

第10章 「ポインタ」のまとめ

10.1 用語のまとめ

関数の引数 (復習) C言語の関数呼出しは _____ である。つまり、ふつうは関数の中で呼出し側の変数を変更することはできない。

教 p.226

アドレス C言語の変数には (register 変数などの例外を除き) メモリ上の _____ (address) がある。

教 p.227

アドレス演算子 単項 `&` 演算子を変数 (や配列の要素など=演算子の左辺に置けるもの) に適用すると、そのアドレスが得られる。単項 `&` 演算子は _____ とも呼ばれる。

教 p.228

アドレスを `printf` で出力するときの書式指定は `%p` である。

ポインタ アドレスを変数に格納することができる。このような変数を _____ という。ポインタ型の変数の宣言は `*` を変数名の前につける。

教 p.229

```
int sato = 178;
int *isako;
isako = &sato;    /* isako は int 型のオブジェクトのアドレスを格納することができる。 */
                  /* “isako は sato を指す” という。 */
```

- `isako` は `int` 型のオブジェクトを指すためのポインタ型 (“`int` へのポインタ型”) である。 (`*` の部分が主で `int` が従である。)

なお、型の一部であることを強調するために `*` を型のほうにくっつけて、

```
int* isako;
```

のような書き方をすることもある。(意味はまったく変わらない。)

- ポインタ変数宣言の `*` はそれぞれの変数名の前につける必要がある。

```
int *isako, hiroko; /* isako は int へのポインタ型、hiroko は int 型になる。 */
```

```
int* isako, hiroko; /* int のほうに*をくっつけても同じ。
                    isako は int へのポインタ型、hiroko は int 型になる。 */
```

```
int *isako, *hiroko; /* isako も hiroko も int へのポインタ型になる。 */
```

間接演算子 ポインタの前に単項_演算子をつけると、それが指すオブジェクトそのものを表す。単項*演算子は _____ と呼ばれる。

```
*hiroko = 180; /* hirorko の指すオブジェクトに 180 を代入 */
/* この例の場合 masaki = 180 と同じ効果を持つ */
```

関数の引数としてのポインタ 関数の引数としてポインタを渡すと、次のようなことが可能になる。

- 呼出し側の変数の値を書き換える。
(C言語の関数の引数は値渡しなので、ポインタを使わなければ、呼出し側の変数の値を書き換えることはできない。)
- 複数の値を返す。

教 p.216

scanf 関数とポインタ scanf 関数で int 型 (%d に対応) や、double 型 (%lf に対応) を受取るとき、変数の前に&をつけたのも、 _____ である。

教 p.238

ポインタの型 ポインタの型は、それが指すオブジェクトの型と正しく対応しなければならない。

教 p.240

この点は、ポインタのインクリメント・デクリメント(後述)のとき、さらに重要になる。

ポインタは配列の要素を指すこともできる。

```
int vc[5] = {10, 20, 30, 40, 50};
int *ptr = &vc[0];
```

&vc[0] で配列の先頭要素のアドレスを表す。(p.177 演算子の一覧表参照)

注意:

- `int *ptr = &vc[0];` のようにポインタ変数の宣言と初期化を一度に行う宣言は、

```
int *ptr;
ptr = &vc[0];
```

と同じ意味になる。

```
int *ptr;
*ptr = &vc[0];
```

ではないので、注意が必要である。

- `ptr + i` は、(配列の要素の型にかかわらず) `ptr` が指す要素の `i` 個後ろの要素を指すポインタになる。
- `* (ptr + i)` は `ptr[i]` と書くこともできる。(2つの書き方はCではまったく等価である。)
- `[]` を伴わず現れた配列名は、 _____ と見なされる。(ただし、`sizeof` のパラメータとなるときなど、いくつかの例外がある。)
 - `scanf` で文字列 (%s に対応) を受取るために `char` の配列を渡すとき、&を付けない(教科書 p.212) のは、このためである。
 - ポインタ同士の比較 (`==`, `<`, `>` など) や減算 (`-`) も可能である。

ポインタと配列

教 p.255

ポインタのインクリメント・デクリメント 配列の要素を指すポインタをインクリメント(++)・デクリメント(--)すると、それぞれ配列の _____ ・ _____ を指すようになる。

教 p.244

配列の受渡し 関数の仮引数の宣言としては、int vc[] と int *vc はまったく同一である。このプログラムの関数 int_set の定義は、

```
void int_set (int *vc, int no, int val) {  
    ...  
}
```

と書いてもまったく意味は同じである。

ポインタ使用上の注意 以下のプログラムはどこが間違っているか？

1.

```
#include <stdio.h>  
  
/*--- n1 と n2 の和・差を sum と diff に格納 ---*/  
void sum_diff(int n1, int n2, int *sum, int *diff) {  
    *sum = n1 + n2;  
    *diff = (n1 > n2) ? n1 - n2 : n2 - n1;  
}  
  
int main(void) {  
    int na, nb;  
    int *wa, *sa; /* ← ここを変えた */  
  
    puts("二つの整数を入力してください。");  
    printf("整数 A :"); scanf("%d", &na);  
    printf("整数 B :"); scanf("%d", &nb);  
  
    sum_diff(na, nb, wa, sa);  
  
    printf("和は%d です。 %n 差は%d です。 %n", *wa, *sa);  
  
    return (0);  
}
```

2.

```
#include <stdio.h>

int *sum_diff(int n1, int n2) {
    int ret[2];

    ret[0] = n1 + n2;
    ret[1] = (n1 > n2) ? n1 - n2 : n2 - n1;

    return &ret[0]; /* 先頭要素へのポインタを返す */
}

int main(void) {
    int na, nb;
    int *wasa;

    puts("二つの整数を入力してください。");
    printf("整数 A :"); scanf("%d", &na);
    printf("整数 B :"); scanf("%d", &nb);

    wasa = sum_diff(na, nb);

    printf("和は%d です。 ¥n 差は%d です。 ¥n", wasa[0], wasa[1]);

    return (0);
}
```

大抵の場合、問題がないように実行されてしまうので、こういう間違いは余計に怖い。