ソフトウェア概論 A/B

-- ポインター(2)/スタック/prinf/落穂拾い --

数学科 栗野 俊一 / 渡辺 俊一

2015/12/18 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
- 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 : 20151218
  - 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

- 今後の予定(後ろから)
  - 2015/01/22 (講議最終日)
    - 試験を行う
  - 2015/01/15 (講議最終日前)
    - 前半は、落穂拾い (2) / 後半は、模擬試験を行う
  - 2015/01/08 / 2015/01/01
    - 冬期休暇期間中：この講議はない
  - 2014/12/25
    - 後期予備日
  - 2014/12/18 (本日)
    - データ構造 (9) / ポインター(2)・落穂拾い (1)

前回(2015/12/11)の内容

- □ 前回の内容
  - ○ ビット処理 (前回の積み残し)
    - ▷ 計算機内のデータ : 単なる bit 列
    - ▷ 色々な型のデータがあるようにみえるのは「コーディング」のマジック
    - ▷ ビット単位での操作(演算)が可能 : 論理積、論理和、etc..
  - ○ ポインター型 : データ構造 (8)
    - ▷ ポインター値 : アドレス値 + 型情報 (サイズ + 操作方法)
    - ▷ ポインター値の作り方 : 変数名(左辺値を持つ物)の前に & をつける
    - ▷ ポインター値の利用 : * を前につけると「変数(左辺値)」に変る
    - ▷ ポインター型とポインター変数 : 「型」+「*」で、「その型へのポインター型」を表す
      - ◇ ポインタ値をもつ変数(ポインター変数)が宣言できる
  - ○ ポインター値の操作
    - ▷ アドレス値 : ポインター値に 1 を加えるとアドレス値がサイズだけ増加
      - ◇ ※ ポインター値の差は整数値になる
    - ▷ キャスト : 「(ポインタ型)」をポインター値の前に付ける事により、型情報が変更できる

お知らせ

- 本日の予定
  - 引数とスタック
    - 加変長引数
    - printf/scanf
  - 変数のスコープとエクステンション
    - 大域変数/静的変数/動的メモリ管理(malloc/free)
  - 落穂拾い(1)
    - 共用型 / バイナリファイル / etc..
- 演習
  - 課題の提出

前回 (2015/12/11) の課題

- 前回 (2015/12/11) の課題
  - 課題 20151204-01: (前々回の積み残し)
    - ファイル名 : 20151204-01-QQQQ.c (QQQQ は学生番号)
    - 内容 : ポインターを利用して、整数変数の値を正値にする

本日の課題 (2015/12/18)

- 本日 (2015/12/18) の課題
  - 課題 20151211-01:
    - ファイル名 : 20151211-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : ポインター演算子を利用して構造体を操作
    - 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 20151218-01:
    - ファイル名 : 20151218-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : Point2D 型に対応した myprintf を拡張して作る
    - 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)
- ※
  - 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)

ポインターの様々な応用

- ポインターの様々な応用
  - 大量の情報(構造体の値)のコピーを防ぐ
    - 関数への引数に指定された値はコピーされる(コストが高い)
    - ポインター値を渡せば、値のコピーを為ずに参照できる(値を変更されるリスク)
  - 「配列名と添字の対」を「ポインター値」として扱う
    - 特に、「ポインター値」を関数の返り値として利用する場合に便利
  - 呼出し側で用意した変数の値を変更したい
    - リスクが大きいので、勧められない
    - 単純変数の場合は、値を渡し、返り値を代入しなおした方が良い
  - 動的な構造の作成
    - サイズが可変なデータ構造(動的構造)を扱うために必須
    - 3 年次の「アルゴリズムとデータ構造」で学ぶ：スタック、リスト、木構造、etc..

スタック構造

- □ スタック構造 (FILO : First In Last Out)
  - ○ 複数のデータを保存する仕組の一つ (cf. Queue : FIFO)
  - ○ 最初に入れた物 (First In) が最後に出て来る (Last Out)
    - ▹ 最近入れたものが最初に取り出される (棚上げ)
  - ○ 基本操作(典型例の一つ/幾つか流儀があるが、互換である)
    - ▹ void Push(Type data) : Data をスタックに積む(入れ込む)
    - ▹ Type Pop(void) : Data をスタックから下す(取り出す)
- □ スタックの実装
  - ○ 配列(スタック本体) + スタックポインター (sp) の組み合わせ [001,002]
    - ▹ 設計 : スタックは 0 から使用 / sp は「次」を指す
    - ▹ (別設計例) : スタックは最後から使用 / sp は「今」を指す

printf の謎とリスク

- printf の謎
  - 疑問：可変な引数の謎
    - 引数の個数が違っても良い理由は？
    - 引数の型がが違っても良い理由は？
  - 答え：書式の解釈と型変換
    - 書式から、引数の個数や、型情報を取り出す
    - キャスト演算を利用して、メモリ上の値を適切に取り出す
- printf/scanf のリスク
  - 型情報：安全にメモリを利用する仕組
    - コンパイラが型情報を利用して、サービスすると同時に安全性を担保
  - printf/scanf：自ら(関数/プログラマ)が型情報を操作している (高機能の理由だが危険な理由でもある)
    - 書式指定と引数の指定の対応を誤ると大変
    - 特に、scanf は、メモリの内容を書き換えてしまうので、更に危険

ファイルとファイル I/O

- ファイル：データを大量(250 G)に、永続的に保存する仕組(だが、遅い)
  - cf. メモリ：高速だが、少量(8G)だし、電源を切ると消えてしまう
- ファイル I/O
  - コンピュータが「直接」処理できるのは、メモリ上のデータ
    - ファイルに記録されている情報は直接操作できない
    - ファイル上の情報を、一旦メモリに読込み(In)、処理したら、ファイルに書き出す(Out)必要がある
  - フィアイル上の全ての情報を同時に全てメモリに取り込むの非効率
    - ファイル上の一部の情報を参照(メモリに取込み)ながら、処理をする形が最適
  - ファイルポインター：ファイルの一部を参照するための仕組
- ファイル操作
  - Open(ファイルを開く)：ファイルの利用を開始する事
  - Close(ファイルを閉じる)：ファイルの利用を終了する事
    - ファイルは必ず Open してから使い、最後に必ず Close する
  - Read/Write：ファイルとメモリの間でデータのやり取りを行う

C 言語でのファイル操作

- C 言語でのファイル操作
  - C 言語ではファイル操作を行う関数があり、それで操作する
    - ※ printf/scanf 等と同様「言語の機能」ではない(ライブラリで実現)
  - FILE 構造体とファイルポインター
    - 使用中のファイル(開いている)の情報(現在の参照場所)を保持する
  - C 言語のファイル操作関数
    - 関数 fopen の返り値として得られ、関数 fclose の引数で使用を終了
    - fscan/fprintf : 値が可変長な形式で表現されたファイルの I/O
    - fread/fwrite : 値が固定長な形式(メモリと同じ形式)で表現されたファイルの I/O