

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

12. Dezember 2022

Hardwarenahe Programmierung

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

- 1 Einführung**
- 2 Einführung in C**
- 3 Bibliotheken**
- 4 Hardwarenahe Programmierung**
- 5 Algorithmen**
- 6 Objektorientierte Programmierung**
 - ...
 - 6.3 Unions
 - 6.4 Virtuelle Methoden
 - 6.5 Beispiel: Graphische Benutzeroberfläche (GUI)
 - 6.6 Ausblick: C++
- 7 Datenstrukturen**
 - 7.1 Stack und FIFO
 - 7.2 Verkettete Listen
 - 7.3 Bäume

6.4 Virtuelle Methoden

```
void print_object (t_object *this)
{
    if (this->base.type == T_INTEGER)
        printf ("Integer:_%d\n", this->integer.content);
    else if (this->base.type == T_STRING)
        printf ("String:_%s\n", this->string.content);
}
```

if-Kette:
wird unübersichtlich



Zeiger auf Funktionen

```
void print_integer (t_object *this)
{
    printf ("Integer:_%d\n", this->integer.content);
}
```

```
void print_string (t_object *this)
{
    printf ("String:_%s\n", this->string.content);
}
```

6.4 Virtuelle Methoden

Zeiger auf Funktionen

```
void (*print) (t_object *this);
```


das, worauf print zeigt,
ist eine Funktion

- Objekt enthält Zeiger auf Funktion

```
typedef struct  
{  
    void (*print) (union t_object *this);  
    int content;  
} t_integer;
```

6.4 Virtuelle Methoden

Zeiger auf Funktionen

```
void (*print) (t_object *this);
```


das, worauf print zeigt,
ist eine Funktion

- Objekt enthält Zeiger auf Funktion
- Konstruktor initialisiert diesen Zeiger

```
t_object *new_integer (int i)
{
    t_object *p = malloc (sizeof (t_integer));
    p->integer.print = print_integer;
    p->integer.content = i;
    return p;
}
```

```
typedef struct
{
    void (*print) (union t_object *this);
    int content;
} t_integer;
```

6.4 Virtuelle Methoden

Zeiger auf Funktionen

```
void (*print) (t_object *this);
```


das, worauf print zeigt,
ist eine Funktion

- Objekt enthält Zeiger auf Funktion
- Konstruktor initialisiert diesen Zeiger
- Aufruf: „automatisch“ die richtige Funktion

```
for (int i = 0; object[i]; i++)  
    object[i]—>base.print (object[i]);
```

```
typedef struct  
{  
    void (*print) (union t_object *this);  
    int content;  
} t_integer;
```

6.4 Virtuelle Methoden

Zeiger auf Funktionen

```
void (*print) (t_object *this);
```


das, worauf print zeigt,
ist eine Funktion

- Objekt enthält Zeiger auf Funktion
- Konstruktor initialisiert diesen Zeiger
- Aufruf: „automatisch“ die richtige Funktion
- in größeren Projekten:
Objekt enthält Zeiger auf Tabelle von Funktionen

```
typedef struct  
{  
    void (*print) (union t_object *this);  
    int content;  
} t_integer;
```

6.5 Beispiel: Graphische Benutzeroberfläche (GUI)

```
#include <gtk/gtk.h>
```

```
int main (int argc, char **argv)
```

```
{  
    gtk_init (&argc, &argv);  
    GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);  
    gtk_window_set_title (GTK_WINDOW (window), "Hello");  
    g_signal_connect (window, "destroy", G_CALLBACK (gtk_main_quit), NULL);  
    GtkWidget *vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 5);  
    gtk_container_add (GTK_CONTAINER (window), vbox);  
    gtk_container_set_border_width (GTK_CONTAINER (vbox), 10);  
    GtkWidget *label = gtk_label_new ("Hello, world!");  
    gtk_container_add (GTK_CONTAINER (vbox), label);  
    GtkWidget *button = gtk_button_new_with_label ("Quit");  
    g_signal_connect (button, "clicked", G_CALLBACK (gtk_main_quit), NULL);  
    gtk_container_add (GTK_CONTAINER (vbox), button);  
    gtk_widget_show (button);  
    gtk_widget_show (label);  
    gtk_widget_show (vbox);  
    gtk_widget_show (window);  
    gtk_main ();  
    return 0;  
}
```



**Praktikumsversuch:
Objektorientiertes Zeichenprogramm**

6.6 Ausblick: C++

```
typedef struct  
{  
    void (*print) (union t_object *this);  
} t_base;
```

```
typedef struct  
{  
    void (*print) (...);  
    int content;  
} t_integer;
```

```
typedef struct  
{  
    void (*print) (union t_object *this);  
    char *content;  
} t_string;
```

6.6 Ausblick: C++

```
struct TBase  
{  
    virtual void print (void);  
};
```

```
struct TInteger: public TBase  
{  
    virtual void print (void);  
    int content;  
};
```

```
struct TString: public TBase  
{  
    virtual void print (void);  
    char *content;  
};
```

Hardwarenahe Programmierung

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

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Hardwarenahe Programmierung
- 5 Algorithmen
- 6 Objektorientierte Programmierung
 - ...
 - 6.3 Unions
 - 6.4 Virtuelle Methoden
 - 6.5 Beispiel: Graphische Benutzeroberfläche (GUI)
 - 6.6 Ausblick: C++
- 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

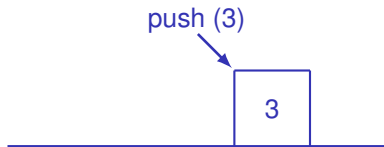
→ FIFO

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

7 Datenstrukturen

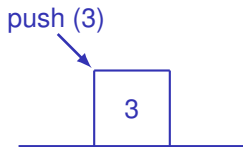
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

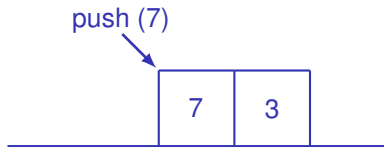


LIFO = Stack = Stapel

7 Datenstrukturen

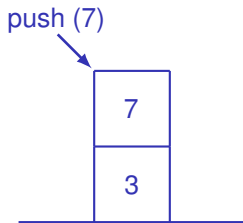
7.1 Stack und FIFO

„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“

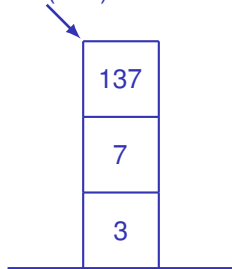
push (137)



FIFO = Queue = Reihe

„Last In – First Out“

push (137)

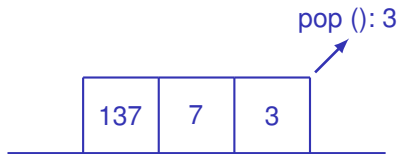


LIFO = Stack = Stapel

7 Datenstrukturen

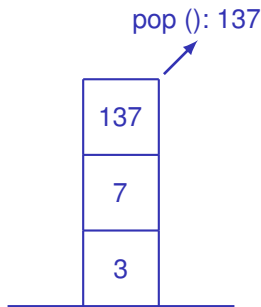
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

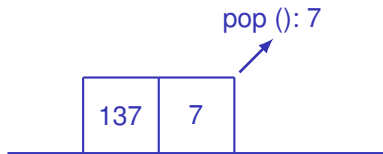


LIFO = Stack = Stapel

7 Datenstrukturen

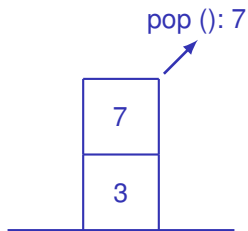
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

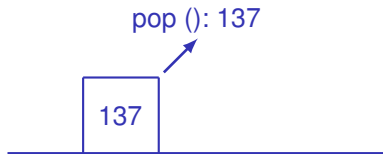


LIFO = Stack = Stapel

7 Datenstrukturen

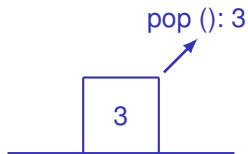
7.1 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“

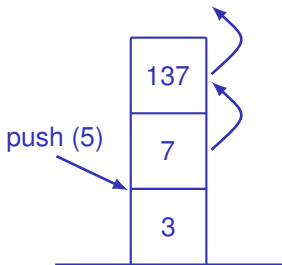


LIFO = Stack = Stapel

7 Datenstrukturen

7.1 Stack und FIFO

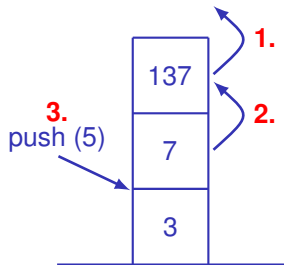
Array (Stack, FIFO):
in der Mitte einfügen



7 Datenstrukturen

7.1 Stack und FIFO

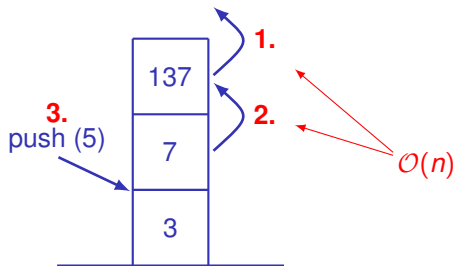
Array (Stack, FIFO):
in der Mitte einfügen



7 Datenstrukturen

7.1 Stack und FIFO

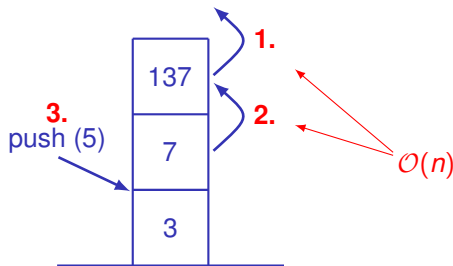
Array (Stack, FIFO):
in der Mitte einfügen



7 Datenstrukturen

7.1 Stack und FIFO

Array (Stack, FIFO):
in der Mitte einfügen

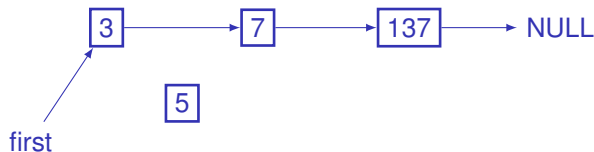


In Array (Stack, FIFO) ...

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

7 Datenstrukturen

7.2 Verkettete Listen

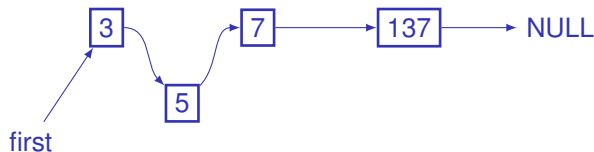


- Jeder Datensatz enthält einen Zeiger auf das nächste Element.
- Beim letzten Element zeigt der Zeiger auf **NULL**.
- Eine Variable zeigt auf das erste Element.
- Wenn die Liste leer ist, zeigt die Variable auf **NULL**.

→ (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:
 ~~$\mathcal{O}(n \log n)$~~ $\mathcal{O}(n^2)$

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

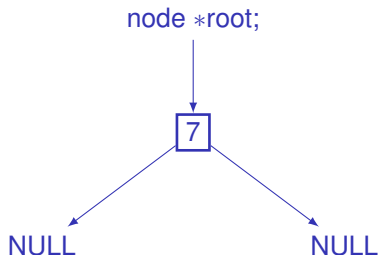
```
node *root;
```

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

7 Datenstrukturen

7.3 Bäume

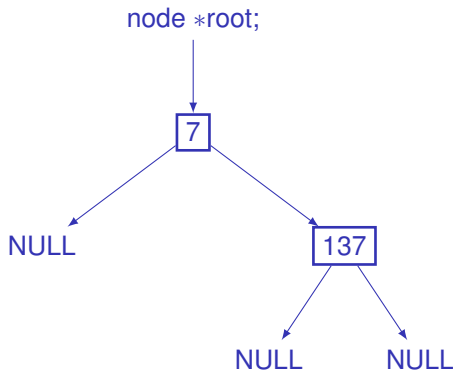
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```



7 Datenstrukturen

7.3 Bäume

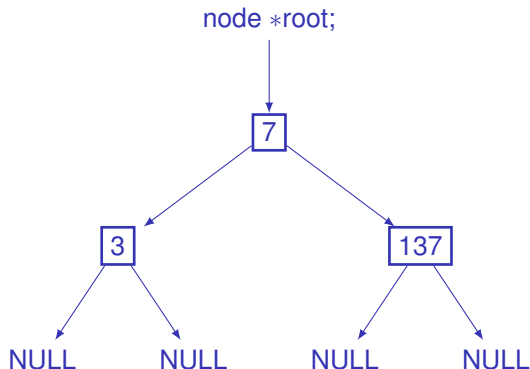
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```



7 Datenstrukturen

7.3 Bäume

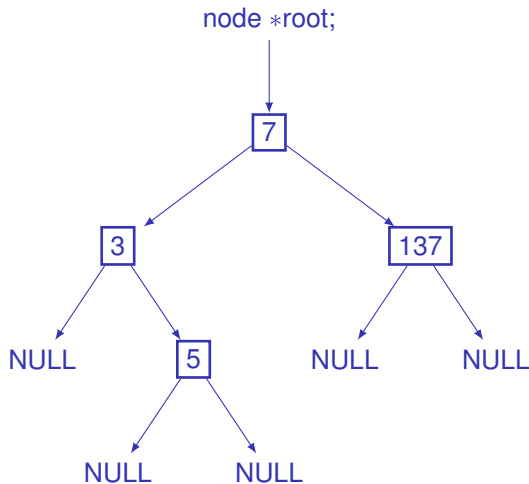
```
typedef struct node
{
    int content;
    struct node *left, *right;
} node;
```



7 Datenstrukturen

7.3 Bäume

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

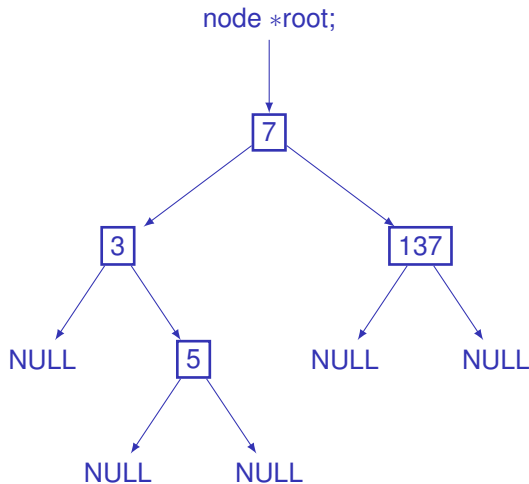


7 Datenstrukturen

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, $\mathcal{O}(n \log n)$

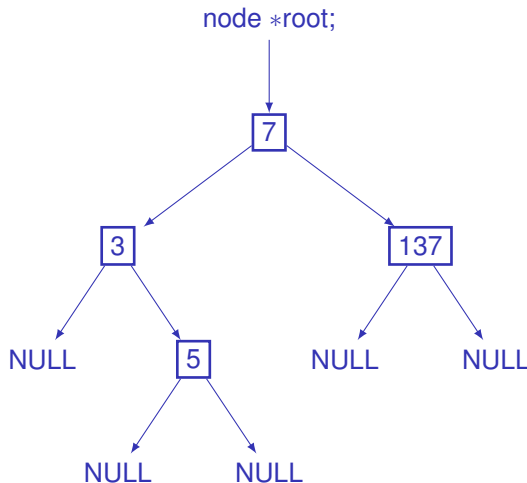


7 Datenstrukturen

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



Hardwarenahe Programmierung

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

- 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**