```cpp
#include <iostream>
#include "IntList.h"
#include "FehlerWert.h"

int main() {
    IntList list, list2;

    list.print();
    list.insert(1, -1);
    list.insert(1, -1);
    list.insert(1, -1);
    list.insert(1, -1);
    list.print();

    list2 = list;

    list2.print();

    FehlerWert a(10.0, 2.0), b(5.0, 1.0);
    FehlerWert s = a + b;
    std::cout << s.wert() << " +- " << s.absolut() << " (" << s.relativ() << "%)" << std::endl;
    FehlerWert p = a * b;
    std::cout << p.wert() << " +- " << p.absolut() << " (" << p.relativ() << "%)" << std::endl;

    return 0;
}
```

File - /home/christian/studium/ipi-ws2014/Z08/IntList.h

```
 1 class IntList {
 2 public:
 3     IntList();
 4
 5     IntList(const IntList &other);
 6
 7     IntList &operator=(const IntList &other);
 8
 9     ~IntList();
10
11     int getCount();
12
13     bool isEmpty();
14
15     void print();
16
17     void insert(int element, int position);
18
19     void remove(int position);
20
21     int getElement(int position);
22
23 private:
24     struct IntListElem {
25         IntListElem *next;
26         int value;
27     };
28     int count;
29     IntListElem *first;
30
31     IntListElem *findElement(int position);
32 };
```

```cpp
#include "IntList.h"
#include <iostream>

IntList::IntList() {
    this->count = 0;
    this->first = 0;
};

IntList::IntList(const IntList &other) {
    this->count = 0;
    this->first = 0;

    IntListElem *el;
    el = other.first;
    while (el) {
        this->insert(el->value, -1);
        el = el->next;
    }
};

IntList &IntList::operator=(const IntList &other) {
    if (this != &other) {
        while (this->getCount()) {
            this->remove(0);
        }

        IntListElem *el;
        el = other.first;
        while (el) {
            this->insert(el->value, -1);
            el = el->next;
        }
    }
    return *this;
};

IntList::~IntList() {
    if (this->getCount() > 0 && this->first != 0) {
        IntListElem *cur = this->first;
        IntListElem *next = cur->next;
        while (next) {
            cur->value = 0;
            delete cur;
            cur = next;
            next = cur->next;
        }
    }
};

int IntList::getCount() {
    return this->count;
};

bool IntList::isEmpty() {
    return this->getCount() > 0;
};


void IntList::print() {
    if (this->getCount()) {
        IntListElem *el = this->first;
        std::cout << "[" << el->value;
        el = el->next;
        while (el) {
            std::cout << "," << el->value;
            el = el->next;
        }
        std::cout << "]" << std::endl;
    } else {
        std::cout << "empty list" << std::endl;
    }
};

void IntList::insert(int element, int position) {
    if (position > 0 && position < this->getCount()) {
        IntListElem *elmToMod = this->first;
        for (int i = 0; i < position; i++) {
            elmToMod = elmToMod->next;
        }
        elmToMod->value = element;
    } else if (position == this->getCount()) {
        IntListElem *elmToInsert = new IntListElem;
        elmToInsert->value = element;
        elmToInsert->next = 0;
        IntListElem *elmAfter = this->first;
        if (position == 0) {
            this->first = elmToInsert;
        } else {
            while (elmAfter->next) {
                elmAfter = elmAfter->next;
            }
            elmAfter->next = elmToInsert;
```

```
 93         }
 94         this->count++;
 95     } else if (position == -1) {
 96         this->insert(element, this->getCount());
 97     }
 98 };
 99
100 void IntList::remove(int position) {
101     IntListElem *elToDel, *elPrev;
102     elToDel = findElement(position);
103     if (position == 0) {
104         if (elToDel) {
105             this->first = elToDel->next;
106             this->count--;
107         }
108     } else if (position > 0 && position < this->getCount()) {
109
110         findElement(position - 1)->next = elToDel->next;
111         this->count--;
112     }
113     delete elToDel;
114
115 };
116
117 int IntList::getElement(int position) {
118     IntListElem *el = findElement(position);
119     return el ? el->value : 0;
120 };
121
122 IntList::IntListElem *IntList::findElement(int position) {
123     IntListElem *el = this->first;
124     for (int i = 0; i < position && el; i++) {
125         el = el->next;
126     }
127     return el;
128 };
129
```

```
 1 class FehlerWert {
 2 public:
 3     FehlerWert(double nominal_value, double std_dev);
 4 
 5     FehlerWert(const FehlerWert &other);
 6 
 7     FehlerWert operator+(const FehlerWert &other);
 8 
 9     FehlerWert operator*(const FehlerWert &other);
10 
11     FehlerWert &operator=(const FehlerWert &other);
12 
13     double wert();
14 
15     double absolut();
16 
17     double relativ();
18 private:
19     double nominal_value;
20     double std_dev;
21 
22     FehlerWert();
23 };
```

File - /home/christian/studium/ipi-ws2014/Z08/CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.4)
project(Z08)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")

set(SOURCE_FILES main.cpp IntList.cpp FehlerWert.cpp)
add_executable(Z08 ${SOURCE_FILES})
```

```cpp
#include <math.h>
#include "FehlerWert.h"

FehlerWert::FehlerWert(double nominal_value, double std_dev) {
    this->nominal_value = nominal_value;
    this->std_dev = std_dev;
};

FehlerWert::FehlerWert(const FehlerWert &other) {
    this->nominal_value = other.nominal_value;
    this->std_dev = other.std_dev;
}

FehlerWert FehlerWert::operator+(const FehlerWert &other) {
    return FehlerWert(
            this->nominal_value + other.nominal_value,
            sqrt(this->std_dev * this->std_dev + other.std_dev * other.std_dev)
    );
};

FehlerWert FehlerWert::operator*(const FehlerWert &other) {
    double nominal_value, std_dev;
    nominal_value = this->nominal_value * other.nominal_value;
    std_dev = sqrt(
            (this->std_dev / this->nominal_value) * (this->std_dev / this->nominal_value)
                    + (other.std_dev / other.nominal_value) * (other.std_dev / other.nominal_value)
    );
    FehlerWert tmp(nominal_value, std_dev);
    return tmp;
};

FehlerWert &FehlerWert::operator=(const FehlerWert &other) {
    if (this != &other) {
        this->nominal_value = other.nominal_value;
        this->std_dev = other.std_dev;
    }

    return *this;
};

double FehlerWert::wert() {
    return this->nominal_value;
};

double FehlerWert::absolut() {
    return this->std_dev;
};

double FehlerWert::relativ() {
    return this->std_dev / this->nominal_value;
};

FehlerWert::FehlerWert() {

};
```