1;
  do {
    printf("\n\n総得点 = %d\n",p=game());    /*  得点をpに入れておく。  */
    if (p>top) {
      top=p;
      printf("Your Name = ");scanf("%s",name);
    }
    printf("Do you want play again? (y/n) : ");
  } while (getch()=='y');
  printf("\n\nHigh Score is %d by %s.\n",top,name);
}
```

A.28 dice4.c (2)

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

int game();         /*  コンパイル時にエラーがでないようにする。  */
int top;            /*  topとnameはread_score()でも使うので大域変数にする  */
char name[16];

void read_score(){  /*  何も返さないのでvoid  */
```

```
   FILE *fp;

   fp=fopen("dice.max","r");
   fscanf(fp,"%s",name);
   fscanf(fp,"%d",&top);
   fclose(fp);
}

void write_score(){  /*  何も返さないのでvoid  */
   FILE *fp;

   fp=fopen("dice.max","w");
   fprintf(fp,"%s\n%d\n",name,top);
   fclose(fp);
}

main(){
   int p;

   read_score();
   do {
      printf("\n\n総得点 = %d\n",p=game());    /*  得点をpに入れておく。  */
      if (p>top) {
         top=p;
         printf("Your Name = ");scanf("%s",name);
      }
      printf("Do you want play again? (y/n) : ");
   } while (getch()=='y');
   printf("\n\nHigh Score is %d by %s.\n",top,name);
   write_score();
}
```

●歴代の成績表示付き

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

int game();
int top;
char name[16];

void read_score(){
   FILE *fp;

   fp=fopen("dice.max","r");
   while (fscanf(fp,"%s",name)!=-1) {
      fscanf(fp,"%d",&top);   /*  名前があれば得点もあるので無条件に読んでも問題はない  */
      printf("%d by %s.\n",top,name);
   }
   fclose(fp);
   printf("\nHit Any Key!! ");/*  これと次の行が無いとすぐゲームが始まってしまう  */
   getch();
}

void write_score(){
   FILE *fp;

   fp=fopen("dice.max","a");    /*  ここだけ w が a に変っている  */
   fprintf(fp,"%s\n%d\n",name,top);
   fclose(fp);
}
```

```
main(){    /*  write_score() の位置だけ変更あり  */
   int p;

   read_score();
   do {
      printf("\n\n総得点 = %d\n",p=game());
      if (p>top) {
         top=p;
         printf("Your Name = ");scanf("%s",name);
         write_score();  /*  記録が更新された時のみファイルに書き込む  */
      }
      printf("Do you want play again? (y/n) : ");
   } while (getch()=='y');
   printf("\n\nHigh Score is %d by %s.\n",top,name);
}
```

A.29 wc.c

```
#include <stdio.h>

main(){
   char line[80];
   int i;

   for (i=0;gets(line)!=NULL;i++);
/* i=0; while (gets(line)!=NULL) i++;            while を使った場合     */
/* i=-1; do { i++; } while (gets(line)!=NULL);   do while を使った場合  */
   printf("There are %d lines.\n",i);
}
```

A.30 tail.c

次のプログラムは文字配列を利用して全てのデータを記憶する方式です。問題となるのは配列の大きさよりも多くの行数のデータを入力すると配列が溢れて場合によってはプログラムが暴走する点です。

```
#include <stdio.h>

main(){
   char lines[100][80];  /*  長さ80文字迄の文字列変数を100個  */
   int i,j;

   for (i=0;gets(lines[i])!=NULL;i++);   /*  ファイルをlines[]に全て読み込む  */
   for (j=i-5;j<i;j++) puts(lines[j]);   /*  終わりの５行を出力する           */
}
```

ではどうしたらよいか？最終的に必要なのは５行だけなので、その分だけ配列を用意します。最初の５行はそのまま順番に入れて行き、６行目があればもう最初の１行目のデータは不用なのでそこに、６行目の内容を入れます。７行目があれば２行目のデータも不用になります。

```
#include <stdio.h>

main(){
   char lines[6][80]; /* 最後に読んだ行が消えるので１つ余分に */
   int i,n;

   for (n=0;gets(lines[n++%6])!=NULL;) ;  /* %は余りを求める演算子 */
   for (i=0;i<5;i++) puts(lines[n++%6]);
}
```

A.31 wc2.c

まずはstring.hを使った違反版です。

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

main(){
   char line[80];
   int i,s;

   s=0;   /*  これを忘れても動くけど減点の対象  */
   for (i=0;gets(line)!=NULL;i++) s=s+strlen(line);
   printf("There are %d characters in %d lines.\n",s,i);
}
```

次に自分でstrlen()を定義すれば、一応完成です。

```
#include <stdio.h>

int strlen(char *p) {
   int i;
   for (i=0;*p++ != '\0';i++);
   return (i);
}

main(){   /*  こちらは変更なし  */
   char line[80];
   int i,s;

   s=0;
   for (i=0;gets(line)!=NULL;i++) s=s+strlen(line);
   printf("There are %d characters in %d lines.\n",s,i);
}
```

プログラム中でstrlen()は1回しか呼んでいませんし、strlen()の定義自体も短いのでmain()の中に埋め込むともう少し短いプログラムになります。

```
#include <stdio.h>

main(){
   char line[80],*p;
   int i,s;

   s=0;
   for (i=0;gets(line)!=NULL;i++)
      for (p=line;*p++ != '\0';s++);
   printf("There are %d characters in %d lines.\n",s,i);
}
```

A.32 cal1.c

```
#include <stdio.h>

void calmonth(int month, int *day, int *week){
   switch (month){
      case  1 : *day=31;*week=6;break;
      case  2 : *day=28;*week=2;break;
      case  3 : *day=31;*week=2;break;
      case  4 : *day=30;*week=5;break;
```

```
      case  5 : *day=31;*week=0;break;
      case  6 : *day=30;*week=3;break;
      case  7 : *day=31;*week=5;break;
      case  8 : *day=31;*week=1;break;
      case  9 : *day=30;*week=4;break;
      case 10 : *day=31;*week=6;break;
      case 11 : *day=30;*week=2;break;
      case 12 : *day=31;*week=4;          /* ここのbreak;は不要 */
   }
}

main(){
   int i,j,k;

   printf("Month = ");scanf("%d",&i);
   calmonth(i,&j,&k);
   printf("Day = %d, Start from = %d.\n",j,k);
}
```

曜日を計算で求めるのはそんなに難しくありません。calmonth()の定義のみ示します。

```
void calmonth(int month, int *day, int *week){
   int i,s;

   s=0;*day=0;   /*  sに毎月の日数を加えて総日数を求める。  */
   for (i=1;i<=month;i++) {
      s=s+*day;
      switch (i){
         case  1 : *day=31;break;
         case  2 : *day=28;break;
         case  3 : *day=31;break;
         case  4 : *day=30;break;
         case  5 : *day=31;break;
         case  6 : *day=30;break;
         case  7 : *day=31;break;
         case  8 : *day=31;break;
         case  9 : *day=30;break;
         case 10 : *day=31;break;
         case 11 : *day=30;break;
         case 12 : *day=31;
      }
   }
   *week=(s+6) % 7;
}
```

A.33 ttest.c

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

char text[5][80];

void read_text(char *filename){
   FILE *fp;
   int  i;

   fp=fopen(filename,"r");
   for (i=0;i<5;i++){
      fgets(text[i],80,fp);  /*  80は読み取る文字列の最大の長さ。 */
```

```
      text[i][strlen(text[i])-1]='\0'; /* fgets() が付ける文字列の最後の\n を除く。*/
   }
   fclose(fp);
}

int type_line(int i){
   int j, m;

   m=0;   /*  m で誤って押した数を数える。 */
   for (j=0;text[i][j]!='\0';j++){
      while (text[i][j]!=getch()) m++; /* getch() で読んだ文字が違ったら m を増やす。*/
      printf("%c",text[i][j]);
   }
   printf("\n\n");
   return (m);
}

void type_test(){
   int i, miss, all;

   miss=all=0;
   for (i=0;i<5;i++) {
      all=all+strlen(text[i]);
      printf("%s\n",text[i]);
      miss=miss+type_line(i);
   }
   printf("Your Score is %d %%\n",miss*100/all);
}

main(){
   read_text("type.txt");
   type_test();
}
```

A.34 ntype.c

ttype.c の main のみを直します。

```
main(int argc, char *argv[]){
   if (argc==2) read_text(argv[1]);
   else         read_text("type.txt");
   type_test();
}
```

A.35 play.c

```
#include <stdio.h>

char q[100][400];           /* ２つの関数で用いる変数は大域変数にする */
int  sn[100],to[100][10];   /* q は問題文、sn は選択子の数、to は行き先  */

void read_data(char *filename){
   FILE *fp;
   int  i,j;

   fp=fopen(filename,"r");
   while (fscanf(fp,"%d",&i)!=-1){
      fgets(q[i],300,fp);
      fscanf(fp,"%d",&sn[i]);
      for (j=1;j<=sn[i];j++) fscanf(fp,"%d",&to[i][j]);
```

```
   }
   fclose(fp);
}

int play_data(int i){
   char *p;
   int j;

   for (j=0;*q[j]!='\0';j++)      /* 問題を１文字ずつ表示する。*/
      if (q[j]=='/') printf("\n");  /* ／だったら改行する。       */
      else           printf("%c",q[j]);
   do {
      printf("  ==> ");scanf("%d",&j); /* 番号を入れてもらって。。。*/
   } while (j>sn[i]);                  /* 大き過ぎたらやり直し。     */
   return (to[i][j]);  /* 入れた番号をちゃんと行き先画面の番号に直して返します。 */
}

main(int argc, char *argv[]){
   int i;

   read_data(argv[1]);  /* １つ目のパラメタがファイル名 */
   i=1;
   while (i=play_data(i));
}
```

このプログラムだと、パラメタ無しで起動したり、ファイル名がまちがっていたりすると正しく動きません。