WBS TRAINING

Programmierung(2)

UNSER ♥ SCHLÄGT FÜR BILDUNG

Agenda

- Lokale und Globale Variablen
  - Definition
  - Motivation
  - Beispiele
- Hashfunktionen
  - Definition
  - Motivation
  - Beispiele
- Fachpraktische Anwendungen

Lokale und Globale Variablen – Definition

- Wir haben bereits erfahren, dass wir alle Variablen, mit denen innerhalb eines (Haupt-, oder Unter-) Programmes gearbeitet werden sollen, zuvor deklarieren und definieren müssen.
- Bisher haben wir uns aber noch keine Gedanken darüber gemacht „wo“ diese Variablen nach ihrer Deklaration und Definition gültig sind.
- Tatsächlich haben wir bisher durchgehend mit sogenannten „**Lokalen Variablen**“ gearbeitet, die nur innerhalb jenes Blockes gültig sind, in denen diese eingeführt wurden:
  - So hatten wir Variablen schon von Beginn im main deklariert und definiert
    => diese waren dann aber auch nur im main (also nur „lokal“) gültig
  - Später hatten wir auch Variablen in Funktionen (Unterprogrammen) deklariert und definiert
    => diese waren nur innerhalb ihrer Funktion (und also ebenfalls nur „lokal“) gültig
- Im Folgenden werden wir nun aber auch Variablen kennenlernen, die buchstäblich „überall“ (also im gesamten Quellcode) gültig sind. Diese Variablen werden dann entsprechend „**global**“ genannt.
- Da globale Variablen im main UND in allen Funktionen gültig sind, könnte man gerade als Programmieranfänger dazu neigen, diese bevorzugt einzusetzen. Dies sollte aber unbedingt vermieden werden (siehe die Überlegungen zur **Motivation** von lokalen und globalen Variablen)

Lokale und Globale Variablen – Motivation

- So wie Ihre Umschulung „modular“ aufgebaut ist (also aus „Bausteinen“ besteht) so kann auch ein Quellcode „modular“ strukturiert sein, wenn nicht alle Funktionalitäten im main codiert, sondern auf (zahlreiche) Funktionen verteilt wurden.
- **Der Einsatz von Funktionen bietet einige bereits angesprochenen Vorteile:**
  - Der Quellcode wird übersichtlicher
  - Die eingesetzten Funktionen existierten bereits oder können zukünftig wiederverwendet werden
  - Die Modularisierung in unterschiedliche Funktionen erlaubt das parallele Programmieren mehrerer Programmierer
- **Mit dem Einsatz von Haupt- und Unterprogrammen entsteht aber das folgende Problem:**
  - Wenn innerhalb des Hauptprogrammes bestimmte Variablen verwendet werden (z.B. der klassische Schleifenzähler „i“ innerhalb einer FOR-Schleife) und die selben Variablennamen auch in Funktionen verwendet werden, die aus dem Hauptprogramm heraus aufgerufen werden, so würde es zu Konflikten kommen, sofern das Programm nicht entscheiden könnte, „welches i“ jeweils gerade gemeint sei.
  - Hinzu kommt, dass der Programmierer einer (wiederverwertbaren) Funktion nicht wissen kann, innerhalb welcher Programme seine Funktion zukünftig verwendet werden wird. Daher kann er dann aber auch nicht wissen, welche Variable-Bezeichnungen dort verwendet werden.
  - Umgekehrt würde die Einsetzbarkeit von Funktionen extrem leiden, wenn Programmierer stets darauf zu achten hätten, welche Variable-Bezeichnungen für sie „noch“ zulässig seien und welche bereits von einer Funktion „besetzt“ wären.

Lokale und Globale Variablen – **Motivation**

- Diese Probleme können umgangen werden, wenn Variablen nur eine „lokale Gültigkeit“ besitzen.

  => eine lokale Variable „i“ im main stünde dann nicht mehr im Konflikt mit einer anderen Variable „i“ aus einer Funktion
- Beide Variablen hätten in diesem Fall zwar die selbe Bezeichnung, würden ihre Werte aber an **unterschiedlichen Speicherstellen** ablegen, so dass es intern zu keinem Missverständnis kommen kann, da stets klar ist, „welches i“ an welcher Stelle jeweils gemeint sei. (Wir werden uns hierzu natürlich im Folgenden noch konkrete Beispiele anschauen)
- Wenn irgend möglich, so sollten also in Haupt- und Unter-Programmen ausschließlich lokale Variablen eingesetzt werden, um entsprechende Konflikte von vornherein auszuschließen.
- Dennoch kann es in Einzelfällen durchaus sinnvoll sein, auch „Globale Variablen“ einzuführen. Dies ist der Fall, wenn ein Wert über den gesamten Quellcode hinweg zur Verfügung stehen muss (z.B. der Highscore eines Spielprogrammes).
- Falls man allerdings im Rahmen einer möglichst seltenen Ausnahme eine globale Variable verwendet, so bietet es sich an, diese mit einem „exotischen“ Namen zu versehen, um unbeabsichtigte Namensgleichheiten zu vermeiden.
- Falls nämlich eine globale Variable eingeführt wurde, und innerhalb des main oder einer Funktion eine weitere gleichnamige lokale Variable deklariert wird, so gilt die sogenannte „**Überdeckung**“, nach der die globale Variable lokal ausgeblendet wird und nur die lokale Variable gilt! (Auch hierzu werden wir uns im Folgenden ein Beispiel anschauen)

Lokale und Globale Variablen – **Beispielaufgabe(1)**

Aufgabenstellung

- Das Programm besteht aus einer einzigen FOR-Schleife, die 5-mal durchlaufen wird.
- Pro Durchlauf wird …
  - der aktuelle Zählerwert der Schleife auf der Konsole ausgegeben
  - die Funktion „loop“ aufgerufen:
    - Die Funktion besteht ebenfalls aus einer einzigen FOR-Schleife, die 5-mal durchlaufen wird.
    - Pro Durchlauf der (Funktion)-Schleife wird der aktuelle Zählerwert auf der Konsole ausgegeben

Hinweis:

Bei der Besprechung aller Beispiele der heutigen Vorlesung werden wir auf die entsprechende Darstellungen als **PAP**, **Struktogramm** oder **Pseudocode** verzichten können, da hierzu keine eigenständige Symbolik existiert. Zudem wird der Gültigkeitsbereich von Variablen innerhalb der graphischen Darstellung von Programmen üblicherweise nicht problematisiert.

Lokale und Globale Variablen – **Beispielaufgabe(1a)** – **Quellcode**

```
#include<stdio.h>

loop()
{
    int i;
    for(i=0;i<5;i++)
    {
        printf("i aus der Funktion: %d\n",i);
    }
}

main()
{
    int i;
    for(i=0;i<5;i++)
    {
        printf("i aus dem main: %d\n",i);
        loop();
    }
}
```

```
i aus dem main: 0
i aus der Funktion: 0
i aus der Funktion: 1
i aus der Funktion: 2
i aus der Funktion: 3
i aus der Funktion: 4
i aus dem main: 1
i aus der Funktion: 0
i aus der Funktion: 1
i aus der Funktion: 2
i aus der Funktion: 3
i aus der Funktion: 4
i aus dem main: 2
i aus der Funktion: 0
i aus der Funktion: 1
i aus der Funktion: 2
i aus der Funktion: 3
i aus der Funktion: 4
i aus dem main: 3
i aus der Funktion: 0
i aus der Funktion: 1
i aus der Funktion: 2
i aus der Funktion: 3
i aus der Funktion: 4
i aus dem main: 4
i aus der Funktion: 0
i aus der Funktion: 1
i aus der Funktion: 2
i aus der Funktion: 3
i aus der Funktion: 4
```

Lokale und Globale Variablen – **Beispielaufgabe(1b)** – **Quellcode**

```
#include<stdio.h>

int i; // Position der Globalen Variable-Deklaration:
Unterhalb der Präprozessoranweisungen und oberhalb der
Funktionen

loop()
{
    // int i;
    for(i=0;i<5;i++)
    {
        printf("i aus der Funktion: %d\n",i);
    }
}

main()
{
    // int i;
    for(i=0;i<5;i++)
    {
        printf("i aus dem main: %d\n",i);
        loop();
    }
}
```

Unerwünschte Ausgabe:

```
i aus dem main: 0
i aus der Funktion: 0
i aus der Funktion: 1
i aus der Funktion: 2
i aus der Funktion: 3
i aus der Funktion: 4
```

Erläuterung:
Der globale Wert von i wird durch die Schleife in der Funktion bis auf den Wert 5 hochgezählt
=>
Die Schleife im main bricht bereits Nach dem ersten Durchlauf ab.

WBS TRAINING

Lokale und Globale Variablen – **Beispielaufgabe(2)**

Aufgabenstellung

- Das Programm besteht erneut aus einer einzigen Schleife, die 1000 mal durchlaufen wird. Pro Durchlauf:
  - wird eine Zufallszahl **x** zwischen 1 und 100 ausgelost
  - wird die (globale!) Variable **max** auf das neue Maximum aktualisiert, falls x größer als der bisherige Wert von max ist
  - wird die Funktion intMAX aufgerufen, falls x =100
  - wird der aktuelle Wert von max ausgegeben, falls sich max geändert hat
- Für die Funktion intMAX gilt:
  - intMax hat keinen Übergabewert und keinen Rückgabewert
  - intMax setzt die (globale!) Variable **max** auf 0

Lokale und Globale Variablen – **Beispielaufgabe(2a)** – **Quellcode**

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


int max;

initMAX()
{
   max=0;
}

main()
{
   srand(time(NULL));
   rand();

   int i,x;

   initMAX();
   for(i=0;i<1000;i++)
   {
      x=rand()%100+1;
      if(x>max)
      {
         max=x;
         if(x==100) initMAX();
         printf("Aktuelles Maximum: %d\n",max);
      }
   }
}
```

```
Aktuelles Maximum: 32
Aktuelles Maximum: 85
Aktuelles Maximum: 92
Aktuelles Maximum: 93
Aktuelles Maximum: 97
Aktuelles Maximum: 99
Aktuelles Maximum: 0
Aktuelles Maximum: 52
Aktuelles Maximum: 69
Aktuelles Maximum: 86
Aktuelles Maximum: 95
Aktuelles Maximum: 97
Aktuelles Maximum: 99
Aktuelles Maximum: 0
Aktuelles Maximum: 75
Aktuelles Maximum: 0
Aktuelles Maximum: 84
Aktuelles Maximum: 90
Aktuelles Maximum: 99
Aktuelles Maximum: 0
Aktuelles Maximum: 20
Aktuelles Maximum: 86
Aktuelles Maximum: 94
Aktuelles Maximum: 96
Aktuelles Maximum: 98
Aktuelles Maximum: 0
Aktuelles Maximum: 73
Aktuelles Maximum: 86
Aktuelles Maximum: 90
Aktuelles Maximum: 95
Aktuelles Maximum: 96
Aktuelles Maximum: 0
Aktuelles Maximum: 15
Aktuelles Maximum: 72
Aktuelles Maximum: 94
Aktuelles Maximum: 97
Aktuelles Maximum: 99
```

Lokale und Globale Variablen – **Beispielaufgabe(2b)** – **Quellcode**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


//int max;

initMAX()
{
   int max;
   max=0;
}

main()
{
   srand(time(NULL));
   rand();

   int max;
   int i,x;

   initMAX();
   for(i=0;i<1000;i++)
   {
      x=rand()%100+1;
      if(x>max)
      {
         max=x;
         if(x==100) initMAX();
         printf("Aktuelles Maximum: %d\n",max);
      }
   }
}
```

Unerwünschte Ausgabe:

```
Aktuelles Maximum: 91
Aktuelles Maximum: 96
Aktuelles Maximum: 98
Aktuelles Maximum: 100
```

Erläuterung:
initMax wird bedeutungslos, da
die lokale Variable max in initMAX
ungleich der lokalen Variable max
des Hauptprogrammes.
(max bleibt daher **im main** undefiniert)
=>
max im Hauptprogramm wird
nicht mehr auf 0 gesetzt
Daher endet die Ausgabe nach dem
Erstmaligen Auslosen von „100“

(falls es überhaupt zu einer Ausgabe kommt, was nicht der Fall wäre, wenn max „zufälligerweise“ einen Startwert > 100 hätte)

Überdeckung – Lokale und Globale Variablen – **Beispielaufgabe(2c)** – Quellcode

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


int max;

initMAX()
{
   int max;
   max=0;
}

main()
{
   srand(time(NULL));
   rand();

   // int max;
   int i,x;

   initMAX();
   for(i=0;i<1000;i++)
   {
      x=rand()%100+1;
      if(x>max)
      {
         max=x;
         if(x==100) initMAX();
         printf("Aktuelles Maximum: %d\n",max);
      }
   }
}
```

Globale Variable „max“

Lokale Variable „max“
(hier gilt nur die lokale Variable, denn die globale wird überdeckt)

Hier gilt **nur die globale** Variable, denn eine gleichnamig lokale wurde im main nicht deklariert
⇒ KEINE Überdeckung
⇒ initMax() bleibt wirkungslos
⇒ eventuell gibt es keine Ausgabe
(falls der „zufällige“ Startwert von max >100)

Unerwünschte Ausgabe:

```
Aktuelles Maximum: 91
Aktuelles Maximum: 96
Aktuelles Maximum: 98
Aktuelles Maximum: 100
```

Erläuterung:
initMax wird bedeutungslos, da die lokale Variable max in initMAX die globale Variable max **überdeckt**
=>
max im Hauptprogramm wird nicht mehr auf 0 gesetzt

WBS TRAINING

Hashfunktion – Definition

- Das Verb „to hash“ bedeutet soviel wie „zerhacken“.
- Und genau dies ist auch die Aufgabe einer Hashfunktion, die einer (in der Regel) sehr großen Anzahl von möglichen Eingabewerten eine (in der Regel) deutlich geringere Anzahl von Rückgabewerten zuordnet, und also anschaulich gesprochen einen „breiten Input“ in einen „schmalen Output“ zerhackt.
- Formal heißt dies: Hashfunktionen sind üblicherweise „nicht injektiv“, denn sehr unterschiedliche Eingabewerte können im Einzelfall zum identischen Rückgabewert führen.
- Der Rückgabewert einer Hashfunktion wird als „**Hashwert**“ bezeichnet.
- Aus Sicht eines Programmierers sind Hashfunktionen allerdings „ganz gewöhnliche“ Funktionen, so dass wir bei diesem Thema inhaltlich nichts neues lernen werden müssen.
- Stattdessen werden zum Ende dieser Vorlesung zwei Beispielaufgaben vorgestellt, an Hand derer typische Anwendungsfälle von Hashfunktionen demonstriert werden können.
- Zu diesem Zweck werden wir zuvor aber noch 2 wichtige Motivationen für den Einsatz von Hashfunktionen ansprechen, die wir dann entsprechend bei den Beispielaufgaben aufgreifen werden

Hashfunktion – Motivation

- Überall dort, wo wir mit **Codewerten** (Kundennummer, ID, Geheimnummer, Personalausweisnummer …) arbeiten, kann der Wunsch bestehen …
  a) deren Korrektheit zu testen, und/oder
  b) versehentliche Tippfehler bei deren Eingabe erkennen zu können

  In solchen Fällen bietet es sich dann aber an, dem eigentlichen Codewert eine „**Kontrollnummer (Hashwert)**“ zuzuweisen, so dass nur die Kombination aus „Codenummer + zulässige Kontrollnummer“ akzeptiert wird, während alle anderen Kombinationen als „Fälschung“ oder „Tippfehler“ erkannt werden können.

- Ferner können Hashwerte im Zusammenhang mit vorliegenden Codewerten aber auch dazu genutzt werden, eine **Zuordnung** vorzunehmen:
  a) Ausgehend von **Personalausweisnummer X** kann mittels der berechneten **Sachbearbeiter-Nummer Y (Hashwert)** ermittelt werden, welcher Sachbearbeiter einer Behörde für welchen Bürger zuständig ist.
  b) Ausgehend von **Mitarbeiternummer X** kann mittels der berechneten **Regionen-Kennziffer Y (Hashwert)** ermittelt werden, welcher Mitarbeiter in welcher Region eingesetzt wird.
  c) **…**

Hashfunktion – **Beispielaufgabe(1)**

Aufgabenstellung

- Das Programm fragt **4 einstellige positive ganzen Zahlen** ab und speichert diese in den Variablen **a**, **b**, **c** und **d**.

  (diese 4 Ziffern könnten zum Beispiel eine **4-stellige Mitarbeiternummer** darstellen)
- Anschließend wird eine **Hashfunktion** aufgerufen:
  - Funktionsname: ….. hash1
  - Übergabewerte: ….. Integer a, b, c und d
  - Funktionalität: …….. berechnet die Summe aller 4 Variablen und ermittelt davon den Rest Modulo 10
  - Rückgabewert: ……. der ermittelte Rest
- Der Rückgabewert (Hashwert) von hash1 wird auf der Konsole ausgegeben und das Programm endet.

  (dieser Hashwert könnte zum Beispiel als eine **Regionen-Kennziffer** interpretiert werden)

Hinweis:

Wie bereits angesprochen, werden wir auf eine Darstellung als **PAP**, **Struktogramm** oder **Pseudocode** verzichten.

Hashfunktion – **Beispielaufgabe(1)** – **Quellcode**

```
#include <stdio.h>


int hash1(int a, int b, int c, int d)
{
   int hashwert;
   hashwert=(a+b+c+d)%10;
   return hashwert;
}

main()
{
   int i,arr[4],hashwert;

   for(i=0;i<4;i++)
   {
      printf("Geben Sie bitte die %d. Ziffer ein: ",i+1);
      fflush(stdin);
      scanf("%d",&arr[i]);
   }

   hashwert=hash1(arr[0],arr[1],arr[2],arr[3]);

   printf("\nDer Mitarbeiter mit Nr. %d%d%d%d wird in Region %d eingesetzt\n",arr[0],arr[1],arr[2],arr[3],hashwert);
}
```

```
Geben Sie bitte die 1. Ziffer ein: 4
Geben Sie bitte die 2. Ziffer ein: 7
Geben Sie bitte die 3. Ziffer ein: 1
Geben Sie bitte die 4. Ziffer ein: 1

Der Mitarbeiter mit Nr. 4711 wird in Region 3 eingesetzt
```

WBS TRAINING

Hashfunktion – **Beispielaufgabe(2)**

Aufgabenstellung

- Das Programm fragt **4 einstellige positive ganzen Zahlen** ab und speichert diese in den Variablen **a**, **b**, **c** und **d**.

  (die ersten 3 Ziffern könnten zum Beispiel eine **3-stellige Mitarbeiternummer** darstellen, die vierte Ziffer könnte die **Kontrollnummer** sein)
- Anschließend wird eine **Hashfunktion** aufgerufen:
  - Funktionsname: ….. hash2
  - Übergabewerte: ….. Integer a, b, c und d
  - Funktionalität: …….. berechnet den Hashwert: Summe aus a, b und c und davon den Rest Modulo 10
  - Rückgabewert: ……. 0, falls Hashwert UNGLEICH d (=> der 4-stellige Mitarbeitercode ist inkorrekt)
    1, falls Hashwert GLEICH d (=> der 4-stellige Mitarbeitercode ist korrekt)
- In Abhängigkeit vom Rückgabewert wird auf der Konsole ausgegeben:
  - „Mitarbeitercode ist korrekt“ (falls Hashwert=1)
  - „Mitarbeitercode ist inkorrekt“ (falls Hashwert=0)

Hashfunktion – **Beispielaufgabe(2)** – **Quellcode**

```
#include <stdio.h>

int hash2(int a, int b, int c, int d)
{
   int hashwert;
   hashwert=(a+b+c)%10;
   if(hashwert!=d) return 0;
   return 1;
}

main()
{
   int i,arr[4],rueckgabewert;

   for(i=0;i<4;i++)
   {
      printf("Geben Sie bitte die %d. Ziffer ein: ",i+1);
      fflush(stdin);
      scanf("%d",&arr[i]);
   }

   rueckgabewert=hash2(arr[0],arr[1],arr[2],arr[3]);

   if(rueckgabewert==0) printf("\nDer Mitarbeitercode %d%d%d%d ist NICHT korrekt.\n",arr[0],arr[1],arr[2],arr[3]);
   else printf("\nDer Mitarbeitercode %d%d%d%d ist korrekt.\n",arr[0],arr[1],arr[2],arr[3]);
}
```

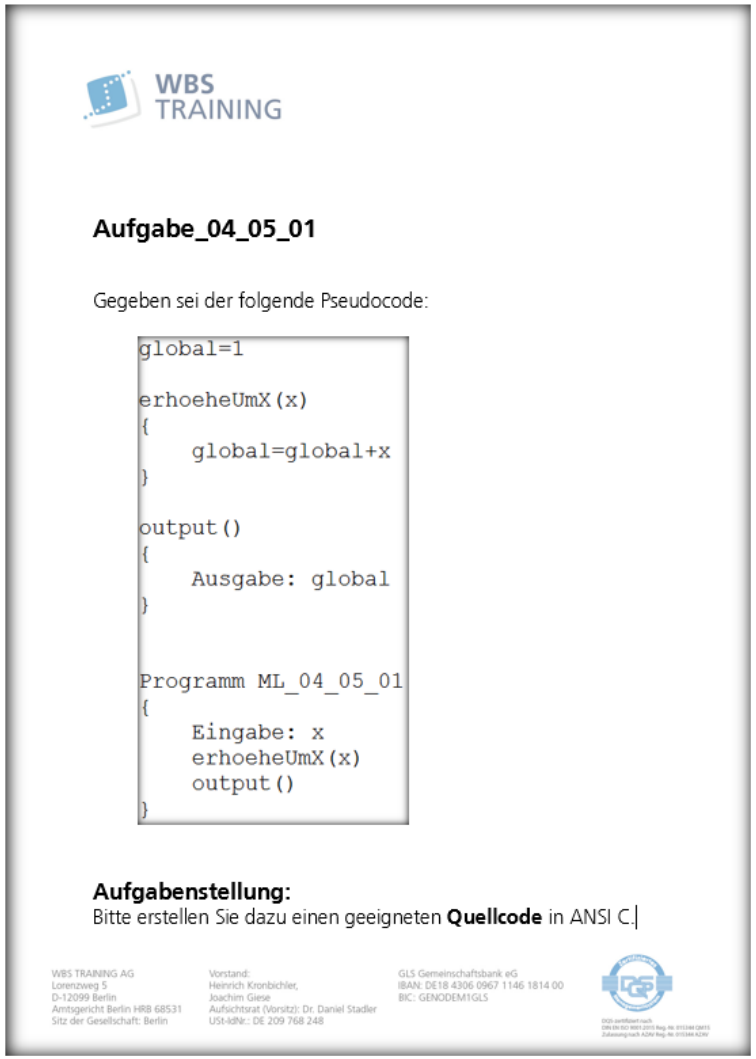
```
Geben Sie bitte die 1. Ziffer ein: 1
Geben Sie bitte die 2. Ziffer ein: 2
Geben Sie bitte die 3. Ziffer ein: 3
Geben Sie bitte die 4. Ziffer ein: 4

Der Mitarbeitercode 1234 ist NICHT korrekt.
```

```
Geben Sie bitte die 1. Ziffer ein: 4
Geben Sie bitte die 2. Ziffer ein: 7
Geben Sie bitte die 3. Ziffer ein: 1
Geben Sie bitte die 4. Ziffer ein: 2

Der Mitarbeitercode 4712 ist korrekt.
```

Globale Variable – Gemeinsame Übung A_04_05_01



WBS TRAINING

Aufgabe_04_05_01

Gegeben sei der folgende Pseudocode:

```
global=1

erhoeheUmX(x)
{
    global=global+x
}

output()
{
    Ausgabe: global
}


Programm ML_04_05_01
{
    Eingabe: x
    erhoeheUmX(x)
    output()
}
```

Aufgabenstellung:
Bitte erstellen Sie dazu einen geeigneten **Quellcode** in ANSI C.

WBS TRAINING AG
Lorenzweg 5
D-12099 Berlin
Amtsgericht Berlin HRB 68531
Sitz der Gesellschaft: Berlin

Vorstand:
Heinrich Kronbichler,
Joachim Giese
Aufsichtsrat (Vorsitz): Dr. Daniel Stadler
USt-IdNr.: DE 209 768 248

GLS Gemeinschaftsbank eG
IBAN: DE18 4306 0967 1146 1814 00
BIC: GENODEM1GLS

WBS TRAINING

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!