

Test 2 / Teil 3

Test2-Teil3.pptx

Instanzen und Vektor

Im Folgenden sind nicht immer die Lösungen angegeben, aber Sie können im Zweifelsfall selber den Compiler „fragen“.

Test 2 - Teil 3 pp. 1

Die verschiedenen Programmierparadigmen von C++

```
void pferd7() {  
    std::vector<int> vecI;  
    for (int i = 0; i < 11; ++i) {  
        vecI.push_back(i);  
    }  
}
```

Fortsetzung nächste Folie

Die verschiedenen Programmierparadigmen von C++

```
void pferd7() {  
    std::vector<int> vecI;  
    for (int i = 0; i < 11; ++i) {  
        vecI.push_back(i);  
    }  
    for (int i = 8; i > 5; --i) {  
        datei << vecI[i] << " ";  
    }  
}
```

Ausgabe

```
1. 8 7 6  
2. 7 6 5  
3. 8 7 6 5  
4. 7 6 5 4
```

Die verschiedenen Programmierparadigmen von C++

```
void pferd7() {  
    std::vector<int> vecI;  
    for (int i = 0; i < 11; ++i) {  
        vecI.push_back(i);  
    }  
    for (int i = 3; i < 6; i++) {  
        datei << vecI.at(i) << " ";  
    }  
}
```

Ausgabe?

1. 4 5 6
2. 3 4 5 6
3. 3 4
4. 3 4 5

Die verschiedenen Programmierparadigmen von C++

```
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Fortsetzung nächste Folie

Die verschiedenen Programmierparadigmen von C++

```
void pferd8() {  
    std::vector<Pferd*> vecPferd;  
    for (int i = 2; i < 4; ++i) {  
        Pferd* phlp = new Pferd(i);  
        vecPferd.push_back(phlp);  
    }  
    datei << "Ende\n";  
}
```

Wie viele Pferde werden erzeugt (Auf dem Stack und auf dem Heap)?

1. 0
2. 1
3. 2
4. 3
5. 4

```
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Die verschiedenen Programmierparadigmen von C++

```
void pferd8() {  
    std::vector<Pferd*> vecPferd;  
    for (int i = 2; i < 4; ++i) {  
        Pferd* phlp = new Pferd(i);  
        vecPferd.push_back(phlp);  
    }  
    datei << "Ende\n";  
}
```

```
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Wie viele Pferde werden nach Ausführung der Funktion zerstört, d.h. wie viel Destruktor-Aufrufe?

1. 0
2. 1
3. 2
4. 3
5. 4

Die verschiedenen Programmierparadigmen von C++

```
void pferd8() {  
    std::vector<Pferd*> vecPferd;  
    for (int i = 2; i < 4; ++i) {  
        Pferd* phlp = new Pferd(i);  
        vecPferd.push_back(phlp);  
    }  
    datei << "Ende\n";  
}
```

Ausgabe?

1. +P2 +P2 +P3 +P3 Ende
2. +P2 +P3 Ende -P3 - P2
3. +P2 +P3 Ende -P2 - P3
4. +P2 +P3 Ende

Die verschiedenen Programmierparadigmen von C++

```
void pferd9() {  
    // mit 5 Pferd initialisieren  
    std::vector<Pferd> vecPferd(5);  
    for (int i = 2; i < 4; ++i) {  
        Pferd gaul(i);  
        vecPferd[i] = gaul;  
        datei << " loop ";  
    }  
    datei << "Ende\n";  
}
```

```
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Die verschiedenen Programmierparadigmen von C++

```
void pferd9() {  
    // mit 5 Pferd initialisieren  
    std::vector<Pferd> vecPferd(5);  
}
```

Ausgabe?

1. keine
2. +P-1 +P-1 +P-1 +P-1 +P-1
3. +P-1 +P0 +P1 +P2 +P3

```
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Die verschiedenen Programmierparadigmen von C++

```
void pferd9() {  
    // mit 5 Pferd initialisieren  
    std::vector<Pferd> vecPferd(5);  
  
    for (int i = 2; i < 4; ++i) {  
        Pferd gaul(i);  
        vecPferd[i] = gaul;  
        datei << " loop ";  
    }  
}  
  
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Ausgabe nur in der Schleife

1. keine
2. +P2 +P2 loop -P2 +P3 +P3 loop -P3
3. +P2 loop -P2 +P3 loop -P3
4. +P2 loop +P3 loop -P3 -P2

Die verschiedenen Programmierparadigmen von C++

```
void pferd9() {  
    // mit 5 Pferd initialisieren  
    std::vector<Pferd> vecPferd(5);  
  
    for (int i = 2; i < 4; ++i) {  
        Pferd gaul(i);  
        vecPferd[i] = gaul;  
        datei << " loop ";  
    }  
}  
  
class Pferd {  
public:  
    Pferd(int n = -1) {  
        nr = n; datei << "+P:" << n << " "; }  
    ~Pferd() {datei << "-P:" << nr << " "};  
private:  
    int nr;  
};
```

Ausgabe **nur** in der Schleife

1. keine
2. +P2 +P2 loop -P2 +P3 +P3 loop -P3
3. +P2 loop -P2 +P3 loop -P3
4. +P2 loop +P3 loop -P3 -P2

„vecPferd[i] = gaul;“ ruft den

Standard – Zuweisungsoperator von Pferd auf.

Es wird dort zwar ein neues Pferd erzeugt, aber es erfolgt keine Ausgabe, denn der Standardkopierkonstruktor von Pferd gibt nichts aus.

Das Zerstören über vecPferd ist später.

Somit ist die Ausgabe „ +P2 loop -P2 +P3 loop -P3“.

Die verschiedenen Programmierparadigmen von C++

```
void pferd9() {  
    // mit 5 Pferd initialisieren  
    std::vector<Pferd> vecPferd(5);  
  
    for (int i = 2; i < 4; ++i) {  
        Pferd gaul(i);  
        vecPferd[i] = gaul;  
        datei << " loop ";  
    }  
    datei << "Ende\n";  
}
```

Ausgabe nach der Schleife?

1. Ende
2. Ende -P-1 -P-1 -P2 -P3 -P-1
3. Ende -P-1 -P-1 -P-1 -P-1 -P-1
4. Ende -P-1 -P-1 -P-1 -P-1 -P-1 -P2 -P3

Die verschiedenen Programmierparadigmen von C++

```
void pferd9() {  
    // mit 5 Pferd initialisieren  
    std::vector<Pferd> vecPferd(5);  
  
    for (int i = 2; i < 4; ++i) {  
        Pferd gaul(i);  
        vecPferd[i] = gaul;  
        datei << " loop ";  
    }  
    datei << "Ende\n";  
}
```

Ausgabe nach der Schleife?

1. Ende
2. Ende -P-1 -P-1 -P2 -P3 -P-1
3. Ende -P-1 -P-1 -P-1 -P-1 -P-1
4. Ende -P-1 -P-1 -P-1 -P-1 -P-1 -P2 -P3

vecPferd enthält 5 Instanzen von Pferd.

gaul ist bereits (zweimal) in der Schleife (erzeugt und) zerstört worden

Somit werden 5 Destruktoren aufgerufen

Die Ausgabe ist somit „Ende -P-1 -P-1 -P2 -P3 -P-1“.

Möglichkeiten sich die Speicherbelegung
per Programm zu veranschaulichen

Es ist immer nur ein Modell.

Die verschiedenen Programmierparadigmen von C++

```
void streichen(Tisch& t1) {
    t1.setBeine(18);
}

void beizen(Tisch t2) {
    t2.setBeine(25);
}

void tisch6() {
    Tisch eiche(7);
    Tisch linde(13);
    streichen(eiche);
    beizen(linde);
}
```

```
class Tisch {
public:
    Tisch(int n = -6) {
        bein = n; datei <<
            "+T:" << n << " ";
    }
    ~Tisch() { datei << "-T:"
        << bein << " "; };
    void setBeine(int n) {
        bein = n; }
private:
    int bein;
};
```

Die verschiedenen Programmierparadigmen von C++

```
void streichen(Tisch& t1) {
    t1.setBeine(18);
}
void beizen(Tisch t2) {
    t2.setBeine(25);
}
void tisch6() {
    cout << "Bytes von Tisch: " <<
        sizeof(Tisch) << "\n";
    Tisch eiche(7);
    Tisch linde(13);
    PrintAddr(eiche);
    streichen(eiche);
    PrintAddr(linde);
    beizen(linde);
    PrintAddrDiff(linde, eiche);
}
```

Die verschiedenen Programmierparadigmen von C++

```
void streichen(Tisch& t1) {
    t1.setBeine(18);
}
void beizen(Tisch t2) {
    t2.setBeine(25);
}
void tisch6() {
    cout << "Bytes von Tisch: " <<
        sizeof(Tisch) << "\n";
    Tisch eiche(7);
    Tisch linde(13);
    PrintAddr(eiche);
    streichen(eiche);
    PrintAddr(linde);
    beizen(linde);
    PrintAddrDiff(linde, eiche);
}
```

```
#define PrintAddr(var) \
    cout << #var " Adr: " << \
    reinterpret_cast<void*>(&var) \
    << "\n"
```

```
#define PrintAddrDiff(var1, var2) \
    cout << #var1 <<"- " << #var2 " Diff: " << \
    reinterpret_cast<int>(reinterpret_cast<void*>(&var2)) - \
    reinterpret_cast<int>(reinterpret_cast<void*>(&var1)) \
    << "\n"
```

Die verschiedenen Programmierparadigmen von C++

```
void streichen(Tisch& t1) {
    t1.setBeine(18); PrintAddr(t1);
}
void beizen(Tisch t2) {
    t2.setBeine(25); PrintAddr(t2);
}
void tisch6() {
    cout << "Bytes von Tisch: " <<
        sizeof(Tisch) << "\n";
    Tisch eiche(7);
    Tisch linde(13);
    PrintAddr(eiche);
    streichen(eiche);
    PrintAddr(linde);
    beizen(linde);
    PrintAddrDiff(linde, eiche);
}
```

Debug-Modus

```
--
Bytes von Tisch: 4
eiche Adr: 00DAF870
t1 Adr: 00DAF870
linde Adr: 00DAF864
t2 Adr: 00DAF78C
linde- eiche Diff: 12
```

Die verschiedenen Programmierparadigmen von C++

```
void nb1(NullByte& t1) {
    PrintAddr(t1);
}
void nb2(NullByte t2) {
    PrintAddr(t2);
}

void nullByteTest() {
    cout << "Bytes von NullByte: " <<
        sizeof(NullByte) << "\n";
    NullByte eiche(7);
    NullByte linde(13);
    PrintAddr(eiche);
    nb1(eiche);

    PrintAddr(linde);
    nb2(linde);
    PrintAddrDiff(linde, eiche);
}
```

```
// Klasse ohne Attribute
class NullByte {
public:
    NullByte(int n = -6) {
        datei << "+NB:" << n
            << " ";
    }
    ~NullByte() { datei << "-
        NB:"; };
};
```

Debug-Modus

```
Bytes von NullByte: 1
eiche Adr: 00DAF873
t1 Adr: 00DAF873
linde Adr: 00DAF867
t2 Adr: 00DAF78C
linde- eiche Diff: 12
```

Es ist nur ein Modell

Debug-Modus

```
Bytes von Tisch: 4  
eiche Adr: 00DAF870  
t1 Adr: 00DAF870  
linde Adr: 00DAF864  
t2 Adr: 00DAF78C  
linde- eiche Diff: 12
```

```
Bytes von NullByte: 1  
eiche Adr: 00DAF873  
t1 Adr: 00DAF873  
linde Adr: 00DAF867  
t2 Adr: 00DAF78C  
linde- eiche Diff: 12
```

Release-Modus

```
Bytes von Tisch: 4  
eiche Adr: 00CFF8A8  
t1 Adr: 00CFF8A8  
linde Adr: 00CFF8A4  
t2 Adr: 00CFF8A0  
linde- eiche Diff: 4
```

```
Bytes von NullByte: 1  
eiche Adr: 00CFF8CF  
t1 Adr: 00CFF8CF  
linde Adr: 00CFF8CE  
t2 Adr: 00CFF8C8  
linde- eiche Diff: 1
```

Vergrößerung von Tisch um zwei weitere int-Attribute

Release-Modus

Debug-Modus

```
Bytes von Tisch: 12  
eiche Adr: 0019FD54  
t1 Adr: 0019FD54  
linde Adr: 0019FD40  
t2 Adr: 0019FC60  
linde- eiche Diff: 20
```

```
. Bytes von NullByte: 1  
, eiche Adr: 0019FD5F  
t1 Adr: 0019FD5F  
▪ linde Adr: 0019FD53  
t2 Adr: 0019FC78  
linde- eiche Diff: 12
```

```
Bytes von Tisch: 12  
eiche Adr: 005FF9F8  
t1 Adr: 005FF9F8  
linde Adr: 005FFA10  
t2 Adr: 005FFA04  
linde- eiche Diff: -24
```