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.

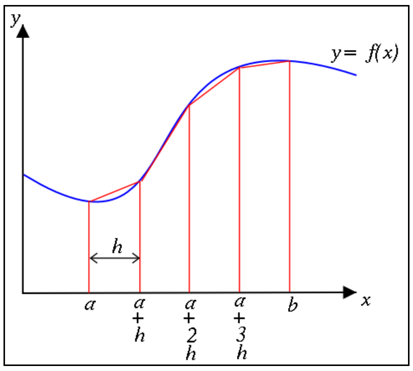
3.14.

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*.