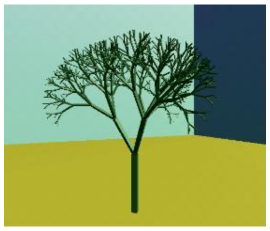

GATS Tutorial

Temperature Table (in C++)

Author: Garth Santor
Editors: Trinh Hān
Version/Copyright: 1.1.1 (2024-01-16)

Overview

One of the most common introductory exercises in computer programming education is the Celsius-Fahrenheit table. Everybody does it! The first time I coded this I was in elementary school and used BASIC. Since then, I've had to do this project in Pascal, Fortran, C, C++, Java, Python, and COBOL. I even coded once in VAX assembler.

Ask yourself, "How long after writing 'hello, world!' did I write the temperature table program?

What we will do here in this challenge is take this age-old project and push it to professional level code. The challenge will be presented as a series of mini challenges that will take us from a rudimentary implementation to a polished rigorous implementation.

The problem

Create a console application that prints out a conversion table for Fahrenheit and Celsius temperatures.

Phase 0 – Background

You can't solve a problem that you don't understand. If you aren't well versed in temperature physics, do some quick research on *temperature*, *Fahrenheit*, and *Celsius*.

The first task is to identify the conversion formula. In the 1970's we would normally convert from Celsius to Fahrenheit as most North Americans were raised on Fahrenheit temperatures and Celsius was for Europeans or scientists.

Where to look? The quickest most reliable first stop is usually Wikipedia. However, there is nothing like searching a true authority for your answer.

Try:

- The National Weather Service (US government).
- The Canadian Government Weather Service (Ministry of the Environment)

Research or derive the formula for converting temperatures from Fahrenheit to Celsius and Celsius to Fahrenheit.

Phase 1 – Conceptual demonstration

The first program simply demonstrates that our formulae are correct.

Write a C console application that prints a simple table, the first column Fahrenheit, the second Celsius. The rows run from -40 °F to 140 °F in intervals of 10 °F.

Phase 2 – Basic I/O

The second version will solicit input from the user: the low, high, and interval, and provide prompts. Output should be rounded to 1 decimal place, and printed in 10-character columns.

Phase 3 – Flexible input

The third version will accept the table range in any order. If the largest value is first, then the table will be printed in descending order, otherwise it will be printed in ascending order.

Phase 4 – Dynamic Columns

Make each column one character wider than the largest number in the column.

Put any repeated operations into functions.

Phase 5 – Optional Input

If the user enters no parameters, the start will be -40, the end 140, and the interval 5.

If the user enters only one parameter, it is the table end point and the start point is zero with an increment of 1.

If the user enters two parameters, they are the start and end values of the table. The interval is 1.

If the user enters three parameters, they are the start, end, and increment values of the table.

Question: how do you input nothing or values?

Phase 6 – more to come…

Look for more phases in version 2

The Solutions

Present here are the series of solutions to the problem in C.

Phase 0 – The formulae

The websites with reliable information:

- Wikipedia: https://en.wikipedia.org/wiki/Conversion_of_units_of_temperature
- Canadian Ministry of the Environment: https://weather.gc.ca/canada_e.html
  - Formulae found in the glossary: https://climate.weather.gc.ca/glossary_e.html under ‘celsius’.
- US National Weather Service: https://www.weather.gov/
  - Temperature converter: https://www.weather.gov/epz/wxcalc_tempconvert
  - Formulae document: https://www.weather.gov/media/epz/wxcalc/tempConvert.pdf

Fahrenheit

The Fahrenheit scale was proposed by German physicist Daniel Gabriel Fahrenheit in 1724. Temperatures are expressed in **degrees Fahrenheit** (**°F**). The lower defining point (0 °F) was chosen as the freezing temperature of a brine made from equal parts of ice, ammonium chloride (a salt), and water. This leads to the following Fahrenheit temperature facts:

- Freezing point of water: 32 °F
- Mean human body temperature: 98.6 °F
- Boiling point of water: 212 °F

Celsius

The Celsius scale is an SI unit (International System of Units) created by Swedish astronomer Anders Celsius in 1742. Temperatures are expressed in **degrees Celsius** (**°C**). The lower defining point (0 **°C**) was chosen as the freezing temperature of water, and the upper defining point (100 **°C**) was chosen as the boiling temperature of water.

The math

The formula from Celsius to Fahrenheit is constructed by converting the range [0 °C, 100 °C] to the range [32 °F, 212 °F]. Since the Celsius temperature is already zero-based, we can directly convert the range [0, 100] to the equivalent normalized Fahrenheit range [32-32, 212-32] or [0, 180]. The conversion factor is $\frac{180}{100}$ which can be reduced to $\frac{9}{5}$. The lower bound adjustment of 32 can then be added giving the final formula:

$$F = \frac{9}{5}C + 32$$

The reverse formula can be derived via algebra:

$$\begin{aligned}
F &= \frac{9}{5}C + 32 \\
F - 32 &= \frac{9}{5}C + 32 - 32 \\
F - 32 &= \frac{9}{5}C \\
(F - 32) \times 5 &= \frac{9}{5}C \times 5 \\
5(F - 32) &= 9C \\
\frac{5(F - 32)}{9} &= \frac{9C}{9} \\
\frac{5}{9}(F - 32) &= C
\end{aligned}$$

Phase 1 – Conceptual Demonstration

See Project **TT Phase 1: TT_Phase_1_main.cpp**

```
	constexpr double fahr_start{ -40.0 };
	constexpr double fahr_end{ 140.0 };
	constexpr double fahr_interval{ 10.0 };

	for (double fahr_temp{ fahr_start }; fahr_temp <= fahr_end; fahr_temp += fahr_interval) {
		const double celc_temp{ 5.0 / 9.0 * (fahr_temp - 32.0) };
		cout << fahr_temp << " F\t" << celc_temp << " C\n";
	}
```

Notes:

1. **Use of double**: go for precision over speed. I use type double for all real-number computation except where memory needs to be minimized and I know that the computations will not require more than 6 digits of precision.
2. **Use of const and constexpr:** use const or constexpr for anything that you don't need to modify. This may help you catch bugs. Use constexpr if the value never changes – ever. Use const if the value never changes after it is computed at runtime.
3. **Use of real literals and not integer literals:** we are performing a physical world computation – so real numbers. It shows our intent and avoids mistakes such as accidental integer division.
   $\frac{9.0}{5.0} = 1.8$ but $\frac{9}{5} = 1$ due to integer truncation.

Phase 2 – Basic I/O

See Project **TT Phase 2: TT_Phase_2_main.cpp**

```
#include <iostream>
#include <iomanip>
using namespace std;

	// Get table parameters from the user
	cout << "Temperature Table 1.0.0: (c) 2020-24, Garth Santor\n";
	cout << "Enter a low and high Fahrenheit temperature and interval: ";
	double fahr_start{ -40.0 };
```

```
        double fahr_end{ 140.0 };
        double fahr_interval{ 10.0 };
        cin >> fahr_start >> fahr_end >> fahr_interval;

        // Output the table
        cout << fixed << setprecision(1);
        for (double fahr_temp{ fahr_start }; fahr_temp <= fahr_end; fahr_temp += fahr_interval) {
                const double celc_temp{ 5.0 / 9.0 * (fahr_temp - 32.0) };
                cout << setw(10) << fahr_temp << " F\t" << setw(10) << celc_temp << " C\n";
        }
```

Notes:

1. **`#include <iomanip>`**: The `<iomanip>` library adds formatting objects to the <iostream> library. We'll use `fixed`, `setw()`, and `setprecision()` from that library.
2. **Use of single line input**: I prefer single-line input as it is more user friendly, and easier to advance into more complex and subtle input.
3. **'fixed' and 'setprecision()':** are placed once outside the loop as the change the output state and apply to all future outputs of real numbers.
4. '**`setw()`**': is placed infront of each variable as it only applies to the next output.

Phase 3 – Flexible input

See Project **TT Phase 3: TT_Phase_3_main.cpp**

The input section is the same, we only change the table.

```
#include <cmath>


        // correct interval direction
        fahr_interval = abs(fahr_interval);
        if (fahr_start > fahr_end) {
                fahr_interval = -fahr_interval;
        }

        // Output the table
        bool descending_table = fahr_interval < 0;


        // Output the table
        cout << fixed << setprecision(1);
        for (double fahr_temp{ fahr_start };
                descending_table ? fahr_temp >= fahr_end : fahr_temp <= fahr_end;
                fahr_temp += fahr_interval)
        {
                const double celc_temp{ 5.0 / 9.0 * (fahr_temp - 32.0) };
                cout << setw(10) << fahr_temp << " F\t" << setw(10) << celc_temp << " C\n";
        }
```

Notes:

1. The conditional operator used in the for-loop isn't particular efficient, but acceptable as the cout statement completely dominates the run-time performance of this loop.

Phase 4 – Dynamic Columns

See Project **TT Phase 4: TT_Phase_4_main.cpp**

Here, I introduce several functions (as the calculations will be repeated). The overdue Fahrenheit to Celsius conversion, a function to compute the number of digits in the whole part of a real number, and a function to pick the maximum value of two unsigned integers.

```
constexpr double fahr_to_cel(double fahrenheit) {
	return 5.0 / 9.0 * (fahrenheit - 32.0);
}


inline int num_whole_digits(double number) {
	return (int)floor(log10(abs(number))) + 1;
}
```

The longest numbers to print are either the first or the last value of each column.

```
// column width is the maximum # of digits +1 for '.', +1 for fraction, +1 for padding
const int fcw{ max(num_whole_digits(fahr_start), num_whole_digits(fahr_end)) + 3 };
const int ccw{ max(num_whole_digits(fahr_to_cel(fahr_start)),
num_whole_digits(fahr_to_cel(fahr_end))) + 3 };
```

The print statement can now use dynamic columns.

```
cout << setw(fcw) << fahr_temp << " F " << setw(ccw) << celc_temp << " C\n";
```

Notes:

1. I've made the functions inline or constexpr as it is very small (the call overhead probably exceeds the computation). Since the num_whole_digits() function contains non-constexpr functions (*floor, log10*) I had to make it inline.
2. I've added 3 to the column widths to accommodate the decimal place, the fractional part, and a possible negative symbol.

Phase 5 – Optional Input

See Project **TT Phase 5: TT_main5.c**

The input code is modified to read a full input line as text as a string.

Replace

```
	double fahr_start{ -40.0 };
	double fahr_end{ 140.0 };
	double fahr_interval{ 10.0 };
	cin >> fahr_start >> fahr_end >> fahr_interval;
```

With

```
	string input_line;
	getline(cin, input_line);
	istringstream sin(input_line);
	sin >> fahr_start >> fahr_end >> fahr_interval;
```

Notes:

1. 'getline' was selected because it reads up until a newline or end-of-file – the exact behaviour I'm looking for.
2. Any variables not read will have their initialized values.

Phase 6 - …

Look for more in version 2.0.0

Document History

Version	Date	Activity
0.0.0	2020-02-25	Document created (C version)
1.0.0	2020-02-26	C version published.
1.1.0	2024-01-03	C++ version published.
1.1.1	2024-01-16	Removed a **Note** from Phase #1 left over from the ‘C’ version of this document.