Praxis der Programmierung

Dynamische Datentypen

Institut für Informatik und Computational Science
Universität Potsdam

Henning Bordihn

Einige Folien gehen auf A. Terzibaschian zurück.

Dynamische Datentypen

Dynamische Datentypen – Motivation

Aufgabe: Kundenverzeichnis verwalten

- Zu jeder Zeit soll ein neuer Kunde hinzukommen oder entfernt werden können.
- Verzeichnis ist Liste von Kunden, deren Länge sich zur Laufzeit dynamisch ändert.
- Wie kann man das in C umsetzen?

Ausgangspunkt

zusammengesetzter Datentyp `Customer`

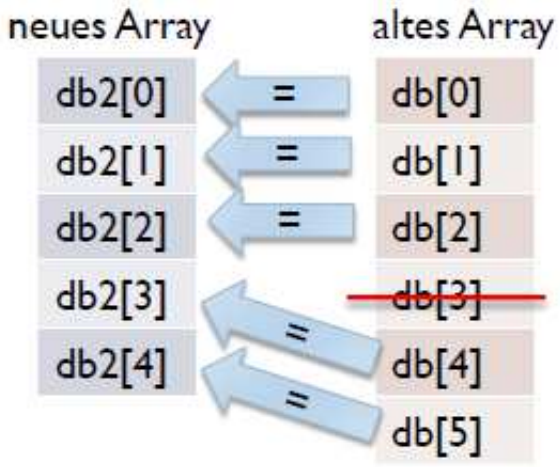
```
struct customer {
   short id;
   char name[64];
   char adress[64];
   short phone;
   float b_volume;
};

typedef struct customer Customer;

Customer ctms[1024];   // Groesse unveraenderlich
```

Kundenverwaltung mit Arrays

- Beispiel: Kunde löschen (aus Array???)
- erster Ansatz: – neues Array anlegen mit einem Element weniger
  – alles kopieren außer zu löschendes Element



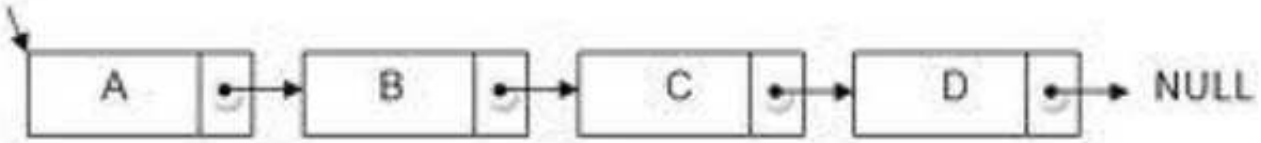
- sehr viele Kopien entstehen; Indexzuordnung ändert sich ständig

Einfach verkettete Listen

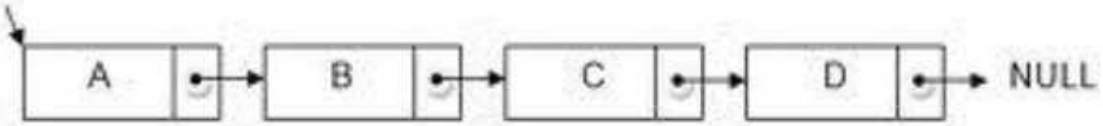
- zweiter Ansatz: Nutzung dynamischer Datenstrukturen
  Beispiele: verkettete Listen, Bäume, Mengen, ...

- in fast allen modernen Sprachen vordefiniert (Standard-Bibliotheken);
  in C leider keine eingebaute verkettete Liste

- Konzept einfach verkettete Liste:
  neben den Daten wird ein Pointer auf das nächste Listenelement gespeichert



Kundenverwaltung mit einfach verketteten Listen



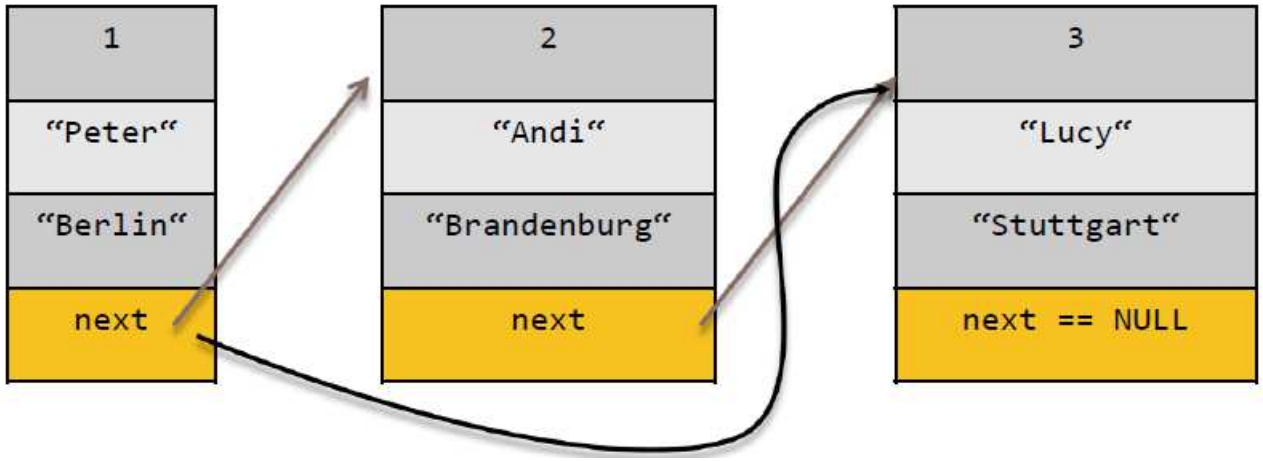
- zusätzliche Membervariable vom Typ Pointer auf Kunde (`Customer`)

```
struct customer {
    short id;
    char name[64];
    char adress[64];
    short phone;
    float b_volume;
    struct customer * next;
}
```

- NULL-Pointer zum Auffinden des Listenendes
- Elemente liegen *nicht* hintereinander im Speicher (*Warum nicht?*)

Arbeit mit einfach verketteten Listen

- Einfügen Löschen, Umordnen durch Umsetzen der Pointer,
  z.B. Löschen eines Listenelements ("Andi"):



kein Kopieren der Daten

- Suchen von Elementen, Bestimmen der Länge, Ausgabe aller Elemente usw.
  durch Verfolgung der Pointer

Aufgabe: einfach verkettete Liste ganzer Zahlen

Erstellen Sie eine C-Datei mit

1. einem Typ `listelement` zum Speichern eines `int` und der Adresse des nächsten Listenelements,
2. einem Typ `list` zum Adressieren einer Liste aus solchen Elementen.

   $\rightsquigarrow$ *Wie kann dieser Typ realisiert werden?*

Aufgabe: einfach verkettete Liste ganzer Zahlen

Erstellen Sie eine C-Datei mit

1. einem Typ `listelement` zum Speichern eines `int` und der Adresse des nächsten Listenelements,
2. einem Typ `list` zum Adressieren einer Liste aus solchen Elementen.
   $\rightsquigarrow$ *Wie kann dieser Typ realisiert werden?*

```
typedef
struct le {
   int value;
   struct le * next;
} listelement;

typedef listelement * list;   // pointer to first element
```

Operationen für einfach verkettete Listen

Ergänzen Sie je eine Funktion

1. zum Einfügen eines Listenelements als neues erstes Element,

2. zum Ausgeben aller Elemente einer Liste.

Operationen für einfach verkettete Listen

Ergänzen Sie je eine Funktion

1. zum Einfügen eines Listenelements als neues erstes Element,

2. zum Ausgeben aller Elemente einer Liste.

```
list insert(int v, list l) {
```

```
void print_list(list l) {
```

Operationen für einfach verkettete Listen

Ergänzen Sie je eine Funktion

1. zum Einfügen eines Listenelements als neues erstes Element,

2. zum Ausgeben aller Elemente einer Liste.

```
list insert(int v, list l) {
   list new;
   new = malloc(sizeof(listelement));
   new->value = v;
   new->next = l;
   return new;
}
```

```
void print_list(list l) {
   if (l == NULL) printf("leer");
   else
      while (l != NULL) {
         printf("%d ", l->value);
         l = l->next;
      }
}
```

Testen der einfach verkettete Listen

Ergänzen Sie eine main-Funktion, in der Sie Ihre Operationen mit

1. einer leeren Liste,
2. einer Liste mit einem Element,
3. einer Liste mit mindestens drei Elementen

testen.

- Eine leere Liste ist eine, deren "erstes Element" eine Nullreferenz ist.
- Nutzen Sie zum Erzeugen von Listen Ihre Funktion zum Einfügen von Elementen, wobei Sie mit einer leeren Liste beginnen.

Operationen für einfach verkettete Listen (2)

Ergänzen Sie je eine Funktion

1. zum Berechnen der Listenlänge (Anzahl der Elemente),
2. zum Entfernen des ersten Listenelements. *Denken Sie an Speicherfreigaben!*

Operationen für einfach verkettete Listen (2)

Ergänzen Sie je eine Funktion

1. zum Berechnen der Listenlänge (Anzahl der Elemente),
2. zum Entfernen des ersten Listenelements. *Denken Sie an Speicherfreigaben!*

```
int length(list l) {
```

```
list delete_head(list l) {
```

Operationen für einfach verkettete Listen (2)

Ergänzen Sie je eine Funktion

1. zum Berechnen der Listenlänge (Anzahl der Elemente),
2. zum Entfernen des ersten Listenelements. *Denken Sie an Speicherfreigaben!*

```
int length(list l) {

   int count = 0;
   while (l != NULL) {
        count++;
        l = l->next;
   }
   return count;
}
```

```
list delete_head(list l) {

   if (l == NULL) return l;
   list new = l->next;
   free(l);
   return new;
}
```

3. Testen mit allen Listen!

Operationen für einfach verkettete Listen (3)

Ergänzen Sie eine Funktion zum Löschen der gesamten Liste.
Denken Sie wieder an die Freigabe des Speichers und an das Testen!

Operationen für einfach verkettete Listen (3)

Ergänzen Sie eine Funktion zum Löschen der gesamten Liste.
Denken Sie wieder an die Freigabe des Speichers und an das Testen!

```
void delete_all(list l) {
   if (l == NULL) return;
   list next = l->next;
   free(l);
   delete_all(next);
}
```

```
void delete_all(list l) {
    list next;
    while (l != NULL) {
        next = l->next;
        free(l);
        l = next;
    }
}
// alternative Implementierung
```

Operationen für einfach verkettete Listen (4)

Ergänzen Sie eine Funktion, die den Index des ersten Elements ermittelt, das die als Parameter übergebene Zahl als Wert speichert.

Operationen für einfach verkettete Listen (4)

Ergänzen Sie eine Funktion, die den Index des ersten Elements ermittelt, das die als Parameter übergebene Zahl als Wert speichert.

```
int contains(int v, list l) {
   int index = 1;
   while (l != NULL) {
        if (l->value == v)    // found
            return index;
        else {                  // not yet found
            index++;
            l = l->next;
        }
   }
   return 0;                    // not in list (NULL reached)
}
```

Operationen für einfach verkettete Listen (5)

Ergänzen Sie eine Funktion, die das erste Elements löscht,
das die als Parameter übergebene Zahl als Wert speichert.

Operationen für einfach verkettete Listen (5)

Ergänzen Sie eine Funktion, die das erste Elements löscht,
das die als Parameter übergebene Zahl als Wert speichert.

```
list delete(int v, list l) {
   int index = contains(v, l);
   if (index == 0) return l;       // true if l empty or v not in l
   if (index == 1)
        return delete_head(l);
   int i;
   list e, tmp = l;
   for(i=1; i<index-1; ++i)
        tmp = tmp->next;           // tmp points to predecessor
   e = tmp->next;                  // e: element to be deleted
   tmp->next = e->next;
   free(e);
   return l;
}
```