Template Linked List

Overview

Usual Linked List implementation in C must have specific type of data to be held. In case if user wants to create another linked list with it's own data, he need to write basicly a new linked list for it. Functions for adding the data to the list must be written based on this specific type.

Another point is that if user want to insert data to a specific index of a linked list, the list must initialize itself first with empty nodes up to that index (if index exceed size of a list) and after take care of inserting data to this place. The worst thing here is that all empty nodes for example for integer value are not really empty and takes memory away (if the data of a linked list not a pointer).

Template Linked List (TLL) will handle all initial steps of creating nodes with specific data type and creating functions to insert this data. Moreover, user can create as much TTL with different data types as he wants in a very simple steps. At the end interface of doing this is simple, most work with a memory has already done by TTL algorithm and user just need to keep track of what kind of data in the list. TTL is a nested implementation of some of C++ templates usage.

How it's works

First the header needs to be included:

```
#include "TemplatedLinkedList.h"
```

After user need to declare in the same way as global variable will be:

```
typedef TLinkedList(MyNodeName, int) MyTLinkListInt;
CreateFuncForTLinkedList(MyTLinkListInt, int);
```

where:

MyNodeName -	Name of the node (struct) (user don't require to use it later)
int -	Data type (can be any, but not "char *", for string separate typedef of struct containing char[] need to be initialized and passed)
MyTLinkListInt -	It is a name of node of a linked list

CreateFuncForTLinkedList will define functions which can be used to add data to the list. Each function name will start with a name of node of a linked list + _funcName. For example, MyTLinkListInt_push(...)

These two commands need to be used only in the order shown above!

At the end our template linked list can be used in these ways:

```
//Header of a linked list (must be declared as NULL)
MyTLinkListInt *myTListS = NULL;
int number = 0;

// Insert the data into Linked List to the end
MyTLinkListInt_push(&myTListS, &(number));
MyTLinkListInt_replaceByIndex(&myTListI, &(number), 5);

printMyListInt(&myTListS){
    MyTLinkListInt *ptrCurrentNode = *head;
    int indexCounter = 0;
    while (ptrCurrentNode != NULL) {
        if (ptrCurrentNode->data != NULL)
            printf("Node %d: %d\r\n", indexCounter, *(ptrCurrentNode->data));
        ptrCurrentNode = ptrCurrentNode->next;
        indexCounter++;
    }
}
```

More information

- Template linked list is flexible in size. If user wants to insert data somewhere out of boundaries, TTL will create empty nodes with only 4 byte size each. It will not allocate any additional memory for data.

Disadwantage:

- Hard to debug, since the function for a template linked list and basicly node defined by using macros.