

ALGORITHMEN UND DATENSTRUKTUREN

ÜBUNG 6: ARRAYS & LISTEN

Eric Kunze

`eric.kunze@mailbox.tu-dresden.de`

TU Dresden, 04.12.2020

Arrays

Deklaration (als EBNF): `ElementType Ident { [Number] } ;`

- ▶ `ElementType ...` Typ eines Eintrags
- ▶ `Number ...` Anzahl der Einträge
- ▶ `Ident ...` Bezeichner des Arrays

Beispiele:

- ▶ `Liste: int liste[5];`
- ▶ `Matrix: int matrix[3][4];`

Initialisierungen:

- ▶ `int liste[5] = {2,7,0,-4,1};`
- ▶ `int matrix[3][4] = { {1,2,3,4}, {5,6,7,8}, {2,3,4,5} };`

Zuweisungen: *Indizierung beginnt bei 0*

- ▶ `liste[2] = 16; \rightsquigarrow [2,7,16,-4,1]`
- ▶ `matrix[1][3] = 0; \rightsquigarrow $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 0 \\ 2 & 3 & 4 & 5 \end{pmatrix}$`

AUFGABE 1

Ist die Zeichenkette ein Palindrom, so soll der aktuelle Parameter `korrekt` im Aufruf von `palindrom1` den Wert 1 annehmen, sonst 0.

falscher Code:

```
1    palindrom1(char feld[], int l, int korrekt) {  
2        int i = 1;  
3        l = l - 1;  
4        while (i < l && korrekt) {  
5            korrekt = feld[i] == feld[l];  
6            i = i + 1;  
7        }  
8        return korrekt;  
9    }
```

AUFGABE 1

richtiger Code:

```
1 void palindrom1(char feld[], int l, int *korrekt)
2 {
3     int i = 0;    // muss mit 0 statt 1
4                   // initialisiert werden
5     l = l - 1;
6     *korrekt = 1; // muss mit 1 initialisiert werden
7     while (i < l && *korrekt) {
8         *korrekt = feld[i] == feld[l];
9         i = i + 1;
10        l = l - 1; // l muss dekrementiert werden
11    }
12    // kein return, weil rueckgabetyt void
13 }
```

Verkettete Listen

VERKETTETE LISTEN

Wir betrachten *verkettete* Listen.

- Listenelemente
- Verkettungen



VERKETTETE LISTEN

```
1 typedef struct element *list;  
2 struct element {int value; list next};
```

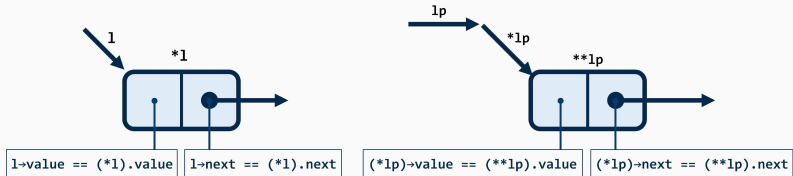
- ▶ `struct element` definiert einen neuen Datentypen
- ▶ `int value` ist der Wert des Listenelements
- ▶ `typedef` definiert nur einen Alias
 - ▷ Wir dürfen jeden Pointer auf ein `struct element` auch einfach `list` nennen.
- ▶ `list next` ist ein Element vom Typ `list`
 - ▷ ein Pointer auf ein `struct element` (das nächste Listenelement)

```
struct element { int value; list next }
```

```
typedef struct element *list
```



DIE OPERATOREN $\&$, $*$, \rightarrow , $.$

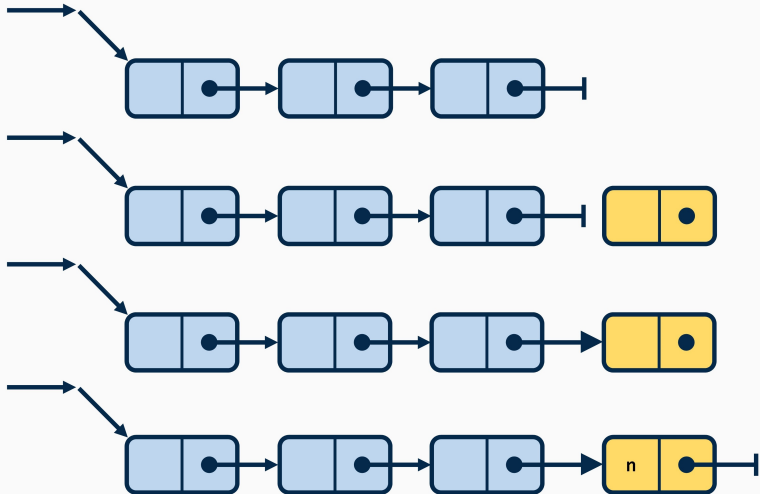


Die Operatoren $\&$ und $*$ binden schwächer als $.$ und \rightarrow .

- ▶ $l \rightarrow \text{value} == (*l). \text{value}$
- ▶ $\&l \rightarrow \text{next} == \&((*l). \text{next})$

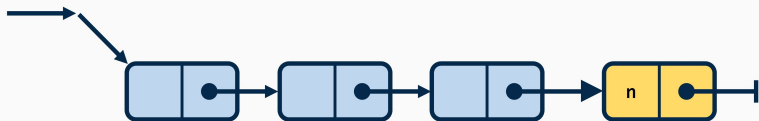
AUFGABE 2 — TEIL (A)

Anfügen an eine Liste



AUFGABE 2 — TEIL (A)

Anfügen an eine Liste



- ▶ Gehe zum Ende der Liste
- ▶ Allokieren neuen Speicherplatz und verknüpfe mit neues Element mit bisheriger Liste
 - ▷ Nachfolger-Pointer des bisherigen letzten Listenelements auf auf neues Listenelement zeigen lassen
- ▶ Fülle das Listenelement mit Schlüsselwert und Nachfolger (= NULL)

Anfügen an eine Liste

- ▶ Gehe zum Ende der Liste
- ▶ Allokieren neuen Speicherplatz und verknüpfe mit neues Element mit bisheriger Liste
 - ▷ Nachfolger-Pointer des bisherigen letzten Listenelements auf auf neues Listenelement zeigen lassen
- ▶ Fülle das Listenelement mit Schlüsselwert und Nachfolger (= NULL)

```
1 void append(list *lp, int n){  
2     while(*lp != NULL)  
3         lp = &((*lp)->next) ;  
4     (*lp) = malloc(sizeof(struct element));  
5     (*lp)->key = n;  
6     (*lp)->next = NULL;  
7 }
```

Liste erstellen (mit `append`)

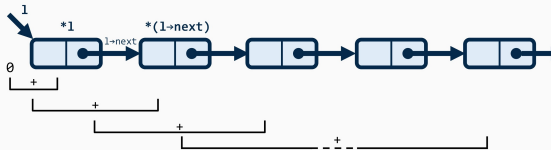
- ▶ erzeuge leere Liste
- ▶ hänge Listenelemente an leere Liste an (durch Aufrufe von `append`)

```
1 list l = NULL;  
2 append(&l, 4);  
3 append(&l, 2);  
4 append(&l, 0);
```

AUFGABE 2 — TEIL (C)

Summe einer Liste — iterativ

- addiere Schlüsselwerte in Hilfsvariable `result` auf
- laufe Liste entlang → Pointer immer weiter „schalten“

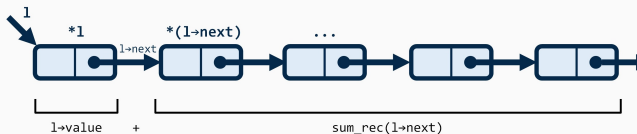


```
1  int sum_it(list l) {  
2      int result = 0;  
3      while (l != NULL) {  
4          result = result + l->value;  
5          l = l->next;    // "start"zeiger weiterschalten  
6      }  
7      return result;  
8  }
```

AUFGABE 2 — TEIL (C)

Summe einer Liste — rekursiv

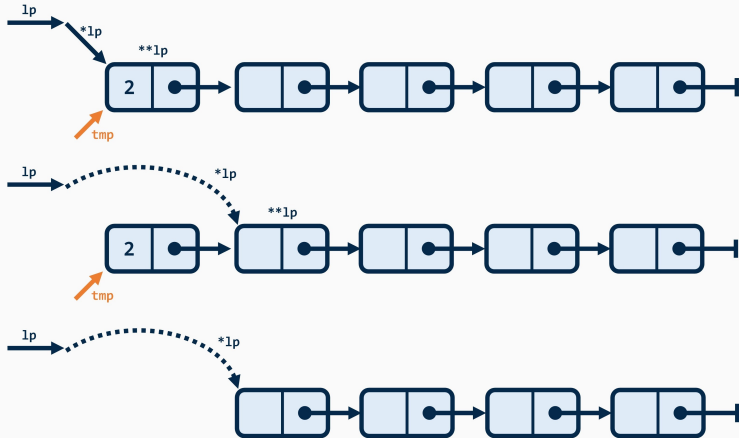
- definiere Basisfall: leere Liste → Rückgabewert 0
- rufe Funktion rekursiv auf — mit `next`-Pointer als neuen Startpointer



```
1 int sum_rec(list l) {  
2     if (l == NULL) return 0;  
3     /* nach letztem element nichts mehr addieren */  
4     return l->value + sum_rec(l->next);  
5     /* nimm key und addiere summe der restliste */  
6 }
```

AUFGABE 2 — TEIL (D)

Löschen von Elementen



Löschen von Elementen — iterativ

- ▶ laufe Liste entlang, solange nicht am Ende angekommen
- ▶ zeigt *lp auf einen geraden Schlüssel → *löschen*
 - ▷ merke Zugriff auf *lp (zu löschen) in tmp
 - ▷ überspringe zu lösches Element
 - ▷ befreie zu lösches Element (Zugriff via tmp)

```
1 void rmEvens_it(list *lp) {  
2     while (lp != NULL && *lp != NULL) {  
3         if ((*lp)->value % 2 == 0) {  
4             list tmp = *lp;  
5             *lp = (*lp)->next;  
6             free(tmp);  
7         } else  
8             lp = &(*lp)->next;  
9     }  
10 }
```

Löschen von Elementen — rekursiv

- ▶ Basisfall: keine Liste oder Liste leer → tue nichts
- ▶ Fallunterscheidung bzgl. Schlüsselwert
 - ▷ gerade: löschen & Speicher befreien wie in iterativer Variante
 - ▷ ungerade: überspringe dieses Element
- ▶ verfare so weiter mit der Restliste

```
1 void rmEvens_rec(list *lp) {  
2     if (lp == NULL || *lp == NULL) return ;  
3     if ((*lp)->value % 2 == 0) {  
4         list tmp = *lp;  
5         *lp = (*lp)->next;  
6         free(tmp);  
7     } else  
8         lp = &(*lp)->next;  
9     rmEvens_rec(lp);  
10 }
```

AUFGABE 2 — TEIL (E)

neue Liste ohne bestimmte Elemente

- ▶ erzeuge neue Liste
- ▶ laufe durch alte Liste durch – betrachte Schlüsselwerte
 - ▷ gerade: Element überspringen
 - ▷ ungerade: an neue Liste anfügen

```
1  list odds(list lp){
2      list erg = NULL; /* erzeuge neue liste */
3      while (lp != NULL ){
4          if (lp->value % 2 != 0)
5              append(&erg, lp->value);
6          lp = lp->next;
7      }
8      return erg;
9  }
```

Next Level: Bäume