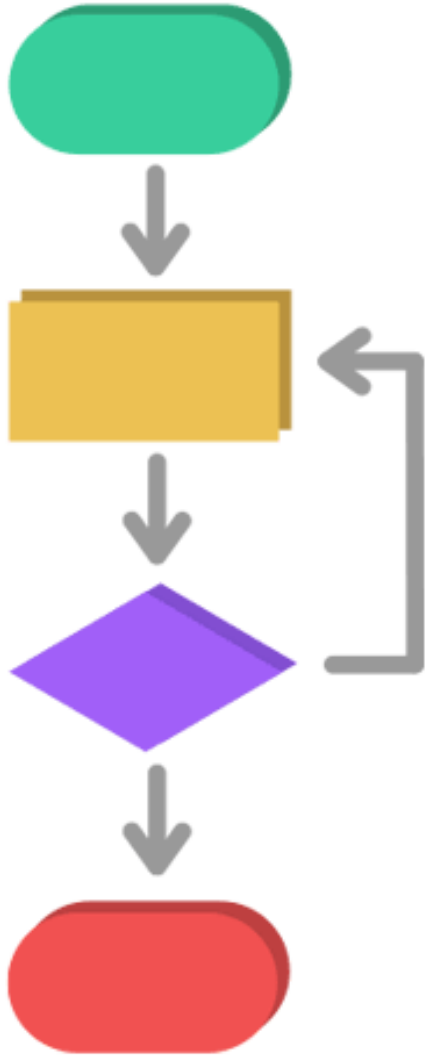
Algorithmen Tutorium

Beginn: 16:15



10.11.2021

Organisation

Hashing

- Erstmal eine Funktion f(x) = y
  - Verkleinern der Anzahl der möglichen Werte
  - Weitere Gütekriterien (z.B. Wahrscheinlichkeit von Kollisionen)
- Erlaubt schnelleren Zugriff (vgl. Array bei Blatt 03 1c))
- Wichtig bei Krypto (Surjekivität)
- Einfache Hashfunktion: f(x) = x mod y

Hashing (Hash Tabellen)

- Hash-Tabelle mit einer Zelle pro Hashwert
  - Manchmal ein Array der Größe k oder eine Liste pro Hashwert
- Wozu Hash Tabellen?
  - Relativ kleine Größe
  - Kollisionserkennung
  - Schneller Elementzugriff

Hashing (Hash Tabellen Beispiel)

- h(x) = x modulo 10 und x [111, 3, 182, 12345, 9000]

h(x)	0	1	2	3	4	5	6	7	8	9
x										

- Ist 42 in der Tabelle?
- Füge 42 hinzu

Hashing (Hash-Kollisionen)

- Was tun bei gleichen Werten (Hash-Kollisionen)?
- Generell: Gute Hashfunktion wählen
- Was nicht geht
  - Element überschreiben
  - Nicht hinzufügen
- Was geht
  - Nächsten freien Platz besetzen (in eine Richtung)
  - Alle Elemente zyklisch shiften
  - Evtl. andere Hash-Tabelle
  - …

Hashing mit Verdrängung

- Verdrängungstaktik: Lineares Sondieren mit Intervall 1
  - Nächster freier Platz „rechts“ von h(x)
  - Falls wir ans Ende der HT kommen, mach bei erstem Eintrag weiter
  - Ende bei freiem Platz oder nach einem Durchlauf
- Füge 42 hinzu

! Lineares Sondieren ist eine eher schlechte Strategie !

h(x)	0	1	2	3	4	5	6	7	8	9
x	9000	111	182	3		12345				

- Was wenn HT komplett voll ist?
  - Error!

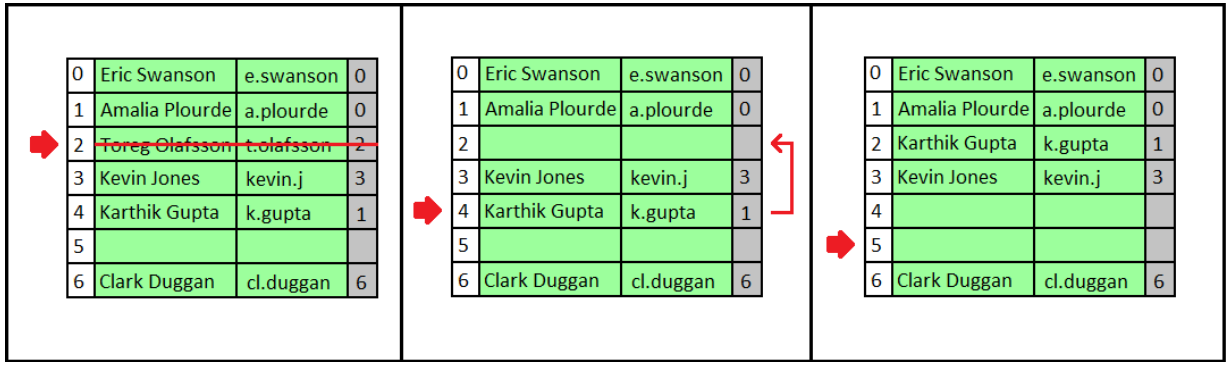
Hashing mit Verdrängung

- Suchen von x mit Verdrängung
  - Schaue bei h(x) in der HT
  - Sondiere ggf. weiter bis ein freier Platz oder x gefunden ist
- Suche 182, 42 und 32

	2	3	4	5	6	
…	182	3	42	12345		…

Hashing mit Verdrängung

- Löschen von x mit Verdrängung
  - Suche nach x
  - Wenn x in HT Lösche Element
  - Verschiebe evtl. Hash-Kollisionen



https://commons.wikimedia.org/wiki/File:Linear_Probing_Deletion.png

- Verdrängung kostet -> Vermeiden!!!

Objektorientierung C++

Wieso C++?

- C ist oft kompliziert
- C kann „zu wenig“
  - Z.B. C hatte ursprünglich keine Booleans
- Moderne Sprachen nutzen wiederverwertbare Objekte (Klassen)
- C++ als „Erweiterung“ von C (C++ ≙ C + 1)

Was sind Objekte?

- Drücken Funktionalität & Zusammengehörigkeit aus
- Beliebig oft replizierbar
- Unabhängig voneinander
- Erlauben gezielte Zugriffssteuerung (im Gegensatz zu `Strukts`)
- Erlauben Vererbung (= Erweiterung) von Klassen
- u.v.m.

C++ Objekte (.h-Files)

- Werden in `class <class_name> { … }` im Header definiert
- Einteilung der Zugriffsrechte: `private, public, protected, <nichts>`
- Methoden und Funktionsdefinition normal im Header
- Vererbung `class <class_name> : <parent_class> { … }` im Header
  - Definierte Klasse hat alle (nicht privaten) Definitionen der Elternklasse
  - Überschreibung vorhandener Funktionen möglich

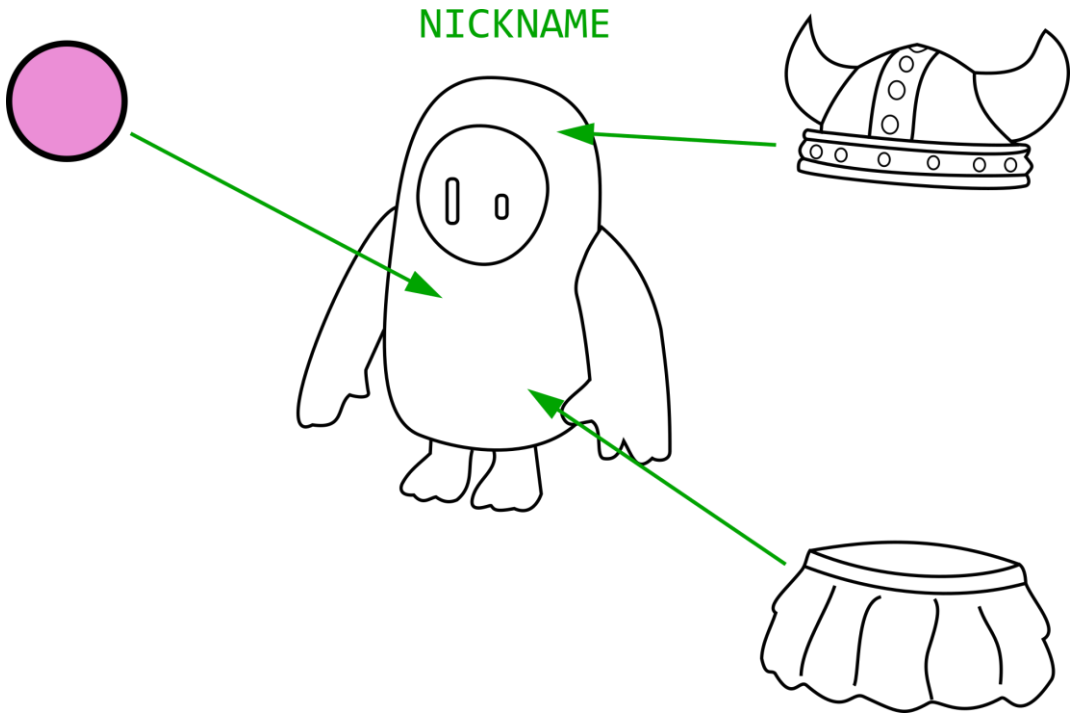
C++ Objekte (.cpp-Files)

- Klassen-Methoden müssen implementiert werden
- Neuer Funktionsname `<class_name>::<function_name>(<args>… ) { … }`
- Impliziter Zugriff auf Klassenvariablen möglich
- Zugriff auf das aktuelle Objekt: `this.`
- Namenskonflikte von Klassen und Funktionsparametern (Vorsicht!)
  - `void setX(int x) { this.x = x; }`
  - `void setX(int _x) { x = _x; }`
  - `void setX(int newX) { x = newX; }`
  - …

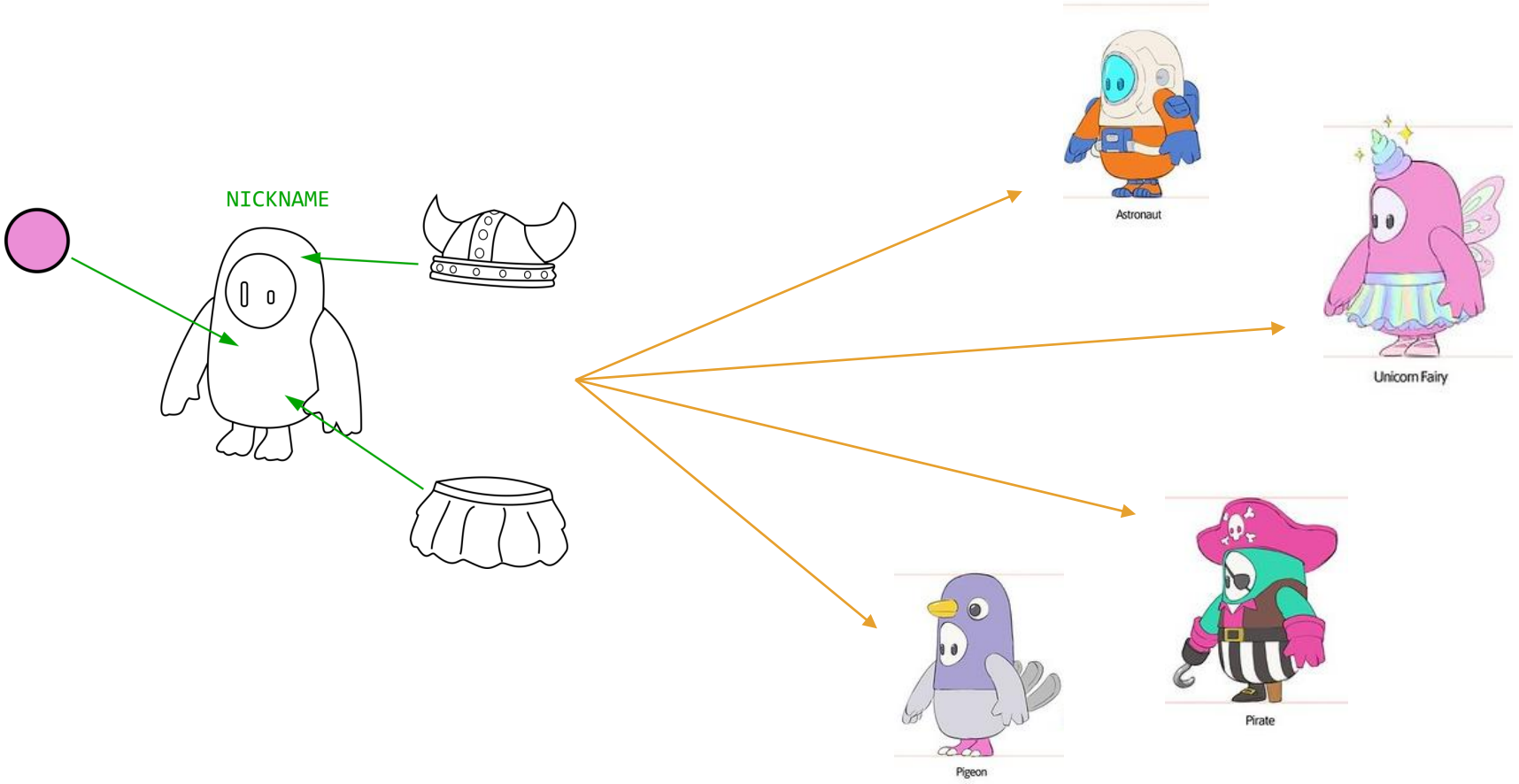
C++ (Kon-/Destruktoren)

- Konstruktoren
  - Wird beim Erstellen eines neuen Objekts aufgerufen
  - Funktionsname wie Klassenname
  - Default (implizit): <class_name>() { … }
- Destruktoren
  - Wird beim Löschen/Zerstören eines Objekts aufgerufen
  - Funktionsname wie Klassenname aber mit „~“ und keine Parameter
  - Default (implizit): ~<class_name>() { … }
- Funktionen werden implizite erstellt und aufgerufen

Beispiel „Fall-Mensch“



Beispiel „Fall-Mensch“



Misc

- `#pragma once` verhindert doppelte `#includes`
- Ausführen im Terminal: `make && ./ex03`