5. Semester, 1. Prüfung

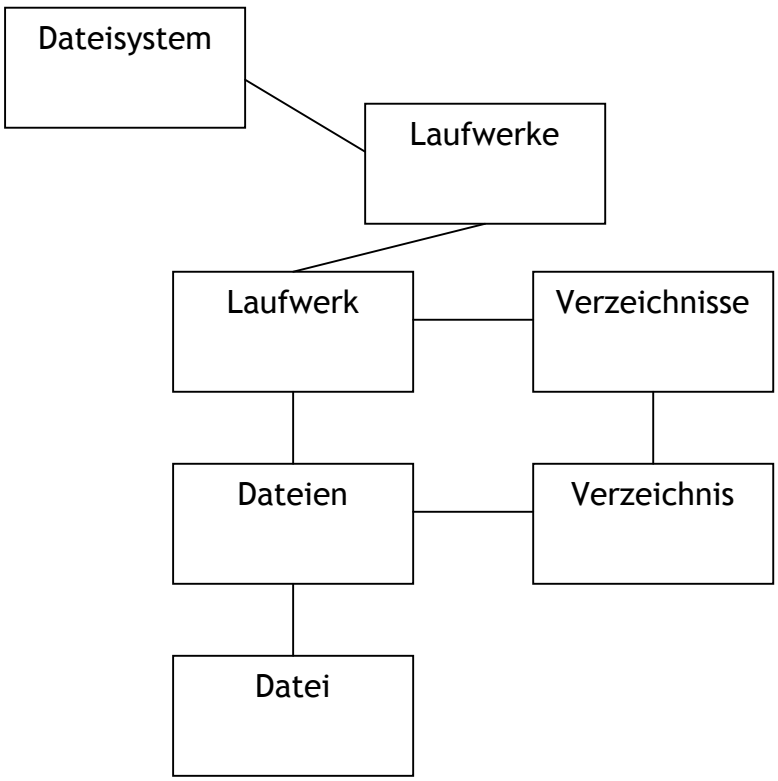
Name	

- Die gesamte Prüfung bezieht sich auf die Programmierung in C++!
- Prüfungsdauer: 90 Minuten
- Mit Kugelschreiber oder Tinte schreiben
- Lösungen können direkt auf die Aufgabenblätter geschrieben werden
- PC's sind nicht erlaubt
- Unterlagen und Bücher sind erlaubt
- Achte auf Details wie Punkte, Kommas und Semikolons

Aufgabe	Punkte	
Analyse und Design	13	
Exceptions	12	
Programmverständnis	14	
Weitere Fragen	11	
Total	50	

1. Analyse und Design

a) Zeichne für folgende Problemstellung die möglichen Klassen auf (als einfaches Klassendiagramm) und verdeutliche ihre Abhängigkeiten mit Linien untereinander. Du musst keine Methoden, Funktionen oder Datenelemente definieren.
Ein Dateisystem wie es auf Windows verwendet wird ist hierarchisch aufgebaut. Das Dateisystem beinhaltet Laufwerke. In einem Laufwerk können sich Dateien und Verzeichnisse befinden. Ein Verzeichnis kann eine oder mehrere Dateien beinhalten. Ein Verzeichnis kann auch andere Verzeichnisse beinhalten, was aber zu keinen besonderen Klassen führt. (7 Punkte).
Definiere für die Klassen, die aus einer Mehrzahl einer anderen Klasse bestehen eine Kollektion, indem du mit typedef und einer Containerklasse aus der STL einen Typen definierst.
(typedef std::xxx …). (3 Punkte)
(Total 10 Punkte)




```
#include <vector>
typedef std::vector<Laufwerk> Laufwerke;
typedef std::vector<Verzeichnis> Verzeichnisse;
typedef std::vector<Datei> Dateien;
```

b) Es werden die Minimal-Temperaturen für jeden Tag in einem Monat gesammelt. Diese Daten, die als **double** vorliegen, werden später für Abfragen gebraucht, bei denen man für ein beliebiges Datum direkt auf den entsprechenden Wert zugreifen möchte, ohne durch alle Werte hindurch zu schreiten (iterieren).
Was für eine Container-Klasse, die du kennst würdest du hier einsetzen? (1 Punkt)
Definiere einen neuen Datentypen mit *typedef*, der solche Elemente in einem Container beinhaltet. Vergiss nicht die nötigen *#include*-Anweisungen und die *namespace*-Spezifizierer
(1 Punkt)
Erzeuge eine Variable von diesem Datentypen. (1 Punkt)
(Total 3 Punkte)

```
vector

#include <vector>

typedef std::vector<double> Temperaturen;

Temperaturen minimalTemperaturen;
```

2. Exceptions

a) Gegeben sei folgende Klasse *MathException*. Diese Klasse dient dazu, Ausnahmen in mathematischen Funktionen anzuzeigen, indem ein Objekt dieser Klasse geworfen wird.
Schreibe die Funktion *Verdoppeln* fertig. Diese Funktion berechnet das Doppelte (Argument * 2) des Argumentes (unsigned short), welches der Funktion übergeben wird und gibt das Ergebnis zurück. Falls das Argument aber grösser als 32767 (=2^15-1) ist, soll die Funktion eine *MathException* werfen, da das einen Überlauf erzeugt. Beachte den Konstruktor von *MathException*. Schreibe die Funktion *Verdoppeln* auf ungefähr 6 Zeilen. (3 Punkte)
Schreibe darunter eine einfache main-Funktion, welche die Funktion *Verdoppeln* aufruft, verwende aber Ausnahmebehandlung und werte die gefangene Ausnahme aus. Code etwa 9 Zeilen (3 Punkte)
(Total 6 Punkte)

```
#include <iostream>
#include <string>

enum ErrorType
{
    Overflow,
    DivisionByZero,
    Underflow
};

class MathException
{
public:
    MathException(ErrorType type)
        :_type(type)
    {
    }

    std::string getErrorText() const
    {
        std::string error;
        switch(_type)
        {
            case Overflow:
                error = "Overflow";
                break;
            case DivisionByZero:
                error = "Division by zero";
                break;
            case Underflow:
                error = "Underflow";
                break;
            default:
                break;
        }
        return error;
    }
private:
    ErrorType _type;
};
```

```
unsigned short Verdoppeln(unsigned short argument)
{
	// Hier Code ergänzen, der das argument prüft
	// und evtl. eine MathException wirft, sonst
	// aber die Berechnung durchführt.
	if(argument > 32767)
	{
		throw MathException(Overflow);
	}
	return 2*argument;
}
```

```
// Hier die Funktion main ergänzen!

int main()
{
	try
	{
		unsigned short test = Verdoppeln(342);
	}
	catch(const MathException& e)
	{
		std::cout << e.getErrorText() << std::endl;
	}
	return 0;
}
```

b) Betrachte folgende Klasse *Dictionary*. Sie wird verwendet um Wörter von Deutsch nach Englisch zu übersetzen. Schreibe ein kleines Programm (main-Funktion), das
1. ein Dictionary Objekt erzeugt
2. ein Wortpaar in den Diktionär einfügt (addNew)
3. Dieses soeben eingefügte Wort nachsieht (lookup)
4. den Test durchführt ob ein Wort zu einem andern gehört (test)
Baue auch Ausnahmebehandlung ein!
Es gibt etwa 13 Zeilen Code anzufügen. (6 Punkte)

```
#include <map>
#include <string>
#include <iostream>

class Dictionary
{
private:
    std::map<std::string, std::string> _words;
public:
    void addNew(const std::string& german,
                const std::string& english)
    {
        if(_words.end() != _words.find(german))
        {
            std::string error("Word already defined");
            throw error;
        }
        _words[german] = english;
    }
    std::string lookup(const std::string& german)
    {
        if(_words.end() == _words.find(german))
        {
            std::string error("Word not found");
            throw error;
        }
        return _words[german];
    }
    bool test(const std::string& german,
              const std::string& english)
    {
        if(_words.end() == _words.find(german))
        {
            std::string error("Word not found");
            throw error;
        }
        std::string& word = _words[german];
        return (word == english);
    }
};
```

```
int main()
{
	// Hier Code ergänzen
	try
	{
		Dictionary meinDix;
		meinDix.addNew(„hallo“, „hello“);
		std::string wort = meinDix.lookup(„hallo“);
		bool gleich = meinDix.test(„hallo“, „hello“);
	}
	catch(const std::string& exception)
	{
		std::cout << „Fehler : “ << exception << std::endl;
	}
	return 0;
}
```

3. Programmverständnis

a) Was gibt folgendes Programm aus? (3 Punkte)

```
#include <list>
#include <string>
#include <iostream>

int main()
{
    const char* Namen[] =
    {
        "Zacharias",
        "Johann",
        "Fred",
        "Sepp",
        "Adam"
    };

    std::list<std::string> namensliste;
    const long count = sizeof(Namen) / sizeof(Namen[0]);
    for(long i = 0; i < count-1; ++i)
    {
        namensliste.push_back(Namen[i]);
    }
    namensliste.sort();
    namensliste.push_back(Namen[count-1]);
    std::list<std::string>::iterator it = namensliste.begin();
    std::list<std::string>::iterator end = namensliste.end();
    for( ; it != end; ++it)
    {
        std::cout << *it << std::endl;
    }
    return 0;
}
```

```
Fred
Johann
Sepp
Zacharias
Adam
```

b) Schreibe im folgenden Codeabschnitt die *while*-Schleife als *for*-Schleife, so dass genau der gleiche Code abgearbeitet wird.
(3 Punkte)

```
std::vector<int> zahlen;
std::vector<int>::iterator it = zahlen.begin();
std::vector<int>::iterator end = zahlen.end();

while(it != end)
{
    std::cout << *it << std::endl;
    ++it;
}


std::vector<int>::iterator it = zahlen.begin();
std::vector<int>::iterator end = zahlen.end();
for( ; it != end; ++it)
{
    std::cout << *it << std::endl;
}
```

c) Gegeben ist folgende Typdefinition für *Namen*. Definiere einen weiteren Datentypen *Namensliste* unter Verwendung der list-Klasse aus der STL (1 Punkt).
Schreibe eine Funktion *SortiereNamen*, die als Argument eine solche *Namensliste* erhält und diese sortiert (Methode sort aufrufen).
Beachte dabei wie das Argument übergeben wird!
(3 Punkte) (Total 4 Punkte)

```
#include <string>
#include <list>

typedef std::string Namen;

typedef std::list<Namen> Namensliste;


void SortiereNamen(Namensliste& namen)
{
    namen.sort();
}
```

d) Ergänze in folgender Klasse *Dictionary* eine Methode *print*, die alle Einträge in der Kollektion *_words* auf dargestellte Art in der Console ausgibt. Die Methode kann direkt inline definiert werden. Ca. 9 Zeilen Code. (4 Punkte)

```
// Hier die gewünschte Ausgabe
Deutsches Wort : Englisches Wort
Hallo : Hello
Folgen : Follow

// Hier die Klasse Dictionary
#include <map>
#include <string>
#include <iostream>

class Dictionary
{
private:
    typedef std::map<std::string, std::string> Words;
    Words _words;
public:
    // Hier Methode print ergänzen
    void print()
    {
        Words::iterator it = _words.begin();
        Words::iterator end = _words.end();
        for( ; it != end; ++it)
        {
            std::cout << it->first << " : ";
            std::cout << it->second << std::endl;
        }
    }









};
```

4. Weitere Fragen

a) Schreibe eine Funktion *Addition*, die zwei Variablen vom Typ *unsigned int* addiert und das Ergebnis als Return-Wert (auch unsigned int) zurückgibt.
(3 Punkte)

```
unsigned int Addition(unsigned int a, unsigned int b)
{
   return a+b;
}
```

b) Schreibe nun eine template-Funktion *Addition*, die man für beliebige Datentypen verwenden kann. (4 Punkte)

```
template<class T>
T Addition(const T& a, const T& b)
{
   return a + b;
}
```

c) Was schreibst du in folgende Funktion, wenn du nicht willst, dass das Argument den Wert 0 hat? Bau etwas ein, das das Programm zum Absturz bringt, falls sich ein Programmierer nicht daran hält!
(2 Punkte)

```
#include <crtdbg.h> // oder #include <cassert>
// Bitte diese Funktion nicht mit 0 als Argument aufrufen!
void Funktion(unsigned int argument)
{
    // Ergänze hier den „Absturz“ falls das Argument 0 ist
    _ASSERT(argument != 0);



}
```

d) Mit welcher Methode kann man in einer list oder einem vector ein Element einfügen? (1 Punkt)

push_back

e) Willst Du fehlerfreie Programme schreiben? (1 Punkt) Ja