A practical guide to C++

Janis Fehr
fehr@informatik.uni-freiburg.de

SommerCampus2004

Kurstag 3: Agenda

- Wiederholung Kurstag 2
- Die Standard Template Library
    - Container
    - Algorithmen
    - Speicherverwaltung
- Templates
- Exeptions

Wiederholung Kurstag 2

Vererbung:

```
class Parent
{
   public:
      void getA();
      int foo();
      ...
   private:
      ...
}

class Child : public Parent
{
   public:
      void getB();
      int foo();
   private:
      ...
}
```

Basis Klasse

Vererbung

Zugrifsrechte auf die geerbten Methoden

Eigene Methode

Überschreibe geerbte Methode

Wiederholung Kurstag 2

Bei Projekten wird die Definition der Klassen von der Implementation getrennt.

MyClass.hh

```
#include<iostream>

Class MyClass
{
   public:
      int foo(int a);
      ...
   private:
      ...
};
```

MyClass.cc

```
#include ''MyClass.hh''

int MyClass::foo(int a)
{
         ...


}
```

programm.cc

```
#include ''MyClass.hh''
int main()
{
   MyClass a = new MyClass();
```

Die Standard Template Library

Die STL erweitert C++ um viele Datenstrukturen, Funktionen und Algorithmen.

- Strings
- Streams
- Numerik
- **Container**
  - list
  - vektor
  - map
- **Algorithmen**
  - sort
  - for_each
  - ...

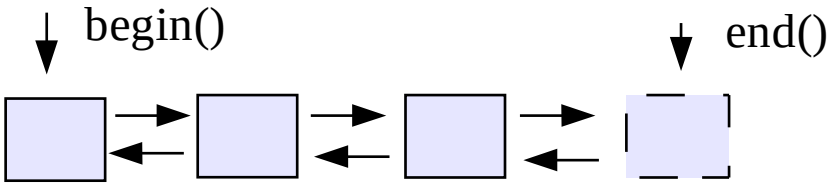
```
#include <vector>
...

vector<int> myVector;
```

Container könne beliebige typen aufnehmen

Standard Container

Vector

Dynamischer Array:



Einfügen: z.B.: `myVector.push_back(item)`

Element Adressierung: `myVector[i]`

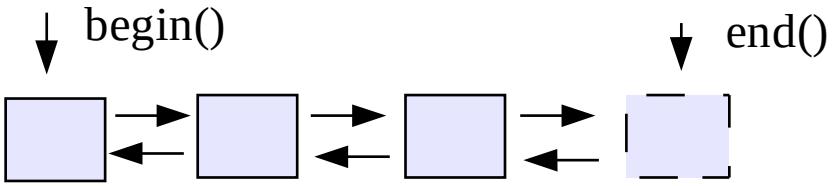
Algorithmen: `myVector.sort()`

Größe ändern: `myVector.resize(newsize)`

Weitere Container Klassen:

List, Stack, Map, Queue, Set ... alles was man aus Info 2 kennt

Container Iteratoren



Mit Iteratoren lassen sich die Elemente leicht durchlaufen.

```
std::vector<int>::iterator p;

for(p=myVector.begin();p!=myVector.end();++p)
{
   std::cout<<*p;
}
```

Def. Iterator

Zeiger auf das aktuelle Element

Elementzugriff

Speicher Organisation in der STL

- Die STL legt alle Objekte auf dem Heap an
- Container haben einen internen Garbage Collector
  - Automatisches Vergrößern
  - Automatisches Verkleinern
  - Automatisches Delete
- Die STL ist Ausnahme fest
- Die STL ist sehr schnell
- Die STL ist Speicher optimal
- Die STL ist **einfach** :-)

Projekt:

Schreibt eine Klasse Bild, welche die Pixel aufnimmt.

- Die Pixel sollen in einem Container gehalten werden
- Die Größe des Bildes soll veränderbar sein
- Adressierung einzelner Pixel
- Einlesen von Pixeln durch einen Stream (Überlade <<)
- Ausgabe der einzelnen Farbwerte von Pixeln

Wie funktioniert die STL intern ?

Funktions Templates:
Schreibe eine Funktion unabhängig vom bearbeiteten DatenTyp.

```
double min_double(double a, double b)
{
   if(a<b) return a;
   return b;
}

int min_int(int a, int b)
{
   if(a<b) return a;
   return b;
}
```

Funktions Templates

```
template<typename DataType>
DataType min(DataType a, DataType b)
{
   if(a<b) return a;
   return b;
}

...

int main()
{
   int g,j;
   int l = min<int>(a,b);
   ...
}
```

Compiler

```
int min(int a, int b)
{
   if(a<b) return a;
   return b;
}
```

Klassen Templates

```
template<typename DataType, int MaxSize>
class MyVector
{
   public:
      DataType& operator[](int index)
      {
         return data[index];
      }
   private:
      DataType data{MaxSize];
}
```

Template Typ und Parameter

Überlade den [] Operator

Interne Datenhaltung

Spezialisierung

```
//allgemeine Version
template<typename T, int size>
void MyVector::multWith(double a)
{
   for(int i=0;i<size;i++)
   {
      data[i]*=a;
   }
}
```

Skalare Multiplikation für MyVector

```
template<>
void MyVector<float,3>::multWith(double a)
{
   data[0]*=a;
   data[1]*=a;
   data[2]*=a;
}
```

Schneller ohne Schleife

Projekt

Schreibt eine Bild Template Klasse, die Pixel und BinPixel aufnehmen kann.

Exeptions

Fehler können nicht immer schon vor der Laufzeit ausgeschlossen werden, z.B. Division by Zero oder dem Versuch ein File zu Öffnen das es nicht gibt.

Fängt man diese Fehler nicht ab, beendet sich das Programm vorzeitig :-(

Natürlich kann man in jeder Methode selbst eine Fehlerbehandlung implementieren, aber bei Libraries z.B. ist so kein Forwarding des Fehlers an das externe Programm möglich...

try, throw, catch ...

```
try
{
    std::ifstream file(''data.txt'');

    std::string header;

    file >> header;

    if (header == ''P1'')
    {
        throw WrongFormatError();
    }
}

catch(FileOpenError& err)
{
    std::cout<<''file not found'';
}

catch(WrongFormatError& err)
{
    std::cout<<''file in wrong format'';
}

catch(...)
{
    std::cout<<''unknown Error'';
}
```

Try Block,

manuelle Exeption

Catch automatischen Exeption

Catch manuelle Exeption

Wirklich drei Punkte !

Projekt

Schreibt eine Klasse read_ppm(), die Bilder auslesen kann. Benutzt Exeptions um mögliche Fehler abzufangen.