Crashkurs C++ - Teil 1

- Intro
- Speicherverwaltung
  - Variablen, Pointer, Referenzen
  - Felder
  - statische & dynamische Allozierung

Birgit Möller & Denis Williams
AG Bioinformatik & Mustererkennung
Institut für Informatik
Martin-Luther-Universität Halle-Wittenberg

Intro

- C++ ist die objektorientierte Weiterentwicklung von C
- C++ ist maschinennah implementiert

  $\longrightarrow$ direkte Manipulation des Speichers möglich !!!
- C++ erfordert eine explizite Speicherverwaltung (keine *garbage collection*!)
- hohe Flexibilität (aber auch höhere Komplexität. . .) durch
  - Pointer-Datentyp
  - Mehrfachvererbung
  - Templates

Speicherverwaltung

Der Kern des Ganzen:

C++ unterscheidet zwischen einer Variablen und ihrer Position im Speicher

- daraus folgen grundsätzlich 3 Arten von Variablen-/Objekt-Datentypen:
  - einfache Variablen beliebigen Typs, z.B. *int, double, bool*
  - Zeigervariablen bestimmten Typs, z.B. *int*, double*, bool**
  - Referenzen auf Variablen bestimmten Typs, z.B. *int&*

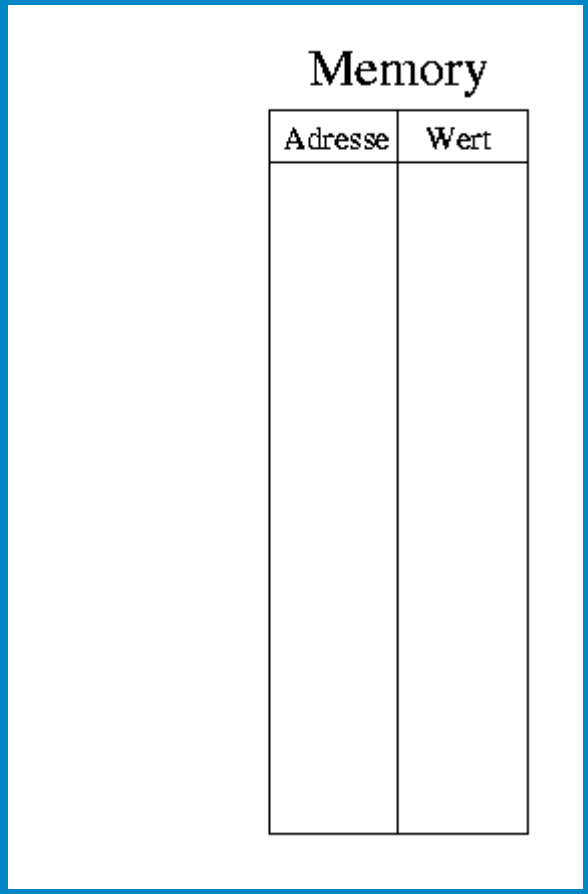
```
int a;    // Variable des Typs Integer

int* z;   // Variable des Typs "Zeiger auf Integer"

int& r;   // Referenz des Typs Integer
```

Speicherverwaltung - Variablen & Zeiger

Beispiel:

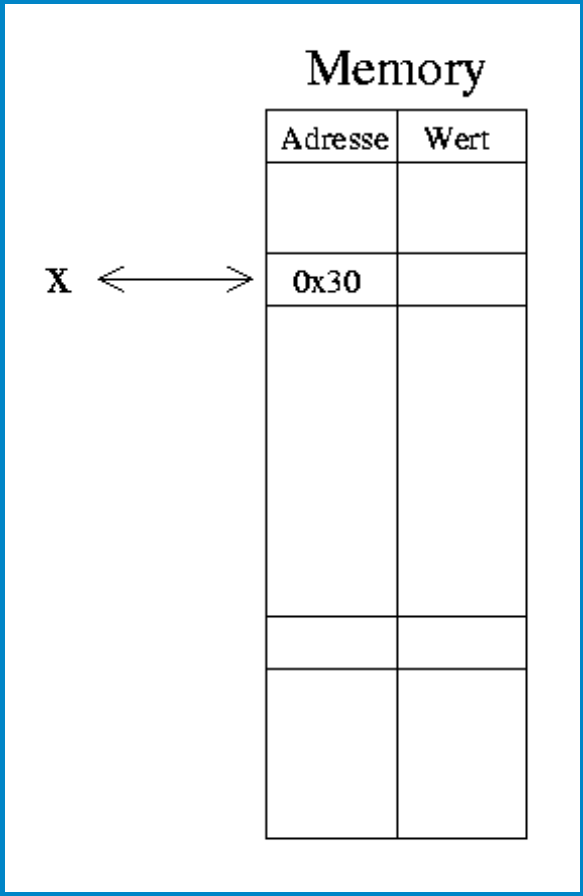


Speicherverwaltung - Variablen & Zeiger

Beispiel:

- Variable x vom Typ T:

```
T x;
x = 50;
```

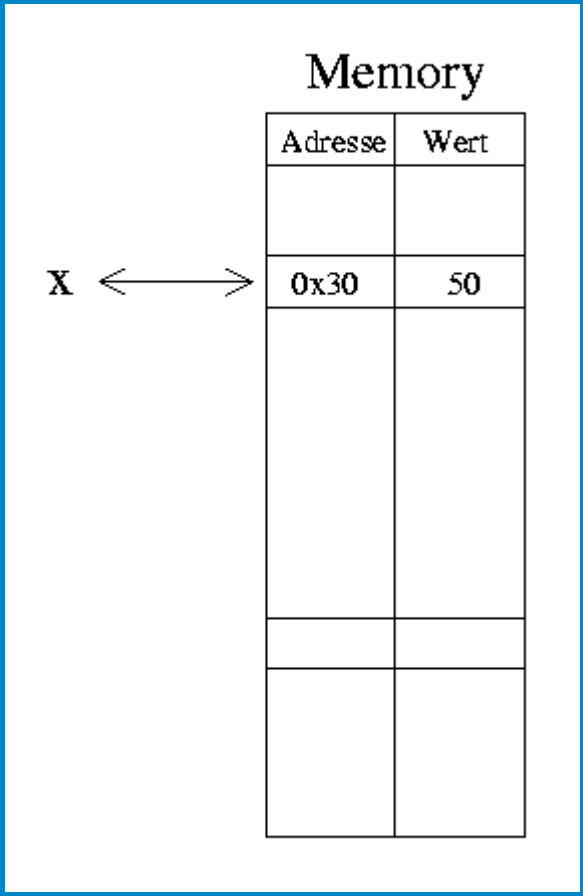


Speicherverwaltung - Variablen & Zeiger

Beispiel:

- Variable x vom Typ T:

```
T x;
x = 50;
```



Speicherverwaltung - Variablen & Zeiger

Beispiel:

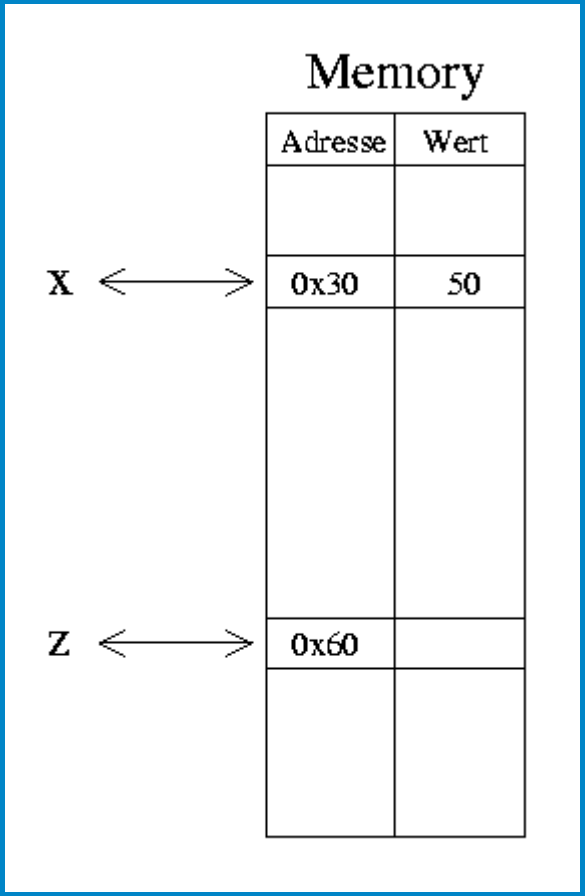
- Variable x vom Typ T:

```
T x;
x = 50;
```

- Variable z vom Typ ”Zeiger auf T”:

```
T* z;
```

- z zeigt an, wo eine Variable im Speicher steht



Speicherverwaltung - Variablen & Zeiger

Beispiel:

- Variable x vom Typ T:

```
T x;
x = 50;
```

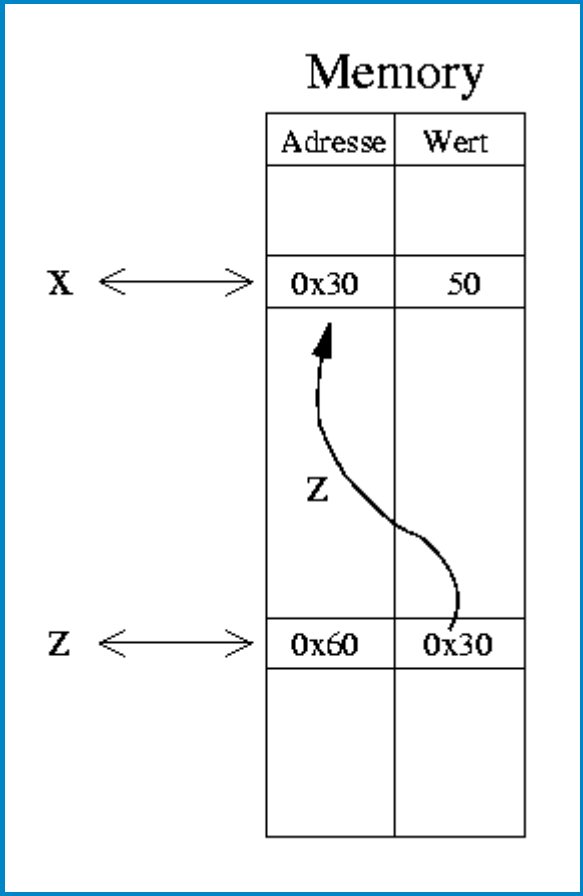
- Variable z vom Typ ”Zeiger auf T”:

```
T* z;
```

- z zeigt an, wo eine Variable im Speicher steht

- Zuweisung einer Adresse an z mit &:

```
z = &x;
```



Speicherverwaltung - Variablen & Zeiger

Beispiel:

- Variable x vom Typ T:

```
T x;
x = 50;
```

- Variable z vom Typ ”Zeiger auf T”:

```
T* z;
```

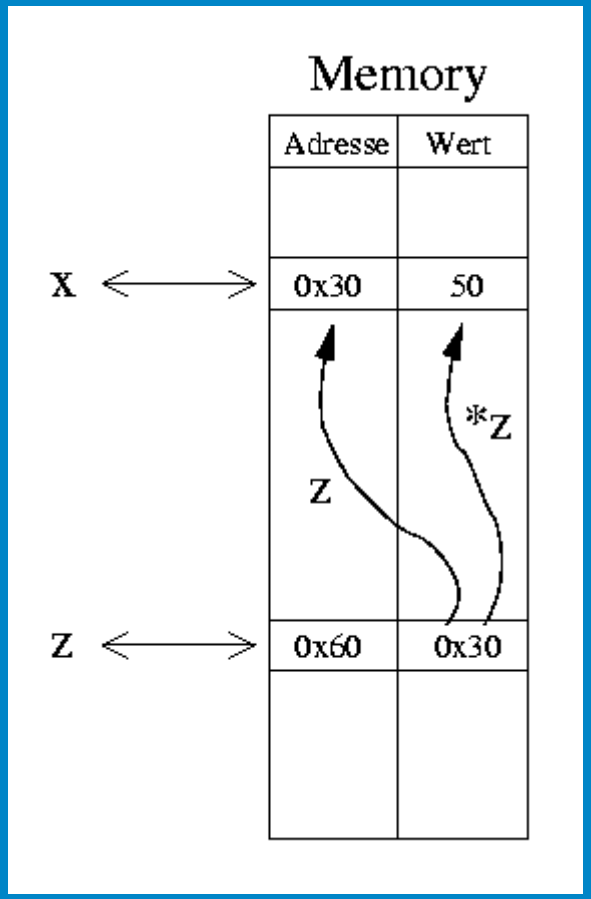
- z zeigt an, wo eine Variable im Speicher steht

- Zuweisung einer Adresse an z mit &:

```
z = &x;
```

- Zugriff auf Inhalte von x per Dereferenzierung *:

```
*z liefert 50
```



Speicherverwaltung - Variablen & Zeiger

Beispiel:

- Variable x vom Typ T:

```
T x;
x = 50;
```

- Variable z vom Typ ”Zeiger auf T”:

```
T* z;
```

- z zeigt an, wo eine Variable im Speicher steht

- Zuweisung einer Adresse an z mit &:

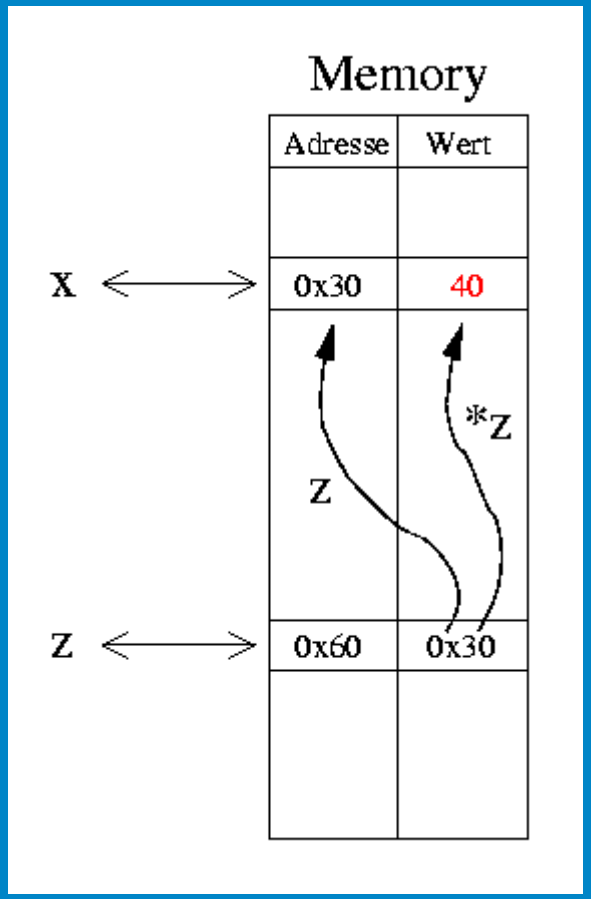
```
z = &x;
```

- Zugriff auf Inhalte von x per Dereferenzierung *:

```
*z liefert 50
```

- wird x verändert, ändert sich nur *z, z selbst nicht!

```
 x = 40;
*z = 40;
```



Speicherverwaltung - Referenzen

- eine **Referenz** ist ein Alias für eine Variable
- Referenzen sind **immer** mit schon existierenden Variablen verknüpft

  → Initialisierung direkt bei der Deklaration !

```
int& a;          // geht nicht, Referenz uninitialisiert

int  b;
int& a = b;      // ok, a verweist auf b
```

- jede Änderung wirkt sich jeweils auf Variable und Referenz aus
- wird die Variable gelöscht, wird auch die Referenz ungültig

  (Achtung bei Rückgabewerten von Funktionen!!!)

- auf eine Variable kann es mehrere Referenzen geben,

  aber Referenzen sind immer eindeutig

Speicherverwaltung - Pointer-Arithmetik

Pointer-Arithmetik

- Zeigervariablen sind Variablen mit besonderem Inhalt
- nutzbar wie jede andere Variable

  $\Rightarrow$ insbesondere auch in arithmetischen Berechnungen

”Pointerarithmetik: Fluch und / oder Segen von C / C++”

effizienter Speicherzugriff vs. Sicherheit / Robustheit

```
int a = 42;

int* z = &a;

z = z + 3;    // springe 3 Positionen weiter im Speicher !!!

(*z) --> ???!
```

Speicherverwaltung - Arrays

Anwendungsbeispiel für Zeigervariablen: Felder (Arrays)

- ein Array entspricht einer Menge von linear hintereinander angeordneten Speicheradressen

```
int array[10];       // Feld der Groesse 10 fuer 10 Integerwerte

feld[0];             // Zugriff auf das erste bzw. sechste Feld
feld[5];
```

- Zugriff auf einzelne Elemente mit dem ”[ ]”-Operator
- Arrays und Pointer sind gewissermaßen dual

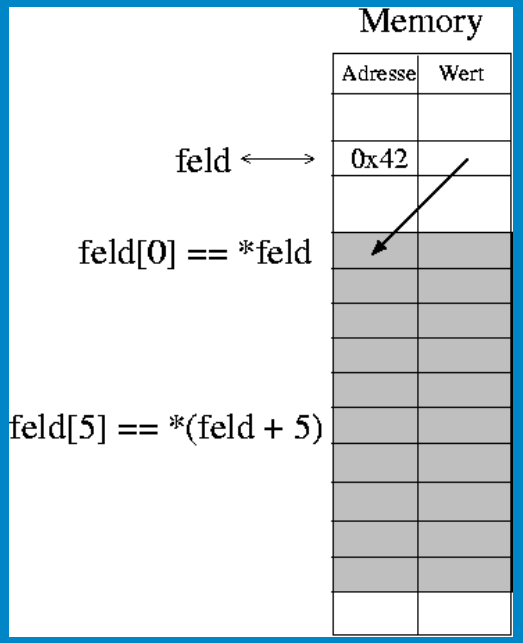
  $\Rightarrow$ Pointer-Arithmetik als alternative Zugriffmöglichkeit

Speicherverwaltung - Arrays

- ein Feldbezeichner entspricht einem Zeiger auf die Anfangsadresse eines Speicherbereichs

```
int feld[10];

int* z = feld;
```



- der ”[ ]”-Operator ist damit Derefenzierungs-Operator

```
feld[0] == *z

feld[5] == *(z+5)
```

Speicherverwaltung - Allokation

C++ unterscheidet 2 Arten der Speicherverwaltung / -allokation:

- **statisch:**

  Speicher wird vom Programm automatisch reserviert und wieder freigegeben
- **dynamisch:**

  Speicher wird explizit zur Laufzeit alloziert und freigegeben

Dynamische Allokation ist notwendig, wenn

- der Speicherbedarf zur Compile-Zeit unbekannt ist
- Objekte global sichtbar sein sollen

Dynamische Allokation basiert auf Zeigervariablen!

Speicherverwaltung - Allokation

Grundidee:

- fordere explizit Speicher an und verwalte diesen über Zeiger
- Operatoren:
  - *new* : reserviert Speicher für ein Objekt / eine Variable

    ```
    T* new T();
    ```

  - *delete* : gibt reservierten Speicher wieder frei

    ```
    void delete(T*);
    ```

```
helloWorldClass* h;

h = new helloWorldClass();

h->sayHello();                    // ok, Speicher wurde alloziert

delete h;                         // nie vergessen!!!
```

Speicherverwaltung

- wenn ein allozierter Bereich nicht mehr referenziert wird, gibt es Speicherlecks....

```
void alloziereSpeicher() {
  helloWorldClass* h= new helloWorldClass();    // Speicherleck!!!
}

int main(int argc, char **argv) {               // die main-Funktion
  alloziereSpeicher();
};
```

- Faustregel:

  zu jedem *new* im Programm muss es ein korrespondierendes *delete* geben!

Besonderheit: Allokation von Speicher für Felder

- Operatoren: *new [ ]* und *delete [ ]*

```
int *feld = new int[10];

delete [] feld;
```

Literatur

Weiterführende Literatur:

- Einführungsskript:

  ”C++ für Java-Programmierer” (Skript der Technischen FH Berlin)

- Bjarne Strousstrup, ”Die C++ Programmiersprache”,

  die *Bibel* für C++-Programmierer

- Prinz, Prinz, ”Objektorientierte Programmierung mit C++”
- Lippman, ”C++ Primer”
- Scott, Meyers, ”(Mehr) Effizient C++ programmieren”

. . . ansonsten gilt: **”Versuch macht klug”**