ソフトウェア概論 A/B

-- 代入/while/メモリモデル --

数学科 栗野 俊一

2012/10/05 ソフトウェア概論

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
  - ○ ガイダンス (前期と同じ)
    - ▷ ルール : 他人に迷惑をかけるな (講議中は私語厳禁)
    - ▷ 心掛け : とにかく手を動かす(習うより慣れろ)
    - ▷ 評価 : レポート重視 / PC を用いて試験を行う
  - ○ 前期の復習 (プログラム作成に必要な最低限の事は学んだ)
    - ▷ プログラム/プログラミング/プログラミング言語/C 言語/コンパイラ
    - ▷ MinGW/プログラム作成手順(編集,コンパイル,リンク,実行)
    - ▷ 様々なファイル(ソース[C]ファイル,オブジェクトファイル,実行ファイル)
    - ▷ 「Hello, World」/main 関数/printf 関数
    - ▷ 関数 (関数定義 / 関数呼び出し / 引数[変数] / 返り値[return 文])
    - ▷ 命令の組み合わせ (順接 / 分岐 [if 文] / 再帰 [繰返し])
    - ▷ コーディング(情報の表現:情報と数値の対応関係)
    - ▷ データ型(表現形式と操作)/変数の型宣言/様々な型(整数,文字,浮動小数点数)
  - ○ 代入
    - ▷ 「=」(代入演算子)を利用して、変数の値を書き換える事ができる
      - ◇ [注意] 変数の「前」の内容は、失われる
      - ◇ [注意] 「=」は「代入」であり「等しい(==)」という意味では *ない*

本日の予定

- 講義
  - 代入(再)
  - 変数宣言(再)
  - while
  - メモリモデル
- 演習
  - 課題の提出

本日の課題 (2012/10/05)

- 先週の課題
  - 課題 1:
    - ファイル名 : 20121005-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : 成績処理を行う
    - 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)
    - 引数で自分の成績(S,A,B,C,D)をいれて、順位を表示させてみよ
- 今週 (2012/10/05) の課題
  - 課題 1:
    - ファイル名 : 20121005-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : 代入を用いた四則の計算
    - 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 : 20121005-3-QQQQ.c (QQQQ は学生番号)
    - 内容 : while と代入を用いた階乗の計算
    - 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)

代入(復習)

- □ 代入とは？
  - ○ 変数の値を「変更」する事
    - ▷ 言い替えると..？「変数は代入が行われると変数の値が変化する」
  - ○ 変数は常に値を持つが..
    - ▷ 代入された前と後では「値が変化」する
- □ 変数に値を代入するには？
  - ○ 代入文 (「変数名 = 式」) を使う (sample-016)
    - ▷ 式の値が計算された上で、変数に代入される
    - ▷「=(等号)」を使っているが、「等しい」という意味ではない
    - ▷ C 言語では「等しい」という意味には「==」を使う
  - ○ 変数の値は何度でも参照可能 (sample-017)
    - ▷ 計算結果を何度も利用する時には変数に入れておく
  - ○ 代入の右の式の中で自分自身の現在の値が利用できる (sample-018)
    - ▷ やっぱり、「等号」じゃない

代入と関数引数(復習)

- 関数引数は「代入」か？
  - 代入は、「変更」がおきる
    - 元々変数がもっていた値が「失われて」しまう
  - 関数引数は、変数の値を決めるが、「変更」するわけではない
    - 変数の値が保存されている
    - 同じ「名前」を利用していても「異なる変数 (アドレスが異なる)」

局所変数宣言(再)

- 引数でない変数
  - ブロックの先頭で変数宣言すれば、新しい変数が利用できる
    - メモリの一部がその変数名に割り当てられる
  - 引数との違い
    - 宣言直後は値が定まっていない(何が入っているかは不明)
    - 必ず、値を代入してから利用すること(可能なら初期代入すること)
    - 「初期代入」=,「変数宣言」+「代入」(実は微妙に違うが..)
- 名前の有効範囲
  - 「変数名」の有効範囲は宣言の後からブロックの終りまで
    - ブロックの外からはその名前が利用できない
    - 同じ名前で宣言しても異なるメモリになる(可能性が高い)
  - 名前とメモリは「独立」な事に注意
    - 名前は利用できなくてもメモリは利用できる(ポインターの利用)
- 仮引数は特殊な局所変数
  - 仮引数は呼び出し側で初期値を指定する事ができる

入力・処理・出力(再)

- 変数を利用したプログラムパターン
  - 入力・処理・出力の三つの部分からなる
    - 入力：変数を初期化する
    - 処理：(計算を利用しつつ..) 変数の値を変更する
    - 出力：変数の値を「結果」として出力する
  - 関数定義もこの形
    - 引数の値の設定、関数の本体、値の返却

while 文

- while 文
  - 命令を繰り返すための「構文」
    - while (「繰返し条件」) {「繰返す命令」}
    - 「繰返し条件」が成立している限り、「繰返す命令」を繰り返す
  - while 文を利用する上での注意
    - 「繰返す命令」によって「繰返し条件」(内の変数の値)が変化しないと不味い
    - もし、そうでなければ「無限 Loop」(それはそれで利用価値はあるが..)
    - 「繰返す命令」の中に「代入文」(変数の値の変更)が不可欠
- 二つの繰返し手段 (while vs 再帰)
  - while 文と再帰は共に、「繰返し」(同じ文を複数回繰り返す)手段
    - その意味で、機能は同じ
  - while 文は代入文が不可欠 / 代入文との相性が良い
    - 代入文の効果 :「効率」を高める事ができる

while 文から再帰へ

- while 文は、再帰で書く事ができる
  - while 文のパターン

```
「引数」=「値」;
result =「初期値」;
while (「条件」(「引数」) ) {
    result =「処理関数」( result,「引数」 );
    「引数」=「更新処理」(「引数」);
}
```

  - 再帰関数のパターン
    - 関数呼出

```
result = 関数(「値」);
```

    - 関数定義

```
func(「引数」 ) {
    if (「条件」(「引数」) ) {
        return「処理関数」( func(「更新関数」(「引数」),「引数 );
    } else {
        return「初期値」;
    }
```

コンピュータによる情報の表現

- コンピュータは電気で動く
  - 電気と「計算」の関係は？
- 電気回路のスイッチ
  - 「on：通電 / off：断線」の二つ状態を表現できる
    - cf. スイッチが on なら電球が光るが、off なら消える
  - スイッチの on/off を 1/0 に対応させる
    - 1/0 を「電気回路の on/off」でコーディングする
    - on と off, Yes と No, 真と偽, etc..
- スイッチ一つで表現できる「情報量」
  - 1/0 の二つの状態の内の一つが表現できる
    - 1 bit の情報が表現できる
    - bit はデジタルコンピュータが扱う情報量の最小単

bit 列による「情報」の表現

- 複数のスイッチ (bit) による表現力
  - n 個数のスイッチ (bit) で、2^n の状態が表現可能
    - 1 byte = 8 bit : 256 個の状態が表現可能
- コーディングによる情報の表現
  - コーディングの規則は「表」でよい
    - 「文字」の時は「ASCII Code 表」を使った
    - 半角英数記号の個数は 100 個程度なので 7 bit で表現可能
    - C 言語の「char 型」は 1 byte (= 8 bit > 7 bit)
    - 全角文字は 6879 文字 (JIS X 0208 の場合) あるので 1 byte では無理
  - 整数値 (-2,147,483,648～2,147,483,647) : 32 bit int
- sizeof 演算子
  - その型の byte 数を表す
    - sizeof(char) -> 1 / sizeof(int) -> 4

二進法

- 二進法とは
  - 2 を底とする位取り記数法
    - cf. 十進法は、10 を底とする位取り記数法
    - cf. 十六進法は、16 を底とする位取り記数法
- 二進法の性質
  - 数を 0/1 の二つの記号(数字)の並びで表現できる
    - cf. 十進法は、0 ～ 9 の十個の記号(数字)が必要となる
    - bit を単位とするコンピュータと相性がよい
- コンピュータでの数の表現
  - 基本は二進法の数 (0/1 の並び) と bit 並びを対応付けする
    - 非負の数値を表現する場合 : unsigned (負号なし) 数
    - 正負の数値を表現する場合 : signed (負号あり) 数

2の補数表現

- 1の補数表現とは
  - bit の 0/1 を反転させたもの
    - 例 (8 bit) : 01001000 -> 10110111
- 2の補数表現とは
  - 1の補数表現に 1 を加えた物
    - 例 (8 bit) : 01001000 -> 10111000
- コンピュータでの負号付き整数
  - 非負の数は、最上位 bit (負号 bit) が 0 で残りは二進数
    - 最上位 bit (負号 bit) で正負が判る
  - 負の数は、正の数の 2 の補数表現
    - 例 (8 bit) : 72 -> 01001000 / -72 -> 10111000

bit (論理)演算

- bit (論理)演算 (boolean)
  - 真偽値 (True / False) の演算
    - 論理積 (かつ / and / &) : 共に真なら真、それ以外は偽
    - 論理和 (または / or / |) : どちらか一方でも真なら真、それ以外は偽
    - 論理否定 (でない / not / ~) : 偽なら真、それ以外は偽
    - 排他論理和/論理差 (異なる / xor / ^) : 両方が異れば真、それ以外は偽
  - これら演算はスイッチで表現できる
- 1 bit の足し算 (ハーフアダー)
  - 1 桁の二進数の和は、2 桁になる可能性がある
    - それぞれの桁の計算は、論理演算で表現可能
- 論理演算による「計算」
  - 基本的な数値演算も論理演算の組み合わせで計算できる
    - 電気機械であるコンピュータが計算できる理由
    - より詳しい内容は、電子工学や電気工学の先生に聞こう

オーバーフロー

- コンピュータの情報
  - 有限のサイズをもつ
    - 表現可能な範囲が限られている
- 現実の世界 (例えば整数)
  - 無限の範囲をもつ
    - コンピュータの表現に対応しないものがある
    - 表現できる数を計算した結果が、再び表現できない可能性がある
- オーバーフローとは
  - 計算結果が表現可能な値の上限を超えること
    - signed char (8 bit : -128 ～ 127) の場合は 127 + 1 の結果は..
  - 浮動小数点数 (double) でも同様な事がおきる

型の性質

- コンピュータによる「情報」の表現
  - コンピュータでは数値(二進数)しか扱えない
    - 「情報」を「直接」は表現できない
    - コーディング(二進数と「情報」の対応)によって「間接」的に扱う
- 型
  - C 言語の中で「情報」の表現方法に名前を付けたもの
    - コーディングの名前と考えてもよい
    - 整数の表現 : int / 文字の表現 : char / 実数値の表現 : double
  - 型による違い
    - 計算結果が異なる : 3/2 -> 1, 3.0/2.0 -> 1.5
    - サイズが事なる : sizeof (int) -> 4, sizeof (double) -> 8

メモリモデルとポインター値

- メモリ
  - 複数のセルの並び
    - 個々のセルにはアドレス(番地)がつき、区別される
    - 個々のセルは 1 byte のサイズを持つ
    - セルの並びで一つの情報を記憶する (cf. sizeof)
- 変数とは？
  - 複数の連続したセルに「型」をつけたもの
    - 型によって、振舞が異なる事に注意
- ポインター値
    - 型と「アドレス値」をもつ
    - 「アドレス値」は、メモリの番地と同じ
    - 変数名の前に &' を付けるとポインター値が得られる
    - ポインター値の前に '*' を付けると元の変数と同じ振舞をする
      - 「 v == *(&v) 」が恒等的に成立する

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

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