J3プログラミング　　ポインタ入門

第一章　ポインタとは何か

ポインタについて話す前に、変数の知識を、おさらいしておきます。

１．０　変数の話

計算機を用いてデータを処理する時、我々はプログラムを書きます。プログラムを習い始めた頃、”データを格納する入れ物を変数と呼ぶ”と習いました。それは直感的に正しいのです。しかしみなさんも、３年ですし、もう少し詳しく変数について説明しても良い頃でしょう。

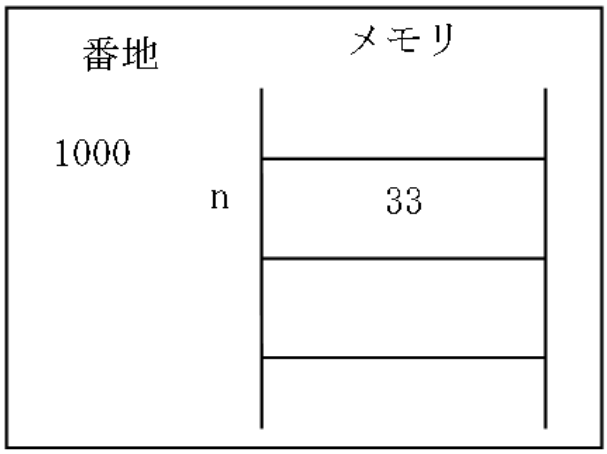
変数とは、端的に言えば、メモリ上に設けられた、データを保存する領域です。Ｃ言語では、データを保存する領域（所在地、番地）をいちいちプログラマが覚える必要はありません。Ｃ言語では

”記憶領域に愛称をつけて、この愛称をプログラム中で記述する”

という方式が採用されており、その愛称＝記憶領域の所在地となっています。例えば、メモリ上のいずれかの記憶領域に３３という整数値を記憶させる処理を考えます。この処理は

ナントカ番地に、ナントカバイトの広さの領域をとって、そこに３３を入れろ

と命令する必要があります。しかし、Ｃ言語では変数ｎを宣言し、int n; と記述すれば、コンパイラが勝手にｎのために番地と領域を割り当ててくれます。次に、変数ｎに３３を入れろ(n=33;)と命じて、割り当てた領域に、３３を格納します。例えばｎに対して、１０００番地が割り当てられたとすれば、１０００番地からはじまる領域に３３が格納されます。



概念図：変数宣言時の番地と領域の割り当て

このように、Ｃ言語プログラムにおける変数は、実体は

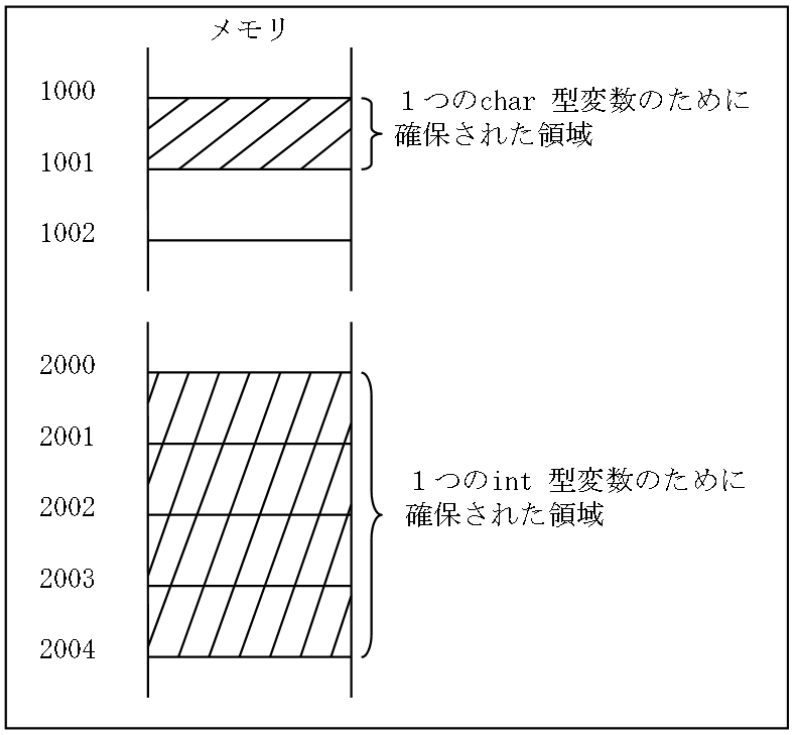
”メモリ上に割り当てられた記憶領域”

であると考えて差し支えないでしょう。この記憶領域にデータを出し入れする為のテクニックをこれから学んでいきます。習得するにはたくさんの時間と少しの努力が必要です....。まあまあ、あせらずに、準備から。

１．１　変数について～型の概念～

Ｃ言語の基本的な変数は、各々の表現能力に応じて名前が付けられています。例えば、８ビットのサイズの整数値（-128～127)を扱う変数を char 型 と言い、３２ビットのサイズの整数値(-2147483648～2147483647)を扱う変数を int 型､同じ３２ビットサイズの０以上の整数値（0～4294967295）を扱う変数を unsigned int 型（以後 unsigned 型と略する）と言っています。

大きな範囲の数を扱う変数に対しては、当然大きな記憶領域を必要としますので、int 型変数の為に確保された記憶領域は、char 型変数のために確保された記憶領域より広くなります。



概念図：型の違いによる記憶領域の大小

int 型,char 型のほかにも、double 型と言うのもあります。これらデータ型は基本的に与えられているモノですが、自分で型を作ることも出来るので扱える型は無限にあります。

この先、新しい型をいろいろ紹介していく予定です。
注）上に述べたデータ型の範囲は、計算機演習室のマシンを利用する場合に当てはまる事です。環境が異なれば、事情は変わります。自分の所有する処理系が、それぞれのデータ型にどれくらいのサイズを割り当てているか（たとえば、int 型変数に何バイト使っているか）を知っておくのも意味のあることです。

１．２　変数の大きさ～sizeof～

上に述べたように変数の型は、名称に対応してサイズが割り当てられており、そのサイズに応じたメモリ上の領域を占めています。データ型のサイズを知りたいとき、sizeof という演算を行います。

例題）char 型変数のサイズを表示せよ（単位はバイト）。

```
main(){
        printf("%d¥n", sizeof(char));
}
```

もしくは

```
main(){
        char    a;
        printf("%d¥n",sizeof(a));
}
```

といった具合に使います。

例題）char 型の成分１０個の配列を宣言し、この配列のサイズを求めよ（sizeof 演算子は、配列名に対しても有効です）。

```
main(){
        char a[10];
        printf("%d¥n",sizeof(a));
}
```

練習問題 1.1.1

（１）上の例を入力し、実行させよ。

（２）unsigned 型変数、int 型変数、double 型変数、成分１０個の int 型配列についてもそれらのサイズを求めよ。

上の例題と練習から int 型のサイズは４バイト、char 型は１バイトとわかります。

１．２　型変換－その１－

例題）　異なる型の変数同士で代入を行い、どのように処理されるか見てみよう。

```
main(){
        int         i=256;
        double    d=3.14;
        d=i;
        printf("%f¥n"d);
        i=3.14;
        printf("%d¥n",i);
}
```

実行結果

256.000000

3

異なる型の変数同士で代入を行った場合、上のように、型は適当に自動変換されます。計算式内でも型変換が自動的に行われる場合があります。例えば

int / double

というタイプの計算をした場合は、分子も分母も double 型として解釈され、計算が行われます。

例）

```
main ()
{
   int    j = 2;
   double d=1.5;
   printf ("%d / %f = %f¥n", j, d, j/d);   }
```

実行結果

2 / 1.500000 = 1.333333

同様に double / int の結果は double になります。

１．３　型変換－その２－

１．２節で述べた型変換の条件が満たされると、自動的に型変換がなされます。従って、int / int という計算の場合は小数で答えがでるのではなく、小学生で習った割り算になります。

例）

```
main ()
{
```

```
   int i = 3, j = 2;
   double d;
   d = i / j;
   printf ("%d / %d = %f¥n", i, j, d);
}
```

結果

3 / 2 = 1.000000

しかし、プログラマーの都合で、3/2 は、1.5 になってほしいこともあるでしょう。そんなときは、手動の型変換（キャスト）を使います。

例）

```
main ()
{
   int i = 3, j = 2;
   double d;
   d = (double)i / j;
   printf ("%d / %d = %f¥n", i, j, d);
}
```

実行結果

3 / 2 = 1.500000

上のプログラムの

(double)　i

という式は、”int 型の変数 i を、ここだけ double 型として扱え”　という意味です。すると (double)i / j　は前節で述べた型変換の規則により、 結果は double 型になり、上記の結果になります。

練習問題 1.3.1

以下の操作を行った場合、変数の値はどう変わるか。プログラムを作って確認せよ。

（１）double 型の変数を int 型にキャストした場合

（２）int 型の変数に double 型の変数を代入した場合

（３）int 型変数に char 型変数を代入した場合、

練習問題 1.3.2

以下のプログラムを実行してみよ。何故そのような動作をするか考えよ。

```
main(){
```

```
        int i;
        char a[20];
        unsigned int b[20];
        gets(a);
        for (i=0;i<=19;i++) b[i]=a[i];
        printf(“%s”,b);
        printf(“¥n”);
        for (i=0;i<=19;i++) printf(“%c”,b[i]);
}
```

１．４　アドレス演算子

　Ｃ言語では変数が宣言されると、メモリ上の領域が、変数のサイズ分だけ確保されます。宣言した変数がメモリのどの番地に確保されたかを知りたいときはアドレス演算子（&）を用いると便利です。

例）

```
main(){
        int         n;
        printf("%x¥n",&n);  }
```

実行結果

7ffff9f4

上のプログラムで &n は、上の行で確保された変数ｎのアドレスを意味します。printf("%x¥n",&n)　でその値を１６進数で表示しました。

（注）アドレスを１６進数で表示する為の、printf の書式は本来、

```
        printf("%p",&n)
```

ですが、ここでは既知の書式を使いました。今後はアドレスを表示するときはこの書式(%p)を使ってください。次のプログラムは、大変重要な示唆を含んでいます。

例）

```
main(){
        int       n;
        printf("%p¥n",&n );
        printf("%p¥n",&n + 1 );
}
```

実行結果

0x7ffff9f4

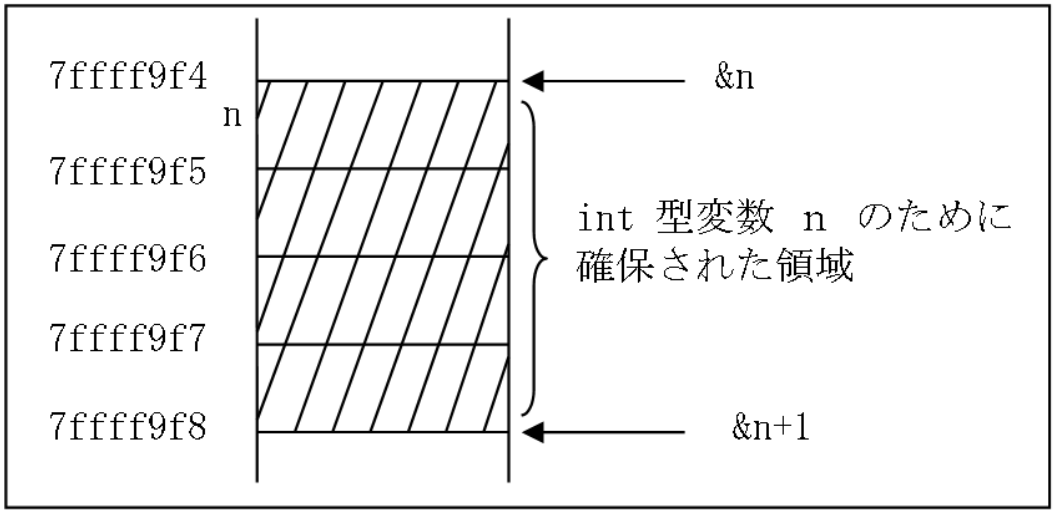
0x7ffff9f8

ｎのアドレスに１を足しただけなのに、結果は４増えています。

これはなぜでしょうか..... ？

実は、コンパイラは以下のように解釈しています。

&n + 1 とは「変数ｎの一つ隣にある int 型変数」のアドレスを意味する。



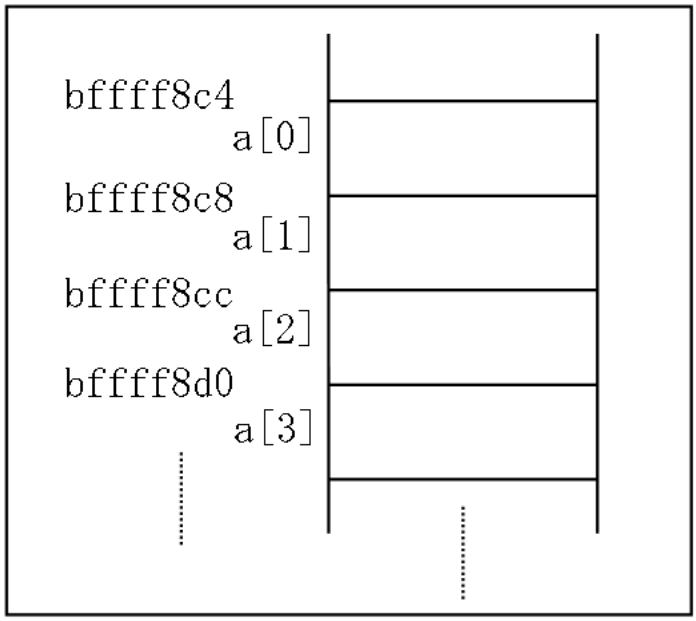
図：&n+1＝&n＋sizeof(n)

実際には、上のプログラムでは、変数ｎの隣に int 型変数は確保されていません。しかし、あたかも存在するものとして計算されます。そう言うわけでｎのアドレスに、ｎのサイズ（ すなわち sizeof(n) ）を足した値が、&n +1 の値になります。

練習問題 1.4.1　int 型変数を２個宣言し、それらのアドレスを画面に表示するプログラムを作成せよ。

練習問題 1.4.2　int 型で成分１０の配列 a を宣言し、a[0]～a[9]のアドレスを表示するプログラムを作成せよ。

練習問題の実行結果から、メモリ上の a[0]～a[9]の位置関係が理解できたでしょうか。



図：配列宣言による変数の連続的確保

ここから分かるように、int 型宣言を２度行っても、変数がメモリ上連続的に確保されるとは限りません。しかし、配列を宣言すると、int 型変数が連続的に確保されます。

１．５　アドレスを格納する為の変数

　前節では、アドレス演算子をもちいて int 型変数のアドレスを見てみました。アドレスも数値ですし、上で見たような特殊な演算をするのですから、

　　アドレスを格納する為の変数

が必要になります。アドレスを格納する為の変数の型を

　　ポインタ型

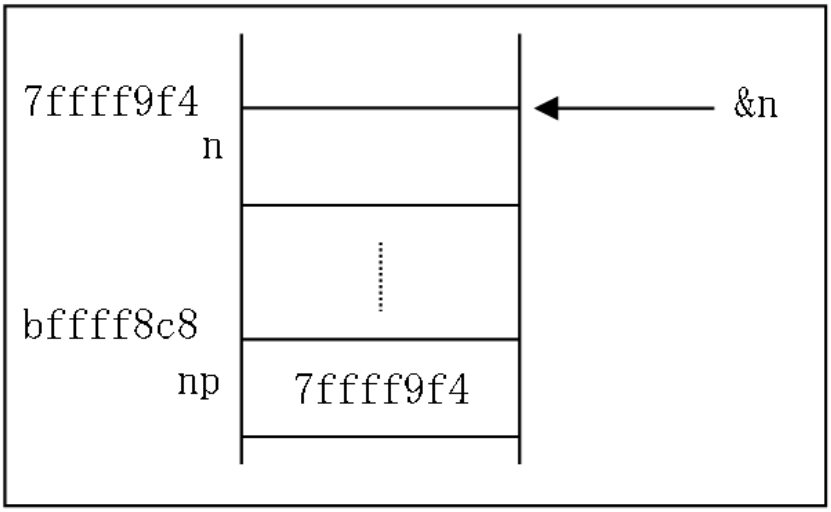
と言います。次の例では、ポインタ型の変数 np を宣言し、np に変数 n のアドレスを代入しています。

例）

```
main(){
        int     n;
        int     *np;    /* ポインタ型変数　np の宣言 */
        printf("%p¥n",&n);
        np=&n;
        printf("%p¥n",np);
}
```

実行結果

0x7ffff9f4
0x7ffff9f4



図：ポインタ型 np は変数 n のアドレスを格納するための変数

例)

```
main(){
        int     n;
        int     *np;
        printf("%p¥n",&n);
        np=&n;
        printf("%p¥n",np+1);
}
```

結果
0x7ffff9f4
0x7ffff9f8
注)
ポインタ型変数は、実在する変数のアドレスを格納する為の変数です。ですから、ポインタ型変数に、実在しない変数のアドレスを代入しても無意味です。
例)

```
main(){
        int     n;
        int     *np;
        np=&n +2;       /*  実在しない変数のアドレスが代入されている  */
}
```

１．６　ポインタ型変数のサイズ

例１）

int へのポインタ型変数、double へのポインタ型変数、char へのポインタ型変数を宣言し、それらのサイズを表示させよ。

```
main(){
        int     *ip;    /* 好きな名前を付けて良い　*/
        char    *cp;
        double  *dp;
        printf("sizeof(ip)=%d ¥n",sizeof(ip));
        printf("sizeof(cp)= %d ¥n", sizeof(cp));
        printf("sizeof(dp)= %d ¥n",sizeof(dp));
}
```

実行結果

```
sizeof(ip)=4
sizeof(cp)= 4
sizeof(dp)= 4
```

この例から

　　任意の変数へのポインタ型変数はすべて同サイズ

という類推ができます。考えてみれば、どれもアドレスを格納するのですから同じサイズで良いはずです。計算機演習室の環境では、

　　ポインタ型変数と、int 型変数は同じサイズ（で４バイト）

のようです。

例２）

（１）int 型変数 n を宣言して適当な値を代入せよ。

（２）変数 n のアドレスを適当なポインタ型変数 x に代入せよ。

（３）変数 x の値とアドレスを表示せよ。

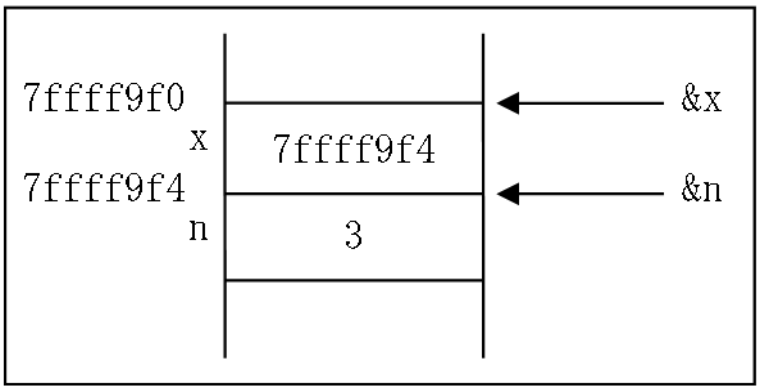
```
main(){
        int     n=3;
        int     *x;
        x = &n;
        printf("x= %p ¥n&x = %p ¥n",x,&x);
}
```

実行結果

x= 0x7ffff9f4

&x = 0x7ffff9f0

例２のプログラムにおけるメモリの利用状況を図示します。



概念図：例２におけるメモリの利用状況

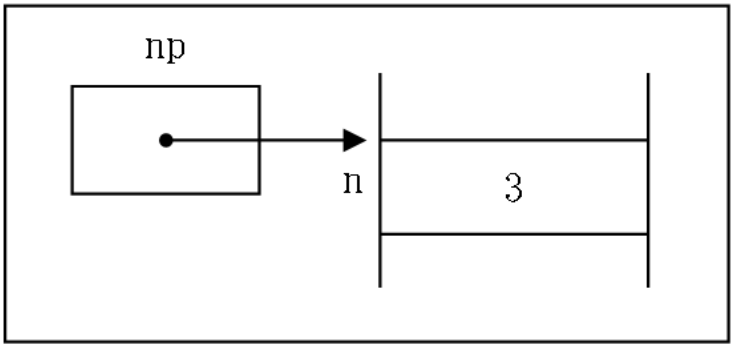
１．７　ポインタ変数が指す内容

変数の型とアドレスが分かっていれば、その変数の占めている領域が分かります（当たり前ですが、これば重要。変数の型とアドレス、どちらの情報が欠けても、変数の占めている領域を特定することは不可能です)。すなわち、変数の型とアドレスのみから、そのアドレスに格納されているデータを入手する事が出来るわけです。本節ではその方法を学びます。

ポインタ型変数、例えば int へのポインタ型変数は、int 型変数のアドレスを格納しています。この状況を

ポインタ型変数が int 型変数を指している

と呼びます。次の例題の方法を使えば、指される int 型変数の値を調べることができます。



概念図：ポインタ型変数 np が int 型変数 n を指す

例題）ポインタ型変数が指す変数の内容を表示せよ。

```
main(){
        int     n=3;
        int     *np;
        np=&n;                  /*  np にはｎのアドレスが代入されている  */
        printf("%d¥n",*np);     /*  *np と書けば、ｎの値を意味する  */
}
```

*np で、np が指している変数の値を意味しています。この記述は、ポインタ型変数の宣言

```
        int     *np;
```

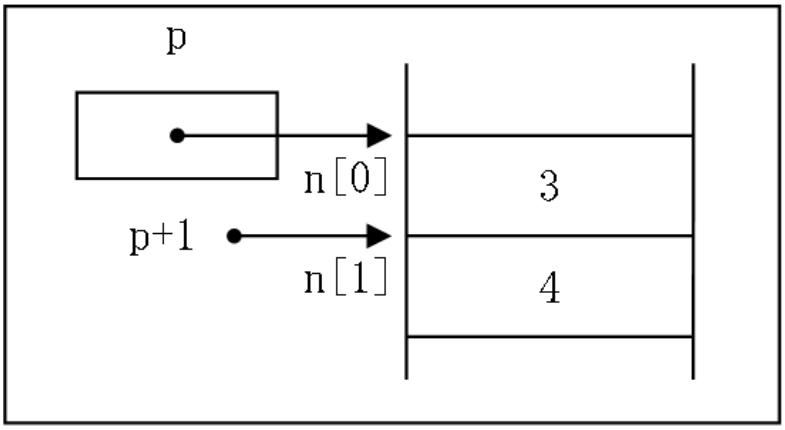
を、つい思い出してしまいます。*という記号は、

宣言で用いたときは、ポインタ型を意味し、
式の中で用いた場合は、ポインタ型変数が指す変数の内容を意味する

と覚えてください（＜==超重要です、忘れないように)。少し紛らわしいですが...。
さて、以下の問題は、これまでの内容の総まとめです。その前に例題を少し。

例題）ポインタ演算を用いて変数の内容を参照する
（１）int 型配列（成分２個） n を宣言する。n[0]=3,n[1]=4 とする。
（２）int へのポインタ型変数 p を宣言する。
（３） p に n[0]のアドレスを代入する。
（４）n[1]の値を２通りの方法で表示する。
解答例)

```
main(){
        int     n[2];
        int     *p;
        n[0]=3;
        n[1]=4;
        p=&n[0];
        printf("%d¥n",n[1]);
        printf("%d¥n",*(p+1));          /* p+1 は p が指している int 型変数 n[0]の
                                           隣の int 型変数（すなわち n[1]）
                                           のアドレスになる  */
}
```



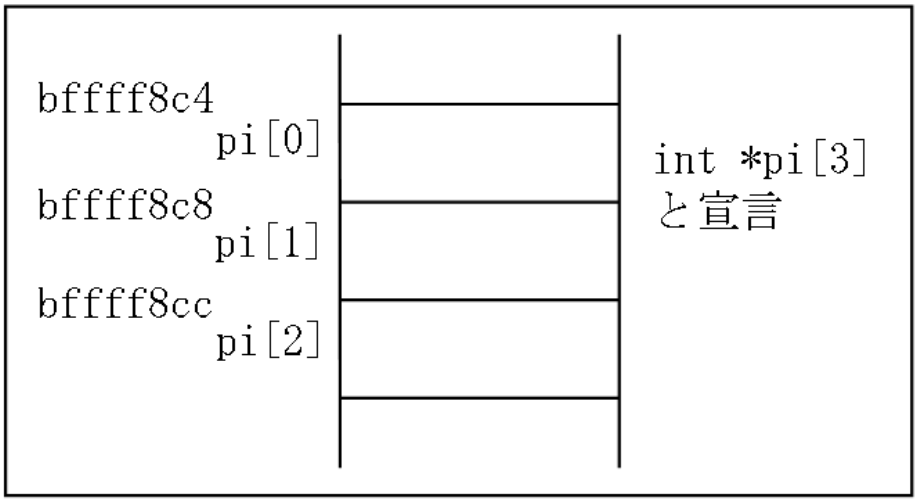
図：p+1 は p の隣を指す

実行結果

4

4

以後、int へのポインタ型を int *型と呼ぶことにします。同じ原理で、unsigned 型へのポインタ型は unsigned *型と言います。以後、char *型、double *型などという言葉がでてきても驚かないでください。int *型も、他の変数と同じく、メモリに確保される 4 バイトの領域として考えてよいので、これをいくつか連続して確保することも考えられます。



概念図：int *型の配列

例題）

（１）int 型変数 i,j を宣言する。適当に値を代入する。

（２）int *型の配列 pi（成分２個）を宣言する。

（３）pi[0]には i のアドレス、pi[1]には j のアドレスを代入。

（４） ２通りの方法で i と j の値を表示する。

解答例）

```
main(){
        int     i=3,j=4;
        int *pi[2];
        pi[0]=&i;
        pi[1]=&j;
        printf("%d %d¥n",i,j);
        printf("%d %d¥n",*pi[0],*pi[1]);
}
```

実行結果

3 4

3 4

練習問題 1.7.1

以下のプログラムを批判せよ。

```
main(){
        int     *p;
        *p=97;
        printf("%d",*p);
}
```

練習問題 1.7.2　以下のプログラムを書け

（１）unsigned 型変数を３個（a,b,c)宣言し、

それぞれに以下の値を代入せよ。

a には b のアドレスを unsigned に型変換した値（すなわち(unsigned)&b)

b には c のアドレスを unsigned に型変換した値（すなわち(unsigned)&c)

c には a のアドレスを unsigned に型変換した値（すなわち(unsigned)&a)

（２）変数 c の値を１６進数で表示したい。３種類の方法を考えよ。

ヒント：”unsigned 型へのポインタ型”に型変換する方法：

        (unsigned *)変数名

練習問題 1.7.3　以下のプログラムを書け。（５）以降はチャレンジとする。

（１）int 型の成分２個の配列 n を宣言する。n[0]=3,n[1]=4 とする。

（２）int *型変数（int へのポインタ型変数） y を宣言する。

（３） y には n[0]のアドレスを代入する

（４）n[1]の値を２通り以上の方法で表示する。
（５）int **型変数（int *型変数へのポインタ型変数） z を宣言する。
（６） z には y のアドレスを代入する。
（７））n[1]の値を３通り以上の方法で表示する
練習問題 1.7.4　以下のプログラムを書け
（１）unsigned 型の配列（成分２）を３個(a[2],b[2],c[2])宣言せよ。
（２）
a[1]には b[0]のアドレスを unsigned に型変換した値（すなわち(unsigned)&b[0])
b[1]には c[0]のアドレスを unsigned に型変換した値（すなわち(unsigned)&c[0])
c[1]には a[0]のアドレスを unsigned に型変換した値（すなわち(unsigned)&a[0])
を代入する。
（３）a[0],b[0]c[0]には好きな整数値を一つ代入
（４）c[1]の値を表示する方法を、３通り以上考えよ。
練習問題 1.7.5（チャレンジ）　以下のプログラムを書け
（１）unsigned 型の配列（成分２０）を３個(a[20],b[20],c[20])宣言せよ。
（２）
a[19]には b[0]のアドレスを unsigned に型変換した値（すなわち(unsigned)&b[0])
b[19]には c[0]のアドレスを unsigned に型変換した値（すなわち(unsigned)&c[0])
c[19]には a[0]のアドレスを unsigned に型変換した値（すなわち(unsigned)&a[0])
を代入する。
（３）a[0]～a[18],b[0]～b[18],c[0]～c[18]には好きな文字列を代入
（４）c[0]から c[18]の間に入れた文字列を２通りの方法で表示せよ。
（５）c[0]から c[18]の間に入れた文字列を３通り以上
　　の方法で表示せよ。

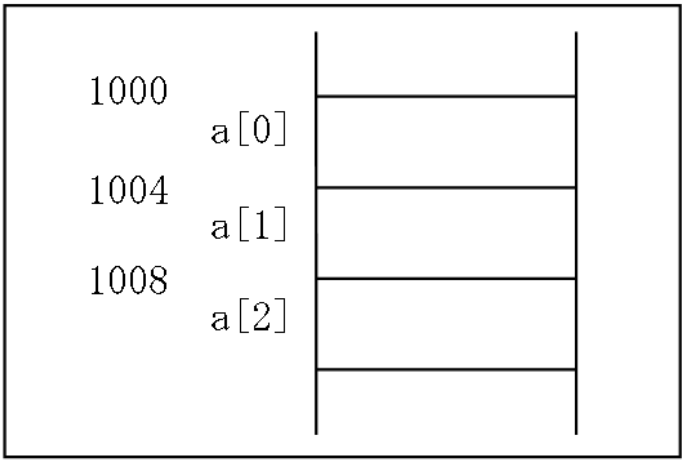
補足１）
ポインタ型変数（アドレスを格納する変数）の概念は、int や char 等基本的な型ばかりでなく、任意の型に対して有効です。例えば、”int *型変数”のアドレスを格納する変数（int **型）を考えることもできます。さらに” int **型変数”のアドレスを格納する変数（int ***型）も考えることができます。
例題）以下のプログラムの実行結果を予測せよ。

```
main(){
        int     i=3;
        int     *pi;            /*  pi は int *型変数                   */
        int     **ppi;          /*  ppi は int *へのポインタ型変数      */
                                /*  すなわち int **型                   */
```

```
        int     ***pppi;        /*  pppi は int **へのポインタ型変数      */
                                /*  すなわち int  ***型                   */
        pi=&i;
        ppi=&pi;
        pppi=&ppi;
        printf("%d",***pppi);
}
```

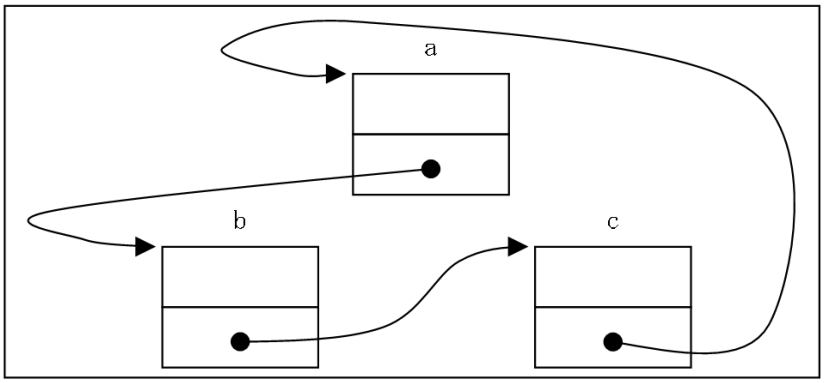
補足２）構造体について

すでに見てきたとおり、int 型の配列とは、int 型変数を、メモリ上に連続的に並べた型でした。メモリ上に連続して並んでいる事を、常に認識している必要は無く、int 型変数が一塊りになっている様な感覚で、配列を取り扱って構いません。



図：int 型の配列は連続して並ぶ

しかし、上の練習問題 1.7.4 のように、異なる型のデータを一塊りにして扱いたい場合も在ります。

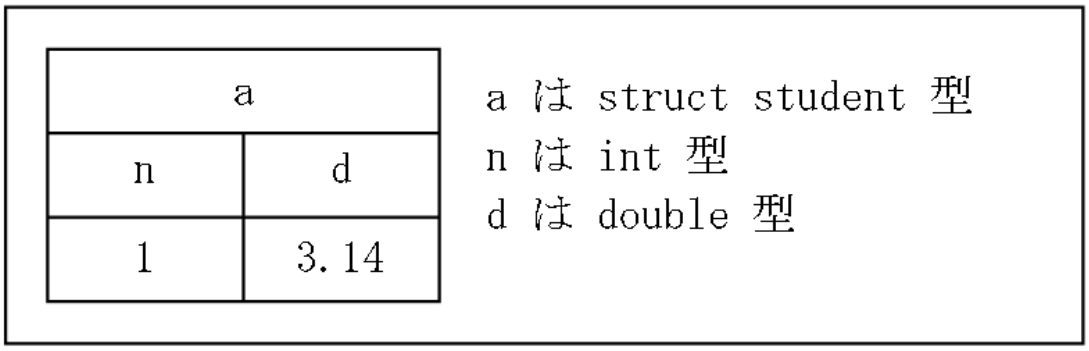


図：練習問題 1.7.4 の概念図

このような場合は、構造体のテクニックが必要です。構造体は
「異なるデータ型を一塊りにして、一つの新しいデータ型として扱う」データ型です。

例題）int 型と double 型をひとまとめにして扱う。

```
struct   student{                    /* struct student という新しい型を定義する   */
         int       n;                /* int 型と double 型で作った配列の様なモノ  */
         double    d;
};
main(){
         struct    student a;        /* struct student  型の変数 a を宣言        */
         a.n=1;                      /*  変数 a のメンバー n （int 型）に 1 を代入  */
         a.d=3.14;                   /*  変数 a のメンバー d （double 型）に 3.14 を代入*/
         printf("%d %f¥n",a.n,a.d);
}
```



上の例題における構造体の概念図

例題）int 型と int*型をひとまとめにして扱う。

```
struct   rensyuu{            /* struct rensyuu という新しい型を定義する   */
         int       n;
         int       *p;
};
```

```
main(){
        struct   rensyuu a,b,c;      /* struct rensyuu 型の変数 a,b,c を宣言する */
        a.n=1;
        b.n=2;
        c.n=3;
        a.p=&(b.n);
        b.p=&(c.n);
        c.p=&(a.n);
        printf("%p",*(b.p));
}
```

例題）char*型と unsigned 型と char*型をひとまとめにして扱う。

```
struct   names{              /* struct names という新しい型を定義する   */
        char       *name;
        unsigned gakuseki;
        char       *p;
};
main(){
        struct   names a,b,c;      /* struct names 型の変数 a,b,c を宣言する */
        a.name="wada";
        a.gakuseki=994154;
        b.name="wadawada";
        b.gakuseki=994155;
        c.name="wadawadawada";
        c.gakuseki=994156;

        a.p=b.name;
        b.p=c.name;
        c.p=a.name;
        printf("%s",b.p);
}
```

練習問題 1.7.6　以下のプログラムの構造体宣言部は何を意味するか考えよ。プログラムはどんな実行結果になるか。

```
struct   names{
        char       *name;
        unsigned gakuseki;
```

```
        struct names *next;
};
main(){
        struct   names a,b,c;
        a.name="wada";
        a.gakuseki=994154;
        a.next = &b;
        b.name="wadawada";
        b.gakuseki=994155;
        b.next=&c;
        c.name="wadawadawada";
        c.gakuseki=994156;
        c.next=&a;
        printf("%s¥n",c.name);
        printf("%s¥n",(*b.next).name);
        printf("%s",(*(*a.next).next).name);
}
```

----------以下すべてチャレンジ----------

上の練習問題 1.7.6 で見たように、「自分と同じ型の構造体」を指すポインタ型変数を、成分に持つ構造体を考えることが出来ます。このような構造体をつかった練習問題を解いてみましょう。

練習問題 1.7.7

```
struct   word_data{
        char        *word;
        struct word_data *next_left_data;
        struct word_data *next_right_data;
};
```

上の構造体 word_data をつかって、以下のプログラムをつくれ。
のべ１０個以下の単語（単語の文字数は２０以下）からなる英文をキーボードから入力し、構造体の配列（サイズ１０）に格納したい。
（１）word 成分には単語そのものを格納する。
（２）次の単語が、辞書式順序で言って後ならば、next_left_data にその単語（word_data）のアドレスを入れ、next_right_data には NULL を代入する。先ならば、next_right_data にその単語（word_data）のアドレスを入れ、next_left_data には NULL を代入する。
例文として this is a pen を考えると、以下のような概念図となる。

練習問題 1.7.7 の図