*Lehrstuhl für Informatik 3 – Prof. Dr.-Ing. M. Nagl*
*RWTH Aachen*

# Grundgebiete der Informatik 2
# Lösungsvorschlag zur Probeklausur

*Prof. Dr.-Ing. M. Nagl*
*Simon Becker*
*Ulrike Ranger*
*René Wörzberger*

# Aufgabe 1 a) (1)

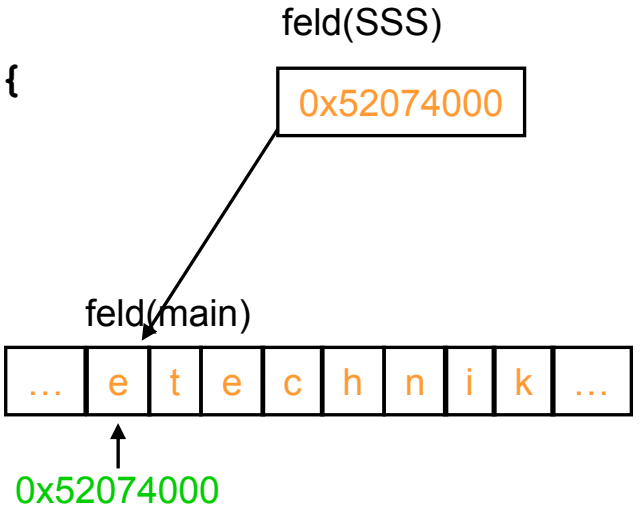
- Beschreiben Sie den Unterschied zwischen *Call-by-Value* und *Call-by-Reference*.
  - CbV: Wert wird kopiert, Änderungen nicht beim Aufrufer sichtbar.
  - CbR: Anderer Name für übergebene Variable, Änderungen beim Aufrufer sichtbar.

# Aufgabe 1 a) (2)

```
void SortStraightSelection(FeldTyp *feld, int start, int ende) {
    char elem;
    int k;

    for (int i = start; i <= ende - 1; i++) {
        k = i;
        elem = (*feld)[i];
        for (int j = i + 1; j <= ende; j++) {
            if ((*feld)[j] == elem) {
                ...
            }
        }
        ...
    }
    start++;
}

int main()  {
    FeldTyp feld = { 'e', 't', 'e', 'c', 'h', 'n', 'i', 'k' };
    int start = 0, schluss = 7;
    SortStraightSelection(&feld, start, schluss);
    ...
}
```

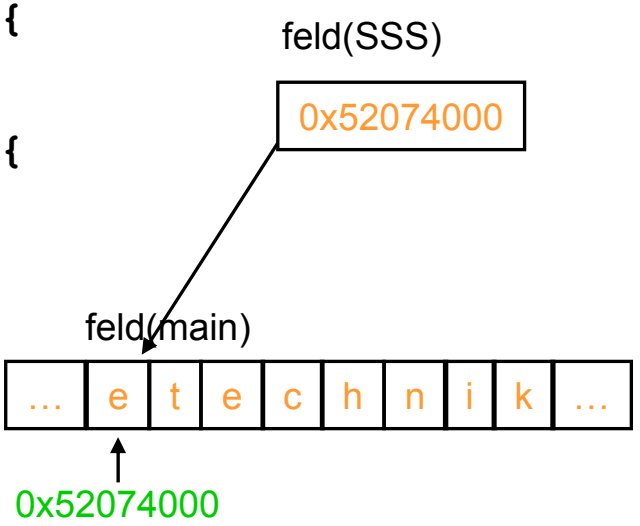


# Aufgabe 1 a) (3)

```
void SortStraightSelection(FeldTyp &feld, int start, int ende) {
    char elem;
    int k;

    for (int i = start; i <= ende - 1; i++) {
        k = i;
        elem = feld[i];
        for (int j = i + 1; j <= ende; j++) {
            if (feld[j] == elem) {
                ...
            }
        }
        ...
    }
}

int main()  {
    FeldTyp feld = { 'e', 't', 'e', 'c', 'h', 'n', 'i', 'k' };
    int start = 0, schluss = 7;
    SortStraightSelection(feld, start, schluss);
    ...
}
```



# Aufgabe 1 b)

- Worin besteht der wesentliche Unterschied zwischen einem Feld und einem Verbund?
  - Feld ist Zusammenfassung von Werten des gleichen Typs.

```
typedef char FeldTyp[100];
FeldTyp feld ={ 'e', 't', 'e', 'c', 'h', 'n',
'i', 'k' };
```

  - Verbund ist Zusammenfassung unterschiedlicher Komponenten.

```
struct Person{
    string Name;
    int Alter;
    string Adresse;
}
```

# Aufgabe 1 c)

- Wie ist die Lebensdauer einer lokalen Variablen definiert?
  - Nur innerhalb des Blocks, in dem die Deklaration steht.

```
void SortStraightSelection(FeldTyp *feld, int start, int ende) {
    char elem;

    for (int i = start; i <= ende - 1; i++) {
        int k = i;
        elem = (*feld)[i];
        for (int j = i + 1; j <= ende; j++){ … }
    }
}

int main()  {
    FeldTyp feld = { 'e', 't', 'e', 'c', 'h', 'n', 'i', 'k' };
    int start = 0, schluss = 7;
    SortStraightSelection(&feld, start, schluss);
    …
}
```

# Aufgabe 1 d) (1)

- Im folgenden soll die EBNF für die einfache Programmiersprache EASY angegeben werden.
- Beispiel:

```
PROGRAM beispiel (
        summe = 16.06 + 2006 ;
        produkt = 5 * 9.2006 ;
)
```

- Die Nichtterminalsymbole **`digit`** und **`letter`** sind gegeben.

# Aufgabe 1 d) (2)

Jedes EASY-Programm (**`program`**) wird mit dem Schlüsselwort **`PROGRAM`** eingeleitet.

Anschließend folgt ein Bezeichner (**`identifier`**) als Programmname.

Nach dem Bezeichner können beliebig viele Anweisungen (**`assign`**) vorkommen.

Alle Anweisungen sind innerhalb eines Paares runder Klammern eingeschlossen.

```
program ::= „PROGRAM“ identifier „(“
            {assign} „)“ ;
```

# Aufgabe 1 d) (3)

Jedes **`assign`** besteht aus einem **`identifier`** gefolgt von einem Gleichheitszeichen (=).

Nach dem Gleichheitszeichen folgt die eigentliche Berechnung, die sich aus zwei Zahlen (**`number`**) und einem dazwischenliegenden Operator (**`operator`**) zusammensetzt.

Jedes **`assign`** wird von einem Semikolon (`;`) abgeschlossen.

```
assign ::= identifier „=“ number operator
           number „;“ ;
```

# Aufgabe 1 d) (4)

Da es sich hier um eine einfache Programmiersprache handelt, sind als **`operator`** nur die Grundrechenarten **`+`**, **`-`**, **`*`** und **`/`** zugelassen.

```
operator ::= „+“ | „-“ | „*“ | „/“ ;
```

# Aufgabe 1 d) (5)

Jede **`number`** ist positiv und besteht aus mindestens einem **`digit`**.

Anschließend kann optional ein Punkt ( **`.`** ) folgen, hinter dem wieder mindestens ein **`digit`** steht.

```
number ::= {digit}+ [ „.“ {digit}+ ] ;
```

oder

```
number ::= digit {digit} [ „.“
           digit {digit} ] ;
```

# Aufgabe 1 d) (6)

Ein Bezeichner (**`identifier`**) ist eine Folge von **`letter`** und **`digit`**, wobei jeder Bezeichner mit einem Buchstaben anfangen muss.

```
identifier ::= letter
                  { ( letter | digit )} ;
```

# Aufgabe 1: Anmerkungen

- Wissensfragen
  - Frage genau lesen
  - Auf die Begriffe der Frage eingehen

- EBNF-Aufgabe
  - Terminalsymbole markieren (alle!)
  - EBNF-Syntax (nicht Syntaxdiagramm)
  - { } vs. { }$^+$
  - Text genau lesen

# Aufgabe 2 a)

```
void SortStraightSelection(FeldTyp &feld; int start, int ende) {
    char elem;
    int k;

    for (int i = start; i <= ende - 1; i++) {
        k = i;
        elem = feld[i];
        for (int j = i + 1; j <= ende; j++) {
            if (feld[j] == elem) {
                k = j;
                elem = feld[j];
            }
        }
        feld[k] = feld[i];
        feld[i] = elem;
    }
    return 0;
}
int main()  {
    FeldTyp feld ={ 'e', 't', 'e', 'c', 'h', 'n', 'i', 'k' };
    SortStraightSelection(feld, 0, 7);
    for (int i=0;i< 8;i++) {
        std::cout << feld[i];
    }
    return 0;}
```

# Aufgabe 2 b)

```
void f(int *b, int &c) {
    int a = 2;
    *b += 3;
    c = (*b)*a;
}


int main() {
    int y = 2, z = 4;

    f(&y, z);
    f(&y, z);

    return 0;
}
```

| y | z | |
|---|---|---|
| 5 | 10 | Nach 1. Aufruf |
| 8 | 16 | Nach 2. Aufruf |

# Aufgabe 2: Anmerkungen

- Syntaxfehler suchen
  - Keine Semantik Fehler suchen („Das Feld wird nicht ganz durchlaufen.“, oder „Der Algorithmus sortiert falsch.“)
  - Kein *falscher* Vergleichsoperator in **`std::cout <<`** …
  - Derselbe Buchstabe kann öfter in einem **`char`**-Feld auftauchen
  - Nicht nötig: **`using namespace std;`**

- Laufzeitsemantik
  - Auf Papier durchspielen

# Aufgabe 3a

a) Deklaration des Verbundtyps Schueler bestehend aus zwei Komponenten

| 1. Variante | 2. Variante |
|---|---|
| `struct Schueler {`<br>`  string name;`<br>`  int note;`<br>`};` | `typedef struct {`<br>`  string name;`<br>`  int note;`<br>`} Schueler;` |

beliebte Fehler:

- Abschließendes Semikolon vergessen
- Verwechselung **Verbund**- mit **Feld**typ
- Falsche Typen für die Komponenten (Text lesen!)

# Aufgabe 3b und 3c

b) **Konstante ganze** Zahl deklarieren und 10 zuweisen

```
const int KLASSENGROESSE = 10;
```
oder
```
int const KLASSENGROESSE = 10;
```

c) Feld**typ** **`KlausurErg_FT`** definieren. Elemente sind **`Schueler`**. Größe ist **`KLASSENGROESSE+1`**.

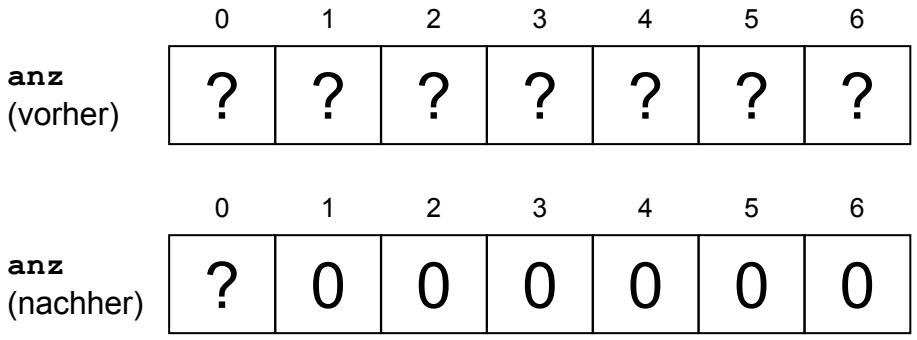
```
typedef Schueler
KlausurErg_FT[KLASSENGROESSE+1];
```

Beliebter Fehler: Datenobjekt statt Typ deklariert.

# Aufgabe 3d (1. Schritt)

- Deklaration eines Ganzzahlfeldes **`anz`** der Größe 7 und Initialisierung mit 0 für jedes Element (aus **`anz[0]`**)

```
int anz[7];
for (int j = 1; j <= 6; j++) {
     anz[j] = 0;
}
```




Beliebte Fehler:

- Felddeklaration vergessen
- Deklaration der Laufvariablen vergessen
- „Off-by-one-Fehler“

# Aufgabe 3d (2. Schritt)

- Berechnung der Notenhäufigkeit in `feld`.

```
for (int i = 1; i <= feldgroesse; i++) {
     int j = feld[i].note;
     anz[j]++;
}
```

**feld**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ? | Meier 2 | Schm. 1 | Beck. 2 | Xu 2 | Zimm. 4 | Heller 5 | Haase 5 | Rang. 5 | Fuß 6 | Held 2 |

**anz** (nachher)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| ? | | | | | | |

Beliebte Fehler:

- **`.note`** vergessen.
  Evaluiert dann zum Verbund statt zur Note

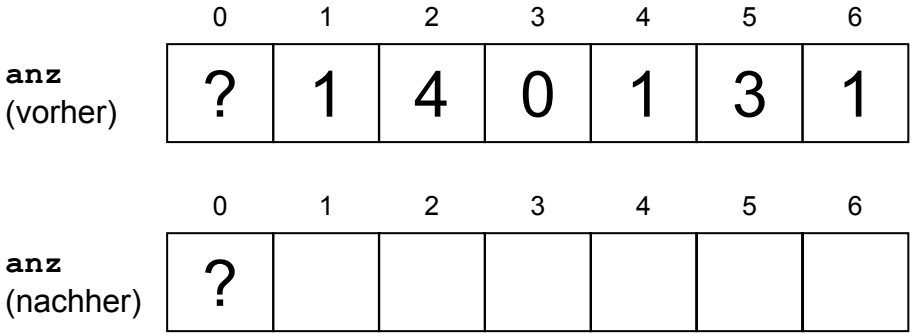
- Umständliche Feldindizierung

```
for (int k = 1; k <= 6; k++) {
  if (k == feld[i].note) {
  anz[k]++;
}
```

# Aufgabe 3d (3. Schritt)

- Anzahlen aufsummieren

```
for (int j = 1; j <= 5; j++) {
     anz[j+1] = anz[j] + anz[j+1];
}
```




Hinweis:

```
anz[j+1] += anz[j];
```

geht auch

# Aufgabe 3d (4. Schritt)

- Personen von **`feld`** mittels **`anz`** an richtige Stelle in **`sortFeld`** kopieren

```
for (int i = 1; i <= feldgroesse; i++) {
     int k = feld[i].note;
     sortFeld[anz[k]] = feld[i];
     anz[k]--;
}
```

**`feld`**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ? | Meier 2 | Schm. 1 | Beck. 2 | Xu 2 | Zimm. 4 | Heller 5 | Haase 5 | Rang. 5 | Fuß 6 | Held 2 |

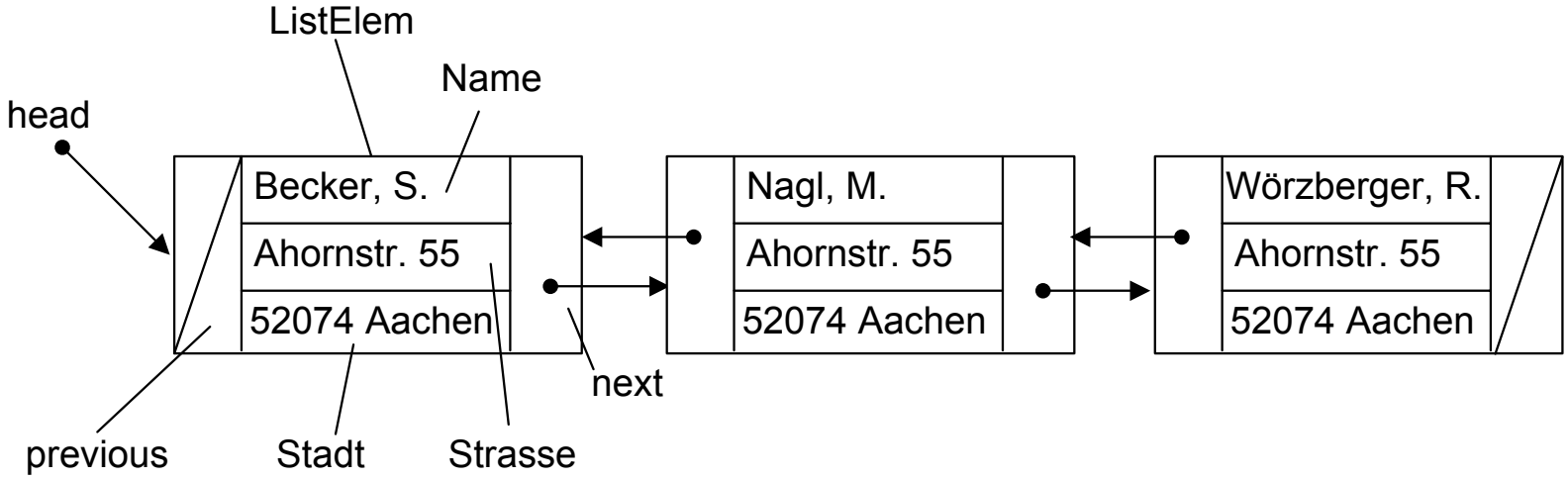
**`anz`** (nachher)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| ? | 1 | 5 | 5 | 6 | 9 | 10 |

**`feld`**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| ? | | | | | | | | | | |

# Aufgabe 4 a)




```
struct ListElem {
   string Name;
   string Strasse;
   string Stadt;
   ListElem *next;
   ListElem *previous;
}
ListElem *head;
```

# Aufgabe 4 b)

**`Addrs_insert`**:

neues Element an richtiger Stelle einfügen

- Spezialfälle:
  - Am Anfang einfügen (`findElementBefore` gibt `NULL` zurück)
  - Kein Nachfolger (Also Vorsicht beim Setzen des `previous`-Zeigers des Nachfolgers)

- Aufgaben:
  - neues Element erstellen und initialisieren (vorgegeben)
  - Einfügestelle bestimmen (Aufruf von `findElementBefore` ist vorgegeben)
  - Einhängen (vorne oder hinter passendem Element)
  - Dabei doppelte Verzeigerung beachten

# Aufgabe 4 b) (1)

```
void Addrs_insert(string name, …) {

    ListElem *newElem = new ListElem;
    newElem->Name=name;
    //…

    ListElem *before=findElementBefore(name);
    Vorgänger gefunden?
    if (before!=NULL) {
        //…

    } else {
       //…

    }
```

Element erzeugen und initialisieren

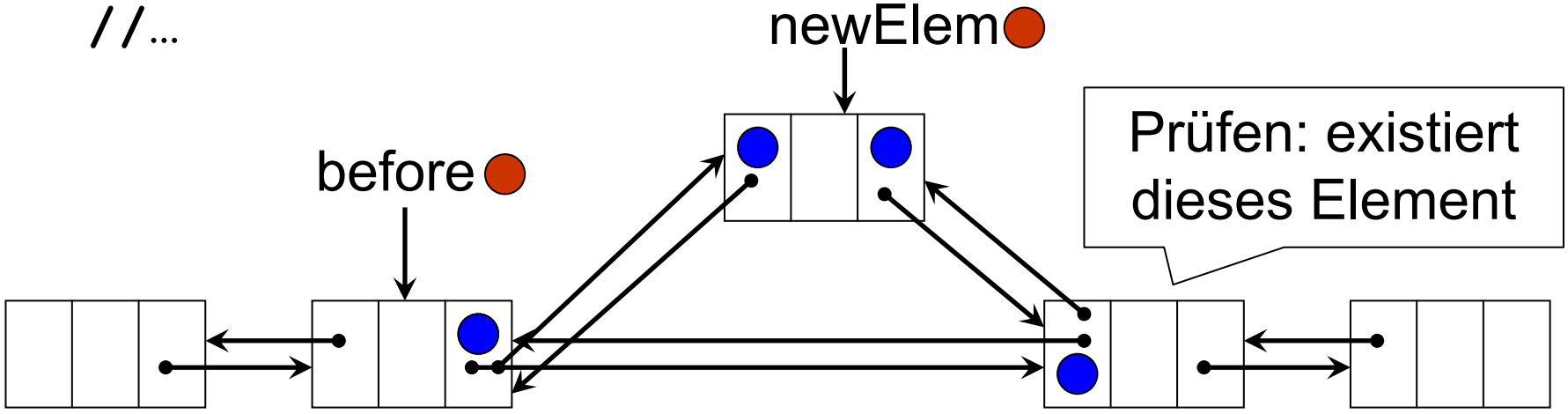
Einfüge-stelle

hinter Vorgänger einfügen und Rest verzeigern

Am Anfang einfügen, behandelt auch Rest der Liste

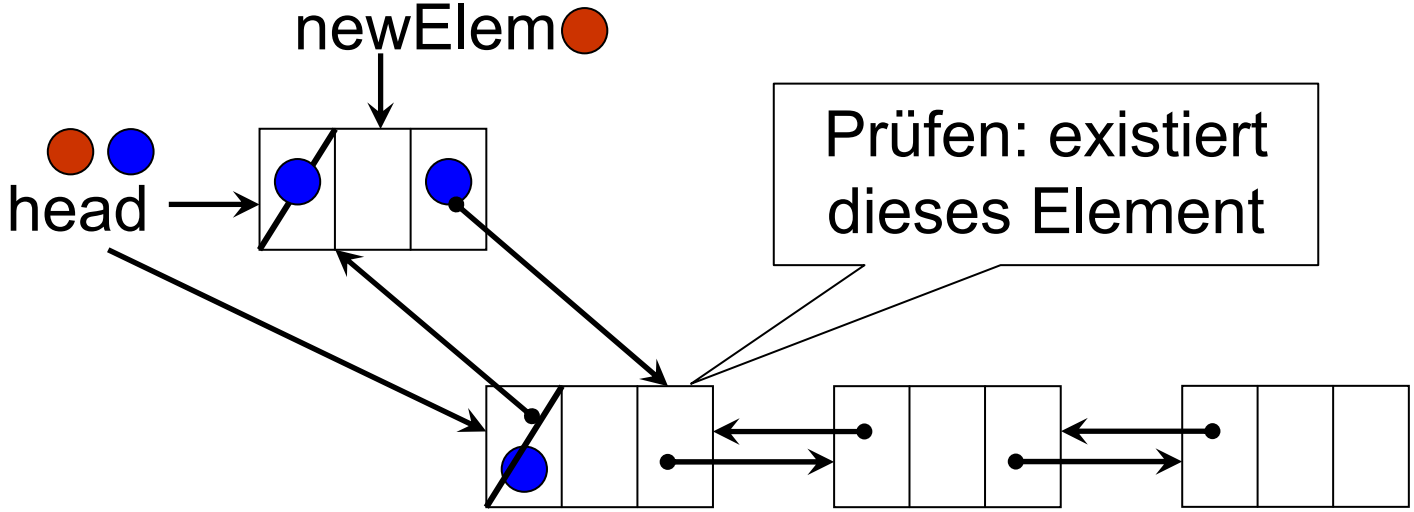
# Aufgabe 4 b) (2)

```
if (before!=NULL) {
    newElem->next=before->next
    if (newElem->next!=NULL)
        newElem->next->previous=newElem;
    before->next=newElem;
    newElem->previous=before;
} else {
        //…
}
```



# Aufgabe 4 b) (2)

```
  } else {
      // before==NULL, Einfügen am Anfang
      newElem->next=head;
      if (head!=NULL)
         head->previous=newElem;
      head=newElem;
      newElem->previous=NULL;
  }
```



# Aufgabe 4 c)

**`findElementBefore`**:

Finde Element mit größtem Namen, der kleiner ist als der übergebene.

- Spezialfälle:
  - Liste ist Leer.
  - es gibt kein Element mit einem kleineren Namen.
  - Element muss hinten angehängt werden.
- Aufgaben:
  - Liste leer?
  - Erstes Element größer als übergebener Name?
  - Durch Liste laufen, bis nächstes Element größer.
- verschiedene Möglichkeiten durch doppelte Verzeigerung der Liste

# Aufgabe 4 c)

```
ListElem *findElementBefore(string name) {

   ListElem *current=head;

   if (head==NULL || head->name>name) return NULL;

   while (current->next!=NULL &&
          current->next->name<name) {
      current=current->next;
   }

   return current;
}
```

Liste leer oder 1. El. größer

Laufe bis zum letzten El. oder bis **nächster** Name größer. Reihenfolge!

# Aufgabe 4 c) Alternativen

- Weitere Möglichkeiten:
  1. Liste von vorne bis hinten bzw. zum ersten größeren Element durchlaufen, dabei in zweitem Laufzeiger immer das vorherige Element merken.
  2. Liste von vorne bis hinten bzw. zum ersten größeren Element durchlaufen und mit ->previous den Vorgänger ermitteln. Vorsicht: wenn Laufzeiger gleich NULL, dann ->previous nicht gültig, daher Sonderfall für Listenanfang

# Aufgabe 4 c) Alternative 1

```
ListElem *previous=NULL;
ListElem *current=head;

while (current!=NULL && current->Name<name)
{
     previous=current;
     current=current->next;
}

return previous;
```

# Aufgabe 4 c) Alternative 2

```
ListElem *current=head;
while (current!=NULL) {
   if (current->Name<name) {
      if (current->next!=NULL) {
         current=current->next;
      } else {
         return current;
      }
   } else {
      return current->previous;
   }
}

return NULL;
```

# Aufgabe 4: Anmerkungen

- Nicht alle Teile einer Datenstruktur in einem `struct`
- head nicht vergessen!
- Beim Laufen durch Datenstrukturen nicht den Anker verändern.
- Aufgabenstellung lesen:
  - `findElementBefore` verwenden,
  - Kopfzeiger (`head`) deklarieren
  - kein Array
  - hier waren nur 3 strings gefragt, keine weiteren Daten
- Spezialfälle
  - Anker umsetzen
  - vor Zugriff auf Komponente prüfen, ob Zeiger auf Objekt != NULL