Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. K. D. Müller-Glaser
Prof. Dr.-Ing. J. Becker
Prof. Dr. rer. nat. W. Stork

Übung04: Informationstechnik (IT)

Tobias Schwalb & Michael Tansella

Teil1: Besprechung der Übungsaufgaben 3.01-3.06

Inhalt: Übung04 – Teil 1

Karlsruher Institut für Technologie

- Aufg. 3.01: Verständnisfragen
- Aufg. 3.02: Funktionen – Deklaration, Prototypen, Aufruf
- Aufg. 3.03: Funktionen und Header-Dateien
- Aufg. 3.04: Programmstrukturen
- Aufg. 3.05: Gültigkeitsbereich
- Aufg. 3.06: Hash-Tabelle

Aufg. 3.01: Verständnisfragen Lsg. (1)

a) Ein Funktionsaufruf ist ein Ausdruck, dessen Typ bestimmt ist durch
   i. die an die Funktion übergebenen Argumente. Falsch
   ii. die im Funktionskopf deklarierten Parameter. Falsch
   iii. den Return-Wert der Funktion. Richtig

b) Der Prototyp einer Funktion stellt dem Compiler Informationen über
   i. den Return-Typ der Funktion bereit. Richtig
   ii. die Namen der Parameter bereit. Falsch
   iii. den Typ jedes Parameters bereit. Richtig

c) Ein Compiler erkennt eine falsche Anzahl von Argumenten nicht. Falsch

Aufg. 3.01: Verständnisfragen Lsg. (2)

d) In C++ ist eine Standard-Header-Datei
   i. eine Objektdatei. Falsch
   ii. eine ausführbare Datei. Falsch
   iii. eine Textdatei. Richtig

e) Zur Ausführung der Anweisung `cout << "Himm…!" << endl;` genügt es, die Header-Datei `iostream` in Ihrem Programm zu inkludieren. Falsch

f) Die in den C-Header-Dateien (Kennung.h) enthaltenen Variablen sind
   i. lokal deklariert. Falsch
   ii. global deklariert. Richtig
   iii. im Namensbereich `std` deklariert. Falsch

Aufg. 3.01: Verständnisfragen Lsg. (3)

g) Ein außerhalb einer Funktion definiertes Objekt wird als global bezeichnet.

h) Eine Funktion kann innerhalb einer anderen Funktion definiert werden. Falsch

Aufg. 3.02: Funktionen - … Lsg. (1)

a) Bestimmen Sie die Fehler in folgenden Prototypen:

i. `double calculate double x, double y;`
Lösung: `double calculate( double x, double y );`

ii. `void myFunc( int n, m );`
Lösung: `void myFunc( int n, int m );`

iii. `int your-Func();`
Lösung: `int your_Func();`

iv. `Bool test( void );`
Lösung: `bool test( void );`

Aufg. 3.02: Funktionen - … Lsg. (2)

b) Welche der folgenden Funktionsaufrufe sind korrekt? Wenn der Funktionsaufruf nicht korrekt ist beschreiben Sie den Fehler.

i.
```
int max( int a, int b, int c );
int result = max( 7, 12 );
```
Lösung: Die Anzahl der Parameter im Prototyp stimmt nicht mit der Anzahl Argumente beim Aufruf der Funktion überein.

ii.
```
double square( double wert );
double x = 2.1;
cout << square( x );
```
Lösung: Richtig

Aufg. 3.02: Funktionen - … Lsg. (3)

b) Welche der folgenden Funktionsaufrufe sind korrekt? Wenn der Funktionsaufruf nicht korrekt ist beschreiben Sie den Fehler.

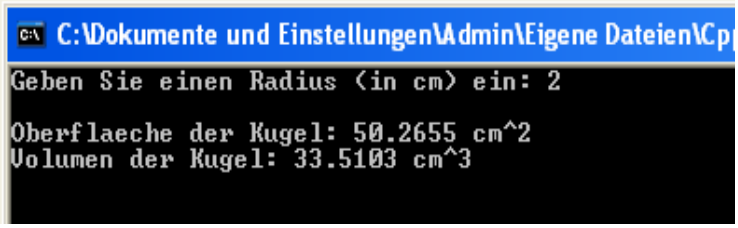
iii.
```
int random( void );
random( 1 );
```
Lösung: Unzulässig. Die Funktion `random()` enthält keine Argumente.

iv.
```
int random( int a );
random( 1 );
```
Lösung: Richtig. Auch wenn die Funktion einen Rückgabewert hat, muss dieser nicht verwendet / zugewiesen werden.

Aufg. 3.03: Funktionen und Header-Dateien

- Erstellen Sie ein Programm, das die Oberfläche und das Volumen einer Kugel mit dem Radius r berechnet. Dazu soll der Radius über die Tastatur eingelesen werden und später das Ergebnis auf dem Bildschirm ausgegeben werden.

- Die Konstante pi (= 3,1415927) und die zwei Funktionen, die jeweils die Oberfläche und das Volumen berechnen, sollen in einer eigenen Header-Datei implementiert werden. Diese soll wiederum in die Hauptdatei eingebunden werden.



$$O(r) = 4 \cdot pi \cdot r^2$$

$$V(r) = \frac{4}{3} \cdot pi \cdot r^3$$

Aufg. 3.03: Funktionen + Header Lsg. (1)

main.cpp

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

using namespace std;

int main() {
  double r = 0;

  cout << "Geben Sie einen Radius (in cm) ein: ";
  cin >> r;

  if( cin == false )
    cout << "Ihre Eingabe ist nicht korrekt!" << endl;
  else if( r < 0 )
    cout << "Sie haben einen negativen Wert eingegeben!" << endl;
  else {
    cout << "Oberflaeche ist: " << Oberflaeche( r ) << " cm2" << endl;
    cout << "Volumen ist: " << Volumen( r ) << " cm3" << endl;
  }
  return 0;
}
```

Radius abfragen und einlesen

Eingaben überprüfen – ungültig / negativ
Funktionen aufrufen / Ergebnisse ausgeben

Aufg. 3.03: Funktionen + Header Lsg. (2)

kugel.h

```
#ifndef _KUGEL_
#define _KUGEL_
  #include <cmath>
  const double PI = 3.1415927;

  double Oberflaeche( double a );
  double Volumen( double b );
#endif
```

Konstanten

Prototypen

kugel.cpp

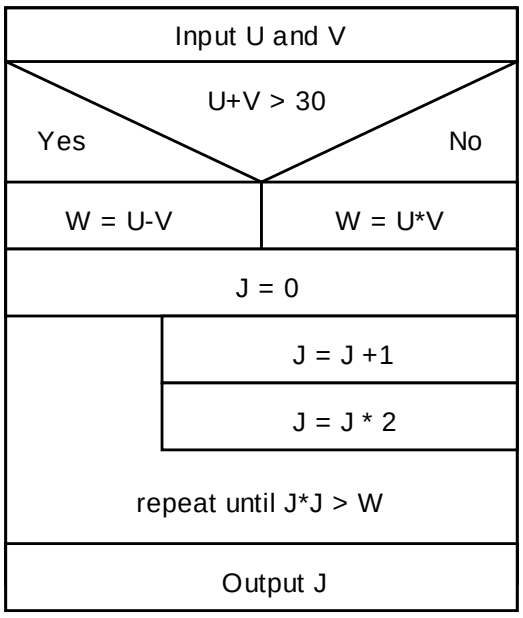
```
#include "kugel.h"

double Oberflaeche( double a ) {
  double erg = 0;
  erg = 4 * PI * pow( a, 2.0 );
  return erg;
}

double Volumen( double b ) {
  double erg = 0;
  erg = 4.0 / 3 * PI * pow( b, 3.0 );
  return erg;
}
```

Implementierung

Funktion in **<cmath>** für x^y

Aufg. 3.04: Programmstrukturen Lsg.

- Ein Algorithmus ist festgelegt durch das folgende Nassi-Shneiderman-Diagramm



Bestimmen Sie den Ausgang J für die folgenden Eingänge (es soll kein Programm geschrieben werden):

a) U = 20, V = 5
   Lösung: 14

b) U = 30, V = 30
   Lösung: 2

c) U = 19, V = 2
   Lösung: 14

Aufg. 3.05: Gültigkeitsbereich Lsg.

```
double fq = 0.6180339887;    //Fibonacci-Quotient
static int b = 10000;

int hash ( unsinged int x ) {
  double temp = x * fq - (int)( x * fq );
  return b * temp;
}
```

fq, globale Variable: steht im ganzen Programm allen Funktionen zur Verfügung

b, statische globale Variable: steht im ganzen Programm allen Funktionen zur Verfügung

static ist an der Stelle überflüssig!
Variable **b** wird nur ein mal angelegt da global!

x, Übergabeparameter → lokale Variable: nur innerhalb der Funktion **hash()** verfügbar

temp, lokale Variable: nur innerhalb der Funktion **hash()** verfügbar

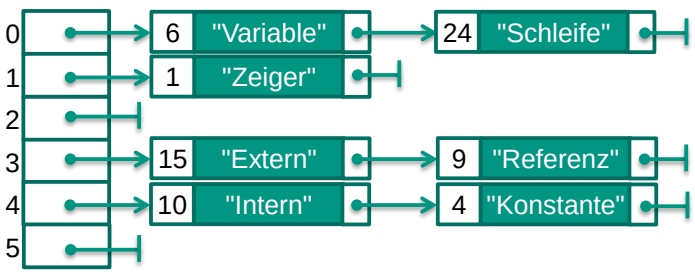
Wegen 0 ≤ **temp** < 1 und **b** = 10000
→ 0 ≤ **b * temp** ≤ 9999

Aufg. 3.06: Hash-Tabelle Lsg. (1)

a) Gegeben sind folgende Schlüssel-Wert-Paare, und eine Hash-Tabelle, die die mathematische Funktion h(k) = k modulo 6 erfüllt. Dabei bezeichnet k den Schlüssel. Kollisionen bei dieser Hash-Tabelle sind durch verkettete Listen aufzulösen.

Schlüssel	Wert
10	"Intern"
15	"Extern"
6	"Variable"
4	"Konstante"
1	"Zeiger"
9	"Referenz"
24	"Schleife"

Ordnen Sie die Schlüssel-Wert-Paare in die Hash-Tabelle ein.



Aufg. 3.06: Hash-Tabelle Lsg. (2)

b) Es wird nun nach einem bestimmten Paar anhand dessen Schlüssels in der Hash-Tabelle gesucht. Beschreiben Sie hierzu eine Vorgehensweise in Form eines Pseudo-Codes.

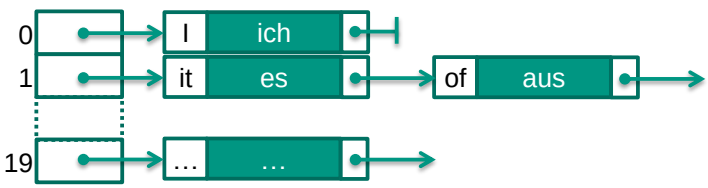
Berechne den Index im Array mit der Hash-Funktion und dem Schlüssel
Wenn die Liste, auf welche das ArrayElement verweist == Leer
Dann Element nicht gefunden & Suche beenden
Sonst Solange Nicht am Ende der Liste angekommen
Wenn Schlüssel des aktuellen Elements == Gesuchter Schlüssel
Dann Element gefunden & Suche beenden
Sonst Gehe zum nächsten Element in der Liste
Wenn Schlüssel des letzten Elements == Gesuchter Schlüssel
Dann Element gefunden & Suche beenden
Sonst Element nicht in der Liste vorhanden & Suche beenden

Aufg. 3.06: Hash-Tabelle Lsg. (3)

c) Gegeben sei ein Englisch→Deutsch Wörterbuch, welches Wörter mit einer Länge von 1 bis 20 Zeichen enthält. Um die Suchzeit nach der Übersetzung eines bestimmten Wortes zu verkürzen, soll eine Hash-Tabelle verwendet werden. Entwerfen Sie ein Konzept für eine Hashtabelle und geben Sie die dazugehörige Hash-Funktion an. Dabei sollen das englische Wort der Schlüssel und die deutsche Übersetzung den dazugehörigen Wert bilden.

Schlüssel (Englische Wörter) werden anhand ihrer Länge in einer Hashtabelle eingeordnet. Die Länge bildet dabei den Index im Array.

h(k) = Länge(k) – 1

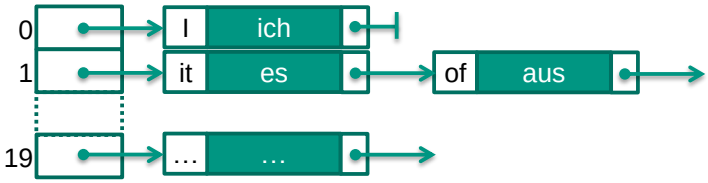


→ Auch andere Möglichkeiten denkbar (Anzahl der Konsonanten, …)

Aufg. 3.06: Hash-Tabelle Lsg. (4)

Schlüssel (Englische Wörter) werden anhand ihrer Länge in einer Hashtabelle eingeordnet. Die Länge bildet dabei den Index im Array.

h(k) = Länge(k) – 1



- Verkürzung der Suchzeit (Betrachtung aller Möglichkeiten)

  Alle Wörter (bei einer Länge von maximal 20): $\sum_{n=1}^{20} 26^n = 2{,}07 \cdot 10^{28}$

  Alle Wörter (bei einer Länge von genau 20): $26^{20} = 1{,}99 \cdot 10^{28}$ $(96{,}15\%)$

  Alle Wörter (bei einer Länge von genau 6): $26^6 = 3{,}09 \cdot 10^8$ $(1{,}49 \cdot 10^{-18}\%)$

  Alle Wörter (bei einer Länge von genau 2): $26^2 = 676$ $(3{,}26 \cdot 10^{-24}\%)$

  → Dieses Verfahren eignet sich allgemein, vor allem für kurze Wörter