

# 5주차 : 함수와 재귀함수

---

강사: 김현수

# 0. 목차

---

# 0. 목차

1. 함수란?
2. 지역변수와 전역변수
3. 함수의 선언
4. 재귀함수란?
5. 문제 풀이
6. 부록

# 1. 함수란?

---

# 1. 함수란?

소입설에서 파이썬으로 함수를 다들 만들어 보았을거예요! (배웠겠죠..?)

Q. 그럼 다음 두 함수의 차이점이 뭘까요?

```
def printSum(a, b):  
    print(a + b)
```

```
x = 1  
y = 2
```

```
printSum(x, y)
```

```
def returnSum(a, b):  
    return a + b
```

```
x = 1  
y = 2
```

```
print(returnSum(x, y))
```

A. 바로 return 이 있느냐 없느냐 차이입니다!

파이썬에서는 내마음대로 return을 적어도 되고~ 안 적어도 되고~ 그랬지만  
간깐한 C언어는 처음부터 return을 할 건지 안 할 건지 딱 명시해주어야 해요.

# 1. 함수란?

```
def printSum(a, b):  
    print(a + b)
```

VS

```
3 void printSum(int a, int b) {  
4     printf("%d\n", a + b);  
5     return;  
6 }
```

```
def returnSum(a, b):  
    return a + b
```

VS

```
8 int returnSum(int a, int b) {  
9     return a + b;  
10 }
```

파이썬 함수와 C로 만든 함수 비교해보기  
각각의 차이점이 느껴지나요?

# 1. 함수란?

```
def printSum(a, b):  
    print(a + b)
```

VS

```
3 void printSum(int a, int b) {  
4     printf("%d\n", a + b);  
5     return;  
6 }
```

```
def returnSum(a, b):  
    return a + b
```

VS

```
8 int returnSum(int a, int b) {  
9     return a + b;  
10 }
```

함수는 다음과 같은 구조로 이루어집니다.

리턴타입 함수이름(매개변수)  
{ 함수 정의 }

함수 선언의 첫째 줄 “리턴타입 함수이름(매개변수)” 이 부분을 함수 원형 (function prototype)이라고 부릅니다. 함수에서 가장 중요한 정보를 담고있어요.

어떤 정보들을 담고있는지는 다음장에서 비교해볼거예요!

# 1. 함수란?

```
3 void printSum(int a, int b) {  
4     printf("%d\n", a + b);  
5     return;  
6 }
```

```
8 int returnSum(int a, int b) {  
9     return a + b;  
10 }
```

Function prototype이 가지고있는 정보

## 1. 매개변수(parameter)

Python : 변수 이름만 쓰면 됨.

C-style : 변수 자료형까지 써줘야 됨.

## 2. 리턴타입

Python : 정해두지 않아도 됨.

C-style :

리턴 값이 int라면 int, 없다면 void 등으로  
명확하게 적어 주어야 함. void의 경우 return;을  
적어줘도 되고 안 적어줘도 됨.



# 1. 함수란?

```
12 int main(void) {  
13     int x = 1;  
14     int y = 2;  
15  
16     printSum(x, y);  
17     returnSum(x, y);  
18  
19     return 0;  
20 }
```

흔히 보던 main함수의

return 타입은 ? int

parameter은 ? void (매개변수 없다는 뜻. 아예 void도 안 적어도 됨.)

return 문장은 ? return 0;

(근데 유일하게 main함수 한정으로 return타입이 int인데도 불구하고  
return 0;를 생략 해도 된다!)

## 2. 지역변수와 전역변수

---

## 2. 지역변수와 전역변수

Q. 다음 코드가 틀린 이유는?

```
1  #include <stdio.h>
2
3  void printSum() {
4      printf("%d\n", x + y);
5  }
6  int main(void) {
7      int x = 1;
8      int y = 2;
9
10     printSum();
11
12     return 0;
13 }
14
```

식별자 "y"이(가) 정의되어 있지 않습니다.

A. int x, int y는 main함수 안에서 선언된 지역변수이기 때문!

## 2. 지역변수와 전역변수

### 1. 함수 안에서 선언하기(지역변수) => 함수 밖에서는 사라진다

```
1  #include <stdio.h>
2
3  void printSum() {
4      printf("%d\n", x + y);
5  }
6  int main(void) {
7      int x = 1;
8      int y = 2;
9
10     printSum();
11 }
```

단점 : main 함수 안에서 선언한 x, y는  
다른 함수인 printSum에서는 접근 불가 (빨간줄 보이나요)

### 2. 함수 밖에서 선언하기(전역변수) => 아무데서나 쓸 수 있다

```
1  #include <stdio.h>
2
3  int x = 1;
4  int y = 2;
5  void printSum() {
6      printf("%d\n", x + y);
7  }
8  int main(void) {
9      printSum();
10 }
```

장점 : 아예 함수 밖에서 선언한 x, y는 아무데서나 접근 가능  
=> 일일이 함수의 파라미터로 넘기기 귀찮으니까 웬만한 변수는 전역변수로 선언하자 !  
+ 전역 변수는 초기화하지 않아도 디폴트값(0, \0, false 등)으로 초기화 됨.  
+ 크기가 큰 배열의 경우는 전역변수로만 선언가능하기도 하다.  
-> 뭐다? 전역 변수가 짱이다ㅇㅇ

## 2. 지역변수와 전역변수

1. 함수 안에서 선언하기(지역변수) => 함수 밖에서는 사라진다

```
1  #include <stdio.h>
2
3  void printSum() {
4      printf("%d\n", x + y);
5  }
6  int main(void) {
7      int x = 1;
8      int y = 2;
9
10     printSum();
11 }
```

장점 : 소입설 과제 등 실제 개발을 할 때는 지역변수로 선언해야  
정보가 안전하게 보관이 된다.

2. 함수 밖에서 선언하기(전역변수) => 아무데서나 쓸 수 있다

```
1  #include <stdio.h>
2
3  int x = 1;
4  int y = 2;
5  void printSum() {
6      printf("%d\n", x + y);
7  }
8  int main(void) {
9      printSum();
10 }
```

단점 : 실제 개발을 할 때 여러 사람이 함께 작업하는 경우 전역변수로  
선언한다면 의도치 않게 변수가 잘못 쓰일 수도 있다.  
PS할 때만 적극 지향하자!

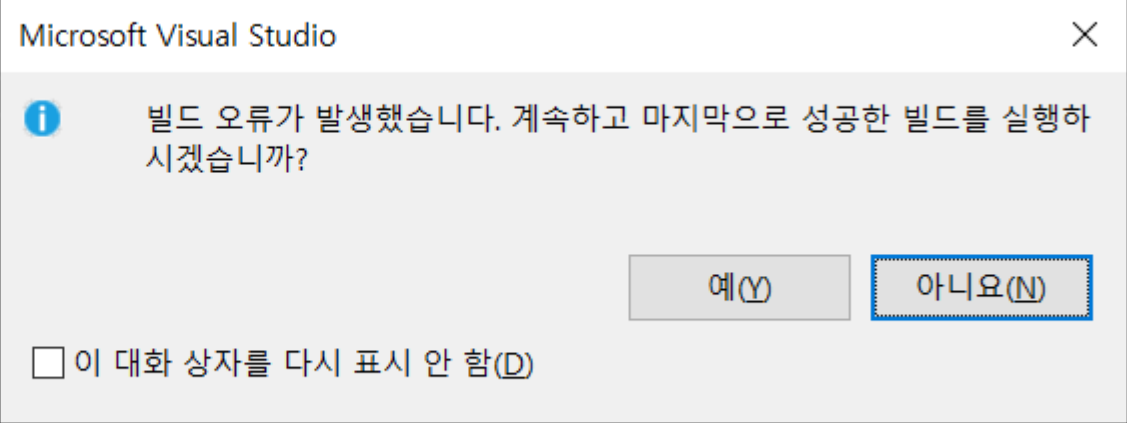
### 3. 함수의 선언

---

### 3. 함수의 선언

Q. 다음 코드가 틀린 이유는?

```
1  #include <stdio.h>
2
3  int x = 1;
4  int y = 2;
5
6  int main(void) {
7      printSum();
8  }
9
10 void printSum() {
11     printf("%d\n", x + y);
12 }
```



A. printSum이 선언(10번줄)되기도 전에 main에서 호출(7번줄)했기 때문!

# 3. 함수의 선언

## 방법 1. main함수 위에다가 선언한다

```
1  #include <stdio.h>
2
3  int x = 1;
4  int y = 2;
5  void printSum() {
6      printf("%d\n", x + y);
7  }
8  int main(void) {
9      printSum();
10 }
```

장점 : 간단하다. PS할 때는 주로 이렇게 한다.

## 방법 2. main함수 밑에다가 정의하되, main위에 프로토타입을 적어준다 (3번줄)

```
1  #include <stdio.h>
2
3  void printSum(int x, int y);
4  int main(void) {
5      printSum(1, 2);
6  }
7  void printSum(int x, int y) {
8      printf("%d\n", x + y);
9  }
10
```

장점 : 소입설과 창소프에서 주로 이렇게 하게 된다.

함수가 많~아질 경우 main이 너무 밀려나는 것을 방지할 수 있다.



### 3. 함수의 선언

```
1 #include <stdio.h>
2
3 void printSum(int x, int y) {
4     if(is_natural(x))
5         printf("%d\n", x + y);
6 }
7 bool is_natural(int a) {
8     if (a > 0) return true;
9     return false;
10 }
11 int main(void) {
12     printSum(1, 2);
13 }
```

오류 목록

전체 솔루션 1 오류 0 경고

C3861 'is\_natural': 식별자를 찾을 수 없습니다.

#### 방법 1. main함수 위에다가 선언한다

단점 : 함수가 여러 개 일 경우 먼저 쓰는 함수를 먼저 정의해야 한다.

특징 : 근데 당연한 이야기다. 눈 크게 뜨고 잘 보면 된다. 이 방법을 지향하자.

```
1 #include <stdio.h>
2
3 void printSum(int, int b);
4 int main(void) {
5     printSum(1, 2);
6 }
7 void printSum(int x, int y) {
8     printf("%d\n", x + y);
9 }
```

#### 방법 2. main함수 밑에다가 정의하되, main위에 프로토타입을 적어준다

특징 : 선언할 때는 변수 이름 안 써줘도 되고 정의할 때와는 다른 걸 써줘도 됨

단점 : 귀찮다. 귀찮은 건 속도가 생명인 PS에서는 지양해야 한다.

소입설 과제할 때만 쓰자.

### 3. 함수의 선언

여기서 잠깐! 여기 왜 return이 2번이나 쓰였을까?

```
7  bool is_natural(int a) {  
8      if (a > 0) return true;  
9      return false;  
10 }
```

return은 결과값을 호출한 곳으로 반환하는 역할도 있지만 이 함수를 끝내는 역할도 있다!  
while문의 break랑 비슷한 느낌이다. 특정 조건에서 함수를 끝내고 싶다면 return 해주면 된다.  
반환형이 void인 함수도 return;이 있는 이유인 것 같다.(강사 뇌피셜)  
(근데 객지를 듣다 보니 다른 이유가 있는 것 같긴 하다. 여튼 이런 쓰임새가 있다ㅎ)

## 4. 재귀함수란?

---

(왠지 어느 순간부터 반말로 바뀐 기분이다)

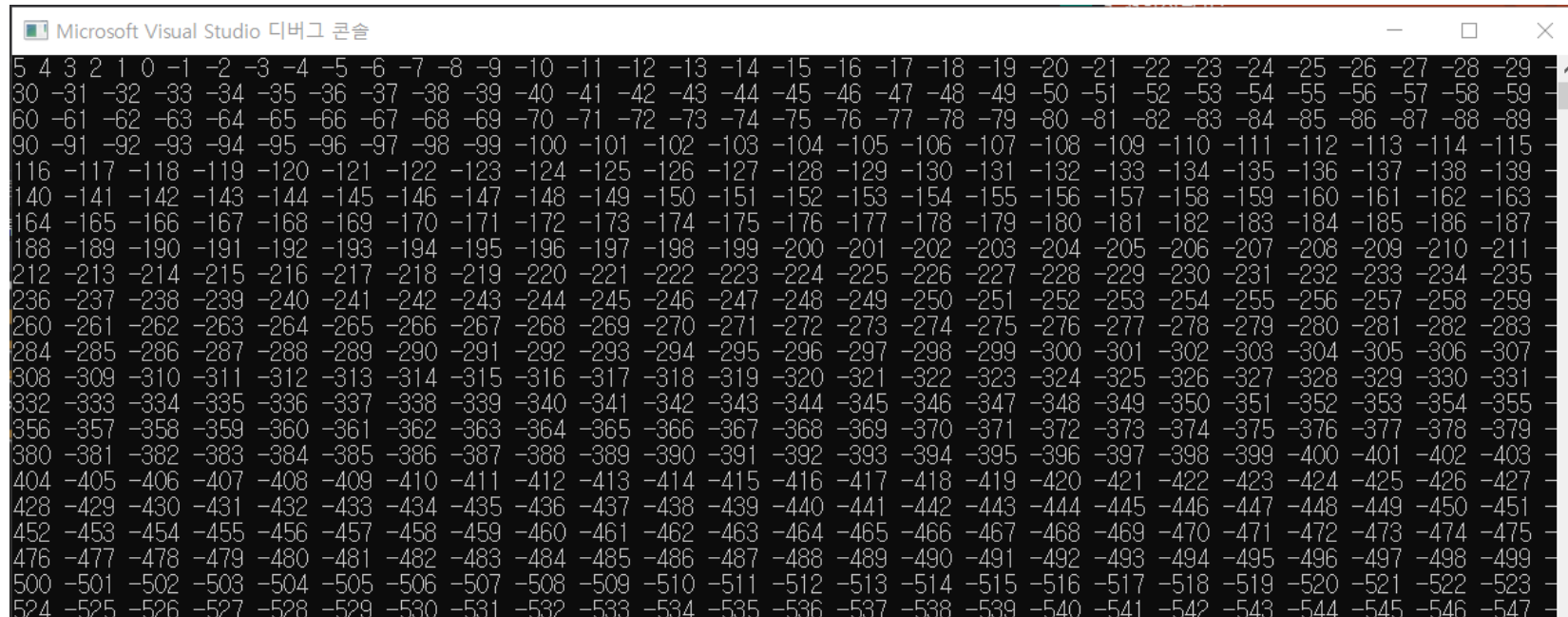
## 4. 재귀함수란?

재귀 함수(recursion)란 함수 안에서 자기 자신을 호출하는 함수입니다!  
recursion 함수 안에서 recursion이 호출된 것이 보이시나요?

```
1  #include <stdio.h>
2
3  void recursion(int i) {
4      printf("%d ", i); //i를 출력하고
5      recursion(i - 1); //재귀 호출
6  }
7
8  int main() {
9      recursion(5);
10 }
11
```

자 그럼 이 코드의 출력 결과가 어떻게 나올까요?!

## 4. 재귀함수란?



```
Microsoft Visual Studio 디버그 콘솔
5 4 3 2 1 0 -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 -101 -102 -103 -104 -105 -106 -107 -108 -109 -110 -111 -112 -113 -114 -115 -
116 -117 -118 -119 -120 -121 -122 -123 -124 -125 -126 -127 -128 -129 -130 -131 -132 -133 -134 -135 -136 -137 -138 -139 -
140 -141 -142 -143 -144 -145 -146 -147 -148 -149 -150 -151 -152 -153 -154 -155 -156 -157 -158 -159 -160 -161 -162 -163 -
164 -165 -166 -167 -168 -169 -170 -171 -172 -173 -174 -175 -176 -177 -178 -179 -180 -181 -182 -183 -184 -185 -186 -187 -
188 -189 -190 -191 -192 -193 -194 -195 -196 -197 -198 -199 -200 -201 -202 -203 -204 -205 -206 -207 -208 -209 -210 -211 -
212 -213 -214 -215 -216 -217 -218 -219 -220 -221 -222 -223 -224 -225 -226 -227 -228 -229 -230 -231 -232 -233 -234 -235 -
236 -237 -238 -239 -240 -241 -242 -243 -244 -245 -246 -247 -248 -249 -250 -251 -252 -253 -254 -255 -256 -257 -258 -259 -
260 -261 -262 -263 -264 -265 -266 -267 -268 -269 -270 -271 -272 -273 -274 -275 -276 -277 -278 -279 -280 -281 -282 -283 -
284 -285 -286 -287 -288 -289 -290 -291 -292 -293 -294 -295 -296 -297 -298 -299 -300 -301 -302 -303 -304 -305 -306 -307 -
308 -309 -310 -311 -312 -313 -314 -315 -316 -317 -318 -319 -320 -321 -322 -323 -324 -325 -326 -327 -328 -329 -330 -331 -
332 -333 -334 -335 -336 -337 -338 -339 -340 -341 -342 -343 -344 -345 -346 -347 -348 -349 -350 -351 -352 -353 -354 -355 -
356 -357 -358 -359 -360 -361 -362 -363 -364 -365 -366 -367 -368 -369 -370 -371 -372 -373 -374 -375 -376 -377 -378 -379 -
380 -381 -382 -383 -384 -385 -386 -387 -388 -389 -390 -391 -392 -393 -394 -395 -396 -397 -398 -399 -400 -401 -402 -403 -
404 -405 -406 -407 -408 -409 -410 -411 -412 -413 -414 -415 -416 -417 -418 -419 -420 -421 -422 -423 -424 -425 -426 -427 -
428 -429 -430 -431 -432 -433 -434 -435 -436 -437 -438 -439 -440 -441 -442 -443 -444 -445 -446 -447 -448 -449 -450 -451 -
452 -453 -454 -455 -456 -457 -458 -459 -460 -461 -462 -463 -464 -465 -466 -467 -468 -469 -470 -471 -472 -473 -474 -475 -
476 -477 -478 -479 -480 -481 -482 -483 -484 -485 -486 -487 -488 -489 -490 -491 -492 -493 -494 -495 -496 -497 -498 -499 -
500 -501 -502 -503 -504 -505 -506 -507 -508 -509 -510 -511 -512 -513 -514 -515 -516 -517 -518 -519 -520 -521 -522 -523 -
524 -525 -526 -527 -528 -529 -530 -531 -532 -533 -534 -535 -536 -537 -538 -539 -540 -541 -542 -543 -544 -545 -546 -547 -
```

5부터 시작해서 -1 된 결과가 끝도 없이 출력되다가 오류를 뱉어내고 종료됩니다...

만약 내가 출력하고 싶은 것이 5 4 3 2 1 까지라면 어떤 코드를 추가하면 좋을까요~?

## 4. 재귀함수란?

```
1  #include <stdio.h>
2
3  void recursion(int i) {
4      if (i <= 0) return; //기저 조건
5
6      printf("%d ", i); //i를 출력하고
7
8      recursion(i - 1); //재귀 호출
9  }
10 int main() {
11     recursion(5);
12 }
```

i가 0이하일 때부터는 더 이상 i를 출력하지도 말고  
recursion(i-1)을 호출하지도 말고 그냥 함수를 끝내자!

-> 4번 줄에 `if (i <= 0) return;`을 추가해준다.

출력 결과 : 5 4 3 2 1

이렇게 재귀 함수가 필요한 만큼만 돌고 멈출 수 있도록  
컷트해주는 역할을 하는 코드를

“기저 조건(basis)”이라고 합니다!

기저 조건은 함수의 제일 위쪽에 적어줍니다.

## 4. 재귀함수란?

아까와는 조금 달라진 예시이다.

```
1  #include <stdio.h>
2
3  void recursion(int i) {
4      if (i <= 0) return; //기저 조건
5
6      recursion(i - 1); //재귀 호출하고
7
8      printf("%d ", i); //i를 출력
9  }
10 int main() {
11     recursion(5);
12 }
```

출력 결과 : 1 2 3 4 5

- 아까 -

```
1  #include <stdio.h>
2
3  void recursion(int i) {
4      if (i <= 0) return; //기저 조건
5
6      printf("%d ", i); //i를 출력하고
7
8      recursion(i - 1); //재귀 호출
9  }
10 int main() {
11     recursion(5);
12 }
```

출력 결과 : 5 4 3 2 1

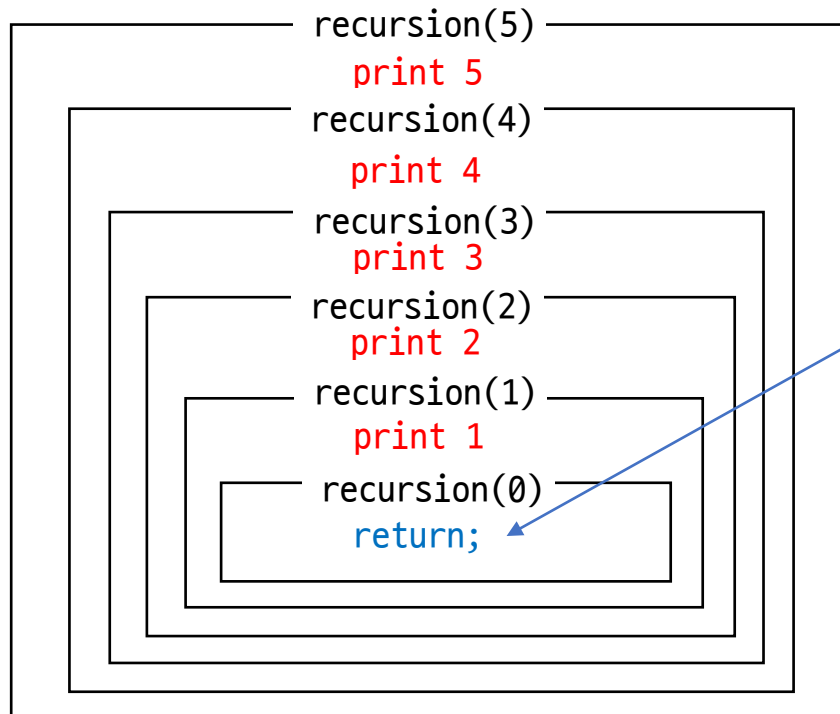
왜 둘이 다른 결과가 출력되는지 알겠나요..?

```

1  #include <stdio.h>
2
3  void recursion(int i) {
4      if (i <= 0) return; //기저 조건
5
6      printf("%d ", i); //i를 출력하고
7
8      recursion(i - 1); //재귀 호출
9  }
10 int main() {
11     recursion(5);
12 }

```

출력 : 5 4 3 2 1



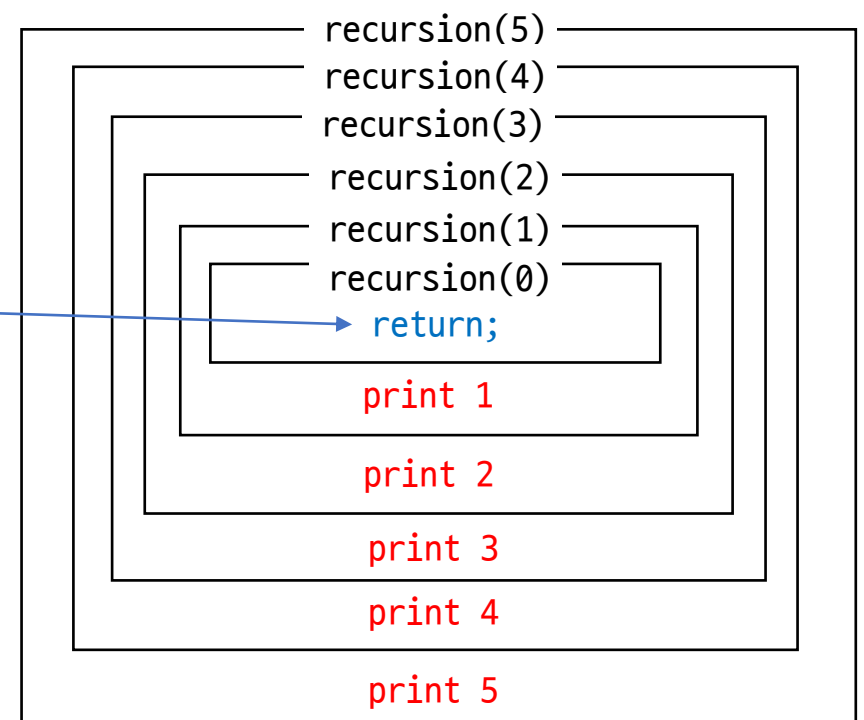
두 예제가 돌아가는  
과정을 그림으로 나타냄

```

1  #include <stdio.h>
2
3  void recursion(int i) {
4      if (i <= 0) return; //기저 조건
5
6      recursion(i - 1); //재귀 호출하고
7
8      printf("%d ", i); //i를 출력
9  }
10 int main() {
11     recursion(5);
12 }

```

출력 : 1 2 3 4 5



함수 제일 안쪽에서  
return;되는 부분이  
바로 4번줄 기저 조건  
에 걸린 순간이다.



## 4. 재귀함수란?

재귀함수를 이용하는 다른 예제를 하나만 더 볼게요.

왜냐하면 피피티랑 안 친한 강사가 도형 그리기가 너무 힘들기 때문이에요ㅎ

지난 시간에 배운 피보나치 수열을 배열이 아닌 재귀함수로 표현할 수 있어요!

- 배열 version -

```
1  #include <stdio.h>
2
3  int fibo[10];
4  int main() {
5      fibo[1] = 1;
6      fibo[2] = 1;
7      for (int i = 3; i <= 5; i++) {
8          fibo[i] = fibo[i - 1] + fibo[i - 2];
9      }
10     printf("%d\n", fibo[5]);
11 }
```

- 재귀 함수 version -

```
1  #include <stdio.h>
2
3  int fibo(int i) {
4      if (i == 1 || i == 2) return 1; //기저 조건
5
6      return fibo(i - 1) + fibo(i - 2);
7  }
8  int main() {
9      printf("%d\n", fibo(5));
10 }
```

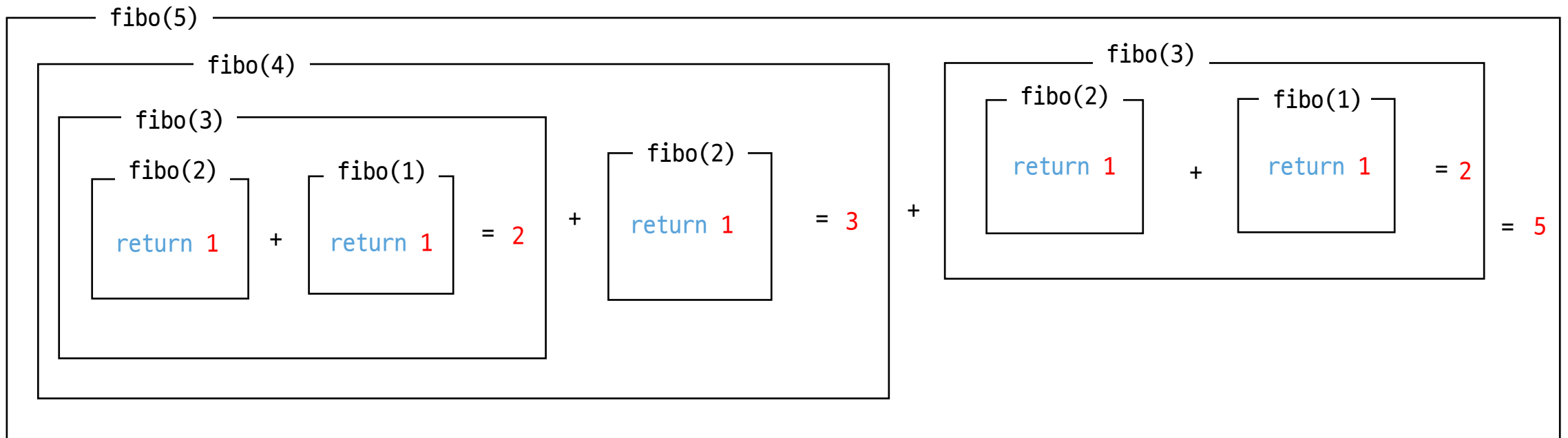
출력 결과 : 5

1 1 2 3 5.. 이렇게 가는 피보나치 수열의 n번째 수를 구하는 코드입니다!

출력 결과 : 5

```
1  #include <stdio.h>
2
3  int fibo(int i) {
4      if (i == 1 || i == 2) return 1; //기저 조건
5
6      return fibo(i - 1) + fibo(i - 2);
7  }
8  int main() {
9      printf("%d\n", fibo(5));
10 }
```

점화식이 메인같지만 결국 리턴이 되는 근본적인 숫자는 기저조건에서 발생한다는 점을 눈여겨보세요!



## 5. 문제 풀이

---

해설 보기를 좋아하는 강사는 마음 같아서는 모든 문제의 해설을 넣고 싶었지만 체력의 한계로 2문제만 넣었다.

B0J 2257 화학식량

# 2257번 화학식량

0. 문제 요약 : 분자식 (ex  $\text{CH}(\text{CO}_2\text{H})_3$ )이 주어졌을 때 이 화학식의 화학식량을 구하라.
1. 처음 생각 :  $\text{CH}_2\text{O}$  정도의 간단한 식이라면 for문을 돌면서 각 원자의 원자량을 최종 sum에 더해 나가면 된다.
2. 문제점 :  $\text{CH}(\text{CO}_2\text{H})_3$ 의 경우 괄호의 화학식량을 통째로 3배 해서 전체 화학식량에 더해야 한다. 단순히 for문으로는 지나쳐온 괄호의 화학식량을 따로 찾아내어 3배하기가 쉽지 않다.
3. 접근 : 이번주는 재귀함수를 배웠다. 함수를 만들어보자. 어떤 함수가 필요할까?
4. 아이디어 : 괄호 단위로 화학식량을 계산해서 리턴해주는 함수를 만들자!
5. 해결 : 재귀 함수 설계하기

# 2257번 화학식량

## 5. 해결 : 재귀 함수 설계하기 - 준비물

### [전역변수]

입력 받는 화학식 -> 알파벳과 숫자가 섞여있으니 char배열로 선언! -> `char chemi[105];` //모든 칸 `\0`로 초기화  
배열을 탐색할 인덱스 `i` -> `int i;` //0으로 자동 초기화

### [함수]

```
int sol()
{
    int m = 0; // 괄호 단위의 화학식량을 저장. 리턴의 대상.
    // 구현 내용
    return m;
}
```

# 2257번 화학식량

```
1  #include <stdio.h>
2  char chemi[105]; //입력 받는 화학식 배열
3  int i;           //배열을 순회할 인덱스
4
5  int sol() { // 괄호 단위로 화학식량을 계산한다.
6      int m = 0;
7
8      while (1) //반복문으로 chemi[i]를 살펴본다. while문 밖에서 return m
9      {
10         if (chemi[i] == 0) break; //문자열이 끝나면 더이상 m증가 나
11
12         if (chemi[i] == '(') { //여는 괄호를 만나면
13             i++;               //괄호 다음 인덱스로 이동.
14             m += sol();        //재귀 호출! 방금 만난 괄호의 화학식량을 구해와서 더한다.
15         }
16
17         if (chemi[i] == ')') { //닫는 괄호를 만나면
18             i++;               //괄호 다음 인덱스로 이동.
19             if ('0' < chemi[i] && chemi[i] <= '9') { //괄호 옆에 숫자가 있다면 ex (H)2
20                 m = m * (chemi[i] - '0');           // (괄호 안의 화학식량) * (옆 숫자)
21                 i++;                                 //숫자 다음 인덱스로 이동
22             }
23
24             //괄호 옆에 숫자가 있든 없든 m은 괄호의 총 화학식량을 가지고 있다.
25             return m; //지금까지의 질량을 리턴한다.
26         }
27     }
28 }
```

여기까지가 함수가 시작하고 끝나는 조건들이다.  
기저조건을 떠올려주면 된다.

재귀 함수를 설계하는 데에 있어서 가장 중요한  
부분이다.

이 밑으로는 화학식량을 계산하는 과정을 구현한다.

인덱스 다루는 부분이 조금 복잡하다..  
나도 이 코드 짜고나서 F11 계속 눌러봤다.

```

27     if (chemi[i] == 'C') { // C를 만났다!
28         i++;
29         if ('0' < chemi[i] && chemi[i] <= '9') { // C 옆에 숫자가 있다면
30             m += 12 * (chemi[i] - '0'); // (C의 원자량 12) * (옆의 숫자)
31             i++; // 숫자 다음 인덱스로 이동
32         }
33
34         // C 옆에 숫자가 없다면
35         else m += 12; // 그냥 12만 더해주기
36     }
37     else if (chemi[i] == 'H') { // C랑 똑같다
38         i++;
39         if ('0' < chemi[i] && chemi[i] <= '9') {
40             m += 1 * (chemi[i] - '0');
41             i++;
42         }
43         else m += 1;
44     }
45     else if (chemi[i] == 'O') { // C랑 똑같다
46         i++;
47         if ('0' < chemi[i] && chemi[i] <= '9') {
48             m += 16 * (chemi[i] - '0');
49             i++;
50         }
51         else m += 16;
52     }
53 }
54 // 널문자를 만나서 while문을 break 했다면
55 return m; // 지금까지 구한 화학식량을 리턴해서 프린트 하자.
56 }
57 int main(void) {
58     scanf("%s", chemi);
59
60     printf("%d", sol());
61 }

```

화학식량 계산하기.

지금까지 앞에서 봤던 재귀함수 상자 그림을 직접 그려보고 이해해보자. 문제에 나와있는 (H)<sub>2</sub>(O)와 CH(CO<sub>2</sub>H)<sub>3</sub>으로 실험해보면 충분하다.

3주차에서 배운 디버깅도 적극적으로 활용해보자.



B0J 11729

하노이 탑의 이동순서

# 11729번 하노이 탑의 이동순서

0. 문제 요약 : 1번 기둥에 있는 원판들을 3번 기둥으로 옮겨라. 단 원판은 항상 크기순으로 쌓여야 한다.
1. 처음 생각 : 아무 생각이 안 난다.
2. 그 다음 생각 : 예제처럼 원판 3개를 옮기는 그림을 그려본다. 3개 주제에 많이 복잡하다. 원판 20개면...어휴
3. 접근 : 역시나 함수를 만들자. 원판 개수를 입력하면 옮기는 횟수를 알아서 구해오는 똑똑한 함수가 있을 거라고 무작정 믿어보자.
4. 정보 : 사실 재귀함수의 근본적 파워는 일단 지금은 값을 모르겠지만 이 함수가 알아서 그 값을 구해올 것이라는 믿음에서 출발한다. 피보나치 함수를 떠올려보자. 나는 5번째 피보나치 수도 모르고 4번째도 모르고 3번째도 모른다. 하지만 fibo함수가 알아서 4번째 피보나치 수를 잘 구해올 것이라는 믿음을 가지고 fibo(5)에서 fibo(4) + fibo(3)을 리턴한다. fibo함수는 기저조건이 있기때문에 적절한 순간에 적절한 수를 리턴하고 값이 쌓여서 적절한 값을 찾아낸다.
5. 아이디어 : 일단 알아서 횟수를 구해오는 함수라고 이름 붙이고 '필요한 점화식'과 '적절한 기저조건'을 만들자.

# 11729번 하노이 탑의 이동순서

if N이 1이라면? 즉 원판 1개짜리 탑을 1번 기둥에서 3번 기둥으로 이동시키려면?  
그냥 이동시키면 된다○○

if N이 2라면 ?

맨 위 원판을 2번 기둥으로 이동시키고 맨 아래 원판을 3번 기둥으로 이동시킨 뒤  
2번 기둥에 있는 원판을 다시 3번 기둥으로 이동시킨다.

if N이 3이라면?

맨 아래 원판을 제외한 원판들을 2번 기둥으로 옮긴다. 방법은? N이 2일 때 원판 옮긴 방법을 쓰면 된다.

맨 아래 원판을 3번 기둥으로 옮긴다. 2번 기둥에 있는 원판들을 3번 기둥으로 옮긴다. 방법은? 아까와 같다.

if N이 N이라면?

맨 아래 원판을 제외한 N-1개 원판들을 2번 기둥으로 옮긴다. 방법은? N-1개 원판을 옮긴 방법을 쓰면 된다.

맨 아래 원판을 3번 기둥으로 옮긴다. 2번 기둥에 있는 N-1개 원판들을 3번 기둥으로 옮긴다. 방법은? 아까와 같다.

# 11729번 하노이 탑의 이동순서

```
1  #include <stdio.h>
2
3  int K; // 원판을 옮긴 횟수
4
5  // 원판 n개를 옮길때 필요한 이동 횟수를 알아서 구해주는 함수
6  void hanoi(int n) {
7
8      if (n == 1) // 원판 1개짜리 하노이 탑이라면
9      {
10         //그냥 출발 기둥에서 목적지 기둥으로 옮기면 끝
11         K++; //옮긴 횟수 1 증가
12         return;
13     }
14
15     // 여러개짜리 하노이 탑이라면
16     hanoi(n - 1); // 출발 기둥의 원판 n-1개를 경유 기둥으로 옮기고
17
18     K++; // 마지막 원판을 목적지 기둥으로 옮긴뒤 (옮긴 횟수 1 증가)
19
20     hanoi(n - 1); // 경유 기둥에 쌓여있는 원판 n-1개를 목적지 기둥으로 이동
21 }
```

원판  $n$ 개를 출발기둥(from)에서 목적지기둥(to)로 옮기기 위한 횟수를 구해주는 함수이다.

(Line 8~13)기저조건. 원판 0개짜리 탑까지 파고들지 않게 원판 1개에서 컷트 해준다.

(Line 16) 출발 기둥의 원판  $n-1$ 개를 경유 기둥으로 옮겨 놓기 위한 이동 횟수는  $\text{hanoi}(n-1)$  함수가 알아서  $K$ 에 체크해 놓을 거라고 믿는다.

(Line 18) 출발 기둥에 남아있는 제일 큰 원판 1개를 목적지 기둥에 옮기면 횟수  $K$ 는 1 증가.

(Line 20) 경유 기둥에 뒀던  $n-1$ 개의 원판을 목적지 기둥으로 옮기는 이동 횟수는 역시나  $\text{hanoi}(n-1)$  함수가 알아서  $K$ 에 체크해 놓을 거라고 믿는다.

뭐 하는 것도 없으면서 믿는다고만 하는 것 같지만 돌려보면 답이 잘 나온다. 이게 바로 재귀함수의 힘이다. 잘 몰라도 대충 잘 되기를 간절히 바라면 저절로 된다! 나도 이 함수는 그림 그려보라고 하지는 않을 거다. 애는 좀 너무 많다.

# 11729번 하노이 탑의 이동순서

```
23 //어디에서 어디로 옮겼는 지 출력해주는 함수
24 // 원판 개수, 출발 기둥, 목적지 기둥, 경유 기둥
25 void hanoi2(int n, int from, int to, int by) {
26
27     if (n == 1) // 원판 1개짜리 하노이 탑이라면
28     {
29         //그냥 출발 기둥에서 목적지 기둥으로 옮기면 끝
30         printf("%d %d\n", from, to);
31         return;
32     }
33
34     // 여러개짜리 하노이 탑이라면
35     hanoi2(n - 1, from, by, to); // 출발 기둥의 원판 n-1개를 경유 기둥으로 옮기고
36
37     printf("%d %d\n", from, to); // 마지막 원판을 목적지 기둥으로 옮긴뒤
38
39     hanoi2(n - 1, by, to, from); // 경유 기둥에 쌓여있는 원판 n-1개를 목적지 기둥으로 이동
40 }
41
42 int main() {
43     int N;
44     scanf("%d", &N);
45
46     // 원판 옮긴 횟수 출력
47     hanoi(N);
48     printf("%d\n", K);
49
50     //N개의 원반을 1번 기둥에서 3번 기둥으로 옮기고싶다. 옮기는 과정 출력
51     hanoi2(N, 1, 3, 2);
52 }
```

원판  $n$ 개를 출발기둥(from)에서 목적지기둥(to)로 옮기는 과정을 출력 해주는 함수이다.

hanoi2함수의 구조는 hanoi랑 완전히 동일하다. K++부분을 print로 바꿔주기만 하면 된다. 왜냐?

아까  $n-1$ 개 옮기는 횟수는 hanoi( $n-1$ )이 알아서 구해주겠지 ~ 하고 넘었갔었다.

하지만 엄밀히 말하면  $n-1$ 개를 한꺼번에 옮길 수는 없다.

결국 실제로 원판을 옮기는 행위는

1. 맨 아래 원판을 옮길때
2. 원판 1개짜리 탑을 옮길때

이 두가지 순간에만 일어난다. 우리는 그 두 순간에만 적절히 K++을 해줬기에 hanoi함수가 잘 작동해줬던 것이다.

우리의 믿음은 근거 있는 믿음이었던 것이다!

그리하여 원판 옮기는 과정을 출력하는 함수는 기존 함수의 K++자리에서 옮긴 경로를 print해주면 된다.

# 과제

# 과제

– 반복문이 먼저 떠오르더라도 재귀 함수로 풀어보자

B0J 10870번 피보나치 수 5

B0J 9095번 1,2,3 더하기

B0J 16076번 휴식이 필요해

B0J 2609번 최대공약수와 최소공배수

B0J 2257번 화학식량

B0J 11729번 하노이 탑 이동 순서

B0J 1914번 하노이 탑

## 6. 부록

---



# # Stack-Overflow

앞 슬라이드의 함수 그림을 잘 보면

recursion(5)가 끝나기 전에

recursion(4)가 들어오고

recursion(3)이 들어오는 식으로

함수가 겹겹이 쌓여나가죠?

끝날 때는 제일 나중에 들어왔던 recursion(0)부터  
들어온 역순으로 함수가 끝나죠?

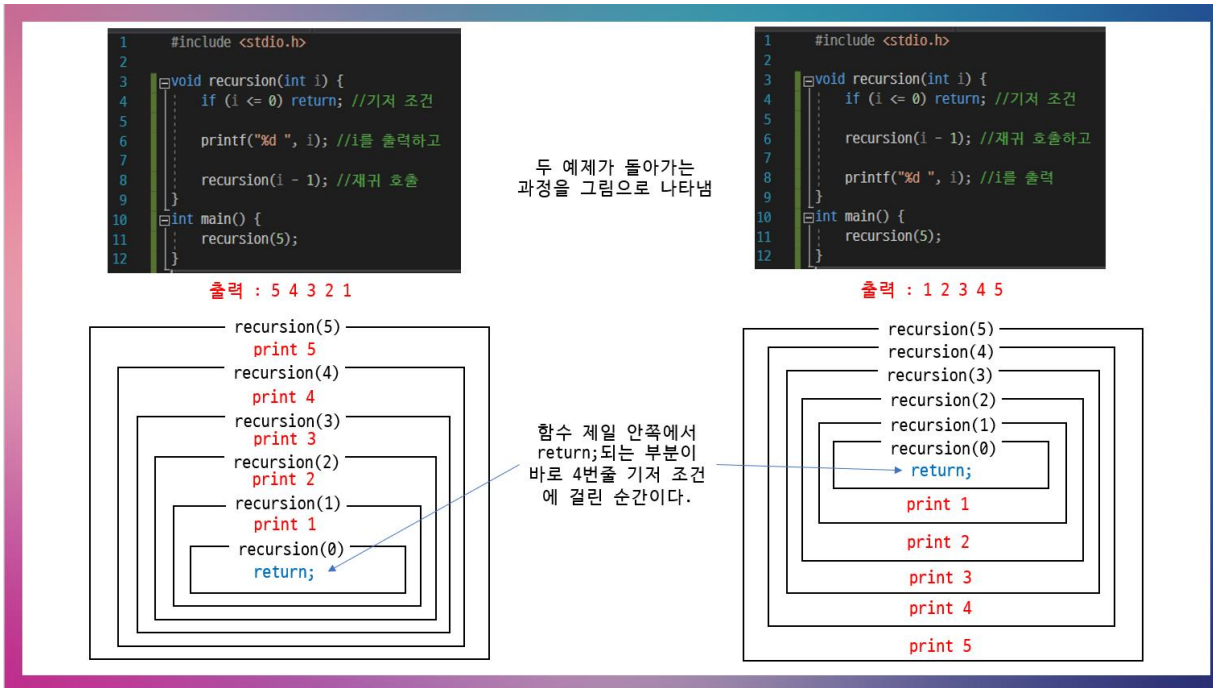
이런 걸 스택(stack)이라고 해요!

맨 마지막에 들어온 것이 제일 먼저 나가는 구조인데  
자세하게는 나중에 배울거예요.

만약 기저조건을 없었을 때처럼 함수를 끝내는 버튼이 없다면  
끊임없이 함수가 쌓여가다가 어느 시점에서 터져버릴거예요.

이를 Stack-Overflow 라고 합니다!

보통은 run-time error로 만나게 될 거예요.



# # include <stdio.h>

우리가 흔히 쓰는 printf, scanf도 다 함수이다.

이 함수들의 정의와 선언은 어디에 되어있을까...?

바로 제목으로 커다랗게 쓰여있는 <stdio.h> 헤더파일이 관련이 있다!

자세한 구조는 창소프때 배우도록 하자.

stdio.h는 standard input output의 약자로

입출력에 관한 함수 printf, scanf등을 담고있다.

이외에도 <algorithm>, <string>, <string.h> 등 다양한 헤더파일이 존재한다.

일부러 .h가 있는 헤더와 없는 헤더를 섞어서 나열해봤다.

이들의 차이점은 .h가 있는 헤더는 C 함수가 들어있는 헤더이고

.h가 없는 헤더는 C++ 함수가 들어있는 헤더이다.

고로 string과 string.h는 아예 다른 헤더이다.

요즘(?)에는 헤더파일에 .h를 안 쓰도록 하는 추세라고 한다.

.h 대신 <cstdio>, <cstring> 등 앞에 c를 붙여서 나타낼 수 있다.

어디서 이런 헤더를 마주쳐도 당황하지 말자 !

# # define sum(x, y) x + y

함수의 종류가 사실 한 가지 더 있다.  
바로 저 제목과 같은 매크로 함수이다.

```
1  #include <stdio.h>
2  #define sum(x, y) x + y
3
4  int main(void) {
5      printf("%d", sum(1,2));
6  }
```

출력 결과 : 3

하지만 잘 안 쓰는 이유는? 위험하기 때문이다.

```
1  #include <stdio.h>
2  #define mul(x, y) x * y
3
4  int main(void) {
5      printf("%d", mul(5,2));
6  }
```

출력 결과 : 10

여기까지는 문제가 없다.

```
1  #include <stdio.h>
2  #define mul(x, y) x * y
3
4  int main(void) {
5      printf("%d", mul(2+3,2));
6  }
```

출력 결과 : 8

??

x \* y 에 2+3 \* 2 이렇게 들어간 것이다.

연산자 우선순위에 의해 2 + 3\*2 = 8로 계산됨..

#define mul(x, y) (x)\*(y) 로 선언해야 사고를 막을 수 있다.

# # define MAX 100'000

## 입력

$n$ 과  $x_i$ 가 주어진다.  $n$ 은 10만 이하이고,  $x_i$ 는 정수이다.

지난주 과제였던 귀찮아 문제의 일부이다.  
입력이 최대 10만개까지 들어올 수 있다.  
이를 위해서 10만칸짜리 배열을 만들 것이다.

## 입력

첫째 줄에  $n$ 이 주어진다.  $n$ 은 90보다 작거나 같은 자연수이다.

그 다음 문제이다. 입력이 최대 90개까지 들어올 수 있다.  
이를 위해서 90칸짜리 배열을 만들 것이다.

=>

문제를 계속 풀다 보면 배열을 거의 매번 만들게 된다.  
이때 배열을 조금 덜 번거롭게 만드는 방법이 있다.

```
# define MAX 100'000
int arr[MAX];
int sum[MAX];
```

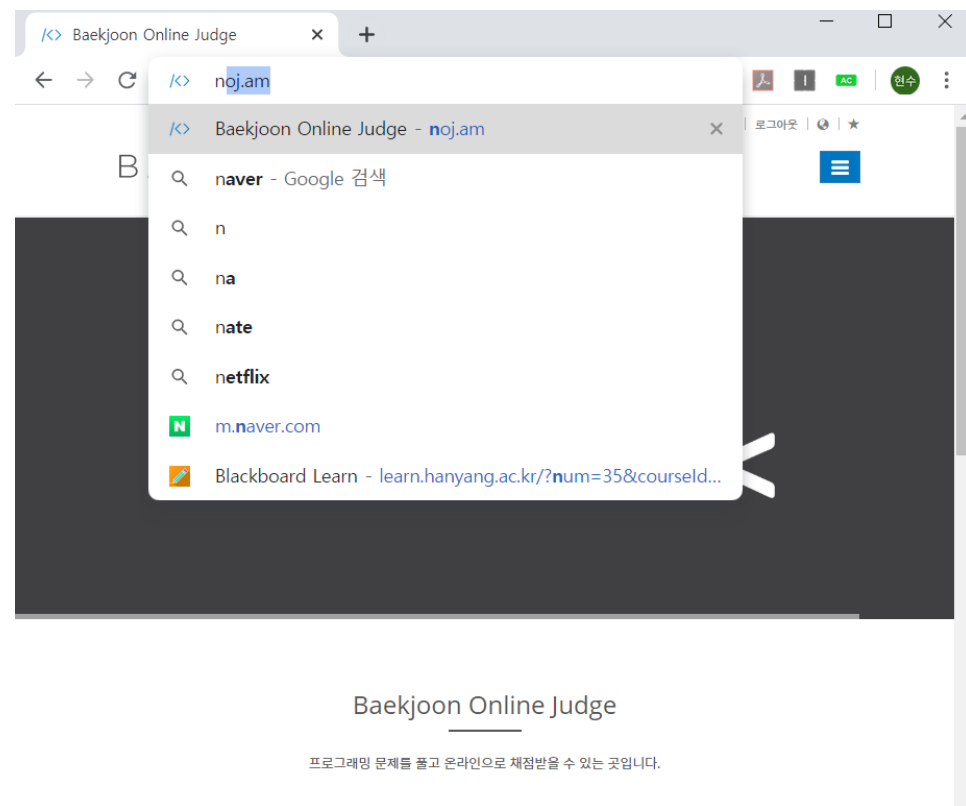
이렇게 선언해주면 된다!  
여러 개의 배열을 한꺼번에 선언할 수 있고  
다른 문제를 풀 때는 MAX 숫자만 바꿔주면 된다.

내가 0을 몇 개나 붙였는지 헷갈린다면?  
100'000처럼 중간에 작은 따옴표를 붙여줘도 된다!

# #noj.am/

주소창에 noj.am 을 입력하면 백준 사이트로 이동한다.

noj.am/16076 등 문제 번호도 입력하면 해당 문제로 이동한다.



끝

---

높임말과 반말을 넘나드는 정신없는 피피티를  
읽어주셔서 감사합니다