Exercise 8.1

Solution Hints

Solutions for this question provided by Scott:

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

typedef struct list_node_t{
    char c;
    struct list_node_t *next;
} list_node;

typedef list_node * list;

list new_node(char c);
void append_node(list*,list);
int length_node_r(list);
int length_node_l(list);
void dup_node_r(list);
void dup_node_l(list);
void reverse_r(list*);
void reverse_l(list*);
void print_node(list);
list seq_r(char,char);
list seq_l(char,char);
list copy_r(list);
list copy_l(list);

int main(void){
    list l2;
    list l = new_node('a');
    append_node(&l, new_node('b'));
    dup_node_r(l);
    dup_node_l(l);
    reverse_l(&l);
    l = seq_l('a','d');
    l2 = copy_l(l);
```
list new_node(char ch) {  
    list l;
    
    l = malloc(sizeof(list));
    
    l→c = ch;
    l→next = NULL;
    return l;
}

For the character list type CharList from the lecture, write both recursive and iterative functions that perform the following tasks:

(a) Calculate the length of a list.

**Solution Hints**

```c
int length_node_r(list l){
    if(l == NULL)
        return 0;
    else
        return length_node_r(l→next);
}
```

```c
int length_node_i(list l){
    int i = 0;
    while(l ≠ NULL){
        l = l→next;
        i++;
    }
    return i;
}
```

(b) Duplicate each list element, thus turning for example “abcd” into “aabbcccd”.

**Solution Hints**

```c
void dup_node_i(list l){
    list tmp;
    
    while(l ≠ NULL){
        tmp = l→next;
        l→next = new_node(l→c);
        l→next→next = tmp;
        l = tmp;
    }
}
```
(c) Given two characters \( x \leq y \), produce a list containing in sequence all characters from \( x \) to \( y \) inclusively.

**Solution Hints**

```c
list seq_i(char x, char y){
    list l=NULL, tmp;
    for(y>=x;y--){
        tmp = malloc(sizeof(list_node));
        tmp->c = y;
        tmp->next = l;
        l = tmp;
    }
    return l;
}
```

```c
list seq_r(char x, char y){
    list l;
    if(x>y)
        return NULL;
    else{
        l = new_node(x);
        l->next = seq_r(x+1,y);
        return l;
    }
}
```

(d) Produce a copy of a list.

**Solution Hints**

```c
list copy_r(list src){
    list trg;
    ```
if(src==NULL)
    return NULL;
else{
    trg=malloc(sizeof(list_node));
    trg→c=src→c;
    trg→next=copy_r(src→next);
    return trg;
}
}

list copy_i(list src){
    list trg,tmp,head=NULL;
    for(;src != NULL; src=src→next) {
        tmp = malloc(sizeof(list_node));
        tmp→c=src→c;
        tmp→next = NULL;

        if(head!=NULL)
            trg→next=tmp;
        else
            head=tmp;
        trg=tmp;
    }
    return head;
}
(e) Reverse a list.

Solution Hints

void reverse_r(list *listhead){
    list l;

    if(*listhead == NULL)
        return;
    else{
        reverse_r(&(*(listhead)->next));
        l = *listhead;
        while(l->next != NULL)
            l = l->next;
        l->next = *listhead;
        *(listhead) = l = l->next;
        *(listhead) = NULL;
    }
}

void reverse_i(list *listhead){
    list current,newcurrent,target;

    target = NULL;
    current = (*listhead);

    while(current != NULL){
        newcurrent = current->next;
        current->next = target;
        target = current;
        current = newcurrent;
    }
    *listhead = target;
}

Solution Hints

Additional material:

void print_node(list l){
    if(l==NULL){
        printf("\n");
    }
    else{
        printf("%c",l->c);
        printf("%c",l->c);
        print_node(l->next);
    }
}
typedef struct {
    struct AppNode
    } Solution

One aspect to keep in mind is that it should be reasonably easy to add and delete single appointments.

Exercise 8.3 — Calendar (ctd.)
For the calendar application of Exercise 6.2, adapt the Day data type to allow an arbitrary number of appointments, and adapt your find function accordingly.

One aspect to keep in mind is that it should be reasonably easy to add and delete single appointments.

Solution Hints

typedef struct { int hour, minutes; } MyTime;

typedef struct {
    MyTime begin, end;
    char * title;
    char * comment;
} Appointment;

typedef struct ANstruct {
    Appointment data;
    struct ANstruct * next;
} AppNode;

typedef AppNode * AppList;

typedef struct {
    MyTime sunrise, sunset;
    AppList appointments;
} Day;

void find(int monthsNum, int monthStart[], int yearLen, Day cal[], char * (*check)(Appointment a)) {
    int i;
char * message;
AppList l;
for ( i=0; i<yearLen; i++ ) {
    l = cal[i].appointments;
    while ( (l != NULL) ) {
        if ( (message = check(l->data)) ) {
            printf("%s ", message);
            printDate(monthsNum, monthStart, i);
        }
        l = l->next;
        }
    }
}

Exercise 8.4 — Number Lists  (51% of Midterm 3, 2005)
The following C type definitions will be used to define “number lists” as singly-linked lists of int elements:

```c
typedef struct NumListNodeStruct {
    int elem;
    struct NumListNodeStruct * next;
} NumListNode;
```

typedef NumListNode * NumList;

The considered number lists will always have their elements in ascending order.
(The items are independent of each other!)

(a) ≈ 12%  **Implement** the summing up of all the elements in a list.

Define two versions: one recursive and one iterative function.

**Document** the function interface!

**Solution Hints**

/* sum() returns the sum of all elem fields; the argument list is passed by value. */
int sum(NumList d) {
    if ( (d == NULL) ) return 0;
    else return d->elem + sum(d->next);
}

int sumIter(NumList d) {
    int r = 0; /* result accumulator */
    while ( (d != NULL) ) {
        r += d->elem; d = d->next;
    }
    return r;
}

(b) ≈ 39%  **Design** and **implement** a function that splits a list into two sub-lists, one containing all the even numbers from the original list, and the other all the odd numbers from the original list (both in
ascending order). Carefully document the function interface.

**Solution Hints**

Argument is destructively updated to loose all its even elements, which are returned as result:

```c
NumList splitOutEven(NumList * list) {
    NumList result = NULL;
    NumList * resultEnd = &result;

    while ( *list ) {
        if ( ( *list )->elem % 2 ) { // odd
            list = &(( *list )->next);
        } else { // even
            *resultEnd = *list;
            *list = (*list)->next;
            resultEnd = &(( *resultEnd )->next);
            *resultEnd = NULL;
        }
    }
    return result;
}
```