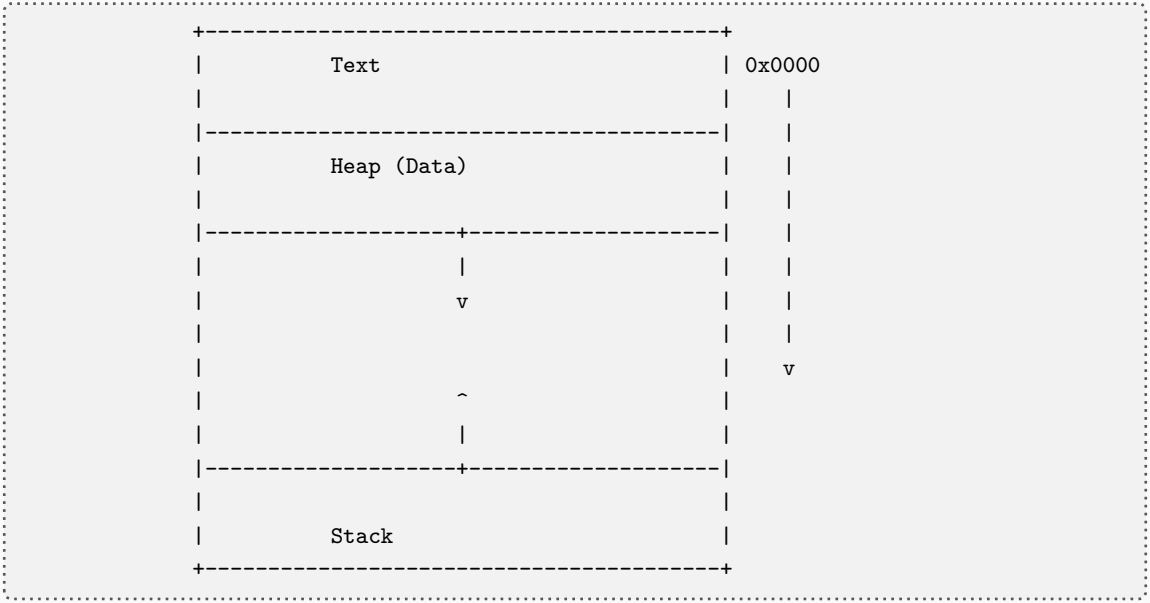
# C++: Pointer und Referenzen

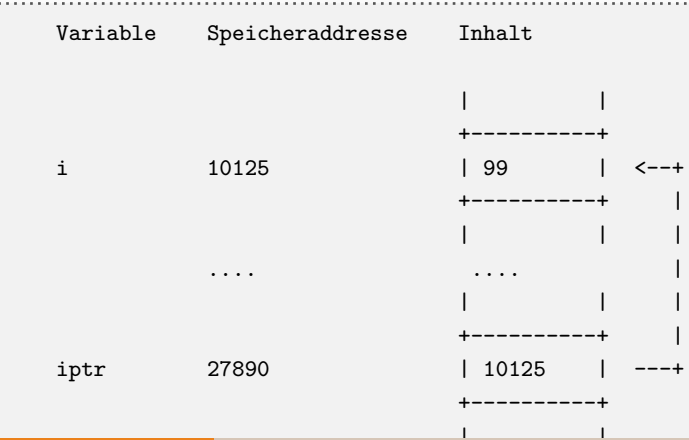
Carsten Gips (HSBI)

# Virtueller Speicher



# Konzept eines Pointers

```c
int i = 99;
int *iptr;

iptr = &i;  /* Wert von iptr ist gleich Adresse von i */
*iptr = 2;  /* Deferenzierung von iptr => Veränderung von i */
```



# Speicherverwaltung

- C: **Funktionen** zur Verwaltung dynamischen Speichers: `malloc()`, `free()`, ... (in `<stdlib.h>`)

```c
int *p = (int*) malloc(sizeof(int));
int *pa = (int*) malloc(4*sizeof(int));

free(p);
free(pa);
```

- C++: **Operatoren**: `new` und `delete`

```c
int *p = new int;
int *pa = new int[4];

delete p;
delete [] pa;
```

# Pointer und Arrays

Ein Array-Name ist wie ein *konstanter* Pointer auf Array-Anfang: `a[i] == *(a+i)`

```
char a[6], c, *cp;

&a[0] == a;
cp = a;

c = a[5];
c = *(a+5);
c = *(cp+5);
c = cp[5];

a = cp;  /* FEHLER */
a = &c;  /* FEHLER */
```

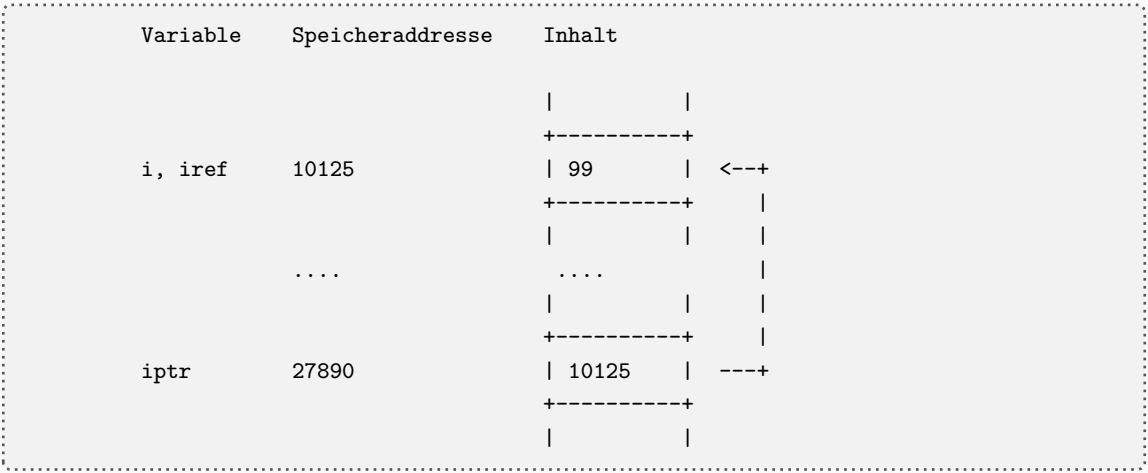
# Referenzen in C++

```
Typ & Name = Objekt;
```

```
int i=2;
int j=9;

int &r=i;     // Referenz: neuer Name fuer i
r=10;         // aendert i: i==10
r=j;          // aendert i: i==9

int &s=r;     // aequivalent zu int &s = i;
```

# Referenzen bilden Alias-Namen

```
int i = 99;
int *iptr = &i;

int &iref = i;    // Referenz: neuer Name fuer i
```



# Call-by-Reference Semantik in C++

```
void add_5_ptr(int *x) { *x += 5; }
void add_5_ref(int &x) {  x += 5; }

int main() {
    int i=32;

    add_5_ptr(&i);
    add_5_ref( i);
}
```

# Rückgabe von Werten per Referenz

```
int &fkt1(const int &, const char *);
int *fkt2(const int &, const char *);
```

# Rückgabe von Werten per Referenz

```
int &fkt1(const int &, const char *);
int *fkt2(const int &, const char *);
```

- Vorsicht mit lokalen Variablen (Gültigkeit)!

```
int &fkt1(const int &i, const char *j) {
    int erg = i+1;
    return erg;   // Referenz auf lokale Variable!
}
int *fkt2(const int &i, const char *j) {
    int erg = i+2;
    return &erg;  // Pointer auf lokale Variable!
}
int main() {
    int &x = fkt1(2, "a");  // AUTSCH!!!
    int *y = fkt2(2, "b");  // AUTSCH!!!
    int  z = fkt1(2, "c");  // OK
}
```

# Wrap-Up

- Virtueller Speicher: Segmente: Text, Stack, Heap
- Pointer sind Variablen, deren **Wert als Adresse** interpretiert wird
  - Deklaration mit `*` zwischen Typ und Name
  - Adressoperator `&` liefert die Adresse eines Objekts
  - Dereferenzierung eines Pointers mit `*` vor dem Namen
- Array-Name ist konstanter Pointer auf Array-Anfang
- Pointer haben Typ: Pointerarithmetik berücksichtigt Speicherbreite des Typs
- C++-Referenzen als Alias-Namen für ein Objekt
  - Deklaration: `Typ &ref = obj;`
  - Fest mit Objekt verbunden
  - Zugriff auf Referenz: Direkter Zugriff auf das Objekt

# LICENSE