Stosując standardowe metody wczytywania danych z pliku tekstowego, możemy umieścić liczniki i mianowniki wszystkich ułamków w tablicach, na przykład:

```
int licz[1000], mian[1000];
```

Spostrzegawczy czytelnik zapewne zauważył, że wczytywanie danych do tablicy nie jest konieczne, gdyż w każdym z zadań wystarczy jednorazowo sekwencyjnie przejrzeć ciąg danych od początku do końca. Dla czytelności rozwiązania przyjmujemy jednak, że dane są wczytane do tablicy i każde z zadań rozwiązujemy osobno.

65.1.

Naturalnym sposobem rozwiązania zadania 1 wydaje się porównywanie **wartości** ułamków, zapamiętanych w zmiennych typu float lub double[6] w C/C++. Aby odróżnić ułamki o tej samej wartości, musimy wówczas porównywać ich mianowniki. Zadanie sprowadza się wówczas do zaimplementowania algorytmu szukania **minimum rozszerzonego**. Z dwóch ułamków o tej samej wartości, ale różnych mianownikach, za mniejszy uznajemy ten, który ma mniejszy mianownik.

Powyższe rozwiązanie obarczone jest jednak błędem zaokrągleń związanych z reprezentacją zmiennopozycyjną liczb. Może to prowadzić do sytuacji, w których porównanie dwóch ułamków o tej samej wartości nie wskaże na ich równość. Aby się przekonać o takiej możliwości, polecamy czytelnikowi sprawdzenie działania następującego fragmentu kodu programu:

```
double x,y,z,v, a,b;

x=1.0; y=3.0;
z=2.0; v=6.0;
b=x/y;
cout << "Roznica: " << z/v-b<< endl;
```

Dlatego w omawianym rozwiązaniu będziemy porównywać ułamki, sprowadzając je do wspólnego mianownika. Unikniemy wówczas operacji na liczbach w reprezentacji zmiennoprzecinkowej (float/double). Sprawdzenie, czy l_1 / m_1 jest mniejsze od (lub równe) l_2 / m_2, sprowadza się do sprawdzenia, czy $l_1 \cdot m_2$ jest mniejsze od (lub równe) $l_2 \cdot m_1$:

```
bool mniejszy(long l1, long m1, long l2, long m2){
    return (l1*m2<l2*m1);
}
bool rowny(long l1, long m1, long l2, long m2){
    return (l1*m2==l2*m1);
}
```

Ułamek o najmniejszej wartości i najmniejszym mianowniku nazwijmy *minimalnym*. Stosując powyższą funkcję, poszukujemy ułamka minimalnego w pętli, przechowując licznik i mianownik ułamka minimalnego spośród dotychczas przejrzanych w zmiennych `minL`

---

[6] Pamiętać przy tym należy, że np. w języku C podstawienie x=a/b dla zmiennych a i b typu int oraz x typu float da w wyniku zaokrąglenie w dół a/b do liczby całkowitej; aby uzyskać wynik dokładny, konieczne jest wymuszenie konwersji typu, np. przez zastąpienie instrukcji x=a/b przez x=1.0*a/b lub x=(float)a/(float)b.

i `minM`. Wartości te zmieniają się zawsze po napotkaniu ułamka o wartości mniejszej od `minL/minM` lub ułamka o wartości `minL/minM`, ale o mianowniku mniejszym od `minM`. Poniżej prezentujemy fragment kodu realizujący opisany sposób rozwiązania:

```
int minL, minM;

minL=licz[0];
minM=mian[0];
for(int i=1; i<1000; i++){
    if (mniejszy(licz[i],mian[i],minL,minM)){
        minL=licz[i]; minM=mian[i];
    }
    else if (rowny(licz[i],mian[i],minL,minM)){
            if (mian[i]<minM){
                minL=licz[i]; minM=mian[i];
            }
        }
}
```

65.2.

Wiadomo, że ułamek jest w postaci nieskracalnej wtedy i tylko wtedy, gdy największy wspólny dzielnik licznika i mianownika jest równy 1. Aby sprowadzić ułamek do postaci nieskracalnej, wystarczy zatem podzielić zarówno licznik, jak i mianownik ułamka przez ich największy wspólny dzielnik. Dlatego przydatnym narzędziem do rozwiązania zadania może być funkcja wyznaczająca największy wspólny dzielnik dwóch liczb. Poniżej podajemy taką funkcję, opartą na algorytmie Euklidesa:

```
int  nwd(int n, int m){

    if (m==0) return n;
    return nwd(m,n%m);
}
```

Przy skorzystaniu z powyższej funkcji zadanie sprowadza się do zliczenia liczby takich $i \in [0, 999]$, że `nwd(licz[i], mian[i])` jest równe 1. Poniżej podajemy przykładową implementację takiego zliczania:

```
int ileN=0;

for(int i=0; i<1000; i++){
    if (nwd(licz[i],mian[i])==1)
        ileN++;
}
```

Zwróćmy jednak uwagę, że zadanie to można rozwiązać, nawet nie wiedząc o zależności między największym wspólnym dzielnikiem licznika i mianownika a nieskracalnością ułamka. Dla ułamka o postaci a / b wystarczy sprawdzić dla każdej z liczb $i \in [2,\min(a,b)]$, czy i dzieli zarówno a, jak i b. Poniżej prezentujemy fragment takiego rozwiązania:

```
int mniejsza, j, ileN=0;
bool czyN;
```

```
for(int i=0; i<1000; i++){
    if (licz[i]<mian[i]) mniejsza = licz[i];
    else mniejsza = mian[i];
    j=2; czyN = true;
    while (j<=mniejsza){
        if ((licz[i]%j==0) && (mian[i]%j==0))
            czyN=false;
            j++;
    }
    if (czyN) ileN++;
}
```

Po wykonaniu powyższego kodu liczba ułamków w postaci nieskracalnej znajdzie się w zmiennej `ileN`.

Zwróćmy jeszcze na koniec uwagę na to, że powyższy algorytm można znacznie przyspieszyć, kończąc pętlę `while` po znalezieniu pierwszego wspólnego dzielnika licznika i mianownika, a więc wtedy, gdy wartość zmiennej `czyn` zmieni się na `false`.

65.3.

Zauważmy, że do uzyskania nieskracalnej postaci ułamka a / b wystarczy podzielić a oraz b przez `nwd(a,b)`, tj. przez największy wspólny dzielnik a i b. Inaczej mówiąc, licznik nieskracalnej postaci to `a/nwd(a,b)`, a mianownik to `b/nwd(a,b)`. Na przykład

$$\frac{8}{12} = \frac{8/\text{nwd}(8,12)}{12/\text{nwd}(8,12)} = \frac{2}{3}$$

Naturalnym rozwiązaniem zadania jest więc wyznaczenie sumy wartości `licz[i]/nwd(licz[i],mian[i])` dla $i \in [0,999]$. Poniżej przykładowa implementacja takiego rozwiązania, gdzie wynik umieszczany jest w zmiennej `sumL`:

```
long d, sumL=0;
for(int i=0; i<1000; i++){
    d=nwd(licz[i],mian[i]);
    sumL = sumL + licz[i]/d;
}
```

65.4.

Wiedząc, że każdy mianownik jest dzielnikiem liczby b, możemy wszystkie ułamki przekształcić do postaci, w której mianownik jest równy b. Dokładniej, korzystamy z tożsamości: $\frac{x}{y} = \frac{x \cdot b/y}{b}$. Zadanie sprowadza się wówczas do zsumowania wartości `licz[i]·b/mian[i]` dla $i \in [0,999]$. Przy takim rozwiązaniu pojawia się jednak pewna pułapka, polegająca na tym, że niektóre wartości `licz[i]·b` dla pewnych i przekraczają zakres liczb typu long w popularnych kompilatorach C/C++. Dlatego konieczne jest użycie typu long long. Poniżej przykładowy fragment takiego rozwiązania, w którym `liczS/b` jest równe sumie ułamków `licz[0]/mian[0],…,licz[i]/mian[i]`:

```
long long mianS, liczS, nowyLicz;
liczS=0;
mianS=2*3*5*7*13;

mianS=mianS*mianS/13;
for(int i=0; i<ile; i++){
    nowyLicz = licz[i]*mianS/mian[i];
    liczS+= nowyLicz;
}
```