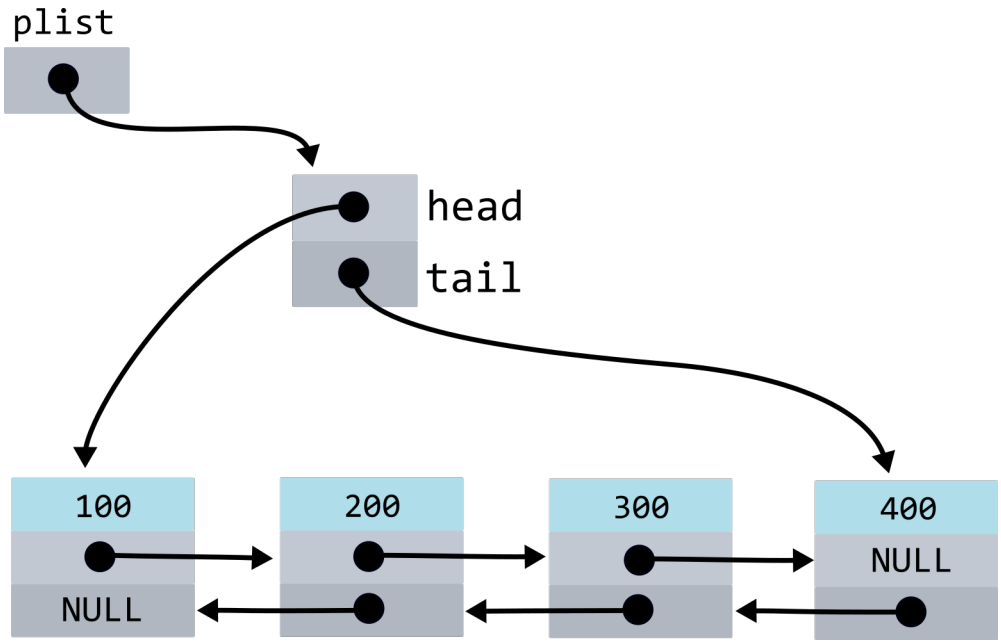
Семинар #11: Связный список. Домашнее задание.

На классном занятии был пройден односвязный список. Он имеет несколько преемущевств по сравнению с массивом, например, быстрая вставка элементов в начало и быстрое удаление из начала. Однако у него есть также ряд недостатков. К примеру, чтобы удалить элемент из конца односвязного списка, нужно пройти по всему списку, чтобы найти указатель на предпоследний узел, а это долго ($O(N)$). Аналогичная проблема есть и при вставке/удалении в середину списка. Эти проблемы решаются в двусвязном списке. В таком списке в каждом узле хранится не только указатель на следующий элемент, но и указатель на предыдущий. Также, помимо указателя на начало списка(`head`) будем отдельно хранить указатель на конец списка(`tail`). В результате все операции по вставке и удалению будут занимать $O(1)$.

Связный список используется там где нужно хранить набор элементов, в который нужно часто добавлять и удалять элементы. Также связный список используется как составная часть других структур данных (хештаблицы, графы и др.).

Операция	Массив	Односвязный список	Двусвязный список
Доступ по номеру	$O(1)$	$O(N)$	$O(N)$
Поиск	$O(N)$	$O(N)$	$O(N)$
Вставка в начало	$O(N)$	$O(1)$	$O(1)$
Вставка в конец	$O(1)$	$O(N)$	$O(1)$
Вставка в середину (по указателю на элемент)	$O(N)$	$O(1)$ после и $O(N)$ до элемента	$O(1)$
Удаление из начала	$O(N)$	$O(1)$	$O(1)$
Удаление из конца	$O(1)$	$O(N)$	$O(1)$
Удаление из середины (по указателю на элемент)	$O(N)$	$O(N)$	$O(1)$

Для реализации двусвязного списка на языке `C` дополнительно к структуре `Node` создадим структуру `List`, которая будет хранить 2 указателя `head` и `tail`.



Задачи

Начальный код в файле `doublylist.c`. В этом файле уже написаны следующие функции:

- `List* list_create()` – инициализирует список (создаёт и возвращает список нулевого размера). Эта функция выделяет память под структуру `List`, задаёт `head` и `tail` и возвращает указатель на эту структуру.
- `void list_print(const List* plist)` – распечатывает все элементы списка.
- `void list_add_last(List* plist, int x)` – добавляет элемент `x` в конец списка.

Напишите следующие функции для работы с двусвязным списком:

1. `int list_size(const List* plist)` – возвращает количество элементов списка.
2. `void list_add_first(List* plist, int x)` – добавляет элемент `x` в начало списка. Отдельно рассмотрите случай, когда список пуст.
3. `void list_insert_after(List* plist, Node* p, int x)` – добавляет элемент `x` сразу после узла, на который указывает `p`. Нужно рассмотреть несколько случаев:
   (a) Когда список пуст
   (b) Когда `p` указывает на последний элемент
   (c) Остальное
4. `int list_remove_first(List* plist)` – удаляет элемент из начала списка и возвращает его значение. Не забудьте изменить `head`. Отдельно рассмотрите случаи когда список пуст и когда он состоит из одного элемента.
5. `int list_remove_last(List* plist)` – удаляет элемент из конца списка и возвращает его значение. Не забудьте изменить `tail`. Отдельно рассмотрите случаи когда список пуст и когда он состоит из одного элемента.
6. `int list_remove(List* plist, Node* p)` – удаляет элемент на который указывает `p` и возвращает его значение. Нужно рассмотреть несколько случаев:
   (a) Когда список пуст. Нужно написать сообщении об ошибке и выйти(`exit(1)`).
   (b) Когда список состоит из одного элемента.
   (c) Когда элементов больше 1, а `p` указывает на первый элемент.
   (d) Когда элементов больше 1, а `p` указывает на последний элемент.
   (e) Остальное
7. `Node* list_find(const List* plist, int x)` – ищет элемент `x` в связном списке и возвращает указатель на соответствующий узел. Если такого элемента нет, то функция должне вернуть `NULL`.
8. `int list_destroy(List* plist)` – освобождает всю память, выделенную под список. Так как память выделялась под каждый элемент отдельно, то освобождать нужно также каждый элемент по отдельности. Также нужно не забыть освободить структуру `List`.
9. Реализовать абстрактный тип данных стек(Queue) на основе двусвязного списка.
10. В этой задаче вам нужно будет создать заголовочные файлы (другое название `header`-файлы). Это файлы с расширением `.h`, которые содержат код и подключаются к программе с помощью директивы `#include`. Директива `#include` ищет файл(в текущей директории или в специальных системных папках) и просто вставляет всё содержимое этого файла на место директивы.

    Создать `header`-файлы `doublylist.h` и `queue.h` (если сделали предыдущую задачу), в которых будет храниться реализация двусвязного списка и очереди. Создайте новую программу `main.c`, в которой протестируйте ваши реализации, включив заголовочные файлы `doublylist.h` и `queue.h`. Чтобы избежать многократного включения `header`-файлов используйте директиву
    `#pragma once`

    В качестве примера можете посмотреть решения классных задач `list.h` и `stack.h`.