U8 7. Übung

- Dynamische Speicherverwaltung
- Generisches Sortieren
- Aufgabe 7

U8-1 Dynamische Speicherverwaltung

- Erzeugen von Feldern der Länge n:
  - mittels: `void *malloc(size_t size)`

```
struct person *personen;
personen = (struct person *)malloc(sizeof(struct person)*n);
if(personen == NULL) ...
```

  - mittels: `void *calloc(size_t nelem, size_t elsize)`

```
struct person *personen;
personen = (struct person *)calloc(n, sizeof(struct person));
if(personen == NULL) ...
```

  - `calloc` initialisiert den Speicher mit `0`
  - `malloc` initialisiert den Speicher nicht

- Freigeben von Speicher

```
void free(void *ptr);
```

  - nur Speicher, der mit einer der alloc-Funktionen zuvor angefordert wurde, darf mit free freigegeben werden!

U8-1 Dynamische Speicherverwaltung

- Längenänderung von dynamisch allokierten Feldern:

```
void *realloc(void *ptr, size_t size)
```

- Die Position im Speicher kann sich hierbei evtl. ändern
  - ➤ An das Feld angrenzender Speicher ist bereits belegt
  - ➤ **realloc** sucht einen neuen, ausreichend großen Speicherbereich und kopiert das ursprüngliche Feld an dessen Anfang
  - ➤ **realloc** liefert einen Zeiger auf die neue Position zurück

- Beispiel: Erweiterung des allokierten Felds um 10 weitere Strukturen

```
neu = (struct person *)realloc(personen,
                               (n+10) * sizeof(struct person));
if(neu == NULL) ...
```

- Wenn realloc keinen ausreichend großen Bereich findet, wird `NULL` zurückgegeben und der ursprüngliche Bereich bleibt erhalten.

U8-2 Generisches Sortieren mit qsort

1 Funktion *qsort(3)*

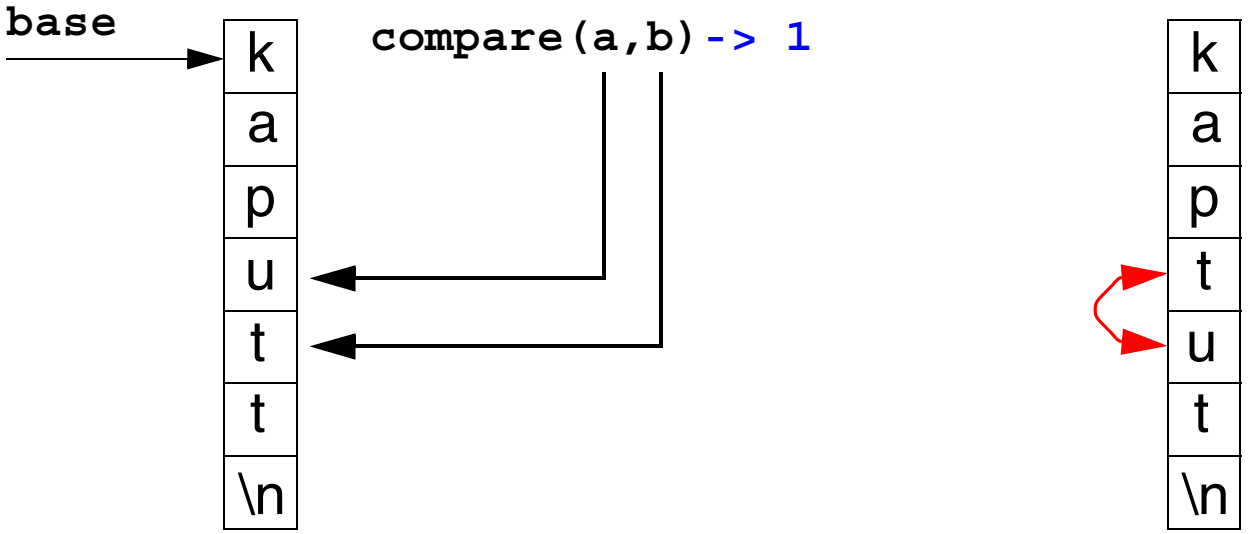
- Prototyp aus `stdlib.h`:

```
void qsort(void *base,
           size_t nel,
           size_t width,
           int (*compare) (const void *, const void *));
```

- Bedeutung der Parameter:
  - **base** : Zeiger auf das erste Element des Feldes, dessen Elemente sortiert werden sollen
  - **nel** : Anzahl der Elemente im zu sortierenden Feld
  - **width**: Größe eines Elements
  - **compare**: Vergleichsfunktion
- Sortiert ein Feld von beliebigen Elementen

2 Arbeitsweise von *qsort(3)*

- **qsort** vergleicht je zwei Elemente mit Hilfe der Vergleichsfunktion `compare`
- sind die Elemente zu vertauschen, dann werden die entsprechenden Felder komplett ausgetauscht, z.B.:

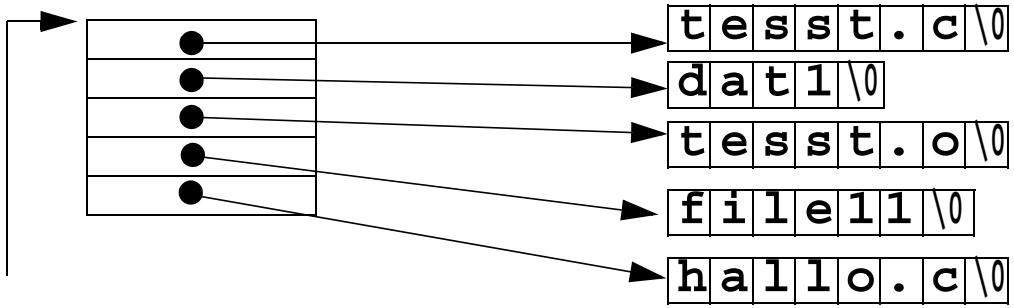


3 Vergleichsfunktion

- Die Vergleichsfunktion erhält Zeiger auf Feldelemente, d.h. die übergebenen Zeiger haben denselben Typ wie das Feld
- Die Funktion vergleicht die beiden Elemente und liefert:
  - <0, falls Element 1 kleiner bewertet wird als Element 2
  - 0, falls Element 1 und Element 2 gleich gewertet werden
  - >0, falls Element 1 größer bewertet wird als Element 2
- Beispiel
  - ’z’, ’a’ ➠ 1
  - 1, 5 ➠ -1
  - 5,5 ➠ 0
  - "foo", "bar" ➠ 1

4 Beispiel: Strings sortieren

- Strings: '\0'-terminierte Zeichenfolgen *beliebiger* (unbekannter) Länge
  - **qsort** sortiert Elemente *fester* Länge
  - direkte Sortierung mit **qsort** daher nicht möglich
- Ein Feld aus Zeigern auf die Strings sortieren
  - die Zeiger haben alle die gleiche *feste* Länge
  - Feldelemente haben den Typ `char *`




```
char **strings = malloc(n * sizeof(char *));
```

4 Sortieren des Zeigerfeldes mit qsort

- Funktion qsort aufrufen

```
void qsort(void *base,
           size_t nel,
           size_t width,
           int (*compare) (const void *, const void *));
```

- Bedeutung der Parameter:
  - **base** : Zeiger auf erstes Element des zu sortierenden Feldes *(Zeiger auf den Zeiger auf ersten String)*
  - **nel** : Anzahl der Elemente im zu sortierenden Feld *(Zahl der Zeiger/Strings)*
  - **width**: Größe eines Elements *(Größe eines char-Zeigers - sizeof)*
  - **compare**: Zeiger auf Vergleichsfunktion

4 Beispiel: Compare Funktion

- Lösung für die compare-Funktion:
  - **qsort** übergibt der compare-Funktion Zeiger auf Elemente
  - **compare** wird also mit Zeigern auf `char *` aufgerufen ➠ `char **`
  - **strcmp(3)** wird verwendet, um die Strings zu vergleichen

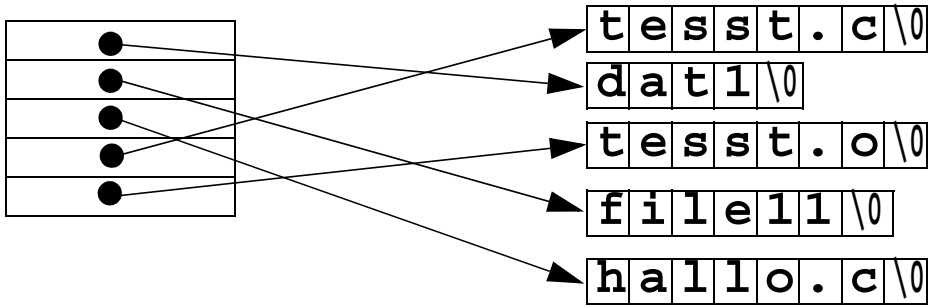
```
int compare(const char **a, const char **b) {
        return strcmp( *a, *b );
}
```

- Aufruf von qsort (`nel=5`) im Beispiel

```
qsort(strings, nel, sizeof(char *),
    (int (*) (const void *, const void *)) compare);
```

  - Funktionstyp der **compare**-Funktion ist generisch (mit `void *`)
  - Typ der **compare**-Funktion muss für den **qsort**-Aufruf gecastet werden

4 Resultat nach dem Aufruf von qsort



- die Strings stehen unverändert, nur die Zeiger im Feld wurden sortiert

U8-3 Aufgabe 7

- Einfaches Sortierprogramm
- Wörter zeichenweise von der Standardeingabe einlesen
  - ein Wort pro Zeile (Wörter können auch Leer-/Sonderzeichen enthalten)
  - maximale Zahl der Zeichen unbekannt
  - **jede** Zeile endet mit einem Newline Zeichen ('\n')
  - Speicher muss dynamisch allokiert und nachgefordert werden (womit?)
- Dabei die Zahl der Wörter zählen und gültige C-String erzeugen (wie?)
- Array für Zeiger auf jedes Wort allokieren (wie gross?)
- Zeigerarray mit Zeigern auf die einzelnen Wörter füllen
- Zeigerarray mit **qsort(3)** und **strcmp(3)** sortieren
- Wörter ausgeben, ein Wort pro Zeile