Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

Computer Programming 143
Rekenaarprogrammering 143

Practical 5
***Prakties* 5**

2025

Aim of Practical 5: / *Doel van Prakties 5:*

a) Write simple functions
Skryf eenvoudige funksies

b) Use functions from the C Standard `math.h`, `stdlib.h` and `time.h` libraries
Gebruik funksies uit die C Standaard `math.h`-, `stdlib.h`- en `time.h`-biblioteke

c) Combine functions to achieve complex functionality
Kombineer funksies om komplekse funksionaliteit te verkry

d) Write recursive functions
Skryf rekursiewe funksies

Instructions / *Instruksies*

1. Attendance is **compulsory** for all the practical sessions of your assigned group. See the module framework for more details.
   *Bywoning is **verpligtend** vir al die praktiese sessies van jou groep. Raadpleeg die modulegids vir meer inligting.*

2. Each practical will be followed up with a compulsory practical test. This test will take place during the last 30 minutes of each practical session. See the module framework for more details.
   Elke prakties word deur 'n verpligte toets opgevolg. Die toets vind plaas gedurende die laaste 30 minute van elke praktika sessie. Raadpleeg die modulegids vir meer inligting.

3. You may use code from this practical to complete the test as well as Code::Blocks.
   Jy kan kode van hierdie prakties sowel as Code::Blocks gebruik om die toets te voltooi.

4. If you miss more than two tests for whatever reason, you will receive an **incomplete** for the module. See the module framework for more details.
   *As jy meer as twee toetse om watter rede ook al mis, sal jy 'n **onvoltooid** vir die module ontvang. Raadpleeg die modulegids vir meer inligting.*

5. You must do all assignments **on your own**. Students are encouraged to help each other **understand** the problems and solutions, but each should write his/her own code. By simply copying someone else's code or solutions, you will not build an understanding of the work and be reported for plagiarism.
   *Jy moet alle opdragte **op jou eie** doen. Studente word aangemoedig om mekaar te help om die probleme en oplossings te **verstaan**, maar elkeen moet sy/haar eie kode skryf. Deur eenvoudig die kode of oplossings van iemand anders te kopieër, sal jy nie 'n begrip van die werk opbou nie en word dit vir plagiaat aangemeld.*

6. You are responsible for your own progress. Complete the whole practical assignment in your own time if you could not do so in the practical session. Ensure that you understand the practical work. Check your work against the memorandum that will be posted by the end of the practical sessions on STEMLearn.
   Jy is verantwoordelik vir jou eie vordering. Voltooi die hele praktiese voorskrif op jou eie tyd indien jy dit nie kon doen in die praktiese sessies= nie. Sorg dat jy die praktiese werk verstaan. Kontroleer jou werk teen die memorandum wat aan die einde van die praktiese sessies op STEMLearn geplaas sal word.

7. Create a new project for each question and name it consistently. For example, the project for Question A of Practical 4 should be called **Assignment4A**. Make sure that this is the only open project in the workspace before compiling the program.
   *Skep 'n nuwe projek vir elke vraag en gebruik eenvormige name. Die projek vir Vraag A van Prakties 4 moet byvoorbeeld genoem word **Assignment4A**. Maak seker dat hierdie projek die enigste oop projek in die werkruimte is voordat jy die program vertaal.*

8. Include a comment block at the top of each source file according to the format given. It must include the correct filename and date, your name and student number, the copying declaration, and the title of the source file.
   Sluit 'n kommentaarblok aan die bokant van elke bronnelêer in volgens die gegewe formaat. Dit moet die korrekte lêernaam en datum, jou naam en studentenommer, die kopiëringverklaring en die titel van die bronlêer bevat.

9. Read each question completely before you start solving it.
   Lees elke vraag volledig voordat jy begin om dit op te los.

10. For each question, either draw a flowchart or write pseudocode to describe your algorithm **before** you start programming.
    *Vir elke vraag, teken 'n vloeidiagram of skryf pseudokode om jou algoritme te beskryf **voordat** jy begin programmeer.*

11. **Indent your code correctly.** Making your code readable is not beautification, it is a time- and life-saving habit. Adhere to the standards (refer to the documents on STEMLearn).
    ***Spasieer jou kode korrek.** Om jou kode leesbaar te maak is nie onnodig nie, dit is 'n tyds- en lewensreddende gewoonte. Volg die standaard (verwys na die dokumente op STEMLearn).*

12. Comment your code sufficiently well. It is required for you and others to understand what you have done.
    Voeg genoeg kommentaar by jou kode. Dit word vereis sodat jy en ander kan verstaan wat jy gedoen het.

13. Ensure that you back up the project folder for each question (e.g. by copying it to a flash drive).
    Maak seker dat jy die projekvouer rugsteun ("back up") vir elke vraag (bv. deur dit na 'n "flash drive" te kopieer.

14. After you have completed the practical assignment, do the practice test on STEMLearn to prepare for the weekly practical test.
    Nadat jy die praktiese opdrag voltooi het, doen die oefentoets op STEMLearn om voor te berei vir die praktiese toets.

Question A

Goal: / *Doel:*
Write a function that does complex rounding by using `math.h` functions.
Skryf 'n funksie wat komplekse afronding doen deur `math.h`-funksies te gebruik.

Problem description / *Probleembeskrywing*

1. The function `floor(x)` in in the `math.h` library rounds the floating-point number `x` *down* (i.e., to the nearest integer that is smaller than or equal than `x`). Similarly, the function `ceil(x)` rounds the floating-point number `x` *up*. These functions can also be used to do more complex rounding. For example, `floor(10 * x) / 10` will round `x` down to the first decimal (e.g., `floor(10 * 12.345) / 10` will produce `12.3`).
   Die funksie `floor(x)` in die `math.h`-biblioteek rond die wisselpunt-getal `x` na onder *(d.w.s., na die naaste heelgetal wat kleiner of gelyk aan `x` is). Die funksie `ceil(x)` rond die wisselpunt-getal `x` soortgelyk na* bo. *Hierdie funksies kan gebruik word om meer komplekse afronding te doen. As 'n voorbeeld, `floor(10 * x) / 10` sal `x` na onder rond tot die eerste desimaal (bv., `floor(10 * 12.345) / 10` sal `12.3` produseer.)*

2. Since coins smaller than 5c are not used anymore, the amounts used in cash transactions should be rounded to 5c. Your task is to write a function that takes an amount of money as argument and returns the amount rounded to 5c. The rounding has to happen in the following way: Positive amounts must be *rounded down* and negative amounts must be *rounded up*. Use the function prototype below.
   Aangesien munte kleiner as 5c nie meer gebruik word nie, moet die bedrae wat gebruik word in kontant-transaksies afgerond word na 5c. Jou taak is om 'n funksie te skryf wat 'n bedrag geld as argument neem en die getal wat afgerond is na 5c terugstuur. Die afronding moet soos volg geskied: Positiewe bedrae moet na onder gerond *word, en negatiewe bedrae moet* na bo gerond *word. Gebruik die funksieprototipe hieronder.*

```
double round_to_5c(double amount);
```

3. Write a `main` function that reads in an amount of money from the user, calls the `round_to_5c` function, and displays the rounded amount.
   Skryf 'n `main`-funksie wat 'n bedrag geld van die gebruiker inlees, die `round_to_5c`-funksie roep, en die afgeronde bedrag vertoon.

4. Sample output: / *Voorbeeld van die afvoer:*
   Sample 1: / *Voorbeeld 1:*

```
Enter an amount of money (R.c): 1954.19
The amount rounded to 5c: R1954.15
```

   Sample 2: / *Voorbeeld 2:*

```
Enter an amount of money (R.c): -203.63
The amount rounded to 5c: R-203.60
```

Question B

Goal: / *Doel:*
Combine several custom functions as well as functions of the `stdlib.h` and `time.h` libraries to perform a statistical analysis of a game of chance.
Kombineer verskeie eie funksies asook funksies uit die `stdlib.h`- en `time.h`-biblioteke om 'n statistiese analise van 'n kansspel te verrig.

Problem description / *Probleembeskrywing*

1. Consider the following game of chance: The game starts by flipping a (fair) coin. If the result is heads, you receive R1 and the game continues; however, if the result is tails, you receive R0 and the game ends. If the game continues, the coin is flipped again and you receive the same payoff. The game therefore continues until tails first appears, and your total winnings is the number of consecutive heads tosses from the start of the game (in rands).
   Beskou die volgende kansspel: Die spel begin deur 'n (regverdige) muntstuk op te skiet. As die resultaat kop is, kry jy R1 en die spel gaan voort, maar as die resultaat stert is, kry jy R0 en die spel eindig. Indien die spel voortgaan, word die muntstuk weer opgeskiet en jy ontvang dieselfde uitbetaling. Die spel gaan dus voort totdat stert vir die eerste keer verskyn, en jou totale uitbetaling is die getal van opeenvolgende kop-resultate vanaf die begin van die spel (in rand).

2. Monte Carlo simulations are repeated simulations of systems with random behaviour. By recording the result of every simulation and combining the results, one can calculate properties of the random system. For example:
   Monte-Carlo-simulasies is herhaalde simulasies van stelsels met lukrake ("random") gedrag. Deur die resultaat van elke simulasie aan te teken en die resultate te kombineer, kan 'n mens eienskappe van die lukrake stelsel bereken. Byvoorbeeld:

   (a) If the simulation is repeated N times and the numerical result of the nth simulation is given by x_n, the expected value (or statistical mean) of the result x can approximately be calculated as
   Indien die simulasie N keer herhaal word en die numeriese resultaat van die $n$de simulasie gegee word deur x_n, kan die verwagte waarde (of statistiese gemiddeld) van die resultaat x benaderd bereken word as

   $$E[x] \approx \frac{1}{N}\sum_{n=1}^{N} x_n.$$

   (b) For each simulation n, if the outcome A_n is logical (either 0 = false, or 1 = true), then the probability of A happening can be approximated by the fraction of simulations where $A = 1$. This is expressed mathematically as:
   Vir elke simulasie n, indien die uitkoms A_n logies is (óf 0 = vals, óf 1 = waar), dan kan die waarskynlikheid dat A gebeur benader word deur die fraksie van simulasies waar $A = 1$. Dit word wiskundig uitgedruk as:

   $$P[A] \approx \frac{1}{N}\sum_{n=1}^{N} A_n.$$

The larger the number of simulations (N), the more accurate the answers will be.
Hoe groter die hoeveelheid simulasies (N), hoe meer akkuraat sal die antwoorde wees.

3. Your task is to use Monte Carlo simulations to determine the following two properties of the game of chance described in point 1:
Jou taak is om Monte-Carlo-simulasies te gebruik om die volgende twee eienskappe van die kansspel wat in punt 1 beskryf is te bereken:

   (a) The expected (or mean) payoff of a game
   Die verwagte (of gemiddelde) uitbetaling van 'n spel

   (b) The probability that the payoff of a game will reach R5 (i.e., be ≥ R5)
   Die waarskynlikheid dat die uitbetaling van 'n spel R5 sal bereik (m.a.w., ≥ R5)

4. Your program should have the following structure and behaviour:
Jou program moet die volgende struktuur en gedrag hê:

   (a) The result of a coin toss should be represented by the following enumeration:
   Die resultaat van die opskiet van 'n muntstuk moet deur die volgende "enumeration" voorgestel word:

```
enum CoinTossResult {HEADS, TAILS};
```

   (b) You should have a function that simulates one coin toss and returns the result. The function prototype should be:
   Jy moet 'n funksie hê wat die opskiet van een muntstuk simuleer en die resultaat terugstuur. Die funksie-prototipe moet wees:

```
enum CoinTossResult coinToss(void);
```

   (c) You should have a function that simulates one game and returns the payoff of the game (in rands). This function should call function `coinToss`. The function prototype should be:
   Jy moet 'n funksie hê wat een spel simuleer en die uitbetaling vir die spel (in rand) terugstuur. Hierdie funksie moet funksie `coinToss` roep. Die funksie-prototipe moet wees:

```
int playGame(void);
```

   (d) You should have a function called `expectedPayoff` to simulate 1 000 000 games (i.e., perform Monte Carlo simulation). The purpose of the function is to determine the expected payoff (or mean payoff) of a game, using the method described in point 2a. The function should call `playGame` and return the expected (mean) payoff. The function prototype should be:
   Jy moet 'n funksie genaamd `expectedPayoff` hê om 1 000 000 spele te simuleer (d.w.s., doen Monte-Carlo-simulasie). Die doel van die funksie is om die verwagte (of gemiddelde) uitbetaling van 'n spel te bereken, deur die metode beskryf in punt 2a te gebruik. Die funksie moet `playGame` roep en die verwagte (gemiddelde) uitbetaling terugstuur. Die funksie-prototipe moet wees:

```
float expectedPayoff(void);
```

(e) You should have a function called `probabilityThatPayoffReaches` to simulate 1 000 000 games (i.e., perform Monte Carlo simulation). The purpose of the function is to determine the probability that the payoff received in a game reaches R5 (i.e., be ≥ R5), using the method described in point 2b. The function should take a threshold as argument (which will be set to R5 when the function is called), it should call `playGame`, and return the probability that the payoff reaches the threshold. The function prototype should be:
Jy moet 'n funksie genaamd `probabilityThatPayoffReaches` hê om 1 000 000 spele te simuleer (d.w.s., doen Monte-Carlo-simulasie). Die doel van die funksie is om die waarskynlikheid dat die uitbetaling vir 'n spel R5 bereik (d.w.s., ≥ R5) te bereken, deur die metode van punt 2b te gebruik. Die funksie moet 'n drumpelwaarde as argument neem (wat gestel gaan word tot R5 wanneer die funksie geroep word), dit moet `playGame` roep, en die waarskynlikheid dat die uitbetaling die drumpelwaarde bereik terugstuur. Die funksie-prototipe moet wees:

```
float probabilityThatPayoffReaches(int threshold);
```

(f) In the `main` function, you should ensure that the seed of the random number generator is different for each program execution, as well as call functions `expectedPayoff` and `probabilityThatPayoffReaches` and display the results.
In die `main`-funksie moet jy verseker dat die saad van die toevalsgetal-generator verskillend is vir elke uitvoering van die program, asook die funksies `expectedPayoff` en `probabilityThatPayoffReaches` roep en die resultate vertoon.

5. Sample output of the program: / *Voorbeeldafvoer van die program:*

Sample 1: / *Voorbeeld 1:*

```
Expected payoff: R0.999728
Probability that payoff reaches R5: 3.133%
```

Sample 2: / *Voorbeeld 2:*

```
Expected payoff: R1.001598
Probability that payoff reaches R5: 3.136%
```

Note: / *Neem kennis:*
The exact output of your program will likely vary due to randomness. To reduce the differences between results, you can increase the number of simulations.
Die presiese afvoer van jou program sal waarskynlik varieer as gevolg van willekeurigheid ("randomness"). Om die verskille tussen die resultate te verminder, kan jy die hoeveelheid simulasies vermeerder.

For interest's sake: / *Vir interessantheid:*
If you have the necessary probability theory background, it is easy to calculate the results of this question analytically. Can you see how to do this?
As jy die nodige agtergrond in waarskynlikheidsteorie het, is dit maklik om die resultate van hierdie vraag analities te bereken. Kan jy sien hoe om dit te doen?

The St. Petersburg paradox: / *Die St. Petersburg-paradoks:*
If you change the rules of the game of chance in this question such that the payoff *doubles* for each subsequent coin flip, then the expected payoff for the game is infinite. Does this make sense to you? (If you want to read more, search for "St. Petersburg paradox".)
As jy die reëls van die kansspel in hierdie vraag aanpas sodat die uitbetaling verdubbel *vir elke opeenvolgende opskiet van die muntstuk, dan is die verwagte uitbetaling vir die spel oneindig. Maak dit sin vir jou? (As jy meer wil lees, soek vir "St. Petersburg paradox".)*

Programming guidelines / *Programmeringsriglyne*

1. First design and implement function `coinToss`. Write code in your `main` function to test function `coinToss`, and verify that it works as expected.
   Ontwerp en implementeer eers funksie `coinToss`. Skryf kode in jou `main`-funksie om funksie `coinToss` te toets, en verifieer dat dit werk soos verwag.

2. Then design and implement function `playGame`, which should use function `coinToss`. Write code in your `main` function to test function `playGame`, and verify that it works as expected.
   Ontwerp en implementeer dan funksie `playGame`, wat funksie `coinToss` moet gebruik. Skryf kode in jou `main`-funksie om funksie `playGame` te toets, en verifieer dat dit werk soos verwag.

3. Lastly, design and implement functions `expectedPayoff` and `probabilityThatPayoffReaches`, which should both use function `playGame`. Write the code in your `main` function to call these functions and produce the required output.
   Laastens, ontwerp en implementeer funksies `expectedPayoff` en `probabilityThatPayoffReaches`, wat beide funksie `playGame` moet gebruik. Skryf kode in jou `main`-funksie om hierdie funksies te roep en die vereiste afvoer te lewer.

Question C

Goal: / Doel:
Write a recursive function to determine the greatest common divisor of two integers.
Skryf 'n rekursiewe funksie om die grootste gemene deler van twee heelgetalle te bereken.

Problem description / *Probleembeskrywing*

1. The greatest common divisor (GCD) of two integers is the largest positive integer that divides the two integers[1]. For example, the GCD of 40 and 24 is 8, or written differently, $\text{GCD}(40,24) = 8$.
   Die grootste gemene deler (GCD, n.a.v. "greatest common divisor") van twee heelgetalle is die grootste positiewe heelgetal wat die twee heelgetalle deel[2]*. Byvoorbeeld, die GCD van 40 en 24 is 8, of anders geskryf, GCD*$(40,24) = 8$.

2. The GCD can be calculated using the Euclidean algorithm, which is recursively defined as:
   Die GCD kan bereken word met die Euclidiese algoritme, wat rekursief gedefinieer is as:
   $$\text{GCD}(x,y) = \begin{cases} |x| & \text{if } y = 0 \\ \text{GCD}(y, x \bmod y) & \text{if } y \neq 0 \end{cases}$$
   For example, the GCD of 40 and 24 can be calculated as follows:
   Byvoorbeeld, die GCD van 40 en 24 can soos volg bereken word:
   $$\begin{aligned} &\text{GCD}(40,24) = \text{GCD}(24, 40 \bmod 24) = \text{GCD}(24,16) \\ &\text{GCD}(24,16) = \text{GCD}(16, 24 \bmod 16) = \text{GCD}(16,8) \\ &\text{GCD}(16,8) = \text{GCD}(8, 16 \bmod 8) = \text{GCD}(8,0) \\ &\text{GCD}(8,0) = 8 \end{aligned}$$

3. Design and implement a recursive function to calculate the GCD[3]. Use the following function prototype:
   Ontwerp en implementeer 'n rekursiewe funksie om die GCD te bereken[4]*. Gebruik die volgende funksie-prototipe:*
   ```
   int GCD(int a, int b);
   ```

4. Write a `main` function that reads in two integers from the user, and then calculates and displays the GCD of the two integers by calling function `GCD`.
   Skryf 'n `main`-funksie wat twee heelgetalle van die gebruiker inlees, en dan die GCD van die twee heelgetalle bereken en vertoon deur funksie `GCD` te roep.

[1] A *divisor* of two integers is also called a *factor* of the two integers.
[2] 'n *Deler* van twee heelgetalle word ook 'n *faktor* van die twee heelgetalle genoem.
[3] To implement this function, you will probably want to take the absolute value of an integer. You can do this by using the function `abs()` that is available from the `stdlib.h` library.
[4] Om hierdie funksie te implementeer sal jy waarskynlik die absolute waarde van 'n heelgetal wil neem. Om dit te doen kan jy die funksie `abs()` gebruik wat beskikbaar is vanaf die `stdlib.h`-biblioteek.

5. Sample output of the program: / *Voorbeeldafvoer van die program:*

Sample 1: / *Voorbeeld 1:*

```
Enter number 1: 5929
Enter number 2: 2310

The greatest common divisor of 5929 and 2310 is 77
```

Sample 2: / *Voorbeeld 2:*

```
Enter number 1: -27
Enter number 2: -36

The greatest common divisor of -27 and -36 is 9
```

Question D

Goal: / Doel:
Write and combine several functions to calculate the sum of two fractions.
Skryf en combineer verskeie funksies om die som van twee breuke te bereken.

Problem description / *Probleembeskrywing*

1. The least common multiple (LCM) of two integers is the smallest positive integer that is divisible by both integers. For example, the LCM of 40 and 24 is 120, or written differently, $\mathrm{LCM}(40,24) = 120$.
   Die kleinste gemene veelvoud (LCM, n.a.v. "least common multiple") van twee heelgetalle is die kleinste positiewe heelgetal wat deelbaar is deur beide heelgetalle. Byvoorbeeld, die LCM van 40 en 24 is 120, of anders geskryf, LCM$(40,24) = 120$.

   The LCM can be calculated from the GCD as follows:
   Die LCM kan soos volg vanaf die GCD bereken word:

$$\mathrm{LCM}(x,y) = \frac{|x| \cdot |y|}{\mathrm{GCD}(x,y)}$$

2. To add two fractions, one can use the following procedure, using the sum of $\frac{1}{40}$ and $\frac{1}{24}$ as example:
   Om twee breuke te sommeer, kan jy die volgende prosedure volg, met die som van $\frac{1}{40}$ *en* $\frac{1}{24}$ *as voorbeeld:*

   (a) Set a common denominator equal to the LCM of the denominators of the two fractions.
   Stel 'n gemene noemer gelyk aan die LCM van die noemers van die twee breuke.

$$\mathrm{LCM}(40,24) = 120$$

   (b) Convert both fractions to use the common denominator.
   Verander beide breuke om die gemene noemer te gebruik.

$$\frac{1}{40} = \frac{1 \cdot \frac{120}{40}}{120} = \frac{3}{120} \qquad \frac{1}{24} = \frac{1 \cdot \frac{120}{24}}{120} = \frac{5}{120}$$

   (c) Add the two numerators.
   Sommeer die twee tellers.

$$\frac{3}{120} + \frac{5}{120} = \frac{8}{120}$$

   (d) Simplify the resulting fraction by dividing the numerator and denominator by their GCD.
   Vereenvoudig die resulterende breuk deur die teller en noemer te deel deur hul GCD.

$$\mathrm{GCD}(120,8) = 8 \;\Rightarrow\; \frac{\frac{8}{8}}{\frac{120}{8}} = \frac{1}{15}$$

3. Your task is to design and write a program that adds two fractions. Your program should have the following structure and behaviour:
Jou taak is om 'n program te ontwerp en te skryf wat twee breuke sommeer. Jou program moet die volgende struktuur en gedrag hê:

   (a) Use the function `GCD` that you wrote in the previous question to calculate the GCD.
Gebruik die funksie `GCD` wat jy in die vorige vraag geskryf het om die GCD te bereken.

   (b) You should have a function that calculates the LCM; it should call the function `GCM`. Use the following function prototype:
Jy moet 'n funksie hê wat die LCM bereken; dit moet die funksie `GCM` roep. Gebruik die volgende funksie-prototipe:

```
int LCM(int a, int b);
```

   (c) You should have a function that takes the numerators and denominators of two fractions as arguments and displays the sum of the two fractions. Use the following function prototype:
Jy moet 'n funksie hê wat die tellers en noemers van twee breuke as argumente neem en die som van die twee breuke vertoon. Gebruik die volgende funksie-prototipe:

```
void print_sum_of_fractions(int num1, int den1, int num2, int den2);
```

   This function should display the resulting fraction as a proper fraction or a mixed number. It should also check whether both denominators are strictly positive (i.e., not zero or negative), and otherwise display an error message.
Hierdie funksie moet die resulterende breuk as 'n egte breuk ("proper fraction") of 'n gemengde getal vertoon. Dit moet ook kyk of beide noemers streng positief is (d.w.s., nie nul of negatief nie), en andersins 'n foutboodskap vertoon.

   (d) The `main` function should read in the numerators and denominators of the two fractions, and then call the `print_sum_of_fractions` function.
Die `main`-funksie moet die tellers en noemers van die twee breuke inlees, en dan die `print_sum_of_fractions`-funksie roep.

4. Sample output of the program: / *Voorbeeldafvoer van die program:*

   Sample 1: / *Voorbeeld 1:*

```
Enter the numerator of fraction 1: 1
Enter the denominator of fraction 1: 6
Enter the numerator of fraction 2: 1
Enter the denominator of fraction 2: 3

The sum of 1/6 and 1/3 is 1/2
```

   Sample 2: / *Voorbeeld 2:*

```
Enter the numerator of fraction 1: -5
Enter the denominator of fraction 1: 6
Enter the numerator of fraction 2: 2
Enter the denominator of fraction 2: 7

The sum of -5/6 and 2/7 is -23/42
```

Sample 3: / *Voorbeeld 3:*

```
Enter the numerator of fraction 1: 5
Enter the denominator of fraction 1: 6
Enter the numerator of fraction 2: 8
Enter the denominator of fraction 2: 9

The sum of 5/6 and 8/9 is 1 13/18
```

Sample 4: / *Voorbeeld 4:*

```
Enter the numerator of fraction 1: 3
Enter the denominator of fraction 1: 2
Enter the numerator of fraction 2: -5
Enter the denominator of fraction 2: 2

The sum of 3/2 and -5/2 is -1
```

Sample 5: / *Voorbeeld 5:*

```
Enter the numerator of fraction 1: 1
Enter the denominator of fraction 1: 0
Enter the numerator of fraction 2: 1
Enter the denominator of fraction 2: 2

Error: The denominator of a fraction must be strictly positive!
```

Programming guidelines / *Programmeringsriglyne*

1. First design and implement function `LCM`. Write code in your `main` function to test function `LCM`, and verify that it works as expected.
   Ontwerp en implementeer eers funksie `LCM`. Skryf kode in jou `main`-funksie om funksie `LCM` te toets, en verifieer dat dit werk soos verwag.

2. Write the "skeleton" of function `print_sum_of_fractions`: Only display the two fractions supplied by the user for now. In your `main` function, add code that reads the two fractions from the user and then calls function `print_sum_of_fractions`. Verify that everything works as expected.
   Skryf die "raamwerk" van die funksie `print_sum_of_fractions`: Vertoon slegs die twee breuke wat deur die gebruiker verskaf word vir nou. In jou `main`-funksie, voeg kode by wat die breuke van die gebruiker inlees en dan die funksie `print_sum_of_fractions` roep. Verifieer dat alles werk soos verwag.

3. Add the check for invalid denominators. Verify that this works.
   Voeg die toets vir ongeldige noemers. Verifieer dat dit werk.

4. Systematically[5] implement the algorithm that adds two fractions. For now, only aim to produce a simple fraction (i.e., of the form $\frac{a}{b}$). Verify that it works for multiple different cases.
   Implementeer die algoritme wat twee breuke sommeer sistematies[6]*. Vir nou, mik slegs om 'n eenvoudige breuk te bereken (d.w.s., van die vorm* $\frac{a}{b}$*). Verifieer dat dit werk vir verskeie verskillende gevalle.*

5. Lastly, design and implement an algorithm to display improper fractions correctly. Verify that your program works for all the test cases.
   Laastens, ontwerp en implementeer 'n algoritme om onegte ("improper") breuke korrek te vertoon. Verifieer dat jou program werk vir alle toetsgevalle.

[5] To implement an algorithm *systematically* means to incrementally implement small parts of the algorithm, and after every addition, to check that code written so far works as expected (by writing test code or using the debugger).

[6] Om 'n algoritme *sistematies* te implementeer beteken om stuk-vir-stuk klein dele van die algoritme te implementeer, en om na elke byvoeging te kyk of die kode wat tot dusver geskryf is werk soos verwag (deur toetskode te skryf of die ontfouter te gebruik).