Dynamische Container C

Dokumentation

Lukas Momberg [11141259]

Dennis goßler [11140150]

Inhaltsverzeichnis

[Daten Container in C 2](#_Toc82826586)

[Basic Containers 2](#_Toc82826587)

[Generelle Container 2](#_Toc82826588)

[Zusätzliche Tests 3](#_Toc82826589)

[Kompilierung 3](#_Toc82826590)

[Getestete Systeme 3](#_Toc82826591)

[Wie man Kompiliert 3](#_Toc82826592)

[Windows 3](#_Toc82826593)

[Collection Error Codes 4](#_Toc82826594)

[List 5](#_Toc82826595)

[List - Funktionen: 5](#_Toc82826596)

[Codebeispiel: 6](#_Toc82826597)

[Dictionary 7](#_Toc82826598)

[Dictionary - Funktionen: 7](#_Toc82826599)

[Codebeispiel: 8](#_Toc82826600)

[Stack 9](#_Toc82826601)

[Stack - Funktionen: 9](#_Toc82826602)

[Codebeispiel: 10](#_Toc82826603)

[Queue 11](#_Toc82826604)

[Queue- Funktionen: 11](#_Toc82826605)

[Codebeispiel: 12](#_Toc82826606)

[LinkedList 13](#_Toc82826607)

[LinkedList - Funktionen: 13](#_Toc82826608)

[Codebeispiel: 14](#_Toc82826609)

[String 15](#_Toc82826610)

[String - Funktionen: 15](#_Toc82826611)

[Codebeispiel: 16](#_Toc82826612)

Daten Container in C

Um Daten zu speichern werden Container benötigt. Dieses speichern in Container kann ziemlich nervig sein, also haben wir ein paar Lösungen in der Programmiersprache C entwickelt.

Jede moderne Programmiersprache hat heutzutage schon irgendeine Art von Generischen Typen, oft gekennzeichnet als T für Type. Dann haben wir die Programmiersprache C. Klassisches Altes-C hat dieses Funktion leider nicht, es wurde jedoch in C11 in gewisser Weise hinzugefügt.

Wie können wir trotzdem eine Generische Liste in älteren Versionen von C nutzen? In diesem Projekt geht es um diese Frage.

Basic Containers

Basis Container sind oft in Programmiersprachen integriert, da sie fundamentale Funktionen bieten.

* Stack (Stapel)
* Queue (Warteschlangen)
* List (Listen)
* LinkedList (Verkettete Listen)
* Dictionary (Wörterbücher)

Generelle Container

Dann haben wir mehr generelle Container die etwas genauer sind. Diese sind auch oft in Programmier-Sprachen enthalten

* String (Zeichenketten)
* NumberLists (Numerische Listen)
* ...

Zusätzliche Tests

Natürlich brauchten wir auch Tests um zu schauen ob die Container auch wie erwartet funktionieren. Um dies zu überprüfen haben wir extra Code angefertigt und Abläufe genau beobachtet.

Kompilierung

Getestete Systeme

Der Code wurde in Visual Studio unter Windows 10 x86 und x64 kompiliert

Zusätzlich wurde der Code unter Linux mint & Ubuntu x86 und x64 getestet. Das hat zusätzlich eine extra makefile benötigt. Diese wurde neben der Visual Studio Projektdatei hinzugelegt.

Wie man Kompiliert

Windows

Öffne das Projekt in Visual Studio und klicke das Test-Projekt und wähle es als Start-Projekt aus. Ohne diese Einstellung wird möglicherweise versuch eine .lib Datei zu öffnen, dort wird dann ein Fehler angezeigt.

*Linux*

Navigiere zu der makefile. Dort befindet sich auch ein Skript das zum kompilieren genutzt werden kann. Es erstellt einen bin Order für Temporäre Daten, danach kompiliert es und startet das fertige Programm danach.

Vergiss nicht die Skript-Datei ausführbar zu machen mit chmod +x FileName.sh. Dann zum Öffnen ./Filename.sh

# 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 |  |  |  |  |  | - | - |

## List - Funktionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktionsname | Rückgabewert | Parameter | Nutzen |
| ListInitialize | void | List\* list, unsigned int count,  size\_t sizeOfSingleElement | Inizialisierung des Containers |
| ListDestruction | void | List\* list | Gibt Speicher frei |
| ListItemInsertAt | CollectionError | List\* list, unsigned int indexValue, void\* value | Fügt/ ersetzt das Element beim Index |
| ListItemAdd | CollectionError | List\* list, unsigned int index,  void\* out | Fügt Wert zur Liste hinzu |
| ListItemGet | CollectionError | List\* list, void\* value | Nimmt einen Wert aus dem Container |
| ListItemRemove | CollectionError | List\* list, unsigned int index | Entfernt ein Wert und |
| ListClear | CollectionError | List\* list | Setzt alle Werte auf NULL |

## Codebeispiel:



# 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.

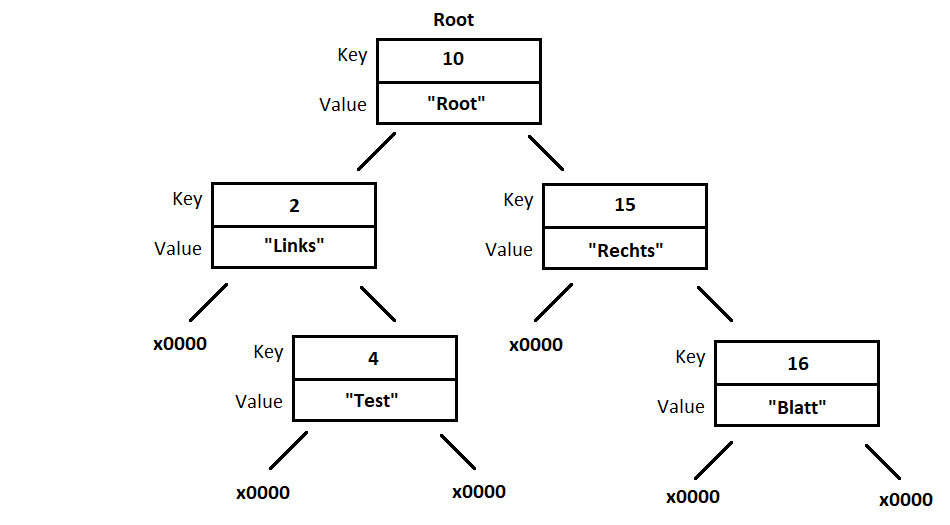


Abbildung 1

## 

## Dictionary - Funktionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktionsname | Rückgabewert | Parameter | Nutzen |
| DictionaryInitialize | void | Dictionary\* dictionary, unsigned int sizeOfKey, unsigned int sizeOfValue | Inizialisierung des Containers |
| DictionaryDestruction | void | Dictionary\* list | Gibt Speicher frei |
| DictionaryContainsKey | char | Dictionary\* dictionary,  void\* key | Gibt 0 / 1 zurück, ob Item vorhanden ist |
| DictionaryAdd | CollectionError | Dictionary\* dictionary,  void\* key, void\* value | Fügt Wert zum Dictionary hinzu |
| DictionaryGet | CollectionError | Dictionary\* dictionary,  void\* key, void\* out | Nimmt einen Wert aus dem Container |
| DictionaryRemove | CollectionError | Dictionary \* list, unsigned int index | Entfernt ein Wert und gibt ihn frei |

## Codebeispiel:



# Stack

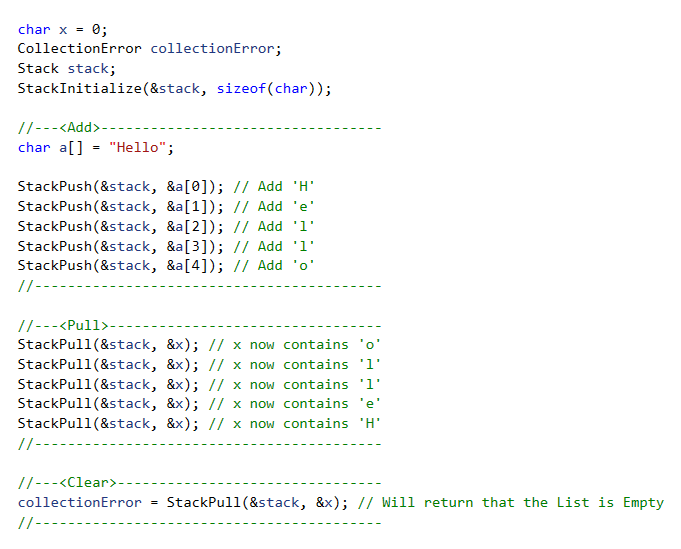
Daten werden gestapelt, der ältere Wert wird vom neuern verdeckt. Es kann immer ein Wert hinzugefügt werden, beim Entfernen wird der neueste Wert entfernt. First In Last Out (FILO). Diese Vorgänge sind sehr schnell und sind generell sicher.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 |

## Stack - Funktionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktionsname | Rückgabewert | Parameter | Nutzen |
| StackInitialize | void | Stack\* stack  unsigned int sizeOfSingleElement | Inizialisierung des Containers |
| StackClear | void | Stack\* stack | Löschen aller Daten des Containers |
| StackPush | CollectionError | Stack\* stack  void\* element | Fügt Wert zum Container hinzu |
| StackPull | CollectionError | Stack\* stack  void\* element | Nimmt einen Wert aus dem Container |

## Codebeispiel:



# Queue

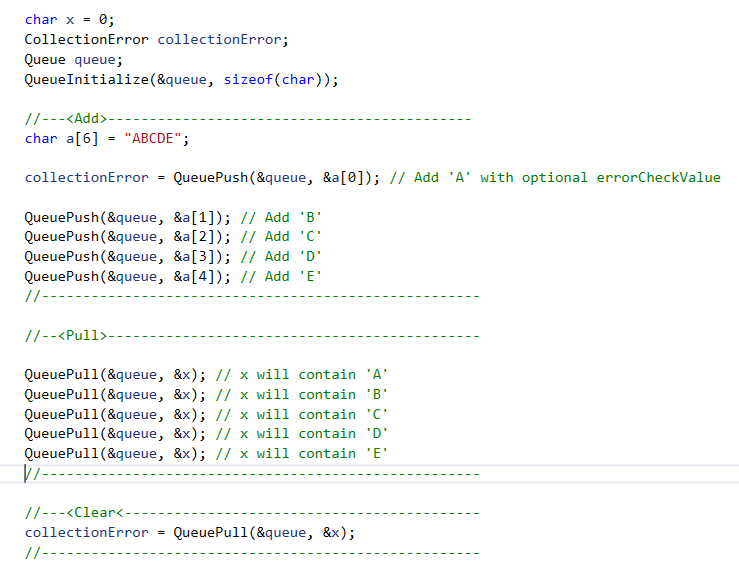
Daten werden in einer Liste gespeichert, der älteste Wert wird hier entnommen. First in First out (FIFO). Da Daten von Vorne entnommen werden entsteht ungenutzter Speicher. Hier ist zu Achten, dass dieser Speicher möglichst freigegeben wird, natürlich ist eine umbauen des Speichers bei jedem Zugriff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

## Queue- Funktionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktionsname | Rückgabewert | Parameter | Nutzen |
| QueueInitialize | void | Queue\* queue  unsigned int sizeOfSingleElement | Inizialisierung des Containers |
| QueueClear | void | Queue\* queue | Löschen aller Daten des Containers |
| QueuePush | CollectionError | Queue\* queue  void\* element | Fügt Wert zum Container hinzu |
| QueuePull | CollectionError | Queue\* queue  void\* element | Nimmt einen Wert aus dem Container |

## Codebeispiel:



# 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 |  |  |  | … |

## LinkedList - Funktionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktionsname | Rückgabewert | Parameter | Nutzen |
| LinkedListInitialize | void | LinkedList\* linkedList  unsigned int sizeOfElement | Inizialisierung des Containers |
| LinkedListClear | void | LinkedList\* linkedList | Löschen aller Daten des Containers |
| LinkedListInsert | CollectionError | LinkedList\* linkedList  unsigned int index  void\* element | Fügt Wert zum Container hinzu |
| LinkedListAddToEnd | CollectionError | LinkedList\* linkedList  void\* element | Fügt Wert am Ende des Containers an |
| LinkedListRemoveAtIndex | CollectionError | LinkedList\* linkedList  unsigned int index  void\* element | Entnimmt Wert aus gegebener Position |
| LinkedListGetElement | CollectionError | LinkedList\* linkedList  unsigned int index  void\* element | Liest Wert aus gegebener Position |
| LinkedListGetNode | CollectionError | LinkedList\* linkedList  unsigned int index  LinkedListNode\*\* linkedListNode | Liest Knoten aus gegebener Position |
| LinkedListGetLastElement | CollectionError | LinkedList\* linkedList  LinkedListNode\*\* linkedListNode | Liest letzten Knoten |

## Codebeispiel:

# 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();

## String - Funktionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Funktionsname | Rückgabewert | Parameter | Nutzen |
| StringInitialize | void | String\* string, char\* inputString | Inizialisierung des Containers |
| StringDestruction | void | String\* string | Gibt Speicher frei |
| StringCharInsertAt | CollectionError | String\* string, unsigned int indexValue, char value | Fügt/ ersetzt einen Char beim Index |
| StringCharAdd | CollectionError | String\* string, char addChar | Fügt einen Char an die nächste freie Stelle hinzu |
| StringCharGet | CollectionError | String\* string, unsigned int index, char\* out | Gibt ein Char aus dem String zurück |
| StringConcat | CollectionError | String\* string, char\* addString | Vereint ein String Objekt mit einem char\* |
| StringGetFullString | char\* | String\* list | Gibt den String als char\* zurück. Muss wieder an den Ram freigeben! |

## Codebeispiel:

