Programmering for computerteknologi Hand-in Assignment Exercises

Week 10: Passing functions as arguments to other functions

Written by: Alexander A. Christensen (202205452)

Disclaimer: Due to errors with CMake that neither me, nor the TAs have solved, the test-cases have not been run. Instead, the functions have been manually tested.

The code can still be found at https://github.com/Aarhus-University-ECE/assignment-10-A-CHRI

Exericse 1

Given a linked list, we wish to print out its content recursively.

```c
void print_list(node *p) {
  /* Base case: Last element in the list (next = NULL)*/
  if (p->next == NULL) {
    printf("%d", p->value);
  }
  /* Recursive step: */
  else {
    print_list(p->next);
    printf("%d", p->value);
  }
}
```

Exercise 2

We wish to use recursion to sum the squared values of a linked list.

```c
int sum_squares(node *p) {
  /* Pre: Non-empty list */
  assert(p != NULL);

  /* Base case: Last element in the list */
  if(p->next == NULL) {
    return (p->value)*(p->value);
  }
  /* Recursive step: */
  else {
    return (p->value)*(p->value) + sum_squares(p->next);
  }
}
```

Exercise 3

Creating a `map` function, we have to remember to allocate a new list. Besides this the `map` function works mostly like a standard `fold` function would.

```c
typedef int (*fn_int_to_int)(int);

int square(int x) { return x * x; }

node *map(node *p, fn_int_to_int f)
{
  /* Pre: Non-empty list */
```

```
  assert(p != NULL);

  /* Base case: Last node */
  if(p->next == NULL) {
    node * q = make_node( f(p->value), NULL);
    return q;
  }
  /* Recursive step: */
  else {
    node * q = make_node( f(p->value), NULL);
    q->next = map(p->next, f);
    return q;
  }
}
```

Exercise 4

A binary tree has the following properties, which has been implemented.

- `Insert(x,t)` - Insert item `x` into tree `t`
- `Remove(x,t)` - Remove item `x` from tree `t`
- `Contains(x,t)` - Return`true` if the tree `t` contains item `x`. Return `false` otherwise
- `Initialize(t)` - Create an empty tree
- `Empty(t)` - Boolean function

```
#include "btree.h"
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

struct tree_node *Insert(int x, tree_node *t)
{
    // Insert item x into the tree t

    /* Base case */
    if (t == NULL)
    {
        tree_node *new = (tree_node *)malloc(sizeof(tree_node));
        new->item = x;
        new->left = NULL;
        new->right = NULL;
        return new;
    }

    /* Recursive step */
    if (x <= t->item)
    {
        t->left = Insert(x, t->left);
    }
    else if (x > t->item)
    {
        t->right = Insert(x, t->right);
    }

    /* Return the tree */
    return t;
}

struct tree_node *Remove(int x, tree_node *t)
```

```
{
    // Remove one item from the tree t
    /* Base case */
    if (t == NULL)
        return t;

    /* Traverse left or right depending on value */
    if (x < t->item)
        t->left = Remove(x, t->left);

    else if (x > t->item)
        t->right = Remove(x, t->right);

    /* Check if the current node is the correct node */
    else
    {
        /* Case 1: No child or 1 child */
        if (t->left == NULL)
        {
            tree_node *temp = t->right;
            free(t);
            return temp;
        }
        else if (t->right == NULL)
        {
            tree_node *temp = t->left;
            free(t);
            return temp;
        }

        /* Case 2: Node with two children */
        /* Find the leftmost leaf in the right subtree*/
        tree_node *temp = t->right;
        while (temp && temp->left != NULL)
            temp = temp->left;

        t->item = temp->item;

        t->right = Remove(temp->item, t->right);
    }
    return t;
}

int Contains(int x, tree_node *t)
{
    /* If node is NULL */
    if (t == NULL)
        return 0;

    /* Traverse the tree in-order */
    if (t->item == x)
        return 1;

    if (x <= t->item)
        return Contains(x, t->left);

    if (x > t->item)
        return Contains(x, t->right);

    return 0;
}

struct tree_node *Initialize(tree_node *t)
{
    // Create an empty tree
```

```
101     t = NULL;
102     return t;
103 }
104
105 int Empty(tree_node *t)
106 {
107     return t == NULL;
108 }
```