ソフトウェア概論 A/B

-- メモリモデルとポインター (1) --

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 : 20141205
  - 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)
  - 2015/01/23 (講議最終日)
    - 試験を行う
  - 2015/01/16 (講議最終日)
    - 模擬試験を行う
  - 2015/01/09
    - 月曜授業日 (ソフトウェア概論はない)
  - 2015/01/02, 2014/12/26
    - 冬期休暇期間中 : この講議はない
  - 2014/12/19, 2014/12/12
    - 通常講議 : メモリモデルとポインター(2) / 落穂拾い
  - 2014/12/05 (本日)
    - メモリモデルとポインター(1)

前回 (2014/11/28) の復習

- 前回 (2014/11/28) の復習
  - オセロゲーム盤を作る(5)
    - コマの引っ繰りかえし(完全版)
    - コンピュータのプレイ
  - C 言語
    - 複雑な条件式

お知らせ

- 本日の予定
  - 講義 (メモリモデルとポイター)
    - メモリモデルとは？
    - 変数とメモリ
    - ポインター型 (sizeof 演算子)
  - 引数とスタック
    - 加変長引数
  - printf/scanf
- 演習
  - 課題の提出

前回 (2014/11/28) の課題

- 前回 (2014/11/28) の課題
  - 課題 1:
    - ファイル名 : 20141128-1-QQQQ.zip (QQQQ は学生番号)
    - 内容 : 本日(2014/11/28) の段階でまでに作られたファイル集 (v?.zip)
    - ファイル形式 : zip 形式

本日の課題 (2014/12/05)

- 本日 (2014/12/05) の課題
  - 課題 1:
    - ファイル名 : 20141205-1-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 : 20141205-2-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)

メモリ

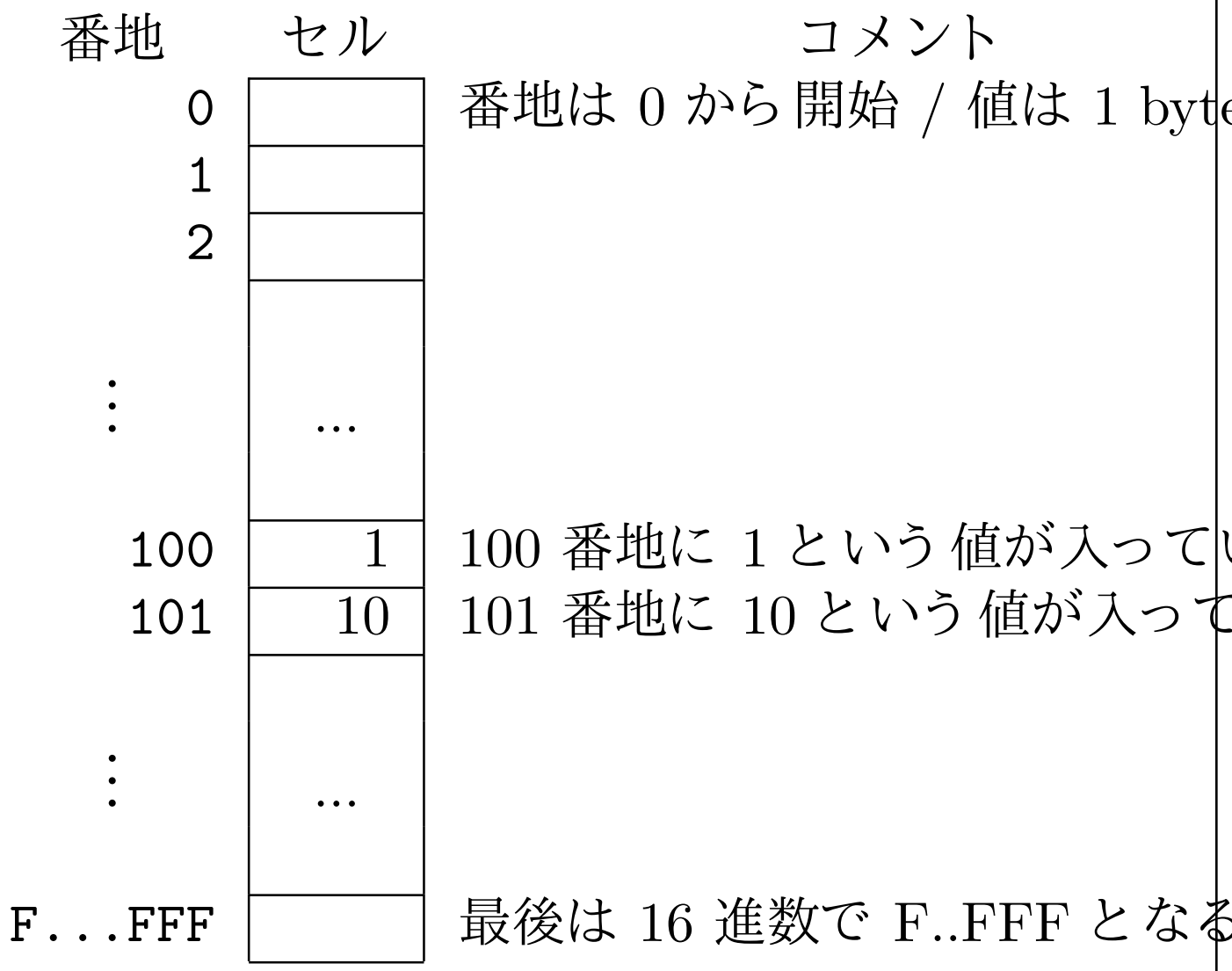
- (主)メモリ(記憶領域)とは？
  - 情報を記憶する小さなメモリセルの集まり
    - 一つのセルでは 0 ～ 255 の 256 (= 2^8 : byte) 種類の状態の内の一つ(情報)が記録されている
    - 個々のセルには(その位置を表す)番地(アドレス)がついている
  - [アナロジ] メモリ:ホテル, セル:部屋, アドレス:部屋番号, 情報:宿客
- メモリの操作：メモリセルの「記憶能力」(sample-001.c)
  - 情報の記録 (set_memory_value_at)
    - アドレスと記録する情報を指定して、そのアドレスのセルに情報を記録する
  - 情報の参照 (get_memory_value_at)
    - アドレスを指定して、そのアドレスのセルに記録された情報を取り出す

メモリセルの性質

- メモリセルの性質
  - 情報の参照は何度でもできる (sample-002.c)
    - 最後に記録した情報は何度でも取り出せる (cf. 不思議なポケット)
  - 記録できるのは一つだけ
    - 最後に記録したものだけが記録され、参照できる (sample-002.c)
  - メモリセルは独立 (sample-003.c)
    - 異るメモリセル(メモリセルの区別は番地で行う)は独立に振る舞う
  - メモリセルの記憶容量 (sample-004.c)
    - メモリセルが記録できるのは 0 ～ 255 (1 byte 内) の数値

メモリのイメージ

- 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



メモリモデル

- メモリモデル
  - C 言語の変数のモデルの一つで、「変数をメモリセルの組み合わせ」として理解する
    - C 言語の「変数の振舞い」を「考えるための仕組み(モデル)」
    - !! 「何かモデル」とは何かを理解するために利用可能な、「より簡単な仕組み」の事」
    - !! 「C 言語の変数」を「メモリモデル」を通じて理解する/ 簡単なの理解しやすい
- 実は..
  - 多くの場合、「C 言語の変数」は実際に「メモリセルの組み合わせ」になっている
    - 変数の性質(代入)はメモリの性質(記憶能力)から説明できる
- char 型変数とメモリモデル (sample-005.c)
  - char 型変数は、一つのメモリセルだと考える事ができる
    - char 型変数は address を持つ
  - char 型変数をメモリセルと同様に扱う事ができる

メモリモデルと配列

- 文字列とメモリモデル (sample-006.c)
  - 文字列は、文字の並び
    - 文字は char 型変数で記録できるので、文字列は char 型変数の並び
- 文字変数の並びと文字列 (sample-007.c)
  - アドレスが判れば、変数の内容をアドレス経由で操作できる
- 配列宣言 (sample-008.c)
  - 配列とは
    - 「複数の変数の並び」の事 (個々の変数を「配列の要素」と呼ぶ)
  - 一次元の配列宣言 (sample-008.c)
    - 変数と同様に型名の後ろに「配列名[サイズ]」の形で宣言
    - 「サイズ」の個数だけの変数が宣言される。
    - 配列の要素は「配列名[0] ～ 配列名[サイズ-1]」という「名前」になる
    - 添字：「[」と「]」の間には整数値が指定でき、配列の何番目の要素かを表す

文字列と文字型の一次元配列の関係

- 文字列と文字型の一次元配列の関係
  - C 言語の文字列
    - 文字の並んだ物 (文字コードが連続に記録されている)
  - C 言語の文字型変数
    - 文字コードを一つだけ記憶できる
  - C 言語の文字型の一次元配列
    - 複数の文字型変数が並んだもの
  - C 言語の文字列型の一次元の配列で文字列を記憶する事ができる
- 文字列の一部の操作方法 (sample-009.c)
  - 文字配列の要素を変更すればよい
- 文字列を利用した文字配列の初期化 (sample-010.c)
  - 文字配列の要素を文字列を利用して初期化できる

配列の添字, 間接(参照)演算子,アドレス演算子

- □ 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 (sample-011.c)
    - ▹ *"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

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 (sample-012.c)
    - 二つの添字からアドレスを計算すればよい (sample-013.c)
  - 始めから二次元の配列を宣言する事ができる (sample-014.c)
    - 配列の要素のアドレス計算は、自動的に行われる (sample-015.c)
    - 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 (sample-016.c)
- 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 (sample-017.c)
  - その型の変数が、何個(byte)のセルを占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 (sample-018.c)
  - 「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