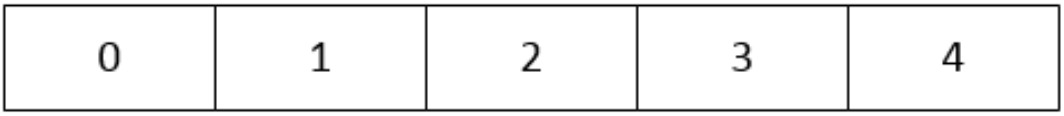
Линейные структуры

Массив

Линейный массив, также известный как одномерный массив, представляет собой структуру данных, которая состоит из элементов, расположенных в отдельных ячейках памяти и доступ к которым осуществляется посредством индексирования. Каждый элемент массива занимает последовательное место в памяти и имеет свой уникальный индекс, начиная с нуля.

Линейные массивы широко используются для решения различных задач в программировании, таких как хранение и обработка данных, сортировка, поиск и манипулирование элементами. Операции над линейными массивами выполняются эффективно, благодаря прямому доступу к элементам по индексу.



Преимущества линейного массива

- Простота в использовании: Линейный массив прост в реализации и использовании, так как элементы хранятся последовательно и имеют простую индексацию.
- Быстрый доступ к элементам: Так как элементы массива хранятся последовательно, доступ к любому элементу осуществляется за константное время O(1) путем указания его индекса.
- Удобство при выполнении операций: Линейный массив позволяет легко выполнять различные операции, такие как добавление элементов в конец массива, удаление элементов из массива и доступ к элементам по индексу.
- Компактность: Линейный массив занимает минимальное пространство в памяти при хранении элементов, так как нет необходимости в дополнительных указателях или ссылках между элементами.
- Скорость выполнения операций поиска и сортировки: В линейном массиве операции поиска и сортировки являются эффективными, особенно если массив отсортирован и используется оптимальный алгоритм поиска или сортировки.

Недостатки:

Нельзя удалять или добавлять элементы, у таких массивов фиксированная длина

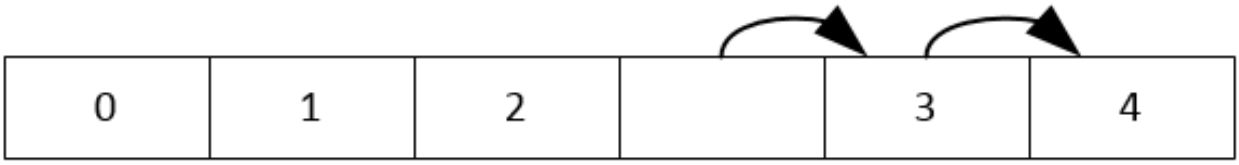
Динамический массив

Динамический массив — это массив, который может менять свой размер. Основные языки программирования в своих стандартных библиотеках поддерживают динамические массивы. В C++ это vector. В Java это ArrayList. В C# это List. Все они являются динамическими массивами. В своей сути динамический массив — это простой массив, однако имеющий ещё два дополнительных блока данных. В них хранятся действительный размер простого массива и объём данных, который может на самом деле храниться в простом массиве. Динамический массив может выглядеть примерно так:

```
// Внутреннее устройство класса динамического массива
sometype *internalArray;
unsigned int currentLength;
unsigned int maxCapacity;
```

Элемент internalArray указывает на динамически размещаемый буфер. Действительный массив буфера хранится в maxCapacity. Количество использовуемых элементов задаётся currentLength.
Добавление к динамическому массиву

При добавлении объекта к динамическому массиву происходит несколько действий. Класс массива проверяет, достаточно ли в нём места. Если currentLength < maxCapacity, то в массиве есть место для добавления. Если места недостаточно, то размещается больший внутренний массив, и всё копируется в новый внутренний массив. Значение maxCapacity увеличивается до нового расширенного значения. Если места достаточно, то добавляется новый элемент. Каждый элемент после точки вставки должен быть скопирован на соседнее место во внутреннем массиве, и после завершения копирования пустота заполняется новым объектом, а значение currentLength увеличивается на единицу.

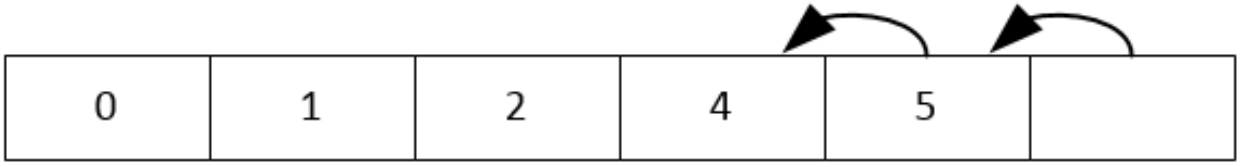


Поскольку необходимо перемещать каждый объект после точки вставки, то наилучшим случаем будет добавление элемента к концу. При этом нужно перемещать ноль элементов (однако внутренний массив всё равно требует расширения). Динамический массив лучше всего работает при добавлении элемента в конец, а не в середину.

При добавлении объекта к динамическому массиву каждый объект может переместиться в памяти. В таких языках, как C и C++, добавление к динамическому массиву означает, что ВСЕ указатели на объекты массива становятся недействительными.

Удаление из динамического массива

Удаление объектов требует меньше работы, чем добавление. Во-первых, уничтожается сам объект. Во-вторых, каждый объект после этой точки сдвигается на один элемент. Наконец, currentLength уменьшается на единицу.



Как и при добавлении к концу массива, удаление из конца массива является наилучшим случаем, потому что при этом нужно перемещать ноль объектов. Также стоит заметить, что нам не нужно изменять размер внутреннего массива, чтобы сделать его меньше. Выделенное место может оставаться таким же, на случай, если мы позже будем добавлять объекты.

Удаление объекта из динамического массива приводит к смещению в памяти всего после удалённого элемента. В таких языках, как C и C++, удаление из динамического массива означает, что указатели на всё после удалённого массива становятся недействительными.

Недостатки динамических массивов

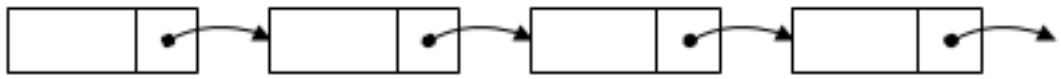
Допустим, массив очень велик, а вам нужно часто добавлять и удалять объекты. При этом объекты могут часто копироваться в другие места, а многие указатели становиться недействительными.

Связные списки

Массив — это непрерывный блок памяти, и каждый элемент его расположен после другого. Связанный список — это цепочка объектов. Связанные списки тоже присутствуют в стандартных библиотеках основных языков программирования. В C++ они называются *list*. В Java и C# это *LinkedList*. Связанный список состоит из серии узлов. Каждый узел выглядит примерно так:

```
// Узел связанного списка
sometype data;
Node* next;
```

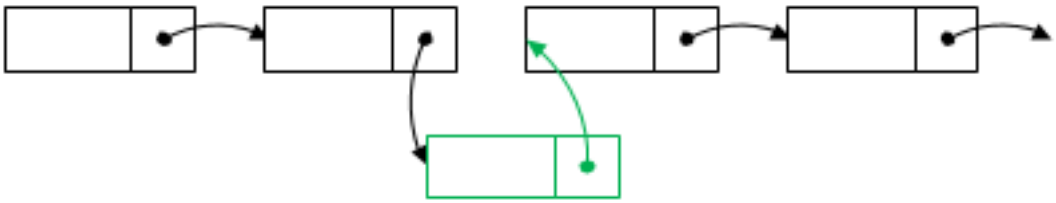
Он создаёт структуру такого типа:



Каждый узел соединяется со следующим.

Добавление к связанному списку

Добавление объекта к связанному списку начинается с создания нового узла. Данные копируются внутрь узла. Затем находится точка вставки. Указатель нового узла на следующий объект изменяется так, чтобы указывать на следующий за ним узел. Наконец, узел перед новым узлом изменяет свой указатель, чтобы указывать на новый узел.



Удаление из связанного списка

При удалении объекта из связанного списка находится узел перед удаляемым узлом. Он изменяется таким образом, чтобы указывать на следующий после удалённого объекта узел. После этого удалённый объект можно безопасно стереть.

Преимущества связанного списка

Самое большое преимущество связанного списка заключается в добавлении и удалении объектов из списка. Внесение изменений в середину списка выполняется очень быстро. Помните, что динамический массив теоретически мог вызывать смещение каждого элемента, а связанный список сохраняет каждый другой объект на своём месте.

Недостатки связанного списка

Вспомните, что динамический массив — это непрерывный блок памяти.

Если вам нужно получить пятисотый элемент массива, то достаточно просто посмотреть на 500 «мест» вперёд. В связанном списке память соединена в цепочку. Если вам нужно найти пятисотый элемент, то придётся начинать с начала цепочки и следовать по её указателю к следующему элементу, потом к следующему, и так далее, повторяя пятьсот раз.

Произвольный доступ к связанному списку выполняется очень медленно.

Ещё один серьёзный недостаток связанного списка не особо очевиден. Каждому узлу необходимо небольшое дополнительное место. Сколько ему нужно места? Можно подумать, что для него нужен только размер указателя, но это не совсем так. При динамическом создании объекта всегда существует небольшой запас. Некоторые языки программирования, например, C++, работают со страницами памяти. Обычно страница занимает 4 килобайта. При использовании операторы добавления и удаления, размещается целая страница памяти, даже если вам нужно использовать только один байт.

В Java и C# всё устроено немного иначе, в них есть специальные правила для небольших объектов. Для этих языков не требуется вся 4-килобайтная страница памяти, но всё равно у них есть небольшой запас. Если вы используете стандартные библиотеки, то о втором недостатке волноваться не нужно. Они написаны таким образом, чтобы минимизировать занимаемое впустую место.

Заключение

Эти три типа (массив, динамический массив и связанный список) создают основу почти для всех более сложных контейнеров данных. При учёбе в колледже одним из первых заданий в изучении структур данных становится собственная реализация классов динамического массива и связанного списка.

Эти структуры являются в программировании фундаментальными. Не важно, какой язык вы будете изучать, для работы с данными вы будете их использовать.