第１３話（Ｃ言語のポイント）

タヌキ、コンピュータ言語は数多くあるのですが、それらは手続き型言語とオブジェクト指向言語に大別されます。オイラが手続き型言語の中でも最も重要だ、と考える言語は、Ｃ言語なのだ。Ｃ言語はアセンブラ言語に近い言語だし、オブジェクト指向言語にも近い言語なのだ。さらに、Linux の各種 OS は、Ｃ言語でプログラミングされているのだ。
Ｃ言語を勉強する時も、変数や配列とか、既存のプログラムを利用するインクルードなどの基本から始めるのが定石だが、それが必要だと思うタヌキは、解説本を買って勉強してくれ。

Ｃ言語のポイントはずばり、
ポインタ変数と構造体だ！

ポインタ変数は変数名の前にアスタリスク（*）が付くよ！

int　*a;

ポインタ変数の整数定義だ。でも普通の数値変数のように

a = 56;（これはダメ）

エラーになります。

a = メモリのアドレス;（これはＯＫ）

ただし、メモリに実際に存在する１個のアドレスしか代入できません。適当なアドレスはダメです。

```
int   *a;
```

以下のようにすれば、ポインタ変数に直接アドレスを代入することは可能です。

```
a = (int*)0x0000ff11
```
（16 進数 8 桁）

しかし、直接代入アドレスがプログラムの途中のアドレスならば暴走したり、OS 自体が破損する危険がありますので、普通は行いません。

基本的には、以下のように変数を経由してアドレスを代入します。

```
int b = 20;

a = &b;
```

となります。

右のイメージ図を使うと、

＆ｂの値は［00000100］になり、ａの値も先頭のアドレス［00000100］になります。

メモリのイメージ図

アドレス	変数名ｂ	
(0x)00000100	00000000	整数４バイト
00000101	00000000	
00000110	00000000	
00000111	00010100	（10 進の 20）

変数ｂは変数として使えますし、変数ｂの値の記憶場所のアドレスはポインタ変数のａに保存されているので、必要に応じて利用することができます。

キツネ、了解！

int（整数）型で指定した変数ｂは、メモリに 4 バイトの記憶領域を確保し、その記憶領域の先頭のアドレスが 16 進数（0ｘ）で表わすと［00000100］であるとすると、そのアドレスがポインタ変数ａに代入されるということだな。あくまでも値２０（２進数の 00010100）が保存されるので無く、その場所（アドレス）が保存されるということだな。OK、OK、わかった。

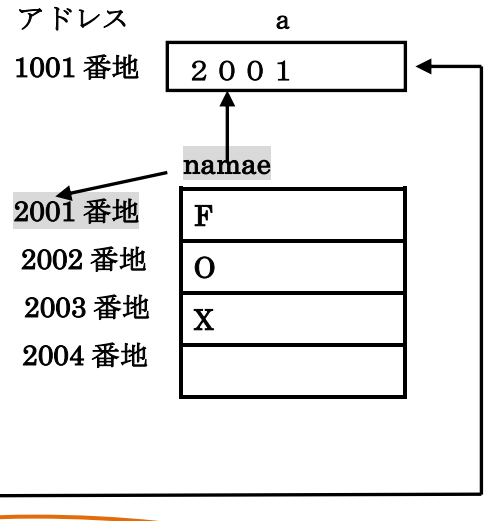
整数 (int) 型のポインタ変数の話をしたので、次は文字 (char) 型のポインタ変数の話をしなければならないな。
文字型宣言の char は英語の character(文字)からきています。
また、それに併せて配列の話もしなければならないな。
C 言語で扱われる配列はポインタ変数でもあるのだ。
ポインタ変数 char *a は変数 char b に対応し、変数 b の 1 バイトに対応しています。int b は 4 バイトですので char *a を int b のポインタ変数として使うことはできません。エラーになります。
文字型のポインタ変数には、文字型の変数、整数型のポインタ変数には、整数型の変数というように型は揃えなければなりません。

C 言語の変数宣言

```
char   *a   ;

char namae[4]   ;

   namae[0]=”F”;
   namae[1]=”O”;
   namae[2]=”X”;

a = namae ;
```

※　ポインタ変数 a に入る値は何？

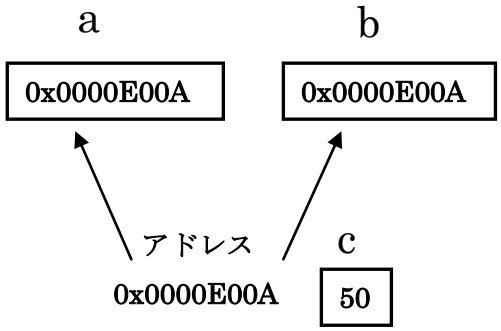
メモリのイメージ図



なるほど、左のプログラムは、文字型のポインタ変数宣言と文字型の配列宣言をする。次に配列の添え字 0 には “F” を 1 には “O”、2 には “X” を代入する。その 3 個の文字が、メモリに記憶されるイメージが右のイメージ図だ。その記憶領域に便宜上の 2001 から 2004 番地までのアドレスを付けているのか。ポインタ変数*a は配列とは別の領域に作られ、そのアドレスを 1001 番地としているのか。ということは、配列名 namae は配列の最初のアドレス（2001 番地）を保存しているポインタ変数なのか。だから namae の値を a に代入するから a の値は［2001］になるのだな。納得！

ポインタ変数を使うと、以下のような奇妙なことがおきるよ。
タヌキ、わかるかな。

```
int   *a, *b;
int   c;

c = 50;
a = &c;
b = a;
```



このプログラムを実行すると*a と*b の値は 50 になるよ！

わかる、わかる。
＆c で 50 の値を記憶している記憶領域のアドレスをポインタ変数 a に代入し、さらに a からポインタ変数 b に変数 c のアドレスを渡しているのだな。こうすれば、アドレスを通して c の値 50 を共有することができるのだな。また、普通の変数でも＆を付けることによって変数の記憶領域のアドレスを取り出すことができるのだ。奇妙だけど、勉強になった。
オイラ、チョット考えたのだが、このアドレス渡しを使えば、COBOL と FORTRAN という異なる言語間のデータの受け渡しが可能になるのでは。当然、COBOL と C 言語間でも可能だろうけど。

タヌキ、おめえ、進化したな！
タヌキの言う通り、アドレス渡しを使えば、異なる言語間のデータの受け渡しは可能だよ。実際に複雑な計算は C 言語で行い、結果は、COBOL で保管する、ということに使われているのだ。

さて、次は２番目のポイントの構造体の話だ。
構造体は、データをパッケージにしたものだ。
パッケージの中には、各種変数、ポインタ変数、
配列などを詰め込むことができ、そのパッケージ
は一活して扱うことができるのだ。

構造体

```
  struct        student{
(構造体)宣言   構造体名（型）
      char name[20];  メンバ変数
     文字型   配列（20 個、２０バイト）
      int kokugo;
     整数型
      int sansu;
     整数型
      }
```

形の宣言（形のレイアウト）

メンバ変数

student
name[0]
name[1]
name[2]
name[18]
name[19]
kokugo
sansu

こんな形はどうかね・・・・？
番地（アドレス）も何も与えられていない
有るのは形だけ。

構造体の特徴は、データ入れる枠組みだけ作っておく、という点です。その枠組みに拘束される形で、氏名：山田、国語：70、算数：80、氏名：鈴木、国語：60、算数：90 という具合にデータを入力し、実体（インスタンス）となることです。Java 言語のクラスと似ていますが、構造体は各種変数しか定義できません。それに対してクラスはデータと実行する処理（プログラム）を定義できます。

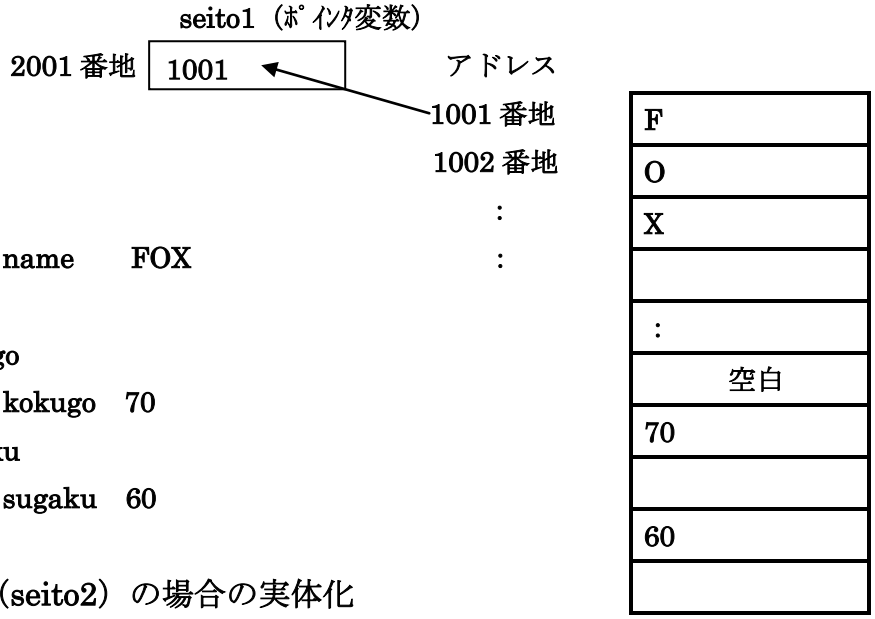
さて、次は構造体の実体化（インスタンス）の例を示すよ。

実体化例０１

struct　student　seito1 = { “FOX”,70,60} ;

（構造体の student）型　変数（seito1）

実体化して初めてアドレスが付与される

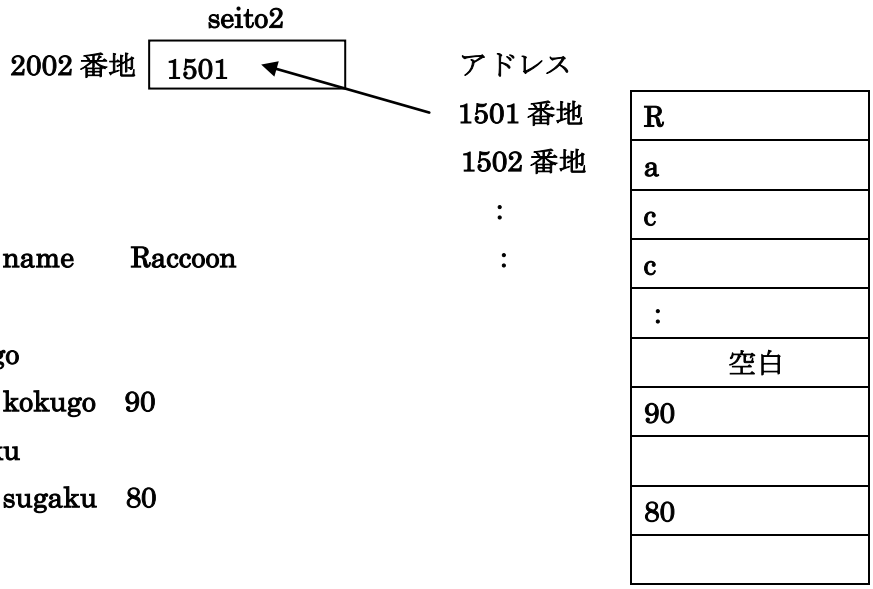


同様に生徒２（seito2）の場合の実体化

struct　student　seito2 = {“Raccoon”,90,80} ;

（構造体の student）型　変数（seito2）

実体化して初めてアドレスが付与される



なるほどね。１人分のデータが一括で入力できるのは便利だ。入力ミスも若干防げそうだ。
コンピュータは、使う人間に便利なように改良されているから構造体というものを考え出しても不思議は無いけどね。

前にも言ったように構造体とクラスは違うけれども、構造体と関数ポインタを使えば、疑似クラスを作ることができます。クラスは（データ＋処理）をカプセル化したものです。構造体は、メンバー変数（つまりデータ）のみです。関数は処理の塊ですから構造体に関数を付加すれば疑似クラスを作ることが可能なのは理解できますね。疑似クラスの説明をするとC言語が難しく見えてくるので、止めるね。ここで理解してほしいのは、手続き型言語のC言語がオブジェクト指向言語のC++やJava言語に繋がっている、ということなんだ。

次は第１４話でクラスを含めたJava言語のポイントの話をしようか。