Programmieren in C++

SS 2011

Vorlesung 12, Mittwoch 27. Juli 2011

(Hilfestellung Projekt, Evaluation)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Erfahrungen mit dem 11. Übungsblatt (Projekt .h Dateien)
  - Dies ist die **vorletzte** Vorlesung
  - Evaluationsbogen zur Veranstaltung
- Hilfestellung für das Projekt
  - friend, forward Deklaration
  - Advanced File I/O (Ordner auslesen, Dateitypen, ...)
  - Tastaturevents abfangen
  - Allgemeine Tipps zum Klassendesign
  - Spezielle Tipps zum Klassendesign für Search und Pong
- Übungsblatt 12
  - Evaluierungsbogen abgeben + Projekt weiter machen

Evaluation

UNI FREIBURG

- Bogen bis spätestens nächste Woche abgeben
  - … und wenn möglich schneller
  - Abgabe über das Forum, siehe Übungsblatt
  - Bitte nehmen Sie sich Zeit dafür!
  - Insbesondere für die Freitextkommentare!

Erfahrungen mit dem Ü11 (Projekt .h)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 27.7 15:50
  - Interessant .h Dateien zu schreiben ohne Implementierung
  - Aber auch schwierig / Am Anfang skeptisch
  - Man erkennt schnell, welche Probleme auftreten können etc.
  - Erstellen der .h Dateien ist schon ein Großteil des Projektes
  - Gibt es einen buildbot der Testfälle schreibt?
  - C/C++ Methoden für das Dateisystem, ein HowTo wäre nett
  - Was tun, wenn sich Klassen gegenseitig brauchen?
  - Suchmaschine: welche Arten von Dateien, was wenn sich eine Datei ändert, auf welchem Betriebssystem?
  - Gespannt auf die Korrektur dieses Übungsblattes
  - Nächstes Übungsblatt hoffentlich nicht so viel Arbeit

Erfahrungen mit dem Ü11 (Projekt .h)

UNI FREIBURG

- Zusammenfassung / Auszüge Stand 27.7 15:50
  - Schade, dass Suchen und Indizieren zwei verschiedene Programme sind, sie haben so viel gemeinsam
  - Beschreibung auf dem Wiki sehr hilfreich
  - Darf man eine std::map oder so benutzen
  - Ein Übungsblatt der etwas anderen Art
  - Projekte sind super
  - Suchmaschine ist klasse
  - Suchmaschine ist langweilig

friend 1/3

UNI FREIBURG

- Manchmal will man für eine Klasse Zugriff auf die
  - … (privaten) Membervariablen einer anderen Klasse
  - Und zwar ohne dass die eine Klasse von der anderen erbt
  - Dann kann eine Klasse die andere zum **friend** erklären

```
class A { ... };

class B {
  friend class A;  // Allow A to access the members of B.
};
```

  - Das ist nicht symmetrisch, Objekte vom Typ A haben jetzt Zugriff auf die (auch privaten) Membervariablen von B, aber nicht umgekehrt
  - Dafür müsste man in die Deklaration von class A schreiben friend class B … was ja auch sinnvoll ist, warum?

friend 2/3

- Ein praktisches Beispiel
  - Spielfeld in einer Klasse, Spielball und Schläger in separaten Klassen, die sollten dann aber Zugriff zum Beispiel auf die Dimensionen des Spielfeldes haben

```
class PongBall {
 private:
  PongScreen* _screen;  // Object for the screen as a whole.
}

class PongScreen {
 private:
  int _maxX, _maxY;  // Dimensions of the screen.
  friend PongBall;  // Allow PongBall to access members.
};
```

friend 3/3

- Friend Methoden
  - Man kann auch statt einer ganzen Klasse nur einer ausgewählten Methode oder globalen Funktion Zugriff gewähren

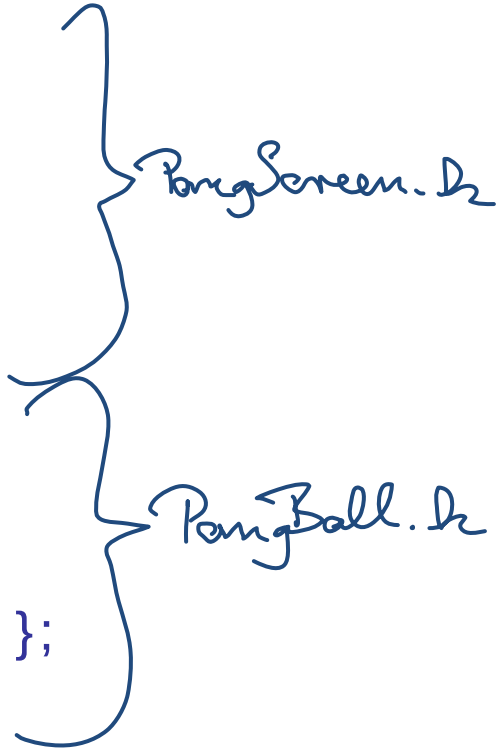
```
class A
{
  ...
  friend void myFunction(int x, int y);
};

void myFunction(int x, int y)
{
  // Can now access private members from A
}
```

Forward Deklarationen 1/4

- Oft brauchen sich Klassen gegenseitig

```
#ifndef PONGSCREEN_H_
#define PONGSCREEN_H_
#include "./PongBall.h"
class PongScreen { ... PongBall _ball; ... };
#endif  // PONGSCREEN_H_
```

```
#ifndef PONGBALL_H_
#define PONGBALL_H_
#include "./PongScreen.h
class PongBall { ... PongScreen* _screen; ... };
#endif  // PONGBALL_H_
```



  - Problem: Wird zuerst PongScreen.h kompiliert, kennt er beim Inkludieren von PongBall.h die Klasse PongScreen nocht nicht ... und andersrum entsprechend, was tun?

Forward Deklarationen 2/4

UNI FREIBURG

- Lösung: forward Deklarationen
  - Um ein Objekt einer Klasse zu deklarieren, braucht man vorher die volle Klassendeklaration

```
#ifndef PONGSCREEN_H_
#define PONGSCREEN_H_
#include "./PongBall.h";  // Full declaration required.
class PongScreen { ... PongBall _ball; ... };
#endif  // PONGSCREEN_H_
```

  - Für einen Zeiger auf eine Klasse reicht eine **forward** Deklar.

```
#ifndef PONGBALL_H_
#define PONGBALL_H_
class PongScreen;  // Forward declaration suffices.
class PongBall { ... PongScreen* _screen; ... };
#endif  // PONGBALL_H_
```

Forward Deklarationen 3/4

UNI FREIBURG

- Hintergrund
  - Eine Vorwärtsdeklaration ist wie ein Versprechen an den Compiler, dass die Datei dann später noch deklariert wird

    (wird sie das nicht, gibt es eine Fehlermeldung)
  - Warum geht das nur bei Zeigern nicht bei Objekten?
  - Weil der Compiler bei der Deklaration wissen muss, wieviel Platz er für diese Variable reservieren soll
    - Bei einem Objekt hängt das von der Klasse ab
    - Bei einem Zeiger auf ein Objekt ist das unabhängig von der Klasse immer gleich (4 bytes bei 32-Bit Maschinen, 8 bytes bei 64-Bit Maschinen)
  - Außerdem muss der Compiler wissen, welche Konstruktoren es für ein Objekt gibt

Forward Deklarationen 4/4

UNI FREIBURG

- Include oder forward Deklaration?
  - Wenn man die Wahl hat, sollte man immer die forward Deklaration vorziehen
  - Die Dateien werden dann in einer nachvollziebareren Reihenfolge eingelesen, was bei evtl. Fehlersuche hilft
  - Das klappt allerdings nicht bei template Klassen

```
class Array<int>;  // Will not compile.

template class Array<int>;  // Explicit instantiation,
                            // not a forward declaration.
```

  - Insbesondere muss man also alle STL Klassen, die man benötigt, wirklich includen

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

Advanced File I/O 1/3

UNI FREIBURG

- Auslesen der Dateien in einem Ordner
  - Das geht mit **opendir**, **readdir** und **closedir**

```
#include <sys/types.h>
#include <dirent>

DIR* dir = opendir("my_folder");  // Open directory.
if (dir == NULL) { ...}  // Check for error!
struct dirent entry;  // An entry in the directory.
while (true)
{
  entry = readdir(dir);  // Read next entry in directory.
  if (entry == NULL) return;
  printf("%s\n", entry.d_name);  // File name of entry.
}
closedir(dir);
```

Advanced File I/O 2/3

UNI FREIBURG

- Es gibt verschiedene Typen von Dateien
  - Normale Dateien, Ordner, symbolische Links, etc.
  - Den Typ einer Datei (und mehr!) bekommt man mit **stat**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

struct stat info;  // Variable holding stat information.
int ret = stat("my_file.txt", &info);
if (ret == -1) { ... }  // Some problem occurred.
if (S_ISREG(info.st_mode)) printf("Regular file!\n");
if (S_ISDIR(info.st_mode)) printf("Directory!\n");
if (S_ISLNK(info.st_mode)) printf("Symbolic link!\n");
printf("File size in bytes = %zu\n", info.st_size);
```

Advanced File I/O 3/3

UNI FREIBURG

- Rekursives Auslesen eines Ordners
  - Geht mit einer Kombination von opendir, readdir, closedir und stat nach dem Muster der vorherigen Beispielprogramme
  - Siehe ReadDirMain.cpp im SVN unter vorlesungen/vorlesung-12

Tastaturevents einlesen 1/2

UNI FREIBURG

- Für viele UIs will man non-blocking und no-echo
  - non-blocking: das Programm wartet nicht
  - no-echo: Zeichen wird nur gelesen, nicht gedruckt
  - Das geht mit ncurses ganz einfach

```
#include <ncurses.h>

initscr();  // Initializes the screen, assigns stdscr.
cbreak();  // No input buffering.
noecho();  // Don't echo input characters on screen.
nodelay(stdscr, true);  // Don't wait for input when getch().
while (true) {
  c = getch(); // Checks keyboard input non-blocking no-echo.
  ...
}
```

Tastaturevents einlesen 2/2

UNI FREIBURG

- Mehrere Tasten gleichzeitig?
  - Die meisten Tastaturen liefern überhaupt nicht die notwendige Information dafür
  - Was aber praktisch alle Tastaturen können, ist herausfinden ob zusätzlich zu einer anderen taste shift, ctrl, alt, etc. gedrückt wurden … oder Kombinationen davon
  - Das kann man aber mit ncurses nicht leicht abfragen
  - Ist aber für das Spiel auch nicht unbedingt notwendig
  - Wer da was rausfindet, bitte Post auf dem Forum, damit die anderen auch was davon haben

Tipps zum Klassendesign 1/4

UNI FREIBURG

- Was ist ein gutes Klassendesign, Phase 1
  - Alles was vom Gefühl her ein sinnvolles eigenes Objekt ist erstmal in eine separate Klasse

    Spielfeld, Spielball, Schläger, ...

    Index, Invertierte Liste, Suchanfrage, ...
  - Dann schauen, aus was dieses Objekt "besteht" (Membervariablen) und was es alles können sollte (Methoden)

Tipps zum Klassendesign 2/4

UNI FREIBURG

- Was ist ein gutes Klassendesign, Phase 2
  - Wenn eine Klasse zu groß / komplex wird, überlegen einen Teil in eine eigene Klasse daraus zu machen
    - zum Beispiel die Funktionalität zur Verarbeitung einer Suchanfrage nicht in der Klasse InvertedIndex, sondern in einer eigenen Klasse QueryProcessor
    - ebenso eine separate Klasse ExcerptsGenerator zur Generierung der Ausgabe für eine Trefferdatei
  - Umgekehrt sollten Klassen auch nicht trivial sein / werden (z.B. nur eine Membervariable und / oder eine Methode)
    - zum Beispiel ist es nach der o.g. Auslagerung von QueryProcessor und ExcerptsGenerator fraglich, ob man die Klasse InvertedIndex noch braucht

Tipps zum Klassendesign 3/4

UNI FREIBURG

- Was ist ein gutes Klassendesign, Phase 3
  - Wenn Klassen etwas gemeinsam haben, an **Vererbung** denken, und das Gemeinsame in eine Oberklasse auslagern
    - zum Beispiel Oberklasse PongObject für PongBall und PongPaddle, mit einer move() Funktion etc.
  - Wenn in einer Klasse ein Typ austauschbar ist und man sie für mehrere dieser Typen braucht, an **templates** denken
  - **ABER:** zusätzliche Abstraktion ist schwerer zu verstehen, also nur wenn nötig, nicht um der Abstraktion willen!
    - zum Beispiel hängt es von der Realisierung des Spieles ab, ob die Oberklasse PongObject wirklich was bringt
  - Sowas findet man am besten raus, indem man einfach mal mit einem Design anfängt, und dann schaut wie es passt

Tipps zum Klassendesign 4/4

UNI FREIBURG

- Spezielle Tipps zu Search und Pong
  - Im SVN unter vorlesungen/vorlesung-12 finden Sie einen Vorschlag für ein Klassendesign für Search und Pong
  - Der Einfachheit halber steht dort alles in einer Datei

    SearchEngingeClassDesignMain.cpp

    PongClassDesignMain.cpp

    Das sollen Sie natürlich nicht so machen
  - Das ist nur ein Vorschlag (von mir) und nicht verbindlich
  - Insbesondere habe ich selber noch keinen der Teile implementiert, gut möglich, dass ich bei der Implementierung feststellen würde, dass man die eine oder andere Sache besser anders macht

Literatur / Links

UNI FREIBURG

- Advanced File I/O
  - man 3 opendir
  - man 3 readdir
  - man 3 closedir
  - man 2 stat
- Friendship
  - http://www.cplusplus.com/doc/tutorial/inheritance/
- Forward Deklarationen
  - http://google-styleguide....Header_File_Dependencies

    (Der Google-Style Guide ist überhaupt sehr interessant)