



# Grundlagen der Programmierung in C

## Dynamic Memory Allocation

Wintersemester 2005/2006  
G. Zachmann  
Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)



## Problem



- Wir müssen "dynamische" Daten im Speicher halten, von denen wir selbst zur Laufzeit die Lebenszeit bestimmen
  - globale oder lokale Variablen funktionieren nicht
- Idee:
  - Pointer verwenden, um auf diese "dynamischen" Daten zu zeigen
  - Neue Konstrukte in Sprache einbauen zur Erzeugung/Löschen solcher Daten
  - Diese dynamischen Daten von unten nach oben wachsen lassen
- Heap mit **new** und **delete**



## Sprachelemente für dynamische Variablen



- Erzeugung:

`new T`

mit *T* bekannter Typ, Resultat = Pointer auf *T*.

- Beispiele:

```
int* ip = new int;  
float* fp = new float;  
myStruct* sp = new myStruct;
```

- Löschen:

`delete pointer`

wobei *pointer* zuvor mit `new` alloziert worden ist.

- Beispiele:

```
delete ip;  
myStruct* sp2 = sp;  
delete sp2;
```



## Lebenszeit dynamisch allozierter Variablen



- Keine Initialisierung (bei built-in Types)
- Lebensende: bei `delete`
- Muß man `delete` machen?
  - Nein: gesamter Speicher wird vom OS am Ende des Programms freigegeben
  - Ja: sonst Memory-Leaks
  - Ja: Finden von Memory-Leaks klappt sonst nicht (mit mem-checker)

## Zugriff auf dynamische Variablen

- Immer nur über lokale oder globale Variablen:

```

myStruct* sp;

int main()
{
    sp = new myStruct;
    ...
}

```

The diagram illustrates memory allocation. A vertical stack represents memory. A blue box labeled 'sp' is located in the upper part of the stack. An arrow points from the 'sp' box down to a green box labeled 'myStruct' located in the lower part of the stack. Another arrow points from the 'sp' box to the right, indicating its location in memory.

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Dynamic Memory Allocation, 7

## Beispiel: Daten an Liste anhängen

- Erinnerung:

```

struct ListElement
{
    int data;
    ListElement* next;
};

struct List
{
    ListElement* first;
    ListElement* last;
    int n_elements;
};

```

The diagram shows a linked list structure. At the top, a 'List' struct is represented as a rounded rectangle containing the number '4' and labeled 'n\_elems'. Two arrows labeled 'first' and 'last' point from the 'List' struct to the first and last nodes of the list, respectively. The list consists of four 'ListElement' nodes, each represented as a rounded rectangle with a 'data' field and a 'next' field. The nodes contain the characters 'a', 'b', 'c', and 'd' in order. The 'next' field of the last node points to 'NULL'. A label 'ListElement' is placed below the last node.

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Dynamic Memory Allocation, 8

```
// n int's von stdin lesen und an list anhaengen
for ( int i = 0; i < n; i ++ )
{
    int a;
    scanf( "%d", &a );

    // neues ListElement mit a generieren
    ListElement* e = new ListElement;
    e->data = a;
    e->next = NULL;

    // e an list anhaengen
    list->last->next = e;
    list->last = e;
}
```

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Dynamic Memory Allocation, 9

## Speicherbugs

- Memory Leak:
  - Ursache: unbenutzter Speicher wird nicht mehr freigegeben
    - Eigtl. Ursache: man bekommt Adresse des Blocks nicht mehr
  - Folge: Programm belegt irgendwann gesamten Speicher (berühmtes Beispiel: Windows98)
- Beispiel:

```
int foo( void )
{
    MyStruct* sp = new MyStruct;
    ...
    return sp->i;
}
int main( void )
{
    int i = foo();
    ...
}
```

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Dynamic Memory Allocation, 10

- Dangling Pointer:
  - Ursache: Speicher wird verwendet, nachdem schon freigegeben
  - Mögliche Folgen:
    - Falsche Werte in anderen dynamischen Variablen
    - Werte sind später verändert
  - Beispiel:

```
struct MyStruct
{
    int m;
};
MyStruct* s1 = new MyStruct;
s1->m = 17;
delete s1;
MyStruct* s2 = new MyStruct;
s2->m = 42;
printf("%d\n", s1->m );
```

Ausgabe
42

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Dynamic Memory Allocation, 12

- Double Delete:
  - Spezieller Fall von "dangling pointer"
  - Ursache: Speicher wird 2x freigegeben
  - Folge: man löscht evtl. Speicher, der jetzt einem anderen Programmteil "gehört"
  - Beispiel:

```
MyStruct* s1 = new MyStruct;
...
delete s1;
MyStruct* s2 = new MyStruct;
...
delete s1;
```

G. Zachmann Grundlagen der Programmierung in C - WS 05/06 Dynamic Memory Allocation, 13



- Radikale Lösung: Programmiersprachen "ohne" Zeiger (Java, Python)
  - Alle Objekte von zusammengesetztem Typ existieren prinzipiell auf dem Heap
  - Variablen mit zusammengesetztem Typ sind prinzipiell Zeiger
  - Programmierer sieht die Zeiger nicht (kein Sprachkonstrukt dafür)
  - Sprache (Interpreter / Virtual Machine) erkennt, wenn Objekt vom Programm nicht mehr zugreifbar ist (kein Zeiger führt mehr darauf)
  - Garbage-Collector läuft im Hintergrund ständig mit
- Langfristig für nicht zeitkritische App.s der Weg