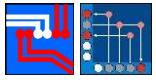
Datenstrukturen in C/C++

Zeiger

Prof. Dr. Roland Dietrich



Programmieren mit Zeigern

- Definition von Zeigervariablen

  *Bezugstyp *Variable ;*

- Dereferenzieren

  **Variable = x ;* *(x Variable des Bezugstyps)*

  *x = *Variable ;*

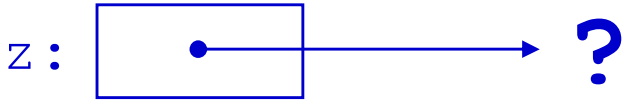
```
int *z;
```

→ z zeigt „irgendwo“ hin

→ z Dereferenzieren von z ist „gefährlich“.

Situation im Speicher:

Stack
(automatisch verwaltet)



Programmieren mit Zeigern

- Speicherplatz anfordern auf dem **„*Heap*“** (→ Operator **`new`**)

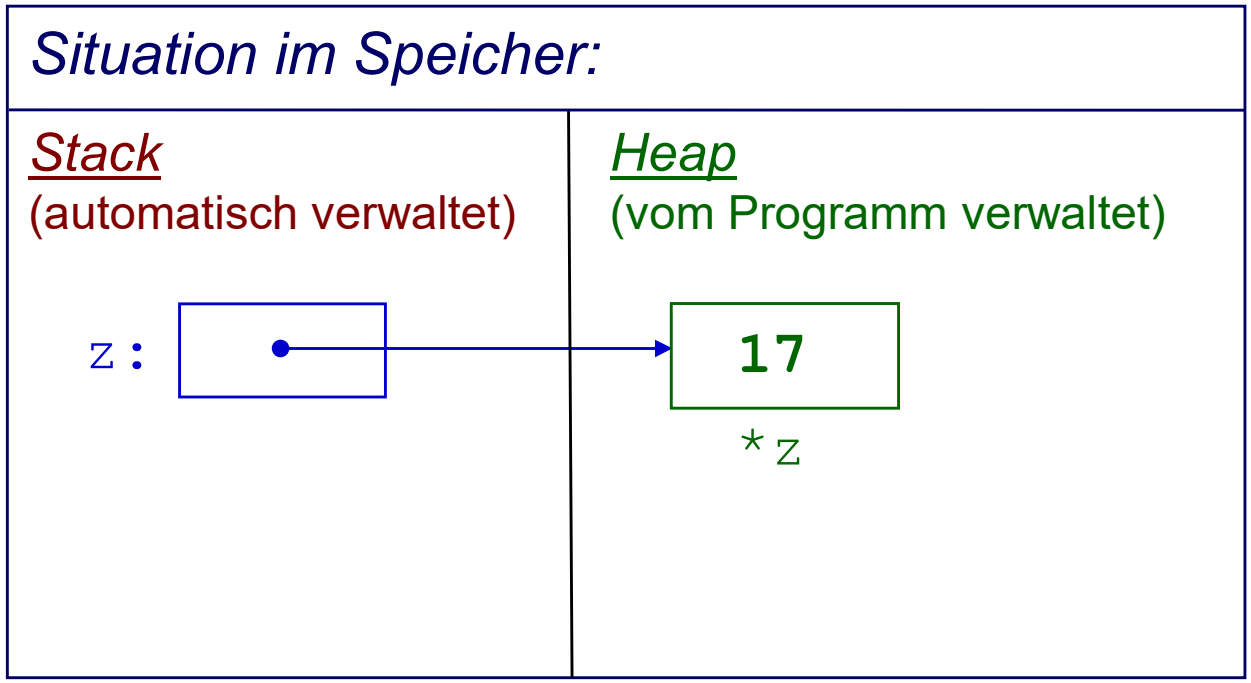
Zeigervariable = ***new*** *Bezugstyp;*

```
int * z ;

z = new int;
```

→ `z` „zeigt“ auf eine neue (anonyme) Variable auf dem Heap

→ Zugriff auf diese Variable durch „Dereferenzieren“:

```
*z = 17;

cout << *z; // Ausgabe: 17
```



*Beachte: Falls der Bezugstyp eine Klasse ist, wird bei **new** der **Konstruktor** der Klasse aufgerufen! (Das Objekt beginnt seine „Lebenszeit“).*

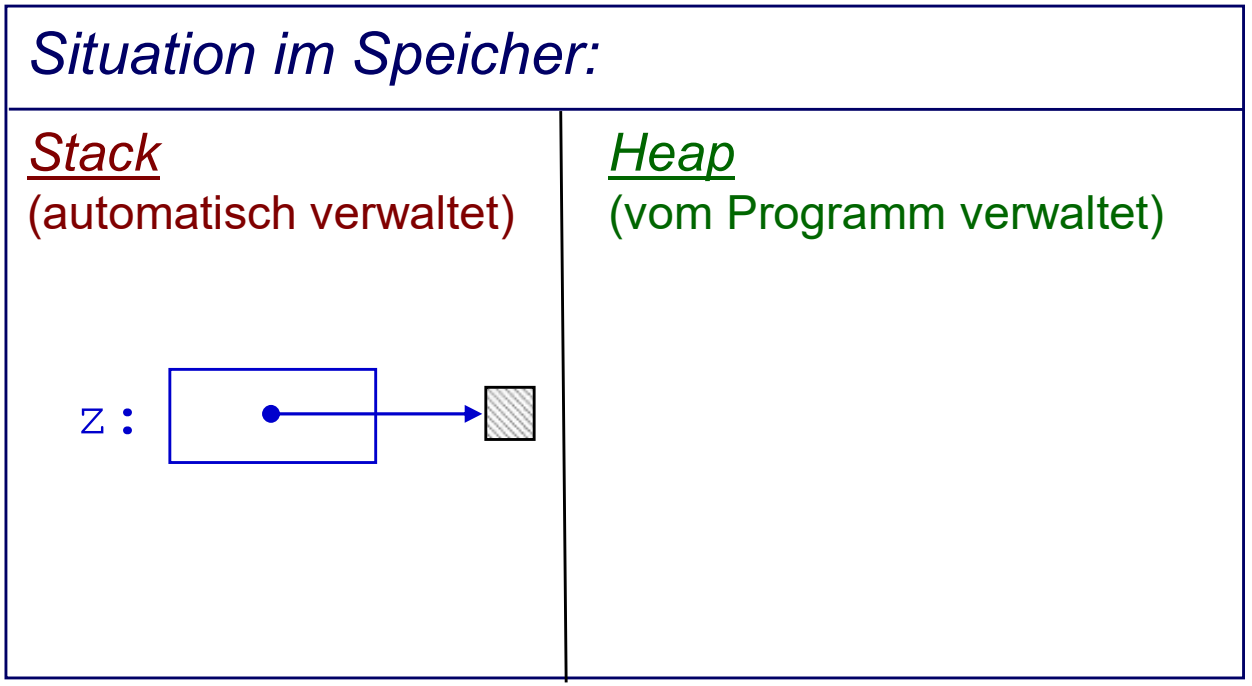
Programmieren mit Zeigern

- Der „Null“-Zeiger

```
int * z = NULL ;
```

→ z zeigt „nirgendwo“ hin

→ Dereferenzieren von z führt zu Laufzeit-Fehler („Absturz“)



Bemerkungen:

- Die symbolische Konstante `NULL` hat den numerischen Wert `0`
- Statt `NULL` kann immer auch `0` verwendet werden:

```
int * z = 0;
```

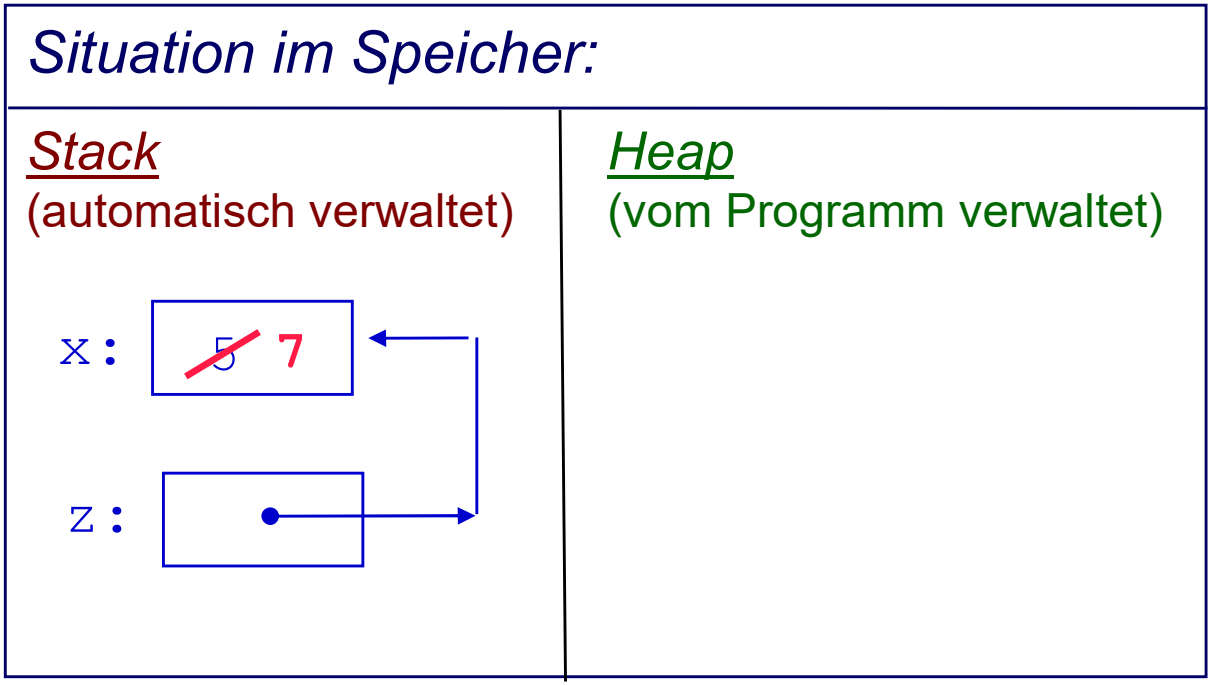
- Falls eine Zeigervariable nicht sofort (oder nicht mehr) verwendet wird, sollte sie immer mit `NULL` initialisert werden!

Programmieren mit Zeigern

- Zuweisung von Adressen (→ Operator **&**)

Zeigervariable = & x ; (x muss eine Variable des Bezugstyps sein)

```
int x = 5;
int *z;

z = &x;
```



→ *z und x bezeichnen denselben Speicherplatz!

```
cout << x << *z; // Ausgabe: 55

*z = 7;

cout << x; // Ausgabe 7!
```

Programmieren mit Zeigern

➢ Zuweisung von anderen Zeigervariablen

Zeigervariable = andere_Zeigervariable;

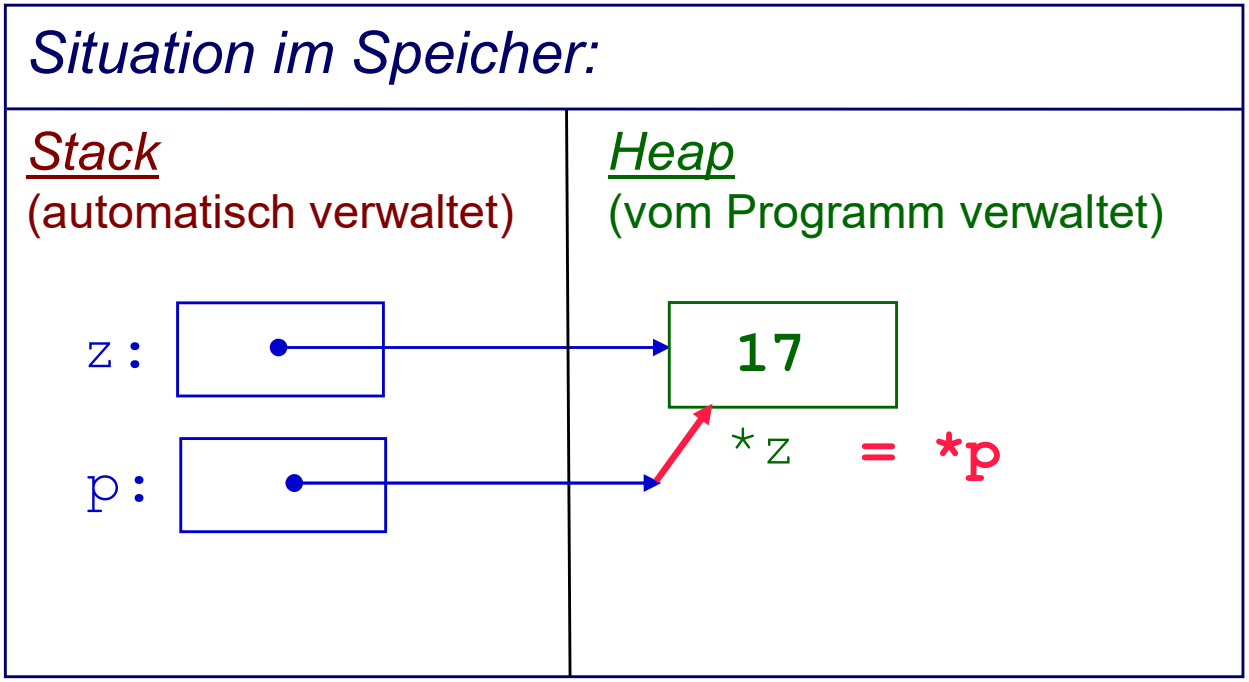
→ beide Zeigervariablen müssen denselben Bezugstyp haben

→ andere_Variable sollte definiert sein

```
int *z, *p;

z = new int;

p = z;
```

→ `z` und `p` zeigen auf dieselbe Variable im Heap

```
*p = 17;

cout << *p << *z;

// Ausgabe: 1717
```



Programmieren mit Zeigern

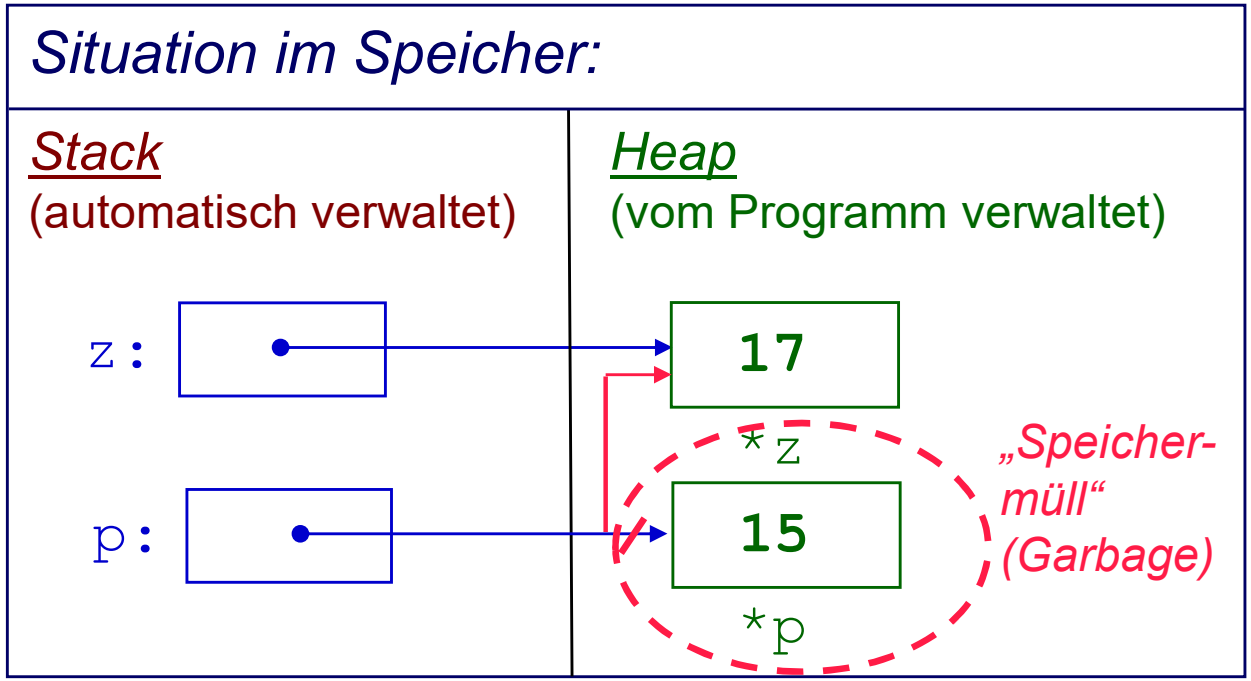
- Vorsicht: „Speichermüll“!

  → *Unerreichbare Speicherplätze auf dem Heap*

```
int *z, *p;

z = new int;
*z = 17;

p = new int;
*p = 15;

p = z;
```

→Das „alte“ *p ist nicht mehr erreichbar!

→Speicherplatz ist nie mehr nutzbar!

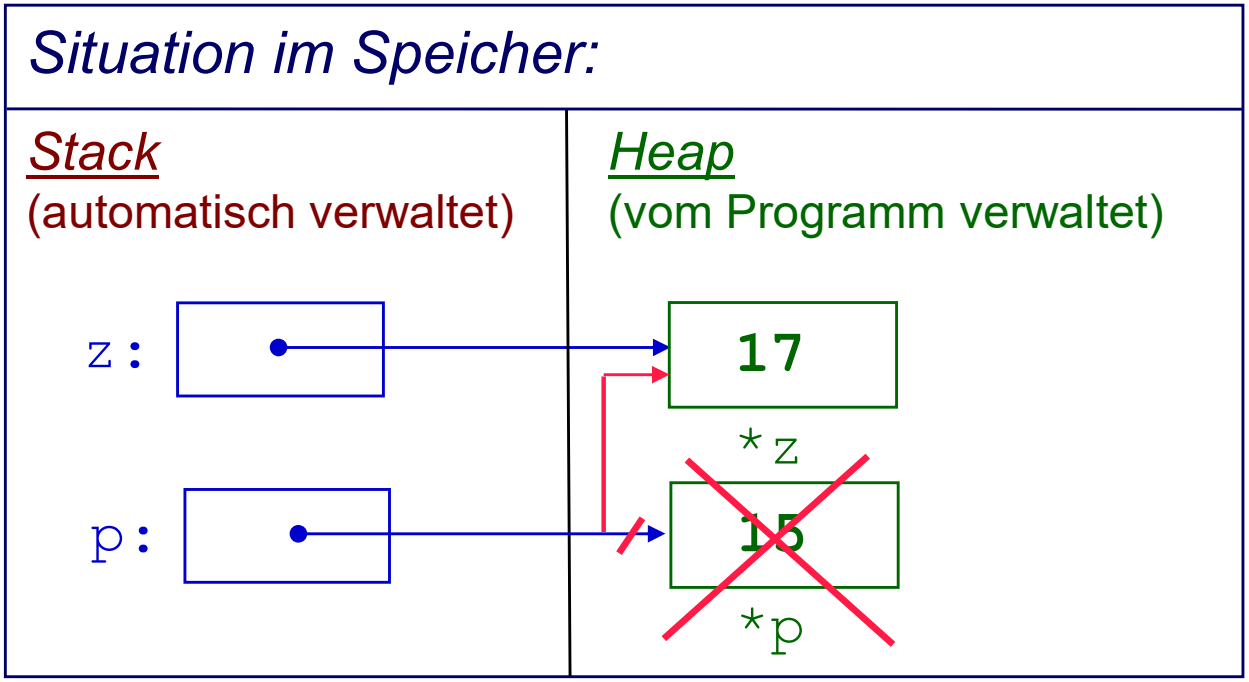


Programmieren mit Zeigern

- Freigeben von Speicher (→ Operator **`delete`**)

`delete` *Zeigervariable;*

```
int *z, *p;

z = new int;
*z = 17;

p = new int;
*p = 15;

...

delete p;

p = z;
```



→ Der vom „alten“ `*p` belegte Speicherplatz kann „wieder verwendet“ werden.

*Beachte: Falls der Bezugstyp eine Klasse ist, wird bei **delete** der **Destruktor** der Klasse aufgerufen! (Das Objekt beendet seine „Lebenszeit“).*

Zeiger und Felder

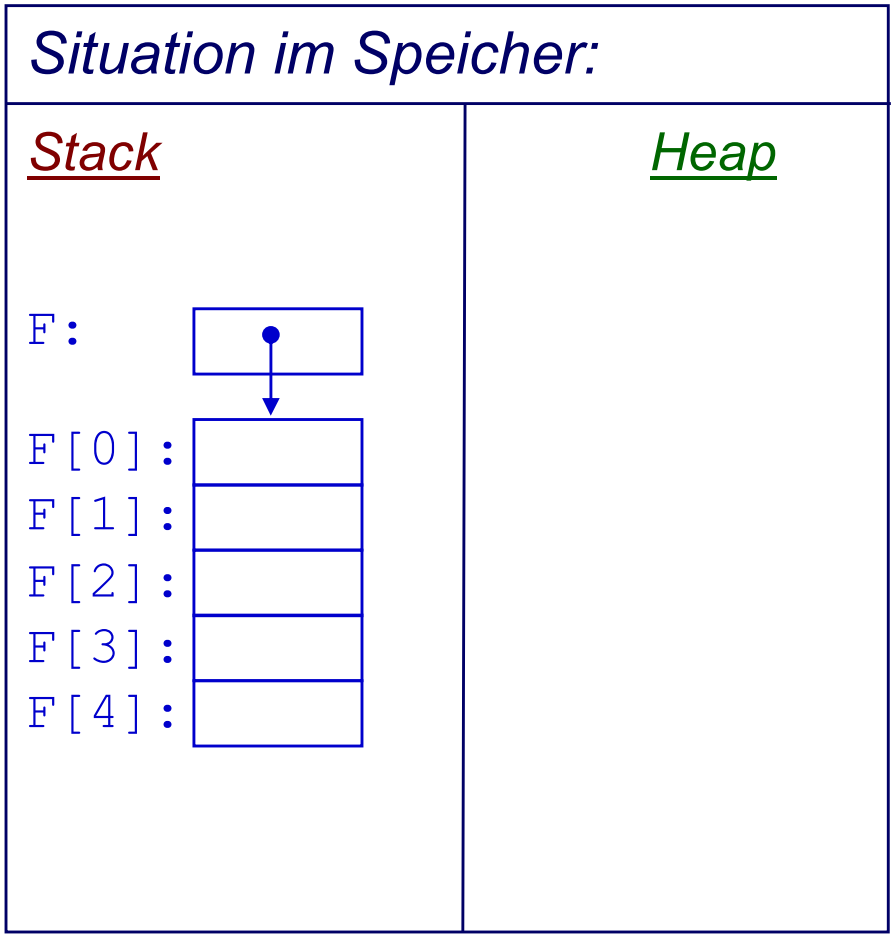
➢ Statische Felder

Elementtyp FeldVariable[Anzahl];

- Werden vollständig auf dem Stack angelegt
- Die *FeldVariable* ist ein konstanter Zeiger auf den Anfang des Feldes
- Indizieren ≈ Dereferenzieren

```
int F[5] ;

F[0] == *F ;
F[1] == *(F+1);
F[2] == *(F+2);
F[3] == *(F+3);
F[4] == *(F+4);
```

- Bei Parameterübergabe wird nur der Zeiger übergeben



Zeiger und Felder

➢ Kopieren von Feldern

- Zuweisungen an (statische) Feldvariablen nicht möglich

```
int F[5];
int G[5];

G = F;    // G ist konstanter Zeiger!
```

- Elementweises Kopieren von Feldern erforderlich:

```
for (int i = 0; i<5; i++)
  G[i] = F[i];
```

Felder und Zeiger

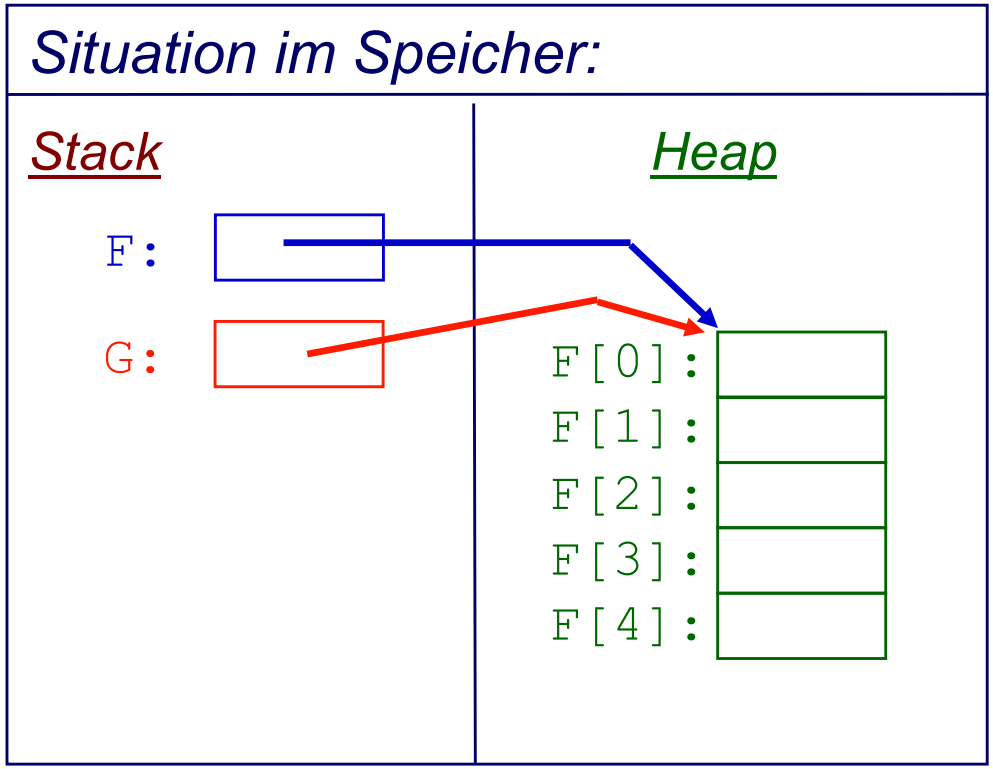
- Dynamische Felder;

  *Elementtyp * Variable;*

  *Variable = **new** Elementtyp [Anzahl];*

  - Zeigervariable auf dem Stack
  - Feldelemente dynamisch auf dem Heap angelegt
  - Indizieren/Dereferenzieren wie bei statischen Feldern.

```
int * F;
F = new int[5];
```



  - Zuweisungen an solche Feld-Variablen sind möglich

```
int * G;
G = F;        // Achtung, es gibt nur ein Feld!
```

Zeiger und Felder

➢ Vorteil von Dynamischen Feldern

- Feldgröße kann zur Laufzeit ermittelt werden:

```
int d;
int * F;

cout << "Wie groß soll das Feld werden? ";
cin >> d;

F = new int[d];
```

- Speicherplatz für das Feld kann wieder freigegeben werden:

```
delete[] F;
```