

情報基礎 B
第 9 回, 第 10 回
アカデミック・スキル II
C 言語プログラミング (1)

長江 剛志

(nagae@m.tohoku.ac.jp)

東北大学大学院工学研究科
技術社会システム専攻

2015 年 6 月 5 日 (金)
+ 6 月 14 日 (金) の分

目次

はじめに

6月5日(金) 第9回 ソースファイルの作成, コンパイル, 実行

6月19日(金) 相当分 第10回 変数, 標準入力, 四則演算

サンプルプログラムのダウンロード

本講義で使用するサンプルプログラム (hello.c, calc1.c, calc2.c, calc3.c) は、いずれも ISTU (<http://www.istu.jp>) からダウンロードできます。6月5日(金)の講義は盛り沢山なので、予めダウンロードしておくといでしょう。

- ▶ ISTU の右下の 受講授業科目 から、金曜の 3 時限 情報基礎 B を選択
- ▶ 科目共通教材の中から 配布資料 第 9 回, 第 10 回 C 言語サンプルプログラム を選択

目次

はじめに

6月5日(金) 第9回 ソースファイルの作成, コンパイル, 実行

6月19日(金) 相当分 第10回 変数, 標準入力, 四則演算

今日やること

C 言語プログラミング用のディレクトリの準備

C 言語ソースファイルの作成

gcc によるコンパイル

自分の学籍番号と名前を表示させる

C 言語プログラミング用のディレクトリの準備

1. Terminal を起動し, mkdir コマンドを使って

- ▶ ~/Documents/prog/
- ▶ ~/Documents/prog/01_hello

という 2 階層のディレクトリを作る. -p オプションを使うと一度に作成できる.

```
$ mkdir -p ~/Documents/prog/01_hello
```

2. cd コマンドを使って, 現在の作業ディレクトリを ~/Documents/prog/01_hello に変更する.

C 言語プログラミングのワークフロー

1. gedit など で C 言語ソースファイル (.c) を作成・編集する.
2. Terminal 上で gcc コマンドを用いて C 言語ソースファイルをコンパイルし, 実行ファイルを生成する.
3. Terminal 上で実行ファイルを走らせ, 望んだ通りに実行されなければ 1 に戻る.

初めてのC言語プログラム(hello.c)の作成

最初のステップとして、画面上に Hello world と出力するプログラムを作成してみる。

1. Terminal から gedit を起動し、
~/Documents/prog/01_hello/hello.c というテキスト・
ファイルを新たに作成する。

```
$ pwd # 現在の作業ディレクトリを確認
~/Documents/prog/01_hello
$ gedit hello.c & # &を付けバックグラウンドで起動
```

2. hello.c に以下を記載し、保存する。

```
1 #include <stdio.h> /* 標準入出力ライブラリ */
2 int main(void)      /* main 関数の引数と戻り値の定義 */
3 {                  /* main 関数の始まり */
4     printf("Hello world\n"); /* printf 関数の呼び出し */
5     return 0;       /* 戻り値として 0 を返す */
6 }                  /* main 関数の終わり */
```

- ▶ 枠の外側の 1~6 は参考のために表示させた **行番号** なので入力しない。
- ▶ /* から */ までの間は **コメント** なので入力しなくてもよい。

hello.c のコンパイルと実行

1. Terminal 上で ls コマンドを使い、現在の作業ディレクトリに hello.c があるか確認する。

```
$ ls ↵  
hello.c
```

hello.c が表示されることを確認する。

2. ソースファイルから **実行ファイル** を生成するには gcc コマンドを用いる。-o オプションを使用することで、**実行ファイル名を指定** できる (省略した場合は a.out という名前になる)。例えば、hello.c から hello という実行ファイルを作るには、Terminal から以下のように入力する。

```
$ gcc -o hello hello.c ↵
```

3. 生成された実行ファイル (./hello) を呼び出してみる。

```
$ ./hello ↵  
Hello world
```

Hello world と表示される。

hello の前の ./ は、実行ファイルが **現在のディレクトリにある** ことを明示するために必要。

hello.c の解説 (1)

▶ 1 行目

```
#include <stdio.h>          /* 標準入出力ライブラリ */
```

は, #include という **マクロ** を用いて, **標準入出力** (ディスプレイへの表示やキーボードからの入力) を行なうための **ライブラリ** stdio の **ヘッダファイル** を読み込んでいる.

実は, C 言語は, それ単体では, **画面に文字を出力することすらできない**. C 言語では, 画面/ファイルへの出力や, キーボード/ファイルからの入力, 複雑な科学技術計算, 乱数の発生といった機能を使うためには, その機能に応じた **ライブラリ** を読み込まなくてはならない.

必要なライブラリだけを読み込むことで, 実行ファイルのサイズを小さくでき, コンパイルにかかる時間を減少させられる.

hello.c の解説 (2)

- ▶ 2～6 行目は, main() という特殊な **関数** を定義している. C 言語で生成された実行ファイルは, (Terminal などから) 起動されると main() 関数を実行した後, (main() が正常に終了すれば), 自身も終了する.
- ▶ 2 行目

```
int main(void)                /* main 関数の引数と戻り値の定義 */
```

は, main() の **引数** と **戻り値** を定義している.

- ▶ main の前の int は, main() が **整数型** の **戻り値** を持つことを表している
- ▶ main の後の (void) は, main() が **引数** を持たないことを表している
- ▶ main() の **中身** は, 3 行目の { から 6 行目の } までの間に記述されている.

hello.c の解説 (3)

▶ 4 行目:

```
printf("Hello world\n");    /* printf 関数の呼び出し */
```

は, `stdio` ライブラリに収められている `printf` という関数に, `"Hello world\n"` という **文字型配列** の **定数** を与えて呼び出すことを指示している.

- ▶ `printf()` の機能 (の 1 つ) は, **引数に与えられた文字列を画面に出力** することである. ここで与えている文字列の最後の `\n` は, この場所で出力を **改行** させる記号 (**エスケープ文字**) である. よく使われるエスケープ文字としては, `\t` (タブ文字を出力), `\` (ダブルクォート), `\\` (バックスラッシュ) などがある.

▶ 5 行目:

```
return 0;                    /* 戻り値として 0 を返す */
```

は, `return` という **命令** を使って, 整数型の定数 `0` を **戻り値** として `main` 関数から抜け出すことを指示している.

セミコロン (;) について

- ▶ C 言語では、原則として、それぞれの **文** の末尾には **セミコロン (;)** をつける。セミコロンをつけ忘れると、**コンパイルエラー** が山盛り出る。
- ▶ `#include` などの **マクロ** は文ではないので ; はつかない
- ▶ `main` 関数の **定義ブロック** (`int main(void){...}`) や、後半で出てくる `if(...){...}else{...}` や `for(..;..;..){...}` ループなどの **ブロック** の末尾ではセミコロンを省略できる。

空白/改行とインデントについて (1)

- ▶ C 言語では、**評価式** 中の複数の **空白** や **改行** はいくつあっても 1 つ分として評価される。(マクロや文字列の中は評価式では無いので該当しない)。例えば, `hello.c` は

```
#include <stdio.h>
int main(void){printf("Hello world\n");}
```

と書いても

```
#include <stdio.h>
int main
(void){
    printf
    ( "Hello world\n" );
}
```

と書いても全く同じものとして評価される。

- ▶ ソースコードを **読み易く** する (i.e. コーディングを容易にし、バグを発見し、保守し易くする) ために **インデント** は必須。

空白/改行とインデントについて (2)

インデント (字下げ) 関数や `if...else` などのブロックであることが判るように、各行の先頭に **ブロックの深さ** に応じた適当な数の空白を入れること.

- ▶ `gedit` などのエディタでは **構文解析** をして **自動的に** インデントを入れてくれる.

レポート課題 III-1

レポート課題 III-1 (学籍番号と名前を表示させる)

自分の学籍番号と名前を、以下のような形式:

```
B5TB9999  
仙台 太郎
```

で出力する C 言語プログラムを作り、その **ソースファイル** と **出力結果** を提出せよ。ただし、下記を満足すること。

- ▶ 提出するファイル名は `B5TB9999_myname.c` (ソースファイル) および `B5TB9999_myname.txt` (実行結果) とせよ。
- ▶ 学籍番号と名前の後で、それぞれ **1 回ずつ改行** せよ。
- ▶ 出力結果は、Terminal に表示された結果をコピー&ペーストするか、**リダイレクト** (後述) を使うこと。

提出期限：2015 年 6 月 18 日 (木)

レポート課題 III-1 の評価基準

必須要素

守られていない場合は減点

- ▶ 提出ファイル名 は適切か
- ▶ C 言語ソースファイル と 出力結果 を提出しているか
- ▶ gcc でコンパイルでき、生成したファイルを実行できるか
- ▶ 学籍番号、氏名の後で 1 回 1 行 1 行 改行 されているか

加点要素

特に無し

レポート課題 III-1 の進め方 (1)

1. mkdir コマンドを用いて
~/Documents/report/Report-III/Report-III-1/ という
ディレクトリを作成 (-p オプションを使うと便利).
2. Home でのマウス操作や cp コマンドなどで hello.c をコ
ピーし, 1. で作ったディレクトリの下に
B5TB9999_myname.c という名前で保存 する.
3. cd コマンドを用いて, 現在の作業ディレクトリ を
~/Documents/report/Report-III/Report-III-1/ に変更
する.
4. gedit を用いて, B5TB9999_myname.c の "Hello world\n"
の部分を, 自分の学生番号と名前に 書き換える.
 - ▶ 学籍番号と名前でそれぞれ 1 回づつ printf() を呼び出して
もよい.
 - ▶ \n を複数使い, 学籍番号と名前を 1 つの文字列にまとめても
よい.

レポート課題 III-1 の進め方 (2)

5. gcc コマンドを用いて B5TB9999_myname.c を **コンパイル** し、実行ファイルを作成する。
 - ▶ -o オプションを使って実行ファイル名を myname などとしてみてもよい。
6. **実行ファイルを起動** し、学籍番号と名前が適切に表示されたかを確認する。
 - ▶ 実行ファイル名の前には、現在の作業ディレクトリを表す ./ が必要。
7. **実行結果** を B5TB9999_myname.txt として保存する。
 - ▶ **gedit を使う方法:**
gedit を起動し、Terminal 上の実行結果をコピー&ペーストし、B5TB9999_myname.txt という名前で保存する。
 - ▶ **リダイレクトを使う方法:**
Terminal 上で、コマンドの後に > hogehoge.txt とすると、コマンドを実行した結果を hogehoge.txt というファイルに出力できる。つまり、実行ファイルが ./myname という名前だった場合、

レポート課題 III-1 の進め方 (3)

```
$ ./myname > B5TB9999_myname.txt
```

とすることで、実行結果をテキストファイルに保存できる。
なお、この操作によって元の B5TB9999_myname.txt 内容は
上書き され、失われるので注意。

▶ **(応用編) 追記リダイレクト:**

既に存在するテキストファイルに実行結果を (上書きではなく) **追記** したい場合は >> を使う。例えば、

```
$ ./myname >> B5TB9999_myname.txt
```

とすると、B5TB9999_myname.txt の **末尾** に ./myname の実行結果が **追加** される。

目次

はじめに

6月5日(金) 第9回 ソースファイルの作成, コンパイル, 実行

6月19日(金) 相当分 第10回 変数, 標準入力, 四則演算

整数型変数とその出力

hello.c で printf 関数に与えた "Hello world\n" は、
文字型配列 の 定数.

数値計算など、プログラムの実行中に値を変えるには 変数 を使う必要がある. 変数を使った四則演算を行ない、その結果を表示させるプログラムをいくつか作成してみる.

今日やること

変数の宣言, 代入, 出力

四則演算

キーボードから入力された値の読み取り

入力された3桁の整数の順序を反転させた整数を出力させる

ディレクトリの準備

ディレクトリの準備

Terminal 上から `mkdir` コマンドを使って

```
~/Documents/prog/02_calc
```

というディレクトリを作る.

現在の作業ディレクトリの変更

`cd` コマンドを使って, 現在の作業ディレクトリを
`~/Documents/prog/02_calc` に変更する.

変数に値を代入し，表示させる (calc1.c)

やりたいこと

変数を宣言し，値を代入し，その値を出力する.

gedit を起動して ~/Documents/prog/02_calc/calc1.c というテキスト・ファイルを新たに作成し，以下を記載して保存する.

```
1 #include <stdio.h>          /* 標準入出力ライブラリ */
2 int main(void)              /* main 関数の引数と戻り値の定義 */
3 {                            /* main 関数の始まり */
4     int a;                  /* 整数型の変数 a を宣言 */
5     a = 3;                  /* aに3を代入 */
6     printf("aの値は%dです\n", a); /* aの値を出力 */
7     return 0;               /* 戻り値として 0 を返す */
8 }                            /* main 関数の終わり */
```


calc1.c のコンパイルと実行

1. Terminal 上で ls コマンドを使い、現在の作業ディレクトリに calc1.c があるか確認する.


```
$ ls   
calc1.c
```

calc1.c が表示されることを確認する.

2. gcc コマンドを用いてソースファイルから **実行ファイル** を生成する. -o オプションを用いて、実行ファイル名を calc1 とする.

```
$ gcc -o calc1 calc1.c 
```

3. 生成された実行ファイル (./calc1) を呼び出してみる.

```
$ ./calc1   
a の値は3です
```

a の値が表示される.

4. 第5行目の a = 3 の数値を「5」や「10」などにし、再度コンパイル・実行すると、それに合わせて出力結果が変わる.

calc1.c の解説 (1)

- ▶ 1～3 行目: hello.c と同じ
- ▶ 4 行目:

```
int a; /* 整数型の変数 a を宣言*/
```

「整数(int)型の変数として a を使う」ことを **宣言** している.
C 言語では, **あらゆる変数** が, 使う前に **宣言** されていなければならない.

- ▶ 5 行目

```
a = 3; /* aに3を代入 */
```

上で宣言した変数 a に「3」という定数を **代入** している.

- ▶ 6 行目

```
printf("aの値は%dです\n", a); /* aの値を出力 */
```

calc1.c の解説 (2)

printf (*print formatted*) 関数の **書式つき出力** 機能を使い、
a=3 を出力している。文字列中の %d のように % で始まる部分
は **書式化文字列** と呼ばれる。 %d は「与えられた引数を **整数**
と解釈して出力しなさい」という意味で、2 番目以降の引数
に与えられる値 (この場合は a の中身) に置き換えられる。

変数の使い方 (1)

宣言

全ての`変数`が、使われる前に `宣言` されていなければならない。
変数の宣言は、

```
int a;
```

のように「`変数の型 変数名;`」という形式. `同じ型` の `複数の変数` を定義する場合は、変数を `コロン (,)` で区切る:

```
int a, b;
```

変数の型

C 言語をはじめ、多くのプログラム言語では変数に `型` が存在する。「型」を決めることで、用途に応じた適切な分量のメモリの割り当てや、効率的に処理を行なえる。

本講義で明示的に用いる変数の型は `整数 (int) 型` のみである。

変数の使い方 (2)

変数名

C 言語で使える 変数名 には以下の制約がある.

- ▶ 変数名に **使える文字** は, 以下の 3 種類のみ.
 1. 半角英字 (大文字・小文字). 大文字と小文字は **区別** される.
 2. 半角数字
 3. 半角アンダースコア (_).
- ▶ **数字で始まる** 変数名 (例えば 1a) は使ってはいけない.
- ▶ **予約語** (int, double, long, short, switch, break, return, for, while, do など) は使用できない.

2つの変数の差・商・剰余を表示させる (calc2.c)

やりたいこと


変数を **2つ** 用意し, その **差** および **商・剰余** を表示させるプログラムを作る.

gedit を起動して ~/Documents/prog/02_calc/calc2.c というテキスト・ファイルを新たに作成し, 以下を記載して保存する.

```
1 #include <stdio.h>          /* 標準入出力ライブラリ */
2 int main(void)              /* main 関数の引数と戻り値の定義 */
3 {                            /* main 関数の始まり */
4     int a = 8;               /* 整数型変数 a を宣言し初期化 */
5     int b = 5;               /* 整数型変数 b を宣言し初期化 */
6
7     printf("a=%d, b=%d\n", a, b); /* aとbの値を表示 */
8     printf("差:%d\n", a-b);      /* 差を表示 */
9     printf("商:%d, 剰余%d\n", a/b, a%b); /* 商と剰余を表示 */
10
11     return 0;                /* 戻り値として 0 を返す */
12 }
```


calc2.c のコンパイルと実行

1. Terminal 上で ls コマンドを使い、現在の作業ディレクトリに calc2.c があるか確認する.


```
$ ls   
calc2.c
```

calc2.c が表示されることを確認する.

2. gcc コマンドを用いてソースファイルから **実行ファイル** を生成する. -o オプションを用いて、実行ファイル名を calc2 とする.

```
$ gcc -o calc2 calc2.c 
```

3. 生成された実行ファイル (./calc2) を呼び出してみる.

```
$ ./calc2   
a=8, b=5  
差:3  
商:1, 剰余3
```


calc2.c の解説 (1)

- ▶ 1～3 行目: calc1.c と同じ
- ▶ 4～5 行目:

```
int a = 8;          /* 整数型変数 a を宣言し初期化 */
int b = 5;          /* 整数型変数 b を宣言し初期化 */
```

整数(int)型の変数 a, b を宣言している. calc1.c との違いは, それぞれの変数を宣言すると **同時** に = を使い, 8 と 5 という値で **初期化** している. C 言語では, 変数を宣言しただけだとその値が **不定**. 初期化しておくと「値を代入する前に使ってしまう」という問題を回避できる.

- ▶ 7～9 行目:

```
printf("a=%d, b=%d\n", a, b); /* aとbの値を表示 */
printf("差:%d\n", a-b);      /* 差を表示 */
printf("商:%d, 剰余%d\n", a/b, a%b); /* 商と剰余を表示 */
```

printf 関数を使って計算結果を表示.

calc2.c の解説 (2)

- ▶ 1 番目の引数の文字列内で %d を複数個使った場合, それぞれが 2 番目以降の引数に 順に 置き換えられる. この場合, 最初の %d が a の値, 2 つめの %d が b の値に置き換えられる.
- ▶ 2 番目以降の引数に 計算式 を与えると, その計算結果で置き換えられる.
- ▶ C 言語で使われる 算術演算子 は以下の通り

演算	演算子	使用例
和	+	a + b
差	-	a - b
積	*	a * b
商	/	a / b
剰余	%	a % b

- ▶ C 言語の演算子の優先順位は算数と同じ. 乗除 (* / %) が加減 (+ -) より優先される. () を使えば優先順位を変更できる (例: 5+3/4 は 5 と評価されるが, (5+3)/4 は 2 と評価される).

calc2.c の解説 (3)

- ▶ C 言語にはべき乗を行う **演算子** は存在しないので, * を繰返し使う (一般にべき乗を計算したい場合には `<math.h>` ライブラリ中の `pow` 関数を使う)

キーボードから変数の値を入力する (calc3.c)

やりたいこと

キーボードから入力された2桁の整数の1の位と10の位を入れ替えたものを出力するプログラムを作る。

gedit を起動して ~/Documents/prog/02_calc/calc3.c というテキスト・ファイルを新たに作成し、以下を記載して保存する。

```
1 #include <stdio.h>          /* 標準入出力ライブラリ */
2 int main(void)              /* main 関数の引数と戻り値の定義 */
3 {                            /* main 関数の始まり */
4     int a;                  /* 整数型の変数 a を宣言 */
5     scanf("%d", &a);        /* キーボードからaに値を読み込む */
6
7     a = (a / 10) + (a % 10) * 10; /* 1の位と10の位を入れ替える。
8                                     ・ (a / 10) = 10の位
9                                     ・ (a % 10) = 1の位
10                                    なので、(a % 10)を10倍したものに
11                                    (a / 10)を加えれば、1の位と10の位を
12                                    入れ替えられる。 */
13     printf("%d\n", a);      /* aの2乗を表示 */
14
15     return 0;               /* 戻り値として 0 を返す */
16 }
```

calc3.c のコンパイルと実行

1. Terminal 上で ls コマンドを使い、現在の作業ディレクトリに calc3.c があるか確認する.

```
$ ls  
calc3.c
```

calc3.c が表示されることを確認する.

2. gcc コマンドを用いてソースファイルから **実行ファイル** を生成する. -o オプションを用いて、実行ファイル名を calc3 とする.

```
$ gcc -o calc3 calc3.c
```

3. 生成された実行ファイル (./calc3) を呼び出してみる.

```
$ ./calc3  
28  
82
```


入力待ち状態になるので適当な数値を入れて

calc3.c の解説 (1)

- ▶ 1～4 行目: calc1.c と同じ
- ▶ 5 行目:

```
scanf("%d", &a); /* キーボードからaに値を読み込む */
```

scanf 関数を使って、キーボードから入力された値を a に格納する。

- ▶ scanf の文字列中の %d は「入力を **整数** と解釈しなさい」という意味で、入力された値は 2 番目以降の引数 に格納される。
 - ▶ 値を収める変数の名前には、& (**アンパサンド**) を付ける必要がある (付けないと誤動作する)。
 - ▶ scanf の文字列には \n は不要 (\n をつけると  を 2 回入力しなければならない)
- ▶ 7～12 行目:

```
a = (a / 10) + (a % 10) * 10; /* 1の位と10の位を入れ替える。
```

変数 a の **1 の位** と **10 の位** を入れ替えたものを再び a に代入

calc3.c の解説 (2)

- ▶ a の 10 の位 は $a / 10$ ($a \div 10$ の商) に等しい
- ▶ a の 1 の位 は $a \% 10$ ($a \div 10$ の剰余) に等しい
- ▶ a の 1 の位 を 10 倍したもの ($(a \% 10) * 10$) と a の 10 の位 ($(a / 10)$) の和は, 「a の 1 の位 と 10 の位 を入れ替えたもの」に等しい
- ▶ 代入によって元の a の値は 上書き されることに注意
- ▶ 12 行目までは コメント.
 - ▶ コードだけでは不明確な意図 (何のために (why), 何をしているのか (what)) をコメントにする.
 - ▶ 「どうやっているのか (how)」や「ソースコードの翻訳」はコメントとして不適. 例えば, 「a を 10 で割った剰余を 10 倍したものに a を 10 で割った商を加える」というコメントはソースコードを翻訳してるだけで役に立たない.
- ▶ 13 行目

```
printf("%d\n", a);          /* aの2乗を表示 */
```

入れ替えた結果を表示

レポート課題 III-2 (1)

レポート課題 III-2 (3 桁の整数の順序を反転させる)

キーボードから入力された **3 桁の整数** (例えば 735) と, その **順序を反転** させたもの (537) を以下の形式:

```
735 <-> 537
```

で出力するプログラムを作り, その **ソースファイル** と, 以下の 4 つの整数

123, 601, 556, 888

に対する **出力** を提出せよ. ただし, 下記を満足すること.

- ▶ 提出するファイル名は B5TB9999_reverse.c (ソースファイル) および B5TB9999_reverse.txt (実行結果) とせよ.

(続く)

レポート課題 III-2 (2)

課題 III-2 の仕様 (続き)

- ▶ 出力結果は, Terminal に表示された結果をコピー&ペーストするか, >> を用いた **追記リダイレクト** (第9回参照) を使うこと.
- ▶ 「入力された整数」もしくは「順序を反転させた整数」が **3桁にならない場合** の動作は問わない. (例えば 100 を反転させた 001 は3桁の整数にならないので, 100 を入力された時の動作は問題としない)
- ▶ ソースファイルの各部分がどのような機能を果たすのかを **コメント** で説明せよ. 特に, **入力された整数を反転させる部分** は「用いた式の **意図**」を明確に記述すること.

(さらに続く)

レポート課題 III-2 (3)

課題 III-2 の仕様 (さらに続き)

- ▶ 今回までの講義で紹介されていない C 言語の機能 (分岐, 繰返しや配列など) を使ってもよい. ただし,
 - ▶ 該当する部分でどのような処理が行なわれるのかを コメント を用いて記載すること
 - ▶ 整数型 以外の変数や配列は使わないこと (数値を文字列にして逆順にして数値に戻す, とかはナシ).

提出期限: 2015 年 6 月 18 日 (木)

レポート課題 III-2 の評価基準

必須要素

守られていない場合は減点

- ▶ 提出ファイル名 は適切か
- ▶ C 言語ソースファイル と 出力結果 を提出しているか
- ▶ gcc でコンパイルでき、生成したファイルを実行できるか
- ▶ 123, 601, 556, 888 の入力に対して 適切な出力 がされるか
- ▶ 式や処理について 十分なコメント が記載されているか

加点要素 (1)：技術の習得

- ▶ 講義で使っていない機能 (分岐・繰返し・配列など) の利用 (ただし、十分なコメントが付されている場合に限る)

加点要素 (2)：創意工夫

- ▶ 指定されていない数値 (ただし多くても 10 個まで) についての実行結果

レポート課題 III-2 の進め方 (1)

1. `mkdir` コマンドを用いて
~/Documents/report/Report-III/Report-III-2/ というディレクトリを作る (-p オプションを使うと便利).
2. Home でのマウス操作や `cp` コマンドなどで `calc3.c` をコピーし, 1. で作ったディレクトリの下に
`B5TB9999_reverse.c` という名前で保存する.
3. `cd` コマンドを用いて, 現在の作業ディレクトリを
~/Documents/report/Report-III/Report-III-2/ に変更する.
4. `gedit` を用いて, `B5TB9999_reverse.c` を「3桁の整数を反転させたもの」を出力するように変更する.
 - ▶ 入力された値と反転させた値を出力する必要があるので, 入力を格納する変数 (a) とは **別の変数** (例えば b) が必要.
 - ▶ 入力された値の「1の位」「10の位」「100の位」をそれぞれ取り出す方法を考えよう.

レポート課題 III-2 の進め方 (2)

1 の位 a を 10 で割った剰余 ($735 \% 10 = 5$)

100 の位 a を 100 で割った商 ($735 / 100 = 7$)

10 の位 「 a の下 2 桁」を 10 で割った商 ($35 / 10 = 3$) に等しい. a の下 2 桁 を取り出すには？

5. gcc コマンドを用いて B5TB9999_reverse.c をコンパイルし、実行ファイルを作成する.
 - ▶ -o オプションを使って実行ファイル名を reverse などとしてみてもよい.
6. 実行ファイルを起動し、様々な入力に対して指定された形式通りに出力されるかを確認する.
 - ▶ 実行ファイル名の前には、現在の作業ディレクトリを表す ./ が必要.
7. 実行結果を B5TB9999_reverse.txt として保存する.
 - ▶ **gedit を使う方法:**
gedit を起動し、Terminal 上の実行結果をコピー＆ペーストし、B5TB9999_reverse.txt という名前で保存する.

レポート課題 III-2 の進め方 (3)

- ▶ **追加** リダイレクトを使う方法:

Terminal 上で、コマンドの後に `>> hoge hoge.txt` とすると、コマンドを実行した結果を `hoge hoge.txt` というファイルに **追記** できるつまり、実行ファイルが `./reverse` という名前だった場合、

```
$ ./reverse >> B5TB9999_reverse.txt
```

とすることで、実行結果をテキストファイルに **追記** できる。
(`>>` の代わりに `>` を使うと追記ではなく **上書き** になるので注意)