

Angewandte Informatik

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

29. Januar 2018

Angewandte Informatik

Hardwarenahe Programmierung

<https://gitlab.cvh-server.de/pgerwinski/hp.git>

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Hardwarenahe Programmierung
- 5 Algorithmen
- 6 Objektorientierte Programmierung
- 7 Datenstrukturen
 - 7.1 Stack und FIFO
 - 7.2 Verkettete Listen
 - 7.3 Bäume

7 Datenstrukturen

7.1 Stack und FIFO

Im letzten Praktikumsversuch:

- Array nur zum Teil benutzt
- Variable speichert genutzte Länge
- Elemente hinten anfügen oder entfernen

→ Stack

- hinten anfügen/entfernen: $\mathcal{O}(1)$
- vorne oder in der Mitte einfügen/entfernen: $\mathcal{O}(n)$

Auch möglich:

- Array nur zum Teil benutzt
- 2 Variable speichern genutzte Länge (ringförmig)
- Elemente hinten anfügen oder vorne entfernen

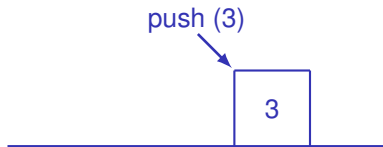
→ FIFO

- vorne oder hinten anfügen oder entfernen: $\mathcal{O}(1)$
- in der Mitte einfügen/entfernen: $\mathcal{O}(n)$

7 Datenstrukturen

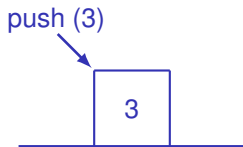
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

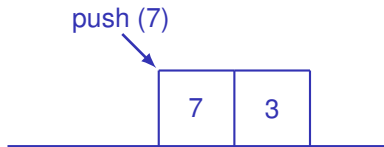


LIFO = Stack = Stapel

7 Datenstrukturen

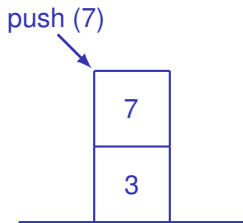
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“



LIFO = Stack = Stapel

7 Datenstrukturen

7.1 Stack und FIFO

„First In – First Out“

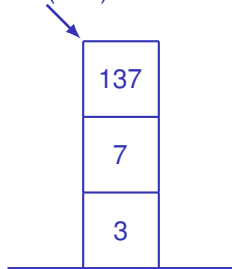
push (137)



FIFO = Queue = Reihe

„Last In – First Out“

push (137)

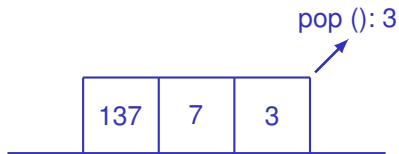


LIFO = Stack = Stapel

7 Datenstrukturen

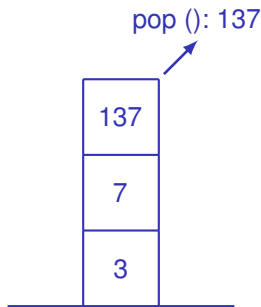
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

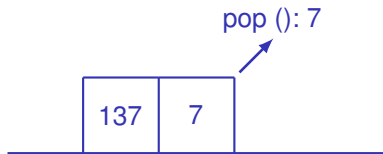


LIFO = Stack = Stapel

7 Datenstrukturen

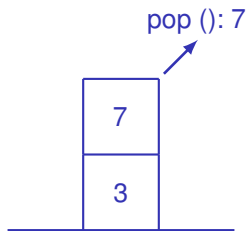
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

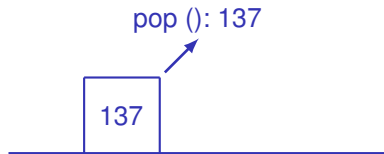


LIFO = Stack = Stapel

7 Datenstrukturen

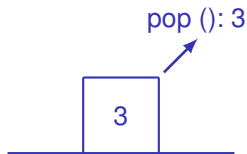
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

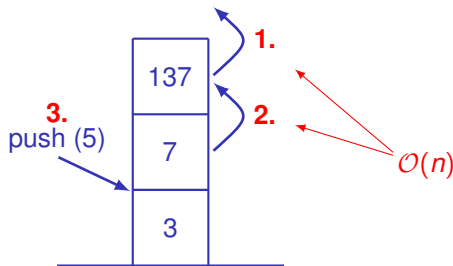


LIFO = Stack = Stapel

7 Datenstrukturen

7.1 Stack und FIFO

Array (Stack, FIFO):
in der Mitte einfügen

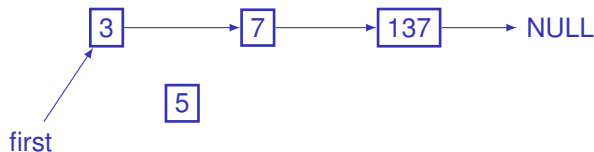


In Array (Stack, FIFO) ...

- einfügen: $\mathcal{O}(n)$
- suchen: $\mathcal{O}(n)$
- geschickt suchen: $\mathcal{O}(\log n)$
- beim Einfügen sortieren:
 ~~$\mathcal{O}(n \log n)$~~ $\mathcal{O}(n^2)$

7 Datenstrukturen

7.2 Verkettete Listen

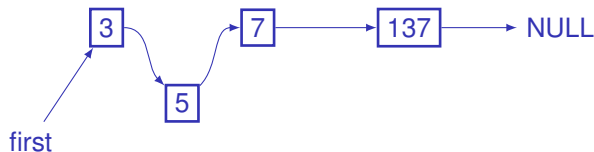


- Jeder Datensatz enthält einen Zeiger auf das nächste Element.
- Beim letzten Element zeigt der Zeiger auf **NULL**.
- Eine Variable zeigt auf das erste Element.
- Wenn die Liste leer ist, zeigt die Variable auf **NULL**.

→ (einfach) **verkettete Liste**

7 Datenstrukturen

7.2 Verkettete Listen



- Jeder Datensatz enthält einen Zeiger auf das nächste Element.
- Beim letzten Element zeigt der Zeiger auf **NULL**.
- Eine Variable zeigt auf das erste Element.
- Wenn die Liste leer ist, zeigt die Variable auf **NULL**.

→ (einfach) **verkettete Liste**

7 Datenstrukturen

7.2 Verkettete Listen

In Array (Stack, FIFO) ...

- in der Mitte einfügen: $\mathcal{O}(n)$
- wahlfreier Zugriff: $\mathcal{O}(1)$
- suchen: $\mathcal{O}(n)$
- geschickt suchen: $\mathcal{O}(\log n)$
- beim Einfügen sortieren:
 ~~$\mathcal{O}(n \log n)$~~ $\mathcal{O}(n^2)$

In (einfach) verkettete/r Liste ...

- in der Mitte einfügen: $\mathcal{O}(1)$
- wahlfreier Zugriff: $\mathcal{O}(n)$
- suchen: $\mathcal{O}(n)$
- ~~geschickt~~ suchen: $\mathcal{O}(n)$
- beim Einfügen sortieren:
 ~~$\mathcal{O}(n \log n)$~~ $\mathcal{O}(n^2)$

In (ausbalancierten) Bäumen ...

- in der Mitte einfügen: $\mathcal{O}(\log n)$
- wahlfreier Zugriff: $\mathcal{O}(\log n)$
- suchen: $\mathcal{O}(\log n)$
- beim Einfügen sortieren:
 $\mathcal{O}(n \log n)$

7 Datenstrukturen

7.3 Bäume

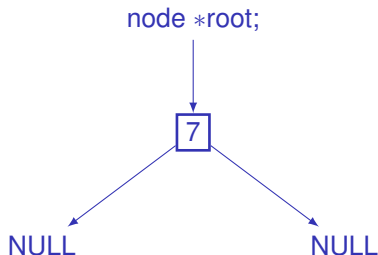
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```

```
node *root;
```

7 Datenstrukturen

7.3 Bäume

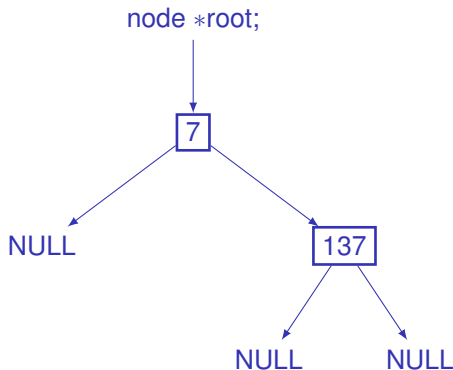
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```



7 Datenstrukturen

7.3 Bäume

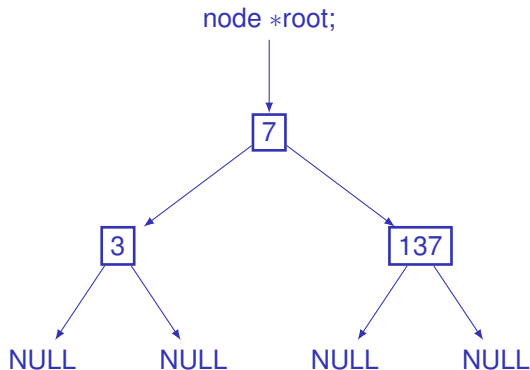
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```



7 Datenstrukturen

7.3 Bäume

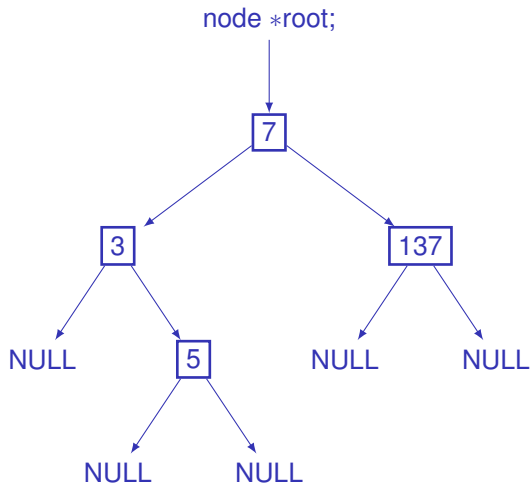
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```



7 Datenstrukturen

7.3 Bäume

```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```

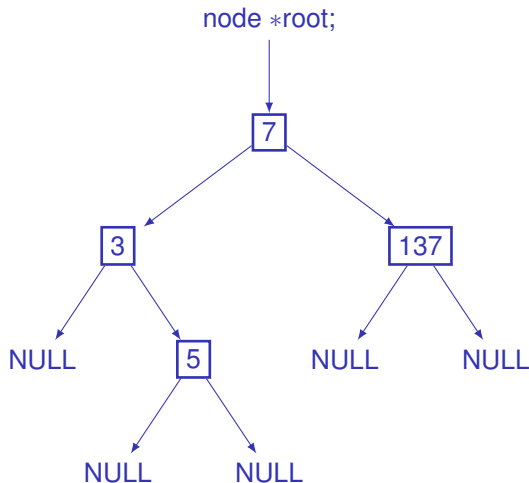


7 Datenstrukturen

7.3 Bäume

```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```

- Einfügen: rekursiv, $\mathcal{O}(\log n)$
- Suchen: rekursiv, $\mathcal{O}(\log n)$
- beim Einfügen sortieren:
rekursiv, $\mathcal{O}(n \log n)$

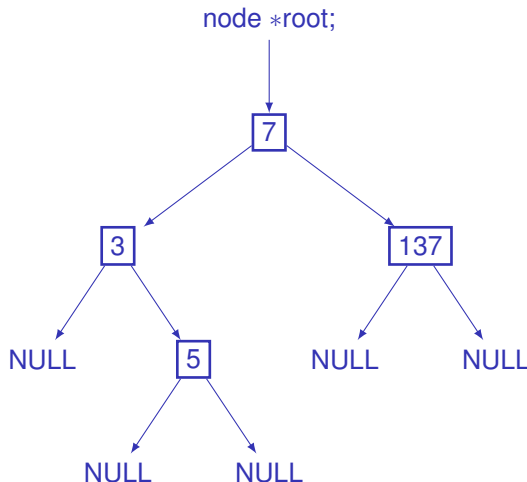


7 Datenstrukturen

7.3 Bäume

```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```

- Einfügen: rekursiv, $\mathcal{O}(\log n)$
- Suchen: rekursiv, $\mathcal{O}(\log n)$
- beim Einfügen sortieren:
rekursiv, $\mathcal{O}(n \log n)$
- **Worst Case: $\mathcal{O}(n^2)$**
vorher bereits sortiert
→ balancierte Bäume
Anwendung: Datenbanken



Angewandte Informatik

Hardwarenahe Programmierung

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Algorithmen
- 5 Hardwarenahe Programmierung
- 6 Objektorientierte Programmierung
- 7 Datenstrukturen
 - 7.1 Stack und FIFO
 - 7.2 Verkettete Listen
 - 7.3 Bäume

*Viel Erfolg
bei der Klausur!*