expected
Обработка ошибок в C++23

Илья Быконя
АО ВНИИЖТ

Существующие способы: errno

```
auto file = fopen("...", "...");
if (errno != 0) {
   perror("Unable to open file");
   return -1;
}
```

Существующие способы: коды ошибок

```
int32_t result{};
const auto [ptr, error_code] =
    from_chars(source.data(), source.data() + source.size(), result);

if (error_code == errc::invalid_argument) {
    cerr << "Invalid argument" << endl;
} else if (error_code == errc::result_out_of_range) {
    cerr << "Result out of range" << endl;
} else {
    cout << "Result: " << result << endl;
}
```

Существующие способы: out-параметры

```
QString str{ /*...*/ };
bool isValidConversion{ true };
auto num = str.toInt(&isValidConversion);
```

Существующие способы: исключения

```
void some_func() { throw runtime_error{ "..." }; }

try {
   some_func();
} catch (const runtime_error& error) {
   cerr << "Error: " << error.what() << endl;
}
```

Существующие способы: чудеса отказоустойчивости

```
QApplication app{ argv, argc };

while (true) {
   try {
      app.exec();
   } catch (...) { }
}
```

expected<T, E>

```
expected<int32_t, runtime_error> result = /*...*/
if (result.has_value()) {
   cout << "Value: " << result.value() << endl;
} else {
   cerr << "Error: " << result.error().what() << endl;
}
```

unexpected<E>

```
expected<int32_t, system_error> to_integer(string_view source) {
    int32_t result{};
    const auto [ptr, error_code] =
        from_chars(source.data(), source.data() + source.size(), result);

    if (error_code == errc::invalid_argument) {
        return unexpected(system_error{
            make_error_code(error_code), "Source must be an int value"
        });
    } else if (error_code == errc::result_out_of_range) {
        return unexpected(system_error{
            make_error_code(error_code), "Number larger than an int"
        });
    } else {
        return result;
    }
}
```

unexpected<E>

```
const auto result = to_integer("-12345");
if (result.has_value()) {
   cout << "Value: " << result.value() << endl;
} else {
   cout << "Error: " << result.error().what() << endl;
}
```

expected<T, E> transform/transform_error

```
F1(T) -> R
F2(E) -> U

expected<T, E>::transform(F1) -> expected<R, E>
expected<T, E>::transform_error(F2) -> expected<T, U>
```

expected<T, E>
transform/transform_error

```
//expected<int32_t, system_error>
const auto result_1 = to_integer("-12345");

//expected<string, system_error>
const auto result_2 = result.transform([](int32_t number) {
   return to_string(number);
}).transform_error([](const auto& error) {
   std::cout << "Error: " << error.what() << std::endl;
   return error;
});
```

expected<T, E>
and_then/or_else

```
F1(T) -> expected<R, E>
F2(E) -> expected<T, U>

expected<T, E>::and_then(F1) -> expected<R, E>
expected<T, E>::or_else(F2) -> expected<T, U>
```

expected<T, E>
and_then/or_else

```
struct extract_json_error: public exception{};
struct load_config_error : public exception{};

expected<json, extract_json_error> to_json(const string& source);
expected<json, load_config_error> load_default_config();
```

expected<T, E>
and_then/or_else

```
//expected<string, extract_json_error>
const auto body = expected<string, extract_json_error>{ "..." };
//expected<string, extract_json_error>
const auto params = body.and_then(
  [](const string& raw) ->expected<json, extract_json_error> {
    return to_json(raw);
  }).or_else(
  [](const extract_json_error&) ->expected<json, load_config_error> {
    return load_default_config();
  });
```

expected

Где стоит использовать

- Обёртки вокруг функций с out-параметрами, кодами ошибок или errno
- Рекурсивные функции
- Точки взаимодействия с внешним миром: файловая система, парсеры
- Любые системы, где ошибка и её отсутствие — приблизительно равновероятные события

expected

Где лучше оставить исключения

- При обработке действительно редких ситуаций (bad_alloc)
- При обработке ошибок программирования/развёртывания (отсутствие конфигурационного файла)
- Когда в функции есть множество типов возможных ошибок, которые она должна отдавать «наверх», т.к. не может их обработать