基礎C言語［入門編］

コンピュータの基本から理解する
プログラミング

課題の解答と解説

第 1 章

課題 1

西暦 2000 年問題とは，1900 年代に，コンピュータが「年」を西暦の「19」を省略して下 2 桁で表現していたために生じた問題です。1999 年は，この表現では「99」になります。

ここで，1999 年から 2000 年になるときが問題でした。99 に 1 を加えて 100，その下 2 桁で「00 年」になるわけですが，「00 年」は，この表現方法では，1900 年を指します。つまり，**2000 年を迎えたときに，1900 年になったとみなしてコンピュータが処理をする恐れがある**ことでした。

この問題に対して，1990 年代末，全世界的に対策が行われました。そして，2000 年の幕開けは，情報技術に携わる人々にとって緊張の瞬間でした。結果，対策が功を奏してか，幸い大きなトラブルはありませんでした。

対策されなかった場合，重大なものではライフラインのストップ，金融の混乱などがあると予想されていました。原子力発電所，軍事施設で大きな問題が生じた場合は，全世界的な危機が訪れたとも言われています。

このように，日常に深く根付いたコンピュータが，不具合によって生活を危機に陥れる恐れがある，そんな世界に私たちは生きています。

課題 2

家電製品にも，現在は当たり前のようにコンピュータが組み込まれています。このようなコンピュータは，**組み込みコンピュータ**，**マイコン**などと呼ばれます。

洗濯機に組み込まれるマイコンは，モータの動きを制御したり，表示板があるものはそれを制御したりしています。高度なものでは，洗濯槽内に水温・水位・汚れセンサなどを備え，マイコンがそれを受け取って，洗濯のしかたを調整します。

課題 3

私たちは普段，「缶飲料を買う」と何気なく考え，それを行っています。こういったことを問題として認識し，定義することが，プログラミングにおいて必要になってきます。

「缶飲料を買う」は，どこまで明確に定義するかという程度の問題がありますが，たとえば，「自動販売機にお金を投入し，希望する缶飲料に対応するボタンを押して，出てきた商品を取り出す」ではいかがでしょうか。

課題 4

箇条書きで表してみましょう。

- 自動販売機に，欲しい商品が買えるのに十分なお金を投入する。
- 対応する商品のボタンを押す。
- 商品が出てきたら，それを取り出す。
- 釣り銭が出てきたら，それを取り出す。

でいかがでしょうか。もっと細かく考えれば，売り切れのとき，自動販売機が故障したとき，なども考慮することになるでしょう。

課題 5

前者の場合です。

- A を B にコピー：A は 3，B は 3
- B を A にコピー：A は 3，B は 3

これでは両方 3 になってしまい，確かに交換できていません。

一方，後者の場合は，

- A を X にコピー：A は 3，B は 5，X は 3
- B を A にコピー：A は 5，B は 5，X は 3
- X を B にコピー：A は 5，B は 3，X は 3

で，A は 5，B は 3 となり，確かに交換できています。もとの A の内容を，X に「逃がして」おくのがポイントです。

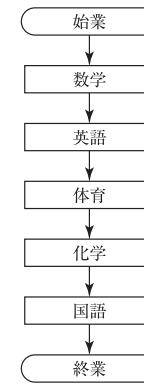
第 2 章

課題 6

たとえば，

- 1 時間目が数学
- 2 時間目が英語
- 3 時間目が体育
- 4 時間目が化学
- 5 時間目が国語

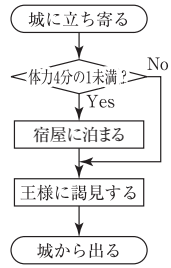
とすると，右図のような流れ図が書けるでしょう。本文 27 ページに掲げた注意事項を守って書けているか，確認してください。



課題 7

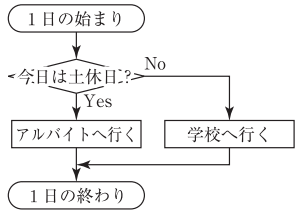
これは，右の図のような流れ図になるでしょう。「体力が最大の 4 分の 1 未満か」を条件として分岐しています。条件が成立したときのみ，「宿屋に泊まる」が実行されます。

判断の記号で分岐させる書き方，合流させるときの書き方に注意してください。



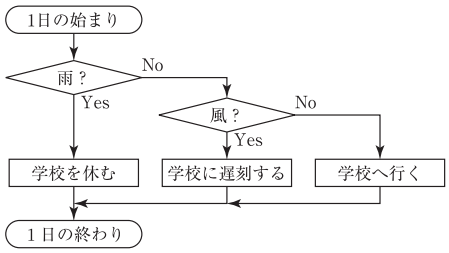
課題 8

流れ図は，基本的な 2 分岐の形式で，次のように書けるでしょう。



課題 9

本文 32 ページの流れ図で，最初に判断しているのは「風が吹いているかどうか」です。ですから，風が吹いていて，しかも雨が降っている場合は，最初の「風が吹いているかどうか」の判断で Yes となり，「学校に遅刻」します。これを，「お休み」にするには，「風が吹いているかどうか」「雨が降っているかどうか」の判断の順序を入れ替えて，下図のような流れ図を書けばよいのです。

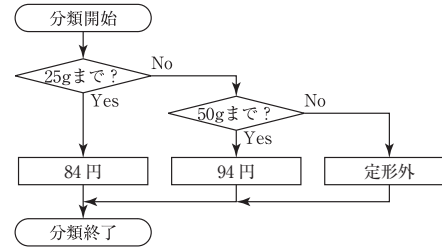


ここで理解していただきたいのは，**条件が複雑になってくると，それを判断する順序が重要になることがある**，ということです。

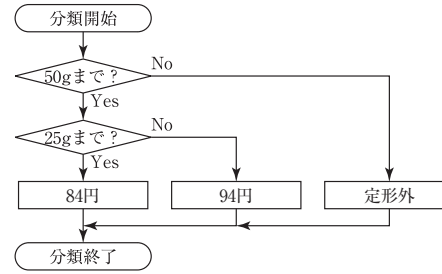
課題 10

まず，この分類手順で，3 分岐していることを見抜いてください。25 g までならば 84 円，50 g までならば 94 円，ですが，94 円になるケースは，「25 g 超 50 g まで」，さらに言い換えると，「25 g までではないが 50 g まで」であることです。ですから，まず 25 g までかどうかを調べ，さもなければ 50 g までかどうかを調べます。

ここで，「25 g まで」を「50 g まで」の前に判断している点が重要です。最初に「50 g までか」を調べると，これに 25 g までのケースも含まれますから，25 g までなのに「94 円」という分類結果が得られてしまいます。

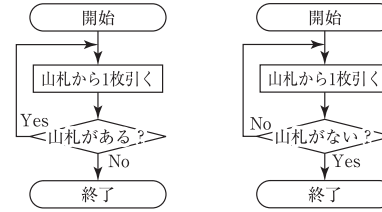


どうしても「50 g までか」を先に判断したければ，次のような流れ図を書けばよいのですが，プログラムとして表すことを考えると，先にあげたもののほうが一般的でしょう。



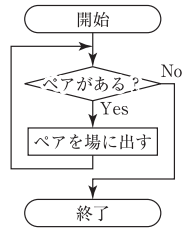
課題 11

本文 33･34 ページを参考に流れ図を作成しましょう。「条件が Yes のとき繰り返し」「条件が No のとき繰り返し」の流れ図は，それぞれ次のようになります。



繰り返しを含む手順を考えるとき，その繰り返しは条件が成り立っている**間**（Yes の間）繰り返されるのか，条件が成り立つ**まで**（Yes になるまで〈No の間〉）繰り返すのか，意識しておく必要があります。それを，流れ図あるいはコードに適切に反映させなければなりません。

課題 12



作成する流れ図では，「配られた手札にいきなりペアがない」場合もありますから，まずペアがあるか判断する前判定型の繰り返しを使用します。ペアを場に出しきったら，「ペアがある？」の判断が「No」になりますから，そこで処理は終了します。

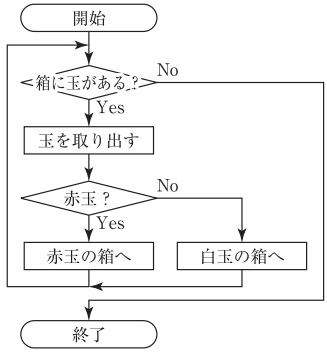
なお，問題文では手札がなくなってしまう場合は考えなくてもよいとありますが，手札がなくなれば，ペアがあることはありえず，「ペアがある？」の判断は No になって処理が終了します。ですから，実はこの流れ図は「ペアを場に出し尽くして手札がなくなる」状況も考慮されています。

課題 13

この処理は，「箱から玉をなくなるまで取り出す」という繰り返しと，「取り出した玉の色で入れる箱を分ける」分岐が含まれています。

箱の中の玉は 0 個の場合もありますから，前判定型の繰り返しで取り出します。取り出した玉について，赤玉かを判断し（白玉かを判定しても同様です），それぞれの箱に入れます。

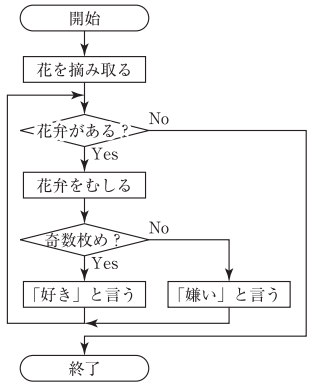
これらから，次のような流れ図が作成できればよいでしょう。



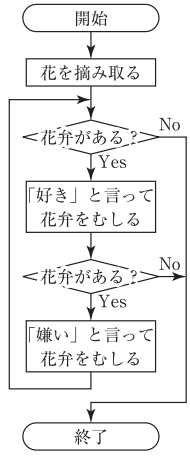
課題 14

「花弁をむしる」とき，奇数枚目が「好き」，偶数枚目が「嫌い」になります。ですから，花弁があるかどうかを調べ，奇数枚目だったら「好き」と言ってむしり，偶数枚目だったら「嫌い」と言ってむしればよいわけです。

ここで，直接には現れていませんが，花弁をむしる人はむしった花弁が奇数枚目か偶数枚目かを覚えていなければなりません。プログラムをより指向して考えるならば，これは **2.7**（本文 40 ページ）以降で述べる「変数」として扱う必要があります。



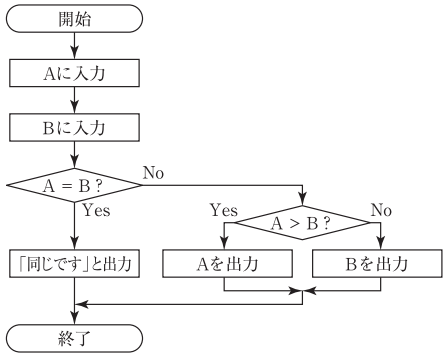
もう 1 つの考え方も示しておきましょう。この「好き」「嫌い」と言って花弁をむしる操作を，2 回を 1 組として考えます。すると，右図のような流れ図ができあがります。これは，継続するかの判定が 2 箇所にある，変則的な繰り返しです。このような繰り返しも，理論的には前判定型のみに書き換えることができます。



課題 15

A と B に同じ値が入力された場合を別扱いすることで，全体として 3 分岐になることに注意してください。

最初に A と B が等しいかどうか判断するならば，次のような流れ図に書き換えられます。



上記のほかにも，

- まず「A > B」を判断（Yes ならば A を出力）
- さもなければ「A < B」を判断（Yes ならば B を出力）
- さもなければ A = B（「同じです」と出力）

のような書き方もできるでしょう。

課題 16

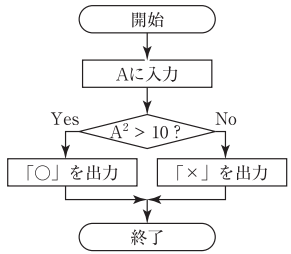
入力した数を覚えておくのに変数が必要となることに気付いてください。たとえば A としましょう。これに入力した数の 2 乗を計算して分岐します。

さて，ここで注意しなければならないのは，「10 を超える」という部分です。このとき，**10 は含みません**（不等号は「>」になります）。

- **「未満」「超える」はその数は含まない**
- **「以上」「以下」はその数を含む**

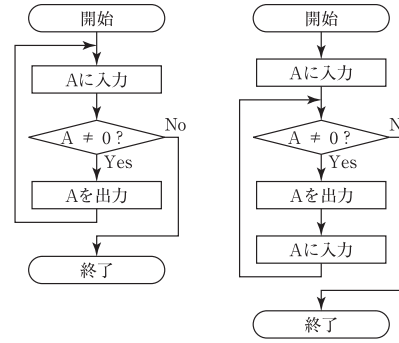
ことに注意してください。

なお，**2.9**（本文 44 ページ）で扱う，変数への代入を使えば，A に値を入力した後，別の変数（たとえば B）に A の 2 乗を計算して代入しておき，その（B の）値が 10 を超えるかどうかで分岐する流れ図も書けます。



課題 17

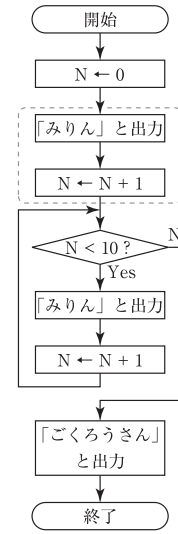
本文 43 ページの流れ図は，入力された値（A）を直ちに出力しています。0 が入力されたときにそれを出力させずに終了するには，出力する前に A の値を調べ，0 だったら終了してしまえばよいのです。ですから，次の左図のように書き換えればよいでしょう。



これを，問題の指示に従って前判定型繰り返しで書き直したものが，上の右図です。**2.6.1**（本文 38 ページ）で述べた方法に従って書き直しています。継続判定の前にある「A に入力」を繰り返しの外に出し，繰り返す処理の末尾に追加します。

課題 18

2.9.2（本文 45 ページ）の流れ図は，後判定型の繰り返しを用いています。これは，**2.5.2**（本文 36 ページ）で述べたように，1 回分をまず実行するという方法で，図のように前判定型に書き換えればよいのです。

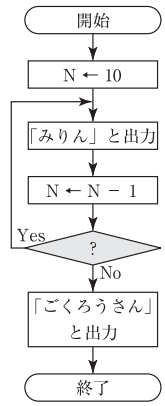


さて，この流れ図は，前判定型に書き換えたときの「まず実行する 1 回分」にあたる破線部は，不要です。「まず 1 回実行しなければ継続の判断ができない」というものではないからです。

ですから，右図の破線部がない流れ図でも正解です。むしろ，本文の流れ図は 10 回繰り返すことがわかっているから（0 回ということがないから）後判定型で書けるのであって，より一般的には，（0 回もありえますから）この，上図の破線部がない形式の流れ図のほうがよいということになります。

課題 19

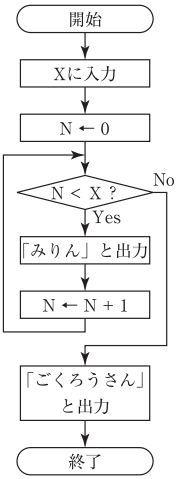
Nを10からカウントダウンさせてみましょう。カウントダウンですから，「N ← N + 1」は「N ← N − 1」に書き換わります。さて，このとき，継続条件をどう定めると10回繰り返せるか，が考えどころです。



流れ図は右の図のようになります。「？」で示された判断の記号に入るとき，Nは，1回目は9，2回目は8，……，10回目は0です。10回実行したら終了するのですから，「Nが0になったらNoになる条件」が入ります。これは，2通り考えることができて，「N > 0」または「N ≧ 1」です。このいずれも，Nが減らされていき，0になったら成立しなくなります。

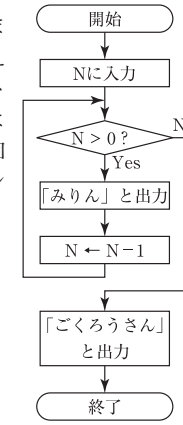
課題 20

問題の条件「入力した回数が0または負の場合，1回も表示させない」より，この流れ図では繰り返しが1回も実行されない可能性があります。こういった場合は，前判定型の繰り返しを使わなければなりません。ですから，**課題 18** で書き換えた流れ図に基づいて考える必要があります。



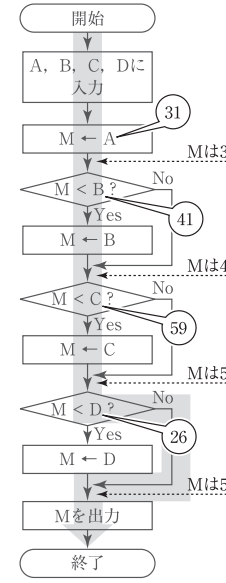
入力した回数は，たとえばXという変数に記憶します。Nを0から始めて，X未満の間，カウントアップしながら繰り返せばよいでしょう（本文の流れ図では，10未満の間繰り返して10回実行していることから類推してください）。

一方で，**課題 19** のようにカウントダウン式で繰り返すこともできます。終了時に，繰り返す回数を覚えておく必要がないならば，カウントアップの場合に必要だったXのような変数が不要になり，繰り返す回数をNに入力し，そこからカウントダウンしていくことになります。



課題 21

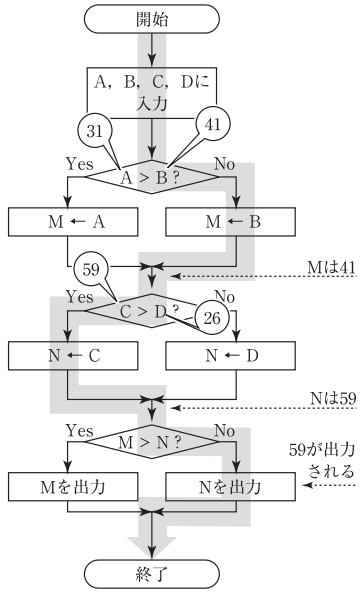
まず，**2.10.1**（本文46ページ）の流れ図で考えてみましょう。



最初に，MにAの値31が代入されます。判断「M < B ？」において，M（31）よりB（41）のほうが大きいですから，「Yes」に分岐してMは41になります。次の判断「M < C ？」は，M（41）よりC（59）のほうが大きく，「Yes」に分岐してMは59になります。次の判断「M < D ？」は，M（59）よりD（26）のほうが大きいのではありませんから，「No」に分岐してMの値はそのままです。かくして，最後に確かにA，B，C，Dの値の最大値59が出力されています。

次に，**2.10.2**（本文47ページ）の流れ図を考えましょう。

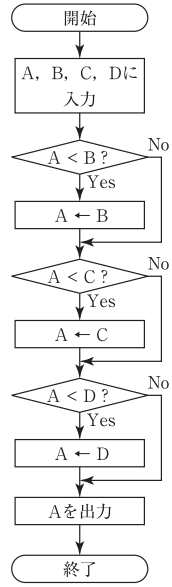
最初の判断「A > B ？」について，A（31）とB（41）ではAのほうが大きいのではありませんから，「No」に分岐してMは41になります。次に，「C > D ？」は，C（59）とD（26）ではCのほうが大きく，「Yes」に分岐してNは59になります。最後の判断「M > N ？」は，Mが41，Nが59で「No」に分岐して，Nの値すなわち59が出力されます。確かにこの方法でも，A，B，C，Dの値の最大値59が出力されています。



課題 22

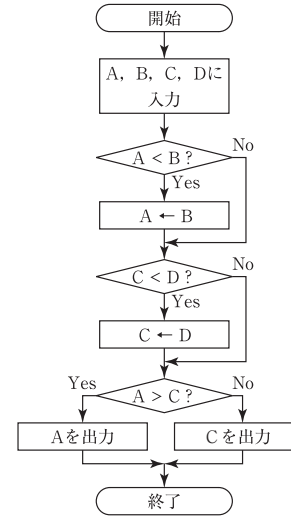
2.10.1（本文 46 ページ）の流れ図では，入力した数を記憶する変数 A，B，C，D のほかに，最大値を記憶する変数 M を使っています。**2.10.1** の流れ図では，終了時に A，B，C，D の値は入力されたときのままです。

A の値が破壊されることと引き換えに，右に示すような，M を使わない流れ図に書き換えることができます。



課題 23

2.10.2（本文 47 ページ）の流れ図も，A，C の値が破壊されることと引き換えに，M，N を使わない流れ図に書き換えられます。



課題 24

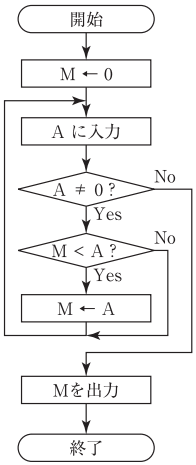
この処理では，「勝ち抜き式」で最大値を求める方法が定石です。ですから，

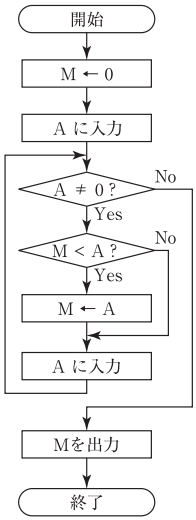
- 正の数を入力する
- 入力した数が暫定最大値を超えていたら，それを更新する

を繰り返します。この繰り返しは，入力した数が 0 になるまで(0 でない間)行われます。入力した値が 0 かどうかは，入力後にチェックすればよいでしょう。

一方，暫定最大値について考えます。この問題では正の数が入力されることがわかっていますから，0 で初期化しておけばよいでしょう。なぜなら，初回の正の値の入力は，0 より大きく，暫定最大値となるからです。

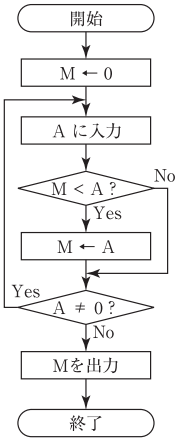
以上から，流れ図は次の左図のように書けます。さらに，これを前判定型の繰り返しを用いて書き換えると，次の右図のようになります。流れ図中で，入力した値を記憶する変数を A，暫定最大値を記憶する変数を M としています。





ここで，いずれの流れ図でも，初回に 0 を入力すると，ただちに M が出力されますが，その値は 0 であり，問題の条件を満たしています。

なお，この問題に限っては，後判定型の繰り返しを用いて，右図のような流れ図を書いてもかまいません。本来ならば，A に入力した直後，その値をチェックする必要がありますが，A が 0 ならば暫定最大値の更新は起こらない（「M < A ?」の判断で必ず「No」になる）ことがわかっていますから，その後に A が 0 かどうかをチェックしてもかまわないのです。この流れ図でも，初回に 0 を入力したら，「0」が出力されることを確認してください。



課題 25

それぞれの場合について，考えてみましょう。それぞれの表で，繰り返しが進むと右列へ進んでいきます。

Nの初期値：0	Mが0	Mが1	Mが2
N≦M ?	Yes No	Yes Yes No	Yes Yes Yes No
Nは偶数 ?	Yes	Yes No	Yes No Yes
S←S+N（Sの値）	0	0	0 2
N←N+1（Nの値）	1	1 2	1 2 3
出力されるSの値	0	0	2

Nの初期値：1	Mが0	Mが1	Mが2
N≦M ?	No	Yes No	Yes Yes No
Nは偶数 ?		No	No Yes
S←S+N（Sの値）			2
N←N+1（Nの値）		2	2 3
出力されるSの値	0	0	2

Nの初期値：2	Mが0	Mが1	Mが2
N≦M ?	No	No	Yes No
Nは偶数 ?			Yes
S←S+N（Sの値）			2
N←N+1（Nの値）			3
出力されるSの値	0	0	2

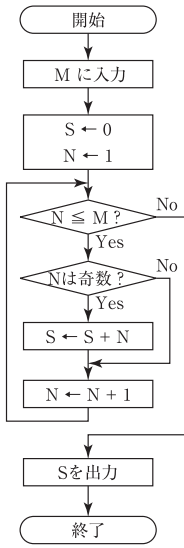
いずれの場合も，

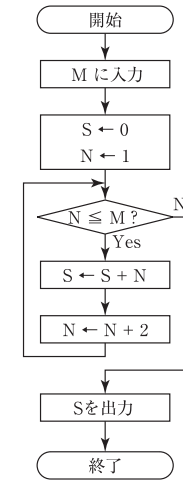
- M が 0 のとき S は 0（0 以下の偶数の和は 0）
- M が 1 のとき S は 0（1 以下の偶数の和は 0）
- M が 2 のとき S は 2（2 以下の偶数の和は 2）

になり，正しい結果となっていることがわかります。

課題 26

2.11.1（本文 49 ページ）の方針で奇数の和を求めるならば，奇数かどうかを判断して，奇数ならば足す，ということになるでしょう。ですから，本文 49 ページの流れ図（**図 2-23**）の，**「N は偶数？」の部分を，「N は奇数？」に書き換えればよい**ことになります（書き換えた流れ図は下の左図）。ただし，このときの N の初期値は 0 または 1 でなければなりません（N を 2 で初期化すると，最初の奇数「1」を加え損ねてしまいます）。





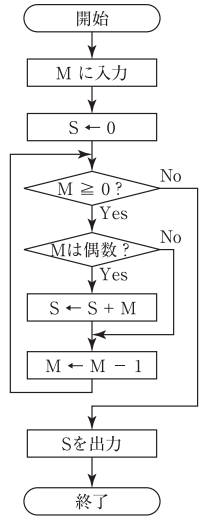
2.11.2（本文 50 ページ）の方針ならば，加えるべき数を 1，3，5，……，と増やしていけばよいことになります。本文 50 ページの流れ図（**図 2-24**）では，N を 0 から始めて，0，2，4，……，と加えていますから，奇数を加えるには，N を 1 から始めて，1，3，5，……，と加えればよいわけです。ですから，**N を 0 でなく，1 で初期化する**ように書き換えればよいでしょう。書き換えた流れ図を，前ページの右図に示しておきます。

課題 27

課題 19（本文 45 ページ）を参考にしましょう。入力された M を 1 ずつ減らしながら，M が偶数のときに加えていきます。

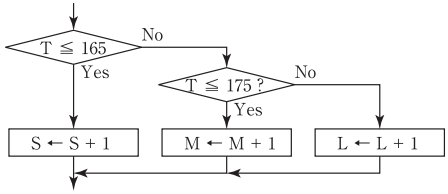
ここで問題になるのは，繰り返しの継続条件です。最小の偶数（2）が加えられたら終了すればよいのです。ですから，「M ≧ 2 ？」などが条件として考えられるでしょう。

書き換えた流れ図は右に示します。ここでは繰り返しの継続条件が「M ≧ 0 ？」になっていますが，**課題 25** と同様に考察すれば，これでも正しいことがわかります（「M ≧ 1 ？」でもかまいません）。



課題 28

分岐する部分の流れ図は，以下のようになります。



「小さいほうから」分岐するので，「T が 165 **以下**かどうか」さもなければ「T が 175 **以下**かどうか」で分岐します。S サイズは身長 165 cm 以下ですから（未満ではない），これを注意する必要があります。

課題 29

長い文章を書くときの「トップダウン式の書き方」は，文章の構成を考えながら，最初に章割りをし，章を節で割り，……，をして，節や項目（もっと細かい場合は，段落・文）で書くべきことを定めてから文章として完成させる方法です。

一方，「ボトムアップ式の書き方」では，主題に沿ったキーワード・文を作り，それらを並べ替えたりして最終的に長い文章として完成させます。

これらを，プログラムにおけるトップダウン・ボトムアップと対比させて考えてみましょう。

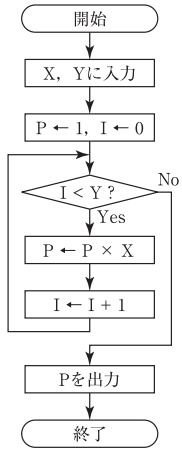
課題 30

まず，「X の Y 乗」について，Y を変えて確認してみましょう。

- X の 0 乗：1
- X の 1 乗：X
- X の 2 乗：X × X
- X の 3 乗：X × X × X
- ……

ここで，X を Y 回かけあわせたものが「X の Y 乗」です。X の 0 乗も考慮すると，「1 に X を Y 回かけたもの」が「X の Y 乗」ということになります（X の 1 乗は 1 × X，X の 2 乗は 1 × X × X，……，と考えるのです）。

ですから，流れ図は，「Y 回かける」を繰り返しで実現します。P に「X の Y 乗」を計算し，I で何回かけたかをカウントするものとすると，右図のようになります。P を 1 で初期化している点に注意してください。右図では I を 0 で初期化していますが，I を 1 で初期化した場合は，繰り返しの継続判定は「I ≦ Y ？」になります。



ここで，Y に 0 が入力されたときは，繰り返しが 1 度も実行されず，P の初期値 1 が出力される（「0 乗」の正しい結果です）ことも確認してください。

なお，**課題 19・27** と同様に考えて，入力した Y をカウントダウンして回数を数えれば，I は不要になります。

課題 31

問題を確認しましょう。これはつまり，

- X の 0 乗：1
- X の 1 乗：X
- X の 2 乗：X^2
- X の 3 乗：X × X^2
- X の 4 乗：X^2 × X^2

• ……

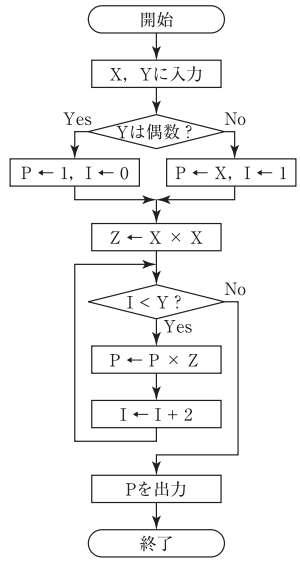
と計算する，ということです。

課題 30 と比較すると，

- X でなく，X^2 をかけあわせる
- Y が偶数のときは 1 に，奇数のときは X に（X^2 を）かけていく

という違いがあると考察できます。ですから，**課題 30** の流れ図に基づいて，

- Y の偶数・奇数で分岐して P の初期値を定める
- X^2 を計算しておき，繰り返しではこれをかける（かける回数には注意する）



と変更すればよいことになります。これらから流れ図を作成すると，上の図のようになるでしょう。

ここで注意したいのは繰り返しの回数です。この流れ図では，I で X を何回かけたかを記憶しています。Y が偶数のときは 0，奇数のときは 1 で初期化しておきます。そして，1 回繰り返されるたび I を 2 増やしています。これで正しい結果が得られることを確認してください。

課題 32

課題 31 の流れ図では，繰り返しに入る前に 1 回の判断と 1 回のかけ算が行われることに注意します。

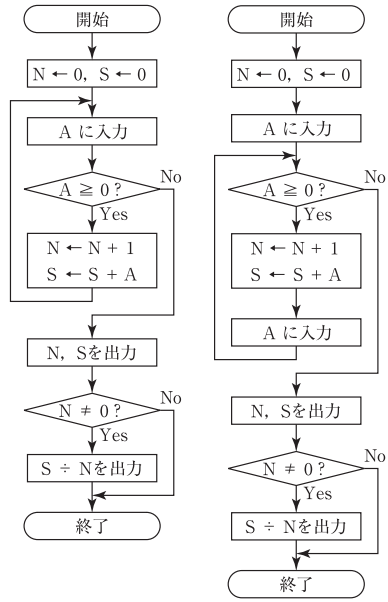
判断と乗算の回数をまとめたものが下の表です。**課題 31** の方法は，Y が 0，1，2 程度では **課題 30** の方法に比べて有利ではないですが，Y が大きくなるほど有利になってきます。言い換えると，**課題 31** の方法は，最初に「余計な」処理をすることと引き換えに，Y が大きいときに効率的に処理できます。

Yの値	課題30 判断	課題30 乗算	課題31 判断	課題31 乗算
0	1	0	2	1
1	2	1	2	1
2	3	2	3	2
3	4	3	3	2
4	5	4	4	3
5	6	5	4	3
6	7	6	5	4
7	8	7	5	4
8	9	8	6	5
9	10	9	6	5
10	11	10	7	6

課題 33

この流れ図では，「負の値が入力されるまで（0 以上の値が入力されている間）」繰り返すことになります。その中で，データ数，合計を計算していきます。平均は，最後に，合計をデータ数で割って求めます。

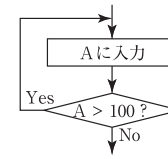
これより，次のような流れ図が書けるでしょう。単純に考えれば左の流れ図のようになるでしょうが，繰り返しを前判定型に書き換えると右のようになります。



データ数を N，合計を S に計算しています。いずれも，更新は「足し込む」ことで行われるので，0 での初期化が必要です。平均は，データ数が 0 だと「0 での割り算」をしてしまいますから，N が 0 でないかどうかで分岐して，平均を求めています。

課題 34

0～100 の入力しか受け付けないようにするには，0～100 以外が入力されたときに（繰り返しで）入力しなおしとすればよいのです。ここで，「0～100 以外」というのは「0 未満か，100 を超えるとき」ですが，このプログラムでは 0 未満（負）の入力は入力の終了を表しますから，入力しなおしとする条件は「100 を超えるとき」です。この判定は，入力した後で行わないと意味がないですから，後判定型繰り返しで書くのが都合



がよいでしょう。しかして，**課題 33** の流れ図の「A に入力」の部分（右の流れ図では 2 か所）を，前ページに示すように書き換えればよいということになります。

課題 35

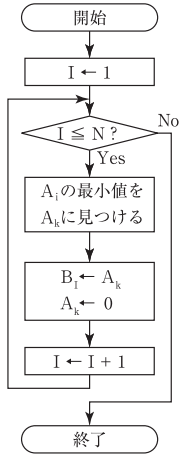
問題の指示どおりに，大ざっぱな流れ図を作るところから始めましょう。

この手順は，

- A_1〜A_N から最小値を見つけ（A_k），B_1 に代入する
- A_1〜A_N から最小値を見つけ（A_k），B_2 に代入する
- ……

を B_N まで繰り返すことになります。毎回 A_k には 0 が代入され，最小値を探すときにはこの 0 は無視します。

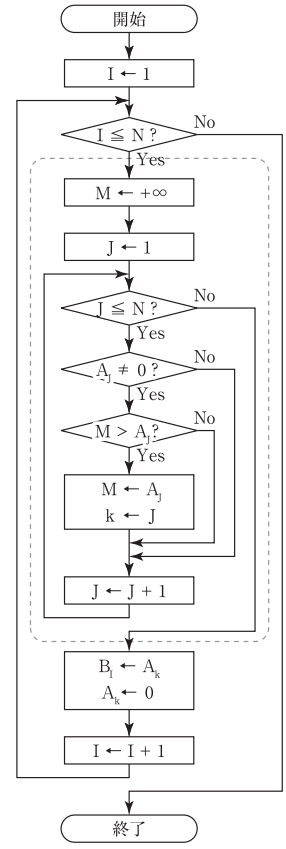
ここまでで，流れ図は大ざっぱに次の図のように書けるはずです。



次に，「A_i の最小値を A_k に見つける」の部分を考えます。これは，定法である「勝ち抜き式」を使えばよいでしょう。ただし，0 が記憶されている変数はスキップします。

手順としては，暫定最小値を M に記憶して，最小値がある A の中の番号 k とその最小値を覚えればよいのです。ここで，M の初期値は「絶対に最小値になり得ない値」として，正の無限大としておけばよいでしょう（実際のプログラムでは，「十分大きな値」とします）。

以上から，「A_i の最小値を A_k に見つける」の部分の流れ図が作れて，次のように流れ図が完成します。破線で囲んだ部分が，「A_i の最小値を A_k に見つける」部分です。



なお，この手順は，本質的には **7.3**（本文 168 ページ）で扱う「選択ソート」と呼ばれる並べ替えのアルゴリズムです。**7.3** を学習した後，本問を見返してみてください。

第 3 章

課題 36

`printf()` 関数内の，「`"`」でくくられた文字列を変更すれば，出力される文字列も変更されることが確認できるでしょう。

ここで，「`"`」そのものが出力できない（コンパイルエラーになる）ことに気付くかもしれません。この場合，「`"`」の前に円記号を付けて，「`¥"`」としてください。これが，出力時に「`"`」に置き換わります。

課題 37

printf() 関数を増やせば，何回かに分けて文字列を出力できます。ここで，「¥n」がないと，改行されずに出力されます。

なお，「¥n」は文字列の最後にある必要はなく，1つの文字列を出力させている途中で改行，ということもできます。

課題 38

それぞれ確認しましょう。

- x：変数名として使用できます。
- money$：「$」は変数名に使用できない記号です。ですから，**変数名として使用できません。**
- player1：変数名として使用できます。
- cardNumber：変数名として使用できます。
- int__：C 言語にとって特別な意味を持つ「int」が含まれていますが，「int」そのものではありません。また，下線「_」は変数に使用してよい記号です。ですからこれは，変数名として使用できます。
- 500yen：英字と数字からなってはいますが，数字から始まっているので，**変数名として使用できません。**
- myTV：変数名として使用できます。

課題 39

この課題を実行しようとしてコンパイルすると，警告が出るかもしれません。しかし無視して実行してみてください。

出力される値は，環境によってさまざまです。ここに出力例を書くことはできません。しかし少なくとも，0 に初期化されているなどという，都合の良いことにはなっていないことが多いでしょう。

課題 40

プログラム全体を，次のように書き換えてください。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a;
5:
6:    a = 3;
7:    printf("%d¥n", a);
8:    a = 5;
9:    printf("%d¥n", a);
10: }
```

ここで，8 行で a に 5 を代入することで，それまでの a の値 3 は失われます。

課題 41

7 行を次のように書き換えてください。

```
7:    a = a * a * a;
```

C 言語に累乗の演算子はありませんから，「a の 3 乗」を計算するには，a を 3 回かけなければなりません。

課題 42

プログラムは，次のようになります。演算子に注意してください。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, b, c, d, e, x, y;
5:
6:    x = 8;
7:    y = 5;
8:    a = x + y;
9:    b = x - y;
10:   c = x * y;
11:   d = x / y;
12:   e = x % y;
13:   printf("%d¥n", a);
14:   printf("%d¥n", b);
15:   printf("%d¥n", c);
16:   printf("%d¥n", d);
17:   printf("%d¥n", e);
18: }
```

結果については，x / y に注意しましょう，「整数割る整数」で 1 が出力されているはずです。

課題 43

①は，a + 5 です。

②は，-b - 6 です。

③は，x * y * z です。かけ算の記号が省略されていること，C 言語の乗算の演算子は「*」であることに注意してください。

④は，(a + b) / c です。除算の演算子は「/」です。

⑤は，x / (y * z) です。x / y * z ではありません。

⑥は，a * a * b * b * b です。C 言語に累乗の演算子はありませんから，乗算で書き表しましょう。

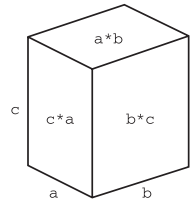
⑦は，b * b - 4 * a * c です。累乗と，計算の優先

順位（加減算より乗除算が先）に注意しましょう。

⑧は，a * ((a + b) / 2 + a * b) です。かっこは () しか使用しない点に注意してください。

課題 44

直方体の体積は，「縦×横×高さ」ですから，a * b * c です。表面積は，「縦×横」の面，「横×高さ」の面，「高さ×縦」の面が 2 面ずつあります。ですから，2 * (a * b + b * c + c * a) となるでしょう。



これから，体積を v，表面積を s に計算するとして，次のようにプログラムを作成できます。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a = 2, b = 3, c = 5;
5:    int v, s;
6:
7:    v = a * b * c;
8:    s = 2 * (a * b + b * c + c * a);
9:    printf("%d¥n", v);
10:   printf("%d¥n", s);
11: }
```

なお，printf() 関数で複雑な式の値を出力する方法（本文 76 ページ）を使えば，変数 v，s は不要で，体積・表面積の値を printf() 関数のかっこ内で直接計算できます。

課題 45

scanf() 関数の使い方に注意してください。読み込む変数名の前に付ける「&」を忘れないようにしましょう。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, b;
5:
6:    scanf("%d", &a);
7:    scanf("%d", &b);
8:    printf("%d¥n", a + b);
9:    printf("%d¥n", a - b);
10:   printf("%d¥n", a * b);
11:   printf("%d¥n", a / b);
12: }
```

課題 46

課題 44 のプログラムを，次のように書き換えてください。変数 a，b，c の初期化は，scanf() 関数で読み込まれるのですから，不要です。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, b, c;
5:    int s, v;
6:
7:    scanf("%d", &a);
8:    scanf("%d", &b);
9:    scanf("%d", &c);
10:   v = a * b * c;
11:   s = 2 * (a * b + b * c + c * a);
12:   printf("%d¥n", v);
13:   printf("%d¥n", s);
14: }
```

第 4 章

課題 47

① a は 2 ではありませんから，成立しています。

② a は b より大きくはないですから，成立していません。

③ a + 2 は 5 で，b の値と等しくなります。したがって，成立しています。

④ a / b は整数どうしの割り算ですから，0 になります。したがって，成立しています。

⑤ a - b は −2 で，0 以上ではありません。したがって，成立していません。

⑥ (a + b) % 2 は a + b（8）を 2 で割った余りで，0 です。したがって，成立しています。ここで，x が y の倍数かは（x が y で割り切れるか）は x % y == 0 を調べればよいことを知っておきましょう。

課題 48

次のようなプログラムになります。入力された 2 つの整数が等しいときには，いずれを出力しても同じであることに留意してください。

```
1:  #include <stdio.h>
2:
```

```
3:  main() {
4:    int a, b;
5:
6:    scanf("%d", &a);
7:    scanf("%d", &b);
8:    if (a > b) {
9:      printf("%d¥n", a);
10:   } else {
11:     printf("%d¥n", b);
12:   }
13: }
```

これは，**2.8.1**（本文 42 ページ）の流れ図を C 言語プログラムで書いたものです。そしてこれは，**1.5.4**（本文 13 ページ）に掲げたものと，等価なプログラムです。

課題 49

偶数か奇数かは，「2 で割り切れるかどうか」です。ですから，剰余の演算子を使います。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a;
5:
6:    scanf("%d", &a);
7:    if (a % 2 == 0) {
8:      printf("偶数です。¥n");
9:    } else {
10:     printf("奇数です。¥n");
11:   }
12: }
```

課題 50

実係数の 2 次方程式 $ax^2+bx+c=0$ の解は，判別式 $D=b^2-4ac$ の符号によって判別されます。D が 0 以上ならば実数解を持ち，0 未満ならば実数解を持ちません。これより判別の手順を整理すると，次のようになります。

- a が 0 ならば，2 次方程式ではない
- そうでなくて，D が 0 以上ならば実数解を持つ
- そうでないときは，実数解を持たない

これは，次のようにプログラムできます。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, b, c;
5:
6:    scanf("%d%d%d", &a, &b, &c);
7:    if (a != 0) {
8:      if (b * b - 4 * a * c >= 0) {
9:        printf("実数解を持つ¥n");
10:     } else {
11:       printf("実数解を持たない¥n");
12:     }
13:   } else {
14:     printf("2 次方程式ではありません¥n");
15:   }
16: }
```

4.3（本文 88 ページ）で扱う `if …… else if ……` を使えば，次のようにも書けるでしょう。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, b, c;
5:
6:    scanf("%d%d%d", &a, &b, &c);
7:    if (a == 0) {
8:      printf("2 次方程式ではありません¥n");
9:    } else if (b * b - 4 * a * c >= 0) {
10:     printf("実数解を持つ¥n");
11:   } else {
12:     printf("実数解を持たない¥n");
13:   }
14: }
```

課題 51

4 つの数の最大値を求めるアルゴリズムは，**2.10**（本文 46 ページ）で扱いました。**2.10** の流れ図を，C 言語プログラムで表せばよいでしょう。

「勝ち抜き式」であれば，次のようなプログラムになります。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, b, c, d, m;
5:
6:    scanf("%d%d%d%d", &a, &b, &c, &d);
7:    m = a;
```

```
    if (m < b) {
      m = b;
    }
    if (m < c) {
      m = c;
    }
    if (m < d) {
      m = d;
    }
    printf("最大値は%d¥n", m);
}
```

また,「トーナメント式」であれば,次のようになります。

```
#include <stdio.h>

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

  scanf("%d%d%d%d", &a, &b, &c, &d);
  if (a > b) {
    m = a;
  } else {
    m = b;
  }
  if (c > d) {
    n = c;
  } else {
    n = d;
  }
  if (m > n) {
    printf("最大値は%d¥n", m);
  } else {
    printf("最大値は%d¥n", n);
  }
}
```

課題 52

条件を整理しましょう。

- a に負の値が入力されたら，対応していないと出力
- そうでなくて，b に負の値が入力されたら，対応していないと出力
- そうでなくて，b が 0 のときは，0 で割り算できないと出力
- そうでないときは，$a \div b$ の商と余りを出力

これを，if …… else if ……の形式で表すことに注意して，次のようにプログラムが書けます。

```
#include <stdio.h>

main() {
  int a, b;

  scanf("%d%d", &a, &b);
  if (a < 0) {
    printf("負の整数には対応していません¥n");
  } else if (b < 0) {
    printf("負の整数には対応していません¥n");
  } else if (b == 0) {
    printf("0で割り算はできません¥n");
  } else {
    printf("商：%d¥n", a / b);
    printf("余り：%d¥n", a % b);
  }
}
```

課題 53

9 行を，「printf("抽せんしました。福引き券はあと%d 枚です¥n", a);」と書き換えて実行させればよいでしょう。

1 回実行されるごとに福引き券は 1 枚ずつ減っていき，「あと 0 枚」が出力されると終了するはずです。

課題 54

入力された正の整数が 2 で何回割れるかを調べることになります。これは繰り返しで実現しますが，どんなときに繰り返しが終了するかが問題です。これは，2 で割り切れなくなったとき，すなわち，奇数になったときです。ですから，繰り返しの継続条件は「偶数である間」です。

プログラムは，n に正の整数を入力して，それが 2 で割り切れる間（2 で割った余りが 0 である間），n を 2 で割り，割り切れた回数をカウントする変数 k を 1 増やしています。

```
#include <stdio.h>

main() {
  int n, k = 0;

  scanf("%d", &n);
  while (n % 2 == 0) {
    n = n / 2;
```

```
9:      k = k + 1;
10:   }
11:   printf("k = %d¥n", k);
12: }
```

課題 55

流れ図を素直にコーディングすると，次のようになります。変数は a と n が必要です。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, n;
5:
6:    n = 0;
7:    do {
8:      scanf("%d", &a);
9:      n = n + 1;
10:   } while (a != 0);
11:   printf("%d¥n", n);
12: }
```

「最後の 0 をカウントしない」については，最後は必ず 0 で，それをカウントして繰り返しが終了しているのですから，11 行を n - 1 を出力するように書き換えるのが簡単でしょう。

課題 56

合計が 100 以下の間（「100 を超えたら」は 100 を含まないので，その反対の条件は 100 を含む「100 以下」です）繰り返すことになります。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, s = 0;
5:
6:    while (s <= 100) {
7:      scanf("%d", &a);
8:      s = s + a;
9:    }
10:   printf("%d¥n", s);
11: }
```

このプログラムについては，最初は合計は 0 で，必ず 1 回は入力されなければいけませんから，do……while ループを使っても書けます。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, s = 0;
5:
6:    do {
7:      scanf("%d", &a);
8:      s = s + a;
9:    } while (s <= 100);
10:   printf("%d¥n", s);
11: }
```

課題 57

求められているプログラムの流れ図は，**課題 17**（本文 43 ページ，解答は 5 ページ）で作成しています。これにのっとってコーディングすると，次のようになります。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a;
5:
6:    scanf("%d", &a);
7:    while (a != 0) {
8:      printf("%d¥n", a);
9:      scanf("%d", &a);
10:   }
11: }
```

課題 58

求められているプログラムの流れ図は，**課題 24**（本文 48 ページ，解答は 7 ページ）で作成しています。ここでは，前判定型繰り返し（while ループ）を使った流れ図でコーディングしてみます。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, m;
5:
6:    m = 0;
7:    scanf("%d", &a);
8:    while (a != 0) {
9:      if (m < a) {
10:       m = a;
11:     }
```

```
12:     scanf("%d", &a);
13:   }
14:   printf("%d¥n", m);
15: }
```

課題 59

① a = a / 4 と書き換えられます。

②基本に忠実に書き換えれば，a = a + (b * 2) です。ここで，乗算のほうが加算より優先順位は高いですから，a = a + b * 2 でもよいでしょう。

③ a = a % (b + 1) です。この式のかっこは，ないといけません。

④ a -= 4 と書き換えられます。

⑤ a += 5 と書き換えられます。

⑥ a *= b + c と書き換えられます。b + c のかっこはなくてかまいません。

⑦これは，代入演算子で書き換えることはできません。

⑧ a += b / c と書き換えられます。もとの式で b / c が先に計算される（a = a + (b / c) である）ことに注意しましょう。

⑨ a *= a と書き換えればよいでしょう。

課題 60

8 行を「a = a - 1;」から「a--;」または「--a;」に書き換えてください。動作は変化しないはずです。

なお，「a--;」「--a;」はこれだけで a の値が変化しますから，改めて代入の必要はない（「a = a--;」などと書かない）ことに注意してください。

課題 61

4.8.2（本文 103 ページ）の**プログラム 15** は，6〜8 行の for ループで，「i を 1 から 10 までカウントして，10 回の繰り返し」をしています。これを，入力した値（たとえば n）までカウントするようにすれば，n 回の繰り返しになります。

このことがわかれば，プログラムは次のように書き換えられるでしょう。

```
1: #include <stdio.h>
2:
3: main() {
4:   int i, n;
5:
6:   scanf("%d", &n);
7:   for (i = 1; i <= n; i++) {
8:     printf("みりん¥n");
9:   }
10:  printf("ごくろうさん¥n");
11: }
```

課題 62

1 から n までの整数の和は，公式 $n(n+1)/2$ でただちに求められますが，ここでは繰り返しを使って $1+2+\cdots+n$ と求めることにしましょう。

n までの和を求めるとすると，

- 合計を記憶する変数 s を用意する
- i について，1 から n まで繰り返す
- i の値を s に足し込んでいく

という処理を行えばよいでしょう。ここで，合計を記憶する s は，0 で初期化しておかないといけないことに注意します。プログラムは，次のようにコーディングできるでしょう。

```
1: #include <stdio.h>
2:
3: main() {
4:   int n, i, s;
5:
6:   scanf("%d", &n);
7:   s = 0;
8:   for (i = 1; i <= n; i++) {
9:     s += i;
10:  }
11:  printf("%d¥n", s);
12: }
```

課題 63

プログラム 16（本文 106 ページ）の方針では，「i が 2 の倍数のとき」に加えています。ですから，この部分を「i が 3 の倍数のとき」に書き換えればよいのです。すなわち，8 行を

```
8:     if (i % 3 == 0) {
```

と書き換えます。

プログラム 17（本文 107 ページ）の方針は，「i に 2 を加えながら繰り返し，偶数のみを足していく」です。これを，「i に 3 を加えながら繰り返し」に変更すればよいのです。ですから，7 行を

```
7:   for (i = 0; i <= n; i += 3) {
```

と書き換えてください。

課題 64

約数をリストアップするには，1 からその数までで割ってみて，割り切れるかどうかを調べます。ここで，繰り返しを使います。「割り切れるかどうか」に剰余の演算子を使うことにも注意しましょう。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, i;
5:
6:    scanf("%d", &a);
7:    for (i = 1; i <= a; i++) {
8:      if (a % i == 0) {
9:        printf("%d¥n", i);
10:     }
11:   }
12: }
```

課題 65

平均を求めるには，データ数と，入力されたデータの合計が必要です。0 が入力されるまで，値が入力されるたびに，入力された数を合計に足し込み，データ数を 1 増やします。繰り返しは，「最後の 0 をカウントしない」のですから，**課題 17**（本文 43 ページ，解答は 5 ページ）のような形式になるはずです。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int a, n = 0, s = 0;
5:
6:    scanf("%d", &a);
7:    while (a != 0) {
8:      s += a;
9:      n++;
10:     scanf("%d", &a);
11:   }
12:   if (n == 0) {
13:     printf("データはありません¥n");
14:   } else {
15:     printf("平均は%d¥n", s / n);
16:   }
17: }
```

課題 66

まず，外側のループは，i について 0，1，2，……，9 と繰り返されることを確認してください。そして，内側のループは，毎回，0 から i 未満で繰り返されます。ですから，

- i が 0 のとき：継続条件は「j < 0」で 0 回
- i が 1 のとき：継続条件は「j < 1」で 1 回
- i が 2 のとき：継続条件は「j < 2」で 2 回
- i が 3 のとき：継続条件は「j < 3」で 3 回
- i が 4 のとき：継続条件は「j < 4」で 4 回
- i が 5 のとき：継続条件は「j < 5」で 5 回
- i が 6 のとき：継続条件は「j < 6」で 6 回
- i が 7 のとき：継続条件は「j < 7」で 7 回
- i が 8 のとき：継続条件は「j < 8」で 8 回
- i が 9 のとき：継続条件は「j < 9」で 9 回

繰り返されます。内側のループは，合計 45 回繰り返されます。

課題 67

4.11.2（本文 112 ページ）の繰り返しを参考にすれば，次のようなプログラムが書けるでしょう。1 から開始させ，与えられた数以下で繰り返している点に注意してください。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int i, j, n;
5:
6:    scanf("%d", &n);
7:    for (i = 1; i <= n; i++) {
8:      for (j = 1; j <= i; j++) {
9:        printf("%d ", j);
10:     }
11:     printf("¥n");
12:   }
13: }
```

課題 68

いくつかの考え方があります。

最も単純なのは，次のように作成することでしょう。しかし，芸がありません。

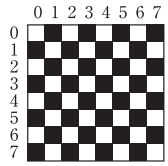
```
1:  #include <stdio.h>
2:
3:  main() {
```

```
    printf("□■□■□■□■¥n");
    printf("■□■□■□■□¥n");
    printf("□■□■□■□■¥n");
    printf("■□■□■□■□¥n");
    printf("□■□■□■□■¥n");
    printf("■□■□■□■□¥n");
    printf("□■□■□■□■¥n");
    printf("■□■□■□■□¥n");
}
```

少し考えれば，「□」から始まる行と「■」から始まる行が交互に現れるのですから，それを 1 組として 4 回繰り返す，というプログラムが書けるでしょう。

```
#include <stdio.h>

main() {
  int i;

  for (i = 0; i < 4; i++) {
    printf("□■□■□■□■¥n");
    printf("■□■□■□■□¥n");
  }
}
```

さらに，チェス盤のマスに図のように番号を振ると，縦の番号と横の番号の偶数・奇数が一致しているマスは「□」，一致していないマスは「■」であることがわかります。ですから，2 重のループを用いて，次のようなプログラムが作成できます。




```
#include <stdio.h>

main() {
  int i, j;

  for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
      if (i % 2 == j % 2) {
        printf("□");
      } else {
        printf("■");
      }
    }
    printf("¥n");
  }
}
```

このプログラムは少々複雑です。しかし，繰り返しの回数を変更することで，縦・横任意のサイズの市松模様を出力できます。

課題 69

約数の個数を数えるのは，**課題 64**（本文 107 ページ，解答は 18 ページ）と同じ方針で，与えられた数を 1 からその数まで繰り返しを使って割ってみればよいでしょう。これを，1 から入力された整数まで行うので，「約数の個数を調べる繰り返し」を繰り返す（二重の繰り返し）ことになります。

```
#include <stdio.h>

main() {
  int n, i, j, c;

  scanf("%d", &n);
  for (i = 1; i <= n; i++) {
    c = 0; /* 約数の個数は c に数える */
    for (j = 1; j <= i; j++) {
      if (i % j == 0) {
        c++; /* 割り切れたら約数の個数を
                1 増やす */
      }
    }
    printf("%d：%d 個¥n", i, c);
  }
}
```

第 5 章

課題 70

double 型変数を使うときの，printf()，scanf() 関数の使い方に注意しましょう。

```
#include <stdio.h>

main() {
  double r;

  scanf("%lf", &r);
```

```
    printf("円周：%f¥n", 2 * 3.14159 * r);
    printf("面積：%f¥n", 3.14159 * r * r);
}
```

課題 71

① 1。これは，整数どうしの割り算です。ですから，小数点以下が捨てられます。

② 2.25。a は int 型ですが，小数点以下を伴う 4.0 によって a の値が double 型に変換されます。

③ 1.8。b の値を double 型に変換しています。これによって，a の値も double に変換されます。

④ 0.8。double 型の x によって b が double 型に変換されます。

⑤ 1。x を int 型に変換しています。

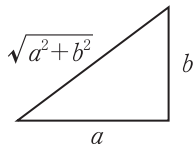
⑥ 1。a / b（整数どうしの割り算）を計算した後，double 型に変換しています。ですから，小数点以下は得られません。

⑦ 1。(b + x) で，b の値は double 型に変換されています。

⑧ 2。

課題 72

三平方の定理を使います。平方根を求めるのに sqrt() 関数を使用しますが，この使い方に注意してください。




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

main() {
    double a, b;

    scanf("%lf%lf", &a, &b);
    printf("%f¥n", sqrt(a * a + b * b));
}
```

課題 73

2 次方程式 $x^2+ax+b=0$ に対しての解の公式 $x=\frac{-a\pm\sqrt{a^2-4b}}{2}$ を用いて求めます。このとき，判別式（根号の中身） $D=a^2-4b$ が負になると実数解を持たないことに注意します。複号によって 2 つの解がもたらされることを利用すると，同じ計算を何度も実行しなくて済みます。

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

main() {
    double a, b, xr, xi;

    scanf("%lf%lf", &a, &b);
    if (a * a - 4 * b < 0) {
        printf("実数解がありません¥n");
    } else {
        xr = -a / 2;
        xi = sqrt(a * a - 4 * b) / 2;
        printf("x = %f, %f¥n", xr + xi,
        xr - xi);
    }
}
```

課題 74

課題 73 の 2 次方程式において，判別式が負になると，虚数解（$x=\frac{-a\pm\sqrt{4b-a^2}\,i}{2}$）を持ちます（$i$ は虚数単位）。判別式の値によって分岐して，虚数解の場合も出力させればよいわけです。ここで，虚数はそのままでは扱えませんから，実部と虚部を別々に計算して，出力のときに虚数らしく出力させる，ということになります。

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

main() {
    double a, b, d, xr, xi;

    scanf("%lf%lf", &a, &b);
    d = a * a - 4 * b;
    xr = -a / 2;
    if (d < 0) {
        /* 虚数解の場合 */
        xi = sqrt(-d) / 2;
        printf("x = %f +/- %fi¥n", xr, xi);
    } else {
        /* 実数解の場合 */
        xi = sqrt(d) / 2;
        printf("x = %f, %f¥n", xr + xi,
```

```
        xr - xi);
18:   }
19: }
```

第6章

課題 75

①要素の型は int，要素番号は 0〜99 です。
②要素の型は int，要素番号は 0〜200 です。
③要素の型は double，要素番号は 0〜4 です。

課題 76

配列の要素を繰り返しで処理するのは常套手段です。1 つの「型」として覚えてしまうとよいでしょう。

①

```
for (i = 0; i < 20; i++) {
  n[i] = 3;
}
```

②

```
for (i = 0; i < 100; i++) {
  x[i] = 99.9;
}
```

課題 77

① 要素数は 5 で，a[0] が 1，a[1] が 3，a[2] 〜 a[4] は 0 です。初期化値が与えられなかった要素は 0（a[2] 〜 a[4]）に初期化されます。

②要素数は 4 で，b[0] が 3，b[1] が 1，b[2] が 4，b[3] が 1 です。要素数を与えない場合，要素数は初期化値の数で決まります。

課題 78

プログラムで行わなければならないことを整理しましょう。

- 10 個の整数を入力し，平均を求める
- 求めた平均よりも大きいものを数え上げ，出力する

ここで，入力したデータは，「平均よりも大きいものを数え上げる」ときに必要になりますから，配列変数に記憶しておかなければなりません。

したがって，平均を計算するところまでは，**プログラム 22**（本文 134 ページ）と同じです。そのあと，入力された 10 個の整数について，平均を超えているかを調べていき，その数を（次に示したプログラムでは oa に）数えていきます。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int i, x[10], oa;
5:    double a = 0.0;
6:
7:    /* 10 個の値を入力して平均を計算 */
8:    for (i = 0; i < 10; i++) {
9:      scanf("%d", &x[i]);
10:     a += x[i];
11:   }
12:   a /= 10;
13:
14:   /* 平均を超えているものを数え上げる */
15:   oa = 0;
16:   for (i = 0; i < 10; i++) {
17:     if (x[i] > a) {
18:       oa++;
19:     }
20:   }
21:   printf("平均を超えているものは %d 個 ¥n",
       oa);
22: }
```

課題 79

たとえば，入力が有効であったかを調べる変数 v を用意し，有効ならば 1，そうでないならば 0 を値に持つと約束します。これで，0 未満または 100 を超える入力の場合にこの v を 0 とし，繰り返させればよいでしょう。**プログラム 23**（本文 138 ページ）で，int 型の変数 v を定義し，12 行を次のように書き換えます。

```
do {
  v = 1;
  scanf("%d", &x);
  if (x < 0) {
    v = 0;
  }
  if (x > 100) {
    v = 0;
  }
} while (v == 0);
```

6.6.3（本文 146 ページ）の論理和演算子を使えば，v を使わず次のように書けます。

```
do {
  scanf("%d", &x);
} while (x < 0 || x > 100);
```

課題 80

このプログラムでは，1〜20 の整数を 5 個，入力します。ここで，

- 5 個の要素を持つ配列に，入力された数を覚える
- 20 個の要素を持つ配列に，ある数が入力されたかどうかを覚える

という 2 つの方針が考えられます。簡単なのは，後者でしょう。

ここでは，入力した整数との対応をとりやすいように，要素番号 20 まで，すなわち 21 個の要素を持つ配列 a[] を考えます。これについて，x が入力されたら a[x] を 1，されていなければ 0 と約束します。入力後に，a[x] の値によって「*」「.」を出力させればよいのです。

```
#include <stdio.h>

main() {
  int a[21], x, i;

  /* a[] の要素を 0（入力されていない）で初期化 */
  for (i = 0; i < 21; i++) {
    a[i] = 0;
  }
  /* 5 個の整数を入力 */
  for (i = 0; i < 5; i++) {
    scanf("%d", &x);
    a[x] = 1;
  }
  /* a[] の値を調べて出力 */
  for (i = 1; i <= 20; i++) {
    if (a[i] == 1) {
      printf("*");
    } else {
      printf(".");
    }
  }
  printf("¥n");
}
```

もし，前者（入力した数を覚える）の方針でプログラムを作成した場合，出力の際に配列の値を調べなければいけないので，少々厄介です。

```
#include <stdio.h>

main() {
  int a[5], i, j, f;

  /* 5 個の整数を入力 */
  for (i = 0; i < 5; i++) {
    scanf("%d", &a[i]);
  }
  /* 入力された値を調べながら出力 */
  for (i = 1; i <= 20; i++) {
    f = 0; /* a[] に i が見つかったら 1 にする */
    for (j = 0; j < 5; j++) {
      if (a[j] == i) {
        f = 1;
      }
    }
    if (f == 1) {
      printf("*");
    } else {
      printf(".");
    }
  }
  printf("¥n");
}
```

課題 81

「2 つめの日付が 1 つめの日付より前」とはどんな状況かを考えます。1 つめの日付を m1 月 d1 日，2 つめの日付を m2 月 d2 日とします。

- m1 < m2 ならば，1 つめの日付のほうが前です。
- m1 == m2 ならば，d1 > d2 のときに 2 つめの日付のほうが前になります。
- m1 > m2 ならば，2 つめの日付のほうが前です。

2 つめの日付が前のときに，m1 と m2，d1 と d2 の値を交換すればよいでしょう。ここで，同日の場合は交換してもしなくてもよいことに注意してください。

書き換えは，**プログラム 24**（本文 142 ページ）について，一時的な変数 t を定義しておき（変数の値を交換するときの手順を復習してください），11 行と 12 行の間に次のコードを追加することになります。

```
if (m1 > m2) {
  t = m1; m1 = m2; m2 = t;
  t = d1; d1 = d2; d2 = t;
} else if (m1 == m2) {
  if (d1 > d2) {
    t = m1; m1 = m2; m2 = t;
```

```
      t = d1; d1 = d2; d2 = t;
    }
  }
```

または，**6.6.3**（本文 146 ページ）の論理和・論理積演算子を用いれば，追加部分は次のように書けます。

```
  if (m1 > m2 || (m1 == m2 && d1 > d2)) {
    t = m1; m1 = m2; m2 = t;
    t = d1; d1 = d2; d2 = t;
  }
```

課題 82

たとえば，823 円について考えます。これは，500 円玉 1 枚，100 円玉 3 枚，10 円玉 2 枚，1 円玉 3 枚で出すことになります。これは，残額に対し最も高額な貨幣を出すという手順になります。つまり，

- 残額 823 円：500 円玉 1 枚
- 残額 323 円：100 円玉 3 枚
- 残額 23 円：50 円玉は出せない
- 残額 23 円：10 円玉 2 枚
- 残額 3 円：5 円玉は出せない
- 残額 3 円：1 円玉 3 枚

という手順です。ここで 100 円玉を出す場面を見てみると，「323 円に対し，100 円玉を 3 枚出し，残額が 23 円」と，枚数と残額が商と余りの関係になっています。

それで，プログラムは，金種を配列に記憶しておき，前述の「商と余りの関係」を使って，何枚ずつになるかを求めていきます。

```
1:  #include <stdio.h>
2:
3:  main() {
4:    int cur[10] = { 10000, 5000, 2000,
        1000, 500, 100, 50, 10, 5, 1 };
5:    int money, x, i;
6:
7:    /* 金額を入力 */
8:    scanf("%d", &money);
9:    /* 枚数を計算 */
10:   for (i = 0; i < 10; i++) {
11:     x = money / cur[i];
12:     if (x != 0) {
13:       printf("%d 円：%d 枚¥n", cur[i], x);
14:       money %= cur[i];
15:     }
16:   }
17: }
```

ここで，10〜16 行の繰り返しでは「すべての金種について」調べていますが，「残金がある間」調べるという考え方でもかまいません。その場合，10 行は，

```
10:   for (i = 0; money > 0; i++) {
```

となります。このとき，「i < 10」の条件がなくても，残額は最終的に 1 円玉を使って 0 にされる（繰り返しが終了する）ことが保証されていることに注意してください。

課題 83

①「b は 6 ではないかといったら，そうではない（b は 6）」なので，**偽**です。

②a + b は 11，c は 8 ですから，「a + b > c」は**真**です。

③b % a は 1，c / b は 1 です（整数どうしの割り算に注意）。ですから，「b % a == c / b」は**真**です。

④「a > 0」は真ですが，! 演算子で真偽を反転しているので，**偽**です。

⑤a == b は偽ですが，c > b は真です。|| 演算子で一方でも真ならば全体が**真**になります。

⑥c - a > b は偽で，&& 演算子は両方真でないと全体が真になりませんから，この時点で全体が**偽**であることがわかります（これがショートサーキットです）。念のため a > c を調べると，こちらも偽です。

課題 84

それぞれの生の数値が何を表しているかに注意して書き換えましょう。

```
1:  #include <stdio.h>
2:
3:  #define NDATA 40
4:  #define RANK 11
5:  #define STEP 10
6:
7:  main() {
8:    int i, j, x, r[RANK];
9:
10:   /* r[] を初期化 */
11:   for (i = 0; i < RANK; i++) {
12:     r[i] = 0;
13:   }
14:   /* NDATA 個のデータを入力して集計 */
15:   for (i = 0; i < NDATA; i++) {
16:     scanf("%d", &x);
17:     r[x / STEP]++;
18:   }
```

```
19:   /* ヒストグラムを表示 */
20:   for (i = 0; i < RANK; i++) {
21:     printf("%3d: ", i * STEP);
22:     for (j = 0; j < r[i]; j++) {
23:       printf("*");
24:     }
25:     printf("¥n");
26:   }
27: }
```

課題 85

記号定数が定義されていますから，3〜5 行を，

```
3:  #define NDATA 60
4:  #define RANK 21
5:  #define STEP 5
```

と書き換えます。記号定数を使っていないと，この変更がことのほか面倒であることは想像できるでしょう。

課題 86

① 8 × 7 で 56 個です。x[7][6] まで使えます。
② 10 × 15 で 150 個です。v[9][14] まで使えます。
③ 4 × 3 × 6 で 72 個です。a[3][2][5] まで使えます。

課題 87

2 重の for ループを使います。

```
for (i = 0; i < 10; i++) {
  for (j = 0; j < 20; j++) {
    a[i][j] = i + j;
  }
}
```

課題 88

2 次元配列の扱いに注意して，次のようにプログラムを作成すればよいでしょう。

```
1:  #include <stdio.h>
2:
3:  #define SIZE 10
4:
5:  main() {
6:    int a[SIZE][SIZE], x, y, z, i;
7:
8:    /* a[][] を 0 で初期化する */
9:    for (y = 0; y < SIZE; y++) {
10:     for (x = 0; x < SIZE; x++) {
11:       a[x][y] = 0;
12:     }
13:   }
14:   /* x, y, z の組を 10 回入力する */
15:   for (i = 0; i < 10; i++) {
16:     scanf("%d%d%d", &x, &y, &z);
17:     a[x][y] = z;
18:   }
19:   /* a[][] を表形式で出力する */
20:   for (y = 0; y < SIZE; y++) {
21:     for (x = 0; x < SIZE; x++) {
22:       printf("%d", a[x][y]);
23:     }
24:     printf("¥n");
25:   }
26: }
```

第 7 章

課題 89

まず，番兵を使わず，**プログラム 27**（本文 160 ページ）と同様に作成します。入力された数より大きい値が見つかったら停止するのですから，継続条件は「入力された数以下である間」です。これより，プログラムは，次のようになります。

```
1:  #include <stdio.h>
2:
3:  #define N 10
4:
5:  main() {
6:    int a[N] = { 31, 41, 59, 26, 53, 58,
       97, 93, 23, 84 };
7:    int i, x;
8:
9:    /* 探索する値を入力 */
10:   printf(" 探索する値：");
11:   scanf("%d", &x);
12:   /* 線形探索 */
13:   for (i = 0; i < N && a[i] <= x; i++)
14:     ;
15:   /* 探索結果の出力 */
16:   if (i == N) {
17:     printf("%d より大きい値はありません。
        ¥n", x);
```

```
18:    } else {
19:      printf("%d より大きい値 %d が位置 %d に
         見つかりました。¥n", x, a[i], i);
20:    }
21:  }
```

次に，番兵を使う場合を考えます。等しい値を探すときは，配列の末尾に探す値と同じ値を入れて，繰り返しを停止させるのでした。ですからこの場合も，配列の末尾に繰り返しを停止させるような値を入れればよいのです。この課題では，入力した値より大きい値を探すのですから，配列の末尾に，入力した値より確実に大きい値を入れておけばよいでしょう。たとえば，入力した数より 1 大きい値を入れれば，確実に停止します。ですから，前掲のプログラムにおいて，

- 6 行で定義する配列の要素数を `N + 1` にする
- 11 行と 12 行の間で，「`a[N] = x + 1;`」として番兵を仕込む
- 13 行を「`for (i = 0; a[i] <= x; i++)`」と書き換える

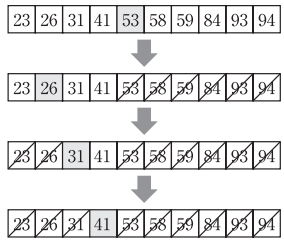
と書き換えればよいことになります。

課題 90

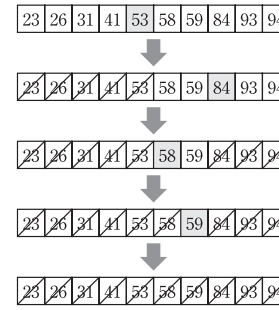
「41」を探してみましょう。

- 探索対象は `a[0]`〜`a[9]`：`(0 + 9) / 2` は 4。`a[4]` は 53 で，`a[4]`〜`a[9]` に 41 はない
- 探索対象は `a[0]`〜`a[3]`：`(0 + 3) / 2` は 1。`a[1]` は 26 で，`a[0]`〜`a[1]` に 41 はない
- 探索対象は `a[2]`〜`a[3]`：`(2 + 3) / 2` は 2。`a[2]` は 31 で，`a[2]` に 41 はない
- 探索対象は `a[3]`：この値は 41 なので，探すべき値 41 が見つかった

となります。



「62」が見つからない手順について，本文 164 ページで述べていますが，これを図で表すと次のようになります。



課題 91

線形探索の場合から考えてみましょう。`a[0]`〜`a[15]` で考えます。

`a[0]` から値を発見するのに繰り返しが 1 回行われます。`a[1]` から値を発見するときは 2 回です。以下同様に，`a[15]` からは 16 回繰り返されます。この平均は，8.5 回です。

次に，二分探索は，次のように値が探されます。着目する要素の番号を書き出していくと，

- `a[0]`：4 回（7 → 3 → 1 → 0）
- `a[1]`：3 回（7 → 3 → 1）
- `a[2]`：4 回（7 → 3 → 1 → 2）
- `a[3]`：2 回（7 → 3）
- `a[4]`：4 回（7 → 3 → 5 → 4）
- `a[5]`：3 回（7 → 3 → 5）
- `a[6]`：4 回（7 → 3 → 5 → 6）
- `a[7]`：1 回（7）
- `a[8]`：4 回（7 → 11 → 9 → 8）
- `a[9]`：3 回（7 → 11 → 9）
- `a[10]`：4 回（7 → 11 → 9 → 10）
- `a[11]`：2 回（7 → 11）
- `a[12]`：4 回（7 → 11 → 13 → 12）
- `a[13]`：3 回（7 → 11 → 13）
- `a[14]`：4 回（7 → 11 → 13 → 14）
- `a[15]`：5 回（7 → 11 → 13 → 14 → 15）

これより，平均は 3.375 回です。

この回数をプログラムで調べるには，線形探索（**プログラム 28**〈本文 162 ページ〉）では，たとえばプログラムの最後（21 行と 22 行の間）で `i + 1` を出力させればよいでしょう。

```
printf("繰り返し回数：%d¥n", i + 1);
```

二分探索では，**プログラム 29**（本文 165 ページ）において，回数を数える変数を 1 で初期化して用意し（n としましょう），繰り返しごとに n を 1 ずつ増やします。20 行と 21 行の間に

```
    n++;
```
を書き加えればよいでしょう。そして，最後（27 行と 28 行の間）に，n の値を出力させます。

```
    printf("繰り返し回数：%d¥n", n);
```

課題 92

空所 [あ] では，整列前の a[] の要素を出力しています。本文 172 ページの出力例に倣えば，次のようなコードを補充することになります。

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

空所 [い] は，a[i + 1] から a[N - 1] の最小値を探す処理です。勝ち抜き式で最小値を求めます。k に記憶されるのは，最小値ではなく，最小値を持つ要素番号であることに注意してください。

```
    for (j = i + 1; j < N; j++) {
      if (a[j] < a[k]) {
        k = j;
      }
    }
```

空所 [う] は，a[i] と a[k] の値を交換する処理です。一時的な変数を使うという定法に従って，次のように補充します。

```
    t = a[i]; a[i] = a[k]; a[k] = t;
```

空所 [え] では，整列後の a[] の要素を出力させています。これは，空所 [あ] に補充したコードと同じでよいでしょう。

以上から，完全な形の選択ソートのプログラム（**プログラム 30**〈本文 172 ページ〉を完成させたもの）は，次のようになります。

```
 1: #include <stdio.h>
 2:
 3: #define N 10
 4:
 5: main() {
 6:   int a[N] = { 31, 41, 59, 26, 53, 58,
        97, 93, 23, 84 };
 7:   int i, j, k, t;
 8:
 9:   /* 整列前の a[] の内容を出力 */
10:   printf("整列前：");
11:   for (i = 0; i < N; i++) {
12:     printf("%d ", a[i]);
13:   }
14:   printf("¥n");
15:   /* 選択ソート */
16:   for (i = 0; i < N - 1; i++) {
17:     /* a[i]～a[N - 1] の最小値がある要素番号
          を見出す */
18:     k = i; /* まず要素番号 i を仮の最小とする
          */
19:     for (j = i + 1; j < N; j++) {
20:       if (a[j] < a[k]) {
21:         k = j;
22:       }
23:     }
24:     /* a[i] と a[k] を交換 */
25:     t = a[i]; a[i] = a[k]; a[k] = t;
26:   }
27:   /* 整列済みの a[] の内容を出力 */
28:   printf("整列後：");
29:   for (i = 0; i < N; i++) {
30:     printf("%d ", a[i]);
31:   }
32:   printf("¥n");
33: }
```

課題 93

a[] を出力させる処理は，**課題 92** の空所 [あ] または [え] と同様の処理です。ただし，変数 i がその外の for ループで使われているので，別の変数でループを制御します。

```
    for (j = 0; j < N; j++) {
      printf("%d ", a[j]);
    }
    printf("¥n");
```

これを追加して実行すると，本文 170 ページの図のように整列が行われていることを確認できます。

課題 94

昇順（小さい順）に並べるときには，未整列部分の**最小値**を探していました。これを，**最大値**を探すようにすれば，降順（大きい順）に並べることができます。

もとのプログラムで，未整列部分の最小値を探しているのは空所 [い] の部分です。これを，次のように書き換えれば，降順に並べ替えられます。

```
    for (j = i + 1; j < N; j++) {
      if (a[j] > a[k]) {
        k = j;
```

```
    }
  }
```

課題 95

比較の回数を数えます。比較の回数は，j での繰り返しの回数と同じであることに気付いてください。

- i が 0 のとき，j は 1〜9 まで繰り返し：9 回
- i が 1 のとき，j は 2〜9 まで繰り返し：8 回
- ……
- i が 8 のとき，j は 9 で実行：1 回

ですから，合計で**比較は 45 回**です。

交換は，i での繰り返しに 1 回につき 1 回実行されますから，**9 回**です。

配列の要素数が 20 個になると，比較は i が 0 のとき 19 回，1 のとき 18 回，……，18 のとき 1 回，で合計 190 回になります。交換は，19 回です。

同様に，要素数が 30 個になると，比較は i が 0 のとき 29 回，1 のとき 28 回，……，28 のとき 1 回，で合計 435 回です。交換は，29 回です。

比較の回数は，要素数が 10 から 20 と 2 倍になると，4 倍強になっています。10 から 30 と 3 倍になると，10 倍弱になっています。これは，およそ**要素数の 2 乗に比例**する増え方です。一方，交換の回数は，**要素数にほぼ比例**しています。

ですから，選択ソートは，データ数が多くなってくると，その回数がデータ数の 2 乗に比例する，比較の手間がかさんできます。

比較・交換の回数をプログラムで調べるには，**課題 92** で完成させたプログラムについて，比較については 19 行と 20 行の間で，交換については 24 行と 25 行の間で，用意した適当な変数を 1 ずつ増やして回数を数えればよいでしょう。

課題 96

空所 [あ]，[い] ともに，a[] の内容を出力させています。自由に使える変数に注意して，**課題 92**（本文 173 ページ，解答は 26 ページ）と同じく，いずれも次のようなコードを補充すればよいでしょう。

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

課題 97

課題 96 に準じた，a[] の内容を出力させるコードを追加します。ただし，変数 i はその外側のループで使用していますので，異なる変数をループの制御に使いましょう。

```
for (j = 0; j < N; j++) {
  printf("%d ", a[j]);
}
printf("¥n");
```

本文 175 ページの図と同様な，整列の経過が出力されるはずです。

課題 98

昇順に並べるときは，「整列済み部分を後ろから見ていき，挿入したい値より**小さい**値が初めて現れた位置の次」に差し込みます。降順の場合は，「挿入したい値より**大きい**値が初めて現れた位置の次」に差し込めばよいのです。

ですから，**プログラム 31**（本文 178 ページ）の 15〜17 行を，次のように書き換えます。

```
15:     for (j = i - 1; i >= 0 && a[j] < t;
         j--) {
16:       a[j + 1] = a[j];
17:     }
```

課題 99

プログラム 31（本文 178 ページ）の 15 行から「j >= 0」の条件をはずすと，

```
for (j = i - 1; a[j] > t; j--) { …… }
```

と繰り返されることになります。番兵を仕込んで，確実にこの繰り返しを終わらせるには，a[0] が t 以下ならばよいわけです。ですから番兵は「a[0] = t;」と仕込めばよいでしょう。

これを踏まえてプログラムを書き換えると，次のようになります。a[1]〜a[N] を整列させる点にも注意してください。

```
1:  #include <stdio.h>
2:
3:  #define N 10
4:
5:  main() {
6:    int a[N + 1] = { 0, 31, 41, 59, 26,
       53, 58, 97, 93, 23, 84 };
7:    int i, j, t;
8:
9:    /* 整列前の a[] の内容を出力 */
10:   printf("整列前：");
```

```
11:   for (i = 1; i <= N; i++) {
12:     printf("%d ", a[i]);
13:   }
14:   printf("¥n");
15:   /* 挿入ソート */
16:   for (i = 2; i <= N; i++) {
17:     t = a[i];
18:     a[0] = t; /* 番兵を仕込む */
19:     for (j = i - 1; a[j] > t; j--) {
20:       a[j + 1] = a[j];
21:     }
22:     a[j + 1] = t;
23:   }
24:   /* 整列後の a[] の内容を出力 */
25:   printf("整列後：");
26:   for (i = 1; i <= N; i++) {
27:     printf("%d ", a[i]);
28:   }
29:   printf("¥n");
30:   return 0;
31: }
```

課題 100

空所［あ］，［う］ともに，a[] の内容を出力させています。これは，**課題 96**（本文 178 ページ，解答は 27 ページ）と同じです。次のコードを補充すればよいでしょう。

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

空所［い］は，昇順に並べるにあたって a[j] > a[j + 1] のとき，すなわち並べたい順序になっていないときに実行されます。本文 181 ページの説明で，この場合はこの 2 つの要素の値を交換します。ですから，交換の定法に従って，

```
t = a[j];
a[j] = a[j + 1];
a[j + 1] = t;
```

を補充します。

課題 101

課題 97（本文 178 ページ，解答は 27 ページ）と同様です。外側のループで i が使われていることに注意して，

```
for (j = 0; j < N; j++) {
  printf("%d ", a[j]);
}
printf("¥n");
```

を追加すれば，本文 182 ページの図のような，整列の経過が出力されます。

課題 102

バブルソートの核心は，「並べたい順序と異なっている箇所を，入れ替えまくって並べたい順序に直していく」ことです。そして，これを行っているのは，**プログラム 32**（本文 183 ページ）の 15〜17 行です。**昇順**に並べるとき，a[j] > a[j + 1]，すなわち a[j] と a[j + 1] が**降順**の関係ならば入れ替えています。ですから，**降順**に並べるときは，a[j] と a[j + 1] が**昇順**の関係のときに入れ替えればよいのです。ですから，15 行を，

```
15:       if (a[j] < a[j + 1]) {
```

と書き換えれば，降順に整列されます。

課題 103

1 セットごとの交換の回数を int 型変数 ex に数えあげるものとし，プログラムの冒頭（**プログラム 32**〈本文 183 ページ〉の 7 行）で定義しておきます。

未整列の要素をひととおり調べて，並べたい順序と異なる部分を入れ替える「1 セット」は，14〜18 行の繰り返しです。この中で起こる交換を数え上げるのですから，13〜19 行を次のように書き換えればよいでしょう。

```
for (i = 0; i < N - 1; i++) {
  ex = 0; /* 交換回数をリセット */
  for (j = 0; j < N - 1 - i; j++) {
    if (a[j] > a[j + 1]) {
      t = a[j];
      a[j] = a[j + 1];
      a[j + 1] = t;
      ex++;
    }
  }
  printf("交換：%d 回¥n", ex);
}
```

ここで，配列が整列済みならば，交換は発生しません。すなわち，j についての繰り返しが終わった時点で，ex は 0 のままです。このとき，i についての繰り返しが途中であっても，i についての繰り返しを終了させます。そのためには，i についての for ループを，

```
for (i = 0; i < N - 1 && ex > 0; i++) {
  ……
}
```

としておけば，ex が 0 のとき繰り返しが終了します。ここで考慮しなければならないのは初回です。初回，ex は不定ですから，0 より大きいダミーの値を代入しておき，i の初回の繰り返しを実行させます。

以上から，**プログラム 32** 全体を書き換えると，次のようになります。

```
#include <stdio.h>

#define N 10

main() {
  int a[N] = { 31, 41, 59, 26, 53, 58,
   97, 93, 23, 84 };
  int i, j, t, ex;

  /* 整列前の a[] の内容を出力 */
  printf("整列前：");
  for (i = 0; i < N; i++) {
    printf("%d ", a[i]);
  }
  printf("¥n");
  /* バブルソート */
  ex = 1; /* 繰り返し実行のためのダミー値 */
  for (i = 0; i < N - 1 && ex > 0; i++) {
    ex = 0;
    for (j = 0; j < N - 1 - i; j++) {
      if (a[j] > a[j + 1]) {
        t = a[j];
        a[j] = a[j + 1];
        a[j + 1] = t;
        ex++;
      }
    }
  }
  /* 整列後の a[] の内容を出力 */
  printf("整列後：");
  for (i = 0; i < N; i++) {
    printf("%d ", a[i]);
  }
  printf("¥n");
}
```

課題 104

この書き換えでは，内側の j についての繰り返しが N - 1 から始まり，j の値を減らしていくようになっています。本文中の**プログラム 32** では，「最大値を未整列部分の末尾に送るように」交換をしていきましたが，繰り返しが配列の末尾から先頭に向かいますから，「最小値を未整列部分の先頭に送るように」交換していくことになります。i が 0 のとき，最小値は a[0] に送られ，i が 1 では a[1] に送られ……，と繰り返されていきます。

ここで［え］はいったんおいて，［お］を見てみましょう。ここでは交換を行います。前の行で a[j - 1] と a[j] を比較して，降順になっていたら入れ替えますから，

```
t = a[j - 1];
a[j - 1] = a[j];
a[j] = t;
```

を補充します。すると，初回に a[0] までが交換の対象となるためには，j - 1 が 0 になるまで繰り返せばよいわけで，このとき j は 1 です。ですから［え］は「i が 0 のとき j が 1 まで繰り返されるように」，

```
j > i
```

を補充します（「j >= i」だと 1 回余計に繰り返されてしまいます）。

第 8 章

課題 105

①3，4，5 の中央値は 4 です。

②引数となっている式を計算すると，順に 6，9，5 です。ですから式の値は 6 です。

③引数は，順に，5，8，1 です。中央値は 5，これに 4 を加えますから，式の値は 9 です。

④2 番目の引数は mid(1, 8, 5) で，この値は 5 です。したがって，10，5，4 の中央値を求めることになり，式の値は 5 になります。

⑤mid(3, 7, 2) は 3，mid(4, 6, 8) は 6 です。よって式全体の値は 18 になります。

課題 106

①戻り値の型は int，引数の型は，1 つめが int，2 つめも int です。

②戻り値の型は int，引数は 3 つとも int 型の値を受け取ります。

③「戻り値の型」が void なので，戻り値はありません。int 型の引数を 1 個とります。

④戻り値の型は double です。1つめの引数は double 型，2つめの引数は int 型です。

⑤戻り値の型は int です。引数はとりません。

課題 107

① int sqr(int a)

② double expedSum(double x, double y, int n)

③ void printStatus(void)

課題 108

①引数の値に1を加えたものを返すだけです。

```
int plus1(int a) {
  return a + 1;
}
```

ただし，「1加える」に対して「return a++;」では，うまくいきません。「a++」は，実行後に a の値は1増えますが，「a++」という式の値は増える前の値だからです。ですから，上記のように素直に「a + 1」を返しましょう。

②180度がπラジアンですから，180で割って円周率をかければ換算できます。

```
float deg2rad(float deg) {
  return deg / 180 * 3.14159;
}
```

③「差の絶対値」ですから，2つの数の差が負だったら符号を入れ替えて正の値にして返します。if 文で分岐すればよいでしょう。

```
int iabs(int a, int b) {
  if (a - b < 0) {
    return b - a;
  }
  return a - b;
}
```

④1分は60秒，1時間は3600秒です。

```
int inSec(int h, int m, int s) {
  return h * 3600 + m * 60 + s;
}
```

⑤x の y 乗は，繰り返しによって x を y 回かければ求められます。

```
int ipow(int x, int y) {
  int p = 1, i;

  for (i = 0; i < y; i++) {
    p *= x;
  }
  return p;
}
```

⑥まず，1は素数ではありません。そして，素数は，1と自分自身以外では割り切れない正の整数ですから，簡単には，2以上自分自身未満で割り切れるかを（繰り返しで）調べればよいことになります。2以上自分自身未満のいずれかで割り切れたら，ただちに「素数ではない」と判定できます(その時点で0を返せます)。一度も割り切れなかったら，1を返せばよいでしょう。1は素数でないことに注意して，次のように書けます。

```
int isPrime(int n) {
  int i;

  if (n == 1) {
    return 0; /* 1 は素数でない */
  }
  for (i = 2; i < n; i++) {
    if (n % i == 0) {
      return 0; /* 一度でも割り切れた
      ら素数ではない */
    }
  }
  return 1; /* 一度も割り切れなかったら
  素数 */
}
```

ここで，「2」が正しく判定されるかを確認しておくと，繰り返しは1回も実行されませんから，ただちに1が返され素数と判定されることになります。「1」は上記のように特別扱いしないと，2のときと同様に繰り返しが実行されず1を返して素数と判定してしまいます。

課題 109

a，*b*，*c*，*d*の最大値は，『*a*，*b*の最大値』と『*c*，*d*の最大値』の最大値」と考えることができます。トーナメント式で求めていると考えてもよいでしょう。ですから，max2() を使って，次のように書けます。

```
int max4(int a, int b, int c, int d) {
  int m, n;

  m = max2(a, b);
  n = max2(c, d);
  return max2(m, n);
}
```

さらに，上記は，m，n を使わずに次のように書けます。

```
int max4(int a, int b, int c, int d) {
  return max2(max2(a, b), max2(c, d));
}
```

さて，4つの数を入力して最大値を出力するプログラム

は，このmax4()が定義されているとして，**プログラム39**（本文202ページ）のmain()関数を次のように書き換えて作ることができます。

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

  a = scanInt();
  b = scanInt();
  c = scanInt();
  d = scanInt();
  printf("%d¥n", max4(a, b, c, d));
}
```

課題 110

次のように書き換えられます。**プログラム44**（本文211ページ）に比べて冗長な感じがすることでしょう。

```
1:  #include <stdio.h>
2:
3:  int main(void) {
4:    int n, r, c, i;
5:
6:    scanf("%d%d", &n, &r);
7:    c = 1;
8:    for (i = 1; i <= n; i++) {
9:      c *= i;
10:   }
11:   for (i = 1; i <= r; i++) {
12:     c /= i;
13:   }
14:   for (i = 1; i <= n - r; i++) {
15:     c /= i;
16:   }
17:   printf("%d個から%d個取る組み合わせ：
      %d¥n", n ,r, c);
18:   return 0;
19: }
```

課題 111

順列を求める式も，階乗を使って表されています。ですから，**プログラム44**（本文211ページ）の，組み合わせを階乗で求めている部分（18行）を，順列の式に書き換えればよいのです。出力の体裁（19行）も整えるならば，次のように書き換えればよいでしょう。

```
18:   c = fact(n) / fact(n - r);
19:   printf("%d個から%d個取る順列：%d¥n", n, r, c);
```

課題 112

まず，指示された関数fact2()を作りましょう。

```
int fact2(int n, int m) {
  int i, p = 1;

  for (i = n; i <= m; i++) {
    p *= i;
  }
  return p;
}
```

さて，fact()の代わりにこのfact2()を使って組み合わせの数を求めるプログラムを作ります。fact2()で計算される値，すなわち，$n\cdot(n+1)\cdot\cdots\cdot(m-1)\cdot m$を$f(n, m)$と表すことにします。

${}_nC_r=\dfrac{n!}{r!\cdot(n-r)!}$について，次の2通りの表し方が考えられます。

$${}_nC_r=\frac{n!}{r!\cdot(n-r)!}=\frac{n\cdot(n-1)\cdot\cdots\cdot2\cdot1}{r\cdot(r-1)\cdot\cdots\cdot2\cdot1}\cdot\frac{1}{(n-r)!}$$

$$=\frac{n\cdot(n-1)\cdot\cdots\cdot(r+2)\cdot(r+1)}{(n-r)!}=\frac{f(r+1, n)}{f(1, n-r)}$$

$${}_nC_r=\frac{n!}{r!\cdot(n-r)!}=\frac{n\cdot(n-1)\cdot\cdots\cdot2\cdot1}{(n-r)\cdot(n-r-1)\cdot\cdots\cdot2\cdot1}\cdot\frac{1}{r!}$$

$$=\frac{n\cdot(n-1)\cdot\cdots\cdot(n-r+2)\cdot(n-r+1)}{r!}=\frac{f(n-r+1, n)}{f(1, r)}$$

これらのどちらの式を使ってもよいのですが，fact2()の値が大きくならないほうを選んだほうが，対応する範囲が広くなるでしょう。fact2()でかけ合わせる数（繰り返しの回数）が少ないほうがよいと考えられます。そこで，

- $n-r\leqq r$のときは前者の式
- $n-r>r$のときは後者の式

で求めることにしましょう。

以上から，fact2()を定義しておき，**プログラム44**（本文211ページ）main()関数中，18行を，次のように書き換えます（fact()の定義は削除します）。

```
if (n - r <= r) {
  c = fact2(r + 1, n) / fact2(1, n - r);
} else {
  c = fact2(n - r + 1, n) / fact2(1, r);
}
```

これで対応できる値の範囲が広がりますが，1つ，注意

しなければならないことがあります。rが0のときです。rが0ならば，組み合わせの数は常に1になります。rが0のときは，必ず$n-r>r$の場合，すなわち後者の式で計算されます。このとき，fact2(n + 1, n)，fact2(1, 0)という呼び出しが起こります。これは，fact2()を作ったときには想定していないものです。実際，この呼び出しに対する戻り値は，いずれも1になります。したがって組み合わせの数として1が求められます。つまり，「たまたま」正しい答えが得られています。

課題 113

まず，指示された関数dabs()を作ります。

```
double dabs(double x, double y) {
  if (x > y) {
    return x - y;
  }
  return y - x;
}
```

これを使って，main()関数を次のように作成すればよいでしょう。

```
#include <stdio.h>

int main(void) {
  double x0, y0, x1, y1;

  scanf("%lf%lf%lf%lf",
    &x0, &y0, &x1, &y1);
  printf("マンハッタン距離：%f¥n",
    dabs(x0, x1) + dabs(y0, y1));
  return 0;
}
```

課題 114

この課題では，自作した関数dabs()の代わりに，<math.h>を読み込むことで使うことができる関数fabs()を使います。「#include <math.h>」で情報を読み込んだら，fabs()を定義したりすることは不要であることに注意してください。

```
1:  #include <stdio.h>
2:  #include <math.h>
3:
4:  int main(void) {
5:    double x0, y0, x1, y1;
6:
7:    scanf("%lf%lf%lf%lf", &x0, &y0, &x1,
        &y1);
8:    printf("マンハッタン距離：%f¥n",
        fabs(x1 - x0) + fabs(y1 - y0));
9:    return 0;
10: }
```

課題 115

この関数では，受け取った配列の内容を書き換えて呼び出し元に影響させています。size個の要素数の配列の要素番号は，0からsize - 1であることに注意してください。

```
void sqrAll(int a[], int size) {
  int i;

  for (i = 0; i < size; i++) {
    a[i] *= a[i];
  }
  return;
}
```

なお，この関数は戻り値がなく，必ず最後まで実行されますから，return文はなくてもかまいません。

課題 116

関数内で渡ってきた配列の値を操作してしまうと，呼び出し元に影響が出てしまいます。この場合，a[]の各要素の値の絶対値を計算して，a[]の要素に書き戻してから出力させる方法を使うと，呼び出し元に影響が出てしまいます。

```
void printAbs(int a[], int size) {
  int i;

  for (i = 0; i < size; i++) {
    if (a[i] > 0) {
      printf("%d¥n", a[i]);
    } else {
      printf("%d¥n", -a[i]);
    }
  }
  return;
}
```

この場合も，最後のreturn文はなくてもかまいません。

課題 117

この関数は，「実質的に配列が戻る」関数です。配列 c[] を呼び出し元で用意し，引数として渡し，これに処理結果を格納して戻しています。ここで，c[] の値は変更しますが，a[]，b[] は値を変更してはならないことに注意してください。

```
void subSqr(int a[], int b[], int c[],
    int size) {
  int i;

  for (i = 0; i < size; i++) {
    c[i] = (a[i] - b[i])
      * (a[i] - b[i]);
  }
  return;
}
```

課題 118

2 次元配列を受け取る場合です。この関数は，10×10 の配列を渡すと約束していますから，要素数はわかっているものと考えます。

```
int gridSum(int a[10][10]) {
  int i, j, sum = 0;

  for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
      sum += a[i][j];
    }
  }
  return sum;
}
```

ここで，関数の仮引数は，1 次元めの要素数は書かずに「int a[][10]」でもかまいません。しかし，2 次元めの「[10]」は「[]」と書いてはいけません。

課題 119

空所 [い] から見ていきましょう。この式が真になるとただちに（関数末尾の −1〈＝見つからなかった〉でなく）i を返しますから，これは「見つかったとき」です。ですから，「a[i] == x」を補充します。

空所 [あ] は，**プログラム 27**（本文 160 ページ）では配列の終端までの繰り返しと見つかったかどうかの判定を行っていました。ですがここでは，見つかったかどうかは空所 [い] で判定していますから，配列の終端までの繰り返しだけでよいことになります。配列の要素数が引数 size で与えられていることに注意して，「i < size」を補充します。

ここで，x が見つからずに繰り返しが配列の終端まで達して終了すると −1 が返されます。これは，「見つからない」を表すということですから，期待どおりの結果です。

課題 120

プログラムは，問題なく実行できるでしょう。しかし，ipow() 関数内で，a，b の値を出力させようとすると，「変数が宣言されていない」といったコンパイルエラーが報告されるはずです。

課題 121

8.13.2 の プ ロ グ ラ ム 47（本文 227 ページ）の，sumSubAry() 関数は，渡された配列 a[] について，a[s] から s[e] までの合計を返します。ここで作成したい関数 sumAry() は，全要素の合計ですから，要素番号 0 から size − 1 の合計を返すことになります。よって，要求されている関数は，次のように書けます。

```
int sumAry(int x[], int size) {
  return sumSubAry(x, 0, size - 1);
}
```

課題 122

定法の勝ち抜き式で最小値を求めましょう。ただし，最小値ではなく「最小値を持つ要素番号」を返す点に注意してください。まず，x[s] を暫定最小値として，x[s + 1] 以降の要素に対して，これまでの暫定最小値を下回る値が現れたら，暫定最小値が現れた要素番号を更新していきます。

```
int minIdx(int x[], int s, int e) {
  int i, m;

  m = s; /* 暫定最小値は x[s] にあるとする */
  for (i = s + 1; i <= e; i++) {
    if (x[m] > x[i]) {
      m = i;
    }
  }
  return m;
}
```

課題 123

10 個（記号定数 N で表しておきます）の値を入力して，昇順に並べる場合で考えてみましょう。minIdx() も含めた完全な形のプログラムで示しておきます。**7.3**（本文 168 ページ）と**課題 92**（本文 173 ページ，解答は 26 ページ）

も参考にしてください。

```
1:  #include <stdio.h>
2:
3:  #define N 10
4:
5:  int minIdx(int x[], int s, int e) {
6:    int i, m;
7:
8:    m = s; /* 暫定最小値はx[s]にあるとする */
9:    for (i = s + 1; i <= e; i++) {
10:     if (x[m] > x[i]) {
11:       m = i;
12:     }
13:   }
14:   return m;
15: }
16:
17:
18: int main(void) {
19:   int a[N], i, m, t;
20:
21:   /* N個の値を入力する */
22:   for (i = 0; i < N; i++) {
23:     scanf("%d", &a[i]);
24:   }
25:   /* 選択ソート */
26:   for (i = 0; i < N - 1; i++) {
27:     m = minIdx(a, i, N - 1);
28:     t = a[i]; a[i] = a[m]; a[m] = t;
29:   }
30:   /* 整列結果を出力 */
31:   for (i = 0; i < N; i++) {
32:     printf("%d ", a[i]);
33:   }
34:   printf("¥n");
35:   return 0;
36: }
```

降順に並べる場合は，昇順に並べるときの操作を，配列の末尾から行えばよいでしょう（配列の末尾から昇順に並べれば，先頭からは降順になります）。すなわち，上記のプログラムの26～29行を，次のように書き換えます。

```
for (i = N - 1; i >= 1; i--) {
  m = minIdx(a, 0, i);
  t = a[i]; a[i] = a[m]; a[m] = t;
}
```

なお，課題の指示には反しますが，範囲内で**最小値**を持つ要素番号を返すminIdx()ではなく，**最大値**を持つ要素番号を返す関数を作成すれば，先頭から処理して降順に並べることができます。

課題 124

まず，求められている関数accum()を作成しましょう。累計は，accum()内のstaticな変数に保持し，それに引数を加えて返せばよいでしょう。

```
int accum(int x) {
  static int sum = 0;

  sum += x;
  return sum;
}
```

この関数のsumは，関数の実行が終了しても消滅せず，プログラム開始時に一度だけ0に初期化されます。

main()関数も含めて，プログラムを示します。

```
1:  #include <stdio.h>
2:
3:  int accum(int x) {
4:    static int sum = 0;
5:
6:    sum += x;
7:    return sum;
8:  }
9:
10:
11: int main(void) {
12:   int a;
13:
14:   scanf("%d", &a);
15:   while (a > 0) {
16:     accum(a);
17:     scanf("%d", &a);
18:   }
19:   printf("合計：%d¥n", accum(0));
20:   return 0;
21: }
```

main()関数内の変数は，入力された数を記憶する変数aのみで，合計を保持する変数はありません。

16行でaccum()を呼び出し，accum()内の変数sumに値を累積させています。この関数はその時点までの累計を返しますが，ここでは戻り値を捨てています。

0以下の数が入力されるとwhileループを抜けます。

19 行で合計を出力しますが，accum(0) を呼び出すことでこれまでの累計を得ています（これまでの合計に 0 を加えて返すのですから，以前の値と同じです）。

付 録

課題 125

6.3（本文 130 ページ）のプログラムをコンパイルして hensa.exe が得られているとすると，

```
hensa.exe > result.txt
```

と実行させます。

ファイルへのリダイレクトをしていないときと同様に 10 個の値を入力すると，画面への出力なく終了するでしょう。かわりに，result.txt ができていますから，これをテキスト・エディタで開いてみてください。結果が書き込まれているはずです。

課題 126

テキスト・エディタで，たとえば，40 個の値を改行で区切って記述して，score1.txt として保存してください。これを，**6.4 のプログラム 23**（本文 138 ページ）に入力します。すなわちプログラムをコンパイルして，histgram.exe が得られているとすると，

```
histgram.exe < score1.txt
```

と実行させます。これで，キーボードから入力することなく，score1.txt の内容が入力されたものとして出力が得られるでしょう。

さらに，結果を graph.txt に書き出すには，次のように実行させます。

```
histgram.exe < score1.txt > graph.txt
```

課題 127

score2.txt が用意されていれば，**課題 126** での graph.txt への書き出しに続けて，次のように実行します。

```
histgram.exe < score2.txt >> graph.txt
```

出力リダイレクトに，「>」の代わりに「>>」を使っている点に注意してください。

課題 128

条件を素朴にコーディングすれば，次のようになるでしょう。

```
#include <stdio.h>

int main(void) {
  int y;

  scanf("%d", &y);
  if (y % 4 == 0) {
    if (y % 100 == 0) {
      if (y % 400 == 0) {
        printf("うるう年です¥n");
      } else {
        printf("うるう年ではありません¥n");
      }
    } else {
      printf("うるう年です¥n");
    }
  } else {
    printf("うるう年ではありません¥n");
  }
  return 0;
}
```

しかし，条件を次のように読み替えれば，プログラムは少し単純になります。

- 400 で割り切りれる年はうるう年。
- 400 で割り切れない，100 で割り切れる年はうるう年でない。
- 100 で割り切れない，4 で割り切れる年はうるう年。

これは次のようコーディングできます。

```
#include <stdio.h>

int main(void) {
  int y;

  scanf("%d", &y);
  if (y % 400 == 0) {
    printf("うるう年です¥n");
  } else if (y % 100 == 0) {
    printf("うるう年ではありません¥n");
  } else if (y % 4 == 0) {
    printf("うるう年です¥n");
  } else {
    printf("うるう年ではありません¥n");
  }
  return 0;
}
```

さらに条件を整理すれば，うるう年は，「400 で割り切れるか，または，100 で割り切れない 4 の倍数の年」です。

ですから，|| および && 演算子を使えば，次のようにまとめられます。

```
#include <stdio.h>

int main(void) {
  int y;

  scanf("%d", &y);
  if (y % 400 == 0 ||
   y % 100 != 0 && y % 4 == 0) {
    printf("うるう年です¥n");
  } else {
    printf("うるう年ではありません¥n");
  }
  return 0;
}
```

課題 129

西暦 2019 年が令和元年ですから，2019 年以降について，西暦から 2018 を減じたものが和暦年になります。同様に，西暦 1989 年が平成元年ですから，1989 年から 2018 年については，西暦から 1988 を減じれば和暦年になります。また，西暦 1926 年が昭和元年ですから，1926 年から 1988 年については，西暦から 1925 を減じたものが和暦年です。このことから，プログラムは次のように作成できます。

```
#include <stdio.h>

int main(void) {
  int y;

  scanf("%d", &y);
  if (y >= 2019) {
    printf("令和%d年¥n", y - 2018);
  } else if (y >= 1989) {
    printf("平成%d年¥n", y - 1988);
  } else {
    printf("昭和%d年¥n", y - 1925);
  }
  return 0;
}
```

課題 130

入力部分は，「正の値である間繰り返し」です。ですから，この部分は，a に読み込むものとして，定法に従って，

```
scanf("%d", &a);
while (a > 0) {
  ……
  scanf("%d", &a);
}
```

のような形式になるでしょう。

そして，求める値が個数・最大値・平均です。平均は，合計と個数から求められますから，入力の繰り返しの中で，個数・合計・最大値を求めていけばよいことになります。

入力が完了したら，個数・最大値・平均を出力して終了します。個数が 0 のときは特別扱いすることに注意してください。

```
#include <stdio.h>

int main(void) {
  int a, s, n, m;

  s = 0; n = 0; m = 0;
  /* 整数を入力して個数・合計・最大値を求める */
  scanf("%d", &a);
  while (a > 0) {
    n++;
    s += a;
    if (a > m) {
      m = a;
    }
    scanf("%d", &a);
  }
  /* 個数・最大値・平均を出力して終了 */
  if (n == 0) {
    printf("データがありません¥n");
  } else {
    printf("個数：%d¥n", n);
    printf("最大値：%d¥n", m);
    printf("平均：%f¥n", (double)s / n);
  }
  return 0;
}
```

課題 131

与えられた配列｛17, 32, 5, 8, 7, 56, 88, 77, 29, 35｝に，「27」を追加すると大きいほうから何番目になるか考えます。これは，6 番目になります。降順（または昇順）に並び替えられていればひと目でわかりますが，そうでなくても，自分より大きい数がいくつあるか数えればよいのです。

この場合，27 より大きいのは 32，56，88，77，35 の 5 つですから，27 が 6 番目に大きい値に相当します。

この方針でプログラムします。何番目に大きいかは変数 rank に求められますが，自分より大きい値が 1 つもない場合は「いちばん大きい」ことに注意します。

```
#include <stdio.h>

#define N 10

int main(void) {
  int a[N] = { 17, 32, 5, 8, 7, 56, 88,
   77, 29, 35 };
  int x, i, rank;

  scanf("%d", &x);
  rank = 1;
  for (i = 0; i < N; i++) {
    if (a[i] > x) {
      rank++;
    }
  }
  printf("%d¥n", rank);
  return 0;
}
```

課題 132

基本的には「最小値を求めるプログラム」ですが，「差の絶対値が最小」のものを求める点が異なっています。また，条件として，「最も近い値が複数ある場合は小さい方を出力」があります。

数の近さを調べるのに絶対値を何度も使いますから，関数として作っておきましょう。なお，整数の絶対値を求める関数は，自作しなくても，「#include <stdlib.h>」で読み込まれる関数 int abs(int x) として利用できます。

プログラムは，次のようになります。a[0] を仮の最も近い値として，a[1]～a[N - 1] に対してそれよりも近い値を探しています。mabs が仮の最も近い値に対する差の絶対値，near が仮の最も近い値を保持しています。入力した値と a[] の着目している値との差の絶対値を cabs に求めておき，比較しています。仮の最も近い値が更新される条件は，「cabs < mabs」はもちろんですが，「cabs == mabs && a[i] < near」（最も近い値が複数ある場合は小さい方）でもよいことに注意します。

```
#include <stdio.h>

#define N 10

int abs(int x) {
  if (x > 0) {
    return x;
  }
  return -x;
}


int main(void) {
  int a[N] = { 17, 32, 5, 8, 7, 56, 88,
   77, 29, 35 };
  int cabs, mabs, near, i, x;

  scanf("%d", &x);
  near = a[0]; mabs = abs(x - a[0]);
  for (i = 1; i < N; i++) {
    cabs = abs(x - a[i]);
    if (cabs < mabs ||
     cabs == mabs && a[i] < near) {
      mabs = cabs; near = a[i];
    }
  }
  printf("%d¥n", near);
  return 0;
}
```

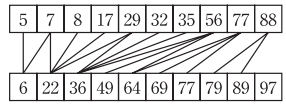
課題 133

まずは最も単純な方法で考えましょう。a[] のそれぞれの値を，b[] から探索します。探索にあたって，b[] は整列済みで，**7.2**（本文 163 ページ）の二分探索が使えますが，プログラムを簡単化するために **7.1**（本文 158 ページ）の線形探索を使うことにします。次に示すようなプログラムになるでしょう（ただし，a[]，b[] の初期化値は省略しています）。

```
#include <stdio.h>

#define N 20

int main(void) {
  int a[N] = { …… };
  int b[N] = { …… };
```

```
    int i, j;

    /* a[]のすべての要素について調べる */
    for (i = 0; i < N; i++) {
      /* a[i]に等しい値をb[]から探す */
      for (j = 0; j < N && a[i] != b[j];
           j++)
        ;
      if (j != N) {
        printf("%d¥n", a[i]);
      }
    }
    return 0;
}
```

一方で，a[]もb[]も昇順に整列済みであることを使えば，次のような手順が考えられます。



たとえば，a[i]が31，b[j]が41だったとします。これは一致していません。では，これらの大きいほう，41はa[]のどこに存在しうるかというと，a[i + 1]以降です。同様に考えていけば，図のように比較していけば，同じ値があるか確かめられるということです（線を引いた組み合わせを比較するということです）。

この手順に基づけば，次のようにプログラムできます。

```
#include <stdio.h>

#define N 20

int main(void) {
  int a[N] = { …… };
  int b[N] = { …… };
  int i, j;

  i = 0; j = 0;
  while (i < N && j < N) {
    if (a[i] == b[j]) {
      printf("%d¥n", a[i]);
      i++; j++;
    } else if (a[i] > b[j]) {
      j++;
    } else {
      i++;
    }
  }
  return 0;
}
```

なお，課題の指示には関係がありませんが，上掲の2つのプログラムは，a[]とb[]に同じ値があって，その値が配列に複数個あったときの挙動が異なります。

課題 134

入力された値は覚えておかなければなりませんから，配列変数を使います。構成比を求めるにあたり，合計も求めておく必要があります。

プログラムは次のように作成されます。x[]が入力されたデータ，nが入力されたデータの個数，sumが合計です。

```
#include <stdio.h>

#define NMAX 100

int main(void) {
  int x[NMAX + 1], sum, n, i;

  /* 値の入力 */
  sum = 0; n = 0;
  scanf("%d", &x[n]);
  while (x[n] != 0) {
    sum += x[n];
    n++;
    scanf("%d", &x[n]);
  }
  /* 入力データと構成比を出力 */
  for (i = 0; i < n; i++) {
    printf("#%2d: %d %f%%¥n", i, x[i],
      (double)x[i] / sum);
  }
  return 0;
}
```

データをファイルから入力させて実行するには，**B.5**（本文248ページ）の入力リダイレクトを使ってください。

課題 135

問題の指示どおり，w[]を｛50, 100, 150, 250, 500, 1000｝，f[]を｛120, 140, 210, 250, 390, 580｝としましょう。

たとえば，郵便物の重量（weightとします）が140 g

であるとき，この値はw[0]，w[1]より大きく，w[2]以下であることがわかります。そしてこのときの料金は「150 g まで」の 210 円（f[2]）です。つまり，料金は，weight <= w[i] となる最小の i に対して，f[i] となります。そのような i が存在しない場合，1 kg（1000 g）を超えていますから，規格外，となります。

以上の議論からプログラムを作成すると，次のようになります。

```
#include <stdio.h>

#define CLASS 6

int main(void) {
  int w[CLASS] = { 50, 100, 150, 250,
   500, 1000 };
  int f[CLASS] = { 120, 140, 210, 250,
   390, 580 };
  int weight, i;

  scanf("%d", &weight);
  for (i = 0; i < CLASS
   && w[i] < weight; i++)
    ;
  if (i == CLASS) {
    printf("規格外です。¥n");
  } else {
    printf("%d円です¥n", f[i]);
  }
  return 0;
}
```

課題 136

2 次元配列を用いたプログラムを考えます。nに対して ${}_nC_n$まで求めますから，記号定数 N に対して c[N][N] まで使えるように配列変数 c[][] を用意します。そして，繰り返しを用いて組み合わせの数を求めます。${}_nC_r$について，rは 0 からnまでであることに注意します。そして，本文中の関係式を用いますが，rが 0 またはnのときに特別扱いすることに注意してください。

```
#include <stdio.h>

#define N 5

int main(void) {
  int c[N + 1][N + 1], n, r;

  for (n = 0; n <= N; n++) {
    for (r = 0; r <= n; r++) {
      if (r == 0 || r == n) {
        c[n][r] = 1;
      } else {
        c[n][r] = c[n - 1][r - 1]
         + c[n - 1][r];
      }
      printf("%d ", c[n][r]);
    }
    printf("¥n");
  }
  return 0;
}
```

一方，1 次元配列を使って求める場合，配列の要素は N に対して N + 1 個用意します。ここで注目するのは，rが 0 またはn以外のとき，${}_nC_r$を求めるのに必要なのは ${}_{n-1}C_{r-1}$と ${}_{n-1}C_r$だけであるということです。すなわち，${}_nC_r$が求められたら，${}_{n-1}C_k$について$k \geqq r$であるものは必要なくなるということです。このことから，あるnについてrが大きいほうから${}_nC_r$を求めていけば，1 次元配列だけで正しく組み合わせの数を求められます。

```
#include <stdio.h>

#define N 5

int main(void) {
  int c[N + 1], n, r;

  for (n = 0; n <= N; n++) {
    c[0] = 1; c[n] = 1;
    for (r = n - 1; r >= 1; r--) {
      c[r] += c[r - 1];
    }
    for (r = 0; r <= n; r++) {
      printf("%d ", c[r]);
    }
    printf("¥n");
  }
  return 0;
}
```

なお，${}_nC_r$ に対して，${}_nC_r = {}_nC_{n-r}$ の関係がありますから，上記のプログラムで r が大きいものから組み合わせの数を求めているのは，r が小さいものから組み合わせの数を求めていることと同等です。ですから，r が大きいほうから求めて，r が小さいほうから出力させるのではなく，r が大きいほうから求めてそのまま出力させる，次のようなプログラムでも，結果は同じになります。

```
1:  #include <stdio.h>
2:
3:  #define N 5
4:
5:  int main(void) {
6:    int c[N + 1], n, r;
7:
8:    for (n = 0; n <= N; n++) {
9:      for (r = n; r >= 0; r--) {
10:       if (r == 0 || r == n) {
11:         c[r] = 1;
12:       } else {
13:         c[r] += c[r - 1];
14:       }
15:       printf("%d ", c[r]);
16:     }
17:     printf("¥n");
18:   }
19:   return 0;
20: }
```

課題 137

まず，「a[s] から a[e - 1] の要素を 1 つずつ要素番号が大きい要素に移す」ことから考えます。単純に考えれば，int 型の変数 i を用意して次のように書けそうですが，これではうまくいきません。

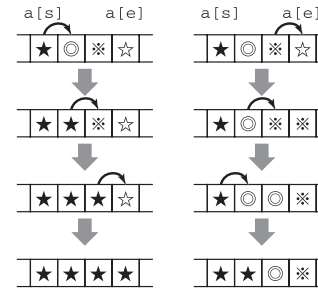
```
for (i = s; i < e; i++) {
  a[i + 1] = a[i];
}
```

なぜなら，この手順では，

- a[s] を a[s + 1] にコピー
- a[s + 1] を a[s + 2] にコピー
- ……
- s[e - 1] を a[e] にコピー

と進んでいって，a[s] から a[e] まですべて，もとの a[s] の値になってしまうからです（下図左）。解決策は，上記の手順を逆向きに（後ろから）行うことです（下図右）。

```
for (i = e - 1; i >= s; i--) {
  a[i + 1] = a[i];
}
```



これに，最初に上書きされてしまう a[e] の値を保管して，最後に a[s] に代入する処理を加えれば，関数は完成です。

```
1:  void rotateR(int a[], int s, int e) {
2:    int i, tmp;
3:
4:    tmp = a[e];
5:    for (i = e - 1; i >= s; i--) {
6:      a[i + 1] = a[i];
7:    }
8:    a[s] = tmp;
9:  }
```

課題 138

プログラム 31（本文 178 ページ）に基づいて考えましょう。

挿入ソートの核となる処理は，未整列の部分の先頭を，整列済み部分の適切な位置に差し込み，以降を 1 つずつ後ろにずらすことです。これは，前もって「適切な位置」を見つけて，その位置と未整列部分の先頭について，**課題 137** の rotateR() を実行することと同じです。

ですから，rotateR() が定義されているものとして，**プログラム 31**（本文 178 ページ）の 12～19 行を，次のように書き換えます。未整列部分の先頭が a[i]，挿入位置は a[j + 1] であることを確認してください。

```
/* 挿入ソート */
for (i = 1; i < N; i++) {
  for (j = i - 1; j >= 0
    && a[j] > a[i]; j--)
    ;
  rotateR(a, j + 1, i);
}
```

課題 139

正の整数 n について，n % 10 はその数の下 1 桁の数字を与えます。また，10 で割れば，一の位がなくなって，十の位が一の位に，百の位が十の位に，……，と桁が 1 つずつずれていきます。これらを繰り返していけば，下から順に各桁の数字を取り出していくことができます。これらの和を求めて返せば，sumFig() ができあがります。

```
1:  int sumFig(int n) {
2:    int sum = 0;
3:
4:    while (n > 0) {
5:      sum += n % 10;
6:      n /= 10;
7:    }
8:    return sum;
9:  }
```

課題 140

「9 の倍数」と「各桁の数字の和が 9 の倍数になる数」が同値だということを確かめるのに，たとえば，

- 上記のいずれかの条件の数を出力
- 9 の倍数であればそれを付記
- 各桁の数字の和が 9 の倍数であればそれを付記

していき，出力された数すべてに「9 の倍数」「各桁の数字の和が 9 の倍数」とマークされていることを確認すればよいでしょう。

```
1:  #include <stdio.h>
2:
3:  #define LIMIT 20000
4:
5:  /* 各桁の数字の和を求めて返す */
6:  int sumFig(int n) {
7:    int sum = 0;
8:
9:    while (n > 0) {
10:     sum += n % 10;
11:     n /= 10;
12:   }
13:   return sum;
14: }
15:
16:
17: int main(void) {
18:   int i;
19:
20:   for (i = 1; i <= LIMIT; i++) {
21:     if (i % 9 == 0 || sumFig(i) % 9 == 0) {
22:       printf("%d ", i);
23:       if (i % 9 == 0) {
24:         printf("9 の倍数 ");
25:       }
26:       if (sumFig(i) % 9 == 0) {
27:         printf(" 各桁の数字の和が 9 の倍数 ");
28:       }
29:       printf("¥n");
30:     }
31:   }
32:   return 0;
33: }
```

このプログラムでは，結果の確認は人間が行います。一方で，

- 9 の倍数であり，各桁の数字の和が 9 の倍数
- 9 の倍数であるが，各桁の数字の和が 9 の倍数でない
- 9 の倍数でないが，各桁の数字の和が 9 の倍数

である自然数をカウントして，後ろ 2 つの条件を満たす数が 0 であることを確認すれば，人間が結果を確認する手間は省けるでしょう。

課題 141

本問は，**課題 31**（本文 55 ページ，解答は 9 ページ）の手順をプログラムにするというものです。

これは，x の y 乗において，x の 2 乗を計算しておき，

- y が奇数のときは，x に「x の 2 乗」を何回かかける
- y が偶数のときは，1 に「x の 2 乗」を何回かかける

という方法で求めるのでした。

コメントを頼りに空所を補充していきましょう。

空所 [あ] は，上記の「y が奇数のときは，**x に**……」と，「y が偶数のときは，**1 に**……」とで分岐している条件です。成立したとき，1 であった p の値を x にし，かけ合わせる回数である y を 1 減じています。これは，y が奇数のときの処理ですから，「**y % 2 == 1**」を補充します。

空所 [い] は，「x の 2 乗」を計算しています。ですから，「**x * x**」を補充します。

空所 [う]，[え] は，p に x の 2 乗をかける繰り返しに関してです。かける回数は y の半分ですが，回数をカウントするための変数を用意していませんから，y の値を使ってカウントします。x の 2 乗をかけるには，y

が2以上である必要がありますから，空所［う］は「**y >= 2**」です。14行までで y は偶数になりますから，「y > 0」でもよいでしょう。空所［え］については，16行が1回実行されるたびに「x を2回かけた」ことになるのですから，y を2ずつ減じます。ですから「**y -= 2**」を補充します。

課題 142

y が0のとき，8行の if 文の条件は成立せず，p は1のままです。そして，15行からの for ループ「for (; y >= 2; y -= 2) { …… }」は初回から継続条件が偽ですから，1回も実行されません。ですから，p は1のまま，18行で出力されます。これは，「x の0乗」として正しい結果です。

かけ算の回数は，**課題 32**（本文55ページ，解答は10ページ）で数えています。**課題 141** のプログラムにおいても同じです。具体的な回数は**課題 32** の解答を確認してください。プログラム中でかけ算の回数をカウントしてもよいでしょう（13行でも1回かけ算をしていることに注意してください）。**課題 141** のプログラムは，「x の2乗を計算する」ことと引き換えに，繰り返しでのかけ算の回数を半分に抑えます。ですから，x の y 乗において，単純に「1に x を y 回かける」アルゴリズムと比較すると，かけ算の回数の差は，y が大きくなるほど広がっていきます。

課題 143

プログラムを，問題の指示と照らし合わせながら見てみましょう。

変数 n は，素因数分解したい正の整数です。［う］の条件が満たされている間，p が出力され，n は p で割られます。そして，満たされないと，p は1増やされます。このことから，p は「これから割ってみようとする整数」であることがわかります。

空所［あ］には，割ってみる数 p の初期値を補充します。素因数分解は，最小の素数（2）から割ってみることを始めますから，補充するのは「**2**」です。

空所［い］が満たされなくなると，繰り返しが終了します。これは，「1になったら素因数分解が完了する」に対応していて，「**n > 1**」または「n != 1」とすればよいでしょう。

空所［う］は，これが満たされている間 n は p で割られるのですから，「n が p で割り切れる間」すなわち「**n % p == 0**」を補充します。

課題 144

書き換えた部分について，［え］の条件が満たされると p が出力されています。p の出力は，「n が p で割り切れたとき」に起こりますから，［え］には「**n % p == 0**」が補充されます。出力後に n を p で割りますから，空所［お］には「**n /= p**」を補充します。

さて，n が p で割り切れないときは，p を「次の整数」にして，n を割り切ることができるか調べます。したがって，空所［か］には「**p++**」を補充すればよいでしょう。

課題 143 では，この部分は while ループで記述されていますが，本課題のように書き換えても結果は同じになります。

課題 145

空所［あ］は，「配列の全要素を0にする」という定法を補充します。a[N] まで使うことに注意して，次のようなコードを補充すればよいでしょう。

```
for (i = 0; i <= N; i++) {
  a[i] = 0;
}
```

11〜19行の繰り返しで，整数の列を調べます。整数の列は2から N まで調べるのですから，空所［い］には「**2**」を補充すればよいでしょう。

空所［う］で，「素数が見つかった」というのは，着目している数（i）がまだ消されていないときです。i が消されたかどうかは a[i] の値（0が消されていない，1が消された）によりますから，ここには「**a[i] == 0**」を補充します。

空所［え］および［お］は，「i の倍数を消す（i の倍数 j について，a[j] を1にする）」処理です。i の倍数は，i，i + i，i + i + i，……，ですから，j を i から始めて i ずつ増やしていけばよいのです。すなわち，［え］は「**i**」，［お］は「**j += i**」を補充すればよいでしょう。または，i の倍数を消していくプロセスで，i 自身はもう調べてしまっていますから，空所［え］は「i の2倍」として「i + i」から始めてもかまいません。

基礎 C 言語［入門編］――コンピュータの基本から理解するプログラミング 付録 PDF

「課題の解答と解説」

牛田啓太　著
発行　株式会社技術評論社