放送大学集中面接授業

プログラミング実習 (C 言語)

2008 年度 1 学期

（神奈川学習センター）

講師：坂口　利裕[1]

実施日 2008.8.7〜8

[1] 公立大学法人 横浜市立大学 国際総合科学研究院 *E–mail* : sakkun@yokohama-cu.ac.jp

目 次

図 目 次

表 目 次

第I部

本文

はじめに

この講義では，コンピュータの仕組みを概説し，その機能を自由に使うための技術— **プログラミング** — についての実習を行います。

２ 日間のわずかな時間で行いますので，コンピュータの基本的操作方法 — キーボードやマウスの使い方 — についてはほとんど触れません。つまり，コンピュータ操作の初心者向けではありません。反面，内容的には，ごく入門的なものにならざるを得ません。プログラミングという技術を完全に体得するには時間が必要ですし，技術そのものも変化してきているからです。

そういう意味では，キーボードに多少慣れてきた方に**取っ掛かりとしてのプログラミングの体験**をしてもらうことと，**ハードウェアとソフトウェアとの関わり合い方を理解**してもらうこと，に絞って進めて行こうと思います。

この２ 日間では，次のような事柄について学習していくことを目標としています。

- ハードウェアの仕組みとソフトウェアの役割
- オペレーティングシステムとアプリケーションとの関係
- プログラミングの一般的知識
- プログラミング言語によるデータの表現とデータ加工の表現
- アルゴリズムによる問題の定式化とデータの構造化
- 発展的学習のためのヒント

授業は，全体での解説，コンピュータと体を使った実習，頭を使った練習，の繰返しで進めていきます。また，１時限の間にも１度か２度休憩をはさんでいきます。解説の場面では，じっくりと話を聞いてもらいますが，その他の場面では面接授業であることの利点を最大限に生かすよう，質問を積極的に行なってください。自分だけが分かっていないのでは，と思うことはありません。そのような場合，他の人も同様に分かっていないことの方が圧倒的に多いものです。

中級者の方は

なお，受講者の中には，(C 言語に限らず）過去にプログラミングを経験されている方（中級者）もいらっしゃると思います。そのような方にとっては，少々物足りない授業内容となるのは否めません。そこで，中級者の方にも時間を有効に使っていただくために，演習問題を準備してあります。実習用パソコンの基本操作が理解できたら，授業の進行とは別に自由に取り組んでみて下さい。

第1章　プログラミング総論

1.1　ハードウェアとソフトウェア

実際のプログラミングを学習する前に，コンピュータの構成要素— ハードウェアとソフトウェア — の整理をしておきましょう。

1.1.1　ハードウェアの概要

現在のコンピュータのハードウェアは，図 1.1 に示されているように，大きくは5つの装置から成り立っています。

全体の動作を決める**プログラム**（命令コード＝**マシン語**）は主記憶装置にあり，マイクロプロセッサによって自動的に読みとられ，同じく主記憶装置にあるデータを加工していくように作られています。主記憶装置上のデータやプログラムは二進数で表現された数値（信号）であり，マイクロプロセッサから見ると両者の区別は基本的にはありません。

黎明期のコンピュータでは，プログラムといっても数多くのスイッチの切り替えや配線の組み替えによって行っていました[1]。

この方式は，融通性には欠けるものの動作は速くなるという利点があります。しかし，コンピュータの能力が次第に評価され，動作速度が改善されてきた結果，より融通性のあるプログラム方式が求められ，ハードウェアではなくソフトウェアによる方式（**ストアードプログラム**）が一般的となってきたわけです。

このことにより，同一のハードウェア上で「ワープロ」や「表計算」「ゲーム」といった異なるプログラムを利用することが可能となってきたわけです。

1.1.2　ソフトウェア—OS とアプリケーション

ここで問題となるのは「利用したいプログラムをどのようにして主記憶装置にセットするか」になります。実は，プログラムを主記憶装置にセットするためにもプログラムを用います。このような目的に作られたものを一般に**基本プログラム**あるいは **オペレーティングシステム**（略して **OS** [2]）と呼んでいます。

---

[1] **ワイヤードプログラム**と呼ばれることがあります。世界初の電子計算機といわれる ENIAC（アメリカ，図 1.2 参照）もこの方式によるものです。このため現在のコンピュータと方式が異なるので,「世界初とは呼べない」という見方もあります。

[2] Operating System

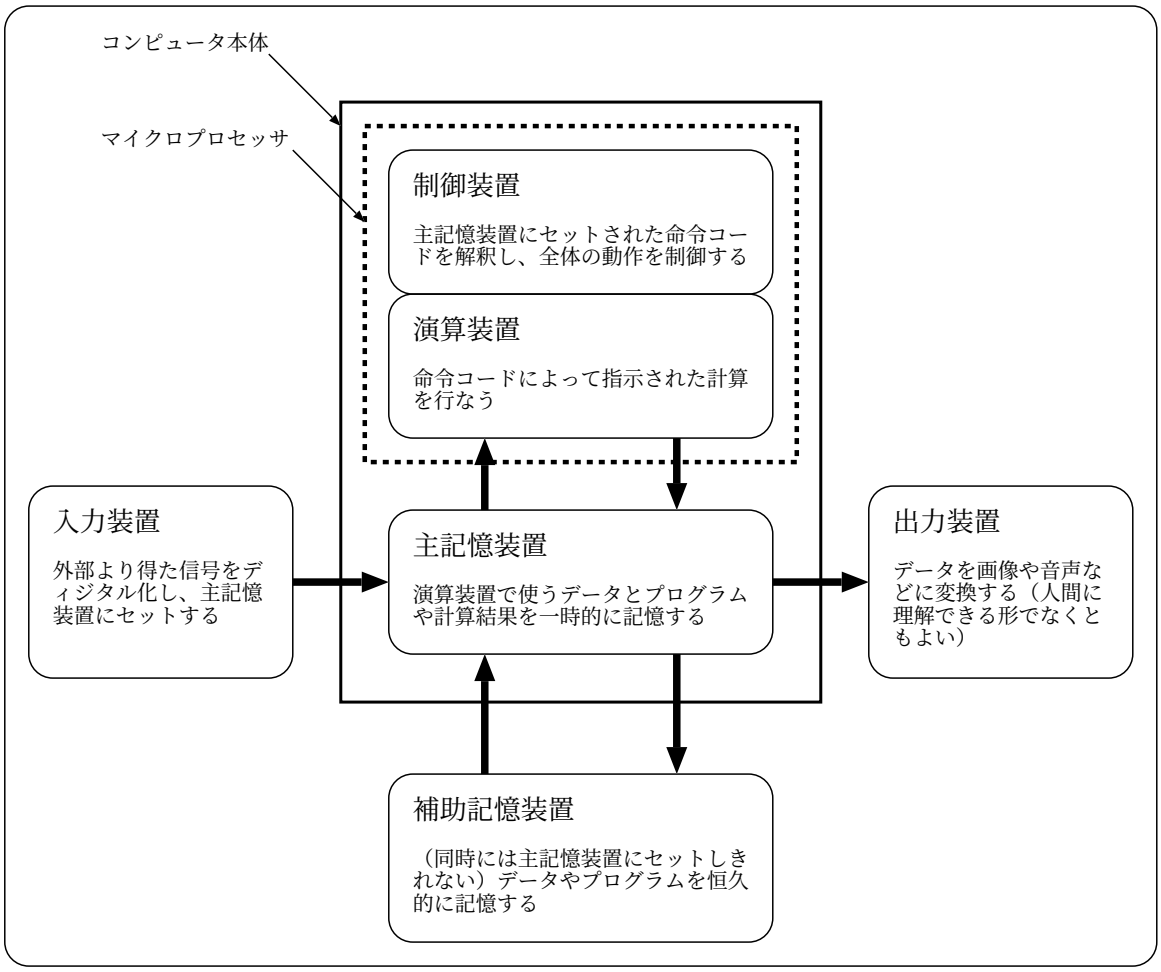


図 1.1: コンピュータの基本構成

OS は，補助記憶装置上にデータやプログラムを**ファイル**という単位で整理・保管し，必要に応じて主記憶装置にセットする機能や，複数のプログラムを疑似的に同時進行させる機能（**マルチタスク機能**），入出力装置の制御を行い，ユーザからの操作命令に従って動作させるといった機能（**ユーザインタフェースの提供**）など，コンピュータを利用するためになくてはならない機能を持っているわけです。

ところで，基本プログラムである OS はどうやって主記憶にセットされるのでしょうか？これは，主記憶装置の一部分を **ROM** [3]で作っておき，ここに OS を読み込むのに必要なプログラムを記憶させておくことで実現しています。そのようなプログラムを **IPL** [4]，

[3]Read Only Memory。読み出し専用メモリのこと。これに対して内容を自由に変更できるメモリを **RAM**（Random Access Memory）と呼ぶ

[4]Initial Program Loader

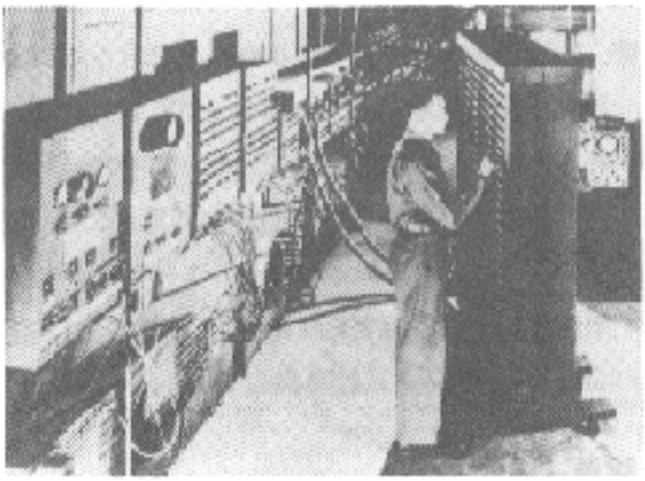


図 1.2: 世界初の電子計算機 ENIAC

ブートプログラム，あるいは，**BIOS**[5]と呼びます。

これに対して，具体的な特定の目的を持つプログラムを，**応用（アプリケーション）プログラム**（俗に，ソフト，アプリなどとも）と呼びます。「ワープロ」も「表計算」も応用プログラムの一種というわけです。

1.2 プログラミング言語

プログラムの表現形式あるいは表記のルールには，さまざまなものがあります。意思疎通に使われるという類似性から，これらの表現形式や表記ルールのことを**言語**と呼んでいます。ここでは，プログラミングのための言語 — プログラミング言語 — 全般について整理しておきましょう。

1.2.1 低級言語と高級言語

前節で概観したように，コンピュータのプログラムの実体は主記憶装置（メインメモリ）に記憶されたマシン語であり，これをマイクロプロセッサが自動的に読みとり順番に実行していきます。

このマシン語は，二進数で表現された命令コードですので，人間にとっては無意味な数字でしかありません。したがって，これらの数字を適切な順序に並べて意味のある働きをコンピュータにさせようという作業（つまりプログラミング）は，人間とってはやや苦痛を伴う作業になってしまいます。また，ハードウェアが備えている機能，あるいは，一つ

[5]Basic Input Output System

の命令コードで行える作業というのは，我々が想像するよりもシンプルで低機能です[6]。このため，マシン語を使って高機能なプログラムを作ろうとすれば，それが「キーボードから数字をいくつか読みとって和をディスプレイに表示する」といった最も簡単なプログラムであっても，数十から数百の命令コードを組み合わせなくてはなりません。

そこで，まず，命令コードを記号におきかえてプログラムを組み立てるという形になってきました。この記号による表現のことを**ニモニック**[7]**表記**と呼び，ニモニック表記によって体系づけられた言語を**アセンブリ言語** と呼びます。

さらに，よく使われる（あるいは共通に使われる）機能を一つの記号で表現し，プログラムはその記号の組み合わせによって組み立てていくという形になってきました。つまり，一つの記号で指示できる動作の内容が多くなった，あるいは高機能になったということです。ハードウェアのレベルから見れば，より高機能な記号によってプログラムを表現できるようになったということで，現在の一般のプログラミングはこれらの言語（**高級言語** ）を用いて行います。これに対して，アセンブリ言語（やその実体であるマシン語）のことを**低級言語** と呼びます。低級言語は，コンピュータの構造などの知識を要することや，誤りの訂正作業[8]が困難なため，ごく限られた場合にのみ使われるようになってきました。

1.2.2　高級言語の種類

ところで，一口に高級言語といっても，用途によって様々なものがあります。汎用的なプログラミング言語だけを挙げても表 1.1 のようにさまざまです。

表 1.1: 汎用プログラミング言語の種類

名称	名称の由来
BASIC	Beginner's All-purpose Symbolic Instruction Code
Fortran	FORmular TRANslator
COBOL	COmmon Business Oriented Language
C	（next B）
C++	increased C（C をひとつ繰り上げたもの ++は C の演算子）
Pascal	（人名）
PL/I	Programming Language One
LISP	LISt Processor
Prolog	PROgramming in LOGic

代表的なものの一般名称のみ挙げてあります。この他にも Perl（パール），Java（ジャヴァ），Tcl（ティックル），Ruby（ルビー） なども利用されています。

[6]おそらくは想像を絶するほどにシンプルです。事実，パソコン用のプロセッサでは，計算命令としては，算術演算（加減乗除）と論理演算（AND，OR，NOT など）操作程度のものしか準備されていません。

[7]mnemonic

[8]プログラムの誤りをバグ（bug=虫）と呼ぶことがあります。この虫を退治するという意味で訂正作業をデバッグ（debug）と呼びます。

汎用的な言語に対して，特定目的に設計されたものとして表 1.2 のようなものが挙げられます[9]。

表 1.2: 特定目的のプログラミング言語の例

用途	代表的なプログラミング言語
統計処理	BMD, S, SPSS, SAS
シミュレーション記述	DYNAMO, GPSS
ページ記述	TEX, PostScript
ページ記述（ハイパーテキスト）	HTML

1.2.3 C 言語

さて，この講義でとりあげるのは，C 言語です。適用分野も幅広く，利用できる環境もパソコンからスーパーコンピュータにまで拡がっているので，現在では，最も利用されている言語といっていいでしょう[10]。

プログラミングの学習用・入門用としては敬遠されがちですが，言語それ自体の文法は他の言語に比べるとむしろ平易で，覚えるべきキーワード（固有の働きを表わす単語）も少ないのが特徴です。キーワードが少ないことは，反面，同じ内容のプログラムを何通りにも表現できることにつながるので，大規模なプログラムを作成する時にはそれなりの注意が必要となりますが，最初の取っ掛かりのための言語として十分利用できます。

1.3 プログラミングツール

人間向けの表記ルールにしたがったプログラムを，ハードウェアの理解できるマシン語へと換える必要があります。そのためのプログラム— プログラミングツール — についてまとめておきます。

1.3.1 プログラミングツールの役割

前節では，いわば「共通規格」としてのプログラミング言語について概説しました。

しかし，現実には，その規格に則って表現された文書をコンピュータという機械が理解できる形にする必要があります。つまり，人間にとって都合のいい形で表現されたものを本来のマシン語に翻訳するための手段が必要となります。

[9] ちなみに，このテキストは TEX を使用して作成しています。

[10] **インターネット**を使って入手できる**フリーソフトウェア**のほとんども C 言語で作られています。ソースファイルまで公開されている場合に限られますが，入手したプログラムに自分なりの改良を加えられるという点で，C 言語の内容が読めるということは，かなりのアドバンテージになります

現在では，こうした作業を補助するためにも，ソフトウェアを使って行います。このソフトウェアを一般的に**プログラミングツール**あるいは**言語プロセッサ**と呼び，言語別・機種別・価格別にソフトウェア会社から販売されています。フリーソフトもあります。

1.3.2　プログラミングツールの内容

一般には，次のような名称のソフトウェアを組み合わせて使います。

- テキストエディタ
  プログラミング言語で表記されたプログラムをコンピュータファイルとして保存したり，訂正を加えるための道具。簡単なワープロだと考えればよい。
- コンパイラ
  テキストエディタで作成されたプログラムをコンピュータ向けの命令コード（マシン語）に翻訳する。インタプリタと異なり，プログラム全体を一括処理する。
- インタプリタ
  プログラムを対話式にコンピュータ向けの命令コード（マシン語）に翻訳する。コンパイラと異なり，プログラム全体を一括処理しないため，動作をチェックしながら作成作業をすすめられる。
- リンケージエディタ（リンカー）
  共通の機能（データの入出力など）をまとめたライブラリプログラムと，コンパイラによって得られたユーザ作成のプログラムを結合して，コンピュータで実行できる形式に整える。
- ローダ
  実行形式のプログラムを主記憶装置に読み込み，実際の動作を行なわせる。通常，OS に機能の一部として組み込まれている（独立したプログラムではないことが多い）。
- デバッガ
  実行形式のプログラムのエラー訂正作業（デバッグ）に使用する。本来は一括で実行されるプログラムを文単位に実行させたり，メモリーの内容を表示させる機能を持っている。文法的なエラーではない，論理的なエラーの発見に主として用いる。

最近では，Windows などのような **GUI**[11] 環境に対応して，これらのソフトウェアを統合したような製品も多くなっています。

1.3.3　実習室のプログラミングツール

実習室のパソコンには，プログラミングツールとして，授業で使用する Microsoft 社の統合型開発環境 (IDE[12]) Visual Studio .NET（ドットネット）の他，Sun Micro System 社の Java 言語開発キット (SDK[13])，汎用の統合型開発環境 Eclipse などが組込まれています。

ここでちょっと休憩

[11] Graphical User Interface
[12] Integrated Development Environment
[13] Software Development Kit

第2章　実習（1）

【重要】

実習室のパソコン利用にあたっては，部外者が操作できないように，利用開始時に各自に与えられている**ユーザ名**と**パスワード**とが必要です。事前に，事務室の窓口にて各自で確認しておいてください。
実習室備え付けの利用者用マニュアルもありますので，適宜参考にしてください。

2.1　実習のための準備

いよいよ，プログラミングの体験学習を始めます。まずは，実習室のパソコンの起動と，学習用のプログラムのダウンロードです。

2.1.1　パソコンの起動と Windows へのログオン

キーボードと本体の関係やスイッチ類の場所をあらかじめ確認しておいてから，電源を入れて下さい。

電源を入れてしばらくすると，画面上に「ログオンするには Ctrl+Alt+Del キーを押してください」というメッセージのボックス[1]が表示されます。指示に従って，キーボードの Ctrl キーを押しながら（指は離さず），さらに Alt キーを押しながら，Delete キーを押して，その後，すべてのキーから指を離します。

「ログオン情報」というタイトルのダイアログが表示されたら，キーボードを使って，「ユーザ名」欄と「パスワード」欄に正しく文字を入力します。なお，パスワード欄に入力した文字はすべて「*」という文字になりますので，文字数だけ確認してください。

OK ボタンをクリックしてしばらく待つと， Windows のデスクトップが表示されます。

2.1.2　プログラムのコピー

インターネット上に用意したプログラムを個人用のディスク領域にコピーして利用します。以下の説明に従って操作してください。

1. デスクトップ上の「Internet Explorer」をダブルクリックします。放送大学のトップページが表示されます。

[1] Windows の世界では**ダイアログ**と呼んでいます。

2. 「アドレス」をクリックして，キーボードから
   `http://sakkun.cc.yokohama-cu.ac.jp/`
   と入力し，Enterキーを押します。

3. 表示されたページの中から「2008 年度担当授業一覧」をクリックし，さらに一覧表の中の「放送大学」「プログラミング実習（C 言語)」の「教材」をクリックします。

4. 画面の中から 「プログラム例のセットアップ」 の部分をクリックします。

5. ダイアログが開かれたら,「実行」をクリックします。セキュリティの警告ダイアログが表示されてもそのまま続行させてください。

6. 保存先を示すダイアログが開かれたら,「z:￥」であることを確認して，OKをクリックします。

7. 自動的にプログラムの解凍が行なわれ，z:ドライブに samples フォルダが作成され，このフォルダ中にソースファイル kinshu1.c，ftest1.c〜ftest3.c，sample1.c〜sample9.c がコピーされます。

8. Internet Explorer を閉じます。

9. ファイルが正しくコピーされたことを確認するには，スタートボタンから「マイドキュメント」（z:￥ドライブに与えられている別名＝個人の記憶場所）を起動することで参照できます（ファイルをダブルクリックすると，手順がややこしくなるので，ファイル名の確認だけにしておきましょう)。

以上で，プログラムのコピーは終りです。誤操作などでプログラムを削除してしまわない限り，繰り返して操作する必要はありません

2.2 プログラミングツールの起動

いよいよ，プログラミングツールを使ってのプログラミング体験です。

2.2.1 プログラミングツールの起動

ダウンロードしてきたプログラムは**ソースプログラム**と呼ばれる状態であるため，後述する**ビルド**操作をしないとその機能を利用すること（**実行**）ができません。ソースプログラムを使って，実行可能な状態のプログラム（**実行可能プログラム**）を作成したり，機能の一部を改変（**編集**）するためにプログラミングツールを使用します。

より具体的な手順は，別冊資料「操作手順補足資料」（以下【補足資料】と表記）を準備してありますので，こちらを参照してください。

【補足資料】の手順１〜３に従って，プログラミングツールを起動します。

2.2.2 プログラムの読込み

【補足資料】の手順４～８に従って，実行させたいプログラムをプログラミングツールの中に読み込んできます。

まず， sample1.c から始めます。したがって，手順５で設定する プロジェクト名 [2]は sample1 となります。最後に，画面上に第 II 部に掲載してあるリスト（sample1.c）の一部が表示されれば OK です。

内容の確認は，通常のワープロなどと同様に必要に応じてスクロールバーをマウスで操作するか，カーソル移動キー（←・→・↑・↓）やページ制御キー（Page Up・Page Down）で操作します。画面上での色の違いなどを観察してみましょう。

2.2.3 プログラムの翻訳と実行のさせ方

【補足資料】の手順９～１０に従って，ビルド（翻訳）作業とプログラムの実行を行います。

教材によっては，あえて未完成のままのものもありますが，sample1.c の場合は，何も手を加えずに，手順通りに行えば正常に翻訳ができるはずです。

ビルドが正常に終ったことを確認したら，続けて実行させてみましょう。実行ウインドウに

```
This is a sample program.
Press any key to continue
```

と表示されるはずです。最初の行は，このプログラムによって得られた実行結果そのもので，２行目は，実行がすべて終った後に表示されるメッセージです。

動作を始めたプログラムに不具合があって（意図に反して）停止しない場合は，キーボードの Ctrl を押しながら C のキーを押すと，止まります。

2.3 プログラム例（sample1.c）の内容

最初に触れて頂いたプログラムは，画面上にメッセージを表示する，といった内容を持つプログラムです。具体的に，ソースプログラムのどの箇所でそれが表現されているか，じっくりと観察してみましょう。

[2]一つの実行プログラムを複数のソースプログラムを使って作ることがあり，互いに関連するソースプログラムの集合体のことを**プロジェクト**と呼んでいます。

2.3.1　C 言語による画面表示（データ出力）

このプログラムのポイントは，34 行目の printf というキーワード[3]で始まる部分です。観察することで，一対の記号文字 " で括られた部分が，ほぼそのまま画面に表示されたことに気がつかれるでしょう。

例外は，\n（画面上では￥n となっているかも知れません）となっている部分で,「画面上で改行せよ」という特別な文字（改行文字）の表記ルールに従っています。この改行文字が含まれていないと，表示される文字列の長さが画面の幅を超えるまで文字が次々と横に並んでいきます。例外のルールは，他にもいくつかありますが，とりあえず，改行文字の表記だけ了解しておいてください。これ以外の文字は，画面にそのまま表示されるというわけです。

2.3.2　C 言語の文法

各プログラムの部分には，そのプログラムのポイントとなる事項について簡単に説明を記入してあります。この部分は，プログラムの実行内容にはまったく無縁の「覚書き」としてだけの役割を持っています。この部分 — 注釈 — の書き方には早めに馴染むようにしてください。なお，より詳しい説明は，本文中のプログラムの説明や文法編（第 III 部）も参考にしてください。

2.4　プログラム例（sample1.c）の改変

それでは，前節の内容が本当に飲み込めたか，プログラムに手を加えてみることで，検証してみましょう。

2.4.1　プログラムの編集方法

プログラムの中には，意図的に不具合が生じるようにしてあるものを混ぜています。このようなプログラムの場合には，内容を各自で手直ししてから実行させる必要があります。実験的にエラーメッセージの表示のされ方を見てみるというのも，ひとつの学習の仕方ですが，やはり，正常な動作をするプログラムに仕立ててみないとつまらないでしょう。

編集作業の方法は，いわゆるワープロと同様に考えてください。以下に，編集方法の概略のみを整理しておきます。

- 修正箇所への移動は，カーソル移動キー（←・→・↑・↓）を用いるか，マウスのクリックによって行なう。
- 文字の挿入モードと上書きモードとの切り替えは Insert キーを使う。Visual C++の起動直後は，挿入モードで始まっているので，通常は使用する必要はないが，うっかり上書きモードにしてしまったような場合に使用する。

[3]正確にはキーワードではありません。正しくは関数名と呼びます。

- 削除は，Back Spaceキーか Deleteキー を用いる。
- 日本語入力の ON/OFF は，半角/全角キーを押す。
- 表示画面の移動は，スクロールバーをマウスでドラッグするか，Page Up・Page Down キーを用いて行なう。

2.4.2 プログラムの改変

では，以上の編集方法にしたがって，sample1.c に手を加えましょう。

改変の内容は，いたって単純です。「画面上に表示される内容を好みのものに変えて」みてください。日本語入力に慣れている方は，日本語のメッセージでももちろん構いません。

プログラムに手を加えたら，ビルド操作を行う前に,「ファイル」メニューから「sample1.c の保存」を選びます（いわゆる「上書き保存」をしておきます)。ビルド操作がうまく正常終了したら，もう一度実行させてみましょう。

なお，オリジナルのプログラムを別に保存しておきたい場合は，**編集作業に入る前に**，以下の作業をしておきます。

1. 「ファイル」メニューから「名前を付けて sample1.c を保存」を選び，たとえば sample1a.c などに付け変えておきます。
2. この場合，エディタウィンドウは付け変えた後のファイルに切り変わってしまうので，必要が無ければ，いったん「ファイル」メニューから「閉じる」で，改名後のファイルを閉じておきます。
3. 元のファイルを呼び出すために,「ソリューションエクスプローラ」の「ソースファイル」にあるファイル名をダブルクリックします。

ここでちょっと休憩

第3章　実習（2）

3.1　プログラム例（sample2.c）の実行

sample1.cは，プログラミングツールの使い方を知ってもらうための例題でもありました。続いては，簡単な計算を行なうためのプログラムに挑戦してみます。

3.1.1　ソースプログラムの保存とワークスペースの解放

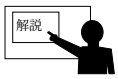
画面上には，まだ，sample1.cが見えている状態だと思います。このままでは，別のプログラムに対する操作ができませんので,「ファイル」メニューから「**ソリューションを閉じる**」を使って，必ずプログラムのウィンドウを閉じておきます。この手順を省いてしまうと，別のプログラムを正しく実行することができませんので注意してください。

3.1.2　プログラム例（sample2.c）の実行

うまく空っぽの状態になったら,【補足資料】の手順４～８を繰返して別のプログラムを呼び出して実行させてみます。

順序にしたがって，sample2.cを使いますから，プロジェクト名をsample2に読み変えることなどに留意しつつ操作してください。実行方法はまったく同じですから，各自で手順の復習を兼ねて，プログラムを実行させてみてください。

3.2　問題の抽象化



sample2.cでは，簡単な計算問題を表現しています。通常のプログラムでは，与えられたデータに応じて何らかの加工（計算とは限りません）を行ないます。その手始めとして重要な概念 — 変数と，変数を使った問題の抽象化 — がこのプログラムのポイントです。

3.2.1　問題の抽象化（あるいは定式化）

プログラミングでは，解きたい問題を算数の文章問題を解くようなやり方で，抽象化しておくことが重要になります。素朴な計算問題では，いわゆる「公式」を思い浮かべればよいでしょう。

つまり，

底辺の長さが 6m で高さは 4m であるような三角形の土地がある。
この土地の面積を求めなさい。

というのが「具体」的な問題であるのに対して，数値によらない以下のような表現を用いるのが「抽象」的問題というわけです。

底辺の長さが D で高さは H であるような三角形の土地がある。
この土地の面積を求めなさい。

私たちは，三角形の面積の公式 — つまり面積を S とすれば，

$$S = \frac{1}{2}DH$$

で求められること — を知っています。いわゆる「代数」の基本が理解できているかどうかが肝心であるといえます。

3.2.2 変数の作り方

プログラミング言語は，ほとんどの場合,「代数（あるいは変数）」によって問題を解くように作られています。変数は，一定のルールの下で表記する記号です[1]。上にあげた三角形の求積問題では，S，D，H をそのまま変数として利用できます。

3.2.3 公式の作り方

変数と変数との関連を等式で表現したものを公式と呼んでいるわけですが，プログラミング言語ではどのように表現してもいいわけではありません。表現上は，公式なのですが，プログラミング言語の世界では，これは「ある計算の手続き」を示すものとして扱われます。たとえば，$4x + 3 = 0$ というのも，一種の公式と考えられなくはないですが，プログラミングでは，これをもう少し丁寧に x の計算手順を示すように，$x = -3 \div 4$ の形で表記せねばならないということです。このように，等式の左側（左辺）は，必ず単独の変数となるようにしておきます。

このようなことから，プログラミング言語における公式は，通常，**代入（文）**という言葉で表現されます。

[1] 詳細は文法編（第 III 部）を参照してください

3.2.4 C 言語における変数名の決定

変数名は，アルファベットで表記されるルールなので，互いに重複しないように気をつけて決定します。変数の名前に迷ったら，ローマ字表記を採用するのが手っ取り早いでしょう。英単語を使ってもよいのですが，後述の「関数」や決まった目的にしか使えない「予約語」と重複する場合もあるので，（すべての予約語と関数名を暗記しているなら別ですが）避けた方が無難です。

底辺の長さ	→	D あるいは teihen
高さ	→	H あるいは takasa
求める土地の面積	→	S あるいは menseki

3.2.5 変数の属性の決定

変数の名前が決まったら，その属性を決めます。当面は数値しか扱わないことにして，「整数」か「実数」かだけを考えます。迷うなら実数を使います。こうして決めた属性に対して，int あるいは float というキーワードの後に変数の名前を並べます。最初の変数名とキーワードの間は，１個以上のスペースを置き，同じ属性のものどうしは，カンマ「,」で区切ってまとめておきます。最後の名前の後ろには，一つのプログラム文の終端を示すセミコロン「；」を置いておきます。

```
底辺の長さ，高さ，求める土地の面積はいずれも半端 (小数点以下) が
ありえるので,「実数」(float) の属性とする
  ↓
float teihen, takasa, menseki ;
```

3.2.6 代入文の表現

計算式で表現できる演算記号[2]に注意して，あらかじめ考えておいた「公式」を（変数を使って）表記します。セミコロンを忘れがちなので気をつけましょう。

```
float teihen, takasa, menseki ;
menseki = teihen * takasa / 2 ;
```

プログラムの主要な部分はこれでおしまいです。思ったより簡単でしょう？

3.2.7 計算結果の表示

計算結果の表示には，printf を使います。

```
printf("Menseki = %g\n", menseki) ;
```

[2]詳細は文法編（第 III 部）を参照してください

さあ，ここまでの部分を順序よく並べると，次のようになります。変数の属性の指示は必ず先頭に置きます（途中に置くと，ルール違反となって「文法エラー」と扱われます)。

```
float teihen, takasa, menseki ;
menseki = teihen * takasa / 2 ;
printf("Menseki = %g\n", menseki) ;
```

これで，ほぼ完成ですが，前後におまじないをつけておきます。おまじないの意味については，当面は深く考えなくてかまいません。

ところで，このままでは，肝心のデータがありません。とりあえず，代入文を使って具体的な数値を与えておくことにします。なお，すべての文は上の行から下の行に向かって（同じ行の中では左から右に向かって）順番に実行されていきますので，データを与える部分は，少なくとも公式（代入文）による計算処理よりも上の行に置きます。

```
#include <stdio.h>
int main() {
  float teihen, takasa, menseki ;
  teihen = 6 ;
  takasa = 4 ;
  menseki = teihen * takasa / 2 ;
  printf("Menseki = %g\n", menseki) ;
  return 0 ;
}
```

プログラム例 sample2.c の本質の部分は，このようにして組み立てられたものだったわけです。

3.3 プログラム例（sample2.c）の改変

前節の説明を十分飲み込めれば，sample2.c の全体の様子が見えてきたことでしょう。そこで，sample1.c と同様に，sample2.c にも手を加えてみましょう。

3.3.1 改変のテーマ

次のような改変を実際に行なってみてください。

- データの数値を適当なものに変更して，計算結果が変わることを確認する。
- 底辺と面積を与えて，この三角形の高さを求めるプログラムに変えてみる。
- 円の半径をデータとして与えて，この円の面積を求めるプログラムに変える。

ここでちょっと休憩

第4章　プログラムの構造化（1）— 条件判断 —

4.1　プログラムとデータの分離

変数を使うことによって「公式」による計算問題を表現できることは理解できたと思います。次のステップとして，異なる値のデータに対する答をプログラムそのものは改変せずに行なうための手段 — データ入力 — について学習しましょう。

4.1.1　データ入力

プログラム例 sample2.c では，データ（底辺が 6，高さが 4 であるといった情報）をプログラムの内部で表現してありました。しかし，これではこれ以外の数値の答を得ようとすると，練習して頂いたように，プログラムそのものの改変を必要とします。

これでは，せっかく「公式」で表現したことのメリット — 問題の抽象化 — が，生かしきれていません。そこで，データをプログラムの内部に埋め込むのではなく，プログラムの動作時点でキーボードから直接取り込めるように改変してみます。

C 言語では，これらデータの読み取りのためにも関数を使用します。表記のルールにしたがっておけば（そのからくりを知らなくとも），キーボードから，データを直接読み取ることができます。データの読みとりには，関数 scanf を使用します。データを与えていた行を，それぞれ次のようにそっくり入れ替えます。この時，プログラム中には具体的な数値が不要になることに注意してください。**プログラムが動作を始めてから**，キーボードで任意の数値を入力できるように書き直そうとしているのですから。

```
teihen = 6 ;   →   scanf("%g", &teihen) ;
takasa = 4 ;   →   scanf("%g", &takasa) ;
```

scanf を用いてキーボードから**実数型（float 型）のデータを読ませる**には，上の例のように，まず，第 1 引数の文字列に %g を書いておきます[1]。次に，第 2 引数として，キーボードから与えた数値を記憶させたい変数の名前に記号&を添えたものを書いておきます。この記号の意味については，当面はおまじないと考えておきましょう。

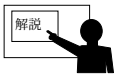
実際に改変して実行させると，実行ウィンドウには，これまでとは違って，すぐには答が出てきません。キーボードから 6 Enter と押すと，最初の scanf に対応した動作が行な

[1] 読ませたいデータが整数型（int 型）の場合には，代わりに%d を用います

われ，変数 teihen に数値 6 がセットされます。続いて，4 Enter と押すと，変数 takasa に数値 4 がセットされ，続いて計算が行なわれて答が表示されます。

このような改変によって，別の数値のデータでも，プログラムを改変することなく（実行操作は必要ですが）計算させることが可能となります。すなわち，プログラムとデータとが分離された形となったわけです。

4.2　条件判断



さて，公式で表現できるような計算問題であれば，プログラムで表現できるようになりました。しかし，コンピュータの能力はこれだけではありません。次のステップとして，**条件判断**を行なわせるプログラムの表現について学びましょう。

4.2.1　if 文による条件判断

新たに登場するキーワードは，if です。このキーワードを用いると，データに応じた計算 — たとえば，数値の大きさによって公式が変わってくるような場合 — の表現が可能になります。表記の例は，文法編を参照して頂くとして，具体的な問題で考えてみましょう。

sample2.c で，三角形の求積問題をとりあげましたので，ついでにというわけではありませんが，三角形の種別を判定するプログラムを考えてみます。

図 4.1 のような三角形を考えて，各辺の長さを a, b, c とします。この a, b, c の大小関係によって，三角形の種類 — 正三角形とか二等辺三角形などの区分 — を判断するプログラムを作成してみます。

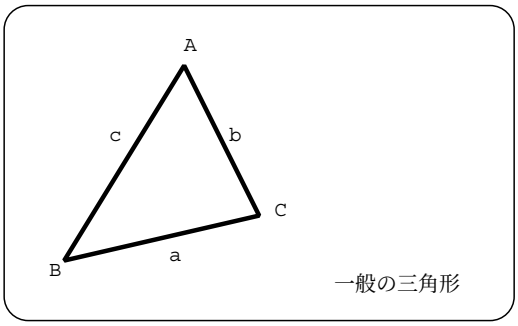


図 4.1: 三角形と辺の長さ

いきなり，すべての種類を判断するのは難しいかも知れませんので，まず，一番特徴的な正三角形であるかどうかを判断してみましょう。言葉で表せば,「すべての辺の長さが同じ」ものが正三角形です。

4.2.2 条件の表現

C 言語では，ある値と別の値が等しいかどうか，などといった状態を**関係演算**で表現することができます。ただし，常に２つの値の関係しか判断できないので，３辺が相等しいということを表現するには，一工夫必要です。３つの辺の長さが等しいことは，例えば，a と b が等しく，**かつ**，a と c が等しい，と２つの関係式に分解します。結果として，C 言語では，

```
a == b && a == c
```

と表現できます。b == c は必要ないことは，ちょっと考えれば分かりますね？
これを if と組み合わせれば,「正三角形かどうかを判断する」文は,

```
if( a == b && a == c ) printf("正三角形です\n") ;
```

と表現できます。この時，printf は，条件が満たされれば実行されますが，条件が満たされない場合には（あたかも無かったかのように）実行されません。このようにして，条件に応じて，実行する内容を制御することが可能になるわけです。

4.3 プログラム例（sample3.c）の実行

前節の内容を具体化したプログラムを実行させてみましょう。このプログラムは，データ入力を必要としますので，動作し始めたら適当な数値を３回入力してください。

4.4 プログラム例（sample3.c）の改変

このプログラムでは，辺の大きさの判定のみを行なっています。より緻密に判定できるように，角度の判定を行なわせるようにしてみましょう。

4.4.1 判定方法の吟味

まず，オリジナルのプログラムの内容を吟味してみましょう。このプログラムのポイントは，もちろん，if の使い方です。C 言語のプログラムのままでは，分かりづらいので，**フローチャート**と呼ばれる図式を用いて表現してみましょう。

フローチャートに用いる図式は，厳密には 10 種類以上に及びますが，簡単には，図 4.2 で示されている図式の使い分けができれば十分です。

このうち，if による条件判断に対応するには，菱形の図式です。菱形のいずれかの頂点に，流れ込むような矢線を描き，別の２つの頂点から流れ出るように矢線を描きます。これらを用いると，オリジナルの sample3.c の判定部は，図 4.3 のように表現できます。

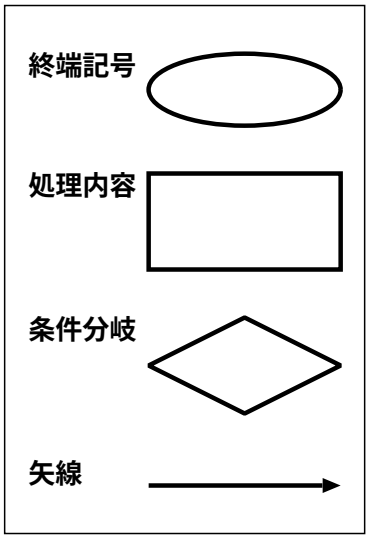


図 4.2: フローチャートで使われる主な図式

ここで，角度の条件を追加するにはどうすればよいでしょう。頭の中だけではなかなか整理しづらいので，表形式にまとめてみましょう。表 4.1 のように，表頭（ひょうとう）に辺の長さによる分類，表側（ひょうそく）に角度による分類，とすると見通しがよくなります。表は，意図的に不完全な状態にしてありますので，各自で空欄の部分を埋めてみてください。

表 4.1: 辺の長さと頂点の角度による三角形の分類

		3 辺の長さの関係			
		3 辺が同じ	2 辺が同じ	すべて異なる	
頂点の角度の大きさ	1 つが鈍角				鈍角三角形
	1 つが直角				直角三角形
	すべて鋭角				鋭角三角形
		正三角形	二等辺三角形	不等辺三角形	

空欄にあてはまる言葉は，正三角形，直角二等辺三角形，鈍角不等辺三角形，のような名前です。空欄のままの部分（理論上あり得ない形）もありますので，ご用心。

うまく整理ができたら，続いてフローチャートの状態で，どのように判定していけば，効果的に分類できるか考えて，図式化してみましょう。

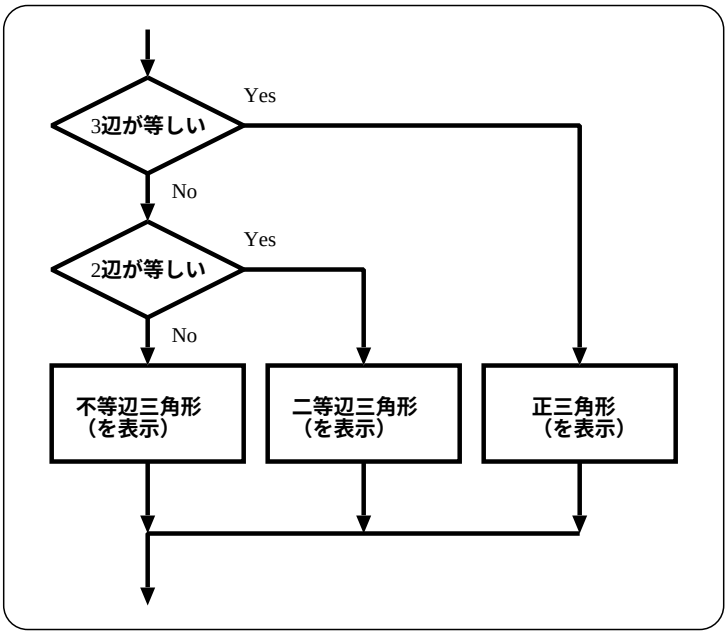


図 4.3: 三角形の種別判定を示すフローチャート

4.4.2 判定方法のプログラム化 — コーディング —

判定方法を十分吟味したら，具体的に C 言語での表現に作り変えてみます。なお，頂点のひとつが直角かどうかは，

```
a*a+b*b == c*c || a*a+c*c == b*b || b*b+c*c == a*a
```

で，判定できます。鈍角かどうかの判定はどうすればよいか，応用問題として考えてみましょう。

4.5 終了の手順

すべての練習問題をこなせた人も，そうでない人も，お疲れ様でした。最後に，起動したコンピュータの終了方法について説明しますので，必ず，以下の手順にしたがって電源を切ってください。

4.5.1 プログラミングツールの終了手順

作業を終えて，プログラミングツールを終了させる時は,「ファイル」メニューの中から「終了」を選ぶか，ウィンドウを閉じる操作を行ないます。作業中のプログラムに手を加えたまま，保管していないような場合には，警告が表示されますので，保管の要・不要をよく考えて，適切なボタンで応答してください。

4.5.2　Windows の終了

パソコンの電源を切る時は，通常の Windows の場合と同じように，起動してあるアプリケーションを終了させてから,「スタート」ボタン→「シャットダウン」と辿ってください。Windows がパソコンの電源を自動的に切ってくれますので，電源スイッチには触れずにしばらく待ちます。長くても 1 〜 2 分程度で自動的に電源が切れます。

第5章　実習（3）

5.1　プログラミング作業の流れ

1日目に行なった実習内容を，知識として整理しておきましょう。

5.1.1　プログラミングツールの役割の整理

プログラミングの一般的流れと，プログラミングツールとの関係を図 5.1 に示します。

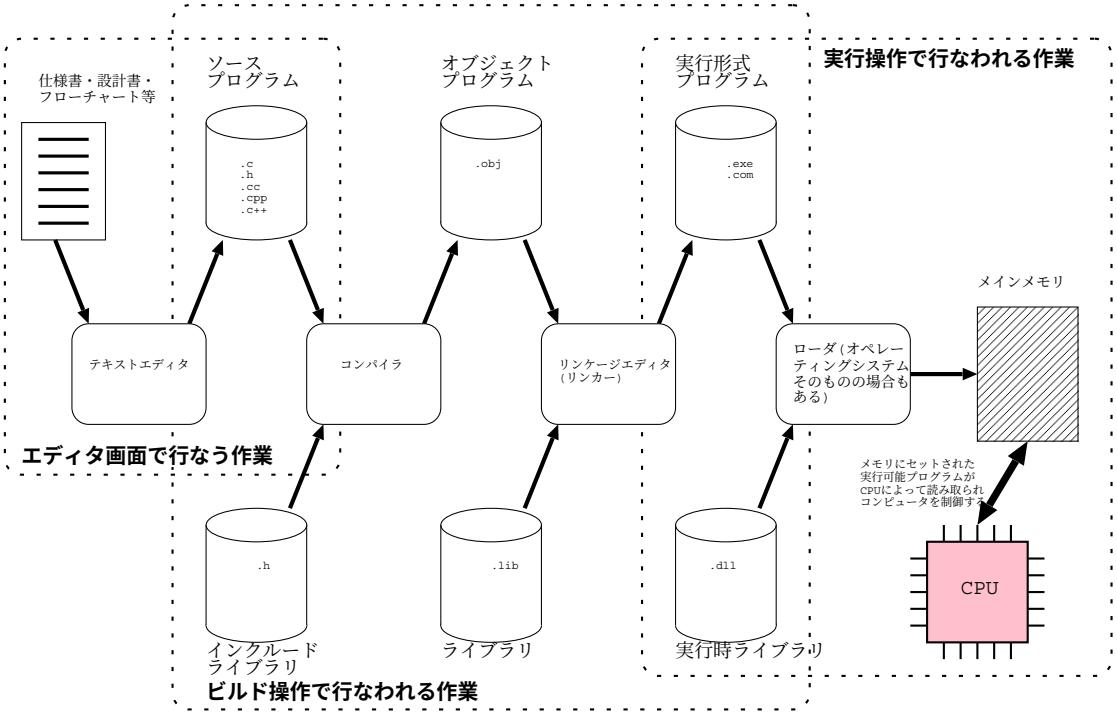


図 5.1: プログラムの作成・翻訳から実行までの流れ

授業では，統合型のツール Visual Studio .NET 2003 を使用したため，これほど明確には切り分けられていませんが，操作がどの段階に当てはまるか，図を参考に各自で整理しておきましょう。

5.2　プログラム例（sample4.c）の実行

第1日で登場した文法のほぼすべてを網羅したプログラムを用意してあります。内容は「じゃんけんゲーム」です。まずは，素直に実行させて遊んでみてください。

5.2.1　プログラムの内容

このプログラムの第1のポイントは，ぐう，ちょき，ぱあ，をそれぞれ 0, 1, 2 と数値化して表現している点です。このように,（コンピュータの得意な）数値データ以外の情報も数値化することで，コンピュータで処理することが可能になります。現実のデータをできるだけ効果的に表現するという技法も，プログラミングにおいては重要です。

第2のポイントは，**疑似乱数**を使っている点です。シミュレーションを行うような場合に，なるべくデタラメなデータを必要とすることがありますが,（正確さが命である）コンピュータでデタラメなデータを作り出すのは実は難しい話です[1]。とりあえず，コンピュータの内部でサイコロを振っているようなもの，と考えておきましょう。具体的には，関数 srand で乱数の系列の初期値を定めます。この初期値から始めて次々と乱数を取り出すのに，関数 rand を使用します。同じ初期値に対しては，同じ乱数系列が発生[2]しますので，これを回避するために，実行開始時点の時刻を関数 time で取り出して，これを初期値として利用しています。

5.3　練習問題（sample5.c）の改変

第1日の内容を，プログラム作成を通して着実なものにしてみましょう。

5.3.1　プログラムの完成

プログラムを読み込んでみればお分かりのように，このプログラムは，このままでは何もしないプログラムです。問題の定式化の練習にもなっていますので，プログラム内の注釈を手がかりに必要な文を追加して，完全なプログラムに仕上げてみましょう。

基本的には，注釈の下に空けてある行（たとえば，23 行目）に，その注釈に見合った文を追加していきます。大まかには，

- データ（身長，体重）用の変数名を決める。
- 計算結果（ローレル指数あるいは肥満度）用の変数名を決める。
- 身長・体重を読み取る文を考える。

[1] コンピュータによる乱数発生の手法についてだけで，本が1冊書ける程度の内容を含んでいる，ということで想像してください

[2] 再現実験のようなものであれば，これはこれで好ましい性質と言えます。

- 身長・体重から肥満度を求める文を考える。
- 計算結果を表示する文を考える。

という流れになるでしょう。

5.3.2 プログラムの改変

完成して，計算結果も正しく表示できるようになったら，以下のような改変を試みてください。

- ローレル指数の値は大まかには，100 未満だと痩せすぎ，140 を超えると太りすぎ，とみなされる。if 文による判定文を追加して，判定結果を表示できるようにしてみよ。
- ローレル指数は，肥満度を示す指標としてはやや古く，最近では BMI[3]が用いられる。定義式は，

$$BMI = \text{体重}\ (kg) \div \text{身長}\ (m)^2$$

（単位に注意）となっている。この BMI も計算して表示できるようにしてみよ。
- BMI による標準体重は，BMI の値が 22 となるような体重とされている。このことから，標準体重の値を表示できるようにしてみよ。
- 標準体重と実際の体重との乖離度を百分率で表示できるようにしてみよ。（文字 % を表示するには，書式文字列中で %% とダブらせておきます）

ここでちょっと休憩

[3]Body Mass Index の略

第6章　プログラムの構造化（2）— 反復処理 —

6.1　反復処理

単純な公式による計算や条件判断を伴うプログラムであれば，ここまでの知識で十分表現できるようになりました。しかし，コンピュータを使うことのメリットの一つに大量のデータを処理できるという側面があります。最初のステップとして，同種のデータに対する計算処理の反復の表現を学びましょう。

6.1.1　飛び越し

これまでのプログラムは，答を求めて結果を表示できるのは，１組のデータに限られていました。別のデータに対する答が欲しい場合は，再び実行操作をしなければなりません。そこで，複数組のデータに対しても１度の実行操作で済むように工夫してみましょう。

これまでの知識でも，次のように表現すれば，複数組のデータに対する計算が可能になります（与えられた半径を持つ円の面積を求めるプログラム，前後のおまじないの部分は省略してあります）。

```
float hankei, menseki ;
printf("Hankei ?") ; scanf("%g",&hankei) ;
menseki = hankei*hankei*3.14 ;
printf("Menseki = %g\n",menseki) ;
printf("Hankei ?") ; scanf("%g",&hankei) ;
menseki = hankei*hankei*3.14 ;
printf("Menseki = %g\n",menseki) ;
```

} 必要な数だけ繰返し

２組であればこのままで構いませんが，３組，４組，…，と増えてくると，いつかは行きづまってしまうのは目に見えています。そこで，goto というキーワードを使って，次のように表現してみましょう。

```
 float hankei, menseki ;
hajime:   この場所に「hajime」という名前を付ける
 printf("Hankei ?") ; scanf("%g",&hankei) ;
 menseki = hankei*hankei*3.14 ;
 printf("Menseki = %g\n",menseki) ;
goto hajime ;   「hajime」と名付けられた場所に実行順序が移る
```

} これらの文は再利用される

キーワード goto は，直後に示された名前の場所に実行順序を強制的に変更する，という働きを持っています。このことにより，上から下へという暗黙の実行順序を変えることができ，結果的に 1 度実行した文を再度実行させたり，if と同じように，文を飛び越して実行させるという表現が可能になります。

6.1.2　制限を設けた反復処理

さて，goto を単独で用いたままでは，計算が無限回行なわれることになります。意図的にこのような無限ループを構成することもありますが，一般には，たとえば 5 回だけ繰り返したい，というように回数を制限したり，正しいデータが与えられるまでは入力を繰り返す，というような条件を課します。

そこで，if を使って，反復の回数を制限してみます。何回繰り返したかを判断するために，カウント用の変数 c を追加します。

```
float hankei, menseki ;
int c ;
c = 0 ;   カウンタをリセット
hajime:   この場所に「hajime」という名前を付ける
printf("Hankei ?") ; scanf("%g",&hankei) ;
menseki = hankei*hankei*3.14 ;                  } これらの文は再利用される
printf("Menseki = %g\n",menseki) ;
c = c+1 ;   カウンタを 1 つ繰り上げる
if( c<5 ) goto hajime ;   「hajime」と名付けられた場所に実行順序が移る
```

6.1.3　構文による反復表現（1） — do 〜 while() —

if と goto を組み合わせると，制限を設けた反復が表現できることが可能になることは分かりました。C 言語では，反復の表現に向いた別の構文も用意されています。前述とまったく同じ内容を，do と while というキーワードを使って，次のようにも表現できます。

```
float hankei, menseki ;
int c ;
c = 0 ;   カウンタをリセット
do {   反復範囲の始まりを示す
printf("Hankei ?") ; scanf("%g",&hankei) ;
menseki = hankei*hankei*3.14 ;                  } これらの文は再利用される
printf("Menseki = %g\n",menseki) ;
c = c+1 ;   カウンタを 1 つ繰り上げる
} while( c<5 ) ;   反復範囲の終りと継続の条件を示す
```

この構文は，少なくとも 1 回は反復範囲の文が実行されることが特徴です。一般に，条件後置型あるいは **REPEAT–UNTIL** 型とも呼ばれます。

6.1.4 構文による反復表現（2） — while() と for() —

if による条件判断のタイミングを変えると，次のような表現でも反復を表現できます。

```
float hankei, menseki ;
int c ;
c = 0 ;
hajime:
if( c>=5 ) goto owari ;
  printf("Hankei ?") ; scanf("%g",&hankei) ;
  menseki = hankei*hankei*3.14 ;
  printf("Menseki = %g\n",menseki) ;
  c = c+1 ;
goto hajime ;
owari:
```

if と goto の部分を，while で置き換えると，次のような表現となります。ラベルが必要でなくなったことと，条件判断の式の違いに注意してください。

```
float hankei, menseki ;
int c ;
c = 0 ;
while( c<5 ) {
  printf("Hankei ?") ; scanf("%g",&hankei) ;
  menseki = hankei*hankei*3.14 ;
  printf("Menseki = %g\n",menseki) ;
  c = c+1 ;
}
```

for を用いると，次のようにさらに簡潔に表現できます。

```
float hankei, menseki ;
int c ;
for( c = 0 ; c<5 ; c=c+1 ) {
  printf("Hankei ?") ; scanf("%g",&hankei) ;
  menseki = hankei*hankei*3.14 ;
  printf("Menseki = %g\n",menseki) ;
}
```

いずれの場合も，最悪の場合には1回も反復範囲の文が実行されないことがある，という特徴を持ちます。一般に，条件前置型あるいは **DO–WHILE** 型とも呼ばれます。

6.2 プログラムの新規作成

反復処理の表現に慣れてもらうために，実際にプログラムを作成してみましょう。これまでとは違い，プログラム例としては準備してありませんので，新規にプログラムを作成していきます。

6.2.1　新しいプログラムの作り方

新規のプログラムを作る場合には，プログラミングツールを起動した後（別のプログラムを実行していた場合は，ソリューションを閉じて空っぽの状態にした後）,【補足資料】の手順１３〜１４に従って操作していきます。

プロジェクトを作成するところまでは，これまでと同じですが，今回はソースプログラム自体も新規作成するところが異なっています。

あらかじめ，ソースプログラムの名前を hanpuku.c [1]と決めておきます。したがって，作成するプロジェクト名も hanpuku とし，作成した空(から)の状態のプロジェクトに，新規項目として C++ソースファイルを追加するように指示をしていきます。ソースファイルの保管場所も例題と同じ z:￥samples にしておきましょう。

無事，空っぽの状態のウィンドウが開かれたら，C 言語のプログラムを入力していきます。行の最後は，必ず Enter キーで終えていきましょう。最初のビルド操作の前に上書き保存しておけば，実行の方法は，既存のプログラムの場合とまったく同じとなります。

6.2.2　練習プログラム（hanpuku.c）の内容

新規作成の状態になったら，次の内容を打ち間違いに注意しながら入力していきましょう。最初の数字は，行番号を示すものですから入力しません。

```
1  #include <stdio.h>
2  int main( )
3  {
4    int c ;
5    for( c=1 ; c<=10 ; c=c+1 ) {
6      printf( "%d\n", c ) ;
7    }
8    return 0 ;
9  }
```

プログラムの内容は,「1 から 10 までの数値を順番に表示」するというものです。

6.3　練習プログラム（hanpuku.c）の改変

練習プログラムが正しく動作するようになったら，これまでと同様に，プログラムに手を加えて,（for による）反復表現の方法の理解を深めてみましょう。

[1]Visual Studio で作成すると， C++ 言語 (C 言語の上位言語) のソースファイルとして追加されるため，拡張子.c の代わりに拡張子.cpp が付加されます。

6.3.1 改変のテーマ

次のような内容となるように改変してみてください。

- 1 から 20 まで，順番に表示されるようにしてみる。
- 50 以下の偶数のみを順番に表示されるようにしてみる。
- 100 以下の 7 の倍数のみを順番に表示されるようにしてみる。

ここでちょっと休憩

第7章　データの構造化（１）— 配列変数 —

7.1　単純変数と配列変数

大量データに対する処理は，反復処理で表現可能になりました。ここでは，次の段階として，１度読み込んだ大量のデータを２度以上に渡って再使用しなければならないような状況 — たとえば，100 組のデータを読んで，これを大小順に並べ替えて表示する— でのデータの記憶のさせ方について学びます。

7.1.1　配列変数

これまでの変数（区分するために**単純変数**とも呼ばれます）は，暗黙の内に「同時には１つの内容しか記録できない箱」として扱われています。つまり，新しい内容を記憶させた時点で，それまで記憶されていた内容は失われてしまうわけです。

計算の過程で，たとえば同時に 5 個のデータを必要とする場合には，異なる名前の 5 つの変数を用意しなければなりません。ところが，反復処理の解説でも行なったように，10 個，100 個，···，と数が増えるにしたがって，いつかは破綻してしまいます。

そこで，同じ名前でありながら，複数のデータを同時に記憶できる仕組みが考え出されました。これまでは「１つの変数には１つの記憶場所」を割り当てましたが，このような変数 — **配列変数** — は，１つの変数に複数の記憶場所を割り当てることができます。それぞれの記憶場所のことを**配列要素**と呼び，割り当てられた記憶場所の先頭からの順序番号を使って，データを記憶させたり，記憶させたデータを取り出して使用します。

C 言語では，配列変数は，通常の変数と同様のルールにしたがって名前を付け，配列要素のそれぞれを示すために，[ ] で括った要素番号（あるいは添字（そえじ））を具体的な数字，あるいは，計算式によって表現します。要素の数は，定義をする時点で決定しておきます。

なお，要素番号は，**0 から始まる整数**で表現します。たとえば，x という名前の 5 つの実数型の要素を持つ配列変数は，`float x[5] ;` のように宣言し，このうち 3 番目の要素を使用する時は，`x[2]` と表記します。

7.2　プログラム例（sample6.c と練習問題 sample7.c）の実行

この 2 つのプログラムは，5 つのデータの合計値を求める，という内容を，単純変数，配列変数で表現したものです。同じデータに対して，実行結果が一致することを確認してみてください。

7.3　練習問題（sample7.c）の改変

sample7.c では，要素番号に計算式も利用できるという特徴が生かされていません。要素番号が規則的な変化をしていることを利用して，反復処理で表現し直してみましょう。

7.3.1　データ入力の工夫

カウンタとして適当な変数を定義し，for() で表現してみましょう。

7.3.2　合計の求め方の工夫

合計は，オリジナルのプログラムではひとつの式で表現してありますが，これを，次のように分解すれば，解決の糸口が見えるでしょう。

```
goukei = 0 ;
goukei = goukei + data[0] ;
goukei = goukei + data[1] ;
goukei = goukei + data[2] ;
goukei = goukei + data[3] ;
goukei = goukei + data[4] ;
```

規則性をうまくとらえて，for() で表現してみましょう。

それぞれうまく表現し直せたら，データ数が 10 個の場合に対応できるように変えてみてください。

7.4　プログラム例（sample8.c）の実行

既出のじゃんけんゲームを配列変数で表現し直してみたものです。起こり得る状態が，限られているような時は，if の羅列で判定するよりも，このような**表引き**を用いた判定の方が高速になることがあります。新しい文法としては，switch と case による多重分岐の表現がありますが，文法編を参考に各自で解読してみてください。

ここでちょっと休憩

第8章　より複雑な問題の解き方

8.1　アルゴリズム

条件判断や反復処理の表現をひととおり学んだところで，より複雑な問題の解き方を考えてみます。一定の手順で計算や条件判断を繰り返すことで必ず答が導き出される方法を**算法**あるいは**アルゴリズム**と呼びます。アルゴリズムで重要なのは，正しく答が導けることはもちろんですが，できるだけ少ない手順でそれが実行できること，すなわち効率です。いかに効率のよいアルゴリズムを見つけることができるかが，よいソフトウェア開発の鍵と言っていいでしょう。しかし，古典的なアルゴリズムを知っておくことで，自力で開発する労力を軽減できます。ここでは，代表的なアルゴリズムをいくつか紹介し，具体的なプログラムとして表現してみます。

8.1.1　代表的なアルゴリズム（1） — ユークリッドの互除法 —

最も古くから知られているアルゴリズムに，2つの（正の）整数の最大公約数を求めるための，**ユークリッドの互除法**が挙げられます。

アルゴリズムは，誰がやっても必ず答が導き出されなければなりません。直観的に求めるといったあいまいさが入っていては失格です。正の整数を a，b とすると，次のように表現できます。

1. $a > b$ となっていなければ（$a \leq b$ であれば），a と b を交換する。
2. a を b で割った時のあまりを，あらたに a とする。
3. a が 0 でなければ，1. に戻って繰り返す。
4. a が 0 となったら，b が（元の a と b との）最大公約数である。

この手順 — すなわちアルゴリズム — に忠実にしたがえば，必ず答が求まります。実際に，紙と鉛筆で試してみてください。

アルゴリズムの形にさえなっていれば，プログラムとして表現することが可能です。以下は，C 言語による具体例です。実際に，`gcd.c` という名のファイルに作って試してみましょう（例によって左端の番号は行番号なので入力は不要）。

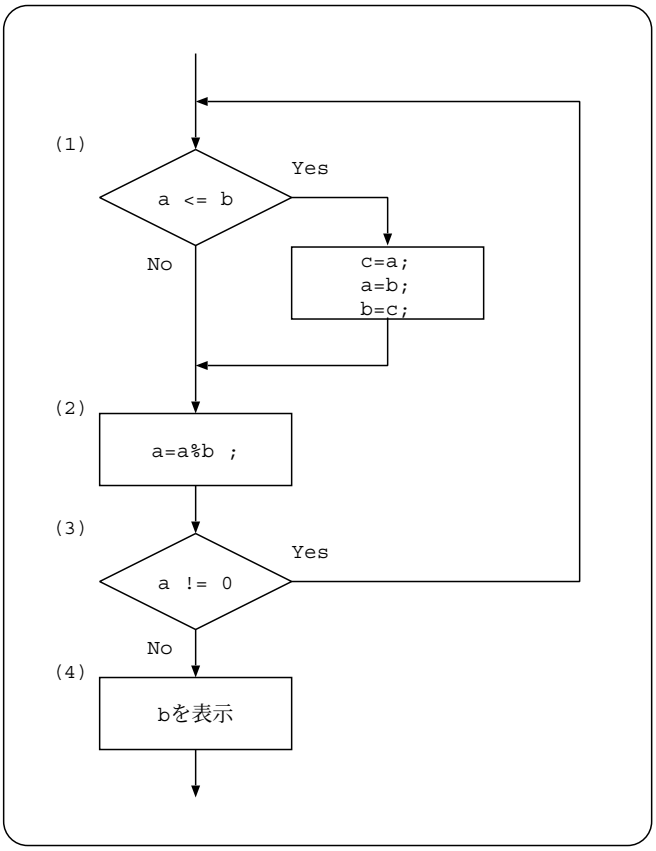


図 8.1: 最大公約数を求めるアリゴリズム（ユークリッドの互除法）

```
 1  #include <stdio.h>
 2  int main( )
 3  {
 4    int a, b, c ;
 5    printf("a ? ") ; scanf("%d",&a) ;
 6    printf("b ? ") ; scanf("%d",&b) ;
 7    printf("%d と %d の最大公約数は",a,b) ;
 8    do {
 9      if( a<=b ) { c=a ; a=b ; b=c ; } /* 1. */
10      a = a%b ;                        /* 2. */
11    } while( a>0 ) ;                   /* 3. */
12    printf( " %d です\n", b ) ;        /* 4. */
13    return 0 ;
14  }
```

8.1.2 代表的なアルゴリズム（2） — 並べ替え（ソート） —

コンピュータを使うことを前提としたアルゴリズムとしては，並べ替え（ソート）が筆頭に挙がります。実用的な面でも，ソートされたデータ群から特定のデータを見つけ出すのと，ソートされていないデータ群から見つけ出すのとでは効率に著しい差が出てきますので，非常に重要な技法であると言えます。

ソートされていない場合には，データ数 n のデータ群から特定のデータであるかどうかは，まさにシラミ潰しに（`if` を使って）検査する必要があります。1回の検査に一定の時間を要するので，データ数が100倍になれば検査時間も100倍 — 別の表現をするなら，データ数 n に比例した時間数 — かかるわけです。

これに対して，ソートされている場合には，検査の方法を工夫して，検査時間をデータ数の $\log_2 n$ 倍程度に抑えることができます。データ数が100倍になっても，$\log_2 100 =$ 約6.64倍にしかなりません。いかに，ソートされているデータ群が重宝なものかお分かり頂けるでしょう。

ソートの技法そのものもさまざまなものが提案されています。名前だけを挙げておくと，単純選択法，バブル（泡立ち）法，シェル法，クイック法などが代表的です。平均的な効率としては，シェル法やクイック法が一番良いと言われていますが，アルゴリズムはやや難解です。ここでは，最も単純な方法（半面，効率はよくない）である単純選択法について紹介しましょう。興味のある方は，アルゴリズムの入門書などを参考に勉強してみてください。

単純選択法による並べ替えのアルゴリズムは，次のように表現できます。

まず，n 個のデータ群を $x_1, x_2, \cdots, x_n$ と表記します。これを昇順（小さなものから大きなもの）に並べるものとします。

アルゴリズムA

1. カウンタ i を用意し， $i \leftarrow 1$ とする。
2. $x_i \sim x_n$ の中の最小値を示す添字を i_{min} とする（アルゴリズムBによる）。
3. i と i_{min} が異なる場合は，x_i と $x_{i_{min}}$ を交換する。
4. カウンタ i を1つ増やす。
5. $i < n$ ならば， 2. に戻って手順を繰り返す。
6. $i = n$ ならば終了する（ソートされたデータ群が求められた）。

最小の値を持つ添字を探す部分のアルゴリズムは，次のようなアルゴリズムにより実現できます。

アルゴリズムB

1. 求める添字 i_{min} を用意し， $i_{min} \leftarrow i$ とする。
2. カウンタ j を用意し，$j \leftarrow i+1$ とする。
3. $x_j < x_{i_{min}}$ ならば， $i_{min} \leftarrow j$ とする。

4. カウンタ j を 1 つ増やす。
5. $j < n$ ならば， 3. に戻って手順を繰り返す。
6. $j = n$ ならば終了する（i_{min} に最小の値を示すデータの添字が求まった）。

C 言語のプログラムに直すには，反復処理と条件判断，および配列変数の表現の違いに注意します。具体的なプログラムの作成は，次の練習課題としましょう。

8.2 練習問題（sample9.c）の改変

前節で解説した，単純選択法を用いてソートを行なうプログラムを組み立ててみます。

8.2.1 単純選択法のフローチャート表現

アルゴリズムをいきなりプログラムの形に直すのは，いささか辛いものがありますので，まず，図 8.2 にフローチャートとして整理しておきましょう。C 言語での表現を意識して，配列の添字の表現を多少変更してあります。

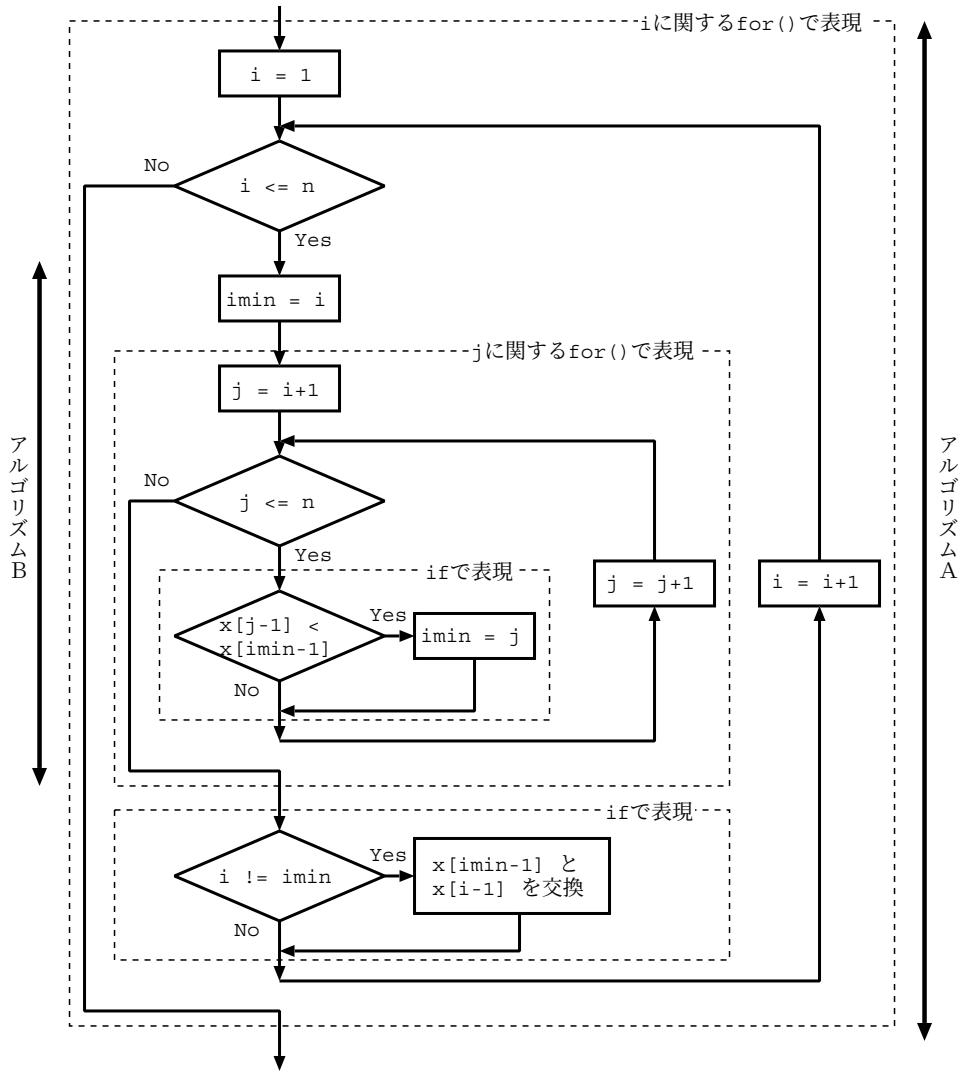


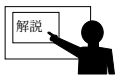
図 8.2: 単純選択法のフローチャート表現

8.2.2　単純選択法のプログラム表現

では，おおまかなプログラムの構造を下に示します。図 8.2 のフローチャートを参照して，空いている部分（□の部分）を補ってから，プログラムの改変にチャレンジしてみましょう。

```
for( i=1 ; □ ; □ ) {
  imin = i ;
  for( j = i+1 ; □ ; □ ) {
   if( □ )
     □ ;
  }
  if( □ ) {
    xdummy = x[i-1] ;
    x[i-1] = x[imin-1] ;
    x[imin-1] = xdummy ;
  }
}
```

8.3　プログラムの構造化とフローチャート



第１日のプログラムに比べると，大がかりなプログラムの作成を手掛けてみました。第１日に学習した条件判断に加えて，反復処理，配列変数による大量データの表現，アルゴリズムの重要性について学習したわけですが，どれが一番難しかったでしょうか？アルゴリズム表現というと，非日常的な言い回しを連想しがちですが，よくよく考えると私たち人間も無意識にアルゴリズムに沿った行動をしてはいないでしょうか？日常の行動を振り返ってみて，アルゴリズムやフローチャートで表現するというのも，よい訓練になることでしょう。

ここでちょっと休憩

第9章　大量データへの対応

9.1　データのファイル化

すでに，プログラムとデータの分離の方法や，大量データの表現に向いた配列変数については学習してきました。ここでは，それをさらに進める形で，データをファイルとして保管し，保管されたデータを自動的に読み取ったり，計算結果をファイルとして残すための方法 — ファイル入出力 — について学びます。

9.1.1　C 言語におけるファイル処理のポイント

ファイルに保管されているデータを読み取って処理するプログラムを，ftest1.c として準備してあります（81 ページを参照のこと）。

このプログラムのポイントは，fopen 関数，fscanf 関数，fclose 関数と，FILE **型変数**の使い方です。

FILE **型**の変数には，ファイルの実体をプログラム中で指定するための識別情報が格納されます。というのも，一般に，ひとつのプログラム中で複数のファイルを同時に扱う必要があるからです。このため，どのファイルからデータを読んで，その結果をどのファイルに格納するのかを厳密に指定するために使用されるわけです。

なお，宣言部分では， FILE *fp ; のように，変数名の前に*が余分についています。ここでは,「そのように宣言するものだ」とだけ覚えておきましょう。

実際にファイルからデータを読む場面では，scanf の代わりに，fscanf を用います。前述したように「どのファイルから」を明示する必要があります。たとえば,

```
int kosuu ;
fscanf("nyuuryokufairu", "%d", &kosuu) ; /* 間違い */
```

のように，先頭のパラメータとしてファイルの名前を直接指定することも考えられますが，数回に分けて読み取ったりする場合，ファイル名をいちいち指定するのも面倒ですし，長い名前のファイルの場合には，つづりを間違う危険もあります。また，特定のファイルだけでなく，ユーザの指定したファイルに対応できる必要もあるでしょう。そこで，C 言語に限らず，一般のプログラミング言語では，ファイル名を直接に指示するのではなく，番号などで区別するようにしてあります。つまり，正しくは，FILE 型として宣言された変数を先頭のパラメータに用いて，次のように表記します。

```
int kosuu ;
FILE *fp ;
fscanf(fp, "%d", &kosuu) ; /* （文法的には）正しい使い方 */
```

さて，文法的にはこれで正しいのですが，このままでは,「どのファイルから」は分かりません。毎回ファイル名を指示する代わりに，fopen を用いて，実際のファイル名を元にした識別情報を FILE 型変数に代入しておきます。

```
int kosuu ;
FILE *fp ;
fp = fopen("nyuuryokufairu", "r") ;
fscanf(fp, "%d", &kosuu) ; /* 正しい使い方 */
```

fopen の最初のパラメータが実際のファイル名です。２番目のパラメータは，モードと呼ばれるもので，このデータを読み取りに使うのか，書き出しに使うのかの区分を指示します[1]。かえって手間のようにも見えますが，fscanf を何度も使う場面では，こちらの方がすっきりとしますし，ファイル名のつづりミスを気にしなくてもいいわけです。

必要な情報を読み取ったら，ファイルの使用を終えたことを OS に知らせておきます[2]。これには，fclose を使用しますが，ここでも「どのファイル」を明示するために，FILE 型変数を使用します。したがって，一般的には，次のような使い方となります。

```
int kosuu ;
FILE *fp ;
fp = fopen("nyuuryokufairu", "r") ;
fscanf(fp, "%d", &kosuu) ;
... /* 必要な数だけ fscanf を使用 */
...
fclose(fp) ;
```

なお，ftest1.c では，fscanf() の値を 2 と比較してあります。これは，正しく読み取れたデータ数を返すという fscanf の性質を使って，ファイルに格納されているデータをすべて読み取ったかどうかの判定を行なうための手段です。これによって，データ数 n を自動的にカウントしているわけです。

以上の説明を元にプログラム全体の解読を行なってみてください。他にも，これまで説明していない事柄がいくつかありますので，ある程度予想を立てて自分なりの解釈を試みたり，文法編（第 III 部）を参考に解読にチャレンジしてみましょう。

---

[1] コンピュータのファイルは，原則として, すでに記録されている内容を先頭から順番に読み取るか，新たにファイルを作成して内容を追記するか，どちらかの処理が使用されます。一方向にだけデータが流れていくことから，このような方式を**ストリーム入出力**と呼びます（stream とは” 流れ” の意）。ファイルの内容の一部を書き換える方法もありますが，処理がやや繁雑なのでここでは扱いません。

[2] さもないと，別のプログラムがそのファイルを読み取ることができなくなります

9.2 データファイルの作成とプログラム例（ftest1.c）によるファイル処理

それでは，プログラム例によるデータファイルの処理を行なってみましょう。データファイルについては準備していませんので，データの準備から始めます。

9.2.1 データファイルの作成

まず，データファイルを用意しましょう。

データファイルは，ソースプログラムと同じ要領でも作成できますが，Windows の標準アクセサリである「メモ帳」を使った方が，より簡単でしょう。

80 ページの説明とデータ例を参照して作業します。

余力のある方は（ファイル名を変えて）複数のファイルを準備してみましょう。

9.2.2 プログラム例（ftest1.c）の実行と改変

最低限１つのデータファイルが準備できたら，例題プログラム ftest1.c を実行してみてください。上のステップで作成したデータファイルの名前を，たとえば，z:￥samples￥test.txt のようにフルパス名で入力すると，データの内容そのものは自動的にファイルから読み取って，データの一覧表が表示されるはずです。

余力のある人は，このプログラムに「得点」の「平均点」を求めたり,「最低点」「最高点」を求める文を追加してみてください。また，データファイルを複数準備できた方は，データを取り換えて実行してみて，それぞれ正しく処理されることを確認してみてください。

ここでちょっと休憩

第10章　プログラムの構造化とデータの構造化

10.1　プログラムの構造化（3） — 関数定義によるモジュール化 —

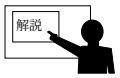
ファイル入出力とは直接には関係しませんが，プログラムが複雑になってくると，使用するデータや文，構文が入り混じってきて，プログラムのデバッグ作業や新たな機能の追加作業が困難になりがちです。このような場合には，機能的なまとまりを独立したモジュール[a]として再構成し，モジュールの呼出しを行なうように元のプログラムを改変することで効率化を図れます。

[a] C 言語の場合は関数，一般的には**サブルーチン**あるいは**プロシジャ**と呼ばれます。

10.1.1　プログラム例（ftest2.c）の実行

具体的な例として，プログラム例 ftest1.c と同じ内容を，モジュール化して表現したものを ftest2.c に示します。このプログラムでは,「データファイル名の設定」「データの読み込み」「データ一覧表の表示」といった，3つの機能ごとにモジュールを作成してみました。結果として，主プログラム（関数 main）の内容が簡潔になったことが分かります。

10.2　データの構造化（2） — 構造体 —



次に，データの構造化を考えてみましょう。すでに，配列変数という形で，ひとつの変数に複数の情報を混在させて記憶する表現方法については学んできました。しかし，配列変数では,「同種の」データしか記憶できません。プログラム例 ftest1.c や ftest2.c では，1人分のデータが，名前（文字列）と得点（実数データ）という異なる種類のデータから成っています。これらのプログラムのように，それぞれ別の属性を持つ変数として定義するというのもひとつの手ですが，個人のデータが複数の属性から成り立っているという表記の方が自然なことも多いものです。C 言語では，このような場合のデータ表現の方法として，struct 型という属性を利用できます。

10.2.1　プログラム例（ftest3.c）の実行

具体的な例として，プログラム例 ftest2.c と同じ内容を，構造体（struct 型で定義したデータのテンプレート）を用いて表現したものを ftest3.c に示します。

余力のある人は，構造体に,（たとえば，身長・体重・年齢・性別のような）別の属性を定義してみてください。データファイルにも手を加えて，これらの具体的なデータを入力できるようにも改変できますので挑戦してみましょう。

ここでちょっと休憩

第11章　総まとめとQ＆A

11.1　総まとめ

２ 日間に渡ったプログラミング体験はいかがだったでしょうか。冒頭にも述べてあるように，プログラミングのすべてを体験できるわけではありません。難解とか容易とかの判断は難しいかも知れませんが，この授業で目指していたことなどを整理して，今後の学習の指針について考えてみましょう。

11.1.1　授業のまとめ

この授業では，２つの大きな目的を掲げていました。

- 取っ掛かりとしてのプログラミングの体験をしてもらうこと
- ハードウェアとソフトウェアとの関わり合い方を理解してもらうこと

前者については，授業に（手と頭をフルに使って）参加した方はすべて当てはまることになります。

後者については，参加者それぞれで異なってくるかも知れません。別途配布する予定のアンケート風の自己診断表で確かめてみてください。また，授業を通じて，新しい発見や再発見がもしあったとすれば，それを文章として表現することで，より鮮明な形で自分の力にすることができると思います。

内容的には，C 言語という現在もっともポピュラーなプログラミング言語のひとつを使って，ソフトウェアの読み書きを実際に体験して頂いたことになります。C 言語を覚えること自体は，実はさほど重要なことではなく，プログラムというものの本質を理解するには，指示した手順に従ってキマジメに仕事をこなしてくれるコンピュータの特性の理解が必要だということに気がつくと思います。したがって，ソフトウェアの良否は，プログラムの表記方法だけではなく，巧妙に熟慮して作られた手順 — アルゴリズムに関わるのだということを理解して頂ければ，この授業での目的は完全に達成できたといって良いと思います。

11.1.2　発展的学習のために

付録ＡとＢに，発展的学習のための参考書の選び方についてまとめてあります。一読してみてください。特に重要なキーワードを挙げるならば，以下のようになるでしょう。

- オブジェクト
- データ構造
- アルゴリズム

次のステップを目指すならば，このようなキーワードを含む参考書を中心に例題の豊富なものから選ぶのが無難でしょう。

11.2　総合練習とＱ＆Ａ

残りの時間は，各自で 2 日間でつまづいたところを中心に再度練習問題をこなすなどの時間に充ててください。また，授業では，あえて避けた点や触れられなかった点も多々あります。この 2 日間の内容への質問はもちろんのこと，日頃疑問に思っていたことへの質問なども遠慮なくしてください。

第II部

プログラム例と演習問題

第12章　プログラム例

以下のプログラムは，私の本務校である横浜市立大学のホームページにあらかじめ登録してあります。インターネットエクスプローラを使って手元のパソコンにコピーした上で利用してください。

プログラムによっては，意図的にそのままでは動作しないようにしてあります。復習のために利用したり，補足的な説明のメモに役立ててください。

12.1　第１日で使用するプログラム

12.1.1　プログラム例（sample1.c）

```
 1 /* sample1.c: 一番簡単なプログラム  */
 2
 3 /* このプログラムのポイント          */
 4 /* 1:注釈                            */
 5 /* 2:’関数’定義                      */
 6 /* 3:データの’型’                    */
 7 /* 4:標準関数による表示（データ出力）*/
 8 /* 5:文字列の表現                    */
 9
10 /* ←この間は注釈とみなされる
11     （途中で改行されていてもよい）→ */
12
13 /* #include はプリプロセッサ指令と呼ばれるおまじない */
14 /* stdio.h はヘッダファイルと呼ばれるファイルの名前  */
15 #include <stdio.h>
16
17 /* ’関数’の定義
18    ここでは int 型（整数型）の値を持つ
19    main という名前の関数を定義している。
20    一般的には,
21
22    関数の型名 関数の名前 ( パラメータの並び )
23    {
24       この関数で行う手続きの表現
25    }
26 */
27
28 int main( )
29 {
30     /* 標準の出力関数 printf を用いて文字列を表示 */
31     /* "と"で括られた部分は’文字列’と呼ぶ。\n は
32        キーボードで入力できないような特殊文字（主に
33        制御文字）の表記法。’\n’は，改行コードを表す */
34     printf("This is a sample program.\n") ;
35
36     /* この関数の値を 0 にして関数の参照元（この場
37        合はオペレーティングシステム）に戻る          */
38     return 0 ;
39 }
```

12.1.2 プログラム例（sample2.c）

```
 1 /* sample2.c: 簡単なデータ表現と計算の表現 */
 2
 3 /* このプログラムのポイント             */
 4 /* 1: データの記憶場所（変数）の定義方法 */
 5 /* 2: 式による計算の表現                */
 6 /* 3: 文の並び順                        */
 7
 8 #include <stdio.h>
 9
10 int main()
11 {
12   /* 記憶場所の宣言 --- 変数と型定義 --- 常に関数の先頭で行なう
13      実数を扱える型 float の変数（データの記憶場所）の宣言 */
14   float teihen, takasa, menseki ;
15   /* 整数（小数部がないもの）を扱う型としては int を用いる */
16
17   /* 計算の表現 --- 代入と式 --- 表現している順に計算される */
18   /* 底辺（を表す teihen という変数）を 6 とする */
19   teihen = 6 ;
20   /* 高さを 4 とする */
21   takasa = 4 ;
22
23   /* 三角形の求積公式を使った計算 */
24   menseki = teihen * takasa / 2 ;
25
26   /* 結果の表示 --- データの出力 */
27   /* 出力用関数 printf を用いる
28      第１引数の文字列で ’書式’ を指示し
29      第２引数以降に対応するデータを指示する */
30   /* %g は、この部分に，float 型のデータの内容を 10 進数表記で
31      埋め込むように指示している。順序は、第２以降の引数の
32      指定の順序に従う。それ以外の文字は（sample1.c の場合と同様に）
33      そのまま画面に表示される。
34      データが整数（int 型）の場合は %d を使用する */
35   printf( "Menseki =  %g\n", menseki ) ;
36
37   return 0 ;
38 }
```

12.1.3　プログラム例（sample3.c）

```
 1 /* sample3.c: データ入力と条件判断を伴うプログラム */
 2
 3 #include <stdio.h>
 4
 5 /* このプログラムのポイント                  */
 6 /* 1: 標準入力関数 scanf によるデータ入力 */
 7 /* 2: if 〜 else 構文による判断処理        */
 8
 9 int main( )
10 {
11   float a,b,c ;
12
13   printf("三角形の形を推定します\n") ;
14
15   printf("三角形の第１辺の長さは？" ) ;
16   /* データのキーボードからの入力 --- 入力関数 scanf を使用 */
17   /* 入力用関数 scanf を用いる
18      第１引数の文字列で ’書式’ を指示し
19      第２引数以降に対応するデータの格納場所（変数のアドレス）を指示する */
20   scanf( "%g", &a ) ;
21   /* %g は，float 型用 int 型の場合は %d を用いる
22      &は，直後の変数のアドレスを求めるための特別な記号 */
23   /* 実際のデータは，プログラムが動作を始めてから，キーボードで入力する */
24
25   printf("三角形の第２辺の長さは？") ; scanf( "%g", &b ) ;
26   printf("三角形の第３辺の長さは？") ; scanf( "%g", &c ) ;
27
28   /* 入力された値の確認のため，そのまま表示してみる */
29   printf("a,b,c=%g,%g,%g\n",a,b,c) ;
30
31   /* if を使った単純な条件判断 */
32   if( a<=0 || b<=0 || c<=0 ||
33       a+b<=c || a+c<=b || b+c<=a ) {
34     printf("与えられた長さの辺では，三角形は作れません\n") ;
35     return 0 ; /* ここで関数の動作を止めて，オペレーティングシステムに戻る */
36   }
37
38   /* if の組み合わせによる複雑な判定 */
39   if( a==b && b==c ) {
40     printf("この三角形は正三角形です\n") ;
41   }
42   else if( a==b || b==c || c==a ) {
43     printf("この三角形は二等辺三角形です\n") ;
44   }
45   else {
46     printf("この三角形は不等辺三角形です\n") ;
47   }
48
```

```
49   return 0 ;
50 }
```

12.1.4　プログラム例（sample4.c）

```
 1 /* sample4.c: 疑似乱数を用いたじゃんけんゲーム */
 2
 3 /* このプログラムのポイント                                  */
 4 /* 1: 標準入力関数 scanf によるデータ入力                   */
 5 /* 2: if 〜 else 構文による判断処理                         */
 6 /* 3: 疑似乱数を利用するための srand() と rand() の使い方 */
 7
 8 #include <stdio.h>
 9 #include <stdlib.h> /* srand(),rand() を利用する時に必要 */
10 #include <time.h>   /* time() を利用するときに必要       */
11
12 int main( )
13 {
14     int your_hand, my_hand ;
15
16     printf("じゃんけんしましょ .......\n\n") ;
17
18     /* 乱数の初期値設定 */
19     srand( time(NULL) ) ; /* time() は現在時刻を秒単位で求める関数 */
20     /* 乱数を使って,  0,1,2 のいずれかの値を決める */
21     my_hand = rand() % 3 ; /* % は剰余 (あまり) を求める算術演算 */
22
23     printf("私の手は決まりました\n\n") ;
24
25     /* scanf を使って, 人間の手を入力 */
26     printf("あなたの手を数字で入れて下さい\n") ;
27     printf("  [0:ぐう 1:ちょき 2:ぱあ] -> ") ;
28     scanf("%d", &your_hand) ;
29
30     if( your_hand < 0 || your_hand >2 ) {
31       printf("いんちきはやめて！\n") ;
32       return 1 ; /* 強制的に OS に戻る (1 はエラー発生を意味するコード) */
33     }
34
35     if( my_hand == your_hand ) {
36       printf("\n 残念あいこでした！\n") ;
37     }
38     else if( my_hand == 0 ) {
39       if( your_hand == 1 ) printf("\n あんたの負け！\n") ;
40       else                 printf("\n あんたの勝ち！\n") ;
41     }
42     else if( my_hand == 1 ) {
43       if( your_hand == 2 ) printf("\n あんたの負け！\n") ;
44       else                 printf("\n あんたの勝ち！\n") ;
45     }
46     else {
47       if( your_hand == 0 ) printf("\n あんたの負け！\n") ;
48       else                 printf("\n あんたの勝ち！\n") ;
```

```
49      }
50
51      return 0 ;
52 }
```

12.1.5　練習問題（sample5.c）

```
 1 /* sample5.c: 練習問題 */
 2
 3 #include <stdio.h>
 4
 5 /* 一日目の総合練習です。ローレル指数と呼ばれる肥満度
 6    の指標を計算させるプログラムを作ります。
 7
 8    ポイント:
 9    1) 入出力 scanf, printf の使い方
10    2) 計算式の表現
11    3) データ宣言
12
13    *実行例 (このような表示になるように作ってみる)*
14
15     身長 (cm) を入力 -> 180
16     体重 (kg) を入力 -> 60
17     あなたの肥満度は 102.881 です
18 */
19
20 int main( )
21 {
22     /* 体重と身長を表すデータ (変数) の宣言 */
23
24     /* ローレル指数を表すデータの宣言 */
25
26
27     /* データの入力 (体重 (kg) と身長 (cm)) */
28
29
30     /* ローレル指数の計算
31        計算式
32          ローレル指数 = 体重× 10000000 ／ (身長×身長×身長)
33        を参考にする */
34
35
36     /* 計算結果の表示 */
37
38
39     return 0 ;
40 }
```

12.2 第２日で使用するプログラム

12.2.1 プログラム例（sample6.c）

```
 1 /* sample6.c: 単純変数を用いたデータの集計 */
 2
 3 /* このプログラムのポイント        */
 4 /* 1:単純変数を用いたデータの集計 */
 5
 6 #include <stdio.h>
 7
 8 int main( )
 9 {
10   /* 5つのデータに対応する変数 */
11   float data1, data2, data3, data4, data5 ;
12   float goukei ;
13
14   printf("5つの数値データの合計値を求めます\n") ;
15   printf(" 1 番目のデータ？") ; scanf("%g",&data1) ;
16   printf(" 2 番目のデータ？") ; scanf("%g",&data2) ;
17   printf(" 3 番目のデータ？") ; scanf("%g",&data3) ;
18   printf(" 4 番目のデータ？") ; scanf("%g",&data4) ;
19   printf(" 5 番目のデータ？") ; scanf("%g",&data5) ;
20
21   goukei = data1+data2+data3+data4+data5 ;
22   printf("合計は %g です\n",goukei) ;
23
24   return 0 ;
25 }
```

12.2.2　練習問題（sample7.c）

```
 1 /* sample7.c: 配列変数を用いたデータの集計 */
 2
 3 /* このプログラムのポイント        */
 4 /* 1:配列変数の定義方法            */
 5 /* 2:式中での配列変数の使い方      */
 6 /* 3:配列変数を用いたデータの集計 */
 7
 8 /* このままでは，配列変数を使っていることの利点がないので,
 9    テキストに従って，改変していく */
10
11 #include <stdio.h>
12
13 int main( )
14 {
15   /* 5 つのデータに対応する変数 */
16   float data[5] ; /* 宣言時の [ ] 内の数字は要素の数 */
17   float goukei ;
18
19   /* 配列名 [ 番号 ] の形で単純変数と同等に扱える */
20   printf("5 つの数値データの合計値を求めます\n") ;
21   printf(" 1 番目のデータ？") ; scanf("%g",&data[0]) ; /* 0 から始まる！ */
22   printf(" 2 番目のデータ？") ; scanf("%g",&data[1]) ;
23   printf(" 3 番目のデータ？") ; scanf("%g",&data[2]) ;
24   printf(" 4 番目のデータ？") ; scanf("%g",&data[3]) ;
25   printf(" 5 番目のデータ？") ; scanf("%g",&data[4]) ;
26
27   goukei = data[0]+data[1]+data[2]+data[3]+data[4] ;
28   printf("合計は %g です\n",goukei) ;
29
30   return 0 ;
31 }
```

12.2.3 プログラム例（sample8.c）

```
 1 /* sample8.c: 配列を用いたじゃんけんゲームの改良 */
 2
 3 /* このプログラムのポイント              */
 4 /* 1: 配列による勝敗判定の方法          */
 5 /* 2: 変数への初期値設定                */
 6 /* 3: switch 〜 case 構文による判断処理 */
 7
 8 #include <stdio.h>
 9 #include <stdlib.h>
10 #include <time.h>
11
12 int main( )
13 {
14   int your_hand, my_hand ;
15   /* 勝敗を決める配列を二次元の配列として表現する */
16   /* それぞれの手は 0,1,2 で表現されているので，3 x 3 のマトリクスを想定し，
17      勝敗を例えば -1:人間の負け，0:引き分け，1:人間の勝ち で表現しておく。
18         →コンピュータの手
19             0    1    2
20           +----+----+----+
21         0 |  0 |  1 | -1 | ↓
22           +----+----+----+ 人
23         1 | -1 |  0 |  1 | 間
24           +----+----+----+ の
25         2 |  1 | -1 |  0 | 手
26           +----+----+----+   */
27   int shouhai[3][3] = { {  0, 1,-1 },    /* [0][0] [0][1] [0][2] */
28                         { -1, 0, 1 },    /* [1][0] [1][1] [1][2] */
29                         {  1,-1, 0 } } ; /* [2][0] [2][1] [2][2] */
30   /* 一般に，型宣言をする時にのみ，初期値を設定できる。単純変数の場合は，
31        型名 変数名 = 初期値 ;
32      配列変数の場合は，初期値をブロック化する。２次元配列の場合，
33      並びの順序にも注意 */
34
35   srand( time(NULL) ) ;
36   printf("じゃんけんしましょ .......\n\n") ;
37   my_hand = rand() % 3 ;
38   printf("私の手は決まりました\n\n") ;
39   printf("あなたの手を数字で入れて下さい\n") ;
40   printf("  [0:ぐう 1:ちょき 2:ぱあ] -> ") ;
41   scanf("%d", &your_hand) ;
42   if( your_hand < 0 || your_hand >2 ) {
43     printf("いんちきはやめて！\n") ;
44     return 1 ;
45   }
46
47   /* switch による条件判断 */
48   switch( shouhai[your_hand][my_hand] ) {
```

```
49     case -1: printf("あんたの負け！\n") ;       break ;
50     case  0: printf("残念！あいこでした！\n") ; break ;
51     case  1: printf("あんたの勝ち！\n") ;       break ;
52     default: printf("内部エラー！\n") ;
53   }
54
55   return 0 ;
56 }
```

12.2.4 練習問題（sample9.c）

```
 1 /* sample9.c: 練習問題 */
 2
 3 /* このプログラムのポイント      */
 4 /* 1: 配列変数の用い方           */
 5 /* 2: 反復処理の表現             */
 6 /* 3: 代表的なアルゴリズムの表現 */
 7
 8 #include <stdio.h>
 9
10 int main( )
11 {
12     float x[5], xdummy ;
13     int   n, i, j, imin ;
14
15     n = 5 ;
16     /* データ入力 */
17     printf("%d 個のデータを入力してください\n", n) ;
18     for( i=1 ; i<=n ; i=i+1 ) {
19       printf("%d 番目のデータを入力してください -> ", i) ;
20       scanf("%g", &x[i-1]) ;
21     }
22
23     /* 入力された順に表示 */
24     printf("入力されたデータは") ;
25     for( i=1 ; i<=n ; i=i+1 ) printf(" %g",x[i-1]) ;
26     printf(" でした\n") ;
27
28     /* 昇順に並べ替え（単純選択法によるソート）
29        テキストの説明にしたがって，文を追加していく */
30
31
32     /* 結果を表示 */
33     printf("データを昇順に並べ替えると") ;
34     for( i=1 ; i<=n ; i=i+1 ) printf(" %g",x[i-1]) ;
35     printf(" となりました\n") ;
36
37     return 0 ;
38 }
```

12.2.5　ファイル入力の練習

まず，以下のデータファイルを作成しておいてからとりかかりましょう。

ファイル名の最後は,「.txt」となっていますので，注意して下さい。Windows の標準のテキスト形式となっているので，スタートボタンから「すべてのプログラム」→「アクセサリ」と辿ったところにある「メモ帳」を使うのが簡単です。

保存場所は教材のフォルダ（z:￥samples）と同じで構いません。

ポイントは，各行に名前と得点を一つ以上のスペースで区切ってあげることです。

12.2.6　データファイル（test.txt）

```
Asai   90
Mori   70
Kanai 100
Hori   80
```

なお,（当然と言えば当然なのですが）このファイルは「実行」できませんから，注意してください。

12.2.7 ファイル入力（ftest1.c）

指定されたファイルからデータを読みとって，データ数をカウントします。

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define NMAX 100
 5
 6 char   namae[NMAX][21] ;
 7 float  tokuten[NMAX] ;
 8 int    n ;
 9
10 int main( )
11 {
12   char   fairumei[100] ;
13   int    k ;
14   FILE   *fp ;
15
16   printf("Enter the name of datafile: ") ; scanf("%s",fairumei) ;
17
18   /* Open the file */
19   fp=fopen(fairumei,"r") ;
20   /* Error check */
21   if( fp == NULL ) {
22     printf("Can't open file %s\n",fairumei) ;
23     return 1 ;
24   }
25   /* Loop until encounter EOF */
26   n = 0 ;
27   while( n<NMAX &&
28          fscanf(fp,"%s %g",namae[n],&tokuten[n])==2 )
29     n = n+1 ;
30   /* Close the file */
31   fclose(fp) ;
32
33   printf("Number of data:%3d\n", n) ;
34   printf("%3s %-16s%8s\n", "No.", "Name", "Tokuten") ;
35   for( k=0 ; k<n ; k=k+1 )
36     printf("%3d %-16s%8.2f\n", k+1, namae[k], tokuten[k]) ;
37
38   return 0 ;
39 }
```

12.2.8　プログラムの構造化（ftest2.c）

main 関数以外の関数を自分で定義し，利用します。

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define NMAX 100
 5
 6 char  fairumei[100] ;
 7 char  namae[NMAX][21] ;
 8 float tokuten[NMAX] ;
 9 int   n ;
10
11 void ReadFileName( )
12 {
13   printf("Enter the name of datafile: ") ;  scanf("%s",fairumei) ;
14 }
15
16 int ReadFile( )
17 {
18   FILE  *fp ;
19
20   /* Open the file */
21   fp=fopen(fairumei,"r") ;
22   /* Error check */
23   if( fp == NULL ) {
24     printf("Can't open file %s\n",fairumei) ;
25     return 1 ; /* fail */
26   }
27   /* Loop until encounter EOF */
28   n = 0 ;
29   while( n<NMAX &&
30          fscanf(fp,"%s %g",namae[n],&tokuten[n])==2 )
31     n = n+1 ;
32   /* Close the file */
33   fclose(fp) ;
34   return 0 ; /* success */
35 }
36
37 void PrintList( )
38 {
39   int k ;
40   printf("Number of data:%3d\n", n) ;
41   printf("%3s %-16s%8s\n", "No.", "Name", "Tokuten") ;
42   for( k=0 ; k<n ; k=k+1 )
43     printf("%3d %-16s%8.2f\n", k+1, namae[k], tokuten[k]) ;
44 }
45
46 int main( )
47 {
```

```
48   /* get name of the datafile */
49   ReadFileName() ;
50   /* read data */
51   if( ReadFile() == 1 ) return 1 ; /* if fail, exit */
52   /* print list of data */
53   PrintList() ;
54
55   return 0 ;
56 }
```

12.2.9　データの構造化（ftest3.c）

個人ごとのデータを，構造化されたデータとして扱います。いわゆるデータベースで管理されるデータも，こうした構造化が行われています。

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 #define NMAX 100
 5
 6 struct Kojin {
 7   char  namae[21] ;
 8   float tokuten ;
 9 } ;
10
11 char   fairumei[100] ;
12 struct Kojin KojinData[NMAX] ;
13 int    n ;
14
15 void ReadFileName( )
16 {
17   printf("Enter the name of datafile: ") ;  scanf("%s",fairumei) ;
18 }
19
20 int ReadFile( )
21 {
22   FILE  *fp ;
23
24   /* Open the file */
25   fp=fopen(fairumei,"r") ;
26   /* Error check */
27   if( fp == NULL ) {
28     printf("Can't open file %s\n",fairumei) ;
29     return 1 ; /* fail */
30   }
31   /* Loop until encounter EOF */
32   n = 0 ;
33   while( n<NMAX &&
34          fscanf(fp,"%s %g",
35                 KojinData[n].namae,&KojinData[n].tokuten)==2 )
36     n = n+1 ;
37   /* Close the file */
38   fclose(fp) ;
39   return 0 ; /* success */
40 }
41
42 void PrintList( )
43 {
44   int k ;
45   printf("Number of data:%3d\n", n) ;
46   printf("%3s %-16s%8s\n", "No.", "Name", "Tokuten") ;
```

```
47   for( k=0 ; k<n ; k=k+1 )
48     printf("%3d %-16s%8.2f\n", k+1, KojinData[k].namae, KojinData[k].tokuten) ;
49 }
50
51 int main( )
52 {
53   /* get name of the datafile */
54   ReadFileName() ;
55   /* read data */
56   if( ReadFile() == 1 ) return 1 ; /* if fail, exit */
57   /* print list of data */
58   PrintList() ;
59
60   return 0 ;
61 }
```

第13章　演習問題

13.1　単純な計算プログラム

練習問題のプログラム `sample5.c` を参考にして，以下にあげるような問題を解くためのプログラムを自身で作成してみよ。

- 距離 (km) と所要時間 (分←注意！) から，平均時速 (km/h) を求めるプログラム
- 正 n 角形のひとつの内角の大きさを求めるプログラム
- 初速度 v_0 (m/秒) (≥ 0) で落下し始めた物体の t 秒後の速度 v (m/秒) と位置 y (m) を求めるプログラム (重力加速度は 9.8m/秒^2 とする)

13.2　条件判断を利用したプログラム

単純な一元一次方程式

$$ax + b = 0$$

を x について解くためのプログラムを作成せよ。プログラムの条件は，以下の通り。

- 係数 a と b を読み込んで解を表示するようにする。
- 除算できる条件を考慮すること。すなわち，
  1. $a = 0$ かつ $b = 0$ の時
  2. $a = 0$ かつ $b \neq 0$ の時
  3. $a \neq 0$ の時

  の 3 ケースについて正しく解を求められるようにする。

13.3　反復処理を伴うプログラム (1)

`for()` を二重に使うと，行列風の表示を行うことができる。最も単純な行列表 – 九九の表 – を表示するプログラムを作成してみよ。

13.4　反復処理を伴うプログラム (2)

じゃんけんゲーム（sample8.c）に手を加えて，勝負がつくまで反復する（あいこだったら，もう一度勝負する）ようなプログラムにしてみよ。for() ， while() いずれを用いてもよい（goto は使わないこと）。

まず，フローチャートなどを使って充分考えてから取り掛かること。

13.5　金種計算

総合的な問題として，別紙に用意した「演習問題–金種計算」を実際に行ってみよ。

第III部

C言語の文法（要約）

第14章　１日目に学習する文法

14.1　語句

プログラムを構成する文字の列で，それ以上には分割できない単位として扱われるもの。

14.1.1　名前

名前は，形式的には，アルファベット[1]と数字[2]の列で表記された記号である。
処理系によっても異なるが，最大の長さは 30 文字程度である。
名前は，予約語や変数名，関数名などに用いられる。

14.1.2　予約語

名前のうち，与えられた以外の意味に用いることができないもの。主な予約語を以下に列挙する（用途については省略）。

●主な予約語（この他にも存在する）

```
auto break case char continue default double else enum extern
float for if int long register short signed sizeof static struct
switch typedef union unsigned void while
```

14.1.3　文字列表記

１対の記号"で括ったものを，文字列と呼ぶ。文字列は，内部的には文字型（char 型）の配列データとして処理される。文字列の内部には，名前で用いられる英数字以外のすべての文字（日本語も含む）が利用できる。ただし，"それ自体などの特殊な文字や非印刷文字を表現するには，エスケープ記号（\，日本語キーボードでは ¥ ）との組合せで表現する必要がある。

●文字列の例

```
"abc"
"日本語"
```

[1] a〜z，A〜Z，例外的にアンダースコア_ も含まれる。大文字・小文字も区別されるので注意
[2] 0〜9，ただし最初の文字としては使えない

14.1.4　区切り記号

語句を区切るための文字で，空白文字，タブ文字（キーボードの Tab キー），改行文字（キーボードの Enter キー）がある。また，注釈（文字列を「/*」と「*/」で括ったもの）も 1 個の空白と同等に扱われる。

14.1.5　定数

数値定数として，以下のものを用いることができる。

- 整数（10 進数，8 進数，16 進数，リテラル表現）
- 実数

10 進数は，0〜9 の文字を用いて表現する。先行する 0 は省略する。

● 10 進数の例
```
0 123 51 -49
```

8 進数は，0〜7 の文字を用いて表現する。10 進数と区別するために，必ず 0 を先行させる。

● 8 進数の例
```
0 0135 077 -023
```

16 進数は，0〜9 および a〜f，A〜F の文字を用いて表現する。10 進数，8 進数と区別するために，0x または 0X を先行させる。

● 16 進数の例
```
0 0x12 0xAB 0X16 -0xa8
```

リテラル表現は，1 バイトの整数値を表す特別なもので，特に文字型と呼ばれることがある。単引用符 ’ で文字を括って表現する。値は，その文字の内部表現の値（通常は ASCII コードの値）となる。非印刷文字（制御用文字）を表すには，エスケープ記号 \ に続けて 3 桁の 8 進数を記入するか，\x に続けて 2 桁の 16 進数を記入する。または，表 14.1 のように定められた記号を用いる。

実数定数は，10 進数によって表記された整数部と小数部を小数点 . で連結したものである。また，その後に文字 e または E に続く指数（底は 10）を連結した科学表記を用いることもできる。

●実数値の例
```
0.0 3.1416 1.23e6 4.1e-3 -5.34
```

上の例で，3 番目の数値は 1230000.0 （$= 1.23 \times 10^6$），4 番目の数値は 0.0041 （$= 4.1 \times 10^{-3}$）と評価される。

表 14.1: リテラルによる数値表現

表現	意味	式の値（ASCII の場合）
'a'	文字 a の内部コード	97
'\006'	8 進数 006	6
'\x10'	16 進数 0x10	16
'\\'	文字 \ の内部コード	92
'\''	文字 ' の内部コード	39
'\a'	Alert 制御文字（警報コード）	7
'\b'	Back Space 制御文字（後退コード） Back Space キーに対応	8
'\t'	TAB 制御文字（タブコード） Tab キーに対応	9
'\n'	New Line 制御文字（改行コード）	10
'\f'	Form Feed 制御文字（改頁コード）	12
'\r'	Return 制御文字（復帰コード）	13

14.1.6 演算子

算術演算

いわゆる加減乗除を表現する（表 14.2）。

表 14.2: 算術演算用の記号

記号	演算内容	式の例	式の値
＋	加算	1+2	3
－	減算	5-1	4
＊	乗算	2*3	6
／	除算	5/3	1 （整数の場合は結果は切捨て）
%	剰余（あまり）	5%3	2 （整数どうしのみ）
－	負数	-2	-2

関係演算

関係演算は，数値の大小を判定する演算である。判定結果は，式が成立する時は 1 ， 式が成立しない時は 0 ，となる（表 14.3）。

表 14.3: 関係演算用の記号

記号	式の意味	式の例	式の値
==	両辺が等しい	5==3	0
!=	両辺が等しくない	5!=3	1
<	左辺が小さい	5<3	0
>	右辺が小さい	5>3	1
<=	左辺≦右辺	5<=3	0
>=	左辺≧右辺	5>=3	1

論理演算

論理演算は，論理否定（NOT）／論理積（AND）／論理和（OR）の 3 種類である（表 14.4）。通常は，関係演算式の論理的な結合に用いる。演算結果は，成立する時に 1，成立しない場合は 0 となる。

表 14.4: 論理演算用の記号

記号	式の意味	式の例	式の値
!	論理否定。（単項）式を否定	! （1==2）	1
&&	論理積。両辺が同時に成立。 左辺が成立しない時は右辺は評価されない	5<3 && 1<2	0
||	論理和。両辺のどちらかが成立。 左辺が成立する時は右辺は評価されない	1<2 || 2<3	1

その他の演算用記号

演算の順序を明示したり，優先順位の順序を変更するには，1 対の括弧 () を用いる。多重になるような場合でも，同じ記号を用いることに注意しよう。

●括弧を用いた演算式の例

```
（1+2）*3
（（4+6）/7）/8
```

C 言語では，3 項演算と呼ばれる演算や，データのビット（2 進数の各桁）ごとの論理演算を行なうビット演算もあるが，初心者には必要ないであろう。

この他，代入演算と呼ばれるものがあるが，これを理解するには，**変数**という概念が必要なので別に解説する。

14.2 変数とデータの型および代入演算

14.2.1 変数の表記方法

プログラムの中では，データは変数として表現される。変数の表記は，前述の名前のルールに従う。

●変数の例

```
a
b1
xyz
A
my_name
```

●正しくない例

`1st`	第 1 文字が数字
`if`	予約語（後述）と同じ綴り
`My Name`	使用できない文字（空白）を含んでいる

変数を用いることにより，抽象的な式表現ができる。

●変数による式表現の例

`kyori / jikan`	「速度」の公式
`（ joutei + katei ） * takasa / 2`	「台形の面積」の公式

このように，式の一部に変数を使用すると,「その時点での」変数の内容（数値）が実際の計算に使用される。このことを**参照**と呼ぶ。

14.2.2 データの型

変数を使用する際は，その属性をコンパイラに対して指示しておく必要がある。属性とは，主に，その変数が保持するデータの形式であり，**型**と呼ぶ。C 言語では，基本の型として，表 14.5 のようなものがあらかじめ用意されている。

ある変数の型を指定するには，型名の後に変数を添える。複数の変数に対して，同一の型を指定するには，カンマで区切ればよい。

●変数の型宣言の例

```
float takasa ;
short a, b, c ;
```

実際のコーディングでは，このように文の区切りとしてのセミコロン ; を添える必要がある。セミコロンは必ずしも同一行にある必要はなく，また同一行に複数の宣言を書いてもよい。

表 14.5: C 言語における基本の型

型	データの占める容量	型名	主な特徴・注意事項
整数	1 バイト	char	文字型とも呼ぶ
	2 バイト	short	short int も同じ
	4 バイト	long	long int も同じ
	2 または 4 バイト	int	コンパイラに依存
実数	4 バイト	float	有効数字の桁数はおおむね 6 桁
	8 バイト	double	有効数字の桁数はおおむね 15 桁

整数のグループでは，予約語 unsigend を各型名の前に付加することで，符号無しのデータを扱うことができる。(例 unsigned char=符号無しの 1 バイト型整数 (0〜255 を表現))

14.2.3　代入演算

変数の重要な特徴として，**代入**操作ができることが挙げられる。

これは，変数の内容を決定させる操作である。最も単純なものは，単一の等号記号= を用いて，次のように表現できる。

●単純代入の例

```
float takasa, teihen, menseki ; /* 変数の型宣言 */
takasa = 3 ;                    /* 変数 takasa に 3 を記憶させる */
teihen = 5 ;                    /* 変数 teihen に 5 を記憶させる */
menseki = takasa * teihen / 2 ; /* 変数 menseki に右辺の計算結果を記憶させる */
```

代入操作は，式そのものを記憶させているのではなく，実際の計算結果を記憶することに注意しよう。

また，上記の例のように，型宣言は先に行ない，各文はセミコロンで区切ることにも注意すること。

単純代入の他にも，C 言語固有の代入演算として，表 14.6 のようなものが用意されている。

表 14.6: C 言語における代入演算

記号	意味
=	右辺の結果を単純に代入
+=	右辺の結果を加えて代入
-=	右辺の結果を減じて代入
*=	右辺の結果を乗じて代入
/=	右辺の結果で除して代入
%=	右辺の結果による剰余を代入

14.3 文（その 1）

14.3.1 単純文

宣言や式にセミコロン; を添えたもの。

14.3.2 複合文

複数の文の前後を「{」と「}」で括ったもの。ひとつの文と同等に扱われる。

14.3.3 条件文

予約語 if を用いて，条件式（関係演算式やそれらを論理演算で結合したもの）によって，文の実行を制御できる。

●形式 1
```
if （ 条件式 ） 文
```

例） `if （ x < 0 ） x = -x ;`

●形式 2
```
if （ 条件式 ） 文 1 else 文 2
```

例） `if （ x < 0 ） y = -x ; else y = x ;`

図式で表現すると，図 14.1 のようになる。

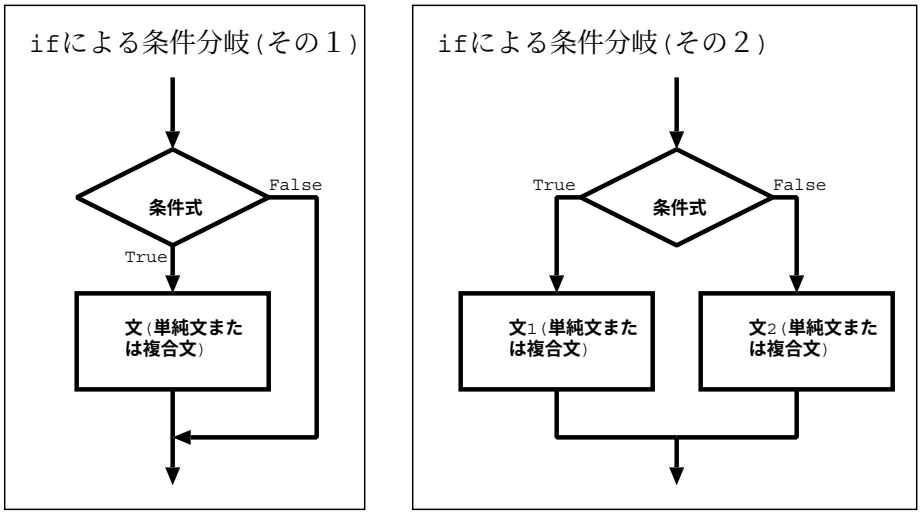


図 14.1: if による条件文の動作の流れ

14.4　入出力関数

14.4.1　出力関数 printf

printf は，標準ライブラリ関数の一つで，内部データ（定数，変数，式）の値を，ディスプレイ上に書式付けて表示させる。

●形式
```
printf（ 書式文字列 【, データの並び】 ） ;
```

関数は，関数名（ここでは printf) の直後に「(」と「)」で括られたパラメータを指定して表現する。パラメータは正式には**引数**（ひきすう）と呼ばれ，関数の処理に必要な情報を与えるものと解釈できる。

printf の第 1 引数は，**書式**を表現した文字列である。書式は，文字%と英字の組合せで表現され，データの種類に応じて使い分ける。書式以外の文字が文字列にある場合は，そのままディスプレイに表示される。

データは，定数，変数，式のいずれかである。必要に応じて，複数のデータをカンマ，で区切って指定する。

書式の主なものは，表 14.7 のとおりである。

表 14.7: 出力時の書式

書式	意味	対応するデータの型
%c	整数を文字に変換して出力	char
%hd	整数を 10 進数に変換して出力	short
%ld	整数を 10 進数に変換して出力	long
%d	整数を 10 進数に変換して出力	int
%f	実数を 10 進数に変換して出力	float
%g	実数を 10 進数に変換して出力	float
%lf	実数を 10 進数に変換して出力	double
%lg	実数を 10 進数に変換して出力	double
%s	文字列を出力	文字列（char の配列）

●使用例
```
int a ;
a = 3 ;
printf（"a = %d\n", a） ;
```

●表示結果
```
a = 3
```

14.4.2 入力関数 scanf

scanf は，標準ライブラリ関数の一つで，キーボードから与えられた文字を，内部データ（変数）の値に変換する。

●形式

```
scanf（ 書式文字列 【, 変数アドレスの並び】 ） ;
```

scanf の第１引数は，**書式**を表現した文字列である。書式は，文字%と英字の組合せで表現され，データの種類に応じて使い分ける。

第２引数以降には，変数のアドレス（記憶場所の位置）を指示する。変数名の直前に，アドレス演算子&を添えればよい。必要に応じて，複数のデータをカンマ , で区切って指定する。

書式の主なものは，表 14.8 のとおりである。

表 14.8: 入力時の書式

書式	意味	対応するデータの型
%c	１文字読みとって変数に設定	char
%hd	文字列を 10 進の整数とみなして変数に設定	short
%ld	文字列を 10 進の整数とみなして変数に設定	long
%d	文字列を 10 進の整数とみなして変数に設定	int
%f	文字列を 10 進の実数とみなして変数に設定	float
%g	文字列を 10 進の実数とみなして変数に設定	float
%lf	文字列を 10 進の実数とみなして変数に設定	double
%lg	文字列を 10 進の実数とみなして変数に設定	double

●使用例

```
int a ;
scanf（"%d", &a） ;
```

実行後の a の値は，キーボードから与えた文字列による。

```
キーボードから与えた文字　a の値
123[Enter]                123
54.3[Enter]                54 （.3 は整数でないので無視される）
```

第15章　２日目に学習する文法

15.1　文(その2)

単純な実行処理 (直線的な実行) ばかりでなく，反復や選択を表現するための構文です。

15.1.1　goto

キーワード goto に続けて，場所を表す**ラベル**名を記述する。（if などと組み合わせて）実行の流れをラベル名の直後に移すために用いる。ラベル名は，変数名と同様のルールに従った英数字で表し，定義部ではコロン: （セミコロン; と区別すること）を添える。プログラム全体の見通しが悪くなるので多用するべきではない[1]。

●形式

```
goto ラベル名
...
ラベル名:      同じ関数内の任意の場所（分岐後の文の直前）に記述する
```

●例

```
   ...
hajime:            ラベル定義
   ...
   ...
   goto hajime ;   先頭に戻る
```

15.1.2　do 〜 while

REPEAT–UNTIL 型の反復処理を表現する。

●形式

```
do 文 while ( 式 )
```

do 〜 while の構文を一般的にフローチャートで示すと，図 15.1 のようになる。

[1]事実,（考え抜かれた）良いプログラムであれば，ほとんどの場合 goto を必要としない

●例
```
float a ;
...
do {
  scanf("%g",&a) ;
} while( a<0 ) ;       a に非負の値が入力されるまで繰り返す
```

（単純な）while に比べると，文が少なくとも 1 回実行されたあとで継続条件は評価される点が異なる。

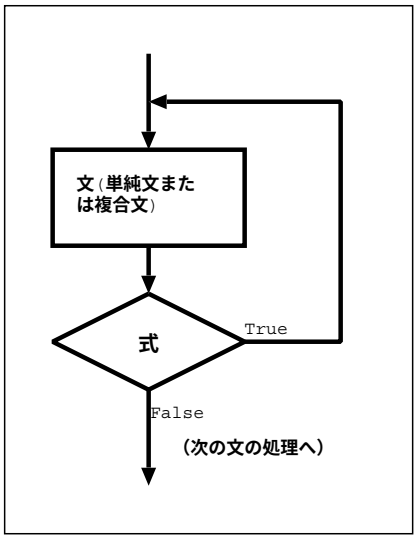


図 15.1: do 〜 while 構文のフローチャート表現

15.1.3 while

DO–WHILE 型の反復処理を表現する。

●形式
```
while ( 式 ) 文
```

for に比べると，括弧内には（省略不可能な）継続条件を示す式のみを記述する点が異なる。while の構文を一般的にフローチャートで示すと，図 15.2 のようになる。

●例
```
float a ;
...
while( 1 ) {
  scanf("%g",&a) ;
}
```

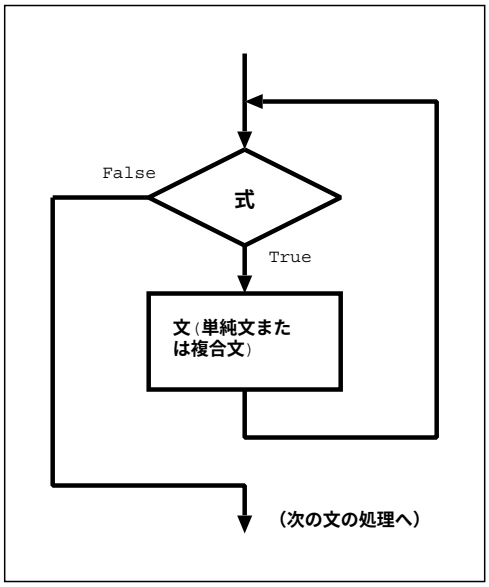


図 15.2: while 構文のフローチャート表現

15.1.4 for

一般の反復処理を表現する。

●形式

```
for ( 式1 ; 式2 ; 式3 ) 文
```

for の直後に一対の括弧 ( ) を添え，2つのセミコロン; によって，3つの部分に分ける。それぞれの内容は，次のとおり。

- 式 1 は，反復の開始時に 1 度だけ実行される内容で,「初期化」とも呼ばれる。
- 式 2 には，反復の条件を表現する。「継続条件」とも呼ばれる。
- 式 3 には，2度目以降の反復処理に先立って実行される内容で,「再初期化」とも呼ばれる。

一般的には，適当な変数を制御用に用いて，ある一定の数だけを反復させるように表現する。例えば，1 から 10 までの整数を表示させるには，次のように表現する。

●使用例

```
int x ;
…
for ( x=1 ; x<=10 ; x+=1 ) printf("X=%d\n",x) ;
```

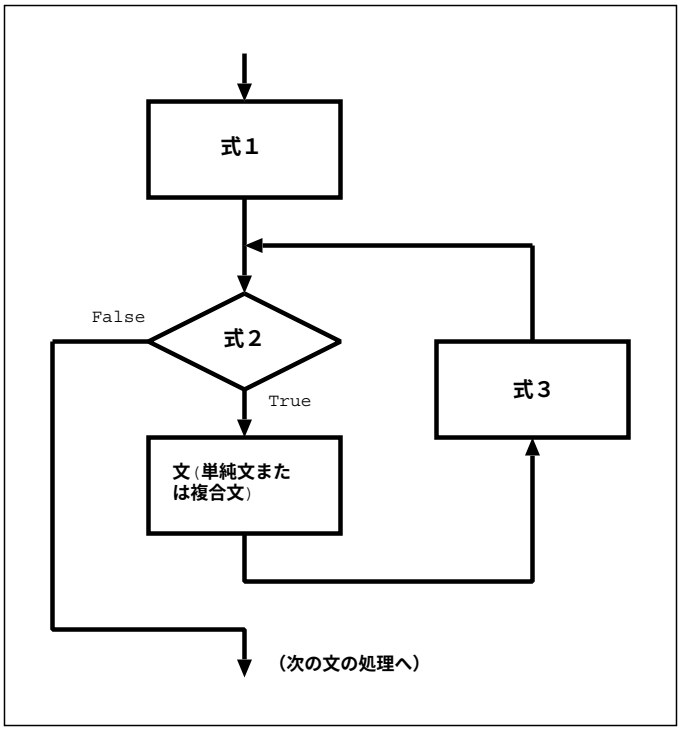


図 15.3: for() 構文のフローチャート表現

15.1.5 break

反復処理の中で break キーワードを用いると，最も近い反復処理が中断される。for・while・do 〜 while いずれにも使用できる[2]。

15.1.6 switch〜case

整数式の結果に応じて，３パターン以上の分岐処理を表現する。

●形式

```
switch ( 整数式 ) {
  case 定数 1 : 【 文の並び 1… 】【break;】
  case 定数 2 : 【 文の並び 2… 】【break;】
       …
  case 定数 n : 【 文の並び n… 】【break;】
   【default:   【 文の並び … 】】
 }
```

文の並びの最後が break 文でない場合には，そのまま直下の文の実行が行なわれることになるので注意すること。フローチャートで表現すると，図 15.4 のようになる。

[2] というよりは，中断命令は break １種類しかない

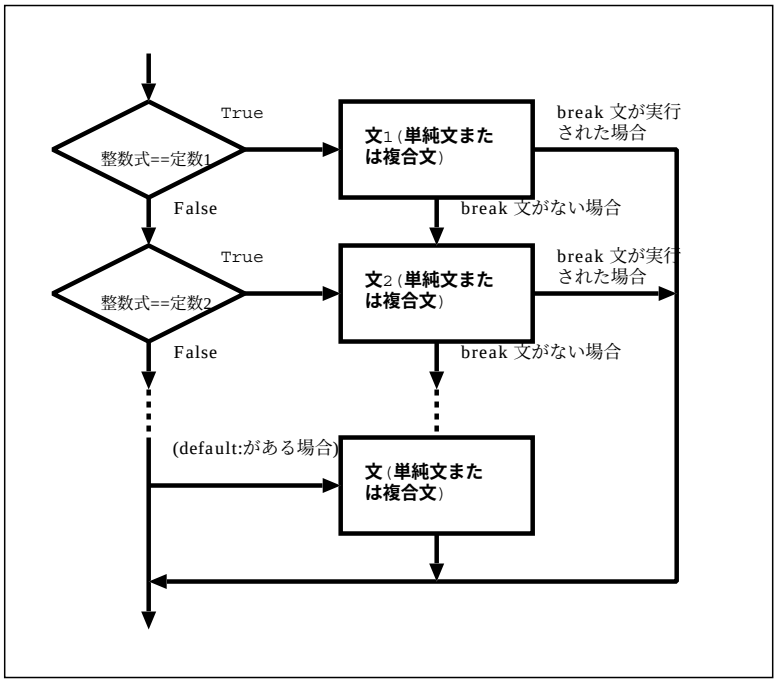


図 15.4: switch〜case 構文のフローチャート表現

15.2 配列型データ

配列は，数学でいうところの「ベクトル」や「行列」,「数列」を表現するための記法である。

配列には通常の変数と同様の名前を付けて用いる。ただし，この名前に [ ] を添えた形式で，その要素番号（先頭を 0 とする順序番号）を定数あるいは整数式で表現する。

●使用例（参照の場合）

```
f[n+2] = f[n]+f[n+1] ;
```

配列の要素の数は無制限には使用できないので，宣言時にその寸法を指定する。この寸法は定数でなければならない[3]。

●宣言の形式

基底の型（要素の型） 配列名 [要素の数（寸法）] ;

二次元以上の配列を使用するには， [ ] をその次元数だけ添えればよい。

●二次元配列の宣言例

```
float MatrixA[3][5] ;  /* 3x5 の行列 */
```

[3]実行時に大きさを変えることも可能であるが，動的割当て（*Dynamic Allocation*）という手法を用いる必要がある

15.3　標準ライブラリ関数

汎用的に利用される機能は,「ライブラリ」という形で提供される。具体的には，利用者は機能を実現する関数の使用方法を理解すればよい（その内容にまで立ち入る必要はない。あるいは自分でわざわざ組み立てなくてもよい)。

15.3.1　ヘッダファイル

標準のライブラリ関数を使用するには，`#include` を用いてヘッダファイルを指示しておく。ヘッダファイルは，機種やシステムに依存する部分を吸収する形で書かれた，主としてマクロの定義や関数のプロトタイプ宣言を含んだファイルである。その内容は，基本的には C 言語の文法に従ったものである。

主なものを 表 15.1 に示す。

表 15.1: 主な標準ヘッダファイル

名前	内容
`stdio.h`	入出力関数が含まれる
`string.h`	文字列処理関数。
`ctype.h`	文字処理関数 (大文字化・小文字化といったもの)
`math.h`	数学関数。三角関数などが含まれる
`stdlib.h`	汎用データ処理関数。ソート・サーチなども含まれる
`time.h`	時刻・日付・時間関数。

15.3.2　`stdio.h` に含まれる主な関数

`stdio.h` は，最も頻繁に用いられる。なぜならば，通常のプログラムでは，何らかの形で（コンピュータの）外部とデータのやりとりが発生するからである。

15.3.3　`string.h` に含まれる主な関数

文字列は `char` 型の配列であるため，データ全体の照合などは，その要素ごとに行なうのが原則である。しかし，これをプログラマが毎回作成するのは無駄が多いので，汎用関数として準備されている。

表 15.2: stdio.h に含まれる主な関数

関数名と形式	内容
printf(fmt,data...)	書式付き出力
scanf(fmt,data...)	書式付き入力
fopen(name,mode)	ファイルのオープン
fclose(file)	指定したファイルのクローズ
fprintf(file,fmt,data...)	ファイルへの書式付き出力
fscanf(file,fmt,data...)	ファイルからの書式付き入力

fmt，name，mode は文字列 (char の配列)。data は，任意のデータ (書式の内容に依存する)。file と fopen() の戻り値は FILE のポインタ型。

表 15.3: string.h に含まれる主な関数

関数名と形式	内容
strcat(s,t)	文字列 s のうしろに文字列 t を連結
strcpy(s,t)	文字列 s の領域へ文字列 t を複写
strchr(s,c)	文字列 s の中で最初に現れる文字 c の位置 (ないときは NULL)
strcmp(s,t)	文字列 s と t の大小関係 (一致は 0)
strlen(s)	文字列 s の長さ (終端符\0 直前までのバイト数)

s，t はいずれも 文字列（char の配列またはポインタ型)。
c は int 型。strcat(),strcpy(),strchr() の戻り値は文字列（char のポインタ型） 。
strcmp(),strlen() の戻り値は int 型。

15.3.4 ctype.h に含まれる主な関数

C 言語はシステムプログラミングにも利用されるので，非常にプリミティブなデータ—すなわち文字単位での処理を行なうことが多い。その基本となる関数は標準ライブラリとして提供されている。

15.3.5 math.h に含まれる主な関数

C 言語は，応用的ソフトウェア — 特に数値計算など — にも用いられるので，基本的な算術関数も標準ライブラリとして準備されている。

表 15.4: ctype.h に含まれる主な関数

関数名と形式	内容
isalpha(c)	c が英字 (a-zA-Z) かどうか
isupper(c)	c が大文字かどうか
islower(c)	c が小文字かどうか
isdigit(c)	c が数字かどうか
isalnum(c)	c が英字または数字かどうか
isspace(c)	c が空白類かどうか
toupper(c)	c の大文字
tolower(c)	c の小文字

c はいずれも int 型。関数の戻り値も int 型。

表 15.5: math.h に含まれる主な関数

関数名と形式	内容
sin(x)	x の正弦 (単位はラジアン)
cos(x)	x の余弦 (単位はラジアン)
tan(x)	x の正接 (単位はラジアン)
exp(x)	指数関数 e^x
log(x)	自然対数
log10(x)	常用対数
sqrt(x)	x の平方根
pow(x,y)	冪乗 x^y
fabs(x)	x の絶対値
ceil(x)	x より小さくない最小の整数
floor(x)	x より大きくない最大の整数

x はいずれも double 型。関数の戻り値も double 型。

15.4 ファイル入出力

プログラムで大量のデータ処理を行なうような場合には，実行のたびにキーボードからデータを入力する方法は効率的でない。入力ミスが発生しやすく，実行条件を変えての再試行が困難となるからである。このような場合には，データの部分だけをファイルに納めておくとよい。

また，プログラムで処理した結果をそのままワープロの原稿として利用したい場合がある。ディスプレイに表示されている処理結果や，印刷したものから手入力すると入力ミスも犯しやすく，大量の結果が出てくる場合には作業自体が煩雑になるからである。

プログラミング言語ではこのような処理をサポートするための機能が備わっているのが普通である。C 言語では，ファイル入出力関数を用いる。

15.4.1 FILE 型変数

プログラムでは，ファイル型（FILE）の変数[4]を用いて表現する。ファイル型変数の役割は，複数のファイルをプログラム中で区別（識別）することと，それぞれのファイルの処理状態（どこまで読み込んだかなど）やファイルの情報（ファイル名など）を管理する内部データとの結びつけを行なうことである。

15.4.2 fopen 関数によるファイルの指定

次にファイル型変数に fopen 関数を用いて値を設定する。fopen の第一引数にはファイル名を表す文字列（変数でもよい）を与える。第二引数は，そのファイルの使い方を指示する文字列（変数でもよい）を与える。指示できる内容は，表 15.6 の通りである。

fopen 関数の処理が成功すると，ファイル型変数には NULL 以外の値[5]が設定される。NULL は，標準ヘッダファイル stdio.h で定義されるマクロ記号で，通常その値は 0 となっている。もし，NULL になった場合には，何らかのエラーが生じていることを意味するので，if 構文を使って必ずチェックしておく。

15.4.3 データの入出力

実際のデータの入出力は，scanf のファイル対応版である fscanf や printf のファイル対応版である fprintf を用いる。第 1 引数にファイル変数を指示する以外は，使い方はほぼ同じである。入出力の関数は，必要なだけ繰り返して使用すればよい。

[4]正確には FILE 型のポインタ変数
[5]正確にはファイル管理用の内部データのアドレス

表 15.6: fopen 関数で使用するファイルモード

文字列 (ファイルモード)	意味
r	入力。ファイルの先頭から順番に読み込む。
w	出力。ファイルの先頭から順番に書き出す。
a	追加出力。ファイルの最後に追加して出力する。
b	データ内容がバイナリ型であることを明示する。rb のように組み合わせて使用する。
t	データ内容がテキスト型であることを明示する。rt のように組み合わせて使用する。

b と t は UNIX の処理系では意味がない（指示しても無視される）。両者の違いは，MS-DOS（Windows も含む）での行末を表すコード（'\r' と '\n'）や Macintosh での行末コード（'\r'）を，UNIX でのコード（'\n'）に置き換えるかどうかという点である。b も t 指示しない場合（デフォルト）は，テキストモードである。

15.4.4　fclose 関数による後始末

必要な処理が終ったら fclose 関数を用いて，後始末を行なっておく。入力用の場合はほとんど効果がないが，出力ファイルは，メモリに残っている内容を実際にディスクに書き出して，ディレクトリ情報を更新するという重要な処理が行なわれるので，確実に使用すること[6]。

15.5　構造型データ

（一般の）配列変数では，各要素の型が一定である。例えば，元素の周期律表をプログラム内で表現する時には，記号は文字列型，原子番号は整数型，原子量は実数型で表現するのが自然であるので，例えば，次のように表現できる。

```
char  atom_symbol[20][3] ; /* 原子記号  2 文字の文字列が 20 個分 */
short atom_number[20] ;    /* 原子番号 */
float atom_weight[20] ;    /* 原子量   */
```

むろん，これでも不都合はないが,「原子記号」「原子番号」「原子量」は「原子」の属性であるということを積極的に表現するために C 言語には struct キーワードが用意されており，データの**構造化**を表現することができる。

一般的な使い方は，次の通りである。

●形式

```
struct 【構造体名】 { /* 構造体名は省略してもよい */
```

[6]万一忘れても多くの処理系ではプログラム自体が終了する際に自動的にファイルもクローズされる（が，例外もあるので必ず使っておくこと）

```
        メンバの定義 （名前とそのデータタイプの宣言）
        ・
        ・
    } ;
```

例えば,「原子」を構造化データで表現すれば,

```
 struct atom {
    char  symbol[3] ;
    short number ;
    float weight ;
} ;
```

とでもなるだろう。こうして定義された構造体は，ひとつのデータタイプとして登録され，基本データタイプである int や float などと同様に，データの型定義に利用できる。ゆえに 20 個分のデータは,

```
 struct atom atoms[20] ;
```

のように定義できる（このように配列にもなる)。このように，常に struct atom をワンセットで扱う必要があるので，少々煩わしい。これを避けるには，#define による置き換え，あるいは typedef による置き換えを用いる[7]。

●形式

typedef 元の型名 新しい型名 ;

●使用例

```
typedef struct atom ATOM ;
ATOM atoms[20] ;
```

こうして定義されたデータは，単純な代入のみが許される（例えば，atoms[0] = atoms[1] ; は正しい)。通常の演算はできないが，その構成要素（メンバ）を指定することで可能である。

メンバの指定は，ピリオド (.) を用いる。例えば，atoms[0] の「原子量」は，atoms[0].weight と表現できる。

[7] #define はプリプロセッサ指令のひとつで，記号の置換を行なう。一般的には，円周率πの値を #define PI 3.1416 のように記号として定義するのに利用する。
typedef は，ユーザの型定義にのみ使用できる予約語である

付 録A　初心者のための参考書

「C 言語」に関する文献は，コンピュータ専門コーナーを備えている程度の書店であれば，選択に困るほど豊富に存在します。

文法的な話を比較的平易に解説しているものとしては，参考文献 [10] がおすすめできます。

また，多くの場合中級〜上級者向けとされていますが，プログラミングそのものの話を（いわば哲学的に）丁寧に解説しているものとしては，参考文献 [3] [6] があげられます。ただし，やや古いので現状で入手は困難かも知れません。

「味気ない文法というのはどうも苦手」な方は，具体的な問題に即し，その解決手段としてのプログラム作成というスタイルのものがよいでしょう。一例として，参考文献 [5] [11] [12] [13] などがあげられるでしょう。

コンピュータそのものの発展経過や動作原理を平易に解説したものとしては，参考文献 [7] がおすすめです。

付 録B　発展的学習のための参考書

書法

プログラミングそのものが理解できるようになり，さらに上位クラスの勉強を望む人には，まず，プログラムの「書き方」に関する参考文献 [4] や C 言語の設計にも携わった Kernighan の最近の著書 [2] がおすすめです。

論理

次の段階としては，やはりプログラムの論理的構成を意識しながらの演習ということになるでしょう。

参考文献 [1] [8] はいずれも UNIX オペレーティングシステムを前提に書かれた本ですが，丁寧かつ深いレベルでの議論－マシンやオペレーティングシステムそのもののを中心とした解説－がなされているので，プロフェッショナルになろうという方には必読といってもいい位のレベルの本と言えます。

オブジェクト指向

私自身はそこまで到達していない（あるいはそこまでは不要なので手をつけていない）のですが，現在のプログラミングは「オブジェクト指向」が主体になっています。

C 言語を発展させて，このようなオブジェクト指向に対応させた言語として「C++」（C のインクリメント－「しーぷらぷら」などと読む）があります。詳細な話については，参考文献 [9] を参照してください。

数値計算以外の話題

また，この授業では扱えなかった話題として，文字列処理，グラフィクス表現，ネットワーク処理，データベースとの連携などがあげられます。C 言語でも，追加的な開発環境（画像処理ライブラリなど）を準備することで十分対応できますが，OS の違い（プラットホームの差）が十分意識できている必要もあります。

Web プログラミング

ここ 10 年程で急速に普及したインターネットを前提とした環境では，単独のマシンによる処理はもちろん，サーバーとクライアントとの連携による処理も一般化してきました。これらの時代背景を受けて，いわゆる Web プログラミング（HTML，XML などの文書構造，CGI などによるリモート操作技術，サーバー特化型あるいはクライアント特化型のプログラミング等々）も重要な技術となっています。

こうした要求に対応する言語のひとつとして，Java も C の次に学ぶべき言語の候補として挙げられるでしょう。

関連図書

[1] Andrew Binstock, JohnRex：岩谷宏・訳. Ｃ言語で書くアルゴリズム. ソフトバンク, 1996.

[2] Brian W. Kernighan, RobPike：福崎俊博・訳. プログラミング作法. アスキー, 2000.

[3] Brian W. Kernighan, P.J.Plauger：木村泉・訳. ソフトウェア作法 (さくほう). 共立出版, 1981.

[4] Brian W. Kernighan, P.J.Plauger：木村泉・訳. プログラム書法［第２版］. 共立出版, 1982.

[5] 大森清美. 新編 魔方陣. 冨山房, 1992.

[6] 黒川利明. 作品としてのプログラム. 岩波書店, 1990.

[7] 坂村健. 痛快！コンピュータ学. 集英社文庫. 集英社, 2002.

[8] リチャード・スティーヴンス：大木敦雄・訳. 詳解ＵＮＩＸプログラミング. ソフトバンク, 1994.

[9] Ｂ．ストラウストラップ：齋藤信男他・訳. プログラミング言語Ｃ＋＋ 第２版. トッパン, 1993.

[10] 戸川隼人. ザ・Ｃ［第２版］. サイエンス社, 1997.

[11] 平野広美. Ｃでつくるニューラルネットワーク. パーソナルメディア, 1991.

[12] 三矢直城・田中一男. Ｃ言語による実用ファジィブック. ラッセル社, 1989.

[13] Ｌ．ラーダ・Ｄ．ネルソン：山下倫範・訳. パソコンで解く数学パズル. 海文堂, 1992.

索 引