Compilerbau CS1019

Th. Letschert

TH Mittelhessen Gießen

University of Applied Sciences

C / C++

- Installation
- Kurzeinführung

Unterlagen zu C

Literaturhinweise

C wird in der Veranstaltung KSP (siehe https://homepages.thm.de/~hg53/ksp-ws1617/index.html) gelehrt. Wir verweisen auf die dort empfohlene Literatur

Ansonsten der ewige C-Klassiker:

- **Kernighan, B.W., Ritchie, D.M.: Programmieren in C, 2.Auflage, ANSI C, 1990.**

Online-Quellen

https://de.wikibooks.org/wiki/C-Programmierung ***Ein Beispiel von tausenden im Netz***

… und natürlich nicht zu vergessen ;) :

- **https://homepages.thm.de/~hg51/Veranstaltungen/Programmierung1/index.html**
- **https://homepages.thm.de/~hg51/Veranstaltungen/Programmierung2/index.html**

Installation von C

GCC

Als C-Installation empfehlen wir den GCC. Siehe **https://gcc.gnu.org**.

Installation – generell

Für die Installation von C empfehlen wir generell die Verwendung des Paket-Managers Ihres Systems. Auf Linux-Systemen ist gcc in der Regel installiert.

Installation – Mac

Auf Macs empfehlen wir ***Xcode*** mit den ***Command line Tools***

Sie können C-Programme natürlich auch mit Eclipse entwickeln ohne selbst direkt in Kontakt mit der Kommandozeile zu kommen. Installieren Sie dazu die Eclipse IDE for C/C++-Developers.

Es wird aber dringend empfohlen sich ein wenig mit einer Linux-/Unix-artigen Entwicklungs-Umgebung vertraut zu machen!

Dies ist heute ohne großen Aufwand auf allen relevanten Plattformen recht einfach möglich.

Installation von C auf einem Mac

Xcode : Entwicklungsplattform auf dem Mac

Xcode ist üblicherweise unter den Anwendungen/ Programmen in `/Applications/Xcode.app/` zu finden.

`xcode-select -p` *zeigt den Installationspfad an*

Sollte Xcode nicht installiert sein, dann bitte über den App-Store installieren.

Command Line Tools

gcc gehört zu den Command Line Tools

Mit dem kommando

`gcc –version`

finden Sie heraus ob gcc installiert ist.

Die *command line tools* machen aus Ihrem Mac ein vollwertiges Unix-System.

Es ist darum nicht notwendig neben OS-X ein weiters Unix-System – z.B. Linux – zu installieren siehe: https://developer.apple.com/opensource/

Installation von C auf einem Mac

Beispiel *command line tools* im Einsatz:

```
$ cat -> hello.c
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello, world!\n");
  return 0;
}^d
$ gcc -g -Wall -std=c89 -pedantic -o hello hello.c
$ ./hello
Hello, world!
$
```

^d: *Eingabeende mit crtl-d anzeigen!*

Installation von C auf Windows

Installieren Sie Windows 10

Aktivieren Sie das „Windows Subsystem for Linux (Beta)“

- **entspricht in etwa den *comand line tools* auf einem Mac**
- **Details bitte googeln**

Installation von C

Jetzt haben Sie C als Teil einer Unix-artigen Entwicklungsumbebung installiert.

Machen Sie sich (bedarfsgetrieben) mit den allerwichtigsten Kommandos vertraut:

- **cd, ls, cat**
- **vi**
- **mkdir, chmod, chown**
- **...**

Google hilft

Einfache Programmelemente

Einfache Programmelemente sind in C/C++ sehr ähnlich zu Java
(Java wurde als Vereinfachung von C++ konzipiert)

– Kontrollstrukturen

Variablendefinitionen, Anweisungen und Schleifen sind praktisch identisch

```java
package p;

public class EinfachesProgramm {

  public static void main(String[] args) {
    int[] a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int sum = 0;

    // addiere alle geraden Zahlen in a
    for (int i=0; i< 10; i++) {
      if (a[i] % 2 == 0) {
        sum = sum + a[i];
      }
    }

    System.out.println(sum);
  }

}
```

Java

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

int main() {
  int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int sum = 0;

  // addiere alle geraden Zahlen in a
  for (int i=0; i< 10; i++) {
    if (a[i] % 2 == 0) {
      sum = sum + a[i];
    }
  }

  cout << sum;

  return 0;
}
```

C++

Einfache Programmelemente

Einfache Programmelemente sind in C/C++ sehr ähnlich zu Java

– Funktionen (statische Methoden)

Definition und Verwendung von Funktionen ist praktisch identisch

```java
package p;

public class EinfachesProgramm {

  private static int addEven(int x, int y) {
    return  y%2 == 0 ? x+y : x;
  }

  public static void main(String[] args) {
    int[] a = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    int sum = 0;

    // addiere alle geraden Zahlen in a
    for (int i=0; i< 10; i++) {
      sum = addEven(sum, a[i]);
    }

    System.out.println(sum);
  }

}
```

Java

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


int addEven (int x, int y) {
  return y%2 == 0 ? x+y : x;
}

int main() {
  int a[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int sum = 0;

  // addiere alle geraden Zahlen in a
  for (int i=0; i< 10; i++) {
    sum = addEven(sum, a[i]);
  }

  cout << sum;

  return 0;
}
```

C++

Einfache Programmelemente

Funktionen : Parameterübergabe

Der Prozess der Parameterübergabe kann in C/C++ fein gesteuert werden

- In Java werden Parameter stets in der gleichen Art übergeben
- In C/C++ ist die Art der Parameterübergabe beeinflussbar

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

void f1(int x) {
  x = x+1;
}

void f2(int & x) {
  x = x+1;
}


int main() {
  int a = 1;
  int b = 1;
  f1(a);
  f2(b);
  cout << "a = " << a << e
  cout << "b = " << b << e
  return 0;
}
```

Wertübergabe
x wird mit dem Wert des Parameters initialisiert

&: Adress- (Referenz-) Übergabe
x wird mit der Adresse des Parameters initialisiert

f1 verändert sein Argument nicht / arbeitet auf Kopie
f2 verändert sein Argument / arbeitet auf dem Original

```
a = 1
b = 2
```

```
void f3(const int & x) {
  VERBOTEN: x = x+1;
}
```

const: *Adressübergabe mit dem Versprechen nichts zu ändern.*

Einfache Programmelemente

Einfache Programmelemente sind in C/C++ sehr ähnlich zu Java

– Datentypen

Elementare Datentypen sind sehr ähnlich, allerdings erlaubt es C/C++ oft das Wissen auszunutzen, dass alle Werte letztlich nur Bits (int-Werte) sind.

Das erhöht die Fehleranfälligkeit aber auch die Flexibilität – die allerdings so meist nur bei Hardware-nahen Programmen benötigt wird.

Zeichen + int-Wert

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

int main() {
  char  c = 'a';
  cout << c << endl;
  c = c+1;
  cout << c;
  return 0;
}
```

```
a
b
```

Enum + int-Wert

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

int main() {
  enum  Tag {Mo, Di, Mi, Do, Fr, Sa, So};
  Tag t = Do;
  cout << "Sa hat den Code " << t+2 << endl;
  return 0;
}
```

```
Sa hat den Code 5
```

Vermischung von bool und int

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

int main() {
  int i = 0;
  if (i) { cout << "i ist Null" << endl; }
  return 0;
}
```

Zahl mit und ohne Vorzeichen interpretieren.

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

int main() {
  int x = -1;
  unsigned int y = x;
  cout << x << endl;
  cout << y << endl;
  return 0;
}
```

```
-1
4294967295
```

Bibliotheken

C++ kommt mit einer mächtigen Bibliothek – *Standard Template Library* STL

Die Funktionalität ist vergleichbar aber nicht nicht deckungsgleich

Ist nicht so standardisiert, dass sie auf jeder Plattform völlig gleich ist

```cpp
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

int main() {
  string name, vorname, nachname;

  cout << "Vorname: ";
  cin >> vorname;

  cout << "Nachname: ";
  cin >> nachname;

  name = vorname + " " + nachname;
  cout << "Name = " << name << " hat " << name.length() << " Buchstaben" << endl;

  cout << "Wurzel der Länge ist " << sqrt(name.length()) << endl;
}
```

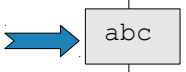
```
Vorname: Thomas
Nachname: Letschert
Name = Thomas Letschert hat 16 Buchstaben
Wurzel der Länge ist 4
```

Speicherverwaltung

Pointer: Speicheradressen können explizit verarbeitet werden

C/C++ erlaubt den expliziten Zugriff auf und
das Rechnen mit Adressen von Speicherplätzen („Zeiger“ / „Pointer“)

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

int main() {
  char x[] = {'a', 'b', 'c', 0 };

  char * p = &x[0];

  while (*p != 0) {
     cout << *p;
     p++;
  }
  cout << endl;

}
```



char * ist der Typ „Zeiger auf ein Zeichen“

&x[0] ist der Zeiger auf x[0]

***p** ist der Wert (im Speicherplatz) auf den (der Wert von) p zeigt.

p++ setzt den Zeiger p auf den nächsten Speicherplatz.

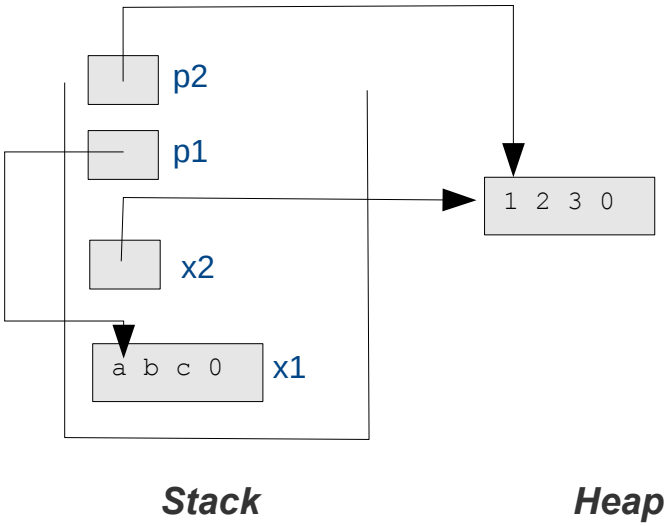
Speicherverwaltung

Der Speicherort kann explizit definiert werden

C/C++ erlaubt es explizit festzulegen, ob Daten im Stack oder im Heap abgelegt werden.

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

int main() {
  char   x1[] = {'a', 'b', 'c', 0 };
  char * x2 = new char[4]{1, 2, 3, 0 };

  char * p1 = &x1[0];
  char * p2 = x2;

  while (*p1 != 0) {
    *p2++ += *p1++;
  }

  p2 = x2;
  while (*p2 != 0) {
     cout << *p2;
     p2++;
  }
  cout << endl;

}
```

bdf



2 Arrays: einer im Stack einer im Heap

Programmorganisation

Quellcode in Dateien und der Prozess des Übersetzens

Die Struktur Quellcodes in Dateien und der Übersetzungsprozesse muss in C/C++ selbst organisiert werden. Die Modularisierung ist nicht (unbedingt) an Klassen gebunden wie in Java. Der Programmierer muss Abhängigkeiten im Code selbst auflösen.

*Compiler: lies diese Datei beim Übersetzen, damit du weist was für ein Ding **f** ist!*

```
#include <iostream>
#include "Modul_2.h"

using namespace std;

int main() {
  int x = 5;
  cout << f(x) << endl;
  return 0;
}
```

Datei: Modul_1.cpp

Modul_1
logische Einheit, eine Datei

```
#ifndef MODUL_2_H_
#define MODUL_2_H_


int f(int x);

#endif /* MODUL_2_H_ */
```

Datei: Modul_2.h

```
#include "Modul_2.h"

int f(int x) {
    return 2*x + 5;
}
```

Datei: Modul_2.cpp

Modul_2
logische Einheit, zwei Dateien

C / C++

```
public class Modul_1 {

  public static void main(
                String[] args) {
     int x = 5;
     System.out.println(Modul_2.f(x));
  }

}
```

Modul_1: Klasse und Datei

```
public class Modul_2 {

  public static int f(int x) {
    return 2*x + 5;
  }

}
```

Modul_2: Klasse und Datei

Java

Programmorganisation

Java hat recht strikte Vorgaben zur Programmorganisation:

- Modul („Zusammengehöriges“) ~ Klasse
- Klasse ~ Datei
- Paket ~ Verzeichnis
- Der Compiler sucht selbstständig alle notwendigen Information die er zum Übersetzen braucht in den Dateien. die er im Klassenpfad findet
- Zur Ausführungszeit sucht die JVM alle Klassen die zur Ausführung gebraucht werden auf dem Klassenpfad

In C/C++ ist die Programmorganisation frei und auf Basis von Dateien selbstbestimmt:

- Die Programmorganisation richtet sich stets (und tunlichst) nach Konventionen
  (StandardwerK: John Lakos *Large-Scale C++ Design*, Addison-Wesley 1996, mehre Hundert Seiten !!)
- Module ~ eine bis mehrere Dateien
- Logische Abhängigkeiten werden über Inklude-Direktiven (Compiler lies das auch noch) im Quellcode bekannt gegeben.
- Inklude-Direktiven beziehen sich auf Header-Dateien
- Header-Dateien enthalten die für die Übersetzung anderer Dateien notwendigen (reduzierten) Informationen über dieses Modul

Klassendefinition

Java und C++ haben *im Wesentlichen* das gleiche Konzept der Objektorientierung

- public, private, static haben gleiche Bedeutungen
- private ist *default*
- Klassen in C++ haben einen Deklarationsteil und eine Definitionsteil
  - Deklaration: Attribute, Methodenköpfe
  - Definitionsteil: Methodenimplementierung

  Sinn: *Der Definitionsteil muss via inklude-Direktive benutzenden Modulen/Dateien bekannt gegeben werden. Der Definitionsteil wird für die Übersetzung anderer Module/Dateien nicht gebraucht.*
- ToString wird nicht als Methode definiert.

C++ erlaubt die genaue Kontrolle des Speichers

Java: Alle Objekte sind im Heap

C++ : Objekte können im Heap oder in Stack gespeichert werden

Klassen können /Zeiger-/Heap-orientiert oder Stack-orientiert definiert werden

C/C++ vs Java : Klassendefinition

Klassendefinition

Java und C++ haben im Wesentlichen das gleiche OO-Konzept

Java

```
public class Vektor {
  private double x;
  private double y;

  public Vektor(double x, double y) {
    this.x = x;
    this.y = y;
  }

  public static Vektor NullVektor = new Vektor(0,0);

  public Vektor add(Vektor that) {
    return new Vektor(this.x+that.x, this.y+that.y);
  }

  public String toString() {
    return "[" + x + ", " + y + "]";
  }

  public static void main(String[] args) {
    Vektor v1 = new Vektor(1,2);
    Vektor v2 = v1.add(new Vektor(2,2));
    v2.add(Vektor.NullVektor);
    System.out.println(v2);
  }
}
```

→ `[3.0, 4.0]`

C++

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


class Vektor {
friend ostream& operator<<(ostream& s,
                           const Vektor& v);
  double x;
  double y;
public:
  Vektor(double x, double y);
  Vektor add(Vektor that);
  static Vektor NullVektor;
};


Vektor::Vektor(double xx, double yy) {
  x = xx;
  y = yy;
}
Vektor Vektor::add(Vektor that) {
  return Vektor(x+that.x, y+that.y);
}
Vektor NullVektor = Vektor(0, 0);

ostream& operator<<(ostream& s, const Vektor& v) {
  return s << "[" << v.x << ", " << v.y << "]";
}

int main() {
  Vektor v1(1, 2);
  Vektor v2 = v1.add(Vektor(2,2));
  v2 = v2.add(NullVektor);
  cout << v2 << endl;
  return 0;
}
```

→ `[3.0, 4.0]`

Klassendefinition

in ordentlicher und modularisierter Form

Vektor Modul / Klasse

Datei: Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

#include <iostream>
using namespace std;

class Vektor {
friend ostream& operator<<(ostream &, const Vektor &);
  double x;
  double y;
public:
  Vektor(double, double);
  Vektor add(const Vektor that) const;
  static Vektor NullVektor;
};

extern Vektor NullVektor;

#endif /* VEKTOR_H_ */
```

Test.cpp

```
#include <iostream>
#include "Vektor.h"
using namespace std;

int main() {
  Vektor v1(1, 2);
  Vektor v2 = v1.add(Vektor(2,2));
  v2 = v2.add(NullVektor);
  cout << v2 << endl;
  return 0;
}
```

Datei: Vektor.cpp

```
#include "Vektor.h"

Vektor::Vektor(double x, double y) : x(x), y(y) {}

Vektor Vektor::add(const Vektor that) const { return Vektor(x+that.x, y+that.y); }

Vektor NullVektor = Vektor(0, 0);

ostream& operator<<(ostream& s, const Vektor& v) {return s << "[" << v.x << ", " << v.y << "]";}
```

C/C++ vs Java : Klassendefinition

Klassendefinition

in ordentlicher und modularisierter Form

friend : Zugriff auf Privtes erlauben
„A friend is someone who may touch your private parts.“

Datei: Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

#include <iostream>
using namespace std;

class Vektor {
friend ostream& operator<<(ostream &, const Vektor &);
  double x;
  double y;
public:
  Vektor(double, double);
  Vektor add(const Vektor & that) const;
  static Vektor NullVektor;
};

extern Vektor NullVektor;

#endif /* VEKTOR_H_ */
```

h-Datei
enthält die exportierten Definitionen des Moduls: das was der Compiler für die Übersetzung anderer Module von diesem wissen muss.

cpp-Datei
enthält ausführbaren Code, den Code aus dem der Compiler Maschinen-Befehle machen soll.

const hinter Methodenkopf:
Methode wird das Objekt nicht verändern
const vor Adressparameter:
Methode wird den Parameter nicht verändern

Datei: Vektor.cpp

```
#include "Vektor.h"

Vektor::Vektor(double x, double y) : x(x), y(y) {}

Vektor Vektor::add(const Vektor & that) const { return Vektor(x+that.x, y+that.y); }

Vektor NullVektor = Vektor(0, 0);

ostream& operator<<(ostream& s, const Vektor& v) {return s << "[" << v.x << ", " << v.y << "]";}
```

Klassendefinition

„Stack-orientiert“ oder „Zeiger- / Heap-orientiert“

in C++ kann der Speicherort aller Daten genau kontrolliert werden, Klassen können alternativ Zeiger-/Heap-orientiert oder Zeigerlos-/Stack-orientiert definiert werden.

```
class Vektor {
friend ostream& operator<<(ostream& s, const Vektor& v);
  double x;
  double y;

public:
  Vektor(double x, double y);
  Vektor add(Vektor that);
  static Vektor NullVektor;
};
```

```
Vektor v(1,2);
```

Zeiger-lose (Stack-) Variante
(so in Java nicht möglich)

```
class Vektor {
friend ostream& operator<<(ostream& s, const Vektor * v);
  double x;
  double y;
public:
  Vektor(double x, double y);
  Vektor * add(Vektor * that);
  static Vektor * NullVektor;
};
```

```
Vektor v = new Vektor(1,2);
```

Zeiger- (Heap-) Variante
(entspricht weitgehend der Java-Version)

C/C++ vs Java : Klassendefinition

Klassendefinition Zeiger-/Heap-orientierte Version komplett (~ Java-Version)

Vektor Modul / Klasse

Datei: Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

#include <iostream>
using namespace std;

class Vektor {
friend ostream& operator<<(ostream& s, const Vektor * v);
  double x;
  double y;
public:
  Vektor(double x, double y);
  Vektor * add(Vektor * that);
  static Vektor * NullVektor;
};

extern Vektor * NullVektor;

#endif /* VEKTOR_H_ */
```

Test.cpp

```
#include <iostream>
#include "Vektor.h"
using namespace std;

int main() {
  Vektor * v1 = new Vektor(1, 2);
  Vektor * v2 = v1->add(
                        new Vektor(2,2));
  v2 = v1->add(NullVektor);
  cout << v2 << endl;
  return 0;
}
```

Datei: Vektor.cpp

```
#include "Vektor.h"

Vektor::Vektor(double x, double y) { this->x = x;  this->y = y; }

Vektor * Vektor::add(Vektor * that) {  return new Vektor(x+that->x, y+that->y);  }

Vektor * NullVektor = new Vektor(0, 0);

ostream& operator<<(ostream& s, const Vektor * v) {
  return s << "[" << v->x << ", " << v->y << "]";
}
```

C/C++ vs Java : Klassendefinition

Überladene Operatoren

Operatoren können wie Methoden umdefiniert („überladen“) werden

Datei: Vektor.h

```
#ifndef VEKTOR_H_
#define VEKTOR_H_

#include <iostream>
using namespace std;

class Vektor {
friend ostream & operator<<(ostream& s, const Vektor& v);
  double x;
  double y;
public:
  Vektor(double x, double y);
  Vektor operator+ (const Vektor & that) const;
  static Vektor NullVektor;
};

extern Vektor NullVektor;

#endif /* VEKTOR_H_ */
```

Datei: Test.cpp

```
#include <iostream>
#include "Vektor.h"
using namespace std;

int main() {
  Vektor v1(1, 2);
  Vektor v2 = v1 + Vektor(2,2);
  v2 = v1 + NullVektor;
  cout << v2 << endl;
  return 0;
}
```

Vektor-Addition mit
Vektor + Vektor

Datei: Vektor.cpp

```
#include "Vektor.h"

Vektor::Vektor(double xx, double yy) : x(xx), y(yy) {}

Vektor Vektor::operator+ (const Vektor & that) const {
  return Vektor(x+that.x, y+that.y);
}
Vektor NullVektor = Vektor(0, 0);

ostream& operator<<(ostream& s, const Vektor& v) {
  return s << "[" << v.x << ", " << v.y << "]";
}
```

Destruktoren

In c++ muss der Heap selbst verwaltet werden.

Destruktoren helfen dabei

delete ruft den Destruktor des Gelöschten automatisch auf

Rekursive Aufrufketten zur Speicherbereinigung möglich und üblich
(delete ~> Destruktor ~> delete ~ Destruktor ~> …)

```
#ifndef MENGE_H_
#define MENGE_H_

#include <iostream>
using namespace std;

class Menge {
  friend ostream& operator<<(ostream &, const Menge &);

  int * elemente; // zeigt auf die Elemente im Heap
  int anzahl;  // aktuelle Anzahl der Elemente
  int groesse; // Groesse von elemente

public:
  Menge();      // leere Menge
  Menge(int);   // Menge mit einem Element
  Menge(int, int);  // Menge mit 2 Elementen

  ~Menge();          // Destruktor

  void fuegEin(int e);  // Fuege Element ein
};

#endif /* MENGE_H_ */
```

Klassen deren Objekte Speicherplatz auf dem Heap beanspruchen, müssen für dessen Freigabe sorgen.

Destruktoren / Beispiel Menge

```
#include "Menge.h"

Menge::Menge() : elemente(new int[5]), anzahl(0), groesse(5) {}

Menge::Menge(int e) : elemente(new int[5]), anzahl(1), groesse(5) { elemente[0] = e; }

Menge::Menge(int e1, int e2) : elemente(new int[5]), anzahl(2), groesse(5) {
  elemente[0] = e1;
  elemente[1] = e2;
}

Menge::~Menge() { delete elemente; }

void Menge::fuegEin(int e) {
  if (anzahl == groesse) {
    int nGroese = groesse*2;
    int * nElemente = new int[nGroese];
    for (int i=0; i<anzahl; i++) { nElemente[i] = elemente[i]; }
    groesse = nGroese;
    elemente = nElemente;
  }
  elemente[anzahl] = e;
  anzahl++;
}

ostream& operator<<(ostream& s, const Menge& m) {
  s << "[";
  if (m.anzahl > 0) { s << m.elemente[0]; }
  for (int i=1; i<m.anzahl; i++) { s << ", " << m.elemente[i]; }
  return s << "]" << endl;
}
```

Destruktor: Gibt den Platz frei und rufe – falls definiert – weitere Destruktoren.

C/C++ vs Java : Klassendefinition

Vererbung

- Der Vererbungsmechanismus in C++ ist reichhaltiger / komplexer und
  kann feiner gesteuert werden als in Java
- Zeiger- / Heap-basierte Klassen verhalten sich dabei anders
  als zeigerlose / Stack-basierte Klassen (Polymorphismus nur via Zeiger)
- Interfaces gibt es nicht als eigenständiges Konzept

C/C++ vs Java : Klassendefinition

Vererbung in C++

Beispiel Java-Interface in C++

Interface

```
public interface Menge {
  void fuegEin(int e);
}
```

Implementierung

```
import java.util.ArrayList;
import java.util.List;

public class ListenMenge implements Menge {
  private List<Integer> elemente = new ArrayList<>();

  @Override
  public void fuegEin(int e) {
    elemente.add(e);
  }

}
```

Verwendung

```
public class MengenTest {

  public static void main(String[] args) {
    Menge m = new ListenMenge();
    m.fuegEin(5);
  }

}
```

Menge.h

```
#ifndef MENGE_H_
#define MENGE_H_

class Menge {
public:
  virtual void fuegEin(int) = 0;
  virtual ~Menge() {}
};
#endif /* MENGE_H_ */
```

ListenMenge.h

```
#ifndef LISTENMENGE_H_
#define LISTENMENGE_H_

#include <list>
#include "Menge.h"
using namespace std;

class ListenMenge : public Menge {
  list<int> elemente;
public:
  ListenMenge();
  virtual void fuegEin(int);
};

#endif /* LISTENMENGE_H_ */
```

ListenMenge.cpp

```
#include "ListenMenge.h"

ListenMenge::ListenMenge()  {}

void ListenMenge::fuegEin(int e) {
  elemente.push_front(e);
}
```

MengenTest.cpp

```
#include <iostream>
#include "Menge.h"
#include "ListenMenge.h"
using namespace std;

int main() {
  Menge * m = new ListenMenge();
  m->fuegEin(5);
  return 0;
}
```

Übersetzung und Ausführung

- **Compilierte Sprachen**

  Sowohl Java als auch C++ werden übersetzt: es sind beides compilierte Sprachen

  im Gegensatz zu interpretierten Sprachen deren Quellprogramme direkt von anderen Programmen ausgeführt werden

- **Bytecode vs Maschinencode**

  Java-Code wird zu Bytecode übersetzt zu dessen Ausführung ein weiteres Programm benötigt wird

  C/C++-Code wird in Maschinencode übersetzt der direkt von der CPU ausgeführt werden kann.

Sprachkonzepte

Java als vereinfachtes und standardisiertes C++

C++ bietet in vielen Punkten Wahlfreiheit an denen Java nur eine Standardlösung bietet:

- **Speicherverwaltung**

  In C/C++ selbst organisiert mit vielen (komplexen) Sprachkonstrukten zur Feinsteuerung

  In Java: Standardlösungen von den nicht abgewichen werden kann.

- **Programmorganisation**

  Java gibt Programmstruktur und Datei-Organisation vor. C++ überlässt es komplett dem Entwickler (de facto werden in C/C++ stets Konventionen befolgt)

- **Vererbung**

  Java bietet Interfaces und eine Form der Vererbung.

  C++ erlaubt einen direkten Zugriff auf mehrere Mechanismen der Vererbung

  (Was wird wird wie übernommen / überschrieben, ..)

- **Weiteres**

  Bei weiteren nicht angesprochenen Sprachelementen (z.B. Generics vs Templates) ist die Sachlage entsprechend. C++: mächtig, komplex, viele Möglichkeiten. Java: relativ einfache Standardlösungen.

- **Java füllt Lücken in C++**

  Für einige Dinge bei denen man bei C++ auf externe Lösungen zurückgreifen muss, bietet Java eine Sprach-integrierte Lösung. (z.B. GUIs, Threads, STL-Bibliotheken).

C/C++ vs Java : Zusammenfassung

"The world is built on C++"
Herb Sutter – C++ Urgestein

Einsatz

C: Hardware-nahe (System-) Programme

C ist die Sprache der Wahl für Programme die direkt mit dem Betriebssystem oder der Hardware interagieren.

C: C++ ohne Klassen, Objekte, Templates ...

http://www.youtube.com/watch?v=i2fhNVQPb5I&feature=related

C++: hoher Lern- / Entwicklungs-Aufwand

Der Aufwand für ein C++-Entwicklerteam lohnt sich nur für professionelle Entwicklungen und für Programme, an die

- **hohe Anforderungen an Geschwindigkeit, Effizienz gestellt werden und die**
- **Hardware- oder Betriebssystem-nah sind**

C++ ist mächtig aber schwierig und bietet sehr viele Möglichkeiten - auch die etwas falsch zu machen. Es ist eine Sprache für Profis und völlig ungeeignet für Leute die nicht genau wissen was sie tun. (Dieser Rat wird leider zu selten beachtet!)

Nur wer C++ verstanden hat darf sich zu Recht als Software-Profi bezeichnen!

C++ wird immer noch weiterentwickelt

- **verbesserte Effizienz der Programme**
- **verbesserte Produktivität der Entwickler**

http://www.youtube.com/watch?v=x1TsOHyJPpw&feature=related

C/C++ ist tief im Herzen von allen wichtigen Dingen.