5-monadic-types

June 3, 2024

1 Towards monads

C ++ 17 introduced the `std::optional` type, which is said somehow **monadic**. Below, we will briefly present it, together with `std::expected`, and touch on the functional problematic of those kinds of types.

1.1 When A is optional: `std::optional<A>` (C++17)

The parameterized class `std::optional<A>` allows to store or not a value of type `A` in a more readable and more efficient way than a `std::pair<T,bool>` or `std::variant<T,bool>`, and a easier way than raising an exception.

When such a variable is converted to a boolean, we will get `true` if the value of type`A` is available and `false` otherwise.

[1]:
```
#include <string>
#include <iostream>
#include <optional>
```

[5]:
```
std::optional<double> divide( double num, double denom ) {
  if (denom) return (num/denom) ;
  else return std::nullopt ;
}
```

[3]:
```
template< typename A >
std::ostream & operator<<( std::ostream & os, std::optional<A> const & opt )
 { if (opt) return (os<<opt.value()) ; else return (os<<"nothing") ; }
```

[4]:
```
std::cout<<"2/3: "<<divide(2,3)<<std::endl ;
std::cout<<"3/0: "<<divide(3,0)<<std::endl ;
```

```
2/3: 0.666667
3/0: nothing
```

An optional variable is empty until it is assigned a value. It can be emptied by a call to `reset ()` or by assigning it the special value `std::nullopt`.

1.2 With an error value: `std::expected<T,E>` (C++23)

Similar to `std::optional<T>`, but combined with an alternative code error of any type.

[6]:
```
%%file tmp.expected.cpp

#include <string>
#include <iostream>
#include <expected>
#include <cmath>

std::expected<double,std::string> divide_and_sqrt( int num, int denom ) {
  if (denom==0) return std::unexpected("cannot divide by 0") ;
  double d { static_cast<double>(num)/denom } ;
  if (d<0) return std::unexpected("cannot sqrt a negative number") ;
  else return sqrt(d) ;
}

template< typename T, typename E >
std::ostream & operator<<( std::ostream & os, std::expected<T,E> const & exp ) {
  if (exp) {
    return (os<<exp.value()) ;
  } else {
    return (os<<"("<<exp.error()<<")") ;
  }
}

int main() {
  std::cout<<" 2/3: "<<divide_and_sqrt(2,3)<<std::endl ;
  std::cout<<" 3/0: "<<divide_and_sqrt(3,0)<<std::endl ;
  std::cout<<"-2/3: "<<divide_and_sqrt(-2,3)<<std::endl ;
}
```

```
Writing tmp.expected.cpp
```

[8]:
```
!rm -f tmp.expected.exe && g++ -std=c++23 tmp.expected.cpp -o tmp.expected.exe
```

[9]:
```
!./tmp.expected.exe
```

```
 2/3: 0.816497
 3/0: (cannot divide by 0)
-2/3: (cannot sqrt a negative number)
```

2 Questions ?

2.1 Exercise 1 (simple)

If your compiler supports C++17: 1. Modify `mysqrt` below so that it returns an `std::optional<double>` rather than a `double`. 2. Adapt `divide` so that it accepts optional inputs,

Optionally, if your compiler support C++23: 1. Modify `divide` so that it returns an expected output, which detect an imaginary numerator, imaginary denominator or a zero denominator. 2.

Adapt `operator<<` so that it accepts an expected input.

[29]:
```
%%file tmp.monadic-types.cpp

#include <cmath>
#include <iostream>
#include <optional>

double mysqrt( double d ) {
  return std::sqrt(d) ;
}

std::optional<double> divide( double d1, double d2 ) {
  if (d2==0) return std::nullopt ;
  else return d1/d2 ;
}

template< typename T>
std::ostream & operator<<( std::ostream & os, std::optional<T> const & opt ) {
  if (opt) { return (os<<opt.value()) ; }
  else { return (os<<"(nothing)") ; }
}

int main() {
  std::cout<<divide(mysqrt(10),0)<<std::endl ;
  std::cout<<divide(mysqrt(-10),mysqrt(10))<<std::endl ;
  std::cout<<divide(mysqrt(10),mysqrt(-10))<<std::endl ;
  std::cout<<divide(mysqrt(10),mysqrt(10))<<std::endl ;
}
```

```
Overwriting tmp.monadic-types.cpp
```

[27]:
```
!rm -f tmp.monadic-types.exe && g++ -std=c++23 tmp.monadic-types.cpp -o tmp.
↪monadic-types.exe
```

[28]:
```
!./tmp.monadic-types.exe
```

```
(nothing)
-nan
-nan
1
```

2.2 Exercise 2 (difficult, C++23)

Rather than modifying `divide`, write a high-order function `raise`, which take as input a function of type `std::optional<OutputType> (&)( InputType1, InputType2 )`, and return a function of type `std::expected<OutputType,std::string> (&)( std::optional<InputType1>, std::optional<InputType2> )`, which is calling the former one when its input optionals have values, or return a comment about lacking values or lacking result. Then, `raise` can help to reuse

the original `divide` without modifying it.

[12]:
```
%%file tmp.monadic-types.cpp

#include <cmath>
#include <iostream>
#include <optional>
#include <expected>

template< typename OutputType, typename InputType1, typename InputType2 >
auto raise( std::optional<OutputType>(&f)(InputType1,InputType2) )
 {
  return [f]( std::optional<InputType1> input1, std::optional<InputType2>␣
 ↪input2 ) -> std::expected<OutputType,std::string>
   {
    // TO BE COMPLETED
   } ;
 }

std::optional<double> mysqrt( double d ) {
 if (d<0) return std::nullopt ;
 else return std::sqrt(d) ;
}

std::optional<double> divide( double d1, double d2 ) {
  if (d2==0) return std::nullopt ;
  else return d1/d2 ;
}

template< typename T, typename E >
std::ostream & operator<<( std::ostream & os, std::expected<T,E> const & exp ) {
  if (exp) { return (os<<exp.value()) ; }
  else { return (os<<"("<<exp.error()<<")") ; }
}

int main() {
  std::cout<<raise(divide)(mysqrt(10),0)<<std::endl ;
  std::cout<<raise(divide)(mysqrt(-10),mysqrt(10))<<std::endl ;
  std::cout<<raise(divide)(mysqrt(10),mysqrt(-10))<<std::endl ;
  std::cout<<raise(divide)(mysqrt(10),mysqrt(10))<<std::endl ;
}
```

```
Overwriting tmp.monadic-types.cpp
```

[10]:
```
!rm -f tmp.monadic-types.exe && g++ -std=c++17 tmp.monadic-types.cpp -o tmp.
 ↪monadic-types.exe
```

[11]:
```
!./tmp.monadic-types.exe
```

```
10
-nan
```

2.3 Exercise 3 (difficult)

What happens if you want a `mysqrt` of `mysqrt` ? Can you add another flavor of `raise` which would solve the issue ?

[1]:
```
%%file tmp.monadic-types.cpp

#include <cmath>
#include <iostream>
#include <optional>

// TO BE COMPLETED

std::optional<double> mysqrt( double d ) {
 if (d<0) return std::nullopt ;
 else return std::sqrt(d) ;
}

int main()
 {
  std::cout<<mysqrt(mysqrt(10))<<std::endl ;
  std::cout<<mysqrt(mysqrt(-10))<<std::endl ;
 }
```

```
Writing tmp.monadic-types.cpp
```