Christian-Albrechts-Universität zu Kiel

Rechenzentrum

C++ als Erweiterung von C

Simone Knief – Rechenzentrum der CAU

Überblick

- Einführung
- Ojektorientierte Konstrukte
- Unterschiede zwischen C und C++
- Strukturen in C++
- Member-Funktionen
- Objekte und Klassen
- Vererbung

Geschichte von C++

- ab 1980 entwickelt von Bjarne Stroustrup unter dem Namen "C mit Klassen"
- 1983 wird die neu entwickelte Sprache in C++ umbenannt
- 1985 erster kommerzieller Compiler wird entwickelt
- 1998 Standard der Sprache C++ wird von der ISO (International Organization for Standardization) genormt
- 2003: Überarbeitung und Erweiterung von C++

- Weiterentwicklung von ANSI-C
- enthält die Funktionalität von C mit Erweiterungen für die objektorientierte Programmierung
- unterstützt prozedurale und objektorientierte Programmierung
- ermöglicht die Entwicklung und Wartung von sehr großen Programmpaketen mit mehr als 10.000 Codezeilen

- C gehört in die Kategorie der prozeduralen Programmierung
- ein praktisch sequentieller Programmablauf
- Programm wird in kleine Teilaufgaben zerlegt
- Teilprogramme werden als Funktionen oder Prozeduren bezeichnet
- einzelne Programmteile sind wieder verwendbar

- wird mit OOP abgekürzt
- nutzt Objekte, die über bestimmte Eigenschaften und Funktionen verfügen
- Eigenschaften und Funktionen eines Objektes bilden eine eigenständige Einheit
- Objekt:
  - unabhängiger und wieder verwendbarer Codeabschnitt
  - führt eine spezifische Aufgabe aus
  - speichert festgelegte Daten
- Klassen fassen Objekte mit den gleichen Eigenschaften zusammen
- objektorientiertes Programmieren erfordert objektorientiertes Denken

- 4 Säulen der objektorientierten Programmierung
- Objekte werden durch folgende Hauptmerkmale definiert
  - Polymorphismus
  - Kapselung
  - Vererbung
  - *Wiederverwendbarkeit*

Polymorphismus

- Vielgestaltigkeit
- Programm kann sich automatisch anpassen
- macht den Code wieder verwendbarer
- Überladen von Funktionen
  - Programm enthält mehrere Funktionen mit dem gleichen Namen
  - Beispiel: Kreis zeichnen:
    - 3 Punkte auf Umfang angeben
    - Mittelpunkt und Radius angeben
    - Mittelpunkt und 1 Punkt auf Umfang angeben
  - Beispiel: Quadrat zeichnen
    - x- und y-Wert eines Punktes und die Breite wird angegeben
    - x- und y-Wert von 2 Punkten wird angegeben
  - in C nicht korrekt
  - In C++ erlaubt
  - Programm bestimmt beim Aufruf welche Funktion verwendet wird
- Überladung von Operatoren möglich

Kapselung

- es lassen sich in sich abgeschlossene Objekte erzeugen
- Verschiedene Objekte sind unabhängig voneinander
- Kapselung von Funktionalität und Daten
- Objekte können leichter wieder verwendet werden
- realisiert die Funktionalität einer Black-Box
- Anwender muss nicht wissen wie eine Berechnung durchgeführt wird
- Kenntnis von Ein- und Ausgabeinformation reicht aus

Vererbung

- Erzeugung neuer Objekte, die die Eigenschaft vorhandener Objekte erweitern
- Ein Objekt übernimmt alle Datenkomponenten eines bereits definierten Objektes und erweitert oder modifiziert diese

- Beispiel: Quadratobjekt
  - x- und y-Koordinaten einer Punktes
  - Länge der Seite
  - Zeichen mit dem das Quadrat gezeichnet werden soll
  - Funktion, die die Fläche eines Quadrates berechnet

  → per Vererbung kann Quadratobjekt zu einem Würfelobjekt erweitert werden

Objektorientierter Code (Beispiel)

```
// C++-Programm mit den Klassen quadrat und wuerfel
#include <iostream.h>

// Einfache quadrat-Klasse---> Kapselung
class quadrat {
public:
quadrat();
quadrat(int);
int laenge;
long raum();
int zeichnen();
};

// Einfache wuerfel-Klasse, die von quadrat erbt -->
      Vererbung
class wuerfel: public quadrat {
public:
wuerfel( int );
long raum();
};
```

```
// Konstruktor für quadrat ---> Polymorphismus
quadrat::quadrat()
{
laenge = 4;
}


// Überladener Konstruktor für quadrat
quadrat::quadrat( int init_laenge )
{
laenge = init_laenge;
}
```

- Welche Änderungen und Erweiterungen gibt es gegenüber ANSI-C ?
- Wie gebe ich Informationen auf dem Bildschirm aus ?
- Wie lese ich Daten von der Tastatur ein ?
- Wie arbeite ich mit überladenen Funktionen ?
- Wie übergebe ich Standardwerte als Funktionsparameter ?
- Was sind Inline-Funktionen ?

Änderungen gegenüber ANSI-C

- Kommentare können auf 2 Arten gekennzeichnet werden
- unterschiedliche Platzierung von Variablendeklarationen
- unterschiedliche Prinzipien der Ein-und Ausgabe von Informationen
- Funktionsdeklaration ohne Angabe eines Rückgabewertes
- Funktionen kann eine variable Anzahl von Parametern übergeben werden
- Funktionen können überladen werden
- Inline-Funktionen

- Kommentare in C
  /* Kommentartext */

- Kommentare in C
  /* Kommentartext */

- Kommentare in C++
  - /* Kommentartext*/

  - einzeilige Kommentare mit einen // kennzeichnen
  - alles ab // bis zum Zeilenende ist ein Kommentar
  - Beispiele:

```
int i;  // Laufzeitvariable
float summe;     // Summe der Schleife
```

```
/* Ausgabe in einem  C-Programm */
#include <stdio.h>
int main ()
{
     printf("Viel Spass \n");
     return 0;
}
```

```c
/* einfaches C-Programm */
#include <stdio.h>
int main ()
{
  printf("Viel Spass \n");
     return 0;
}
```

```cpp
/* einfaches C++-Programm */
#include <iostream>
using namespace std;
int main()
{
  cout << "Viel Spass \n";
return 0;
}
```

C-Programme übersetzen

- Code hat die Endung .c
- Programmübersetzung:
  - gcc –o bsp.exe bsp.c
- Programmaufruf
  - ./bsp.exe

C++-Programme übersetzen

C-Programme übersetzen

- Code hat die Endung .c
- Programmübersetzung:
  - gcc –o bsp.exe bsp.c
- Programmaufruf
  - ./bsp.exe

C++-Programme übersetzen

- Code hat die Endung .cpp
- Programmübersetzung:
  - g++ -o bsp.exe bsp.cpp
- Programmaufruf
  - ./bsp.exe

Ausgabe von Variablen

```
/* Variablen in C ausgeben */
#include <stdio.h>
int main()
int x=100, y=50, summe=0;
{
  summe=x+y;
  printf("Summe von %d und %d ist %d \n", x,y,summe);
  return 0;
}
```

Ausgabe von Variablen in C++

```
/* Variablen in C++ ausgeben */
#include <iostream>
using namespace std;
int main()
{
int x=100, y=50, summe=0;
  summe=x+y;
  cout << "summe von " << x << " und " << y << " ist" << summe <<"\n";
return 0;
}
```

Ausgabe von Variablen in C++

```
/* Variablen in C++ ausgeben */
#include <iostream>
using namespace std;
int main()
{
int x=100, y=50, summe=0;
  summe=x+y;
  cout << "summe von " << x << " und " <<
      y << " ist" << summe <<"\n";
return 0;
}
```

- Programmübersetzung:
  - g++ -o cout2.exe cout2.cpp
- Programmausführung:
  - ./cout2.cpp
- Programmausgabe:
  summe von 100 und 50 ist 150

```
// cout mit unterschiedlichen Datentypen verwenden
#include <iostream>
using namespace std;
int main()
{
  int   an_int    = 123;
  float a_float   = 123.456;
  char  a_char    = 'A';
  char a_string[]  = "Ein String";
  bool  a_boolean = true;

  cout << "\n";
  cout <<    "Ein int:    " << an_int    << “\n”;
  cout <<    "Ein float:  " << a_float   << “\n”;
  cout <<    "Ein char:   " << a_char    << “\n”;
  cout <<    "Ein string: " << a_string  << “\n”;
  cout <<    "Ein bool:   " << a_boolean << “\n”;

  return 0;
}
```

Ausgaben von Variablen in C++

```
#include <iostream>
using namespace std;
int main()
{
  int   an_int    = 123;
  float a_float   = 123.456;
  char  a_char    = 'A';
  char a_string[]  = "Ein String";
  bool  a_boolean = true;

  cout << "\n";
  cout <<    "Ein int:    " << an_int    <<
       "\n";
  cout <<    "Ein float:  " << a_float   <<
       "\n";
  cout <<    "Ein char:   " << a_char    <<
       "\n";
  cout <<    "Ein string: " << a_string  <<
       "\n";
  cout <<    "Ein bool:   " << a_boolean <<
       "\n";
  return 0;
}
```

- Programmübersetzung:
  - g++ -o cout3.exe cout3.cpp
- Programmausführung:
  - ./cout3.exe
- Programmausgabe:

  Ein int: 123

  Ein float: 123.456

  Ein char: A

  Ein string: Ein String

  Ein bool: 1

```
/* Einlesen von Daten in C */
include <stdio.h>
int main()
{
int x,y;
printf("Bitte geben Sie 2 Zahlen ein \n“);
scanf("%d %d",&x,&y);
return 0;
}
```

```
/* Einlesen von Daten in C++ */
#include <iostream>
using namespace std;
int main()
{
int x,y;
cout << "Bitte geben Sie 2 Zahlen ein \n";
cin >> x >> y;
cout << "Sie haben die Zahlen " << x <<" und " << y <<" eingegeben \n";
return 0;
}
```

```
/* Einlesen von Daten in C++ */
#include <iostream>
using namespace std;
int main()
{
int x,y;
cout << "Bitte geben Sie 2 Zahlen ein \n");
cin >> x >> y;
cout << "Sie haben die Zahlen " << x <<"
     und " << y <<" eingegeben \n";
return 0;
}
```

- Program,mübersetzung:
  - g++ -o cin.exe cout2.cpp
- Programmausführung:
  - ./cin.exe
- Programmausgabe:

  Bitte geben Sie 2 Zahlen ein

  3 5

  Sie haben die Zahlen 3 und 5 eingegeben

- (objektorientierte) Streambibliothek
  #include <iostream>
  using namespace std;

- Ausgabe von Informationen
  - Textausgabe
    cout << "Hallo \n";
  - Variablenausgabe
    cout << " X hat den Wert " << x <<"\n";

- Einlesen von Informationen:
  cin >> xwert >> ywert;

- C++ kennt auch alle Datentypen von C
- 2 zusätzliche Datentypen
  - Datentyp bool
    - boolsche Zahl
    - belegt nur 1 Byte
    - Wert ist entweder wahr oder falsch
  - Datentyp class
    - Daten können in einer Klasse verpackt werden
    - Klassen definieren Objekte in C++

Variablendeklaration in C

- Variablen müssen zu Beginn eines Block deklariert werden
- werden meistens am Anfang einer Funktion deklariert

- Beispiel

```
#include <stdio.h>
int count=0;
int main()
{
 for (count=0;count <10; count++)
    printf("count ist %d \n",count);
 return 0;
}
```

Variablendeklaration in C

- Variablen müssen zu Beginn eines Block deklariert werden
- werden meistens am Anfang einer Funktion deklariert

- Beispiel

```
#include <stdio.h>
int main()
{
 for (int count=0;count <10; count++)
    printf("count ist %d \n",count);
 return 0;
}
```

Variablendeklaration in C

- Variablen müssen zu Beginn eines Block deklariert werden
- werden meistens am Anfang einer Funktion deklariert

- Beispiel

```
#include <stdio.h>
int main()
{
 for (int count=0;count <10; count++)
    printf("count ist %d \n",count);
 return 0;
}
```

→ in C nicht möglich

- Variablen können zu jeder Zeit deklariert werden
- müssen weiterhin vor der ersten Verwendung deklariert werden

- Beispiel

```
#include <iostream.h>
int main()
{
 for (int count=0;count <10; count++)
    cout << "count ist " << count << "\n" ;
 return 0;
}
```

Variablendeklaration in C++

- Variablen können zu jeder Zeit deklariert werden
- müssen weiterhin vor der ersten Verwendung deklariert werden
- Möglichkeit sollte nur sparsam verwendet werden
- Code wird schnell sehr unübersichtlich
- geeignet für Schleifenvariablen

Strings in C++

- C++ kennt keinen Datentyp für Strings
- es gibt eine Klasse string, in deren Variablen man Strings abspeichern kann
- zusätzliche Header-Datei erforderlich

```
#include <string>
using namespace std:
```

- String deklarieren:

```
string vorname;
```

- String initialisieren:

```
vorname="Peter";
```

Strings in C++: Beispiel

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string vorname="Peter";
    string nachname="Müller";
    cout << "Name: " << vorname << nachname ;
    return 0;

}
```

Strings aneinanderhängen

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string vorname="Peter";
    string nachname="Lustig";

    string ausgabe, ausgabe2;
    ausgabe=vorname+nachname;
    ausgabe2=vorname+ " "+ nachname;
    cout << ausgabe << "\n";
    cout << ausgabe2 << "\n";

    return 0;
}
```

```
#include <iostream>
#include <string>
using namespace std:
int main()
{
string vorname="Peter";
string nachname="Lustig"
string ausgabe,ausgabe2;;
ausgabe=vorname+nachname;
ausgabe2=vorname+ " "+nachname;
cout << ausgabe << "\n";
cout << ausgabe2<< "\n";
return 0;
}
```

Programmausgabe:

PeterLustig

Peter Lustig

Strings ersetzen

- Methode replace ersetzt Teile in einem String

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
string ausgabe="Hallo Hans,  wie geht es Dir?";
string name="Helga";
cout<< ausgabe << "\n";
ausgabe.replace(6,4,name);
cout <<ausgabe <<"\n";

return 0;
}
```

Strings ersetzen

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
string ausgabe="Hallo Hans,  wie geht es
     Dir?";
string name="Helga";
cout<< ausgabe << "\n";
ausgabe.replace(6,4,name);
cout <<ausgabe <<"\n";
return 0;
}
```

- Programmausgabe:
  Hallo Hans, wie geht es Dir?
  Hallo Helga, wie geht es Dir ?

Strings einfügen

- Methode insert fügt Teile in einen String ein

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
string ausgabe="Hallo Klaus,  wie geht es ?";
string name=" und Helga ";
cout<< ausgabe << "\n";
ausgabe.insert(11, name);
cout <<ausgabe <<"\n";

return 0;
}
```

Strings einfügen

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
string ausgabe="Hallo Klaus,  wie geht
     es?";
string name=" und Helga ";
cout<< ausgabe << "\n";
ausgabe.insert(10, name);
cout <<ausgabe <<"\n";
return 0;
}
```

- Programmausgabe:
  Hallo Klaus, wie geht es?
  Hallo Klaus und Helga, wie geht es?

Strings löschen

- Methode erase löscht Teile in einen String

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
string ausgabe="Hallo Klaus,  wie geht es Dir?";
cout<< ausgabe << "\n";
ausgabe.erase(5, 6);
cout <<ausgabe <<"\n";

return 0;
}
```

Strings löschen

```
#include <iostream>
#include<string>
using namespace std;
int main()
{
string ausgabe="Hallo Klaus,  wie geht es
      Dir?";
cout<< ausgabe << "\n";
ausgabe.erase(5, 6);
cout <<ausgabe <<"\n";

return 0;
}
```

- Programmausgabe:
  Hallo Klaus, wie geht es Dir?
  Hallo, wie geht es Dir?

C++: Zeilenweise einlesen

- shift-Operator >> liest nur bis zum nächsten Whitespace-Zeichen ein

- getline()
  - Funktion zum Einlesen einer ganzen Zeile
  - allgemeine Struktur:
    - getline (cin,stringname);
    - Bsp.: getline (cin, name);

- Funktionen überladen
- Standardwerte für Funktionsparameter
- Inline-Funktionen

Funktionen überladen

- einfachste Form des Polymorphismus
- ein Funktionsname kann mehrmals definiert werden
- Programme werden anpassungsfähiger
- Benutzer kann Funktion auf unterschiedliche Arten aufrufen
- Programm trifft die richtige Zuordnung anhand der Signatur der Funktion
- Signatur:
  - Kombination aus Funktionsnamen mit Reihenfolge und Typen der Parameter
  - Rückgabewert gehört nicht zur Signatur
- Funktionsaufrufe müssen sich in einem der beiden Punkte unterscheiden
  - Anzahl der Parameter
  - Datentypen der Parameter

- *in C: Funktion wird nur anhand Ihres Namens identifiziert*

Funktionen überladen

- Beispiel: Quadratfläche berechnen
  - angegeben wird der x- und y-Wert von einem Punkt und die Seitenlänge
  - angegeben werden die x- und y-Werte von zwei Punkten

- mögliche Funktionsheader:
  - int quadrat (int xwert, int ywert, int laenge);
  - int quadrat(int xwert, int ywert, int xwert2, int ywert2);

- mögliche Funktionsaufrufe
  - fläche=quadrat(xwert, ywert,laenge);
  - fläche=quadrat(xwert1,ywert1, xwert2, ywert2)

Funktion überladen (Parameteranzahl)

```
#include <iostream>
using namespace std;
int produkt (int xwert);
int produkt (int xwert, int ywert);
int main()
{
   int xwert=3, ywert=5, ergebnis1=0,
     ergebnis2=0;
   ergebnis1=produkt(xwert);
   ergebnis2=produkt(xwert,ywert);
   cout << "produkt aus 1 Wert ist " << ergebnis1
     << "\n";
   cout << produkt aus 2 Werten ist " <<
     ergebnis2 << "\n";

     return 0;
}
```

```
int produkt (int xwert)
{
    int tmp1=0;
    tmp1=xwert*xwert;
     return (tmp1);
}
int produkt (int xwert, int ywert)
{
     int tmp2=0;
     tmp2=xwert*ywert;
     return (tmp2);
}
```

Funktion überladen (Parameteranzahl)

```
#include <iostream>
using namespace std;
int produkt (int xwert);
int produkt (int xwert, int ywert);
int main()
{
   int xwert=3, ywert=5, ergebnis1=0, ergebnis2=0;
   ergebnis1=produkt(xwert);
   ergebnis2=produkt(xwert,ywert);
   cout << "produkt aus 1 Wert ist " << ergebnis1 << "\n");
   cout << produkt aus 2 Werten ist " << ergebnis2 << "\n";

      return 0;
}
int produkt (int xwert)
      {….
      }
int produkt (int xwert, int ywert)
      {….
      }
```

- Programm übersetzen
  - g++ -o produkt.exe produkt.cpp
- Programm ausführen:
  - ./produkt.exe
- Programmausgabe:
  Produkt aus 1 Wert ist 9
  Produkt aus 2 Werten ist 15

Funktionen überladen (Datentypen)

```
#include <iostream>
using namespace std;
int division (int xwert, int ywert);
float division (float awert, float bwert);
int main ()
{
int div1=0,xwert=5, ywert=3;
float div2=0,awert=10.0,bwert=4.0;
div1=division(xwert,ywert);
div2=division(awert,bwert);
cout << "Division von " << xwert <<" und "
     <<ywert << "ist " <<  div1 <<"\n";
cout << "Division von " << awert << " und
     " << bwert << "ist" << div2 << "\n";
return 0;
}
```

```
int division (int xwert, int ywert)
{
     int tmp1=0;
     tmp1=xwert/ywert;
     return(tmp1);
}
float division (float awert, float bwert)
{
     float tmp2=0.0;
     tmp2=awert/bwert;
     return (tmp2);
}
```

Funktionen überladen (Datentypen)

```
#include <iostream>
using namespace std;
int division (int xwert, int ywert);
float division (float awert, float bwert);
int main ()
{
int div1=0,xwert=5, ywert=3;
float div2=0,awert=10.0,bwert=4.0;
div1=division(xwert,ywert);
div2=division(awert,bwert);
cout << "Division von " << xwert <<" und " <<ywert << "ist " <<
        div1 <<"\n";
cout << "Division von " << awert << " und " << bwert << "ist" <<
        div2 << "\n";
return 0;
}
int division (int xwert, int ywert)
{….
}
float division (float awert, float bwert)
{…}
```

- Programm übersetzen:
  - g++ -o div.exe divison.cpp
- Programm ausführen:
  - ./div.exe
- Programmausgabe:
  Division von 5 und 3 ist 1
  Division von 10.0 und 4.0 ist 2.5

Standardwerte als Funktionsparameter

- in der Parameterliste wird ein Standardübergabewert angegeben
- sinnvoll, wenn ein Parameter oft den gleichen Wert annimmt
- Funktion passt sich an, wenn ein Parameter im Aufruf nicht enthalten ist
- default-Wert wird nur im Funktionsprototypen angegeben

Standardwerte für Parameter (Beispiel 1)

```
/* Preisangabe in Euro oder DM */
#include <iostream>
using namespace std;
void Preisangabe (float preis, char waehrung[]="Euro");
int main ()
{
Preisangabe(9.99);
Preisangabe(19.56,"DM");
return 0;
}
void Preisangabe (float preis, char waehrung[])
{
    cout << "Der Preis betraegt " << preis <<  waehrung << "\n";
    return ;
}
```

Funktionen überladen (Datentypen)

```
/* Preisangabe in Euro oder DM */
#include <iostream>
using namespace std;
void Preisangabe (float preis, char
      waehrung[]="Euro");
int main ()
{
Preisangabe(9.99);
Preisangabe(19.56,"DM");
return 0;
}
void Preisangabe (float preis, char
      waehrung[])
{
      cout << "Der Preis betraegt " << preis <<
      waehrung << "\n";
      return ;
}
```

- Programm übersetzen:
  - g++ -o preis.exe preis.cpp
- Programm ausführen:
  - ./preis.exe
- Programmausgabe:

  Der Preis beträgt 9.99Euro
  Der Preis beträgt 19.56DM

Standardwerte als Parameter (Beispiel 2)

```
#include <iostream>
using namespace std;
void rechteck (int breite=3,int
     laenge=3,char zeichen = 'X');
int main()
{
  cout << "\nRechteck( 8, 2, \'A\' );\n";
  rechteck( 8, 2, '*' );
  cout << "\nRechteck( 4, 5 );\n";
  rechteck( 4, 5 );
  cout << "\nRechteck( 2 );\n";
  rechteck( 2 );
  cout << "\nRechteck( );\n";
  rechteck( );
  return 0;
}
```

```
void rechteck ( int breite, int laenge, char
     zeichen )
{
  int ctr1 = 0;
  int ctr2 = 0;
  for (ctr1 = 0; ctr1 < laenge; ctr1++ )
  {
    cout << "\n";
    for ( ctr2 = 0; ctr2 < breite; ctr2++)
    {
      cout << zeichen;
    }
  }
   cout << "\n";
}
```

Standardwerte als Funktionen: Beispiel 2

- Rechteck (8,2,'A');
  AAAAAAAA
  AAAAAAAA

- Rechteck (4,5);
  xxxx
  xxxx
  xxxx
  xxxx
  xxxx

- Rechteck (2);
  xx
  xx
  xx

- Rechteck ();
  xxx
  xxx
  xxx

Inline-Funktionen

- Funktionsaufrufe kosten Zeit
- Funktion wird als inline deklariert
  - Compiler ersetzt Funktionsaufruf durch Code der Funktion
  - Ausführungszeit wird geringer
  - Programmcode wird länger
- Beispiele
  - inline float quadrat (float xwert)
  - inline float haelfte (float ywert)
- nur sinnvoll, wenn Ausführungszeit klein ist im Vergleich zum Verwaltungsaufwand
- nur eine Empfehlung an den Compiler

Inline-Funktion (Beispiel)

```
#include <iostream>
using namespace std;
float quadrat (float zahl);
float haelfte (float zahl);
int main()
{
float zahl=0;
cout << "Eine Zahl eingeben \n";
cin >> zahl;
cout <<"Quadriert: " << quadrat(zahl) << "\n";
cout << "Haelfte: " << haelfte(zahl) << "\n";
return 0;
}
inline float quadrat (float zahl)
     {return(zahl*zahl);}
inline float haelfte (float zahl)
     {return (zahl/2);}
```

Inline-Funktion (Beispiel)

```
#include <iostream.>
using namespace std;
float quadrat (float zahl);
float haelfte (float zahl);
int main()
{
float zahl=0;
cout << "Eine Zahl eingeben \n";
cin >> zahl;
cout <<"Quadriert: " << quadrat(zahl);
cout << "Haelfte: " << haelfte(zahl);
return 0;
}
inline float quadrat (float zahl)
      return(zahl*zahl);
inline float haelfte (float zahl)
      return (zahl/2);
```

```
#include <iostream>
using namespace std;
int main()
{
float zahl=0;
cout << "Eine Zahl eingeben \n";
cin >> zahl;
cout <<"Quadriert: " << (zahl*zahl);
cout << "Haelfte: " << (zahl/2);
return 0;
}
```

- Strukturen
- Memberfunktionen
- Klassen
- Vererbung

Komplexe Datentypen: Strukturen

- ist ähnlich einem Feld
- Zusammenfassung mehrerer Variablen zu einem neuen Namen
- Variablen können unterschiedliche Datentypen haben
- Variablen zu einem Thema werden zusammengefasst
- Variablen einer Struktur werden als Elemente bezeichnet
- jedes Element kann jederzeit angesprochen werden
- können wie normale Variable behandelt werden
- sinnvoll, wenn Informationen der Gruppe bearbeitet werden sollen
- Beispiele:
  - Personendaten (Name, Vorname, Straße, Ort)
  - Koordinaten eines Punktes (x-, y- und z-Koordinaten)
  - Datum (Tag, Monat, Jahr)

Deklaration einer Struktur

- allgemein:

```
struct structname {
        Datentyp strukvar1;
        Datentyp strukvar2;
        ...
        Datentyp strukvarn;
};
```

- Beispiel:

```
struct zeit {
        int stunden;
        int minuten;
        int sekunden;
};
```

Strukturvariable definieren

- Bereitstellung von Speicherplatz
- bei der Deklaration

```
struct koord {
        int x;
        int y;
}; erste,zweite;
```

- separat

```
struct koord erste,zweite;
```

Zugriff auf Strukturelement

- Strukturelemente lassen sich wie normale Variablen verwenden
- Zugriff über Strukturelement-Operator (.)
- wird auch als Punktoperator bezeichnet
- Punktoperator verbindet Strukturvariable mit einer Komponente
- Element ansprechen mit:

  varname.strukvar;

```
#include <iostream>
using namespace std;
struct zeit {
	int stunden;
	int minuten;
};
int main()
{
struct zeit start={10,20};
struct zeit ende={18,34};
cout << "Startzeit: " << start.stunden << "\n";
cout << "Endzeit: " << ende.minuten  << "\n";
return 0;
}
```

Strukturen verwenden

```
#include <iostream>
using namespace std;
struct zeit {
        int stunden;
        int minuten;
};
int main()
{
struct zeit start={10,20};
struct zeit ende={18,34};
cout << "Startzeit:" << start.stunden <<
     "\n";
cout << "Endzeit: " << ende.minuten  <<
     "\n";
return 0;
}
```

- Programm übersetzen:
  - g++ -o struc1.exe struc1.cpp
- Programm ausführen:
  - ./struc1.exe
- Programmausgabe:

  Startzeit: 10

  Endzeit: 34

```
#include <iostream>
using namespace std;
struct zeit {
        int stunden;
        int minuten;
};
int main()
{
struct zeit start={10,20};
struct zeit neuezeit={00,00};
cout << "alte  Zeit vorher " << start.stunden << "\n";
cout << "neue Zeit vorher " << neuezeit.stunden << "\n";
neuezeit.stunden=start.stunden+5;
cout << "alte  Zeit" << start.stunden << "\n";
cout << "neue Zeit" << neuezeit.stunden << "\n";
return 0;
}
```

```
#include <iostream>
using namspace std;
struct zeit {
        int stunden;
        int minuten;
};
int main()
{
struct zeit start={10,20};
struct zeit neuezeit={00,00};
cout << "alte  Zeit vorher " << start.stunden << "\n";
cout << "neue Zeit vorher " << neuezeit.stunden <<
      "\n";
neuezeit.stunden=start.stunden+5;
cout << "alte  Zeit" << start.stunden << "\n";
cout << "neue Zeit" << neuezeit.stunden << "\n";
return 0;
}
```

- Programmübersetzung:
  - g++ -o struc2.exe struc2.cpp
- Programmausführung:
  - ./struc2.exe
- Programmausgabe:

```
alte Zeit vorher: 10
neue Zeit vorher: 0
alte Zeit: 10
neue Zeit: 15
```

Strukturen an Funktionen übergeben

```
#include <iostream>
using namespace std;
struct daten {
      float betrag;
      char vorname[30];
      char nachname[30];
} rec;
void ausgabe (struct daten x);
int main ()
{
cout << "Bitte Vor und Nachnamen eingeben \n";
cin >> rec.vorname >> rec.nachname;
cout << "Bitte die Spendenhöhe eingeben \n";
cin >> rec.betrag;

ausgabe (rec);

return 0;
}
```

```
void ausgabe (struct daten x)
{
cout << "Spender " << x.vorname <<
     x.nachname << "gab " << x.betrag <<
     "Euro \n";
}
```

Strukturen an Funktionen übergeben

```
#include <iostream>
using namespace std;
struct daten {
      float betrag;
      char vorname[30];
      char nachname[30];
} rec;
void ausgabe (struct daten x);
int main ()
{
cout << "Bitte Vor und Nachnamen eingeben \n";
cin >> rec.vorname >> rec.nachname;
cout << "Bitte die Spendenhöhe eingeben \n";
cin >> rec.betrag;

ausgabe (rec);

return 0;
}
```

```
void ausgabe (struct daten x)
{
cout << "Spender " << x.vorname <<
     x.nachname << "gab " << x.betrag <<
     "Euro \n";
}
```

Programmausgabe:
Bitte Vor- und Nachnamen eingeben
Hans Meier
Bitte die Spendenhöhe eingeben
333
Spender Hans Meier gab 333 Euro

- Strukturen in C:
  - es können nur Variablen in einer Struktur definiert werden
- Strukturen in C++:
  - Es können Daten und Funktionen definiert werden
  - → Memberfunktionen

Member-Funktion: Definition in Struktur

```
#include<iostream>
using namespace std;
struct zeit {
        int stunden;
        int minuten;
        void zeitausgeben(void)
        {
        cout << stunden <<“.”<< minuten<<“\n”;
        }
};
int main ()
{
struct zeit start={10,20};
struct zeit ende={12,20};
cout << "Startzeit ";
start.zeitausgeben();
cout << "Endzeit ";
ende.zeitausgeben();
return 0;
}
```

Programmausgabe:

Startzeit: 10:20

Endzeit: 12:20

Member-Fkt.: Definition außerhalb der Struktur

```
#include<iostream>
using namespace std;
struct zeit {
        int stunden;
        int minuten;
        void zeitausgeben(void);
};
int main ()
{
struct zeit start={10,20};
struct zeit ende={12,20};

cout << "Startzeit ";
start.zeitausgeben();
cout << "Endzeit ";
ende.zeitausgeben();

return 0;
}
```

```
void zeit::zeitausgeben(void)
{
cout << stunden <<":" << minuten<< "\n";
}
```

Programmausgabe:

Startzeit: 10:20
Endzeit: 12:20

Member-Funktion: Beispiel 2

```
#include<iostream>
using namespace std;
struct zeit {
        int stunden;
        int minuten;
        void zeitausgeben(void);
        void addstunde(void);
};
int main ()
{
struct zeit start={10,20};
struct zeit ende={12,20};
struct zeit neuezeit={00,00};
neuezeit=start;
neuezeit.addstunde();
cout << "Startzeit ";
start.zeitausgeben();
cout << "Endzeit ";
ende.zeitausgeben();
cout << "Neue Zeit ";
neuezeit.zeitausgeben();
return 0;
}
```

```
void zeit::zeitausgeben(void)
{
cout << stunden <<":" << minuten<< "\n";
}
void zeit::addstunde(void)
{
stunden=stunden+1;
}
```

Member-Funktion (Beispiel 2)

```
#include<iostream>
using namespace std;
struct zeit {
        int stunden;
        int minuten;
        void zeitausgeben(void);
        void addstunde(void);
};
int main ()
{
struct zeit start={10,20};
struct zeit ende={12,20};
struct zeit neuezeit={00,00};
neuezeit=start;
neuezeit.addstunde();
cout << "Startzeit ";
start.zeitausgeben();
cout << "Endzeit ";
ende.zeitausgeben();
cout << "Neue Zeit ";
neuezeit.zeitausgeben();

return 0;
}
```

```
void zeit::zeitausgeben(void)
{
cout << stunden <<":" << minuten<< "\n";
}
void zeit::addstunde(void)
{
stunden=stunden+1;
}
```

Programmausgabe:

Startzeit: 10:20
Endzeit: 12:20
Neue Zeit: 11:20

- Ideen der objektorientierten Konzepte:
  - Daten und Programmcode werden zusammengefasst
  - Operationen und Methoden stellen die Schnittstelle nach außen dar
    - Geheimhaltungsprinzip: Daten von außen nicht sichtbar
    - Konzept: Black Box
  - Eigenschaften und Methoden können durch Vererbung weiter gegeben werden

- abstrahiert Gegenstände der realen Welt
- in einer Klasse werden Daten, die ein Objekt beschreiben und Prozeduren, die ein Objekt verändern in einer Struktur zusammen gefasst
- in der OOP werden
  - Klassen gebildet, um Gruppen von Objekten zu beschreiben
  - Objekte erstellt, um einen Gegenstand der realen Welt abzubilden
  - Nachrichten zwischen Objekten verschickt
- andere Sichtweise der Programmierung:
  - keine top-down-Programmierung
  - Programm wird aus Modulen aufgebaut, die unabhängig voneinander getestet werden können

Objekte

- abstrahieren einen realen Gegenstand
- haben bestimmte Eigenschaften und Methoden
- berichten über ihren Zustand und können diesen über Methoden verändern
- kommunizieren mit anderen Objekten über Schnittstellen
- gekapselte Datenstruktur
- enthält sowohl Datenkomponenten als auch zu ihrer Bearbeitung dienende Methoden

- zentrale Datenstrukturen in C++
- sind eine Erweiterung des komplexen Datentyps struct
- Sammlung von Variablen kombiniert mit einer Gruppe von Funktionen
- fassen Objekte mit den gleichen Eigenschaften und Methoden zusammen
- kapseln zusammengehörige Daten und Funktionen vom Rest des Programms ab
- Elementvariablen: Variablen einer Klasse
- Elementfunktionen o. Methoden: Funktionen in einer Klasse

- Beispiel:
  - Klasse Verkehrsteilnehmer: Auto, Fahrrad, Fußgänger
  - Klasse Verkehrszeichen: rote Ampel, grüne Ampel, Stop-Schild
  - Klasse KFZ: Audi, Golf, Seat
  - Klasse Auto: Reifen, Motor, Autoradio, sitzen

- Bestandteile einer Klasse:
  - Datenelemente
    - Attribute
  - Elementfunktionen
    - Methodem
    - dienen zum Nachrichtenaustausch
  - Zugriffsbeschränkungen für die Elemente
    - Festlegung, wer auf die Elemente dieser Klasse zugreifen darf

Klassen (Definition)

- erfolgt mit dem Schlüsselwort class
- Klasse Deklaration:

```
class zeit {
        int stunden;
        int minuten;
};
```

- Objekte deklarieren:

```
zeit startzeit;
zeit endzeit;
```

- Objekt ist eine eigenstämdige Instanz einer Klasse
- mit Objekten kann man arbeiten
- Zugriff auf die einzelnen Element erfolgt wie bei den Strukturen (Punktoperator)

```
#include<iostream>
using namespace std;
class zeit {
        int stunden;
        int minuten;
        int sekunden;
        void zeitausgeben(void);
        void addstunde(void);
};
int main ()
{
zeit startzeit={10,20,55};
zeit endzeit={12,20,55};
 zeit neuezeit={00,00,00};
neuezeit=startzeit;
neuezeit.addstunde();
cout << "Startzeit ";
startzeit.zeitausgeben();
cout << "Endzeit ";
endzeit.zeitausgeben();
cout << "Neue Zeit ";
neuezeit.zeitausgeben();
return 0;
}
```

```
void zeit::zeitausgeben(void)
{
cout << stunden <<":" << minuten<<":"
     <<sekunden << "\n";
}
void zeit::addstunde(void)
{
stunden=stunden+1;
}
```

--> Fehler bei der Übersetzung !!

Klassen: Zugriffstypen

- C++ kennt 3 zusätzliche Schlüsselwörter für die Zugriffserlaubnis:
  - public (öffentlich)
  - private (privat)
  - protected (geschützt)
- per default sind alle Mitglieder einer Klasse privat (Datenelemente und Memberfunktionen)
- in Strukturen ist der Zugriff per default öffentlich

Klassen: private Daten

- auf private Elemente kann man nur über die Member-Funktion zugreifen
- kein Zugriff von außen möglich, d.h. Elemente sind versteckt
- public: auf diese Daten und Funktionen kann auch von außen zugegriffen werden

- Beispiel:

```
class koord{
        int xwert;
        int ywert;
  };
  int main()
  {
  koord punkt
  punkt.xwert=1;
   return 0;
  }
```

- auf private Elemente kann man nur über die Member-Funktion zugreifen

- Beispiel:

```
class koord{
        int xwert;
        int ywert;
 };
 int main()
 {
 koord punkt
 punkt.xwert=1;   --> Fehler, da Daten privat sind
 return 0;
 }
```

- auf private Elemente kann man nur über die Member-Funktion zugreifen
- Beispiel:

```
class koord{
        public:
        int xwert;
        int ywert;
 };
 int main()
 {
 koord punkt
 punkt.xwert=1;
 return 0;
 }
```

```
#include<iostream>
using namespace std;
class zeit {
    public:
        int stunden;
        int minuten;
        void zeitausgeben(void);
        void addstunde(void);
};
int main ()
{
zeit startzeit={10,20};
zeit endzeit={12,20};
 zeit neuezeit={00,00};
neuezeit=startzeit;
neuezeit.addstunde();
cout << "Startzeit ";
startzeit.zeitausgeben();
cout << "Endzeit ";
endzeit.zeitausgeben();
cout << "Neue Zeit ";
neuezeit.zeitausgeben();
return 0;}
```

```
void zeit::zeitausgeben(void)
{
cout << stunden <<":" << minuten << "\n";
}
void zeit::addstunde(void)
{
stunden=stunden+1;
}
```

Programmausgabe:

Startzeit: 10:20

Endzeit: 00:00

Neue Zeit: 11:20

Zugriffsfunktion

- steuern den Zugriff auf eine Klasse
- erlauben die Verkapselung der Funktionalität und der Daten
- Code wird dadurch leichter wartbar
- Zugriffsschutz ist ein wichtiger Teil des objektorientierten Programmierens

Zugriffsbeschränkungen

- public:
  - nachfolgende Elemente u. Methoden unterliegen keiner Zugriffsbeschränkung
  - können beliebig verändert und aufgerufen werden
- private:
  - nachfolgende Elemente u. Methoden nur innerhalb der Klasse zugreifbar
  - keine Veränderung von außen möglich

Vererbung

- stellt eine Beziehung zwischen Klassen dar
- jede Klasse erbt von Ihrer Basisklasse Eigenschaften und Methoden
- geerbte Eigenschaften und Methoden können überlagert und ergänzt werden

Basis- und Subklassen

- Basisklassen:
  - vererben Eigenschaften und Methoden an Subklassen
  - können selbst Subklassen von anderen Klassen sein

- Subklassen:
  - verfügen über alle Eigenschaften der Basisklasse
  - können geerbte Eigenschaften verändern
  - können zusätzliche Eigenschaften und Methoden haben

- Basisklasse: Fahrzeuge
  - Landfahrzeuge: Eisenbahn, Auto, Bus
  - Wasserfahrzeuge: Segelboot, Motorboot, Kanu
  - Luftfahrzeuge: Flugzeug, Zeppelin, Heissluftballon

Basisklasse definieren

```
#include <iostream>
using namespace std;
class Vogel
{public:
 void fliege();
};
int main()
{
cout << “Basisklasse: \n“ ;
Vogel vogel1;
vogel1.fliege();
return 0;
}
void Vogel::fliege()
{
cout << "Ich kann fliegen \n";
}
```

Basssklasse definieren

```
#include <iostream>
using namespace std;
class Vogel
{public:
 void fliege();
};
int main()
{
cout << “Basisklasse: \n“ ;
Vogel vogel1;
vogel1.fliege();
return 0;
}
void Vogel::fliege()
{
cout << "Ich kann fliegen \n";
}
```

- Programmausgabe

  Basisklasse:
  Ich kann fliegen

Subklasse definieren

```
#include <iostream>
using namespace std;
class Vogel
{public:
  void fliege();
};
class Singvogel:public Vogel
{
public:
void sing();
};
int main()
{
Vogel vogel1;
Singvogel saenger;
cout << "Basisklasse: \n";
vogel1.fliege();
cout << "Subklasse: \n";
saenger.fliege();
saenger.sing();
return 0;

void Vogel::fliege()
{
cout << "Ich kann fliegen \n";
}
void Singvogel::sing()
{
cout << "Ich kann singen \n";
}
```

- Programmausgabe:

```
Basisklasse:
Ich kann fliegen
Subklasse:
Ich kann fliegen
Ich kann singen
```

Zugriffsrechte Basisklasse

- public:
  - public-Elemente sind in der Basisklasse und der Subklasse öffentlich
- private:
  - Basisklasse und Subklasse haben eigene Elemente
  - Subklasse hat keinen Zugriff auf private Elemente der Basisklasse
- protected:
  - Elemente der Basisklasse sind vor einem Zugriff von außen geschützt
  - Subklassen können auf protected Elemente der Basisklasse zugreifen

Übungsblatt