ソフトウェア概論 A/B

-- メモリモデル(2)/型/ポインター/printf --

数学科 栗野 俊一

伝言

私語は慎むように !!

- 教室に入ったら
  - 直に Note-PC の電源を入れておく
    - Network に接続し、当日の資料に目を通す
    - skype に Login する
    - Windows Update をしておこう
- 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
- 作業
  - Web 履修科目登録の確認
    - CST Portal も確認しておきましょう

前回の復習

- □ 講義
  - ○ メモリモデル
    - ▷ メモリ：アドレスのついたセル(1byte を記憶)の集まり
    - ▷ メモリ(セル)操作：アドレスを指定して、情報の読み書きができる
  - ○ メモリモデルと C 言語の変数との関係
    - ▷ char 型変数は、セルと同じと見做せる(変数をセルとして解釈可能)
    - ▷ メモリアドレスと変数名が対応
  - ○ 文字型配列：文字変数の並び
    - ▷「セルの並び」(連続したアドレスを持つセルの集まり)と解釈可能
    - ▷ 配列名は、先頭の要素の「アドレス」を持つ
  - ○ アドレス操作
    - ▷ アドレス演算子「&」:「&変数名」で「アドレス」を得る事ができる
    - ▷ 間接演算子「*」:「*アドレス」は、「変数名」と同じ振舞をする
    - ▷ 添字演算子「[]」:「アドレス[n]」は、「*(アドレス+n)」と同じ

本日の予定

- 講義
  - メモリモデル(2)
    - 他の型の変数とメモリモデルの関係
    - ポインター型 (sizeof 演算子)
  - 複合型
    - 多次元配列と構造体
  - 引数とスタック
    - 加変長引数
    - printf/scanf
- 演習
  - 課題の提出

本日の課題 (2012/10/26)

- 今週 (2012/10/26) の課題
  - 課題 1:
    - ファイル名 : 20121026-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : 単純なオセロボード
    - 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)
  - 課題 2:
    - ファイル名 : 20121026-2-QQQQ.c (QQQQ は学生番号)
    - 内容 : printf/scanf の利用
    - 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)
- 先週(2012/10/12)の課題
  - 課題 1:
    - ファイル名 : 20121026-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : メモリ操作での和
    - 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)
  - 課題 2:
    - ファイル名 : 20121026-2-QQQQ.c (QQQQ は学生番号)
    - 内容 : アドレスを利用した間接参照
    - 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)

多次元の配列

- 二次元の配列
  - 一次元の配列を二次元的に利用する事ができる
    - 二つの添字からアドレスを計算すればよい
  - 始めから二次元の配列を宣言する事ができる
    - 配列の要素のアドレス計算は、自動的に行われる
    - 実態は「(一次元)配列の(一次元)配列」だが、慣例により「二次元配列」と呼ぶ
  - 応用例
    - 行列は、二次元配列で表現可能
- 多次元の配列
  - 一つの型から新しい配列型を作るだけなのでいくらでも大丈夫(?)

整数型とメモリモデル

- 整数型とメモリモデル
  - 文字型変数はメモリセル一つに対応
    - では、整数型変数は ... ? / 実は、連続したメモリセルに保存される
- sizeof 演算子
  - その型の変数が、何個(byte)のセルをしめるかを教えてくれる
    - sizeof(char) == 1
    - sizeof(int) == 4 (32 bit の場合)
  - int 型の変数は 4 つのセルで表現される

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
  - 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 : 実をいえば今迄の内容なら問題なし
  - 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

データの混在した出力

- □ データの混在した出力
  - ○ 出力の場合に文字列の中に数値を含める事が多い
    - ▹ 一行の出力に複数の関数の呼び出し : 煩雑なかんじがする
  - ○ パターンが固定ならば、ある程度は簡便化できる
    - ▹ 単に関数にしても、お余り得した気分になっていない
- □ 文字列の中に数値を埋め込む事を考える
  - ○ 文字列の中に数値を表現する特殊な文字列を含め、数値に置き換える
    - ▹ '%' を特別な意味に利用
  - ○ 更に、複数の整数値が扱えるようにする
    - ▹ 引数が可変長になる (関数宣言の引数の "..." に注意)
  - ○ 整数値だが、8 進や 16 進で出したい場合も考える
    - ▹ '%' の後ろの一文字で判断 (d : 10 進 / o : 8 進 / x : 16 進)
    - ▹ '%' 自身を出力するために "%%" で、'%' を出力する
  - ○ 更に、色々な型の値の出力を考える
    - ▹ c : 文字 / s : 文字列 / f : 浮動小数点数
- □ printf : ライブラリ関数
  - ○ 書式付きの出力関数

printf の書式

- printf 関数
  - 色々な数値を出力形式(format)を指定して出力する関数
    - printf (char *format, ...); // 引数の個数は可変長
    - ex. printf ("答は %d です\n", 1 + 2 + 3); →「答は 6 です[改行]」
- 書式指定の一般形式
  - <書式指定> ::= (<文字> | "%%" | <書式>)*
    - 書式指定は、文字(それ自身)か、"%%" ('%' 一文字) か、書式(数値表示)
  - <書式> ::= '%' [<精度>] <型指定>
    - 書式は '%' で始まり、省略可能な精度記述の後に型指定がある
  - <精度> ::= ['-'] <整数> ['.' <整数>]
    - 精度は '-' (省略可能) を先行した整数で、'.' 以下が追加できる
  - <型指定> := 'd' | 'c' | 's' | 'f' | ..
    - 型指定は、一文字で、そのデータの型を表現する

scanf : 書式付の入力

- scanf 関数
  - 色々な数値を出力形式(format)を指定して入力する
    - scanf (char *format, ...); // 引数の個数は可変長
    - ex. scanf ("%d", &n); // n は整数型 / 引数はポインターを指定
- scanf の書式
  - 基本は、printf と同じ