CS 320 C Programming Assignment #2

50 points **Due in edoras account: Wednesday, October 7 11:59pm**

The Assignment

For this assignment, you will write a linked list application in the C programming language. This program uses C features: FILE, perror, .h file, prototypes, struct, typedef, malloc, free, pointers, returning struct from a function, and makefile. Your project structure is:

- llist.h : definition of linked list node and function prototypes (provided)
- llist.c : implementation of a linked list
- p2.c : driver (client) for the linked list
- makefile : a Unix utility for building the project (provided)

Deliverables

The name of your C program files must be `p2.c` and `llist.c`. These must be in the directory named `p2` within your handin folder:

```
~/handin/p2/
            llist.c
            p2.c
```

Carefully note the name of all directories. Use lowercase for directories and program files. Your programs must compile on edoras with the compile commands utilized in the makefile.

Development

To build the program on edoras in your sandbox, the files needed are:

```
~/sandbox/p2/
             llist.c
             llist.h
             inputfile.txt (and other input files as needed)
             makefile
             p2.c
```

While you are developing and testing your program, your sandbox will also contain:

```
            llist.o     llist.c compiled object file
            p2.o        p2.c compiled object file
            p2          executable application file
```

Requirements p2.c

Write a C program that exercises the functions of your linked list (`llist.c`).

Header File

For this part of the assignment, your `p2.c` file should include **only** the header file "`llist.h`". No other `#include` directives are permitted in your `p2.c` program file.

Functions and Procedures

main—Your p2.c should do these steps in order:

1. Output student/account info
2. Error check command line arguments using perror()
3. Open file (error check) using fopen()
4. Read in file content into a char buffer and echo buffer using `printf("> %s\n", buffer);`
5. Create list from buffer using strToList()
6. print() the newly created list
7. Print size()
8. insert() and 'o' after the letter 't' if 't' is in the list. Do nothing otherwise.
9. Print size()
10. print() the list
11. delete() the third letter in the list you may assume the list has at least three elements)
12. deleteList()
13. fclose() file pointer
14. return(0)

Output:

Your program will first output printId(). Print the output in the following form; the **bold print** indicates what the user typed in:

Example 1 (input.txt contains one word—September):

```
[cssc0000 ~]$ p2 input.txt

Program 2, cssc0000, Firstname Lastname

> September
September: S
eptember: e
ptember: p
tember: t
ember: e
mber: m
ber: b
er: e
r: r
S e p t e m b e r
9
S e p t o e m b e r
10
S e t o e m b e r
```

```
deleting S
deleting e
deleting t
deleting o
deleting e
deleting m
deleting b
deleting e
deleting r
```

Example 2: missing command line argument

```
[cssc0000 ~]$ p2

Program 2, cssc0000, Firstname Lastname

Input file error: ...
```

Example 3: bad file name

```
[cssc0000 ~]$ p2 inpputttt.txt

Program 2, cssc0000, Firstname Lastname

Error opening fiie: ...
```

Additional Requirements p2.c

- Error checking is required for this assignment.

  If an input filename is not given on the command line, the program should call perror("Input file error) and then exit the program with -1.

  If a bad input filename is not given (or another reason such that the file cannot be opened for reading), the program should call perror("Error opening file") and then exit the program with -1.

- You may assume the input file data types are string values.
- Your source code file will be compiled on edoras with the `makefile` provided.
- Provide documentation at the top of the file that includes a program description, date, program name, class (CS320), and your name.
- Formatting exactly as shown will give you maximum output points.
- Some functions have output. See example output for pattern matching.

Requirements llist.c

This C program implements a linked list. It contains list operations (defined in `llist.h`) for:

- Creating a list
- Counting the elements
- Looking up an element
- Inserting an element
- Deleting an element
- Deleting all elements
- Printing the elements

Header File

For this part of the assignment, your `llist.c` file should include **only** the header file "`llist.h`". No other `#include` directives are permitted in your `llist.c` program file.

Additional Requirements llist.c

- The list must be dynamic, that is, each element will be stored in a structure allocated in memory at runtime.
- You must use recursion where indicated.
- Your source code file will be compiled on edoras with the `makefile` provided.
- Provide documentation at the top of the file that includes a program description, date, program name, class (CS320), and your name.
- Formatting exactly as shown will give you maximum output points.
- The instructor may use a different driver than your p2.c to test your llist.c, so plan to test your code beyond the capabilities of p2.c.

Header File

Your application requires a header file called `llist.h` which is provided. You may **not** alter it.

```
/* llist.h */
#include <stdio.h>
#include <stdlib.h>

typedef char Data;   /* store char in list */

struct llist {
   Data data;
   struct llist *next;
};
typedef struct llist Node;
typedef Node * Link;

/* strToList
 *    Recursively creates a dynamic list from a string (char[] in C0
 *       prints current string with each call to strToList
 *    Returns head of new list
 */
Link strToList(char s[]);

/* size
 *    Recursively counts elements in the list
 *    Returns int
 */
int size(Link head);

/* find
 *    Recursively iterates through the list to locate the Data element
 *    Returns NULL if list is empty or it reaches end of list (NULL)
 *    Returns Link a pointer to the Data element if found
 */
Link find(Data c, Link head);

/* insert
```

```
 *    Adds element q to the list after element p1 and before element p2
 *    Returns void
 *    Assumptions: arguments passed to insert are valid Link elements
 */
void insert(Link p1, Link p2, Link q);

/* delete
 *    Removes element q whose predecessor is element p
 *    Returns void (q contains a pointer to the removed element)
 */
void delete(Link p, Link q);

/* deleteList
 *    Recursively deletes an element from the list, freeing memory
 *       as each element is removed. Prints the element being deleted.
 *    Returns void
 */
void deleteList(Link head);

/* print
 *    Recursively prints the Data elements in the list
 *    Returns void
 */
void print(Link head);
```

makefile

Our application uses separate files: a header file for structure/type definitions and function prototypes to be reused by other files; a linked list utility; and a driver file. The C source files can be separately compiled, which is efficient for both the programmer and the machine. The compiled files are linked together to form the executable file for the application.

The Unix *make* utility defines file dependencies and recompiles just the modules that have been modified since the last build. The default name of the file used by *make* is *makefile.* In class (or with a short demo) you will learn about makefile organization and usage. Below is the makefile for Program 2.

```
#Program 2 makefile

p2 : p2.o llist.o
        gcc -g -Wall -o p2 p2.o llist.o

p2.o : p2.c llist.h
        gcc -g -Wall -c p2.c

llist.o : llist.c llist.h
        gcc -g -Wall -c llist.c

clean :
        rm p2.o llist.o

cleanall :
        rm p2.o llist.o p2
```