Aufgabe 6.2.

Es ist zuerst notwendig einen Namespace *MDate* zu erstellen. Ich orientiere mich dabei stark an den Beispielen aus der Vorlesung. Die Basisdatentypen entsprechen denen aus Aufgabe 6.1..

PrimitiveTypes.h:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.1
//
// author:            Stephan Brumme
// last changes:      November 23, 2000

#ifndef __PRIMITIVETYPES_H__
#define __PRIMITIVETYPES_H__

// define the basic types
// names are closely related to their Pascal equivalents

typedef int   Ordinal;
typedef bool  Boolean;
typedef float Real;
typedef char* String;

#endif
```

MDate.h:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.2.
//
// author:            Stephan Brumme
// last changes:      December 29, 2000

#ifndef __MDATE_H__
#define __MDATE_H__

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


// define namespace MDate
namespace MDate
{
	// struct which holds a single date
	typedef struct
	{
		Ordinal Day;
		Ordinal Month;
		Ordinal Year;
	} TDate;


	// initializes a TDate struct with the current date
	void Today(TDate& TheDate);

	// initializes a TDate struct with a specified date
	// !!! does not verify for validity !!!
	void InitDate(TDate& TheDate, Ordinal TheDay, Ordinal TheMonth, Ordinal TheYear);

	// compares two dates
	Boolean EqualValue(const TDate& Date1, const TDate& Date2);

	// copy a date
	Boolean Copy(TDate& Date1, const TDate& Date2);

	// displays a TDate
	void Show(const TDate& TheDate);
};

#endif
```

MDate.cpp:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.2.
//
// author:            Stephan Brumme
// last changes:      December 29, 2000


// include the header file
#include "MDate.h"

// standard timer library
#include <time.h>

// needed to display data in MDate::Show(..)
#include <iostream>


// initializes a TDate struct with the current date
void MDate::Today(TDate& TheDate)
{
       // the following code is taken from MSDN

       // use system functions to get the current date as UTC
       time_t secondsSince1970;
       time(&secondsSince1970);

       // convert UTC to local time zone
       struct tm *localTime;
       localTime = localtime(&secondsSince1970);

       // store retrieved data
       TheDate.Day   = localTime->tm_mday;
       TheDate.Month = localTime->tm_mon + 1;
       TheDate.Year  = localTime->tm_year + 1900;
}


// initializes a TDate struct with a specified date
// !!! does not verify for validity !!!
void MDate::InitDate(TDate& TheDate, Ordinal TheDay, Ordinal TheMonth, Ordinal TheYear)
{
       // nothing special ...
       TheDate.Day   = TheDay;
       TheDate.Month = TheMonth;
       TheDate.Year  = TheYear;
}


// compares two dates
Boolean MDate::EqualValue(const TDate& Date1, const TDate& Date2)
{
       return ((Date1.Day   == Date2.Day  ) &&
                   (Date1.Month == Date2.Month) &&
                      (Date1.Year  == Date2.Year));
}


// copy a date
Boolean MDate::Copy(TDate& Date1, const TDate& Date2)
{
       // both dates must differ
       if (EqualValue(Date1, Date2))
              return false;

       Date1.Day   = Date2.Day;
       Date1.Month = Date2.Month;
       Date1.Year  = Date2.Year;

       // successfully done
       return true;
```

```
}


// displays a TDate
void MDate::Show(const TDate& TheDate)
{
	std::cout << TheDate.Day << "." << TheDate.Month << "." << TheDate.Year << std::endl;
}
```

Das Haus verwaltet seine Räume in einem Array aus den Datentyp *TRoom*. Dieser wiederum ist bekannt aus den vorherigen Übungen.

MRoom.h:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.1
//
// author:            Stephan Brumme
// last changes:      December 29, 2000

#ifndef __ROOM_H__
#define __ROOM_H__


#include "PrimitiveTypes.h"


// define the namespace Room
namespace MRoom
{
	// Data structure representing a room unit
	struct TRoom
	{
		Ordinal NumberOfRooms;
		Ordinal Area;
	};


	// Initializes the TRoom structure
	void Init(TRoom &roo, Ordinal nor, Ordinal ar);

	// Compares two exemplars
	// returns "true" if attributes of both are equal; "false" otherwise
	Boolean EqualValue(TRoom roo1, TRoom roo2);

	// Copies the attributes of roo2
	// returns "true" if successful, "false" if no memory allocated
	Boolean Copy(TRoom& roo1, TRoom roo2);

	// Returns the NumberOfRooms attribute
	Ordinal GetNumberOfRooms(TRoom roo);

	// Sets the NumberOfRooms attribute
	void SetNumberOfRooms(TRoom &roo, Ordinal nor);

	// Returns the Area attribute
	Ordinal GetArea(TRoom roo);

	// Displays the attributes
	void Show(TRoom roo);
}

#endif
```

MRoom.cpp:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.1
//
// author:                   Stephan Brumme
```

```
// last changes:      December 29, 2000


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

// open std namespace
using namespace std;



// define the namespace Room

// Initializes the TRoom structure
void MRoom::Init(TRoom &roo, Ordinal nor, Ordinal ar)
{
       roo.NumberOfRooms = nor;
       roo.Area = ar;
}


// Compares two exemplars
// returns "true" if attributes of both are equal; "false" otherwise
Boolean MRoom::EqualValue(TRoom roo1, TRoom roo2)
{
       return ((roo1.Area == roo2.Area) &&
                   (roo1.NumberOfRooms = roo2.NumberOfRooms));
}


// Copies the attributes of roo2
// returns "true" if successful, "false" if no memory allocated
Boolean MRoom::Copy(TRoom& roo1, TRoom roo2)
{
       if (EqualValue(roo1, roo2))
              return false;

       roo1.Area = roo2.Area;
       roo1.NumberOfRooms = roo2.NumberOfRooms;
       return true;
}


// Returns the NumberOfRooms attribute
Ordinal MRoom::GetNumberOfRooms(TRoom roo)
{
       return roo.NumberOfRooms;
}


// Sets the NumberOfRooms attribute
void MRoom::SetNumberOfRooms(TRoom &roo, Ordinal nor)
{
       roo.NumberOfRooms = nor;
}


// Returns the Area attribute
Ordinal MRoom::GetArea(TRoom roo)
{
       return roo.Area;
}


// Displays the attributes
void MRoom::Show(TRoom roo)
{
       cout<<"Es sind "<<roo.NumberOfRooms<<" Raeume mit einer Flaeche von "<<roo.Area
              <<"."<<endl;
}

// end of namespace Room
```

Die einzelnen Funktionen von *MHouse* entstammen auch vorherigen Übungen, so dass ich in der Regel mit Cut'n'Paste bekannten und getesteten Code verwende.

MHouse.h:

```
//////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.2.
//
// author:                    Stephan Brumme
// last changes:      December 29, 2000

#ifndef __MHOUSE_H__
#define __MHOUSE_H__

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

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


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

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

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

	// data struct THouse for a single house
	typedef struct
	{
		TDate   DateOfFoundation;
		Ordinal Count;
		Ordinal Cursor;
		TArrayOfRooms PSet;
	} THouse;


	// initializes the THouse structure
	void Init(THouse &house);

	// compares two exemplars
	// returns "true" if attributes of both are equal; "false" otherwise
	Boolean EqualValue(const THouse& house1, const THouse& house2);

	// copies the attributes of house2
	// returns "true" if successful, "false" if no memory allocated
	Boolean Copy(THouse& house1, const THouse& house2);

	// retrieve date of foundation
	TDate GetDateOfFoundation(const THouse& house);

	// get number of rooms
	Ordinal Card(const THouse& house);

	// add a new room to a house
	Boolean Insert(THouse& house, const TRoom& room);

	// returns the first room of a house
	Boolean GetFirst(THouse& house, TRoom& room);

	// returns the last room of a house
	Boolean GetNext(THouse& house, TRoom& room);

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

```
        // returns the room the cursors points to
        Boolean GetCurrent(const THouse& house, TRoom& room);

        // deletes the room the cursor points to
        Boolean Scratch(THouse& house);

        // displays the attributes
        void Show(THouse house);
};

#endif
```

MHouse.cpp:

```
////////////////////////////////////////////////////////
// Softwarebauelemente I, Aufgabe M6.2.
//
// author:            Stephan Brumme
// last changes:      December 29, 2000


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


using namespace MDate;


// initializes the THouse structure
void MHouse::Init(THouse &house)
{
        house.Count  = 0;
        house.Cursor = -1;
        Today(house.DateOfFoundation);
}


// compares two exemplars
// returns "true" if attributes of both are equal; "false" otherwise
Boolean MHouse::EqualValue(const THouse& house1, const THouse& house2)
{
        // 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 houses are NOT recognized !!!

        // are houses 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 houses are equal
        return true;
}


// copies the attributes of house2
// returns "true" if successful, "false" if no memory allocated
Boolean MHouse::Copy(THouse& house1, const THouse& house2)
{
        // are both houses equal ?
        if (EqualValue(house1, house2))
                return false;
```

```
        // 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 MHouse::GetDateOfFoundation(const THouse& house)
{
        TDate date;
        MDate::Copy(date, house.DateOfFoundation);

        return date;
}


// get number of rooms
Ordinal MHouse::Card(const THouse& house)
{
        return house.Count;
}


// add a new room to a house
Boolean MHouse::Insert(THouse& house, const MRoom::TRoom& room)
{
        // 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 house
Boolean MHouse::GetFirst(THouse& house, MRoom::TRoom& room)
{
        // is house empty ?
        if (house.Count == 0)
                return false;

        // set cursor
        house.Cursor = 0;

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


// returns the last room of a house
Boolean MHouse::GetNext(THouse& house, MRoom::TRoom& room)
{
        // 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(house, room);
}


// looks for a given room and sets cursor, if possible
Boolean MHouse::Find(THouse& house, const MRoom::TRoom& room)
{
	// is house 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 MHouse::GetCurrent(const THouse& house, MRoom::TRoom& room)
{
	// 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 MHouse::Scratch(THouse& house)
{
	// is house empty and cursor valid ?
	if ((house.Count == 0) ||
		(house.Cursor < 0) ||
		(house.Cursor >= house.Count))
		return false;

	// move all rooms beyond house.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 MHouse::Show(THouse house)
{
	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;
}
```