C++ Lab 6: Subprograms (part 1) 3 classes

Objectives

1. Organize your code into sub-programs.
2. Write sub-programs in C++.
3. Manipulate parameters passed by value.
4. Distinguish between variable and parameter.

General comments

1. Read the entire question.
2. Write the algorithmic specification (to be copied as a comment under the prototype).
3. Prepare a set of test cases.
4. Write the function code
5. Automatically test each sub-program with the test sets provided in main(). Leave your tests in your program.

WORK TO HAND IN: upload your files for exercise 3, including test cases.

Exercise 1 *(Paper and pencil)*: from prototypes to specifications and from specifications to prototypes

A subprogram prototype has to specify how the subprogram is to be used.
Consider the following example:

```
bool isPpositive(int);
```

- What is the name of this subprogram?
- Is it a function or a procedure? Why?
- What is the status of its parameters? (D, R or D/R)
- Write its algorithmic specification.

We want this subprogram to return true if the number passed as a parameter is strictly positive and false otherwise. Write its definition.

Exercise 2: First program using three files

From now on, your programs will have to be organized using three files, as explained in class:

- a source code file for the main program,
- a header file for the declarations (prototypes) of subprograms,
- a source code file for the definitions of subprograms.

a) Create a project "**computations**" and add to it the 3 following files. To do so, see below: "Additional Code::Blocks information"

- **computations.h**: will contain the sub-program prototypes. Remember the preprocessor directives.
- **computations.cpp**: wll contain the sub-program definitions. Remember to include the file **computations.h**.
- **main.cpp**: will contain the main function. Remember to include the **computations.h** file.

Reminder: never include a .cpp file, never compile a .h file.

b) Copy the prototype of the **isPositive** function in the **computations.h** file and its definition in the **computations.cpp** file.

c) We want to test that the **isPositive** subprogram functions correctly. What are the different cases that need to be tested? Add statements to the **main.cpp** file to do these tests. Do not ask the user for values. To test different cases, it suffices to call the **isInterval** subprogram with well-chosen constant values and to display the result.

Additionnal Code::blocks information

To add a header file (.h) to an existing project (also consult the Codeblocks file on Moodle).

- Select : File → New → File
- Select " C/C++ header"
- Enter the full path name of the file to be created (**interval.h**). Click on "**...**" and store it in the same directory as the other files of the project.
- Check that the "Header guard word" field is not empty and that the box "Add file to active project" has been checked.
- Select all target files by clicking on **All**.
- Click on Finish.

Proceed in a similar fashion to add the **computations.cpp** source code file to your project (this time, you will select "C/C++ source"). Notice that the **computations.cpp** file, once added, is empty. Your project should now contain 2 source code files and 1 header file.

Exercise 3: A menu to choose among several alternatives

We want a program that asks the user to enter a number ***n*** and then displays a menu giving the following options:

1. Compute and display the sum of the integers ranging between 1 and ***n***
2. Compute and display factorial ***n***.
3. Display the list of the divisors of ***n.***
4. Quit.

Reminder: the factorial of a positive integer n (n!) is the product of all positive integers less than or equal to n. For example, 5! = 5 x 4 x 3 x 2 x 1 = 120.

- Each option has to be implemented by a separate subprogram.
- The statements for inputting **n** and displaying the results have to be part of the main program (unless the purpose of the sub-program is to trigger a display).
- The menu will be implemented by a function that returns an integer corresponding to the option selected by the user.

Your program has be written using the 5 following files:

- **main.cpp**: containing the source code for the main program.
- **menu.cpp**, **menu.h**: containing respectively the definition and the prototype of the menu function. The header file also contains the declaration of a **constant for each option proposed by the menu**.
- **computations.cpp**, **computations.h**: containing respectively the definitions and the prototypes of the various computation subprograms.

Work to be done

1. Use the **computations** project created for exercise 2. It needs to include **5 files**:
   - **main.cpp** (comment out the statements used for the preceding exercise),
   - **menu.h** and **menu.cpp** (provided on Moodle). Take a look at what they contain. Note that constants are defined
   - and **computations.h**, **computations.cpp** (created for exercise 2).
2. Using the subprogram **isPositive**, write the beginning of the main.cpp program so that it asks the user to enter a number and checks that it is positive. The program must ask again if this is not the case. Test.
3. Using the subprograms defined in **menu.cpp**, expand the **main.cpp** program to do the following: After inputting the number n, call up the **menu** function (displaying the menu and reading the user's response) and perform a do/while loop to find out what computation the user wants to perform until the user asks to quit (choosing the QUIT response). Test all possible options by simply displaying the user's choice (for instance: "You selected option 3").

4. Write the algorithmic specification of the **sum** function, then add the prototype and the definition of this function to, respectively, the **computations.h** and **computations.cpp** files. Careful: this function returns the result (no displays).

   Then modify the main program so that option 1 calls the **sum** function and displays the result. Test different cases.

5. Proceed in the same way for the other options (factorial and divisors)

Upload exercise 3, including tests (an archive containing the entire project directory)

Exercise 4: From specifications to prototypes, with (R) and (D/R) parameters

Using the C++ class notes volume 2 page 23 (summary for prototypes), review the following algorithmic specifications (from the TD4, therefore in French) and translate them into C++ prototypes:

Procédure afficheSpecial (x, y)
*{Affiche **x** fois la valeur **y**}*
paramètres (D) x, y : entiers

Fonction estDiviseur(x,y) retourne booléen
*{Retourne VRAI si **x** est un diviseur de **y**, et FAUX sinon}*
paramètres (D) x, y: entiers

Fonction encore() **retourne booléen**
{Retourne VRAI si l'utilisateur veut recommencer le traitement en cours, FAUX sinon}
paramétres (aucun)

Procédure sommeDesDiviseurs (nb, somme)
{renvoie dans **somme** la somme des diviseurs de l'entier **nb**}
paramètres (D) nb : **entier**
(R) somme : **entier**

Fonction nbOccurrences(tab, val) retourne entier
{Retourne le nombre d'occurrences d'un entier donné dans un tableau d'entiers}
paramètres (**D**) tab : **tableau d'entiers**
(**D**) val : **entier**

Procédure doubler(tab2D)
{prend un tableau 2D d'entiers et remplace toutes les valeurs de ce tableau par son double.}
parametres (D/R) tab2D : **tableau d'entier à 2 dimensions**

Fonction tousEgaux (tab2D) retourne **booléen**
{ prend un tableau 2D d'entiers et retourne vrai si ce tableau contient des valeurs qui sont toutes identiques.
On suppose que le tableau n'est pas vide.}
paramètres (**D**) tab2D: **tableau d'entiers à 2 dimension** *{non vide}*

Programming Vocabulary Lab6

Most terms are very similar in English: **procedures**, **functions**, **parameters** and **arguments**.

The French term "sous-programme" is translated here as "**subprogram**", these are sometimes called "subroutines". Two fundamental kinds of subprograms are procedures and functions. Some C++ programming websites (and books) only refer to functions, since procedures are functions that return void.

The transmission modes (D), (R) and (D/R) will be translated as (G) for given, (R) again for returned, and (G/R) for both modes.

Preprocessor directives

From http://www.cplusplus.com/doc/tutorial/preprocessor/

"Preprocessor directives are lines included in the code of programs preceded by a hash sign (#). These lines are not program statements but directives for the preprocessor. The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is actually generated by regular statements"