ソフトウェア概論 A/B

-- ポインター(2) --

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
- 廊下側の一列は遅刻者専用です(早く来た人は座らない)
- 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

今後の予定

- 今後の予定(後ろから)
  - 2014/01/24 (講議最終日)
    - 試験を行う
  - 2014/01/17 (講議最終日)
    - 1 限 : 落穂拾い
    - 2 限 : 模擬試験を行う
  - 2014/01/10 (本日)
    - 講議最終日 (メモリモデルとポインター (2))

前回 (2013/12/20) の復習

- 講議
  - メモリモデル：メモリを「その操作と挙動」から理解する(何であるかは問わない)
    - メモリとは：セルの集まり (セルは独立で、アドレスで区別)
    - セルの属性：アドレス(番地:場所を表現)と内容(記憶している情報)をもつ
    - セルへの操作：Write (情報の書き込み) / Read (情報の読み出し)
    - セルの挙動：何度でも Read できる / Write すると前の情報はなくなる
  - メモリモデルと C 言語の変数
    - C 言語の「変数」は「メモリ」上の複数のセルからなる
    - 「char 型変数」は「一つのセル」で実現されている
  - メモリモデルと C 言語の配列
    - 「配列」は、「セルの並び」
    - 「配列の要素の指定」は、「先頭のセルからのオフセット」で行う(添字)

本日の予定

- 講義
  - ポインター型 (sizeof 演算子)
    - キャスト
  - 引数とスタック
    - 加変長引数 : printf/scanf
- 演習
  - 課題の提出

前回 (2013/12/20) の課題

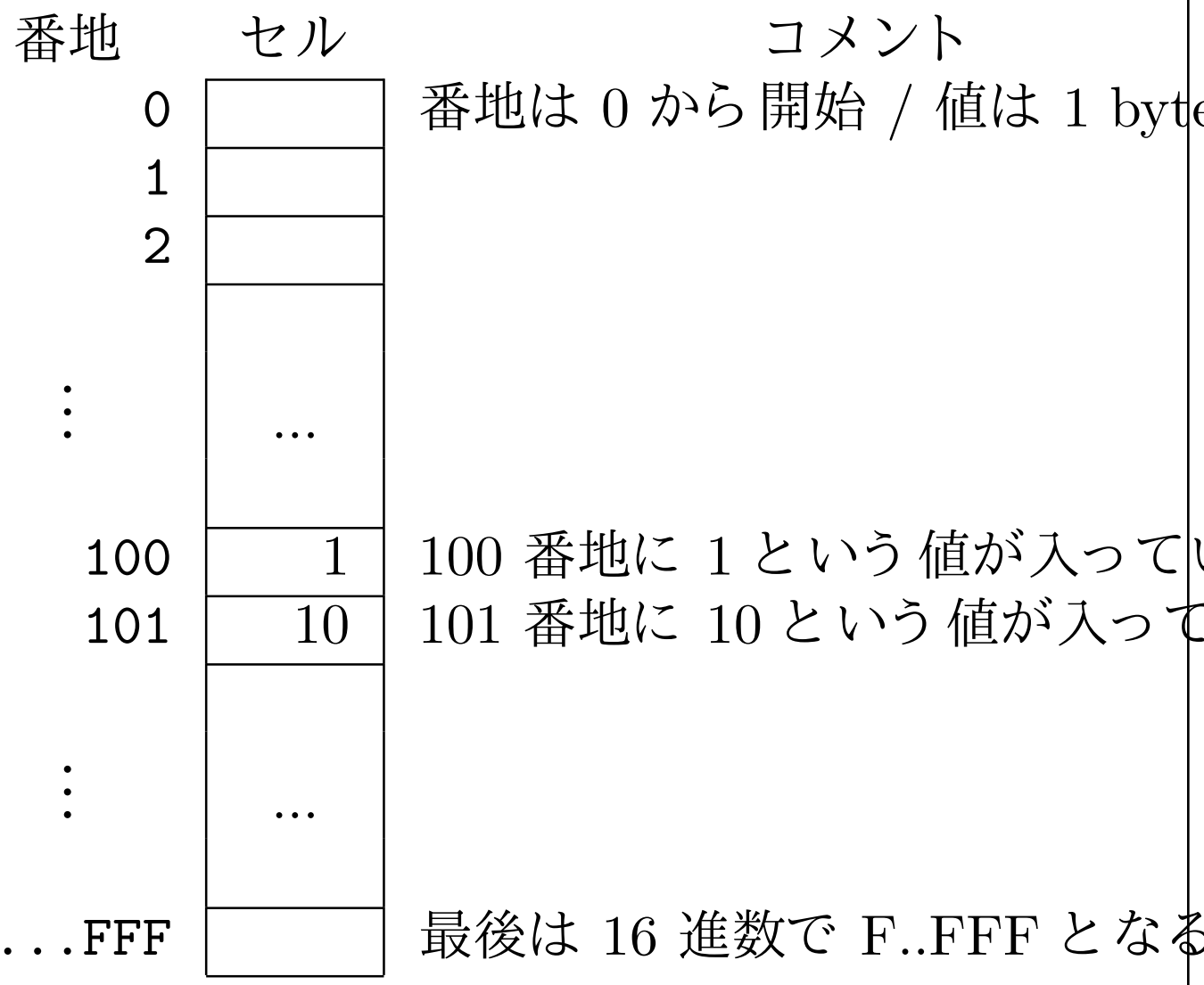
- 前回 (2013/12/20) の課題
  - 課題 1:
    - ファイル名 : 20131220-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : メモリ操作での和
    - ファイル形式 : テキストファイル(C 言語プログラムファイル)
  - 課題 2:
    - ファイル名 : 20131220-2-QQQQ.c (QQQQ は学生番号)
    - 内容 : アドレスを利用した間接参照
    - ファイル形式 : テキストファイル(C 言語プログラムファイル)

本日の課題 (2014/01/10)

- 本日 (2014/01/10) の課題
  - 課題 1: [sample-001.c 参照]
    - ファイル名 : 20140110-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : myprintf で Point2D 型を出力する %D が扱えるように拡張する
    - 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)

メモリのイメージ (再)

- □ メモリ
  - ○ セルの並んだ物
    - ▷ セルのサイズは 1 byte
  - ○ アドレス(番地)がついている
    - ▷ アドレスは 0..0 ～ F..F (16 進)
  - ○ セルの機能 (変数と同じ)
    - ▷ 情報を記録できる
    - ▷ 情報を取り出せる



□ 文字列の操作 (復習)

- 「*」:間接(参照)演算子：文字列の先頭の文字を取り出す
  - ▷ *"abc" == ’a’
- ○「[]」:添字演算子：「[n]」で n (整数値)で「n+1番目の文字」意味する
  - ▷ "abc"[0] == ’a’, "abc"[1] == ’b’, ..
- ○「*」と「[]」の関係；文字列[n] == *(文字列+n)
  - ▷ "abc"[0] == *("abc"+0) == *("abc") == *"abc" == ’a’

□ アドレス演算子「&」

- ○ アドレス演算子「&」は 間接演算子「*」の逆演算を行う
  - ▷ (&(*("abc"))) == "abc"
- ○ 変数に関しては逆が成立する
  - ▷「*(&var)」==「var」
- ○ アドレス演算子「&」の正体
  - ▷ 変数に対応したメモリセルの「アドレス」を得る演算子

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

- ポインター値
  - 「&」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
- ポインター型変数
  - ポインター値を値としてもつ変数の事
    - ※「ポインター型変数」の事を単に「ポインター」と呼ぶ事が多い

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

左辺値と右辺値

- □ 右辺値とは
  - ○ 代入によって、記録される情報の事
    - ▹ 変数は、右辺値を記録しており、右辺値を取り出す事ができる
    - ▹ ※ 普通に「値」という場合は、「右辺値」を意味している
- □ 左辺値とは
  - ○ 代入操作を行う場合に「代入の対象(セル列)を指す」情報
    - ▹ ※ 代入文の左には、「左辺値」が必要 (代入の時だけ考える特殊な値)
    - ▹ 例 1:「変数名」は、「その変数を代入の対象として指す左辺値」になる
  - ○ 左辺値は、代入文の左に現れた時にのみ、「そのままの形」で利用される
    - ▹ 他の場所(例えば、代入文の右)に現れた場合は、「右辺値」に変換される
    - ▹ その「右辺値」は、「左辺値が示す対象(セル列)が記録している値」になる
    - ▹ 例 2:「変数名」が代入文の右に現れた場合は、「変数の記録している値」になる
    - ▹ 例 3: 「a = a + 1」の中の「左の a」は「左辺値」、「右の a」は「右辺値」になる
  - ○ 左辺値は、「ポインター値の素」と考えてもよい
    - ▹ 「&」は「左辺値」から、「ポインター値」を作る演算子(アドレス演算子)
    - ▹ 「*」は「ポインター値」から、「左辺値」を作る演算子(間接参照演算子)

ポインター値を利用した間接参照

- □ 関数引数の確認
  - ○ 関数の引数は、代入文の右辺と同じ
    - ▹ ※ 実引数の値は、関数の仮引数に「代入」される
  - ○ 変数の値を変更するには、変数の左辺値が必要
    - ▹ 呼び出された関数内では、呼び出す関数の変数を変更できない(原則として)
- □ ポインター値と間接参照
  - ○ 変数名から、ポインター値を作る事ができる
  - ○ ポインター値は「右辺値」なので、関数の引数に渡す事ができる
  - ○ 間接参照演算子で、「ポインター値」から「左辺値」を作る事ができる
    - ▹ 関数に変数のポインター値を渡し、呼出した関数内で、変数の値を変更できる
- □ scanf の機能
  - ○ scanf で、「&変数名」と指定するのは、「ポインター値」を渡すため

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？: 局所変数だけしか利用しないなら OK
  - この違いが問題になるのは？
    - 大域/静的変数への「代入」操作が行われる(副作用がある)場合は駄目

スタック (Stack)

- スタック (FILO : First in Last Out)
  - 情報を「先入れ、後出し」方式で保存できる「袋」
    - push X : X を Stack に保存する(積む)
    - top (n) : Stack の先頭の情報を取り出す (n がある場合は n 番目)
    - pop : Stack の先頭を取り除く
- スタックの実現
  - 連続したセルの並び(Cell[]) と、その先頭アドレス(SP) で表現可能
    - push X : Cell[SP++] = X
    - top(n) : Cell[SP-n]
    - pop : SP--;

様々なポインター値

- 配列名
  - 「配列名」は、配列の先頭の要素のポインター値を持つ「定数(ポインター)値」を持つ
    - 例 int a[10]; の「a」は、「&a[0]」と同じ意味
- 関数名
  - 「関数名」は、関数の記録されている「関数ポインター値」を持つ
    - 「関数呼出し」は、「関数ポインター値()」の形で行われる
  - ※「関数ポインター値」は、「Code のアドレス値」を持つ

関数呼出しとスタック

- 関数呼出しの仕組
  - 実引数の値をスタックに積む
  - 現在の位置の次の場所(Code のアドレス値)をスタックに保存して、関数の場所に行く
- 呼び出された関数の振舞
  - 局所変数の分だけスタックポインターをずらす
  - 関数の本体を実行
- return 命令の振舞
  - 関数の値を記録し、スタックポインターを戻し、スタックの返り場所に行く

加変長関数の仕組

- 引数への参照
  - スタックポインターからの相対位置で参照
    - 引数が適切な所にあるかどうか？：実は何の保証もない
    - 関数のプロトタイプ宣言を使って、チェックはできる
  - 固定長引数の場合
    - 参照する個数と、位置は、固定なので予め解っているので簡単
- 可変長引数の場合
  - 参照する個数と、位置を、その場で計算する必要がある
    - ポインター計算と、キャストで、処理可能
    - 引数に関する情報が必要
- printf/scanf
  - 関数としては特別なものではない(単なる可変長引数関数)
    - 書式文字列には、「引数に関する情報」が含まれている