Projekte auf Header- und Quellcodedateien verteilen

Bei großen Projekten dauert das Kompilieren nach jeder Änderung sehr lange, wenn der gesamte Code übersetzt werden muss. Außerdem ist es umständlich, Funktionen in andere Programme zu übernehmen. Deshalb verteilt man Programme auf mehrere Dateien.

Quellcode-Dateien

In Quellcode-Dateien mit der Endung `.cpp` oder `.c` stehen Funktionen, darunter auch das Hauptprogramm. Es bleibt dem Entwickler überlassen, zusammengehörige Funktionen miteinander in eine Datei zu schreiben.

Der Compiler bearbeitet bei jeder Übersetzung nur die Quellcode-Dateien, die sich seit dem letzten Lauf geändert haben. Später setzt der Linker automatisch alle vorhandenen Objektdateien zusammen.

Header-Dateien

Header-Dateien mit der Endung `.h` enthalten Funktionsprototypen, Präprozessor-Konstanten und ähnliches. Zu jeder Quellcode-Datei `xyz.cpp` - ausgenommen das Hauptprogramm - sollte es eine Header-Datei `xyz.h` geben.

In den Quellcode-Dateien bindet man mit Präprozessor-Anweisungen alle Header ein, die man braucht. Außerdem ist es üblich, dass jede Quellcodedatei ihren eigenen Header einbindet, auch wenn das manchmal nicht nötig ist. Typischerweise würde also in der Datei `xyz.cpp` die Anweisung `#include "xyz.h"` stehen.

Die Anführungszeichen bei `include` bedeuten, dass die Datei im gleichen Ordner wie die Quellcodedatei gesucht wird. Spitze Klammern stehen dagegen für den Ordner, der sonst für Header vorgesehen ist.

Bei Header-Dateien aus der C++-Standardbibliothek fehlt die Endung `.h`. Beispiel: `#include <iostream>`

Wo bindet man andere Headerdateien ein?

Dateien, die andere Programmierer brauchen, um Funktionen nutzen zu können, werden in der Headerdatei eingebunden, und solche, die nur für die Implementierung der Funktionen wichtig sind, in der Quellcodedatei.

Mehrfaches Einbinden vermeiden

Am Anfang von Header-Dateien können Sie bei manchen Präprozessoren `#pragma once` schreiben. Das sorgt dafür, dass die Datei garantiert nur einmal pro Quellcode-Datei eingebunden wird.

Leider gehört diese Anweisung nicht zum C++-Standard. Wenn der Präprozessor sie nicht versteht, muss man eine andere Konstruktion verwenden:

```
#ifndef  DATEINAME_H  // ifndef = if not defined
#define  DATEINAME_H  // Dateinamen als leere Konstante definieren

// ... eigentlicher Inhalt der Header-Datei ...

#endif
```

Der Assistent für Headerdateien von Codeblocks verwendet dieses Verfahren mit einem längeren Namen für die Konstante.

Beispiel

Die Lösung einer Übungsaufgabe zu Referenzparametern soll fachgerecht auf einzelne Dateien verteilt werden.

```
#include <iostream>
using namespace std;
#include <cmath>
#define PI 3.1415927

void dreieck(double &a, double &b, const double c, const double alphaGrad);
// Die Funktion erwartet den Winkel alpha in Grad.

int main()
{
          double a, b, c, winkelalpha;

          cout << "c: ";
          cin >> c;
          cout << "alpha: ";
          cin >> winkelalpha;

          dreieck(a, b, c, winkelalpha);

          cout << endl;
          cout << "a: " << a << endl;
          cout << "b: " << b << endl;
}

void dreieck(double &a, double &b, const double c, const double alphaGrad)
{
          double alphaBogen = alphaGrad / 360 * 2 * PI;
          a = c * sin(alphaBogen);
          b = c * cos(alphaBogen);
}
```

Headerdatei dreieck.h

In die Headerdatei gehört alles, was das Hauptprogramm und andere Funktionen brauchen, um die Funktion `dreieck()` aufzurufen. In diesem Beispiel ist das nur der Prototyp. Wenn die Datei `cmath` und die Konstante `PI` zum Benutzen der Funktion nötig wären, kämen auch sie in die Headerdatei. Hier ist das aber nicht notwendig.

Wie in jeder Headerdatei steht in der ersten Zeile die Präprozessor-Anweisung gegen doppeltes Einbinden.

```
#ifndef DREIECK_H_INCLUDED
#define DREIECK_H_INCLUDED

void dreieck(double &a, double &b, const double c, const double alphaGrad);
// Die Funktion erwartet den Winkel alpha in Grad.

#endif // DREIECK_H_INCLUDED
```

Quellcodedatei dreieck.cpp

Die Funktionsdefinition und die Präprozessor-Anweisungen, die nur zu ihr gehören, wandern in eine eigene Quellcodedatei.

Außerdem ist es üblich, in jede Quellcodedatei die dazugehörige Headerdatei einzutragen. Hier wäre das nicht nötig, aber es dient der Fehlersicherheit für den Fall, dass später Einträge in `dreieck.h` dazukommen, die sowohl im Hauptprogramm als auch in der Funktion vorhanden sein müssen.

```
#include "dreieck.h"
#include <cmath>
#define PI 3.1415927

void dreieck(double &a, double &b, const double c, const double alphaGrad)
{
          double alphaBogen = alphaGrad / 360 * 2 * PI;
          a = c * sin(alphaBogen);
          b = c * cos(alphaBogen);
}
```

Hauptprogramm

Die Quellcodedatei mit dem Hauptprogramm ist jetzt deutlich kürzer. Neu kommt nur eine Zeile hinzu, die unsere Headerdatei einbindet.

```
#include <iostream>
#include "dreieck.h"

using namespace std;

int main()
{
          double a, b, c, winkelalpha;

          cout << "c: ";
          cin >> c;
          cout << "alpha: ";
          cin >> winkelalpha;

          dreieck(a, b, c, winkelalpha);

          cout << endl;
          cout << "a: " << a << endl;
          cout << "b: " << b << endl;
}
```

Die Beschreibung wurde möglichst allgemein gehalten. Trotzdem müssen Sie bei jedem Projekt selbst darüber nachdenken, welche Zeile wohin gehört. Es gibt kein Strickmuster, das immer passt.

Aufgabe 1: Vorhandenes Programm aufteilen

Teilen Sie ein funktionierendes Programm, das mindestens eine Funktion enthält, ordentlich auf!

Aufgabe 2: Palindromzahlen

Schreiben Sie eine Funktion `bool istPalindrom(unsigned int zahl)`, die prüft, ob eine Palindromzahl übergeben wurde, und als Ergebnis `true` oder `false` zurückgibt! Eine Palindromzahl ist eine Zahl, die von vorne und von hinten gelesen das Gleiche ergibt.

Erstellen Sie von Anfang an eine Headerdatei `palindrom.h` und eine Quellcodedatei `palindrom.cpp`, damit Sie das Programm nicht nachträglich aufteilen müssen. Schreiben Sie auch ein kleines Hauptprogramm zum Ausprobieren.

- **Einfache Variante:** Die Funktion muss nur mit Zahlen bis 100 funktionieren. Sie können mit einer Mehrfachauswahl arbeiten, denn in diesem Intervall gibt es nur 19 Palindromzahlen.
- **Verbesserte Variante zum Üben:** Die Funktion muss mit allen Werten funktionieren, die der Datentyp des Parameters darstellen kann.

Expertenaufgabe: Snake

Sie haben neulich das Steuern des Cursors mit der Tastatur geübt. Programmieren Sie jetzt ein kleines, mögliches farbiges Spiel für die Kommandozeile, zum Beispiel *Snake*!