Stephan Brumme, SST, 3.FS, Matrikelnr. 70 25 44
mail@stephan-brumme.com

Aufgabe M7.2

Die einzelnen Tests werden sequentiell abgearbeitet und erzeugen dabei Statusmeldungen auf der Standardausgabe. Im realen Umfeld empfiehlt es sich, diese in eine Datei umzuleiten, insbesondere wenn die Ausgaben länger sind, da ich keine Warteschleifen etc. eingebaut habe.

Zusätzlich zu den vorgegebenen Test-Aufrufen musste ich noch etwas Code zur Initialisierung der verwendeten Variablen schreiben. Um mir dabei Arbeitsaufwand zu sparen, benutze ich die Variablen in mehrfach, falls dabei Abhängigkeiten auftreten, so habe ich dies in den entsprechenden Kommentaren vermerkt.

Alle verwendeten Module entsprechen genau den Versionen, die schon für die vorherigen Aufgaben erstellt habe. Ich führe sie hier nicht auf, sie sind aber im zugehörigen Zip-Archiv.

M07_2.cpp:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M7.2.
//
// author:            Stephan Brumme
// last changes:      January 13, 2001


// import cout to display some data
#include <iostream>
#include "PrimitiveTypes.h"
#include "MDate.h"
#include "MRoom.h"
#include "MHouse.h"

// open std namespace
using namespace std;


void main()
{
       // declare our variables
       MHouse::THouse hou, hou1, hou2;
       MRoom::TRoom   roo, roo1, roo2;
       Boolean t1,t2,t3;
       Ordinal n1,n2;

       // initialize them
       MHouse::Init(hou);
       MHouse::Init(hou1);
       MHouse::Init(hou2);

       MRoom::Init(roo,  2, 3);
       MRoom::Init(roo1, 4, 5);
       MRoom::Init(roo2, 10, 10);

       t1 = t2 = t3 = false;
       n1 = n2 = 0;


       // Case 1
       // require:    the sets hou1 and hou2 are exemplars of MHouse::THouse
       // sequence:   MHouse::Copy(hou1, hou2, t1);
       //             MHouse::EqualValue(hou1, hou2, t2);
       // ensure:     if the value of t1 is TRUE then t2 is TRUE, too

       cout<<"Case 1"<<endl;

       // ensure that hou1 differs from hou2, otherwise Copy will fail
       if (!MHouse::Insert(hou1, roo1))
       {
              cout<<"preparation for case 1 failed !!!"<<endl;
              exit(1);
       }

       // first step of Case 1
       t1 = MHouse::Copy(hou1, hou2);
```

```
cout<<"t1: "<<t1<<endl;

// second step only neccessary when first succeeded
if (t1)
{
	t2 = MHouse::EqualValue(hou1, hou2);

	cout<<"t2: "<<t2<<endl<<endl;
}

if (!(t1&&t2))
{
	cout<<"Case 1 failed !!!"<<endl<<"test aborted."<<endl;
	exit(1);
}


// Case 2
// require:    the set hou as an exemplar of MHouse::THouse is not full
//             and roo1, roo2 are exemplars of MRoom::TRoom
// sequence:   MHouse::Insert(hou, roo1, t1);
//             MHouse::Find(hou, roo1, t2);
//             MHouse::GetCurrent(hou, roo2, t3);
// ensure:     the elements roo1 and roo2 have the same value
//             (MRoom::EqualValue); the values of t2 and t3 are TRUE

cout<<"Case 2"<<endl;

// try first command
t1 = MHouse::Insert(hou, roo1);
cout<<"t1: "<<t1<<endl;

if (t1)
{
	// only if first command was successful
	t2 = MHouse::Find(hou, roo1);
	cout<<"t2: "<<t2<<endl;

	if (t2)
	{
		// only if second command was successful
		t3 = MHouse::GetCurrent(hou, roo2);
		cout<<"t3: "<<t3<<endl;
	}
}

cout<<"roo1==roo2: "<<MRoom::EqualValue(roo1, roo2)<<endl<<endl;

if (!(t2 && t3 && MRoom::EqualValue(roo1, roo2)))
{
	cout<<"Case 2 failed !!!"<<endl<<"test aborted."<<endl;
	exit(2);
}


// Case 3
// require:    the set hou as an exemplar of MHouse::THouse contains roo
//             (MHouse::Find)
// sequence:   MHouse::Find(hou, roo, t1);
//             MHouse::Scratch(hou, t2);
//             MHouse::Find(hou, roo, t3);
// ensure:     t2 is TRUE, t3 is FALSE

cout<<"Case 3"<<endl;

// we already inserted roo1 into hou in Case 2
// so our roo is roo1 in this Case !
t1 = MHouse::Find(hou, roo1);
cout<<"t1: "<<t1<<endl;

if (t1)
{
	t2 = MHouse::Scratch(hou);
	cout<<"t2: "<<t2<<endl;
```

```
        if (t2)
        {
                t3 = MHouse::Find(hou, roo1);
                cout<<"t3: "<<t3<<endl<<endl;
        }
}

if (!(t1 && t2 && !t3))
{
        cout<<"Case 3 failed !!!"<<endl<<"test aborted."<<endl;
        exit(3);
}


// Case 4
// require:    the set hou as an exemplar of MHouse::THouse is not full
//             it does not contain roo (MHouse::Find)
//             the set has the cardinality (MHouse::Card) of n1
// sequence:   MHouse::Insert(hou, roo, t1);
//             MHouse::Card(hou, n2);
// ensure:     t2 is equal to n1+1

cout<<"Case 4"<<endl;

// prepare Case 4
if (MHouse::Find(hou, roo))
{
        cout<<"preparation for case 1 failed, hou DOES contain roo !!!"<<endl;
        exit(4);
}

n1 = MHouse::Card(hou);
cout<<"n1: "<<n1<<endl;
// preparation done right now

// Case 4 starts here
t1 = MHouse::Insert(hou, roo);
cout<<"t1: "<<t1<<endl;

if (t1)
{
        // get cardinality
        n2 = MHouse::Card(hou);
        cout<<"n2: "<<n2<<endl<<endl;
}

// success ?
if (!(t1 && (n2 == n1+1)))
{
        cout<<"Case 4 failed !!!"<<endl<<"test aborted."<<endl;
        exit(4);
}


// Case 5
// require:    the set hou as an exemplar of MHouse::THouse contains roo
//             (MHouse::Find) and has the cardinality (MHouse::Card) of n1
// sequence:   MHouse::Find(hou, roo, t1);
//             MHouse::Scratch(hou, t2);
//             MHouse::Card(hou, n2);
// ensure:     n2 is equal to n1-1

cout<<"Case 5"<<endl;

// roo was inserted in Case 4 into hou, we re-use it
// n1 was sets properly in Case 4, too

t1 = MHouse::Find(hou, roo1);
cout<<"t1: "<<t1<<endl;

if (t1)
{
        t2 = MHouse::Scratch(hou);
        cout<<"t2: "<<t2<<endl;
```

```
                if (t2)
                {
                        n2 = MHouse::Card(hou);
                        cout<<"n2: "<<n2<<endl<<endl;
                }
        }

        // success ?
        if (!(n2==n1-1))
        {
                cout<<"Case 5 failed !!!"<<endl<<"test aborted."<<endl;
                exit(5);
        }


        // done
        cout<<"test frame succesfully executed."<<endl;
}
```