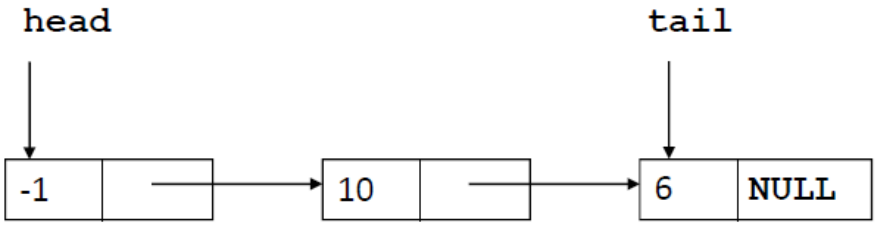
LAB 7 — Pointers to Pointers and File I/O

Problem A

1. Specification

Refer to the lecture nodes for the description of a singly linked list. In our implementation, the linked list stores non-negative integers. A dummy node with the data value -1 is used to simplify insertions and deletions. See the diagram below.

Write a C program to implement the insertion and deletion operations.



Note: The difference between this lab exercise and problem B of Assignment 1 is that pointer variables `head` and `tail` are no longer global variables. They are now local variables defined inside the `main` function and passed to the insertion and deletion functions.

2. Implementation

- The program to be submitted is named `slist.c`. **Use the given template `slist.c`** and fill in your code. **Submit only file `slist.c`.**
- You are also given a file named `slistMain.c` to test your code. Do not submit file `slistMain.c`.
- The first function to be implemented is `insert()`. See file `slist.c` for its specification. The new element is to be inserted at the end of the list. If a new node cannot be created (e.g., insufficient memory), the function calls function `prtError()` to display an error message and exit the program using `exit(1)`.
- The second function to be implemented is `removeFirst()`. See file `slist.c` for its specification. If the list is empty (i.e., no element other than the dummy node), the function calls function `prtError()` to display an error message and returns -1. Otherwise, it removes the first element (i.e., the node right behind the dummy node) and returns the data (integer) of the removed node.
- You may define your own variables inside functions `insert()`and `removeFirst()`.

- In file `slist.c` you are given three utility functions: `init()`, `prtError()`and `prtList()`. DO NOT modify these functions.
- Do not modify the function and structure definitions in file `slist.c`.
- To compile both files `slist.c` and `slistMain.c`, use the following command:

```
cc slist.c slistMain.c
```

3. Sample Inputs/Outputs

See file `slist.out` for the output from running programs `slist.c` and `slistMain.c`.

Problem B

1. Specification

Write a C program to input student records from a file. Each entry of the input file has the following format:

[First name] [Last name] [Assignment 1 mark] [Assignment 2 mark]

Copy the input data to an output file, and add one more field which records the average of the two assignment marks. So each entry of the output file has the following format:

[First name] [Last name] [Assignment 1 mark] [Assignment 2 mark] [Average mark]

2. Implementation

- The program to be submitted is named `marks.c`. **Use the given template `marks.c`** and fill in your code. **Submit only file `marks.c`.**
- To compile the program, use the following command: **`cc marks.c –o marks`**
- The file names are input as command line arguments. File names are less than 30 characters long.
- Sometimes users may forget the command syntax and they may type only the command "`marks`", or enter only one file name. In that case, display the following reminder message on the standard output:

```
Usage: marks [input_file] [output_file]
```

- Assignment marks are integers. Average marks are of type `float`. Output the average marks with one decimal digit.

- If a file cannot be opened for read or write, display an error message and exit the program using `exit(1)`.
- Use `fscanf()` to read from the input file and `fprintf()` to write to the output file.
- You may define your own variables inside function `main()`, and implement your own function(s) in file `marks.c`.

3. Sample Inputs/Outputs

See file `marks_snapshot` for a snap shot of program execution, and contents of the input and output files.

Common Notes

- Complete the header in files `slist.c` and `marks.c` with your student and contact information.
- Assume that all inputs are valid. No error checking is required on inputs.