
14.7. Regex Exceptions

When regular expressions are parsed, things can become very complicated. The C++ standard library provides a special exception class to deal with regular-expression exceptions. This class is derived from `std::runtime_error` (see Section 4.3.1, page 41) and provides an additional member `code()` to yield an error code. This might help to find out what's wrong if an exception is thrown when processing regular expressions.

Unfortunately, the error codes returned by `code()` are implementation specific, so it doesn't help to print them directly. Instead, you have to use something like the following header file to deal with regex exceptions in a reasonable way:

Click here to view code image

```
// regex/regexexception.hpp

#include <regex>
#include <string>

template <typename T>
std::string regexCode (T code)
{
    switch (code) {
      case std::regex_constants::error_collate:
        return "error_collate: "
               "regex has invalid collating element name";
      case std::regex_constants::error_ctype:
        return "error_ctype: "
               "regex has invalid character class name";
      case std::regex_constants::error_escape:
        return "error_escape: "
               "regex has invalid escaped char. or trailing escape";
      case std::regex_constants::error_backref:
        return "error_backref: "
               "regex has invalid back reference";
      case std::regex_constants::error_brack:
        return "error_brack: "
               "regex has mismatched '[' and ']'";
      case std::regex_constants::error_paren:
        return "error_paren: "
               "regex has mismatched '(' and ')'";
      case std::regex_constants::error_brace:
        return "error_brace: "
               "regex has mismatched '{' and '}'";
      case std::regex_constants::error_badbrace:
        return "error_badbrace: "
               "regex has invalid range in {} expression";
      case std::regex_constants::error_range:
        return "error_range: "
               "regex has invalid character range, such as '[b-a]'";
      case std::regex_constants::error_space:
        return "error_space: "
               "insufficient memory to convert regex into finite state";
      case std::regex_constants::error_badrepeat:
        return "error_badrepeat: "
               "one of *?+{ not preceded by valid regex";
      case std::regex_constants::error_complexity:
        return "error_complexity: "
               "complexity of match against regex over pre-set level";
      case std::regex_constants::error_stack:
        return "error_stack: "
               "insufficient memory to determine regex match";
    }
    return "unknown/non-standard regex error code";
}
```

The detailed explanation written in parentheses after the name of the error code is taken directly from the specification of the C++ standard library. The following program demonstrates how to use it:

Click here to view code image

```
// regex/regex5.cpp

#include <regex>
```

```
#include <iostream>
#include "regexexception.hpp"
using namespace std;

int main()
{
  try {
    // initialize regular expression with invalid syntax:
    regex pat ("\\\\.*index\\{([^}]*)\\}",
               regex_constants::grep|regex_constants::icase);
    ...
  }
  catch (const regex_error& e) {
        cerr << "regex_error: \n"
             << " what(): " << e.what() << "\n"
             << " code(): " << regexCode(e.code()) << endl;
  }
}
```

Because we use the `grep` grammar here but do escape the characters `{` and `}`, the program might have an output such as the following:

Click here to view code image

```
regex_error:
 what(): regular expression error
 code(): error_badbrace: regex has invalid range in {} expression
```