



Hochschule  
Bonn-Rhein-Sieg  
University of Applied Sciences

# Zeiger und Speicherverwaltung in C

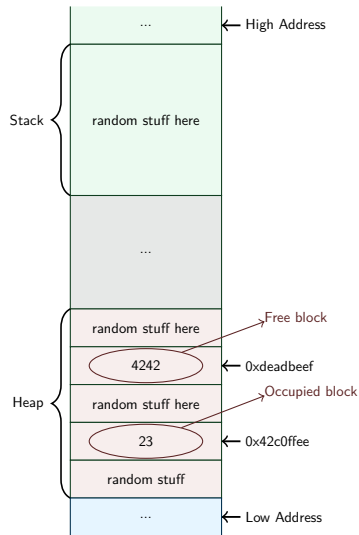
---

Michael Welle  
michael.welle@h-brs.de  
18. Juni 2019

# Ausgangszustand

- ▶ Stack und Heap sind mit zufälligen Daten gefüllt
- ▶ Block an Adresse 0x42c0ffee auf dem Heap sei belegt (Wert: 23)

```
int main() {  
    ...  
}
```

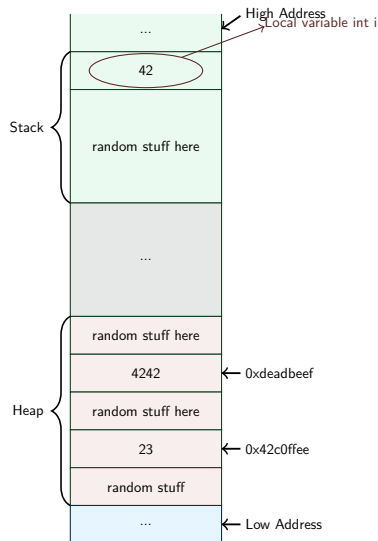


# Lokale Variable auf dem Stack anlegen I

- Typ: int
- Wert: 42

```
int main() {  
    int i = 42;  
  
    printf("i: %d\n", i);  
}
```

- Ausgabe: i: 42

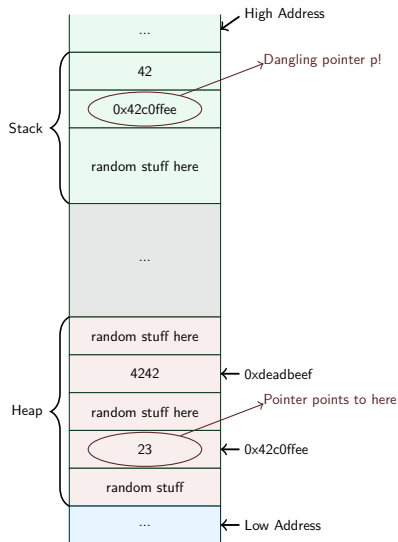


# Lokale Variable auf dem Stack anlegen II

- ▶ Typ: pointer to int
- ▶ Wert: Adresse zufällig
- ▶ Obacht: 'Dangling Pointer'

```
int main() {  
    int i = 42;  
    int *p;  
  
    printf("p: %p *p: %d\n",  
          p, *p);  
}
```

- ▶ Ausgabe: p: 0x42coffee \*p: 23

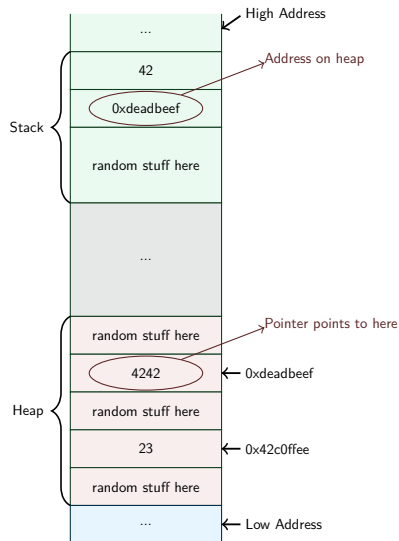


# Speicher allozieren

- ▶ `malloc()` alloziert Speicher
  - ▶ Rückgabewert ist bei Erfolg Adresse des Speicherblocks
  - ▶ Adresse liegt auf dem Heap

```
int main() {  
    int i = 42;  
    int *p;  
  
    p = malloc(sizeof(int));  
    printf("p: %p *p: %d\n",  
          p, *p);  
}
```

- ▶ Ausgabe: p: 0xdeadbeef \*p: 4242

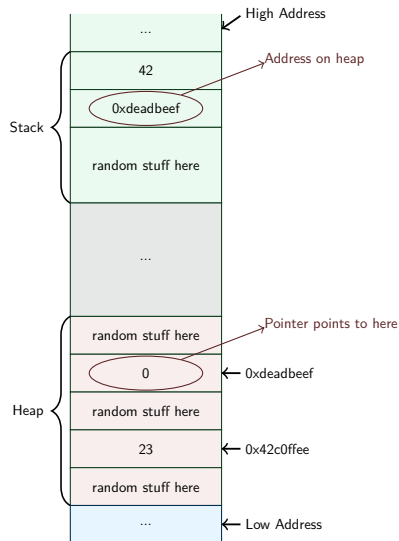


# Wert schreiben

- Zeiger dereferenzieren: \*p

```
int main() {  
    int i = 42;  
    int *p;  
  
    p = malloc(sizeof(int));  
    *p = 0;  
    printf("p: %p *p: %d\n",  
           p, *p);  
}
```

- Ausgabe: p: 0xdeadbeef \*p: 0



# Speicher freigeben

- ▶ `free()` ist inverse Operation zu `malloc()`
- ▶ Freigegebene Zeiger verweisen auf zufällige Werte

```
int main() {  
    int i = 42; int *p;  
    p = malloc(sizeof(int));  
    *p = 0;  
    free(p);  
    printf("p: %p *p: %d\n",  
          p, *p);  
}
```

- ▶ Ausgabe: p: 0xdeadbeef \*p: 0

