第５回

〔３〕ポインタの演算

例題１－２６

文字型変数と実数型変数へのポインタ変数を pc，pf として，それぞれを増分（+1）したとき，ポインタ変数の値がどうなるかを確認するプログラムをつくりなさい。

▼出力結果１－２６

```
pc = 12FF7C,    pf = 12FF74
++pc = 12FF7D, ++pf = 12FF78
size of char  = 1
size of float = 4
```

考え方

ポインタの演算として，増分が指定されているが，ポインタの演算をまとめると，次のようになる。

①ポインタをあるデータのアドレスや 0 で初期化する。
②ポインタを増分または減分する。
③ポインタに整数を加減算する。
④二つのポインタを比較する。
⑤同じデータ型を示す二つのポインタを減算する。

この例題では，プログラムの構成ではなく，演算後のポインタの値に注目する。

▼プログラム１－２６

```
01      /* E1-26 */
02      /* ポインタの演算 */
03      #include <stdio.h>
04
05      main()
06      {
07          char c, *pc;
08          float f, *pf;
09          pc = &c;
10          pf = &f;
11          printf("  pc = %X,    pf = %X¥n", pc, pf);
12          ++pc;
13          ++pf;
14          printf("++pc = %X, ++pf = %X¥n", pc, pf);
15          printf("size of char  = %d¥n", sizeof(char));
16          printf("size of float = %d¥n", sizeof(float));
17      }
```

《単項演算子 sizeof の活用》

Ｃには，任意のオブジェクトのサイズを計算するのに使える単項演算子 sizeof がある。式

```
    sizeof オブジェクト
```

と

```
    sizeof (型名)
```

は，指定したオブジェクトまたは型のサイズをバイト数で表した整数を与える。

例えば，

```
    int i, j, k, ii[20];
    i = sizeof(int);
    j = sizeof i;
    k = sizeof ii;
```

はそれぞれどんな値になるか考えてみなさい。

さらに，ポインタ変数のサイズも

```
    sizeof(char *), sizeof(int *), sizeof(float *)
```

を用いて調べてみましょう。

練習問題１８

１８−１．double 型へのポインタ変数 p が宣言されていて，適当なアドレスで初期化されているとします。このとき，このポインタ変数 p に 2 を加えると，p の値はどうなるでしょう。この動作を確認するため，次のコードを完成してみなさい。その上で，p の値の変化がどうしてそうなるかを説明しなさい。

```
/* Q18-1 */
/* ポインタ変数の加減算 */
#include <stdio.h>

main()
{
    double *p;

    ここに適当なコードを挿入して下さい

    printf("元のアドレス = %x¥n", p);
    p += 2;
    printf("2 を加えた後  = %x¥n", p);
}
```

１８−２．int へのポインタ変数 ip を以下のように宣言して，初期化する。

```
int x = 1, y;
int *ip;
ip = &x;
```

このとき，下記[1]-[5]の場合について

- ポインタ ip そのものがどう変化するか（あるいは変化しないか）
- ポインタ ip がさすものがどう変化するか（あるいは変化しないか）

を答えなさい。

```
[1] *ip = *ip + 10;
[2] *ip += 1;
[3] ++*ip;
[4] (*ip)++;
[5] y = *ip++;
```

〔４〕配列とポインタ

例題１－２７

初期化された整数配列の各要素をポインタ演算により順次出力させるプログラムをつくりなさい。ただし，合計もポインタを使って求めること。

▼出力結果１－２７

```
0      11      22      33      44
合計 = 110
```

考え方

ポインタの演算をすれば，任意の配列要素を参照できる。順次出力であるから，演算には増分演算子を用いる。

合計は，間接演算子*で値を累積する。

▼プログラム１－２７

```
01      /* E1-27 */
02      /* ポインタによる配列操作 */
03      #include <stdio.h>
04      #define SIZE 5
05
06      main()
07      {
08          static int digit[SIZE] = {0, 11, 22, 33, 44};
09          int i, *pi, sum = 0;
10          pi = digit;
11          for (i = 0; i < SIZE; i++) {
12              printf("%5d", *pi);
13              sum += *pi++;
14          }
15          printf("¥n 合計 = %d¥n", sum);
16      }
```

《ポインタと配列の強い関係》

配列の添え字を使って実行できる操作は，ポインタでもできる。

```
int a[10], *pa, i;

pa = &a[0];  /* pa = a; でも同じ */
```

このとき，配列とポインタは同じように扱える。例えば，a[i] への参照は，*(a+i) とも書けるし，さらに驚くべきことに，pa[i] や *(pa+i) とも同じ参照となる。すなわち，配列名 a はポインタのように扱えるし，逆に，ポインタ pa は配列名のように扱える。
大きな違いは，ポインタは変数であるため pa = a や pa++ は意味のある演算であるのに対して，配列名 a は変数でなく定数であるため a = pa や a++ のような構文は正しくない点である。このことをしっかり理解することが大切である。

例題１－２８

キーボードから文字列を入力すると，入力した順番と逆の順番で文字列を出力するプログラムを作りなさい。ポインタ演算を利用すること。また，文字列の入力には，ライブラリ関数である gets 関数を用いること。

▼出力結果１－２８

```
文字列を入力してください！
(abc+XYZ)*456 = ?↵
? = 654*)ZYX+cba(
```

考え方

二つのポインタ変数を用いる。ポインタ１を文字列の最初にセットし，ポインタ２を文字列の最後にセットして，ポインタ変数の差がなくなるまでポインタ２を減らしながら逆の順番で１文字ずつ出力する。

▼プログラム１－２８

```
01      /* E1-28 */
02      /* 逆順の文字列 */
03      #include <stdio.h>
04      #define SIZE 256
05
06      main()
07      {
08          char str[SIZE];
09          char *pa, *pb;
10          pa = pb = str;
11          printf("文字列を入力してください！¥n");
12          gets(str);      /* scanf("%s", str); とするとどうなるか */
13          while (*pa)
14              ++pa;
15          while (--pa >= pb)
16              putchar(*pa);
17          putchar('¥n');
18      }
```

練習問題１９

１９-１．100 個の整数型配列を宣言して，各要素に 2～200 の偶数値を格納し，配列要素の総和を求めて出力するプログラムをつくりなさい。ただし，配列への値の格納，総和の計算ともポインタを用いて実行すること。

１９-２．文字列を入力すると，英小文字を大文字に変換して出力するプログラムをポインタを用いてつくりなさい。但し，ライブラリ関数 toupper を用いないこと。また，文字列の入力には，ライブラリ関数である gets 関数を用いること。

《ヒント》英小文字を大文字に変換するには，その小文字に'A' - 'a'を加えればよい。勿論，これがうまくいくのは，ASCII コード表での英小文字と英大文字がそれぞれ順に並んでいることによる。

〔５〕ポインタの配列

例題１－２９

文字列を指すポインタの配列を宣言して,その内容を入れ換えるプログラムをつくりなさい。ただし,ポインタへのポインタを宣言して配列とポインタの両方の操作を行い,同じ結果が得られることを確認すること。

▼出力結果１－２９

```
ZERO->ONE->TWO->THREE->FOUR->
ZERO->ONE->TWO->THREE->FOUR->
FOUR->THREE->TWO->ONE->ZERO->
```

考え方

これは文字列の配列（2 次元）であると考えがちだが,1 次元の配列である。二つの間接演算子がついたポインタ変数を宣言しなければならない。複雑に見えるが,ポインタ変数が何を指しているのかを考えてみればよい。

▼プログラム１－２９

```
01      /* E1-29 */
02      /* ポインタの配列 */
03      #include <stdio.h>
04      #define SIZE 5
05      #define NL "¥n"
06
07      main()
08      {
09          int i;
10          char *ptr[SIZE], **pa, *pb;
11          pa = ptr;
12          ptr[0] = "ZERO";
13          ptr[1] = "ONE";
```

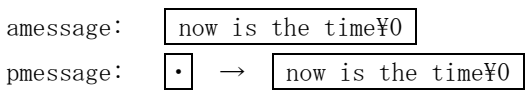
```
14          ptr[2] = "TWO";
15          ptr[3] = "THREE";
16          ptr[4] = "FOUR";
17          for (i = 0 ; i < SIZE ; i++)
18              printf("%s->", ptr[i]);
19          printf(NL);
20          for (i = 0 ; i < SIZE ; i++)
21              printf("%s->", *(pa+i));
22          printf(NL);
23          for (i = 0 ; i < SIZE/2 ; i++) {
24              pb = ptr[i];
25              ptr[i] = ptr[SIZE-1-i];
26              ptr[SIZE-1-i] = pb;
27          }
28          for (i = 0 ; i < SIZE ; i++)
29              printf("%s->", *pa++);
30          printf(NL);
31      }
```

《文字ポインタと文字配列の違い》

次の定義の間には重要な違いがある。

```
char amessage[] = "now is the time";    /* 配列 */
char *pmessage = "now is the time";     /* ポインタ */
```

次の図で考えてみよう。



pmessage は変数であり，このポインタを他の場所を指すように後で変えることができる。一方，amessage は初期値となる文字列"now is the time"を格納したメモリの先頭アドレスを指す定数である。

《例題》次の２つのコードを比較してみましょう。

```
#include <stdio.h>

main()
{
    char *p = "now is the time";
    char *q;

    puts(p);
    q = p;
    /* q[0] = 'N'; とできるだろうか */
    puts(q);
}
```

```
#include <stdio.h>
#include <string.h>

main()
{
    char *p = "now is the time";
    char a[256];

    puts(p);
    strcpy(a, p);
    /* a = p; とできるだろうか */
    /* a[0] = 'N'; とできるだろうか */
    puts(a);
}
```

練習問題２０

ポインタの配列（*name[10]）を宣言し，次のような名前を 10 個セットしなさい。

″taro″, ″jiro″, ″hanako″, ″hirosi″, ″keiji″, ″minoru″, ″takashi″, ″rie″, ″mariko″, ″keiko″

この名前をアルファベット順（辞書での並び順）に整列（並べ換え）させるプログラムをポインタへのポインタを使ってつくりなさい。

《ヒント》文字列の大小比較(辞書での並び順の比較)にはライブラリ関数の strcmp(s1, s2) を利用すればよい。

この関数の戻り値は，次のようになる。

s1 ＞ s2 のとき，　0 より大きい整数　←　s1 の方が辞書では後に出てくる

s1 ＝ s2 のとき，　0

s1 ＜ s2 のとき，　0 より小さい整数　←　s2 の方が辞書では後に出てくる

Ⅱ．関数

１．関数の概念

Ｃにおける関数とは，一つのまとまった機能をもつプログラムの単位であり，関数を用いることによってプログラムの単純化がはかられ，正確でわかりやすいプログラムの作成が容易になる。

各関数の定義は次の形をしている。

```
return 型　関数名（引数の宣言）
{
  宣言と文
  return 式;
}
```

引数の宣言は，引数の型と名前の並びである。また，return 文は呼ばれた関数の値を，呼んだプログラムに返すための機構である。

一つのプログラムでは，関数間のやりとりは，引数と関数からの戻り値と外部変数を通して行われる。関数はソースファイル上では任意の順序に並べてよく，ソースプログラムは複数のファイルに分割できる。但し，関数の分割は許されない。

また，ソースプログラムの先頭で，関数のプロトタイプ宣言を次のように行う。

```
return 型　関数名（引数の宣言）；
```

但し，引数の宣言の部分は，引数の型のみ（名前なしの）の並びでよい。

２．関数の使い方

例題２－１

整数型のデータを入力すると，その値の 2 倍を出力するプログラムをつくりなさい。ただし，計算する部分を関数にしなさい。

▼出力結果２－１

整数を入力して下さい！
5↵
5 の 2 倍は 10 です

考え方

入出力の機能と計算処理の機能を分割する。

▼プログラム２－１

```
01      /* E2-1 */
02      /* 関数の例 */
03      #include <stdio.h>
04      int mult(int);
05
06      main()
07      {
08          int a;
09
10          printf("整数を入力して下さい！¥n");
11          scanf("%d", &a);
12          printf("%d の 2 倍は%d です", a, mult(a));
13      }
14
15      int mult(int x)     /* 関数の定義 */
16      {
17          return x * 2;
18      }
```

３．アドレスを渡す関数

Ｃの関数では，データの受け渡しの方法に，値を渡す（call by value）方法とアドレスを渡す（call by reference）方法がある。

(a)値を渡す方法

呼び出し側の関数から変数 a をユーザ関数の変数 b に渡した後は，変数 a，変数 b はそれぞれ全く独立した変数として扱われる。変数 b が関数の呼び出し側に（引数を介してユーザ関数側から呼び出し側に値を返すことができない）影響することはない。

したがって，関数の独立性を高めることができる。

(b)アドレスを渡す方法

呼び出し側の関数から変数 a のアドレス&a をユーザ関数のポインタ変数*b に渡した場合，間接演算子を用いて*b により呼び出し側の実引数 a の内容を参照できる。すなわち，呼び出し側とユーザ関数側とで引数を介して，双方向のデータ授受を行うことができる。

特に，配列を引数として渡すと，アドレスをたよりに配列を操作することができる。

例題２－２

２個の整数型データｉとｊを初期設定し，ｉとｊの値を交換するプログラムをつくりなさい。ただし，値を交換する部分は関数にしなさい。

▼出力結果２－２

```
i = 10, j = 20
i = 20, j = 10
```

考え方

アドレスを受け取る場合は，ポインタを利用する。また，データの入れ換えは，作業変数を介して行う。

▼プログラム２－２

```
01      /* E2-2 */
02      /* アドレスを渡す関数 */
03      #include <stdio.h>
04      void swap(int *, int *);
05
06      main()
07      {
08          int i = 10, j = 20;
09
10          printf("i = %d, j = %d¥n", i, j);
11          swap(&i, &j);    /* 関数呼出し */
12          printf("i = %d, j = %d¥n", i, j);
13      }
14
15      void swap(int *p1, int *p2)
16      {
17          int work;          /* 作業変数 */
18
```

```
19          work = *p1;
20          *p1 = *p2;
21          *p2 = work;
22      }
```

練習問題２１

２１−１．２つの正の整数 u，v の最大公約数 gcd(u，v)を求めるユークリッドのアルゴリズムについて，繰り返し版のプログラムをつくりなさい。尚，ユークリッドの方法は，「u が v より大きいならば，u と v の最大公約数が v と u-v の最大公約数に等しい」ことに基づく。関数のプロトタイプ宣言は次のようにしなさい。

```
int gcd(int, int);
```

▼出力例

```
最大公約数を求めます！
２つの正整数を入力してください：124 432↵
124 と 432 の最大公約数は 4 です
```

２１−２．（難問）キーボードから 0〜9 の数字 d と，正の整数 n を読み込み，n 以下の整数 x で，x と x の 2 乗の 10 進数表示の両方に d を含むものをすべて出力するプログラムをつくりなさい。ただし，次のプロトタイプ宣言で示すように，整数 x と d を仮引数として，x と x の 2 乗の 10 進数表示の両方に d を含む場合に 1，そうでない場合に 0 を返す関数 hit_digit を作成しなさい。

```
int hit_digit(int x, int d);
```

▼出力例

```
0〜9 の数字 d を入力して下さい：7↵
正の整数 n を入力して下さい：100↵
```

該当するものは次の通り：
27　　74　　76　　87

全部で 4 個ありました

２１－３．（難問）多項式を計算する効率的な方法に Horner の方法がある。それは次のような式の変形に基づいている。

$$P_5(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$
$$= (((a_5x + a_4)x + a_3)x + a_2)x + a_1)x + a_0$$

上式を用いて，double 型の変数 x，double 型の係数配列 a，x の最大次数 n を引数として，Pn(x)の値を返す関数 horner をつくりなさい。
プロトタイプ宣言は次の通りです。

```
double horner(double x, double *a, int n);
```

この関数を利用して，次の出力例のように動作するプログラムをつくりなさい。

▼出力例

```
Horner の方法で多項式を計算します！
次数 n を入力して下さい：5↵
実数係数を順に入力して下さい：
a[5] = 1.0↵
a[4] = 2.0↵
a[3] = 3.0↵
a[2] = 4.0↵
a[1] = 5.0↵
a[0] = 6.0↵
x の値を入力して下さい：1.1↵

多項式の値 = 24.871710
```