Boguslaw Cyganek AGH University of Krakow, Poland

CppCon 2024 Poster Proposal

Introduction (required): Title and brief overview of what the poster reports on.

Title:

Pipeline architectures in C++: overloaded pipe operator | std::expected and its monadic operations

Brief overview:

Functional programming in C++ is gaining importance and is attracting programmers. One of its most characteristic patterns is composition of functions in the form of a pipeline pattern. Since C++20 we can use the ranges library with its characteristic function composition abilities thanks to the overloaded pipe operator. In this poster I show how to implement **a custom pipeline framework** that employs **`std::expected`**, available since C++23. An overloaded custom pipe operator | will be presented, as well as I'll explain `std::expected` and especially its **monadic operations** that can be tricky in practice. All these together present *novel and efficient programming paradigms* in C++.

Relevance (required): Why the work is interesting, and what problems it addresses.

I recently started writing the second volume of my book about C++ [1][3]. One of the topics was to describe the behavior of the overloaded pipe operator | in `std::ranges`. I came across Ankur's Satle lecture [4] and decided to expand on this topic – I used the `std::expected`, that came with C++23, which is perfect for this type of pipelines. What's more, `std::expected` itself allows you to build pipelines – this time using its *monadic* operations. In this poseter I will explain details how to build your own pipeline architecture in modern C++ environment, using the overloaded pipe operator, as well as `std::expected` which undoubtedly deserves popularity in the C++ programming community.

In this presentation, we return to the well-known **pipeline architecture** [5] but in a new version, made possible thanks to modern C++. This is a very universal pattern that fits into **functional programming**.

This poster touches upon the following topics of C++: *functional programming* (pipeline and pipe operator), modern C++, libraries and frameworks of general interest (*std::expected*), practical experiences using C++ in real-world applications, a new/renewed programming pattern to known use cases, parallelism/multi-processing (planned), *concepts* and generic programming (for std::expected).

Discussion (required): Technical description of the presented topic.

The main two topics that will be discussed are: (i) pipeline pattern with the custom overloaded pipe `operator |`, and (ii) `std::expected` – its principal functionality together with the monadic operations. Together they provide novel ways of programming in C++. The plan is as follows:

1. **Short intro to functional programming in C++:**
   - ❑ Basic concepts, imperative vs. declarative, lazy eval, monads;
   - ❑ Function composition and a range adaptor closure object (RACO);

2. What is a pipe and a pipeline in software engineering?

- ❑ Pipes in Unix/Linux;
- ❑ The famous competition: Donald Knuth vs. Doug McIlroy;

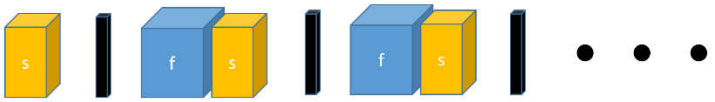
```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -rn |
sed ${1}q
```

3. Overloading the pipe operator |

- ❑ Overloaded pipe operator in `std::ranges`;
- ❑ Building and using our own pipeline: `operator |` – the *basic* example:

```
using Function = std::function< std::string( std::string && ) >;

auto operator | ( std::string && s, Function f ) -> std::string
{
        return f( std::move( s ) );
}
```



- The | operator simply calls `f` providing it with `s`.
- `s` needs to be `std::move`'d, since it is a named object here.
- The callable `f` must be able to accept `std::string &&` as its parameter and return `std::string`

A PROBLEM… what to do if one link in the above pipeline cannot complete its operation and transmit its result because an error occurred?

A SOLUTION… throw or use std::expected

4. Let's get to know `std::expected`:

- ❑ Basic operations – for example:

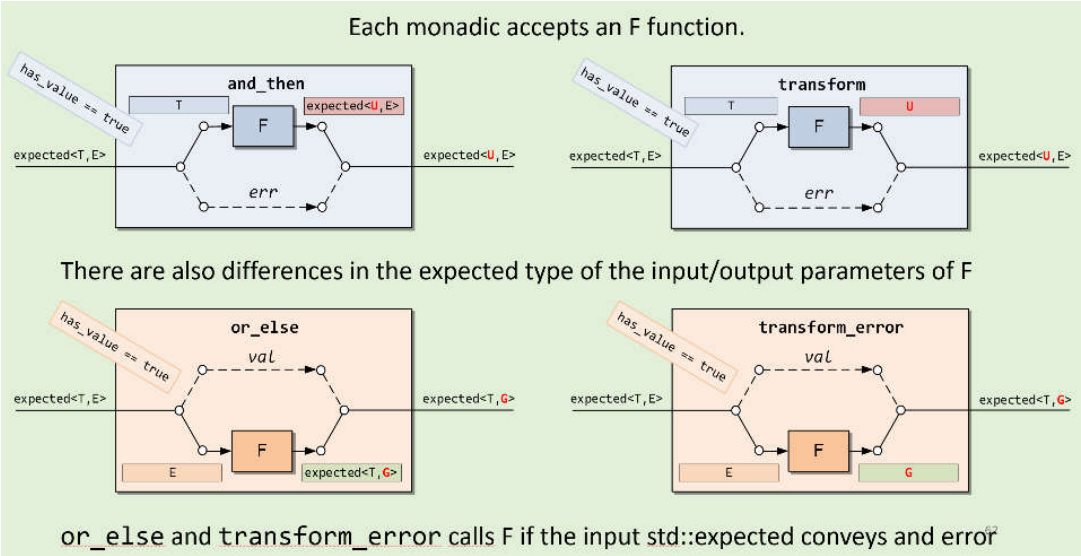
Member	Description
`constructor`	Constructs the `std::expected<T,E>` object with zero or a number of parameters. Can use `std::in_place` to ensure in place construction. Can be created from a single `std::unexpected` object.
`operator bool` `has_value`	Returns true if `std::expected` contains a valid object, false otherwise.
`operator ->` `operator *`	Access the expected value. UB if has_value returns false.
`value`	Returns a reference to the expected value, if present. Returns nothing if T is void. Otherwise throws `std::bad_expected_access`
`error`	Returns a reference to the unexpected value, if present. UB if has_value returns true.
`value_or( d )`	Returns the contained value if present, otherwise returns d. Not declared if T is void.
`operator ==`	Compare optional objects

- Using `std::expected` in our own pipeline framework with custom pipe operator;

  Ex: Custom overloaded pipe operator for pipe-line with `std::expected`

```
template < typename T, typename E, typename Function >
requires       std::invocable< Function, T >
               && is_expected< typename std::invoke_result_t< Function, T > >
constexpr auto operator | ( std::expected< T, E > && ex, Function && f )
                              -> typename std::invoke_result_t< Function, T >
{
       return ex ? std::invoke( std::forward< Function >( f ),
                        * std::forward< std::expected< T, E > >( ex ) ) : ex;
}
```

- How to create a concept for `std::expected`;
- Monadic operations of `std::expected` **explained**: `and_then`, `or_else`, `transform`, and `transform_error`;



5. Conclusions and further steps:

- Other pipe libraries;
- Applications;

Completion Status (required): Work that has been completed, and work that is expected to be completed before the poster presentation at CppCon.

The main concept and software are ready, as this is part of the book that I haven't published yet. An additional topic that would be nice to do is a parallelized version.

Supporting Material: If applicable, results and references to work, for example, GitHub.

The entire code with examples will be available on *github* [2].

[1] Bogusław Cyganek: *Introduction to Programming with C++ for Engineers*, Wiley-IEEE 2021
[2] https://github.com/BogCyg
[3] https://home.agh.edu.pl/~cyganek/
[4] Ankur Satle: *Functional Composable Operations with Unix-Style Pipes in C++* - CppCon 2022, https://www.youtube.com/watch?v=L_bomNazb8M
[5] Wikipedia: *Pilpeline*, 2024, https://en.wikipedia.org/wiki/Pipeline_(software)