MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS
CNG213 Data Structures – Worksheet 4

Fundamentals of Linked Lists
(Pointers and Dynamic Memory Allocation)

Purpose:

This worksheet is concerned with the fundamentals of linked lists. You are not expected to complete the entire worksheet in class. Work on as many problems as you can; you are expected to work and complete the remaining problems on your own. You can use this worksheet for practice and to test your understanding of the fundamentals of linked lists.

At the end of this worksheet, you will understand:

1. static vs. dynamic memory allocation
2. traversal of linked lists
3. simple algorithms and pointer manipulation for linked lists

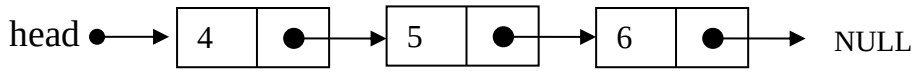
A linked list is recursively defined as

a. an empty list, or
b. a node followed by a list

This recursive definition gives rise to the following declaration for linked lists:

```
struct Node
{
int val;
struct Node *next;
};
```

A picture might look as follows:



For the questions from 1 to 13, you have to use the given CNG213_Worksheet4_Template.c file

1) Implement a C function `StaticBuild1234()` that builds a linked list containing four elements by using static memory allocation.
2) Implement an iterative C function `PrintList()` that traverses a linked list and prints out the values.
3) Implement a C function `DynamicBuild1234()` that builds a linked list containing four elements by using dynamic memory allocation. Let the function return the pointer to the first element of the list as a result.
4) Modify `DynamicBuild1234` function such that it creates a linked list without an assumption related to the length of the linked list. When the user enters -1 as a value, the linked list should finish.
5) Write a C function called `SearchItem()` that takes a linked list and an integer, and checks if this integer is in the list. If this value is in the list then it returns 1 (TRUE) and otherwise returns 0 (FALSE) if this value is not in the list.
6) Write a C function called `MinValue()` that takes a linked list and returns the minimum value in this linked list. Assume that the list does not include repeated values.
7) Write a function called `SumList()` that takes a linked list and returns the total of the values in the given linked list.
8) Implement a recursive C function `RecursivePrintList()` that traverses a linked list and prints out the values.

9) Implement a C function `ReversePrintList()` which prints the elements in a linked list in reverse order using recursion. In other words, the last element in the list is printed first, then the second-last element all the way to the first element in the linked list which is printed last. Think about iterative and recursive solutions; the iterative solution will contain two nested while loops.

10) Implement a C function `CopyList()` that takes a linked list and makes a copy of it. Let the function return the pointer to the first element of the list as a result.

11) Implement a C function `DestroyList()` that traverses a linked list and frees the memory occupied by the list elements one element at a time.

12) Implement a C function `ReverseList()` that takes a linked list and creates a new linked list where the elements of the original list appear in reverse order in the new list. You may give a recursive or an iterative solution. Let the function return the pointer to the first element of the reversed list as a result. Think about iterative and recursive solutions.

13) Write a program which will do following;

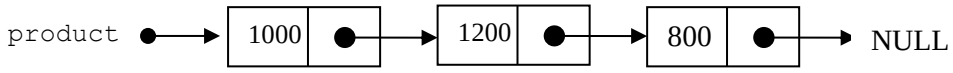
    a) `Node` declaration of a Linked List with prices of a product.

    b) A function `InsertProduct()` that creates a linked list of the `product` and fills it with the prices per year taken from the user.

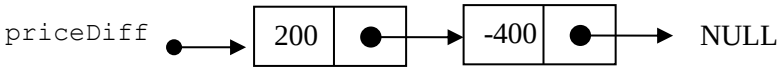
    c) A function `PriceChanges()` that takes the linked list of the `product` as an input, and then creates a linked list of a `priceDiff` and fills it with the yearly price differences and then returns.

    d) A function `PrintPriceChanges()` that takes the linked list of `priceDiff` as an input, traverses and prints the changes accordingly.

    If the input prices are as 1000, 1200 and 800 (please note that you cannot make any assumptions regarding to the size of the list (number of years)), then list formed by `InsertProduct()` should be as follows;



    Then the return list from `PriceChanges()` should be as follows;



    A sample run would be as follows:

```
************************************************
1) Create yearly price for the product
2) Display yearly price changes for the product
3) Exit
What would you like to do? 1
************************************************
How many years? 3
Enter prices for the product: 1000 1200 800
Prices for the product is created successfully!
************************************************
1) Create yearly price for the product
2) Display yearly price changes for the product
3) Exit
What would you like to do? 2
************************************************
```

```
Changes per year are as follows:
Year 1: Price increased 200 £
Year 2: Price decreased 400 £
**************************************************
1) Create yearly price for the product
2) Display yearly price changes for the product
3) Exit
What would you like to do? 3
```