

10주 완성 C++ 코딩테스트 알고리즘 교안

10 WEEK CODING TEST BASIC NOTE

C++는 어렵지 않다! 알고리즘문제를 풀때만 나오는
C++의 기본과 자료구조와 코테에 자주나오는 로직의 핵심

C++의 기초	5
1.1 C++ 코딩을 위한 준비	5
Window에서 시작하기	5
Mac에서 시작하기	8
참고 : brew	9
참고 : bits/stdc++.h	9
참고 : 삼성코딩테스트에서 bits/stdc++.h를 쓸 수 있나요?	10
1.2 C++ 의 기본	16
예제로 이해하는 C++	16
형(타입, type)	17
1. void : 리턴하는 값이 없다.	17
2. char, 문자	18
3. string, 문자열	18
아스키코드	19
4. bool, 참과 거짓	21
5. int, 4바이트짜리 정수	21
6. long long, 8바이트짜리 정수	21
7. double, 실수형	22
8. unsigned long long, 8바이트짜리 양의 정수	22
입력	22
따닥따닥 붙어있는 것을 입력받는 법.	23
1.string으로 변환	23
2 scanf로 받기	24
getline	25
입력이 계속해서 이어질 때	26
출력	27
cout	27
실수형 출력	27
printf	27
형변환	29
1. double을 int형으로 만들기	29
2. 문자를 숫자로, 숫자를 문자로	29
문자열	30
split	31
atoi(s.c_str())	32
true와 false	33
1.3 자료구조	33
pair와 tuple	34
sort() 함수	35

vector	38
vector를 함수 매개변수로 만들어 변화시키기	40
Array	41
배열의 초기화 : fill과 memset	42
배열의 복사 : memcpy	43
배열이란?	43
랜덤 접근과 순차적 접근	44
2차원배열	44
map	45
unordered_map	48
set	49
multiset	50
stack	51
queue	53
deque	54
struct(구조체)	55
priority queue	59
자료구조 복잡도 정리	62
1.4 포인터	63
포인터	63
이터레이터와 포인터	66
call by reference와 value	67
배열과 포인터	68
1.5 수학	69
순열과 조합	69
순열	69
조합	73
조합과 순열의 특징	77
정수론	78
최대공약수와 최소공배수	78
모듈러 연산	79
에라토스테네스의 체	79
등차수열의 합	80
승수 구하기	81
제곱근 구하기	81
1.6 질문으로 배우는 코딩테스트에 자주나오는 필수로직	82
lower_bound와 upper_bound	82
참고 : 이터레이터와 end()	83
시계방향과 반시계방향 회전	85

배열의 합, accumulate	85
배열 중 가장 큰 요소, max_element	86
배열 부분 회전	86
함수인자로 전달해서 변수 수정하기	86
n진법 변환	87
내림차순 정렬	88
커스텀 정렬	89
2차원배열을 수정하는 함수	89
2차원배열을 회전하는 함수	90
1.7 코드를 구축하는 법	90
전역변수와 변수명을 간결하게.	90
동적할당과 정적할당	91
참고 : malloc	92
배열의 경우 조금 더 넓게	92
빠른 속도로 코딩하자!	92
외워야 할 숫자	92
1.8 자주 실수하는 것들	93
bits/stdc++.h에서 기본적으로 사용할 수 없는 변수명	93
입출력 싱크	94
스택오버플로(stack overflow)	94
변수 초기화 문제	95
실수형 연산	95
문자열 크기 선언	95
참조 에러(reference Error)	95
UB(Undefined Behavior)	95
endl보다는 “\n”을 써라.	96
1.9 알고리즘을 공부하는 자세	98
올바른 과정	98
다양하게 풀 수 있을까?	98
손코딩하라.	99
1.10 마치며	99

C++의 기초

코딩테스트를 정복하기 위한 첫걸음입니다. C++의 기초인 형, 입출력, 자료구조 등을 배워보겠습니다.

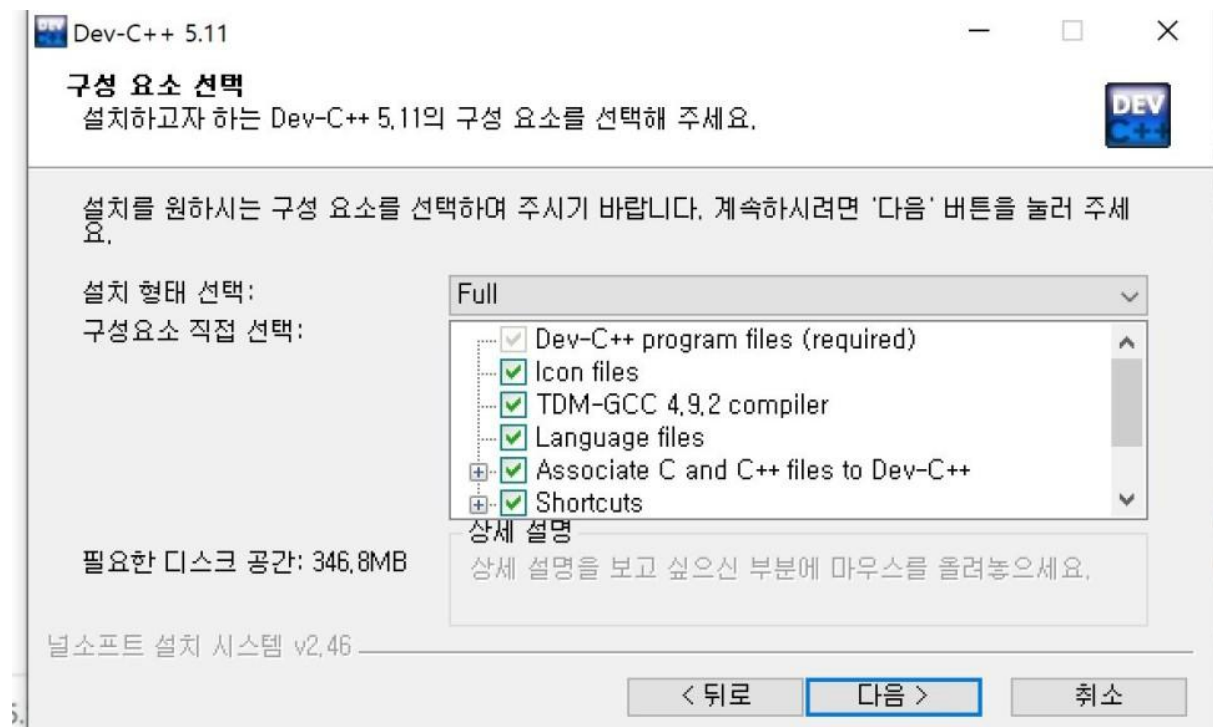
1.1 C++ 코딩을 위한 준비

지금부터 C++로 알고리즘 공부를 시작해보겠습니다. C++은 매우 어려운 언어이기 때문이기 때문에 겁을 먹지만 사실 그건 그저 두려움일뿐, 알고리즘을 하기 위한 알고리즘용 C++은 쉽습니다. 이것들만을 배운다면 쉽게 알고리즘을 구현할 수 있습니다. 또한 C++로 알고리즘을 배우면 알고리즘을 공부할 때 해설코드도 많고 제출하면 코드속도도 빠르며 추후 자바스크립트나 파이썬으로 확장하기도 좋아서 공부하기에 매우 적합합니다.

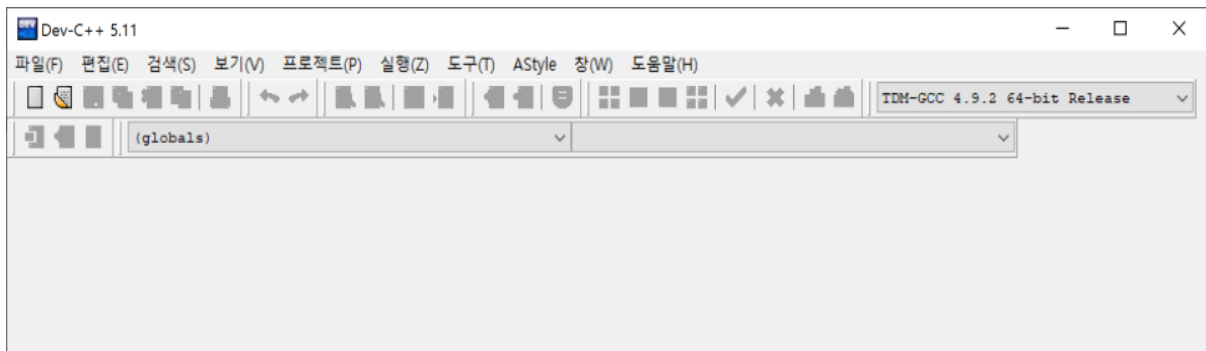
Window에서 시작하기

C++로 코딩을 하기 위해선 컴파일러와 편집기가 필요합니다. 이를 모두 한번에 해결해주는 DevC++를 설치해봅시다. 아래의 링크로 들어가셔서 다운로드를 받아보겠습니다.

- <https://sourceforge.net/projects/orwelldevcpp/>



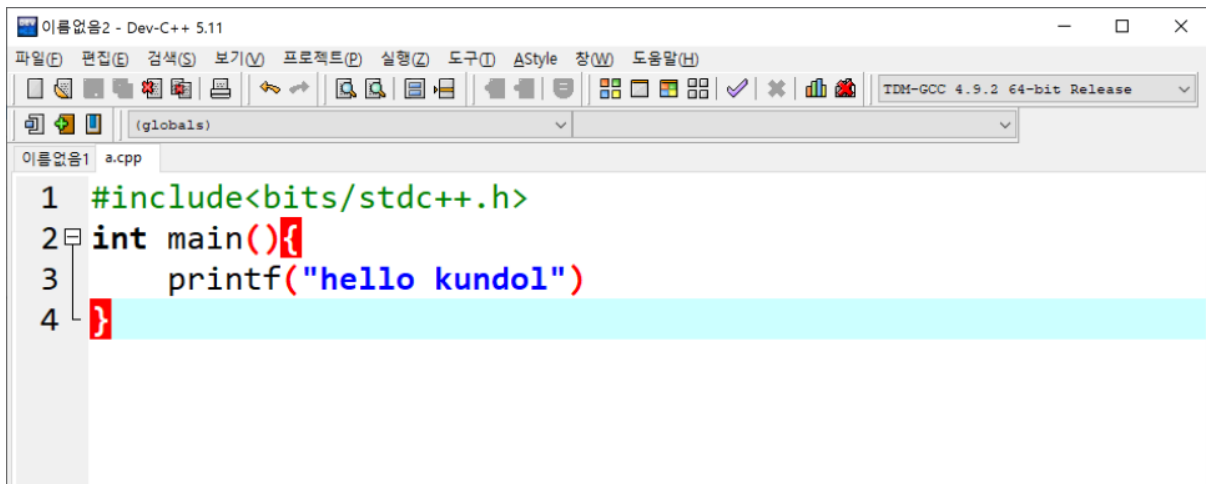
설치를 누르다 보면 이렇게 있는데 즉 TDM-GCC 컴파일러가 자동으로 깔아줍니다. 즉, 편집기 프로그램 내 컴파일러를 활용할 수 있기 때문에 컴퓨터에 따로 컴파일러를 설치하지 않아도 됩니다.



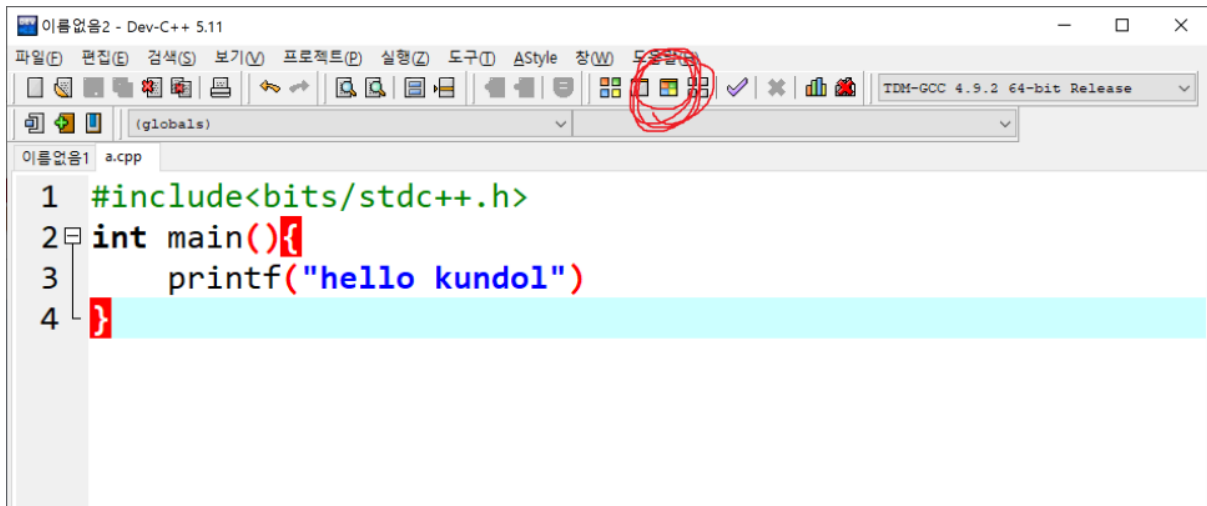
자 이렇게 킨 상태로 [파일] > [새로만들기] > [소스파일] 클릭을 한 이후 아래의 소스코드를 적어봅시다.

```
#include<bits/stdc++.h>
int main(){
    printf("hello kundol");
}
```

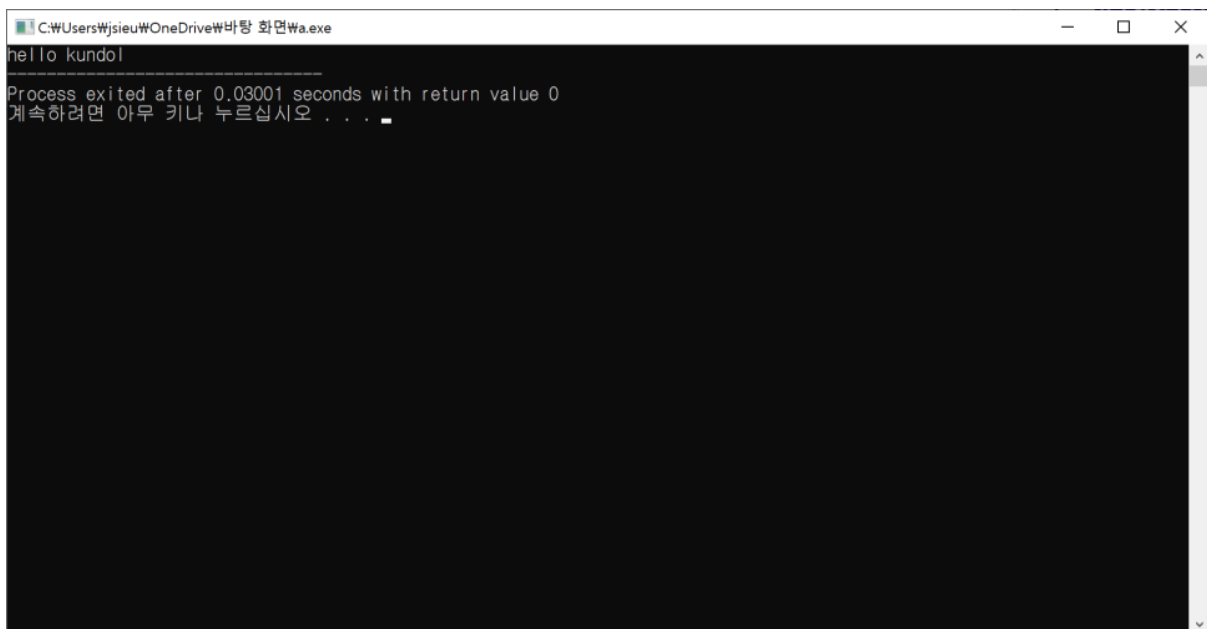
그 이후에 [파일명].cpp로 저장합니다. a.cpp로 저장해봅시다.



그렇게 되면 위의 그림처럼 나타나게 됩니다.



그리고 저 동그라미 부분을 클릭하면 컴파일 이후 프로그램이 실행되게 됩니다.



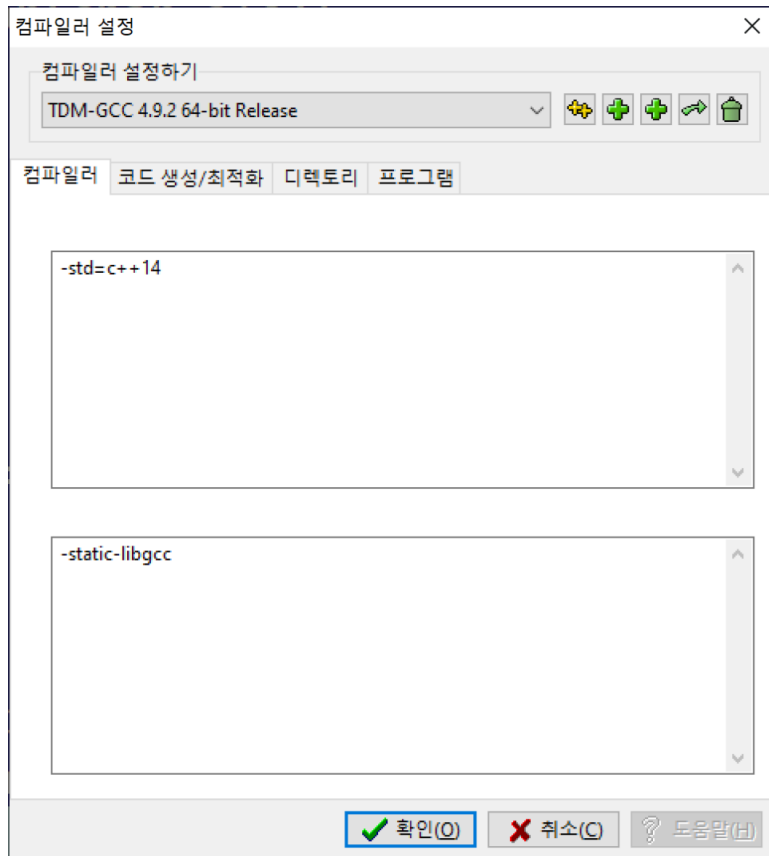
지금 이렇게 실행하면 C++ 낮은 버전으로 실행되게 되는데 [도구] > [컴파일러 설정]에 들어가서 아래 그림처럼 C++14 버전으로 컴파일하게 위와 같이 설정해 줍니다.

아래 글을 복붙해서 진행하면 됩니다.

```

-std=c++14
-static-libgcc

```



윈도우에서 C++ 프로그램을 실행 할 모든 준비가 완료되었습니다.

Mac에서 시작하기

맥에서는 보통 xcode와 vscode 편집기를 통해 코드를 구축하며 이 책에서는 vscode로 설명합니다.
먼저 brew를 통해 gcc를 설치합니다.



```
brew install gcc
```



```
juhongmin — -bash — 80x24
Removing: /Users/juhongmin/Library/Logs/Homebrew/libyaml... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/openjdk@11... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/readline... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/utf8proc... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/lz4... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/ruby-build... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/sqlite... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/gettext... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/tcl-tk... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/apr... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/autoconf... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/m4... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/pcre2... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/openssl@1.1... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/pcre... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/jenkins... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/python@3.8... (3 files, 185.4KB)
)
Removing: /Users/juhongmin/Library/Logs/Homebrew/groonga... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/subversion... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/ruby... (64B)
Removing: /Users/juhongmin/Library/Logs/Homebrew/rbenv... (64B)
Pruned 0 symbolic links and 10 directories from /usr/local
hmJu-MacBookPro:~ juhongmin$
```

참고 : brew



brew는 macOS에서 편하게 패키지를 관리해주는 애플리케이션입니다. 맥에서 어떠한 패키지를 설치한다면 이를 통해 설치하는 것이 일반적입니다.

window의 Dev C++와는 달리 bits/stdc++.h를 include하려면 별도의 작업이 필요합니다.

참고 : bits/stdc++.h

bits/stdc++.h는 C++의 표준 라이브러리가 모두 포함된 헤더입니다. 이를 사용하면 iostream, cstdio 등 여러 라이브러리에 들어있는 함수 등을 하나하나 신경 쓸 필요 없이 코딩에 집중할 수 있습니다.

참고 : 삼성코딩테스트에서 bits/stdc++.h를 쓸 수 있나요?

21년 기준 삼성 코딩테스트의 경우 이 라이브러리를 쓸 수 없으며 코딩테스트 공지에서 탑재되는 주어진 라이브러리만을 써야 합니다. 그렇기 때문에 bit/stdc++.h는 붙이지 못합니다. 그러나 걱정할 필요는 없습니다. “이 강의를 통해 배운 함수들을 써가며 이거 되는건가? 어 cout이 안되네? 그러면 printf를 써야지” 하면서 나아가면 되는겁니다.

하지만 불안할 수도 있는 분들을 위해 준비했습니다. 불안하다면 iostream, stdio.h, string.h, algorithm 의 함수들 정도는 외워갑시다. 많이 나오는 함수들만 준비했습니다.

예를 들어 iostream을 include 하면 cin, cout 등을 쓸 수 있습니다.

```
#include <iostream>
using namespace std;
int n, m;
int main(){
    cin >> n >> m;
}
```

iostream의 함수들

- swap
- getline
- clear
- fill
- tie
- precision
- sync_with_stdio
- cin
- cout
- stdio.h의 함수들
- printf
- scanf
- puts

string.h의 함수들

- memcpy
- memset
- size_t

algorithm의 함수들

- find
- swap
- fill
- remove
- unique
- rotate
- shuffle
- sort
- stable_sort
- lower_bound
- upper_bound
- min
- max
- min_element
- max_element
- next_permutation
- prev_permutation

아래의 코드들을 모두 복사합니다. 또는 아래 링크를 통해 가져올 수 있습니다.

<https://raw.githubusercontent.com/wnghdcjfe/wnghdcjfe.github.io/master/bits/stdc++.h>

```
#ifndef _GLIBCXX_NO_ASSERT
#include <cassert>
#endif
#include <cctype>
#include <cerrno>
#include <cfloat>
#include <ciso646>
#include <climits>
#include <locale>
#include <cmath>
#include <csetjmp>
#include <csignal>
```

```

#include <cstdarg>
#include <cstddef>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>

#if __cplusplus >= 201103L
#include <complex>
#include <cfenv>
#include <cinttypes>
#include <cstdbool>
#include <cstdint>
#include <ctgmath>
#include <cwchar>
#include <cwctype>
#endif

// C++
#include <algorithm>
#include <bitset>
#include <complex>
#include <deque>
#include <exception>
#include <fstream>
#include <functional>
#include <iomanip>
#include <ios>
#include <iosfwd>
#include <iostream>
#include <istream>
#include <iterator>
#include <limits>
#include <list>
#include <locale>
#include <map>
#include <memory>
#include <new>
#include <numeric>
#include <ostream>
#include <queue>
#include <set>
#include <sstream>
#include <stack>

```

```

#include <stdexcept>
#include <streambuf>
#include <string>
#include <typeinfo>
#include <utility>
#include <valarray>
#include <vector>

#if __cplusplus >= 201103L
#include <array>
#include <atomic>
#include <chrono>
#include <condition_variable>
#include <forward_list>
#include <future>
#include <initializer_list>
#include <mutex>
#include <random>
#include <ratio>
#include <regex>
#include <scoped_allocator>
#include <system_error>
#include <thread>
#include <tuple>
#include <typeindex>
#include <type_traits>
#include <unordered_map>
#include <unordered_set>
#endif

```

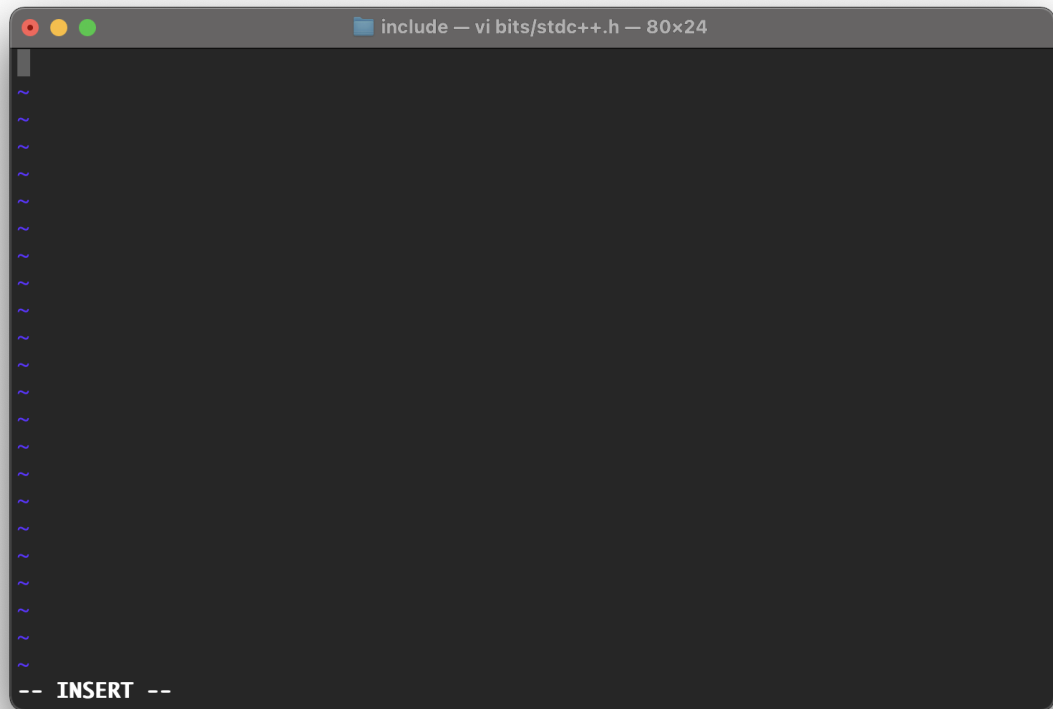
그리고 아래의 명령어를 구동시킵니다.

```

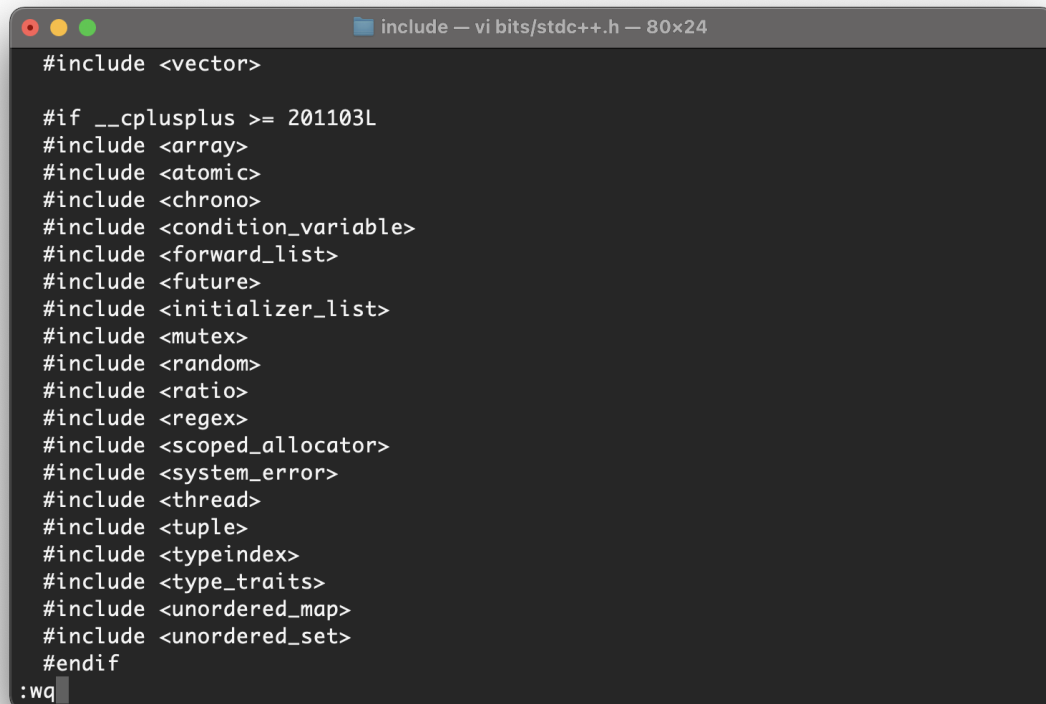
cd /usr/local/include
mkdir bits
cd bits
vi stdc++.h

```

이 후 vi 편집기가 나오면 [a]를 눌러서 삽입모드로 바꿉니다.



그 다음 코드들을 붙여 넣습니다.



```
#include <vector>

#if __cplusplus >= 201103L
#include <array>
#include <atomic>
#include <chrono>
#include <condition_variable>
#include <forward_list>
#include <future>
#include <initializer_list>
#include <mutex>
#include <random>
#include <ratio>
#include <regex>
#include <scoped_allocator>
#include <system_error>
#include <thread>
#include <tuple>
#include <typeindex>
#include <type_traits>
#include <unordered_map>
#include <unordered_set>
#endif

:wq
```

[esc]를 누른 다음 [:wq]를 눌러 저장을 하고 빠져나오면 맥에서 bits/stdc++.h를 쓸 준비가 완료되었습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    cout << 1 << "\n";
    return 0;
}
```

이 후 윈도우처럼 위코드처럼 a.cpp라는 파일을 만든 이 후 다음의 명령어를 실행합니다.

```
g++ -std=c++14 -Wall a.cpp -o test.out
```

a.cpp라는 파일을 엄격하게(-wall) C++14버전으로 컴파일해서 test.out라는 파일을 만든다(-o test.out)는 뜻입니다.

```
./test.out
```

그 후 위 명령어로 실행하면 됩니다.

1.2 C++ 의 기본

예제로 이해하는 C++

자 입력 받은 문자열을 출력하는 프로그램을 하나 만들어보도록 하겠습니다.

```
#include <bits/stdc++.h> // --- (1)
using namespace std; // --- (2)
string a; // --- (3)
int main(){
    cin >> a; // --- (4)
    cout << a << "\n"; // --- (5)
    return 0; // - (6)
}
```

이렇게 만들고 실행시키면... wow 라고 입력을 하면 wow가 출력이 됩니다. 차근차근 설명해보죠.

이 때 중요한 것은 main함수를 하나만 만들어야 하며 무조건 만들어야 한다는 것입니다. C++은 main함수를 중심으로 돌아갑니다. 컴파일이 시작되면 전역변수초기화 라이브러리 import등의 작업이 일어나고 main함수에 얹혀있는 함수들이 작동이 되고 main함수가 0을 리턴하며 프로세스가 종료되게 됩니다.

1. 헤더파일입니다. STL 라이브러리를 **import**합니다. 이 중 bits/stdc++.h는 모든 표준 라이브러리가 포함된 헤더입니다. 그래서 **queue**나 **stack** 등을 사용할 때 일일이 거기에 맞춰 **#include<stack>** 등 입력할 필요가 없습니다. 저것만 입력해놓고 빠르게 알고리즘만 신경 쓰면 됩니다.

2. **std**라는 **namespace**를 사용한다는 뜻입니다. **cin**이나 **cout** 등을 사용할 때 원래는 **std::cin** 이렇게 호출을 해야 하는데 **std**를 기본으로 설정한다는 뜻입니다. 이 네임스페이스는 같은 클래스 이름 구별, 모듈화에 쓰입니다.

3. 문자열을 선언했습니다. <타입> <변수명> 이렇게 선언합니다. **string**이라는 타입(형)을 가진 **a**라는 변수명입니다. 이 때 예를 들어 **string a = "큰돌"** 이라고 해봅시다. 이럴 때 **a**를 **lvalue**라고 하며 큰돌을 **rvalue**라고 합니다. **lvalue**는 추후 다시 사용될 수 있는 변수이며 **rvalue**는 한번쓰고 다시 사용되지 않은 변수를 말합니다.

4. 입력입니다. 대표적으로 **cin**, **scanf**가 있습니다. 후에 좀 더 자세하게 설명합니다.

5. 출력입니다. 대표적으로 **cout**과 **printf**가 있습니다. 후에 좀 더 자세하게 설명합니다.

6. **return 0**입니다. 프로세스가 일이 정상적으로 마무리되며 process **exit** call을 한다는 뜻입니다.

참고로 **stdlib.h** 를 보면 다음과 같이 설명되어있습니다.

```
#define EXIT_SUCCESS    0
#define EXIT_FAILURE    1
```

0을 **return**해야 프로세스를 성공적으로 종료했다는 의미입니다.

이렇게 문자열을 입력받아 출력하는 함수를 만들었습니다.

형(타입, type)

C++ 엄격한 타입시스템 언어입니다. 즉, type을 설정해줘야 합니다. 또한 매개변수의 수나 타입 등에 따라 함수가 다르게 설정됩니다.

즉, func(int a, int b)와 func(int a)는 엄연하게 다른 함수로 인식됩니다.

알고리즘에서 자주 나오는 형은 다음과 같습니다. 이를 배워보겠습니다.

```
void, char, string, bool, int, long long, double, unsigned long long
```

1. void : 리턴하는 값이 없다.

```
#include <bits/stdc++.h>
using namespace std;
int ret = 1;
void a(){
    ret = 2;
    cout << ret << "\n";
    return;
}
int main(){
    a();
    return 0;
}
```

a라는 함수가 ret을 바꾸고 아무것도 리턴하지 않음을 보여줍니다. 이렇게 아무것도 리턴하지 않을 때는 void를 씁니다.

함수를 선언할 때 어떤 형을 반환하는지 명시해주어야 하고 이를 return하는 값과 맞춰주어야 합니다. 위의 코드는 아무것도 반환하지 않는 void형이며 만약 int를 리턴하는 함수는 int, double를 리턴하는 함수는 double를 써야 합니다.

```
#include <bits/stdc++.h>
using namespace std;
double a(){
    return 1.2333;
}
int main(){
    double ret = a();
    cout << ret << "\n";
}
```

```
    return 0;
}
```

위의 코드는 double형을 리턴하는 함수입니다. return 하는 값이 double형이며 함수의 형도 double형인 것을 볼 수 있죠?

함수를 선언할 때는 항상 호출되는 위쪽 부분에 선언을 해야 합니다. 위의 코드를 보면 a()라는 함수를 위에 선언했고 main에서 a()라는 함수를 호출하는 것을 볼 수 있죠?

물론 형과 인자만 선언을 해 놓고 아래쪽에 해서 모듈화를 하는 방법이 있는데 알고리즘은 시간싸움입니다. 그냥 위에다 선언합시다.

2. char, 문자

"이렇게 선언해야 하며 1바이트의 크기를 가집니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    char a = 'a';
    cout << a << "\n";
    return 0;
}
```

문자 a를 선언하고 a를 출력한 예입니다. 이렇게 한 문자만 들어갑니다.

3. string, 문자열

char을 아래의 코드처럼 배열로 선언해 여러개의 문자열을 받을 수도 있습니다.

```
char s[10];
```

하지만 문자열 string으로 선언을 해서 긴 문자열을 할당할 수 있습니다. 보통 char[]을 쓰는 것보다 string을 써서 문제를 푸는 것을 추천드립니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string a = "wow fantastic";
    cout << a << "\n";
    return 0;
}
```

이 문자열은 각각의 문자요소를 a[0], a[1]... 배열처럼 접근할 수 있습니다. 문자열은 문자로 이루어진 배열이니까요.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string a = "wow";
    a += " ";
    a += "fantastic";
    cout << a.size() << "\n";
    cout << a << "\n";
    return 0;
}
```

기존값에 더할 때 += 를 쓰면 더할 수 있으며 길이를 출력할 때는 size 라는 메소드를 씁니다.

만약 숫자로 된 문자에서 1을 더해준다면 어떻게 될까요? 바로 아스키코드에서 + 1한 값이 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);
    string s = "123";
    s[0]++;
    cout << s << "\n"; // 223
    return 0;
}
```

위의 코드를 보면 123에서 s[0]에 1을 더해 223이 되었는데 이는 아스키코드 49에서 1을 더한 값인 50이 가리키는 값이 2이기 때문에 123 에서 223이 되는 것입니다. 즉, 문자열에서 + 하는 연산은 “아스키코드”를 기반으로 수행됩니다.

문자 변수는 C++ 또는 C에서는 해당 문자 자체가 아닌 아스키(ASCII) 값(0에서 127 사이의 정수)으로 저장되어 구현됩니다. 예를 들어 'A'의 ASCII 값은 65입니다. 이것이 의미하는 바는 문자 변수에 'A'를 할당하면 'A' 자체가 아니라 65가 해당 변수에 저장된다는 것입니다.

아스키코드

아스키코드는 1963년 미국 ANSI에서 표준화한 정보교환용 7비트 부호체계이며 000(0x00)부터 127(0x7F)까지 총 128개의 부호가 사용됩니다. 1바이트를 구성하는 8비트 중에서 7비트만 쓰도록 제정된 이유는, 나머지 1비트를 통신 에러 검출을 위한 용도로 비워두었기 때문입니다.

이는 영문 키보드로 입력할 수 있는 모든 기호들이 할당되어 있는 가장 기본적인 부호 체계입니다.

10진수	부호	10진수	부호	10진수	부호	10진수	부호
032		056	8	080	P	104	h
033	!	057	9	081	Q	105	i
034	"	058	:	082	R	106	j
035	#	059	;	083	S	107	k
036	\$	060	<	084	T	108	l
037	%	061	=	085	U	109	m
038	&	062	>	086	V	110	n
039	'	063	?	087	W	111	o
040	(064	@	088	X	112	p
041)	065	A	089	Y	113	q
042	*	066	B	090	Z	114	r
043	+	067	C	091	[115	s
044	.	068	D	092	\	116	t
045	-	069	E	093]	117	u
046	.	070	F	094	^	118	v
047	/	071	G	095	_	119	w
048	0	072	H	096	'	120	x
049	1	073	I	097	a	121	y
050	2	074	J	098	b	122	z
051	3	075	K	099	c	123	{
052	4	076	L	100	d	124	
053	5	077	M	101	e	125	}
054	6	078	N	102	f	126	~
055	7	079	O	103	g		

문자들은 이렇게 숫자로 변환되어 표시됩니다.

만약 'a'라는 문자를 숫자로 변환하고 싶다면 (int)'a'를 하면 되는데 이 때 아스키코드를 기반으로 변환되며 97로 변환됩니다.

```
#include<bits/stdc++.h>
```

```
using namespace std;
int main(){
    ios_base::sync_with_stdio(false); cin.tie(NULL); cout.tie(NULL);
    char a = 'a';
    cout << (int)a << '\n';

    return 0;
}
// 97
```

4. bool, 참과 거짓

1바이트, true또는 false 입니다. 1 또는 0으로 선언해도 무방합니다.

5. int, 4바이트짜리 정수

4바이트짜리 정수를 사용할 때 쓰입니다. 표현범위는 -2,147,483,648 ~ 2,147,483,647 입니다. 약 20억까지 표현할 수 있습니다. 즉, 문제를 푸는 코드에 들어가있는 값들의 예상값이 20억을 넘어간다면 int가 아닌 long long을 써야 합니다.

또한 문제를 풀 때는 이상한 문제가 아니라면 int의 최대값으로 20억까지가 아닌 987654321 또는 1e9를 씁니다. 왜냐하면 이 INF를 기반으로 INF + INF라는 연산이 일어날 수도 있고 INF * 2연산, 그리고 INF + 작은 수 연산이 일어날 때 오버플로를 방지할 수 있기 때문입니다.

```
const int INF = 987654321;
const int INF2 = 1e9;
```

const 는 수정할 수 없는 변수를 말합니다. 보통 INF 같은 것이나 방향벡터를 나타내는 dy, dx에 const를 씁니다. C++로 프로젝트를 한다면 const를 많이 쓰는데 그 이유는 이 const를 붙임으로써 이 변수는 수정하면 안되겠구나 하며 주석처리를 하지 않아도 되는 유지보수성이나 이 변수를 만약 수정한다면 쉽게 에러탐지를 할 수 있는 장점이 있습니다.

6. long long, 8바이트짜리 정수

8바이트짜리 정수입니다. 범위는 -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807입니다. int로 표현이 안될 때 쓰면 됩니다. 예를 들어 문제의 최대범위가 int로는 처리할 수 없는 30억이면 어떻게 해야할까요? 바로 long long을 써야 합니다. 보통은 아래와 같이

INF와 비슷한 이유로 1e18로 정의를 해놓고 쓰면 되나 이는 문제마다 다르니 참고만 해주시면 됩니다.

```
typedef long long ll;  
ll INF = 1e18;
```

7. double, 실수형

실수형입니다. float도 있는데 double이 더 정확합니다. double을 쓰도록 합시다.

```
double pi = 3.221;
```

8. unsigned long long, 8바이트짜리 양의 정수

부호가 없는 정수입니다. -를 표현할 수 없고 그 범위를 몽땅 + 범위에 추가한 겁니다. 아주 가끔 씁니다.

- 범위 : 0 ~ 18,446,744,073,709,551,615

입력

우리는 항상 입력을 받아 문제를 풀어야 합니다. 물론 프로그래머스나 다른 사이트에서는 그러지 않을 수도 있지만 우리는 백준문제를 기본으로 합니다. 백준문제는 항상 입력을 받게 되어있습니다. 그렇다면 어떻게 입력을 받을까요?

```
#include <bits/stdc++.h>  
using namespace std;  
int a;  
int main(){  
    cin >> a;  
    scanf("%d", &a);  
    return 0;  
}
```

보통은 이렇게 cin과 scanf로 받습니다. 저기 %d는 int형을 받겠다라는 뜻이며 %lf로는 실수형을 받을 수 있습니다. %c로는 char형이 있습니다.

문제에서 형식을 기반으로 입력이 주어지지 않은 경우 cin을 쓰는 것이 좋습니다. cin은 개행문자(띄어쓰기, 엔터)를 구분하여 입력을 받습니다.

예를 들어 3.22로 소수점을 두고 입력형식이 주어진 상태로 오는 경우가 있습니다. 이를 어떻게 받아야 할까요? 바로 아래와 같이 받아야 겠죠? 이렇듯, scanf를 쓰는 때는 입력형식이 정해져있을 때 받습니다. 또한 이렇게 실수형으로 입력이 올 때 정수형으로 입력을 받고 싶다면 이렇게 하기도 합니다. 추후 설명하겠지만 실수형연산은 너무나도 정신건강에 해롭습니다. 되도록이면 정수형변환을 해서 하는 것이 좋습니다.

```
scanf("%d.%d", &m1, &m2); //3.22 을 받을 때
```

코드	설명
d	int
c	char
s	string
lf	double
ld	long long

따닥따닥 붙어있는 것을 입력받는 법.

```
4 4
1000
0000
0111
0000
```

이렇게 딱따닥 입력이 붙어서 주어지는 경우가 있습니다. 어떻게 하면 좋을까요?
2가지 방법이 있습니다.

1.string으로 변환

첫번째는 string으로 받아 변환하는 방법입니다. cin으로 받을 때는 개행문자(띄어쓰기, 한줄띄기)까지 받을 수 있다라는 것을 기억해주세요.

```
#include<bits/stdc++.h>
using namespace std;
int n, m, a[10][10];
string s;
int main(){
    cin >> n >> m;
    for(int i = 0; i < n; i++){
        cin >> s;
        for(int j = 0; j < m; j++){
            a[i][j] = s[j] - '0';
        }
    }
}
```

```

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                cout << a[i][j];
            }
            cout << '\n';
        }
    }
    /*
    4 4
    1000
    0000
    0111
    0000
    1000
    0000
    0111
    0000

    */

```

위의 코드처럼 문자열 s를 받아 문자열을 문자로 분해(s[j]) 해서 형변환 s[j] - '0'을 통해 숫자를 int형 배열인 a[i][j] 배열에 넣는 것을 볼 수 있습니다.

2.scanf로 받기

두번째 방법, scanf로 바로 받을 수 있습니다.

```

#include <bits/stdc++.h>
using namespace std;
int a[10][10], n, m;
int main(){
    cin >> n >> m;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            scanf("%1d", &a[i][j]);
        }
    }
    return 0;
}

```

이렇게 앞에 1을 붙이면 “한자리의 int”만을 받겠다라는 뜻이 되어 받을 수 있습니다.

0과 1은 int고 한자리씩 받으면 되니 이렇게 받을 수 있는 셈이죠.

그렇다면 0과 1이 아닌 abcd 이렇게 딱딱딱 붙어있는 문자를 받는다고 했을 때는 어떨까요?


```

#include<bits/stdc++.h>
using namespace std;
char a[10][10];
int main(){
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 4; j++){
            scanf(" %c", &a[i][j]);
        }
    }
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 4; j++){
            cout << a[i][j];
        }
        cout << '\n';
    }

    return 0;
}
/*
LLMM
MMLL
*/

```

이런식으로 앞에 띄어쓰기를 붙인 것을 볼 수 있는데요. 문자를 입력받게 되면 엔터도 “문자”로 취급되어 입력되기 때문에 해당 부분을 고려해서 코드를 구축해야 합니다.

숫자인 d로 입력을 받게 되면 이런 현상이 일어나지 않지만 c로 받게 되면 특수문자, 엔터를 문자로 취급해서 입력을 받기 때문입니다.

getline

제가 아까 cin이 개행문자를 구분해서 받는다고 했죠? 그렇다면 “엄준식 화이팅” 이런 문자열은 어떻게 한번에 받을 수 있을까요? 이럴 땐 getline으로 받으면 됩니다. 아래의 코드는 정답 코드입니다.

```


#include<bits/stdc++.h>
using namespace std;
string s;
int main(){
    getline(cin, s);
    cout << s << '\n';
    return 0;
}
/*
엄준식 화이팅
*/

```

엄준식 화이팅

*/

```
b.cpp
1 #include<bits/stdc++.h>
2 using namespace std;
3 string s;
4 int main(){
5     getline(cin, s);
6     cout << s << '\n';
7     return 0;
8 }
```



입력이 계속해서 이어질 때

그런데 이런 문제가 있습니다. 문제가 입력을 주다가 안줄 때 끝난다고 명시되어있습니다. 그럴 땐 아래와 같이 코드를 구축하면 됩니다.

```
while (scanf("%d", &n) != EOF)
while (cin >> n) // cin으로는 이렇게 하면 됩니다
```

아래는 실행하기 좋도록 설정한 모습입니다.

1안) scanf로 할 때

```
#include <bits/stdc++.h>
using namespace std;
//1안
int n;
int main(){
    while (scanf("%d", &n) != EOF) {
        cout << 1 << '\n';
    }
}
```

2안) cin으로 할 때

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

//2안
int n;
int main(){
    while (cin >> n) {
```

```

        cout << 1 << '\n';
    }
}

```

출력

출력입니다. 크게 cout과 printf가 있습니다.

cout

cout << 출력할 것 << "\n"해서 하는게 일반적입니다. 저렇게 한줄 띄어쓰기를 넣어도 되고 답이 한칸띄어쓰기를 원한다면 cout << "출력할 것" << " "; 이렇게 넣어도 됩니다. 문자열 출력에는 보통 cout을 써서 하는 것이 좋습니다.

실수형 출력

만약 cout을 사용해 정수부분 + 소수부분 해서 6자리까지 출력한다면 어떻게 해야할까요?
(소수부분은 반올림)

```

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
double a = 1.23456789;
int main(){
    cout << a << "\n"; // 1.23457
    cout.precision(7);
    cout << a << "\n"; // 1.234568
    return 0;
}

```

이렇게 precision을 통해 정해주면 됩니다.

printf

형식을 지정해서 출력할 때 좋습니다. 문자열을 출력할 때는 보통 cout을 쓰는 것이 좋으며 c_str()를 통해 printf로도 출력이 가능합니다. %다음에 오는 것으로 출력해야 할 변수의 유형을 정한다음에 출력할 수 있습니다. 예를 들어 문제에서 홍철 1 : 지수 2 이렇게 출력해라라고 한다면 printf("홍철 %d : 지수 %d", a, b)이렇게 해놓으면 되겠죠?

예를 들어 printf의 출력형식을 이용해 소수점 6자리까지, 그리고 2를 02로 만들어서 출력한다면 어떻게 해야할까요? 앞서 입력에서 설명했듯이 "형식"을 지정할 수 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
double a = 1.23456789;
int b = 2;
int main(){
    printf("%.6lf\n", a);
    printf("%02d\n", b);
    // 1.234568
    // 02
    return 0;
}
```

코드	설명
d	int
c	char
s	string
lf	double
ld	long long

```
#include <bits/stdc++.h>
using namespace std;
int a = 1;
char s = 'a';
string str = "어벤져스";
double b = 1.223123;

int main(){
    printf("아이엠어 아이언맨 : %d\n",a);
    printf("아이엠어 아이언맨 : %c\n",s);
    printf("아이엠어 아이언맨 : %s\n",str.c_str());
    printf("아이엠어 아이언맨 : %lf\n",b);
    return 0;
}
/*
아이엠어 아이언맨 : 1
아이엠어 아이언맨 : a
아이엠어 아이언맨 : 어벤져스
*/
```

```
아이엠어 아이언맨 : 1.223123
*/
```

형변환

형변환이 문제에 나올 수 있습니다. 갑자기 int형인 것을 double형으로 등 형을 변환하는게 나올 때는 어떻게 해야 할까요?

1. double을 int형으로 만들기

그냥 앞에다가 int로 선언해주기만 하면 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    double ret = 2.12345;
    int n = 2;
    int a = (int)round(ret / double(n));
    cout << a << "\n";
    return 0;
}
```

또한 같은 타입끼리 연산을 하는 것이 중요합니다. 예를 들어 double은 double끼리 나눠야 합니다. double과 int가 나눠진다면 어떻게 될까요? double이 됩니다. 이런게 순서나 타입간의 관계에 따라 결정이 되는데 이런걸 생각하느니 차라리 double형은 double형 끼리 연산하고 int는 int형끼리 연산하게끔 하는게 "맞왜틀"에 빠지지 않을 가능성을 높여줍니다. **“형과 형을 똑같이 해주어야 합니다.”** 그리고 저렇게 출력할 때 (int)로 형변환을 해줬죠? 저렇게 하는 겁니다.

문제를 드리죠.

```
int a = (int) p * 100
int a = (int) 100 * p
```

위의 코드 2개가 같을까요? 전자가 형변환이 될 겁니다. 순서를 주의해주세요!

2. 문자를 숫자로, 숫자를 문자로

예를 들어 소문자로 된 문자를 숫자로 바꾸는 로직이 필요하다 생각해봅시다. 어떻게 해야 할까요? 정답은 아스키코드를 이용하는 것입니다. 앞서 설명했듯이 A ~ Z 는 65 ~ 90 / a ~ z 는 97 ~ 122의 아스키 코드를 가지고 있습니다.

예를 들어 a부터 시작해 z부터 입력을 받는데 이를 정수 0~ 26까지 표현하고 싶다면 다음과 같이 작성합니다.

```
#include <bits/stdc++.h>
```

```
using namespace std;
int main(){
    char a = 'a';
    cout << (int)a - 97 << "\n";
    return 0;
}
```

이렇게도 똑같은 의미입니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    char a = 'a';
    cout << (int)a - 'a' << "\n";
    return 0;
}
```

문자열

C++에서 많이 사용되는 문자열 함수는 몇개 없습니다.

```
#include<bits/stdc++.h>
using namespace std;
string dopa = "amumu is best";
int main(){
    cout << dopa << "\n";
    if(dopa.find("amumu") != string::npos){
        cout << "dopa속에 아무무있다!\n";
    }
    cout << dopa.substr(0, 3) << "\n";
    reverse(dopa.begin(), dopa.end());
    cout << dopa << "\n";

    return 0;
}
/*
amumu is best
dopa속에 아무무있다!
amu
tseb si umuma
*/
```

- reverse : 원래의 문자열을 바꿔버립니다. begin과 end를 통해 전체를 바꿔도 되고 dopa.begin(), dopa.begin() + 3 이런식으로 부분만 바꿔버릴 수도 있습니다.

- substr : 시작지점으로부터 몇개의 문자열을 뽑아냅니다.(시작지점, 몇개) 이렇게 2개의 매개변수가 들어가게 됩니다. 만약 시작지점만 넣게 되면 마지막까지 문자열을 뽑아냅니다.
- find : 어떠한 문자열이 들어있나 찾습니다. 만약 찾지 못한다면 문자열의 끝 위치인 string::npos를 반환합니다.

split

C++에서는 불행하게도 문자열을 기반으로 split함수를 지원하지 않습니다. 따라서 다음과 같이 구현해야 합니다. 문자열을 문자열을 기준으로 나누는 split함수를 이렇게 만들어봅시다.

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

vector<string> split(string input, string delimiter) {
    vector<string> ret;
    long long pos = 0;
    string token = "";
    while ((pos = input.find(delimiter)) != string::npos) {
        token = input.substr(0, pos);
        ret.push_back(token);
        input.erase(0, pos + delimiter.length());
    }
    ret.push_back(input);
    return ret;
}

vector<string> split_debug(string input, string delimiter) {
    vector<string> ret;
    long long pos = 0;
    string token = "";
    while ((pos = input.find(delimiter)) != string::npos) {
        long long pos = input.find(delimiter);
        cout << "POS : " << pos << '\n'; // 15 12 15 15
        if(pos == string::npos)break;
        token = input.substr(0, pos);
        ret.push_back(token);
        input.erase(0, pos + delimiter.length());
    }
    ret.push_back(input);
    return ret;
}

int main(){
    string s = "안녕하세요 큰돌이는 킹갓제너럴 천재입니다 정말이에요!";
```

```

string d = " ";
vector<string> a = split(s, d);
string s2 = "aaaa bbb ccccc ee dddddddddd!";
vector<string> c = split_debug(s2, d);
// 4 3 5 2 띄어쓰기를 찾아 해당 첫번째 인덱스를 반환한다.
// 문자열이 erase되기 때문에 aaaa, bb 이런식으로 되는 것도 생각.
for(string b : a) cout << b << "\n";
/*
POS : 4
POS : 3
POS : 5
POS : 2
안녕하세요
큰돌이는
킹갓제너럴
천재입니다
정말이에요!
*/
}

```

복잡해보이지만 다음코드의 3줄만 외우면 됩니다.

```

while ((pos = input.find(delimiter)) != string::npos) {
    token = input.substr(0, pos);
    ret.push_back(token);
    input.erase(0, pos + delimiter.length());
}

```

분해할 기준이 되는 delimiter를 기본으로 input에서 그걸 찾고 그걸 기반으로 input을 앞에서부터 지워가며 다시 찾는 로직입니다. 그러면서 최종배열인 ret에다 넣어가며 split이 완성된 배열을 완성시키는 것이죠.

atoi(s.c_str())

문자열을 int로 바꿔야 할 상황이 있습니다. 예를 들어 입력이 "amumu" 또는 0 이렇게 온다라고 했을 때 문자열, string으로 입력을 받아 입력받은 글자가 문자열인지 숫자인지 확인해야 하는 로직이 필요할 때 말이죠.

다음 코드 처럼 입력받은 문자열 s를 기반으로 atoi(s.c_str())로 씁니다. 이렇게 보면 만약 입력받은 문자열이 문자라면 0을 반환하고 그게 아니라면 숫자를 반환하게 됩니다

```

#include <bits/stdc++.h>
using namespace std;
int main(){
    string s = "1";
}

```



```

string s2 = "amumu";
cout << atoi(s.c_str()) << '\n';
cout << atoi(s2.c_str()) << '\n';
return 0;
}
/*
1
0
*/

```

true와 false

C++에서는 0이면 false, 0이 아닌 값들은 모두 true가 됩니다.

```

#include<bits/stdc++.h>
using namespace std;
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
    int a = -1;
    cout << bool(a) << "\n";
    a = 0;
    cout << bool(a) << "\n";
    a = 3;
    cout << bool(a) << "\n";
}
/*
1
0
1
*/

```

1.3 자료구조

자료구조는 왜 존재할까요? 데이터를 효율적으로 접근하고 수정할 수 있게 만들어 놓은 자료의 조직을 말하고 데이터 간의 관계, 그리고 데이터에 적용할 수 있는 함수나 명령을 의미하며 "자료의 처리"를 좀 더 효율적으로 하기 위한 조직을 말합니다. 즉, 어떠한 연산, 어떠한 로직이 많이 발생되었을 때 그것들을 효율적으로 처리하기 위해서 필요한 것이죠.

pair와 tuple

두가지의 값을 담아야 할 때가 있습니다. 그럴 때 쓰는 pair, 그리고 세가지 이상의 값을 넣을 때 쓰는 tuple입니다. 여기서 tie는 pair또는 tuple에 들어있는 두가지 이상의 값을 끄집어낼 때 사용됩니다.

```
#include<bits/stdc++.h>
using namespace std;
pair<int, int> pi;
tuple<int, int, int> tl;
int a, b, c;
int main(){
    pi = {1, 2};
    tl = make_tuple(1, 2, 3);
    tie(a, b) = pi;
    cout << a << " : " << b << "\n";
    tie(a, b, c) = tl;
    cout << a << " : " << b << " : " << c << "\n";
    return 0;
}
```

pair의 경우 {a, b} 또는 make_pair(a, b)로 만들 수 있습니다. 그저 2개의 원소를 담은 바구니를 생각하면 됩니다. 이 때 보통은 a = pi.first; b = pi.second 이런식으로 끄집어내야 하는데 tie(a, b) = pi 이렇게 끄집어 낼 수 있는 것이죠. first와 second라는 코드를 쓰지 않으니 너무나도 편하게 요소를 끄집어낼 수 있습니다. 물론 이 때 a와 b는 변수로 선언되어야 합니다.

```
#include<bits/stdc++.h>
using namespace std;
pair<int, int> pi;
tuple<int, int, int> ti;
int a, b, c;
int main(){
    pi = {1, 2};
    a = pi.first;
    b = pi.second;
    cout << a << " : " << b << "\n";
    ti = make_tuple(1, 2, 3);
    a = get<0>(ti);
    b = get<1>(ti);
    c = get<2>(ti);
    cout << a << " : " << b << " : " << c << "\n";
    return 0;
}
/*
1 : 2
```

```
1 : 2 : 3
```

```
*/
```

위 코드처럼 tie를 쓰지 않으면 조금 더 코드양이 많아지죠? 그래서 tie를 주로 씁니다.

예를 들어 pair형태나 tuple형태의 값들이 들어가고 이를 정렬한다면 어떻게 해야할까요?

sort() 함수

먼저 sort() 를 배워보겠습니다. sort는 배열 등 컨테이너들의 요소를 정렬하는 함수입니다. 보통 array나 vector를 정렬할 때 씁니다.

- 컨테이너 : 같은 타입의 여러 객체를 저장하는 집합

sort에 들어가는 매개변수로는 3가지가 있으며 2개는 반드시 넣어야 하며 한개는 선택입니다. (넣어도 되고 안 넣어도 됩니다. 커스텀하게 정렬하고 싶을 때 넣습니다.)

sort(first, last, *커스텀비교함수)

이렇게 들어갑니다. first는 정렬하고 싶은 배열의 첫번째 이터레이터, last는 정렬하고 싶은 배열의 마지막 이터레이터를 넣으면 됩니다. 또한 [first,last) 라는 범위를 갖습니다. 즉, first는 포함하고 last는 포함하지 않는다는 의미입니다. 그렇기 때문에 예를 들어 크기가 5인 a라는 배열 전체를 sort한다고 하면 sort(a[0],a[0] + 5)라고 해야 합니다. 즉, last에 배열의 마지막요소가 아닌 그 “다음”의 위치를 넣어주어야 합니다.

다시 말하자면 a[0] + 5는 배열의 마지막원소가 아닙니다. 마지막원소는 a[0] + 4이겠죠? 하지만 sort의 특성상 last)로 last는 포함하지 않기 때문에 마지막 원소 그 다음의 위치를 가리켜야 하는 것이죠.

또한 내가 만약 3번째 요소까지만 정렬하고 싶다면 sort(a[0], a[0] + 3)이렇게 하면 됩니다. 그렇게 하면 a[0] + 2 까지 정렬됩니다.

3번째 인자로는 커스텀비교함수(선택)이 있습니다. sort에 커스텀비교함수를 넣지 않으면 기본적으로 오름차순이며 이를 3번째 인자에 greater<int>()를 넣어 내림차순등으로 변경할 수 있습니다. 참고로 less<int>()을 통해 오름차순으로 정렬할 수 도 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
vector<int> a;
int b[5];
int main(){
    for(int i = 5; i >= 1; i--) b[i - 1] = i;
```

```

    for(int i = 5; i >= 1; i--) a.push_back(i);
    // 오름차순
    sort(b, b + 5);
    sort(a.begin(),a.end());
    for(int i : b) cout << i << ' ';
    cout << '\n';
    for(int i : a) cout << i << ' ';
    cout << '\n';

    sort(b, b + 5, less<int>());
    sort(a.begin(),a.end(), less<int>());
    for(int i : b) cout << i << ' ';
    cout << '\n';
    for(int i : a) cout << i << ' ';
    cout << '\n';

    //내림차순
    sort(b, b + 5, greater<int>());
    sort(a.begin(),a.end(), greater<int>());
    for(int i : b) cout << i << ' ';
    cout << '\n';
    for(int i : a) cout << i << ' ';
    cout << '\n';

    return 0;
}
/*
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
5 4 3 2 1
5 4 3 2 1
*/

```

먼저 따로 설정하지 않으면 first, second, third 순으로 차례차례 오름차순 정렬됩니다.

```

#include<bits/stdc++.h>
using namespace std;
vector<pair<int, int>> v;
int main(){

```

```

    for(int i = 10; i >= 1; i--){
        v.push_back({i, 10 - i});
    }
    sort(v.begin(), v.end());
    for(auto it : v) cout << it.first << " : " << it.second << "\n";
    return 0;
}
/*
1 : 9
2 : 8
3 : 7
4 : 6
5 : 5
6 : 4
7 : 3
8 : 2
9 : 1
10 : 0
*/

```

위 코드처럼 오름차순 정렬된 것을 볼 수 있습니다. 여기서 `for(auto it : v) cout << it.first << " : " << it.second << "\n";` 이 부분은 `for(pair<int,int> it : v`로 할 수도 있습니다. vector v에 있는 “요소”들을 끄집어내서 순회한다는 의미죠. `v[0]`, `v[1]` 따위로 접근한다는 의미입니다. `auto`는 형이 정해져있지 않은 형을 뜻합니다. `pair<int, int>` 보다 `auto`가 코드수가 더 짧기 때문에 `pair`나 `tuple`에 있는 값이나 `struct`에 있는 값을 기반으로 순회할 때 이런 걸 쓰면 조금 더 빠르게 코드를 짤 수 있습니다.

자 그렇다면 내림차순 정렬을 하고 싶을 때는 어떻게 해야 할까요? 또 `first`는 내림차순, `second`는 오름차순 정렬하고 싶다면요? 바로 커스텀 연산자를 만들면 됩니다. `sort` 함수에 3번째 인자는 커스텀 오퍼레이터를 넣는 인자입니다. 커스텀으로 설정한 정렬함수를 집어넣어서 “커스텀”하게 정렬할 수 있는 것이죠.

```

#include<bits/stdc++.h>
using namespace std;
vector<pair<int, int>> v;
bool cmp(pair<int, int> a, pair<int, int> b){
    return a.first > b.first;
}
int main(){
    for(int i = 10; i >= 1; i--){
        v.push_back({i, 10 - i});
    }
}

```

```

    sort(v.begin(), v.end(), cmp);
    for(auto it : v) cout << it.first << " : " << it.second << "\n";
    return 0;
}
/*
10 : 0
9 : 1
8 : 2
7 : 3
6 : 4
5 : 5
4 : 6
3 : 7
2 : 8
1 : 9
*/

```

- 참고로 커스텀 오퍼레이터는 struct를 설명할 때 자세히 배웁니다.

vector

벡터(vector)는 동적으로 요소를 할당할 수 있는 동적 배열입니다. 컴파일 시점에 개수를 모른다면 벡터를 써야 합니다. 또한, 중복을 허용하고 순서가 있고 랜덤 접근이 가능합니다. 탐색과 맨 뒤의 요소를 삭제하거나 삽입하는 데 $O(1)$ 이 걸리며, 맨 뒤나 맨 앞이 아닌 요소를 삭제하고 삽입하는 데 $O(n)$ 의 시간이 걸립니다.

```

#include <bits/stdc++.h>
using namespace std;
vector<int> v;
int main(){
    for(int i = 1; i <= 10; i++)v.push_back(i);
    for(int a : v) cout << a << " ";
    cout << "\n";
    v.pop_back();

    for(int a : v) cout << a << " ";
    cout << "\n";

    v.erase(v.begin(), v.begin() + 1);

    for(int a : v) cout << a << " ";
    cout << "\n";
}

```

```

    auto a = find(v.begin(), v.end(), 100);
    if(a == v.end()) cout << "not found" << "\n";

    fill(v.begin(), v.end(), 10);
    for(int a : v) cout << a << " ";
    cout << "\n";
    v.clear();
    cout << "아무것도 없을까?\n";
    for(int a : v) cout << a << " ";
    cout << "\n";
    return 0;
}
/*
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
not found
10 10 10 10 10 10 10 10
아무것도 없을까?
*/

```

뒤부터 요소를 더하는 `push_back()`, 맨 뒤부터 지우는 `pop_back()`, 지우는 `erase`, 요소를 찾는 `find`(vector의 함수가 아니라 algorithm에서 제공하는 함수입니다.), 배열을 초기화 하는 `clear()`이라는 함수가 있습니다. 또한 `push_back()`과 같은 기능을 하는 `emplace_back()`도 있습니다. 이 함수가 더 빠르지만 알고리즘을 할 때 `push_back()`을 써도 무방합니다.(시간차이가 그렇게 많이 나지 않습니다.)

```
for(int a : v )
```

잠시 위의 코드를 보겠습니다. 이는 "vector의 요소를 순차적으로 탐색한다"라는 뜻입니다. 그렇다면 vector에 pair이라는 값이 들어가면 `for(pair<int,int> a : v)`이런식으로 정의해야 해야겠죠? `for(int i = 0; i < v.size(); i++) v[i]` 이런식으로 index를 기반으로 접근하는 것과 똑같은 의미입니다.

또한 vector에 크기를 정해놓고 할 수도 있습니다.

```

#include <bits/stdc++.h>
using namespace std;
vector<int> v(5, 100);
int main(){
    for(int a : v) cout << a << " ";
}

```

```

    cout << "\n";
    return 0;
}
/*
100 100 100 100 100
*/

```

위코드는 5크기의 vector를 선언하고 100으로 채운 모습입니다.

vector를 함수 매개변수로 만들어 변화시키기

함수 매개변수로 넘겨서 변화시키고 싶을 때는 다음과 같이 넘기면 됩니다. call by reference로 넘겨서 수정하게 해야 합니다.

```

#include<bits/stdc++.h>
using namespace std;
void f(vector<int> &v){
    v[0] = 100;
}
int main(){
    vector<int> v;
    for(int i = 1; i <= 3; i++)v.push_back(i);
    f(v);
    for(int i : v) cout << i << " ";
    return 0;
}
// 100 2 3

```

만약 vector<int> v[10] 이런꼴의 10개의 vector는 어떻게 넘길까요?

```

#include<bits/stdc++.h>
using namespace std;
void f(vector<int> v[10]){
    v[0][0] = 1000;
}
int main(){
    vector<int> v[10];
    v[0].push_back(100);
    f(v);
    for(int i : v[0]) cout << i << " ";
    return 0;
}
// 1000

```


또한 2차원 vector의 경우 어떻게 넘길까요?

이렇게 넘기면 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
void f(vector<vector<int>> &v){
    vector<int> vv;
    vv.push_back(10000);
    v.push_back(vv);
}
int main(){
    vector<vector<int>> v;
    f(v);
    for(int i : v[0]) cout << i << " ";
    return 0;
}
// 10000
```

Array

정적배열입니다. 연속된 메모리 공간이며 스택에 할당됩니다. 컴파일단계에서 크기가 결정됩니다. c스타일과 std스타일이 있는데 c스타일을 중심으로 배웁니다. c스타일은 `int a[10]` 이렇게 선언하는 것이며 std스타일은 `array<int, 10> a;` 이렇게 선언하는 것을 말합니다.

```
#include <bits/stdc++.h>
using namespace std;
int v[10];
int main(){
    for(int i = 1; i <= 10; i++)v[i - 1] = i;
    for(int a : v) cout << a << " ";
    cout << "\n";

    auto a = find(v, v + 10, 100);
    if(a == v + 10) cout << "not found" << "\n";

    fill(v, v + 10, 10);
    for(int a : v) cout << a << " ";
    cout << "\n";
}
```

```

        return 0;
    }
    /*
    1 2 3 4 5 6 7 8 9 10
    not found
    10 10 10 10 10 10 10 10 10 10
    */

```

뭔가 vector에 비해 좀 텅텅 비었습니다. erase, push_back 등의 메서드들이 없기 때문입니다.

배열의 크기를 먼저 선언해 준 후 배열의 요소를 할당해 준 뒤 이런식으로 진행해야 합니다. 이 때 크기는 문제에서 필요한 “최대크기”로 선언해야 합니다.

배열의 초기화 : fill과 memset

위의 코드를 보면 fill을 통해 초기화를 진행한 것을 볼 수 있습니다. fill(시작값, 끝값, 초기화하는값)이렇게 하면 됩니다. 이는 memset으로도 할 수 있으며 memset은 바이트단위로 초기화를 하며 보통 0, -1, 하나의 문자로 초기화를 할 때 사용합니다. memset(초기화하는 배열, 값, 배열의 크기) 이렇게 사용합니다.

```

//vector 초기화 하기
vector<int> v[10]; //v벡터를 10개를 생성합니다.
vector<int> v(n, 0); 0이라는 value를 가진 n개의 벡터를 생성
//vector로 2차원 배열 매트릭스 만들기
vector<vector<int> > checked;
vector <vector<int> > v(n + 1 , vector<int> (n + 1, 0));
//fill로도 초기화할 수 있다.
fill(v.begin(), v.end(), 0);

//배열초기화 shortcut : 전체 0으로 초기화한다. 일부 컴파일러에서 통하지 않을
수도 있습니다.
int dp[10] = {0,};
int dp[10][10] = {{0, } };
fill(dist, dist + MAX_N, 0);

//부분초기화 : 0번째를 0, 1번째를 1로 초기화한다.
int a[n] = {0,1};

memset(check, 0, sizeof(check));
//fill을 사용한 2차원 배열 초기화는 이렇습니다.
for(int i = 0; i < 10; i++) fill(dp[i], dp[i]+10, 0);
fill(&arr[0][0], &arr[0][0] + n*m, k) // 이런식으로 한번에 초기화를 할 수
있습니다.

```

배열의 복사 : memcpy

어떤 변수의 메모리에 있는 값들을 다른 변수의 “특정 메모리값”으로 복사할 때 사용합니다. 주로 배열을 복사할 때 사용합니다. 아래와 같이 어떤 배열을 수정할 때 원본 배열을 온전히 저장하고 싶을 때 다음과 같이 씁니다. 아래의 모습은 temp라는 배열에 a를 담아두고 a를 수정하는 로직을 구현한 뒤 a라는 배열에 다시 예전 온전한 a를 담은 temp를 이용해 다시 a를 만드는 모습입니다.

```
memcpy(temp, a, sizeof(a));  
//a 라는 배열을 수정하는 로직  
memcpy(a, temp, sizeof(temp));
```

예를 들어 이렇게 temp라는 배열에 a라는 배열의 모든 것이 담기는 것을 보 수 있습니다.

```
#include <bits/stdc++.h>  
using namespace std;  
int a[5];  
int temp[5];  
int main(){  
    for(int i = 0; i < 5; i++)a[i] = i;  
    memcpy(temp, a, sizeof(a));  
    for(int i : temp) cout << i << ' ';  
    return 0;  
}  
/*  
0 1 2 3 4  
*/
```

배열이란?

배열(array)은 같은 타입의 변수들로 이루어져 있고, 크기가 정해져 있으며, 인접한 메모리위치에 있는 데이터를 모아놓은 집합입니다. C++에서는 vector나 array로 구현을 하죠. 또한, 중복을 허용하고 순서가 있습니다.

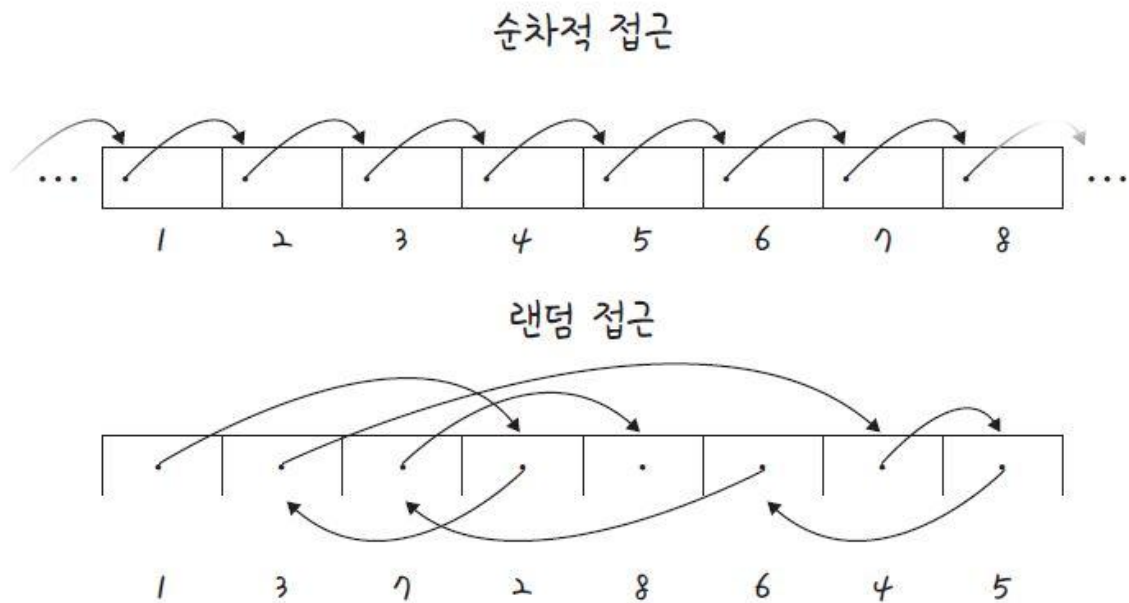
여기서 설명하는 배열은 ‘정적 배열’을 기반으로 설명합니다. 탐색에 $O(1)$ 이 되어 랜덤접근(random access)이 가능합니다. 삽입과 삭제에는 $O(n)$ 이 걸립니다. 따라서 데이터추가와 삭제를 많이 하는 것은 연결 리스트, 탐색을 많이 하는 것은 배열로 하는 것이 좋습니다.

배열은 인덱스에 해당하는 원소를 빠르게 접근해야 하거나 간단하게 데이터를 쌓고 싶을때 사용합니다.

랜덤 접근과 순차적 접근

직접 접근이라고 하는 랜덤 접근은 동일한 시간에 배열과 같은 순차적인 데이터가 있을때 임의의 인덱스에 해당하는 데이터에 접근할 수 있는 기능입니다. 이는 데이터를 저장된 순서대로 검색해야 하는 순차적 접근과는 반대입니다.

▼그림 5-4 랜덤 접근과 순차적 접근



2차원배열

2차원배열은 단순히 차원을 늘려서 만들면 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
const int mxy = 3;
const int mxx = 4;

int a[mxy][mxx];
int main(){
    for(int i = 0; i < mxy; i++){
        for(int j = 0; j < mxx; j++){
            a[i][j] = (i + j);
        }
    }
}
```

```

//good
for(int i = 0; i < mxy; i++){
    for(int j = 0; j < mxx; j++){
        cout << a[i][j] << ' ';
    }
    cout << '\n';
}

//bad
for(int i = 0; i < mxx; i++){
    for(int j = 0; j < mxy; j++){
        cout << a[j][i] << ' ';
    }
    cout << '\n';
}
return 0;
}
/*
0 1 2 3
1 2 3 4
2 3 4 5
0 1 2
1 2 3
2 3 4
3 4 5
*/

```

2차원배열을 만들었고 출력한 예제입니다. 이 때 중요한 점이 있는데 첫번째 차원 >> 2번째 차원 순으로 탐색하는게 성능이 좋습니다. 이는 C++에서 캐시를 첫번째 차원(여기서는 y좌?) 를 기준으로 하기 때문에 캐시관련 효율성 때문에 탐색을 하더라도 순서를 지켜가며 해주어야 하는 것이죠.

map

예를 들어서 "이승철" : 1, "김현영" : 2 이런식으로 string : int 형태로 값을 할당해야 할 때 있죠? 그럴 때 map을 씁니다.

key와 value형태로 이루어져 있고 레드-블랙트리라는 구조를 내장하고 있습니다. 그리고 데이터를 삽입할 때 "정렬된"위치에 삽입되게 됩니다.

map은 unordered_map과 map 두가지가 있습니다. 하나는 정렬을 보장하지 않는, 하나는 정렬을 보장하는 자료구조 입니다. map<string, int> 이런식으로 됩니다. 배열과 비슷하게 clear로 맵에 있는

모든 요소들을 삭제할 수 있으며 아래를 보시면 됩니다. size()로 map의 크기를 구할 수 있으며 erase로 해당 key값을 지울 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
map<string, int> mp;
int main(){
    //이렇게 넣기도 가능하고
    mp.insert({"test1", 1});
    //이렇게 넣을 수도 있습니다.
    mp.emplace("test5", 5);
    //또한 이렇게 변경도 가능, 추가할 수도 있습니다. 아래를 권장합니다.
    mp["test1"] = 4;

    for(auto element : mp){
        cout << element.first << " :: " << element.second << '\n';
    }
    //map의 find메소드는 찾지 못하면 end() 이터레이터를 반환합니다.
    auto search = mp.find("test4");
    if(search != mp.end()){
        cout << "found :" << search -> first << " " << (*search).second
<< '\n';
    }else{
        cout << "not found.." << '\n';
    }
    //이런식으로 바로 int형을 증가시킬 수 있습니다.
    mp["test1"]++;
    cout << mp["test1"] << "\n";

    cout << mp.size() << "\n";
    mp.erase("test1");
    cout << mp.size() << "\n";

    return 0;
}
/*
test5 :: 5
test1 :: 4
not found..
5
2
1
*/
```

map을 순회할 때는 key값을 first, value값은 second로 탐색이 가능합니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    map<string, int> _map;
    _map["큰돌"]++;
    _map["큰돌"]++;
    for(auto c : _map){
        cout << c.first << " : " << c.second << "\n";
    }
    return 0;
}
/*
큰돌 : 2
*/
```

다음 코드처럼 map의 경우 해당 인덱스에 참조만 해도 맵에 값이 생기며 맵의 요소가 생기게 됩니다. int형 같은 경우 0으로. string 같은 경우 빈문자열로 들어가게 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
map<int, int> mp;
map<string, string> mp2;
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
    cout << mp[1] << '\n';
    cout << mp2["aaa"] << '\n';
    for(auto i : mp) cout << i.first << " " << i.second;
    cout << '\n';
    for(auto i : mp2) cout << i.first << " " << i.second;

    return 0;
}
/*
0

1 0
aaa %
*/
```

그렇기 때문에 다음 코드처럼 “맵에 요소가 있는지 없는지”를 확인하고 맵에 데이터를 할당하는 부분의 로직을 다음과 같이 구축할 수 있습니다.

```
#include <bits/stdc++.h>
```

```

using namespace std;
map<int, int> mp;
map<string, string> mp2;
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
    if(mp[1] == 0){
        mp[1] = 2;
    }
    for(auto i : mp) cout << i.first << " " << i.second;

    return 0;
}
/*
1 2
*/

```

unordered_map

- map : 정렬이 됨 / 레드블랙트리 기반 / 탐색, 삽입, 삭제에 $O(\log N)$ 이 걸림
- unordered_map : 정렬이 안됨 / 해시테이블 기반 / 탐색, 삽입, 삭제에 평균적으로 $O(1)$, 가장 최악의 경우 $O(N)$

unordered_map 은 다음과 같이 씁니다.

```

#include<bits/stdc++.h>
using namespace std;
unordered_map<string, int> umap;
int main(){

    umap["bcd"] = 1;
    umap["aaa"] = 1;
    umap["aba"] = 1;
    for(auto it : umap){
        cout << it.first << " : " << it.second << '\n';
    }
}
/*
정렬이 되지 않는다.
aba : 1
aaa : 1
bcd : 1
*/

```


알핏보면 정렬이 필요로 하지 않은 문제에는 unordered_map을 써야 할 것같지만 제출해보면 시간초과가 나기도 합니다. 이는 가장 최악의 경우 O(N)이기 때문이죠. 그냥 되도록 map을 쓰는 것을 권장합니다.

set

셋(set)은 특정 순서에 따라 고유한 요소를 저장하는 컨테이너이며, 중복되는 요소는 없고 오로지 희소한(unique) 값만 저장하는 자료 구조입니다.

다음 코드처럼 pair나 int형을 집어넣어서 만들 수 있습니다. 중복된 값은 제거되며 희소한 값만을 출력하는 것을 볼 수 있습니다. 나머지 사항은 map과 똑같습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    set<pair<string, int>> st;
    st.insert({"test", 1});
    st.insert({"test", 1});
    st.insert({"test", 1});
    st.insert({"test", 1});
    cout << st.size() << "\n";
    set<int> st2;
    st2.insert(1);
    st2.insert(2);
    st2.insert(2);
    for(auto it : st2){
        cout << it << '\n';
    }

    return 0;
}
/*
1
1
2
*/
```

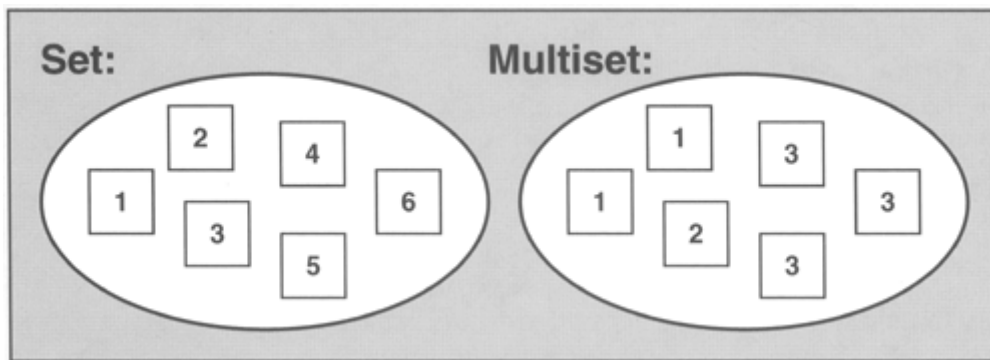
```
#include <bits/stdc++.h>
using namespace std;
int main(){
```

```

set<pair<string, int>> _set;
_set.insert({"test", 1});
_set.insert({"test", 1});
_set.insert({"test", 1});
_set.insert({"test", 1});
cout << _set.size() << "\n";
return 0;
}
/*
1
*/

```

multiset



multiset은 중복되는 원소도 집어넣을 수 있는 자료구조입니다. key, value 형태로 집어넣을 필요도 없고 넣으면 자동적으로 정렬되는 편리한 자료구조입니다. 아래처럼 erase, find, insert가 가능합니다.

```

#include <bits/stdc++.h>
using namespace std;
multiset<int> s;
int main() {
    s.insert(12);
    s.insert(10);
    s.insert(2);
    s.insert(10);
    s.insert(90);
    s.insert(85);
    s.insert(45);

    cout << "Multiset elements after sort\n";
    for (auto it = s.begin(); it != s.end(); it++)
        cout << *it << ' ';
    cout << '\n';
}

```

```

auto it1 = s.find(10);
auto it2 = s.find(90);

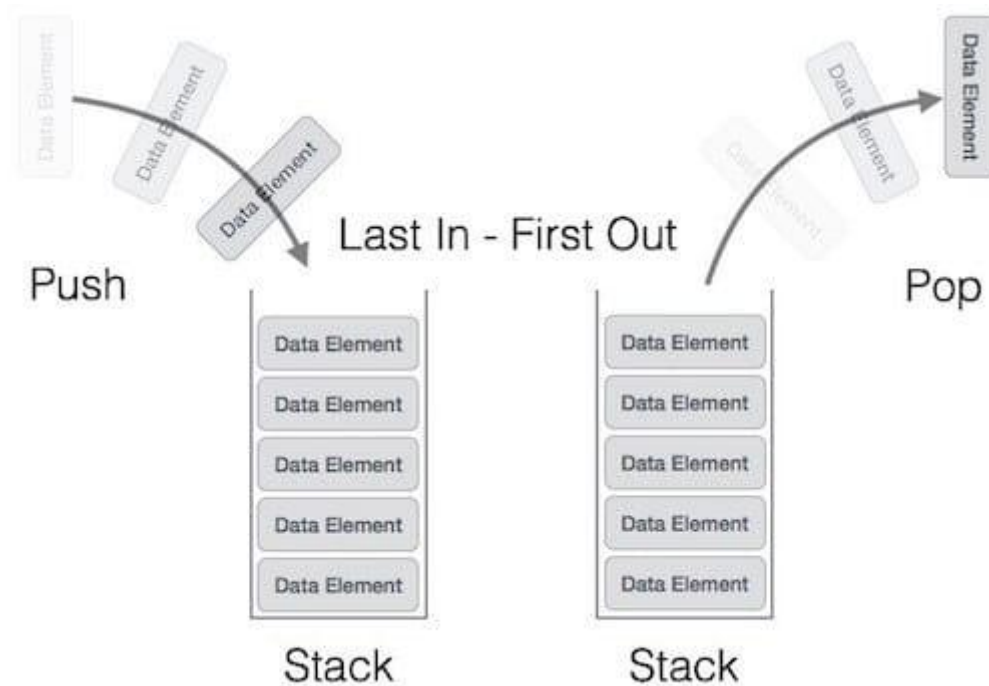
// elements from 10 to elements before 90
// erased
s.erase(it1, it2);

cout << "Multiset Elements after erase:\n";
for (auto it = s.begin(); it != s.end(); it++)
    cout << (*it) << ' ';
s.erase(s.begin());
cout << "\n";
for (auto it = s.begin(); it != s.end(); it++)
    cout << (*it) << ' ';
return 0;
}
/*
Multiset elements after sort
2 10 10 12 45 85 90
Multiset Elements after erase:
2 90
90
*/

```

stack

스택은 가장 마지막으로 들어간 데이터가 가장 첫 번째로 나오는 성질(LIFO, Last In First Out)을 가진 자료 구조입니다. 재귀적인 함수, 알고리즘에 사용되며 웹 브라우저 방문 기록 등에 쓰입니다. 삽입 및 삭제에 $O(1)$, 탐색에 $O(n)$ 이 걸립니다.



LIFO, 가장 마지막으로 들어간 데이터가 가장 첫번째로 나오는 구조를 지녔습니다. 문자열 폭발, 아름다운 괄호만들기 짝을 찾기 등 문제에 쓰입니다. “교차하지 않고” 라는 문장이 나오면 스택을 사용할지 말지를 의심해봐야 합니다.

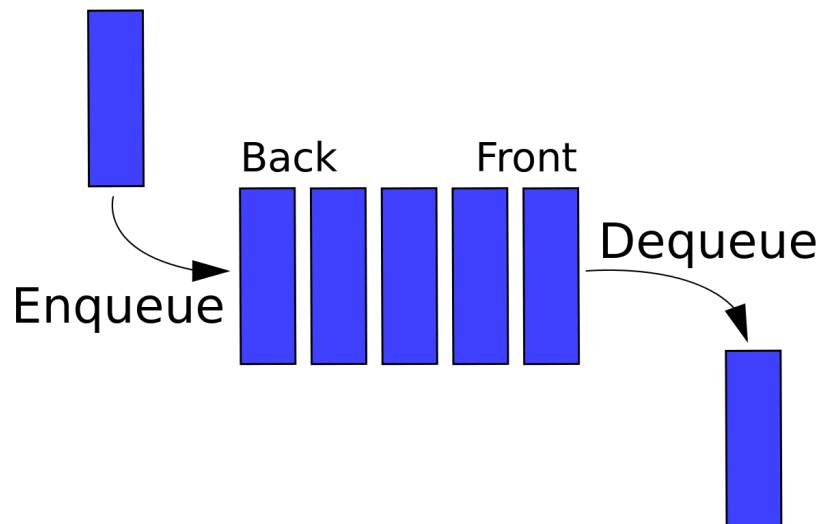
```
#include<bits/stdc++.h>
using namespace std;
stack<string> stk;
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    stk.push("엄");
    stk.push("준");
    stk.push("식");
    stk.push("화");
    stk.push("이");
    stk.push("팅");
    while(stk.size()){
        cout << stk.top() << "\n";
        stk.pop();
    }
}
/*
팅
이
화
식
준
*/
```

연

*/

queue

큐(queue)는 먼저 집어넣은 데이터가 먼저 나오는 성질(FIFO, First In First Out)을 지닌 자료 구조이며, 나중에 집어넣은 데이터가 먼저 나오는 스택과는 반대되는 개념을 가졌습니다. 삽입 및 삭제에 $O(1)$, 탐색에 $O(n)$ 이 걸립니다.



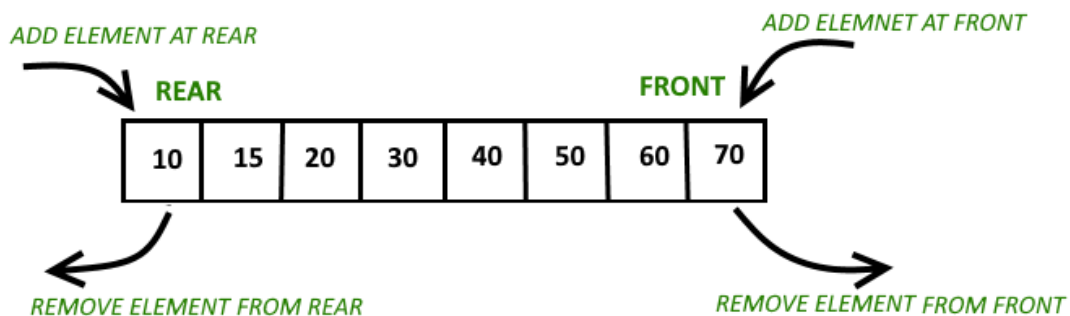
주로 BFS에 쓰입니다. 선입선출구조(FIFO)로써 이렇게 쓰입니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    queue<int> q;
    q.push(1);
    cout << q.front() << "\n";
    q.pop();
    cout << q.size() << "\n";
    return 0;
}
/*
1
0
*/
```

보통 queue는 아래와 같이 size()와 함께 쓰입니다. pop()을 하게 되면 size가 작아지겠죠? 그러다가 0이 되면 false가 되면서 while문이 종료가 되어 루프를 빠져나오게 되는 로직입니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    queue<int> q;
    for(int i = 1; i <= 10; i++)q.push(i);
    while(q.size()){
        cout << q.front() << ' ';
        q.pop();
    }
    return 0;
}
/*
1 2 3 4 5 6 7 8 9 10
*/
```

deque



queue는 앞에서만 끄집어낼 수 있죠? 이것은 앞뒤로 참조가 가능한 자료구조입니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    deque<int> dq;
    dq.push_front(1);
    dq.push_back(2);
    dq.push_back(3);
    cout << dq.front() << "\n";
    cout << dq.back() << "\n";
    cout << dq.size() << "\n";
    dq.pop_back();
    dq.pop_front();
    cout << dq.size() << "\n";
}
```

```

    return 0;
}
/*
1
3
3
1
*/

```

struct(구조체)

커스텀한 구조체를 형성하기 위해 class와 struct를 써야 합니다. 다만 struct만 알아도 충분합니다. 먼저 구조체에 대해서 설명해보겠습니다. 보통의 구조체는 다음과 같은데요.

```

struct Point{
    int y, x;
    Point(int y, int x) : y(y), x(x){}
    Point(){y = -1; x = -1; }
    bool operator < (const Point & a) const{
        if(x == a.x) return y < a.y;
        return x < a.x;
    }
};

```

int y, x;

- 이 구조체의 변수들입니다.

Point(int y, int x) : y(y), x(x){}

- y, x를 받아 생성한다라는 의미입니다. class의 constructor라는 매직메서드를 생각하면 됩니다. 이 구조체를 기반으로 객체를 생성할 때 y, x를 받아 생성한다라는 의미입니다.

Point(){y = -1; x = -1; }

- 만약 y, x가 정해지지 않은 경우 기본값으로 -1, -1를 집어넣는다는 의미입니다.

bool operator < (const Point & a) const{

if(x == a.x) return y < a.y;

return x < a.x;

}

- 해당 구조체를 기반으로 만들어진 객체들끼리 비교해야 하는 경우가 있습니다. 예를 들어 PointA < PointB 처럼 말이죠. 그럴 때 비교하는 "기준"을 잡는 겁니다. 1순위는 x, 2순위는 y를 기반으로 크고 작음을 판단합니다.

```
// 1) 커스텀한 정렬이 필요할 경우
struct Point{
    int y, x;
    Point(int y, int x) : y(y), x(x){}
    Point(){y = -1; x = -1; }
    bool operator < (const Point & a) const{
        if(x == a.x) return y < a.y;
        return x < a.x;
    }
};

// 2) 정렬이 필요하지 않을 경우
struct percent{
    int x, y;
    double w, d, l;
} a[6]; //간단한 struct
```

커스텀한 무언가를 진행하고 싶다면 구조체를 통해서 해야 합니다. 예를 들어 2차원적인 자료구조는 pair를 사용하면 되지만 x, y, z 등.. 여러가지 인자들이 나오고 커스텀한 솔팅이 필요하다면 역시나 구조체를 사용하는 것이 좋습니다. 정렬이 필요하지 않은 경우 2번처럼 그냥 단순하게 구조체를 설정하면 됩니다.

하지만 정렬, 즉, 커스텀정렬이 필요할 때 1번처럼 해야 합니다.

- 커스텀 솔팅이란 x를 1순위로 오름차순, y를 2순위로 내림차순...이렇게 복잡한 정렬을 말합니다.

위의 코드에서 1번 방법은 조금은 복잡하게 오퍼레이터를 사용하는 경우입니다. 1순위로 x를 오름차순 정렬, 2 순위로 y를 기준으로 오름차순 정렬하는 구조체입니다. 저렇게 구조체만 만들어 놓고 해당 구조체를 담은 vector를 정렬, 즉 sort(v.begin(), v.end()) 식으로 한다면 미리 설정해 놓은 정렬기준에 맞춰 정렬이 될 것입니다.

자, 저 **bool operator**을 보죠. 이것은 연산자(operator) 오버로딩입니다. 이는 말 그대로 연산자를 오버로딩(하위 클래스에서 재정의)하는 것이죠. 연산자는 <, >, 등이 있고 이를 오버로딩한다는 것입니다.

예를 들어 숫자 1, 2, 3을 정렬한다고 해보죠. 그냥 1, 2, 3 에는 <라는 연산만이 필요하죠?

하지만 구조체를 정렬해본다고 해볼게요.

```
a.x = 1
b.x = 2
c.x = 3
```

이런식으로 x라는 속성값에 어떠한 값이 담기는데 이걸 그냥 <라는 연산자로 정렬이 불가능하겠죠? 즉, 구조체에 맞는 연산자가 필요하다 이런겁니다.


```

    bool operator < (const Point & a) const{
        if(x == a.x) return y < a.y;
        return x < a.x;
    }

```

지금 보면 들어온 Point형 a에 대해 1순위 x를 기반으로 오름차순, 2순위 y를 기반으로 오름차순 정렬한 것을 볼 수 있습니다.

```

#include<bits/stdc++.h>
using namespace std;
struct Point{
    int y, x;
    Point(int y, int x) : y(y), x(x){}
    Point(){y = -1; x = -1; }
    bool operator < (const Point & a) const{
        if(x == a.x) return y < a.y;
        return x < a.x;
    }
};
vector<Point> v;
int main(){
    for(int i = 10; i >= 1; i--){
        Point a = {i, i};
        v.push_back(a);
    }
    sort(v.begin(), v.end());
    for(auto it : v) cout << it.y << " : " << it.x << '\n';
    return 0;
}
/*
1 : 1
2 : 2
3 : 3
4 : 4
5 : 5
6 : 6
7 : 7
8 : 8
9 : 9
10 : 10
*/

```

자 10, 10 ... 이런식으로 넣었는데도 불구하고 1, 2, 이렇게 정렬이 된 것을 볼 수 있죠?

자, y, x, z라는 속성이 있다고 해봅시다. x를 1순위로 오름차순으로 정렬하고 y가 2순위로 내림차순 z가 3순위로 오름차순 정렬이라는 문제가 있다면 어떻게 해야 할까요?

```
struct Point{
    int y, x, z;
    Point(int y, int x, int z) : y(y), x(x), z(z){}
    Point(){y = -1; x = -1; z = -1; }
    bool operator < (const Point & a) const{
        if(x == a.x) {
            if(y == a.y) return z < a.z;
            return y > a.y;
        }
        return x < a.x;
    }
};
```

이렇게 구조체를 구축해야 겠죠?

이러한 커스텀 정렬이 필요없는 경우라면 2번과 같이 간단히 구조체를 만들면 됩니다. 클래스로 하는 방법도 있지만 struct 하나만으로 충분합니다.

또한 이렇게 구조체 자체내에 bool operator ... 하고 커스텀 정렬을 구현했지만 이를 따로 떼어서 하는 방법도 있습니다.

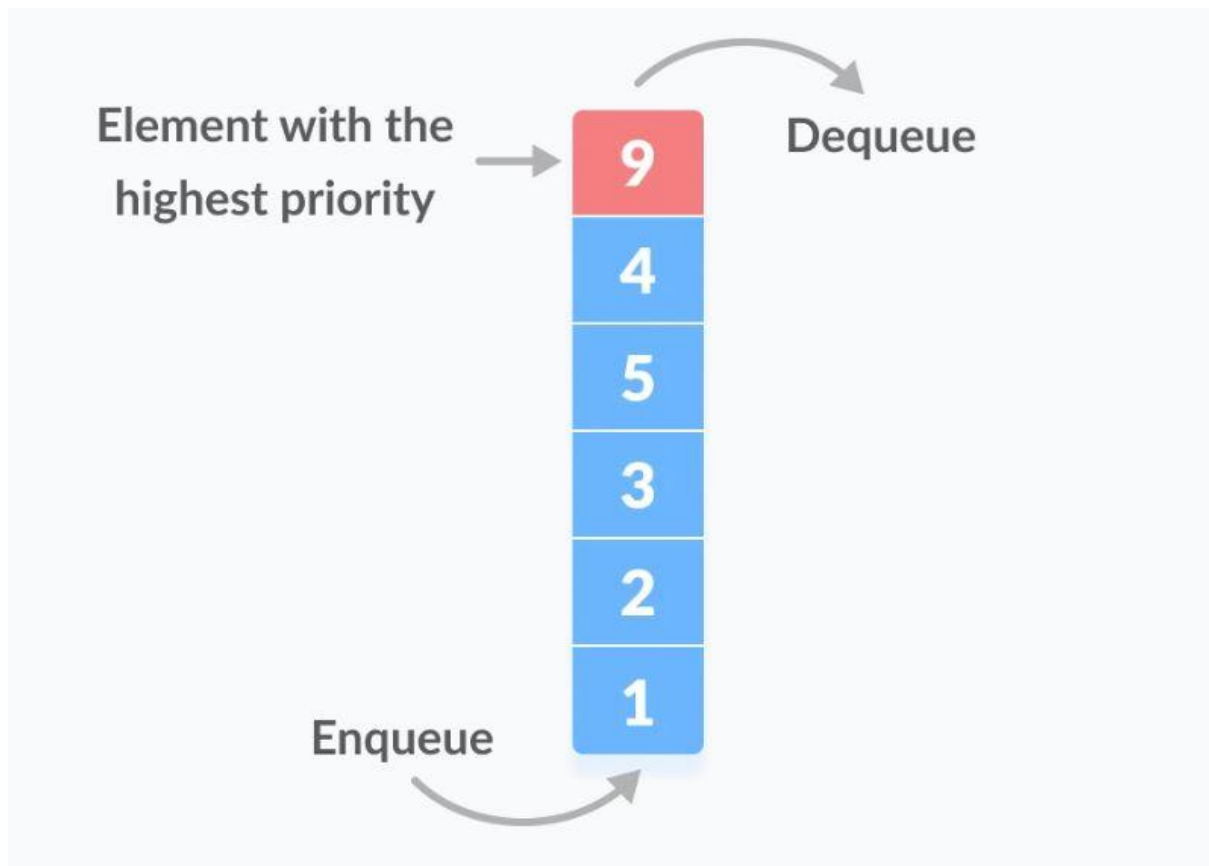
```
#include<bits/stdc++.h>
using namespace std;
struct Point{
    int y, x;
};
bool cmp(const Point & a, const Point & b){
    return a.x > b.x;
}
vector<Point> v;
int main(){
    for(int i = 10; i >= 1; i--){
        v.push_back({i, 10 - i});
    }
    sort(v.begin(), v.end(), cmp);
    for(auto it : v) cout << it.y << " : " << it.x << "\n";
    return 0;
}
/*
1 : 9
```

```

2 : 8
3 : 7
4 : 6
5 : 5
6 : 4
7 : 3
8 : 2
9 : 1
10 : 0
*/

```

priority queue



우선순위 큐입니다. 내부구조는 heap으로 구현되어있으며 주로 다익스트라, 그리디 등에 쓰입니다. 다음 코드 처럼 greater 를 써서 오름차순, less를 써서 내림차순으로 바꿀 수 있습니다.

```

#include <bits/stdc++.h>
using namespace std;
priority_queue<int, vector<int>, greater<int> > pq; //오름차순
//priority_queue<int, vector<int>, less<int> > pq; // 내림차순
int main(){

```

```

    pq.push(5);
    pq.push(4);
    pq.push(3);
    pq.push(2);
    pq.push(1);
    cout << pq.top() << "\n";
    return 0;
}
/*
1
*/

```

int 뿐만 아니라 struct 등 다른 자료구조를 넣어서 할 수 있습니다.

우선순위큐는 priority_queue<자료형, 구현체, 비교연산자> 가 들어가는 것을 볼 수 있습니다.

```

#include <bits/stdc++.h>
using namespace std;
struct Point{
    int y, x;
    Point(int y, int x) : y(y), x(x){}
    Point(){y = -1; x = -1; }
    bool operator < (const Point & a) const{
        return x > a.x;
    }
};

priority_queue<Point> pq;
int main(){
    pq.push({1, 1});
    pq.push({2, 2});
    pq.push({3, 3});
    pq.push({4, 4});
    pq.push({5, 5});
    pq.push({6, 6});
    cout << pq.top().x << "\n";
    return 0;
}
/*
1
*/

```

위 코드를 보면 분명 < 연산자에 x > a.x를 했기 때문에 내림차순으로 정렬되어 6이 출력이 되어야 하는데 1이 출력되죠? 우선순위큐에 커스텀 정렬을 넣을 때 반대로 넣어야 하는 특징 때문입니다.

```
bool operator < (const Point & a) const{
    return x > a.x;
}
```

위 부분을 아래 코드처럼 바꿔보겠습니다.

```
#include <bits/stdc++.h>
using namespace std;
struct Point{
    int y, x;
    Point(int y, int x) : y(y), x(x){}
    Point(){y = -1; x = -1; }
    bool operator < (const Point & a) const{
        return x < a.x;
    }
};

priority_queue<Point> pq;
int main(){
    pq.push({1, 1});
    pq.push({2, 2});
    pq.push({3, 3});
    pq.push({4, 4});
    pq.push({5, 5});
    pq.push({6, 6});
    cout << pq.top().x << "\n";
    return 0;
}
/*
6
*/
```

지금 보시면 $x > a.x$ 가 $x < a.x$ 로 바뀐 모습입니다. 우선순위큐에 커스텀 정렬을 넣을 때는 반대라고 생각하시면 됩니다. 가장 최소를 끄집어 내고 싶은 로직이라면 $>$, 최대라면 $<$ 이런식으로 설정하면 됩니다.

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

```

    int y, x;
};
struct cmp{
    bool operator()(Point a, Point b){
        return a.x < b.x;
    }
};
priority_queue<Point, vector<Point>, cmp> pq;
int main(){
    pq.push({1, 1});
    pq.push({2, 2});
    pq.push({3, 3});
    pq.push({4, 4});
    pq.push({5, 5});
    pq.push({6, 6});
    cout << pq.top().x << "\n";
    return 0;
}
/*
6
*/

```

물론 이렇게도 가능합니다.

자료구조 복잡도 정리

다음은 자주 쓰는 자료 구조의 시간 복잡도를 나타낸 모습입니다. 보통 시간 복잡도를 생각할 때 평균, 그리고 최악의 시간 복잡도를 고려하면서 씁니다.

▼ 표 5-1 자료 구조의 평균 시간 복잡도

자료 구조	접근	탐색	삽입	삭제
배열(array)	$O(1)$	$O(n)$	$O(n)$	$O(n)$
스택(stack)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
큐(queue)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
이중 연결 리스트(doubly linked list)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
해시 테이블(hash table)	$O(1)$	$O(1)$	$O(1)$	$O(1)$
이진 탐색 트리(BST)	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
AVL 트리	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
레드 블랙 트리	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

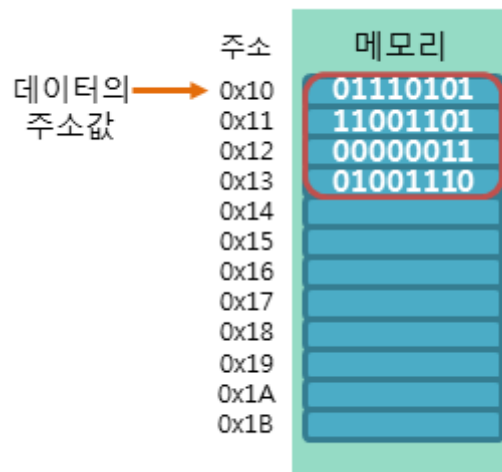
▼ 표 5-2 자료 구조 최악의 시간 복잡도

자료 구조	접근	탐색	삽입	삭제
배열(array)	$O(1)$	$O(n)$	$O(n)$	$O(n)$
스택(stack)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
큐(queue)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
이중 연결 리스트(doubly linked list)	$O(n)$	$O(n)$	$O(1)$	$O(1)$
해시 테이블(hash table)	$O(n)$	$O(n)$	$O(n)$	$O(n)$
이진 탐색 트리(BST)	$O(n)$	$O(n)$	$O(n)$	$O(n)$
AVL 트리	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
레드 블랙 트리	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

1.4 포인터

포인터

데이터의 주소값이란 해당 데이터가 저장된 메모리의 시작 주소를 의미합니다. 주소값은 1바이트 크기의 메모리 공간으로 나누어 표현하는 것을 말합니다.



C++에서 포인터와 연관되어 사용되는 연산자는 다음과 같습니다.

주소연산자 &와 참조 연산자 * 가 있습니다. 주소 연산자는 변수의 이름 앞에 사용하여, 해당 변수의 주소값을 반환합니다. '&'기호는 앰퍼샌드(ampersand)라고 읽습니다.

참조 연산자는 포인터의 이름이나 주소 앞에 사용하여, 포인터에 저장된 주소에 저장되어 있는 값을 반환합니다.

'*'기호는 역참조 연산자로 에스크리티터(asterisk operator)라고도 불립니다. C++에서 *라는 별표 기호는 사용하는 위치에 의해 다양한 용도로 사용됩니다. 이항 연산자로 사용하면 곱셈 연산으로, 포인터의 선언 시에도, 메모리에 접근할 때도 사용됩니다.

아래의 예제를 보면서 이해를 하면 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    string a = "abcda";
    string * b = &a;
    cout << b << "\n";
    cout << *b << "\n";
    return 0;
    /*
    0x6ffdf0
    abcda
    */
}
```

전에 설명했던 map의 예제를 다시보겠습니다.

```
#include <bits/stdc++.h>
```



```

using namespace std;
int v[10];
int main(){
    unordered_map<string, int> umap;
    //이렇게 넣기도 가능하고
    umap.insert({"test1", 1});
    //이렇게 넣을 수도 있습니다.
    umap.emplace("test5", 5);
    //또한 이렇게 변경도 가능, 추가할 수도 있습니다. 아래를 권장합니다.
    umap["test1"] = 4;

    for(auto element : umap){
        cout << element.first << " :: " << element.second << '\n';
    }
    //map의 find메소드는 찾지 못하면 end() 이터레이터를 반환합니다.
    auto search = umap.find("test4");
    if(search != umap.end()){
        cout << "found :" << search->first << " " << (*search).second << '\n';
    }else{
        cout << "not found.." << '\n';
    }
    //이런식으로 바로 int형을 증가시킬 수 있습니다.
    umap["test1"]++;
    cout << umap["test1"] << "\n";

    cout <<umap.size() <<"\n";
    umap.erase("test1");
    cout <<umap.size() <<"\n";

    return 0;
}
/*
test5 :: 5
test1 :: 4
not found..
5
2
1
*/

```

주목할 부분은 아래의 코드입니다. find로 map에서 데이터를 끄집어내면 이터레이터를 반환하고 이는 포인터입니다. 이를 통해 2가지 방법으로 요소를 끄집어낼 수 있습니다.

1. -> 인 화살표 함수를 통해 C++은 객체의 요소에 접근할 수 있습니다. 이 때 포인터를 통해서 사용하는데 문법은 다음과 같습니다.

```
(pointer_name)->(variable_name)
```

2. 에스터리키(*)를 사용해서 값을 끄집어낼 수도 있습니다. 포인터가 주소값이니 *를 통해 값을 반환할 수 있는 것이죠.

```
search -> first << " " << (*search).second
```

지금 보시는 것처럼 search라는 포인터를 기반으로 umap의 키인 first와 umap의 값인 second를 끄집어내는 것을 볼 수 있습니다. 맵에서 키는 first로 접근하고 값은 second로 접근합니다.

이터레이터와 포인터

이터레이터는 컨테이너(배열 등)의 메모리 주소를 가리키는 데 사용되며 주소값을 바로 반환하지 않는 포인터의 일종입니다. 이 이터레이터의 함수 중 많이 쓰는 함수로는 begin()과 end() 두개가 있습니다.

1. begin() : 이 함수는 컨테이너의 시작 위치를 반환하는 데 사용됩니다.
2. end() : 이 함수는 컨테이너의 끝 + 1의 위치를 반환하는 데 사용됩니다.

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

vector<int> v;

int main(){
    for(int i = 1; i <= 5; i++)v.push_back(i);
    for(int i = 0; i < 5; i++){
        cout << i << "번째 요소 : " << *(v.begin() + i) << "\n";
        cout << &*(v.begin() + i) << '\n';
    }
    // cout << v.begin() << '\n'; //에러
}
/*
0번째 요소 : 1
0xba1430
1번째 요소 : 2
0xba1434
2번째 요소 : 3
0xba1438
3번째 요소 : 4
0xba143c
4번째 요소 : 5
```

```
0xba1440
```

```
*/
```

앞의 코드를 보면 vector v에 1이라는 요소를 삽입했고 해당 요소의 주소값을 &*를 통해 추출했으며 *를 통해 주소값을 기반으로 1이라는 값을 추출한 것을 볼 수 있습니다.

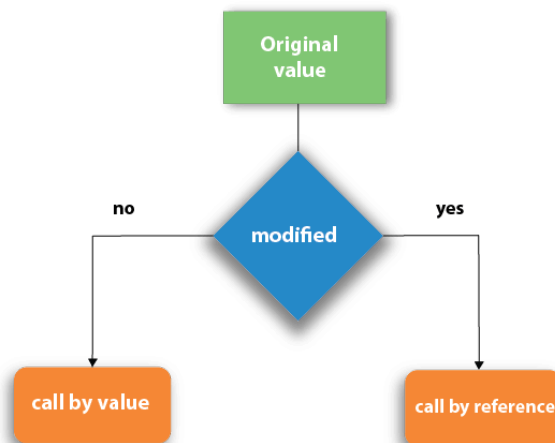
그리고 바로 v.begin()으로 주소값을 출력하려고 하면 에러가 발생하는 것을 볼 수 있습니다. 테스트하고 싶다면 주석을 해제하시면 됩니다. 즉, 이터레이터는 바로 주소값을 반환하지않아서 만약 이터레이터를 기반으로 주소값을 출력하려면 &*를 통해 출력해야 하는 것을 볼 수 있습니다.

call by reference와 value

어떠한 값을 함수로 넘겨서 변하게 만들고 싶을 때는 주소값을 넘겨야 합니다. 그저 값을 넘기면 바뀌지 않습니다.

```
#include<bits/stdc++.h>
using namespace std;
int idx = 2;
// call by reference
void go(int &idx){
    idx = 1;
}
// call by value
void go2(int idx){
    idx = 100;
}
int main(){
    go(idx);
    cout << idx << '\n'; // 1 : 바뀌었다.
    go2(idx);
    cout << idx << '\n'; // 1 : 바뀌지 않았다.
}
```

주소라는 것은 “메모리의 위치”이며 변수에 기록된 값은 그 메모리에 기록된 하나의 값입니다. 그렇기 때문에 만약 어떠한 것을 함수를 기반으로 변화시키고 싶다면 값이 아닌 주소를 넘겨 해당 메모리 위치로 들어가 변화시켜야 합니다.



배열과 포인터



배열의 이름은 배열의 주소값의 첫번째 시작 주소를 가리킵니다.

따라서 아래처럼 넘겨도 값들이 수정됩니다.

```
#include<bits/stdc++.h>
using namespace std;
void go(int a[]){
    a[2] = 100;
}
int a[3] = {1, 2, 3};

int main(){
    go(a);
}
```

```

    for(int i : a) cout << i << '\n';
}
/*
1
2
100
*/

```

1.5 수학

코딩테스트에 주로 나오는 수학적 개념들을 배워봅시다.

순열과 조합

축구선수 12명이 서로(2명씩) 인사하면 몇가지 경우의 수가 나올까요? 답은 66입니다. 왜 66일까요? 경우의 수라고 했을 때 기본적으로 순열과 조합이 생각나야 합니다. 자 그럼 경우의 수를 구하는 방법 중 기초가 되는 순열, 조합에 대해 알아보도록 하겠습니다.

순열

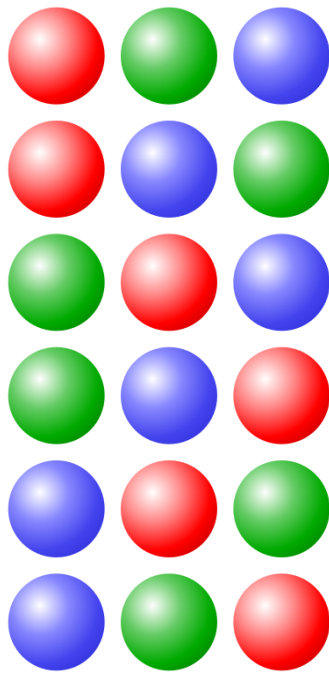
먼저 순열, permutation이란 순서가 정해진 임의의 집합을 다른 순서로 섞는 연산을 말합니다. 그러니까 1, 2, 3이렇게 있을 때 1, 3, 2 이런식으로 다른 순서로 섞는 연산을 순열이라고 합니다. 그리고 n 개의 집합 중 n 개를 고르는 순열의 개수는 $n!$ 이라는 특징을 가지고 있습니다.

예를 들어 3개의 자연수(1, 2, 3)를 이용해 만들 수 있는 3자리 자연수는 몇개 일까요? 123, 132, 213, 231, 312, 321 이렇게 6개 입니다. 그렇다면 3개의 자연수(1, 2, 3)를 이용해 만들 수 있는 1자리 자연수는 몇개일까요? 3개입니다. 전자는 3개중 3개를 뽑는 것이라 $3!$ 이 되고 후자는 3개중 1개를 뽑는 것이라 3이 됩니다.

$${}_n P_r = \frac{n!}{(n-r)!}$$

위와 같은 질문의 답은 위의 공식에 따라 몇개인지 결정이 됩니다. 예를 들어 3개 중 3개를 뽑는다면 $3!/(3-3)!$ 이 되고 3개 중 1개를 뽑는다면 $3!/(3-1)!$ 이 되는 것이죠.

예를 들어 서로다른 색깔을 가진 3개의 공에 대해 3개를 “순서와 관계없이” 뽑는 경우의 수는 어떻게 될까요? 바로 6개고 다음과 같은 그림입니다.



앞의 그림과 같이 순서를 바꿔서 놓으며 경우의 수를 찾는 것을 순열이라고 합니다.

그렇다면 이를 코드로 어떻게 구현할 수 있을까요?

첫번째 방법은 `next_permutation`과 `prev_permutation`을 이용하는 방법입니다. `next_permutation`은 오름차순의 배열을 기반으로 순열을 만들 수 있으며 `prev_permutation`은 그와 반대인 내림차순의 배열을 기반으로 순열을 만들 수 있습니다.

매개변수로는 순열을 만들 범위를 가리키는 `[first, last)`를 집어 넣습니다. 순열을 시작할 범위의 첫번째 주소, 그리고 포함되지 않는 마지막 주소를 넣어서 만듭니다.

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
void printV(vector<int> &v){
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << " ";
    }
    cout << "\n";
}
int main(){
    int a[3] = {1, 2, 3};
```

```

vector<int> v;
for(int i = 0; i < 3; i++)v.push_back(a[i]);
//1, 2, 3부터 오름차순으로 순열을 뽑습니다.
do{
    printV(v);
}while(next_permutation(v.begin(), v.end()));
cout << "-----" << '\n';
v.clear();
for(int i = 2; i >= 0; i--)v.push_back(a[i]);
//3, 2, 1부터 내림차순으로 순열을 뽑습니다.
do{
    printV(v);
}while(prev_permutation(v.begin(), v.end()));
return 0;
}

```

결과

```

1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
-----
3 2 1
3 1 2
2 3 1
2 1 3
1 3 2
1 2 3

```

앞의 코드를 보면 배열의 `begin()`, `end()`를 넣은 것을 볼 수 있는데요.

여기서 `end()`는 해당 리스트의 마지막 요소보다 한칸 뒤의 주소값을 가리킵니다.

들어가는 매개변수가 `[first, last)` 이렇게 들어가니 두번째 인자로는 포함되지 않을 값을 집어넣으면 됩니다.

자, 여기서 배열의 종점인 `end()`를 넣지 않고 다른 걸 넣을 수도 있습니다. 어디서부터 어디까지의 순열을 만드는 것인데 배열의 일부만을 고치고 싶을 때도 있으니까요.

```

do{
    printV(v);
}while(next_permutation(v.begin(), v.begin() + 2));

```

예를 들어 앞의 코드 같은 경우는 배열의 0, 1 만의 순서를 고치게 됩니다. [v.begin(), v.begin() + 2) 이라는 범위를 만든 거니까요.

두번째 방법은 재귀를 이용한 방법입니다.

```
#include <cstdio>
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
int a[3] = {1, 2, 3};
vector<int> v;
void printV(vector<int> &v){
    for(int i = 0; i < v.size(); i++){
        cout << v[i] << " ";
    }
    cout << "\n";
}
void makePermutation(int n, int r, int depth){
    if(r == depth){
        printV(v);
        return;
    }
    for(int i = depth; i < n; i++){
        swap(v[i], v[depth]);
        makePermutation(n, r, depth + 1);
        swap(v[i], v[depth]);
    }
    return;
}
int main(){
    for(int i = 0; i < 3; i++)v.push_back(a[i]);
    makePermutation(3, 3, 0);
    return 0;
}
```

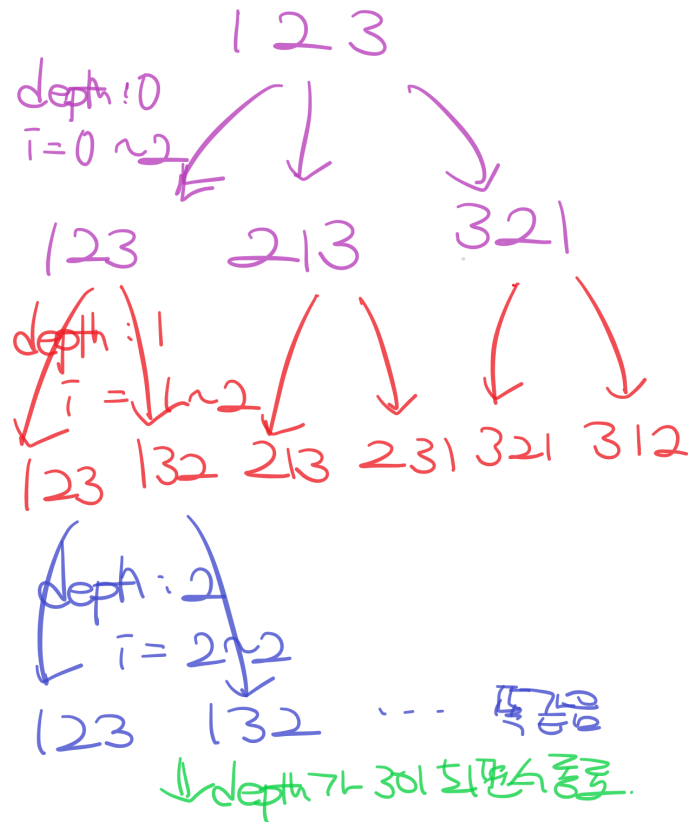
결과

```
1 2 3
1 3 2
2 1 3
```



```
2 3 1
3 1 2
3 2 1
```

그림으로 다시 설명해보면 다음과 같습니다. depth는 트리의 레벨이자 높이가 되며 이런식으로 뻗어나가다가 종료 됩니다.



이해가 안된다면 makePermutation 함수에 `cout << n << " : " << r << " : " << depth << '\n'` 등으로 넣어서 디버깅을 하면서 함수가 어떻게 실행되는지 알아보면 이해가 쉽습니다. 재귀함수는 함수가 함수를 연이어 호출하는 함수를 말합니다. 디버깅하면서 이해하면 쉽습니다. (정말로요.)

조합

자 그렇다면 조합을 하는 방법은 무엇일까요?

조합에서 “순서”는 없습니다. 그저 몇명을 뽑아서 갈 것인가를 쓸 때 조합을 씁니다. 순서따윈 상관없고 오로지 몇명을 “다양하게” 뽑을 때 사용하는 것입니다.

먼저 조합의 공식은 다음과 같습니다.

Formula

$${}_nC_r = \frac{n!}{r!(n-r)!}$$

${}_nC_r$ = number of combinations

n = total number of objects in the set

r = number of choosing objects from the set

예를 들어 5개 중에 3개를 뽑는다고 하면... $5 * 4 / 2$ 가 되어 10개가 되는 것이죠. 이렇게 몇개가 나오는지 알았으니 이제 해당 경우의 수를 구하는 방법에 대해 알아보겠습니다.

1. 재귀함수

이를 구현하는 방법 중 하나는 재귀함수를 이용한 것입니다. 5개 중에서 3개를 뽑는다는 것을 만들어보죠. 인덱스를 출력하는 함수입니다.

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

int n = 5, k = 3, a[5] = {1, 2, 3, 4, 5};
void print(vector<int> b){
    for(int i : b)cout << i << " ";
    cout << '\n';
}
void combi(int start, vector<int> b){
    if(b.size() == k){
        print(b);
        return;
    }
    for(int i = start + 1; i < n; i++){
        b.push_back(i);
        combi(i, b);
        b.pop_back();
    }
    return;
}

int main() {
```

```

    vector<int> b;
    combi(-1, b);
    return 0;
}
/*
0 1 2
0 1 3
0 1 4
0 2 3
0 2 4
0 3 4
1 2 3
1 2 4
1 3 4
2 3 4
*/

```

2. 중첩for문

또한 다음과 같이 이렇게 중첩for문으로도 만들 수 있습니다. r이 작을 때는 이렇게 구현합니다. 아래와 같은 경우는 r이 3개이기 때문에 3중 for문으로 만들었습니다. 만약 저 r이 10이거나 5라면 많은 중첩for문을 만들기는 쉽지 않겠죠? 그럴 때는 위의 재귀함수를 이용하는 것입니다.

```

#include <cstdio>
#include <algorithm>
#include <vector>
#include <iostream>
using namespace std;

int n = 5;
int k = 3;
int a[5] = {1, 2, 3, 4, 5};
int main() {
    for(int i = 0; i < n; i++){
        for(int j = i + 1; j < n; j++){
            for(int k = j + 1; k < n; k++){
                cout << i << " " << j << " " << k << '\n';
            }
        }
    }
    return 0;
}
/*

```

```

0 1 2
0 1 3
0 1 4
0 2 3
0 2 4
0 3 4
1 2 3
1 2 4
1 3 4
2 3 4
*/

```

또한 이는 다음코드와 똑같은 의미를 가집니다. 순서만 다를 뿐이죠. “뽑는 것”은 똑같습니다.

```

#include<bits/stdc++.h>
using namespace std;
int n = 5;
int k = 3;
int a[5] = {1, 2, 3, 4, 5};
int main() {
    for(int i = 0; i < n; i++){
        for(int j = 0; j < i; j++){
            for(int k = 0; k < j; k++){
                cout << i << " " << j << " " << k << '\n';
            }
        }
    }
    return 0;
}
/*
2 1 0
3 1 0
3 2 0
3 2 1
4 1 0
4 2 0
4 2 1
4 3 0
4 3 1
4 3 2
*/

```

만약 r이 2, 즉 2개를 뽑는 것이라면 이렇게 작성할 수 있습니다.

```

#include <stdio>
#include <algorithm>
#include <vector>
#include <iostream>
using namespace std;

int n = 5;
int k = 2;
int a[5] = {1, 2, 3, 4, 5};
int main() {
    for(int i = 0; i < n; i++){
        for(int j = i + 1; j < n; j++){
            cout << i << " " << j << '\n';
        }
    }
    return 0;
}
/*
0 1
0 2
0 3
0 4
1 2
1 3
1 4
2 3
2 4
3 4
*/

```

조합과 순열의 특징

조합과 순열 모두 이러한 특징을 갖습니다.

$${}_nC_r = {}_nC_{(n-r)}$$

즉, nCr 이나 $nC(n - r)$ 이나 똑같다라는 것입니다. 예를 들어 9개 중에 2개를 뽑는 것은 9개 중에 7개를 뽑는 것과 동일하기 때문입니다. (어차피 2개를 뽑으면 나머지 7개가 나오니까요.) 이는 순열도 동일합니다. $nPr = nP(n - r)$

이외에도 파스칼의 삼각형 같은 특징이 있지만 주로 나오진 않아 생략합니다.

- 링크 : 파스칼의 삼각형

https://ko.wikipedia.org/wiki/%ED%8C%8C%EC%8A%A4%EC%B9%BC%EC%9D%98_%EC%82%BC%EA%B0%81%FD%98%95

정수론

최대공약수와 최소공배수

최대공약수 gcd는 다음과 같이 구합니다.

```
int gcd(int a, int b){
    if(a == 0) return b;
    return gcd(b % a, a);
}
```

최소공배수란 lcm은 $(a * b / (a \text{와 } b \text{의 최대공약수}))$ 이며 다음과 같이 구합니다.

```
#include<bits/stdc++.h>
using namespace std;
int gcd(int a, int b){
    if(a == 0) return b;
    return gcd(b % a, a);
}
int lcm(int a, int b){
    return (a * b) / gcd(a, b);
}
int main(){
    int a = 10, b = 12;
    cout << lcm(a, b) << '\n';
    return 0;
}
/*
60
*/
```

*/

모듈러 연산

0. $a \equiv b \pmod n$ 과 $b \equiv c \pmod n$ 은 $a \equiv c \pmod n$ 을 의미
1. $[(a \pmod n) + (b \pmod n)] \pmod n = (a+b) \pmod n$
2. $[(a \pmod n) - (b \pmod n)] \pmod n = (a-b) \pmod n$
3. $[(a \pmod n) * (b \pmod n)] \pmod n = (a*b) \pmod n$

에라토스테네스의 체

소수가 아닌 값들에 대한 불리언 배열을 만들어 소수만을 걸러낼 수 있는 방법입니다.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

```
vector<int> era(int mx_n){
    vector<int> v;
    for(int i = 2; i <= mx_n; i++){
        if(che[i]) continue;
        for(int j = 2*i; j <= mx_n; j += i){
            che[j] = 1;
        }
    }
    for(int i = 2; i <= mx_n; i++) if(che[i] == 0)v.push_back(i);
    return v;
}
```

하지만 이는 배열의 크기가 필요하기 때문에 배열의 크기가 일정 수준(1000만 이상)을 벗어나면 쓰기가 힘듭니다. 이럴 때는 일일이 소수를 판별하는 bool 함수를 만들어주어야 합니다.

코드는 다음과 같습니다.

```

#include<bits/stdc++.h>
using namespace std;
bool check(int n) {
    if(n <= 1) return 0;
    if(n == 2) return 1;
    if(n % 2 == 0) return 0;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) return 0;
    }
    return 1;
}

int main(){
    for(int i = 1; i <= 20; i++){
        if(check(i)){
            cout << i << "는 소수입니다.\n";
        }
    }
    return 0;
}
/*
2는 소수입니다.
3는 소수입니다.
5는 소수입니다.
7는 소수입니다.
11는 소수입니다.
13는 소수입니다.
17는 소수입니다.
19는 소수입니다.
*/

```

등차수열의 합

기본적인 등차수열의 합 정도는 알아야 합니다. 기본적으로 1씩 증가하는 수열의 경우 아래와 같은 합이 나옵니다. 즉, n 이 3이면 6이 나옵니다. $1 + 2 + 3$ 이기 때문이죠.

$$\sum_{k=1}^n k = \frac{n(n+1)}{2}$$

승수 구하기

정수론은 아니나 승수를 구하는 로직이 필요할 때가 있습니다. 2의 2승, 3승 이런 로직 말이죠. 그럴 땐 pow를 사용하면 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n = 4;
    int pow_2 = (int)pow(2, n);
    cout << pow_2 << '\n';
    return 0;
}
/*
16
*/
```

pow 함수는 다음 코드 처럼 double형 인자를 2개를 받고 기본적으로 double을 반환해줍니다. 따라서 int형으로 사용하고 싶다면 (int)로 형변환을 꼭 해주어야 합니다.

```
pow(double base, double exponent);
```

제곱근 구하기

제곱근은 어떻게 구할까요? 예를 들어 9의 제곱근은 3이며 16의 제곱근은 4입니다. sqrt 함수로 구현할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n = 16;
    int ret = (int)sqrt(n);
    cout << ret << '\n';
    return 0;
}
/*
4
*/
```

이는 기본적으로 double형을 매개변수로 받고 double형을 리턴하기 때문에 int로 쓸 거면 int 형변환을 꼭 해주어야 합니다.

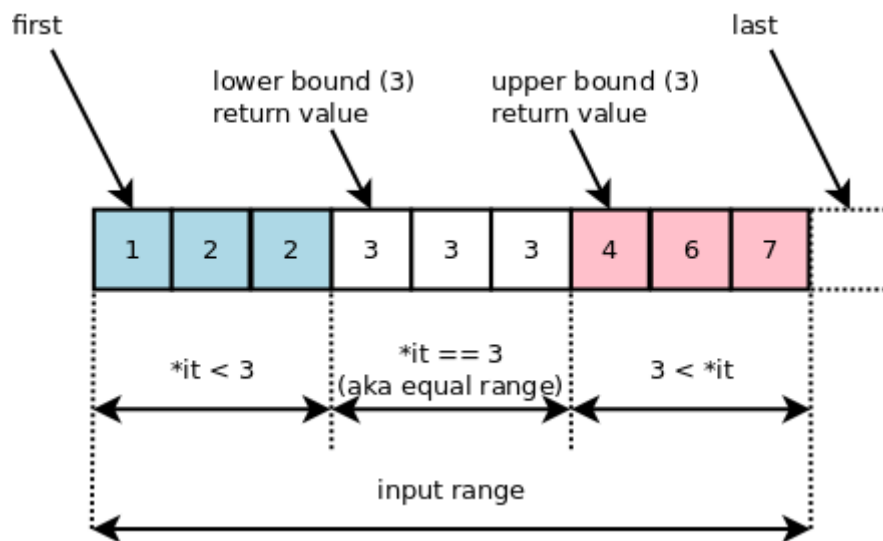
```
sqrt(double num);
```

1.6 질문으로 배우는 코딩테스트에 자주나오는 필수로직

코딩테스트에 자주 나오며 꼭 알아야 하는 로직, 그리고 앞서 배운 것들을 질문을 통해 훈련합니다. 아래부터 나오는 여러가지 질문과 답이 나옵니다. 바로 답을 보는 것이 아니라 한번 코드로 구현을 생각해보고 답을 보면서 훈련해봅시다.

lower_bound와 upper_bound

예를 들어 정렬된 배열에서 어떤 값이 나오는 지점이나 어떤 값이 나오기전의 위치를 반환하려면 어떻게 해야 할까요? 또한 이분탐색을 쉽게 함수로 구현하려면 어떻게 해야할까요?



이를 위한 쉬운 함수인 lower_bound와 upper_bound를 알아봅시다.

```
#include <cstdio>
#include <vector>
#include <algorithm>
using namespace std;
int main(){
    vector<int> v;
    int a[5] = {1, 2, 2, 2, 3};
    for(int i = 0; i < 5; i++){
        v.push_back(a[i]);
    }
    int x = 2;
    int c = (int)(upper_bound(v.begin(),v.end(),x) -
lower_bound(v.begin(),v.end(),x));
    int f = (int)(lower_bound(v.begin(),v.end(),x) - v.begin());
    int t = (int)(upper_bound(v.begin(),v.end(),x) - v.begin());
```

```

    int f2 = *lower_bound(v.begin(),v.end(),x);
    int t2 = *upper_bound(v.begin(),v.end(),x);
    printf("%d의 갯수 : %d, 시작되는 점 : %d 끝나는 점 : %d\n", x, c, f,
t);
    printf("lower bound가 시작되는 점의 값 : %d, upper bound가 시작되는
점의 값 : %d\n", f2, t2);

    c = (int)(upper_bound(a, a + 5, x) - lower_bound(a, a + 5, x));
    f = (int)(lower_bound(a, a + 5, x) - a);
    t = (int)(upper_bound(a, a + 5, x) - a);
    f2 = *lower_bound(a, a + 5, x);
    t2 = *upper_bound(a, a + 5, x);
    printf("%d의 갯수 : %d, 시작되는 점 : %d 끝나는 점 : %d\n", x, c, f,
t);
    printf("lower bound가 시작되는 점의 값 : %d, upper bound가 시작되는
점의 값 : %d\n", f2, t2);

    return 0;
}
/*
2의 갯수 : 3, 시작되는 점 : 1 끝나는 점 : 4
lower bound가 시작되는 점의 값 : 2, upper bound가 시작되는 점의 값 : 3
2의 갯수 : 3, 시작되는 점 : 1 끝나는 점 : 4
lower bound가 시작되는 점의 값 : 2, upper bound가 시작되는 점의 값 : 3
*/

```

lower_bound는 0번째 배열의 원소부터 찾아서 어떠한 값의 "이상이 되는 위치"를 반환합니다.

upper_bound는 그 값이 시작되기 전의 위치를 반환합니다. 예를 들어 1, 2, 2, 3, 4이라는 배열에서 2를 찾는다면 4, 3 순으로 찾고 어라? 2를 발견했다? 그렇다면 그 전의 값인 3의 위치를 반환하는 것입니다. 근데 만약에 4를 찾는다면.. 그 전의 값인 5번째 요소라는 값을 반환합니다. 이 때 반환되는 값은 이터레이터이기 때문에 배열의 처음 주소값인 v.begin() 또는 a[0],을 빼주어서 int형으로 몇번째인지를 파악할 수 있습니다.

또한 반환되는 것은 주소값이 반환되기 때문에 그 주소값이 몇번째임을 알려면 주소값의 첫번째 값을 빼주면 된다는 것입니다.

참고 : 이터레이터와 end()

이터레이터는 STL안의 데이터들을 가리키는 포인터입니다. begin()은 해당 리스트의 처음 요소의 주소값을 가리키고 end()는 해당 리스트의 마지막 요소보다 한칸 뒤의 주소값을 가리킵니다. 네 v[i]

!= v.end())라는 코드를 많이 보셨을 텐데요. 리스트를 다 순회하고 리스트의 끝에 도착했다는 것을 가리킵니다.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main(){
    vector<int> a;
    for(int i=1 ; i<= 3; i++)a.push_back(i);
    for(int i=5 ; i<= 10; i++)a.push_back(i);
    cout << lower_bound(a.begin(), a.end(), 4) - a.begin() << "\n"; // 3
    return 0;
}
```

또 하나의 예제입니다. 4를 찾으려고 하니 3을 반환하는 것을 볼 수 있습니다. 1, 2, 3, 5의 5를 가리키는 인덱스를 반환합니다. 즉, 4의 이상인 첫번째 지점을 반환하죠?

```
#include <bits/stdc++.h>

using namespace std;
vector<int> v;
int main(){
    for(int i = 2; i <= 5; i++)v.push_back(i);
    v.push_back(7);
    // 2 3 4 5 7
    cout << upper_bound(v.begin(), v.end(), 6) - v.begin() << "\n";
    cout << lower_bound(v.begin(), v.end(), 6) - v.begin() << "\n";
    cout << upper_bound(v.begin(), v.end(), 9) - v.begin() << "\n";
    cout << lower_bound(v.begin(), v.end(), 9) - v.begin() << "\n";
    cout << upper_bound(v.begin(), v.end(), 0) - v.begin() << "\n";
    cout << lower_bound(v.begin(), v.end(), 0) - v.begin() << "\n";
    /*
    4
    4
    5
    5
    0
    0
    */
}
```

만약 원소를 찾을 때 못찾을 경우 위 코드 처럼 반환하게 됩니다. 값이 없다면 그 근방지점을 반환하는 것을 봐주세요. 추후 이분탐색을 배울 때 이분탐색의 결과도 이와 비슷하게 출력되게 됩니다.

시계방향과 반시계방향 회전

시계방향, 반시계 방향으로 회전해야 하는 로직을 짤다면 어떻게 해야 할까요?

```
#include<bits/stdc++.h>
using namespace std;
vector<int> v;
int main(){
    for (int i=1; i<10; ++i) v.push_back(i); // 1 2 3 4 5 6 7 8 9

    //rotate(v.begin(),v.begin()+1, v.end());
    // 2 3 4 5 6 7 8 9 1 앞으로 할 땐 이렇게
    rotate(v.begin(),v.begin() + v.size() - 1, v.end());
    // 9 1 2 3 4 5 6 7 8 뒤로 갈 땐 이렇게
    for (std::vector<int>::iterator it=v.begin(); it!=v.end(); ++it)
        std::cout << ' ' << *it;
    std::cout << '\n';
}
```

여기서 rotate 함수에 들어가는 인자는 모두 이터레이터이며 first, middle, last라고 부릅니다. first와 last 사이에 있는 부분배열에서 middle이 가리키는 요소가 first로 가며 회전하는 것을 의미합니다. 예를 들어 v.begin에서 v.end는 배열 전체를 가리키며 여기서 v.begin + 1이 v.begin으로 밀려 간다는 것을 의미하니 앞으로 한칸 회전이 되겠죠?

배열의 합, accumulate

배열의 합을 쉽고 빠르게 구해주는 함수가 있습니다.

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

int main(){
    vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int sum = accumulate(v.begin(), v.end(), 0);
    cout << sum << '\n'; // 55
}
```

배열 중 가장 큰 요소, max_element

배열 중 가장 큰 요소를 추출하는 함수가 있습니다.

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

int main(){
    vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int a = *max_element(v.begin(), v.end());
    cout << a << '\n'; // 10
}
```

가장 작은 요소는 min_element라는 함수를 사용하면 됩니다.

배열 부분 회전

예를 들어 1번부터 4번까지의 배열만 한칸씩 앞으로 회전하는 로직을 구현하려면 어떻게 해야 할까요?

```
#include <bits/stdc++.h>
using namespace std;
vector<int> v;
int main(){
    for(int i= 1; i <= 6; i++) v.push_back(i);
    int i = 1;
    int temp = v[i];
    v[i] = v[i + 1];
    v[i + 1] = v[i + 2];
    v[i + 2] = v[i + 3];
    v[i + 3] = temp;
    for(int i : v) cout << i << ' ';
}
/*
1 3 4 5 2 6
*/
```

함수인자로 전달해서 변수 수정하기

인자로 전달해서 변수를 수정하는 2가지 방법은 무엇일까요?

```
#include <stdio>
void b(int a){
```

```

    a = 2;
}
void b2(int & a){
    a = 2;
}
void b3(int * a){
    *a = 3;
}
int main() {
    int a = 1;
    b(a);
    printf("%d\n", a); // 1
    b2(a);
    printf("%d\n", a); // 2
    b3(&a);
    printf("%d\n", a); // 3
    return 0;
}

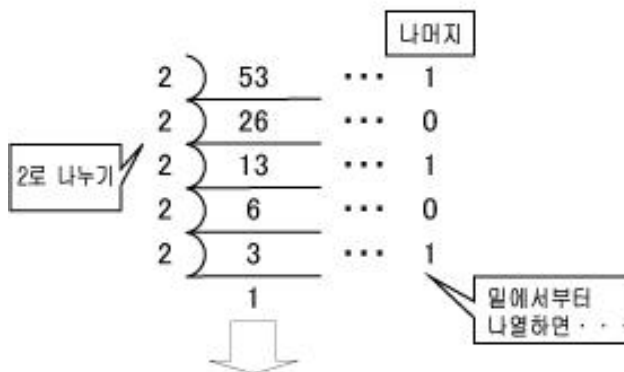
```

n진법 변환

어떤 숫자 n에서 이를 b진법으로 바꾸는 방법은?

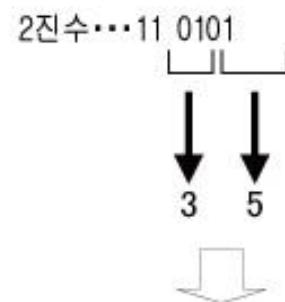
2진법을 만들 때 나누기를 하고 몫을 이용하죠? 이런 그림을 많이 보셨을 겁니다.

(예) 10진수 53을 2진수로 변환



53 (10진수) = 11 0101 (2진수)

(예) 2진수 110101을 16진수로 변환



11 0101 (2진수) = 35 (16진수)

아래의 코드는 10진법을 2진법으로 바꾸는 코드입니다.

```

#include<bits/stdc++.h>
using namespace std;
vector<int> v;
int main(){
    int n = 100;

```

```

int b = 2;
while(n > 1){
    v.push_back(n % b);
    n /= b;
}
if(n == 1)v.push_back(1);
reverse(v.begin(), v.end());
for(int a : v) {
    if(a >= 10)cout << char(a + 55);
    else cout << a;
}
return 0;
}

```

여기서 b를 3으로 바꾸면 10진법을 3진법으로 바꾸는 방법이 됩니다. 진법들을 테스트하다가면서 익히면 됩니다.

내림차순 정렬

vector<int>에서 그냥 sort를 하면 오름차순 정렬이라고 배웠습니다. 내림차순 정렬한다면 어떻게 될까요?

```

#include <bits/stdc++.h>
using namespace std;
vector<int> v;
int main(){
    for(int i= 1; i <= 6; i++) v.push_back(i);
    sort(v.begin(), v.end(), greater<int>());
    for(int i : v) cout << i << ' ';
}
/*
6 5 4 3 2 1
*/

```

만약 pair<int, int>를 기반으로 한다면 이렇게 하면 됩니다. greater<int>의 int는 들어가는 “형”을 뜻합니다.

```

#include <bits/stdc++.h>
using namespace std;
vector<pair<int, int>> v;
int main(){
    for(int i= 1; i <= 6; i++) v.push_back({i, i});
    sort(v.begin(), v.end(), greater<pair<int, int>>());
    for(auto i : v) cout << i.first << ' ';
}

```



```

}
/*
6 5 4 3 2 1
*/

```

커스텀 정렬

커스텀정렬 bool cmp(int a, int b)를 구현해서 vector<int>를 내림차순 정렬해보세요.

```

#include <bits/stdc++.h>
using namespace std;
vector<int> v;
bool cmp(int a, int b){
    return a > b;
}
int main(){
    for(int i= 1; i <= 6; i++) v.push_back(i);
    sort(v.begin(), v.end(), cmp);
    for(int i : v) cout << i << ' ';
}
/*
6 5 4 3 2 1
*/

```

2차원배열을 수정하는 함수

2차원배열을 인자로 넘겨서 수정하는 함수를 구현하시오.

```

#include<bits/stdc++.h>
using namespace std;
void b(int a[][5]) {
    a[0][4] = 44;
}
int main(){
    int a[3][5] = {
        { 1, 2, 3, 4, 5 },
        { 6, 7, 8, 9, 10 },
        { 11, 12, 13, 14, 15 }
    };
    b(a);
    cout << a[0][4] << "\n";
    return 0;
}

```

2차원배열을 회전하는 함수

2차원배열을 90도 회전시키는 함수를 구현하시오.

```
// 왼쪽으로 90도
void rotate90(vector<vector<int>> &key){
    int m = key.size();
    vector<vector<int>> temp(m, vector<int>(m, 0));
    for(int i = 0; i < m; i++){
        for(int j = 0; j < m; j++){
            temp[i][j] = key[j][m - i - 1];
        }
    }
    key = temp;
    return;
}

// 오른쪽으로 90도
void rotate90(vector<vector<int>> &key){
    int m = key.size();
    vector<vector<int>> temp(m, vector<int>(m, 0));
    for(int i = 0; i < m; i++){
        for(int j = 0; j < m; j++){
            temp[i][j] = key[m - j - 1][i];
        }
    }
    key = temp;
    return;
}
```

1.7 코드를 구축하는 법

알고리즘 코드를 구축할 때는 간결하고 빠르게 짜야 합니다. 내가 10분만에 짰는데 다른사람은 1시간만에 구축했다면? 저를 뽑겠죠? 시간은 중요한 요소입니다.

지역변수와 변수명을 간결하게.

그렇기 때문에 기존의 프로그래밍을 할 때 배웠던 변수명을 신경쓰고 지역변수를 권장하는 것과는 상반됩니다. 그래서 아래와 같이 구축하는 것을 추천드립니다.

```
#include<bits/stdc++.h>
```

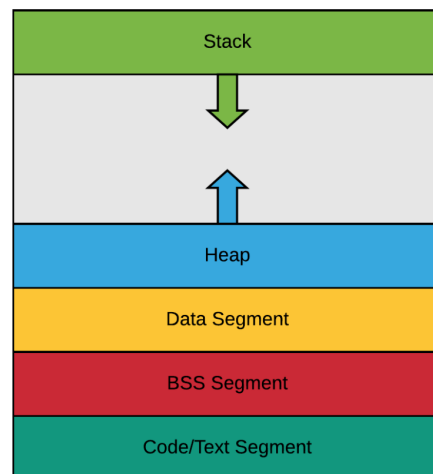
```
using namespace std;
int a, b, c;
int main(){
    a = 2;
    cout << a << " : " << b << " : " << c << "\n";
    return 0;
}
```

전역변수는 지역변수보다 더 많은 크기의 요소를 담을 수 있고 초기화하지 않은 전역변수는 지역변수가 쓰레기값으로 초기화되는 것과는 달리 예상할 수 있는 0으로 초기화되기 때문에 전역변수를 많이 써야 합니다. 그리고 변수명은 항상 간결하게 씁시다. count는 cnt로 result는 ret으로 하는 등 변수명을 짧게 써야 합니다.

동적할당과 정적할당

정적할당

정적할당의 경우 BSS segment와 Data segment, Code segment로 나뉘집니다.



BSS segment는 초기화가 되지않은 전역변수, static으로 선언한 변수들, 0으로 초기화가 되는 것들을 뜻하고 Data segment은 초기화가 8, 1..로 할당된 변수를 뜻합니다. 이렇게 나뉘지는 이유는 프로그램이 컴파일되고 Object코드가 되고 링킹이 되어 프로그램이 되서 실행될 때 Object코드의 효율성을 위해서이지요. 어차피 0으로 초기화를 하기 때문에 사이즈만을 ObjectCode에 넣으면 되는 것이고 그러기 위해서 나눕니다. 참고로 const ... 로 선언한 변수는 Data segment에 저장되게 됩니다.

동적할당

동적할당은 Stack과 Heap으로 나뉘집니다.

Stack 메모리영역은 함수가 실행함에 따라 늘어돌고 줄어돌게 됩니다. 지역변수, 매개변수, 함수가 실행하는 그 시점의 주소값을 담고 함수가 실행할 때 잠깐 사용되는 영역입니다. BSS segment의 경우 처음 선언하게 되면 프로그램 실행이 끝날 때까지 남아있고 이건 그렇지 않습니다.

Heap은 동적으로 할당되는 변수들을 담습니다. 프로그램이 실행, 런타임될 때 할당되므로 모든 애플리케이션이 함께 사용할 수 있는 메모리 영역이 되며 꼭 해제(deallocate)를 해줘야 합니다.

참고 : malloc

malloc의 경우 void형 포인터를 반환하니 int형으로 변환할 때는 (int *)를 해줘야 합니다. vector같은건 당연히 동적할당이 되며 내부적으로 heap영역을 사용합니다. 또한 메모리는 프로그램이 시작될 때 할당이 되며 프로그램이 끝나면 OS에게 돌려주게 됩니다.

배열의 경우 조금 더 넓게

```
int a[10004];
```

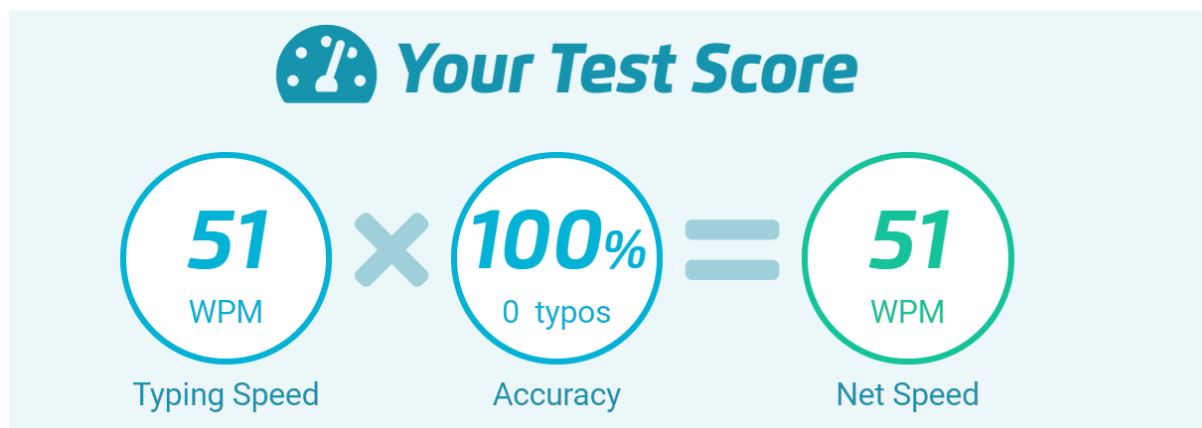
예를 들어 10000의 최대범위를 가지는 문제가 있다면 이런식으로 4정도 여유공간을 주는 게 좋습니다. 예를 들어 10000이라면 10000 + 4로 10004로 해야 하는 것이죠. 이를 통해 오버플로에 대한 신경을 덜 쓰게 만듭니다.

빠른 속도로 코딩하자!

코딩테스트는 시간을 가르는 시험입니다. 빠르게 코딩을 하는 것은 전체적으로 문제를 다 푸는데 도움이 됩니다. 아래의 사이트로 가면 타이핑을 연습할 수 있습니다.

<https://www.typingtest.com/>

참고로 필자의 타이핑 실력은 51wpm입니다.



외워야 할 숫자

시간복잡도를 어렵잡아 계산할 때 자주 나오는 수들. 이정도는 외워주자.

- $10! = 3628800$
- $2^{10} = 1024$
- $3^{10} = 59049$

1.8 자주 실수하는 것들

bits/stdc++.h에서 기본적으로 사용할 수 없는 변수명

bits/stdc++.h는 모든 라이브러리를 로드하기 때문에 다른 라이브러리에서 전역변수로 쓰고 있는 변수명을 쓰지 못합니다.

따라서 해당 변수명 같은 경우 그럴 경우에는 define을 써주어야 합니다. define을 걸어서 다른 임의의 문자열로 바꾸어 버린다음 다음 코드처럼 구축하면 됩니다.

```
#define y1 xoxoxoxo
```

대표적으로 쓰지 못하는 변수명으로는 다음과 같습니다.

- y1
- time
- prev
- next

예시는 다음과 같습니다.

```
#include <bits/stdc++.h>
using namespace std;
#define prev aaa
#define next aaaa
int prev[4];
int main() {
    cout << prev[0] << '\n';

    return 0;
}
```

또한 보통 함수로 사용되는 이름이나 매개변수로 들어가는 이터레이터의 이름 등은 define을 걸어도 쓰지 못합니다. 즉, 해당 부분은 변수명으로 선언하면 안됩니다.

예를 들어 다음코드에서 주석처리를 해제해서 sort를 하게 되면 에러가 발생합니다. sort의 end라는 이터레이터를 사용하기 때문이죠.

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

```
vector<int> v;
int main() {
    for(int i : {1, 2, 3, 4, 5}) v.push_back(i);
    //sort(v.begin(), v.end());
}
```

입출력 싱크

```
ios_base::sync_with_stdio(false);
cin.tie(NULL);
cout.tie(NULL);
```

위 코드는 입출력싱크를 맞추는 것을 해제한다는 의미입니다. c++에서 대표적인 입출력 함수로는 cin, cout, scanf, printf가 있고 이는 시스템콜관련 함수이므로 같이 쓰일 때는 입출력싱크를 맞춰주어야 합니다.

또한 보통 cin, cout과 scanf, printf 중 scanf, printf가 빠릅니다.왜냐면 cin, cout은 c라이브러리인 stdio의 버퍼와 동기화하느라 시간을 소비하기 때문이죠.

이 때 cin, cout의 시간을 빠르게 해주는 방법이 있습니다. 바로 위코드처럼 싱크를 해제하여 버퍼 동기화를 하지 않는 것이죠. 하지만 이렇게 했을 때 반드시!!cin, cout를 쓸 때 scanf, printf를 쓰지 말아야 합니다. 동기가 풀리니 모르는 사이드 이펙트가 발생할 수도 있는 것이죠.

스택오버플로(stack overflow)



함수의 호출이 무한히 반복되면 발생합니다. 함수가 무한적으로 호출하나 확인해야 합니다. 함수는 스택에 쌓이면서 실행됩니다. 스택의 모든 공간을 다 차지하고 난 후 더 이상의 여유 공간이 없을 때

또 다시 스택 프레임을 저장하게 되면, 해당 데이터는 스택 영역을 넘어가서 저장되게 되며 이러한 에러가 발생합니다.

변수 초기화 문제

항상 변수가 초기화가 되어있나 확인해야 합니다.

실수형 연산

부동소수점 소수표현은 언제나 제한된 정확도를 가지게 됩니다. 부동소수점 값들이 정확한 값이 아니기 때문입니다. 따라서 `==` 등의 연산을 사용하는 것은 거의 불가능합니다. 따라서 `if(fabs(a-b) < 1e-9)` 이런식으로 해야 되며 보통은 실수형연산은 되도록 안하는게 정신건강에 이롭습니다.

문자열 크기 선언

예를 들어 100개짜리 문자를 입력받는다. 또한 이것 `string`이 아니라 `char[]` 로 한다면 `char[101]`로 선언해야 합니다. C++, C에서 문자는 null로 종료되는 것이 원칙이므로 마지막에 무조건 널문자인 `'\0'`에 해당하는 바이트가 붙습니다

```
//문제 : 입력되는 문자열의 길이는 최대 100
char str[100]; // Bad
char str[101]; // Good
```

참조 에러(reference Error)

queue나 stack에서 top이나 pop 연산을 할 때 항상 size를 체크해야 합니다. 아래코드 처럼 구축해야 합니다. size가 없을 때 참조에러가 발생할 수 있습니다.

```
if(q.size() && q.top == value)
```

UB(Undefined Behavior)

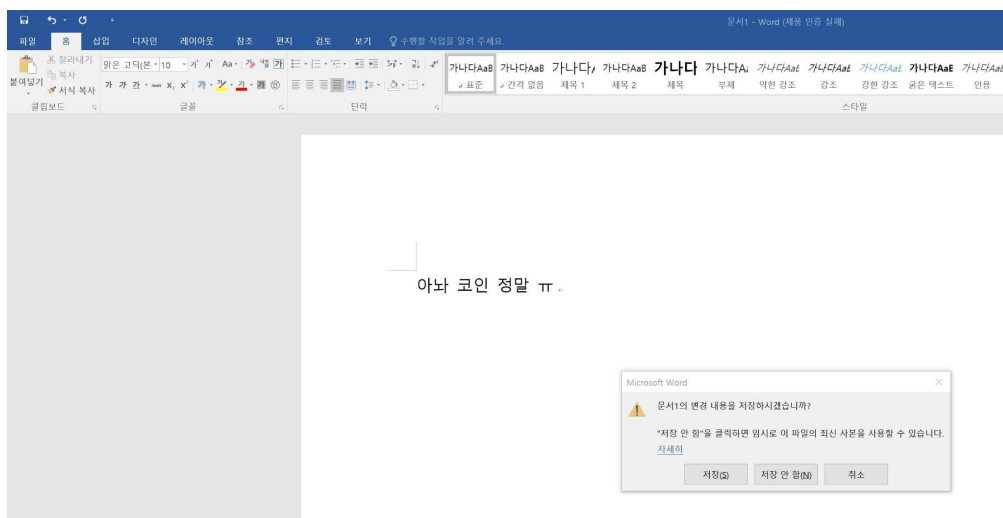
배열 인덱스 밖으로 벗어난 곳을 참조하는 경우가 대표적이며 예상치 못한 결과가 나타났을 때 UB가 발생했다고들 합니다.

endl보다는 “\n”을 써라.

endl보다는 ‘\n’을 쓰는 것이 속도측면에서 좋습니다. endl은 ‘\n’ + 버퍼플러시라고 생각하면 됩니다. 출력 싱크를 보다 “정확하게” 만들어줍니다.

여기서 버퍼플러시란 임시 저장 영역에서 컴퓨터의 영구 메모리로 컴퓨터 데이터를 전송하는 것을 말합니다.

파일을 변경하면 한 컴퓨터 화면에서 볼 수 있는 변경 사항이 일시적으로 버퍼에 저장되고 사용자에게 작업에 의해 하드디스크라는 영구저장소로 플러시될지, 소멸될지가 정해집니다.



예를 들어 워드문서를 열어 임시 파일을 생성했을 때 그냥 “저장하지 않고 닫기”를 하게 되면 자동으로 소멸됩니다. 그러나 저장하게 되면 해당 문서에 대한 변경 사항이 버퍼에서 하드 디스크의 영구 저장소로 플러시됩니다.

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    //버퍼플러시 전
    for (int i = 1; i <= 5; ++i){
        cout << i << "\n";
        this_thread::sleep_for(chrono::seconds(1));
    }
    cout << endl;
    //버퍼 플러시 후
    for (int i = 1; i <= 5; ++i){
        cout << i << "\n" << flush;
        this_thread::sleep_for(chrono::seconds(1));
    }
}
```



```

    cout << endl;

    //버퍼 플러시 후 : 위와 동일한 코드
    for (int i = 1; i <= 5; ++i){
        cout << i << endl;
        this_thread::sleep_for(chrono::seconds(1));
    }
    cout << endl;
    return 0;
}
/*
1
2
3
4
5

1
2
3
4
5

1
2
3
4
5
*/

```

위의 코드를 보면 << "\n"; 와 "\n" << flush; 와 << endl;가 있습니다.

"\n" << flush; 와 << endl;는 같다고 보시면 됩니다. 즉, endl은 "\n"에 플러시가 추가된 것이죠.

<< "\n";의 경우 한개씩 1, 2, 3, 4, 5가 나와야 하는데 “어떤 경우에는” 한번에 출력이 될 수도 있습니다. 플러시는 이를 무조건 순차적으로 출력되게 해주는 것이죠. 왜냐면 콘솔창으로 바로 “출력을 플러시”해주는 역할을 하기 때문입니다.

이 차이입니다.

그러나 코딩테스트에는 이러한 출력의 정확한 “시간적 차이”는 중요하지않습니다. 어쨌든 1, 2, 3, 4, 5 순차적으로 출력이 되기 때문에 좀 더 빠른 것을 써야 하는 코딩테스트 특성상 버퍼플러시가 들어가있지 않는 “\n”을 쓰는 것이 좋습니다.

코딩테스트는 빠르게 푸는 것이 중요하다는 사실을 잊지 마세요. :)

1.9 알고리즘을 공부하는 자세

올바른 과정

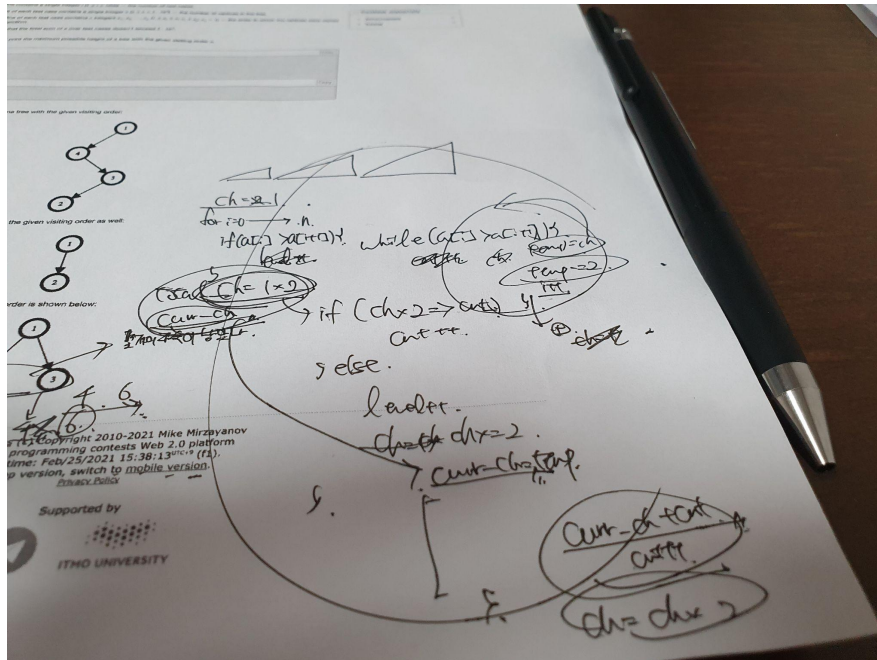
혼자 알고리즘을 공부할 때는 항상 핸드폰을 끄고 초집중해서 공부를 하시는 것을 추천드립니다. 또한 하루 3 ~ 4시간 이상은 하셔야 실력이 늡니다. 다음은 푸는 방법입니다.

1. 고민 : 1 ~ 2시간 이내
2. 문제풀이
3. 답지 보며 자신의 코드와 비교
4. 만약 틀릴 경우 일주일 내내 틀린 문제를 처음부터 다시 풀기(이 문제를 처음 봤을 때 30분내에 풀정도로)

다양하게 풀 수 있을까?

다양하게 풀 생각을 해야 합니다. 어떠한 문제를 예를 들어 void형 함수를 구현해서 풀었다고 해봅시다. 그 다음에는 int형으로 구현을 하거나 `vector<int>` 형을 리턴하는 함수 등 어떠한 것을 다양하게 리턴하는 함수로 만들어서 푸는 훈련 또한 진행하셔야 합니다.

손코딩하라.



* 필자도 이렇게 손코딩합니다.

손코딩은 필수입니다. 어느정도 난이도의 문제는 손코딩을 하지 않고 풀 수도 있겠지만 먼저 이렇게 풀겠다 감을 잡고 들어가는데 있어서 손코딩을 하시는 게 좋습니다. 또한 문제를 만약에 못 풀었을경우에도 손코딩으로 복습하는것도 추천드립니다. 손코딩이라는 것은 별거 아닙니다. 자신이 이해하는 수준으로 코드를 손으로 쓰는 것을 말합니다. 앞의 그림처럼 저가 같은 경우는 for반복문을 for i = 0 -> n 이런식으로 표시해서 코딩을 합니다. “자신이 알아들을 수 있는” 방법으로 하시면 됩니다.

1.10 마치며

이렇게 C++의 기초를 다뤄보았습니다. 이외에도 보시면 좋은 링크 몇개를 남깁니다.

- UB : <https://www.secmem.org/blog/2020/01/17/c-c++-and-ub/>
- 정렬알고리즘 정리 : <https://blog.naver.com/jhc9639/221338179774>