Introduction to Coroutines

This lesson gives an overview of coroutines, predicted to be introduced in C++20.

WE'LL COVER THE FOLLOWING

Coroutines are functions that can suspend and resume their execution while keeping their state. The evolution of functions goes one step further in C++20.

What I present in this section as a new idea in C++20 is actually quite old. The term coroutine was coined by Melvin Conway; He used it in his publication on compiler construction in 1963. Likewise, Donald Knuth called procedures a special case of coroutines. Sometimes, it just takes a while to get your ideas accepted.

With the new keywords `co_await` and `co_yield`, C++20 will extend the execution of a C++ function with two new concepts.

Thanks to `co_await expression` it will be possible to suspend and resume the execution of the `expression`. If you use `co_await expression` in a function `func`, the call `auto getResult= func()` will not block if the result of the function is not available. Instead of resource-consuming blocking, you have resource-friendly waiting.

`co_yield expression` allows it to write a generator function that returns a new value each time. A generator function is a kind of data stream from which you can pick values. The data stream can be infinite, therefore, we are in the center of lazy evaluation with C++.

A Generator Function

The following program is as simple as possible. The function `getNumbers`

The following program is as simple as possible. The function `getNumbers` returns all integers from `begin` to `end`, incremented by `inc`. In that case,

`begin` has to be smaller than `end`, and `inc` has to be positive.

```
// greedyGenerator.cpp

#include <iostream>
#include <vector>

std::vector<int> getNumbers(int begin, int end, int inc = 1){

  std::vector<int> numbers;
  for (int i = begin; i < end; i += inc){
    numbers.push_back(i);
  }

  return numbers;

}

int main(){

  std::cout << std::endl;

  const auto numbers= getNumbers(-10, 11);

  for (auto n: numbers) std::cout << n << " ";

  std::cout << "\n\n";

  for (auto n: getNumbers(0, 101, 5)) std::cout << n << " ";

  std::cout << "\n\n";

}
```

Of course, I am reinventing the wheel with `getNumbers` because that job could be done with `std::iota` since C++11.

Two observations about the program are important. On one hand, the `numbers` vector in line 8 always gets all values. This holds even if I'm only interested in the first 5 elements of a vector with 1000 elements. On the other hand, it's quite easy to transform the function `getNumbers` into a lazy generator.

```
// lazyGenerator.cpp

#include <iostream>
```

```
#include <vector>

generator<int> generatorForNumbers(int begin, int inc = 1){


  for (int i = begin;; i += inc){
    co_yield i;
  }

}

int main(){

  std::cout << std::endl;

  const auto numbers= generatorForNumbers(-10);

  for (int i= 1; i <= 20; ++i) std::cout << numbers << " ";

  std::cout << "\n\n";

  for (auto n: generatorForNumbers(0, 5)) std::cout << n << " ";

  std::cout << "\n\n";

}
```

While the function `getNumbers` in the file greedyGenerator.cpp returns a `std::vector<int>`, the coroutine `generatorForNumbers` in `lazyGenerator.cpp` returns a generator. The generator `numbers` in line 18 or `generatorForNumbers(0, 5)` in line 24 returns a new number on request. The query is triggered by the range-based for-loop; to be more precise, the query of the coroutine returns the value `i` via `co_yield i` and immediately suspends its execution. If a new value is requested, the coroutine resumes its execution exactly at that place.

The expression `generatorForNumbers(0, 5)` in line 24 is a just-in-place usage of a generator.

I want to explicitly stress one point: the coroutine `generatorForNumbers` creates an infinite data stream because the for-loop in line 8 has no end condition. This is fine if I only ask for a finite number of values such as in line 20, but this will not hold for line 24 since there is no end condition. Therefore, the expression runs *forever*. Because coroutines are a totally new concept to C++, I want to provide a few details about them.