C Programming

2_linked_lists

Списки

Список — это структура данных, которая хранит элементы в линейном порядке и позволяет вставлять и удалять элементы в любом месте последовательности. Списки могут быть односвязными, двусвязными или круговыми (циклическими).

Одно из главных преимуществ списков - быстрая вставка/удаление элементов.

В статическом массиве вставка/удаление новых элементов невозможна.

В динамическом возможна, но необходимо двигать все оставшиеся элементы массива, чтобы освободить место для нового элемента.

Таблица временной сложности

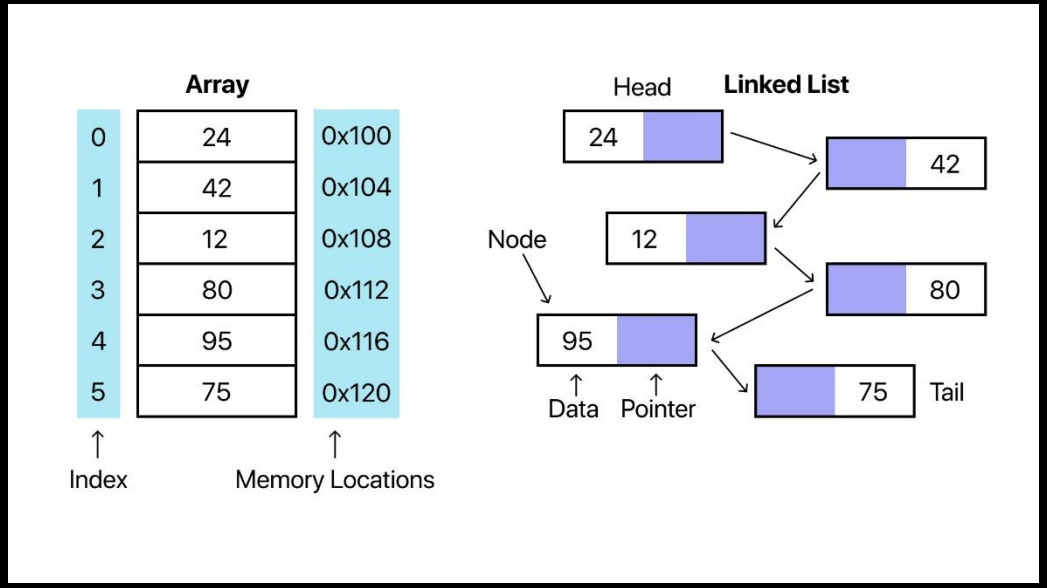
Параметр	Массив (статический)	Массив (динамический)	Односвязный список	Двусвязный список
Доступ	O(1)	O(1)	O(n)	O(n)
Вставка/ Удаление в начало	--- (размер фиксированный)	O(n) (нужно двигать элементы)	O(1)	O(1)
Вставка/ удаление в конец	--- (размер фиксированный)	O(1) (если есть место в массиве), O(n) (если места нет - копирование); удаление O(1)	O(n)	O(1)
Вставка/ удаление в середину	--- (размер фиксированный)	O(n) (нужно двигать элементы)	O(1) (но добираться до середины O(n))	O(1) (но добираться до середины O(n))
Поиск	O(n)	O(n)	O(n)	O(n)

Связанные списки (Linked List)

Связанный список представляет собой последовательность узлов, где каждый узел состоит из двух компонентов:

1. Данные: любые значения, например, число, структуры и т.д.
2. Указатель: ссылка на следующий узел в списке.

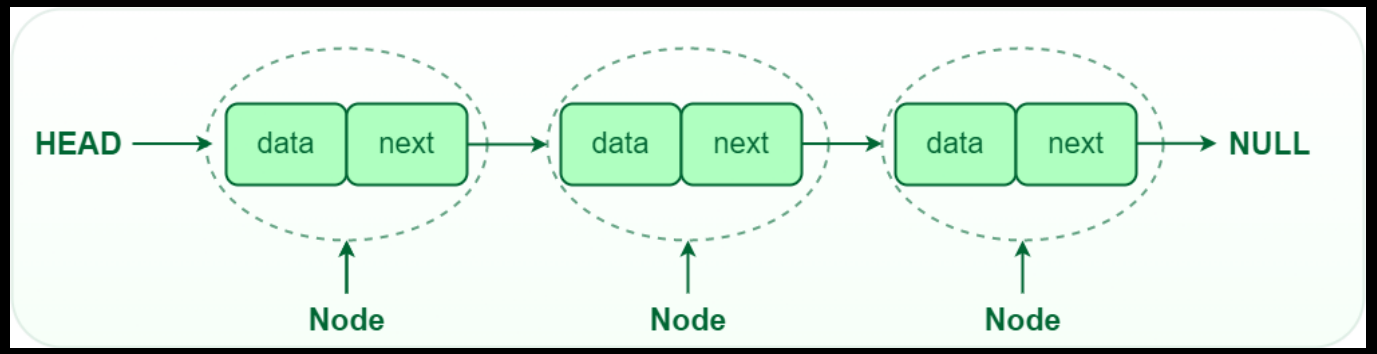
В отличие от массивов, связанные списки не хранят элементы в смежных ячейках памяти. Каждый узел указывает на следующий, образуя цепочечную структуру. Чтобы получить доступ к любому элементу (узлу), необходимо последовательно пройтись по всем узлам перед ним.



Односвязный список (Singly Linked List)

Односвязный список состоит из узлов. В каждом узле хранится указатель на следующий узел и сами данные. Указатели последнего узла (хвост - tail) указывают на NULL, что указывает на конец связанного списка (т.е. дальше нет узлов)

```
struct node
{
    int data;
    struct node *next;
};
typedef struct node node_t;
```



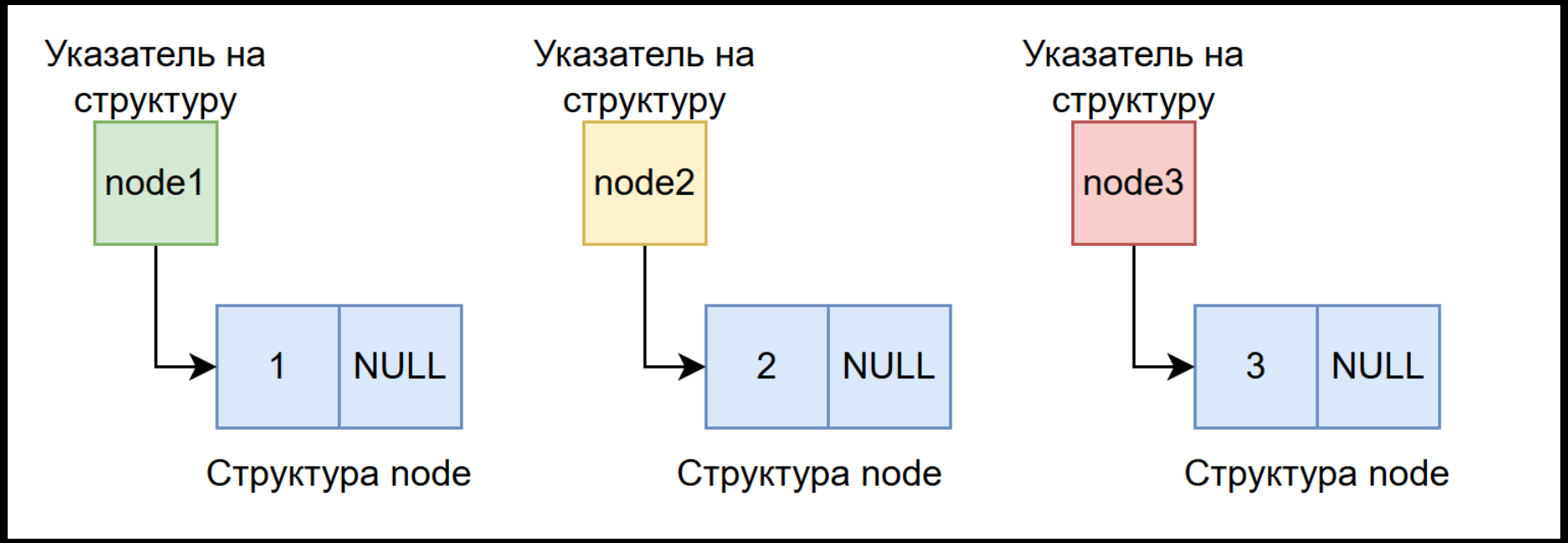
Пример структуры узла и односвязного списка из 3 узлов

Создаем узлы

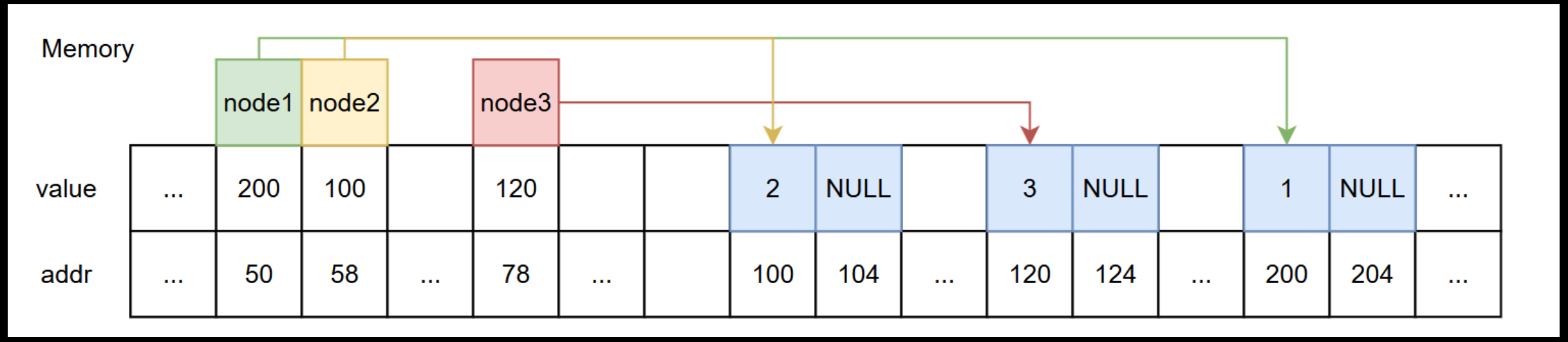
```
node_t *newNode(int data);

. . .

node_t *node1 = newNode(1);
node_t *node2 = newNode(2);
node_t *node3 = newNode(3);
```



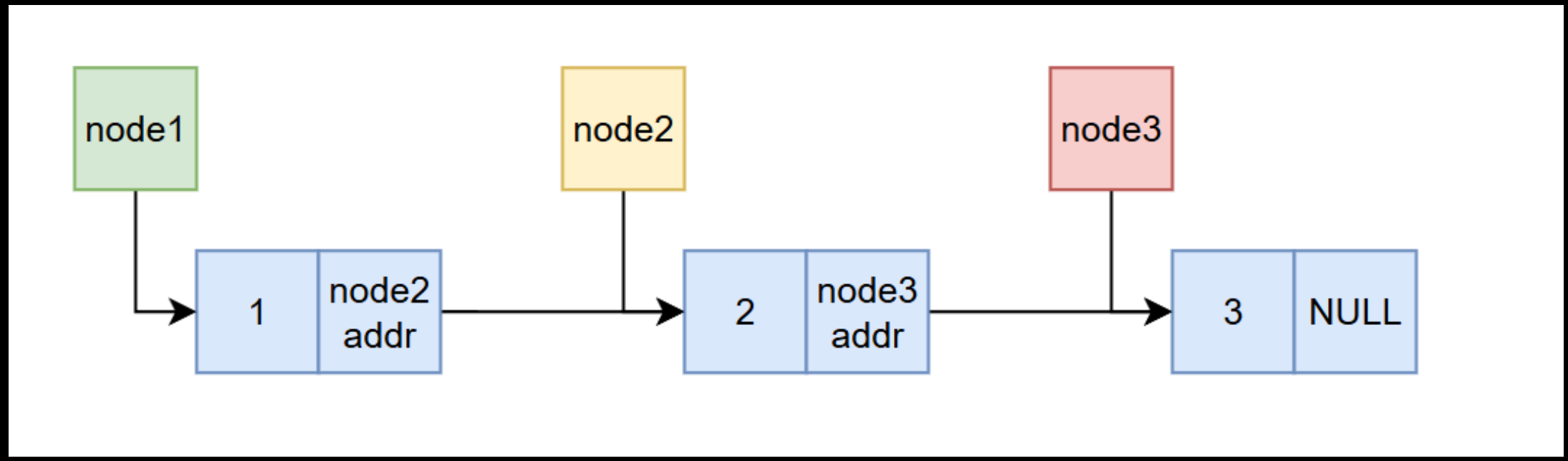
Создаем узлы



Мы выделили память под три структуры node, заполнили их числами и адрес начала каждой структуры записали в указатели node1, node2 и node3 соответственно

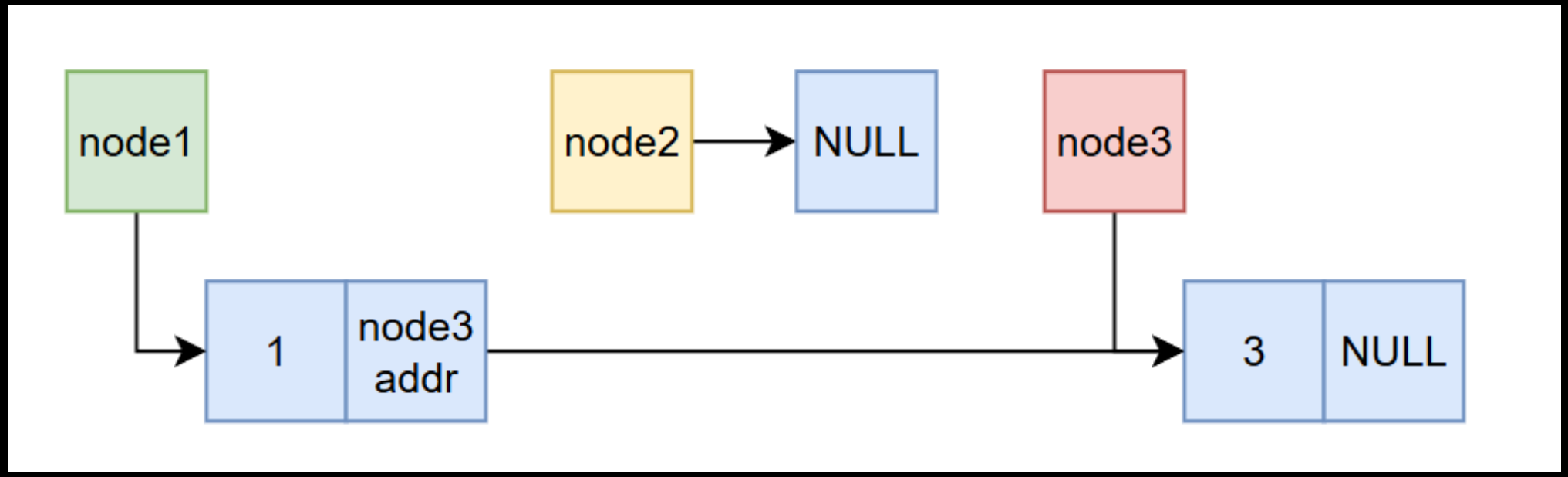
Указатели node1, node2 и node3 находятся в стеке и вероятнее всего будут лежать рядом друг с другом и друг за другом, а вот сами структуры находятся в куче и могут лежать совсем не по порядку

Связываем узлы




```
node1->next = node2;     node2->next = node3;
```

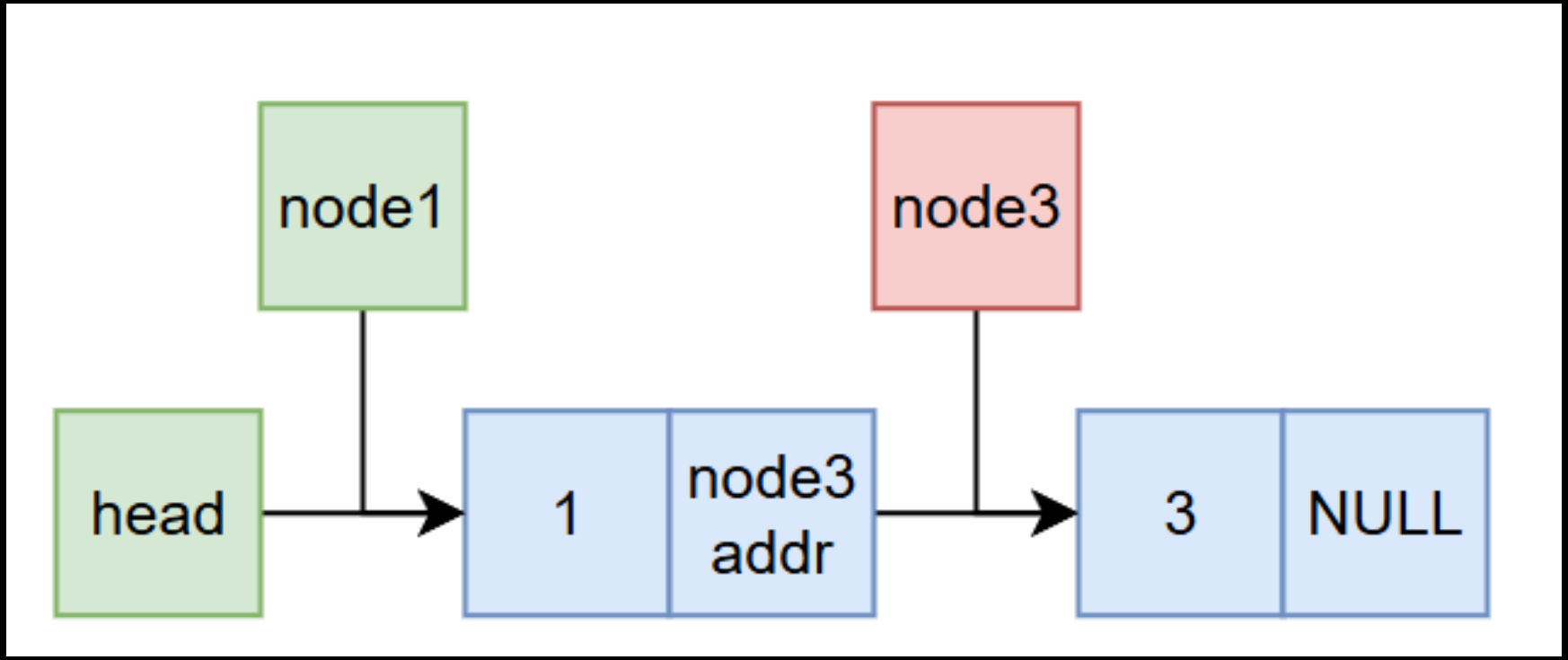
Развяжем узел 2 (delete node 2)



В создании списка таким образом есть нюанс: под каждый узел мы создавали свою переменную-указатель со своим именем, а это не серъезно если мы захотим расширяться. В реальности пишут отдельные функции для добавления узла в список, например в конец, начало, между двумя любыми (в середину).

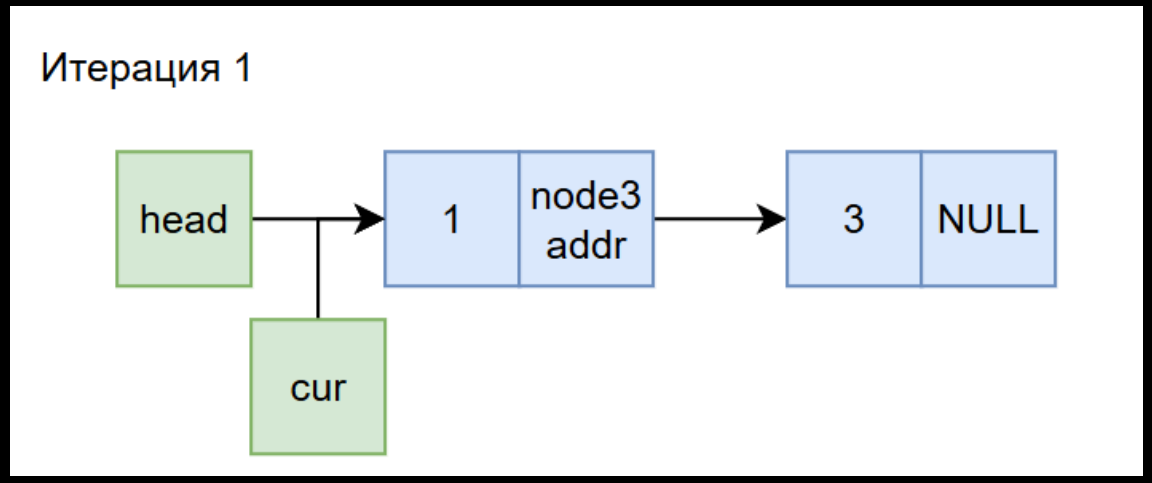
Добавим голову к списку

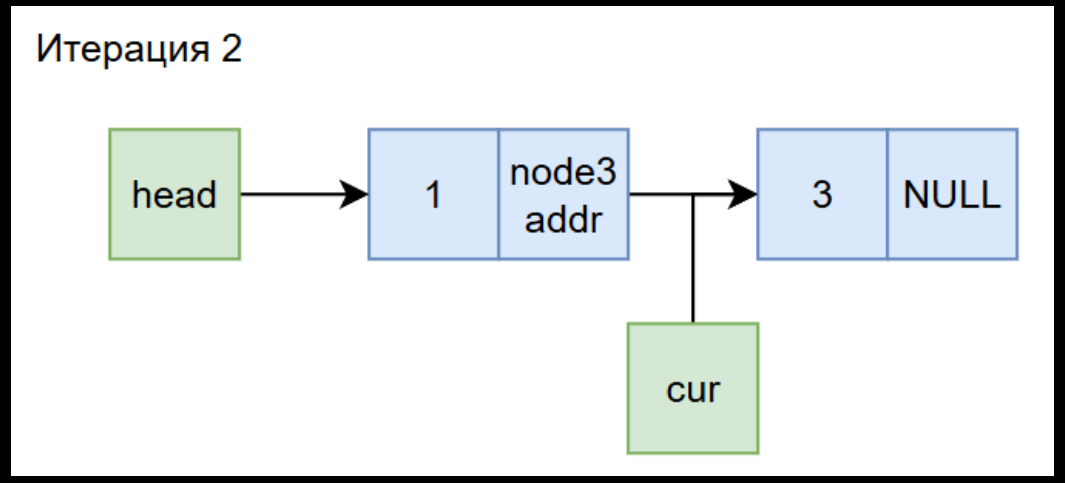
```
node_t* head = NULL; // Список пустой
head = node1; // привяжемся к списку 1 -> 3
```

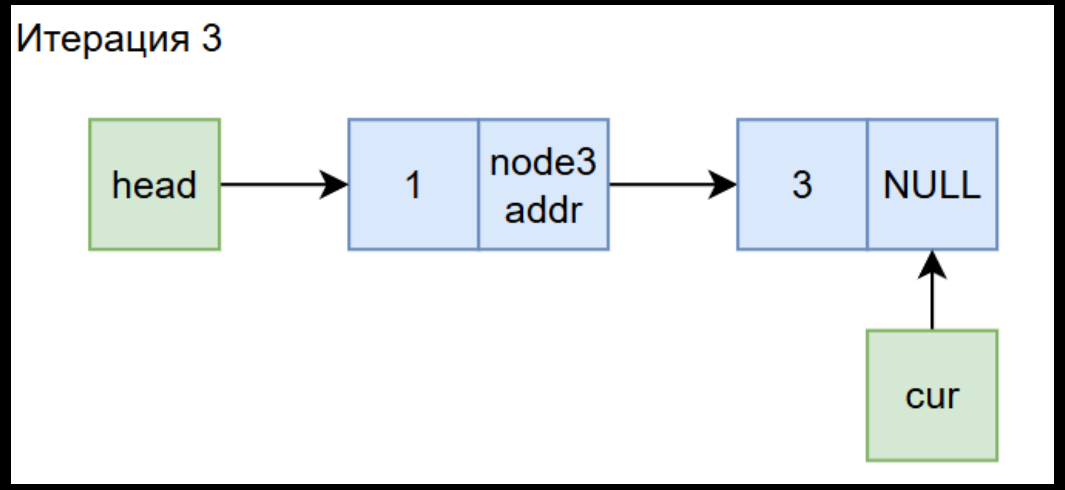


Проход по списку и его вывод

- Пока текущий узел не равен NULL:
  - Печатаем на экран данные
  - Приравниваем текущий указатель на следующий
- Список закончился







Длина списка

```
// Возвращает длину списка (int)
int lengthList(node_t* head);
```

- Длина списка = 0
- Пока текущий узел не равен NULL:
  - Длина списка + 1
  - Приравниваем текущий указатель на следующий
- Возвращаем длину

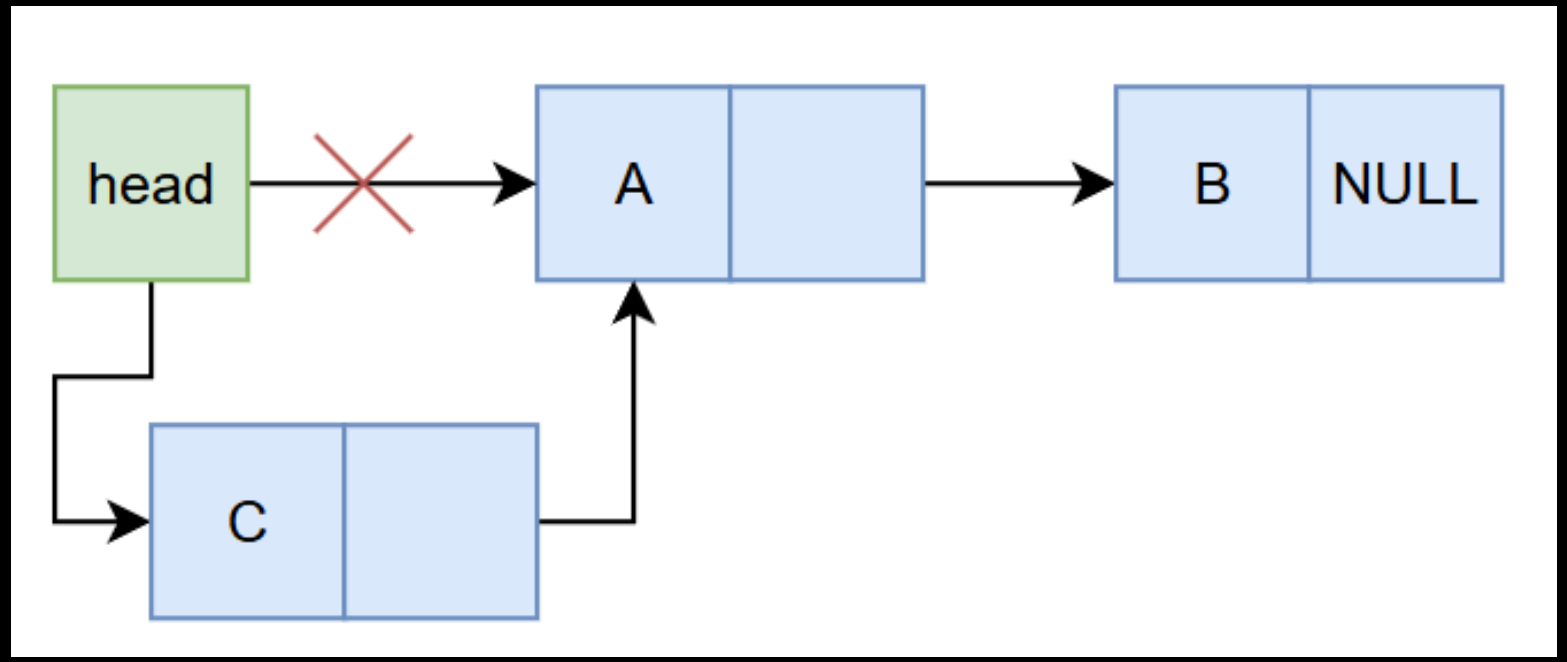
Поиск в списке

```
// Функция поиска, проходим по списку,
// если элемент в списке вернем true иначе false
bool searchList(node_t* head, int target);
```

- Пока текущий узел не равен NULL:
  - Если data = искомой
    - Возвращаем true (искомый элемент есть в списке)
  - Приравниваем текущий указатель на следующий

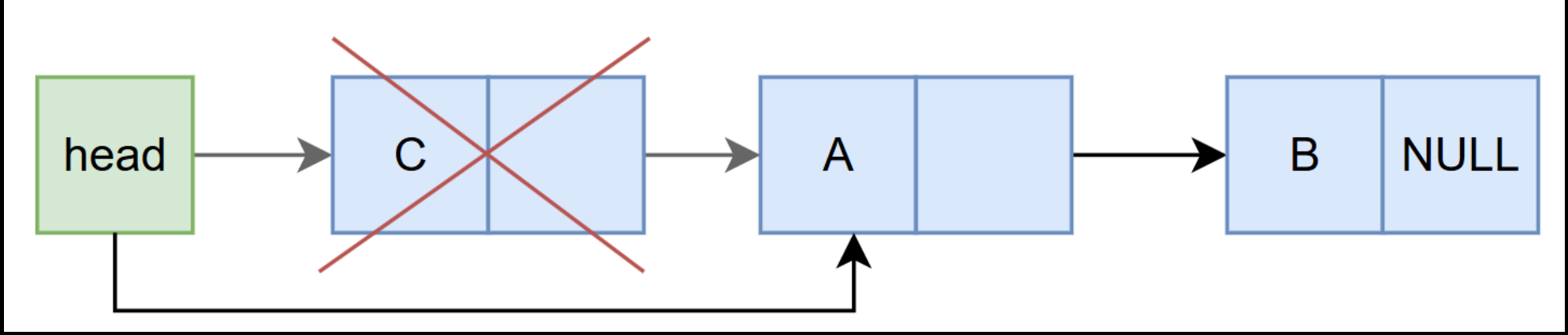
Добавление элемента в начало

- Создать новый узел с заданным значением.
- Установить следующий указатель нового узла на текущий head.
- head должен указывать на новый узел.
- Вернуть новую голову списка.



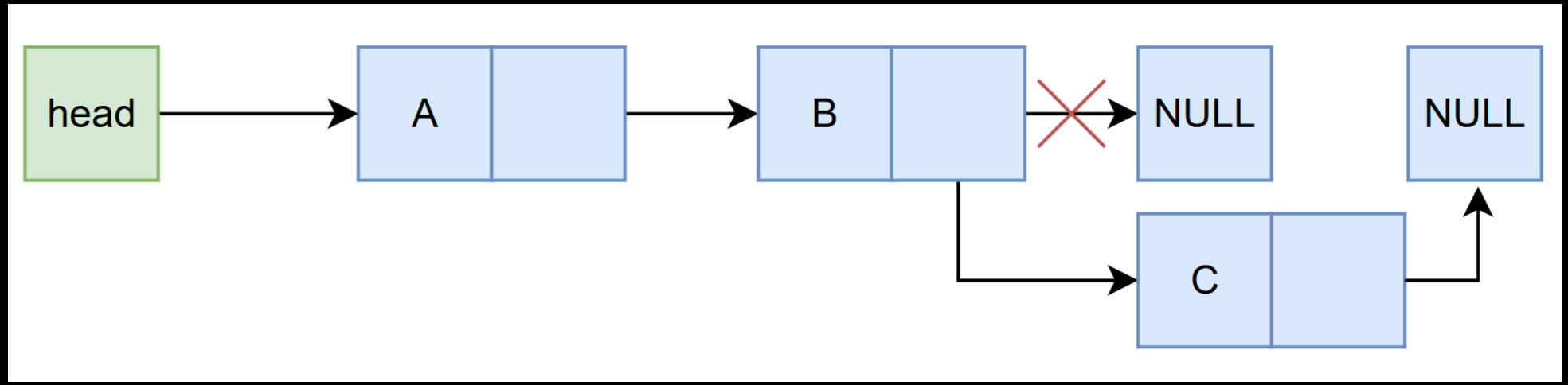
Удаление первого узла

- Проверить, имеет ли голова значение NULL
  - Если да, то вернуть NULL (список пуст)
- Сохранить текущий head во временной переменной
- Переместить указатель головы на следующий узел
- Удалить временный узел
- Вернуть новый head связанного списка



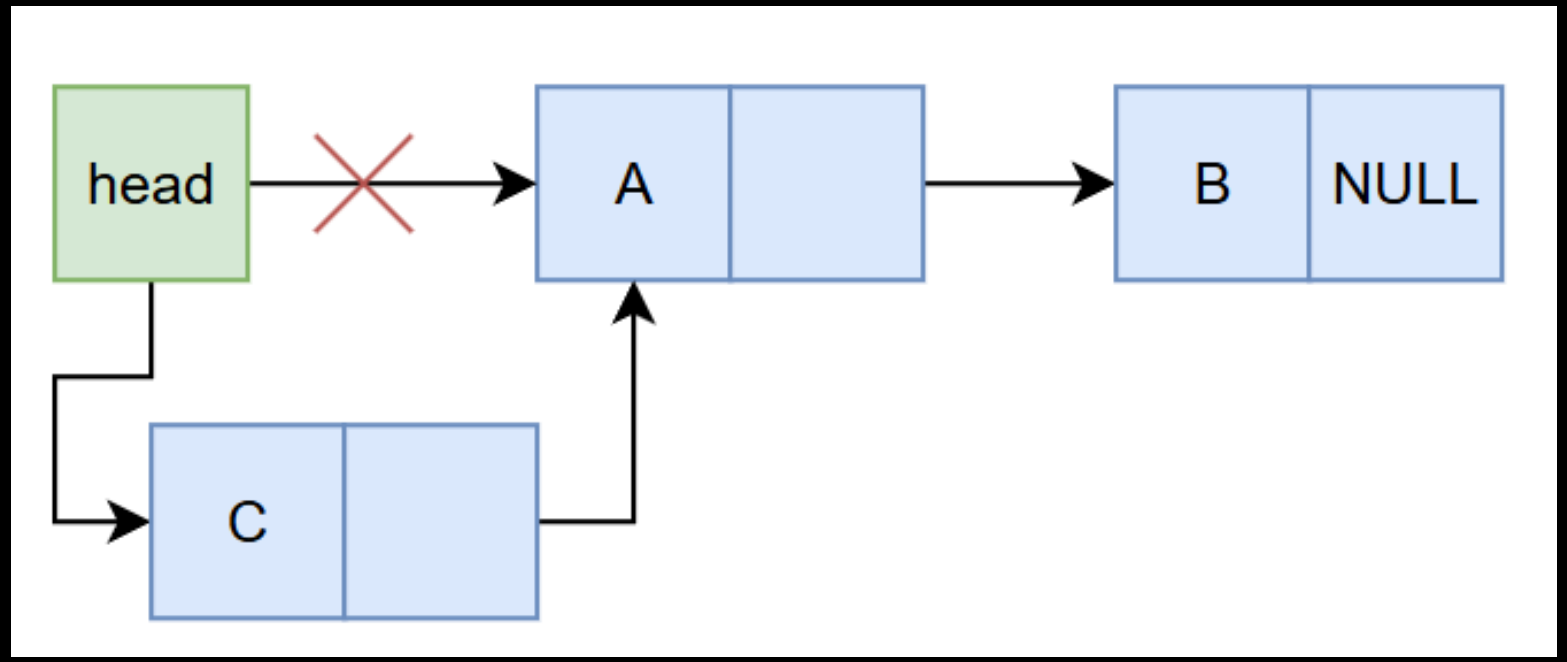
Добавление элемента в конец

- Создать новый узел с заданным значением.
- Проверить пуст ли список:
  - Если да, сделать новый узел головой и вернуть
- Перемещаться по списку до тех пор, пока не будет достигнут последний узел
- Связать новый узел с текущим последним узлом, установив указатель следующего узла у последнего узла на новый узел



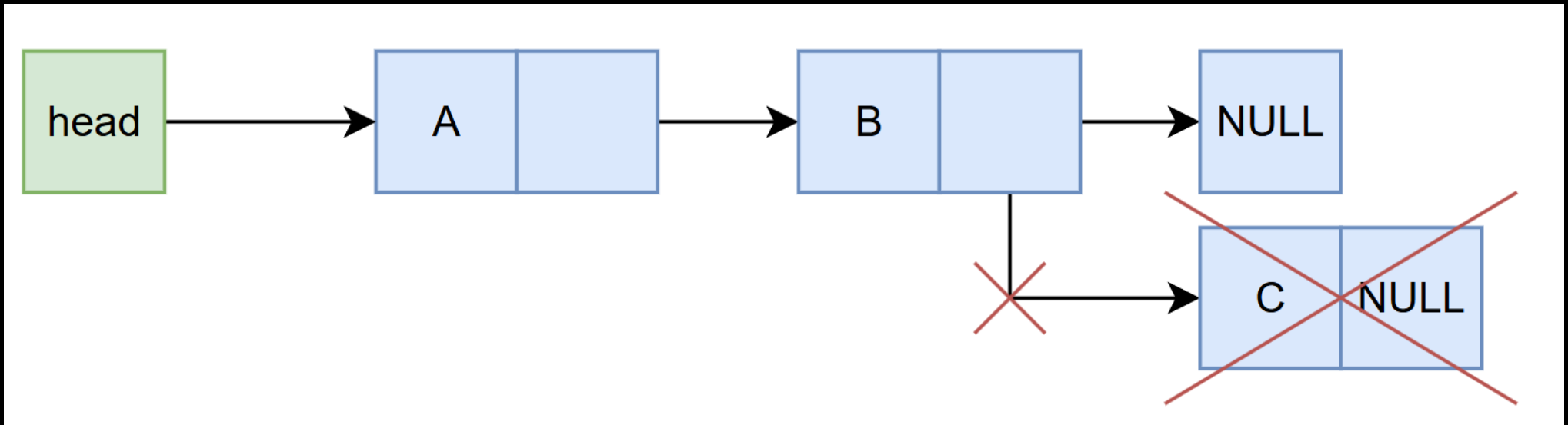
Добавление элемента в начало списка

- Создать новый узел с заданным значением.
- Установить следующий указатель нового узла на текущий head.
- head должен указывать на новый узел.
- Вернуть новую голову списка.



Удаление последнего узла

- Проверить пуст ли список (head = NULL)
  - Если да, вернуть NULL (список пуст)
- Проверить, равен ли следующий узел от головы NULL (только один узел в списке - голова)
  - Если да, удалить head и вернуть NULL
- Пройти по списку, чтобы найти предпоследний узел (второй с конца)
- Удалить последний узел (узел после предпоследнего)
- Установить следующий указатель предпоследнего узла в NULL
- Возвращаем head списка



Вставка в середину и удаление из середины

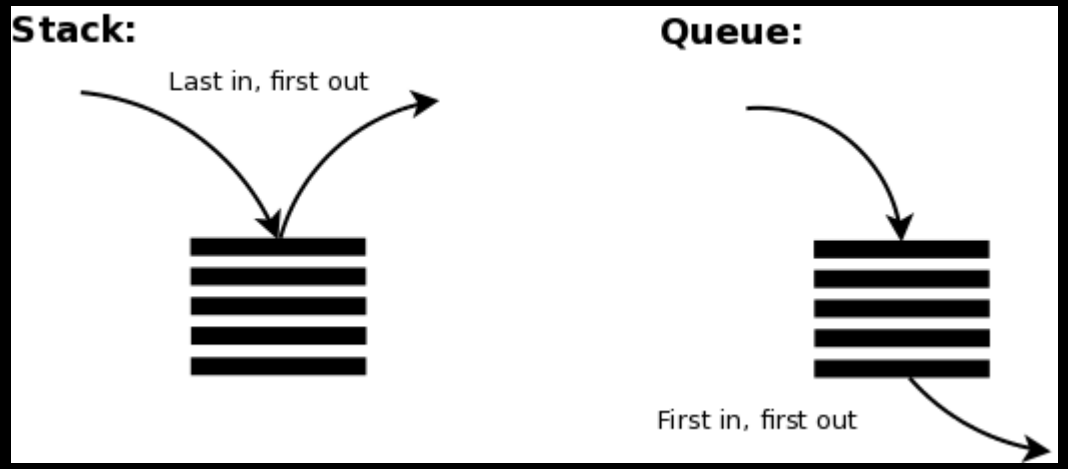
Здесь я уже устал рисовать и расписывать - поэтому самостоятельно.
Что для этого нужно сделать?
Понять все примеры выше с поиском и вставкой/удалением, а также не забывать вначале проверку на пустой список, чтоб не схватить сегу (segfault)

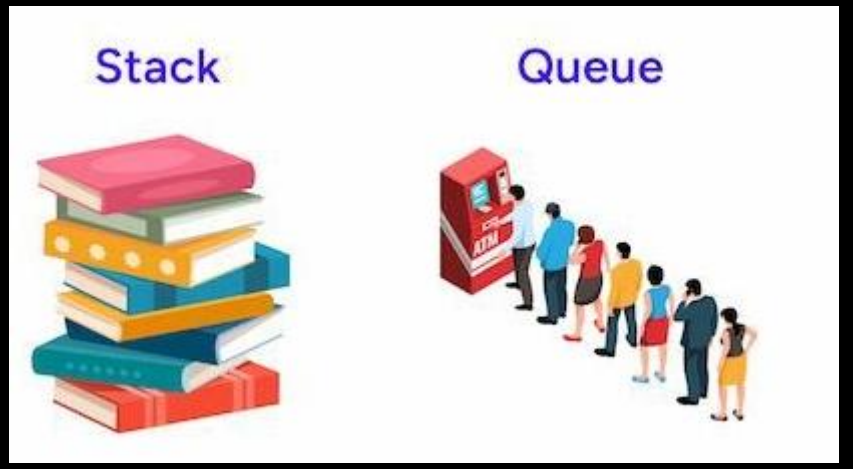
Стек и Очередь

Стек (англ. stack - стопка) - еще одна из простейших структур данных. Для описания стека часто используется аббревиатура LIFO (Last In, First Out), подчёркивающая, что элемент, попавший в стек последним, первым будет из него извлечён. Классический стек поддерживает всего лишь три операции:

- Добавить элемент в стек push() (Сложность: O(1))
- Извлечь элемент из стека pop() (Сложность: O(1))
- Проверить, пуст ли стек isEmpty() (Сложность: O(1))

Очередь (англ. queue) поддерживает тот же набор операций, что и стек, но имеет противоположную семантику. Для описания очереди используется аббревиатура FIFO (First In, First Out), так как первым из очереди извлекается элемент, раньше всех в неё добавленный.



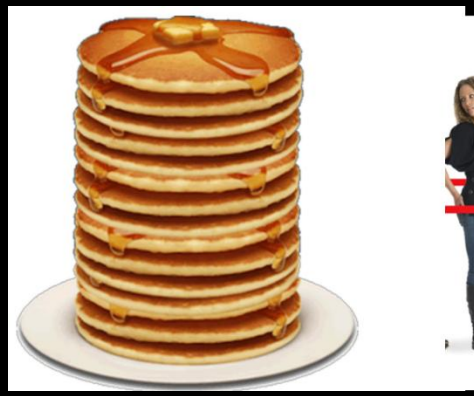


Примеры из реальной жизни

- Стеки используются для кнопок отмены в различных программах. Самые последние изменения помещаются в стек. Даже кнопка «Назад» в браузере работает с помощью стека, в который помещаются все недавно посещенные веб-страницы

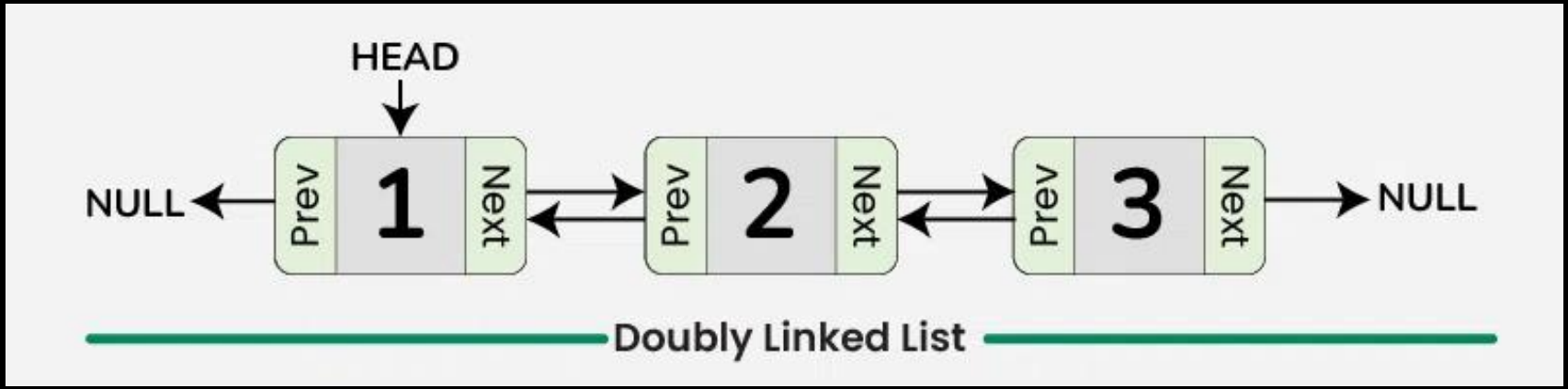
- Очереди: Представим, что у вас есть веб-сайт, который предоставляет файлы тысячам пользователей. Вы не можете обслуживать все запросы одновременно, вы можете обрабатывать только, например, 100 за раз. Справедливой политикой будет подача в порядке очереди: обслуживайте 100 человек за раз в порядке их поступления.

- Аналогично в многозадачной операционной системе ЦП не может выполнять все задания одновременно, поэтому задания необходимо группировать в пакеты, а затем планировать в соответствии с некоторой политикой. Опять же, очередь может быть подходящим вариантом в этом случае



Двусвязный список

```
node_t {
    node_t *prev; // указатель на предыдущий узел
    int data; // любые данные
    node_t *next; // указатель на следующий узел
};
```



Двусвязный список

Плюсы:

- Эффективный обход в обоих направлениях: двусвязные списки обеспечивают эффективный обход списка в обоих направлениях.
- Также может использоваться для реализации стека или очереди: двусвязные списки можно использовать для реализации как стеков, так и очередей, которые являются распространенными структурами данных, используемыми в программировании.

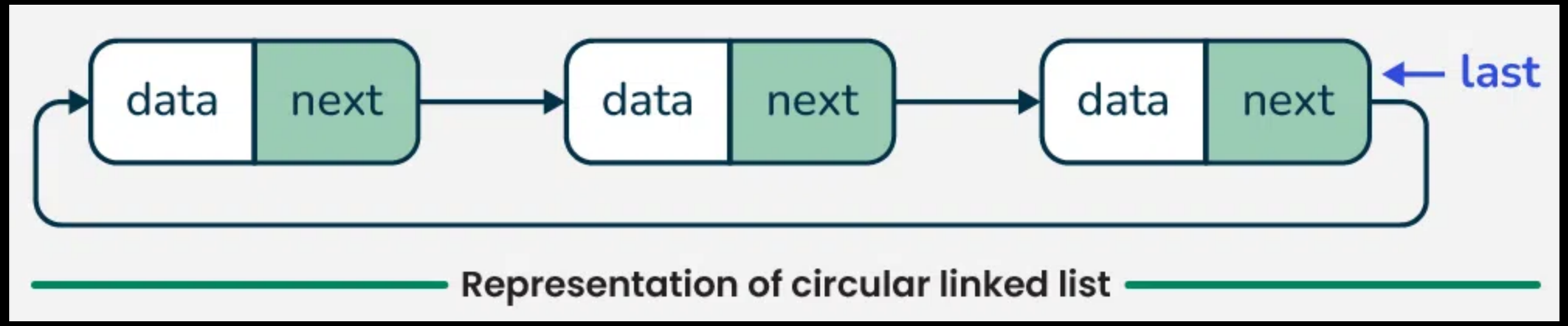
Минусы:

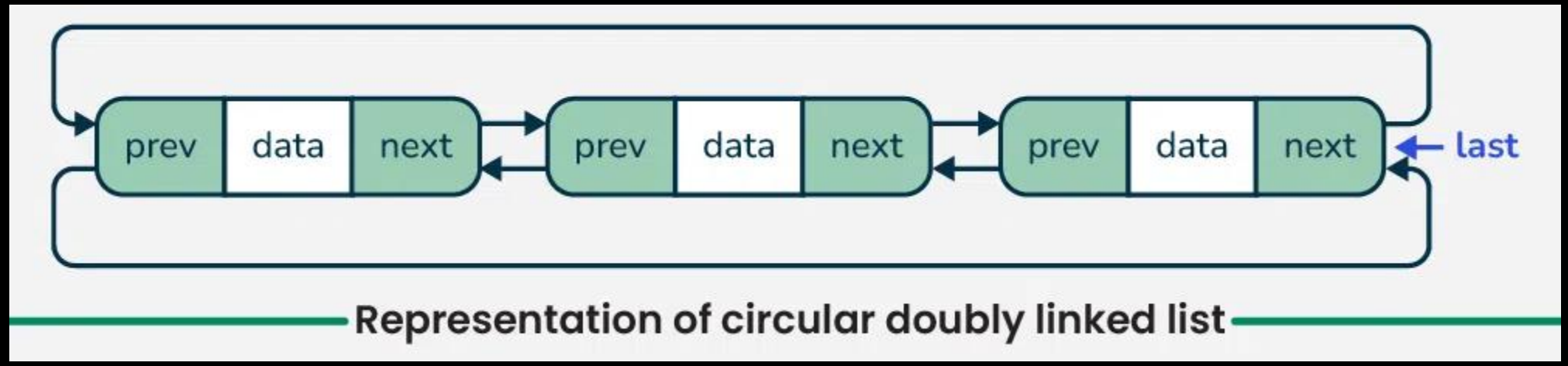
- Более сложный, чем односвязные списки: двусвязные списки более сложны, чем односвязные, поскольку для каждого узла требуются дополнительные указатели.
- Больше накладных расходов на память: двусвязные списки требуют больше накладных расходов на память, чем односвязные списки, поскольку каждый узел хранит два указателя вместо одного.

Примеры применения:

- Реализация функций отмены и повтора действий в текстовых редакторах.
- Реализация кэша, где требуется быстрая вставка и удаление элементов.
- Управление историей браузера для навигации между посещенными страницами.
- Приложения для музыки для управления списками воспроизведения и эффективной навигации по песням.
- Реализация структур данных, таких как Дек (Deque - двусторонняя очередь), для эффективной вставки и удаления на обоих концах.

Циклический список



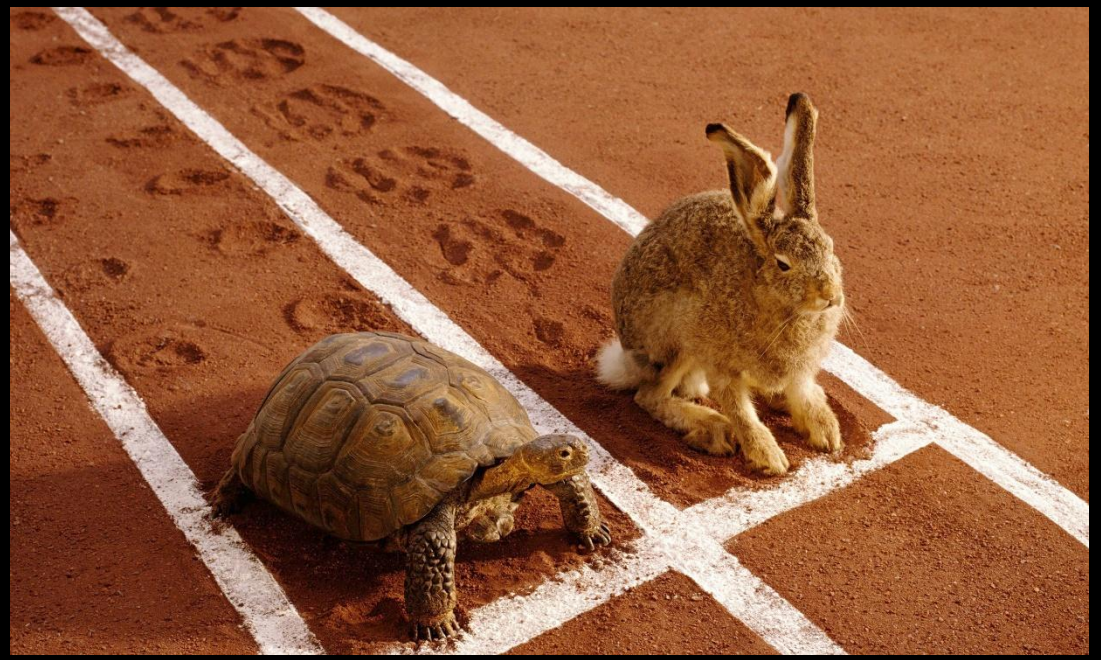


Поиск цикла в списке

Как определить - список циклический или нет?

Воспользуемся алгоритмом Флойда "Черепаха и заяц" (или бывает кролик вместо зайца). Пусть за одну итерацию первый указатель (черепаха) переходит к следующему элементу списка, а второй указатель (заяц) на два элемента вперед. Тогда, если эти два указателя встретятся, то цикл найден, если дошли до конца списка, то цикла нет.

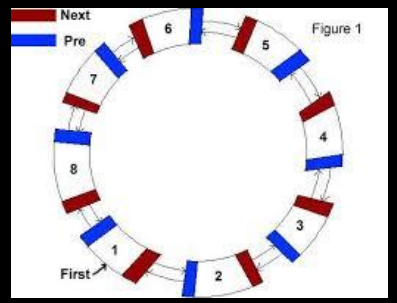
Алгоритм с маркировкой (см. в гугле)

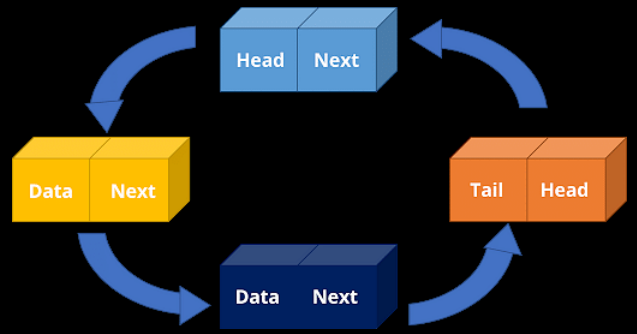


+/- Circular LL

Плюсы:

- В кольцевом связанном списке последний узел указывает на первый узел, т.е. нет ссылок на NULL.
- Мы можем обойти список из любого узла и вернуться к нему без необходимости перезапуска с головы, что полезно в приложениях, требующих кольцевой итерации.
- Кольцевые связанные списки могут легко реализовывать кольцевые очереди, где последний элемент соединяется с первым, что позволяет эффективно управлять ресурсами.
- В кольцевом связанном списке каждый узел имеет ссылку на следующий узел в последовательности. Хотя он не имеет прямой ссылки на предыдущий узел, как в двусвязном списке, мы все равно можем найти предыдущий узел, обойдя список.





Минусы:

- Обход кольцевого связанного списка без четкого условия остановки может привести к бесконечным циклам
- Сложна (hard)

Применение: кольцевых связанных списков

- Он используется для разделения ресурсов между разными пользователями, как правило, с помощью механизма планирования Round-Robin
- В многопользовательских играх кольцевой связанный список может использоваться для переключения между игроками. После хода последнего игрока список циклически возвращается к первому игроку.
- В медиаплеерах кольцевые связанные списки могут управлять плейлистами, что позволяет пользователям непрерывно перебирать песни.

Выводы по спискам

Связанный список (Linked List) — это линейная структура данных, которая используется для хранения набора данных с помощью узлов

- Последовательные элементы соединены указателями/ссылками.
- Последний узел связанного списка указывает на null.
- Точка входа связанного списка называется головой.
- Обычные вариации связанных списков — это Singly, Doubly, Singly Circular и Doubly Circular.

Всегда необходимо учитывать плюсы и минусы структур данных при их выборе. Например, если вам нужно быстрое время доступа, массивы могут быть лучшим выбором, но если вам нужно часто вставлять или удалять элементы, списки могут быть лучшим выбором

Преимущества связанных списков

- Вставка и удаление в любой точке связанного списка занимает O(1) времени. Тогда как в структуре данных массива вставка/удаление в середине занимает O(n) времени.

- Эта структура данных проста и может также использоваться для реализации стека, очередей и других абстрактных структур данных. Большинство языковых библиотек используют связанный список внутренне для реализации этих структур данных.

- Связанный список может оказаться более эффективным по сравнению с массивами в случаях, когда мы не можем заранее угадать максимально допустимое количество элементов. Даже с динамическими массивами, такими как vector в C++ или list в Python или ArrayList в Java внутренняя работа включает в себя освобождение всей памяти и выделение большего фрагмента, когда вставки происходят за пределами текущей емкости.

Недостатки связанных списков:

- Медленное время доступа: доступ к элементам в связанном списке может быть медленным, так как вам нужно пройти по связанному списку, чтобы найти нужный элемент, что является операцией O(n)

- Указатели или ссылки: использование указателей и ссылок для доступа к следующему узлу вызывают сложности для не понимающих что происходит

- Расход памяти: Связанные списки занимают больше памяти, т.к. каждый узел в списке требует доп. памяти для хранения указателя на следующий узел

- Неэффективность кэширования: Связанные списки неэффективны в отношении кэширования, так как память не является непрерывной. Это означает, что при обходе связанного списка вы вряд ли получите нужные вам данные в кэше, что приведет к промахам кэша и низкой производительности.

Применения связанных списков в реальном мире

- Список песен в музыкальном проигрывателе связан с предыдущей и следующей песнями.
- В веб-браузере URL-адреса предыдущей и следующей веб-страницы могут быть связаны с помощью кнопок «предыдущая» и «следующая» (двухсвязный список)
- В средстве просмотра изображений предыдущее и следующее изображения могут быть связаны с помощью кнопок «предыдущая» и «следующая» (двухсвязный список)
- Циклические связанные списки могут использоваться для реализации действий в циклическом режиме, когда мы переходим к каждому элементу по одному через время t
- Связанные списки предпочтительнее массивов для реализации структур данных Queue, Stack и Deque из-за быстрых удалений (или вставок) из начала/конца связанных списков.

https://github.com/kruffka/C-Programming