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

Aufgabe M6.5.

Nach außen hin ist nur der abstrakte Datentyp

```
typedef void* UHouseADT;
```

sichtbar. Er schützt vor ungewollten Zugriffen auf die interne Datenstruktur

```
typedef struct
{
      TDate   DateOfFoundation;
      Ordinal Count;
      Ordinal Cursor;
      MHouseADT::TArrayOfRooms PSet;
} THouseADTIntern;
```

In den einzelnen Routinen muss nun zusätzlich ein Typ-Casting vorgenommen werden. Dies geschieht i.d.R. durch Zeilen der Form

```
THouseADTIntern* house = (THouseADTIntern*) HouseADT;
```

Über Zugriffe auf `house` kann nun der abstrakte Datentyp manipuliert werden. In der Funktion `Insert` ist dies beispielhaft umgesetzt worden:

```
// add a new room to a HouseADT
Boolean MHouseADT::Insert(UHouseADT HouseADT, const MRoom::TRoom& room)
{
      THouseADTIntern* house = (THouseADTIntern*) HouseADT;

      // is any free space available ?
      if (house->Count >= ROOMS)
            return false;

      // copy room, care for errors
      if (!MRoom::Copy(house->PSet[house->Count], room))
            return false;

      // set cursor
      house->Cursor = house->Count;
      // increase size of array
      house->Count++;

      return true;
}
```

Die Dateien *PrimitiveTypes.h*, *MRoom.** und *MDate.** sind unverändert, ich führe sie hier nicht auf, sie finden sich in aber im beigelegten Zip-Archiv. Vom Aufbau her ähneln *MHouseADT.** sehr stark *MHouse.** aus der Aufgabe 6.2., deshalb belasse ich es bei den Kommentare, die direkt im Quellcode sind.
In *M06_5.cpp* finden sich lediglich einige Zeilen Code, die dem Test der Routinen dienen.

MHouseADT.h:

```
////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.5.
//
// author:            Stephan Brumme
// last changes:      January 07, 2001


#ifndef __MHOUSEADT_H__
#define __MHOUSEADT_H__

// we need the CEDL-types Ordinal, ...
#include "PrimitiveTypes.h"

// import MDate
#include "MDate.h"
// import MRoom
#include "MRoom.h"


namespace MHouseADT
{
	// first the abstract data type
	typedef void* UHouseADT;


	// open namespaces MRoom and MDate
	using namespace MRoom;
	using namespace MDate;

	// max. number of rooms in a HouseADT
	const Ordinal ROOMS = 20;

	// array of pointers containing the rooms
	typedef TRoom TArrayOfRooms[ROOMS];

	// constructs a HouseADT
	void NewHouseADT(UHouseADT* HouseADT);

	// destructs a HouseADT
	void DeleteHouseADT(UHouseADT* HouseADT);

	// initializes the UHouseADT structure
	void Init(UHouseADT HouseADT);

	// compares two exemplars
	// returns "true" if attributes of both are equal; "false" otherwise
	Boolean EqualValue(UHouseADT HouseADT1, UHouseADT HouseADT2);

	// copies the attributes of HouseADT2
	// returns "true" if successful, "false" if no memory allocated
	Boolean Copy(UHouseADT HouseADT1, UHouseADT HouseADT2);

	// retrieve date of foundation
	TDate GetDateOfFoundation(UHouseADT HouseADT);

	// get number of rooms
	Ordinal Card(UHouseADT HouseADT);

	// add a new room to a HouseADT
	Boolean Insert(UHouseADT HouseADT, const TRoom& room);

	// returns the first room of a HouseADT
	Boolean GetFirst(UHouseADT HouseADT, TRoom& room);

	// returns the last room of a HouseADT
	Boolean GetNext(UHouseADT HouseADT, TRoom& room);
```

```
        // looks for a given room and sets cursor, if possible
        Boolean Find(UHouseADT HouseADT, const TRoom& room);

        // returns the room the cursors points to
        Boolean GetCurrent(UHouseADT HouseADT, TRoom& room);

        // deletes the room the cursor points to
        Boolean Scratch(UHouseADT HouseADT);

        // displays the attributes
        void Show(UHouseADT HouseADT);
};

#endif
```

MHouseADT.cpp:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.5.
//
// author:            Stephan Brumme
// last changes:      January 07, 2001


#include <iostream>
#include "MDate.h"
#include "MRoom.h"
#include "MHouseADT.h"


using namespace MDate;


// data struct THouseADTIntern for a single HouseADT
// needed to hide the internal structure of the ADT
typedef struct
{
        TDate   DateOfFoundation;
        Ordinal Count;
        Ordinal Cursor;
        MHouseADT::TArrayOfRooms PSet;
} THouseADTIntern;


// constructs a HouseADT
void MHouseADT::NewHouseADT(UHouseADT* HouseADT)
{
        *HouseADT = malloc(sizeof(THouseADTIntern));
        Init(*HouseADT);
}


// destructs a HouseADT
void MHouseADT::DeleteHouseADT(UHouseADT* HouseADT)
{
        delete(*HouseADT);
        HouseADT = NULL;
}


// initializes the UHouseADT structure
void MHouseADT::Init(UHouseADT HouseADT)
{
        THouseADTIntern* house = (THouseADTIntern*) HouseADT;

        house->Count  = 0;
        house->Cursor = -1;
        MDate::Today(house->DateOfFoundation);
}


// compares two exemplars
```

```
// returns "true" if attributes of both are equal; "false" otherwise
Boolean MHouseADT::EqualValue(UHouseADT HouseADT1, UHouseADT HouseADT2)
{
	THouseADTIntern* house1 = (THouseADTIntern*) HouseADT1;
	THouseADTIntern* house2 = (THouseADTIntern*) HouseADT2;

	// verify number of rooms and date of foundation
	if (!(house1->Count == house2->Count &&
		MDate::EqualValue(house1->DateOfFoundation, house2->DateOfFoundation)))
		return false;

	// dates and numbers of rooms are equal
	// now compare each room
	// !!! they must be in the same order, shuffled HouseADTs are NOT recognized !!!

	// are HouseADTs empty ? => they are equal
	if (house1->Count == 0)
		return true;

	// compare each room
	for (Ordinal nRun = 0; nRun < house1->Count; nRun++)
		if (!MRoom::EqualValue(house1->PSet[nRun], house2->PSet[nRun]))
			return false;

	// all rooms are equal, so the HouseADTs are equal
	return true;
}


// copies the attributes of HouseADT2
// returns "true" if successful, "false" if no memory allocated
Boolean MHouseADT::Copy(UHouseADT HouseADT1, UHouseADT HouseADT2)
{
	// are both HouseADTs equal ?
	if (EqualValue(HouseADT1, HouseADT2))
		return false;

	THouseADTIntern* house1 = (THouseADTIntern*) HouseADT1;
	THouseADTIntern* house2 = (THouseADTIntern*) HouseADT2;

	// copy all attributes
	house1->Count  = house2->Count;
	house1->Cursor = house2->Cursor;
	MDate::Copy(house1->DateOfFoundation, house2->DateOfFoundation);

	if (house2->Count > 0)
		for (Ordinal nRun = 0; nRun < house2->Count; nRun++)
			MRoom::Copy(house1->PSet[nRun], house2->PSet[nRun]);

	// successfully done
	return true;
}


// retrieve date of foundation
TDate MHouseADT::GetDateOfFoundation(UHouseADT HouseADT)
{
	TDate date;
	MDate::Copy(date, ((THouseADTIntern*)HouseADT)->DateOfFoundation);

	return date;
}


// get number of rooms
Ordinal MHouseADT::Card(UHouseADT HouseADT)
{
	return ((THouseADTIntern*)HouseADT)->Count;
}


// add a new room to a HouseADT
Boolean MHouseADT::Insert(UHouseADT HouseADT, const MRoom::TRoom& room)
{
	THouseADTIntern* house = (THouseADTIntern*) HouseADT;
```

```
        // is any free space available ?
        if (house->Count >= ROOMS)
                return false;

        // copy room, care for errors
        if (!MRoom::Copy(house->PSet[house->Count], room))
                return false;

        // set cursor
        house->Cursor = house->Count;
        // increase size of array
        house->Count++;

        return true;
}


// returns the first room of a HouseADT
Boolean MHouseADT::GetFirst(UHouseADT HouseADT, MRoom::TRoom& room)
{
        THouseADTIntern* house = (THouseADTIntern*) HouseADT;

        // is HouseADT empty ?
        if (house->Count == 0)
                return false;

        // set cursor
        house->Cursor = 0;

        // and return room
        return GetCurrent(HouseADT, room);
}


// returns the last room of a HouseADT
Boolean MHouseADT::GetNext(UHouseADT HouseADT, MRoom::TRoom& room)
{
        THouseADTIntern* house = (THouseADTIntern*) HouseADT;

        // verify that current cursor position is valid
        // they must be a next room, otherwise function will fail
        if ((house->Cursor < 0) ||
                (house->Cursor >= house->Count))
                return false;

        // set cursor to next room
        house->Cursor++;

        return GetCurrent(HouseADT, room);
}


// looks for a given room and sets cursor, if possible
Boolean MHouseADT::Find(UHouseADT HouseADT, const MRoom::TRoom& room)
{
        THouseADTIntern* house = (THouseADTIntern*) HouseADT;

        // is HouseADT empty ?
        if (house->Count == 0)
                return false;

        for (Ordinal nRun=0; nRun<house->Count; nRun++)
                if (MRoom::EqualValue(house->PSet[nRun], room))
                {
                        house->Cursor = nRun;
                        return true;
                }

        // no room found, cursor remains unchanged
        return false;
}


// returns the room the cursors points to
```

```
Boolean MHouseADT::GetCurrent(UHouseADT HouseADT, MRoom::TRoom& room)
{
	THouseADTIntern* house = (THouseADTIntern*) HouseADT;

	// verify that room exists
	if ((house->Cursor < 0) ||
		(house->Cursor >= house->Count))
		return false;

	return MRoom::Copy(room, house->PSet[house->Cursor]);
}


// deletes the room the cursor points to
Boolean MHouseADT::Scratch(UHouseADT HouseADT)
{
	THouseADTIntern* house = (THouseADTIntern*) HouseADT;

	// is HouseADT empty and cursor valid ?
	if ((house->Count == 0) ||
		(house->Cursor < 0) ||
		(house->Cursor >= house->Count))
		return false;

	// move all rooms beyond HouseADT.Cursor one position ahead
	for (Ordinal nRun = house->Cursor; nRun < house->Count; nRun++)
		// verify copy
		if (!MRoom::Copy(house->PSet[nRun], house->PSet[nRun+1]))
			return false;

	// reset Cursor and Count
	house->Cursor--;
	house->Count--;

	// we're done
	return true;
}


// displays the attributes
void MHouseADT::Show(UHouseADT HouseADT)
{
	THouseADTIntern* house = (THouseADTIntern*) HouseADT;

	using namespace std;

	// display general room info
	cout<<"Das Haus besteht aus "<<house->Count<<" Zimmern."<<endl;
	cout<<"Das aktuelle Zimmer ist "<<house->Cursor<<endl;

	// foundation date
	cout<<"Das Haus wurde gebaut am ";
	MDate::Show(house->DateOfFoundation);
	cout<<endl;

	// display each room
	if (house->Count > 0)
		for (Ordinal nRun=0; nRun<house->Count; nRun++)
		{
			cout<<"Raum "<<nRun<<": ";
			MRoom::Show(house->PSet[nRun]);
		}
	else
		cout<<"Das Haus ist leer."<<endl;
}
```

M06_5.cpp:

```
////////////////////////////////////////////////////////
~
// Softwarebauelemente I, Aufgabe M6.5.
~
//
~
```

```
// author:            Stephan Brumme
~
// last changes:      January 07, 2001
~

~

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

// open some namespaces
using namespace std;
using namespace MHouseADT;

void main()
{
       MDate::TDate MyDate;
       MDate::Today(MyDate);
       MDate::Show(MyDate);


       UHouseADT MyHouse, MyHouse2;
       MRoom::TRoom MyRoom;

       NewHouseADT(&MyHouse);
       NewHouseADT(&MyHouse2);

       Show(MyHouse);

       MRoom::Init(MyRoom, 2, 30);

       Insert(MyHouse, MyRoom);
       Insert(MyHouse, MyRoom);

       Show(MyHouse);

       Copy(MyHouse2, MyHouse);

       Show(MyHouse2);

       DeleteHouseADT(&MyHouse);
       DeleteHouseADT(&MyHouse2);
}
```