数组与链表

	数组	链表
读取	O(1)	O(n)
插入	O(n)	O(1)
删除	O(n)	O(1)

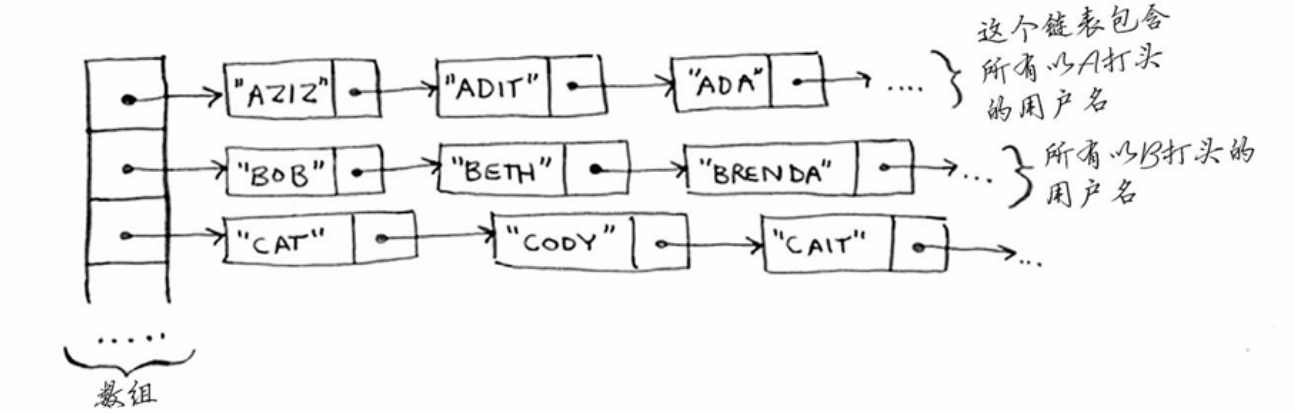
O(n):线性时间

O(1):常量时间

comment:： 链表擅长插入和删除，而数组擅长随机访问。

链表数组

> e.g.，Facebook存储用户信息时使用的既不是数组也不是链表。假设Facebook使用 的是一种混合数据：链表数组。这个数组包含26个元素，每个元素都指向一个链表。例如，该数组的第一个元素指向的链表包含所有以A打头的用户名，第二个元素指向的链表包含所有以B打头的用户名，以此类推。



> 假设Adit B在Facebook注册，而你需要将其加入前述数据结构中。因此，你访问数组的第一个元素，再访问该元素指向的链表，并将Adit B添加到这个链表末尾。 现在假设你要查找Zakhir H。因此你访问第26个元素，再在它指向的链表（该链表包含所有以z打头的用户名）中查找Zakhir H。

选择排序

C++代码实现如下:

```cpp
#include <iostream>
using namespace std;

// 选择排序函数（原地排序）
void selectionSort(double arr[], int n)
{
```

```
    for (int i = 0; i < n - 1; i++)
    {
        // 找到未排序部分的最小元素索引
        int min_idx = i;
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
            {
                min_idx = j;
            }
        }
        // 交换找到的最小元素到当前位置
        double temp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = temp;
    }
}

int main()
{
    cout << "请输入待排序数组中总的元素个数: ";
    int n;
    cin >> n;

    if (n <= 0)
    {
        cout << "无效的数组大小!" << endl;
        return 1;
    }

    // 动态分配内存
    double* numbers = new double[n];

    cout << "请依次输入待排序的元素的值(用空格或回车分隔):" << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> numbers[i];
    }

    // 执行选择排序
    selectionSort(numbers, n);

    // 输出排序结果
    cout << "排序结果: ";
    for (int i = 0; i < n; i++)
    {
        cout << numbers[i];
        if (i < n - 1) cout << " ";
    }
    cout << endl;

    // 释放内存
    delete[] numbers;
```

```
    return 0;
}
```

上面展示的代码采用的是原地排序,下面给出另外一种思路(不过这个程序在内存释放上存在一些问题)

```
#include <iostream>
using namespace std;

// 前置声明
template<typename T>
T pop(T*& arr, int& size, int index);
template<typename T>
void append(T*& arr, int& size, T element);

// 查找最小元素的索引
int find_smallest(double* arr, int size)
{
    if (size == 0)
        return -1; // 处理空数组

    double smallest = arr[0];
    int smallest_index = 0;
    for (int i = 1; i < size; i++)
    {
        if (arr[i] < smallest)
        {
            smallest = arr[i];
            smallest_index = i;
        }
    }
    return smallest_index;
}

// 选择排序
void selection_Sort(double* arr, int size)
{
    double* result = nullptr;   // 初始化为空指针
    int result_size = 0;        // 结果数组大小

    int current_size = size;    // 保存当前数组大小

    while (current_size > 0)
    {
        int smallest_index = find_smallest(arr, current_size);
        if (smallest_index == -1)
            break; // 安全检查

        double tmp = pop(arr, current_size, smallest_index);
        append(result, result_size, tmp);
    }

    // 输出排序结果
```

```
    for (int i = 0; i < result_size; i++)
    {
        cout << result[i];
        if (i < result_size - 1)
            cout << " ";
    }
    cout << endl;

    delete[] result; // 释放结果数组
}

// 弹出指定索引的元素并缩小数组
template<typename T>
T pop(T*& arr, int& size, int index)
{
    T target = arr[index];
    T* tmp = new T[size - 1];

    // 复制索引前的元素
    for (int i = 0; i < index; i++)
    {
        tmp[i] = arr[i];
    }

    // 复制索引后的元素
    for (int i = index; i < size - 1; i++)
    {
        tmp[i] = arr[i + 1];
    }

    delete[] arr; // 释放原数组
    arr = tmp;    // 更新指针
    size--;       // 减小大小

    return target;
}

// 向数组末尾添加元素
template<typename T>
void append(T*& arr, int& size, T element)
{
    T* tmp = new T[size + 1];

    // 复制原数组内容
    for (int i = 0; i < size; i++)
    {
        tmp[i] = arr[i];
    }

    // 添加新元素
    tmp[size] = element;

    // 释放原数组（如果是首次添加，arr 可能是 nullptr）
    if (arr)
```

```
    {
        delete[] arr;
    }

    arr = tmp; // 更新指针
    size++;    // 增加大小
}

int main()
{
    cout << "请输入待排序数组中总的元素个数: ";
    int n;
    cin >> n;


    cout << "请依次输入待排序的元素的值: ";
    double* num = new double[n];
    for (int i = 0; i < n; i++)
    {
        cin >> num[i];
    }

    selection_Sort(num, n);

    delete[] num;
    return 0;
}
```

python代码实现如下:

```
def findSmallest(arr):
    smallest = arr[0]
    smallest_index = 0
    for i in range(1,len(arr)):
        if arr[i]<smallest:
            smallest = arr[i]
            smallest_index = i
    return smallest_index

def selectionSort(arr):
    newArr = []
    for i in range(len(arr)):
        smallest = findSmallest(arr)
        newArr.append(arr.pop(smallest))
    #     pop(i)移除arr中索引为i的元素,并返回该元素的值
    #     append(x)将x添加到列表newArr的末尾
    return newArr
print(selectionSort([5,3,6,2,10]))
```

conclusions:

1. 计算机内存犹如一大堆抽屉。
2. 需要存储多个元素时，可使用数组或链表。
3. 数组的元素都在一起。
4. 链表的元素是分开的，其中每个元素都存储了下一个元素的地址。
5. 数组的读取速度很快。
6. 链表的插入和删除速度很快。
7. 在同一个数组中，所有元素的类型都必须相同（都为int、double等）