Die Programmiersprache C++

Die Programmiersprache C++

Inhalt

1 - Unterschiede C und C++

Die Programmiersprache C++

1 – Unterschiede C und C++

- Historie
- Kompatibilität
- Was man in C++ anders macht

Die Programmiersprache C++

Historie

- In den Anfängen der Programmierung waren die Programme, die erstellt wurden meist klein (wenige Kilobyte) und in Assembler geschrieben.
- Mit C konnte man dann prozedural und sogar strukturiert programmieren.
- Programme und Bibliotheken konnten von verschiedenen Programmierern verstanden und genutzt werden.
- Erst die Objektorientierte Programmierung z.B. mit C++ ermöglichte ein sicheres reibungsloses Arbeiten an grossen Projekten...

Die Programmiersprache C++

Kompatibilität

- Ein C++ Compiler erzeugt meist im ersten Schritt C Code, der erst in weiteren Schritten in Assembler umgewandelt wird.
- So ergibt sich automatisch eine gewisse Kompatibilität von C++ zu C.
- C++ kann sowohl prozedural als auch objektorientiert programmiert werden.
- Die Datenkapselung ist wesentlich konsequenter handhabbar als in C.

Die Programmiersprache C++

Was man in C++ anders macht...

In C++ sollten folgende Dinge vermieden werden, da vor allem die Typsicherheit nicht gewährleistet ist:

- #define Textersetzung ohne Typenprüfung
- malloc liefert void pointer zurück, der typenlos ist
- unions
- void * ist typenlos
- casts Typekonvertierung kann Fehler provozieren

Die Programmiersprache C++

Was man in C++ anders macht...

Statt Konstanten mit #defines zu definieren können wir mit const eine Konstante definieren. Speichereffizienter ist jedoch die enum Anweisung:

```
#include <stdio.h>

int main(int argc, char **argv){
  enum Color { red, green, blue };
  Color r = red; // Value range check by compiler
  switch(r)
  {
    case red  : printf("red\n");   break;
    case green: printf("green\n"); break;
    case blue : printf("blue\n");  break;
  }
}
```

Die Programmiersprache C++

Was man in C++ anders macht...

Die const Anweisung kann auch in Zusammenhang mit Pointern verwendet werden.

```
char c1 = 'A', c2='B';
char* p1
const char* p2;                 // pointer is protected
char* const p3=&test;           // contents is protected
const char* const p4= &test;    // both protected

p1 = &c1;
*p1 = 'X‘;
p2 = &c1;     // not possible! Pointer would be changed
*p2 = 'X‘;
p3 = &c1;
*p3 = 'X';    // not possible! Contents would be changed
p4 = &c1;     // not possible! Pointer would be changed
*p4 = 'X';    // not possible! Pointer would be changed
```

Die Programmiersprache C++

Was man in C++ anders macht...

In C++ kann auch eine Methode (Funktion) als const definiert werden. Dann können die Klassenattribute nicht durch diese Funktion verändert werden. Beispiel dafür sind getter Methoden.

```
int fkt() const; // class-attributes can not be changed!
```

2 - Neue Sprachmittel

Die Programmiersprache C++

2 – Neue Sprachmittel

- Deklaration für einen Block
- Struct
- Typenkonversion
- Defaultparameter
- Referenzen
- New und delete
- Namespaces

Die Programmiersprache C++

Deklaration für einen Block

Zunächst kann man in C++ eine Variable auch im laufenden Programmcode deklarieren z.B.:

```
for (int i=0; i<10; i++){  // variable i is only valid insode of
                           // the „for“ loop
}
// or it might be declared inside of a Block
{
 int zahl=0;
 zahl+=1;
} // after the block the variable does not exist anymore!
```

Die Programmiersprache C++

Struct

Der Ausdruck struct kann bei der Instanziierung (Deklaration) weggelassen werden

```
struct Schwein{
    int groesse;
    int gewicht;
};
Schwein berta; // in C++ style ...
               // in plain C a struct would be necessary before the
               // Schwein
```

Die Programmiersprache C++

Typenkonversion

Die Typenkonversion zwischen bool und int ist definiert:

```
bool b = 100; // äquivalent zu wahriable=true;
int ergebnis=false;// false->0, true->1
bool bIsOk=(b != ergebnis); // ergibt in diesem Fall true
```

Die Programmiersprache C++

Default Parameter

Man kann bei Funktionen "Default Parameter" angeben. Sind "Default Parameter" angegeben, so kann man die letzten Argumente beim Aufruf der Funktion weglassen. Diese werden dann mit den Defaultwerten belegt:

```
// Funktionsdeklaration
int function(int a, int b=1, int c=2);
// Aufruf mit a=1,b=2,c=2
function(1,2);
```

Die Programmiersprache C++

Referenzen

Mit vorangestelltem & wird die Deklaration einer Variable zur Deklaration einer Referenz:

```
int Zahl=5;
int &verweisAufZahl = Zahl; // Deklaration einer Referenz!
int pointer*;

verweisAufZahl = 6; // damit ändert man die Variable Zahl auf 6
pointer=&verweisAufZahl; // nun zeigt auch die Variable pointer
                         // auf die Adresse der Variablen "Zahl"
```

Die Programmiersprache C++

Besonders häufig werden Referenzen bei Funktionsübergaben verwendet:

```
void addiereUndSubtrahiere(int& erg1, int& er2,int zahl1, int zahl2)
{
    erg1=zahl1+zahl2;
    erg2=zahl1-zahl2;
}
int main(int argc, char** argv)
{
  int a=1,b=1;
  int erg1,erg2;
  addiereUndSubtrahiere(erg1,erg2,a,b); // equivalent: erg1=a+b;
                                        //             erg2=a-b;
}
```

Die Programmiersprache C++

new und delete

Statt wie in ANSI-C mit malloc Speicher zu reservieren bietet C++ die Operatoren "new" und "delete" bzw. "new[]" und "delete[]".

```
int *feld;

feld = new int[100]; // Speicher reservieren

feld[0] = 123;       // Speicher nutzen

delete[] feld;       // Speicher freigeben
```

Die Programmiersprache C++

namespaces

- Wenn in einem grossen Projekt z.B. in zwei Packages jeweils eine Funktion mit gleichem Namen und gleichen Übergabeparametern existieren, so kann ein C wie ein C++ Compiler beim Aufruf dieser Funktion nicht entscheiden, welche der zwei Implementationen verwendet werden soll.
- Um existierenden Code so abzukapseln, dass später doch beide Funktionen aufgerufen werden können, benutzt man Namespaces.
- Der Aufruf wird dann mit dem scope operator :: realisiert.
- Anonyme namespaces sind nur im Modul sichtbar: namespace{...}

Die Programmiersprache C++

```
namespace package1
{
 void f(){
   // implemetation von package1
 }
}
namespace package2
{
 void f(){
   // implemetation von package2
 }
}
int main(void)
{
 package1::f(); // ruft f von package1 auf
 package2::f(); // ruft f von package2 auf
}
```

3 - Klassen, Objekte und Methoden

Die Programmiersprache C++

3 - Klassen Objekte und Methoden

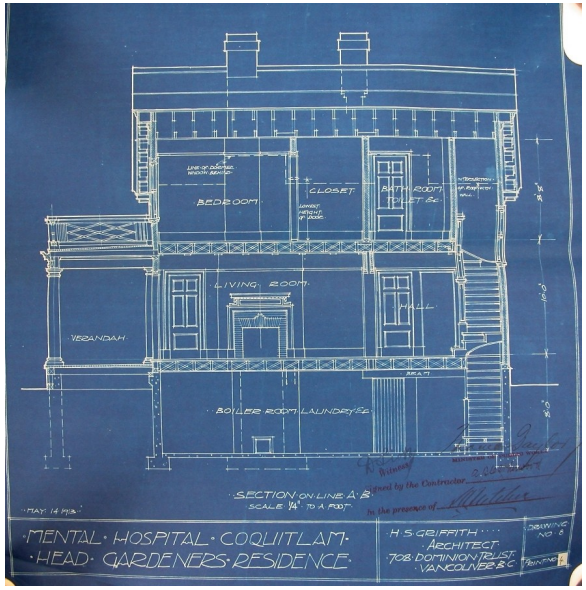
- Was sind Objekte, Attribute und Methoden
- Unterschiede zwischen Class und Struct
- Zugriffsschutz
- Explizit und implizit inline
- Statische Attribute und statische Methoden
- Konstruktoren und Destruktoren
- Der Kopierkonstruktor (Teil 1)
- Weiterleiten von Initialisierungen

Die Programmiersprache C++

Klassen und Objekte

- Eine Klasse ist ein Bauplan während ein Objekt die mit Hilfe des Bauplans erstellte Umsetzung ist.

Klasse



Objekt

Die Programmiersprache C++

Was sind Objekte, Attribute und Methoden

- In der objektorientierten Programmierung werden reale Objekte (z.B. ein Schein) in einer Klasse modelliert um dann als modelliertes Objekt (z.B."berta") instantiiert und genutzt zu werden.
- Attribute des Objektes sind Variablen, die das Objekt beschreiben.
- Methoden des Objektes sind Funktionen, die zum Objekt gehören.

Die Programmiersprache C++

Unterschied zwischen Class und Struct

- In C++ kann "struct" durch "class" ersetzt werden.
- Der wesentliche Unterschied zum struct ist, dass innerhalb einer Klasse normalerweise auch die Funktionen implementiert werden, die das Objekt modellieren und auf die Membervariablen zugreifen.
- Ausserdem sind Variablen, die in einer Klasse implementiert werden nur von deren Funktionen zugreifbar, es sei denn man schreibt das Schlüsselwort `public:` davor.
- Membervariablen sind die Variablen, die innerhalb der Klasse deklariert werden und das Objekt beschreiben.

Die Programmiersprache C++

Zugriffsschutz

- Streng genommen greift man in C++ von ausserhalb einer Klasse nur mit Hilfe sogenannter setter und getter Funktionen auf eine Membervariable der Klasse zu. (z.B. setGewicht oder getGewicht)
- Damit ist die Klasse optimal gekapselt.
- Schreibt man private: vor Membervariablen oder Memberfunktionen, so ist der Zugriff von ausserhalb des instantiierten Objektes nicht möglich.
- Die Variablen und Funktionen sind geschützt. Schreibt man public: vor die Variablen oder Funktionen, so kann auch von auserhalb auf die Funktionen und Variablen zugegriffen werden.

Die Programmiersprache C++

Explizit und implizit inline

- Inline Methoden können vom compiler wie ein Makro an die Stelle des Aufrufs kopiert werden.
- Das spart den Overhead beim Aufruf, braucht aber mehr Programmspeicher.
- Wenn die Methoden innerhalb der Klassendeklaration implementiert sind, sind sie implizit inline.
- Mit dem Schlüsselwort **inline** vor einer Memberfunktion kann eine Funktion auch explizit als inline deklariert werden.

Die Programmiersprache C++

Im Header File: Cschwein.h

```
class CSchwein
{
 private:       // von extern nicht sichtbar
 int gewicht; // Variable oder Attribut
 public:        // auch von ausserhalb sichtbar
 CSchwein(int groesse=0,int gewicht=0){// Konstruktor wird beim instantiieren
                                       // eines Objekts aufgerufen
                                       // wenn im Headerfile implementiert,
                                       // dann automatisch inline (wie Makro)
  this->gewicht = gewicht; // Variablen initialisieren
 }
 ~CSchwein(); // Destruktor wird aufgerufen, wenn Objekt zerstört wird
 void setGewicht(int gewicht);  // Deklaration der Methode,Memberfunktion
 int getGewicht();             // Deklaration der Methode,Memberfunktion
}
```

Die Programmiersprache C++

Im cpp File: Cschwein.cpp

```
CSchwein::~CSchwein(){ // Destruktor wird aufgerufen, wenn Objekt zerstört
   // typischerweise stehen hier die delete Aufrufe
}
void CSchwein::setGewicht(int gewicht)  // Methode, Memberfunktion
{
  this->gewicht = gewicht;
}
int CSchwein::getGewicht()              // Methode, Memberfunktion
{
  return this->gewicht;
}
```

Die Programmiersprache C++

```
// --- Main.cpp File ------------------------------------------

int main(void)
{
 CSchwein berta; // Berta wird instantiiert und ist dann Objekt
                 // oder Instanz der Klasse CSchwein
 berta.setGewicht(200); // Berta ist 200kg schwer
 int gewicht=berta.getGewicht();
}
```

Die Programmiersprache C++

Statische Attribute und statische Methoden

- Methode, die nicht als static deklariert werden bekommen vom Compiler als erstes Argument den **this** pointer, einen Zeiger auf die eigene Instanz der Klasse.
- Statische Methoden bekommen vom compiler keinen **this** pointer.
- Sie können nur auf statische Attribute zugreifen!
- Statische Attribute (Variablen) in einer Klasse existieren genau einmal pro Klasse schon bevor überhaupt eine Instanz der Klasse existiert.

Beispiel: Keine Instanz von einem Konto, aber Zinssatz kann schon festgelegt werden, da setZinssatz() und f_zinssatz als static deklariert sind! Statische Attribute müssen explizit initialisiert werden.

Die Programmiersprache C++

```
class Konto //im Header File
{
 float kontostand;
 static float zinssatz; // für alle Instanzen ist der Zinssatz
                        // gleich, da diese Variable nur einmal
 public:                // im Speicher existiert!
 static void setZinssatz(float zins){
    zinssatz=zins;
 }
 static float getZinssatz(){
    return zinssatz;
 }
};
float Konto::zinssatz; // macht dem Linker die Variable bekannt
int main(void){ // im cpp File
 Konto::setZinssatz(0.3);
 printf("Zinssatz = %f\n",Konto::getZinssatz());
}
```

Die Programmiersprache C++

Konstruktoren und Destruktoren

- Der Konstruktor ist eine Memberfunktion, die aufgerufen wird, wenn eine Instanz eines Objekts instantiiert (bzw. deklariert) wird.
- Dort können Membervariablen initialisiert werden. Wird das Objekt zerstört, z.B. durch beenden des Programms, so wird der Destruktor aufgerufen.
- Im Destruktor sollten alle mit "new" reservierten Speicher mit "delete" wieder freigegeben werden.

Die Programmiersprache C++

```
CSchwein::CSchwein(int groesse=0,int gewicht=0)
          // Konstruktor wird beim instantiieren eines Objekts
          // aufgerufen
{ int *feld;
  feld = new int[100];// Speicher reservieren
  feld[0]=123;        // Speicher initialisieren
  ...
}
CSchwein::~CSchwein() // Destruktor wird aufgerufen, wenn die
                      // Instanz des Objektes zerstört wird
{
delete[] feld;    // Speicher freigeben
}
```

- Sowohl ein Konstruktor als auch ein Destruktor hat keinen Rückgabewert!

Die Programmiersprache C++

Der Kopierkonstruktor (Teil 1)

- Eine Besonderheit stellt der Kopierkonstruktor dar. Es gibt ihn ohne dass man ihn implementiert:

```
Schwein berta(200,100);
Schwein olaf(berta);  // hier kopiert der Copy-Konstruktor
                      // die Daten von berta in die Instanz olaf
Schwein rudolf=berta; // hier kopiert der Copy-Konstruktor
                      //die Daten von berta in die Instanz rudolf

rudolf=olaf; // kopiert alle Variablen von olaf in rudolf, aber
             // mittels des = Operators!

fuettere(olaf); // Call by value: Kopierkonstruktor wird benutzt
rudolf=groesstesSchwein(); // Return value: Kopierkonstruktor benutzen
```

Die Programmiersprache C++

- Der Kopierkonstruktor kann zu unerwarteten Ergebnissen führen, vor allem wenn im eigentlichen Konstruktor mehr gemacht wird, als die Variablen zu initialisieren.
- z.B. werden Pointer auch nur kopiert und zeigen dann auf die gleichen Speicherbereiche wie die Original Instanz
- Man kann sich z.B. einen eigenen Kopierkonstruktor schreiben (siehe Teil 2):

```
Schwein::Schwein(const Schwein& T) {
  // here the copy-constructor is implemented!!!
}
```

- Möchte man verhindern, dass ein Kopierkonstruktor benutzt wird, so definiert man sich eine Kopierkonstruktor Funktion als private: …

Die Programmiersprache C++

Weiterleiten von Initialisierungen

- Werden Objekte (wie im Unteren Beispiel departure und arrival) innerhalb einer Klasse instantiiert und deren Konstruktor aufgerufen, so kann vom Klasseneigenen Konstruktor Daten über die "Member Initialization List" (mit Komma getrennte Liste hinter Konstruktor) weitergeleitet werden

Die Programmiersprache C++

```
class CflightConnection{
 public:
 //   Daten werden an weitere Konstruktoren weitergeleitet
 CflightConnection(const char*depart, const char* arrive,
             int flighnumber):departure(depart), arrival(arrive){
 }
 private:
 Ctime departure; // hier wird der Konstruktor mit den Daten
                  // der Weiterleitung des obigen eigenen
                  // Konstruktors gefüttert.
 Ctime arrival;   // hier wird der Konstruktor mit den Daten
                  // der Weiterleitung des obigen eigenen
                  // Konstruktors gefüttert.
}
```

Die Programmiersprache C++

- Dabei könnten „departure“ und „arrival“ auch const oder eine Referenz & sein:

```
const Ctime departure; // const variablen können nur mit Member
                       // Initialization List oder direkt
                       // initialisiert werden
Ctime &arrival;        // referenz kann nur mit Member
                       // Initialization List oder direkt
                       // initialisiert werden
```

4 - Überladen von Funktionen und Operatoren

4 - Überladen von Funktionen und Operatoren

- Überladen von Funktionen
- Der Kopierkonstruktor (Teil 2)
- Name Mangeling
- Extern “c“
- Überladen von Operatoren

Die Programmiersprache C++

Überladen von Funktionen

- In C war es nicht möglich Funktionen mit gleichem Funktionsnamen aber unterschiedlichen Parametern zu definieren.
- In C++ ist dies möglich:

```
int addiere(int a,int b);          // 1. Funktion
int addiere(float a, float b);     // 2. Funktion
int addiere(char a, double b);     // 3. Funktion
```

- Für den Aufruf entscheidet dabei der Compiler welche der deklarierten Funktionen aufgerufen wird. Z.B. ruft addiere(1.2f,1.3f) die 2.Funktion auf.

Die Programmiersprache C++

Überladen von Funktionen

- So kann auch der Konstruktor und der Kopierkonstruktor überladen werden.

```
class MyClass
{
 MyClass(); // constructor
 MyClass(int a); // overloaded constructor
}
```

Die Programmiersprache C++

Der Kopierkonstruktor (Teil 2)

- Der Kopierkonstruktor wird aufgerufen, sobald eine Instanz einer Klasse einer anderen Instanz der selben Klasse zugewiesen wird.
- Das kann entweder durch den = Operator geschehen oder bei der Instanziierung der Klasse:

```
cSchein Berta;

cSchein Heinrich(Berta);   // Kopiert Berta nach Heinrich

cSchwein Heinrich = Berta; // nutzt auch den Kopierkonstruktor
```

Oder

```
Heinrich = Berta; // Kopiert Berta nach Heinrich mittels
                  // der Operator=() Funktion
```

Die Programmiersprache C++

- Wurde kein Kopierkonstruktor implementiert, so wird der Standard Kopierkonstruktor verwendet. Dieser kopiert einfach die Inhalte aller Variablen der Quelle in die entsprechenden Variablen des Zielobjektes.
- Mit Vorsicht zu genießen ist die Kopie von Zeigern.
- Im obigen Beispiel wird Schwein H`einrich` als eine Kopie von `Berta` erzeugt. Falls `Berta` Zeiger auf eigene Variablen beinhaltet, so werden diese Zeiger auch kopiert.
- Sie zeigen dann in der Kopie (in diesem Fall Zeiger von `Heinrich`) immer noch auf Variablen von `Berta`.

Die Programmiersprache C++

- Eine Implementation eines Kopierkonstruktors kann beispielsweise folgendermassen aussehen:

```
class Date
{
 public:

 int tag,monat,jahr;

 Date();                  // Konstruktor
 ~Date();                 // Destruktor
 Date(const Date& d){     // Kopierkonstruktor
    tag   = d.tag;
    monat = d.monat;
    jahr  = d.jahr;
 }
}
```

Die Programmiersprache C++

NameMangeling

- Der Compiler führt ein NameMangling durch, d.h. die Funktion wird mit den aufgerufenen Datentypen in einem String codiert...z.B. addiere@@@int@@@int.
- Da die Syntax des NameMangling von Compiler zu Compiler unterschiedlich ist, kann das zu Problemen bei Bibliotheken führen.
- Nutzt man `extern"C"` so können Funktionen nicht überladen werden, aber ein NameMangling findet auch nicht statt.

Die Programmiersprache C++

NameMangeling

- Der Compiler führt ein NameMangling durch, d.h. die Funktion wird mit den aufgerufenen Datentypen in einem String codiert...z.B. addiere@@@int@@@int.
- Da die Syntax des NameMangling von Compiler zu Compiler unterschiedlich ist, kann das zu Problemen bei Bibliotheken führen.
- Nutzt man `extern"C"` so können Funktionen nicht überladen werden, aber ein NameMangling findet auch nicht statt.

Die Programmiersprache C++

extern ”c“

```
extern "c"
{
 int addiere(int a, int b);
}
```

- So ist es möglich auch C oder C++ Funktionen zu nutzen, die mit einem anderen Compiler geschrieben wurden.

Die Programmiersprache C++

Überladen von Operatoren

- Nicht nur Funktionen können überladen werden, sondern auch Operatoren.
- Die Syntax einer Funktion, die einen Operator überlädt lautet wie folgt:

Außerhalb einer Klasse:

```
datentyp operator+(datentyp leftHandSide, datentyp rightHandSide)
```

Innerhalb einer Klasse:

```
datentyp operator+(datentyp rightHandSide)
```

Das leftHandSide Argument, also die links vom Operator stehende Variable, ist dann das Objekt, auf das gerade der this pointer zeigt!

Die Programmiersprache C++

```
class Vector{
  const int SIZE=3;
  double v[SIZE];
public:
  Vector(double x,double y,double z)
  {
  v[0]=x;   v[1]=y;    v[2]=z;
  }
  // --------------- Operator überladen ------------- //
  Vector operator+(const Vector& v2){
    Vector t(0,0,0);
    for (int i=0;i<SIZE;i++){
       t.v[i]=v[i]+v2.v[i];
    }
    return t;
  }
};
// ---------------------- main ---------------------//
int main(void) {
 vector v1(1,2,3), v2(4,6,5), v3(0,0,0);
 v3 = v1+v2; // funktioniert nur weil der Operator+ überladen ist
}
```

Die Programmiersprache C++

Überladen von Operatoren: Bedingungen

- Nicht überladen werden können:

```
?:      Bedingter Ausdruck
sizeof  Sizeof Operator
.       Punktoperator
::      Scopeoperator
.*      Komponentenzeiger
```

Die Programmiersprache C++

Überladen von Operatoren: Bedingungen

- Mindestens ein Operand muss abstrakter (z.B. selbst definierter) Datentyp sein.
- Die Regeln der C Operatoren sollten eingehalten werden (z.B.Auswertereihenfolge).
- Die Anzahl der Parameter ist vorgegeben.
- Überlädt man beipielsweise den = Operator, so sollte man darauf achten, dass die operator= Methode auch einen Rückgabewert hat (in diesem Fall *this), denn sonst ist obj1=obj2 möglich, nicht aber obj1=obj2=obj3.

Die Programmiersprache C++

Überladen von Operatoren: Bedingungen

- Für den Fall, dass der lefthand Operator nicht vom Typ der Klasse ist, gibt es keine Möglichkeit innerhalb der Klasse eine Operatorüberladung zu implementieren.
- Für einen solchen Fall kann man die Operatorüberladung auch ausserhalb der Klasse schreiben.
- Die Klasse, von dessen Typ der erste operator ist, wird dann normalerweise als friend deklariert, und kann somit auch auf private: Elemente zugreifen.
- Auch hier gilt, dass man für call by value und für den Rückgabewert einen Kopierkonstruktor braucht.

Die Programmiersprache C++

- Beispiel: 2 Objekte verschiedener Klassen sollen addiert werden:

```
class DoubleVector{
public:
  enum {SIZE=3};
  double v[SIZE];
  DoubleVector(double x,double y,double z){
    v[0]=x;   v[1]=y;    v[2]=z;
  }
};
class IntVector
{
public:
  enum {SIZE=3};
  int v[SIZE];
  IntVector(double x,double y,double z){
    v[0]=x;   v[1]=y;    v[2]=z;
  }
};
```

Die Programmiersprache C++

- Manchmal steht die Überladung auch ausserhalb der Klasse

```
// the lefthandside operator is not of type doubleVector
// so the function must be implemented outside of the class
// intVector
// ---------------- Operator overloading -------------------
DoubleVector operator+(const IntVector& v1, const DoubleVector& v2){
  DoubleVector t(0.0,0.0,0.0);
  for (int i=0;i<DoubleVector::SIZE;i++){
     t.v[i] = v1.v[i]+v2.v[i];
  }
  return t;
}
// --------------------- main ----------------------------
int main(void) {
 intVector v1(4,6,5);
 DoubleVector v2(1.1, 2.2, 3.3);

 DoubleVector v3 = v1+v2; // the function operator+() is called
}
```

Die Programmiersprache C++

Man kann z.B. auch einen Typecast überladen/ einbauen:

```
class CBruch
{
public:
 CBruch(int zaehler, int nenner) : m_zaehler(zaehler),
                                   m_nenner(nenner)
 {}// Konstruktor mit initialisierung von m_nenner und m_zaehler

 operator float()   // typecast implementieren!!!
 {
   return ((float)(m_zaehler)/(float)(m_nenner));
 }
 operator double()   // typecast implementieren!!!
 {
   return ((double)(m_zaehler)/(double)(m_nenner));
 }
int m_zaehler,m_nenner;
}
```

Die Programmiersprache C++

- Zusätzlich könnten die Operatoren überladen werden, die das Bruchrechnen übernehmen:

```
CBruch operator*(CBruch bruch)
 {
   CBruch ergebnis;
   ergebnis.m_zaehler=this->m_zaehler*bruch.m_zaehler;
   ergebnis.m_nenner=this->m_nenner*bruch.m_nenner;
   return ergebnis;
 }
 CBruch operator+(CBruch bruch)
 {
   CBruch ergebnis;
   ergebnis.m_zaehler=this->m_zaehler*bruch.m_nenner +
                      bruch.m_zaehler*this->m_nenner;
   ergebnis.m_nenner=this->m_nenner*bruch.m_nenner;
   return ergebnis;
 }
```

5 - Vererbung

Die Programmiersprache C++

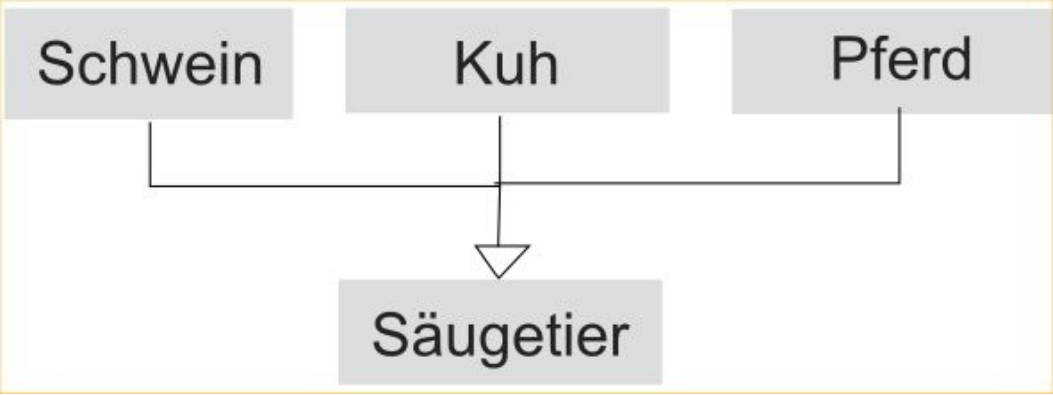
5 - Vererbung

- Grundlagen
- Überschreibung von Funktionen
- Aufruf der Konstruktoren
- Friend
- Virtuelle Funktionen und Polymorphismus
- Zugriffsrechte
- Nachtrag Kopierkonstruktor (Teil 3)

Die Programmiersprache C++

Grundlagen

- Besonders effizient programmieren kann man mit Hilfe von Abstraktion. Wenn zum Beispiel "Kuh", "Schwein" und "Pferd" = "Saeugetiere" sind könnte man zunächst die Klasse Säugetiere implementieren und davon "Kuh", "Schwein" und "Pferd" ableiten.



Die Programmiersprache C++

- In anderen Worten: Man kann die Attribute und Methoden, die für alle Säugetiere gelten an die einzelnen Saeugetiere vererben.
- Einzelne Methoden, wie z.B. das "grunzen()" des Schweines, könnten dann für das vom Säugetier abgeleitete Schein extra implementiert werden
- Konstruktoren und Destruktoren werden nicht vererbt

Die Programmiersprache C++

```
class CSaeugetier // Basisklasse
{
 friend Class CLieberGott; // Freunde können auf alle Attribute
          // der Basisklasse zugreifen, auch auf die privaten!
 public:
 void fressen();
 void wachsen();
 protected:       // auf protected: Attribute kann von der
                  // abgeleiteten Klasse zugegriffen werden
 int groesse;
 int gewicht;
 private:         // auf private: Attribute kann von der abgeleiteten
                  // Klasse nicht zugegriffen werden
}
class Kuh : public CSaeugetier  // von CSaeugetier  abgeleitete Klasse
{
 public:
 void muhen();
}
```

Die Programmiersprache C++

Und noch das Schwein und das Pferd...

```
class Schwein : public CSaeugetier    // von CSaeugetier
                                      // abgeleitete Klasse
{
 public:
 void grunzen();
}

class Pferd : public CSaeugetier     // von Csaeugetier
                                     // abgeleitete Klasse
{
 public:
 void wiehern();
}
```

Die Programmiersprache C++

Überschreiben von Funtionen

- Zusätzlich kann man in der abgeleiteten Klasse eine Funktion der Basisklasse überschreiben.
- Das heißt man kann eine Funktion mit gleichem Namen und gleichen Parametern implementieren.
- Bei einer Instanz der abgeleiteten Klasse wird dann diese Funktion statt der überschriebenen Basisklassenfunktion aufgerufen.

Die Programmiersprache C++

Aufruf der Konstruktoren

- Bei Instanziierung eines Objektes einer abgeleiteten Klasse wird zuerst der parameterlose Konstruktor der Basisklasse aufgerufen, dann der Konstruktor der abgeleiteten Klasse.
- Bei mehrfach vererbten Klassen wird zuerst der parameterlose Konstruktor der Basisklasse, dann der der abgeleiteten Klasse, dann der der davon abgeleiteten Klasse... bis zum Aufruf des Konstruktors der aufgerufenen Klasse (der auch Parameter beinhalten kann).

Die Programmiersprache C++

Friend

- Im obigen Beispiel wird die Klasse "CLieberGott" als "friend" deklariert.
- Dadurch kann jede Methode der Klasse "CLieberGott" alle Attribute der Klasse "CSaeugetier" ändern und lesen, auch wenn sie **private** sind!

  Anmerkung: Mit **friend:** definierte Klassen werden nicht als friend mit vererbt.

Die Programmiersprache C++

Virtuelle Funktionen und Polymorphismus

- Arbeitet man mit Zeigern vom Typ Basisklasse "CSaeugetier" auf eines seiner abgeleiteten Klasseninstanzen z.B. "Schwein", so kennt der Compiler nicht die Funktion "grunzen()", da es keine Funktion der Basisklasse ist.
- Wenn man die Methode "grunzen()" als virtuell in der Klasse CSaeugetier deklariert und imlementiert, so kennt der Compiler auch diese Funktion. (z.B. virtual void grunzen();)
- Schreibt man in der Basisklasse hinter einer virtual void Funktion()=0, so muss diese Funktion in jeder abgeleiteten Klasse implementiert werden! Man spricht dann von rein virtuellen Funktionen.
- Eine Klasse, die ein oder mehrere rein virtuelle Methoden enthält kann nicht instanziiert werden

Die Programmiersprache C++

- **Anmerkung:** Mit virtuellen Methoden werden Virtuelle Methodentabellen angelegt und für jede Instanz der Klasse Zeiger auf die Virtuelle Methodentabelle. So erkauft man sich die Flexibilität der virtuellen Methoden besonders bei vielen Instanzen mit Speicherplatz und Laufzeit.

- **Anmerkung:** Klassen, die mindestens eine virtuelle Funktion mit "=0" implementieren heißen abstrakte Klassen!

Die Programmiersprache C++

```
class CSaeugetier
{
 private:
 int groesse;
 int gewicht;

 public:
 void fressen();
 void wachsen();
 virtual void grunzen();
 virtual void wiehern();
 virtual void muhen();
}
```

Die Programmiersprache C++

Zugriffsrechte

- public: Auf public Member kann von überall drauf zugegriffen werden!
- protected: Auf protected Member kann von der Klasse, in der sie deklariert sind und deren abgeleiteten Klassen zugegriffen werden!
- private: Auf private Member kann nur von der Klasse zugegriffen werden, in der sie deklariert sind!
- Hinter der abgeleiteten Klasse steht mit : getrennt die Art der Ableitung. Die Art kann private, protected oder public sein. Damit kann man die geerbten Funktionen und Variablen noch mal extra schützen. Mit public werden die Zugriffsrechte der Basis beibehalten, mit protected sind sie mindestens protected, mit private sind sie alle private!

Die Programmiersprache C++

Ableitung	Privileg der Basisklasse	Privileg der abgeleiteten Klasse
:public	public:	public
	protected:	protected
	private:	Kein Zugriff
:protected	public:	protected
	protected:	protected
	private:	Kein Zugriff
:private	public:	private
	protected:	private
	private:	Kein Zugriff

Nachtrag Kopierkonstruktor (Teil 3)

- Wenn man bei der Instanziierung einer Basisklasse die von ihr abgeleiteten Klassen Instanz zuweist, so hat der Compiler alle Daten zur Verfügung um die Basisklasse zu füllen. Andersrum würde der Compiler eine Fehlermeldung ausgeben, da bei der abgeleiteten Klasse mehr Attribute sein könnten, die dann nicht aus der Basisklasse kopiert werden können.

```
// so geht's...
CAbgeleiteteKlasse abgeleitet;
CBasisKlasse basis=abgeleitet; // ruft den Kopierkonstruktor
// und kopiert alle Attribute die die Basisklasse benötigt
```

6 - Templates

Die Programmiersprache C++

6 - Templates

- Grundlagen
- Funktionstemplates
- Klassentemplates

Die Programmiersprache C++

Grundlagen

- Manchmal wünscht man sich Funktionen, die mit einem beliebigen Datentyp arbeiten. Diese Möglichkeit bieten Templates (Schablonen)
- Es gibt Funktionstemlates und Klassentemplates

Die Programmiersprache C++

Funktionstemplates

- Hier ein Beispiel einer Funktion, die das Maximum zweier Instanzen zurückgibt. Wird die Templatefunktion mit verschiedenen Datentypen aufgerufen, so erzeugt der Compiler die Funktion für jeden der genutzten Datentypen neu.

```
template <typename T>
T max(T parameter1, T parameter2)
{
 if (parameter1>parameter2){
    return parameter1;
 }
 else{
    return parameter2;
 }
}
```

Werden mehr als ein Templatedatentyp verwendet, so kann man diese mit Komma innerhalb der spitzen Klammern abtrennen.

```
template <typename T, typename U>
```

Die Programmiersprache C++

Funktionstemplates

- Der Aufruf dieser Funktionstemplates kann so aussehen wie bisher:

```
int a = max(5,7);
```

  In dem Fall wird 5 und 7 als Zahlen des Typs Integer interpretiert.

- Man kann den Datentyp jedoch auch explizit angeben:

```
int a = max <int> (5,7);
```

Die Programmiersprache C++

Klassentemplates

- Bei Klassentemplates können Attribute der Klasse und Funktionsparameter und Rückgabewerte der Methoden vom Template-Typ sein.
- Templateklassen können vererbt werden und können virtuelle Funktionen enthalten!

Die Programmiersprache C++

```
template <class T>
class CMathVector
{
public:
  enum{MAX=3}
  T mathVector[MAX];

  T mittelwert(void)
  {
    T sum=0;
    for (int i=0;i<MAX;i++){
        sum+=mathVector[i];
    }
    return sum/MAX;
  }
}
```

```
// im cpp File Instantiierung:

int main(void)
{
CMathvector <double> dblVekt;
double mittel;

dblVekt.mathVector[0] = 1.234;
dblVekt.mathVector[1] = 0.5643;
dblVekt.mathVector[2] = 0.543;
mittel = dblVekt.mittelwert();
return 0;
}
```

Die Programmiersprache C++

```
// alternative Implementierung
// ausserhalb der Klassendefinition
template <class T>
T CMathVector<T>::mittelwert(void)
{
  T sum;
  for (int i=0;i<MAX;i++){
      sum+=mathVector[i];
  }
  return sum/MAX;
}
```

7 - Streams

Die Programmiersprache C++

7 - Streams

- Grundlagen
- Datei schreiben
- Datei lesen

Die Programmiersprache C++

Grundagen

- Bindet man <iostream> ein, so kann man Zeichen von der Tastatur einlesen und auf dem Bildschirm ausgeben. Dazu nutzen wir den << und den >> Operator:

```
#include <iostream>
using namespace std;
int main(void)
{
 int zahl;
 cout << "geben Sie eine Zahl ein:"; // ersetzt print und erkennt Datentyp
 cin >> zahl;     // liest Zahl von der Tastatur
 cout << "Sie haben folgende Zahl eingegeben:" << zahl << endl;
                  // endl ist Zeilenumbruch und forciert das ausgeben
                  // bzw. leeren des Ausgabebuffers
 return 0;
}
```

Die Programmiersprache C++

Datei schreiben

- In C++ gibt es eine Stream Erweiterung, die für das File-Handling zuständig ist:

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
int main(void)
{
 ofstream outFile; // für Dateien, die geschrieben werden
 outFile.open("test.txt",ios::out); // öffnen der Datei zum schreiben
 if (!outFile){
    cout << "Ausgabedatei kann nicht geöffnet werden!\n\n";
 }
 else{
   outFile << "Dies ist ein BeispielTEXT" << flush;
 }
 outFile.close();
}
```

Die Programmiersprache C++

Datei lesen

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
int main(void){
 string inp;
 ifstream inFile;  // für Dateien, die geöffnet werden
 inFile.open("test.txt",ios::in); // öffnen der Datei zum lesen
 if (!inFile){
    cout << "Eingabedatei kann nicht geöffnet werden!\n\n";
 }
 else{
    while(inFile.getline(inp,1)){
       cout << inp << flush; // flush bedeutet Buffer lehren
    }
 }
 inFile.close();
}
```

8 - Strings

8 - Strings

- Grundlagen
- Operatoren und Methoden
- Beispiel

Die Programmiersprache C++

Grundlagen

- Der Vorteil von String zum ursprünglichen C-Sting oder array von char ist:
  - \0 kann im String vorkommen ohne das es das Ende des Strings markiert!
  - es gibt zahlreiche Möglichkeiten, mit denen man Strings manipulieren kann z.B. kann man Strings addieren
  - Man kann den String mit rezise vergrößern
  - alle Methoden sind sicher!!!
- Um die String Klasse benutzen zu können muss sie zunächst includiert werden.

Die Programmiersprache C++

Operatoren und Methoden

- Es gibt zahlreiche Operatoren auf Strings:
  - + konkateniert Strings
  - == vergleicht Strings auf Gleichheit
  - != vergleicht Strings auf Ungleichheit
  - <, > vergleicht Strings im Bezug auf lexikalische Reihenfolge
  - <=, >= vergleicht Strings im Bezug auf lexikalische Reihenfolge
- Es gibt zahlreiche Funktionen
  - length() gibt die Länge des Strings zurück
  - insert(n,s) fügt einen String s an Position n ein
  - erase(p,n) entfernt n Zeichen ab Position p
  - find(s) sucht den String s und gibt die Position zurück
  - ...

Die Programmiersprache C++

Beispiel

```
#include <string>
#include <iostream>
using namespace std;

int main(void){
 string nachname("Walter");
 string vorname("Dieter");
 string name = vorname+" "+nachname;
 if (vorname=="Otto")
    cout << name << endl;
 if (vorname.find("ie")>0){
    cout << "Die Buchstaben ie befinden sich an Stelle "
         << vorname.find("ie") << " im Vornamen " << vorname
         << "." << endl;
 }
 return 0;
}
```

9 - Casting und Fehlerbehandlung

Die Programmiersprache C++

9 - Casting und Fehlerbehandlung

- Casting
- Fehlerbehandlung in C
- Fehlerbehandlung in C++

Die Programmiersprache C++

Casting

Es gibt in C++ weitere Cast Operatoren:

```
static_cast<xyz*>(p)              // prüft Verwandschaftsbeziehung
reinterpret_cast<char*>(&org);    // hier kann zu etwas komplett anderem gecastet
                                  // werden.
const_cast<char*>(q)='B';         // selbst konstanten können überschrieben werden!
dynamic_cast<CKlasse*>(pA)        // Prüft zur Laufzeit ob cast korrekt und gibt
                                  // sonst NULL zurück
```

- Beim dynamic_cast wird eine vtable (virtuelle Methodentabelle) in das Programm eingebunden. Damit ist der Dynamic_cast für embedded Systeme ungeeignet.

Die Programmiersprache C++

Fehlerbehandlung in C

Es gibt zweierlei Arten der Fehlerbehandlung:

In C kann man Programme bei Fehlern mittels assertion kontrolliert beenden:

```
#include <assert.h>
...
assert(true);  // führt keine Fehlerbehandlung aus
assert(false); // führt eine Fehlerbehandlung aus
...
assert(denominator!=0); // Fehler wenn denominator == 0
result = nominator/denominator;
```

Die Programmiersprache C++

Fehlerbehandlung in C++

Mittels Exceptions try-catch:

```
#include <iostream>
 int result, nominator=0, denominator=0;
 try
 {
    result = nominator/denominator;
 }
 catch(...) // statt ... kann hier auch ein Ausnahmetyp stehen
 {  // springt mittels einer exception hier rein um den Fehler zu behandeln!
    std::cout << "Division durch Null!"<<flush;
 }
```

Die Programmiersprache C++

10 - C++ 11

Die Programmiersprache C++

10 - Der neue C++ Standard

- Neue Schlüsselworte
- Die range basierte For Schleife
- funktionale Programmierung (Lambda Funktionen)
- generische Programmierung (zusätzliche Möglichkeiten)
- Multithreading
- Atomare Datentypen
- Asynchrone Aufgaben
- Reguläre Ausdrücke

Die Programmiersprache C++

Neue Schlüsselworte

Um die Programmierung zu vereinfachen wurden neue Schlüsselworte eingeführt:

Beim neuen Datentyp **auto** wird der eigendliche Datentyp erst bei der Initialisierung der Variablen definiert. Das macht besonders bei Templates Sinn.

```
auto var = x + y; // sind x und y Objekte einer Klasse,
                  // so ist auch var Objekt dieser Klasse
```

Um den Datentyp zu ermitteln liefert decltype den Datentyp zurück.:

```
typedef decltype(var) myDataType;
myDataType var2; // var2 hat den selben Datentyp wie var
```

Die Programmiersprache C++

Die range basierte For- Schleife

Um sich wenig um die Größe von Feldern kümmern zu müssen gibt es nun die range basierte for Schleife:

```
std::vector<int> vec({ 1, 2, 3, 4, 5, 6 });
for (const auto x : vec){
     std::cout << x << std::endl;
}
```

Die Schleife iteriert über alle Elemente vec.

Die Programmiersprache C++

Funktionale Programmierung

Ab C++11 sind auch die sogenannten Lambda-Expressions der funktionalen Programmierung möglich.

```
std::vector<int> v = {15, -2, 4, -1};
std::sort(v.begin(), v.end(), [](int a, int b) {
   return abs(a)<abs(b); });
```

Innerhalb der Funktionen kann man zwar Membervariablen auslesen und sich Hilfsvariablen deklarieren und benutzen. Es können aber keine Membervariablen beschrieben werden.
In Eckigen Klammern [] wird die Bindung zu den lokalen Variablen angegeben, in runden Klammern () die Parameter

Die Programmiersprache C++

Generische Programmierung

Als generischer Programmiererweiterung werden ab C++11 folgende eingeführt, hier aber nicht näher beschrieben:

- Templates, die beliebig viele Parameter annehmen
- Zusicherungen, die zur Compilezeit ausgewertet werden
- Konstanten, die zur Compilezeit ausgewertet werden
- Aliase Templates, um einfache Namen für teilweise gebundene Templates zu definieren

Multithreading

Es gibt im Wesendlichen 3 Varianten einen Thread zu erzeugen:

1. mit Hilfe einer Funktion
2. mit Hilfe eines Objektes
3. mit Hilfe einer Lambda Expression

Die Programmiersprache C++

Multithreading (mit Funktion)

```
#include <iostream>
#include <thread>
void helloFunction() {
  std::cout << "C++11 Thread with function"<< std::endl;
}
int main() {
 std::thread t1(helloFunction);
 t1.jion();
}
```

Die Programmiersprache C++

Multithreading (mit Objekt)

```
#include <iostream>
#include <thread>
class HelloFunctionObject {
  public:  void operator()() const {
        std::cout << "C++11 Thread with Object" << std::endl;
  }
}
int main() {
 HelloFunctionObject helloFunctionObject;
 std::thread t2(helloFunctionObject);
 t2.jion();
}
```

Die Programmiersprache C++

Multithreading (mit Lambda Ausdruck)

```
#include <iostream>

#include <thread>

int main() {

 std::thread t3([]{std::cout << "C++11 Thread with lambda
   function" << std::endl;});

 t3.jion();

}
```

Die Programmiersprache C++

Atomare Datentypen

- Atomare Datentypen sind Datentypen, die nicht aus anderen Datentypen zusammengesetzt sind.
- Auf diese Datentypen können atomare Operationen ausgeführt werden.
- Atomare Operationen sind Operationen, die nicht unterbrochen werden können.
- Es gibt bereits viele atomic Datentypen, die einen äquivalenten build in Typ haben: z.B. atomic_bool, atomic_char, atomic_int etc.
- Mit Hilfe des `std::atomic-Klassen-Template` ist es möglich eigene atomare Typen zu implementieren.

Die Programmiersprache C++

Asynchrone Aufgaben

- Sie sind unter dem Namen Futures bekannt
- Der Compiler kümmert sich selbst um die Verwaltung und prüft ob es z.B. sinnvoll ist einen neuen Thread zu erzeugen
- std::async liefert ein std::future Objekt zurück, von welchem man zu einem späteren Zeitpunkt das Ergebnis abrufen kann
- Sollte das Ergebnis noch nicht vorliegen blockiert der Thread

Die Programmiersprache C++

Asynchrone Aufgaben(Beispiel)

```
#include <future>
int product(int a, int b) {
 return a*b;
}
int main() {
 int a = 20;
 int b = 10;
 std::future<int> futureSum = std::async([=](){return a+b;});
 auto futureProduct = std::async(&product,a, b);
 std::cout << futureSum.get() << std::endl;
 std::cout << futureProduct.get() << std::endl;
}
```

Die Programmiersprache C++

Reguläre Ausdrücke

Mit Hilfe der regukären Ausdrücke können Zeichenfolgen durchsucht und ersetzt werden.

Ab C++11 werden reguläre Ausdrücke der folgenden Grammatiken unterstützt:

- ECMAScript
- basic
- extended
- awk
- grep
- egrep