Advanced Programming Tutorial
Worksheet 2: Data types

Keywords: C++ containers, `std::vector`, `std::array`, ranged `for`

Prepare!

Before coming to the tutorial, and so that the time is enough to solve the exercise, revise the slides of the lecture unit "Variables and data types" and answer the following questions:

- ☐ What is the difference between the stack and the heap?
- ☐ What is integer division?
- ☐ What do curly braces define?
- ☐ How can we define arrays of dynamic size in C++?
- ☐ Look at the code file `caseTrack.cpp`: which parts of the code will you need to change, and which can you ignore? What does the function `populate_vector` already implement?

Exercise 1: C++ containers

You are prototyping an application to visualize **daily** case numbers in each country for a global pandemic. You would probably want to get this data from a database, however, for the purposes of prototyping, you decide to mock the actual data using random data. For this exercise, you will be working in `caseTrack.cpp`:

1. In `main`, create a vector `global_cases` to store the latest case numbers, that is the case numbers for today, for all 195 countries in the world. There is a one-to-one correspondence between the `global_cases` vector and the `countries` array in `caseTrack.cpp`: `global_cases[i]` holds the number of cases reported in `countries[i]` today.
2. Complete the implementation of the `populate_vector` function to populate `global_cases` with dummy data.
3. Find the country with the most cases today and print its name and the number of cases to the terminal.
4. Uncomment the call to `write_to_csv` and plot the file with your favorite external tool.
5. **Idea:** Refactor your code to make task 3 easier. You may want to read about std::pair or std::tuple.
6. **Idea:** Your prototype is coming along nicely, but now you want to extend it to show data for cruise ship cases as well. How would you add this to your existing prototype? Is there anything about the structure of your program that you would want to change?

Exercise 2: Matrix operations

Now that we know about contiguous memory access, let's see how we can use our new data types to create matrices and how we can iterate through them. Matrix-vector products appear everywhere, so let's implement one from scratch!

1. Define an `std::vector<int> v` and print all of its elements. Use a ranged for-loop.
2. Move this printing operation to a function with the following signature and return type: `void print_vector(std::vector<int> v)`.
3. Get the elements of `v` from the user via the keyboard, without assuming the size. Use a while-loop. Hint: (see learncpp.com.)

```cpp
while (std::cin >> x) { // Read and return false if not integer (try . )
  v.push_back(x);       // Add x to the vector
}
std::cin.clear();       // Clear the error state of cin (revert to true)
std::cin.ignore();      // Ignore the last element from the input buffer
```

4. If `v` has N elements (hint: `size()`), get an $N \times N$ matrix `A` from the user and print it. You can define a 2D matrix as a vector of vectors: `std::vector<std::vector<int>>`. Use two classical nested for-loops.
5. Compute $m = A \cdot v$ using two classical nested for-loops. Print the result.
6. How can we reduce the memory footprint of the printing functions, while avoiding modifying the printed data by accident? Your IDE may already be suggesting a solution.

Advice / C++ Core Guidelines

The C++ Core Guidelines are a "collaborative effort led by Bjarne Stroustrup, much like the C++ language itself", which aims to "help people to use modern C++ effectively". You can always refer to them if you are confused about best practices. Start with the following:

- In.0: Don't panic! All will become clear in time.
- By default, make objects immutable
- Prefer using STL vector by default unless you have a reason to use a different container.
- ES.11: Use `auto` to avoid repeating type names.
- ES.20: Always initialize an object.

Epilogue

In every simulation code, a central piece is matrix operations. We typically don't implement such operations ourselves, but rely on existing libraries, as we will see in Worksheet 5. Library developers, however, do write such operations, with optimization in mind. We will invest the last part of the course on optimization of matrix operations.

Chair of Scientific Computing in Computer Science – Technical University of Munich
Last updated on October 30, 2024