FH-OÖ Hagenberg/HSD
SDP3, WS 2025
Übung 6

Name: Simon Offenberger / Simon Vogelhuber — Aufwand in h: siehe Doku.

Mat.Nr: S2410306027 / S2410306014 — Punkte:

Übungsgruppe: 1 — korrigiert:

Beispiel 1 (24 Punkte) Dateisystem-Simulation: Entwerfen Sie aus der nachfolgenden Spezifikation ein Klassendiagramm, instanzieren Sie dieses und implementieren Sie die Funktionalität entsprechend:

Ein Dateisystem für ein einfaches, eingebettetes System besteht aus Dateien, Ordner und Verweise auf Dateien, Ordner oder weitere Verweise. Ein Ordner kann Dateien, Verweise und weitere Ordner beinhalten. Dateien, Ordner und Verweise werden über einen Namen spezifiziert, der verändert werden kann.

Eine Datei hat zusätzlich folgende Eigenschaften:

- aktuelle Dateigröße in Bytes
- Größe eines Blockes auf dem Speichermedium in Bytes
- Anzahl der reservierten Blöcke

Die Größe eines Blockes und die Anzahl der reservierten Blöcke kann für jede Datei bei der Erzeugung unterschiedlich festgelegt werden. Ein nachträgliches Ändern dieser Eigenschaften ist nicht möglich!

Das Schreiben in eine Datei wird durch eine Methode `Write(size_t const bytes)` simuliert. Achten Sie darauf, dass die Datei nicht größer werden kann als der für die Datei reservierte Speicher!

Implementieren Sie zur Erzeugung von Dateien, Ordner und Verweise eine einfache Fabrik.

Implementieren Sie einen Visitor (`Dump`) der alle Dateien, Verweise und Ordner in hierarchischer Form ausgibt. Die Ausgabe soll sowohl auf der Standardausgabe als auch in einer Datei möglich sein!

Implementieren Sie einen Visitor (`FilterFiles`) der alle Dateien herausfiltert deren aktuelle Größe innerhalb eines vorgegebenen minimalen und maximalen Wertes liegt. Ein zusätzlicher Filter soll alle Verweise herausfiltern. Die Filter sollen in der Lage sein, alle gefilterten Dateien mit ihrem vollständigen Pfadnamen auszugeben! Bei der Filterung von Verweisen muss zusätzlich auch der

Name des Elementes auf das verwiesen wird ausgegeben werden.

Implementieren Sie einen Testtreiber der ein hierarchisches Dateisystem mit mehreren Ebenen erzeugt und die zu implementierenden Besucher ausführlich testet!

Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese. Verfassen Sie weiters eine Systemdokumentation (entsprechend den Vorgaben aus Übung1)!

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!

Systemdokumentation
Projekt Filesystem

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 26. November 2025

Inhaltsverzeichnis

1 Organisatorisches 6
1.1 Team . . . 6
1.2 Aufteilung der Verantwortlichkeitsbereiche . . . 6
1.3 Aufwand . . . 7

2 Anforderungsdefinition (Systemspezifikation) 8
2.1 Systemüberblick . . . 8
2.2 Funktionale Anforderungen . . . 8
2.2.1 Dateien . . . 8
2.2.2 Ordner . . . 9
2.2.3 Verweise . . . 9
2.3 Erzeugung der Elemente . . . 9
2.4 Besucher (Visitor) Anforderungen . . . 9
2.4.1 Visitor: Dump . . . 9
2.4.2 Visitor: FilterFiles . . . 10
2.5 Testanforderungen . . . 10

3 Systementwurf 11
3.1 Klassendiagramm . . . 11
3.2 Designentscheidungen . . . 12

4 Dokumentation der Komponenten (Klassen) 12

5 Testprotokollierung 13

6 Quellcode 14
6.1 Object.hpp . . . 14
6.2 FSObjectFactory.hpp . . . 15
6.3 FSObjectFactory.cpp . . . 16
6.4 Filesystem.hpp . . . 17
6.5 Filesystem.cpp . . . 18
6.6 FSObject.hpp . . . 19
6.7 FSObject.cpp . . . 20
6.8 File.hpp . . . 21
6.9 File.cpp . . . 22

6.10 IFolder.hpp 23
6.11 Folder.hpp 24
6.12 Folder.cpp 25
6.13 Link.hpp 26
6.14 Link.cpp 27
6.15 IVisitor.hpp 28
6.16 FilterVisitor.hpp 29
6.17 FilterVisitor.cpp 30
6.18 FilterFileVisitor.hpp 31
6.19 FilterFileVisitor.cpp 32
6.20 FilterLinkVisitor.hpp 33
6.21 FilterLinkVisitor.cpp 34
6.22 DumpVisitor.hpp 35
6.23 DumpVisitor.cpp 36
6.24 main.cpp 37
6.25 Test.hpp 38

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.at
- Simon Vogelhuber, Matr.-Nr.: S2410306014, E-Mail: Simon.Vogelhuber@fh-hagenberg.at

1.2 Aufteilung der Verantwortlichkeitsbereiche

- Simon Offenberger
  - Design Klassendiagramm
  - Implementierung und Test der Klassen:
    - IVisitor,
    - FilterVisitor,
    - FilterFileVisitor,
    - FilterLinkVisitor,
    - DumpVisitor und
    - FSObjectFactory
  - Implementierung des Testtreibers
  - Dokumentation
- Simon Vogelhuber
  - Design Klassendiagramm

  - Implementierung und Komponententest der Klassen:
    * FSObject
    * File,
    * iFolder,
    * Folder und
    * Link
  - Implementierung des Testtreibers
  - Dokumentation

1.3 Aufwand

- Simon Offenberger: geschätzt 7 Ph / tatsächlich 9 Ph
- Simon Vogelhuber: geschätzt 8 Ph / tatsächlich 7 Ph

2 Anforderungsdefinition (Systemspezifikation)

Das zu entwickelnde System dient der Simulation eines einfachen Dateisystems für ein eingebettetes System. Ziel ist es, die Struktur und das Verhalten eines hierarchischen Dateisystems softwaretechnisch abzubilden und durch geeignete Entwurfsmuster (Composite, Factory, Visitor) erweiterbar und wartbar zu gestalten. Die Anforderungen ergeben sich aus der gegebenen Systemspezifikation der Übung und beschreiben die grundlegenden funktionalen Eigenschaften des Systems sowie alle wesentlichen Rahmenbedingungen für das Verständnis.

2.1 Systemüberblick

Das System verwaltet drei Arten von Dateisystemelementen:

- **Dateien**
- **Ordner**
- **Verweise** (Referenzen auf Dateien, Ordner oder weitere Verweise)

Diese Elemente bilden gemeinsam eine hierarchische Struktur, in der Ordner beliebige Kombinationen dieser Elemente enthalten können. Jedes Element besitzt einen Namen, der nachträglich veränderbar ist.

2.2 Funktionale Anforderungen

2.2.1 Dateien

Eine Datei verfügt über folgende unveränderliche Eigenschaften, die bei ihrer Erzeugung festgelegt werden:

- Blockgröße auf dem Speichermedium (Bytes)

- Anzahl reservierter Blöcke

Zusätzlich wird die aktuelle Dateigröße in Bytes verwaltet. Das Schreiben in eine Datei erfolgt über:

- `Write(size_t const bytes)`

Die Datei darf niemals größer werden als der durch die reservierten Blöcke bereitgestellte Speicher.

2.2.2 Ordner

Ein Ordner kann beliebig viele Dateien, Verweise und weitere Ordner enthalten. Er bildet die Grundlage des hierarchischen Dateisystems.

2.2.3 Verweise

Ein Verweis referenziert exakt ein Zielobjekt (Datei, Ordner oder weiteren Verweis). Der Name des Verweises kann verändert werden, zusätzlich muss der Name des Zielobjekts im Rahmen der Filterausgabe ausgegeben werden.

2.3 Erzeugung der Elemente

Für die Erstellung aller Dateisystemelemente ist eine einfache **Fabrik** zu implementieren. Diese kapselt die Instanziierungslogik und stellt sicher, dass die Objekterzeugung einheitlich erfolgt.

2.4 Besucher (Visitor) Anforderungen

2.4.1 Visitor: Dump

- Gibt die gesamte Dateisystemhierarchie aus.

- Ausgabe sowohl auf der Standardausgabe als auch in einer Datei.
- Muss Dateien, Ordner und Verweise in strukturierter Form darstellen.

2.4.2 Visitor: FilterFiles

- Filtert Dateien anhand eines minimalen und maximalen Größenschwellwerts.
- Filtert optional alle Verweise.
- Ausgabe aller gefilterten Dateien mit ihrem vollständigen Pfad.
- Bei Verweisen muss zusätzlich der Name des referenzierten Zielobjekts ausgegeben werden.

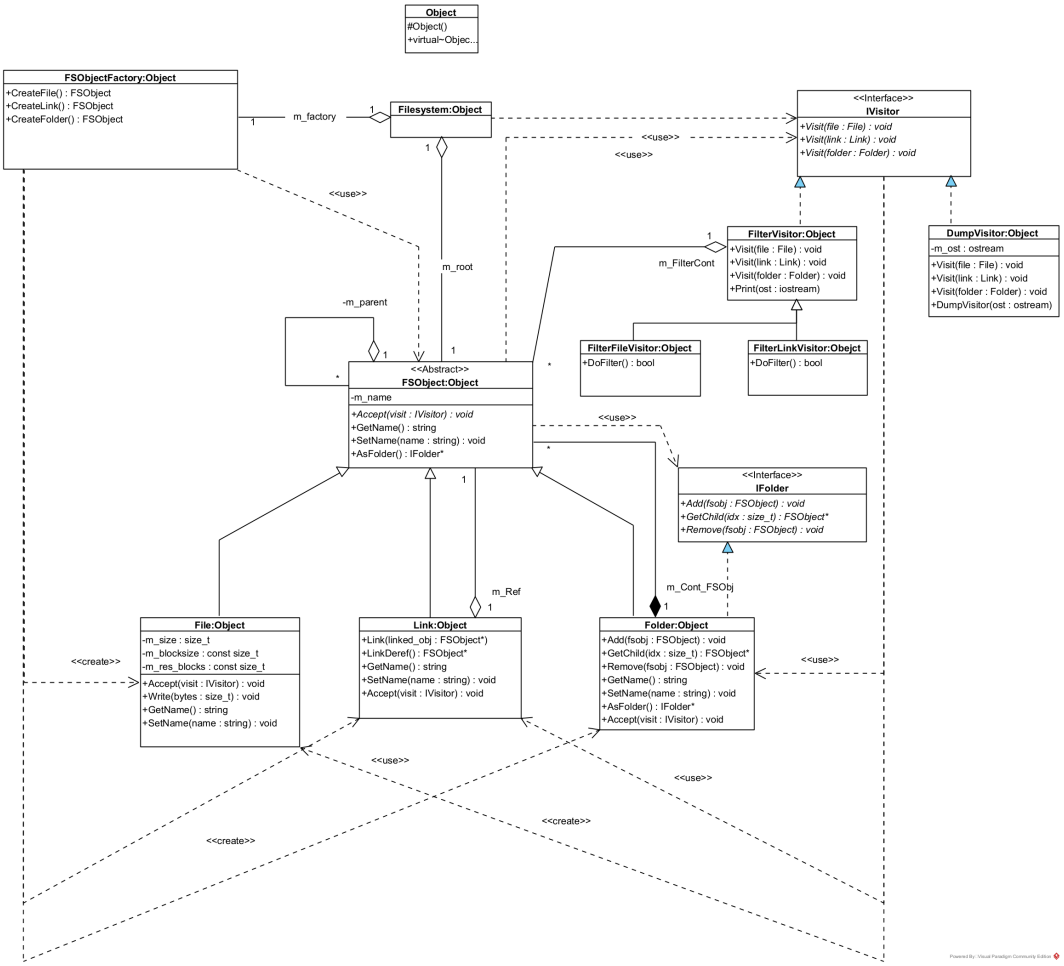
2.5 Testanforderungen

Ein Testtreiber muss:

- ein mehrstufiges hierarchisches Dateisystem erzeugen,
- alle implementierten Besucher umfassend testen,
- die Funktionsweise demonstrieren und Testausgaben bereitstellen.

3 Systementwurf

3.1 Klassendiagramm



3.2 Designentscheidungen

4 Dokumentation der Komponenten (Klassen)

Die HTML-Startdatei befindet sich im Verzeichnis ./../doxy/html/index.html

5 Testprotokollierung

6 Quellcode

6.1 Object.hpp

```
1  #ifndef OBJECT_H
2  #define OBJECT_H
3
4  #include <string>
5
6  class Object{
7  public:
8
9     inline static const std::string ERROR_NULLPTR = "ERROR Nullptr";
10
11 protected:
12
13
14    Object(){};
15 public:
16    virtual ~Object(){}
17 };
18
19 #endif // OBJECT_H
```

6.2 FSObjectFactory.hpp

```
1  /*****************************************************************//**
2   * \file
3   * \brief
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef FS_OBJECT_FACTORY_HPP
9  #define FS_OBJECT_FACTORY_HPP
10
11 #include "Object.h"
12
13 class FSObjectFactory : public Object
14 {
15 public:
16 private:
17 };
18
19 #endif
```

6.3 FSObjectFactory.cpp

6.4 Filesystem.hpp

```
1  /*****************************************************************//**
2   * \file
3   * \brief
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef FILE_HPP
9  #define FILE_HPP
10
11 #include "FSObject.hpp"
12
13 class File : public FSObject
14 {
15 public:
16
17
18
19
20 private:
21    size_t m_size;
22    const size_t m_blocksize;
23    const size_t res_blocks;
24 };
25 #endif
```

6.5 Filesystem.cpp

6.6 FSObject.hpp

```cpp
/*****************************************************************//**
 * \file
 * \brief
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef FS_OBJECT_HPP
#define FS_OBJECT_HPP

#include "Object.h"
#include "IVisitor.hpp"
#include "IFolder.hpp"

#include <memory>

class FSObject : public Object
{
public:
   using Sptr = std::shared_ptr<FSObject>;
   using Wptr = std::weak_ptr<FSObject>;

   virtual void Accept(IVisitor& visit) = 0;

   virtual IFolder::Sptr AsFolder();

   std::string_view GetName();

   void SetName(std::string_view name);

private:
};

#endif
```

6.7 FSObject.cpp

6.8 File.hpp

```
1  /*****************************************************************//**
2   * \file
3   * \brief
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef FILE_HPP
9  #define FILE_HPP
10
11 #include "FSObject.hpp"
12
13 class File : public FSObject
14 {
15 public:
16
17
18
19
20 private:
21    size_t m_size;
22    const size_t m_blocksize;
23    const size_t res_blocks;
24 };
25 #endif
```

6.9 File.cpp

6.10 IFolder.hpp

```cpp
/*****************************************************************//**
 * \file
 * \brief
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef IFOLDER_HPP
#define IFOLDER_HPP

#include "FSObject.hpp"

#include <memory>

class IFolder
{
public:

   using Sptr = std::shared_ptr<IFolder>;

   virtual void Add(FSObject::Sptr fsobj) = 0;

   virtual FSObject::Sptr GetChild(size_t idx) = 0;

   virtual void Remove(FSObject::Sptr fsobj) = 0;

   virtual ~IFolder() = default;

private:
};

#endif
```

6.11 Folder.hpp

```
/*****************************************************************//**
 * \file
 * \brief
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef FOLDER_HPP
#define FOLDER_HPP

#include "IFolder.hpp"
#include <memory>

class Folder : public IFolder, public Object
{
public:

   using Sptr = std::shared_ptr<Folder>;
   using Wptr = std::shared_ptr<Folder>;

   virtual void Add(FSObject::Sptr fsobj) override;

   virtual FSObject::Sptr GetChild(size_t idx) override;

   virtual void Remove(FSObject::Sptr fsobj) override;

private:
};

#endif
```

6.12 Folder.cpp

6.13 Link.hpp

```
1  /*****************************************************************//**
2   * \file
3   * \brief
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef LINK_HPP
9  #define LINK_HPP
10
11 #include "FSObject.hpp"
12
13 class Link: public FSObject
14 {
15 public:
16
17    explicit Link(FSObject::Sptr linked_obj);
18
19    FSObject::Sptr operator*();
20
21 private:
22
23    FSObject::Wptr m_Ref;
24 };
25
26 #endif
```

6.14 Link.cpp

6.15 IVisitor.hpp

```cpp
/*****************************************************************//**
 * \file
 * \brief
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef IVISITOR_HPP
#define IVISITOR_HPP

#include "Link.hpp"
#include "File.hpp"
#include "Folder.hpp"

class IVisitor
{
public:

   virtual void Visit(const Folder & folder)=0;

   virtual void Visit(const File & file)=0;

   virtual void Visit(const Link & Link)=0;

   virtual ~Visitor() = default;

private:
};

#endif
```

6.16 FilterVisitor.hpp

```cpp
/*****************************************************************//**
 * \file
 * \brief
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef FILTER_VISITOR_HPP
#define FILTER_VISITOR_HPP

#include "IVisitor.hpp"
#include "FSObject.hpp"

#include <vector>

class FilterVisitor : public IVisitor
{
public:

   virtual void Visit(const Folder& folder) override;

   virtual void Visit(const File& file) override;

   virtual void Visit(const Link& Link) override;

protected:

   virtual bool DoFilter()=0;

   using TContFSobj = std::vector<FSObject::Wptr>;

private:

   TContFSobj m_FilterCont;
};

#endif
```

6.17 FilterVisitor.cpp

6.18 FilterFileVisitor.hpp

```
1  /*****************************************************************//**
2   * \file
3   * \brief
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef FILTER_VISITOR_HPP
9  #define FILTER_VISITOR_HPP
10
11 #include "FilterVisitor.hpp"
12
13 class FilterFileVisitor : public FilterVisitor
14 {
15 public:
16
17 protected:
18
19    virtual bool DoFilter() override;
20
21 private:
22
23
24
25 };
26
27 #endif
```

6.19 FilterFileVisitor.cpp

6.20 FilterLinkVisitor.hpp

```
1  /*****************************************************************//**
2   * \file
3   * \brief
4   *
5   * \author Simon
6   * \date   November 2025
7   *********************************************************************/
8  #ifndef FILTER_LINK_VISITOR_HPP
9  #define FILTER_LINK_VISITOR_HPP
10
11 #include "FilterVisitor.hpp"
12
13 class FilterLinkVisitor : public FilterVisitor
14 {
15 public:
16
17 protected:
18    virtual bool DoFilter() override;
19
20 private:
21 };
22
23 #endif
```

6.21 FilterLinkVisitor.cpp

6.22 DumpVisitor.hpp

```
/*****************************************************************//**
 * \file
 * \brief
 *
 * \author Simon
 * \date   November 2025
 *********************************************************************/
#ifndef IVISITOR_HPP
#define IVISITOR_HPP

#include "IVisitor.hpp"

#include <iostream>

class DumpVisitor : public IVisitor
{
public:

   DumpVisitor(std::ostream& ost) : m_ost{ ost } {}

   virtual void Visit(const Folder& folder) override;

   virtual void Visit(const File& file) override;

   virtual void Visit(const Link& Link) override;

private:

   std::ostream & m_ost;
};

#endif
```

6.23 DumpVisitor.cpp

6.24 main.cpp

```
1
2
3 int main()
4 {
5
6 }
```

6.25 Test.hpp

```
/*****************************************************************//**
 * \file   Test.hpp
 * \brief  File that provides a Test Function with a formated output
 *
 * \author Simon
 * \date   April 2025
 *********************************************************************/
#ifndef TEST_HPP
#define TEST_HPP

#include <string>
#include <iostream>
#include <vector>
#include <list>
#include <queue>
#include <forward_list>

#define ON 1
#define OFF 0
#define COLOR_OUTPUT ON

// Definitions of colors in order to change the color of the output stream.
const std::string colorRed = "\x1B[31m";
const std::string colorGreen = "\x1B[32m";
const std::string colorWhite = "\x1B[37m";

inline std::ostream& RED(std::ostream& ost) {
   if (ost.good()) {
      ost << colorRed;
   }
   return ost;
}
inline std::ostream& GREEN(std::ostream& ost) {
   if (ost.good()) {
      ost << colorGreen;
   }
   return ost;
}
inline std::ostream& WHITE(std::ostream& ost) {
   if (ost.good()) {
      ost << colorWhite;
   }
   return ost;
}

inline std::ostream& TestStart(std::ostream& ost) {
   if (ost.good()) {
      ost << std::endl;
      ost << "****************************************" << std::endl;
      ost << "          TESTCASE START          " << std::endl;
      ost << "****************************************" << std::endl;
      ost << std::endl;
   }
   return ost;
}

inline std::ostream& TestEnd(std::ostream& ost) {
   if (ost.good()) {
      ost << std::endl;
      ost << "****************************************" << std::endl;
      ost << std::endl;
   }
   return ost;
}

inline std::ostream& TestCaseOK(std::ostream& ost) {

#if COLOR_OUTPUT
   if (ost.good()) {
      ost << colorGreen << "TEST OK!!" << colorWhite << std::endl;
   }
#else
```

```
73     if (ost.good()) {
74        ost <<  "TEST␣OK!!" <<  std::endl;
75     }
76  #endif // COLOR_OUTPUT
77
78     return ost;
79  }
80
81  inline std::ostream& TestCaseFail(std::ostream& ost) {
82
83  #if COLOR_OUTPUT
84     if (ost.good()) {
85        ost << colorRed << "TEST␣FAILED␣!!" << colorWhite << std::endl;
86
87     }
88  #else
89     if (ost.good()) {
90        ost << "TEST␣FAILED␣!!" << std::endl;
91
92     }
93  #endif // COLOR_OUTPUT
94
95     return ost;
96  }
97
98  /**
99     * \brief function that reports if the testcase was successful.
100    *
101    * \param testcase String that indicates the testcase
102    * \param succsessful true -> reports to cout test OK
103    * \param succsessful false -> reports test failed
104    */
105 template <typename T>
106 bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
107    if (ostr.good()) {
108 #if COLOR_OUTPUT
109       if (expected == result) {
110          ostr << testcase << std::endl <<  colorGreen << "[Test␣OK]␣" << colorWhite <<"Result:␣(Expected:␣" << std::boolalpha << expect
   << std::noboolalpha << std::endl << std::endl;
111       }
112       else {
113          ostr << testcase << std::endl << colorRed << "[Test␣FAILED]␣" << colorWhite << "Result:␣(Expected:␣" << std::boolalpha << expe
   << std::noboolalpha << std::endl << std::endl;
114       }
115 #else
116       if (expected == result) {
117          ostr << testcase << std::endl << "[Test␣OK]␣"  << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣==" << "␣Result:␣"
118       }
119       else {
120          ostr << testcase << std::endl  << "[Test␣FAILED]␣"  << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣!=" << "␣Resul
121       }
122 #endif
123       if (ostr.fail()) {
124          std::cerr << "Error:␣Write␣Ostream" << std::endl;
125       }
126    }
127    else {
128       std::cerr << "Error:␣Bad␣Ostream" << std::endl;
129    }
130    return expected == result;
131 }
132
133 template <typename T1, typename T2>
134 std::ostream& operator<< (std::ostream& ost,const std::pair<T1,T2> & p) {
135    if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
136    ost << "(" << p.first << "," << p.second << ")";
137    return ost;
138 }
139
140 template <typename T>
141 std::ostream& operator<< (std::ostream& ost,const std::vector<T> & cont) {
142    if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
143    std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
144    return ost;
145 }
```

```
146
147 template <typename T>
148 std::ostream& operator<< (std::ostream& ost,const std::list<T> & cont) {
149    if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
150    std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
151    return ost;
152 }
153
154 template <typename T>
155 std::ostream& operator<< (std::ostream& ost,const std::deque<T> & cont) {
156    if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
157    std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
158    return ost;
159 }
160
161 template <typename T>
162 std::ostream& operator<< (std::ostream& ost,const std::forward_list<T> & cont) {
163    if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
164    std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
165    return ost;
166 }
167
168
169 #endif // !TEST_HPP
```