

10주 완성 알고리즘 교안

version.23.05.26

해당 교안을 무단으로 복제, 업로드, 배포, 도용할 경우
저작권법 제 97조의 저작재산권침해죄에 해당하며
5년이하의 징역 또는 5천만원 이하의 벌금에 처해질 수 있습니다.

목차

C++ 10주완성 코딩테스트	9
1.1 C++ 프로그램 설치	9
Window에서 시작하기	9
DevC++이 안될 때 해결방법	12
Mac에서 시작하기	12
brew	13
bits/stdc++.h	14
VS Code의 run task를 활용한 실행방법	19
1.2 기본	22
예제로 이해하는 C++	22
typedef	23
define	24
STL	24
알고리즘	24
컨테이너	25
이터레이터	25
펑터	25
1.3 입력과 출력	26
입력	26
cin	26
scanf	27
scanf를 활용해 실수타입을 정수타입을 받아보기	28
getline	29
출력	31
cout	31
cout의 실수 타입 출력	31
printf	32
1.4 타입과 타입 변환	33
타입	34
1. void : 리턴하는 값이 없다.	34
2. char, 문자	35
3. string, 문자열	36
+=	38
begin()	38
end()	38
size()	38
insert(위치, 문자열)	38

erase(위치, 크기)	38
pop_back()	39
find(문자열)	39
substr(위치, 크기)	39
아스키코드와 문자열	39
아스키코드	40
reverse()	41
split()	42
범위기반 for 루프	44
atoi(s.c_str())	44
4. bool, 참과 거짓	45
5. int, 4바이트짜리 정수	46
int 연산	46
const 키워드	47
오버플로	47
언더플로	47
6. long long, 8바이트짜리 정수	48
7. double, 실수 타입	48
8. unsigned long long, 8바이트짜리 양의 정수	49
pair와 tuple	49
auto 타입	51
타입변환	52
double을 int형으로 만들기	52
타입변환시 주의할점	54
문자를 숫자로, 숫자를 문자로	54
1.5 메모리와 포인터	55
메모리	55
포인터	57
포인터의 개념	58
포인터의 크기	59
역참조 연산자	59
array to pointer decay	60
프로세스 메모리 구조와 정적할당과 동적할당	61
정적할당	62
동적할당	63
1.6 이터레이터	64
begin()	65
end()	65
advance(iterator, cnt)	65

Q. 이터레이터와 포인터의 차이	65
Q. 이터레이터 = 일반화된 포인터 ?	66
1.7 함수	66
fill()과 memset()	66
fill()	66
왜 fill()로 전체초기화를 해야할까?	68
memset()	69
쓰지 말아야 할 초기화 방법 {0, }	70
memcpy()와 copy()	71
memcpy()	71
copy()	74
sort()	75
unique()	78
lower_bound() 와 upper_bound()	80
accumulate()	84
max_element()	85
min_element()	85
1.8 자료구조	86
vector	86
push_back()	88
pop_back()	88
erase()	88
find(from, to, value)	89
clear()	89
fill(from, to, value)	89
범위기반 for 루프	89
vector의 정적할당	90
2차원 배열	91
Array	92
2차원배열과 탐색을 빠르게 하는 팁	93
list	94
싱글연결리스트	95
이중연결리스트	96
원형연결리스트	96
원형싱글연결리스트	96
원형이중연결리스트	96
push_front(value)	97
push_back(value)	97
insert(idx , value)	97

erase(idx)	98
pop_front()	98
pop_back()	98
front()	98
back()	98
clear()	98
map	100
insert({key , value})	101
[key] = value	102
[key]	102
size()	102
erase(key);	102
find(key)	102
for(auto it : mp)	102
for(auto it = mp.begin(); it != mp.end(); it++)	102
mp.clear();	102
맵을 쓸 때 주의할 점	102
unordered_map	105
set	106
Q. set과 unique 중 어떤 것을 써야 할까?	106
multiset	109
stack	110
push(value)	111
pop()	111
top()	112
size()	112
queue	112
push(value)	113
pop()	113
size()	113
front()	113
deque	113
struct	114
랄로 구조체 정의하기	115
Point 구조체 정의하기	116
Q. 왜 오버라이딩이 아니라 오버로딩일까?	118
구조체 기반 sort를 사용할 때 주의 할 점	118
3개의 멤버변수 정렬하기	120
vector에다 struct 넣고 정렬하기	120

priority queue	122
int형 우선순위큐	123
구조체를 담은 우선순위큐	123
자료구조 시간복잡도 정리	125
1.9 값의 의한 호출과 참조에 의한 호출	126
매개변수	126
값에 의한 호출	126
참조에 의한 호출	127
참조에 의한 호출로 넘겨야 할 때	129
성능에 의한 시간초과 예	129
1.10 배열 수정하기	130
Array의 요소 수정하기	130
2차원 배열 수정하기	131
vector	131
array	131
1.11 재귀함수와 수학	132
재귀함수	132
순열과 조합	134
순열	134
next_permutation과 prev_permutation	136
next_permutation() 사용시 주의할 점	138
재귀를 이용한 순열	140
재귀를 이용한 순열 - 디버깅코드	141
조합	142
재귀를 이용한 조합	143
중첩for문	144
조합의 특징 : $nCr = nC(n - r)$	146
정수론	147
최대공약수와 최소공배수	147
모듈러 연산	148
에라토스테네스의 체	148
등차수열의 합	150
등비수열의 합	151
승수	152
제곱근 구하기	152
1.12 코딩테스트 필수로직	153
1차원 배열 회전	153
rotate()를 이용한 방법	154
반시계방향 구축	154

시계방향 구축	156
직접 구현하는 방법	157
2차원 배열 회전	158
2차원 배열 대칭	161
n진법 변환	162
1.13 코딩테스트 팁	164
지역변수 보다는 전역변수를, 변수명을 간결하게.	164
지역변수 보다는 전역변수를	164
변수명은 간결하게	165
배열의 경우 조금 더 넓게	165
초기값은 답의 범위 밖에서.	166
빠른 속도로 코딩하자	166
이정도 숫자는 외우자	167
1.14 맞왜틀 팁	168
bits/stdc++.h에서 기본적으로 사용할 수 없는 변수명	168
입출력 싱크	169
스택오버플로	170
변수 초기화 문제	170
실수형 연산의 제한된 정확도	171
문자열 크기 선언	172
참조 에러	172
UB	173
endl보다는 “\n”을 써라.	173
구현문제를 잘 푸는 방법	175
1.15 자주 묻는 질문	175
Q. 삼성코딩테스트에서 bits/stdc++.h를 쓸 수 있나요?	175
iostream의 자주 쓰는 함수	176
string.h의 자주 쓰는 함수	177
algorithm의 자주 쓰는 함수	177
Q. 따닥따닥 붙어있는 것을 어떻게 입력받죠?	177
1.string으로 변환	178
2.scanf로 받기	179
3.char타입 & cin	180
Q. 문제에서 입력의 끝을 정하지 않은 경우	181
1안) scanf로 할 때	182
2안) cin으로 할 때	182
Q. process exit call이란?	182
Q. 코테에서 입력이 2차원 배열로 주어졌어요. 어떻게 하죠?	184
1.16 알고리즘을 공부하는 자세	185

집중하자.	185
다양하게 풀 수 없을까?	185
손코딩하라.	186
1.17 재밌게 공부하는 방법	187

C++ 10주완성 코딩테스트

알고리즘을 위한 C++을 시작해보겠습니다. C++은 어려운 언어이지만 알고리즘을 위한 C++은 쉽습니다. C++이 아닌 다른 언어를 배운 적이 있다면 쉽게 다가갈 수 있으며 최소 3일, 최대 2주로 계획을 잡고 차근차근 배워 나갑시다.

C++이 코딩테스트 언어로 좋은 이유는 필자의 다음 영상을 참고해주세요.

- <https://youtu.be/qR44fv4D8C0>

1.1 C++ 프로그램 설치

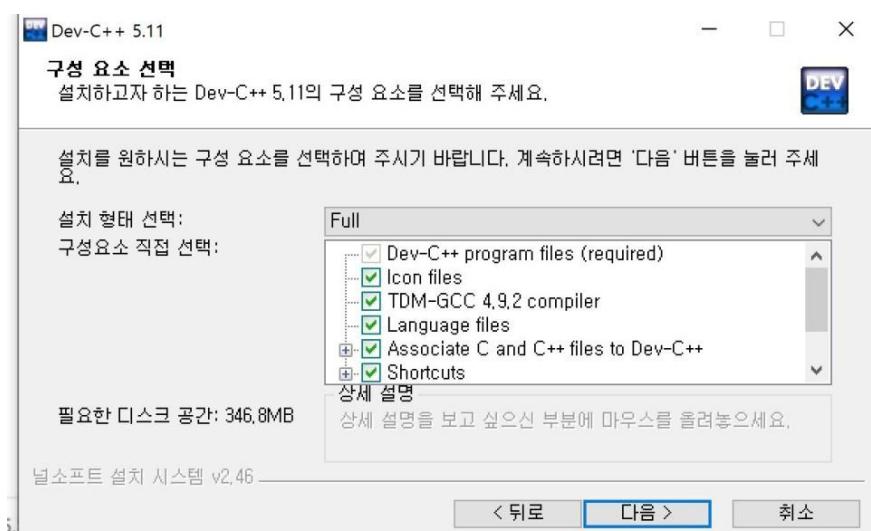
C++을 위한 에디터, 컴파일러 등을 설치해보겠습니다. 이는 OS마다 조금 다릅니다.

Window에서 시작하기

Window에서는 DevC++ 에디터를 추천합니다. 자체 컴파일러를 지원해줘서 컴퓨터에 별도의 gcc를 설치 하지 않아도 되며 OS에 종속적이지 않아 디버깅하기가 편리합니다. gcc란 GNU 프로젝트에서 개발된 컴파일러로써 C, C++, Object-C 등 여러 언어들에 대한 컴파일을 지원합니다. 다음 링크로 들어가 다운로드를 받습니다.

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

다음 그림처럼 모든 체크박스에 체크를 하면서 설치를 하면 됩니다.



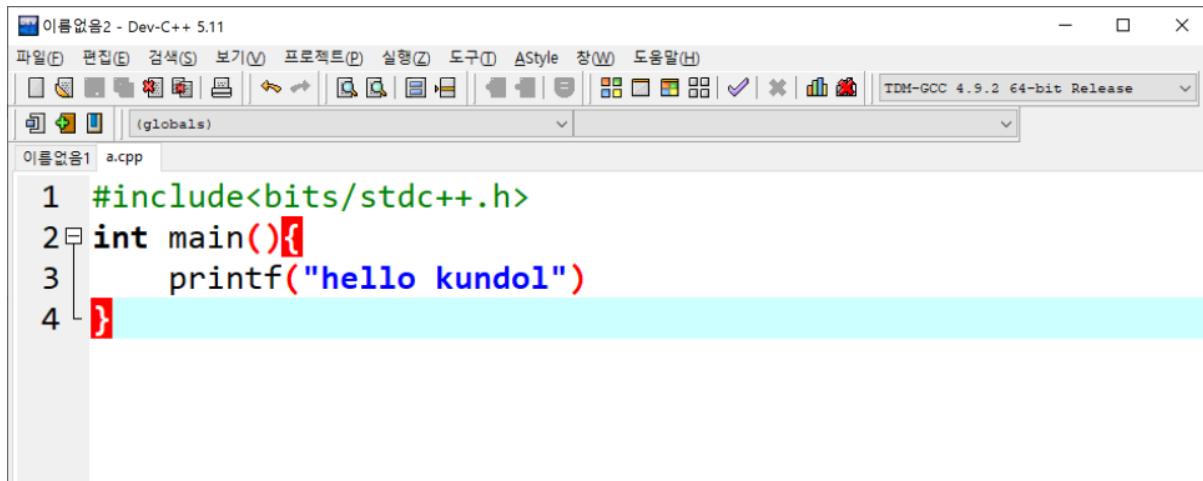
그 다음, 실행 시킨 이후 왼쪽 상단의 [파일] > [새로만들기] > [소스파일] 클릭을 합니다.



다음 코드를 적어봅시다.

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

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



그 다음 저 네모를 클릭합니다. 저 네모는 컴파일 후 프로그램을 실행시키는 아이콘입니다.

The screenshot shows the Dev-C++ IDE interface. The title bar says "이름없음2 - Dev-C++ 5.11". The menu bar includes 파일(F), 편집(E), 검색(S), 보기(V), 프로젝트(P), 실행(R), 도구(I), AStyle, 창(W), 도움말(H). The toolbar has various icons. The status bar at the bottom right says "TDM-GCC 4.9.2 64-bit Release". The code editor window contains a file named "a.cpp" with the following code:

```
1 #include<bits/stdc++.h>
2 int main(){
3     printf("hello kundol")
4 }
```

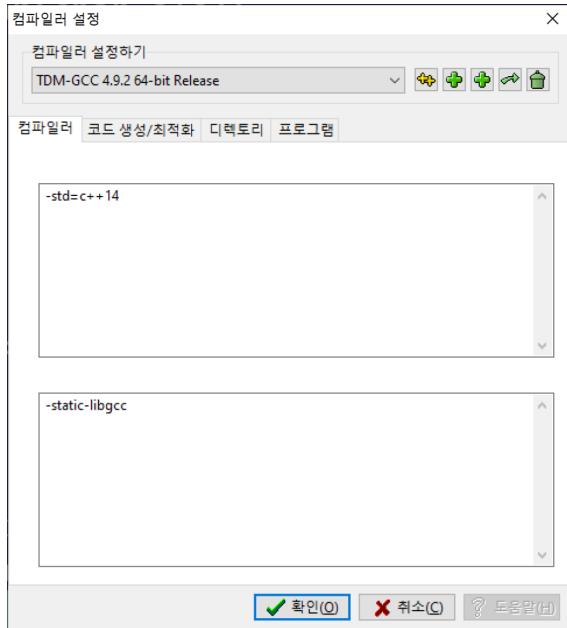
짠, 우리의 첫번째 C++ 프로그램이 완성되었습니다.

The screenshot shows a terminal window titled "C:\Users\jsieu\OneDrive\바탕 화면\wa.exe". The command "hello kundol" is entered. The output shows the program's execution: "Process exited after 0.03001 seconds with return value 0" and "계속하려면 아무 키나 누르십시오 . . ." (Press any key to continue...).

하지만 한가지 더 준비사항이 남아있습니다. [도구] > [컴파일러 설정]에 들어가서 아래 그림처럼 C++14버전으로 컴파일하게 위와 같이 설정해 줍니다. 아래 글을 복사합시다.

```
-std=c++14  
-static-libgcc
```

그리고 다음과 같이 붙여넣기를 각각 해주면 됩니다 .



윈도우에서 C++를 할 모든 준비가 완료되었습니다.

DevC++이 안될 때 해결방법

가끔 DevC++로 프로그램을 실행시키면 failed to execute라고 뜨면서 안될 때가 있습니다.
이 경우 다음과 같은 방법으로 시도해봅시다.

1. 드라이브 수정 : A드라이브가 아닌 다른 드라이브로 파일 경로를 수정해봅시다.
2. 파일 경로 수정 : 공백이 포함된 파일 경로를 수정해봅시다.

ex)

```
이전 경로 c:/Users/jorge santos/programa1.cpp  
새 경로 c:/Users/jorgesantos/programa1.cpp
```

Mac에서 시작하기

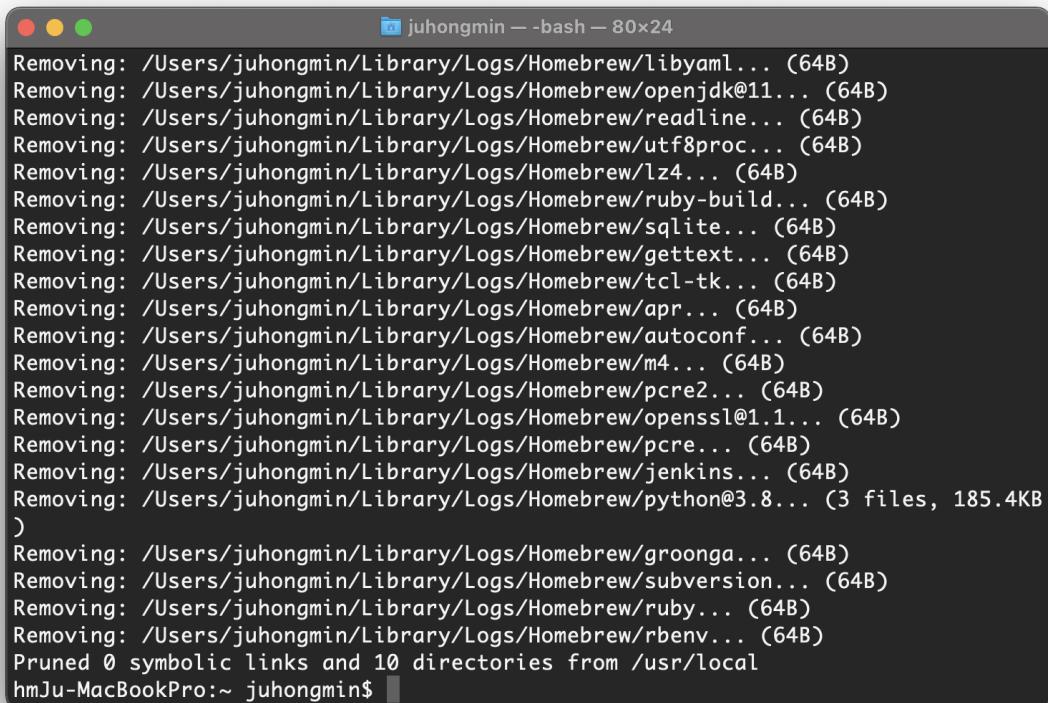
맥에서는 보통 xcode 또는 VS Code 편집기를 통해 진행합니다.

이 강의에서는 VS Code로 편집기를 추천하며 VS Code설치를 해보겠습니다.



먼저 brew를 통해 gcc를 설치합니다.

```
brew install gcc
```



A screenshot of a macOS terminal window titled "juhongmin — bash — 80x24". The window contains a list of files and directories being removed by Homebrew. The log starts with "Removing:" followed by various paths such as "/Users/juhongmin/Library/Logs/Homebrew/libyaml...", "/Users/juhongmin/Library/Logs/Homebrew/openjdk@11...", and so on. It continues with removing "/Users/juhongmin/Library/Logs/Homebrew/readline...", "/Users/juhongmin/Library/Logs/Homebrew/utf8proc...", and other packages like lz4, ruby-build, sqlite, gettext, tcl-tk, apr, autoconf, m4, pcre2, openssl, pcre, jenkins, python@3.8, groonga, subversion, ruby, and rbenv. The log concludes with "Pruned 0 symbolic links and 10 directories from /usr/local". The prompt at the bottom is "hmJu-MacBookPro:~ juhongmin\$".

brew



brew는 macOS에서 편하게 패키지를 관리해주는 애플리케이션입니다.

보통 맥에서 어떠한 패키지를 설치하고 싶다면 이를 통해 설치합니다.

[참고] GCC란 유닉스/리눅스 계열 플랫폼의 표준 컴파일러이며 C(gcc), C++(g++), Objective-C(gobjc), Fortran(gfortran), Ada(gnat), Go(gccgo), D(gdc)의 컴파일을 지원합니다.

bits/stdc++.h

우리는 bits/stdc++.h를 쓸 예정인데요. Mac은 window의 Dev C++와는 달리 bits/stdc++.h를 include하려면 별도의 작업이 필요합니다.

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

아래의 코드들을 모두 복사하거나 아래 링크로 가서 코드를 모두 복사합니다.

<https://raw.githubusercontent.com/wnghdcife/wnghdcife.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 <iostream>
#include <ios>
#include <iostfwd>
#include <iostream>
#include <iostream>
#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>

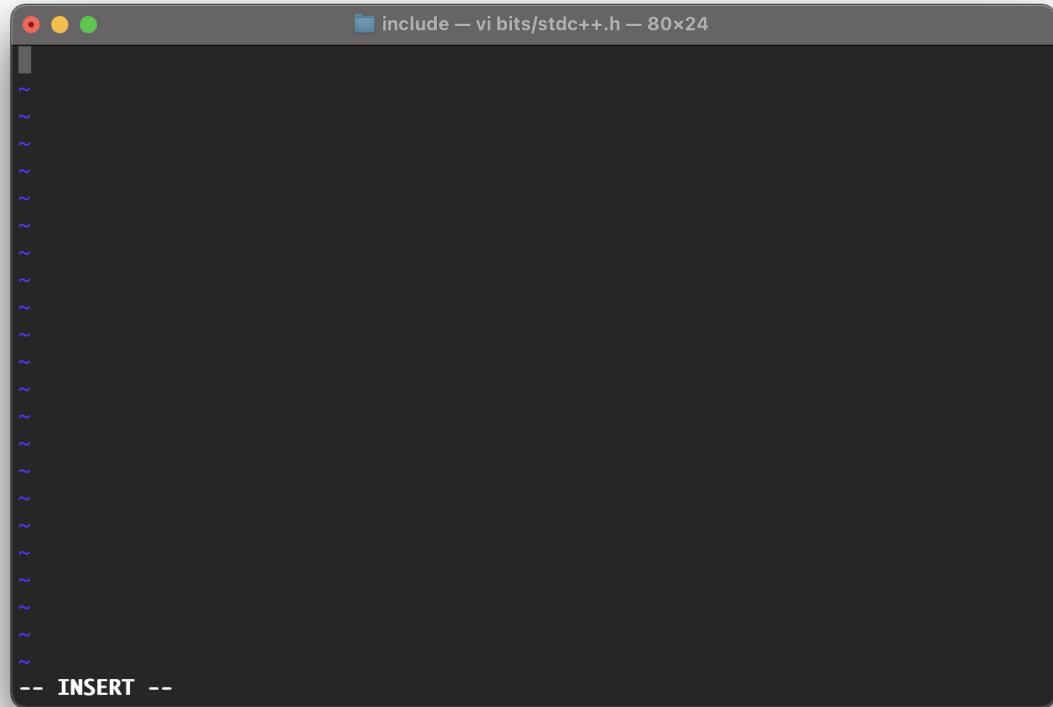
#if __cplusplus >= 201103L
#include <array>
#include <atomic>
#include <chrono>
#include <condition_variable>
#include <forward_list>
#include <future>
```

```
#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
```

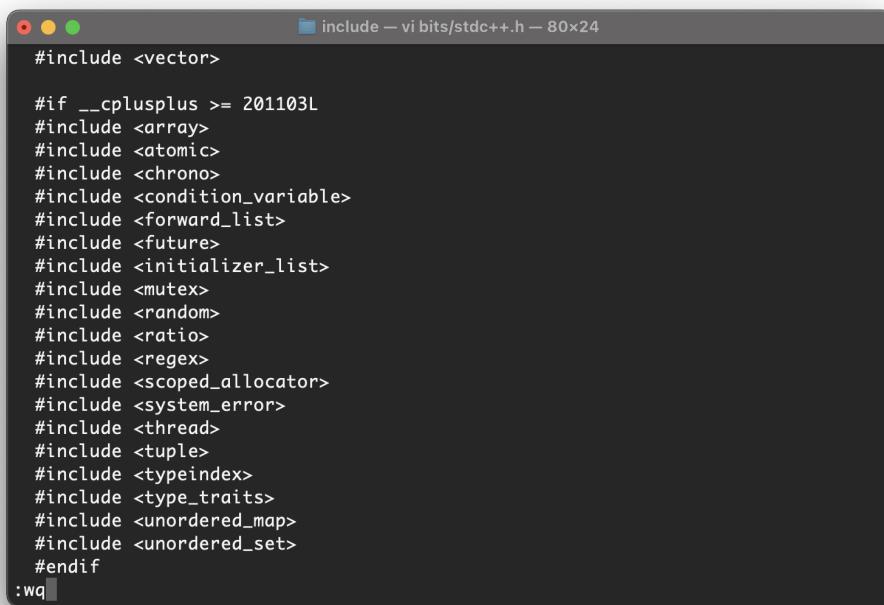
그리고 아래의 명령어를 구동시킵니다.

```
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 끝입니다.

드디어 맥에서 bits/stdc++.h를 쓸 준비가 완료되었습니다.

참고로 m1 맥북은 경로를 다르게 설정해야 할 수도 있습니다. 이는 m1 맥북이라고 다 안되는게 아닙니다. 필자의 노트북은 m1 맥북이며 앞의 설명만으로 준비가 되었습니다. 그러나 안되는 분들도 있는데요.

```
cd /Library/Developer/CommandLineTools/usr/include
mkdir bits
cd bits
vi stdc++.h
```

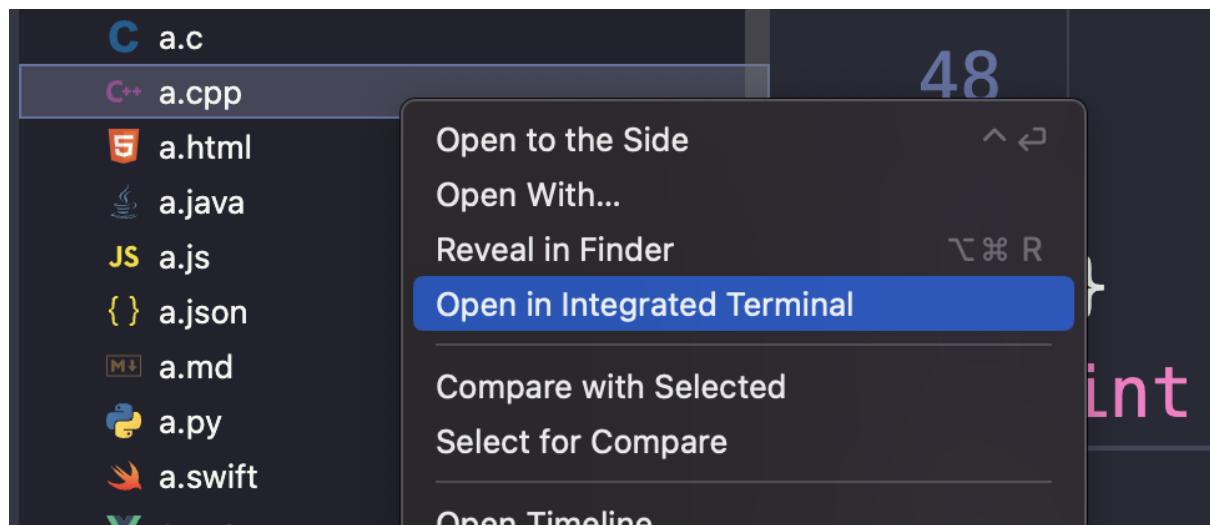
앞의 명령어들을 통해 include폴더를 들어간다음. bits 이름을 가진 폴더를 만들어 앞의 과정을 진행하면 됩니다. 명령어가 동작이 안될 때는 sudo mkdir 등으로 앞에 sudo를 붙여서 실행시켜주세요.

이 후 어떠한 경로내에서 a.cpp라는 파일을 만든 이 후 다음 코드를 작성합니다.

a.cpp

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

그 이후 해당 경로 내에서 터미널을 열어놓습니다.



열려진 명령어창에 다음의 컴파일명령어를 실행합니다.

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

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

```
./test.out
```

그 후 앞의 실행명령어로 실행하면 끝입니다. 우리의 첫번째 C++ 프로그램이 완성되었습니다!

코드가 수정된다면 앞의 컴파일 명령어를 다시 치고 실행 명령어를 쳐야 합니다. 다만, 해당 명령어를 매번 키보드로 쳐서 입력할 필요는 없습니다.

먼저 컴파일명령어는 길기 때문에 어딘가 파일에 저장해놓고 복붙을 하는게 좋습니다. 필자같은경우 a.md 파일에 이렇게 복붙하려고 저장해놓습니다.

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

그리고 나서 복붙해서 입력할 때도 티이 있습니다.

```
• (base) zagabi@zagabiui-MacBookPro ccc % g++ -std=c++14 -Wall a.cpp -o test.out
◦ (base) zagabi@zagabiui-MacBookPro ccc % ./test.out
```

필자 같은 경우 앞의 그림처럼 명령어들을 쳐 놓고 키보드로 위버튼 ↑ 을 누르면 전에 사용했던 코드가 그대로 나옵니다. 즉, 매번 키보드로 저 코드를 치는 것이 아니라 위버튼 ↑ 만으로 해당 코드가 나오게 해 a.cpp 만을 만들어 놓고 다양한 문제들을 빠르게 풀 수 있습니다.

VS Code의 run task를 활용한 실행방법

명령어말고도 VS Code의 run task를 활용하는 방법도 있습니다. (필자는 이 방법은 추천드리지 않습니다. 오히려 더 복잡합니다.)

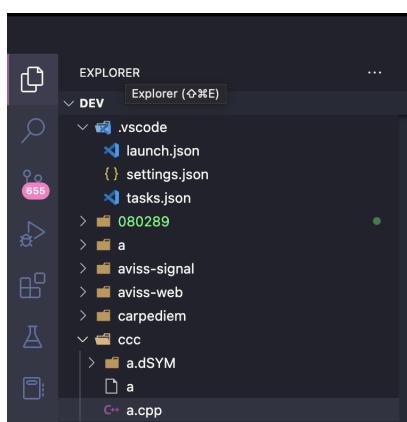
일단 clang이 설치되어있나 확인합니다.

```
clang --version
```

만약 설치가 안되어있다면 다음 명령어로 설치합니다.

```
xcode-select --install
```

그 후 .vscode의 task.json을 수정합니다.



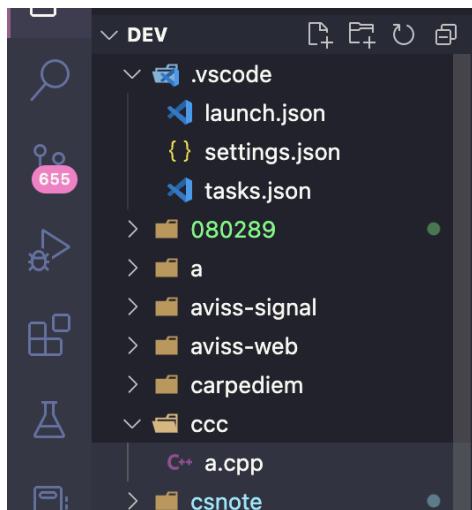
task.json에 이 부분을 추가합니다.

```
{  
    "type": "shell",  
    "label": "C/C++: clang++ build active file",  
    "command": "/usr/bin/clang++",  
    "args": [  
        "-std=c++17",  
        "-stdlib=libc++",  
        "-g",  
        "${file}",  
        "-o",  
        "${fileDirname}/${fileBasenameNoExtension}"  
    ],  
    "options": {  
        "cwd": "${workspaceFolder}"  
    },  
    "problemMatcher": ["$gcc"],  
    "group": {  
        "kind": "build",  
        "isDefault": true  
    },  
    "detail": "Task generated by Debugger."  
},
```

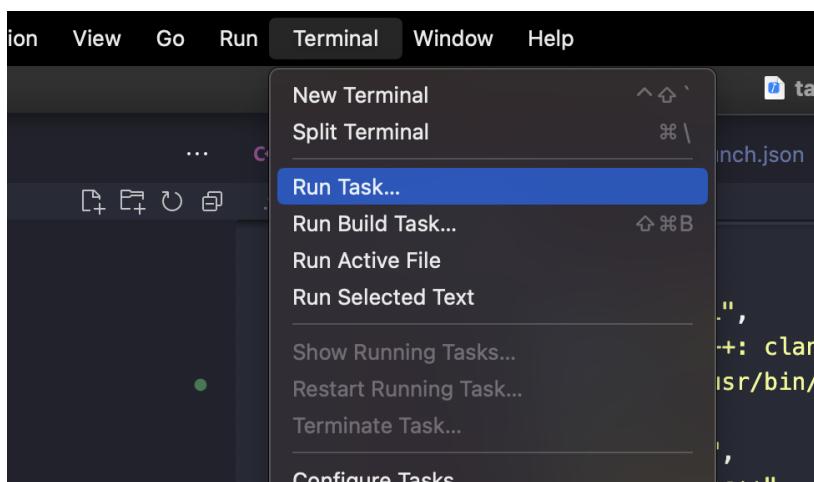
```
2     "tasks": [  
3         {  
4             "type": "shell",  
5             "label": "C/C++: clang++ build active file",  
6             "command": "/usr/bin/clang++",  
7             "args": [  
8                 "-std=c++17",  
9                 "-stdlib=libc++",  
10                "-g",  
11                "${file}",  
12                "-o",  
13                "${fileDirname}/${fileBasenameNoExtension}"  
14            ],  
15            "options": {  
16                "cwd": "${workspaceFolder}"  
17            },  
18            "problemMatcher": ["$gcc"],  
19            "group": {  
20                "kind": "build",  
21                "isDefault": true  
22            },  
23            "detail": "Task generated by Debugger."  
24        },  
25    ]
```

앞의 그림처럼 추가된 모습입니다.

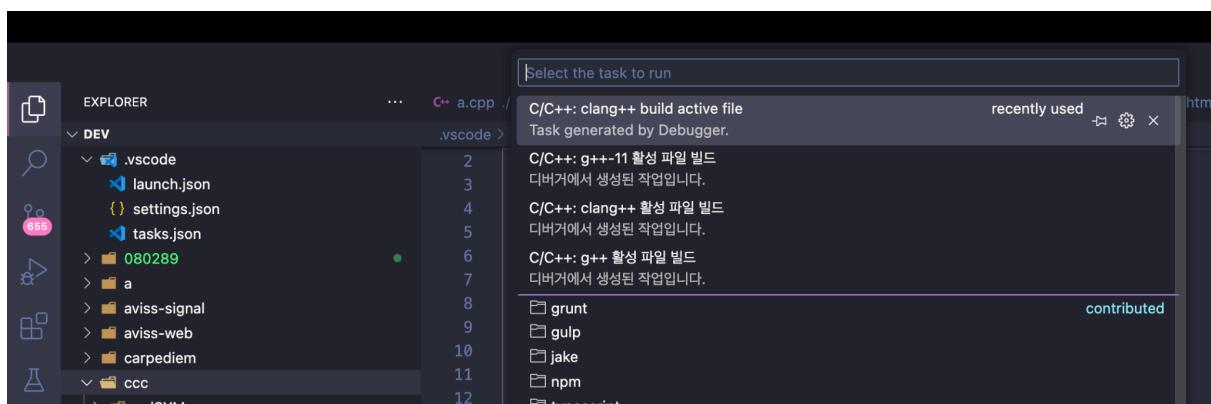
그 다음 a.cpp라는 파일 하나를 만들고 이 파일만을 놓는 폴더 하나를 만듭니다. 저 같은 경우 ccc라는 폴더 안에 a.cpp라는 파일만을 만들어 놓았습니다.



그 다음 Terminal >> run task를 누릅니다.



다음 그림처럼 clang을 선택해 빌드합니다.



그 이후에는 a 실행파일이 나오며 다음과 같은 명령어를 통해 실행할 수 있습니다.

```
./a
```

1.2 기본

예제로 이해하는 C++

입력 받은 문자열을 그대로 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)
}
```

예를 들어 이 프로그램은 실행시키고 wow라고 입력을 하면 wow가 출력되는 프로그램이죠.

```
• zagabi@zagabiui-MacBookPro dev % g++ -std=c++14 -Wall a.cpp -o test.out
• zagabi@zagabiui-MacBookPro dev % ./test.out
wow
wow
```

주석을 하나하나 설명하자면 다음과 같습니다.

1. 헤더파일을 include 시킨다. bits/stdc++.h는 C++의 모든 표준 라이브러리가 포함된 헤더파일입니다. 이를 include라는 지시문을 통해 이 프로그램에 포함시킨다라는 의미입니다.

2. std라는 네임스페이스(namespace)를 사용한다. 네임스페이스란 많은 라이브러리를 불러서 사용하다보면 변수명 중복이 발생할 수 있는데 이를 방지하기 위해서 변수명에 범위를 걸어놓는 것을 의미합니다. cin이나 cout 등을 사용할 때 원래는 std라는 네임스페이스를 통해 std::cin 이렇게 호출을 해야 하는데 std를 기본으로 설정해서 cin, cout으로 호출할 수 있게 합니다.

3. 문자열변수 a를 선언했다. <타입> <변수명> 이렇게 선언합니다. string이라는 타입을 가진 a라는 변수입니다. 이 때 예를 들어 string a = “큰돌”이라고 해봅시다. 이럴 때 a를

lvalue라고 하며 큰돌을 rvalue라고 합니다. lvalue는 왼쪽에 정의되면 추후 다시 사용될 수 있는 변수이며 rvalue는 오른쪽에 정의되면 한번쓰고 다시 사용되지 않을 변수를 말합니다.

4. 변수 a를 입력받다. 대표적으로 입력함수로는 cin, scanf가 있습니다. 후에 좀 더 자세하게 설명합니다.

5. 변수 a를 출력한다. 대표적으로 출력함수는 cout과 printf가 있습니다. 후에 좀 더 자세하게 설명합니다.

6. main함수를 종료시키는 return 0입니다. 프로세스를 정상적으로 마무리한다는 의미입니다.(process exit call success) 참고로 C++로 만든 프로그램은 실행시 main() 함수 하나를 찾고 그곳부터 실행을 시작합니다. 즉, cpp 파일당 반드시 하나의 main함수를 만들어야 합니다.

typedef

typedef를 통해 타입의 이름을 새로이 별칭으로 정의하고 실제 타입이름 대신 별칭으로 사용할 수 있습니다. 이를 통해 C++에서 이미 정의된 타입 또는 사용자가 정의한 타입(struct 또는 class)보다 더 짧거나 의미있는 이름을 지을 수 있습니다. 사용법은 다음과 같습니다.

```
typedef <타입> <별칭>
```

다음 코드처럼 int라는 타입을 i라는 새로운 별칭으로 바꿔서 표현하는 것을 볼 수 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
typedef int i;
int main(){
    i a = 1;
    cout << a << '\n';
    return 0;
}
/*
1
*/
```

define

define을 통해 상수, 매크로를 정의할 수 있습니다.

```
#define <이름> <값>
```

다음 코드처럼 PI라는 상수를 정의했으며 for 반복문을 loop라는 문자열로 치환한 것을 볼 수 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
#define PI 3.14159
#define loop(x,n) for(int x = 0; x < n; x++)

int main(){
    cout << PI << '\n';
    int sum = 0;
    loop(i, 10){
        sum += i;
    }
    cout << sum << '\n';
    return 0;
}
/*
3.14159
45
*/
```

STL

C++은 STL(Standard Template Library)을 제공하며 이는 자료구조, 함수 등을 제공하는 라이브러리를 뜻합니다. 알고리즘, 컨테이너, 이터레이터, 평터 이렇게 4가지를 제공합니다. 우리가 C++로 vector라는 자료구조를 쓴다면 sort()함수를 쓸 수 있는 것은 다 STL 덕분입니다. C++의 장점 중의 하나죠.

알고리즘

정렬, 탐색 등에 관한 함수로 이루어져 있습니다. sort()가 대표적입니다.

컨테이너

컨테이너는 여러가지 의미로 쓰입니다. 클라우드 서비스의 컨테이너도 있고 물건을 많이 담을 수 있는 컨테이너 박스라는 의미도 있죠. 여기서의 컨테이너는 C++에서 제공하는 자료구조를 의미합니다.

시퀀스 컨테이너, 연관 컨테이너, 정렬되지 않은 연관 컨테이너, 컨테이너 어댑터가 있습니다.

- 시퀀스 컨테이너(sequence container)는 데이터를 단순히 저장해 놓는 자료구조를 뜻하며 array, vector, deque, forward_list, list 가 있습니다.
- 연관 컨테이너(associative container)는 자료가 저장됨에 따라 자동정렬되는 자료구조를 말합니다. 중복키가 가능한 것은 이름에 multi가 붙습니다. set, map, multiset, multimap이 있습니다.
- 정렬되지 않은 연관 컨테이너(unordered associative container)는 자료가 저장됨에 따라 자동정렬이 되지 않는 자료구조를 말합니다. unordered_set, unordered_map, unordered_multiset, unordered_multimap이 있습니다.
- 컨테이너 어댑터(container adapter) 는 시퀀스 컨테이너를 이용해 만든 자료구조를 뜻하며 stack, queue는 deque로 만들어져 있으며 priority_queue는 vector을 이용해 힙 자료구조로 만듭니다.

자료구조는 추후 자세히 배웁니다. 일단 이렇게 있구나 라고 생각만 하고 들어갑시다.

이터레이터

추후 자세히 배웁니다.

펑터

펑터란 함수 호출 연산자를 오버로드하는 클래스의 인스턴스를 말합니다. 여기서는 배우지 않습니다.

1.3 입력과 출력

프로그램의 기초가 되는 입력과 출력에 대해 배워보겠습니다. 어떤 코딩테스트 환경은 입출력을 신경쓰지 않아도 되는 경우도 있지만 입출력을 신경써야 되는 경우도 발생하니 배우는 것이 좋습니다.

입력

대표적으로 cin 과 scanf가 있습니다. cin은 개행문자까지 입력을 받고 scanf는 형식을 지정해서 입력을 받습니다.

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

cin

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

다음 코드처럼 입력에는 “양영주 바보”를 입력했는데 출력에는 양영주만 나타나는 것을 볼 수 있습니다. 개행문자까지만 입력을 받는 것이죠.

```
#include <bits/stdc++.h>
using namespace std;
string a;
int main(){
    cin >> a;
    cout << a << "\n";
    return 0;
}
/*
입력
양영주 바보

출력
양영주
```

```
 */
```

그렇다면 개행문자를 넣어서 2개의 문자열을 입력하면 어떻게 될까요?

다음코드처럼 모든 문자열이 올바르게 입력받아지는 것을 볼 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
string a, b;
int main(){
    cin >> a >> b;
    cout << a << "\n";
    cout << b << "\n";
    return 0;
}
/*
입력
양영주 바보

출력
양영주
바보
*/
```

scanf

scanf는 첫번째 매개변수로 받는 형식을 지정해서 입력을 받습니다.

```
int scanf ( const char * format, ... );
```

%d는 int 타입.

%lf는 double 타입.

%c는 char 타입을 받습니다.

```
#include <bits/stdc++.h>
using namespace std;
int a;
double b;
char c;
int main(){
    scanf("%d %lf %c", &a, &b, &c);
    printf("%d\n", a);
    printf("%lf\n", b);
    printf("%c\n", c);
```

```

    return 0;
}
/*
입력
23330
233.23123
a

출력
23330
233.231230
a
*/

```

scanf는 주로 입력형식이 까다롭거나 이를 이용해야할 때 사용하는게 좋으며 보통은 cin을 쓰는게 좋습니다.

scanf로 받을 수 있는 타입과 형식은 다음과 같습니다.

형식	타입
d	int
c	char
s	string
lf	double
ld	long long

scanf를 활용해 실수타입을 정수타입을 받아보기

3.22처럼 double 타입으로 들어오는 것을 double 타입으로 받을 수 있지만 scanf를 이용해서 int타입 2개를 만들어 받을 수 있습니다.

```

#include <bits/stdc++.h>
using namespace std;
int a, b;
double c;
int main(){
    scanf("%d.%d", &a, &b);
    printf("\n%d %d\n", a, b);

    scanf("%lf", &c);
    printf("%lf\n", c);
    return 0;
}
/*

```

```
입력  
3.22  
3.22  
  
출력  
3.22  
3.220000  
*/
```

사실 실수 타입의 연산은 굉장히 까다롭기 때문에 이런 트릭을 써서 정수타입 기반의 연산을 유도하기도 합니다.

[참고] 실수형 연산의 제한된 정확도에서 더 자세히 배웁니다.

getline

제가 아까 cin이 개행문자까지 받는다고 했는데 한꺼번에 받으려면 어떻게 해야 할까요?

예를 들어 “엄준식 화이팅” 이런 문자열은 어떻게 한번에 받을 수 있을까요?

이럴 땐 getline으로 받으면 됩니다.

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

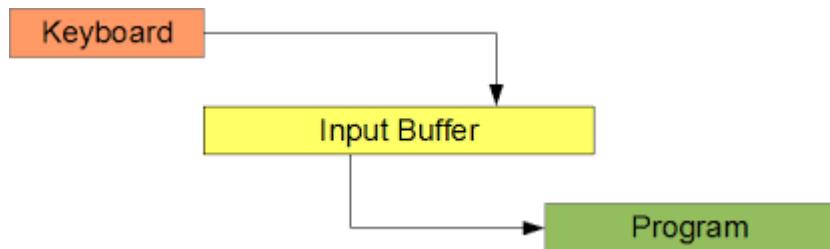
```

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 }

```

C:\Users\jhc\Desktop\dev\b.exe
엄준식 화이팅
엄준식 화이팅

하지만 `cin`으로 T 개의 `getline`을 받을지를 설정하고 T 개 만큼 `getline`의 입력이 들어오는 상황이 있습니다. 그럴 때는 특정 문자열을 기반으로 버퍼플래시를 하고 받아야 합니다. 이 이유는 `cin`으로 입력을 받을 때 개행문자 직전 까지 입력을 받게 되고 이 때문에 중간에 위치한 버퍼에 `\n`이 남아있게 되는 것이죠. 이를 없애주기 위해 `getline(cin, bufferflush)`를 해주어야 합니다.



참고로 개행문자는 한 칸 띄어쓰기, 한줄 띄어쓰기를 일컫습니다.

```

#include <bits/stdc++.h>
using namespace std;
int T;
string s;
int main(){
    cin >> T;
    string bufferflush;
    getline(cin, bufferflush);
    for(int i = 0; i < T; i++){
        getline(cin, s);
        cout << s << "\n";
    }
    return 0;
}

```

출력

출력입니다. cout과 printf가 있습니다.

cout

cout << 출력할 것 << "\n"해서 하는게 일반적입니다. 저렇게 “한줄” 띄어쓰기를 넣어도 되고 “한칸” 띄어쓰기를 원한다면 cout << "출력할 것" << " "; 이렇게 넣어도 됩니다. << 이어서 입력할 문자열이나 문자를 넣어도 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
string a = "도랄팍 화이팅!";
string b = "윤하 노래 너무 좋다...";
int main(){
    cout << a << '\n';
    cout << a << " " << " " << b << '\n';
    return 0;
}
/*
도랄팍 화이팅!
도랄팍 화이팅! 윤하 노래 너무 좋다...
*/
```

cout의 실수 타입 출력

cout은 기본적으로 실수를 출력하면 일부분만 출력이 됩니다. 사용해 실수 타입을 출력하고자 한다면 어떻게 해야 할까요? 예를 들어 문제에서 소수자리 6자리까지 반올림해서 출력해야 한다고 해봅시다.

다음 코드처럼 cout.precision(자릿수 + 1); 를 걸어 정해주면 됩니다.

```
#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;
}
```

printf

printf는 형식을 지정해서 출력할 때 좋습니다. 다음 코드처럼 형식(format)을 정한 후 다음 매개변수로 변수를 넣으면 형식에 맞춰 출력됩니다.

```
int printf ( const char * format, ... );
```

예를 들어 문제에서 홍철 1 : 지수 2 이런 형식으로 출력하라고 한다면 어떻게 해야 할까요?
다음 코드처럼 하면 됩니다.

[참고] 형식의 타입은 scanf에서 설명한 표를 참고해주세요.

```
#include<bits/stdc++.h>
using namespace std;
int a = 1, b = 2;
int main(){
    printf("홍철 %d : 지수 %d\n", a, b);
    return 0;
}
/*
홍철 1 : 지수 2
*/
```

또 하나 예를 들죠. printf의 출력형식을 이용해 소수점 6자리까지 그리고 2를 02, 또는 12는 12 이런식으로 출력한다면 어떻게 해야 할까요?

앞서 입력에서 설명했듯이 형식을 지정할 수 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
double a = 1.23456789;
int b = 12;
int c = 2;
int main(){
    printf("%.6lf\n", a);
    printf("%02d\n", b);
    printf("%02d\n", c);
    return 0;
}
/*
1.234568
12
02
```

```
 */
```

다음 코드를 보죠. scanf 처럼 형식을 지정해서 출력하는 모습입니다.

```
#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
*/
```

앞의 코드를 보면 string str을 출력할 때는 c_str() 함수를 걸어서 출력하고 있는데 문자열을 printf로 출력할 때 주의할 점입니다.

printf 를 기반으로 문자열을 출력하려면 string을 문자열에 대한 포인터(char *) 타입으로 바꿔주어야 하기 때문입니다.(그래서 보통 문자열을 출력할 때는 간단히 cout을 사용하는 것이 좋습니다.)

1.4 타입과 타입 변환

C++ 엄격한 타입시스템 언어이기 때문에 매번 타입(type)을 설정해줘야 합니다.

또한 매개변수의 수나 타입 등에 따라 함수를 다르게 인식합니다. 예를 들어 func(int a, int b)와 func(int a)는 매개변수의 수만이 다르지만 엄하게 다른 함수로 인식됩니다.

타입

다음 타입들은 알고리즘에서 자주 나오는 타입 8개입니다.

사실 이것보다 long double 등 “조금은” 더 있지만 이 정도만 배워도 충분합니다.

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

차근차근 배워보겠습니다.

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 않는 함수에는 void로 선언합니다.

함수를 선언할 때 어떤 타입을 반환하는지 명시해주어야 하고 이를 return하는 값과 맞춰주어야 하는데 예를 들어 아무것도 반환하지 않는 void가 아닌 double 타입을 반환하는 함수는 어떻게 정의할 수 있을까요?

다음 코드 처럼 a()라는 함수는 double 타입으로 정의된 것을 볼 수 있습니다. 1.2333이라는 double 타입의 변수를 return 하는 것을 볼 수 있습니다.

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

```
    cout << ret << "\n";
    return 0;
}
```

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

다음 코드처럼 타입과 인자만 선언을 해 놓고 아래쪽에 함수를 정의하는 식으로 선언부와 정의부를 나눠서 함수를 설정하는 방법 또한 있습니다.

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

double a(){
    return 1.2333;
}
```

그러나 알고리즘은 시간싸움입니다. 이렇게 2번 함수를 적게되면 시간낭비입니다. 그냥 위에다 선언과 정의를 한꺼번에 합시다.

2. char, 문자

작은 따옴표 “ 이렇게 선언해야 하며 1바이트의 크기를 가집니다.

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

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

그렇다면 char을 리턴하는 함수는 어떻게 정의할 수 있을까요?

다음 코드처럼 정의할 수 있겠죠?

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

3. string, 문자열

앞서 배운 char을 아래의 코드처럼 char[] 배열로 선언하거나 그냥 string으로 선언해 여러개의 문자모음이자 문자배열인 문자열을 선언할 수 있습니다.

```
char s[10];
string a;
```

이 강의에서는 문자열은 char[] 과 string 중 string으로 진행하며 문자열은 string으로 선언하는 것을 추천합니다.

다음 코드처럼 string으로 선언했고 마치 배열처럼 a[0]으로 접근하거나 통째로 a로 출력하는 것을 볼 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string a = "나는야";
    cout << a[0] << "\n";
    cout << a[0] << a[1] << a[2] << '\n';
    cout << a << "\n";
    string b = "abc";
    cout << b[0] << "\n";
    cout << b << "\n";
    return 0;
}
/*?
?
```

```
나  
나는야  
a  
abc  
*/
```

그러나 앞의 코드를 보면 한글로 선언한 a의 경우 a[0]을 출력했을 때 이상한 문자가 나타난 것을 볼 수 있습니다.

문자열을 선언하고 a[0], a[1] 이렇게 접근한다는 의미는 0번째, 1번째 1바이트씩 출력한다는 것을 의미합니다. 영어는 한 글자당 1바이트지만 한글은 한 글자당 3바이트입니다.

그렇기 때문에 이러한 현상이 발생합니다. 그러나 abc와 같은 영어는 제대로 잘 나오는 것을 볼 수 있습니다.

다음코드는 string에서 많이 사용하는 메서드를 사용한 코드입니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string a = "love is";
    a += " pain!";
    a.pop_back();
    cout << a << " : " << a.size() << "\n";
    cout << char(* a.begin()) << '\n';
    cout << char(* (a.end() - 1)) << '\n';
    // string& insert (size_t pos, const string& str);
    a.insert(0, "test ");
    cout << a << " : " << a.size() << "\n";
    // string& erase (size_t pos = 0, size_t len = npos);
    a.erase(0, 5);
    cout << a << " : " << a.size() << "\n";
    // size_t find (const string& str, size_t pos = 0);
    auto it = a.find("love");
    if (it != string::npos){
        cout << "포함되어 있다." << '\n';
    }
    cout << it << '\n';
    cout << string::npos << '\n';
    // string substr (size_t pos = 0, size_t len = npos) const;
    cout << a.substr(5, 2) << '\n';

    return 0;
}
/*
love is pain : 12
```

```
l  
n  
test love is pain : 17  
Love is pain : 12  
포함되어 있다.  
0  
18446744073709551615  
is  
*/
```

+ =

메서드는 아니며 문자열에서 문자열을 더할 때 보통 $+=$ 를 써서 문자열 또는 문자를 더합니다. `push_back()`라는 메서드가 있지만 이는 문자하나씩밖에 더하지 못해 보통은 $+=$ 를 씁니다.

begin()

문자열의 첫번째 요소를 가리키는 이터레이터를 반환합니다. 이 이터레이터를 기반으로 `*`를 통해 해당 위치의 값을 가져올 수 있습니다.

[참고]이터레이터와 `*` 는 추후 배웁니다.

end()

문자열의 마지막 요소 그 다음을 가리키는 이터레이터를 반환합니다.

참고로 `begin()`과 `end()`는 자료구조인 `vector`, `Array`, `연결리스트`, `맵`, `셋`에서도 존재하며 똑같은 의미를 지닙니다.

size()

문자열의 사이즈를 반환합니다. $O(1)$ 의 시간복잡도를 가집니다.

[참고] 시간복잡도는 1주차 개념강의에서 배웁니다.

insert(위치, 문자열)

특정위치에 문자열을 삽입합니다. $O(n)$ 의 시간복잡도를 가집니다.

erase(위치, 크기)

특정위치에 크기만큼 문자열을 지웁니다. $O(n)$ 의 시간복잡도를 가집니다.

pop_back()

문자열 끝을 지웁니다. O(1)의 시간복잡도를 가집니다.

find(문자열)

특정 문자열을 찾아 위치를 반환합니다. 만약 해당 문자열을 못 찾을 경우 string::npos를 반환하며 O(n)의 시간복잡도를 가집니다.

string::npos는 size_t 타입의 최대값을 의미합니다. size_t 타입의 최대값은 OS에 따라 달라지며 64비트 운영체제라면 64비트 부호가 없는 최대정수, 32비트 운영체제라면 32비트 부호가 없는 최대 정수값을 가집니다. 필자의 컴퓨터는 64비트 운영체제이기 때문에 18446744073709551615라는 값을 가집니다.

substr(위치, 크기)

특정 위치에서 크기만큼의 문자열을 추출합니다. O(n)의 시간복잡도를 가집니다.

아스키코드와 문자열

만약 숫자로 된 문자에서 ++증감연산자를 통해 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;
}
```

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

문자열을 이루는 문자는 아스키코드 값(0에서 127 사이의 정수)으로 저장되어 구현됩니다.

예를 들어 'A'의 아스키코드 값은 65입니다. 이것이 의미하는 바는 문자 변수에 'A'를 할당하면 'A' 자체가 아니라 65라는 숫자가 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 부호체계입니다.

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	0	103	g		

앞의 표는 아스키 코드 중 제어 문자와 확장 아스키 코드를 제외한 부호(영문 자판에 사용되는 부호)를 정리한 것이며 이렇게 숫자로 변환되어 표시되는 것을 알 수 있습니다. 문자와 숫자가 매핑되어있는 표라고 생각하면 됩니다.

앞의 코드를 모두 외울 필요는 없습니다. 제가 빨강으로 설정한 2가지 97 : a / 65 : A 정도만 외워줍시다. 이걸 외운다면 d는 a부터 시작해 3번째이니 $97 + 3 = 100$ 의 아스키코드값을 가지는 구나라고 알 수 있기 때문이죠.

앞의 표에서 봤듯이 아스키코드는 0 ~ 127까지의 숫자를 지원합니다. 그렇기 때문에 아스키코드로 어떤 로직을 구축한다 했을 때 127이 넘어가는 숫자를 만들면 에러가 발생하니 조심해야 합니다.

앞서서 문자 A는 65로 저장된다고 했죠? 그렇다면 문자 a는 어떻게 저장이 될까요?
a는 아스키코드에서 봤듯이 97로 저장이 됩니다. 이를 확인하고 싶다면 타입 변환을 하면 됩니다.

(int)'a'를 통해 문자 char을 정수 int로 변환할 수 있는데 이를 하게 되면 다음코드처럼 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
```

reverse()

string은 reverse()라는 메서드를 지원하지 않습니다. 문자열을 거꾸로 뒤집고 싶다면 STL에서 지원하는 함수인 reverse()를 쓰면 됩니다.

```
void reverse (BidirectionalIterator first, BidirectionalIterator last);
```

reverse() 함수는 void 타입으로 아무것도 반환하지 않습니다. 그리고 원본 문자열도 바꿔버립니다.

다음 코드처럼 구축이 가능하며, a.begin() + 3 처럼 시작위치를 바꿔 뒤집고 싶은 부분만을 바꿀 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    string a = "It's hard to have a sore leg";
    reverse(a.begin(), a.end());
    cout << a << '\n';
    reverse(a.begin(), a.end());
    cout << a << '\n';
    reverse(a.begin() + 3, a.end());
    cout << a << '\n';

    return 0;
}
/*
gel eros a evah ot drah s'tI
It's hard to have a sore leg
It'gel eros a evah ot drah s
*/
```

split()

0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

코딩테스트에서는 문자열을 split() 하는 로직이 많이 등장합니다. split() 함수란 다른 프로그래밍 언어에서도 문자열을 특정 문자열을 기준으로 쪼개어서 배열화시키는 함수라는 의미로 사용되는데 C++에서는 STL에서 split() 함수를 지원하지 않습니다. 그래서 만들어야 합니다.

보통 다음과 같이 구현합니다. 시간복잡도는 O(n)의 시간복잡도를 가집니다.

```
#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;
}

int main(){
    string s = "안녕하세요 큰돌이는 킹갓제너럴 천재입니다 정말이에요!", d = " ";
    vector<string> a = split(s, d);
    for(string b : a) cout << b << "\n";
}
/*
안녕하세요
큰돌이는
킹갓제너럴
천재입니다
정말이에요!
*/

```

복잡해보이지만 다음코드처럼 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());
}

```

코드를 분석하면 다음과 같습니다.

input에서 delimiter를 찾습니다. 못 찾을 때까지는 이 루프는 반복됩니다.

```
while ((pos = input.find(delimiter)) != string::npos)
```

찾았다면 해당 pos까지 해당 부분 문자열을 추출합니다. 예를 들어 abcd에서 d를 찾았다면 pos는 3을 반환하게 되고 3만큼 substr을 하기 때문에 abc를 추출하게 됩니다.

```
    token = input.substr(0, pos);
```

그 다음 이 추출한 문자열을 ret이란 배열에 집어 넣습니다.

```
ret.push_back(token);
```

그리고 앞에서 부터 문자열을 지웁니다. abcdabc에서 d가 delimiter이라면 pos = 3, delimiter의 사이즈는 1이기 때문에 앞에서 부터 4의 크기의 문자열을 제거해 abc만 남게 됩니다.

```
input.erase(0, pos + delimiter.length());
```

범위기반 for 루프

앞의 코드를 보면 범위기반 for루프가 있는 것을 볼 수 있는데 C++11부터 범위기반 for 루프가 추가되어서 이를 쓸 수 있습니다.

```
for ( range_declaration : range_expression )
    loop_statement
```

다음 코드처럼 vector 내의 있는 요소들을 쉽게 순회할 수 있습니다. 1번코드와 2번코드는 같은 의미입니다. for({타입} {임시변수명} : {타입을 담은 컨테이너}) 이렇게 쓰입니다.

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    vector<int> a = {1, 2, 3};
    for(int b : a) cout << b << "\n"; // 1
    for(int i = 0; i < a.size(); i++) cout << a[i] << "\n"; // 2
    return 0;
}
/*
1
2
3
1
2
3
*/
```

[참고] vector는 추후 배웁니다.

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 쓰입니다. 이렇게 보면 만약 입력받은 문자열이 문자라면 0을 반환하고 그게 아니라면 숫자를 반환합니다.

```
#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
*/
```

4. bool, 참과 거짓

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

참고로 C++에서는 0이면 false, 0이 아닌 값들은 모두 true가 되며 bool()을 통해 간단하게 bool형으로 형변환이 가능합니다.

```
#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
*/
```

5. int, 4바이트짜리 정수

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

- $2,147,483,647$ 은 $2^{31} - 1$ 입니다.

```
int a = 10;
```

또한 문제를 풀 때는 이상한 문제가 아니라면 int의 최대값으로 20억까지가 아닌 987654321 또는 $1e9$ 를 씁니다.

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

앞의 코드처럼 const int로 선언을 해서 쓰는 것이죠.

왜냐하면 이 INF를 기반으로 $INF + INF$ 라는 연산이 일어날 수도 있고 $INF * 2$, 그리고 $INF +$ 다른 수 연산이 일어날 때 int의 최대값을 넘어가는 오버플로를 방지할 수 있는 장점이 있기 때문입니다.

int 연산

int로 선언한 변수끼리 연산을 하게 되었을 때 실수가 나온다면 소수점 아래에 있는 수는 모두 버림이 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL); cout.tie(NULL);
    int a = 3;
    int b = 2;
    cout << a / b << '\n'; // 1
    double c = 3;
    double d = 2;
    cout << c / d << '\n'; // 1.5
}
```

이 현상은 int 타입끼리만 그렇지, 앞의 코드처럼 double 타입은 버림없이 1.5가 나오는 것을 알 수 있습니다.

const 키워드

const 키워드는 수정할 수 없는 변수를 정할 때 쓰입니다. 보통 INF 같은 것이나 방향벡터를 나타내는 dy, dx에 const를 씁니다.

예를 들어 문제에서 주어진 맵의 크기가 10 * 10 이기 때문에 10 * 10 2차원 배열을 만들어야 할 경우가 있습니다.

이 경우 이렇게 하는게 좋습니다.

```
const int mx = 10;
int a[mx][mx];
```

어차피 10이라는 숫자는 변하지 않는 상수이기 때문에 “미리” 설정해놓고 이를 기반으로 맵의 크기를 설정하는 것이죠. 이렇게 하면 갑자기 a[10][10]인데 a[10][1]으로 설정한다던가 등의 실수를 방지할 수 있습니다.

오버플로

오버플로(overflow)란 타입의 허용범위를 넘어갈 때 발생하는 에러를 뜻합니다. 예를 들어 int 타입이라면 2,147,483,647을 넘어간다면 에러가 발생하게 됩니다.

다음 코드 처럼 최대범위를 벗어나게 되면 최댓값 + 1이 아닌 최솟값으로 돌아가게 되버립니다. 이를 UB(unexpected Behavior)라고도 합니다. 예측할 수 없는 값이 나타나는 것이죠.

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

int main(){
    int a = 2147483647;
    cout << a << '\n';
    a++;
    cout << a << '\n';
}
/*
2147483647
-2147483648
*/
```

언더플로

오버플로와는 반대로 취급할 수 있는 결과값보다 작아지게 되면 언더플로가 발생됩니다.

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

```
using namespace std;

int main(){
    int a = -2147483648;
    cout << a << '\n';
    a--;
    cout << a << '\n';
}
/*
-2147483648
2147483647
*/
```

6. long long, 8바이트짜리 정수

8바이트짜리 정수입니다.

범위는 $-9,223,372,036,854,775,808 \sim 9,223,372,036,854,775,807$ 입니다. 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 $1e18$ 로 정의를 해놓고 쓰면 되나 이는 문제마다 다르니 참고만 해주시면 됩니다.

- $9,223,372,036,854,775,807$ 은 $2^{63} - 1$ 입니다.

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

7. double, 실수 타입

실수타입입니다. 8바이트이며 소수점 아래로 15자리 까지 표현이 가능합니다. 참고로 실수타입을 표현하는 타입은 float도 있는데 이는 4바이트, 소수점 아래로 7자리까지 표현이 가능합니다. float과 double 중 double이 더 정확하게 표현이 가능하니 double을 쓰는게 더 좋습니다.

```
double pi = 3.14159;
```

8. unsigned long long, 8바이트짜리 양의 정수

부호가 없는 정수입니다. long long에서 -로 표현할 범위를 몽땅 + 범위에 추가한 타입입니다. 아주 가끔 쓰며 음수를 표현할 수 없습니다.

- 범위 : 0 ~ 18,446,744,073,709,551,615

pair와 tuple

pair와 tuple은 타입이나 자료구조는 아닙니다. C++에서 제공하는 utility 라이브러리 헤더의 템플릿 클래스이며 자주 사용되기 때문에 알아보겠습니다.

pair는 first와 second라는 멤버변수를 가지는 클래스이며 두가지 값을 담아야 할 때 씁니다. tuple은 세가지 이상의 값을 담을 때 쓰는 클래스입니다.



먼저 tie를 써서 pair이나 tuple로부터 값을 끄집어내는 코드입니다.

```
#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 원래는 a = pi.first; b = pi.second 이런식으로 끄집어내야 하는데 tie를 쓰게 되면 tie(a, b) = pi 이렇게 끄집어 낼 수 있습니다.

first와 second라는 코드를 쓰지 않으니 너무나도 편하게 요소를 끄집어낼 수 있습니다. 물론 이 때 a와 b는 앞서서 변수로 선언되어야 합니다.

pair<int, int> a = {1, 2};
 tie(c, d); a.first a.second
 tie(c, d) = a
 ↳ first와 second를
 쓰지 않고 끄집어내기

다음 코드는 tie를 쓰지 않는 코드입니다. pair은 first, second로 값을 끄집어 내며, tuple은 경우 get<0>, get<int> .. 이런식으로 값을 꺼내야 합니다.

[참고] 필자는 일일히 get<0>.. 로 꺼내는 것은 별로라고 생각해서 보통 3가지 이상의 멤버변수가 필요하면 tuple보다는 struct를 쓰는 편입니다.

```

#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
*/

```

auto 타입

auto는 타입추론을 하여 결정되는 타입입니다. 다음 코드처럼 b라는 변수를 auto로 선언했는데 rvalue가 1인 것을 통해 자동적으로 int 타입의 변수를 선언했듯이 쓸 수 있는 것을 볼 수 있습니다.

```

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

int a = 1;
auto b = 1;
int main(){
    cout << b << '\n';
}
/*
1
*/

```

auto 타입은 주로 복잡하고 긴 타입의 변수명을 대신할 때 쓰입니다.

예를 들어 다음 코드처럼 pair<int, int> it가 아닌 auto it로 조금 더 짧게 선언할 수 있습니다.

```

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

```

```

for(int i = 1; i <= 5; i++){
    v.push_back({i, i});
}
for(auto it : v){
    cout << it.first << " : " << it.second << '\n';
}
for(pair<int, int> it : v){
    cout << it.first << " : " << it.second << '\n';
}
return 0;
}
/*
1 : 1
2 : 2
3 : 3
4 : 4
5 : 5
1 : 1
2 : 2
3 : 3
4 : 4
5 : 5
*/

```

타입변환

만약 int타입인 것을 double타입으로 타입을 변환해야한다면 어떻게 해야할까요?

(바꿀타입) 기준변수 이런식으로 하면 됩니다.

double을 int형으로 만들기

다음 코드 처럼 double형인 변수의 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;
}

```

여기서 중요한 점이 있는데 같은 타입끼리 연산을 하는 것이 중요합니다. 지금 보시는 것처럼 ret이란 double 타입에 n을 double로 만들어서 연산하는 것을 볼 수 있습니다.

double은 double끼리 나눠야 합니다. int를 double로 나누면 double 타입의 결과값이 나오는데 이를 코딩테스트 때 신경쓰기란 어려운 일입니다. 차라리 double은 double끼리 연산하고 int는 int끼리 연산하게 타입변환을 해놓고 연산하는게 "맞왜틀"에 빠지지 않을 가능성을 높여줍니다.

[참고] 산술표현식을 평가할 때 같은 타입을 가져야 하나 이게 맞지 않을 경우 암시적 형변환(Implicit type conversion)이 일어납니다. 이 때 다음과 같은 우선순위를 거쳐 형변환이 일어납니다. 예를 들어 double과 float끼리 연산이 일어난다면 double로 통일되어 값을 반환합니다.

우선순위는 다음과 같습니다.

- long double (highest)
- double
- float
- unsigned long long
- long long
- unsigned long
- long
- unsigned int
- int (lowest)

이 때문에 vector의 size()를 기반으로 음수가 나올 수 있는 연산을 할 때 주의 해야 합니다. vector의 size()라는 메서드는 unsigned int를 반환합니다. 따라서 v.size() - 10 이렇게 연산을 할 때 결과값은 음수가 나와야 하지만 unsigned int와 int와의 연산이기 때문에 unsigned int가 나오며 따라서 아주 큰 양수가 나오게 될 수 있습니다.

bad

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    vector<int> a = {1, 2, 3};
    cout << a.size() - 10 << '\n'; // 18446744073709551609
    return 0;
}
```

good

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

```
int main(){
    vector<int> a = {1, 2, 3};
    cout << (int)a.size() - 10 << '\n'; // -7
    return 0;
}
```

타입변환시 주의할점

다음 2개의 코드는 같을까요?

```
int a = (int) p * 100 // 1
int a = (int) 100 * p // 2
```

1번 코드만 타입변환이 됩니다. 순서에 주의해주세요!

문자를 숫자로, 숫자를 문자로

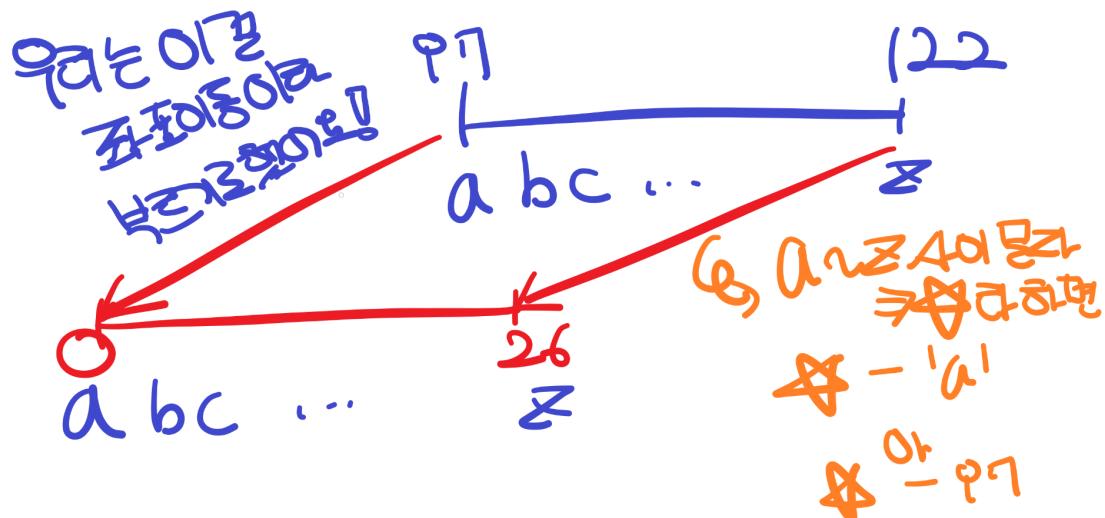
예를 들어 소문자로 된 문자를 숫자로 바꾸는 로직이 필요하다 생각해봅시다. 어떻게 해야 할까요? 정답은 아스키코드를 이용하는 것입니다. 앞서 설명했듯이 A ~ Z 는 65 ~ 90 / a ~ z 는 97 ~ 122의 아스키 코드를 가지고 있습니다.

a부터 시작해 z부터 입력을 받는데 이를 정수 0 ~ 25까지 표현하고 싶다면 다음과 같이 작성합니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    char a = 'a';
    cout << (int)a << '\n';
    cout << (int)a - 97 << "\n";
    cout << (int)a - 'a' << "\n";
    return 0;
}/*
97
0
0
*/
```

(int)a는 97이라는 값을 가집니다. 그리고 이를 97을 빼면 0이죠. 또한 'a'를 빼게 되면 자동적으로 아스키코드에 매핑되어있는 97을 빼게 되어서 0이 되게 됩니다. 이는 일종의

좌표이동입니다. a ~ z를 표현하려면 123의 공간이 필요한데 이를 통해 26의 공간만으로 표현할 수 있습니다.



1.5 메모리와 포인터

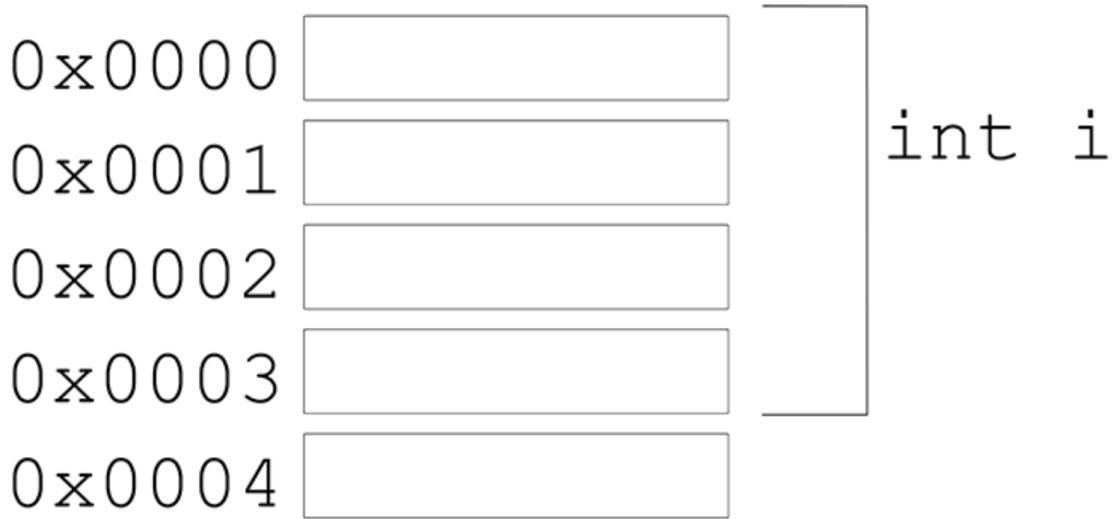
0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

메모리

자, 다음 코드를 작성하면 어떤 일이 벌어질까요?

```
int i;
```

컴퓨터의 메모리는 메모리 셀의 연속과 같으며 각 셀의 크기는 1바이트이고 고유한 주소가 있습니다. 이 코드를 구동한 제 컴퓨터의 메모리는 128GB입니다. 이 메모리에서 4바이트 정수타입인 int 타입의 변수를 저장하기 위해 128GB 중 4바이트의 메모리 영역을 예약합니다.



이렇게 예약한 메모리 영역인 0x0000, 0x0001.. 등의 주소가 가리키는 영역에 int i라는 변수가 들어가게 됩니다.

여기서 이 변수의 메모리 주소란 변수가 사용하는 메모리 주소의 첫번째를 가리킵니다.

앞의 그림에서 변수 i의 메모리 주소는 첫번째 주소인 0x0000이 되겠죠?

자 그럼 i의 메모리 주소를 출력해볼까요?

메모리 주소는 16진수로 표기가 되고 C++에서는 &연산자(ampersand, 앤퍼샌드)를 통해 변수의 메모리 주소를 얻을 수 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
int i;
int main(){
    cout << &i << '\n';
    return 0;
}
// 0x100bbc000
```

참고로 원래는 저 그림대로라면 0x0000을 가리켜야 하는데 이는 OS, 실행시간 등에 따라 주소할당이 달라지기 때문에 저러한 주소가 나타나게 되는 것입니다.

자 그러면 이 변수에 0이라는 값을 할당하면 어떤 일이 일어날까요?

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

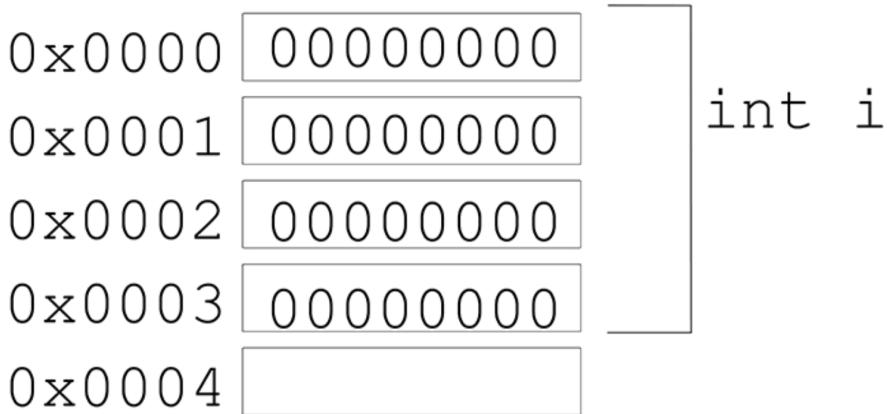
```

int i;
int main(){
    cout << &i << '\n';
    i = 0;
    cout << &i << '\n';

    return 0;
}
/*
0x100f28000
0x100f28000
*/

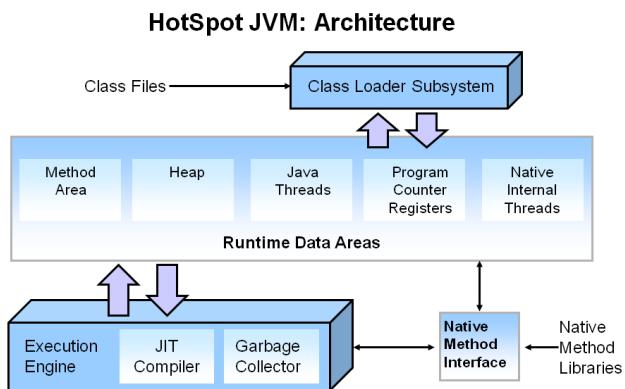
```

앞의 코드처럼 `i`에 0을 할당하면 다음 그림처럼 방금 예약한 메모리 영역에 해당 값을 저장하게 됩니다. 0이든 2든 3이든 다른 값을 넣어도 주소는 변하지 않습니다.



포인터

메모리관리는 언어마다 조금은 다르게 관리가 됩니다. 자바, 파이썬, 자바스크립트라는 언어로는 개발자가 직접 변수에 메모리를 할당하거나 해제할 수 없고 가비지컬렉터를 통해 이를 수행합니다.



하지만 하위레벨 언어인 C, C++ 등은 가비지컬렉터가 없으면 대신 개발자가 직접 필요한 메모리를 예약하고 해제할 수 있으며 포인터 또한 지원합니다.

그래서 메모리와 포인터를 설명할 때 C++로 제가 지금 설명하고 있는 것이죠.

포인터의 개념

변수의 메모리 주소를 담는 타입이 바로 포인터입니다. 포인터는 메모리 동적 할당, 데이터를 복사하지 않고 함수 매개변수로 사용, 클래스 및 구조체를 연결할 때 사용됩니다.

ex) 연결리스트의 노드

```
class Node {
public:
    int data;
    Node* next;
};
```

다음 코드를 보면 `int * a`라는 `&i`라는 `i`의 주소를 담는 포인터를 정의한 것을 볼 수 있습니다. <타입> * 형태로 포인터를 정의합니다. 예를 들어 `string` 타입 변수의 메모리주소를 담을 때는 `string *` 하고 선언을 해야 합니다.

C++에서 *라는 별표는 에스터리스크(asterisk operator)라고도 불립니다.

```
#include<bits/stdc++.h>
using namespace std;
int i;
string s = "kundol";
int main(){
    i = 0;
    int * a = & i;
    cout << a << '\n';
    string * b = &s;
    cout << b << '\n';
```

```
    return 0;  
}
```

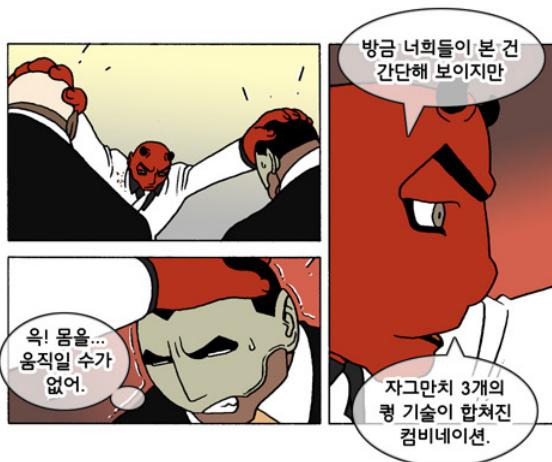
포인터의 크기

또한, 포인터의 크기는 OS가 32bit라면 4바이트, 64bit라면 8바이트로 고정되어있습니다. 어떠한 타입이든(string, char, int 등) 상관없이 무조건 4바이트 아니면 8바이트로 고정됩니다. 이는 집 주소의 크기는 집의 크기와 관련이 없다는 것을 생각하시면 됩니다.

포인터의 크기는 고정되어있지 않으며 OS마다 다를 수 있습니다. 보통 4바이트 또는 8바이트의 크기를 가집니다.(예를 들어 1바이트짜리 char 타입의 변수의 포인터 크기는 1바이트가 아니라는 것이죠.)

역참조 연산자

C++에서 *은 기호는 사용하는 위치에 의해 다양한 용도로 사용됩니다. 이항 연산자로 사용하면 곱셈 연산으로, 포인터 타입의 선언, 역참조(dereference)로 메모리를 기반으로 변수의 값에 접근할 때도 사용됩니다. (무려 3가지나 가능하다고??) 이 중 역참조를 알아보도록 하겠습니다.



다음 코드처럼 b라는 포인터를 정의했고 이 포인터에 * 연산자를 통해 역참조를 걸어 주소값을 기반으로 값을 끄집어낸 것을 볼 수 있습니다.

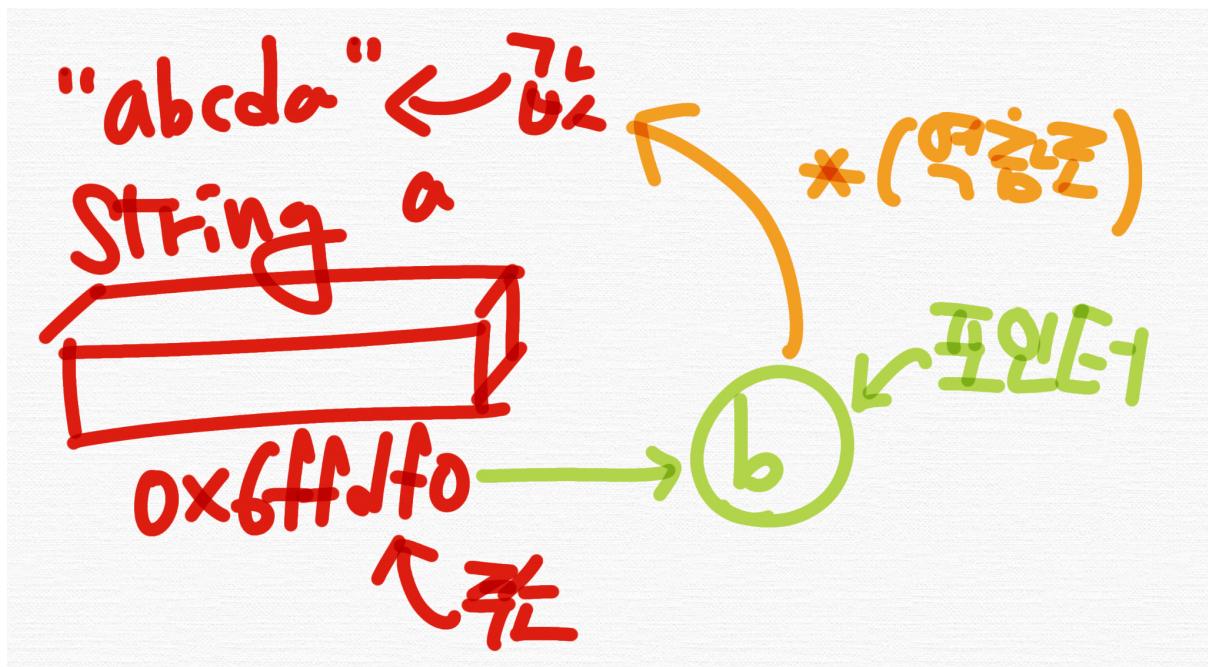
```
#include<bits/stdc++.h>  
using namespace std;
```

```

int main(){
    string a = "abcda";
    string * b = &a;
    cout << b << "\n";
    cout << *b << "\n";
    return 0;
/*
0x6ffdfo
abcda
*/
}

```

다음 그림처럼 주소값을 담은 포인터에 역참조연산자를 통해 값을 참조할 수 있는 것을 볼 수 있습니다.



array to pointer decay

우리가 배열의 이름을 주소값으로 쓸 수 있다고 들어는 봤죠?

이는 정확히 array to pointer decay를 의미합니다.

배열이 포인터로 부식(decay)되는 현상을 말합니다. 이는 배열의 이름을 T^* 라는 포인터에 할당하면서 $T[N]$ 이란 배열의 크기 정보 N 이 없어지고 첫번째 요소의 주소가 바인딩되는 현상을 의미합니다.

즉 배열의 크기 정보가 있었는데.. 없어지는 것이죠

있었는데.. 없어졌다?

이를 통해 배열의 이름은 배열의 첫번째 주소로써 쓸 수 있습니다.

참고로 vector는 안됩니다. array만 가능합니다.

다음 코드는 포인터 c에 할당한 배열의 이름이 &a[0]과 같은 의미로 쓰이는 것을 볼 수 있습니다.

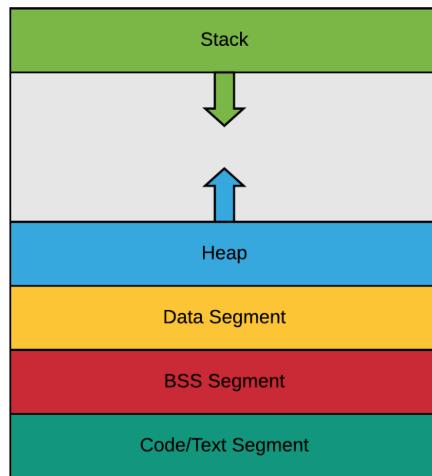
```
#include<bits/stdc++.h>
using namespace std;
int a[3] = {1, 2, 3};
int main(){
    int * c = a;
    cout << c << "\n";
    cout << &a[0] << "\n";
    cout << c + 1 << "\n";
    cout << &a[1] << "\n";
    return 0;
}
/*
0x472010
0x472010
0x472014
0x472014
*/
```

프로세스 메모리 구조와 정적할당과 동적할당

코드를 작성해서 컴파일하고 프로그램을 실행시킬 때 어떠한 구조로 메모리에 할당이 될까요?

운영체제는 프로세스에 적절한 메모리를 할당하는데 다음 구조를 기반으로 할당합니다.

위에서부터 스택(stack), 힙(heap), 데이터 영역(BSS segment, Data segment), 코드 영역(code segment)으로 나눠집니다. 스택은 위 주소부터 할당되고 힙은 아래 주소부터 할당 됩니다.

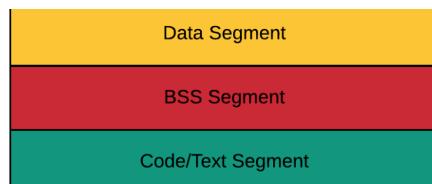


- 스택 : 지역변수, 매개변수, 함수가 저장되고 컴파일 시에 크기가 결정됩니다. 그러나 함수가 함수를 호출 하는 등에 따라 런타임시에도 크기가 변경됩니다. [동적인 특징]
- 힙 : 힙은 동적 할당할 때 사용되며 런타임 시 크기가 결정됩니다.[동적인 특징]
- 데이터영역 : BSS 영역과 Data 영역으로 나뉘고 정적할당에 관한 부분을 담당합니다. [정적인 특징]
- 코드영역 : 소스코드 들어감. [정적인 특징]

정적할당

정적할당은 컴파일단계에서 메모리를 할당하는 것을 말합니다.

BSS segment와 Data segment, code / text segment로 나뉘어서 저장됩니다.



BSS segment는 전역변수, static, const로 선언되어있는 변수 중 0으로 초기화 또는 초기화가 어떠한 값으로도 되어 있지 않은 변수들이 이 메모리 영역에 할당됩니다.

```
#include<bits/stdc++.h>
using namespace std;
int a;
int b = 0;
const int c = 0;
int main(){
    static int d;
    static int e = 0;
    return 0;
}
```

```
}
```

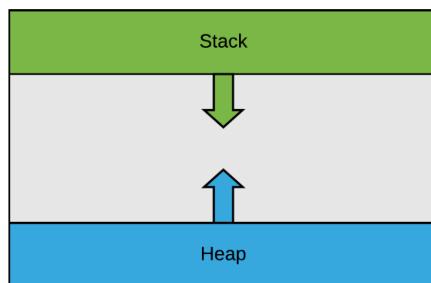
Data segment은 전역변수, static, const로 선언되어있는 변수 중 0이 아닌 값으로 초기화된 변수가 이 메모리 영역에 할당됩니다.

```
#include<bits/stdc++.h>
using namespace std;
int a = 1;
const int b = 2;
int main(){
    static int c = 3;
    return 0;
}
```

code / text segment는 프로그램의 코드가 들어갑니다.

동적할당

동적할당은 런타임단계에서 메모리를 할당받는 것이며 Stack과 Heap으로 나눠집니다.



Stack은 지역변수, 매개변수, 실행되는 함수에 의해 늘어나거나 줄어드는 메모리 영역입니다.

함수가 호출될 때마다 호출될 때의 환경 등 특정 정보가 stack에 계속해서 저장됩니다.

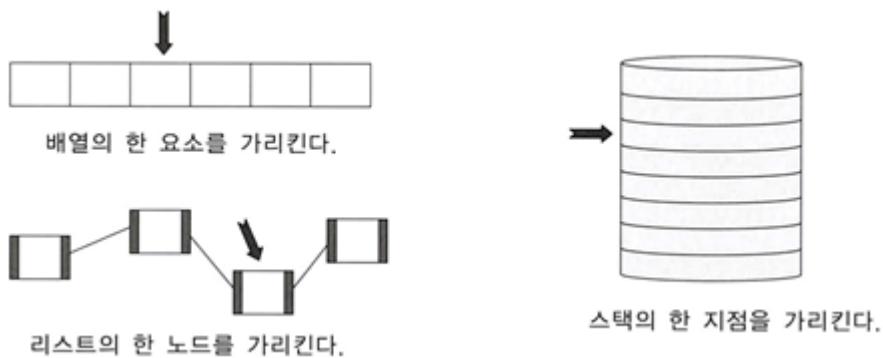
또한, 재귀함수가 호출된다고 했을 때 새로운 스택 프레임이 매번 사용되기 때문에 함수 내의 변수 집합이 해당 함수의 다른 인스턴스 변수를 방해하지 않습니다.

즉, 재귀함수 내의 지역변수로 선언하게 되면 해당 변수는 독립적으로 작용하며 다른 함수에 있는 변수에 영향을 끼치지 않습니다.

Heap은 동적으로 할당되는 변수들을 담습니다. malloc(), free() 함수를 통해 관리할 수 있으며 동적으로 관리되는 자료구조의 경우 Heap영역을 사용합니다. 예를 들어 vector는 내부적으로 heap영역을 사용합니다.

1.6 이터레이터

이터레이터는 컨테이너에 저장되어 있는 요소의 주소를 가리키는 개체를 말하며 포인터를 일반화한 것을 말합니다. vector, map 등 각각 다르게 구현된 컨테이너들을 일반화된 이터레이터를 통해 쉽게 순회할 수 있습니다.



다만 주소값을 바로 반환하지는 못하며 &*를 통해 한단계 더 거쳐서 가리키는 해당 요소의 주소값을 반환할 수 있습니다.

```
#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';
    }
    for(auto it = v.begin(); it != v.end(); it++){
        cout << *it << ' ';
    }
    cout << '\n';
    for(vector<int>::iterator it = v.begin(); it != v.end(); it++){
        cout << *it << ' ';
    }
    auto it = v.begin();
    advance(it, 3);
    cout << '\n';
    cout << *it << '\n';

    // cout << v.begin() << '\n'; //예외
}
```

```
/*
0번째 요소 : 1
0x135e067f0
1번째 요소 : 2
0x135e067f4
2번째 요소 : 3
0x135e067f8
3번째 요소 : 4
0x135e067fc
4번째 요소 : 5
0x135e06800
1 2 3 4 5
1 2 3 4 5
4
*/
```

앞의 코드를 보면 `vector<int>:: iterator`를 `auto`로도 선언할 수 있는 것을 알 수 있는데 `vector<int>:: iterator`가 너무 길기 때문에 간단하게 이터레이터를 선언하기 위해 `auto`로 선언한 것입니다.

이터레이터의 함수 중 많이 쓰는 함수로는 `begin()`과 `end()`, `advance()` 3개가 있습니다.

begin()

컨테이너의 시작 위치를 반환합니다.

end()

컨테이너의 끝 다음의 위치를 반환하는 데 사용됩니다. 보통 `it != v.end()` 이런 식의 코드를 많이 보셨을 텐데요. 컨테이너를 다 순회하고 컨테이너의 끝에 도착했다는 것을 가리킵니다.

advance(iterator, cnt)

해당 `iterator`를 `cnt`까지 증가시킵니다.

Q. 이터레이터와 포인터의 차이

이터레이터는 어떠한 컨테이너(배열, 맵 등)의 범위 안에서 일부 요소를 가리키며 해당 요소들을 순회할 수 있는 개체입니다. 이는 컨테이너의 개체를 참조하는 것이기 때문에 이 자체를 제거할 수 없습니다.

반면 포인터는 변수의 메모리 주소를 저장하는 개체이며 포인터는 delete를 통해 포인터를 제거할 수 있습니다.

Q. 이터레이터 = 일반화된 포인터 ?

먼저 일반화(generalization)는 여러 사례들의 공통되는 속성을 일반적인 개념으로 추상화의 한 형태를 말합니다.

이터레이터는 컨테이너의 구조나 컨테이너 안에 들어가 있는 요소의 타입과는 상관없이 컨테이너에 저장된 데이터를 순회하는 과정을 담당합니다. 즉, 각각의 다른 요소들을 쉽게 탐색할 수 있게 “일반화”한 장치입니다.

1.7 함수

코딩테스트에 자주 나오며 중요한 함수들을 알아봅시다.

fill()과 memset()

fill()과 memset()은 배열을 초기화 할 때 쓰입니다. fill은 모든 값으로 초기화 할 수 있습니다. 0, 1, 100 등 모든 숫자로 초기화가 가능합니다. 반면, memset()같은 경우 -1, 0 으로만 초기화가 가능합니다. memset()을 쓰다보면 0, -1 이외의 값으로 초기화하다 틀리는 경우가 있어 fill()을 추천하지만 memset도 배워두는게 좋습니다.

작은 차이지만 0, -1로 초기화하는 경우 fill보다 memset이 더 빠르기 때문에 이 때문에 시간적으로 더 최적화를 시킬 수도 있거든요.

fill()

fill()은 $O(n)$ 의 시간복잡도를 가지며 fill(시작값 - first, 끝값 - last, 초기화하는값 - val)로 배열에 들어가는 값을 초기화합니다. 모든 값을 기반으로 초기화가 가능하며 [first, last)까지 val로 초기화합니다.

fill()로 배열의 모든 값이 아닌 일부값을 초기화하는 경우도 있지만 보통은 전체를 초기화하는게 좋습니다.

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

[참고] [은 포함하다라는 수학적기호,)은 포함하지 않는다는 기호입니다. 즉, 시작값은 포함하고, 끝값은 포함하지 않고 초기화한다는 의미입니다.

```
#include <bits/stdc++.h>
using namespace std;
int a[10];
int b[10][10];
int main() {
    fill(&a[0], &a[10], 100);

    for(int i = 0; i < 10; i++){
        cout << a[i] << " ";
    }
    cout << '\n';
    fill(&b[0][0], &b[9][10], 2);
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
            cout << b[i][j] << " ";
        }
        cout << '\n';
    }
    return 0;
}
/*
100 100 100 100 100 100 100 100 100 100
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
*/
```

앞의 코드를 보면 a[0]부터 시작해 a[9]까지 초기화하고 싶기 때문에 a[0], a[10] 이 두개의 인자를 fill() 함수의 인자로 집어넣는 것을 볼 수 있습니다. 그래야 last인 a[10]은 포함하지 않고 그 이전값인 a[9]까지 전부 초기화하는 것을 볼 수 있습니다.

다음코드처럼 배열의 이름을 기반으로 초기화할 수 있습니다.

```
#include <bits/stdc++.h>
using namespace std;
int a[10];
int b[10][10];
int main() {
```

1차원의 경우 a , $a + 10$, 즉, 배열의 이름 + 숫자로 가능하지만 2차원 이상일 경우에는 반드시 $\&b[0][0]$ + 숫자로 해야 한다는 것을 기억해주세요.

왜 fill()로 전체초기화를 해야할까?

앞서서 `fill()`은 배열의 전체초기화를 해야한다고 했습니다.

왜 그럴까요?

다음 코드처럼 저는 8×8 정사각형으로 초기화를 시키고 싶어서 다음과 같이 초기화를 시켰을 뿐인데 제 생각과는 다르게 초기화가 되는 것을 볼 수 있습니다.

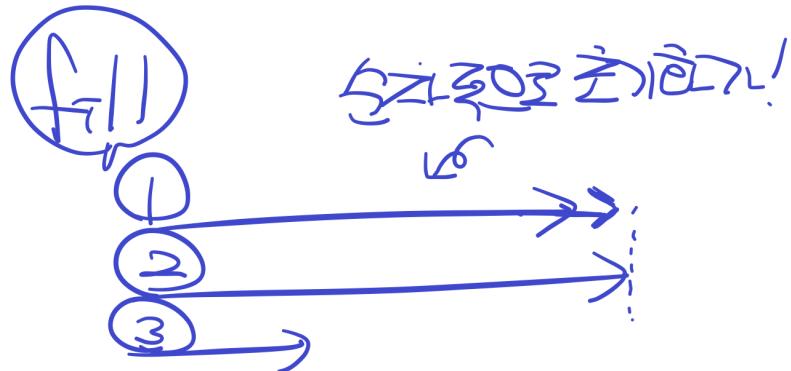
```
#include<bits/stdc++.h>
using namespace std;
int a[10][10];
int main(){
    cin.tie(NULL); cout.tie(NULL);
    fill(&a[0][0], &a[0][0] + 8 * 8 , 4);
    for(int i = 0; i < 10; i++){
```

```

        for(int j = 0; j < 10; j++){
            cout << a[i][j] << " ";
        }
        cout << '\n';
    }
    return 0;
}
/*
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4
4 4 4 4 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
*/

```

이는 초기화를 시킬 때 순차적으로 1열에 있는 요소들을 초기화하고 그 다음 2열, 3열 이런식으로 초기화가 일어나기 때문에 그렇습니다.



따라서 전체 초기화를 하는게 좋습니다.

memset()

memset()은 바이트단위로 초기화를 하며 0, -1, char형의 하나의 문자(a, b, c, 등..)으로 초기화를 할 때만 사용합니다.

```
void * memset ( void * ptr, int value, size_t num );
```

memset(배열의 이름, k, 배열의 크기) 이렇게 사용합니다.

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

```

using namespace std;
const int max_n = 1004;
int a[max_n];
int a2[max_n][max_n];
int main() {
    memset(a, -1, sizeof(a));
    memset(a2, 0, sizeof(a2));
    for(int i = 0; i < 10; i++) cout << a[i] << " ";
    return 0;
}
// -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 %

```

앞의 코드를 보듯이 fill 보다는 간편하게 초기화하는 것을 볼 수 있습니다. (특히 2차원배열은요.)

0 또는 -1이란 값으로 초기화할 때는 memset을 쓰는 것이 좋습니다.

그러나 0, -1 이외의 숫자는 절대 이 memset()으로 초기화 못하니 주의해주세요!

쓰지 말아야 할 초기화 방법 {0, }

간혹가다 이런 코드를 보신 적이 있을 겁니다.

```
int a[5] = {0, };
```

앞의 코드 또한 0으로 초기화 한다는 의미입니다. int형 a라는 5의 크기를 가진 int형 a라는 배열을 0으로 모두 초기화한다는 의미이지요.

문법은 다음과 같습니다.

```
T myarray[N] = {0, };
```

하지만 이렇게 초기화하는 것은 초반에 한번하는 정적초기화로써만 유효합니다. 동적초기화로써는 동작하지 않습니다.

예를 들어 다음과 같은 코드에서 초기화는 잘 동작하지 않습니다.

```

#include <bits/stdc++.h>
using namespace std;
int main()
{
    int cnt = 0;
    int a[5] = {0, };
    while(++cnt != 10){
        for(int i = 0; i < 5; i++) a[i] = i;
        a[5] = {0, };
        for(int i : a) cout << i << ' ';
        cnt++;
    }
}

```

```

    return 0;
}
// 0으로 초기화가 되지 않음.
// 0 1 2 3 4

```

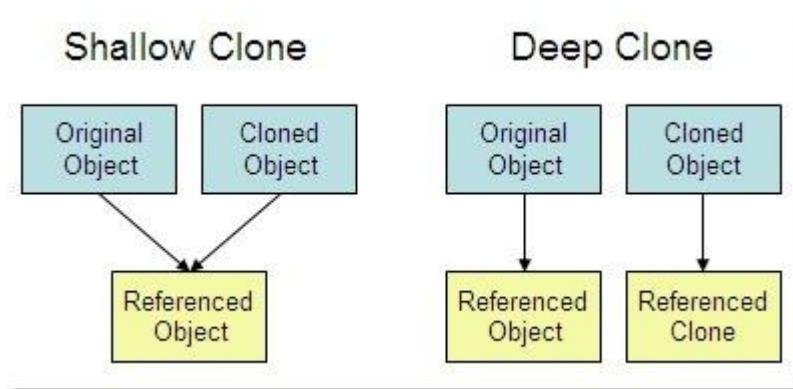
따라서 이런 실수를 방지하기 위해 memset()이나 fill()을 쓰는게 좋습니다.

memcpy()와 copy()

어떤 변수를 깊은 복사할 때 memcpy()와 copy()를 씁니다. memcpy()는 Array끼리 복사할 때 쓰고 copy()는 Array, vector에 모두 쓰입니다.

[참고] 얕은 복사와 깊은 복사

참고로 얕은 복사(Shallow copy)는 메모리 주소값을 복사한 것이라 복사한 배열을 수정하면 원본 배열이 수정되는 복사방법이며 깊은 복사(Deep copy)는 새로운 메모리 공간을 확보해 완전히 복사해 복사한 배열을 수정하면 원본 배열은 수정되는 않는 복사방법을 의미합니다.



memcpy()

memcpy()는 어떤 변수의 메모리에 있는 값들을 다른 변수의 “특정 메모리값”으로 복사할 때 사용합니다. Array를 깊은 복사할 때 쓰입니다.

```
void * memcpy ( void * destination, const void * source, size_t num );
```

이처럼 v라는 Array를 ret에다가 복사하는 것을 볼 수 있습니다.

```

#include<bits/stdc++.h>
using namespace std;
int main(void) {
    int v[3] = {1, 2, 3};
    int ret[3];
    memcpy(ret, v, sizeof(v));
    cout << ret[1] << "\n";
    ret[1] = 100;
}

```

```

        cout << ret[1] << "\n";
        cout << v[1] << "\n";
    return 0;
}
/*
2
100
2
*/

```

제대로 깊은 복사가 되어 `ret`을 수정하더라도 `vector v`는 수정되지 않는 것을 볼 수 있습니다.

예를 들어 `a`라는 원본배열이 수정되는 로직이 있습니다. 근데 그 다음에 이 원본배열이 수정되지 않은 상태값을 기반으로 또 어떠한 로직이 필요할 때가 있습니다.

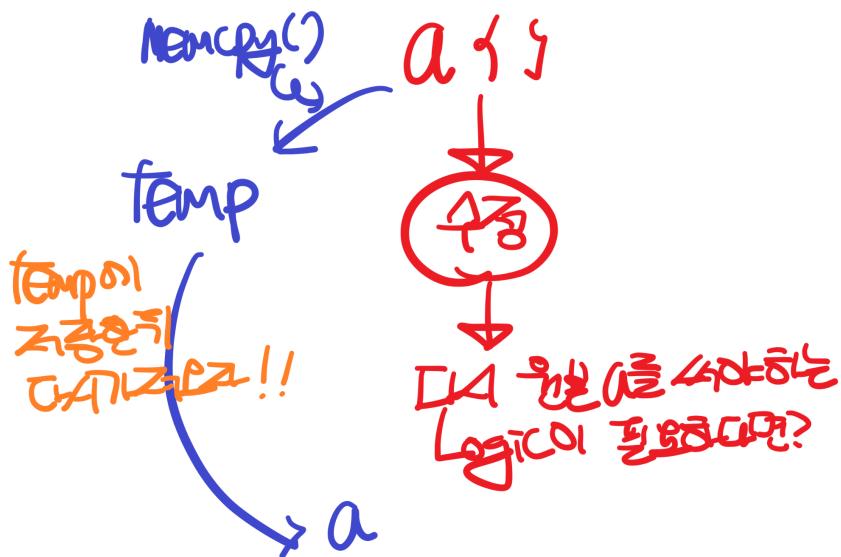
그럴 때 `memcpy`를 주로 씁니다. (물론 다른 이유로도 쓰지만요.)

아래의 모습은 `temp`라는 배열에 `a`를 담아두고 `a`를 수정하는 로직을 구현한 뒤 `a`라는 배열에 다시 예전 온전한 `a`를 담은 `temp`를 이용해 다시 `a`를 만드는 모습입니다.

```

#include <bits/stdc++.h>
using namespace std;
int a[5], 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 << ' ';
    cout << '\n';
    // 원본 배열 a를 수정하여 출력하는 로직
    // a를 수정해서 ~~를 더하는 로직이 될 수 있겠죠?
    a[4] = 1000;
    for(int i : a) cout << i << ' ';
    cout << '\n';
    // 그 다음 다시 temp를 기반으로 원본배열을 담아 둠.
    memcpy(a, temp, sizeof(temp));
    for(int i : a) cout << i << ' ';
    cout << '\n';
    return 0;
}
/*
0 1 2 3 4
0 1 2 3 1000
0 1 2 3 4
*/

```



하지만 주의해야할 점이 있습니다. `memcpy()`는 `vector`에서는 깊은 복사가 되지 않습니다.

```
#include<bits/stdc++.h>
using namespace std;
int main(void) {
    vector<int> v {1, 2, 3};
    vector<int> ret(3);
    memcpy( &ret, &v, 3*sizeof(int) );

    cout << ret[1] << "\n";
    ret[1] = 100;
    cout << ret[1] << "\n";
    cout << v[1] << "\n";
    return 0;
}
/*
2
100
100
*/
```

앞의 코드처럼 `ret[1]`을 수정했더니 `v[1]`도 수정되는 것을 볼 수 있습니다.

이는 `memcpy()`는 TriviallyCopyable인 타입이 아닌 경우 함수 자체가 제대로 동작하지 않습니다.

[참고] `memcpy()`의 TriviallyCopyable

Copies count bytes from the object pointed to by src to the object pointed to by dest.

If the objects overlap, the behavior is undefined. If the objects are not trivially copyable (e.g. scalars, arrays, C-compatible structs), the behavior is undefined.

다음 코드처럼 `is_trivial`를 통해 해당 타입이 TriviallyCopyable한지 확인할 수 있는데 `vector`는 그렇지 않는 것을 볼 수 있습니다.

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

int main()
{
    if (is_trivial<vector<int>>())
        cout << "Hello Kundol!\n";
}
// kundol이 출력되지 않음
```

`memcpy`는 `Array`에서만 동작한다는 것을 기억해주세요.

copy()

`memcpy`()와 똑같은 동작을 하는 함수입니다. `vector`와 `Array` 모두 쓰일 수 있습니다.

```
copy (InputIterator first, InputIterator last, OutputIterator result)
```

만약 `vector v`를 `ret`에다가 옮기고 싶다면 다음과 같이 하면 됩니다.

`v` : 복사당하는 `vector` / `ret` : 복사하는 `vector`

```
copy(v.begin(), v.end(), ret.begin());
```

```
#include<bits/stdc++.h>
using namespace std;
int main(void) {
    vector<int> v {1, 2, 3};
    vector<int> ret(3);
    copy(v.begin(), v.end(), ret.begin());
    cout << ret[1] << "\n";
    ret[1] = 100;
    cout << ret[1] << "\n";
    cout << v[1] << "\n";
    return 0;
}
/*
2
100
2
*/
```

다음과 같이 `ret`이란 `vector`에 `vector v`의 값이 잘 들어간 것을 볼 수 있습니다. 이 때 복사하는 `vector`와 복사당하는 `vector`의 크기를 맞춰주는 것이 중요합니다. `v`의 크기는 3이며, `ret`의 크기도 3으로 설정된 것을 볼 수 있습니다. 그리고 깊은 복사가 되어 `ret`을 수정하더라도 `v`에는 아무런 영향을 미치지 않는 것을 볼 수 있습니다.

Array는 다음과 같이 쓰입니다.

```
#include<bits/stdc++.h>
using namespace std;
int n = 3;
int main(void) {
    int v[n] = {1, 2, 3};
    int ret[n];
    copy(v, v + n, ret);
    cout << ret[1] << "\n";
    ret[1] = 100;
    cout << ret[1] << "\n";
    cout << v[1] << "\n";
    return 0;
}
/*
2
100
2
*/
```

만약 Array `v`의 크기가 5라면 `copy(v, v + 5, ret)` 이런식의 코드가 되겠죠? +숫자는 해당 배열의 크기인 것을 주의해주세요.

sort()

`sort()`는 배열 등 컨테이너들의 요소를 정렬하는 함수입니다. 보통 array나 `vector`를 정렬할 때 쓰이며 $O(n \log n)$ 의 시간복잡도를 가지는 함수입니다. `sort()`에 들어가는 매개변수로는 3가지가 있으며 2개는 필수로 넣어야 하며 한개는 선택이며 커스텀 정렬하고 싶을 때 넣습니다.

sort(first, last, *커스텀비교함수)

이렇게 들어갑니다. `first`는 정렬하고 싶은 배열의 첫번째 이터레이터, `last`는 정렬하고 싶은 배열의 마지막 이터레이터를 넣으면 됩니다. 또한 `[first, last)`라는 범위를 갖습니다. 즉, `first`는 포함하고 `last`는 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”的 위치를 넣어주어야 합니다.

다시 말하자면 a[0] + 5는 배열의 마지막원소가 아닙니다. 마지막원소는 a[0] + 4이겠죠?
또한 내가 만약 3번째 요소까지만 정렬하고 싶다면 sort(a[0], a[0] + 3)이렇게 하면 됩니다.
그렇게 하면 a[0] + 2 까지 정렬됩니다.

sort()의 세번째 매개변수, 커스텀비교함수를 넣지 않으면 기본적으로 오름차순이며 이를 커스텀비교함수에 greater<타입>()를 넣어 내림차순으로 변경할 수 있습니다. 참고로 less<타입>()을 통해 오름차순으로 정렬할 수도 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
*/

```

또한 pair를 기반으로 만들어진 vector의 경우 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) 할 수도 있습니다. vector v에 있는 “요소”들을 끄집어내서 순회한다는 의미죠. v[0], v[1] 따위로 접근한다는 의미입니다.

자 그렇다면 first는 내림차순, second는 오름차순 정렬하고 싶다면 어떻게 해야할까요?
바로 커스텀 비교함수 cmp를 만들어 매개변수로 넣으면 됩니다. (보통 cmp라는 함수명을
많이 씁니다.)

```
#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
*/
```

원래는 sort()를 하면 first가 오름차순으로 정렬되지만 first를 기준으로 내림차순으로 정렬된
것을 볼 수 있습니다.

unique()

0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

unique는 범위안의 있는 요소 중 앞에서부터 서로를 비교해가며 중복되는 요소를 제거하고
나머지 요소들은 삭제하지 않고 그대로 두는 함수입니다. O(n)의 시간복잡도를 가집니다.

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

```

        v.push_back(i);
        v.push_back(i);
    }
    for(int i : v) cout << i << " ";
    cout << '\n';
    // 중복되지 않은 요소로 채운 후, 그 다음 이터레이터를 반환한다.
    auto it = unique(v.begin(), v.end());
    cout << it - v.begin() << '\n';
    // 앞에서부터 중복되지 않게 채운 후 나머지 요소들은 그대로 둔다.
    for(int i : v) cout << i << " ";
    cout << '\n';
    return 0;
}
/*
1 1 2 2 3 3 4 4 5 5
5
1 2 3 4 5 3 4 4 5 5
*/

```

하나 더 예를 들어볼게요.

```

#include <bits/stdc++.h>
using namespace std;
vector<int> v {1, 1, 2, 2, 3, 3, 5, 6, 7, 8, 9};
int main () {
    auto it = unique(v.begin(), v.end());
    for(int i : v) cout << i << " ";
    cout << '\n';
    return 0;
}
/*
1 2 3 5 6 7 8 9 7 8 9
*/

```

자 이 경우 unique()는

1 1 서로를 비교하며 1

2 2 서로를 비교하며 2

3 3 서로를 비교하며 3

그리고 쪽 5, 6, 7, 8, 9를 배열에 채워넣습니다.

채워넣지 않은 영역의 요소는 건드리지 않구요.

그래서 1 2 3 5 6 7 8 9 7 8 9이 반환됩니다.

뭔가 쓰기 어렵죠? 그렇기 때문에 unique()를 쓸 경우 꼭 sort()와 같이 써야 합니다.

왜냐하면 앞의 경우 처럼 앞에서 부터 서로를 비교해가며 중복된 요소를 제거하기 때문에 sort()를 써야 “우리가 예상하는 로직”인 중복된 수를 제거한 배열이 나오게 됩니다.

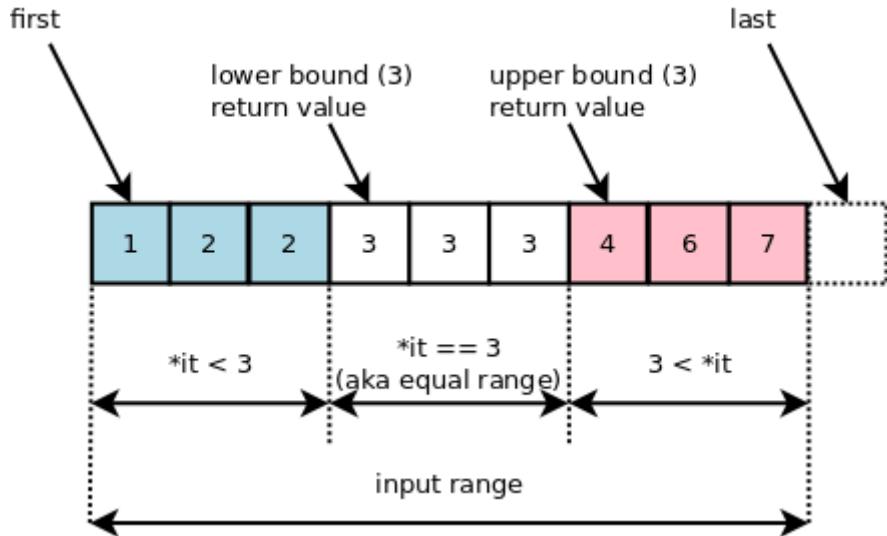
따라서 다음코드와 같이 unique()는 sort(), erase(unique()) 와 함께 쓰는 것이 좋습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    vector<int> s {4, 3, 3, 5, 1, 2, 3};
    s.erase(unique(s.begin(),s.end()),s.end());
    for(int i : s) cout << i << " ";
    cout << '\n';
    vector<int> s2 {4, 3, 3, 5, 1, 2, 3};
    sort(s2.begin(), s2.end());
    s2.erase(unique(s2.begin(),s2.end()),s2.end());
    for(int i : s2) cout << i << " ";
    return 0;
}
/*
4 3 5 1 2 3
1 2 3 4 5
*/
```

lower_bound() 와 upper_bound()

정렬된 배열에서 어떤 값이 나오는 첫번째 지점 또는 초과하는 지점의 위치를 찾으려면 어떻게 해야할까요?

또한 이분탐색을 쉽게 함수로 구현하려면 어떻게 해야할까요?



이를 쉽게 해주는 함수인 `lower_bound()`와 `upper_bound()`를 알아봅시다.

[참고] 이 함수들은 꼭 정렬된 배열에서만 써야 합니다. 정렬되지 않은 배열을 기반으로 하게 되면 예상하지 못한 결과가 나오게 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main(){
    vector<int> a {1, 2, 3, 3, 3, 4};
    cout << lower_bound(a.begin(), a.end(), 3) - a.begin() << "\n"; // 2
    cout << upper_bound(a.begin(), a.end(), 3) - a.begin() << "\n"; // 5
    return 0;
}
```

앞의 코드처럼 숫자 3을 찾는다고 했을 때 `lower_bound()`는 2, `upper_bound()`는 5를 반환하고 있습니다. 3이 시작되는 최소 시작점은 `lower_bound()`, 이를 초과하는 지점은 `upper_bound()`로 찾을 수가 있습니다.

`a.begin()`은 왜 빼는 걸까요? `lower_bound()`, `upper_bound()`는 해당 자료형으로부터 이터레이터를 반환합니다. 따라서 몇번째를 추려내려면 이 이터레이터에서 `begin()`을 빼주어야 합니다.

`lower_bound()`로 나오는 이터레이터가 어떤 값을 갖는지 확인해볼까요? 바로 확인은 안되고 &*를 통해 확인할 수 있습니다.

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

```

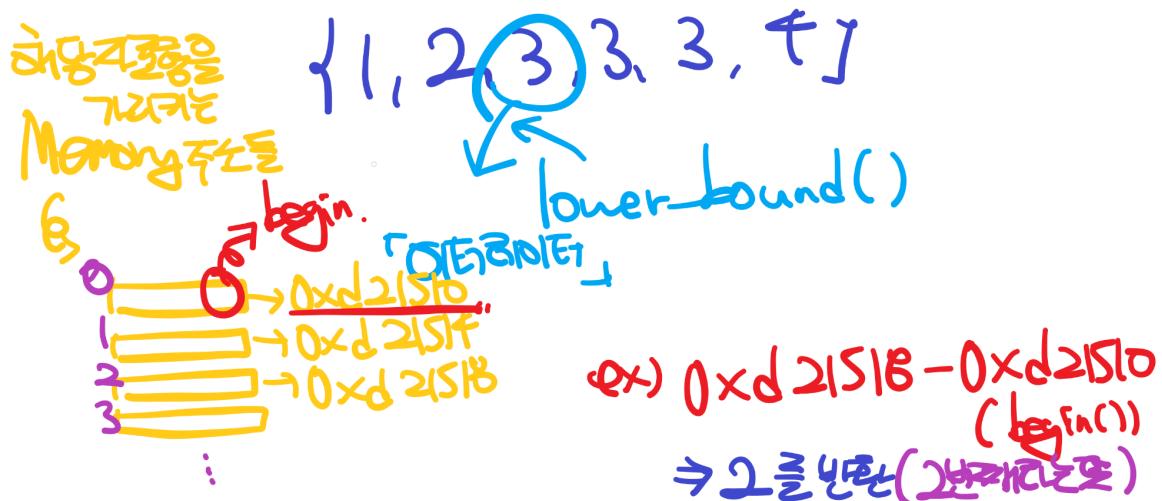
vector<int> a {1, 2, 3, 3, 3, 4};
cout << &*lower_bound(a.begin(), a.end(), 3) << "\n";
cout << &a.begin() << "\n";
cout << &(a.begin() + 1) << "\n";
return 0;
}
/*
0xd21518
0xd21510
0xd21514
*/

```

참고로 주소값은 실행환경에 따라 달라질 수 있습니다.

이런식으로 어떤 주소값을 반환하는 것을 알 수 있습니다.

다음 그림처럼 배열에 쌓아진 요소들은 각각의 주소값을 가지고 있습니다. 이 주소값을 기반으로 몇번째 요소인지를 뽑아낼 수 있죠.



[참고] 주소값끼리 -하게 되면 해당 주소값에서 몇번째에 이 요소가 들어있음을 반환하게 됩니다.

다음코드처럼도 할 수 있습니다. (이 방법은 보통은 사용되지는 않습니다.)

```

#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int main(){
    vector<int> a {1, 2, 3, 3, 3, 4};
    cout << &*lower_bound(a.begin(), a.end(), 3) - &a.begin() << "\n";
    vector<int> b {0, 0, 0, 0};
    cout << &(b.begin() + 3) - &b.begin() << '\n';
}

```

```
    return 0;
}
/*
2
3
*/
```

다음 코드를 통해 lower_bound가 가리키는 요소를 출력할 수도 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
vector<int> a {1, 2, 3, 3, 4, 100};
int main(){
    cout << *lower_bound(a.begin(), a.end(), 100) << "\n";
    return 0;
}
/*
100
*/
```

또한 이를 응용해서 숫자 3이 몇개 들어가 있는지도 확인할 수 있죠.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
vector<int> a {1, 2, 3, 3, 3, 3, 4, 100};
int main(){
    cout << upper_bound(a.begin(), a.end(), 3) - lower_bound(a.begin(), a.end(),
3) << "\n";
    return 0;
}
/*
4
*/
```

만약 해당 요소가 없을 경우 다음코드처럼 그 근방지점을 반환합니다.

```
#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";
// 2 3 4 5 7
// 0 1 2 3 4 에서 근방지점인 4번째 (7보다 6이 더 작으므로)
cout << lower_bound(v.begin(), v.end(), 6) - v.begin() << "\n";
// 2 3 4 5 7
// 0 1 2 3 4 에서 근방지점인 4번째 (7보다 6이 더 작으므로)
cout << upper_bound(v.begin(), v.end(), 9) - v.begin() << "\n";
// 2 3 4 5 7
// 0 1 2 3 4 에서 근방지점인 5번째(7보다 9가 더 크므로)
cout << lower_bound(v.begin(), v.end(), 9) - v.begin() << "\n";
// 2 3 4 5 7
// 0 1 2 3 4 에서 근방지점인 5번째(7보다 9가 더 크므로)
cout << upper_bound(v.begin(), v.end(), 0) - v.begin() << "\n";
// 2 3 4 5 7
// 0 1 2 3 4 에서 근방지점인 0번째(0보다 2가 더 크므로)
cout << lower_bound(v.begin(), v.end(), 0) - v.begin() << "\n";
// 2 3 4 5 7
// 0 1 2 3 4 에서 근방지점인 0번째(0보다 2가 더 크므로)
}

/*
4
4
5
5
0
0
*/

```

accumulate()

배열의 합을 쉽고 빠르게 구해주는 함수로는 accumulate가 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

}

```

함수하나로 vector v의 요소들의 합을 쉽게 구하는 모습입니다.

max_element()

배열 중 가장 큰 요소를 추출하는 함수, max_element입니다. 이 함수는 이터레이터를 반환하기 때문에 *를 통해 값을 끄집어낼 수 있고 이를 기반으로 해당 최댓값의 인덱스 또한 뽑아낼 수 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());
    auto b = max_element(v.begin(), v.end());
    cout << a << '\n'; // 10
    cout << (int)(b - v.begin()) << '\n'; // 9
}
```

max_element의 implement

```
template <class ForwardIterator>
ForwardIterator max_element ( ForwardIterator first, ForwardIterator last )
{
    if (first==last) return last;
    ForwardIterator largest = first;

    while (++first!=last)
        if (*largest<*first)
            largest=first;
    return largest;
}
```

앞의 코드처럼 ForwardIterator라는 이터레이터를 반환하는 모습을 볼 수 있습니다.

min_element()

배열 중 가장 작은 요소를 추출하는 함수, min_element입니다. 이 함수는 이터레이터를 반환하기 때문에 *를 통해 값을 끄집어낼 수 있고 이를 기반으로 해당 최솟값의 인덱스 또한 뽑아낼 수 있습니다.

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

int main(){
    vector<int> v = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

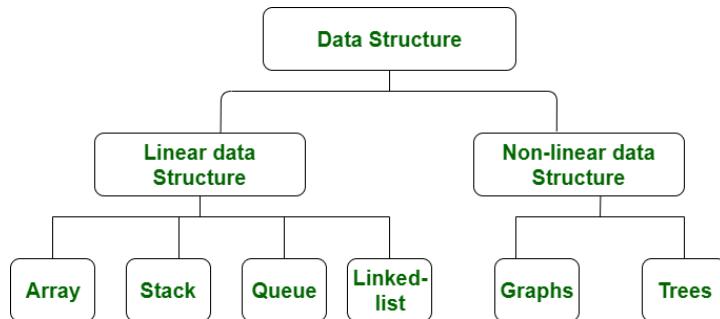
```

int a = *min_element(v.begin(), v.end());
auto b = min_element(v.begin(), v.end());
cout << a << '\n'; // 1
cout << (int)(b - v.begin()) << '\n'; // 0
}

```

1.8 자료구조

자료구조란 데이터들과의 관계, 함수, 명령 등의 집합을 의미하며 데이터에 대해 효율적으로 접근, 수정 등 데이터의 처리를 효율적으로 할 수 있게 만든 구조를 의미합니다.

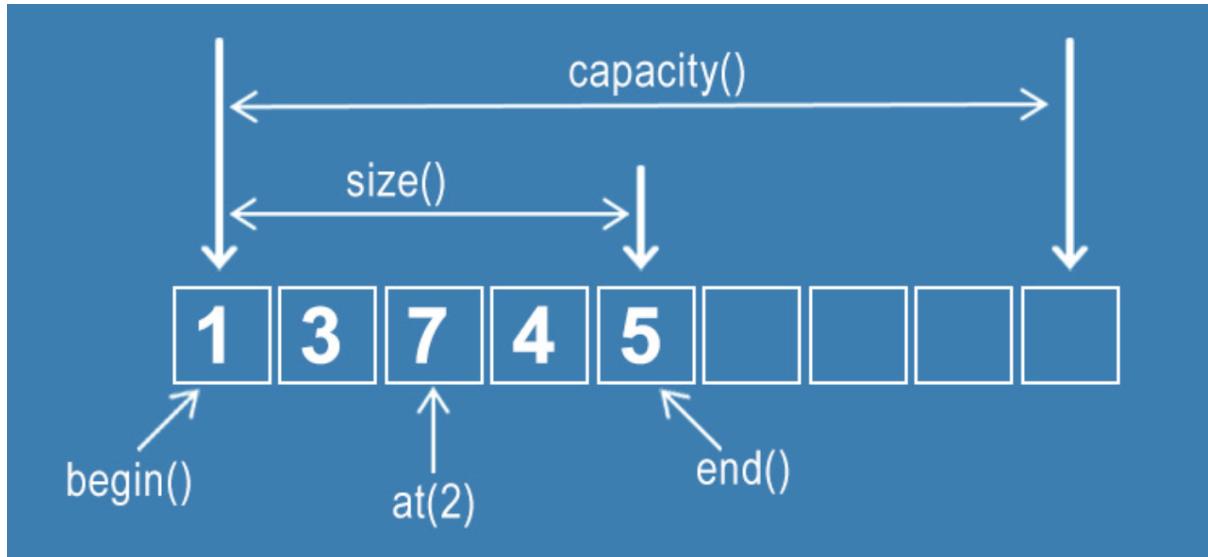


선형자료구조로는 Array, Stack, Queue, Linked_list가 있으며 비선형자료구조로는 Graph, Tree가 있습니다.

vector

vector는 동적으로 요소를 할당할 수 있는 동적 배열입니다.

만약 컴파일 시점에 사용해야 할 요소들의 개수를 모른다면 vector를 써야 합니다. 연속된 메모리 공간에 위치한 같은 타입의 요소들의 모음이며 숫자인덱스를 기반으로 랜덤접근이 가능하며 중복을 허용합니다.



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.

```
vector<타입> 변수명;
```

앞의 코드처럼 선언되어서 쓰입니다.

예를 들어 `int`형 `vector`를 정의하고 싶다면 `vector<int>`로 정의해야 합니다.

이는 다른 자료구조도 동일합니다. 예를 들어 제가 `string`을 담는 `set`을 정의하고 싶다면 `set<string>`으로 정의해야 합니다.

다음 코드는 `vector`의 예입니다.

```
#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() + 3);

    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
4 5 6 7 8 9
not found
10 10 10 10 10 10
아무것도 없을까?
*/

```

push_back()

vector의 뒤에서부터 요소를 더합니다. 참고로 push_back()과 같은 기능을 하는 emplace_back()도 있습니다.

[참고] emplace_back()이 더 빠르지만 시간차이는 별로 나지 않습니다.

pop_back()

vector의 맨 뒤의 요소를 지웁니다.

erase()

```

iterator erase (const_iterator position);
iterator erase (const_iterator first, const_iterator last);

```

erase로 한 요소만을 지운다면 erase(위치)로 쓰이지만 [from, to)로 지우고 싶다면 erase[from, to)를 통해 지웁니다.

[참고] (와)은 해당요소를 포함하지 않는 구간, [와]는 해당요소를 포함한다는 수학적 기호입니다.

[a, b] 폐구간 { $x | a \leq x \leq b$ }

find(from, to, value)

vector의 메서드가 아닌 STL 함수입니다. [from, to) 에서 value를 찾습니다. vector 내의 요소들을 찾고 싶을 때 이를 통해 찾습니다. O(n)의 시간복잡도를 가집니다.

clear()

vector의 모든 요소를 지웁니다.

fill(from, to, value)

vector 내의 value로 값을 할당하고 싶다면 fill을 써서 채웁니다. 보통 이를 ~~한 값으로 초기화라고 부릅니다. [from, to) 구간에 value를 초기화 합니다.

범위기반 for 루프

C++11부터 범위기반 for 루프가 추가되어서 이를 쓸 수 있습니다. vector만 쓸 수 있는 것은 아니고 순회할 수 있는 컨테이너인 vector, Array, map 등도 사용 가능합니다.

```
for ( range_declaration : range_expression )
    loop_statement
```

문법은 다음과 같습니다.

```
for(<해당 컨테이너에 들어있는 타입> 임시변수명 : 컨테이너)
```

다음 코드처럼 사용됩니다. 만약 vector안에 pair가 들어간다면 pair를 써서 범위기반 for루프를 써야 합니다.

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

using namespace std;
vector<int> v {1, 2, 3};
int main(){
    for(int a : v) cout << a << " ";
    cout << "\n";
    for(int i = 0; i < v.size(); i++) cout << v[i] << ' ';
    cout << "\n";
    vector<pair<int, int>> v2 = {{1, 2}, {3, 4}};
    for(pair<int, int> a : v2) cout << a.first << " ";
}
```

```
/*
1 2 3
1 2 3
1 3
*/
```

다음 코드는 `vector<int>` 내의 있는 요소들을 탐색한다는 코드입니다.

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

이는 다음 코드와 같은 의미입니다.

```
for(int i = 0; i < v.size(); i++) v[i]
```

vector의 정적할당

`vector`라고 해서 무조건 크기가 0인 빈 `vector`를 만들어 동적할당으로 요소를 추가하는 것은 아닙니다. 애초에 크기를 정해놓거나 해당 크기에 대해 어떠한 값으로 초기화 해놓고 시작할 수도 있습니다.

다음 코드는 5개의 요소를 담을 수 있는 `vector`를 선언하고 모든 값을 100으로 채운 모습입니다.

```
#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
*/
```

또한 이런 식으로도 가능합니다.

```
#include <bits/stdc++.h>
using namespace std;
vector<int> v{10, 20, 30, 40, 50};
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
```

```

cout.tie(NULL);
for(int i : v){
    cout << i << " ";
}
return 0;
}
//10 20 30 40 50

```

2차원 배열

vector를 이용한 2차원 배열을 만드는 3가지 방법입니다.

```

#include<bits/stdc++.h>
using namespace std;
vector<vector<int>> v;
vector<vector<int>> v2(10, vector<int>(10, 0));
vector<int> v3[10];
int main(){
    for(int i = 0; i < 10; i++){
        vector<int> temp;
        v.push_back(temp);
    }
    return 0;
}

```

v는 vector안의 vector가 들어가 있는 2차원 배열 타입을 선언합니다.

그 이후 v에 temp라는 vector를 push_back해서 2차원 배열을 만듭니다.

v2는 10 * 10 짜리 크기의 2차원배열을 바로 만듭니다. 0으로 초기화까지 한 것을 볼 수 있습니다.

v3는 10개 짜리 배열을 선언한 것을 볼 수 있습니다. 이는 v와 똑같은 2차원배열입니다. v 같은 경우 vector 10개를 담으니까요.

다음 그림을 참고해주세요.

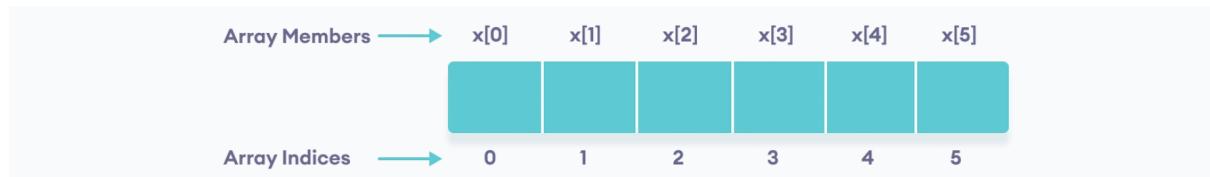
V : $\{ \underbrace{\dots}_{\text{push}} \}$ $\xrightarrow{\text{temp라는 빈배열}} \{ 1, 1, 1, \dots \}$
 $x[0]$

V2 : $\{ \{ 0, 0, 0, \dots \}, \{ 0, 0, 0, \dots \}, \dots \}$
 0으로 초기화한 size=10짜리

V3: $\{ \{ 1, 1, 1, 1, \dots \} \}$
 1로 초기화 빈배열 10짜리를
 넣는다.

Array

정적 배열입니다. 선언할 때 보통 크기를 설정해서 연산을 진행합니다. 연속된 메모리 공간에 위치한 같은 타입의 요소들의 모음이며 숫자인덱스를 기반으로 랜덤접근이 가능하며 중복을 허용합니다.



선언타입은 c스타일과 std스타일이 있는데 이 강의에서는 c스타일을 중심으로 배웁니다. c스타일은 `int a[10]` 이렇게 선언하는 것이며 std스타일은 `array<int, 10> a;` 이렇게 선언하는 것을 말합니다.

vector와는 달리 메서드가 없습니다. 배열의 크기를 정해서 선언할 수도 있습니다.

ex) `int a[3]`

크기를 정하지 않고 선언하되 배열을 중괄호로 요소들을 할당할 수도 있습니다.

ex) `int a2[] = { 1, 2, 3, 4 }`

앞의 코드처럼 하게 되면 자동적으로 rvalue의 크기로 할당되어 `int a2[]`는 `int a2[4]`와 같은 의미를 가지게 됩니다.

```

#include <bits/stdc++.h>
using namespace std;
int a[3] = {1, 2, 3};
int a2[] = {1, 2, 3, 4};
int a3[10];
int main(){
    for(int i = 0; i < 3; i++) cout << a[i] << " ";
    cout << '\n';
    for(int i : a) cout << i << " ";

    for(int i = 0; i < 4; i++) cout << a2[i] << " ";
    cout << '\n';
    for(int i : a2) cout << i << " ";

    for(int i = 0; i < 10; i++) a3[i] = i;
    cout << '\n';
    for(int i : a3) cout << i << " ";
    return 0;
}
/*
1 2 3
1 2 3 1 2 3 4
1 2 3 4
0 1 2 3 4 5 6 7 8 9
*/

```

지금까지 설명한 Array, vector와 같은 배열은 같은 타입의 변수들로 이루어져 있고, 크기가 정해져 있으며, 인접한 메모리위치에 있는 데이터들을 모아놓은 집합입니다. C++에서는 vector나 array로 구현을 하죠.

2차원배열과 탐색을 빠르게 하는 팁

2차원배열은 단순하게 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
*/

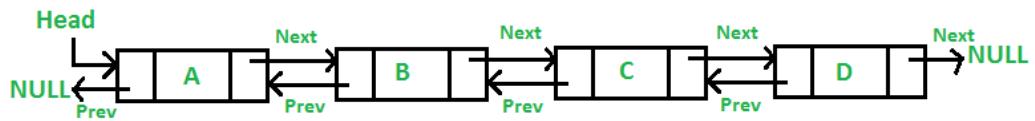
```

2차원배열을 만들었고 출력한 예제입니다. 이 때 중요한 점이 있는데 앞의 코드에 good과 bad 코드에 주목해주세요.

첫번째 차원 >> 2번째 차원 순으로 탐색하는게 성능이 좋습니다. 이는 C++에서 캐시를 첫번째 차원(여기서는 y가 되겠죠?) 를 기준으로 하기 때문에 캐시관련 효율성 때문에 탐색을 하더라도 순서를 지켜가며 해주는게 좋습니다.

list

연결리스트(linked list)입니다. 요소가 인접한 메모리 위치에 저장되지 않는 선형 자료 구조입니다. 랜덤접근은 불가능하며 오로지 순차적 접근만 가능합니다.
요소들은 next, prev라는 포인터로 연결되어 이루어지며 중복을 허용합니다.



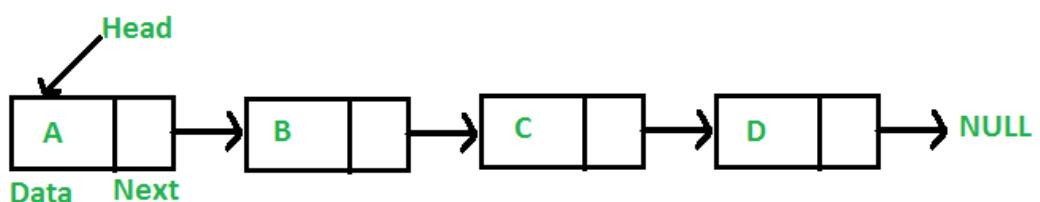
이는 데이터를 감싼 노드를 포인터로 연결해서 공간적인 효율성을 극대화시킨 자료 구조입니다. 삽입과 삭제가 $O(1)$ 이 걸리며 k 번째 요소를 참조한다 했을 때 $O(k)$ 가 걸립니다. 노드는 다음과 같이 이루어져있습니다. (싱글연결리스트 기준) data, 그리고 노드와 노드를 잇게 만드는 next라는 포인터입니다.

```
class Node {
public:
    int data;
    Node* next;
    Node(){
        data = 0;
        next = NULL;
    }
    Node(int data){
        this->data = data;
        this->next = NULL;
    }
};
```

연결리스트는 싱글연결리스트, 이중연결리스트, 원형연결리스트 크게 3가지가 있습니다.

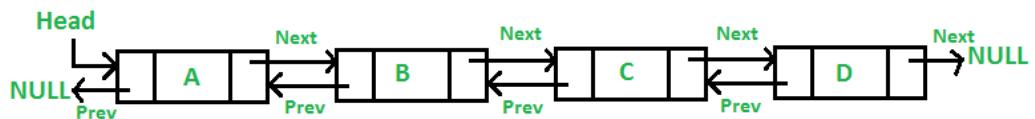
싱글연결리스트

싱글연결리스트(Singly Linked List)는 next 포인터밖에 존재하지 않으며 한 방향으로만 데이터가 연결됩니다.



이중연결리스트

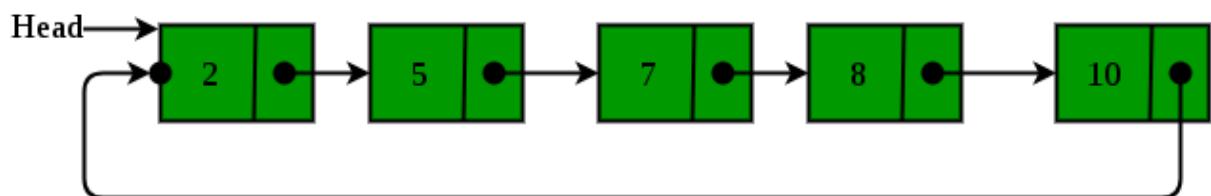
이중연결리스트(Doubly Linked List)는 prev, next 두개의 포인터로 양방향으로 데이터가 연결됩니다.



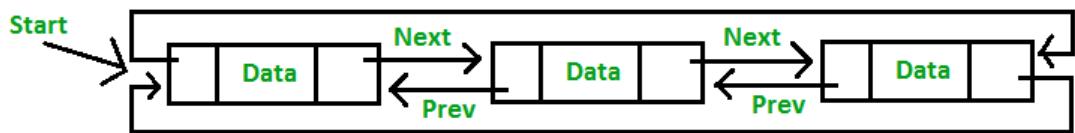
원형연결리스트

원형연결리스트(Circular Linked List)는 마지막 노드와 첫번째 노드가 연결되어 원을 형성합니다. 싱글연결리스트 또는 이중연결리스트로 이루어진 2가지 타입의 원형연결리스트가 있습니다.

원형싱글연결리스트



원형이중연결리스트



C++에서는 `list`로 이중연결리스트를 쉽게 구현할 수 있습니다.

```

#include <bits/stdc++.h>
using namespace std;
list<int> a;
void print(list <int> a){
    for(auto it : a) cout << it << " ";
    cout << '\n';
}
int main(){
    for(int i = 1; i <= 3; i++)a.push_back(i);
    for(int i = 1; i <= 3; i++)a.push_front(i);

    auto it = a.begin(); it++;
    a.insert(it, 1000);
    print(a);

    it = a.begin(); it++;
    a.erase(it);
    print(a);

    a.pop_front();
    a.pop_back();
    print(a);

    cout << a.front() << " : " << a.back() << '\n';
    a.clear();
    return 0;
}
/*
3 1000 2 1 1 2 3
3 2 1 1 2 3
2 1 1 2
2 : 2
*/

```

push_front(value)

리스트의 앞에서 부터 value를 넣습니다.

push_back(value)

뒤에서 부터 value를 넣습니다.

insert(idx , value)

```
iterator insert (const_iterator position, const value_type& val);
```

요소를 몇번째에 삽입합니다.

erase(idx)

```
iterator erase (const_iterator position);
```

리스트의 idx번째 요소를 지웁니다.

pop_front()

첫번째 요소를 지웁니다.

pop_back()

맨 끝 요소를 지웁니다.

front()

맨 앞 요소를 참조합니다.

back()

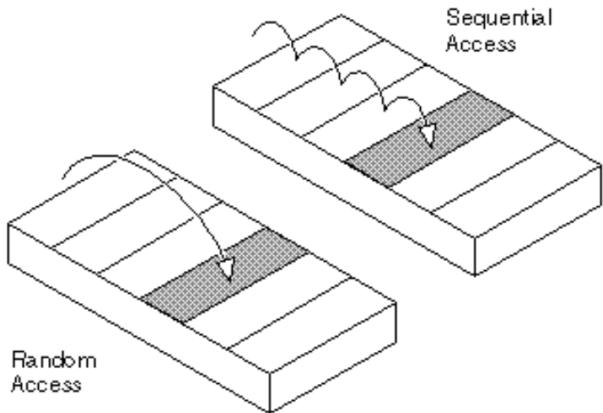
맨 뒤 요소를 참조합니다.

clear()

모든 요소를 지웁니다.

[참고] 랜덤접근과 순차적 접근

직접 접근이라고 하는 랜덤 접근(random access)은 동일한 시간에 배열과 같은 순차적인 데이터가 있을 때 임의의 인덱스에 해당하는 데이터에 접근할 수 있는 기능입니다. 이는 데이터를 저장된 순서대로 검색해야 하는 순차적 접근(sequential access)과는 반대입니다.

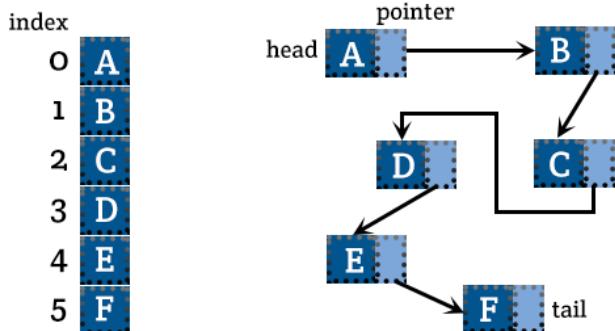


vector, Array와 같은 배열은 랜덤접근이 가능해서 k번째 요소에 접근할 때 $O(1)$ 이 걸리며 연결리스트, 스택, 큐는 순차적 접근만이 가능해서 k번째 요소에 접근할 때 $O(k)$ 이 걸립니다.

[참고] 배열과 연결리스트 비교

면접에서 많이 나오는 질문 중 하나입니다. 배열과 연결리스트를 많이 비교합니다.

Array Linked List



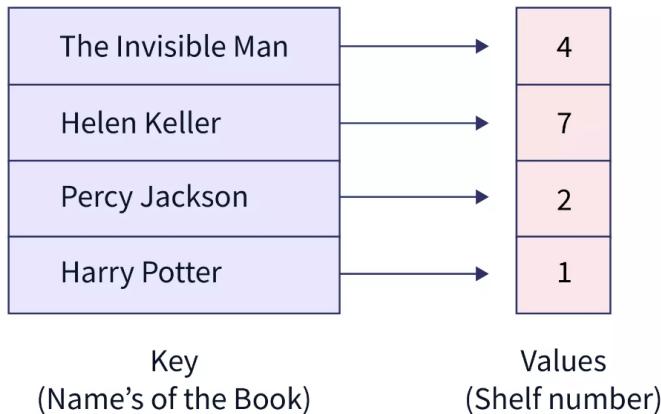
배열은 연속된 메모리 공간에 데이터를 저장하고 연결리스트는 연속되지 않은 메모리 공간에 데이터를 저장합니다.

또한, 배열은 삽입과 삭제에는 $O(n)$ 이 걸리고 참조에는 $O(1)$ 이 걸립니다. 연결리스트는 삽입과 삭제에 $O(1)$ 이 걸리고 참조에는 $O(n)$ 이 걸립니다.

따라서 데이터추가와 삭제를 많이 하는 것은 연결 리스트, 참조를 많이 하는 것은 배열로 하는 것이 좋습니다.

map

map은 고유한 키를 기반으로 키 - 값(key - value) 쌍으로 이루어져 있는 정렬된(삽입할 때마다 자동 정렬된) 연관 컨테이너입니다. 레드 - 블랙트리로 구현됩니다. 레드 - 블랙트리로 구현되어있기 때문에 삽입, 삭제, 수정, 탐색이 $O(\log N)$ 의 시간복잡도를 가집니다.



고유한 키를 갖기 때문에 하나의 키에 중복된 값이 들어갈 수 없으며 자동으로 오름차순 정렬되기 때문에 넣은 순서대로 map을 탐색할 수 있는 것이 아닌 아스키코드순으로 정렬된 값을 기반으로 탐색하게 됩니다. 또한 대괄호 연산자 []로 해당 키로 직접 참조할 수 있습니다.

예를 들어서 "이승철" : 1, "김현영" : 2 이런식으로 string : int 형태로 값을 할당해야 할 때 있죠? 그럴 때 map을 씁니다. 맵의 키와 같은 string이나 int 뿐만 아니라 다양한 값이 들어갈 수 있습니다.

[참고] 레드블랙트리는 균형 이진 검색 트리이며 삽입, 삭제, 수정, 탐색에 $O(\log N)$ 의 시간복잡도가 보장됩니다. 이진 검색 트리는 2주차 개념강의 때 배웁니다.

```
#include <bits/stdc++.h>
using namespace std;
map<string, int> mp;
string a[] = {"주홍철", "양영주", "박종선"};
int main(){
    for(int i = 0; i < 3; i++){
        mp.insert({a[i], i + 1});
        mp[a[i]] = i + 1;
    }
    // mp에 해당 키가 없다면 0으로 초기화되어 할당됨. (int의 경우)
    // 만약 mp에 해당 키가 없는지 확인하고 싶고
    // 초기값으로 0으로 할당되지 않아야 되는 상황이라면
```

```

// find를 써야 함.
cout << mp["kundol"] << '\n';
// 대괄호로 수정 가능.
mp["kundol"] = 4;
cout << mp.size() << '\n';
mp.erase("kundol");
auto it = mp.find("kundol");
if(it == mp.end()){
    cout << "못찾겠네 꾀꼬리\n";
}
mp["kundol"] = 100;

it = mp.find("kundol");
if(it != mp.end()){
    cout << (*it).first << " : " << (*it).second << '\n';
}
for(auto it : mp){
    cout << (it).first << " : " << (it).second << '\n';
}
for(auto it = mp.begin(); it != mp.end(); it++){
    cout << (*it).first << " : " << (*it).second << '\n';
}
mp.clear();

return 0;
}
/*
0
4
못찾겠네 꾀꼬리
kundol : 100
kundol : 100
박종선 : 3
양영주 : 2
주홍철 : 1
kundol : 100
박종선 : 3
양영주 : 2
주홍철 : 1
*/

```

insert({key , value})

map에다 {key, value}를 집어 넣습니다.

[key] = value

대괄호[]를 통해 key에 해당하는 value를 할당합니다.

[key]

대괄호[]를 통해 key를 기반으로 map에 있는 요소를 참조합니다.

size()

map에 있는 요소들의 개수를 반환합니다.

erase(key);

해당 키에 해당하는 요소를 지웁니다.

find(key)

map에서 해당 key를 가진 요소를 찾아 해당 이터레이터를 반환합니다. 만약 못찾을 경우 mp.end() 해당 map의 end() 이터레이터를 반환합니다.

for(auto it : mp)

범위기반 for루프로 map에 있는 요소들을 순회합니다. map을 순회할 때는 key는 first, value는 second로 참조가 가능합니다.

for(auto it = mp.begin(); it != mp.end(); it++)

map에 있는 요소들을 이터레이터로 순회할 수 있습니다.

mp.clear();

map에 있는 요소들을 다 제거합니다.

맵을 쓸 때 주의할 점

다음 코드처럼 map의 경우 해당 인덱스에 참조만 해도 맵의 요소가 생기게 됩니다.

만약 map에 해당 키에 해당하는 요소가 없다면 0 또는 빈문자열로 초기화가 되어 할당됩니다. (int는 0, string이나 char은 빈 문자열로 할당됩니다.)

할당하고 싶지 않아도 대괄호[]로 참조할경우 자동으로 요소가 추가가 되기 때문에 조심해야 합니다.

```
#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

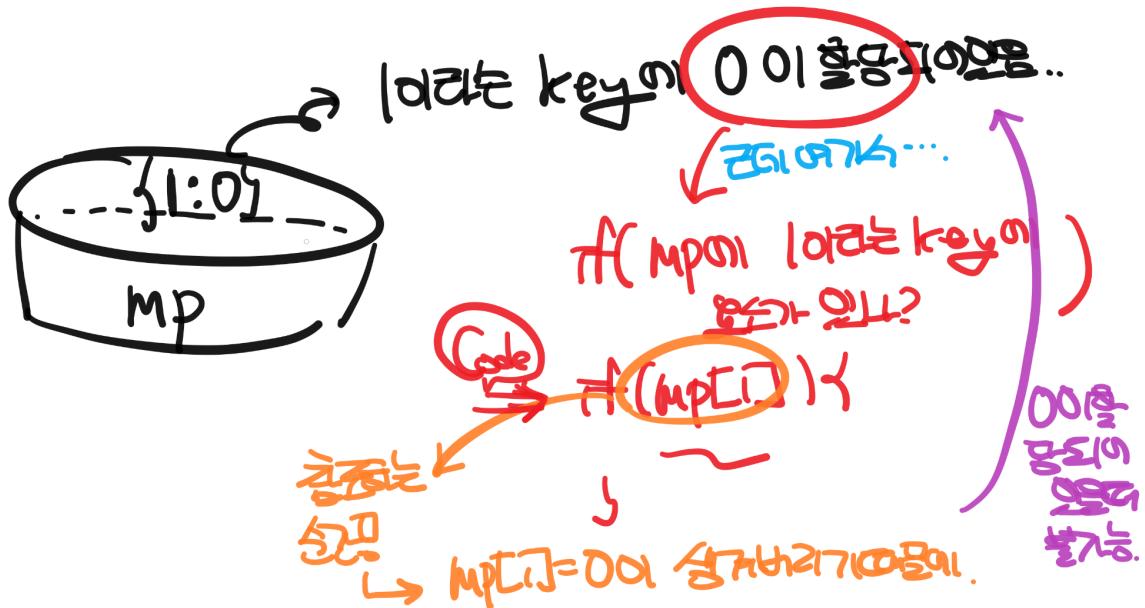
1 0
aaa %
*/
```

맵에 요소가 있는지 없는지를 확인하고 맵에 요소를 할당하는 로직은 다음코드처럼 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
*/
```

다만 앞의 코드는 문제에서 해당 키에 0이 아닌 값이 들어갈 때 활용이 가능합니다.
이미 if문 안에 $mp[1] == 0$ 을 해버린 순간 $mp[1] = 0$ 이 할당되어버리기 때문입니다.



만약 문제에서 0이 들어가는 것을 비교하기 귀찮다면 다음 코드를 기반으로 작성하면 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);
    if(mp.find(1) == mp.end()){
        mp[1] = 2;
    }
    for(auto i : mp) cout << i.first << " " << i.second;

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

unordered_map

앞서 설명한 map은 정렬이 되는 반면 unordered_map은 정렬이 되지 않은 map이며 메서드는 map과 동일합니다.

map과 unordered_map을 비교하면 다음과 같습니다.

- map : 정렬이 됨 / 레드블랙트리기반 / 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_map은 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
*/
```

문제에 정렬이 필요로 하지 않은 문제에는 unordered_map을 써도 될 것 같지만 제출해보면 시간초과가 나기도 합니다. 이는 가장 최악의 경우 $O(N)$ 이기 때문이죠. 되도록 map을 쓰는 것을 권장합니다.

set

셋(set)은 고유한 요소만을 저장하는 컨테이너입니다. 중복을 허용하지 않습니다. map처럼 {key, value}로 집어넣지 않아도 되며 다음 코드처럼 pair나 int형을 집어넣어서 만들 수 있습니다. 중복된 값은 제거되며 map과 같이 자동 정렬됩니다. 메서드는 map과 같습니다.

```
#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(2);
    st2.insert(1);
    st2.insert(2);
    for(auto it : st2){
        cout << it << '\n';
    }
    return 0;
}
/*
1
1
2
*/
```

Q. set과 unique 중 어떤 것을 써야 할까?

앞서서 unique를 통해 중복된 요소를 제거하는 로직을 배웠습니다. set을 통해서도 제거할 수 있죠. 그렇다면 어떤 것을 사용하는게 더 좋을까요? set을 사용해도 되고, unique와 erase를 사용해도 됩니다.

예를 들어 vector에다가 담아야 하는 로직이 있다고 해볼게요.

그러면 만약 set을 사용할경우

1. 중복된 배열 vector 생성됨

2.set 사용해서 중복제거

3.다시 새로운 vector를 만들어 요소를 집어넣음.

이를 통해 새로운 vector와 set 2개의 자료구조가 "더" 만들어지게 되는 것을 알 수 있습니다.

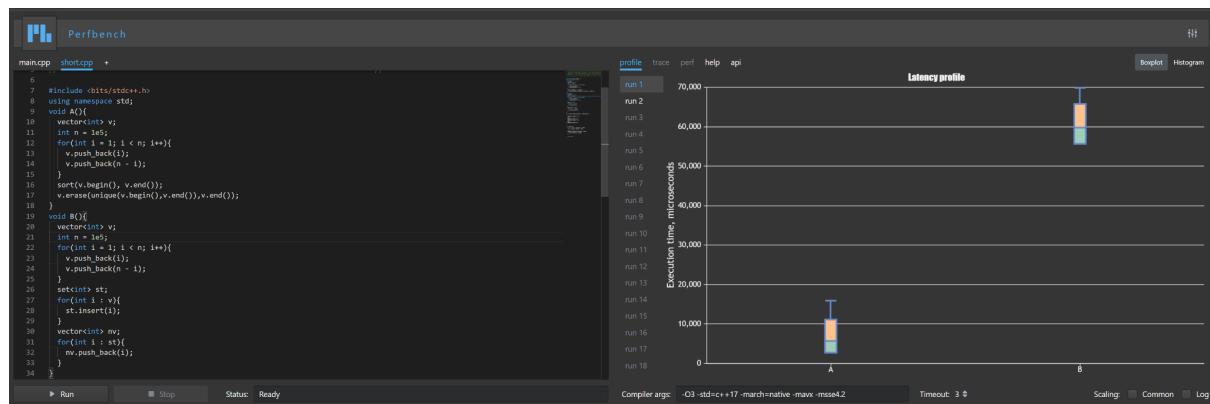
하지만 unique()와 erase()를 사용한다면 그냥 해당 중복된 배열 vector를 기반으로 재탕해서 사용해도 되는 장점이 있죠.

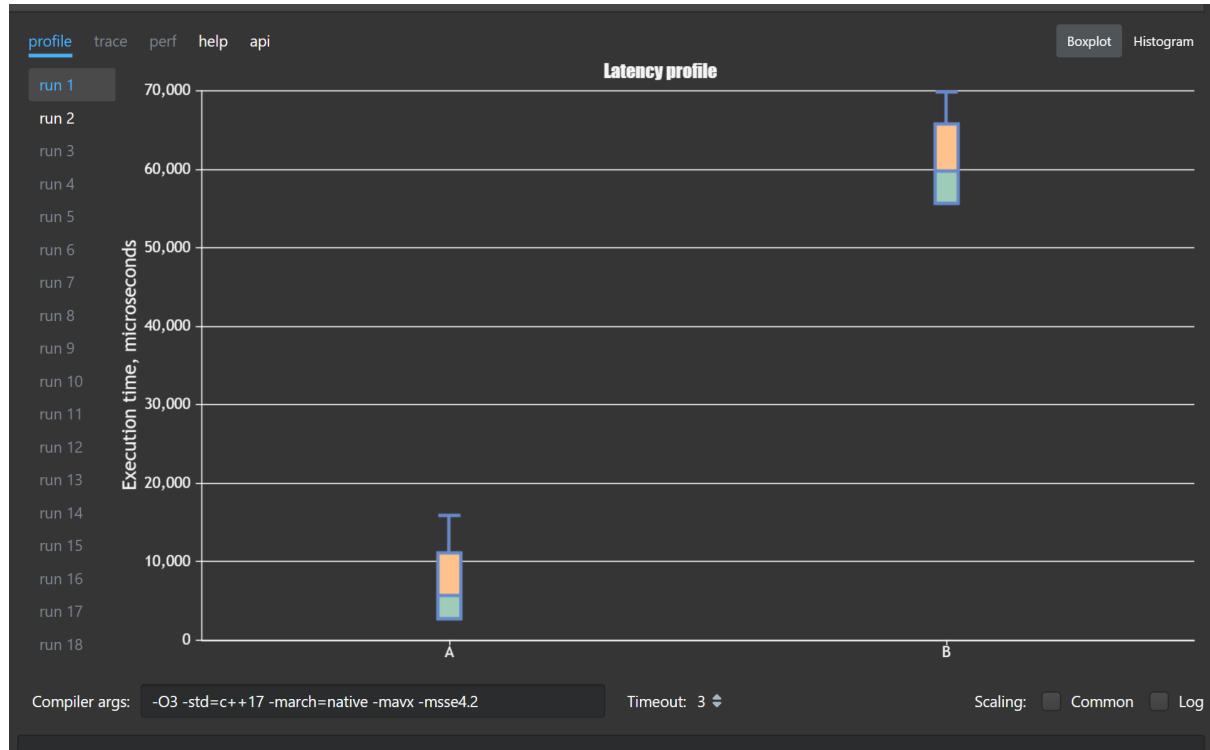
이 두개의 로직을 C++ 벤치마크로 테스팅을 해봤습니다.

<https://perfbench.com/>

(앞의 링크로 가서 테스팅을 할 수 있습니다.)

다음 그림처럼 set보다는 unique()와 erase() 사용하는 코드가 더 좋은 것을 알 수 있습니다.





다만 이는 필요한 로직에 따라 달라질 수 있으니 참고만 해주세요.

코드

```
#include <bits/stdc++.h>
using namespace std;
void A(){
    vector<int> v;
    int n = 1e5;
    for(int i = 1; i < n; i++){
        v.push_back(i);
        v.push_back(n - i);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
}
void B(){
    vector<int> v;
    int n = 1e5;
    for(int i = 1; i < n; i++){
        v.push_back(i);
        v.push_back(n - i);
    }
    set<int> st;
    for(int i : v){
        st.insert(i);
    }
    vector<int> nv;
    for(int i : st){
```

```

        nv.push_back(i);
    }
}

void test_latency(size_t iteration) {

    PROFILE_START("A");
    A();
    PROFILE_STOP("A");
    PROFILE_START("B");
    B();
    PROFILE_STOP("B");
}

int main() {
    const size_t warmups = 1000;
    const size_t tests = 100;

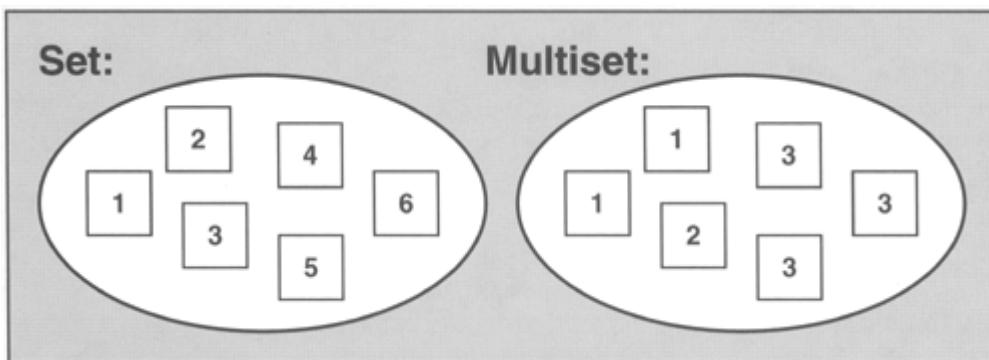
    PROFILE_RUN_ALL(warmups, tests,
        test_latency(__loop));
}

return 0;
}

```

multiset

multiset은 중복되는 요소도 집어넣을 수 있는 자료구조입니다. 다음 그림은 set과 multiset을 비교한 그림입니다.



key, value 형태로 집어넣을 필요도 없고 넣으면 자동적으로 정렬됩니다.

메서드는 map과 같습니다.

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

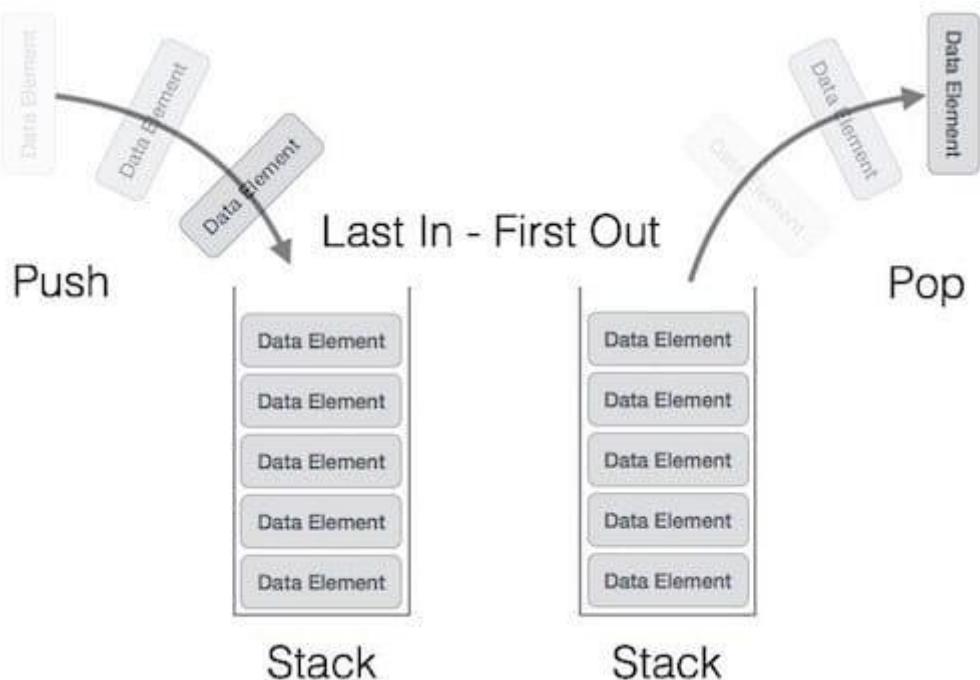
```

using namespace std;
multiset<int> s;
int main() {
    for(int i = 5; i >= 1; i--){
        s.insert(i);
        s.insert(i);
    }
    for(int it : s) cout << it << " ";
    cout << '\n';
    return 0;
}
/*
1 1 2 2 3 3 4 4 5 5
*/

```

stack

스택은 가장 마지막으로 들어간 데이터가 가장 첫 번째로 나오는 성질인 후입선출(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. 탐색에 O(n)이 걸리는 이유는 n번째 요소를 찾는다고 가정하면 계속해서 앞에 있는 요소를 끄집어내는 과정을 n 번 반복해야 찾을 수 있기 때문입니다.



LIFO, 가장 마지막으로 들어간 데이터가 가장 첫번째로 나오는 구조를 지녔습니다. 다음과 같이 엄준식 화이팅이 순차적으로 들어갔는데 나오는 것은 반대로 나오는 것을 알 수 있습니다.

```
#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();
    }
}
/*
TING
EYHAWKSUJUN
*/
```

스택은 주로 문자열 폭발, 아름다운 팔호만들기, 짹찾기 키워드를 기반으로 이루어진 문제에서 쓰일 수 있습니다.

또한, “교차하지 않고”라는 문장이 나오면 스택을 사용해야 하는 것은 아닐까? 염두해야 합니다.

push(value)

해당 value를 스택에 추가합니다.

pop()

가장 마지막에 추가한 요소를 지웁니다. (가장 위에 있는 요소를 지운다고도 합니다.)

top()

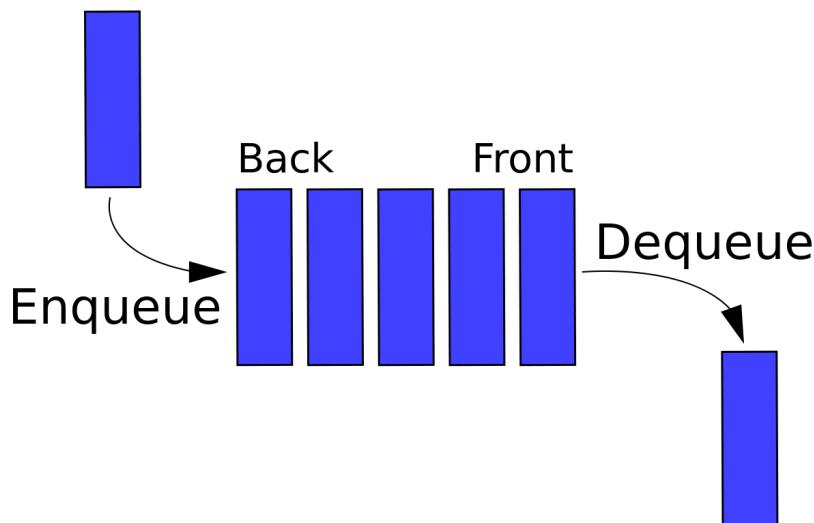
가장 마지막에 있는 요소를 참조합니다. (가장 위에 있다고도 합니다.)

size()

스택의 크기입니다.

queue

큐(queue)는 먼저 집어넣은 데이터가 먼저 나오는 성질인 선입선출(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.



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

push(value)

value를 큐에 추가합니다.

pop()

가장 앞에 있는 요소를 제거합니다.

size()

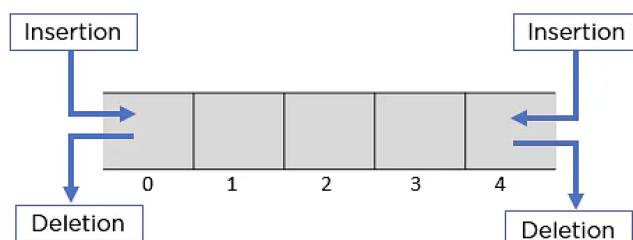
큐의 크기입니다.

front()

가장 앞에 있는 요소를 참조합니다.

deque

앞서 설명한 queue는 앞에서만 끄집어낼 수 있다면 이것은 앞뒤로 삽입, 삭제, 참조가 가능한 자료구조입니다.



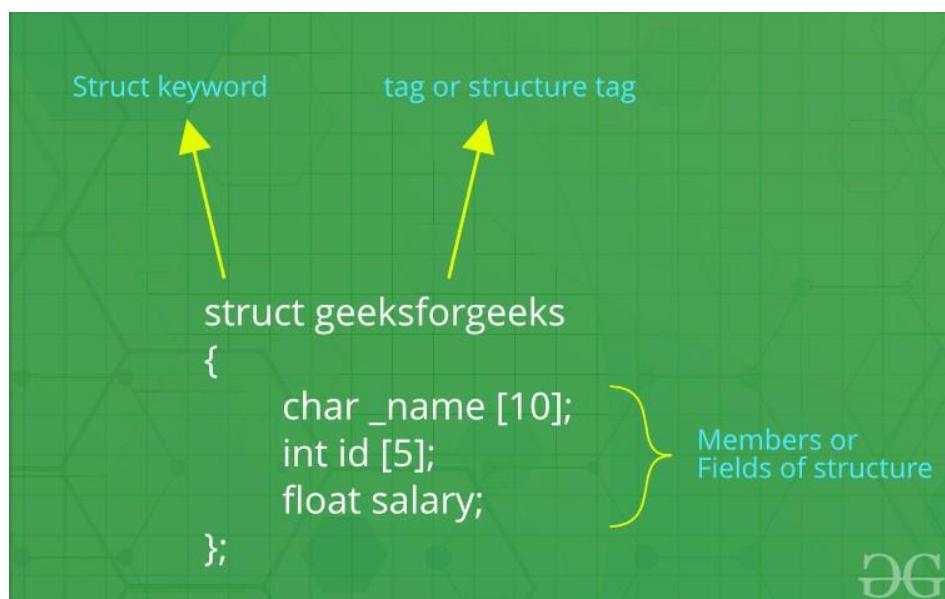
```

#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

구조체라 불리는 struct는 C++에서 제공하는 자료구조가 아닌 개발자의 커스텀한 자료구조를 의미합니다. 커스텀하게 정렬을 추가하고 싶거나 문제에서 여러개의 변수 예를 들어 a, b, c, d, e 가 들어간 자료구조가 필요하다면 struct를 사용해야 합니다.



예를 들어 int 타입의 2개의 멤버변수, double 타입의 3개의 멤버 변수가 필요하다고 해보죠.

[참고] 멤버변수란 클래스 또는 구조체 내부의 변수이자 메소드 밖에 있는 변수를 뜻합니다.

랄로 구조체 정의하기

다음 코드를 보면 Ralo라는 int 타입의 2개의 멤버변수, double 타입의 3개의 멤버 변수를 가진 간단한 구조체를 형성한 것을 볼 수 있습니다. 정해지지 않고 커스텀하게 만든 것을 볼 수 있으며 vector에도 집어넣은 모습을 볼 수 있습니다. 또한 만약 값을 집어넣지 않은 경우 0으로 초기화되는 것을 볼 수 있습니다. char 또는 string으로 선언한 경우 값을 집어넣지 않게 되면 빈문자열이 들어가게 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
struct Ralo{
    int a, b;
    double c, d, e;
};
void print(Ralo ralo){
    cout << ralo.a << " " << ralo.b << " " << ralo.c << " " << ralo.d << " " <<
ralo.e << '\n';
}
int main(){
    Ralo ralo = {1, 1, 1, 1, 1};
    print(ralo);
    vector<Ralo> ret;
    ret.push_back({1, 2, 3, 4, 5});
    ret.push_back({1, 2, 3, 4, 6});
    ret.push_back({});
    ret.push_back({1, 3});
    for(Ralo ralo : ret){
        print(ralo);
    }
    return 0;
}

/*
1 1 1 1 1
1 2 3 4 5
1 2 3 4 6
0 0 0 0 0
1 3 0 0 0
*/
```

자, 그러면 이 간단한 구조체를 집어넣은 vector를 정렬한다면 어떻게 해야할까요? a를 1순위로, b를 2순위로 오름차순으로 정렬하고 싶다면 어떻게 해야할까요?

또한 지금은 아무것도 집어넣지 않을 경우 0 또는 빈 문자열이 들어가게 되는데 초기값을 설정하고 싶다면 어떻게 해야할까요?

이렇게 요구사항이 많아지게 되면 조금은 복잡하게 구조체를 구축해야 합니다.

Point 구조체 정의하기

다음 코드는 조금은 복잡한 Point라는 구조체를 정의했습니다. 구조체를 기반으로 정렬하는 연산, 그리고 초기값 설정이 필요하다면 다음과 같은 형태로 구조체를 정의해야 합니다.

```
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;
    }
};
```

코드 하나하나씩 살펴보겠습니다.

구조체의 멤버변수 y, x를 정의합니다.

```
int y, x;
```

y, x를 받아 멤버변수를 생성한다라는 의미입니다.

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

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

```

// Java bean for Person
public class Person {
    // Private variables
    private String firstName;
    private String lastName;
    // Constructor
    public Person(String firstName, String lastName) {
        setFirstName(firstName);
        setLastName(lastName);
    }
    // Getter and setter for variable

```

[JAVA의 constructor 매직메서드]

```

> class Vehicle {
    constructor(make, model, color) {
        this.make = make;
        this.model = model;
        this.color = color;
    }
}

```

[자바스크립트의 constructor 매직메서드]

초기값 설정 부분입니다.

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

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

연산자(operator) 오버로딩입니다. 이는 말 그대로 연산자를 오버로딩(하위 클래스에서 재정의)하는 것이죠. 연산자는 <, >, 등이 있고 이를 오버로딩한다는 것입니다.

```

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

```

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

이 비교는 단순하게 if문으로 비교할 때도 정의해야하지만 정렬할 때도 정의해주어야 합니다. 정렬이란 요소들을 비교해가며 정렬하는 것이기 때문입니다.

앞의 코드를 기반으로 정렬하면 x가 1순위, y가 2순위로 "오름차순"으로 정렬됩니다.

다음 코드를 좀 더 보죠.

만약 `{1, 2}`, `{2, 3}`이 만나면 어떻게 될까요? `x`는 서로 같지 않기 때문에 바로 밑의 `return`문으로 내려가게 되고 `x`를 비교해서 `{1, 2}`가 더 작기 때문에 `{1, 2}`와 `{2, 3}`을 비교했을 때 `{1, 2}`가 더 작다는 결론을 내리게 됩니다. 만약 `x`가 같다면 이와 같은 논리로 `y`를 기반으로 비교해서 해당 요소가 더 작다는 결론을 내리겠죠?

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

단순하게 `int` 타입의 변수 2개 `int a = 1, int b = 2`의 크기 비교하는 것처럼 구조체끼리도 크고 작음을 비교할 때는 이렇게 어떠한 멤버변수를 기준으로 할 것인지 등을 정해주어야 합니다.

Q. 왜 오버라이딩이 아니라 오버로딩일까?

왜냐하면 `operator +`, `-`, `*`, `/` 등의 의미를 변경하지 않으면 그저 오퍼레이터의 대상이 바뀌는 것 뿐이기 때문입니다. 교안에서는 `<`라는 연산자를 오버로딩하는데 이는 그냥 `int`, `float`의 기본타입이 아니라 좀 더 복잡한 어떠한 객체 `struct`를 기반으로 확장하는 것 뿐이기 때문이죠.

구조체 기반 `sort`를 사용할 때 주의 할 점

앞의 코드를 보면 `<` 오퍼레이터를 기준으로 `struct`를 구축된 것을 볼 수 있습니다. 이를 기반으로 이러한 `struct`를 구현한 변수들을 `sort`를 하는 상황이 생기겠죠? 이 때문에 저 `struct`의 오퍼레이터는 `>`가 아니라 `<` 오퍼레이터를 기준으로 설정되어야 합니다.
`sort()`함수 자체가 `<` 오퍼레이터를 기준으로 정렬하기 때문입니다.

참고로 `struct` 내의 오퍼레이터 오버로딩 하지 않고 `compare` 함수를 만들어서 할 수도 있습니다. 이 때 `sort`를 사용한다면 동일하게 `<` 오퍼레이터를 기준으로 `compare`함수를 정의해야 합니다.

`string` 으로 이루어진 배열을 정렬한 코드

```

#include <bits/stdc++.h>
using namespace std;
bool compare(string a, string b){
    if(a.size() == b.size()) return a < b;
    return a.size() < b.size();
}
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    string a[3] = {"111", "222", "33"};
    sort(a, a + 3, compare);
    for(string b : a) cout << b << " ";
    return 0;
}
// 33 111 222

```

숫자를 담고 있지만 string으로 설정된 변수들을 정렬할 때 앞의 코드와 같은 compare()함수를 써야 합니다. string은 서로 비교할 때 왼쪽에서부터 아스키코드순으로 비교합니다.

예를 들어 “111”과 “222”를 비교한다면 왼쪽에서부터 아스키코드순으로 비교하기 때문에 “111”이 “222”보다 더 작은 숫자로 인식합니다. 그러나 “111”과 “33”을 비교한다면 “111”

“33”

이렇게 왼쪽에서부터 아스키코드순으로 비교하기 때문에 “33”이 “111”보다 더 크다고 인식해버립니다. 때문에 항상 숫자로 이루어진 문자열을 비교할 때는 사이즈확인 로직을 넣는게 중요합니다.

Ralo struct를 compare 함수를 통해 정렬한 코드

```

#include <bits/stdc++.h>
using namespace std;
struct Ralo{
    int a, b;
};

bool compare(Ralo A, Ralo B){
    if(A.a == B.a) return A.b < B.b;
    return A.a < B.a;
}
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    Ralo a[3] = {{1, 2}, {1, 3}, {0, 4}};
    sort(a, a + 3, compare);
    for(Ralo A : a) cout << A.a << " : " << A.b << "\n";
}

```

```

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

```

1순위로 Ralo의 a를 오름차순으로, 2순위로 Ralo의 b를 오름차순으로 정렬되는 코드입니다.

3개의 멤버변수 정렬하기

자, 이제 3개의 멤버변수인 y, x, z가 필요하다고 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? 참고로 초기값은 -1이 아니라 다른 수를 집어넣어도 됩니다. 문제에 맞춰 설정해주면 됩니다.

vector에다 struct 넣고 정렬하기

다음은 vector에 Point라는 struct를 넣어서 정렬하는 모습입니다.

```

#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(){

```

```

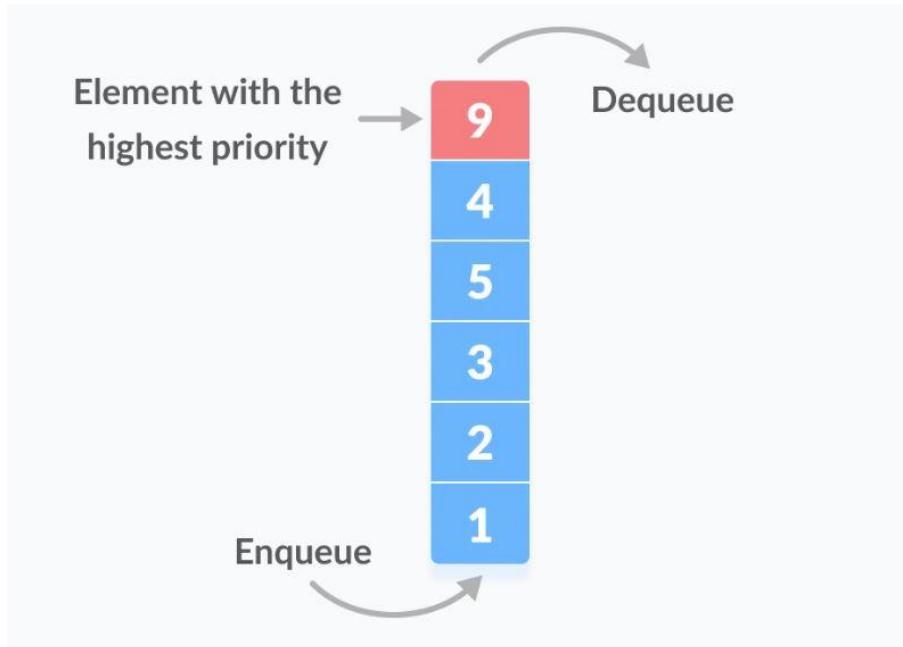
for(int i = 10; i >= 1; i--){
    v.push_back({i, 10 - i});
}
sort(v.begin(), v.end(), cmp);
for(auto it : v) cout << it.y << " : " << it.x << "\n";
return 0;
}
/*
1 : 9
2 : 8
3 : 7
4 : 6
5 : 5
6 : 4
7 : 3
8 : 2
9 : 1
10 : 0
*/

```

[참고] 커스텀한 자료구조를 만들 때 보통 class와 struct를 쓰지만 코딩테스트에서는 struct만 알아도 충분합니다. 둘의 차이는 struct의 멤버변수는 기본적으로 public이며 상속이 안되며 class의 멤버변수는 기본적으로 private이며 상속이 됩니다.

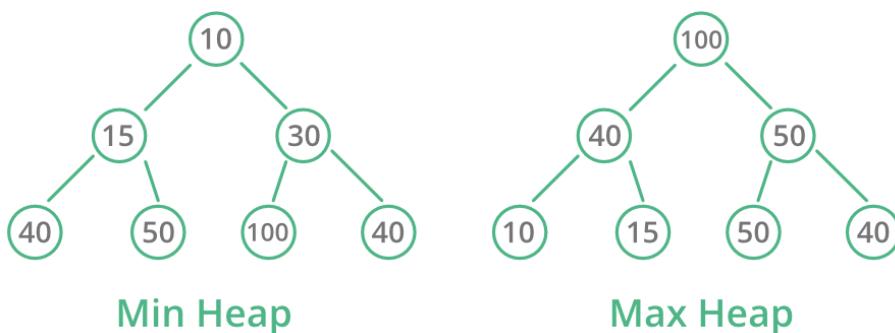
priority queue

우선순위 큐(priority queue)는 각 요소에 어떠한 우선순위가 추가로 부여되어있는 컨테이너를 말합니다.



우선순위 큐에서 우선순위가 높은 요소는 우선순위가 낮은 요소보다 먼저 제공됩니다. 일부 구현에서 두 요소의 우선 순위가 같으면 대기열에 포함된 순서에 따라 제공됩니다. 다른 구현에서 동일한 우선 순위를 가진 요소의 순서는 정의되지 않은 상태로 유지됩니다.

[참고] 힙은 완전이진트리로 최소힙 또는 최대힙이 있으며 삽입, 삭제, 탐색, 수정에 대해 $O(\log N)$ 의 시간복잡도를 갖습니다. 최대 힙은 루트 노드에 최대값이 있고, 최소 힙은 루트 노드에 최소값이 있는 힙을 말합니다.



int형 우선순위큐

단순하게 int형 우선순위큐는 다음 코드 처럼 greater<타입> 을 써서 오름차순, less<타입>을 써서 내림차순으로 바꿀 수 있습니다. 기본값은 내림차순이라 단순하게 priority_queue<타입>을 쓰면 해당 타입에 대한 내림차순으로 설정됩니다.

```
#include <bits/stdc++.h>
using namespace std;
priority_queue<int, vector<int>, greater<int> > pq; //오름차순
priority_queue<int> pq2; // 내림차순
priority_queue<int, vector<int>, less<int> > pq3; // 내림차순
int main(){
    for(int i = 5; i >= 1; i--){
        pq.push(i); pq2.push(i); pq3.push(i);
    }
    while(pq.size()){
        cout << pq.top() << " : " << pq2.top() << " : " << pq3.top() << '\n';
        pq.pop(); pq2.pop(); pq3.pop();
    }
    return 0;
}
/*
1 : 5 : 5
2 : 4 : 4
3 : 3 : 3
4 : 2 : 2
5 : 1 : 1
*/
```

메서드는 stack과 같습니다. size(), top(), pop(), push()가 있습니다.

구조체를 담은 우선순위큐

int 뿐만 아니라 구조체(struct) 등 다른 자료구조를 넣어서 할 수 있습니다. 구조체를 담은 우선순위를 볼까요?

```
#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
*/

```

앞의 코드를 보면 분명 < 연산자에 x > a.x를 했기 때문에 분명 내림차순으로 정렬되어 6이 출력이 되어야 하는데 1이 출력되죠?

이는 우선순위큐에 커스텀 정렬을 넣을 때 반대로 넣어야 하는 특징 때문입니다.

반대로 해볼까요?

```

#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.

물론 다음코드 처럼도 가능합니다.

```
#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
*/
```

자료구조 시간복잡도 정리

다음은 지금까지 설명했던 자료구조의 최악의 시간 복잡도입니다.

자료구조	참조	탐색	삽입	삭제
배열	$O(1)$	$O(n)$	$O(n)$	$O(n)$
스택	$O(n)$	$O(n)$	$O(1)$	$O(1)$
큐	$O(n)$	$O(n)$	$O(1)$	$O(1)$
연결리스트	$O(n)$	$O(n)$	$O(1)$	$O(1)$

맵	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
---	-------------	-------------	-------------	-------------

스택과 큐 같은 경우 가장 앞에 있는 요소를 참조한다고 생각하면 $O(1)$ 이지만 중간에 있는 요소를 참조한다고 했을 때 랜덤접근이 아닌 순차접근만 되기 때문에 $O(n)$ 의 시간이 걸립니다.

1.9 값에 의한 호출과 참조에 의한 호출

함수에 값을 전달 할때 값에 의한 호출(call by value)과 참조에 의한 호출(call by reference)과 두가지가 있습니다.

매개변수

먼저 그전에 매개변수에 대해 알아봅시다. 매개변수(parameter)는 함수에 전달되는 값이자 함수 괄호 안에 선언된 값을 말합니다. 쉼표로 각 매개변수를 다른 매개변수와 구분합니다. 다음은 $n1, n2$ 가 매개변수가 됩니다.

```
int add( int n1, int n2 )
{
    return (n1+n2);
}
```

Parameters

값에 의한 호출

값에 의한 호출(call by value)은 매개변수로 전달되는 변수를 모두 함수 내부에서 복사해서 함수를 실행하는 방법입니다. 함수 내부에서 전체 복사가 일어나고 실제 변수와는 다른 주소에 할당되기 때문에 실제 변수와 매개변수로 전달된 변수는 다른 주소값을 가집니다. 즉, 함수 내부에서 이 복사본이 변경되더라도 원본값은 변경되지 않습니다.

다음 코드를 보면 a 를 넘겨서 a 에 $+= 10$ 을 했습니다. 함수 내부에서는 11이 찍힐지언정, 원본변수인 $main()$ 함수의 a 는 수정되지 않습니다.

```

#include <bits/stdc++.h>
using namespace std;
int add(int a, int b){
    a += 10;
    cout << a << '\n';
    return a + b;
}
int main(){
    int a = 1;
    int b = 2;
    int sum = add(a, b);
    cout << a << '\n';
    return 0;
}
/*
11
1
*/

```

vector를 값에 의한 호출을 하면 다음과 같이 됩니다. vector를 넘겨서 수정해도 아무런 반영이 되지 않습니다.

```

#include<bits/stdc++.h>
using namespace std;
vector<int> v(3, 0);
void go(vector<int> v){
    v[1] = 100;
}
int main(){
    go(v);
    for(int i : v) cout << i << '\n';
}
/*
0
0
0
*/

```

참조에 의한 호출

참조에 의한 호출(call by reference)은 “변수의 주소”를 매개변수로 함수에 전달하는 방법입니다. 함수 내부에서 해당 매개변수를 변경하게 되면 실제 원본변수에도 반영이 됩니다.

다음 코드는 매개변수로 &a를 넘겼고 해당 함수에서 a를 수정했고 이 부분이 원본 변수에도 반영된 모습입니다.

```
#include <bits/stdc++.h>
using namespace std;
int add(int &a, int b){
    a += 10;
    cout << a << '\n';
    return a + b;
}
int main(){
    int a = 1;
    int b = 2;
    int sum = add(a, b);
    cout << a << '\n';
    return 0;
}
/*
11
11
*/
```

vector를 참조에 의한 호출을 하면 다음과 같이 됩니다. 넘겨서 수정했더니 원본 vector에 반영이 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
vector<int> v(3, 0);
void go(vector<int> &v){
    v[1] = 100;
}
int main(){
    go(v);
    for(int i : v) cout << i << '\n';
}
/*
0
100
0
*/
```

참조에 의한 호출로 넘겨야 할 때

primitive한 타입, 예를 들어 double, int 등은 “값에 의한 호출”로 넘기는게 좋습니다. 복사가 일어나지만 간단하기 때문에 복사에 대한 코스트가 크지 않습니다. 오히려 함수 내부에서 직접적으로 참조할 수 있기 때문에 더 빠릅니다.

하지만 reference한 타입 중 복잡한 struct나 많은 요소를 가진 배열을 배열이 차지하는 메모리가 많을 때는 참조로써 매개변수를 넘기는 게 좋습니다. 왜냐하면 참조없이 전달하게 될 경우 전체 복사를 해야 하는데 이 때 드는 코스트가 더 크기 때문입니다.

여기서 코스트란 해당 복사에 드는 CPU 시간, 메모리를 말합니다.

[참고] primitive와 reference 타입

primitive한 타입은 다음을 말합니다.

- byte, short, int, long, float, double, boolean, char

reference 타입은 인스턴스화 가능한 모든 클래스 및 배열을 말합니다.

- String, struct, class, int[], string[] 등

성능에 의한 시간초과 예

예를 들어 다음은 문제 : 암기왕입니다.

<https://www.acmicpc.net/problem/2776>

문제를 풀 필요는 없습니다. 이 문제의 경우 어떠한 배열에 100만개의 요소를 담아야 합니다. 많이 담아야 하죠?

이 문제의 해설코드는 다음과 같습니다.

1번 : 맞은 코드

<http://boj.kr/6a21a43f74594cf08963ad0de77ec89a>

2번 : 시간초과가 나는 코드

<http://boj.kr/e2e36a91783849bd8b0d27dc777c6479>

해당 코드의 차이점은 뭘까요? &말고는 다른 점이 없습니다. 이 문제는 100만개 정도의 배열을 만들어서 함수에 넘겨야 하는데 이렇게 많은 배열을 복사할 경우 참조에 의한 호출을 쓰는게 좋습니다.

하지만 참조에 의한 호출이던, 값에 의한 호출이던 그 이후에 로직 자체가 배열을 복사하는 로직이라면 굳이 참조에 의한 호출을 하는 것과 값에 의한 호출을 하는 것의 차이는 사라집니다.

ex)

```
void copy_v(vector<int>& a) {
    auto copy = a;
}
```

1.10 배열 수정하기

Array의 요소 수정하기

Array의 요소를 수정할 때는 다음처럼 크기를 정하지 않은 int a[], 또는 배열의 크기인 int a[3], 배열의 포인터인 int * a를 넘겨서 수정할 수 있습니다.

```
#include<bits/stdc++.h>
using namespace std;
int a[3] = {1, 2, 3};
void go(int a[]){
    a[2] = 100;
}
void go2(int a[3]){
    a[2] = 1000;
}

void go3(int *a){
    a[2] = 10000;
}

int main(){
    go(a); cout << a[2] << '\n';
    go2(a); cout << a[2] << '\n';
    go3(a); cout << a[2] << '\n';
}
/*
100
1000
10000
*/
```

2차원 배열 수정하기

vector

2차원 vector의 경우 다음과 같이 수정하면 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
vector<vector<int>> v;
vector<vector<int>> v2(10, vector<int>(10, 0));
vector<int> v3[10];
void go(vector<vector<int>> &v){
    v[0][0] = 100;
}
void go2(vector<vector<int>> &v){
    v[0][0] = 100;
}
void go3(vector<int> v[10]){
    v[0][0] = 100;
}
int main(){
    vector<int> temp;
    temp.push_back(0);
    v.push_back(temp);

    v3[0].push_back(0);

    go(v); go2(v2); go3(v3);
    cout << v[0][0] << " : " << v2[0][0] << " : " << v3[0][0] << '\n';
    return 0;
}
//100 : 100 : 100
```

array

다음 코드와 같이 수정합니다.

```
#include<bits/stdc++.h>
using namespace std;
int a[3][3] = {{1, 2, 3}, {1, 2, 3}, {1, 2, 3}};
void go(int a[][3]){
    a[2][2] = 100;
}
void go2(int a[3][3]){
    a[2][2] = 1000;
}
```

```

int main(){
    go(a); cout << a[2][2] << '\n';
    go2(a); cout << a[2][2] << '\n';
}
/*
100
1000
*/

```

1.11 재귀함수와 수학

재귀함수와 코딩테스트에 주로 나오는 수학적인 개념들을 배워봅니다.

재귀함수

0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

재귀함수란 아래의 3가지 특징을 가진 함수를 말합니다.

- 재귀함수(Recursion)는 정의 단계에서 자신을 재참조하는 함수
- 전달되는 상태인 매개변수가 달라질 뿐 똑같은 일을 하는 함수
- 큰 문제를 작은 부분문제로 나눠서 풀 때 사용합니다.

예를 들어 팩토리얼(factorial)은 1부터 해당 항까지 곱하는 함수입니다. 이를 재귀함수로 구현하면 다음과 같습니다.

Factorial of a Number

$$n! = n * (n-1) * (n-2) * \dots * 1$$

```

#include <bits/stdc++.h>
using namespace std;
int fact_rec(int n){
    if(n == 1 || n == 0) return 1;
    return n * fact_rec(n - 1);
}
int fact_for(int n){
    int ret = 1;

```

```

for(int i = 1; i <= n; i++){
    ret *= i;
}
return ret;
}
int n = 5;
int main() {
    cout << fact_for(n) << '\n';
    cout << fact_rec(n) << '\n';
    return 0;
} // 120 120

```

예를 들어 피보나치의 합은 다음과 같이 구축할 수 있습니다.

피보나치 수 F_n 는 다음과 같은 초기값 및 점화식으로 정의되는 수열이다.

$$F_1 = F_2 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \in \{3, 4, \dots\})$$

0번째 항부터 시작할 경우 다음과 같이 정의된다.

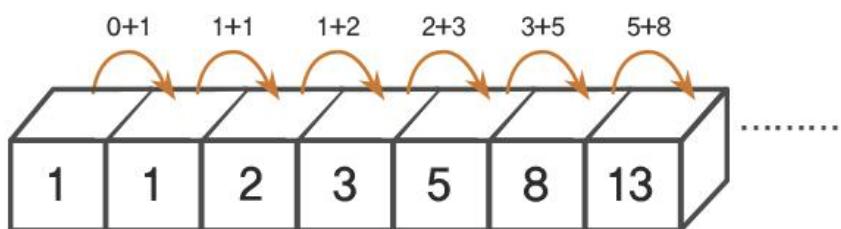
$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \in \{2, 3, 4, \dots\})$$

피보나치 수의 처음 몇 항은 (0번째 항부터 시작할 경우) 다음과 같다.

피보나치의 합을 구하면 다음과 같이 1, 1, 2, 3, 5, 8, 13 ... 이 나오게 됩니다.



```

#include <bits/stdc++.h>
using namespace std;
int fibo(int n){
    cout << "fibo : " << n << '\n';
    if(n == 0 || n == 1) return n;
    return fibo(n - 1) + fibo(n - 2);
}
int n = 5;
int main() {

```

```

    cout << fibo(n) << '\n';
    return 0;
}
/*
fibo : 5
fibo : 4
fibo : 3
fibo : 2
fibo : 1
fibo : 0
fibo : 1
fibo : 2
fibo : 1
fibo : 0
fibo : 3
fibo : 2
fibo : 1
fibo : 0
fibo : 1
5
*/

```

재귀함수를 사용할 때는 다음과 같은 주의사항을 지켜주어야 합니다.

- 반드시 기저사례를 써야 한다. (종료조건) fact를 보면 다음과 같은 기저사례가 있습니다.

```
if(n == 1 || n == 0) return 1;
```

- 사이클이 있다면 쓰면 안된다. ex) f(a)가 f(b)를 호출한 뒤 f(b)가 다시 f(a)를 호출하는 것
- 반복문으로 될 거 같으면 반복문으로. (함수호출에 대한 코스트가 듭니다.)

순열과 조합

축구선수 12명이 서로(2명씩) 인사하면 몇 가지 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

0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

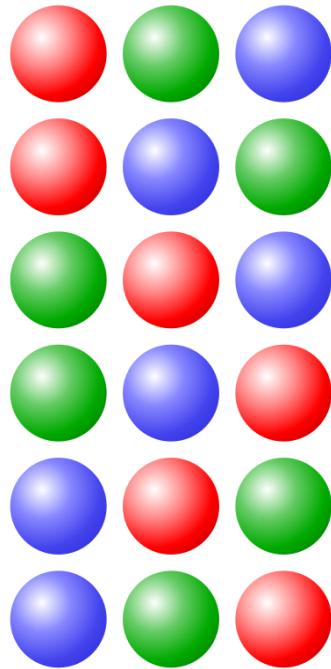
먼저 순열, permutation이란 순서가 정해진 임의의 집합을 다른 순서로 섞는 연산을 말합니다. 그러니까 1, 2, 3 이렇게 있을 때 1, 3, 2 이런식으로 다른 순서로 섞는 연산을 순열이라고 합니다. 그리고 n 개의 집합 중 n 개를 고르는 순열의 개수는 n!이라는 특징을 가지고 있습니다.

예를 들어 3개의 자연수(1, 2, 3)를 이용해 만들 수 있는 3자리 자연수는 몇개 일까요? 123, 132, 213, 231, 312, 321 이렇게 6개입니다. 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 관계있이” 뽑는 경우의 수는 어떻게 될까요? 바로 6개고 다음과 같은 그림입니다.



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

그렇다면 이를 코드로 어떻게 구현할 수 있을까요?

next_permutation과 prev_permutation

첫번째 방법은 `next_permutation`과 `prev_permutation`을 이용하는 방법입니다. `next_permutation`은 “오름차순의 배열”을 기반으로 순열을 만들 수 있으며 `prev_permutation`은 그와 반대인 “내림차순의 배열”을 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 <bits/stdc++.h>
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 vector의 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 번째의 순서만을 고치게 됩니다. [v.begin(),
v.begin() + 2)이라는 범위를 만든 거니까요.

next_permutation() 사용시 주의할 점

next_permutation()과 prev_permutation() 두 가지 함수 중 보통은 next_permutation()을 쓰는데 그 때 배열을 “오름차순”으로 정렬을 해서 쓰는게 중요합니다.

다음 코드와 같이 정렬되어있지 않은 a는 순열의 모든 경우의 수가 나오지 않고 정렬된 b는 순열의 모든 경우의 수가 나오는 것을 볼 수 있습니다. 또한 앞선 코드와는 달리 vector가 아닌 정적 배열 Array를 기반으로 했습니다.

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

    cout << "정렬되어있지 않은 배열기반\n";
    int a[] = {1, 3, 2};
    do{
        for(int i : a) cout << i << " ";
        cout << '\n';
    }while(next_permutation(a, a + 3));
    cout << "정렬된 배열기반\n";
    int b[] = {1, 2, 3};
    do{
        for(int i : b) cout << i << " ";
        cout << '\n';
    }while(next_permutation(b, b + 3));
}
/*
정렬되어있지 않은 배열기반
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
정렬된 배열기반
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
*/
```

참고로 이는 next_permutation()이라는 함수가 해당 배열의 그 “다음번째 순열”을 만들어내는 함수이기 때문에 그렇습니다. 예를 들어 1 3 2라는 배열로 시작한다면 다음 순열은 2 1 3 / 2 3 1 ... 이기 때문에 그런 것들만 뽑아내기 때문이죠. 올바르게

오름차순으로 정렬시켜 맨 처음부터인 1 2 3 / 1 3 2 .. 이런식으로 뽑아내게 만들어야 합니다.

또한, 앞의 코드는 정적배열 Array를 사용했기 때문에 vector와 같은 begin(), end()를 쓸 수 없습니다.

```
while(next_permutation(a, a + 3));
```

다만 배열의 첫번째 주소인 a와 배열의 끝 다음의 주소인 a + 3을 넣은 것을 볼 수 있습니다. 이는 다음 코드와 같은 의미가 됩니다. 정적배열 a의 순서를 정렬되게 수정한 뒤 코드를 바꿔보았습니다. 참고로 + 3은 배열의 크기가 3이기 때문에 배열의 끝 다음의 주소가 3이기 때문에 + 3인 것입니다. 만약 배열의 크기가 10이라면 + 10을 해야겠죠?

다음의 3개의 코드는 모두 같은 의미입니다.

1)

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a[] = {1, 2, 3};
    do{
        for(int i : a) cout << i << " ";
        cout << '\n';
    }while(next_permutation(&a[0], &a[3]));
}
```

앞의 코드는 &a[3]을 통해 배열의 마지막 인덱스 다음을 가리키고 있습니다.

2)

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a[] = {1, 2, 3};
    do{
        for(int i : a) cout << i << " ";
        cout << '\n';
    }while(next_permutation(&a[0], &a[0] + 3));
}
```

앞의 코드는 &a[0] + 3을 통해 배열의 마지막 인덱스 다음을 가리키고 있습니다. 주소값에서 +1, +2 ... 를 통해 다음 주소값을 가리킬 수 있기 때문입니다.

3)

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

```

using namespace std;
int main(){
    int a[] = {1, 2, 3};
    do{
        for(int i : a) cout << i << " ";
        cout << '\n';
    }while(next_permutation(a, a + 3));
}

```

앞의 코드는 배열의 이름은 pointer to decay가 되기 때문에 이름을 주소값으로 활용한 코드입니다.

재귀를 이용한 순열

0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

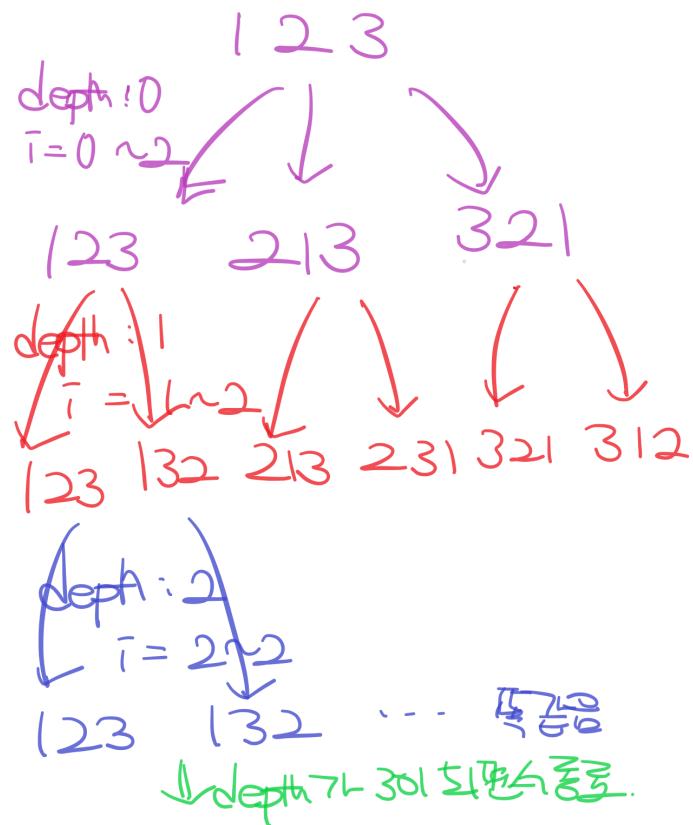
두번째 방법은 재귀를 이용한 방법입니다.

```

#include <bits/stdc++.h>
using namespace std;
int a[3] = {1, 2, 3};
int n = 3, r = 3; // r을 바꿔가면서 연습해보세요. :)
void print(){
    for(int i = 0; i < r; i++){
        cout << a[i] << " ";
    }
    cout << "\n";
}
void makePermutation(int n, int r, int depth){
    if(r == depth){
        print();
        return;
    }
    for(int i = depth; i < n; i++){
        swap(a[i], a[depth]);
        makePermutation(n, r, depth + 1);
        swap(a[i], a[depth]);
    }
    return;
}
int main(){
    makePermutation(n, r, 0);
    return 0;
}

```

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.

재귀를 이용한 순열 - 디버깅코드

실제로 앞의 코드에 디버깅 코드를 넣은 모습입니다. 이걸 실행해보면서 자신이 상상했던 로직과 비교해가며 이해를 하면 됩니다.

```
#include <bits/stdc++.h>
using namespace std;
int a[3] = {1, 2, 3};
int n = 3, r = 3; // r을 바꿔가면서 연습해보세요. :)
void print(){
    for(int i = 0; i < r; i++){
        cout << a[i] << " ";
    }
    cout << "\n";
}
void makePermutation(int n, int r, int depth){
    if(r == depth){
        print();
    }
}
```

```

        return;
    }
    for(int i = depth; i < n; i++){
        cout << i << " : " << depth << "를 바꾼다!\n";
        swap(a[i], a[depth]);
        makePermutation(n, r, depth + 1);
        cout << i << " : " << depth << "를 다시 바꾼다!\n";
        swap(a[i], a[depth]);
    }
    return;
}
int main(){
    makePermutation(n, r, 0);
    return 0;
}
/*
0 : 0를 바꾼다!
1 : 1를 바꾼다!
2 : 2를 바꾼다!
1 2 3
...
*/

```

조합

0주차 개념강의에 해당 파트 설명강의가 제공됩니다. 함께 보면서 공부해주세요.

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

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

${}_n C_r$ = 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.

재귀를 이용한 조합

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
*/
```

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 <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 = 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 <cstdio>  
#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
*/

```

조합의 특징 : $nCr = nC(n - r)$

$$nC_r = nC_{(n-r)}$$

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 나오기 때문이죠.

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%ED%98%95

이렇게 순열과 조합에 대해 알아봤습니다. 순열과 조합은 경우의 수를 기반으로 푸는 문제에 많이 활용됩니다. 예를 들어 여러가지 경우의 수를 생각해야 하는데 순서를 바꿔서 몇개를 뽑는다. 몇개를 설정한다. 라고 했을 때는 순서가 상관있으니 순열을 써야 하고, 순서를 바꾸지 않고 그저 몇개를 뽑는게 중요하다. 설정하는게 중요하다라는 게 있다면 조합을 써야 합니다.

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

다음 코드는 max_n 까지의 소수를 만들어서 출력하는 코드입니다.

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

const int max_n = 40;
bool che[max_n + 1];
// 예를 들어 40을 넣었을 때 che[40]이 참조가 되므로 max_n + 1을 넣어야 함.
// max_n까지의 소수를 만드는 함수.
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;
    }
int main(){
    vector<int> a = era(max_n);
    for(int i : a) cout << i << " ";
}

```

하지만 앞의 코드는 배열의 크기가 필요하기 때문에 배열의 크기가 일정 수준(1000만 이상)을 벗어나면 쓰기가 힘듭니다. 이럴 때는 일일히 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.

먼저 1부터 시작해 1씩 증가하는 수열의 합, 즉 1, 2, 3, 4, 5..의 합의 경우 아래와 같은 식으로 구할 수 있습니다.

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

예를 들어 $n = 5$ 일 때의 등차수열의 합은 다음과 같습니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    int n = 5;
    int ret = 0;
    for(int i = 1; i <= n; i++){
        ret += i;
    }
    cout << ret << '\n';
    cout << n * (n + 1) / 2 << '\n';
    return 0;
}
/*
15
15
*/
```

앞의 코드를 통해 for문을 사용하지 않고 $O(1)$ 만에 등차수열의 합을 구하는 것을 볼 수 있습니다.

그렇다면 초항이 3이고 등차가 5, n 이 5인 등차수열의 합은 어떻게 구할까요?

3, 8, 13, 18, 23의 합 = 65를 말이죠.

아래와 같은 수식으로 구할 수 있습니다. 등차수열의 합 공식입니다.

$$S_n = \frac{n(a + l)}{2}$$

다음코드는 등차수열의 합 공식을 이용한 코드입니다.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    int n = 5;
    int a = 3, l = 23;
    cout << n * (a + l) / 2 << '\n';
    return 0;
}
/*
65
*/
```

등비수열의 합

등비로 이루어진 배열의 합을 구하라는 로직 나올 수 있습니다. 이 때 활용되는 공식입니다.

$$\textcircled{1} \quad \frac{a(r^n - 1)}{r - 1} \quad (a: \text{초항} / r: \text{공비} / n: \text{더하는것의 개수})$$

$$\textcircled{2} \quad \frac{a}{1 - r} \quad (a: \text{초항} / r: \text{공비})$$

예를 들어 초항이 1이고 $r = 2$ 이고 n 은 4인 배열인 {1, 2, 4, 8} 의 합을 구해봅시다.
다음코드처럼 구할 수 있습니다.

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

```

int a = 1, r = 2, n = 4;
vector<int> v;
cout << a * ((int)pow(2, n) - 1) / (r - 1);
cout << '\n';
for(int i = 0; i < n; i++){
    v.push_back(a);
    a *= r;
}
for(int i : v) cout << i << ' ';
}
/*
15
1 2 4 8
*/

```

승수

승수를 구하는 로직이 필요할 때가 있습니다. 2의 2승, 3승 이런 로직 말이죠. 그럴 땐 pow()를 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.

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

따라서 int형으로 사용하고 싶다면 (int)로 형변환을 꼭 해주는게 중요합니다.

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 리턴합니다.

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

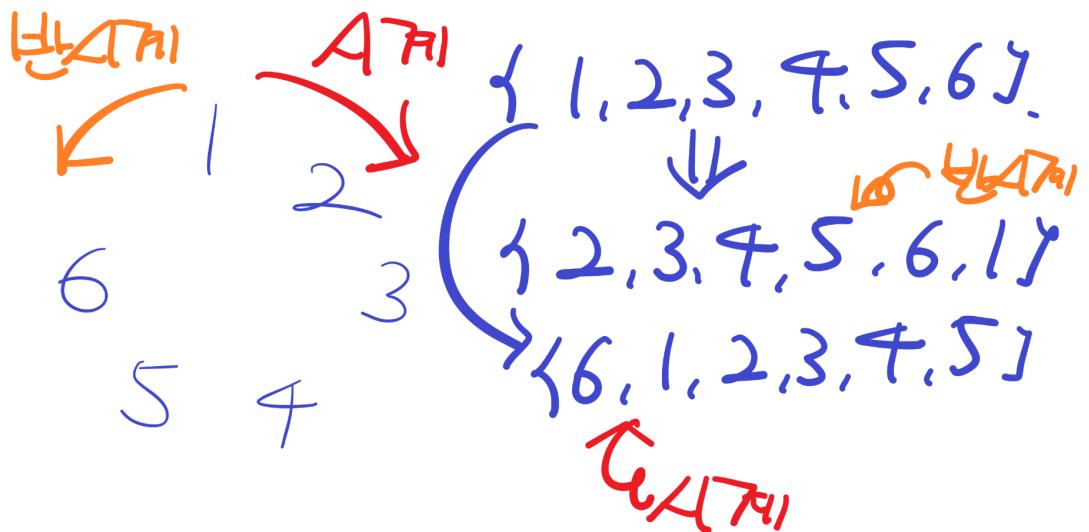
따라서 int형으로 사용하고 싶다면 (int)로 형변환을 꼭 해주는게 중요합니다.

1.12 코딩테스트 필수로직

코딩테스트에 자주 나오며 꼭 알아야 하는 로직을 배워보겠습니다.

1차원 배열 회전

배열을 시계방향 또는 반시계방향으로 회전해야 하는 로직을 구축해야 한다면 어떻게 해야할까요? 예를 들어 배열 {1, 2, 3, 4, 5, 6}에서 시계방향으로 한칸 움직이면 {6, 1, 2, 3, 4, 5}가 되고 반시계방향으로 한칸 움직이면 {2, 3, 4, 5, 6, 1}이 되는 것을 볼 수 있습니다.



어떻게 구현할 수 있을까요?

rotate()를 이용한 방법

첫번째는 rotate()를 이용한 방법입니다.

```
ForwardIterator rotate (ForwardIterator first, ForwardIterator middle,
ForwardIterator last);
```

rotate()에는 회전할 구간인 [first, last)를 집어놓고 몇칸 정도 회전할지를 집어넣으면 됩니다.

반시계방향 구축

배열을 한칸씩 반시계방향으로 왼쪽으로 이동해보겠습니다.

```

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

```

앞의 코드처럼 {1, 2, 3, 4, 5, 6}이라는 배열의 전체가 {2, 3, 4, 5, 6, 1}로 한칸씩 왼쪽으로 당겨지는 것을 볼 수 있습니다.

만약 2칸을 이동한다면 어떻게 해야할까요?

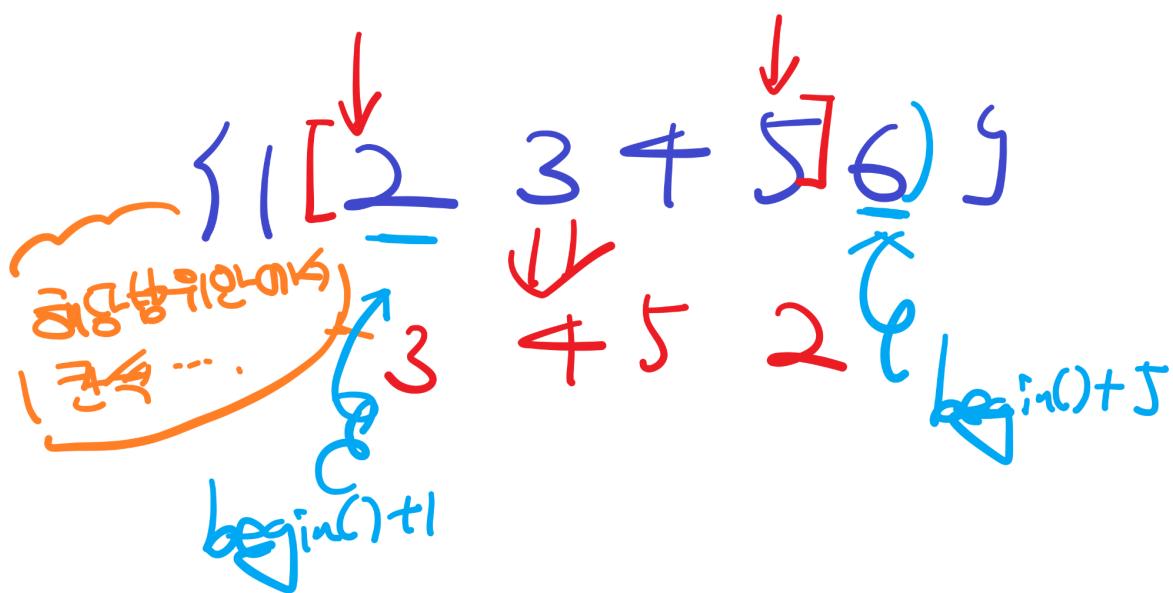
```

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

```

앞의 코드처럼 begin()부터 begin() + 2를 하면 됩니다. 전체를 기반으로 k칸을 이동하고 싶다면 begin() + k 이런식으로 집어넣으면 됩니다.

만약 배열의 전체가 아니라 일부분만 회전시키고 싶다면 어떻게 해야할까요?



앞의 그림처럼 $\{1, 2, 3, 4, 5, 6\}$ 중에서 $\{2, 3, 4, 5\}$ 부분만 한칸씩 이동시켜서 $\{1, 2, 3, 4, 5, 6\}$ 을 $\{1, 3, 4, 5, 2, 6\}$ 이렇게 만드는 것이죠.

다음 코드처럼 구축하면 됩니다. 시작지점 $v.begin() + 1$, 해당 부분으로부터 1칸이라면 시작지점으로부터 $+1$ 을 한 범위인 $v.begin() + 2$ 이렇게 하고 $v.begin() + 1$ 부터 시작해 $v.begin() + 5$ 까지 회전시킬 것이기 때문에 이렇게 넣어주면 됩니다.

다만 마지막 $v.begin() + 5$ 처럼 1번 ~ 4번까지 회전한다고 했을 때 마지막 지점은 포함되지 않는 끝점을 넣어줘야 하는 것을 주의해주세요.

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

시계방향 구축

그렇다면 시계방향은 어떻게 해야할까요? 앞서 설명한 반시계방향과 같이 하되 $begin()$, $end()$ 가 아니라 $rbegin()$, $rend()$ 를 사용하면 됩니다.

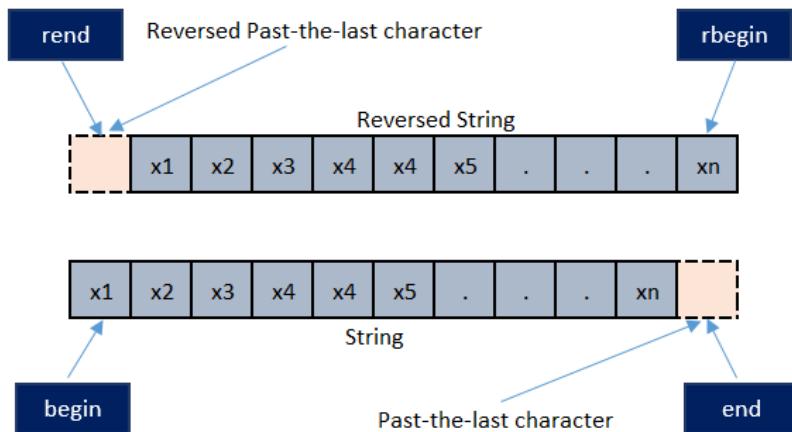
```
#include <bits/stdc++.h>
```

```

using namespace std;
int main(){
    vector<int> v = {1, 2, 3, 4, 5, 6};
    rotate(v.rbegin(), v.rbegin() + 1, v.rend());
    for(int i : v) cout << i << ' ';
}
/*
6 1 2 3 4 5
*/

```

`begin()`은 배열의 0번째부터 시작하는 것이라면 `rbegin()`은 배열의 마지막번째를 나타내는 이터레이터이며 `rend()`은 오른쪽부터 시작해 배열의 시작 전의 위치를 나타내는 이터레이터입니다.



직접 구현하는 방법

만약 함수를 사용하지 않고 앞의 코드를 구현한다면 어떻게 해야할까요? 1번 ~ 4번까지의 요소를 1칸씩 회전한다고 해보죠.

이런식으로 `temp`를 기반으로 코드를 구축하면 됩니다.

```

#include <bits/stdc++.h>
using namespace std;
int main(){
    vector<int> v = {1, 2, 3, 4, 5, 6};
    int i = 1;

```

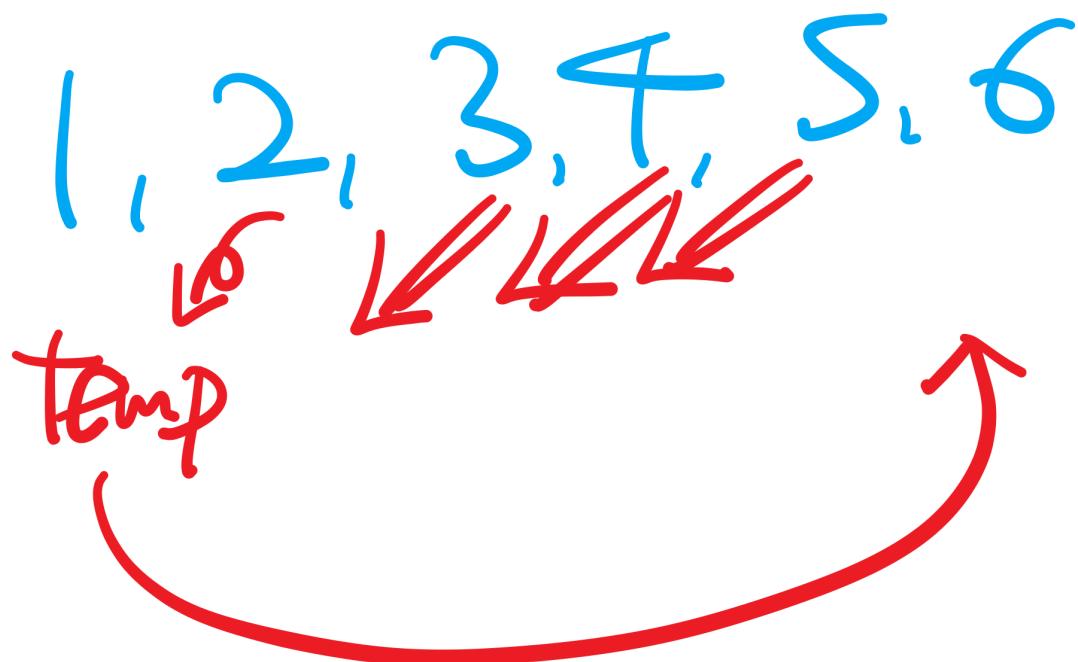
```

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
*/

```

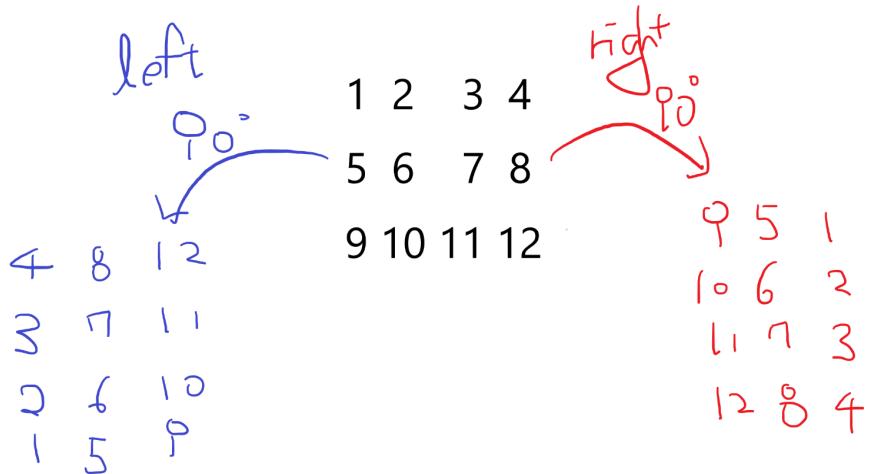
앞의 코드처럼 맨 앞을 temp에 담아놓고 뒤에 있는 것들을 배열에 요소에 집어넣은 다음, 마지막 temp를 마지막 요소에 집어넣는 코드입니다.

그림으로 설명하자면 다음과 같습니다.



2차원 배열 회전

2차원배열을 90도 회전시켜야 하는 경우가 있습니다.



앞의 그림처럼 2차원배열을 왼쪽 혹은 오른쪽으로 돌려야 하는 경우 어떻게 해야 할까요?
이 때 Array로 하는 것은 힘들고 vector를 기반으로 하는게 좋습니다.

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

void rotate_left_90(vector<vector<int>> &key){
    int n = key.size();
    int m = key[0].size();
    vector<vector<int>> temp(m, vector<int>(n, 0));

    for(int i = 0; i < m; i++){
        for(int j = 0; j < n; j++){
            temp[i][j] = key[j][m - i - 1];
        }
    }
    key.resize(m);
    key[0].resize(n);

    key = temp;
    return;
}

void rotate_right_90(vector<vector<int>> &key){
    int n = key.size();
```

```

int m = key[0].size();
vector<vector<int>> temp(m, vector<int>(n, 0));

for(int i = 0; i < m; i++){
    for(int j = 0; j < n; j++){
        temp[i][j] = key[n - j - 1][i];
    }
}
key.resize(m);
key[0].resize(n);

key = temp;
return;
}

int main(){
ios::sync_with_stdio(0); cin.tie(0);
vector<vector<int>> a = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12},
};
rotate_right_90(a);
for(int i = 0; i < m; i++){
    for(int j = 0; j < n; j++){
        cout << a[i][j] << " ";
    }
    cout << '\n';
}
return 0;
}

```

앞의 코드는 n과 m이 다를 때의 코드입니다.

resize()는 vector의 크기를 재할당하는 메서드입니다.

이 때 n과 m이 같다면 resize()부분이 사라지므로 다음코드처럼 더 간단해집니다.

만약 외우기 힘들다면 이 코드, rotate_right_90만이라도 외워주는게 좋습니다.

i, j = n - j - 1, i 라고 반복하면서 외우면 쉽습니다.

```

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

```

```

const int n = 3;
void rotate_right_90(vector<vector<int>> &key){
    vector<vector<int>> temp(n, vector<int>(n, 0));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            temp[i][j] = key[n - j - 1][i];
        }
    }
    key = temp;
    return;
}
int main(){
    ios::sync_with_stdio(0); cin.tie(0);
    vector<vector<int>> a = {
        {1, 2, 3},
        {5, 6, 7},
        {9, 10, 11},
    };
    rotate_right_90(a);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            cout << a[i][j] << " ";
        }
        cout << '\n';
    }
    return 0;
}

```

2차원 배열 대칭

예를 들어

1 2 3

4 5 6

7 8 9

라는 2차원 배열을

1 4 7

2 5 8

3 6 9

로 대칭시켜야 한다면 어떻게 해야할까요?

똑같은 크기의 배열을 하나 준비해서 j, i 이런식으로 담으면 됩니다.

```
#include<bits/stdc++.h>
using namespace std;
vector<vector<int>> v = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int b[3][3];
int main(){
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            b[j][i] = v[i][j];
        }
    }

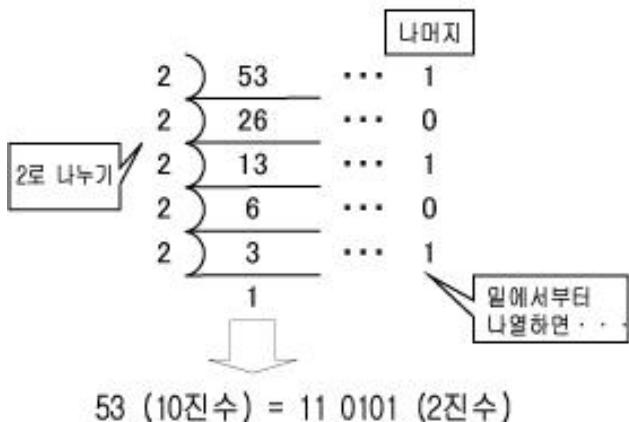
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            cout << b[i][j] << " ";
        }
        cout << '\n';
    }
    return 0;
}
/*
1 4 7
2 5 8
3 6 9
*/
```

n진법 변환

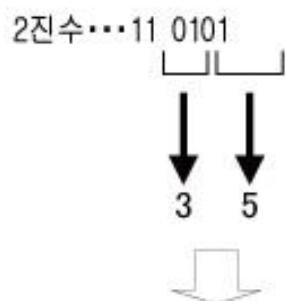
어떤 숫자 n에서 이를 b진법으로 어떻게 바꿀 수 있을까요?

2진법을 만들 때 나누기를 하고 몫을 이용하죠? 이런 그림을 그대로 코드로 구현하면 됩니다.

(예) 10진수 53을 2진수로 변환



(예) 2진수 110101을 16진수로 변환



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) 이 조건은 16진법 변환을 위해 필요함.
        // (16진법 : 0 ~ F로 표현하는 방법)
        if(a >= 10)cout << char(a + 55);
        else cout << a;
    }
    return 0;
}
```

여기서 b 를 3으로 바꾸면 10진법을 3진법으로 바꾸는 방법이 됩니다. 진법들을 테스팅해가면서 익히면 됩니다.

1.13 코딩테스트 팁

코딩테스트를 볼 때의 팁을 알려 드리겠습니다.

지역변수 보다는 전역변수를, 변수명을 간결하게.

지역변수 보다는 전역변수를

예를 들어 a, b, c를 출력하는 로직이 있다고 할 때 이런식으로 지역변수보다는 전역변수로 둬서 로직을 구축하는 것이 좋습니다.

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

지역변수는 stack에 할당이 되고 예측할 수 없는 쓰레기값이 할당이 됩니다. 전역변수는 bss segment 또는 data segment에 들어가면서 값을 초기화 하지 않은 경우 0으로 초기화가 됩니다.

또한, 지역변수로 선언하게 되면 stack에 쌓이는데 이 때 OS에서 성능상의 이유로 stack 영역에 메모리 제한을 걸어버립니다. 그렇기 때문에 많은 배열을 선언하지 못합니다. 다음과 같은 코드를 볼까요? (보통 지역변수 int형 배열은 25만까지 가능하다고 알려져 있습니다.) 전역변수로 tree를 10,000,000개나 설정한 코드는 잘 구동이 됩니다.

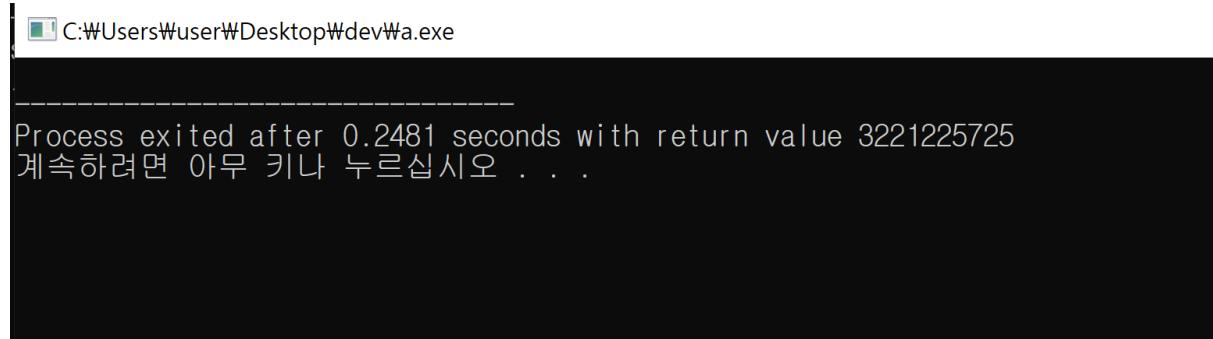
```
#include<bits/stdc++.h>
using namespace std;
int a, b;
int tree[10000000]; // 잘 됨.
int main(){
    cin >> a >> b;
}
```

그러나 다음과 같이 지역변수로 선언한 코드는 그 아래의 로직이 구동되지 않습니다.

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

```
int a, b;
int main(){
    int tree[10000000]; // 잘 되지 않음.
    cin >> a >> b;
}
```

실행화면



```
C:\Users\user\Desktop\dev\aa.exe
-----
Process exited after 0.2481 seconds with return value 3221225725
계속하려면 아무 키나 누르십시오 . . .
```

변수명은 간결하게

코딩테스트는 주어진 “시간” 내의 문제를 빠르게 그리고 잘 푸는 것이 핵심이다라는 것을 명심하고 또 명심합시다. 그렇기 때문에 변수명은 항상 간결하게 써야 합니다. count는 cnt로 result는 ret으로 하는 등 변수명을 짧게 써야 합니다. 문제에서 일곱난쟁이라고 했다고 sevendwarf 이런식으로 변수명을 짓는 시간도, 그 변수명을 치면서 낭비되는 시간도 아깝습니다.

배열의 경우 조금 더 넓게

예를 들어 int 형 10000의 최대범위를 가지는 문제가 있다면 이런식으로 4나 3 정도의 여유공간을 주는 게 좋습니다.

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

이를 통해 오버플로에 대한 신경을 덜 쓰게 만듭니다.

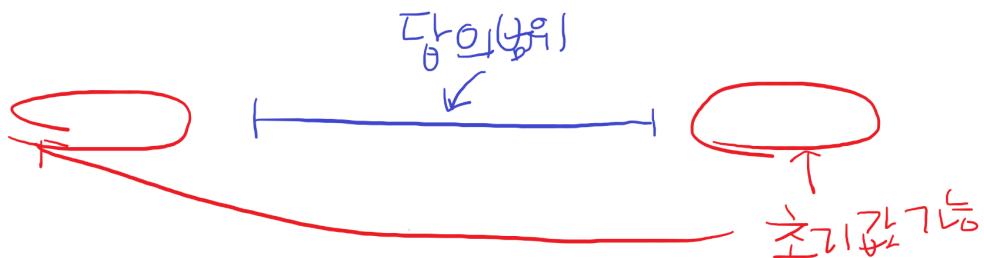
예를 들어 문제의 범위는 0 ~ 9 정도의 수가 나오기 때문에 board[10][10]으로 선언을 한 상황일 때 다음과 같은 코드를 작성하게 될 때가 있습니다. 이 때 이 코드는 문제가 없을까요?

```
if (visited[i][j]) {
    cur_money += board[i][j];
    cur_money += board[i - 1][j];
    cur_money += board[i + 1][j];
```

예를 들어 문제에서 i 가 9가 된 순간에 $board[10][9]$ 이런식으로 참조하게 됩니다. 이 경우 오버플로가 발생하게 되는 것이죠. 제가 정의한 배열의 최대 범위를 초과하게 되는 것입니다. 문제를 풀다보면 필요한 로직을 작성하는게 우선시 되다 보니 이런 범위초과에 대한 생각을 잘 못하게 됩니다. 그냥 넉넉히 추가합시다!

초기값은 답의 범위 밖에서.

어떠한 변수를 초기값을 잘못 설정해서 맞왜틀에 걸리는 경우가 있습니다. 초기값은 답의 범위밖에서 잡으면 됩니다.



문제의 “답”의 범위가 만약 $-1000 \sim 1000$ 이고 최솟값을 구하라라는 문제가 있습니다. 그렇다면 메인 로직은 $ret = \min(ret, value)$ 이런식으로 ret 을 갱신해 나가는 것 일겁니다. 그렇다면 초기 ret 은 어떻게 잡는게 좋을까요? 바로 1004 이런식으로 문제의 답의 범위 밖에서 잡는게 좋습니다.

그렇다면 최댓값을 구하라라는 문제는 -1004 를 기반으로 $ret = \max(ret, value)$ 로 ret 을 갱신해 나가야겠죠?

빠른 속도로 코딩하자

코딩테스트에서 중요한 것 중 하나는 속도입니다. 코드를 구축할 때는 간결하고 빠르게 짜야 합니다. 실제로 제가 아는 네이버 개발자의 경우 코딩테스트를 10분안에 모두 풀고 제출했고(All solved) 이 기록을 기반으로 면접관들이 지인을 좋게 봐서 합격했다고 합니다. 합격한 뒤에도 정말 10분안에 푼거 맞냐고 경이롭다는식으로 물어볼정도로 코테를 다 푸는 실력이라면 속도도 중요하다는 것입니다.

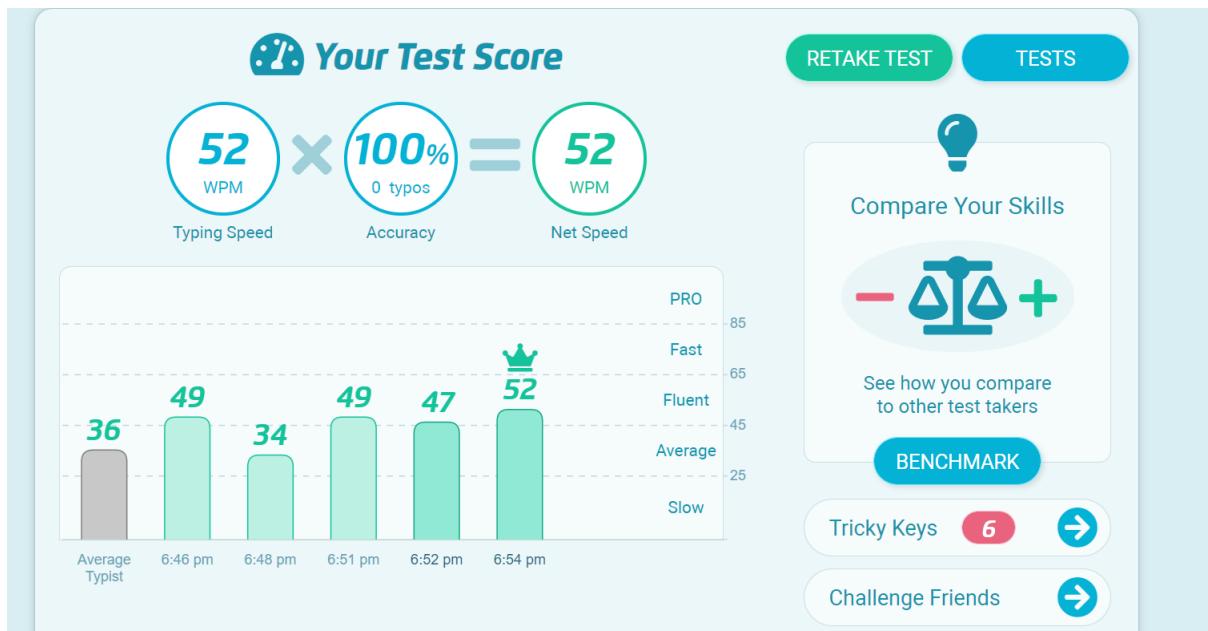
저같은 경우도 네이버면접을 볼 때 라이브코딩테스트 - DP문제를 면접관이 준비했고 3분안에 해당문제를 풀었고 해당 면접관이 좋게 봐서 합격했던 기억이 있습니다.

이렇게 빠르게 코딩테스트를 보려면 빠르게 문제지문을 해석하는 실력, 정확도, 유연하고 빠른 사고력, 그리고 여기서 말할 타자속도 또한 중요합니다.

아래의 사이트로 가면 타이핑을 연습할 수 있습니다.

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

참고로 필자의 타이핑 실력은 52wpm입니다. 45이상이 되도록 연습해봅시다.



이정도 숫자는 외우자

시간복잡도를 어림잡아 계산할 때 자주 나오는 수들. 이정도는 외워줍시다.

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

외울 때 그냥 $10!$ 은 360만, 3^{10} 은 6만 정도... 이렇게 외우면 됩니다.

1.14 맞왜틀 팁

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 버린 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 발생됩니다. 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

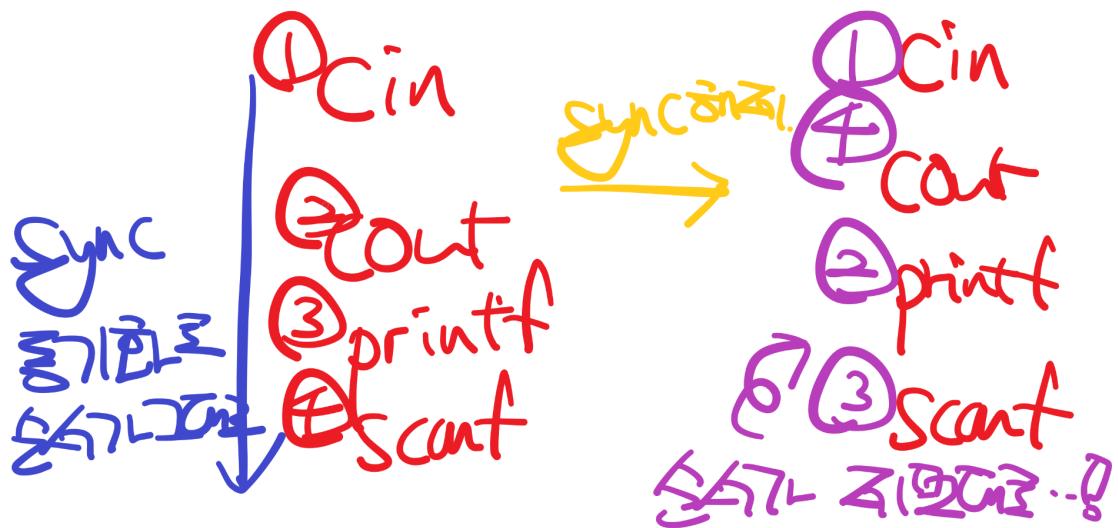
입출력 싱크를 풀어서 cin, cout의 속도를 빠르게 설정하는 방법이 있습니다. 시간초과가 났을 때 다음 코드처럼 구축한다면 시간초과가 통과하는 경우도 있습니다.

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

C++에서 대표적인 입출력 함수로는 cin, cout, scanf, printf가 있고 이는 시스템콜관련 함수이므로 같이 쓰일 때는 입출력싱크를 맞춰주어야 합니다.

보통은 cin, cout과 scanf, printf 중 scanf, printf가 빠릅니다. 왜냐하면 cin, cout은 c라이브러리인 stdio의 버퍼와 동기화하느라 시간을 소비하기 때문이죠.

이 때 앞의 코드처럼 싱크를 해제하여 버퍼 동기화를 하지 않게 하여 cin, cout의 구동속도를 빠르게 해줄 수 있습니다. 하지만 이렇게 했을 때 반드시 !!cin, cout를 쓸 때 scanf, printf를 쓰지 말아야 합니다.



앞의 그림처럼 순서를 맞춰주는 동기가 풀려서 순서가 제멋대로 실행될 수 있습니다.
앞의 코드를 쓴다면 반드시 cin, cout만 써야 합니다.

스택오버플로



스택오버플로(stack overflow)는 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.

예를 들어 문제에서 테스트케이스가 여러개 주어지고 주어진 테스트케이스 당... 하는 문제가 있습니다. 그럴 경우 보통 다음과 같은 코드 그리고 while 루프 안에서 로직이 이어질텐데 이 안에서 변수 초기화가 잘 되었는지 항상 확인합시다.

```
cin >> t;
while(t--)
    // 로직
```

실수형 연산의 제한된 정확도

실수형연산은 제한된 정확도를 가집니다.

예를 들어 0.1을 10번 더한다고 해봅시다.

```
#include<bits/stdc++.h>
int main(){
    if(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 1){
        printf("clear\n");
    }
}
```

분명히 0.1을 10번 더했는데도 불고하고 clear를 출력되지 않습니다. 컴퓨터는 숫자들을 십진법으로 저장하는게 아니라 이진법으로 저장하게 됩니다.

0.1은 즉, 0.00011 ... 으로 “무한 소수”로 굉장히 복잡한숫자로 저장이 되기 때문에 해당 숫자끼리의 연산은 정확하지 않을 수 있습니다.

보통 실수형을 다룰 때는 float아니면 double을 다루는데 float은 4바이트짜리 타입이고 최대 소수점 7자리까지 정확하게, double은 8바이트짜리 타입이고 최대 소수점 15자리까지 정확하게 저장할 수 있습니다. 따라서 좀 더 정확한 double을 쓰는게 좋습니다.

다만, == 식의 연산은 앞의 코드처럼 잘되지 않는 경우가 많습니다. 애초에 정확하지 않으니까요. 따라서 실수형 연산은 다음과 같이 오차를 기반으로 해당 오차미만의 값인지를 기반으로 구축하는게 좋습니다. ($1e-9$ 는 10^{-9} 을 뜻합니다.)

```
#include<bits/stdc++.h>
int main(){
    if(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 - 1 < 1e-9){
        printf("clear\n");
    }
}
```

```
}
```

이를 기반으로 프레임화시키면 다음과 같은 코드를 생각할 수 있습니다.

```
#include<bits/stdc++.h>
double a = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
double b = 1;
int main(){
    double value = a - b;
    if(value < 1e-9){
        printf("clear\n");
    }
}
```

하지만 애초에 문제를 풀 때 실수형이 나온다면 되도록 정수형으로 바꿔서 푸는 것이 좋습니다.

문자열 크기 선언

예를 들어 100개짜리 문자를 입력받는다. 또한 이걸 string이 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

queue나 stack에서 top이나 pop 연산을 할 때 항상 size를 체크해야 합니다. 아래코드처럼 구축해야 합니다.

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

만약 해당 자료구조에 아무것도 없는데 해당 자료구조 내의 요소를 참조하려고 할 경우 참조에러(reference Error)가 발생할 수 있습니다.

UB

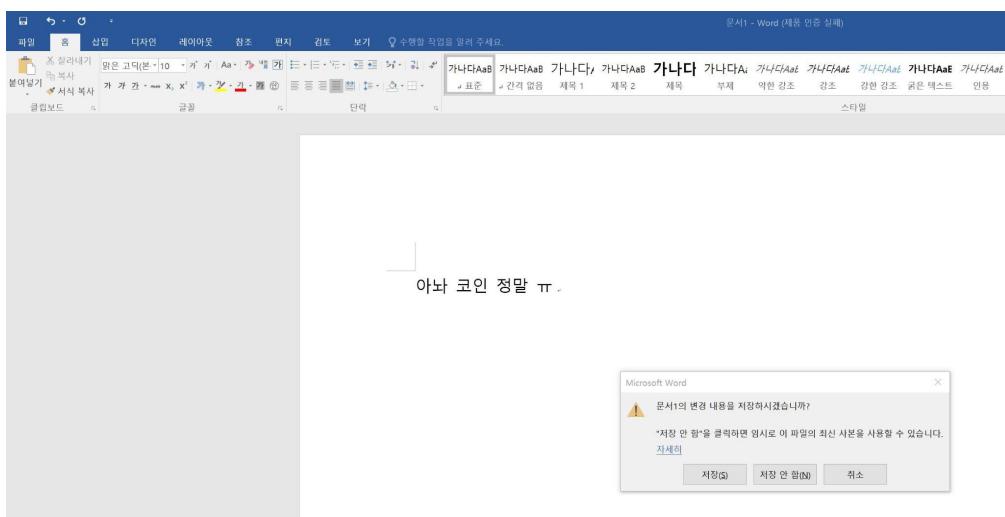
배열 인덱스 밖으로 벗어난 곳을 참조하는 경우가 대표적이며 예상치 못한 결과가 나타났을 때 UB(Undefined Behavior)가 발생했다고 합니다. 이외에도 잘못된 로직을 구축했을 때 예상한 결과와 다른 결과값 또는 에러가 발생했을 때 UB가 떴다고 말합니다.

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 사용자에 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
*/

```

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 중요하지 않습니다. 어쨌듯 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. :)

구현문제를 잘 푸는 방법

영상을 참고해주세요.

<https://youtu.be/DBXEWJx2mlw>

1.15 자주 묻는 질문

Q. 삼성코딩테스트에서 bits/stdc++.h를 쓸 수 있나요?

21년까지는 삼성전자 코딩테스트의 경우 bits/stdc++.h 라이브러리는 쓰지 못하였습니다.

그러나 22년 기준, 사용 가능하다고 알려져있습니다.

삼성 코딩테스트와 같은 환경이라고 알려져있는 삼성전자 SW 역량테스트 A형의 사용 가능한 라이브러리 목록입니다. 다음과 같이 제한 없음이라고 되어있는 것을 볼 수 있습니다.



상시 SW 역량테스트 구성

평가기준 : TestCase 전체 Pass, 실행속도, 코드리뷰 등

구분	검정시간	지원언어	사용 가능한 라이브러리	샘플문제	추천 연습문제
A형	3시간	C/C++/Java/Python	제한 없음	풀어보기	D2~4

다만 21년 기준으로 돌아갈 수 있기 때문에 해당 부분을 위한 설명을 덧붙입니다.

라이브러리마다 제공되는 함수가 있으며 내가 만든 로직에 어떠한 함수가 들어간다면 해당 라이브러리에서 제공하는지를 알아야 합니다.

그러나 사실 저는 이러한 라이브러리를 외우는 것은 추천드리지 않습니다. 왜냐하면 cin 이 안돼? scanf를 쓰면 되지 등으로 내가 필요한 함수인데 이 함수가 안되면 같은 기능을 하는 함수를 쓰면 되기 때문입니다. 예를 들어 코딩테스트 환경에서 다음과 같이 iostream만을 지원한다면 scanf 말고 cin을 쓰면 되는 것입니다.

```
#include <iostream>
```

그러나 불안하신 분들을 위해 준비했습니다. 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 자주 쓰는 함수

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

string.h의 자주 쓰는 함수

- memcpy
- memset
- size_t

algorithm의 자주 쓰는 함수

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

Q. 따닥따닥 붙어있는 것을 어떻게 입력받죠?

이렇게 따닥따닥 입력이 붙어서 주어지는 경우가 있습니다.

어떻게 입력을 받아야 할까요?

```
1000
0000
0111
0000
```

두가지 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
*/
```

문자열 s를 받아 문자열을 문자로 분해(s[j]) 해서 타입변환 s[j] - '0'을 통해 숫자를 int 타입 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.

그러나 scanf는 문자타입(char)을 입력할 때는 좀 주의해야 합니다.

다음 코드처럼 띄어쓰기를 %c앞에 넣어서 입력을 받아야 합니다.

```
scanf(" %c", &a[i][j]);
```

예를 들어 다음 문자열을 입력받는다고 해보겠습니다.

```
LLMM  
MMLL
```

그럼 다음과 같이 코드를 구축해야 합니다.

```
#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++){
            printf("%c", a[i][j]);
        }
    }
}
```

```

        printf("\n");
    }
    return 0;
}
/*
입력
LLMM
MMLL

출력
LLMM
MMLL

*/

```

숫자인 d로 입력을 받게 되면 이런 현상이 일어나지 않지만 c로 받게 되면 특수문자, 엔터를 문자로 취급해서 입력을 받기 때문에 이렇게 신경을 써주어야 합니다.

3.char타입 & cin

다만, char 타입으로 2차원 배열을 정의했을 때는 다음코드처럼 cin으로 그냥 받을 수 있습니다.

```

#include<bits/stdc++.h>
using namespace std;
char a[54][54];
int main(){
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 4; j++){
            cin >> a[i][j];
        }
    }
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 4; j++){
            cout << a[i][j] << " ";
        }
        cout << '\n';
    }
}
/*
LLMM
MMLL
*/

```

cin으로 int타입을 입력받게 되면 여러개의 문자들이 입력으로 오는 것을 기다렸다가 개행문자까지 입력을 받지만 char 같은 경우 한 문자까지밖에 입력을 못받기 때문에 개행문자가 없어도 개행문자가 있는 것과 같은 역할을 합니다.

따라서 다음코드와 똑같은 효과를 발휘합니다.

```
#include<bits/stdc++.h>
using namespace std;
char a[54][54];
int main(){
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 4; j++){
            cin >> a[i][j];
        }
    }
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 4; j++){
            cout << a[i][j] << " ";
        }
        cout << '\n';
    }
}
/*
L L M M
M M L L
*/
```

필자는 웬만하면 cin으로 입력받고 cout으로 출력하는 것을 추천드립니다.

Q. 문제에서 입력의 끝을 정하지 않은 경우

보통의 문제는 n번까지 입력을 받는다. 이런게 일반적이지만... 이런 문제가 있습니다.

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';
    }
}
```

Q. process exit call이란?

프로그램이 실행이 되면 메모리에 올라가 프로세스가 됩니다. 이 프로세스를 종료시킨다라는 의미입니다.

stdlib.h 를 보면 다음과 같이 설명되어있습니다.

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

즉, 0은 성공적으로 종료했다를 의미하고 1은 실패적으로 종료했다를 의미합니다. 그렇기 때문에 return 0으로 프로세스를 성공적으로 종료한다라는 코드를 씁니다.

이러한 것을 main 함수가 아닌 다른 함수에서도 강제적으로 선언할 수 있습니다. exit(0);을 통해서 강제적으로 main함수를 종료할 수 있는 것이죠.

일곱난쟁이 문제를 풀면서 알아보겠습니다.

<https://www.acmicpc.net/problem/2309>

이 문제를 푸는 방법은 여러가지가 있지만 이렇게도 풀어봅시다.

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

using namespace std;

int a[9];
void printV(vector<int> v) {
    for (int i : v) cout << i << '\n';
}
void combi(int start, vector<int> &b) {
    if (b.size() == 7) {
        int sum = accumulate(b.begin(), b.end(), 0);
        if (sum == 100){
            printV(b);
            exit(0);
        }
    }
    for (int i = start + 1; i < 9; i++) {
        b.push_back(a[i]);
        combi(i, b);
        b.pop_back();
    }
    return;
}

int main() {
    for(int i = 0; i < 9; i++) cin >> a[i];
    sort(a, a + 9);
    vector<int> v;
    combi(-1, v);
    return 0;
}
```

이 문제는 난쟁이 7명의 키의 합, $\text{sum} == 100$ 일 때 출력을 하고 종료를 해야 하는 문제이자, $\text{sum} == 100$ 인 경우의 수가 여러가지가 있을 수 있습니다.

즉, $\text{sum} == 100$ 이라면 main함수를 종료시켜야 합니다. 만약 combi라는 함수만을 종료시키려고 return 을 쓴다면 $\text{sum} == 100$ 인 경우의 수가 2개이상일 경우 2번 출력이 되서 이 문제는 틀리게 되겠죠?

따라서 combi 함수에서 if($\text{sum} == 100$)일 때 return을 하는 것이 아닌 exit(0)으로 process exit call을 해야 합니다.

코드로 다시 말하자면, 다음과 같은 코드처럼 return이 아니라 exit(0)을 해서 함수가 아니라 main함수를 종료시켜야 합니다.

```

//before
    if (sum == 100){
        printV(b);
        return;
    }
//after
    if (sum == 100){
        printV(b);
        exit(0);
    }

```

Q. 코테에서 입력이 2차원 배열로 주어졌어요. 어떻게 하죠?

코딩테스트에서 입력이 [[0, 0, 1], [1, 0, 1], [0,1,0]] 이런식으로 주어진 경우 어떻게 입력을 받나요?

```

#include<bits/stdc++.h>
using namespace std;
vector<vector<int>> v = {{0, 0, 1}, {1, 0, 1}, {0,1,0}};

int main(){
    int n = v.size();
    int m = v[0].size();
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << v[i][j] << " ";
        }
        cout << '\n';
    }
    return 0;
}

```

출력:

```

0 0 1
1 0 1
0 1 0

```

2차원 배열의 경우 높이는 `v.size()`, 너비는 첫번째 요소의 길이를 통해 추출하면 됩니다.

1.16 알고리즘을 공부하는 자세

집중하자.

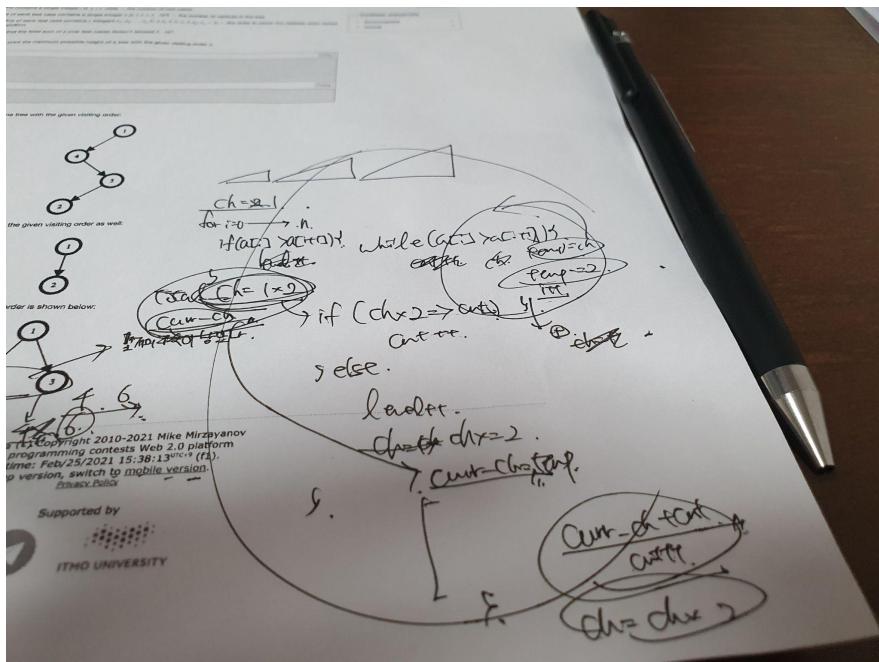
혼자 알고리즘을 공부할 때는 항상 핸드폰을 끄고 초집중해서 공부를 하시는 것을 추천드립니다. 또한 하루 3 ~ 4시간 이상은 하셔야 실력이 늵니다. 다음은 푸는 방법입니다.

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

다양하게 풀 수 없을까?

다양하게 풀 생각을 해야 합니다. 어떠한 문제를 예를 들어 `void`형 함수를 구현해서 풀었다고 해봅시다. 그 다음에는 `int`형으로 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 $for i = 0 \rightarrow n$ 이런식으로 표시해서 코딩을 합니다. “자신이 알아들을 수 있는” 방법으로 하시면 됩니다.

이렇게 알고리즘 용 C++의 기초를 다뤄보았습니다. 코테 말고도 만약 면접을 본다면 정렬알고리즘 정도는 외워야 합니다. 다음 링크를 추천합니다.

- 정렬알고리즘 정리 : <https://blog.naver.com/jhc9639/221338179774>

1.17 재밌게 공부하는 방법

제 강의 내에 있는 문제들을 모두 풀기란 어렵습니다. 문제수도 많고 가면 갈수록 난이도가 높은 문제들이 있기 때문이죠. 다만, 그 어려움을 이겨내면 코딩테스트에 합격할 수 있습니다. 저를 믿고 꾸준하게 풀면 되는데 이번에는 꾸준하게 풀 수 있도록 재밌게 푸는 방법을 얘기해보겠습니다.

그룹을 만들어 연습을 만듭시다. 그리고 그 연습을 깨가며 문제를 풀어봅시다.

<https://www.acmicpc.net/group/list>

참고로 그룹은 문제를 50개 이상 풀어야 만들 수 있기 때문에 50개정도는 풀고 진행하면 됩니다.

그룹에는 두명 이상의 아이디가 필요할 때는 자신의 아이디, 그리고 제 부캐인 chovy3을 집어넣어서 진행하면 됩니다.

지금은 한명의 아이디로도 그룹을 만들 수 있습니다.

메인	문제집	문제집 만들기	채점 현황	연습	1주차	연습 만들기	랭킹	게시판	글쓰기	파일	설정	관리
A - 패션왕 신해빈												
B - 팰린드롬 만들기												
C - 주몽												
D - 좋은 단어												
E - 곱셈												
F - 1												

저와 함께라면 반드시 코딩테스트 따위, 정복할 수 있습니다.

저만 믿고 지금부터 8주차까지 힘내봅시다.