

# 数値解析学 2 GW 中の課題レポート

19C1123 横尾陸

2021 年 5 月 26 日

## 課題

微妙に異なる 2 つの文字列のマッチング

## 1 目的

文字列 2 行 + マッチングを表す o と x の行を出力とするプログラムを作成する .

## 2 方法

この課題に合うようなプログラムを作成する .

### 2.1 提案するアルゴリズム

1. 文字を入力する .
2. 入力された 2 つの文字列数を比べる .  
(a) 文字数が同じだったら 1 つずつ文字を比べる .
3. 多い方に文字数があるように空白を追加する .
4. 前から順に 1 文字ずつ比べる .
5. 違う箇所を記憶させておく .
6. 違う箇所から 1 文字ずつ後ろにずらしていく .
7. もう 1 度前から順に比較する .
8. 文字列と o と x を出力する .

作成したプログラムを Listing1 に示す .

Listing 1 作成したプログラム

---

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 class Matching{
6 private:
```

```

7   std::string str;
8   std::vector<bool> flag;
9   bool fl;
10  std::string maru, batu;
11  int kurikaesi;
12  int num;
13 public:
14  Matching()
15  {
16      maru = 'o';
17      batu = 'x';
18      num=1;
19      fl=false;
20  }
21  ~Matching()
22  {
23  }
24
25  std::string input_string()
26  {
27      std::cout << "string" << num << ":";
28      std::cin >> str;
29      num++;
30      return str;
31  }
32
33  void comparison_string_length(std::string &str1, std::string &str2)
34  {
35      int size = str1.size() - str2.size();
36      int num=0;
37      fl = (size < 0);
38
39      while(!(str1.size() == str2.size())){
40          if(fl){
41              str1.push_back(' ');
42          }else{
43              str2.push_back(' ');
44          }
45      }
46      kurikaesi = str1.size();
47      if(fl){
48          str1.swap(str2);
49      }
50      if(size == 0){
51          for(int i=0;i<str1.size();i++){
52              flag.push_back( str1.at(i) == str2.at(i));

```

```

53     }
54 }else{
55     for(int i=0;i<str1.size();i++){
56         if(!(str1.at(i) == str2.at(i))){
57             num=i;
58             break;
59         }
60     }
61     for(int i=str1.size()-1;i>num;i--){
62         str2.at(i) = str2.at(i-1);
63         str2.at(i-1) = ' ';
64     }
65
66     for(int i=0;i<str1.size();i++){
67         flag.push_back( str1.at(i) == str2.at(i));
68     }
69 }
70 if(f1){
71     str1.swap(str2);
72 }
73 }
74
75 void output_marubatu(){
76     for(int i=0;i<kurikaesi;i++){
77         if(flag.at(i)){
78             std::cout << maru << " ";
79         }else{
80             std::cout << batu << " ";
81         }
82     }
83     std::cout << std::endl;
84 }
85
86 void output_string(std::string out){
87     for(int i=0;i<out.size();i++){
88         std::cout << out.at(i) << " ";
89     }
90     std::cout << std::endl;
91 }
92 };
93 int main(){
94     std::string str1, str2;
95     Matching match;
96     str1 = match.input_string();
97     str2 = match.input_string();
98

```

```

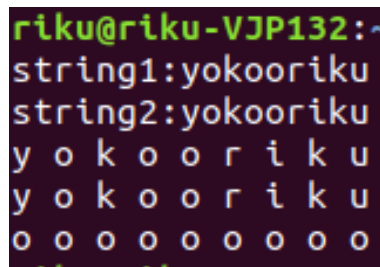
99  match.comparison_string_length(str1, str2);
100
101  match.output_string(str1);
102  match.output_string(str2);
103
104  match.output_marubatu();
105
106  return 0;
107 }

```

---

### 3 結果

プログラムを実行した結果です .

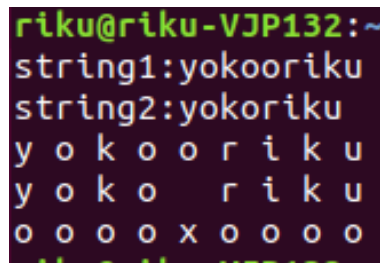


```

riku@riku-VJP132:~
string1:yokooriku
string2:yokooriku
y o k o o r i k u
y o k o o r i k u
o o o o o o o o o

```

図 1 文字列が同じ場合

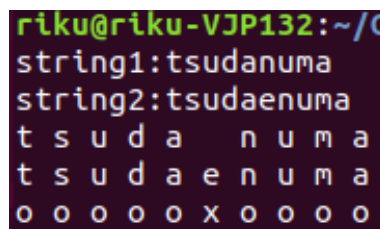


```

riku@riku-VJP132:~
string1:yokooriku
string2:yokoriku
y o k o o r i k u
y o k o   r i k u
o o o o x o o o o

```

図 2 文字列が異なる場合 1



```

riku@riku-VJP132:~/G
string1:tsudanuma
string2:tsudaenuma
t s u d a   n u m a
t s u d a e n u m a
o o o o o x o o o o

```

図 3 文字列が異なる場合 2

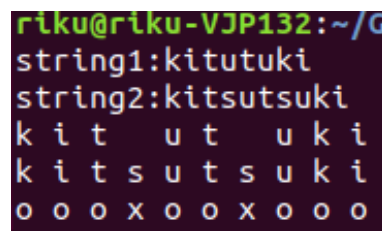
図 1 は 2 つの文字列が同じ場合を比較している . 図 2,3 は文字列が異なる場合を比較している . 1 文字のず

れなら比較できていることがわかる．冗長にはなってしまうが Listing1 の 65 行目に Listing2 を追加すると 2 文字のズレにも対応できる．

Listing 2 追加するプログラム

```
1      for(int i=0;i<str1.size();i++){
2          if(!(str1.at(i) == str2.at(i))){
3              if(num < i){
4                  num = i;
5                  fl2 = true;
6                  break;
7              }
8          }
9      }
10     if(fl2){
11         for(int i=str1.size()-1;i>num;i--){
12             str2.at(i) = str2.at(i-1);
13             str2.at(i-1) = ' ';
14         }
15     }
```

図 4 は 2 文字のズレを実行した結果です．



A terminal window showing the output of a program. The prompt is 'riku@riku-VJP132:~/G'. The program outputs 'string1:kitutuki' and 'string2:kitsutsuki'. Below these, the strings are aligned character by character. 'kitutuki' is aligned under 'kitsutsuki'. The alignment shows that 'kitutuki' is shifted one character to the right relative to 'kitsutsuki'. The alignment is as follows:

k	i	t	u	t	u	k	i		
k	i	t	s	u	t	s	u	k	i
o	o	o	x	o	o	x	o	o	o

図 4 2 文字違う場合

## 4 考察

自分のプログラムの書き方が悪いかもしれないが 1 文字のズレから 2 文字のズレに対応させるだけでプログラムが長くなってしまい．文字列のズレが増える数だけ対応させようと思ったらそれだけプログラムが長くなってしまったと思う．それを今やっている機械学習を用いたら，文字列のマッチングができるのではないかと考えた．