**4AHIT**

04.11.2013

**Funktionspointer und Dynamische Datenstrukturen**

**Protokoll**

**Ayvazyan Ari & Belinic Vennesa**

**Inhaltsangabe**

[Git-Link 1](#_Toc371187140)

[Aufgabenstellung 1](#_Toc371187141)

[Designüberlegung 2](#_Toc371187142)

[Arbeitsaufteilung 3](#_Toc371187145)

[Aufwandschätzung 3](#_Toc371187148)

[Endzeitaufteilung 3](#_Toc371187149)

[Arbeitsdurchführung 4](#_Toc371187150)

[Testbericht 5](#_Toc371187151)

[Quellenangaben 8](#_Toc371187152)

**Git-Link**

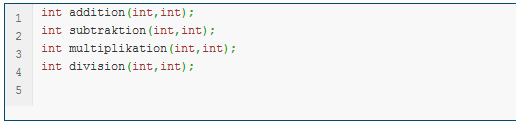
https://github.com/aayvazyan-tgm/FunktionspointerUndDatenstrukturen

# Aufgabenstellung

Diese Aufgabe ist zwei-geteilt und soll auch als Gruppenarbeit (2 Personen) realisiert werden. Vergessen Sie nicht auf die Dokumentation!

Taschenrechner

Implementieren Sie einen einfachen Taschenrechner mittels eines Funktionspointers. Verwenden Sie dabei ein fp-Array. Die folgenden Funktionen müssen angesprochen werden:



Die Operanden sowie der Operator sollen per CLI eingegeben werden können (Programm-Argumente sind nicht vorgesehen). Vergessen Sie nicht auf das Testen!

Addressbuch

Schreiben Sie ein C-Programm, welches ein einfaches Adressbuch implementiert. Verwenden Sie dabei zwei Structs (PERSON, ADDRESS) um die notwendigen Informationen abzulegen. Die Ein- und Ausgabe soll über ein einfaches CLI-Menü erfolgen. Verwenden Sie für die einzelnen Funktionalitäten auch entsprechende Funktionen, welche Sie mittels call-by-reference implementieren.  
  
Verwenden Sie dabei einen dynamischen Speicher um beim Hinzufügen von neuen Personen nicht auf einen statischen Wert beschränkt zu sein!

Bitte beachten Sie, dass der Zuweisungsoperator bei String-Zuweisungen nicht verwendet darf (siehe "C von A bis Z").

# Designüberlegung

Taschenrechner

Die Eingabe des Users erfolgt über das CLI, hier soll bei falscher Eingabe angegeben werden wie das CLI zu benutzen ist.

Es werden neben der Main Methode, welche das CLI auswertet noch vier weitere Funktionen benötigt (Addition, Subtraktion, Multiplikation, Division).

Die CLI wird ausgewertet und an die Funktionen weitergegeben um ihre Ausgabe an den User weiter zu geben.

Adressbuch

Die Überlegung zur Umsetzung des Addressbuches ist folgende:

Structs

Es soll zwei Structs geben, eins für die Person und eins für die Addresse.  
Das für die Person besteht aus ihren Vorname und Nachnamen, und das für die Addresse aus der Straße, der Straßennummer, der Postleitzahl und dem Ort.

Eingaben

Beim Starten des Programms soll der User gefragt werden ob er eine Eintrag zum Addressbuch hinzufügen will (Antwort mit Ja-Nein-Entscheidung). Die Eingabe der Entscheidung wird so lange wiederholt bis der User einen gültigen Wert eingibt.

Wenn das Addressbuch schon mindestens einen Eitrag enthält soll der User gefragt werden ob eine Eintrage hinzufügen oder löschen will, diese Einbage soll auch so lange wieder holt werden bis der User einen gültige Eingabe tätigt.

Das Löschen soll über den Index funktionieren.

# Arbeitsaufteilung

Die Arbeit wird so aufgeteilt, dass jedes Gruppenmitglied, soweit wie möglich, von einander unabhängig arbeiten kann. Deshalb werden den jeweiligen Gruppenmitglieder ganze Programme als Arbeitspakete zugeteilt. Jedes Gruppenmitglied testes die von ihm/ihr geschriebenen Files so weit es nötig ist.

Vennesa

## Addressbuch

Ari

## Taschenrechner

|  |  |  |  |
| --- | --- | --- | --- |
|  | *geschätzte Zeit* | *tatsächliche Zeit* | *Kommentar* |
| **Ayvazyan Ari** | (3)+3 | 7 | Drei Stunden wurden bereits vor der Aufwandschätzung investiert. |
| **Belinic Vennesa** | 4 | 9 |  |

# Aufwandschätzung

|  |  |  |
| --- | --- | --- |
| Programm | | Aufwand in h für das Programm |
| Taschenrechner | | (3)+3 |
| Addressbuch | | 4 |
| SUMME | **10** | |

# Endzeitaufteilung

|  |  |  |
| --- | --- | --- |
| Programm | | Aufwand in h für das Programm |
| Taschenrechner | | 7 |
| Addressbuch | | 9 |
| SUMME | **16** | |

# Arbeitsdurchführung

Taschenrechner

Zuerst wurde ein Prototyp zum „Proof of Concept“ erstellt, welcher ohne CLI arbeitete und nur eine Funktion enthielt, nachdem dies funktionierte wurden die restlichen Funktionen implementiert. Dabei traten keine großen Probleme auf, die Hauptarbeit bestand aus der Recherche wie Pointer in C funktionieren. Anschließend wurde die CLI implementiert, hier musste erst recherchiert werden wie die Parameter übergeben werden und wie char arrays miteinander Verglichen werden können. Die Arbeit konnte erfolgreich nach 5 Stunden investierter Zeit abgeschlossen werden, wovon eine viertel Stunde in Tests investiert wurde. Die Dokumentation nahm insgesamt 2 Stunden in Anspruch.

Addressbuch

Anfänglich entwickelte ich erst einmal nur die Eingaben (keine Funktionen), also ich "simulierte" sozusagen das Hinzufügen und das Löschen indem ich die Anzahl der "aktuellen" Einträge ausgabe. Dass habe ich so gemacht da ich an diesem Tag gefehlt habe als dieses Thema durchgenommen wurde, um noch ein wenig zu recherchieren[1,2].

Danach fing ich an die Methode zu implementieren, als erstes das Hinzufügen. Dabei ergaben sich keine größeren Problem, sondern eher kleinere, doch davon gab es mehrere, sodass es schwer war festzustellen voher der (die) Fehler kam(en).

Danach fing ich an das Löschen zu implementieren, der Code hat noch nicht wirklich funktioniert, aber ich bin davon ausgegangen, dass der Fehler bei der Ausgabe liegt, und habe dies erst einmal außen vor gelassen. Als dann das Löschen auch nicht funktioniert (gelöscht wird über den Index) war mit klar, dass der Fehler beim Speichern der Werte liegt.

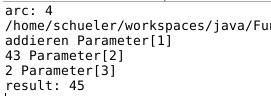
Nach mehrmahligem durchlesen und Recherchen[1,2] bin ich darauf gekommen das ich meine Arrays der jeweiligen Structs bei jedem neuem einlesen auf NULL setzte. Zwischenzeitlich habe ich aber die Löschen-Funktion entfernt um einen besseren Überblick über die Fehler zu haben, als der Fehler beseitigt war fügte ich die Funktion wieder hinzu.

Als alles funktioniert hat, so wie es sollte, lagerte ich die Includes, Defines und Prototypen in ein eigenes Headerfile aus.

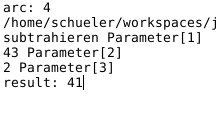
# Testbericht

Taschenrechner

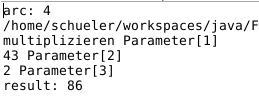
*Addieren:*

**

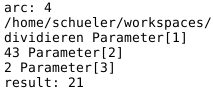
*Subtrahieren:*

**

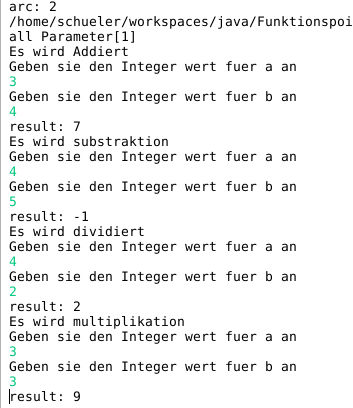
*Multiplizieren:*

**

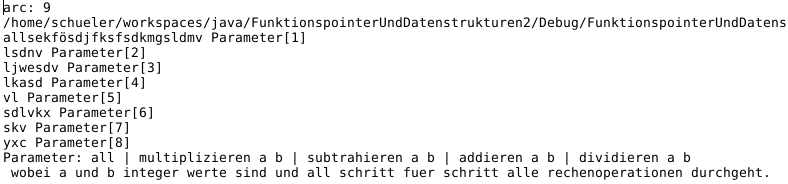
*Dividieren:*



*Geführte Rechenoperationen (all)*



*Falscher Input:*

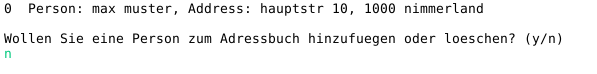
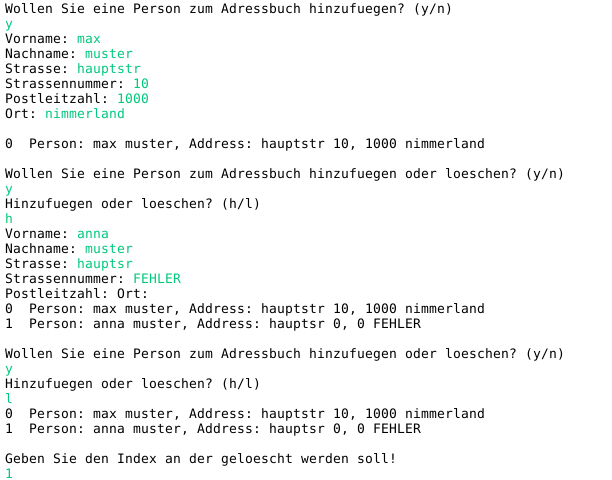


Addressbuch

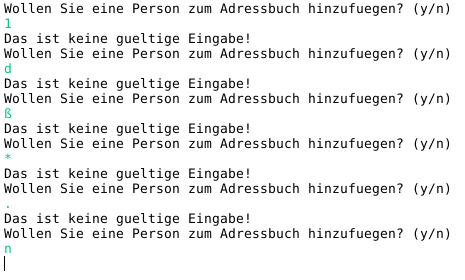
Beim Addressbuch werden die meisten Eingabe so lange wiederholt bis ein gültiger Wert eingegeben wird. Dies ist nicht bei den Eingaben für einen Eintrag so, die Daten werden so fehlerhaft abgespeichert. Da die fehlerhaften Einträge einfach wieder gelöscht werden können. Beim der fehlerhaften Eingabe vom Index, beim Löschen, wird einfach nur ausgegeben das der Index ungültig ist.

**Hier sind einige Testfälle:**

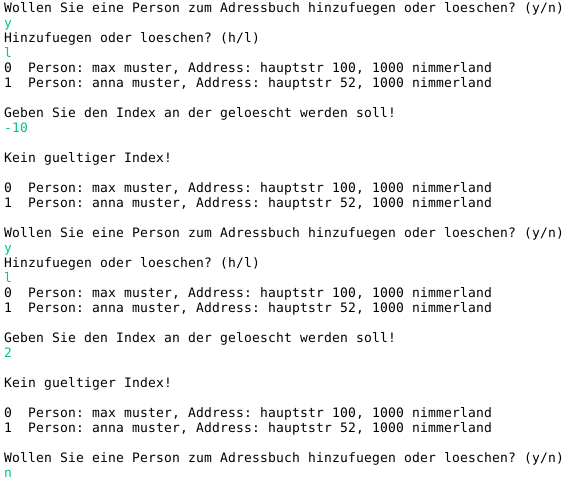
**Hier starte ich das Programm und füge zuerst ein Person mit gültige Eingaben hinzu, und die zweite mit ungültigen. Den 2.Eintrag lösch ich dann, und beende das Programm.**



**Hier starte ich das Programm und versuche fehlerhafte Werte einzugeben und am Ende eine 'n', lso um es zu beenden.**



**Hier habe ich schon 2 Einträge hinzugefügt, und versuche ungültige Indizes einzugeben.**



# Quellenangaben

1. Jürgen Wolf, "Speicher dynamisch reservieren mit »realloc()« und »calloc()«", aktualisierte und erweiterte Auflage 2009, online verfügbar: http://openbook.galileocomputing.de/c\_von\_a\_bis\_z/014\_c\_dyn\_speicherverwaltung\_008.htm#mjaf39b063d2cf58fabe72746ed364671f, zuletzt besucht am: 31.10.2013
2. Jürgen Wolf, "Dynamisches Array mit flexiblen Elementen", aktualisierte und erweiterte Auflage 2009, online verfügbar: http://openbook.galileocomputing.de/  
   c\_von\_a\_bis\_z/021\_c\_dyn\_datenstrukturen\_005.htm, zuletzt besucht am: 31.10.2013
3. Jürgen Wolf, "Direkte Sprünge mit »goto«", aktualisierte und erweiterte Auflage 2009, online verfügbar: http://openbook.galileocomputing.de/c\_von\_a\_bis\_z/  
   008\_c\_kontrollstrukturen\_012.htm, zuletzt besucht am: 31.10.2013