

ソフトウェア概論 B

数学科 吉開範章, 渡辺俊一 (栗野 俊一)

2009/11/20 ソフトウェア概論

お知らせ

□資料

<http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino>

前回のまとめ 1: メモリとポインター値

□ メモリモデル

- 計算機にはメモリがある(メモリはセルの集まり)
- セルの性質
 - ▶ アドレス(番地)をもつ: メモリ内でのそのセルの位置を表現
 - ▶ セルは一つでは **1 byte** しか記憶できない
 - ▶ ワード: 複数のセルをまとめることにより、大きな数が記憶ができる

□ 変数とメモリ

- 変数は、ワードによって実現されている
 - ▶ 変数はワードの先頭アドレスで区別される: ポインター

□ ポインター値の属性 (元々変数が持つ属性)

- アドレス情報
 - ▶ ワードの先頭セルのアドレス値 (`%p` で表示することができる)
- 型情報
 - ▶ `byte` 数 (`sizeof` で調べることができる)
 - ▶ セルの内容をどう解釈するか..

前回のまとめ 2: ポインター値の危険性

□ 変数とポインター値

- アドレス演算子 (&): 変数からポインター値を取り出す
- 間接演算子 (*): ポインター値から変数を作る
 - ▶ 「&」と「*」は逆演算

□ 型キャスト 「(type *)」

- アドレス値(単なる数)から、ポインター値を作る
 - ▶ ポインターは「変数に対応しないメモリ」を指す可能性がある
 - ▶ 「変数に対応しないメモリ」を変更すると大変なことに!!

□ ポインター値の危険性

- 「変数に対応しないメモリ」を操作する可能性
 - ▶ 「C 言語」を本質的に「強力」にしている: 現場で利用される理由
 - ▶ 「C 言語」を本質的に「危険」にしている: 初心者に向かない理由
- 変な事が起きたら「ポインターを疑え」

□ 安全なポインターの使い方

- 適切な変数を指すポインター値だけを利用すればよい

本日の概要：ポインタ(2)

- C 言語のポインタを理解する
 - ポインタ演算
 - ポインタ変数
 - 配列とポインタの関係
- これまでの内容とポインタ
 - 暗黙の内に利用されていたポインタ値
 - ▶定数ポインタとしての配列名
 - 訳けも別らずに利用されていたポインタ値
 - ▶scanf におけるポインタ値の利用
- ポインタの便利な使い方
- 課題

ポインタ演算 (復習とまとめ)

□ ポインタ値の属性(採録)

- アドレス値：計算によって変更できる
- 型情報：型キャストによって変更できる

□ ポインタ値の作り方

- 変数からアドレス演算 (&) で作る
 - ▶ ほぼ、安全だが、それ単独では、あまり意味がない値
- アドレス値とキャストで作る
 - ▶ 危険を内包するが、潜在能力の高い値
- 他のポインタから計算で作る
 - ▶ 「便利さ」を追加できる(もともと安全ならこれも安全..?)

□ ポインタ値の計算

- ポインタ値に整数を加える：ポインタ値
 - ▶ ポインタのアドレス値だけを変更
 - ▶ アドレス値は、「加える整数 × sizeof(型)」だけ変化する
- ポインタ値同士の引き算：整数値
 - ▶ 「アドレス値の差 ÷ sizeof(型)」になる

ポインター型変数

□ ポインター型変数とは？

- 「ポインター値」を値としてもつ「変数」
- ポインター型変数の宣言の仕方
 - ▶ ポインター型変数名(pv)の前に「*」を付ける
 - ▶ `Type *pv;` // 変数「pv」の「型」は「Type へのポインター型」
 - ▶ 「ポインター型変数名(pv)の前に(*)を付ける」と「Type 型」になる

□ 「ポインター型変数」を「ポインター」と呼ぶ理由

- ポインター変数に(*)を付けると「他の変数」になる (別名)
 - ▶ 他の変数を指す(ポインティングする)ので、「ポインター」
- 「他の変数」を「間接的」に表現する

□ 間接 vs 直接

- 変数名：直接、「変数」を指定する
 - ▶ 対応が固定されているので、簡単だが、柔軟性がない
- ポインター：間接的に「変数」を指定する
 - ▶ 「ポインター値の代入」によって対応関係を作成する(代入が不可欠)
 - ▶ 対応を後から変更できるので、柔軟性がある(ポインターの利用価値)

ポインタの利用法

□ 別名 (`alisa` ; エイリアス)

- ポインタ型変数(`xp`)に変数(`xv`)のアドレスを代入する

- ▷ `*xp` を `xv` の代わりに利用できる

- ▷ `*xp` が `xv` の「別名」になる

□ 利用例

- 変数宣言

- `int iv; // iv を ip とは「別に用意」する必要がある`

- `int *ip; // ip 自身の宣言 (指し先の型へのポインタ型)`

- 別名としての利用

- `ip = &iv; // ip に iv のアドレスを代入する(ip が iv を指す)`

- `// これがないと *ip は使えない(典型的なバグの原因)`

- `*ip = 1; // iv = 1; と同じ (別名になっている)`

- `printf ("%d\n", *ip);`

- `// printf ("%d\n", iv); と同じ`

ポインター値の利点(1): 値であること

□ 関数の引数

- 「値」しか渡す事ができない
- 呼び出し側の変数の値を変更するには .. ?
 - ▶ 「関数値」を返し、その値を呼び出し側で変数に設定
 - ▶ 変数の値の変更(副作用)は呼び出し側で行う

□ 関数引数としてのポインター値

- 関数の実引数として、ポインター値を指定できる
 - ▶ 実際に渡されるのは、「アドレス値」だけ...
 - ▶ 値の受け手としての仮引数変数をポインター型変数として宣言
 - ▶ 与えられたポインター値を利用して、呼び出し側の変数を参照できる
- 副作用(変数の内容の変更)が(見えない)関数の中で行われる
 - ▶ プログラムを理解しにくくする原因
 - ▶ あまり利用すべきでない

□ 過去の例

- `scanf` の引数
 - ▶ `scanf` ってよくわからない

scanf 再考

□ scanf の機能

- キーボードから入力された値を変数に代入する
 - ▶ scanf 関数の中で代入を行う
 - ▶ ポインターの指定が必須
 - ▶ 「おまじない」の「&」
- 色々な型の変数を扱う
 - ▶ 「型」情報はどうする？
 - ▶ 最初の引数の文字列の中で指定
 - ▶ 「暗記必須」の「%」

□ scanf の利用

- 知らず知らずの内にポインターを利用していた...
 - ▶ 問題がおきたら、scanf を疑え

配列名とポインタ

- 配列とポインタは密接な関係がある
 - 配列名の出現できるところには、ポインタ式が書ける
 - ▶ というか、配列名の代わりにポインタを書くことが多い
 - C 言語でポインタが多用される訳
 - ▶ 柔軟性のある配列名として利用するため
 - ▶ C 言語のほとんどの状況では、ポインタが不要
- 配列名はポインタ一定数
 - 配列名は、ポインタ値を持つ
 - ▶ 具体的には配列の先頭の要素へのポインタ
- 配列の要素への参照は、実は、間接参照
 - $$\text{array}[\text{index}] == *(\text{array} + \text{index})$$
 - 配列を利用すること、すなわちポインタの利用だった..

再び scanf 再考

□ scanf での文字列入力

```
char line[10];
```

```
scanf ( "%s", line ); /* 何故か '&' がない.. */  
/* line はポインターなので.. */
```

□ 途中から入力するには..

```
char line[10] = "abcdef";
```

```
scanf ( "%s", line + 3 ); /* "xy" と入力すると.. */  
printf ( "%s\n", line ); /* "abcxy" と後が上書きされる */
```

□ 準備した領域より長い文字列をいれたらどうなるか？

- 危険なポインターの利用になってしまう...

- 「正しい文字列」の入力方法

- ▷ scanf で長さ指定

- ```
scanf ("%9s", line);
```

- ▷ fgets を利用する。

- ```
fgets ( line, 10, stdin );
```

課題

□課題は、次の Web Page の内容を参照してください。

<http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino>

おわり

終了