Zeiger

Benjamin Kuffel

Universität Hamburg

Proseminar „C- Grundlagen und Konzepte“

Inhalt

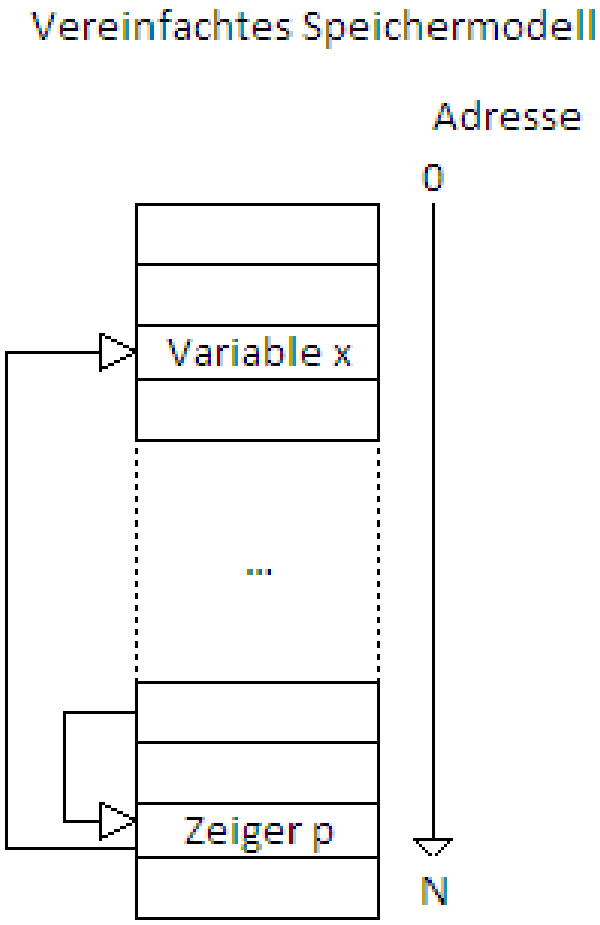
- Zeiger
- Zeigerdeklaration und –operationen
- Adressoperator
- Zeigerarithmetik
- *Type cast* mit Zeigern
- Wozu dienen Zeiger?
- Heap-Speicher
- Arrays
- Arrays und Zeiger
- void-Zeiger und Null-Zeiger
- Funktionszeiger
- Unterschied zwischen Call by Value und Call by Reference

Zeiger

- Alle Programm-Objekte liegen im Speicher

- Jedes Objekt besitzt eine Adresse im Speicher (wie eine Hausnummer)

- Häufig werden Objekte durch Namen zugegriffen (z.B. Variable „x“)

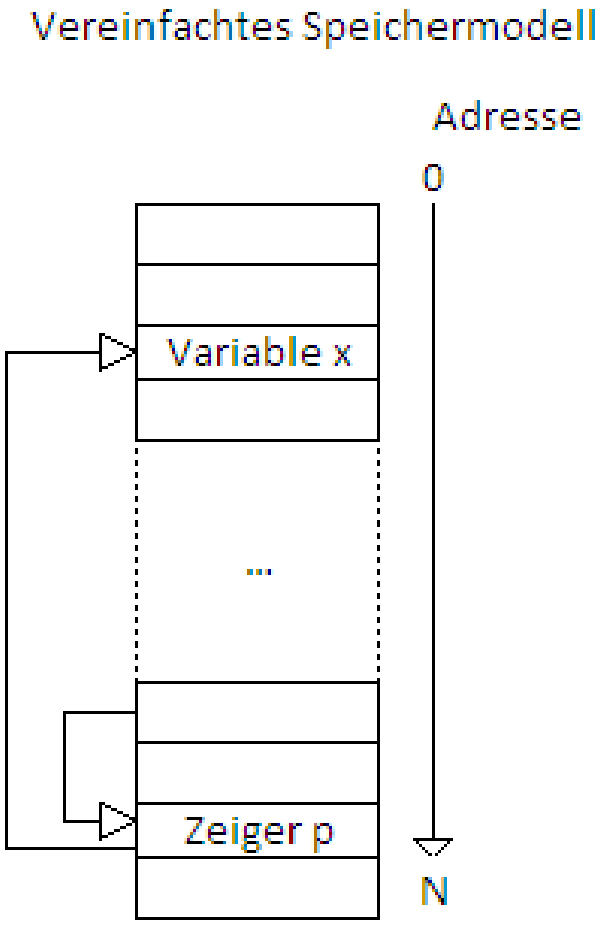


Zeiger

- Oft ist auch der indirekte Zugriff auf ein Objekt über seine Adresse sinnvoll

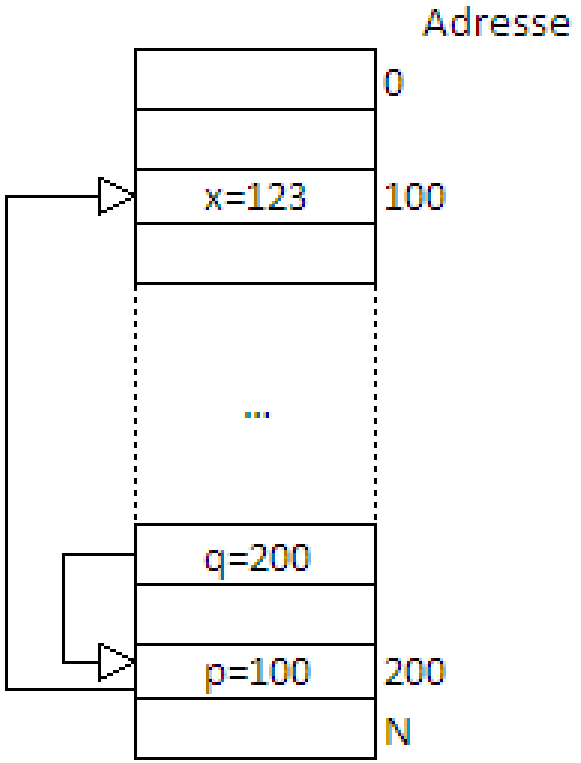
- Objekt-Adressen werden in Zeigern gespeichert

- Zeiger sind selbst Objekte mit Adressen, daher sind auch Zeiger auf Zeiger möglich



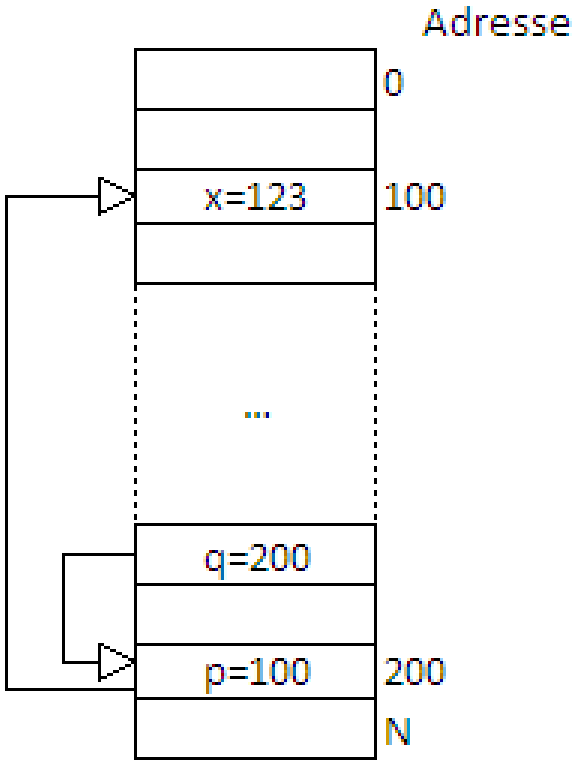
Zeigerdeklaration und -operationen

- Zeiger werden mit „*“ deklariert
- int *p;
- int **q; (wie (int*)* q;)
- int *p, q; //int-Zeiger p und int-Variable q
- int *p,*q; //zwei int-Zeiger p und q



Zeigerdeklaration und -operationen

- Der Zugriff auf x erfolgt mit „*“ Operator
- *p liefert den Wert 123 (int)
- *q liefert den Wert 100 (intptr)



Adressoperator

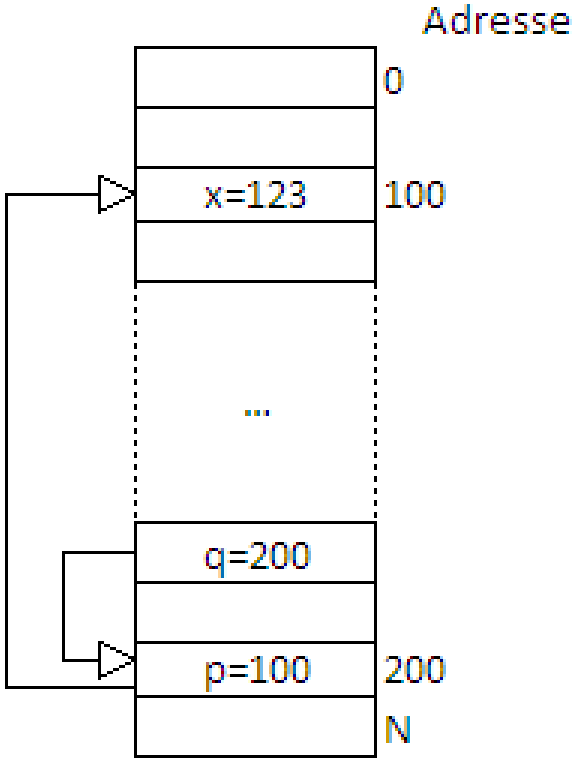
- Erzeugung von Referenzen durch den Adressoperator „&“

- p = &x;

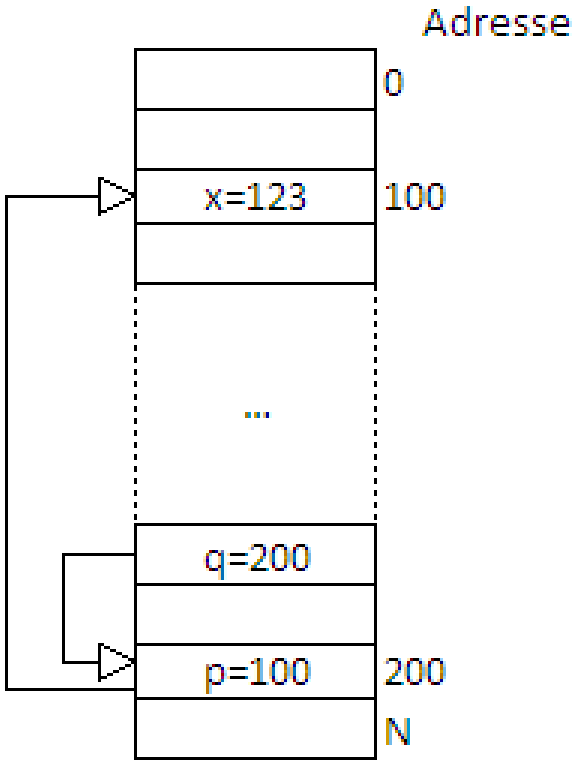
  // p wird die Adresse von x zugewiesen, „zeigt“ auf x

- q = &p;

  // „zeigt“ auf p

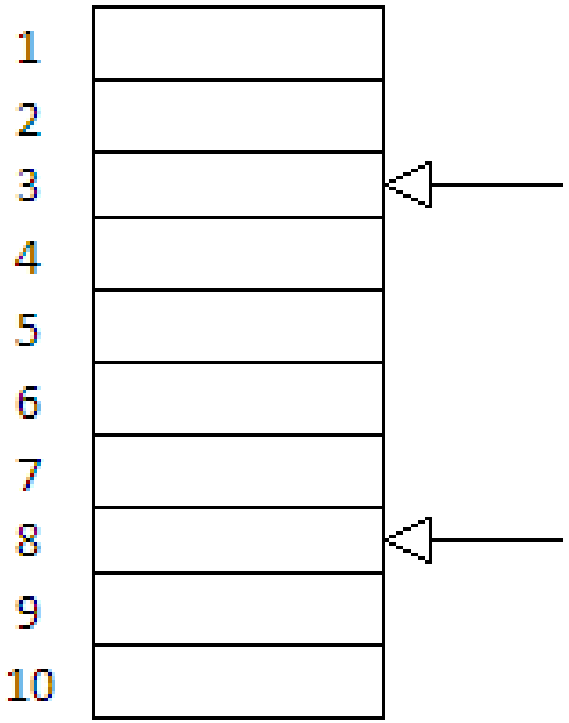


- int x; // normale int-Variable
- int *p; // Zeiger auf einen
  int-Wert im Speicher
- p = &x; // p = Adresse(x),
  d.h. „p auf x zeigen lassen“



Zeigerarithmetik

- *p = p +5 //erhöht *p um 5
- *p = *p +1
- *p + = 1
- ++*p
- (*p)++

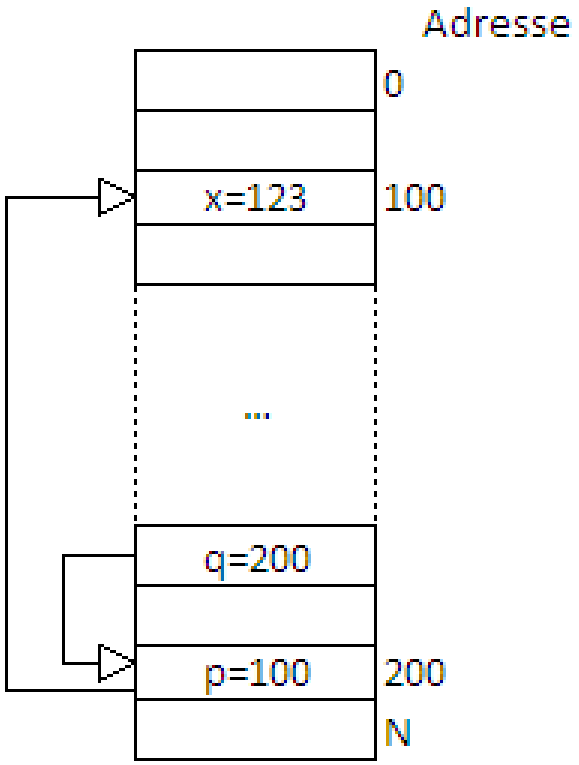


Type cast mit Zeigern

- Zeiger sind im Prinzip identisch mit (positiven) Integer-Zahlen

- Daher können Zeiger und int ineinander umgewandelt werden

- Diese Eigenschaft bringt große Flexibilität, jedoch auch Gefahren, z.B. möglicher Zugriff auf geschützte Speicherbereiche

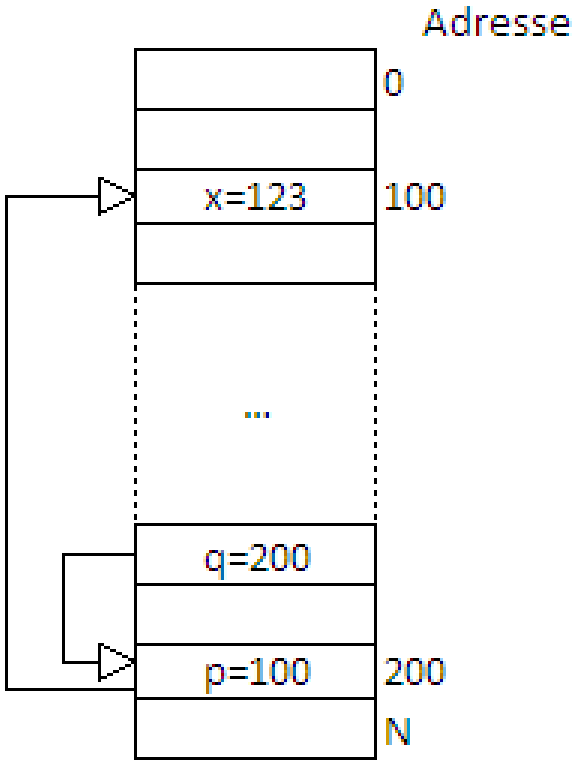


Type cast mit Zeigern

```
p = (int*)x;

y = *p;
  // lädt y mit dem Inhalt von
     Speicherstelle 123


 x = (int)q;
  // lädt x mit 200
```



Type cast mit Zeigern

- (x) y // wandelt die Variable für den
  Compiler in den Typ x
- x* // Zeiger auf Typ x
- *y // Die Speicherzelle, auf die y verweist
- x*y // y ist eine Variable des Typs Zeiger-auf-x

Wozu dienen Zeiger?

- Falls mittels eines einzigen Objektes (Zeiger) im Programmverlauf auf verschiedene andere Objekte zugegriffen werden soll

- Vor allem sinnvoll bei dynamischen Datenstrukturen

- Größe der Datenobjekte während Programmentwicklung noch nicht festgelegt

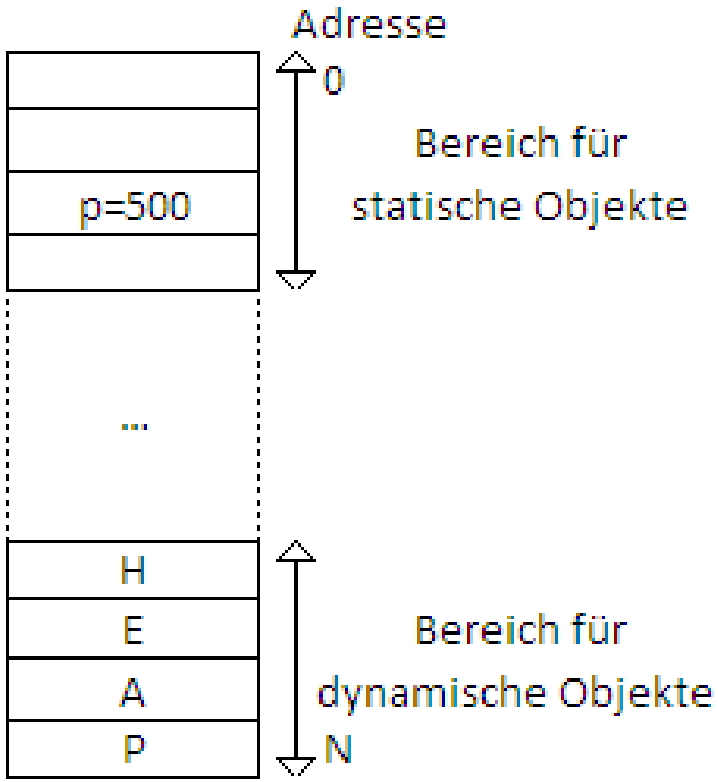
- Zeiger kann dynamisch Speicher anfordern und wieder freigeben.

Heap-Speicher

- Zugriff auf den Heap wie folgt:

```
int* p = (int*) malloc(sizeof(int));
```

- „malloc“ (*memory allocation*)



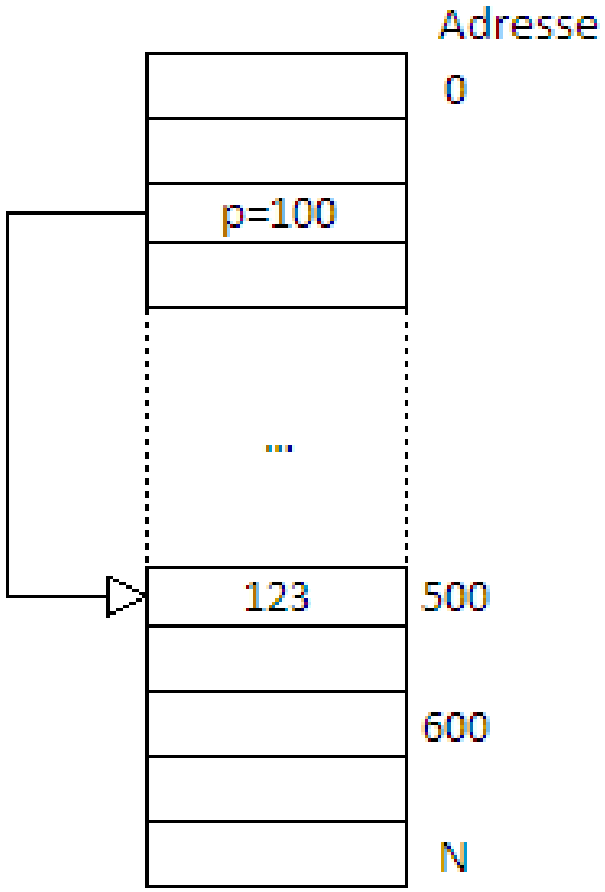
Heap-Speicher

- int*p=(int*)malloc(sizeof(100));

  //Adressen 500-599 durch *malloc* reserviert

- *p = 123;

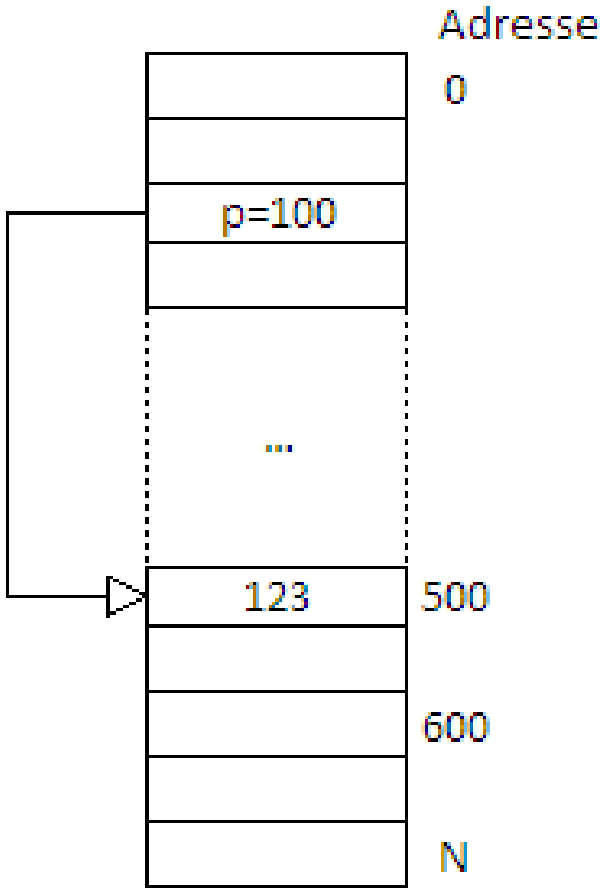
  //weist 123 an erste Speicherzelle im Block zu



Heap-Speicher

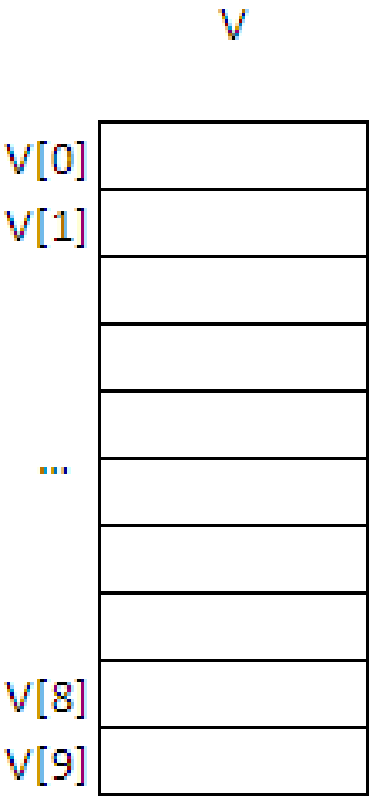
- Nach Verwendung sollte Heap-Block wieder freigegeben werden, um den Speicherplatz für andere Zwecke wiederverwenden zu können:

```
free(p);   //Block 500-599 freigeben
```



Arrays

- Die **Indizierung** beginnt bei 0.
- D.h. int V[10], nur V[0]….V[9]

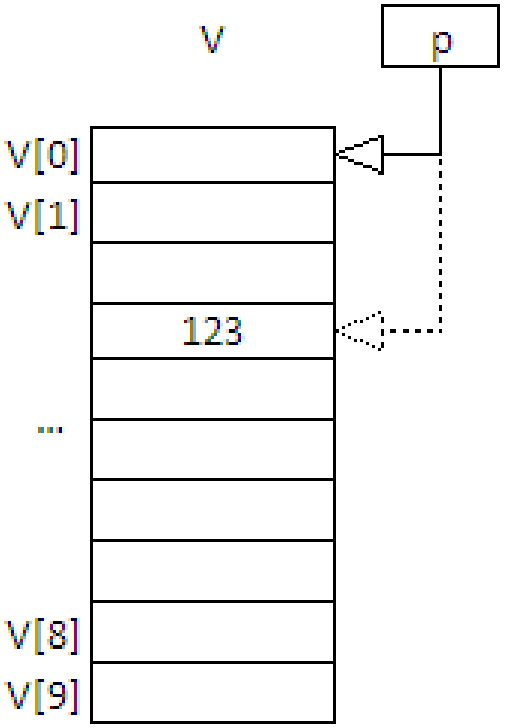


Arrays und Zeiger

- Index i bezeichnet den Abstand (*offset*) des i-ten Elementes von der Basisadresse V (= &V[0])

- Operator „[...]“ entspricht Addition von Array-Basisadresse und Index

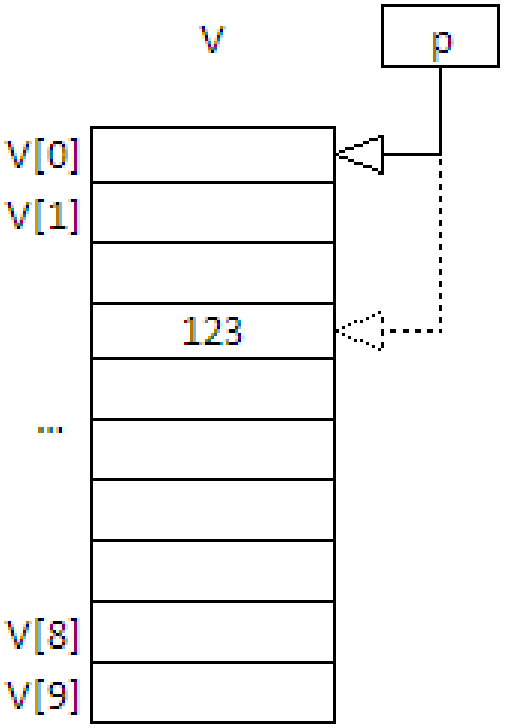
- V[i]=*(V+i)=*(i+V)



Arrays und Zeiger

- p=&V[0]; oder p=V;
  //lässt p auf Anfang von V zeigen

- V[3] = 123
- p[3] = 123
- *(p+3) = 123



void-Zeiger

void*

- untypisierter Zeiger
- Kompatibilität zu jedem Datentyp

Null-Zeiger

NULL

- ein Zeiger mit einem speziellen Wert (Nullwert)
  „Zeige auf keine gültige Adresse-Wert“

Funktionszeiger

- Zeiger können nicht nur auf Variablen, sondern auch auf Funktionen verweisen

```
int addition (int a, int b)
{
   return a+b;
}
int subtraktion (int a, int b)
{
   return a/b;
}
int main()
{
   int (*rechenoperation)(int, int) = 0; // Anlegen eines Funktionszeigers, Initialisierung mit 0
   *rechenoperation(2,5);
   rechenoperation = &addition;
   *rechenoperation(8,5);
   rechenoperation = &subtraktion
}
```

Call by Value

```
#include <stdio.h>
void func(int wert)
{
wert += 5;
printf("%i\n", wert);
}
int main()
{
int zahl = 10;
printf("%i\n", zahl);
func(zahl);
printf("%i\n", zahl);
return 0;
}
```

10 15 10

Call by Reference

```
#include <stdio.h>
void swap(int *x, int *y)
{
int tmp;
tmp = *x;
*x = *y;
*y = tmp;
}
int main()
{
int x = 2, y = 5;
printf("Variable x: %i, Variable y: %i\n", x, y);
swap(&x, &y);
printf("Variable x: %i, Variable y: %i\n", x, y);
return 0;
}
```

Variable x: 2, Variable y: 5 Variable x: 5, Variable y: 2

Zusammenfassung

- Zeiger, Zeigerdeklaration und –operationen, Zeigerarithmetik
- Adressoperator
- *Type cast* mit Zeigern
- Wozu dienen Zeiger?
- Heap-Speicher
- Arrays und Zeiger
- void-Zeiger und Null-Zeiger
- Funktionszeiger
- Unterschied zwischen Call by Value und Call by Reference

Quelle

- www.wikipedia.de
- www.wikibooks.org
- Buch Programmieren in C, Verlag Hanser
- Buch C von A bis Z, Verlag Galileo Computing