

# ソフトウェア概論 A/B

-- 分割コンパイルと makefile --

数学科 栗野 俊一 / 渡辺 俊一 ( TA: 栗原 望 / 小嶋 仁子 [M2] )

2018/04/27 ソフトウェア概

# 伝言

---

## 私語は慎むように !!

- 出席パスワード : 20180427
- 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>
- 廊下側の一行は遅刻者専用です(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

# 前回(2018/04/20)の復習

---

## □ 前回(2018/04/20)の内容

### ○ プログラムとは

- ▶ 計算機への指示(作業手順)を記述したもの

### ○ コンパイルとは

- ▶ 人間に判り易い形式(C 言語)から計算機が実行できる形(機械語)に変換する

### ○ C 言語

- ▶ printf : メッセージを出力する関数
- ▶ 順接 : 命令を並べると、その順序に実行される
- ▶ 関数 : 幾つかの命令列に名前を付けたもの

## □ 演習

### ○ Compile の仕方を覚える

### ○ プログラムを書いてみよう

- ▶ Hello, World (単純で完全なプログラム/プログラム作成の出発点)
- ▶ 関数を並べてみよう(順接) / 関数を作ってみよう(命令に名前を付ける)

# 前回 (2018/04/20) の課題

---

## □ 前回 (2018/04/20) の課題

### ○ 課題 20180420-01:

- ▶ ファイル名 : 20180420-01-QQQQ.c (QQQQ は学生番号)
- ▶ 内容 : 「Hello, 自分の名前」を 3 回出力する C 言語のプログラム

### ○ 課題 20180420-02:

- ▶ ファイル名 : 20180420-02-QQQQ.c (QQQQ は学生番号)
- ▶ 内容 : 「Hello, 自分の名前」を表示する関数を作成しなさい

### ○ 課題 20180420-03:

- ▶ ファイル名 : 20180420-03-QQQQ.c (QQQQ は学生番号)
- ▶ 内容 : 「Hello, 自分の名前」を100回以上出力する C 言語のプログラム

## □ 提出するファイル形式

- 全てテキストファイル(C 言語プログラムファイル)
- 提出先は CST Portal II

# 本日の課題 (2018/04/27)

---

## □ 今回 (2018/04/27) の課題

### ○ 課題 20180427-01:

- ▶ ファイル名 : 20180427-01-YYYY.c (YYYY は学生番号)
- ▶ 内容 : 童謡の歌詞を出力するプログラムを作成しなさい
- ▶ 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)
- ▶ 可能な限り引数付きの関数で..
- ▶ 曲は何でもよい

# make と makefile

---

## □ make とは

- 様々な操作を自動的に行ってくれるコマンド
  - ▶ 「ファイルを『作成』する」ので「make (作る)」という名前

## □ Makefile

- 様々な「ファイルの作成方法」を記述したファイル
  - ▶ make は Makefile を読み込んで、ファイルを作成する
- cf. 料理で考えると、Makefile がレシピで、make が料理人、ファイルが料理

## □ Makefile の記述方法

- 基本の形式は次の形
  - <<作りたいファイル>> : <<材料となるファイル>> ...
  - <<TAB コード>><<作りために実行する命令>>

- 例: (hello.exe を作る Makefile)

```
hello.o : hello.c # hello.c を材料に hello.o を作る
    cc -c hello.c # hello.o を作るには「cc -c hello.c」とする
hello.exe : hello.o # hello.o を材料に hello.exe を作る
    cc -o hello.exe hello.o
```

# Makefile の色々

---

## □ Makefile には色々な便利な機能がある

### ○ 変数 : 文字列に名前を付ける事ができる

▶ 変数(の値の)定義 : 「変数名」=「値」とすると変数に値の内容を割り当てる

▶ 変数(の値の)参照 : 「\$(変数名)」と書くと「変数の

### ○ 例: (hello.exe を作る Makefile)

```
BASE=hello      # 変数 BASE に hello という値を割り当てる
```

```
${BASE}.o : ${BASE}.c # ${BASE} はどれも hello に置き換わる
```

```
    cc -c ${BASE}.c # 結果的に「cc -c hello.c」と同じ
```

```
${BASE}.exe : ${BASE}.o
```

```
    cc -o ${BASE}.exe ${BASE}.o
```

# 関数 (再録)

---

## □ 関数

- 「命令列」に「名前」を付けたモノ
  - ▶ 名前を指定して「呼び出す」だけで、その命令列が実行できる

## □ 関数定義：新しい関数を定義し、プログラム内で利用出来るようにする

- 「命令列」を「{」と「}」で囲って、それに「名前」を付ける
  - ▶ この「命令列」を関数の「本体」と呼び、「名前」を「関数名」と呼ぶ
  - ▶ 「void」とか「()」は、今回は説明しない

## □ 関数呼び出し

- 関数名を指定する事により、関数の本体の命令列が実行できる
  - ▶ 「()」は今回は説明しない

## □ 関数の効用

- 「名前」が付くのでプログラムが理解り易くなる
  - ▶ 「命令列」に対し、「意味のある名前」を付ければ、意味が一目で解る
  - ▶ 同じ名前の関数呼び出しは、同じ命令の実行である事が保証される
- 関数を利用すると、プログラムが短くなる



# 関数の作り方 (その 1)

---

## □ 関数の作り方(引数がない場合)

- 関数の「名前」を決める

- ▷ cf. subfunc

- どの部分の「命令列」を関数にするかを決める

- 関数にしたい「命令列」を切出し、**main** の外に出し、「ブロック」にする

- ▷ 「命令列」を「ブロック」にするには '}' と '}' で囲めば良い

- ▷ 名前を付ける ( cf. void subfunc() )

- 元々の命令が在った所に「関数呼び出し」を置く

- ▷ cf. subfunc();

# 関数呼び出しの挙動

---

- 関数呼び出しは、次の様に振舞う
  - 関数呼び出しのある場所から関数本体の先頭に行く
  - 関数の本体を実行する
  - 関数呼び出しのあった場所の次に戻る
- 関数の「引数(ひきすう)」とは
  - 関数の振舞いを変更するための「情報 (パラメータ)」
    - ▶ 同じ関数でも「引数」が異れば異なる「振舞い」をする
- 引数付きの関数の呼び出し
  - 関数の中の「仮引数変数」に、「実引数の値」が入っている

# 分割コンパイル

---

## □ C 言語で記述されたプログラムの構造

### ○ main 関数が必ず必要

▶ 他の関数は main 関数から呼び出される

### ○ 関数の定義

▶ ソースファイル (\*.c) の中に記述して、コンパイルする

▶ 同じファイル内である必要はない

## □ 分割コンパイル

### ○ 関数を別のファイルで定義し、個々にコンパイルする事

▶ 後でリンクにより一つの実行ファイルにまとめる

# make と分割コンパイル

---

- 分割コンパイルは複数のファイル进行处理
  - 作業も面倒だし、間違いも起きやすい
    - ▷ コンパイルの手順を記述してコンピュータにやらせちゃおう
- **Makefile**
  - コンパイルの手順などを記述したファイル
- **make**
  - **Makefile** を読んで、コンパイルを自動的に行ってくれる

# 関数の作り方 (その 2)

---

## □ 関数の作り方(引数のある場合)

- ほとんど、同じ部分を探す
- 異なる部分には名前を付ける (関数の引数)
  - ▶ 異なる部分を変数に置き換える
- 変数に置き換えた後の命令列を関数の本体にする
- 関数呼び出しでは、引数に対応する値を指定する
  - ▶ 関数の実行時は、関数本体内の変数が、実引数の値に置き換わって動く

# プログラミング：積み木の積み上げ

---

## □ プログラムを作成する行為は..

### ○ 沢山の部品を組み上げて、製品を完成させる行為

▶ ライブラリは、「部品」に相当

▶ 「大きな部品」があれば、「ちょっと、飾るだけで完成」する

▶ cf. カレーのレトルトパックがあれば、御飯を炊くだけでカレーライス完成

### ○ 「大きな部品」は高機能

▶ 扱いが難しい (API をよく理解する必要がある)

## □ Top-Down と Bottom-Up

### ○ Top-Down : 上から下に向けて進める

▶ 目的を分割して分かり易くする (設計)

### ○ Bottom-Up : 下から上に向けて進める

▶ 機能を組み上げて、目的を実現してゆく (コーディング)

### ○ Top-Down に考えて、Bottom-Up に作成する