Linköping University
Department of Computer and Information Science (IDA)
The UPP group 2024-09-02

Selection and repetition

Aim

In this assignment you will practice the usage of 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 must pay special attention to how you express yourself and that you format the code in a clear and readable way. Tasks can be solved by any control structure, but most often one is more suitable than the others. In addition you should make an effort to produce clear and structured program output.

Reading instructions

- Control statements (`if`, `do`, `while`, `for`, `switch`)
- Data types (`int`, `char`, `float`, `double`, `std::string`)
- 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%)

In this assignment it is important that the program supports the user in case they input erroneous data (i.e. if the floating point numbers entered do not fulfill the requirements specified above). The examples presented further down demonstrates what the program is supposed to do with specific inputs. If you enter different values the data in the table will of course 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.

What is and isn't a resonable input is largely up to you. But the idea is that this program is used by customers of a store, so it might be a good idea to assume that any input that leads to the store having to pay the customer would be considered unresonable.

A convenient way to start writing 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 (printing the table 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 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.

Below are two example runs of the program, given the instructions above. Your program should look very similar (the values might be slightly different, but more-or-less the same) to the examples. The formatting of the table must be identical. The input from the user is in bold italics, just to help you interpret the example as before.

Requirement: 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 (rounding errors).

Hint: `0.1 + 0.1 + 0.1` is not *necessarily* the same as `3*0.1` due to rounding errors. For floating point numbers it is generally better to do one multiplication rather than repeated addition. Make sure to keep this in mind when calculating the values in your table.

Note: The number of rows in the tables below have been reduced to better fit on the page. Your program should *not* do this. Your program must print all the rows.

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: -5.00
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
```

Note: The last "price without tax"-value in example 2 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 stride length. In this case, the last value inside the range `[10, 12]` that can be reached with stride length `0.3` is `10 + 6*0.3 = 11.80`.

Example three shows how the user enters erronous data. It is your task to think of more cases with unresonable data and ensure that the program responds with an appropriate error message. The program must not terminate when unresonable input is encountered, the should instead be given the option to enter correct data. **We do not handle the case when the user enter letters when asked for numbers.**

Resonable test cases to start with can be (any combination of):

```
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 all the test cases you have tried and their expected output. Every time you make a change to your program you should verify that the program still works by running all these test cases again (your change might fix one test case, but you must also make sure that it doesn't break another case). A good tester looks "outside of the box". You should never expect the user to understand instructions or what to do with the program. In this assignment we will only use certain inputs to your program, so you don't have to go overboard. All values entered will be decimal numbers with no more than two decimals. All values will be in the interval `[-100.00, +100000.00]`, so testing should only be done within these parameters.