ソフトウェア概論 A/B

-- オセロゲーム盤(3) --

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 : 20141114
  - 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

前回 (2014/11/07) の復習

- 前回 (2014/11/07) の復習
  - ソフトウェアライフサイクル：ソフトウェアの作成手順
    - (上流) 要求分析/設計/コーディング/テスト/保守 (下流)
  - ソフトウェア開発の技法：ソフトウェアの作成技法
    - ウォータフォール：上流から下流への水が流れるように(後戻りしない..)
    - プロトタイピング：一部の機能だけを先に一通り作る(結果を見ながら設計)
    - テストファースト：テスト項目を先に作成してから、テストに合う様に作成
  - C 言語：配列引数の特性
    - 関数の引数に「配列名」を渡す事ができる
    - あたかも「配列全体」が渡されているように振る舞う (関数内で、配列要素の変更ができる)
    - 【比較】通常の変数は「値のコピー」が渡される(関数内で、変数の値は変更できない)
  - オセロゲーム盤を作る (2)
    - キーボードからの入力
    - 着手チェックとコマの引っ繰りかえし (一方向)

お知らせ

- 本日の予定
  - オセロゲーム盤を作る(3)
    - 着手チェックとコマの引っ繰りかえし(完全版)
    - ボードの保存と呼出し
  - C 言語
    - 再帰と繰返し
    - 分割コンパイルと Makefile (make)
    - FILE I/O
- 本日の目標
  - 演習
    - オセロゲーム盤を動かしてみる

前回 (2014/11/07) の課題

□ 前回 (2014/11/07) の課題

- 課題 1:
  - ファイル名 : 20141114-1-QQQQ.c (QQQQ は学生番号)
  - 内容 : 本日(2014/11/14) の段階でまでに作られた othello.c (v?.c)
  - 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 (2014/11/14)

- 本日 (2014/11/14) の課題
  - 課題 1:
    - ファイル名 : 20141114-1-QQQQ.c (QQQQ は学生番号)
    - 内容 : 左の着手条件のチェックを行う
    - 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)
  - 課題 2:
    - ファイル名 : 20141114-2-QQQQ.zip (QQQQ は学生番号)
    - 内容 : 本日(2014/11/14) の段階でまでに作られたファイル集 (v?.zip)
    - ファイル形式 : zip 形式

C 言語：条件分岐

- 条件分岐 (条件によって二つの命令の一方を実行)
  - if 文 の構文
    - if (「条件」) {「条件成立時の命令」} else {「条件不成立時の命令」}
  - if 文 の意味
    - まず「条件」をチェック (0 なら偽、それ以外なら真)
    - 「条件」が真なら「条件成立時の命令」を実行、そうでなければ「条件不成立時の命令」を実行
- if 文の色々なパターン
  - else 節の省略：「条件不成立時の命令」がない時
    - 「if (「条件」) {「条件成立時の命令」}」だけで「else 以下」は省略
  - 「命令が一つ」の時：
    - 「if (「条件」)「条件成立時の命令」」だけで「{」、「}」は不要
    - 注意：「{」、「}」の省略はお勧めできない !!

Othello 盤：コマの引っ繰りかえし

- 着手チェック：その場所にコマが置けるか？
  - 着手条件 (仕様を確認)
    - その場所にコマがない
    - その場所にコマを置く事により相手のコマを引っ繰り返せる
- コマを引っ繰り返せる
  - 何れかの方向 (8 方向ある) で、相手のコマが挟める
    - 個々の方向で確認し、どこか一つでも引っくり返せれば OK
    - ※ 挟んだ相手のコマは引っくり返して、自分のコマにできる
- その方向で相手のコマが挟める
  - その方向に相手のコマが(一つ以上)並んでいて、その先に自分のコマがある

Othello 盤：その方向で相手のコマが挟める

- 仕様：要求された「機能」の内容
  - その方向で相手のコマが挟める(再)
    - その方向に相手のコマが(一つ以上)並んでいて、その先に自分のコマがある
- 設計：仕様の実現手段を考える
  - P(dir) :「dir の方向でコマが挟める」という事を確認する述語
  - 発想：一般的な事を考える前に、具体例で考える (右方向なら..)
  - 事例 (* で o を挟んだ状態..)
    - P(右) : *o*, *oo*, *ooo*, *oooo*, ..
  - 一番左の二つを外してみる.. (規則性が見付かるか ..)
    - Q(右) : *, o*, oo*, ooo*, ..
  - Q(右) の再帰的な定義
    - Q(右) : 今みている所が '*' か、'o' なら、一つ右が Q(右)
  - P(右) の Q(右) を利用した定義
    - P(右) : 今みている所 ' ' で、右が 'o' で、その先が Q(右)

C 言語：分割コンパイルと make

- 分割コンパイル
  - C 言語のプログラム
    - 複数の関数定義の記述 (但し、少なくても main 関数が一つ含まれている)
  - *.c (C 言語のソースファイル) には複数の関数の定義ができる
    - 一つのソースファイルに、全ての関数の定義をいれてもよい (may)
    - 関数の定義を複数のソースファイルに分けて定義してもよい (may)
  - ファイルを分けた場合
    - 個々にソースコードをコンパイル (分割コンパイルする) : オブジェクトファイルができる
    - 最後に全てのオブジェクトファイルをまとめる (リンクする)
- make と makefile
  - 分割コンパイルは、作業量が増える
  - コンパイル作業を自動化したい
    - makefile を作成し、make コマンドで作業する

分割コンパイルの利点

- 分割コンパイルの欠点
  - コンパイルの作業が大変になる
    - これは make に任せれば OK
  - プログラム全体の効率が(多少)悪くなる事がある
    - これは、コンピュータのハードの進化(高速化/大容量化)に任せてよい
- 分割コンパイルの利点
  - 「分割されている」事自身：影響範囲が狭まる
    - 一度に関係する範囲が小さくなる：「人間(プログラマ)」に取って大変重要
  - 「再利用」ができる
    - 別のプログラムで、同じソースファイル/オブジェクトが利用できる
    - チェックも別々に行える (単体テスト)
- ソフトウェア工学の原則
  - 目的：ソフトウェアの生産性を高める事
  - 手段
    - 再利用：典型的で、最も成功している(成功する)手段
    - 分割：再利用のために必要で、かつ、問題を簡便化するための手法

C 言語：プロトタイプ宣言とヘッダーファイル

- ファイル分割を行う
  - 関数定義が複数のファイルに分割される
    - 関数間の引数の型チェックの情報が必要になる
- プロトタイプ宣言
  - 関数の返り値と引数の型を宣言する
    - 型チェックに利用される
  - プロトタイプ宣言の構文
    - 「extern 関数宣言の頭部 ;」: extern を先行し、体部に「;」を付ける
    - 例 :「extern func (double v);」
- ヘッダーファイル (*.h)
  - プロトタイプ宣言を共有するための仕組
    - プロトタイプ宣言を記述したファイル(ヘッダーダイル)を作り include する
    - 関連する定数宣言や型宣言なども一緒にいれるとよい

ソフトウェア工学：単体テスト

- 単体テスト
  - 個々の関数を単独でテストする
    - プログラムは、複数の部品から成るので、個々の部品のテストを行う
    - 部品の一つが駄目なら全体も駄目に決っている
    - 全体のテスト(結合テスト)の前、全ての単体テストを行っておく
    - <注意> 全ての単体テストが成功しても結合テストが成功する保証はない
- テスト main 関数
  - 単体テストを行うために、作られる main 関数
    - 単体テストは、テストする対象の関数と main 関数から、テストプログラムを作成し実行する事で実現
- make と単体テスト
  - 単体テストも makefile に記述しておくとよい
    - 「テスト・ファースト」プログラミングスタイルを身に付ける