

C

Startpunkt

Was hiervon sind valide Deklarationen der Main Funktion? (ankreuzen)

int main(int argc, char *argv[]) { return 0; }	<input checked="" type="checkbox"/>
int main(){ return 0; }	<input checked="" type="checkbox"/>
float main(){ return 0; }	<input type="checkbox"/>
void main() { return 0; }	<input checked="" type="checkbox"/>
int main(int argc, char **argv) { return 0; }	<input checked="" type="checkbox"/>
int main(void) { return 0; }	<input checked="" type="checkbox"/>
char* main() { return 0; }	<input checked="" type="checkbox"/>

Makefile

Situation: Zwei Quelldateien main.c und summe.c sowie die Headerdatei summe.h, die main.c ruft die Funktion int make_sum(int a, int b).

Schreiben Sie ein makefile, welches die Dateien kompiliert und eine ausführbare Datei „main“ erstellt. Beachten Sie dabei, dass die Anweisungen die Abhängigkeiten der Quelldateien beachten (Bonus wenn beim Kompilieren der richtige C-Standard angegeben wird)

main.c

```
#include <stdio.h>
#include "summe.h"

int main(void)
{
    printf("Summe von 9 und 21 ist %d\n", make_sum(9,21));
}
```

summe.c

```
int make_sum(int a, int b)
{
    return a+b;
}
```

summe.h

```
int make_sum(int a, int b);
```

makefile

```
main: main.c summe.o summe.h
        gcc main.c -o main summe.o -std=c11

summe.o: summe.c
        gcc summe.c -c -std=c11
```

Structs und Unions

Sie sehen das Programm auf der rechten Seite, es ist laut C11 Standard valide und kompiliert ohne Fehler.

Wie sieht die Ausgabe des Programms aus?

s.a =	65
s.b =	66
u.a =	66
u.b =	B

Structs und Unions 2

Welche der drei unteren Aussagen trifft zu?

Die Struktur s ist...

kleiner als das Union u	<input type="checkbox"/>
gleich groß wie das Union u	<input type="checkbox"/>
größer als das Union u	<input checked="" type="checkbox"/>

```
#include <stdio.h>

struct teststruct {
    int a;
    char b;
};

union testunion {
    int a;
    char b;
};

int main()
{
    /* Auszug aus der ASCII Tabelle
     * 65 = A
     * 66 = B*/
    struct teststruct s;
    s.a = 65;
    s.b = 66;

    printf("s.a = %d\n", s.a);
    printf("s.b = %c\n", s.b);

    union testunion u;
    u.a = 65;
    u.b = 66;

    printf("u.a = %d\n", u.a);
    printf("u.b = %c\n", u.b);

    return 0;
}
```

```
#include <stdio.h>

struct point_s {
    int x;
    char y;
};

//hier kommt die Typdefinition hin:
typedef struct point_s point;

int main()
{
    struct point_s p1 = { 5, 7 };
    point p2 = { 3, 2 };

    printf("p1.x = %d, p1.y = %d\n", p1.x, p1.y);
    printf("p2.x = %d, p2.y = %d\n", p2.x, p2.y);

    return 0;
}
```

TypeDef

Ergänzen Sie den nebenstehenden Programmcode so, dass ein neuer Typ definiert wird mit Namen point, der auf die Struktur point_s zeigt, sodass der untere Code nach C11 Standard valide ist und kompiliert.

Zeigerarithmetik und Arrays

Was sind laut C11 Standard valide Deklaration für ein Array aus Integer Werten? Wie groß ist das Array(nichts angeben, falls es sich um eine nicht valide Deklaration handelt)

Deklaration	valide?	Maximale Anzahl der enthaltenen Elemente?
int arr1 [] = { 2 };	☒	1
int arr2[];	☐	
int arr3[1];	☒	1
int arr4[2] = { 6 };	☒	2

Zeigerarithmetik und Arrays 2

Ergänzen Sie den unteren Programmcode so, dass die Funktion void uppercase(?) einen Zeiger erwartet, dessen Wert überprüft wie im Programmcode angegeben und ihn ggf. auf einen neuen Wert setzt. Beim Aufruf der Funktion void uppercase(?) soll ein Zeiger auf den char c1 übergeben werden, nicht c1 selber.

```
#include <stdio.h>

void uppercase(char* c)
{
    char tmp = (*c); //Wert von Übergabeparameter der Variable tmp zuweisen

    //Prüfe ob der Wert im lowercase Bereich liegt
    if(tmp >= 'a' && tmp <= 'z')
    {
        tmp = tmp - 'a' + 'A';
        (*c) = tmp; //Weise dem Speicher, auf den der Zeiger zeigt, den Wert von tmp zu
    }
}

int main()
{
    char c1 = 'a';

    printf("c1 = '%c'\n", c1);

    printf("making c1 uppercase..\n");
    uppercase(&c1);
    printf("c1 = '%c'\n", c1);

    return 0;
}
```

Die Ausgabe des korrekt vervollständigten Programms würde dann so aussehen

```
c1 = 'a'
making c1 uppercase..
c1 = 'A'
```

Zeigerarithmetik und Arrays 3

Vervollständigen Sie die Funktion `void uppercase_string(char * str)` so, dass alle Elemente eines übergebenen nullterminierten Chararrays uppercase sind

```
#include <stdio.h>

int work(char* str);
void uppercase_string(char* str);

int main()
{
    char str1[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
    char str2[] = "world";

    work(str1);
    work(str2);

    return 0;
}

int work(char* str)
{
    printf("str = '%s'\n", str);
    printf("making str uppercase..\n");
    uppercase_string(str);

    printf("str = '%s'\n", str);
}

void uppercase_string(char* str)
{
    int i;
    for(i=0; *(str+i) != '\0'; i++)
    {
        if((*(str+i) >= 'a') && (*(str+i) <= 'z'))
        {
            *(str+i) = *(str+i) - 'a' + 'A';
        }
    }
}
```

In C gibt es von Haus aus nicht den Datentyp `String`, stattdessen wird dieser Datentyp über ein Chararray simuliert, dessen letztes Element ein `'\0'` ist, sodass man nicht die Länge des Arrays angeben muss. Dieser simulierte `String` lässt sich einmal über die ganz normale Arraydeklaration definieren, wobei das letzte Element ein `'\0'` ist (in der Aufgabe Variable `str1`). Der Nullterminator muss nicht am Ende des eigentlichen Arrays stehen, aber dies hat zur Folge, dass Funktionen wie z.B. `printf()` alle Elemente nach dem Nullterminator ignorieren – gleichbedeutend ist das Verhalten undefiniert wenn ein so simulierter `String` keinen Nullterminator enthält. Weiterhin kann man ein nullterminiertes Chararray auch durch die Deklaration mit doppelten Anführungszeichen erzeugen (in der Aufgabe Variable `str2`), was deutlich einfacher und natürlicher ist, aber komplett gleichbedeutend mit der Arraynotation ist.

Die Ausgabe des korrekt vervollständigten Programms würde dann so aussehen

```
str = 'hello'
making str uppercase..
str = 'HELLO'
str = 'world'
making str uppercase..
str = 'WORLD'
```

Funktionszeiger

Vervollständigen Sie das untere Programm so, dass die Funktion int work(int a, ?) einen int und einen Funktionszeiger, welcher als Parameter sowie als Rückgabe einen int hat, erwartet. Die Funktion int work(int a, ?) ruft den Funktionszeiger auf und gibt den Wert zurück.

Weiterhin soll die int main() Funktion zwei Funktionszeiger erstellen, mit denen nachher die Funktion int work(int a, ?) aufgerufen wird. Der erste Funktionszeiger fp1 soll auf die Funktion int add_two(int a) verweisen, der zweite fp2 auf die Funktion int add_multiply_by_three(int a).

```
#include <stdio.h>

int add_two(int a)
{
    return a+2;
}

int multiply_by_three(int a)
{
    return a*3;
}

int work(int a, int (*fp)(int))
{
    return fp(a);
}

int main()
{
    int (*fp1)(int) = add_two;
    int (*fp2)(int) = multiply_by_three;

    int a = 7;
    printf("work(a, fp1) = %d\n", work(a, fp1));
    printf("work(a, fp2) = %d\n", work(a, fp2));

    return 0;
}
```

Die Ausgabe des korrekt vervollständigten Programms würde dann so aussehen

```
work(a, fp1) = 9
work(a, fp2) = 21
```

Funktionszeiger 2

Was gibt das untere Programm bei Ausführung aus?

```
#include <stdio.h>

int add(int a, int b)
{
    return a+b;
}

int subtract(int a, int b)
{
    return a-b;
}

int multiply(int a, int b)
{
    return a*b;
}

int divide(int a, int b)
{
    return a/b;
}

int main()
{
    int (* fp1) (int, int) = add;
    int (* fp2) (int, int) = &subtract;
    int (* fp3) (int, int) = *multiply;
    int (* fp4) (int, int) = **divide;

    int a = 7;
    int b = 3;
    printf("fp1 = %d\n", (*fp1) (a, b)); → add
    printf("fp2 = %d\n", fp2(a, b)); → subtract
    printf("fp3 = %d\n", (*fp3) (a, b)); → multiply
    printf("fp4 = %d\n", fp4(a, b)); → divide

    printf("etwas Action reinbringen..\n");
    fp2 = fp1;
    fp1 = fp3;
    printf("fp1 = %d\n", (*fp1) (a, b)); → multiply
    printf("fp2 = %d\n", fp2(a, b)); → add
    printf("fp3 = %d\n", (*fp3) (a, b)); → multiply
    printf("fp4 = %d\n", fp4(a, b)); → divide

    return 0;
}
```

Ausgabe:

```
fp1 = 10
fp2 = 4
fp3 = 21
fp4 = 2
etwas Action reinbringen..
fp1 = 21
fp2 = 10
fp3 = 21
fp4 = 2
```

Datentypen und ihre Größen

Wie groß ist ein int laut C11 Standard? (nur eine Lösung ist korrekt)

8 bit	<input type="checkbox"/>
16 bit	<input type="checkbox"/>
32 bit	<input type="checkbox"/>
64 bit	<input type="checkbox"/>
Bitte ein Bit	<input type="checkbox"/>
128 bit	<input type="checkbox"/>
abhängig vom Betriebssystem	<input checked="" type="checkbox"/>

Wie groß ist ein char laut C11 Standard? (nur eine Lösung ist korrekt)

genau 8 bit	<input type="checkbox"/>
mindestens 8 bit	<input checked="" type="checkbox"/>
genau 16 bit	<input type="checkbox"/>
mindestens 16 bit	<input type="checkbox"/>

Datentypen und ihre Größen 2

Wir haben folgenden Programmcode

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

int main()
{
    int arr1[5] = { 1, 2, 3 };
    int arr2[] = { 1, 2, 3 };
    int* arr3 = (int*) malloc(sizeof(int) * 5);

    printf("sizeof(int) = %d\n", sizeof(int)); → x
    printf("sizeof(int*) = %d\n", sizeof(int*)); → y
    printf("sizeof(arr1) = %d\n", sizeof(arr1)); → x . 5 , da zur Compilezeit bekannt
    printf("sizeof(arr2) = %d\n", sizeof(arr2)); → x . 3
    printf("sizeof(arr3) = %d\n", sizeof(arr3)); → y

    return 0;
}
```

Wie sieht die Ausgabe aus? (kreise die richtige Ausgabe ein)

(INFO: die Rückgabe von sizeof(int) und sizeof(int*) wurde mit einer Variable realisiert, da der Rückgabewert abhängig vom System ist)

a)

```
sizeof(int) = x
sizeof(int*) = y
sizeof(arr1) = x*5
sizeof(arr2) = x*3
sizeof(arr3) = x*5
```

b)

```
sizeof(int) = x
sizeof(int*) = y
sizeof(arr1) = x*5
sizeof(arr2) = x*3
sizeof(arr3) = y
```

c)

```
sizeof(int) = x
sizeof(int*) = y
sizeof(arr1) = x*3
sizeof(arr2) = x*3
sizeof(arr3) = y
```

Abhängigkeitsbibliotheken

Sie haben die Quelldateien auto.c, fahrrad.c die zu einer Library fahrzeug.a hinzugefügt werden sollen via den Anweisungen im makefile und dem Konsolentool „ar“

auto.c

```
#include <stdio.h>
#include "fahrzeug.h"

Auto new_auto(char* bezeichnung, int ps, double preis)
{
    Auto a = { bezeichnung, ps, preis };
    return a;
}

void print_auto(Auto a)
{
    printf("bezeichnung = %s, ps = %d, preis = %.2f", a.bezeichnung, a.ps, a.preis);
}
```

fahrrad.c

```
#include <stdio.h>
#include "fahrzeug.h"

Fahrrad new_fahrrad(char* bezeichnung, double preis)
{
    Fahrrad f = { bezeichnung, preis };
    return f;
}

void print_fahrrad(Fahrrad f)
{
    printf("bezeichnung = %s, preis = %.2f", f.bezeichnung, f.preis);
}
```

fahrzeug.h

```
#ifndef FAHRZEUG_H_
#define FAHRZEUG_H_

typedef struct {
    char* bezeichnung;
    int ps;
    double preis;
} Auto;

Auto new_auto(char* bezeichnung, int ps, double preis);
void print_auto(Auto a);

typedef struct {
    char* bezeichnung;
    double preis;
} Fahrrad;

Fahrrad new_fahrrad(char* bezeichnung, double preis);
void print_fahrrad(Fahrrad f);

#endif
```

main.c

```
#include <stdio.h>
#include "fahrzeug.h"

int main()
{
    Auto a = new_auto("Ford Fiesta", 42, 1337.0);
    Fahrrad f = new_fahrrad("Hollandrad", 420.0);

    printf("Auto a: ");
    print_auto(a);
    printf("\n");

    printf("Fahrrad f: ");
    print_fahrrad(f);
    printf("\n");

    return 0;
}
```

makefile

```
main: main.c fahrzeug.a fahrzeug.h
    gcc main.c -o main fahrzeug.a -std=c11

fahrzeug.a: auto.o fahrrad.o
    ar fahrzeug.a auto.o fahrrad.o

auto.o: auto.c fahrzeug.h
    gcc auto.c -c -std=c11

fahrrad.o: fahrrad.c fahrzeug.h
    gcc fahrrad.c -c -std=c11
```

Bonus:

Warum sind in der Headerdatei fahrzeug.h diese Compileranweisungen enthalten?

```
#ifndef FAHRZEUG_H_
#define FAHRZEUG_H_

[...]

#endif
```

Dies nennt man Includeguards,
die davor schützen, dass ein und dieselben
Funktionen mehrfach importiert werden -
denn sobald sie einmal importiert sind,
wird der Code nicht erneut importiert

malloc & free

Vervollständigen Sie das Programm so, dass die Funktion `char* create_string(int length, char init)` ein Chararray dynamisch mittels der Library `stdlib.h` erzeugt mit `length` Elementen, die den Wert von `init` haben. Zudem soll das Chararray nullterminiert sein.

Weiterhin soll die Funktion `void delete_string(char* str)` ein dynamisch erzeugtes Chararray übergeben bekommen und dieses mittels der Library `stdlib.h` löschen

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

char* create_string(int length, char init)
{
    char* str = (char*) malloc(sizeof(char)*length+1);
    int i;
    for(i=0; ; < length ; i++)
    {
        str[i] = init;
    }
    str[length] = '\0'; // nullterminierung
    return str;
}

void delete_string(char* str)
{
    free(str);
}

int main()
{
    char* str1 = create_string(5, 'a');
    char* str2 = create_string(9, 'B');

    printf("str1 = %s\n", str1);
    printf("str2 = %s\n", str2);

    delete_string(str1);
    delete_string(str2);

    return 0;
}
```

Die Ausgabe des korrekt vervollständigten Programms würde dann so aussehen

```
str1 = aaaaa
str2 = BBBBBBBBB
```

Weitere Themen

- Compilierung
- makefile
- Typumwandlung
- logische Operatoren
- formatierte Ausgabe mit printf
- Speicherklassen: register, static, extern, static (global)

Diese Themen kamen ebenso in der Vorlesung vor, aber sind hier nicht oder nur halbherzig behandelt worden.

C++

Klassen und Vererbung

Erstellen Sie eine Klasse Auto mit den folgenden Eigenschaften:

- Sie hat zwei (private) Attribute
 - Einen std::string namens „name“
 - Einen int namens „baujahr“
- Einem öffentlichen Konstruktor der einen std::string und einen int übergeben bekommt und die oben genannten Attribute initialisiert. Diesen bitte in der Headerdatei Auto.h implementieren
- Einen öffentlichen Destruktor der vollständigkeit halber, der jedoch nichts macht, da die Klasse Auto nur Wertesobjekte enthält dessen Destruktor implizit aufgerufen wird. Diesen bitte in der Headerdatei Auto.h implementieren
- Eine öffentliche Funktion std::string getBeschreibung(). Diese bitte in der Headerdatei Auto.h angeben und in der Auto.cpp implementieren

Benutzen Sie die dabei vorgegebenen Textboxen in denen breite Teile implementiert sind.

makefile

```
main: main.cpp Auto.h Auto.o
    g++ main.cpp -o main Auto.o -std=c++11

Auto.o: Auto.cpp Auto.h
    g++ Auto.cpp -c -std=c++11

clean:
    rm Auto.o main
```

main.cpp

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

int main()
{
    Auto* a1 = new Auto("Trabant 601", 1976);
    Auto* a2 = new Auto("Porsche 911", 1963);

    std::cout << a1->getBeschreibung() << std::endl;
    std::cout << a2->getBeschreibung() << std::endl;

    return 0;
}
```

Die Ausgabe des korrekt vervollständigten Programms würde dann so aussehen

```
Auto: name='Trabant 601', baujahr=1976
Auto: name='Porsche 911', baujahr=1963
```

Auto.h

```
#ifndef AUTO_H
#define AUTO_H

#include <string>

class Auto {
private:
    std::string name;
    int baujahr;
public:
    Auto(std::string name, int baujahr) : name(name), baujahr(baujahr) {}

    std::string getBeschreibung();
};

#endif
```

Auto.cpp

```
#include "Auto.h"

// hier "std::string getBeschreibung()" implementieren
std::string Auto::getBeschreibung()
{
    std::string out = "Auto: name=" + name + ", baujahr=" + std::to_string(baujahr);
    return out;
}
```

Klassen und Vererbung 2, Polymorphie und virtuelle Funktionen
In der folgenden Aufgabe haben wir die zwei Klassen Dreirad und Panzer, die von der abstrakten Klasse Fahrzeug ableiten.

Wie lautet die Ausgabe(ganz unten)?

makefile

```
main: main.cpp Fahrzeug.h Fahrzeug.o Dreirad.h Dreirad.o Panzer.h Panzer.o
g++ main.cpp -o main Fahrzeug.o Dreirad.o Panzer.o -std=c++11

Dreirad.o: Dreirad.cpp Dreirad.h Fahrzeug.o
g++ Dreirad.cpp -c Fahrzeug.o -std=c++11

Panzer.o: Panzer.cpp Panzer.h Fahrzeug.o
g++ Panzer.cpp -c Fahrzeug.o -std=c++11

Fahrzeug.o: Fahrzeug.cpp Fahrzeug.h
g++ Fahrzeug.cpp -c -std=c++11

clean:
rm Fahrzeug.o Panzer.o Dreirad.o main
```

Fahrzeug.h

```
#ifndef FAHRZEUG_H
#define FAHRZEUG_H

#include <string>

class Fahrzeug
{
private:
    int baujahr;

public:
    Fahrzeug(int _baujahr) : baujahr(_baujahr) {};
    ~Fahrzeug() {};
    int inline getBaujahr() { return baujahr; };
    std::string virtual getBeschreibung() = 0;
    std::string getTyp();
};

#endif
```

Fahrzeug.cpp

```
#include "Fahrzeug.h"

std::string Fahrzeug::getTyp()
{
    return "Fahrzeug";
```

Panzer.h

```
#ifndef PANZER_H
#define PANZER_H

#include <string>
#include "Fahrzeug.h"

class Panzer : Fahrzeug
{
private:
    std::string name;

public:
    Panzer(std::string _name, int _baujahr) : name(_name), Fahrzeug(_baujahr) {}

    ~Panzer() {}

    std::string getBeschreibung();
    std::string getTyp();
};

#endif
```

Panzer.cpp

```
#include "Panzer.h"

std::string Panzer::getBeschreibung()
{
    std::string out = "Panzer: name=" + name + ", baujahr=" + std::to_string(getBaujahr());

    return out;
}

std::string Panzer::getTyp()
{
    return "Panzer";
}
```

Dreirad.h

```
#ifndef DREIRAD_H
#define DREIRAD_H

#include <string>
#include "Fahrzeug.h"

class Dreirad : Fahrzeug
{
private:
    double preis;

public:
    Dreirad(double _preis, int _baujahr) : preis(_preis), Fahrzeug(_baujahr) {};
    ~Dreirad() {};
    std::string getBeschreibung();
    std::string getTyp();
};

#endif
```

Dreirad.cpp

```
#include "Dreirad.h"

std::string Dreirad::getBeschreibung()
{
    std::string out = "Dreirad: preis=" + std::to_string(preis) + "$, baujahr=" + std::to_string(getBaujahr());
    return out;
}

std::string Dreirad::getTyp()
{
    return "Dreirad";
}
```

main.cpp

```
#include <iostream>
#include "Fahrzeug.h"
#include "Dreirad.h"
#include "Panzer.h"

int main()
{
    Panzer* p = new Panzer("Leopard 2", 1979);
    Dreirad* d = new Dreirad(59.99, 2011);

    std::cout << "Beschreibung: " << p->getBeschreibung() << ", Typ: " << p->getTyp() << std::endl;
    std::cout << "Beschreibung: " << d->getBeschreibung() << ", Typ: " << d->getTyp() << std::endl;

    Fahrzeug* f1 = (Fahrzeug*) p;
    Fahrzeug* f2 = (Fahrzeug*) d;

    std::cout << "Beschreibung: " << f1->getBeschreibung() << ", Typ: " << f1->getTyp() << std::endl;
    std::cout << "Beschreibung: " << f2->getBeschreibung() << ", Typ: " << f2->getTyp() << std::endl;

    return 0;
}
```

Ausgabe:

```
Beschreibung: Panzer: name='Leopard 2', baujahr=1979 , Typ: Panzer
Beschreibung: Dreirad: preis=59.99, baujahr=2011 , Typ: Dreirad
Beschreibung: Fahrzeug: name='Leopard 2', baujahr=1979 , Typ:Fahrzeug
Beschreibung: Fahrzeug: preis=59.99, baujahr=2011 , Typ:Fahrzeug
```

Bonus:

Was bedeutet, dass die Klasse Fahrzeug abstrakt ist?

Es können keine Objekte dieser Klasse erzeugt werden

Bonus2:

Warum ist die Klasse Fahrzeug abstrakt?

Da die Funktion getBeschreibung() nicht virtual ist.

Bonus3:

Was bedeutet das inline bei der Funktion int inline getBaujahr() in der Klasse Fahrzeug?

Vor- und Nachteile nennen sowie die Funktionsweise.

inline ist eine Compileranweisung, dass die Funktion getBaujahr nicht einen eigenen Frame bekommen soll, sondern überall wo der Funktionsaufruf passiert, wird der Inhalt der Funktion hineinkopiert.

Page 20 | 59

Vorteil: schneller

Nachteil: größerer Quellcode

Info: Eine Compileranweisung, NICHT verpflichtend

Klassen und Vererbung 3, Mehrfachvererbung

Wieso ist Mehrfachvererbung mit Vorsicht zu benutzen?

Bei Mehrfachdefinition von Funktionen kann es zu Uneindeutigkeiten kommen, wobei der Compiler nicht feststellen kann, welche Funktion aufgerufen werden soll.

Wieso kann der g++ Compiler die folgende Ergänzung zur Klassenhierarchie von „Klassen und Vererbung 2“ nicht kompilieren? (Beim Compilieren von Panzerdreirad.o meckert er)

Der Aufruf der Funktion getBaujahr() ist in beiden Superklassen definiert, womit der Compiler nicht weiß, welches er aufrufen soll

Panzerdreirad.h

```
#ifndef PANZERDREIRAD_H
#define PANZERDREIRAD_H

#include <string>
#include "Panzer.h"
#include "Dreirad.h"

class Panzerdreirad : Panzer, Dreirad
{
    private:
    public:
        Panzerdreirad(std::string _name,
double _preis, int _baujahr) : Panzer(_name,
_baujahr), Dreirad(_preis, _baujahr) {};
        ~Panzerdreirad() {};
        std::string getBeschreibung();
        std::string getTyp();
};

#endif
```

Panzerdreirad.cpp

```
#include "Panzerdreirad.h"

std::string Panzerdreirad::getBeschreibung()
{
    std::string out = "Panzerdreirad: name=" +
getName() + ", preis=" + std::to_string(getPreis()) + ",
baujahr=" + std::to_string(getBaujahr());

    return out;
}

std::string Panzerdreirad::getTyp()
{
    return "Panzerdreirad";
}
```

makefile

```
main: main.cpp Panzerdreirad.h Panzerdreirad.o
g++ main.cpp -o main Fahrzeug.o Dreirad.o Panzer.o -std=c++11

Panzerdreirad.o: Panzerdreirad.cpp Panzerdreirad.h Dreirad.o Dreirad.h
Panzer.o Panzer.h
g++ Panzerdreirad.cpp -c Dreirad.o Panzer.o -std=c++11

Dreirad.o: Dreirad.cpp Dreirad.h Fahrzeug.o
g++ Dreirad.cpp -c Fahrzeug.o -std=c++11

Panzer.o: Panzer.cpp Panzer.h Fahrzeug.o
g++ Panzer.cpp -c Fahrzeug.o -std=c++11

Fahrzeug.o: Fahrzeug.cpp Fahrzeug.h
g++ Fahrzeug.cpp -c -std=c++11

clean:
rm Fahrzeug.o Panzer.o Dreirad.o Panzerdreirad.o main
```

Namespaces, Streams, Operatorenüberladung

Sie haben eine Klasse Auto und sollen diese in den verschachtelten Namespace Fh::Pk2 packen (das heißt erst ein Namespace Fh, in dem ein zweiter Namespace Pk2 sich befindet, in dem sich schließlich die Klasse Auto befindet).

Weiterhin soll der Streamoperator << für eine Objektreferenz von der Klasse Auto überladen werden, die dafür benötigte Deklaration und Definition wurde bereits indiziert, wo sie hingehört – diese bitte auch in den verschachtelten Namespace mit einbringen. Es sollen dabei die Eigenschaften des Objektes ausgegeben werden mittels Nutzung der Funktionen getName() und getBaujahr().

Kleiner Tipp: Die Klasse ostream, der Stream cout sowie die Funktion endl sind im Namespace „std“ definiert, beachten Sie dies bei der Funktion int main().

main.cpp

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

using namespace std;
using namespace Fh::Pk2;

int main()
{
    Auto a { "Chevrolet Camaro", 2009 };
    cout << "a: " << a << endl;
    return 0;
}
```

makefile

```
main: main.cpp Auto.h Auto.o
        g++ main.cpp -o main Auto.o -std=c++11

Auto.o: Auto.cpp Auto.h
        g++ Auto.cpp -c -std=c++11

clean:
        rm Auto.o main
```

Auto.cpp

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

namespace Fh {
    namespace Pk2 {
        // hier den überladenen Streamoperator implementieren
        std::ostream& operator<< (std::ostream& stream, Auto a)
        {
            stream << "name=" << a.getName() << ", baujahr=" << a.getBaujahr();
            return stream;
        }
    }
}
```

Auto.h

```
#ifndef AUTO_H
#define AUTO_H

#include <string>

namespace Fl {
    namespace Pk2 {
        class Auto
        {
            private:
                string name;
                int baujahr;

            public:
                Auto(string _name, int _baujahr)
                    : name(_name), baujahr(_baujahr)
                {
                    // hier gibt es nichts zu tun
                }

                ~Auto()
                {
                    // Auto hat keine dynamisch erzeugten Objekte
                }

                string inline getName() const { return name; }

                int inline getBaujahr() const { return baujahr; }
            };

            // hier den Streamoperator << überladen
            std::ostream& operator<<(std::ostream& stream, Auto& a);
        }
    }
}
```

Die Ausgabe des korrekt vervollständigten Programms sieht etwa so aus:

```
a: name='Chevrolet Camaro', baujahr=2009
```

Bonus:

In welchem Teil vom Speicher befindet sich das Objekt a zum Aufruf von der Ausgabe via cout?

Stack	■
statischen Speicher	□
Kornspeicher	□
Haldenspeicher(Heap)	□

Exceptions

Was kann man alles bei C++ mit dem Ausdruck throw werfen?

Typ	valide?
Subklassen der Klasse std::exception	✗
Subklassen der Klasse std::error	✗
Integer	✗
Strings	✗
Objekte jeder Klasse	✗

Ergänzen Sie die untere Funktion double divide(double a, double b) so, dass sie eine Ausnahme wirft, falls b dem Wert 0 entspricht.

Weiterhin **ergänzen** Sie die Funktion int main(), sodass der von Ihnen geworfene Typ gefangen wird und in der Konsole eine Info darüber ausgegeben wird.

main.cpp

```
#include <iostream>

using namespace std;

double divide(double a, double b)
{
    if(b == 0)
        throw runtime_error("Division durch Null");
    return a / b;
}

double test(double a, double b)
{
    cout << "divide(" << a << ", " << b << ") = " << divide(a, b) << endl;
}

int main()
{
    try {
        double x = 0.0;
        double y = 5.0;
        double z = -3;

        test(x, y);
        test(x, z);
        test(y, x);
        test(y, z);
        test(z, x);
        test(z, y);

        catch(exception e) {
            cout << "Es ist ein Fehler aufgetreten" << endl;
        }
    }
    return 0;
}
```

Die Ausgabe des korrekt vervollständigten Programms sollte etwa so aussehen:

divide(0, 5) = 0
divide(0, -3) = -0
Es ist ein Fehler aufgetreten

Exceptions 2, Friend Funktionen

Sie haben den folgenden Quellcode vorliegen in der Datei main.cpp (in zwei Teile gesplittet, da zu wenig platz) von einem unsauber arbeitenden Kollegen und wollen wissen, was der Code macht.

Es wird darin eine Exception InvalidArgumentException definiert sowie eine Klasse Box

Tipp: cout und cerr geben beide auf die Konsole aus und sind bei diesem Beispiel äquivalent zu behandeln

Wie lautet letztendlich die Ausgabe? (siehe unten)

Bonus:

Was ist das besondere von Friend Funktionen bei C++? (sieht man unten)

Sie können auf private Member von Klassen zugreifen

main.cpp – Teil1

```
#include <iostream>
#include <exception>

using namespace std;

class InvalidArgumentException : exception {
    const char* cause;

public:
    InvalidArgumentException(const char* _cause) : cause(_cause) {}

    const char* what() const throw() { return cause; }
};

class Box {
    double breite;

public:
    friend void printBreite(const Box& box);
    void setBreite( double b );
};

void Box::setBreite(double b) {
    if(b >= 0)
        breite = b;
    else
        throw InvalidArgumentException("breite muss eine natuerliche Zahl sein");
}

void printBreite(const Box& box) {
    cout << "printBreite > breite=" << box.breite << endl;
}
```

main.cpp – Teil 2

```
void testBox(Box* box, double breite) {
    try {
        if(!box) throw runtime_error("Nullpointer");
        cout << "testBox > Breite wird auf " << breite << " gesetzt" << endl;
        box->setBreite(breite);
        cout << "testBox > Breite wird ausgegeben" << endl;
        printBreite( (*box) );
    } catch(const InvalidArgumentException &iae) {
        cerr << "testBox > InvalidArgumentException: " << iae.what() << endl;
    } catch(const std::exception &e) {
        cerr << "testBox > Exception: " << e.what() << endl;
    }
}

int main() {
    try {
        Box box1;
        cout << "Teste box1" << endl;
        testBox( &box1, 10.0 );
        cout << endl;

        Box box2;
        cout << "Teste box2" << endl;
        testBox( &box2, -1.0 );
        cout << endl;

        Box* box3 = nullptr;
        cout << "Teste box3" << endl;
        testBox( box3, 10.0 );
        cout << endl;

        Box* box4 = new Box();
        cout << "Teste box4" << endl;
        throw 42;
        testBox( box4, 7.0 );
        cout << endl;
    } catch( ... ) {
        cerr << "Ein unbekannter Fehler ist aufgetreten." << endl;
    }

    return 0;
}
```

Wie lautet die Ausgabe?

Teste box1
testBox > Breite wird auf 10.0 gesetzt
testBox > Breite wird ausgegeben
printBreite > Breite = 10.0

Teste box2
testBox > Breite wird auf -1.0 gesetzt
testBox > InvalidArgumentException: Breite muss eine natürliche Zahl sein

Teste box3
testBox > Exception: Nullpointer

Teste box4
Ein unbekannter Fehler ist aufgetreten.

Templates, dynamische Speicherverwaltung

Sie haben die vorliegenden Quelldateien, die die Klassen LinkedList und LinkElement definieren und testen

Vervollständigen Sie die Klasse LinkedList und LinkElement so, dass die enthaltenen Werte **dynamisch erzeugt werden** und erstellen Sie für die beiden Klassen ebenso einen **Destruktor**. Die entsprechenden Stellen sind mit Kommentaren kenntlich gemacht worden.

Wie lautet die Ausgabe? (auch lösbar, wenn Sie nicht das Programm vervollständigen konnten)

Wieso muss die LinkedList.cpp in der Hauptdatei verlinkt sein und kann nicht als Objektdatei realisiert werden? **weil der Compiler die Templatefunktionen erst erstellt, wenn sie benötigt werden und daher kann keine Objektdatei der LinkedList erzeugt werden - weil erst in der main Methode bekannt ist, welche Typen T annehmen kann**

```
#ifndef LINKEDLIST_H
#define LINKEDLIST_H

#include "LinkElement.h"

namespace Fh
{
    namespace Pk2
    {
        template<class T>
        class LinkedList
        {
            private:
                LinkElement<T>* head;
                LinkElement<T>* tail;

            public:
                LinkedList()
                {
                    head = nullptr;
                    tail = nullptr;
                }

                ~LinkedList()
                {
                    // löschen Sie alle Elemente
                    while(remove(0));
                }

                void add(const T& o);
                const T* get(int pos) const;
                int remove(int pos);
        };
    }
}

#endif
```

Ausgabe

```
list.get(0) = 1
list.get(1) = 2
list.get(2) = 3
list.get(3) = nullptr

list.remove(1) = 1
list.get(0) = 1
list.get(1) = 3
list.get(2) = nullptr
list.get(3) = nullptr

list.remove(1) = 1
list.get(0) = 1
list.get(1) = nullptr
list.get(2) = nullptr
list.get(3) = nullptr

list.remove(0) = 1
list.get(0) = nullptr
list.get(1) = nullptr
list.get(2) = nullptr
list.get(3) = nullptr

list.remove(0) = 0
list.get(0) = nullptr
list.get(1) = nullptr
list.get(2) = nullptr
list.get(3) = nullptr
```

makefile

```
main: main.cpp LinkedList.cpp LinkedList.h LinkElement.h
      g++ main.cpp -o main -std=c++11

clean:
      rm main
```

LinkElement.h

```
#ifndef LINKELEMENT_H
#define LINKELEMENT_H

namespace Fh
{
    namespace Pk2
    {
        template<class T>
        class LinkElement
        {
            private:
                const T* object;
                LinkElement* next;

            public:
                LinkElement(const T* o, LinkElement* n)
                    : object(o), next(n)
                {}

                ~LinkElement()
                {
                    // löschen Sie das dynamisch erstellte object
                    delete object;
                }

                const T* getObject() const
                {
                    return object;
                }

                LinkElement* getNext() const {
                    return next;
                }

                void setNext(LinkElement* n) {
                    next = n;
                }
        };
    }
}
```

LinkedList.cpp

```
#include "LinkedList.h"

using namespace Fh::Pk2;

template<class T>
void LinkedList<T>::add(const T& o) {
    // Erzeugen Sie dynamisch ein LinkElement.
    // Benutzen Sie den Kopierkonstruktor um den Wert von o zu kopieren
    // und erzeugen sie so ein dynamische erstelltes Element,
    // welches sie beim Erstellen des LinkElements benutzen

    LinkElement<T>* tmp = new LinkElement<T>(new T(o), nullptr); ;
    if(head == nullptr) {
        head = tmp;
        tail = head;
    } else {
        tail->setNext(tmp);
        tail = tmp;
    }
}

template<class T>
const T* LinkedList<T>::get(int pos) const {
    LinkElement<T>* tmp = head;
    int i = 0;
    while(i++ < pos) {
        if(tmp == nullptr)
            return nullptr;
        tmp = tmp->getNext();
    }

    if(tmp == nullptr)
        return nullptr;
    else
        return tmp->getObject();
}

template<class T>
int LinkedList<T>::remove(int pos) {
    LinkElement<T>* tmp = head;
    LinkElement<T>* prev = nullptr;
    int i = 0;
    while(i++ < pos) {
        if(tmp == nullptr)
            return 0;
        prev = tmp;
        tmp = tmp->getNext();
    }

    if(tmp == nullptr) {
        return 0;
    } else {
        if(prev == nullptr) {
            head = nullptr;
            tail = nullptr;
        } else {
            tail = prev;
            tail->setNext(tmp->getNext());
        }
        // Lösche tmp
        delete(tmp);
    }
    return 1;
}
```

Main.cpp

```
#include <iostream>
#include <string>
#include "LinkedList.cpp"

using namespace std;
using namespace Fh::Pk2;

std::ostream& operator<<(std::ostream& stream, const string* o) {
    if(o)
        stream << (*o);
    else
        stream << "nullptr";
    return stream;
}

template<class T>
int printList(LinkedList<T>& list, int size) {
    int i;
    for(i=0; i < size; i++) {
        cout << "list.get(" << i << ") = " << list.get(i) << endl;
    }
}

int main() {
    LinkedList<string> list;

    list.add("1");
    list.add("2");
    list.add("3");

    // drucke die Liste das erste mal aus
    printList(list, 4);
    cout << endl;

    // entferne das zweite Element und gebe Liste aus
    cout << "list.remove(1) = " << list.remove(1) << endl;
    printList(list, 4);
    cout << endl;

    // entferne das zweite Element und gebe Liste aus
    cout << "list.remove(1) = " << list.remove(1) << endl;
    printList(list, 4);
    cout << endl;

    // entferne das erste Element und gebe Liste aus
    cout << "list.remove(0) = " << list.remove(0) << endl;
    printList(list, 4);
    cout << endl;

    // entferne das erste Element und gebe Liste aus
    cout << "list.remove(0) = " << list.remove(0) << endl;
    printList(list, 4);
    cout << endl;

    return 0;
}
```

Weitere Themen

- formatierter Ausgabe mit cout
- default arguments
- Destruktorenreihenfolge
- delete[]
- detaillierte Referenzen
- mutable Attribute
- statische Elementfunktionen
- Typinformationen zur Laufzeit

Diese Themen kamen ebenso in der Vorlesung vor, aber sind hier nicht oder nur halbherzig behandelt worden.

C#

Klassen und Vererbung, Output und Input, Namespace

Erstellen Sie die Klasse Tanne so, dass sie eine Subklasse von Pflanze ist.

Die Klasse Tanne soll ein privates Attribut haben, mit der eine im Konstruktor angegebene Höhe gespeichert wird und beim Aufruf von getHoehe() ausgegeben wird.

Weiterhin ergänzen Sie die Funktion Main in der Klasse Program so, dass die untere Ausgabe erzeugt wird.

Zusätzlich fügen Sie bitte an das Ende der Funktion Main einen Funktionsaufruf ein, der eine Zeile einliest, damit das Programm beim Starten nicht sofort beendet.

Ausgabe

```
t.getHoehe() = 21,9  
p.getHoehe() = 21,9
```

Program.cs

```
using System;  
  
namespace Klassen_und_Verbung  
{  
    static class Program  
    {  
        static void Main(string[] args)  
        {  
            Tanne t = new Tanne(21.9);  
            // hier kommt die erste Ausgabe hin  
            Console.WriteLine("t.getHoehe() = {0}", t.getHoehe());  
            Pflanze p = t;  
            // hier kommt die zweite Ausgabe hin  
            Console.WriteLine("p.getHoehe() = {0}", p.getHoehe());  
            // hier kommt das Zeilen einlesen hin  
            Console.ReadLine();  
        }  
    }  
}
```

Pflanze.cs

```
namespace Klassen_und_Verbung  
{  
    abstract class Pflanze  
    {  
        public abstract double getHoehe();  
    }  
}
```

Tanne.cs

```
namespace Klassen- und Vererbung
{
    class Tanne : Pflanze
    {
        private double Hoche;
        public Tanne(double Hoche)
        {
            this.Hoche = Hoche;
        }
        public override double getHoche()
        {
            return Hoche;
        }
    }
}
```

Klassen und Vererbung2, Überschreiben von Funktionen

Wie lautet die Ausgabe bei dem unteren Programmcode?

Benutzen Sie die dafür vorgesehene Textbox

(wir nehmen an, dass ein double kommassepariert ausgegeben wird, bspw. „6,3“)

Bonus:

Welche Funktionen müssen von (nicht abstrakten) Klassen implementiert werden, wenn sie von der Klasse Pflanze ableiten? (kreuzen Sie an)

Funktion	muss abgeleitet werden?
GetHoehe()	<input checked="" type="checkbox"/>
GetName()	<input type="checkbox"/>
GetBeschreibung()	<input type="checkbox"/>

Ausgabe

a.GetHoehe() = 6,3, a.GetName() = Akazie, a.GetBeschreibung() = eine Akazie
p.GetHoehe() = 6,3, p.GetName() = Pflanze, p.GetBeschreibung() = eine Akazie

Program.cs

```
using System;

namespace Klassen_und_Vererbung2
{
    static class Program
    {
        static void Main(string[] args)
        {
            Akazie a = new Akazie(6.3);
            Console.WriteLine("a.GetHoehe() = {0}, a.GetName() = {1}, a.GetBeschreibung() = {2}",
                a.GetHoehe(), a.GetName(), a.GetBeschreibung());

            Pflanze p = a;
            Console.WriteLine("p.GetHoehe() = {0}, p.GetName() = {1}, p.GetBeschreibung() = {2}",
                p.GetHoehe(), p.GetName(), p.GetBeschreibung());

            Console.ReadLine();
        }
    }
}
```

Pflanze.cs

```
namespace Klassen_und_Vererbung2
{
    abstract class Pflanze
    {
        public abstract double GetHoehe();

        public string GetName()
        {
            return "Pflanze";
        }

        public virtual string GetBeschreibung()
        {
            return "eine Pflanze";
        }
    }
}
```

Akazie.cs

```
namespace Klassen_und_Vererbung2
{
    class Akazie : Pflanze
    {
        double Hoehe;

        public Akazie(double Hoehe)
        {
            this.Hoehe = Hoehe;
        }

        public override double GetHoehe()
        {
            return Hoehe;
        }

        new public string GetName()
        {
            return "Akazie";
        }

        public override string GetBeschreibung()
        {
            return "eine Akazie";
        }
    }
}
```

Exceptions und Operatorenüberladung

Sie haben die Klasse Vector und ein Hauptprogramm, was diese Klasse testet.

Erweitern Sie die Klasse Vector so, dass man zwei Objekte der Klasse Vector addieren kann via dem „+“ Operator. Sollte einer der Werte in den Vektoren Null oder negativ sein, soll eine Exception geworfen werden mit einer beliebigen Nachricht.

Weiterhin **erweitern** Sie im Hauptprogramm die Methode Vector testVector(Vector a, Vector b), sodass eine ggf. auftretende Exception gefangen und ausgegeben wird (die Nachricht wird im Klassenattribut „Message“ gespeichert).

Wie lautet die Ausgabe? Auch lösbar, wenn obere Aufgabe nicht bearbeitet (werden Objekte ausgegeben, die null sind wird nichts ausgegeben, Bsp: „Console.WriteLine(„test: {0}“, null);“ gibt „test: “ auf der Console aus)

Vector.cs

```
using System;

namespace Exceptions_und_Operatorenueberladung
{
    class Vector
    {
        private int x, y;

        public Vector(int x, int y)
        {
            this.x = x;
            this.y = y;
        }

        // hier Operator überladen
        public static Vector operator+(Vector a, Vector b)
        {
            if ((a.x <= 0) || (a.y <= 0) || (b.x <= 0) || (b.y <= 0))
                throw new Exception("Werte müssen positiv sein");
            else
                return new Vector(a.x+b.x, a.y+b.y);
        }

        public override string ToString()
        {
            return string.Format("x={0}, y={1}", x, y);
        }
    }
}
```

Program.cs

```
using System;
namespace Exceptions_und_Operatorenueberladung
{
    class Program
    {
        static Vector testVector(Vector a, Vector b)
        {
            Vector result = null;
            // hier die Addition testen und ggf. eine Exception fangen
            try
            {
                result = a + b;
            }
            catch (Exception e)
            {
                Console.WriteLine("Ein Fehler ist aufgetreten: {0}", e.Message);
            }
            return result;
        }

        static void Main(string[] args)
        {
            Vector v1 = new Vector(4, 2);
            Vector v2 = new Vector(1, 9);
            Vector v3 = new Vector(0, 9);

            Console.WriteLine("Addiere v1 und v2");
            Vector v4 = testVector(v1, v2);
            Console.WriteLine("Addiere v1 und v3");
            Vector v5 = testVector(v1, v3);
            Console.WriteLine("Addiere v2 und v3");
            Vector v6 = testVector(v2, v3);

            Console.WriteLine("v1: {0}", v1);
            Console.WriteLine("v2: {0}", v2);
            Console.WriteLine("v3: {0}", v3);
            Console.WriteLine("v4: {0}", v4);
            Console.WriteLine("v5: {0}", v5);
            Console.WriteLine("v6: {0}", v6);

            Console.ReadLine();
        }
    }
}
```

Ausgabe

Addiere v1 und v2
Addiere v1 und v3
Ein Fehler ist aufgetreten: Werte müssen positiv sein
Addiere v2 und v3
Ein Fehler ist aufgetreten: Werte müssen positiv sein
v1: x=4, y=2
v2: x=1, y=9
v3: x=0, y=9
v4: x=5, y=11
v5:
v6:

Delegates, anonyme Funktionen und Lambda-Ausdrücke

Definieren Sie ein delegate Ausgabe, welches nichts zurückgibt und als Übergabeparameter einen string hat.

In der Main Funktion initialisieren Sie bitte das delegate Ausgabe mit der Funktion GutenTag. Danach fügen Sie zu dem delegate Objekt erst eine anonyme Funktion hinzu und daraufhin einen Lambda-Ausdruck. Als letztes entfernen Sie die Funktion GutenTag von dem delegate Objekt. Die Ausgabe können Sie unten ansehen, bitte beachten Sie dies bei der Programmierung der anonymen Funktion und des Lambda-Ausdrucks.

Program.cs

<pre>using System; namespace Delegates_anonyme_Funktionen_Lambda { class Program { // definiere das Delegate Ausgabe mit der gleichen Signatur, // wie die Funktion GutenTag public static void GutenTag(string name) { Console.WriteLine("GutenTag > {0}", name); } static void Main(string[] args) { // initialisiere das delegate ausgabe mit der Funktion GutenTag Ausgabe ausgabe = new Ausgabe(GutenTag); // füge eine anonyme Funktion hinzu ausgabe += delegate (string name) { Console.WriteLine("anonyme Funktion > {0}", name); }; // füge einen Lambda-Audruck hinzu ausgabe += (string s) => Console.WriteLine("Lambda > {0}", s); ausgabe("Peter Zwegat"); // entferne die Funktion GutenTag ausgabe -= GutenTag; ausgabe("Niki Lauda"); Console.ReadLine(); } } }</pre>	Ausgabe
	GutenTag > Peter Zwegat anonyme Funktion > Peter Zwegat Lambda > Peter Zwegat anonyme Funktion > Niki Lauda Lambda > Niki Lauda

Events, partielle Klassen

Sie haben die partiell definierte Klasse Stundenplan gegeben, die mit Namen der Stunden und die Länge der Stunden initialisiert werden kann (siehe Main Funktion).

Man kann einen Stundenplan starten, dann wird simuliert, dass jede übergebene Stunde nacheinander eintritt. Dabei wollen wir, dass beim Anfang und Ende einer Stunde die Schüler informiert werden können und wir realisieren dies über ein Event.

Definieren Sie in der Klasse Stundenplan zwei Events StundeBegonnen und StundeBeendet vom delegate Typ Ausgabe (welches dieselbe Signatur besitzt wie die Funktion ZeigeStartAn der Klasse Bildschirm)

Lösen Sie die erzeugten Ereignisse an gegebener Stelle in der Klasse Stundenplan aus.

Erweitern Sie das Hauptprogramm so, dass die Funktion ZeigeStartAn des Bildschirms bildschirm1 vom Event StundeBegonnen des Stundenplans stundenplan1 ausgelöst wird. Das gleiche machen Sie mit der Funktion ZeigeEndeAn und dem Event StundeBeendet.

Die erzeugte Ausgabe können Sie unten sehen.

Program.cs

```
using System;
using System.Threading;

namespace Events
{
    class Program
    {
        static void Main(string[] args)
        {
            Tuple<string, int>[] stunden1 = new Tuple<string, int>[]
            {
                new Tuple<string, int>("Sport", 3),
                new Tuple<string, int>("Pause", 1),
                new Tuple<string, int>("Deutsch", 3),
                new Tuple<string, int>("Große Pause", 2),
                new Tuple<string, int>("Mathe", 3)
            };
            Stundenplan plan1 = new Stundenplan(stunden1);

            Bildschirm bildschirm1 = new Bildschirm("Sekretariat");

            // Fügen Sie die Funktionen ZeigeStartAn und ZeigeEndeAn
            // als Eventlistener hinzu
            plan1.StundeBegonnen += bildschirm1.ZeigeStartAn;
            plan1.StundeBeendet += bildschirm1.ZeigeEndeAn;
            plan1.Start();

            while(plan1.Laufend())
            {
                Thread.Sleep(1000);
            }

            Console.ReadLine();
        }
    }
}
```

Ausgabe:

```
Bildschirm("Sekretariat"): 'Sport' hat begonnen
Bildschirm("Sekretariat"): 'Sport' wurde beendet
Bildschirm("Sekretariat"): 'Pause' hat begonnen
Bildschirm("Sekretariat"): 'Pause' wurde beendet
Bildschirm("Sekretariat"): 'Deutsch' hat begonnen
Bildschirm("Sekretariat"): 'Deutsch' wurde beendet
Bildschirm("Sekretariat"): 'Große Pause' hat begonnen
Bildschirm("Sekretariat"): 'Große Pause' wurde beendet
Bildschirm("Sekretariat"): 'Mathe' hat begonnen
Bildschirm("Sekretariat"): 'Mathe' wurde beendet
```

Stundenplan1.cs

```
using System;
using System.Threading;

namespace Events
{
    partial class Stundenplan
    {
        private Tuple<string, int>[] stunden;
        private bool running = false;

        // definieren Sie ein delegate Ausgabe mit der selben
        // Signatur wie die Funktionen ZeigeStartAn/ZeigeEndeAn
        // der Klasse Bildschirm
        public delegate void Ausgabe(string s);

        // definieren Sie das Ereignis StundeBegonnen vom Typ Ausgabe
        public event Ausgabe StundeBegonnen;
        // definieren Sie das Ereignis StundeBeendet vom Typ Ausgabe
        public event Ausgabe StundeBeendet;

        public Stundenplan(Tuple<string, int>[] stunden)
        {
            this.stunden = stunden;
        }

        public void Start()
        {
            if (!running)
            {
                running = true;

                Thread t = new Thread(Berechne);
                t.Start();
            }
        }

        public void Stop()
        {
            running = false;
        }

        public bool Laufend()
        {
            return running;
        }
    }
}
```

Stundenplan2.cs

```
using System.Threading;

namespace Events
{
    partial class Stundenplan
    {
        public void Berechne()
        {
            for (int i = 0; (running) && (i < stunden.Length); i++)
            {
                string name = stunden[i].Item1;
                int laenge = stunden[i].Item2;

                // lösen Sie ein StundeBegonnen Ereignis aus
                Stunde Begonnen(name);
                Thread.Sleep(laenge * 1000);
                // lösen Sie ein StundeBeendet Ereignis aus
                Stunde Beendet(name);
            }

            running = false;
        }
    }
}
```

Bildschirm.cs

```
using System;

namespace Events
{
    class Bildschirm
    {
        private string bezeichnung;

        public Bildschirm(string bezeichnung)
        {
            this.bezeichnung = bezeichnung;
        }

        public void ZeigeStartAn(String s)
        {
            Console.WriteLine("Bildschirm(\"{0}\"): '{1}' hat begonnen", bezeichnung, s);
        }

        public void ZeigeEndeAn(String s)
        {
            Console.WriteLine("Bildschirm(\"{0}\"): '{1}' wurde beendet", bezeichnung, s);
        }
    }
}
```

Weitere Themen

- is/typeof Operatoren
- Zeiger und sizeof und unsafe Codeblöcke
- mehrdimensionale arrays, rectangular oder jagged
- formatierte Ausgabe mit WriteLine
- unterschied Structs und Klassen
- Sichtbarkeiten(public, protected, internal, internal protected, private)
- Verdeckung von Klassenmitgliedern
- sealed Klassen und Methoden
- interfaces
- call-by-reference, Schlüsselwort ref
- (automatic) properties, Vermeidung von Redundanzen
- generische Datentypen, Constraints
- Indexer
- Observer Modell
- Typumwandlung, checked und unchecked

Diese Themen kamen ebenso in der Vorlesung vor, aber sind hier nicht oder nur halbherzig behandelt worden.

C, C++, C# und Java – Unterschiede und Besonderheiten

Speicherverwaltung

C

Zeichnen Sie zwei **Speicherabbilder** bei Ausführung des Programms main.c, einmal vor pos1 und eines vor pos2

Beschriften Sie die Speicherbereiche mit den korrekten Namen, auch Bereiche die ggf. nicht zum Einsatz kommen.

main.c

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

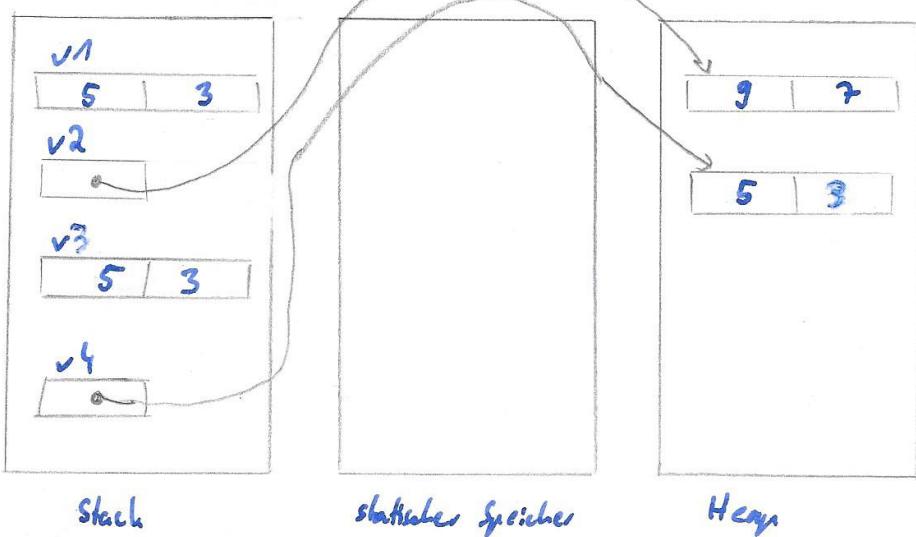
typedef struct
{
    int x, y;
} vector;

int main()
{
    vector v1 = { 5, 3 };
    vector* v2 = (vector*) malloc(sizeof(vector));
    v2->x = 9;
    v2->y = 7;

    vector v3 = v1;

    vector* v4 = (vector*) malloc(sizeof(vector));
    v4->x = v1.x;
    v4->y = v1.y;
    // pos1
    free(v2);
    // pos2
    return 0;
}
```

erstes Abbild:

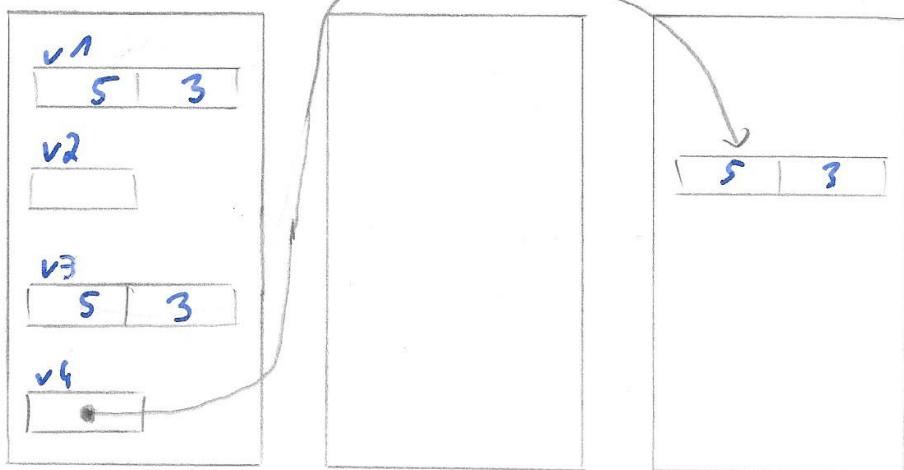


Stack

statischer Speicher

Heap

zweites Abbild:



C++

Zeichnen Sie zwei **Speicherabbilder** bei Ausführung des Programms main.c, einmal vor pos1 und eines vor pos2

Beschriften Sie die Speicherbereiche mit den korrekten Namen, auch Bereiche die ggf. nicht zum Einsatz kommen.

main.cpp

```
#include <iostream>

using namespace std;

class Vector
{
    int x, y;

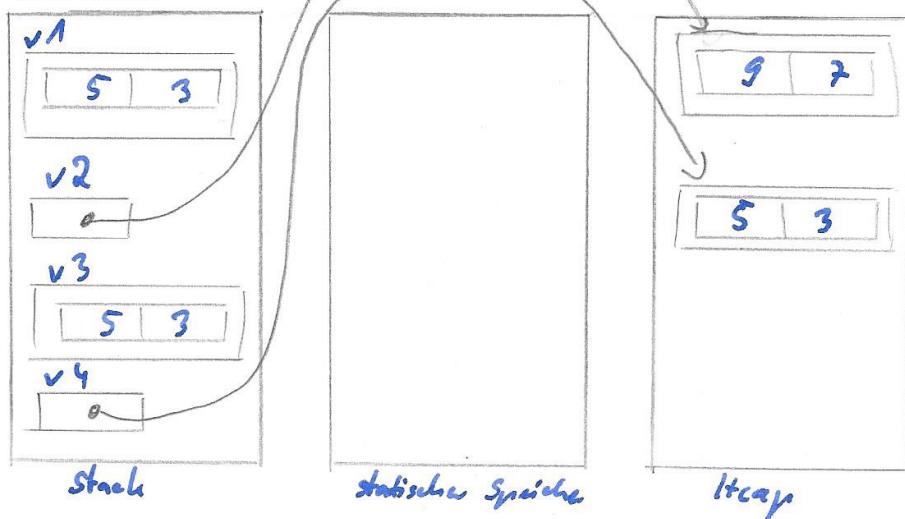
    public:
        Vector(int _x, int _y) : x(_x), y(_y) {};
        const int* getX() const { return &x; }
        const int* getY() const { return &y; }
};

int main()
{
    Vector v1(5, 3);
    Vector* v2 = new Vector(9, 7);

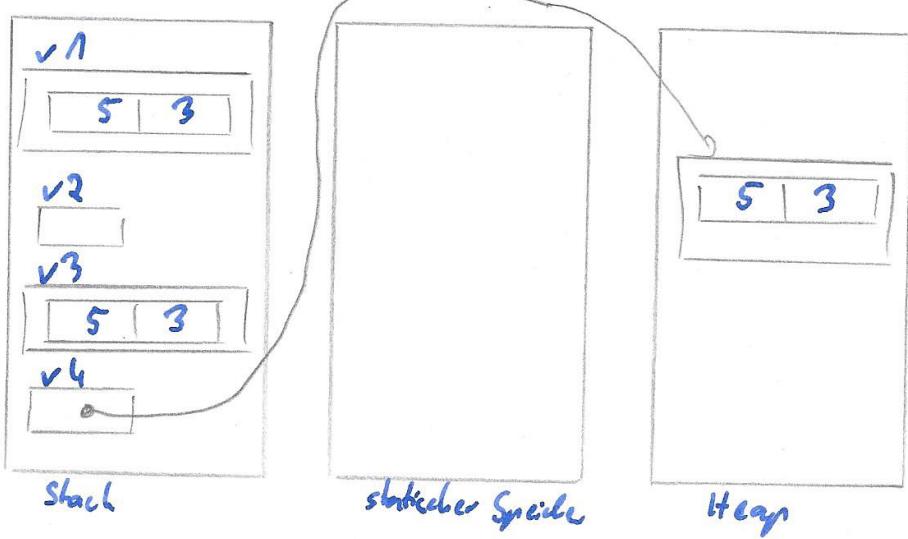
    Vector v3 = v1;

    Vector* v4 = new Vector(v1);
    // pos1
    delete(v2);
    // pos2
    return 0;
}
```

erstes Abbild:



zweites Abbild:



C#

Zeichnen Sie ein **Speicherabbild** bei Ausführung des Programms main.c, vor pos1

Beschriften Sie die Speicherbereiche mit den korrekten Namen, auch Bereiche die ggf. nicht zum Einsatz kommen.

Program.cs

```
using System;

namespace Speichermodell
{
    class Vector
    {
        int x;
        int y;

        public Vector(int x, int y)
        {
            this.x = x;
            this.y = y;
        }

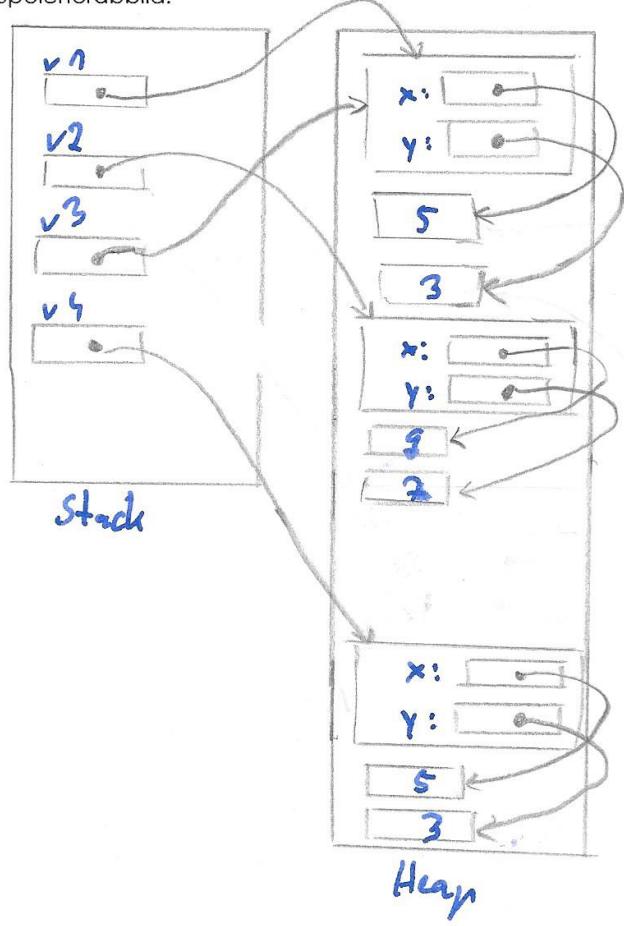
        public Vector(Vector v) : this(v.x, v.y)
        {
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Vector v1 = new Vector(5, 3);
            printVector(ref v1);
            Vector v2 = new Vector(9, 7);
            printVector(ref v2);

            Vector v3 = v1;
            printVector(ref v3);

            Vector v4 = new Vector(v1);
            printVector(ref v4);
            // pos1
            Console.ReadLine();
        }
    }
}
```

Speicherabbild:



Java

Zeichnen Sie ein **Speicherabbild** bei Ausführung des Programms main.c, vor pos1

Beschriften Sie die Speicherbereiche mit den korrekten Namen, auch Bereiche die ggf. nicht zum Einsatz kommen.

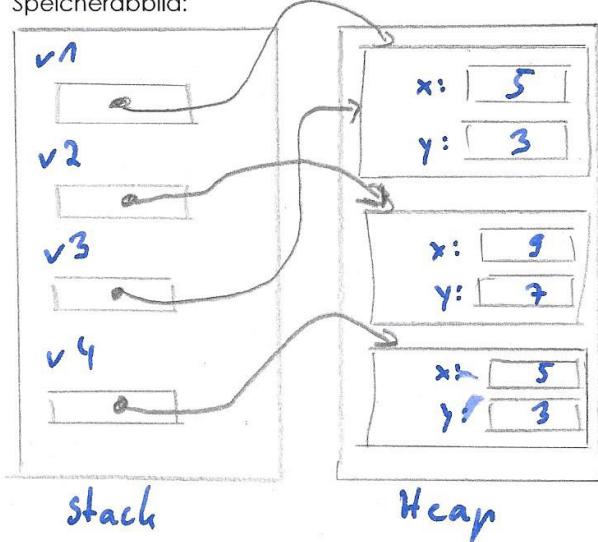
Vector.java

```
public class Vector {  
    public int x, y;  
  
    public Vector(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public Vector(Vector v) {  
        this(v.x, v.y);  
    }  
}
```

Main.java

```
public class Main {  
    public static void main(String[] args) {  
        Vector v1 = new Vector(5, 3);  
        Vector v2 = new Vector(9, 7);  
  
        Vector v3 = v1;  
  
        Vector v4 = new Vector(v1);  
  
        // pos1  
    }  
}
```

Speicherabbild:



Welche Sprache ist dies?

Programmausschnitt

```
public class Auto {  
    public int baujahr;  
  
    public Auto(int baujahr) {  
        this.baujahr = baujahr;  
    }  
  
    public Auto(Auto a) {  
        this(a.baujahr);  
    }  
}
```

Antwort(mehrere möglich): *C#, Java*

Welche Sprache ist dies? 2

Programmausschnitt

```
using Fh;  
  
namespace Test  
{  
    class Haus  
    {  
        int nummer;  
    }  
}
```

Antwort(mehrere möglich): *C#*

Welche Sprache ist dies? 3

Programmausschnitt

```
int i = 42;  
int* ptr = &i;
```

Antwort(mehrere möglich):

C, C++, C#(in einem unsafe Block)

Eigenschaften der Sprachen

Kreuzen Sie an, welche Sprachen welche Eigenschaften bzw. Konzepte besitzen

	C	C++	C#	Java
Klassen		X	X	X
Interfaces			X	X
Mehrfachvererbung		X		
plattformunabhängiger Code			X	X
Zeiger	X	X	X	
Typinformationen zur Laufzeit		X	X	X
Garbagecollector			X	X
finale Klassen (nicht ableitbar)			X	X
Exceptionhandling		X	X	X
globale Variablen / Funktionen	X	X		
structs	X	X	X	
unions	X	X		
Operatorenüberladung		X	X	
implizites Casting	X	X	X	X
explizites Casting	X	X	X	X
virtuelle Funktionen		X	X	theoretisch jede Funktion in Java
abstrakte Funktionen (echt virtuell)		X	X	
dynamische Bindung	X	X	X	