Q1 課題 「構造体と関数」

西野 順二 (MI プログラム)　（original : 山崎匡）

2018 年 4 月 9 (I3)・12 (I1)・13 (I2) 日

1　まず出席確認

/ced-home/staff/q/q1/shusseki-iN を実行せよ。ここで、N はクラス番号 $N \in \{1, 2, 3\}$ である。一応やり方は、端末エミュレータで

```
[a1709999@purple99 ~/]$ /ced-home/staff/q/q1/shusseki-iN
```

である。

2　次に準備

自分のホームディレクトリ[*1]の下に q1 というディレクトリを作成し、その下に移動せよ。やり方は、端末エミュレータで

```
[a1709999@purple99 ~/]$ mkdir q1
[a1709999@purple99 ~/]$ cd q1
[a1709999@purple99 ~/q1]$
```

という感じ。

次に、/ced-home/staff/q/q1 の下にある q1-で始まるファイル名のファイルを全てコピーせよ。以下のようにすれば良い。

```
[a1709999@purple99 ~/q1]$ cp /ced-home/staff/q/q1/q1-* .
```

3　はじめに

算数の世界では

$$\frac{1}{3} \times 3 = 1 \tag{1}$$

となるのが普通だが、同じ事を計算機にやらせようとして C 言語で

ソースコード 1　q1-badexample1.c

```
1 #include<stdio.h>
2 int main(void)
3 {
4   int a = 1/3;
```

[*1] まあどこでもいいけど、特にこだわりがなければ$HOME で。

```
5   int b = 3;
6   int c = a * b;
7   printf("%d\n", c);
8   return 0;
9 }
```

というプログラムを書いて実行するとなんと答えは 0 になる。

これは `int` 型の場合 1/3 が切り捨てられて 0 になるからである。C 言語に新たに有理数型とそれに伴う演算を追加し、このような計算を可能にするのが今回の演習課題である。なお、今回の課題は P 演習の復習も兼ねている。ウォーミングアップとして楽しんで欲しい。

4 課題説明

以下ではまず `q1-1.c` を例にして進める。ファイルを開いて眺めながら以下を読み進めよ。

4.1 有理数型の定義

有理数は n/d という形をとる。ここで n, d はそれぞれ整数であり、n は分子 (numerator)、d は分母 (denominator) である。また $d \neq 0$ である。このように有理数は 2 つの整数からなる数なので、構造体 `rational` を

ソースコード 2 `q1-1.c`

```
4 typedef struct {
5   int n, d; // n: 分子, d: 分母
6 } rational;
```

と定義すれば、新たに有理数型 `rational` を定義できたことになる。

4.2 有理数型の演算

続いて演算を定義していく。

4.2.1 変数の代入

有理数を 1 つ変数に代入する関数 `setr` を定義しよう。

ソースコード 3 `q1-1.c`

```
8  rational setr(int n, int d)
9  {
10   rational z;
11   if (d == 0) {
12     fprintf(stderr, "Denominator␣must␣be␣nonzero.\n");
13     exit(1);
14   }
15   z.n = n;
16   z.d = d;
17   return z;
18 }
```

この関数は、2 つの整数 n, d を引数として取り、それぞれを分子、分母とする有理数 z を定義して値として返すものである。これを使えば

ソースコード 4 q1-1.c

```
27  rational r = setr(1, 3); // z = 1/3
```

とすると、変数 r に有理数 1/3 が代入される。なお、分母が 0 はありえない状況なので、d が 0 の場合はエラーを出して終了することにする (11–14 行目)。

注意: なお、C99 以降では rational z = {.n = 1, .d = 2}; のように書ける。

4.2.2 変数の表示

変数に代入された有理数を表示する関数 printr も定義しよう。

ソースコード 5 q1-1.c

```
20 void printr(rational z)
21 {
22   printf("%d/%d\n", z.n, z.d); // -> 1/3
23 }
```

この関数は、有理数 z を引数として取り、その分子 (z.n) と分母 (z.d) をそれぞれ n/d の形式で表示するものである。これを使えば先ほど定義した有理数 r に対して

ソースコード 6 q1-1.c

```
28  printr(r);
```

とすると 1/3 と表示される。

これで最低限、変数の代入と表示ができる。

[チェックポイント]

ここで一回プログラムを動かしてみよう。q1-1.c をコンパイルして実行してみよ。念のため、コンパイルは

```
[a1709999@purple99 ~/q1]$ gcc -Wall -std=c99 -o q1-1 q1-1.c
```

とすれば良く、実行は

```
[a1709999@purple99 ~/q1]$ ./q1-1
1/3
[a1709999@purple99 ~/q1]$
```

とすれば良い[*2]。

動作を確認したら、引き続き 2 つの有理数同士の演算を定義していこう。ここからサンプルプログラムは q1-2.c になるので、それを開いて一緒に読み進めよう。

4.2.3 かけ算

(約分しなければ) かけ算は簡単。素直に

$$\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd} \tag{2}$$

を計算する関数を mulr とすると、

[*2] gcc の場合は-std=c99 をつけておくと良い。

ソースコード 7 q1-2.c

```
25 rational mulr(rational x, rational y)
26 {
27   int n = x.n * y.n;
28   int d = x.d * y.d;
29   rational z = setr(n, d);
30   return z;
31 }
```

となる。2 つの有理数 x, y を引数として取り、その分子同士、分母同士をかけて新たに有理数 z を求めて返す関数である[*3]。

4.2.4 足し算

(約分しなくても) 足し算はちょっと面倒くさい。足し算は

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd} \tag{3}$$

となるので、これを計算する関数を addr とすると、

ソースコード 8 q1-2.c

```
33 rational addr(rational x, rational y)
34 {
35   int n = x.n * y.d + y.n * x.d;
36   int d = x.d * y.d;
37   rational z = setr(n, d);
38   return z;
39 }
```

となる。2 つの有理数 x, y を引数として取り、式 (3) を計算して新たに有理数 z を求めて返す関数である。

ここまでで、かけ算と足し算を試してみよう。q1-2.c をコンパイルして実行すると、次の関数が実行されるようになっている。

ソースコード 9 q1-2.c

```
55 void example2(void)
56 {
57   rational x = setr(1, 2); // x = 1/2
58   rational y = setr(2, 3); // y = 2/3
59
60   rational a = mulr(x, y); // a = x * y
61   printr(a); // -> 2/6
62
63   rational b = addr(x, y); // a = x + y;
64   printr(b); // -> 7/6
65 }
```

結果として以下のように表示されるはずである。

```
[a1709999@purple99 ~/q1]$ gcc -Wall -o q1-2 q1-2.c
[a1709999@purple99 ~/q1]$ ./q1-2
```

[*3] わざわざ変数 n, d を定義する必要はないし setr する必要もないけれど、まあ教育的配慮。

```
2/6
7/6
[a1709999@purple99 ~/q1]$
```

[初級課題 1]

引き算する関数 subr、割り算する関数 divr を作成し、q1-2.c に追加せよ。約分はしなくて良い。作成したらチェックプログラムを走らせて採点を行う。以下をよく読んで注意深く作業すること。

1. q1-2.c には既に関数 subr, divr のひな形が用意されている。これを埋めて関数を完成させる。

ソースコード 10 q1-2.c

```
41 rational divr(rational x, rational y)
42 {
43   rational z;
44
45   // ここを埋める
46
47   if (もし割り算した結果分母が 0だったら) {
48     fprintf(stderr, "Division␣by␣zero.\n");
49     exit(1);
50   }
51
52   // ここを埋める
53
54   return z;
55 }
56
57 rational subr(rational x, rational y)
58 {
59   rational z;
60
61   // ここを埋める
62
63   return z;
64 }
```

注意として、割り算の場合、0 で割ってしまうことが考えられる。そのような場合は、“Division by zero.” というメッセージを表示して停止する (47–50 行目)[*4]。

2. 次に、関数 main の中身を修正して、example2() をコメントアウトし、代わりに check1() を有効にする。以下のようになれば良い。

ソースコード 11 q1-2.c

```
int main(int argc, char *argv[])
{
  // example2();
  check1(argc, argv);
  return 0;
```

[*4] こういう例外処理はとても大事。

```
}
```

3. このプログラムをコンパイルすると、以下のように実行することが可能になる。また、関数 subr, divr が正しく書けていれば、以下の出力が得られる。

```
[a1709999@purple99 ~/q1]$ gcc -Wall -std=c99 -o q1-2 q1-2.c
[a1709999@purple99 ~/q1]$ ./q1-2 1 2 3 4
-2/8
4/6
[a1709999@purple99 ~/q1]$
```

   プログラム名の後に並べた 4 つの数字に注目しよう。これは 2 つの有理数を与えており、具体的には、1 つ目の有理数の分子、1 つ目の有理数の分母、2 つ目の有理数の分子、2 つ目の有理数の分母、を与えている。つまり、$1/2$, $3/4$ という有理数を与えており、出力されているのはそれぞれ $1/2 - 3/4$, $1/2 \div 3/4$ の計算結果である $-2/8$, $4/6$ である。

4. もし関数が正しく書けたと確信したら、以下のようにしてプログラムの動作をチェックせよ。

```
[a1709999@purple99 ~/q1]$ /ced-home/staff/q/q1/check1-iN q1-2.c
```

   ここで、N は自分のクラス番号 ($N \in \{1, 2, 3\}$) である。check1-iN は、適当なテストケースをランダムに発生させて動作をチェックするプログラムである。動作が正しければ、「正解」と出力して加点する。間違っていた場合は、どういうテストケースでどう間違えたかを出力する。

5. チェックプログラムは何回実行してもかまわない。好きなだけ試すこと。
6. なお、自分の現在の点数は

```
[a1709999@purple99 ~/q1]$ /ced-home/staff/q/q1/score-iN
```

   で調べられる。同様に N は自分のクラス番号 ($N \in \{1, 2, 3\}$) である。

4.3 型の異なる 2 つの変数同士の演算

ところで、この段階で `1/3 * 3 = 1` を計算させようとして、

ソースコード 12 q1-badexample2.c

```
22 rational z = setr(1, 3); // r = 1/3
23 int a = 3;
24 int b = z * a; // 1/3 * 3 = 1???
25 printf("%d\n", b);
```

と書いてみてもコンパイル時エラーとなる (試してみよ)。なぜなら、整数型と新しく定義した有理数型の間の演算規則はまだ定義してないからである。これをやるためには、まず

1. 整数を一回有理数に変換し、
2. 有理数として計算し、
3. 最後に整数に戻す

という操作が必要である。

[初級課題 2]

整数を有理数に変換する関数 itor[*5]、有理数を整数に変換する関数 rtoi[*6] を作成せよ。ひな形は q1-3.c である。それぞれ以下のような関数になる。

ソースコード 13 q1-3.c

```
55 rational itor(int a)
56 {
57   rational r;
58
59   // a から r を計算する
60   ここを埋める
61
62   return r;
63 }
64
65 int rtoi(rational r)
66 {
67   int a;
68
69   // r から a を計算する
70   ここを埋める
71
72   return a;
73 }
```

整数を有理数に変換する場合は、分母を 1 にすればよい。逆に、有理数を整数に変換する場合、割り切れない場合は切り捨てることにしよう。

これの関数 itor, rtoi を作成し、コンパイルして実行せよ。main 関数から関数 example3 が実行されて、itor, rtoi がどちらも正しく動作していれば 1 が出力されはずである。

動作を確認したらチェックプログラムを走らせて採点を行う。やり方は初級課題 1 と同様に、以下のようにする。

1. 関数 main の中身を修正して、example3() をコメントアウトし、代わりに check2() を有効にする。以下のようになれば良い。

ソースコード 14 q1-3.c

```
int main(int argc, char *argv[])
{
  // example3();
  check2(argc, argv);
  return 0;
}
```

2. このプログラムをコンパイルすると、以下のように実行することが可能になる。また、関数 itor, rtoi が正しく書けていれば、以下の出力が得られる。

[*5] integer to rational
[*6] rational to integer

```
[a1709999@purple99 ~/q1]$ gcc -Wall -std=c99 -o q1-3 q1-3.c
[a1709999@purple99 ~/q1]$ ./q1-3 1 3 3
1
```

プログラム名の後に並べた 3 つの数字に注目する。最初の 2 つは有理数の分子と分母、最後の 1 つは整数である。つまり $1/3, 3$ という 2 つの数字を与えている。チェックはこの 2 つの数のかけ算を行うので、$1/3 \times 3 = 1$ が出力される。

3. もし関数が正しく書けたと確信したら、以下のようにしてプログラムの動作をチェックせよ。

```
[a1709999@purple99 ~/q1]$ /ced-home/staff/q/q1/check2-iN q1-3.c
```

ここで、N は自分のクラス番号 ($N \in \{1, 2, 3\}$) である。後は初級課題 1 と同様。

[中級課題 1]

ちょっと唐突だが 2 つの整数の最大公約数を計算する関数 gcd を作成せよ。プログラムのひな形が q1-4.c にあるのでそれを使うこと。ところでユークリッドの互除法は習ってますよね。

動作を確認したらチェックプログラムを走らせて採点を行う。これまでと同様に、main 関数内の example4() をコメントアウトし、コメントアウトされている check3() を有効にする。その後チェックプログラム check3-iN を使ってテストする。

[中級課題 2]

q1-3.c, q1-4.c を組み合わせて、これまでに作成した setr, addr, subr, mulr, divr を、かならず約分した有理数を返すように修正した q1-5.c を作成せよ (ヒント: 関数 gcd を使う)。q1-5.c のひな形が用意されているので、それを修正すること。例えば

ソースコード 15 q1-5.c

```
1 rational x;
2 x = setr(6, 8);
3 printr(x);
```

というコードは $6/8$ ではなく $3/4$ を出力するはずである。

動作を確認したらチェックプログラムを走らせて採点を行う。これまでと同様に、main 関数内の example5() をコメントアウトし、コメントアウトされている check4() を有効にする。その後チェックプログラム check4-iN を使ってテストする。

[上級課題]

N 個の有理数を足し合わせる関数 reductionr を作成せよ。プログラムのひな形は q1-6.c である。関数定義は

ソースコード 16 q1-6.c

```
1 rational reductionr(int n, rational ary[]);
```

とする。ここで n は足し合わせる有理数の個数、ary[] は有理数の配列である。例えば

ソースコード 17 q1-6.c

```
1 void example6(void)
2 {
3   rational ary[3];
4   ary[0] = setr(1, 2);
5   ary[1] = setr(2, 3);
6   ary[2] = setr(3, 4);
7   rational z = reductionr(3, ary);
8   printr(z);
9 }
```

を実行すると、$1/2 + 2/3 + 3/4 = 23/12$ を出力するはずである。

作成したら、例によって example6() をコメントアウトしてかつ check5() を有効にし、チェックプログラム check5-iN に与えて動作をテストせよ。