5. Semester, 2. Prüfung

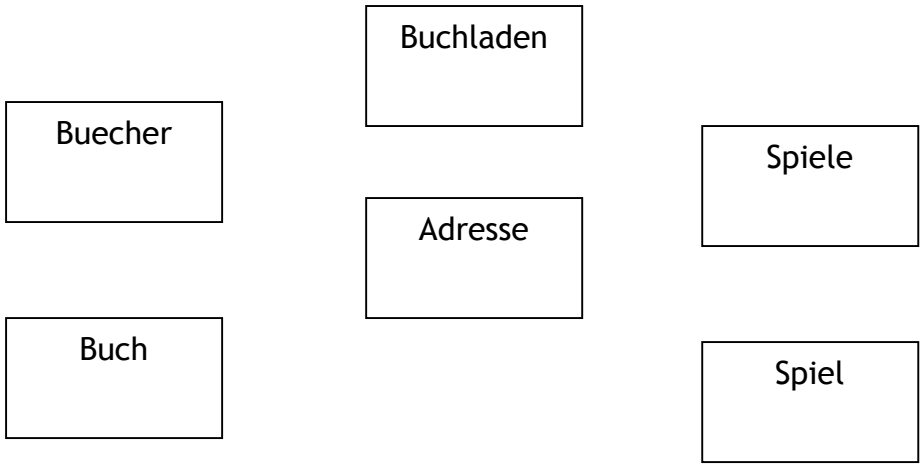
Name	

- 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	10	
Windows API	17	
Templates	5	
Programmverständnis	10	
Weitere Fragen	8	
Total	50	

1. Analyse und Design

a) Versuche aus folgender Beschreibung ein einfaches Klassendiagramm zu erstellen. Finde die Klassen, oder falls vorhanden die Attribute zu einer Klasse. Klassen, die in irgend einer Beziehung zueinander stehen, können mit einer Linie verbunden werden.
Ein Buchladen verkauft Bücher, jedes Buch hat einen Preis. Dieser Buchladen verkauft auch Computerspiele, wobei jedes Computerspiel auch einen Preis hat. Der Buchladen hat selbstverständlich eine Adresse. (6 Punkte)
Definiere für die Sammlungen von Objekten, die du erkennst den jeweiligen Datentypen mit der passenden Containerklasse aus der STL! (4 Punkte) (Total 10 Punkte)




```
#include <vector>

typedef std::vector<Buch> Buecher;
typedef std::vector<Spiel> Spiele;
```

2. Windows API

a) Öffne den seriellen Port für synchrones (nicht OVERLAPPED) Lesen und Schreiben (2 Punkte).
Schreibe das Zeichen ‚A' in den seriellen Port. (2 Punkte)
Schreibe das Wort „Hallo" in den seriellen Port. (2 Punkte)
Schliesse den Port wieder (1 Punkt) (Total 7 Punkte)

```
#include <windows.h>

int main()
{
    HANDLE h = CreateFile(„COM1,
                          GENERIC_READ | GENERIC_WRITE,
                          0,
                          0,
                          OPEN_EXISTING,
                          FILE_ATTRIBUTE_NORMAL,
                          0);
    DWORD written = 0;
    WriteFile(h,
              „A“,
              1,
              &written,
              0);
    WriteFile(h,
              „Hallo“,
              5,
              &written,
              0);
    CloseHandle(h);
    return 0;
}
```

b) Definiere ein Element der Struktur, mit der die Timeouts beim seriellen Port definiert werden und initialisiere sie mit 0! (2 Punkte)

```
COMMTIMEOUTS cto = { 0 };
```

c) Schreibe eine Klasse *Event* mit einem Konstruktor und einem Destruktor. Diese Klasse hat den Zweck ein Windows Event so zu kapseln, dass alle Resourcen (z.B. Handles) sicher wieder freigegeben werden. Übergebe dem Konstruktor als Argument den Datentypen mit dem ein Event von Windows identifiziert wird. (4 Punkte)
Ein Event kann gesetzt (set) und zurückgesetzt werden (reset). Schreibe also zwei Methoden, mit denen das Event gesetzt und zurückgesetzt werden kann. (4 Punkte)
Definiere einen Typumwandlungsoperator, der das Event-Objekt in den zugrundeliegenden Windows-Datentypen (siehe Konstruktor-Argument) umwandeln kann. (2 Punkte)
Um unsere Klasse zu komplettieren fehlt noch der Zuweisungsoperator! Definiere diesen so, dass er als Argument den Windows-Datentypen hat, mit dem ein Event identifiziert wird (wie Konstruktor). (2 Punkte)
Du kannst alles inline definieren oder aber zuerst die Klasse und danach die Methoden getrennt davon definieren.
(Total 8 Punkte)

```
class Event
{
    public:
        Event(HANDLE h) : m_h(h) {}
        ~Event() { CloseHandle(h); }

        void set() { SetEvent(m_h); }
        void reset() { ResetEvent(m_h); }

        operator HANDLE() { return m_h; }

        void operator=(HANDLE h) { m_h = h; }

    private:
        HANDLE m_h;
};
```

3. Templates

a) Betrachte folgenden Code, der jeweils ein Element in einer Kollektion findet.
Schreibe darunter eine template-Funktion *find*. Dieser Funktion wird nicht nur eine Kollektion übergeben sondern auch ein Element, vom Datentypen, wie er in der Kollektion ist. Die Funktion soll einen iterator auf das gefundene Element zurückgeben. Darunter (nächste Seite) steht das *main*, wie es nachher aussehen soll. (5 Punkte)

```
#include <vector>
#include <string>

using namespace std;

int main()
{
    vector<double> someDoubles;
    // hier würde Code stehen, der viele Elemente einfügt
    double dElementToFind = 3.9;

    vector<double>::iterator dit = someDoubles.begin();
    vector<double>::iterator dend = someDoubles.end();
    for( ; dit != dend; ++dit)
    {
        if(*dit == dElementToFind)
        {
            // der Iterator dit zeigt jetzt
            // auf das gefundene Element
            break;
        }
    }

    vector<string> someStrings;
    // hier würde Code stehen, der viele Elemente einfügt
    string sElementToFind = "test";

    vector<string>::iterator sit = someStrings.begin();
    vector<string>::iterator send = someStrings.end();
    for( ; sit != send; ++sit)
    {
        if(*sit == sElementToFind)
        {
            // der Iterator sit zeigt jetzt
            // auf das gefundene Element
            break;
        }
    }
    return 0;
};
```

```
int main()
{
    vector<double> someDoubles;
    // hier würde Code stehen, der viele Elemente einfügt
    double dFind = 3.9;
    vector<double>::iterator dit = find(someDoubles, dFind);

    vector<string> someStrings;
    // hier würde Code stehen, der viele Elemente einfügt
    string sFind = "test";
    vector<string>::iterator sit = find(someStrings, sFind);

    return 0;
}

// Hier template-Funktion find anfügen

template<class T>
vector<T>::iterator find(vector<T>& collection, const T& el)
{
    vector<T>::iterator it = collection.begin();
    vector<T>::iterator end = collection.end();
    for( ; it != end; ++it)
    {
        if(el == *it)
        {
            break;
        }
    }
    return it;
}
```

4. Programmverständnis

a) Was gibt folgendes Programm auf der Konsole aus (beachte auch den Zeilenumbruch)? (4 Punkte)

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

using namespace std;
// map von string nach long
// als Typ definieren

typedef map<string, long> BankCodes;

int main()
{
    BankCodes meineCodes;

    meineCodes["ZKB"] = 12;
    meineCodes["Visa"] = 22;
    meineCodes["Post"] = 16;
    meineCodes["UBS"] = 45;

    string z("ZKB");
    BankCodes::iterator it = meineCodes.begin();
    BankCodes::iterator end = meineCodes.end();
    for( ; it != end; ++it)
    {
        cout << setbase(16);  // Achtung Achtung Achtung
        if(z == it->first || 45 == it->second)
        {
            cout << it->first;
            cout << " : ";
            cout << it->second << endl;
        }
    }

    return 0;
}
```

```
UBS : 2d
ZKB :  c
```

b) Definiere eine Klasse *SystemFehler*. Diese Klasse ist eine Sammlung von Fehlern, die als *std::string* vorliegen. Das heisst die Klasse *SystemFehler* soll von einer geeigneten Container-Klasse ableiten. Es soll im Programm nur eine Element der Klasse *SystemFehler* existieren, das heisst es ist ein Singleton.
Beginne mit den nötigen #includes und vergiss nicht die namespaces. Definiere danach die Basisklasse als Sammlung (Container) von *std::string*. Definiere nun die Klasse *SystemFehler* als abgeleitete Klasse unter Verwendung des Singleton-Patterns. Schreib zuletzt den Code, der zur Klasse *SystemFehler* gehört, ausser du hast alles inline definiert. (ca. 14 Zeilen Code insgesamt) (6 Punkte)

```
#include <vector>
#include <string>

using namespace std;

typedef vector<string> FehlerStrings;

class SystemFehler : public FehlerStrings
{
public:
    static SystemFehler& get();

private:
    SystemFehler();
};

SystemFehler::SystemFehler()
{
}

SystemFehler& SystemFehler::get()
{
    static SystemFehler fehler;
    return fehler;
}
```

5. Weitere Fragen

a) Schreibe eine Funktion *GenerateMail*. Diese Funktion hat als Argumente zwei strings *Name*, *Domain* und als Rückgabewert einen string. Der Rückgabewert soll einer mail-Adresse dieser Form entsprechen : *Name@Domain*. (2 Punkte)

```
#include <string>
using namespace std;

string GenerateMail(const string& name, const string& domain)
{
    return name + „@“ + domain;
}
```

b) Du willst ein Programm schreiben, dass möglichst schnell ist. Wieviele zusätzliche Threads erzeugst du? (1 Punkt)

```
keinen
```

c) Betrachte die Funktion *ausgabe*, die 1'000'000'000 mal aufgerufen wird. Was änderst du, damit das Programm schneller wird? (2 Punkte)

```
void ausgabe(std::string sehrLangerString)
{
    cout << sehrLangerString << endl;
}
```

```
Referenz von string als Argument:

void ausgabe(cosnt std::string& sehrLangerString)
{
    cout << sehrLangerString << endl;
}
```

d) Wandle diese while-Schleife in eine for-next Schleife, so dass genau der gleiche Code ausgeführt wird. (2 Punkte)

```
string test(„Hallo“);
string::iterator it = test.begin();

while(it != test.end())
{
    cout << *it << endl;
    ++it;
}

for(string::iterator it = test.begin();
    it != test.end();
    ++it)
{
    cout << *it << endl;
}
```

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

JAAAAAA