Zeiger II

Lernziele

- Sie können ein Array als Parameter definieren und einer Funktion übergeben.
- Sie können via Zeiger die einzelnen Bytes einer Integer-Variablen ausgeben.

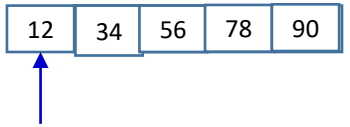
Array und Zeiger

- Der Name eines Arrays (ohne Angaben des Index) ist ein konstanter Zeiger auf den Anfang des Arrays.

```
int list[10] = { 12, 34, 56, 78, 90 };

// iptr zeigt auf das erste Element von list:
int *iptr = list;  //äquivalent: iptr=&list[0]

iptr++;             //iptr++ zeigt auf das nächste Element
*iptr = 99;         //äquivalent:list[1]=99;
*(iptr++) = 66;     //äquivalent:list[2]=66;
```



Array_Zeiger2.cpp

Zeichenkette und Zeiger

Eine Zeichenkette kann folgenderweise definiert und initialisiert werden:

```
char text[] = "ein Versuch ";
```

Mit dieser Definition wird der nötigen Speicherbereich reserviert und die Zeichenkette dort hinein kopiert. Der Name *text* ist ein *konstanter Zeiger*, welcher die Angangsadresse dieses Speicherbereichs speichert.

Falsch: `text = "ein Test ";`
Richtig: `strcpy_s(text, "ein Test");`

Zeichenkette_Zeiger.cpp

Arrays und Funktionen

- Arrays können an Funktionen übergeben werden, aber nicht von Funktionen zurückgeliefert werden.
- Arrays werden immer By Reference übergeben, d.h. es wird nur die Anfangsadresse als Zeiger übergeben.

```
int Sum(int list[], int anz) {
        int sum = 0;
        for (int i = 0; i < anz; i++) {
                sum += list[i];
        }
       return sum;
}
```

```
int main()
{
       int list[] = { 1,2,3,4,5,6,7,8,9 };
       printf("sum=%d\n", Sum(list, 9));
}
```

ArrayPara.cpp

Arrays und Funktionen

Ein Array kann mit folgenden drei Schreibweisen als Parameter einer Funktion deklariert werden. Dabei wird unabhängig von der verwendeten Schreibweise des Parameters seine Anfangsadresse als Zeiger übergeben.

1) int Sum(int list[], int anz);

Die Grössenangabe ist optional, und wird auch nicht überprüft. Die effektive Grösse des Arrays wird als weiteren Parameter übergeben.

2) int Sum(int list[10], int anz);

*3) int Sum(int *list, int anz);*

Aufruf:

```
int a[] = {1,2,3,4,5};
int sum = Sum(a, 5);
```

ArrayPara.cpp

Zeigerarithmetik

Mit einem Zeiger sind folgende Operationen erlaubt:

- **Addition** mit einem Int-Wert : +,++

  Achtung: Die Zeiger werden um die Grösse des Datentyps auf den sie zeigen, inkrementiert.

```
Bsp 1:
    int i = 1;
    int *pt = &i;       :  0x8049dd0
    pt++;               :  0x8049dd4
        =>+ 4 Bytes :   neue Adresse = alte Adresse + Anzahl Bytes des Datentyps
Bsp 2:
    int list[MAX] = { 12, 34, 56, 78, 90 };
    int *iptr = list;  //iptr zeigt auf das erste Element.
    iptr++;            //Nun zeigt iptr auf das zweite Element.
```

- Subtraktion von zwei Zeigern-,--

  => Die Grösse des Adressbereiches

- Vergleich zweier Zeiger mit <, >, >=,<=,==,!=

ZeigerInkrement.cpp

Zuweisung von Zeiger

Zeiger können einander zugewiesen werden:

1) Wenn sie vom selben Typ sind, also auf denselben Typ zeigen:

```
int zahl = 1234;
int *pch = &zahl;
```

2) Wenn sie nicht vom selben Typ sind, ist ein explizites Typecast notwendig:

```
int zahl = 0xFFFEDCBA;  //hexadezimalzahle Darstellung
unsigned char* pch;

pch = (unsigned char*)&zahl; //pch speichert die Adresse des niedrigsten Bytes von zahl

printf("Das erste Byte: %X\n", *pch);   //Ausgabe: BA
```

Generischer Zeiger

- Defintition

  Ein Generischer Zeiger kann auf irgendeinen Typ von Variablen zeigen:
  void *pt;

- Zuweisung

```
int *ipt;
void *pt;

pt = ipt;                 //OK
ipt = (int *) pt;    //Typecast
```

Zeiger auf Zeiger

- Zeiger, der auf eine andere Zeigervariable zeigt:

Datentyp **Zeigervariable

- Beispiel:

```
int number = 0xFFFEDCBA;   //hexadezimalzahl
Int *ptr = &number;
Int **pptr = &ptr;

printf("number = %X; *ptr = %X; **ptr = %X", number, *ptr, **pptr);
```

ZeigerAufzeiger.cpp