C 言語におけるポインタ

- 変数はメモリ上のどこかに格納されている
  - 変数の値: メモリに格納されている数値
  - アドレス: 変数の値が格納されているメモリ上の番地
- ポインタ変数
  - 値としてアドレスを格納する変数のこと
  - ポインタ変数の値 (アドレス) とポインタ変数のアドレスは異なるものであることに注意
- ポインタ変数の宣言、代入、実体へのアクセス
  - 整数型ポインタ変数の宣言: `int *p;`
  - 整数型変数の宣言: `int q;`
  - 変数 `q` のアドレスをポインタ変数 `p` に代入: `p = &q;`
  - ポインタ変数 `p` に格納されているアドレスに格納されている値の参照 (間接参照): `*p`

ポインタの例 (1)

- 例 2.5.1 (ハンドブック 2.5 節)

```
#include <stdio.h>
int main() {
  int q = 200;
  int* p = &q;
  printf("q␣is␣%d␣and␣*p␣is␣%d.\n", q, *p);
  return 0;
}
```

  - q のアドレスを p に代入
  - q と*p の値を出力 → 両者とも 200

ポインタの例 (2)

- 例 2.5.2 (ハンドブック 2.5 節)

```
#include <stdio.h>
int main() {
  int q;
  int* p = &q;
  *p = 300;
  printf("q␣is␣%d␣and␣*p␣is␣%d.\n", q, *p);
  return 0;
}
```

  - q のアドレスを p に代入
  - *p に 300 を代入 (ここで q=300; と書いても等価)
  - q と*p の値を出力 → 両者とも 300

関数呼び出し (ポインタ渡し)

- 例 2.7.4 (ハンドブック 2.7 節)

```
#include <stdio.h>
void division(int divident, int divisor, int* quotient,
              int* residual) {
  *quotient = divident / divisor;
  *residual = divident % divisor;
}
int main() {
  int josuu = 3;
  int hi_josuu = 13;
  int shou, amari;
  division(hi_josuu, josuu, &shou, &amari);
  printf("%d␣/␣%d␣=␣%d␣...␣%d\n", hi_josuu, josuu,
         shou, amari);
}
```

間違った例 (値渡し)

- 例 2.7.5 (ハンドブック 2.7 節)

```
#include <stdio.h>
void division(int divident, int divisor, int quotient,
              int residual) {
  quotient = divident / divisor;
  residual = divident % divisor;
}
int main() {
  int josuu = 3;
  int hi_josuu = 13;
  int shou, amari;
  division(hi_josuu, josuu, shou, amari);
  printf("%d␣/␣%d␣=␣%d␣...␣%d\n", hi_josuu, josuu,
         shou, amari);
}
```

  - 誤った答えが出力される。なぜ？

一次元配列

- (静的) 一次元配列 (ハンドブック 2.3.1 節)

```
double v[10];
v[0] = 1.0;
v[1] = 2.0;
...
```

要素数はコンパイル時にすでに決まっている定数でなければならない

- (動的) 一次元配列 (ハンドブック 2.12 節)

```
double *v; /* ポインタ */
v = (double*)malloc((size_t)(10 * sizeof(double));
...
free(v); /* 確保した領域を開放 */
```

実行時に要素数を指定可能

ポインタと一次元配列

- 一次元配列を表す変数は、(実は) 最初の要素を指すポインタ (ハンドブック 2.5.3 節)
  - `v` と `&v[0]` は等価
  - `(v+2)` と `&v[2]` は等価
  - `*v` と `v[0]` は等価
  - `v[i]` は `*(v+i)` の簡略化した書き方 (糖衣構文)
- C 言語では配列の添字は 0 から始まることに注意
- `double v[10];` と宣言した場合、`v[0]` 〜 `v[9]` の 10 個の要素を持つ配列が作られる。`v[10]` は存在しない。値を代入したり参照しようとするとエラーとなる (`malloc` で動的に作成した場合も同様)
  - 一次元配列の例: array.c
- 関数に配列を渡すときには、そのサイズ (長さ) と先頭要素のアドレスを渡す
  - 関数への配列の渡し方: array2func.c