

## Hilfsblatt für C++ Prüfungen im 5. Semester

Hilfsblatt für C++ Prüfungen im 5. Semester	1
Klassen	2
Win-32 Programmierung	3
Handles	3
Dateien, Files	3
Threads	3
Events	4
Funktionen	5
Einfache Funktionen	5
Template Funktionen	5

## Klassen

Eine Klasse hat häufig folgende Form:

```
#include <string>

class Klassenname
{
    public:
        // Konstruktor
        Klassenname(const std::string& konstruktorArgument)
        {
            _datenelement = konstruktorArgument;
        }
        // Destruktor zum Aufräumen
        // z.B. Speicher freigeben, HANDLES schliessen etc.
        ~Klassenname()
        {
        }
    private:
        std::string _datenElement;
}; // Semikolon nicht vergessen!
```

Zum Beispiel hier eine Klasse Person:

```
#include <string>

class Person
{
    public:
        Person(const std::string& vorname,
               const std::string& nachname)
        {
            _vorname = vorname;
            _nachname = nachname;
        }
        ~Person()
        {
            // Hier wäre gar kein Destruktor nötig
        }
    private:
        std::string _vorname;
        std::string _nachname;
};
```

## Win-32 Programmierung

### *Handles*

Der Datentyp **HANDLE** wird verwendet um verschiedene WIN-32 Objekte wie Dateien, Fenster und Threads zu identifizieren.

Wenn ein solches Objekt nicht mehr gebraucht wird muss das HANDLE mit **CloseHandle** geschlossen werden.

### *Dateien, Files*

Dateien sind auch WIN-32 Objekte, die über ein HANDLE identifiziert werden. Wir verwenden Dateien auch um den serielle Port zu öffnen. Es ist möglich von Dateien zu lesen und schreiben ohne auf das Ergebnis zu warten -> asynchrone Dateioperationen.

Verwendete Funktionen und Strukturen:

- **CreateFile**(„dateiname“, ...) zum Öffnen oder Erzeugen einer Datei. Als Dateiname kann auch der Name eines COM-Portes angegeben werden (z.B. „COM1“)
- **ReadFile** zum Lesen aus einer Datei
- **WriteFile** zum Schreiben in eine Datei
- **CloseHandle**, zum schliessen der Datei
- **HANDLE** zur Identifizierung einer Datei
- **OVERLAPPED** Struktur für asynchrone Dateioperationen, wird bei **ReadFile** und **WriteFile** angegeben

### *Threads*

Threads zur Ausführung paralleler Aufgaben. zB. Lesen und Schreiben von Daten (serieller Port), andere Hintergrundaufgaben, komplexe Berechnungen, etc.

Verwendete Funktionen und Strukturen:

- **CreateThread**, zur Erzeugung des Threads
- WIN-32 Events (siehe weiter unten) zum Signalisieren von Ereignissen
- **HANDLE**, **CreateThread** erzeugt ein HANDLE, über den man den Thread identifizieren kann.
- Mit **WaitForSingleObject** kann darauf gewartet werden, dass ein Thread beendet wird. Dann ist das erste Argument beim Aufruf von **WaitForSingleObject** das HANDLE des Threads.

### *Events*

Mit Events lassen sich Ereignisse melden, vor allem, wenn man von einem Thread zum anderen kommunizieren will. Normale Variablen (z.B. bool) sollten hierfür nicht verwendet werden.

Verwendete Funktionen und Strukturen:

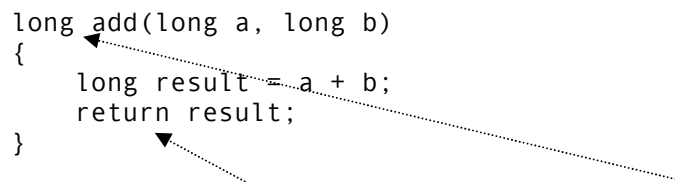
- **CreateEvent** zum Erzeugen von Events,  
Beispiel:  
`HANDLE h = CreateEvent(0, FALSE, FALSE, 0);`
- **CloseHandle** zum Schliessen des Events
- **WaitForSingleObject** um darauf zu warten, dass ein Event gesetzt wird. Beispiel:  
`WaitForSingleObject(h, 1000);`  
Warte 1000 Millisekunden, dass das Event h gesetzt wird.

## Funktionen

### *Einfache Funktionen*

Manchmal muss man eine einfache Funktion definieren. Es geht dabei nur darum die Aufgabenstellung richtig zu interpretieren, die Funktion selber sollte recht einfach sein, z.B.:

```
long add(long a, long b)
{
    long result = a + b;
    return result;
}
```



Berechnet die Funktion irgendetwas, hat sie einen Rückgabtyp! Berechnet die Funktion aber nichts, weil sie z.B. nur etwas ausgibt, dann ist der Rückgabtyp **void**. Es braucht dann auch kein **return xxx**. Hier jedoch, wird etwas berechnet und dieses Ergebnis zurückgegeben.

### *Template Funktionen*

Solche einfache Funktionen lassen sich einfach als **template** Funktionen definieren, wo dann der Datentyp beliebig sein kann.

Finde heraus, wo der Datentyp ersetzt werden muss. Hier sind es die Argument und natürlich auch der Rückgabewert:

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

Überall wo vorher der Datentyp long stand, konnte dieser mit dem Template-Argument T ersetzt werden!