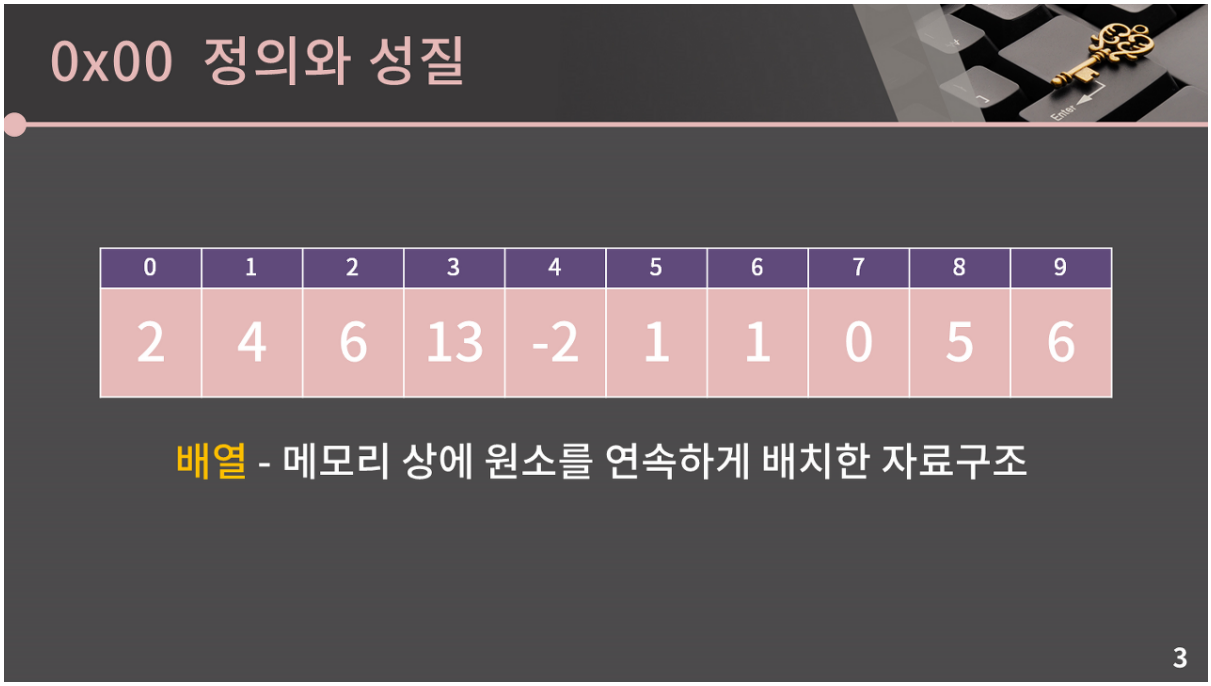
바킹독 0x03 - 배열

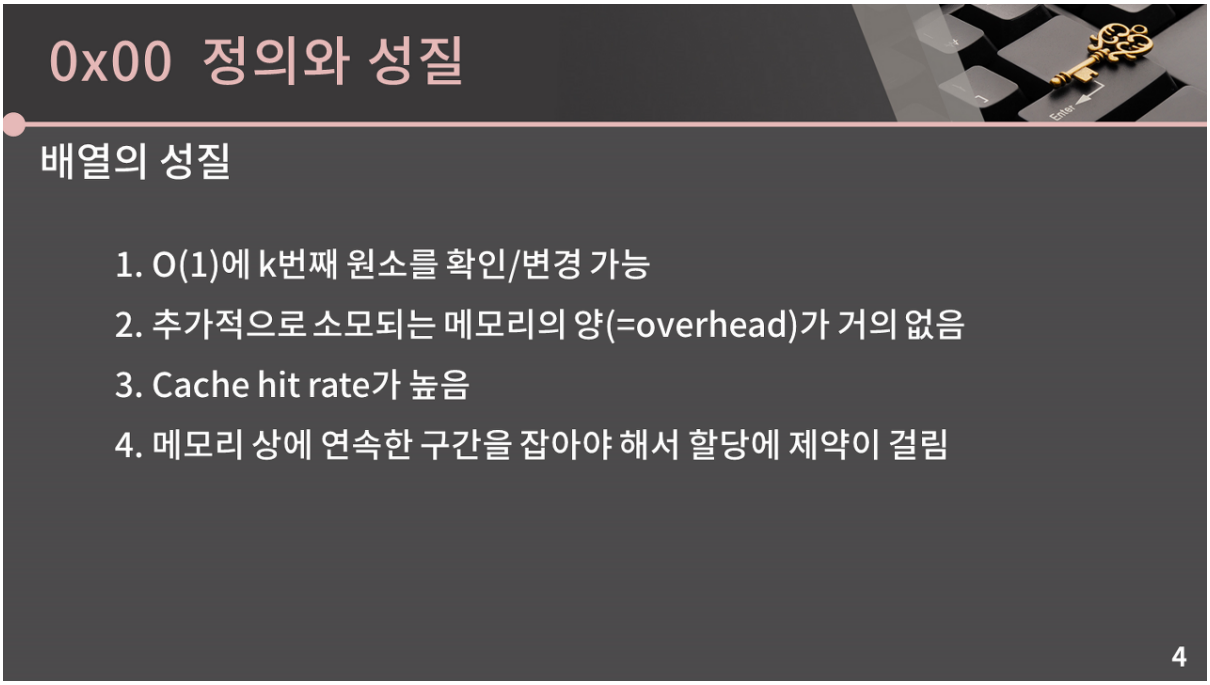
태그	
공개여부	☑
날짜	
작성일자	



배열은 메모리 상에 원소를 연속하게 배치한 자료구조입니다.

원래 C++에서는 예를 들어 int arr[10]; 으로 배열을 선언한 뒤에는 arr의 길이를 변경하는게 불가능하지만,

자료구조로써의 배열에서는 길이를 마음대로 늘리거나 줄일 수 있다고 생각하겠습니다.



- **배열은 메모리 상에 원소를 연속하게 배치한 자료구조이라서 k번째 원소의 위치를 바로 계산할 수 있게 됩니다. 시작 주소에서 k칸 만큼 오른쪽으로 가면 되기 때문입니다. 그렇기 때문에 k번째 원소를 O(1)에 확인하거나 변경할 수 있습니다.**
- 메모리는 다른 자료구조와 다르게 추가적으로 소모되는 메모리의 양이 거의 없습니다.
- 메모리 상에 데이터들이 붙어있으니까 Cache hit rate가 높습니다.
- 메모리 상에 연속한 구간을 잡아야 하니 할당에서 다소 제약이 있습니다.

특징들을 간략하게 살펴봤는데, 1번 성질은 사실 저희가 너무 당연하게 활용했던 성질이고, 2번, 3번, 4번 성질은 굳이 신경쓰지 않아도 코딩테스트를 칠 때에는 별로 상관이 없습니다.

배열의 기능과 구현

- O(1)

배열의 임의의 위치에 있는 원소를 확인/변경

배열의 끝에 원소를 추가(끝자리에 값을 쓰고 길이를 1 증가)

배열의 마지막 원소를 제거(길이를 1 줄이면 됨)

- O(N)

<배열의 크기(입력 데이터의 크기)가 N이라고 가정>

임의의 위치(끝을 제외한 모든 위치)에 있는 원소를 추가/제거

최악의 경우 (index 0에 원소를 추가/제거) N-1개의 원소를 땡겨야 하므로

시간복잡도 = O(n)

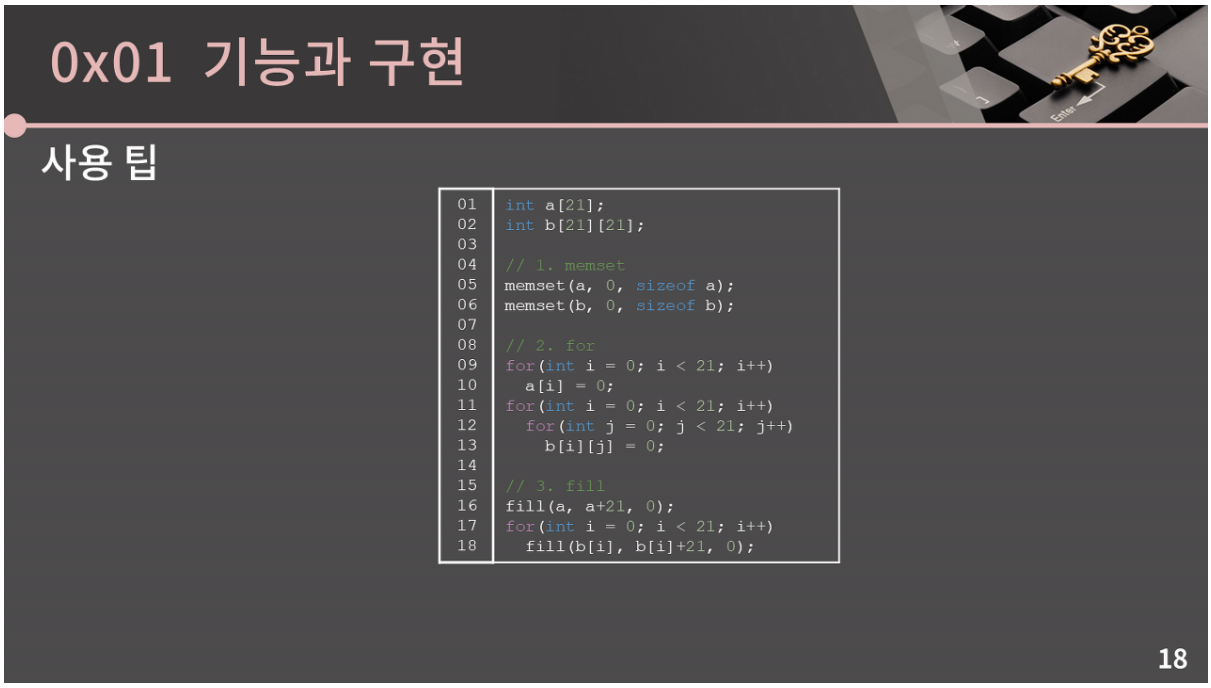
ex. 2번 원소를 지우게 되면 오른쪽에 위치한 모든 원소를 한칸씩 땡김

-> 배열의 특징인 '연속성' 생각 !

```
#include <bits/stdc++.h>
  using namespace std;

  void insert_test();
  void erase_test();
  void insert(int idx, int num, int arr[], int& len);
  void erase(int idx ,int arr[], int& len);
  void printArr(int arr[], int& len);


  int main(void) {
    insert_test();
    erase_test();
  }

  void insert_test(){
    cout << "***** insert_test *****\n";
    int arr[10] = {10, 20, 30};
    int len = 3;
    insert(3, 40, arr, len); // 10 20 30 40
    printArr(arr, len);
    insert(1, 50, arr, len); // 10 50 20 30 40
    printArr(arr, len);
    insert(0, 15, arr, len); // 15 10 50 20 30 40
    printArr(arr, len);
  }

  void erase_test(){
    cout << "***** erase_test *****\n";
    int arr[10] = {10, 50, 40, 30, 70, 20};
    int len = 6;
    erase(4, arr, len); // 10 50 40 30 20
    printArr(arr, len);
    erase(1, arr, len); // 10 40 30 20
    printArr(arr, len);
    erase(3, arr, len); // 10 40 30
    printArr(arr, len);
  }

  void insert(int idx, int num, int arr[], int& len){
      len++;
      for(int i = len-1; i > idx; i--)
      {
          arr[i] = arr[i-1];
      }
      arr[idx] = num;
  }

  void erase(int idx, int arr[], int& len){
      for(int i = idx; i < len-1; i++)
      {
          arr[i] = arr[i+1];
      }
      len--;
```

```
}

void printArr(int arr[], int& len){
  for(int i = 0; i < len; i++) cout << arr[i] << ' ';
  cout << "\n\n";
}
```

전체를 특정 값으로 초기화시킬 때 어떻게 하면 효율적으로 할 수 있을까?



- 그냥 for문을 돌면서 값을 하나하나 다 바꾸는 방식, 코드가 조금 투박하긴 하지만 실수할 여지가 없기 때문에 무난하고 좋습니다.
- algorithm 헤더의 fill 함수를 이용하는 방식, fill 함수는 실수할 여지도 별로 없고 코드도 짧으니 익숙해진다면 가장 추천하는 방식입니다.
- first는 초기화 하고 싶은 부분의 시작 주소, last는 끝 주소, val은 초기화할 값이다. first는 포함하고, last는 포함하지 않음을 주의한다. [first, last)

```
void fill (ForwardIterator first, ForwardIterator last, const T& val)

fill(a, a+21, 0); //1차원 배열

for(int i = 0; i < 21; i++) //2차원 배열
{
fill(b[i], b[i]+21, 0);
}
```

STL vector

http://www.cplusplus.com/reference/vector/vector/

vector는 배열과 거의 동일한 기능을 수행하는 자료구조이다.

배열과 마찬가지로 원소가 메모리에 연속하게 저장되어 있기 때문에 O(1)이며, 인덱스를 가지고 각 원소로 접근 가능하다.

크기를 자유자재로 늘이거나 줄일 수 있다는 점에서 배열과 차이가 있다.

vector example

```
#include <bits/stdc++.h>
  using namespace std;

  int main(void) {
    vector<int> v1(3, 5); // {5,5,5}, 3개의 원소를 가지는 int형 vector v1 생성 후 5로 초기화
    cout << v1.size() << '\n'; // 3, vector v1의 원소 갯수
    v1.push_back(7); // {5,5,5,7}, vector v1의 끝에 7 추가

    vector<int> v2(2); // {0,0}, 2개의 원소를 가지는 int형 vector v2 생성, 기본값(0)으로 초기화
    v2.insert(v2.begin()+1, 3); // {0,3,0}, vector v2의 인덱스 1에 3 삽입

    vector<int> v3 = {1,2,3,4}; // {1,2,3,4}, int형 vector v3 생성, 1,2,3,4로 초기화
    v3.erase(v3.begin()+2); // {1,2,4}, vector v3의 인덱스 2 제거

    vector<int> v4; // {}, int형 vector v 생성
    v4 = v3; // {1,2,4}
    cout << v4[0] << v4[1] << v4[2] << '\n';//124
    v4.pop_back(); // {1,2}, vector v4의 끝에 있는 값 삭제
    v4.clear(); // {}, vector v4의 모든 값 삭제
    }
```

v.begin() : vector v 시작점 주소

v.begin()+x : vector v의 인덱스 x 주소

v.end() : vector v의 끝+1 주소

- STL의 iterator 개념 추가 공부 필요, vector<int>::iterator 타입

vector에서 =를 사용하면 deepcopy 발생. v4=v3; 이후 v4를 바꿔도 v3에는 영향없음.

- **깊은 복사(Deep Copy)**는 '실제 값'을 새로운 메모리 공간에 복사하는 것을 의미하며, **얕은 복사(Shallow Copy)**는 '주소 값'을 복사한다는 의미입니다.

> **얕은 복사**의 경우 주소 값을 복사하기 때문에, **참조하고 있는 실제값은 같습니다. (원본을 참조함)**

insert, erase는 배열과 비슷하게 시간복잡도 O(N)

push_back, pop_back는 O(1)

range-based for loop : vector에 있는 모든 원소를 순회하는 방법

c++11부터 지원하며, vector 외에도 list, map, set등에서도 사용 가능하다.

```
vector<int> v1 = {1,2,3,4,5,6};

  //1. range-based for loop (c++11부터 지원)
  for(int e : v1) //e에 v1의 원소들이 하나씩 들어가는 for문
    cout << e << ' ';

  //2. not bad
  for(int i = 0; i < v1.size(); i++)
    cout << v1[i] <<' ';

  //3. WRONG, 쓰지마세요
  for(int i = 0; i <= v1.size()-1 ; i++)
    cout << v1[i] << ' ';
```

range-based for loop를 쓸 때 주의점

1.

int e : v1이라 하면 v1에서 복사된 값이 e로 들어감

int& e : v1이라 하면 v1의 원본이 e로 들어가기 때문에 e를 바꾸면 원본인 v1에서도 해당 원소의 값이 바뀜

2.

기본적으로 vector의 size 메소드는 시스템에 따라 unsigned int 혹은 unsigned long long을 반환한다.

3번 WRONG case에서 v1이 빈 vector일 경우 v1.size()-1은 (unsigned int)0 - (int)1이고, unsigned int와 int를
연산하면 unsigned int로 자동 형변환이 발생하기 때문에 연산 결과는 -1이 아니라 4294967295가 된다.
(unsigned int underflow!)

```
#include <bits/stdc++.h>
  using namespace std;

  int main(void){
      string word;
      cin >> word;
      int a[26] = {0,};
     // for(int i = 0; i < word.length(); i++)
     // {
     //     a[word.at(i) - 'a']++;
     // }
      for(auto c : word) //auto를 적으면 굳이 자료형을 적어주지 않아도 됨
      {
          a[c-'a']++;
      }

      for(int i = 0; i< 26; i++)
      {
          cout << a[i] << " ";
      }
  }
```

범위기반 for문 (range based for)에서 auto keyword 사용가능