Klausurvorbereitung

Konstruktor / Destruktor

Klassen

- Was gehört zur Minimalen Standardschnittstelle
- Was zeichnet gute (Klassen-) Schnittstellen aus.

- Beispiel für CallByValue und CallByReference
- Test für die Funktionen

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
int   geh;
public:
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
             cout <<"-M " << geh;}
};
```

```
void f() {
          Mitarbeiter m1(4);
          Mitarbeiter m2(3);
}
```

Zuweisungsoperator und Kopierkonstruktur der Klasse angeben.

Mit _-_ im Folgenden immer Lösung gekennzeichnet

1. +M 4 + M 3 -M 4 – M 3
2. +M 4 + M 3
3. +M 4 + M 3 -M 3 – M 4
4. Keine Ausgabe

Ergebnis:
__ 1 _ 2 _-_ 3 __4

Kopierkonstruktor und Zuweisungsoperator von Mitarbeiter

```
class Mitarbeiter  {
int   geh;
public:
 Mitarbeiter(int g)     {geh = g;  cout << "+M "  << geh;}
~Mitarbeiter()   { cout <<"-M " << geh;}
 // Kopierkonstruktor
 Mitarbeiter(const Mitarbeiter& m2)     {geh = m2.geh;}
// Kopierkonstruktor andere Lösung
 Mitarbeiter(const Mitarbeiter& m2)  :geh(m2.geh) {}

// Zuweisungsoperator
 Mitarbeiter& operator=(const Mitarbeiter& m2)     {geh = m2.geh; return *this;}
// Alternativ
 Mitarbeiter& operator=(const Mitarbeiter& m2)     {
   if (this != *m2) {geh = m2.geh;}
   return *this;   }

};
```

Kopierkonstruktor und Zuweisungsoperator von Mitarbeiter

```
// Header-Datei: Mitarbeiter.h
class Mitarbeiter  {
int   geh;
public:
 Mitarbeiter(int g)     {geh = g;  cout << "+M "  << geh;}
~Mitarbeiter()   { cout <<"-M " << geh;}
Mitarbeiter(const Mitarbeiter& m2) ;
Mitarbeiter& operator=(const Mitarbeiter& m2);
};
```

```
// Source-Datei: Mitarbeiter.cxx
Mitarbeiter::Mitarbeiter(const Mitarbeiter& m2)
   {geh = m2.geh;}

Mitarbeiter& Mitarbeiter::operator=(const Mitarbeiter& m2)      {
   geh = m2.geh; return *this;
}
```

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
int   geh;
public:
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
             cout <<"-M " << geh;}
};
```

```
void f() {
    Mitarbeiter m1(4);
    Mitarbeiter* p = new Mitarbeiter(5);
}
```

1. +M 4 + M 5 -M 5 – M 4
2. +M 4 + M 5 – M 4
3. +M 4 + M 5 -M 4 – M 5
4. +M 4 + M 5

Ergebnis:
__ 1 _-_ 2 __ 3 __4

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
int   geh;
public:
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
             cout <<"-M " << geh;}
};
```

```
void f() {
      Mitarbeiter m1(4);
      Mitarbeiter* p= new Mitarbeiter(5);
      delete p;
}
```

1. +M 4 + M 5 -M 5 – M 4
2. +M 4 + M 5 – M 4
3. +M 4 + M 5 -M 4 – M 5
4. +M 4 + M 5 – M 4

Ergebnis:
_-_ 1 _ 2 __ 3 __4

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
public:
   int   geh;
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
             cout <<"-M " << geh;}
};
```

```
void help(Mitarbeiter & am) {
      am.geh = 44;
      cout << "Ende“;
}
void f() {
      Mitarbeiter m1(4);
      help(m1);
}
```

```
1000 m1.geh    4     44
1004 am :    1000
```

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
public:
   int   geh;
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
             cout <<"-M " << geh;}
};
```

```
void help(Mitarbeiter*  am) {
      am = new Mitarbeiter(300);
      cout << "Ende";
}
void f() {
      Mitarbeiter m1(4);
      help( &  m1);
}
```

1. +M 4 + M 300 –M 300 Ende – M4
2. +M 4 + M 300 Ende –M300 – M4
3. +M 4 + M 300 Ende – M 4
4. +M 4 + M 300 Ende – M 300

Ergebnis:
__ 1 _ 2 _-_ 3 __4

Speicherbelegung zeichnen !!!

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
public:
   int   geh;
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
             cout <<"-M " << geh;}
};
```

```
void help(Mitarbeiter*  am) {
      am = new Mitarbeiter(300);
      cout << "Ende";
}
void f() {
      Mitarbeiter m1(4);
      help( &  m1);
}
```

1. +M 4 + M 300 –M 300 Ende – M4
2. +M 4 + M 300 Ende –M300 – M4
3. +M 4 + M 300 Ende – M 4
4. +M 4 + M 300 Ende – M 300

Ergebnis:
__ 1 _ 2 _-_ 3 __4

Speicherbelegung zeichnen !!!

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
public:
   int   geh;
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
            cout <<"-M " << geh;}
};
```

```
void help(Mitarbeiter*&  am) {
      am = new Mitarbeiter(300);
      cout << "Ende";
}
void f() {
      Mitarbeiter m1(4);
      Mitarbeiter* p = &m1;
      help( p);
}
```

1. +M 4 + M 300 –M 300 Ende – M4
2. +M 4 + M 300 Ende –M300 – M4
3. +M 4 + M 300 Ende – M 4
4. +M 4 + M 300 Ende – M 300

Ergebnis:
__ 1 _ 2 _-_ 3 __4

Speicherbelegung zeichnen !!!

Clicker: Wie ist die Bildschirm-Ausgabe bei Aufruf von f?

```
class Mitarbeiter  {
public:
   int   geh;
   Mitarbeiter(int g)     {geh = g;
              cout << "+M "  << geh;}
   ~Mitarbeiter()   {
            cout <<"-M " << geh;}
};
```

```
void help(Mitarbeiter*& am) {
      am = new Mitarbeiter(300);
      cout << "Ende";
}
void f() {
      Mitarbeiter m1(4);
      Mitarbeiter*  p = &m1;
      help( p);
      cout <<   p->geh << " ";
}
```

1. +M 4 +M4 + M 300 –M 300 Ende 4 – M4
2. +M 4 + M 300 Ende 300 –M300 – M4
3. +M 4+ M 300 Ende 4 –M 4
4. +M 4+ M 300 Ende 300 –M 4

Ergebnis:
__ 1 _ 2 __ 3 _-_4

Speicherbelegung zeichnen !!!

Schrittweise Ausgabe

```
class Person {
public:
   Person(string s) {na = s; datei<<
        " +P "<< na;}
   Person() {na = "?"; datei <<
        " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
        << na;}
   ~Person() { datei << " -P " << na;
        }
private:
   string na;
};
```

```
void Erz5() {
shared_ptr<Person> p1(new
        Person("Hans"));
datei << " Ende\n"; /* 1 */
shared_ptr<Person> p2 =
   make_shared<Person>(*p1);
datei << " Ende\n "; /* 2 */
p1 = make_shared<Person>("Jack");
datei << " Ende\n"; /* 3 */
} /* 4 */
```

Zusammengesetzte Objekte

Überblick

```
class Kleidung {
public:
   int   code;
   Kleidung(int k=-1) {
      code = k;
      cout << "+K " << k;
   }
   ~Kleidung() {
      cout << "-K " << code;
   }
};
ostream& operator<<(ostream&
      str, const Kleidung& k) {
   str << k.code << "**";
   return str;
}
```

```
class Mitarbeiter {
   Kleidung hose;
   Kleidung socke;
public:
   Mitarbeiter(int h, int s) :
      hose(h), socke(s) {
      cout << "+M " << h << s;
   }
   ~Mitarbeiter() {
      cout << "-M "<<hose<<socke;
   }
};
```

```
void f1() {
   Mitarbeiter hans(4, 9);
   cout << "\nEnde\n";
}
```

Überblick

```
class Kleidung {
public:
   int   code;
   Kleidung(int k=-1) {
      code = k;
      cout << "+K " << k;
   }
   ~Kleidung() {
      cout << "-K " << code;
   }
};
ostream& operator<<(ostream&
      str, const Kleidung& k) {
   str << k.code << "**";
   return str;
}
```

```
class Mitarbeiter {
   Kleidung hose;
   Kleidung socke;
public:
   Mitarbeiter(int h, int s) :
      hose(h), socke(s) {
      cout << "+M " << h << s;
   }
   ~Mitarbeiter() {
      cout << "-M "<<hose<<socke;
   }
};
```

```
void f1() {
   Mitarbeiter hans(4, 9);
   cout << "\nEnde\n";
}
```

+K 4+K 9+M 49
Ende
-M 4**9** -K 9-K 4

Neuer Konstruktor bei Mitarbeiter

```
class Kleidung {
public:
   int   code;
   Kleidung(int k=-1) {
      code = k;
      cout << "+K " << k;
   }
   ~Kleidung() {
      cout << "-K " << code;
   }
};
ostream& operator<<(ostream&
      str, const Kleidung& k) {
   str << k.code << "**";
   return str;
}
```

```
class Mitarbeiter {
   Kleidung hose;
   Kleidung socke;
public:
 Mitarbeiter() {
   hose = Kleidung(4);
   socke = Kleidung(14);
   cout << "+M " << hose << socke;
   }
~Mitarbeiter() {
      cout << "-M "<<hose<<socke;
   }
};
```

```
void f2() {
   Mitarbeiter ina;
   cout << "\nEnde\n";
}
```

Zusammenfassung / Lösung

```
class Kleidung {
public:
   int   code;
   Kleidung(int k=-1) {
      code = k;
      cout << "+K " << k;
   }
   ~Kleidung() {
      cout << "-K " << code;
   }
};
ostream& operator<<(ostream&
      str, const Kleidung& k) {
   str << k.code << "**";
   return str;
}
```

```
class Mitarbeiter {
   Kleidung hose;
   Kleidung socke;
public:
 Mitarbeiter() {
   hose = Kleidung(4);
   socke = Kleidung(14);
   cout << "+M " << hose << socke;
   }
~Mitarbeiter() {
      cout << "-M "<<hose<<socke;
   }
};
```

```
void f2() {
   Mitarbeiter ina;
   cout << "\nEnde\n";
}
```

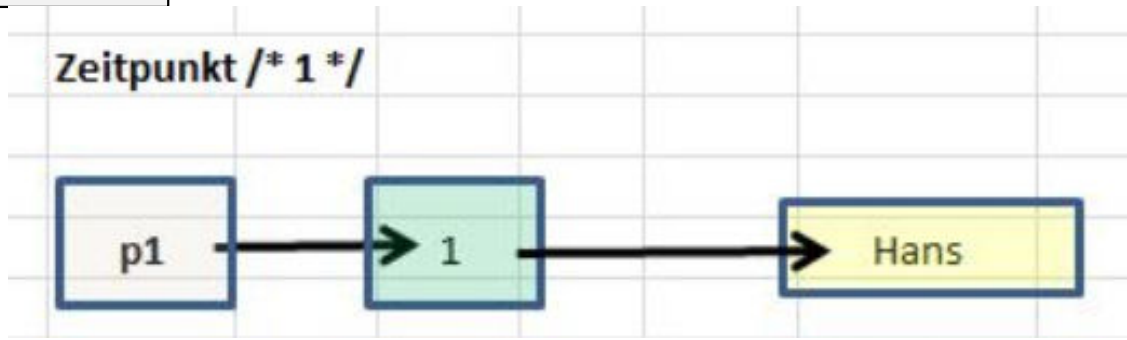
```
+K -1+K -1+K 4-K 4+K 14-K 14+M 4**14**
Ende
-M 4**14**-K 14-K 4
```

Schrittweise Ausgabe 1

Welche Ausgabe ergibt sich bis /* 1 */?

```
class Person {
public:
   Person(string s) {na = s; datei<<
        " +P "<< na;}
   Person() {na = "?"; datei <<
        " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
        << na;}
   ~Person() { datei << " -P " << na;
        }
private:
   string na;
};
```

```
void Erz5() {
shared_ptr<Person> p1(new
        Person("Hans"));
datei << " Ende\n"; /* 1 */

shared_ptr<Person> p2 =
   make_shared<Person>(*p1);
datei << " Ende\n "; /* 2 */
p1 = make_shared<Person>("Jack");
datei << " Ende\n"; /* 3 */
} /* 4 */
```



Welche Ausgabe ergibt sich zwischen /* 1 */ und /* 2 */

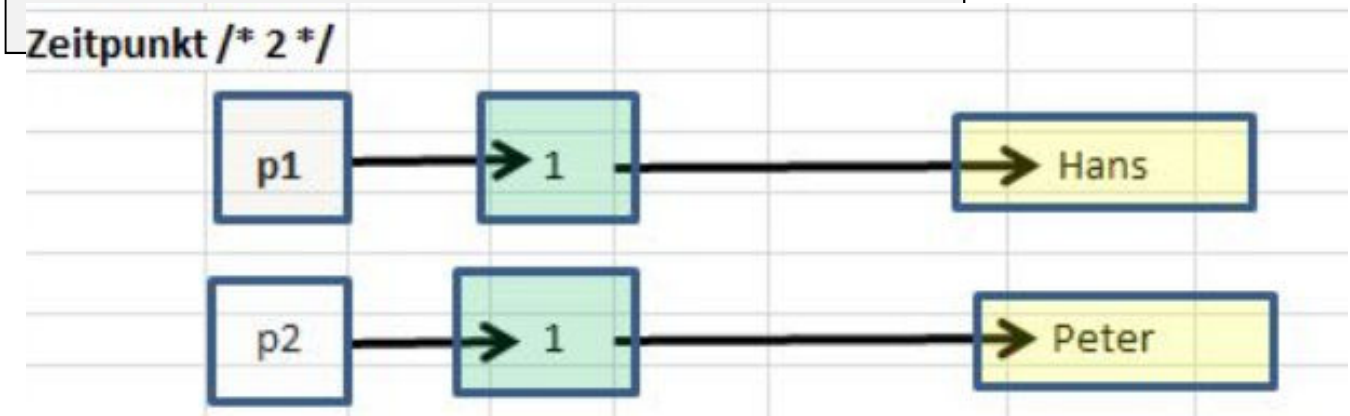
```
class Person {
public:
   Person(string s) {na = s; datei<<
        " +P "<< na;}
   Person() {na = "?"; datei <<
        " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
        << na;}
   ~Person() { datei << " -P " << na;
        }
private:
   string na;
};
```

```
void Erz5() {
shared_ptr<Person> p1(new Person("Hans"));
datei << " Ende\n"; /* 1 */

shared_ptr<Person> p2 =
   make_shared<Person>(*p1);
datei << " Ende\n "; /* 2 */

p1 = make_shared<Person>("Jack");
datei << " Ende\n"; /* 3 */
} /* 4 */
```

Welche Ausgabe ergibt sich zwischen /* 2 */ und /* 3 */

```
class Person {
public:
   Person(string s) {na = s; datei<<
        " +P "<< na;}
   Person() {na = "?"; datei <<
        " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
        << na;}
   ~Person() { datei << " -P " << na;
        }
private:
   string na;
};
```

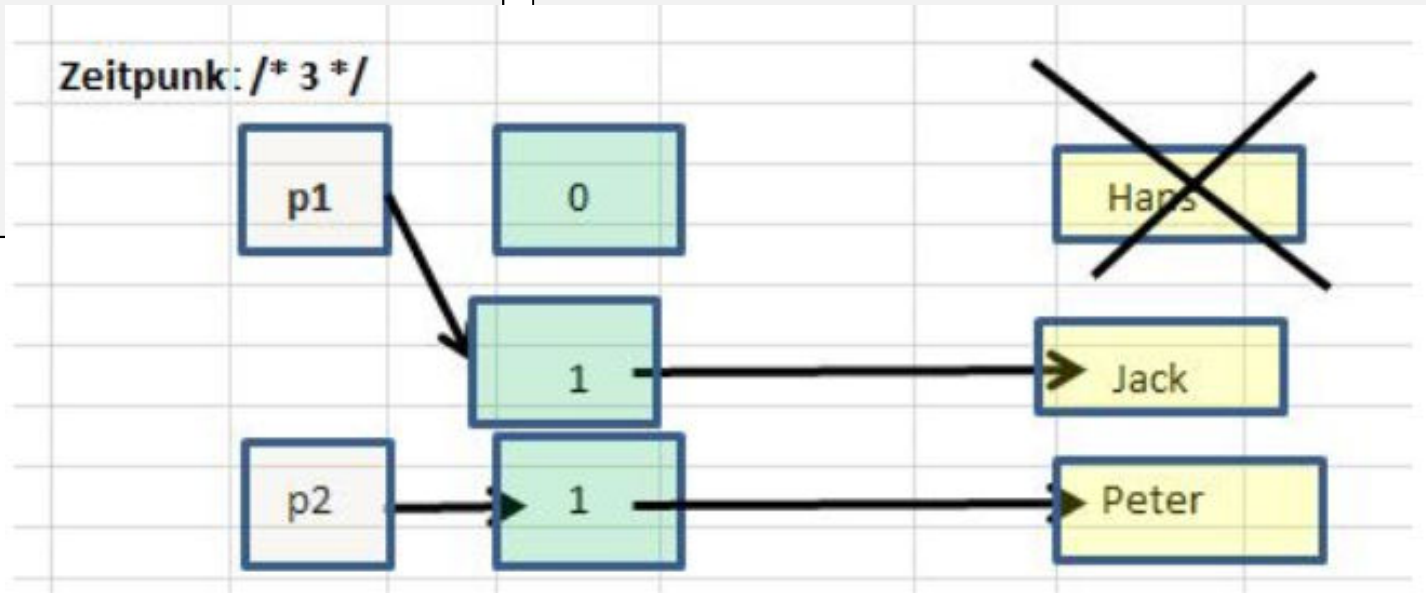
```
void Erz5() {
shared_ptr<Person> p1(new Person("Hans"));
datei << " Ende\n"; /* 1 */
shared_ptr<Person> p2 =
   make_shared<Person>(*p1);
datei << " Ende\n "; /* 2 */

p1 = make_shared<Person>("Jack");
datei << " Ende\n"; /* 3 */

} /* 4 */
```



Welche Ausgabe ergibt sich zwischen /* 3 */ und /* 4 */

```
class Person {
public:
   Person(string s) {na = s; datei<<
         " +P "<< na;}
   Person() {na = "?"; datei <<
         " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
        << na;}
   ~Person() { datei << " -P " << na;
         }
private:
   string na;
};
```

```
void Erz5() {
shared_ptr<Person> p1(new Person("Hans"));
datei << " Ende\n"; /* 1 */
shared_ptr<Person> p2 =
   make_shared<Person>(*p1);
datei << " Ende\n "; /* 2 */
p1 = make_shared<Person>("Jack");
datei << " Ende\n"; /* 3 */

} /* 4 */
```



Schrittweise Ausgabe

```
class Person {
public:
   Person(string s) {na = s; datei<<
        " +P "<< na;}
   Person() {na = "?"; datei <<
        " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
        << na;}
   ~Person() { datei << " -P " << na;
        }
private:
   string na;
};
```

```
void Erz5() {
shared_ptr<Person> p1(new
        Person("Hans"));
datei << " Ende\n"; /* 1 */
shared_ptr<Person> p2 =
   make_shared<Person>(*p1);
datei << " Ende\n "; /* 2 */
p1 = make_shared<Person>("Jack");
datei << " Ende\n"; /* 3 */
} /* 4 */
```

+P Hans Ende
+PCop Peter Ende
+P Jack -P Hans Ende
-P Peter -P Jack

Schrittweise Ausgabe 2

Überblick

```
class Person {
public:
   Person(string s) {na = s; datei<<
          " +P "<< na;}
   Person() {na = "?"; datei <<
          " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
          << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

```
void Erz6() {
shared_ptr<Person>p1(new
        Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */
   if (p1 != nullptr)    {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else    {
      shared_ptr<Person> p3(p1);/* YY*/
   }
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
   p4 = make_shared<Person>("Mike");
   datei << " Ende\n"; /* 4 */
} /* 5 */
```

Welche Ausgabe ergibt sich bis /* 1 */?

```
class Person {
public:
   Person(string s) {na = s; datei<<
          " +P "<< na;}
   Person() {na = "?"; datei <<
          " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
          << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

```
void Erz6() {
shared_ptr<Person>p1(new
        Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */

   if (p1 != nullptr)    {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else    {
      shared_ptr<Person> p3(p1);/* YY*/
   }
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
   p4 = make_shared<Person>("Mike");
   datei << " Ende\n"; /* 4 */
} /* 5 */
```

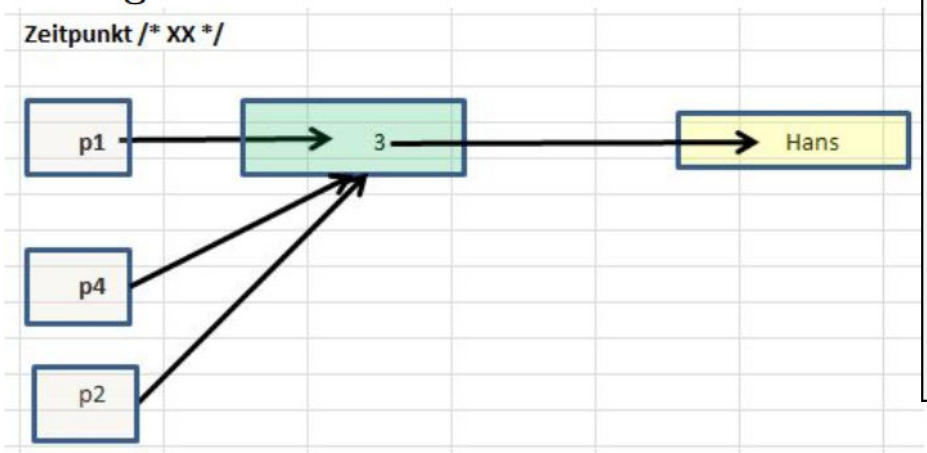
Welche Ausgabe ergibt sich zwischen /* 1 */ und /* 2*/?

```
class Person {
public:
   Person(string s) {na = s; datei<<
         " +P "<< na;}
   Person() {na = "?"; datei <<
         " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
         << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

```
void Erz6() {
shared_ptr<Person>p1(new Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */

    if (p1 != nullptr)    {
       shared_ptr<Person> p2 = p1;
       p4 = p1; /* XX */
    }
    else     {
       shared_ptr<Person> p3(p1);/* YY*/
    }
    datei << " Ende\n "; /* 2 */

   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
   p4 = make_shared<Person>("Mike");
   datei << " Ende\n"; /* 4 */
} /* 5 */
```

Welche Ausgabe ergibt sich zwischen /* 2 */ und /*3*/?

```
class Person {
public:
   Person(string s) {na = s; datei<<
         " +P "<< na;}
   Person() {na = "?"; datei <<
         " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
         << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

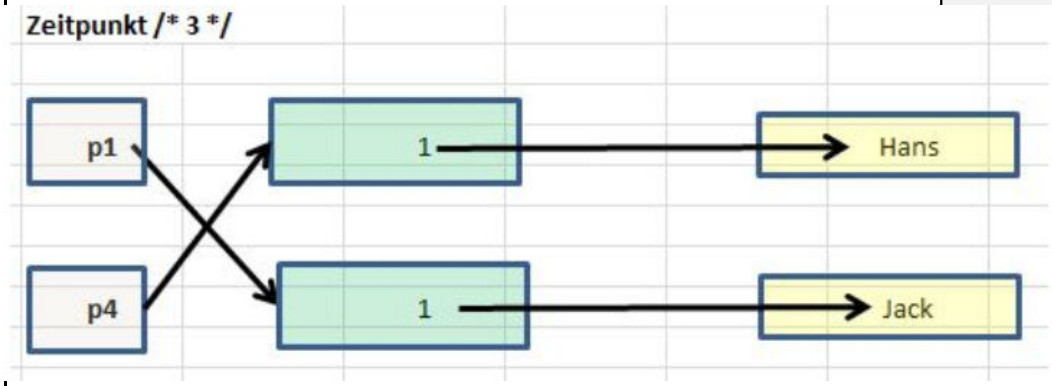



```
void Erz6() {
shared_ptr<Person>p1(new Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */
   if (p1 != nullptr)    {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else    {
      shared_ptr<Person> p3(p1);/* YY*/
   }
   datei << " Ende\n "; /* 2 */

    p1 = make_shared<Person>("Jack");
    datei << " Ende\n"; /* 3 */

   p4 = make_shared<Person>("Mike");
   datei << " Ende\n"; /* 4 */
} /* 5 */
```

Welche Ausgabe ergibt sich zwschen /* 3 */ und /*4*/?

```
class Person {
public:
   Person(string s) {na = s; datei<<
         " +P "<< na;}
   Person() {na = "?"; datei <<
         " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
         << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

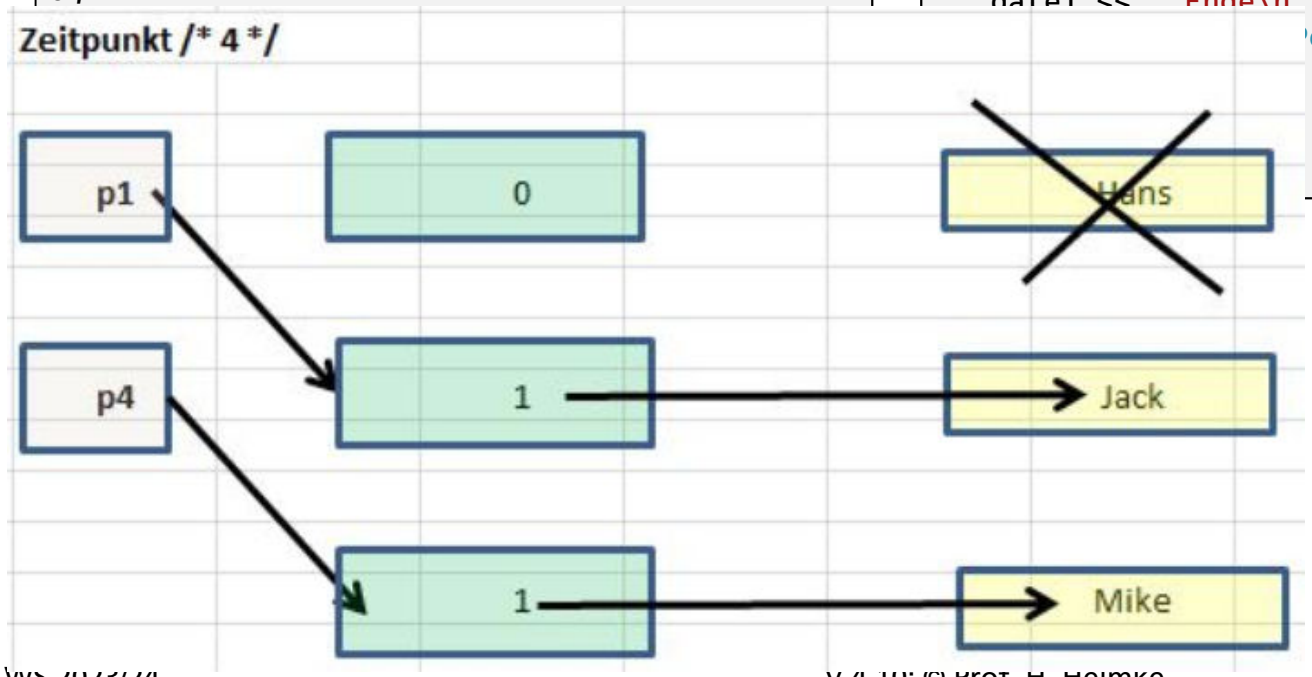
```
void Erz6() {
shared_ptr<Person>p1(new Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */
   if (p1 != nullptr)    {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else    {
      shared_ptr<Person> p3(p1);/* YY*/
   }
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */

    p4 = make_shared<Person>("Mike");
    datei << " Ende\n"; /* 4 */

5 */
```



Welche Ausgabe ergibt sich bei /* 5 */?

```
class Person {
public:
   Person(string s) {na = s; datei<<
          " +P "<< na;}
   Person() {na = "?"; datei <<
          " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
          << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

```
void Erz6() {
shared_ptr<Person>p1(new Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */
   if (p1 != nullptr)    {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else    {
      shared_ptr<Person> p3(p1);/* YY*/
   }
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
erson>("Mike");
; /* 4 */
```



Überblick

```
class Person {
public:
   Person(string s) {na = s; datei<<
          " +P "<< na;}
   Person() {na = "?"; datei <<
          " +PDef " << na;}
   Person(const Person&) {
      na = "Peter"; datei << " +PCop "
          << na;}
   ~Person() { datei << " -P " << na; }
private:
   string na;
};
```

```
void Erz6() {
shared_ptr<Person>p1(new
        Person("Hans"));
   shared_ptr<Person> p4;
   datei << " Stop\n"; /* 1 */
   if (p1 != nullptr)    {
      shared_ptr<Person> p2 = p1;
      p4 = p1; /* XX */
   }
   else    {
      shared_ptr<Person> p3(p1);/* YY*/
   }
   datei << " Ende\n "; /* 2 */
   p1 = make_shared<Person>("Jack");
   datei << " Ende\n"; /* 3 */
   p4 = make_shared<Person>("Mike");
   datei << " Ende\n"; /* 4 */
} /* 5 */
```

+P Hans Stop
Ende
+P Jack Ende
+P Mike -P Hans Ende
-P Mike -P Jack