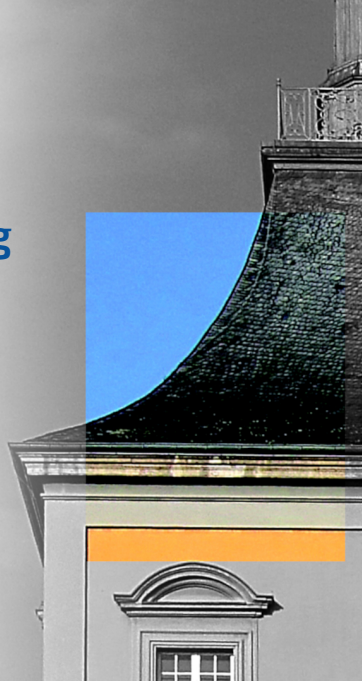
UNIVERSITÄT BONN

Algorithmen und Programmierung

Algorithmen II

Dr. Felix Jonathan Boes
boes@cs.uni-bonn.de
Institut für Informatik



Abstrakte Datentypen und Datenstrukturen

Übergeordnetet Frage

Wie speichern Informatiker:innen „gruppierte“ Daten effizient?

UNIVERSITÄT BONN

Abstrakter Datentyp

Definition (Abstrakter Datentyp)

Ein **abstrakter Datentyp** ist ein (mathematisches oder formalisiertes) Modell für eine Kollektion von Objekten der selben Art, zusammen mit einer Menge von Operationen, die auf dieser Kollektion definiert ist.
Dabei ist sowohl der Speicherverbrauch sowie die Laufzeiten der Operationen in Abhängigkeit der Anzahl der enthaltenen Objekte angegeben.

Datenstruktur

Definition (Datenstruktur)

Eine **Datenstruktur** ist eine explizite Spezifikation eines Objekts, zur Speicherung einer Kollektion von Objekten der selben Art, zusammen mit einer Menge Operationen, die auf dieser Kollektion definiert ist.
Eine Datenstruktur **realisiert** einen abstrakten Datentyp, wenn die Spezifikation der Datenstruktur das Modell des abstrakten Datentyps realsiert und dabei der zugehörige Speicherverbrauch sowie die zugehörigen Laufzeiten eingehalten werden.

Diskussion

Abstrakte Datentypen helfen dabei Algorithmen zu entwerfen und die zugehörigen Laufzeiten zu analysieren.

Wir sind nur an abstrakten Datentypen interessiert, die auch von einer Datenstruktur realisiert (werden können).

Haben Sie Fragen?

Zusammenfassung

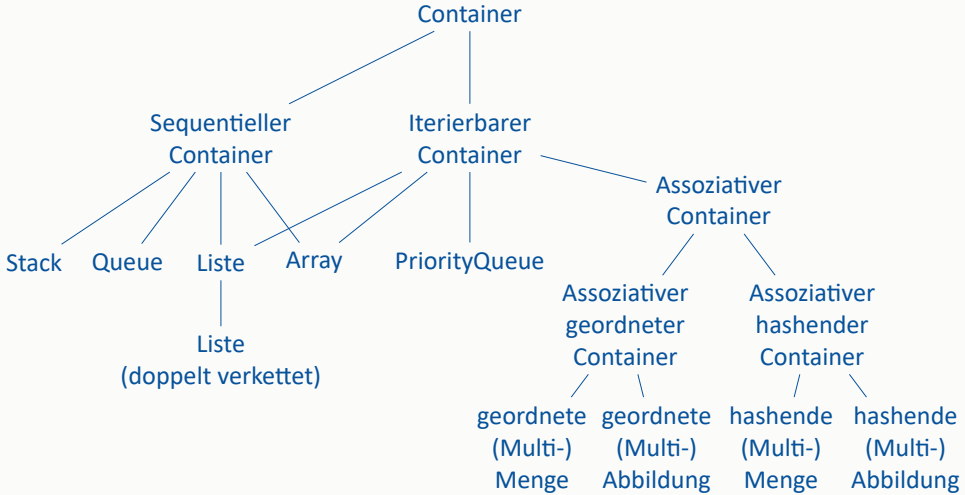
Abstrakte Datentypen modellieren Kollektionen von Objekten, definiert welche Operationen dabei zur Verfügung stehen und wie effizient diese Operationen sind

Datenstrukturen spezifizieren wie Kollektionen von Objekten und zugehörige Operationen verwirklicht werden.

Abstrakte Datentypen werden von Datenstrukturen realisiert

UNIVERSITÄT BONN

Die (wichtigsten) abstrakten Datentypen im Überblick



Offene Fragen

Wie sind die wichtigsten abstrakten Datentypen definiert?

Durch welche Datenstrukturen werden sie realisiert?

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung

Offene Fragen

Was sind sequentielle Container?

Durch welche Datenstrukturen werden sie realisiert?

UNIVERSITÄT BONN

Sequentieller Container (moralisch)

Ein **sequentieller Container** ist ein abstrakter Datentyp, zum Speichern von Objekten desselben Typs T, der folgende Bedingungen erfüllt.

- Die Anzahl der gespeicherten Objekte wird in konstanter Zeit zurückgegeben.
- Es gibt eine **Reihenfolge** auf den enthaltenen Objekten.
- Die einfügenden und entfernenen Operationen erhalten die Reihenfolge.
- Die einfügenden und entfernenen Operationen haben eine Laufzeit von $\mathcal{O}(n)$.
- Falls der Speicherverbrauch des Typs T beschränkt ist, dann liegt der Speicherverbrauch des Container in $\mathcal{O}(n)$.

Bemerkung: Bei spezielleren, sequentiellen Containern ist der Laufzeitaufwand oft $\mathcal{O}(\log(n))$ oder $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Sequentieller Container (formaler)

Ein **sequentieller Container** ist ein abstrakter Datentyp, zum Speichern von Objekten desselben Typs T, der folgende Bedingungen erfüllt. Dabei bezeichnet *n* die Anzahl der gespeicheten Objekte.

- Die Anzahl der gespeicherten Objekte wird in konstanter Zeit zurückgegeben.
- Die enthaltenen Objekten sind **linear geordnet** und die einfügenden und entfernenen Operationen sind mit dieser Ordnung **verträglich**.
- Die einfügenden Operationen fügen ein noch nicht gespeichertes Objekt vor allen, nach allen **oder** zwischen zwei aufeinanderfolgenden gespeicherten Objekte ein. Der Aufwand liegt in $\mathcal{O}(n)$.
- Die entfernenden Operationen entfernen ein gespeichertes Objekt vor allen anderen, nach allen anderen **oder** zwischen zwei direkten Nachbarn. Der Aufwand liegt in $\mathcal{O}(n)$.
- Falls der Speicherverbrauch des Typs T beschränkt ist, dann liegt der Speicherverbrauch des Container in $\mathcal{O}(n)$.

UNIVERSITÄT BONN

Speziellere sequentielle Container

Die folgenden, abstraten Datentypen sind speziellere sequentielle Container. Sie sind für die Informatik grundlegend.

- (dynamisches) Array
- Liste (einfach oder doppelt verkettet)
- Queue
- Stack

Im Folgenden führen wie die oben genannten abstrakten Datentypen ein. Außerdem geben wir jeweils eine realisierende Datenstruktur an.

Im Abschluss diskutieren wir, wie diese abstrakten Datentypen in C++ realisiert werden.

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung - Dynamisches Array

Offene Fragen

Was ist ein dynamisches Array?

Durch welche Datenstruktur wird es realisiert?

UNIVERSITÄT BONN

Dynamisches Array als abstrakter Datentyp

Ein sequentieller Container ist ein **dynamisches Array**, falls die folgenden Eigenschaften erfüllt sind.

- Die Reihenfolge der Objekte ist verträglich mit den Indizes $0, 1, 2, 3, \ldots$ und der Zugriff auf die Objekte ist mithilfe dieser Indizes möglich.
- Objekte können an beliebigen Positionen eingefügt werden.
- Objekte können von beliebigen Positionen entfernt werden.
- Das Einfügen oder Entfernen am Ende hat eine durchschnittliche Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Dynamisches Array realisiert durch eine Datenstruktur I

Objekte desselben Typs haben im virtuellen Adressraum dieselbe Größe[1]. Um ein dynamisches Array als Datenstruktur zu realisieren, wird ein zusammenhängendes Stück Speicher für Objekte desselben Typs reserviert.

Für Objekte vom Typ T der Größe s wird ein dynamisches Array der Länge *n* wie folgt realisiert.

A[0]	A[1]	A[2]	A[...]	A[n-1]

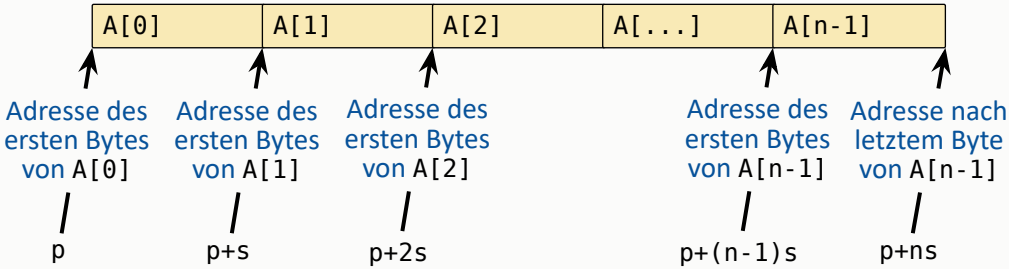
[1] Objekte die zur Laufzeit dynamischen Speicher auf dem Heap reservieren haben streng genommen keine feste Größe. Allerdings haben die Referenzen, mit welchen auf den dynamischen Speicher zugegriffen werden, eine feste Größe. Für die obige Diskussion reicht diese Erkenntnis aus.

UNIVERSITÄT BONN

Dynamisches Array realisiert durch eine Datenstruktur II

Für Objekte vom Typ T der Größe *s* wird ein dynamisches Array der Länge *n* wie folgt realisiert.

Das Objekt zum Index 0 liegt an einer festen Adresse *p*. Das Objekt zum Index 1 liegt an der Adresse $p + s$. Allgemein liegt das Objekt zum Index *i* an der Adresse $p + i \cdot s$.



UNIVERSITÄT BONN

Dynamisches Array realisiert durch eine Datenstruktur III

Für das Einfügen am Ende reserviert man zunächst einen (großen) Block der direkt anschließt. Beim Einfügen nutzt man entweder den noch verfügbaren Platz oder reserviert vorher einen weiteren, anschließenden Block.

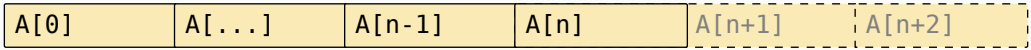
Der Aufwand ist offenbar $\mathcal{O}(1)$, falls kein Speicher reserviert werden muss. Da nur selten[2] Speicher reserviert werden muss, beträgt die durchschnittliche Laufzeit in jedem Fall $\mathcal{O}(1)$.

[2] Eine Möglichkeit ist es, 2^k neue Plätze zu reservieren, sobald das Array 2^k Plätze besetzt hat.

UNIVERSITÄT BONN

Dynamisches Array realisiert durch eine Datenstruktur III

Für das Einfügen am Ende reserviert man zunächst einen (großen) Block der direkt anschließt. Beim Einfügen nutzt man entweder den noch verfügbaren Platz oder reserviert vorher einen weiteren, anschließenden Block.



Der Aufwand ist offenbar $\mathcal{O}(1)$, falls kein Speicher reserviert werden muss. Da nur selten[2] Speicher reserviert werden muss, beträgt die durchschnittliche Laufzeit in jedem Fall $\mathcal{O}(1)$.

[2] Eine Möglichkeit ist es, 2^k neue Plätze zu reservieren, sobald das Array 2^k Plätze besetzt hat.

UNIVERSITÄT BONN

Dynamisches Array realisiert durch eine Datenstruktur IV

Um ein Objekt an einem gegebenen Index einzufügen werden alle bereits gespeicherten Objekte ab dem Index um eine Position verschoben. Falls nötig wird (wie oben) vorher ein weiterer, anschließender Block reserviert.

Der Aufwand ist dabei $\mathcal{O}(n)$, falls kein Speicher reserviert werden muss. Da nur selten Speicher reserviert werden, muss beträgt die durchschnittliche Laufzeit in jedem Fall $\mathcal{O}(n)$.

Das Entfernen von Elementen verhält sich analog. Auch hier sind die Laufzeitanforderungen für Arrays erfüllt.

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung - Liste

Offene Fragen

Was ist eine einfach verkettete Liste?

Durch welche Datenstruktur wird sie realisiert?

UNIVERSITÄT BONN

Einfach verkettete Liste als abstrakter Datentyp

Ein sequentieller Container ist eine **einfach verkettete Liste**, falls die folgenden Eigenschaften erfüllt sind.

- Das erste Objekt der Liste kann zurückgegeben werden.
- Zu jedem Objekt kann der direkten Nachfolger zurückgegeben werden (falls sich das genannte Objekt nicht am Ende der Liste befindet).
- Objekte können an beliebigen Positionen eingefügt werden. Falls das Objekt nicht vor allen anderen eingefügt werden soll, muss der Vorgänger des neuen Objekts angegeben werden.
- Objekte können von beliebigen Positionen entfernt werden. Falls nicht das erste Objekt entfernt werden soll, muss der Vorgänger dieses Objekts angegeben werden.
- Jede hier genannte Operation hat eine Laufzeit von $\mathcal{O}(1)$.

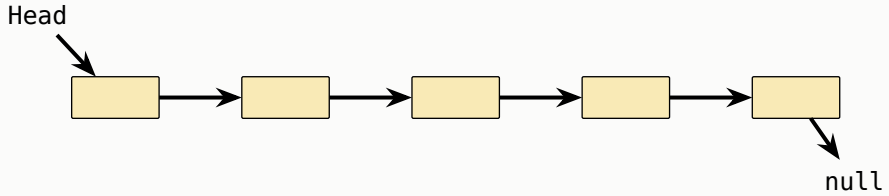
UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur I

Die Objekte einer Liste werden in sogenannten Nodes (Knoten) gespeichert. Jeder Knoten enthält zusätzlich die Information wer der direkte Nachfolger ist (falls ein Nachfolger existiert).

Falls kein Nachfolger existiert, wird das im Knoten codiert. Üblicherweise sagt man dann, dass dieser Knoten nichts (`null`) referenziert.

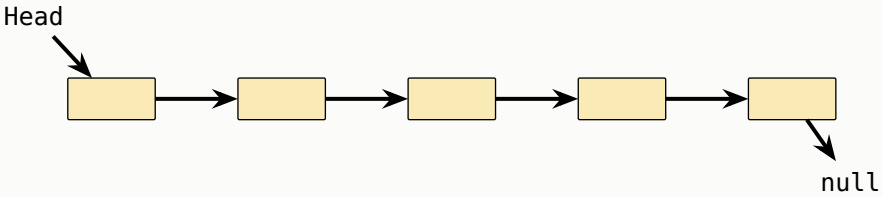
Die Liste merkt sich welcher Knoten der Erste ist. Dieser Knoten wird oft Head (Kopf) genannt.



UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur II

Das Einfügen eines Objekts vor allen anderen wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt.



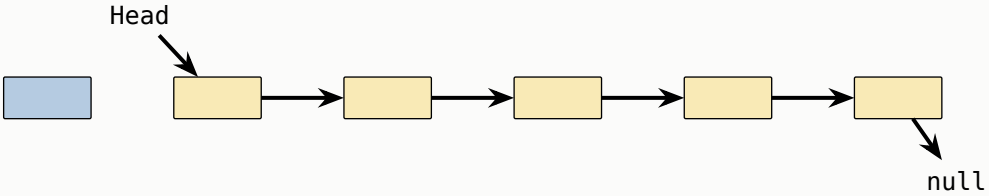
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen des ersten Element verläuft analog. Das Entfernen des ersten Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur II

Das Einfügen eines Objekts vor allen anderen wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt.



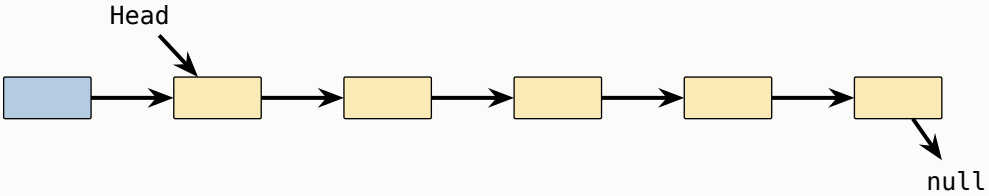
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen des ersten Element verläuft analog. Das Entfernen des ersten Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur II

Das Einfügen eines Objekts vor allen anderen wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt.



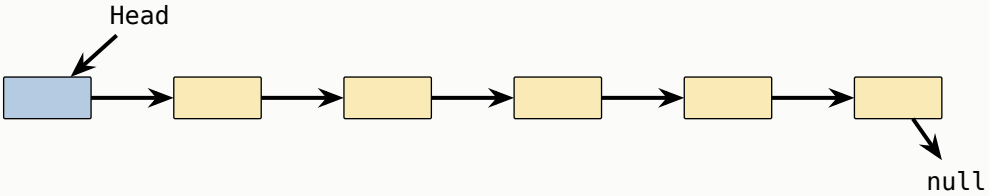
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen des ersten Element verläuft analog. Das Entfernen des ersten Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur II

Das Einfügen eines Objekts vor allen anderen wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt.



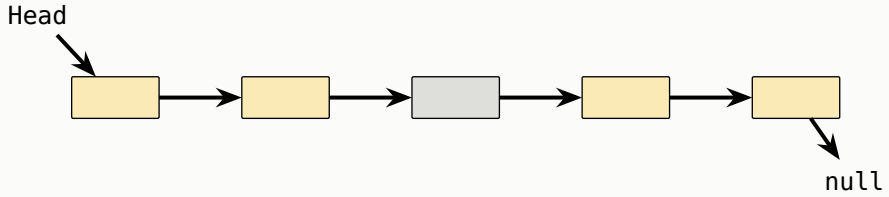
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen des ersten Element verläuft analog. Das Entfernen des ersten Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur III

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



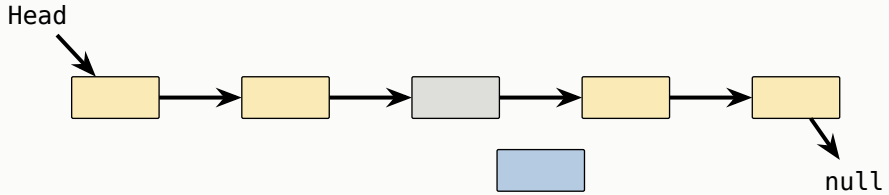
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen eines Elements verläuft analog. Das Entfernen eines Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur III

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



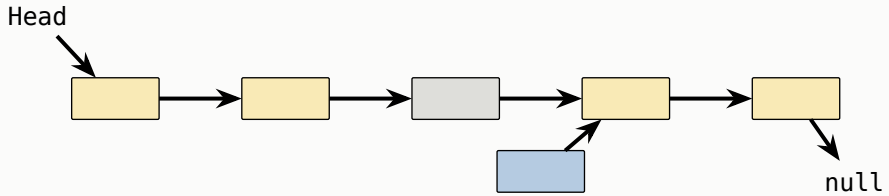
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen eines Elements verläuft analog. Das Entfernen eines Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur III

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



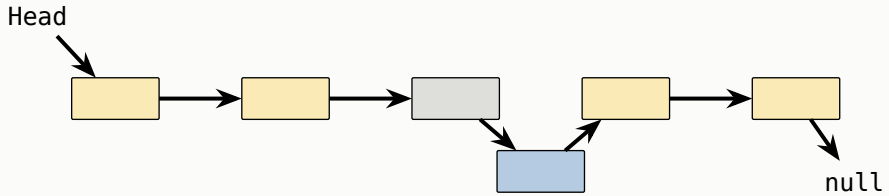
Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen eines Elements verläuft analog. Das Entfernen eines Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Einfach verkettete Liste realisiert durch eine Datenstruktur III

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Das Entfernen eines Elements verläuft analog. Das Entfernen eines Elements hat ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung - Liste (doppelt verkettet)

Offene Fragen

Was ist eine doppelt verkettete Liste?

Durch welche Datenstruktur wird sie realisiert?

UNIVERSITÄT BONN

Doppelt verkettete Liste als abstrakter Datentyp

Ein sequentieller Container ist eine **doppelt verkettete Liste**, falls die folgenden Eigenschaften erfüllt sind.

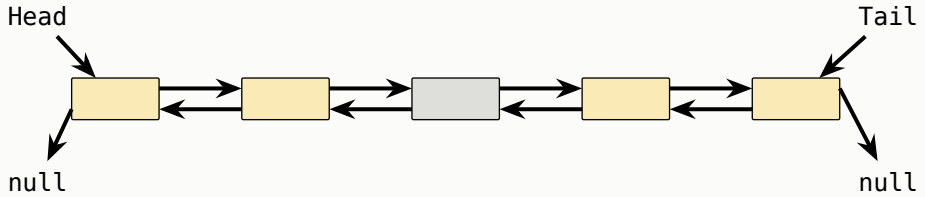
- Das erste und letzte Objekt der Liste kann zurückgegeben werden.
- Zu jedem Objekt kann der direkten Nachfolger und Vorgänger zurückgegeben werden (falls existent).
- Objekte können an beliebigen Positionen eingefügt werden. Falls das Objekt nicht vor allen anderen eingefügt werden soll, muss der Vorgänger des neuen Objekts angegeben werden.
- Objekte können von beliebigen Positionen entfernt werden. Hierbei muss kein Vorgägner angegeben werden.
- Jede hier genannte Operation hat eine Laufzeit von $\mathcal{O}(1)$.

Die doppelt verkettete Liste wird ähnlich zur einfach verketteten Liste realisiert.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur I

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.

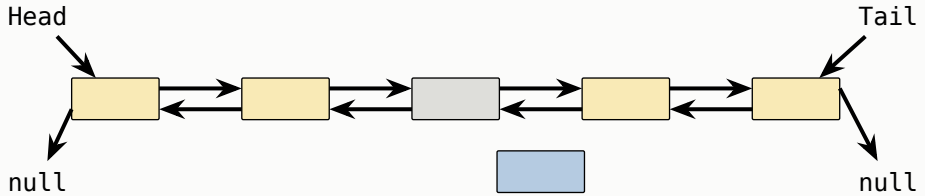


Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Einfügen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur I

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.

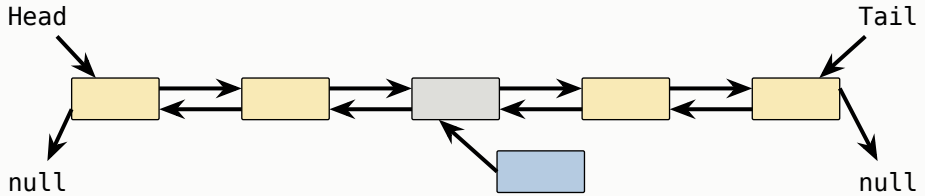


Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Einfügen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur I

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.

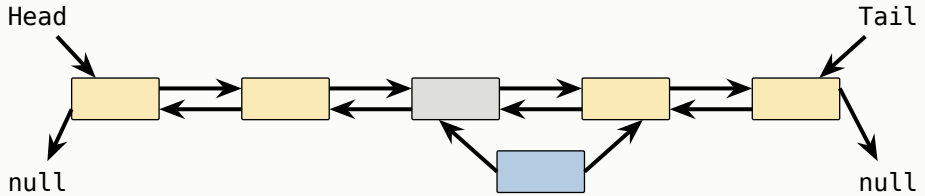


Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Einfügen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur I

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.

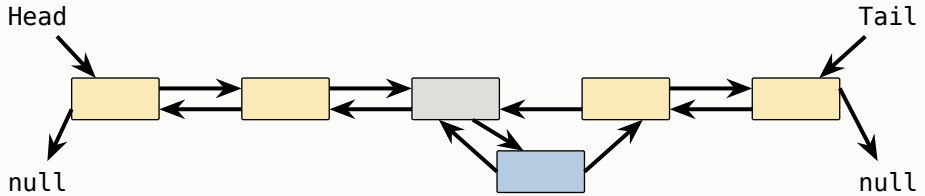


Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Einfügen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur I

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.

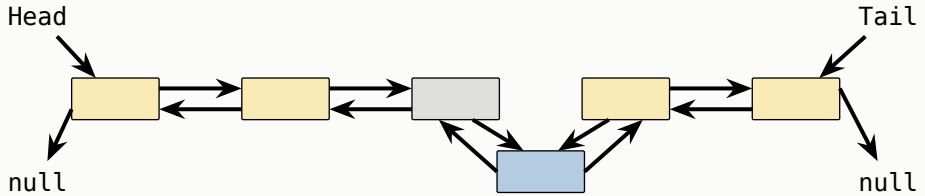


Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Einfügen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur I

Das Einfügen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.

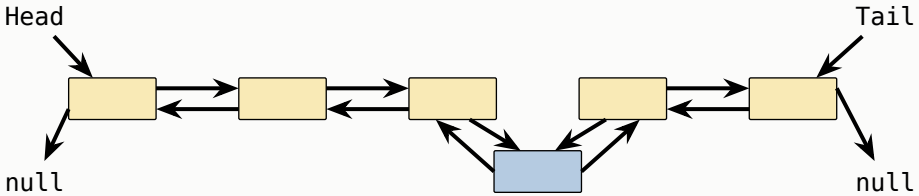


Das Einfügen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Einfügen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur II

Das Entfernen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, zu entfernende Knoten ist dabei blau gefärbt.

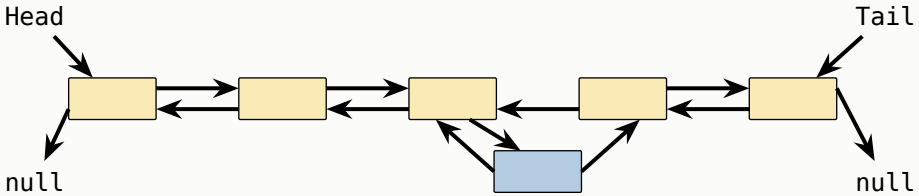


Das Entfernen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur II

Das Entfernen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, zu entfernende Knoten ist dabei blau gefärbt.

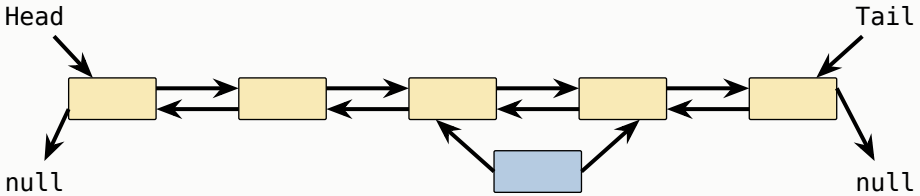


Das Entfernen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur II

Das Entfernen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, zu entfernende Knoten ist dabei blau gefärbt.

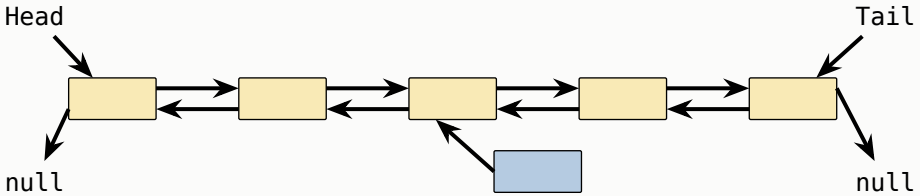


Das Entfernen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur II

Das Entfernen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, zu entfernende Knoten ist dabei blau gefärbt.

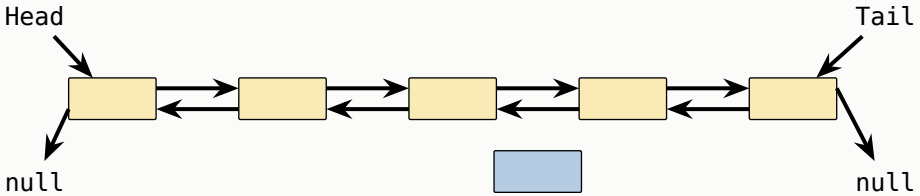


Das Entfernen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Doppelt verkettete Liste realisiert durch eine Datenstruktur II

Das Entfernen eines Objekts bei gegebenem Vorgänger wird hier bildlich erklärt. Der zugehörige, zu entfernende Knoten ist dabei blau gefärbt.



Das Entfernen hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang und am Ende verläuft analog und hat offenbar ebenfalls eine Laufzeit von $\mathcal{O}(1)$.

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung - Queue

Offene Fragen

Was ist eine Queue?

Durch welche Datenstruktur wird sie realisiert?

UNIVERSITÄT BONN

Queue
als abstrakter Datentyp

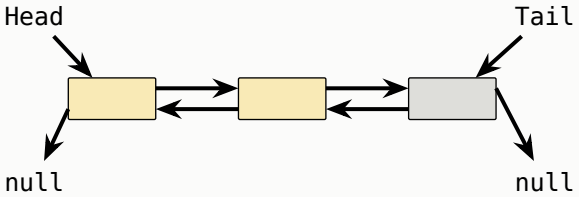
Ein sequentieller Container ist eine **Queue**, falls die folgenden Eigenschaften erfüllt sind.

- Objekte können am Ende der Queue eingefügt werden.
- Objekte können am Anfang der Queue entfernt werden.
- Jede hier genannte Operation hat eine Laufzeit von $\mathcal{O}(1)$.

Die Queue wird ähnlich zur doppelt verketteten Liste realisiert.

Queue
realisiert durch eine Datenstruktur

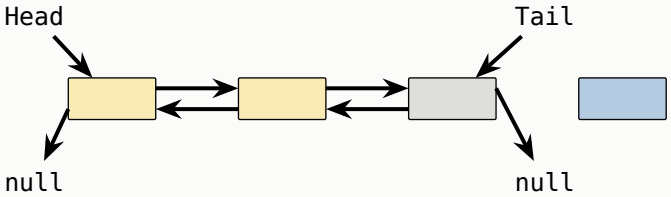
Das Einfügen eines Objekts am Ende wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen am Ende hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang verläuft analog und hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Queue
realisiert durch eine Datenstruktur

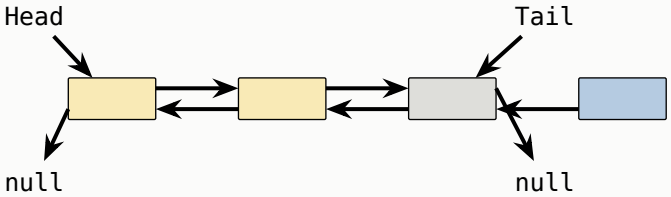
Das Einfügen eines Objekts am Ende wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen am Ende hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang verläuft analog und hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Queue
realisiert durch eine Datenstruktur

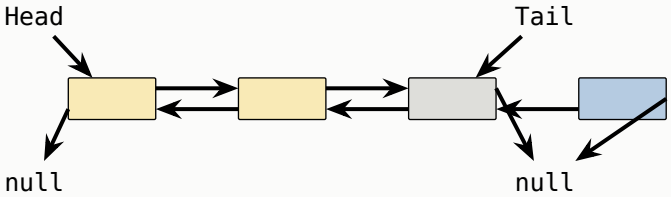
Das Einfügen eines Objekts am Ende wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen am Ende hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang verläuft analog und hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Queue
realisiert durch eine Datenstruktur

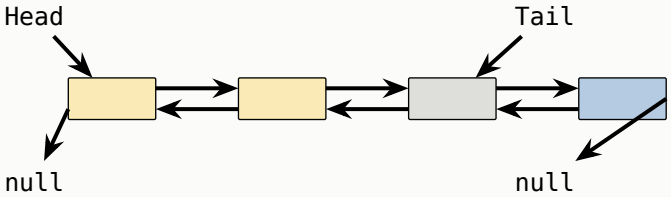
Das Einfügen eines Objekts am Ende wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen am Ende hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang verläuft analog und hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Queue
realisiert durch eine Datenstruktur

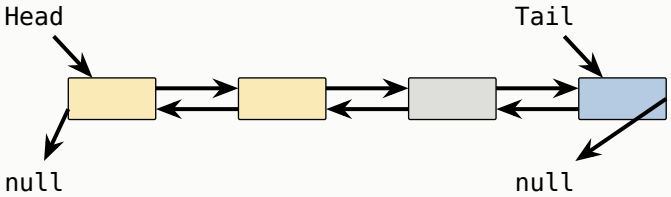
Das Einfügen eines Objekts am Ende wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen am Ende hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang verläuft analog und hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Queue
realisiert durch eine Datenstruktur

Das Einfügen eines Objekts am Ende wird hier bildlich erklärt. Der zugehörige, neue Knoten ist dabei blau gefärbt. Der gegebene Vorgänger ist grau eingefärbt.



Das Einfügen am Ende hat offenbar eine Laufzeit von $\mathcal{O}(1)$. Das Entfernen am Anfang verläuft analog und hat offenbar eine Laufzeit von $\mathcal{O}(1)$.

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung - Stack

Offene Fragen

Was ist ein Stack?

Durch welche Datenstruktur wird er realisiert?

UNIVERSITÄT BONN

Stack
als abstrakter Datentyp

Ein sequentieller Container ist ein **Stack**, falls die folgenden Eigenschaften erfüllt sind.

- Objekte können am Anfang des Stacks eingefügt werden.
- Objekte können am Anfang des Stacks entfernt werden.
- Das Einfügen oder Entfernen am Anfang hat eine durchschnittliche Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Stack
realisiert durch Datenstrukturen

Stacks werden durch die Datenstruktur realisiert, die auch ein Array oder eine Liste realisieren.

Wir erinnern uns außerdem daran, wie Stackframes lokale Variablen von Funktionen organisieren. Dabei bemerken wir, dass Stackframes einen Stack realisieren.

Stacks sind sehr schnell realisiert, implementiert und sind deshalb weit verbreitet.

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Sequentielle Container und deren Realisierung - Zusammenfassung

Zwischenfazit

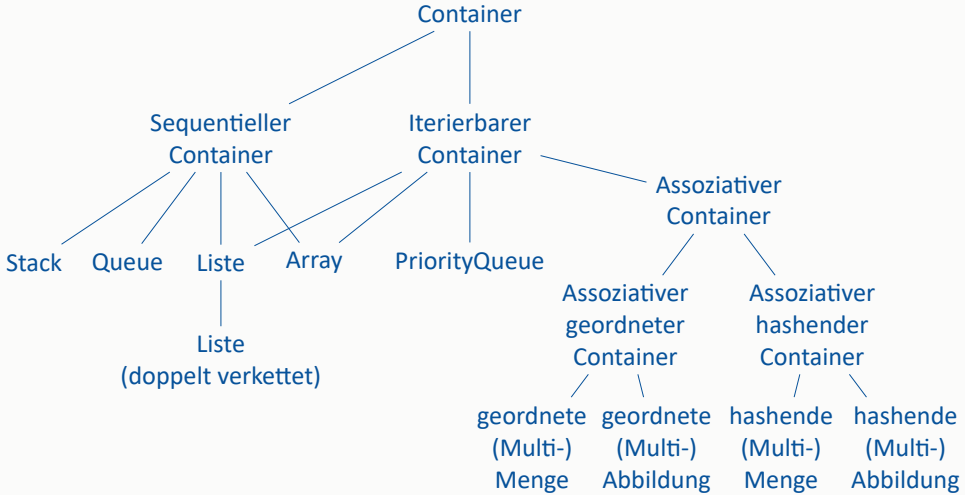
Wir haben die folgenden, sequentiellen Container als abstrakte Datentyp kennen gelernt

Für jeden abstrakten Datentyp haben wir eine realisierende Datenstruktur kennen gelernt

Wir diskutieren noch, wie diese Datenstrukturen in C++ realisiert werden

UNIVERSITÄT BONN

Die (wichtigsten) abstrakten Datentypen im Überblick



Haben Sie Fragen?

Dynamische Speicherverwaltung

Motivation

Erinnerung

Programmiersprachen organisieren Variablen im virtuellen Speicher

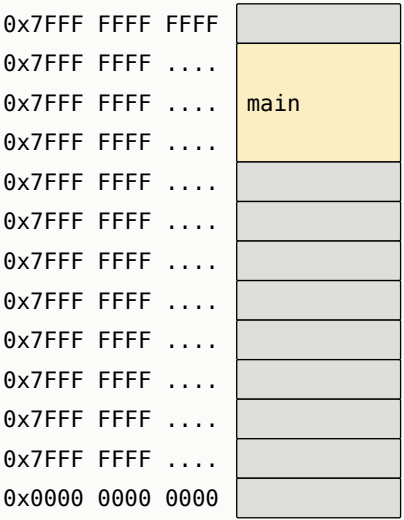
Der Stackspeicher ist funktionsorientiert: lokale Variablen existieren bis die Funktion zurückkehrt

Wiederholung
Stackframes am Beispiel

```
void g() {
  ...
}

void f() {
  ...
  g();
  ...
}

int main() { ←
  f();
  return 0;
}
```



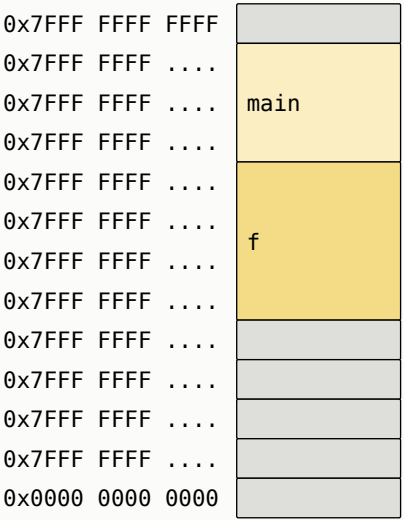
UNIVERSITÄT BONN

Wiederholung
Stackframes am Beispiel

```
void g() {
  ...
}

void f() {   ←
  ...
  g();
  ...
}

int main() {
  f();       ←
  return 0;
}
```



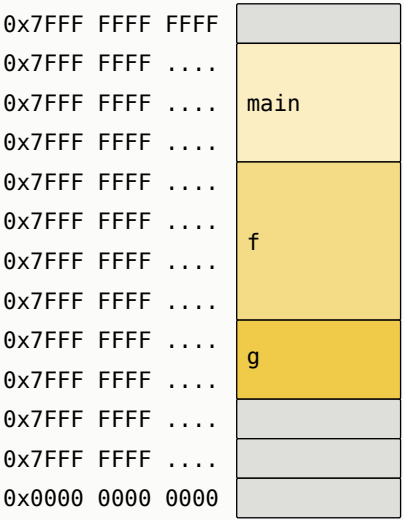
UNIVERSITÄT BONN

Wiederholung
Stackframes am Beispiel

```
void g() {      ←
  ...
}

void f() {
  ...
  g();          ←
  ...
}

int main() {
  f();          ←
  return 0;
}
```



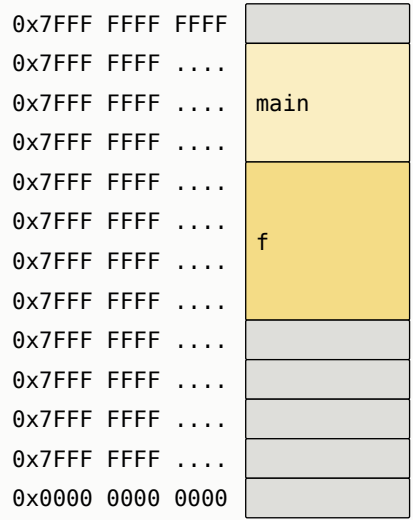
UNIVERSITÄT BONN

Wiederholung
Stackframes am Beispiel

```
void g() {
  ...
}

void f() {
  ...
  g();
  ...
}

int main() {
  f();
  return 0;
}
```



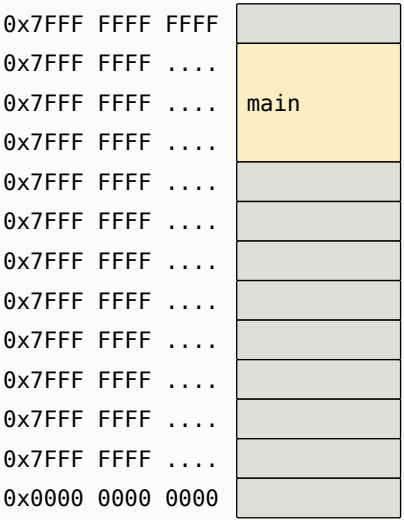
UNIVERSITÄT BONN

Wiederholung
Stackframes am Beispiel

```
void g() {
  ...
}

void f() {
  ...
  g();
  ...
}

int main() {
  f();
  return 0;  ←
}
```

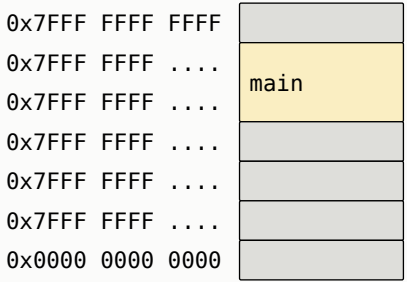


Wie implementiert man Listen in C++?

Wir wollen Listen in C++ implementieren. Der unternommene Implementierungsversuch führt ganz natürlich zu einer Frage, die wir im Folgenden beantworten.

```
class Liste {
  void push_front(int x) {
    // Wo und wie wird x gespeichert?
  }
};

int main() {
  Liste l;          ←
  l.push_front(42);
}
```



Beim Einfügen eines neuen Knoten, muss dieser Knoten erstellt werden und die Rückkehr der Funktion `push_front` überleben. Dazu muss der neue Knoten dynamisch erzeugt und auf dem Heapspeicher[3] verwaltet werden.

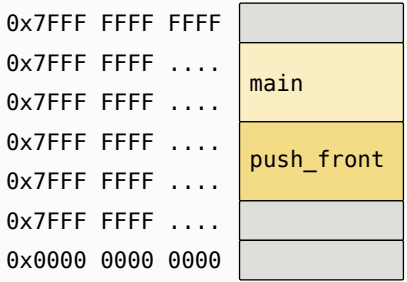
[3] Erinnern Sie sich an Vorlesung *Imperative Programmierung mit C++ Teil 3 & 5*

Wie implementiert man Listen in C++?

Wir wollen Listen in C++ implementieren. Der unternommene Implementierungsversuch führt ganz natürlich zu einer Frage, die wir im Folgenden beantworten.

```
class Liste {
  void push_front(int x) { ←
    // Wo und wie wird x gespeichert?
  }
};

int main() {
  Liste l;
  l.push_front(42); ←
}
```



Beim Einfügen eines neuen Knoten, muss dieser Knoten erstellt werden und die Rückkehr der Funktion `push_front` überleben. Dazu muss der neue Knoten dynamisch erzeugt und auf dem Heapspeicher[3] verwaltet werden.

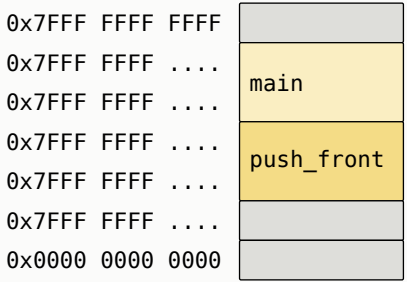
[3] Erinnern Sie sich an Vorlesung *Imperative Programmierung mit C++ Teil 3 & 5*

Wie implementiert man Listen in C++?

Wir wollen Listen in C++ implementieren. Der unternommene Implementierungsversuch führt ganz natürlich zu einer Frage, die wir im Folgenden beantworten.

```
class Liste {
  void push_front(int x) {
    // Wo und wie wird x gespeichert?
  } ←
};

int main() {
  Liste l;
  l.push_front(42); ←
}
```



Beim Einfügen eines neuen Knoten, muss dieser Knoten erstellt werden und die Rückkehr der Funktion `push_front` überleben. Dazu muss der neue Knoten dynamisch erzeugt und auf dem Heapspeicher[3] verwaltet werden.

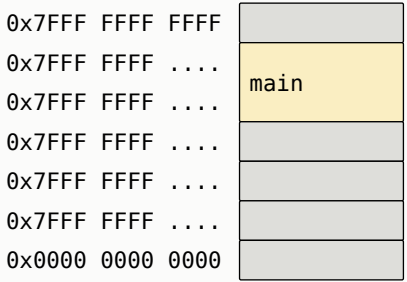
[3] Erinnern Sie sich an Vorlesung *Imperative Programmierung mit C++ Teil 3 & 5*

Wie implementiert man Listen in C++?

Wir wollen Listen in C++ implementieren. Der unternommene Implementierungsversuch führt ganz natürlich zu einer Frage, die wir im Folgenden beantworten.

```
class Liste {
  void push_front(int x) {
    // Wo und wie wird x gespeichert?
  }
};

int main() {
  Liste l;
  l.push_front(42); ←
}
```



Beim Einfügen eines neuen Knoten, muss dieser Knoten erstellt werden und die Rückkehr der Funktion `push_front` überleben. Dazu muss der neue Knoten dynamisch erzeugt und auf dem Heapspeicher[3] verwaltet werden.

[3] Erinnern Sie sich an Vorlesung *Imperative Programmierung mit C++ Teil 3 & 5*

Dynamische Speicherverwaltung

Dynamische Speicherverwaltung im Allgemeinen

Offene Frage

Wie löst man die Lebensdauer eines Objekts von der Funktionsrückkehr?

UNIVERSITÄT BONN

Virtuelle Adressräume im Allgemeinen

Auf aktuellen Endnutzerrechnern steht jedem Nuzterprozess ein eigener, virtueller Adressraum zur Verfügung. Imperative und objektorientierte Programmiersprachen organisieren Variablen in diesem virtuellen Speicher.

Der virtuelle Adressraum besteht vereinfacht gesagt aus **Programmspeicher**, **Stackspeicher** und **Heapspeicher**.

UNIVERSITÄT BONN

Der Stackspeicher im Allgemeinen

Der **Stackspeicher** ist funktionsorientiert. Bei jedem Funktionsaufruf wird ein (zur Funktion gehörender) Stackframe angelegt. Die lokalen Variablen der Funktion werden dort organisiert.

Bei der Funktionsrückkehr wird der zugehörige Stackframe entfernt. Die angelegten lokalen Variablen sind anschließend nicht mehr verfügbar. Durch das Entfernen des Stackframe der **zurückkehrenden Funktion** wird der Stackframe der **aufrufenden Funktion** wieder erreicht.

UNIVERSITÄT BONN

Der Heapspeicher und Adressvariablen im Allgemeinen

Wir haben bei unserem Implementierungsversuch verstanden, dass es konzeptionell nötig ist, die Lebensdauer eines Objekts von der Funktionsrückkehr zu entkoppeln. Um dies in imperativen und objektorientierten Programmiersprachen zu erreichen, werden **Adressvariablen** und der **Heapspeicher** verwendet.

UNIVERSITÄT BONN

Adressvariablen im Allgemeinen

Eine **Adressvariable** speichert **virtuelle Adressen** eines **festgelegten Typs**. Der Wert einer Adressvariable ist also immer eine virtuelle Adresse. Ob an der gegebenen Adresse ein Objekt lebt, muss durch die Programierer:innen sichergestellt werden.

Da jede Adressvariable die Adresse eines festen Typs speichert, interpretiert die Adressvariable dir dort liegenden Bytes als die Bytes des festgelegten Typs. Durch diese, direkte Interpretation dieser Bytes, kann auf dem dort liegenden Objekt lesend und verändernd operiert werden.

Da sich Adressvariablen den Ort eines Objekts merken, werden Adressvariablen auch **Zeiger** ode **Pointer** genannt. Deshalb wird der Wert einer Adressvariable auch durch einen, auf den Ort zeigenden Pfeil dargestellt.

UNIVERSITÄT BONN

Der Heapspeicher im Allgemeinen

Der **Heapspeicher** wird verwendet, um Objekte **dynamisch** zu organisieren, also unabhängig vom aktuellen Stackframe.

Die jeweilige Programmiersprache definiert, wie Objekte auf dem Heapspeicher angelegt werden und wie diese wieder entfernt werden. In jedem Fall wird bei der Objekterzeugung auf dem Heap sowohl das Objekt angelegt als auch ein Zeiger auf dieses Objekt zurückgegeben. Bei dynamischen Arrays haben wir das schon (kurz) gesehen.

UNIVERSITÄT BONN

Wie werden Objekte vom Heap entfernt?

Die jeweilige Programmiersprache legt fest, wann und wie Objekte vom Heap entfernt werden. Jede Programmiersprache legt dabei fest, wieviel Freiheit die Nutzer:innen zur Verfügung haben. Hierbei gibt es nicht das *allgemein beste Vorgehen*.

UNIVERSITÄT BONN

Typische Strategien um Objekte vom Heap zu entfernen

Die folgenden drei Strategien zur Objektentfernung finden oft Anwendung.

Explizites Aufräumen (C, C++) Die Nutzer:innen müssen einen expliziten Befehl abgeben. Hier ist maximale Effizienz und minimale Nachvollziehbarkeit gegeben. Wird das Aufräumen an einer Heapadresse vergessen, lebt der verwendete Heapspeicher bis zum Programmende.

Implizites Aufräumen (C++) Der Speicher wird aufgeräumt, falls gar kein Zeiger mehr auf den zugehörigen Heapspeicher zeigt. Hier ist sehr hohe Effizienz und viel Nachvollziehbarkeit gegeben. Aber, falls auf dem Heap ein Kreis von aufeinander zeigenden Zeigern existiert, lebt der verwendete Heapspeicher bis zum Programmende.

Vollautomatisch (Java, Python) Alle Stackvariablen sind Adressvariablen und alle Objekte liegen auf dem Heap. Regelmäßig werden alle Objekte entfernt, die nicht mehr vom Stack aus erreichbar sind. Hier ist weniger Effizienz und sehr hohe Nachvollziehbarkeit gegeben. Der Heapspeicher wird garantiert aufgeräumt.

Zusammenfassung

Man löst die Lebensdauer eines Objekts von der Funktionsrückkehr, indem die Variable auf dem Heap mithilfe von Adressvariablen organisiert wird

Wie Objekte vom Heapspeicher entfernt werden, wird durch die Programmiersprache festgelegt

Haben Sie Fragen?