Prof. Dr. Andreas Butz

Dipl.-Medieninf. Hendrik Richter
Dipl.-Medieninf. Raphael Wimmer

Computergrafik 1 Übung

Einführung in C++

C++

- prozedurale Programmiersprache mit Zusatzfunktionen
- Erweiterung von C (Objektorientierung)
- Ziel: Effizienz
- Erweiterbarkeit durch Definierbarkeit neuer Typen
- Mehrfachvererbung ist in C++ möglich, in Java jedoch nicht.
- Definition/Deklaration und Implementierung von Variablen/Klassen + Methoden wird in C++ üblicherweise getrennt (muss aber nicht) -> Header und Source-Files

http://www.cs.tamu.edu/news/items?id=1797

Bjarne Stroustrup
(* 11. Juni 1950 in Århus, Dänemark) Professor der Informatik an der Texas A&M University. Bekanntheit erlangte er vor allem durch die Entwicklung der Programmier-sprache C++.

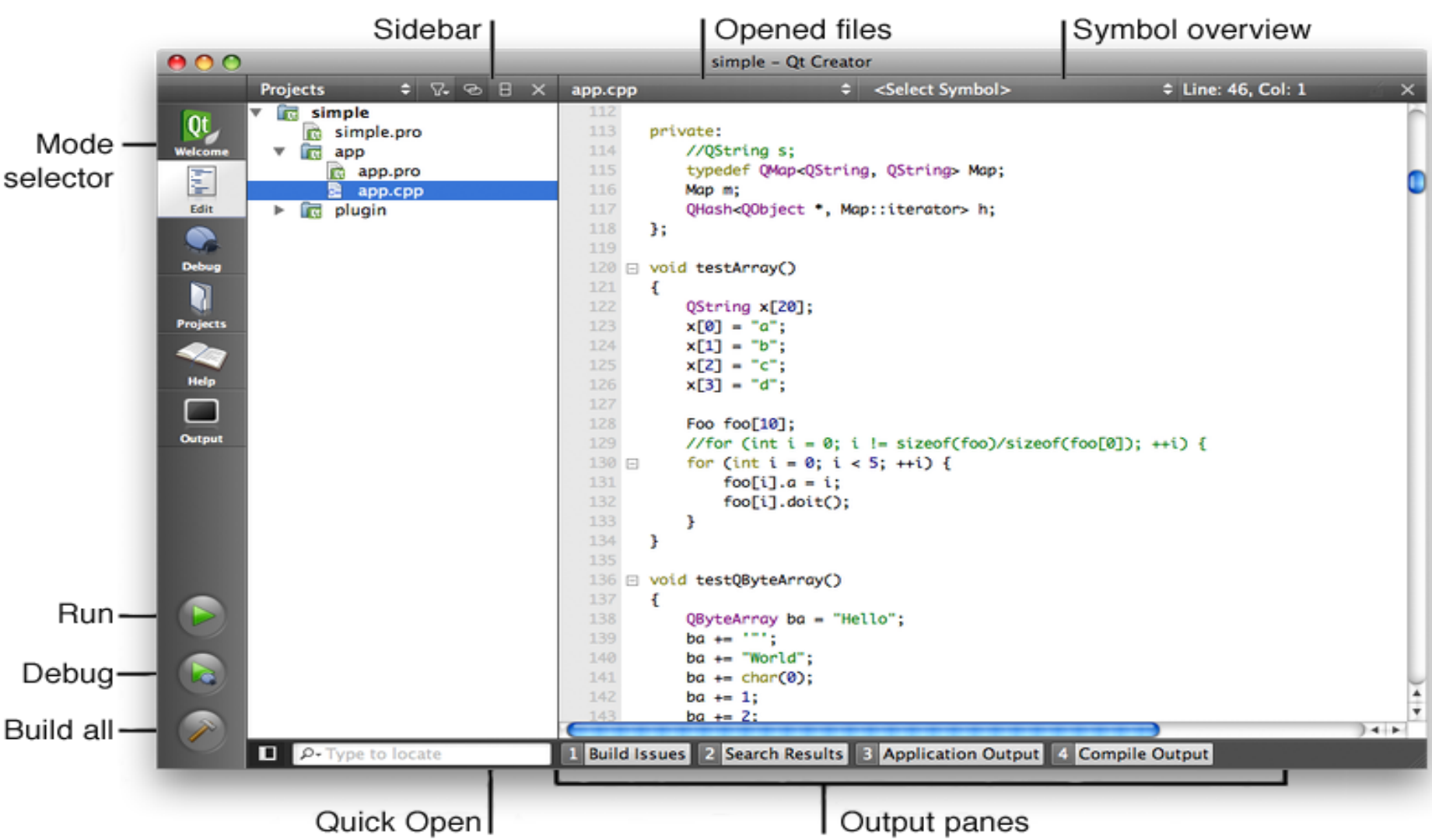
Quelle: Wikipedia

QT

http://www.epyx-online.de/wp-content/2009/10/qt-logo.jpg

- http://qt.nokia.com/products
- „cross-platform application and UI framework“

- C++ Klassenbibliothek
  - GUI
  - 3D Grafik mit OpenGL
  - Multithreading
  - Embedded Windowing System
  - Inter-Object Communication
  - 2D Grafik
  - …
- QT Creator: Cross-Platform DIE
  - C++ code editor
  - …




```
112
113   private:
114       //QString s;
115       typedef QMap<QString, QString> Map;
116       Map m;
117       QHash<QObject *, Map::iterator> h;
118   };
119
120   void testArray()
121   {
122       QString x[20];
123       x[0] = "a";
124       x[1] = "b";
125       x[2] = "c";
126       x[3] = "d";
127
128       Foo foo[10];
129       //for (int i = 0; i != sizeof(foo)/sizeof(foo[0]); ++i) {
130       for (int i = 0; i < 5; ++i) {
131           foo[i].a = i;
132           foo[i].doit();
133       }
134   }
135
136   void testQByteArray()
137   {
138       QByteArray ba = "Hello";
139       ba += '"';
140       ba += "World";
141       ba += char(0);
142       ba += 1;
143       ba += 2;
```

Aufbau eines C++ Programms

http://de.wikibooks.org/wiki/C%2B%2B-Programmierung/_Inhaltsverzeichnis

http://tutorial.schornboeck.net/inhalt.htm

Aufbau eines C++ Programms

- Jedes (lauffähige) C++ Programm benötigt einen Einstiegspunkt.
- Das ist die reservierte Methode `int main()`
  - `int main()`
  - `int main(int argc, char** argv)`
- Die `main` Methode ist statisch d.h. sie kommt pro Prozess nur *einmal* vor.
- Die `main` Methode wird als erstes ausgeführt, wenn ein Programm gestartet wird.
- Wenn die `main` Methode beendet wird, wird auch das Programm beendet.

Aufbau eines C++ Programms

ausprobieren!

Ein einfaches Beispielprogramm:

```
#include <iostream>
int main()
{
  std::cout << “CG 1” << std::endl;
  return 0;
}
```

- Kompilieren z.B. durch
  `gcc cplus.cpp –o cplus` oder
  `C:\Qt\2010.02.1\mingw\bin>`
  `mingw32-g++.exe cplus.cpp -o cplus`
- Ausgabe bei Programmausführung: ***CG 1***
- Rückgabewert 0 bedeutet, dass das Programm ohne Fehler beendet wurde
- Rückgabewerte ungleich 0 zeigen an, mit **welchem** Fehler das Programm beendet wurde (werden vom Programmierer festgelegt).

Namespaces

Namensräume (Namespaces)

- C++ Programme sind oft modular zusammen gesetzt.
- Klassen-/Funktionsnamen können also im Konflikt zu einander stehen
  - Code kommt von vielen verschiedenen Quellen
- Um Konflikte zu vermeiden gibt es sogenannte `Namespaces`
- Im Beispielprogramm haben Sie dieses Konstrukt kennen gelernt:
  `std::cout << “foo” << std::endl;`
- `cout` ist eine Funktion des Namespace `std`.
- Normalerweise müssen Funktionen immer mit ihrem Namespace-Präfix aufgerufen werden.
- Außer es wird eine `using` Direktive verwendet.

```
using namespace std;
…
cout << “Blah blah blah!” << endl;
```

Eigene Namensräume definieren

- Um Konflikte mit Funktionen im globalen Namensraum (ohne Präfix) zu vermeiden sollten eigene Programme immer einen eigenen Namespace definieren.

```
namespace A {
    typedef … String; // definiert Datentyp A::String
    void foo (String);       // definiert A::foo(String)

}
```

- Das Schlüsselwort `namespace` ordnet die darin befindlichen Symbole dem angegebenen Gültigkeitsbereich zu
- `A::String s1;` ist also unterschiedlich von `B::String s2;`

Kontrollstrukturen

Kontrollstrukturen

Drei Schleifen (analog zu Java):

- `for([start];[bedingung];[ende]){…}`
- `while([bedingung]){…}`
- `do{…}while([bedingung]);`
- …

Beispiele:

- `for(int i=0; i<5; i++) {…}`
- `while(i < 5) {…}`
- `do {…} while(i < 5);`
- …

Ein-/Ausgabe

Unformatierte Ausgabe

Unformatierte Ausgabe:

- Ausgabestrom: `std::cout`
- Format muss nicht angegeben werden, da sich dies aus dem Datentyp ergibt (ähnlich der Variante `println()` in Java)
- Nicht auf Standardtypen beschränkt
- Alle Teil-Zeichenketten werden mit dem Operator << getrennt

Beispiel:

```
int i = 7;
std::cout << “#: “ << i << std::endl;
```

Formatierte Ausgabe

Formatierte Ausgabe:

- Funktion `printf()`
- Argumente: Eine Zeichenkette plus ALLE vorkommenden Variablen

Beispiel:

```
printf(“int: %d, hex %x \n”, 42, 42);
```

Weitere Erläuterungen:

http://www.cplusplus.com/ref/cstdio/printf.html

Benutzereingaben

Eingaben:

- Eingabestrom: `cin`
- Kein `endl` erforderlich
- `fflush(stdin)` leert den Eingabestrom

Beispiel:

```
int i;
cin >> i;        // Einlesen eines int
fflush(stdin);
```

Bibliotheken

Die Standardbibliothek STL

- STL = Standard Template Library
- C++ hat keine geschlossene API wie Java (es gibt aber einen ANSI Standard)
- Viele elementare Datentypen und Algorithmen sind in der STL implementiert (Vektoren, Listen, Iteratoren, Sortieralgorithmen, …)
- Ausführliche Beschreibungen finden Sie im Web und in [N. Josuttis] oder [B.Strousstrup]

Beispiel


```
#include <iostream>
#include <string>

int main() {
    std::string s = “Was ist die Lösung”;
    std::cout<< s << std::endl;
    std::cin>> s;
    if (s.compare(“per aspera ad astra”)==0) {
      std::cout<< “Richtig!”<< std::endl;
    }
    return 0;
}
```

Bibliotheken (Libraries)

Bibliotheken (*.lib,*.so,*.dll) sind vorkompilierte

Binärdateien ohne Einstiegspunkt:

- Zeitvorteil, da nicht immer alles neu kompiliert werden muss
- Wiederverwendbarkeit
- Platz- und Geschwindigkeitsvorteil, da häufig benötigter Code nur einmal auf der Platte gespeichert werden muss (und teilweise auch nur einmal im Speicher)

Fremde Bibliotheken

Einbinden von fremden Bibliotheken

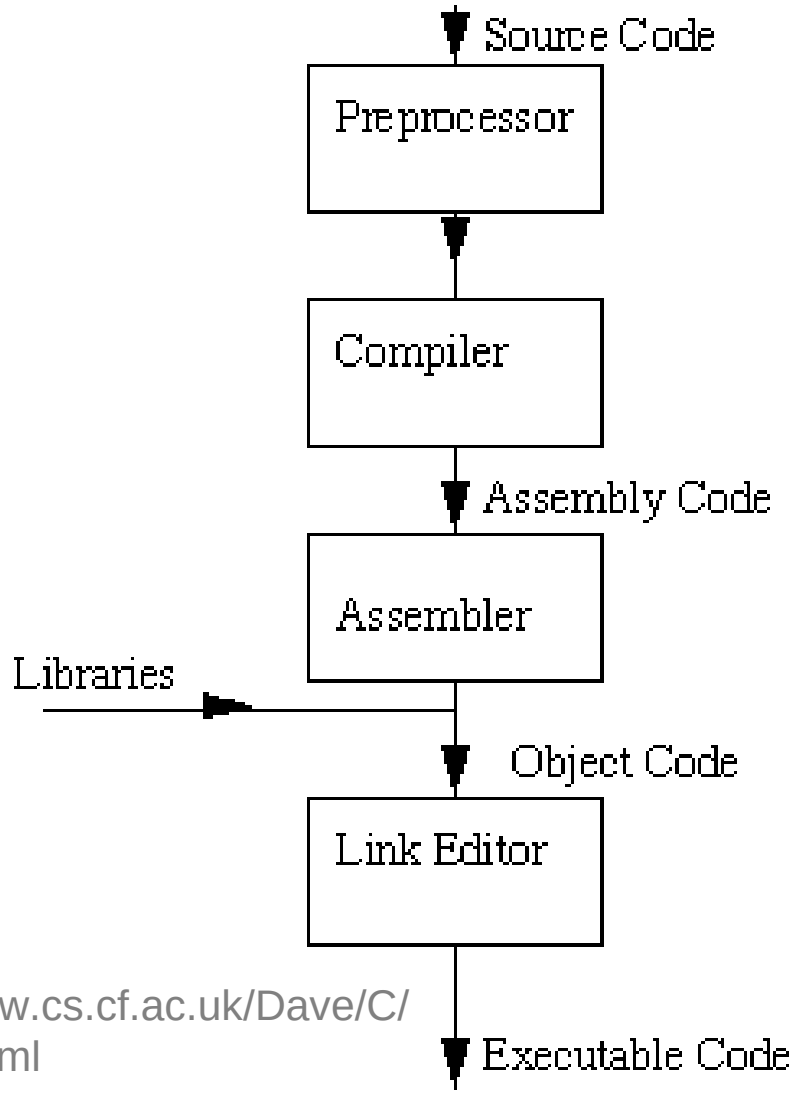
- Im eigenen Code den/die Header einbinden (für die Methoden-Deklarationen)
- Den Linker-Suchpfad so anpassen, dass Bibliotheken gefunden werden können (lib path)
- Den Compiler-Suchpfad so anpassen, dass Headerdateien gefunden werden können (include path)

Präprozessor, Compiler, Linker

Compiler Collection

Compiler Collection

- Eine Sammlung von Programmen, die man zur Programmerstellung benötigt.
- **Präprozessor** – Textersetzung in Sourcefiles vor dem eigentlichen Kompilieren
- Übersetzer [**Compiler**] Übersetzt Programm(fragmente) von C++ in Binärcode (maschinenlesbar)
- Verknüpfer [**Linker**] setzt Binärcodefragmente zu lauffähigem Programm zusammen



http://www.cs.cf.ac.uk/Dave/C/node3.html

Präprozessor

- Anweisungen für den Präprozessor fangen mit # an
- `#include "file.h"` fügt die Header-Datei *.h ein
- `#define FOO 1` ersetzt überall im Programmtext `FOO` mit 1

```
#ifdef BAR
    Code 1
#else
    Code 2
#endif
```

- Fügt im Programmtext Code 1 ein, falls BAR definiert wurde, ansonsten Code 2

Präprozessor Anwendungen

– Schutz vor Mehrfach-Einbindung von Headern

```
#ifndef __MYCLASS_H__
#define __MYCLASS_H__

#include …

class MyClass { … };

#endif  // __MYCLASS_H__
```

– Ersetzungsworte immer in Großbuchstaben

Linker

- Das eigentliche Erstellen von Programmen wird vom Linker erledigt
- Linker ersetzt symbolische Funktionen in Binärcode mit Adressen im Speicher.
- Außerdem werden eigene Programmfragmente und verwendete Bibliotheken zusammen gesetzt.
- Zwei Arten des Verknüpfens möglich:
  - Statisches Verknüpfen
  - Dynamisches Verknüpfen

Statisches Verknüpfen:

- Einfachste Methode
- Erzeugt eine große Datei
- Bei Änderungen an der Bibliothek muss neu kompiliert werden

Dynamisches Verknüpfen:

- Komplizierter Mechanismus der auf unterschiedlichen BS verschieden funktioniert.
- Kleinere, modulare Dateien
- Bibliotheken können von mehreren Programmen gleichzeitig benutzt werden
- Bei Änderungen an der Bibliothek muss nicht neu kompiliert werden

Header-Dateien

Header Dateien

Die Header Datei

- Enthält Definitionen und Deklarationen -> eigentliche Implementierung erfolgt dann in der Source (.cpp) Datei.
- Außerdem enhält der Header Präprozessoranweisungen, z.B. #include um externe Klassen einzubinden.

1. Klassendeklaration
```
class MyClass {…};
```
2. Typdefinition
```
struct Position { int x, y };
```
3. Aufzählungen
```
enum Ampel { red, yellow, green };
```
4. Funktionsdeklaration
```
int rechtEckFlaeche ( int width, int height);
```
5. Konstantendefinition
```
const float pi = 3.141593;
```
6. Datendeklaration
```
int a = null;
```
7. Präprozessoranweisungen
```
#include <iostream>
#include „myheader.h“
#define VERSION12
#ifdef VERSION12
```

Header-Datei

myClass.h

```
class MyClass : MyParentClass
{ // die Klasse MyClass erbt von MyParentClass
   public:
        MyClass();                  //standard Konstruktor
        MyClass(std::stringtext); //zweiter Konstruktor
        virtual ~MyClass;           //Destruktor

        virtual int func()=0;       //eine rein virtuelle(=abstrakte)
   Funktion
        static double funct();      //eine statische Funktion
        static int m_someNumber;    //eine statische Membervariable

   protected:
        virtual int fun();          //eine virtuelle Funktion

   private:
        void fu();                  //eine Funktion
        std::string m_someString; //eine Membervariable
};
```

Implementierung

myClass.cpp

```
#include “myClass.h”

int MyClass::m_someNumber(5);

MyClass::MyClass(){                          //Standardkonstruktor
   m_someString= “EineIntialisierungvomText”;
}
MyClass::MyClass(std::stringtext)    //zweiter Konstruktor
   m_someString= text;
}

MyClass::~MyClass(){}                        //Destruktor

void MyClass::fu(){}                         //Impl. einer Funktion
int MyClass::fun(){return 4;}                //Impl. einer virtuellen Funktion
double MyClass::funct(){return 2;}    // Impl. einer statischen Funktion
```

Nachtrag: Virtuelle Funktionen

Virtuelle Funktionen

- Member Funktionen können als virtuell deklariert werden
  - Virtuelle Funktionen können in abgeleiteten Klassen überschrieben werden
  - Keyword *virtual*
  - Normalerweise neue Funktionalität in abgeleiteter Klasse
- Auch nicht virtuelle Funktionen können überschrieben werden
  - Nicht virtuelle Methoden werden zur Compilezeit ausgewählt
  - Virtuelle Methoden werden zur Laufzeit ausgewählt

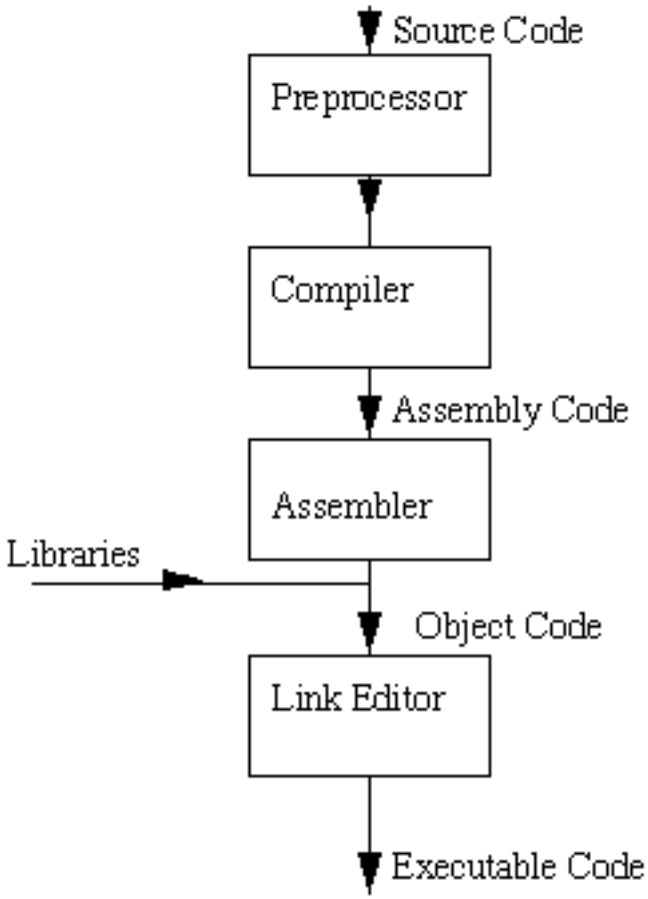
Beispiel: Virtuelle Funktionen

```
class Window // Base class
{
  public:
     virtual void Create() // virtual function
     {
          cout << "Base class Window„ << endl;
     }
};
```

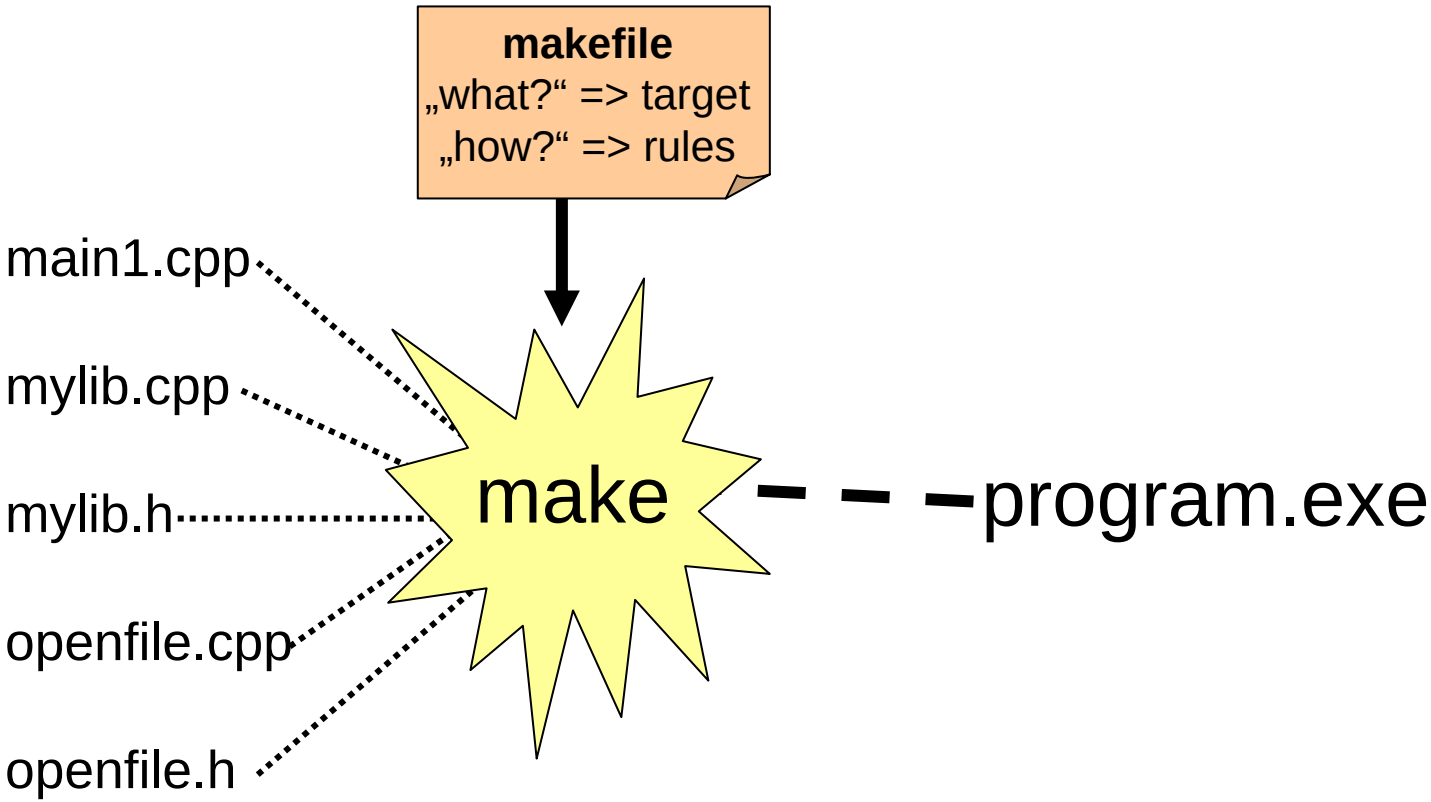
```
class Button : public Window
{
  public:
     void Create()
     {
         cout << "Derived class Button„ << endl;
     }
};
```

```
void main()
  {
       Window  *x, *y;

       x = new Window();
       x->Create();

       y = new Button();
       y->Create();
  }

  Ausgabe:
      Base class Window
      Derived class Button
```

Make


Source Code
Preprocessor
Compiler
Assembly Code
Assembler
Libraries
Object Code
Link Editor
Executable Code

Makefiles



http://faculty.stedwards.edu/laurab/help/makefilehelp.html

```
#makefile

program.exe : main1.o mylib.o openfile.o
	g++ -o program.exe main1.o mylib.o openfile.o

main1.o: main1.cpp openfile.h mylib.h
	g++ -c main1.cpp

mylib.o: mylib.cpp mylib.h
	g++ -c mylib.cpp

openfile.o: openfile.cpp openfile.h
	g++ -c openfile.cpp

clean:
	rm *.o program.exe

# END OF MAKE FILE
```

tab

Programm „builden“:
`mingw32-make –f makefile`

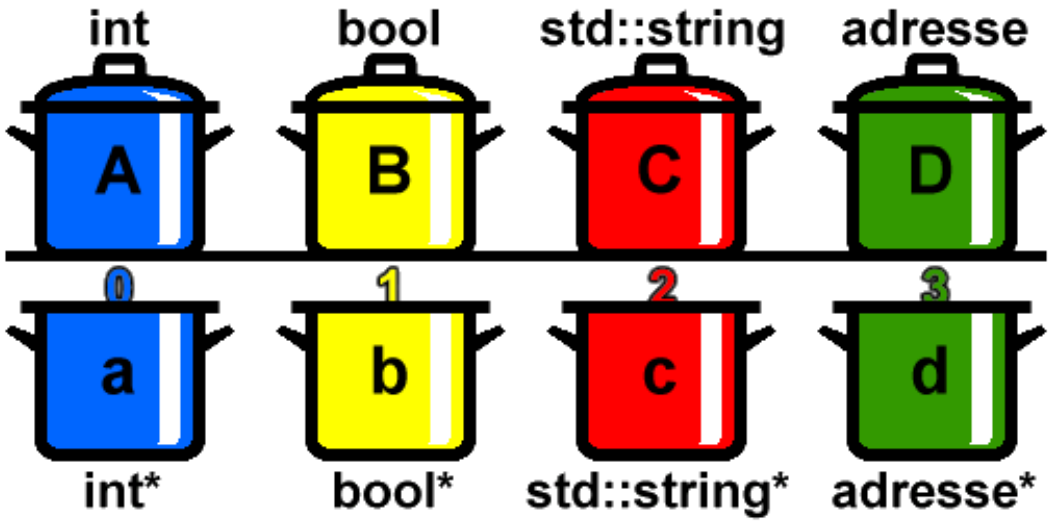
program.exe und .o files aufräumen:
`mingw32-make –f makefile clean`

http://faculty.stedwards.edu/laurab/help/makefilehelp.html

Referenzen & Pointer

http://home.netcom.com/~tjensen/ptr/pointers.htm

Pointer



http://www.highscore.de/cpp/einfuehrung/zeiger.html

&A liefert Speicheradresse von A

int* a speichert die Adresse von A. Adresse a hat den Datentyp int*

***a** liefert die Variable, deren Positionsnummer in a gespeichert ist

Pointer Notation

C++ ist formatfrei, deswegen gibt es unterschiedliche

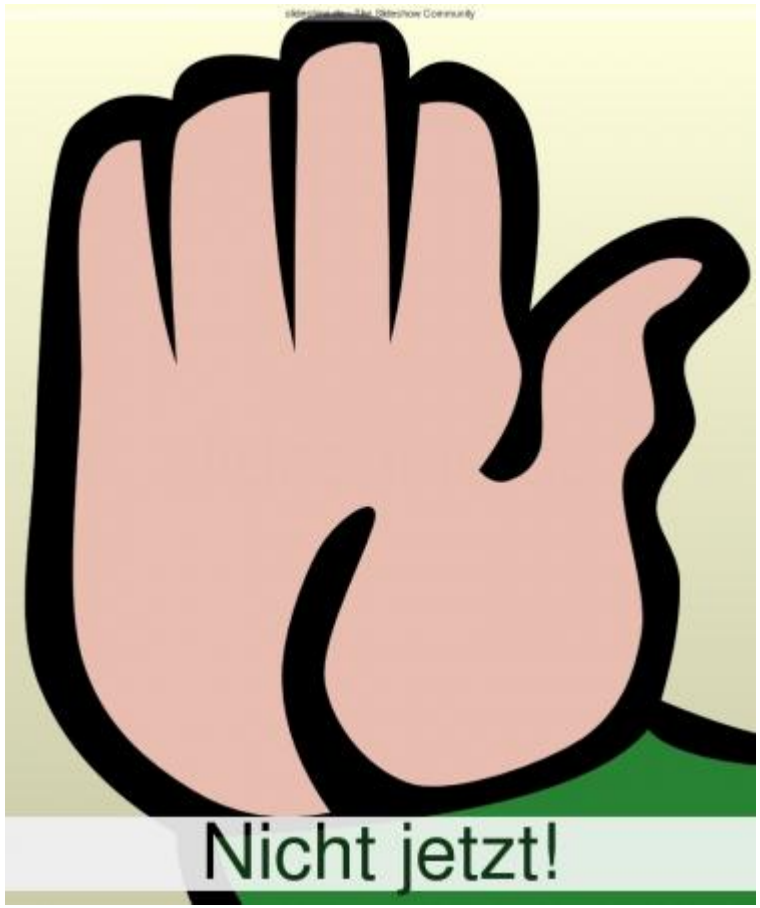
Notationen für Pointer:

```
int *xp;
int * xp;
int*xp;
int* xp;          // alle Notationen sind äquivalent
```

Vorsicht bei Verwendung der letzten Notation:

```
int* p1, p2;      // Erzeugt?
int *p1, p2;
```

Speicherzuweisung



http://www.slideshine.de/28645

Arrays und Pointer

Arrays (Felder):

- Anlegen eines Arrays: `int werte[10];`
- Zugriff auf Feldelemente mit `[]` Operator

Beispiel:

```
// ein Byte Array (10MByte) um den Wert 5 erhöhen
char* array = new char[10000000];
for (int i=0; i < 10000000; i++) {
  array[i] += 5;
}
delete[] array;
```

Pointer und Geschwindigkeit

Effizientes Programmieren mittels Zeigerarithmetik

```
char* array= new char[10000000];    // Zeigt auf erstes Element
char* endOfArray= array+10000000;   // Zeigt auf letztes+1
   Element

for (char* i=array; i < endOfArray; i++) {
   (*i) += 5;   //Wert des Elements, auf das Zeiger i zeigt
}
delete[] array;
```

http://campar.in.tum.de/twiki/pub/Chair/TeachingWs09PP/01_Einfuehrung_Cpp.pdf

Diese Implementierung ist funktional identisch

ABER deutlich schneller. Warum?

Call-by-value vs. Call-by-Reference

Auch die Geschwindigkeit von Methodenaufrufen kann durch den Einsatz von Pointern beeinträchtigt werden.

```
Eintrag suchFunktion(string s, Telefonbuch t) {
   //kopiert beim Aufruf das ganze Telefonbuch
   ...
   return e;
}
```

```
Eintrag suchFunktion(string* s, Telefonbuch* t) { // kopiert nur 4 Byte
   ...
   return e;
}
```

```
Eintrag* suchFunktion(string* s, Telefonbuch* t) {
   // wenn man den Eintrag ändert, ändert sich auch der im Telefonbuch!
   ...
   return e;
}
```

Literatur

Weiterführende Literatur

- Nicolai Josuttis, “Objektorientiertes Programmieren in C++”, ISBN 3-8273-1771-1
- Bjarne Stroustrup, “The C++ Programming Language“, ISBN-13: 978-0201700732
- http://www.cppreference.com/wiki/
- http://www.cplusplus.com/reference/stl/
- http://www.cplusplus.com/ref/cstdio/

Vielen Dank!

Nächstes Mal:

C++,

Qt,

Debugging

http://www.stevenbrown.ca/blog/archives/225