1 Organisatorisches

1.1 Team

- Patrick Felixberger, s1110306005
- Johannes Selymes, s1110306025

1.2 Aufteilung

- Patrick Felixberger
  - Planung
  - Implementierung und Testen der Klassen Room, Roomplan
  - Klassendiagramm
- Johannes Selymes
  - Planung
  - Implementierung und Testen der Klassen Side, Wall, Entry
  - Dokumentation

1.3 Aufwand

- geschätzt: 12h
- tatsächlich: Johannes (5h), Patrick (6h)

2 Systemspezifikation

Es soll eine Software für die Verwaltung von Räumen erstellt werden. Es soll einen Raumplan geben, der mehrere Räume beinhalten kann. Die Räume werden in dem Raumplan nur vertikal angeordnet, jeder Raum hat also maximal 2 Nachbarn. Jeder Raum hat 4 Seiten die jeweils eine Wand oder ein Durchgang sein können. Wenn zwei Räume aneinander grenzen müssen die angrenzenden Seiten den gleichen Typ haben. Eine Wand hat eine Farbe; ein Durchgang ist offen oder zu und gibt die angrenzenden Räume bekannt.
Der Raumplan kann Räume hinzufügen, sie aber nicht mehr entfernen. Wenn ein Raum nicht passend ist (aufgrund falsches Seiten Typs) wird er nicht hinzugefügt und es wird ein Fehler ausgegeben. Der Raumplan kann auch alle Räume angeordnet ausdrucken.

3 Systementwurf

3.1 Klassendiagramm

...

3.2 Komponentenübersicht

- Klasse Roomplan: Verwaltet mehrere Rooms und kann diese mit Print() ausgeben.
- Klasse Room: Enthält Zeiger auf vier Sides , die jeweils Wall oder Entry sein können
- Klasse Side: Abstrakte Basisklasse für Wall und Entry
- Klasse Wall: Enthält einen Member der die Farbe speichert
- Klasse Entry: Enthält Zustand (offen, geschlossen) und Id der angrenzenden Räume

4 Komponentenentwurf

4.1 Klasse Objekt

Basis aller Basisklassen. Enthält einen virtuellen Destruktor und die enum Typen:

- Typ - Entweder WALL oder ENTRY
- Colour - Verschiedene Farben: WHITE, GREEN, BLUE, RED
- Dir - Enthält Himmelsrichtungen: NORTH, SOUTH, EAST, WEST

4.2 Klasse Roomplan

Klasse zu Verwaltung aller Räume. Enthält einen Vector der alle Räume speichert und einen Zähler der mit zählt wie viele Räume schon erstellt wurden. IDs der Räume beginnen bei 0.

Methoden:
void Print() const
Diese Funktion gibt die enthaltenen Räume untereinander aus. Für Wände werden Sternchen ausgegeben und für Durchgänge Sternchen mit DD dazwischen. Wenn zwei Durchgänge aneinanderschließen, wird nur einer ausgegeben.

void Roomplan::Add(Typ n, Typ s, Typ e, Typ w, Colour const & col, bool open)
Diese Funktion bekommt die Typen der vier Wände übergeben. Es wird überprüft, ob der Raum angefügt werden kann. Wenn dies möglich ist, wird der Raum mittels push_back in den vector eingefügt, sonst wird ein Fehler auf die Konsole ausgegeben. Bei dieser Funktion haben alle Wände die gleiche Farbe und die Durchgänge sind alle offen oder geschlossen.

void Roomplan::Add(Side* NS, Side* SS, Side* ES, Side* WS)
Diese Funktion bekommt die fertigen Side* Objekte übergeben, die entweder Wall oder Entry sein können. Es wird auch wieder überprüft, ob der Raum zum darüberliegenden passt.

int GetCounter () const
Diese Funktion gibt den derzeitigen Raumzähler zurück. Der Raumzähler zählt mit wieviele Räume schon erstellt wurden.

4.3 Klasse Room

Klasse die vier Zeiger auf Side (bzw. Wall oder Entry) hat, die die Seiten des Raumes darstellen. Hat auch einen Memeber mId welcher die Id des Raums angibt.

Konstruktoren:
Room(Typ n, Typ s, Typ e, Typ w, size_t id, Colour const &col, bool const &open)
Dieser Konstruktor bekommt die Typen der vier Wände, die Id, die Farbe für alle Wände und den Zustand der Durchgänge. Je nach dem ob die Seite eine Wand oder ein Durchgang ist, wird die entsprechende mit new erstellt.

Room::Room(Side* NS, Side* SS, Side* ES, Side* WS, size_t id)
Hier werden die Seiten als Zeiger übergeben. Hier müssen die Objekte, die hinter den Zeigern stehen, kopiert werden. Das wird mithilfe der Funktion Clone() erreicht, welche bei der Klasse

Side, Wall und Entry noch genauer erklärt wird. Wenn man die Zeiger einfach normal zuweisen würde, wäre es eine shallow-copy.

Copy Konstruktor:
Room::Room (Room const& room)
Hier werden die Zeiger auf die Side* Objekte auch mittels Clone kopiert.

Destruktor:
Löscht die Zeiger auf die Seiten und setzt sie zu 0.

Get Methoden:
Die Get...Wall() Methoden geben den Typ der jeweiligen Seite zurück.
Die Funktion GetRoomId() gibt die Id zurück.

void Room::PrintRoomInfo () const
Diese Funktion gibt die Id und die Information der einzelnen Seiten aus. Ruft PrintInfo() von Side auf, welches dort näher beschrieben wird.

4.4 Klasse Side

Die Klasse Side ist eine abstrakte Basisklasse für die Klassen Wall und Entry. Enthält nur pure-virtual Funktion, kann also auch nicht implementiert werden.

Konstruktoren:
Side () {}
Default Konstruktor.
Side (Typ typ) : mTyp(typ) {}
Direktes Zuweisen des Typs.
pure virtual Methoden:
virtual Typ GetTyp () const = 0;
Gibt Typ der Seite zurück (WALL/ENTRY).
virtual void PrintInfo () const = 0;
Gibt Infos über die abgeleitete Klasse aus.
virtual Side *Clone() = 0;
Klont die abgeleitete Klasse.

4.5 Klasse Wall

Die Klasse Wall repräsentiert eine Wand. Ist abgeleitet von Side und implementiert dessen Methoden.

Konstruktoren:
Wall () : mColour(WHITE), Side(WALL) {}
Default Konstruktor.
Wall (Colour col) : mColour(col), Side(WALL) {}
Konstruktor der die Farbe übergeben bekommt und diese richtig setzt.
Wall (Wall const& wall)
Copy - Konstuktor. Kopiert Typ und Colour vom übergebenen Objekt.

Get Methoden:
GetTyp () gibt Typ zurück.
GetCoulour () gibt Farbe zurück.
PrintInfo () gibt Typ und Farbe auf der Konsole aus.
Clone Methode:
Wird verwendet um ein Objekt der Klasse Wall zu kopieren. Erstellt eine neues Objekt Wall und gibt den Zeiger darauf zurück.

4.6 Klasse Entry

Die Klasse Entry repräsentiert einen Durchgang. Ist abgeleitet von Side und implementiert dessen Methoden.
Konstruktoren:
Entry (int currentId, bool opened, Dir const& dir)
Dieser Konstruktor bekommt die derzeitige Id des Raumplans, ob der Durchgang geöffnet ist und die Richtung übergeben.
Typ, Id und Opened werden entsprechend gesetzt. Wenn der Durchgang nach Norden zeigt, wird die Id des angrenzenden Raums auf id - 1 gesetzt, wenn er nach Süden zeigt auf id + 1 und sonst auf -1 also der Durchgang geht nach aussen.
Entry::Entry (Entry const& entry)
Copy Konstruktor: Kopiert Member vom übergebenen Objekt.
Get Methoden und Print
Typ GetTyp () const;
bool IsOpened () const;
int GetMyRoomId () const;
int GetOtherRoomId () const;
void PrintInfo () const;
Geben die jeweiligen Informationen zurück bzw. druckt sie auf der Konsole aus.
Clone Methode:
Wird verwendet um ein Objekt der Klasse Entry zu kopieren. Erstellt eine neues Objekt Entry und gibt den Zeiger darauf zurück.

5 Testprotokollierung

6 Source Code

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile    : Roomplan.h
3  // Author      : Patrick Felixberger
4  // Date        : 17. 10. 2012
5  // Description : class Roomplan.
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////////////////////////////////////
9  #ifndef ROOMPLAN_H
10 #define ROOMPLAN_H
11
12 #include <iostream>
13 #include <vector>
14
15 #include "Objekt.h"
16 #include "Room.h"
17
18
19 /**
20  * class Roomplan.
21  * Roomplan contains all created Rooms and organize it.
22  */
23 class Roomplan: public Objekt {
24 public:
25    Roomplan () : mCount(0) {}
26
27    //Roomplan();
28    virtual ~Roomplan();
29
30    /**
31     * Public member with no arguments which returns void.
32     */
33    void Print() const;
34
35    /**
36     * Public member with 4 arguments which returns void.
37     * @param n typ of northwall
38     * @param s typ of southwall
39     * @param e typ of eastwall
40     * @param w typ of westwall
41     */
42    void Add(Typ n, Typ s, Typ e, Typ w, Colour const &col = WHITE, bool
          open = false);
43
44    void Add(Side* NS, Side* SS, Side* ES, Side* WS);
45
46    int GetCounter () const;
47 private:
48    std::vector<Room> mList;
49    size_t mCount;
50
51 };
52
53
54
55 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile    : Roomplan.cpp
// Author      : Patrick Felixberger
// Date        : 17. 10. 2012
// Description : class Roomplan.
// Remarks     : -
// Revision    : 0
////////////////////////////////////////////////////////////////////////
#include <iostream>

#include "Roomplan.h"
#include <algorithm>

using namespace std;

//size_t Roomplan::mCount = 0;

//Roomplan::Roomplan() {}
Roomplan::~Roomplan() {
   mList.clear();
}

//void PrintSideInfo (Side* const side) {
// side->PrintInfo();
//}

void PrintRoomInfoFunktor (Room const& room) {
   cout << "RAUM ----------" << endl;
   room.PrintRoomInfo();
   cout << endl << endl;
}


void Roomplan::Print() const {
   cout << endl << "Printing Roomplan:" << endl;
   for(size_t j = 0; j < mList.size(); j++) {
      if(mList[j].GetNorthWall() == WALL) cout << "**********" << endl;
      else if (j == 0) {
         cout << "****DD****" << endl;
      }

      for(int i = 0; i < 10; i++) {
         if(mList[j].GetWestWall() == WALL) {
            cout << "*        ";
         } else {
            (i == 4 || i == 5) ? (cout << "D") : (cout << "*");
            cout << "        ";
         }
         if(mList[j].GetEastWall() == WALL) {
            cout << "*" << endl;
         } else {
            (i == 4 || i == 5) ? (cout << "D") : (cout << "*");
            cout << endl;
         }
      }

      if(mList[j].GetSouthWall() == WALL) cout << "**********" << endl;
      else cout << "****DD****" << endl;
   }

```

```
61     for_each (mList.begin(),mList.end(),PrintRoomInfoFunktor);
62
63  }
64
65  void Roomplan::Add(Typ n, Typ s, Typ e, Typ w, Colour const &col, bool open
       ) {
66     if(!mList.empty()) {
67        if(mList[mCount-1].GetSouthWall() == n) {
68           Room tmp(n, s, e, w, mCount++, col, open);
69           mList.push_back(tmp);
70        } else {
71           cout << "Roomplan::ADD(): ERROR, can't add a WALL and ENTRY!" <<
                endl;
72           cout << "Must be WALL-WALL or ENTRY-ENTRY!" << endl;
73        }
74     } else {
75        Room tmp(n, s, e, w, mCount++, col, open);
76        mList.push_back(tmp);
77     }
78  }
79
80
81
82  void Roomplan::Add(Side* NS, Side* SS, Side* ES, Side* WS) {
83     if(!mList.empty()) {
84        //wenn
85        if(mList[mCount-1].GetSouthWall() == NS->GetTyp()) {
86           mList.push_back(Room(NS,SS,ES,WS,mCount++));
87        } else {
88           cout << "Roomplan::ADD(): ERROR, can't add a WALL and ENTRY!" <<
                endl;
89           cout << "Must be WALL-WALL or ENTRY-ENTRY!" << endl;
90        }
91     } else {
92        mList.push_back(Room(NS,SS,ES,WS,mCount++));
93     }
94  }
95
96
97
98  int Roomplan::GetCounter () const{
99     return mCount;
100 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile    : Room.h
3  // Author      : Patrick Felixberger
4  // Date        : 17. 10. 2012
5  // Description : class Room.
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////////////////////////////////////
9  #ifndef ROOM_H
10 #define ROOM_H
11
12 #include <iostream>
13 #include "Objekt.h"
14 #include "Side.h"
15 #include "Wall.h"
16 #include "Entry.h"
17
18
19 /**
20  * class Room.
21  * Creates a Room which helds 4 Walls/Entrys
22  */
23 class Room: public Objekt {
24 public:
25    /**
26     * CTor with 7 arguments
27     * @param n typ of northwall
28     * @param s typ of southwall
29     * @param e typ of eastwall
30     * @param w typ of westwall
31     * @param id ID of the room
32     * @param col Colour of walls if any exist
33     * @param open state of the entrys if any exist
34     */
35    Room(Typ n, Typ s, Typ e, Typ w, size_t id, Colour const &col, bool
          const &open);
36
37    /**
38     * CTor with 5 arguments
39     * @param NS typ of northwall
40     * @param SS typ of southwall
41     * @param ES typ of eastwall
42     * @param WS typ of westwall
43     * @param id ID of the room
44     */
45    Room(Side* NS, Side* SS, Side* ES, Side* WS, size_t id);
46
47    //copy ctor
48    Room (Room const& room);
49
50    //Destruktor um Objekt zu zerstoeren
51    ~Room();
52
53
54    /**
55     * Public member with no arguments which returns an enum.
56     * @return The typ of the northwall.
57     */
58    Typ GetNorthWall() const;
59
```

```
60      /**
61       * Public member with no arguments which returns an enum.
62       * @return The typ of the southwall.
63       */
64      Typ GetSouthWall() const;
65
66      /**
67       * Public member with no arguments which returns an enum.
68       * @return The typ of the eastwall.
69       */
70      Typ GetEastWall() const;
71
72      /**
73       * Public member with no arguments which returns an enum.
74       * @return The typ of the westwall.
75       */
76      Typ GetWestWall() const;
77
78      size_t GetRoomId () const;
79
80      /**
81       * Public member with no arguments.
82       * Shows information of room.
83       */
84      void PrintRoomInfo () const;
85
86
87
88  private:
89
90      /**
91       * Private variable.
92       * Points to the northwall
93       */
94      Side *mN;
95
96      /**
97       * Private variable.
98       * Points to the southwall
99       */
100     Side *mS;
101
102     /**
103      * Private variable.
104      * Points to the eastwall
105      */
106     Side *mE;
107
108     /**
109      * Private variable.
110      * Points to the westwall
111      */
112     Side *mW;
113
114     /**
115      * Private variable.
116      * ID number of the room.
117      */
118     size_t mID;
119 };
```

```
120
121
122 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile    : Room.cpp
// Author      : Patrick Felixberger
// Date        : 17. 10. 2012
// Description : class Room.
// Remarks     : -
// Revision    : 0
////////////////////////////////////////////////////////////////////////
#include <iostream>

#include "Room.h"


using namespace std;

Room::Room(Typ n, Typ s, Typ e, Typ w, size_t id, Colour const &col, bool
    const &open) {
   try {
      if(n == WALL) {
         mN = new Wall(col);
      } else {
         mN = new Entry(id,open,NORTH);
      }

      if(s == WALL) {
         mS = new Wall(col);
      } else {
         mS = new Entry(id,open,SOUTH);
      }

      if(e == WALL) {
         mE = new Wall(col);
      } else {
         mE = new Entry(id,open,EAST);
      }

      if(w == WALL) {
         mW = new Wall(col);
      } else {
         mW = new Entry(id,open,WEST);
      }


      mID = id;
   }
   catch(...) {
      cout << "Room::Room(): Error while creating new walls!" << endl;
   }
}

Room::Room(Side* NS, Side* SS, Side* ES, Side* WS, size_t id) {
   try {

      mN = NS->Clone();
      mS = SS->Clone();
      mE = ES->Clone();
      mW = WS->Clone();

      mID = id;
   }
```

```
60     catch(...) {
61        cout << "Room::Room(): Error while creating new walls!" << endl;
62     }
63  }
64
65  Room::Room (Room const& room) {
66     mN = room.mN->Clone();
67     mS = room.mS->Clone();
68     mE = room.mE->Clone();
69     mW = room.mW->Clone();
70
71     mID = room.GetRoomId();
72  }
73
74  Room::~Room() {
75     delete mN; mN = 0;
76     delete mS; mS = 0;
77     delete mE; mE = 0;
78     delete mW; mW = 0;
79  }
80
81
82
83
84
85  Typ Room::GetNorthWall() const {
86     return mN->GetTyp();
87  }
88
89  Typ Room::GetSouthWall() const {
90     return mS->GetTyp();
91  }
92
93  Typ Room::GetEastWall() const {
94     return mE->GetTyp();
95  }
96
97  Typ Room::GetWestWall() const {
98     return mW->GetTyp();
99  }
100
101 size_t Room::GetRoomId () const {
102    return mID;
103 }
104
105
106 void Room::PrintRoomInfo () const {
107    std::cout << "ID: " << mID << std::endl;
108    std::cout << "Nordseite: " << std::endl;
109    mN->PrintInfo();
110    std::cout << "Suedseite: " << std::endl;
111    mS->PrintInfo();
112    std::cout << "Ostseite: " << std::endl;
113    mE->PrintInfo();
114    std::cout << "Westseite: " << std::endl;
115    mW->PrintInfo();
116 }
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile    : Side.h
3  // Author      : Johannes Selymes
4  // Date        : 25. 10. 2012
5  // Description : Side of a room.
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef SIDE_H
11 #define SIDE_H
12
13 #include "Objekt.h"
14
15 /**
16  * class Side.
17  * Abstract base class for Wall and Entry
18  */
19 class Side: public Objekt {
20 public:
21    virtual ~Side() {}
22    /**
23     * Public pure virtual function which returns the type of the current
24     * object, only for derived classes Wall or Entry
25     * @return The type of the object.
26     */
27    virtual Typ GetTyp () const = 0;
28
29    virtual void PrintInfo () const = 0;
30
31    //virtual Side& operator= (Side const& side) = 0;
32
33    virtual Side *Clone() = 0;
34
35 protected:
36    /**
37     * Default CTor with no arguments
38     */
39    Side() {};
40
41    /**
42     * CTor with 1 argument
43     * Creates a side with the specified typ
44     */
45    Side (Typ typ) : mTyp(typ) {}
46
47    /**
48     * Protected variable.
49     * Contains type of derived class
50     * Typ is enum of WALL or ENTRY
51     */
52    Typ mTyp;
53 };
54
55 #endif
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile    : Wall.h
3  // Author      : Johannes Selymes
4  // Date        : 25. 10. 2012
5  // Description : Wall - derived from Side
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef WALL_H
11 #define WALL_H
12
13 #include "Objekt.h"
14 #include "Side.h"
15
16 /**
17  * class Wall.
18  * Derived from Side, defines which
19  * colour has the wall
20  */
21 class Wall: public Side {
22 public:
23
24    Wall () : mColour(WHITE), Side(WALL) {}
25
26    /*Wall (Side* const side);*/
27
28    Wall (Colour col) : mColour(col), Side(WALL) {}
29    //Wall& operator= (Wall const& wall);
30
31    //copy constr
32    Wall (Wall const& wall);
33
34    virtual ~Wall() {}
35
36    /**
37     * Is the overridden function of
38     * Side::GetTyp ,
39     * @return WALL.
40     */
41    Typ GetTyp () const;
42    Colour GetColour () const;
43    void PrintInfo () const;
44
45    /*virtual Side& operator= (Side const& side);*/
46
47    Side *Clone();
48
49 private:
50    /**
51     * Private variable.
52     * Contains colour of the Wall
53     */
54    Colour mColour;
55 };
56
57
58 #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile    : Wall.cpp
// Author      : Johannes Selymes
// Date        : 25. 10. 2012
// Description : Wall - Implementation
// Remarks     : -
// Revision    : 0
////////////////////////////////////////////////////////////////////////

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



Wall::Wall (Wall const& wall) {
   mTyp = wall.GetTyp();
   mColour = wall.GetColour();
}



Typ Wall::GetTyp ()const{
   return mTyp;
}

Colour Wall::GetColour () const {
   return mColour;
}

void Wall::PrintInfo () const {
   std::cout << "Typ: " << "WALL" << std::endl;
   std::cout << "Farbe: " << mColour << std::endl;
}


//Wall& Wall::operator= (Wall const& wall) {
// mTyp = wall.GetTyp();
// mColour = wall.GetColour();
// return *this;
//}


//Side& Wall::operator= (Side const& side) {
// *this = side;
// return *this;
//}

Side * Wall::Clone() {
   return new Wall(*this);
}
```

```
1  ////////////////////////////////////////////////////////////////////////
2  // Workfile    : Entry.h
3  // Author      : Johannes Selymes
4  // Date        : 25. 10. 2012
5  // Description : Entry - derived from Side.
6  // Remarks     : -
7  // Revision    : 0
8  ////////////////////////////////////////////////////////////////////////
9
10 #ifndef ENTRY_H
11 #define ENTRY_H
12
13 #include "Side.h"
14 //#include "Roomplan.h"
15
16 /**
17  * class Entry.
18  * Derived from Side, defines if the
19  * door is opened or not and saves
20  * the id of the own room and from the other room
21  */
22 class Entry: public Side {
23 public:
24    //Entry () : mOpened (false), mMyRoomId (-1), mOtherRoomId (-1), Side(
          ENTRY) {}
25    //Entry (bool const &open) : mOpened (open), mMyRoomId (-1),
          mOtherRoomId (-1), Side(ENTRY) {}
26
27    //nur den eigentlich, aber impl von room nimmt 2
28    //Entry (int mMyRoom, int mOtherRoom) :
29    //mOpened (false), mMyRoomId (mMyRoom), mOtherRoomId (mOtherRoom), Side(
          ENTRY) {}
30
31
32    //bester Ctor
33    Entry (int currentId, bool opened, Dir const& dir);
34
35    //copy
36    Entry (Entry const& entry);
37
38    virtual ~Entry () {}
39
40    /**
41     * Is the overridden function of
42     * Side::GetTyp ,
43     * @return ENTRY.
44     */
45
46
47    /*Entry& operator= (Entry const& entry);*/
48
49    Typ GetTyp () const;
50    bool IsOpened () const;
51    int GetMyRoomId () const;
52    int GetOtherRoomId () const;
53    void PrintInfo () const;
54
55
56    /*virtual Side& operator= (Side const& side);*/
57
```

```
58      Side *Clone();
59
60  private:
61      /**
62       * Private variable.
63       * Says if the entry is opened
64       */
65      bool mOpened;
66      /**
67       * Private variable.
68       * Is the Id of the Room which has this entry
69       */
70      int mMyRoomId;
71      /**
72       * Private variable.
73       * Is the Id of the Room where the entry leads to
74       */
75      int mOtherRoomId;
76  };
77
78
79
80
81
82
83  #endif
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile    : Entry.cpp
// Author      : Johannes Selymes
// Date        : 25. 10. 2012
// Description : Entry - Implementation
// Remarks     : -
// Revision    : 0
////////////////////////////////////////////////////////////////////////

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

Entry::Entry (int currentId, bool opened, Dir const& dir) {
   mTyp = ENTRY;
   mOpened = opened;

   mMyRoomId = currentId;
   //first room with id 0

   if (dir == NORTH) {
      mOtherRoomId = (currentId - 1);
   }
   else if (dir == SOUTH) {
      mOtherRoomId = (currentId + 1);
   }
   else {
      mOtherRoomId = - 1; //undefined, entry to outside
   }
}

Entry::Entry (Entry const& entry) {
   mTyp = entry.GetTyp();
   mOpened = entry.IsOpened();
   mMyRoomId = entry.GetMyRoomId();
   mOtherRoomId = entry.GetOtherRoomId();
}

//get
Typ Entry::GetTyp () const{
   return mTyp;
}
bool Entry::IsOpened () const {
   return mOpened;
}

int Entry::GetMyRoomId () const {
   return mMyRoomId;
}
int Entry::GetOtherRoomId () const {
   return mOtherRoomId;
}

void Entry::PrintInfo () const {
   std::cout << "Typ: " << "ENTRY" << std::endl;



   std::cout << "Tuer offen: " << std::boolalpha << mOpened << std::endl;
   std::cout << "Mein Raum: " << mMyRoomId << std::endl;
   std::cout << "Anderer Raum: " << mOtherRoomId << std::endl;
```

```
61 }
62
63 //Entry& Entry::operator= (Entry const& entry) {
64 // mTyp = entry.GetTyp();
65 // mOpened = entry.IsOpened();
66 // mMyRoomId = entry.GetMyRoomId();
67 // mOtherRoomId = entry.GetOtherRoomId();
68 // return *this;
69 //}
70 //
71 //
72 //Side& Entry::operator= (Side const& side) {
73 // *this = side;
74 // return *this;
75 //}
76
77 Side* Entry::Clone() {
78    return new Entry(*this);
79 }
```

```cpp
////////////////////////////////////////////////////////////////////////
// Workfile    : main.cpp
// Author      : Patrick Felixberger, Johannes Selymes
// Date        : 17. 10. 2012
// Description : Testtreiber.
// Remarks     : -
// Revision    : 0
////////////////////////////////////////////////////////////////////////
#include <iostream>
#include <vld.h>
#include "Objekt.h"
#include "Roomplan.h"


using namespace std;


int main() {

   Roomplan rp1;
   //Side* NW = new Wall(GREEN);
   //Side* SW = new Wall(GREEN);
   //Side* EW = new Wall(GREEN);
   //Side* WW = new Wall(GREEN);

   /*&Wall(GREEN),&Wall(GREEN),&Wall(GREEN),&Entry(rp.GetCounter(),-1)*/



   rp1.Add(&Entry(rp1.GetCounter(),true,NORTH),&Wall(RED),&Wall(BLUE),&Wall
       (GREEN));
   rp1.Add(&Entry(rp1.GetCounter(),false,NORTH),&Wall(GREEN),&Wall(GREEN),&
       Wall(GREEN));
   rp1.Add(WALL, WALL, WALL, WALL);
   rp1.Add(ENTRY, WALL, ENTRY, WALL);

   rp1.Add(ENTRY,ENTRY,WALL,ENTRY);
   rp1.Add(WALL, ENTRY, WALL, ENTRY);
   rp1.Add(ENTRY, ENTRY, ENTRY, WALL);

   rp1.Print();

   //Angabe
   Roomplan rp2;
   rp2.Add(&Wall(),&Entry(rp2.GetCounter(),true,SOUTH),&Wall(WHITE),&Wall(
       GREEN));
   rp2.Add(ENTRY,WALL,ENTRY,WALL);
   rp2.Add(&Wall(),&Wall(RED),&Wall(GREEN),&Entry(rp2.GetCounter(),false,
       WEST));

   rp2.Print();



   return 0;
}
```

7 Testausgaben