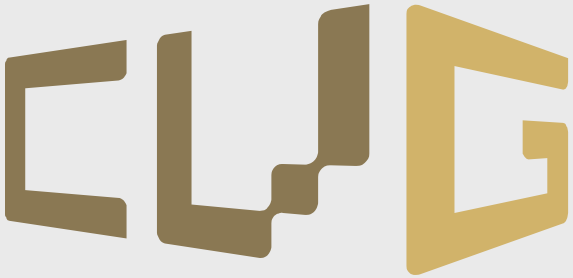


Computer Vision
and Geometry Lab

Informatik I for D-MAVT

Exercise Session 8

Nachbesprechung

- Nachbesprechung Übung 6
- Pointer/Referenzen/Funktionen und verkettete Listen
- Rekursion
- Vorbesprechung Übung 8

Nachbesprechung

- Pointer und Arrays

```
#include <iostream>
using namespace std;

int main() {

   float* a = new float[4];
   a[0] = 5;

   float* b = new float;
   *b = 4;

   delete a;
   delete b;

}
```

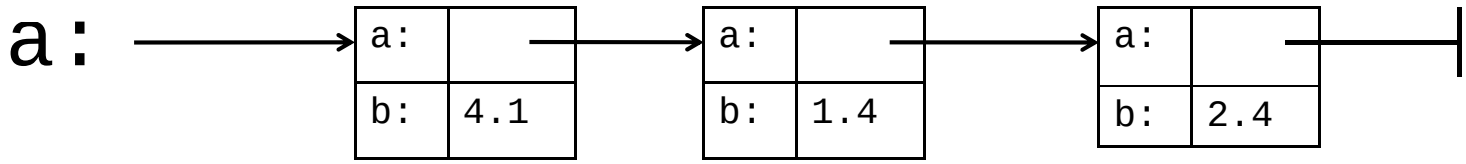
Wieso braucht es bei der Definition vom ersten Element des float arrays a " a[0] = 5; " und bei der Definition des floats b " *b = 4; "?

1. Element: `a[0] = 5  *a = 5`
2. Element: `a[1] = 5  *(a+1) = 5`

`b[0] = 4  *b = 4`

Übung 6: Nachbesprechung

- Aufgabe 2:
  - Struct Darstellung



Übung 6: Nachbesprechung

- Aufgabe 3:
  - Dynamische Arrays

```
int arrayLength;
int* array = new[arrayLength];
cin >> arrayLength;

for(int i = 0; i < arrayLength; i++){
  array[i] = i;
}
```

```
int arrayLength;
int* array;
cin >> arrayLength;
array = new[arrayLength];

for(int i = 0; i < arrayLength; i++){
  array[i] = i;
}
```

  - Grösstes Element

```
int maxVal;
int maxIdx;

for(int i = 0; i < arrayLength; i++){
   if (array[i] > maxVal){
      maxVal = array[i];
      maxIdx = i;
   }
}
```

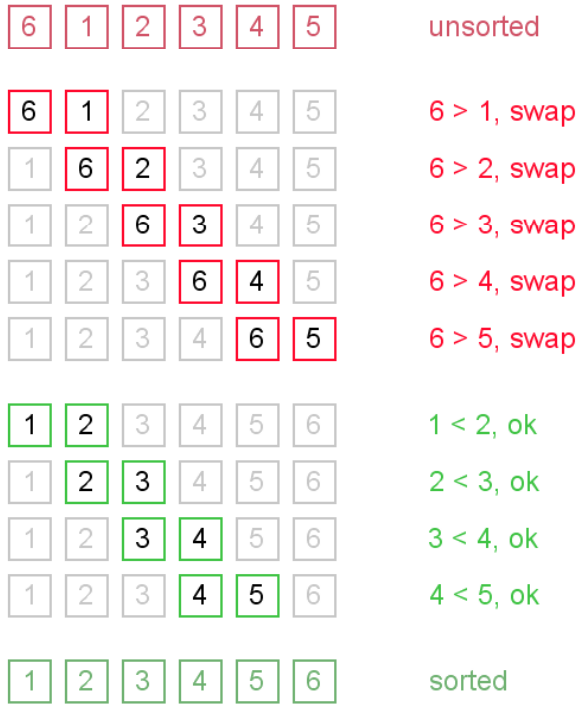
```
int maxIdx = 0;

for(int i = 1; i < arrayLength; i++){
   if (array[i] > array[maxIdx])
      maxIdx = i;
}
```

Übung 6: Nachbesprechung

- Aufgabe 3:
  - Sortieren (Bubble sort)

```
for(int i = n-1; i > 0; i--){
    for(int j = 0; j < i; j++){
       if(array[j] < array[j+1]){
          tmp = array[j];
          array[j] = array[j+1];
          array[j+1] = tmp;
       }
    }
}
```



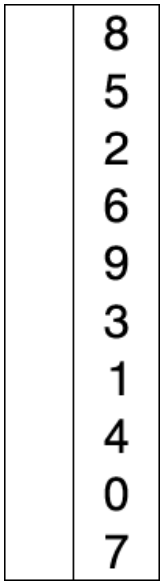
http://fairuzelsaid.wordpress.com

Übung 6: Nachbesprechung

- Aufgabe 3:
  - Sortieren (Selection sort)

```
for(int i = n-1; i > 0; i--){
    maxIdx = i;

    for(int j = 0; j < i; j++){
       if(array[maxIdx] < array[j])
          maxIdx = j;
    }
    tmp = array[i];
    array[i] = array[maxIdx];
    array[maxIdx] = tmp;
}
```



wikipedia.org

Pointer/Referenzen und Funktionen

- Nachbesprechung Übung 6
- Pointer/Referenzen und Funktionen
- Rekursion
- Vorbesprechung Übung 8

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
};
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x	12324234.23
0x102C	p.y	98678768.23
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x	12324234.23
0x102C	p.y	98678768.23
0x1028	pp	0x1012324
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x	12324234.23
0x102C	p.y	98678768.23
0x1028	pp	0x1012324
0x1024	ppp	0x1343521
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

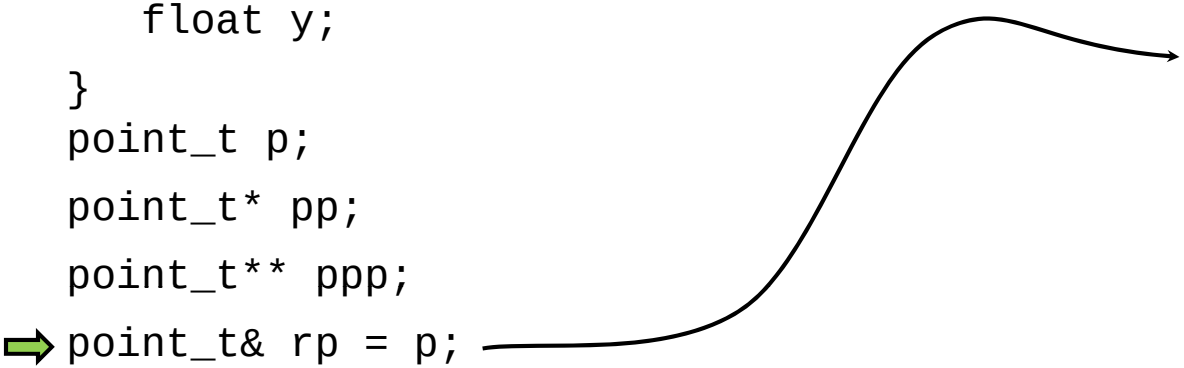
Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```



Stack (vereinfacht)

0x1030	p.x, rp.x	12324234.23
0x102C	p.y, rp.y	98678768.23
0x1028	pp	0x1012324
0x1024	ppp	0x1343521
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	1.0
0x102C	p.y, rp.y	98678768.23
0x1028	pp	0x1012324
0x1024	ppp	0x1343521
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	1.0
0x102C	p.y, rp.y	2.0
0x1028	pp	0x1012324
0x1024	ppp	0x1343521
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	1.0
0x102C	p.y, rp.y	2.0
0x1028	pp	0x1030
0x1024	ppp	0x1343521
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	1.0
0x102C	p.y, rp.y	2.0
0x1028	pp	0x1030
0x1024	ppp	0x1028
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	5.0
0x102C	p.y, rp.y	2.0
0x1028	pp	0x1030
0x1024	ppp	0x1028
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	5.0
0x102C	p.y, rp.y	3.0
0x1028	pp	0x1030
0x1024	ppp	0x1028
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Pointer

```
struct point_t{
   float x;
   float y;
}
point_t p;
point_t* pp;
point_t** ppp;
point_t& rp = p;

p.x = 1.0;
p.y = 2.0;

pp = &p;
ppp = &pp;

pp->x = 5.0;
(*ppp)->y = 3.0;
rp.x = 10;
```

Stack (vereinfacht)

0x1030	p.x, rp.x	10.0
0x102C	p.y, rp.y	3.0
0x1028	pp	0x1030
0x1024	ppp	0x1028
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Repetition: Funktionen

- Entscheide, ob mit dieser Funktionssignatur die Multiplikation (c = a*b) implementiert werden kann
- Falls „Ja“, bestimme für jede Funktionssignatur wie die Funktion aufgerufen werden kann

`void mul(int a, int b, int c);`	`//Nein, call by value`

`void mul(int& a, int b, int* c);`	`// Ja`
	`int a = 3, b = 4, c;`
	`mul(a, b, &c);`

`void mul(const int a, cont int b, const int* c);`	`// Nein, c ist Pointer auf Konstante`

`void mul(const int a, cost int b, int* const c);`	`// Ja, c ist konstanter Pointer auf`
	`// Variable`
	`int a = 3, b = 4, c;`
	`mul(a, b, &c);`

`void mul(int& a, int& b, int& c);`	`// Ja`
	`int a = 3, b = 4, c;`
	`mul(a, b, c);`

Repetition: Verkettete Listen

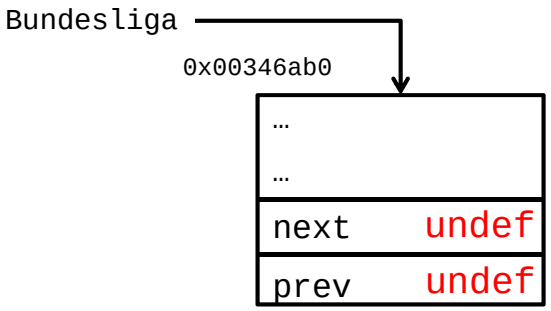
```
int main() {
    Mannschaft* Bundesliga = new Mannschaft;
    Bundesliga->Name = "FCB";
    Bundesliga->prev = NULL;
    Bundesliga->next = new Mannschaft;
    Bundesliga->next->Name = "S04";
    Bundesliga->next->prev = Bundesliga;
    Bundesliga->next->next = new Mannschaft;
    Bundesliga->next->next->Name = "Bremen";
    Bundesliga->next->next->prev = Bundesliga->next;

    Bundesliga->next->next->next = NULL;

    Mannschaft* ptr = Bundesliga->next;
    Bundesliga->next = Bundesliga->next->next;
    Bundesliga->next->prev = Bundesliga;
    ptr->next = NULL;
    ptr->prev = Bundesliga->next;
    Bundesliga->next->next = ptr;

    return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

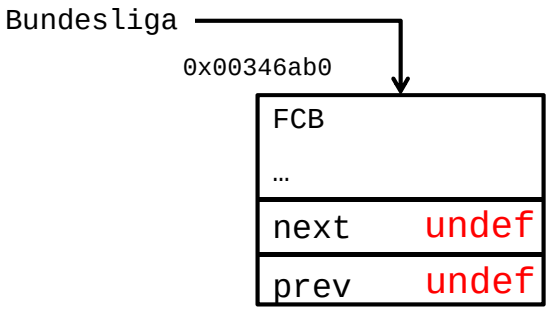
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

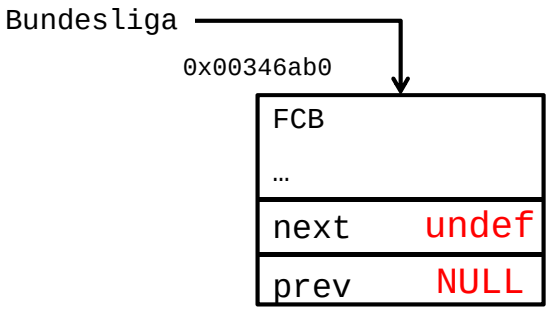
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

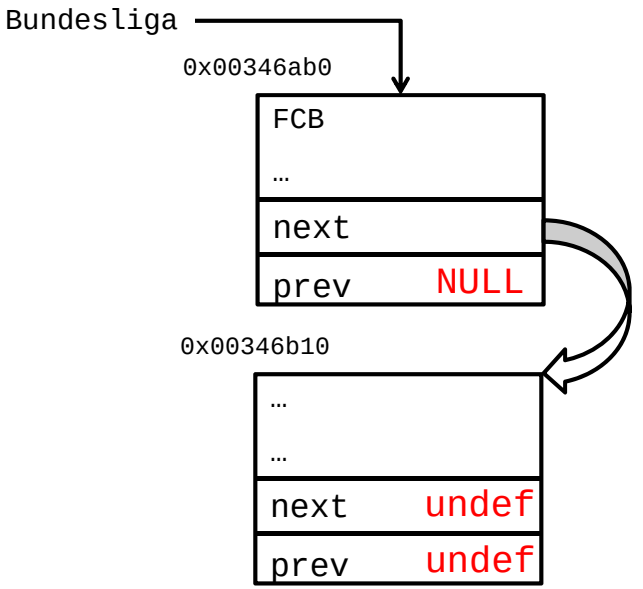
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

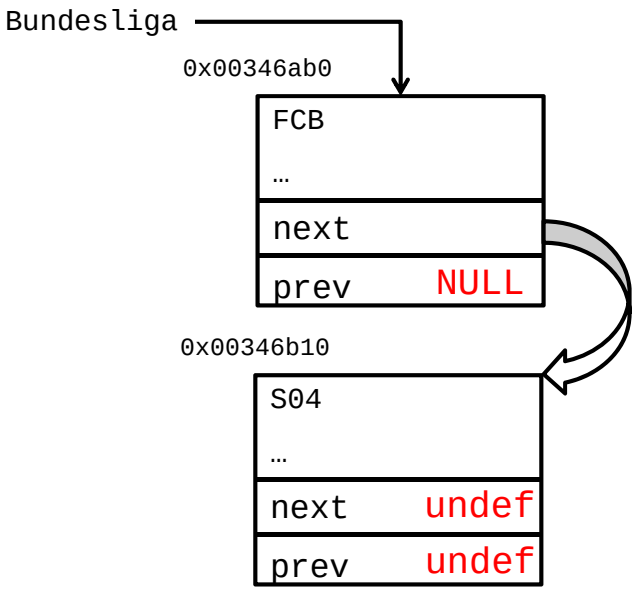
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

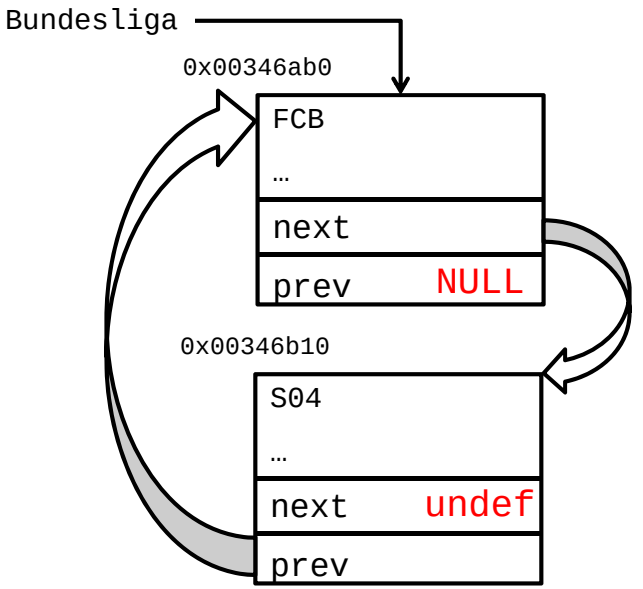
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

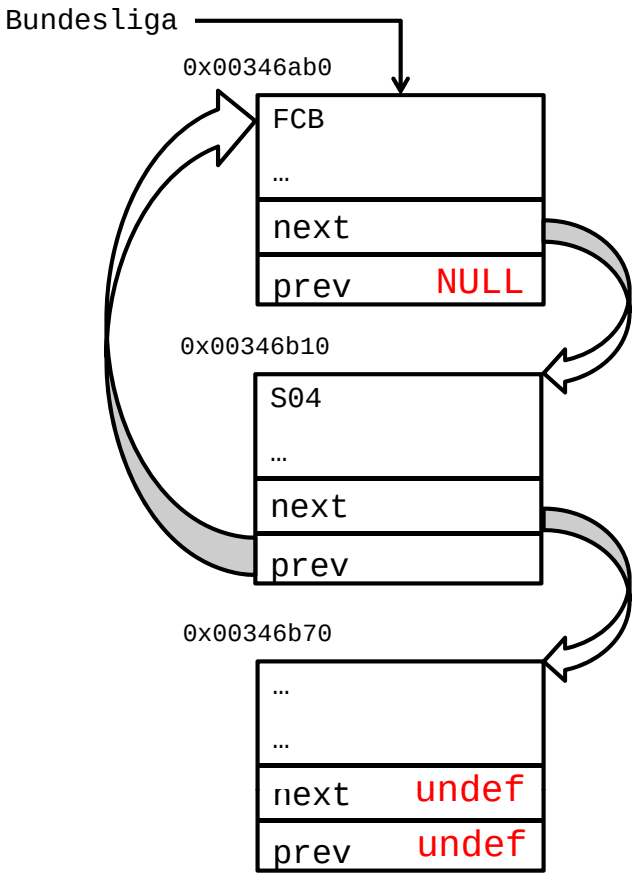
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

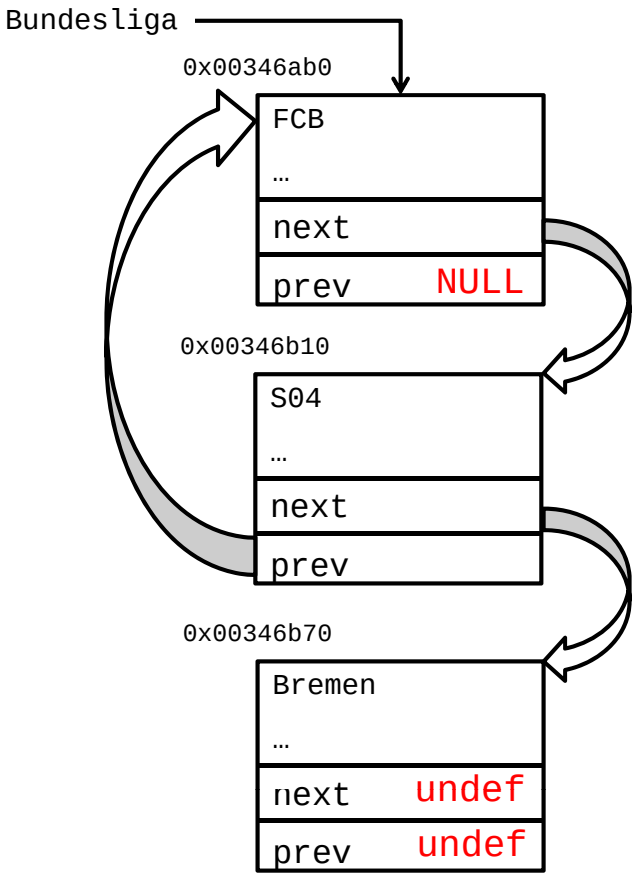
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

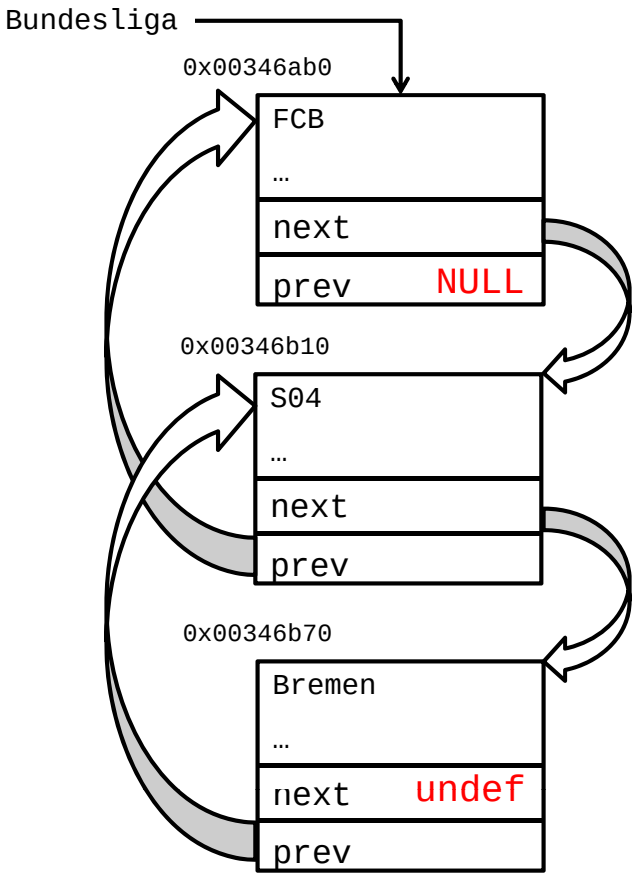
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

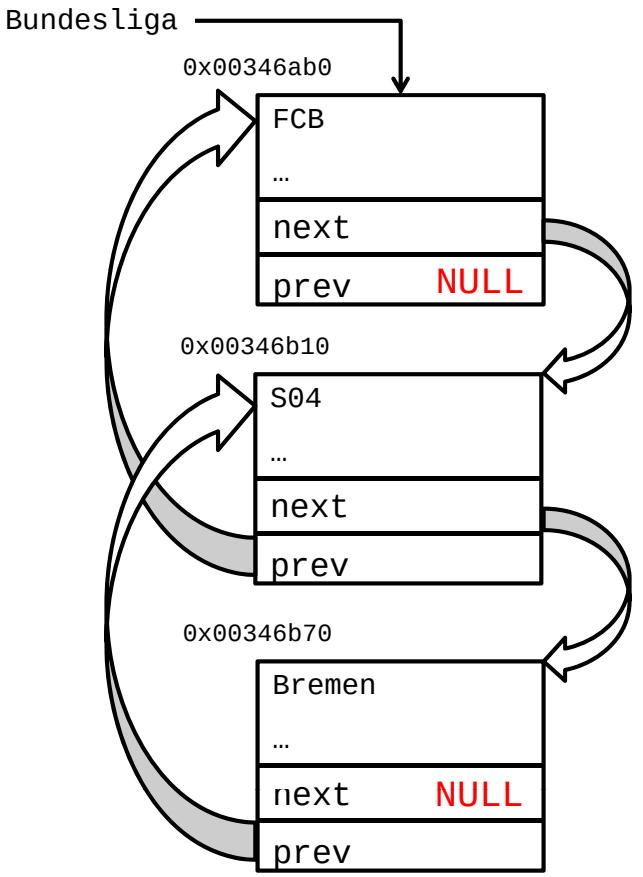
```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```

```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```



Repetition: Verkettete Listen

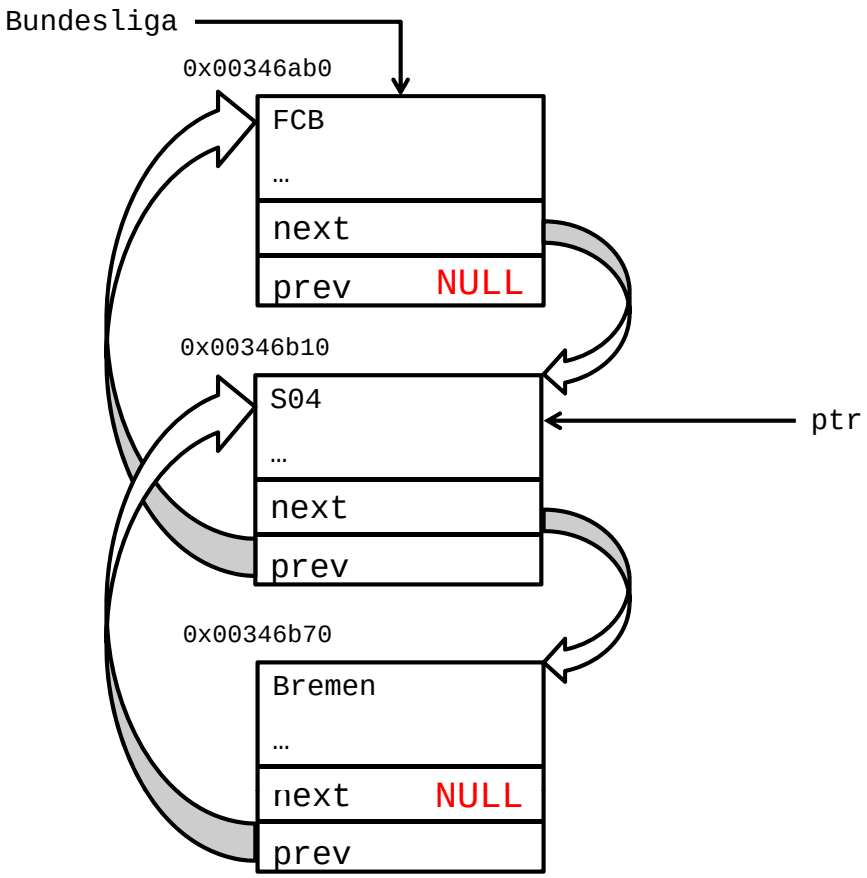
```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```

```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```



Repetition: Verkettete Listen

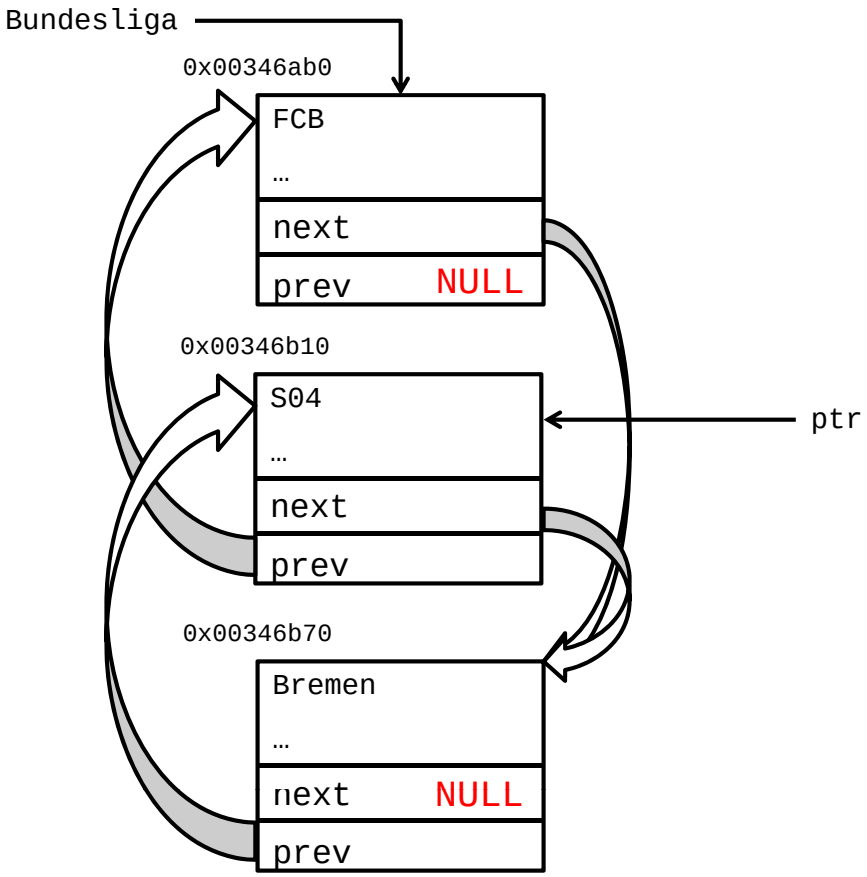
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

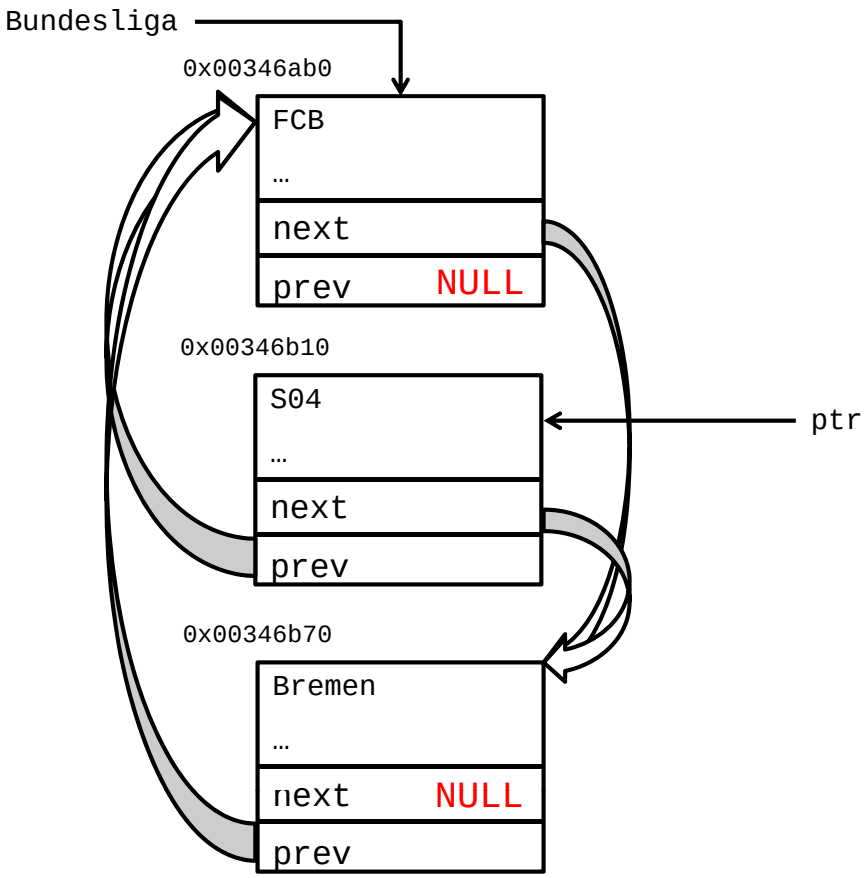
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

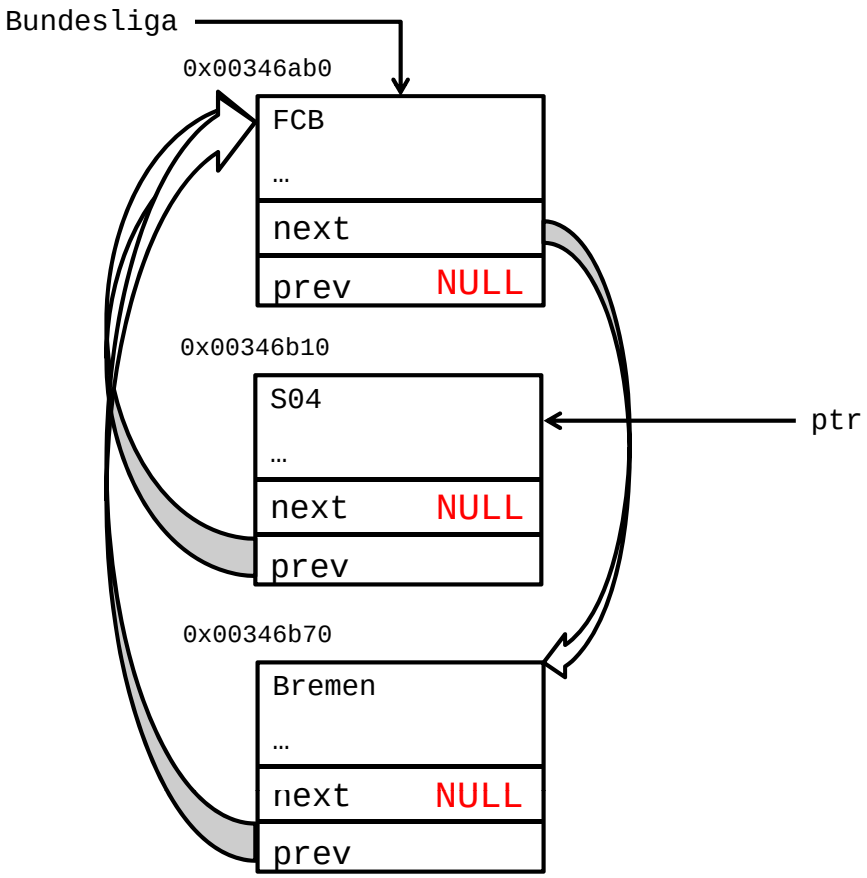
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

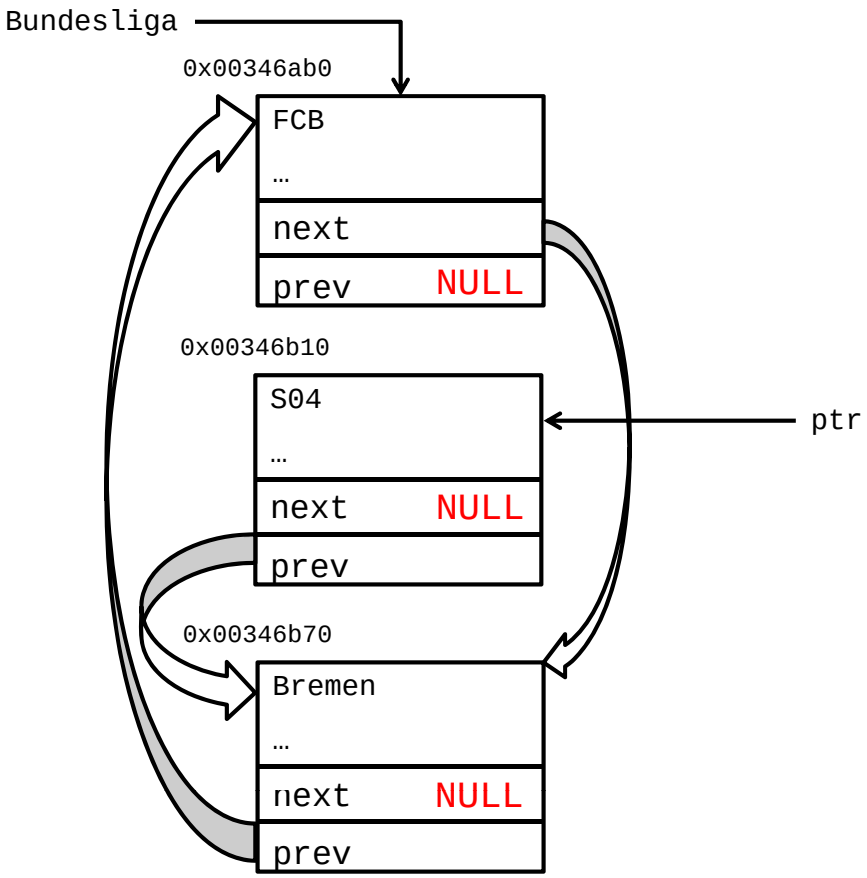
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Repetition: Verkettete Listen

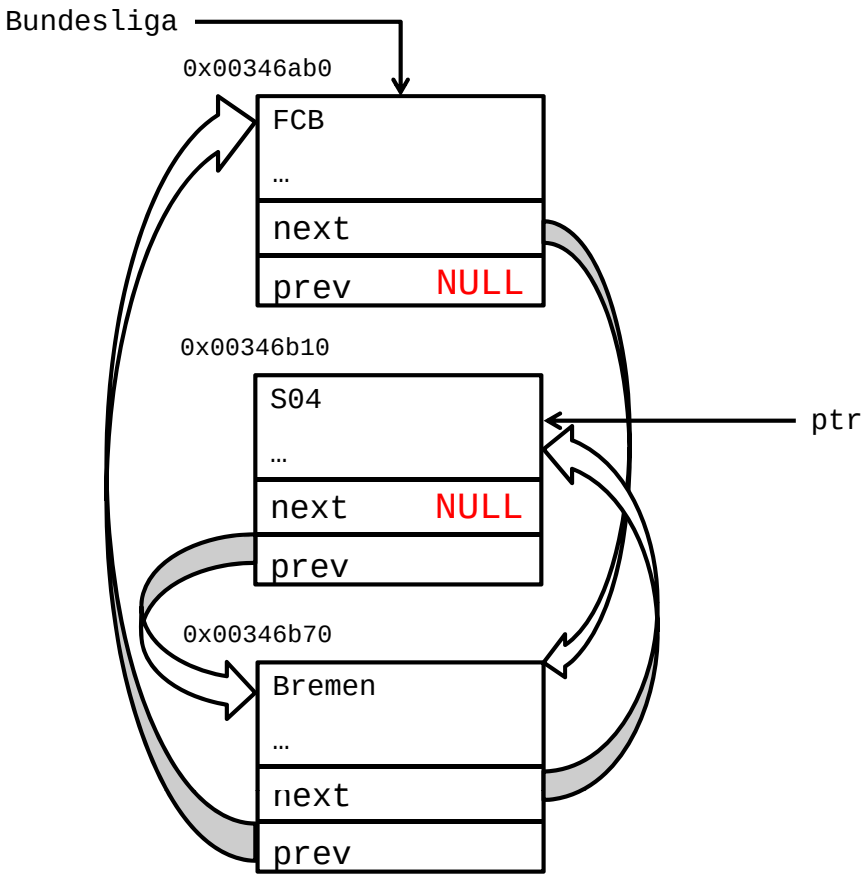
```
int main() {
        Mannschaft* Bundesliga = new Mannschaft;
        Bundesliga->Name = "FCB";
        Bundesliga->prev = NULL;
        Bundesliga->next = new Mannschaft;
        Bundesliga->next->Name = "S04";
        Bundesliga->next->prev = Bundesliga;
        Bundesliga->next->next = new Mannschaft;
        Bundesliga->next->next->Name = "Bremen";
        Bundesliga->next->next->prev = Bundesliga->next;

        Bundesliga->next->next->next = NULL;

        Mannschaft* ptr = Bundesliga->next;
        Bundesliga->next = Bundesliga->next->next;
        Bundesliga->next->prev = Bundesliga;
        ptr->next = NULL;
        ptr->prev = Bundesliga->next;
        Bundesliga->next->next = ptr;

        return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    …
    Mannschaft* next;
    Mannschaft* prev;
};
```



Nachbesprechung

- Nachbesprechung Übung 6
- Pointer/Referenzen und Funktionen
- Rekursion
- Vorbesprechung Übung 8

Rekursion

- Rekursion: Funktion die sich selbst aufruft

```
int main()
{
   int a = sum(3);
   // a = 6
   return 0;
}
```

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

- Abbruchbedingung –> endet Rekursion
- Für jeden Funktionsaufruf wird ein neuer Satz lokaler Variablen angelegt

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

0x1048		
0x1044		
0x1040		
0x103C		
0x1038		
0x1034		
0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

0x1048	b	3
0x1044		
0x1040		
0x103C		
0x1038		
0x1034		
0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

0x1048	b	3
0x1044	a	0
0x1040		
0x103C		
0x1038		
0x1034		
0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038		
	0x1034		
	0x1030		
	0x102C		
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038		
	0x1034		
	0x1030		
	0x102C		
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
    if(b > 0)
        return b + sum(b-1); //Rekursion
    else
        return 0; //Abbruchbedingung
}
```

```
int main()
{
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030		
	0x102C		
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030		
	0x102C		
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
  if(b > 0)
    return b + sum(b-1); //Rekursion
  else
    return 0; //Abbruchbedingung
}
```

```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030	return	0x1262209
sum	0x102C	b	1
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

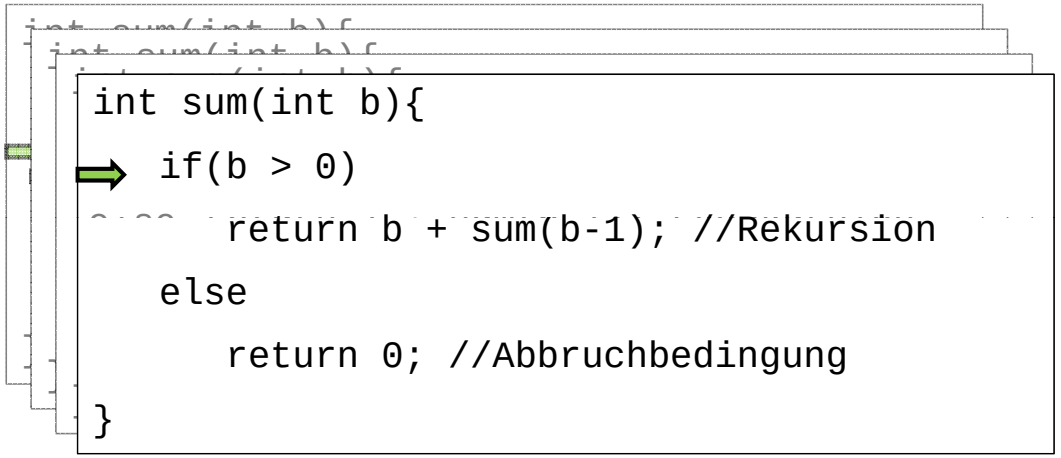
```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030	return	0x1262209
sum	0x102C	b	1
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:



```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

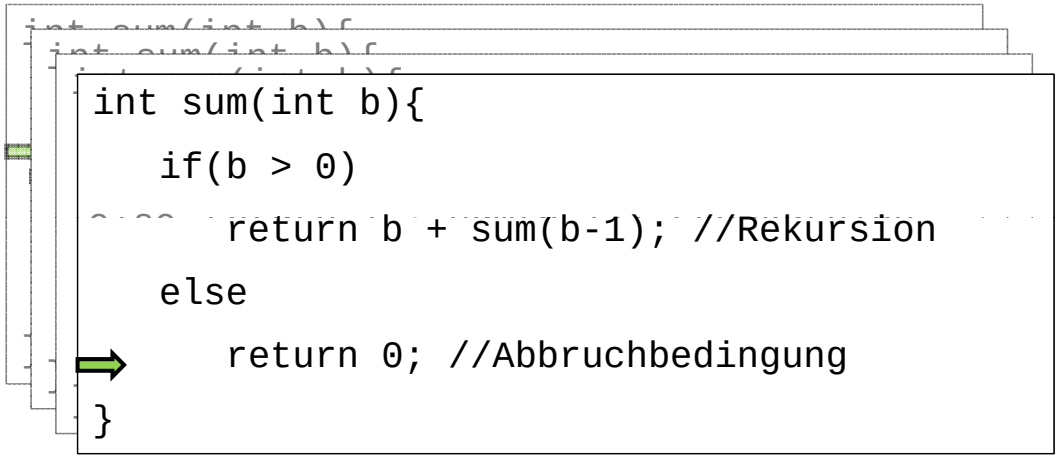
```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030	return	0x1262209
sum	0x102C	b	1
	0x1028	return	0x1344209
sum	0x1024	b	0
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:



```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030	return	0x1262209
sum	0x102C	b	1
	0x1028	return	0
sum	0x1024	b	0
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

0

1

```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	0x4562209
sum	0x1034	b	2
	0x1030	return	1
sum	0x102C	b	1
	0x1028	return	0
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

1

2

```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	0x4564789
sum	0x103C	b	3
	0x1038	return	3
sum	0x1034	b	2
	0x1030	return	1
	0x102C		
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}
```

(3 = b; 3 = sum(b-1))

```
int main()
{
   int b = 3;
   int a = 0;
   a = sum(b);
   // a = 6
   return 0;
}
```

Stack (vereinfacht)

main	0x1048	b	3
	0x1044	a	0
	0x1040	return	6
sum	0x103C	b	3
	0x1038	return	3
	0x1034		
	0x1030		
	0x102C		
	0x1028		
	0x1024		
	0x1020		
	0x1020		
	0x100C		
	0x1008		
	0x1004		
	0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return 1 + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

main

0x1048	b	3
0x1044	a	6
0x1040	return	6
0x103C		
0x1038		
0x1034		
0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return 1 + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

main

0x1048	b	3
0x1044	a	6
0x1040	return	6
0x103C		
0x1038		
0x1034		
0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Rekursion II

- Beispiel:

```
int sum(int b){
   if(b > 0)
      return b + sum(b-1); //Rekursion
   else
      return 0; //Abbruchbedingung
}


 int main()
 {
    int b = 3;
    int a = 0;
    a = sum(b);
    // a = 6
    return 0;
 }
```

Stack (vereinfacht)

0x1048		
0x1044		
0x1040		
0x103C		
0x1038		
0x1034		
0x1030		
0x102C		
0x1028		
0x1024		
0x1020		
0x1020		
0x100C		
0x1008		
0x1004		
0x1000		

Rekursion

- Beispiel II

```
void f(int a)
{
   cout << a << endl;
   if( a > 0)
      f(a-1);
}


int main()
{
  f(4);
  return 0;
}
```

Output:

4, 3, 2, 1, 0

```
void f(int a)
{
   if( a > 0)
      f(a-1);
   cout << a << endl;
}


int main()
{
  f(4);
  return 0;
}
```

Output:

0, 1, 2, 3, 4

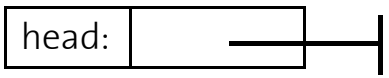
Nachbesprechung

- Nachbesprechung Übung 6
- Pointer/Referenzen und Funktionen
- Rekursion
- Vorbesprechung Übung 8

Übung 8: Aufgabe 1 – FIFO-Liste

- FIFO – Listen (First in, First out)
  - Element welches als erstes eingefügt wurde wird als erstes wieder entfernt
  - Funktionen
    - void enter(int a) – Fügt Element am Listenende ein
    - int leave() – Entfernt Element am Listenkopf

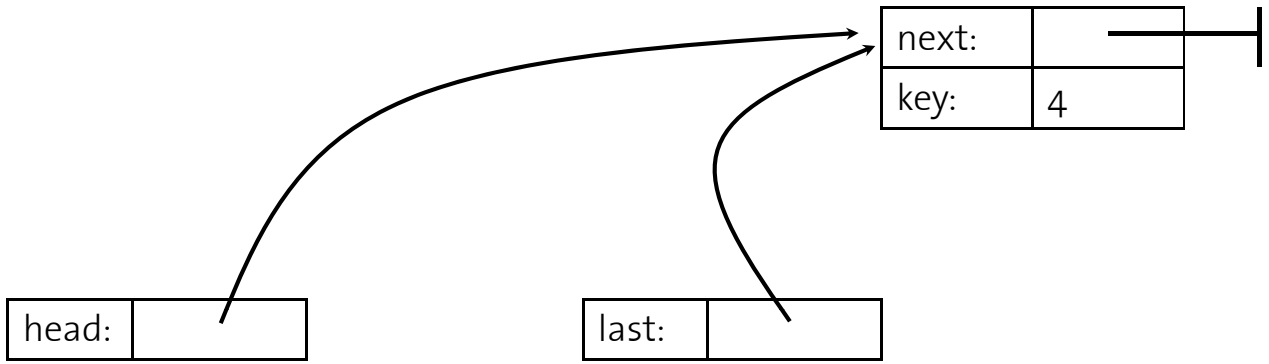
- Beispiel: enter(4), enter(11), leave(), leave()



Übung 8: Aufgabe 1 – FIFO-Liste

- FIFO – Listen (First in, First out)
  - Element welches als erstes eingefügt wurde wird als erstes wieder entfernt
  - Funktionen
    - void enter(int a) – Fügt Element am Listenende ein
    - int leave() – Entfernt Element am Listenkopf

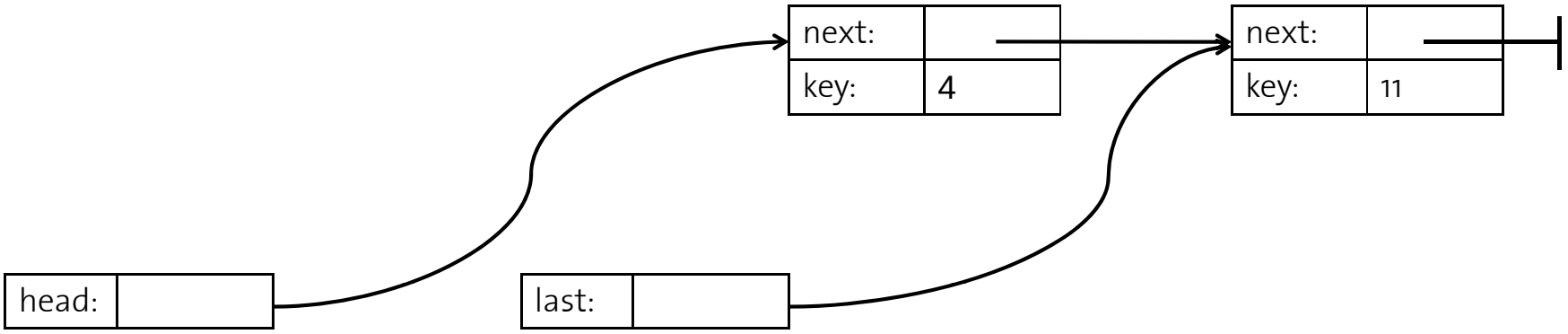
- Beispiel: enter(4), enter(11), leave(), leave()



Übung 8: Aufgabe 1 – FIFO-Liste

- FIFO – Listen (First in, First out)
  - Element welches als erstes eingefügt wurde wird als erstes wieder entfernt
  - Funktionen
    - void enter(int a) – Fügt Element am Listenende ein
    - int leave() – Entfernt Element am Listenkopf

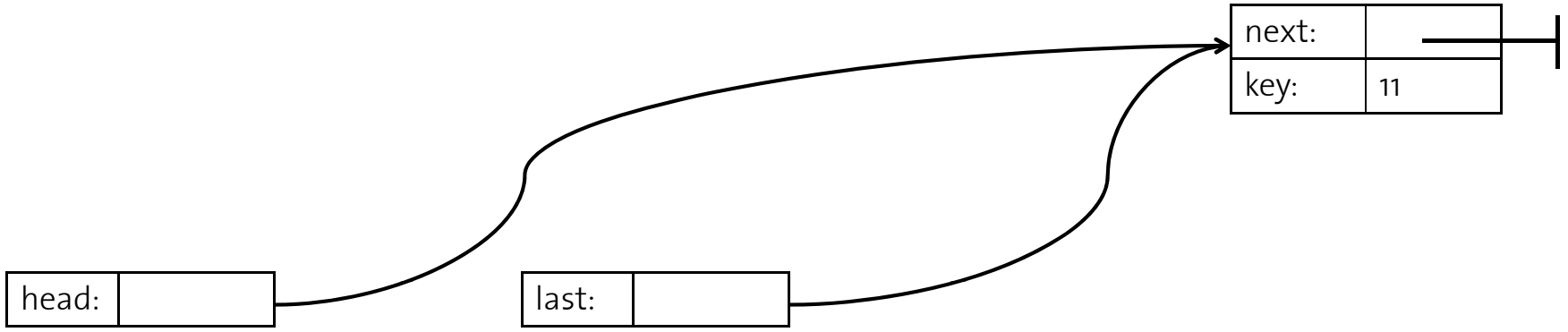
- Beispiel: enter(4), enter(11), leave(), leave()



Übung 8: Aufgabe 1 – FIFO-Liste

- FIFO – Listen (First in, First out)
  - Element welches als erstes eingefügt wurde wird als erstes wieder entfernt
  - Funktionen
    - void enter(int a) – Fügt Element am Listenende ein
    - int leave() – Entfernt Element am Listenkopf

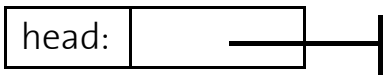
- Beispiel: enter(4), enter(11), leave(), leave()



Übung 8: Aufgabe 1 – FIFO-Liste

- FIFO – Listen (First in, First out)
  - Element welches als erstes eingefügt wurde wird als erstes wieder entfernt
  - Funktionen
    - void enter(int a) – Fügt Element am Listenende ein
    - int leave() – Entfernt Element am Listenkopf

- Beispiel: enter(4), enter(11), leave(), leave()



Übung 8: Aufgabe 2 - Stack

- Datenstruktur

```
struct stack_t{
   double* array;
   int numberOfElements;
   int sizeOfArray;
};
```

- Stack Funktionen:
  - `stack_t init()`
    - Initialisiert den Stack
  - `double pop(stack_t& stack)`
    - Die Anzahl Elemente auf dem Stack muss > 0 sein, sonst gib Fehler aus
    - Entferne letztes Element und gib es als Rückgabewert zurück

Übung 8: Aufgabe 2 - Stack

- Stack Funktionen:
  - `void push(stack_t& stack, double element)`
    - Ist das Array voll (stack.numberOfElements == stack.sizeOfArray), dann alloziere ein neues Array doppelter Grösse
    - Ersetze das alte Array mit dem neuen Array (kopiere alle Elemente)
    - Füge das neue Element dem Stack hinzu.
  - `int size(const stack_t& stack)`
    - Gib die aktuelle Grösse des Stacks zurück (numberOfElements)
  - `void clear(stack_t& stack)`
    - Räumt die Datenstruktur auf (dealloziere das Array)

Übung 8: Aufgabe 3 – UPN Taschenrechner

- UPN: Umgekehrte Polnische Notation

```
2*(4+3*1)          Vs.        2 4 3 1 * + *
```

  - Vorteil: Keine Klammern notwendig, vereinfacht das Parsen

- Algorithmus:
  - Eingabe einlesen:

```
while(cin >> input)
```

  - Überprüfe die Eingabe:
    - "+, –,*,/": Hole zwei Operanden vom Stack und lege das Ergebnis wieder auf den Stack

```
if(strcmp(intput, "+") == 0){
    opA = pop(stack);
    opB = pop(stack);
    push(stack, opA + opB);
}
```

Übung 8: Aufgabe 3 – UPN Taschenrechner

- Überprüfe die Eingabe (cont):
  - Operand (0.5, 1, 5, ...)
    - Konvertiere Operand (char Array) zu `float` und lege das Ergebnis auf den Stack

```
push(stack, atof(input))
```

  - ";" : Ende des Terms erreicht
    - Sind mehr als ein Element auf dem Stack -> Fehler
    - Ausgabe des Resultats (letztes Element auf dem Stack)

```
if (size(stack) != 1)
    cout << "Invalid expression!" << endl;
else
    cout << "Result: " << pop(stack) << endl;
```