1a) rein(rein(rein(neu(), oben(raus(raus(s)))),
oben(raus(s))),
oben(s))

b) rein(rein(rein(neu(), oben(s)),
oben(raus(s))),
oben(raus(raus(s)))) ✓ 8

2a) Sobald f2 = f1; ausgeführt wird,
~~wird~~ besitzt f2 den selben pointer
wie f1. Nach dem Block wird f2
wieder gelöscht, der Pointer zeigt also
auf leeren Speicher. ✓

Um das zu beheben ist es nötig einen
benutzerdefinierten Copy-Konstruktor zu
schreiben ✓

```
BadField ( const BadField &a){
    m_size = a.m_size;
    m_data = new float[m_size];
    for (int i=0; i < m_size; i++){
        m_data[i] = a.m_data[i];
    }
}
```

✓ 8

2b) Es handelt sich um eine Zuweisung bei test2();.
Wenn man den = Operator wie folgt überlädt, wird beachtet, dass das zu kopierende Objekt nicht das übergebene Objekt ist, und dass die Objekte eine unterschiedliche Länge haben: ✓

```
BadField & operator=( const BadField& a) {
    if (&a != this) {
        if (m_size != a.m_size){
            delete m_data;
            m_size = a.m_size;
            m_data = new float[m_size];
        }
    for (int i=0 ; i < m_size ; i++){
        m_data[i] = a.m_data[i];}
    }
    return *this;
}
```

✓ 8

3) In Zeile 26 wird d1.do_derived() aufgerufen.
Der Aufruf wird in Zeile 12 die Methode
do_derived_private() aufrufen. Dieser Aufruf
führt in Zeile 15 die Methode do_base_private()
aus, auf die nicht zugegriffen werden kann. D.h.:
in Zeile 15 wird eine nicht-zugängliche Funktion
aufgerufen, da diese den Zugriffsmodifikator
private hat. ✓
In Zeile 27 wird d1.do_base aufgerufen. Dies
ist weiter kein Problem, da do_base den Zugriffs-
modifikator public hat.
In Zeile 28 wird d2.do_derived aufgerufen. Es wird
also in Zeile 20 die vererbte Methode do_base
aufgerufen. Diese Methode ist public, wird aber in
Zeile 18 als private base vererbt, weshalb die
Methode do_base nicht zugänglich ist.
In Zeile 29 wird d2.do_base() aufgerufen - d2 hat
wie beschrieben die Klasse base als private
vererbt, do_base bleibt also nicht zugänglich ✓

6