プログラミング入門

配列の考え方

会津大学
d8161105 渡部未来 2014 6/21
資料公開URL : https://github.com/ababup1192/prog0

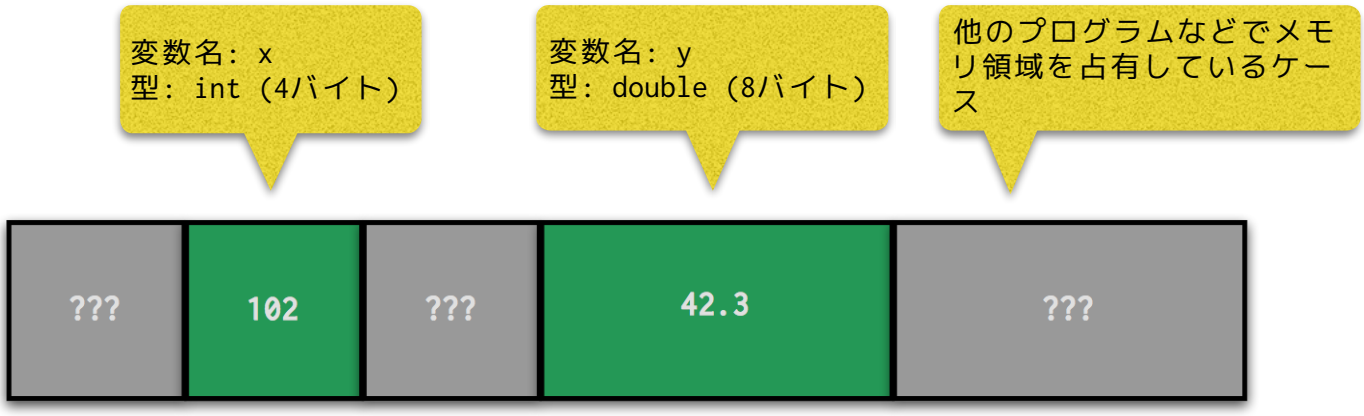
目的

　プログラミング入門における配列とループは大きな関門である。配列やループは、「イメージがしづらい」「プログラミング特有な考えだ」と考えてしまう。そのため何となくコードをいじってみることで偶然課題を解いてしまうと、きちんとした知識が定着されずに苦手になってしまいがちだ。この資料では、配列の基本的な考え方をキチンと学び、C言語のキモである配列とループをマスターしよう！

変数

　配列を押さえる前に、変数について復習しよう。変数とは**メモリ領域**に**型**と**名前**を定め、そこに値を、格納(代入)したり、呼び出したり、再代入したりすることができる仕組みのことである。

```
int x = 102;     // 変数宣言・初期化
double y = 42.3; // 変数宣言・初期化
```



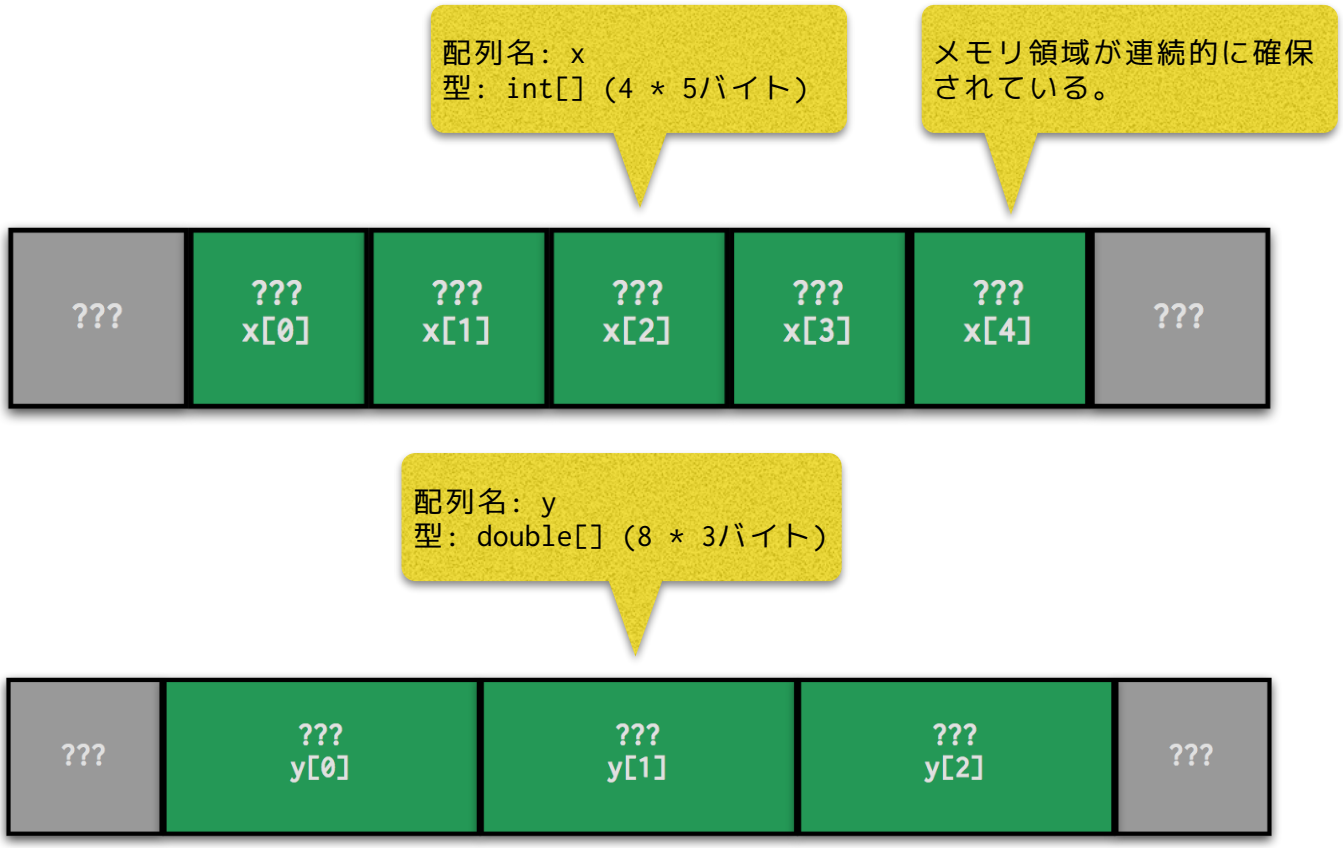
メモリ領域イメージ(変数)

変数で注意するポイントは、変数を**連続的に宣言**したとしても、メモリ領域上では**連続的な確保が保証されない**という点である。多くのケースでは、隣り合うメモリ領域には、他のプログラムなどで格納した値がそのまま残っている。初期化し忘れた変数の値を出力した場合に変な値(例: `-1020320`, `3021302`など)が出力された経験があると思う。ある時点で、変数内にどのような変数が格納されているかを意識することが、C言語を扱う上での基本かつ必須テクニックである。

一次元配列

先ほどの変数の節では連続的にメモリ領域を確保できないと説明をした。メモリ領域を連続的に確保する仕組みが**配列**である。

```
int x[5];          // int型5つ分のメモリ領域を確保
double y[3];       // double型３つ分のメモリ領域を確保
float a=5, b[a];   // このような動的な領域確保は許されない
```



メモリ領域イメージ(配列)

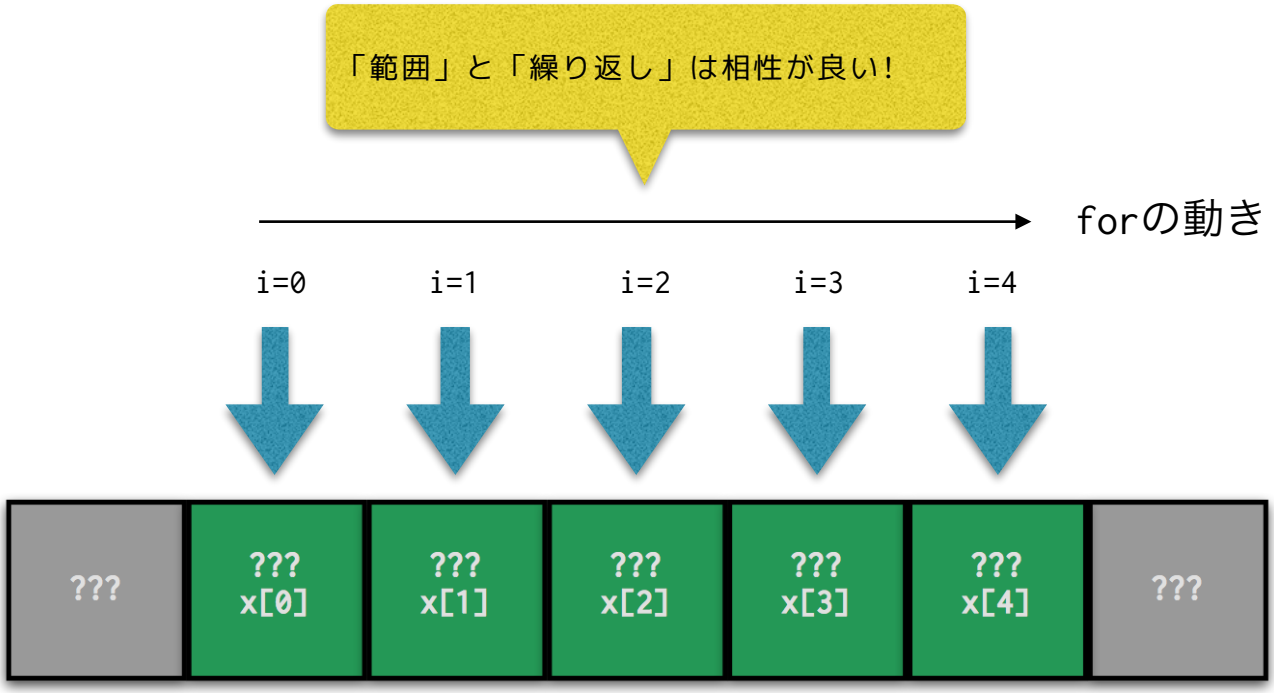
配列は変数宣言とは違い、メモリ領域を確保するためにどれだけの範囲の領域を確保するか最初に決める必要がある。確保する範囲は静的(コンパイル時)に確定するため、宣言時に変数ではなく定数を使わなければならない。また配列で割り振られた値(**要素**)一つ一つにアクセスするには、**添字**を指定する必要がある。そのため配列のそのもののコピーも値一つ一つにアクセスする必要がある。さらに変数同様に配列はメモリ領域を確保しただけでは、初期化は行われない点に注意しよう。

```
int x[5],y[5];
printf(“%d” ,x[2]);  // 配列xの3つ目の値を参照
x = y;               // 配列の値のコピーはされない。
x = 3;               // 添字を指定せずに配列のコピーはできない。x[0]=y[0];…
```

一次元配列とループ

　配列の節では連続的なメモリ領域を確保し各要素にアクセスする手段を学んだ。ただ値のアクセスに添字をその度に指定して扱うのは面倒である。一般的に配列をアクセスするには繰り返し処理を利用する。基本的に扱いたいメモリ領域の範囲は自明なはずなので、forループが適任である。(もちろんメモリ領域の範囲が不明な場合はwhileを使うケースもある)

```
int i, x[5];          // int型5つ分、メモリ領域を確保する。
for(i=0; i<5; i++){ // アクセスする範囲は0~4と、自明である
      x[i] = i+1;    // 1,2,3,4,5 という値が配列に代入される。
}
```



配列とループ

添字を指定した配列の値は、ただの変数と変わらない。つまり「初期化」「代入・scanfなどによる変更」「printfによる表示」が行える。これらの操作が混同しないために、forループは配列・操作ごとに用意したほうが良い。同一の配列ならばforループの範囲は変わらないので、記述は使いまわすことができる。配列とは、値を順序だてて並べた数学的な「列」と同義である。

これまでのまとめ

・変数とはメモリ領域の確保と、その領域に格納された要素へアクセスする仕組み
・配列とは「連続的に確保したメモリ領域(変数の列)」を扱う仕組み
・添字を使うことで、ただの変数としてアクセスすることができる
・配列と繰り返し処理は、相性が良い

二次元配列

　これまでメモリ領域と変数、配列の仕組みを学んできた。２次元配列とは列を複数行(行列)扱うための仕組みである。

```
int x[3][5];    // 3行5列の二次元配列(行列)
```

二次元配列イメージ

添字が２つに増えて少々複雑に感じるかもしれないが、キチンと理解すれば簡単である。1つの添字が行、2つ目の添字が列に対応している。行を固定してしまえば、ただの一次元配列と変わらないのである。
※ 資料作成者の場合は数学も苦手なので、2次元配列=行列でも少しピンと来ないため、原点が左上の(y, x)の点の集合という理解をしている。自分なりに考えやすいイメージに変換して知識の定着をがんばってみよう。

```
matrix[行][列]          // 行列的理解
linear[y座標][x座標]    // 一次元関数的理解
```

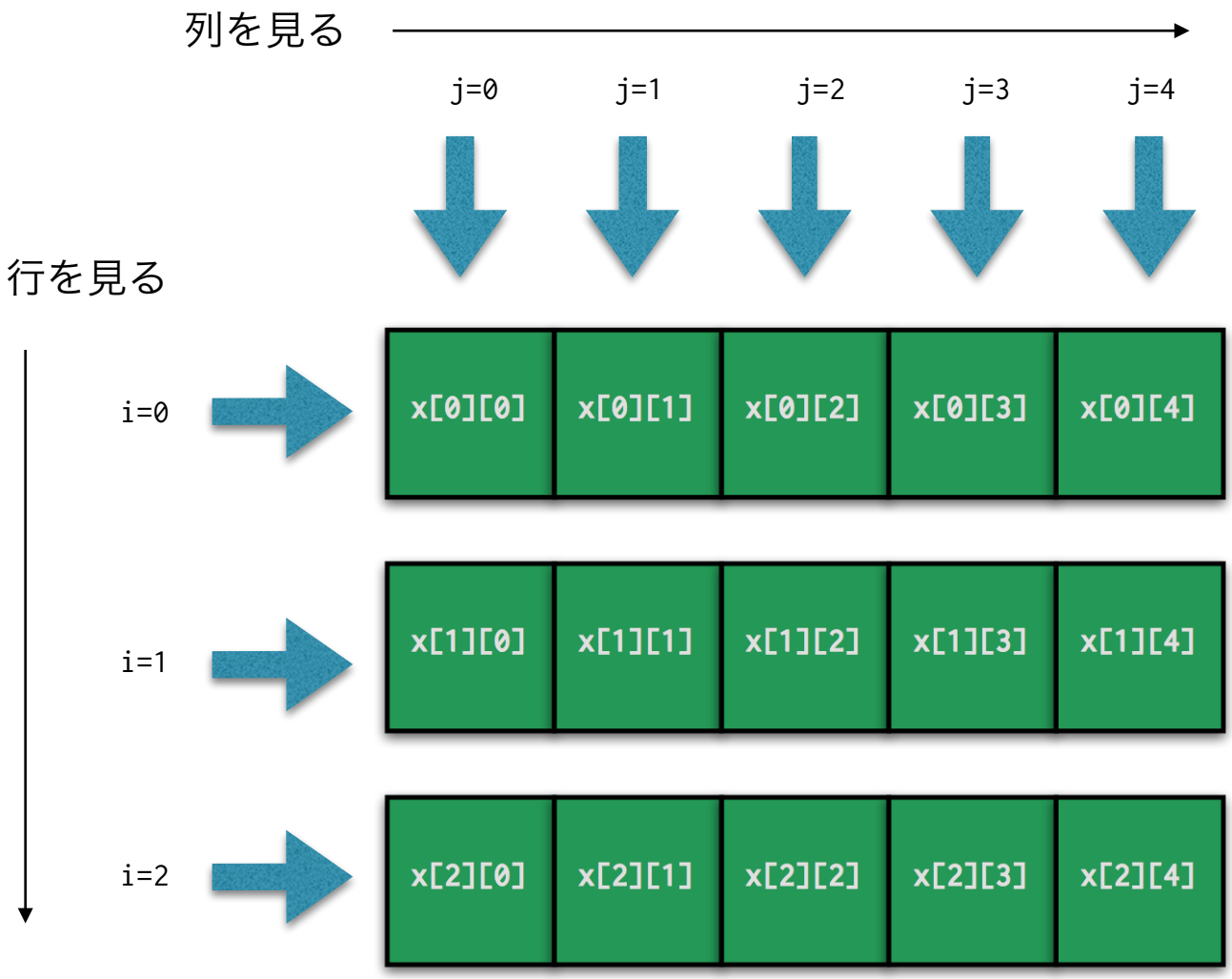
二次元配列とループ

二次元配列は一次元配列と同様に繰り返し処理を使うことでアクセスを簡略化することができる。行と列でカウンタ変数が２つ必要なことに注意しよう。

```
int i,j;    // i行,j列
int x[3][5]; // 3行5列の二次元配列

for(i=0; i<3; i++){           // 行を見るループ
      for(j=0; j<5; j++){     // 列を見るループ
            x[i][j] = i+1;
      }
}

//  1 1 1 1 1    という3*5の行列が得られる。
//  2 2 2 2 2
//  3 3 3 3 3
```



バグや予定外の動きを防止するために、プログラムを動かさなくてもカウンタ変数の動き、二次配列の動きが追えるようにしておこう。

二次元配列のパターン

　二次元配列を扱うには繰り返し処理を使うが、単純に二次元配列 = 行・列の二重ループと考えてしまうと深みにハマってしまうケースが多い。そこでよくある２つのパターンを紹介するので、二次元配列の扱いに困ったら、以下のパターンを試してみてほしい。

列を固定するパターン

　例えば教室にいる３列目の生徒だけのテストの情報を取得したいケースを考えてみよう。単純な行列の二重ループだと全ての生徒に処理を行なってしまう。そのときは列を固定することで単純なループ１つで、処理を行えばよい。

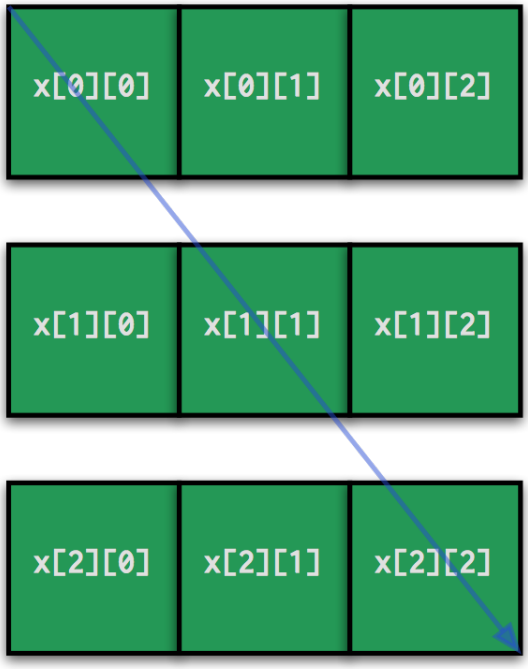
```
int i;                                          // i行
int x[5][7];                                    // 5行7列の合計35人いる教室

// 何らかの入力処理を挟んだとする。

for(i=0; i<5; i++){
      printf(“%d人目: %d点”,i+1,x[i][2]);      // 3列目なので、列は2で固定
}
```

法則性のあるパターン

　3*3の単位行列を作るケースを考えてみよう。単位行列は対角成分(斜め)が1、それ以外が0である行列のことである。この場合、単純にソースコードをにらめっこしていても拉致があかないので、メモリ領域のイメージを図示し、添字を書き込んでみるとある法則性がわかる。対角成分は行と列の添字が同様のときが1, それ以外が0なのである。場合分けが発生したときは、if文を利用することで法則を満たすことができる。




```
int i,j;
int x[3][3];      // 3行3列の行列

for(i=0; i<3; i++){
      for(j=0; j< 3; j++){
            if(i == j){
                  x[i][j] = 1;
            } else{
                  x[i][j] = 0;
            }
      }
}
```

まとめ

　二次元配列は一見すると何だか取っ付きにくい複雑な機能に感じられるかもしれないし、今まで勉強した全ての知識を総動員しなければならないので大いなる壁と感じてしまうかもしれない。ただ、数学的に考えれば行列、１次関数の点の集まり。プログラム的に考えても、一次配列の集まり。一次配列も単なる変数の列に過ぎないのだ。それほど悲観せず、甘く見過ぎずに辛抱強く勉強して欲しい。

　アドバイスとしては難しい問題に対して、いくらモニターを見つめても、ソースコードの文字を見つめても解決しない事のほうが多い。そのようなとき、資料作成者である僕は、ひたすら紙とペンに思考のイメージを書いて、それをそのままプログラムとしてつくり上げることがある。これは個人的な考えだけではなく、企業や研究者も基本的に実践していることなので是非、皆さんにも実践してもらいたい。変数・配列に関するイメージの書き方は、この資料のイメージ図を参考にしてもらえると、かなり理解の手助けになると信じている。そして、お絵かきは楽しい！プログラミングを楽しんで覚えよう！