ソフトウェア概論 A/B

-- 関数再び, 値と型と式 --

数学科 栗野 俊一 / 渡辺 俊一

2013/06/21 ソフトウェア概論

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

前回 (2013/06/14) の復習

- 前回 (2013/06/14) の復習
  - 前回 (2013/06/14) から二周目に入っている
    - 前回の最初は、当然「Hello World」
    - 意味があり完全に動く最も簡単なプログラム : プログラム作成の出発点

Hello, World (再)

- Hello, World
  - まず、「動く」事が大事
    - 内容の意味がわからなくても、必要な部分だけ変更すればよい
  - 差分プログラミング (必要な部分だけ改良する)
    - 「巨人の肩に乗る」(c) アイザック・ニュートン
- 「お呪い」の解説
  - #include : ファイルの読込みを行う
    - <stdio.h> → c:\mingw\include\stdio.h
    - "foobar.h" → 同じフォルダか -I で指定したフォルダ
  - 引数の (int argc, char *argv[])
    - argv[0] : コマンド名を表す文字列
    - argv[1] ～ : コマンドラインの引数の 1 つ目 ～
  - int ～ return : プログラムを呼び出す
    - コマンドプロンプトで、ERRORLEVEL に保存される
    - if errorlevel n で、利用する事ができる
- ポイント
  - 新しい話 :「そんなんもあったな」でよい (また、やるし..)
  - 二度目の話 : 基本は、今回でしっかり身に付ける (でも、またやるが..)

様々な入力と出力

- 入力
  - お呪い : #include "s_input.h"
  - 入力される「モノ」
    - 整数 : s_input_int()
    - 文字 : s_input_char()
    - 文字列 : s_input_string()
- 出力
  - お呪い : #include "s_print.h"
  - 出力される「モノ」
    - 整数 : s_print_int(整数)
    - 文字 : s_print_char(文字)
    - 文字列 : s_print_string(文字列)
    - 改行 : s_print_newline()

お知らせ

- 本日の予定
  - 関数(再)
  - 値と型と型宣言
  - 整数型/浮動小数点数型
- 本日の目標
  - 演習
    - 課題の提出

前回 (2013/06/14) の課題

- 前回 (2013/06/14) の課題
  - 課題 1:
    - ファイル名 : 20130614-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : コマンド引数に指定した言語名よって挨拶を行う
    - 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)

本日の課題 (2013/06/21)

- 本日 (2013/06/21) の課題
  - 課題 1:
    - ファイル名 : 20130621-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : キーボードから入力された二つの整数型の値の四則と余りの結果を表示する
    - 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 2:
    - ファイル名 : 20130621-2-QQQQ.c (QQQQ は学生番号)
    - 内容 : キーボードから入力された二つの浮動小数点数型の四則の結果を表示する
    - 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)

値と型

- Data(データ) と Code(コード)
  - Data : C 言語のプログラムで「操作の対象」となる物
  - Code : C 言語のプログラムで「操作を行う」命令部分
    - 共にメモリに記録されている (将来説明する)
- 値(あたい)と型(かた)
  - Data は、「値」と「型」を持つ
    - 型 : Data がどんな集合の要素で、どの様な演算ができるかを決める属性
    - 値 : 型で指定された集合の要素の一つ (値が同じでも型が異れば異る物)
  - Data の値と型の例
    - 整数値の「1」: 型は「整数型」で、値は 1
    - 文字の「1」: 型は「文字型」で、値は '1'
    - 一文字からなる文字列「1」: 型は「文字列型(仮)」で、値は "1"
  - 型が異れば、できる「演算」も異るし、「結果」も異る
    - 例1 : 9 + 1 = 10, '9' + 1 = ':', "9" + 1 = ""

式

- 式：複数の Data から新しい Data を作る表現
  - 例 : 1 + 1 → 2 , *"abc" → 'a', "abc" + 1 → "bc"
  - 「式」から Data を得る事ができる (つまり、値と型が得られる)
  - 即値(式) : 定数を表す表現
    - 1 : 整数型の 1 という値を持つ Data を表す「即値式」
    - '1' : 文字型の '1' という値を持つ Data を表す「即値式」
    - "1" : 文字列型の "1" という値(?)を持つ Data を表す「即値式」
  - 演算子との組み合わせ
    - 整数の四則 : 12 + 5 → 17, 12 - 5 → 7, 12 * 5 → 60, 12 / 5 → 2
    - 「+」,「-」,「/」,「*」は、二つの整数型の Data から一つの整数型の Data を作る
    - 文字列の先頭 : *"abc" → 'a' : 型が変化している事に注意 !!
  - 関数呼出し : 関数は Data を返す
    - 例 1 : s_input_int() : キーボードから入力された整数に対応する Data
    - 例 2 : s_input_string() : キーボードから入力された文字列
    - 例 3 : rand() : 呼び出す毎に異る整数が返る

関数(再び)

- 関数
  - 関数とは => 命令列に名前を付けたもの
    - 関数定義時：自分で、命令列に名前を付けて、新しい関数が作れる
    - 関数実行時：名前を指定する(関数呼び出し)だけで、その命令列が実行できる
  - 引数付き関数とは => 命令の一部を仮引数で表現した関数
    - 関数定義時：命令の一部を仮引数(変数)で表現し、そこでは内容を指定しない
    - 関数実行時：仮引数に入る内容は関数呼び出し時に「実引数」で指定する
  - ライブラリ => 他の人が作った関数定義の集まり
    - 中身を知らなくても、機能が解れば、関数を呼び出す事で利用可能
    - cf. printf, s_midi_play, ..

関数と引数(再)

- 関数宣言の頭部
  - 「void 関数名 (char * 仮引数 [, char * 仮引数 ..])」
    - 関数の前には「void」を付ける (お呪い)
    - 仮引数の前には「char *」を付ける (お呪い)
  - 仮引数 :「文字列」の代わりに利用できる「変数」
- 関数宣言の体部
  - 「 { 命令列 }」
    - 命令列の中で「文字列」が表れて良い所に「変数」が利用できる
- 「char *」とは何か？
  - 引数の「型」宣言 (「文字列」を表現する『もの』)
    - 引数に何が入るか(情報の種類)を指定している
  - 引数に入る情報の種類によって、「型の宣言」も変更する必要がある
    - 今迄は「文字列」しか扱わなかったので「char *」固定だった
    - 他の種類のデータを処理するなら対応して変更する (cf. char)
- 関数の引数の型の一致
  - 仮引数(の変数)の型と、実引数(の値)の型は一致している必要がある

引数付き関数の作り方(再)

- □ 引数とは
  - ○ 関数に与える事により、関数にその引数に対応した挙動をさせるもの
    - ▷ 引数付き関数の定義：引数の値によって挙動が変る
    - ▷ 引数付き関数の利用：指定したい挙動をさせるための値を指定する
    - ▷ cf. 三角関数：引数の角度によって異なる値を返す
- □ 引数付き関数の作り方
  - ○ 似ている二つ関数を一つの引数付き関数にまとめる
    - ▷ 関数の本体の部分を、同じ部分と違う部分に分ける
    - ▷ 違う部分は「変数」に置き換えて、一つの関数定義にまとめる
    - ▷ 関数の仮引数の所に、「変数」を追加する
    - ▷ 呼出す側は、実引数に、「違っていた部分の内容」を指定する

条件分岐(再)

- 引数の内容によって振舞いを「大幅」に変更したい
  - if 文と strcmp 関数を利用して対応できる
    - strcmp 関数：二つの文字列を比較する
  - if (!strcmp (A, B)) { X } else { Y }
    - A と B が同じなら X を、そうでなければ Y を行う
  - 「else if」を使うと更に複数の命令が選べる
    - if (C1) { P1 } else if (C2) { P2 } .. else { Pn }
    - C1 の時 P1、そうでなく C2 の時は P2 .. いずれでもないと Pn
  - おまじない
    - #include <string.h>
  - strncmp (A, B, N);
    - A と B の先頭の N 文字だけを比較する
    - !strncmp ("abc", "abz", 3); : 等しくない
    - !strncmp ("abc", "abz", 2); : 等しい

再帰呼び出し(再)

- 文字列を順番にみてゆく
  - 「"abc" + 1」は「"bc"」と同じ振舞いをする
    - どうゆう仕組かは今回は説明しない
  - 次々と 1 を加えれば、どんどん短かくなる
    - 最も短かくなったかは、空文字(「""」)と比較すれば判定できる
- 再帰呼び出し
  - 普通の関数は、別の関数を呼出す事ができた
    - 「自分の中」で「自分自身」を呼出す事ができる !! : 再帰呼び出し
- 再帰呼び出しと帰納法
  - 再帰呼び出しは、帰納法の考え方で問題を解く場合に利用できる
    - 再帰呼び出しが上手く行く事は、帰納法で証明できる (数学との関係)
  - 再帰呼び出しをする場合は次の二点が重要 (帰納法と同じ)
    - 最も小さい場合 (ここでは、文字列が "" の場合) には終了する
    - そうでない時は、再帰呼び出しするが、その時には文字列を短くする

再帰呼び出しの考え方(再)

- 目標
  - 「全部」をやりたい
    - でも一挙にはできない
- 対策
  - そこで問題を二つに分ける
    - 扱いやすい一部分：これは、そのまま対処してしまう
    - 残り全部：(残り)「全部」なので、再帰呼び出しする
- 注意点
  - 「全部」が空っぽの時を忘れずに処理する

三つの基本制御構造と万能性(再)

- 三つの基本制御構造
  - f を関数, A,B を命令、p(x) を条件とする時、次の三つの基本構造がある
  - [順接] f() { A B }
    - f は A をしてから B をする
  - [分岐] f(x) { if (p(x)) { A } else { B } }
    - f は p(x) が成立すれば A そうでなければ B をする
  - [繰返] f(x) { if (p(x)) { A f(x') } else {} }
    - f は p(x) が成立する限り A を行う
    - x' は x から計算される
- 万能性
  - 任意のプログラムこの三つの基本制御構造で構成可能
    - 「三つの基本制御構造」を憶えればあとは組み合わせを考えるだけ !!

整数型

- 整数型
  - C 言語での整数
    - 表現できる範囲は限られている
    - 32bit の場合は −2147483648 から 2147483647
  - 宣言：int で行う
  - 計算：四則が可能 +, -, *, /
    - / は整数割り算なので、小数点以下は切捨てになる
  - 比較：大小比較、等号、不等号が使える
    - a > b : a が b より大きい
    - a >= b : a が b 以上
    - a == b : a と b が等しい (= でないことに注意 !!)
    - a != b : a と b が等しくない
- 整数型の出力 (当分は..)
  - s_print.h の中の s_print_int を使う (sample-001.c)

浮動小数点数型

- double 型
  - 小数点付きの数を表現する
    - C 言語内での表現：小数点付きの数 (cf. 123.456)
    - 出力：s_print_double (s_print.h 内)
    - 入力：s_input_double (s_input.h 内)
  - 様々な数学的な関数が利用できる
    - sin/cos/exp/log/etc.. cf #include <math.h>
  - (当然) 四則の計算ができる：3.0/2.0 → 1.5 (cf. 3/2 → 1)