

勉強すれば誰にでも分かる
C言語のプログラミング入門

逆瀬川浩孝

2017 年 9 月

序文

このテキストは経営システム工学科「情報処理基礎演習」で使用する C 言語プログラミングのための解説書兼自習書です。C 言語に初めて接する人、初めて同然の人にも分かるように、実際の計算でよく使われる基本的な文法を分かりやすく解説するとともに、初歩的なアルゴリズム（算法）のいくつかを紹介してあります。

Excel や統計パッケージ、シミュレータなどを使えば、プログラミングをしなくても結果が得られる場合が少なくありませんが、それですべてが解決するわけではありません。少なからぬ学生が卒業研究の時に自分のプログラムを作って計算していることが、その証拠です。そうなったときにあわてないように、基本をしっかり身につけてください。

このテキストは、「文法」と「算法」の 2 本立てになっています。C 言語は「言語」ですから「文法」があるだろうということはあらためて説明しなくても理解できると思いますが、「算法」というのは始めて聞く人が多いかもしれません。算法はアルゴリズムの日本語訳で、問題を解く計算の手順（計「算」作「法」）のことです。文法（「文」章作「法」）を知らなければプログラムが書けないのと同じように、算法を知らなければ問題を解くプログラムを書くことは出来ません。「問題」といってもその種類は無限にあり、そのすべてに対応することは出来ませんが、基礎的ないくつかの考え方を知っておくことで、未知の問題解決へのとっかかりを自分で見つける力が養成されるはずです。

本文にざっと目を通してみれば分かるように、文法に関しては「実習」「解説」「練習問題」の繰り返しになっています。まずは「実習」のプログラムを自分で入力し、ちゃんと動くことを確かめた上で、解説を読み、プログラムの構成と出力結果がどのように結びついているのかを理解してください。「練習問題」は、基本的な事項が理解できていれば、十分に自力で解けるようなものがほとんどでしょう。「練習問題」の解答はあえて付けてありません。実習のプログラムを参考にしてトライしてみてください。算法については、それまでの文法の知識を使って解けるような問題を取り上げ、計算手順を詳しく書いてありますので、プログラムを省略しているものもあります。説明にしたがってプログラムを作りながら読み進めていってください。ところどころに、「息抜きのページ」として、遊びのプログラムも挿入されていますので、試してみてください。

新しいプログラミング言語を習得するのに良い方法の一つは、良いプログラムをたくさ

ん覚えてまねすることです。「実習」のプログラムは筆者の経験上、もっとも分かりやすいと思われるスタイルで書かれています。これがベストというつもりはありませんが、最初のうちは、そのプログラムを丸暗記するつもりで覚えてください。それをもとに、いろいろな練習問題を解いていく間に自分の思考パターンに合った固有のプログラミングスタイルが身についていくことでしょう。

ややこしいことは後回しで、とにかくやってみよう、というスタイルで書かれている箇所も多いので、疑問に思うところもあるかもしれません。分からないところや説明が先送りになっている箇所があったら、先ずこのテキストの**索引を調べてください**。先の方に答えが書いてあるかもしれません。巻末の付録には「良くある質問」をまとめてあります。そこで答えが見つかるかもしれません。あるいは、プログラミングが得意そうな友達に聞いてごらん下さい。あるいは、ネットで調べてご覧下さい。分からないことを文章にして検索すると結構いろいろな記事が見つかります。でも、教えてもらう、あるいは教え合うのは、考え方とか、文法、アルゴリズムなどで、プログラムそのものをコピーペーストしてはいけません。プログラムを書くのは自分です。考えても分からないことがあったら質問のメールをください。

記述には間違いがないように、十分注意しているつもりですが、勘違いや、見落としがあるかもしれません。間違いを見つけたらなるべく早く教えてください。また、訂正はなるべく早くテキストのサポートページ

<http://www.f.waseda.jp/sakas/cProgramming/>

に載せるつもりです。

このテキストは、何も知らない人がプログラミング能力を効率的に身につけるには、どのような順番でどのようなプログラムを書くことがよいのか、著者なりに最善と思われる記述を心がけているつもりですが、思わぬ反応が出てきて教えられることが多々ありました。この本は、今までに受講した多くの学生のそのようなコメントによって改良を重ねてきました。すべての受講生に感謝します。また、1章のソフトウェアの使い方については経営システム工学科後藤正幸氏のアイディアを参考にさせていただきました。記して感謝します。

「学問に王道無し」をもじっていえば「プログラミングに王道無し」です。地道に努力し、なるべくたくさんのプログラムを「自分で」書いてください。ちゃんと勉強すればそれなりのことはあるはずです。成功を祈ります。

2017.9.15

逆瀬川浩孝

目次

第 1 章	初めての C	1
1.1	C 言語とは	1
1.2	学習の仕方	2
1.3	Hello C world.	4
1.4	プロジェクトの追加	9
1.5	レポート作成	11
第 2 章	データの入出力と簡単な計算	15
2.1	基本入出力と計算	15
2.2	プログラミング練習	23
2.3	数学関数	27
2.4	プログラムのデバッグとチェックリスト	29
2.5	章末演習問題	32
第 3 章	条件分岐とフィードバック	33
3.1	条件付き分岐 (if ... else)	33
3.2	フィードバック構造 (while, do ... while)	40
3.3	擬似乱数 (rand())	47
3.4	章末演習問題	51
第 4 章	算法 1: 数論	53
4.1	アルゴリズム (算法)	53
4.2	約数	56
4.3	素数	57
4.4	素因数分解	59
4.5	最大公約数、ユークリッド互除法	61
4.6	拡張ユークリッド互除法	63
4.7	漸化式	67

4.8	章末演習問題	72
第 5 章	繰り返しと配列変数	75
5.1	繰り返し構造 (for)	75
5.2	配列変数	80
5.3	多重の for、複雑な繰り返し構造	87
5.4	章末演習問題	91
第 6 章	算法 2：統計計算	93
6.1	データの入力	93
6.2	標本平均、標本分散の計算	95
6.3	最大値、順位の計算	97
6.4	度数分布	99
6.5	数の表現と計算の正しさ	101
6.6	章末演習問題	106
第 7 章	算法 3：数論 (続)	109
7.1	2 進数と 10 進数	109
7.2	素数、再訪	112
7.3	大きい数のべき乗計算	114
7.4	暗号を作る	117
7.5	章末演習問題	122
第 8 章	算法 4：ソート・マージ	123
8.1	選択法ソート	123
8.2	バブルソート	127
8.3	挿入法ソート	129
8.4	マージ	132
8.5	その他のソートアルゴリズム	135
8.6	章末演習問題	139
第 9 章	関数定義	141
9.1	関数定義プログラム	141
9.2	配列の受け渡し	145
9.3	戻り値のない関数	150
9.4	データ共有、間接参照	154
9.5	参考 ポインタ	158

9.6	2次元配列データの受け渡し	162
9.7	章末演習問題	168
第 10 章	算法 5：再帰計算	169
10.1	再帰的関数定義	169
10.2	ユークリッドの互除法、再訪	173
10.3	クイックソート	175
10.4	ハノイの塔	179
10.5	エイトクィーン問題	182
10.6	章末演習問題	188
第 11 章	算法 6：方程式を解く	191
11.1	二分法	191
11.2	ニュートン法	196
11.3	連立一次方程式を解く	199
11.4	章末演習問題	204
第 12 章	算法 7：モンテカルロ法	205
12.1	モンテカルロ法	205
12.2	モンテカルロ法の推定精度	213
12.3	擬似乱数の生成	215
12.4	章末演習問題	220
第 13 章	ファイル入出力	221
13.1	ファイルへの出力 (fopen, fclose, fprintf)	221
13.2	ファイルからの入力 (fscanf)	225
13.3	章末演習問題	228
付録 A	記号一覧表	229
A.1	算術演算子	229
A.2	代入演算子	229
A.3	関係演算子	230
A.4	論理演算子 (優先順)	230
A.5	データ型 (ビット数) (例外あり)	230
A.6	フォーマット指定子 (printf, scanf)	231
A.7	エスケープ文字 (printf)	231
A.8	予約語	231

A.9	ヘッダファイル (<code>#include</code> で使用)	232
A.10	ファイルのモード (<code>fopen</code>)	232
付録 B	良くある質問	233
B.1	システムの問題	233
B.2	プログラムの書き方	233
B.3	エディタの使い方	234
B.4	プログラムの実行	234
B.5	デバッグ	235
B.6	文法：データ入力、表示	236
B.7	文法：配列	236
B.8	文法：関数	237
索引		239

第 1 章

初めての C

1.1 C 言語とは

C 言語は英語のような外国語の一つで、コンピュータに仕事（計算）をさせるときに用いる、コンピュータと会話するための言葉（の体系）です。言葉の体系ですから、文法があり、構文があります。アメリカ育ちなので英語をベースにしていますが、データを扱うために必要な、普段とはちょっと違う単語や略語が付加されていたりします。英語がベースとは言っても、コンピュータが行う仕事の指示を与えるだけですから、6 年勉強しても苦手な人が多い英語に比べて、使う単語の数はいくつもあります。

include, main, void, print(f), system, pause, scan(f), int(eger), double, while, if, else, switch, case, default, do, break, continue, unsigned, for, long, float, return, define, (f)open, (f)close, char(acter), put(s), get(s), getchar, const(ant), exit, allocate, free

こんなところです。文法も単純で、このテキストをマスターすればかなり複雑な文章（プログラムと言います）が書けるようになります。ただし、ほかの外国語と違って言語といっても意思疎通する相手はコンピュータです。コンピュータは人間と違って「おもいやり」などなく、融通が利きません。文法にとってもやかましく、厳密に文法通りでないプログラムを書くとまったく通じません。スペルが違うとか、あるべき記号がないとか、「;」を「:」と書いてしまうとか、とにかく 1 個所でも間違いがあると、どんなに長いプログラムでも全部が無効になってしまうので、注意深くなければいけません。

ここでは Microsoft Visual Studio2010 C++ Express（以下 VC2010 という）を使うことを想定して解説します。書かれている内容について使用例を見ながら、全部覚えてください。中学校の英語の勉強を始めた頃を思い出してください。とにかく、きちんと覚えて、繰り返し使うこと（プログラムを書くこと）、これが上達の早道です。

この演習では時間の関係で C 言語だけを取り上げます。使用するアプリケーションプログラムは C++ となっています。C++ とは何か、ということについて少し説明しましょう。「++」は C 言語の仕様で言えば 1 を足すという意味です（索引で「++」を調べてみてください）。C の前身は B という言語でした。ですから、C の次の世代の言語は D... かとと思ったら C+1 が出てきたというわけです。新たに提案された言語は**オブジェクト指向**という新しい考え方に基づいて設計されていますが、その基本的な考え方は C を踏襲しています。それは C 言語がかなり完成されたものであるという証拠と言っても良いでしょう。

C 言語は**手続き型**とも呼ばれ、あらかじめ決められた手順にしたがってプログラムが実行されるような仕組みになっています。それに対してオブジェクト指向のプログラムでは、オブジェクトと呼ばれる独自の機能を持つ複数の主体を組み合わせ、それらの主体同士がメッセージを交換しながら一連の仕事を仕上げていくという仕組みを持っています。と言われてもよく分からないでしょう、詳しくは JAVA の解説書を読んでください。

C 言語以外にも、プログラミング言語は無数といって良いほどあり、それぞれの特徴を生かした適用分野で使われています。C はそれらの中にあって、最も歴史があり、多方面で使われているものの一つと言って良いでしょう。実社会で仕事をする場合、必ずしも C を使うとは限りませんが、その考え方やプログラミングの仕方は共通なところが多々あり、C できちんとプログラムが組めれば、新しい言語を使いこなすことは難しいことはありません。実際、例えば、このテキストに載っているプログラム例は入出力の部分にちょっとだけ手を加えるだけで JAVA のプログラムに読み替えることが出来ます。

このテキストはもちろん C 言語のプログラミングを目的で書かれたものですが、他の言語のプログラミングについても有効な事項の解説にも気を配ったつもりです。特にアルゴリズムについては、プログラミング言語によらず、共通の内容になっています。プログラムを書きながら、その考え方についてきっちりと身につけるように勉強してください。

1.2 学習の仕方

この演習書では、C 言語についての基本的な文法を学び、その道具を使って、実際の計算がどのような手順で行われるか、ということを学びます。計算の手順を**アルゴリズム（算法）**と言います。例えば、2 桁同士のかけ算は、まず乗数の 1 の位の数と被乗数を掛け、次に乗数の 10 の位の数字に被乗数を掛け、それを 10 倍したものを前の結果に加える、というような手順で答えを求めますが、その手順を記述したものがアルゴリズムです。

コンピュータは単純な四則演算は出来ませんが、それらを組み合わせて数学的な問題を解くためには、その解き方をコンピュータの理解できる計算要素に分解して、教え込まなければいけません。それがアルゴリズムなのです。コンピュータが理解できる文章規則について、このテキストの「文法編」で解説します。そして、いくつかの典型的な数学の問題をどのようにアルゴリズムで記述するか、というについて「算法編」で解説します。

アルゴリズムの善し悪しで計算時間が大幅に違うということはよくあることです。常に計算の効率を考えなければいけないということは、普段の生活と同じですが、コンピュータの世界ではことのほか重要になります。基本算法をしっかりと身に着けてください。

このテキストには C 言語のプログラムの基本的なことはだいたい書かれていますので、指示通りコンピュータにプログラムを入力しながら読めば、ある程度のレベルまで到達できるはずです。プログラムの書き方を覚えるためには読み方を学ぶのがいちばんです。最初のうちはまず「**実習**」のプログラムをそのまま入力して実行させ、得られた結果と解説の説明を読みながらプログラムの各行を理解してください。これはいわば英文和訳のようなもの（C 文和訳？）です。

ある程度読めるようになったら和文 C 訳に挑戦してください。それまでの知識で解けるような「**練習問題**」を用意していますので、積極的に挑戦してください。物足りない人のためには、さらに章末の「**演習問題**」が用意されています。

指示通りにやっても動かない、説明文が理解できない、など、分からない場合は遠慮なく質問してください。ただし、自分はこうやった、こう考えた、でもわからない、という作業履歴を添えて質問してください。頻度の高いいくつかの質問については付録（良くある質問）に回答がありますので、まずはそれをチェックしてください。また、その他の良くある質問については、このテキストのサポートページ

<http://www.f.waseda.jp/sakas/cProgramming/>

に載せましたので、参考にしてください。サポートページにはテキストに書ききれなかった内容や、典型的なプログラム例などもありますので、参照してください。

このテキストだけでは不安だ、という人は C 言語の解説書、自習書が本屋に行けば数多く発見出来ます。定評のあるものを参考書に挙げましたが、この本でなければいけない、ということはありませんので、自分のレベルにあった本を見つけてください。

練習問題 1.1 C 言語で用いられるという 1 ページの英単語をすべて和訳しなさい。一部、想像力が必要なものもあります。

参考書

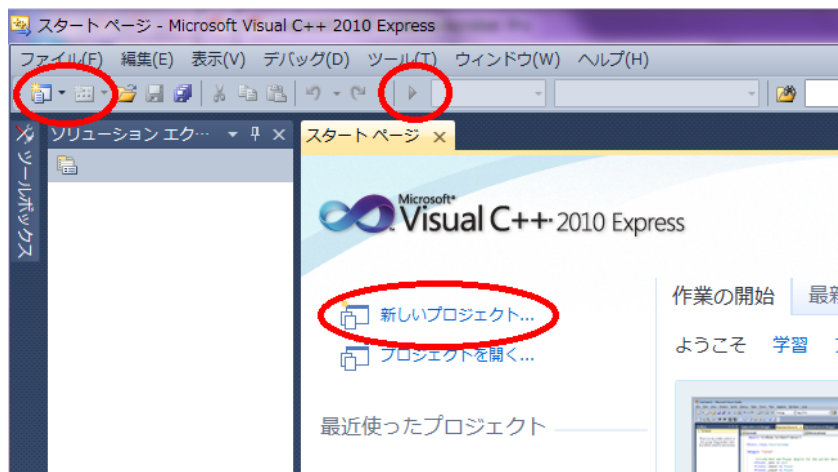
1. 林晴比古「新訂新 C 言語入門」（ビギナー編）、ソフトバンク、2008。
2. 柴田望洋「新版明解 C 言語入門編」、ソフトバンク、2004。
3. 高橋麻奈「やさしい C」、ソフトバンク、2007。

1.3 Hello C world.

C のプログラミングを一通り経験するために、簡単なプログラムを入力して実行する、という一連の作業を実習しましょう。VC2010 の豊富な機能の中から、簡単なプログラム演習をするのに必要な部分だけを取り出して使用します。プロジェクト (project)、ソリューション (solution)、など、専門用語がたくさん出てきますが、一々用語や機能を説明しながら進めると煩わしいので、最初はそういうものだと思って、書かれた通りに実行しなさい。

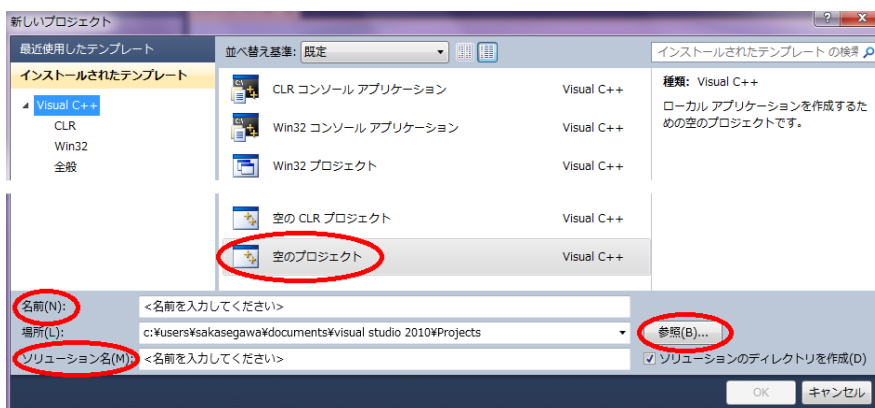
実習 1 (最初の C プログラム)

1. **(フォルダ作成)** 最初に、この演習で作成するプログラムを一括して保存するフォルダを「マイドキュメント」の中に作ってください。「C 言語演習」というような名前がよいでしょう (が、ご自由に)。
2. **(スタートページ)** ソフトウェア VC2010 を立ち上げると「スタートページ」というウィンドウが表示されます (下図)。メニューとして「ファイル」「編集」「表示」などが表示され、メニューバーの下にアイコン (ボタンと言います) が並んでいます。ボタンの上にマウスを置くとボタンの名前が表示されます。左から順に「新しいプロジェクト」「新しい項目の追加」「ファイルを開く」「保存」などなど。特に良く使用されるのは最初の二つと少し右にある右向き三角形 (「開始」ボタン) です。覚えておいてください。ボタンの下に「ソリューションエクスプローラ」画面と「スタートページ」というタブ付きの小画面が表示されます。



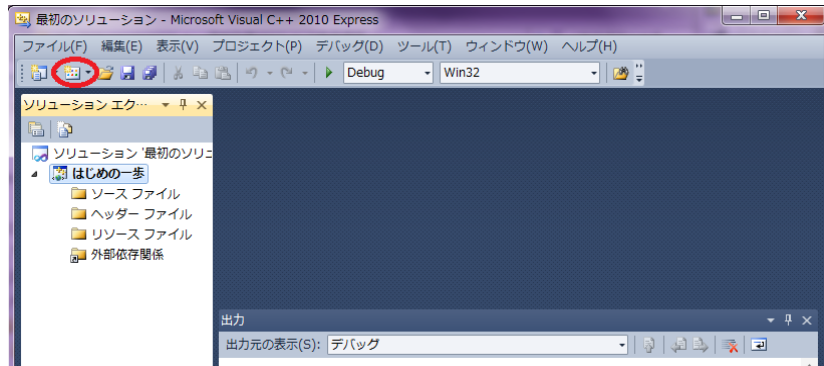
3. (新規プロジェクト作成) 早速、左端のアイコン「**新しいプロジェクト**」ボタンをクリックしなさい。^{*1}

(⇒ 「新しいプロジェクト」ウィンドウが表示される)



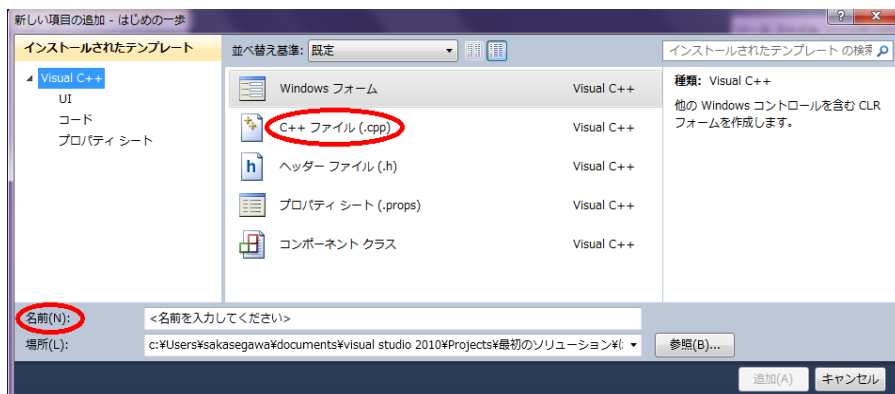
4. 表示される「新しいプロジェクト」ウィンドウの左側にある「インストールされたテンプレート」から「Visual C++」を選択し、
- (a) 中央の枠から「**空のプロジェクト**」をクリックし、「名前」枠に「はじめの一步」（自分で付けたい名前で構いません）と入力し、
 - (b) 「場所」枠の右にある「参照」ボタンをクリックして（ワープロで文書を保存するときのような）ファイル指定用のダイアログを表示させ、ステップ1で作った「C 言語演習」フォルダを選び
 - (c) 「ソリューション名」は「名前」と同じ文字列が入っているはずですので、そのままでも良いし、別の名前、例えば「最初のソリューション」というような名前を入力しても構いません。
 - (d) 「OK」ボタンをクリックする。
- (⇒ 「ソリューションエクスプローラ」ウィンドウに「はじめの一步」という名前のプロジェクトが表示される)

^{*1} あるいは、「スタートページ」タブの付いた小画面の左上にある「新しいプロジェクト...」という文字列をクリックしても同じ。あるいは Ctrl キーと Shift キーを押しながら「N」キーを押しても同じです。



5. (新しい項目追加) 次いでアイコンの左端から 2 番目、「新しい項目の追加」ボタンをクリックしなさい*2。

(⇒ 「新しい項目の追加」ウィンドウが表示される)



6. 表示される「新しい項目の追加」ウィンドウで

- (a) 「インストールされたテンプレート」から「Visual C++」をクリック、
- (b) 真ん中の枠から「C++ ファイル」をクリック、
- (c) 下の枠の「名前」に「hello.c」と入力して(「.c」を付けるのを忘れないように)
- (d) 「追加」ボタンをクリックします。

(⇒ 「hello.c」というタブの付いた小画面が表示される)

7. (プログラムの入力) プログラムは「直接入力モード(半角)」で入力するのが原則です。キーボード左上の「半角/全角」キーを押すたびに、画面右下に「あ」と

*2 あるいは、「ソースファイル」の文字列の上でマウスを右クリックしてポップアップメニューを表示させ、「追加」→「新しい項目」メニューを選択しても同じ。あるいは Ctrl キーと Shift キーを押しながら「A」キーを押しても同じです。

「A」の表示が変わるアイコンがありますが、それが「A」の状態を直接入力モードと言います。直接入力モードにしてから次のプログラムを入力しなさい。各行の先頭の数字は説明のために付けたものですので入力しないでください。各行の終わりでは「Enter」キーを押しなさい。4行目の終わりで Enter キーを押すとカーソル位置が先頭に戻りません（**字下げ**あるいは**インデント**と言います）が、そのままの位置で、`printf` から始まる 5 行目を入力しなさい。「;」はセミコロンという記号です。「:」（コロン）と間違えないように。また「()」と「{ }」の違いも注意しなさい。

プログラム例

```
1: // はじめての C プログラム
2: #include <stdio.h>
3: #include <stdlib.h>
4: int main() {
5:     printf("Hello C world!¥n");
6:     system("pause");
7: }
```

8. **(デバッグ／ビルド)** ファンクションキーの 5 番、「**F5**」キーを押しなさい。あるいは同じことですが、右向きの三角形の形をしたアイコン（「デバッグの開始」ボタン）をクリックしなさい。そうすると、「ビルドしますか」というダイアログボックスが出ますので、内容を確認してエンターキーを押しなさい。
9. **(実行結果)** もし、プログラムが正しく入力されているとすると、プログラムを入力した小画面の下の「出力」というタイトルの付いたウィンドウにいろいろな文字列が表示された後、黒いウィンドウ（**DOS 画面**という）が表示され、「Hello C world!」という文字列と、「続行するには何かキーを押してください…」という文字列が表示され、そのすぐ右でカーソルが点滅した状態になります。これが、入力したプログラムが正しく実行された結果です。指示にしたがってなんでもいいからキーを押すと、もとのプログラム入力画面に戻ります。
10. プログラムが正しく入力されていない場合は、「出力」ウィンドウに「0 正常終了、1 失敗」というような表示があり、その上にいろいろなエラーメッセージが表示されます。エラー情報の文字の上をダブルクリックするとエラーのある箇所に点滅するカーソルが表示されますので、それを頼りに間違いを発見し修正しなさい。例えば、良くあることですが、6 行目の最後に「;」を入力し忘れてビルドすると「構文エラー」という文字列が表示されるでしょう。
11. VC2010 を終了するには、右上のクローズボックスをクリックするか、「ファイル」→「終了」メニューを選択しなさい。

実習 1 の解説 チェックボックスに ✓ を入れながら読みなさい。

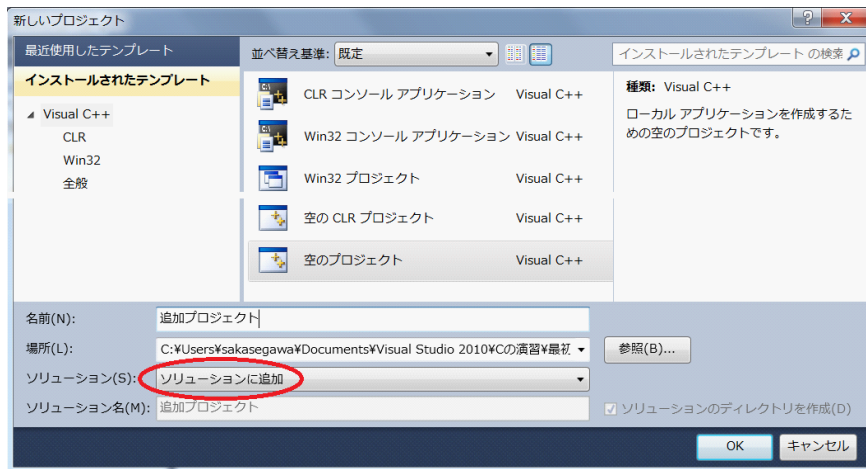
- 1. **(直接入力モード)** プログラム作成時の入力文字は、半角の英数字と記号です。「半角英数モード」ではなく「**直接入力モード**」を選択しなさい。但し、5 行目の「"」に囲まれている場所には日本語を入力することが出来ます。
- 2. **(拡張子)** 「.c」は**拡張子**といい、このファイルが C のプログラムであることを MS-VS2010 に知らせます。これにより、include の色が変わったり、自動的に字下げが行われたりします。
- 3. **(ビルド、コンパイル、リンク)** 「**ビルド**」では、コンピュータは入力されたプログラムが正しいかどうかをチェックし、正しくなければエラーメッセージを表示して止まります。正しければ `stdio.h` を include し、コンピュータが実行できるような一連の命令に置き換えます。これを**コンパイル&リンク**と言います。
- 4. **(デバッグ)** ビルドした後に「ビルドエラーが発生しました」と表示された場合は、画面下部の「出力」ウィンドウ画面にそのエラーの原因を見ることができます。「warning (警告)」は無視して良い場合もありますが、「error」は修正しないと先へ進めません。エラーメッセージの上をダブルクリックすると、エラーが発見されたプログラムの場所を指し示してくれます。それを参考にしてプログラムを修正して実行し直しなさい。この作業を**デバッグ**と言います。ときには、間接的なエラーメッセージというものもあって、ほかの部分の直すと自然に解消されるようなエラーもありますから、エラーメッセージが大量に表示されたときは、全部修正しようと思わず、「絶対間違い」という箇所だけ直してコンパイルし直す、またエラーが出てきたらそのときはまた考える、というように作業すると良いでしょう。
- 5. **(行番号表示)** エラーメッセージには行番号が添えられていて、何行目にエラーがあるか教えてくれます。ダブルクリックするとその行が表示されますが、その都度ダブルクリックするのも面倒です。といって、上から何行目、ということ数を数えるのも煩わしい。行番号を表示させる機能があるので、それを利用しましょう。「ツール」→「オプション」メニューを選択すると、「オプション」ウィンドウが表示されます。その左欄にある「テキストエディター」の左にある右向き三角形をクリックしていろいろなメニューを表示させ、「C/C++」の左にある右向き三角形をクリックし「全般」をクリックします。そして、右欄に表示される「表示」の下にある「行番号」にチェックマークを付けます。これで、行番号が表示されるはずです。

1.4 プロジェクトの追加

このテキストは、見本のプログラムを入力して実行させ、その結果を見ながらいろいろな知識を解説する、という手順になっています。以前に作った（入力した）プログラムを後で参照したり再利用したりすることもあるので、**それぞれのプログラムは別々のプロジェクトとして保存**しておいてください。実習 1 にしたがってプロジェクト毎に新規ソリューションを作成しても良いのですが、せっかく階層構造になっているので、同じようないくつかのプロジェクトを一つのソリューションとしてまとめておくと、後の管理がしやすくなります。すでにあるソリューションにプロジェクトを追加する手順を説明します。実習 1 のプロジェクトを作成した直後を想定します。

実習 2（プロジェクトの追加）

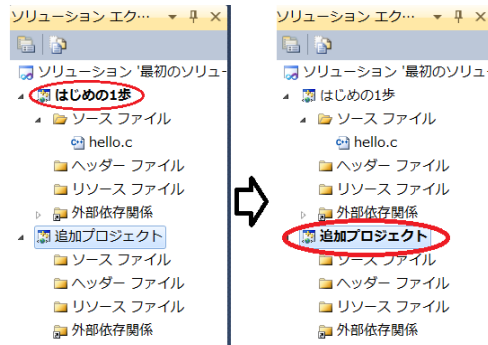
1. 「新しいプロジェクト」ボタンをクリックして「新しいプロジェクト」ウィンドウを表示させると、下の方にある「ソリューション」という文字の右に「新しいソリューションを作成する」という文字列が表示されていますから、その上をマウスでクリックし、表示されるメニューから「**ソリューションに追加**」を選択します。



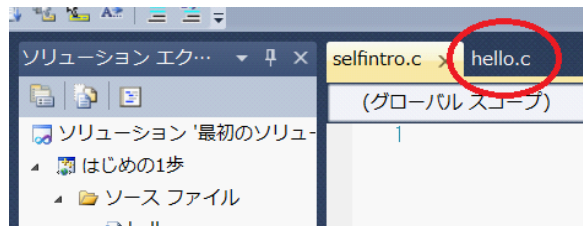
2. 「名前」の欄に適当な文字列を入力して「OK」ボタンをクリックします。これで、左のソリューションエクスプローラウィンドウに、新規作成したプロジェクトが追加されるはずです。
3. (スタートアップ)「プロジェクト」メニューの「**スタートアッププロジェクトに設定**」サブメニューを選びます*3。これで、新しいプロジェクトが太文字になり、今

*3 新たに表示されたプロジェクト名の上でマウスを右クリックし、表示されるメニュー（ショートカットメ

までのような作業が開始できます（これをしないと、前のプログラムが実行されてしまいます）。



4. 「新しい項目の追加」ボタンをクリックし、「selfintro.c」という C++ ファイルを新規作成します（実習 1 の手順参照）。
5. 新規作成された白紙の上にあるタグの横に実習 1 で作った「hello.c」タブがあるのでクリックし、全部選択し（Ctrl キーを押しながら「a」キーを押す、以降 Ctrl+a キーを押すと書く）、コピーします（Ctrl+c キーを押す）。



6. 再び「selfintro.c」のウィンドウに戻り、ペーストします（Ctrl+v キーを押す）。
7. 4 行目の「Hello C world!」の部分を「私は x x x です」という自己紹介の文字列に変えてください。
8. その行の最後の「;」の後ろにカーソルを置いて Enter キーを押して新しい行を挿入し、そこに「printf("...¥n");」という文字列を入力します（「...」の部分は自由な自己紹介を入力）。同じ手順で、試しにもう一行入力してみてください。
9. 入力し終わったら、F5 キーを押して正しく動くことを確かめなさい。

実習 2 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **（スタートアッププロジェクト）** 一つのソリューションには複数のプロジェクトを

ニューという）から選択することもできます。

組み込むことができますが、実行させたときに最初に動かすプロジェクトは決まっています。それを**スタートアッププロジェクト**と言います。新たに追加したプロジェクトを実行させたいのであれば、この例のようにそれをスタートアッププロジェクトにすると明示的に指定しなければいけません。

- 2. **(新たなプログラムの作成)** 新規にプログラムを作る場合、「#include...」から入力するのは面倒です。決まり切った命令文がいくつかありますから、すでに作った正しいプログラムをコピーペーストして、必要な箇所だけ追加修正する、というやり方は作業効率を上げるのに役立つでしょう。
- 3. **(新しい行の追加、字下げ)** プログラムに新しい行を追加する場合は**必ず**前の行の最後にカーソルを置いて Enter キーを押し、VC2010 の指示する字下げ位置から入力しなさい。

(追加の実習：4 行目の最後の「;」を削除し「)」の後で Enter キーを押すと、4 行目よりさらに字下げされてカーソルが点滅することを確認してください。こうなったときは、**バックスペースで開始位置を調整してはいけません**。前の行の入力が正しくないので、すぐにその間違いをチェックしなさい)。

(重要) 新しい行を挿入する場合は、「必ず」挿入する行の直前の行末にカーソルを置き、「Enter」キーを押しなさい。

- 4. **(ショートカットキー)** Ctrl+c はショートカットキーと呼ばれています。マウスで編集メニューをプルダウンさせコピーメニューを選択する、という動作と同じ結果を得ることが出来ます。ショートカットキーを使えば、作業効率が上がるので、積極的に使えるようにしておきなさい。ショートカットキーが何か忘れてしまった場合は、メニューをプルダウンさせると、目的のメニューの右に表示されています。ショートカットキーは Ctrl+v キーのように書くことにします。Ctrl+c (コピー)、Ctrl+v (ペースト)、Ctrl+z (やり直し) は覚えておきなさい。

1.5 レポート作成

次の実習は C のプログラムをレポートで提出する場合を想定した作業手順を体験するものです。実習 2 のプログラムとその実行結果をワードにまとめる作業を実習します。

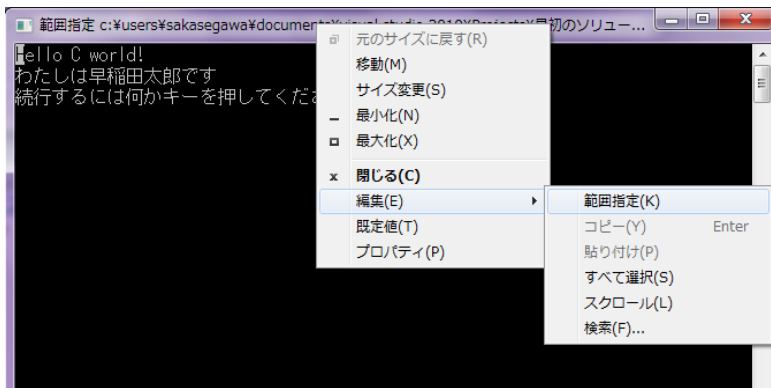
実習 3 (レポート作成)

1. **(プログラムのコピー)** プログラムを実行させ DOS 画面に正しい結果が表示されたことを確認したら、プログラムを全部選択し (Ctrl+a キー)、コピーしなさい (Ctrl+c キー)。
2. **(ワードへの貼り付け)** ワードを立ち上げ、「プログラム」と入力してから、Ctrl+v

キーを押すと、プログラムが表示されます。ん、日本語の表示がおかしいですね。やり直し (Ctrl+z キー)。マウスを右クリックし、表示されるメニューの中から「貼り付けのオプション (テキストのみ保持)」メニューを選択しなさい。

3. **(フォントの変更)** プログラムを表示させるときは、読みやすさを考えて VC2010 のエディタのように、「i」も「m」も同じ幅になるフォント (**等幅フォント**という) にするのが習慣です。貼り付けたプログラム全体を選んで、「ホーム」タブの「フォント」グループの中にあるフォントリストの中から「MS 明朝」を選びなさい。
4. **(実行結果のコピー)** DOS 画面に表示された実行結果の文字列を MS-Word へ貼り付ける場合、必要部分を選択して Ctrl+c キーを押しても反応しません。次の手順にしたがってください：

- (a) DOS 画面 (実行結果の表示される黒い画面) が表示されている状態で、タイトルバー (青い部分) に (ボタンを押さずに) マウスカーソルを移動させ、右ボタンをクリックし、表示されるメニューから「編集」→「範囲指定」メニューを選ぶ。
- (b) DOS 画面の左上で長方形が点滅を始めるので、マウスを使って実行結果部分をドラッグする (選んだ部分が反転する)
- (c) 「Enter」キーを押す (反転した部分が元に戻り、クリップボードに反転した部分がコピーされる)



5. **(実行結果の貼り付け)** MS-Word に戻り、「実行結果」と入力してから、Ctrl+v キーを押すと、いま範囲指定したものが挿入されるはずです。

実習3の解説 チェックボックスに✓を入れながら読みなさい。

- 1. **(MS-Word のスペルチェック)** MS-Word でプログラムを張り付けると、単語の下に赤い波線が付いてしまうことがあります。これは MS-Word の「スペルチェック機能」が働いているためです。「ファイル」→「オプション」メニューをえらび、

表示される「Word のオプション」ウィンドウで「文章校正」をクリックし、「Word のスペルチェックと文章校正」項目にあるすべての項目のチェックを外しなさい。

- 2. (MS-Word のオートコレクト) MS-Word の余計なお節介の一つに、英語の文章の最初は小文字だったら大文字にする、というのがあります。C の命令は全部小文字ですから困る場合があります (int が Int になるので、プログラムとしては間違い、当然添削に引っかかる)。「ファイル」→「オプション」メニューをえらび、表示される「Word のオプション」ウィンドウで「文章校正」をクリックし、「オートコレクション」ボタンをクリック、「文の先頭文字を大文字にする」のチェックを外して「OK」ボタンをクリックしなさい。

1.5.1 レポートの書き方

以上は、C のプログラムとその実行結果をワードに貼り付ける場合の手順を示したのですが、レポートそのものの書き方にも注意が必要です。レポートは人に見せるためのものですから、受け取った人があなたの記述した内容を正しく理解できるように、**相手の立場に立って**表現を工夫しなければいけません。入学時に配られた「**学習ガイド**」にある「**レポートの書き方**」を一度通読しなさい。

特にプログラムの添削の場合、プログラムだけ見せられてもなかなか理解するのが難しいものです。人のプログラムを読むのは大変！ そこで、次のような点に注意してレポート作成しなさい。

1. 問題の要点を書く (問題文を書き写す必要はありませんが、プログラムがどのような問題を解くためのものなのか、要点をまとめてください)
2. どのような方針でプログラムを作るのかを説明する (最初は方針も何も、順番に書くだけですが、そのうち、問題が複雑になってきたときに重要になります)。
3. プログラムと実行結果を載せる。プログラムは要所にコメントを付ける。正しい字下げを守る、フォントは等幅フォントにする。実行結果は必要十分に。
4. 工夫したところをアピールする。
5. 不幸にして完成しなかった場合は、作業手順を書き、どこまで到達できたかを説明する。エラーメッセージがあれば、それを添える。出来なかったからといって出さない、というよりは、まし。

練習問題 1.2 自分の名前、住所、郵便番号、電話番号、メールアドレス、(そのほか好きなもの)を見やすく表示させるプログラムを書きなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の 3 段階で各項目を評価し、F がある場合は、今のうちに復習してください。

- ☐ ☐ 新規プロジェクトの作成
- ☐ ☐ C ファイルの作成、拡張子 (.c)
- ☐ ☐ 直接入力モード
- ☐ ☐ C プログラムの入力、インデント (字下げ)
- ☐ ☐ ビルド、デバッグ、F5 キー
- ☐ ☐ 新規プロジェクトの追加
- ☐ ☐ スタートアッププロジェクト
- ☐ ☐ 新しい行の追加の仕方 (行末で Enter キー)
- ☐ ☐ ショートカットキー、特に a,c,v,z
- ☐ ☐ ワードへのプログラムの貼り付けと等幅フォント (MS 明朝)
- ☐ ☐ 実行結果 (DOS 画面) のワードへの貼り付け方法

第 2 章

データの入出力と簡単な計算

文法編その 1

この章では、データを入力し、加工して（計算して）その結果を表示する、というもっとも基本的な手順のプログラムを書けるようになるための実習を行います。主な実習項目は以下のようなものです。

- **データの入力**：scanf、データ型、変数とその宣言
- **結果の表示**：printf、フォーマット指定子
- **計算**：四則演算規則、代入文
- **数学関数の使い方**：数学関数ライブラリー

2.1 基本入出力と計算

キーボードからデータを入力して、簡単な計算結果を表示する、という基本中の基本を実習します。コンピュータプログラムの基本形は、(1) データを入力して、(2) 計算して、(3) その結果を表示する、(4) それを繰り返す、という 4 つの要素から成り立っています。Excel の計算と対比させて考えるとわかりやすいでしょう。例題として、ドルで買い物をしたときに円に換算するといくらになるか、ということを Excel を使って計算する場合を考えましょう。例えばセル A2:A4 に

	A	B
1	ドルを円に換算する	
2	=103.45	1ドルが103.45円ということ
3	=98.89	買い物金額(ドル)
4	=A2*A3	円に換算した買い物金額

と入力すると、セル A3 にレシートの金額を新たに入力するたびにセル A4 に円に換算された金額が表示されます。

C 言語の場合も同じような手順を踏みますが、Excel との違いはシートがコンピュータの内部にあって見えないことです。そこで、データを入力したり、結果を表示する、という人間との情報のやりとりの手順を明示的に指示する部分が必要です。それがどのようなものか説明するよりは、それを実現したプログラムを体験したほうが理解しやすいと思うので、早速その作業に取りかかりましょう。

最初に、1.4 節の手順に従い、

1. 新しいプロジェクトを追加してそれをスタートアッププロジェクトに指定し、
2. 新しい C++ ファイルを作り（ファイル名の最後に「.c」を忘れずに）、
3. 前に作ったプログラムの `printf` 文を除いた部分をコピーしておいてください。

実習 4 次のプログラム例は、ドル金額を円に換算して表示するプログラムです。この通りに入力しなさい。但し、先頭の数字は後の説明で使うために付いているだけなので、入力しないように。また、行と行の間の破線も読みやすくするためのものなので入力しないこと。上の手順に従っていれば、新しい入力画面には 2,3,4 行目と 13,14 行目は入力済みです。**4 行目の最後にカーソルを置いて Enter キーを押すところから始めなさい。**5 行目の字下げは、入力が正しければこの見本ようになりますので、気にしないで入力続けなさい。普通に入力していて、各行の開始位置がこの例のようにならなければどこかに入力ミスがあります。スペースキーやバックスペースキーで調整しないで、入力した中にあるミスを修正しなさい。特に、行末の「;」（セミコロンという）**を忘れないように**に注意しなさい。9 行目の「/*」から行末までは入力しなくて構いません。

プログラム例

```

1:  // 買い物金額のドル換算、円換算
-----
2:  #include <stdio.h>
3:  #include <stdlib.h>
4:  int main() {
-----
5:      int yen;
6:      double dollar, RATE=103.45;
-----
7:      while(1) {
8:          printf("買い物金額（ドル）を入力しなさい：");
9:          scanf("%lf", &dollar);    /* "%lf"は「エルエフ」 */
10:         yen = dollar * RATE;
11:         printf("%lf ドルは %d 円です。¥n", dollar, yen);
12:     }

```

```
13:      system("pause");
14:  }
```

プログラムの実行

プログラムを入力し終わったら、**F5 キー**を押しなさい*¹。あるいはマウスを使って、画面上部の「右向きの緑色の三角形」アイコンをクリックしても同じです。小さなウィンドウが表示され、「ビルドしますか?」と聞いてくるので Enter キーを押すと（「はい」ボタンをクリックしても同じ）、下方のウィンドウに文字列が何行か表示され、エラーがなければ DOS 画面が表示され、「買い物金額...」という 8 行目のメッセージが表示されて入力待ちになります。適当な買い物金額を入力すると、直ちに円に換算された金額が表示され、また入力待ちになります。

計算するデータがなくなったら DOS 画面の右上にある「**閉じる**」ボタンをクリックして閉じなさい。

プログラムに入力ミスがあると、「ビルドエラーが発生しました」というダイアログボックスが表示されるので Enter キーを押すと（「いいえ」ボタンをクリックしても同じ）、画面下方の「出力」ウィンドウに、そのエラー情報が書き込まれているので、それを参考に修正（**デバッグ**という）しなさい。例えば、「dollar」という文字列を 10 行目だけ「doller」と入力してしまった場合は、

```
...¥lesson1.c(10): error C2065: 'doller': 定義されていない識別子です
```

というようなメッセージが表示されます。「定義? 識別子?」が何か分からなくても構いません。doller の文字をみて、「入力ミスだ」と気が付くだけで十分です（それくらいは気が付きなさい）。「(10)」はエラーの見つかった行番号で、その文字列の上をダブルクリックするとプログラムの 10 行目にマーカーが表示されるので、間違いをチェックし、修正しなさい。

実習 4 の解説 チェックボックスに ✓ を入れながら読みなさい。

□ 1. **（プログラムの構成）** プログラム例は破線で 5 つの部分に区切られています。それぞれは次のような意味を持っています。

1. コメント行（1 行目）：プログラマーのメモ、入力しなくても良い。
2. 開始の定型文（2 - 4 行目）：理屈は抜きにして、とにかくこのまま書く（これが C

*¹ これは「デバッグ」メニューの「デバッグ開始」サブメニューを選択したのと同じ動作をするショートカットキーです。

のプログラムであることをシステムに伝える)。

3. 変数宣言 (5 - 6 行目) : 次の計算部分で使う「変数」を宣言 (事前登録) している。
4. 計算部分 (7 - 12 行目) : 実質の計算内容を記述する。ここを自分で書けるようにするのがプログラミングの目的。
5. 終了前の定型文 (13 - 14 行目) : このまま書く。

これ以降の実習プログラムでは、2 と 5 の部分を (場合によっては 1 も) 省略し、3 と 4 の部分だけを書くことにします。「プログラムを入力し」と書かれた場合は「2 と 5 の部分を補ってプログラムを入力し」と読み替えてください。

- 2. (書き方) 5 行目以降が実際に実行される部分ですが、「実行文は「;」で区切り、一つ以上の実行文を「{」「}」で囲んだものを実行文群として扱うことができる、というルールがあります。「.c」ファイルで作業する場合、「;」の後に改行すると次の行の開始位置は前の行と同じ、それ以外は字下げされます。例外として「}」で終わると「字上げ」されます (12 行目、14 行目)。改行、字下げはプログラマーの見やすさのためにあり、VC2010 にとってはあってもなくても同じです*2。1 行目にある「//」だけは例外で、「//」があると、そこから行の終わりまでは VC2010 は読み飛ばします。**コメント**と言い、何を書いても構いません。9 行目の「/*」「*/」もコメントを書く時に使われますが、「//」と違い、途中で改行されていても、これに挟まれている部分は VC2010 は読み飛ばします。
- 3. (プログラム構造、構文) 8 行目から 11 行目までを実行文の集まり (実行文群) と考えて B と置き換えると、7 行目から 12 行目までは

```
while(1) {B}
```

とまとめることが出来ます。これは **while 構文** と呼ばれ、「{」と「}」で挟まれた部分 (8 行目から 11 行目まで) を (無限に) 繰り返す」という意味があります (詳しくは次章で説明します)。同じように考えて、5 行目から 13 行目も実行文群として A と置き換えると、4 行目以降をまとめて

```
int main() {A}
```

と書くことができます。これは、A が最初に実行されるプログラムであることを意味します。こうして、一つ一つの実行文から離れて、全体像を把握しておくことは、プログラミング作業の見通しを良くします。3 章以降でもこのような表現を使うことにします。当面のプログラミングは、この A の部分の書き方を学ぶことです。

- 4. (データ型) コンピュータですから、数が基本です。数には **int 型** と **double 型** の 2

*2 2 行目から 14 行目まで改行無しに書いても同じ結果になりますが、そんなプログラムは見たくないですね。

種類があります。int 型は整数のみ^{*3}、double 型は実数（整数を含みます）という違いがあります。同じ数でも「3」と書くと int 型、「3.0」と書くと double 型になります。その違いは割り算で現れます。割り算の演算記号は「/」です。「7.0/4.0」は普通に 1.75 ですが、「7/4」は整数計算され、1 になります。

- 5. **（変数、宣言文、初期値）** キーボードからデータを入力したり、計算結果を保存するために変数が使われます。変数は英数字を使った文字列で、英字から始めなければいけません。但し、main とか printf のように VC2010 が特別の意味に使っている単語（**予約語**という）は使えません（1.1 節参照）。Excel と違って大文字と小文字は別物として扱われます（例 yen と YEN は別の変数）。変数は最初に宣言しなければ使えません。5 行目は、「yen を int 型変数として使う」という宣言文、6 行目は「dollar, RATE を double 型変数として使い、RATE の**初期値**を 103.45 とする」という宣言文です。詳細は後で説明しますが、変数を宣言すると、その変数の内容を記憶するためにコンピュータのメモリが確保され、そのメモリの内容を変数名で参照することが出来るようになります。
- 6. **（データ表示 printf、書式文字列、フォーマット指定子）** 文字列や計算結果をディスプレイに表示するために printf 文を使います。printf はプリントエフと読みます。最後の「f」はフォーマット（format 書式）の f で、最初の"で囲まれた文字列を**書式文字列**といい、表示の書式を指定します。8 行目は文字列を表示するだけですが、11 行目は、変数の内容を表示するために、**フォーマット指定子**と呼ばれる %d,%lf や改行を意味する「\n」が含まれています。「%d」は int 型データを表示するときに使い、「%lf」（パーセント エル エフ）は double 型データを表示するときに使います^{*4}。それ以外の文字列は、その通りに表示されます。書式文字列に続けて、書式文字列に含まれるフォーマット指定子に対応するデータ（定数、変数、数式）をカンマ区切りで並べます。printf が実行されると、書式文字列のすべてのフォーマット指定子に対応するデータに置き換えた文字列を作り、それをディスプレイに表示します。
- 7. **（データ入力 scanf）** 9 行目は、「キーボードから double 型数を入力して、変数 dollar に記憶させる」という意味です。キーボードからデータを入力するときは scanf 文を使います。scanf はスキャンエフと読みます。printf 文と同じように、書式文字列に続けて、変数名を書きます。書式文字列はフォーマット指定子だけを書きます。int 型数を入力したい場合は「%d」、double 型数を入力したい場合は「%lf」を使います。もちろん、変数はそれぞれの型で事前に宣言されたものでなければいけません。変数名の前には必ず「&」という記号を付けます。

^{*3} int は integer（整数）の略。

^{*4} %f でも良いのですが、次の scanf の場合と揃えておくことにします。

入力するデータが複数ある場合は、フォーマット指定子を並べ（間にスペースを入れても構いません）、変数はカンマで区切って並べます。データを入力する際は、複数のデータをスペース区切りで入力（カンマ区切りだとエラーになります）した後に Enter キーを押せば、一度に入力することができます。

（重要）データを入力する場合は、変数の前に「&」をつけなければいけない。

- 8. **（代入文）** 10 行目の「=」は、右辺の数式を計算して、その結果を左辺の変数に代入するという意味があります。6 行目の「=」も同じ意味です。

（重要）「=」記号は数学で使う等号（左辺と右辺は等しい）ではありません。

- 9. **（キャスト）** 10 行目のように、double 型数を int 型数に代入するときに小数点以下の数字が失われます。VC2010 はビルドするときにこれをめざとく見つけて、

「warning C4244: '==' : 'double' から 'int' への変換です。データが失われる可能性があります。」

という警告メッセージを出します*5。警告を受けたくない場合は、プログラムの中に double 型数を int 型数に変換する命令を使って、次のように書きます。

```
yen = (int)(dollar * RATE);
```

「(int)」を**キャスト**といい、その後の括弧に囲まれた部分（括弧がなければ直後の数、あるいは変数）を計算した結果の小数点以下を切り捨てるという意味を持ちます。「(int)」の後の () を省略して「(int)dollar * RATE」とすると「((int)dollar) * RATE」と書いたのと同じで、先ず dollar を int 型に置き換えて（小数点以下を切り捨てて）から、それを円に直して、その小数点以下を切り捨てるという意味になるので、結果が違ってきます。int 型数を double 型数に変換する場合は「(double)」を使います。例えば「(double)n / m;」のように。

- 10. **（演算記号）** 計算式は、足し算引き算は「+」「-」、掛け算は「×」ではなくて「*」、割り算は「÷」ではなくて「/」、分数表現は使えない（入力しようがない）、ということに注意しなさい。a,b が int 型の時、「a/b」は商、「a%b」は「あまり」が計算されます。「*」「/」が「+」「-」に優先するという**自然な演算の順番を変えるための括弧**は「(」「)」しか使えません。割り算の場合、分子（被除数）あるいは分母（除数）が計算式の場合は、境界が分かるように「(」「)」でくくらないといけません。（例「(1+3)/(2*3-1)」と「(1+3)/2*3-1」は違う）。また、Excel でべき乗の計算を「^」で表しましたが、C では別の意味に使われるので、使えません。

*5 エラーがなくて実行してしまうと気がつきませんが、画面下の「出力」画面の「出力元の表示」から「ビルド」を選べると警告 warning メッセージを見ることが出来ます。

- 11. (プロンプト) 8 行目はなくても構いませんが、実行直後に DOS 画面でカーソルが点滅しているだけだと何をして良いか分からないので、このような指示を表示する (プロンプトという) ことは有効で、良い習慣です。

冒頭に書いた、同じ内容を計算する Excel と対比させるとわかりやすいかもしれません。セル A2 は 6 行目の「RATE=103.45;」に対応し、セル A3 は 9 行目に対応し、セル A4 は 10 行目に対応します。Excel ではデータを入力すると直ちに計算結果がセル A4 に表示されますが、C では目に見えるシートがないので、セル A2 に RATE、セル A3 に dollar という名前をつけ、そのセルに値を入力するために 6 行目のような「代入文」を使ったり、9 行目のような scanf という命令を使います。さらに、セル A4 を画面に表示させるために printf 文が必要になります。セル A3 に別のドル金額を入力する、という動作を 7 行目の while 命令で実現しています。

実習 5 実習 4 のプログラムの 11 行目と 12 行目の間に次の 4 行を挿入して実行しなさい。但し、新しい行を挿入する場合は、必ず、前の行の最後 (この場合は 11 行目の「;」の後) にカーソルを置いて Enter キーを押し、字下げされている位置から入力すること (この場合は 11 行目の「printf」の開始位置に来るはず)。字下げの位置を変えないこと。11.4 行目の「%.2lf」は「%lf」の「%」と「lf」間に「.2」を挿入したものです。

プログラム例

```
11.1:      printf("買い物金額 (円) を入力しなさい:");
11.2:      scanf("%d", &yen);
11.3:      dollar = yen / RATE;
11.4:      printf("%d 円は %.2lf ドルです。¥n", yen, dollar);
11.5:      printf("dollar は %lf のままです。¥n", dollar);
```

実習 5 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (計算結果のデータ型) 11.3 行目のドルを計算する式で、yen という int 型数を RATE という double 型数で割っていますが、このような計算は**混合計算**といい、結果は double 型になります。もっと複雑な数式の場合、演算規則の優先度にしたがって順番に計算されますが、double 型数が出てきた段階で int 型数の double 型数への変換が実行されます。例えば、m,n を int 型、x を double 型とすると、「m/n/x」は m/n で切り捨て型の割り算を実行した後、その結果を double 型に変換して x で割りますが、「m/x/n」は最初から double 型の割り算が実行されるので、両者の結果が違ってくる可能性があります。
- 2. (フォーマット指定子のオプション) 11.4 行目を実行すると、どんな計算をしても小数点以下 2 桁しか表示されません。「%.2lf」(21 ではなくて、「2」「エル」) の

「.2」は小数点以下 2 桁までを表示することを指示するオプションです。別のオプションとして、「%10.21f」とすると、表示範囲全体の長さを指定することも出来ます。例えば、12.3456 を「%10.21f」で表示すると、小数点 3 桁以下を四捨五入した 12.35 の前に 5 個の空白文字を付けて、長さ 10 の文字列として表示されます。ただし、それは表示結果を変えるだけで、変数の中身を変えているわけではないことが 11.5 行目で分かります。

とりあえず、これだけの知識でいろいろなプログラムを書いてみよう。習うより慣れろ、です。

練習問題 2.1 実習のプログラムを書き換えて、為替レートを `scanf` 文で入力できるようにし、いろいろな為替レートで計算できるようにしなさい。

2.2 プログラミング練習

覚えた知識は実際にプログラムを書くことによって記憶が定着します。これまでの予備知識を整理しておきますので、これらを使って、例題に挑戦してください。

1. 数には `double` 型数（普通の数全体）と `int` 型数（整数、小数点のない数）とがある。（例 同じ数でも 3 と書くと `int` 型、3.0 と書くと `double` 型）
2. 四則演算の演算子は「+」「-」「*」「/」、Excel のようなべき乗はない。
3. `int` 型数同士の計算結果は `int` 型数になる。（例 $(1+2)/2$ は 1.5 ではなくて 1）
4. `int` 型数同士の割り算のあまりを計算する演算子は「%」。（例 $7/3$ は 2、 $7\%3$ は 1）
5. `double` 型数同士、あるいはどちらか一方が `double` 型数の場合、計算結果は `double` 型数。（例 $(1+2)/2$ は 1、 $(1+2)/2.0$ は 1.5）
6. `double` 型数を `int` 型変数に代入すると、小数部が切り捨てられる。
7. 使用する変数は、開始の定型文の直後で宣言しなければいけない。
8. 代入文の左辺は必ず変数が一つだけ。右辺の結果を左辺の変数に代入するという意味がある。左辺と右辺が等しいということではない。

アドバイス

開始の定型文（実習 4 の 2 行目から 4 行目）と終了前の定型文（同じく 13 行目と 14 行目）はどんな問題にも共通なので、新しいプログラムを作る場合は、新規作成したファイルに、前に作った正しく動くプログラムのその部分をコピーペーストしてから作業を始めると、入力ミスもなくなり、作業効率が上がります。各自、工夫しなさい。

例題 2.1 2つの `int` 型数 a, b をキーボードから入力して、実数計算としての $\frac{a}{b^3}$ を計算して表示するプログラムを書きなさい。

ヒント：`int` 型数のまま割り算 (a/b) をすると切り捨てられるので、`double` 型数に置き換えて計算する必要があります。その場合は `double` 型変数を定義して (`double c;`) そこに `int` 型数を代入すれば良い (`c = b;`) のですが前のページで説明したキャストを使うと、いちいち変数を定義しなくても `double` 型の計算ができます (`(double)a/b` のように)。 b^3 の計算は、Excel の計算に慣れているとうっかり「`b^3`」と書いてしまいが、「`^`」は別の意味で使われているので間違いになります。「`b*b*b`」としなければいけません。

ほとんど答え：

プログラム例

```
int a,b;
double c;
scanf("%d %d", &a,&b);
c = a;
printf("%d / %d^3 は %lf¥n", a, b, c/(b*b*b));
// あるいは
printf("%d / %d^3 は %lf¥n", a, b, (double)a/(b*b*b));
```

補足説明 複数のデータを入力させたい場合は、このように一行にまとめることができます。入力する場合は、数をスペースで区切ります（カンマで区切るとエラーになります）。

結果を表示するフォーマット指定子は当然「%lf」です。「c/(b*b*b)」の代わりに「c/b/b/b」と書いても構いません。うっかりミスと思いたいのですが、こういう計算で括弧を付けずに「c/b*b*b」と書く人が少なくありません。コンピュータはプログラマーの意図通りには動いてくれません。書かれたプログラムを規則に従って忠実に実行するだけです。「a/b^3」もよく犯す間違いなので注意しなさい。

例題 2.2 3つの int 型数をキーボードから入力し、その平均値を計算するプログラムを書きなさい。

ヒント：int 型数でも平均値を計算すると小数点以下の数が必要になります。したがって、平均値を計算するときは double 型で計算する必要があります（規則 3）。そこで、「(a+b+c)/3.0」のような書き方になるでしょう。

ほとんど答え：（小数点以下の表示桁数指定を使っています）

プログラム例

```
scanf("%d %d %d", &a, &b, &c);
printf("平均値は %.3lf¥n", (a+b+c)/3.0); // 「31」は3とエルです
```

補足説明 キャストを使って「(double)(a+b+c)/3」と書いても構いませんが、定数が含まれる場合は「3.0」のようにした方が簡単、入力ミスの可能性が小さくなります。「(a.0+b+c)/3」と書いた人がいた（！）けれど、そんな規則はありません、気持ちは分かるけれど。

例題 2.3 2つの double 型数 x,y をキーボードから入力して、x/y の整数部分と小数部分を計算し、表示するプログラムを書きなさい。

ヒント：int 型変数を宣言して、 x/y をその変数に代入すると整数部分だけが記憶されます（ドルを円に直した計算を思いだそう）。小数部分は普通の割り算 (x/y) から整数部分 ($(\text{int})(x/y)$) を引けば良い。「 $(\text{int})(x/y)$ 」は 2 度出てくるので、いったん int 型変数に置き換えた方が見やすいでしょう。

ほとんど答え（もうそろそろ見ないでも）：

プログラム例

```
double x, y;
int m;
scanf(..., &x, &y);
m = (int)(x / y);
printf(..., x, y, m, x/y-m);
```

補足説明 変数 m を使わないでキャストを使って「 $(\text{int})(x/y)$, $x/y-(\text{int})(x/y)$ 」という式を printf 文に直接書き込むと 3 行で書けます。しかし、プログラムは短ければ良いというものでもありません。この場合は、このように、一時変数を使った方がプログラムがすっきりします。また、同じものを 2 度入力しなくて済むので煩わしくなく、キー入力の回数が減ってミスが減ります。

int 型変数へ代入する場合は 4 行目のようなキャストは必要ありませんが、こうしておけば、切り捨てたということが明確になるので、そうすることをお勧めします。

例題 2.4 3 桁の一桁の数 ($0, 1, \dots, 9$) をキーボードから入力して、それを 3 桁の数に直し、その数とそれを 2 倍したものを表示するプログラムを書きなさい。

ヒント： a, b, c を 3 桁の数値に直すには $100a + 10b + c$ とします。printf 文に直接書き込んでも良いが、同じ計算式を 2 度書くのは煩わしいので、3 桁の数を変数に記憶させた方が良いでしょう。

ほとんど答え：

プログラム例

```
d = 100*a + 10*b + c;
printf( ... , d, 2*d);
```

例題 2.5 4 桁の数を一つの数としてキーボードから入力し、各桁の数字を逆順に表示する。例えば「1259」と入力すると「9521」と表示する。

ヒント: a の 1 の位の数は 10 で割ったあまりを計算すれば良い「 $a \% 10$ 」。10 の位の数は、 a を 10 で割った商の 1 の位の数に等しい ($1234 \% 10 = 4$, $1234 / 10 = 123$, $123 \% 10 = 3$, ...)

ほとんど答え:

プログラム例

```

a1 = a % 10;
a = a / 10;
a10 = a % 10;
a = a / 10;
...
printf( ... , a1, a10, a100, a1000);

```

例題 2.6 時分秒の 3 つの数 h, m, s をキーボードから入力して、それを秒数に直す。 h, m, s はいずれも `int` 型で非負、 $m, s < 60$ 。逆に、秒数を入力し、それを h 時間 m 分 s 秒の形式に置き換える。

ヒント: 上の二つの問題と同じ。秒数に直すには、10 倍、100 ($= 10^2$) 倍する代わりに、60 倍、 60^2 倍すればよい。($3600h + 60m + s$)。時分秒の秒を取り出すには、秒数を 60 で割ったあまりを計算すればよい。このような繰り上がりの数体系は 60 進法と言います。

練習問題 2.2 次のプログラムを実行したときに表示されるものを書きなさい。

```

// 問題 (1)
int a=1, b=20, c, d, e;
c = a - b;
d = b + c;
e = d - c;
printf("a = %d, b = %d; e = %d, d = %d\n", a, b, e, d);

```

```

// 問題 (2)
int a=1, b=20, d, e;
e = a;
d = b;
a = a - b;
b = b + a;
a = b - a;
printf("e = %d, d = %d; a = %d, b = %d\n", e, d, a, b);

```

2.3 数学関数

指数関数、対数関数、三角関数のような関数を使って計算をしたい場合は、Excel のようにシステムの用意した関数を使います。Excel で「=SQRT(2)」と入力すると 2 の平方根が計算されるように、VC2010 でも関数電卓で計算できるような基本的な数学関数は、プログラムで使えるようになっています。但し、全部小文字です。C は大文字と小文字を区別します。

実習 6 次のプログラムを入力して前書きと後書きを補って実行しなさい。

プログラム例

```
// 数学ライブラリーの使用
// #include <math.h> が必要
1:    double a, b, c, s;
2:    printf("平方根を計算します。 非負の数を入力しなさい ");
3:    scanf("%lf", &a);
4:    b = sqrt(a);
5:    printf("%lf の平方根は %lf, それを 2 乗すると %lf¥n",a,b,b*b);
```

実習 6 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (**数学関数ライブラリ**) 4 行目で出てくる `sqrt(...)` は平方根を計算する場合の書き方です。このように、数学で出てくる $\sqrt{x}$, x^y , $\log x$, e^x , $\sin x$, $|x|$ などの関数は、Excel の関数記号のように、値を与えるとその値に対応する関数値を計算してくれる関数群が提供されていて、必要な箇所に関数名を書けば答えが得られるようになっています。このような関数値を計算するプログラムを一つにまとめたものを数学関数ライブラリーといい、これを呼び出すことによって計算が実行されます。
- 2. (**math.h**) 数学関数ライブラリーを使う場合は、プログラムの先頭に

`#include <math.h>`

という宣言文を書く必要があります。この書き方は「`#include <stdio.h>`」の場合と同じです。丸暗記しなさい。

- 3. 数学関数ライブラリーには次のような関数が用意されています。Excel と違って、C では小文字と大文字を区別しますので、`SQRT` と書いても平方根の計算はしてくれません、注意しなさい。

数学関数	意味	C ライブラリー関数
$\sqrt{\quad}$	平方根	<code>sqrt(.)</code>
x^a	べき関数	<code>pow(x,a)</code>
$ \quad $	絶対値	<code>fabs(.)</code>
e^s	指数関数	<code>exp(.)</code>
<code>log()</code>	対数関数	<code>log(.)</code>
<code>sin()</code> , <code>cos()</code> , <code>tan()</code> など	三角関数	<code>sin()</code> , <code>cos()</code> , <code>tan()</code> , ...

参考記事 ヘッダファイル

(全体の流れを理解するための必須知識ではありませんが、知っているとお見通しが良くなるでしょう。余裕があるときに読んでみてください。)

`<math.h>` や前置定型文の中にある「`.h`」の付いた文字列は**ヘッダファイル**と呼ばれる、システムが用意したプログラムや小道具の集合を表す記号です。

`stdio.h` というヘッダファイルは `printf` や `scanf` のような標準的な画面表示やキーボード入力のための必要な小道具 (標準入出力 STanDard Input Output) を集めたもの、`stdlib.h` (STanDard LIBrary) は最後の「`system("pause")`」 (実行を一時的に中断する) を実行するための小道具などが集められているものでした。`math.h` は数学関数プログラムを集めたものです。

このようなヘッダファイルがあるために、プログラマは必要に応じて小道具の入っているヘッダファイルを `include` して、VC2010 が用意したたくさんのプログラムを利用することが出来るようになっています。

練習問題 2.3 「`sqrt`」の代わりに「`SQRT`」と入力してビルドするとどうなるか、エラーメッセージを調べなさい。

練習問題 2.4 x^y を C の数学関数 `pow` を使う方法 (`pow(x,y)`) と、 $x^y = \exp(y \log x)$ という関係式を利用する方法で計算し、その結果を比較しなさい (違いを詳しく調べるために、フォーマット指定子のオプションを利用しなさい)。

練習問題 2.5 三角形の3つの辺の長さ a, b, c を入力してその面積をヘロンの公式を使って計算するプログラムを書きなさい。但し、ヘロンの公式は $s = (a + b + c)/2$ として、三角形の面積は $\sqrt{s(s-a)(s-b)(s-c)}$ で与えられるというものです。

練習問題 2.6 三つの数 x, y, z を入力して、その平均 (算術平均、相加平均)、幾何平均 (相乗平均)、調和平均を計算するプログラムを書きなさい。但し、幾何平均は $\sqrt[3]{xyz}$ 、調和平均は $3/(1/x + 1/y + 1/z)$ で与えられます。

2.4 プログラムのデバッグとチェックリスト

エラーに遭遇したとき慌てなくてすむように、いろいろなエラー経験をしておきましょう。

実習 7 実習 4 のプログラムを使いましょう。元のプログラムを残しておきたい場合は、全部を新しいプログラムファイルにコピーペーストしてから始めなさい。以下の各項目毎に、元のプログラムを変えてビルドして、表示されるエラーメッセージを理解しなさい。

- (1) 「`#include`」の「`#`」を入力し忘れて「`include`」と入力する。
- (2) 「`<stdio.h>`」ではなくて「`<studio.h>`」と入力する（良くある間違い）
- (3) 「`int main ()`」の「`()`」を入力し忘れた場合
- (4) 5 行目の「`;`」を入力し忘れた場合
- (5) 5 行目の宣言文を書かなかった場合
- (6) 9 行目の「`&dollar`」を「`&doller`」とした場合（変数のスペルミス）
- (7) 「`scanf`」で変数の前に「`&`」を忘れた場合
- (8) 「`printf`」の書式文字列の最後の「`"`」を忘れた場合（良くある間違い）
- (9) 11 行目で「`,yen`」を書き忘れた場合

実習 7 の解説

(1) 「`#include`」の「`#`」を入力し忘れた場合：ビルドする前にプログラムを見てみると、何カ所かに**赤い波線**が引かれているのが観察できます。これはエディタのエラーチェック機能で、文法違反があるということを示しています。このような赤い波線が表示された場合は、ビルドする前にその付近をチェックすることでエラーを発見することができるかもしれません。今は、エラーメッセージを体験する、という実験なので、そのままビルドします（F5 キーを押しなさい）。結果は、ん！？ 大量の意味不明なエラーメッセージが表示されるでしょう。「うわーっ」と慌てないで、冷静になろう。行番号を見ると、作成したはずのない行番号でエラーが発生していることが分かります。こういう場合は、分かるエラーだけを修正して、もう一度ビルドしなさい。ほとんどのエラーメッセージが消えて無くなるはずです。

(2) 「`<stdio.h>`」ではなくて「`<studio.h>`」と入力した場合：リンクする前にエラーで止まってしまいます。エラーメッセージをみれば対応可能でしょう。

(3) 「`int main ()`」の「`()`」を入力し忘れた場合：これは単純、でも良くやる間違い。

(4) 「`;`」を入力し忘れた場合：5 行目末の「`;`」を削除すると、プログラムの中に、**赤い波線**が表示されます。無視して、あるいは気がつかずにそのままビルドすると、6 行目にエラーが見つかったというメッセージが表示されるでしょう。セミコロンがないと、5

行目と6行目はつながっているものと見なされるので、6行目の「;」までを一つの実行文と見なしているためです。もっとも、この間違いは、入力時に字下げされるので気がつくはずですが、気がつかないかもしれません。

(5),(6) 宣言文で名前を間違えた場合、宣言を忘れた場合：よくある間違いですが、これもすぐに気がつくでしょう。

(7) 「scanf」で変数の前に「&」を忘れた場合：これはビルドでエラーが出ないケースで、発見がやっかいです。すでに実習4で経験済みですが、エラーメッセージの種類は一通りではありません。とりあえず継続して、「閉じるボタン」で実行終了させるのが良いでしょう。実行時のエラーの主要原因ですが、間違いが発見しづらいものの一つですので、覚悟しておいてください。

(8) 「printf」の書式文字列の最後の「」を忘れた場合：最初の入力時ならば字下げが不正になるので入力時に気がつくでしょう。プログラムをいじっているうちにそうなってしまった場合は赤い波線が表示されるはずです。

(9) 「printf」文で、フォーマット指定子に対応するデータがない場合：実行結果がおかしいのはこのような場合が少なくありません。表示させる変数を書いたつもりになっていることが多いので、対応関係を調べてください。

プログラミングにエラーはつきものです。自分の作り出したエラーは自分で修復できなければいけません。その際、参考になるのは、ビルドした際に表示されるエラーメッセージです。それは、下方の「出力」ウィンドウで、「出力元の表示：ビルド」タブに表示されています。それを積極的に利用して、自力で間違いを発見できるようにしておいてください。

課題

今までの演習の中で経験したエラーのトップスリー（ワーストスリー？）を書きなさい。それ以外のよく間違える項目も含めてチェックリストにまとめ、このテキストの裏表紙に書き留めておきなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ データ型、「double 型, int 型」
- ☐ ☐ 変数と型宣言文、予約語
- ☐ ☐ `scanf` の使い方、「%,&」の意味
- ☐ ☐ `printf` の使い方、書式文字列
- ☐ ☐ フォーマット指定子とそのオプション、「%d,%lf,%10.2lf,...」
- ☐ ☐ コメント
- ☐ ☐ `while` 構文、`while(1) { ... }`
- ☐ ☐ 数式の書き方、「*,/,%」の意味、かっこの使い方
- ☐ ☐ 代入文
- ☐ ☐ キャスト
- ☐ ☐ 数学関数
- ☐ ☐ デバッグ、エラーメッセージの見方、利用法

2.5 章末演習問題

問題 2.1 double 型の小数点以下 4 桁以上ある正の数 a を入力し、小数点以下 3 桁以下を四捨五入したものを b とするプログラムを書き、結果が正しい（小数点 3 桁以下が 0 になっている）ことを確認しなさい。

問題 2.2 double 型の正の 2 数 a, b を入力し、 a を b で割ったものの小数点以下の部分を四捨五入した数（整数）を int 型変数 c に代入するプログラムを書き、その結果が正しいことを確認しなさい。

問題 2.3 入力された身長（センチ）と体重（キロ）から Body Mass Index (BMI、ボディマス指数) を計算するプログラムを書きなさい。但し、BMI とは、体重を、メートル換算した身長の 2 乗で割ったものです。例えば、170cm, 65.5kg の人の BMI は $65.5/1.7^2 = 22.7$ と計算されます。

問題 2.4 マラソンの 5 キロのラップタイムからゴールタイムを予測するプログラムを書きなさい。入力データは、スタート地点から計測点までの距離（5 キロ単位）、直前の 5 キロのラップタイム（何分何秒）、それまでの通算所要時間（何時間何分何秒）とします。例えば、「20, 15, 0, 1, 0, 0」と入力されたら（ということは、15 キロから 20 キロのラップタイムは 15 分 0 秒、20 キロの所要時間は 1 時間 0 分 0 秒ということ）、「ゴールタイムは 2 時間 6 分 35 秒です」と表示する。

問題 2.5 入学年度と 3 桁の通し番号を入力して、学籍番号のチェックディジットを計算するプログラムを書きなさい。また、通し番号とチェックディジットの間に「-」を入れた形式で学籍番号を表示するプログラムを書きなさい。プログラムのチェックのために、学籍番号が 1 桁、2 桁、3 桁の場合の 3 通りのケースについてチェックディジットを計算しなさい。但し、学籍番号のチェックディジットは次のようにして計算されます。

1. 創造理工学部を表す X 、経営システム工学科を表す C をそれぞれ 7, 8 に置き換え、それらを m_1, m_3 とします ($m_1 = 7, m_3 = 8$)。
2. 入学年度の下 1 桁の数を m_2 とします。
3. 3 桁の通し番号を 100 の位の数、10 の位の数、1 の位の数に分解し、それぞれ m_4, m_5, m_6 とします。
4. $(m_1, m_2, m_3, m_4, m_5, m_6)$ と $(2, 3, \dots, 7)$ の内積を M とします。
5. M を 11 で割った余りの数を K とすると、 $11 - K$ の 1 の位の数チェックディジットです。

第 3 章

条件分岐とフィードバック

文法編その 2

これまでの計算は簡単な例しか出さなかったので、書かれている順番に計算していけば結果を導くことができましたが、実際の問題では条件付き分岐や繰り返しなどが入り組んだ複雑な構造を扱う必要があります。この章ではプログラムの中核部分となる「計算」について、その計算を実行するためのプログラムの構造について解説します。主な実習項目は以下のようなものです。

- **条件付きジャンプ**：if ... else 構文、関係演算子、論理演算子
- **フィードバック構造**：while 構文、do ... while 構文
- break 文、continue 文

3.1 条件付き分岐 (if ... else)

プログラムは基本的に上から下へ実行されますが、入力されたデータ、あるいは計算結果にしたがって、ある部分だけを実行し他の命令をスキップするというように、その流れを変えたい場合があります。それを実現するのが if 文で、プログラムの基本構造の一つです。英文の「If you have any question **then** please let me know.」のようなもので、英文で「If P then A else B.」と書くべきところを C では「if (P) {A} else {B}」と書きます。Excel などコンピュータの計算道具には必ず含まれる基本中の基本です。考え方をしっかり身につけてください。

文法予備知識

1. ある条件を満たす場合だけ実行させたい場合は「if (条件式)」文を使う。
2. 条件式は条件演算子 (>, <, >=, <=, ==, !=) と論理演算子 (&&, ||, !) を使って表現する。
3. ある条件を満たす場合とそうでない場合に実行させる命令が違う場合は「if(条件式) {A} else {B}」と書く (条件式が成り立つ場合は A を実行し、さもなければ B を実行する)。

実習 8 次のプログラム(前書き(include + main() {})と後書き(system("pause");})は省略してあります、自分で補いなさい)は試験の判定プログラムです。入力して実行させ入力データと表示結果の関係を調べなさい(実習4のプログラムをコピーし、変数宣言文と、while ループの中身を書き換えれば、作業量が軽減されます)。

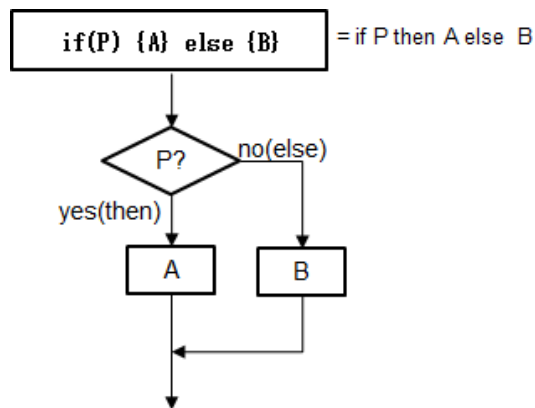
プログラム例

```
// 2 分岐判定 (if...else)
1:      int ten, PASS=60;
2:      while(1) {
3:          printf("試験の点数を入力しなさい...");
4:          scanf("%d", &ten);
5:          if (ten >= PASS) printf("合格です¥n");
6:          else printf("不合格です¥n");
7:      }
```

実習 8 の解説 チェックボックスに ✓ を入れながら読みなさい。

□ 1. (if...else) 5 行目 6 行目の「if ... else」は英語の意味通りに解釈します。もし (if) ten が 60 以上だったら「合格」、さもなければ (else) 「不合格」と表示されます。「if」の次にある () の中は条件式で、「>=」は「≥」と同じ意味です。「else」に続けて、if の条件式が成り立たない場合に実行されるべき命令を書きます。もし、条件が成り立たない場合に実行されるべき命令がなければ else そのものを省略することが出来ます。今の場合、合格通知だけ出せばよいのであれば 6 行目を書く必要はありません。流れを図に表すと次のようになるでしょう。

□ 2. (関係演算子) 条件式には次のような記号があります。「<=」は「≤」の代わり、



「>=」は「≥」の代わりです。「>」と「=」の順番に注意しなさい。「<=」「<」は意味をなしません。「!=」は「≠」の代わり、「==」は代入文と違い、左と右が等しいという意味です。これらは**関係演算子**と呼ばれます。特に間違いやすいのは「==」でしょう。「if(a = b)」と書いても文法的に間違いではありませんが、等号記号は代入文と見なされ、実際の計算結果は意図したこととは異なります。たまたまそれで正しい結果を導く場合もあるかもしれませんが、そんな幸運を期待してはいけません。ビルドしたとき何も警告を表示しません。注意しなさい。

a < b	a は b より小さい
a <= b	a は b 以下
a > b	a は b より大きい
a >= b	a は b 以上
a == b	a は b は等しい
a != b	a は b と等しくない

実習 9 次のプログラムは試験の集計プログラムです。(1) 入力して実行しなさい。(2) 8 行目の「else {」と 16 行目を削除して実行しなさい。データは「90 80」としなさい。なぜそうなったかを考えなさい。(3) 12 行目最後の「{」と 15 行目を削除して実行しなさい。データは「80 80」としなさい。結果はどうになりましたか。その理由を考えなさい。(4) 9 行目の等号を一つだけにして実行しなさい。データは「80 90」としなさい。

プログラム例

```

// 多分岐判定 (if...else if)
1:     int ten1, ten2;
2:     printf("試験の点数、2 回分を入力しなさい...");

```

```

3:      scanf("%d %d", &ten1, &ten2);
4:      if (ten2 < ten1) {
5:          printf("%d 点下がりました ",ten1-ten2);
6:          printf("復習が必要です。次回は頑張って¥n");
7:      }
8:      else {
9:          if (ten2 == ten1) {
10:             printf("変わりませんね¥n");
11:          }
12:          else {
13:             printf("努力しましたね、");
14:             printf("%d 点良くなっています¥n", ten2-ten1);
15:          }
16:      }

```

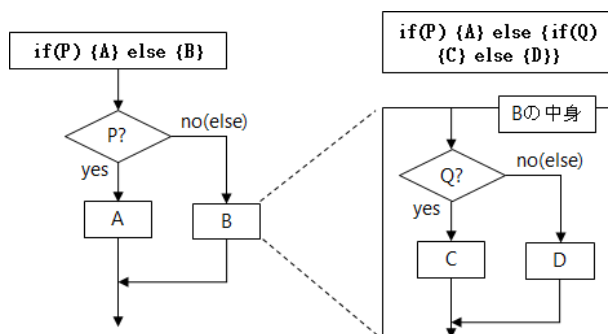
実習 9 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (コードブロック) 4 行目の最後の「{」と 7 行目の「}」に囲まれている 2 つの printf 文は、4 行目の if 文で「ten2 < ten1」という条件が成り立っているときに実行したい命令です。このように、if 文の条件式の直後に「{」が書かれている場合は、条件が成り立っている場合、それとペアになっている「}」までの間の実行文をすべて実行する、という決まりがあります。「{」で囲まれた実行文群は**コードブロック**と呼ばれます。同じように、else の直後にコードブロックが書かれている場合は、条件が成り立たなかった場合に、そのすべてを実行すると約束されています。この場合、9 行目から 15 行目がそれにあたります。したがって、「ten2 < ten1」が成り立っていないとき、すなわち「ten2 >= ten1」のとき、9 行目が実行され、「ten2 == ten1」ならば 10 行目が、「ten2 > ten1」ならば 13 行目と 14 行目が実行されることになります。9 行目から 15 行目全体が if...else 構文になっていますが、このように、コードブロックの中には何を書いても自由です。if 構文を入れ子にすることによって複雑な分岐も表現することができます。プログラムの流れを図にしたのが次の図です。

- 2. 条件判定の一般的な構文は次のように書けます。

if(条件式 P) { 実行文群 A }

あるいは



```
if(条件式 P) { 実行文群 A } else { 実行文群 B }
```

「実行文群」は単独の文でも `{ }` で囲まれたコードブロック（複文）でも構いません。単独の文の場合でも、ブロック構造をはっきりさせるために `{ }` で囲んでおくことは良い習慣です。上の実習例では 9 行目の `if` から 15 行目までが「実行文群 B」に該当しますが、「実行文群 B」自身もまた「`if...else`」構文になっています。16 行目の `else` は、「`else if (ten2 > ten1)`」の意味ですが、それ以外の場合は処理がすでに終わっていて、ここには回ってこないで、改めて書く必要はありません。

- 3. (論理演算子) 「もし $x < y < z$ ならば」とか、「もし $|x - y| > 1$ ならば」というように条件がちょっと複雑になると「かつ」「または」「ではない (否定)」という論理記号

<code>&&</code>	かつ (and)
<code> </code>	または (or)
<code>!</code>	ではない (not)

を使って、条件式を組み合わせることが必要になります。これらは**論理演算子**と呼ばれます。例えば、「 $x < y < z$ 」は「 $x < y$ かつ $y < z$ 」と等しいので、「 $x < y \&\& y < z$ 」、「 $|x - y| > 1$ 」は「 $x - y > 1$ または $x - y < -1$ 」と等しいので、「 $x - y > 1 \ || \ x - y < -1$ 」とします。「`|`」は Shift キーを押しながらキーボード右上の「`≠`」キーを押してください。「`&&`」は「`||`」に優先します。「`!`」は否定の記号ですが、混乱を避けるために、なるべく肯定表現を使う方が間違いが少ないでしょう。例えば、「 x がマイナスでなければ」という条件は言葉通り書けば「`if(!(x<0))`」となりますが、ちょっとあたまを使えば「`if(x>=0)`」と同じことが分かります、その方がわかりやすいでしょう？

これらの論理演算子を組み合わせればどんな複雑な条件式でも表現可能ですが、記述する際は誤解を避けるために、なるべく括弧を多用して「`( )`」だけしか使えま

せんが)、優先順位を明らかにするのは良い習慣です。

練習問題 3.1 次の条件を C の文法を使って表現しなさい。(1) $x = 0$ または 1 ($x=0||1$ ではない)、(2) $z \neq 0$ 、(3) $|a| > 1$ 、(4) $a \neq 0$ かつ $b \neq 0$ 、(5) $0 \leq x < 2$ 、(6) $x - y = \pm 1$ 、(7) $x = y = z$ 、(8) $x > y > z$ 、(9) $x > \max\{y, z\}$ 、(10) x と y は 0 ではなく、異符号を持つ。

練習問題 3.2 x はもし負ならば 0、さもなければ x のまま、つまり、 $\max\{x, 0\}$ を新たな x とする、ということを条件式を使って表しなさい。

練習問題 3.3 試験の点数を入力し、点数が 90 点以上は A 評価、80 点以上 90 点未満は B 評価、70 点以上 80 点未満は C 評価、60 点以上 70 点未満は D 評価、60 点未満は F 評価と表示するプログラムを書きなさい。

練習問題 3.4 double 型の数 a, b を入力させ、一次方程式 $ax + b = 0$ を解くプログラムを書きなさい (いつでも解が求められるのでしたっけ?)。

練習問題 3.5 西暦年号で 4 で割れる年はうるう年と決められていますが、例外があって、100 で割れる年はうるう年になりません。それにも例外があって 400 で割り切れる年はうるう年になります。したがって 2000 年はうるう年でした。2100 年はうるう年ではありません。西暦年号を入力して、その年がうるう年かどうかを判定するプログラムを書きなさい。

3.1.1 switch 構文

条件が int 型数の値によって決まるような場合、switch 文を使うとプログラムがすっきりします。

実習 10 (1) 次のプログラムを入力して実行しなさい。(2) 6 行目を削除して実行させ、0 を入力してごらんください。

プログラム例

```
// 多項分岐 (switch)
1:     int k;
2:     printf("あなたは家で 1 日平均何時間くらい勉強しますか...");
3:     scanf("%d", &k);
4:     switch (k) {
5:     case 0: printf("それはあんまりだ! (k=%d)¥n", k);
6:         break;
7:     case 1:
```

```
8:      case 2: printf("継続は力なり！ (k=%d)¥n", k);
9:          break;
10:     default: printf("運動もしていますか？ (k=%d)¥n", k);
11:     }
```

実習 10 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (switch) 条件が、このように int 型数の値によって決まる場合、switch 文を使うと、if ... else if ... の繰り返しを避けることが出来ます。4 行目の k という変数は int 型でなければいけません、その k の内容によって、case xx と書かれたところにジャンプします。例えば、k=1 の場合は 5, 6 行目をスキップして、「case 1:」と書かれた 7 行目から開始します。5, 7, 8, 10 行目のコロン (:) の前にある文字列はラベルと呼ばれていて、「しおり、見出し」のような役割をします。
- 2. (break) k=0 の場合、5 行目に飛んで printf を実行すると、次の 6 行目に進み break という文字列にぶつかります。break 命令は、現在の switch 構文を脱出せよ、という意味を持ちます。したがって、11 行目までをスキップして 11 行目の次にある命令文実行に移ります。もし、6 行目の break 命令文がない場合は、5 行目のあと順番通り続けて最初の命令文がある 8 行目を実行しますので、おかしいことになります。
- 3. (default) 10 行目の「default」というラベルは、k の値に等しい case ラベルが見つからなかった場合にジャンプする場所です。何もする必要がなければ書く必要はありません。この場合は、k=0, 1, 2 以外の場合、すなわち 3 時間以上勉強する人のためのメッセージを表示するために使われています（ここを選ぶ人がたくさんいることを期待したい）。

練習問題 3.6 実習 10 のプログラムを、switch を使わずに、if ... else 構文で書き直しなさい。

3.2 フィードバック構造 (while, do ... while)

前節の条件分岐構造とこの節で解説するフィードバック構造がプログラムの骨格を作る二大要素で、これを組み合わせると、プログラミングの世界は無限に広がります。というか、ほとんどの計算はこの二つの組み合わせで表現されます。

文法予備知識

1. ある条件を満たしている限り、命令文(達)を実行させたい場合は「while(条件式) {」と「}」で命令文群を囲む。
2. 先ず命令文群を実行し、その結果がある条件を満たしている限り、その命令文群の実行を繰り返したい場合は「do {」と「} while(条件式);」で命令文群を囲む。
3. 命令文群の途中で脱出したい場合は「if... break」を使う。
4. 命令文群の途中から先をスキップして条件判定に飛びたい場合は「if ... continue」を使う。

3.2.1 while 構文

条件が成り立っている間は同じプログラムをひたすら繰り返す、というフィードバックループは while 構文で実現されます。実習 4 に出てきた while(1) に条件付きジャンプの要素を取り入れたものです。

実習 11 次のプログラムを入力して実行しなさい。

プログラム例

```
// 条件停止繰り返し計算 (while)
1:      int ten, omake;
2:      while(1) {
3:          printf("試験成績を入力しなさい ");
4:          scanf("%d", &ten);
5:          omake = 0;
6:          while(ten + omake < 60) {
7:              printf("再試験成績を入力しなさい ");
8:              scanf("%d", &ten);
9:              omake += 10;
10:         }
```

```

11:         if(omake > 0) printf("%d 回でようやく",omake/10+1);
12:         printf("合格です。¥n");
13:     }

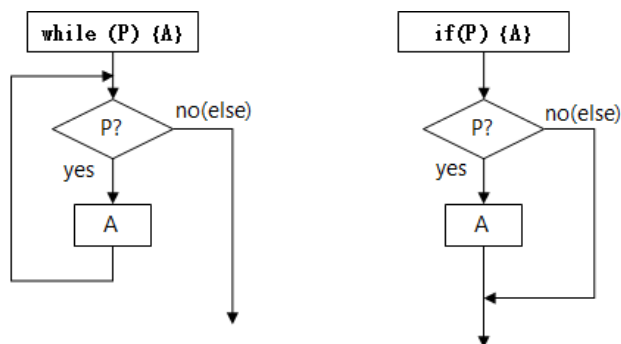
```

実習 11 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (while) 6 行目から 10 行目が新たに出来た条件付きの while 構文です。「ten+omake < 60」という条件が成り立っている間このフィードバックループを繰り返す、という命令を表します。while ループは次のような構造をしています。

while(条件式 P) { 実行文群 A }

「条件式」には判定すべき条件を指定する式、「実行文群 A」は命令文を並べたコードブロックです。英語から想像されるとおり、「条件式 P」が成り立つ間は、「実行文群 A」を実行しなさい」という命令を表します。先ず「条件式」をチェックし、それが「真（正しい）」の場合に限り「実行文群」を実行します。「実行文群」を実行し終わったら最初に戻り「条件式」をチェックします。「実行文群」の実行結果によって「条件式」の内容が変わり、それが「偽（間違い）」になったとき、この構文から抜け出します。流れを図で表すと、次のようになります。if 構文と対比させて理解なさい。



- 2. プログラムの動きを見てみましょう。最初 6 行目に来たとき、カッコの中の条件式がチェックされます。この場合は入力された点数が 60 点に達しているかどうかを判定します。60 点以上ならばループをスキップしますが、さもなければ再試験を受け、再試験一回毎に合格点を下げる、という評価基準にしたがい、「げた」を計算して再び 6 行目の条件式チェックに戻ります。最初の点が合格点ならば 7 行目から 9 行目は一回も実行されません。

- 3. (条件式の解釈) 2行目の「while(1){ ... }」は、すでに実習4で経験済みです。while 構文と見比べると、「1」が条件式を表していることになりますが、「1」が成り立つとはどういう意味でしょうか？ これは次のように解釈されます。コンピュータは条件式（論理式）を見るとそれが正しければ「真 true」、間違っていれば「偽 false」という論理値に置き換え、false ならば 0、true ならば 0 以外の数（例えば 1）に置き換えます。例えば、 $x = 1, y = 2, z = 1$ としたとき、「 $(x < y) \&\& (y < z)$ 」という条件式は $x < y$ が「真」、 $y < z$ が「偽」「真&&偽」は「偽」したがって「 $(x < y) \&\& (y < z)$ 」は「偽」で「0」という具合です。ということは、条件式の代わりに数値を書いても、それが 0 ならば「偽」、0 以外ならば「真」と解釈してくれるに違いない。ということは while(1) と書けば、常に条件成立となるはず、というわけです。したがって、「while(1) { ... }」あるいは「do{ ... } while(1);」と書けば、いつも条件は成立しているので { ... } の中身を無限に繰り返すことになります（無限ループと言います）。無限ループは、この例のように、いろいろなデータを使ってプログラムの動きを確かめたい場合、一々ビルドをしなくて済むので、積極的に利用しましょう。
- 4. (代入演算子) 9行目の記号「+=」はC独特のもので、「omake=omake+10」と入力したのと同じです。ある変数に新しいデータを累積する、あるいは除いていく、という計算は非常にしばしば出てきます。同じ変数を2度書くのは面倒くさい、入力ミスをする機会が増える、ということから、9行目のような記法が生まれました。「a -= b;」は「a = a - b;」と書いたのと同じです。さらに「a *= 2;」は「a = a*2;」つまり元の数値を2倍する、ということを表し、「b /= 10;」は「b = b/10;」つまり、元の数値を10で割る、ということを表しています。これらを併せて代入演算子と言います。

3.2.2 do ... while 構文

while 構文でよく間違えるのは、条件判定に使われる変数の値が確定しないままにこの命令を実行してしまうことです。最初にチェックする条件がない場合、とりえず実行文を実行し、その結果を見てから条件チェックをする、という構文が「do ... while」構文です。

実習 12 次のプログラムは、金利を入力して、複利計算で元利合計が2倍になるまでの経過を計算するものです。入力して実行させ、どういう状態で計算を終了したか観察なさい。

プログラム例

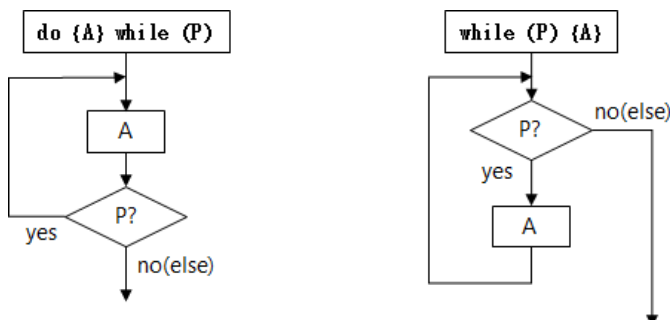
```
// 条件停止繰り返し計算 (do while)
1:   int year;
2:   double rate, sum;
3:   printf("複利計算します。金利は？ (%% 入力) ");
4:   scanf("%lf", &rate);
5:   year = 0;
6:   sum = 1;
7:   do {
8:       year = year + 1;
9:       sum *= 1 + rate / 100;
10:      printf("%d 年後の元利合計は %.2lf 円¥n", year, sum);
11:  } while(sum < 2);
12:  printf("倍増するのは %d 年後¥n", year);
```

実習 12 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (do...while) while(...) {...} と同じようなフィードバック構造を持った構文として、do {...} while(...) 構文があります。その構造は次の通り。

do{ 実行文群 A } while(条件式 P) ;

「while ...」の場合とは実行文群と条件式の位置がひっくり返っています。フローチャートで描くと次のようになります。while 構文と対比させてその違いを理解しなさい。



- 2. まず、「実行文群 A」を実行し、「条件式 P」が満たされるかどうか調べ、満たされていれば (while)、また「実行文群 A」にもどる。さもなければループ終了です。このように、while ループと違い、とりあえず 1 回ループの中身を計算して、その

結果を見て判定する、というのが「do...while ループ」です。

- 3. 実習のプログラムでは、とりあえず、7行目を無視して10行目まで実行します。その後、11行目の条件判定で `sum` が2未満ならば8行目に戻って再度実行します。8行目から10行目までを何回か繰り返して `sum` が2を超えたら12行目を表示しておしまいになります。
- 4. 3行目 `printf` 文の書式文字列中にある「`%`」は「`%`」を文字として表示したい場合に使う書式です。「`%`」単独で使うと、VC2010は `%d` あるいは `%lf` が来るものと期待しますが、その期待が満たされないので、エラーになります。

3.2.3 while ... if - break (continue) 構文

`while` 構文はループの最初で条件を判定し、`do ... while` 構文はループの最後で判定する、という規則でしたが、ループの途中で判定条件が判明し、最後に判定したのでは遅すぎる、という場合はどうすれば良いのでしょうか。その場合は、実習10の解説2項にヒントがあります。ループの途中に条件文を挿入し、条件が整ったら `break` するようにすれば良いのです。

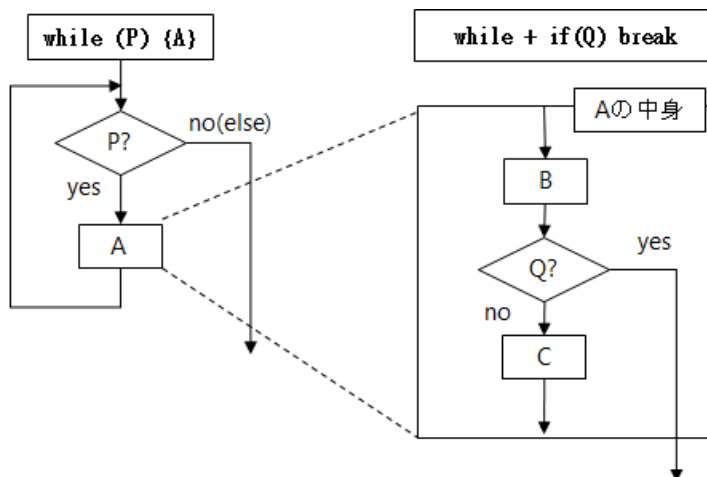
実習 13 次のプログラムを入力して実行し、どういう場合にループを脱出するか調べなさい。

プログラム例

```
// 条件停止繰り返し計算 (while if break)
1:   int pocket, price;
2:   printf("所持金を入力しなさい：");
3:   scanf("%d", &pocket);
4:   while(pocket > 0) {
5:       printf("買い物金額を入力しなさい：");
6:       scanf("%d", &price);
7:       if(price > pocket) break;
8:       pocket -= price;
9:       printf("残額は %d 円¥n", pocket);
10:  }
11:  if(pocket == 0) {
12:      printf("残額ゼロ！¥n");
13:  } else {
14:      printf("%d 円足りません！¥n", price-pocket);
15:  }
```

実習 13 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (**while ... if break**) **break** は実習 10 の **switch** 構文のところで使われましたが、それは「**switch** 構文から脱出する」という目的で使われていました。**while** ループの中で使われる場合は「**while** ループを脱出する」という命令になります。プログラムの流れを図に表すと次のようになるでしょう。



- もし、実行文 **B** がない場合はループの最初に **if** 文が実行されるので、通常の **while** 構文と同じ、実行文 **C** がないと、ループの最後に **if** 文が実行されるので、(最初に 1 回だけ判定のある) **do-while** 構文と同じになります。
- 2. 実習のプログラムでは、4 行目の条件判定で、最初の所持金がゼロあるいはマイナスならば、何もしないで 11 行目へ飛びます。さもなければショッピングに出かけ、買い物金額を入力し、7 行目でその金額と所持金額を比較するという条件判定を実施し、所持金が足りなければループを脱出して 11 行目へ飛びます。さもなければ支払って所持金を計算し直し、買い物を続けます。いずれにしてもループを脱出して最初に実行するのは 11 行目です。このとき、変数 **pocket** を見ればどちらから脱出してきたかが分かるので、それを使って、状況に応じた表示を選択することができるのです。
- 3. (**if - continue 構文**) **break** はループを抜けるという意味ですが、条件を満たしたらループを抜ける代わりに、残りの仕事をスキップしてループする、という処理の仕方もあります。上の図で言うと、「条件 **Q** を満たしていたら **C** を実行しない (しかしループは抜けない)」という動きをさせたい場合です。その場合、**break** の代わりに **continue** を使うという構文があります。逆に言えば、「条件 **Q** が成り立たなければ **C** を実行する」という条件文と同じことですから、**continue** を使わなくても「**if (!Q) {C}**」と書けばよいので、新たな知識は必要ありません。しかし、

実行する内容 (C) が多い場合は、`continue` 文を使って書いた方が見やすくなります。

練習問題 3.7 キーボードから順に数を入力し、0 が入力されたら、前回の 0 からそれまでに入力された数の合計を表示する、というプログラムを書きなさい。例えば、1, 4, -1, 0, 3, 2, 0, 5, -5, 0 という順に数が入力されたら、「1, 4, -1, 0」が入力された後に「4」、「3, 2, 0」が入力された後に「5」、「5, -5, 0」が入力された後に「0」が表示されるように。

ヒント：いつ次の 0 が入力されるか分かりませんから、入力される毎に累計を計算しておかなければいけません。そのために一時記憶として代入文が必要です。0 が入力されたら、それまでの累計をゼロクリアすることを忘れずに。

練習問題 3.8 「キーボードから `double` 型数を二つ (a, b とする) 入力し、 $a > b$ ならば「 a は b より大きい」と表示し (もちろん a, b は入力された数値で表示)、 $a < b$ ならば「 b は a より大きい」と表示し、等しければ「 a と b は等しい」と表示する」、ということを $a = b$ となるまで繰り返す、というプログラムを書きなさい。

3.3 擬似乱数 (rand())

サイコロを振って出る目を記録したとしましょう。それは1から6までの6個の数字がランダムに並んでいて、記録の一部から「次の」数を予測することはできません。このような数列は**乱数列**と呼ばれています。乱数は先の読めない展開の必要なゲームのプログラムでも多用されますが、12章で学ぶように、経営システムの問題解決においても必要とされる重要な道具となっていますので、ここで解説しておきます。

Cの関数として、この乱数を「計算する」`rand`があります。サイコロを振った時に出る目の数は物理現象ですから、正確には乱数を計算することは不可能です。しかし、計算した数がプログラムを書いた人以外には予想もつかないものであれば、それを乱数に代わるものとして受け入れることは許されるかもしれません。本当の乱数ではないという意味で、正確には**擬似乱数**というべきですが、コンピュータの世界ではそれ以外の乱数は出てこないで、ただ乱数と言うことにします。

`rand` は0以上 `RAND_MAX` 以下ということだけが分かっている `int` 型数を生成します。`RAND_MAX` はシステムが決めた定数です。次の実習でその関数の振る舞いを調べてみましょう。

実習 14 (1) 次のプログラムを実行させて、その結果を記録しなさい。「... の勝ち」と表示されたら、「0」を入力しなさい。5通りの結果を書いたら、プログラムを強制終了させなさい（「閉じる」ボタンをクリック）。**(2)** 同じプログラムを何回か実行させて、その結果を今記録したものと比較しなさい。

プログラム例

```
// ライブラリー関数 (擬似乱数)
// #include <math.h> が必要です
1:     int coin;
2:     printf("RAND_MAX = %d = %lf¥n", RAND_MAX, pow(2.0,15)-1);
3:     while(1) {
4:         coin = rand();
5:         if(coin < RAND_MAX/2) printf("あなたの勝ち¥n");
6:         else printf("私の勝ち¥n");
7:         scanf("%d", &coin);
8:     }
```

実習 14 の解説 チェックボックスに ✓ を入れながら読みなさい。

□ 1. (**rand 関数**) `rand()` は**乱数** (random number、正確には**擬似乱数**) を生成する

関数です。rand 関数を使うためには `stdlib.h` というヘッダファイルを `include` する必要があります。このテキストでは標準のプログラムとしてすでに `include` 済みですので、新たには何もする必要がありません。

- 2. (**マクロ記号定数**) 関数 `rand` は引数を持ちません。`rand()` を実行させると 0 以上 `RAND_MAX` 以下の `int` 型数を生成します。2 行目から分かるように、`RAND_MAX` は $2^{15} - 1 = 32767$ と定義されています。`RAND_MAX` は**マクロ記号定数**と呼ばれ、「`RAND_MAX` という文字が出てきたらそれを 32767 に置き換えること」と決められています。`stdlib.h` というヘッダファイルの中でそのことが定義されています。
- 3. (**べき乗関数pow**) 2 行目の `pow` はべき乗を計算する数学関数で、この場合は $2^{15} = 32768$ を計算しています。C では、Excel のように「`a^b`」という記号は使えないので、この関数を使って計算しなければいけません。しかし、「`^`」は別の意味として定義されているので、エラーにはなりません。うっかり使っても気が付きにくいので注意下さい。
- 4. 5 行目の条件式は、もし `rand` 関数が本当にさいころのようなでたらめな数を生成するのであれば、平均的に 2 回に 1 回は正しいので、「勝ち」と「負け」がでたらめな順番で表示されても良いのですが、実習 (2) で分かったことは、勝負の結果はいつも「勝ち」「負け」「勝ち」「負け」「負け」という同じパターンになることです。予想が付かないのは一回目だけ、というようになりますが、これでは賭になりませんね。どういことでしょうか？
- 5. `rand()` は、でたらめ「らしく」に並んでいる数列 $r_1, r_2, \dots$ を順番に取り出すように作られたプログラムなのです。ビルドした後、最初に `rand` が呼ばれたときは r_1 を関数値とする、という約束があります。したがって、ビルドしなおすと、何回やっても結果は同じなのです。いったん結果が分かってしまうと、あとは乱数とはいえなくなります。

原因が分かれば、その欠点を回避し、本当の賭に近づける工夫を考えることができます。

実習 15 (1) 実習 14 のプログラムの 1 行目の後に、次の 4 行を挿入して実行下さい。(2) 複数回実行し、入力した数と表示される結果の関係を調べ下さい。(3) `seed` に同じ数を入力した場合にどうなるかを調べ下さい。

プログラム例

```
unsigned int seed;
printf("なにか数を入力しなさい ");
scanf("%d", &seed);
srand(seed);
```

実習 15 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (unsigned int 型) sign は符号のことです。unsigned int 型は 0 から $2^{32} - 1$ までの正の数のみを扱うことが出来る int 型のことですが、詳細は索引で調べてください。
- 2. (**乱数の種** (たね) と srand) rand は引数を持ちませんが、生成する数列 $\{r_i\}$ は **種 (seed)** と呼ばれる数を元にして生成されています。VC2010 の種の既定値は決まっているので、ビルドの後はいつも r_1 から取り出されます。種を変えることによって数列 $\{r_i\}$ の取り出す位置を変えることが出来ます。種を変える関数が srand です。srand の引数に適当な正の整数を指定することで種を変えることが出来ます。逆に、同じ種を入力すれば常に同じ「乱数」を生成することができます (実習 (3))。やたらに変えても意味ないので、srand 関数を呼び出すのは、main プログラムで変数を宣言した直後に一回だけ実行させるようにしなさい。
- 3. (**time.h**) プログラムを実行するたびに種を入力するのは煩わしいし、同じ数を入力してしまうかもしれません。そこで、プログラムで seed を自動生成する方法があります。それは VC2010 の持っている体内時計を使うことです。time(NULL) という関数を呼ぶと、1970 年からの通算秒数 (現在 13 億秒程度です) を計算して返してきますから、それを利用するのです。

```
srand(time(NULL)*12345);
```

という一行を main プログラムの開始直後に挿入しなさい。time(NULL) は絶えず更新されますから、ビルドするたびに別の数となり、rand() 関数の数列の開始位置が変わります。ただ、実習で試してもらったように、その違いがわずかな場合は開始位置が同じになる危険性があるので、「*12345」のような演算をして 1 秒の違いを増幅させておく方が安全です。time() 関数を使うためには、time.h というヘッダファイルをインクルードする必要があります。

練習問題 3.9 「 $6 * \text{rand}() / (\text{RAND_MAX} + 1)$ 」とすると、int 型の切り捨て計算によって 0 から 5 までの整数値が生成できます。それに 1 を足せば、サイコロの目になるでしょう。サイコロを 100 回振って出た目を記録したときに得られるような、1 から 6 までの数がランダムに並ぶ長さ 100 の数列を表示するプログラムを書きなさい。

練習問題 3.10 「 $2 * \text{rand}() / (\text{RAND_MAX} + 1)$ 」とすると、0 か 1 の数になります。0 を「裏」、1 を「表」とすればコイン投げの代わりに使えそうです。あるいはまた、それを 2 倍して 1 を引くと ± 1 の数が生成され、+1 を勝ち、-1 を負けと考えれば、対等な賭けの記録として使えそうです。10 回賭けをして勝ち負けの記録と、通算成績 (勝ち越し、負け越し?) を計算するプログラムを書きなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F評価の項目について猛勉強しなさい。

- ☐ ☐ if 構文、if ... else 構文
- ☐ ☐ 条件式、関係演算子「>,<,>=,<=,==,!=」、論理演算子「||,&&,!」
- ☐ ☐ switch 構文、case 文
- ☐ ☐ while 構文
- ☐ ☐ do ... while 構文
- ☐ ☐ 代入演算子、「+=,-=,*=,/=」
- ☐ ☐ if ... break 命令文
- ☐ ☐ if ... continue 命令文
- ☐ ☐ 擬似乱数、rand 関数

3.4 章末演習問題

問題 3.1 2 次方程式 $ax^2 + bx + c = 0$ の解を計算するプログラムを書きなさい。

ヒント： $a = 0$ の場合は 1 次方程式、不定か不能の可能性があるので、 $b = 0$ かどうかをチェックする必要があります。2 次方程式の場合は判別式の符号を判定する必要があります。虚数解？

問題 3.2 (suica あるいは pasmo (JR、私鉄、バスのプリペイドカード) の自動改札精算用プログラム) 金額 (カードの残額) と運賃を入力し、**(1)** 足りなければ「料金不足です、精算してください」と表示し、**(2)** カード残額が 1000 円以上あり、運賃を引いた残りが 1000 円未満になるならば、運賃を引いた残額とともに「1000 円未満になりました」と表示する、プログラムを書きなさい。繰り返し入力できるようにしなさい。

問題 3.3 ジュースの自動販売機の釣り銭 (金種と枚数) を計算するプログラムを書きなさい。受け付ける金種は 1000 円札、500 円玉、100 円玉、50 円玉、10 円玉とし、商品価格は 120 円と 150 円の 2 種類とします。入力の仕方も考えなさい。

問題 3.4 成田空港駐車場の料金は次のようになっています。3 時間 30 分までは 30 分ごとに 260 円、3 時間 30 分を超えて 24 時間まで 2060 円、24 時間を超える 120 時間までは 24 時間ごとに 2060 円、120 時間を超える 24 時間ごとに 520 円です。入庫した日とその時刻、現在の日時を入力して駐車料金を表示するプログラムを書きなさい。但し、入庫した日と出庫した日は同じ月に属するものとします。

問題 3.5 **(1)** 西暦年月日を入力して、その日の曜日表示するプログラムを書きなさい。あなたの誕生日は何曜日？ フランス革命 (1789 年 7 月 14 日) は何曜日でしたか？ アメリカの独立 (1776 年 7 月 4 日) は何曜日でしたか？ **(2)** 西暦年と月を入力して、その月のカレンダーを表示するプログラムを書きなさい。閏年の判定法は練習問題 3.5 にあります。

ヒント：基準日からの通算日数を計算し (ちょっと大変)、それを 7 で割ったあまりを計算すればよい。例えば 1600 年 1 月 1 日は土曜日です。曜日を調べるだけならば、1 年ごとに曜日は 1 日か 2 日 (閏年の場合) ずれることが分かっているので、通算の年数と閏年の回数、その年の月日の年間通算日を合計したものを 7 で割ったあまりで求めることもできます。

息抜きのページ

次のプログラムを実行して、表示にしたがってデータを入力しなさい。

プログラム例

```
// 数当てゲーム（百五減算）
#include <stdio.h>
int main() {
    int n, r3, r5, r7;
    while(1) {
        printf("2桁の整数を心に思ってください");
        printf("少し暗算してもらいます、よろしいですか ¥n");
        printf("その数を3で割ったあまりはいくつですか？ ");
        scanf("%d", &r3);
        printf("その数を5で割ったあまりはいくつですか？ ");
        scanf("%d", &r5);
        printf("その数を7で割ったあまりはいくつですか？ ");
        scanf("%d", &r7);
        n = 35*(3-r3) + 21*r5 + 15*r7;
        printf("あなたの思った数は %d ですね？ ¥n¥n", n%105);
    }
}
```

第 4 章

算法 1：数論

コンピュータは問題を解くために使う道具ですが、使うためにはコンピュータとの対話が成立しなければ困ります。2,3 章で学んだのは、人間がコンピュータを扱う際に使用する言語の書き方の規則、すなわち文法、でした。これでコンピュータに問題を理解させることはできるようになりましたが、問題を解けるようになるには、コンピュータにその解き方を教えないとその先へ進めません。問題を解く手順を与えるのが**算法**、あるいは**アルゴリズム**です（計「算」作「法」）。

この章では、最初にアルゴリズムの書き方について解説した後、基本的な算法をいくつか紹介し、その書き方を実習します。この章で学ぶのは、次のような事項です。

- 約数、素数、素因数分解
- 公約数、最大公約数、ユークリッドの互除法、拡張ユークリッドの互除法
- 漸化式

4.1 アルゴリズム（算法）

コンピュータに問題を解かせる場合は、どんな簡単な問題でも、その解く手順をコンピュータが理解できるようにいちいち段階を追って記述することが必要になります。**問題を解く手順**のことを**アルゴリズム**と言います。もう少し正確に言うと、その記述にしたがって作業したとき、**有限の時間で問題に解があるかないかが分かり、解がある場合は、その解を具体的に求めることが出来る手順のことをアルゴリズムと言います。**

アルゴリズムの書き方には、C のプログラムと違って厳密な規則はありません。それにしたがって作業をすれば問題が解ける、という最低条件を満たしていれば、自由に書いて構わないのですが、そうはいつでも、経験的に「良い」アルゴリズムを書くための指針というものがあるので、それについて解説します。

アルゴリズムを実際を書く前にすべきことがあります。それは、いうまでもなく、問

題の解き方を知ることです。与えられた問題の解き方を調べるときに実際に行うであろう作業手順が参考になります。

問題の解き方を考える場合は、**分析（分解）と統合**ということの重要性が強調されます。問題を部分問題に分解し、それらの解を統合すれば全体の解に到達しやすい、ということを表したものです。その過程を実践する際に有効なのは、3章で示した**フローチャート**のように、思考の流れを図示する道具（**アイディアプロセッサ**という名前で呼ばれることも多い）や、本の見出し（章、節、項）のように、アイディアをその重要度に応じて**階層化**して書く**アウトラインプロセッサ**などで、それらを活用すると作業が効率よく進みます。xx プロセッサと言っても特別な道具は不要で、構造を意識して計算手順を書いてもらなさい、という程度のことです。フローチャートもこのテキストにある程度の簡易的なもので構いません。

アウトラインの例として、2次方程式 $ax^2 + bx + c = 0$ を解く場合を考えましょう。「2次」方程式といっても $a \neq 0$ とは限らないので、 $a = 0$ の場合は別に考えなければいけません。 $a = 0$ の場合、今度は $b = 0$ かそうでないかによって扱いが変わってきます。 $b = 0$ の場合は $c = 0$ かそうでないかによって変わってきます。というわけで、最初は x^2 の係数が0かそうでないかを調べ、 $a \neq 0$ になって初めて判別式が登場します。

これを階層的箇条書き（アウトライン）にすると、次のようになるでしょう。

- $a \neq 0$ ならば
 - 2次方程式を解く
- $a = 0$ ならば
 - $b \neq 0$ ならば、1次方程式を解く
 - $b = 0$ ならば、不定か不能 ($c = 0$ か否かに依存)

このように、部分作業ごとに字下げし、作業内容をメモしておく、上のレベルだけ読んだ場合に、全体として何をしているか一目で理解できるようになります。

これで細部を埋めればアルゴリズムの完成です。アルゴリズムは、**箇条書き**で、上から順番に読みます。箇条書きは数字の通し番号（字下げされると、枝番が付きます）とし、各項を**ステップ**と言います。何も指示がなければ次の行（ステップ）に進みます。途中で「アルゴリズム終了」という指示があれば、そこでおしまいです。また、最後の行（ステップ）にフィードバックの指示がなければ、そこで終了します。

1. $a = 0$ ならば (1 次以下の方程式なので) ステップ 3 へ。
2. (2 次方程式の場合) $D = b^2 - 4ac$ として、
 $D = 0$ ならばステップ 2.2 へ、 $D < 0$ ならばステップ 2.3 へ。
 - 2.1 ($D > 0$ の場合) 異なる 2 実解 $\frac{-b+\sqrt{D}}{2a}$, $\frac{-b-\sqrt{D}}{2a}$ が答え、アルゴリズム終了。
 - 2.2 ($D = 0$ の場合) 重解で $-\frac{b}{2a}$ が答え、アルゴリズム終了。
 - 2.3 ($D < 0$ の場合) 共役複素解 $\frac{-b+\sqrt{-D}i}{2a}$, $\frac{-b-\sqrt{-D}i}{2a}$ が答え、アルゴリズム終了。
3. (一次以下の方程式) $b \neq 0$ ならば $-\frac{c}{b}$ が答え、アルゴリズム終了。
4. ($b = 0$ の場合) $c = 0$ ならば「不定」、さもなければ「不能」が答え。

ステップ 1 で $a \neq 0$ に対する指示はありませんからステップ 2 へ進みます。ステップ 2 で $D > 0$ に対する指示はありませんからステップ 2.1 へ進みます。ステップ 4 では次の行き先の指示がありませんので、そこでアルゴリズム終了です。

アルゴリズムが「良い」「悪い」という場合の基準はいろいろあると思いますが、その基本にあるのは「読みやすさ」です。読みにくさの要因はいろいろありますが、主な要因の一つは「枝分かれ」と「フィードバック」によって思考の流れが乱される、ということでしょう。「枝分かれ」すなわち、条件分岐では条件を満たす場合に実行する範囲を明確にすること、条件が満たされなかった場合にどこへ分岐するのかが分かること、が重要です。C のプログラムの字下げを参考にして記述を工夫しなさい。

4.1.1 アルゴリズムの検証

アルゴリズムは書いたらおしまい、ではありません。アルゴリズムが意図したとおりに動くかどうか調べ、いくつかの具体例で正しい答えが導出できることを確認しなければいけません。この確認作業を怠ると、プログラミング作業でデバッグに苦しむことになってしまいます。確認の作業でもやはり階層化の考え方が重要です。まず、大方針が間違っていないかを調べ、それが済んだら各部分作業の方針が間違っていないか、細部の扱いが間違っていないか、というように、各段階ごとに、確認することで、効率的な確認作業が出来るようになります。

練習問題 4.1 西暦年月日が与えられたとき、その曜日を求める際のアイディアをまとめるためにフローチャートを使って作業しなさい。その結果をもとに、アルゴリズムを書きなさい (3 章末演習問題参照)。

練習問題 4.2 自動販売機のつり銭を計算するアルゴリズムを書きなさい (3 章末演習問題参照)。

4.2 約数

アルゴリズムの最初の例として、約数を取り上げます。ここでは数と言えば正の整数のことを指すことにします。ある整数 a の**約数**とは、その数を割り切ることができる整数のことです。言い換えれば、整数 b が整数 a の約数であるとは、 a/b が整数になることを言います。例えば、 $a = 12$ ならば「1, 2, 3, 4, 6, 12」が約数です。2 以上のどんな数 a でも ($a/1$ と a/a は整数になりますから) 1 と a という二つの約数 (**自明な約数**と言います) がありますから、興味のあるのは、それ以外の約数です。

約数を求めるには、2 から順番に実際に割ってみて割り切れるかどうかを確かめるしかありません。自明な約数以外では最大の約数の候補は $a/2$ ですから、2 から $a/2$ まで確かめれば十分ですが、 b が a の約数ならば $c = a/b$ も約数になります。したがって、2 から順に調べていった場合、 $b(> \sqrt{a})$ が約数ならば、それは $c = a/b(< \sqrt{a})$ という約数が見つかったとき、すでに分かっていたことになります。というわけで、2 から $\sqrt{a}$ まで調べれば十分ということになります。こう考えることで、約数を全部リストアップする、次のアルゴリズムを得ることができます。

アルゴリズム 1 正の整数 a のすべての自明でない約数を見つけるアルゴリズム

1. $b = 2$ とする。
2. $b^2 \geq a$ ならばステップ 5 へ。
3. a/b が整数ならば、 b と a/b を a の約数として表示。
4. $b + 1$ を新たな b としてステップ 2 へもどる。
5. $b^2 = a$ ならば、 b を約数として表示。

ステップ 2 の判定条件はステップ 4 を繰り返すことによりいつかは満たされますから、このアルゴリズムは必ず有限回で終了します。

実習 16 このアルゴリズムを実現する `while` 構文を使った C のプログラムを書いて、 $a = 36$ と入力したとき、「2 18 3 12 4 9 6」と表示されることを確かめなさい。`while` の条件文はステップ 2 の条件式を否定したものになることに注意しなさい (`while(b*b < a) { ... }`)。そのプログラムを使って 1234567890 のすべての約数を計算しなさい。

練習問題 4.3 このアルゴリズムを $b = 2$ から $b = a/2$ まで一つずつ順番に調べ、大きさの順に表示する方法に書き直して C のプログラムを書き、実習の例と同じ $a = 1234567890$ を使って計算時間を比較しなさい。

4.3 素数

2, 3, 5, 7, 11, 13, ... などは、自明な約数、すなわち 1 とその数自身以外に約数を持ちません。2 以上で、このような性質を持つ数のことを**素数**と言います。素数でない数は**合成数**と呼ばれています。素数は数学的に面白い性質を持ち、様々な分析の対象になっていますが、実用的にも、インターネットのセキュリティ問題で重要な役割を果たしています。

ある数が素数かどうかを判定するには、自明な約数がないことを確かめればよいので、すべての約数をリストアップするアルゴリズム 1 を一部修正するだけで済みます。

アルゴリズム 2 $a \geq 2$ が素数かどうかを判定するアルゴリズム

1. $b = 2$ とする。
2. $b^2 > a$ ならば、 a は素数、アルゴリズム終了。
3. a/b が整数ならば、 a は素数でない、アルゴリズム終了。
4. $b + 1$ を新たな b としてステップ 2 へもどる。

このアルゴリズムを C のプログラムで書いてみましょう。

実習 17 (1) $n = 71$ としたとき、コンピュータを使わずに紙と鉛筆だけを使って、アルゴリズム 2 がどのように推移するのか、どういう経過をたどってどういう結論が出るのか、計算しなさい。 $n = 91$ の場合はどうですか。**(2)** アルゴリズム 2 を実現する C のプログラムを書きなさい。但し、ステップ 2 のあとに b と a/b を表示する `printf` 文を挿入して計算がどのように進行するかチェックしなさい。**(3)** 223092871 は素数ですか。901860961 は素数ですか。判定しなさい。

練習問題 4.4 7 ケタの最小の素数を見つけなさい。

このアルゴリズム 2 で $a = 83$ が素数かどうか判定する場合、 $b = 2, 3, 4, 5, 6, 7, 8, 9$ についてチェックすることになりますが、明らかに 4, 6, 8, 9 は判定する必要がないことが分かるでしょう。2 で割り切れなければ 4, 6, 8 では割り切れることはなく、3 で割り切れなければ 9 で割り切れることはないからです。2 の倍数と 3 の倍数をすべて取り除くと、2, 3 以外の素数は少なくとも $6k \pm 1$ という形式でなければいけないことになります。このことを利用すると、次のような改良アルゴリズムが得られます。

アルゴリズム 3 $a \geq 2$ が素数かどうかを判定するアルゴリズム、その 2

1. $a = 2, 3$ ならば素数、アルゴリズム終了。
2. a を 6 で割った余りが 0, 2, 3, 4 ならば a は素数ではない、アルゴリズム終了。
3. $b = 5, c = 1$ とする。

4. $b^2 > a$ ならば、 a は素数、アルゴリズム終了。
5. a/b が整数ならば、 a は素数でない、アルゴリズム終了。
6. $-c$ を新たな c 、 $b+3+c$ を新たな b として、ステップ4へもどる。

練習問題 4.5 アルゴリズム3を実現するCのプログラムを書きなさい。

練習問題 4.6 (双子の素数) 17 と 19 のように、隣同士の奇数が両方とも素数になるような素数の組を**双子の素数**と言います。10000 より大きな最小の双子の素数を見つけてください。

ヒント：奇数を $6k+i$ ($i=1,3,5$) と書くと、 $6k+3$ は3で割り切れるので、5以上の奇素数は $6k \pm 1$ の形をしています。 $6k \pm 1$ が両方とも素数になる場合が双子の素数です。 $6k-1$ が素数でなければ $6k+1$ が素数であるかどうかをチェックする必要はありません。

参考

素数についてのいくつかの話題をまとめておきます。興味のある人はインターネットで調べてみてください。

1. 素数は無限にあることは背理法で証明することができます。もし有限個しかないとしたら、それらのすべての積に1を足した数は、それらの有限個に含まれない新たな素数ということを示すことができます。というわけで、最初の仮定、素数は有限個しかない、に矛盾します。
2. (**素数定理**) x 以下の素数の個数を $\pi(x)$ とすると、十分大きな x に対して $\pi(x) \div \frac{x}{\log x}$ となることが知られています。
3. 双子の素数は無限にあると予想されていますが、まだ証明されていません。これまでに発見された最大の双子の素数は $3756801695685 \times 2^{6666669} \pm 1$ で 200700 桁の数ようです (出典 <http://mathworld.wolfram.com/TwinPrimes.html>, 2016/9/1)
4. (メルセンヌ素数) $2^p - 1$ の形をした素数を**メルセンヌ素数**と言います。 $p = 2, 3, 5, 7, 13, 17, 19, 31, \dots$ などの場合が素数であることが知られています。これらの p はすべて素数ですが、 $2^{11} - 1 = 2047$ は 23 で割り切れるので素数ではありません。素数かどうかをチェックする高速の判定法があるために、大きな素数を見つけるのに使われています。2016 年夏現在で最大のメルセンヌ素数は $p = 74,207,281$ とした場合で、10 進 22,338,618 桁です (出典 <http://www.mersenne.org/> 2016/9/1)。この p が素数かどうか、自分の作ったプログラムで確かめてごらん。

4.4 素因数分解

素数以外のどんな数でも、自明でない 2 つ以上の約数の積として表されます。特に、素数の約数を**素因数**と言いますが、素因数だけの積として表すことを**素因数分解**と言います。ある数の素因数分解を求めたいとすると、定義に従えばあらかじめ素数を求めておく必要がありますが、その必要はありません。実際、ある数 a の最小の約数 b は、約数を見つけるアルゴリズムに従い、2 から順に割り切れるまで 1 ずつ増やしてチェックすれば求まります。 b は素数のはずですから、 a を素数 b と a より小さい数 a/b との積に分解できます ($a = b \times \frac{a}{b}$)。 a/b の最小の約数は b 以上ですので (なぜでしょう?)、今度は b から順に割り切れるまで 1 ずつ増やしてチェックすれば a/b の最小 (素数の) 約数は分かります。このような計算を続けると、いつかは a/b が素数になるときが来ます。これで素因数分解は完成です。

アルゴリズム 4 $a \geq 2$ の素因数分解のアルゴリズム

1. $b = 2$ とする。
2. $b^2 > a$ ならば、 a を素因数として表示、アルゴリズム終了。
3. a/b が整数ならば、 b を素因数として表示、 a/b を新たな a としてステップ 2 へもどる。
4. $b + 1$ を新たな b としてステップ 2 へもどる。

実習 18 (1) アルゴリズム 4 を使って、コンピュータを使わずに、 $a = 132$ の素因数分解がどのように行われるか、変数表を使って調べなさい。ステップ 3 は全部で何回実行しますか? ステップ 4 は全部で何回実行しますか?

ステップ	a	b	表示
1	132	2	
2			
...			

(2) アルゴリズム 4 を実現する C のプログラムを書きなさい。結果は、例えば「 $28 = 2 \ 2 \ 7$ 」のようにしなさい。それを使って、 $a = 1234567890$ の素因数分解を実行しなさい。

ヒント: ステップ 2 の「 a を素因数として表示」はフィードバックループの外で実行するようにすれば、「`while(b*b <= a) { ... }`」という構文が使えます。また、ステップ 3 からステップ 4 を介さずにフィードバックループを続けるには `continue` 文を使います。

練習問題 4.7 a の一番小さい素因数を b としたとき、 a/b の一番小さい素因数は b 以上であることを説明しなさい (できれば証明しなさい)。

練習問題 4.8 a が素数の場合は、「... は素数です」と表示するプログラムを書きなさい。

練習問題 4.9 同じ素因数がある場合は、例えば、「 $180 = 2^2 \cdot 3^2 \cdot 5$ 」のように、Excel で使われるべき記号「^」を使ってまとめて表示するようなプログラムを書きなさい。

練習問題 4.10 アルゴリズム 4 は $b = 2, 3, 4, \dots$ が素因数になるかどうかをチェックしていますが、明らかに無駄です。せめて 4 以上の偶数はチェックしないようなアルゴリズムを書き、そのプログラムを書きなさい。

4.5 最大公約数、ユークリッド互除法

2つの整数の共通の約数を**公約数**（共通約数）と言い、公約数の中で最大のものを**最大公約数 g.c.d.**と言います。例えば、12の約数は「1,2,3,6,12」、18の約数は「1,2,3,6,9,18」ですから、12と18の公約数は1,2,3,6で、6が最大公約数です。

二つの整数 $a, b (a > b)$ の最大公約数を見つける手順を考えましょう。 a が b で割り切れるならば b が最大公約数になります。割り切れない場合、 a を b で割ったあまりを c とすると、 b と c の最大公約数は a と b の最大公約数と同じになります。なぜならば、 $c = a - b \times k$ ならば、 c は a と b の公約数で割り切れなければならないからです。

b と c の最大公約数を求める問題は、はじめの a と b の問題に比べて小さくなっているので、 b, c を新たな a, b としてこの手順を繰り返せば、いつかは a が b で割り切れるようになり、アルゴリズムは必ず終了し、最大公約数が求まります。

この考え方を手順として表したのが次のアルゴリズムです。

アルゴリズム 5 a, b の最大公約数を計算するアルゴリズム

1. a を b で割ったあまりを c とする。
2. $c = 0$ ならば b が最大公約数、アルゴリズム終了。
3. b を新たな a 、 c を新たな b としてステップ1へもどる。

上の説明では、簡単のために $a > b$ としましたが、このアルゴリズムにはそのような制約がありません。なぜならば、 $a < b$ の場合、ステップ1で $c = a$ となり、ステップ3で a と b が入れ替わり、 $a > b$ となるからです。

このように割り算のあまりを利用して最大公約数を計算する手順は**ユークリッドの互除法**と呼ばれています。

アルゴリズムを書いたら、それが正しく動くことを確認する作業が必要です。いきなりプログラムを書き始めてはいけません。アルゴリズムの確認のもっとも効果的な方法は、原始的ですが、簡単な例を使って変数がどのように動いているかを全部省略なしに調べることです。変数の動きを記録するには Excel のような表を作って作業するとよいでしょう。例えば、 $a = 34, b = 14$ としたとき、次のような経過をたどるでしょう。その結果、最大公約数が2ということが分かります。

a	b	c
34	14	6
14	6	2
6	2	0

実習 19 (1) $a = 50, b = 71$ とした場合のアルゴリズムの動きを調べるために、上のよう

な a, b, c の変数表を作り、値がどのように変わりながら最大公約数が計算されているかをチェックしなさい。(2) アルゴリズム 5 の記述に従ってユークリッドの互除法を実現する C のプログラムを書きなさい。但し、ステップ 1 の後に a, b, c を表示させる `printf` 文を挿入し、上の実習 (1) のとおりに動いていることを確かめなさい。(3) そのプログラムを使って、317811 と 121393 の最大公約数を計算しなさい。

ヒント：「`while(1) { ... }`」ループの中に、ステップ 2 を「`if(...) break;`」として挿入するのが自然でしょう。「 b が最大公約数」というのは実行文ではないので、ループを終了させたあとに、ループの外側で表示させるようにします。

練習問題 4.11 3 つの整数の最大公約数を計算するアルゴリズムを書き、C のプログラムを書きなさい。それを使って 2574, 4752, 396 の最大公約数を求めなさい。

ヒント： a, b の最大公約数を d とすると、 a, b, c の最大公約数と c, d の最大公約数は等しい。

練習問題 4.12 2 つの整数の最小公倍数 g.c.m. を計算するアルゴリズムを書き、C のプログラムを書きなさい。それを使って、82 と 34 の最小公倍数を求めなさい。

ヒント：2 数の**最小公倍数**は、それぞれの数の共通の倍数のうち、最小のものです。 a, b の最大公約数を c とすると、 $a/c, b/c$ は整数で、その最大公約数は 1 です。このとき、 a, b の最小公倍数はどのように表されるか考えなさい。

4.6 拡張ユークリッド互除法

二つの正整数 a, b は、その最大公約数が 1 のとき、**互いに素**と言います。互いに素な二つの整数 a, b に対して、

$$ax + by = 1 \quad (4.1)$$

という式を満たす整数 x, y が存在するか？ という問題を考えます。この方程式は未知数の数が 2 つに対して方程式の本数が 1 本しかなく、解が一意に定まらないので、一次**不定方程式**と呼ばれます。例えば、 $26x + 11y = 1$ という式に対しては $(x, y) = (3, -7), (x, y) = (-8, 19)$ などが解になっていることが確かめられます。^{*1}

これを解くために、ユークリッドの互除法が使われます。というよりは、ユークリッドの互除法の計算の過程に少し余分な手間を掛けるだけで、最大公約数を求めるのと同時に、解を計算することができます。具体的な例で、解の導出を観察してみましょう。ユークリッドの互除法を実行するとき、あまりを元の数 a, b であらわすような計算を付け加えておくことにします。 $a = 26, b = 11$ を例として計算すると次のようになります。

1. 最初は、 $26/11$ のあまりは $4 = 26 - 2 \times 11 = a - 2b$ です。
2. 次は、 $11/4$ のあまりは $3 = 11 - 2 \times 4 = b - 2(a - 2b) = 5b - 2a$
3. 次いで、 $4/3$ のあまりは $1 = 4 - 3 = (a - 2b) - (5b - 2a) = 3a - 7b$

最後に得られた式と式 (4.1) を見比べれば $x = 3, y = -7$ が解であることが分かります。式 (4.1) を解くつもりなどなかったのに、最大公約数を計算していたらいきなり答えが出できたということになります。

この計算はベクトル表記を使うと、その計算規則が明確になります。内積 $ax + by$ はベクトル表記で $(a, b) \begin{pmatrix} x \\ y \end{pmatrix}$ と表すことが出来ます。線形代数の知識から

$$\begin{aligned} c(a, b) \begin{pmatrix} x \\ y \end{pmatrix} &= (a, b) \begin{pmatrix} cx \\ cy \end{pmatrix} \\ (a, b) \begin{pmatrix} x \\ y \end{pmatrix} + (a, b) \begin{pmatrix} z \\ w \end{pmatrix} &= (a, b) \begin{pmatrix} x + z \\ y + w \end{pmatrix} \end{aligned}$$

という規則があることに注意すると、ユークリッドの互除法の計算は、次のようにベクトル表記することができます。ここで、 $26 = a = (a, b) \begin{pmatrix} 1 \\ 0 \end{pmatrix}, 11 = b = (a, b) \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ と書けることを利用しています。

^{*1} さらに、 $x = 3 + 11k, y = -7 - 26k (k = 0, \pm 1, \pm 2, \dots)$ という無限の解があることを確かめてください。一つの解が見つければ、このようにして無限個の解が生成できます。解が一つに定まらないという意味で「不定」です。

$$\begin{aligned}
4 &= 26 - 2 \times 11 = (a, b) \begin{pmatrix} 1 \\ 0 \end{pmatrix} - 2(a, b) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = (a, b) \begin{pmatrix} 1 \\ -2 \end{pmatrix} \\
3 &= 11 - 2 \times 4 = (a, b) \begin{pmatrix} 0 \\ 1 \end{pmatrix} - 2(a, b) \begin{pmatrix} 1 \\ -2 \end{pmatrix} = (a, b) \begin{pmatrix} -2 \\ 5 \end{pmatrix} \\
1 &= 4 - 3 = (a, b) \begin{pmatrix} 1 \\ -2 \end{pmatrix} - (a, b) \begin{pmatrix} -2 \\ 5 \end{pmatrix} = (a, b) \begin{pmatrix} 3 \\ -7 \end{pmatrix}
\end{aligned}$$

最後に得られた式は $3a - 7b = 1$ と同じです。内積の一方のベクトルは常に一定なので、各式の最右辺で右側のベクトルだけ取り出して（横ベクトルとして）並べると次のようになります。

i	c_i	d_i	x_i	y_i	$ax_i + by_i (= c_i)$
0	26	-	1	0	26
1	11	2	0	1	11
2	4	2	1	-2	4
3	3	1	-2	5	3
4	1	-	3	-7	1

c 列に元の数とあまりを、 d 列に商を書くことにすると、ユークリッドの互除法を使い、

$$\begin{aligned}
d_i &= \left\lfloor \frac{c_{i-1}}{c_i} \right\rfloor \\
c_{i+1} &= c_{i-1} - d_i c_i
\end{aligned} \tag{4.2}$$

と表すことができます。但し、 $\lfloor x \rfloor$ は x の整数部分を表す記号です。 x, y 列にベクトルの要素を書くようにすると、 $i+1$ 行目の (x_{i+1}, y_{i+1}) は、これもベクトル計算を使って $(x_{i-1}, y_{i-1}) - d_i(x_i, y_i)$ によって機械的に計算することができます。それを要素ごとに書くと

$$\begin{aligned}
x_{i+1} &= x_{i-1} - d_i x_i \\
y_{i+1} &= y_{i-1} - d_i y_i
\end{aligned}$$

となり、 c_{i+1} を計算する式 (4.2) と同じになっていることが分かります。したがって、次の命題が導かれたことになります。

命題 4.1 a, b が互いに素ならば、式 (4.1) を満たす整数解が存在する。その解は上の手順によって、実際に求めることが出来る。

以上の手順をアルゴリズムとしてまとめておきます。

アルゴリズム 6 一次不定方程式 $ax + by = 1$ を解くアルゴリズム

1. $c_0 = a, x_0 = 1, y_0 = 0, c_1 = b, x_1 = 0, y_1 = 1, i = 1$ とする。
2. $d_i = \left\lfloor \frac{c_{i-1}}{c_i} \right\rfloor$ 、 $c_{i+1} = c_{i-1} - d_i c_i$ 、 $x_{i+1} = x_{i-1} - d_i x_i$ 、 $y_{i+1} = y_{i-1} - d_i y_i$ を計算する。
3. $c_{i+1} > 1$ ならば、 $i + 1$ を新たな i としてステップ 2 へもどる。
4. (さもなければ) x_{i+1}, y_{i+1} が a, b の係数 (答え)、アルゴリズム終了。

但し、 $\lfloor x \rfloor$ は x 以下の最大整数を表す記号です。この不定方程式を解く手順は**拡張ユークリッド互除法**として知られています。

実習 20 (1) 97 と 35 の最大公約数が 1 であることを実習 19 のプログラムで確認し、 $a = 97, b = 35$ としたときの式 (4.1) の解を計算するための上のような $\{c_i, d_i, x_i, y_i\}$ の表を作りなさい。(2) 得られた表の動きを参考にして、拡張ユークリッド互除法のアルゴリズム 6 を理解しなさい。(3) 拡張ユークリッド互除法のアルゴリズム 6 を実現したプログラムを作り、 $a = 631, b = 239$ としたときの式 (4.1) の解を求めなさい。

ヒント：添え字付きの変数がたくさん出てきますが、その都度別の変数を使う必要はありません。 $c_0 = a, c_1 = b$ であまり c_2 を計算したら、次は c_1, c_2 を「新たな a と b 」として次のあまり c_3 を計算する、ということを繰り返します。したがって、 $\{c_i\}$ は a, b, c の三つの変数があれば計算できます。 $\{x_i\}, \{y_i\}$ に対しても同じような工夫をすれば良いのです。

参考 百五減算

息抜きのページ (52 ページ) に掲載した数あてゲームは百五減算として昔から知られた数論の知識を使ったゲームです。問題はこうです。

二桁の正整数を 3 で割ったら r_3 あまり、5 で割ったら r_5 あまり、7 で割ったら r_7 あまるとしたら、元の正整数はなあーんだ？

プログラムを読むと、その数は $35(3 - r_3) + 21r_5 + 15r_7$ を 105 で割ったあまりとして計算できるよう。例えば、79 の場合、3 で割ると 1 あまり、5 で割ると 4 あまり、7 で割ると 2 あまり、したがって、この式に代入すると、 $35 \times 2 + 21 \times 4 + 15 \times 2 = 184$ となり、 $184 - 105 = 79$ が計算できました。なぜそうなるか、その理屈は命題 4.1 を使って説明されます。3, 5, 7 が互いに素で、 $3 \times 5 \times 7 = 105$ 、ということがポイントです。

3 と $5 \times 7 = 35$ は互いに素ですから、命題 4.1 を適用すると、 $3x + 35y = 1$ となる x, y があります。実際 $x = 12, y = -1$ がそうです。したがって、その両辺に r_3 を掛けると、 $36r_3 - 35r_3 = r_3$ となります。2 つの数が等しければ、それを 3 で割ったときのあまりも

等しくなる、という自明な関係を使うと、

$$-35r_3 = r_3 \pmod{3} \quad (4.3)$$

が成り立ちます。但し、 $z = x \pmod{y}$ は $|x - z|$ が y で割り切れる、ということを表す記法です。同じように、5 と $3 \times 7 = 21$ 、7 と $3 \times 5 = 15$ も互いに素なので、 $-4 \times 5 + 21 = 1$ 、 $-2 \times 7 + 15 = 1$ という関係から

$$\begin{aligned} 21r_5 &= r_5 \pmod{5} \\ 15r_7 &= r_7 \pmod{7} \end{aligned} \quad (4.4)$$

が導かれます。

ここで、 $M = -35r_3 + 21r_5 + 15r_7$ と置くと、 M を 3,5,7 で割ったときのあまりがそれぞれ r_3, r_5, r_7 となります。したがって、 M に 105 を足したり引いたりして、2 桁の数になるようにすれば、それが答えです。

理屈が分かってみれば、問題を作ることもできるようになるでしょう。例えば、7 の代わりに 2 とすると、

$$\begin{cases} -7 \times 2 + 15 = 1 \\ -3 \times 3 + 10 = 1 \\ -5 + 6 = 1 \end{cases} \Rightarrow \begin{cases} 15r_2 = r_2 \pmod{2} \\ 10r_3 = r_3 \pmod{3} \\ 6r_5 = r_5 \pmod{5} \end{cases}$$

が成り立つので、30 以下の数で、2,3,5 で割ったらそれぞれ r_2, r_3, r_5 となった場合、 $M = 15r_2 + 10r_3 + 6r_5$ を計算して 30 で割ったあまりを計算すれば、元の数を出すことができます（三十減算）。実際、22 とした場合、 $r_2 = 0, r_3 = 1, r_5 = 2$ ですから、 $M = 10 + 2 \times 6 = 22$ となり、確かに求めることができました。

練習問題 4.13 3,5,7 に変わる三つの互いに素な数を使って、百五減算のような数当てゲームを考えてください。計算が簡単なものは見つかるでしょうか？

4.7 漸化式

コンピュータで計算する場合、足し算や掛け算の四則演算は一回につき二つの数を対象とした演算しかできません。例えば、 $1+2+3$ が 6 になるのは、 $1+2=3$ を一旦計算し、その次に $3+3$ を計算してようやく答えが 6 ということが分かるのです。したがって、多項式の計算では、2 項演算（つまり二つの数同士の演算）の繰り返しになるように、お膳立てをするのがプログラムを書く人の仕事になります。その場合に有用な考え方が漸化式です。例えば、 $a_1 + a_2 + \cdots + a_n (\equiv S_n)$ を計算したい場合、

$$\begin{aligned} S_k &= S_{k-1} + a_k, & k = 1, 2, \dots, n \\ S_0 &= 0 \end{aligned}$$

と書き換えたものが漸化式です。こうすると、 $S_1, S_2, \dots$ はすべて 2 項演算で計算できますから、C のプログラムで書くことが出来ます。 $a_k = k$ としたときの（1 から n までの総和を求める）アルゴリズムは次のようになるでしょう。

アルゴリズム 7 1 から n までの和を計算する。

1. $S = 0, k = 1$ とする。
2. $k > n$ ならばアルゴリズム終了。
3. S に k を加える。
4. k に 1 を加えてステップ 2 へ戻る。

ステップ 2,3,4 は「while($k \leq n$) { ... }」で実現できることが理解できるでしょう。

実習 21 次のプログラムは 1 から n までの総和を計算するプログラムです。入力して実行しなさい。

プログラム例

```
// 漸化式
1:      int k, n, S;
2:      printf("正の整数を入力しなさい ");
3:      scanf("%d", &n);
4:      S = 0;
5:      k = 1;
6:      while(k <= n) {
7:          S = S + k; // S += k と同じ
8:          k = k + 1;
9:      }
10:     printf("%d までの総和は %d です\n", n, S);
```

実習 21 の解説 チェックボックスに✓を入れながら読みなさい。

- 1. 7行目が漸化式に当たります。代入文の意味を思い出しなさい。右辺の計算結果を左辺の変数に代入する、ということです。同じ変数名を使っているとしても左辺と右辺ではその内容が異なります。右辺の S が S_{k-1} 、左辺の S が S_k を表していると考えればよいでしょう。
- 2. **(変数の初期化)** $S_0 = 0$ に当たる部分が4行目です。もし、4行目を書かないとすると、変数 S には何が記憶されているか分かりません。うまくいけば (S に0が記憶されていれば) 正解が得られますが、そのような偶然を期待してはいけません。
- 3. 文法的に難しい箇所はないはずです。例えば、最初に入力した数が3ならば、7,8行目をちょうど3回だけ繰り返して10行目に抜けるので、答えは6になります。

漸化式の例をいくつか挙げておきましょう。

例 4.1 べき乗 a^n は $a^n = a \times a^{n-1}$ と表すことができますから、

$$\begin{aligned} S_k &= a \times S_{k-1}, \quad k = 1, 2, \dots, n \\ S_0 &= 1 \end{aligned}$$

とすれば、 $a^n = S_n$ です。

例 4.2 2項係数は

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!} = \frac{n!}{k!(n-k)!}$$

と定義されていますが、これは

$$\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1} = \frac{n-k+1}{k} \binom{n}{k-1}$$

という関係があるので、漸化式で計算することができます。あるいはまた、

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

とも変形できるので、

$$\begin{aligned} S_{n,k} &= S_{n-1,k-1} + S_{n-1,k}, \quad k = 1, 2, \dots, n-1; n = 2, 3, \dots \\ S_{n,0} &= S_{n,n} = 1 \end{aligned}$$

のように表せば、二重の漸化式になります。

例 4.3 単純な漸化式なのにおもしろい性質を持ったものにフィボナッチ数列があります。

$$\begin{aligned}x_n &= x_{n-1} + x_{n-2}, \quad n = 2, 3, \dots \\x_1 &= x_0 = 1\end{aligned}$$

によって定義される数列をフィボナッチ数列と言います。計算してみると、 $x_2 = 2, x_3 = 3, x_4 = 5, x_5 = 8, \dots$ となり、 x_n は急速に大きくなりそうですが、 x_{n+1}/x_n がきれいな値に収束することが知られています。

例 4.4 次の漸化式は分数が出てきます。実数列を生成します。

$$\begin{aligned}z_n &= \frac{1}{1 + z_{n-1}}, \quad n = 2, 3, \dots \\z_1 &= 1\end{aligned}$$

順番に計算していくと、 $z_2 = \frac{1}{2}, z_3 = \frac{2}{3}, z_4 = \frac{3}{5}, z_5 = \frac{5}{8}, \dots$ ん？ どこかで見たような？ 分母の数字を並べると 2, 3, 5, 8, ... これはフィボナッチ数列ではないか。実際 $\{x_n\}$ をフィボナッチ数列として、 $z_n = x_{n-1}/x_n$ とすると、

$$z_{n+1} = \frac{1}{1 + x_{n-1}/x_n} = \frac{x_n}{x_n + x_{n-1}} = \frac{x_n}{x_{n+1}}$$

となって、確かに $\{z_n\}$ はフィボナッチ数列の隣同士の二数の比になっていました。

練習問題 4.14 $1 - 2 + 3 - 4 + 5 - \dots + (-1)^{n-1}n$ を漸化式で表し、それを計算するプログラムを書きなさい。

練習問題 4.15 $z_n = 1/(1 + z_{n-1})$ を計算するプログラムを書き、 z_n がどういう値になっていくか調べなさい。どうしてそうなるか、理屈を考えなさい（数学の問題です）。

練習問題 4.16 $a = 8191 (= 2^{13} - 1)$ に対して、 a^n を 10000 で割ったあまりを計算するプログラムを書いて、最初の 100 項を計算しなさい。1 行に 10 個ずつ表示させ、その数列の規則性を調べなさい。

ヒント： a^n を 10000 で割ったあまりは a^{n-1} を 10000 で割ったあまりに a を掛けたものを計算してそれを 10000 で割ったあまりに等しい、ということを使いなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F評価の項目について猛勉強しなさい。

- ☐ ☐ アルゴリズム
- ☐ ☐ 約数、自明な約数
- ☐ ☐ 素数
- ☐ ☐ 素因数、素因数分解
- ☐ ☐ 公約数、最大公約数、互いに素
- ☐ ☐ ユークリッド互除法
- ☐ ☐ 不定方程式、拡張ユークリッド互除法
- ☐ ☐ 漸化式

参考 実習プログラムの例（主要部分のみ）

実習 16（すべての約数を求める）のプログラム例（主要部分のみ）

```
b = 2;
while(b*b < a) {
    if(a%b == 0) printf(" %d %d", b, a/b);
    b = b+1;
}
if(b*b == a) printf(" %d", b);
```

実習 17（素数の判定）のプログラム例（主要部分のみ）

```
b = 2;
while(b*b <= a) {
    if(a%b == 0) {
        printf("%d は素数ではない\n", a);
        break;
    }
    b = b+1;
}
```

```
if(b*b > a) printf("%d は素数\n", a);
```

実習 18 (素因数分解) のプログラム例 (主要部分のみ)

```
b = 2;
while(b*b <= a) {
    if(a%b == 0) {
        printf(" %d", b);
        a = a/b;
        continue;
    }
    b = b+1;
}
printf(" %d\n", a);
```

実習 19 (最大公約数を求める) のプログラム例 (主要部分のみ)

```
while(1) {
    c = a % b;
    if(c == 0) break;
    a = b;
    b = c;
}
```

実習 20 (拡張ユークリッド互除法) のプログラム例 (主要部分のみ)

```
c0 = a; c1 = b;
x0 = 1; x1 = 0;
y0 = 0; y1 = 1;
do {
    d = c0 / c1;
    c2 = c0 - d*c1; c0 = c1; c1 = c2;
    x2 = x0 - d*x1; x0 = x1; x1 = x2;
    y2 = y0 - d*y1; y0 = y1; y1 = y2;
} while(c1 > 1);
```

4.8 章末演習問題

問題 4.1 フィボナッチ数列の指定された項の数だけ計算して表示するプログラムを書きなさい。40 項がどれくらいの大きさ（桁数）になるか予想してから、実際の計算で予想が当たったかどうか確かめなさい。

問題 4.2 フィボナッチ数列を $\{x_n\}$ としたとき、次が成り立つことを、プログラムを使って数値的に確かめなさい。(1) $x_0 + x_1 + \cdots + x_n$ は $x_{n+2} - 1$ に等しい。(2) $x_0 + x_2 + \cdots + x_{2n}$ は x_{2n+1} に等しい。(3) $x_1 + x_3 + \cdots + x_{2n-1}$ は $x_{2n} - 1$ に等しい。(4) $x_0^2 + x_1^2 + x_2^2 + \cdots + x_n^2$ は $x_{n+1} \times x_{n+2}$ に等しい。これらが成り立つことは、いずれも数学的帰納法で簡単に証明できます。

問題 4.3 $p_0 = 3, p_1 = 0, p_2 = 2, p_n = p_{n-2} + p_{n-3} \ (n \geq 3)$ によって定義される数列は**ペラン数**と呼ばれています。ペラン数を計算する C のプログラムを書き、 p_{75} までのペラン数についてチェックしなさい。また、 p_n が n で割り切れるような n を表示するプログラムを書き、その結果から何か分かることがあれば説明しなさい。

問題 4.4 6 の約数 $\{1, 2, 3, 6\}$ の自分自身以外の合計は 6、28 の 28 以外の約数 $\{1, 2, 4, 7, 14\}$ の合計は 28 になります。このように、ある数の自分自身以外の約数を全部足すとその数自身になるような数を**完全数**と言います。与えられた数が完全数かどうかを判定するプログラムと、それを検算するプログラムを書きなさい。そのプログラムを使って、28 の次に小さい完全数を求め、その数の全部の約数を表示しなさい。10000 までに完全数はいくつあるでしょうか。

問題 4.5 $3^2 + 4^2 = 5^2$ となるような 3 個の自然数の組は**ピタゴラス数**と呼ばれています。合計が 100 以下のピタゴラス数の組み合わせを計算するプログラムを書きなさい。(6, 8, 10) もピタゴラス数になりますが、そういう「おもしろくない」解を表示させないプログラムも考えなさい。

息抜きのページ

あなたの運勢を占います。次のプログラムを実行して、表示にしたがってデータを入力しなさい。

プログラム例

```
// 運勢判断
#include <stdio.h>
int main() {
    char b[100];
    int i, m;
    unsigned int n;
    while(1) {
        printf("あなたのお名前は? ");
        gets (b);
        printf("生まれた日、月、年の合計を入力しなさい  ");
        scanf("%d", &m);
        n = 1;
        for(i=0; i<m; i++) n *= 123456789;
        n = (n/10000) % 5;
        switch (n) {
            case 0: printf("%s さんは大吉です! ¥n¥n", b);
                    break;
            case 1: printf("%s さんは中吉です! ¥n¥n", b);
                    break;
            case 2:
            case 3: printf("%s さんは吉です! ¥n¥n", b);
                    break;
            case 4: printf("%s さんは凶です! ¥n¥n", b);
                    break;
        }
    }
}
```

第 5 章

繰り返しと配列変数

文法編その 3

フィードバック構造で良く出てくるのが、決められた回数、同じような計算を繰り返し実行する、という場合です。そのような計算のパターンをパッケージ化したのが「for」文です。それと密接に結びついた配列変数についても解説します。主な実習項目は以下のようなものです。

- **繰り返し構造**：for 構文、while 構文との違い、多重 for 構造
- **配列変数**：その宣言、1 次元配列、2 次元配列

5.1 繰り返し構造 (for)

if 文、while 構文に続いて、プログラムの流れを変える 3 番目の構文は「for 文」です。while で書けなくはないのですが、決まり切った手順をパターン化した命令文なので、あればとても便利、しかし思い込みの誤用も多発します。正しい知識を身につけてください。

文法予備知識

1. 命令文群を決められた回数を繰り返す場合は「for(...) {」と「}」で命令文群を囲む。
2. n 回繰り返す場合の標準的な書き方は「for(k=0; k<n; k++)」。

実習 22 次のプログラムは $1 + 2^2 + 3^2 + \dots + n^2$ を計算するプログラムです。入力して

実行しなさい。

プログラム例

```
// for 構文
1:   int k, n, S;
2:   printf("正の整数を入力しなさい ");
3:   scanf("%d", &n);
4:   S = 0;
5:   for(k=1; k<=n; k++) {
6:       S += k*k; // S = S+k*k 同じ*1
7:   }
8:   printf("%d までの 2 乗和は %d です\n", n, S);
```

実習 22 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. 6 行目の「 $k*k$ 」を「 k 」に代えれば、実習 21 のプログラムが解く問題と全く同じです。したがって、このプログラムの 5 行目から 7 行目と、実習 21 の 5 行目から 9 行目は全く同じ動き方をします。実習 21 の動きを改めてまとめてみると、次のようになるでしょう。

(1) k の初期化、(2) k の判定、(3) 計算、(4) k を増やして (2) へ戻る

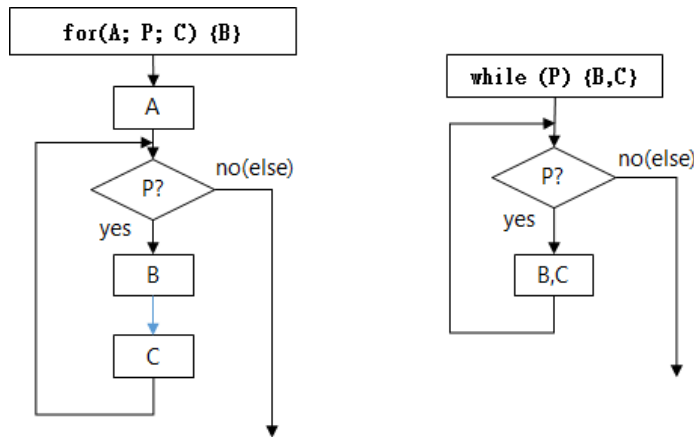
- 2. (for 文の構造) このように、同じような作業を決められた回数繰り返す、という計算はプログラムでしょっちゅう出てきます。そこで、上の 1,2,4 ステップをまとめてパッケージ化し、新たに一つの命令文にしたのが for 構文で、この実習プログラムの 5 行目のように書きます。一般に for ループの構造は次のような形になります。 k を動かしながら k の大きさによってループを続けるかどうかを決めるので、 k のことを**制御変数**ということがあります。

for(初期値設定 A; 条件式 P; 実行文 C) { 実行文群 B }

フローチャートで描けば次の左図のようになりますが、「while() { ... }」とほとんど同じ、ということが分かるでしょう。

- 3. (インクリメント演算子) 5 行目にある「 $k++$ 」は、「 $k=k+1$ 」を意味する C 言語独特の記法です。「 $++$ 」はインクリメント演算子と呼ばれます。C++ の「 $++$ 」はこの記法に由来しています（第一章参照）。実習プログラムの 9 行目、あるいは動く範囲を 1 ずらした「for($k=0$; $k<n$; $k++$)」は特によく使われるパターンですので、丸暗記しなさい。

*1 「+=」は代入演算子 (3.2.1 ページ参照)



- 4. 初学者にとってこの for 文は鬼門のようです。いつも同じようなフィードバックループを while 構文で書いているうちに、だんだん面倒くさくなってきたベテランのプログラマーが符丁化したようなものですから、プログラム経験のない人にとっては理屈抜きでただただ丸暗記するしかないのです。ですから、慣れない間は、for 文を while 文を使って書き直す練習をたくさんしていただくことを勧めます。最初は実習 21 のプログラムの方がわかりやすいでしょう。それでも分からないときは、次のような変数表を用意し、変数の値がどのように変化するか、代入文の実行に合わせて表を埋めていくようにしなさい。例えば、 $n=3$ としたときには表は以下になるでしょう。

k	$k \leq n$	S
1	正しい	1
2	正しい	5
3	正しい	14
4	正しくない	↓

自分で for 文を使うようなプログラムを書く場合には、いきなり `for(k=1; ...` と書き始めないで、小さい問題にしてフィードバックなしに全部書いてみる、ということが必要です。

- 5. **(カンマ演算子)** 実習プログラムでは「初期値設定」も「実行文 C」も命令は一つだけでしたが、二つ以上の実行文を書くことが出来ます。その場合複数の実行文を「;」ではなくカンマでつなげるという約束になっています。例えば、4 行目と 5 行目を一緒にして「`for(S=0,k=1; k<=n; k++) {`」と書いても構いません。S=0 と k=1 の間のカンマは**カンマ演算子**と呼ばれています。^{*2}

^{*2} それならば、実行文 B もカンマ演算子を使って実行文 C の前に書いても良いはずと考えると

練習問題 5.1 0 から $n-1$ までの数を 1 行に 10 個ずつきれいに表示するプログラムを書きなさい（主要部分だけでよい）。

練習問題 5.2 for 構文を使って $2^k (k = 1, 2, \dots, n)$ を表示させるプログラムを書きなさい。実行時に n を入力させるようにしなさい。表示は %15d のようなフォーマット指定子を利用して、見やすくなるように工夫しなさい。^{*3}

実習 23 次のプログラムを良く読んで理解し、実行結果がどのようになるかを紙に書いてから、実際に入力して実行させ、最初紙に書いたものと比べなさい。

プログラム例

```
// for ループのプログラム、その 2
1:      int k, n=3;
2:      for(k=n; k>0; k--) {
3:          printf("for その 2: k=%d, k>0 ?¥n", k);
4:      }
5:      printf("for その 2: k=%d, k>0 ?¥n", k);
```

実習 23 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **（ディクリメント演算子）** for 文のよく出てくるもう一つのパターンです。2 行目に出てくる「k--」はディクリメント演算子と呼ばれ、「k=k-1」と同じ意味です。「for(A;P;C) B」構文は、「 $A \rightarrow P \rightarrow B \rightarrow C \rightarrow P \rightarrow B \rightarrow C \rightarrow \dots$ 」のように推移するという流れがきちんと理解されていれば問題ないでしょう。
 $n=3$ なので、 $k=3, 2, 1$ の順に printf 文が実行され、 $k=0$ になったときにループから追い出されます。「while(1)」の理屈を説明した 42 ページの文章を思い出すと、条件式の「 $k>0$ 」は「k」でも良いことが分かります。
- 2. **（for と while の使い分け）** どのような場合に for 構文を使い、どのような場合に while 構文（あるいは do...while）を使うかということについての決まりはありませんが、for 構文は「典型的な」while 構文を符丁化したものなので、それ以外は while 構文を使いなさい。典型的というのは、「for(k=0; k<n; k++)」「for(k=n; k>0; k--)」のように、制御変数が等差数列で増えたり減ったりするようなフィードバックループのことです。

「for(S=0,k=1; k<=n; S+=k,k++);」という「シンプルな」プログラムができます。確かに動きますが、普通はこういうプログラムは書きません。カンマ演算子で書くのは $S=0$ のような変数の初期化に限る、としておいた方が無難です。

^{*3} フォーマット指定子で「%」と「d」の間の数字は表示領域の桁数を指定します（21 ページ参照）。

但し、while 構文は for 構文と違って初期値の設定がなく、最初に条件式の判定があります。条件式の中身は while 文の中で計算されて変わる変数を含むことが多いのですが、最初の判定ではまだ計算されていません。したがって、そのような場合は必ず最初は条件式が判定できるように while に入る前に初期条件を設定しておくか、do ... while 構文を使いなさい。

for 文の別のパターンをいくつかあげておきましょう。

1. 「for(k=1; k<=9; k+=2) {printf(" %d", k);}」: 「k+=2」は k を 2 ずつ増やす、ということを意味しますから、k は奇数だけで 9 まで増えます。k<11 と書いても同じことです。結果は「1 3 5 7 9」。
2. 「for(k=7; k>0; k-=3) {printf("%d,", k);}」: これは 3 ずつ減らすという意味です。結果は「7,4,1,」。
3. 「for(k=5; k>0; k=++) {printf("%d ", k);}」: これは暴走します。慌てずに、実行画面の「閉じるボタン」をクリックしなさい。
4. 「for(k=2; k<1000; k=k*k) {printf("%d ", k);}」: for 文でも書けますが、いつループを抜け出すか、すぐには分からないので、こういう場合は while 構文を使いなさい。結果は「2 4 16 256」。

練習問題 5.3 次のように与えられた数列の規則を理解し、それを for 構文で表すとしたら「初期値設定、条件判定、実行文 C、実行文群 B」が何に対応するかを考え、実際に for 構文を使って、その数列を表示させるプログラムを書きなさい。(1) 3, 6, 9, ..., 30、(2) 1, 2, 4, 5, 7, 8, 10, 11, ..., 28, 29、(3) 1, -2, 3, -4, 5, -6, 7。

練習問題 5.4 キーボードから正整数 n を入力させ、 n 以下の偶数を大きいものから順に表示するようなプログラムを書きなさい。for 文を使うプログラムと、while 文を使うプログラムを書いて、その動きを比較しなさい。

練習問題 5.5 整数を入力し、その数以下の奇数の総和を計算するプログラムを書きなさい。0 以下の数字が入力されたら「作業終わり」と表示してプログラムを終了させるようにしなさい。

練習問題 5.6 for 構文の動きを理解した上で、「for(;;)」が「while(1)」と同じ無限ループを形成するということの理由を説明しなさい (for でも書ける、という例ですので、使わないほうが無難です)。

5.2 配列変数

入力したデータを利用するには変数が必要でした。統計処理のように大量のデータを扱う場合、そのすべてに別々の名前を与えることは容易ではありません。その場合は、数学のベクトルや行列の記法にヒントを得て、添え字付き変数の考え方を導入します。但し、Cでは下付き文字は扱えないので、それに変わる特別な記号が必要になります。それが配列変数です。配列変数を導入することで、計算の世界がまた大きく広がることになります。

5.2.1 1次元配列変数

まずは、数学の感覚でプログラムを書いたときの VC2010 の反応を見ておくことにしましょう。

実習 24 $a_1, a_2, \dots, a_{100}$ という 100 個のデータをキーボードから入力して、その合計を表示するという仕事をさせるつもりで次のようなプログラムを書きました。結果がどうなるかビルドする前に予想しなさい。そして実際にビルドしなさい。

プログラム例

```
// 不特定多数の変数を扱う（数学流に）
1:   int a1, a2, ..., a100, goukei;
2:   printf("100 個の数を入力しなさい  ");
3:   scanf("%d %d ... %d", &a1, &a2, ..., &a100);
4:   goukei = a1 + a2 + ... + a100;
5:   printf("goukei = %d¥n", goukei);
```

実習 24 の解説

数学的記法では、「以下同様に」という意味をこめて「...」と書きますから、問題文にあるような仕事をするプログラムを書きたいのだなあ、という気持ちは分かりますが、残念ながらコンピュータはこのような約束を知りません。C の文法にはこのような記法がありませんから、上のプログラムをビルドすればエラーが発生します。「それくらいの書き方も理解してくれないの」とぼやいても仕方ありません。

このプログラムが正しい C のプログラムにするためには、 a_1 から a_{100} まで 100 個の変数を全部宣言し、3,4 行目にも 100 個の変数を明示的に書けばよい「だけ」ですが、かなり大変です。1000 個の合計となると、...、ばからしくなってきます。致命的なのは、合計する個数 n が実行時にないと分からない場合です。それではプログラムの書きようがありません。

添え字を下付き文字とせずに変数名に `[k]` という形式の添え字を付け加えた配列変数を導入することで上の問題を一挙に解決することが出来ます。

文法予備知識

1. ベクトルのような数の集まり ($a_0, a_1, \dots, a_{10}$) を表現するために、変数にカギ括弧付きの数を添えたもので表す (`a[0], a[1], \dots, a[10]`)。
2. 配列変数は、あらかじめ使用する最大の個数を宣言する必要がある。
3. 配列変数の大きさを 3 と宣言した場合、使える変数名は `a[0], a[1], a[2]` の三つである (添え字は 0 から始まる)。

実習 25 次のプログラムは、実行したときに決まる個数とその個数分のデータを入力して、その合計を計算するものです。入力して実行しなさい。

プログラム例

```
// 不特定多数の変数を扱う (C の流儀で)
1:   int a[1000], goukei, k, n;
2:   printf("データの個数を入力しなさい ");
3:   scanf("%d",&n);
4:   printf("%d 個の数を入力しなさい ",n);
5:   for(k=0; k<n; k++)
6:       scanf("%d", &a[k]);
7:   goukei = 0;
8:   for(k=0; k<n; k++)
9:       goukei += a[k];
10:  printf("goukei = %d¥n", goukei);
```

実習 25 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (**配列変数**) 変数名の後に「`[]`」を付けたものを**配列変数**と言い、配列変数から `[]` を除いたものを**配列名**と言います。`[]` の中の数を**添え字**と言い、添え字には `int` 型数、あるいは `int` 型変数を書きます。1 行目が `int` 型の配列変数の宣言方法です。`[]` の中には正整数を書きます。変数を書くことは許されません。この宣言によって、1000 個の添え字付き変数が一度に定義されたことになります。添え字の範囲は 0 から始まる、と約束されていますので、プログラムで使える変数名は `a[0]` から始まり `a[999]` までの 1000 個です。数学で使う場合、ベクトルや数列の添え字は多くの場合 1 から始まりますが、情報数学では 0 から始めた方が何かと便利な

ので、Cでも0から始めるという決まりがあります。したがって、大きさ n の配列を宣言した場合、添え字の動く範囲は0から $n-1$ までの n 通りですので注意が必要です。

- 2. 6行目や9行目のように、配列変数は通常の変数が使えるところではどこでも同じように使うことができます。もちろん代入文の左辺にも使えます。
- 3. **(範囲外の参照)** 2章の最初に、変数はコンピュータのメモリに記憶されるということを説明しました。配列変数はメモリの連続した領域に記憶されます。プログラム例では1000個の `int` 型データを記憶できるような連続領域を確保します。その先頭から順番に、`a[0], a[1], ..., a[999]` という変数名で参照することが出来ます。しかし、プログラムに `a[k]` と書かれている場合、VC2010はそれが `a[0]` から $k+1$ 番目に記憶されている変数名と解釈し、 k が0以上999以下であるかどうかをチェックしません。そのため、範囲外を参照してもビルドの際にエラーにはなりません。

そのまま実行して、もし $k=1000$ となった場合、`a[1000]` という変数名を扱うことになりますが、その変数の参照するメモリに何が記憶されているかは分からないので、その結果プログラムがどのような計算をするかは予想ができません。たまたまうまくいくかもしれませんが、そのような幸運を期待してはいけません。添え字の範囲はプログラムを書く人がきちんと管理する必要があります。コンピュータのメモリと変数の関係の詳細は9章で説明します。

- 4. **(データの入力)** 複数のデータを連続して入力する場合、`scanf` 文は一回につき1個のデータを読み込むようになっていますが、キーボードから入力するときは、スペース区切りで1行にいくつものデータを同時に入力することができます。

実習 26 次のプログラムは、配列に入力したデータを部分的に表示するものです。入力して実行しなさい。10行目で入力するデータは、**(1)** 2,4、**(2)** 10,20 としなさい。

プログラム例

```
// 配列変数
1:   int k, na, nb;
2:   int a[]={3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3};
3:   while(1) {
4:       printf("¥n 何番目から何番目までを表示しますか ");
5:       scanf("%d %d", &na, &nb);
6:       for(k=na-1; k<=nb-1; k++) {
7:           printf("a[%d] = %d¥n", k, a[k]);
8:       }
9:   }
```

実習 26 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **(初期値の設定)** 配列要素にも、宣言文で初期値を設定することができます。2 行目
がその書き方の一例です。配列の大きさ（確保すべき要素の数）を書く場所が空欄
になっていますが、これは初期値で指定された数がちょうど収まるような大きさ、
この場合は 16、が宣言されたことと同じになります。これにより、「`int a[16];`」
と宣言し、「`a[0]=3;a[1]=1;...;a[15]=3;`」という 16 個の代入文を書いたの
と同じことになります。もちろん、配列の大きさを明示的に宣言してもかまいません。
明示的に宣言した場合はそちらが優先されます。初期値が足りない場合は、最
初の添え字部分にだけ初期値が設定され、残りの配列部分に何が入るか不定です。
- 2. **(文法違反?)** 入力データとして「10 20」とした場合、`k` の動く範囲は 9 から 19 ま
でですから、`a[16]`, `a[17]`, ... などの定義されていない変数が使われることにな
ります。文法違反にはならないため、ビルドしても何もエラー表示はされずに、実
行すると「何か」が表示されます。詳しくは実習 25 の解説（範囲外の参照）を読
みなさい。この場合は「入力数字は 1 から 16 まで」というように、表示させてか
ら入力を促すとか、あるいは、範囲外の数を入力した場合は「それはおかしい、再
入力しなさい」というメッセージを表示するようにすべきです。
- 3. **(double 型の配列)** `double` 型の配列も `int` 型の場合と同様、「`double b[5];`」の
ようにして宣言することができます。

練習問題 5.7 `a[0]`, `a[1]`, ..., `a[n-1]` が与えられているとき、先頭から順番に、1 行に
5 個ずつ表示するプログラムを書きなさい。配列変数は `int` 型とします（以下の 3 問も
同様）。

練習問題 5.8 `a[0]`, `a[1]`, ..., `a[n-1]` が与えられているとき、`a[n-1]`, `a[n-2]`, ..., `a[0]`
の順に表示するプログラムを書きなさい。

練習問題 5.9 `a[0]`, `a[1]`, ..., `a[n-1]` が与えられているとき、`a[1]` の中身を `a[0]` に、
`a[2]` の中身を `a[1]` に、...、`a[n-1]` の中身を `a[n-2]` に移し替え、最初にあった `a[0]`
の中身を `a[n-1]` に移し替えるプログラムを書きなさい。配列要素の中身が変わっている
ことに注意。

練習問題 5.10 `a[0]`, `a[1]`, ..., `a[n-1]` が与えられているとき、`a[0]`, ..., `a[n-2]` を
`a[1]`, ..., `a[n-1]` に移し替え、最初にあった `a[n-1]` を `a[0]` に移し替えるプログラム
を書きなさい。

練習問題 5.11 実習 26 のプログラムの入力部分を書き換えて、範囲外の数や、`na > nb` と
いう数が入力された場合は再入力させるようにしなさい。

ヒント：continue 文を使いなさい。

5.2.2 2次元配列変数

統計計算では、Excel シートのような 2 次元の表の形をしたデータを扱うことも多くなります。こういうデータを扱うために、行列 $A = (a_{ik})$ のような二つの添え字付き変数があれば便利です。配列変数を定義したときと同様に、`[i]` というような表記を添え字の代わりに使い、`[]` を二つ並べることで C でも行列形式の変数を扱うことができます。

文法予備知識

1. Excel シートに書き込まれた 2 次元データは、プログラムでは `a[i][j]` のような二つの添え字付き変数で表現される
2. 「`int a[2][3];`」と宣言したとき、コンピュータは `a[0][0]` , `a[0][1]` , `a[0][2]` , `a[1][0]` , `a[1][1]` , `a[1][2]` の順番で一次元配列として記憶する

実習 27 (1) 次のプログラムを入力し、実行しなさい。(2) 3 行目の「`j<2`」を「`j<3`」と書き換えて実行しなさい。どうなりましたか。(3) 4 行目の「`k<3`」を「`k<4`」と書き換えて実行しなさい。どうなりましたか。

プログラム例

```
// 2次元配列変数
1:   int a[2][3]={0,1,2,3,4,5};
2:   int j, k;
3:   for(j=0; j<2; j++) {
4:       for(k=0; k<3; k++) {
5:           printf("a[%d][%d] = %d\n", j, k, a[j][k]);
6:       }
7:   }
8:   printf("\na[0][3] = %d, a[2][0] = %d\n", a[0][3], a[2][0]);
```

実習 27 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. 線形代数で出てくる行列をコンピュータで表現するときは、添え字が二つの配列変数が必要です。それは 1 行目のように、先に定義した 1 次元配列変数の後にさらに「`[3]`」という 2 番目の添え字を追加することで表現します。いわば「`a[i]`」を一つの配列名と見なして、その配列の何番目、という指定の仕方をすると考えれば

よいでしょう。数学の記号と対応付けるとすれば、行列 $A = (a_{ik})$ をプログラムで表現したとき、 a_{ik} が `a[i][k]` という変数、 $(a_{i1}, a_{i2}, \dots, a_{in})$ という行ベクトルが `a[i]` という配列名に対応し、行列 A そのものが `a` という 2 次元配列名に対応していると考えてください。

- 2. 1 行目が 2 次元配列変数の定義の仕方です。1 次元配列の宣言同様、配列の範囲は正整数で定義します。変数は使えません。ここでは 2 行 3 列の行列を定義したことになります。添え字付きの配列名を定義していると考えれば、上の約束にしたがって

```
a[0]={0,1,2}, a[1]={3,4,5}
```

という二つの行ベクトルが定義されたと読みます。したがって、要素ごとに書けば、先頭から順に、

```
a[0][0]=0, a[0][1]=1, a[0][2]=2,  
a[1][0]=3, a[1][1]=4, a[1][2]=5
```

という 6 個の代入文が書かれているのと同じで、1 次元の配列同様、添え字は 0 から始まることに注意してください。

- 3. 「2 次元配列」というと、Excel の表のようなものがコンピュータの中にあるかのように想像してしまいがちですが、コンピュータの中でそのような表が作られるわけではありません。詳しくは 146 ページで説明しますが、上の説明でも分かるように、2 次元配列を決められた規則にしたがって 1 列（1 次元配列）に並べ、その順にメモリが割り当てられます。
- 4. (2 次元配列の添え字) 一般に「`A[m][n]`」と宣言された場合、変数 `A[i][k]` は `A` の先頭 (`A[0][0]`) から数えて $n \times i + k + 1$ 番目の変数になります。1 次元配列の場合と同様に、添え字の範囲は、 i が 0 から $m-1$ まで、 k が 0 から $n-1$ でなければいけません。しかし、添え字が決められた範囲にないとしても、文法違反にはならないので、プログラマーが自分で管理しなければいけません。この実習プログラムの場合、1 行目の宣言文によって $2 \times 3 = 6$ 個の `int` 型データを記憶できるようなメモリの連続した領域が確保され、その先頭から順番に `a[0][0]`, `a[0][1]`, ..., `a[1][2]` という 6 個の変数名で参照することが出来るようになります。一方、`a[i][j]` という変数は `a[0][0]` から数えて $i \times 3 + j + 1$ 番目のデータ（を記憶する領域）に付けられた名前と約束されていますので、`a[0][3]` は定義された添え字の範囲を逸脱していますが、計算規則により、`a[1][0]` と同じ場所を指す変数ということになります。また、`a[2][0]` と書いてもエラーにはならず、`a[0][0]` から 7 番目というありもしないデータを参照することになります。
- 5. 4 行目から 6 行目は 3 行目の `for` 構文の中に入っている `for` 構文です。定義通り考えると、 $j=0$ として 4 行目から 6 行目を実行し、 $j=1$ として 4 行目から 6 行目を実行し、 $j=2$ となったところで 3 行目から 7 行目のループを脱出して 8 行目へ移り

ます。このような 2 重ループはこれからも頻繁に出てきますが、ややこしくなったら、for 文を分解して、繰り返しの回数を全部書いてみる、ということで解決されるはずです。直感に頼らず、地道に作業しなさい。

練習問題 5.12 10 行 10 列の 2 次元の int 型配列を定義して、a[0][0] から始まる 3 行 4 列の行列の要素 12 個を入力し、各行の行和を計算して、元の行列と一緒に表示するプログラムを書きなさい。行列は 2 次元の表らしく表示させること。

練習問題 5.13 二つの行列 $(a_{ik}; i = 1, 2, \dots, m, k = 1, 2, \dots, n), (b_{ik}; i = 1, 2, \dots, m, k = 1, 2, \dots, n)$ の和 $(c_{ik} = a_{ik} + b_{ik})$ を計算するプログラムを書きなさい。計算結果は 2 次元の行列のように表示させなさい。添え字は数学風に 1 から始まるものとします。したがって、0 行の要素、0 列の要素は使いません。データ型は int 型とし、宣言文では少し余裕を持って 2 次元配列を確保しなさい。

練習問題 5.14 2 項係数 $\binom{n}{k}$ は前章の例 4.2 で定義されています。2 次元配列 c[11][11] を使って、 $c[n][k] = \binom{n}{k}$ となるような行列を計算しなさい。但し、 $n = 0, 1, \dots, 10, k = 0, 1, \dots, n$ の範囲を動くものとします。表示の仕方も工夫しなさい。

5.3 多重の for、複雑な繰り返し構造

for 文のなかにまた for 文があるという構文は、これから良く現れます。だいたいは定型のパターンですから、覚えてしまえばどうってことはありません。数学の数列の和の計算で、

$$\sum_{i=1}^n \sum_{j=1}^i a_{ij}, \quad \sum_{i=1}^n \sum_{j=i}^n a_{ij}, \quad \sum_{i=1}^n \sum_{j=n-i}^n a_{ij}$$

のような二重のシグマ記号は慣れていると思いますが、考え方はプログラミングでも同じです。for 文の場合は、添え字が小さくなりながら動くこともある、というのがちょっとやっかいです。

実習 28 次のプログラムを実行したとき、変数 k, i がどのように変わるか、その結果何が表示されるか、コンピュータを使わずに予想して、紙に書きなさい。前後に必要な部分を付け加えてコンピュータで実行し、その書いた結果を確かめなさい。1 行目の「4」を一般の数にした場合に、 k, i がどんな関係を持って動くのかを説明しなさい。

プログラム例

```
// 二重の for ループ
1:  int k, i, n=4;
2:  for(k=0; k<n; k++) {
3:      printf("k=%d < %d:  ", k, n);
4:      for(i=k; i<n; i++) {
5:          printf("(%d, %d) ", k, i);
6:      }
7:      printf("\n");
8:  }
```

実習 28 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. for 文はある特殊なフィードバック構造をパッケージ化してコンパクトな記号に置き換えたものです。慣れている人はこれで十分に分かりやすく、悩むこともないのですが、慣れていない人にとってはどのタイミングで変数が変化するのか、比較がどのタイミングで行われるのか、神経をすり減らすことになるでしょう。実習 22 の解説を良く読んで、実習 21 のような while 文で書き直してみるということが理解の助けになるかもしれません。2 行目と 4 行目を while 構文を使って書き直すと、次のようになります。

```
k=0; while(k<4) { ... i=k; while(i<4) { ... i++;} ... k++;}
```

- 2. for の制御変数を表にまとめるというのは良い習慣です。表示内容と合わせてまとめたのが次の表です。このような地道な努力を惜しまないように。

k	i の動く範囲	表示
0	0,1,2,3	k=0 < 4: (0, 0) (0, 1) (0, 2) (0, 3)
1	1,2,3	k=1 < 4: (1, 1) (1, 2) (1, 3)
2	2,3	k=2 < 4: (2, 2) (2, 3)
3	3	k=3 < 4: (3, 3)

- 3. (ブレークポイント) コンピュータのデバッグ機能をうまく使うと理解の助けになる場合もあります。6 行目の左端をクリックするとそこに赤い円盤が表示されます。これをブレークポイントと言います*4。その状態から F5 キーを押して実行させると、外側の for ループで k=1、内側の for ループで i=0 として 5 行目を実行し「k=0 < 4 : (0,0)」と表示したところで一時停止 (ブレーク) しますので、そこまでの動きをじっくり検討する時間を持つことができます。次いで、VC2010 のプログラムを入力したウィンドウに戻って F5 キーを押すと、i=1 として 5 行目を実行し「(0,1)」と表示してまた停止します。実行画面が前面に出ている状態で F5 キーを押しても反応しないので注意してください。実行画面が見えるように、VC2010 の画面を小さくする必要があるかもしれません。工夫してください。以下同様にして、F5 キーを押すたびに for 文の制御変数の値が更新され、プログラムの進行をチェックすることができます。ブレークポイントを 6 行目ではなく、8 行目に設定すると、k=0 の行全体が表示されたところで一時停止します。ブレークポイントはいくつでも設定することができますし、ブレークポイントをクリックすれば消すことは簡単にできますので、printf 文と組み合わせてうまく使うことによって、デバッグの効率を上げることができるようになるでしょう。うまくつきあってください。

練習問題 5.15 実習 28 の 2 行目と 4 行目を次で差し替えたときに表示される内容を書きなさい。

- (1) for(k=0; k<4; k++) と for(i=0; i<4-k; i++)
- (2) for(k=0; k<4; k++) と for(i=4; i>k; i--)
- (3) for(k=0; k<4; k++) と for(i=4-k; i>0; i--)
- (4) for(k=3; k>0; k--) と for(i=4-k; i<3; i++)

*4 同じことですが、その行にカーソルを置いてマウスを右クリックし、「ブレークポイント」→「ブレークポイント挿入」メニューをえらんでも同じ結果になります。

練習問題 5.16 n に 4 を入力すると、1 行目に「0 1 2 3」、2 行目に「3 0 1 2」、3 行目に「2 3 0 1」、4 行目に「1 2 3 0」と表示するようなプログラムを書きなさい。

ヒント：「%」（あまり計算）を使いなさい。

練習問題 5.17 2 以上の正整数 n を入力すると、足して n 以下になる二つの正整数のペアを辞書的順序で表示するプログラムを書きなさい。辞書的順序とは、辞書のように 1 つめの数字の小さい順、1 つめの数字が同じならば 2 つめの数字の小さい順、という並びを言います。例えば $n = 4$ とすると、「(1,1)(1,2)(1,3)(2,1)(2,2)(3,1)」と並びます。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の 3 段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ for 構文、制御変数
- ☐ ☐ インクリメント演算子、「++、--」
- ☐ ☐ カンマ演算子
- ☐ ☐ 配列変数、宣言の仕方、「[]」の使い方、添え字の範囲
- ☐ ☐ 配列変数の定義範囲外の参照
- ☐ ☐ 2 次元配列変数
- ☐ ☐ デバッグとブレークポイント

5.4 章末演習問題

問題 5.1 「二つの `int` 型数 a, b を入力し、 $a > b$ ならば $a, a-1, \dots, b$ と表示し、 $b > a$ ならば $b, b-1, \dots, a$ と表示する」という作業を $a = b$ でない限り繰り返す、というプログラムを作りなさい。

問題 5.2 正の整数を入力し、それが素数かどうかを判定するプログラムを `for` 文を使って書きなさい。

問題 5.3 正の整数 n のすべての約数を大きさの順に表示するプログラムを書きなさい。

問題 5.4 $n^2 + n + 41 (n = 0, 1, 2, \dots, 39)$ がすべて素数であることをチェックするプログラムを書きなさい（オイラーの式）。

問題 5.5 ボーリングのスコアシートを計算するプログラムを書きなさい。ストライクは 2 投、スピアは 1 投のプレミアムがカウントされます、という説明で分からなかったらネットで調べてください。

問題 5.6 一つの行列 $(a_{ik}; i = 1, 2, \dots, m, k = 1, 2, \dots, n)$ と一つのベクトル $(b_1, b_2, \dots, b_n)$ の積 $(c_i = \sum_{k=1}^n a_{ik} b_k)$ を計算するプログラムを書きなさい。データは `double` 型としなさい。

問題 5.7 二つの行列 $(a_{ik}; i = 1, 2, \dots, m, k = 1, 2, \dots, n), (b_{ik}; i = 1, 2, \dots, n, k = 1, 2, \dots, p)$ の積 $(c_{ik} = \sum_{j=1}^n a_{ij} b_{jk})$ を計算して、新たな行列 (c_{ik}) を作るプログラムを書きなさい。データは `double` 型としなさい。

問題 5.8 正方行列 $(a_{ik}; i = 1, 2, \dots, m, k = 1, 2, \dots, m)$ の n 乗を計算するプログラムを書きなさい。データは `double` 型としなさい。

息抜きのページ

簡単なじゃんけんゲームです。後出しはしません、信用して遊んでください。

プログラム例

```
// じゃんけん
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main( ) {
    int comp, you, win=0, lose=0;
    char* s[3];
    s[0] = "グー"; s[1] = "チョキ"; s[2] = "パー";
    srand ((unsigned int) time (NULL)*314159265);
    printf("じゃんけんをしましょう ¥n");
    while(1) {
        comp = 3 * rand() / (RAND_MAX+1) + 1;; // 私の手
        printf("じゃんけんぽん
        (1:グー、2:チョキ、3:パー ?) ... "); // 前の行の続き
        scanf("%d", &you);
        if (you < 1 || you > 3) {
            printf("まじめにやってよ ('0')^^! ¥n¥n");
            continue;
        }
        printf("あなたは%s、わたしは%s、", s[you-1], s[comp-1]);
        if (you == comp) printf("あいこです ¥n");
        else if ((you+3-comp)%3 == 2) {
            win++;
            printf("あなたの勝ちです ¥n");
            if (rand() / (RAND_MAX+1.0) < 0.1)
                printf("なんか、あと出しっぽいなあ ¥n");
        } else {
            lose++;
            printf("コンピュータの勝ちです ¥n");
        }
        printf("あなたの %d 勝 %d 敗です ¥n¥n", win, lose);
    }
}
```

第 6 章

算法 2：統計計算

実験データや調査データを統計処理する場合にコンピュータが必須になります。その処理の基本的な計算手順（アルゴリズム）について解説します。この章で扱うのは次のような計算のアルゴリズムです。

- データの入力、保存
- 標本平均、標本分散、標本標準偏差の計算
- 最大値、最小値、順位の計算
- 度数分布

最後に有効桁について、注意事項を解説します。

6.1 データの入力

統計処理の前にすべてのデータをキーボードから入力して配列に記憶させることを考えます。C の配列の添字は 0 から始まりますが、たいていの統計の教科書でデータの添字は 1 から始まります。それに合わせて、ここでも添字 1 の配列要素に 1 番目のデータを記憶させることにします。配列は、プログラムを実行する前に（データの個数が分かる前に）その大きさを定数で指定しなければいけません。データの個数が分からないと言っても、キーボードから入力できる個数には限りがあります。変数が足りなくならないように、十分に大きめの配列を宣言しておきなさい。

キーボードから多数のデータを入力する場合はデータ個数分の `scanf` 命令を実行すれば良いのですが、入力の終了をどうやってプログラムに知らせるか、工夫が必要です。よく使われるのは**ストップコード**と呼ばれる特別なデータを用意して、それが入力されたらデータ入力終了とすることです。例えば、データは非負の数ということが分かっている場合、負数が入力されたらデータ入力を終了する、と約束するのです。この場合、負数がストップコードになります。データ入力ルーチンをアルゴリズム風を書けば次のようになる

でしょう。入力終了したとき、データは $a[1]$, $a[2]$, ..., $a[n]$ に記憶されます。

1. $n = 0$ とする。
2. 一つのデータを変数 `temp` に入力する。
3. `temp` がストップコードならば入力終了。
4. $n + 1$ を新たな n とし、 $a[n]$ に `temp` を代入し、ステップ 2 へ。

データの個数があらかじめ分かっている場合は、最初にデータ個数を入力させ、データを入力しながらその個数を数えることで、ストップコードに頼ることなく入力終了を知ることが出来ます。この場合は `for` 構文、データ個数が未知の場合は `while` 構文が目には浮かびますが、どの場合もどちらの構文を使ってもプログラム可能です。

実習 29 (1) `while` 構文を使って、データ個数が未知の場合のデータ入力プログラムを書き、入力されたデータを整った表形式で表示するようにしなさい。データは `double` 型とし、ストップコードは変数名 `STOPCODE` で定義するようにしなさい (`double STOPCODE=...; ... if(x == STOPCODE) ...`)。 (2) `for` 構文を使って、データ個数が既知の場合のデータ入力プログラムを書きなさい。データの個数は最初にキーボードから入力させるものとし、適当なプロンプト文を使いなさい。

複数のデータを入力する場合は、一つのデータを入力する毎に Enter キーを押しても良いのですが、スペースキーで区切ってデータを 1 行に並べて入力し、最後に Enter キーを 1 回押すだけで一度に入力することが出来ます。

練習問題 6.1 データ個数が未知の場合、`for` 構文を使ってデータ入力プログラムを書きなさい。

練習問題 6.2 入力したデータを $a[0], a[1], \dots, a[n-1]$ に記憶させたい場合は、どこを修正すればよいでしょうか。

6.2 標本平均、標本分散の計算

データの特徴をつかむために基本統計量と呼ばれるいくつかの特性量が利用されます。データの中心（重心）を表す標本平均、ばらつきを表す標本分散、標本標準偏差、そして、最大値、最小値、範囲、中央値、などです。まず、標本平均と標本分散を計算します。 n 個のデータを $x_1, x_2, \dots, x_n$ とすると、**標本平均** $\bar{x}$ 、**標本分散** s^2 、**標本標準偏差** s は次のように定義されます。

$$\bar{x} = \frac{1}{n} \sum_{k=1}^n x_k, s^2 = \frac{1}{n} \sum_{k=1}^n (x_k - \bar{x})^2, s = \sqrt{\frac{1}{n} \sum_{k=1}^n (x_k - \bar{x})^2}$$

標本分散 s^2 ($n-1$ で割ったものは**不偏分散**と呼ばれますが、それを標本分散と呼ぶ人もいます) を計算する場合、定義式通りだと、先ず標本平均 $\bar{x}$ を求め、それから偏差 $x_k - \bar{x}$ の 2 乗を累積することになりますが、式を変形しておくと、 $\bar{x}$ と並行して計算できることが分かります。

$$s^2 = \frac{1}{n} \sum_{k=1}^n (x_k - \bar{x})^2 = \frac{1}{n} \sum_{k=1}^n x_k^2 - \frac{2\bar{x}}{n} \sum_{k=1}^n x_k + \frac{1}{n} \sum_{k=1}^n \bar{x}^2 = \frac{1}{n} \sum_{k=1}^n x_k^2 - \bar{x}^2$$

データの総和は、式で書けばシグマ記号で終わりですが、C で計算する場合は、データの一つ一つを累積するという手順をプログラムにする必要があります。

アルゴリズム 8 $a_1, a_2, \dots, a_n$ の平均値を計算する

1. $z = 0, k = 1$ とする。
2. もし、 $k > n$ ならばステップ 5 へ。
3. $z + a_k$ を新たな z とする。
4. $k + 1$ を新たな k として、ステップ 2 へ。
5. z/n を平均値とする、アルゴリズム終了。

ステップ 1, 2, 4 は典型的な for 構造ですのでステップ 3 を for 文で囲めばプログラム完成です。67 ページのアルゴリズム 7 でも説明がありますが、そこでは (for 構文を説明する前だったので) while 構文を使っていました。

標本分散を計算する場合も、データの 2 乗の総和を計算する部分がプログラム化出来れば、あとは単純な計算です。標本標準偏差は数学関数の平方根関数 `sqrt` を使って計算します。

実習 30 データを入力する実習 29 のプログラムを利用してデータを入力し、標本平均、標本分散、標本標準偏差を計算して、その結果を表示する C のプログラムを書きなさい。平方根を計算するために数学関数ライブラリーをインクルードする必要があります。

実習 30 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **(変数の初期化)** データの累積値を計算するためには累積する変数を最初に 0 としておかなければいけません。これは**変数の初期化**とすることがあります。慣れたプログラマーでも忘れることがありますので注意しなさい。宣言文はプログラムでその変数を使うということを言っているだけです、その値（初期値）がどうなっているか不明です。使う前に必ずゼロを代入する必要があります。

(重要) 累積する場合は累積する変数を初期化すること。

- 2. **(初期化のタイミング)** 変数の初期化は変数を使う直前、つまり、for 文の直前に実行させるようにしなさい。変数に 0 を代入することは宣言文の中でもできるので、変数を宣言すると同時に初期化するプログラムを書く人が多いのですが、これは悪い習慣です。
- 3. **(平均値の計算)** 平均値を計算する場合、データが int 型の場合は要注意です。int 型変数で累積値を計算すると、(int 型の) データの個数で割ったとき、小数点以下が切り捨てられてしまい、正しい平均値が得られません。データが int 型でも、後で平均を計算することが分かっているような場合は、累積する変数を double 型にしておくのが良いでしょう。

練習問題 6.3 データが int 型なので、int 型変数の sum にデータを累積し、標本平均を計算するために「average = (double)(sum/n);」と書いたのですが正しい結果が表示されません。どこがいけないのでしょうか。

ヒント：索引で「キャスト」を調べなさい。

6.3 最大値、順位の計算

最小値は最大値の計算の符号を変えただけです。最大値の計算だけ説明します。コンピュータは人間と違って、いっぺんに三つ以上のものを比べることができません。そこで、順番に、1 番目のデータと 2 番目のデータを比べる、そのうちの大きいものと 3 番目のデータとを比べる、その結果の大きいものと 4 番目のものと比べる、... というようにひたすら二つの数の比較を繰り返す必要があります。

アルゴリズム 9 $a_1, a_2, \dots, a_n$ の最大値を求める

1. a_1 を仮の最大値として、変数 big に代入する。
2. big と a_2 を比べて、 a_2 が大きければ big を a_2 で置き換える。
3. big と a_3 を比べて、 a_3 が大きければ big を a_3 で置き換える。
4. big と a_4 を比べて、...
5. big と a_n を比べて、 a_n が大きければ big を a_n で置き換える。

この結果、big が $a_1, a_2, \dots, a_n$ の最大値になります。このように、順番に書いてみれば、ステップ 2 の 2 を k に置き換えたものを $k = 2, 3, \dots$ と順番に繰り返せば良いのだな、ということが分かります。ちゃんとした記述のアルゴリズムにする場合は、ステップ 2 の 2 を k に置き換えたものを for 文のような制御変数を使ったフィードバック構造で囲めば良いのです。

練習問題 6.4 「...」を使わないアルゴリズムを書きなさい。

練習問題 6.5 big の最初の値設定で、「big=a[1]」でなく「big=0」とし、 k を 1 から n までループさせるプログラムを書いたところ、時々おかしい結果が表示されます。何がいけないのでしょうか。

実習 31 (1) 練習問題 6.4 を参考に、最小値を計算するアルゴリズムを書きなさい。(2) それをもとに、入力された n 個のデータの最大値と最小値を求めるプログラムを書きなさい。(3) 先の実習 30 で作ったプログラムに、最大値、最小値、範囲を計算するルーチンを組み込み、プログラムを完成させ、入力して実行しなさい。「範囲」とは最大値と最小値の差です。

練習問題 6.6 最大値の入っている配列要素の位置（添え字）を表示するプログラムを書きなさい。(1,5,8,9,2 ならば「4」のように) データは添え字 1 から入っているものとします。

6.3.1 順位の計算

あるデータが全体の中で何番目に大きいか（相対順位）、ということを知りたい場合があります。全部大きさの順に並べ替えてみれば分かりますが、それほど大げさにしなくても、自分より大きなデータがいくつあるか勘定すれば、自分の相対順位が計算できます。自分より大きなデータが m 個あった場合は相対順位は $m + 1$ です。同じデータがあった場合でもこれで正しい相対順位が計算できます。

アルゴリズム 10 ある k に対して、 $a_1, a_2, \dots, a_n$ の中での a_k の相対順位を求める

1. $m = 1, i = 1$ とする。
2. $a_i > a_k$ ならば m に 1 加える。
3. $i + 1$ を新たな i として、 $i \leq n$ ならばステップ 2 へもどる。
4. さもなければ、 m が相対順位、アルゴリズム終了。

練習問題 6.7 $a[1], \dots, a[n]$ にデータが入力されているとき、 k を入力させて $a[k]$ の相対順位を計算するプログラムを書きなさい。

6.4 度数分布

データの範囲を適当に分割して、各小区間にデータがいくつつ含まれているのか集計したものを度数分布と言います。平均値、標準偏差のような基本統計量だけではつかみきれないデータの特徴を知ることが出来ます。

サイコロの目のように、取り得る値が比較的少数の離散値の場合は、各数値を取るデータの個数を集計すれば度数分布を得ることが出来ます。アルゴリズムは以下の通り。

アルゴリズム 11 $1, 2, \dots, m$ という値を取るデータ $a_1, a_2, \dots, a_n$ の度数分布を計算する。

1. $N_1 = N_2 = \dots = N_m = 0$ とする。
2. $i = 1, 2, \dots, n$ に対して、 N_{a_i} を 1 増やす。

アルゴリズムとして書くとなにやら七面倒くさそうですが、なに、子供でもやっていることですよ。

実習 32 (1) 上のアルゴリズムの記述にしたがって、離散データの度数分布を求めるプログラムを書きなさい。**(2)** それらをテストするために、練習 3.9 を参考にしてサイコロ振りで得られるような目の数を 100 個生成し、それらの度数分布を計算するプログラムを書きなさい。

m が大きくなったり、データが `double` 型数だったりすると、単純に集計したのでは同じデータが 1 個か 2 個しかないという、意味のない度数分布になってしまうので、そのような場合は、取り得る値を適当な区間に分割して、各区間に入るデータの度数を計算します。適当な区間 $c_0 < c_1 < c_2 < \dots < c_m$ を設定し、 $i = 1, 2, \dots, m$ に対して、 c_{i-1} 以上 c_i 未満のデータの個数を N_i とすれば、 $\{N_1, \dots, N_m\}$ が度数分布になります。 c_0 未満、あるいは c_m 以上になる値は**はずれ値**と呼ばれます。はずれ値も度数分布に取り込むために、 c_0 未満のデータの個数を N_0 、 c_m 以上のデータの個数を N_{m+1} も付け加えて $\{N_0, N_1, \dots, N_{m+1}\}$ とすれば完璧です。

アルゴリズム 12 `double` 型データ $a_1, a_2, \dots, a_n$ の度数分布を計算する。但し、区間の境界値を $c_0, c_1, c_2, \dots, c_m$ とする。 c_0 より小さいデータ、 c_m 以上のデータは、それぞれひとまとめにする。

1. $N_i = 0 (i = 0, 1, \dots, m+1)$ とする。 $c_{-1} = -\infty, c_{m+1} = \infty$ とする (∞ と言っても実際の無限大ではなく、扱うデータの範囲の下限、上限のつもりです)。
2. $k = 1, 2, \dots, n$ について以下の処理をする： $i = 0, 1, 2, \dots, m+1$ について、 $c_{i-1} \leq a_k < c_i$ となった i に対して N_i を 1 増やす。

度数分布を作るために、「もし c_0 より小さければ N_0 を 1 増やす、さもなければもし c_1 より小さければ N_1 を 1 増やす、さもなければもし c_2 より小さければ N_2 を 1 増やす、…」という条件判定が必要なので、各データに対して最大 $m+1$ 回の比較が必要です。

実際の集計では、小区間の幅は同じとすることが多いでしょう。その場合は比較を使わずに、データの分類される区間を「計算」することが出来ます。区間の幅を H とすると、 $c_i = c_0 + i \times H$ と書けるので、ステップ 2 の「 $c_{i-1} \leq a_k < c_i$ 」という条件は

$$c_0 + (i-1)H \leq a_k < c_0 + iH \Leftrightarrow i-1 \leq \frac{a_k - c_0}{H} < i$$

となります。したがって、はずれ値でなければ $\frac{a_k - c_0}{H}$ の整数部分に 1 を加えれば分類される区間の番号が計算できます。

アルゴリズム 13 double 型データ $a_1, a_2, \dots, a_n$ の度数分布を作成する。但し、各クラスの幅 H は等しいものとする（区間の最小値 c_0 、幅 H 、区間の数 m は与えられる）

1. $N_i = 0 (i = 0, 1, \dots, m+1)$ とする。
2. $k = 1, 2, \dots, n$ について以下の処理をする： $a_k < c_0$ ならば N_0 を 1 増やす。さもなければ $i = \min \{ \lfloor (a_k - c_0) / H \rfloor, m \} + 1$ を計算し、 N_i を 1 増やす（ $\lfloor x \rfloor$ はガウス記号とする、つまり、 x の整数部分を表す。 $\min \{, \}$ は最小値を求める関数）。

実習 33 アルゴリズム 13 にしたがって、度数分布を計算するプログラムを書きなさい。そのプログラムを使って、`rand()` を `RAND_MAX+1.0` で割ったものを二つ足した数を 1000 個生成してその度数分布を表示するプログラムを書きなさい。 $H = 0.1$ としなさい。

ヒント：ガウス記号はキャスト (`int`) を使えば良いでしょう。`min{...}` という関数は数学関数ライブラリーにはないので、条件文を使って計算する必要があります。

練習問題 6.8 度数に応じて "*" という文字を並べたもの（***** のように）を、**左端をそろえて**表示すると棒グラフのように見え、度数の多い少ないが一目で分かるようになります。そのような簡易棒グラフ（ヒストグラムと言います）を度数と一緒に表示するプログラムを書いて、実習 33 のプログラムに追加しなさい。

ヒント：例えば、「`for(k=0; k<m; k++) printf("*"); printf("\n");`」と書くと、 m 個の「*」が表示されます。 m が大きくて棒が 1 行に収まらない場合はどうしましょう？

練習問題 6.9 `rand()/(RAND_MAX+1.0)-0.5` によって生成される数を 12 個足したものをデータとして、度数分布を計算し、簡易ヒストグラムを描くプログラムを書きなさい。

6.5 数の表現と計算の正しさ

コンピュータを使った計算は、扱う数が有限桁しかないということから、思わぬ落とし穴が待ちかまえています。計算上注意が必要な事項をここでまとめておきましょう。

実習 34 次のプログラムを入力して実行しなさい。3行目の「%.01f」は「%」と「1f」の間に「.0」を挿入したものです。

プログラム例

```
// 計算の正しさ
1:      int k, n=100001;
2:      double y=100001;
3:      printf("%d * %d = %d, %.01f¥n", n, n, n*n,y*y);
4:      y = 1;
5:      n = 1;
6:      for(k=1; k<60; k++) {
7:          n *= 2;
8:          y *= 2;
9:          printf("2^%2d = %12d = %25.01f¥n", k, n, y);
10:     }
11:     y = 1;
12:     for(k=1; k<175; k++) {
13:         y *= k;
14:         printf("%3d! = %le¥n", k, y);
15:     }
16:     for(y=0; y<1; y+=0.1) {
17:         printf("y = %1f = %.171f¥n", y, y);
18:     }
```

実習 34 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. **(整数計算のオーバーフロー)** 3行目の結果を見てコンピュータがミスをしたとは思わないでください。コンピュータは数を変数に記憶すると言いましたが、記憶できる桁数には限りがあります。3行目の表示結果は $100001 \times 100001 \approx 10^{10}$ がその限度を超えていることを示しています。
- ☐ 2. **(ビット、バイト)** 限界を知るには、コンピュータが数をどのように記憶しているかを知る必要があります。コンピュータは電圧の強弱でデータを記憶するというのが基本的な仕組みです。強弱を 1,0 に置き換えれば、そこに 2 進数が並んでいると考えても良いことになります。その一つ一つを**ビット** (bit, Binary digIT、つま

り 2 進数字、の省略形)といい、そのビットを $8 = 2^3$ 個集めたものを**バイト**byte と呼び、これが基本単位になります。

- 3. (**int 型数の内部表現**) 1 バイトで表現出来る 2 進数の最大値は $11111111 = 2^8 - 1 = 255$ ですから、1 バイトで 256 種類の数記憶することが出来ます。通常のデータ分析で扱う数はとても 256 通りでは収まりきらないので、一挙に 4 バイト (32 ビット) を記憶の単位とすることにして、それを **int 型数**と名付けました。これで 10 進数 10 桁程度 ($2^{32} = 4,294,967,296 < 10^{10}$) の数が表現出来るようになりました。int 型では、負数も扱う必要があるので、半分を負の数とし、 $-2147483648(-2^{31})$ から $2147483647(= 2^{31} - 1)$ までの整数を表現することにしてあります。
- 4. (**整数演算の限界、精度**) int 型数同士の計算は 2 進法で実行され、範囲外の大きな数が出てきた場合は下 32 ビットを残して後は消されてしまいます。これを**オーバーフロー**と言います。言い換えれば 2^{32} で割った「あまり」を答えにするのです。3 行目の表示結果は 2^{32} で割ったあまりに等しいはずですが。といっても 2 進数のオーバーフローではピントこないの、分かりやすく示したのが 6 行目から 10 行目の結果です。確かに $2^{32} = 2^{33} = \dots = 0 \pmod{2^{32}}$ になっていますね。但し、 $a \bmod b$ は a を b で割った時のあまりを表す記号です。
- 5. (**浮動小数点表示、仮数、指数**) double 型数は、例えば 6.0221×10^{23} (アボガドロ数) や 1.3807×10^{-23} (ボルツマン定数) のように、非常に大きい数や小さい数を同時に扱えるようにするために考えられた表現法で、コンピュータでは、**仮数 $\times 2^{\text{指数}}$** という形式で記憶されています。6.0221 や 1.3807 を仮数、23 や -23 を指数と言います。人間にとっては $\times 10^n$ の形の方がわかりやすいのですが、コンピュータの世界は 2 進数なので、 $\times 2^n$ の形が使われます。
- 仮数と指数両方の数を記憶させるには 4 バイトでは不足なので、一挙に倍の 8 バイトに拡張しました。8 バイト、64 ビットの中で、指数部と符号で 11 ビット、仮数部が 53 ビットで表現されています。これが **double 型数**の表現です。表示結果を見ると、 $2^{57} \approx 10^{17}$ あたりから最下位ビットが怪しくなっていることが分かります。仮数部の正確な計算は 10 進数で 15 桁くらいとを考えてください。誤差を含まない桁を**有効桁**と言います。一方、指数部は 11 ビットのうち 1 ビットは符号に取られるので、上限は $2^{1024} \approx 10^{308}$ 、絶対値の下限は $2^{-1024} \approx 10^{-308}$ という範囲の数を扱うことが出来ます。
- 6. (**フォーマット指定子%le**) 11 行目から 15 行目では大きい数の見本として $k!$ を計算しています。 $70! > 10^{100}$ という大きな数を「%lf」で表示させると 2 行にまたがってしまうし、末端の桁まで正しいというわけではありません。またすべての数字が重要というわけではなく、先頭の何桁が分かれば良いので、仮数部の先頭の数桁と、指数の桁数を表示させています。「 $1.197857\text{e}+100$ 」というのは 1.197857×10^{100} の C 流の表現で、これを浮動小数点表示と言います。このように表示させるため

に「%le」（「パーセント」「エル」「イー」という新たなフォーマット指定子を使っています。なお、浮動小数点数を入力することはあまりないと思いますが、その場合も「%lf」と同じように「%le」とし、「6.022e23」のように入力します。^{*1}

- 7. **（実数計算の精度）** 16 ~ 18 行目の for ループでは 0 から 0.9 までは表示されるべきなのですが、実際の実行結果は 1.0 まで表示されています。これもコンピュータの「実数」がどのようなものであるかを暗示しています。17 行目の「%lf」は小数点 7 桁目以下を四捨五入したものを表示するので「1.0」ですが、「%.17lf」で表示させると「y+=0.1」を 10 回繰り返しても 1 には到達しないことが分かります。ここでも 2 進数の問題が絡んでいます。0.1 を 2 進小数に直すと（問題 7.1 参照）無限小数になるので、有限桁に納めるときに真の値よりわずかに小さくなってしまいます。したがって、それを 10 回足しても 1 にならないのです。ここでは「for(k=0; k<10; k++) {y=0.1*k;...}」のように、制御変数には int を使うべきです。そこで注意、

（重要）for 文の制御変数にはこのように実数型数を使ってはいけません。

9 行目の「%12d, %25.0f」、17 行目の「%.17lf」などの表示については、すでに 21 ページで説明しているので、そこを良く読みなさい（索引で「フォーマット指定子」、「表示桁数」を調べなさい）。

練習問題 6.10 2013 年 9 月にボイジャー 1 号が人類史上初めて太陽系圏を脱出して恒星間空間に飛び出したというニュースが流れていました。丸 36 年間かかったそうです。そこで問題、ボイジャーの秒速は 17 キロメートルということです。さて、現在地球から何キロくらい離れているのでしょうか、計算しなさい。int 型と double 型で計算して比較しなさい。

練習問題 6.11 「double x; for(x=1; x>0; x-=0.1) printf("%lf¥n", x);」と書いたとき、表示される内容を書きなさい。

^{*1} 「lf」は「long float」の頭文字です。昔メモリが貴重だった頃は、小数点付き数を 4 バイトで表現して、float 型と呼んでいました。4 バイトでは仮数部が圧倒的に不足していたので、倍の 8 バイトで表現する型を新たに作ったとき、long float 型と言えば良かったのに、それを double 型と言ってしまったのですね。「%le」のエルも同じ発想で、「long」のエルです。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の 3 段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ データの入力法、データの個数が既知の場合、未知の場合
- ☐ ☐ 標本平均の計算
- ☐ ☐ 標本分散・標本標準偏差の計算
- ☐ ☐ 最大値・最小値の計算
- ☐ ☐ 順位の計算
- ☐ ☐ 度数分布の作り方
- ☐ ☐ ヒストグラムの描き方
- ☐ ☐ `int` 型データの内部表現とオーバーフロー
- ☐ ☐ `double` 型データと浮動小数点表示

参考 実習プログラムの例（主要部分のみ）

実習 29（個数既知のデータ入力）のプログラム例（主要部分のみ）

```
scanf("%d", &n);
for(k=1; k<=n; k++) {
    scanf("%lf", &a[k]);
}
```

実習??（個数未知のデータ入力）のプログラム例（主要部分だけ）

```
n = 0;
do {
    scanf("%lf", &a[n+1]);
    if (a[n+1] <= 0) break;
    n++;
} while(n < 1000);
```

実習 30（標本平均と標本分散）のプログラム例（主要部分だけ）

```
double average, variance, sum, sum2;
```

```
sum = sum2 = 0;
for(k=1; k<=n; k++) {
    sum += a[k];
    sum2 += a[k]*a[k];
}
average = sum / n;
variance = sum2 / n - average * average;
```

実習 31 (最大値と最小値) のプログラム例 (主要部分だけ)

```
big = a[1];
small = a[1];
for(k=2; k<=n; k++) {
    if(a[k] > big) big = a[k];
    if(a[k] < small) small = a[k];
}
```

実習 33 (度数分布) のプログラム例 (主要部分だけ)

```
if (a[k] < c0) dosu[0]++;
else {
    i = (int)((a[k] - c0) / H) + 1;
    if(i > m+1) i = m+1;
    dosu[i]++;
}
```

6.6 章末演習問題

問題 6.1 採点競技で、極端な点数を排除するために、一番高い点と一番低い点を除いて平均値を計算することがあります。これを**トリム平均値**（刈り込み平均値）と言います。 n 個のデータの最大値と最小値を除いた $n - 2$ 個のトリム平均値を計算するプログラムを書きなさい。

問題 6.2 配列を使わずに、データを入力しながら、それまでに入力したデータの中の最大値と 2 番目に大きい数を、入力された順番とともに表示するプログラムを書きなさい。

問題 6.3 学籍番号順に並んでいる、数学、英語、プログラミングの試験成績を学籍番号（3 桁の整数）と共に入力し、合計点を計算し、順位を計算して、「学籍番号、順位、合計点、3 科目の点数」の一覧表を学籍番号順に表示するプログラムを書きなさい。

ヒント：順位は、すべてのデータに対して相対順位を計算するアルゴリズムを適用すれば良い。

問題 6.4 度数分布のヒストグラムを簡易棒グラフを使って表示するプログラムを書きなさい。但し、度数が多くなっても、最大度数の棒の高さ（*の個数）が 50 に収まるように、また度数が 0 でないクラスには「*」を少なくとも一つ表示するようにしなさい。例えば、最大度数が 154、最小度数が 1 という場合、度数 5 を一つの「*」で表すようにすれば、棒の高さは 31 ですむ。

ヒント：最大度数を求めて、例えば、最大度数が 50 以下ならば倍率は 1、50 を超えるようならば最大度数の高さが 50 になるようにして、全体を縮小しなさい。

問題 6.5 n 個の `double` 型数 $a_1, a_2, \dots, a_n$ を読み込んで度数分布を計算するプログラムを書きなさい。但し、下限値 c_0 とクラスの幅 H は入力されたデータから計算して、きりの良い数に決めるものとする。例えば、最小値が 28.1、最大値が 934.4 ならば $(c_0, H) = (28.1, 90.63)$ とするよりは $(c_0, H) = (0, 100)$ とした方がスマートです。また、クラスの個数については、 n がある程度大きければ「 $2 \log_2 n + 1$ に近い整数（スタージェスの公式）」という決め方があります。

ヒント：対数関数を使いなさい。

息抜きのページ

マスターマインド、あるいはヒットアンドブローという数当てパズルゲームです。

プログラム例

```
/* ヒットアンドブロー、あるいはマスターマインド (hit and blow)
 * 4桁の数字を当てるゲーム。数字と桁が合っていれば「ヒット」
 * 数字は合っている、桁が違う場合は「ブロー」
 * 4桁の試みの数字を入力すると、ヒットとブロー（ヒット以外）の個数が返ってくる
 * この情報を元に、最小の回数で正しい4桁の数を当てるのが目的。
 * ブローの数え方：正解と予想、それぞれの各数字の度数分布を計算し、最小値を合
計する
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int board[4], guess[4];
    int k, n, i, nb, ng, hit, blow, ato;
    srand((unsigned int) time(NULL) * 9998799);
    while(1) {
        printf("¥n ヒットアンドブローです（同じ数はありません） ¥n");
        // 出題
        for(k=0; k<10; k++) board[k] = k;
        for(k=0; k<4; k++) {
            i = rand() * 10 / (RAND_MAX+1);
            n = board[i];
            board[i] = board[k];
            board[k] = n;
        }
        ato = 12;
        do {
            // 予想
            printf("予想する 4 個の数をスペース区切りで入力しなさい  ");
            scanf("%d %d %d %d",
                , &guess[0], &guess[1], &guess[2], &guess[3]);

            // ヒットとブローの数を数える
            hit = 0;
            for(k=0; k<4; k++)
                if(guess[k] == board[k]) hit++;
        } while(ato > 12);
    }
}
```

```
        blow = 0;
        for(k=0; k<10; k++) {
            ng = nb = 0;
            for(i=0; i<4; i++) {
                if(guess[i] == k) ng++;
                if(board[i] == k) nb++;
            }
            blow += (ng < nb ? ng : nb);
        }
        // 判定
        if(hit < 4) {
            printf("ヒットは %d 個, ブローは %d 個でした。"
                  , hit, blow-hit);
            printf("残りはあと %d 回です。¥n¥n", --ato);
        } else {
            printf("あたり!!! ¥n¥n");
        }
    } while(hit < 4 && ato > 0);

    if( hit < 4)
        printf("残念!! 正解は %d %d %d %d でした ¥n¥n",
              board[0], board[1], board[2], board[3]);
    }
}
```

第 7 章

算法 3：数論（続）

この章では 4 章に続いて、数を対象とした代表的なアルゴリズム

- 2 進数と 10 進数
- 素数、エラトステネスの篩
- 大きな数のべき乗
- 暗号、RSA 暗号

について解説します。

7.1 2 進数と 10 進数

人間世界で使われている 10 進法、10 進数は、人間の指が 10 本あることが起源だと言われています。コンピュータの世界では電気信号の on-off が世界の中心にありますから、データも on-off に対応させて 1,0 の二つの数字だけから組み立てられています。それが 2 進法、2 進数です。10 進数の 234 は $2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$ の簡易表現ですが、それと同じで、2 進数 1011 は $2^3 + 0 \times 2^2 + 2^1 + 2^0$ の簡易表現です。10 進数と 2 進数の間の変換を考えるために、 n 桁の 10 進数

$$a_n \times 10^n + a_{n-1} \times 10^{n-1} + \dots + a_1 \times 10 + a_0$$

と、各桁の数値を並べた $n+1$ 次元ベクトル $\{a_n, a_{n-1}, \dots, a_1, a_0\}$ とを同一視します。ここで、 $a_0, a_1, \dots, a_{n-1}$ は 0 から 9 まで、 a_n は 1 から 9 までの数字です（ディジット digit という、デジタルの語源）。 m 桁の 2 進数

$$b_m \times 2^m + b_{m-1} \times 2^{m-1} + \dots + b_1 \times 2 + b_0$$

も同様に、 $m+1$ 次元ベクトル $\{b_m, b_{m-1}, \dots, b_1, b_0\}$ と同一視します。ここで、 $b_m = 1$ で、 $b_0, b_1, \dots, b_{m-1}$ は 0 か 1 です（ビット bit という、BInary digiT の略）。2 を p に変

えれば一般の p 進法、 p 進数も定義できます。

7.1.1 10 進数から 2 進数へ

10 進数を 2 進数表記に変えるためには、2 で割ってその「あまり」を計算するという操作を繰り返すことによって得られます（10 進数を 13 とすれば、2 で割ってあまりを取るという操作を続けると、1(商は 6), 0(3), 1(1), 1 となり、2 進数は 1101 となります）。但し、結果は下の位から順に計算されるので、表示させる場合はそれを逆順にしなければいけません。したがって、計算される各桁の数字を一旦配列変数を使って記憶させる必要があります。

アルゴリズム 14 10 進数 n を 2 進数 $\{b_m, b_{m-1}, \dots, b_1, b_0\}$ に変換する

1. $m = 0$ とする。
2. $n \div 2$ のあまりを b_m 、商を新たな n とする。
3. $n > 0$ ならば $m + 1$ を新たな m としてステップ 2 へ（ $n = 0$ ならばアルゴリズム終了）。

ステップ 2 の「 $\div 2$ 」を「 $\div p$ 」とすれば、10 進数から p 進数への変換アルゴリズムになることに注意してください。例えば、10 進数の 99 を 8 進数に直すと、 $99\%8 = 3$, $12\%8 = 4$ なので、143 になります。

実習 35 アルゴリズム 14 を C のプログラムに書いて、正しく動くことを確認しなさい。但し、2 進数は配列に記憶させるものとし、`printf` を使って新たな n と配列の変更箇所を表示させ、アルゴリズムの進行をチェックしなさい。

実習 35 の解説 チェックボックスに ☒ を入れながら読みなさい。

- ☐ 1. `for` 構文の構造をしているので、典型的な `for` 文を使おうとすると、制御変数に関する上限制約がないので戸惑うかもしれません。そこで、`for` を使うのをやめ、`do ... while` 構文を使い「`do { ... } while(n>0);`」とするのが自然な対処法ですが、`for` 構文をもう一度読み直すと、フィードバックする条件式は、なにも制御変数にこだわる必要がないことに気づきます。ということは「`n>0`」を条件式とすれば `for` 構文でも書けることになります。`for` 構文に慣れると、この書きの方がプログラムは読みやすいかもしれません。
- ☐ 2. 「`do { ... } while(n>0);`」のスタイルで書いた場合、アルゴリズム通りに書けば「`b[m] = n % 2; m++;`」という命令文がコードブロックの中に含まれますが、これを一つの命令文で「`b[m++] = n % 2;`」とする書き方があります。両者は全く同じ動きをします。C のプログラムを見ていると良く出てくるので覚えてお

くと良いでしょう。

7.1.2 2進数から10進数へ

2進数を10進数に直す場合は定義の式にしたがって $b_k \times 2^k$ をひたすら計算すればよいのですが、2のべき乗をその都度計算する代わりに、ホーナー Horner の方法と呼ばれる次のような計算方法が知られています。

$$\begin{aligned} b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2 + b_0 \\ = (((b_4 \times 2 + b_3) \times 2 + b_2) \times 2 + b_1) \times 2 + b_0 \end{aligned}$$

こうすることによって、べき乗の計算を回避することが出来ます。

アルゴリズム 15 2進数 $\{b_m, b_{m-1}, \dots, b_1, b_0\}$ を10進数に変換する

1. $n = b_m, k = m - 1$ とする。
2. $k < 0$ ならば、 n が答え、アルゴリズム終了。
3. $2 \times n + b_k$ を新たな n 、 $k - 1$ を新たな k として、ステップ2へ戻る。

ステップ3の「 $2 \times$ 」を「 $p \times$ 」とすれば、 p 進数を10進数に変換するアルゴリズムになることに注意してください。

実習 36 アルゴリズム15をCのプログラムに書いて、正しく動くことを確認しなさい。但し、`printf` を使って k と新たな n を表示させ、アルゴリズムの進行をチェックしなさい。

ヒント：ノーヒント

練習問題 7.1 10進数の各桁の数字を逆順に並べた数を生成し、元の数とその数の和を計算するプログラムを書きなさい。

7.2 素数、再訪

4.3 章で、ある数が素数かどうかを判定するプログラムを書きましたが、ここではある数以下の素数を一度にまとめて求める方法として良く知られた**エラトステネス**（Eratosthenes、人名）の**篩**（ふるいと読む、「ふるいにかける」のふるい）について説明します。原理はとても簡単です。ある数以下のすべての数の集合から2の倍数を除き（ふるい落とし）、3の倍数を除き、5の倍数を除き、7の倍数を除き、...、ということを繰り返せば約数のない数、つまり素数が求まるはずだ、ということです。この手順をアルゴリズム風にかくと次のようになります。

アルゴリズム 16 エラトステネスの篩（ふるい）による素数の生成

1. $a_k = k(k = 2, 3, \dots, N)$ とする、 $m = 2$ とする。
2. もし、 $m^2 > N$ ならばステップ5へ。
3. $a_{2m}, a_{3m}, \dots$ を0とする。
4. $m + 1$ を新たな m としてステップ2へ戻る。
5. $a_k(k = 2, 3, \dots, N)$ の中で0でないものが素数。

ステップ3の判定条件は、アルゴリズム2の考え方を使っています。このアルゴリズムは間違いではありませんが、かなり杜撰です。例えば、 $m = 4$ のとき、ステップ2で0にしようとしている変数は、すでに $m = 2$ の時に0になっていますので、不要です。ステップ2に「もし $a_m > 0$ ならば」という挿入句を入れるだけでずいぶん作業が短縮されます。また、 $m = 3$ のとき、 a_{2m} は $m = 2$ の時にすでにふるい落とされてゼロになっていますから、 a_{m^2} から始めれば十分です。というわけで、これらの考察の結果を取り入れた改良型を次に示します。

アルゴリズム 17 エラトステネスの篩（ふるい）による素数の生成（改良版、3行目以外と同じ）

1. $a_k = k(k = 2, 3, \dots, N)$ とする、 $m = 2$ とする。
2. もし、 $m^2 > N$ ならばステップ5へ。
3. もし $a_m > 0$ ならば、 $a_{m^2}, a_{m(m+1)}, \dots$ を0とする。
4. $m + 1$ を新たな m としてステップ2へ戻る。
5. $a_k(k = 2, 3, \dots, N)$ の中で0でないものが素数。

練習問題 7.2 $N = 100$ として、次の数表を使って、このアルゴリズムの手順に従い素数

を求めなさい。但し、素数でないものを 0 とする代わりに数字の上に $\times$ を付けなさい。

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100

実習 37 アルゴリズム 17 にしたがって素数を計算し、素数だけをきれいに整えて表示させるプログラムを書きなさい。そのプログラムを使って、1000 までの素数の個数を数えなさい。

3 桁の最大の素数は 997 です。検算に使いなさい。

練習問題 7.3 アルゴリズム 17 は N 以下の整数をすべてふるいの対象にしていますが、そもそも、最初から $a_k = k$ としないで、 $a_k = 2k + 1$ のように、偶数をふるい落としたところからスタートすれば、配列は半分の大きさで済みます。そのようなスリム化したエラトステネスの篩アルゴリズムを書き、それを実現するプログラムを書きなさい。

練習問題 7.4 上のプログラムを利用して 10000 までの素数を計算し、**(1)** k 番目の素数が `prime[k]` となるような配列を作りなさい。**(2)** 1 行に 10 個、5 行おきに空行が入るような素数表を作りなさい。

7.3 大きい数のべき乗計算

例えば、 12345^{6789} の下4桁の数はいくつですか、という問題を解くことを考えます。詳しい話は後の7.4節で説明しますが、これはインターネットで重要な暗号技術に密接に関係しています。実際の問題では、もっと大きな数のべき乗を使うのですが、ここでは、この程度の数にとどめておきましょう。この程度でも、理屈を知らないと結構大変な計算になりそうです。

一般に、5桁程度の三つの正整数 a, b, M が与えられたとき、 a^b を M で割ったあまりはいくつですか、という問題を解くのが目標です。数 n を M で割ったあまりを $n \bmod M$ と書くことにします。例えば、 $123 \bmod 11 = 2$ です。その計算は、123を11で割って11あまり2、だから答えは2、となります。 $13^4 \bmod 8$ をどうやって計算するか考えましょう。これくらいの数ならば、 $13^4 = 28561$ だから、それを8で割ってあまりは1、というように、 a, b が小さければ直接計算しても何の困難もありませんが、大きくなるにつれてすぐに破綻してしまうでしょう（ $12345^{6789} \bmod 10000$ っていくつ?）。そこで、数学の知識を使って問題を小さくすることを考えます。

$13 = 8 + 5$ となることから $13^4 \bmod 8 = (8 + 5)^4 \bmod 8 = 5^4 \bmod 8$ が成り立ちます（なぜでしょう?）。一般に $a = pM + q$ と書けたとすると

$$a^b \bmod M = (pM + q)^b \bmod M = q^b \bmod M$$

が成り立つので、一般性を失うことなく、最初から a は M より小さい数と仮定して構いません。

また、二つの数の積の mod 計算に対して次の式が成り立ちます（ $n = uM + v, m = wM + z$ と置いて代入すると分かるはず）。

$$nm \bmod M = ((n \bmod M)(m \bmod M)) \bmod M$$

この式を使えば、 $a^b \bmod M$ を、それと同じ答えを持つ規模の小さい問題に置き換えることが出来ます。実際、 $a^2 \bmod M = a'$ と置くと、

- b が偶数 ($b = 2u$) ならば、 $a^b = (a^2)^u = (a')^u$ と表され
- b が奇数 ($b = 2u + 1$) ならば、 $a^b = (a^2)^u \times a = a(a')^u$ となり、

結局、 $u(< b)$ 乗のあまりを計算する問題が解ければ元の問題の答えが見つかります。

この手順を繰り返せば、いつかは1乗のあまり（つまり元の数）を計算する問題にたどり着くことが出来ます。これは2進法の考え方を利用したものです。

例えば、 $12345^{6789} \bmod 10000$ を、この手順を適用して計算してみましょう。 $a =$

$12345, b = 6789, M = 10000$ です。

$$\begin{aligned}
 12345^{6789} \bmod M &= 2345^{6789} \bmod M = 2345(2345^2 \bmod M)^{3394} \bmod M \\
 &= 2345 \times 9025^{3394} \bmod M \\
 9025^{3394} \bmod M &= (9025^2 \bmod M)^{1697} \bmod M = 625^{1697} \bmod M \\
 625^{1697} \bmod M &= 625(625^2 \bmod M)^{848} \bmod M = 625^{849} \bmod M \\
 &\dots
 \end{aligned}$$

結局、最終的には二つの 4 桁の数を掛けて下 4 桁を取る、という計算を繰り返すだけで、答えが求まることが分かりました。

アルゴリズム 18 正整数 a, b, M が与えられたとき $a^b \bmod M$ を求める

1. $z = 1, q = a \bmod M$ とする。
2. もし $b = 0$ ならば z が答え、アルゴリズム終了。
3. b を 2 で割ったあまりを c 、商を新たな b とする。
4. もし $c = 1$ ならば $(zq) \bmod M$ を新たな z とする。
5. $q^2 \bmod M$ を新たな q として、ステップ 2 へ戻る。

練習問題 7.5 $a = 3, b = 11, M = 10$ として、アルゴリズム 18 を適用し、終了するまでに z, q, b がどのように変わっていくかを、コンピュータを使わずに計算して表にまとめなさい。

進行状態	z	q	b	c
初期値	1	3	11	—
ステップ 3, 4, 5	3	9	5	1
ステップ 3, 4, 5(2 回目)				
ステップ 3, 4, 5(3 回目)				
ステップ 3, 4, 5(4 回目)				

実習 38 アルゴリズム 18 を実現する C のプログラムを書き、 a, b, M を入力する命令を追加し、 z, q, b, c の値の変化を表示させる `printf` 文を適切な場所に挿入して、プログラムが正しく動くことを確認しなさい。また、検証用のテストデータ： $3^{21} \bmod 100 = 3, 8^{10} \bmod 100 = 24, 12345^{6789} \bmod 10000 = 5625$ を使って正しく動くことを確認しなさい。

実習 38 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **（アルゴリズム表記のすすめ）** アルゴリズム 18 はほとんど C のプログラムの一行一行に対応しています。アルゴリズムさえしっかり書いておけば、プログラムを書く手間、デバッグの手間は半減どころか、比較にならないくらい軽減されます。計算のロジックを考えてアルゴリズムをラフに設計してから、細かい手順書に上げていくのが、まだるっこしいようでも、長い目を見たとき、有効な方法です。そういう習慣を付けてください。
- 2. **（unsigned int 型）** int 型データの範囲は -2^{31} 以上 $2^{31} - 1$ 以下の整数ですが、この計算のように符号を必要としない計算では 2^{32} 通りの数を全部非負整数として扱った方が、扱える数の範囲が倍に広がります。unsigned int 型は int 型の符号部分 1 ビットをなくして 32 ビット全部を正の整数の表現に充てるというデータ型です。その結果、unsigned int 型データの範囲は 0 以上 $2^{32} - 1$ 以下の整数になります。
- 3. **（オーバーフロー）** このアルゴリズムは 2 つの整数の積の計算が必要ですが、6.5 節で説明したように、その結果が 2^{32} 以上になるとオーバーフローしたものは失われ、下 31 ビットだけが残ります。しかし、プログラムに書かれた数式そのものは間違いでもなんでもありませんので、最終的にもっともらしい数字が表示されますから、間違いを発見することはやっかいです。それを回避する簡便な方法は、すべての数を 2^{15} 以下に納めるか（unsigned int 型ならば 2^{16} 以下）、あるいは計算を double 型で実行しすべての数を 2^{28} 以下に納めることです。

練習問題 7.6 $a = 12345, b = 6789$ として、 $M = 100, 1000, 10000, 100000$ の場合の結果を比較検討しなさい。何が分かりますか。

練習問題 7.7 この実習プログラムで必ず正しい結果が出るような a, b, M の範囲を調べ、不正な入力データは再入力を促すような働きを付け加えなさい。

7.4 暗号を作る

秘密の通信をする場合に暗号を使うという話は誰でも1度は聞いたことがあるでしょう。最近ではインターネットでオンラインショッピングをするとき、名前やクレジットカードの番号を送る場合に「暗号を使っていますから安全です」というようなメッセージが表示されたりします。インターネットの情報は電気信号ですから、その気になれば全部「盗み読み」する事が可能です。もしクレジットカードの番号をそのままインターネットで送った場合は、他の人に盗み読まれて大きな被害に遭う危険性が出てきます。そこで、その情報をすぐには解読できないように暗号化する必要があるわけです。

暗号として有名なのはシーザー暗号と呼ばれる簡単な「換字法」です。次の文章はシーザー法で暗号化したものです。解読できますか。

しひうわじおふくえびこばけむそく

シーザー法は一定の規則で並べた文字を少しずらして読み替える、という方法です。例えば、五十音順に並んでいる文字を1文字ずらして、「あ」を「い」、「い」を「う」、... のように読み替えると「さよなら」は「しらにり」になります。2文字ずらすと「すりぬる」となります。規則を知らない人がこれを見たとき、「これは「さよなら」を暗号化したものだ」と判断できますか（上の暗号文は最後の「むそく」を意味ある言葉に変換できれば解読！）。メッセージを伝えたい人には事前にずらす字数を伝えておきます。そうすれば、暗号を受け取った人は簡単に元に戻せますが、暗号化のルールを知らない人にはちんぷんかんぷんです。^{*1}

暗号の世界ではこのずらす文字数のことを鍵、あるいはコードブックと言っています。暗号が暗号であるためにはこの「鍵」がメッセージの送り手、受け手に共有されなければいけません。そして、それは第三者に対して秘密にされなければ意味がありません。また、いつも同じ「鍵」だといずれバレてしまうでしょうから、時々は更新する必要がありますが、それは当事者同士で直接受け渡す必要があります。これが暗号を使う場合のアキレス腱と言われています。

最近もっとも注目を浴びている暗号化の方法はこのアキレス腱をなくしてしまった公開鍵暗号、あるいはRSA暗号と呼ばれている、とても巧妙な方法です。なにしろ、それまで「鍵」は秘密にしておくもの、秘密でなければ「鍵」とは言えない、ということを議論の余地無く当然のこととして受け入れていたのに、それを公開してしまおうというのですから、最初に発表されたときは大騒ぎでした。

普通の「鍵」をイメージして説明すると、今までの暗号は鍵のコピーを相手に渡して、その鍵で施錠したものを相手に届ける、という方法です。途中で暗号化された情報が盗ま

^{*1} シーザー暗号の例は、あいうえおの表で2文字ずつずらしたものです。「このあんごうは...」。

れたとしても、第三者は合い鍵でもなければ開けられないので、中身は解読されることはありませんが、鍵を第三者に対して秘密にしたままどこかで受け渡さなければいけません。

公開鍵暗号は二つの別々な鍵を必要とします。一つは秘密ですが、もう一つはコピーをたくさん作ってばらまいておきます。ばらまかれた（公開された）鍵は施錠出来ますが、一旦施錠したら秘密の鍵を使わないと開かない仕組みになっています。さて秘密の鍵を持っている人（Bさんとしましょう）にメッセージを送りたい場合はBさんのばらまいた鍵を使って施錠してBさんに送ります。Bさんは自分だけが持っている鍵を使ってその施錠を解くことができる、というわけです。秘密の鍵はBさんだけのものですから、秘密が第三者にもれるということはありません。

その鍵に「素因数分解」が出てくるといったらびっくりする人も多いでしょう。普通の人は、素因数分解とか素数なんて純粋に数学（算数）の話で、実用とはほど遠いものと思っていて当然ですね。しかし、整数論の知識を利用した次のような方法が、公開鍵暗号を実用的なものにするのに役立っています。簡単な例で説明しましょう。

例えば銀行口座の暗証番号のような1234という数を誰にも知られずに相手（Bさんとしましょう）に伝えたいという場合を考えましょう。Bさんは秘密鍵1367と公開鍵59を用意します。もう一つ公開されている数1817が必要なのですが、これは慣例で鍵とは呼ばれていません。このとき、送る側は 1234^{59} （1234の59乗）を1817で割ったときのあまりを計算して、それを1234の暗号文としてBさんに送ります。この場合は941になります。941を受け取ったBさんは自分の秘密鍵1367を使って 941^{1367} （941の1367乗）を1817で割った「あまり」を計算します。その結果がなんと、1234となり、みごとに最初の暗証番号が復元しました！！！？？（7.3節「大きなべき乗計算」で作ったプログラムを動かして確かめてください）

$$\begin{array}{ccc} 1234 & \rightarrow & 1234^{59} \bmod 1817 \\ \parallel & & \parallel \\ 941^{1367} \bmod 1817 & \leftarrow & 941 \end{array}$$

このなんとも不思議な計算は数論という数学の1分野の理論を使って正当化されます。先ず $N = 1817 (= 23 \times 79)$ は二つの素数の積ならば何でも良いとします。23, 79はたしかに素数です。次に、 $M = (23 - 1) \times (79 - 1) = 1716$ としたとき、秘密鍵、公開鍵の二つの数1367, 59は M と互いに素で、 $1367 \times 59 (= 80653)$ を $M (= 1716)$ で割るとあまりが1になるものであれば何でも良いものとします。

秘密鍵を決める手順を一般的に書くと次のようになります。二つの素数を p, q とし、 $pq = N, (p - 1)(q - 1) = M$ とします。そして M と互いに素となる二つの数 A, B で、その積を M で割ったあまりが1になるような数を求めると、それが二つの鍵になります。どちらを秘密鍵とし、どちらを公開鍵とするかは自由です。

秘密にしたい数を x としたとき、 A を公開鍵とすれば x の暗号 y は $y = x^A \bmod N$ に

なります。 y は $y^B \bmod N$ を計算すると x に等しくなるというのが上の計算結果です。最初の計算は「暗号化」、2 番目の計算は「復号化」と呼ばれています。なぜこれで暗号文が復号出来るのか、という部分が数論なのです（「任意の x に対して $x^M \bmod N = 1$ が成り立つ」という命題がこの暗号の根拠に使われています）。

これが *RSA* 暗号の原理です（*R, S, A* は考案者 3 人の頭文字です）。二つの素数が決まると、すべてのからくりが明らかになります。上の例でも 1817 が 23 と 79 の積に分解できる、ということが計算できると、公開鍵 59 から秘密鍵 1367 を計算することができるので（どうやって？）暗号システムが「解読」されたことになります。このように、公開されている N を二つの素数の積に分解することができれば秘密でも何でも無くなるのですが、もし N が 100 桁くらいの数だとすると、それを因数分解することは容易なことではありません。これがこの暗号システムの「強さ」の秘密です。

練習問題 7.8 3 桁くらいの素数を二つ見つけ、それをもとに *RSA* 暗号システムを作りなさい。それを使って友達と暗号通信を試してごらんください。

練習問題 7.9 59 に何を掛けたら、その積を 1716 で割ったあまりが 1 になりますか。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の 3 段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ p 進数
- ☐ ☐ 2 進数と 10 進数の相互変換
- ☐ ☐ エラトステネスの篩（ふるい）
- ☐ ☐ 大きい数のべき乗
- ☐ ☐ シーザー暗号
- ☐ ☐ （参考）RSA 暗号（公開鍵暗号）

参考 実習プログラムの例（主要部分のみ）

実習 35 (10 進数から 2 進数へ変換) のプログラム例（主要部分のみ）

```
// for 構文を使う場合
printf("%d の 2 進数は ", n);
for(m=0; n>0; m++) {
    b[m] = n%2;
    n = n/2;
}
for(k=m-1; k>=0; k--) printf("%d", b[k]);
// do ... while 構文を使う場合
m = 0;
do {
    b[m++] = n%2;
    n /= 2;
} while(n>0);
```

実習 37 (エラトステネスのふるい) のプログラム例（主要部分のみ）

```
for(i=2; i<=n; i++) pr[i] = i;
for(m=2; m*m<=n; m++) {
    if(pr[m] == 0) continue;
    for(i=m*m; i<=n; i+=m) pr[i] = 0;
```

```
}
k = 0;
printf("%d までの素数表\n");
for(i=2; i<=n; i++) {
    if(pr[i] > 0) {
        k++;
        printf("%5d", pr[i]);
        if(k%10 == 0) printf("\n");
    }
}
```

実習 38 (大きなべき乗の計算) のプログラム例 (主要部分のみ)

```
z = 1;                // ステップ 1
q = a % M;
while(b > 0) {         // ステップ 2
    c = b % 2;         // ステップ 3
    b /= 2;
    if (c > 0) {       // ステップ 4
        z = (z * q) % M;
    }
    q = (q * q) % M;   // ステップ 5
}
```

7.5 章末演習問題

問題 7.1 小数点以下の数にも2進数表現があります。10進数の0.1は

$$2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \dots$$

と書けますから、2進小数では0.0001100110011...という循環小数になります。0以上1未満の数を2進数表示するプログラムを書きなさい。

ヒント：2進小数 $0.a_1a_2a_3\dots$ は、

$$\frac{a_1}{2} + \frac{a_2}{2^2} + \frac{a_3}{2^3} + \dots$$

の各係数を並べたもので、 $a_1, a_2, a_3, \dots$ は0か1です。したがって、(10進小数を)2倍するとその整数部分は a_1 、それを除いて2倍するとその整数部分は a_2 、...のように、小数部分を2倍して整数部分を取り出す、ということを繰り返すことにより、 $a_1, a_2, a_3, \dots$ が取り出せます。

問題 7.2 10進数を8進数に変換するプログラムを書きなさい。10進数で $64 = 8^2$ は8進数で100、10進数で $100 = 64 + 32 + 4$ は8進数で144です。なお、フォーマット指定子「%o」はint型数を8進数で表示する場合に使います。これを検算に利用しなさい。

問題 7.3 10進数を16進数に変換するプログラムを書きなさい。ただし、16進数の一つの桁は16通りの記号が必要なので、0から9までと $A(10), B(11), \dots, F(15)$ を使いなさい。例えば、 $255 = 15 * 16 + 15$ なので、16進数で書けば FF 。16進数の一桁を表すために「char hex[]="0123456789ABCDEF";」という文字配列を使うと良いでしょう。文字を表示するフォーマット指定子は%cです。10進数で $267 = 16^2 + 11$ は16進数で10Bです。なお、フォーマット指定子「%p」はint型数を16進数で表示する場合に使います。これを検算に利用しなさい。

問題 7.4 非負の数 a を入力して、 a 以上 $a + 10000$ 以下の素数の個数を数えるプログラムを書きなさい。それを使って $a = 10000, 100000, 1000000$ としたときの結果を比較しなさい。

ヒント：エラトステネスのふるいを使いなさい。ふるい始める位置をどうやって計算するかが鍵です。10000以下の素数をあらかじめ計算して配列に記憶しておくとう良いでしょう。練習問題7.4参照。

第 8 章

算法 4：ソート・マージ

データの集合を大きさの順に整列させる作業を**ソート**sort と言います。回収したテスト解答用紙を学籍番号順に並べるとか、ある商品の販売価格を安いもの順に並べるとかがソートの例です。もっと大がかりなものとしては、google で検索したときのページの表示があります。該当するページが何万ページあろうと、ある数値基準（検索語との一致度）で検索語に最も近いものから順にソートして表示します。ソートはコンピュータ処理の基本と言っても良く、多くのアルゴリズムが提案されています。アルゴリズムの書き方に慣れるという目的からは格好の題材ですので、実際にプログラムを組みながら読み進めてください。取り上げられるアルゴリズムは以下の通り。

- 選択法ソート
- バブルソート
- 挿入法ソート
- マージ：挿入法、縊り合わせ（よりあわせ）法
- シェーカーソート
- シェルソート

8.1 選択法ソート

でたらめに並んだ数を大きさの順に整列させる場合、人間ならばざっと見渡して大きいものを選び出すというように、いくつかのものを同時に比較することが出来ますが、コンピュータの場合は同時に比較できる数は二つだけです。「二つの数の大小を比較して、必要ならば入れ替える」という操作だけを使って整列させる、というのがコンピュータによるソートのアルゴリズムを作る際の制約です。

数の集合を小さいもの順（**昇順**と言います、大きいもの順は**降順**）に並べ替えたい場合、もっとも単純な方法の一つは、一番小さなものを選んで先頭に置き、それを除いて一番小

さいものを次に並べ、その二つを除いて一番小さいものをその次に並べ、... とするのが自然でしょう。あるいは、逆に、一番大きいものを選んで最後に置き、それを除いて一番大きいものを最後から二番目に並べ、その二つを除いて一番大きいものをその前に並べ、... としても同じことです。

数列 $a_1, a_2, \dots, a_n$ を小さいもの順に並べるという問題を、最初の方法で解く場合の具体的な手順を説明しましょう。最初のステップは n 個の中の最小値を見つけることです。それが a_i だったとすると、次にやることは a_1 と a_i の中身を入れ替えることです。これで最小値が a_1 にセットされました。その次は（新たな） $a_2, a_3, \dots, a_n$ に対して今の手順を繰り返すことです。すなわち、 $a_2, a_3, \dots, a_n$ の最小値を見つけて、それと a_2 の中身を入れ替えるのです。以下同様。例えば、 $\{4, 6, 2, 5, 3, 1\}$ を並べ替える手順を追いかけて、数列の変化を一覧表にしたのが次の表です。

m	数列	$a_m, \dots, a_n$ の最小値	交換	交換の後
1	$\{4, 6, 2, 5, 3, 1\}$	1	a_1 と a_6	$\{1, 6, 2, 5, 3, 4\}$
2	$\{1, 6, 2, 5, 3, 4\}$	2	a_2 と a_3	$\{1, 2, 6, 5, 3, 4\}$
3	$\{1, 2, 6, 5, 3, 4\}$	3	a_3 と a_5	$\{1, 2, 3, 5, 6, 4\}$
4	$\{1, 2, 3, 5, 6, 4\}$	4	a_4 と a_6	$\{1, 2, 3, 4, 6, 5\}$
5	$\{1, 2, 3, 4, 6, 5\}$	5	a_5 と a_6	$\{1, 2, 3, 4, 5, 6\}$

一般的に、「 $a_m, a_{m+1}, \dots, a_n$ の最小値を見つけて、それと a_m の中身を入れ替える」という作業を $M(m)$ とすると、 $M(1), M(2), \dots, M(n-1)$ を順番に実行することによって、最終的に $a_1, a_2, \dots, a_n$ が昇順に並び変わることが分かるでしょう。C のプログラム風には書くと次のようになります。

```
for(m=1; m<n; m++) M(m);
```

これが**選択法ソート**と呼ばれる並べ替えのアルゴリズムです。このように、全体の構造をはっきりさせるために、細部を省略して $M(m)$ という記号にまとめ、骨格が分かるように書いたプログラム風の記述を**擬似コード**pseudo codeと言います。いきなりきちんとしたアルゴリズムを書く前に、このようなラフな書き方で記述してみると、問題の整理に役立ち、プログラミングの作業も短縮されるはずです。

作業 $M(m)$ はアルゴリズム風には書けば次のように書けるでしょう。

アルゴリズム 19 $M(m)$: $a_m, a_{m+1}, \dots, a_n$ の最小値と a_m を交換する

1. $i = m, j = m + 1$ とする。
2. $a_j < a_i$ ならば j を新たな i とする。
3. $j + 1$ を新たな j として、 $j \leq n$ ならばステップ 2 へ。
4. a_m と a_i の中身を入れ替える。

a_m が最小値の場合、ステップ 4 は実行する必要はありませんが、こう書いても間違いではありません。

選択法ソートのアルゴリズムは、この作業 $M(m)$ を使って次のように書くことができます。といっても、for 構文の復習を兼ねて、丁寧に書いてください（ n は普通 2 以上なので、必ず条件が満たされる最初の条件判定を省いてあります）。

アルゴリズム 20 選択法ソート： $a_1, a_2, \dots, a_n$ を小さいもの順（昇順）に並べ替える

1. $m = 1$ とする。
2. $M(m)$ を実行する。
3. $m + 1$ を新たな m として、 $m < n$ ならばステップ 2 にもどる。

作業 $M(m)$ のステップ 1,2,3 も for 構文で書けますから、結局プログラムは二重の for ループ（フィードバックループ）が必要になります。

実習 39 (1) アルゴリズム 20 に従って「2,1,4,3」というデータを昇順に並べ替えるとき、上のような表を作って、配列の中身がどのように変わっていくかを確かめなさい。**(2)** アルゴリズム 20 を実現する C のプログラムを書き、それが正しいことを検証するプログラムを追加して実行しなさい。

実習 39 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. **（擬似コードの奨め）** アルゴリズムにはきちんと書かなければいけないと同時に、読んで分かりやすくなければいけないという、場合によっては相容れない要請があります。選択法ソートのアルゴリズムはアルゴリズム 20 のステップ 2 の代わりにアルゴリズムを書き込んだものですが、いきなりそれを読まされても、制御変数の k, j がどのように動くのかすぐには分からないかもしれません。このように部分作業を記号にまとめて、それだけのアルゴリズムを別に作っておけば分かりやすいアルゴリズムを書くことができます。いきなり C のプログラムを書くのではなく、まずはアルゴリズムを書く、ややこしかったら擬似コードで書いてみる、という手間をかけることによってプログラミング作業がスムーズにはかどります。そういう習慣を身に付けてください。
- ☐ 2. **（地道な検証作業）** アルゴリズムが正しく動くことを確認するためには、一般の変数ではなく、具体的な小さな数、例えば「2,1,4,3」というデータがどのように並べ替えられていくのか、コンピュータを使わずに、プログラム 1 行 1 行の変数の動きを、前ページの表のように紙に書いていくことが必要です。
- ☐ 3. 作ったプログラムは、いったんこれが正しいと思いこんでしまったら、他人から見れば明らかな間違いでもなかなか見つからないものです。自分で間違いを発見できるような技術を身につけておくことが大切です。といっても、そんな難しい技術が

必要ということではなく、丁寧に調べるだけなのですがね。

- 4. (**変数の中身の入れ替え**) $a[m]$ と $a[i]$ を入れ替えるという作業は、人によってはかなり難航するようです。二つの変数の中身を入れ替えることを、二つのガラスの液体を入れ替える問題に置き換えて説明しましょう。ガラス A とガラス B の中身を入れ替えるために、別の二つのガラス C と D を用意して、ガラス A の中身をガラス C に、ガラス B の中身をガラス D に移し、その後 C の中身を B に、D の中身を A に移す、という手順を考える人が少なからずいます。間違いではありませんが、A の中身を C に移したあと、B の中身を直接 A に移してしまえば、ガラス D は必要ありませんね。こうして「 $b=a[i]$; $a[i]=a[m]$; $a[m]=b$;
」というプログラムができあがります。

練習問題 8.1 一番小さいものを最初に置く、という代わりに、「一番大きいものを最後に置く」という作業を繰り返しても昇順に並べ替えることが出来ます。「 $a_1, a_2, \dots, a_m$ の最大値を見つけて、それを a_m と入れ替える」という作業を $L(m)$ とすると、 $L(n), L(n-1), \dots, L(2)$ を順番に実行することによって、最終的に $a_1, a_2, \dots, a_n$ が昇順に並び変わることを確かめ、この手順を実現するアルゴリズムを書き、C のプログラムに書き換えなさい。

8.2 バブルソート

小さい順に整列しているならば、どんな隣同士の数を比較しても、必ず小さいものが先に来ているはずです。そこで、あらゆる隣同士を比較して、大きいものが先に来ていたら入れ替える、ということをひたすら繰り返せば、いつかは小さい順に並べ替えることが出来るのではないかと考えられます。

勝手な2数を比較するのではなく、系統的にやると、先頭から順に隣同士を比較して逆転していたら入れ替えるということを最後まで繰り返すと、最大値は常に入れ替えの対象となるので、この作業が終わったとき、最大値が最後に来ているはずです。もう一度、この作業を繰り返すと2番目に大きい数が最後から2つめに来るはずです。また、最後の数と比較する必要はないことに注意してください。以下同様に進行して、最終的に昇順に並んだ数列が得られます。

$a_1, a_2, \dots, a_n$ を先頭から順に隣同士を比較し、大きいものを後ろに送るという作業を $W(n)$ と名付けることにすると、 $W(n)$ を実行することによって n 番目の要素に全体の最大値が置かれることになります。次にやらなければいけないのは $W(n-1)$ です。なぜならば、 a_n は最大値ですので、隣同士で順番が逆転している可能性があるのは $n-1$ 番目までだからです。というわけで、 $W(n), W(n-1), \dots, W(2)$ の順に作業を続けると、 $a_1, a_2, \dots, a_n$ を小さいもの順に並べ替えることが出来ます。C のプログラム風によく書く次のようになるでしょう。

```
for(m=n; m>=2; m--) W(m);
```

例えば、 $\{4, 6, 2, 5, 3, 1\}$ を並べ替える手順を追いかけてみましょう。縦に見てください。

$W(6)$	$W(5)$	$W(4)$	$W(3)$	$W(2)$
$\{4, 6, 2, 5, 3, 1\}$	$\{4, 2, 5, 3, 1, 6\}$	$\{2, 4, 3, 1, \dots\}$	$\{2, 3, 1, \dots\}$	$\{2, 1, \dots\}$
$\{4, 6, 2, 5, 3, 1\}$	$\{2, 4, 5, 3, 1, 6\}$	$\{2, 4, 3, 1, \dots\}$	$\{2, 3, 1, \dots\}$	$\{1, 2, \dots\}$
$\{4, 2, 6, 5, 3, 1\}$	$\{2, 4, 5, 3, 1, 6\}$	$\{2, 3, 4, 1, \dots\}$	$\{2, 1, 3, \dots\}$	
$\{4, 2, 5, 6, 3, 1\}$	$\{2, 4, 3, 5, 1, 6\}$	$\{2, 3, 1, 4, \dots\}$		
$\{4, 2, 5, 3, 6, 1\}$	$\{2, 4, 3, 1, 5, 6\}$			
$\{4, 2, 5, 3, 1, 6\}$				

このように、隣同士を比較して、大きいものが後に来るように入れ替える、ということを繰り返すと、有限回後には全体が小さいもの順に並び変わります。このようなソートの方法を**バブルソート**と言います。表を左に90度回転させて、数字の一つ一つを水中の気泡(bubble)に見立てると、「6」がだんだん上に浮き上がり、ついで「5」が浮き上がり、というようにバブルが浮き上がっていくような動きをすることから、このような名前が付

いています。

アルゴリズムとして記述すると次のようになります。ステップ2から4までが $W(m)$ のアルゴリズムです。

アルゴリズム 21 バブルソートにより $a_1, a_2, \dots, a_n$ を小さいもの順（昇順）に並べ替える

1. $m = n$ とする。
2. $i = 1$ とする。
3. もし $a_i > a_{i+1}$ ならば a_i と a_{i+1} を入れ替える。
4. $i + 1$ を新たな i として、もし、 $i < m$ ならばステップ3へ。
5. $m - 1$ を新たな m として、もし $m \geq 2$ ならばステップ2へ。

ステップ1,5とステップ2,4は典型的なfor構文で、それぞれ「for($m=n$; $m \geq 2$; $m--$)」、「for($i=1$; $i < m$; $i++$)」で表現できます。但し、for文ならば、ステップ1の次、ステップ2の次に条件判定が必要ですが、 $m \geq 2$ ならば、最初の条件判定はクリアするので、アルゴリズムではその表記を省略してあります。変数の中味を入れ替える方法は選択法ソートのプログラムにあります。

実習 40 配列データをバブルソートによって昇順に並べるためのプログラムを作りなさい。データはint型、変数内容の入れ替えが起きるたびにprintf文を使って、配列内容がどう変わっていくのかを表示させなさい。完成したプログラムにテストデータ $\{a_i\} = \{4, 6, 2, 5, 3, 1\}$ を入力して、アルゴリズムの動きを確認しなさい。

練習問題 8.2 バブルソートでint型配列にあるデータを降順に並べ替えるプログラムを作り、正しく動作することをチェックしなさい。

練習問題 8.3 逆順に隣同士を比較して小さいものを前に出す、ということを繰り返しても昇順に並べ替えることができます。 $\{a_i\} = \{4, 6, 2, 5, 3, 1\}$ ならば、まず1が先頭に移り ($\{1, 4, 6, 2, 5, 3\}$)、次いで2が2番目に並び ($\{1, 2, 4, 6, 5, 3\}$)、... という動きをします。この手順によるバブルソートのアルゴリズムを書き、Cのプログラムを書きなさい。

8.3 挿入法ソート

最初から n 個のものを一度に並べ替えようと思わずに、最初二つの数だけで大小を比較して長さ 2 の整列している数列を作る、次に 3 番目の数を、全体が整列するように適切な位置に挿入して、長さ 3 の整列している数列を作る、というように、整列している数列の長さを増やしていくことによって最後に全体が整列するようにする、というようなソートの仕方があります。例えば $\{a_i\} = \{4, 6, 2, 5, 3, 1\}$ ならば、まず $\{4, 6\}$ を整列し（最初から整列済み）、次いで $\{4, 6, 2\}$ を $\{2, 4, 6\}$ と並べ替え、続けて、 $\{2, 4, 6, 5\}$ を $\{2, 4, 5, 6\}$ に並べ替え、という具合に進みます。

新たな数を挿入するとき、すでに整列している数列に挿入するだけですから、挿入箇所を見つけるためには、数列の全部の要素と比較する必要はなく、大きい方から順番に比較していったら、挿入箇所が見つかったら、そこから先は比較する必要がないということに注意してください。このようなソート方法は**挿入法**と呼ばれます。

すでに最初の m 個が小さいもの順に整列されている ($a_1 \leq a_2 \leq \dots \leq a_m$) ときに、新たな数 c を追加する、という作業を $I(m, c)$ として、そのアルゴリズムを考えてみましょう。

1. c と a_m を比べ、 c が大きければ a_{m+1} に c を代入して作業終了、さもなければ a_{m+1} に a_m を代入する、というのが最初の手順。
2. 終了していなければ、次いで c と a_{m-1} を比べ、もし c が大きければ a_m に c を代入して作業終了、さもなければ a_m に a_{m-1} を代入する、というのが次の手順。
これを順番に続け、
3. それまでに終了していなければ、 c と a_1 を比べ、もし c が大きければ a_2 に c を代入して作業終了、さもなければ a_2 に a_1 を代入、 a_1 に c を代入して全部終了。

例えば、 $\{4, 6, 2, 5, 3, 1\}$ を並べ替える手順を追いかけてみましょう。最初は 4 だけが整列していると考えます。作業は $I(1, 6)$ から始まります。最初の二つは整列済みですから、次は $I(2, 2)$ です。次いで、順に、 $I(3, 5)$, $I(4, 3)$, $I(5, 1)$ の順に実行すると、並べ替えが完了します。数列の動きをまとめたのが次の表です。この表で、同じ数値が並んでいるのは、上のアルゴリズムで「 a_m に a_{m-1} を代入」した直後だからです。

$I(1, 6)$	$I(2, 2)$	$I(3, 5)$	$I(4, 3)$	$I(5, 1)$
$\{4, \mathbf{6}, \dots\}$	$\{4, 6, \mathbf{2}, \dots\}$	$\{2, 4, 6, \mathbf{5}, \dots\}$	$\{2, 4, 5, 6, \mathbf{3}, 1\}$	$\{2, 3, 4, 5, 6, \mathbf{1}\}$
	$\{4, 6, 6, \dots\}$	$\{2, 4, \mathbf{5}, 6, \dots\}$	$\{2, 4, 5, 6, 6, 1\}$	$\{2, 3, 4, 5, 6, 6\}$
	$\{\mathbf{2}, 4, 6, \dots\}$		$\{2, 4, 5, 5, 6, 1\}$	$\{2, 3, 4, 5, 5, 6\}$
			$\{2, \mathbf{3}, 4, 5, 6, 1\}$	$\{2, 3, 4, 4, 5, 6\}$
				$\{2, 3, 3, 4, 5, 6\}$
				$\{1, 2, 3, 4, 5, 6\}$

作業 $I(m, c)$ は整列している長さ m の数列と一つの数 c を、整列している長さ $m+1$ の数列に置き換えています。ということは、この作業を $m = 1, 2, \dots$ と繰り返すことによって任意の長さの整列した数列を作り出すことも出来るということになりませんか。その通り、 $I(1, a_2), I(2, a_3), \dots, I(n-1, a_n)$ と続けることによって数列 $a_1, a_2, \dots, a_n$ を昇順に並べることが出来ます。

```
for(m=1; m<=n-1; m++) I(m, a[m+1]);
```

実際、 $I(1, a_2)$ を終わると（新しい） a_1 と a_2 が昇順（ $a_1 \leq a_2$ ）に並び、 $I(2, a_3)$ を終わると $a_1 \leq a_2 \leq a_3$ 、 $I(3, a_4)$ を終わると $a_1 \leq a_2 \leq a_3 \leq a_4$ 、... となることが分かるでしょう。

平均的には $I(m, c)$ の作業に必要な比較の回数は m の半分で済みますから、データの個数が大きくなると、バブルソートよりは早くソートが完了することが直感的に理解できます。アルゴリズムの形にまとめておきましょう。

アルゴリズム 22 挿入法ソートにより $a_1, a_2, \dots, a_n$ を小さいもの順（昇順）に並べ替える

1. $m = 1$ とする。
2. $c = a_{m+1}, i = m$ とする。
3. もし、 $c \geq a_i$ ならばステップ5へ、さもなければ、 $a_{i+1} = a_i$ とする。
4. $i - 1$ をあらたな i とする。もし $i \geq 1$ ならばステップ3へ。
5. $a_{i+1} = c$ とする。
6. $m + 1$ をあらたな m として、もし $m \leq n - 1$ ならばステップ2へ戻る。

ステップ2,3,4,5が $I(m, a_{m+1})$ の作業に対応することを確認してください。これも典型的な for 構文ですが、最初の条件判定は省略されていると考えてください。

実習 41 (1) $\{2, 4, 5, 3, 6, 1\}$ を昇順に並べるために挿入法を使うとして、どのような経過をたどって答えに到達するのか、コンピュータを使わずに、紙に書いてごらんない。(2) アルゴリズム 22 を実現する C のプログラムを書き、そのプログラムが正しく動くことを検証するプログラムを追加して実行しなさい。途中経過を printf 文を使って表示させること。

練習問題 8.4 経営実験で課題を終えた学生の学籍番号（3桁の整数）を逐次入力し、それまでに課題を終えた学生の学籍番号を昇順に並べた一覧表を表示するプログラムを書きなさい。

練習問題 8.5 n 個のデータが $b[0], b[1], \dots, b[n-1]$ に記憶されている場合に、それを挿入法によって降順にソートするアルゴリズムを書き、それを実現する C のプログラムを書きなさい（添え字の動き方に注意）。

8.4 マージ

二つの整列した数列（例えば、「1,4,5」と「2,3,8,9」）を一つの整列した数列にまとめる作業を**マージ**mergeと言います。二つの数列を併せて一つの整列した数列を作るのであれば、一方の数列の後に他方の数列を並べて一つの数列としてからそれをソートすれば良いので、特に新しいアルゴリズムは必要でないと思われます。しかし、すでに部分的に並んでいるという情報を使わない手はありません。

ある長い整列した数列に小数のデータを追加するという場合は、挿入法が使われます。この場合、追加のデータについては整列しているという情報は使わないので、整列してなくても構いません。二つの数列が共に大きい場合は、両方の整列情報を使った有効な方法があるので、それについて解説しましょう。

8.4.1 挿入法

長さ n の昇順に並んだ数列 $a_1 \leq a_2 \leq \dots \leq a_n$ に（少数の）整列しているとは限らないデータ $b_1, b_2, \dots, b_m$ を追加して一つの昇順のリストを作る場合は、挿入法ソートで定義した作業 $I(k, x)$ を繰り返し適用する、という方法が有効です。実際、 $\{a_1, a_2, \dots, a_n\}$ に対して $I(n, b_1)$ を実行すると $a_1, a_2, \dots, a_n, b_1 (= a_{n+1})$ が昇順に並び、 $I(n+1, b_2)$ を実行すると $a_1, a_2, \dots, a_n, a_{n+1}, b_2$ が昇順に並び、... という手順を繰り返すことで、 $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m$ を昇順に整列させたものが得られるからです。この手順を擬似コードで書くと次のようになるでしょう

```
for(k=1; k<=m; k++) I(n+k-1, b[k]);
```

練習問題 8.6 上の挿入法によるマージの方法を実現するアルゴリズムを書きなさい。

実習 42 上の練習問題で書いたアルゴリズムを実現する C のプログラムを書き、二つの配列データをマージするプログラムを完成させ、それが正しく動くことを検証するためのプログラムを付け加えて実行しなさい。途中経過を `printf` 文を使って表示させること。

8.4.2 縊り合わせ（よりあわせ）法

挿入法アルゴリズムでは、 $\{a_k\}$ が整列済みであるということを利用していますが、 $\{b_k\}$ が整列済みであることは仮定していません。 $\{a_k\}$ も $\{b_k\}$ も整列している場合は、両方を対等に扱い、小さいものから順に取り出して並べていく、という方が賢そうです。

二つの列の先頭同士を比較して、小さい方を列から取り出して第三の列に付け加える、ということを繰り返せば、二つの列が空になったところで第三の列に昇順に並んだものが

できます。普段何気なくやっているやり方でも、これを規則化して、コンピュータに教えるとなると、細かいところに神経を使わなくてははいけません。

$a_1, a_2, \dots, a_{i-1}$ と $b_1, b_2, \dots, b_{j-1}$ がすでに $c_1 \leq c_2 \leq \dots \leq c_{k-1}$ のように整列しているものとします。ただし、 $k = i + j - 1$ です。

1. もし $a_i \leq b_j$ ならば、 $c_k = a_i$ として、 $i + 1$ を新たな i とする。
2. さもなければ $c_k = b_j$ とし、 $j + 1$ をあらたな j とする。

という作業を $T(k)$ とすれば、 $i = j = 1$ として $T(1), T(2), \dots, T(n + m)$ を順番に実行することにより、 $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m$ の中の 1 番小さいもの、2 番目に小さいもの、... が順番に取り出されて一つの昇順に並んだ数列が生成されることが分かります。ちょうど 2 本の糸をより合わせていくような動きをするために、このアルゴリズムは**縫り合わせ法**と呼ばれます。

```
for(k=1; k<=n+m; k++) T(k);
```

しかし、丁寧にチェックしてみると、これだけでは不都合が生じることが分かります。実際、 $\{a_i\} = \{1, 2\}, \{b_i\} = \{3, 4\}$ として、この手順を追いかけると、 $(i, j) = (1, 1), (2, 1), (3, 1)$ と変化するので、3 巡目に定義されていない a_3 を参照することになるからです。したがって、 $i > n, j > m$ となった場合の処理を追加する必要があります。

追加処理は面倒なので、 a_{n+1}, b_{m+1} にすべての数よりも大きな数をあらかじめセットしておく、という簡易的方法があります。int 型ならば $2^{31} - 1 = 2147483647$ とすれば十分でしょう。但し、そのためには、配列要素を一つ余分に確保しておく必要があります。

練習問題 8.7 縫り合わせ法によるマージの完全なアルゴリズムを書きなさい。

実習 43 次のプログラムは縫り合わせ（よりあわせ）法マージのプログラムの主要部です。データの入出力部分を追加して全体を完成させ、入力して実行しなさい。

プログラム例

```
// 二つの配列をマージする（縫り合わせ法、昇順）
1:  int a[1000], b[1000], c[1000], n, m, k, j, i;
2:  a[n+1] = b[m+1] = 2147483647;
3:  i = j = 1;
4:  for(k=1; k<=n+m; k++) {
5:      if (a[i] <= b[j]) c[k] = a[i++];
6:      else c[k] = b[j++];
7:  }
```

実習 43 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. 2行目の命令は「 $a[n+1] = 21\dots; b[m+1] = 21\dots;$ 」という二つの命令を一緒にまとめたもので、正式に認められた記法です。いくつかの変数に同じ値を代入する場合、二つの等式を書くよりは、このように書いた方が、何をしているかが分かりやすいでしょう。「2147483647」は $2^{31} - 1$ で、`int` 型数の最大値です。
- 2. 5行目の代入文は「 $c[k] = a[i]; i++;$ 」という二つの命令を一緒にまとめたもので、正式に認められた記法です。5,6行目が作業 $T(k)$ を表していることを確認しなさい。

練習問題 8.8 $a[n+1], b[m+1]$ を利用してはいけない、という制約があった場合、上の実習プログラムをどのように書き換えればよいですか。

8.5 その他のソートアルゴリズム

興味がある人はインターネットで調べてみてください、まだいくらでもあります。ここではあと二つを紹介します。

8.5.1 シェーカーソート（双方向バブルソート）

バブルソートは「大きいものは右へ」ということを繰り返していましたが、同時に、「小さいものは左へ」という操作でも同じことですので、それらを交互に行うようにしたのが、シェーカーソートです。シェーカーはカクテルを作るときに使う道具で、行ったり来たりイメージからのネーミングでしょう。

アルゴリズム 23 シェーカーソートにより $a_1, a_2, \dots, a_n$ を小さいもの順（昇順）に並べ替える

1. $\text{left} = 0, \text{right} = n - 1, \text{dir} = 1$ とする。
2. ($\text{dir} = 1$ の場合) $i = \text{left}$ から $i = \text{right} - 1$ まで「もし a_i が a_{i+1} より大きかったら a_i と a_{i+1} を入れ替える」ということを繰り返す。最後の入れ替えが起きた i を right とする。入れ替えが起きなかったら $\text{right} = \text{left}$ とする。
3. ($\text{dir} = -1$ の場合) $i = \text{right}$ から $i = \text{left} + 1$ まで「もし a_i が a_{i-1} より小さかったら a_i と a_{i-1} を入れ替える」ということを繰り返す。最後の入れ替えが起きた i を left とする。入れ替えが起きなかったら $\text{left} = \text{right}$ とする。
4. もし $\text{left} < \text{right}$ ならば $\text{dir} = -\text{dir}$ としてステップ 2 へもどる。

練習問題 8.9 数列 $\{4, 6, 3, 8, 2, 1, 7, 5\}$ をシェーカーソートで昇順に並べ替えようとすると、どのような経過をたどりますか、コンピュータを使わずに紙に書いてご覧なさい。

練習問題 8.10 アルゴリズムにしたがってシェーカーソートのプログラムを書き、正しく動くことを確認しなさい。但し、変数が変わるたびに `printf` を使って配列を表示させ、アルゴリズムの進行とともに配列の並び方がどのように変わっていくのかをチェックしなさい。テストデータとしては、順番の入れ替わりさえ確認できればよいので、`int` 型とし、1 から n までの整数を適当に入れ換えたものを使いなさい。

ヒント： `while(left < right) {if (dir > 0) {for(j=left,...)`

8.5.2 シェルソート

ソートは数列の長さが長くなると、その手間が指数的に増大します。また、挿入法は「ある程度」整列している数列をきちんと整列するための効率的な方法として知られています。そこで、最初にとびとびの（長さの短い）部分列をソートして、あらかじめ小さいものは前の方に、大きいものは後ろの方に集まるような前処理をしてから全体を（挿入法で）ソートする、という方法で手間を省くことができるだろうと考えたのが、シェルさんの考えたシェルソートです。前処理もソートは挿入法で実行します。

部分列を抽出するために、 $k_1 = 1$ として、 $k_i = 3k_{i-1} + 1$ によって生成される数列を $k_i \leq \frac{n}{9}$ となるまで計算し、その最大の添え字 ($k_i \leq \frac{n}{9} < k_{i+1}$ を満たす i) を m とします。まず k_m おきの部分列 ($a_1, a_{1+k_m}, a_{1+2k_m}, \dots; a_2, a_{2+k_m}, a_{2+2k_m}, \dots$ など、全部で k_m 個ある) を挿入法で整列させます。次いで、 k_{m-1} 個おきの部分列 (全部で k_{m-1} 個ある) を挿入法で整列させます。以下同様に部分列を長くしながら、最後は全体 ($k_1 = 1$ 個おき) を挿入法で生成すれば完成。例えば、 $n = 100$ 程度ならば、 a_1, a_2, a_3, a_4 から始まる 4 個おきの 4 つの部分列を挿入法でソートしてから、全体を挿入法でソートします。実際に $\{k_i\}$ を計算してみればわかりますが、 $n = 30$ 程度ならば普通の挿入法と変わりません。

アルゴリズム 24 シェルソートにより $a_1, a_2, \dots, a_n$ を小さいもの順 (昇順) に並べ替える。

1. $k_1 = 1$ として、 $k_i = 3k_{i-1} + 1, i = 2, 3, \dots$ という数列を作り、 $k_i \leq \frac{n}{9}$ を満たす最大の k_i を k_m とする。
2. $j = 1$ から k_m まで $L = \lfloor (n - j) / k_m \rfloor$ として、「 $a_j, a_{j+k_m}, a_{j+2k_m}, \dots, a_{j+Lk_m}$ を挿入法で昇順に並べる」ということを繰り返す。但し、 $\lfloor x \rfloor$ は x の整数部分を表す記号とする。
3. $m > 1$ ならば、 $m - 1$ をあらたな m としてステップ 2 へもどる。

練習問題 8.11 $n = 36$ として、 $a_i = 37 - i$ としたとき、最初のステップ 2 を終了した時点で数列の順序がどのようになるか計算しなさい。単純な挿入法で整列する場合と比較してどういう改良がなされたのかを説明しなさい。

練習問題 8.12 上のアルゴリズムにしたがってシェルソートのプログラムを書き、 $n = 40$ のときに正しく動くことを確認しなさい。但し、変数が変わるたびに `printf` を使って配列を表示させ、アルゴリズムの進行とともに配列の並び方がどのように変わっていくのかをチェックしなさい。テストデータとしては、順番の入れ替わりさえ確認できればよいので、`int` 型とし、1 から n までを適当に入れ替えた数列を使いなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ 昇順ソート、降順ソート
- ☐ ☐ 選択法ソート
- ☐ ☐ 擬似コード
- ☐ ☐ 二つの変数の中身の入れ替え
- ☐ ☐ バブルソート
- ☐ ☐ 挿入法ソート
- ☐ ☐ 挿入法マージ
- ☐ ☐ 縊り合わせ（よりあわせ）法マージ

参考 実習プログラムの例（主要部分のみ）

実習 39（選択法によるソート）のプログラム例（主要部分のみ）

```
for(m=1; m<n; m++) {  
    i = m;  
    for(j=m+1; j<=n; j++) {  
        if(a[j] < a[i]) i = j;           // ステップ 2  
    }  
    c = a[i], a[i] = a[m], a[m] = c;    // ステップ 4  
}
```

実習 40（バブルソート）のプログラム例（主要部分のみ）

```
for(m=n; m>=2; m--) {  
    for(i=1; i<m; i++) {  
        if(a[i] > a[i+1]) {             // ステップ 3  
            c = a[i], a[i] = a[i+1], a[i+1] = c;  
        }  
    }  
}
```

実習 41 (挿入法によるソート) のプログラム例 (主要部分のみ)

```
for(m=1; m<n; m++) {
    c =a[m+1];
    for(i=m; i>0; i--) {
        if(c >= a[i]) break;    // ステップ 3
        a[i+1] = a[i];        // ステップ 3
    }
    a[i+1] = c;                // ステップ 5
}
```

実習 42 (挿入法によるマージ) のプログラム例 (主要部分のみ)

```
for(k=1; k<=nb; k++) {
    m = n + k - 1;
    c = b[k];
    for(i=m; i>0; i--) {
        if(c >= a[i]) break;
        a[i+1] = a[i];
    }
    a[i+1] = c;
}
```

140 ページの続き

```
    a = 1 - a;
    printf("残り %d 個です¥n", n);
} while(n>0);
if(a == 1) printf("わたしの勝ち¥n");
else printf("あなたの勝ち¥n");
printf("続けますか？ (1.. はい、0.. いいえ)  ");
scanf("%d", &a);
if(a == 0) break;
}
}
```

8.6 章末演習問題

問題 8.1 $n(< 100)$ 人分の身長と体重のデータが通し番号付きで 2 次元の配列 `double taikaku[][]` に記憶されているものとします。このとき、各人の BMI を計算し、BMI の大きいもの順に並べ替えて、 n 人分の「通し番号、身長、体重、BMI」の一覧表を作成するプログラムを書きなさい。 $n = 5$ として、適当な数値を入力し、プログラムが正しく動くかどうかをチェックしなさい。BMI の定義は索引を調べなさい。

ヒント：Excel のように、2 次元の配列をイメージすると分かりやすいかもしれません。Excel で、ある列を基準にソートすると、行単位で入れ替えが起きます。BMI を並べ替えるとき、ついでに通し番号、身長と体重も並べ替えれば良いでしょう。

問題 8.2 1 から n までの数字をランダムに並べ替えた数列（**ランダム置換**という、トランプカードをよく切って並べたようなもの）を生成するプログラムを書き、正しく動くことをチェックしなさい。例えば、 $n = 5$ ならば「1,4,3,5,2」のように。

ヒント：試験の成績順に並べ替えた学籍番号はランダムに並ぶ？ 試験の点の代わりに乱数を入力したらどうなる？

問題 8.3 1 以上 99 以下の数の中からランダムに 24 個取り出し、大きさの順に表示するプログラムを書きなさい。できれば、一行に 5 つずつ、但し、3 行目だけ 4 つとし、3 行目の 2 番目と 3 番目の数字の間にゼロを挿入して表示すること。

ヒント：一つ一つランダムに取り出すと考えずに、1 から 99 までの数のランダム置換を作って先頭の 24 個を取り出せば良い。

3	12	18	19	20
22	24	28	30	34
37	40	0	46	63
68	70	71	77	84
85	87	88	89	95

息抜きのページ

石取りゲームです。最後の一つを取ったら勝ち。

プログラム例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
// 石取りゲーム（山崩し）
int main() {
    int n, m, a;
    srand ((unsigned int) time (NULL)*314159265);
    printf("石取りゲームをしましょう\n");
    printf("1 回に 1 個以上 3 個以下の石をとらなければいけません\n");
    printf("最後の石を取った方が勝ちです\n");
    while(1) {
        printf("先攻なら 1、さもなければ 0 を入力しなさい  ");
        scanf("%d", &a);
        n = rand() % 16 + 10;
        printf("石の個数は %d 個です\n", n);
        do {
            if(a == 0) {
                if(n%4 != 0) m = n%4;
                else m = rand()%3 + 1;
                printf("わたしは %d 個取りました\n", m);
                n -= m;
            } else {
                printf("取る石の個数を入力しなさい  ");
                do {
                    scanf("%d", &m);
                    if(m>=1 && m<=3) break;
                    printf("反則です、もう一度  ");
                } while(1);
                n -= m;
            }
        }
    }
}
```

この続きは 138 ページへ。

第 9 章

関数定義

文法編その 4

関数は C 言語の基本構造です。今まで何も意識せずに使ってきた「main」も「printf」も「scanf」も関数の一つの例です。その関数についてまとめて解説します。関数をうまく使いこなせれば、プログラミングの作業効率が劇的に改善されるでしょう。主な実習項目は以下のようなものです。

- **関数の定義**：関数値と return 文、void 型関数、プロトタイプ
- **実引数と仮引数**：配列データの引き渡し
- **ポインタ**：アドレス渡し、アドレス演算子、間接参照、間接演算子

9.1 関数定義プログラム

関数といえば、すでに `sqrt(x)` や `pow(a,n)` などの数学関数を知っています。6 章で学んだ平均値や最大最小値などの統計計算も、Excel のように関数になっていれば、細かい計算式を気にせず、関数呼び出しだけで結果を得ることができて便利です。ない関数は自分で作ってしまう、というのが C の流儀です。

文法予備知識

1. 自分で、数学関数のような関数を定義することができる。
2. 関数を定義するプログラムは、「`double myMax (double x, double y)`
`{...}`」のように、「関数の型、関数名、独立変数のリスト」を指定してから、「`{...}`」の中に関数値を計算する実行文を書く。
3. 「`return`」の後に計算された関数値を指定する。

実習 44 次に示すプログラム 1、プログラム 2 を同じソリューションの別々のプロジェクトとして作成し、それぞれを実行して、その結果を比較しなさい。スタートアッププロジェクト設定を忘れないように

プログラム例

// 最大値を求める：プログラム 1

```
1: int main() {
2:     double a, b, c;
3:     scanf("%lf %lf", &a, &b);
4:     if(a > b) c = a;
5:     else c = b;
6:     printf("%lf と %lf の最大値は %lf¥n", a, b, c);
7:     system("pause");
8: }
```

// 最大値を求める：プログラム 2

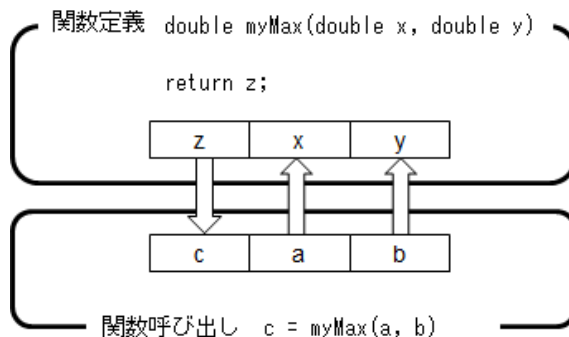
```
1: double myMax(double x, double y) {
2:     double z;
3:     if(x > y) z = x;
4:     else z = y;
5:     return z;
6: }
7: int main() {
8:     double a, b, c;
9:     scanf("%lf %lf", &a, &b);
10:    c = myMax(a, b);
11:    printf("%lf と %lf の最大値は %lf¥n", a, b, c);
12:    system("pause");
13: }
```

追加の実習 プログラム 2 の 1 行目から 6 行目を 13 行目の後に移動させてビルドするとどうなるか調べなさい。

関数プログラムを利用するためにデータの受け渡しの仕様（約束事）を理解することが必要です。次の図を見ながら説明を読みなさい。四角が変数の記憶域に対応しており、矢印はデータのコピーが転送されるという意味です。

実習 44 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. プログラム 1 と、プログラム 2 では、表示結果が全く同じになります。プログラム 2 の 3,4 行目の計算手順はプログラム 1 の 4,5 行目のそれと（変数名は違いますが）



全く同じです。プログラム 2 は、プログラム 1 の実質計算部分 (4,5 行目) を (数学関数を呼ぶように) `c=myMax(a,b)` という命令で置き換え、追い出された 4,5 行目は前後に「何か」を付加して main プログラムの前に置いた、という構成になっています。1 行目から 6 行目までを `myMax` という名前の**関数の定義プログラム**といい、その関数プログラムを呼び出すときは 10 行目のように書きます。

- 2. (**関数プログラムの書き方**) プログラム 2 の 1 行目から 6 行目までが関数プログラムの標準的な書き方です。1 行目を「`int main() {`」と置き替えれば、今まで書いてきたプログラムと変わらないことが分かるでしょう。1 行目を分解すると、それぞれ

```
double          関数値 (計算結果) は double 型である
myMax           関数の名前は myMax とする
double x, double y  x,y が独立変数で両方とも double 型である
```

という意味を持っています。

- 3. (**関数名**) 関数名は変数名と同じように、英文字で始まる文字列ですが、2 文字目以降は英数字あるいはアンダースコア (「`_`」、キーボード右下「ろ」のキーにある) でなければいけません。`myMax` のように 2 つ以上の単語を付けて 1 つの名前にする場合、2 つ目以降の単語を大文字で始めるか `my_max` のように単語と単語をアンダースコアでつなげると識別しやすくなります。
- 4. (**引数、仮引数、実引数**) 関数の独立変数のことを**引数** (ひきすう) と呼び、関数定義プログラムの中では特に**仮引数**、関数を呼ぶ側 (プログラム 2 では main プログラム) で書かれる独立変数は**実引数**と呼ばれます。この例では `x,y` が仮引数、`a,b` が実引数です。仮引数の宣言は、型が同じでも一つ一つの変数に必ずデータ型を指定する必要があります (「`double x, double y`」を「`double x,y`」と書くのは間違いです)。実引数の並びと仮引数の並びは対応しています。仮引数の個数分、実引数を用意しなければいけません。実引数の中身は仮引数に引き継がれて計算されますから、データの型は一致させておく必要があります。

- 5. (**return 文、戻り値、関数値**) 実際の最大値を計算する部分は、関数を使わない場合のプログラムと同じですから説明の必要はないでしょう。5 行目にある「**return**」は関数プログラムをそこで打ち切り、**return** の後に変数名や計算式が書いてあれば、その値（あるいは計算結果）を関数値とする、という命令語です。関数値は**戻り値**とも呼ばれます。
- 6. (**関数呼び出し、実引数**) 10 行目の `c=myMax(a,b)` の右辺は数学関数の `pow(a,n)` などと同じ形式です。`c=pow(2,3)` と書くと `c` に $8(=2^3)$ が代入されるのと同じように、これで、二つの `double` 型変数 `a,b` の大きい方の数が `c` に代入されます。`a,b` は関数定義 (1 行目) の `double x, double y` に対応しています。`a` が `x` に、`b` が `y` に代入され、計算結果が **return** 文によって関数値として返されるという仕組みです。実引数はこの例のような変数名でも、「`myMax(0,a+b)`」のように定数や、数式であっても構いません。
- 7. (**関数定義プログラムを置く場所**) 追加の実習では、10 行目の `myMax(a,b)` と書かれたところで「関数 `myMax` は定義されていません。`int` 型の値を返す外部関数と見なします」という警告 (warning) メッセージが表示されるはずです。これは、`myMax(a,b)` はユーザの定義する関数と解釈され、(ユーザがどこかで) 定義した `int` 型関数を呼び出したものと (VC2010 が好意的に) 解釈し、とりあえず作業を続けます。だから警告止まりです。作業を続けていくと、`main` プログラムの後に `myMax` という名前の `double` 型関数定義プログラムを発見して戸惑います。なぜならば、同じ名前の関数はすでに `int` 型関数として定義されたものと解釈されているからです。同じ名前の関数を定義することは出来ませんから「再定義されています」というエラーメッセージが表示されて終了します。関数定義プログラムは、最初に呼び出されるとき、定義済みでなければいけません。多くの関数が定義されるようになると、その前後関係をちゃんと管理するのは煩わしくなります。その場合は後述する**プロトタイプ**を利用します。

練習問題 9.1 (追加の実習の続き) 関数 `myMax` を `int` 型として作り直して (`main` プログラムの後において) ビルドしたときどのようなことが起きるか推測し、その推測が正しいかどうかを実際にプログラムを作って確かめなさい。

練習問題 9.2 試験の点数 (`int` 型)、平均点 (`double` 型)、標準偏差 (`double` 型) を与えて偏差値を計算する関数 `int hensachi(int, double, double)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。但し、偏差値は「(点数-平均値) / 標準偏差」を 10 倍したものを四捨五入し、それに 50 を足したものの、として計算されます。

9.2 配列の受け渡し

二つの数の最大値ではなく、 n 個のデータの最大値 $\max\{x_0, x_1, \dots, x_{n-1}\}$ を計算する関数を作るにはどうすればよいでしょうか。C で $x_0, x_1, \dots, x_{n-1}$ という不特定多数のデータを扱う場合は配列を使うということを学びましたが、その配列の内容を全部関数プログラムに渡すのは手間がかかりそうです。パソコンでよく使われる表計算ソフトの Excel には最大値を計算する関数がありますが、計算するときは `max(A3:A22)` のような書き方をします。この場合 `A3:A22` はデータではなく、データが記入されているセルの番号（場所）です。この考え方は使えそうです。

文法予備知識

1. 配列変数を独立変数にしたい場合は、数そのものではなく、数の記憶されている場所の情報を引き渡す。
2. 関数定義プログラムで、仮引数が配列変数であることを宣言するために「`a[]`」のように書く。
3. 関数定義プログラムで、引き渡された配列変数に値を代入した場合は、元の配列の中身が書き換えられるので、`return` 文を使う必要がない。

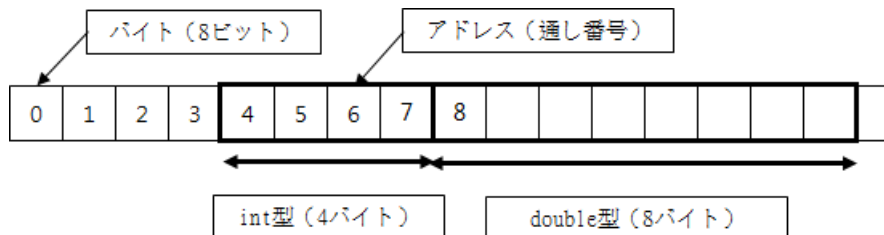
実習 45 次のプログラムは `int` 型配列データの最大値を計算するプログラムです。入力して実行しなさい。

プログラム例

```
// 配列データの最大値を計算する
1: int maxv(int x[], int m) {
2:     int i, z = x[0];
3:     for(i=1; i<m; i++) {
4:         if(x[i] > z) z = x[i];
5:     }
6:     return z;
7: }
// maxv 関数のテストプログラムの主要部
8: int c, n=6, a[6] = {4,2,1,5,6,3};
9: c = maxv(a, n);
10: printf("%d 個の最大値は %d\n", n, c);
```

実習 45 の解説 チェックボックスに ✓ を入れながら読みなさい。

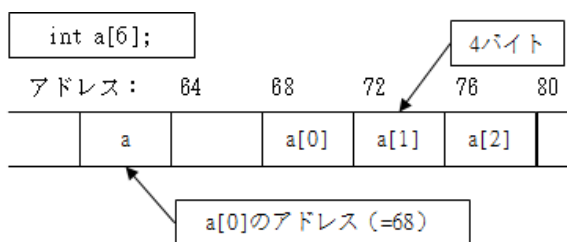
- 1. 関数を定義してデータを受け渡すとき、受け渡すデータの個数が少数ならばそれらを全部引数に並べて書くという実習 44 の方法で良いのですが、データが大量になるとそうもいきません。データの個数が不定の場合は対処しようがありません。そこで配列を使った受け渡しというアイディアが生まれました。そのアイディアを正しく理解するためにはコンピュータメモリのアドレスという概念を知る必要があります。
- 2. **(アドレス)** コンピュータのメモリ（記憶装置）は一本の長い映画フィルム（あるいはカメラロール）のようなものと考えてください。一つ一つのコマには先頭から**アドレス**（番地とも言います）と呼ばれる通し番号が付けられています。一つのコマは8等分され、一つ一つの区画は0か1の数を記憶することができます。区画のことを**ビット**、8ビットまとめたものを**バイト**と言います。したがって、1バイト（一コマ）には $256 (= 2^8)$ 通りの数（2進数で 00000000 から 11111111）を記録することができます。int 型数は連続した4バイトを使い（ワードということがあります）記録できる数の種類は 2^{32} 通りになります。double 型数は連続した8バイトを使って記憶されます（ダブルワード）。配列変数は連続したワード、あるいはダブルワードを使ってデータを記憶します（詳しい内容については101ページ参照）。



- 3. **(変数、配列とアドレス)** 変数が宣言されると、その変数に代入される値を記憶する4バイト（int 型変数の場合）ないし8バイト（double 型変数の場合）のメモリが割り当てられ、変数名とその先頭アドレスの対応表が作られます。そのアドレスをその変数のアドレスということがあります。プログラムでその変数が使われると、この「変数アドレス対応表」からその変数のアドレスが分かり、そのアドレスを持つメモリのデータと変数が同一視されます。

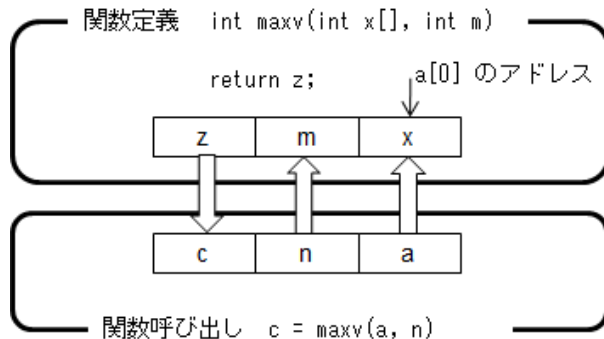
配列が宣言された場合は、配列の大きさ（ n とします）の4倍（int 型配列の場合）、あるいは8倍（double 型配列の場合）の連続したバイト数のメモリが確保され、0から $n-1$ までの添え字のついた変数が定義されますが、さらに、配列名という変数が定義され、そこには添え字0の変数の先頭アドレスが記憶されます。そして、変

数アドレス対応表には配列名とそのアドレスだけが登録され、添え字付き変数とそのアドレスは登録されません。なぜならば、添え字付き変数のアドレスは、添え字 0 の変数のアドレスと添え字から計算できるからです。例えばプログラム例のように「`int a[6];`」と宣言すると、ワード 6 つ分 24 バイトの領域が割り当てられ、先頭から順番に `a[0]`, `a[1]`, ... という名前が対応付けられますが、変数アドレス対応表には、配列名 `a` と `a[0]` の先頭アドレスのペアだけが登録されます。プログラムに、例えば「`a[1]`」という名前があったら、VC2010 は変数 `a` の中身（アドレス、図の例では「68」）を取り出し、それに 4 を足して「72」とし、72-75 番地のワードと同一視します。



- 4. **(アドレス渡し)** さてそこで、配列データを受け渡す方法です。データの個数が（この例では 6 ですが一般には）不定なので、9.1 節のようにデータのコピーを渡すことはできません。その代り、配列の添え字 0 の要素のアドレスを実引数とする、という方法を考えます。実習プログラムの 9 行目にある `maxv(a,n)` がそれです。配列の最初の要素のアドレスが分かれば、それを頼りにすべての添え字付き変数のアドレスが分かりデータにアクセスすることができるということを利用するのです。実引数 `a` に対応する関数定義プログラムの仮引数は 1 行目のように「`int x[]`」と書きますが、定義プログラムで定義されるのは変数 `x` だけで、そこには実引数の内容、すなわち配列 `a` の先頭アドレス、がコピーされます。仮引数 `x[]` のカギ括弧は**配列添え字演算子**と言います。大きさのない配列宣言のようなもので、定義プログラムの中で `x` は配列名のように振る舞い、添え字付き変数として使うことができます。したがって、関数定義プログラム内部で `x[0]` と書けば、それは `a[0]` と同じメモリに付けられた変数になります。関数定義プログラムでは配列の領域を確保していないことに注意してください。このようなデータの受け渡し法（参照法）を**アドレス渡し**と呼んでいます。変数の受け渡しを図にまとめたものを次ページに示します。

次は、関数定義プログラムで計算した配列データと呼び出した側が受け取る例です。



実習 46 次の dosukeisan 関数は n 回の繰り返しサイコロ振りの結果が入力された配列 x[] を受け取り、その度数分布 d[] を計算し、不正データを除いた正しいデータの個数を関数値として返す関数です。dosu を表示させるなど、main プログラムを完成させて実行しなさい。

プログラム例

```
// サイコロ数列の度数分布を作る
1: int dosukeisan(int x[], int d[], int n) {
2:     int i, m=0;
3:     for(i=1; i<=6; i++) d[i] = 0;
4:     for(i=0; i<n; i++)
5:         if(x[i] >= 1 && x[i] <= 6) m++, d[x[i]]++;
6:     return m;
7: }
// main プログラムの主要部分
8: int dice[1000], dosu[10], n, i, m;
9: for(i=0; i<n; i++) dice[i] = 6*rand() / (RAND_MAX+1) + 1;
10: m = dosukeisan(dice, dosu, n);
```

実習 46 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. 配列データを渡すために配列名というアドレスをコピーするという考え方は、データがどこで作られるかということには無関係です。実習 45 では関数定義プログラムで main プログラムのデータを使うためにアドレス渡しをしましたが、関数定義プログラムで計算した配列データを main プログラムで利用するためにアドレス渡しを利用することもできます。
- ☐ 2. 10 行目で関数 dosukeisan を呼ぶとき、2 番目の実引数である dosu は大きさ 10 の配列という実体はありますが、何のデータも入っていません。一方、関数定義プ

プログラムの `d` は実体はなく、実引数の `dosu` と同じ場所を参照しているだけです。したがって、5 行目の計算で配列 `d` に入力されるデータは、アドレス渡しの規定により、直接実引数配列 `dosu` に入力されます。したがって、10 行目を実行した後、配列 `dosu` は目的の度数分布が計算されていることになります。

- 3. (**カンマ演算子**) 5 行目の条件文の後半は、「`if(...)` {`m++`; `h[x[i]]++`;}」と書いたのと同じです。実行文を「`,`」でつなげて 1 行にまとめると、コードブロックと同じで、一塊の命令群とみなされると決められています。この「`,`」は**カンマ演算子**と呼ばれます。`for` 文の初期設定が複数の命令文になった時は「`,`」で区切るという約束がありましたが、これもカンマ演算子の例です。

練習問題 9.3 `double` 型配列 `x` に対して、`x[0]` から `x[n-1]` までの総和を計算する関数 `double sum(double[], int)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

練習問題 9.4 `int` 型配列 `x` に対して、`x[0]` から `x[n-1]` までの最大値から最小値を引いたもの（範囲という）を計算する関数 `int range(int[], int)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。但し、最小値を配列の n 番目、最大値を $n+1$ 番目に代入して返すようにし、結果は範囲、最小値、最大値の 3 つを表示させるようにしなさい。

練習問題 9.5 `int` 型配列 `x` に対して、`x[0]` から `x[n-1]` までの平均値を計算する関数 `double average(int[], int)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

練習問題 9.6 二つの配列データを要素毎に掛けてそれらの総和（つまり内積）を計算する関数 `double innerProduct(double[], double[], int)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

練習問題 9.7 買い物金額 (`int` 型)、支払い金額 (`int` 型) に対して釣り銭 (`int` 型) の金種と枚数を計算する関数 `int change(int, int, int[])` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

練習問題 9.8 配列データ `x` の順番を逆にした新たなベクトル `z` を作る関数 `void reverse(int x[], int z[], int)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

9.3 戻り値のない関数

関数というと、何か一つの値が計算されるように思われますが、そういう制約はありません。Cにおける関数は、数学関数のような関数値を計算するものもありますが、何かまとまった仕事（計算）をする単位として利用される方が多いくらいです。

文法予備知識

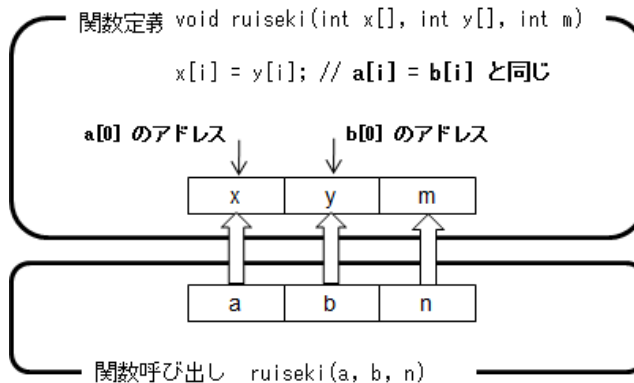
1. 関数値（戻り値）を計算しない「関数」もある（「サブルーチン（部分処理）」という呼び方もある）。
2. プログラムを読みやすくするために、まとまった処理手順はなるべく別のサブルーチンとして定義すると良い。

実習 47 次のプログラムを入力して実行しなさい。

プログラム例

```
// 配列データの累積を計算する
1: void ruiseki(int x[], int y[], int m) {
2:     int i;
3:     y[0] = x[0];
4:     for(i=1; i<m; i++) {
5:         y[i] = y[i-1] + x[i];
6:     }
7: }
8: void printArray(int z[], int n) {
9:     int i;
10:    for(i=0; i<n; i++) {
11:        printf(" %5d", z[i]);
12:        if((i+1)%5 == 0) printf("¥n");
13:    }
14:    if(n%5 != 0) printf("¥n");
15: }
// main 関数の中身
16: int k, n=9, a[] = {1,2,3,4,5,6,7,8,9}, b[9];
17: printArray(a, n);
18: ruiseki(a, b, n);
19: printArray(b, n);
```

追加の実習 1行目の最後の「{」を「;」に置き換えたものを新たに作って1行目の前に挿入し、1行目から7行目をmainプログラムの最後に移してビルドを実行しなさい。



実習 47 の解説 チェックボックスに✓を入れながら読みなさい。

- ☐ 1. printArray は int 型配列データを1行に5個ずつ形を整えて表示する「関数」です。今までと違うのは、
 - 関数値の型を書く場所に「void」と書いてある
 - return 文が無い
 - 関数呼び出しが代入文になっていない。
 という点です
- ☐ 2. (void 型) 関数 ruiseki には返すべき関数値がありません。その場合は関数名の型として「void」と書く決まりです。void は「空(くう)」という意味です。
- ☐ 3. (return 文の機能) return 文は、関数プログラムを打ち切り、返す関数値を指定する、という二つの機能がありましたが、printArray 関数はいずれの機能も使っていないので、return 文がありません。最後の「}」に行き当たれば、関数プログラムは自然に終了するのです。ruiseki 関数は返す関数値がベクトルで、アドレス渡しをしているのでこれまた return 文は不要です。逆に、関数定義プログラムの途中で return; と書いてある場合は、そこで関数プログラムの実行を終了します。ループ脱出の break 命令と同じようなものです。
- ☐ 4. 関数呼び出しは、17-19 行目のように関数名を書くだけです。関数値が定義されないで代入文の右辺に書かれても、代入すべき値がありません。代入文には書けないので、ただ、関数名を書くしかありません。
- ☐ 5. (関数 = 「ひとまとまりの仕事の単位」) 同じ void 型の関数でも、ruiseki 関数は計算したものをアドレス渡しで呼び出した側に返しているのです、関数値が複数ある

関数と考えることができますが、`printArray` 関数は入力データを加工して呼び出した側に返すという意図は全くありません。その意味で、数学で使われる関数とは異質なものですが、C 言語では「ひとまとまりの仕事の単位」を関数と言います。

- 6. (**プロトタイプ**) 実習 44 で、関数呼び出しの後に関数定義プログラムを置くとビルドエラーになることを体験しましたが、追加の実習では、1 行を挿入することでエラーとならずに済みました。関数定義プログラムの 1 行目だけ取り出して「{」の代わりに「;」で閉じて実行文にしたものを関数の**プロトタイプ**と言います。関数名の宣言文のようなものです。関数定義プログラムをたくさん使うようになると、どの関数が定義済みで、どの関数がまだ未定義かチェックするのが大変になりますので、このように余計な作業を避けるために、必ず最初にプロトタイプを書いておく、というのは良い習慣です。

関数の効用、階層化

関数定義プログラムを書く代わりに、17 行目と 19 行目の 2 箇所に 10~14 行目を、18 行目に 3 行目から 6 行目を（変数名を変えて）書き込んだ 1 つの `main` プログラムを作っても同じ結果が得られます。`for` 文が 3 つ入ったものと、この実習プログラムと比べ、どちらが見やすいかは明らかでしょう。関数はこのように、仕事の単位を明確にし、その流れをつかむのに大いに役立っています。もっと複雑なプログラムになった時、この違いは際立ってきます。

複雑な問題が与えられたら、いくつかの解けそうな部分問題に分割し、それらが解けたとすると、それらを組み合わせることで、元の問題を見つけることができるはず、というアプローチが効果的です。もし部分問題がそのまま解けそうになければ、それをさらに部分問題に分割し、それらが解けたとすると、...、というように問題の構造を階層化することで、考える範囲を狭め、問題解決に近づくことができます。

「問題を解く」ことは「アルゴリズムを書くこと」と同義のようなものですからアルゴリズムも**階層化**することで見通しがよくなります。小さな部分問題を解くアルゴリズムを関数と考えると、アルゴリズム全体を、それらの関数名だけを使ってプログラム化することができます。残された仕事は、個々の関数の定義プログラムを書くこと「だけ」です。8 章で説明した擬似コードを思い出してください。

関数名だけを使ったプログラムのデバッグは、部分的アルゴリズムの交通整理のようなものですから、デバッグそのものはそれほどむづかしいことはありません。実習のプログラムで言えば、`main` プログラムには「配列 `a` を表示」「配列 `a` の累積を配列 `b` に作る」「配列 `b` を表示」という三つの関数があるだけです。個々の関数定義プログラムのデバッグは、与えられた小さなアルゴリズムを正確に実現しているかどうかをチェックするだけ

なので、関数を使わない長いプログラムのデバッグに比べればはるかに容易な作業です。

こうすることで、実用的に重要な副産物も得られます。関数定義プログラムは関数呼び出しのルールを守ってさえいれば、中身がどのように書かれていようとも問題ありません。ということは、プログラム作成を多くの人が分担し、並行作業することで、工期を短縮できるということです。さらに、もし過去に作った関数定義プログラムを（数学関数ライブラリのように）まとめておけば、それをそのまま流用できないまでも、ゼロからスタートする必要はない、ということになります。

あるいは、関数定義プログラムを計算専用の代行業者と考え、関数を呼び出すことをその業者に外注する、と考えると分かりやすいかもしれません。仮引数は作業依頼書のフォーマット、実引数はその依頼書に書かれたデータ、「`return z`」は外注先が計算結果を書いた報告書を提出することと対応付けられます。アドレス渡しは、いわば自社のデータベースが載っている URL、あるいはファイル名を外注先に教えることに対応していて、外注先は直接そのデータベースを使って計算し、計算が終わったときは、計算終了しました、結果はデータベースに書き込んだので見てください、と言っているようなものです。報告書がいらない場合は `void` 型で、「`return z`」文はありません。

関数定義プログラムの骨格だけ取り出すと、例えば

```
void ruiseki(...) { ... }
```

となります。同じように `main` プログラムの骨格だけ取り出すと

```
int main( ) { ... }
```

となり、いずれも、

「関数値の型」＋「関数名」＋「仮引数の並び」＋「実行文」

の形をしていることが分かります。実際、`main` プログラムは `main` という関数名を持った関数定義プログラムだったのです。C プログラムは一つの `main` 関数と 0 個以上の関数定義プログラムを集めたもので、`main` 関数は実行時に最初に呼び出される関数、と決められています。`main` 関数は `int` 型ですから、本来であれば `return` 文が必要ですが、省略可能なので書いてありません。

`printf`, `scanf` も関数名で、それを実行文に書くことで、関数定義プログラムが実行され、画面表示やキーボード入力が行われますが、その定義プログラムは `#include <stdio.h>` を実行することによってビルドの際に `main` 関数に組み込まれることになっています。

練習問題 9.9 テストの点数 (`int` 型) に対して、60 点未満ならば「不合格です」、60 点以上ならば「合格です」、90 点以上の場合にはさらに「大変優秀です」と表示する関数 `void hyoka(int)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

9.4 データ共有、間接参照

9.2 節で関数に大量のデータを渡す方法を学びましたが、実際に渡しているのは配列変数の最初の要素のアドレスで、それ以外のデータは、配列要素が連続したメモリを使って記憶されているという事実を利用し、そのアドレスから計算によって記憶場所を特定し参照しているに過ぎないことが分かりました。そこで用いられたアドレス渡しは、間接参照というプログラミングの基本技法を利用したものです。

データはメモリに記憶され、そのメモリにつけられた名前（変数名）を書くことで利用することができました。このようなデータの利用の仕方は**直接参照**と呼ばれます。利用することを参照というのです。それに対して、メモリのアドレスを指定し、そのアドレスに記憶されているデータ、という参照の仕方を**間接参照**と言います。例えば、配列変数 `a[0], a[1], ...` が定義されている場合、`a[0]` に記憶されているデータは、変数名 `a[0]` によって直接参照され、（配列名 `a` は `a[0]` のアドレスを記憶しますから）配列名 `a` によって間接参照されています。

多くのプログラムが同一のデータを扱う場合、9.1 節のようにデータをコピーして利用するのは分かりやすく簡単な方法ですが、オリジナルが変更されるとそのたびに更新しなければならず、その状態を管理することは結構面倒です。そうではなく、アドレスをコピーしてそのアドレスからオリジナルのデータを間接参照する方法を使えば、データが変更されても常に最新のデータを利用することができるようになります。

配列名と添え字 0 の配列要素の関係を利用した 9.2 節のプログラムは間接参照のためのアドレス渡しの例でしたが、もう一つの例が `scanf` 関数にあります。

文法予備知識

1. 変数の前に「&」を付けたものは、変数のアドレスを表す。
2. アドレスを記憶する変数の前に「*」を付けると、その変数で間接参照されるデータを表す。
3. 関数の引数がアドレス渡しの場合、`return` 文なしで、その変更結果が呼び出したプログラムに受け継がれる。

実習 48 次のプログラムを入力して実行しなさい。

プログラム例

// アドレス演算子と配列

```
1:      int a, b[1];
2:      scanf("%d", &a);
3:      printf("a = %d, &a = %d¥n", a, &a);
4:      scanf("%d", b);
5:      printf("b[0] = %d, b = %d¥n", b[0], b);
```

実習 48 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (**アドレス演算子**) `scanf` 関数の 2 番目の実引数には必ず `&` を付ける必要がありました。`&` は**アドレス演算子**といい、変数の前にこれを付けたものは、その変数のアドレスを表すと決められています。2 行目で 2 と入力し、3 行目を実行すると、例えば「a = 2, &a = 11859756」のように表示されるでしょう。2 番目のアドレスの値は VC2010 が割り当てたもので、その都度異なるかもしれません。
- 2. (**scanf 関数の動き**) 2 行目のように書いて `scanf` 関数を呼び出すと、`scanf` 関数定義プログラムは、入力を必要とする変数 `a` のアドレスを受け取り、キーボードから入力したデータを、そのアドレスで間接参照した `a` に記憶させることができます。`scanf` 関数を終了して呼び出し元に返ってきたとき、必要とする値はすでに `a` に記憶されているので、9.1 節のような `return` 文によるデータの受け渡しは不要です。
- 3. (**配列名とアドレス演算子**) `scanf` 関数でアドレス演算子を使わないでも実行できることを示したのが 4 行目の実行文です。`scanf` 関数の 2 番目の引数はアドレスですから、配列名 `b` と書いても良いことになります。このとき、`scanf` 関数が間接参照するのは `b[0]` ですから、キーボードから入力された値は `b[0]` に記憶されていることが 5 行目で確かめることができます。

これらの実験から、アドレス演算子を使う方法と、大きさ 1 の配列を使う方法は全く同じということが分かります。ということは、配列を知っていれば、アドレス演算子を導入しなくても済みそうですが、一々大きさ 1 の配列を定義するのは煩わしいので、アドレス演算子を使うこのような方法は高い利用価値があります。

次の実習プログラムは、間接参照でデータを受け渡す関数定義プログラムの例です。

実習 49 次の `myMaxMin` は二つの数の最大値、最小値を同時に計算して返す関数です。`main` プログラムを完成させて実行し、結果を調べなさい。

プログラム例

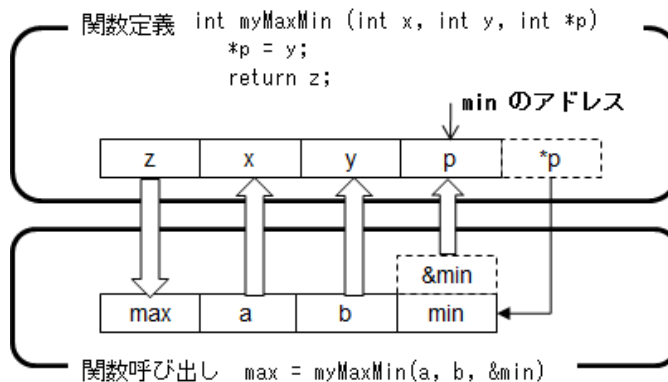
```
// & を使ったアドレス渡し
1: int myMaxMin(int x, int y, int *p) {
2:     int z;
```

```

3:      if(x > y) z = x, *p = y; // カンマ演算子
4:      else z = y, *p = x;
5:      return z;
6:  }
    // main プログラムの主要部
7:      int min, max, a, b;
8:      scanf("%d %d", &a, &b);
9:      max = myMaxMin(a, b, &min);
10:     printf("%d と %d の最大値は %d 最小値は %d\n", a, b, max, min);

```

追加の実習 4行目の後に p を "%d" で表示させる printf 文を挿入し、9行目の後に &min を "%d" で表示させる printf 文を挿入し、その結果を比較しなさい。



実習 49 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. 関数 myMaxMin で返す値は 2 つあるので、9.1 節の方法だけでは実現できません。そこで登場するのが、scanf で用いた「&」付き変数、すなわちアドレス渡しの方法です。8 行目で scanf を実行すると、変数 a, b の内容が書き換えられているのと同様に、9 行目を実行すると、変数 min の中身が最小値に書き換えられていることが分かります。
- ☐ 2. (間接演算子) アドレスを実引数とした場合、仮引数は 1 行目のように、変数の前に「*」を書くと決められています。この例では、変数 p に変数 min のアドレス (のコピー) が記憶されます (追加の実習で確認しなさい)。*は間接演算子、あるいは間接参照演算子と言い、変数の前に添えると、その変数 (アドレスが記憶されている) によって間接参照されるデータを表す、と約束されています。この場合は p によって間接参照されているのは min なので、*p は変数 min と書いても同じということができます。仮引数の記法は、間接参照されるデータ *p が int 型である、と

いうことを表したものと解釈することができます。

上の関数定義プログラム中 `*p` と書けばそれは `min` と同じことであり、`*p` を代入文の左辺に書けば、`min` に直接代入したのと同じ結果になります。したがって、関数呼び出しが終了した時、変数 `min` は関数によって書き換えられた内容が記憶されているので、`return` 文を使うことなく関数の計算内容が呼び出し元に返されたことになります。

- 3. (**ポインタ**) 上のプログラムの `p` や配列名のように、アドレスを記憶する変数のことをポインタあるいはアドレス変数と呼びます。間接参照するデータが `int` 型ならば `int` 型ポインタ、`double` 型ならば `double` 型ポインタと言います。ポインタはデータを間接参照するためのものですが、間接参照できるためには、アドレスに加えてデータの占有範囲が分かっている必要があります。したがって、ポインタを定義する場合は、必ずデータ型を添える必要があります。
- 4. 追加の実習から分かるように、`p` に記憶されるデータは、VC2010 によって割り振られたアドレスの値ですから、あなたの書いたプログラムとは何の関係もありません。したがって、`p` という変数そのものを数式の中に書いても無意味なことが分かります。ポインタの前に間接演算子があって初めて変数として使える、と考えてください。但し、「`*`」なしのポインタを数式に書いても文法エラーにはならず実行され、おかしい結果が得られることになるでしょう。その間違いをチェックするのはプログラムを書く人自身です。

練習問題 9.10 二つの `int` 型数 `x,y` を入れ替える関数 `void swap(int *x, int *y)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

ヒント：126 ページを参照。

練習問題 9.11 秒数 (`int` 型) `n` に対して、`hh` 時 `mm` 分 `ss` 秒表示に置き換える関数 `void henkan(int n, int *hh, int *mm, int *ss)` を作り、テストプログラムを書いて、正しく動くことを確認しなさい。

練習問題 9.12 3 個の `double` 型数 `a,b,c` を入力すると、2 次方程式 $ax^2 + bx + c = 0$ の解を計算する関数 `int qequation(double a, double b, double c, double *x, double *y)` を作りなさい。但し、 $a = 0$ の時は関数値を 0 として終了、それ以外の時、重解ならば関数値を 1 として `x` に解を代入、2 実解があるならば関数値を 2 として `x,y` にその解を代入、複素数解ならば関数値を 3 として `x` に実数部、`y` に虚数部を代入、することとします。テストプログラムを書いて、正しく動くことを確認しなさい。

9.5 参考 ポインタ

ポインタはアドレスを記憶する変数で、これを使ってデータを間接参照することにより、複数の関数の間でデータ共有することができます。このテキストのレベルでは必ずしもポインタの詳しい知識は必要としませんが、間接参照というプログラミングの基本概念を実現するための必須の考え方なので、少し解説します。

このテキストでは、ポインタは関数定義プログラムにおける仮引数として導入されましたが、ポインタをそれ独自で定義することができます。p を int 型ポインタとして定義するには「`int *p;`」と書きます。これは関数定義プログラムでアドレスを実引数とした場合の仮引数の書き方と同じで、間接参照した*p が int 型であると宣言することで、変数 p がポインタであることを間接的に定義しています。同様に、変数 q が double 型ポインタであることを定義するために「`double *q`」と書きます。^{*1}

ポインタはあるデータが記憶されているアドレスであり、データをどこに記憶させるかは VC2010 が決めることで、ユーザは関与できませんから、ポインタの値はユーザが勝手に決めることはできません。最初に変数 a を定義し、アドレス演算子を用いて、「`p = &a;`」とすることにより、システムによって変数 a に割り当てられたアドレスが p に代入されます。このとき、p は a をポイントする、あるいは p は a のポインタである、という言い方をします。

文法予備知識

1. アドレスを記憶する変数をポインタという。
2. ポインタは間接参照したデータのデータ型を宣言することで間接的に定義される。
3. 配列要素 `a[i]` が定義されているとき、`a` は `a[0]` のポインタ、`a+1` は `a[1]` のポインタになる。

実習 50 次のプログラムはポインタの宣言、代入文と、その結果を確かめるものです。入力して実行し、その結果を表示させ、解説を読んで理解しなさい（各行の//以降は入力不要）。

プログラム例

^{*1} 「`int* p`」と書くこともできます。「`int*`」を「int 型ポインタ」というデータ型と考えると、この方がすっきりしていますが、2つのポインタを同時に定義しようとして「`int* p, q;`」と書くと「`int *p; int q;`」の意味に解釈されてしまいますので、推奨されません。

// 変数のアドレス表示、ポインタ

```
1:      int a=8;
2:      int *p; // ポインタ p の宣言
3:      p = &a; // p に変数 a の記憶場所（アドレス）を代入する
4:      printf("a = %d, &a = %d, *(&a) = %d¥n", a, &a, *(&a));
5:      printf("*p = %d, p = %d, &(*p) = %d¥n", *p, p, &(*p));
6:      *p = -1;
7:      printf("a = %d, *p = %d¥n", a, *p);
8:      p[0] = -2;
9:      printf("a = %d, p[0] = %d¥n", a, p[0]);
```

実習 50 の解説 チェックボックスに ✓ を入れながら読みなさい。

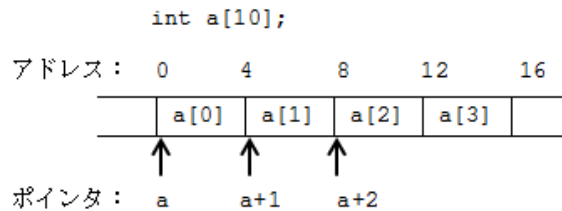
- 1. 2 行目でポインタ p を定義していますが、ポインタを機能させるためには必ず 3 行目のような代入文が必要です。これにより変数 a はポインタ p によって間接参照され、変数 a と、変数 p に間接演算子を付けた *p が同一視されます。4,5 行目はそれを確認したものです。4,5 行目の最後の表示はアドレス演算子と間接演算子との関係を確認したものです。p=&a ならば *p=a なので、p=&(*p), a=*p=*(&a) が成り立ち、両者は逆の関係があることが分かります。
- 2. 6 行目はポインタの間接参照を代入文の左辺に使った例です。実際に代入される変数は a であることが 7 行目の結果から確認できます。
- 3. 8 行目で定義されていない配列変数が使われていますが、これは実習 48 にあるように、ポインタが大きさ 1 の配列変数を定義するようなものだ、ということと関係しています。p[0] と書けば、配列名 p は p[0] のポインタですが、もともと p は a のポインタだったので、p[0] と a は同一視され、これも変数 a への代入文になるのです。

9.5.1 ポインタと配列

配列名（変数）もアドレスを記憶するのでポインタですが、このポインタには、配列を定義すると同時にそのアドレスが代入されて固有の値になり、ほかの値に変更することはできません。例えば、「int a[10];」のように配列を定義すると、その配列名 a は int 型ポインタとなり、添え字が 0 の変数 a[0] のアドレスが記憶され (a = &a[0];)、a に間接演算子を付けた *a は a[0] と同じになります。

0 以外の添え字付き変数のアドレスは a[0] のアドレスに int 型データのバイト数である 4 の倍数を掛けたものになっているので、a から計算することができます。a の値に 1 や 2 など 4 の倍数以外の数を足したアドレスはワードの先頭ではないので間接参照でき

ず、ポインタにはなりません。そこで、 $a+1$ は普通の足し算とは異なり、 a のアドレスに `int` 型データ一つ分のバイト数を足したアドレスを表す記号と約束します。したがって、 $a+1, a+2, \dots$ が $a[1], a[2], \dots$ のポインタになります。これにより、 $*(a+1)$ と $a[1]$ は同一視されます。もし a が `double` 型の配列名（ポインタ）ならば、 $a+1$ は a に（`double` 型データ一つ分のバイト数である）8 を足した数になります。



練習問題 9.13 「`int a[]={3,1,4,1,5,9};`」と定義されたとき、 $a+1$, $a+2$, $a+5$ と a の差がいくつになるか予想し、実際にその値を表示するプログラムを実行して確かめなさい。また、 $*a$, $*(a+1)$, $*(a+2)$, $*(a+5)$ がいくつになるか調べなさい。また、配列を `double` 型で定義した場合に、同じ問題を解きなさい。

この仕組みが分かれば、配列の一部を実引数として関数を呼び出す場合の実引数の書き方が理解できるようになります。

実習 51 次のプログラムは、添え字 1 から n までに記憶されている n 個のデータの最大値を求めるのに、実習 45 で作った関数 `maxv` を使った例です。入力して実行しなさい。

プログラム例

```
// 配列とポインタ
1: int maxv(int *x, int n); // プロトタイプ
// main プログラムの主要部
2: int a[100], n, i;
3: scanf("%d", &n);
4: for(i=1; i<=n; i++) scanf("%d", &a[i]);
5: printf("max = %d\n", maxv(a+1,n));
```

実習 51 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. 1 行目は、関数 `maxv` のプロトタイプです。実習 45 で、配列データを受け渡す場合は、実引数に配列名を書き、仮引数は「`int x[]`」と書く、と説明しましたが、実際に受け渡しされるのは、添え字 0 の変数のアドレスだということが分かりまし

た。ポインタという言葉を知っていれば、これは「引数を添え字 0 の変数のポインタとする」と言えば十分です。1 つ目の仮引数が「`int *x`」となっているのはそれを表したものです。もちろん「`int x[]`」と書いても間違いではありません。関数定義プログラムの中でも、添え字付き変数とポインタによる間接参照は同一視されますから、`x[i]` の代わりに `*(x+i)` と書くこともできます。

- 2. 5 行目の `a+1` は `a[1]` のポインタです。これにより、`maxv` の最初の仮引数は `a[1]` のアドレスを記憶しますから、`*(x+i)` (あるいは `x[i]`) は `a[i]` ではなく `a[i+1]` と同一視されます。その結果、`x[0], ..., x[n-1]` の最大値を計算する関数定義プログラムを呼ぶと、`a[1], ..., a[n]` の最大値を関数値として返ってきます。もちろん、`a+1` ではなく `&a[1]` と書いても同じ結果が得られます。

練習問題 9.14 `a[0], ..., a[n-1]` を `int` 型データとしたとき、次のプログラムを実行して表示されるものは何ですか？

```
int n=6, a[] = {1,2,3,4,3,2};
int *p = a;
for(i=1; i<n; i++) if(a[i] > *p) *p = a[i];
printf("*p = %d, p = %d¥n", *p, p);
```

練習問題 9.15 バブルソートの関数プログラムを、配列変数を使わずにポインタだけを使って書き直しなさい。

9.6 2次元配列データの受け渡し

データ集計や、数学の行列計算などで、下付添え字が2個付くような変数（行列要素）を使った計算が良く出てきます。Cでそのようなデータを扱う場合は、2次元の配列を使用します。2次元配列と言っても、コンピュータの内部にExcelのシートのような構造があるわけではなく、アドレスをソフト的に変換して、バーチャルに2次元配列を作り出しているだけ、ということを5.2.2節で学びました。2次元配列（行列）を引数として渡す場合、添え字の組からその添え字付き変数のアドレスが計算できるためには、先頭アドレスに加えて、2行目の始まる場所を指定する必要があります。

文法予備知識

1. 2次元データを引数とする場合は、2行目の開始位置を陽に指定しなければいけない。
2. そのために、2番目の添え字の動く範囲を指定する方法と、各行の開始位置をポインタ配列で指定する方法がある。
3. プログラム全体で共通に使用される変数を、グローバル変数として定義できる。

実習 52 次は n 人の学生の m 回分の試験成績が一覧表になっているものを読み込み、各学生の平均点を計算するプログラムです。(1) 入力して実行しなさい。データは適当に作りなさい。(2) 8行目「`c[][MDIM]`」を「`c[][]`」、あるいは「`c[][10]`」として実行しなさい。

プログラム例

```
// 関数ヘデータの受け渡し (2次元配列の場合)
1: #define MDIM 20
2: double average(int w[], int n) {
3:     int i;
4:     double sum=0;
5:     for(i=1; i<=n; i++) sum += w[i];
6:     return sum / n;
7: }
8: void scoring (int n, int m, int c[][MDIM], double d[]) {
9:     int j;
10:    for(j=1; j<=n; j++) d[j] = average(c[j], m);
```

```
11: }  
    // main プログラムで scoring 関数を呼び出す  
12:     double A[200][MDIM], B[200];  
13:     int n, m;  
14:     scoring (n, m, A, B);
```

実習 52 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **(マクロ定義)**「#define 文字列 1 文字列 2」は、**マクロ定義**と言い、プログラムに含まれる「文字列 1」をすべて「文字列 2」に置き換えるという命令文です。「文字列 1」は**マクロ記号定数**と呼ばれます。1 行目により「MDIM」はすべて「20」に置き換えられます。同じでなければならぬ定数や文字列（命令文でもよい）を複数個所で書かなければいけないとき、マクロ記号定数を利用すると、デバッグ作業が軽減されると同時にプログラムも見やすくなるというメリットがあります。乱数を扱ったときに出てきた `RAND_MAX` もマクロ記号定数です。
- 2. **(2次元配列の受け渡し)** 2次元配列は、コンピュータメモリでは1次元配列として記憶されるという説明をしました。したがって、2次元配列を複数のプログラムで共有するためには、先頭アドレスに加えて、2行目がどこから始まるか（1行目の要素数はいくつか）ということが分かっている必要があります。8行目で2次元配列を受け取る仮引数の書き方を示しています。2次元配列なので、`[ ]`を2つ付けることに加えて、2番目の添え字の動く範囲を指定します。これは呼び出し元で定義された配列の列数に合わせなければいけないので、マクロ記号定数を使っています。先頭アドレスは実引数によって指定することにし、実引数は1次元配列の場合同様、配列名を書くことになっています。実習 (2) の結果から、その指定がない、あるいは間違っている場合、正しい結果が得られないことが分かります。
- 3. **(2次元配列の行ベクトル配列名)** 2次元配列 `A` を定義すると、(この例の) 配列要素は順番に `A[0][0]`, `A[0][1]`, ..., `A[0][19]`, `A[1][0]`, ..., `A[1][19]`, `A[2][0]`, ... のように名前が付けられます。最初の 20 個は `A[0]` を配列名と思えば、大きさ 20 の 1次元配列と考えることができます。実際 2次元配列の 2番目の添え字を取ったもの `A[i]` は `A[i][0]` のアドレスを表すポインタ、という決まりがあります。したがって、10行目にある `average` 関数呼び出しの実引数として `c[j]` とあるのは、`j` 番目の学生の成績ベクトル（行ベクトル）の先頭アドレスを指定したものです。`average` 関数はそれを受けて `j` 番目の学生の平均点を計算して返してきます。

関数を呼び出す際は実引数と仮引数の間で情報をやりとりすると約束されているのに、仮引数として（実引数にはない）配列の 2 番目の添字の上限値が設定されているのはいか

にも不自然です。また、`scoring` 関数を別々のプログラムから呼び出す場合、どのプログラムでも同じ大きさの配列を宣言しておかなければならず、メモリの無駄遣いが発生する恐れもあります。

2次元の配列は1次元配列をつなげたものなので、そのすべての先頭アドレスを配列として実引数とすればよいのではないか、というアイディアがあります。

実習 53 次は実習 52 をポインタを使って書き直したものです。マクロ定義は使わず（1行目は削除）、`average` 関数（2行目から7行目）は共通です。実行してその結果が実習 52 と同じになることを確かめなさい。

プログラム例

```
// ポインタ配列を使って 2 次元配列を受け渡す
8: void scoring (int n, int m, int *pp[], double d[]) {
9:     int j;
10:    for(j=1; j<=n; j++) d[j] = average(pp[j], m);
11: }
// main プログラムで scoring 関数を呼び出す
12:    double A[200][20], B[200];
13:    int n, m;
14:    int *p[200]
15:    for(i=0; i<200; i++) p[i] = A[i];
16:    scoring (n, m, p, B);
```

実習 53 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. **（ポインタ配列）** 配列の要素がポインタであるような配列をポインタ配列といい、14 行目のように定義します。2 次元配列 `A[i][j]` の 2 番目の添え字を取ったもの `A[i]` は 2 次元配列変数の `i` 行目の先頭アドレス（ポインタ）を表すということを思い出すと、15 行目はポインタ配列に 2 次元配列の各行のポインタを記憶させていることになります。
- ☐ 2. 仮引数の「`*pp[]`」は、`pp` が配列名で、配列要素 `pp[i]` に間接演算子を付けたものが `int` 型という意味ですから、`pp[i]` はポインタということになります。したがって、10 行目は実習 52 と同じように、2 次元配列 `A` の `j` 行目の先頭アドレスを実引数としたことになります。
- ☐ 3. **（ポインタのポインタ）** 実引数が配列名の時、仮引数で「`pp[]`」と書いても「`*pp`」と書いても同じということでしたから、8 行目の `*pp[]` は `**pp` と書くことができます。また、配列変数「`p[j]`」はポインタと間接演算子を使って「`*(p+j)`」と書いても同じ、ということから、10 行目の `pp[j]` は `*(pp+j)` と書いても同じです（こ

れらのことを、実際に実行して確かめなさい)。間接演算子がついていますが、そこで間接参照されるのはアドレスで、データはそのアドレスを使って間接参照する(間接参照の間接参照)ということから「**p」は**ポインタのポインタ**とも呼ばれます。

実習 54 実習 52 のプログラムで 1 行目、8 行目、10 行目、12 行目、14 行目を次のように替えて実行しなさい。

プログラム例

```
1: double A[200][20];
8: void scoring (int n, int m, double d[]) {
10:     for(j=1; j<=n; j++) d[j] = average(A[j], m);
12:     double B[200];
14:     scoring (n, m, B);
```

実習 54 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. (**グローバル変数、ローカル変数**) これまで、1 行目のような変数の宣言文は、関数定義プログラムの一部として書かれていましたが、このように、関数定義とは無関係にいきなり宣言文を書いて変数を定義することが許されます。このようにして定義される変数は**グローバル変数**と言い、どの関数からも同じ名前を使って参照することができるので、一つの(オリジナル)データが常に共有されています。それに対して、今までのように関数定義プログラムの中で定義される変数は**ローカル変数**ということがあります。

上のプログラム例では、学生の成績表をグローバル変数で定義してデータを共有しているので、関数間のデータのやり取りの際に成績表の配列アドレスを受け渡す必要がなくなります。したがって、8 行目の仮引数リストには 2 次元配列がなく、14 行目の実引数にもありません。10 行目の共有データを使う箇所では定義されたグローバル変数名がそのまま使われています。

- 2. (**変数定義の有効範囲**) ある関数で定義された変数(つまり、ローカル変数)はその関数の内部だけで有効なので、別の関数で同じ名前がローカル変数として定義されていてもその内容はお互いに何の影響もありません。ある関数で、グローバル変数と同じ名前の変数をローカル変数として定義した場合は、その関数ではローカル変数が優先され、グローバル変数としては使えなくなります*2。グローバル変数と同じものをローカル変数として定義し、グローバル変数のつもりで参照するという勘違いはよくある発見しにくいバグです。混乱を避けるために、グローバル変数と

*2 定義された変数の有効範囲を、その変数のスコープということがあります。

して定義されている変数名はローカル変数として定義しない方が賢明です。

- 3. (**グローバル変数濫用の戒め**) グローバル変数を使えば、直接参照でデータを共有することができます。それならば、変数は全部グローバル変数として定義しておけば、アドレス渡し、間接参照という面倒な手段は必要ないのではないか、という疑問が生じるでしょう。簡単なプログラムならばその通りです。しかし、複雑な仕事をするプログラムを書くために関数が多く定義され、たくさんの人が関わるようになると、同じ名前を別の目的で使うことにより意図しない書き換えが生じるなどの不都合が多くなります。混乱を避けるために、グローバル変数はなるべく使わないというのが C のプログラミングスタイルです。

練習問題 9.16 実習 52 のプログラムに、各回の試験の全学生の平均点を $n+1$ 行目 ($A[n+1][1], A[n+1][2], \dots$) に計算するプログラムを付け加えなさい。

練習問題 9.17 大きさ $m \times n$ の行列 A の指定した二つの行 x, y を入れ替える、という作業をする関数 `void irekae(int A[][10], int m, int n, int x, int y)` を作り、テストプログラムを書いて、それが正しく動くことを確かめなさい。

練習問題 9.18 大きさ $m \times n$ の二つの行列 A, B の和を新たな行列 C に計算する関数 `void madd(int A[][10], int B[][10], int C[][10], int m, int n)` を作り、テストプログラムを書いて、それが正しく動くことを確かめなさい (練習問題 5.13 の関数プログラム版)。

練習問題 9.19 $n \times m$ 行列 A と $m \times h$ 行列 B の積を計算する関数プログラムを書きなさい。配列変数はグローバル変数で定義しなさい (問題 5.7 の関数プログラム版)。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F評価の項目について猛勉強しなさい。

- ☐ ☐ 関数の定義、プロトタイプ
- ☐ ☐ 仮引数、実引数、データの受け渡し
- ☐ ☐ `return` 文
- ☐ ☐ 関数の呼び出し
- ☐ ☐ 配列データの受け渡し
- ☐ ☐ データの記憶場所とアドレス（番地）
- ☐ ☐ ポインタ、アドレス変数
- ☐ ☐ アドレス渡し
- ☐ ☐ `void` 型関数
- ☐ ☐ 直接参照と間接参照
- ☐ ☐ アドレス演算子「&」と間接（参照）演算子「*」
- ☐ ☐ 2次元配列データの受け渡し
- ☐ ☐ グローバル変数、ローカル変数
- ☐ ☐ マクロ定義

9.7 章末演習問題

問題 9.1 正の整数 n の自分自身以外の約数の中で最大のものを計算する関数 `MaxFactor(int n)` を書きなさい。

問題 9.2 Excel の `RANDBETWEEN(m,n)` は二つの整数 $m < n$ に対して、 m 以上 n 以下の整数乱数を生成する関数です。それと同じ機能を持つ C の関数プログラム `int randBetween(int,int)` を書きなさい。

問題 9.3 `int` 型配列を昇順にソートする関数 `void sortupI(int[], int)` を書きなさい。

問題 9.4 1 から 52 までの数をランダムに 4 つの組に分けて、組ごとに整列して表示するプログラムを書きなさい。

ヒント：規則数列をランダムに並べ替えるプログラムは 8 章末演習問題にあります。組ごとに整列するには、上の問題で作った `sortupI` 関数を利用し、実習 51 のようにすればよいでしょう。1 から 52 までの数を n とすると、「 $(n-1)\%13+1$ 」は 1,2,...,13(A,2,3,...,9,10,J,Q,K とする)、 $\lceil (n-1)/13 \rceil$ は 0,1,2,3(S,H,D,C とする) となるので、トランプを切って 4 人に配ったときの手札を表示するプログラムと考えられるので、これでゲームが作れる？

問題 9.5 n 次正方行列 a のべき乗を行列 c に計算する関数プログラム `mpow(double a[][10], double c[][10], int n, int m)` を書きなさい（問題 5.8 の関数プログラム版）。

問題 9.6 実習 43 のプログラム（マージ）をポインタを使って書き直し、テストデータを使ってチェックしなさい。

問題 9.7 今までに作った複雑そうなプログラムをいくつか取り出し、「小さな仕事単位」に分けてそれぞれを関数化したものと、それらと呼ぶ `main` プログラムという構成に作り変えてもらなさい。

第 10 章

算法 5：再帰計算

数列の漸化式は、 $a_n = a_{n-1} + a_{n-2}$ のように、 n 項と $n-1$ 以下の項との関係を表したもので、それにより、与えられた初期値を使って一般項を順番に生成することができました。漸化式のように数式に表せない問題でも、規模の大きな問題を同じ種類の小さい規模の問題と関係付けることにより、問題を解く方法が有効です。このような考え方を一般に**再帰**と言います。この章では、再帰的方法を実現するプログラムの書き方を実習した後、次のような問題たちを取り上げて、解説します。

- ユークリッドの互除法
- クイックソート
- ハノイの塔
- エイトクィーン問題

10.1 再帰的関数定義

漸化式を C プログラムで計算することを考えます。例えば、1 から n までの和を $f(n)$ とすると、 $f(n)$ は 1 から $n-1$ までの和に n を足したものですから、漸化式 $f(n) = f(n-1) + n$ が得られます。これを計算するアルゴリズムは次のようになります。

アルゴリズム 25 $f(n)$ (1 から n までの和、 n は 1 以上の整数) を計算する

1. $n = 1$ ならば 1 が答え、アルゴリズム終了
2. さもないければ、 $f(n-1) + n$ が答え、アルゴリズム終了

ステップ 2 で出てくる $f(n-1)$ がいくつになるか分かりませんが、このアルゴリズムの n には特別な意味はありませんから、 n を $n-1$ に置き換えれば、同じアルゴリズムを使って $f(n-1)$ を計算することができます。その答えにはまた $f(n-2)$ という未知の値

が出てくる、というように、 $f(1)$ にたどり着くまでこのサイクルが続きます。アルゴリズムの中でそのアルゴリズムを必要としている（呼び出している）という意味から、このようなアルゴリズムは**再帰的アルゴリズム**とといい、アルゴリズムの中からそのアルゴリズムを呼び出すことを**再帰呼び出し**と言います。

今までのアルゴリズムと違い、アルゴリズム 25 が終了しても、その解がいくつになるのか、これを読んだだけでは分かりません。しかし、確かにこのアルゴリズムは有限ステップで答えを見つけて終了することが分かります。実際、 $f(3)$ を計算する場合、

ステップ 2 で $f(2)$ が必要になり、

それを計算するアルゴリズムのステップ 2 で $f(1)$ が必要になり、

それを計算するアルゴリズムのステップ 1 で、 $f(1) = 1$ が分かり、

それを使って $f(2) = f(1) + 2 = 3$ が分かり

それを使って $f(3) = f(2) + 3 = 6$ が分かる

という動きの後にアルゴリズムが終了します。再帰呼び出しのアルゴリズムが停止するためには、必ずステップ 1 のような記述が必要です。これを再帰アルゴリズムの**停止条件**と言います。

実習 55 次はアルゴリズム 25 を使って n までの総和を計算する関数プログラムです。

(1) テストする main プログラムを付け加えて実行し、 $f(4)$ を計算しなさい。(2) 4 行目の先頭に「//」を挿入して実行し、どうなるか調べなさい。

```

1:  int f(int n) {
2:      int sum;
3:      printf("f(%d) の呼び出し \n", n);
4:      if(n == 1) return 1;    // ステップ 1
5:      sum = f(n-1) + n;      // ステップ 2
6:      printf("f(%d) の計算終了 \n", n);
7:      return sum;
8:  }
```

実習 55 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. (**再帰呼び出し、再帰的プログラム**) プログラム例 3 行目のように、関数定義プログラムの中で、自分自身を呼び出していますが、このようなやり方を**再帰呼び出し**と言い、再帰呼び出しを含むプログラムを**再帰的プログラム**と言います。挿入した printf 文により、 $f(4)$ の計算が終了する前に $f(3)$ が呼び出されていることが分かります。
- ☐ 2. (**停止条件**) 実習 (2) を実行して分かるように、4 行目の停止条件がないと、 $f(3)$ を呼び出した後 $f(2)$ が呼び出され、 $f(1)$, $f(0)$, $f(-1)$, ... が次々と再帰的に呼び出され続けるので実行停止しません。このような状態を「プログラムは暴走している」

と言います。上のプログラムでも、この場合は n は 1 以上という暗黙の了解でアルゴリズムを作っていますので、 $f(0)$ を計算しようとするすると停止条件に引っかからないので暴走することをお確かめください。

- 3. **(再帰呼び出しとメモリ)** 再帰呼び出しは、その考え方に慣れると便利で、プログラムも簡単になりますが、良いことだけではありません。関数プログラムの途中で別の関数を呼び出した場合、元の関数プログラムは後で戻ってきたときに備えてそのときの状態を保存しなければいけません。その保存のために使われる仕組みを**スタック**と言います。カフェテリアにあるトレイを積み重ねたものを想像してください。一番上のトレイを取り出すと下のトレイがポップアップし、トレイを上を追加すると、それまであったトレイが下に押し下げられます。この例では、 $f(4)$ の計算途中で $f(3)$ を呼び出すとき、そのプログラムの途中経過をスタックに記録します。 $f(3)$ は $f(2)$ を呼び出すとき、その途中経過をスタックに記録します。 $f(4)$ の途中経過はその下に追いやられます。 $f(2)$ が終了すると、 $f(3)$ の途中経過がポップアップされて $f(3)$ を再開します。このようにして、再帰呼び出しされるたびに、途中経過がスタックに記憶されます。実習 (2) はそのスタック内容を表示させたものです。
- 4. **(メモリ爆発)** したがって、あまり再帰呼び出しを続けると、スタックが成長し、もし 2 行目のような条件文に引っかからないと、スタックは増える一方です。関数プログラムがもっと複雑で変数の数も多い場合、その途中経過をすべてスタックするとなると、使用するメモリが多くなって大変です。ある程度きちんとした見積もりをしてから再帰呼び出しを使う必要があります。最悪の場合、スタックに収まりきらなくなると、プログラムは止まってしまいます。実習 (3) は、その状況を強制的に体験するものでした。停止条件を除いてしまったので、再帰呼び出しを際限なく繰り返し、その結果スタックがあふれ実行が中断され、ダイアログボックスが表示されました。そのエラーメッセージの中に「stack overflow」という文字列を発見するでしょう。スタックが溢れて (overflow) 停止しました、ということです。この状態を**メモリ爆発**、といったりします。
- 5. この状態で止まった場合は、実行画面の閉じるボタンが機能していません。「中断」ボタンをクリックし、Shift を押しながら F5 キーをクリックしなさい。実行画面が消えて VC2010 の画面に戻ると、プログラムを入力する場所に、入力したはずのない文字列が表示されるかもしれませんが、気にしないでそのタブを閉じてください。

練習問題 10.1 正の整数 n の階乗 $n!$ は 1 から n までの整数の積と定義されています。これは、再帰的に $n! = n \times (n-1)!$ と定義されます。これを利用して $n!$ を計算する再帰的プログラム `double factorial(int)` を書きなさい。 $n!$ はすぐに大きくなってしまふの

で、関数の戻り値を `double` 型にしてあります。結果を表示させるときは「%30.01f」というフォーマット指定子を使いなさい。

練習問題 10.2 2 項係数 $\binom{n}{k}$ は次の式で定義されています。

$$\binom{n}{k} \equiv \frac{n(n-1)\cdots(n-k+1)}{k!} = \frac{n!}{k!(n-k)!}$$

これは $\binom{n}{k} = \binom{n}{k-1} \times \frac{n-k+1}{k}$ という漸化式を満たします。停止条件は $\binom{n}{0} = 1$ です。この漸化式を使って 2 項係数を計算する関数 `int binomialC (int n, int k)` を書き、それをチェックするプログラムを書きなさい。 $\binom{n}{k} = \frac{n}{k} \times \binom{n-1}{k-1}$ として計算することもできます。そのプログラムも書き、両者の結果を比較しなさい。扱う数はすべて `int` 型なので、`int` 型だけで計算しようとする、プログラムの書き方次第でおかしな結果になることがあります。なぜでしょう？ これは数学の問題です。アルゴリズムを書くときは数学の知識も必要になるのです。

練習問題 10.3 $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ という関係を使って、2 項係数 $\binom{n}{k}$ を計算する関数プログラムとそれをチェックするプログラムを書きなさい。

ヒント：再帰呼び出しを 2 回必要とします。

10.2 ユークリッドの互除法、再訪

4.5 節で出てきた、二つの正整数の最大公約数を計算する問題を再び取り上げます。124 と 34 の最大公約数がいくつになるか即答できないとしても、「それは $22(= 124 - 34 * 3)$ と 36 の最大公約数と同じ」と答えても間違いではありません。これは再帰の思考法です。このように、問題をそれと等価な小さな問題に置き換えて解く、という考え方はとても有効です。

二つの数の最大公約数を求めるユークリッドの互除法のアルゴリズムは 4.1 節で学びましたが、これを再帰的方法で書き換えてみましょう。2 数 a, b が与えられたとき、 a を b で割ったあまりを c とすると、 $c \neq 0$ ならば、 b と c の最大公約数は a と b の公約数と同じ、というのがユークリッド互除法の根拠でした。 b と c の最大公約数は無理に求めなくても、 a と b の公約数を求めるアルゴリズムがあればそれを利用すればよいでしょう。というわけで、次の再帰的アルゴリズムが書けます。

アルゴリズム 26 (ユークリッドの互除法、再帰的表現) 二つの正整数 a, b の最大公約数を計算する。最大公約数を $g.c.d.(a, b)$ と書く

1. $a \div b$ のあまりを c とする。
2. もし $c = 0$ ならば、 b が答え、アルゴリズム終了。
3. (さもなければ) $g.c.d.(b, c)$ が答え、アルゴリズム終了。

a と b の最大公約数が分からないから教えてください、と言っているのに「それは b と c の最大公約数です」と言われると、はぐらかされた感じがしますが、言っていることは間違いではありません。 $g.c.d.(b, c)$ を知りたければ、もう一度このアルゴリズムを適用すればよいのです。もし、 $a \geq b$ ならば、 (b, c) は (ベクトルとして) (a, b) よりも確実に小さくなっていますから、このような操作を繰り返すうちにそのうち必ずステップ 2 で終了するはずですが、実際、このアルゴリズムは必ず有限回で終了します ($a < b$ の場合はどうなりますか、 c がいくつになるか計算しなさい)。

とはいえ、その計算過程は一度は実際に数値例を追いかけて見ないと分からないでしょう。そのために、変数の一覧表を作って経過を追ったのが次の表です ($a = 124, b = 34$ の場合)。

再帰回数	a	b	c	最大公約数
1	124	34	22	$g.c.d.(34, 22)$
2	34	22	12	$g.c.d.(22, 12)$
3	22	12	10	$g.c.d.(12, 10)$
4	12	10	2	$g.c.d.(10, 2)$
5	10	2	0	2

人間ならば繰り返し 3 回目くらいで「答えは 2」と分かるでしょうが、コンピュータは暗算が出来ないので、とにかく $c = 0$ まで突き進みます。でもそんな労力は気に掛けるコンピュータではありませんから任せておきなさい。

実習 56 再帰呼び出しを使ったアルゴリズム 26 を使って最大公約数を求める関数 `int rgcd(int, int)` を書き、変数の動きを確認する上のような変数表を表示する命令文を付け加えなさい。その関数プログラムをチェックする `main` プログラムを書いて結果を確認しなさい。また、再帰を使わないで作った 4.5 節のプログラムの結果と比較しなさい。

ヒント：アルゴリズムの各ステップから自然に C の命令文が思い浮かぶはずです。

練習問題 10.4 配列 $a[0], a[1], \dots, a[n-1]$ に入った n 個の整数の最大公約数を計算する関数を書き、チェックプログラムを作って正しく動くことを確認しなさい。関数の名前と引数の並びは `int gcds(int a[], int n)` としなさい。

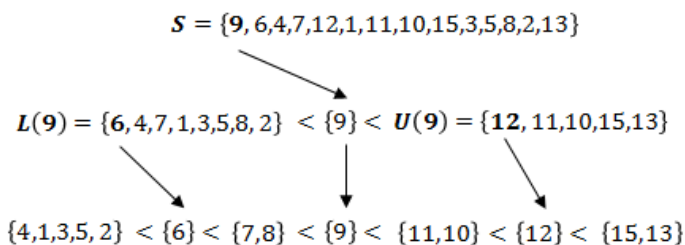
10.3 クイックソート

並べ替え（ソート）のアルゴリズムで、一つ一つを並べ替えをしたつもりはないのに、いつの間にか整列してしまったという、摩訶不思議なアルゴリズムがあります。この方法について説明しましょう。

基本的な考え方（方針）は、データの集合を、ある数より大きいものと小さいものの二つの組に分ける、ということを繰り返せば、いつかは全体が大きさの順に並ぶだろう、ということです。

例えば、 $\{9, 6, 4, 7, 12, 1, 11, 10, 15, 3, 5, 8, 2, 13\}$ という数列を昇順に並べるとしましょう。

1. 先頭の 9 と 2 番目以降の数を順番に比較していくと、9 より小さい組 $\{6, 4, 7, 1, 3, 5, 8, 2\}$ と、9 より大きい (9 以上) の組 $\{12, 14, 11, 10, 15, 13\}$ に分けられます。
2. 続いて、 $\{6, 4, 7, 1, 3, 5, 8, 2\}$ について、同じように、先頭の 6 と 2 番目以降の数を比較していくと、6 より小さいもの $\{4, 1, 3, 5, 2\}$ と、6 より大きいもの $\{7, 8\}$ に分けられます。
3. $\{12, 14, 11, 10, 15, 13\}$ についても同じように、先頭の 12 を基準に小さいものと大きいものに分けると $\{11, 10\}$ と $\{14, 15, 13\}$ になるでしょう。
4. ここまでの作業の結果、 $\{4, 1, 3, 5, 2\}\{6\}\{7, 8\}\{9\}\{11, 10\}\{12\}\{14, 15, 13\}$ というように、集合として大小関係のあるいくつかの部分集合に分けることができました。ここまで来れば、あとはもう一息。



この考え方を記号を使って整理しておきましょう。小さいもの順に並べ替えたい数の集合を S として、 S の任意の要素（ここでは先頭の数としています）を a とし、 S を a より小さいものと大きいものに分けるという作業を $Q(S, a)$ と名付けましょう。このとき、 a より小さいものの集合を $L(a)$ 、大きいものの集合を $U(a)$ とします。上の例では、

$$a = 9, L(9) = \{6, 4, 7, 1, 3, 5, 8, 2\}, U(9) = \{12, 14, 11, 10, 15, 13\}$$

です。 S と a が与えられた時、 $L(a)$ と $U(a)$ を生成する、というのが $Q(S, a)$ の作業内容です。

上の数値例を新たな記号と対応させておきましょう。

1. $S = \{9, 6, 4, 7, 12, 1, 11, 10, 15, 3, 5, 8, 2, 13\}$ に対して $a = 9$ とすると $L(9) = \{6, 4, 7, 1, 3, 5, 8, 2\}$, $U(9) = \{12, 14, 11, 10, 15, 13\}$ 。
2. $S = L(9) = \{6, 4, 7, 1, 3, 5, 8, 2\}$ に対して、 $a = 6$ とすると、 $L(6) = \{4, 1, 3, 5, 2\}$, $U(6) = \{7, 8\}$ 。
3. $S = U(9) = \{12, 14, 11, 10, 15, 13\}$ に対して、 $a = 12$ とすると、 $L(12) = \{11, 10\}$, $U(12) = \{14, 15, 13\}$ 。

このように、作業 $Q(S, a)$ を実行してできる新たな集合 $L(a), U(a)$ の要素の個数が 3 以上ある場合は、それらを新たな S と思って $Q(S, a)$ を実行するというのを繰り返すことにより、要素数が 2 以下の集合に分解することができます。ルールが分かれば、後は機械的に適用するだけです。この続きを書くと、

4. $S = \{4, 1, 3, 5, 2\}$ に対して、 $a = 4$ とすると、 $L(4) = \{1, 3, 2\}$, $U(4) = \{5\}$ 。
5. $S = \{1, 3, 2\}$ に対して、 $a = 1$ とすると、 $L(1) = \phi$, $U(1) = \{3, 2\}$ 。
6. $S = \{14, 15, 13\}$ に対して、 $a = 14$ とすると、 $L(14) = \{13\}$, $U(14) = \{15\}$ 。

これらの結果から得られる小集合たちを大きさの順位並べると、

$$\{1\}\{3, 2\}\{4\}\{5, 6\}\{7, 8\}, \{9\}\{11, 10\}\{12\}\{13\}\{14\}\{15\}$$

という順番に並ぶでしょう。要素が二つしかない集合は直接要素同士を比較すればよいので、結局 $Q(S, a)$ という作業を 6 回繰り返すことによって、最初に与えられた S がほぼ昇順に並んだことになります。

やっていることはきわめて単純ですが、すべての手順を丁寧に書こうとすると結構大変です。しかし、アルゴリズムとして考えると、手順としては数の集合をある条件を満たすように二つの部分集合（と一つの数）に分解する、ということしかやっていません。したがって、アルゴリズム、そしてそれを実現するプログラムは驚くほど簡単です。分解された集合に対して、同じ手順を繰り返し適用するわけですから、これは再帰そのものです。まとめておきましょう。

アルゴリズム 27 (クイックソート) $a_1, a_2, \dots, a_n$ を小さいものから順に並べ替える。 $quickSort(a_1, a_2, \dots, a_n)$ は $a_1, a_2, \dots, a_n$ を昇順に並べる作業を表す記号と同時に、その結果の数列も表すものとします

1. n が 1 以下ならばおしまい。
2. $n = 2$ ならば、二つを比較して小さいものが先になるように並べる、アルゴリズム

終了。

3. $a_2, \dots, a_n$ の中で a_1 以下のものを $b_1, b_2, \dots, b_m$ 、 a_1 より大きいものを $c_1, c_2, \dots, c_k$ とする ($m + k = n - 1$)。
4. $b_1, b_2, \dots, b_m$ を $\text{quickSort}(b_1, b_2, \dots, b_m)$ で小さいものから順に並べ替える。
5. $c_1, c_2, \dots, c_k$ を $\text{quickSort}(c_1, c_2, \dots, c_k)$ で小さいものから順に並べ替える。
6. $\{\text{quickSort}(b_1, b_2, \dots, b_m), a_1, \text{quickSort}(c_1, c_2, \dots, c_k)\}$ が答え。

これでおしまい。集合として小さい数の組が大きい数の前に来るように並べる、というだけで、ステップ 2 を除けば前後の数の大小を直接比較しているわけではありませんが、再帰のおかげで大きさの順に並べかえられるのですね。上で調べた数値例はこの手順に則っていることが確認できるでしょう。

C のプログラムにする場合は、 $b_1, b_2, \dots, b_m$ 、 $c_1, c_2, \dots, c_k$ をそれぞれ一時的に別の配列に記憶させるという方法が頭に浮かびます。やってみましょう。

実習 57 長さ n の int 型配列 $a[1], a[2], \dots, a[n]$ が与えられたとき、 $a[2], \dots, a[n]$ の中で $a[1]$ 以下のものを配列 $b[1], b[2], \dots, b[m]$ に、 $a[1]$ より大きいものを配列 $c[1], c[2], \dots, c[k]$ に記憶させるプログラムを書きなさい。それを利用して、アルゴリズム 27 を実現する再帰関数 $\text{quicksort}(\text{int } a[], \text{int } n)$ を書き、正しく動くことが確認できるプログラムを書きなさい。

ヒント：プログラムの主要部：

```
if(a[i] > a[1]) { m++; b[m] = a[i]; }
else { k++; c[k] = a[i]; }
```

ソートのアルゴリズムがどれくらい速いかということを比較する場合に、二つの数の大小を比較する回数が何回必要か、という尺度を使います。クイックソートの場合は、最初の数の並び方によって比較の回数が変わってきてしまいますから、乱数を使ってデータを生成し、並べ替え終わるまでの比較回数を数える、という実験を何回か繰り返し、その平均値を使って比較します。次の表は、並べ替えるデータの個数 N を大きくしたとき、単純な選択法アルゴリズムに対してどれくらい少なくて済むかを計算したものです。

N	100	1000	10000	100000
選択法	4950	499500	5×10^7	5×10^9
クイックソート	648	10987	1.56×10^5	2×10^6

この表から、選択法はデータ数が 10 倍になると比較は 100 倍になっているのに対して、クイックソートは 10 数倍と抑えられていることが分かります。

練習問題 10.5 クイックソートと選択法ソートの計算速度を比較するプログラムを書き、上の表を確かめなさい。テストデータは $\text{rand}()$ を使って生成しなさい。

実際の問題でソートする場合、要素数が小さいうちは実習プログラムでも対処出来ますが、要素数が大きくなるとそのうち動かなくなります。先に説明したように、メモリ爆発が起きるからです。ではどうするか。上のアルゴリズムは説明を分かりやすくするために、「大きい組」と「小さい組」用に別々の配列を使いましたが、全体の要素の個数が増えるわけではありません。b 配列と c 配列ができたなら、a 配列は不要になります。したがって、a 配列の前半に b 配列、後半に c 配列が来るように、a 配列の中身の入れ替えだけで処理可能なはずで。実際、次のような手順により、実現可能です。最初の S を例にとって説明しましょう。

1. $\{9, 6, 4, 7, 12, 1, 11, 10, 15, 3, 5, 8, 2, 13\}$ に対して、先頭の 9 より小さいものを一つずつ前に送り、9 よりも大きいものを見つけたら、次は後ろから順に前に調べて、9 より小さいものが見つけたらそこで停止する、という処理により、 $\{6, 4, 7, 7, \mathbf{12}, 1, 11, 10, 15, 3, 5, 8, \mathbf{2}, 13\}$ となる（最初 7 のあった場所は 7 を前に移しただけですので 7 のままです）。
2. 停止したところの 2 を 12 の前に、12 を 2 のあったところに移す： $\{6, 4, 7, \mathbf{2}, 12, 1, 11, 10, 15, 3, 5, 8, \mathbf{12}, 13\}$ （5 番目の 12 も上と同じ理由で 12 のままです）。
3. 12 のあった次から、9 より小さいものを一つずつ前に送る、12 の一つ前から順に前に調べて、9 より小さいものが見つかったらそこで停止する、という処理により、 $\{6, 4, 7, 2, 1, 1, \mathbf{11}, 10, 15, 3, 5, 8, 12, 13\}$ となる。
4. 停止したところの 8 を 11 の前に、11 を 8 のあったところに移す： $\{6, 4, 7, 2, 1, \mathbf{8}, 11, 10, 15, 3, 5, \mathbf{11}, 12, 13\}$ 。

というような操作を続けると、最終的に $\{6, 4, 7, 2, 1, 8, 5, 3, 15, \mathbf{15}, 10, 11, 12, 13\}$ となるので、3 の後の 15 を 9 と入れ替えると、 $\{6, 4, 7, 2, 1, 8, 5, 3, 9, \mathbf{15}, 10, 11, 12, 13\}$ となり、9 より小さいものが 9 より前に、9 より大きいものが 9 より後に並ぶことになります。ステップ 2,4 のように、小さいものと大きいものを交換する場所をきちんとトレースできていることがこの処理の基本です。

練習問題 10.6 上の説明を理解し、予備の配列を使わず、データの配列だけを使ってクイックソートを実現する C のプログラムを書きなさい。

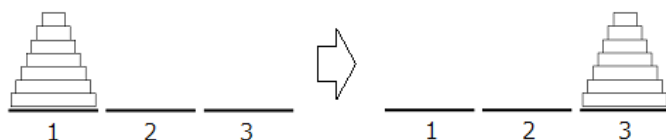
ヒント $\{15, 10, 11, 12, 13\}$ を並べ替える際の実引数については、実習 51 を参考にしなさい。

10.4 ハノイの塔

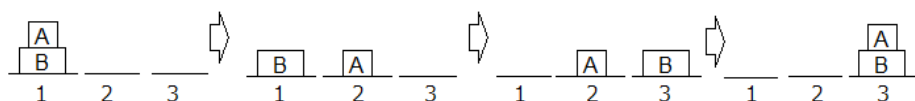
「ハノイの塔」は再帰的アルゴリズムの代表例として有名なパズルの問題です。

問題

三つの「台」と大きさの違う 7 枚の「円盤」があります。円盤は三つの台のどれかに積み重ねておかなければいけません。同じ台にある円盤は必ず大きいものが小さいものの下になるように重ねるものとします。今、7 枚の円盤はすべて台 1 の上に置かれています（下図左）。ある台の一番上にある円盤 1 枚を別の台（の円盤の上）に移す、という操作だけを繰り返すことによって、円盤をすべて台 3 へ移動する（下図右）最小の手順を求めてください。



円盤が二つ (A, B) だけならば、次のようにすれば移動できます。

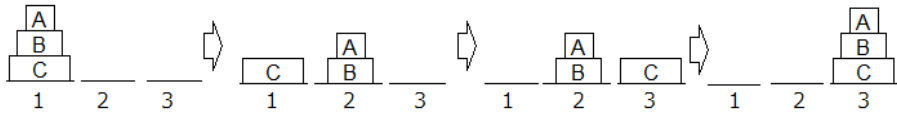


言葉で書くと、「円盤 A を台 1 から台 2 へ移し、円盤 B を台 1 から台 3 へ移し、円盤 A を台 2 から台 3 へ移す」となりますが、移す円盤は台を指定すると自動的に決まるので、「台 1 の一番上にある円盤を台 2 に移動する」という動作を表す記号を「 $1 \rightarrow 2$ 」とすると、この解は、「 $1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 3$ 」と書けば十分です。この動作をまとめて $H_2(1, 3)$ と表すことにします：

$$H_2(1, 3) : 1 \rightarrow 2, 1 \rightarrow 3, 2 \rightarrow 3$$

円盤が三つ (A, B, C) の場合は、上二つの円盤 A, B を「糊付け」して一つの円盤だと思えば、円盤が二つの場合と同じようにして、次のようにして移動できるでしょう。

円盤 A, B を台 2 に移すときに円盤 C は無視してよいので、最初の円盤二つの場合と同じようにすればよい、但し、移す台の番号が違うだけです。これは $H_2(1, 2)$ と書けます。同じように、台 2 の円盤 A, B を台 3 へ移す場合も $H_2(2, 3)$ と書くことができます。結局、全体として、 $H_2(1, 2), 1 \rightarrow 3, H_2(2, 3)$ とすることで作業完成！ そこで、3 個の円盤を



台 1 から台 3 へ移動するという動作を表す記号を $H_3(1, 3)$ と書いて、その解を次のように表すことができます：

$$H_3(1, 3) : H_2(1, 2), 1 \rightarrow 3, H_2(2, 3)$$

$H_3(1, 3)$ の問題を、円盤二つの小さい問題の解 $H_2(1, 2)$ を使って解く、という再帰の考え方を使った解法です。

円盤が四つ (A, B, C, D) の場合は、... 聡明な読者はもうお分かりでしょう。円盤 A, B, C を糊付けして台 2 へ移し ($H_3(1, 2)$)、 D を 3 へ移し ($1 \rightarrow 3$)、台 2 の 3 枚円盤を台 3 へ移す ($H_3(2, 3)$)、として完成です。すなわち、

$$H_4(1, 3) : H_3(1, 2), 1 \rightarrow 3, H_3(2, 3) \quad (10.1)$$

がその答えです。

結局、最初に与えられた問題の答えは次で与えられます：

$$H_7(1, 3) : H_6(1, 2), 1 \rightarrow 3, H_6(2, 3) \quad (10.2)$$

おしまい。円盤 7 個の問題を円盤 6 個の二つの問題に分割して、解が求まったら「 $1 \rightarrow 3$ 」を間に挟んで二つの解をつなげるだけ、という再帰の構造ができあがります。円盤 6 個の問題は二通り作る必要があるように見えますが、2 つの台を i, j とすると、3 番目の台の番号は $6 - i - j$ と表すことができるので、 m 個の円盤を台 i から台 j に移す問題を解くプログラムが一つあれば十分です。というわけで、次のようなアルゴリズムができます。

アルゴリズム 28 m 枚の円盤を台 i から台 j へ移動する際の円盤の動き求める。それを $hanoi(m, i, j)$ と書く。

1. もし $m = 1$ ならば、 $hanoi(m, i, j) = \{(i \rightarrow j)\}$ としてアルゴリズム終了。
2. $hanoi(m - 1, i, 6 - i - j)$ の次に $(i \rightarrow j)$ 、それに続けて $hanoi(m - 1, 6 - i - j, j)$ を並べたものが答え、アルゴリズム終了。

このように、大きさ n の問題をそれより小さな（大きさ $n - 1$ の）問題に分解して考えるということは、再帰的アルゴリズムの特徴です。慣れるまでは時間がかかるでしょうが、分かっしまえば、これほど便利な、あるいは怠惰な方法はありません。

練習問題 10.7 上のアルゴリズムを使うと、2 枚の円盤を台 i から台 j へ移す場合の答えは、 $\{(i \rightarrow 6 - i - j), (i \rightarrow j), (6 - i - j \rightarrow j)\}$ と書けることを確かめなさい。

実習 58 (1) 4 枚の円盤を使ったハノイの塔パズルの解を（コンピュータを使わずに）書きなさい。(2) m 枚の円盤を台 i から台 j に移すというハノイの塔のパズルを解く関数プログラム `void hanoi(int m,int i,int j)` とそれをチェックするプログラムを書き、実行しなさい。

ヒント：といってもアルゴリズムがそのまま C のプログラムになっているようなものです。

ステップ 1 は「`if(n==1) printf("%d -> %d, ",i,j);`」

ステップ 2 は「`hanoi(...); printf(...); hanoi(...);`」とすれば完成。

練習問題 10.8 ハノイの塔のパズルで、一つの円盤を移すのに 1 秒かかるとして、完成するまでに要する最短時間を計算しなさい。一般に、円盤が n 枚あるハノイの塔のパズルを完成するのに要する最短時間を $f(n)$ としたとき、 $f(n)$ を計算する漸化式を作り、 $n = 64$ の場合の所要時間を予想した後に実際に計算しなさい。漸化式を解いて、 $f(n)$ の陽解を求め、計算結果を検算しなさい。

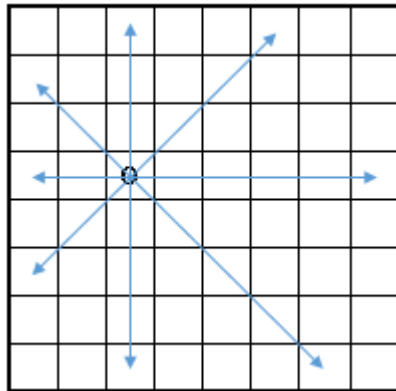
ヒント：再帰的アルゴリズムの考え方を使って $f(n)$ と $f(n-1)$ を結び付けなさい。 $f(1) = 1, f(2) = 3, f(3) = 7\dots$ 、 $f(n) + 1$ はいくつ？ このパズルが考案されたとき、 $n = 64$ のパズルの解を実際に実行し、64 枚の円盤が移し替えられたとき世界は崩壊する、という逸話（宣伝文句）が添えられていたらしい。

10.5 エイトクィーン問題

チェスというゲームのボードを使ったパズルです。

問題 オセロゲームのような正方格子からなるボードを用意します。マス数はチェスに倣って縦、横 8 マスずつとします。そのマスの中に 8 個のコマを置くのですが、マスを縦方向に見ても、横方向に見ても、斜め $\pm 45^\circ$ 方向に見ても、コマが一つしかない、というように配置しなさい。

チェスのクィーン（オセロで使う「コマ」という言葉を当てはめることにしましょう）は、ちょうど、将棋の飛車と角の動きを合わせたような動きをします。この問題は 8 個のコマを、お互いの利き筋（図の矢印上のマス目のこと）を避けてうまく配置できるか、という問題と言い換えることができるので、**エイトクィーン問題**（8 人の女王問題？）と呼ばれています。



この問題を解くには、ただひたすら、8 個のコマの配置が条件を満たすかどうかチェックするしかありません。いかにきちんとすべてのケースをリストアップできるかということが、この種の問題を解くための鍵です。コマの配置を動かすために、各列を順番に調べていく、というように考えると、8 重の for 文、という構造が頭に浮かびます。各列とも 8 マスありますから、コマの置き方は $8^8 = 16777216$ 通りあって、それをすべてチェックすれば解が見つかります、が大変です。しかし、**バックトラック**という再帰の考え方を使うと、すっきりと整理することが出来ます。

コマの配置は 2 次元のベクトルで表現することが出来ますが、各列ごとに考えるとコマは一つしか置けない（縦の利き筋制約）ので、必要な情報は各列のどの行にコマを置くかという 8 つの数だけです。そこで、マス (j, k) にコマを置く（ k 列目には j 行目のマスにコマを置く）ということを $Q_k = j$ によって表すことにします。 $Q_1, Q_2, \dots, Q_8$ に 1 から

8 までの数字を重複無く割り当て（行の利き筋制約）、それらが斜め 45 度の利き筋制約を満たせば、それが問題の答えです。利き筋の制約を記号で表すと、

- $\{Q_1, Q_2, \dots, Q_8\}$ は $\{1, 2, \dots, 8\}$ を並べ替えたものでなければいけない（行の利き筋制約）
- $Q_k + k (k = 1, 2, \dots, 8)$ はすべて異なる（ $y = -x + c$ 方向の利き筋制約）
- $Q_k - k (k = 1, 2, \dots, 8)$ はすべて異なる（ $y = x + c$ 方向の利き筋制約）

となります。

問題を解く場合は、八つのコマをいっぺんに動かそうとしないで、順番に一つずつ固定して考える方が問題が小さくなって考えやすいでしょう。1 列目はどこに置いても構いません。2 列目は 1 列目の数の ± 1 以内でなければどこにでも置けます。3 列目は、... だんだん場合の数が大きくなって複雑になります。こういう場合は、1, 2, 3, ... という具体的な数を使って解を求めるよりは、いきなり k 列目にコマを置く問題を考えた方が楽です。具体例を考えるより抽象的に考えた方がわかりやすい！

最初から順番に各列に一つずつ置いて $k - 1$ 列まで置けたとしましょう（ $Q_1, Q_2, \dots, Q_{k-1}$ が決まる）。 $k - 1$ 個のコマのすべての利き筋が空いている状態です。次にやるべき仕事は k 列目の何行目にコマが置けるか、調べることです。これを問題 $P(k)$ としましょう。

まず $k \leq 8$ の場合、問題 $P(k)$ では、

1. すでに決まっている $Q_1, Q_2, \dots, Q_{k-1}$ に抵触しないように k 列の何行目にコマが置けるか調べ、
2. それが見つければその行番号を Q_k として、次の問題 $P(k + 1)$ に進む（問題を先送り）。
3. どの行にもコマが置けない場合は、最初の $k - 1$ 列の配置： $Q_1, Q_2, \dots, Q_{k-1}$ の解は存在しないので、 $P(k - 1)$ を解くところからやり直し（ $Q_1, Q_2, \dots, Q_{k-1}$ を使った問題 $P(k)$ は解く必要がない）。

となります。コマが置けるかどうかは、すでにおいたコマの利き筋になっていないかどうかを確かめれば良いでしょう。 j 行目にコマを置けるための条件は、すべての $i = 1, \dots, k - 1$ に対して、 $Q_i = j$ でも $Q_i \pm i = j \pm k$ （複号同順）でもない、ということです。

うまくいくとステップ 2 によって問題 $P(9)$ に到達します。問題 $P(9)$ は「 $Q_1, Q_2, \dots, Q_8$ が決まっているという条件で 9 列目にコマを置く問題」ですが、そんな問題は解く必要がありません、「 $Q_1, Q_2, \dots, Q_8$ が決まっている」のですから。つまり $P(9)$ を目指して頑張ればよいのです。そこで、アルゴリズムは次のようになるでしょう。

アルゴリズム 29 $P(k)$ ：エイトクィーン問題を解くためのパーツ

1. $k = 9$ ならば $(Q_1, Q_2, \dots, Q_8)$ を表示しておしまい。
2. $j = 1$ とする。
3. (j, k) にコマが置けるならば（利き筋チェック）、 $Q_k = j$ として $P(k+1)$ を解く。
4. j に 1 を加え、 $j \leq 8$ ならばステップ 3 へ戻る。
5. さもないければ、それ以上の解がないのでアルゴリズム終了。

問題をこのように整理しておく、エイトクィーンの問題を解くということは、 $P(1)$ から始めて、最後に $P(9)$ まで到達することに他なりません。 $P(1)$ から順に解いてみることにしましょう。同じことなので、説明を簡略化するために 6×6 の盤で考えましょう。実際に描いて、コマを置きながら読みなさい。

$P(1)$ のステップ 3 で $Q_1 = 1$ 、 $P(2)$ を呼ぶ

$P(2)$ のステップ 3 で $Q_2 = 3$ 、 $P(3)$ を呼ぶ

$P(3)$ のステップ 3 で $Q_3 = 5$ 、 $P(4)$ を呼ぶ

$P(4)$ のステップ 3 で $Q_4 = 2$ 、 $P(5)$ を呼ぶ

$P(5)$ のステップ 3 で $Q_5 = 4$ 、 $P(6)$ を呼ぶ

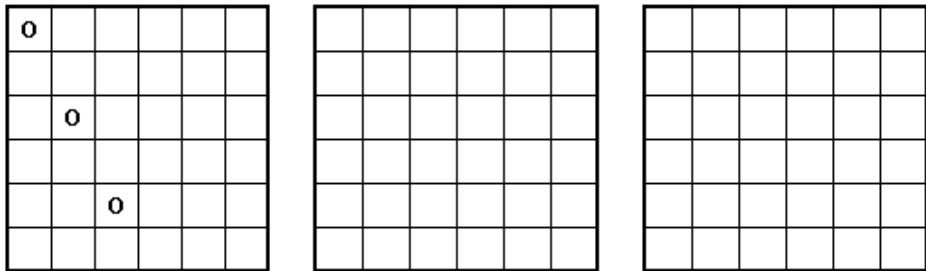
$P(6)$ のステップ 5 を実行して $P(6)$ を終了（どこにも置けない）

$P(5)$ のステップ 5 を実行して $P(5)$ を終了

$P(4)$ のステップ 5 を実行して $P(4)$ を終了

$P(3)$ のステップ 3 で $Q_3 = 6$ 、 $P(4)$ を呼ぶ

...



ここまでの一連の作業をまとめて、 $P(k)$ を呼び出された順に並べると、 $P(1) \rightarrow P(2) \rightarrow P(3) \rightarrow P(4) \rightarrow P(5) \rightarrow P(6) \rightarrow P(5) \rightarrow P(4) \rightarrow P(3) \rightarrow P(4) \rightarrow \dots$ となりました。ステップ 3 を実行するときに k が 1 増え、ステップ 5 を実行するときに k が 1 減る、という「行きつ戻りつ」の動きを繰り返すため、この考え方は**バックトラック法**（後戻り法？）と呼ばれ、この種の問題を解くための標準的な手法になっています。

実習 59 アルゴリズム 29 を使って、 $P(1)$ を計算すると、ステップ 3 で $Q_1 = 1$ となり、次に $P(2)$ に進む。ついで、 $Q_2 = 3$ として $P(3)$ に進む。以下、順に進行して、最初にそれ以上ステップ 3 を実行できず、ステップ 5 を実行するときの k の値は 7 であることを確かめ、そのときの $Q_1, Q_2, \dots, Q_6$ を求めなさい。その経験を元に、エイトクィーン問題を解くプログラムを書きなさい。

例えば $Q_1 = 1$ とした場合の解は、4 通り、 $(Q_2, \dots, Q_8)$ は 5863724, 6837425, 7468253, 7582463 となります。チェックに使ってください。

練習問題 10.9 クィーンが 8 個とは限らず、一般の n としたとき、いわば n クィーン問題を解くプログラムを書き、 $n = 4, 5, 6, 7, 9$ の場合の解を求めなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ 再帰的アルゴリズムと停止条件
- ☐ ☐ 再帰的関数定義、再帰呼び出し
- ☐ ☐ メモリ爆発
- ☐ ☐ 再帰をつかったユークリッド互除法
- ☐ ☐ クイックソート
- ☐ ☐ ハノイの塔パズル
- ☐ ☐ エイトクィーンパズル

参考 実習プログラムの例（主要部分のみ）

実習 57 (クイックソート) のプログラム例

```
void quicksort(int a[], int n) {
    int b[100], c[100], ib=0, ic=0, k;
    // 停止条件
    if(n <= 1) return;
    if(n == 2) {
        if(a[1] <= a[2]) return;
        k = a[1], a[1] = a[2], a[2] = k;
        return;
    }
    // a[1] を基準に、「下」組と「上」組に分類
    for(k=2; k<=n; k++) {
        if(a[k] <= a[1]) b[++ib] = a[k];
        else c[++ic] = a[k];
    }
    // 「下」組を整列
    quicksort(b, ib);
    a[ib+1] = a[1];
    for(k=1; k<=ib; k++) a[k] = b[k];
```

```
// 「上」組を整列
quicksort(c, ic);
for(k=1; k<=ic; k++) a[ib+1+k] = c[k];
}
```

実習 59 (エイトクィーンパズルの解) のプログラム例

```
int check(int Q[], int j, int k) {
    int i;
    for(i=1; i<k; i++) {
        if(j == Q[i]) return 0;
        if(j+k == Q[i]+i) return 0;
        if(j-k == Q[i]-i) return 0;
    }
    return 1;
}

void eightqueen(int Q[], int k) {
    int i, j;
    if(k == 9) {
        printf("found : ");
        display(Q);
        return;
    }
    for(j=1; j<=8; j++) {
        if(check(Q,j,k) == 1) {
            Q[k] = j;
            eightqueen(Q, k+1);
        }
    }
}
```

10.6 章末演習問題

問題 10.1 7.3 節で計算した大きな数のべき乗（のあまり） $a^b \bmod M$ を計算する再帰呼び出しを使った関数 `int bigpowerR(int a, int b, int M)` を書き、それをチェックするプログラムを書きなさい。 $\bmod M$ は「 M で割ったあまり」という計算を表す記号です。

ヒント： $b = 2u$ ならば、 $a^b \bmod M = (a^2)^u \bmod M$

問題 10.2 4.7 節で説明したフィボナッチ数列 $x_n = x_{n-1} + x_{n-2}$ を再帰呼び出しで計算する関数 `int fibonacciR(int)` を書きなさい。再帰呼び出しのない方法で計算するプログラムを書き、最初の 40 項を計算するのに、両方でどれくらい計算時間が違うかを調べなさい。

ヒント：初期条件と停止条件に注意しなさい。

問題 10.3 クイックソートのプログラムを、配列を使わずにポインタを使ったプログラムを書きなさい。

問題 10.4 ハノイの塔の解手順ではなく、3 つの台の円盤がどのように変わっていくか、その経過を表示するプログラムを書きなさい。例えば、円盤の番号を大きいものから並べて表すとすると、次のようにすれば経過が分かるでしょう。

台 1	台 2	台 3	注釈（表示不要）
321	-	-	最初の配置
32	-	1	円盤 1 を台 3 に移す
3	2	1	円盤 2 を台 2 に移す
3	21	-	円盤 1 を台 2 に移す
-	21	3	円盤 3 を台 3 に移す

...

...

ヒント：グローバル変数を使っても良い。その方が楽。

問題 10.5 エイトクイーンパズルの解を全部求めなさい。90 度あるいは 180 度回転させると同じになるとか、鏡に映すと同じになるような解は一つと数えると、異なる並べ方が何通りあるか調べなさい。

息抜きのページ

サッカーのペナルティキックで、キーパーがシュートのコースを読む、というゲームです。

プログラム例

```
/* PK戦のシミュレーション
 * あなたはキーパー、PKのコースを読みなさい。
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    int play, kick;
    char s[5][10] = {"左上", "左下", "真ん中", "右下", "右上"};
    char h[3][40];

    printf("¥n PK戦 で遊びましょう ¥n");
    printf("あなたはキーパーです。
        左上、左下、真ん中、右下、右上のどれかを選んでください ¥n");
    while(1) {
        printf("¥n キッカー登場です ¥n");
        printf("どっちへ飛びますか
            (1:左上、2:左下、3:真ん中、4:右下、5:右上) ");
        scanf("%d", &play);
        printf("シュート!! ¥n");
        kick = rand() % 6;
        if(kick == 0) {
            printf("ナンということ、外しましたあああ!! ¥n¥n");
            continue;
        }
        printf(" +=====+¥n");
        strcpy(h[0], " | |");
        strcpy(h[1], " | |");
        strcpy(h[2], " | |");
        switch(kick-1) {
            case 0: h[0][6] = 'o'; break;
            case 1: h[2][6] = 'o'; break;
            case 2: h[0][18] = 'o'; break;
            case 3: h[2][29] = 'o'; break;
            case 4: h[0][29] = 'o'; break;
```

```
    }
    switch(play-1) {
        case 0:  strncpy(&h[0][8], ">====", 5); break;
        case 1:  strncpy(&h[2][8], ">====", 5); break;
        case 2:  strncpy(&h[1][17], "AAA", 3);
                 strncpy(&h[2][17], "A A", 3); break;
        case 3:  strncpy(&h[2][23], "====<", 5); break;
        case 4:  strncpy(&h[0][23], "====<", 5); break;
    }
    puts(h[0]); puts(h[1]); puts(h[2]);
    printf("=====¥n");
    if(play == kick) {
        printf("止めました！！ ファインセーブ！！");
    } else {
        printf("ゴォォール！！ ");
        if(play != 3 && kick != 3 && abs(play-kick) == 1)
            printf("惜しかったなあ！！！！");
        else printf("残念でした ");
    }
    printf("¥n¥n");
}
}
```

第 11 章

算法 6：方程式を解く

1 変数の方程式 $f(x) = 0$ を満たす x の値、すなわち方程式の解を求める問題は、数学の基本問題としていろいろな場面で出てきます。2 次方程式 $ax^2 + bx + c = 0$ には解の公式があって、係数 a, b, c が与えられると、陽に解を計算することが出来ます。次数が大きくなったり、 $f(x)$ が多項式以外の一般の関数形の場合、解が $x = \dots$ という形で陽に与えられることはまれです。その場合は、数値的に求める必要があり、その計算法を知らなければいけません。

変数が複数になった場合の連立方程式も、やはり多くの場面で出てきますが、特別な工夫が必要です。これらについて解説しましょう。ここで取り上げるアルゴリズムは次のようなものです。

- 1 変数方程式の求解：二分法、ニュートン法
- 連立一次方程式の求解：消去法

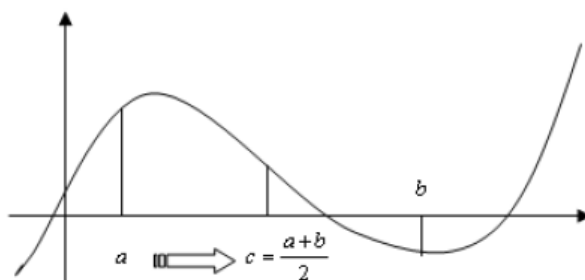
11.1 二分法

$f(x) = 0$ を満たす $x = x^*$ の値（のうちの一つ）を求める問題を考えてみましょう。 x^* を方程式 $f(x) = 0$ の解、あるいは零点（ゼロ点）と言います。原始的ですが、 $f(x)$ が連続ならばどんな場合にも適用できる方法として二分法があります。中間値の定理の応用で、次のような命題が成り立つことが二分法の根拠です。

命題 11.1 $y = f(x)$ が連続関数で、もし $f(a)$ と $f(b)$ の符号が違う二つの実数 $a < b$ を見つけることができれば、 $f(x) = 0$ を満たすような x が区間 (a, b) の中に存在する。

この命題を使えば、方程式 $f(x) = 0$ の数値近似解を計算することが出来ます。もし命題のような a, b が見つかったら、その中点を $c = (a + b)/2$ とします。 $f(c) = 0$ ならばそれで目的達成、 $x^* = c$ です。そうでない場合、 $f(c)$ は $f(a)$ か $f(b)$ のどちらかと異符号

になりますから、異符号になった区間に命題を適用すれば、区間 (a, c) あるいは区間 (c, b) の中に方程式の解があるはずです。これで、探索範囲が半分になりました。これを 10 回続けると探索範囲はほぼ 1000 分の 1 になります。さらに繰り返せば、近似の誤差はいくらでも小さくすることが出来ます。



解を含む区間をまた半分にして、... というように、区間の幅を半分にしなが、解の存在範囲を狭めていくことによって最終的に解に到達しよう、という方法を**二分法**と言います。アルゴリズム風を書けば次のように書けるでしょう。

アルゴリズム 30 $f(a_0) \times f(b_0) < 0$ となるような 2 数 $a_0 < b_0$ が与えられているとき、区間 (a_0, b_0) における $f(x) = 0$ の解を二分法で求める。許容誤差を ε とする。

1. $k = 0$ とする。
2. k を 1 増やし、 $c = (a_{k-1} + b_{k-1})/2$ とする。
3. もし、 $f(c) = 0$ ならば c が解、アルゴリズム終了。
4. もし、 $f(a_{k-1}) \times f(c) < 0$ ならば $a_k = a_{k-1}, b_k = c$ とする。さもないければ $a_k = c, b_k = b_{k-1}$ とする。
5. もし、 $b_k - a_k > \varepsilon$ ならばステップ 2 へ戻る。
6. $(a_k + b_k)/2$ が方程式の近似解。

「許容誤差」は解の精度を決める基準となる数で、問題によって異なる解の精密さの尺度です。繰り返しのたびに解の探索範囲 $[a_k, b_k]$ は半減して行きますから、30 回も繰り返せば実用的には十分の精度の解が得られることが期待できます。

実習 60 二分法のアルゴリズムによって次の方程式の解を求めるプログラムを書きなさい。 $b_k - a_k$ を表示させて、収束の仕方を調べなさい。但し、 $f(x)$ を定義する関数を関数プログラムとして定義し、解くべき方程式が変わったら、その関数プログラムを差し替えるだけで答えが求まるようなプログラムにきなさい。(1) $x - \cos(x) = 0$ 、(2) $x^2 - 2 = 0$ 、(3) $x^3 - x - 1 = 0$ 。

ヒント：アルゴリズム 30 では、計算の進行をはっきりさせるために添え字付き変数を使っていますが、プログラミングで配列変数を使う必要はありません。定義する関数プログラムの中身は `return` 文だけですが、それでも関数化しておいた方が都合が良い。

11.1.1 再帰を使った二分法

二分法で、 $[a, b]$ の中点 c に対して、もし $f(a) \times f(c) < 0$ とすると、 $[a, b]$ で二分法を適用して求められる解と $[a, c]$ で二分法を適用して求められる解は同じものです。したがって、区間 $[a, b]$ で解を求める問題は $f(c)$ の符号次第で、区間 $[a, c]$ で解を求める問題か、区間 $[c, b]$ で解を求める問題に帰着できます。つまり、再帰構造をしています。したがって、アルゴリズムを再帰的な記述で書き換えることも出来るはずです。

$f(a) \times f(b) < 0$ となる $a, b (a < b)$ 二数が与えられたとき、 $[a, b]$ にある方程式の解（の一つ）を $B(a, b)$ とすると、 $B(a, b)$ は次のようにして再帰アルゴリズムで記述することができます。

アルゴリズム 31 $B(a, b)$ （区間 $[a, b]$ における $f(x) = 0$ の解）を二分法で求める

1. $f(a) = 0$ ならば a 、 $f(b) = 0$ ならば b が解、アルゴリズム終了。
2. $c = (a + b)/2$ とする。
3. $f(c) = 0$ ならば c が解、アルゴリズム終了。
4. $b - a < \varepsilon$ ならば c が解、アルゴリズム終了。
5. もし、 $f(a) \times f(c) < 0$ ならば $B(a, c)$ が解、さもなければ $B(c, b)$ が解。

練習問題 11.1 この再帰のアルゴリズム 31 が正しく動くことを納得できるように説明しなさい。

実習 61 方程式 $x - \cos(x) = 0$ の解を再帰的な二分法のアルゴリズムを使って求める関数定義プログラム

```
double nibun(double a, double b)
```

を作りなさい。それを検証するプログラムを書いて正しいことを確認しなさい。関数プログラムは実習 60 で作ったものを使いなさい。

11.1.2 関数ポインタ

関数 `nibun` の定義プログラムではゼロ点を求める関数 `f` を定義する関数プログラムを別に用意して、その関数名を「`if(f(a)*f(c) < 0) ...`」のように使っています。実習とは別の関数のゼロ点を求めたければ、関数 `f` の中身を書き換えるか、`nibun` 関数の内部の関数名 `f` を新しい関数名に書き換えるかしなければいけません。中身を書き換えるとい

うことは前に使った関数が消されてしまうので望ましくありません。また、関数名がいつも `f` というのもおもしろくありません。もし、関数名を引数として引き渡すことができるのであれば、関数名が違って、`nibun` プログラムには手を付けずに、いろいろな関数に対応できるプログラムが書けそうです。

関数プログラムの関数名も変数の一種で、その関数を定義しているプログラムが記憶されている領域の先頭アドレスを記憶します。すなわち、関数名という変数は、配列名がその配列の定義されている領域の先頭アドレスを記憶するポインタであったと同じように、ポインタとして定義されています。関数名はそのため、関数名は**関数ポインタ**とも呼ばれます。そのポインタを実引数として指定すれば（配列名で配列変数がアクセスできるように）関数プログラムにアクセスすることができるようになります。これが関数ポインタの考え方です。

関数ポインタを使った関数定義を考えましょう。関数ポインタの受け渡しは、ポインタを使ったアドレス渡しの方法と全く同じです。関数ポインタを使った関数 `nibun` のプロトタイプを次のように変更します。

```
double nibun(double(*F)(double), double a, double b);
```

ポインタの後に、関数を計算するのに必要な引数の型のリストをカッコでくくって付け加えところが通常のアドレス渡しの場合と違うところです。

次に、関数 `nibun` の中で関数名 `f` をすべて `(*F)` に書き換えます。`*F` を括弧でくくることを忘れないように。例えば、条件判定の命令文は「`if((*F)(a)*(*F)(c) < 0) { ... }`」のようになります。

また、関数 `nibun` を呼ぶときは、`(*F)` に対応する実引数として関数名（ポインタ）を書きます。例えば、二つの関数定義プログラム `kansuu1` と `kansuu2` に対して

```
nibun(kansuu1,a1,b1)
nibun(kansuu2,a2,b2)
```

のように。

実習 62 関数 `nibun` を、関数名を実引数として利用できるように書き換えなさい。それを使って、実習 60 で計算した三つの関数を別々の名前を付けて定義し、三つの方程式の解を同時に求める一つのプログラムを作りなさい。

ヒント：同時に求めるというのは、例えば、次のようにするという事です。

```
printf("一番目の方程式の解は %lf¥n", nibun(kansuu1,a1,b1));
printf("二番目の方程式の解は %lf¥n", nibun(kansuu2,a2,b2));
...
```

練習問題 11.2 実習 62 のプログラムで、関数ポインタの名前を `F` としたとき、関数 `nibun`

の中で「`printf("%d", F);`」という命令を実行しなさい。また、`main` 関数で実変数として使用する関数名を `kansuu1` としたとき、「`printf("%d", kansuu1);`」という命令を実行しなさい。その結果から何が分かりましたか。

11.2 ニュートン法

二分法は関数が連続であればどんな場合にも適用できますが、解に到達するまでにかなり時間がかかるのが欠点です。 $f(x)$ が微分できる場合は、導関数の情報を利用した、収束が早い（少ない繰り返して許容精度以下の解が求まる）ニュートン法アルゴリズムがよく知られています。

$f(x) = 0$ の解の一つを x^* とします。 $f(x)$ は微分可能なので、解 $x = x^*$ の付近の点 $x = x_0$ における $f(x)$ の接線

$$y - f(x_0) = f'(x_0)(x - x_0)$$

が存在し、その接線が x 軸と交わる点 x_1 は（うまくすれば） x_0 よりもっと x^* に近いと考えられます。 x_1 は上の式で $y = 0$ とおけば

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

のように計算できます。

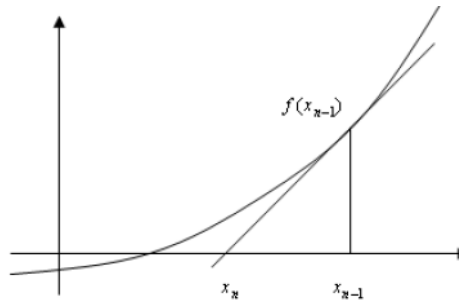
同じ議論を $x = x_1$ で繰り返せば、 x^* により近い x_2 が求まります。この x_2 も

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

によって計算できます。一般に、 x_{n-1} が与えられたら

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

によって x_n を計算する、ということを繰り返すと、いずれは x_n は x^* に収束することが期待できます。このようにして方程式の解を見つける方法を**ニュートン法**と言います。



以上の手順をアルゴリズムとしてまとめておきます。

アルゴリズム 32 $f(x) = 0$ の解をニュートン法で求める。許容誤差を ε とする

1. 初期値 x_0 を与える。 $k = 0$ とする。
2. k を 1 増やす。 $x_{k-1} - f(x_{k-1})/f'(x_{k-1})$ を x_k とする。
3. もし、 $|x_k - x_{k-1}| > \varepsilon$ ならばステップ 2 へ戻る。
4. x_k が方程式の解。

例えば、 $f(x) = x^2 - 2$ とすると、 $f(x) = 0$ の解は $\sqrt{2} = 1.414213562\dots$ ですが、これをニュートン法で解くと次のような経過をたどり、4 回の反復で解に到達しています。

x	$f(x)$	$f'(x)$
2	2	4
1.5	0.25	3
1.416666667	0.006944444	2.833333333
1.414215686	6.0073×10^{-6}	2.828431373
1.414213562	$(4.51061 \times 10^{-12})$	(2.828427125)

初期値の値 x_0 として、必ずしも x^* の付近の値を与えなくても、多くの場合収束してくれることが分かっていますが、たまたま $f'(x_{n-1}) = 0$ となるような場合はうまく動いてくれません。また、例えば、 $f(x) = x^3 - 3x^2 + x + 3$ の場合、 $x_0 = 1$ とすると、

$$x_1 = 1 - \frac{2}{-2} = 2, x_2 = 2 - \frac{1}{1} = 1$$

となって、二つの値を行ったり来たりするだけになってしまいます。こういう場合の脱出方法を考えておく必要がありますが、ここではこれ以上深入りはしないことにします。

さらに、この方法で計算できるのは一つの解だけなので、例えば $f(x) = x^3 - x = x(x-1)(x+1)$ のように、解がたくさんある場合、すべての解を見つけるためには、初期値をおのおのの解の近くに設定して、一つずつ別々に計算する必要があります。

実習 63 (1) ニュートン法で $x^3 - x = 0$ の解を計算するプログラムを書きなさい。初期値の値をいろいろ変えてテストしなさい。 $|x_k - x_{k-1}|$ を表示させて収束の仕方を調べなさい。(2) 初期値を正の値とし、許容誤差をいろいろに変えて、ニュートン法の解と真の値 (a の平方根) との差の絶対値がどれくらいの大きさになるか実験しなさい。(3) 方程式 $f(x) = 0$ の解をニュートン法で求める関数 (関数名を `newton` としなさい) を書いて、それを使って $x^3 - x = 0$ の解 (のうちの一つ) を求めなさい。但し、(1) と (3) では許容誤差 ε は 10^{-8} としなさい。C で 10^{-8} は `1e-8` と表記します。

ヒント：関数 $f(x)$ と導関数 $f'(x)$ は別々の関数定義プログラムを定義しなさい。出来る人は関数ポインタを使ったプログラムにも挑戦しなさい。

11.2.1 数値微分

ニュートン法を使うには導関数が必要です。解きたい方程式ごとに、関数と導関数の二つの関数定義プログラムを作るのは面倒です。関数だけ定義すれば、その導関数は自動的に計算できるような方法を考えましょう。これは数値微分法といって、きちんと作ろうとするとかなりやっかいですが、ここでは簡易的な方法で済ませましょう。関数 $f(x)$ が $x = a$ で微分可能ならば、十分に小さい $h > 0$ に対して、

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$$

であることは分かっているので、これを使って導関数を近似計算すれば良いでしょう。導関数をこのように近似する方法は**中心差分法**と言います。そうすると、次のようなプログラムで関数 F の導関数 dF が計算できることが分かります。

プログラム例

```
double dF(double (*F)(double), double x) {  
    double h=0.0001;  
    return ((*F)(x+h) - (*F)(x-h)) / (2*h);  
}
```

導関数の定義関数プログラムを上の中点差分法で計算するようにしておけば、関数形が変わっても、その関数の関数定義プログラムだけを書き換えればニュートン法が使えます。

関数ポインタを使ってニュートン法のプログラムを書いた場合は、導関数は関数形に依存しないので、導関数については関数ポインタなしで呼び出すことが可能です。

練習問題 11.3 中心差分法の精度は h の値に大きく依存することは理解できます。その近似計算の精度を調べるプログラムを書き、そのプログラムを使って、 $f(x) = x^4$ としたとき、 $f'(0.5)$ を $h = 0.1, 0.01, 0.001, 0.0001, 0.00001$ として近似計算したものと、真の値との差がどうなるか調べなさい。

11.3 連立一次方程式を解く

次のような連立一次方程式

$$\begin{cases} 2x + 4y + 6z = 8 & (1) \\ 2x + 6y + 2z = 6 & (2) \\ 3x + 5y + 7z = 9 & (3) \end{cases}$$

を解く場合は、例えば $(1) - (2)$ と、 $(1) \times 3 - (3) \times 2$ を計算して y, z の連立一次方程式を作り

$$\begin{cases} -2y + 4z = 2 \\ 2y + 4z = 6 \end{cases}$$

それを解いて $z = 1, y = 1$ を求め、それを (1) に代入して $x = -1$ を求める、という手順でした。どう組み合わせたら計算しやすい式が得られるか、係数を見て消去しやすい変数を見つけるといのが、解くための工夫だったりします。

コンピュータでは、係数が何であっても、計算の手間は変わらないので、以下で説明するような機械的に出来る方法が用いられます。それには線形代数の知識（というほどのものではありませんが）を使います。係数だけを取り出して並べた 3×3 の行列を係数行列と言い A と書きます。右辺の定数を並べたベクトルを定数ベクトルと言い b と書きます。 $x = (x, y, z)^T$ と書くと（は転置を意味する記号です）、与えられた連立方程式は

$$Ax = b$$

とまとめることができます。

命題 11.2 ある方程式だけ定数倍したもので置き換えても、解は同じ。

例えば、(1) 式だけ 2 で割って (0.5 倍して)

$$\begin{cases} x + 2y + 3z = 4 & (4) \\ 2x + 6y + 2z = 6 & (2) \\ 3x + 5y + 7z = 9 & (3) \end{cases}$$

という連立一次方程式を作ったとしても、その解は前の連立一次方程式の解と同じです。

命題 11.3 ある式を、その式と別の式を定数倍したものを足したもの（あるいは引いたもの）で置き換えても解は同じ。

例えば、(4) 式を 2 倍したものを (2) 式から引いたものを (2) 式と置き換えると

$$\begin{cases} x + 2y + 3z = 4 \\ 2y - 4z = -2 \\ 3x + 5y + 7z = 9 \end{cases}$$

という連立一次方程式が出来ますが、その解は元の連立一次方程式の解と同じです。

この二つの性質を使って、連立一次方程式の係数を変形していく（ゼロの係数を増やす）ことを考えます。連立一次方程式の上の変形は、係数行列と定数ベクトルだけを取り出してその動きを追いかけた方が、何をやっているのかが明らかになって分かりやすいので、

$$Ax = b \Leftrightarrow \begin{pmatrix} 2 & 4 & 6 \\ 2 & 6 & 2 \\ 3 & 5 & 7 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 9 \end{pmatrix} \Leftrightarrow \left[\begin{array}{ccc|c} 2 & 4 & 6 & 8 \\ 2 & 6 & 2 & 6 \\ 3 & 5 & 7 & 9 \end{array} \right]$$

という形で表現して、係数行列、定数ベクトルの動きだけを追いかけることにします。慣れれば便利ですが、最初は違和感があるかもしれません。慣れないうちは、1 列目の数字に x 、2 列目の数字に y 、3 列目の数字に z を掛けて足したものが 4 列目の数字に等しい、という式に翻訳しながら読み進めるようにしなさい。

連立一次方程式を解く手順

1. 1 行目を（対角要素の）2 で割り、その 2 倍を 2 行目から引き、その 3 倍を 3 行目から引く（1 行目を除き、1 列目の要素を 0 にする）。

$$\left[\begin{array}{ccc|c} 2 & 4 & 6 & 8 \\ 2 & 6 & 2 & 6 \\ 3 & 5 & 7 & 9 \end{array} \right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 2 & 6 & 2 & 6 \\ 3 & 5 & 7 & 9 \end{array} \right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 2 & -4 & -2 \\ 0 & -1 & -2 & -3 \end{array} \right]$$

2. 2 行目を（対角要素の）2 で割り、その 2 倍を 1 行目から引き、その 1 倍を 3 行目に加える（2 行目を除き、2 列目の要素を 0 にする）

$$\Rightarrow \left[\begin{array}{ccc|c} 1 & 2 & 3 & 4 \\ 0 & 1 & -2 & -1 \\ 0 & -1 & -2 & -3 \end{array} \right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 7 & 6 \\ 0 & 1 & -2 & -1 \\ 0 & 0 & -4 & -4 \end{array} \right]$$

3. 3 行目を（対角要素の）-4 で割る。その 7 倍を 1 行目から引き、-2 倍を 2 行目に加える（3 列目の対角要素を除く要素を 0 にする）

$$\Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 7 & 6 \\ 0 & 1 & -2 & -1 \\ 0 & 0 & 1 & 1 \end{array} \right] \Rightarrow \left[\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{array} \right]$$

最後に得られたものを、連立一次方程式の形に直すと

$$\begin{cases} x = -1 \\ y = 1 \\ z = 1 \end{cases}$$

これが解です。

この手順では、命題 11.2, 11.3 だけを使って連立方程式を変形し、各変数について、一つの式を除いてその係数をゼロにする（その式から消去する）ということを順番に繰り返しています。計算のしやすい変数を見つけて... というようなことは考えずに、与えられた変数の並びに従って機械的に消去を繰り返します。この解き方をガウスの**消去法**、あるいは**掃き出し法**と言います。

実習 64 (1) 次の連立一次方程式に対して、消去法の手順を実践し、解を求めなさい。

$$\begin{cases} 2x - 2y + z = -3 \\ x + y + 2z = 1 \\ x + 2y + 3z = 2 \end{cases} \Rightarrow \left[\begin{array}{ccc|c} 2 & -2 & 1 & -3 \\ 1 & 1 & 2 & 1 \\ 1 & 2 & 3 & 2 \end{array} \right]$$

(2) 上の手順を参考に、一般の係数行列と定数ベクトルが与えられた 3 元連立一次方程式を解く C のプログラムを作りなさい。但し、どの段階でも対角要素はゼロにならないことを期待して構いません。

for 文が何重にもなってきますので、注意深い作業が必要です。が、むづかしいところはありません、面倒なだけです。「面倒くさい」ことをいやがるようでは、プログラミングは上達しません。

変数を一つの式を除いて連立方程式の各式から消去していくという消去法の基本的な考え方はこれまでの説明通りですが、何か不都合はないでしょうか？ 実習 64 の但し書きは何のために書かれているのでしょうか？

例えば、連立一次方程式の最初の式が $2x + 4y + 6z = 8$ ではなくて $4y + 6z = 8$ だとしたら、「対角要素で割って」という手順が実行できません。上の手順は計算途中も含めて、すべての対角要素が 0 にならないことが大前提です。しかし、このような場合でも、可能な限り計算を続けることが可能です。それには、二つの式を入れ替えた連立一次方程式

$$\begin{cases} 4y + 6z = 8 & (1') \\ 2x + 6y + 2z = 6 & (2) \\ 3x + 5y + 7z = 9 & (3) \end{cases} \Rightarrow \begin{cases} 2x + 6y + 2z = 6 & (2) \\ 4y + 6z = 8 & (1') \\ 3x + 5y + 7z = 9 & (3) \end{cases}$$

が最初の連立一次方程式と同じ解を持つ、ということを思い出すだけで十分です。これは行列操作で言えば、「二つの行を入れ替える」ことに当たります。

そこで、もし計算の途中で対角要素が 0 になったら、その列の中でその行よりも下ににある要素の中でゼロでないものを探し、ゼロでない要素を持つ行とその行を入れ替えてから上の手順に持ち込みます。対角要素が 0 で、その列の下部分がすべて 0 ならば、与えられた連立一次方程式には解がありません。係数行列が正則でない（係数行列の逆行列が存在しない）、という場合に当たります。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ 二分法による方程式の求解
- ☐ ☐ 再帰を使った二分法アルゴリズム
- ☐ ☐ ニュートン法
- ☐ ☐ 数値微分
- ☐ ☐ 連立方程式を解く消去法アルゴリズム

参考 実習プログラムの例（主要部分のみ）

実習 62（再帰的二分法により方程式の解を求める）のプログラム例

```
double nibun(double (*F)(double), double xa, double xb) {
    double xc = (xa+xb)/2, EPS=1e-8;
    if((*F)(xa) == 0) return xa;
    if((*F)(xb) == 0) return xb;
    if((*F)(xc) == 0) return xc;
    // 停止条件
    if(xb - xa < EPS) return (xa+xb)/2;
    // 再帰呼び出し
    if((*F)(xa) * (*F)(xc) < 0) {
        return nibun(F, xa, xc);
    } else {
        return nibun(F, xc, xb);
    }
}
```

実習 63（ニュートン法により方程式の解を求める）のプログラム例

```
// 関数定義
double F3(double x) {
    return x*x*x - x;
}
```

// 中心差分法による近似導関数

```
double dF(double (*F)(double), double x) {  
    return 3*x*x - 1;  
}
```

// ニュートン法

```
double newton(double (*F)(double), double x) {  
    double y, EPS=1e-8;  
    do {  
        y = x - (*F)(x) / dF(F,x);  
        printf("%lf, %le¥n", y, fabs(y-x));  
        if(EPS > y - x && y - x > -EPS) break;  
        x = y;  
    } while(1);  
    return y;  
}
```

実習 64 (連立一次方程式の解を求める) のプログラム例

```
void linearequations(double a[][10], double b[], int n) {  
    int i, j, k;  
    for(i=1; i<=n; i++) a[i][n+1] = b[i];  
    // 係数行列の上三角化  
    for(i=1; i<=n; i++) {  
        for(j=i+1; j<=n+1; j++) a[i][j] /= a[i][i];  
        a[i][i] = 1;  
        for(k=1; k<=n; k++) {  
            if(k == i) continue  
            for(j=i+1; j<=n+1; j++) a[k][j] -= a[k][i]*a[i][j];  
            a[k][i] = 0;  
        }  
        display(a, n); // 途中経過表示用関数  
    }  
}
```

11.4 章末演習問題

問題 11.1 実習 63 のプログラムの導関数定義プログラムを、数値微分で近似的に導関数の値を計算する関数に置き換えて作り直しなさい。正確な導関数を使った場合と、中心差分を使った近似導関数を使った場合で、解の違いを調べなさい。

問題 11.2 対角要素がゼロになったら行を入れ替えて計算するというチェックを取り入れた n 元連立一次方程式を解くアルゴリズムを書きなさい。そのアルゴリズムを基にプログラムを作りなさい。また、入れ替えてもゼロでない要素が見つからない場合は解がないと表示して停止するというようなチェック機能を付け加えて、 n 元連立一次方程式を解く完全なプログラムを作りなさい。

問題 11.3 上の問題で書いたアルゴリズムをちょっと変えるだけで、 $Ax = b, Ax = c$ という二つの連立一次方程式を同時に解くことが出来ます。どうすればよいでしょうか。

問題 11.4 その方法を拡張すると、同じような方法で正方行列の逆行列を計算するアルゴリズムが出来ます。それを説明しなさい。

第 12 章

算法 7：モンテカルロ法

ランダムな要素のない数学の問題を確率論を使って定式化し直し、乱数を使って近似的に解を求める（推定する）方法を**モンテカルロ法**、あるいは**モンテカルロシミュレーション**と言います。例えば、定積分を求める問題は被積分関数の不定積分が陽に書けない場合は数値的に計算するしかありませんが、それは適当な確率変数の期待値とみなすことができるので、乱数を使った期待値の推定問題に持ち込むことができます。それ以外にも、コンピュータの計算能力をフルに生かして数値的に答えを探索するこの方法は、多くの分野で使われている、きわめて有効な方法です。

- モンテカルロ法の手順
- 推定精度
- 擬似乱数の生成アルゴリズム

について解説します。

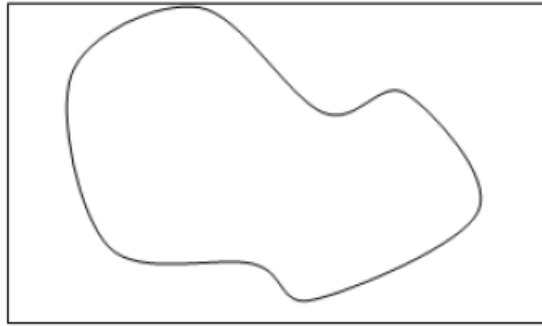
この計算法の理屈を正確に理解するためには確率論の知識、特に期待値の知識、が必要です。詳しくは「確率とその応用」（逆瀬川著、サイエンス社）6 章を見てください。

12.1 モンテカルロ法

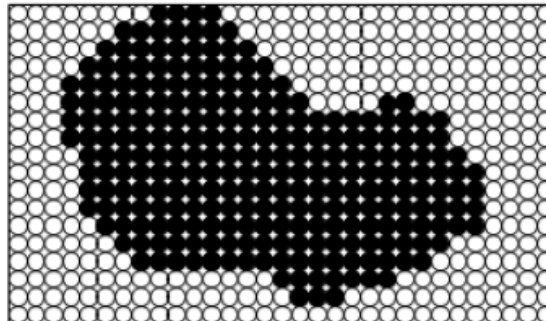
定積分の計算を考えます。定積分は被積分関数の不定積分を使って計算できましたが、微分と違い不定積分はいつもきれいな関数形で書けるとは限りません。例えば、正規分布の密度関数で出てくる e^{-x^2} という関数の不定積分は初等関数の範囲で表すことが出来ません。したがって、そのような関数を被積分関数とする定積分を求めるためには、数値的な方法をとるしかありません。ある関数の定積分を数値的に計算する方法は数値積分法といって、昔から研究が盛んに行われていますが、ここでは乱数を使って**推定する**方法を説明します。

12.1.1 面積を推定する

定積分は被積分関数と x 軸の間の面積を計算するものです。そこで、始めに、関数を離れて一般の図形の面積について考えてみましょう。例えば、曲線に囲まれた土地の面積を知りたいときは、三角測量と言って、土地を近似的に小さな三角形に分解して、一つ一つの三角形の辺の長さを正確に測り、面積を正確に求めて積み上げるという方法が用いられます。そんな精密な測量をしなくても、航空写真と「ビー玉」だけを使ってアバウトに面積を見積もる方法があります。



まず（航空写真の）図形を、全体が含まれるような長方形で覆い、次に長方形いっぱいにビー玉を敷き詰めます。図形に含まれるビー玉の個数を勘定して、全体のビー玉の個数で割れば、目的の図形の長方形に対する相対面積が、アバウトですが計算できます。その数字に長方形の面積を掛ければ求めたい図形の面積が求まります。ビー玉の大きさを小さくして行けばどんどん正確な値に近づいていくことが期待できます。



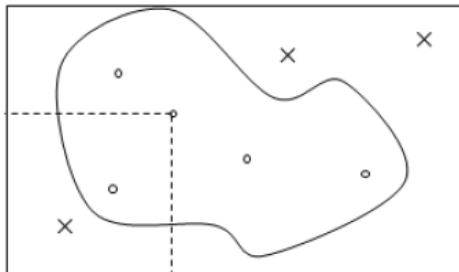
この方法の欠点は、ビー玉を芥子粒のように小さくしたとき、数えるのが面倒になることです。そこで、次のような無精なことを考えます。芥子粒を敷き詰めたら図形に含まれる芥子粒だけに黒く色を付けます（図の●は図形に入っている芥子粒、○は図形からはず

れている芥子粒だと思ってください)。境界がでこぼこしていますが、芥子粒のように小さければなめらかになるので、気にしない。

色づけが終わったら芥子粒を全部集めて袋に入れてよくかき混ぜ、その中から一掴みの芥子粒を取りだして色の付いた芥子粒の相対個数を数えます。相対個数とは、取りだした芥子粒の総個数の中にある色の付いた芥子粒の個数の割合です。芥子粒が良くかき混ぜられていれば、この相対個数は芥子粒全体に含まれる色つき芥子粒の割合、すなわち図形の相対面積、とほぼ等しいと考えてよいでしょう。味噌汁の味見をするときに良くかき混ぜればひとすくい全体で味が分かる、ということと同じ理屈です。このようにして、一部分の情報だけから全体の姿を推し量る方法を**標本調査**と言います。

記号を使って書いておきましょう。取り出した芥子粒の総数を N 、色の付いた芥子粒の数を M 、長方形の面積を S とすると、 $S \frac{M}{N}$ はほぼ図形の面積に等しいということです。直観的には、 N を相当に大きくしないと誤差が大きく使い物にならないと思いますが、確率論を使って計算してみると、意外に少ない数でかなり正確な値が把握できることが分かります。実際、標本調査の代表例として内閣支持率調査を取り上げると、1 億人の意見を知るために必要な調査対象者、つまり N は 2000 程度です。

この「芥子粒実験」をコンピュータで代行させることを考えます。長方形の縦横の辺を RAND_MAX ($\text{rand}()$ 関数の上限値) + 1 等分して細かな網目を作り、各マスに芥子粒があると考えます。そうすると、($\text{rand}()$, $\text{rand}()$) はランダムに選ばれた一つのマス目 (芥子粒の座標) とみなすことができます。したがって、 $2N$ 個の $\text{rand}()$ を使えば N 個の芥子粒をランダムに選ぶことができるので、そのうちで対象図形に含まれる芥子粒の個数を M とすれば、図形の相対面積が M/N で近似できることは実際の芥子粒を使った実験と同じです。このように、ランダムな要因の入り込む余地のない面積という確定的な値を、標本調査の考え方を適用して**乱数を使って推定**するのが**モンテカルロ法**、あるいは**モンテカルロシミュレーション**と呼ばれる方法です。

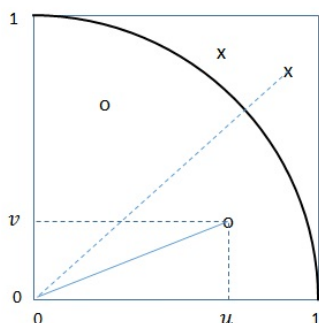


このやり方で、単位円の 4 分の 1 の面積を「推定」してみましょう。答えは $\frac{\pi}{4}$ ですから、その推定値から円周率が分かる、ということになります。

$$\frac{M}{N} \approx \frac{\pi}{4} \Rightarrow \pi \approx 4 \frac{M}{N}$$

芥子粒を使って円周率が推定できる！

四分円を覆う長方形は、一辺が半径の長さ 1 を持つ正方形とすればよいでしょう。この単位正方形と単位円の四分円とに対して上の「芥子粒実験」を適用します。ランダムなマス目（の左下座標）は `rand()/(RAND_MAX+1.0)`（あるいは `(double)rand()/(RAND_MAX+1)`）を 2 つ組み合わせれば生成できます。この実験を手順としてまとめたのが次のアルゴリズムです。



アルゴリズム 33 (モンテカルロ法による円周率の推定)

1. `rand() / (RAND_MAX+1.0)` を 2 つ生成して、それらを u, v とし、 $u^2 + v^2 \leq 1$ ならば 1、さもなければ 0 とする。
2. この計算を N 回繰り返して、 N 個の計算結果の平均値を計算する。
3. それを 4 倍したものを円周率 π の推定値とする。
4. このような実験を繰り返して、推定値のばらつきを調べ、推定の誤差の程度を知る。

以降、`rand()/(RAND_MAX+1.0)` によって生成される数を $[0, 1)$ **一様乱数** と呼ぶことにします。`rand()` は 0 以上 `RAND_MAX` 以下なので、0 以上 1 未満になるからです。

練習問題 12.1 今までの説明では、条件を満たす点の個数の相対度数を計算していましたが、その説明と、ステップ 2 の平均値が同じであることを説明しなさい。

実習 65 このアルゴリズムを使って、円周率を推定するプログラムを書き、 $N = 1000$ として推定値を計算するという実験を 5 回繰り返し、5 個の数値のばらつきを調べなさい。さらに 100 回繰り返して、それらの推定値の平均値と標準偏差を計算し、実験結果のヒストグラムを描きなさい（平均値と標準偏差の計算プログラムと、ヒストグラム作成プログラムは自分のプロジェクトの中から探して利用しなさい）。

ヒント：プログラムの主要部は例えば、次のようになるでしょうか？

プログラム例

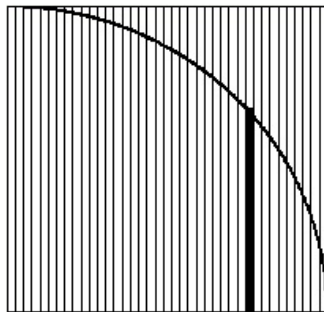
```
u = rand() / (RAND_MAX+1.0); v = rand() / (RAND_MAX+1.0);
if(u*u+v*v<=1) count++;
```

rand 関数は予想の付かないでたらしめな数を生成するので、アルゴリズム 33 のステップ 3 で得られる推定値は計算のたびに異なる値が得られるはずですが、アルゴリズムのステップ 1 から 3 を 1 回だけやって推定値を一つだけ計算しても信頼性は少ないことを、ステップ 4 の繰り返し実験の結果から確認しなさい。

練習問題 12.2 (RAND_MAX+1.0) で割らずに RAND_MAX で割れば、1 も含めて閉区間 [0, 1] での一様乱数ができると考えて「u = rand() / RAND_MAX;」と書きましたがうまくいきません。なぜでしょうか？ 意図を正しく達成するためにはどうすればよかったのでしょうか。

12.1.2 別の推定法

芥子粒を敷き詰める代わりに細い針金を縦にびっしり並べて図形にかかった部分に色を付けておき、それらの中から何本かの針金を取り出す、という方法でも面積を推定することができます。芥子粒の場合は、取り出した芥子粒の中の色の付いた芥子粒の個数を数えました。針金の場合は取り出した針金の色の付いた部分の長さを測るという手間をかけることにします。



すべての針金について色の付いた部分の長さを測り、針金の幅（太さ）を掛ければ図形を短冊の集まりとして近似した面積が求めることができます。数学的に考えると、四分円の面積は

$$S = \int_0^1 \sqrt{1-x^2} dx$$

と表現され、その積分は

$$S \cong \sum_{i=0}^{L-1} (x_{i+1} - x_i) \sqrt{1 - x_i^2}$$

によって近似できるという、リーマン積分の考え方を適用したものと考えられます。

リーマン積分の場合は $L-1$ 個の分点 $\{x_i\}$ は必ずしも等間隔でなくても良いのですが、めんどうなので、 $x_{i+1} - x_i = h = \frac{1}{L}$ (一定) とすると

$$S \cong h \sum_{i=0}^{L-1} \sqrt{1 - x_i^2} = \frac{1}{L} \sum_{i=0}^{L-1} \sqrt{1 - x_i^2}$$

となり、面積は $\{\sqrt{1 - x_i^2}; i = 0, \dots, L-1\}$ の平均値で近似できることが分かります。ここからがモンテカルロ法です。 L がとても大きければ、 $\{\sqrt{1 - x_i^2}; i = 0, \dots, L-1\}$ を全部調べなくても、それらの中からいくつかを取り出したものの平均値で近似できるのではないかと考えます。芥子粒実験になぞらえて言えば、びっしり敷き詰めた針金からランダムに何本かの針金を選んで四分円にかかる長さを測定し、その平均値を推定値とするのです。

そこで乱数の登場です。 $[0, 1)$ 一様乱数、すなわち区間 $[0, 1)$ で一様分布する数を $u_1, u_2, \dots, u_N$ として、 $\sqrt{1 - u_1^2}, \sqrt{1 - u_2^2}, \dots, \sqrt{1 - u_N^2}$ の平均値を計算すると $S = \pi/4$ が推定できることになります。アルゴリズムとしてまとめておきましょう。アルゴリズム 33 と比較対照しながら読みなさい。

アルゴリズム 34 モンテカルロ法による円周率の推定 (針金実験)

1. $[0, 1)$ 一様乱数 u を一つ生成し、 $y = \sqrt{1 - u^2}$ とする。
2. このような実験を N 回繰り返して、 N 通りの実験結果 $y_1, y_2, \dots, y_N$ を得る。
3. その平均値 $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$ を 4 倍したものの $4\bar{y}$ を円周率の推定値とする。
4. このような実験を繰り返して、推定値のばらつきを調べ、推定の誤差の程度を知る

実習 66 このアルゴリズムを使って、円周率を推定するプログラムを書き、 $N = 1000$ として推定値を計算するという実験を 5 回繰り返し、5 個の数値のばらつきを調べなさい。さらに 100 回繰り返して、それらの推定値の平均値と標準偏差を計算し、実験結果のヒストグラムを描き、実習 65 の結果と比較しなさい。

定積分のモンテカルロ法による推定

一般に、区間 $[0, 1]$ で定義された、値が 0 以上 1 以下の関数 $f(x)$ を考えると、その定積分は関数 $f(x)$ と x 軸、 $x = 0$ と $x = 1$ の二つの直線で囲まれた図形の面積に等しくなりますから、その関数を単位正方形で覆えば、上の「芥子粒実験」「針金実験」によって

定積分 $\int_0^1 f(x)dx$ が推定できることになります。アルゴリズム 33 の「芥子粒実験」では、四分円を表す関数 $f(x) = \sqrt{1-x^2}$ の定積分にこの考えを適用しました。

アルゴリズム 33 のステップ 1 の判定条件を書き換えると、 $v \leq \sqrt{1-u^2} = f(u)$ となりますが、一般の関数の定積分もこれが判定条件になります。すなわち、2 つの $[0, 1)$ 一様乱数 u, v を生成し、 $v \leq f(u)$ ならば 1、さもなければ 0 というデータをたくさん集めて平均を計算したものが定積分の推定値となります。一方、アルゴリズム 34 の「針金実験」では、 $f(u)$ という関数値を集めて平均を取れば定積分の推定値が得られます。

アルゴリズム 33 はせっかく関数値 $f(u)$ を計算したのに、別の様乱数を使って、その乱数が関数値よりも大きい小さいかによって 0 か 1 という数値に置き換えたものしか利用していません。したがって、アルゴリズム 34 に比べて雑な推定になり、推定値のばらつきが大きくなるだろう、ということは直感的に分かります。二つの実習でもそのような結果が得られるでしょう。その理屈については、次節で説明します。

練習問題 12.3 $f(x) = 1 - (2x-1)^2$ に対して $\int_0^1 f(x)dx$ の値を「針金実験」のモンテカルロ法で推定し、 N の大きさによって推定値がどれくらいばらつくのかを調べなさい。

12.1.3 確率の問題を解く

ある事象の起きる確率を求めるために、乱数を使ってランダム事象を擬似的に作り出し、その事象が起きたかどうかを調べる、という実験を繰り返し、事象の起きた相対度数（全体の実験回数の中の起きた回数の割合）を確率の推定値とする、という方法があります。これもモンテカルロ法の適用分野で、モンテカルロシミュレーション、あるいはモンテカルロ実験と言います。例えば、ポーカーでワンペアができる確率を求めたいとすれば、何度も何度も 5 枚カードを配って、ワンペアのできた相対度数を計算すれば、目的の事象の起きる確率のおおよその値が分かるでしょう。実際にトランプを配って実験するのは大変ですが、コンピュータの乱数を使えば、ポーカーゲームを模倣し、事象の起きる確率を推定することができます（9 章末問題）。

次の実習は、「2 枚のコインを投げたとき、表が 2 枚出るか、1 枚出るか、1 枚も出ないか、3 通りあるので、その確率はそれぞれ 3 分の 1 である」という「主張」が正しいかどうかをチェックするためにモンテカルロシミュレーションを適用した例です。実際にコインを投げる代わりに、乱数を使って、コンピュータで確率実験を実現できます。`rand()` 関数は 0 以上 `RAND_MAX` 以下の乱数を生成しますから、そのうちの半分を「おもて」、残りを「うら」と読み替えればよいでしょう（3.3 節の練習問題参照）。実際に実験してみましょう。

実習 67 次のプログラム（主要部）は 2 回のコイン投げで表の出た回数を数える実験で、

結果を度数分布にまとめようとしているものです。入出力部分を追加してプログラムを完成させ、実験回数 n を 10, 100, 1000, 10000, ... と増やしていったときに表が 1 回だけ出る相対度数（度数を全体の度数で割ったもの）を計算し、結果がどうなるか、調べなさい。

プログラム例

```
1:      for(i=0; i<=2; i++) kekka[i] = 0;
2:      for(i=0; i<n; i++) {
3:          omote = 0;
4:          if(rand() <= RAND_MAX/2) omote++;
5:          if(rand() <= RAND_MAX/2) omote++;
6:          kekka[omote]++;
7:      }
```

実習 67 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. 2 回のコイン投げを実施するために `rand()` 関数を 2 回呼んで、その都度、「おもて」の出た回数をカウントして、それを度数分布に追加する、というだけのプログラムです。 n を増やすにつれて `kekka[1]` の相対度数がだんだんと 0.5 に近づいていく様子が観察できたはずですが（結果が 3 通りだからと言って確率は 3 分の 1 ではない）。
- 2. もっと複雑な乱数を必要とする場合は、 $[0, 1)$ 一様乱数を使って目的の乱数に変換する方が分かりやすいでしょう。この場合は「`if(rand()/(RAND_MAX+1.0) < 0.5)` ...」となります。

練習問題 12.4 サイコロを 5 回振って、その目の合計が 25 を超える確率をモンテカルロ実験で推定しなさい。

練習問題 12.5 ポーカーで、最初に配られる 5 枚の中にワンペアができて確率をモンテカルロ実験で推定しなさい（真の値は「組み合わせ」の数を計算すれば分かる）。

12.2 モンテカルロ法の推定精度

実習 65 や実習 66 を実行すると、5 通り、あるいは 100 通りの推定結果は異なり、どの数値が「正しい」のか判断できません。推定結果の数値を正しく読み取るヒントは、この実習の元になった「芥子粒実験」（あるいは「針金実験」）にあります。同じ数の芥子粒を取り出したとき、毎回同じ個数の色付き粒が含まれていることは期待できませんが、元の色付き粒の割合と「おおよそ」同じ相対個数が抽出されることは期待できます。そして、実験を繰り返すと色付き粒の割合に近い値ほど得られやすいことも想像できます。実際、実習で描いたヒストグラムは統計で出てくる正規分布の形と似ています。

結局、毎回の推定結果には規則性がなく、ヒストグラムを描くと正規分布に近いグラフが得られることから、実験結果の個々の数値は統計分析で扱う正規母集団からの独立標本と同じような性質を持っていることが期待できます（実際そうなります）。したがって、その実験データを正しく解釈するためには統計の知識が必要です。

今は真の平均値が π と分かっていますが、実際の問題で真の平均値は未知です（だからモンテカルロ法という計算道具を持ち出したのですが）。そこで、実験結果を $x_1, x_2, \dots, x_n$ としたとき、これら n 個のデータからどのようにすれば真の平均値が「推定」できるか、という問題を考えることになります。

誰でも思いつくのは、データの全体の平均値

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n}$$

で真の値を推定することです。この例では $\bar{x} = 3.142$ となるので、ほぼ正確に円周率の推定値が得られたことになります。その推定値がどれくらい正しいのかを知る必要があります。そこで出てくるのが、統計で学んだ信頼区間の考え方です。せっかく学んだことなので、復習しておきましょう。 $x_1, x_2, \dots, x_n$ の標準偏差を

$$s = \sqrt{\frac{1}{n-1} \sum_{k=1}^n (x_k - \bar{x})^2}$$

とします。6 章では $n-1$ ではなく n で割ったものを標本標準偏差と定義しましたが、信頼区間を計算する場合には n ではなく $n-1$ で割ったものを使います。その理屈は統計の教科書を調べてください。

命題 12.1 平均 μ 、標準偏差 σ の正規母集団から大きさ n の標本を取ったとき、 n が十分に大きければ、母平均 μ の 95% 信頼区間は近似的に次で与えられる。

$$\left[\bar{x} - 2 \frac{s}{\sqrt{n}}, \bar{x} + 2 \frac{s}{\sqrt{n}} \right] \quad (12.1)$$

実習 65 の例では、 $n = 100$, $\bar{x} = 3.142$, $s = 0.052$ だったので、これらを代入すると

$$\left[3.142 - 2\frac{0.052}{10}, 3.142 + 2\frac{0.052}{10} \right] = [3.132, 3.152]$$

となります。95% 信頼区間というのですから、逆に言えば 5% 不信頼区間なので、この主張は 5% の間違いを含んでいます。ということは、例えば 20 人が同じような実験をしてこの理論に基づいて信頼区間を計算すると、19 人が正しく、一人は間違った主張をする危険性があるというわけです。あるいはまた、あなたが実習をあと 19 回実行して、上のような計算をしたとすると、平均的に一回は円周率を含まない信頼区間が得られることになります。今回は幸いなことに、この実験結果から計算された信頼区間は真の値を含んでいたため 19 人の方に入ることができました。くどいようですが、信頼区間が真の値を含むということが分かるのは答えを知っているからであって、未知の問題に対して、いつ何時 5% の方になるとも限りません。モンテカルロ法による推定はそのようなリスクがあるのです。

実習 68 (実習 65 の続き) 実験結果から 95% 信頼区間を計算するプログラムを書いて、実験結果を評価しなさい。また、同じ実験を 20 回繰り返して 20 個の信頼区間を作り、推定が失敗した回数を調べなさい。

問題によっては、信頼区間の幅をもっと小さくしたいこともあるかもしれません。信頼区間の幅は式 (12.1) から分かるように、 s に比例し、 n の平方根に反比例します。 n を大きくすれば信頼区間の幅は小さくなりますが、その減少の仕方は緩やかです。

練習問題 12.6 (実習 65 の続き) $N = 100000$ として同じ実験を行いなさい。 $N = 1000$ の場合と比較して分かったことを説明しなさい。

s を小さくするためには実験結果 $x_1, x_2, \dots, x_n$ のばらつきを小さくする、すなわち一回一回の実験の精度を上げることが必要です。そのための工夫の一つとして、前節で芥子粒を針金に変えるという実験を考えました。芥子粒を数える場合は色が付いていれば 1、さもなければ 0 であったのに対して、針金の場合は、色の付いた部分の長さを測るという手間を掛けたおかげで個々のデータのばらつきが小さいことから、推定値のばらつき、すなわち s が小さくなることが期待できます。

実習 69 (実習 66 の続き) 実験結果から 95% 信頼区間を計算するプログラムを書いて、実験結果を評価しなさい。また、同じ実験を 20 回繰り返して 20 個の信頼区間を作り、推定が失敗した回数を調べなさい。実習 68 の結果と比較しなさい。

モンテカルロ法のもう少し詳しい解説は前述の「確率とその応用」に書いてありますので、興味のある人は読んでください。

12.3 擬似乱数の生成

モンテカルロ法を使って計算するために、乱数が必要です。今までは VC2010 の関数 `rand()` を使っていましたが、その中身がどうやって作られているか、調べてみましょう。

きちんと理解するには高度な数学が必要ですが、ここでは実験的に、ランダムそうに見える数列が意外に簡単な方法で生成できることを体験することを目指します。

例えば、33 の 2 乗、3 乗、... を計算してその下 2 桁だけを取り出すと、順に、89, 37, 21, 93, 69, 77, 41, 53, 49, 17, 61, 13, 29, 57, 81, 73, 9, 97, 1 となり、21 乗すると 33 に戻ることが確かめられます。33 の後はまた、89, 37, ... が続きます。このような数列は長さ 20 の**周期**を持つと言います。

この数列の 10 の位の数字だけを取り出して並べると、8, 3, 4, 9, 6, 7, 4, 5, 4, 1, 6, 1, 2, 5, 8, 7, 0, 9, 0, 3, 8, ... となりますが、計算の規則を忘れて、この数列の最初の 10 個の数字だけから 11 個目の数を予測することが出来ますか？ 33 の代わりに適当な数を決めて同じような計算をしてごらん下さい。

33 とか「下 2 桁」ではなくもっと大きな数を取ると、周期はもっと長くなり、数列の次の数を予測することはますます困難になるでしょう。この数列を一般化すると、正の整数 a と P を決め、数列の第 n 項 r_n に a を掛けたものを P で割ってそのあまりを r_{n+1} とする、とまとめることが出来ます。数式で書くと

$$r_{n+1} = a \times r_n \bmod P$$

となります。mod は 6.5 節で出てきましたが、 P で割ったあまりを表す記法です。 $a = 33, P = 100, r_0 = 1$ とすると、確かに上の 2 桁数列が得られることを確かめてください。C の `rand` 関数はこのような考え方を基に作られています。^{*1}

6.5 節で説明したように C の `int` 型数は 4 バイトで記憶され、`int` 型同士の計算でそれ以上大きくなると、オーバーフローとなり、 2^{32} で割ったあまりが計算結果になります。つまり、`int` 型数同士の計算は何も言わなくても $\bmod 2^{32}$ で実行されているので、 $P = 2^{32} = 4294967296$ とすれば、 $\bmod P$ の計算は省略することが出来ます。 P をこのように取った場合、 a として適当に大きな数で、8 で割って 5 が余る数とし、 r_0 として適当な奇数を取ると、周期が十分に長いランダムな数列が生成できることが知られています。

生成される数は 1 以上 2^{32} 未満という中途半端な数なので、それを 2^{32} で割って、0 より大きく 1 未満の `double` 型の数にしておいた方が使い勝手がよいでしょう。このようにして作られたのが、次の実習プログラムの `myRand()` です。

^{*1} 正確な生成法が 237 ページにありますので、参考にしてください。

実習 70 次の関数 `myRand` と `randomDigit` によってどのような数が生成されるのかを調べ、それを適切にテストする `main` 関数を作り実行しなさい。

プログラム例

```

// 1 以上 M 以下のランダムな整数を作る
1:  int randomDigit (int M, unsigned int *seed) {
2:      return (int)(myRand(seed) * M) + 1;
3:  }
// 0 より大きく 1 未満の乱数を作る
4:  double myRand (unsigned int *seed) {
5:      unsigned int A=39984229;
6:      double R32=.2328306365e-9; // =2-32
7:      (*seed) *= A;
8:      return R32 * (*seed);
9:  }
// main 関数の中身
...
10:  unsigned int seed=1234567; // 適当な大きな奇数
...
11:  a[i] = myRand(&seed);
...
12:  b[i] = randomDigit(M, &seed);
...

```

実習 70 の解説 チェックボックスに ✓ を入れながら読みなさい。

- ☐ 1. `myRand()` 関数についてはすでに説明済みです。0 より大きく 1 より小さい `double` 型の乱数を生成します。ここでは $a = 39894229$ という数を使っています。`unsigned int` 型は 0 以上 2^{32} 未満の正の整数だけを表現するデータ型です。`seed` は x_n に対応しています。値が更新されるので、ポインタを使ってアドレス渡しにしています。
- ☐ 2. `randomDigit()` 関数は、「`myRand()*M`」で 0 より大きく、 M より小さい `double` 型の数が生成され、それを `(int)` でキャストしているので、小数点以下が切り捨てられ、0 から $M - 1$ までの整数値となり、それに 1 を加えていますから、1 以上 M 以下の `int` 型乱数（これも一様乱数という）が生成されます。ちょうど「 M 面体」のさいころ（想像上の産物）を振ったときに出る目のような数列のようなものです。

掛け算 1 回でランダムな数列を作るこの方法

$$x_n = ax_{n-1} \bmod P$$

は**乗算合同法**と呼ばれ、擬似乱数の簡易的な生成法として使われます。 x_n を P で割って 0 より大きく 1 未満の数にしたものは $(0, 1)$ 一様乱数と呼ばれます (0,1 という値は取らないので、开区間です)。乗算合同法は、ちゃんと調べると、いくつかの問題が浮かび上がってきます。 x_n は P 未満の数ですから、数列の最初の P 個の中には同じものが混じっているはずで、もし x_n と x_m が同じならば x_{n+1} と x_{m+1} も同じ数になるはずです。この議論を繰り返すと、 x_m から先の数列は x_n から始まる数列と同じ、ということです。このような数列は**周期列**と呼ばれます。ということは、記憶力の優れた人は、次に何が来るか、正確に言い当てる事が出来るということになります。確かにそれはその通りなのですが、 $P = 2^{32}$ 、 a を 8 で割って 5 余る数としたことにより、その周期は $2^{30} (> 10^9)$ になることが分かっているので、そんなに大量の数列を使うのでない限り、周期があるということには気にしなくて良いことになります。

実際のモンテカルロ法の計算では、大量の乱数を必要とし、 2^{30} ではとても足りないという場合もあるので、もっと長い周期を持つ数列を作り出す方法も考えられています (興味のある人は、伏見「乱数」(東大出版会)を読みなさい)。

練習問題 12.7 実習 70 で定義された `randomDigit()` を使って、サイコロ数列 (1 から 6 までの乱数) を 100 個生成し、その度数分布を計算するプログラムを書き、乱数の影響でどれくらい出現度数がばらつくのかを調べなさい。生成する個数を 1000 個、あるいは 10000 個にした場合はどうなりますか? ヒマな人はコンピュータでなく、実際にサイコロを振ってばらつきを調べ、コンピュータ乱数を使った場合の結果と比べてみてもらなさい。

練習問題 12.8 実習 70 で定義された `myRand()` 関数を 2 回使って 2 個の乱数を生成し、その差を計算する、という実験を 1000 回繰り返し、1000 個の値の度数分布を計算して、特徴を説明しなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ モンテカルロ法
- ☐ ☐ 標本調査の方法
- ☐ ☐ モンテカルロ法による円周率の推定
- ☐ ☐ モンテカルロシミュレーションによる事象の生起確率推定
- ☐ ☐ モンテカルロ法による推定精度
- ☐ ☐ 信頼区間
- ☐ ☐ 乗算合同法による擬似乱数列生成
- ☐ ☐ unsigned int 型数 (復習)
- ☐ ☐ アドレス渡し (復習)

参考 実習プログラムの例 (主要部分のみ)

実習 65 (モンテカルロ法で円周率の推定) のプログラム例

```
printf("実験回数と点の個数 ");
scanf("%d %d", &n, &N);
for(i=1; i<=n; i++) {
    count = 0;
    for(j=0; j<N; j++) {
        u = rand() / (RAND_MAX+1.0);
        v = rand() / (RAND_MAX+1.0);
        if(u*u + v*v <= 1) count++;
    }
    est[i] = 4.0 * count / N;
}
histogram(est, n, 2.5, 0.05, 20);
```

実習 66 (実習 65 と違う箇所)

```
for(i=1; i<=n; i++) {
    sum = 0;
```

```
    for(j=0; j<N; j++) {
        u = rand() / (RAND_MAX+1.0);
        sum += sqrt(1-u*u);
    }
    est[i] = 4 * sum / N;
}
```

実習 68 (モンテカルロ法で信頼区間の計算) のプログラム例

```
void histogram(double a[], int n, double c0, double H, int m);
double mean(double a[], int n);
double variance(double a[], int n);
void confidence(double a[], int n, double *lower, double *upper) {
    double av, sd;
    av = mean(a, n);
    sd = sqrt(variance(a, n));
    *lower = av - 2*sd/sqrt(n);
    *upper = av + 2*sd/sqrt(n);
}
// main 関数の主要部
    histogram(data, n, 3.05, 0.01, 20);
    confidence(data, n, &lower, &upper);
    printf("円周率の 95% 信頼区間: [%lf, %lf]¥n", lower, upper);
```

12.4 章末演習問題

問題 12.1 モンテカルロ法による定積分の計算では、関数 $f(x)$ の定義域が $[0, 1]$ で、その値が $0 \leq f(x) \leq 1$ という範囲にある、という制約がありました。しかし、その制約は、「任意の有界閉区間 $[a, b]$ と任意の有界関数」に緩和することができます。どこをどのように修正すればよいか考えなさい。

問題 12.2 上の問題を使って、次の定積分を推定しなさい。被積分関数は標準正規分布の密度関数として知られています。定積分の値は統計の教科書に載っています。

$$\int_{-1.96}^{1.96} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$$

問題 12.3 単位円周上にランダムに 3 点を取ってできる 3 つの弧を直線に伸ばして 3 つの線分を作った時、それらが 3 角形の 3 辺になれる確率をモンテカルロシミュレーションで推定しなさい。

ヒント: 区間 $[0, 1]$ にランダムに 3 点を取って切り離し、左端区間と右端区間はつなげて一つの線分とすると考えても同じ。

問題 12.4 クラス会に 20 人集まったとして、その中に同じ誕生日を持った人がいる可能性はいくつくらいですか、モンテカルロシミュレーションで調べなさい。25 人集まったとしたらどうですか。但し、ある人の誕生日（月と日）は 365 の中からランダムに選ばれるものとします（月、日に分ける必要はない）。

問題 12.5 トランプをよく切って、5 枚のカードを並べたとき、同じ数のペアが二組、残りの 1 枚はそれらの数とは違う数となっている（ポーカーのツーペア）確率はどれくらいになるか、モンテカルロ実験で推定しなさい。

問題 12.6 500 円玉を裏が出るまで投げ続け、それまでに出了た表の回数を n としたら 2^n 万円もらえる、という賭けを考えます。表が出続ければ 1000 万円も夢ではない。この賭けの参加費としていくらまで出せるか、という問題です。そのために、賞金額の期待値を知る必要があります。モンテカルロシミュレーションでこの賭けを擬似体験し、獲得賞金額の平均値を計算しなさい。賭けの回数を増やしていったとき、平均値の増え方を観察し、分かったことを説明しなさい。

第 13 章

ファイル入出力

文法編その 5

集計作業のように大量のデータを扱う場合は、プログラムを実行するたびに一々キーボードからデータ入力しているのは大変です。また、表示する場合も、計算結果が多い場合はディスプレイで表示できる量に制約があるため、古い計算結果は新しいもので上書きされてしまい、せっかく表示させても見直すことができません。このような場合は、あらかじめワープロや Excel でデータファイルを作っておくとか、結果をファイルに記憶させるようにします。こうしておけば、ワープロでファイルを扱うように、入力データや計算結果を保存しておくことができ、入力データをちょっと変えたり、結果を Excel で分析したり、というような作業が簡単にできるようになります。

- **ファイルへの出力**：fopen, fclose, fprintf、FILE 型、出力バッファ
- **ファイルからの入力**：fscanf、入力バッファ、EOF

13.1 ファイルへの出力 (fopen, fclose, fprintf)

計算結果が多数行に渡る場合、ディスプレイに表示させるだけだと、古いデータは消去されてしまう危険性があります。また、表示に時間が取られ非効率です。ディスプレイに表示させる代わりに、あるいは同時に、ファイルへ保存することで、この危険性と非効率性を避けることができます。その方法を解説しましょう。

文法予備知識

1. データをファイルに記憶させる場合は、`fopen`（ファイルを開く）、`fprintf`（データを書き込む）、`fclose`（ファイルを閉じる）という命令文を使う
2. データを書き込む先はファイルではなく、コンピュータ内部の出力バッファである
3. 出力バッファの場所（アドレス）を記憶するために `FILE` 型変数を使う
4. 出力バッファのデータは VC2010 が自動的にファイルに書き込む

実習 71 (1) 次のプログラムを入力して実行しなさい。

(2) このプログラムを保存したフォルダーの中に、1 行目で入力した名前のファイルが見つかるはずですから、エディタを使って開き、どういうデータが書き込まれているかを調べなさい。

(3) 6 行目「"%d "」（「d」の後にスペースあり）の代わりに「"%d"」（「d」の後のスペースを削除）として実行し、作成されたファイルの中味を調べてごらんなさい。

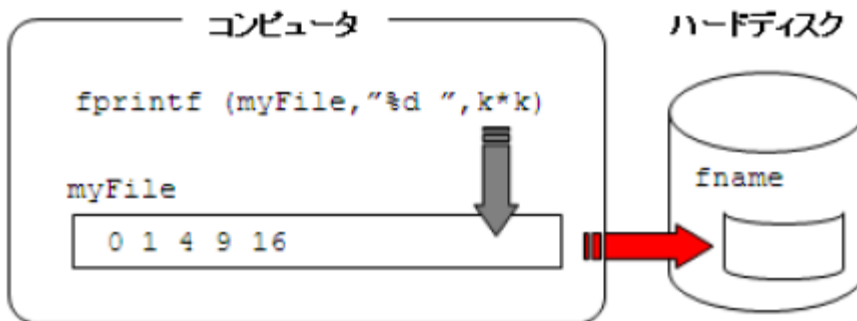
プログラム例

```
// ファイルへの出力
1:      char fileName[]="fileIOtest.txt";
2:      FILE *myFile;
3:      int k;
4:      myFile = fopen (fileName, "a");
5:      for(k=0; k<10; k++) {
6:          fprintf(myFile, "%d ", rand()/4096); // %d の後に空白あり
7:      }
8:      fclose (myFile);
```

実習 71 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **（出力バッファ）** コンピュータを使わずに普通の書類ファイルを使う場合を考えてください。最初にファイルを開き、次いで必要な情報をそこに記録して、最後にファイルを元の場所にしまう（閉じる）、という手順を踏むでしょう。コンピュータでファイルを扱う場合も同じですが、コンピュータの処理速度の違いによる制約を考慮しなければいけません。ファイルは、普通のワープロで用いられるファイルと同じように、ハードディスクや USB メモリに作られますが、そこへのアクセスはコンピュータの内部メモリへのアクセスに比べて時間がかかります。そこで、内部メモリの一部に仮想ファイルとも呼ぶべき**出力バッファ**を設け、そこへいったん

書き込み、ある程度たまったらファイルへ転送する、という手順を踏みます。次の図はそのイメージを描いたものです。ファイルへの転送は VC2010 がやってくれますので、ユーザは気にする必要はありません。



- 2. 実習プログラムの4行目の `fopen` は「出力バッファを開く」命令、6行目の `fprintf` は「出力バッファにデータを書き込む」命令、8行目の `fclose` は「出力バッファを閉じる」命令です。いずれの命令文にも登場する `myFile` は出力バッファを定義する変数で、2行目で定義されています。
- 3. (FILE 型ポインタ、構造体) 出力バッファを定義するためには、書きだすメモリのアドレスや、転送先のファイル名、など様々な仕様を決める必要があります。これらの情報は**構造体**と呼ばれるデータ構造にまとめられ、その構造体には FILE という名前が付けられています。2行目は、(`int *p` から類推できるように) FILE という型の構造体を作成し、その先頭アドレスを `myFile` という変数に記憶する、という命令文になっています。言い換えれば、FILE 型ポインタの `myFile` を定義する宣言文です。
- 4. (データ型としての char 型、char 型配列) ファイルを識別するためにファイル名が必要ですが、ファイル名という文字列を扱うために、新たに char 型というデータ型を導入します。char 型は1バイトからなり、半角一文字を記憶し (char は character の略)、文字列を扱うためには char 型の配列を利用します。プログラム1行目は char 型配列変数の定義と初期値を設定する命令文です。文字列は `printf` 文の最初の引数のように「" "」で囲みます。「.txt」はテキストファイルに付けるファイル識別子で、こうしておくことで、ファイルをダブルクリックするだけでエディタ (ワードのメモ帳) を使って表示させることができるようになります。
- 5. (ファイルの書き込みモード) 4行目の `fopen` で出力バッファとファイル名を結びつけています。2番目の引数の「"a"」は書き込み方式のパラメータで、同じファ

イル名が無ければ新規に作成し、もしその名前を持つファイルがすでにあれば、既存のファイルの記入されたデータのあとに追加して書き込みなさい、という指示を表しています (append の a)。`"a"`の代わりに`"w"`と書くと、既存のファイルがあればいったん消去し、新規に作成します。大事なデータが不用意に消されないために、ふつうは`"a"`を使います。

- 6. (**データの書き込み**) `fprintf` の書き方は、最初の引数のポインタ指定を除けば `printf` の書き方と全く同じです (`printf` の前の `f` は file の `f`)。出力バッファを、見えないディスプレイと考えてください。注意しなければいけないのはデータの区切り方です。実習 (3) のようにスペース区切りなしに書き込んでしまうと、例えば `int` 型数の場合、どこがデータの切れ目だか分からない、巨大な数になってしまいます。
- 7. (結果の確認) プログラムの実行終了後、実習 (2) によって、指定したファイル名を持つファイルが作成されていることを確認することができます。作成したファイルはふつう `C` のプログラムと同じプロジェクトフォルダにあるはずです。

ファイルを扱う場合は、相手が目に見えないだけに、意図したとおりに動いていない場合、何が起きたのかを追跡するのはかなりやっかいです。最初は上のプログラムを丸ごと覚えてプログラムを書き、徐々にバリエーションを増やすようにしなさい。

練習問題 13.1 名前と携帯電話番号だけからなる電話帳ファイルを作り、正しく書き込まれているかどうかチェックしなさい。

練習問題 13.2 `rand()` 関数、あるいは実習 70 で作った `myRand()` (あるいは `randomDigit()`) 使って、大量の 1 桁乱数 (0 から 9 までのランダムな数字) をファイルに書き込みなさい。エディタを使って、生成されたファイルがそれらしいものになっているかどうかをチェックしなさい。

13.2 ファイルからの入力 (fscanf)

データの入力にはミスタイプをしないように最新の注意が必要で、データがたくさんある場合はくたびれます。プログラムのデバッグが繰り返し必要な場合は、同じデータを何度も入力しなければならず、うんざりします。この作業をスキップするために、データをあらかじめファイルに入力しておき、実行時にそのファイルから読み込むことができれば、作業がはかどることは間違いありません。このための手順について解説します。

文法予備知識

1. データをファイルから読み込む場合は、ファイルを開き (fopen)、データを読み込み (fscanf)、閉じる (fclose) という手順を踏む
2. データはファイルから自動的に入力バッファに取り込まれるものを読む
3. 入力バッファのアドレスを記憶するために FILE 型変数を使う
4. 全部読み込むと入力バッファが空になる (EOF という) ので、それを終了の合図とする

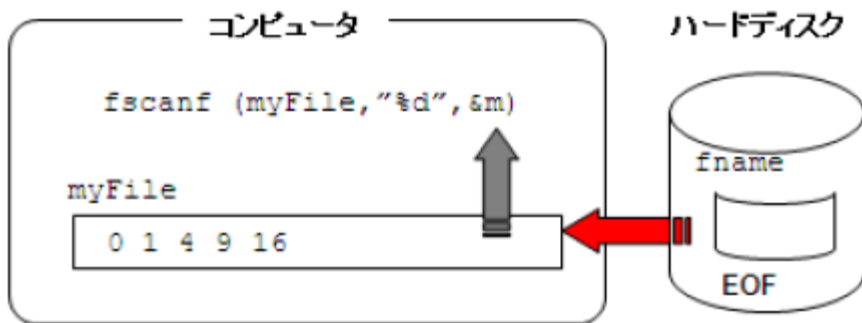
実習 72 次のプログラムを入力しなさい。プログラムを保存したプロジェクトファイルに、実習 71 で作成したファイルをコピーしてからプログラムを実行し、6 行目で、そのファイル名を入力しなさい (ファイル識別子もお忘れなく)。

プログラム例

```
// ファイルからデータ入力
1:   char fileName[50];
2:   FILE *myFile;
3:   int k,m;
4:   printf("ファイル名を入力しなさい  ");
5:   scanf("%s", fileName);
6:   myFile = fopen (fileName, "r");
7:   k=0;
8:   while(fscanf(myFile, "%d", &m) != EOF) {
9:       printf("%d 番目のデータは %d です。¥n", ++k, m);
10:  }
11:  fclose (myFile);
```

実習 72 の解説 チェックボックスに ✓ を入れながら読みなさい。

- 1. **(ファイルからの入力)** あらかじめデータを記録しておいたファイルからデータを読み込む場合は、前節の手順でデータの流れが逆になり、出力バッファが入力バッファに置き換わったと思えば理解は容易でしょう。ここでもイメージ図を先に出しておきます。ファイルの代わりにメモリの一部を**入力バッファ**として設定して、入力バッファを開き、VC2010 がファイルから入力バッファに転送してきたデータを順に読み込み、すべてのデータの読み込みが終わったら入力バッファを閉じる、という手順を踏みます。



- 2. **(ファイル入力モード)** 入力バッファを設定し (2 行目、FILE *myFile)、入力バッファを開き (6 行目、fopen)、入力バッファを閉じる (11 行目、fclose) という手順は、データをファイルに書き込む場合と全く同じです。fopen の 2 つ目の引数「r」は入力の仕方を指定するパラメータで、読み込み専用のファイルを開く、ということを意味しています (read の r)。
- 3. **(入力データ用ファイルの作成、保存)** 入力データはスペースあるいは改行区切りで作成し、実習プログラムの保存されているプロジェクトフォルダに保存しなさい。この実習では、5 行目でファイル名を実行時にキーボードから入力するようにしています。ここで入力した名前が 6 行目のファイル名として使われますので、作成したファイル名と拡張子も含めて完全に一致する文字列を入力しなければいけません。
- 4. **(フォーマット指定子%s)** 文字列を入力する場合のフォーマット指定子は「%s」です。配列名の fileName がポインタであることを思い出しなさい (アドレス渡し)。
- 5. **(データの読み込み)** データを読み込む場合は fscanf 命令を使います。fscanf は、最初の引数のポインタ指定を除けば、キーボードからデータを入力させる時に使った scanf と形式は同じです (scanf の前の f は file の f)。但し、キーボード

入力と違い、ファイルの場合は「これでおしまい」という記号がファイルの最後に記録されているので、これを読み取ったときに入力を終えることができます。その記号を読み取ると、関数 `fscanf` は `EOF` という値（マクロ記号定数）を関数値として返してきます。したがって、8 行目の `while` の条件式は、「`fscanf` を実行し、返ってきた関数値が `EOF` に等しくない（かどうか）」という内容になっています。「`EOF` に等しくないかぎり」は」ということは、ファイルにデータがあるかぎり、ということと同じです。この条件式によって、ファイルのすべてのデータを取り出す、という作業を行うプログラムになっています。この書き方は丸ごと覚えておいてください。

練習問題 13.3 練習問題 13.1 で作成した電話帳のデータを使って、電話番号の末尾が偶数の人だけを表示させるプログラムを書きなさい。

練習問題 13.4 練習問題 13.2 で作ったファイルから乱数データを読み込み、度数分布を計算して簡易棒グラフを表示するプログラムを書きなさい。

本章で学んだ重要事項チェックリスト

本章で学んだ重要事項をまとめておきますので、知識の確認に使ってください。A:テキストなしに説明できる、B:テキストを見れば思い出せる、F:テキストを改めて読み直さないと説明できない、の3段階で各項目を評価し、F 評価がある場合は、今のうちに復習しなさい。

前章の章末にあるチェックリストをもう一度チェックし、F 評価の項目について猛勉強しなさい。

- ☐ ☐ ファイルへの出力、出力バッファ
- ☐ ☐ `fopen, fprintf, fclose` 文
- ☐ ☐ `char` 型、`char` 型配列、フォーマット指定子「`%s`」
- ☐ ☐ `FILE` 型、構造体
- ☐ ☐ ファイルへの書き込みモード「`"a"`」,「`"w"`」
- ☐ ☐ ファイルからの入力、入力バッファ
- ☐ ☐ ファイルからの読み込みモード「`"r"`」
- ☐ ☐ `fscan` 文
- ☐ ☐ EOF

13.3 章末演習問題

問題 13.1 10000 までの素数をファイルに書き込みなさい。そのデータを使って、8 桁の数の因数分解をするプログラムを書きなさい。

問題 13.2 友人 10 人以上の携帯電話データベースを作りなさい。項目は「氏名」「大学名」「年齢」「携帯電話番号」「メールアドレス」の 5 項目とします。

問題 13.3 (続き) そのデータベースを使い、大学毎の人数分布を計算するプログラムを書きなさい。

問題 13.4 受験番号、性別 (0 が男、1 が女)、情報処理、確率、統計、3 科目の試験成績という 5 項目からなる個人データを適当に作り、データベースファイルを作りなさい。

問題 13.5 (続き) そのデータベースを使い、3 科目合計の大きい順に表示するプログラムを書きなさい。

付録 A

記号一覧表

A.1 算術演算子

記号	意味
+	加算（足し算）
-	減算（引き算）、符号の反転
*	乗算（かけ算）
/	除算（割り算）
%	剰余（あまり）

A.2 代入演算子

記号	例	その意味
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	n/=m	n = n / m
%=	n %= m	n = n % m
++	k++	k = k + 1
--	k--	k = k - 1

A.3 関係演算子

記号	意味
>	より大きい
>=	以上
<	より小さい (未満)
<=	以下
==	等しい
!=	等しくない

A.4 論理演算子 (優先順)

記号	意味
!	否定
&&	かつ (論理積 AND)
	または (論理和 OR)

A.5 データ型 (ビット数) (例外あり)

型の名前	意味
int(32)	-2^{31} 以上 $2^{31}-1$ 以下の整数
unsigned int(32)	0 以上 $2^{32}-1$ 以下の整数
double(64)	絶対値 1.7×10^{-308} 以上 1.7×10^{308} 以下、正負
char(8)	-128 以上 127 以下の整数として使える
unsigned char(8)	0 以上 255 以下の整数として使える

A.6 フォーマット指定子 (printf, scanf)

記号	意味
%d	int 型 (整数)
%u	unsigned int 型 (整数)
%lf(%f)	double 型 (%f は printf で使える)
%le(%lE,%e,%E)	科学計算用 double 型 (浮動小数点型数)
%g (%G)	科学計算用 double, int 型 (浮動小数点数)
%c	char (文字一つ、'a')
%s	char (文字列、"string")
%p	pointer (16 進数表示)
%o	8 進数 (符号なし整数)
%x,%X	16 進数 (符号なし整数)
%%	% の表示

A.7 エスケープ文字 (printf)

記号	意味
¥n	改行
¥t	タブ
¥f	改ページ
¥',¥",¥¥	それぞれ文字 ', ", ¥ を表す

A.8 予約語

if, else, while, do, for, goto, break, continue, return
switch, case, default
char, double, float, int, long, short, signed, unsigned
struct, union, void
auto, static, extern, register, const, enum
sizeof, typedef, volatile

A.9 ヘッダファイル (#include で使用)

名前	意味
stdio.h	標準入出力関数
stdlib.h	system()、絶対値、乱数、型変換、メモリ割り当て
math.h	数学関数
time.h	時刻ほか

A.9.1 数学関数 (math.h)

fabs, sqrt, exp, log, log10, pow
 sin, cos, tan, asin, acos, atan, sinh, cosh, tanh
 ceil, floor, fmod

A.9.2 その他の数学関数 (stdlib.h)

abs, labs, rand, srand

A.9.3 標準入出力関数 (stdio.h)

printf, scanf, getchar, putchar, gets, puts
 fprintf, fscanf, fopen, fclose

A.10 ファイルのモード (fopen)

記号	意味
r	ファイルのデータを読み込む
w	データを書き込むためにテキストファイルを作る
a	データをテキストファイルに追加する
w+	読み込み／書き込みのためにテキストファイルを作る
a+	読み込み／書き込みのためにテキストファイルを作るか、あればそこに追加

付録 B

良くある質問

B.1 システムの問題

質問 1 たった 10 行のプログラムを作るのに、「ソリューション」だ、「プロジェクト」だ、と恐ろしい名前が付けられますが、もっと簡単にできる方法はありませんか？

回答 VC2010 を使う限りはあきらめなさい。普通のエディタを使ってプログラムを作成し（字下げサービスはありません）、「コマンドプロンプト」画面を使ってプログラムを実行させるという、プロ仕様の方法があります。興味のある人はネットで調べてください。

B.2 プログラムの書き方

質問 2 良いプログラムの書き方を教えてください。

回答 これは難しい質問だ。「良い」にもいろいろな基準があるでしょうが、初めのうちは**読みやすい**プログラムを心がけてください。そのためには、プログラムの流れが分かる、ということが重要です。if ... else 文、while 構文、for 構文が単純で、部分作業のつながりが分かるようなプログラムを書いてください。そのためにも、注釈をたくさん書く、実行文を短くする、変数は意味ある名前にする、全体構造が分かるように書く、ひとまとまりの作業は関数化しておく、などの注意は常に有益です。

問題が少しややこしくなってきたら、日本語でアルゴリズムを書き、擬似コード化することにより、仕事の流れを把握するようにしなさい。そうすることで作業効率を上げる事が出来るはずです。

B.3 エディタの使い方

質問 3 「int main() {」のあとに Enter キーを押しても字下げされません。なぜ？

回答 字下げは拡張子「.c」の付いたファイルへのサービスです。新規文書を作成したら、直ちに「.c」を付けたファイル名で保存しなさい（実習 1 をやり直しなさい）。

質問 4 プログラムをいじっているうちに字下げがぐちゃぐちゃになってしまいました。一々タブを付けたり取ったりは面倒です。何か良い方法はありませんか。

回答 「編集」→「選択範囲のフォーマット」メニューを実行しなさい。字下げはプログラムの構造を掴む上で重要な情報です。そのために、エディタのサポート機能を活用すべきです。**新しい行を挿入する場合は、前の行の行末にカーソルを置いてから Enter キーを押しなさい。** そうすればぐちゃぐちゃにはなりません。

質問 5 「編集」→「選択範囲のフォーマット」メニューを選んでも、あきらかにおかしい字下げが残ったままです。どうしてですか

回答 **VC2010 にもバグがあります。** すべてをソフトウェアに頼るのは危険です。全部自分でやるのは大変なので手伝ってもらっている、程度に考えなさい。最終的にはすべてあなたの責任です。

B.4 プログラムの実行

質問 6 F5 キーを押しても何も反応しません。パニック！！

回答 1 回実行したあと、実行画面を閉じないで VC2010 を編集していませんか？ あるいは、VC2010 のウィンドウがアクティブになっていますか？ 何かのはずみで、VC2010 のウィンドウ以外のところをクリックするとそういう症状が出ます。

質問 7 プログラムをビルドしてエラーが無く、実行画面も表示されたが、プログラム通りの内容が表示されない。表示されたものは、どうやら前のものらしい。

回答 そのプログラムが**スタートアッププロジェクト**になっていないためと思われます。一つのソリューションの中に複数のプロジェクトを作った場合、F5 キーを押して実行されるのはスタートアッププロジェクトと決められています。実行させたいプロジェクトをスタートアッププロジェクトに変更するには、そのプロジェクトの上でマウスを右クリックし、メニューで指定します。

B.5 デバッグ

エラーを無くすことをデバッグ debug と言います。日本語では「(害) 虫 (bug) 取り (de-)」。

質問 8 コンパイルしたら、エラーが 10 行以上表示されて、途方に暮れています。どうすればよいでしょうか。

回答 エラーが表示されているウィンドウをクリックし、上向き・下向き矢印キーを操作してエラーメッセージの行を選ぶと、そのエラー原因の行が表示されるので、それをダブルクリックすると、プログラムのエラー箇所矢印が表示されます。エラーメッセージを見ながら、自分の書いたプログラムのおかしいところを修正しなさい。

極端な場合、1 箇所を修正しただけで数十行のエラーメッセージがすべて消え消えた、というようなこともありますので、訳の分からないメッセージ、原因が見あたらないメッセージはひとまず置いて、確実なエラーを修正してもう一度ビルドしてみる、という試行錯誤のやり方がよいでしょう（実習 1 のプログラムで、「//」を「/」に換えてビルドしてごらんください。うわー！）。

質問 9 警告のメッセージがあってもエラーが 0 ならば実行画面に移行しますが、そのまま実行しても正しい結果が得られるのですか？

回答 警告の度合いに依ります。例えば、「データが失われた可能性があります」という警告の場合、それが意図的であれば、無視しても構いませんが、うっかりそうしているのであれば、当然報いを受けることになるでしょうね。「scanf」に対して必ず「scanf_s」にしろ、という警告が出ますが、無視してかまいません。毎回同じ警告が出されるのが気持ち悪ければ、指示に従ってください。

質問 10 文法エラーもなく実行しているらしいのだが、うんともすんとも言わない、フリーズか？

回答 `for(k=10; k>0; k++)` のような終了条件を書いていませんか。printf+scanf 文を使って変数の途中経過をトレースしなさい。

質問 11 見た目は間違っていないし、エラーも警告も 0 ですが、実行結果がおかしい。どこがいけないのですか。

回答 こういう質問が多いのですが、よほど明らかなプログラムミス、入力ミスでない限り簡単には答えられません。エラー原因を発見するための特別な処方箋があるわけではな

く、あなたと一緒にプログラムを1行ずつ調べていくしかないのですね。自分で、犯しやすいミスのデータベースを整備して、自分でデバッグできる力を付けていかないと将来困るのはあなたです。忘れがちなのは、

「コンピュータは自分が思ったようには動かない。入力された通りに実行するだけ」

ということです。if...else... や for,while 構文は勘違いが多く発生する可能性があります。どうしても分からないときは、変数の内容を書き換えた後に printf を挿入して、書き換えられた変数の内容を表示し、チェックしなさい。

そもそもアルゴリズムが間違っている、という場合はどうしようもない。一旦思い込んだら何度プログラムを読み返しても気がつかない、ということは良くあります（日常生活でも経験することでしょう）。テストデータとして、エラーが予想されるデータを作り、それを入力してチェックする、ということを繰り返して、自分で気がつくしかない。

ブレークポイントを使って、プログラムの実行中に一時停止させながらプログラムの動きを追うというデバッグのやり方もあります。索引で調べてください。

上のような努力をした上で、万策尽きた、という場合は、注釈付きのプログラムと入力データ、エラーの発生する状況と、エラーメッセージを添えて質問してください。いきなり「このプログラム、どこが悪いのですか」と聞かれても分かりません。読んで分かるような注釈を付けてください。

B.6 文法：データ入力、表示

質問 12 (1) 「scanf("%d ",&a);」と書いたプログラムでデータを入力しても先へ進みません。なぜですか。(2) 「scanf("%d %d",&b,&c);」で「1,2」と入力したのに c におかしな数が入力されている。「scanf("%d,%d",&d,&e);」で「35 12」と入力したのに e におかしな数が入力されている。どうしてですか。

回答 (1) 「%d」の後の空白が余計です、削除しなさい。「それだけで動かないの？」と言われても、そういう約束になっているのでどうしようもない。(2) 「%d」と「%d」の間が空白ならば入力の時も空白で区切り、「,」ならば「,」で区切るという約束です。それぞれ「1 2」、「35,12」と入力していれば良かった。

B.7 文法：配列

質問 13 集計作業でデータを扱うとき、データを記憶させるために2次元の配列を使いたいのですが、1列目は int 型、2列目は double 型とするようなことは出来ますか。

回答 配列を宣言するとき、その配列の型は一つに決まってしまうから、int 型配

列の一部だけを `double` 型とすることは出来ません。`double` 型と宣言しておいて実質 `int` 型を記憶させ、表示の時 `%.01f` という変換指定をするしかありません。**構造体**というデータ型がありますが、これについてはネットで調べてください。

質問 14 配列の大きさを無限大にすることは出来ませんか

回答 ん？ そんなことをしてどうする。人生は有限です、コンピュータのメモリも有限。配列の大きさが分からないので、あらかじめ十分な上限を設定しておきたい、という意味であれば、動的に割り当てる方法を考えなさい。ネットで「`malloc`」を調べてもらなさい。

B.8 文法：関数

質問 15 `rand()` は乱数だと聞いたのですが、実行するといつも最初に生成されるのは 41 です。どうしてですか。

回答 VC2010 の `rand` 関数を呼ぶと、

$$x_n = 214013x_{n-1} + 2531011(\text{mod } 2^{31})$$

という漸化式を 1 回計算して、その結果の上位 15 ビットを `int` 型数として返します (<http://www001.upp.so-net.ne.jp/isaku/rand.html>)。初期値は $x_0 = 1$ と決められていますので、ビルドした直後に `rand` 関数を呼ぶと、 $x_1 = 2745024$ で、その上位 15 ビットを `int` 型数に直すと 41 になります。この「乱数」でゲームを作っても展開が読めてしまうのでおもしろくありません。 x_0 の値を変える関数が `srand` 関数です。同じプログラムで x_0 を何度も変えるのは意味がありませんので、メインプログラムの一番最初に 1 回だけ **`srand` 関数を呼ぶようにしなさい**。なるべく前に使った x_0 とかぶらないために、`time` 関数を利用したりします。

索引

!, 37
 !=, 35
 ", 29
 '"', 8
 <, 35
 <=, 34
 >, 35
 >=, 35
 (), 20, 29
 { }, 18, 36
 *, 20, 156
 **p, 165
 ==, 42
 +, 20
 ++, 76, 111
 +=, 42
 -, 20
 --, 78
 -=, 42
 .2, 22
 /, 20
 /* */, 18
 //, 18
 /=, 42
 :, 39
 ;, 7, 16, 29
 =, 20
 ==, 35
 &, 19, 29, 155
 &&, 37
 ||, 37
 %, 20
 %c, 122
 %d, 19
 %le, 103
 %lf, 19
 %10.2lf, 22
 %o, 122
 %p, 122
 %, 44
 %s, 226
 ¥n, 19
 →, 143

 "a", 223

 break, 39, 45

 case, 39

char 型, 223
 char 型配列, 223
 continue, 45

 default, 39
 (double), 20
 double, 19
 double *q, 158
 double 型数, 19, 102
 double 型数の内部表現, 102
 double 型数の表示, 19
 double 型変数の宣言, 19
 double 型ポインタ, 157, 158
 do while, 43
 do while(1), 42

 else, 34
 EOF, 227
 exp(), 28

 F5 キー, 7, 234
 fabs(), 28
 false, 42
 fclose(), 223
 FILE, 223
 FILE 型ポインタ, 223
 fopen(), 223
 for, 75
 for と while, 78
 fprintf(), 223
 fscanf(), 226

 if, 34
 include, 28, 29
 (int), 20
 int, 19
 int *p, 158
 int 型数, 19, 102
 int 型数の内部表現, 102
 int 型数の表示, 19
 int 型変数の宣言, 19
 int 型ポインタ, 157

 log(), 28

 math.h, 27

 pow(), 28
 printf(), 19

- "r", 226
- rand(), 47, 209, 237
- RAND_MAX, 48
- return, 144, 151

- scanf(), 19
- scanf, 236
- sqrt(), 27, 28
- srand(), 49
- stdio.h, 28
- stdlib.h, 28, 48
- studio.h, 29
- switch, 39
- system(), 28

- time(), 49
- time.h, 49

- unsigned int, 49
- unsigned int 型, 116, 216

- void 型, 151

- "w", 224
- while, 41
- while(1), 18, 42
- while 構文, 18, 40

- RSA 暗号, 117
- 新しいプロジェクト, 5
- アドレス, 146
- アドレス演算子, 155
- アドレス渡し, 147, 216, 226
- あまり, 20
- アルゴリズム, 53
- 暗号, 117
- アンダースコア, 143

- 一様乱数, 208, 217
- インクリメント演算子, 76
- インデント, 7

- エイトクィーン問題, 182
- エラーメッセージ, 17, 29
- エラトステネスの篩, 112
- 円周率, 207

- オーバーフロー, 102, 116
- 大きい数のべき乗, 114, 188
- オブジェクト指向, 2

- 改行, 19
- 階乗, 171
- 階層化, 54
- 階層化 (アルゴリズムの), 152
- 拡張子, 8
- 拡張ユークリッド互除法, 65
- 仮数部, 102
- かつ (and), 37

- 仮引数, 143
- 関係演算子, 35, 230
- 換字法暗号, 117
- 関数値, 144
- 関数と外注計算, 153
- 関数の定義プログラム, 143
- 関数ポインタ, 194
- 関数呼び出し, 144
- 間接参照演算子, 156
- 完全数, 72
- カンマ演算子, 77, 149

- 記憶できる桁数, 101
- 幾何平均, 28
- 基本統計量, 95
- キャスト, 20

- 偽, 42
- 擬似コード, 124
- 擬似乱数, 47
- 行列, 85

- クイックソート, 175

- グローバル変数, 165

- 計算精度, 103

- コードブロック, 36
- コイン投げ, 49
- 公開鍵暗号, 117
- 降順, 123
- 構造体, 223, 237
- 公約数, 61
- コメント, 17, 18
- 混合計算, 21
- コンパイル, 8

- 再帰, 169
- 再帰的アルゴリズム, 170
- 再帰的プログラム, 170
- 再帰呼び出し, 170
- 停止条件, 170
- サイコロの目, 49
- 最小公倍数, 62
- 最大公約数, 61
- サブルーチン, 150
- 算術演算子, 229
- 算法, 53

- シーザー暗号, 117
- シェーカーソート, 135
- シェルソート, 136
- 指数関数, 28
- 指数部, 102
- シミュレーション, 205
- 周期列, 217
- 出力バッファ, 222
- ショートカットキー, 11

ショートカットメニュー, 10

消去法, 201

昇順, 123

初期値の設定, 19, 83

書式文字列, 19

真, 42

信頼区間, 213

字下げ, 7, 234

実行結果のコピー, 12

実引数, 143

16 進数, 122

10 進法, 109

条件式, 36

条件分岐, 33

乗算合同法, 217

推定, 213

推定誤差, 213

推定精度, 213

数学関数ライブラリー, 27

スコープ, 165

スタージェスの公式, 106

スタートアッププロジェクト, 10, 234

スタートページ, 4

スタック, 171

ストップコード, 93

図形の面積, 206

制御変数, 76

セミコロン, 7

選択法ソート, 125

絶対値, 28

漸化式, 67

ソート, 123

素因数分解, 59, 118

相加平均, 28

相乗平均, 28

相對順位, 98

挿入法ソート, 129

添え字, 81

添え字付き変数, 146

添え字の範囲, 82

素数, 57, 112

素数定理, 58

ソリューション, 5

ソリューションエクスプローラー, 4

ソリューションに追加, 9

対数関数, 28

互いに素, 63

種 (seed), 49

代入演算子, 42, 229

代入文, 134

中心差分法, 198

調和平均, 28

直接入力モード, 6

手続き型, 2

データ型, 18, 230

データ入力, 93

データの入力, 19

データの表示, 19

ディクリメント演算子, 78

ディジタル (digital), 109

ではない (論理否定), 37

デバッグ, 8, 17, 29, 235

ブレークポイント, 88

デバッグのヒント

赤い波線, 29

エラーメッセージ, 29

無限ループ, 42

等幅フォント, 12

閉じるボタン, 17

トリム平均値, 106

度数分布, 99

DOS 画面, 7, 17

2 項係数, 68, 172

2 進小数, 122

2 進数, 109

2 進法, 109

2 次元配列, 85, 236

二重ループ, 86

二分法, 192

ニュートン法, 196

入力バッファ, 226

配列添え字演算子, 147

配列変数, 81

配列名, 81, 146

掃き出し法, 201

はずれ値, 99

8 進数, 110, 122

ハノイの塔, 179

バイト, 102, 146

バックトラック法, 184

バブルソート, 127

引数, 143

ヒストグラム, 100

百五減算, 65

表示桁数, 22

標準入出力, 28

標本調査, 207

標本標準偏差, 95

標本分散, 95

標本平均, 95

BMI, 32

ビット, 101, 109, 146
ビルド, 8, 234

p 進数, 110
ピタゴラス数, 72

ファイル出力, 222
ファイル名, 8, 223
ファンクションキー, 7
フィードバック, 18
フィボナッチ数列, 69, 188
フォーマット指定子, 19, 231
双子の素数, 58
不定方程式, 63
浮動小数点表示, 102
不偏分散, 95
フローチャート, 54

ブレイクポイント, 88, 236
分析 (分解) と統合, 54

プロジェクト, 5
プロトタイプ, 144, **152**
プロンプト, 21

平方根, 27, 28
ヘッダファイル, 28, 232
 `math.h`, 28
 `stdio.h`, 28
 `stdlib.h`, 28, 48
 `time.h`, 49
ヘロンの公式, 28
変数, 19
変数の入れ替え, 126, 157
変数の初期化, 96
変数の宣言, 19
変数の有効範囲, 165

べき乗, 28
ベクトル, 80

ペラン数, 72

ホーナーの方法, 111
方程式, 191

ポインタ, 157, 158
ポインタ (FILE 型), 223
ポインタのポインタ, 165
ポインタ配列, 164

マージ, 132
マクロ記号定数, 48, 163
マクロ定義, 163
または (or), 37

無限ループ, 42, 79

メモリ, 19

メモリ爆発, 171
メルセンヌ素数, 58

文字型配列, 122
`mod`, 114
戻り値, 144
モンテカルロ法, 205
モンテカルロシミュレーション, 205

約数, 91

ユークリッドの互除法, 61, 173
有効桁, 102

予約語, 19
縫り合わせ法マージ, 133

ラベル, 39
乱数, 47, 237
乱数の種, 49
ランダム置換, 139

ループ
 `do while`, 43
 `for`, 76
 `while`, 41

連立一次方程式, 199

ローカル変数, 165
60 進数, 26
論理演算子, 37, 230
論理記号, 37

ワード, 146

アルゴリズム

10 進数, 111
2 進数, 110
エイトクィーン問題, 184
エラトステネスのふるい, 112
拡張ユークリッド互除法, 64
クイックソート, 176
最大公約数, 61
最大値, 97
シェーカーソート, 135
シェルソート, 136
自然数の部分和, 67
順位, 98
選択法ソート, 125
素因数分解, 59
挿入法ソート, 130
総和計算 (再帰), 169
素数判定, 57
度数分布, 100
二分法, 192
二分法 (再帰), 193
ニュートン法, 196
ハノイの塔, 180

バブルソート, 128
平均値, 95
冪乗計算, 115
モンテカルロ法 (円周率), 208, 210
約数数え上げ, 56
ユークリッドの互除法, 61
ユークリッドの互除法 (再帰), 173
ランダム置換, 139

演習問題

randBetween, 168
RSA 暗号, 119
suica, pasmo, 51
16 進数, 122
2 項係数, 172
2 進小数, 122
2 次方程式の根, 51
8 進数, 122
うるう年, 38
階乗, 172
カレンダー, 51
簡易棒グラフ, 106
逆行列, 204
行列の積, 166
行列の積, 91
行列のべき乗, 168
行列のべき乗, 91
行列の和, 166
行列の和, 86
さいころ振り, 217
自動ヒストグラム, 106
数値積分, 220
素数判定, 91
チェックディジット, 32
駐車場の料金計算, 51
つり銭, 51
トランプを配る, 168
ビンゴ, 139
冪乗, 188
ボーリングのスコア, 91
BMI, 32
ラップタイム, 32
ランダム置換, 139
連立方程式を解く, 204

プログラム例

2 進数, 120
エイトクイーン (再帰), 187
エラトステネスのふるい, 120
大きい数のべき乗, 121
拡張ユークリッド互除法, 71
擬似乱数, 47, 216
クイックソート (再帰), 186
コイン投げ, 212
最大公約数, 71
最大値と最小値, 105
自然数の部分和, 67
選択法ソート, 137
素因数分解, 71
挿入法ソート, 138
素数判定, 70

多分岐 (switch), 38
データ入力, 104
度数分布, 105
二分法 (再帰), 202
ニュートン法, 202
バブルソート, 137
ヒストグラム, 27
ファイルからの入力, 225
ファイルへの出力, 222
平均と分散, 104
冪乗計算, 121
攪り合わせ法マージ, 133
挿入法マージ, 138
モンテカルロ法 (円周率の推定), 218
モンテカルロ法 (信頼区間), 219
約数, 70
ユークリッド互除法, 71
連立一次方程式, 203
石取りゲーム, 140
運勢判断, 73
数当てゲーム, 52
じゃんけんゲーム, 92
ヒットアンドブロー, 107
ペナルティキックゲーム, 189