WBS TRAINING

Programmierung(2)

UNSER ♥ SCHLÄGT FÜR BILDUNG

Agenda

- Pointer
  - Definition
  - Motivation
  - Referenzieren
  - Dereferenzieren
  - Typ eines Pointers
- Call-by-Reference
  - Definition
  - Veranschaulichung
  - Motivation
  - Beispiele
    - Einfache Variablen
    - Arrays
    - Strings
- Fachpraktische Anwendungen

Pointer – Definition

- Innerhalb des Rechners besitzt jede Speicherstelle eine **eindeutige Adresse**.
- Unter einem **Pointer** versteht man eine Variable, die eine solche Adresse abspeichert.
- **Unterschiedliche** (lokale) **Variablen** (aus unterschiedlichen Anweisungsblöcken) können zwar die selbe Bezeichnung besitzen, ihre **Adressen** sind aber **unterschiedlich**.
- Jeder deklarierten und definierten Variable wird von der **Speicherverwaltung** eine Adresse zugewiesen, die wir bei Bedarf auslesen können.
- Bei Variablen können wir also zwischen dem Wert (**Value**) und der Adresse (**Referenz**) unterscheiden. Der Wert spricht davon „was“ die Variable abspeichert, die Adresse teilt mit „wo“ dieser Wert im Rechner abgespeichert wird.

Pointer – Motivation

- Für den Einsatz von Pointern kann es sehr unterschiedliche Motivationen geben. Viele davon werden wir in diesem Grundlagen-Kurs nicht behandeln, Stichworte für Interessierte:
  - Dynamische Speicherverwaltung
  - Listen
  - Files

- In diesem Kurs werden wir uns stattdessen im Wesentlichen mit den beiden folgenden Anwendungen von Pointern beschäftigen:
  - Zugriff auf lokale Variablen außerhalb ihres Gültigkeitsbereiches (das wird schon heute unser Thema sein)
  - Verweis von einer Strukturvariable auf eine andere (damit werden wir uns erst in einigen Tagen beschäftigen)

Pointer – **Referenzieren**

- Wenn wir eine „normale“ Variable (vom Typ Integer, Character, …) deklarieren und definieren, dann bedeutet der Aufruf ihrer **Bezeichnung**, dass wir von ihrem **Value** sprechen, Beispiel:

```
int x;
x=10;
printf("%d", x); // Hier wird der (Integer-) WERT (Value) von x ausgegeben
```

- Falls wir hingegen von einer „normalen“ Variable nicht den Wert, sondern deren **Adresse** ansprechen wollen, so müssen wir diese **referenzieren**. Dies geschieht mittels Adress-Operator **&**:

```
int x;
x=10;
printf("%p", &x); // Hier wird (als Pointer) die ADRESSE von x ausgegeben
```

Pointer – **Dereferenzieren**

- Wenn wir umgekehrt einen **Pointer** einführen, dann sprechen wir mit dessen Bezeichnung die **Adresse** an, die dieser Pointer abspeichert.

```
int x;
int* pointer;                 // der Stern deutet an, dass hier keine Integer-Variable, sondern ein Pointer eingeführt wird
pointer=&x;                   // hier wird die Adresse von x in pointer abgespeichert
printf("%p", pointer);    // Hier wird der WERT von pointer (also die abgespeicherte Adresse von x) ausgegeben
```

- Falls wir hingegen von einem Pointer nicht die abgespeicherte Adresse, sondern den (an dieser Adresse gespeicherten) **Inhalt** ansprechen wollen, so müssen wir den Pointer **dereferenzieren**.

  Dies geschieht mittels Derefenzierungs-Operator ***:**

```
int x;
x=10;
pointer=&x;
printf("%d", *pointer); // Hier wird (als Integerwert) der INHALT der (von pointer gespeicherten) Adresse ausgegeben.
```

Pointer – Typ eines Pointers

- Da wir unter einem Pointer eine Variable verstehen, die Adressen von Speicherplätzen abspeichert, und es für eine Adresse unbedeutend ist, „was“ sie speichert, könnte man annehmen, dass man bei Pointern nicht zwischen den **Typen** (int, char, float, double …) unterscheiden müsse.
- Dies stimmt allerdings nur zum Teil. So ist es zwar einerseits richtig, dass eine Speicherstelle einen Byte darstellt, für den unerheblich ist, ob der innerhalb der 8 Bit abgespeicherte Binärwert als Ganzzahl, Zeichen oder Kommazahl interpretiert wird, tatsächlich aber ist der Speicherbedarf dieser Typen unterschiedlich und umfasst zumeist **mehr als eine einzige Speicherstelle**!
- Wenn daher von (z.B.) einem Integer-Pointer gesprochen wird, so ist damit streng genommen nur die „**Startadresse**“ gemeint, denn ein Integer umfasst (üblicherweise) einen Speicherbedarf von 4 unmittelbar aufeinanderfolgenden Bytes und nur die Adresse des ersten Byte wird im Pointer abgespeichert.
- Selbiges gilt auch für FLOAT (4 Byte) und DOUBLE (8 Byte), während Character nur 1 Byte benötigen.
- Für einen Pointer muss also nicht nur die Startadresse, sondern auch der Typ bekannt sein, damit klar ist, wie viele Byte hinter der Startadresse zu jener Variablen gehören, auf die der Pointer „zeigt“.

Pointer – Typ eines Pointers

Für alle **Pointer-Typen** gilt, dass diese durch den Zusatz eines Sternes (*****) notiert werden:

```
int         i;
char        c;
float       f;
double      d;

int*        integer_pointer=&i;
char*       character_pointer=&c;
float*      float_pointer=&f;
double*     double_pointer=&d;

// Ausgabe aller ADRESSEN
printf("%p %p %p %p", integer_pointer, character_pointer, float_pointer, double_pointer);

// Ausgabe aller INHALTE an diesen Adressen
printf("%d %c %f %lf", *integer_pointer, *character_pointer, *float_pointer, *double_pointer);
```

Call-by-Reference – Definition

- Wir haben bereits mit Funktionen gearbeitet, die mit Übergabewerten (Parametern) aufgerufen wurden. Diese Parameter waren allerdings **Werte** (Value), keine Adressen (Reference) daher waren unsere bisherigen Funktionsaufrufe von der Form „**Call-by-Value**“.
- Funktionsaufrufe der Form „Call-by-Value“ erhalten also lediglich einen Wert, ohne „wissen zu können“, von welcher Variable dieser Wert stammt. Call-by-Value-Funktionen können daher **keinen Einfluss** auf diese Variablen nehmen.

- Verwendet man hingegen Funktionen, deren Parameter **Pointer** sind, so spricht man beim Aufruf dieser Funktionen von „**Call-by-Reference**“.
- Funktionsaufrufe der Form „Call-by-Reference“ erhalten als Übergabewerte also Adressen. Wenn an diesen Adressen der Inhalt von Variablen gespeichert wurde, dann kann durch einen solchen Funktionsaufruf **Einfluss** auf genau diese Variablen genommen werden.

Call-by-Reference – Veranschaulichung

- Der Unterschied zwischen einem Funktionsaufruf der Form Call-by-Value und Call-by-Referenz kann möglicherweise durch das folgende Beispiel verdeutlicht werden:

Beispiel (Version 1: „Call-by-Value“)

Angenommen, ich teile Ihnen meinen Kontostand mit, dann erfahren sie eine Zahlenwert (also einen **Value**). Wenn ich Sie dann bitte, diesen Wert mit 100 zu multiplizieren, so erhalten Sie mit dem Ergebnis dieser Rechnung zwar eine vergleichsweise große Zahl, … **an meinem Kontostand** änderte sich aber natürlich (leider ☺) **gar nichts!**

- Der Grund ist leicht ausgemacht:

Sie haben lediglich mit einer Zahl herum gespielt, nicht mit einem realen Inhalt. Anschaulich gesprochen „wusste“ der Taschenrechner nicht, ob sie mit 1,80 meine Körpergröße, meinen Kontostand oder irgend etwas anderes meinten. Daher wurde durch die Multiplikation 1,80 * 100 nur ein Value verändert, nicht mein Kontostand, noch meine Körpergröße.

Call-by-Reference – Veranschaulichung

- Betrachten wir nun das selbe Beispiel unter anderen Vorzeichen:

  Beispiel (Version 2: „Call-by-Reference“)

  Angenommen, ich teile Ihnen „die Adresse meines Kontos“ (also meine Kontodaten) mit, dann erfahren Sie eine **Referenz**. Wenn ich Sie dann bitte, den dort vorgefundenen Kontostand auf das 100-fache zu steigern, dann werden Sie dies natürlich einerseits im Interesse einer erfolgreichen Bausteinprüfung tun ☺ … und andererseits **tatsächlich Einfluss auf meinen Kontostand** genommen haben!

- Der entscheidende Unterschied:

  Sie haben nun nicht nur einen Value (Kontostand) erhalten, mit dem Sie „herumspielen“ können, ohne den Inhalt einer Speicherstelle (Konto) tatsächlich zu ändern, **sondern** die Adresse (Kontodaten) einer Speicherstelle (Konto) erfahren. Wenn Sie den Inhalt dieser Speicherstelle dann aber manipulieren, so haben Sie tatsächlich Einfluss auf jene Variable genommen, die an dieser Adresse abgespeichert wird.

Call-by-Reference – Motivation

- Wir haben uns bereits mit **Lokalen** und **Globalen Variablen** beschäftigt.
- Ebenso haben wir bereits festgestellt, dass Globale Variablen **nur in Ausnahmefällen** verwendet werden sollten, um Konflikte zu vermeiden, die sich bei Namensgleichheit weiterer (lokaler) Variablen ergeben könnten.
- Dennoch werden wir im Einzelfall auch Funktionen schreiben wollen, die **Einfluss auf den Inhalt einer lokalen Variable** des main nehmen können, **obwohl** diese lokale Variable ja innerhalb der Funktion (und also außerhalb des main) **nicht gültig** ist. Genau dies wird uns dann aber mit Hilfe von **Call-by-Reference Funktionen** gelingen.
- Ferner werden wir uns für Funktionen interessieren, bei denen mindestens einer der **Übergabewerte** ein (lokales) **Array** des main repräsentiert. In solchen Fällen werden wir stets mit Call-by-Reference Funktionen arbeiten müssen.
- Da **Strings** (als Inhalt eines Character-Arrays) aus dieser Sicht dann aber nur einen Sonderfall darstellen, werden wir entsprechend auch bei all jenen Funktionen mit Call-by-Reference arbeiten müssen, die als Übergabewerte mindestens einen String erhalten.

Call-by-Reference – Beispielaufgabe(1)

Aufgabenstellung

- Das Programm initialisiert zu Beginn einen Character c. Dessen Wert wird zur Kontrolle auf der Konsole ausgegeben. Anschließend wird die Funktion nextCHAR(&c) aufgerufen. Abschließend wird erneut der Wert von c ausgegeben und das Programm endet.

  Funktionsdefinition

  Funktionsname: ……. nextCHAR

  Übergabewerte: ……. 1 Pointer auf ein Character

  Funktionalität: …… erhöht den Inhalt des Characters um 1 (nächstes Zeichen im ASCII-Code)

  Rückgabewert: ……... keiner

Hinweis:

Bei der Besprechung aller Beispiele werden wir erneut auf die entsprechende Darstellungen als **PAP**, **Struktogramm** oder **Pseudocode** verzichten können, da hierzu keine eigenständige Symbolik existiert.

Call-by-Reference – **Beispielaufgabe(1)** – Quellcode

```
#include <stdio.h>

nextCHAR(char* pointer)
{
   *pointer=*pointer+1; // DEREFERENZIERUNG des
Pointers
}

int main(void)
{
   char c='A';

   printf("VOR dem Funktionsaufruf: c=%c\n",c);
   nextCHAR(&c); // Übergabewert ist der POINTER von c
   printf("NACH dem Funktionsaufruf: c=%c\n",c);

   printf("\n\n\n\n");
   system("pause");
   return 0;
}
```

```
VOR dem Funktionsaufruf: c=A
NACH dem Funktionsaufruf: c=B
```

WBS TRAINING

Call-by-Reference – Beispielaufgabe(2)

Aufgabenstellung

- Das Programm deklariert und definiert zu Beginn im main das Array **int arr[10]**. Anschließend wird die Funktion initialisiereINT(arr,10) aufgerufen. Abschließend werden zur Kontrolle alle Felderwerte von arr ausgeben und das Programm endet.

Funktionsdefinition

Funktionsname: ……. initialisiereINT

Übergabewerte: ……. 1 Pointer auf die Startadresse eines Integer-Arrays

1 Integer x

Funktionalität: ……. initialisiert die ersten x Felder des referenzierten Arrays mit dem Wert 0

Rückgabewert: …….. keiner

Call-by-Reference – **Beispielaufgabe(2)** – Quellcode

```
#include <stdio.h>

initialisiereINT(int arr[ ], int x) // alternativ auch möglich:
                                    // initialisiereINT(int* arr, int x)
{
    int i;
    for(i=0;i<x;i++)
    {
        arr[i]=0;
    }
}

int main(void)
{
    int i,arr[10];

    initialisiereINT(arr,10);  // alternativ auch möglich:
                               // initialisiereINT(&arr[0],10);

    for(i=0;i<10;i++)
    {
        printf("arr[%d]=%d\n",i,arr[i]);
    }

    printf("\n\n\n\n");
    system("pause");
    return 0;
}
```

```
arr[0]=0
arr[1]=0
arr[2]=0
arr[3]=0
arr[4]=0
arr[5]=0
arr[6]=0
arr[7]=0
arr[8]=0
arr[9]=0
```

WBS TRAINING

Call-by-Reference – Beispielaufgabe(3)

Aufgabenstellung

- Das Programm initialisiert ein Character-Array mit einem String und einen Character mit einem Zeichen. Anschließend wird die Funktion anzahlX(string,zeichen) aufgerufen und der Rückgabewert auf der Konsole ausgegeben. Danach endet das Programm.

Funktionsdefinition

Funktionsname: ……. anzahlX

Übergabewerte: ……. 1 Pointer auf die Startadresse eines Character-Arrays, das mit einem String gefüllt ist

1 Character X

Funktionalität: ……. zählt, wie oft das in X gespeicherte Zeichen im String vorkommt

Rückgabewert: …….. die ermittelte Anzahl

Hinweis:

Die Länge des Strings muss nicht als Parameter übergeben werden, da die Funktion dies mittels Terminator erkennt.

Call-by-Reference – **Beispielaufgabe(3)** – Quellcode

```
#include <stdio.h>
#include <string.h>

int anzahlX(char text[ ], char X)
{
    int i=0,anzahl=0;
    while(text[i]!='\0')
    {
        if(text[i]==X) anzahl++;
        i++;
    }
    return anzahl;
}

int main(void)
{
    char string[101];
    strcpy(string,"Ein Beispielsatz");
    char zeichen='e';

    int anzahl=anzahlX(string,zeichen);
    printf("Anzahl des Zeichens '%c' im Text \"%s\" ist: %d",zeichen,string,anzahl);

    printf("\n\n\n\n");
    system("pause");
    return 0;
}
```

```
Anzahl des Zeichens 'e' im Text "Ein Beispielsatz" ist: 2
```

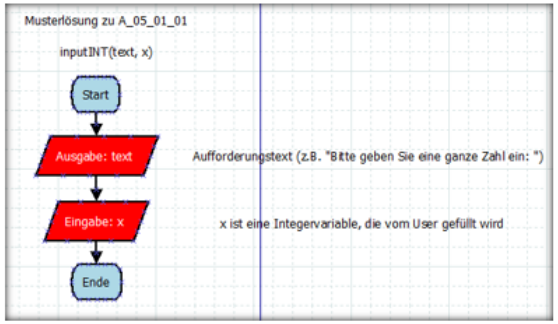
WBS TRAINING

Call-by-Reference – Gemeinsame Übung A_05_01_01

WBS TRAINING

Aufgabe_05_01_01

Gegeben sei der folgende PAP:



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

WBS TRAINING

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!