Specification Document

Omkar Girish Kamath

May 8, 2023

Contents

1 Introduction

This specification document outlines the design ,features ,functions and installation of the **Dynamically Linked Library of Linked List Functions**. The library is designed to provide commonly used operations for creating and manipulating linked lists. The library is written in C programming language and will be compatible with Linux operating systems. The appropriate Makefile utility files are written and provided as well with the library to ensure smooth and hiccup-less usage of the library.

2 Installation in Linux Systems

2.1 Introduction to DLLs

Dynamically linked libraries (DLLs) are shared libraries that contain compiled code and data that can be linked and loaded into a program at run-time, rather than at compile-time. In contrast, static libraries are linked and loaded into the program at compile-time.

When a program is dynamically linked to a DLL, the program will load the DLL into memory at runtime and resolve any unresolved external symbols, such as function names, that are referenced in the program code. This allows multiple programs to share the same DLL, reducing memory usage and making updates to the library easier. The process of using a DLL in a program involves a few steps. The program code contains references to functions or data that are defined in the DLL. The program is compiled with references to the DLL. The program can call functions and access data in the DLL as needed.

2.2 Setting the Environment Variables

The gcc dynamic linker looks for the shared object (.so) file at runtime using **LD_LIBRARY_PATH** and gcc compiler looks for header files at compilation using C_INCLUDE_PATH and to not mention the path explicitly during compilation we need to add the directory containing the shared object (.so) and the header file (libll.h) file to the environment variable **LD_LIBRARY_PATH** and C_INCLUDE_PATH respectively. C_INCLUDE_PATH is used during the compilation process to find header files needed by the code being compiled, while **LD_LIBRARY_PATH** is used at runtime to find shared libraries needed by the executable. But to ensure that the environment variable remembers it even after system reboots, we need to change the configuration file of the user shell. For the sake of example the **Bourne Again Shell (bash)** is considered. So we add the line

export LD_LIBRARY_PATH=/path/to/sharedobjectfile/:$LD_LIBRARY_PATH

export C_INCLUDE_PATH=/path/to/headerfile/:$C_INCLUDE_PATH

3 Intended Usage and Working of The Library

3.1 Makefile

To use this library firstly you need to use the Makefile provided in the directory main_files. Open your shell terminal and use the command "make" in the directory main_files. This will compile the library source files depending upon the dependencies and produce the target shared object file if its not present from before. Then you need to use the "make install" command which will create a directory in home/user/ called lib where the shared object file be copied to.

Similarly another directory is created in home/user/ called include where the header file is copied to. Then the command ldconfig creates a symlink to the shared library in the lib directory.

3.2 Compilation of the File

Now you can use the library, the gcc command would be :

gcc -L /home/user/lib yourfile.c -lll

4 Function Descriptions and their Purpose

4.1 Function Name: _ll_create()

Arguments: None.
Returns: Head pointer to new list (ll *)
Description: Non-User function to create new lists or nodes. Its a private function meant not to be used by the user, rather used by other functions to create nodes. User can use the lladd() function to create and add nodes.

4.2 Function Name: lladd()

Arguments: Takes in a pointer to a list (or NULL if a new list needs to be created) and a pointer to the data payload that will be added to the newly created node.
Returns: Head pointer to either a new list, or the same list if the head argument is non-null (ll *).
Description: It is a User function to create a new list or add nodes to the end of given linked list.

4.3 Function Name: llprint()

Arguments : A starting node from to start printing from and a a function pointer which takes in payload data of a node and returns the string that is to be printed.
Returns : None
Description : User function to print all the nodes occuring after the starting node

4.4 Function Name: llsearch()

Arguments : A starting node pointer to begin searching from, a key a function pointer which takes in payload data of a node and returns the string that is to be printed.

Returns : Pointer (ll *) containing payload data matching with key.
Description : User function to search for the payload data in the linked list

4.5 Function Name: lldelete()

Arguments : Head pointer of the list delete, node marked for deletion and function pointer to delete payload data from nodes.
Returns : NULL pointer if list has 0 elements or has 1 element and user data matches with that node. Original head (ll *) for all other cases.
Description : User function to delete node and its payload data which matches node marked for deletion.

4.6 Function Name: llappend()

Arguments : A node pointer 1 and a node pointer 2.
Returns : head pointer formed afte appending list with head node pointer 2 to tail of list formed by head pointer 1.
Description : User function to append linked list 2 to the end of linked list 1.