```cpp
// RalfsTestfeld.cpp : Diese Datei enthält die Funktion "main". Hier beginnt und endet die Ausführung des Programms.
//

#include <iostream>
#include "RalfsTestfeldDefNSubfunc.h" // Einbinden der Unterfunktionen

using namespace std; // Dann muss es nicht mehr std::cout heißen, sondern es kann auch nur cout verwendet werden



class Testclass
{
public:
    Testclass() {}; // Leerer Konstruktor -> benötigt jetzt die geschweiften Klammern, sonst gibt's nen Fehler
    // Testclass(){ cout << "Konstruktor aufgerufen\n";}; // Konstruktor mit Textausgabe
    // ~Testclass(); // Dekonstruktor
    int var1 = 3;
    const int GetWert(); // Zugriffsfunktion ohne Wertänderung
    void SetWert(int Wert); // Zugriffsfunktion mit Wertänderung

private:
    int var2 = 7; // auf den Wert kann somit nicht direkt zugegriffen werden!
};

const int Testclass::GetWert()
{
    // cout << "(Der Wert im Speicher wird abgerufen.)\n";
    return var2;
}

void Testclass::SetWert(int Wert)
{
    var2 = Wert;
    // cout << "(Wert im Speicher wurde gesetzt.)\n";
}


//-----------------------------------------------------------------
class Oberklasse : public Testclass// Oberklasse soll alles von Testclass erben
{
public:
    // Oberklasse(); // Konstruktor
    // ~Oberklasse(); // Destruktor
    int Ovar1 = 17;
    const int ReadWert(); // Zugriffsfunktion
    void NewWert(int Owert); // Zugriffsfunktion zur Wertänderung
private:
    int Ovar2 = 20;
```

```
49  };
50
51  const int Oberklasse::ReadWert()
52  {
53      // cout << "(Der Wert im Speicher wird abgerufen.)\n";
54      return Ovar2;
55  }
56
57  void Oberklasse::NewWert(int Owert)
58  {
59      Ovar2 = Owert;
60      // cout << "(Wert im Speicher wurde gesetzt.)\n";
61  }
62
63
64  //---------------------------------------------------------------
65  int main()
66  {
67      Testclass tester;
68      cout << "Wert in tester.var2 ist: ";
69      cout << tester.GetWert() << "\n"; // Hier die Klammern nicht vergessen,
          sonst gibt es ne Fehlermeldung
70      cout << "Setzen des Wertes von tester.var2 auf 5 durch Zugriffsfunktion.
          \n";
71      tester.SetWert(5);
72      cout << "Wert in tester.var2 ist: ";
73      cout << tester.GetWert()  << "\n"; // Hier die Klammern nicht vergessen,
          sonst gibt es ne Fehlermeldung
74
75      cout << "wert von tester.var1: ";
76      cout << tester.var1 << "\n";
77  //  ----------------------------------------
78      Oberklasse Otest;
79      cout << "Wert in Otest.Ovar2 ist: ";
80      cout << Otest.ReadWert() << "\n"; // Hier die Klammern nicht vergessen,
          sonst gibt es ne Fehlermeldung
81      cout << "Wert in Otest.var2 ist: ";
82      cout << Otest.GetWert() << "\n"; // Der Standardwert für var2 wird
          ausgegeben, da dieser beim initialisieren/vererben mit diesen Werten
          übernommen wird!
83      // Vergleich mit dem aktuellen Wert im tester Object
84      cout << "Wert in tester.var2 ist: ";
85      cout << tester.GetWert() << "\n"; // Hier die Klammern nicht vergessen,
          sonst gibt es ne Fehlermeldung
86
87      // Verändern des Wertes in Otest
88      cout << "Setzen des Wertes von Otest.Ovar2 auf 42 durch Zugriffsfunktion.
          \n";
89      Otest.NewWert(42);
90      cout << "Setzen des Wertes von Otest.var2 auf 21 durch Zugriffsfunktion.
          \n";
91      Otest.SetWert(21);
92      cout << "Wert in Otest.Ovar2 ist: ";
```

```
93      cout << Otest.ReadWert() << "\n"; // Der Standardwert für var2 wird
          ausgegeben, da dieser beim initialisieren/vererben mit diesen Werten
          übernommen wird!
94      cout << "Wert in Otest.var2 ist: ";
95      cout << Otest.GetWert() << "\n"; // Der Standardwert für var2 wird
          ausgegeben, da dieser beim initialisieren/vererben mit diesen Werten
          übernommen wird!
96
97      // Unterfunktion einbinden
98
99      int Ausgabe = WertBerechnen(15, 17);
100     cout << "Die Unterfunktion berechnet: "<< Ausgabe << "\n" ;
101
102     double AusgabeDouble = WertBerechnen(15.0f, 17.0f);
103     cout << "Die Unterfunktion berechnet: " << AusgabeDouble << "\n";
104
105
106
107 //  ------------------------------------------
108     cout << "Programm ist fertig...\n";
109     return 0;
110 }
111
112
113
114
115 // Programm ausführen: STRG+F5 oder "Debuggen" > Menü "Ohne Debuggen starten"
116 // Programm debuggen: F5 oder "Debuggen" > Menü "Debuggen starten"
117
118 // Tipps für den Einstieg:
119 //   1. Verwenden Sie das Projektmappen-Explorer-Fenster zum Hinzufügen/
      Verwalten von Dateien.
120 //   2. Verwenden Sie das Team Explorer-Fenster zum Herstellen einer
      Verbindung mit der Quellcodeverwaltung.
121 //   3. Verwenden Sie das Ausgabefenster, um die Buildausgabe und andere
      Nachrichten anzuzeigen.
122 //   4. Verwenden Sie das Fenster "Fehlerliste", um Fehler anzuzeigen.
123 //   5. Wechseln Sie zu "Projekt" > "Neues Element hinzufügen", um neue
      Codedateien zu erstellen, bzw. zu "Projekt" > "Vorhandenes Element
      hinzufügen", um dem Projekt vorhandene Codedateien hinzuzufügen.
124 //   6. Um dieses Projekt später erneut zu öffnen, wechseln Sie zu "Datei" >
      "Öffnen" > "Projekt", und wählen Sie die SLN-Datei aus.
```