

cs380su21-meta.sg

[Dashboard](#) / [My courses](#) / [cs380su21-meta.sg](#) / [28 June - 4 July](#) / [Assignment 8 \(Backtracking\)](#).

 [Description](#)


 [Submission](#)

 [Edit](#)

 Submission view

Grade

Reviewed on Tuesday, 6 July 2021, 11:23 PM by Automatic grade
grade: 100.00 / 100.00

Assessment report  [-]
[\[+\]](#)**Summary of tests**

Submitted on Tuesday, 6 July 2021, 11:23 PM ([Download](#))

functions.cpp

```
1  /*!*****
2  \file functions.cpp
3  \author Vadim Surov, Goh Wei Zhe
4  \par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
5  \par Course: CS380
6  \par Section: B
7  \par Programming Assignment 8
8  \date 07-06-2021
9  \brief
10 This file has declarations and definitions that are required for submission
11 *****/
12
13 #include "functions.h"
14
15 namespace AI
16 {
17
18
19 } // end namespace
```

functions.h

```

1  /*!*****
2  \file functions.h
3  \author Vadim Surov, Goh Wei Zhe
4  \par DP email: vsurov@digipen.edu, weizhe.goh@digipen.edu
5  \par Course: CS380
6  \par Section: B
7  \par Programming Assignment 8
8  \date 07-06-2021
9  \brief
10 This file has declarations and definitions that are required for submission
11 *****/
12
13 #ifndef FUNCTIONS_H
14 #define FUNCTIONS_H
15
16 #include <iostream>
17 #include <stack>
18 #include <algorithm>
19 #include <cmath>
20
21 #include "data.h"
22
23 #define UNUSED(x) (void)x;
24
25 class NextLocation_Sudoku1D
26 {
27     AI::MapInt1D* map;
28
29 public:
30
31     NextLocation_Sudoku1D(void* map)
32     : map{ static_cast<AI::MapInt1D*>(map) }{}
33
34     /*!*****
35     \brief
36     A domain specific functor class that is used to find a solution for a
37     one-dimensional Sudoku by using Backtracking algorithm.
38
39     \return
40     Returns next location which is not occupied on the map. If no location,
41     returns Location<> {nullptr, 0};
42     *****/
43     AI::Location<> operator()() const
44     {
45         for (int i = 0; i < map->size; ++i)
46         {
47             if (map->base[i] == 0)
48                 return AI::Location<int>(map->base, i);
49         }
50
51         return AI::Location<int>{ nullptr, 0 };
52     }
53 };
54
55 class NextLocation_Sudoku2D
56 {
57     AI::MapInt2D* map;
58
59 public:
60
61     NextLocation_Sudoku2D(void* map)
62     : map{ static_cast<AI::MapInt2D*>(map) }{}
63
64     /*!*****
65     \brief
66     A domain specific functor class that is used to find a solution for a
67     two-dimensional Sudoku by using Backtracking algorithm.
68
69     \return
70     Returns next location which is not occupied on the map. If no location,
71     returns Location<> {nullptr, 0};
72     *****/
73     AI::Location<> operator()() const
74     {
75         for (int j = 0; j < map->height; ++j)
76         {
77             for (int i = 0; i < map->width; ++i)
78             {
79                 if (map->base[j * map->width + i] == 0)
80                     return AI::Location<int>(map->base, j * map->width + i);
81             }
82         }
83
84         return AI::Location<>{ nullptr, 0 };
85     }
86 };
87
88 class NextCandidate_Sudoku1D
89 {
90     AI::MapInt1D* map;
91
92 public:
93
94     NextCandidate_Sudoku1D(void* map)
95     : map{ static_cast<AI::MapInt1D*>(map) }{}
96
97     /*!*****
98     \brief
99     A domain specific functor class that is used to find a solution for a
100    one-dimensional Sudoku by using Backtracking algorithm.
101
102    \param location
103    Template class for a location on the map used by Backtracking algorithm.
104
105    \return
106    Returns next candidate for a specified location on the map. If no
107    candidate found, returns 0.
108    *****/

```

```

109     int operator()(AI::Location<> location)
110     {
111         int value = map->base[location.getIndex()];
112
113         while (value < map->size)
114         {
115             value++;
116             bool valueFound = false;
117
118             for (int i = 0; i < map->size; ++i)
119             {
120                 if (map->base[i] == value)
121                     valueFound = true;
122             }
123
124             if (!valueFound)
125                 return value;
126         }
127
128         return 0;
129     }
130 };
131
132 class NextCandidate_Sudoku2D
133 {
134     AI::MapInt2D* map;
135
136 public:
137
138     NextCandidate_Sudoku2D(void* map)
139         : map{ static_cast<AI::MapInt2D*>(map) }{}
140
141     /*!*****
142     \brief
143     A domain specific functor class that is used to find a solution for a
144     two-dimensional Sudoku by using Backtracking algorithm.
145
146     \param starting
147     Template class for a location on the map used by Backtracking algorithm.
148
149     \return
150     Returns next candidate for a specified location on the map. If no
151     candidate found, returns 0.
152     *****/
153     int operator()(AI::Location<> location)
154     {
155         int index = location.getIndex();
156
157         int x = index % map->width;
158         int y = index / map->height;
159
160         int value = map->base[y * map->width + x];
161
162         int row_start = y * map->width;
163
164         while (value < (map->width))
165         {
166             value++;
167             bool rowFound = false, columnFound = false, boxFound = false;
168
169             //Check row
170             for (int curr_grid = row_start; curr_grid < row_start + map->width;
171                 ++curr_grid)
172             {
173                 if (curr_grid == index)
174                     continue;
175
176                 if (map->base[curr_grid] == value)
177                     rowFound = true;
178             }
179
180             //Check column
181             for (int i = 0; i < map->height; ++i)
182             {
183                 int curr_grid = i * map->width + x;
184
185                 if (curr_grid == index)
186                     continue;
187
188                 if (map->base[curr_grid] == value)
189                     columnFound = true;
190             }
191
192             //Check box
193             int box_width = std::sqrt(map->width);
194             int box_height = std::sqrt(map->height);
195
196             int startX = x - (x % box_width);
197             int startY = y - (y % box_height);
198
199             for (int i = 0; i < box_width; ++i)
200             {
201                 for (int j = 0; j < box_height; ++j)
202                 {
203                     int curr_grid = ((startY + i) * map->width) + startX + j;
204
205                     if (curr_grid == index)
206                         continue;
207
208                     if (map->base[curr_grid] == value)
209                         boxFound = true;
210                 }
211             }
212
213             if (!rowFound && !columnFound && !boxFound)
214                 return value;
215         }
216

```

```
217         return 0;
218     }
219 };
220
221 namespace AI
222 {
223     // Template class of the Backtracking algorithm.
224     // Parameter NL defines domain-specific NextLocation functor.
225     // Parameter NC defines domain-specific NextCandidate functor.
226     template<typename NL, typename NC>
227     class Backtracking
228     {
229     public:
230         std::stack<Location<>> stack;
231         NL next_location;
232         NC next_candidate;
233
234         Backtracking(void* map = nullptr)
235             : next_location{ map }, next_candidate{ map }{}
236
237         /*!*****
238         \brief
239         Runs the solve function to find solution in a blocking mode
240         *****/
241         void run()
242         {
243             while (!solve()) {}
244         }
245
246         /*!*****
247         \brief
248         One iteration of the search. Used by run() in a blocking running mode or
249         can be called by timer in an non-blocking run
250
251         \return
252         Returns true if location is not found. Else, return false if Sudoku is
253         not solved.
254         *****/
255         bool solve()
256         {
257             if (stack.empty())
258             {
259                 AI::Location<> location = next_location();
260
261                 if (location.notFound())
262                     return true;
263
264                 stack.push(location);
265             }
266
267             AI::Location<> location = stack.top();
268             int candidate = next_candidate(location);
269
270             if (candidate)
271             {
272                 location.setValue(candidate);
273                 AI::Location<> next = next_location();
274
275                 if (next.notFound())
276                     return true;
277
278                 stack.push(next);
279             }
280             else
281             {
282                 location.clearValue();
283                 stack.pop();
284             }
285
286             return false;
287         }
288     };
289 }
290
291 // end namespace
292
293 #endif
```

◀ Showcase: Sudoku

Jump to...

Showcase: Tic-Tac-Toe Game ▶

[VPL](#)

You are logged in as [Wei Zhe GOH](#) ([Log out](#))
[cs380su21-meta.sg](#)
[Data retention summary](#)
[Get the mobile app](#)