listgraph

0.4.0

Generated by Doxygen 1.8.17

Chapter 1

doxygen-cmake-github

Demonstrates Doxygen html generation and publishing on GitHub Pages. The Doxygen files for this project can be seen here.

1.1 How to use

1. Point your browser to this repository (https://github.com/semcneil/doxygen-cmake-github)
2. Press the "Use this template" button
3. Give your repository a new name
4. Write a short (one sentence) description of what your project will do
5. Click the Create repository from template button

1.1.1 VS Code VM Instructions

1. VS Code needs the following extension added:
   (a) C/C++ from Microsoft
   (b) CMake Tools also from Microsoft
2. Connect to Host in New Window
3. Open a terminal (`ctrl+``)
4. Initialize git if you haven't already using the same email you used on your GitHub account:
   (a) `git config --global user.email "you@example.com"`
   (b) `git config --global user.name "Your Name"`
5. Navigate to the parent directory for your project
6. Clone your repository using the URL from the GitHub Code button on your repository and on VS Code either clone repository on the Welcome screen or open the Command Palette (ctrl+shift+P), type `git clone` and select `Git: Clone`
   (a) Select the parent directory for your project

   (b) Open the cloned repository either as prompted or by adding the newly created folder to your workspace by the Welcome tab's Open folder link or File -> Add Folder to Workspace
   (c) If you use the command line `git clone` the authentications for pushing to your online repository are not set up
7. If you wait a bit it should ask you which kit you want to use (at the time of this writing I typically use GCC 9.3.0)
8. Allow Intellisense if prompted
9. Edit README.md to reflect your new project
10. Edit the `project` line in the CMakeLists.txt file to have your project's name and version
11. Edit the `add_executable` line in the CMakeLists.txt file to change the name of the executable file to something relevant
12. Change the @brief, @details, @author, and @date in src/main.cpp
13. To create the PDF on a standard Ubuntu install, the following need to be added: `sudo apt install graphviz texlive-latex-base texlive-latex-recommended texlive-latex-extra`
14. Doxygen also needs installing: `sudo apt install doxygen`
15. In the terminal, change to the `build` directory (should have been automatically generated)
16. Run the following:
   (a) `make`
   (b) `make docs`
   (c) `make pdf`
17. Add the newly named PDF to git staging (`git status` -> `git add docs/yourprojectname.pdf`)
18. Commit all the changes: `git commit -a -m "Initial commit"`
19. Push the changes to GitHub: `git push origin main`
20. Back at your repository on GitHub, refresh the page to show latest commit
21. In the Settings tab, scroll down to GitHub Pages
22. Select "Branch: main" as source and "/docs" as the folder and then press Save
23. Scroll back down to GitHub Pages and click the link to the published site
24. You now have a C++ repository with doxygen output hosted on GitHub Pages
   (a) The link usually doesn't work for a while (minutes to hours). This can be worked around by adding `index.html` to the end of the URL. A second commit will also fix it once the commit propagates over to GitHub Pages.
   (b) You can see the PDF file generated by Doxygen by adding the name of the PDF to the end of the URL. It will be of the form `projectname.pdf` and can be seen in the `docs` folder.
   (c) It can take a few minutes for a new `git push origin main` to propagate over to GitHub Pages
25. Edit README.md to reflect your project usage and point to the Doxygen output for your project
26. Stage the commit (`git add README.md`)
27. Commit (`git commit -a -m "Describe your changes here"`)
28. Push your changes to GitHub as before (`git push origin main`)

1.1.2 General Usage

During normal development, you will change main.cpp, maybe add more files in the src directory, make them, and run them. To update the documentation on the web do the following at a terminal prompt in your project's `build` directory:

1. `make`
2. `make docs`
3. `make pdf` Then in your project's root directory do the following:
4. Check the git status: `git status`
5. `git commit -a -m "Describe your changes since last commit"`
   (a) The `-a` flag is used to commit all the updated documentation files
   (b) VS Code also has git built into it, but the use of branches isn't as easy a workflow as the commandline offers for me (personal opinion).
6. Note that in order for numbered (ordered) lists to work across markdown and Doxygen HTML and PDF outputs they are explicitly numbered vs markdown all being 1. or Doxygen's -#.

1.2 References

1. https://www.doxygen.nl/manual/docblocks.html
2. https://stackoverflow.com/questions/44212101/cmake-how-to-have-add-custom-command-r
3. Very useful overview: https://caiorss.github.io/C-Cpp-Notes/Doxygen-documentation.↩ html
4. https://devblogs.microsoft.com/cppblog/clear-functional-c-documentation-with-sphinx
5. https://vicrucann.github.io/tutorials/quick-cmake-doxygen/
6. https://medium.com/practical-coding/c-documentation-with-doxygen-cmake-sphinx-breat
7. https://stackoverflow.com/questions/18590445/cmake-custom-command-to-copy-and-renam

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Graph Class Reference

Public Member Functions

- Graph (vector< int > a, int input)
- void addEdge (vector< int > adj, int u, int v)
- void delEdge (vector< int > adj, int u, int v)
- bool hasEdge (int i, int j)
- void outEdges (int i, vector< int > &edges)
- void inEdges (int i, vector< int > &edges)
- int nVertices ()
- void bfs (Graph &g, int r)
- void dfs2 (Graph &g, int r)
- void printGraph (vector< int > adj, int V)

Public Attributes

- vector< int > adj

4.1.1 Detailed Description

Definition at line 17 of file main.cpp.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Graph()

```
Graph::Graph (
            vector< int > a,
            int input )  [inline]
```

Definition at line 23 of file main.cpp.

```
23                                  {
24         n = input;
25         adj = a;
26     }
```

4.1.3 Member Function Documentation

4.1.3.1 addEdge()

```
void Graph::addEdge (
            vector< int > adj,
            int u,
            int v )  [inline]
```

Definition at line 28 of file main.cpp.

```
29 {
30     adj.push_back(v);
31     adj.push_back(u);
32 }
```

4.1.3.2 bfs()

```
void Graph::bfs (
            Graph & g,
            int r )  [inline]
```

Definition at line 82 of file main.cpp.

```
82                          {
83     bool *seen = new bool[g.nVertices()];
84     vector<int> q;
85     q.push_back(r);
86     seen[r] = true;
87     while (q.size() > 0) {
88         int i = q.back();
89         cout « endl « i « " > " « "This is BFS" « endl;
90         q.pop_back();
91         vector<int> edges;
92         g.outEdges(i, edges);
93         for (int k = 0; k < edges.size(); k++) {
94             int j = edges[k];
95             if (!seen[j]) {
96                 q.push_back(j);
97                 seen[j] = true;
98             }
99         }
100     }
101     delete[] seen;
102 }
```

4.1.3.3 delEdge()

```
void Graph::delEdge (
            vector< int > adj,
            int u,
            int v )  [inline]
```

Definition at line 33 of file main.cpp.

```
34 {
35     // Traversing through the first vector list
36     // and removing the second element from it
37     for (int i = 0; i < adj.size(); i++) {
38         if (adj[i] == v) {
```

```
39              adj.erase(adj.begin() + i);
40              break;
41          }
42      }
43
44      // Traversing through the second vector list
45      // and removing the first element from it
46      for (int i = 0; i < adj.size(); i++) {
47          if (adj[i] == u) {
48              adj.erase(adj.begin() + i);
49              break;
50          }
51      }
52  }
```

4.1.3.4 dfs2()

```
void Graph::dfs2 (
              Graph & g,
              int r )  [inline]
```

Definition at line 106 of file main.cpp.

```
106                                 {
107      bool *c = new bool[g.nVertices()];
108      vector<int> s;
109      s.push_back(r);
110      while (s.size() > 0) {
111          int i = s.back();
112          cout << endl << i << " > " << "This is DFS"<< endl;
113          s.pop_back();
114          if (c[i] == *c) {
115              c[i] = c;
116              vector<int> edges;
117              g.outEdges(i, edges);
118              for (int k = 0; k < edges.size(); k++)
119                  s.push_back(edges[k]);
120          }
121      }
122  delete[] c;
123  }
```

4.1.3.5 hasEdge()

```
bool Graph::hasEdge (
              int i,
              int j )  [inline]
```

Definition at line 55 of file main.cpp.

```
55                                  {
56      vector<int>::iterator it;
57      it = find (adj.begin(), adj.end(), i);
58      if (it != adj.end())
59          return adj[i];
60  }
```

4.1.3.6 inEdges()

```
void Graph::inEdges (
            int i,
            vector< int > & edges )  [inline]
```

Definition at line 68 of file main.cpp.

```
68                                          {
69     for (int j = 0; j < n; j++) {
70         vector<int>::iterator it;
71         it = find (adj.begin(), adj.end(), i);
72         if (it != adj.end())
73                 edges.push_back(j);
74     }
75 }
```

4.1.3.7 nVertices()

```
int Graph::nVertices ( )  [inline]
```

Definition at line 77 of file main.cpp.

```
77                  {
78     return n * n;
79 }
```

4.1.3.8 outEdges()

```
void Graph::outEdges (
            int i,
            vector< int > & edges )  [inline]
```

Definition at line 63 of file main.cpp.

```
63                                           {
64     for (int k = 0; k < adj.size(); k++)
65         edges.push_back(adj.at(k));
66 }
```

4.1.3.9 printGraph()

```
void Graph::printGraph (
            vector< int > adj,
            int V )  [inline]
```

Definition at line 125 of file main.cpp.

```
126 {
127     int x;
128     for (int v = 0; v < V; ++v) {
129         cout « "vertex " « v « " ";
130         for (auto x : adj)
131             cout « "-> " « x;
132         printf("\n");
133         break;
134     }
135     printf("\n");
136 }
```

4.1.4 Member Data Documentation

4.1.4.1 adj

```
vector<int> Graph::adj
```

Definition at line 21 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/bona/CPTR227/adjacency-graph/src/main.cpp

Chapter 5

File Documentation

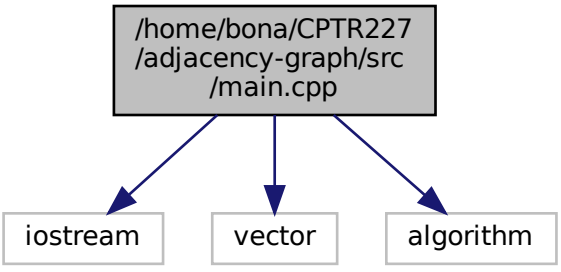
5.1 /home/bona/CPTR227/adjacency-graph/README.md File Reference

5.2 /home/bona/CPTR227/adjacency-graph/src/main.cpp File Reference

This is a graph project.

```
#include <iostream>
#include <vector>
#include <algorithm>
```

Include dependency graph for main.cpp:



Classes

- class Graph

Functions

- int main ()

5.2.1 Detailed Description

This is a graph project.

This is the long brief at the top of main.cpp.

Author

Addis Bogale and Bona Tufa

Date

4/2/2021

5.2.2 Function Documentation

5.2.2.1 main()

```
int main ( )
```

Definition at line 139 of file main.cpp.

```
140 {
141     int v;
142     vector<int> value;
143 
144     Graph test = Graph(value, 10);
145 
146     // Adding edge as shown in the example figure
147 
148 
149     test.addEdge(value, 0, 4);
150     test.addEdge(value,1, 2);
151     test.addEdge(value,1, 3);
152     test.addEdge(value,1, 4);
153     test.addEdge(value,2, 3);
154     test.addEdge(value,3, 4);
155 
156     // Printing adjacency matrix
157     test.printGraph(value, v);
158 
159     // Deleting edge (1, 4)
160     // as shown in the example figure
161     test.delEdge(value, 1, 4);
162 
163     // Printing adjacency matrix
164     test.printGraph(value, v);
165 
166     return 0;
167 }
```

Index