今まで学習したデータ構造

山本昌志*

2005 年 4 月 27 日

1　本日の学習内容

本日は、授業変更に伴いセンターが使えないので、今まで学習した C 言語のデータ構造を復習する。本日の主な学習内容は、以下の通りである。

- これまで学習したデータ構造
- メモリーとデータ
  - 単純型変数
  - 配列
  - 構造体
- 変数の適用範囲

2　はじめに

現在、諸君は C 言語の文法の勉強をしている。実際プログラムの勉強は、文法の勉強だけでは全く不十分である。そのため、この講義では文法の学習が住むと、「アルゴリズムとデータ構造」の学習を進める。アルゴリズムとは問題を解く手順のことである。コンピュータープログラムでは、入力データから目的のデータを作成する手順を言う。プログラマーはアルゴリズムを考え、それをプログラミング言語で表現しなくてはならない。アルゴリズムが大事であることは、万人が認めることである。これは、コンピュータープログラム作成のみならず、数学や電気の問題を解く場合も同じである。科学の問題を解く場合は手順が重要で、それが分かれば、問題が解けたのも同様である。

一方、データ構造となると、少し様子が異なってくる。数学や電気の問題のデータ構造となると、想像がつかない。科学の問題では、データ構造は重要視されないので、それも仕方ないことである。データ構造を気にするのは、なんと言ってもビジネスの分野である。たとえば、私の場合、成績処理のデータ構造を考えなくてはならない。それには、年度と学年、学籍番号、学生氏名、教科名、試験名、点数、レポート提出状況、出欠などのデータを分かり易くまとめなくてはならない。実際には図**??**表計算の Excel を使っている

*独立行政法人　秋田工業高等専門学校　電気情報工学科

が、データ構造という意味では同じである。この例のように、データの集まりのまとめ方をデータ構造という。

プログラムを作成する場合、そのアルゴリズムとデータ構造を考えなくてはならない。アルゴリズムは重要でそれの善し悪しでプログラムの性能が決まるのは、理解できるであろう。しかし、データ構造については、結構無頓着になってしまいがちである。先の成績処理のように単純な問題であれば誰でも同じようなデータ構造を考え、大した問題は生じない。しかし、大量のデータがあるような場合には、いろいろなデータ構造が考えられる。大企業の顧客データなどがそれに当たる。住所や氏名、年齢、何時、どこで、購入した商品のデータがある。これらに加えて、アンケートに答えていれば、それこを数十の項目で、数百万人分のデータがあることも容易に想像できる。このデータをまとめ方、すなわち構造は重要で、その後の処理の方法にも大きく影響するであろう。この構造に従うということは、プログラマーの頭の中に、それがインプットされるので、それを用いた処理のプログラムを書くことになるからである。

また、データ構造が悪いと、効率の悪いプログラムになってしまう。たとえば、ファイルに 100 個の整数が書かれており、その合計を求めるプログラムでは配列を使うべきであろう。変数を使ったプログラムでは、効率が悪くなるのはすぐに分かるであろう。

表 1: データ構造の種類

データ構造	基本データ構造	基本データ型	単純型	整数型
				実数型
				文字型
				論理型
				数え上げ型
			ポインタ型	
		構造型	配列型	
			レコード型	
		抽象データ型		
	問題向きデータ構造	線形リスト	単純リスト	
			双リスト	
			環状リスト	
		木	二分木	完全二分木
				二分探索木
				バランス木
			多分木	
			バランス木	AVL 木
				B 木
		スタック		
		キュー		

3　これまで学習したデータ構造

3.1　単純型変数

単純型の変数は、次のように変数に一つの数値[1]しか代入できないものを言う。

```
char c, h, moji;
int i, j, seisu;
double x, y, jisu;
```

通常、これを変数と言う。変数というと、この単純型を示す場合が多いが、配列や構造体を含める場合もあるから、文脈から適当に判断しなくてはならない。

これのイメージは、図 3 に示しているとおりで、変数とは数値を入れる箱のようなものである。整数型と倍精度実数型の変数は、数学の変数と全く同じである。

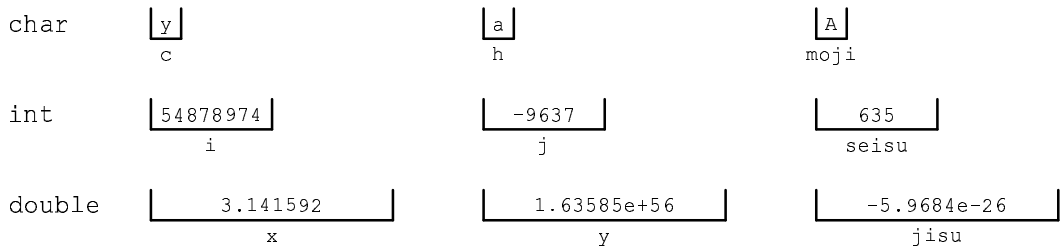

図 1: 変数のイメージ。変数とはデータを入れる箱のようなもの。

図を見て分かるように、箱の大きさが型によって異なる。これは、一つのデータを表現するために必要な情報量が異なるためである。情報量の単位は、ビット (bit) が使われる。2 進数の 1 桁を 1 ビットと言う。8 ビットで 1 バイトとなり、それがコンピューターで使われる基本単位となる。

同じ `int` 型でもいろいろあり、表現できる範囲が異なっている。これは一つの変数の情報量の差から生まれる。C 言語で使われる型によって表現できる範囲を 2 に示す。全ての C 言語は同じとなっておらず、諸君が使っているシステムではこの表のようになっている。いろいろな型があるが、ほとんどの場合、`char`、`int`、`double` で十分である。諸君が作るプログラムでは、これらで十分、間に合うが、問題が生じたときのみ他の型を使えば良い。

[1] 一つの文字のみ代入可能なものは文字型の変数である。文字も整数として扱えるので、この範疇と考える。

表 2: 型によるデータの表現の違い

型	バイト長	範囲	有効精度
char	1	-128〜127	
signed char	1	-128〜127	
unsigned char	1	0〜255	
short int	2	-32768 〜 32767	
signed short int	2	-32768 〜 32767	
unsigned short int	2	0 〜 65535	
int	4	-2147483648 〜 2147483647	
signed int	4	-2147483648 〜 2147483647	
unsigned int	4	0 〜 4294967295	
long int	4	-2147483648 〜 2147483647	
unsigned long int	4	-2147483648 〜 2147483647	
signed long int	4	0 〜 4294967295	
float	4	およそ 10^{-38}〜10^{38}	およそ 6 桁
double	8	およそ 10^{-308}〜10^{308}	およそ 15 桁
long double	12	およそ 10^{-4932}〜10^{4932}	およそ 18 桁

3.2 配列

一次元の配列は数学のベクトルと、二次元の配列は行列とよく似ている。実際、C 言語でベクトルや行列の演算を行うときには、構造が同じ配列を使うことになる。順序づけられた同じ型のデータが複数ある場合、配列の出番となる。添え字 (これが順序を表す) により、それらにアクセスできるので、データの操作が簡単にできる。

配列を使う場合、

```
int i[10], j[100][100];
```

のように宣言を行う。そうすると図**??**のように、メモリー領域が確保され、配列が使えるようになる。この配列のデータにアクセスするためには、配列名と添え字を指定する。次のようにである。

```
i[3]=5;
c=i[3];
```

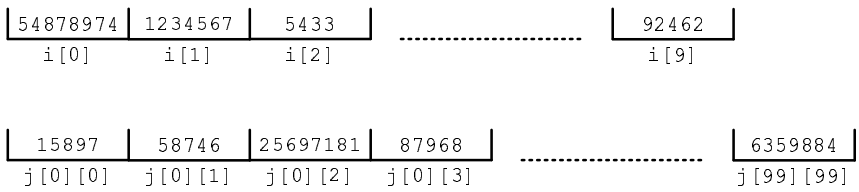

図 2: 配列のイメージ。データを入れる箱がいっぱいある。ただし箱の大きさは全て同じ。

3.3 構造体

配列は同じ型のデータの集まりであったが、構造体は異なる型のデータの集まりである[2]。この構造体を使う場合、

1. 構造体のメンバーを規定する。これにより構造体を定義する。
2. 構造体変数の宣言。これによりメモリーが確保される。

という手順が必要である。このようにして、構造体を定義し、メモリーを確保した後、それを使うことができる。今までは、データの型の内容があらかじめ決まっていたので、最初の手順は不要であった。一方構造体のメンバー、すなわちデータの型はプログラマーが決めなくてはならないので、最初の手順が必要となる。

最初のメンバーの規定は、つぎのように行う。

```
struct kouzoutai{
  char name[8];
  int seisu;
  double jisu;
};
```

これでは、メモリーがまだ確保されていないことに注意が必要である。これは、プログラマーが新たに変数を定義したのと同じである。

そして、この構造体を実際に使う場合には、次のようにしてメモリーを確保する。

```
struct kouzoutai a, b[10];
```

そうすると構造体変数が使用可能となる。この様子を、図**??**

メモリーに確保された構造体のデータにアクセスするためには、ドット (.) 演算子を使うことになる。次のようにである。

```
a.seisu = 5;
b[5].name[3]='a';
x = b[6].jisu;
```

[2]同じ型でも良い

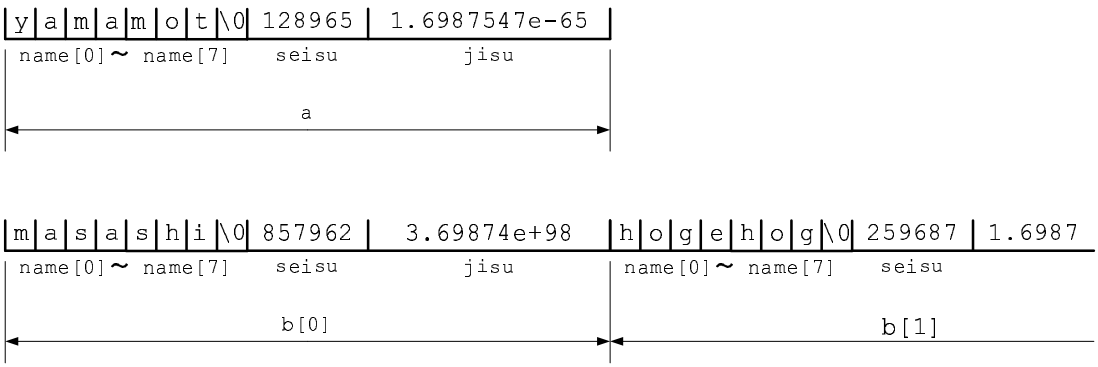

図 3: 構造体のイメージ。データを入れるいろいろな大きさの箱がある。

4　メモリーとデータ

諸君は方程式を使って問題を解く場合、変数というものを使っている。数学の変数と C 言語のプログラム中での変数の決定的な違いは、変数の型の宣言が必要なことである。実際にプログラム中では、

1. 変数や配列、構造体の宣言。構造体の場合は、それに先だって構造体のメンバーを規定しなくてはならない。
2. データ構造に数値を格納する。

という手順が必要である。後者は、数学の変数と似たような使い方をする。前者が数学と異なっており、実際のプログラムのアルゴリズムでは不要に思える。これが必要なのは、コンピューターの都合である。たいていのプログラミング言語では、これを宣言することにより、メモリーを確保する。必要なメモリー量はプログラマーが決めなくてはならない。コンピューターは、このプログラムがどの程度のメモリーが必要か全く分からないからである。

5　変数の適用範囲

宣言された変数が使える範囲を**適用範囲 (Scope)** という。関数の内側で宣言した変数を**ローカル変数**、外側で宣言した変数を**グローバル変数**と呼ぶ。ローカル変数は宣言された関数内でしか使うことができないが、グローバル変数はどこからでも使える。その様子を図 4 に示す。

グローバル変数とローカル変数は同じ名前を使うことができるが、ローカル変数が優先されることになっている。ただし、実際のプログラムでは、このようにするとわかりにくいバグの原因となるので、慎むべきである。

他にいろいろな宣言を行い適用範囲を変えることができるが、これについては将来学習する。

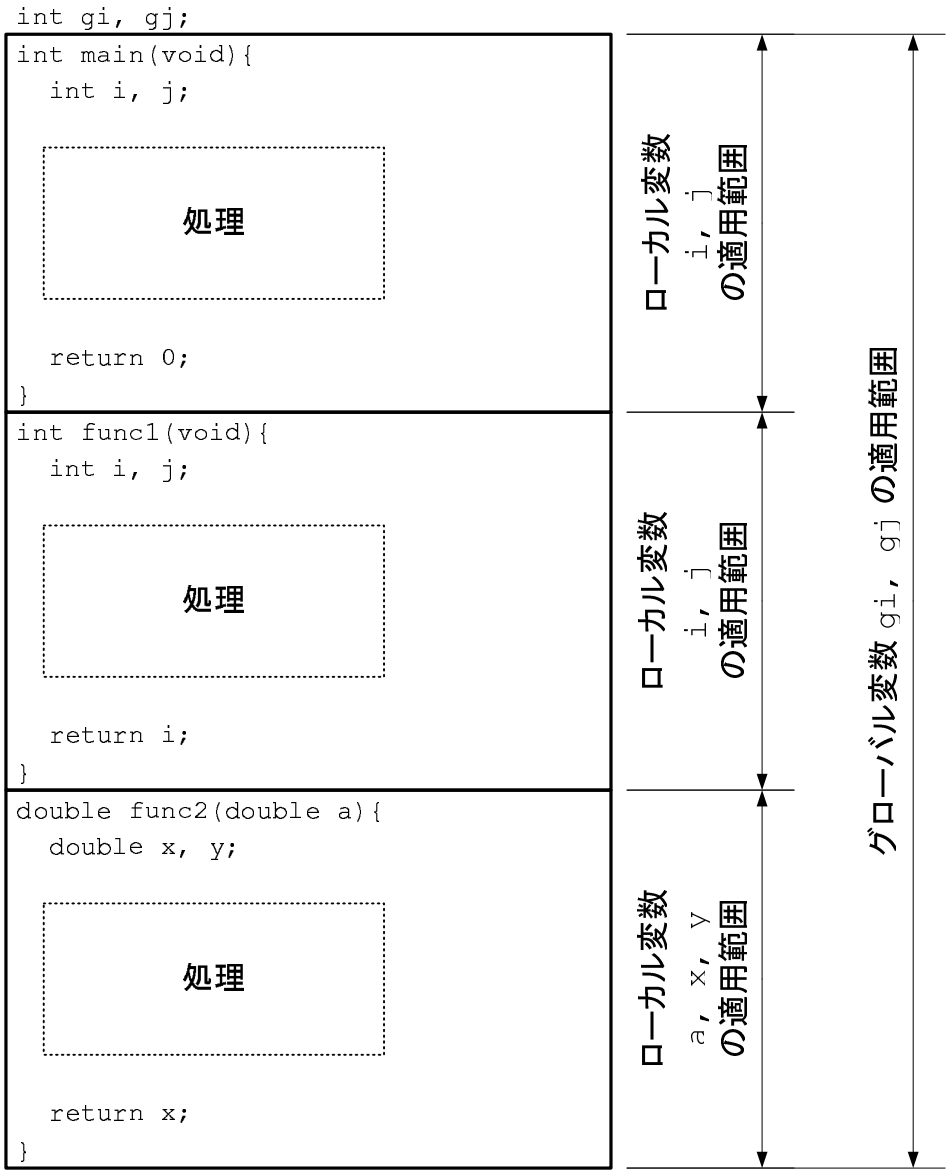

図 4: グローバル変数とローカル変数