Algorithmen und Datenstrukturen (für ET/IT)

Wintersemester 2012/13

Dr. Tobias Lasser

Computer Aided Medical Procedures
Technische Universität München

Wiederholung: Ziele der Vorlesung

Wissen:

- Algorithmische Prinzipien verstehen und anwenden
  $\rightarrow$ Kapitel 1, 2, 4 – 6
- Grundlegende Algorithmen kennen lernen
  $\rightarrow$ Kapitel 4 – 10
- Grundlegende Datenstrukturen kennen lernen
  $\rightarrow$ Kapitel 3, 7
- Bewertung von Effizienz und Korrektheit lernen
  $\rightarrow$ Kapitel 4, 5

Methodenkompetenz:

- für Entwurf von effizienten und korrekten Algorithmen
  $\rightarrow$ Kapitel 4, 5, 6
- zur Umsetzung auf dem Computer
  $\rightarrow$ laufend

Wiederholung

- Beschreibung von Algorithmen und Datenstrukturen benötigt "Sprache"
- Churchsche These: alle gängigen "Sprachen" sind äquivalent!
- Beispiele von "Sprachen":
  - Pseudocode
  - Flussdiagramme
  - C
  - C++
  - Java, C#, Objective-C und vieles mehr!

  Manche Sprachen eignen sich besser für bestimmte Zwecke als andere.
- in AuD: C und C++

Programm heute

1. Einführung

2. Mathematische Grundlagen
   - Mengen
   - Abbildungen
   - Zahldarstellung
   - Boolesche Logik

3. Elementare Datenstrukturen
   - Zeichenfolgen
   - Felder

Definition Feld

Definition Feld

Ein Feld A ist eine Folge von n Datenelementen $(d_i)_{i=1,\ldots,n}$,

$$A = d_1, d_2, \ldots, d_n$$

mit $n \in \mathbb{N}_0$.

Die Datenelemente d_i sind beliebige Datentypen (z.B. primitive).

Feld als sequentielle Liste

Repräsentation von Feld A als sequentielle Liste (oder Array)

- feste Anzahl n von Datenelementen
- zusammenhängend gespeichert
- in linearer Reihenfolge mit Index
- Zugriff auf i-tes Element über Index i: A[i]

Deklaration:

in C: `int feld[n];`

in C++: `std::vector<int> feld(n);`

Operationen auf sequentiellen Listen

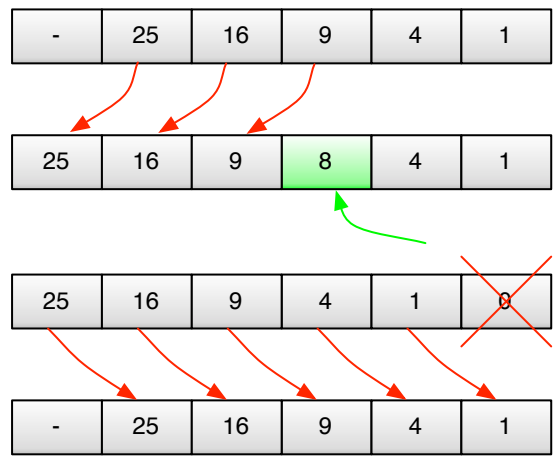
Sei A sequentielle Liste.

Operationen:

- `initialize:` Initialisiere seq. Liste A mit n Elementen
- `destruct:` vernichte seq. Liste A
- `elementAt(i):` greife auf i-tes Element von A zu: A[i]
- `insert:` füge Element in seq. Liste A ein

  (erfordert Umkopieren und evtl. Verlängern von A)
- `erase:` entferne Element aus seq. Liste A

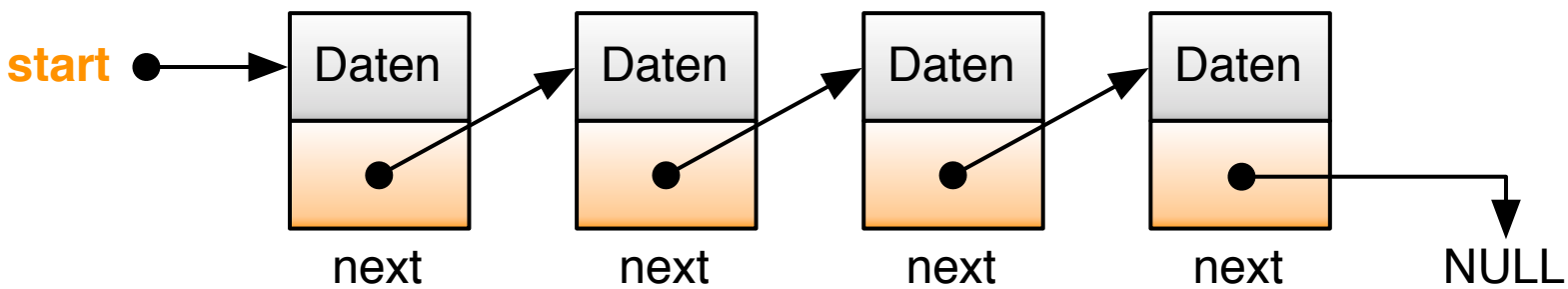
  (erfordert Umkopieren)



Feld als einfach verkettete Liste

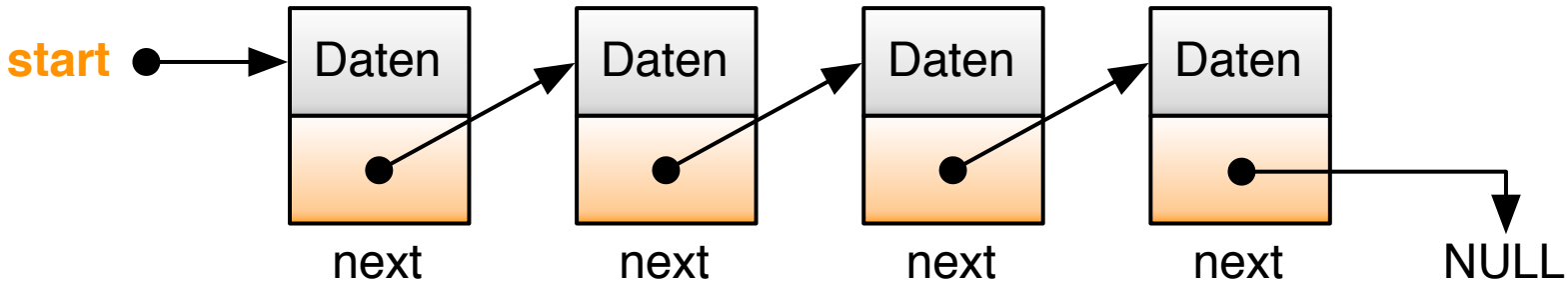
Repräsentation von Feld A als verkettete Liste

- dynamische Anzahl n von Datenelementen
- in linearer Reihenfolge gespeichert (nicht notwendigerweise zusammenhängend!)
- mit Zeigern verkettet



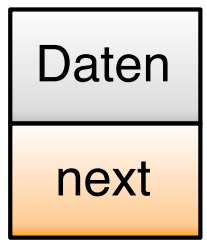
auf Englisch: *linked list*

Verkettete Liste



- Folge von miteinander verbundenen Elementen
- jedes Element d_i besteht aus
  - Daten: Wert des Feldes an Position i
  - next: Zeiger auf das nächste Element d_{i+1}

Node:



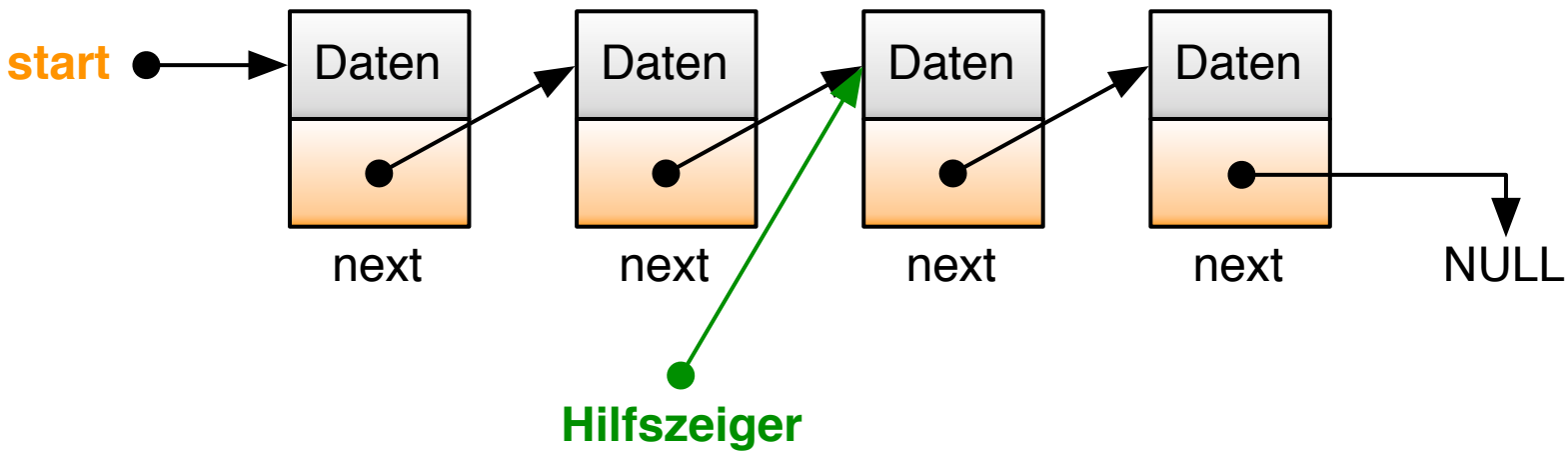
- start ist Zeiger auf erstes Element des Feldes d_1
- letztes Element d_n hat keinen Nachfolger
  - symbolisiert durch NULL-Zeiger

Operationen auf verketteter Liste

Zugriff auf Element i:

- beginne bei start Zeiger
- "vorhangeln" entlang next Zeigern bis zum i-ten Element

Beispiel für i=3:

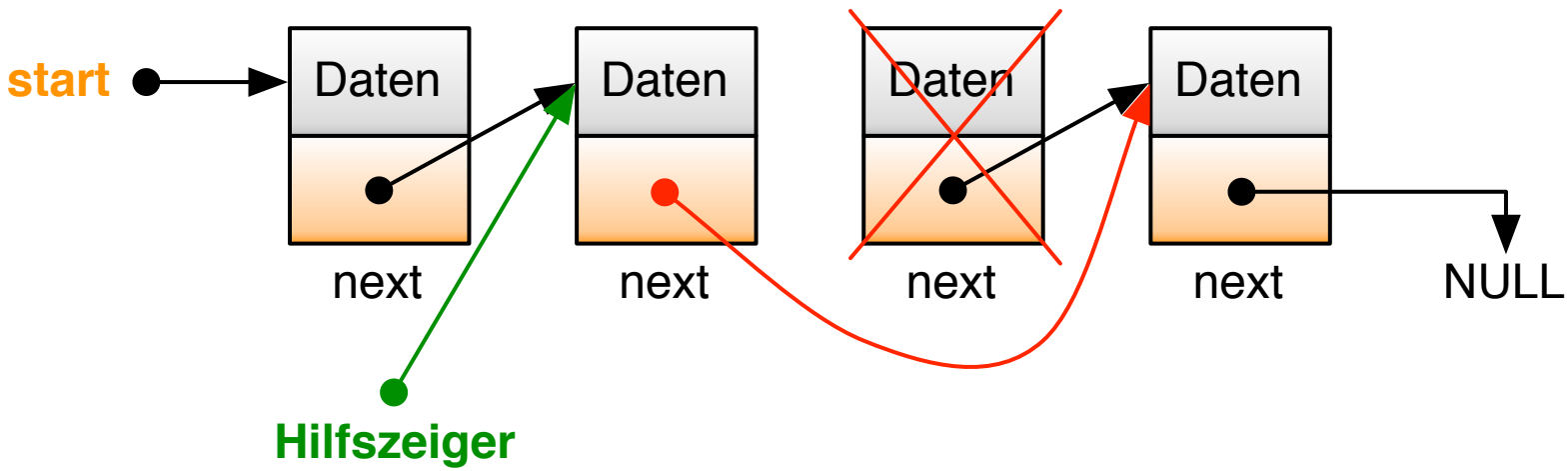


Operationen auf verketteter Liste

Löschen von Element i:

- Zugriff auf Element i-1
- "umhängen" von next Zeiger von Element i-1 auf Element i+1

Beispiel für i=3:

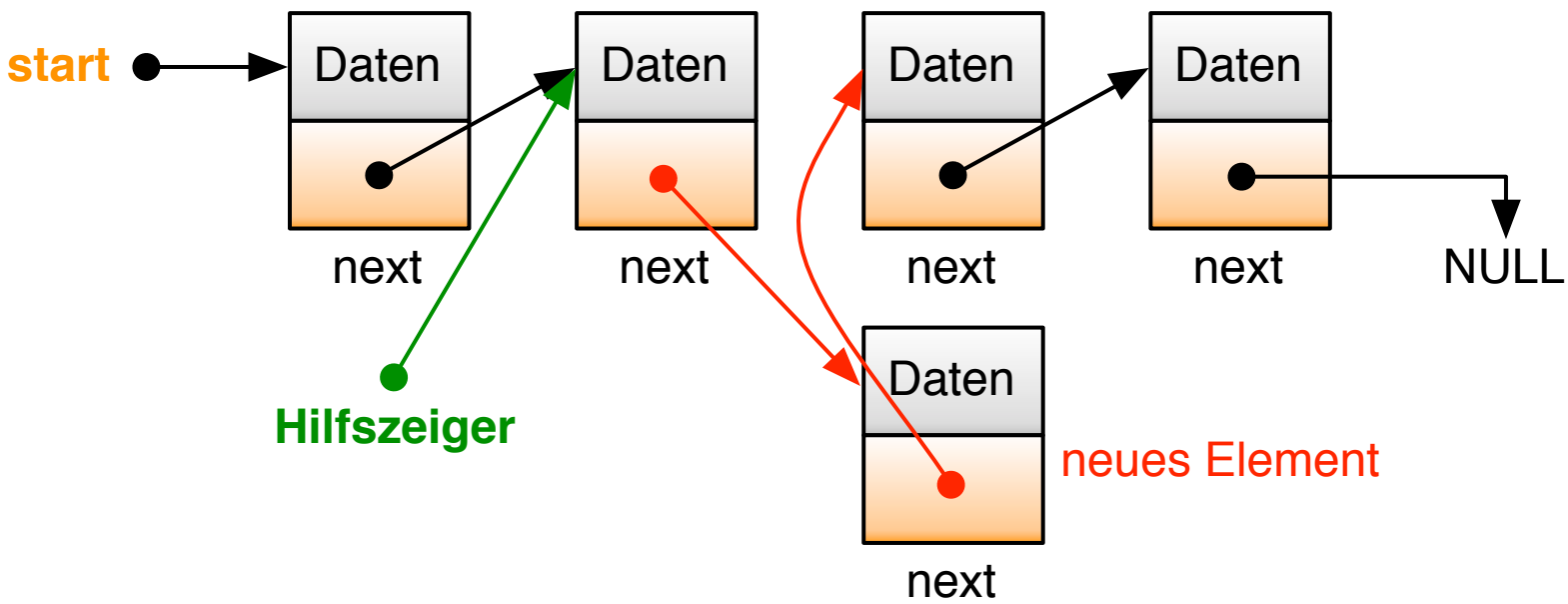


Operationen auf verketteter Liste

Einfügen von Element an Stelle i:

- Zugriff auf Element i-1
- "umhängen" von next Zeiger von Element i-1 auf neues Element
- next Zeiger von neuem Element setzen auf altes Element i

Beispiel für i=3:



Verkettete Liste in C (Auszug)

```c
// Datenstruktur fuer verkettete Liste
struct node {
    struct node* next;
    int Daten;
};


// Funktion zum Anlegen eines Nodes
struct node* initNode(int daten);
// Funktion zum Freigeben einer Liste
void destructList(struct node* anfang);

// Funktion zur Listenausgabe
void printList(struct node* anfang);
```

Verkettete Liste in C (Auszug)

```c
struct node* start = NULL; // Startknoten der Liste

struct node* hilfsZeiger;  // Hilfsvariablen
int i;

// Liste fuellen
start = initNode(0);
start->next = NULL; // Listenende markieren

hilfsZeiger = start;
for (i = 1; i <= 5; ++i) {
    hilfsZeiger->next = initNode(i*i);
    hilfsZeiger = hilfsZeiger->next;
    hilfsZeiger->next = NULL; // neues Ende der Liste
}

printList(start); // Liste durchlaufen und ausgeben

destructList(start);  // Liste freigeben
start = NULL; // und wirklich als leer markieren
```

Verkettete Liste in C (Auszug)

```c
// Funktion zum Anlegen eines Nodes
struct node* initNode(int daten)
{
    struct node* ptr;
    // neuen Knoten anlegen
    ptr = (struct node*) malloc( sizeof(struct node) );

    if (ptr == NULL) { // falls malloc fehlgeschlagen
        // !! ACHTUNG !!
        // hier braeuchten wir Fehlerbehandlung!
        // machen wir hier der Einfachheit halber nicht
    }

    // Daten zuweisen
    ptr->Daten = daten;

    return ptr;
}
```

Verkettete Liste in C (Auszug)

```c
// Funktion zum Freigeben einer Liste
void destructList(struct node* anfang)
{
    struct node* ptr; // Hilfsvariablen
    struct node* temp;

    // Liste durchlaufen und freigeben
    ptr = anfang;
    while ( ptr != NULL) {
        temp = ptr->next; // Nachfolger in temp speichern
        free(ptr);        // aktuellen Knoten freigeben
        ptr = temp;       // zum naechsten Knoten
    }
}
```

Verkettete Liste in C (Auszug)

```c
// Funktion zur Listenausgabe
void printList(struct node* anfang)
{
    struct node* ptr; // Hilfsvariable

    printf("Listen-Inhalt:␣");

    // Liste durchlaufen von anfang
    ptr = anfang;
    while (ptr != NULL) {
        printf("%d␣", ptr->Daten); // Wert ausgeben
        ptr = ptr->next; // zum naechsten Knoten
    }
    printf("\n");
}
```

Verkettete Liste in C++ (Auszug)

```
std::list<int> feld; // Liste initialisieren

// Iterator it als Hilfs"zeiger" auf Listenelemente
std::list<int>::iterator it;

// Liste fuellen
for (int i = 5; i >= 0; --i)
    feld.push_front(i*i);
    // push_front fuegt Element am Anfang ein
    // alternativ: feld.insert(feld.begin(), i*i);

// Liste durchlaufen und ausgeben
std::cout << "Listen-Inhalt:␣";
for (it = feld.begin(); it != feld.end(); ++it)
    std::cout << *it << "␣";
    // *it gibt das Datenelement von Listeneintrag
    // an Stelle it aus
std::cout << std::endl;
```

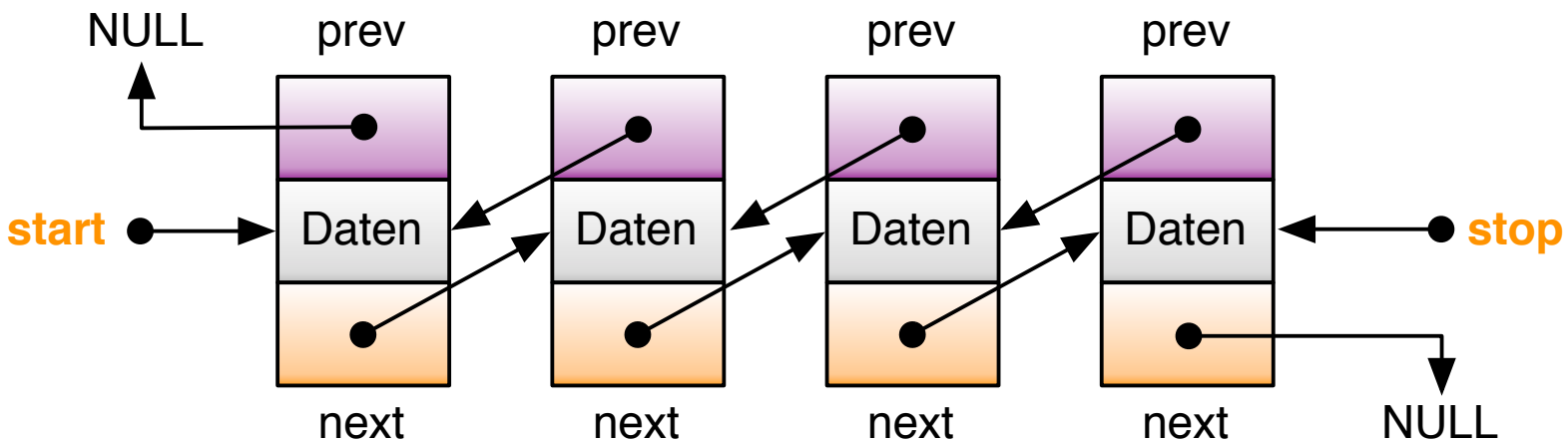
Gegenüberstellung sequentielle Liste und verkettete Liste

Sequentielle Liste	Verkettete Liste
⊕ Direkter Zugriff auf i-tes Element	⊖ Zugriff auf i-tes Element erfordert i Iterationen
⊕ sequentielles Durchlaufen sehr einfach	⊕ sequentielles Durchlaufen sehr einfach
⊖ statische Länge, kann Speicher verschwenden	⊕ dynamische Länge
	⊖ zusätzlicher Speicher für Zeiger benötigt
⊖ Einfügen/Löschen erfordert erheblich Kopieraufwand	⊕ Einfügen/Löschen einfach

Feld als doppelt verkettete Liste

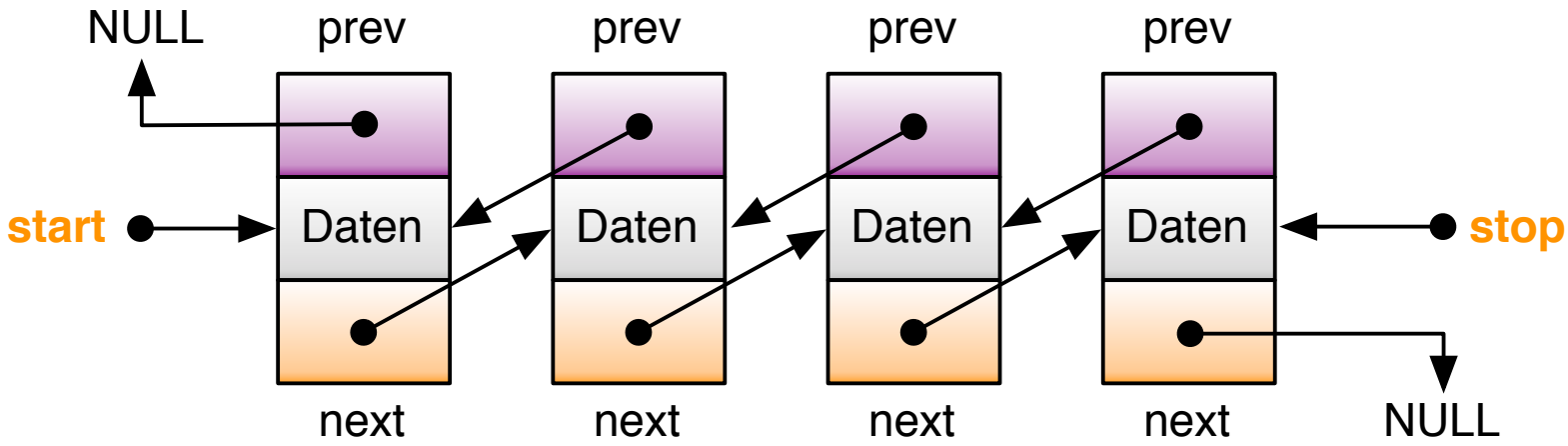
Repräsentation von Feld A als doppelt verkettete Liste

- verkettete Liste
- jedes Element mit Zeigern doppelt verkettet



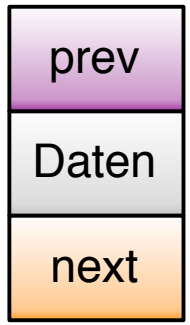
auf Englisch: *doubly linked list*

Doppelt verkettete Liste



- Folge von miteinander verbundenen Elementen
- jedes Element d_i besteht aus
  - Daten: Wert des Feldes an Position i
  - next: Zeiger auf das nächste Element d_{i+1}
  - prev: Zeiger auf das vorherige Element d_{i-1}

Node:



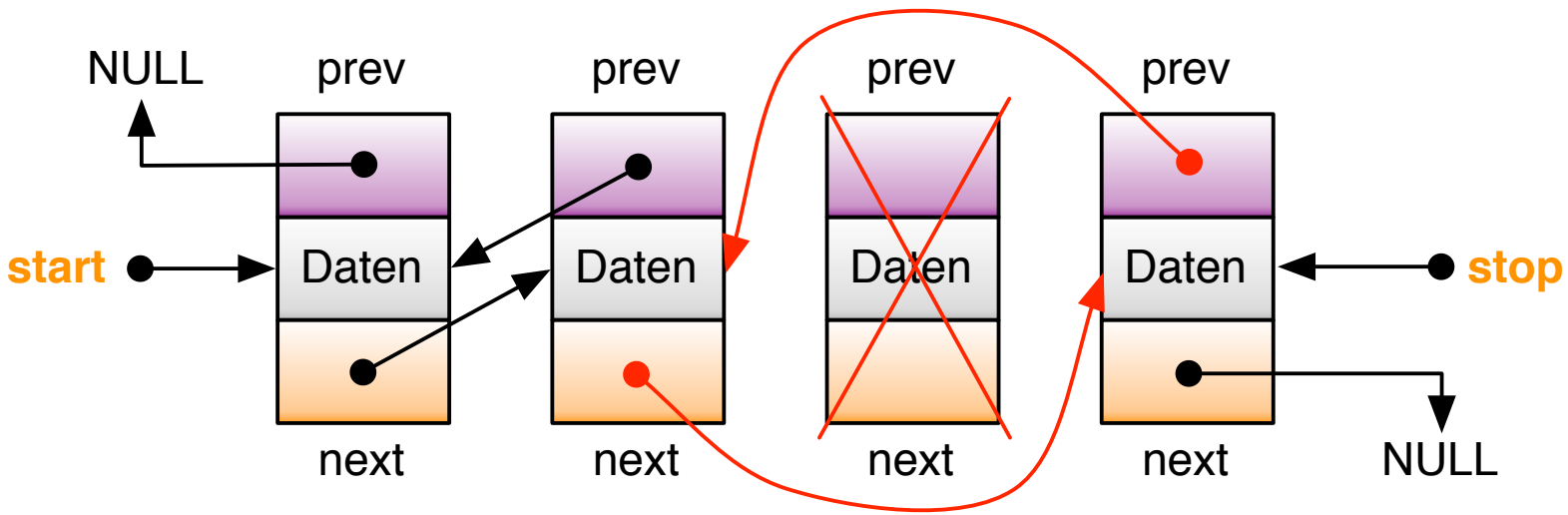
- start/stop sind Zeiger auf erstes/letztes Element des Feldes

Operationen auf doppelt verketteter Liste

Löschen von Element i:

- Zugriff auf Element i
- "umhängen" von next Zeiger von Element i-1 auf Element i+1
- "umhängen" von prev Zeiger von Element i+1 auf Element i-1

Beispiel für i=3:

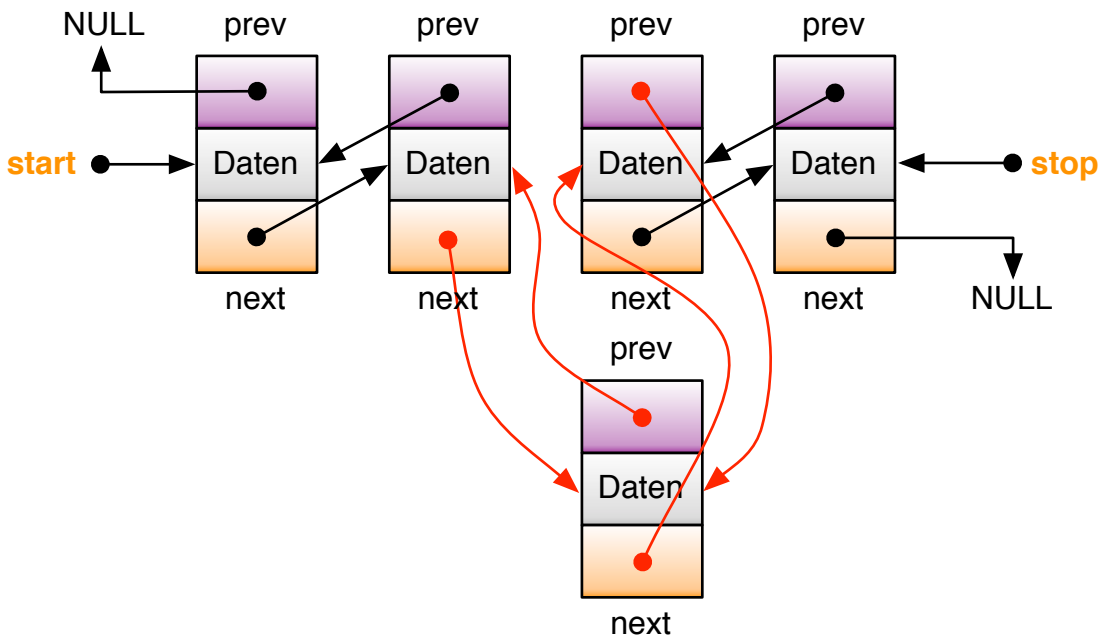


Operationen auf doppelt verketteter Liste

Einfügen von Element an Stelle i:

- Zugriff auf Element i
- "umhängen" von next Zeiger von Element i-1 auf neues Element, sowie "umhängen" von prev Zeiger von altem Element i auf neues Element
- next bzw. prev Zeiger von neuem Element setzen auf altes Element i bzw. Element i-1

Beispiel für i=3:



Eigenschaften doppelt verkettete Liste

Feld A als doppelt verkettete Liste

- Vorteile:
  - Durchlauf in beiden Richtungen möglich
  - Einfügen/Löschen potentiell einfacher, da man sich Vorgänger nicht extra merken muss
- Nachteile:
  - zusätzlicher Speicher erforderlich für zwei Zeiger
  - Zeigerverwaltung komplizierter und fehleranfällig

Doppelt verkettete Liste in C und C++

- in C:

```
struct node {
    int Daten;
    struct node* prev;
    struct node* next;
};

struct node* start;
struct node* stop;
```

- in C++:

```
std::list<int> feld;
```

Zusammenfassung Felder

Ein Feld A kann repräsentiert werden als:

- sequentielle Liste (array)
  - mit fixer Länge
- verkettete Liste (linked list)
  - mit dynamischer Länge
- doppelt verkettete Liste (doubly linked list)
  - mit dynamischer Länge

Eigenschaften:

- einfach und flexibel
- aber manche Operationen aufwendig

Als nächstes $\longrightarrow$ Aufgabe von Flexibilität für Effizienz

Zusammenfassung

1. Einführung

2. Mathematische Grundlagen
   - Mengen
   - Abbildungen
   - Zahldarstellung
   - Boolesche Logik

3. Elementare Datenstrukturen
   - Zeichenfolgen
   - Felder