Aufgabenblatt 12: Dynamische Datenstrukturen (2)

Florian Ludewig (Übungsgruppe 2)

24. Januar 2020

Aufgabe 1 – Doppelt verkettete Listen

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

struct dnode *head, *last;

struct dnode{
  int data;
  struct dnode *next, *prev;
};

struct dnode *mkNode(int val){
  struct dnode *node = NULL;
  if((node = malloc(sizeof (struct dnode))) != NULL) {
    node -> data = val;
    node -> next = node -> prev = NULL;
    return node;
  }
  else return NULL;
}

void printList(void) {
  if (head == NULL) {
    printf("( )");
    return;
  }
  printf("( ");
  struct dnode *tmp = head;
  while(tmp != NULL){
  printf("%d ", tmp -> data);
    tmp = tmp -> next;
  }
  printf(")\n");
}

struct dnode *insert_start(int val) {
  struct dnode *new_node = mkNode(val);
  if (head == NULL) {
    new_node -> next = NULL;
    head = new_node;
    last = head;
  } else {
    new_node -> next = head;
    head -> prev = new_node;
    head = new_node;
  }
  new_node -> prev = NULL;
  return new_node;
}

void remove_element(int val) {
```

```c
  if (head -> next == NULL) return;
  struct dnode *deleted;
  if (head -> data == val) {
    deleted = head;
    head = head -> next;
    head -> prev = NULL;
  } else {
    struct dnode *temp = head;
    while(temp -> data != val && temp -> next != NULL) {
      temp = temp -> next;
    }
    if (temp == NULL) return;
    deleted = temp;
    (temp -> prev) -> next = temp -> next;
    if (temp -> next != NULL) {
      temp -> next -> prev = temp -> prev;
    }
  }
  if (deleted) free(deleted);
}

int main(void) {
  int remove, primes[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
      43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97 };
  for (int i = sizeof(primes) / sizeof(int) - 1; i >= 0; i--) {
    insert_start(primes[i]);
  }
  printf("Zahl eingeben: ");
  scanf("%d", &remove);
  remove_element(remove);
  printList();
  return 1;
}
```

Aufgabe 2 – Binärbäume

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

struct node {
  struct node *left;
  struct node *right;
  int data;
};

struct node *mkNode(int d, struct node *l, struct node *r) {
  struct node *n = NULL;
  if((n = malloc(sizeof (struct node))) != NULL){
    n->data = d; n->left = l; n->right = r; return n;}
    else { return NULL;}
  }

struct node *mkLeaf(int d) {
  return mkNode(d, NULL, NULL);
}

void print_inorder(struct node *n) {
  if(n == NULL) return;
  print_inorder(n->left);
  printf("%c\n", n->data);
  print_inorder(n->right);
}

int count_occurrences(struct node *n, char c) {
  if (n == NULL) return 0;
  int match = (n -> data == c);
  int left_occurrences = count_occurrences(n -> left, c);
  int right_occurrences = count_occurrences(n -> right, c);
  return match + left_occurrences + right_occurrences;
}

int main(void) {
  struct node *tree = mkNode(' ', mkNode('n', mkNode(' ', mkNode('e', mkLeaf
      ('S'), mkNode('i', NULL, mkLeaf('n'))), mkNode('e', mkNode('o', NULL,
      mkLeaf('d')), mkNode('r', NULL, mkLeaf(' ')))), mkNode(' ', mkNode('c'
      , mkLeaf('i'), mkNode('h', NULL, mkLeaf('t'))), mkNode('i', mkNode('s'
      , NULL, mkLeaf('e')), mkNode('n', NULL, mkLeaf(','))))), mkNode('e',
      mkNode('i', mkNode('a', mkLeaf('d'), mkNode('s', NULL, mkLeaf(' '))),
      mkNode(' ', mkNode('s', NULL, mkLeaf('t')), mkNode('h', NULL, mkLeaf('
      i')))), mkNode(' ', mkNode('d', mkNode('r', NULL, mkLeaf(' ')), mkNode
      ('i', NULL, mkLeaf('e'))), mkNode('a', mkNode('F', NULL, mkLeaf('r')),
       mkNode('g', NULL, mkLeaf('e'))))));

  char input;
  printf("Zeichen eingeben: ");
  scanf("%c", &input);
  printf("Das Zeichen '%c' kommt %d mal vor\n", input, count_occurrences(
      tree, input));

  print_inorder(tree);
  return 1;
}
```