Technische Informatik III
Wintersemester 2008/2009

Uwe Kuehn

C-Tutorium

4. Dezember 2008

1 1. Woche

1.1 Pointer & Co.

Das Listing 1 veranschaulicht, wann welcher Wert verändert wird, abhängig z.B. davon ob es sich um einen Pointer handelt oder nicht.

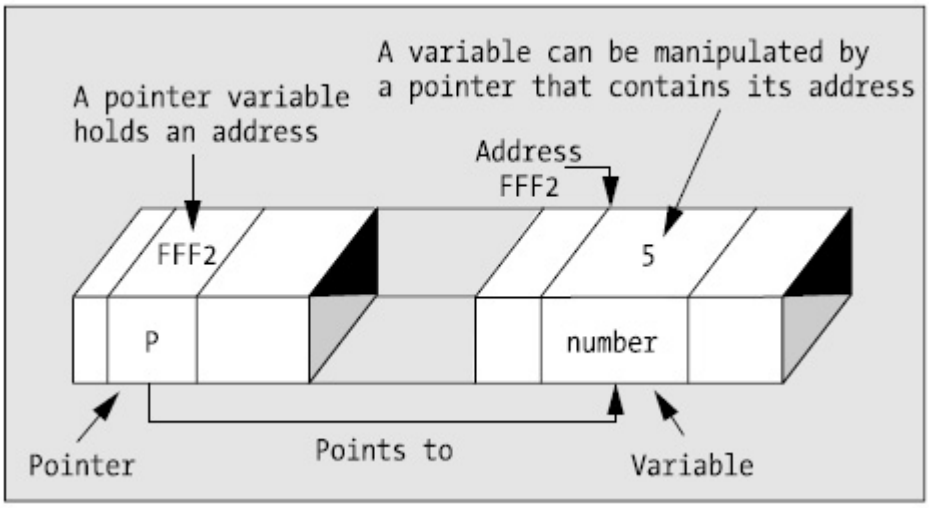
Listing 1: Uebung Woche 1

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

void printAdress(int * ptr){
        printf("printAdress:Adresse␣:%x\n", ptr);
        printf("printAdress:Wert␣vor++␣:%u\n", *ptr);
        (*ptr)++;
        printf("printAdress:Wert␣nach++␣:%u\n", *ptr);
}

void printInt(int i){
        printf("printInt:Wert␣vor++␣:%u\n", i);
        printAdress(&i);
        i++;
        printf("printInt:Wert␣nach++:%u\n", i);
}

void testIncr(int * ptr){
        printf("testIncr:Wert␣vor++␣:%u\n", ptr);
        ptr ++;
        printf("testIncr:Wert␣nach++␣:%u\n", ptr);
}


void testIncr2(int ** ptr){
        int * p = *ptr;
        printf("testIncr2:Wert␣vor++␣:%u\n", p[0]);
        p[0] ++;
        printf("testIncr2:Wert␣nach++␣:%u\n", p[0]);
}

int main(int argc, char *argv[]){
        int myInt = 5;
        int * pMyInt = &myInt;
        printf("main:Wert␣vor␣PrintAdress␣:%u\n", myInt);
        printAdress(&myInt);
        printf("main:Wert␣vor␣PrintAdress␣:%u\n", myInt);
        printInt(myInt);
        printf("main:Wert␣nach␣PrintAdress␣:%u\n", myInt);
        testIncr(&myInt);
        printf("main:Wert␣nach␣testIncr␣:%u\n", *pMyInt);
        testIncr2(&pMyInt);
        printf("main:Wert␣nach␣testIncr2(int)␣:%u\n", myInt);
        printf("main:Wert␣nach␣testIncr2(pInt)␣:%u\n", *pMyInt);
}
```

Listing 2: Ausgabe

```
kuehn@irkutsk:~/RAID/M_Sem_02/c$ ls
adressen.c  arith.c    -f      first.c  node  runcode  u6  zusatz.c
a.out       aufgabe.c  first   main.c   proc  u5       u7
kuehn@irkutsk:~/RAID/M_Sem_02/c$ ./runcode
main:Wert vor PrintAdress :5
printAdress:Adresse :bf90caf0
printAdress:Wert vor++ :5
printAdress:Wert nach++ :6
main:Wert vor PrintAdress :6
printInt:Wert vor++ :6
printAdress:Adresse :bf90cad0
printAdress:Wert vor++ :6
printAdress:Wert nach++ :7
printInt:Wert nach++:8
main:Wert nach PrintAdress :6
testIncr:Wert vor++ :3213937392
testIncr:Wert nach++ :3213937396
main:Wert nach testIncr :6
testIncr2:Wert vor++ :6
testIncr2:Wert nach++ :7
main:Wert nach testIncr2(int) :7
main:Wert nach testIncr2(pInt) :7
kuehn@irkutsk:~/RAID/M_Sem_02/c$
```

1.2 Pointer



[1]

1.3 Struct

Auch wenn prinzipiell die Befehle **struct bn * myNode;** in der Semantik dasselbe meint wie **binaryNode* myNode;**, vorausgesetzt eine Definition wie in Listing 3 existiert, so können wir allein in C das struct nur auf die Form wie hier abgebildet definieren. Der Austausch innerhalb der struct Definition zu Gunsten des definierten Datentypes ist erst in C++ möglich.

[1] Abbildung aus „Beginning C From Novice to Professional“ 4th Edition von Ivor Horton

Listing 3: binaryNode.h

```
//
// struct bn
// - hold an integer value
// - additionally it contains 3 pointers to another nodes
//
typedef struct bn{
        int value;
        struct bn * root;
        struct bn * left;
        struct bn * right;
}binaryNode;

//
// function creates a single node
// without references toanother nodes
// the return value may be the root
//
binaryNode* createNode(int);
```

Listing 4: binaryNode.c

```
#include <stdio.h>
#include "binaryNode.h"

binaryNode* createNode(int value){
        binaryNode* retWert;
        // allocating memory
        retWert = (binaryNode*)malloc(sizeof(binaryNode));
        retWert->value = value;
        retWert->root = retWert->right = retWert->left = NULL;
        return retWert;
}
```

Listing 5: main.c

```
#include <stdlib.h>
#include <stdio.h>
#include "binaryNode.h"

//
// standard start adress of program
//
int main(int argc, char*argv[]){

        return EXIT_SUCCESS;
}
```

Listing 6: Ausgabe

```
kuehn@irkutsk:~/RAID/M_Sem_02/c$ ./runcode
main:Wert vor PrintAdress :5
printAdress:Adresse :bf90caf0
printAdress:Wert vor++ :5
printAdress:Wert nach++ :6
main:Wert vor PrintAdress :6
printInt:Wert vor++ :6
printAdress:Adresse :bf90cad0
printAdress:Wert vor++ :6
printAdress:Wert nach++ :7
printInt:Wert nach++:8
main:Wert nach PrintAdress :6
testIncr:Wert vor++ :3213937392
testIncr:Wert nach++ :3213937396
```

```
main:Wert nach testIncr :6
testIncr2:Wert vor++ :6
testIncr2:Wert nach++ :7
main:Wert nach testIncr2(int) :7
main:Wert nach testIncr2(plnt) :7
kuehn@irkutsk:~/RAID/M_Sem_02/c$
```

2 2. Woche

2.1 Debuggen

Um selbst erstellten Code unter Unix/Linux zu debuggen, gibt es hier auch Werkzeuge, die einem das Leben erleichtern. An dieser Stelle wird der GNU Debugger vorstellt.

Listing 7: buggyCode.c

```
#include <stdlib.h>
//
// standard start adress of program
//
int main(int argc, char*argv[]){
        int * pMyInt;
        *pMyInt = 5;
        return EXIT_SUCCESS;
}
```

Listing 8: Ausgabe

```
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ ls -ali
insgesamt 32
30761026 drwxr-xr-x 2 kuehn others 4096 2008-11-28 08:20 .
30761008 drwxr-xr-x 4 kuehn others 4096 2008-11-28 07:12 ..
30761027 -rw-r--r-- 1 kuehn others  311 2008-11-28 07:16 binaryNode.c
30761030 -rw-r--r-- 1 kuehn others  348 2008-11-28 07:16 binaryNode.h
30761033 -rw-r--r-- 1 kuehn others  159 2008-11-28 08:17 buggyCode.c
30761031 -rw-r--r-- 1 kuehn others  134 2008-11-28 07:16 main.c
30761032 -rwxr-xr-x 1 kuehn others 7182 2008-11-28 07:16 proc
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ gcc -o procNoDebug buggyCode.c
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ gcc -g buggyCode.c

kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ ls -ali
insgesamt 32
30761026 drwxr-xr-x 2 kuehn others 4096 2008-11-28 08:20 .
30761008 drwxr-xr-x 4 kuehn others 4096 2008-11-28 07:12 ..
30761027 -rw-r--r-- 1 kuehn others  311 2008-11-28 07:16 binaryNode.c
30761030 -rw-r--r-- 1 kuehn others  348 2008-11-28 07:16 binaryNode.h
30761033 -rw-r--r-- 1 kuehn others  159 2008-11-28 08:17 buggyCode.c
30761031 -rw-r--r-- 1 kuehn others  134 2008-11-28 07:16 main.c
30761032 -rwxr-xr-x 1 kuehn others 7182 2008-11-28 07:16 proc
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ gcc -o procNoDebug buggyCode.c
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ gcc -g buggyCode.c
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ ls -ali
insgesamt 48
30761026 drwxr-xr-x 2 kuehn others 4096 2008-11-28 08:21 .
30761008 drwxr-xr-x 4 kuehn others 4096 2008-11-28 07:12 ..
30761035 -rwxr-xr-x 1 kuehn others 7821 2008-11-28 08:21 a.out
.........
30761034 -rwxr-xr-x 1 kuehn others 6929 2008-11-28 08:20 procNoDebug

kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ ./a.out
Speicherzugriffsfehler

kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ ./procNoDebug
Speicherzugriffsfehler

kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ gdb ./procNoDebug
GNU gdb 6.4.90-debian
Copyright (C) 2006 Free Software Foundation, Inc.
```

```
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...Using host libthread_db library "/lib/tls/i686/cmov/libth

(gdb) run
Starting program: /home/bude2/kuehn/M_Sem_02/c/node/procNoDebug

Program received signal SIGSEGV, Segmentation fault.
0x08048338 in main ()
(gdb) quit
The program is running.  Exit anyway? (y or n) y

kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ gdb ./a.out
GNU gdb 6.4.90-debian
Copyright (C) 2006 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i486-linux-gnu"...Using host libthread_db library "/lib/tls/i686/cmov/libth

(gdb) run
Starting program: /home/bude2/kuehn/M_Sem_02/c/node/a.out

Program received signal SIGSEGV, Segmentation fault.
0x08048338 in main () at buggyCode.c:7
7               *pMyInt = 5;
(gdb) trace 7
Tracepoint 1 at 0x8048335: file buggyCode.c, line 7.
(gdb) quit
The program is running.  Exit anyway? (y or n) y
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$
```

2.2 insertNode

Wir erweitern unser Beispiel Listing 3 bis 5. In diesem Abschnitt wollen wir einem bestehenden Knoten einen zusätzlich Knoten anhängen.
Wir benutzen folgende Definition für den Binärbaum :

- alle Elemente (Wurzel, Knoten und Blätter) des Baumes tragen eine Information
- jedes Element „kennt“ seinen Elter und die Kinder
- jedes Element hat mindestens 0 und maximal 2 Kinder
- die Wurzel hat keinen Elter

Folgendes Verfahren(Algorithmus) wird beim Knoten einfügen gefordert :

1. gegeben :
   - Pointer auf aktuelles Element, im ersten Schritt auf der Wurzel
   - einzufügender mit createNode(int) Knoten
2. prüfe ob einzufügender $\leq$ aktueller Knotenwert
3. falls Ja $\rightarrow$ 5
4. falls Nein $\rightarrow$ 6
5. - wenn vorhanden, gehe zum linken Knoten als neues aktuelles Element, weiter mit 1
   - sonst, prüfe ob einzufügender $=$ aktueller Knotenwert
   - falls Ja, fertig
   - sonst füge Knoten als rechtes Element ein, fertig
6. - wenn vorhanden, gehe zum rechten Knoten als neues aktuelles Element, weiter mit 1
   - sonst füge Knoten als rechtes Element ein, fertig.

Listing 9: Erweiterung binaryNode.h

```
//
// function counts all elements
//
void insertNode(binaryNode*, binaryNode*);
```

2.3 countElements

Diese Funktion zählt alle Elemente des Baumes(bzw. Teilbaumes) und gibt den Wert zurück.

Listing 10: Erweiterung binaryNode.h

```
//
// function counts all elements
//
int countElements(binaryNode*);
```

2.4 countMaxHeight

Listing 11: Erweiterung binaryNode.h

```
//
// function counts the height of the tree
//
int countHeight(binaryNode*);
```

2.5 Solution

Listing 12: main.c

```
int main(int argc, char*argv[]){
        binaryNode* tree = createNode(10);
        binaryNode* node = NULL;
        printNode(node);
        printNode(tree);

        node = createNode(8);
        insertNode(tree, node);
        printNode(tree);

        node = createNode(14);
        insertNode(tree, node);
        printNode(tree);

        node = createNode(13);
        insertNode(tree, node);
        printNode(tree);

        node = createNode(9);
        insertNode(tree, node);
        printNode(tree);

        node = createNode(9);
        insertNode(tree, node);
        printNode(tree);

        node = createNode(7);
        insertNode(tree, node);
        printNode(tree);


        deleteNode(&tree);
        printNode(tree);

        return EXIT_SUCCESS;
}
```

Listing 13: binaryNode.c

```
int max(int a, int b){
        return a<b?b:a;
}

void insertNode(binaryNode* tree, binaryNode* node){
        if(tree && node){
                if(node->value <= tree->value){
                        if(node->value != tree->value){
                        if(tree->left == NULL){
                                        tree->left = node;
                                        node->root = tree;
                        }
                        else{
                                insertNode(tree->left, node);
                        }
                        }
                }
                else{
                        if(tree->right == NULL){
                                tree->right = node;
                                node->root = tree;
```

```
                        }
                        else{
                                insertNode(tree->right, node);
                        }
                }
        }
}

void printNode(binaryNode* tree){
        if(tree != NULL){
                if(tree->root == NULL)
                        printf("************************\n");

                printf("This is an unempty tree, with %i elements. ", countElements(tree));
                printf("The height is %i.\n", countHeight(tree));
                printf("Node value:%i (", tree->value);
                if(tree->left != NULL)
                        printf("%i, ", tree->left->value);
                else
                        printf("e, ");

                if(tree->right != NULL)
                        printf("%i", tree->right->value);
                else
                        printf("e");

                printf(")\n");

                if(tree->left != NULL)
                        printNode(tree->left);

                if(tree->right != NULL)
                        printNode(tree->right);

                if(tree->root == NULL)
                        printf("************************\n\n");
        }
        else{
                printf("This is an empty tree.\n");
        }
}

int countElements(binaryNode* tree){
        int count = 0;
        count++;

        if(tree->left)
                count += countElements(tree->left);

        if(tree->right)
                count += countElements(tree->right);

        return count;
}

int countHeight(binaryNode* tree){
        int count = 1, left = 0, right = 0;

        if(tree->left)
                left = countElements(tree->left);

        if(tree->right)
                right = countElements(tree->right);

        return max(right, left) + count;
```

```
}

void deleteNode(binaryNode** tree){
        binaryNode* tri = *tree;
        tri->root = NULL;

        if(tri->left){
                deleteNode(&(tri->left));
                tri->left = NULL;
        }

        if(tri->right){
                deleteNode(&(tri->right));
                tri->right = NULL;
        }

        free(tri);
        *tree = NULL;
}
```

Listing 14: Ausgabe

```
cd /home/bude/kuehn/RAID/M_Sem_02/c/node
kuehn@irkutsk:~/RAID/M_Sem_02/c/node$ ./node
This is an empty tree.
************************
This is an unempty tree, with 1 elements. The height is 1.
Node value :10 (e, e)
************************

************************
This is an unempty tree, with 2 elements. The height is 2.
Node value :10 (8, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :8 (e, e)
************************

************************
This is an unempty tree, with 3 elements. The height is 2.
Node value :10 (8, 14)
This is an unempty tree, with 1 elements. The height is 1.
Node value :8 (e, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :14 (e, e)
************************

************************
This is an unempty tree, with 4 elements. The height is 3.
Node value :10 (8, 14)
This is an unempty tree, with 1 elements. The height is 1.
Node value :8 (e, e)
This is an unempty tree, with 2 elements. The height is 2.
Node value :14 (13, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :13 (e, e)
************************

************************
This is an unempty tree, with 5 elements. The height is 3.
Node value :10 (8, 14)
This is an unempty tree, with 2 elements. The height is 2.
Node value :8 (e, 9)
This is an unempty tree, with 1 elements. The height is 1.
Node value :9 (e, e)
This is an unempty tree, with 2 elements. The height is 2.
```

```
Node value :14 (13, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :13 (e, e)
**********

**********
This is an unempty tree, with 5 elements. The height is 3.
Node value :10 (8, 14)
This is an unempty tree, with 2 elements. The height is 2.
Node value :8 (e, 9)
This is an unempty tree, with 1 elements. The height is 1.
Node value :9 (e, e)
This is an unempty tree, with 2 elements. The height is 2.
Node value :14 (13, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :13 (e, e)
**********

**********
This is an unempty tree, with 6 elements. The height is 4.
Node value :10 (8, 14)
This is an unempty tree, with 3 elements. The height is 2.
Node value :8 (7, 9)
This is an unempty tree, with 1 elements. The height is 1.
Node value :7 (e, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :9 (e, e)
This is an unempty tree, with 2 elements. The height is 2.
Node value :14 (13, e)
This is an unempty tree, with 1 elements. The height is 1.
Node value :13 (e, e)
**********

This is an empty tree.
```