Linköping University
Department of Computer and Information Science (IDA)
The UPP group 2015-08-28

Selection and repetition

Aim

You will in this laboration practice on using control structures for selection and repetition in order to provide a more (but not completely) fool-proof program (after all, the ingenuity of complete fools are amazing[1]).

You should pay special attention to how you express yourself in code and how you format it in a clear and readable way. Tasks can be solved by either control structure, but most often one is more suitable than others. In addition you should make effort to produce really clear and structured program output.

Reading instructions

- Control statements (`if`, `do`, `while`, `for`, `switch`)
- Data types (`int`, `char`, `float`, `double`, `std::string`)
- Constants (`const`)
- Expressions, precedence and associativity
- Arithmetic operators (`+`. `-`, `*`, `/`, `%`)
- Comparison and logic operators (`<`, `>`, `<=`, `>=`, `==`, `!=`, `!`, `&&`, `||`)
- Output manipulation (`<iomanip>` library)
- Output of error messages (standard error stream `cerr`)
- Data type strengths and limitations

[1]Not quite a quote from Douglas Adams

Assignment

Write a program that prints a tax (VAT) table. The program is supposed to ask the user for the following values (remember that all values must be reasonable):

- A lower and an upper limit for the prices
- A length for each step in the table (stride)
- Tax percentage (a decimal number between 0 and 100 %)

What's important in this assignment is that the program should support the user, in case the latter enters erroneous data (erroneous is in this case if the floating point number entered is unreasonable compared to the type of data entered). Examples shown below shows how the program is supposed to perform with the some specific input. If you enter different values the data in the table will be different, but the table should keep the same style.

The user will NEVER enter any other data type than float values, and these will always be in the interval `[-100.00, +100000.00]`. In addition, the user will never enter more than 2 decimals.

It is up to you to conclude what input is reasonable. The idea is that the user is buying stuff from a store, for example, and the store will not accept to pay the buyer anything.

A convenient way to start any program is to leave out the error handling, i.e. assume that the user always enters appropriate values. This will allow you to focus on the actual functionality (table part in this case) and getting that right.

Once you get a program that prints a correct table, it is time to move on to the next step - adding the error handling. The user is supposed to get the error messages (and opportunity to enter correct data) as soon as possible after his or her input. The error messages must be informative and give instructions on how to enter correct data.

If you following the work process introduced above and in the previous lab also in the coming assignments you will save time and headache.

Below are two examples of the program, given the instructions above. The program should look like this upon execution. The input from the user is in bold italics, just to help you interpret the instruction as before.

N.B. In order to discover some of the data type limitations you may only use variables of type `int` and `float`. You may think of ways to alter your algorithm to minimize the effect of those limitations.

Example 1

```
INPUT PART
==========
Enter first price: 10.00
Enter last price : 15.00
Enter stride     : 0.5
Enter tax percent: 10.00

TAX TABLE
=========
       Price              Tax      Price with tax
-------------------------------------------------
       10.00             1.00               11.00
       10.50             1.05               11.55
       11.00             1.10               12.10
           .                .                   .
           .                .                   .
       15.00             1.50               16.50
```

Example 2

```
INPUT PART
==========
Enter first price: 10.00
Enter last price : 12.00
Enter stride     : 0.3
Enter tax percent: 20.00

TAX TABLE
=========
       Price              Tax      Price with tax
-------------------------------------------------
       10.00             2.00               12.00
       10.30             2.06               12.36
       10.60             2.12               12.72
           .                .                   .
           .                .                   .
       11.80             2.36               14.16
```

Example 3

```
INPUT PART
==========
Enter first price: -10.00
ERROR: First price must be at least 0 (zero) SEK
Enter first price: 100.00
Enter last price : 101.00
Enter stride     : -10.3
ERROR: Stride must be at least 0.01
Enter stride     : 0.1
Enter tax percent: 12.00

TAX TABLE
=========
       Price              Tax      Price with tax
-------------------------------------------------
      100.00            12.00              112.00
      100.10            12.01              112.11
      100.20            12.02              112.22
           .                .                   .
           .                .                   .
      101.00            12.12              113.12
```

N.B. Notice the last ”Price without tax”-value in example two is 11.80 and not 12.00 (nor is it 12.10)!

This is because the length of the interval isn’t a multiple of the step length. In this case the last multiple inside the interval will be the last price included in the table.

Example three show how the user enters erroneous data at first. It is your task to think of more cases with unreasonable data and ensure the program responds with appropriate error messages. The program must not terminate when unreasonable input is encountered, the user should be given the opportunity to enter correct data. **We will still not handle the case when the user enter letters when asked for numbers.**

Reasonable test cases (to start with) can be (make combinations to verify your program further):

```
First price:    -1  0  10   100
Last price:     -1  0  1    11    12   15   101  100000
Stride:         -1  0  0.1  0.25  0.3  0.5  1    10
Tax percentage: -1  0  1    20    50   100  101
```

HINT: When you test your program it is important to be thorough. You should write down the test cases you want and the expected result. Every time you change something in the program you need to rerun ALL test cases (you may have fixed one case, but at the same time broken another!). A good tester looks “out of the box”. He does not consider the user to be of the kind that understands instructions or what to do with the program, thus expecting the unexpected. In this assignment we already said that we will only use certain input to your program. No values outside the interval `[-100.00, +100000.00]` will be used, and never more than two decimals. And always floating point numbers. This is to limit the scope of the assignment for you.