Computation I 5EIA0
Exercise 1: Calculator (v2.0, April 6, 2021)
Deadline Wednesday 9 September 23:55

In this exercise you will develop step by step a simple calculator whose operation is very similar to the Windows Calculator. Your calculator will accept only integer values as input. These values can be entered by the user both in a decimal or a binary representation. After entering a value, the user must select one of five operations (addition, subtraction, multiplication, division, result). This operation is then applied to the entered value and the (intermediate) result computed so before is printed on the terminal. If you are wondering how such a calculator works, then you can try the Windows Calculator and see for yourself. Of course you can also complete this exercise and try it on your own calculator.

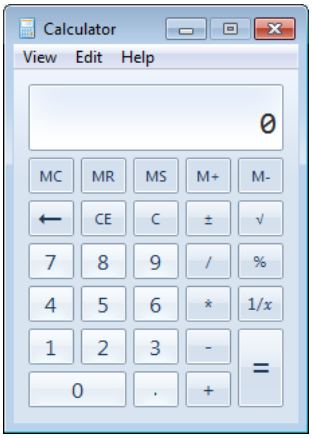

Figure 1: Windows Calculator.

Implementing the complete calculator with all functionality described above is a difficult task. Therefore we will step by step refine the implementation of the calculator. This allows you to test the functionality at intermediate point in time during the development process. Such a development strategy is called *incremental or iterative design* and it is advisable to use this strategy on any future program you will write. It will help you to find errors quickly.

Task 1. Open a new file in 'Kate' or any other text editor which you prefer. Assign a meaningful name to the file (e.g., *calculator.c*).
You can do this exercise in the virtual machine or on the PYNQ board, as you wish. (Both are Ubuntu Linux computers.) On the PYNQ you can use exactly the same procedure as in last week's exercise (make a new directory week2 and copy the Makefile.) However, the libpynq is not required for this exercise, and you can use `gcc calculator.c` to compile your program, and `./a.out` to run it, if you think that's easier.

Task 2. When the calculator is started, the user should be greeted properly. Write a C program that prints the following text to the terminal:

```
** Calculator **
```

As mentioned before, you will develop your calculator using an incremental design process. Therefore, we will start with building a calculator that only accepts decimal values as input. Near the end of this exercise, once the calculator is working properly with decimal numbers, we will extend the calculator to also accept binary numbers as input.

Task 3. Extend the C program that you have made in the previous task such that it asks the user to input a value and subsequently it should read this value from the input and finally it should print the value on the terminal.

Hint: You can use the following C statement to read a single integer value from the input:

```
scanf(" %d", &value);
```

Note that there is a space in front of the `%d`. This ensures that the `scanf` function skips all whitespace characters (e.g., space, tab, newline) in front of the decimal value.

When you run your program it should now produce the following output:

```
** Calculator **
Value?
```

After you entered (for example) the value 123, the terminal should look as follows:

```
** Calculator **
Value? 123
Input: 123
```

Task 4. Extend the C program that you have made in the previous task such that it asks the user indefinitely to input a value. After each time that the user has input a new value, this value should be added to the values previously input to the calculator.

After you entered (for example) the values 100 and 123, the terminal should look as follows:

```
** Calculator **
Value? 100
Input: 100
Result = 100.00
Value? 123
Input: 123
Result = 223.00
Value?
```

Hint: The program you have created in Task 5 will never end. To abort the execution of this program press <ctrl>+<c> on your keyboard.

At this point, your calculator is only able to add numbers together. In the next few tasks you will extend your calculator to support also other operations (i.e., subtraction, multiplication, division).

Task 5. Extend your C program such that it asks the user to input the operation that needs to be performed on the result computed so far and the value that the user has just entered. The operations that need to be supported (at this stage of the development process) are listed in the following table:

Character	Operation
+	addition
-	subtraction
*	multiplication
/	division

Make sure that your program reads one character from the input and subsequently it should perform the requested operation. For example, when the user inputs the character '*' you should multiply the result computed so far and the value that the user just entered. The result of this computation then becomes the intermediate result that is used on the next round of the calculator.

Despite the fact that your calculator accepts only integer values as input, the result might be a floating point number. Hence, the variable that you use to store the intermediate result should be of type `float` and not of type `int`. To verify that your calculator works correctly you could try to compute the result for 45/2. Does your calculator find the correct answer?

Hint: You can use the following C statement to read a single character from the input:

```
scanf(" %c", &c);
```

Note that there is a space in front of the %c. This ensures that the `scanf` function skips all whitespace characters (e.g., space, tab, newline) in front of the character.

A possible run of your program should now look as follows:

```
** Calculator **
Value? 100
Input: 100
Operation? +
Result = 100.00
Value? 2
Input: 2
Operation? *
Result = 200.00
Value? 20
Input: 20
Operation? -
Result = 180.00
Value?
```

The calculator which you have developed in the previous tasks will always use the result computed so far on any of the four operations you implemented in the previous task. When the user makes a mistake, he/she needs to restart the calculator to reset this value or alternatively he/she needs to add the negative value of the current result to reset the value. These are both cumbersome solutions. It would be much more convenient to have an explicit operation to clear the intermediate result.

Task 6. Extend the program such that when the user inputs the character 'c' as an operation, the intermediate result is cleared (i.e., reset to zero).

A possible run of your program should now look as follows:

```
** Calculator **
Value? 100
Input: 100
Operation? +
Result = 100.00
Value? 20
Input: 20
Operation? c
Result = 0.00
Value? 10
Input: 10
Operation? +
Result = 10.00
Value?
```

Currently the user has no mechanism to terminate the execution of the calculator except for pressing <ctrl>+<c> on the keyboard. This is not desirable in our final calculator. Therefore you will add in the next task one additional operation to the calculator. By choosing this operation, the user should be able to terminate the calculator.

Task 7. Extend the program such that when the user inputs the character '=' as an operation, the intermediate result is printed and subsequently the program is terminated (i.e., no new value is requested).

A possible run of your program should now look as follows:

```
** Calculator **
Value? 100
Input: 100
Operation? +
Result = 100.00
Value? 20
Input: 20
Operation? =
Result = 100.00
```

Your calculator is now able to perform all desired operations on values that are input as a decimal number. The specification of our calculator as given at the start of this exercise mentioned that a user should be able to input values both in a decimal as well as a binary representation. You have realized the first part, but as a final step you need to extend the calculator such that it can also deal with values input in a binary representation. Since your calculator needs to accept any integer value as input, it should be able to handle both positive and negative numbers.

Most processors represent signed integers in 2's complement encoding. A `short int` is a sequence of 16 bits: $a_{15}, a_{14}, a_{13}, \ldots, a_1, a_0$. In that case a_{15} is the *most significant* bit and a_0 is the *least significant* bit. The following equation can be used to compute the decimal representation of a 16 bit value:

$$d = -a_{15} * 2^{15} + a_{14} * 2^{14} + \ldots + a_2 * 2^2 + a_1 * 2^1 + a_0 * 2^0$$

For example, the binary number 1100000000000100 corresponds with the decimal value -16380.

Extending the calculator such that it supports values input in a binary notation is a complex task. Therefore we will use the concept of incremental design once more to extend the calculator with this feature. As a first step, the program should be extended such that it asks the user whether he/she wants to enter a decimal value or a binary value. This should of course be done before the user is asked to input the actual value.

Task 8. Extend your program such that it asks the user whether he/she wants to input a binary or decimal value. The user inputs his/her choice by inputting the character b (for binary) or d (for decimal). When the user inputs any other character (e.g., 'q' or 'a'), the assumption is that the user wants to enter the value 0. In that situation, the user should not be asked to input a new value. Instead he/she should immediately be asked to select an operation. **Hint:** use a `switch` statement to realize this task.

When a user inputs the character 'b', the value that is input should (according to the specification of our calculator) be interpreted as a binary number. However, for now you may just treat the value input by the user as a decimal value. Using the concept of incremental design, you will deal with the binary to decimal conversion in a next task.

```
** Calculator **

Binary or decimal? (b/d) d
Value? 100
Input: 100
Operation? +
Result = 100.00
Binary or decimal? (b/d) d
Value? 10
Input: 10
Operation? +
Result = 110.00
Binary or decimal? (b/d) c
Operation? -
Result = 110.00
Binary or decimal? (b/d)
```

Task 9. Add a new function to your program with the following definition:

```
int read_binary_value()
{
        int value = 0;

        return value;
}
```

Adapt the `main` function of your program such that this function is called when the user indicates that he/she wants to input a value in binary representation.

Task 10. Implement the function `read_binary_value`. This function should ask the user to input a 16 bit value in binary representation and convert it to its decimal representation. The resulting (decimal) value should be returned by the function.

Hint: You can use the following C statement to read a single bit from the input:

```
scanf("%1d", &bit);
```

Note that in contrast to other situations in which you used the function `scanf`, there is no space in front of `%1d`. To read all 16 bits you will have to call the `scanf` function 16 times. You should use a loop for this purpose.

Submission: Your final solution must be submitted through OnCourse which will automatically grade this submission. For this you must copy the `7-kitt.c` file to your computer and then upload it on Oncourse (`Exercise 1A: Knight Rider (due 9 Sept 23:55)`). You can resubmit as often as you want until the deadline.

Task 11. Your final task is to safely shut down the PYNQ board. (Or you can keep programming!)

1. Shut down the Ubuntu operating system by executing `sudo halt` on the command line in the terminal. The password is "student". After you press return, any X windows and shared folders that you may have in the Computation virtual machine will disappear.
2. Set the power switch to OFF.
3. Disconnect all cables.
4. You can leave the SD card in the board.
5. Store the PYNQ such that it does not damage during transport.

- **18/2 v2.0** Calculator baseline.