ソフトウェア概論 A/B

-- 代入と制御構造 --

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 : 20150703
  - 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(後ろから)

- 今後の予定
  - 2015/07/24 (講議最終日)
    - 試験 / Note-PC 必須 / PC のトラブル対応はしない / 課題提出最終日
  - 2015/07/17 (講議最終日前)
    - 前期のまとめ / 模擬試験 / Note-PC 必須 / 環境を整える
  - 2015/07/10 (次週)
    - TOEIC の試験で休講？(講義があれば、試験前の落穂拾いを行う)
  - 2015/07/03 (今日)
    - 本日：代入と制御構造

前回(2015/06/26)の復習

- 前回(2015/06/26)の内容
  - 浮動小数点数 (double 型) の性質
    - 小数(実数)を表すための型：小数点が利用できる
    - 計算に誤差を含む場合(殆どの場合)があるので 0 と比較してはいけない
  - 数値計算の例：浮動小数点数 (double 型) の利用
    - 二分探査による方程式の数値解法
    - 定積分の数値計算 (リーマン和 / モンテ=カルロ法)
  - 解析的(数学的)には「解け」なくても、数値的(計算機的)には「解け」る
    - 解析的解：誤差を含まない解 / 数値的解：誤差を含んでも良い解
    - 数値的解を得るためにも「数学」が必要(計算機を学ぶ上で、数学が重要)
- コンピュータによる数の表現の問題
  - コンピュータは「有限の物」しか扱えない
    - 整数型：大きな数を扱う事はできない
    - 浮動小数点数型：誤差を伴う

前回 (2015/06/26) の課題

- 前回 (2015/06/26) の課題
  - 課題 01:
    - ファイル名 : 20150626-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : 一つ浮動小数点数値をキーボードから入力し、その立方根を出力する
    - 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)
  - ※ 課題 02 は、今週 (2015/07/03) に回す

本日 (2015/07/03) の課題

- 本日 (2015/07/03) の課題
  - 課題 2015/06/26-02:
    - ファイル名 : 20150626-02-QQQQ.c (QQQQ は学生番号)
    - 内容 : C-Curve を描画するプログラム
    - 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 2015/07/03-01:
    - ファイル名 : 20150703-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : 方程式 \cos{x} - x = 0 の正値解を数値的に求めなさい
    - 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 2015/07/03-02:
    - ファイル名 : 20150703-02-QQQQ.c (QQQQ は学生番号)
    - 内容 : 標準正規分布 f(x) = \frac{1}{\sqrt{2\pi}}\exp{-x^2/2} に対する定積分 \int_{0}^1 f(x) dx の値を数値的に求めなさい
    - 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)

数学関数

- C 言語の標準ライブラリにある数学関数
  - sin/cos/tan/exp/log .. (詳しくは検索)
- 数学関数の利用
  - ソースファイル (*.c) の中の先頭で #include <math.h> とする
  - リンク時に -lm を付ける

型の変換とキャスト

- 「値」は型を持つ
  - 型が違えば同じ演算を施しても異る結果になる
    - (例) 整数型(int 型) 5/2 = 2, 実数型(double 型) 5.0/2.0 = 2.5
  - 同じ「数」である「1」を表すのに「1(整数型)」と「1.0(実数型)」は異なる振舞をする
- 型の変換(キャスト)
  - 「数」同士 (double と int) であれば、「値」を変更為ずに「型」を変更できる
    - 値の前に「(型名)」とすると、「その型に変換される」
    - 型だけでなく値も変る事もある
  - [例 1 : 整数型→実数型]「(double)1」は「1.0」と同じ振舞をする
  - [例 2 : 実数型→整数型]「(int)1.0」は「1」と同じ振舞をする
  - [例 3 : 実数型→整数型]「(int)1.2」は「1」と同じ振舞をする
    - 実数から整数の場合は値の小数点以下が「切り捨て」られ、整数の値の範囲になる
    - <<注意>>「(double)((int)1.2)」は、「1.0」になってしまう事に注意

暗黙の型変換と型の昇格

- □ 暗黙の型変換
  - ○ 関数の引数に型の異る値を指定され、それが変換可能なら自動的に変換される
    - ▹ 型変換の為には「引数の型の宣言」が利用される事に注意
- □ 型の昇格
  - ○ 四則演算の二つの引数の型が異る場合は、順位の低い方が高い方に変換される
    - ▹ (これまで学んだ型では) char < int < double である
    - ▹ [例] 3/2=1, 3.0/2 = 3/2.0 = 3.0/2.0 = 1.5
  - ○ <<注意>>「文字列(型?)」は(現時点では..)他の型に変換する事に意味がない
    - ▹ 「(int)"123"」とか「(double)"1.23"」とやっても「123」や「1.23」にならない
    - ▹ 前者は意味不明な数値になるし、後者はエラーになる

シンボル定数の定義と define

- □ シンボル定数の定義
  - ○ 定数値に名前をつける事ができる
    - ▹「定数値に名前を付ける」事を「シンボル定数の定義」と呼ぶ
    - ▹ その「名前」が「シンボル定数」となる
    - ▹ 定義後は「定数値」の代りに「名前(シンボル定数)」が使えるようになる
  - ○ シンボル定数の定義方法
    - ▹ #define 定数名 定数値
  - ○ 定義例
    - ▹ #define PI 3.141592
    - ▹ #define EPSILON 0.000001
- □ シンボル定数の効用
  - ○ マジックナンバーの排除
    - ▹ マジックナンバーとは：プログラムの中に散見される「意味不明(マジック)」な定数値の事
    - ▹ マジックナンバーはプログラムを読み難くする
  - ○ マジックナンバーの代りにシンボル定数を利用する
    - ▹「定数名」に「意味のある名前」を使えば、読み易くなる
  - ○「共通の値」を「同時に変更する」事が可能になる
    - ▹ シンボル定数の定義にある定数値を変更するだけ

代入

□代入とは

○概念：「変数」に「値」を「割り当て」る「操作」

- ▹代入「後」は、その変数の値は、代入さ(割当ら)れた値に「変化」する
- ▹代入「前」の値は、「失われ」る
- ▹代入の「前」と「後」という「時間」の概念の把握が必要となる

○表現：代入の構文

- ▹「変数名」=「式」 [例] a=1+2; (変数 a に 3 (= 1+2) を代入)
- ▹「=」は、「代入」を表現する(等号[等しい]ではない!! / 等号は「==」)

□局所変数宣言

○概念：局所変数を宣言する

- ▹関数(ブロック)内のみ(局所的)で有効(利用可能)な変数を宣言する
- ▹「仮引数変数(実は局所変数の一種)」以外にも、変数が増やせる

○表現：局所変数の宣言

- ▹「変数の型名」「変数名」 [例] int a; (整数型の変数 a を宣言)
- ▹cf. 仮引数変数は、実引数の値で、「代入済」の変数
- ▹未代入の変数の値は「未定(プログラムミスの代表例 !!)」
- ▹変数は宣言と同時に「初期化(最初の代入)」できる(すべき) [例] int a=1;

while 文

- while 文
  - 概念 : 繰返しのため構文
    - 同じ命令を繰り返す事ができる (cf. 再帰呼出し)
  - 表現 : while 文
    - while (「条件」) { 「繰り返す命令」 }
    - 「条件」の部分は、if と同じ
    - 「繰り返す命令」の中には、「代入」が必須 (でないと「条件」が変化しない)
- while 文 vs 再帰
  - while 文は常に再帰に変換できる (実は原理的に逆も可能だが自明ではない)
    - func() { while (条件) { 文 } } → func() { if (条件) { 文; func(); } else {} }
  - その意味で、再帰の方が表現力がある(優秀)といえる
    - 逆に(工学のトレードオフの典型例)、while 文の方が「効率」がよい

printf

- printf：超高機能出力関数
  - print with format (書式付き出力)
    - 単なる文字列出力関数ではなかった (cf. s_print_string : 単機能)
  - 「書式('%' + 書式指示)」を指定する事により何(基本型+文字列)が出力できる
    - printf ("%d", 123); / printf ("%f", 1.23); / printf ("%c", 'a'); / printf ("%s", "abc");
  - 文字列の中に出力を埋め込む事ができる
    - int a=123; printf ("int a=%d\n", a);
  - 複数のデータを一度に出力する事ができる
    - int a=123; double b=1.23; printf ("int a=%d, double b=%f\n", a, b);
  - print の動作原理
    - 後期にちゃんと話すので、今回は我慢 !!

scanf

- scanf : 超高機能入力関数
  - scan with format (書式付き入力)
    - 色々な型のデータを読み込む事ができる (cf. s_input_int : 単機能)
  - 「書式('%' + 書式指示)」を指定する事により何(基本型+文字列)が入力できる
    - int a; scanf ("%d", &a);
    - !! a の前の「&」は「お呪い」(後期にちゃんと話す)
  - 書式や機能などについても printf と同様に考えてよい
    - 文字列の中から値を取り出す事もできるのだが.. (結構難しいのでさけるのが無難 ..)