# ソフトウェア概論 B

数学科 吉開範章, 渡辺俊一 (栗野 俊一)

2010/12/10 ソフトウェア概論 B

# 前回のまとめ 1 : ポインター型配列

- 二次元配列 ( int darray[M][N] )
  - C 言語上は「配列の配列」として実現される
    - darray[i] は、 int[N] 型の配列へのポインタ値
    - 一般に n 次元配列は、「n-1 次元配列」の配列
  - メモリーモデル上は一次元配列 ( int sarray[M*N] ) と同じ
    - 左の添字を増やす : アドレス値を sizeof (int[N]) だけ増やす
    - 右の添字を増やす : アドレス値を sizeof (int) だけ増やす
  - darray[i]のサイズが固定(i に無関係に一定)になっている点がミソ

    アドレス値の計算を、 C 言語に任せれば利便性が得られる
- ポインター型配列 ( int *parray[M] )
  - ポインター型変数の配列 ( parray[i] はポインター型変数 )
    - (∀i) parray[i] = darray[i] とすれば、parray は darray の別名になる
  - parray[i] の値を規則的にしなければ、色々な形が表現可能
    - cf. pascal の三角形など

    アドレス値の計算を、自分で行えば柔軟性が得られる

# 前回のまとめ 2 : 動的なメモリーの確保

- auto 変数のメモリー管理
  - メモリーは有限 : 節約するために使いまわしされる
    - 同時に利用されない、異なる auto 変数を同じメモリーに割り当る
  - メモリー管理 : 自動的(auto)に行なわれる(スタックの利用)
    - 変数(配列)のサイズを実行前に定める必要がある(静的なメモリー割り当て)

メモリ管理を C 言語に任せれば利便性が得られる

- malloc/free 関数の利用
  - メモリー管理を自分で行うための仕組
    - 利用したいメモリーを自分で借りて自分で返す (動的なメモリー割り当て)
    - 確保したメモリーを解放し忘れると大変な事に.. (メモリーリーク)
  - malloc : メモリーの確保 / 好きなサイズのメモリーが得られる
    - 引数は、必要なメモリーのサイズで、値はそのメモリーへのポインター値
  - free : メモリーの解放 / malloc で確保したメモリを解放する
    - 引数は、malloc の返値で得られた、自分で確保したメモリーへのポインター値

メモリ管理を、自分で行えば柔軟性が得られる

# 本日の概要：文字列とポインター

- C 言語の文字列：メモリーモデルからの観点
  - 文字配列 (char string[] + EOS) で実現
  - 固定サイズの「配列」 vs 可変サイズの「文字列」
    - 「番兵」の利用
  - 「文字列」リテラルの正体
- 「文字列」の操作
  - 「文字列」を扱うライブラリの確認 ( Text p.260 - 265 )
- 課題
  - 「後文字列」をポインターで表現する
    - <配列名,開始位置> == (配列名+開始位置)

# C 言語の文字列 (再訪)

- 「文字列」とは？
  - 「文字(char 型の値)」の並んだもの
    - 「長さ」という属性を持つ ( 0　)
    - n 番目の「文字」という情報(JIS/ASCII コード)を持つ
- C 言語での「文字列」
  - C 言語では、「文字列を直接表現する手段」がない
    「文字列」を「char 型配列」で表現する「習慣」がある
  - C 言語の「文字列」サポート
    - 「文字列」リテラル
    - 「文字列」処理ライブラリ

# 番兵

- 固定サイズの配列 vs 可変サイズの文字列
  - 可変サイズの文字列を、固定サイズの配列で表現するには？
    - 「長さ」を持つ : 安全だが、複数の情報を扱うので効率が落る
    - 「終り」を持つ : 配列単独で表現できるので効率的だが、危険性が..
  - C 言語の「文字列」表現
    - '\0' を「文字列の終り(EOS : End Of String)」とする
    - 「文字列操作」は、この「形式」を前提に設計されている
- 「番兵」モデル
  - 番兵とは？:「終り」を表現する特別な「値」
    - C 言語の「文字列」では、EOS('\0')が「番兵」
    - EOS は、「文字型の値」だが、「文字を表さない値」となる
  - 番兵方式による「文字列表現」の得失
    - 文字の処理と同時に文字の終りの処理ができる (効率が良い)
    - 文字配列との区別ができない ( 「習慣」が守られないと... )

# 文字列リテラル

- 文字列リテラルとは？
  - C 言語上の表現で、「文字列」の定数を表す [01]
  - 「" (ダブルクォーテーション)」で文字の並びを囲む
    - "abc" は、’a’, ’b’, ’c’ の三つの「文字」からなる長さ三の文字列
    - "" は、一つも「文字」を含まない長さ零の文字列
- その実体は？( メモリーモデルから見れば.. )
  - 「文字列リテラル」そのものは char へのポインター型定数
    - 配列名と同じ性質を持つ
    - ポインター変数に代入できる
  - ポインターの先は？
    - 「文字列」を構成する「文字」の並び + EOS からなる文字型配列
- 文字列リテラルは「定数」か？
  - 「"abc"[1] = ’B’」は何をしているか？
    - 「C 言語」的には、「未定義 : 何が起きても(あるいは起きなくても)よい」

# 文字配列 vs 文字列リテラル

- □ 文字配列
  - ○ 文字変数の配列
    - ▷ 配列の要素は、文字変数
    - ▷ 文字変数なので、文字(を表現する JIS Code: 0　255 の整数値)を入れる事ができる
    - ▷ 「変数」なので「初期化」しないと、「参照は無意味」
  - ○ 「配列名」をもつ
    - ▷ 「配列名」は 「(char *) 型の定数 (ポインター定数)」
- □ 文字列リテラル
  - ○ 文字定数(?)の配列
    - ▷ 実際は文字配列の形で実現されている
    - ▷ 要素値を変更する事ができるかどうかは、System 依存 ( 止めた方がよい )

# 「文字列」を扱う関数

- 文字配列を利用して「文字列」が表現できる [02]
  - EOS ( '\0' ) の有無が問題
- 「配列の大きさ」と「文字列の長さ」は無関係
- C 言語には「文字列」を扱う機能が用意されていない
  - 「文字列」を扱う関数群(標準ライブラリ)でサポート
- 「文字列」の属性
  - 長さ : strlen
  - 構成する文字への参照 : 配列名(ポインター値)を利用する
- 「文字列操作」
  - 文字列のコピー : strcpy
    - 領域が重なる場合は、「未定義」[03]
    - memmove を使う
  - 文字列の追加/比較/検索 : strcat/strcmp/strstr/strchr [04-07]

# 「後文字列」とポインター

## □ 「後文字列」

### ○ 文字列の後の部分を表現する考え (栗野固有概念)

- ▹ 「文字列」と「その部分文字列の開始位置」の対で、「文字列の後半の部分」を表現する
- ▹ 例 1 : 文字列が「abcdef」の時、<「abcdef」,3> は「def」を表す
- ▹ 例 2 : <文字列,0> は文字列と同じものを指す

### ○ 文字列の利用例

- ▹ 2010/11/05 の program/sample-009, 011, 013

## □ 「後文字列」とポインターの関係

### ○ ポインターは、配列の一部の場所を直接指している

- ▹ <文字列,場所> は、ほとんどの場合、ポインター値「文字列 + 場所」で差し替え可能
- ▹ 置き換えられない場合 : 文字列の先頭が知りたい場合 / 何文字目かを知りたい場合

### ○ 課題

- ▹ 「後文字列」をポインターに置き換える

ポインター値「文字列 + 場所」は常に、後文字列「<文字列,場所>」で表現可能

逆が成立する場合もある(文字列自身や、場所が不要な場合)

# 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