

CXX01

# Graph library documentatie

Advanced C Programming

Rene Schouten (0928619)

Raber Ahmad (0921954)

## Interne structuur

Een graph bestaat uit een gelinkte lijst van vertexes. Een vertex is een node in de graph.

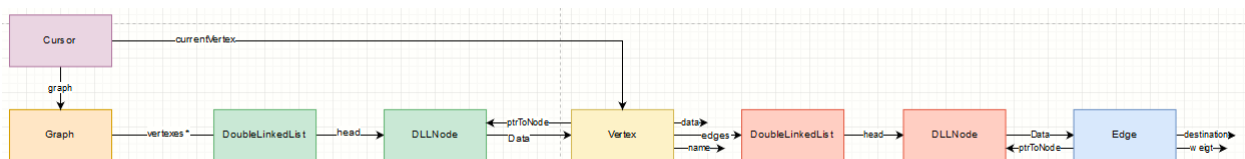
```
typedef struct
{
    DoublyLinkedList *vertices;
} Graph;
```

Een Vertex object heeft een naam, data van de gebruiker, en een lijst met edges, deze edges zijn verbindingen naar andere vertexes. Ook heeft de vertex een pointer naar de DLLNode van de doublelinkedlist dat de library nodig heeft voor het correct verwijderen van vertexes.

De naam van een vertex wordt bij de initialisatie van de vertex gekopieerd en de library is verantwoordelijk voor het vrijmaken van dat geheugen. De data van een vertex is eigendom van de gebruiker en zal niet door de graph library vrijgemaakt worden! Dit omdat het ook geheugen van de stack kan zijn

```
typedef struct
{
    char* name;
    void* data;
    DoublyLinkedList *edges;
    DllNode* ptrToNode;
} Vertex;
```

Een edge is een object dat verwijst naar een andere vertex(destination) en heeft een gewicht(weight) wanneer het gewicht niet gebruikt wordt is het gewicht 0. Ook de edge heeft een pointer naar de DLLNode zodat de library het object goed kan verwijderen.



## API

```
Graph* createGraph(void);
```

Return een graph object, deze graph is nog helemaal leeg.

```
void graphDelete(Graph** ptrToDeleteGraph);
```

verwijdert de graph met alle vertexes en edges uit het geheugen en zet de pointer op NULL. Let op de gebruikersdata word niet verwijderd.

```
Vertex* addVertex(Graph *graph, char* name, void* data);
```

```
//delete a vertex and set the pointer to NULL
```

```
void deleteVertex(Graph *graph, Vertex** ptrToDeleteVertex);
```

```
//returns the number of vertexes in a given graph
```

```
size_t numberOfVertexes(Graph* graph);
```

```
//creates a edge(connection) between vertexes, which can be both directed or undirected
```

```
Edge* createEdge(Vertex* from, Vertex* destination, bool directed);
```

```

//creates a edge(connection) between vertexes, which can be both directed or
undirected. also a weight is added
Edge* createEdgeWithWeight(Vertex* from, Vertex* destination, int weight, bool
directed);

//delete a edge
void deleteEdge(Graph* graph, Edge *toDeleteEdge, Vertex* connectedVertex);

//print a vertex and its connections
void vertexPrintConnections(Graph* graph, Vertex* pointOfView);

//returns the first vertex with the given name, if it don't exist it will
return NULL
Vertex* searchVertexByName(Graph* graph, char* name);

//delete all vertexes and edges
void clear(Graph* graph);

//clear all vertexes that are not connected
void clearFloatingVertexes(Graph* graph);

```

## De cursor library

Een gebruiker van de graph zou niet de interne structuur van de library met doublelinkedlists hoeven kennen, om zonder die kennis door de graph te navigeren is een cursor volgens het cursor design pattern gemaakt.

Een cursor staat op een bepaalde plek in de graph en er van vertex naar vertex door heen navigeren.

```

typedef struct
{
    Graph* graph;
    Vertex* currentVertex;
} GraphCursor;

```

## Cursor API

```
GraphCursor* createCursor(void);
```

Maakt een cursor, deze cursor staat nog niet op een plek in de graph.

```
GraphCursor* copyGraphCursor(GraphCursor* cursor);
```

Kopieer een bestaande cursor.

```
void deleteCursor(GraphCursor** ptrToCursor);
```

verwijder een cursor en zet de pointer naar NULL.

```
void cursorSetCurrentVertex(GraphCursor *cursor, Graph* graph, Vertex*
vertex);
```

zet de cursor op een bepaalde vertex in een graph.

```
int cursorAvailable(GraphCursor* cursor);
```

returnt hoeveel edges de vertex heeft. Wanneer de cursor niet op een vertex staat return het 0.

```
Edge* cursorEdgeAt(GraphCursor* cursor, int index);
```

Return een edge op de zoveelste positive van de vertex. Wanneer het getal groter is dan cursorAvailable() word NULL gereturt.

```
Vertex* cursorAt(GraphCursor* cursor, int index);
```

Return een vertex op de zoveelste positive van de vertex. Wanneer het getal groter is dan cursorAvailable() word NULL gereturt.

```
void cursorMoveInto(GraphCursor* cursor, int index);
```

zet de cursor zoveelste egde positie van de vertex.

```
Vertex* cursorGetCurrentVertex(GraphCursor* cursor);
```

Return de huidige vertex waar de cursor op staat

```
void* cursorGetCurrentData(GraphCursor* cursor);
```

Return de huidige data van de vertex waar de cursor op staat

```
void cursorDeleteCurrentVertex(GraphCursor *cursor);
```

verwijderd de vertex waar de cursor op staat en zet de de cursor positive op NULL.

```
void cursorDeleteEdgeAt(GraphCursor *cursor, int index);
```

verwijderd een edge op een bepaalde index

## Valgrind en Tests

Hieronder zien we alle testresultaten van de Graph als die van de doublelinkedlist

```
cx01-samenwerking -- raber@rabers-mbp -- --samenwerking -- -zsh -- 80x...
- cx01-samenwerking git:(master) * ./a.out
Running test suite with seed 0xcb42d7e...
DLL/graphCreate [OK] [ 0.00001000 / 0.00000800 CPU ]
DLL/graphDelete Empty List [OK] [ 0.00001200 / 0.00000800 CPU ]
DLL/addVertex test [OK] [ 0.00001700 / 0.00001500 CPU ]
DLL/deleteVertex test [OK] [ 0.00001400 / 0.00001100 CPU ]
DLL/numberOfVertices test [OK] [ 0.00001300 / 0.00001100 CPU ]
DLL/createEdge test [OK] [ 0.00001400 / 0.00001300 CPU ]
DLL/createEdge with weight test [OK] [ 0.00001500 / 0.00001100 CPU ]
DLL/deleteEdge undirected test [OK] [ 0.00001500 / 0.00001100 CPU ]
DLL/deleteEdge directed test [OK] [ 0.00001300 / 0.00001000 CPU ]
DLL/print connections test [OK] [ 0.00002000 / 0.00002000 CPU ]
  Connected vertices "vertex2"
  [ 0.00002100 / 0.00001800 CPU ]
DLL/searchVertexByName test [OK] [ 0.00001300 / 0.00001100 CPU ]
DLL/graph clear test [OK] [ 0.00001700 / 0.00001300 CPU ]
13 of 13 (100%) tests successful, 0 (0%) test skipped.
- cx01-samenwerking git:(master) *
```

```
cx01-samenwerking -- raber@rabers-mbp -- --samenwerking -- -zsh -- 112x39
- cx01-samenwerking git:(master) * ./a.out
Running test suite with seed 0x7e8e39fe...
DLL/dllCreate [OK] [ 0.00001100 / 0.00000800 CPU ]
DLL/dllDelete Empty List [OK] [ 0.00000500 / 0.00000200 CPU ]
DLL/dllAddBeforeHead in empty list [OK] [ 0.00001300 / 0.00000900 CPU ]
DLL/dllAddBeforeHead list 1 element [OK] [ 0.00001300 / 0.00001000 CPU ]
DLL/dllAddBeforeHead list 2 elements [OK] [ 0.00001200 / 0.00000900 CPU ]
DLL/dllAddBeforeHead list with NULL element [OK] [ 0.00001300 / 0.00001000 CPU ]
DLL/dllAddAfterTail in empty list [OK] [ 0.00001200 / 0.00000800 CPU ]
DLL/dllAddAfterTail list 1 element [OK] [ 0.00001200 / 0.00000900 CPU ]
DLL/dllAddAfterTail list 2 elements [OK] [ 0.00001200 / 0.00000900 CPU ]
DLL/dllAddAfterTail list with NULL element [OK] [ 0.00001300 / 0.00001000 CPU ]
DLL/dllAddAfter Test [OK] [ 0.00000800 / 0.00000400 CPU ]
DLL/dllDeleteBefore Test [OK] [ 0.00000900 / 0.00000600 CPU ]
DLL/dllDeleteNodes list 1 element [OK] [ 0.00000700 / 0.00000400 CPU ]
DLL/dllDeleteNodes list 2 elements [OK] [ 0.00001200 / 0.00000900 CPU ]
DLL/dllDeleteNodes list 3 elements [OK] [ 0.00000800 / 0.00000500 CPU ]
DLL/dllDeleteNodes with NULL element [OK] [ 0.00000800 / 0.00000600 CPU ]
DLL/dllNumberOfElements [OK] [ 0.00000500 / 0.00000200 CPU ]
  listlength=0 [OK] [ 0.00000500 / 0.00000200 CPU ]
  listlength=1 [OK] [ 0.00000500 / 0.00000200 CPU ]
  listlength=2 [OK] [ 0.00000700 / 0.00000200 CPU ]
  listlength=12 [OK] [ 0.00000500 / 0.00000200 CPU ]
DLL/dllFindFirst [OK] [ 0.00000800 / 0.00000500 CPU ]
DLL/dllFindFirstNULL [OK] [ 0.00000500 / 0.00000400 CPU ]
DLL/dllFindFirstDouble [OK] [ 0.00000700 / 0.00000500 CPU ]
DLL/dllFindLast [OK] [ 0.00000900 / 0.00000700 CPU ]
DLL/dllFindBefore [OK] [ 0.00001000 / 0.00000800 CPU ]
DLL/dllFindAfter [OK] [ 0.00001100 / 0.00000800 CPU ]
DLL/dllDelete Filled [OK] [ 0.00000400 / 0.00000200 CPU ]
  listlength=0 [OK] [ 0.00000400 / 0.00000000 CPU ]
  listlength=1 [OK] [ 0.00000400 / 0.00000200 CPU ]
  listlength=2 [OK] [ 0.00000500 / 0.00000200 CPU ]
  listlength=12 [OK] [ 0.00000500 / 0.00000200 CPU ]
DLL/dllTheListNull [OK] [ 0.00000700 / 0.00000600 CPU ]
DLL/dllTheNodesNull [OK] [ 0.00000700 / 0.00000500 CPU ]
DLL/dllPredicateNull [OK] [ 0.00000700 / 0.00000400 CPU ]
23 of 23 (100%) tests successful, 0 (0%) test skipped.
- cx01-samenwerking git:(master) *
```

Linkst staan de testen van de graph en rechts staan de testen van de doublelinkedlist. Zoals te zien zijn beide testen 100% geslaagd.

De graph is op de volgende punten getest:

- NULL pointers
- Creëren van een graph
- Verwijderen van een graph
- Toevoegen en verwijderen van een vertex
- Retourneren van de juiste aantal Vertexes

- Creëren van een edge met en zonder weigt
- Verwijderen van een edge voor zwel directed als undirected

```

==5932== by 0x100000: createEdge (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x100588: cursorTest (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x100770: munit_test_runner_exec (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007109: munit_test_runner_run_test_with_params (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007613: munit_test_runner_run_test (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007792: munit_test_runner_run_suite (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1100776: munit_test_runner_run (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x111240: munit_suite_main_custom (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x111407: munit_suite_main (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x100701: main (in /home/manjaro/cxx01-samenwerking/a.out)
==5932==
==5932== 56 (24 direct, 32 indirect) bytes in 1 blocks are definitely lost in loss record 10 of 10
==5932== at 0x400777: malloc (vg_replace_malloc.c:309)
==5932== by 0x100702: createEdgeWithWeight (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x10060C: createEdge (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x100774: cursorTest (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007C7C: munit_test_runner_exec (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007109: munit_test_runner_run_test_with_params (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007613: munit_test_runner_run_test (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1007792: munit_test_runner_run_suite (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x1100776: munit_test_runner_run (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x111240: munit_suite_main_custom (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x111407: munit_suite_main (in /home/manjaro/cxx01-samenwerking/a.out)
==5932== by 0x100701: main (in /home/manjaro/cxx01-samenwerking/a.out)
==5932==
==5932== LEAK SUMMARY:
==5932==   definitely lost: 136 bytes in 6 blocks
==5932==   indirectly lost: 96 bytes in 3 blocks
==5932==   possibly lost: 0 bytes in 0 blocks
==5932==   still reachable: 21 bytes in 1 blocks
==5932==   suppressed: 0 bytes in 0 blocks
==5932== Reachable blocks (those to which a pointer was found) are not shown.
==5932== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5932==
==5932== For lists of detected and suppressed errors, rerun with: -s
==5932== ERROR SUMMARY: 6 errors from 6 contexts (suppressed: 0 from 0)
[ 0.0 ] [ 0.01893823 / 0.01953972 CPU ]
10 of 10 (100%) tests successful, 0 (0%) test skipped.
==5919==
==5919== HEAP SUMMARY:
==5919==   in use at exit: 0 bytes in 0 blocks
==5919==   total heap usage: 27 allocs, 27 frees, 10,759 bytes allocated
==5919==
==5919== All heap blocks were freed -- no leaks are possible
==5919==
==5919== For lists of detected and suppressed errors, rerun with: -s
==5919== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
manjaro@manjaro-arch:~$ cd /home/manjaro/cxx01-samenwerking/;

```

```

==5981== For lists of detected and suppressed errors, rerun with: -s
==5981== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[ 0.0 ] [ 0.00590902 / 0.00463737 CPU ]
DLL/dllTheNodeNull
==5982==
==5982== HEAP SUMMARY:
==5982==   in use at exit: 19 bytes in 1 blocks
==5982==   total heap usage: 64 allocs, 63 frees, 20,570 bytes allocated
==5982==
==5982== LEAK SUMMARY:
==5982==   definitely lost: 0 bytes in 0 blocks
==5982==   indirectly lost: 0 bytes in 0 blocks
==5982==   possibly lost: 0 bytes in 0 blocks
==5982==   still reachable: 19 bytes in 1 blocks
==5982==   suppressed: 0 bytes in 0 blocks
==5982== Reachable blocks (those to which a pointer was found) are not shown.
==5982== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5982==
==5982== For lists of detected and suppressed errors, rerun with: -s
==5982== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[ 0.0 ] [ 0.00192359 / 0.00177487 CPU ]
DLL/dllIPredicateNull
==5983==
==5983== HEAP SUMMARY:
==5983==   in use at exit: 21 bytes in 1 blocks
==5983==   total heap usage: 67 allocs, 66 frees, 21,103 bytes allocated
==5983==
==5983== LEAK SUMMARY:
==5983==   definitely lost: 0 bytes in 0 blocks
==5983==   indirectly lost: 0 bytes in 0 blocks
==5983==   possibly lost: 0 bytes in 0 blocks
==5983==   still reachable: 21 bytes in 1 blocks
==5983==   suppressed: 0 bytes in 0 blocks
==5983== Reachable blocks (those to which a pointer was found) are not shown.
==5983== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5983==
==5983== For lists of detected and suppressed errors, rerun with: -s
==5983== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
[ 0.0 ] [ 0.00208619 / 0.00275692 CPU ]
10 of 10 (100%) tests successful, 0 (0%) test skipped.
==5958==
==5958== HEAP SUMMARY:
==5958==   in use at exit: 0 bytes in 0 blocks
==5958==   total heap usage: 65 allocs, 65 frees, 21,063 bytes allocated
==5958==
==5958== All heap blocks were freed -- no leaks are possible
==5958==
==5958== For lists of detected and suppressed errors, rerun with: -s
==5958== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
manjaro@manjaro-arch:~$ cd /home/manjaro/cxx01-samenwerking/;

```

Hierboven zien we screenshot van de valgrind resultaten. De linker is het resultaat van de dllist+ graph en die van de rechter is het valgrind resultaat van alleen de dllist.

We kunnen zien dat de dll alleen geen memory-leaks alles wat wordt gealloceerd wordt ook weer verwijderd. We zien staan dat bij defintely lost, 0 bytes staan.

De graph samen met dll geeft wel aan dat er een memory leak is. We kunnen dat ook weer zien bij defintely lost in de linker afbeelding. Er staat namelijk dat er 136 bytes verloren zijn gegaan in 6 blokken.