SCHETT MATTHIAS

SEN-ÜBUNG 05

Inhaltsverzeichnis

Aufgabe 1

Lösungsidee

Um ein Wertepaar abzuspeichern wurde eine Struktur namens DiveData erstellt. Diese Struktur wird dann in einem std::vector abgespeichert.

```
struct DiveData{
    time_t mTime;
    double mDepth;
    double mUpDown;
};

extern std::vector<DiveData> diveComputer;
```

Das extern vor der Deklaration des Vektors ist notwendig um Mehrfach Definitionen zu verhindern

Zum Abspeichern des Eingabestromes wird der scanner Klasse ein Eingabestrom übergeben und dieser anschließend durchgegangen und gelesen, sollten die Werte nicht den Erwartungen entsprechen wird eine std::exception mit der Meldung <Unknown format> ausgelöst um anzuzeigen, dass ein Fehler aufgetreten ist.

Die Ausgabe erfolgt anschließend in Tabellenform, dafür wurden die Folgenden Manipulatoren erstellt.

```
ostream& hr(ostream& os) {
    return os << "——————————————————————
    ——————————————————————";
}

ostream& colSpace(ostream& os) {
    return os << setw(colSpacing) << " ";
}

ostream& colWidth(ostream& os){
    return os << setw(colWidthNum) << " ";
}

ostream& colFormat(ostream& os) {
    return os << setw(colWidthNum);
}

ostream& formatUpDown(ostream &os){
    return os << right << setiosflags(ios::fixed)
        << setprecision(4);
}

ostream& formatDepth(ostream &os){
    return os << right << setiosflags(ios::fixed)
        << setprecision(2);
}
```

Der Programmcode befindet sich ab Aufgabe 1

Testfälle

Listing 1: Input Testfall 1

```
0 (0.0) 10 (2.5) 50 (6.8) 150 (15.0) 270 (23.88)
    800 (26.0) 1235 (20.5) 1780 (15.8) 2345(8.3)
    3876 (0.0)
```

Listing 2: Input Testfall 2

```
0 0.0) 10 (2.5) 50 (6.8) 150 (15.0) 270 (23.88)
    800 (26.0) 1235 (20.5) 1780 (15.8) 2345(8.3)
    3876 (0.0)
```

Listing 3: Ausgabe

```


 Testfall 01
--------------


Dive Time        Dive Depth        Down/Up
(hh:mm:ss)       (m)              (m/sec)
-------------------------------------------------------------
 00:00:00               0.00
                                         -0.2500
 00:00:10               2.50
                                         -0.1075
 00:00:50               6.80
                                         -0.0820
 00:02:30              15.00
                                         -0.0740
 00:04:30              23.88
                                         -0.0040
 00:13:20              26.00
                                          0.0126
 00:20:35              20.50
                                          0.0086
 00:29:40              15.80
                                          0.0133
 00:39:05               8.30
                                          0.0054
 01:04:36               0.00


 Testfall 02
--------------

Unkown format
```

Aufgabe 2

Lösungsidee

Da eine Lagerverwaltung aus einem Lager und das Lager wiederum aus Artikeln besteht, benötigen wir 2 Klassen.

- Article
- WareHouse

Diese beiden Klassen ermöglichen uns dann die Lagerverwaltung.
Ein Artikel besteht aus :

```
    int mArticleNumber;
    std::string mArticleName;
    size_t mQuantity;
    double mPrice;
```

Jeder dieser Member bestitzt eine Getter und eine Setter Methode die es erlaubt die Werte zu lesen und zu verändern. Weiters wurde operator< überschrieben um die Lagermenge[1] zu vergleichen.

[1] mQuantity

```
bool Article::operator<(Article const & vgl) const
   {
    return (mQuantity < vgl.mQuantity);
}
```

Das Lager besteht aus:

```
    std::vector<Article> mArticles;
    std::string mWareHouseName;
```

Hier bestizt allerdings keiner der Member eine Getter oder Setter Funktion. Es gibt nur eine GetNumberOfArticles Funktion die, die aktuelle Größe des std::vector ausgibt. Um einen neuen Artikel abzuspeichern gibt es die AddArticle(Article const &newArticle) Funtkion, die std::vector::push_back aufruft. Es gibt auch die Möglichkeit aus einem File einzulesen, dabei hilft abermals die scanner Klasse und liest das File ein, tritt bei der Verarbeitung ein Fehler auf, wenn das Format nicht richtig ist, wird eine std::exception geworfen mit der Meldung <Unknown format>. Die Ausgabe erfolgt anschließend im vorgegebenen Tabellenformat.
Der Quellcode findet sich ab Aufgabe 2

Testfälle

Listing 4: Input Testfall 1

```
4711 - "Kabelkanal (3m)" - 10 - 1.50;
1147 - "Installationsrohr (5m)" - 49 - 3.49;
7141 - "Funksteckdose" - 3 - 11.99;
1471 - "Wechselschalter" - 17 - 7.90;
1417 - "Ein-/Ausschalter" - 24 - 6.99;
1714 - "Zeitschaltuhr" - 5 - 33.50;
```

Listing 5: Input Testfall 2

```
4711 - "Kabelkanal (3m)" - 10 -;
1147 - "Installationsrohr (5m)" - 49 - 3.49;
```

Listing 6: Ausgabe

```


 Testfall 01
------------

 Article list of Warehouse TestWarehouse

7141   Funksteckdose            3    11.99
1714   Zeitschaltuhr            5    33.50
4711   Kabelkanal (3m)          10    1.50
1471   Wechselschalter          17    7.90
1417   Ein-/Ausschalter         24    6.99
1147   Installationsrohr (5m)   49    3.49


 Testfall 02
------------

Unknown format
```

Anhang A
Aufgabe 1

Listing A.1: Header für den Tauchcomputer

```
///////////////////////////////////////
// Workfile     : Main.cpp
// Author       : Matthias Schett
// Date         : 12-04-2013
// Description  : Dive Computer
// Remarks      : -
// Revision     : 0
///////////////////////////////////////

#ifndef DIVE_H
#define DIVE_H

#include <ctime>
#include <vector>
#include <ostream>
#include <istream>
#include <exception>

int const colSpacing = 4;
int const colWidthNum = 10;
int const diveDepthPrecision = 2;
int const upDownPrecision = 3;

struct DiveData{
    time_t mTime;
    double mDepth;
    double mUpDown;
};

extern std::vector<DiveData> diveComputer;

//*************************************
// Method:     readDiveData
// FullName:   readDiveData
// Access:     public
// Returns:    void
// Qualifier:
// Parameter: std::istream &is
// Reads dive data from stream and saves it -
   throws exception when an error occurs
//************************************
void readDiveData(std::istream &is);

//************************************
// Method:     printDiveData
// FullName:   printDiveData
// Access:     public
// Returns:    void
// Qualifier:
// Parameter: std::ostream & os
// Prints formatted dive data to stream
//************************************
void printDiveData(std::ostream &os);

#endif
```

Listing A.2: Implementierung des Tauchcomputers

```
///////////////////////////////////////
// Workfile     : Main.cpp
// Author       : Matthias Schett
// Date         : 12-04-2013
// Description  : Dive Computer
// Remarks      : -
// Revision     : 0
///////////////////////////////////////

#include "DiveComputer.h"
#include <string>
#include "scanner.h"
#include <iomanip>

using namespace std;

vector<DiveData> diveComputer (0);

bool isTbReal(scanner &scan){
    return scan.symbol_is_real();
}

bool isTbBracket(scanner &scan){
    return scan.symbol_is_lpar();
}

bool isTbInt(scanner &scan){
```

```cpp
    return scan.symbol_is_integer();
}

time_t parseTime(scanner &scan){
    time_t temp = scan.get_integer();
    scan.next_symbol();
    return temp;
}

double parseDepth( scanner & scan ) {
    scan.next_symbol();
    if(isTbReal(scan)){
        double temp = scan.get_real();
        scan.next_symbol();
        return temp;
    }
    return 0.0;
}

void calcUpDown(DiveData &newData) {
    if(!diveComputer.empty()){
        DiveData oldData = diveComputer.at(
            diveComputer.size() - 1);
        newData.mUpDown = (oldData.mDepth -
            newData.mDepth) / (newData.mTime -
            oldData.mTime);
    } else {
        newData.mUpDown = 0.0;
    }
}

void readDiveData( std::istream &is ){

    scanner scan(is);

    while(!scan.symbol_is_eof()){
        if(isTbInt(scan)){
            time_t parsedTime = parseTime(scan);
            if(isTbBracket(scan)){
                double parsedDepth = parseDepth(
                    scan);
                scan.next_symbol();
                DiveData data;
                data.mDepth = parsedDepth;
                data.mTime = parsedTime;
                calcUpDown(data);
                diveComputer.push_back(data);
            }
        } else {
            throw std::exception("Unkown format");
        }
    }
}

ostream& hr(ostream& os) {
    return os << "
        ________________________________________________________
        ";
}

ostream& colSpace(ostream& os) {
    return os << setw(colSpacing) << " ";
}

ostream& colWidth(ostream& os){
    return os << setw(colWidthNum) << " ";
}

ostream& colFormat(ostream& os) {
    return os << setw(colWidthNum);
}

ostream& formatUpDown(ostream &os){
    return os << right << setiosflags(ios::fixed)
        << setprecision(4);
}

ostream& formatDepth(ostream &os){
    return os << right << setiosflags(ios::fixed)
        << setprecision(2);
}


void printTableHeader(std::ostream &os){
    os << left << colFormat << "Dive Time " <<
        colSpace << colFormat << "Dive Depth " <<
        colSpace << colFormat << "Down/Up";
    os << endl << colFormat << "(hh:mm:ss)"  <<
        colSpace << colFormat << "(m)" << colSpace
         << colFormat << "(m/sec)" << endl << hr <<
         endl;
}

void printDataLine(std::ostream &os, DiveData
    const & data, int i){
    struct tm * ptm = gmtime(&data.mTime);
    if(i != 0){ // Don't print this at the first
        line
        os << colWidth << colSpace << colWidth <<
             colSpace << colFormat << formatUpDown
             << data.mUpDown << endl;
    }
    os << colFormat << put_time(ptm, "%H:%M:%S") <<
         colSpace << colFormat << formatDepth <<
        data.mDepth << endl;
}

void printDiveData( std::ostream &os ){

   printTableHeader(os);

   for(int i = 0; i < diveComputer.size(); i++){
       printDataLine(os, diveComputer.at(i), i);
   }

}
```

Listing A.3: Testtreiber

```
///////////////////////////////////////
// Workfile      : Main.cpp
// Author        : Matthias Schett
// Date          : 12-04-2013
// Description   : Dive Computer
// Remarks       : -
// Revision      : 0
///////////////////////////////////////

#include <vld.h>
#include <iostream>
#include <fstream>
#include <string>
#include "scanner.h"
#include "DiveComputer.h"
#include <exception>

using namespace std;

void printTestHeader(int testNumber, ostream &
    stream){
    stream << endl << endl;
    if(testNumber < 10){
        stream << "Testfall 0" << testNumber;
    } else {
        stream << "Testfall " << testNumber;
    }
    stream << endl << "-----------" << endl <<
        endl;

}

int main(){
    ofstream oFile("OutputA1.txt");
    try{
        printTestHeader(1, oFile);

        ifstream file("Test.txt");
        readDiveData(file);

        printDiveData(oFile);

        printTestHeader(2, oFile);

        ifstream file2("TestIncorrect.txt");
        readDiveData(file2);

        printDiveData(oFile);

        file.close();
        file2.close();

    } catch(std::exception &e){
        oFile << e.what();
    }
    oFile.close();
    cin.get();
    return 0;
}
```

Anhang B
Aufgabe 2

Listing B.1: Header für den Artikel

```
////////////////////////////////////////
// Workfile    : Article.h
// Author      : Matthias Schett
// Date        : 20-04-2013
// Description : Ware house management
// Remarks     : -
// Revision    : 0
////////////////////////////////////////

#ifndef ARTICLE_H
#define ARTICLE_H

#include <string>
#include <exception>

class ArticleException: public std::exception
{

};

class Article {
private:

    int mArticleNumber;
    std::string mArticleName;
    size_t mQuantity;
    double mPrice;

public:
    // Ctr
    Article(int articleNumber, std::string
        ArticleName, size_t quantity, double price)
        ;

    // Dtr
    ~Article();

    // Getters
    int getArticleNumber();
    std::string & getArticleName();
    size_t getQuantity();
    double getPrice();

    // Setters
    void setArticleNumber(int articleNumber);
    void setArticleName(std::string & articleName)
        ;
    void setQuantity(size_t quantity);
    // Throws exception if price is negativ
    void setPrice(double price);

    // compares the quantity of two articles
    bool operator<(Article const & vgl) const;

};

#endif
```

Listing B.2: Implementierung des Artikels

```
////////////////////////////////////////
// Workfile     : Article.cpp
// Author       : Matthias Schett
// Date         : 20-04-2013
// Description  : Ware house management
// Remarks      : -
// Revision     : 0
////////////////////////////////////////
#include "Article.h"
#include <exception>

using namespace std;

Article::Article(int articleNumber, std::string
    articleName, size_t quantity, double price) :
    mArticleNumber(articleNumber), mArticleName(
    articleName), mQuantity(quantity), mPrice(price
    )
{
}

Article::~Article()
{
}

```

```
int Article::getArticleNumber(){
    return mArticleNumber;
}

std::string &Article::getArticleName(){
    return mArticleName;
}

size_t Article::getQuantity(){
    return mQuantity;
}

double Article::getPrice(){
    return mPrice;
}

void Article::setArticleNumber(int articleNumber){
    mArticleNumber = articleNumber;
}

void Article::setArticleName(std::string &
    articleName){
    mArticleName = articleName;
}

void Article::setQuantity(size_t quantity){
    mQuantity = quantity;
}

void Article::setPrice(double price){
    if(price < 0.0){
        throw std::exception("Price is not allowed
             to be negative");
    }
    mPrice = price;
}

bool Article::operator<(Article const & vgl) const
    {
    return (mQuantity < vgl.mQuantity);
}
```

Listing B.3: Header für das Lager

```
////////////////////////////////////////
// Workfile      : Warehouse.h
// Author        : Matthias Schett
// Date          : 20-04-2013
// Description   : Ware house management
// Remarks       : -
// Revision      : 0
////////////////////////////////////////
#ifndef WAREHOUSE_H
#define WAREHOUSE_H

#include "Article.h"
#include <vector>
#include <fstream>
#include <ostream>

class WareHouse {
private:

    std::vector<Article> mArticles;
    std::string mWareHouseName;


    size_t searchLongestName();
    size_t searchHighestArticleNum();
    size_t searchHighestQuantity();
    size_t searchHighestPrice();

public:
    WareHouse(std::string wareHouseName);
    ~WareHouse();

    //************************************
    // Method:     addArticle
    // FullName:   WareHouse::addArticle
    // Access:     public
    // Returns:    void
    // Qualifier:
    // Parameter: Article const & newArticle
    //************************************
    void addArticle(Article const &newArticle);

    //************************************
    // Method:     getNumberOfArticles
    // FullName:   WareHouse::getNumberOfArticles
    // Access:     public
    // Returns:    size_t
    // Qualifier:
    //************************************
    size_t getNumberOfArticles();

    //************************************
    // Method:     readArticlesFromFile
    // FullName:   WareHouse::readArticlesFromFile
    // Access:     public
    // Returns:    void
    // Qualifier:
    // Parameter: ifstream & file
    // Reads articles from a file and adds them
    //************************************
    void readArticlesFromFile(std::ifstream &file)
        ;

    //************************************
    // Method:     printArticleList
    // FullName:   WareHouse::printArticleList
    // Access:     public
    // Returns:    void
    // Qualifier:
    // Parameter: ostream & os
    // Prints the articles to the given stream
    //************************************
    void printArticleList(std::ostream &os);
};
#endif // WAREHOUSE_H
```

Listing B.4: Implementierung des Lagers

```
///////////////////////////////////////
// Workfile      : Warehouse.cpp
// Author        : Matthias Schett
// Date          : 20-04-2013
// Description   : Ware house management
// Remarks       : -
// Revision      : 0
///////////////////////////////////////
#include "WareHouse.h"
#include "scanner.h"
#include <exception>
#include <algorithm>
#include <iomanip>

using namespace std;

// Comparison function object for sort method
class comp
{
public:
    bool operator() (const Article &a, const
        Article &b) const
    {
        return a < b;
    }
};

```

```
int intlen(float start) {
    int end = 0;
    while(start >= 1) {
        start = start/10;
        end++;
    }
    return end;
}

size_t WareHouse::searchLongestName(){
    size_t length = 0;
    for (size_t i = 0; i < mArticles.size() - 1; i
        ++){
        size_t length1 = mArticles.at(i).
            getArticleName().length();
        size_t length2 = mArticles.at(i+1).
            getArticleName().length();
        if(length1 < length2){
            length = length2;
        } else if(length1 > length2){
            length = length1;
        }
    }

    return length;
}

size_t WareHouse::searchHighestArticleNum(){
    size_t length = 0;
    for (size_t i = 0; i < mArticles.size() - 1; i
        ++){
        size_t length1 = mArticles.at(i).
            getArticleNumber();
        size_t length2 = mArticles.at(i+1).
            getArticleNumber();
        if(length1 < length2){
            length = length2;
        } else if(length1 > length2){
            length = length1;
        }
    }

    return intlen(length);
}

size_t WareHouse::searchHighestQuantity(){
    size_t length = 0;
    for (size_t i = 0; i < mArticles.size() - 1; i
        ++){
```

```
        size_t length1 = mArticles.at(i).getQuantity();
        size_t length2 = mArticles.at(i+1).getQuantity();
        if(length1 < length2){
            length = length2;
        } else if(length1 > length2){
            length = length1;
        }
    }

    return intlen(length);
}

size_t WareHouse::searchHighestPrice(){
    size_t length = 0;
    for (size_t i = 0; i < mArticles.size() - 1; i++){
        size_t length1 = mArticles.at(i).getPrice();
        size_t length2 = mArticles.at(i+1).getPrice();
        if(length1 < length2){
            length = length2;
        } else if(length1 > length2){
            length = length1;
        }
    }

    return intlen(length);
}


bool isTbReal(scanner &scan){
    return scan.symbol_is_real();
}

bool isTbString(scanner &scan){
    return scan.symbol_is_string();
}

bool isTbInt(scanner &scan){
    return scan.symbol_is_integer();
}

int parseArticleNumberOrQuantity(scanner &scan){
    if(isTbInt(scan)){
        return scan.get_integer();
    }
    throw std::exception("Unknown format");
}

string parseArticleName(scanner &scan){
    if(isTbString(scan)){
        return scan.get_string();
    }
    throw std::exception("Unknown format");
}

double parsePrice(scanner &scan){
    if(isTbReal(scan)){
        return scan.get_real();
    }
    throw std::exception("Unknown format");
}

WareHouse::WareHouse(std::string wareHouseName) :
   mWareHouseName(wareHouseName), mArticles() {
}

WareHouse::~WareHouse(){
}

void WareHouse::addArticle(Article const &
    newArticle){
    mArticles.push_back(newArticle);
}

size_t WareHouse::getNumberOfArticles(){
    return mArticles.size();
}

void WareHouse::readArticlesFromFile(std::ifstream
    &file){
    scanner scan (file);

    while(!scan.symbol_is_eof()){
        if(isTbInt(scan)){
            int articleNum =
                parseArticleNumberOrQuantity(scan);

            scan.next_symbol();
            scan.next_symbol();
            if(isTbString(scan)){
                string articleName =
                    parseArticleName(scan);
                scan.next_symbol();
                scan.next_symbol();
                if(isTbInt(scan)){
                    size_t quant = parseArticleNumberOrQuantity(scan);
                    scan.next_symbol();
                    scan.next_symbol();
                    if(isTbReal(scan)){
                        double price = parsePrice(scan);
                        scan.next_symbol();
                        scan.next_symbol();
                        Article art (articleNum, articleName, quant, price);
                        mArticles.push_back(art);
                    }
                }
            }
        } else{
            throw std::exception("Unknown format");
        }
    }
}

void WareHouse::printArticleList( std::ostream &os ) {

    sort(mArticles.begin(), mArticles.end(), comp());

    size_t colSpacing = 2;
    size_t prec = 2;
    size_t articleNumLength = searchHighestArticleNum() + colSpacing;
    size_t articleNameLength = searchLongestName() + colSpacing;
    size_t articleQuantityLength = searchHighestQuantity() + colSpacing;
    size_t articlePriceLength = searchHighestPrice() + colSpacing + prec;


    os << "Article list of Warehouse " << mWareHouseName << endl;

    for (std::vector<Article>::iterator it=mArticles.begin(); it!=mArticles.end(); ++it){
        os << setw(articleNumLength) << left << it
            ->getArticleNumber();
        os << setw(articleNameLength) << it->
            getArticleName();
        os << setw(articleQuantityLength) << it->
            getQuantity();
        os << right << setiosflags(ios::fixed) <<
            setw(articlePriceLength) <<
            setprecision(prec) << it->getPrice() <<
             endl;
    }

}
```

Listing B.5: Testtreiber

```
////////////////////////////////////////
// Workfile    : Main.cpp
// Author      : Matthias Schett
// Date        : 20-04-2013
// Description : Ware house management
// Remarks     : -
// Revision    : 0
////////////////////////////////////////
#include <vld.h>
#include "Article.h"
#include "WareHouse.h"
#include <fstream>
#include <iostream>
#include <string>

using namespace std;

void printTestHeader(int testNumber, ostream &
    stream){
    stream << endl << endl;
    if(testNumber < 10){
        stream << "Testfall 0" << testNumber;
    } else {
        stream << "Testfall " << testNumber;
    }
    stream << endl << "------------" << endl <<
        endl;

}


int main(){
    ofstream oFile("OutputA2.txt");
    try{
        printTestHeader(1, oFile);

        WareHouse newWarehouse (string("
            TestWarehouse"));

        ifstream file ("Input.txt");

        newWarehouse.readArticlesFromFile(file);

        newWarehouse.printArticleList(oFile);

        printTestHeader(2, oFile);

        WareHouse newWarehouse2 (string("
            TestWarehouse"));

        ifstream file2 ("InputWrong.txt");

        newWarehouse2.readArticlesFromFile(file2);

        newWarehouse2.printArticleList(oFile);

        file.close();
        file2.close();
    } catch(exception e){
        oFile << e.what();
    }

    oFile.close();

    cin.get();
    return 0;
}
```