

## 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

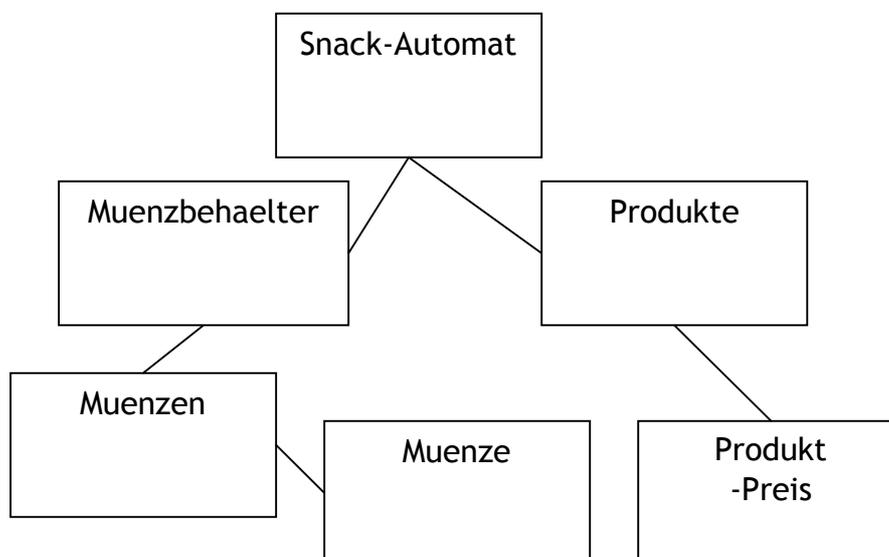
Aufgabe	Punkte	
Analyse und Design		
Windows API		
Programmverständnis		
Weitere Fragen		
<i>Total</i>		

## 1. Analyse und Design

- a) Versuche aus folgender Beschreibung ein einfaches Klassendiagramm zu erstellen. Bei einigen Dingen ist es dir überlassen, ob du eine Klasse definieren willst, oder ob etwas eine einfache Eigenschaft von einer Klasse ist. z.B. kann der Hubraum bei einem Motor nur ein einfaches Datenelement der Klasse Motor sein, oder man kann eine Klasse Hubraum definieren.

Ein Snack-Automat enthält Produkte. Ein einzelnes Produkt hat jeweils einen Preis(als Attribut). Der Snack-Automat hat einen Münzbehälter, darin befinden sich Münzen. Finde die Klassen und ergänze bei diesen die Datenelemente falls du welche findest. (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<Muenze> Muenzen;
typedef std::vector<Produkt> Produkte;
```

## 2. Windows API

- a) Mit welcher Windows-API Funktion kann man vom seriellen Port lesen?  
Es genügt der Name der Funktion.  
(1 Punkt)

ReadFile

- b) Schreibe Code, der mit einer Windows-API Funktion eine bestehende Datei mit dem Namen „Test.txt“, normal öffnet. Schliesse diese Datei danach wieder. (4 Punkte)

```
HANDLE h = CreateFile(„Test.txt“, // Name der Datei
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    0,
                    OPEN_EXISTING, // bestehende Datei öffnen
                    FILE_ATTRIBUTE_NORMAL,
                    0);
CloseHandle(h);
```

- c) Schreibe eine Klasse *SafeHandle* mit nur einem Konstruktor und Destruktor. Diese Klasse hat den Zweck ein HANDLE so zu kapseln, dass es sicher wieder geschlossen wird. Der Konstruktor hat als Argument ein HANDLE welches im Destruktor wieder geschlossen werden muss. (4 Punkte)  
Definiere einen Typumwandlungsoperator HANDLE, so dass ein Objekt der Klasse *SafeHandle* wie ein HANDLE verwendet werden kann. (2 Punkte) (Total 6 Punkte)

```
// alles inline, muss aber nicht sein

class SafeHandle
{
public:
    // Konstruktor
    SafeHandle(HANDLE h)
    {
        m_handle = h; // wäre auch gut in der
                      // initialisierung mit
                      // Doppelpunkt
    }
    // Destruktor
    ~SafeHandle()
    {
        CloseHandle(m_handle);
    }
private:
    HANDLE m_handle;
};

// Teil B kann man direkt oben reinschreiben, ich schreibe
// hier die ganze Klasse noch einmal hin. (Copy Paste)

class SafeHandle
{
public:
    // hier Konstruktor mit Initialisierungsliste
    SafeHandle(HANDLE h) : m_handle(h) {}

    ~SafeHandle()
    {
        CloseHandle(m_handle);
    }

    // Typumwandlung
    operator HANDLE() const
    {
        return m_handle;
    }
private:
    HANDLE m_handle;
};
```

### 3. Programmverständnis

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

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

using namespace std;

int main()
{
    stringstream stream;
    stream << setbase(16);
    stream << "0x" << 25 << endl;
    stream << setbase(10);
    stream << 25 << endl;
    stream << "Hallo" << endl;
    stream << setbase(16);
    stream << "0x" << 25 << endl;
    stream << setbase(10);
    stream << 52 << endl;

    string blah = stream.str();

    string::iterator it = blah.begin();
    string::iterator end = blah.end();

    for( ; it != end; ++it)
    {
        cout << *it;
    }

    return 0;
}
```

```
0x19
25
Hallo
0x19
52
```

- b) Betrachte folgenden Code, der verwendet wird um alle Elemente einer Kollektion auszugeben. Schreibe darunter eine Template-Funktion *ausgabe*, die genau das macht, was zweimal ausprogrammiert wurde. Sie soll von einem beliebigen Datentypen den iterator auf `begin()` und auf `end()` holen und in einer Schleife alle Elemente in `cout` ausgeben. So, dass danach das zweite `main` funktioniert. (4 Punkte)

```
#include <list>
#include <vector>
#include <iostream>
using namespace std;

int main()
{
    list<char> chars;
    list<char>::iterator it = chars.begin();
    list<char>::iterator end = chars.end();

    for( ; it != end; ++it)
    {
        cout << *it << endl;
    }
    // oder
    vector<long> longs;
    vector<long>::iterator itl = longs.begin();
    vector<long>::iterator endl = longs.end();

    for( ; itl != endl; ++itl)
    {
        cout << *itl << endl;
    }

    return 0;
}
```

HIER EIGENE TEMPLATE-FUNKTION : *ausgabe*  
// oben ist der Datentyp der verschieden sein kann  
// unterstrichen

```
template<class T>
void ausgabe(T& kollektion)
{
    T::iterator it = kollektion.begin();
    T::iterator end = kollektion.end();
    for( ; it != end; ++it)
    {
        cout << *it << endl;
    }
}
```

Anleitung für diese Art von Aufgabe:

- Mehrmals fast gleicher Code, aber ein Datentyp ist unterschiedlich. Oben ist der unterschiedliche Datentyp unterstrichen.
- Die Variablen von den unterschiedlichen Datentypen sind meist die Argumente für die Funktion, hier also die Variablen *longs* und *chars*
- Schreibe die template Funktion. Der ganze mehrfach (hier doppelt) vorhandene Code kommt in die Funktion.

```
int main()
{
    list<char> chars;
    ausgabe(chars);
    // oder
    vector<long> longs;
    ausgabe(longs);

    return 0;
}
```

## 4. Weitere Fragen

- a) Hier die Klasse *OberMotz*. Schreibe sie unter Verwendung des Singleton-Patterns so um, dass es nur ein *OberMotz* gibt und man nur auf dieses Objekt zugreifen kann. (3 Punkte)

```
class Obermotz
{
public:
    Obermotz(){};
    ~Obermotz(){};

    void motz(){};
};
```

Anleitung:

- Der Konstruktor wird `private`
- an seine Stelle tritt eine statische Funktion (`getObject` oder so als Name) mit Rückgabewert : Referenz auf ein Objekt der Klasse (hier `Obermotz`)
- In dieser Funktion wird ein statisches Objekt der Klasse (hier `Obermotz`) erzeugt mit `return` zurückgegeben
- Bemerkung. Diese Funktion sollte in einer `cpp` Datei stehen

```
class Obermotz
{
public:
    static Obermotz& getObject();

    ~Obermotz(){};

    void motz(){};
private:
    Obermotz(){};
};
```

```
Obermotz& Obermotz::getObject()
{
    // statisches Objekt erzeugen
    static Obermotz derMotz;
    return derMotz;
}
```

- b) Du hast schreibst Programm, das ständig auf Benutzereingaben wartet. Was für ein Windows-Mechanismus verwendest du um „gleichzeitig“ andere Aufgaben in deinem Programm zu bewältigen? (1 Punkt)

THREADS

- c) Willst Du fehlerfreie Programme schreiben? (1 Punkt)

JA