Systemnahe Programmierung in C (SPiC)

21 Ergänzungen – Zeiger

Jürgen Kleinöder, Daniel Lohmann, Volkmar Sieh

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Sommersemester 2022

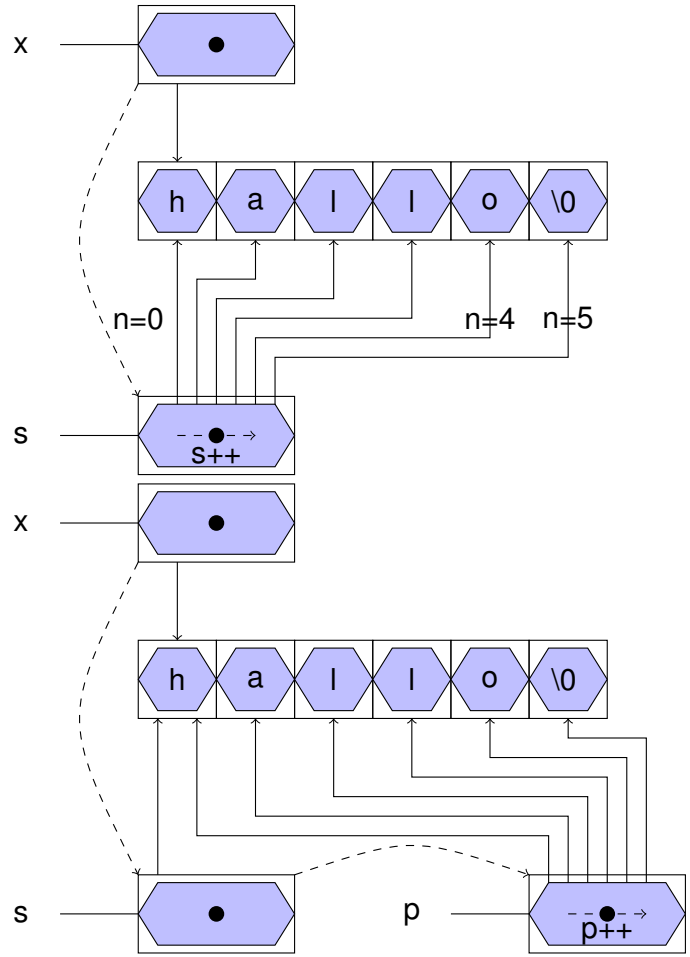
http://sys.cs.fau.de/lehre/SS22/spic

Zeiger, Felder und Zeichenketten

- Zeichenketten sind Felder von Einzelzeichen (`char`), die in der internen Darstellung durch ein `'\0'`-Zeichen abgeschlossen sind
- Beispiel: Länge eines Strings ermitteln – Aufruf `strlen(x);`

```
/* 1. Version */
int strlen(const char *s)
{
    int n;
    for (n = 0; *s != '\0'; n++) {
        s++;
    }
    return n;
}
```

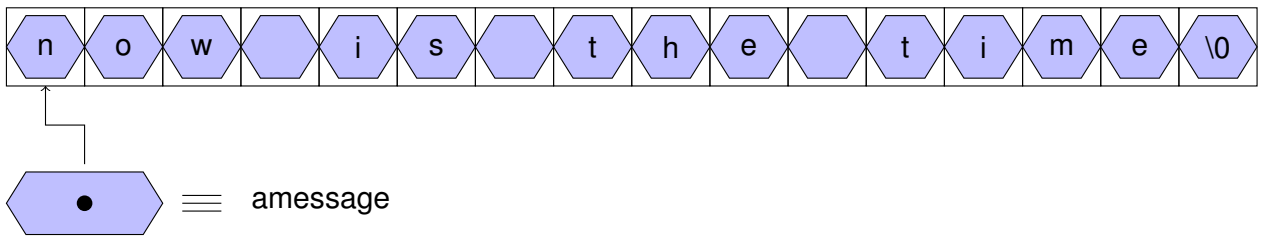
```
/* 2. Version */
int strlen(const char *s)
{
    const char *p = s;
    while (*p != '\0') {
        p++;
    }
    return p - s;
}
```



Zeiger, Felder und Zeichenketten (Forts.)

- wird eine Zeichenkette zur Initialisierung eines `char`-Feldes verwendet, ist der Feldname ein konstanter Zeiger auf den Anfang der Zeichenkette

```
char amessage[] = "now is the time";
```



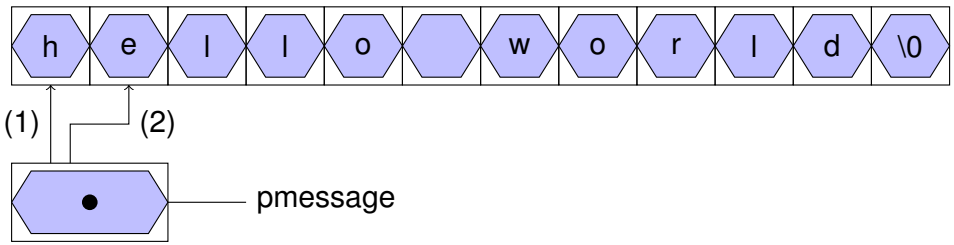
- es wird ein Speicherbereich für 16 Bytes reserviert und die Zeichen werden in diesen Speicherbereich hineinkopiert
- `amessage` ist ein konstanter Zeiger auf den Anfang des Speicherbereichs und kann nicht verändert werden
- der Inhalt des Speicherbereichs kann aber modifiziert werden

```
amessage[0] = 'h';
```


Zeiger, Felder und Zeichenketten (Forts.)

- wird eine Zeichenkette zur Initialisierung eines `char`-Zeigers verwendet, ist der Zeiger eine
  Variable, die mit der Anfangsadresse der Zeichenkette initialisiert wird

```
const char *pmessage = "hello world";    /*(1)*/
```

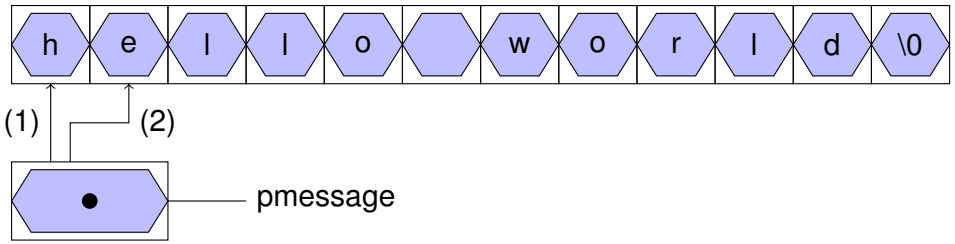



```
pmessage++;    /*(2)*/
printf("%s\n", pmessage); /* gibt "ello world" aus */
```

- die Zeichenkette selbst wird vom Compiler als konstanter Wert (String-Literal) im Speicher angelegt
- es wird ein Speicherbereich für einen Zeiger reserviert (z.B. 4 Byte) und mit der Adresse der Zeichenkette initialisiert

Zeiger, Felder und Zeichenketten (4)

```
const char *pmessage = "hello world";    /*(1)*/
```




```
pmessage++;    /*(2)*/
printf("%s\n", pmessage); /* gibt "ello world" aus */
```

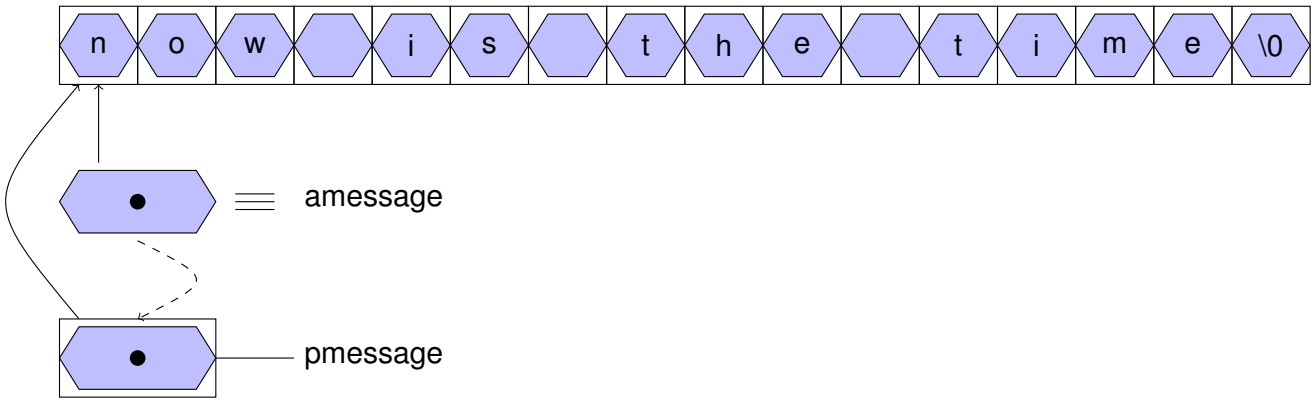
- `pmessage` ist ein variabler Zeiger, der mit dieser Adresse initialisiert wird, aber jederzeit verändert werden darf (`pmessage++;`)
- der Speicherbereich von `"hello world"` darf aber nicht verändert werden
  - der Compiler erkennt dies durch das Schlüsselwort `const` und verhindert schreibenden Zugriff über den Zeiger
  - manche Compiler legen solche Zeichenketten ausserdem im schreibgeschützten Speicher an (=> Speicherschutzverletzung beim Zugriff, falls der Zeiger nicht als `const`-Zeiger definiert wurde)

Zeiger, Felder und Zeichenketten (5)

- die Zuweisung eines `char`-Zeigers oder einer Zeichenkette an einen `char`-Zeiger bewirkt kein Kopieren von Zeichenketten!

```
pmessage = amessage;
```

weist dem Zeiger `pmessage` lediglich die Adresse der Zeichenkette `"now is the time"` zu



- wird eine Zeichenkette als aktueller Parameter an eine Funktion übergeben, erhält diese eine Kopie des Zeigers

Zeiger, Felder und Zeichenketten (6)

- Um eine ganze Zeichenkette einem anderen char-Feld zuzuweisen, muss sie kopiert werden: Funktion `strcpy` in der Standard-C-Bibliothek
- Implementierungsbeispiele:

```
/* 1. Version */
void strcpy(char s[], char t[]) {
    int i = 0;
    while ((s[i] = t[i]) != '\0') {
        i++;
    }
}
```

```
/* 2. Version */
void strcpy(char *s, char *t) {
    while ((*s = *t) != '\0') {
        s++, t++;
    }
}
```

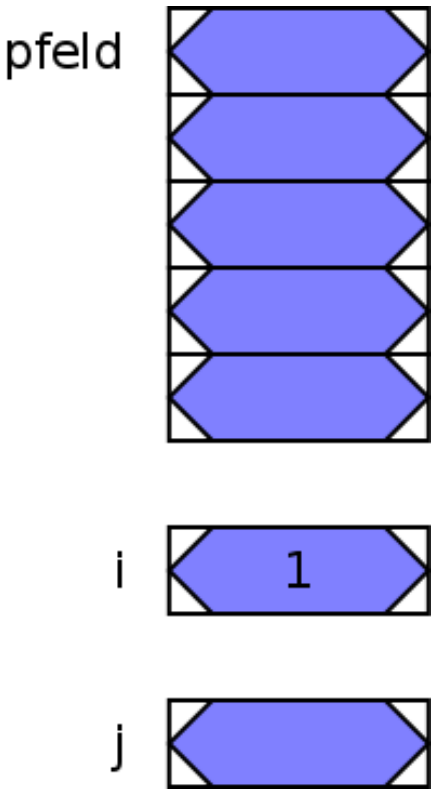
```
/* 3. Version */
void strcpy(char *s, char *t) {
    while (*s++ = *t++) {
    }
}
```

Felder von Zeigern

Auch von Zeigern können Felder gebildet werden

- Deklaration

```
int *pfeld[5];
int i = 1;
int j;
```



Felder von Zeigern (Forts.)

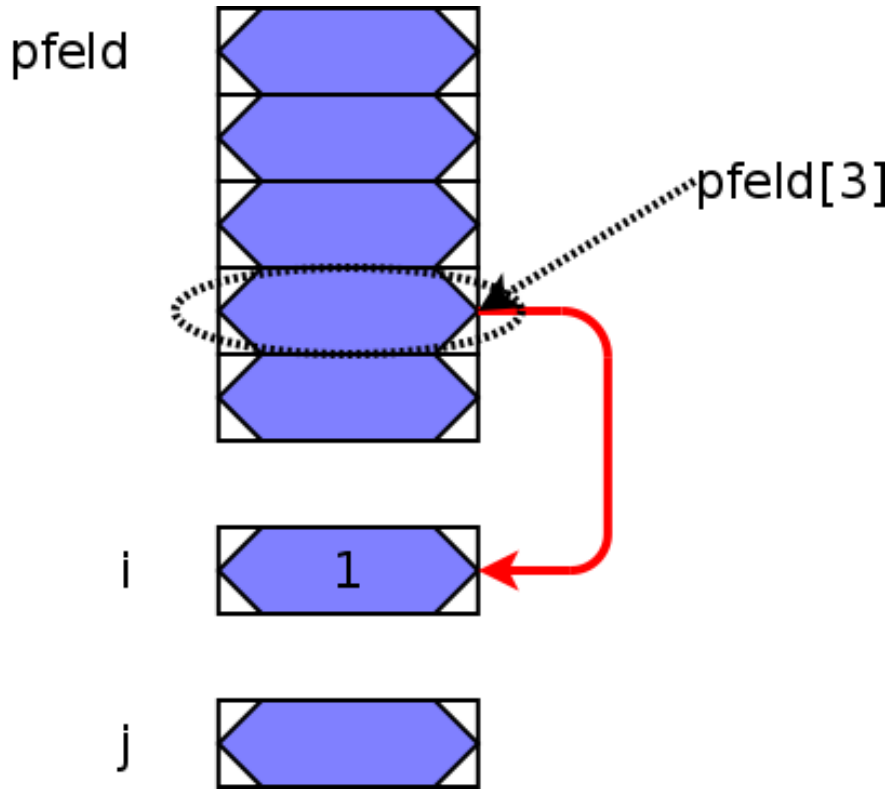
Auch von Zeigern können Felder gebildet werden

- Deklaration

```
int *pfeld[5];
int i = 1;
int j;
```

- Zugriff auf einen Zeiger des Feldes

```
pfeld[3] = &i;
```



Felder von Zeigern (Forts.)

Auch von Zeigern können Felder gebildet werden

- Deklaration

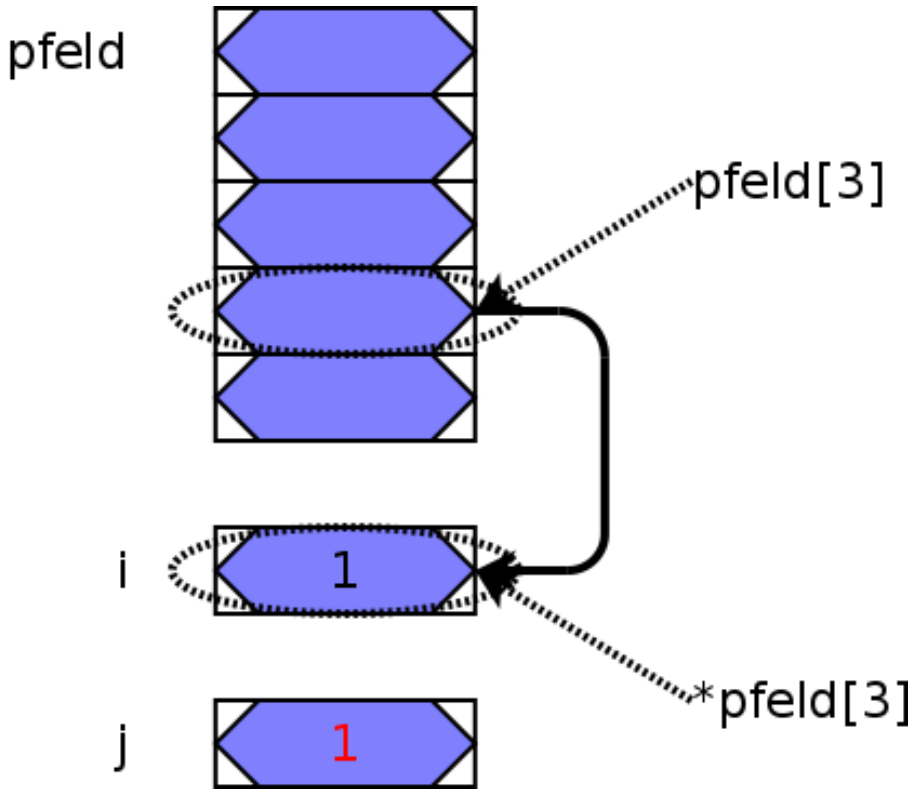
```
int *pfeld[5];
int i = 1;
int j;
```

- Zugriff auf einen Zeiger des Feldes

```
pfeld[3] = &i;
```

- Zugriff auf das Objekt, auf das ein Zeiger des Feldes verweist

```
j = *pfeld[3];
```

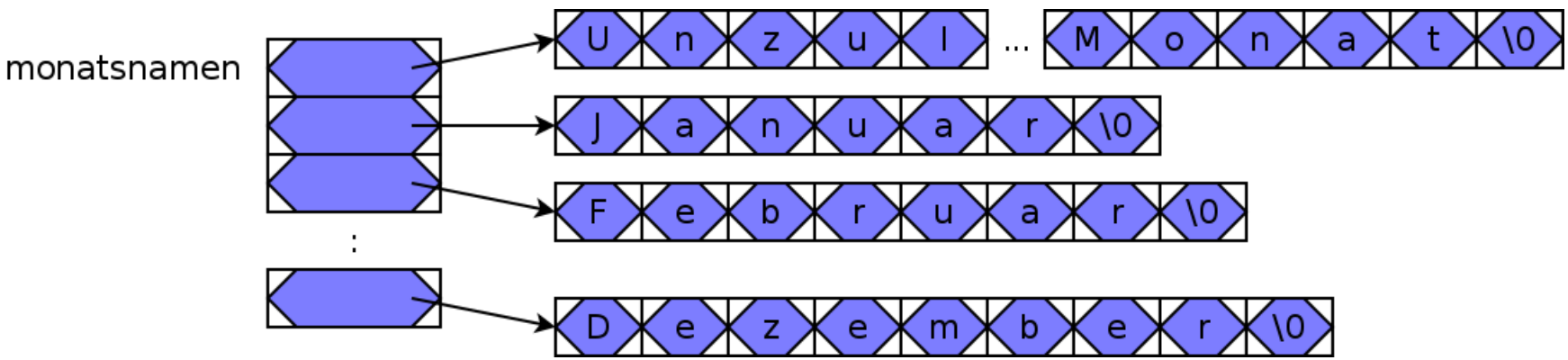


Felder von Zeigern (Forts.)

Beispiel: Definition und Initialisierung eines Zeigerfeldes:

```
const char *
month_name(int n)
{
    static const char *monatsname[] = {
        "Unzulaessiger Monat",
        "Januar",
        ...
        "Dezember"
    };

    return (n < 1 || 12 < n) ? monatsname[0] : monatsname[n];
}
```



Argumente aus der Kommandozeile

- beim Aufruf eines Programms können normalerweise Argumente übergeben werden
- der Zugriff auf diese Argumente wird der Funktion `main()` durch zwei Aufrufparameter ermöglicht (beide Varianten gleichwertig):

```
int
main(int argc, char *argv[])
{
    ...
}
```

```
int
main(int argc, char **argv)
{
    ...
}
```

- der Parameter `argc` enthält die Anzahl der Argumente, mit denen das Programm aufgerufen wurde
- der Parameter `argv` ist ein Feld von Zeigern auf die einzelnen Argumente (Zeichenketten)
- der Programmname wird als erstes Argument übergeben (`argv[0]`)

Argumente aus der Kommandozeile

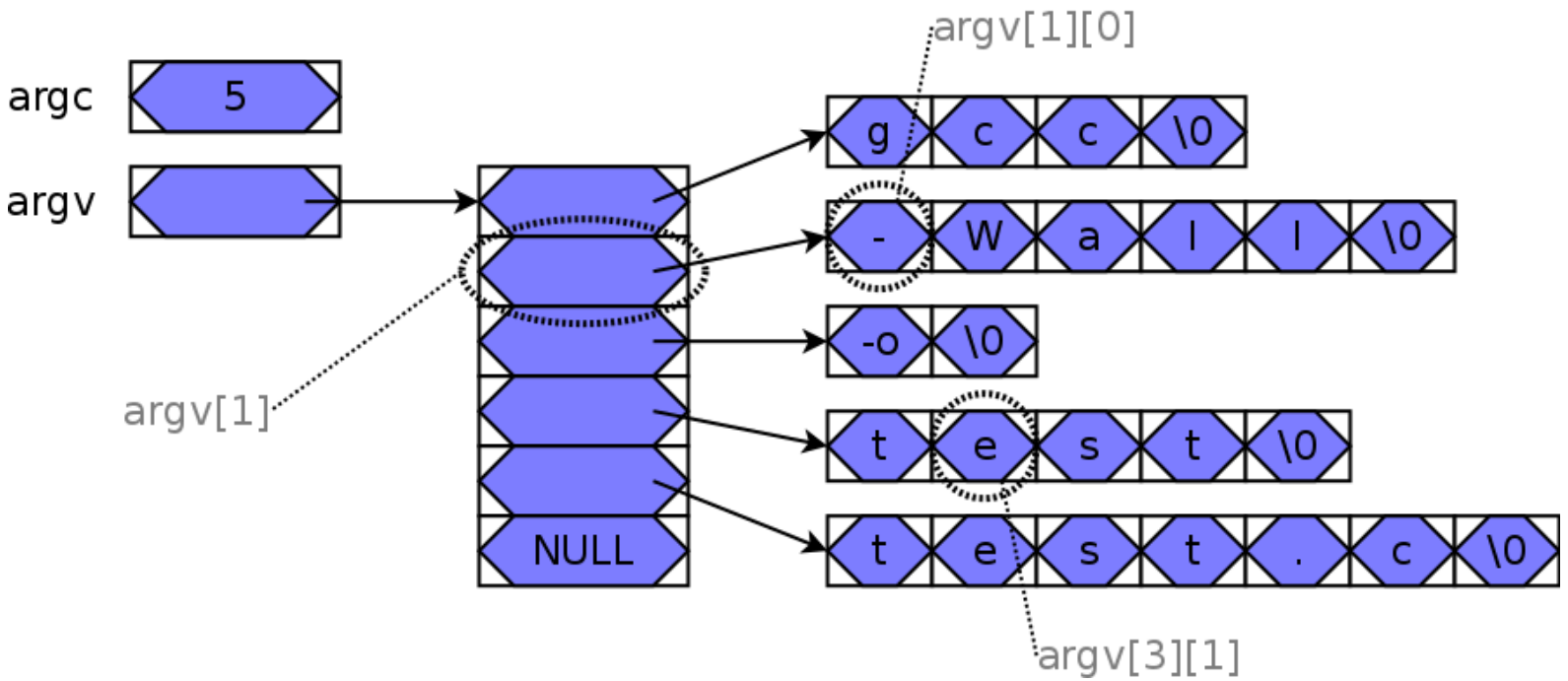
- Kommando:

```
gcc -Wall -o test test.c
```

- C-Datei:

```
...
int main(int argc, char *argv[])
...
```

```
...
int main(int argc, char **argv)
...
```



Argumente – Beispiel

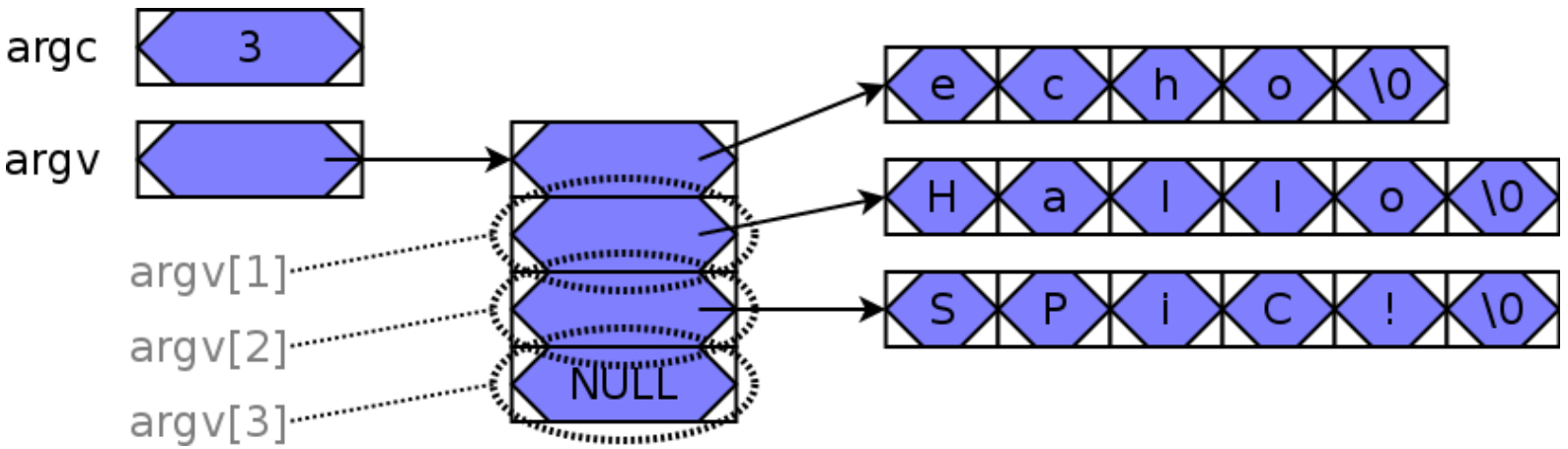
Beispiel: **echo**-Programm

```
~> echo Hallo SPiC!
Hallo SPiC!
~>
```

```c
#include <stdio.h>

int
main(int argc, char *argv[])
{
    for (int i = 1; i < argc; i++) {
        printf("%s ", argv[i]);
    }
    printf("\n");

    return 0;
}
```



Verbund-Datentypen / Strukturen

- Zusammenfassen mehrerer Daten zu einer Einheit
- Struktur-Deklaration

```
struct person {
    char name[20];
    int age;
};
```

- Definition einer Variablen vom Typ der Struktur

```
struct person p1;
```

- Zugriff auf ein Element der Struktur

```
strcpy(p1.name, "Peter Pan");
p1.age = 12;
```


Zeiger auf Strukturen

- Konzept analog zu „Zeiger auf Variable”
  - Adresse einer Struktur mit &-Operator zu bestimmen
- Beispiel

```
struct person stud1;
struct person *pstud;
pstud = &stud1;
```

- Besondere Bedeutung beim Aufbau verketteter Strukturen (Listen, Bäume, ...)
  - eine Struktur kann Adressen weiterer Strukturen desselben Typs enthalten

Zeiger auf Strukturen (Forts.)

- Zugriff auf Strukturkomponenten über Zeiger
- bekannte Vorgehensweise
  - „*”-Operator liefert die Struktur
  - „.”-Operator liefert ein Element der Struktur
  - **Aber:** Operatorenvorrang beachten!

```
(*pstud).age = 21;
```

- syntaktische Verschönerung
  - „->”-Operator

```
pstud->age = 21;
```


Verschachtelte/verkettete Strukturen

- Strukturen in Strukturen sind erlaubt – aber:
  - die Größe einer Struktur muss vom Compiler ausgerechnet werden können
    => Struktur kann sich nicht selbst enthalten
  - die Größe eines Zeigers ist bekannt
    => Struktur kann Zeiger auf gleiche Struktur enthalten
  - Beispiele:

Verkettete Liste:

```
struct list {
    struct list *next;
    struct person stud;
};

struct list *head;
```

Baum:

```
struct tree {
    struct tree *left;
    struct tree *right;
    struct person stud;
};

struct tree *root;
```

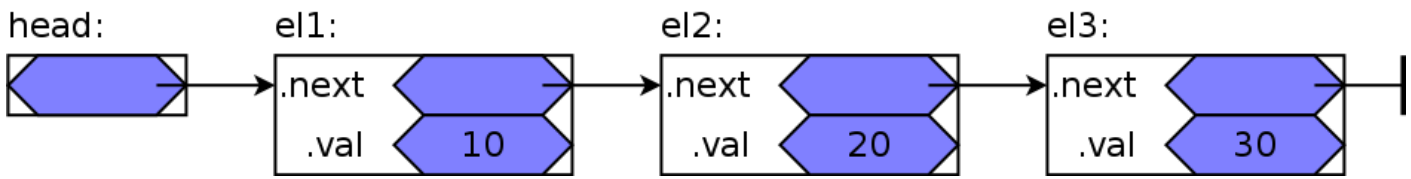

Verkettete Listen

- Mehrere Strukturen desselben Typs werden über Zeiger miteinander verkettet

```
struct list { struct list *next; int val; };

struct list el1, el2, el3;
struct list *head;

head = &el1;
el1.next = &el2; el2.next = &el3; el3.next = NULL;
el1.val = 10;    el2.val = 20;    el3.val = 30;
```



- Laufen über eine verkettete Liste

```
int sum = 0;
for (struct list *curr = head; curr != NULL; curr = curr->next) {
    sum += curr->val;
}
```