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

2013/11/08 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 (2013/10/25) の復習

- 前回 (2013/10/25) の復習
  - オセロゲームのゲーム盤の作成
    - ゲーム盤のデータ構造の設計
    - step by step プログラミング (少しずつ機能を確認しながらプログラムする)
  - C 言語の概念
    - シンボル定数(名前付きの定数)の定義と #define (マジックナンバー)
    - 二次元配列 (配列の配列)
    - for 文
    - break 文

お知らせ

- 本日の予定
  - オセロゲーム盤を作る(その三)
- 本日の目標
  - 演習
    - オセロゲーム盤を動かしてみる(その三)

前回 (2013/10/25) の課題

- 前回 (2013/10/25) の課題
  - なし

本日の課題 (2013/11/08)

- 本日 (2013/11/08) の課題
  - 課題 1:
    - ファイル名 : 20131108-1-QQQQ.mk (QQQQ は学生番号)
    - 内容 : Makefile の作成
    - ファイル形式 : テキストファイル(Makefile 形式)

分割コンパイル

- 分割コンパイル
  - 一つのファイルに全部入れても良いが..
    - 複数に分割する事も出来る
  - 分けるメリット
    - 柔軟性が生れる：リンク時にも制御できる(Test Case)
    - 相互の影響が小くなる：編集時の置換の失敗
  - 分けるデメリット
    - 情報の共有が難しい：ヘッダーファイルの利用
    - コンパイルリンクの手間手間が増える：make & Makefile の利用

ヘッダーファイル

- ヘッダーファイル (*.h)
  - 複数の C ファイル (*.c) で共有する情報を記載したファイル
    - #include を利用して *.c の中に取り込む
  - 共有する情報なので、基本は「宣言」が入る
    - #define 等の「定数定義」
    - typdef 等の「型の定義」
    - 関数のプロトタイプ宣言
    - 大域変数の型宣言
- ヘッダーファイルの構造
  - 同じファイルが二度読み込まれても困らないような仕組
    - #ifdef ～ #endif : ある定数が宣言されていたら、すでに読込み済

関数のプロトタイプ宣言と大域変数の型宣言

- □ 関数のプロトタイプ宣言
  - ○ 関数の引数と返り値を示す「関数名の型」宣言
    - ▹ 構文 : extern 関数の頭部;
    - ▹ 例 : extern void main(int argc, char *argv[]);
  - ○ プロトタイプ宣言では引数の名前を省略できる(が、しないのが普通)
    - ▹ 例 : extern void main(int, char *[]);
- □ グローバル(大域)変数の型宣言
  - ○ グローバル変数の型を宣言する
    - ▹ 構文 : extern グローバル変数の型宣言;
    - ▹ 例 : extern FILE *stdin;
  - ○ グローバル変数を利用する場合は、どこか一つの *.c ファイルで、通常の型宣言を行う
    - ▹ 例 : FILE *stdin;

色々な変数の宣言とスコープ

- 変数の宣言
  - これまでは関数の中だけだった(ローカル変数)
    - 関数の中だけで有効
    - ブロック内だけで有効な変数もつくれた
- 関数の外でも変数は宣言できる(グローバル変数)
  - 複数の関数から共通の変数を利用する事ができる
    - 関数間に変数を経由した「結び付き」ができる
  - グローバル変数の得失
    - 「結び付き」が「便利さを生む」のは事実(便利なので)
    - 常に意識する必要がある(腐れ縁になってしまう可能性が..)
  - グローバル変数の利用はできるだけさける
    - 情報は、引数と返り値でやり取りする
    - ただし、効率は悪くなる(コピーがおきるので..)
    - 効率を高める方法はあるが..(後に、ポインターの話をする)

make と Makefile

- 分割コンパイルの問題点
  - 分割コンパイルは面倒
    - 複数の *.c ファイルのコンパイル / リンク作業
  - 変更の内容によって、作業内容も異なる
    - 面倒だから、全部作り直せばよいのだが.. (時間が掛るかも..)
- make と Makefile
  - make は上記の分割コンパイルの問題点を回避するツール
    - 分割コンパイルの作業を自動的におこなってくれる
  - Makefile
    - make が分割コンパイルの作業を行うために参照する作業情報ファイル

Makefile

- Makefile の構造
  - マクロ宣言 (マクロ名 = マクロの値)
  - 作成規則 (目的、材料、作成方法 の三組を記載する)
    - 作成規則の記述形式:

```
-- 8< ---- 8< ---- 8< ---- 8< ---- 8< ---- 8< --
目的のファイル : 材料とするファイルのリスト
<タブ>作成するための命令 1
<タブ>作成するための命令 2
```

- ..

```
<タブ>作成するための命令 3
-- 8< ---- 8< ---- 8< ---- 8< ---- 8< ---- 8< --
```

    - 事例 : main.o と func.o をリンクして program.exe を作る

```
-- 8< ---- 8< ---- 8< ---- 8< ---- 8< ---- 8< --
program.exe : main.o func.o
    cc -o program.exe main.o func.o
-- 8< ---- 8< ---- 8< ---- 8< ---- 8< ---- 8< --
```

make と Makefile の利用法

- make と Makefile の利用法
  - 基本: make のみ
    - 現在のディレクトリの Makefile という名前のファイル利用
    - 作成されるのは、Makefile の中の最初のターゲット
  - Makefile の指定: make -f foobar.mk
    - 現在のディレクトリの foobar.mk という名前のファイル利用
  - 作業の指定: make abc
    - abc をターゲットして作成