Belegarbeit Programmieren1

Erzeugt von Doxygen 1.8.4

Sam Apr 5 2014 04:10:02

Inhaltsverzeichnis

Kapitel 1

Hauptseite

Autor: Name: N. Schwirz

Version: 0.1.1

Aufgabenstellung

Es werden 4 Belegaufgaben (Aufgabe 1-4) ausgegeben. Die von Ihnen zu lösende Aufgabe ergibt sich aus Matrikelnummer modulo 4 plus 1.

Hinweise zur Lösung:

Alle Aufgaben enthalten die Programmierung eines C-Programmoduls, der eine *doppelt verkettete Liste* bereitstellt. Die Funktionalität dieses Listmoduls soll durch das Headerfile list.h beschrieben sein. Das in list.h vorgegebene Interface ist verbindlich, lediglich die Datentypen tCnct und tList sind eigenständig zu erarbeiten.

Die Daten sind in einer Datei zu speichern und programmintern durch die doppelt verkettete Liste zu verwalten. Haben sich die Daten beim Programmlauf verändert, so sind sie beim Verlassen des Programms zu speichern. Das Programm erlaube in jedem Fall das Erfassen, Löschen, Suchen und Anzeigen der Datensätze, sowie eine sortierte tabellarische Auflistung der Daten.

Die Quelltexte sind sorgsam zu kommentieren. Dazu gehört auch die Urheberschaft im Kopf der Quelltexte.

Für alle verwalteten Daten sind Speicherbereiche in der erfordelichen Größe per *malloc* / *free* bereitzustellen

Das Programm bestehe mindestens aus 3 C-Modulen bestehend aus c- und Headerfile. Dabei realisiere ein Modul die Liste, ein Modul die Benutzerschnittstelle (Menus, Eingabemasken, Anzeige ...) und das dritte Modul die oberflächenunabhängigen Teile, z.B. Dateiarbeit. Die Module sollen einzeln compilierbar sein.

Sofern das Programm keine plattformabhängigen Oberflächenbibliotheken, wie *libforms*, *gtk* oder *ncurses* verwendet, soll das Programm portabel sein und sowohl unter *Windows* als auch unter *Linux/Unix* übersetzt und ausgeführt werden können. In jedem Fall müssen die Programme auf den Rechnern in den Laboren der Fakultät vorgeführt und übersetzt werden können.

Belegaufgabe 2

Programmieren Sie eine Materialverwaltung.

Das Programm soll Datensätze, die Artikelbezeichnung, Artikelnummer und Lagerbestand speichern, verwalten. Der Lagerbestand soll dabei über einen gesonderten Menuepunkt "Zugang/Abgang" gesondert veränderbar sein.

Lösungsansatz und Umsetzung

Die gestellte Aufgabe, eine Materialverwaltung mit *Doppelt verketteter Liste* in C zu programmieren, löste ich mit der vorliegenden Konsolenanwendung. Ich programmierte unter GNU/ Linux und legte großen Wert auf Portabilität. Der vorliegende Quellcode meines Programmes sollte sich somit (zumindest mit Gcc) auch für andere Betriebssysteme kompilieren und ausführen lassen.

Kompilier- und Installationsanleitung

1. Kompilieren: `make {linux|windows}` (einen Überblick über alle Kompilierziele gibt: `make help` und `make all` baut sie alle!) Alternativ kann auch per `gcc *.c -o Materialverwaltung -D{LINUX|WINDOWS}` (vom Programmverzeichnis aus) selbst kompiliert werden.
2. Dokumentation (neu)generieren: `make doku` (Alternativ kann auch die Datei: Readme.txt.md einen Überblick geben.)
3. Ausführen: `Materialverwaltung_xx` (xx steht hier für das Kürzel ihres Betriebssystems etc.)

Unter dem Betriebssystem *Windows* wird zum Kompilieren eine `Cygwin-Umgebung` (siehe auch den `Wikipediaartikel zu Cygwin`) empfohlen, die zuvor installiert werden sollte.

Quellenangabe

- Der Quellcode des Programms: ist von mir selbst geschrieben, ggf. in Anlehung an Studienmitschriften und Praktikumsaufgaben.
- Den Bildschirmschnappschuss des Programmes und die schamatische Darstellung der Datenstruktur erstellte ich selbst
- Das für die Dokumentation verwendete Logo wurde von mir von Wikipedia-Commons (`http://commons.wikimedia.org/wiki/File:Bochs.png`) herutergeladen. Es stammt von der OpenSource-Software *Bochs* (`http://bochs.sourceforge.net/`) und steht unter der GNU General Public License, einer Lizenz, die die Weiterverwendung ausdrücklich erlaubt.

Kapitel 2

Fehlercodeübersicht

Global cmpArtikel (void *pItList, void *pItNew)

3: Ungültiger Sortierschlüssel

Global DEBUG_INIT

100: Logdatei konnte nicht erstellt/ zum Schreiben geöffnet werden.

Global DEBUG_STR (s)

101: Logdatei konnte nicht zum Anhängen geöffnet werden.

Global loadFromFile (char *filename, tList *pList)

1: Die Artikeldatei kann nicht zum Einlesen geöffnet werden.

Global saveToFile (char *filename, tList *pList)

2: Die Artikeldatei kann nicht zum Schreiben geöffnet werden.

Kapitel 3

Datenstruktur-Verzeichnis

3.1 Datenstrukturen

Hier folgt die Aufzählung aller Datenstrukturen mit einer Kurzbeschreibung:

Kapitel 4

Datei-Verzeichnis

4.1 Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

Kapitel 5

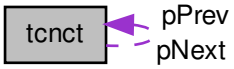
Datenstruktur-Dokumentation

5.1 tcnct Strukturreferenz

Typdeklaration des Connectors (=Listen(verkettungs)element)

`#include <List.h>`

Zusammengehörigkeiten von tcnct:



Datenfelder

- struct tcnct * pPrev

  Zeigt auf das vorherige Element, NULL falls 1. Element.
- struct tcnct * pNext

  Zeigt auf das nachfolgende Element, NULL falls letztes Element.
- void * pData

  Zeigt auf die eigentlichen Daten, dem Datensatz des Listenelements.

5.1.1 Ausführliche Beschreibung

Typdeklaration des Connectors (=Listen(verkettungs)element)

Er stellt den Datentyp der eigentlichen Listenelemente dar, die untereinander verbunden (=connected) - oder besser: verkettet - sind.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- List.h

5.2 tdata Strukturreferenz

Typdeklaration der Datensatzstruktur. Sie enthält die eigentlichen Nutzdaten.

`#include <Materialverwaltung.h>`

Datenfelder

- double Lagerbestand

  Die Artikelnummer (Ganzzahlig)

5.2.1 Ausführliche Beschreibung

Typdeklaration der Datensatzstruktur. Sie enthält die eigentlichen Nutzdaten.

Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

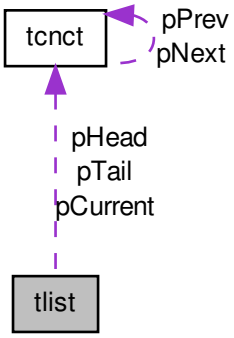
- Materialverwaltung.h

5.3 tlist Strukturreferenz

Typdeklaration der "Doppelt verketteten Liste".

`#include <List.h>`

Zusammengehörigkeiten von tlist:



Datenfelder

- tCnct ∗ pHead

  Zeigt auf das erste Listenelement.
- tCnct ∗ pTail

  Zeigt auf das letzte Listenelement.
- tCnct ∗ pCurrent

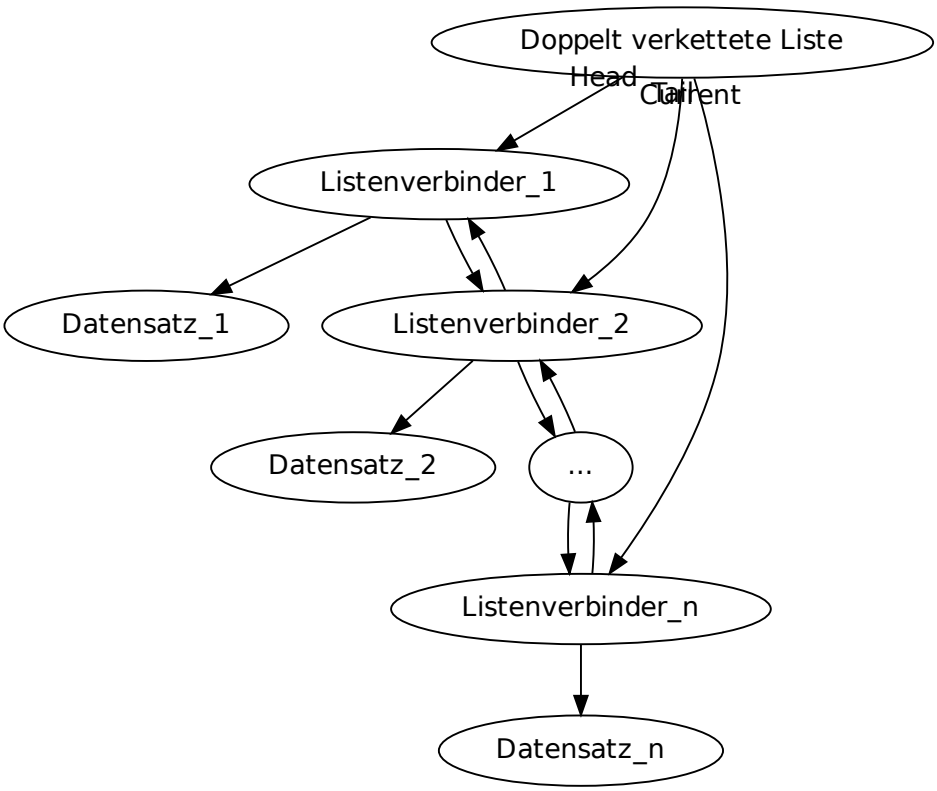
  Zeigt auf das aktuelle Listenelement.

5.3.1 Ausführliche Beschreibung

Typdeklaration der "Doppelt verketteten Liste".

Eine Doppelt verkettete Liste muss zur Laufzeit dynamisch per `createList()` erzeugt und nach Gebrauch per `deleteList()` wieder gelöscht werden. Sie besteht aus Listenelementen vom Typ `tCnct`, die miteinander verkettet werden.

Schematische Darstellung:



Die Dokumentation für diese Struktur wurde erzeugt aufgrund der Datei:

- List.h

Kapitel 6

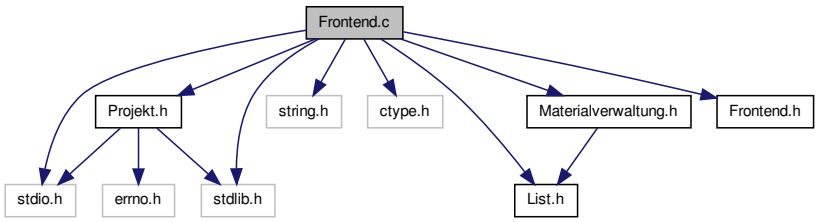
Datei-Dokumentation

6.1 Frontend.c-Dateireferenz

Diese Datei enthält das Frontend der Materialverwaltung.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "Projekt.h"
#include "Materialverwaltung.h"
#include "List.h"
#include "Frontend.h"
```

Include-Abhängigkeitsdiagramm für Frontend.c:



Funktionen

- void showHeader (char text[])

  Gibt den Programmkopf (anpassbare Überschrift) auf einem leeren Bildschirm aus.
- int listDs (tList ∗pList)

  Listet alle in der übergebenen Liste enthaltenen Datensätze auf.
- int cmpArtikel (void ∗pItList, void ∗pItNew)

  Vergleichsfunktion zum sortierten Einfügen neuer Artikel per AddItemToList(). (Weiterleitungsfunktion)
- int cmpArtikelNr (void ∗pItList, void ∗pItNew)

  Vergleichsfunktion zum (nach ArtikelNr. sortierten) Einfügen neuer Artikel per AddItemToList().
- int cmpArtikelBez (void ∗pItList, void ∗pItNew)

  Vergleichsfunktion zum (nach ArtikelBezeichnung. sortierten) Einfügen neuer Artikel per `AddItemToList()`.
- int searchItem (tList ∗pList)

Per Suchmaske kann die Auflistung aller Artikel-Datensätze entsprechend gefiltert werden.

- int inputNewDs (tList ∗pList)

  Manuelle Eingabe eines Artikel-Datensatzes mit automatischer Einsortierung in die Liste.
- int deleteDs (tList ∗pList)

  Löschen eines Artikel-Datensatz per Suchmaskenauswahl oder direkt.
- int materialManagement (tList ∗pList)

  Verwaltung von Zugängen/ Abgängen per Suchmaskenauswahl oder direkt.
- void showHelp ()

  Zeigt die Programmhilfe an.
- int main (int argc, char ∗argv[])

  Die Main-Funktion des Projektes. Sie implementiert das Frontend und die Programmlogik welche sie mit den Daten des Listenmoduls verknüpft.

Variablen

- int sortKey

  Der Sortierschlüssel (.

6.1.1 Ausführliche Beschreibung

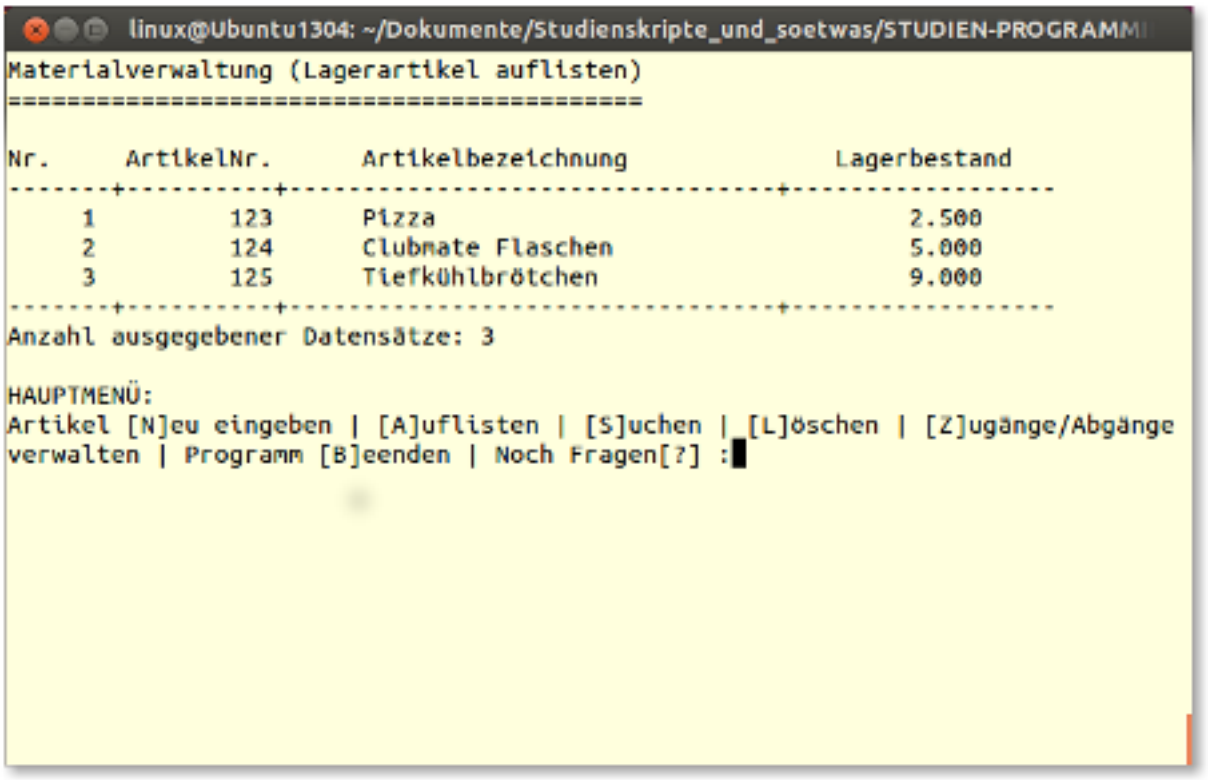
Diese Datei enthält das Frontend der Materialverwaltung. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1



"Die Materialverwaltung in Aktion"

6.1.2 Dokumentation der Funktionen

6.1.2.1 int cmpArtikel (void * *pltList,* void * *pltNew*)

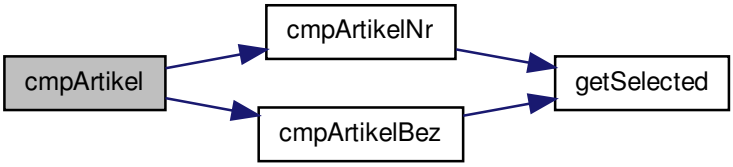
Vergleichsfunktion zum sortierten Einfügen neuer Artikel per AddItemToList(). (Weiterleitungsfunktion)

Diese Funktion ist eine Weiterleitung. Sie leitet Aufrufe entsprechend sortKey an eine spezifischere Vergleichsfunktion weiter. 1= cmpArtikelNr(), 2= cmpArtikelBez()

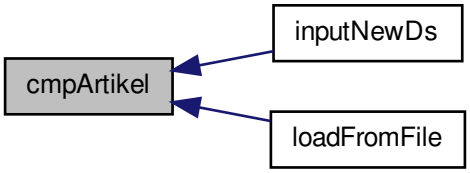
Fehlercodes 3: Ungültiger Sortierschlüssel

Wird benutzt von inputNewDs() und loadFromFile().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



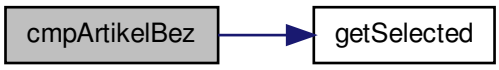
6.1.2.2 int cmpArtikelBez (void * *pltList,* void * *pltNew*)

Vergleichsfunktion zum (nach ArtikelBezeichnung. sortierten) Einfügen neuer Artikel per `AddItemToList()`.

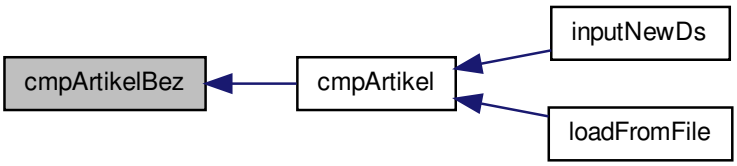
Ähnlich anderer Vergleichsfunktionen, wird -1 (1. Wert ist kleiner als 2.), 0 (beide Werte sind gleich) oder +1 (1. Wert ist größer als 2.) zurückgegeben.

Wird benutzt von cmpArtikel().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.1.2.3 int cmpArtikelNr (void * *pltList*, void * *pltNew*)

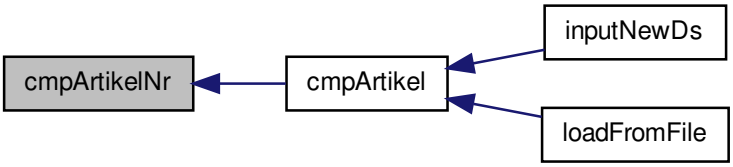
Vergleichsfunktion zum (nach ArtikelNr. sortierten) Einfügen neuer Artikel per AddItemToList().

Ähnlich anderer Vergleichsfunktionen, wird -1 (1. Wert ist kleiner als 2.), 0 (beide Werte sind gleich) oder +1 (1. Wert ist größer als 2.) zurückgegeben.

Wird benutzt von cmpArtikel().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



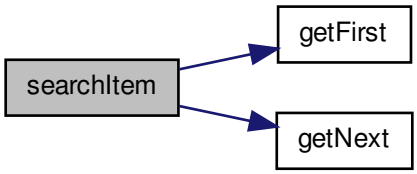
6.1.2.4 int searchItem (tList * *pList*)

Per Suchmaske kann die Auflistung aller Artikel-Datensätze entsprechend gefiltert werden.

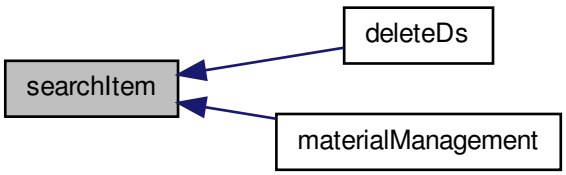
Gibt die Anzahl der zutreffenen Datensätze zurück.

Wird benutzt von deleteDs() und materialManagement().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.1.3 Variablen-Dokumentation

6.1.3.1 int sortKey

Der Sortierschlüssel (.

Siehe auch

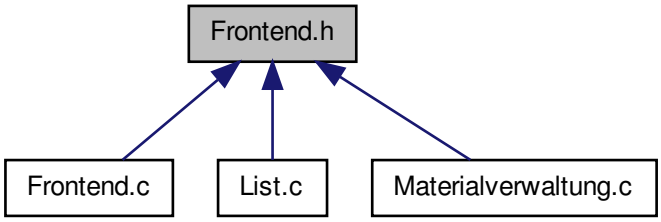
cmpArtikel())

Wird benutzt von cmpArtikel().

6.2 Frontend.h-Dateireferenz

Diese Datei enthält das Frontend-Headerfile der Materialverwaltung.

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- void showHeader (char text[])

  Gibt den Programmkopf (anpassbare Überschrift) auf einem leeren Bildschirm aus.
- int listDs (tList ∗pList)

  Listet alle in der übergebenen Liste enthaltenen Datensätze auf.
- int cmpArtikel (void ∗pItList, void ∗pItNew)

  Vergleichsfunktion zum sortierten Einfügen neuer Artikel per AddItemToList(). (Weiterleitungsfunktion)
- int cmpArtikelNr (void ∗pItList, void ∗pItNew)

  Vergleichsfunktion zum (nach ArtikelNr. sortierten) Einfügen neuer Artikel per AddItemToList().
- int cmpArtikelBez (void ∗pItList, void ∗pItNew)

  Vergleichsfunktion zum (nach ArtikelBezeichnung. sortierten) Einfügen neuer Artikel per `AddItemToList()`.
- int searchItem (tList ∗pList)

  Per Suchmaske kann die Auflistung aller Artikel-Datensätze entsprechend gefiltert werden.
- int inputNewDs (tList ∗pList)

  Manuelle Eingabe eines Artikel-Datensatzes mit automatischer Einsortierung in die Liste.
- int deleteDs (tList ∗pList)

  Löschen eines Artikel-Datensatz per Suchmaskenauswahl oder direkt.
- int materialManagement (tList ∗pList)

  Verwaltung von Zugängen/ Abgängen per Suchmaskenauswahl oder direkt.
- void showHelp ()

  Zeigt die Programmhilfe an.

- int main (int argc, char ∗argv[])

  Die Main-Funktion des Projektes. Sie implementiert das Frontend und die Programmlogik welche sie mit den Daten des Listenmoduls verknüpft.

6.2.1 Ausführliche Beschreibung

Diese Datei enthält das Frontend-Headerfile der Materialverwaltung. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1

6.2.2 Dokumentation der Funktionen

6.2.2.1 int cmpArtikel (void ∗ *pltList,* void ∗ *pltNew*)

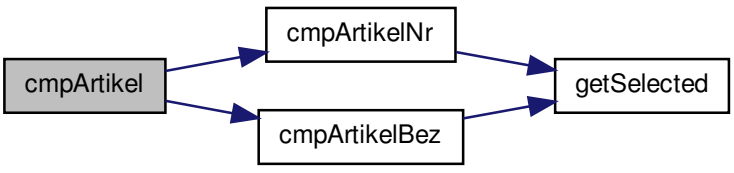
Vergleichsfunktion zum sortierten Einfügen neuer Artikel per AddItemToList(). (Weiterleitungsfunktion)

Diese Funktion ist eine Weiterleitung. Sie leitet Aufrufe entsprechend sortKey an eine spezifischere Vergleichsfunktion weiter. 1= cmpArtikelNr(), 2= cmpArtikelBez()

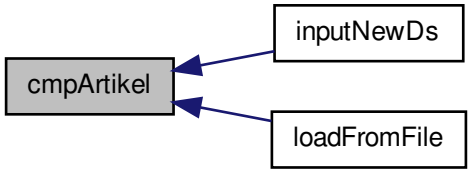
Fehlercodes 3: Ungültiger Sortierschlüssel

Wird benutzt von inputNewDs() und loadFromFile().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.2.2.2 int cmpArtikelBez (void * *pltList*, void * *pltNew*)

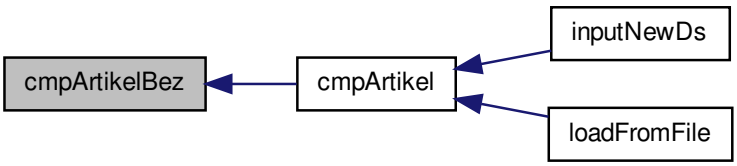
Vergleichsfunktion zum (nach ArtikelBezeichnung. sortierten) Einfügen neuer Artikel per `AddItemToList()`.

Ähnlich anderer Vergleichsfunktionen, wird -1 (1. Wert ist kleiner als 2.), 0 (beide Werte sind gleich) oder +1 (1. Wert ist größer als 2.) zurückgegeben.

Wird benutzt von cmpArtikel().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.2.2.3 int cmpArtikelNr (void * *pltList*, void * *pltNew*)

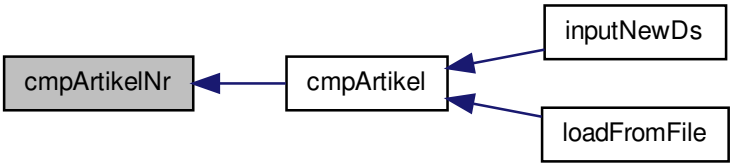
Vergleichsfunktion zum (nach ArtikelNr. sortierten) Einfügen neuer Artikel per AddItemToList().

Ähnlich anderer Vergleichsfunktionen, wird -1 (1. Wert ist kleiner als 2.), 0 (beide Werte sind gleich) oder +1 (1. Wert ist größer als 2.) zurückgegeben.

Wird benutzt von cmpArtikel().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



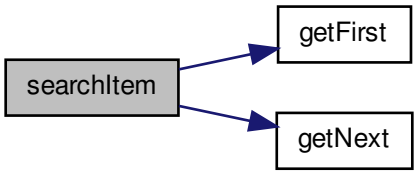
6.2.2.4 int searchItem (tList * *pList*)

Per Suchmaske kann die Auflistung aller Artikel-Datensätze entsprechend gefiltert werden.

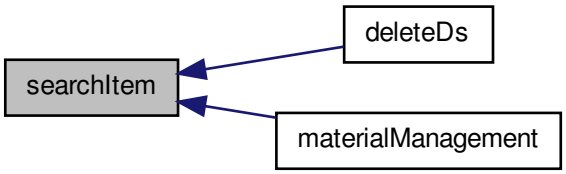
Gibt die Anzahl der zutreffenen Datensätze zurück.

Wird benutzt von deleteDs() und materialManagement().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

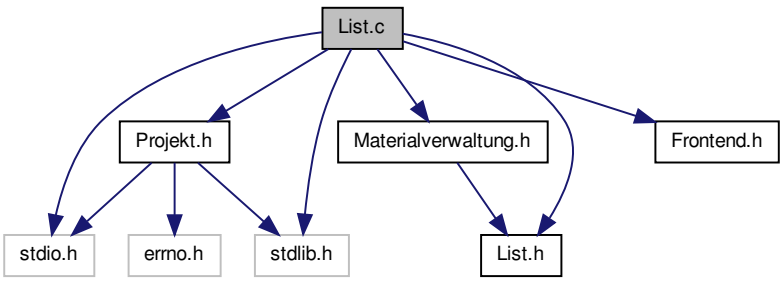


6.3 List.c-Dateireferenz

Diese Datei enthält das Listenmodul der Materialverwaltung.

```
#include <stdio.h>
#include <stdlib.h>
#include "Projekt.h"
#include "Materialverwaltung.h"
#include "List.h"
#include "Frontend.h"
```

Include-Abhängigkeitsdiagramm für List.c:



Funktionen

- tList ∗ createList (void)

  Erzeugt eine leere Liste.
- int deleteList (tList ∗pList)

  Löscht die übergebene Liste samt Inhalt.
- int insertBehind (tList ∗pList, void ∗pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) hinter dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.
- int insertBefore (tList ∗pList, void ∗pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) vor dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.

- int insertHead (tList ∗pList, void ∗pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) am Listenanfang ein und macht dieses Listenelement zum Aktuellen.
- int insertTail (tList ∗pList, void ∗pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) am Listenende ein und macht dieses Listenelement zum Aktuellen.
- int addItemToList (tList ∗pList, void ∗pItem, int(∗fcmp)(void ∗pItList, void ∗pItNew))

  Fügt ein neues Listenelement (samt übergebenen Datensatz) an passender Stelle ein und macht dieses Listenelement zum Aktuellen.
- void removeItem (tList ∗pList)

  Löscht das aktuelle Listenelement samt anhängendem Datenelement.
- void ∗ getSelected (tList ∗pList)

  Gibt (einen Pointer auf) den Datensatz des aktuellen Listenelements zurück.
- void ∗ getFirst (tList ∗pList)

  gibt (einen Pointer auf) den Datensatz des ersten Listenelements zurück und macht dieses Listenelement zum Aktuellen.
- void ∗ getLast (tList ∗pList)

  gibt (einen Pointer auf) den Datensatz des letzten Listenelements zurück und macht dieses Listenelement zum Aktuellen.
- void ∗ getNext (tList ∗pList)

  gibt (einen Pointer auf) den Datensatz des nächsten Listenelements zurück und macht dieses Listenelement zum Aktuellen.
- void ∗ getPrev (tList ∗pList)

  gibt (einen Pointer auf) den Datensatz des vorhergehenden Listenelements zurück und macht dieses Listenelement zum Aktuellen.
- void ∗ getIndexed (tList ∗pList, int Idx)

  gibt (einen Pointer auf) den Datensatz eines Listenelements mit einem bestimmten Index (Listenpositionsnr.) zurück und macht dieses Listenelement zum Aktuellen.

6.3.1 Ausführliche Beschreibung

Diese Datei enthält das Listenmodul der Materialverwaltung. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1

Es wird versucht die Liste nach außen hin zu kapseln, so das sich von außen her nur um die Nutzdaten gekümmert werden braucht. Auf die Liste sollte deshalb von außerhalb nur mit den hier bereitgestellten Funktionen und möglichst nicht direkt zugegriffen werden.

6.3.2 Dokumentation der Funktionen

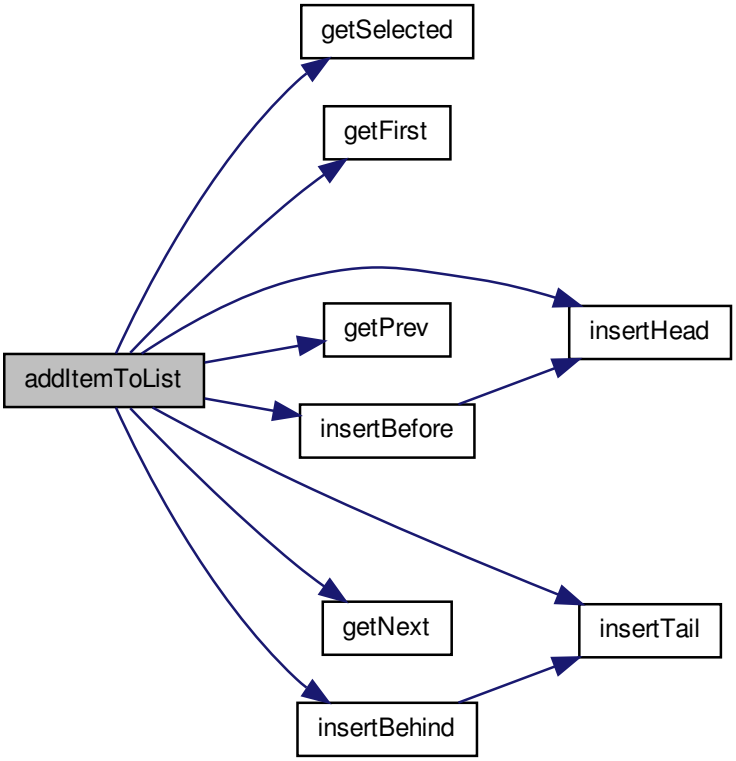
6.3.2.1 int addItemToList (tList ∗ *pList,* void ∗ *pItem,* int(∗)(void ∗pItList, void ∗pItNew) *fcmp*)

Fügt ein neues Listenelement (samt übergebenen Datensatz) an passender Stelle ein und macht dieses Listenelement zum Aktuellen.

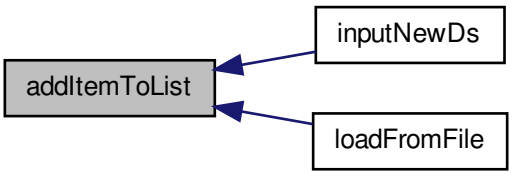
An welcher Stelle das übergebene Element eingefügt wird, wird von der im 3. Param. übergebenen (Vergleichs-)Funktion gesteuert. Durch die Verwendung unterschiedlicher Vergleichsfunktionen sind somit verschiedene Sortierkriterien etwa eine Sortierungen nach unterschiedlichen Feldern möglich

Wird benutzt von inputNewDs() und loadFromFile().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.3.2.2 tList* createList (void)

Erzeugt eine leere Liste.

Es wird ein Pointer auf die erzeugte Liste zurückgegeben. Im Fehlerfall ist dies ein Nullpointer.

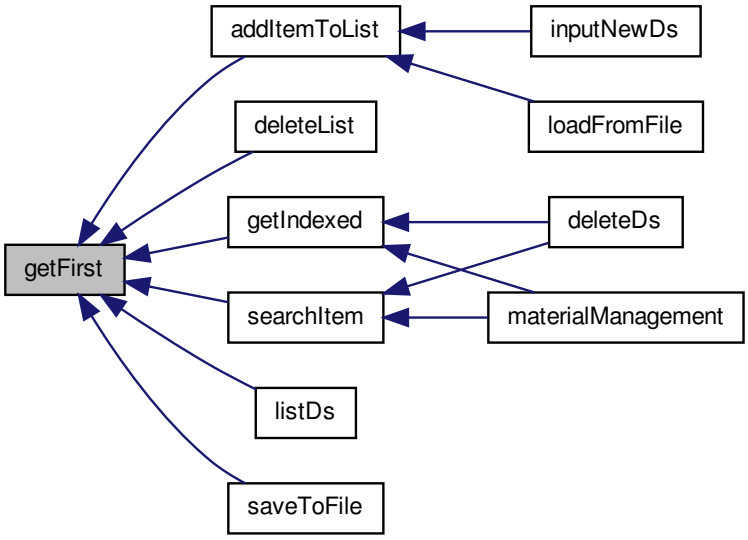
6.3.2.3 void* getFirst (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des ersten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Achtung! Gibt NULL-Pointer zurrück, falls die Liste leer ist.

Wird benutzt von addItemToList(), deleteList(), getIndexed(), listDs(), saveToFile() und searchItem().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



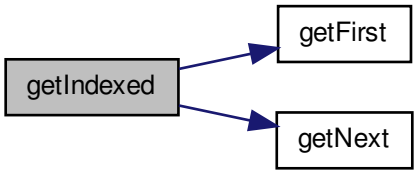
6.3.2.4 void* getIndexed (tList * *pList,* int *Idx*)

gibt (einen Pointer auf) den Datensatz eines Listenelements mit einem bestimmten Index (Listenpositionsnr.) zurück und macht dieses Listenelement zum Aktuellen.

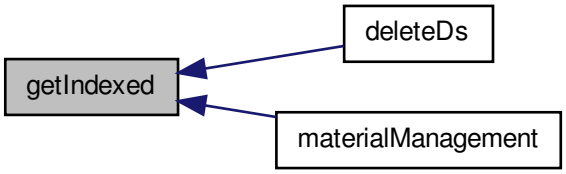
Sollte kein Listenelement mit dem gewünschten Index (der Nr. des Listenelements) existieren, wird ein Nullpointer (!) zurückgegeben.

Wird benutzt von deleteDs() und materialManagement().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.3.2.5 void* getLast (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des letzten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Achtung! Gibt NULL-Pointer zurrück, falls die Liste leer ist.

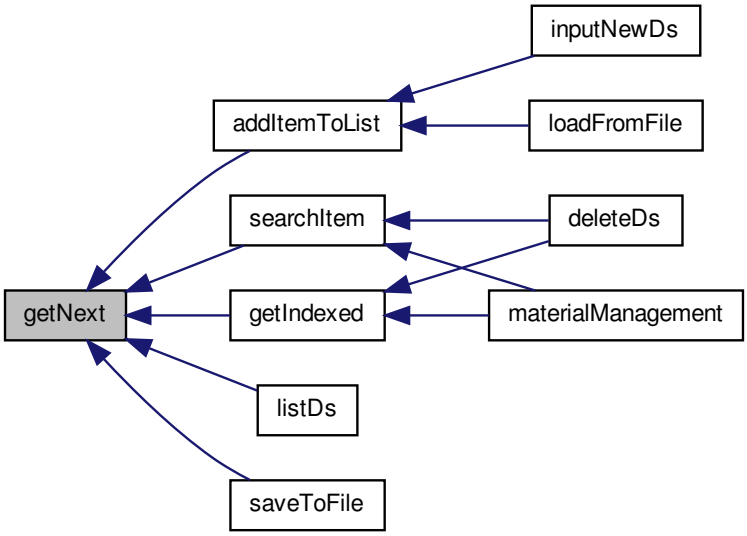
6.3.2.6 void* getNext (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des nächsten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Sollte das aktuelle Listenelement schon das letzte der Liste sein (z.B. in einer leeren Liste), wird ein Nullpointer (!) zurückgegeben.

Wird benutzt von addItemToList(), getIndexed(), listDs(), saveToFile() und searchItem().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



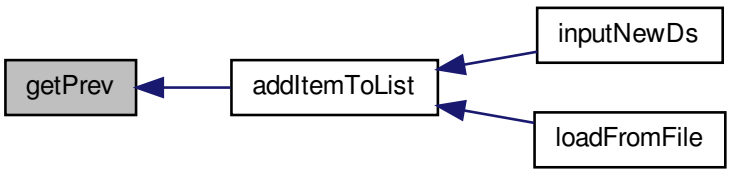
6.3.2.7 void* getPrev (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des vorhergehenden Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Sollte das aktuelle Listenelement schon das erste der Liste sein, wird ein Nullpointer (!) zurückgegeben.

Wird benutzt von addItemToList().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



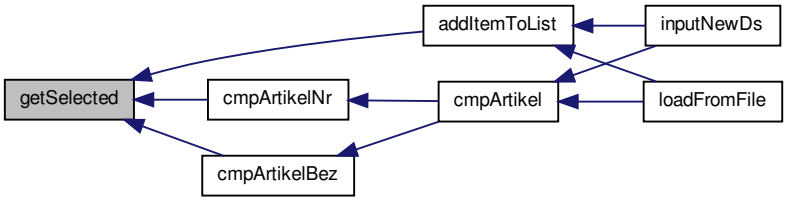
6.3.2.8 void* getSelected (tList * *pList*)

Gibt (einen Pointer auf) den Datensatz des aktuellen Listenelements zurück.

Achtung! Gibt NULL-Pointer zurrück, falls die Liste leer ist.

Wird benutzt von addItemToList(), cmpArtikelBez() und cmpArtikelNr().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.3.2.9 int insertBefore (tList * *pList,* void * *pItemIns*)

Fügt ein neues Listenelement (samt übergebenen Datensatz) vor dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.

Im Zweifelsfall wird das neue Listenelement einfach am Listenanfang eingefügt

Wird benutzt von addItemToList().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

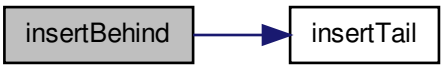
6.3.2.10 int insertBehind (tList * *pList,* void * *pItemIns*)

Fügt ein neues Listenelement (samt übergebenen Datensatz) hinter dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.

Im Zweifelsfall wird das neue Listenelement einfach am Listenende angehangen

Wird benutzt von addItemToList().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

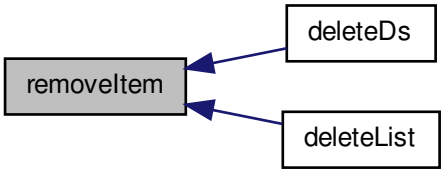
6.3.2.11 void removeItem (tList * *pList*)

Löscht das aktuelle Listenelement samt anhängendem Datenelement.

Achtung! Weist ggf. NULL-Pointer zu, falls nötig.

Wird benutzt von deleteDs() und deleteList().

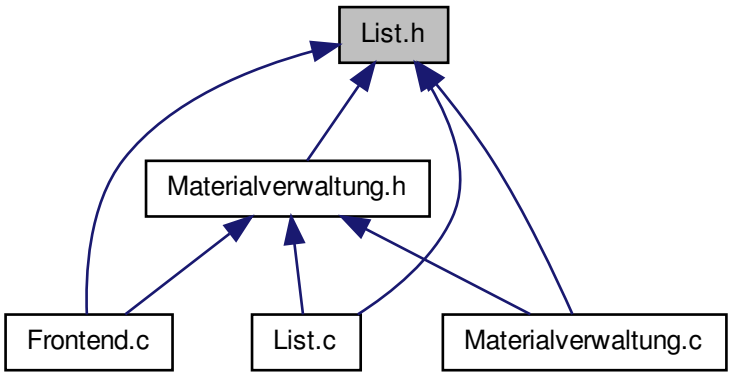
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.4 List.h-Dateireferenz

Diese Datei enthält das vorgegebene Headerfile zum Listenmodul der Materialverwaltung.

Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Datenstrukturen

- struct tcnct

  Typdeklaration des Connectors (=Listen(verkettungs)element)
- struct tlist

  Typdeklaration der "Doppelt verketteten Liste".

Typdefinitionen

- typedef struct tcnct tCnct

  Typdeklaration des Connectors (=Listen(verkettungs)element)
- typedef struct tlist tList

  Typdeklaration der "Doppelt verketteten Liste".

Funktionen

- tList ∗ createList (void)

  Erzeugt eine leere Liste.
- int deleteList (tList ∗pList)

  Löscht die übergebene Liste samt Inhalt.
- int insertBehind (tList ∗pList, void ∗pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) hinter dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.
- int insertBefore (tList ∗pList, void ∗pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) vor dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.
- int insertHead (tList ∗pList, void ∗pItemIns)

Fügt ein neues Listenelement (samt übergebenen Datensatz) am Listenanfang ein und macht dieses Listenelement zum Aktuellen.

- int insertTail (tList *pList, void *pItemIns)

  Fügt ein neues Listenelement (samt übergebenen Datensatz) am Listenende ein und macht dieses Listenelement zum Aktuellen.

- int addItemToList (tList *pList, void *pItem, int(*fcmp)(void *pItList, void *pItNew))

  Fügt ein neues Listenelement (samt übergebenen Datensatz) an passender Stelle ein und macht dieses Listenelement zum Aktuellen.

- void removeItem (tList *pList)

  Löscht das aktuelle Listenelement samt anhängendem Datenelement.

- void * getSelected (tList *pList)

  Gibt (einen Pointer auf) den Datensatz des aktuellen Listenelements zurück.

- void * getFirst (tList *pList)

  gibt (einen Pointer auf) den Datensatz des ersten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

- void * getLast (tList *pList)

  gibt (einen Pointer auf) den Datensatz des letzten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

- void * getNext (tList *pList)

  gibt (einen Pointer auf) den Datensatz des nächsten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

- void * getPrev (tList *pList)

  gibt (einen Pointer auf) den Datensatz des vorhergehenden Listenelements zurück und macht dieses Listenelement zum Aktuellen.

- void * getIndexed (tList *pList, int Idx)

  gibt (einen Pointer auf) den Datensatz eines Listenelements mit einem bestimmten Index (Listenpositionsnr.) zurück und macht dieses Listenelement zum Aktuellen.

6.4.1 Ausführliche Beschreibung

Diese Datei enthält das vorgegebene Headerfile zum Listenmodul der Materialverwaltung. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1

6.4.2 Dokumentation der benutzerdefinierten Typen

6.4.2.1 typedef struct tcnct tCnct

Typdeklaration des Connectors (=Listen(verkettungs)element)

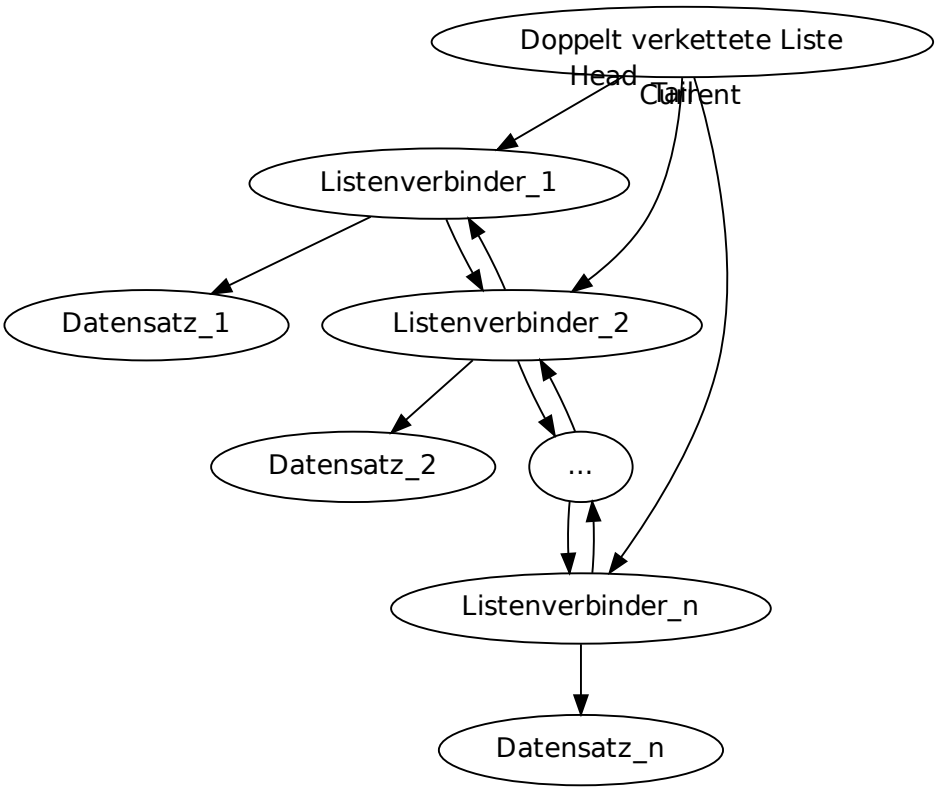
Er stellt den Datentyp der eigentlichen Listenelemente dar, die untereinander verbunden (=connected) - oder besser: verkettet - sind.

6.4.2.2 typedef struct tlist tList

Typdeklaration der "Doppelt verketteten Liste".

Eine Doppelt verkettete Liste muss zur Laufzeit dynamisch per `createList()` erzeugt und nach Gebrauch per `deleteList()` wieder gelöscht werden. Sie besteht aus Listenelementen vom Typ `tCnct`, die miteinander verkettet werden.

Schematische Darstellung:



6.4.3 Dokumentation der Funktionen

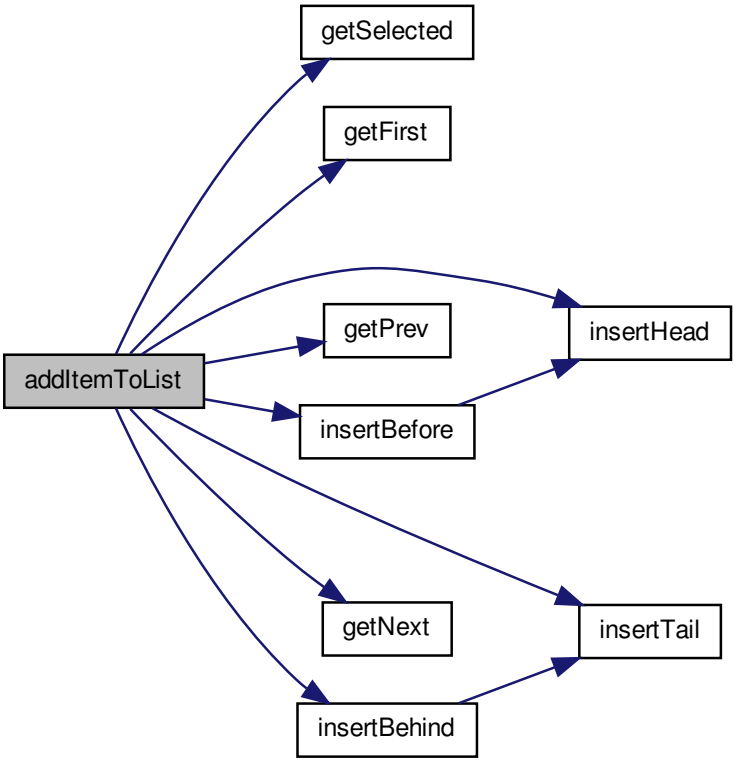
6.4.3.1 int addItemToList (tList * *pList,* void * *pItem,* int(*)(void *pItList, void *pItNew) *fcmp*)

Fügt ein neues Listenelement (samt übergebenen Datensatz) an passender Stelle ein und macht dieses Listenelement zum Aktuellen.

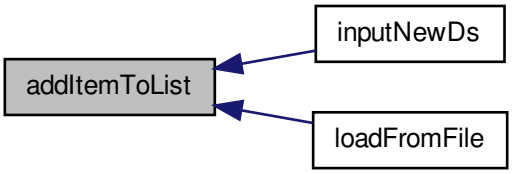
An welcher Stelle das übergebene Element eingefügt wird, wird von der im 3. Param. übergebenen (Vergleichs-)Funktion gesteuert. Durch die Verwendung unterschiedlicher Vergleichsfunktionen sind somit verschiedene Sortierkriterien etwa eine Sortierungen nach unterschiedlichen Feldern möglich

Wird benutzt von inputNewDs() und loadFromFile().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.4.3.2 tList* createList (void)

Erzeugt eine leere Liste.

Es wird ein Pointer auf die erzeugte Liste zurückgegeben. Im Fehlerfall ist dies ein Nullpointer.

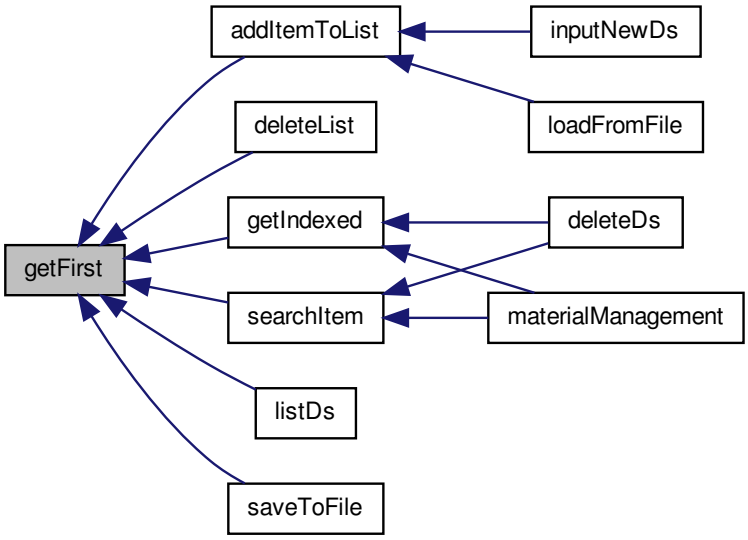
6.4.3.3 void* getFirst (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des ersten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Achtung! Gibt NULL-Pointer zurrück, falls die Liste leer ist.

Wird benutzt von addItemToList(), deleteList(), getIndexed(), listDs(), saveToFile() und searchItem().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



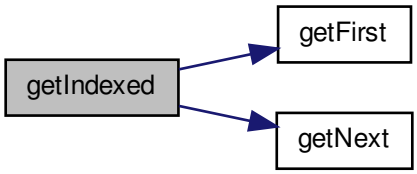
6.4.3.4 void* getIndexed (tList * *pList,* int *Idx*)

gibt (einen Pointer auf) den Datensatz eines Listenelements mit einem bestimmten Index (Listenpositionsnr.) zurück und macht dieses Listenelement zum Aktuellen.

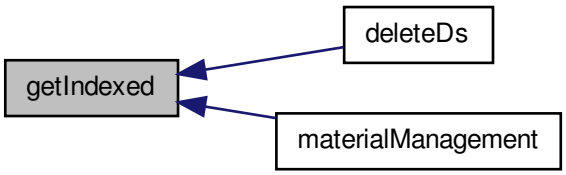
Sollte kein Listenelement mit dem gewünschten Index (der Nr. des Listenelements) existieren, wird ein Nullpointer (!) zurückgegeben.

Wird benutzt von deleteDs() und materialManagement().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.4.3.5 void* getLast (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des letzten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Achtung! Gibt NULL-Pointer zurrück, falls die Liste leer ist.

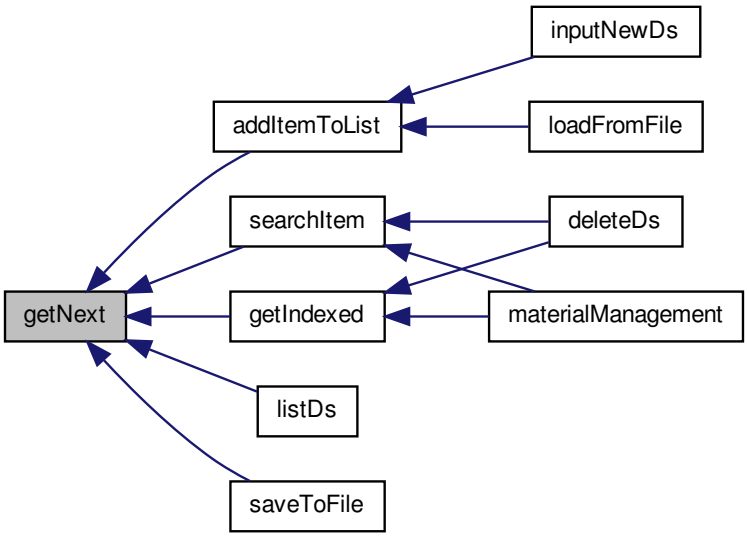
6.4.3.6 void* getNext (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des nächsten Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Sollte das aktuelle Listenelement schon das letzte der Liste sein (z.B. in einer leeren Liste), wird ein Nullpointer (!) zurückgegeben.

Wird benutzt von addItemToList(), getIndexed(), listDs(), saveToFile() und searchItem().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



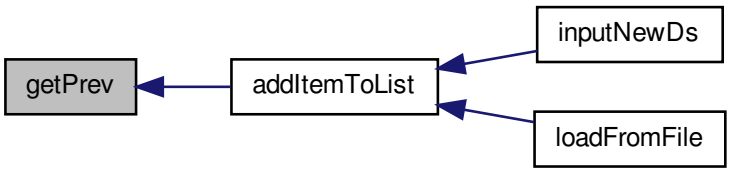
6.4.3.7 void* getPrev (tList * *pList*)

gibt (einen Pointer auf) den Datensatz des vorhergehenden Listenelements zurück und macht dieses Listenelement zum Aktuellen.

Sollte das aktuelle Listenelement schon das erste der Liste sein, wird ein Nullpointer (!) zurückgegeben.

Wird benutzt von addItemToList().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



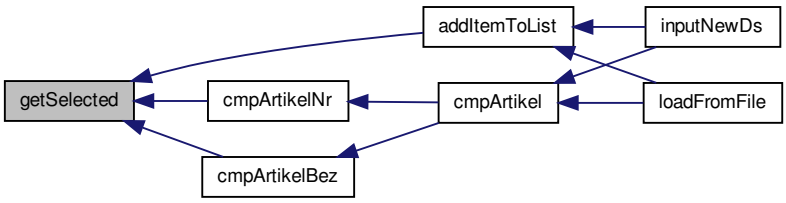
6.4.3.8 void* getSelected (tList * *pList*)

Gibt (einen Pointer auf) den Datensatz des aktuellen Listenelements zurück.

Achtung! Gibt NULL-Pointer zurrück, falls die Liste leer ist.

Wird benutzt von addItemToList(), cmpArtikelBez() und cmpArtikelNr().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



6.4.3.9 int insertBefore (tList * *pList,* void * *pItemIns*)

Fügt ein neues Listenelement (samt übergebenen Datensatz) vor dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.

Im Zweifelsfall wird das neue Listenelement einfach am Listenanfang eingefügt

Wird benutzt von addItemToList().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

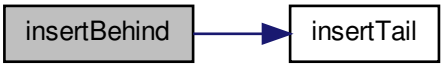
6.4.3.10 int insertBehind (tList * *pList,* void * *pItemIns*)

Fügt ein neues Listenelement (samt übergebenen Datensatz) hinter dem Aktuellem ein und macht dieses Listenelement zum Aktuellen.

Im Zweifelsfall wird das neue Listenelement einfach am Listenende angehangen

Wird benutzt von addItemToList().

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

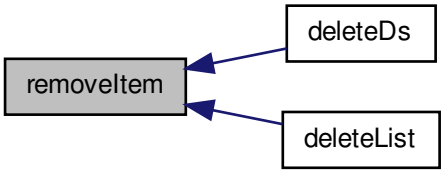
6.4.3.11 void removeItem (tList * *pList*)

Löscht das aktuelle Listenelement samt anhängendem Datenelement.

Achtung! Weist ggf. NULL-Pointer zu, falls nötig.

Wird benutzt von deleteDs() und deleteList().

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

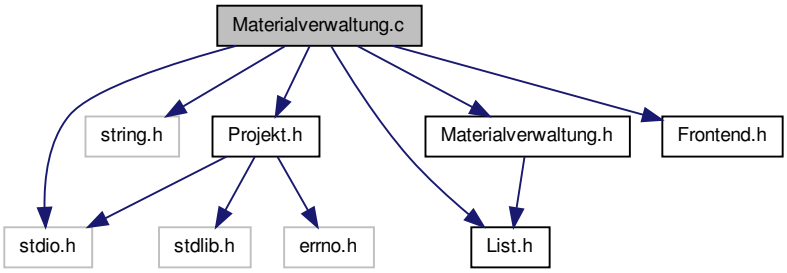


6.5 Materialverwaltung.c-Dateireferenz

Diese Datei enthält allgemeine Funktionen der Materialverwaltung.

```
#include <stdio.h>
#include <string.h>
#include "Projekt.h"
#include "List.h"
#include "Materialverwaltung.h"
#include "Frontend.h"
```

Include-Abhängigkeitsdiagramm für Materialverwaltung.c:



Funktionen

- int loadFromFile (char ∗filename, tList ∗pList)

  Artikel aus übergebener Datei einlesen und in Liste sortiert einfügen.

- int saveToFile (char ∗filename, tList ∗pList)

  Artikel-Datensätze in übergebene Datei speichern.

6.5.1 Ausführliche Beschreibung

Diese Datei enthält allgemeine Funktionen der Materialverwaltung. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1

6.5.2 Dokumentation der Funktionen

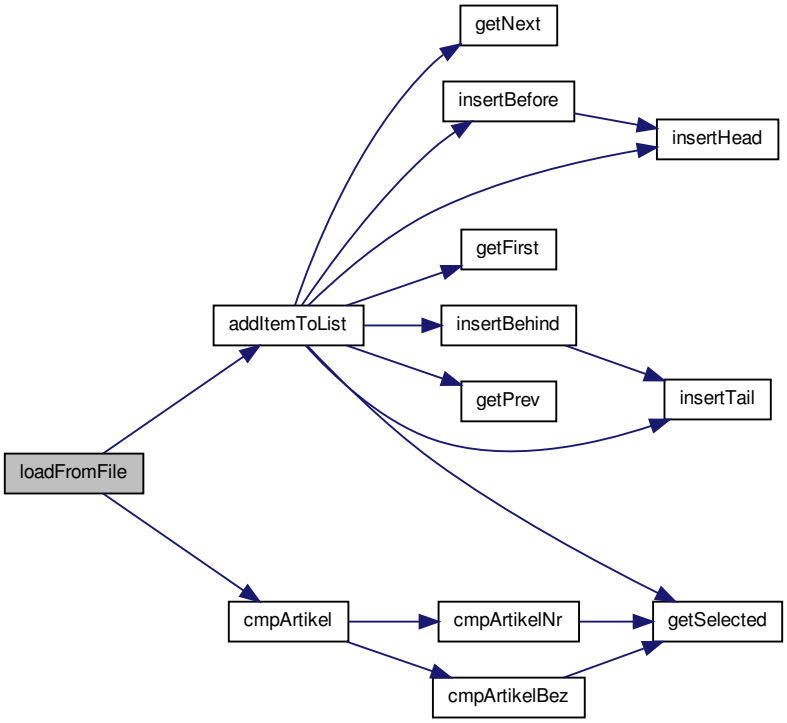
6.5.2.1 int loadFromFile (char ∗ *filename,* tList ∗ *pList*)

Artikel aus übergebener Datei einlesen und in Liste sortiert einfügen.

Gibt #FAIL zurrück, falls ein oder mehr Datensätze nicht richtig gelesen werden konnte, andernfalls #OK.

Fehlercodes 1: Die Artikeldatei kann nicht zum Einlesen geöffnet werden.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



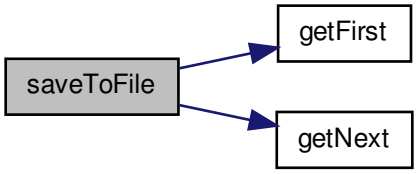
6.5.2.2 int saveToFile (char * *filename,* tList * *pList*)

Artikel-Datensätze in übergebene Datei speichern.

Gibt #FAIL zurrück, falls ein oder mehrere Datensätze nicht richtig gelesen werden konnte, andernfalls #OK.

Fehlercodes 2: Die Artikeldatei kann nicht zum Schreiben geöffnet werden.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

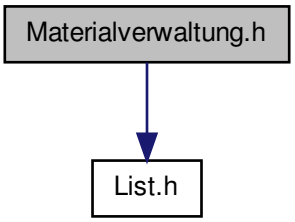


6.6 Materialverwaltung.h-Dateireferenz

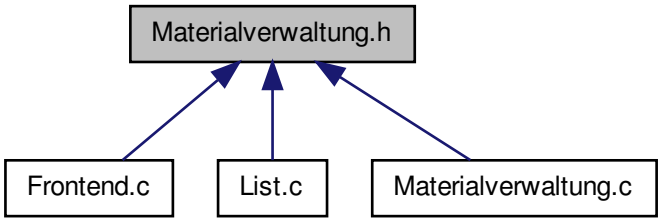
Diese Datei enthält das Headerfile des Materialverwaltungsmoduls.

```
#include "List.h"
```

Include-Abhängigkeitsdiagramm für Materialverwaltung.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Datenstrukturen

- struct tdata

  Typdeklaration der Datensatzstruktur. Sie enthält die eigentlichen Nutzdaten.

Typdefinitionen

- typedef struct tdata tData

  Typdeklaration der Datensatzstruktur. Sie enthält die eigentlichen Nutzdaten.

Funktionen

- int loadFromFile (char ∗filename, tList ∗pList)

  Artikel aus übergebener Datei einlesen und in Liste sortiert einfügen.
- int saveToFile (char ∗filename, tList ∗pList)

  Artikel-Datensätze in übergebene Datei speichern.

6.6.1 Ausführliche Beschreibung

Diese Datei enthält das Headerfile des Materialverwaltungsmoduls. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1

6.6.2 Dokumentation der Funktionen

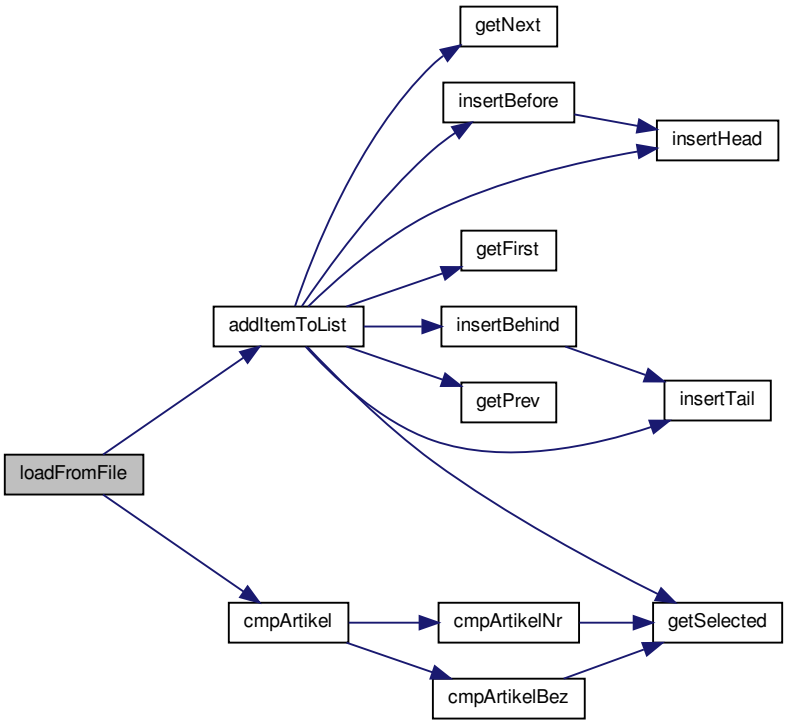
6.6.2.1 int loadFromFile (char * *filename*, tList * *pList*)

Artikel aus übergebener Datei einlesen und in Liste sortiert einfügen.

Gibt #FAIL zurrück, falls ein oder mehr Datensätze nicht richtig gelesen werden konnte, andernfalls #OK.

Fehlercodes 1: Die Artikeldatei kann nicht zum Einlesen geöffnet werden.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



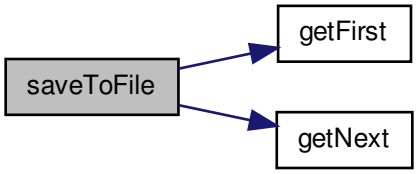
6.6.2.2 int saveToFile (char * *filename*, tList * *pList*)

Artikel-Datensätze in übergebene Datei speichern.

Gibt #FAIL zurrück, falls ein oder mehrere Datensätze nicht richtig gelesen werden konnte, andernfalls #OK.

Fehlercodes 2: Die Artikeldatei kann nicht zum Schreiben geöffnet werden.

Hier ist ein Graph, der zeigt, was diese Funktion aufruft:

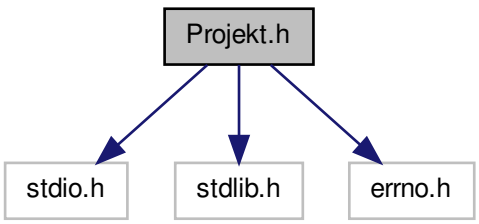


6.7 Projekt.h-Dateireferenz

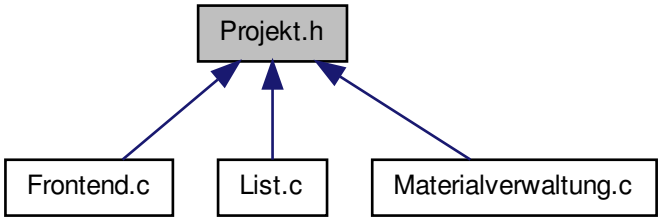
Diese Datei enthält sonstige Funktionen für alle Module der Materialverwaltung.

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
```

Include-Abhängigkeitsdiagramm für Projekt.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Makrodefinitionen

- #define DEBUG

  Das Preprozessorsymbol DEBUG (de)aktiviert die Miteinkompilierung von Makros für Debugmeldungen.
- #define LOGFILE "logfile.txt"

  Protokolldatei für den Debugmodus.
- #define DEBUG_INIT

  Logdatei Neuerstellen oder leeren, falls schon existent.
- #define DEBUG_STR(s)

  Text an Logdatei anhängen.
- #define CLEAR

  Betriebssystemunabhängige Befehle, um den Bildschirm zu löschen.

6.7.1 Ausführliche Beschreibung

Diese Datei enthält sonstige Funktionen für alle Module der Materialverwaltung. Sie ist Bestandteil der Belegarbeit Programmieren1, Aufgabe 2: Materialverwaltung bei Herrn Prof. Beck.

Autor

Name: N. Schwirz

Version

0.1.1

Bemerkungen

Aufruf mit: `gcc -D{LINUX|WINDOWS} *.c`

6.7.2 Makro-Dokumentation

6.7.2.1 #define CLEAR

Wert:

```
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"); \
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
```

Betriebssystemunabhängige Befehle, um den Bildschirm zu löschen.

Je nachdem welches Betriebssystem-Preprozessorsymbol definiert ist, wird ein entsprechender Betriebssystems-pezifischer Befehl verwendet.

Unterstützte Betriebssystemsymbole:

- *LINUX* : `system(clear)`
- *WINDOWS* : `system(cls)`

Vorraussetzung: Erfordert ggf. einen vorherigen Aufruf des Makros DEBUG_INIT, sofern noch nicht geschehen.

Verwendung: `CLEAR`

Falls das Leeren des Bildschirms (per system()-Befehl) fehlschlägt, wird einfach die Fallbackvariante (Leerzeilen per printf()) verwendet. Außerdem wird auch per DEBUG_STR ein entspr. Eintrag in der Protokolldatei (siehe LOGFILE) vermerkt.

Wird benutzt von showHeader().

6.7.2.2 #define DEBUG

Das Preprozessorsymbol DEBUG (de)aktiviert die Miteinkompilierung von Makros für Debugmeldungen.

Zur Verwendung der folgenden Preprozessorinstruktionen muss beim Kompilieren das Symbol DEBUG definiert sein, da sie ansonsten ignoriert werden (sogar in der mit `make doku` erstellten Dokumantation). Dies kann entweder mittels `-DDEBUG` Parameter beim Kompilieren oder am Beginn dieser Datei (bzw. in der Datei, die diese Headerdatei einbindet) geschehen.

6.7.2.3 #define DEBUG_INIT

Wert:

```
FILE* DEBUG_logFile;\
        DEBUG_logFile=fopen(LOGFILE,"w");\
        if (DEBUG_logFile==NULL){\
            perror("Logdatei konnte nicht erstellt/ zum Schreiben geöffnet werden.");\
            exit(100);\
        }\
        fprintf(DEBUG_logFile,"Datum und Zeit der Kompilierung: %s %s ( %s )\n", __DATE__ , __TIME__ ,
      __FILE__);\
        fclose(DEBUG_logFile);
```

Logdatei Neuerstellen oder leeren, falls schon existent.

Verwendung: `DEBUG_INIT`

Dieses Preprozessorsymbol sollte in entspr. Dateien ganz am Anfang eingebunden werden, da die folgenden Makros ggf. darauf aufbauen!

Enthaltene Funktionalität:

- Erstellt die Protokolldatei (siehe: LOGFILE) neu bzw. leert sie, falls sie schon existiert.

Fehlercodes 100: Logdatei konnte nicht erstellt/ zum Schreiben geöffnet werden.

6.7.2.4 #define DEBUG_STR(*s*)

Wert:

```
{ FILE* DEBUG_logfile2;\
        DEBUG_logfile2=fopen("logfile.txt","a");\
            if (DEBUG_logfile2==NULL){\
                    perror("Logdatei konnte nicht zum Anhängen geöffnet werden");\
                    exit(101);\
            }\
        fprintf(DEBUG_logfile2,"%s\n",s);\
        fclose(DEBUG_logfile2);};
```

Text an Logdatei anhängen.

Vorraussetzung: Erfordert einen vorherigen Aufruf des Makros DEBUG_INIT, sofern noch nicht geschehen.

Verwendung: `DEBUG_STR(<Text>)`

Fehlercodes 101: Logdatei konnte nicht zum Anhängen geöffnet werden.

Wird benutzt von cmpArtikel(), loadFromFile() und saveToFile().

Index