```cpp
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <windows.h>
#include <chrono>

#define BLACK "\x1B[0m"
#define RED "\x1B[31m"
#define GREEN "\x1B[32m"
#define YELLOW "\x1B[33m"
#define BLUE "\x1B[34m"
#define MAGENTA "\x1B[35m"
#define CYAN "\x1B[36m"
using namespace std;
using namespace std::chrono;

struct Letter{
    char alphabet;
    const char * color;
};

void createPuzzle(string fn, vector<vector<Letter>> &puzzle, vector<string> &query);
const char * generateColor();
bool HorizontalR(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool HorizontalL(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool VerticalU(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool VerticalD(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool DiagLRU(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool DiagRLU(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool DiagLRD(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
bool DiagRLD(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int &accessCounter);
void bruteSolution(vector<vector<Letter>> &puzzle, vector<string> query);
void printPuzzle(vector<vector<Letter>> puzzle);

int main(){

    vector<vector<Letter>> puzzle;
    vector<string> query;

    createPuzzle("test\\hard-3.txt", puzzle, query);

    printPuzzle(puzzle);

    auto start = high_resolution_clock::now();
    bruteSolution(puzzle,query);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    cout << "Waktu eksekusi fungsi: "<< duration.count() << " microseconds" << endl;
    cout << "\nSolusi :\n";
    printPuzzle(puzzle);

    return  0;
}
```

```
55
56 void createPuzzle(string fn, vector<vector<Letter>> &puzzle, vector<string> &query){
57     char component = '-';
58     bool isPuzzle = true;
59     string myText;
60
61     ifstream readFile(fn);
62     while(isPuzzle){
63         vector<Letter> line;
64         Letter temp;
65         getline(readFile, myText);
66         myText = myText + " ";
67         if(myText.length() == 1){
68             isPuzzle = false;
69         }else{
70             int row = 0;
71             for(auto x : myText){
72                 int col = 0;
73                 if (x != ' '){
74                     component = x;
75                     temp.alphabet = component;
76                     temp.color = BLACK;
77                     line.push_back(temp);
78                 }
79             }
80             puzzle.push_back(line);
81         }
82     }
83     myText = "";
84     while(getline(readFile, myText)){
85         string readyText;
86         //Cleaning data from space symbol
87         for(auto x : myText){
88             if (x != ' '){
89                 readyText = readyText + x;
90             }
91         }
92         query.push_back(readyText);
93     }
94     readFile.close();
95 }
96
97 const char * generateColor(){
98     static int x = -1;
99     vector<const char *> color = {RED,GREEN,BLUE,YELLOW,MAGENTA,CYAN};
100    x += 1;
101    return color[(x%6)];
102 }
103
104 bool HorizontalR(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
105     bool comp = true;
106     int i = 1;
107     int tempy = y+1;
108     while(i < keyword.length() && comp){
109         accessCounter += 1;
110         if(keyword[i] == puzzle[x][tempy].alphabet){
111             i += 1;
112             tempy += 1;
113         }else{
114             comp = false;
```

```
115         }
116     }
117     if (comp){
118         const char * color = generateColor();
119         for(i = 0; i < keyword.length(); i++){
120             puzzle[x][y+i].color = color;
121         }
122     }
123     return comp;
124 }
125 
126 bool HorizontalL(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
127     bool comp = true;
128     int i = 1;
129     int tempy = y-1;
130     while(i < keyword.length() && comp){
131         accessCounter += 1;
132         if(keyword[i] == puzzle[x][tempy].alphabet){
133             i += 1;
134             tempy -= 1;
135         }else{
136             comp = false;
137         }
138     }
139     if (comp){
140         const char * color = generateColor();
141         for(i = 0; i < keyword.length(); i++){
142             puzzle[x][y-i].color = color;
143         }
144     }
145     return comp;
146 }
147 
148 bool VerticalU(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
149     bool comp = true;
150     int i = 1;
151     int tempx = x-1;
152     while(i < keyword.length() && comp){
153         accessCounter += 1;
154         if(keyword[i] == puzzle[tempx][y].alphabet){
155             i += 1;
156             tempx -= 1;
157         }else{
158             comp = false;
159         }
160     }
161     if (comp){
162         const char * color = generateColor();
163         for(i = 0; i < keyword.length(); i++){
164             puzzle[x-i][y].color = color;
165         }
166     }
167     return comp;
168 }
169 
170 bool VerticalD(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
171     bool comp = true;
172     int i = 1;
173     int tempx = x+1;
```

```
174     while(i < keyword.length() && comp){
175         accessCounter += 1;
176         if(keyword[i] == puzzle[tempx][y].alphabet){
177             i += 1;
178             tempx += 1;
179         }else{
180             comp = false;
181         }
182     }
183     if (comp){
184         const char * color = generateColor();
185         for(i = 0; i < keyword.length(); i++){
186             puzzle[x+i][y].color = color;
187         }
188     }
189     return comp;
190 }
191 
192 bool DiagLRU(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
193     bool comp = true;
194     int i = 1;
195     int tempx = x-1;
196     int tempy = y+1;
197     while(i < keyword.length() && comp){
198         accessCounter += 1;
199         if(keyword[i] == puzzle[tempx][tempy].alphabet){
200             i += 1;
201             tempx -= 1;
202             tempy += 1;
203         }else{
204             comp = false;
205         }
206     }
207     if (comp){
208         const char * color = generateColor();
209         for(i = 0; i < keyword.length(); i++){
210             puzzle[x-i][y+i].color = color;
211         }
212     }
213     return comp;
214 }
215 
216 bool DiagRLU(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
217     bool comp = true;
218     int i = 1;
219     int tempx = x-1;
220     int tempy = y-1;
221     while(i < keyword.length() && comp){
222         accessCounter += 1;
223         if(keyword[i] == puzzle[tempx][tempy].alphabet){
224             i += 1;
225             tempx -= 1;
226             tempy -= 1;
227         }else{
228             comp = false;
229         }
230     }
231     if (comp){
232         const char * color = generateColor();
233         for(i = 0; i < keyword.length(); i++){
```

```
234             puzzle[x-i][y-i].color = color;
235         }
236     }
237     return comp;
238 }
239 
240 bool DiagLRD(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
241     bool comp = true;
242     int i = 1;
243     int tempx = x+1;
244     int tempy = y+1;
245     while(i < keyword.length() && comp){
246         accessCounter += 1;
247         if(keyword[i] == puzzle[tempx][tempy].alphabet){
248             i += 1;
249             tempx += 1;
250             tempy += 1;
251         }else{
252             comp = false;
253         }
254     }
255     if (comp){
256         const char * color = generateColor();
257         for(i = 0; i < keyword.length(); i++){
258             puzzle[x+i][y+i].color = color;
259         }
260     }
261     return comp;
262 }
263 
264 bool DiagRLD(vector<vector<Letter>> &puzzle, string keyword, int x, int y, int
    &accessCounter){
265     bool comp = true;
266     int i = 1;
267     int tempx = x+1;
268     int tempy = y-1;
269     while(i < keyword.length() && comp){
270         accessCounter += 1;
271         if(keyword[i] == puzzle[tempx][tempy].alphabet){
272             i += 1;
273             tempx += 1;
274             tempy -= 1;
275         }else{
276             comp = false;
277         }
278     }
279     if (comp){
280         const char * color = generateColor();
281         for(i = 0; i < keyword.length(); i++){
282             puzzle[x+i][y-i].color = color;
283         }
284     }
285     return comp;
286 }
287 void bruteSolution(vector<vector<Letter>> &puzzle, vector<string> query){
288     int rowSize = puzzle.size();
289     int colSize = puzzle[0].size();
290     int totalAccess = 0;
291     for (auto keyword : query){
292         bool found = false;
```

```
293         int len = keyword.length();
294         int accessCounter = 0;
295         int i = 0;
296         while (i<rowSize && !found){
297             int j = 0;
298             while (j<colSize && !found){
299                 accessCounter += 1;
300                 if(keyword[0] == puzzle[i][j].alphabet){
301                     if(j+len <= colSize && !found){
302                         found = HorizontalR(puzzle,keyword,i,j,accessCounter);
303                     }
304                     if(j-len >= -1 && !found){
305                         found = HorizontalL(puzzle,keyword,i,j,accessCounter);
306                     }
307                     if(i-len >= -1 && !found){
308                         found = VerticalU(puzzle,keyword,i,j,accessCounter);
309                     }
310                     if(i+len <= rowSize && !found){
311                         found = VerticalD(puzzle,keyword,i,j,accessCounter);
312                     }
313                     if(i-len >= -1 && j+len <= colSize && !found){
314                         found = DiagLRU(puzzle,keyword,i,j,accessCounter);
315                     }
316                     if(i-len >= -1 && j-len >= -1 && !found){
317                         found = DiagRLU(puzzle,keyword,i,j,accessCounter);
318                     }
319                     if(i+len <= rowSize && j+len <= colSize && !found){
320                         found = DiagLRD(puzzle,keyword,i,j,accessCounter);
321                     }
322                     if(i+len <= rowSize && j-len >= -1 && !found){
323                         found = DiagRLD(puzzle,keyword,i,j,accessCounter);
324                     }
325                 }
326             j+=1;
327             }
328         i+=1;
329         }
330         if (found){
331             cout << "\"" << keyword << "\"" << "\x1B[32m" << " ditemukan\n" << "\x1B[0m";
332         }else{
333             cout << "\"" << keyword << "\"" << "\x1B[31m" << " tidak ditemukan\n" <<
"\x1B[0m";
334         }
335         cout << "\tJumlah perbandingan huruf : " << accessCounter << endl;
336         totalAccess += accessCounter;
337     }
338     cout  << "\nJumlah perbandingan keseluruhan : " << totalAccess<< endl;
339 }
340
341 void printPuzzle(vector<vector<Letter>> puzzle){
342     for(int i = 0; i<puzzle.size(); i++){
343         for(int j = 0; j<puzzle[i].size(); j++){
344             //cout << puzzle[i][j].color << puzzle[i][j].alphabet << " ";
345             printf("%s%c ", puzzle[i][j].color,puzzle[i][j].alphabet);
346         }
347         cout << endl;
348     }
349     cout << endl;
350 }
```