情報科学プロジェクト実験レポート課題

S142063 佐藤涼亮

平成 28 年 11 月 17 日

正規表現

1　課題の内容

正規表現による特定の文字列を取り出すプログラムの作成

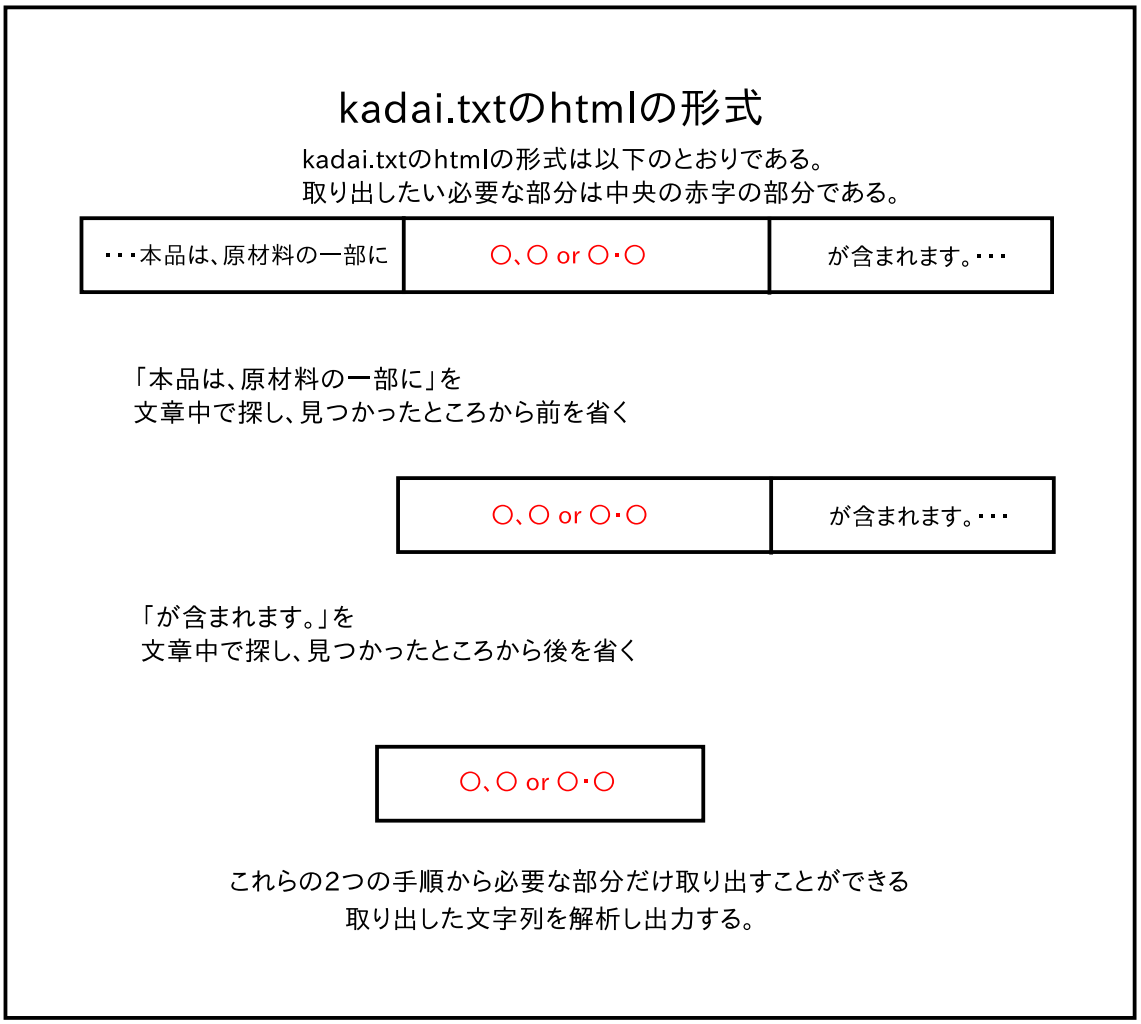
1.1　要点

- C++11 で追加された正規表現の指定方法である std::ECMAScript 構文を使用。
- 標準入力で html の内容を入力する。
- 入力された html の食品アレルギーの原因の文字列を取り出す。
- 日本語を用いるためワイド文字列を使用する。

2　プログラムの説明

今回、正規表現による日本語を含めた文章の字句解析を行う。しかし、日本語のように、漢字 (2 バイト文字) を含むマルチバイト文字列に対して解析を行うのは困難である。そこで、Unicode のようなワイド文字列を用いることで解析を容易に行うことができる。ワイド文字列は、全ての文字を固定長に変換してしまうものである。

html の内容を標準入力により一行ずつ読み取る。読み取った文章をワイド文字列に変換する。文字列を字句解析し、不要な部分を取り除く。取り除き方は次のとおりである。



取り出した部分の解析には、`(([^、・]+)[・、]?)` の原材料の解析パターンを使い行う。原材料の解析パターンでは、マッチした全体 `(([^、・]+)[・、]?)` が 0 番目、原材料のみ `([^、・]+)` が result[1] の 1 番目に格納される。

2.1 目的

C++11 で追加された正規表現の指定方法である std::ECMAScript 構文について学ぶ。

2.2 方法

html に含まれる、食品アレルギーの原因の文字列を、C++11 で追加された正規表現の指定方法である std::ECMAScript 構文を用いて取り出すプログラムの作成。

[1] マッチした情報を格納するクラス

2.3 結果

実行結果

```
$ ./report < kadai.txt
えび かに さけ 鶏肉
乳
大豆 ゼラチン
$
```

原材料のみの出力に成功した。

2.4 考察

原材料のみの出力、および kadai.txt の実行結果と同じ結果が得られた。

3 感想

プログラミング言語の授業で、正規表現を学んだことがあったが、今回は、std::ECMAScript 構文といことで、少し異なった正規表現を学ぶことができた。今までは、マルチバイト文字のみの文字列を字句解析など解析してきたが、ワイド文字に変換することで日本語の解析も行えること学んだ。これらを深く理解することで、インターネットブラウザに標準で備わっている検索機能と同じようなものが作れるのではないかと興味が湧いた。

4 プログラム

ソースコード 1: report.cpp

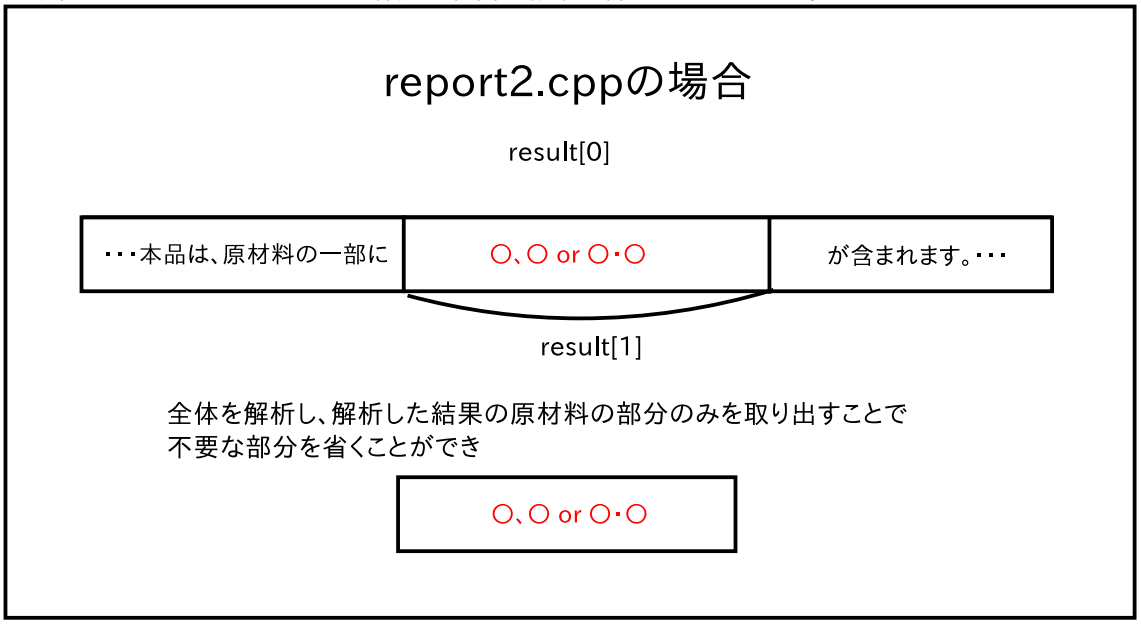
```cpp
#include <iostream>
#include <string>
#include <regex>
#include <locale>
#include <codecvt>

using namespace std;

int main(int argc, char *argv[])
{
  // 正規表現のパターンやオプションを設定するクラスの宣言
  wregex r1(LR"head(本品は、原材料の一部に)head"); // 頭の部分の解析パターン
  wregex r2(LR"materials(([^、・]+)[・、]?)materials"); // 原材料の解析パターン
  wregex r3(LR"tail(が含まれます。)tail"); // 後ろの部分の解析パターン
  string line;
  while (getline(cin, line)) { // 標準入力を一行ずつ読み取る
    // string と wstring を相互変換するクラスの宣言
    std::wstring_convert<std::codecvt_utf8<wchar_t>> cv;
    wstring wline = cv.from_bytes(line); // 読み取った一行をwstring 型に変換
    wsmatch result; // マッチした情報を格納するクラスの宣言
    if(regex_search(wline, result, r1)){ // 対象文字列の中にパターンが含まれるか
      // 文字列の不要な頭の部分を省く
      wline = wline.substr(result.position(0)+result.length(0));
      regex_search(wline,result,r3);
      for(wline = wline.substr(0,result.position(0)); // 文字列の不要な後ろの部分を省く
          regex_search(wline, result, r2); // 残った部分をマッチし続けるまで解析
          wline = wline.substr(result.position(0)+result.length(0))) // 文字列の更新
        cout << cv.to_bytes(result[1]) << "␣"; // マッチした原材料の出力
      cout << endl;
    }
  }
  return 0;
}
```

5　追記

別のパターンでプログラムを作成し、同じ結果を得ることができた。



ソースコード 2: report2.cpp

```
#include <iostream>
#include <string>
#include <regex>
#include <codecvt>
#include <locale>

using namespace std;

int main(int argc, char *argv[])
{
  // 正規表現のパターンやオプションを設定するクラスの宣言
  // 不要な部分を省くための解析パターン
  wregex r1(LR"(本品は、原材料の一部に(.*)が含まれます。)");
  wregex r2(LR"(([^、・]+)[、・]?(.*))"); // 原材料の解析パターン
  string line;
  while (getline(cin, line)) { // 標準入力を一行ずつ読み取る
    // string と wstring を相互変換するクラスの宣言
    std::wstring_convert<std::codecvt_utf8<wchar_t>> cv;
    wstring wline = cv.from_bytes(line); // 読み取った一行をwstring 型に変換
    wsmatch result; // マッチした情報を格納するクラスの宣言
    if (regex_search(wline, result, r1)) { // r1 の解析パターンを検索
      wline = result[1]; // 不要な部分を省き原材料のみに文字列に更新
      while(regex_search(wline,result,r2)){ // r2 の解析パターンを繰り返し検索
        cout << cv.to_bytes(result[1]) << "␣"; // 見つかった原材料の一つを出力
        wline = result[2]; // 残りの原材料に更新
      }
      cout << endl;
    }
  }
  return 0;
}
```