

C 言語によるデータ処理演習

C 言語を使って、データ処理を行ってみましょう。次の 5 つのプログラムを作ります。

演習 1	平均の計算
演習 2	標準偏差の計算
演習 3	相関係数の計算
演習 4	移動平均の計算
演習 5	出現文字の数え上げ

本編では学習していない項目や、プログラミングの技法なども解説します。

ファイルへの入出力

これまで、データはキーボードから入力して、ディスプレイに出力していました。たくさんのデータを手で入力するのは大変なので、ファイルに記録されたデータを読み取ることにします。また、計算した値を後で使うためには、ファイルに記録しておく必要があります。そこで、ファイルへの入出力について簡単に説明します。

fopen() と fclose()

ファイルからデータを読み取る、またはファイルにデータを書き込むためには、ファイルを開く必要があります。そして、ファイルに対するデータの読み書きが終わったら、ファイルを閉じます。ファイルを開くには、**fopen() 関数**を、ファイルを閉じるには **fclose() 関数**を使います。

fopen() 関数の仕様を、表 1 に示します。第 1 引数に指定した名前を持つファイルを、第 2 引数に指定したモードで開きます。第 2 引数には、開いたファイルからデータを読み取りたいときは読み取りモードを表す **"r"** を、開いたファイルにデータを書き込みたいときは書き込みモードを表す **"w"** を指定します。**fopen()** 関数は、ファイルを開くことに成功したら FILE へのポインタ^{*1}を返します。ここでは、それを**ファイルポインタ**と呼びます。

fopen() 関数が返すファイルポインタを通して、ファイルからの読み取り (読み込みモードで開いた場合) と、ファイルへの書き込み (書き込みモードで開いた場合) が行えます。

^{*1} 一般的な処理系では FILE は構造体です。

表 1 fopen() 関数

書式	<code>fopen(path, mode)</code>
引数	ファイル名 <i>path</i> (<code>const char *</code> 型) とモード <i>mode</i> (<code>const char *</code> 型)
処理	<i>mode</i> が "r" なら読み取りモードで、"w" なら書き込みモードで、ファイル <i>path</i> を開く。
戻り値	処理に成功したらファイルポインタ、失敗したら NULL (<code>FILE *</code> 型)
必要ヘッダ	<stdio.h>

`fclose()` 関数の仕様を、表 2 に示します。`fclose()` 関数の引数は 1 つで、`fopen()` 関数の戻り値として得られたファイルポインタ (`FILE *`型) を指定します。

表 2 fclose() 関数

書式	<code>fclose(s)</code>
引数	ファイルポインタ <i>s</i> (<code>FILE *</code> 型)
処理	<i>s</i> に対応するファイルを閉じる。
必要ヘッダ	<stdio.h>

次のソースコードは、ファイルを読み取りモードで開いて閉じる例です。

```

1 FILE *fp;
2
3 /* ファイルを開く */
4 fp = fopen("in.txt", "r");
5 if (fp == NULL)
6 {
7     printf("ファイルを開けませんでした。\n");
8     return 0;
9 }
10
11 /* ここでファイルからデータを読み取る */
12
13
14 /* ファイルを閉じる */
15 fclose(fp);

```

1 行目は、FILE へのポインタ変数 `fp` の宣言です。

4 行目で、`fopen()` 関数を使って、ファイルを開きます。第 1 引数はファイル名で、この例では `"in.txt"` です。このファイルは、カレントディレクトリに置かれている必要があります。第 2 引数には、読み取りモードを表す `"r"` を指定します。`fp` には、`fopen()` 関数から返されるファイルポインタが代入されます。

指定されたファイルが見つからないなどの理由で `fopen()` 関数が失敗したときは、`NULL` を返します。5 行目から 9 行目で、`fopen()` 関数の戻り値が `NULL` であった場合、エラーメッセージを表示して、プログラムを終了します。

15 行目で、`fclose()` 関数を使ってファイルを閉じます。

`fscanf()` と `fprintf()`

ファイルポインタを通してデータを読み取るには `fscanf()` 関数が、ファイルポインタを通してデータを書き込むには `fprintf()` 関数が使えます。

`fscanf()` 関数の第 1 引数にはファイルポインタを指定し、第 2 引数以降は、`scanf()` 関数の第 1 引数以降と同じです。たとえば、`n` が `int` 型の変数なら、

```
fscanf(fp, "%d", &n)
```

は、ファイルポインタ `fp` を通して整数を 1 つ読み取り、変数 `n` に格納します。

これまで、`scanf()` 関数では一度に 1 つのデータしか読み取りませんでした。が、`scanf()` 関数と `fscanf()` 関数は複数の値を一度に読み取ることもできます。スペース、タブ、改行などで区切られた複数の値を `scanf()` 関数で一度に読み取るには、第 1 引数に複数の変換指定を含めます。そして第 1 引数に記載した順に、変換指定に対応する変数へのポインタを、第 2 引数以降に指定します。たとえば、次のソースコードでは、キーボードから文字列、整数、小数を読み取って、それぞれ `name`、`age`、`height` に格納します。

```
1 char name[20];
2 int age;
3 double height;
4
5 scanf("%s%d%lf", name, &age, &height);
```

`fscanf()` 関数も同様です。ファイルポインタ `fp` から読み取る場合は、次のように記述します。

```
1 fscanf(fp, "%s%d%lf", name, &age, &height);
```

`fscanf()` 関数の戻り値は、データを格納した個数です*²。読み取りに失敗したり、ファイルの末尾に達した上でさらに読み取ろうとした場合は、変換指定の個数とは異なる値が返ることになり

*² `scanf()` 関数も同じです。

ます。この戻り値を使えば、次のソースコードのように、データ数のわからないファイルからすべてのデータを読み取ることができます。

```
1  /*
2   * ファイルから name、age、height を読み取れば、処理を実行する。
3   * ファイルの末尾に達するかエラーが起これば、反復を終了する。
4   */
5  while (fscanf(fp, "%s%d%lf", name, &age, &height) == 3)
6  {
7      /* 処理 */
8  }
```

fprintf() 関数は、ファイルにデータを書き込むことができます。fprintf() 関数の第 1 引数にはファイルポインタを指定し、第 2 引数以降は、printf() 関数の第 1 引数以降と同じです。次のソースコードは、カレントディレクトリの out.txt という名前のファイルにデータを書き込みます。

```
1  int n = 10;
2  FILE *fp;
3
4  fp = fopen("out.txt", "w");
5  if (fp == NULL)
6  {
7      printf("ファイルを開けませんでした。\n");
8      return 0;
9  }
10
11  fprintf(fp, "nの値は%dです.\n", n);
12
13  fclose(fp);
```

4 行目で、書き込みモードでファイル out.txt を開きます。このとき、カレントディレクトリに out.txt がなければ作成されます。すでに out.txt があれば、元々 out.txt に記録されていた内容は消えてしまいます。このプログラムを実行すると、“n の値は 10 です。”と改行が記録された out.txt ができます。

fscanf()、fprintf() に必要なヘッダは、いずれも<stdio.h>です。

平均と標準偏差

演習 1 平均の計算

data1.txt には、1 行に 1 人分の身長と体重 (いずれも小数) が、タブ区切りで記録されています。計 30 人分のデータがあります。身長と体重の平均を求めて、表示してください^a。ex180.c は途中まで作ったプログラムです。コメント部分に処理を追加して、正しく動作するプログラムを作ってください。

実行例

身長の平均: 165.4

体重の平均: 62.1

正解例: ex180-2.c

^a 演習 1 と演習 2 の正解例では、数値を小数点以下 1 桁まで表示することになります。

演習 1 の正解例の 17 行目から 22 行目の動作は次のとおりです。while 文の制御式の中で、fscanf() を使ってファイルから身長と体重を読み取ります。fscanf() の戻り値が 2 なら、つまり身長と体重の 2 つのデータが読み取れたら、while 文の制御式が真になり、ループ本体が実行されます。ループ本体では、読み取った身長 (height) を sum_height に、体重 (weight) を sum_weight に加え、num をインクリメントします。fscanf() の戻り値が 2 でないなら、while 文の制御式が偽になり、while 文の実行が終了します。sum_height、sum_weight、num の初期値はいずれも 0 なので、while 文の実行が終了した時点で、sum_height には読み取った身長の合計が、sum_weight には読み取った体重の合計が、num には人数 (身長と体重の組の数) が入っていることになります。

演習 1 の正解例 (ex180-2.c)

```
17     while (fscanf(fp, "%lf%lf", &height, &weight) == 2)
18     {
19         sum_height += height;
20         sum_weight += weight;
21         ++num;
22     }
```

演習 1 では、ファイルから読み取ったデータをその場で処理しました。処理内容によっては、すべてのデータを配列などに記録しておく必要があります。この演習では、配列に記録する場合、十

分な要素数を指定して配列を宣言することになります*³。また、配列の要素数を超えるデータを読み取る可能性は考えないことにします*⁴。

データが $x_1, x_2, \dots, x_n$ の n 個で、その平均が $\bar{x}$ のとき、標準偏差は、次のように計算できます。

$$\sqrt{\frac{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \dots + (x_n - \bar{x})^2}{n}} \quad (1)$$

つまり、標準偏差は、((((平均からの差) の 2 乗) の平均) の平方根) です。

演習 2 標準偏差の計算

data1.txt に記録された身長標準偏差を求めるプログラムを作ってください。ファイルに記録されているデータは多くても 100 人分として、要素数 100 の double 型の配列を作って使ってください。

実行例

身長の標準偏差: 10.1

正解例: ex181.c

演習 2 の正解例 (ex181.c) では、while 文で data1.txt からデータを読み取りながら、配列 height に身長を格納するとともに、平均の計算に必要な身長の合計 (sum1) と人数 (num) を求めます。ファイルを読み終わると、身長の平均 (ave) を計算します。そして、height に格納した身長を先頭から取り出しながら標準偏差を計算します。変数 sum2 には、式 (1) の $(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \dots + (x_n - \bar{x})^2$ に相当する値を格納します。

この正解例では、data1.txt に記録された身長を変数 h に、体重を変数 w に格納していますが、その後の処理には h しか使っていません。変換指定の % の後ろに * を付けると、データを読み取りますが、変数には格納しません。

```
while (fscanf(fp, "%lf%lf", &h, &w) == 2)
```

の代わりに

```
while (fscanf(fp, "%lf%*lf", &h) == 1)
```

とすれば、変数 w は不要です*⁵。

標準偏差は、次の式のように、(((2 乗の平均) - (平均の 2 乗)) の平方根) で計算することもで

*³ 読み取ったデータ数に応じて配列の要素数を拡張する方法もあります。

*⁴ 実用するプログラムでは、ファイルに想定した形式でデータが入っていないかたり、想定を超える数のデータが入っていたりする場合に備えて、エラー処理をする必要があります。

*⁵ この場合、%*lf の 1 に意味がなくなり、コンパイラやその設定によっては警告が表示されます。%*f にすれば警告が表示されなくなります。

きます。

$$\sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n} - \left(\frac{x_1 + x_2 + \cdots + x_n}{n}\right)^2} \quad (2)$$

この式を使えば、読み取った身長を配列に記録しておく必要はありません。この式を使うように改変したソースコードが、ex181-2.c です。

相関係数

1つの対象から2つの項目に関して値が得られるとき、項目間の関連の強さを表す相関係数が計算できます。たとえば、複数の学生の国語の得点と算数の得点から、国語の得点と算数の得点の相関係数を計算できます。相関係数は、-1以上1以下の値です。1に近づくほど、片方の項目の値が増加するともう片方の項目の値も増加する傾向が強くなります。-1に近づくほど、片方の項目の値が増加するともう片方の項目の値が減少する傾向が強くなります。

2つの項目の値の組を (x_i, y_i) とし、 n 個の組 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ があるとします。相関係数を求めるには、まず共分散を計算します。共分散 C_{xy} は、次の式で求められます。

$$C_{xy} = \frac{(x_1 - \bar{x})(y_1 - \bar{y}) + (x_2 - \bar{x})(y_2 - \bar{y}) + \cdots + (x_n - \bar{x})(y_n - \bar{y})}{n}$$

相関係数は、次の式で求められます。

$$\frac{C_{xy}}{(x \text{ の標準偏差})(y \text{ の標準偏差})}$$

演習 3 相関係数の計算

data2.txt には、1 行に 1 人分の国語、算数、社会の得点 (いずれも整数) が、タブ区切りで記録されています。国語の得点と算数の得点の相関係数、および国語の得点と社会の得点の相関係数を求めてください。ファイルに記録されているデータは多くても 100 人分とします。

実行例

国語の得点と算数の得点の相関係数: 0.520063

国語の得点と社会の得点の相関係数: 0.872736

正解例: ex182.c

演習 3 の正解例 (ex182.c) は、次の順に処理を行っています。

1. (23 行目～) `data2.txt` から 1 行ずつデータを読み取る。科目ごとに用意した配列 (`score_koku`, `score_san`, `score_sya`) に得点を格納するとともに、平均の計算に必要な得点の合計 (`sum_koku1`, `sum_san1`, `sum_sya1`) を求める。
2. (33 行目～) 科目ごとに平均 (`ave_koku`, `ave_san`, `ave_sya`) を計算する。
3. (38 行目～) 科目ごとに標準偏差 (`sd_koku`, `sd_san`, `sd_sya`) を計算する。
4. (49 行目～) 国語の得点と算数の得点の共分散 (`cxy`) を計算し、相関係数を表示する。
5. (57 行目～) 国語の得点と社会の得点の共分散 (`cxy`) を計算し、相関係数を表示する。

この正解例は、ソースコードのわかりやすさのため、多少冗長な書き方をしています。
共分散は、次の式で計算することもできます。

$$C_{xy} = \frac{x_1y_1 + x_2y_2 + \cdots + x_ny_n}{n} - \frac{(x_1 + x_2 + \cdots + x_n)(y_1 + y_2 + \cdots + y_n)}{n^2}$$

これと式 (2) を使えば、相関係数を次のように計算できます。

$$c = (x_1y_1 + x_2y_2 + \cdots + x_ny_n) \times n - (x_1 + x_2 + \cdots + x_n)(y_1 + y_2 + \cdots + y_n)$$

$$s_x = (x_1^2 + x_2^2 + \cdots + x_n^2) \times n - (x_1 + x_2 + \cdots + x_n)^2 \quad (3)$$

$$s_y = (y_1^2 + y_2^2 + \cdots + y_n^2) \times n - (y_1 + y_2 + \cdots + y_n)^2 \quad (4)$$

$$\text{相関係数} = \frac{c}{\sqrt{s_x s_y}} \quad (5)$$

式 (5) を使って、配列にデータを記録せずに相関係数を計算するように改変したソースコードが、`ex182-2.c` です*6。

x_i が国語の得点で y_i が算数の得点の場合、式 (3) の s_x の値は 321,564 に、式 (4) の s_y の値は 232,341 になります。式 (5) で $s_x s_y$ を計算すると 74,712,501,324(約 747 億) になり、`int` 型で表せる範囲を超えてしまいます*7。これを、**桁あふれ**や**(算術) オーバーフロー**といいます。桁あふれを避けるために、計算方法を工夫したり*8、`double` 型で計算を行ったりする必要があります。`ex182-2.c` では、 $s_x s_y$ を `double` 型で計算しています*9。

移動平均

`data3.txt` は、2022 年 7 月 1 日から 2022 年 8 月 31 日の、日本国内における新型コロナウイルス感染症の新規陽性者数です*10。1 行が 1 日分のデータで、日付 (文字列) と新規陽性者数 (整数) がタブ区切りで記録されています。このデータでグラフを描くと、図 1 のようになります。

*6 変数 `sum_koku2`, `sum_san2`, `sum_sya2` の使い方が、`ex182.c` とは異なります。

*7 代表的な処理系では、`int` 型で表せる値の上限は約 21 億です。

*8 たとえば、 $\sqrt{s_x s_y}$ ではなく $\sqrt{s_x} \sqrt{s_y}$ で計算すれば、桁あふれは発生しません。

*9 桁あふれが起こることがわかっている部分だけ `double` 型で計算しました。普通はデータの量や値が決まっていますので、桁あふれが起こりそうな部分を予測する必要があります。

*10 データ出典: 厚生労働省

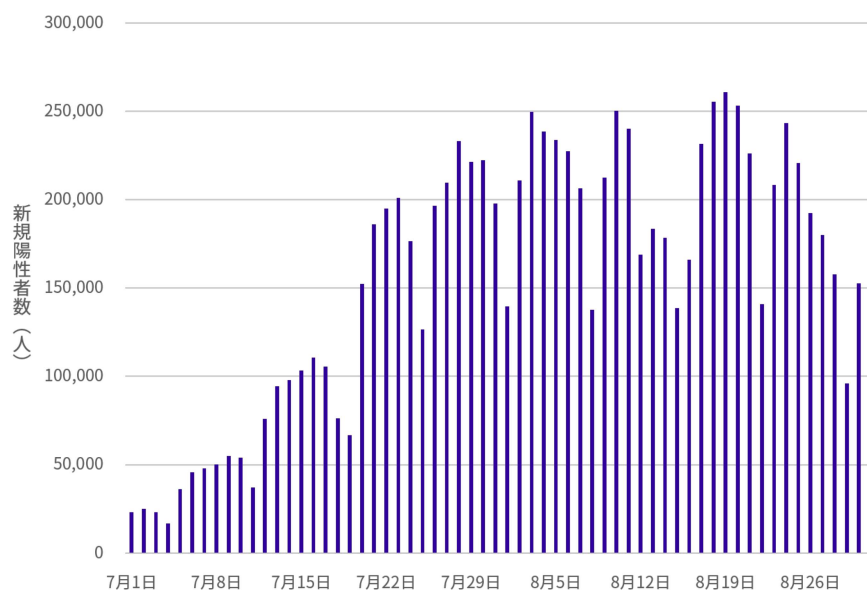


図 1 新型コロナウイルス感染症の新規陽性者数

曜日の影響が大きいので、7 日後方移動平均^{こうほう}を計算することにします。7 日後方移動平均とは、対象日を最終日とする 7 日間の平均です。例えば、7 月 10 日の 7 日後方移動平均は、7 月 4 日から 7 月 10 日の 7 日間の平均です。7 日後方移動平均によって、曜日の影響を除いた傾向を把握することができます。最初の 6 日間は、7 日後方移動平均を計算することができません。data3.txt の 7 日後方移動平均を求めてグラフを描くと、図 2 のようになります。

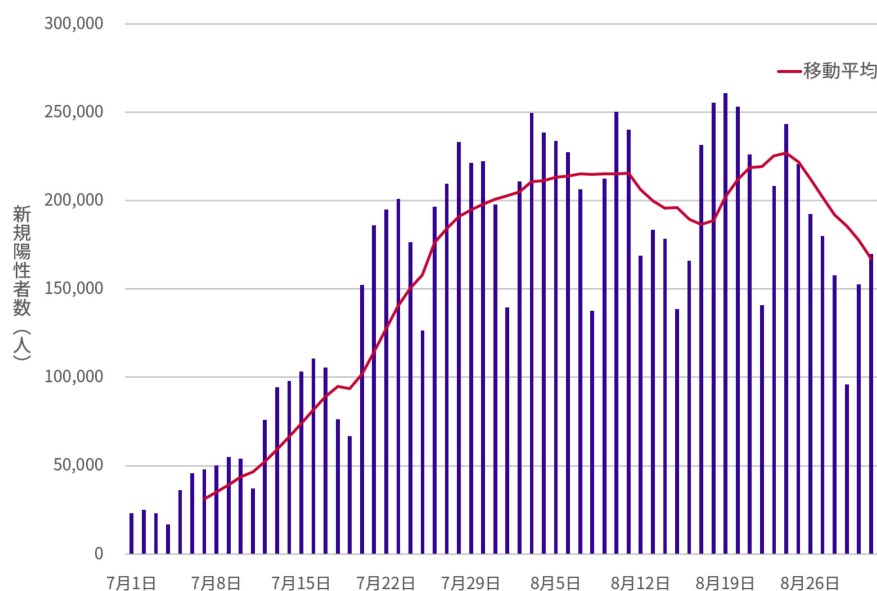


図 2 図 1 のグラフに 7 日後方移動平均を追加

移動平均を求めるときに、対象日ごとに7日分の値を足すのは非効率です。そこで、次のような工夫をします。

i 日目の値を x_i とします。 i 日目の移動平均の計算に使う7日分の値 x_{i-6} 、 x_{i-5} 、 $\dots$ 、 x_i と、その総和 sum を記録しておきます。 i 日目の移動平均は、 $sum \div 7$ で求められます。次の日である $(i+1)$ 日目の移動平均の計算に使うため、 sum から x_{i-6} を引いて x_{i+1} を加えます。また、記録している7日分の値から x_{i-6} を消して、新たに x_{i+1} を加えます。このように、日ごとに最新の7日間のデータの保持と総和の更新を行います。

プログラムでは、7日分の値の記録に要素数7の配列 `buf` を用意して、先頭の要素から順に値を記録することにします。末尾の要素である `buf[6]` に値を格納した次の日は、先頭に戻って `buf[0]` に値を格納します。総和の保持に初期値0の変数 `sum` を用いると、次のような処理を行います。

1日目は、`sum` に x_1 を加算して、`buf[0]` に x_1 を格納します。2日目は、`sum` に x_2 を加算して、`buf[1]` に x_2 を格納します。3日目から6日目も同様の処理を行い、7日目は、`sum` に x_7 を加算して、`buf[6]` に x_7 を格納します。7日目から移動平均が計算できます。8日目は、`sum` から `buf[0]` を引いて x_8 を加えます。そして、`buf[0]` に x_8 を格納(上書き)します。9日目は、`sum` から `buf[1]` を引いて x_9 を加えます。そして、`buf[1]` に x_9 を格納(上書き)します。配列 `buf` の使い方のイメージを、図3に示します。

1日目	[0]	[1]	[2]	[3]	[4]	[5]	[6]	
buf	x_1							sumに x_1 を加え、buf[0]に x_1 を格納
2日目	x_1	x_2						sumに x_2 を加え、buf[1]に x_2 を格納
								⋮
7日目	x_1	x_2	x_3	x_4	x_5	x_6	x_7	sumに x_7 を加え、buf[6]に x_7 を格納
8日目	x_8	x_2	x_3	x_4	x_5	x_6	x_7	sumからbuf[0](x_1)を引いて x_8 を加え、buf[0]に x_8 を格納
9日目	x_8	x_9	x_3	x_4	x_5	x_6	x_7	sumからbuf[1](x_2)を引いて x_9 を加え、buf[1]に x_9 を格納
								⋮

図3 配列 `buf` の使い方

演習 4 移動平均の計算

data3.txt からデータを読み取り、上述の方針に基づいて 7 日後方移動平均を計算して、data3-2.txt に記録するプログラムを作ってください。data3-2.txt は、1 行を 1 日分のデータとして、日付、新規陽性者数、その 7 日後方移動平均をタブ区切りで記録します。ただし、最初の 6 日間は、7 日後方移動平均を記録しません (日付、タブ、新規陽性者数、タブとします)。

このプログラムは、画面には何も表示しません^a。このプログラムが作成する data3-2.txt の正解例が、data3-3.txt です。

正解例: ex183.c

^a ファイルの作成に成功したことを表示すると親切です。

図 3 の動作を実現するためには、次に値を格納する要素 (図 3 で背景色を付けた要素) の添字を保持しておく必要があります。たとえば、1 日目であれば 0、2 日目であれば 1、8 日目であれば 0 です。演習 4 の正解例 (ex183.c) でこの役割を持つ変数が p です。p の値は 1 日分の処理を終えるごとに 1 だけ増やし、6 の次は 0 に戻します。p の更新処理は、次のように行えます。

```
1  /* pの値を1だけ増やし、6を超えたら0に戻す */
2  if (++p > 6)
3  {
4      p = 0;
5  }
```

正解例では、この処理を次のように行っています。

演習 4 の正解例 (ex183.c)

```
37      p = (p + 1) % 7;
```

p を “p + 1 を 7 で割った余り” に更新することによって、p の値を 1 だけ増やし、7 になったら 0 に戻す処理を実現しています。配列の末尾の要素の次に先頭の要素の処理を行うような場合に、このような手法がよく用いられます。

出現文字の数え上げ

英語の文に現れる英字 (アルファベット) は、e や t が多く、z や q が少ないと言われています。本当かどうか確かめてみましょう。

ファイルから文字を 1 つずつ読み取り、それが英字 (a~z) であれば数を数えます。大文字と小文字は区別しないことにして、読み取った文字は小文字に統一します。

ファイルから文字を読み取るには、表 3 に示すライブラリ関数 `fgetc()` を使います。EOF は、`int` 型の負の値の定数 (マクロ) です。

表 3 `fgetc()` 関数

書式	<code>fgetc(fp)</code>
引数	ファイルポインタ <code>fp</code> (<code>FILE *</code> 型)
戻り値	<code>fp</code> から読み取った文字 (<code>int</code> 型)。ファイルの末尾に達した上でさらに読み取ろうとしたり、読み取りに失敗したりした場合は EOF。
必要ヘッダ	<stdio.h>

大文字を小文字にするには、表 4 に示すライブラリ関数 `tolower()` を使います。

表 4 `tolower()` 関数

書式	<code>tolower(c)</code>
引数	文字 <code>c</code> (<code>int</code> 型)
戻り値	<code>c</code> が大文字であれば <code>c</code> の小文字、大文字でなければ <code>c</code> (<code>int</code> 型)
必要ヘッダ	<ctype.h>

文字が英字であるかを調べるには、表 5 に示すライブラリ関数 `isalpha()` を使います。

表 5 `isalpha()` 関数

書式	<code>isalpha(c)</code>
引数	文字 <code>c</code> (<code>int</code> 型)
戻り値	<code>c</code> が英字であれば 0 以外 (真)、そうでなければ 0 (偽) (<code>int</code> 型)
必要ヘッダ	<ctype.h>

それぞれの文字の個数を記録するために、要素数 26 の配列を用意して、先頭から順に a の個数、b の個数、…、z の個数を格納します。そのためには、文字 'a' なら 0、'b' なら 1、…、'z' なら 25 のように、文字から添字を求める必要があります。この演習では、簡易化のため、'a' から 'z' の文字コードが連続していることを前提にします^{*11}。すると、英字から 'a' を引けば、その英字に対応する添字が求められます^{*12}。たとえば、'b' - 'a' は 1、'z' - 'a' は 25 です。また、'a' + 1 を文字として読めば 'b' で、'a' + 25 を文字として読めば 'z' です。

^{*11} ほとんどの処理系では連続しています。

^{*12} 文字定数の型が `int` 型であることを思い出してください。

演習 5 出現文字の数え上げ

上述の方針に基づいて、`data4.txt` に記録された英語の文に現れるアルファベットの個数を数えるプログラムを作ってください。`data4.txt` は用意していないので、適当な英語の文を `data4.txt` というファイル名で保存してお使いください。

以下は、著作権の切れた作品を公開している Project Gutenberg からダウンロードした、Herman Melville の小説 “Moby Dick; Or, The Whale”^aを使った実行例です^b。

実行例

```
a: 79235
b: 17208
c: 23323
  ⋮
y: 17244
z: 637
```

正解例: `ex184.c`

^a <https://www.gutenberg.org/ebooks/2701>

^b ライセンスの説明など、小説以外の部分も含まれています。

演習 5 の正解例 (`ex184.c`) の 7 行目では、配列の宣言時に初期化する場合、値を指定しない要素の値が 0 になる性質を使っています。 `int count[26] = {0};` と宣言すると、すべての要素の値が 0 になります^{*13}。

出現回数の降順に表示するように改変したソースコードが、`ex184-2.c` です。`ex184-2.c` では、26 行 2 列の 2 次元配列を用意し、1 列目に 'a' ~ 'z' を、2 列目にそれぞれの文字の出現回数を格納します。そして、バブルソートによって、2 列目の値の降順に行を並べ替えます。

^{*13} 従来の C 言語の規格では、初期化のための { } の中に少なくとも 1 つは値を指定しないといけないので、`count[0]` だけ初期値として 0 を指定しています。