TDDD38 - Extra lecture

The regex header

Eric Elfving

Department of Computer and Information Science
Linköping University

LINKÖPING UNIVERSITY

a regular expression (sometimes called a rational expression) is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations.

Wikipedia[1]

[1] https://en.wikipedia.org/wiki/Regular_expression

We have the three "standard" method for regular expressions in the STL;

- Searching - find all occurancies of a pattern in string `regex_search`
- Matching - does the string match the pattern? `regex_match`
- Replacing - replace matching substrings `regex_replace`

All require a `regex` object.

class regex

A `regex` object can't be used by itself, but is needed with the regex functions. In the simplest case, a regex object is initiated with the pattern.

```
std::regex pattern {"[0-9]{3,5}"};
```

By default, a modified ECMAScript version of regular expressions is used.[2]

[2] http://en.cppreference.com/w/cpp/regex/ecmascript

Raw strings

A problem with the regex syntax is that backslashes are used alot. Let's say that we want to look for the string `\begin` (yes I'm using LaTeX). Then the regex syntax requires two backslashes. The problem is that C++ also requires two backslashes to get a literal backslash in a string (and not some escape character)...

```
regex r {"\\\\begin"};
```

Raw strings

There is a solution! In C++11, raw strings were added. The syntax is `R"<delimiter>(<raw string>)<delimiter>"`, where `<delimiter>` is an optional 0-16 character string.

```
string s1 = R"(a raw string)";
string s2 = R"RAW(another raw string)RAW";
```

regex_search

The `regex_search` function returns `true` if the pattern is found somewhere in the given string.

```
int main() {
  ifstream students_file {"all_registered_students.txt"};
  string student;
  regex pattern {"TDDD38"};
  while (getline(students_file, student)) {
    if (regex_search(student, pattern)) {
      cout << student;
    }
  }
}
```

regex_search

Can also use a `match_results` (`smatch` for `std::string` members or `cmatch` for c-strings) to store the first matching substring and groups:

```
int main() {
    string s {"this subject has a submarine as a subsequence"};
    smatch m;
    regex e {"\\b(sub)([^ ]*)"};   // matches words beginning by "sub"

    while (regex_search (s,m,e)) {
        for (auto x:m) cout << x << " ";
        cout << endl;
        s = m.suffix().str();
    }
}
```

```
subject sub ject
submarine sub marine
subsequence sub sequence
```

regex_iterator

An alternative to changing the search string in a loop is using a `regex_iterator`

```
string s {"this subject has a submarine as a subsequence"};
regex e {R"<>(\b(sub)([^ ]*))<>"};   // matches words beginning by "sub"

regex_iterator<string::iterator> rit { begin(s), end(s), e };
        // or sregex_iterator
regex_iterator<string::iterator> rend;

for (; rit != rend; ++rit ) {
    for (auto && x: *rit) cout << x << ' ';
    cout << endl;
}
```

Dereferencing a `regex_iterator` gives a `match_results`.

match_results

A `match_results` object works like a container [3], but have some special members...

```
int main ()
{
    cmatch m;
    regex_match( "subject", m, regex{"(sub)(.*)"});
    cout << m.format ("the expression matched [$0].\n")
         << m.format ("with sub-expressions [$1] and [$2].\n");
}
```

```
the expression matched [subject].
with sub-expressions [sub] and [ject].
```

[3] have iterators, size, capacity...

regex_match

Works like `regex_search` but only returns true if the string exactly matches the pattern.

```
bool is_int(string const & str) {
    static regex pattern {"[1-9][0-9]*"};
    return regex_match(str, pattern);
}

int main() {
    string line;
    cout << "Enter an integer: ";
    getline(cin, line);
    while (!is_int(line)) {
        cout << "WRONG, enter a new value: ";
        getline(cin, line);
    }
}
```

Also have a variant with `match_results`.

regex_replace

Returns a copy of the target string with all matched sequences replaced according to a specified format.

```
int main ()
{
    string s {"there is a subsequence in the string\n"};
    regex e {"\\b(sub)([^ ]*)"};

    cout << regex_replace(s, e, "$1-$2");
}
```

```
there is a sub-sequence in the string
```

www.liu.se

LINKÖPING UNIVERSITY