Programming Languages

Exercise book

Edited by

Filipe Rodrigues
Raquel Bernardino

ISEG - Lisbon School of Economics and Management

Worksheet 1

1. *Hello world!*

Write your first `C++` program that displays the message: "*Hello world!*".

2. Output

Without using variables, write a program that prints the following on the screen:

```
Total = 100%
VAT   =  17%
IRS   =  15%
--------------
Net   =  68%
```

3. Input and Output

Write a program that asks the user for their first and last name and their age, then displays a message using this information. Example: If the user's first name is "Pedro", last name is "Costa", and age is "21", the program should output:

```
Hello Pedro Costa, welcome.
It must be great to be 21 years old.
```

4. Basic arithmetic

Write a program that asks the user for one integer and one decimal number. Create three new variables to store their sum, difference, and product, then display the results.

Example: If the numbers entered are 3 and 2.1, the program should output:

```
Sum: 3 + 2.1 = 5.1
Difference: 3 - 2.1 = 0.9
Product: 3 * 2.1 = 6.3
```

5. String concatenation

Write a program that reads three names. Create a variable containing all three names (separated by spaces) and display it on the screen.

6. Rectangle area and perimeter

Write a program to calculate the area and perimeter of a rectangle. The program should ask the user for the height and length, then print both results.

Example: If the height is 2 and the length is 3, the program should output:

```
The rectangle's perimeter is 10 and its area is 6.
```

7. Temperature conversion

Write a program that asks the user for a temperature in degrees Celsius and converts it to Fahrenheit. Note: Fahrenheit = 1.8 * Celsius + 32.

8. Average

Write a program that reads five integer numbers and calculates their average.

Example: If the inputs are 3, 2, 8, 9, and 5, the program should output:

```
The average of the entered values is 5.4.
```

9. Course grade

Write a program that reads a student's grades for 4 tests and a final exam, then calculates their final grade. Assume each test is worth 10% and the exam is worth 60%.

Example: If the test grades are 17, 16, 15, and 18 and the exam grade is 17, the program should output:

```
Your final grade in the course is 16.8.
```

10. Division operators

Write a program that reads two integers a and b and prints their division in the form: "a = quotient * b + remainder". Also display the decimal division result.

Example: If $a = 8$ and $b = 5$, the program should output:

```
8 = 1 * 5 + 3    and    8/5 = 1.6
```

11. Convert seconds

Write a program that converts a number of seconds into hours, minutes, and seconds. For example, 4000 seconds = 1h6m40s.

12. Reverse order

Write a program that asks the user for a three-digit integer and prints the number in reverse order.

Example: If the input is 135, the program should output:

```
The number entered was 135 and its reverse is 531.
```

Worksheet 2

13. Body mass index (BMI)

Write a program that takes as input a person's weight (in kg) and height (in meters), validates the data (terminating the program if any value entered is invalid), and calculates their BMI, classifying the person according to the following scale:

BMI	Classification
≤ 18.5	Underweight or severely underweight
Between 18.5 and 24.9	Normal weight
≥ 25	Overweight or obese

Note: BMI = weight / height^2

Example: When entering weight 59 and height 1.63, the program should output: `Normal weight (BMI = 22.2063)`

When entering weight 59 and height -1.63, the program should output: `The height entered is not valid`

14. Leap year

Write a program that allows the user to enter a year and check whether it is a leap year. A year is a leap year if and only if it is divisible by 4 but not by 100, or if it is divisible by 400. For example, the years 1700, 1800, and 1900 were not leap years, but the years 1600, 2000, and 2012 were.

15. Quadratic equations

Write a program to solve equations of degree less than or equal to two that is robust enough to handle any combination of user inputs. **Hint:** Start by drawing a flowchart explaining how you would solve an equation of degree less than or equal to two.

16. Date

Write a program that asks the user for a date in the format day/month/year and checks whether it is valid. For example, the date 29/02/2022 is not valid, but the date 03/02/1596 is valid.

17. Right triangle

Write a program that takes 3 integer numbers and checks whether they can represent the sides of a right triangle (use the Pythagorean theorem). For example, the values 2, 5, and 9 cannot represent the sides of a right triangle, but the values 3, 4, and 5 can.

18. Age

Write a program that asks the user for their date of birth and today's date, and calculates their age in years.

19. Course grade excluding the lowest score

In a given course, there are 5 assessment components: 4 exercises and one exam. The final grade is calculated using the 3 best exercise scores and the exam grade, where the exam counts for 50% and the exercises count for the remaining 50%. Additionally, a minimum exam grade of 7 is required to pass the course. Write a program that determines the final grade and validates all input values.

Example: When entering the following grades $e_1 = 15$, $e_2 = 13$, $e_3 = 16$, $e_4 = 18$, and $exam = 18$, the program should output:

```
The final grade for the course is 17.1667.
```

Replacing the exam grade with $exam = 6$, the program should output:

```
Minimum exam grade not achieved, student failed.
```

20. Introduction to loops

Without writing the program below on the computer, indicate the console output for $n = 1$ and for $n = 6$.

```
int main()
{
    int n;
    cout << "n: ";
    cin >> n;

    int b = 2;
    string s = "..";

    for(int i = 1; i<n; ++i){
        cout << ".";
        s+="A";
        if(i%2 == 0)
            --b;
        if(i%3 == 0){
            b+=i;
            s ="B";
        }else{
            b=b*i;
            s+="C";
        }
    }

    cout << "\nb= " << b << "\ns= " << s << "\n\n";
    return 0;
}
```

21. Introduction to loops

Execute the following tasks:

i. Print all numbers from 1 to 15, as shown below, using a loop:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

ii. Print all numbers from 15 down to 1 using a loop:

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

iii. Create a loop starting from `i=3` that performs 12 iterations and prints a `+` symbol in each iteration.

iv. Ask the user for an integer n_1. If it is positive, print all numbers from 0 to n_1. Otherwise, print the number zero $|n_1|$ times. Example: For $n_1 = 5$ the expected output is 0 1 2 3 4 5, and for $n_1 = -5$ the expected output is 0 0 0 0 0.

v. Ask the user for a positive integer n_2 and print n_2 asterisks. Continue asking until a valid value is entered.

vi. Print all even numbers between 10 and 50 (inclusive) in two ways: first using a loop with an *if* statement, and then using only a loop.

vii. Calculate the sum of the first n_3 natural numbers using a loop, where n_3 is provided by the user. Example: For $n_3 = 10$, the sum is 55.

viii. Ask the user for two integers a and b such that $a < b$, and calculate the sum of all numbers between a and b (inclusive). Example: For $a = 5$ and $b = 20$, the sum is 200.

ix. Print the squares of the first 10 natural numbers.

x. Ask the user for a positive integer less than 10 and calculate its factorial. Example: $6! = 720$.

xi. Calculate the sum of the numbers between 1 and 100 that are multiples of 2 but not multiples of 4. Result: The sum of the multiple of 2 that are not multiples of 4 between 1 and 100 is 1250.

22. Output writing

Ask the user for an integer m and print the output below (the example corresponds to $m = 6$):

```
Start...
1 [0, 2]
2 [1, 3]
3 [2, 4]
4 [3, 5]
5 [4, 6]
6 [5, 7]
...End!
```

23. *for*/*while* loop

Write a program that asks the user for an integer n and prints all even numbers in the interval $[1, n[$ using a *for* loop. Then repeat the exercise using a *while* loop.

Example: When entering $n = 10$, the output should be:

```
The even numbers in the interval [1,10[ are: 2 4 6 8
```

24. Sum of the first n numbers

Write a program that receives a positive integer n, validates it (continuously asking until a valid value is entered), and calculates the sum from 1 to n. The program should also print the sum of the numbers between 1 and n that are multiples of 3.

Example: For $n = 10$, the program should output:

```
Sum: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
Sum of multiples of 3: 3 + 6 + 9 = 18
```

25. Series sum

Write a program that asks the user for a positive integer N and calculates the sum of the series $\sum_{k=1}^{N} \frac{k}{k+1}$.

Example: $\sum_{k=1}^{5} \frac{k}{k+1} = 3.55$.

26. Loops and stopping condition

Write a program that repeatedly asks the user for positive integers. The program should stop when a non-positive number is entered and then display the minimum, maximum, and average of the positive numbers entered.

Example: For the sequence 5, 7, 3, 2, 20, 21, and -5, the output should be:

```
The smallest value entered was 2, the largest was 21, and the average was 9.66667.
```

27. Number of divisors

Write a program that asks for a positive integer n, validates it (continuing until a valid number is entered), and displays its number of divisors.

Example: The number 12 has 6 divisors.

28. Perfect number

Write a program that asks the user for a positive integer number n and checks whether it is a perfect number. **Note:** A number is perfect if it equals the sum of its proper divisors (divisors other than the number itself).

Example: 6 is a perfect number because its proper divisors are 1, 2, and 3, and 1+2+3=6.

29. Prime number

Write a program that asks the user for an integer number and checks whether it is prime.

30. List of prime numbers

Write a program that asks the user for a positive integer number n and prints all prime numbers between 1 and n.

Example: For $n = 10$, the output should be: `The prime numbers between 1 and 10 are:  2 3 5 7`

31. Greatest common divisor

Write a program that determines the greatest common divisor (GCD) between two integer numbers entered by the user.

Example: GDC(132, 80) = 4 and GDC(150, 50) = 50.

32. Nested loops 1

Ask the user for a positive integer number k (with validation) and print the output below (example for $k = 6$):

```
Start...
1 -> 0
2 -> 1 0
3 -> 2 1 0
4 -> 3 2 1 0
5 -> 4 3 2 1 0
6 -> 5 4 3 2 1 0
...End!
```

33. Output writing 2

Ask the user for a positive integer k (with validation) and print the output below (example for $k = 6$):

```
Start...
1 -> Sum from 1 to 1 = 1
2 -> Sum from 1 to 2 = 3
3 -> Sum from 1 to 3 = 6
4 -> Sum from 1 to 4 = 10
5 -> Sum from 1 to 5 = 15
6 -> Sum from 1 to 6 = 21
...End!
```

34. Nested loops 2

Ask the user for a positive integer k (with validation) and print the output below (example for $k = 5$):

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

35. Nested loops 3

Ask the user for a positive integer k (with validation) and print the output below (example for $k = 7$):

```
1******
12*****
123****
1234***
12345**
123456*
1234567
```

36. *Hi-lo* game

Implement the *Hi-lo* game. First, the program should generate a random number between 1 and 100. The user has 7 attempts to guess the number.

If the user's guess is incorrect, the program should tell whether it is too low or too high. If the guess is correct, the program should display a winning message. If the user does not guess the correct number within 7 attempts, the program should display a losing message and reveal the correct number.

Note: To generate a random integer between 1 and 100, use the following commands:

```
srand (time(NULL));
int randomNumber = rand() % 100 + 1;
```

Example: The program should output:

```
Let's play a game. I'm thinking of a number between 1 and 100
and you have 7 attempts to guess it.

     Attempt #1: 50
     The number entered is too low.

     Attempt #2: 75
     The number entered is too low.

     Attempt #3: 87
     The number entered is too high.

     Attempt #4: 80
     The number entered is too high.

     Attempt #5: 77
     The number entered is too low.

     Attempt #6: 79
     Correct! You win!
```

37. Calendar

Write a program that asks the user for a year, a month and the weekday (1 - Monday, 2 - Tuesday, 3 - Wednesday, 4 - Thursday, 5 - Friday, 6 - Saturday and 7 - Sunday) on which the month starts, and prints that month's calendar.

Example: For year 2026, month 2, and starting day Sunday, the program should output:

```
|M |T |W |T |F |S |S |
|  |  |  |  |  |  | 1|
| 2| 3| 4| 5| 6| 7| 8|
| 9|10|11|12|13|14|15|
|16|17|18|19|20|21|22|
|23|24|25|26|27|28|  |
```

Worksheet 3

38. Construction of vectors

Write a program that:

i) creates and initializes the vector of integer numbers v = (1, 2, 3);

ii) creates a vector of *strings* s of size 2. Then, fill the first position with the string "AA" and the second with the string "BB";

iii) creates a vector of real numbers r without specifying its size. Resize the vector to have size 3 and insert the values 1.5, 5 and 0.6;

iv) creates a char vector c without specifying its size. Without using the *resize* method, add to the vector the values 'a', '*', '+' and 'v';

v) adds the number 4 to the vector v created in point i);

vi) prints to the screen the vectors created in the previous points.

39. Construction of vectors 2

Write a program that automatically creates the following vectors:

i) $v = (1, 2, 3, ..., 25)$;

ii) $u = (2, 4, 6, ..., 50)$;

iii) $z = (0, 1, 0, 1..., 1)$ (vector of length 30);

40. Norm of a vector

Write a program that asks the user for a vector and computes its norm.

Example: When entering the vector $(1, 3, 5, 7, 9)$, the program should print the following output:

```
||(1, 3, 5, 7, 9)|| = 12.8452
```

41. Manipulation of vectors

Write a program that repeatedly asks the user for values and stores them in a vector of integer numbers. After receiving the vector, and without sorting it, the program must:

i) print the vector in reverse order;

ii) calculate the sum of the values entered in the vector;

iii) indicate how many even numbers there are in the vector;

iv) check whether there are negative values in the vector;

v) calculate the minimum value of the vector;

vi) determine the position of the first even number in the vector. If there are no even numbers in the vector, the position -1 should be reported.

vii) determine the position of the last even number in the vector. If there are no even numbers in the vector, the position -1 should be reported.

viii) create a new vector that contains only the odd numbers present in the original vector;

ix) calculate the last index of the maximum value in the vector;

x) check whether there are repeated elements in the vector.

Example: When entering the vector $v = (3, 4, 5, 7, 8, -2, 8, 4)$, we obtain the following output:

```
i)   The vector entered in reverse order is: (4, 8, -2, 8, 7, 5, 4, 3)
ii)  The sum of the values entered in the vector is 37.
iii) 5 even values were entered.
iv)  There are negative values.
v)   The minimum value entered is -2.
vi)  The position of the first even number in the vector is 1.
vii) The position of the last even number in the vector is 7.
viii)The odd numbers in the original vector are: (3, 5, 7).
ix)  The last position where the maximum of the vector appears is 6.
x)   There are repeated numbers in the vector.
```

42. Inner product

Write a program that starts by repeatedly asking the user for real values until a non-numeric value is entered, and stores them in a vector $v1$. Repeat the process to create a second vector $v2$. Check whether the vectors have the same size and, if so, compute the inner product between them.

Example: When entering vectors $(1, 2, 1)$ and the $(-1, 4, 6)$, the output should be: `(1,2,1)*(-1,4,6)=13`

43. Vector of triples

Write a program that asks the user for a vector of integers. After receiving the vector, the program must create a new vector with the same size as the original one and fill it so that each element is three times the corresponding element in the original vector.

Example: When entering the vector $(1, 8, -2)$ the program must create and print the vector $(3, 24, -6)$.

44. Vector with prime numbers

Write a program that asks the user for a vector of positive integer numbers. After receiving the vector, the program must indicate how many and which prime numbers are present in the vector.

Example: When entering the vector $v = (6, 9, 3, 5, 1, 8, 7)$, the program should print the following output:

```
There are 3 prime numbers in the vector.
The prime numbers in the vector are: (3, 5, 7)
```

45. Normalized vector

Write a program that asks the user for a vector of positive real numbers and then normalizes the vector, that is, modifies the vector so that each entry is divided by the maximum of the numbers entered. Print the resulting vector to the screen. Example: When entering $(3, 4, 1, 2)$, the output must be $(0.75, 1, 0.25, 0.50)$.

46. Vector of *strings*

Write a program that asks the user for 7 *strings* and stores them in a vector of *strings*. The program should then modify the vector of *strings* so that all strings with more than 5 characters are replaced by the letter G and the strings with 5 or fewer characters are replaced by the letter P. **Hint:** the method *.length()* gives the number of characters in a *string*. Example: For (bread, rice, potato, flour, pasta, cheese, chocolate), the result must be (P, P, G, P, P, G, G).

47. Matrices

Write a program that reads a square matrix $n \times n$, where the value of n is entered by the user. The program must then print this matrix and compute its trace. **Note:** The trace of a matrix is the sum of the elements on its diagonal.

Example: When entering the matrix $\begin{bmatrix} 1 & 7 & 7 \\ 9 & 2 & 4 \\ 5 & 11 & 3 \end{bmatrix}$ the program should print the following output:

```
The matrix entered was:
1 7 7
9 2 4
5 11 3
The trace of the entered matrix is 6.
```

48. Vectors and matrices

Write a program that reads a square matrix A of size $n \times n$, and a vector $\vec{v}$ of size n. The program must then print the result of the matrix operation $A\vec{v}$. Example:

$$A \times \vec{v} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 8 & 9 \\ 13 & 4 & 2 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 24 \\ 19 \end{bmatrix}$$

49. Matrix norm

Write a program that asks the user for a matrix and computes its infinity norm. **Note:** The infinity norm of a matrix is the maximum of the sums of the absolute values of the elements in the rows of the matrix.

Example:

```
The matrix entered has norm 15 and is:
 1  3  7
-9  2  4
 3 -1  3
```

50. Operations with matrices

Write a program to perform operations between matrices. The program must present the following menu:

Choose one of the options:

1. Matrix addition
2. Matrix subtraction
3. Matrix multiplication
4. Exit

Each time the user selects an option, they must enter two matrices (first indicating their dimensions). The program must check whether the matrices have the correct dimensions for the chosen matrix operation to be carried out and, if so, perform the matrix operation. The resulting matrix must be printed on the screen.

51. Histogram

Write a program that starts by creating a vector of size 90 with the name `random` and fills it with random integers between 0 and 20 (see note). Based on this vector, create a new vector *histogram* such that in the i-th position it contains the number of values equal to i in the vector *random* and print it on the screen. **Note:** To generate a vector of size 90 with random integers between 0 and 20 use the following commands:

```
srand (0);
vector<int> random(90);
for(int i = 0; i < 90; i++) random[i] = rand() % 21;
```

Example: The program should print the following output:

```
Grade | Freq.      |
0     | 4          | ****
1     | 5          | *****
2     | 8          | ********
3     | 5          | *****
4     | 5          | *****
5     | 1          | *
6     | 6          | ******
7     | 8          | ********
8     | 5          | *****
9     | 0          |
10    | 4          | ****
11    | 3          | ***
12    | 3          | ***
13    | 5          | *****
14    | 5          | *****
15    | 6          | ******
16    | 2          | **
17    | 5          | *****
18    | 2          | **
19    | 2          | **
20    | 6          | ******
```

Worksheet 4

52. Functions

Implement the following functions that take three integer values as arguments:

i) A function that returns the maximum of the input values;

ii) A function that returns the minimum of the input values;

iii) A function that returns the mean of the input values;

iv) A function that writes the values in ascending order;

v) A boolean function that checks whether any of the three numbers is repeated.

vi) A function that returns the difference between the maximum and minimum of the input values.

In the *main* function ask the user for 3 integer values and call the implemented functions.

Example: When entering the values 5, 3 and 3, the program should print the following output:

```
The maximum of the input values is 5.
The minimum of the input values is 3.
The mean of the input values is 3.66667.
3 <= 3 <= 5
There are repeated numbers in the input values.
The difference between the maximum and the minimum is 2.
```

53. Power

Create a function that receives two positive values, a base b (real) and an exponent a (integer), and that computes the power b^a. **Note:** Do not use the *pow* function from the *cmath* package.

Example: $2^{11} = 2048$.

54. Factorial

Create a function that, given a positive integer number n, computes the factorial of that number ($n!$).

Example: $7! = 5040$.

55. Fibonacci

The Fibonacci sequence is given by $F_n = F_{n-1} + F_{n-2}$ for $n > 1$, with $F_0 = F_1 = 1$. Write a function that receives a positive integer number n and determines the n-th element of the Fibonacci sequence.

Example: $F_7 = 21$.

56. Pass-by-reference *versus* pass-by-value

Implement the following functions:

```
double Function1(double x){
    x*=10;
    return x;
}
double Function2(double & x){
    x*=10;
    return x;
}
double Function3(const double & x){
    //x*=10; //ERROR!
    double z = x*10;
    return z;
}
```

Next, in your *main* function, write the code below.

```
  double y = 1.0;
  cout <<"value: " << y << " | Function1: " <<  Function1(y) <<" | value: " << y << endl;
  cout <<"value: " << y << " | Function2: " <<  Function2(y) <<" | value: " << y << endl;
  cout <<"value: " << y << " | Function3: " <<  Function3(y) <<" | value: " << y << endl;
  cout <<"----------------------------------------------------------------" << endl;
  cout <<"value: " << y << " | Function1(1.0): " <<  Function1(1.0) << endl;
  //cout <<"value: " << y << " | Function2(1.0): " <<  Function2(1.0) << endl;  //ERROR!
```

Run the program as it is and analyze the output obtained. Then analyze the two commented lines and explain why they cause errors in the program.

57. References 1

Create a void function that receives a vector of integers and computes both the maximum and minimum values stored in it. Then, in the *main* function,ask the user to input a vector of integers, use the function to obtain the maximum and minimum values, and compute the difference between the maximum and the minimum.

Example: The maximum and minimum values of the vector $(20, 30, 26, 15, 19)$ are 30 and 15, respectively, and the difference between them is 15.

58. References 2

Create a function that receives 3 arguments. One of these arguments should be an angle. Use pass-by-reference to return the sine and cosine of this angle.

Example: If the angle $\alpha = 90$ then $\sin\alpha = 0.893997$ and $\cos\alpha = -0.44807$.

59. Functions 2

Implement one function for each of the items i)–x) of Exercise 42. In the *main* function, ask the user for a vector of integer numbers and call each of the functions created appropriately.

60. Functions 3

Implement a function that counts how many times a given number appears in a vector of integers.

Example: The number 5 appears 3 times in the vector $(2, 4, 1, 5, 4, 7, 5, 5, 9)$.

61. Functions 4

Implement a function that receives a vector of integers and an integer value, and returns a new vector obtained from the original one by removing all elements equal to the given value.

Example: Given the vector $(2, 6, 1, 8, 6, 2, 4)$, the vector without the elements equal to 6 is $(2, 1, 8, 2, 4)$.

62. Descriptive statistics

Write a descriptive statistics program that receives N positive real numbers. Create functions to compute and return the mean, the median, the standard deviation (dividing by $N - 1$), and the mode (if all the values entered were integers). **Note:** To compute the mode, you must use an auxiliary function that checks whether all the numbers entered are integers. If they are not, the function must return -1.

Example: When entering the sample $(1, 3, 7, 8, 9, 5, 4, 7, 6, 6, 7)$, the program should print the following output:

```
The mean of the entered values is 5.72727.
The median of the entered values is 6.
The standard deviation of the entered values is 2.3277.
The mode of the entered values is 7.
```

63. Functions and matrices

Perform the following steps:

1. Create a square matrix of order n of integer numbers, where the value of n is requested from the user.
2. Fill this matrix with values requested from the user.
3. Create a function with return type *void* that receives a matrix of integer numbers and prints it.
4. Create a boolean function that indicates whether a square matrix of integer numbers is symmetric or not.
5. Create n square submatrices of order k = *1, ..., n* so that the submatrix of order k consists of the first k rows and k columns of the original matrix. After creating each submatrix, use the previous functions to print them and check whether they are symmetric.

Example: When entering the matrix

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

the program should print the following output:

```
Matrix 1:
1
```

```
is symmetric

Matrix 2:
1 2
2 1
is symmetric

Matrix 3:
1 2 1
2 1 3
4 5 6
is asymmetric
```

64. Recursion

Repeat Exercises 53 (Power), 54 (Factorial), and 55 (Fibonacci) implementing recursive functions.

65. Combination

The combination of n choose k ($n \geq k$) can be computed recursively using the relation $C_k^n = C_k^{n-1} + C_{k-1}^{n-1}$. Write a recursive function that receives as arguments the value of n and the value of k and computes the combination C_k^n.

Example: $C_2^4 = 6$.

66. Determinant of a matrix

Create a recursive function that computes the determinant of a matrix. **Note:** Use Laplace expansion.

Example: The determinant of matrix $\begin{bmatrix} 2 & 3 & 2 \\ 5 & 6 & 5 \\ 7 & 8 & 9 \end{bmatrix}$ is -6.

Worksheet 5

67. Error handling - *throw*

Consider the code below. Protect the function *SquareR* against the introduction of negative arguments by throwing an exception and terminating the program whenever this happens.

```
#include <iostream>
#include <cmath>
using namespace std;
double SquareR (int n){
    return sqrt(n);
}

int main(){
    int n;
    cout << "Value of n: ";
    cin >> n;

    double x = SquareR(n);
    cout << "The square root of " << n << " is " << x << "\n";

    cout << "\nThe program has reached the end \n\n\n";
    return 0;
}
```

68. Error handling - *try...catch*

Consider the code from the previous exercise. Create a new exception for the case in which the user enters a non-numeric value when reading n in the *main* function. Use a *try...catch* block in the *main* function to handle this exception and the one from the previous exercise, displaying a specific error message for each case.

69. Error handling - class `runtime_error`

Protect the code from Exercise 68 against all possible errors using the `runtime_error` class and the *try...catch* block. Display specific error messages for each case.

70. Error handling - class `out_of_range`

Create a zero vector with 3 elements and then print the elements of that vector from positions 0 to 10. First use the `[]` operator and then use the `.at()` method to access the vector positions and observe what happens. What can you conclude?

71. Error handling

Consider the following code:

```
#include <iostream>
using namespace std;
double Division (int numerator, int denominator){
    return (double)(numerator)/denominator;
}

int main(){
    int num, den;
    cout<<"Enter the numerator and denominator values.\n";
    cin >> num >> den;
    double div = Division (num, den);
    cout<<"The result of dividing "<< num << " by "<< den << " is "<< div << ".\n";
    return 0;
}
```

Modify the program to handle all types of errors that may occur during its execution and print specific error messages for each type of error.

Worksheet 6

72. Modules: vectors

Create a module containing functions to manipulate vectors. Create two additional files in your project: a header file and a source file. The functions must be declared in the *header* file and defined in the *source* file.

1. Declare and define a function `X_Norm` that calculates the norm of an integer vector.
2. Declare and define a function `X_Print` that returns a string to print a vector of integer numbers in the format "(1,2,3)".
3. Declare and define a boolean function `X_Exist` that checks whether an element belongs to the vector.
4. Declare and define a function `X_Sort` that receives a vector of integer numbers and returns a vector sorted in ascending order. **Note:** You may not use the `sort` method.
5. Declare and define a function `X_InnerProduct` that computes the inner product of two integer vectors.
6. Declare and define a *void*-type function called `X_Extremize`, which receives a vector of integer numbers and modifies it so that each entry becomes equal to the element with the largest absolute value.

In the *main* function, create a vector of integer numbers and then call all the created functions appropriately. Example:

```
X = (2,7,5,3,11,1)
||X|| = 14.4568
u=5: Is in the vector
u=4: Is not in the vector
The sorted vector X is (1,2,3,5,7,11)
XY = (2,7,5,3,11,1)(1,-3,4,8,-1,9) = 23
The extreme vector of X is (11,11,11,11,11,11)
```

73. *Namespaces*

In the previous exercise, define a *namespace* to which the created functions should belong. Modify the function calls so that the *main* function recognizes them again.

74. Scope tree

Create a new header file and within it create a *namespace* `M_Math` that contains two *namespaces*, `It` and `Rec`. Inside the *It* namespace, declare a function called `Factorial` that calculates the factorial of a number iteratively. Inside the *Rec* namespace, declare another function also called `Factorial` that calculates the factorial of a number recursively. Define the declared functions appropriately in a source file and call them in the `main` function.

75. Module: matrices

Create a module with an appropriate *namespace* containing functions to manipulate square matrices of integers. This module should include a function that:

1. Reads the entries of a matrix and creates it.
2. Prints a matrix.
3. Calculates the trace of a matrix.
4. Performs the addition of two matrices.
5. Performs the subtraction of two matrices.
6. Performs the multiplication of two matrices.

In the *main* function, call all the created functions appropriately. Use a type alias to create an alias for the `vector<vector<int>` data type.

Worksheet 7

76. Classes: Complex

Create a class `Complex` that represents a complex number. The class must include:

i) Private attributes that make sense and public methods to modify and view these attributes;

ii) A default constructor;

iii) A constructor that allows defining the real and imaginary parts of a complex number;

iv) A method *Print* to print the complex number in the form $a + bi$;

v) A method *Modulus* to return the modulus of the complex number;

vi) A method *Negative* to return the additive inverse of the complex number;

vii) A method *Conjugate* to return the conjugate of the complex number;

viii) A boolean method *PureI* that checks whether the complex number is a pure imaginary number.

In the header file you created, declare (outside the class) functions that return the sum, subtraction, and multiplication of two complex numbers. All these functions must be defined in the existing `source` file. In the *main* function, test everything you implemented using the following instructions:

i) Create a complex number z_1 to represent $3 + 2i$ using the second constructor you defined, and print it on the screen using the *Print* method;

ii) Change the real part of z_1 to 7;

iii) Create a complex number z_2 using values entered by the user;

iv) Print the complex numbers z_1 and z_2 on the screen using the *Print* method;

v) Use the default constructor to create a new complex number z_3 where you should store the additive inverse of z_2;

vi) Create a new complex number z_4 that is the conjugate of z_3;

vii) Create a new complex number z_5 that corresponds to the subtraction of z_3 and z_4, and print it on the screen;

viii) Use the *PureI* method to check whether z_5 is a pure imaginary number.

77. Classes: Fraction

Create a class `Fraction` that represents a fraction. This class must be prepared to handle all types of errors that may occur, throwing exceptions with specific error messages whenever necessary. The class must:

1. Store the numerator and denominator of the fraction as private attributes (use `long long int` with `typedef` to simplify).
2. Have a default constructor and an appropriate parameterized constructor.
3. Have a public method that returns the fraction written as a string.
4. Have a private method that returns the greatest common divisor between the numerator and the denominator.
5. Have a public method that simplifies the fraction. This method modifies the fraction but does not print it.

Outside of the class, create one function that receives two fractions and returns their product, and another that returns their sum. In the `main` function, create two fractions and compute their sum and their product, displaying the simplified result.

78. Classes: Book

Create a class that represents a book. The class must have a default constructor and a parameterized constructor. It must have the following private members: title, author, and year of publication. You must create public members that allow modification of the private members and others that return them. The class must also have a public method that prints the book information in the following format:

```
Title: ...
Author: ...
Year: ...
```

In the `main` function, ask the user for the book's title, author, and year, and print the created book.

79. Classes: Polynomial

Create a class that represents a polynomial with integer coefficients. This class must be prepared to handle all possible errors by throwing exceptions when necessary. In this exercise, a *main* function is provided that must be used to test the class and verify the expected output. The only modification allowed in the *main* function is adding a *try..catch* block to handle thrown exceptions. The class must:

1. Store the degree and coefficients of the polynomial.
2. Have an empty default constructor, another that receives the degree of the polynomial and initializes all coefficients to zero, and another that receives the degree and a vector of coefficients.
3. Have methods to get and set a polynomial coefficient.
4. Have a method to print the polynomial.
5. Have a method to calculate the value of the polynomial for a given x.
6. Have a method to change the polynomial's degree. If the new degree is lower than the previous one, the polynomial must be truncated; otherwise, the new coefficients should be requested from the user.

Outside the class, create a function that adds two polynomials.

The program must work with the following `main` function:

```
int main(){
    Polynomial p1(4);
    for(int i=0; i<=4; i++)
        p1.SetCoef(i,1);
    cout << "p1 = "<< p1.Print();

    Polynomial p2(5, {1,-2,0,4,-1, 3});
    cout << "\np2 = "<<p2.Print();
    cout << "\nThe value of p1 at x = 2 is " << p1.Value(2) <<endl;

    p1.ChangeDegree(2);
    cout << "Decrease degree of p1 from 4 to 2\n  p1 = " << p1.Print();

    cout << "\nIncrease degree of p1 from 2 to 5\n";
    p1.ChangeDegree(5);
    cout << "\n p1 = " << p1.Print() << "\n\n";

    Polynomial p3(2, {1,1,1});
    Polynomial p4(4, {2,2,2,2,2});
    Polynomial p5 = Sum(p4, p3);
    cout << "Sum: \n( "<< p3.Print() <<" ) + ( " << p4.Print() << " ) ";
    cout << "=  " << p5.Print() << "\n\n";
    return 0;
}
```

The expected output is:

```
p1 = 1 + x + x^2 + x^3 + x^4
p2 = 1 - 2x + 4x^3 - x^4 + 3x^5
The value of p1 at x = 2 is 31
Decrease degree of p1 from 4 to 2
p1 = 1 + x + x^2
Increase degree of p1 from 2 to 5
v[3] = 6
v[4] = 5
v[5] = 8

p1 = 1 + x + x^2 + 6x^3 + 5x^4 + 8x^5

Sum:
( 1 + x + x^2 ) + ( 2 + 2x + 2x^2 + 2x^3 + 2x^4 ) =  3 + 3x + 3x^2 + 2x^3 + 2x^4
```

80. Classes: Rectangle

Create a class that represents a rectangle. This class must handle all possible errors and must include:

1. Four private members: height, width, and two variables representing the coordinates of the rectangle's top-left corner.
2. A default constructor that assigns the dimensions of a 1×1 square and another that accepts four arguments to initialize the private members.
3. Functions to modify the rectangle's dimensions.
4. A function that returns the area and another that returns the perimeter of the rectangle.
5. A function to display the rectangle on the screen. Assume the screen is a 40×20 character grid, where one horizontal unit corresponds to two characters and one vertical unit to one *new line*.

Implement the `main` function to produce the following output. Lines starting with `>>` represent user input and do not need to appear exactly as shown.

```
The rectangle from the default constructor is:
 __
|__|

> > Height and width of the rectangle: height = 2, width = 4
> > Coordinates (x,y) of the top-left corner of the rectangle: (2, 2)
> > The perimeter of the entered rectangle is 12 and the area is 8.
> > The rectangle entered is:

  ________
 |        |
 |________|
```

81. Classes: Points in $\mathbb{R}^2$

Create the class `PointR2` to represent a point $(x, y) \in \mathbb{R}^2$. Create a default constructor initializing the point as the origin $(0, 0)$, and another that accepts the point's coordinates. The class must have public member functions that provide access to its private members. Additionally, include functions to:

1. Print a point in the format (x, y) (the function must return a string);
2. Determine the Euclidean distance from the point to the origin $(0, 0)$;
3. Determine the Euclidean distance between the point and another point;
4. Rotate the point by α degrees about the origin. **Note:** Rotating (x, y) by α degrees produces $(x \times \cos\alpha - y \times \sin\alpha, y \times \cos\alpha + x \times \sin\alpha)$;
5. Check whether the point is collinear with another point. **Note:** (x_1, y_1) is collinear with (x_2, y_2) if and only if $x_1 \times y_2 = x_2 \times y_1$.

Implement the `main` function to produce the following output. The line starting with `>>` represents user input.

```
The Euclidean distance from the point (1.000000 , 2.000000) to the origin is 2.23607
>> Enter coordinates for a point: (2 , 5)
The distance from the point (1.000000 , 2.000000) to the point (2.000000 , 5.000000) is 3.16228
The point (1.000000 , 2.000000) rotated 90 degrees is (-2.000000 , 1.000000)
The points (-2.000000 , 1.000000) and (2.000000 , 5.000000) are not collinear
```

82. Classes: PL Championship

Implement the classes described below. In addition to the explicitly requested methods, you must also implement a print method in each class. You may implement additional methods if needed.

1. Create a class `Player` with the following private members: the player's name, the player's number in the Players' Union (a unique identifier), the player's physical fitness (an integer between 1 and 100), and the player's talent (an integer between 1 and 100). Implement a function that computes the player's aptitude using the formula:

   $$0.60 \times \text{physical fitness} + 0.40 \times \text{talent}.$$

   In the `Print` method, you must use the manipulators `left` and `setw(·)` from the `iomanip` header to print the player's name left-aligned in a field of width 9.

2. Create a class `Team` with the following private members: the team's name, the sport's name, and a `vector` of `Player` objects. Implement a constructor that receives the team name and the sport name and public access functions for the private members. Also implement functions to add a player to the team and to remove a player from the team.

3. Create a class `Game` with private members representing the two teams that will play. Implement a function that determines the winner of the game, which is the team in which players have the higher average aptitude. If the two average aptitudes are equal, generate a random binary number $r \in \{0, 1\}$ using `r = rand() % 2`. If $r = 0$, the winner is the first team; otherwise, it is the second team.

4. Create a class `League` with the following private members: the league's name, a `vector` of `Team` objects, and a `vector` of `Game` objects.

   The constructor must receive only the league's name. Implement a function to add teams to the league, and another function to create the league's games, assuming that each team plays exactly once against every other team.

   Finally, implement a function that determines the league winner: the team that wins the most games.

The implemented classes must work with the following *main* function.

```
int main(){
    Player j1("Joao", 1, 56, 90); Player j2("Manuel", 2, 78, 20);
    Player j3("Maria", 3, 82, 63); Player j4("Joana", 4, 23, 20);
    Player j5("Rita", 5, 98, 68); Player j6("Francisca", 6, 25, 99);
    Player j7("Caetana", 7, 69, 64); Player j8("Cristovao", 8, 48, 87);

    Team e1 = Team("Team_1", "-");
    e1.AddPlayer(j8); e1.AddPlayer(j7); e1.AddPlayer(j2);
    e1.RemovePlayer(j8);
    Team e2 = Team("Team_2", "-");
    e2.AddPlayer(j8); e2.AddPlayer(j4);
    Team e3 = Team("Team_3", "-");
    e3.AddPlayer(j5); e3.AddPlayer(j6);
    Team e4 = Team("Team_4", "-");
    e4.AddPlayer(j1); e4.AddPlayer(j3);

    League l = League("LeaguePL");
    l.AddTeam(e1); l.AddTeam(e2);
    l.AddTeam(e3); l.AddTeam(e4);
    l.PlayLeague();
    l.Print();
    auto leagueWinner = l.ComputeWinner();
    cout << "The winner of league " << l.GetName() << " is "  << leagueWinner.GetName() << endl;
    return 0;
}
```

The expected output of the program is the following:

```
####################
#       TEAMS      #
####################
Name: Team_1
Cristovao ID: 8 F: 48 T: 87 Apt: 63.6
Caetana   ID: 7 F: 69 T: 64 Apt: 67
Manuel    ID: 2 F: 78 T: 20 Apt: 54.8

Name: Team_2
Cristovao ID: 8 F: 48 T: 87 Apt: 63.6
Joana     ID: 4 F: 23 T: 20 Apt: 21.8

Name: Team_3
Rita      ID: 5 F: 98 T: 68 Apt: 86
Francisca ID: 6 F: 25 T: 99 Apt: 54.6

Name: Team_4
Joao      ID: 1 F: 56 T: 90 Apt: 69.6
Maria     ID: 3 F: 82 T: 63 Apt: 74.4
```

```
####################
#       GAMES      #
####################
Team_1 vs Team_2 -> Team_1 wins
Team_1 vs Team_3 -> Team_3 wins
Team_1 vs Team_4 -> Team_4 wins
Team_2 vs Team_3 -> Team_3 wins
Team_2 vs Team_4 -> Team_4 wins
Team_3 vs Team_4 -> Team_4 wins

The winner of league LeaguePL is Team_4
```

83. Classes: Tic-Tac-Toe

The goal is to implement class `TicTacToe` to play Tic-Tac-Toe. This class must have two attributes: a matrix of `char`, which represents the game board, and a `char`, which indicates the symbol of the next player to play (X or O).

1. Create a constructor that receives the symbol of the first player to play and initializes the game board with the `char` -.
2. Create a method `GetPlayer` that allows access to the player's symbol.
3. Create a *void* method `PrintBoard` that prints the game board to the screen in the format below. **Note:** In the example the empty board is shown.

```
 - | - | -
-----------
 - | - | -
-----------
 - | - | -
```

4. Create a method `MakeMove` with return type void that receives the row and column where the symbol should be placed and that changes the player's symbol.
5. Create a method `GetWinner` that returns the symbol of the winning player, the symbol E in case of a draw, and the symbol - if the game has not yet ended.

Fill in the *main* function shown below with the appropriate method names, replacing the `(...)` so that you can play Tic-Tac-Toe. Whenever the user enters information that is not valid, ask for the information again until a valid value is given.

```
int main(){
    char symbol;
    cout << "Insert the first player to play: ";
    cin >> symbol;
    TicTacToe game = (...);
    int r, c;
    while (true) {
        cout << "Current board:" << endl;
        game.(...);
        cout << "Turn of Player " <<  game.GetPlayer()
             << ". Insert row and column (0-2): ";
        cin >> r >> c;
        game.(...);
        char res = game.(...);
        if (res == 'X' || res == 'O'){
            cout << "Player " << res << " wins!" << endl; break;
        }
        else if (res == 'E'){
            cout << "It is a draw!" << endl; break;
        }
    }
    cout << "Final board:" << endl;
    game.(...);
    return 0;
}
```

Worksheet 8

84. Operator Overloading: Complex

In the same files created for the implementation of the `Complex` class, define the following operators:

i) Binary operators for the addition, subtraction, and multiplication of complex numbers;

ii) A unary operator for the additive inverse (symmetric) of a complex number;

iii) `ostream` and `istream` operators to write and read a complex number, respectively. Use format "a+bi".

Test the operators you implemented with the following *main* function:

```
int main(){
    Complex z1(2,3);
    Complex z2;
    cout << "Enter a complex number: \n";
    cin >> z2;
    cout << "The entered complex number was: " << z2 << endl;
    Complex zSimetric = -z2;
    cout << "The additive inverse of the entered complex number is: " << zSimetric << endl;
    Complex zSum = z1+z2;
    Complex zSubtraction = z1-z2;
    Complex zProduct= z1*z2;
    cout << "Sum: " << zSum << endl;
    cout << "Subtraction: " << zSubtraction << endl;
    cout << "Product: " << zProduct << endl;
    return 0;
}
```

85. Operator Overloading: Polynomial

In the same files created for the implementation of the `Polynomial` class, define the following operators:

i) Binary operators for the addition and subtraction of polynomials;

ii) Unary operators `[ ]` to access and modify the polynomial coefficients;

iii) A unary operator `()` that receives a scalar and computes the value of the polynomial at that scalar;

iv) An `ostream` operator to write a polynomial. Use the format "c_0 + c_1x + ... + c_nx∧n".

Adapt the `main` function from Exercise 80 to use the overloaded operators.

86. Operator Overloading: Fractions

In the same files created for the implementation of the `Fraction` class, define the following operators:

i) Binary operators for the addition, subtraction, multiplication, and division of fractions;

ii) A unary operator for the additive inverse (negation) of a fraction;

iii) `ostream` and `istream` operators to write and read a fraction, respectively. Use the format "numerator/denominator".

87. Operator Overloading:: Date

In this exercise, you will create a class to manipulate dates. The class, called `Date`, must have the day, month, and year as private attributes. The class must also have:

i) A default constructor and another user-defined constructor that validates the given date (it must keep asking the user for dates until a valid date is entered). Create a private method to validate a date;

ii) Public methods that allow viewing the day, month, and year;

iii) Unary operators `++`, which increments the date by one day, and `--`, which decrements the date by one day. Implement only the prefix or the postfix versions, as required by the *main* function below;

iv) `ostream` and `istream` operators to write and read a date, respectively. Use the format "day/month/year" to represent a date.

The operators you implemented must work with the following `main` function:

```
int main(){
    Date d;
    cout << "Enter a date in the format d/m/y:"; cin >> d;
    cout << "\nDate: " << d;
    ++d; ++d;
    cout << "\nDate + 2 days: " << d;
    d--; d--; d--;
    cout << "\nDate - 1 day: " << d;
    return 0;
}
```

88. Operator Overloading: Points in $\mathbb{R}^2$

In the same files created for the implementation of the `PointR2` class, define the following operators:

i) Binary operators for the addition and subtraction of points in $\mathbb{R}^2$;

ii) A binary operator `*` that returns the inner product of two points in $\mathbb{R}^2$;

iii) A unary operator for the additive inverse (negation) of a point in $\mathbb{R}^2$;

iv) `ostream` and `istream` operators to write and read a point, respectively. Use the format "(x, y)" to represent a point in $\mathbb{R}^2$.

89. Operator Overloading: Matrix

Implement a class `Matrix` with private attributes for the number of rows, the number of columns, and the matrix (use as data type a `vector<vector<int>>`).

i) Define a default constructor and another constructor that receives the number of rows and columns of the matrix, as well as the matrix elements in the form of a vector of vectors;

ii) Create the `[ ]` operator to access the elements of the matrix;

iii) Create the `ostream` operator to print the matrix;

iv) Create the operators for matrix addition, subtraction, and matrix multiplication. The operators must recognize the matrix dimensions and therefore verify whether the operations are possible.

Worksheet 9

90. Inheritance and Polymorphism: Buildings

Complete the following steps:

i. Create a base class named `Building` with the following attributes: address, area (in square meters), and floors (number of floors). The class must have a single constructor receiving the required arguments and a method that prints the building information.

ii. Create two derived classes, `House` and `Office`, which inherit publicly from `Building`.

- `House` must have the private attributes number of bedrooms and bathrooms.
- `Office` must have the private attribute number of cubicles.

iii. In each derived class, implement a method that prints the specific information of that class, and the general information of any building. Avoid code repetition in the *header* file(s). For example, call the base-class printing method from the derived-class printing method.

iv. In the `Building` class, declare a pure virtual method named `Evaluate()` that returns the building's value. Implement this method in the derived classes using the evaluation functions below:

Type	Evaluation function
House	$20 \times (\#\text{bedrooms}) + 5 \times (\#\text{bathrooms}) + 10 \times (\#\text{floors})$
Office	$10 \times (\#\text{cubicles}) + 10 \times (\#\text{floors})$

v. Create another class `SingleStory` that inherits publicly from `House` and represents single-storey houses (ground floor only, i.e., no floors above ground). This class must have a single constructor that takes **at most four arguments**.

vi. In function `main()`, create one object of each type: `House`, `Office`, and `SingleStory`. For each object, print all its information, including its evaluation value.

91. Inheritance and Polymorphism: Person

Create four classes that represent people who attend ISEG. The base class, `Person`, should have the following attributes: a name and a date of birth (represented by three integers). In addition, it must include:

- a public method that receives a date (three integers) and computes the person's age on that date;
- a public method that prints the information relates to a person (name and date of birth);
- a public pure virtual method called `Profession()`. This method must be implemented in derived classes and should return the person's profession.

The derived classes must be: `Student`, `Professor`, and `Employee`.

- The class `Student` must have the program name and the average grade as private attributes.
- The class `Professor` must have the following private attributes: category and years of service.
- The class `Employee` must have the following private attributes: department name and years of service.

Each class should have appropriate constructors.

Create a function that takes an object of type `Person` as an argument. This function must call function `Profession()` to demonstrate polymorphism.

92. Inheritance and Polymorphism: Vehicle

Complete the following steps:

i. Create a base class `Vehicle` to store information common to all types of vehicles (brand, year of manufacture, and license plate), along with an appropriate constructor to initialize objects of this class. Create a public method that prints general information about a vehicle following the example below:

```
General Information:
Brand: Opel
Year: 1996
License plate: 12-29-JE
```

ii. Create a class `HeavyVehicle` derived publicly from `Vehicle`. Add attributes specific to `HeavyVehicle`, namely maximum load and whether it is refrigerated or not. Create an appropriate constructor for `HeavyVehicle`.

iii. Create another base class to represent the insurance of a vehicle; call it `Insurance`. This class must contain the attributes insurance validity and insurance cost/value. The validity must be an object of the `Date` class created in Worksheet 8, and the class must have an appropriate constructor. Create a public method that prints the insurance information of a vehicle following the example below:

```
Insurance Information:
Validity: 20/2/2029
Value: 67
```

iv. Modify the class `HeavyVehicle` so that it also inherits from the class `Insurance`.

v. Create a class `MotorVehicle` derived publicly from `Vehicle` and `Insurance`. This class must have a single attribute: the number of wheels. Create an appropriate constructor for this class.

vi. Create a public pure virtual method called `VehicleType()` in the class `Vehicle`. This method must be implemented in the derived classes and should print to the screen the type of vehicle.

vii. Create a global function `PrintType` in the file *main.cpp* that receives an object of type `Vehicle` and prints the general information about that vehicle as well as the specific vehicle type.

viii. In the class `Vehicle`, implement a private method that validates a license plate (use the format AA-00-AA). Useful commands: `.size()` (string length), `.at(i)` (character at position i), `isdigit(c)` (checks whether character `c` is a digit between 0 and 9), and `isupper(c)` (checks whether `c` is an uppercase letter between A and Z). Modify the constructor of `Vehicle` so that it verifies whether the received license plate is valid; if it is not, it must repeatedly ask the user for a new license plate until a valid one is entered.

Worksheet 10

93. Writing files - `ofstream`

Create a program that writes information about a class to a file named "Class.txt". Start by asking the user for the number of students in the class. Then ask the user for each student's name and their three grades, and write them to the file according to the format shown below.

Example file format:

```
John/12/14/15
Mary/13/20/7
...
```

94. Reading files - `ifstream`

Manually create a file "Grades.txt" containing the final grades for a course. Read those grades into your program, storing them in a vector. Then compute the average and print it on the screen.

File data: 12 15 9 18 15 10 9 8 19

95. Playing with files

Create a program that reads a list of integer values from a file called "input.txt" and:

1. Prints the highest value to the screen.
2. Saves the squared values into a new file called "output1.txt".
3. Removes duplicate values and writes the list without duplicates to a new file called "output2.txt".
4. Prints to the screen the longest increasing sequence of values found in the file"output2.txt".

Example:

Content in file "input.txt": 7 2 9 3 4 5 8 7 2 3

Content on screen: 9

Content in file "output1.txt": 49 4 81 9 16 25 64 49 4 9

Content in file "output2.txt": 7 2 9 3 4 5 8

Content on screen: 3 4 5 8

96. New grading scale

Create a program that reads a list of grades (0-20) from a file "StudentGrades.txt". Each line of the file contains a student's name and grade (separated by a space).

1. Assign a classification to each student based on the following scale: Fail (0-9), Pass (10-12), Good (13-15), Very Good (16-18), Excellent (19-20). The program must save the grades in the new scale into a new file called "NewScaleGrades.txt" in the format *"Name Grade"*.
2. Write the 10 highest grades into a new file called "TopGrades.txt"
3. Write the grade histogram into a new file called "Histogram.txt". The histogram should show the number of students who received each classification.

97. Writing files - `ofstream, ifstream, ios_base::out/app, getline`

Create a program that asks the user for a sentence and writes it to a text file. If the file already exists, the program must ask the user whether the user wants to delete the contents of the existing file or append information to it.

98. Alternative to `to_string`: `ostringstream, .str()`

Create a program that asks the user for two real numbers representing the numerator (n) and denominator (d) of a fraction. Then create a `string` to store the expression "`n/d=r`" using the `to_string` function and print that `string`. Then create a new `string` with the same content using an `ostringstream` and print it to the screen. Compare the results.

99. Word counting: `istringstream`

Write a program that reads a sentence entered by the user and stores it in an object of type `string`. Then, using an `istringstream`, determine the number of words in the sentence.

100. Numbering words: `ostringstream`, `istringstream`

Write a program that reads a sentence from the console and creates a `string` with the same sentence where at the end of each word the numerical order of that word appears.

Input: `This is a sentence that has lots of blank spaces`

Output: `This[1] is[2] a[3] sentence[4] that[5] has[6] lots[7] of[8] blank[9] spaces[10]`

101. Futsal teams

Manually create the "Futsal.txt" file containing the ages of players from three futsal teams in the following format:

```
AFT  23  18  19  22  29  30  25  24
WET  20  31  19  27  35  30  27  20
FCR  17  32  22  22  35  37  21  24
```

Write a program that reads the information from this file and prints on the screen the age of the oldest and youngest player on each team.

102. Grades from the Class file

Create a program that reads information about a class (use the file "Class.txt" created in Exercise 94). It must print on the screen the name and the average of each student with two decimal places (use the `setprecision` manipulator available in the `iomanip` library).

Example:

```
John: 18
Mary: 18
```

103. Frequency of each word

Create a program that asks the user for a sentence and indicates how many times each word appears in the sentence.

104. Delete line

Write a program that allows deleting a line from a file. The program must ask the user which line number they want to delete.

105. BMI calculation

Manually create the file "'Info.txt" with the following format "`Number. Name_Age Weight Height`". Then write a program that reads that file and: (i) indicates on the screen which people are under 23 years old; and (ii) writes a file named "BMI.txt" with the person's name and their body mass index (in the format `Number. Name BMI`).

```
1. Maria_25 56 1.75
2. Joao_22 60 1.60
3. Lara_30 52 1.57
4. Pedro_21 80 1.85
...
```

106. Sentence with the most words

Write a program that reads a list of sentences from a file called "Sentences.txt" and prints to the screen the sentence with the most words.

107. Reverse order

Write a program that reads a list of words from a file called "Words.txt" and writes them in reverse order into a file called "ReverseWords.txt"

108. Sorting dates

Write a program that reads a list of dates in the format `DD/MM/YYYY` from a file called "Dates.txt" and writes them in chronological order into a file called "DatesCronologicalOrder.txt".

109. Interpreting mathematical expressions

Write a program that repeatedly asks the user for mathematical expressions and returns their result. The expressions must involve only the basic operations (`-`, `+` and `*`), cannot contain parentheses, and each operator must have a space before and after it. The program ends when the user enters the word "STOP".

Example:

```
-> 2 + 4 * 2 - 7
Result: 3

-> -5 + 10 * 2
Result: 15

-> 2 + 3 + 2 * 2 - 5 - 2 * 6 - 3 - 2 * 5
Result: -21

-> STOP
```