Dynamische Container C

Dokumentation

Lukas Momberg []

Dennis goßler [11140150]

Inhaltsverzeichnis

[Collection Error Codes 2](#_Toc82684874)

[List 3](#_Toc82684875)

[Dictionary 3](#_Toc82684876)

[Stack 4](#_Toc82684877)

[Queue 4](#_Toc82684878)

[LinkedList 5](#_Toc82684879)

[String 5](#_Toc82684880)

# Collection Error Codes

Wenn eine Funktion auf eine Collection fehlschlägt, können Folgende Fehlercodes zurückgegeben:

CollectionError

{

// Successful / No Error

CollectionNoError,

// Some internal allocation failed

CollectionOutOfMemory,

// There is no data to pull.

CollectionEmpty,

// Specified index points not to any data.

CollectionArrayIndexOutOfBounds,

// Specified 'ElementSize' was Zero.

CollectionNoElementSizeSpecified,

// Specified 'data' is Null-Pointer

CollectionElementIsNullPointer,

//Dictionary specific

CollectionKeyAlreadyExists

}

# List

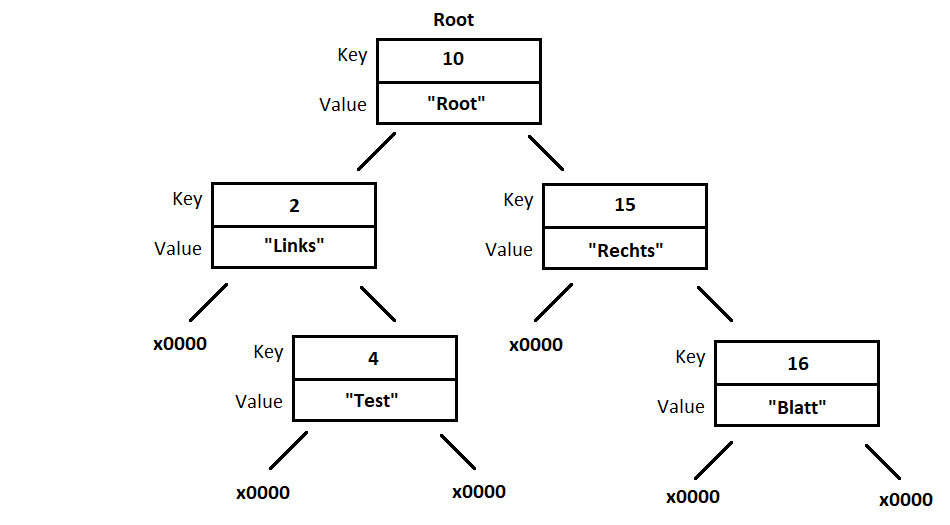
Nach der Initialisierung der Liste, könne Sie Daten mit der Add Funktion hintereinander speichern. Sollte die maximale Größe wird die Liste automatisch erweitert.

[]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Schritt** | **Pos .0** | **Pos.1** | **Pos.2** | **Pos.3** | **Pos.4** | **…** | **Eingabe** | **Ausgabe** |
| 0 – Voher | Null |  |  |  |  |  | - | - |
| 1 – Allocation | \0 | \0 | \0 | \0 |  | … | 4 | - |
| 2 – ADD | A | \0 | \0 | \0 |  | … | A | - |
| 3 – INSERT | A | \0 | D | \0 |  | … | 2, D | - |
| 4 – ADD | A | C | D | \0 |  | … | C | - |
| 5 – GET | A | C | D | \0 |  | … | 2 | D |
| 6 – REMOVE | A | C | \0 | \0 |  | … | 2 | - |
| 7 – GET | A | C | \0 | \0 |  |  | 2 | NULL |
| 7 – Deconstruct | Null |  |  |  |  |  | - | - |

# Dictionary

Das Dictionary wurde als ein Binärer Suchbaum implementiert. Dieser besitzt pro Eintrag einen „Key“ und eine „Value“. Beim Einfügen wird von der Wurzel ab entschieden, ob der jeweilige „Key“ größer oder kleiner ist. Wird eine passende Stelle gefunden wird der Datensatz dort gespeichert. „Key“ Duplikate sind nicht zulässig.



Abbildung

# Stack

Daten werden gestapelt, der ältere Wert wird vom neuern verdeckt. Es kann immer ein Wert hinzugefügt werden, beim Entfernen wird der neueste Wert entfernt. First In Last Out (FILO). Diese Vorgänge sind sehr schnell und sind generell sicher.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Schritt** | **0x01** | **0x02** | **0x03** | **0x04** | **0x05** | **…** | **Eingabe** | **Ausgabe** |
| 0 – Voher | Null |  |  |  |  |  | - | - |
| 1 – Allocation | \0 | \0 | \0 | \0 | \0 | … | - | - |
| 2 – Push | A | \0 | \0 | \0 | \0 | … | A | - |
| 3 – Push | A | B | \0 | \0 | \0 | … | B | - |
| 4 – Push | A | B | C | \0 | \0 | … | C | - |
| 5 – Pull | A | B | \0 | \0 | \0 | … | - | C |
| 6 – Pull | A | \0 | \0 | \0 | \0 | … | - | B |
| 7 – Pull | Null |  |  |  |  |  | - | A |

# 

# Queue

Daten werden in einer Liste gespeichert, der älteste Wert wird hier entnommen. First in First out (FIFO). Da Daten von Vorne entnommen werden entsteht ungenutzter Speicher. Hier ist zu Achten, dass dieser Speicher möglichst freigegeben wird, natürlich ist eine umbauen des Speichers bei jedem Zugriff nicht unbedingt Sinnvoll.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Schritt** | **0x01** | **0x02** | **0x03** | **0x04** | **0x05** | **…** | **Eingabe** | **Ausgabe** |
| 0 – Voher | Null |  |  |  |  |  | - | - |
| 1 – Allocation | \0 | \0 | \0 | \0 | \0 | … | - | - |
| 2 – Push | A | \0 | \0 | \0 | \0 | … | A | - |
| 3 – Push | A | B | \0 | \0 | \0 | … | B | - |
| 4 – Push | A | B | C | \0 | \0 | … | C | - |
| 5 – Pull | A | B | C | \0 | \0 | … | - | A |
| 6 – Pull | A | B | C | \0 | \0 | … | - | B |
| 7 – Pull | Null |  |  |  |  |  | - | C |

# LinkedList

Daten werden in Ketten-Elementen gespeichert. Jedes zwischen Element kennt seinen nächsten Nachbarn. Durch diese Kette kann man jedes Element ansprechen. Das Letze Element hat immer einen Null Wert, da dieser der Letze Wert ist und keinen nächsten Wert besitzt.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Schritt** | **Nr.1** | **Nr.2** | **Nr.3** | **Nr.4** | **Nr.5** | **…** | **Eingabe** | **Ausgabe** |
| 1 | Null |  |  |  |  |  | - | - |
| Null |  |  |  |  | … |
| 2 - Push | A |  |  |  |  | … | A | - |
| Null |  |  |  |  | … |
| 3 - Push | A |  |  |  |  | … | B | - |
| -> | B |  |  |  | … |
|  | Null |  |  |  | … |
| 4 - Push | A |  |  |  |  | … | C | - |
| -> | B |  |  |  | … |
|  | -> | C |  |  | … |
|  |  | Null |  |  | … |
| 5 – Pull(1) | A |  |  |  |  | … | - | B |
| -> | C |  |  |  | … |
|  | Null |  |  |  | … |

# String

Diese Collection soll die Erweiterbarkeit unserer Container verdeutlichen. Dieser Container benutzt den Listcontainer als Basis und erweitert dessen Möglichkeiten. Zum Beispiel ist es möglich an den vorhandenen String ein Zeichen anzuhängen oder einen ganzen String.

„Das ist ein test“ + ‚!‘ -> „Das ist ein test!“ + „TEST“ -> „Das ist ein test!TEST“

StringAdd(); StringConcat(); StringGetFullString();