Umgang mit Pointern

Seminar Effiziente Programmierung

Alexander Lambertz

Informatik
Universität Hamburg

08. November 2012

Gliederung

Was ist ein Pointer?

Definition

A pointer is a value that designates the address (i.e., the location in memory), of some value.

Quelle: `http://en.wikibooks.org/wiki/C_Programming/Pointers_and_arrays`

Der & und der $*$ Operator

Erläuterung des & Operators

Der & Operator gibt die Speicheradresse einer Variable aus.

Erläuterung des $*$ Operators

Der $*$ Operator gibt den Inhalt einer Speicheradresse aus. (Dereferenzierung)

Wo zeigt ein Pointer hin?

Ein Pointer zeigt immer auf die erste Speicheradresse einer Variable.

Wie wird ein Pointer deklariert?

Deklaration

```
1 // e.g. some Integer Pointer
2 int *intPtr;
3 int **ptrToIntPtr;
4 // or some Character Pointer
5 char *charPtr;
6 char **ptrToCharPtr;
```

Beispiel zur Benutzung der Operatoren

Beispiel

```
1  int main()
2  {
3    int integer = 1;
4    int *ptr = &integer;
5
6    printf("integer = %d \n", integer);
7    printf("ptr = %u \n", ptr);
8    printf("*ptr = %d \n", *ptr);
9
10   *ptr = 2;
11
12   printf("integer = %d \n", integer);
13   printf("*ptr = %d \n", *ptr);
14 }
```

Beispiel zur Benutzung der Operatoren

Ausgabe

```
integer = 1
ptr = 1462442760
*ptr = 1
integer = 2
*ptr = 2
```

Wie werden Strings in C abgebildet?

Wie wird ein String in C abgebildet?

Ein String wird in C als ein Array von Charactern abgebildet.

Wie wird ein String in C abgebildet?

Beispiel

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

int main()
{
  char string[]  = "Effiziente Programmierung";

  printf("Gesamter String: %s\n",string);

  printf("strlen(string) = %d\n",strlen(string));

  printf("string[0] = %c\n", string[0]);
  printf("string[24] = %c\n", string[24]);

  printf("sizeof(string) = %u\n", sizeof(string));
  printf("sizeof(char) = %u\n", sizeof(char));
}
```

Wie werden Strings in C abgebildet?

Ausgabe

```
1 Gesamter String: Effiziente Programmierung
2 strlen(string) = 25
3 string[0] = E
4 string[24] = g
5 sizeof(string) = 26
6 sizeof(char) = 1
```

Darstellung des Arrays

E	f	f	...	n	g	\0
0	1	2	...	23	24	25

Pointer auf einen Array

Beispiel: Parameter der Main-Funktion ausgeben

```
1  int main(int argc, char **argv)
2  {
3    char **curArgv;
4
5    printf("argc = %d\n", argc);
6
7    curArgv = argv;
8
9    // Return all Parameters
10   for (int n=0; n<argc; n++)
11   {
12     printf("Adress of argv[%d] = %u\n", n, &(*curArgv));
13     // &(*curArgv) = curArgv
14     printf("argv[%d] = %s\n", n, *curArgv);
15     curArgv++; // Moving the Pointer
16   }
17 }
```

Pointer auf einen Array

Der Aufruf

```
./test abc cde
```

Ausgabe

```
argc = 3
Adress of argv[0] = 1510333208
argv[0] = ./test
Adress of argv[1] = 1510333216
argv[1] = abc
Adress of argv[2] = 1510333224
argv[2] = cde
```

Pointer auf einen Array

Ausgabe

```
...
Adress of argv[0] = 1510333208
...
Adress of argv[1] = 1510333216
...
Adress of argv[2] = 1510333224
...
```

Der Code

```
int main(int argc, char **argv)
    ...
    printf("Adress of argv[%d] = %u\n", n, &(*curArgv));
    ...
    argv++;
    ...
```

Speicherbedarf

Definition

sizeof() liefert die Anzahl der Bytes, die für eine Variable oder einen Datentyp reserviert werden.

Quelle: `http://home.fhtw-berlin.de/~junghans/cref/SYNTAX/sizeof.html`

Was ist die Ausgabe? Warum?

```
1 printf("sizeof(char*) = %d", sizeof(char*));
```

Speicherbedarf

Sizeof Ausgaben

```
1 sizeof(int) = 4
2 sizeof(char) = 1
3 sizeof(int*) = 8
4 sizeof(char*) = 8
5 sizeof(void*) = 8 // Genauer im Abschnitt Typsicherung
```

Warum 8 Byte?

Auf 32 Bit-System sind es 4 Byte und auf einem 16 Bit-System 2 Byte.

Pointer auf einen Array (2)

Beispiel: Argumente der Main-Funktion ausgeben (2)

```
1  int main(int argc, char **argv)
2  {
3     printf("argc = %d\n", argc);
4     printf("Adress of argv = %u\n",(int) &argv);
5     for (int n=0; n<argc; n++)
6     {
7        printf("Adress of argv[%d] = %u\n", n, (int)  &(*(argv+n
             )));
8        printf("Adress of argv[%d][0] = %u\n", n, (int)  &(*(*(
             argv+n))));
9        printf("Adress of argv[%d][1] = %u\n", n, (int)  &(*(*(
             argv+n)+1)));
10       printf("argv[%d] = %s\n", n, *(argv+n));
11       printf("argv[%d][0] = %c\n", n, *(*(argv+n)));
12       printf("argv[%d][1] = %c\n", n, *(*(argv+n)+1));
13    }
14 }
```

Pointer auf einen Array (2)

Ausgabe

```
1  argc = 3
2  Adress of argv = 1503812320
3  Adress of argv[0] = 1503812376
4  Adress of argv[0][0] = 1503812640
5  Adress of argv[0][1] = 1503812641
6  argv[0] = ./test
7  argv[0][0] = .
8  argv[0][1] = /
9  Adress of argv[1] = 1503812384
10 Adress of argv[1][0] = 1503812647
11 Adress of argv[1][1] = 1503812648
12 argv[1] = abc
13 argv[1][0] = a
14 argv[1][1] = b
15 ...
```

Pointer auf einen Array (2)

Visualisierung des Speichers

Speicheradresse	Inhalt
1503812320	1503812376

1503812376	1503812384	1503812392
1503812640	1503812647	1503812651

1503812640	1503812641	1503812642	1503812643
.	/	t	e

Call-by-Value

Beispiel: Call-by-Value

```
1  void callByValue(int valueFunction)
2  {
3    printf("(Function) valueFunction = %d\n", valueFunction);
4    valueFunction = 2;
5    printf("(Function) valueFunction = %d\n", valueFunction);
6  }
7  int main()
8  {
9    int valueMain = 1;
10
11   printf("(MainFunction) valueMain = %d\n", valueMain);
12   callByValue(valueMain);
13   printf("(MainFunction) valueMain = %d\n", valueMain);
14 }
```

Call-by-Value

Ausgabe

```
1 (MainFunction) valueMain = 1
2 (Function) valueFunction = 1
3 (Function) valueFunction = 2
4 (MainFunction) valueMain = 1
```

Call-by-Reference

Beispiel: Call-by-Reference

```
1  void callByReference(int *valueFunction)
2  {
3    printf("(Function) valueFunction = %d\n", *valueFunction);
4    *valueFunction = 2;
5    printf("(Function) valueFunction = %d\n", *valueFunction);
6  }
7  int main()
8  {
9    int valueMain = 1;
10
11   printf("(MainFunction) valueMain = %d\n", valueMain);
12   callByReference(&valueMain);
13   printf("(MainFunction) valueMain = %d\n", valueMain);
14 }
```

Call-by-Reference

Ausgabe

```
1 (MainFunction) valueMain = 1
2 (Function) valueFunction = 1
3 (Function) valueFunction = 2
4 (MainFunction) valueMain = 2 // !!
```

Vergleich Call-by-Value / Call-by-Reference

Beispiel: Call-by-Value

```
1 void callByValue(int valueFunction)
2 {
3   printf("(Function) valueFunction = %d\n", valueFunction);
4   valueFunction = 2;
5   ...
6   callByValue(valueMain);
7   ...
```

Beispiel: Call-by-Reference

```
1 void callByReference(int *valueFunction)
2 {
3   printf("(Function) valueFunction = %d\n", *valueFunction);
4   *valueFunction = 2;
5   ...
6   callByReference(&valueMain);
7   ...
```

Struct

Wie wird ein Struct deklariert?

```
1  struct person {
2    int age;
3    float weight;
4    char name[25];
5  }
6
7  struct person markus, *paul;
```

Operatoren

```
1  markus.age // Struct
2  - oder -
3  paul->age // Pointer to a Struct
```

Struct

Beispiel

```
1  struct person {
2    int age;
3    float weight;
4    char name[25];
5  };
6
7  int main()
8  {
9    struct person markus, *paul;
10   paul = &markus; // Potential Aliaserror?!
11
12   markus.age = 43;
13   markus.weight = 80.5;
14   strcpy(markus.name, "Markus Paul Anders");
15
16   printf("markus.age = %d\n",markus.age);
17   printf("paul->age = %d\n",paul->age);
18   printf("(*paul).age = %d\n",(*paul).age);
19 }
```

Struct

Beispiel

```
1  ...
2  printf("markus.age = %d\n",markus.age);
3  printf("paul->age = %d\n",paul->age);
4  printf("(*paul).age = %d\n",(*paul).age);
5  ...
```

Ausgabe

```
1  markus.age = 43
2  paul->age = 43
3  (*paul).age = 43
```

Typsicherheit in C

Beispiel

```
1  int main(void) {
2    int a = 10;
3
4    int *ptr1;
5    float *ptr2;
6
7    ptr1 = &a;
8    ptr2 = (float*)  &a;
9
10   printf("ptr1 = %d\n", ptr1);
11   printf("ptr2 = %d\n", ptr2);
12
13   printf("*ptr1 = %d\n", *ptr1);
14   printf("*ptr2 = %d\n", *ptr2);
15 }
```

Typsicherheit in C

Ausgabe des Programms

```
1 ptr1 = 1583393524
2 ptr2 = 1583393524
3 *ptr1 = 10
4 *ptr2 = 1583391240 // !!!!
```

Zusammenfassung

C und C++ sind NICHT typsicher.

void-Pointer

Was ist ein void-Pointer?

Void pointers are pointers pointing to some data of no specific type.

Quelle: `http://www.antoarts.com/void-pointers-in-c/`

Dereferenzieren eines void-Pointers

Um einen void-Pointer zu dereferenzieren, muss dieser auf einen spezifischen Typ gecastet werden.

void-Pointer

Beispiel

```
1  #define TYPE_INT 0
2  #define TYPE_FLOAT 1
3  void doubleVal(int type, void *var){
4    if(type==TYPE_INT){
5      *(int*)var*=2;
6    } else if(type==TYPE_FLOAT){
7      *(float*)var*=2;
8    }
9  }
10 ...
11 struct ListNode{
12   struct ListNode *next;
13   void *data;
14 };
15 ...
16 doubleVal(TYPE_INT, &integer);
17 doubleVal(TYPE_FLOAT, &floatingPoint);
```

Quelle: http://www.antoarts.com/void-pointers-in-c/

void-Pointer

Typische Verwendung von void-Pointern

Typischer Weise werden void-Pointer in Funktionen wie memcmp genutzt. Hier ist der konkrete Typ des Pointers irrelevant.

NULL-Pointer Exception

Beispiel

```
1 int main(void) {
2   int *mainPointer=NULL;
3   printf("Content of mainPointer = %d", *mainPointer);
4 }
```

Ausgabe

```
1 Segmentation fault: 11 // Fehler da NULL-Pointer
```

Dangling Pointer

Was ist ein Dangling Pointer?

Dangling pointers and wild pointers in computer programming are pointers that do not point to a valid object of the appropriate type. These are special cases of memory safety violations.

Quelle: `http://en.wikipedia.org/wiki/Dangling_pointer`

Dangling Pointer

Beispiel

```
1  void secureFree(void **functionPointer)
2  {
3    if ( functionPointer != NULL && *functionPointer != NULL)
4    {
5      free (*functionPointer);
6      *functionPointer = NULL;
7      printf("Freed something\n");
8    }
9  }
10 int main(void) {
11   int *integer, *mainPointer;
12   integer = (int*) malloc(sizeof(int));
13   *integer = 5;
14   mainPointer = integer;
15   secureFree((void*) &integer);
16   printf("&mainPointer = %d\n",(int) *mainPointer); // !!
         Undefined!!
17 }
```

Hinweise zu den Folien

1. Bei den Quelltexten müssen folgende Header eingebunden werden:
   1. #include <stdio.h>
   2. #include <stdlib.h>
   3. #include <string.h>

Quellen I


Yashavant Kanetkas
Understanding Pointer in C.
BPB Publications, 2009.


Galileo Computing:: C von A bis Z - 12.11 void-Zeiger
http://openbook.galileocomputing.de/c_von_a_bis_z/012_c_zeiger_011.htm
Erster Zugriff: 04.11.2012


C Referenz -sizeof Operator-
http://home.fhtw-berlin.de/~junghans/cref/SYNTAX/sizeof.html
Erster Zugriff: 04.11.2012


WikiBook: Pointers in C (en)
http://en.wikibooks.org/wiki/C_Programming/Pointers_and_arrays
Erster Zugriff: 29.10.2012

Quellen II

Wikipedia: Operators in C and C++ (en)
http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B
Erster Zugriff: 29.10.2012

StackOverflow: Is the sizeof(some pointer) always equal to four?
http://stackoverflow.com/questions/399003/is-the-sizeofsome-pointer-always-equal-to-four
Erster Zugriff: 29.10.2012

Void pointers in C - AntoArts
http://www.antoarts.com/void-pointers-in-c/
Erster Zugriff: 30.10.2012