Programmiertechnik II

Ralf Herbrich

C++ Einführung

Überblick

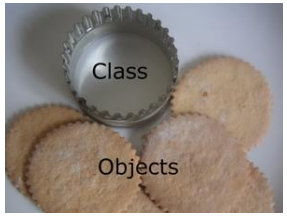
1. Unterschiede zu C
   - Kommentare, Initialisierungen & Speicherverwaltung
   - Primitive Datentypen
   - Zeiger & Referenzen
   - `const` Operator
   - Funktionen & (Operator)-Overloading
   - Templates
2. Abstrakte Datentypen & Klassen in C++
   - Vererbung
   - Spezielle Konstruktoren
3. Standard Template Library

Überblick

1. Unterschiede zu C
   - Kommentare, Initialisierungen & Speicherverwaltung
   - Primitive Datentypen
   - Zeiger & Referenzen
   - `const` Operator
   - Funktionen & (Operator)-Overloading
   - Templates
2. **Abstrakte Datentypen & Klassen in C++**
   - Vererbung
   - Spezielle Konstruktoren
3. Standard Template Library

Abstrakte Datentypen & Objekte

- **Datentyp**: Eine Menge von Daten zusammen mit einer Familie von Operationen
- **Abstrakter Datentyp**: Beschrieben wird die Menge und die Semantik der Operationen, nicht aber die interne Repräsentation der Daten oder die Implementierung der Operationen
- **Klasse**: Beschreibung von Strukturen mit gleichen Eigenschaften. Sie zeichnet sich durch zwei Merkmale aus:
  1. Definiert eine Menge von Daten
  2. Definiert Handlungen auf Daten in Form von Methoden (Funktionen, die der Klasse angehören)
- **Instanz (einer Klasse)**: eine Variable „vom Typ Klasse“
- **Objekt**: Instanz einer Klasse; eine Menge von Daten (definiert in der Klasse) mit den Eigenschaften (Funktionen, Zugriffsrechte…) der Klasse, der es angehört



Klassen in C++

- **Klassendefinition**: In C++ gibt es zwei Möglichkeiten, Klassen zu definieren
  1. `struct <className> {}`: Alle Daten und Methoden sind `public`
  2. `class <className> {}`: Alle Daten und Methoden sind `private`
- **Sichtbarkeit**:
  - `public`: Daten & Methoden sind bei jedem Objekt direkt aufrufbar
  - `private`: Daten & Methoden sind nur innerhalb der Klasse aufrufbar
  - `protected`: Daten & Methoden sind nur innerhalb der Klasse und aller abgeleiteten Klassen aufrufbar
- **`const` Methoden**: Eine Methode, die als `const` deklariert ist, ändert keine Daten des Klasse während der Ausführung

```
class MyInt {
    int i;
  public:
    int getValue() const { return (i); }
}
```

Objektlebensdauer: Konstruktoren & Destruktoren

- Es gibt zwei spezielle Methoden, die mit der **Lebensdauer eines Objektes** zusammenhängen:
  1. **Erzeugung (*constructor*)**: Wird aufgerufen, wenn ein Objekt erzeugt wird (entweder als lokale Variable, Funktionsargument, oder auf dem *heap* wenn `new` benutzt wird)
  2. **Zerstörung *(destructor)***: Wird (implizit) aufgerufen, wenn ein Objekt aus dem *Scope* verschwindet oder explizit vom *heap* wenn `delete` benutzt wird.
- Beide Methoden haben keinen Ergebnistyp
  - **Konstruktoren**: Haben den gleichen Namen wie die Klasse/Struktur
  - **Destruktor**: Haben den gleichen Namen wie die Klasse/Struktur mit vorgestellter Tilde („*not*“ *constructor*)
- Die Daten einer Klasse/Struktur können im Konstruktor initialisiert werden
  - Komma nach dem Konstruktor und Komma-separierte Liste von `<Datenfeld>(<Wert>)`
- **Beispiel**:

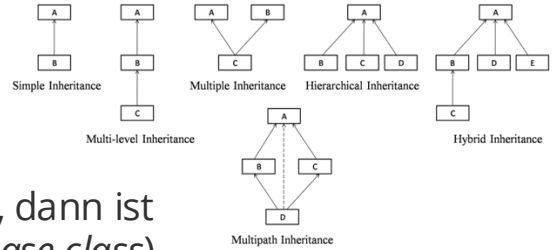
```
class MyInt {
    int value;
  public:
    MyInt(int v=0) : value(v) {}
     ~MyInt () {}
}
```

Überblick

1. Unterschiede zu C
   - Kommentare, Initialisierungen & Speicherverwaltung
   - Primitive Datentypen
   - Zeiger & Referenzen
   - `const` Operator
   - Funktionen & (Operator)-Overloading
   - Templates
2. Abstrakte Datentypen & Klassen in C++
   - **Vererbung**
   - Spezielle Konstruktoren
3. Standard Template Library

Klassenhierarchien: Vererbung

- Klassen haben oft Überlapp von Daten & Methoden miteinander
  - **Beispiel**:
    - Zahlentypen unterstützen arithmetische Operationen *(methods)*
    - Graphische Objekte haben immer eine Größe und Position *(members)*
- Wenn Klasse `B` **mehr/speziellere** Daten & Methoden als Klasse `A` hat, dann ist `B` eine abgeleitete Klasse (*derived class*) von `A` und `A` die Basisklasse (*base class*)
  - C++ erlaubt sowohl mehrere abgeleitete Klassen als auch Basisklassen
- Bei der Klassendeklaration kann man die Basistypen hinter einem `:` angeben (zusammen mit der Sichtbarkeit der Methoden & Daten des Basisklasse)




```
class A {
  private:
    int a;
 protected:
    int b;
  public:
    int c;
}
```

```
class B : public A {
  public:
    B() {
      a = 1;  // error!
      b = 2;
      c = 3;
    }
}
```

```
class B : private A {
  public:
    B() { c = 3; }
}

B var;
var.c = 10;  // error!
```

Vererbung (ctd)

- **Konstruktoren** werden erst in den Basisklassen aufgerufen und als letztes in der abgeleiteten Klasse (**Destruktoren** genau andersherum!)
  - Man kann in der abgeleiteten Klasse explizit einen Basiskonstruktor aufrufen

```
class B : public A {
  public:
    B(int i) : A(i) { … }
}
```

- **Zeiger** eines Basistyps können immer auf eine Instanz eines abgeleiteten Typs zeigen, da alle Daten & Methoden auch Teil des abgeleiteten Typs sind
  - **Aber**: Damit die richtige Funktion zur Laufzeit benutzt wird, muss eine Methode mit dem Schlüsselwort `virtual` versehen werden!
  - Beachte: **Virtuelle Funktionen** sind immer **langsamer**, da sie über Funktionszeiger implementiert werden müssen.

virtual.cpp

Überblick

1. Unterschiede zu C
   - Kommentare, Initialisierungen & Speicherverwaltung
   - Primitive Datentypen
   - Zeiger & Referenzen
   - `const` Operator
   - Funktionen & (Operator)-Overloading
   - Templates
2. Abstrakte Datentypen & Klassen in C++
   - Vererbung
   - **Spezielle Konstruktoren**
3. Standard Template Library

Spezielle Konstruktoren: *Copy* und *Move*

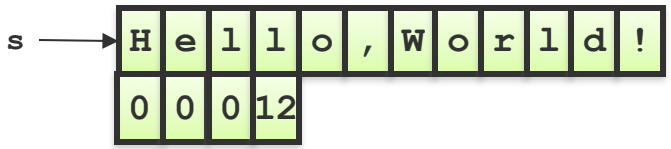
- Objekte werden entweder **kopiert** (*copy*) oder **verschoben** (*move*)
- Verschieben für große Datenmengen kann Zeiger kopieren (Effizienz!)
- **Kopieren**
  1. Konstruktor mit Objekt der gleichen Klasse: `T(const T&) { …}`
     - **Beispiel**: `T t1; T t2 { t1 };`
  2. Zuweisung auf ein bereits exisitierendes Objekt der gleichen Klasse: `T& operator=(const T&) { …}`
     - **Beispiel**: `T t1; T t2; t2 = t1;`
- **Verschieben**
  3. Konstruktor mit Objekt der gleichen Klasse: `T(const T&&) { …}`
     1. **Beispiel**: `T t1; T t2 (std::move(t1));`
  4. Zuweisung eines Rückgabewerts einer Funktion `operator=(const T&&) { …}`
     1. **Beispiel**: `T t1; t1 = f();`

copymove.cpp

Beispiel: `MyString` Klasse

- **C-Zeichenketten**: Zeiger auf Speicher von `char` (endet bei Nullzeichen)

  - **Vorteile**:
    - Kann direkt verkürzt werden.
    - Nur ein Byte extra; egal wie lang die Zeichenkette ist
  - **Nachteile**:
    - Länge bestimmen braucht linearen Zeitaufwand!
- **`MyString` Klasse**: Zeiger auf Speicher von `char` und explizite Länge



mystring.cpp

Überblick

1. Unterschiede zu C
   - Kommentare, Initialisierungen & Speicherverwaltung
   - Primitive Datentypen
   - Zeiger & Referenzen
   - `const` Operator
   - Funktionen & (Operator)-Overloading
   - Templates
2. Abstrakte Datentypen & Klassen in C++
   - Vererbung
   - Spezielle Konstruktoren
3. **Standard Template Library**

Standard Template Library

- In den 1980er Jahren entwickelte von Hewlett Packard
  - Schwerpunkt Datenstrukturen und Algorithmen
  - Generische Funktionen mit Templates (Typvariablen)
- Bestandteile (auszugsweise)
  - Container (Behälterklassen) **PT 2 (erste Hälfte)**
  - Iteratoren
  - Algorithmen **PT 2 (zweite Hälfte)**
  - Zeichenketten
  - Eingabe und Ausgabe
  - Numerik **Mathe II**
  - Zufallszahlengeneratoren und Transformatoren für Wahrscheinlichkeitsverteilungen **Mathe III**
  - Werkzeuge für Multithreading
  - Reguläre Ausdrücke
  - Werkzeuge zur Zeitmessung

Alexander Stepanow (1950 –)

Zusammenfassung

- **Unterschiede zu C**
  - Speicherverwaltung mit `new` and `delete`
  - `bool` Datentyp für Wahrheitswerte
  - Referenzen sind Zeiger, die nur einmal initialisiert werden
  - `const` Operator ist zentral für Korrektheit und Effizienz in C++
  - Funktionen können überladen werden (auch Operatoren!)
  - Templates erlauben Typen zu Variablen zur Compilezeit zu machen
- **Abstrakte Datentypen & Klassen in C++**
  - Klassen verbinden Daten und Methoden, die diese Daten ändern
  - Sichtbarkeiten erlauben Kapselung von Implementierungsdetails
  - Konstruktoren und Destruktoren sind spezielle Methoden
- **Standard Template Library**
  - Im Rest der Vorlesung lernen wir alle Algorithmen und Datenstrukturen direkt kennen

Hasso Plattner Institut
Digital Engineering · Universität Potsdam

Viel Spaß bis zur nächsten Vorlesung!