Inhalt:

1. Übungsblatt 1 - Aufgabe 2d
2. Übungsblatt 5, Aufgabe 4
3. Übungsblatt 6, Aufgabe 3
4. Laufzeitanalyse

Aufg. 1.02: Cacheorganisation Lsg.

d) Berechnen Sie für jeden Typ die in Wirklichkeit notwendige Speicherkapazität für den Cache Baustein

Speichergröße = Datenspeicher + Speicher für Tags + Speicher für Valid-Bits

Anzahl Cache-Blöcke berechnen: Cachegröße/Blockgröße = 512 KB / 8 B = 65536 = 2^{16}

Anzahl Blöcke ist unabhängig von N

Direkt-Abbildend (direct mapped: DM):

Anzahl Cache-Sets = Anzahl Cache-Blöcke

Jedes Cache-Set muss mit Index adressiert werden können → Index = ld(2^{16})

Tag = Adressbreite – Offset – Index = 30bit – ld(8) - ld(2^{16}) = 30 -3 -16 =11

Jedes Byte je Cacheblock adressieren

Nötige Zusatz-Bits: 12 Bit je Cache-Set (Tag+Valid)

Speichergröße = 512 KB + (12 Bit * 2^{16}) / (8 Bit/Byte) = 512 KB + 96 KB = 608 KB

Aufg. 1.02: Cacheorganisation Lsg.

4-Wege assoziativ (set associative: SA)

Anzahl Cache-Sets = Anzahl Cache-Blöcke / 4

Jedes Cache-Set muss mit Index adressiert werden können → Index = ld(2^{16} / 4)

→ Index = ld(Cachegröße / (Blockgröße * N)) = 14 bit

Tag = Adressbreite – Offset – Index = 30bit – ld(8) - 14 = 30 - 3 -14 = 13

Nötige Zusatz-Bits: 14 Bit je Cache-Set (Tag+Valid)

Speichergröße = 512 KB + (14 Bit * 2^{16}) / (8 Bit/Byte) = 512 KB + 112 KB = 624 KB

Voll-assoziativ (fully associative: FA)

Anzahl Cache-Sets = 1 → Index = 0bit

Tag = 30 - 3 = 27

Speichergröße = 512 KB + (28 Bit * 2^{16}) / (8 Bit/Byte) = 512 KB + 224 KB = 736 KB

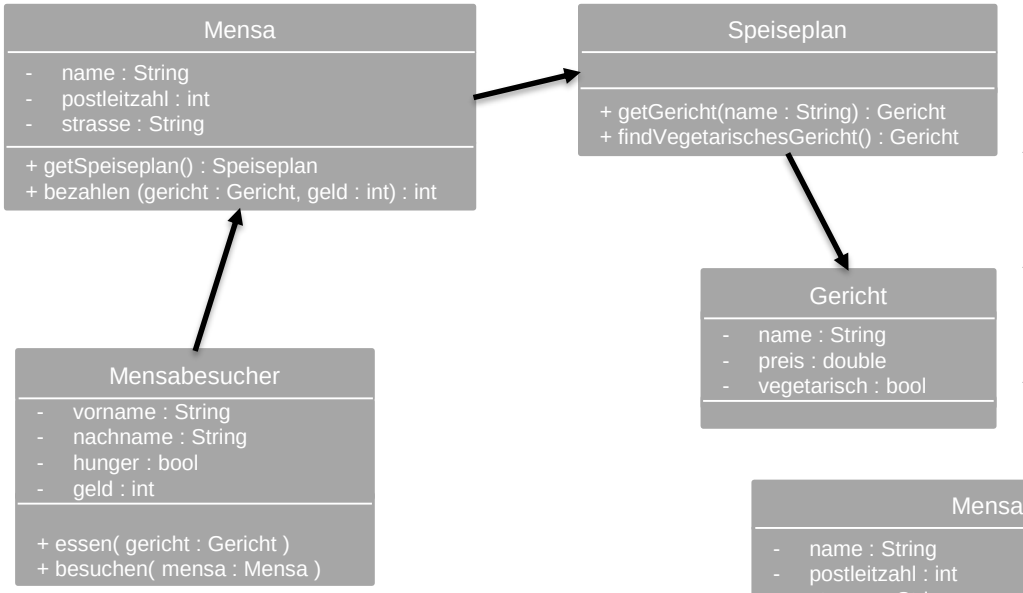
Cacheorganisation

- „Wie kann festgestellt werden, ob ein Block im Cache ist?“
  - Tags dienen zur Prüfung
  - Hauptspeicheradresse des Blocks in Tag, Index und Offset aufteilen
  - Den Tag der Hauptspeicheradresse mit den im Cache gespeicherten Tags vergleichen
  - Wenn Tag des Blocks = Tag im Cache, dann ist Block dort gespeichert

Inhalt:

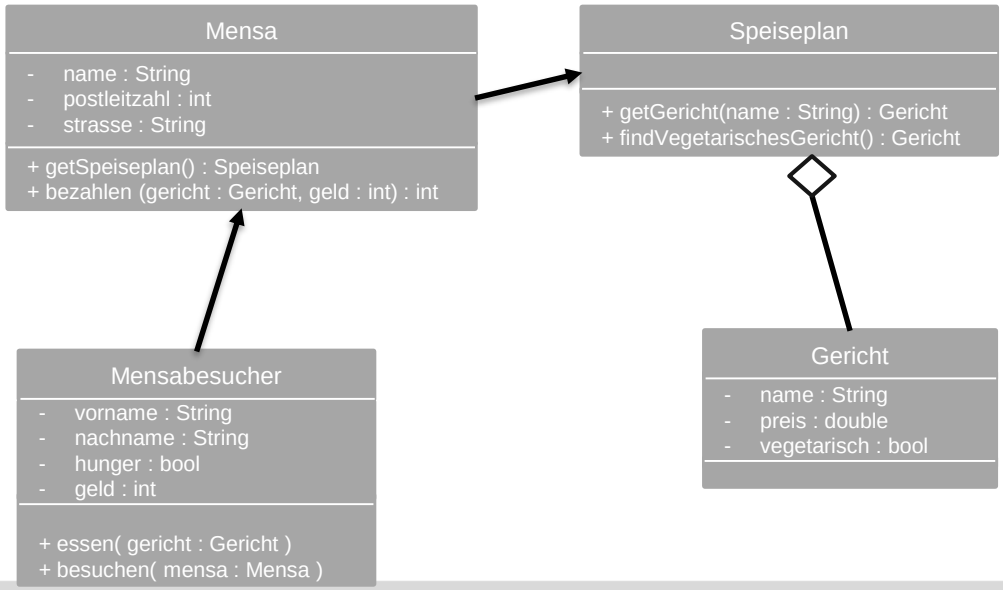
1. Übungsblatt 1 - Aufgabe 2d
2. Übungsblatt 5, Aufgabe 4
3. Übungsblatt 6, Aufgabe 3
4. Laufzeitanalyse

Aufg. 5.04: Modellierung mit Klassendiagrammen Beispiel-Lsg.



„Könnte man das Gericht auch als Teil des Speiseplans ansehen, also eine Aggregation anstelle einer Assoziation zeichnen?“

Aggregation ist hier auch möglich. Unterscheidung zwischen Aggregation und Assoziation meistens schwierig / beides möglich.



Aufg. 5.04: Modellierung mit Klassendiagrammen Beispiel-Lsg.

```
class Mensa {
  private:
    String  name, strasse;
    int postleitzahl;
public:
    Speiseplan getSpeiseplan();
    int bezahlen (Gericht gericht , int geld );
};

class Mensabesucher {
 private:
    String vorname, nachname;
    bool hunger;
    int geld;
public:
    void essen (Gericht gericht);
    void besuchen (Mensa mensa );
};

class Speiseplan {
public:
    Gericht getGericht(String name);
    Gericht findVegetarischesGericht ();
};
```

```
class Gericht {
private:
    String name;
    double preis;
    bool vegetarisch;
};
```

„Wieso wird der Datentyp string immer groß geschrieben?“

Standarddatentypen i.A. klein geschrieben, eigene Klassen / Bibliotheksklassen groß

Inhalt:

1. Übungsblatt 1 - Aufgabe 2d
2. Übungsblatt 5, Aufgabe 4
3. Übungsblatt 6, Aufgabe 3
4. Laufzeitanalyse

Aufg. 6.03: Studentendatenbank Lsg. (5)

studentenverwaltung.cpp

Zeigt auf erstes Element

```
list<Student>::iterator Studentenverwaltung::findStudent( int matrikel ) {
  list<Student>::iterator it;
  for ( it = studentenliste.begin(); it != studentenliste.end(); ++it ) {
    if ( it->getMatrikelnr() == matrikel ) {
      return it;
    }
  }
  return studentenliste.end();
}
```

Iterator wird inkrementiert → zeigt auf das nächste Element der Liste

Nach der Auswertung von it++ oder ++it ist der neue Wert von i in beiden Fällen gleich. Auswertung des Inkrementierungsschrittes ist aber unabhängig von der Ausführung des Schleifenkörpers → für korrekten Ablauf der Funktion egal, ob ++it oder it++

Inhalt:

1. Übungsblatt 1 - Aufgabe 2d
2. Übungsblatt 5, Aufgabe 4
3. Übungsblatt 6, Aufgabe 3
4. Laufzeitanalyse

Aufg. 7.03: Laufzeitanalyse von Insertion Sort

Lsg. (4)

```
INSERTIONSORT( A )
1 for j = 2 to length(A)
2 do key = A[j]
3    i = j – 1
4    while i > 0 and A[i] > key
5    do A[i + 1] = A[i]
6       i = i – 1
7    A[i + 1] = key
```

Zeile	Kostenkoeffizient	Zeit (worstcase)	Zeit (bestcase)
1	c_1	n	n
2	c_2	n-1	n-1
3	c_3	n-1	n-1
4	c_4	$(n^2+n)/2 - 1$	n-1
5	c_5	$(n^2-n)/2$	0
6	c_6	$(n^2-n)/2$	0
7	c_7	n-1	n-1

Wann der Bestcase und der Worstcase eintreten hängt vom Algorithmus ab. Hier: Bestcase bei vorsortiert, Worstcase bei umgekehrter Sortierung.
Asymptotische Laufzeit im Best/Worstcase muss für jeden Algorithmus analytisch bestimmt werden, keine allgemein gültigen „Regeln“

Worstcase:
$T(n) = O(n^2)$

Bestcase:
$T(n) = O(n)$

Vielen Dank für Ihre Aufmerksamkeit

Daniel Grimm
Karlsruher Institut für Technology (KIT) – ITIV
daniel.grimm@kit.edu