Vorlesung 5

Zeiger Grundlage

Themen

- Zeiger und Arrays.
- Zeiger und Funktion.
- Typumwandlung
- Dynamische Speicherverwaltung.

Zeiger und Felder (einschließlich Strings)

- **Felder** (= **Arrays**) und als **String**–Variablen benutzte char–Felder werden sehr oft gebraucht.
- BEISPIELE:

```
int A[20]; // A kann 20 int-Werte aufnehmen.

double x[1000]; // x kann 1000 double-Werte aufnehmen.

char s[81]; // s kann z.B. 81 ( ! ) Übertragungsbytes, aber auch
            // einen max. 80 ( ! ) Zeichen langen Text aufnehmen!
```

Zeiger und Felder (einschließlich Strings)

Was haben Felder hier im Kapitel "**Zeiger**" zu suchen?

- Nun, Felder unterscheiden sich von "einfachen Variablen" nicht nur dadurch, dass sie mehr als nur einen Wert vom betreffenden Typ speichern können (s. Beispiele), sondern, auch in der Bedeutung ihrer Namen :

Der Name einer einfachen Variablen  **Steht für den Inhalt der Variablen!**

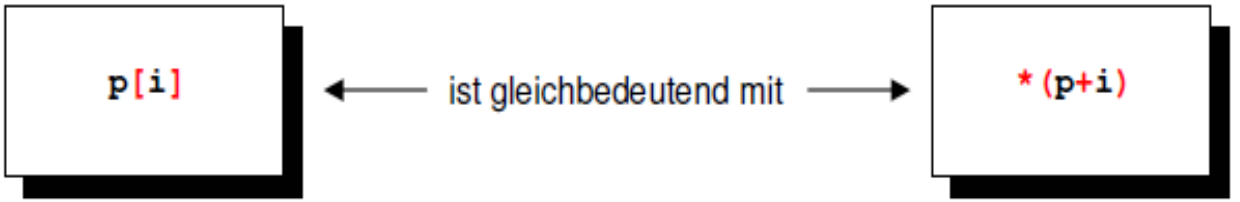
Der Name eines Feldes(=Arrays)  **Steht für die Adresse des Feldes!**

Zeiger und Felder (einschließlich Strings)

```
1  #include <iostream>
2  using namespace std;
3  int main (void)
4  {
5      int i=3 , A[10]={16,5,-1,-8,2,17,30,32,29,25};
6      char c='C' , s[81]="Zeiger sind bockstark!";
7      cout<<"Inhalt der Variablen i :  "<< i<<endl;;
8      cout<<"Adresse der Variablen i : " << &i<<endl;
9      cout<<"Inhalt von A[0] : "<<A[0]<<endl;
10     cout<<"Adresse des Feldes A : "<< A<<endl;
11     cout<<"Probe: Adresse von A[0] : "<< &A[0]<<endl;
12     cout<<"Adresse des 4. A-Elements : "<< A+3<<endl;
13     cout<<"Inhalt des 4. A-Elements : " <<*(A+3)<<endl;
14     cout<<"\n";
15     cout<<"Inhalt der Variablen c : " << c<<endl;
16     cout<<"Adresse der Variablen c : " <<&c<<endl;
17     cout<<"Inhalt von s[0] : " << s[0]<<endl;
18     cout<<"Ausgabe ??? : " << s<<endl;
19     cout<<" Ausgabe?? s[0] : "<< &s[0]<<endl;
20     cout<<"Ausgabe des 4. s-Elements??? : " << s+3<<endl;
21     cout<<"Inhalt des 4. s-Elements : "<<*(s+3)<<endl;
22     system("Pause");
23     return 0;
24 }
25
```

vorlesung5_1

Zeiger und Felder (einschließlich Strings)



Zeiger und Felder (einschließlich Strings)

- Eine Merkwürdigkeit
  Für int *p und int a[10]
  - p = &a[0];
  - p = a;

Zeiger und Arrays: Dereferenzierung

- Eine Merkwürdigkeit
  Für int *p und int a[10]

  p = a;
  - *p = 3;
  - p[0] = 3;

Ist p ein Array?

Vorlesung5_2.cpp

Zeiger und Arrays: Arithmetik

- Zeiger Erhöhung

```
int *p; int a[10];
p = a + 1;
```

  - Wenn **a** 0x00FA3 ist , dann **p**?

Vorlesung5_3.cpp

Arithmetik und Dereferenzierung

- Zeiger Erhöhung
  int *p; int a[10];

  p = a;
  - *(p+1) = 76;
  - *(p+3) = -8;

    Vergleich mit a[?]

Vorlesung5_3.cpp

Unsere erste Funktion mit Zeigern

Zeiger Anwendung

Ziel: Eine Funktion, die eine Variable von *main* verändert. Ohne Rückgabe!

Bisher: übliche Funktion

- Nur durch Rückgabe können wir eine Variable im *main* verändern.

  Typischer Aufruf

```
c = funktion (a, b);
```

Übergabe per Wert

- Keine Möglichkeit ohne Adressen eine in *main* deklarierte Variable zu verändern.

```
funktion (a, b, c);
```

Durch diesen Aufruf werden a, b, c unverändert bleiben

Erklärung

- Durch einen üblichen Aufruf übergeben wir nur Werte in der Funktion
- Die Variablen haben einen Gültigkeitsbereich
  - Die Variablen in *main* können nicht anhand des Namens in einer Funktion verändert werden.
  - Umgekehrt auch für die Variablen in der Funktionen.
- Die Speicheradresse einer Variable ist trotzdem überall gültig!

Von nun an: Nutzung von Zeigern

- Wenn die Adresse der Variable in die Funktion übergeben wird, kann man die Variable direkt in der Funktion verändern.

  Typischer Aufruf mit Adressenübergabe

```
funktion (a, b, &c);
```

  - Von den Variablen a und b nutzt man nur den Wert. Die können nicht in der Funktion verändert werden
  - Von der Variable c übergibt man die Adresse, deshalb ist c verändbar

Vorlesung5_4.cpp

Übergabe per Referenz: *Eigenschaften*

- Bei Deklaration sind Zeigervariablen als Parameter.

```
funktion (int a, int b,int *s,int *d, int *m)
```

Vorlesung5_4.cpp

Übergabe per Referenz: *Eigenschaften*

- Bei Aufruf ist Adressenübergabe

```
funktion (a, b,&sum,&diff,&mult);
```

  - Von den Variablen a und b nutzt man nur den Wert. Die können nicht in der Funktion verändert werden
  - Von den Variablen sum, diff und mult übergibt man die Adresse.

Vorlesung5_4.cpp

Referenz vs. Wert

- Art von Parametern (Variablen und Zeigern)
  - Referenz: Der Parameter ist ein Zeiger (erkennbar durch das Symbol * bei der Deklaration)
  - Wert: Der Parameter ist kein Zeiger

Vorlesung5_4.cpp

Referenz vs. Wert

- Verwendungszweck jeder Art
  - Referenz: Eine Adresse wird übergeben, deren Variable benutzt oder verändert wird.
  - Wert: Ein Wert wird übergeben, der benutzt wird.

Vorlesung5_4.cpp

CAST: Typenumwandlung

- ***Automatische Umwandlungen*** zwischen Datentypen sind immer riskant und manchmal Verboten

```
int a = 1.7; //Verlust von Wert
double *p = &a; //Verboten
```

- Solche Umwandlungen dürfen durch einen ***Cast*** optimal ausgeführt werden.

```
int a = (int) 1.7; //Verlust von Wert
double *p = (double *) &a; //Verboten
```

Vorlesung5_5.cpp

CAST: Typenumwandlung

(Neuer Datentyp) Ausdruck

```
(int) 1.7
```

Wenn a zum Beispiel von Typ char ist

```
(double *) &a
```

Vorlesung5_5.cpp

Vom Problem zum Programm

Anwendung von Zeigern

- Ein Programm, das zuerst einen Array mit 5 Elementen nutzt, danach einen Array mit 2 Elementen.

Vorlesung5_6.cpp

new - delete

- ***new:*** reservieren

```
new int [5];
new int [2];
```

- ***delete:*** freigeben

```
delete [] p;
```

Dynamische Speicherverwaltung

- *Syntax*
  - ***new**:* reservieren

```
new DATENTYP [ANZAHL_DER_ELEMENTE];
```

  - ***delete**:* freigeben

```
delete [] ZEIGER,_DER_DIE_ADRESSE_SPEICHERT;
```

Dynamische Speicherverwaltung

- *Besonderheiten*
  - Immer, wenn Speicherplatz reserviert wird, muss die gelieferte Adresse (in einem Zeiger) **gespeichert** werden.
  - Die Adresse muss immer (in irgendwelchem Zeiger) behalten bis zum **Freigabe**

Vorlesung5_6.cpp (Achtung Kommentar)