Lab 1: Linked List

1 Introduction

The task is to create a telephone directory, where each record stores a name and phone number of one person. The phone number also functions as the ID of a record. The telephone directory shall be implemented as a singly linked list. Operations on the telephone directory such as searching or adding a record shall be implemented as a number of functions working on the linked list nodes.

2 Structure of a Telephone Directory Record

Each record in the telephone directory shall be implemented by a linked list structure containing a name, phone number and a link to the next node. The node structure is briefly described below. The detailed specification of the structure can be found in the header file `lab_lists.h`, which must be included in your `.c` file.

Linked List Structure

Every linked list node shall contain:

1. A *phone number*, which also functions as the ID of a record. A phone number can be implemented as a simple string of digits containing no spaces, dashes or parentheses inside the phone number. Other implementations are also acceptable, e.g. as a char array.
2. The *name of a person* having this phone number. It can be first name, last name or both names. You can choose a reasonable length of a name that will be enough for the purpose of this lab.
3. A pointer to the *next node* of a list.

3 Functions Implementing Operations on Telephone Directory

You shall implement five functions to operate on a telephone directory: adding a record, searching for a record, deleting a record as well as displaying and deleting the entire directory. The functions are briefly described below. The detailed specification of the functions can be found in the header file `lab_lists.h`, which must be included in your `.c` file.

Telephone Directory Operations

The following functions implement operations on a telephone directory:

1. *Display all records* in the telephone directory. The output for each record is the ordinal number of the record in the directory, the phone no., and the name.
2. *Insert a new record* at the beginning of the directory. Duplicates are not allowed. The record data (a phone number and the name) can be input from a console.
3. *Delete a record* with the given phone number. Only an existing record can be deleted.
4. *Query the directory* by phone number or name.
5. *Delete all records* in the telephone directory and set the start pointer to |NULL|. It is supposed to be called before exiting. It is a good practice when a program frees all the allocated memory at some point.

You may want to revise the algorithms from the lecture on linked lists or Ch. 6 of the textbook before you begin the lab work. However, the textbook often introduces algorithms divided into separate cases, e.g. depending on the placement of a node to delete. For this lab, you will have to do some work to implement an algorithm that works in different cases of deletion.

When you test your functions, you need to pay attention not only to normal, usual inputs but also to *boundary test cases* (values at the extreme ends of a given input domain), especially when testing the deletion functions. For example:

1. The list is empty.
2. The list has one element.
3. The list has several elements.
4. The element to be deleted is at the beginning of the list.
5. The element to be deleted is in the middle of the list.
6. The element to be deleted is at the end of the list.

4 Deliverables

The deliverable is the source code that includes the list structure specified in Sect. 2 and functions specified in Sect. 3 as well as the `main()` function that is supposed to demonstrate the functionality of all the telephone directory functions.

The start pointer to the list should be declared in the `main()` function and passed to the operations explicitly. It is not recommended to use global variables as implicit parameters of functions. The header file `lab_lists.h` *must be included* in your implementation. It is up to you to decide how you code the rest of the `main()` function.

The source code has to be thoroughly tested as well as to adhere to the recommendations on *Standard C and portability*, which you can find in Canvas. You can compile your code using more strict flags `-Wall` together with `-pedantic` before submitting the code, e.g.

```
$ gcc -Wall -pedantic program.c -o program
```

Moreover, all the source code has to include reasonable amount of comments, that is the code is supposed to be well-readable. Please include a comment in your program (all `.c` files) with the year and your name to indicate authorship:

```
//2025 Your Name
```