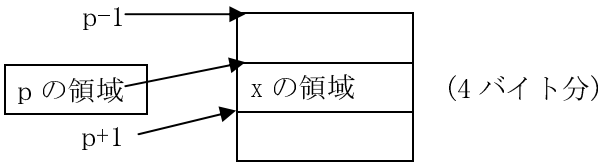
2006 年 12 月 21 日（木）実施

ポインタ演算と配列

前回の例題を通じて，32 ビット CPU の場合には，C 言語ではポインタ変数がメインメモリ上で 4 バイト分の領域を占めることを確認した。int 型変数の場合にもメインメモリ上で 4 バイト分の領域を占めているが，ポインタ変数に格納されているアドレスデータが int 型変数に格納されているデータと同一形式というわけではない。

しかしながら，ポインタ変数に int 型の値を加算または減算することができ，その結果は単にアドレスに整数値を加算または減算したものとは全く異なる意味合いを持つ。例えば，ポインタ変数 p に対して，p+1，p−1 等を**ポインタ演算**と呼び，p が指しているデータのデータ型によって演算結果が指すアドレスが異なる。

例）int x;
　　int *p;
　　p = &x;
　　の場合



（4 バイト分）

また，ポインタ演算の結果がアドレスとなることから，演算結果をポインタ変数に代入することも可能である。これは例えば，ポインタ変数 p に対して，p++，p−− 等が可能であることを意味している。

ただし，ポインタ演算は指しているデータの格納領域の範囲に注意して用いないと，大変危険である。例えば，ポインタ演算のうち，減算によって変数の格納領域を超えてプログラムの格納領域を書き換えてしまったり，甚だしい場合には，システム領域にアクセスしてしまったりする場合がある。UNIX 系の OS であればユーザプログラムの一部が壊れて暴走しても，そのプロセスだけを切り捨てられるメモリ保護機能があるが，OS によってはメモリ保護が不完全なものがあり，ユーザプログラムの暴走はシステムダウンを引き起こす場合がある。

ポインタ演算が用いられる典型的な例としては，ポインタ変数が配列を指す場合が挙げられる。配列名が配列の 0 番の要素の先頭アドレスを表すことから，これをポインタに代入して，i 番の要素の領域を ポインタ変数名 + i で指すことで，for 文等に於いて便利な利用が可能となる。

特に，char 型の配列に文字列を格納して，ポインタ変数に配列の 0 番の要素の先頭アドレスを代入した場合には，ポインタ変数で文字列を扱うことで，配列そのものを操作するよりも便利な利用が可能となる。

なお，文字列リテラルは，その文字列が格納されている領域の先頭アドレスを表すので，次の例のようにポインタ変数への代入が可能である。この場合には，配列の初期化の際のような文字列の文字数+1 の領域を新たに確保することは行われない。

例）char *p = "Programming Language C";

例題 1

次のプログラムを入力し，翻訳・編集して実行形式のファイルを作成し，実行せよ。ここで，

ソースプログラム名は prog10-1.c とする。

```
/* prog10-1.c */
#include <stdio.h>

int main(void)
{
    int x;
    int *p;

    p = &x;
    *p = 100;

    printf("  x の値は%d¥n", x);
    printf(" &x の値は%p¥n", &x);
    printf("  p の値は%p¥n", p);
    printf("p-1 の値は%p¥n", p-1);
    printf("p+1 の値は%p¥n", p+1);

    return 0;
}
```

例題 2

次のプログラムを入力し，翻訳・編集して実行形式のファイルを作成し，実行せよ。ここで，ソースプログラム名は prog10-2.c とする。

```
/* prog10-2.c */
#include <stdio.h>
#define MAX 5

int main(void)
{
    int i, x[MAX];
    int *p;

    p = x;

    for (i=0; i<MAX; i++)
    {
        *(p+i) = i+1;
        printf("  x[%d]の値は%d¥n", i, x[i]);
        printf("*(p+%d)の値は%d¥n", i, *(p+i));
        printf(" &x[%d]の値は%p¥n", i, &x[i]);
        printf("  p+%d の値は%p¥n", i, p+i);
    }

    return 0;
}
```

【解説】

(p+i)はポインタ演算 p+i の結果が指す先，即ちこの例では配列 x の i 番の要素がメインメモリ上で占める領域の中身を表す。従って，(p+i) = i+1; は x[i] = i+1; と同じ働きをする。

※ *p+i は*p を参照してその値に i を加算するので，この例では x[0]+i と同じ働きをする。

例題 3

次のプログラムを入力し，翻訳・編集して実行形式のファイルを作成し，実行せよ。ここで，ソースプログラム名は prog10-3.c とする。

```
/* prog10-3.c */
#include <stdio.h>

int main(void)
{
    char title1[] = "Programming Language C";
    char *title2 = "Programming Language C";
    char *p;

    for (p=title1; *p!='¥0'; p++)
        printf("%s¥n", p);

    for (p=title2; *p!='¥0'; p++)
        putchar(*p);

    return 0;
}
```

【解説】

1. 配列の初期化の際に，代入するデータの要素数（この場合には文字数+1）を以て配列の要素数として設定することができる。この場合，[]内は空とする。
2. putchar は 1 文字を画面に表示するライブラリ関数である。

演習 1

引数として,char 型ポインタ変数を 2 つ持つユーザ定義関数(1 つは書き換わらないよう,const 型修飾子を付ける) char *cpystr(char *a, const char *b) を次のように定義する。この関数は b に複写元，a に複写先の文字列を指すポインタを渡し，a の値を戻す仕様となっている。この関数を利用して，ポインタ変数 mesg が指し示す文字列"Hello World!"を別のポインタ変数 p に複写し，関数の戻り値及び p を printf で%s によって表示するプログラムを作成せよ。ここで，ソースプログラム名は ex10-1.c とする。

```
char *cpystr(char *a, const char *b)
{
    char *c = a;     /* a のアドレスは変えずに指し示す中身だけ書き込んでいくため */

    while (*b)       /* b の指す文字が'¥0'でない間は継続 */
    {
        *c = *b;
        c++;
        b++;
    }                /* この while 文は，while (*c++ = *b++)とも書ける。 */
    *c = '¥0';

    return a;
}
```

【ヒント】ポインタ変数 p が指し示す char 型の配列として充分大きな要素数のものを用意する。