ソフトウェア概論 A/B

-- 型変換とキャスト --

数学科 栗野 俊一 / 渡辺 俊一

2016/12/09 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 : 20161209
- 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
  - 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

今後の予定

- 今後の予定(後ろから)
  - 2017/01/20 (講議最終日)
    - 試験を行う
  - 2017/01/13 (講議最終日前)
    - 模擬試験を行う
  - 2016/12/30 / 2017/01/06
    - 冬期休暇期間中：この講議はない
  - 2016/12/23
    - 天皇誕生日(祝日)
  - 2016/12/16
    - 落穂拾い
  - 2016/12/09 (本日)
    - データ構造 (8) / 動的データ構造 / ファイル I/O

前回(2016/12/02)の内容

- □ 前回(2016/12/02)の内容
  - ○ C 言語の変数の占める「メモリ量」は、「固定(静的:変化しない)」
    - ▹ 宣言された変数の個数だけメモリが消費される
  - ○ どのメモリをどのように利用するかをコンパイラが「事前」に決める事ができる
    - ▹ 「メモリの管理」の「自動化」が可能
    - ▹ 「メモリの領域に名前」を付ける事ができる(-> 変数名)
  - ○ 動的メモリ管理 (<-> 静的メモリ管理)
    - ▹ 「実行時」に「利用するメモリの量を決め」たい (動的:変化する)
    - ▹ alloc 関数(malloc/calloc)で、必要な時に必要なだけ確保できる
  - ○ alloc 関数と free の使い方
    - ▹ ヘッダー "malloc.h" を include
    - ▹ alloc 関数に必要なサイズを指定して、メモリを確保
    - ▹ 成功した場合はポインタ値が、失敗した場合は NULL が返る
    - ▹ 利用が終ったら free で「解放」する (メモリ管理を自分で行う)

前回 (2016/12/02) の課題

- 前回 (2016/12/02) の課題
  - 課題 20161125-01:
    - ファイル名 : 20161125-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : 動的なメモリの利用

本日の課題 (2016/12/09)

- 本日 (2016/12/09) の課題
  - 課題 20161118-02: (これは前回 2016/11/18 の残り)
    - ファイル名 : 20161118-02-QQQQ.c (QQQQ は学生番号)
    - 内容 : アドレスを利用した間接参照
  - 課題 20161209-01:
    - ファイル名 : 20161209-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : 整数値のキュー(queue)
  - 課題 20161209-02:
    - ファイル名 : 20161209-02-QQQQ.c (QQQQ は学生番号)
    - 内容 : 二つのファイルを比較して最初に異る場所を表示する
- ※
  - 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)

動的なデータ構造

- 静的なデータ構造
  - 配列/単純変数：保存できるデータのサイズは固定
    - alloc を使っても、配列ならば、その中に入るデータの個数は固定
    - メモリの無駄を覚悟すれば、「大きなサイズの配列」でも代用可
- 動的なデータ構造
  - 保持できるデータ数が可変長(本質的に動的)
    - alloc を利用しないと扱えない
- 動的なデータ構造の例
  - List (リスト)：複数の要素を「並べた」もの

構造体へのポインター値

- 構造体のタグ
  - 構造体には、構造体を区別する名前(tag:タグ:構造体名)がつけられる
    - struct タグ { .. }
  - 「sturct タグ」は構造体型の名前として利用できる
    - これまでは、これを typedef でサボッていた
  - ポインタによる自己参照を行う場合は、タグがどうしても必要になる
- 構造体へのポインタ
  - 構造体へのポインタを利用して、構造体の要素にアクセスできる
    - p が、要素 x を持つ構造体へのポインタなら (*p).x で参照で可能
    - 構造体へのポインタの要素の参照はよくあるので、省略記号がある
    - 「(*p).x」の代りに「p -> x 」という表現が利用できる

ファイル I/O

- ファイル操作
  - ファイル：データを恒常的記憶するための仕組み
    - メモリの内容は、PC の電源を切ると失われる
  - リダイレクション(復習)
    - 「<」、「>」を利用して、プログラムの入出力をファイルに変更可能
    - せいぜい、入力と出力に一つずつしかファイルが利用できない
  - ファイル I/O ライブラリ
    - 任意のファイルに対する読み(Read/Input)書き(Write/Output)する機能

FILE 構造体

- FILE 構造体による File I/O
  - fopen 関数を利用して、ファイル I/O を管理する FILE 構造体へのポインタ値を得る
    - FILE 構造体は、「どのファイルの何処を参照しているか」を管理する情報
    - FILE 構造体を経由して、少しずつファイルからデータを得る事が可能
    - open に失敗した場合、fopen 関数は NULL を返す
  - 利用が終った FILE 構造体は fclose で必ず close する
    - 特に write 時には close しないとデータが失わる(保存されない)可能性が生じる
    - read の場合も、close しないと「資源の無駄使い」になる
    - cf. memory の alloc/free と同様な関係
  - データの入出力
    - fgetc/fputc : 文字単位の File I/O
    - fscanf/fprintf : ファイルに対する printf/scanf
    - fread/fwrite : ファイルに対する固定長の