1 Organisatorisches

1.1 Team

- Reinhard Penn, s1110306019
- Bernhard Selymes, s1110306024

1.2 Aufteilung

- Reinhard Penn
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Object, Roomlayout, Room, Side, Wall, Door
  - Testen aller Klassen
- Bernhard Selymes
  - Planung
  - Klassendiagramm
  - Implementierung der Klassen Object, Roomlayout, Room, Side, Wall, Door
  - Dokumentation

1.3 Zeitaufwand

- geschätzte Mh: 7h
- tatsächlich: Reinhard (10h), Bernhard (10h)

2 Systemspezifikation

Eine Software für einen Raumplan soll entwickelt werden. Ein Raumplan enthält mehrere Räume, jeder Raum hat 4 Seiten, die wiederum eine Wand oder ein Durchgang sein können. Die Räume sind alle gleich groß und liegen alle untereinander und nicht nebeneinander. Eine Wand hat eine Farbe, ein Durchgang ist offen oder geschlossen und verbindet einen oder zwei Räume - es gibt auch Durchgänge die nach draußen führen. Wenn zwei Durchgänge aufeinander treffen wird nur einer ausgegeben. Bei zwei Wänden werden beide ausgegeben. In der gesamten Datenstruktur können nur Objekte hinzugefügt, aber nicht gelöscht werden. Der gesamte Raumplan kann ausgedruckt werden.

3 Systementwurf

3.1 Klassendiagramm

3.2 Komponentenübersicht

- Klasse ”Object”:
  Basis aller Basisklassen.
- Klasse ”RoomLayout”:
  Beinhaltet die Räume und kann diese mithilfe einer Printfunktion ausgeben.
- Klasse ”Room”:
  Beinhaltet vier Seiten und kann mithilfe einer Printfunktion ausgegeben werden.
- Klasse ”Side”:
  Abstrakte Klasse, von der Wall und Door abgeleitet werden. Speichert die Himmelsrichtung der Seite.
- Klasse ”Wall”:
  Hat einen Member der die Farbe der Wand speichert.
- Klasse ”Door”:
  Speichert ob Door offen oder geschlossen ist und welche Räume von Door verbunden werden.
- Enumeration ”Direction”:
  Hat vier Zustände: North, West, East, South

4 Komponentenentwurf

4.1 Klasse ”Object”

Abstrakte Basisklasse aller Klassen. Von ihr werden alle anderen Klassen abgeleitet. Beinhaltet einen virtuellen Destruktor.

4.2 Klasse ”RoomLayout”

Besitzt eine Liste die Zeiger auf die Klasse Room beinhaltet, einen Zeiger, der auf den vorherigen Raum zeigt und einen Wahrheitswert, der angibt ob im letzten Raum im Süden ein Durchgang war.

Methode ”Print”: Gibt die Räume die in der Liste gespeichert sind der Reihe nach aus und merkt sich immer ob im vorherigen Raum eine Durchgang im Süden war. Wenn ein Durchgang war wird der Durchgang nur einmal ausgegeben.

Methode ”AddRoom”: Fügt Räume zum Raumplan hinzu. Ein Raum wird nur dann hinzugefügt wenn die südliche Seite der vorherigen Raumes mit der nördlichen Seite des Raumes, der eingefügt werden soll, übereinstimmt oder der gesamte Raumplan noch leer ist. Wenn ein Durchgang war: Im vorherigen Raum wird bei Durchgang der aktuelle Raum hinzugefügt und beim Durchgang vom aktuellen Raum wird der vorherigen Raum gespeichert. Weiters wird gespeichert ob der aktuelle Raum einen Durchgang im Süden hat und der Zeiger, der auf den vorherigen Raum gezeigt hat, zeigt jetzt auf den aktuellen Raum.

4.3 Klasse ”Room”

Beinhaltet einen Vektor der Zeiger auf die Klasse Side hat.

Methode ”Print”: Gibt den Raum aus. Durchgang wird im Norden nicht ausgegeben, wenn im letzten Raum im Süden ein Durchgang war. Die restlichen Seiten werden ganz normal ausgegeben, eine Wand im Norden oder Süden wird durch 9 Sterne gekennzeichnet, im Westen oder Osten durch 5. Ein Durchgang wird durch ein ”D” gekennzeichnet.

Methode ”AddSide”: Fügt Seiten zu einem Raum hinzu. Eine Seite wird nur dann hinzugefügt, wenn noch keine Seite für diese Himmelsrichtung existiert. Der Vektor mit den Seiten wird nach dem Einfügen nach der Definiton der Enumeration sortiert: North, West, East, South. Wenn eine Seite ein Durchgang ist, wird im Durchgang der aktuelle Raum hinzugfügt.

4.4 Klasse ”Side”

Abstrakte Klasse, die eine Seite eines Raumes repräsentiert. Enthält einen Member der die ”Direction” der Seite speichert und einen der angibt ob es ein Durchgang oder eine Wand ist.

4.5 Klasse ”Wall”

Repräsentiert eine Wand. Hat einen Konstruktor dem die Farbe und die Richtung der Wand übergeben werden.

4.6 Klasse ”Door”

Repräsentiert einen Durchgang. Hat einen Member der speichert ob die Tür offen oder geschlossen ist und einen Vektor der die Räume speichert, die der Durchgang verbindet.

Methode ”AddRoom”: Fügt zum Durchgang einen Raum hinzu, wenn noch nicht die maximale Anzahl der Räume die ein Durchgang verbinden kann erreicht ist.

4.7 Enumeration ”Direction”

- North
- West
- East
- South

5 Source Code

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Object.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 29.10.2012
5  // Description : Header for Object.cpp
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10
11 #ifndef OBJECT_H
12 #define OBJECT_H
13
14 class Object
15 {
16 public:
17    //virtual Destructor for baseclass
18    virtual ~Object();
19 protected:
20    //Default CTor for baseclass
21    Object();
22 };
23
24 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Object.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Baseclass with protected constructor
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////////


#include "Object.h"

Object::Object()
{}

Object::~Object()
{}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : RoomLayout.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 29.10.2012
5  // Description : Header for RoomLayout.cpp
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10
11 #ifndef ROOMLAYOUT_H
12 #define ROOMLAYOUT_H
13
14 #include <list>
15 #include "Object.h"
16 #include "Room.h"
17
18 typedef std::list<Room*> Rooms;
19 typedef Rooms::const_iterator RoomsItor;
20
21 class RoomLayout :
22    public Object
23 {
24 public:
25    //Default CTor
26    RoomLayout();
27
28    //virtual destructor
29    virtual ~RoomLayout();
30
31    //Prints the rooms in the layout
32    void Print() const;
33
34    //Adds a room to the layout
35    bool AddRoom(Room* room);
36
37 private:
38    Rooms mRooms;
39    Room* prevRoom;
40    bool mWasDoor;
41
42    //Prints one specific room of the layout
43    void PrintRoom(Room* room, bool WasDoor) const;
44 };
45
46 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : RoomLayout.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Implements the class RoomLayout
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////


#include <iostream>
#include <algorithm>
#include <iterator>
#include <string>
#include "RoomLayout.h"


RoomLayout::RoomLayout()
   : mWasDoor(false)
{
}

RoomLayout::~RoomLayout()
{
   RoomsItor itor = mRooms.begin();

   while (itor != mRooms.end())
   {
      delete (*itor);
      ++itor;
   }
}

void RoomLayout::PrintRoom(Room* room, bool WasDoor) const
{
   room->Print(WasDoor);
}

void RoomLayout::Print() const
{
   bool WasDoor = false;

   for (std::list<Room*>::const_iterator first = mRooms.begin() ; first!=
       mRooms.end(); ++first )
   {
      PrintRoom(*first, WasDoor);
      WasDoor = (*first)->IsSouthDoor();
   }
}

bool RoomLayout::AddRoom(Room* room)
{
   try
   {
      if (!room->IsFull())
      {
         std::string ex("Room is not full");
         throw(ex);
      }
      if (mRooms.empty() || mWasDoor == room->IsNorthDoor())
```

```
60          {
61             mRooms.push_back(room);
62
63             if (mWasDoor)
64             {
65                Side* CurrentDoor = room->GetNorthSide();
66                Side* PrevDoor = prevRoom->GetSouthSide();
67
68                prevRoom->AddRoomToDoor(CurrentDoor);
69                room->AddRoomToDoor(PrevDoor);
70             }
71
72             mWasDoor = room->IsSouthDoor();
73             prevRoom = room;
74             return true;
75          }
76          return false;
77       }
78       catch (std::string const& ex)
79       {
80          std::cerr << "Error occured in RoomLayout::AddRoom: " << ex << std::
               endl;
81          return false;
82       }
83       catch (...)
84       {
85          std::cerr << "Unknown Error in RoomLayout::AddRoom" << std::endl;
86          return false;
87       }
88    }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Room.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 29.10.2012
5  // Description : Header for Room.cpp
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10
11 #ifndef ROOM_H
12 #define ROOM_H
13
14 #include <vector>
15 #include <string>
16 #include "Object.h"
17 #include "Side.h"
18
19
20 typedef std::vector<Side*> TVec;
21 typedef TVec::const_iterator TVecItor;
22
23 std::string const WallString_N = "*********";
24 std::string const DoorString_N = "****D****";
25 std::string const SideSpaces = "       ";
26 char const WallSign = '*';
27 char const DoorSign = 'D';
28
29 int const MaxWalls = 4;
30
31 class Room :
32    public Object
33 {
34 public:
35    //Default CTor
36    Room();
37
38    //virtual destructor
39    virtual ~Room();
40
41    //Prints the four sides of the room
42    void Print(bool WasDoor) const;
43
44    //Adds a side to the room
45    bool AddSide(Side* side);
46    //Adds this room to a certain door
47    void AddRoomToDoor(Side* door);
48
49    //Checks if the room has reached the max amount of walls
50    bool IsFull() const;
51    //Checks if the north side is a door
52    bool IsNorthDoor() const;
53    //Checks if the south side is a door
54    bool IsSouthDoor() const;
55
56    //Returns the north side
57    Side* GetNorthSide() const;
58    //Returns the south side
59    Side* GetSouthSide() const;
60
```

```
61 private:
62    std::vector<Side*> mSides;
63 
64    //Prints a horizontal side
65    void PrintWallOrDoorNS(Side* side) const;
66    //Prints the sign of a vertical side
67    void PrintWallOrDoorOW(Side* side) const;
68    //Prints the parts of two parallel walls
69    void PrintTwoWallParts() const;
70    //Prints the Spaces between two vertical sides
71    void PrintSpaces() const;
72 };
73 
74 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Room.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Implements the class room
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////


#include <algorithm>
#include <iterator>
#include <vector>
#include <iostream>
#include <string>
#include "Room.h"
#include "Door.h"


Room::Room()
{
}

Room::~Room()
{
   TVecItor itor = mSides.begin();

   while (itor != mSides.end())
   {
      delete (*itor);
      ++itor;
   }
}

void Room::PrintWallOrDoorNS(Side* side) const
{
   if (!side->IsDoor())
   {
      std::cout << WallString_N << std::endl;
   }
   else
   {
      std::cout << DoorString_N << std::endl;
   }
}

void Room::PrintWallOrDoorOW(Side* side) const
{
   if (!side->IsDoor())
   {
      std::cout << WallSign;
   }
   else
   {
      std::cout << DoorSign;
   }
}

void Room::PrintTwoWallParts() const
{
```

```
61     std::cout << WallSign;
62     PrintSpaces();
63     std::cout << WallSign << std::endl;
64  }
65
66  void Room::PrintSpaces() const
67  {
68     std::cout << SideSpaces;
69  }
70
71  void Room::Print(bool WasDoor) const
72  {
73     try
74     {
75        //Checks if the room has 4 Sides
76        if (mSides.size() != MaxWalls) {
77           std::string ex("Sides are missing");
78           throw(ex);
79        }
80
81        TVecItor itor = mSides.begin();
82
83        //North
84        if (!WasDoor)
85        {
86           PrintWallOrDoorNS(*(itor));
87        }
88        ++itor;
89
90        //West+East
91        PrintTwoWallParts();
92        PrintWallOrDoorOW((*itor));
93        PrintSpaces();
94        ++itor;
95        PrintWallOrDoorOW((*itor));
96        std::cout << std::endl;
97        PrintTwoWallParts();
98        ++itor;
99
100       //South
101       PrintWallOrDoorNS(*(itor));
102    }
103    catch (std::string const& ex)
104    {
105       std::cerr << "Error occured in Room::Print: " << ex << std::endl;
106    }
107    catch (...)
108    {
109       std::cerr << "Unknown Error in Room::Print" << std::endl;
110    }
111 }
112
113 bool CheckSideOrder(Side* side1, Side* side2)
114 {
115    return (side1->getDirection()) < (side2->getDirection());
116 }
117
118 bool Room::AddSide(Side* side)
119 {
120    try
```

```
121     {
122        //checks if max amount of walls have been reached
123        if (mSides.size() >= MaxWalls)
124        {
125           std::string ex("Max amount of walls already reached");
126           throw(ex);
127        }
128        //only inserts walls which doesn't already exist
129        for (TVecItor itor = mSides.begin(); itor != mSides.end(); ++itor)
130        {
131           if ((*itor)->getDirection() == side->getDirection())
132           {
133              return false;
134           }
135        }
136        mSides.push_back(side);
137        //Sorts the Sides
138        std::sort(mSides.begin(),mSides.end(),CheckSideOrder);
139
140        if (side->IsDoor())
141        {
142           AddRoomToDoor(side);
143        }
144        return true;
145     }
146     catch (std::string const& ex)
147     {
148        std::cerr << "Error occured in Room::AddSide: " << ex << std::endl;
149        return false;
150     }
151     catch (...)
152     {
153        std::cerr << "Unknown Error in Room::AddSide" << std::endl;
154        return false;
155     }
156  }
157
158  bool Room::IsNorthDoor() const
159  {
160     return mSides[0]->IsDoor();
161  }
162
163  bool Room::IsSouthDoor() const
164  {
165     return mSides[MaxWalls-1]->IsDoor();
166  }
167
168  void Room::AddRoomToDoor(Side* side)
169  {
170     Door* door = dynamic_cast<Door*>(side);
171     door->AddRoom(this);
172  }
173
174  Side* Room::GetNorthSide() const
175  {
176     return mSides[0];
177  }
178
179  Side* Room::GetSouthSide() const
180  {
```

```
181     return mSides[MaxWalls-1];
182 }
183
184 bool Room::IsFull() const
185 {
186     return mSides.size() == MaxWalls;
187 }
```

```
1  ////////////////////////////////////////////////////////////////////////////
2  // Workfile : Side.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 29.10.2012
5  // Description : Header for Side.cpp
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////////
9
10
11 #ifndef SIDE_H
12 #define SIDE_H
13
14 #include "Direction.h"
15 #include "Object.h"
16
17
18 class Side :
19    public Object
20 {
21 public:
22    //virtual destructor
23    virtual ~Side();
24
25    //Checks if the current object is a door
26    bool IsDoor() const;
27
28    //Gets the direction of the current side
29    Direction getDirection() const;
30
31 protected:
32    bool mIsDoor;
33    Direction mDirection;
34 };
35
36 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Side.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Base functionality for wall and door
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////////


#include "Side.h"


Side::~Side()
{
}

Direction Side::getDirection() const
{
   return mDirection;
}

bool Side::IsDoor() const
{
   return mIsDoor;
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Wall.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Header for Wall.cpp
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////


#ifndef WALL_H
#define WALL_H

#include <string>
#include "Side.h"


class Wall :
   public Side
{
public:
   //CTor with color and direction
   Wall(std::string color, Direction direction);

   //virtual destructor
   virtual ~Wall();

private:
   std::string mColor;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Wall.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Implementation for a wall
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////


#include <iostream>
#include "Wall.h"


Wall::Wall(std::string color, Direction direction)
{
   mColor = color;
   mDirection = direction;
   mIsDoor = false;
}

Wall::~Wall()
{
}
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Door.h
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Header for Door.cpp
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////


#ifndef DOOR_H
#define DOOR_H

#include <vector>
#include "Side.h"
#include "Room.h"


int const MaxRooms = 2;

class Door :
   public Side
{
public:
   //CTor with parameters
   Door(bool isOpen, Direction direction);

   //virtual destructor
   virtual ~Door();

   //Adds a room to the current door if MaxRooms isn't
   //reached yet
   void AddRoom(Room* room);

private:
   bool mIsOpen;
   std::vector<Room*> mRooms;
};

#endif
```

```cpp
////////////////////////////////////////////////////////////////////////////
// Workfile : Door.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Implemantation for a door, with the possibility to connect
//           two rooms to it.
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////////


#include "Door.h"


Door::Door(bool isOpen, Direction direction)
{
   mIsDoor = true;
   mIsOpen = isOpen;
   mDirection = direction;
}

Door::~Door()
{
}

void Door::AddRoom(Room* room)
{
   if (mRooms.size() < MaxRooms)
   {
      mRooms.push_back(room);
   }
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile : Direction.h
3  // Author : Reinhard Penn, Bernhard Selymes
4  // Date : 29.10.2012
5  // Description : Enumeration type for direction
6  // Remarks : -
7  // Revision : 0
8  ////////////////////////////////////////////////////////////////////////
9
10
11 #ifndef DIRECTION_H
12 #define DIRECTION_H
13
14 enum Direction
15 {
16    North,
17    West,
18    East,
19    South
20 };
21
22 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile : Main.cpp
// Author : Reinhard Penn, Bernhard Selymes
// Date : 29.10.2012
// Description : Testdriver
// Remarks : -
// Revision : 0
////////////////////////////////////////////////////////////////////////


#include <iostream>
#include "Room.h"
#include "Wall.h"
#include "Door.h"
#include "RoomLayout.h"


//empty Roomlayout
void testcase0()
{
   std::cout << "Testcase0: Empty roomlayout" << std::endl;

   RoomLayout rl1;

   rl1.Print();

   std::cout << std::endl << std::endl;
}

//Roomlayout with empty Rooms
void testcase1()
{
   std::cout << "Testcase1: Roomlayout with empty rooms" << std::endl;

   RoomLayout rl1;

   Room* r1 = new Room;
   Room* r2 = new Room;

   rl1.AddRoom(r1);
   rl1.AddRoom(r2);

   rl1.Print();

   delete r1;
   delete r2;

   std::cout << std::endl << std::endl;
}

//Roomlayout with wrong Rooms
void testcase2()
{
   std::cout << "Testcase2: Roomlayout with wrong rooms" << std::endl;

   RoomLayout rl1;

   Room* r1 = new Room;
   Room* r2 = new Room;

```

```
61      Side* s1 = new Wall("Green",North);
62      Side* s2 = new Wall("Blue",West);
63      Side* s3 = new Door(true,East);
64      Side* s4 = new Door(false,South);
65
66      Side* s5 = new Wall("Green",North);
67      Side* s6 = new Wall("Blue",West);
68      Side* s7 = new Door(true,East);
69      Side* s8 = new Door(false,South);
70
71      r1->AddSide(s3);
72      r1->AddSide(s1);
73      r1->AddSide(s2);
74      r1->AddSide(s4);
75
76      r2->AddSide(s6);
77      r2->AddSide(s8);
78      r2->AddSide(s7);
79      r2->AddSide(s5);
80
81
82      rl1.AddRoom(r1);
83      rl1.AddRoom(r2);
84
85      rl1.Print();
86
87      delete r2;
88
89      std::cout << std::endl << std::endl;
90  }
91
92  //Roomlayout with correct Rooms
93  void testcase3()
94  {
95      std::cout << "Testcase3: Roomlayout with correct rooms" << std::endl;
96
97      RoomLayout rl1;
98
99      Room* r1 = new Room;
100     Room* r2 = new Room;
101
102     Side* s1 = new Wall("Green",North);
103     Side* s2 = new Wall("Blue",West);
104     Side* s3 = new Door(true,East);
105     Side* s4 = new Door(false,South);
106
107     Side* s5 = new Door(true,North);
108     Side* s6 = new Wall("Blue",West);
109     Side* s7 = new Door(true,East);
110     Side* s8 = new Door(false,South);
111
112     r1->AddSide(s3);
113     r1->AddSide(s1);
114     r1->AddSide(s2);
115     r1->AddSide(s4);
116
117     r2->AddSide(s6);
118     r2->AddSide(s8);
119     r2->AddSide(s7);
120     r2->AddSide(s5);
```

```
121
122
123     rl1.AddRoom(r1);
124     rl1.AddRoom(r2);
125
126     rl1.Print();
127
128     std::cout << std::endl << std::endl;
129 }
130
131
132 //Roomlayout with correct rooms
133 //Trying to add a fifth side to one room
134 void testcase4()
135 {
136     std::cout << "Testcase4: Roomlayout with correct rooms." << std::endl
137         << "Trying to add a fifth side to one room" << std::endl;
138
139     RoomLayout rl1;
140
141     Room* r1 = new Room;
142     Room* r2 = new Room;
143
144     Side* s1 = new Wall("Green",North);
145     Side* s2 = new Wall("Blue",West);
146     Side* s3 = new Door(true,East);
147     Side* s4 = new Wall("Black",South);
148
149     Side* s5 = new Wall("White",North);
150     Side* s6 = new Wall("Blue",West);
151     Side* s7 = new Door(true,East);
152     Side* s8 = new Door(false,South);
153     Side* s9 = new Door(false,North);
154
155     r1->AddSide(s3);
156     r1->AddSide(s1);
157     r1->AddSide(s2);
158     r1->AddSide(s4);
159
160     r2->AddSide(s6);
161     r2->AddSide(s8);
162     r2->AddSide(s7);
163     r2->AddSide(s5);
164     r2->AddSide(s9);
165
166
167     rl1.AddRoom(r1);
168     rl1.AddRoom(r2);
169
170     rl1.Print();
171
172     delete s9;
173
174     std::cout << std::endl << std::endl;
175 }
176
177
178 int main()
179 {
180     testcase0();
```

```
181     testcase1();
182     testcase2();
183     testcase3();
184     testcase4();
185 
186     return 0;
187 }
```

6 Testausgaben

```
Testcase0: Empty roomlayout


Testcase1: Roomlayout with empty rooms
Error occured in RoomLayout::AddRoom: Room is not full
Error occured in RoomLayout::AddRoom: Room is not full


Testcase2: Roomlayout with wrong rooms
*********
*       *
*       D
*       *
****D****


Testcase3: Roomlayout with correct rooms
*********
*       *
*       D
*       *
****D****
*       *
*       D
*       *
****D****


Testcase4: Roomlayout with correct rooms.
Trying to add a fifth side to one room
Error occured in Room::AddSide: Max amount of walls already reached
*********
*       *
*       D
*       *
*********
*********
*       *
*       D
*       *
****D****
```