**・データ構造って何？**

一言でいえば，データをメモリ空間ににどのように格納するのか，その賢い格納の仕方，とでも考えてよい。メモリ空間の物理的な性質から，下記の二種類のデータ構造が考えられる。

ハードウェア的にデフォルトで用意されているデータ構造　　配列

ソフトウェア的にプログラマが作り出すデータ構造　　その他（リスト，木，，，）

**・ハードウェア的にデフォルトで用意されているデータ構造って何？**

まずは，メモリの物理実体についてのイメージを考える。

部屋番号が振られた小部屋が一列にずーっと並んでいるイメージ。

引き出し番号が振られた引き出しが一列にずーっと並んでいるイメージ。

座席番号が振られた椅子が一列にずーっと並んでいるイメージ。

一列に並んでいるので，前後（上下？）のデータを見る（部屋の中を覗く，椅子に座っている人を見る）ことは簡単である。また，部屋番号や座席番号を指定して所望の場所にデータを書き込む，その場所のデータを参照するのも楽ちんである。そのための部屋番号である。C言語では，以下のように宣言すれば，すぐ使える。データ構造という意識はあまり無い，であろう。

```
double data[10];
        data[0] = 3.0;   /* 先頭の部屋に 3.0 というデータを格納する */
        data[1] = 2.3;   /* 次の部屋に，，，，*/
        x = data[8];
```

**・ハードウェア的にデフォルトで用意されているデータ構造だけだと，色々不便である。**

例えば，与えられたデータを，与えられた順に何も考えずに小部屋・引き出し・椅子に格納する（置く）ことを考えてみる。

もちろん，格納は楽である。何も考えずに置いていくだけ，だし。

でも，その後が大変。あの人どの部屋にいたっけ？あの本どの引き出しだっけ？彼女どの席に座ってる？　見つける（検索する）のが一苦労となる。そこで，賢い格納の仕方が必要になる。

**・普通がどうする？　〜整理整頓〜**

部屋の場合，１年生は１階に，２年生は二階に，部屋の割当をする。

引き出しの場合，第一段は書籍，第二段は書類，第三段は文房具を入れる。

座席の場合，小学生は前列，中学生はその後，大人は一番後ろに座らせる。

つまり，データの持つ「何らかの特性」に応じて，また「格納したデータに対して，その後どういう操作が必要になるのか」をよくよく考えて，格納の仕方を「賢く」行なう，ことをやる。

「でもちょっと待って！」

「１階，２階，あるいは，一段目，二段目って，格納するハードウェアそのものに，賢い格納方法を実現する枠組みがあるからできること，じゃない！　上にあるように，メモリーの場合はただただ，ずーっと一列に並んでいるだけ，じゃない！」

「お，鋭い，，，。その通り，一列に並んだだけのメモリーをどう料理するのか？」

**・メモリー（ただただ，一列に続く椅子）に賢く座らせる方法はどうしたらよい？**

まず，座らせた学生にその後何を行なわせたいのか，それを考えてみる。例えば，，，

**座らせた後に学生に行なわせたいこと：**その座席で試験を行ない，座席番号ではなく，学籍番号順に解答用紙を回収したい。でも，学生は来た順に（学籍番号とは無関係に，好き勝手に）座席に座っていく，，。但し，詰めて座っていく，，，。

**一つの解決策：**学生が来て，詰めて座席に座らせる。で，座る度に，その学生にとって，すぐ上の学籍番号（その時点で座席に座っている学生の中で，学籍番号が次に大きい）を持つ学生と，そのすぐ下の学籍番号（次に小さい）を持つ学生との間にロープを張る。解答用紙を回収する時に，学生がロープを引くと，すぐ上（あるいはすぐ下）の学生が座っている椅子が動く仕組み。

試験後に，学籍番号が一番若い人がロープを引くと，次に若い人の椅子が動く。解答用紙をそのロープ沿いに渡して行って，順々に回収すれば，学籍番号順に解答用紙を回収することができる。この時，座席番号は殆ど意味を持たない（配列というデータ構造は実際問題として，物理的には存在しているが，配列，というデータ構造を積極的に使っている訳ではない）。学生を座らせる時に（データを格納する時に），**データとデータを上手く関連づけておけば（今の場合は，学籍番号のすぐ上，すぐ下の学生がどこにいるのかを，ロープを使って把握しておくこと），**その後処理を行なう時に，非常に楽，ということになる。この**「データとデータを上手く関連づけて」格納すること**が，ソフトウェア的にデータ構造を作る，という「表現」に対応する作業である。例えば今の例だと，ロープを張って，すぐ上（下）の学籍番号の学生の椅子に繋げる，というのがポインタ変数のことになる。すぐ上（下）の学生が座っている座席の番号を控えておく，という操作である。すぐ上の学生とすぐ下の学生へのリンクを有する，という意味で，これは双方向リストと言える。ずーっと一列に続く座席には，座席番号が振られ，それは配列と考えられるが，配列というデフォルトで与えられる「物理的構造」の上に，ロープ（ポインタ変数）を使うことで双方向リストというデータ構造をソフトウェア的に，論理的に構築したことになる。

以上，ハードウェア的にデフォルトとして与えられる配列（と呼ばれるデータ構造）に，どのようにしてソフトウェア的に双方向リストと呼ばれるデータ構造を作るのか，を概念的に示した。ハードウェア的なデータ構造とソフトウェア的なデータ構造とが混同している学生がいるように思われたので，参考にして欲しい。なお，本文書を読んだ学生の反応として，

A：「何，当たり前のことを繰り返しているんだ！」
B：「仰ることは理解していますが，ロープを張る，という操作をC言語や，Javaで実際にコーディングする作業になると，とたんに自信がなくなる。」
C：「なぜ，ロープがポインタ変数なのか，それが分からない。」

などがあると思う。Aの学生は，さっさと本文書を捨てて良い。Bの学生は，経験を積むしかありません。「バグのあるコードを書いて，そのバグに悩み，解決して」という経験を積んで，プログラムを書くという作業に要求されるコツを獲得する，しか無いでしょう。Cの学生は，かなり重傷です。恐らく，この実験課題は出来ないでしょう。主任や副主任を担当していなければ，課題をやるよりも，再度教科書を読んで，この文書の意図していることの理解に務めた方がよいかもしれません。

最後に，，，。

峯松の研究室は音声や言語の研究をしていますので，文系出身の方もいます。文系出身の方（Aさん）と電気系出身の方（Bさん）との間で交わされた実際の会話を紹介します。二人とも博士課程の学生でした。

A：「ねえ，B君，このプログラムが思ったように動かないのだけど，ちょっと見てくれない。」
B：「えーっと，うわ，なんだこれ。なんで，こう分かりにくいコーディングをするんですか。もっと可読性の高い（読みやすい）コーディングをして下さいよ。こんなコーディングをするからバグるんですよ。ホント，頭の中を見てみたい，，，。」
A：「あんたが書く英語と一緒よ。よくもあんなに可読性の低い英語が書けるわね。アンタの頭の中だって見てみたいわよ。」
B：「・・・・・・」

隣で私が爆笑しておりました。データ構造を「整理整頓されたデータの格納方法」という言葉で紹介しました。上の可読性が低いコーディングですが，これは，峯松が考えるに，頭が整理整頓されていない状況で書いたコーディング，だと思います。整理整頓されたデータ構造をプログラミングするのに，本人の頭の中が整理整頓されていないのでは，致し方ありません。結局，整理整頓されたコーディングをするためには，常に自分の頭が整理整頓されていないといけません。これは逆に言えば，自分がどのような思考をすると「整理整頓できなくなるのか」という，自分の思考の癖を知っているかどうか，に関係してきます。自分の思考の癖を知っていれば，「ん，このまま頭の中だけで考えていては整理整頓できなくなるな」と感覚すれば，自ずと，紙の上でペンを走らせて何らかの視覚イメージとして今行なっている作業を書き留める（例えばフローチャートを書く）などをすることになると思います。例えば「ポインタのポインタをプログラミングする時は，頭の中だけでやるとどうしてもミスするから，面倒でも紙の上で視覚化して，それをコーディングするように心がけよう」とか，そういう姿勢です。

でも，いくらやっても英語が下手な人がいるように，いくらやっても整理整頓されたコーディングが出来ない人もいます，，，，。厳しいようですが，事実です。