≫ 文法の曖昧さを理解して確実性と再利用性を高める

マイコンC言語 転ばぬ先のつえ

第11回　演算子②…[ ] と * の意味，++a と a++ の違い

鹿取 祐二

本連載では，C言語の言語仕様（文法）の曖昧な部分と，それにより起こる問題を解説します．再利用性と効率が高く，安全かつ安心して使えるソフトウェアが開発できるようになることを目指します．

今回は，添字演算子 [ ] と関節演算子 * の意味と，後置インクリメント（デクリメント）の文法について解説します．〈編集部〉

17 添字演算子 [ ] と間接演算子 * …実は全く同じ意味！

● 文法でも使い分けは決まっていない

配列の要素を参照する [ ] の添字演算子と，ポインタの指す内容を参照する * の間接演算子は，同じ意味です．[ ] だから配列にしか使えないとか，* だからポインタにしか使えないということはありません．

▶一般的な記述

具体的な記述例を見てみましょう．初期値付きの配列と，それを指すポインタが次のように宣言されていたとします．

```
int abc[] = { 10, 20, 30, 40 },
*xyz = abc;
```

この場合，添字演算子の [ ] と間接演算子の * は，次のように使うのが一般的です．

```
abc[1]      // Good!abc[1]の20が評価結果
*(xyz + 1)  // Good!abc[1]の20が評価結果
```

配列名には [ ]，ポインタには * が使われていて，非常に分かりやすい記述です．

▶逆に記述してもOKだが…避けるのがベター

しかし文法では，これらの演算子を次のように使っても良いことになっています．

```
*(abc + 2) // Bad!abc[2]の30が評価結果
xyz[2]     // Bad!abc[2]の30が評価結果
```

これは分かりづらい記述です．配列名に *，ポインタに [ ] が使われていて，各変数の宣言とは逆の記述になっています．もし，冒頭の宣言の記述がなければ，誰しもabcはポインタ変数，xyzは配列だと思うはずです．文法上は許されても，可能であれば避けた方が良い記述です．これは，MISRA-Cにも同様のルールがあります．

● 添字演算子で絶対に避けたい記述

▶式$_1$[式$_2$] の記述で式$_1$を整数型，式$_2$をポインタ型にする

[ ] の添字演算子については，絶対に避けた方が良い記述があります．それは，次のような記述です．

```
3[abc]// Very Bad!abc[3]の40が評価結果
3[xyz]// Very Bad!abc[3]の40が評価結果
```

K&R（カーニハン＆リッチー）だと，この記述は文法違反でした．理由は，[ ] の添字演算子を使った式$_1$[式$_2$] の記述だと，式$_1$はある型へのポインタ型，式$_2$は整数型と決まっていたからです．それがC89からは式$_1$[式$_2$] において，2つの式のうち1つはある型へのポインタ型，もう1つは整数型と改訂されました．

▶なぜC89から改訂されたのか

これは筆者の勝手な見解ですが，K&Rのときから次の2つは同じ意味の記述であると説明がありました．

式$_1$[式$_2$] ←→ *(式$_1$+式$_2$)

しかし，K&Rでは配列演算子だと式$_1$はポインタ型で式$_2$は整数型，一方の間接演算子だと式$_1$と式$_2$はどちらか一方はポインタ型，もう一方は整数型だったのです．もし同じ意味の記述であるならば，配列演算子にだけ制約があるのは不整合である，との理由からC89では変更されたのではないかと思っています．

▶極限の分かりづらさなので使うべからず

結果，3[abc] のように記述できるようになったのですが，これは極限の分かりづらさです．筆者は間違っても使うべき文法ではないと考えています．

18 後置インクリメント（デクリメント）は優先順位と演算タイミングを別々に考えるべし

■ 文法

● 演算子の優先順位を考えると納得できないかも？

後置インクリメント，デクリメントは，前置の場合と合わせて，必ず入門書に説明があります．次のような記述例だと思います．

第2回　整数型②…表現できない値を代入したときの規則（2021年2月号）
第3回　整数型③…整数の格上げと算術変換規則（2021年3月号）
第4回　浮動小数点数型①…内部表現と誤差（2021年5月号）