## 1. Pre-compiled headers

Header-files enthalten **kein Code**! In einem header-file findet man **nur** Definitionen von Klassen, Funktionen, Typen und Konstante. Diese werden in einer gleichnamiger cpp-Datei implementiert.

`#include <myheader.h>` - die Datei findet man im System path
`#include "myheader.h"` - die Datei ist im current directory. In der Praxis erscheint dieser Fall im Labor.

Für jede `myheader.h`-Datei muss es eine myheader.cpp-Datei geben (Achtung! Der Name ist identisch), welche die Implementierungen der Methoden enthält.

**Beispiel:**

main.cpp

```
#include "myheader.h"
#include <iostream>
#include <cassert>

int main()
{
    //my awesome code here!
    return 1;
}
```

myheader.h

```
int myfun1(float, float);
int myfun2(int);
void display(int);
```

myheader.cpp

```
#pragma once
#include "myheader.h"

int myfun1(float, float)
{
    //code here
}

int myfun2(int)
{
    //code here
}

void display(int)
{
    //code here
}
```

In diesem Beispiel müssen alle drei Dateien in demselben Ordner sein.

`#pragma once` dient dazu, die Datei nur ein einziges Mal mein der Kompilation zu importieren. Falls diese linie fehlt, wird bei jedem #include-Statement die Datei eingefügt, was zu Fehlern führt (derselbe Name wird mehrmals definiert). Das nennt man „include guard“.

## 2. Dynamische Allokation

In Unterschied zu Python, ist es in C++ dem Benutzer überlassen, den Speicher zu verwalten. Man allokiert was und wann man braucht, und man dealokkiert, wenn man es nicht mehr braucht. In vielen Fällen weiß man bei der Kompilierung nicht, wie viel Speicherplatz für Listen, Arrays, etc notwendig ist, und diese Information wird nur bei Runtime klar.

```
int * foo;
foo = new int [5];
...
delete[] foo;
```

In diesem Beispiel wird ein **pointer zu einem int** deklariert, d.h. eine Variable die eine Speicheradresse enthält, und den dort gefundenen Wert als int interpretiert. Man kann pointers zu jedwelchen Datentypen haben. Bei diesem Schritt haben wir einen sogenannten *wild pointer*, einen deklarierten nicht initialisierten Pointer. Diese sind **extrem gefährlich**, die Speicheradresse die sie anzeigen kann was immer enthalten. Mit dem Operator new wird der Pointer dann allokiert: man erstellt ein Array for fünf ganzen Zahlen, und foo ist dann die Adresse des Ersten. Am Ende muss die Variable deallokiert sein, damit man den Speicherplatz befreit.

Zusätzliche Informationen hier.

## 3. Friend-Klassen und Funktionen

```
class Node {

private:
    int key;
    Node* next;
    /* Other members of Node Class */

    // Now class  LinkedList can
    // access private members of Node
    friend class LinkedList;
};
```

Einer friend-Klasse ist der Zugriff zu privaten Daten und Methoden der Klasse, wo sie ans friend deklariert wurde. In diesem Beispiel kann die Liste private Mitglieder der Klasse Node benutzen. Sowohl Klassen, als auch Funktionen bzw. Methoden können Friend sein. Die Beziehung ist **nicht** gegenseitig. Falls Klasse A friend mit B ist, dann wird B nicht automatisch friend mit A. Manchmal geht es auch im täglichen Leben so...

## 4. Pair

Das pair template erlaubt es, zwei verschiedene Objekte in einer einheitlichen Variablen zu speichern und manipulieren. Es ähnelt den Tuples und Dictionaries aus Python. Die beiden Elemente können vom jedwelchen Typ sein, und werden mit first und second abgerufen. Die Operatoren =, == und != funktionieren wie erwartet. Die Operatoren <, <=, >, >= vergleichen jeweils die **ersten** Elemente der Paare.

**Beispiel:**

```
// pair::pair example
#include <utility>      // std::pair, std::make_pair
#include <string>       // std::string
#include <iostream>     // std::cout

using namespace std;
//without this, one would have to always write std::pair and std::cout

int main () {
  pair <string,double> product1;                     // default constructor
  pair <string,double> product2 ("tomatoes",2.30);   // value init
  pair <string,double> product3 (product2);          // copy constructor
  if (product3 != product2){
    cout << "Something is wrong!";
  }

  product1 = make_pair(string("lightbulbs"),0.99);
  // using make_pair (move)

  product2.first = "shoes";              // the type of first is string
  product2.second = 39.90;               // the type of second is double

  cout << "The price of " << product1.first << " is $" << product1.second << '\n';
  cout << "The price of " << product2.first << " is $" << product2.second << '\n';
  cout << "The price of " << product3.first << " is $" << product3.second << '\n';
  return 0;
}
```

Output:

```
The price of lightbulbs is $0.99
The price of shoes is $39.9
The price of tomatoes is $2.3
```

## 5. Nutzliche Links

https://stackoverflow.com/

http://www.cplusplus.com/

https://www.google.com/