2018 年度 情報領域演習第三
— 第 3 回 —

はじめに

前回同様，CED において以下のコマンド (青字の部分) を実行することで出席を表明できる．ただし，N は所属するクラス名 1 ，2 ，3 で置き換え，N の前には空白を入れないこと．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/03/checkerN
提出開始: 10 月 xx 日 14 時 40 分
提出締切: 10 月 yy 日 14 時 39 分
ユーザ: p1610999, 出席状況: 2018-10-05-14-58
 問題:    結果 | 提出日時           | ハッシュ値                        |
    1:  未提出 |                    |                                  |
    2:  未提出 |                    |                                  |
                        ⋮
```

出席を表明するには，授業の開始から 30 分以内に実行する必要がある．実行しても出席状況が「欠席」である場合は教員に伝えること．なお，上の出力は例であるので，実際の締切りは各自コマンドで確認せよ．

提出も同じコマンドを用いて以下のように実行する．ただし，N はクラス名 (1 から 3)，X は 1 から 8 のいずれかの問題番号であり，prog.c は提出する C プログラムのファイル名であり，X の前後には忘れずに空白を入れること．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/03/checkerN X prog.c
```

ここで，ファイル名として指定するのは，問題番号 X の問題を解く C プログラムのソースファイルであり，コンパイル済みの実行ファイルではない．ファイル名は特に指定しないが，「英数字からなる文字列.c」などとすることが望ましい．このコマンドを実行することにより，指定されたファイルがコンパイルされ，実行テストプログラムが自動的に起動される．コンパイルに失敗した場合にはエラーとなり，提出されたことにはならないので注意すること．コンパイルが成功した場合には複数回の実行テストが行われ，実行テストにも成功すると「成功」と出力される．実行テストに失敗すると「失敗」と出力されるとともに反例となる入力ファイルのパス が出力されるので，失敗した理由を見つけるための参考にするとよい．ただし，入力のない問題の場合は失敗しても反例は出力されない．

正しく提出できたか確認するためには以下のように出席の表明と同じコマンドを用いて確認できる．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/03/checkerN
提出開始: 10 月 xx 日 14 時 40 分
提出締切: 10 月 yy 日 14 時 39 分
ユーザ: p1610999, 出席状況: 2018-10-05-14-47
 問題:    結果 | 提出日時           | ハッシュ値                        |
    1:  成功 | 2018-10-05-15-33   | 9b762c81932f3980cf03a768e044e65b |
                        ⋮
```

ハッシュ値は，提出した C プログラムのソースファイルの MD5 値である．CED では md5sum コマンドを用いて MD5 値を見ることにより，提出したファイルと手元のファイルが同じものであるかを確認することができる．

```
[p1610999@blue00 ~]> md5sum prog.c
9b762c81932f3980cf03a768e044e65b
```

なお，一度「成功」となった問題に対し，実行テストに失敗するプログラムを送信してしまうと，結果が「失敗」になってしまうので注意すること．

注意事項

演習を始める前に以下のすべての注意事項をよく読んでおくこと．これらの内容の多くは本演習のウェブページ上にも書かれているが，必ずしも読まれていないようなので再掲する．

履修に関する注意事項

- 無断欠席や途中退席はしないこと．やむを得ず欠席・退席する場合には教員に連絡すること (途中退席は記録に残ります)．
- わからない場合は教員や TA，友人などに相談してもよいが，絶対に他人のプログラムを流用しないこと．プログラムの流用が発覚した場合は，本科目の単位だけでなく同じ学期のすべての単位が無効になることがあるので十分に注意せよ．
- 自分が作成したプログラムを他人に見られる状態で置いている学生がすでに多く見つかっている．プログラムを他人に流用されないように必ずパーミッションを設定すること．

  ```
  [p1610999@blue00 ~]> chmod 600 prog1.c
  ```

  と実行すれば，他人がファイル `prog1.c` を読み書きできないようになる．他人が読み書きできないかどうか確認するには，`ls -l` コマンドを用いて

  ```
  [p1610999@blue00 ~]> ls -l prog1.c
  -rw------- 1 p1610999 I16A  123  10月 8 09:35 2018 prog1.c
  ```

  とすれば確認できる (左に表示されるパーミッションが `-rw-r--r--` などとなっている場合には，他人が読める状態であるので上の対処が必要である)．

  なお，プログラムを作るたびにパーミッションを設定するのが面倒であれば，

  ```
  [p1610999@blue00 ~]> mkdir -m 700 jr3
  ```

  のように，中身を他人に見られないディレクトリを作成し，ディレクトリの下でプログラムを作成するとよい．すでに，情報領域演習第三用のディレクトリを用意しているのであれば，

  ```
  [p1610999@blue00 ~]> chmod 700 jr3
  ```

  とすればよい．他人がディレクトリの中を覗けないか確認するには，`ls -ld` コマンドを用いて，

  ```
  [p1610999@blue00 ~]> ls -ld jr3
  -rwx------ 2 p1610999 I16A 4096  10月 8 09:37 2018 jr3
  ```

  とする (パーミッションが `-rw-r-xr-x` や `-rw---x--x` などの場合は対処が必要である)．ディレクトリのパーミッションが `700` であればその中にあるプログラムは他人に見られることはない．

課題に関する注意事項

- 各問題の入出力例の後に**ヒント**を載せていることがあるので，行き詰まった場合は参考にするとよい．
- 実行テストで失敗すると「反例となる入力のファイルのパス (例:`counterexamples/ce_1_xxx.in`) 」が表示される．反例というのは，失敗の原因となった入力であり，失敗の理由を見つけるヒントとなる．手元でコンパイルしたプログラム `a.out` に対して以下の例のように実行しよう．

  ```
  [p1610999@blue00 ~]> ./a.out < counterexamples/ce_1_xxx.in
  ```

- 実行テストプログラムは出力が正しければ「成功」と表示するが，問題の指示に従っていないプログラムである場合には採点の際に正解とならないことに注意せよ．

1 構造体の復習

黄金比 $\varphi = \dfrac{1+\sqrt{5}}{2} \approx 1.618\ldots$ は不思議な性質をもつ数でレオナルド・ダ・ヴィンチを始め，歴史上の人々を魅了してきたが，この科目は情報領域演習なので深入りはしない．ちょっと数学っぽい話になるが，中学校数学程度の知識しか仮定しないので安心してよい．この演習で扱うのは，

$a+b\varphi$ (a,b は整数) の形をした数は，足しても引いても掛けてもこの形で表すことができる

という性質である．たとえば，$1-4\varphi$ と $2+3\varphi$ の和・差・積は，

$$(1-4\varphi)+(2+3\varphi) = (1+2)+(-4+3)\varphi = 3-\varphi$$
$$(1-4\varphi)-(2+3\varphi) = (1-2)+(-4-3)\varphi = -1-7\varphi$$
$$(1-4\varphi)\times(2+3\varphi) = 2-5\varphi-12\varphi^2 = 2-5\varphi-12(\varphi+1) = -10-17\varphi$$

のように，必ず $a+b\varphi$ の形になる (ここで $\varphi^2=\varphi+1$ であることを利用しているが，これは $\varphi = \dfrac{1+\sqrt{5}}{2}$ から簡単に確かめられる)．今回の演習では，このような形の数を C 言語で構造体を用いて

```
1 struct golden {
2   long long int a;
3   long long int b;
4 };
```

で表すことにしよう．たとえば，メンバ `a` の値が `3`，メンバ `b` の値が `2` であるような構造体は $3+2\varphi$ を表し，メンバ `a` の値が `0`，メンバ `b` の値が `-1` であるような構造体は $-\varphi$ を表す．`long long int` 型の代わりに `int` 型を用いることもできるが，後の計算で a や b がとても大きな整数になりうるので `long long int` 型を用いている．なお，`printf` 関数や `scanf` 関数で `long long int` 型の値を扱うには，`%d` の代わりに `%lld` というフォーマットを用いる．

構造体の変数を宣言するときは，

```
1 struct golden x = { 3, 2 };
2 struct golden y = { 0, -1 };
```

のようにすれば $3+2\varphi$ を表す変数 `x` と $-\varphi$ を表す変数 `y` を宣言できるが，

```
1 long long int i = 100;
2 struct golden x;
3 x = { i*2, i+3 }; /* ← 正しくない記述 */
```

のように，構造体の変数に後から代入する場合には `{ }` による記法を用いることはできない．そこで，

```
1 x.a = i*2;
2 x.b = i+3;
```

などとする必要がある．

問題 1

標準入力から与えられる $a+b\varphi$ (a,b は整数) の形の数に対し，その 2 倍を $s+t\varphi$ (s,t は整数) の形で表したときの s と t を空白 1 文字で繋いで標準出力に出力するプログラムを作成せよ．ただし，$a+b\varphi$ の形の数は a，b を空白で繋いだ文字列で表されるものとし，入力に現れる整数の絶対値は 10^{18} 以下と仮定してよい．

入力例	出力例
`3 -1`	`6 -2`

入力例

```
20170123456789 -12345678902017
```

出力例

```
40340246913578 -24691357804034
```

プログラムは上で定義された構造体 golden を用いて，以下の 2 つの関数を含むプログラムとする (他にヘッダファイルのインクルードや golden の定義も必要である).

```c
/* double_golden: 第 1引数を 2倍した数を返す関数 */
struct golden double_golden(struct golden x) {
  struct golden y;
  /* ここは適切に埋める */
  return y;
}
/* 以下はコメント以外は変更しないこと */
int main() {
  struct golden x;
  scanf("%lld␣%lld", &x.a, &x.b); /* &x.a は &(x.a) と同じ意味 */
  x = double_golden(x);
  printf("%lld␣%lld\n", x.a, x.b);
  return 0;
}
```

なお，これらの関数を含まなくてもこの問題の実行テストで「成功」と出力させることはできるが，採点では 正解にはならない ので注意せよ (以下の問題も同様).

問題 2

標準入力から与えられる $a+b\varphi$ (a,b は整数) の形の 2 つの数に対し，それらの和を $s+t\varphi$ (s,t は整数) の形で表したときの s と t を空白 1 文字で繋いで標準出力に出力するプログラムを作成せよ．ただし，入力として与えられる文字列は 2 行で各行は $a+b\varphi$ の形の数を 2 つの整数 a，b を空白で繋いだ文字列で表したものであり，各整数の絶対値は long long int 型の 10^{18} 以下と仮定してよい.

入力例

```
1 -4
2 3
```

出力例

```
3 -1
```

入力例

```
12345678902017 -20170987654321
-20170987654321 -12345678902017
```

出力例

```
-7825308752304 -32516666556338
```

プログラムは上で定義された構造体 golden を用いて，以下の 2 つの関数を含むプログラムとする.

```c
/* add_golden: 第 1引数と第 2引数の和を返す関数 */
struct golden add_golden(struct golden x, struct golden y) {
  struct golden z;
  /* ここを適切に埋める */
  return z;
}
/* 以下はコメント以外は変更しないこと */
int main() {
  struct golden x, y;
  scanf("%lld␣%lld", &x.a, &x.b);
  scanf("%lld␣%lld", &y.a, &y.b);
  x = add_golden(x, y);
  printf("%lld␣%lld\n", x.a, x.b);
  return 0;
}
```

問題 3

標準入力から与えられる $a+b\varphi$ (a,b は整数) の形の 2 つの数に対し，それらの積を $s+t\varphi$ (s,t は整数) の形で表したときの s と t を空白 1 文字で繋いで標準出力に出力するプログラムを作成せよ．ただし，入力として与えられる文字列は 2 行で各行は $a+b\varphi$ の形の数を 2 つの整数 a，b を空白で繋いだ文字列で表したものであり，各整数の絶対値は `long long int` 型の 10^9 以下と仮定してよい．

入力例

```
1 -4
2 3
```

出力例

```
-10 -17
```

入力例

```
123456789 -987654321
987654321 -123456789
```

出力例

```
243865262225270538 -868770005427526293
```

プログラムは上で定義された構造体 `golden` を用いて，以下の 2 つの関数を含むプログラムとする．

```c
/* mult_golden: 第 1引数と第 2引数の積を返す関数 */
struct golden mult_golden(struct golden x, struct golden y) {
  struct golden z;
  /* ここを適切に埋める */
  return z;
}
/* 以下はコメント以外は変更しないこと */
int main() {
  struct golden x, y;
  scanf("%lld␣%lld", &x.a, &x.b);
  scanf("%lld␣%lld", &y.a, &y.b);
  x = mult_golden(x, y);
  printf("%lld␣%lld\n", x.a, x.b);
  return 0;
}
```

ヒント　$a+b\varphi$ の形の数の積は次のようにして計算できる ($\varphi^2=\varphi+1$ を利用している):

$$\begin{aligned}(a_1+b_1\varphi)\times(a_2+b_2\varphi) &= a_1(a_2+b_2\varphi)+b_1\varphi(a_2+b_2\varphi)\\ &= a_1a_2+a_1b_2\varphi+a_2b_1\varphi+b_1b_2\varphi^2\\ &= a_1a_2+a_1b_2\varphi+a_2b_1\varphi+b_1b_2(\varphi+1)\\ &= (a_1a_2+b_1b_2)+(a_1b_2+a_2b_1+b_1b_2)\varphi\end{aligned}$$

問題 4

標準入力から与えられる $a+b\varphi$ (a,b は整数) の形の数と非負整数 n に対し，$(a+b\varphi)^n$ を $s+t\varphi$ (s,t は整数) の形で表したときの s と t を空白 1 文字で繋いで標準出力に出力するプログラムを作成せよ．なお，後に示す漸化式に基づく再帰関数により計算するものとし，標準出力には `mult_golden` を呼び出した回数も出力する．ただし，入力として与えられる文字列は 2 行で 1 行めは整数 a，b を空白で繋ぐことで $a+b\varphi$ を表しており，2 行めは非負整数 n を表す．なお，$(a,b,n)\neq(0,0,0)$ とし，計算中に現れるどの整数も `long long int` 型の範囲を超えないと仮定してよい．

入力例

```
2 -3
5
```

出力例

```
563 -465
5
```

入力例

```
1234 -5678
4
```

出力例

```
1472103282658000 1562935050238800
4
```

入力例

```
1 -1
50
```

出力例

```
20365011074 -12586269025
50
```

プログラムは上で定義された構造体 golden を用いて，以下の大域変数 number と 3 つの関数を含むプログラムとする．関数 mult_golden は number の値を更新していること以外は問題 3 と同じであり，関数 power_golden は，

$$x^0 = 1\ (= 1 + 0\varphi) \qquad x^{k+1} = x \times x^k \quad (k \geq 0)$$

によって計算される再帰関数であるとする ($\times$ のために mult_golden を用いる)．なお，x^1 を計算するには mult_golden を使わなくても可能だが，ここでは上の式の通りに $x^1 = x \times x^0 = x \times 1$ と計算する．

```
1  int number = 0;
2  /* mult_golden: 第 1引数と第 2引数の積を返す関数 */
3  struct golden mult_golden(struct golden x, struct golden y) {
4    number += 1;
5    struct golden z;
6    /* 問題 3 と同じもの */
7    return z;
8  }
9  /* power_golden: 第 1引数の第 2引数乗を返す関数 */
10 struct golden power_golden(struct golden x, int n) {
11   struct golden z;
12   /* ここは適切に埋める */
13   return z;
14 }
15 /* 以下はコメント以外は変更しないこと */
16 int main() {
17   struct golden x;
18   int n;
19   scanf("%lld␣%lld", &x.a, &x.b);
20   scanf("%d", &n);
21   x = power_golden(x, n);
22   printf("%lld␣%lld\n", x.a, x.b);
23   printf("%d\n", number);
24   return 0;
25 }
```

問題 5

標準入力から与えられる $a + b\varphi$ (a, b は整数) の形の数と非負整数 n に対し，$(a + b\varphi)^n$ を $s + t\varphi$ (s, t は整数) の形で表したときの s と t を空白 1 文字で繋いで標準出力に出力するプログラムを作成せよ．問題 4 と同じ内容であるが，後に示す別の漸化式に基づく再帰関数により計算することに注意せよ．この問題においても，標準出力には mult_golden を呼び出した回数も出力する．ただし，入力として与えられる文字列は 2 行で 1 行めは整数 a，b を空白で繋ぐことで $a + b\varphi$ を表しており，2 行めは非負整数 n を表す．なお，$(a, b, n) \neq (0, 0, 0)$ とし，計算中に現れるどの整数も long long int 型の範囲を超えないと仮定してよい．

入力例

```
2 -3
5
```

出力例

```
563 -465
4
```

入力例

```
1234 -5678
4
```

出力例

```
1472103282658000 1562935050238800
3
```

入力例

```
1 -1
50
```

出力例

```
20365011074 -12586269025
8
```

プログラムは上で定義された構造体 golden を用いて，以下の大域変数 number と 3 つの関数を含むプログラムとする．問題 4 と同様に，関数 mult_golden は number の値を更新していること以外は問題 3 と同じである．また，関数 power_golden は，

$$x^0 = 1\ (= 1 + 0\varphi) \qquad x^{2k+1} = x \times x^{2k} \quad (k \geq 0) \qquad x^{2k} = x^k \times x^k (k \geq 1)$$

によって計算される再帰関数であるとするが，$x^{2k} = x^k \times x^k$ のとおりに書いて x^k を 2 度計算するのではなく x^k は 1 度しか計算しないものとする)．

```
1  int number = 0;
2  /* mult_golden: 第 1引数と第 2引数の積を返す関数 */
3  struct golden mult_golden(struct golden x, struct golden y) {
4    number += 1;
5    struct golden z;
6    /* 問題 3 と同じもの */
7    return z;
8  }
9  /* power_golden: 第 1引数の第 2引数乗を返す関数 */
10 struct golden power_golden(struct golden x, int n) {
11   struct golden z;
12   /* ここは適切に埋める */
13   return z;
14 }
15 /* 以下はコメント以外は変更しないこと */
16 int main() {
17   struct golden x;
18   int n;
19   scanf("%lld␣%lld", &x.a, &x.b);
20   scanf("%d", &n);
21   x = power_golden(x, n);
22   printf("%lld␣%lld\n", x.a, x.b);
23   printf("%d\n", number);
24   return 0;
25 }
```

問題 6

フィボナッチ数列とは，$F_0 = 0$，$F_1 = 1$，$F_n = F_{n-1} + F_{n-2}$ $(n \geq 2)$ で定義される数列であり，以下の等式が成り立つことを簡単に示せる:

$$(1 - \varphi)^{n-1} = F_n - F_{n-1}\varphi$$

この等式は，「$(1 - \varphi)^{n-1}$ を $a + b\varphi$ $(a, b$ は整数$)$ の形で表したときの a が F_n そのものである」ということを表している．この性質と問題 5 のプログラムを利用して，フィボナッチ数列の n 番め $(n \geq 1)$ を求めるプログラムを作成せよ．標準入力には正の整数 n $(1 \leq n \leq 90)$ が与えられ，標準出力には F_n と long long int 同士の足し算，掛け算の回数を 3 行に分けて出力するものとする．

入力例
```
10
```
出力例
```
55
15
25
```

入力例
```
42
```
出力例
```
267914296
24
40
```

入力例
```
90
```
出力例
```
2880067194370816120
30
50
```

この例からもわかるように，このプログラムでは，通常は 89 回の足し算が必要な F_{90} の計算が 30 回の足し算と 50 回の掛け算で行うことができる．一方，問題 4 のプログラムを利用すると，F_{90} の計算には 270 回の足し算と 450 回の掛け算が必要であることを考えると，問題 5 のプログラムが非常に効率がよいことがわかる．残念ながら，`long long int` 型だけでは F_{92} までしか計算ができないが，仮にどんな大きな整数も表せる整数型 (多倍長整数) を用いるなら，F_{100000} (2 万桁以上になる！) の計算でも，78 回の足し算と 130 回の掛け算で行うことができる (ここは驚くところ)．

ヒント 関数 `mult_golden` が 1 回呼び出されると，`long long int` 同士の足し算や掛け算が何回ずつ行われているか考えよ．

問題 7

※この問題は発展問題である．時間が余った人や余力のある人は挑戦するとよい．

標準入力から与えられる $a+b\varphi$ (a, b は整数) の形の 2 つの数に対し，1 つめの方が大きければ `1` を，2 つめの方が大きければ `-1` を，どちらも等しければ `0` を標準出力に出力するプログラムを作成せよ．ただし，入力として与えられる文字列は 2 行で各行は $a+b\varphi$ の形の数を 2 つの整数 a，b を空白で繋いだ文字列で表したものであり，各整数は `long long int` 型で絶対値は 10^{10} 以下と仮定してよい．

入力例
```
5 -3
-3 2
```
出力例
```
-1
```

入力例
```
3 -1
-2 2
```
出力例
```
1
```

入力例
```
3 -2
3 -2
```
出力例
```
0
```

問題 8

※この問題は発展問題である．時間が余った人や余力のある人は挑戦するとよい．

標準入力で与えられる整数 n に対し，フィボナッチ数列の n 番め F_n を 2018 で割った余りを標準出力に出力するプログラムを作成せよ．ただし，n は 10^9 以下の正の整数であると仮定してよい．

入力例
```
10
```
出力例
```
55
```

入力例

```
100
```

出力例

```
1705
```

入力例

```
1000000000
```

出力例

```
1997
```

ヒント

- 前回の問題 1 と同じであるが，今回は入力の n が大きいので同じように計算できないことに注意せよ．問題 6 までの考え方を利用するとよいが，`mult_golden` を再定義する必要がある．
- $n+m$ を d で割った余りは,「n を d で割った余り」と「m を d で割った余り」を足したものを d で割った余りに等しい．
- $n \times m$ を d で割った余りは,「n を d で割った余り」と「m を d で割った余り」を掛けたものを d で割った余りに等しい．