

# ソフトウェア概論 A/B

-- ハノイの塔 --

数学科 栗野 俊一 / 渡辺 俊一 (TA: 石川 溪 / 山脇 直樹 [M1])

2019/05/31 ソフトウェア概

# 伝言

---

## 私語は慎むように !!

- 出席パスワード : 20190531
- 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>
- VNC Server Address : 10.9.154.227
  - Password : vnc-2018
- 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

# 前回(2019/05/24)の復習

---

## □ 前回(2019/05/24)の内容

- 「繰り返し」の復習：再帰呼出しする関数の定義は「繰り返し」を表現
  - ▶ 命令の組み合わせ：順接/条件分岐/繰り返し、これがあれば「万能」になる
- Turtle Graphics (亀プログラミング)：画面上の「亀」を制御して、図を書く
  - ▶ Turtle Graphics のライブラリの利用と API の体験
  - ▶ 「巨人の肩に載る」：既存の成果を上手に再利用する
- 「文字」の扱い：C 言語では「文字列」以外に「文字」が扱える
  - ▶ 文字のリテラル(文字自身の表現)：' '+「一文字」+'」
  - ▶ 例：'A' という表現は、「A」という「文字」を表す
  - ▶ 文字の入出力：getchar/putchar 関数で、文字が入出ができる

# お知らせ

---

- 本日(2019/05/31)の予定
  - Miku を動かす
  - ハノイの塔

# 先週 (2019/05/24) の課題

---

## □ 先週 (2019/05/24) の課題

### ○ 課題 20190524-01:

- ▶ ファイル名 : 20190531-01-QQQQ.c (QQQQ は学生番号)
- ▶ 内容 : 与えられた文字列の文字を二度ずつ出力する関数を作成する

### ○ 課題 20190524-02:

- ▶ ファイル名 : 20190531-02-QQQQ.c (QQQQ は学生番号)
- ▶ 内容 : 与えられた文字列を逆順に出力する関数を作成する

## □ 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

# 今週 (2019/05/31) の課題

---

## □ 今週 (2019/05/31) の課題

### ○ 課題 20190531-01:

▶ ファイル名 : 20190531-01-QQQQ.c (QQQQ は学生番号)

▶ 内容 : 高さ 3 のハノイの塔を解く

▶ 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)

## □ 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

# ファイルの入手とインストール(miku)

---

## □ ファイルのダウンロード

- 本日のフォルダ(c:\usr\c\20190531)を作成

- 次の本日 (2019/05/31) のページからファイルをダウンロードする

  - <http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2019/soft/20190531/20190531.html>

- ダウンロードするファイル

  - ▷ miku.zip (c:\usr\c\20190531 に保存し、展開する)

## □ ファイルをダウンロードしたら次の作業を行う

- ubuntu 側で ~/c/20190531/miku に移動する

  - ▷ cd ~/c/20190531/miku

- 「make test」で実行して、ミクが動けば良い

# ファイルの入手とインストール(hanoi)

---

## □ ファイルのダウンロード

- 本日のフォルダ(c:\usr\c\20190531)を作成

- 次の本日 (2019/05/31) のページからファイルをダウンロードする

  - <http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2019/soft/20190531/20190531.html>

- ダウンロードするファイル

  - ▷ hanoi.zip (c:\usr\c\20190531 に保存し、展開する)

## □ ファイルをダウンロードしたら次の作業を行う

- ubuntu 側で ~/c/20190531/hanoi に移動する

  - ▷ cd ~/c/20190531/hanoi

- 「make test」で実行して、ハノイの塔が表示されれば動けば良い

# Hanoi の塔

---

## □ Hanoi の塔の問題

- 三本の棒 ( 1 ~ 3 ) と、大きさの異なる  $N$  枚の円盤がある
- 最初は、その内の一番左の 1 番の棒に、大きさの順に円盤が積まれている
- その円盤を、全て、真ん中の 2 番の棒に移動したい
- ただし、円盤を移動する場合には、次の制限を守る必要がある
  - ▶ 一度に移動できるのは、棒の一番上の一枚の円盤のみである
  - ▶ 空の棒にはどの大きさの円盤も積む事ができる
  - ▶ 既に円盤が積まれた棒には、その一番上の円盤より小さい円盤しか積む事ができない

# Hanoi プログラミング

---

## □ Hanoi プログラミング

- 必要なヘッダー `s_hanoi.h` (先頭で `include` する)

## □ Hanoi API

- `s_hanoi_init ()` : ハノイの塔の初期化

- ▶ `main` の先頭で必ず一回だけ呼び出す必要がある

- `s_hanoi_set ( char *discs )` : ハノイの塔の高さを設定する

- ▶ 高さは、指定した文字列 `discs` の長さになる。

- ▶ 指定しない場合は、3 になる : `s_hanoi_set ( "****" )` と同じ

- `s_hanoi_move ( char *from, char *to )` : `from` にある円盤を `to` に移す

- ▶ `from`, `to` は共に文字列で、"1", "2", "3" の何れかを指定する

- `s_hanoi_clear ()` : 最初の状態に戻す

- ▶ 円盤は、全て、左 (つまり "1") の棒に積まれる

- `s_hanoi_stop` : ハノイプログラムの終了 : `return 0` の前に呼び出す

## □ プログラムの実行

- `make BASE=foobar` で、`foobar.c` から `foobar.exe` が作られる

# Hanoi の塔の問題の解き方 (1)

---

□ 兎に角、小さい数値でやってみる

○ 高さ 3 の場合、次の 7 つの操作で移動できる

- ▷ a) 棒 1 から、円盤 (1) を、棒 3 に移動
- ▷ b) 棒 1 から、円盤 (2) を、棒 2 に移動
- ▷ c) 棒 2 から、円盤 (1) を、棒 3 に移動
- ▷ d) 棒 1 から、円盤 (3) を、棒 2 に移動
- ▷ e) 棒 3 から、円盤 (1) を、棒 1 に移動
- ▷ f) 棒 3 から、円盤 (2) を、棒 2 に移動
- ▷ g) 棒 1 から、円盤 (1) を、棒 2 に移動

# Hanoi の塔の問題の解き方 (2)

---

## □ 分析 : $N=3$ の場合

○ 高さ 3 の塔を、棒 1 から、棒 2 に移動するには...

▷ a) ~ c) : 高さ 2 の塔を 1 から 3 に動かしている (棚上げ)

▷ d) : 円盤 (3) を 1 から 2 に移動している (目的の円盤の移動)

▷ e) ~ g) : 高さ 2 の塔を 3 から 2 に動かしている (棚下し)

○ ポイント (一般化)

▷ 高さ  $N$  の塔を動かすには、円盤 ( $N$ ) を移動する必要がある

▷ 円盤 ( $N$ ) を動かすには、上の  $N-1$  の塔を退かす必要がある(棚上げ)

▷ 円盤 ( $N$ ) を動かした後は、上に  $N-1$  の塔を載せる必要がある(棚下し)

## □ 再起を利用した、一般の $N$ の解法

○ 高さ  $N$  の塔を、棒  $i$  から、棒  $j$  に、棒  $k$  を棚にして移動するには

▷ 高さ  $N-1$  の塔を、棒  $i$  から棒  $k$  に、棒  $j$  を棚にして移動 (再起)

▷ 大きさ  $N$  の円盤を棒  $i$  から棒  $j$  に移動 (単体作業)

▷ 高さ  $N-1$  の塔を、棒  $k$  から棒  $j$  に、棒  $i$  を棚にして移動 (再起)

○ 高さが 0 の塔は(円盤がないのだから)何もしなくてよい

▷ 再起の終了

# 砂漠の旅行の問題

---

## □ 砂漠の旅行の問題

- 現在地点 **S** にいる旅人が、幅 **N** の砂漠を渡って、地点 **G** に行きたい
  - ▷ **S 1 2 3 ... N G**
- 1 升移動する場合は、食料を 1 単位消費する
  - ▷ 砂漠の途中で保持する食料が 0 になったら移動ができなくなる
  - ▷ **G** に達した時点で、食料が 0 になっていてもよい
- 旅人が持つ事ができる食料は最大で 3 つ迄である
- **S** には、食料が無限にあり、いくつでも補給できる
  - ▷ 最初の状態では、砂漠には、食料はない
- 旅人は、食料を持っていれば、砂漠に食料を置く事ができる
  - ▷ 砂漠に置く事ができる食料の個数は最大 2 個である
  - ▷ 旅人は、砂漠に食料があり、保持数が 3 以内なら食料を持つ事ができる

# 数学的帰納法と再起呼び出し

---

## □ 数学的帰納法

- 「P(1)」と、「P(k-1) → P(k)」を利用して、「P(N)」を示す

## □ 再起呼び出し

- `P(N) { if ( N==1 ) { P(1) } else { P(k-1)を利用 } }`

▶ 帰納法の流れをそのまま、プログラムに表現する手段

## □ 事例：帰納法の証明を利用して、再起のプログラムを作る

- 問題：サイズ N のピラミッド作りたい ( P(N) )

- 数学的帰納法による証明

▶ サイズ 0 は、何もしなくてよい

▶ サイズ K のピラミッド作りたい(P(K))なら、P(K-1) の後 K を描けば良い

- 再起呼び出しによる実装 ( `triangle ( char *N )` )

```
void triangle ( char *N ) {  
    if ( !strcmp ( N, "" ) ) { /* サイズ 0 の時 */  
        /* 何もしない */  
    } else { /* サイズが一般の N の時 */  
        triangle ( N + 1 ); /* 文字列は 1 を加えると短くなる */  
        /* 再起呼び出し */  
        printf ( N );    /* N を表示 */  
        printf ( "\n" ); /* 改行 */  
    }  
}
```