1. Data types. Arithmetic expressions. Input and output.

1.1

Write a program that reads a letter or other symbol from the keyboard and shows its ASCII code.

1.2

Write a program that reads 3 integer numbers from the keyboard and shows on the screen their mean value and the difference between each number and the mean value. The dialog with the user must be similar to the following:

```
Please, input 3 integer numbers
A ? 23
B ? 47
C ? 30
mean   = 33.333
A-mean = -10.333
B-mean = 13.667
C-mean = -3.333
```

Implement different versions of the program: using different types for the variables used to store **A**, **B**, **C** and **mean**. Interpret the results.

1.3

The mass of a sphere is given by the expression **M = 4/3(ρπr³)** where **M**, ρ and **r** are, respectively, the mass of the sphere, the specific mass of the material from which it is made, and its radius. Write a program that, given the values of ρ and **r**, determines the value of **M**. The user must be informed about the units used to represent all the values: Kg/m³, m and Kg for ρ, **r** and **M**, respectively. Use a constant to represent the value of π.

1.4

The solution to a system of linear equations in two variables (**x** e **y**)

a·x + b·y = c
d·x + e·y = f

is given by

x = (c·e - b·f) / (a·e - b·d)
y = (a·f - c·d) / (a·e - b·d)

Write a program that reads the values of **a**, **b**, **c**, **d**, **e** e **f** and determines the solution of the corresponding system of equations. Consider only the cases when there is a solution (it is not an impossible or inconsistent system).

1.5

a) Write a program that reads two times, expressed in hours, minutes and seconds, and determines their sum. The dialog with the user must be similar to the following:

```
Time1 (hours minutes seconds) ? 10 35 50
Time2 (hours minutes seconds) ? 15 59 30
Time1 + Time2  =  1 day, 2 hours, 35 minutes and 20 seconds
```

b) Modify the program so that the user must input a separator between hours, minutes and seconds (ex: 10:35:50). Although the separator usually used is '**:**', consider that any separator is valid.

1.6

The area of a triangle can be determined using the Heron's formula: $area = \sqrt{s(s-a)(s-b)(s-c)}$ where *s*, *a*, *b* and *c* are, respectively, the semi-perimeter and the length of the 3 sides. Write a program that reads the coordinates of the 3 vertices of a triangle and calculates the area of the triangle, using that formula. Remember that the distance between 2 points whose coordinates are (*x1*,*y1*) e (*x2*,*y2*) is given by $d = \sqrt{(x2-x1)^2 + (y2-y1)^2}$.

2. Control structures: selection and repetition.

2.1.

Solve again problem **1.4** (solution of a system of linear equations in two variables), so that when the system is impossible or inconsistent (a system having infinite solutions) a message is shown to the user: "impossible system" or "inconsistent system".

2.2.

a) Write a program that reads 3 numbers from the keyboard and determines the largest and the smallest number.

b) Write a program that reads 3 numbers from the keyboard and writes them on the screen, in descending order.

c) Write a program that reads 3 positive numbers from the keyboard and determines if they can represent the length of the 3 sides of a triangle (tip: it is not possible to build a triangle if the sum of the 2 smallest lengths is smaller than the largest length). If any of the numbers is not positive the program must show an error message.

2.3.

Write a program that reads 2 integer numbers from the keyboard and tests whether their sum would produce overflow (the result would be greater than **INT_MAX**) or underflow (the result would be lower than **INT_MIN**). If this happens the program must show the message "sum overflow" or "sum underflow", otherwise it should show the result of the sum.

2.4.

The cost of transporting a certain merchandise is determined, depending on its weight, as follows: if the weight is less or equal to 500 grams the cost is 5 euros; if the weight is between 501 grams and 1000 grams, inclusive, the cost is equal to 5 euros plus 1.5 euros for each additional 100 grams or fraction above 500 grams; if the weight exceeds 1000 grams, the cost is 12.5 euros plus 5 euros for each additional 250 grams or fraction above 1000 grams. Write a program that, given the weight of a certain merchandise, determines the cost of its transportation.

2.5.

Write a program to determine the roots of a quadratic equation $Ax^2+Bx+C=0$, the coefficients ***A***, ***B*** and ***C*** being provided by the user. The program must indicate whether the equation has 2 different real roots, 2 equal real roots or 2 complex roots, and the respective root values , with 3 decimal places.

Example:

```
Solution of Ax^2 + Bx + C = 0
Insert the coefficients (A B C): 2.5 -1 16
The equation has 2 complex roots: 0.200+2.522i  and  0.200-2.522i
```

2.6.

Write a program to determine and write the amount that a depositor can withdraw from the bank, after **n** years of depositing an amount **q**, where **j**% is the annual interest rate. The values of **n**, **q** and **j** must be specified by the user. Assume that interest at the end of each year is accrued to the deposited amount.

2.7.

A number **n** is prime if it is divisible only by itself and by one.

a) Write a program that reads a number from the keyboard and determines if it is prime. Note: it is not necessary to test all divisors in the range **[2..n[**; it is enough to test divisors until the integer part of the square root of **n**.

b) Write a program that writes on the screen all the prime numbers lower than 1000.

c) Write a program that writes on the screen the first 100 prime numbers.

d) Write a program that determines the largest prime number that can be stored in a varible of type **unsigned long**.

2.8.

a) Write a program that displays on the screen a table of sines, cosines and tangents of the angles in the range [0..90] degrees, with intervals of 15 degrees, as shown below (note the particular case of the last line, corresponding to the 90 degree angle).

ang	sin	cos	tan
0	0.000000	1.000000	0.000000
15	0.258819	0.965926	0.267949
30	0.500000	0.866025	0.577350
45	0.707107	0.707107	1.000000
60	0.866025	0.500000	1.732051
75	0.965926	0.258819	3.732051
90	1.000000	0.000000	infinite

b) Change the program you developed in **a)** so that the range limits and the interval of the value of the angles in the table can be specified by the user (for example, if the range is [0..1] and the increment is of 0.1 degrees, the table for the angles of 0, 0.1, 0.2,…, and 1 degree should be displayed).

2.9.

A palindrome is a word, number, phrase, or other sequence of characters which reads the same backward as forward. For example, the following numbers are palindromes: 12321, 555, 45554 and 11611.

a) Write a program that reads a 3-digit integer and determines whether or not it is a palindrome (suggestion: use the division and module operators to separate the integer into its digits.

b) Generalize the program in **a)** in order to treat unsigned integers with a greater number of digits. Note: do not use *arrays* or *vectors* to store the digits.

2.10.

Write a program that reads an integer and breaks it down into prime factors (example: 20 = 2x2x5).

Notes:

- one way to solve this problem would be to start by dividing the number by the first prime number, 2, and continue dividing by 2 until you get non-zero remainder; then divide by the other prime numbers, 3, 5, 7, etc. until the only numbers left are prime numbers;
- an alternative way is to start by dividing the number by 2 and continue dividing by 2 until you get non-zero remainder; then do the same for 3, 4, 5, 6, 7, etc. until the dividend is equal to 1.

2.11.

Write a program to calculate the sum of the first **n** terms (**n** being input by the user) for each of the following series.

a) Series giving the value of the mathematical constant π:

$$4-\frac{4}{3}+\frac{4}{5}-\frac{4}{7}+\frac{4}{9}-\frac{4}{11}+\ldots$$

b) Series giving the value of the mathematical constant *e* :

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots$$

(Tip: Calculate each term from the immediately preceding term)

c) Series giving the value of e^{-x} (com *x* real positivo previamente definido):

$$1-\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\ldots$$

(Tip: Calculate each term from the immediately preceding term. Note: **x** must also be input by the user.)

2.12.

Repeat problem **2.11** so that the user can specify the precision with which he wants the result, that is, the maximum variation between the value of the sum of the series, between two consecutive iterations. Note that the variation can be either positive or negative.

2.13.

Write a program that reads a sequence of integer numbers, and determines and writes the sum, the mean, the standard deviation, the smallest and the largest of the numbers, in the following situations:

a) the length of the sequence is previously indicated by the user;

b) the end of the sequence is indicated by the value 0 (which is not considered to be part of the sequence); at the end, the program must indicate the length of the sequence;

c) the end of the sequence is indicated when the user types end of input (**CTRL-Z** in Windows; **CTRL-D** in Linux).

2.14.

a) The square root of a number **n** can be calculated in an approximate way using the following algorithm, due to Heron of Alexandria:
- start from an initial estimate of the root value, **rq**;
- calculate a new estimate, **rqn**, using the formula: **rqn = (rq + n / rq) / 2**;
- repeat this calculation using the **rqn** value as a new estimate of **rq**.

The following table illustrates the evolution of the square root calculation for the number **n** = 20, based on an initial estimate **rq** = 1.

rq	rqn	rqn^2	dif = n - rqn^2
1	10.500000	110.250000	-90.250000
10.50000	6.202381	38.469532	-18.469532
6.202381	4.713475	22.216844	-2.216844
4.713475	4.478314	20.055300	-0.055300
4.478314	4.472140	20.000001	-0.000039

In general, this algorithm converges quickly to a correct solution, even when the initial estimate is poor. Note that the difference **dif = n - rqn^2**, quickly evolves to zero, so one could use the value of **dif** as a stop criterion, that is, end the iterations when **dif** is less than a small number, **delta** (for example, **delta** = 0.00001). However, it is advisable to limit the number of iterations, repeating, in this case, the calculations until a value of **dif** less than **delta** is reached or a maximum number of iterations, **nMaxIter**, is reached.
Considering the presented example, if **delta** = 0.001 and **nMaxIter** = 3 the final result (**rqn**) would be the one calculated after the 3rd iteration because the maximum number of iterations specified has been reached, although the **delta** value is still greater than 0.001; but if **delta** = 0.001 and **nMaxIter** = 10, the calculation would end after the 5^{th} iteration because in this iteration the absolute value of **dif** is less than 0.001.
Write a program that reads the values of **n**, **delta** and **nMaxIter** and that calculates the square root of **n** using the algorithm described previously. Always use 1 as initial estimate of **rq**.

b) Modify the program in **a)** in order to present the result with the same number of decimal places used in specifying the **delta** value; for that you should find an algorithm to determine the number of decimal places. Also show the value returned by the **sqrt()** function from the C library and compare the values returned by this function and Heron's algorithm.

2.15.

Write a program to test if the user knows the multiplication tables. The program should generate 2 random numbers (between 2 and 9), present them to the user, and ask for the result of multiplying those numbers. After reading the user's answer, it should present an appropriate message taking into account the correctness of the answer and the time it took the user to give it: if the answer is wrong, the user should receive the message "Very Bad", if the answer is correct and took less than 5 seconds, the user should receive the message "Good", if you answer is correct but took between 5 seconds and 10 seconds (inclusive) the user should receive the message "Satisfactory", otherwise the user should receive the message "Insufficient".

2.16.

Write a program to emulate the operation of a basic calculator. In addition to performing the 4 fundamental algebraic operations, addition, subtraction, multiplication and division, the calculator must have a memory where the value shown on the display can be saved, using the **M** command. It must also be possible to clear the contents of the memory, using the **MC** command, add or subtract the current value on the display to the memory, using the **M+** and **M–** commands, or show the contents of the memory on the display using the **MR** command. The command to clear the contents of the display is **C**.

3. Functions.

3.1.

Rewrite the program of problem **1.6**, for calculating the area of a triangle using the Heron formula, in order to use the following functions:

- **double area(double x1, double y1, double x2, double y2, double x3, double y3)** to calculate the area of a triangle whose vertices have coordinates (**x1**,**y1**), (**x2**,**y2**) and (**x3**,**y3**);
- **double distance(double x1, double y1, double x2, double y2)** to calculate the distance between two points whose coordinates are (**x1**,**y1**) and (**x2**,**y2**).

3.2.

Rewrite the programs of problem **2.7**, for determining prime numbers, in order to use a function **isPrime()**, that determines whether the number that it receives as parameter is prime or not. Choose suitable types for the parameter and the return value of the function.

3.3.

Rewrite the program of problem **2.14** in order to use a function to calculate the square root, using the method indicated in that problem. Note that this function must have 3 parameters: the value whose square root is to be calculated, the precision and the maximum number of iterations.

3.4.

In the C/C++ libraries, there is no function to round a decimal number to a specified number of decimal places. However, it is possible to achieve this using the **floor()** function. For example, to round the value of **x** to 2 decimal places), you can use the following operation:

floor(x * 100 + 0.5) / 100

Write a function whose prototype is

double round(double x, unsigned n)

that rounds a floating point number, **x**, to a given number of decimal places, **n**, returning the rounded value.
Develop a program for testing the function; it must asks the user for the values of **x** and **n** and show the result of **round(x,n)**.

3.5.

Euclid's iterative algorithm for determining the greatest common divisor (GCD) of two numbers is based on the following findings:
- if the two numbers are equal, the GCD is given by the value of either one;
- if one of the numbers is zero, the GCD is the other number;
- the GCD of two numbers is not changed if the smaller of the two numbers is subtracted from the larger; by repeatedly applying this rule until the two numbers are equal, the GCD is obtained.

Write a function which returns the GCD of two numbers which it receives as parameters. This function does not write anything on the screen. Implement two different versions of the function, having different prototypes: one in which the GCD is the return value of the function; another in which the GCD is returned through a parameter of the function.

3.6.

Write a function whose prototype is **time_t timeElapsed()** that returns the time (in seconds) that has elapsed since the first time it was called. For example, if the function is called 3 times, at 10:59:55, at 11:00:25 and at 11:00:45, the returned values should be, respectively, 0, 30 and 50. Suggestion: use a local static variable to keep the time of the first call.

3.7.

Write a function **bool readInt(int &x)** that tries to read a valid integer number from the keyboard. If the input is a valid number, it must be returned through parameter **x**, and the return value of the function must be **true**. If the input is invalid, the function must return **false**; in this case the value of **x** has no significance. In both cases, the input buffer must be cleaned before the function returns. Note: if the input contains other characters beyond those that make up a single integer number (ex: **122a** or **123 45**), it must be considered invalid.

3.8.

The C/C++ language does not have the type "fraction" nor, obviously, operators or functions to manipulate fractions. You must develop a set of functions that allow the manipulation of fractions, represented by independent variables that represent the numerator and denominator of each fraction (later you will use a **struct** to represent a fraction).

a) Write a function whose prototype is

bool readFraction(int &numerator, int &denominator)

that reads a fraction, written in the format **numerator/denominator** (ex: **2/3** or **5/12**). The return value of the function must be **true** if the values entered for the numerator and the denominator are valid, and the separator is '**/**', or false otherwise. In the latter case, the values returned, for the **numerator** and **denominator**, must be zero. Note: the function must not write anything to the screen; any input message must be written before the function is called.

b) Write a function whose prototype is

void reduceFraction(int &numerator, int &denominator)

which reduces the fraction whose **numerator** and **denominator** are passed as parameters, dividing them by their greatest common divisor. Suggestion: use the function developed in problem **3.5** to calculate the greatest common divisor of the **numerator** and **denominator**.

c) Write functions to perform the basic operations (addition, subtraction, multiplication and division) on fractions, presenting the result in reduced form. Choose suitable prototypes for these functions.

d) Write a program to test the developed functions.

3.9.

The final result of this problem will be a program to show, on the screen, the calendar of a given year. The development of this program will be done modularly, using a bottom-up approach.

a) A leap year is a year that meets any of the following conditions: it is divisible by 4 but not divisible by 100; however, years divisible by 400, despite being divisible by 100, are considered leap years (eg, the year 2000 was a leap year but the year 2100 will not be). Write a function that has as parameter an integer representing a year and returns a Boolean value, indicating whether the year is leap or not (true if it is and false if it is not). Write a program to test this function.

b) Write a function that has as parameters two integers, representing a month and a year, and returns the number of days of that month, in that year. Note that only the month of February has a variable number of days, depending on whether the year is leap or not.

c) In November 2004, Sohael Babwani published an article in the Mathematical Gazette in which he describes a formula for calculating the day of the week (Sunday, Monday, ...) corresponding to a certain date in the Gregorian calendar. The formula is as follows:

$$ds = \left(\left\lfloor \frac{5 \cdot a}{4} \right\rfloor + c + d - 2 \cdot (s\%4) + 7 \right)\%7$$

where

- $\lfloor\ \rfloor$ – operator that calculates the integer contained in its operand
- % – operator that calculates the remainder of integer division
- *ds* – day of week
- *d* – day of month
- *m* – month number (1-January, 2-February, …)
- *s* – two first digits of the year (ex: if the year is 2010, *s* will take the value 20)
- *a* – two last digits of the year (ex: if the year is 2010, *a* will take the value 10)
- *c* – month code, given by the following table, where *m* represents the month number (1-January, 2-February, ...); note that the *c* depends on whether the year is leap or not but it only differs for the months of January and February.

		c	
month	***m***	**leap**	**non-leap**
January	1	6	0
February	2	2	3
March	3	3	3
April	4	6	6
May	5	1	1
June	6	4	4

		c	
month	***m***	**leap**	**non-leap**
July	7	6	6
August	8	2	2
September	9	5	5
October	10	0	0
November	11	3	3
December	12	5	5

The result, *ds*, should be interpreted as follows: 0 = Saturday, 1 = Sunday, 2 = Monday, etc.
For example, applying the formula to "1 of January of 2011" you get:

$$ds = \left(\left\lfloor \frac{5 \cdot 11}{4} \right\rfloor + 0 + 1 - 2 \cdot (20\%4) + 7 \right) \%7 = 0$$

indicating that the corresponding day of the week is Saturday.
Write a function that has as parameters 3 integer numbers, representing a date (year, month, day), and that returns an integer indicating the corresponding day of the week. Write a program that reads a date and, using this function, writes the name of the corresponding day of the week (Sunday, Monday, ...).

d) Write a function that, using the function developed in the previous paragraph, shows on the screen the calendar of a month/year specified by the user, in a format similar to the following:

```
January/2011
Sun  Mon  Tue  Wed  Thu  Fri  Sat
                                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   29
 30   31
```

e) Finally, write a program that, using the previously developed functions and others that you consider necessary, shows on the screen the calendar of all months of a year indicated by the user.

3.10.

a) Write a function, **factorial_ite(unsigned int n)**, that determines the factorial of a number using an iterative algorithm. Declare the variable that will contain the result as being of type **unsigned long long**. Start by determining which is the largest integer that can be represented in such a variable. Also, determine which is the largest number whose factorial is less than that integer. Note: there is a constant, **ULLONG_MAX**, defined in **<climits>** that contains the value of the largest integer that can be represented in a variable of type **unsigned long long**.

b) Write a recursive function, **factorial_rec(unsigned int n)**, that determines the factorial of a number. The result must also be of type **unsigned long long**.

3.11.

Euclid's recursive algorithm for determining the greatest common divisor (GCD) of two numbers, **m** and **n**, is the following:

- if **m** is divisible by **n**, then GCD(**m**,**n**) is **n**;
- otherwise, GCD(**m**,**n**) is given by GCD(**n**,remainder of the divison of **m** by **n**)

Write a recursive function that determines the greatest common divisor of 2 numbers that it receives as parameters. Write a program for testing that function.

3.12.

Write and test a program that overloads a function, **area()**, that can be used to calculate:

- the area of a circle, given its radius;
- the area of a triangle, given its 3 vertices (remember problem **3.1**);
- the area of a rectangle, given 2 opposite vertices.

3.13.

Consider the following function:

```
int rollDie(int low = 1, int high = 6)
{
  assert(high >= low);
  return (rand() % (high - low + 1)) + low;
}
```

a) Explain what it does. Also, explain the effect of the **assert()** statement.

b) The function has 3 different signatures. Explain why.

c) Is it be possible to overload this function with a function whose prototype is **int rollDie()**? Explain why or why not.

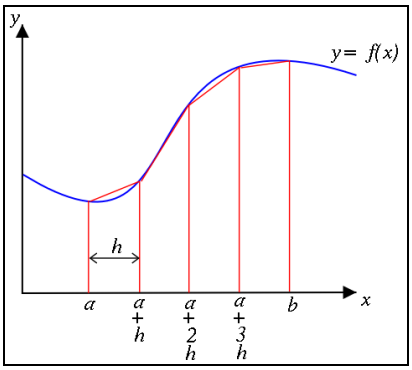
3.14.

The "trapezoidal rule" is a numerical method for calculating an approximate value for the definite integral. To calculate the integral

$$\int_a^b f(x)\,dx$$

the area under the curve *y=f(x)* is divided into *n* regions, each with a width *h* = (*b*-*a*) / *n*. The area of each region is approximated by the area of a trapezoid. The sum of the areas of all the trapezoids gives an approximate value of the definite integral. The area of the *i*-th trapezoid (*i* = 1,2, ...) is given by

$$\frac{h}{2}\big(f(a+(i-1)h)+f(a+ih)\big)$$



In general, the estimate of the integral improves with the decrease of *h*.

a) Write a function whose prototype is

double integrateTR(double f(double), double a, double b, int n),

with the parameters *f* ,*a*, *b* e *n*, above mentioned, that calculates the integral of a function, using this method.

b) Write a program to calculate the following two integrals:

$g(x)=x^2$ when $a=0, b=10$ e $h(x)=\sqrt{4-x^2}$ when $a=-2, b=2$.

The function *h* defines a semi-circle with radius 2. Compare the estimate obtained with the real area of the semi-circle. Try with different values of *n*.