生活社会科学科

専門演習 A テキスト

三木　邦弘

平成14年4月10日

目 次

1　はじめに

平成14年度の私の専門演習Aでは、ワークステーションを使ってCを使用したプログラミングを学びます。これは従来は私の基礎演習の前期で行われていた内容に相当するものですが、昨年度基礎演習で多少はプログラミング風の事もしているし、人数はこれまでの基礎演習と比べても最小だし、3年生なので、もう少し高度な内容にしたいと思います。

プログラミングができるようにして目指そうとしているのは、WWWにおけるサーバーでの処理の実現です。ただ前期は普通のプログラミングのお勉強にして、サーバーでの処理は後期からの課題にします。

1.1　演習の受け方

妙な表題ですが、この演習を受ける皆さんへの要望です。皆さんとしては必修だから仕方がないと言う面も否定できないと思います。ただ他の講義や演習でもそうですが、単位のためだけにこれらを受講するのはもったいないです。この演習は大部分を提出された課題のできで評価しています。例年単位が取れなかったら困ると言うことで他の人の答えを丸写しして提出する人が少なくありません。しかしそれではこの演習で扱っているプログラミングは身につきません。いくら単位が取れてもそれでは意味がありません。そのあたりをよく考えてやってください。提出された課題の評価では、丸写しをした人はできるだけ低く、できが悪くても自力でがんばったと見られるものはできるだけ高くしようとします。これはなかなか難しいし間違えることもあります。間違った場合は指摘してください。

小人数ですが、さらにコミュニケーションの道具としてメーリングリストを今回初めて導入しようと思います。その活用が私にとって初めての専門演習の一つの課題であります。

1.2　プログラミングについて

プログラムを作成することをプログラミングと言います。プログラミングに関しては、「プログラミング演習」とどうしても内容が重複するなどの問題があります。従来はこちらをネットワークに関連したプログラミングと言うことで違いを出そうと考えました。今年度からは「プログラミング論」も担当するようになったので、「プログラミング論」、「プログラミング演習」は特定のプログラミング言語ではなく、様々な言語を扱うものにしようと考えています。よってこの専門演習とは別の視点から「プログラミング演習」ではプログラムを学べると思います。

Cを利用してプログラミングを行うのならば、本来ならば何かCの解説の本を買って頂いてそれを元に演習を進めたいのですが、なかなか適当な本がありません。この演習ではCを使用しますが、文法的な事項は必要最小限とします。そのため通常の本では一通りC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。と言うわけでせっせとテキストを作成したのですが、もっと勉強されたい方は、C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。

1.3　OSについて

この演習ではWindows98と言うOSとUnixと言うOSの両方を利用します。ある意味では両極端のOSを使うことになります。Windows98はマウスなどを使用して初心者にも容易に使えるようなOSです。Unixは何か指示をするならば、必ずキーボードからコマンドを入力しなければなりませんから、馴れない人にはなかなか使いこなせないものです。1995年に登場したWindows95のおかげで、それまで以上に多くの人がパソコンを利用するようになりました。しかし世の中の仕事は必ずしもWindowsでやる方が便利とは限りません。形の定まった仕事の繰り返しを指示するには、マウスで行うよりコマンドをキーボードで入れる形の方がはるかに容易な場合も少なくないのです。

また大きな違いとして一人で使うものか、大勢で使うものかの違いがあります。パソコンは通常一人で利用するので、特に利用者の識別と言う機能がありません。ところが複数で利用することを前提にしたUnixでは、異なる利用者がお互いに干渉しないように設定ができるようになっており、その設定を生かすために利用者の識別を行います。そのために登録名やパスワードが必要になって面倒なのですが、現在のパソコンはLANなどのネットワークに接続されて利用されることが普通になり、するとLANに接続された複数のパソコンを一つのシステムと見なすような場合が増えています。そうなるとUnixのように利用者を識別して運用されるシステムがWindows主体でも避けられない状況です。

2 基本的なことがら

2.1 ワークステーションへの接続

今年のこの演習は生社棟の005室のパソコンを利用しますが、昨年度基礎演習を行った学園センターの学生情報処理実習室のパソコンでも、生社棟の002、004、414室などでもほぼ同様の手順でやれます。

使用する際にはまず、本体のスイッチを押して待つとWindowsが立ち上がってさらにメニューが出てきます。その中で「WSの端末」と言うところをクリックするとワークステーションに接続し利用を開始することができます。ただ生社の演習室では端末のボタンが複数あるので学園センターに関するものを必ず選択してください。

パソコンの利用終了の際には必ず、「利用終了」をクリックしてください。多少面倒でも必ずこの方法で終了するようにしてください。

1. login:のところでグループ名を入れます。間違えたら DEL キーで直せます。
2. name:のところで登録名を入れます。
3. password:のところでパスワードを入れます。

ところでこの後でメニューが表示されます。従来はこれを利用してメールなどを扱っていました。この演習ではUnixのコマンドにも慣れ親しんでいただくために、このメニューが出ないように設定を変更します。そのためにはまず、「設定変更」を選択し、その中でさらに「このメニューを手動起動にする」を選択してください。次回からメニューが出てこなくなります。

2.2 Unixのコマンド

メニューが出ないようにすると、ワークステーションに接続するとプロンプト(入力促進記号とも言われる。ここではcc07[miki]%のようなものである。)が表示されます。何かやってもらうにはここでコマンドを入力し、最後に必ず ↩ を押す必要があります。Windowsのようにメニューから選択すると言うのではないのでコマンドを憶えていないと何もできません。

ここではこの演習で使用する最小限のコマンドを紹介します。

- logout ↩ : ワークステーションの利用を終了するときにはこのコマンドを実行します。時々これをせずに端末ソフトを終了してしまう人が居ますがそのような事がないように。
- elm 相手のメールアドレス ↩ : メールを送る場合はこのようにします。elmとメールアドレスの間には必ず空白を入れます。すると見出しを聞いてくるので、適当に入力して ↩ を押すと、文章入力画面になります。そこでの操作方法は次章に説明があります。入力を終了すると、

```
次の中から選んでそのキーを押してください: s
                メールを送る：s、破棄：f、再編集：e、ヘッダーの変更：h
```

と表示されるので通常は[s]または[↩]を押します。この演習では大抵の課題はメールで送ることになります。その時の送り先のメールアドレスはmiki@ssになります。

- ls[↩]：現在あるファイルの名前を表示するコマンドです。ファイルの名前だけでなく大きさなども知りたいときには、ls -l[↩]のように-lを付けます。
- less ファイル名[↩]： 指定したファイルの内容を表示します。まずファイルの先頭が表示されますので、続きを見たい場合は、スペース、戻りたい場合は[b]、終了するときは[q]を押します。なお、内容が数行しかないようなファイルはcatコマンドを同様に利用するとファイルの内容全体が表示されます。
- cp 元ファイル名 先ファイル名[↩]： ファイルのコピーをします。

  ```
  [miwako]%cp abc efg
  ```

  ファイルabcの内容がファイルefgにコピーされます。
- mv 元ファイル名 先ファイル名[↩]：ファイルの移動をします。cpと異なり元のファイルは消えます。単に名前を変更するのにも使われます。

  ```
  [miwako]%mv abc efg
  ```

  ファイルabcがファイルefgになります。
- rm ファイル名[↩]：指定したファイルを消去します。

  ```
  [miwako]%rm abc
  ```

  ファイルabcが消去されます。
- man コマンド名[↩]：指定したコマンドの説明（マニュアル）を表示します。lessと同様に次の部分を見たいときにはスペースバー、前に戻りたい時には[b]を押します。見終わったら[q]を押します。

  ```
  [miwako]%man man
  ```

  manコマンド自身の説明が表示されます。
- typist[↩]：タイプの練習ソフトを起動することができます。
- letters[↩]：タイプのゲームソフトを起動することができます。
- tetoris[↩]：説明省略。Pで休憩、Qで終了です。もしはまっても知りません。

3　テキスト エディタ

現在のコンピュータは本来の数値情報の処理だけでなく、文字、音声、画像、動画等を扱うことができます。これらの情報はファイルと言う形で記憶されるのが普通です。ファイルとは名前の付いたデータの集まりとも言えます。必ず少なくとも一つの名前が付いています。我々はその名前をもとに自分の処理したいデータをコンピュータに指示することになります。

画像や音声のデータを扱うのは技術的に難しい面がかなりあります。ここではここでは数値や文字データの入力や訂正に用いられるテキストエディタの使用法について述べます。これはワープロと同じようなものですが、データやプログラムを入力する為のものなので、清書をするための機能はほとんど持っていません。

ワークステーションでよく使われているエディタは、vi系とemacs系に分けられます。カーソルを動かすためのキー操作等が大きく異なりますので、どちらかの系統で慣れると他の系統のエディタは使いにくくなります。また同じ系統のエディタは基本的な操作はほとんど変わりません。

vi系の代表のviはUnixに標準装備のエディタです。ですからUnixの動くマシンではどこでも利用可能です。ただ操作に関して、コマンド入力状態とテキスト入力状態と分かれているので、初心者にちょっと分かりにくい点があります。

emacs系はUnixの標準装備ではありませんが、多くのマシンにインストールされています。基本的に入力した文字はテキストとして挿入されるので、ワープロに慣れた人には違和感が少ないようです。ここではngと言うemacs系のテキストエディタについて説明します。

3.1　使い方の基礎の基礎

まず起動をするときは、プロンプトの出ているところで、「ng ファイル名[↩]」と入力します。ngの後は必ず空白が必要です。たとえば、

```
[miwako]%ng abc
```

でabcと言う名前のファイルの内容を編集することができます。

- 入力する文章は入力すればカーソルのある位置にどんどん挿入されます。入れ間違った場合には、[Back space]を押します。カーソルは矢印キーを押すことによって動かすことができます。
- 入力を終えた後は結果を保存しなければなりません。[CTRL]キーを押しながら、[x]、[s]の2つのキーを順番に押します。
- テキストエディタを終了するには、[CTRL]キーを押しながら、[x]、[c]と2つのキーを順番に押します。変更したのに保存をせずに終了しようとすると、保存しますかと聞いてくるので、[y]と押します。

3.2　使い方の基礎

以下では次のような形でキー操作を説明します。

C-x [CTRL]キーを押しながら、x キーを押す。

C-x y C-x を押した後、y キーを押す。

C-x C-y C-x を押した後、C-y を押す。

EC x [ESC]キーを押した後、x キーを押す。

ESC C-x ESC キーを押した後、C-x を押す。

まずカーソルの移動です。一応キーボードの矢印キーでもカーソルが移動できるように設定してありますが、手がホームポジションから離れてしまいますので、できるだけこちらでやるようにしてください。

C-f	カーソルの一文字前進	C-b	カーソルの一文字後退
C-n	カーソルの次行への移動	C-p	カーソルの前行への移動
C-v	次の画面への移動	ESC v	前の画面への移動

通常の文字を入力するとそのままカーソルの位置に挿入されます。カーソルの位置の文字を消去する時は、C-d を入力します。

ちょっと変わったコマンドとして1行の長さを揃えるものがあります。電子メールやニュースのテキストをエディタで入力すると、編集しているうちに行の長さはふぞろいになります。行の長さを揃えたい部分の前後を空行で分離し、その中にカーソルを移動して ESC q を入力すると、各行はほぼ 70 字の所に揃えられます。

3.3 知っていると便利な使い方

1. まずはその他のカーソル移動のコマンドです。

C-e	カーソルの行の終わりへの移動	C-a	カーソルの行の先頭への移動
ESC >	カーソルのファイルの最後への移動	ESC <	カーソルのファイルの先頭への移動

   この他に、C-g を入力すると移動したいのは何行目かを聞いてくるので、行数と ↩ を入力するとその行にカーソルが移動します。

2. カーソル以降の一行削除には、C-k を入力します。大量に削除する場合は、その先頭の位置で C-@ を入力し印 (mark) を付け、最後の次の文字のところで C-w を入力します。このときこの削除した内容はバッファと呼ばれる領域に保存されます。ここには 1 回の削除分しか入りませんが、C-y を入力するとその内容をカーソルの位置に挿入することができます。これを利用してテキストのファイル内での移動が実現できます。コピーをしたい場合には、C-w の代わりに ESC w を使用すれば元の部分が削除されません。

3. 検索をしたい時には、C-s を入力して続けて探したい文字列を入力します。このとき最後に ↩ を入力する必要はありません。入力したものと同じ文字列が C-s を入力した時のカーソルの位置以降にあればそこへカーソルが移動します。さらに一致する文字列の位置へ移動させたいときは、もう一度 C-s を入力します。検索を止めるときは、ESC を押します。C-r は探す方向が現在のカーソル位置より以前になる他は C-s と同じ動作になります。

4. テキスト中もある特定文字列を全て、または部分的に置き換えるときには、ESC % と元の文字列 ↩ と置き換える文字列 ↩ を入力します。すると該当する箇所にカーソルが移動しますので、

!	以下の候補を全て一気に置換する
スペースバー	置換実施後次の候補へ
BS	置換しないで次の候補へ
ESC	置換の終了

   のいずれかのキーを押します。

5. 編集中にちょっとngを中断して他のコマンドを実行したくなることがあります。そのような時には、C-zを入力するとngは一時停止してプロンプトが表示されてコマンドの入力が可能になります。逆に中断しているngに戻る時には、fg↩と入力します。中断しているのを忘れて終了しようとすると

   There are suspended jobs.

   と言われますので、このfgコマンドでngに戻ってngを終了して下さい。

6. 他のファイルの内容をカーソルの位置に取り込むには、C-x iとファイル名と↩を入力します。

演習問題

1. 専門演習のメーリングリストにまだ登録していない者は、下記のURLにアクセスして登録せよ。登録先のメーリングリスト名は「miki02」で、パスワードは「zxcvbnm」である。

   ```
   http://cc01.center.sugiyama-u.ac.jp/~mailing/
   ```

2. elmコマンドを利用してメーリングリストに自己紹介のメールを送れ。メーリングリストに送るときのメールアドレスは、

   ```
   miki02@mlserver.center.sugiyama-u.ac.jp
   ```

4 Cによるプログラミング

4.1 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。

4.2 Cの生立ち

CはUnixと言うオペレーティングシステム(OS: Operating System)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を拡張する形でCが誕生しました。それ以降Unixの大部分はC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。

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。一方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。C はまず後者のような分野で使われるようになりました。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で書かれる事が多くなりました。

現在コンピュータネットワークでよく用いられるワークステーションのOSはUnixです。Unixには標準でCが使用できるようになっているので、様々なプログラムがCで書かれています。

4.3 コンパイルの仕方

我々がCで記述したプログラムはそのままではコンピュータによる実行はできません。通常はコンパイルと言う変換が必要です。その結果できたファイルをコンピュータで実行することにより初めてプログラムが動くことになります。

我々がプログラミング言語で記述したプログラムの事を**ソースプログラム**と呼び、変換して実行できるようになったものを**オブジェクトプログラム**と呼びます。

ワークステーション上でCによるプログラムを作成するには、次のような手順となります。

1. ソースプログラムの入ったファイルを作成する。拡張子は.cにする。
2. gcc ファイル名↩でコンパイルする。何かメッセージが出たら、ソースプログラムの誤りがあるので、修正して再度コンパイルをする。
3. a.out↩でコンパイルしたプログラムを実行することができる。コンパイルの際に特に指定しなければオブジェクトプログラムは全てa.outと言う名前になるために注意すること。
4. 次のプログラムをコンパイルした時に今のオブジェクトプログラムが消えないようにするには、mv a.out 適当な名前↩で名前を変更しておく。以後ここで指定した適当な名前を入れるとプログラムを実行することができる。

実際の実行例を示すと、

1. ng test.c↩と入力して、以下のプログラムを入力してから保存、終了する。

```
#include <stdio.h>

main(){
   printf("This is my first program.\n");
}
```

2. gcc test.c ↩ と入力してコンパイルする。
3. a.out ↩ と入力して、以下の画面に以下のように出力されれば良い。

```
This is my first program.
```

4. mv a.out test ↩ と入力する。時々適当な名前として test.c 等を使う人が居るが、それをするとソースプログラムが消えてしまう。
5. test ↩ とすると先程と同じ結果が得られる。

5　Cの文法

日本語や英語のような普通の言葉に文法があるように、プログラミング言語にも文法があります。普通の言葉以上に厳密なもので、通常は文法的に間違ったものはプログラムとは言えません。また文法だけ知っていても英語が話せないように、実際のプログラムを作成するためには、用例などの経験や知識が必要です。

5.1　Cのプログラムの形

今の段階ではC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。関数については後で詳しく説明します。C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で言う文が;(セミコロン)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ではprintf()と言う関数を使って文字列や計算結果の出力をします。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。) コンパイルの仕方の例では、`printf("This is my first program.\n")`となっていたので、This is my first program.と表示された後で改行がなされています。

表示したい文字列に「” 」が含まれている場合は注意が必要です。

```
printf("He said, "I love you.");
```

とするとIの手前で文字列が終わったことになるので、その後にあるIやloveやyou.は何やねんと言う感じのエラーになります。このような場合は、

```
printf("He said, \"I love you.\".");
```

のように\を補います。

［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れ。(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。

```
*********
* 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では通常-32768から+32767の範囲の値が、ワークステーションでは±約21億までの値が記憶可能です。このように記憶できる値の範囲がコンピュータによって異なるので注意が必要です。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。またこの宣言は{のすぐ後になければなりません。

実際の変数の利用は次のような感じになります。

プログラム	意味	変数aに残る値
a=3;	代入 (変数aに3を入れる)	3
a=3+5;	加算	8
a=5-3;	減算	2
a=5*3;	乗算	15
a=15/3;	除算	5
a=16%3;	剰余（余り）	1
b=3; a=b+3;		6
a=3; a=a+3;		6

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。この他に通常の数式同様に( )を使用して演算の順序を指定する事も可能です。

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%とdの間に桁数を指定する事ができます。%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。

aの値	%d	%4d
1	1	1
12	12	12
123	123	123
1234	1234	1234

［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 x,y;

   x=3;y=5;
   printf("x+y=%d  x-y=%d \n",x+y,x-y);
}
```

上記のプログラムを実行すると、

```
x+y=8  x-y=-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 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。

```
scanf("%d",&i);      変数iにキーボードから入力された10進数を入れる
```

&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は形式として%dが指定された場合には、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。(scanf.c)

```
#include <stdio.h>

main(){
   int x,y;

   printf("x = ");scanf("%d",&x);
   printf("y = ");scanf("%d",&y);
   printf("x+y=%d, x-y=%d, x*y=%d, x/y=%d\n",x+y,x-y,x*y,x/y);
}
```

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］

3つの数を入力するとその平均を出力するプログラムを作れ。ただし変数は2個しか使えないものとする。(heikin3.c)

```
x=10△
y=30△
z=20△
average=20
```

5.5 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;                条件を満たせばAを実行する
```

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;    条件を満たせばAをそうでなければBを実行する
```

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。
```

などがあります。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。

［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");
}
```

［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を作れ。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)

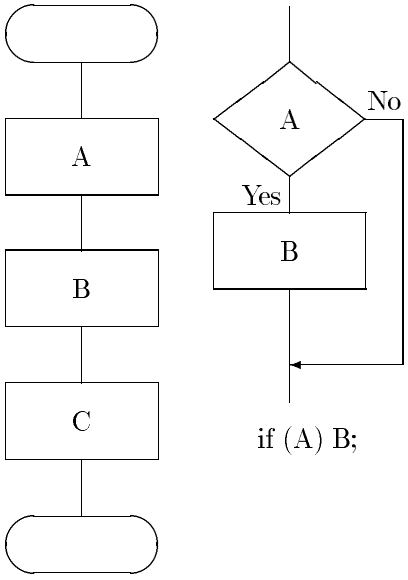
```
Height=177△        Height=150△                    Height=190△
Income=1500△       Income=2000△                   Income=600△
I love you.        Bye bye, you are too small.    Bye bye, you must work hard.
```

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を作れ。(max3.c)

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

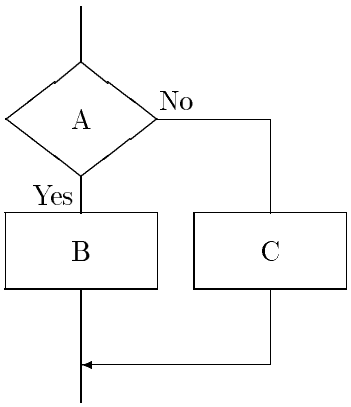
5.6　フローチャート (流れ図)

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。

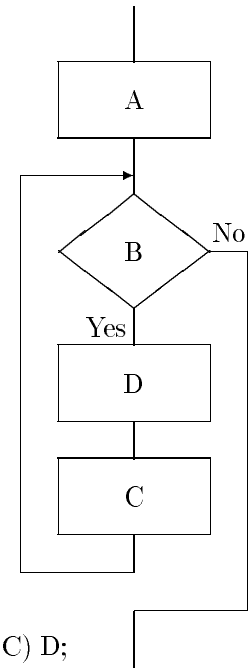


A; B; C;

if (A) B;



if (A) B; else C;



for (A;B;C) D;

必ずしもC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。

5.7　for文

for文はC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 ;      Dを繰り返し実行する
```

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。判りにくい方はフローチャートの例のところにこのfor文をフローチャートで書いたものがあります。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<5;i++) {
   printf("HaHaHa\n");
   printf("CyaCyaCya\n");
}
```

```
for (i=0;i<5;i++)
   printf("HaHaHa\n");
printf("CyaCyaCya\n");
```

出力：

```
HaHaHa
CyaCyaCya
HaHaHa
CyaCyaCya
HaHaHa
CyaCyaCya
HaHaHa
CyaCyaCya
HaHaHa
CyaCyaCya
```

```
HaHaHa
HaHaHa
HaHaHa
HaHaHa
HaHaHa
CyaCyaCya
```

［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<200;i++) printf("cat ");
}
```

［演習問題］

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

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

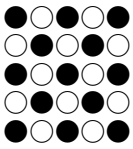
2. for 文を二重に使用して、次のような九九の表を出力するプログラムを作れ。(kuku.c)

```
***  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
```

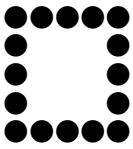
3. 指定した大きさの箱を次のように文字を使用して表示するプログラムを作れ。if 文や%を使用すると簡単にできる。(box.c)

Size=5△



4. 指定した大きさの箱を文字を使用して表示するプログラムを作れ。ただし大きさの指定は 1 以上とする。(ring.c)

Size=5△



5. 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、3kakua.c、3kakub.c、3kakuc.c)

```
Size=8△          Size=7△          Size=6△          Size=5△
x                xxxxxxx             x             xxxxx
xx               xxxxxx             xx              xxxx
xxx              xxxxx             xxx               xxx
xxxx             xxxx             xxxx                xx
xxxxx            xxx             xxxxx                 x
xxxxxx           xx              xxxxxx
xxxxxxx          x
xxxxxxxx
```

5.8 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);
}
```

実行例

```
N = 5△
Sum = 15

N = 10△
Sum = 55
```

［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)

   ```
   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)

   ```
   N=15△
   Kisuwa=64
   ```

［応用問題］

1. ${}_nC_r$(n個の中からr個を取り出す組み合わせの数: ${}_nC_r = \frac{n!}{(n-r)!r!}$)を計算するプログラムを作れ。ただし繰り返しはforを1個だけ使用すること。(ncr.c)

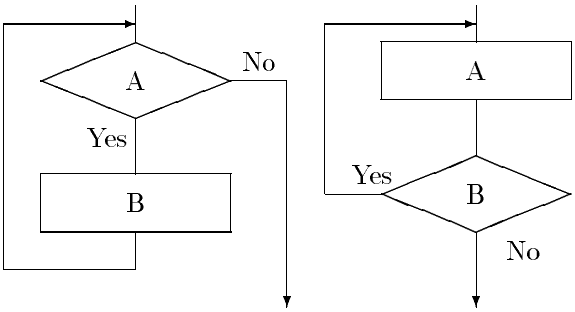
   ```
   N=5△
   R=3△
   nCr=10
   ```

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。(kisuwaa.c)

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

5.9 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;
== for (;A;) B;

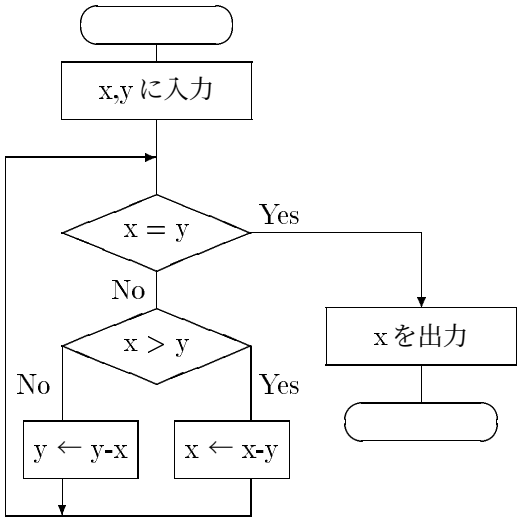
do A; while (B);

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)




```
X=12△
Y=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る。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)

```
x = 5△
x = 16
x = 8
x = 4
x = 2
x = 1
```

［応用問題］

1. 最小公倍数を求めるプログラムを作れ。(lcm.c)

```
X=12△
Y=15△
L.C.M.=60
```

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る。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。演習問題と計算は同じだが出力の形式が異なる事に注意せよ。(test231n.c)

```
x = 5△
1 : 16
2 : 8
3 : 4
4 : 2
5 : 1
```

5.10　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);
}
```

実行例

```
Data = 5△
Data = 3△
Data = 10△
Data = 0△
Max  = 10
```

［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)

```
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）
```

［応用問題］

1. 任意の数のデータの最小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としデータの終わりを示すのに0を用いるものとする。(mmin1.c)

```
Data=20△
Data=30△
Data=10△
Data=40△
Data=0△
Min =10
```

2. 任意の数のデータの最小値を求めるプログラムを作れ。ただしデータの終わりを示すのに0を用いるものとする。なお以下の例のように、最初からデータの終わりの0が入力されても対応するようにする。(mmin2.c)

```
Data=20△
Data=30△
Data=-10△
Data=40△
Data=0△
Min =-10
```

```
Data=0△
No Minimum
```

5.11　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。

5.12　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。。。         */
         break;                               /*  forを中断する。                          */
   if (i==s) printf("%dは素数である。\n",s);  /* forを完了して出てきたときはi==sである。*/
   else      printf("%dは素数ではない。%dで割れる。\n",s,i);
}
```

［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)

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

［応用問題］

1. 与えられた数を素因数分解するプログラムを作れ。(bunkai1.c)

```
N=120△
120=2*2*2*3*5
```

2. 与えられた数を素因数分解するプログラムを作れ。(bunkai2.c)

```
N=120△
120=2^3*3^1*5^1
```

5.13　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。

このとき[ ]の中は変数を含む数式でも構いません。変数の値によって同じものが別の変数を意味するようになります。例えば、scanf("%d",&x); はいつも変数xに入力する意味ですが、scanf("%d",&a[i]); は変数iの値が0ならばa[0]に入力しますし、変数iの値が9ならばa[9]に入力します。だからと言って、宣言した範囲を越えた値を変数iに入れて使用してはいけません。通常[ ]の中の値は宣言した範囲内かどうか調べないので、プログラムや他の変数の内容を破壊する事になります。

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");
}
```

```
How many data? = 4△
data=1△                1はdata[0]に入る
data=2△                2はdata[1]に入る
data=3△                3はdata[2]に入る
data=4△                4はdata[3]に入る
 4 3 2 1
```

i--はi++の反対の働きで、変数iの内容を1減らします。

［演習問題］

上記のプログラムを改良し、データの終わりを0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)

```
data=5△
data=8△
data=9△
data=0△
 9 8 5
```

［応用問題］

分散を定義通り計算するプログラムを作れ。データは1つ以上あり、全て正の値とし、データの終わりは0で示すものとする。分散は与えられた各データから平均を引き、自乗したものの総和を個数で割って求められる。(variance.c)

```
data=10△
data=30△
data=20△
data=40△
data=0△
Mean=25  Variance=125
```

5.14　文字型変数

Cのプログラムでは文字と文字列を区別して扱います。文字は俗に半角と呼ばれる大きさの文字1つを示します。プログラム中では、'a'の様に文字を'(Single quote)で囲います。一方文字列は文字が0個以上連続したものと定義されます。プログラム中では、"abc"の様に文字を"(Double quote)で囲います。0個でも文字列ですので、""も文字列です。また文字には空白も含みますので、"I have a pen."も1つの文字列です。

これらの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。文字列を入れるためには文字型の配列を用います。配列には文字列を構成する文字が1つずつ入りますが、それに加えて最後には必ず'\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。

word	[0]	[1]	[2]	[3]	[4]	...	[29]
内容	'b'	'a'	't'	'\0'	?	...	?

文字型の配列に入った文字列は、1文字ごと扱うことも可能です。上記のような状態で word[0]='h'; を実行するとwordの中の文字列は"hat"になります。ただ一文字しか含まない様な"a"と言う文字列でも'a'と言う文字とは異なるので注意して下さい。

```
#include <stdio.h>

main(){
   int i,Len;
   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にする。&や[ ]は不要 */
   for (Len=0;word[Len]!='\0';Len++) ;     /*  変数1の値を文字列の終わりまで増やす       */
   printf("Word Length=%d\n",Len);
   for (i=Len-1;i>=0;i--)                  /*  単語を逆順に出力する                      */
      printf("%c",word[i]);                /*  文字を出すときには%cを用いる              */
   printf("\n");
}
```

```
Word = computer△
Word Length=8
retupmoc
```

これまでscanf()やprintf()の中で%dは数値を意味していました。同様に%sは文字列を、%cは文字を意味します。文字列の入力の時だけちょっと違うので注意して下さい。

	整数型	文字型	文字列型
宣言	int n;	char c;	char s[80];
入力	`scanf("%d",&n);`	`scanf("%c",&c);`	`scanf("%s",s);`
出力	`printf("%d",n);`	`printf("%c",c);`	`printf("%s",s);`
代入	n=10;	c='x';	不可

［演習問題］

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

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

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

2. 入力された文字列の先頭3文字と残りの文字列を別々に表示するプログラムを作れ。なお、入力される文字列の長さは3文字以上あるものとする。(for文を1個でやろう)(cut3.c)

   ```
   String=personal△
   head=per, body=sonal
   ```

3. 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)

   ```
   Sentence = I have a book△
   Word = I
   Word = have
   Word = a
   Word = book
   ```

5.15　文字列を扱う関数

C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 を文字配列変数（char s[100],t[100];のように宣言した物）とします。

```
gets(s)      sにキーボードより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。
strstr(s,t)  sの中にtが含まれるかどうか調べます。なければNULLを返します。
```

Cでは文字列を代入する事はできません。つまりs=”abc”;はできないので、かわりにstrcpy(s,”abc”);を使います。gets()とscanf()の違いは、入力されたものに空白が含まれるときに、gets()ならば空白も含めて全て変数に入りますが、scanf()ならば空白の手前までしか変数に入りません。puts(s)は、`printf("%s\n",s)`と全く同じです。

注意:これらの関数のうちgets()、puts()以外を使用する際には、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。

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

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

   printf("Sentence = ");gets(sentence);
   printf("Sentence Length = %d\n",strlen(sentence));
   strcpy(sentence,"");strcpy(word,"");
   do {
      strcat(sentence," ");
      strcat(sentence,word);
      printf("Word = ");gets(word);
   } while (strcmp(word,"end")!=0);
   puts(sentence);
}
```

```
Sentence = I love you△
Sentence Length = 10
Word = I△
Word = love△
Word = you△
Word = end△
  I love you
```

［演習問題］

1. 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)

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

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)

   ```
   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.
   ```

5.16 ポインタ

Cを学ぶ上での難関の一つと言われるのが**ポインタ**です。しかしこれを理解しないと書けないプログラムがかなり存在します。またこのポインタの考え方はコンピュータのかなり基本的な動作から来ています。既に配列と言うものを学びましたが、これの処理の際にコンピュータはポインタに相当するハードウェアを使用しています。

ポインタは文字列などの大量のデータを扱う場合によく用いられます。現実世界のものに例えれば、大量の現金の代わりに用いる小切手のようなものです。家などを購入した場合、何千万円分もお札を持っていくのは大変です。その代わりに銀行にこれだけのお金を置いてあるよ、と言う情報である小切手を使います。これならば何億円になっても紙一枚です。長大な文字列を直接渡す代わりに、その文字列がメモリーのどこからあるかを知らせるようにすると、文字列の長さに関らず同じ大きさの情報になるので扱いが簡単になります。

情報がどこにあるかを記憶する変数が**ポインタ変数**です。情報の型によって整数ポインタ変数とか文字ポインタ変数のようになります。ポインタ変数の宣言は変数名の前に*を付けることにより行います。char *p;と記述するとpと言う名前の文字ポインタ変数が使えるようになります。

このように宣言したpには次のような代入が可能になります。

```
char *p;
p="abcdefg";
```

pには文字列”abcdefg”の先頭のアドレス(番地)が入ります。これをprintfで次のように使用することができます。

```
p="abcdefg";
printf("そのまま=%s\n",p);
p=p+3;
printf("ちと違う=%s\n",p);
```

最初のprintfでは「そのまま=abcdefg」と表示されますが、後のprintfでは「ちと違う=defg」と表示されます。また個々の文字を扱いたい場合には、次のように記述します。

```
p="abcdefg";
printf("先頭=%c, 3文字後=%c\n",*p,*(p+3));
```

このprintfは「先頭=a, 3文字後=d」と表示します。さてp=”abcdefg”;の場合は”abcdefg”は変数ではないので中味の変更はできません。中味を変更したい場合は文字配列の中に入れる必要があります。文字配列に入れた文字列のアドレスをポインタ変数に入れるには次のようにします。

```
char s[80],*p,*q;

strcpy(s,"abcdefg");
p=s;
q=&s[3];
printf("先頭から=%s, 3文字後から=%s\n",p,q);
```

配列の先頭アドレスを入れるにはp=sのように配列変数の頭の部分だけ書きます。途中からの場合はq=&s[3]のように&を利用してs[3]の文字のあるアドレスを入れます。配列変数に入った文字列を一文字ずつ表示するプログラムは次のようになります。

```
#include <stdio.h>

main(){
   char word[80],*p;
```

```
   printf("Word = ");gets(s);
   for (p=s;*p!='\0';p++)
      printf("%c",*p);
   puts("");
}
```

ポインタ変数を扱う際に注意しなければならないのは、*を付ける場合と付けない場合を混同しないこと、必ず代入してから使うことです。

［演習問題］

1. 「文字型変数」の章の演習問題をポインタ変数を用いたものに書き直せ。

2. 入力された文字列を半分ずつ表示するプログラムを作れ。なお、入力される文字列の長さは2文字以上あるものとし、forなどの繰り返しやポインタ変数を用いないものとする。(cut2.c)

   ```
   String=personal△
   head=pers, body=onal
   ```

5.17 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を{ }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。

［演習問題］

インチキ占いプログラムを作れ。生年月日と占いたい年月日を入力してもらい、それぞれの年月日の積を加えたもの(生まれた年*生まれた月*生まれた日+知りたい年*知りたい月*知りたい日)を7で割って余りを求める。その値が、

- 0ならば、「最高です」と1〜3のメッセージ
- 1ならば、「新しい恋人に出会うでしょう」と2〜3のメッセージ
- 2ならば、「思わぬ収入があるでしょう」と3のメッセージ
- 3ならば、「待ち人来る」
- 4ならば、「風邪をひくでしょう」

- 5ならば、「財布を落とすでしょう」と4のメッセージ
- 6ならば、「最低です」と5〜4のメッセージ

を表示する。ただしプログラム中に同じメッセージが何度も出てこないようにすること。(uranai.c)

```
あなたの生まれた年 = 1959△
あなたの生まれた月 = 1△
あなたの生まれた日 = 8△
知りたい年 = 1999△
知りたい月 = 6△
知りたい日 = 10△

あなたの1999年6月10日の運勢は、
        新しい恋人に出会うでしょう。
        思わぬ収入があるでしょう。
        待ち人来る。
```

5.18 関数の定義のやり方(1)

C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() の内容
}

main(){
   main() の内容
}
```

main()の内容で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を、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。つまり次のような場合には、i=3 と表示されます。よく間違える点なので、注意してください。

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

tekitou(){
   int i;  /* tekitou() の中では、変数 i は局所変数  */
   i=100;  /* これは局所変数 i に入れるだけ          */
}

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

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。最終的には 4 つの値が height に渡されるために、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］

人間対人間用三目並べのプログラムを作成せよ。関数を利用してできるだけ短いものを目指すこと。(ox1.c)

```
***  Tick-Tack-Toe Game  ***
```

```
     |   |
   1 | 2 | 3
     |   |
  ---+---+---
     |   |
   4 | 5 | 6
     |   |
  ---+---+---
     |   |
   7 | 8 | 9
     |   |

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

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

現在の盤面の状況を記憶するためにbanと言う配列を使用することにする。例えばban[3]の内容は右上隅のマスの状態を示し、0ならば空白、1ならばO、-1ならばXがあるものとする。

プログラムの概略は以下のようなものとし、省略されているinput()やdisplay()の定義の部分を補え。

```
#include <stdio.h>

int ban[10];          /*  桝目は9つだが、1から9を利用するため。 */

input(int x){
   ...                /*  input()関数の中身をここに記述する     */
}

display(){
   ...                /*  display()関数の中身をここに記述する   */
}

main(){
   int i,j;

   puts("  ***  Tick-Tack-Toe Game  ***\n");

   for (i=1;i<=9;i++) ban[i]=0;  /*  全てのマスを空にする   */
```

```
   display();          /*  ban[ ]の表示    */
   j=1;                /*  jでどちらの番かを記憶する。        */
   for (i=0;i<9;i++) {
      input(j);        /*  空いているところに入力してもらう  */
      j=-j;
      display();       /*  ban[ ]の表示    */
   }
}
```

display() の内容をさらに別に定義した関数を複数回呼ぶようなものにすると、全体の行数を減らすことができるので、是非試みること。なお display の中で for 文などの繰り返しは使用しないこと。

5.19 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 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。(前節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から返す値になります。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（$n! = n*(n-1)*(n-2)*...*3*2*1$）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。

［演習問題］

1. n 個のものから r 個取り出す組み合わせの数を${}_nC_r$と記述する。これに関しては、次のような性質が知られている。

   - ${}_xC_1 = x$
   - ${}_yC_y = 1$

- ${}_nC_r = {}_{n-1}C_{r-1} + {}_{n-1}C_r$

この性質を利用して${}_nC_r$を求める関数ncr()を定義せよ。(ncrf.c)

ヒント：関数の先頭部分はint ncr(int n, int r){のようになります。

2. 三目並べのプログラムにおいてme()と言う関数を定義する。これは呼び出し元から数値を一つもらい、その番号のマス目に相当する文字を返すものである。これを利用してdisplay()関数を定義せよ。(ox2.c)

ヒント：変数xに数値(0〜9)が入っているときに、その値の文字と同じ文字('0'〜'9')を返すには、return('0'+x)と書けばよい。

5.20 式の値

C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では文法に従った正しい記述とみなされます。

［演習問題］

三目並べのプログラムに次のような改良を行なえ。(ox3.c)

1. owari()と言う関数を定義する。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を返す。どちらも3つ並んでいない場合には、0を返すような関数である。

2. main()に手を加えて、毎回盤面を表示してからowari()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する。

5.21 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から約21 億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ので工夫が必要になります。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10回振るプログラムです。

```
#include <stdio.h>

main(){
```

```
   int  i;

   for (i=0;i<10;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10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。

これでは困ることが多いので、通常はrand()の計算の元になる数を設定する関数srand()を利用します。ただこの関数を利用してもそこで設定する値はプログラム起動する度に異なる値を設定しないといけないので面倒です。ここではシステムが実行プログラムごとに設定するプロセス番号を求めるgetpid()と言う関数の結果を利用しています。

```
#include <stdio.h>

main(){
   int  i;

   srand(getpid());  /*  実行時のプロセス番号をsrand()に与える。  */
   for (i=0;i<10;i++) printf("%d ",(rand()%6)+1);
   printf("\n");
}
```

```
3 2 5 2 1 6 5 6 1 2

3 2 5 4 3 2 3 4 5 4
```

［演習問題］

三目並べのプログラムに次のような改良を行なえ。(ox4.c)

1. Xの順番の時に、適当に空いた所にXを入れる関数umeru()を作れ。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を入れるものである。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()を呼ぶようする。

2. 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。

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()を作れ。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を返すようにすること。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()を実行するようにする。

［応用問題］

三目並べのプログラムをrand()を使わずに人に負けないものにせよ。

5.22 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。例えば文書が保存できないワープロソフトでは困ります。通常ファイルからプログラムにデータを取り出すことをファイルの読みだし、プログラムからファイルにデータを入れる事をファイルへの書き込みと言います。ファイルにデータを入れることにより、コンピュータの電源が切れてもデータを残すこともできますし、フロッピー等を使用してデータの交換も可能になります。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、

1. 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。通常1つのファイルに同時に読み書きを行うことはしません。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。そのポインターとはそれを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関数にはこのポインターの値が必要です。

2. 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。

3. 全てのデータを読み込んだ後は、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)と読み出し(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); /*   ファイルから読みだした値を画面に表示    */
   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);
}
```

ファイルを読み込もうとしたが指定したファイルが存在しない場合には、fopenはNULLと言う特殊な値を返します。またファイルに書き込もうとしたが何等かの理由でできない場合も同様です。

［演習問題］

1. 実行する度に1つずつ大きな値を表示するプログラムを作れ。(count.c)

2. 指定したファイルが存在すればyes、存在しなければnoと答えるプログラムを作れ。(aru.c)

```
File Name = count.c
Yes
```

5.23 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);
}
```

またファイルから1行丸ごと読み込みたい時にはfgetsを用いますが、この関数はファイルの終わりまで読んでしまってもうデータが無い場合にはNULLと言う値を返します。また読み込んだ文字列の終わりには'\n'が付くので注意が必要です。

```
#include <stdio.h>

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

  fp=fopen("test.c","r");
  while (fgets(line,80,fp)!=NULL) printf("%s",line); /* 80は配列の大きさ */
  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. 指定したファイルの行数を数えるプログラムを作れ。(line.c)

```
File Name = test.c△
There are 5 lines.
```

2. 指定したファイルの先頭の5行を表示するプログラムを作れ。ただし元々内容が5行よりも少ないファイルに対しても適切に対応できるようにせよ。(head.c)

3. 指定したファイルの最後の5行を表示するプログラムを作れ。ただし元々内容が5行よりも少ないファイルに対しても、行数が非常に多いファイルにも適切に対応できるようにせよ。(tail.c)

5.24 実行時のパラメタの取得

Cで書いたプログラムをコンパイルすると、その結果はUnix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。(何をコンパイルしても全てa.outになるが、必要であれば名前を変更すれば良い。) ところが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rm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。

```
[miwako]%rm aaa△
```

上の例では、rmがコマンド名でaaaがパラメタになります。C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の数が入り、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次のように実行すると、

```
[miwako]%a.out 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]

1. 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]

1. 指定したファイルの行数を数えるプログラムを作れ。ただしファイルの指定はパラメタの形で行うものとする。(linec.c)

```
[miwako]%a.out test.c△
There are 5 lines.
```