ソフトウェア概論 A/B

-- 前期の復習 / 標準 I/O ライブラリ / 代入 --

数学科 栗野 俊一 / 渡辺 俊一

2019/09/20 ソフトウェア概論

伝言

私語は慎むように !!

- 出席パスワード : 20190920
- 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
- VNC Server Address : 10.9.154.227
  - Password : vnc-2018
- 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 の電源を入れておく
  - ネットワークに接続しておく
  - 今日の資料に目を通しておく
- 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

ガイダンス

- ガイダンス (5 分で終了予定)
  - 前期(ソフトウェア概論 A)を受けて居た方へ
    - 成績結果に不明な点があれば、講義終了時に申し出る
    - 後期(ソフトウェア概論 B)も、方針は前期と同じです(以上 !!)
  - 以前に、栗野の講議(コンピュータ概論 A/B, ソフトウェア A/B) を受けた事がある方へ
    - 方針は似た様なものです(人間は一緒なので..)
    - 前期を取っていないのなら、LAN/ubuntu の環境を作りましょう : TA を捕まえてください
  - 初めて栗野の講議を受ける人へ
    - 手を挙げてください
    - 必要なら簡単に説明します
    - 詳しくは Web にある以前の資料を参照してください

状況確認

- 状況確認：何かに手を付ける時に最初にやる事
  - 過去(実績)、現在(仕事)、未来(目標)を考える
    - できている事(過去)、したい事(未来)が解れば、すべき事(現在)が解る
  - 「我々はどこから来たのか？我々は何者か？我々はどこへ行くのか？」
    - フランスの画家ポール・ゴーギャンの有名な作品の名前
- 過去：我々はどこから来たのか？
  - ソフトウェア概論 A を学んできた (笑)
    - 詳細は次の OHP から..
- 未来：我々はどこへ行くのか？
  - ソフトウェア概論の目標:
    - 自分の望む事を行うプログラムを C 言語で記述できるようになる
    - そのプログラムを実行する事で望みを叶える事ができるようになる
- 現在：我々は何者か？
  - 何ができるようになってるのか？
    - ソフトウェア概論 A の単位は取れたよね..
  - 過去を振り返って、現状を確認し、未来(ソフトウェア概論 B)に向って出発しよう..

前期の復習 1：学習目標の確認

- 目標の確認
  - 一言で言うと.. プログラムが書けるようになりたい !!
    - では、「プログラムが書ける」とは、どう言う事か？
- 「プログラムが書ける」ために学ばなければならない三つの要素
  - [プログラム理解] プログラムとは何か？
    - 「書く」対象(目標)が解らなければならないので、それ自身を学ぶ必要がある
  - [C 言語] プログラムはどう表現すれば良いか？
    - 「書く」という行為は「表現」行為なので、「表現手段」としての「C 言語」を学ぶ必要がある
  - [操作手順] プログラムはどう作り、どう利用すれば良いか？
    - 「書く」だけでは意味がない、それを「動かす」方法を学ぶ必要がある

前期の復習 2 : 操作手順

- □ プログラムの作成/実行の手順 (コンパイル言語の場合)
  - ○ 起草 [頭] : プログラムの仕様/設計 (アイディア)
    - ▹「どんな(仕様)」プログラムを「どのような手法(設計)」で作成するか
    - ▹「プログラムを実行したら『どうなる』か ?」を「 *予め* 考え」ておく
  - ○ 編集 [エディタ] : ソース(Soruce) プログラムの作成 (*.c)
    - ▹ 自分が望むプログラムをプログラミング言語(C 言語)で表現する
  - ○ 翻訳(コンパイル) [コンパイラ] : オブジェクト(Object) プログラムの作成 (*.o)
    - ▹ ソースを翻訳して、オブジェクトを作成する
    - ▹「表現」に問題があれば、ここで確認(エラーになる)でき、最初に戻って検討する
  - ○ リンク [リンカ] : 実行ファイルの作成 (*.exe)
    - ▹ オブジェクトとライブラリを結合して実行ファイル(アプリファイル)を作成する
    - ▹「部品の揃え方」に問題があれば、ここで確認でき、最初に戻って検討する
  - ○ 実行 : 実行結果の確認 (実行結果:画面出力など)
    - ▹ プログラムを実行し、その結果を見る事
    - ▹ 適切な結果が得られなければ、最初に戻って検討する
- □ 確認実習
  - ○「Hello World」プログラムを作成、実行してみよう

前期の復習 3 : プログラム理解

- □ プログラムとは何か？
  - ○ 計算機への(複数の)指示をまとめたもの (狭義)
    - ▹ 複数の(単純な)指示を適切に組合せる事により、意味のある(複雑な)結果が得られる
    - ▹ 命令の組み合わせ: 文字列を 1 文字ずつ短かくしながら表示する
    - ▹ 意味のある結果 : 文字による三角形が表示される
- □ プログラムを作成するためにどう考えるか 
  - ○ 基本は「分割統治法」を利用
    - ▹ (Top-Down 設計) 全体の問題を複数の小さい問題に分割
    - ▹ (個々の問題の解決) 個別撃破
    - ▹ (Bottom-Up 実装) 解決案を組合せて全体を作る
  - ○ 例
    - ▹ 全体の問題(部屋の片漬け/手がつかない)
    - ▹ 分割 (取り敢えず机の上から, 次にベット上, 床は最後に ...)
  - ○ 分割は「再帰的に」行い、「直にできる事」まで「分解」する
    - ▹ 例 : ベッドは、まず上に乗っている物をどかし、シーツをかえ、下に隠してあるごにょごにょを...
  - ○ 分解が済んだら、個々に解決し、それを組上げる事により、問題全体を解決
    - ▹ 例 : 色々な所から出たゴミは、袋にまとめて、明日の朝出しておく..

前期の復習 3 : プログラム設計

- □ プログラムの作成法
  - ○ 問題を分割すれば良い .. だとしても ???
    - ▹ 「何(what)」を「どのよう(how)」に、「どこ(where)」で分割すべきか ?
    - ▹ 例 : 「ベットの掃除」を「頭側」と「足側」に分ける事が意味があるか ? 右と左では ?
- □ 分割の「パターン」を身に付ける
  - ○ 時間の前後関係 (X は A をやってから B をする) で分ける
    - ▹ 状況 : ベットを片漬けるとゴミが出るが、ゴミ箱が一杯で捨てられない
    - ▹ 時間分割 : まず、ゴミ箱内のゴミを捨てて、次に、ベッドから出たゴミを入れる
  - ○ 条件に応じて対応 (P の時は A をやって、そうでなければ B をする) を分ける
    - ▹ 状況 : 色々な種類のゴミが出て来た
    - ▹ 条件分割 : 紙屑は燃えるゴミの袋に、アルミ缶やペットボトルは資源ゴミの袋に..
  - ○ 同じ事を繰り返す事 (A を繰り返せば済む) で分ける
    - ▹ 状況 : 大量のゴミが出て来きた
    - ▹ 繰返し分割 : 取り敢えず一つの袋に入るだけゴミを袋詰めして外に出す、ゴミがなくなるまで、これを繰り返す
  - ○ 問題解決のパターンが知られている (cf. アルゴリズム論)
    - ▹ 状況 : 本が本棚に雑多に押し込まれていて、どこに何があるかが解らない
    - ▹ ソーティング : 書名順になるように並べ替えれば、簡単に探せるようになる

前期の復習 4 : C 言語学習 (1)

- C 言語 :「A Programming Lanugage C」
  - 「C 言語」は「プログラミング言語」
    - プログラムを記述するための言語
  - 「C 言語」で、プログラムを「記述(表現)」できる
    - 「言語」なので、「単語」と「構文(文法)」がある
    - 利用可能な「単語」と適切な「構文」を使わないと正しい「プログラム」にならない
    - (所謂)「コンパイルエラー」は、「『文法的に正しい文』でない」と言う表示
  - !!! 「文法的に正しい文」が「意味のある文」とは限らない
    - 例 : 日本語における「文法的に正しい」が「意味のない」文:「机がガソリンを食べた」(童話や詩なら...)
- C 言語の理解
  - 利用可能な「単語(基本命令)」と「構文(文法)」を憶えて、「形式」が整られるようにする
  - 単語と構文の意味(機能)を理解し、それから得られる「筋」が通るようにする

前期の復習 5 : C 言語学習 (2)

- プログラムの最小構成を身に付ける
  - 最小構成 : main 関数の作成
    - 例 : Hello, World
- C 言語の基本命令と構文を身に付ける
  - 定義文 (単語に意味や情報を与える記述 [単語定義])
    - 関数定義 : 関数の引数と値の型情報をつけ、その関数を呼出した時に実行される命令を定義し、その関数の呼出しができるようにする
  - 基本命令 (単独で文になるもの)
    - 関数呼出し (printf, s_print_int, .. 様々な基本命令 / ライブラリ理解)
  - 構文規則 (文から新しい文を作る仕組)
    - ブロック構造 (「{」～「}」: 複数の文を一つの文にまとめる仕組)
    - 制御構造 (if 文 / 再帰 / return 文)

前期の復習 5 : C 言語学習 (3)

- C 言語の学習 (現状確認と今後の話)
  - 3 周する予定
    - 現状は(切りが悪いが..) 2 目周の途中
- これからの予定
  - 取り敢えず 2 周目を終らす
    - 代入 / データ構造の話をしたい
  - 次は 3 周目に入るが 3 周目はアプリ開発をメインに進めたい

お知らせ

- 本日(2019/09/20)の予定
  - ガイダンス
  - 前期の復習
  - 代入と制御構造
    - 代入 / 局所変数の宣言 / while 構文
  - 標準 I/O ライブラリ
    - printf/scanf, fprint/fscanf, fopen/fclose
- 本日の目標
  - 演習
    - 課題の提出

今週 (2019/09/20) の課題

- 今週 (2019/09/20) の課題
  - 課題 20190920-01:
    - ファイル名 : 20190920-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : 「Hello World」のプログラム作成
  - 課題 20190920-02:
    - ファイル名 : 20190920-02-QQQQ.c (QQQQ は学生番号)
    - 内容 : printf による書式指定出力
  - 課題 20190920-03:
    - ファイル名 : 20190920-03-QQQQ.c (QQQQ は学生番号)
    - 内容 : scanf による書式指定入力
  - 課題 20190920-04:
    - ファイル名 : 20190920-04-QQQQ.c (QQQQ は学生番号)
    - 内容 : 変数宣言と代入文
  - 課題 20190920-05:
    - ファイル名 : 20190920-05-QQQQ.c (QQQQ は学生番号)
    - 内容 : while 構文
  - ※ ファイル形式は、いずれもテキストファイル(C 言語プログラムファイル)

先週 (2019/07/26) の課題

- 先週 (2019/07/26) の課題
  - なし

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(割り当てら)れた値に「変化」する
- ▹(その変数が保持していた..) 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 構文

- while 構文
  - 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 構文
    - 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 構文の方が「効率」がよい

標準 I/O ライブラリの利用

- 標準 I/O ライブラリ
  - I/O とは Input(入力) と Output(出力) の事
    - Input : プログラムが、プログラムの外から、情報を読み込む(Read)
    - Output : プログラムが、プログラムの外へ、情報を書き出す(Write)
    - [ポイント] 「外」はプログラムの置かれている環境で異る (i.e OS)
    - 例 : 「あいさつ」をするとき日本とアメリカで異るの嫌(だから全て英語で.. それも困るが..)
  - 標準 I/O ライブラリの役割
    - 異る環境でも、プログラムとしては、同じ方法で I/O を行うための仕組
    - 環境の違いはライブラリが吸収 (挨拶は手を振るで OK)
  - 標準 I/O ライブラリの機能
    - I/O する物 : 文字、文字列、数値(基本データ)
    - I/O の相手先 : キーボード(I), ディスプレイ(O),ファイル(I/O)

printf

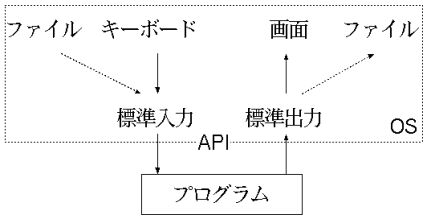
- printf：超高機能出力関数
  - print with format (書式付き出力)
    - 単なる文字列出力関数ではなかった (cf. s_print_string : 単機能)
  - 「書式('%' + 書式指示)」を指定する事により何(基本型+文字列)が出力できる
    - printf ("%d", 123); / printf ("%f", 1.23); / printf ("%c", 'a'); / printf ("%s", "abc");
  - 文字列の中に出力を埋め込む事ができる
    - int a=123; printf ("int a=%d\n",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

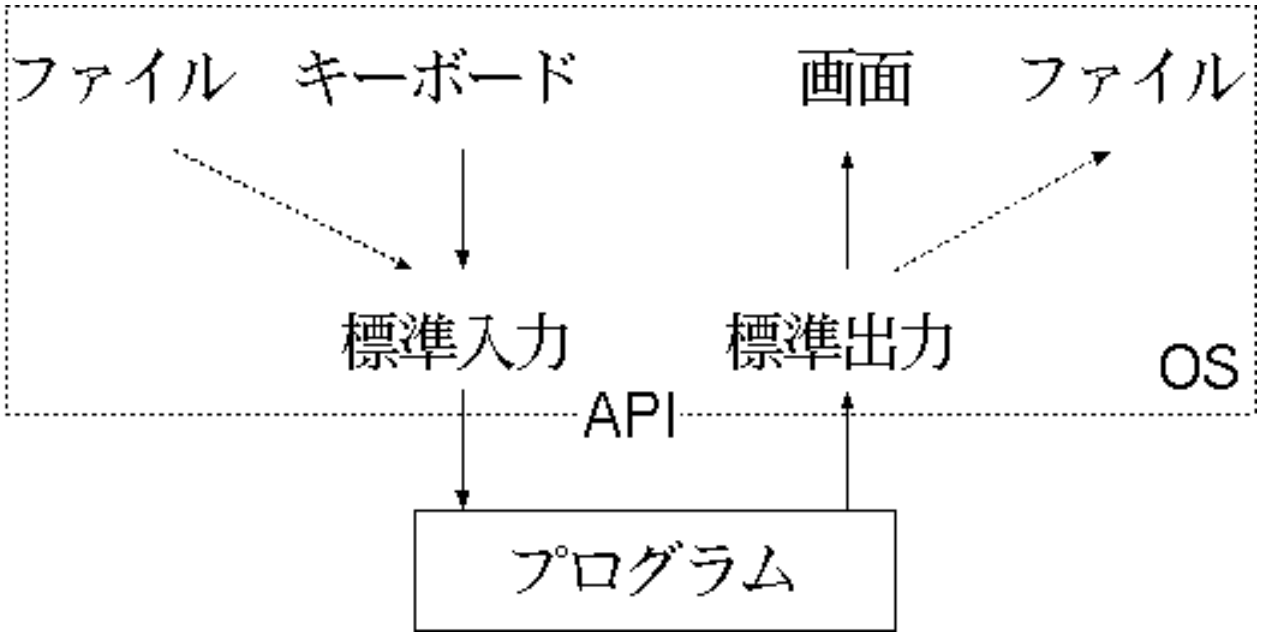
プログラムの「標準 I/O」とは

- □「標準」I/O とは？
  - ○getchar は、キーボードからの入力 / putchar は、画面への出力
    - ▷なら、「標準」は「キーボード」と「画面」の事？
    - ▷No, But, Yes (ちがう、だけど、そうなんだ..)
    - ▷「?? それって、意味不明なんだけど...」
  - ○つまり..
    - ▷「標準入力」は、飽く迄も「標準入力」で、「キーボード」と異る (だから、No)
  - ○しかし..
    - ▷「普段」、「標準入力」は「キーボード」に「対応付け」されている(だから、Yes)
    - ▷「標準出力」と「画面」の関係も同様
  - ○改めて「標準 I/O」とは？
    - ▷そのプログラムにとって、「特に指定為ずに入出する先」の事
    - ▷プログラムは、「入出する先が本当は何になっているか」は、気にしない
    - ▷対応付けは、OS が行うので、「プログラムには無関係」な事柄



リダイレクション

- 標準 I/O とデバイス
  - 「標準 I/O」と、「実際の I/O 先(デバイス)」の対応は？
    - 普段は OS が、「キーボード」と「画面」に対応付けている



(C) 坂本直志(東京電機大学工学部情報通信工学科) http://edu.net.c.dendai.ac.jp/literacy/2004/12/index.xml

- リダイレクション(入出力切り替え)とは
  - 「標準 I/O」の先を「別のデバイス」に切り換える事
    - 本当は「色々な物(デバイス)にリダイレクト(切替)」可能だが、ここではファイルだけ

リダイレクション(redirection)の方法

- コマンドラインでのリダイレクションの方法
  - 「標準出力」を「ファイル」へ「リダイレクション」
    - コマンドラインに「> 出力ファイル名」とすると「標準出力」が「出力ファイル」に切り替わる
    - そのコマンドの標準出力先(これまでは、画面)が「出力ファイル」に切り替わる
  - 「標準入力」を「ファイル」からに「リダイレクション」
    - コマンドラインに「< 入力ファイル名」とすると「標準入力」が「入力ファイル」に切り替わる
    - そのコマンドの標準入力元(これまでは、キーボード)が「入力ファイル」に切り替わる

stdin/stdout

- C 言語での標準入出力 (standartd Input/Output)
  - putchar/getchar, printf/scanf は、「標準入出力」に I/O を行う
    - プログラムで入出力を切り換える事ができる
  - C 言語で標準で利用できる入出力先
    - 標準入力 : stdin / 標準出力 : stdout / 標準エラー出力 : stderr
  - fputc/fgetc, fprintf/fscanf では、標準入出力以外の I/O 先が指定できる
    - putchar(ch) は fputc (ch, stdin) と同じ
    - getchar() は fgetc (stdin) と同じ
    - scanf (fmt, ..) は fscanf (stdin, fmt, ..) と同じ
    - printf (fmt, ..) は fprintf (stdout, fmt, ..) と同じ
  - stderr : 標準エラー出力
    - 標準出力(stdout)が、「本来の情報の出力先」を意味するのに対し、「異状な場合の特別な情報出力」を行う
    - OS で、標準出力をリダイレクトしても、標準エラーは影響を受ない

ファイル I/O

- ファイルを対象とする I/O
  - ファイルは Open してから利用を開始し、利用が終ったら Close する必要がある
    - fopen : ファイルを open する関数 (ファイルを「開く」関数)
    - fclose : ファイルを close する関数 (ファイルを「閉じる」関数)
- ファイルポインター
  - ファイルポインターって？
    - 「ファイル情報管理構造体」へのポインター(詳しくは後日)
  - ファイルを open すると、「ファイルポインター」が手に入る
  - ファイル(外にある)を内部で扱うには「ファイルポインター」を経由する
  - ファイルを close する時にも「ファイルポインター」を指定する
- ファイルへの I/O
  - ファイルポインタを使って、fprintf/fscanf で行う
  - 文字の入出は fputc/fgetc も利用できる
- ファイル I/O の注意
  - ファイルは Open しないと使えない / 利用が終ったら必ず Close する