ソフトウェア概論 A/B

-- データ構造 (8) --

(メモリモデルとポインター)

数学科 栗野 俊一 / 渡辺 俊一

伝言

私語は慎むように !!

- 色々なお知らせについて
  - 栗野の Web Page に注意する事

    http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino
- 講義開始前に済ませておく事
  - PC の電源を入れる
  - ネットワークに接続しておく事
  - 今日の資料に目を通しておく事
- 講義前の注意
  - 講義前は、栗野は準備で忙しいので TA を捕まえてください
- やる気のある方へ
  - 今日の資料は、すでに上っています
    - どんどん、先に進んでかまいません
- 本日の CST Portal の出席パスワード : 20151211
  - 出席は成績に影響しませんが、折角の機能なので、使いましょう

今後の予定

- 今後の予定(後ろから)
  - 2015/01/22 (講議最終日)
    - 試験を行う
  - 2015/01/15 (講議最終日前)
    - 模擬試験を行う
  - 2015/01/08 / 2015/01/01
    - 冬期休暇期間中：この講議はない
  - 2014/12/25
    - 後期予備日
  - 2014/12/18
    - データ構造 (9) / ポインター(2)・落穂拾い
  - 2014/12/11 (本日)
    - データ構造 (8) / ポインター(1) と動的構造

前回(2015/12/04)の内容

- 前回の内容
  - 関数引数の実現
    - メモリ上に保存されている
    - 引数のために使用されたメモリは、再利用される
    - メモリ上の「内容」を、どのような「情報」と取るかは、引数の型で決る
  - 浮動小数点数の内部表現
    - 浮動小数点数が入ったメモリをバイト配列とみなす
    - 関数の引数の型を「敢えて不一致にする」事で、実現

お知らせ

- 本日の予定
  - ビット処理 (前回の積み残し)
  - データ構造 (8)
    - メモリモデルとポインター
- 本日の目標
  - 演習
    - 課題の提出

前回 (2015/12/04) の課題

- 前回 (2015/12/04) の課題
  - なし(前回の課題は、今回に回す)

本日の課題 (2015/12/11)

- 本日 (2015/12/11) の課題
  - 課題 20151204-01: (前回の積み残し)
    - ファイル名 : 20151204-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : ポインターを利用して、整数変数の値を正値にする
  - 課題 20151211-01:
    - ファイル名 : 20151211-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : ポインター演算子を利用して構造体を操作
    - ファイル形式 : テキストファイル(C 言語プログラムファイル)
- ※
  - ファイル形式は、いずれもテキストファイル(C 言語プログラムファイル)

ポインタ値とポインタ型

- ポインタ値
  - 「&」の作る値は実は、単なるアドレス値 *だけ* ではない
    - アドレス値も持つが、それと、「型情報」も持つ
    - 型情報：サイズ + 処理の仕方
    - !! 型情報は、コンパイル時だけ、実行時には解らない(解るのはアドレスだけ)
- ポインター値の型
  - 「型名 *」：「～型へのポインタ型」と読む
    - 「* をつけると「型」と同じになる」の意味
    - char * :「文字列」ではなく、「char 型へのポインタ型」だった

ポインター値の計算

- ポインター値(コンパイル時の情報)
  - 二つの情報をもつ
    - 型情報：何型の情報が入っているものか？
    - アドレス値：どこに入っているか？
- ポインター値の計算
  - 整数値 n を加える事ができる
    - 型情報は変らず、アドレス値だけが変化
    - アドレス値は n × sizeof (型) だけ変化 (n は負の数でもよい)
  - 同じ型のポインター同士なら引き算もできる
    - 結果は整数値で、(アドレス値の差) / sizeof (型) となる
    - p, q が同じポインター型なら「p + (q - p) == q」が恒等的に成立
  - 「キャスト」を利用して、型を変更できる
- ポインター値と添字
  - 恒等的に「 p[n] == *(p + n) 」が成立する

配列とポインタ型

- 配列とポインタ値
  - 「～型一次元の配列名」は、「～型へのポインタ型定数」となる
- 一次元の配列宣言とポインタ型変数宣言
  - 一次元の配列宣言「char a[N];」
    - char 型の変数 a[0] ～ a[N-1] の N 個の変数を宣言
    - 個々の要素変数の型は char 型
    - 配列名「a」は 要素の先頭を指すポインター型定数(a == &a[0])
  - ポインタ型変数宣言「 char *p; 」
    - 「char *」型の変数 p の 1 個の変数を宣言
    - 変数 p の値は不定 (だから *p の値も宣言時点では不定)
    - !! p の値は適切に初期化して利用する必要がある
  - 配列名によるポインター変数の初期化
    - 代入文「p = a」を行うと...
    - 「p[k]」と「a[k]」は全く、同じように振る舞う

関数呼出しとメモリモデル

- 引数付きの関数呼出しの解釈
  - 「int func (x) { return x + 1; }」の時に、「func (5)」とは？
    - これまでは、「5 + 1」に置き換えて考えてきた (数学的解釈)
  - メモリモデルでの解釈
    - 「func (5)」: メモリのどこか (x という名前をつける) に 5 を保存する
    - 「return x + 1;」では、メモリ x から、5 を取り出して計算する
- C 言語ではどちらの解釈が適切か？
  - 実は.. メモリモデルになっている
    - では、数学解釈ではダメなのか : 実をいえば、副作用がなければ大丈夫
  - この違いが問題になるのは？
    - 変数への「代入」操作が行われる場合
    - I/O が行われる場合

ポインター演算子 (->)

- 構造体へのポインターを使った要素の参照
  - 間接演算子(*)と、メンバー参照演算子(.)の組み合わせになる
    - 優先順位が問題になる (「.」の方が優先)
    - 例 : Point2D *ptr の場合 (*ptr).x と記述する
- ポインター演算子 (->)
  - 構造体へのポインターを使った要素の参照を一挙に行う
    - 例 : Point2D *ptr の場合 ptr -> x と記述する