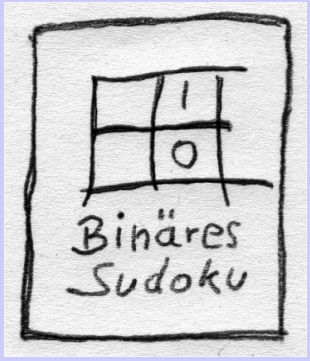
Jörn Fischer
j.fischer@hs-mannheim.de

Willkommen zur Vorlesung

C/C++ Programmierung (CPR)



Vorstellung

Zu meiner Person...

Überblick

Inhalt

1 - Organisatorisches

2 - Einführung

3.1 - Die Programmiersprache C

Datentypen, Präprozessor, Operatoren, Pointer, Strings, Dateizugriffe

3.2 - Die Programmiersprache C++

Referenzen, Static, Konstruktoren/ Destruktoren, Überladen von Funktionen und Operatoren, Vererbung Abstrakte Klassen

1 - Organisatorisches

1 - Organisatorisches

Kapitel 1 - Organisatorisches

- Wie läufts ab?
- Notengebung
- Fragen an mich

1 - Organisatorisches

Wie läufts ab?

- 3 x 5 Blöcke Vorlesung/ Übung
- Übungen macht jeder für sich

  (Nachbarn Fragen ist erlaubt, Source kopieren nicht!)
- Übungen auch zu Hause programmieren!
- Unterlagen auf meiner Homepage

1 - Organisatorisches

Notengebung

- Ich mache mir Notizen zu den Übungen eines jeden Studierenden
- Die Zusatzaufgabe: ein Mini-GO Spiel mit vereinfachten Regeln mittels ChatGPT und einem Monte-Carlo-Search-Algorithmus in C++ zu schreiben ergibt nochmal ebensoviele Punkte, wie man sie für die Übungsaufgaben erhält.

Aus den resultierenden Punkten wird die Note eines jeden Einzelnen berechnet...

1 - Organisatorisches

Fragen an mich...

- Fragen können Sie vor allem in der Vorlesungszeit an mich richten...
- Rückkopplung ist ausdrücklich erwünscht !!!!!

1 - Organisatorisches

Haben Sie Fragen?

2 - Einführung

2 - Einführung

Inhalt

- Warum C/C++
- Was hat das mit Medizin zu tun?

2 - Einführung

Warum C/C++?

- Es gibt bereits einen großen Teil C/C++ Code den man verstehen, warten und erweitern können sollte
- In C/C++ sind auch die meisten Betriebssysteme, Treiber und Embedded Systems programmiert
- C/C++ kann sehr (Laufzeit-)effizient programmiert werden
- Auch Grafikkarten werden vornehmlich in C/C++ programmiert

2 - Einführung

Was hat das mit Medizin zu tun?

- Existierende medizinische Software ist nicht selten in C/C++ geschrieben und Quellcode muss gepflegt, erweitert und gewartet werden.
- Die Prinzipien der Speicherverwaltung, das Arbeiten mit Zeigern und Referenzen funktioniert anders als in Java und hilft einen Blick über den Tellerrand zu bekommen.
- Selbst, wenn man heute viel mit Sprachen wie Python arbeitet sind effiziente Bibliotheken wie z.B. numpy auch in C/C++ geschrieben. So kann man sich selbst effiziente Erweiterungen schreiben, wenn es die noch nicht gibt.

3 - Die Programmiersprache C/C++

3 – Die Programmiersprache C/C++

Kapitel 3 – Die Programmiersprache C

- Literatur
- Unterschiede Java und C/C++
- Datentypen
- Operatoren
- Pointer (Wie und Warum)
- Speicher reservieren
- Typedefs, Structs und Unions

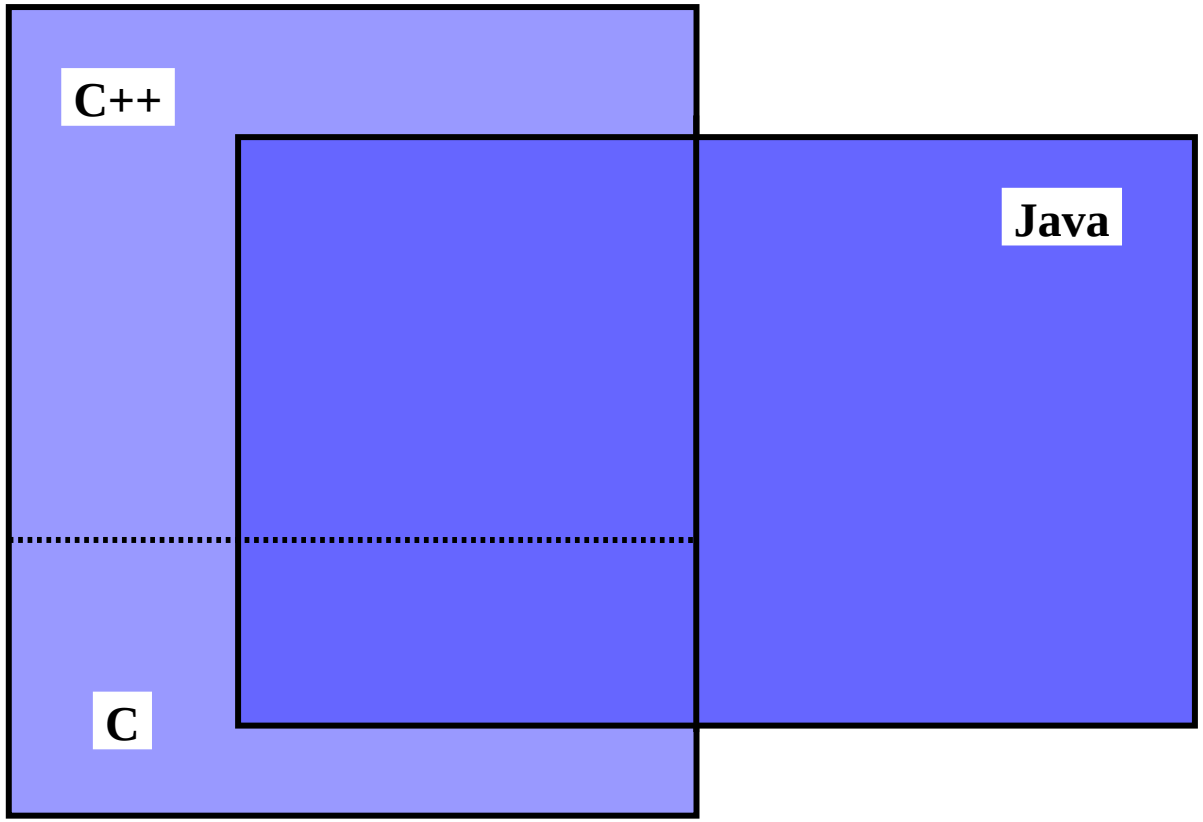
3 – Die Programmiersprache C/C++

Literatur

- B. Kernigan, D. Richie: Programmieren in C; Hanser Verlag 1990
- Bjarne Stroustrup: Die C++ Programmiersprache; Adison-Wesley, 4. Auflage, 2009
- Ulrich Breymann: Der C++-Programmierer; Carl Hanser Verlag, 2. Auflage, 2011
- Dietrich May: Grundkurs Software-Entwicklung mit C++;Springer, 2. Auflage, 2006
- de.wikibooks.org/wiki/C-Programmierung
- de.wikibooks.org/wiki/C++-Programmierung
- www.cplusplus.com
- www.cppreference.com
- www.cpp-tutor.de

3 – Die Programmiersprache C/C++

Unterschiede Java und C/C++



3 – Die Programmiersprache C/C++

Was fehlt in C/C++ (was es in Java gibt)

- In C gibt es keine Klassen und keine Objekte
- In C++ gibt es Klassen aber keine
  - Interfaces
  - Reflections
  - Garbage Collection
- Nur mit Zusatzbibliotheken gibt es
  - Grafikbibliotheken
  - GUI
  - Netzwerk

3 – Die Programmiersprache C/C++

Was fehlt in Java (was es in C++ gibt)

- Präprozessor
- Zeiger
- Effizientes Speichermanagement
- Operatorüberladung
- Mehrfachvererbung
- Namensräume
- Möglichkeiten der systemnahen Programmierung

3 – Die Programmiersprache C/C++

Einschränkungen in C:

- C ist rein prozedural (nicht objektorientiert, keine Vererbung, keine Templates...)
- Es gibt in C keine Referenzen, sondern nur Zeiger
- Lokale Variablen müssen in C immer am Anfang des Blocks definiert werden!
- Es gibt keine Klassen, sondern nur Structs (enthalten keine Funktionen)

3 – Die Programmiersprache C/C++

Welche Synonyme gibt es (Java → C/C++):

- Methoden → Funktionen
- Attribute → Variablen
- Generics → Templates
- abstrakte Methoden → virtuelle Methoden
- Interfaces → rein abstrakte Klassen (kommt dem am nächsten)

3 – Die Programmiersprache C/C++

Zum Einstieg ein Hello World Programm:

C

```
#include <stdio.h>
int main(int argc, char **argv){
     printf("Hello World!");
     return 0;
}
```

C++

```
#include <iostream>
class Hello{
   static void myMain(){
          std::cout << "Hello World!";
   }
};
int main(int argc, char**argv[]){
   Hello::myMain();
   return 0;
}
```

3 – Die Programmiersprache C/C++

- Die „printf“ Anweisung

```
#include <stdio.h>
int main(int argc, char** argv)
{
 printf("Ganze Zahl = %d \n",5);
 printf("Komma Zahl = %f \n",5.5);
 printf("Buchstabe  = %c \n",'A');
}
```

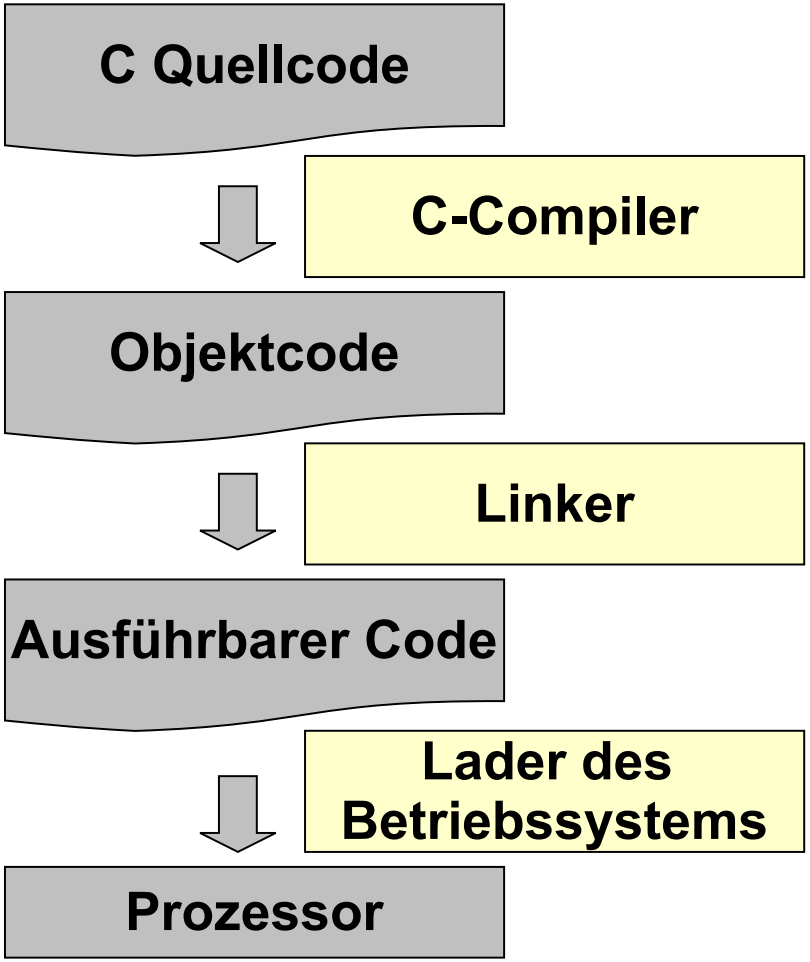
Fromatierungszeichen	Beschreibung
%d	ganzzahlig
%f	Fließkomma
%c	Character (Zeichen)
%s	String
\n	Zeilenvorschub

- Die „scanf“ Anweisung

```
 #include <stdio.h>
 int main(int argc, char** argv)
 {
  int zahl; float kommaZahl; char buchstabe; // declare variable
  scanf("%d",&zahl);                         // read integer
  scanf("%f",&kommaZahl);                    // read float
  scanf("%c",&buchstabe);                    // read char
 }
```

3 – Die Programmiersprache C/C++

Der Compiliervorgang:



- C-Programme liegen als Quellcode vor
- Ein Programm kann aus mehreren Quellcodedateien bestehen.
- Nur Files mit Änderung seit dem letzten Kompiliervorgang werden übersetzt
- Zunächst läuft ein Präprozessor über das Programm
- .c Dateien werden zu .o Object Files
- Dann wird mit Hilfe des Linkers aus den Object Files eine ausführbare Datei erstellt
- int main(int argc, char argv[][])wird beim Starten automatisch aufgerufen

3 – Die Programmiersprache C/C++

Der C/C++ Kompiliervorgang

- Kompilieren (einschließlich Präprozessor)

  gcc -c Hello.c

  (erzeugt Hello.o bzw. .obj)

- Linken:

  gcc -o Hello Hello.o [ggf. weitere .o Dateien]

  (erzeugt ausführbare Datei Hello bzw. Hello.exe)

- Ausführen

  Hello (unter Windows)

  ./Hello (unter Unix/Linux)

3 – Die Programmiersprache C/C++

MakeFiles zur Steuerung des Kompiliervorganges

- Im Makefile werden die Abhängigkeiten definiert

```
CC = /usr/bin/gcc
CFLAGS = -Wall -g -D_REENTRANT
LDFLAGS = -lm -lpthread

OBJ = datei1.o datei2.o datei3.o datei4.o

prog: $(OBJ)
	$(CC) $(CFLAGS) -o prog $(OBJ) $(LDFLAGS)

%.o: %.c
	$(CC) $(CFLAGS) -c $<
```

Mehr dazu: http://www.ijon.de/comp/tutorials/makefile.html

3 – Die Programmiersprache C/C++

Der Präprozessor

- Der Präprozessor ersetzt Quelltext im Programm

```
#include <stdio.h> // hier wird das Headerfile reinkopiert
#include "meinFile.h" // Header aus eigenem Pfad
#define PI 3.1415927  // definiert PI als 3.1415927
#define MAX(a,b) a>b?a:b // Macro Ersetzung
#if DEBUG_MODE // bedingte Übersetung
#else
#elif          // entspricht elseif
#endif
// mehrzeilige Präprozessoranw. mit \ am Ende der Zeile
```

3 – Die Programmiersprache C/C++

#include

Was macht der #include Befehl?

Was ist der Unterschied zwischen

```
#include "string.h"  // sucht zunächst im lokalen Ordner
#include <string.h>  // (c) sucht im standard Include Pfad
#include <string>    // (c++) sucht im standard Include Pfad
```

3 – Die Programmiersprache C/C++

#define

- In C werden Konstanten mit define deklariert.
- Makros ersparen durch Ersetzung im Quelltext den Funktionsaufruf.

```
#define TEXT "Hallo Welt"
#define TRUE 1
#define FALSE 0
#define DEBUG TRUE
#define MAX(a,b) a>b?a:b;
#define MAXX(a,b) if (a>b)                          \
                      printf("a ist groesser");     \
                  else                              \
                      printf("b ist groesser");
```

3 – Die Programmiersprache C/C++

#if #else #endif

Bedingte Compilierung:

```
#ifndef BERECHNUNG_H
#define BERECHNUNG_H
// obige Zeilen sogen dafür, dass das File
// nur einmal eingebunden wird, weil ab dann
// H_BERECHNUNG definiert ist.
...
#if DEBUG == TRUE
   printf("debugvalue=%d\n",debugvalue);
#else
   printf("\n");
#endif
...
#endif // BERECHNUNG_H
```

3 – Die Programmiersprache C/C++

#pragma

Was macht der Befehl #pragma?

- Der #pragma Befehl veranlasst den Präprozessor eine implementierungsspezifische Aktion (z.B. den Coprozessor mit zu benutzen)

```
// Beispiel: Parallelisierung mit Open-MP
#include <cstdio>
#include <omp.h>
int main(int, char*[]){
   #pragma omp parallel
   printf("Dies ist Thread %i von %i.\n",
       omp_get_thread_num(),
       omp_get_num_threads());
   return 0;
}
```

```
Ausgabe auf 2-Core-
Prozessor:

Dies ist Thread 1 von 2
Dies ist Thread 2 von 2
```

3 – Die Programmiersprache C/C++

Die Headerdatei

- In der Headerdatei werden Funktionen, Klassen, Typen deklariert
- Funktionen werden dort i.Allg. nur als Signatur hinterlegt
- So hat man pro Klasse immer ein .cpp und ein .h File
- Auch in c gibt es ein .c und ein.h File, welche gleich heißen
- Dem Compiler reicht das Headerfile mit den Signaturen um Funktionsaufrufe oder Objekt instanziierungen compilieren zu können.
- Libraries liegen dann nur noch als kompilierter Code und Headerfile vor.

3 – Die Programmiersprache C/C++

Die Headerdatei

Vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

class Vector
{
public:
    Vector();
    float getLength();
private:
    float x,y;
};
#endif // VECTOR_H
```

Vector.cpp

```
#include "Vector.h"
#include <math.h>

Vector::Vector()
{
  x=0; y=0;
}
float Vector::getLength()
{
  return sqrt(x*x+y*y);
}
```

3 – Die Programmiersprache C/C++

Der C/C++ Quelltext

- Der Compiler unterscheidet an der Endung des Files, ob das Programm als C Programm (.c Datei) oder als C++ Programm (.cpp Datei) compiliert werden soll
- Im Allgemeinen lassen sich .c Dateien auch als C++ Programm compilieren, indem man die Dateiendung auf .cpp ändert
- C++ Programme, die in einer .c Datei stehen lassen sich typischerweise nicht als .c Datei compilieren

3 – Die Programmiersprache C/C++

Schlüsselwörter:

asm	**auto**	**break**	**case**	*catch*
char	*class*	**const**	**continue**	**default**
delete	**do**	**double**	**else**	**enum**
except	**extern**	*finally*	**float**	**for**
friend	**goto**	**if**	*inline*	**int**
long	*new*	*operator*	*private*	*protected*
public	**register**	**return**	**short**	**signed**
sizeof	**static**	**struct**	**switch**	*template*
this	*throw*	*try*	**typedef**	**union**
unsigned	*virtual*	**void**	**volatile**	**while**

Zusätzlich in C++

3 – Die Programmiersprache C/C++

Elementare Datentypen

Datentyp	Wertebereich	Beschreibung
char	-128..127	ganzzahlig (1 ASC II Zeichen)
short	Bitbreite halb so groß oder gleich int meist: -32768..32767	ganzzahlig
int	Bitbreite des Targets mindestens 16 Bit	ganzzahlig
long int	Bitbreite doppelt so groß wie int	ganzzahlig
float	32 Bit, IEEE 754	Fließkommazahl
double	64 Bit	Fließkommazahl

3 – Die Programmiersprache C/C++

Datentypen

In C99 sind noch folgende Typen definiert in der <stdint.h>:

Datentyp	Wertebereich	Beschreibung
int8_t	-128..127	ganzzahlig
int16_t	-32768..32767	ganzzahlig
int32_t	-2147483648..2147483647	Ganzzahlig

- Zusätzlich gibt es die include Files: <limits.h> und <float.h> in denen die Min und Maxwerte der Datentypen stehen.
- Steht ein unsigned vor dem Datentyp, so beschreibt der neue Datentyp eine positive Zahl, dessen Wertebereich sich in den positiven Bereich verschiebt.

3 – Die Programmiersprache C/C++

Operatoren (Auswertreihenfolge)

Klammern	() []	13
Negation Inkrement Dekrement	- ! ~ ++ --	12
Arithmetische Operatoren	* / %	11
	+ -	10
Shift Operatoren	<< >>	9
Vergleichsoperatoren	> >= < <=	8
	== !=	7
Bitweise Operatoren	&	6
	^	5
	|	4
Logische Operatoren	&&	3
	||	2
Zuweisungsoperator	= += -= *= /= %= >>= <<= &= ^= |=	1

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Operator	Beschreibung
|	bitweise oder (OR)
&	bitweise und (AND)
!	bitweise nicht (NOT)
^	Ausschliessendes oder (XOR)
<<	links Verschiebung (shift left)
>>	rechts Verschiebung (shift right)

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Bitweise ODER (OR):

0	0	0
0	1	1
1	0	1
1	1	1

Beispiel:

$$\begin{array}{ll} & 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} = 553_{dez} \\ \mid & 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} = 1188_{dez} \\ = & 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1_{bin} = 1709_{dez} \end{array}$$

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Bitweise UND (AND):

0	0	0
0	1	0
1	0	0
1	1	1

Beispiel:

$$\begin{array}{ll} & 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} = 553_{dez} \\ \& & 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} = 1188_{dez} \\ = & 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0_{bin} = 32_{dez} \end{array}$$

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Bitweise NICHT (NOT):

0	1
1	0

Beispiel:

~ 0 1 0 0 0 1 0 1 0 0 1 $_{bin}$ = 553_{dez}

= 1 0 1 1 1 0 1 0 1 1 0 $_{bin}$ = 1494_{dez}

Bitbreite ist entscheidend für das Ergebnis

➔ Wird hauptsächlich für true == !false verwendet

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Bitweise ausschließendes ODER (XOR):

0	0	0
0	1	1
1	0	1
1	1	0

Beispiel:

$$\begin{array}{ll} & 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} = 553_{dez} \\ \hat{} & 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} = 1188_{dez} \\ = & 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1_{bin} = 1677_{dez} \end{array}$$

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Links verschieben (SHIFT LEFT):

Nach links verschieben um x Stellen entspricht der Multiplikaton mit 2^x

Beispiel:

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} << 1 = 553_{dez} * 2$

$0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0_{bin} = 1106_{dez}$

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin} << 2 = 553_{dez} * 4$

$1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 0_{bin} = 2212_{dez}$

3 – Die Programmiersprache C/C++

Bitweise Operatoren

Rechts verschieben (SHIFT RIGHT):

Nach rechts verschieben um x Stellen entspricht der Integer-Division durch 2^x

Beispiel:

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin}$ >> 1 = 553_{dez} / 2

$0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0_{bin}$ = 276_{dez}

$0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1_{bin}$ >> 2 = 553_{dez} / 4

$0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0_{bin}$ = 138_{dez}

3 – Die Programmiersprache C/C++

Wie kann man Bits extrahieren?

Gegeben sei eine vorzeichenlose Integerzahl $0x7A_{hex}$.

Wie kann ich die Zahl extrahieren, die zwischen Bit 4 und Bit 7 steht:

Bit	7	6	5	4	3	2	1	0
Wert 0x7A=	0	1	1	1	1	0	1	0
&	1	1	1	1	0	0	0	0
=	0	1	1	1	0	0	0	0
>>4 =	0	0	0	0	0	1	1	1

Man schiebt die Zahl um 4 Bits nach rechts, damit sie rechtbündig ist und verundet sie mit der Zahl, die sich ergibt, wenn man alle zu extrahierenden Bits gleich eins setzt 0b1111: ergebnis=(0x7A >> 4) & 0b1111; // Ergebnis = 7

3 – Die Programmiersprache C/C++

Wie kann man Bits setzen?

Gegeben sei eine vorzeichenlose Integerzahl $0x5A_{hex}$.

Wie kann ich Bit 5 bis Bit 7 setzen?

Bit	7	6	5	4	3	2	1	0
Wert 0x5A =	0	1	0	1	1	0	1	0
| 0xE0	1	1	1	0	0	0	0	0
=	1	1	1	1	1	0	1	0

0x5A | 0xE0 = 0xFA

3 – Die Programmiersprache C/C++

Wie kann man Bits löschen?

Gegeben sei eine vorzeichenlose Integerzahl $0x5A_{hex}$.

Wie kann ich Bit 5 bis Bit 7 löschen?

Bit	7	6	5	4	3	2	1	0
Wert 0x5A =	0	1	0	1	1	0	1	0
& 0x1F	0	0	0	1	1	1	1	1
=	0	0	0	1	1	0	1	0

0x5A &0x1F = 0x1A

3 – Die Programmiersprache C/C++

Pointer

- Zeiger (englisch: pointer) sind Variablen, deren Inhalt eine Speicheradresse ist
- Sie sind auch von einem definierten Typ, wobei der Typ nichts mit dem Inhalt der Variablen zu tun hat, sondern nur mit deren Inkrementierung und Dekrementierung

z.B. wenn double auf einem 8 Bit Prozessor 4 Byte einnimmt

```
double *pointer1 = &variable;

char   *pointer2 = &variable;

pointer1++; // pointer1 wird um 8 inkrementiert

pointer2++; // pointer2 wird um 1 inkrementiert
```

3 – Die Programmiersprache C/C++

Pointer

- Da ein Pointer nach der Deklaration irgendwo in den Speicher zeigt birgt das die Gefahr, dass man über den Pointer irgendwo in den Speicher schreibt.
- Darum ist es wichtig den Zeiger zu initialisieren.
- Falls die Variable, auf die gezeigt werden soll noch nicht bekannt ist sollte der Pointer mit NULL initialisiert werden:

```
int *pZeiger;

pZeiger = NULL;
```

3 – Die Programmiersprache C/C++

Adressoperator

Mit dem Adressoperator & bekommt man die Adresse einer Variablen:

```
int a,b,*pZeiger;
a=5;
pZeiger=&a; // pZeiger zeigt auf die Adresse von a
*pZeiger=6; // schreibt in die Adresse die Zahl 6
            //(äquivalent zu a=6;)
pZeiger=&b; // pZeiger zeigt auf die Adresse von b
*pZeiger=2; // schreibt 2 in b (äquivalent zu b=2;)
```

3 – Die Programmiersprache C/C++

Pointer auf pointer

Es können auch Pointer auf Pointer definiert werden:

```
int a,*pZeiger,**ppZeiger;

pZeiger=&a;                 // zeigt auf Adresse von a

ppZeiger=&pZeiger;          // zeigt auf Adresse von p Zeiger a=5;

*pZeiger=6;                 // entspricht a=6;

**ppzeiger=7;               // entspricht a=7;
```

3 – Die Programmiersprache C/C++

Pointer (Operatorreihenfolge)

Wert an der Stelle, auf die pointer zeigt auslesen und pointer erhöhen:

```
wert=*pointer++
```

...das ist die gebräuchliche Notation

Wert an der Stelle, auf die pointer zeigt auslesen und diesen Wert erhöhen:

```
wert=++*pointer
```

…bitte so NICHT verwenden

…bitte nur einfache Ausdrücke verwenden...Negativbeispiel:

```
char variable[4] = {0,1,2,3};

char *pointer = &variable[0];

*pointer = *pointer++ + ++*pointer;
```

3 – Die Programmiersprache C/C++

Speicher reservieren

Statt Speicher mit z.B. *int varArray[100]* zu reservieren kann man auch die Funktion *malloc()* benutzen.

```
<datentyp> * pVarArray;

int groesse = sizeof(int)*numberOfElements;

pVarArray = malloc(groesse);

if (pVarArray){

   // Speicher wurde reserviert!

}
```

Da der Speicher nicht automatisch freigegeben wird, muss er mit free() wieder freigegeben werden. Sonst können Memory Leaks (Speicherlecks) entstehen.

```
free(pVarArray);
```

3 – Die Programmiersprache C/C++

Strings

Länge eines Strings abfragen mit der Funktion
strlen(<String>)

- liefert Anzahl der Zeichen im übergebenen String als int-Wert.

Beispiele:

```
char *name = "Jörn Fischer";
printf ("%d\n",strlen(name));
```

→ **12**

```
char *a = "a";
printf ("%d\n",strlen(a));
```

→ **1**

3 – Die Programmiersprache C/C++

Strings

String kopieren mit der Funktion
strcpy(<kopie>, <quelle>)

```
// Speicher für neuen String belegen
char *quelle = "Beispieltext";
char *kopie = malloc(strlen(quelle)+1);   // +1 Nullbyte
// String kopieren
strcpy(kopie,quelle);
```

3 – Die Programmiersprache C/C++

Strings

n Zeichen des Strings kopieren mit der Funktion
strncpy(<kopie>, <quelle>, <n>)

```
// reserve memory for the string

char *kopie = malloc(4);

// copy string

strncpy(kopie,quelle, 4);
```

3 – Die Programmiersprache C/C++

Strings

Zwei Strings aneinanderhängen (konkatenieren) mit der Funktion
strcat(<ziel>, <quelle>)

- Zielstring bekommt Quellstring hinten drangehängt

```
char *vorname = „Jörn";

char *nachname =„Fischer";

// Speicher für Zielstring belegen

char *name = malloc(20);       // Platz für 19 Zeichen plus
Nullbyte

strcpy(name,vorname);          // Vornamen kopieren

strcat(name, " ");             // Leerzeichen dazwischen

strcat(name, nachname);        // Nachnamen hinten dran hängen

printf("%s\n",name);           // Gibt „Jörn Fischer" aus
```

3 – Die Programmiersprache C/C++

Strings

n Zeichen des Quellstrings an Zielstring anhängen mit der Funktion
strncat(<ziel>, <quelle>, <n>)

- Zielstring bekommt n Zeichen des Quellstrings hinten drangehängt

```
char *vorname = „Jörn";

char *nachname =„Fischer";

// Speicher für Zielstring belegen

char *name = malloc(20);        // Platz für 19 Zeichen plus
Nullbyte

strcpy(name,vorname);           // Vornamen kopieren

strcat(name, " ");              // Leerzeichen dazwischen

strncat(name, nachname, 3);     // 3 Buchst. des Nachnamens anh.

printf("%s\n",name);            // Gibt „Jörn Fis" aus
```

3 – Die Programmiersprache C/C++

Strings

Inhalt zweier Strings vergleichen mit der Funktion
strcmp(<String1>, <String2>)

- liefert **0**, wenn beide Strings zeichenweise gleich
- liefert +1, wenn String1 größer String2 (lexikalisch nach String2)
- liefert -1 wenn String1 kleiner String2 (lexikalisch vor String2)

!strcmp(<String1>, <String2>)

- liefert **1**, wenn beide Strings zeichenweise gleich, sonst 0.

```
char *t1=„Jörn";
char *t2=„Jörn";
char *m = "Markus";
printf("!strcmp(%s, %s)=%d\n", t1, t2, !strcmp(t1,t2)); // returns 1
printf("!strcmp(%s, %s)=%d\n", t1, m, !strcmp(t1,m));   // returns 0
```

3 – Die Programmiersprache C/C++

Strings

Suchstring im Quellstring herausfinden
char *strstr(<quellString>, <suchString>)

- liefert den Quellstring ab der Stelle, an der der Suchstr. gefunden ist
- liefert NULL, wenn Suchstring im Quellstring nicht enthalten

Beispiel:

```
char *name = "Jörn Fischer";
char *fischer = strstr(name,"Fischer");
printf ("%s\n",fischer);      // Fischer
char *fischer2 = strstr(name,"fischer");
printf ("%s\n",fischer2);     // (null)
char *fischer3 = strstr(name,"Jörn");
printf ("%s\n",fischer3);     // Jörn Fischer
```

3 – Die Programmiersprache C/C++

Dateizugriffe (Fopen(), fprintf(), fclose())

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   int t;
   FILE *out;
   out = fopen("Filename.txt","wb");
   if (out != NULL){
      for (t=0;t<100;t++){
          fprintf(out, "Test\n");
      }

      fclose(out);//closes the File
   }
  return 0;
}
```

```
open("file",xxx);
xxx = w für schreiben
xxx = r für lesen(File existiert)
xxx = r+ für lesen + schreiben
xxx = w+ für lesen + schreiben
xxx = a+ für lesen und erweitern
xxx = wb für binäres schreiben
xxx = rb für binäres lesen

fprintf() schreibt das, was
bei printf() auf dem Bildschirm

landet ins File!
```

3 – Die Programmiersprache C/C++

Dateizugriffe (Fgetc())

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   char letter;
   FILE *in;
   in = fopen("Filename.txt","rb");
   if (in != NULL){
      while(!feof(in)){
         letter=fgetc(in);
         printf("%c",letter);
      }
      fclose(in);
   }
   return 0;
}
```

fgetc liest ein char aus dem File

3 – Die Programmiersprache C/C++

Dateizugriffe (Fputc())

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   int t;
   FILE *out;
   out = fopen("Filename.txt","wb");
   if(out != NULL){
        for (t=0;t<100;t++){
            fputc('A',out);
        }
        fputc('\n',out);
        fclose(out);
   }
   return 0;
}
```

fputc schreibt ein char ins File

3 – Die Programmiersprache C/C++

Dateizugriffe (Fseek())

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   FILE *out;
   out = fopen("Filename.txt",“w");
   if(out != NULL){
      fputs("Dies ist ein Irrtum.",out);
      fseek(out, 13, SEEK_SET);
      fputs("Beispiel!" ,out);
   }
   fclose(in);
   return 0;
}
```

```
fseek(FILE*stream, long int
offset, int origin)
```

offset = Anzahl der bytes vom Ausgangspunkt

origin = SEEK_SET (Fileanfang)
= SEEK_CUR (aktuelle Pos)
= SEEK_END (Fileende)

3 – Die Programmiersprache C/C++

Typedefs

Mit der typedef Anweisung kann man einem Datentyp einen anderen Namen geben:

```
typedef int xxxInteger, xyzInteger;

xxxInteger var1;

xyzInteger var2;
```

3 – Die Programmiersprache C/C++

Strukturen

Strukturen werden in C dazu benutzt, Daten verschiedenen Typs, die zusammen gehören, in einer Datenstruktur zusammenzufassen.

```
struct schwein
{
 char name[20];
 float groesse;
 float gewicht;
}rudi,nele; // rudi und nele sind Instanzen von Schwein
```

3 – Die Programmiersprache C/C++

Strukturen

Eine Instanz eines structs benutzt man folgendermassen:

```
struct schwein berta; // berta is an instance of schwein
strcpy(berta.name,"Berta");
berta.groesse=0.95;
berta.gewicht=200;

struct schwein *pZeigerAufBerta;
pZeigerAufBerta=&berta;
pZeigerAufBerta->groesse += 0.1; // berta.groesse is enlarged
pZeigerAufBerta->gewicht  = 210; // berta.gewicht = 210 kg
```

3 – Die Programmiersprache C/C++

Unions

Im Gegensatz zu einer Struktur kann eine Union in einer Variablen unterschiedliche Typen speichern:

```
union eineVariable
{
 // Instance of eineVariable may be of type int or array
 // of float, but not both

 int intWert;
 float fliesskomaWert[10];
};
```

3 – Die Programmiersprache C/C++

...bis hier ging alles in C

… ab hier gibts C++

3 – Die Programmiersprache C/C++

Kompatibilität C/C++

- Ein C++ Compiler erzeugte früher im ersten Schritt C Code, der erst in weiteren Schritten in Assembler umgewandelt wurde.
- 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.

3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/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;
  }
}
```

3 – Die Programmiersprache C/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! Contents would be changed
```

3 – Die Programmiersprache C/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!
```

Neue Sprachmittel

3 – Die Programmiersprache C/C++

Neue Sprachmittel

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

3 – Die Programmiersprache C/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!
```

3 – Die Programmiersprache C/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
```

3 – Die Programmiersprache C/C++

Typenkonversion

Die Typenkonversion zwischen bool und int ist definiert:

```
bool wahriable=100; // äquivalent zu wahriable=true;

int ergebnis=false;// false->0, true->1

bool bIsOk=(wahriable!=ergebnis); // ergibt in diesem Fall true
```

3 – Die Programmiersprache C/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 Funktion(int a, int b=1, int c=2);

// Aufruf mit a=1,b=2,c=2

Funktion(1,2);
```

3 – Die Programmiersprache C/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"
```

3 – Die Programmiersprache C/C++

Besonders häufig werden Referenzen bei Funktionsübergaben verwendet:

```
void addiereUndSubtrahiere(int& ergebnis1, int& ergebnis2,int zahl1,
   int zahl2)
{
    ergebnis1=zahl1+zahl2;
    ergebnis2=zahl1-zahl2;
}
int main(void)
{
  int a=1,b=1;
  int erg1,erg2;
  addiereUndSubtrahiere(erg1,erg2,a,b); // equivalent: erg1=a+b;
                                        //             erg2=a-b;
}
```

3 – Die Programmiersprache C/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
```

3 – Die Programmiersprache C/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{...}

3 – Die Programmiersprache C/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
}
```

Klassen, Objekte und Methoden

3 – Die Programmiersprache C/C++

Klassen Objekte 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

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/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.
- 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.

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/C++

```
class CSchwein
{
 private: // kann von extern nur über setter und getter Funktionen zugegriffen werden
 int m_groesse; // Variablen oder Attribute
 int m_gewicht; // Variablen oder Attribute

 public: // mit public können die Funktionen auch von ausserhalb genutzt werden

 CSchwein(int groesse=0,int gewicht=0) // Konstruktor wird beim instantiieren eines Objekts
                                       // aufgerufen
 {                                     // wenn im Headerfile implementiert, dann automatisch
                                       // inline (wie ein Makro)
  m_groesse=groesse; // Variablen initialisieren
  m_gewicht=gewicht; // Variablen initialisieren
 }
 ~CSchwein(); // Destruktor wird aufgerufen, wenn die Instanz des objektes zerstört wird
 void setGroesse(int groesse);  // Deklaration der Methode,Memberfunktion oder Elementfunktion
 void setGewicht(int gewicht);  // Deklaration der Methode,Memberfunktion oder Elementfunktion
 int getGroesse();             // Deklaration der Methode,Memberfunktion oder Elementfunktion
 int getGewicht();             // Deklaration der Methode,Memberfunktion oder Elementfunktion
};
```

3 – Die Programmiersprache C/C++

```
CSchwein::~CSchwein() // Destruktor wird aufgerufen, wenn die Instanz des Objektes zerstört wird
{
// typischerweise stehen hier die delete Aufrufe
}
void CSchwein::setGroesse(int groesse)  // Methode,Memberfunktion oder Elementfunktion
{
  m_groesse=groesse;
}
void CSchwein::setGewicht(int gewicht)  // Methode,Memberfunktion oder Elementfunktion
{
  m_gewicht=gewicht;
}
int CSchwein::getGroesse()              // Methode,Memberfunktion oder Elementfunktion
{
  return m_groesse;
}
int CSchwein::getGewicht()              // Methode,Memberfunktion oder Elementfunktion
{
  return m_gewicht;
}
```

3 – Die Programmiersprache C/C++

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

int main(void)
{
 CSchwein Berta; // Berta wird instantiiert und ist dann ein Objekt oder eine Instanz
                 // der Klasse CSchwein
 int Gewicht;

 Berta.setGewicht(200);// Berta ist 200kg schwer
 Gewicht=Berta.getGewicht();
}
```

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/C++

```
class Ckonto //im Header File
{
 float m_Kontostand;
 static float m_Zinssatz; // für alle Instanzen ist der Zinssatz
                          // gleich, da diese Variable nur einmal
                          // im Speicher existiert!
 public:
 static void setZinssatz(float zinssatz){
    m_Zinssatz=zinssatz;
 }
 static float getZinssatz(){
    return m_Zinssatz;
 }
}
int main(void){ // im cpp File
 CKonto::setZinssatz(0.3);
 printf("Zinssatz = %f\n",CKonto::getZinssatz());
}
```

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.

3 – Die Programmiersprache C/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!

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 benutzt
```

3 – Die Programmiersprache C/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: …

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

3 – Die Programmiersprache C/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.
}
```

3 – Die Programmiersprache C/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
```

Überladen von Funktionen und Operatoren

3 – Die Programmiersprache C/C++

Überladen von Funktionen und Operatoren

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

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/C++

Überladen von Funktionen

So kann auch der Konstruktor und der Kopierkonstruktor überladen werden.

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

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
```

3 – Die Programmiersprache C/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 `heinrich` 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`.

3 – Die Programmiersprache C/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;
 }
}
```

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/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.

Überladen von Operatoren

Nicht nur Funktionen können überladen werden, sondern auch Operatoren.

Die Syntax einer Funktion, die einen Operator überlädt lautet wie folgt:

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

3 – Die Programmiersprache C/C++

```
class vector{
  enum {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
}
```

3 – Die Programmiersprache C/C++

Überladen von Operatoren: Bedingungen

Nicht überladen werden können:

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

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/C++

Beispiel: Skalar mal Vektor, Operatorüberladung ausserhalb der Klasse

```
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;
  }
};// --- overloading the operator outside of the class ---
doubleVector operator*(int k, const doubleVector& vec){
  doubleVector t(0.0,0.0,0.0);
  for (int i=0;i<doubleVector::SIZE;i++){
     t.v[i] = k*vec.v[i];
  }
  return t;
}
int main(void) {
 doubleVector v(1.1, 2.2, 3.3);
 doubleVector v2 = 5*v; // the function operator*() is called
}
```

3 – Die Programmiersprache C/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;
}
```

3 – Die Programmiersprache C/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;
 }
```

Vererbung

3 – Die Programmiersprache C/C++

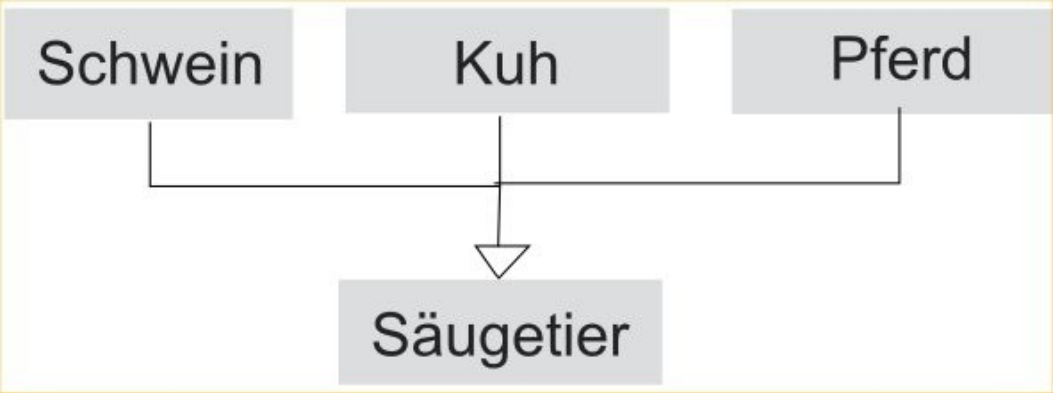
Vererbung in C++

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

3 – Die Programmiersprache C/C++

Grundlagen wie in Java

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.



3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/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();
}
```

3 – Die Programmiersprache C/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();
}
```

3 – Die Programmiersprache C/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.

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).

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/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!

3 – Die Programmiersprache C/C++

```
class CSaeugetier
{

 private:
 int groesse;
 int gewicht;

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

}
```

3 – Die Programmiersprache C/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!

3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/C++

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
```

Templates

3 – Die Programmiersprache C/C++

Templates

- Grundlagen
- Funktionstemplates
- Klassentemplates

3 – Die Programmiersprache C/C++

Grundlagen

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

3 – Die Programmiersprache C/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>
```

3 – Die Programmiersprache C/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);
```

3 – Die Programmiersprache C/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!

3 – Die Programmiersprache C/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;
}
```

3 – Die Programmiersprache C/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;
}
```

Streams

3 – Die Programmiersprache C/C++

Streams

- Grundlagen
- Datei schreiben
- Datei lesen

3 – Die Programmiersprache C/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:"; // Der Text in Anführungszeichen wird
   auf
                                     // dem Bildschirm ausgegeben
 cin >> zahl;      // hier wird eine Zahl von der Tastatur eingelesen,
                   // die mit der Eingabetaste bestätigt werden muss
 cout << "Sie haben folgende Zahl eingegeben:" << zahl << endl;
                   // endl ist Zeilenumbruch und forciert das ausgeben
                   // bzw. leeren des Ausgabebuffers
 return 0;
}
```

3 – Die Programmiersprache C/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();
}
```

3 – Die Programmiersprache C/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();
}
```

Strings

3 – Die Programmiersprache C/C++

Strings

- Vorteile gegenüber C-Strings
- Beispiel

3 – Die Programmiersprache C/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 makiert!
- 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 Stringklasse benutzen zu können muss sie zunächst includiert werden.

3 – Die Programmiersprache C/C++

Beispiel

```
#include <string>
using namespace std;
int main(void)
{
 string Nachname("Walter");
 string Vorname("Dieter");
 string Name;
 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;
}
```

Casting und Fehlerbehandlung

3 – Die Programmiersprache C/C++

Casting und Fehlerbehandlung

- Casting
- Fehlerbehandlung

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/C++

Fehlerbehandlung

Es gibt zweierlei Arten der Fehlerbehandlung in C++:

1. Mittels assertion

```
#include <assert.h>
...
assert(true);  // führt keine Fehlerbehandlung aus
assert(false); // führt eine Fehlerbehandlung aus
```

3 – Die Programmiersprache C/C++

2. Mittels Exceptions try-catch:

```
#include <iostream>
int i=0,j=0;
try
{
   i=i/j;
}
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;
}
```

Der neue C++ Standard

3 – Die Programmiersprache C/C++

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

3 – Die Programmiersprache C/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
```

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/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.join();
}
```

3 – Die Programmiersprache C/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.join();
}
```

3 – Die Programmiersprache C/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.join();

}
```

3 – Die Programmiersprache C/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.

3 – Die Programmiersprache C/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

3 – Die Programmiersprache C/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;

}
```

3 – Die Programmiersprache C/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