2 実習案内 Cプログラミング環境

2–1 実習の進め方

- 基本的には、実習は授業日の2限に行う。

- 1限講義に合わせた内容で、2週間に1つの割合でプログラム作成のレポート課題が課される。

$$\left(\begin{array}{ll} \text{レポート提出の形式} & \Longrightarrow \ 2.4\text{節} \\ \text{レポートの提出先} & \Longrightarrow \ \text{直接担当者へ} \end{array}\right)$$

- 授業日の2限だけでは要求されたプログラムとそれに関連したレポートが完成しないことが予想されるので、他の時間にも自分で十分に実習して下さい。

- **Web上(**

  `http://www.ce.ie.niigata-u.ac.jp/~motoki/ProgrammingAI.html`

)にそれぞれの実習日に関連した連絡事項を配置していますので、実習開始直後にその日の連絡事項を読んで下さい。

注意：

この連絡事項の中には、

◇ 授業に関する連絡(e.g.補講,教室の変更,...)や

◇ レポートプログラム作成上のヒント、注意事項、

◇ コンピュータ利用、LaTeXに関する注意事項、

◇ レポート再提出の指示、

等が含まれるので、必ず読んで下さい。

2–2 Cプログラムの作成と実行

基礎知識：

- Cプログラムのソースファイルの拡張子は `.c` です。
- Cコンパイラを起動するコマンドは `cc` または`gcc` です。
- 何も指定しないと、コンパイル結果は `a.out` というファイルに生成されます。

Cプログラム作成・実行の手順：

例えば、次の様にします。

(1) Cプログラミング用のディレクトリを用意し、そこに移る。

(2) 仮想端末(例えばgnome-terminal)ウィンドウ上で

emacs ファイル名 &

とコマンド入力する。

(但し、Cプログラムのファイル名は .c で終わる様にする。)

(3) Emacsウィンドウと仮想端末ウィンドウの大きさ／位置を調整して画面を見易くする。

(4) Emacsエディタの下でプログラムを作成(または修正)・保存する。

(Emacsエディタはまだ終了しない。)

—

(5) 仮想端末ウィンドウをアクティブ状態にする。

(6) gcc ソースファイル名 または
cc ソースファイル名
とコマンド入力してプログラムをコンパイルする。

(a.out に実行ファイルが出来る。 cc コマンド／C コンパイラについては **2.5** 節を参照。)

(7) コンパイラからのエラーメッセージがある間は次の①～ ③を繰り返し行う。

① エラーメッセージを手掛かりにソースプログラム内の文法的な誤りを見つけ、**Emacs** ウィンドウ上で修正する。

② **Ctrl-x Ctrl-s** とキーを押すことによってソースファイルを更新する。

③ 再度プログラムをコンパイルする。

—

(８) コンパイルが無事(i.e. エラー無しで)終了したら、./a.out とコマンド入力してプログラムを実行する。

(プログラムが終了しない場合は
Ctrl-c キーを押して強制終了させる。)

(９) 実行結果に満足できない場合は、次の①～③を順に行った後にステップ(７)に戻る。

① プログラムの誤り箇所を突き止め、Emacsウィンドウ上で修正する。[誤り箇所が分からない場合は、デバッガ(第6節)を用いてプログラムの実行追跡を行ったりする。]

② Ctrl-x Ctrl-s とキーを押すことによってソースファイルを更新する。

③ 再度プログラムをコンパイルする。

(10) プログラムが正しく動作することが確認されたら、おしまい。

—

例2. 1 (x/y の小数点以下切り上げ) プログラム表示／コンパイル／実行の様子を次に示す。(下線部はキーボードからの入力を表す。)

```
motoki@ap1:~/C-java$  cat lab-ex01-ceiling.c   Enter
#include <stdio.h>

int main(void)
{
  int x, y, sum, ceiling;

  scanf("%d %d", &x, &y);
  sum = x+y;
  ceiling = x/y + (x%y+y-1)/y;  /*x/yの小数点以下切り上げ*/
  printf("%d+%d=%d\n", x, y, sum);
  printf("ceiling(%d/%d)=%d\n", x, y, ceiling);
  return 0;
}
motoki@ap1:~/C-java$  gcc lab-ex01-ceiling.c   Enter
```

```
motoki@ap1:~/C-java$  ./a.out  Enter
8 3  Enter
8+3=11
ceiling(8/3)=3
motoki@ap1:~/C-java$
```

プログラム作成の際に有用なキー／コマンドを次に紹介しておきます。

Emacsの下でのCプログラム作成：

Emacsの下でCプログラムを作成する時(i.e. □.c という名前のファイルを編集する時)は、次のようなキーを使うと字下げが楽に出来ます。

Ctrl-j ··· 改行してから次の行の字下げを行う。
(newline-and-indent)

TAB ··· カーソルのある行の字下げ(調節)を行う。
(c-indent-line)

Esc **Ctrl-** ··· 指定されたリージョンの字下げ(調節)を行う。
リージョン指定は予め行っておく。
(indent-region)

Cプログラムの整形：

Cプログラムの整形・表示に次のUNIX コマンドが役立つことがありますが、実習室の計算機にはインストールされていない様です。

形式:	**indent** [options] [*file-name*]	
options:	-kr	**KernighanとRitchie** のスタイルで整形
	-gnu	**GNU**プロジェクトのスタイルで整形
	-origs	**BSD**の**indent**互換スタイルで整形
	-i*num*	字下げ幅を *num* として整形

Cプログラムに行番号を付ける

gdbデバッガを使う際の様に文書ファイルに行番号を付けて表示したい時、次のUNIXコマンドが役に立ちます。 (gdbデバッガの参考のために使う場合は `-b a` というオプションを付けて実行する。
`cat`コマンドを `-n` オプション付きで実行してもよい。)

形式: **nl** [options] [*file-name*]	
options:	`-b` *type* どの行に番号を付けるかを指定します。 指定できる*type*とその意味は次の通り 。 `a` ··· 全行に番号付け。 `t` ··· 印書可能なテキストのある行だけに番号付け。(デフォルト) `n` ··· 番号づけをしません。 `p` *exp* ··· *exp*という正規表現パターンを含む行だけに番号付け。
その他	オンラインマニュアルを御覧下さい。(`man nl`)

2–3 プログラミング時の注意

- 一度に大勢で同じ課題に取り組むためプリンタの出力が誰のだか分かりにくくなります。そこで、作成する各プログラムの1~3行目は次の様な注釈行にして下さい。

```
/* 科 目 名(○曜○限)                    */
/* ○○学部○○○プログラム                */
/* 各自の学籍番号   各自の氏名           */
```

- 各自のホームディレクトリの下にディレクトリを作り、その中でCプログラム作成等の作業を行って下さい。 その際、そのディレクトリの保護モードを `rwx---r--` に設定して下さい。 例えば、

```
motoki@ap1:~$   mkdir c-lab Enter
motoki@ap1:~$   chmod g-rwx c-lab Enter
```

- キーボードからのデータ入力を終了させる (**i.e.** 入力ファイルを閉じる) ためには、**Ctrl-d** とキーを押します。

  [補足：**UNIX** ではプログラムの停止, 入力の終り, 表示の一時中断, ... といった特殊機能が幾つかのキーに割り当てられており、それをユーザが設定することも出来ます。どの特殊機能がどのキーに割り当てられているかを調べるには `stty -a` とコマンド入力します。]

2–4 レポート提出の形式

レポートとして提出するもの：

各課題について次の3つのものを提出する。

(1) プログラム，実行の様子を含む文書を印刷したもの：

まず、プログラムを (cat または nl コマンドで) 表示，コンパイル，実行している会話の様子をテキストファイルとして記録したもの (ログファイル) を作る。

そして、基本的には、これを基に
①科目名と課題番号，
②提出者の在籍番号と氏名，
③ログファイルの内容，
④プログラムの説明等 (例えばアルゴリズム設計の考え方，
変数の使い方 など；C 文法の説明は不要)，
⑤考察・その他
を順に並べた文書を (できれば LaTeX で) 構成して印刷したものを提出する。

但し、
科目名，課題番号，在籍番号，氏名だけを並べた表紙は不要。

仮想端末上の会話の様子をファイルとして記録するには：

(方法1) 会話の様子をコピー・アンド・ペーストでEmacsに取り込む。
簡単だが会話が長い場合は間違う可能性がある。

(方法2) emacsの中でshellを開き、その会話の様子をファイルに取り込む。 (emacsの中でshellを開くには、Esc x shell とする。)

(方法3) `script` コマンドを用いる。 (⟶ 下で説明します。)

(2) プログラムの(ソース)ファイル：

$\longrightarrow$ 学務情報システムのレポート機能を使う。

(2.1) 提出ファイルの名前に各々の在籍番号等を含ませておく。

課題1, 2, 3の場合は、それぞれ

各自の在籍番号,小文字_rep1.c , ・・・・・・・・・・・・・・・・

という名前のソースファイルを作る。

課題4の場合は、

関連するファイル群をそれぞれ

各自の在籍番号,小文字_Rep4

というディレクトリに入れ、

このディレクトリ以下を**tar**コマンドで1つにまとめ

gzipコマンドで圧縮して

各自の在籍番号_Rep4.tar.gz

という名前のファイルを作る。

(2.2) 統合型学務情報システムにログインする。

(2.3) ログイン画面上部 の
メニューバー内の「レポート・小テスト・アンケート」ボタン、
サブメニュー内の「レポート・小テスト・アンケート提出」ボタン
を順に押す。

(2.4) 現れた一覧表の中から 、
科目名が「プログラミング AI」で
タイトル欄が当該課題番号 (e.g.「実習課題 1」)
の行を選び、右側の提出ボタンを押す。

(2.5) 現れたレポート提出画面上で 、
「ファイル添付」欄下の「参照」ボタンを押し
(2.1) の該当ファイルを指定する。

(３) 印刷物の電子ファイル(pdfファイル)：

学務情報システムのレポート機能を通じて、
(１)の印刷物のpdfファイルを
(２)のソースファイル等と一緒に提出(添付)する。

但し、
課題1~4で添付するpdfファイルの名前は、それぞれ

各自の在籍番号,小文字_rep1.pdf

~ 各自の在籍番号,小文字_rep4.pdf

としておく。

scriptコマンドについて：

画面に表示される会話の様子をそのままログ(log, 日誌)ファイルに記録するためにscriptコマンドが用意されています。scriptコマンドの使い方は次の通りです。

① 仮想端末ウィンドウで

script ログファイル名

と入力して記録を開始する。

② 必要な会話

③ exit と入力して記録を終了する。

④ エディタを使ってログファイルを編集する。

短い会話だと会話の様子をコピー・アンド・ペーストでEmacsに取り込む方が簡単ですが、
長い会話の場合にはscriptコマンドは有用です。

(4) ログファイル編集の作業：

まず^Mを削除します。**Emacs**の場合は、**Esc x replace-string** というコマンドを用いて次の様に...

① カーソルをファイルの先頭にもって来る。

② Esc x replace-string Enter とコマンド入力して文字列の検索置換処理を開始する。

③ 画面下に Replace string: と表示されるが、これに対しては Ctrl-q Ctrl-m Enter と返答する。

④ 画面下に Replace string ^M with: と表示されるが、これに対しては Enter とだけ返答する。

同様に、コマンド行左端に含まれる

”^[]0” ～ ”^G” の部分のコードも全て削除。

次にミスタイプ部も削除する。

注意：

scriptコマンドで記録されるログファイルには、仮想端末上に表示された全ての文字が記録されます。ですから、Ctrl-p や ↑ キーを押して入力コマンドを再利用した場合は途中に現れるコマンドも全て記録され、そのまま印刷するとそれらのコマンドが重なって印刷されます。

⇨ scriptコマンドで会話の記録をとっている場合は、
Ctrl-p や ↑ キーで入力コマンドの再利用を行うのは
避けた方が無難です。

例2. 2 (scriptコマンド；x/y の小数点以下切り上げ) 例2.1のプログラムがレポート課題であった場合は、 実習室で 例えば次の様にします。

```
motoki@ap1:~/C-java$  script report.log  Enter
   ........................ログファイル report.log への記録を開始
Script started, file is report.log
motoki@ap1:~/C-java$  cat ex01-lab.c  Enter
```

```
#include <stdio.h>

int main(void)
{
  int x, y, sum, ceiling;

  scanf("%d %d", &x, &y);
  sum = x+y;
  ceiling = x/y + (x%y + y - 1)/y;     /* x/yの小数点以下切り...
  printf("%d+%d=%d\n", x, y, sum);
```

```
  printf("ceiling(%d/%d)=%d\n", x, y, ceiling);
  return 0;
}
```

```
motoki@ap1:~/C-java$  gcc ex01-lab.c  Enter
motoki@ap1:~/C-java$  ./a.out  Enter
 8 3  Enter
8+3=11
ceiling(8/3)=3
motoki@ap1:~/C-java$  ./a.out  Enter
 999 3  Enter
999+3=1002
ceiling(999/3)=333
motoki@ap1:~/C-java$  exit  Enter
  ..................................ログファイルへの記録を終了
exit
Script done, file is report.log
motoki@ap1:~/C-java$
```

すると、次の様なログファイルが出来ますから、これから

各コマンド行左端の "^[]0"～"^G" の部分と

各行最後の "^M"

を削除することになります。

```
Script started on 2016年03月07日 17時11分33秒
^[]0;motoki@ap1: ~/C-java^Gmotoki@ap1:~/C-java$
                                        cat lab-ex01-ceiling.c^M
#include <stdio.h>^M
^M
int main(void)^M
{^M
  int x, y, sum, ceiling;^M
^M
  scanf("%d %d", &x, &y);^M
  sum = x+y;^M
  ceiling = x/y + (x%y + y - 1)/y;     /* x/y の小数点... */^M
  printf("%d+%d=%d\n", x, y, sum);^M
  printf("ceiling(%d/%d)=%d\n", x, y, ceiling);^M
  return 0;^M
}^M
^[]0;motoki@ap1: ~/C-java^Gmotoki@ap1:~/C-java$
                                        gcc lab-ex01-ceiling.c^M
^[]0;motoki@ap1: ~/C-java^Gmotoki@ap1:~/C-java$ ./a.out^M
8 3^M
8+3=11^M
ceiling(8/3)=3^M
^[]0;motoki@ap1: ~/C-java^Gmotoki@ap1:~/C-java$ ./a.out^M
999 3^M
```

```
999+3=1002^M
ceiling(999/3)=333^M
^[]0;motoki@ap1: ~/C-java^Gmotoki@ap1:~/C-java$ exit^M
exit^M
 Script done on 2016年03月07日 17時13分03秒
```

2–5 Cコンパイラについて

- 実習室の計算機ではC言語のコンパイラが1種類だけ用意されています。コンパイラを起動するためのコマンドとしてとして gcc と cc の2つが用意されていますが、これらの実体は同じです。

  (他に c99-gcc 等もありますが、
  これらは特殊なオプション付きの gcc の様です。)

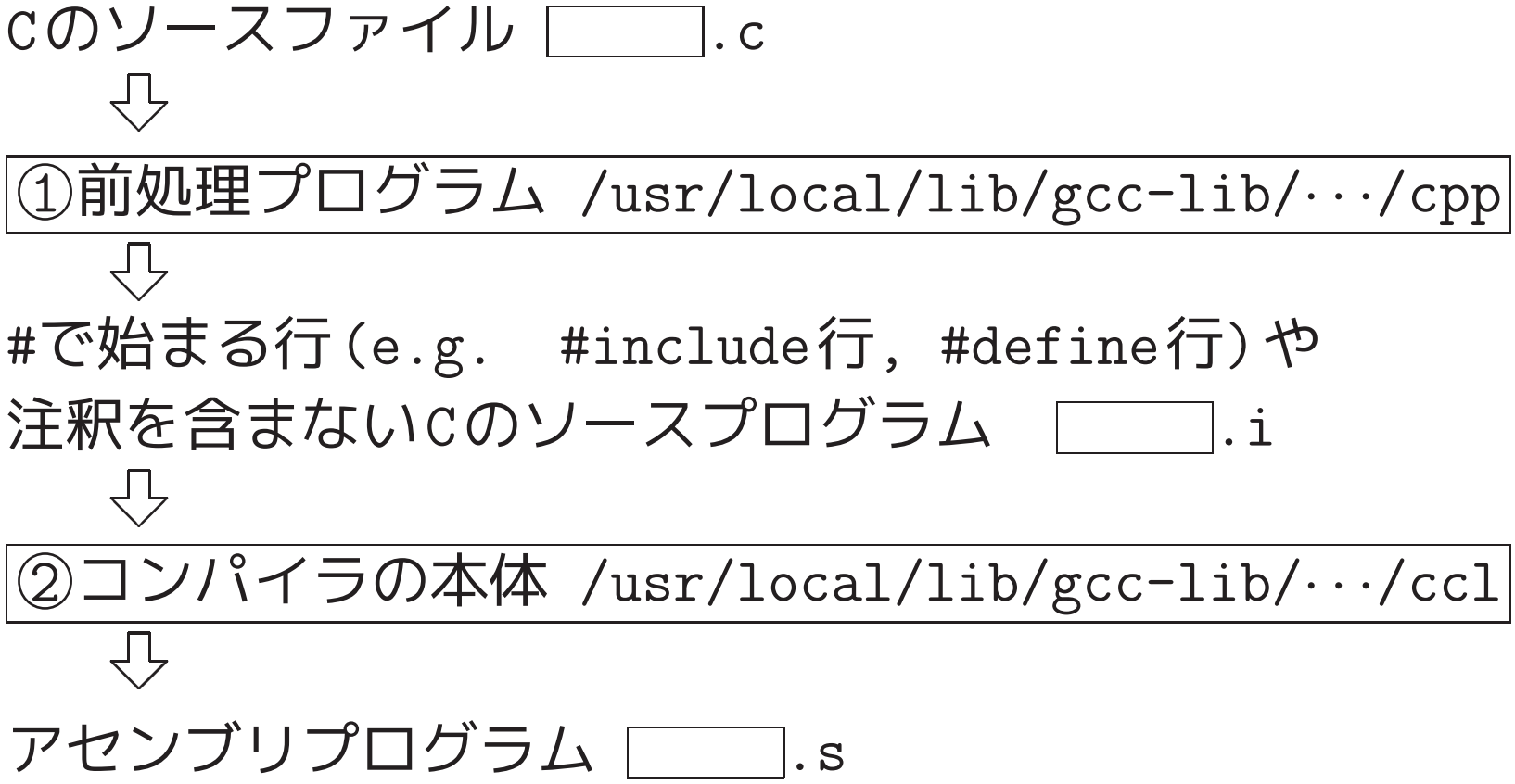
```
motoki@ap1:~$ which gcc
/usr/bin/gcc
```

```
motoki@ap1:~$ ls -l /usr/bin |grep cc
        (省略)
-rwxr-xr-x 1 root  root      428  6月 13  2013 c89-gcc
-rwxr-xr-x 1 root  root      454  6月 13  2013 c99-gcc
lrwxrwxrwx 1 root  root       20  2月 23  2015 cc ->
                                    /etc/alternatives/cc
        (省略)
lrwxrwxrwx 1 root  root        7  2月 25  2015 gcc ->
                                          gcc-4.9
-rwxr-xr-x 1 root  root   353752 11月 28  2012 gcc-4.6
-rwxr-xr-x 1 root  root   832120 12月 26  2014 gcc-4.9
        (省略)
motoki@ap1:~$ ls -l /etc/alternatives/cc
lrwxrwxrwx 1 root root 12  2月 23  2015
                    /etc/alternatives/cc -> /usr/bin/gcc
motoki@ap1:~$
```

- 以下は、cコンパイラについての一般的なお話です。cプログラミングの際の参考にして下さい。[cc と gcc に共通した話です。]

---

cc コマンド / gcc コマンドによる C プログラムの翻訳作業は、次のように行われる。



⇩

アセンブリプログラム □.s

⇩

③最適化プログラム（/usr/･･･??）

⇩

アセンブリプログラム □.s

⇩

④アセンブラ /usr/ccs/bin/as

⇩

オブジェクトプログラム（のファイル） □.o

⇩

⑤リンカ & ローダ /usr/ccs/bin/ld

⇩

実行形式プログラムのファイル

これら5段階の動作を制御するために、ccコマンド／gccコマンドには様々なオプションが用意されています。 例えば gccコマンドについては次の通り。

形式:	gcc [*options*] *files* [-l *library*]		
options:	-c	:	オブジェクトファイルだけを生成し、リンカ&ローダを呼び出さない。
	-g	:	デバッガのための特別なシンボル表もコンパイルの際に生成し、リンカ&ローダ ld に -lg オプションを引き渡す。
	-o ファイル名	:	実行形式プログラムを入れるファイルを a.out ではなくファイル名 とする。
	-O	:	最適化を行う。
	-D 名前 = 定義 -U 名前 -I パス名 :		(説明省略)

-l *library*: -l[x] :	リンカ&ローダにこのオプションをそのまま引き渡し、オブジェクトプログラムをまとめ上げる際、必要に応じてライブラリ（アーカイブファイル）libx.a 内の関数（オブジェクトコード）を用いることを指示する。 標準ライブラリに関する -lc というオプションは cc コマンドに付加しなくても自動的にリンカ&ローダに引き渡されるが、数学ライブラリ関数はANSI標準でないため、数学関数を用いた場合は数学ライブラリ/lib/libm.a の使用をリンカ&ローダに申告するための-lm オプションを cc コマンドの最後に付加しなければならない。

⇨ 講義ノート参照

詳しくは、man gcc でオンラインマニュアルを調べるなり、次の図書を見るなりして下さい。

- 山口和紀&古瀬一隆(監), 新The UNIX SuperText下　改訂増補版, 技術評論社, 2003.

- P.S.Wang, ANSI C & UNIX 下, 共立出版, 1994.

例えば prog.c というCプログラムがある時、

```
% gcc -g prog.c -lm
```

とコマンド入力すると計算機内部では次の様な処理が行われます。

prog.c

①前処理プログラム
/usr/local/lib/gcc-lib/···/cpp
②コンパイラの本体
/usr/local/lib/gcc-lib/···/ccl
④アセンブラ
/usr/ccs/bin/as

prog.o .....次のステップ「⑤リンカ&ローダ」の後で削除される。

---

prog.o .....次のステップ「⑤リンカ&ローダ」の後で削除される。

⑤リンカ&ローダ /usr/bin/ld

/usr/ccs/lib/crt1.o prog.o -lg -lm -lc

オブジェクトプログラムの並び。左から右に見て行った時、未定義の記号は後方で必ず処理される様に並べないといけない。[ここで、/usr/ccs/lib/crt1.o は初期化を行ったり関数 main を呼び出したりする C のスタートアップルーチンである。]

a.out

2–6 lint

lintは、プログラム内に潜んでいてバグの原因になりそうな箇所、資源を無駄に使用している変数定義、他の計算機に移植する際に問題になりそうな箇所をコンパイラより厳しくチェックする為のコマンドですが、実習室の計算機にはインストールされていない様です。

例えば、

```
lint ソースファイル名
```

とか

```
lint -hx ソースファイル名
```

という風に起動します。ここで、

-hオプション はプログラムの書式をチェックすることを指定し、

-xオプション はextern 宣言されているのに使用されてない変数があるかどうかをチェックすること

を指定します。

補足：

元々 lint は古い時代の詳細なエラーチェックを行わないCコンパイラを補完するためのもの。

詳細なエラーチェックを行う現在のCコンパイラを使う場合はあまり使う必要がないかも...。

オプションについて知りたい人は、「古川芳孝、make, lint, dbxの使い方、HBJ出版局」あたりを御覧下さい。

使用例： 旧システム上で、

```
sv01_62: lint lab-ex01-ceiling.c
関数が返す値は常に無視されます
    printf                  scanf
sv01_63:
```