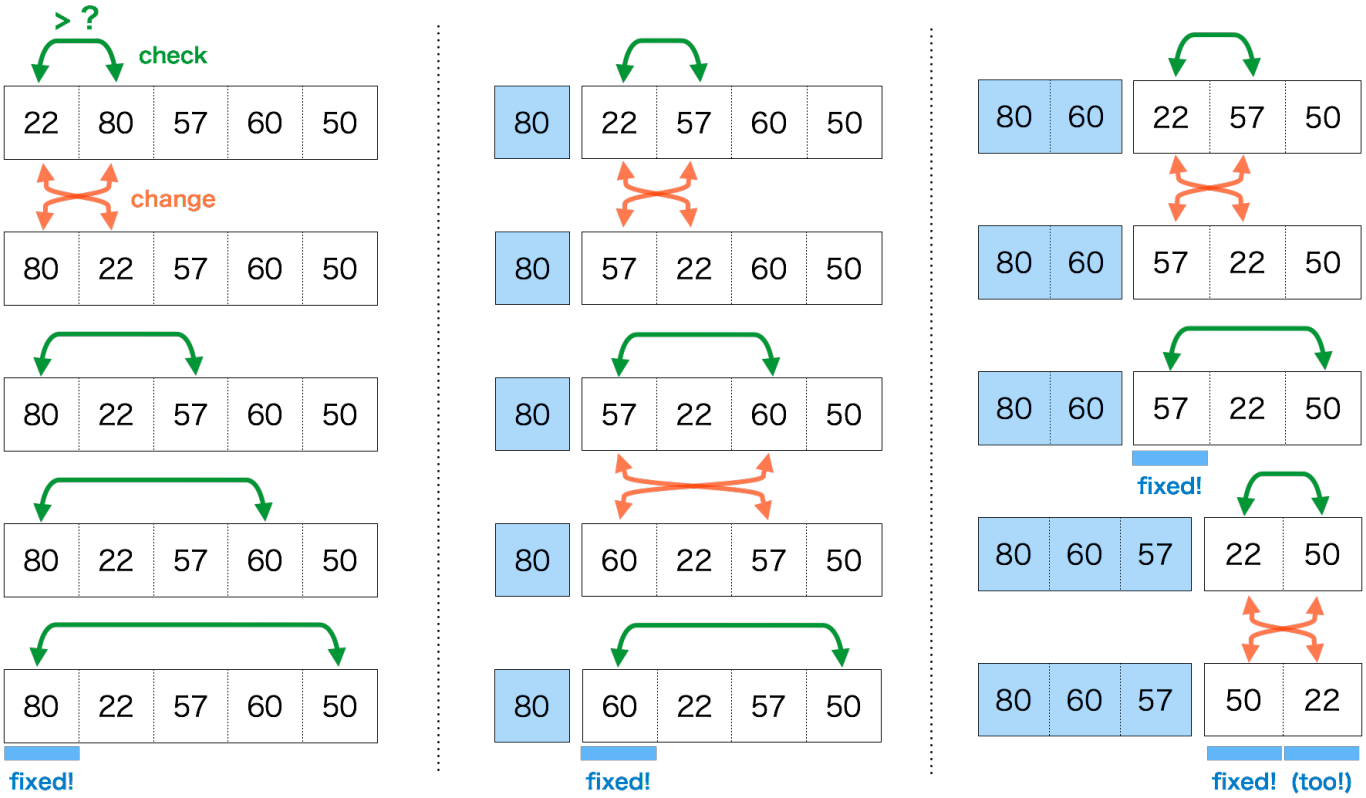
基礎プログラミング演習 II 教材 (#5)

■ ソート（選択ソート）

参考：
教科書のソートアルゴリズムは選択ソート（セレクションソート）と呼ばれるもの。単純で理解しやすいので配列の利用例として用いられたと推測。

教科書 p.197 以降、配列内容のソートをハンドトレースします。
（並べ順が降順になっています。前回教材は昇順ソートでした。）

アルゴリズムとしては、
1. 先頭の要素をそれ以降の要素と比較して、
2. より大きなものを先頭にキープ
3. 最後まで比較し尽くせば、先頭には最大のものがあるはず
4. 先頭の要素を切り離して、残り四つの要素群を対象に 1.〜3.処理を再び実行
5. 最後まで繰り返したら、先頭の要素は残り四つの要素群の最大、つまり全体で見て二番目のものがあるはず
6. これもまた切り離して残り三つの要素群で同じ処理を繰り返し
7. 残り二つでも同様、というかこれで最終処理。大きい方を前にすると最後の要素＝最小であるはず



実際に教科書のコードを見て、紙に配列変数 test[ ] の枠を 5 つ描き、変数 s, t の枠も描いて動きを実際に追いかけて下さい。（ハンドトレースと言います。プログラマは必ずやります。）

□ 入れ替える処理

教科書のコードで、右のような記述があります。これは上の図の赤色の chage 処理部分、つまり二つの要素を入れ替える処理の、手続き型言語での一般的な手法（定石）です。

```
if(test[t] > test[s]) {
    tmp = test[t];
    test[t] = test[s];
    test[s] = tmp;
}
```

・ 変数への代入は「上書き」つまり元の値を失わせてしまう
・ 二つの代入を並行して処理させるような記述が（一般的な言語は）持たない

ことが理由で、例に示したように、どこかにメモリ領域を一つ用意しておき、上書き前に値を保存（退避）しておくのです。
変数名 tmp は temporary（一時的な）の略で、よく使われる名前です。

■ 二次元配列

data[5]

[0]	[1]	[2]	[3]	[4]

★教科書 p.199「多次元配列のしくみを知る」を参照

右図に一次元配列（data[5]）と二次元配列（data[3][5]）の概念的な形を示す。一次元配列は要素が列になって並んでいるもので、並び位置でアクセスする要素を指定できるものだった。二次元配列とは縦横に要素が並んでおり、その並び位置を二つの次元で指定する。

data[3][5]

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]

実験：サンプルプログラムを取得して実行し、動作を確認せよ。このプログラムは右に示したように各要素にデータを入力し、行ごとの和を添えて出力するものである。

```
$ echo '1 2 3 4 5 （中略） 13 14 15' | ./a.out
  1  2  3  4  5 : 15
  6  7  8  9 10 : 40
 11 12 13 14 15 : 65
$
```

押さえて欲しいポイント：

・入力の際の `scanf()` で `&data[i][j]` につけられた & は気にしない
・二重ループで一つ一つの要素に入力していること
・合計用配列（縦横）には初期化が必要であること
・二重ループで一つずつの要素を処理していること
・こうした動きは図を書いて、紙と鉛筆でトレースして納得するのが良い

最も理解して欲しいこと：

結局のところ、
・C 言語では配列変数（という機能）を用意したが、
・配列をひとつの固まりとして処理するための機能は用意されていない
という構造に注目してください。
そうすれば、
・そのため配列を扱う処理は、必ず一つ一つの要素を個別に処理する必要がある
・多くの場合、これはループによって順繰りに処理されることで実現される
という結果を理解できると思います。

□ 課題 1.

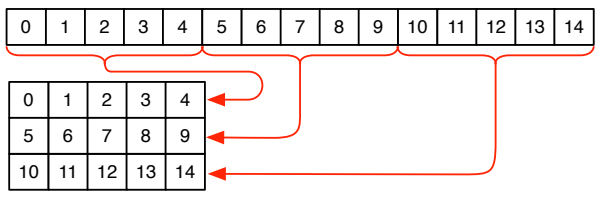
サンプルのプログラムに機能追加して、右図のように縦方向（列方向）についても合計を出力するようにしてください。

```
$ echo '1 2 3 4 5 （中略） 13 14 15' | ./a.out
  1  2  3  4  5 : 15
  6  7  8  9 10 : 40
 11 12 13 14 15 : 65
 -- -- -- -- --
 18 21 24 27 30
$
```

□ メモリ上の要素の並び

C 言語の中では変数があたかも二次元に配置されているように見えますが、実際にコンピュータのハードウェアが持っているメモリは一次元（線形アドレス）で並べられています。つまり一次元のハードウェア資源を使って二次元に見せる必要があります。
実際にはC言語の処理系は右図のような形で一次元に並んだメモリを「折り返す」格好で縦の並びを実現しています。

メモリの上での（物理的な）要素の並び



□ 課題 2. （ビンゴ）

1	3	4
8	2	5
9	7	6

3x3 のビンゴカードを考えてください。
その上で、サンプルプログラムを取得して右（上）の例のように実行してください。

```
$ echo '1 3 4 8 2 5 9 7 6'
| ./a.out
  1  3  4
  8  2  5
  9  7  6
$
```

入力（1 3 4 8 2 5 9 7 6）を二次元配列に設定し、そのまま出力できていることが確認できるでしょう。まずコードと結果を照合して、この動きを把握してください。
本当のビンゴでは、このあとクジを引いて出てきた数字に該当する枠があれば穴を開けていき、最終的に縦横斜めのどれかで三つ穴が並べば当選（ビンゴ！）となります。
これを真似て以下の機能を追加してください。右（下）を参照。

```
$ echo '1 3 4 8 2 5 9 7 6 1
6 2 5 7' | ./a.out
  1  3  4
  8  2  5
  9  7  6
 -- -- --
  .  3  4
  8  .  .
  9  .  .
 -- -- --
Bingo
$
```

1. これに続けて 5 つの数値（1 6 5 2 7）を「くじの値」として与えると、
2. 最終的にどのような形で穴が空いたのか図示し、
3. Bingo 状態があれば Bingo と表示する
4. 無ければ Not Bingo と表示。以下に Not Bingo になる例を示す。

```
$ echo 1 2 3 4 5 6 7 8 9 6 8 2 4 1 | ./a.out
$ echo 1 2 3 4 5 6 7 8 9 2 3 6 4 7 | ./a.out
```

注意：いきなり全部書かずに段階を経て作るように。まず上記 1. 2. 処理を先に作って動作確認し、それが出来たら 3.の判定処理を追加するのです。

□ 課題 3.（配列要素の回転）

右のように 3x3 要素のタイルの配列を 9 つの数値（1 or 0）として与え、それを右に 90 度回転（右に倒す）して表示するプログラムを作って下さい。つまり、
1. 3x3 の二次元配列に値を入力し、
2. そのまま入っている順に出力して内容を確認し、
3. その各要素を別の配列の（回転先に相当する）要素に格納し、
4. そのまま入っている順に出力して内容を確認し、
5. 正しく回転処理ができていることを確認するのです

```
$ echo '1 1 1 0 1 0 0 1 0' | ./a.out
before
■ ■ ■
□ ■ □
□ ■ □
after
□ □ ■
■ ■ ■
□ □ ■
$
```

実行結果は右のような形で出して下さい。なお 1. 2. までの処理を行うサンプルプログラムがあります。
ところで例年、配列要素を別の配列に移したりせず、違う順番で要素を出力（printf() ）することで上と同等の結果（見た目）になるようにしたプログラムを提出する人が居ますが、そんな事をしてはいけません。この課題は「配列要素の操作（異なる配列の間で要素の内容を移す）」ことが目的ですから、そこを外さないように。
注意：いきなりコンピュータに向かってはいけません。詳細な処理手順を決めて紙に書くように。

□ 課題 4. （連続回転）

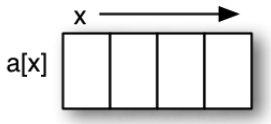
課題 3. で作成したプログラムを発展させて、右のように連続して回転させた結果を出して下さい。
やり方はいろいろあるでしょうが、
・ とりあえず課題 3.で動いたコードを三回続けて書き並べる
などといったことはしないように。
そこはループを使ってやるべきでしょう。
恐らくループを使ってやる場合は、
・ 元の配列 X から（回転を加えて）配列 Y を作る
・ 配列 Y の内容を配列 X にそのまま移す
・ この処理を三回ループさせる
といった格好になるでしょう。
（処理方針によってはループは四回になる場合があります。プログラミングですから唯一の正解なんて無いので。）

```
$ echo '1 1 1 0 1 0 0 1 0' | ./a.out
step 0
■ ■ ■
□ ■ □
□ ■ □
step 1
□ □ ■
■ ■ ■
□ □ ■
step 2
□ ■ □
□ ■ □
■ ■ ■
step 3
■ □ □
■ ■ ■
■ □ □
$
```

■ （参考）多次元の配列について

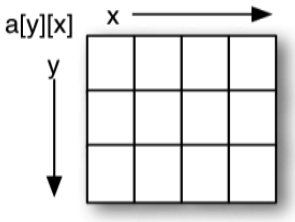
一次元の配列について、例えば右の図のように要素が一列に並んだイメージで考えることが多いでしょう。現在の主流となっているコンピュータのメモリの配置にうまく合う考え方でもあります。

一次元配列 a[4] の例

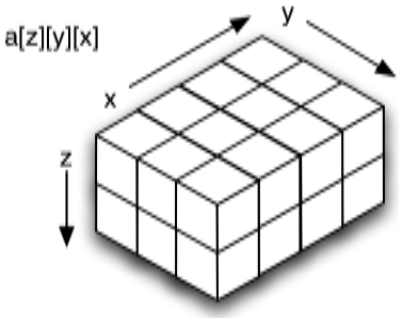


二次元の配列についても下図のように問題なく平面に要素を並べてイメージすることができるとおもいます。三次元の場合も同様に立体でイメージすることができるでしょう。

二次元配列 a[3][4] の例

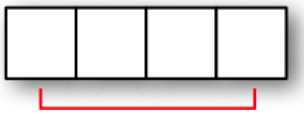


三次元配列 a[2][3][4] の例

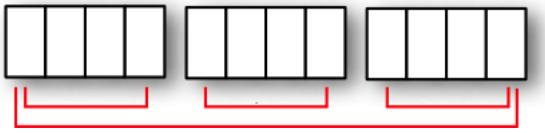


では次元数が更に高次になった場合はどうでしょう。四次元、五次元の配列をうまくイメージすることができますか？

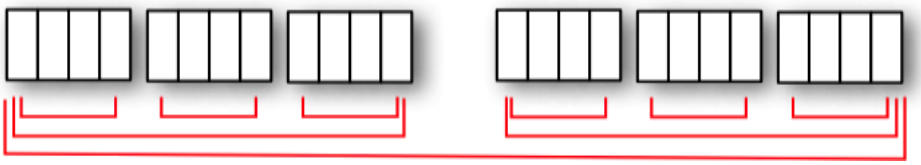
一次元の配列 a[4] はやはり要素が 4 つ並んだものだとしましょう。



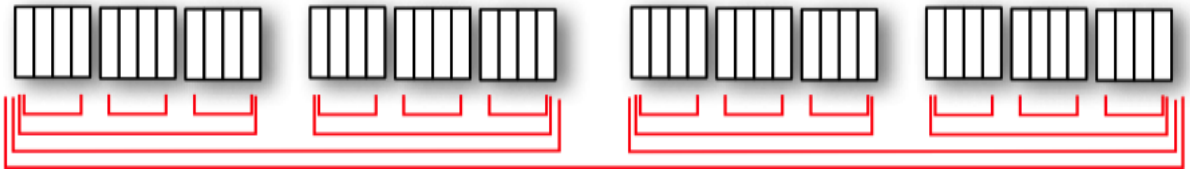
二次元の配列 a[3][4] は、a[4] を 3 セット、更に横に並べたものだと考えてみます。



三次元の配列 a[2][3][4] は、a[3][4] を 2 セット、繰り返して横に並べたものだと考えます。
こうすることで無理なく更により高次元な配列についてもイメージできるのではないかと思います。



例えば四次元の配列ならば下図のようになるでしょう。



実際のアプリケーションで多次元の配列を使う場合は、このように、ある一定のセットを、更に何セットか組み合わせるようなケースが多いでしょう。たとえば一台あたり 10 本の糸巻きがついている糸巻き機が、ある建物では一列に 20 台並んでおり、それが建物あたり 4 列配置されているとします。全 800 本の糸巻きを表す配列変数は spindle[4][20][10] となるでしょう。この建物が一つの工場に 5 つあり、それが 3 つの地域にあるとしたら、配列は自然と spindle[3][5][4][20][10] といった五次元配列となるでしょう。