C언어 스터디
6.5주차

<재귀함수는 사실 DFS이고 스택입니다. (다소 날조)>

잘 생각해 오셨나요?

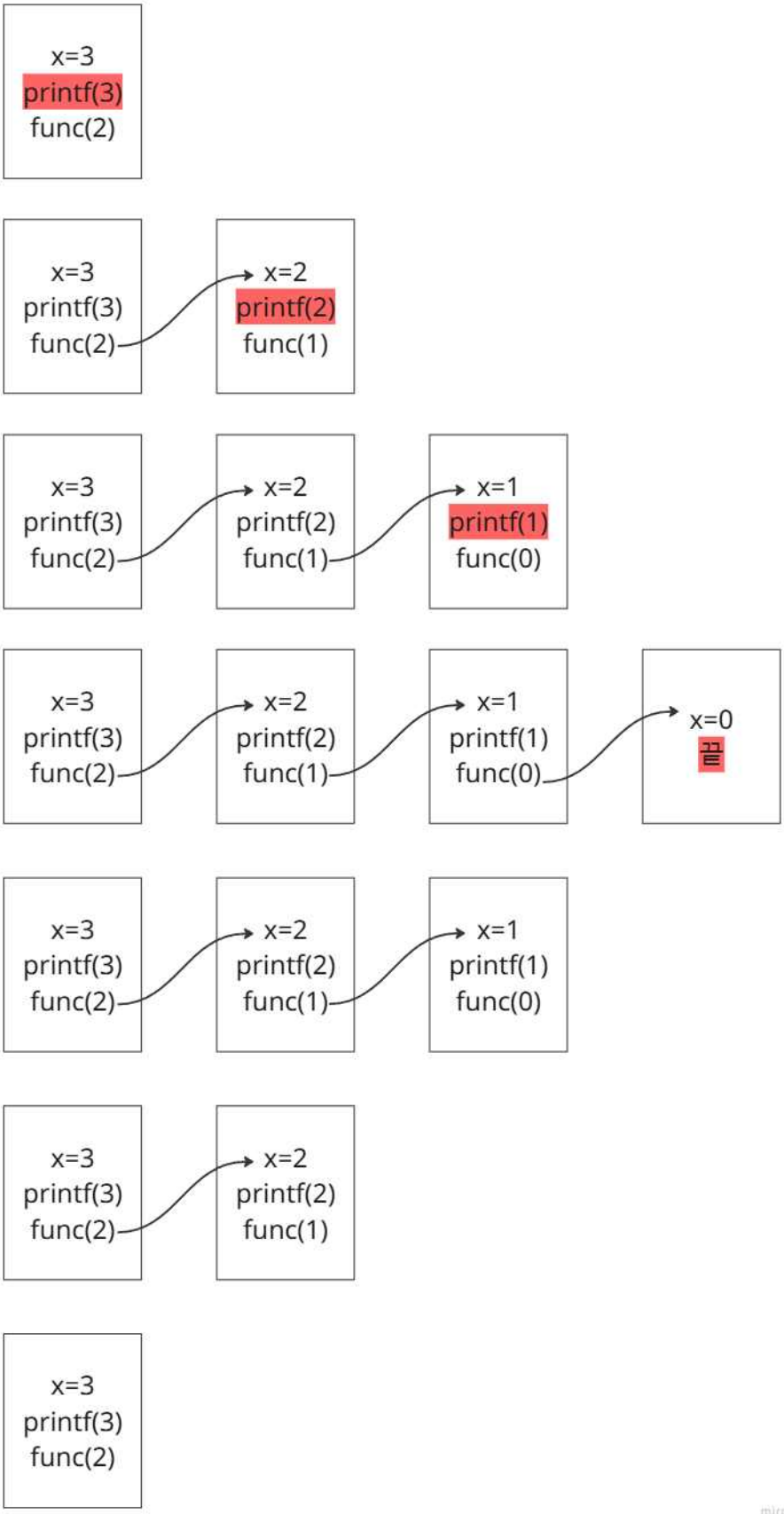
```c
#include<stdio.h>
void func(int x) {
	printf("A");
	func(x);
}
int main(void) {
	func(10);
} // 어떻게 될까요?
```

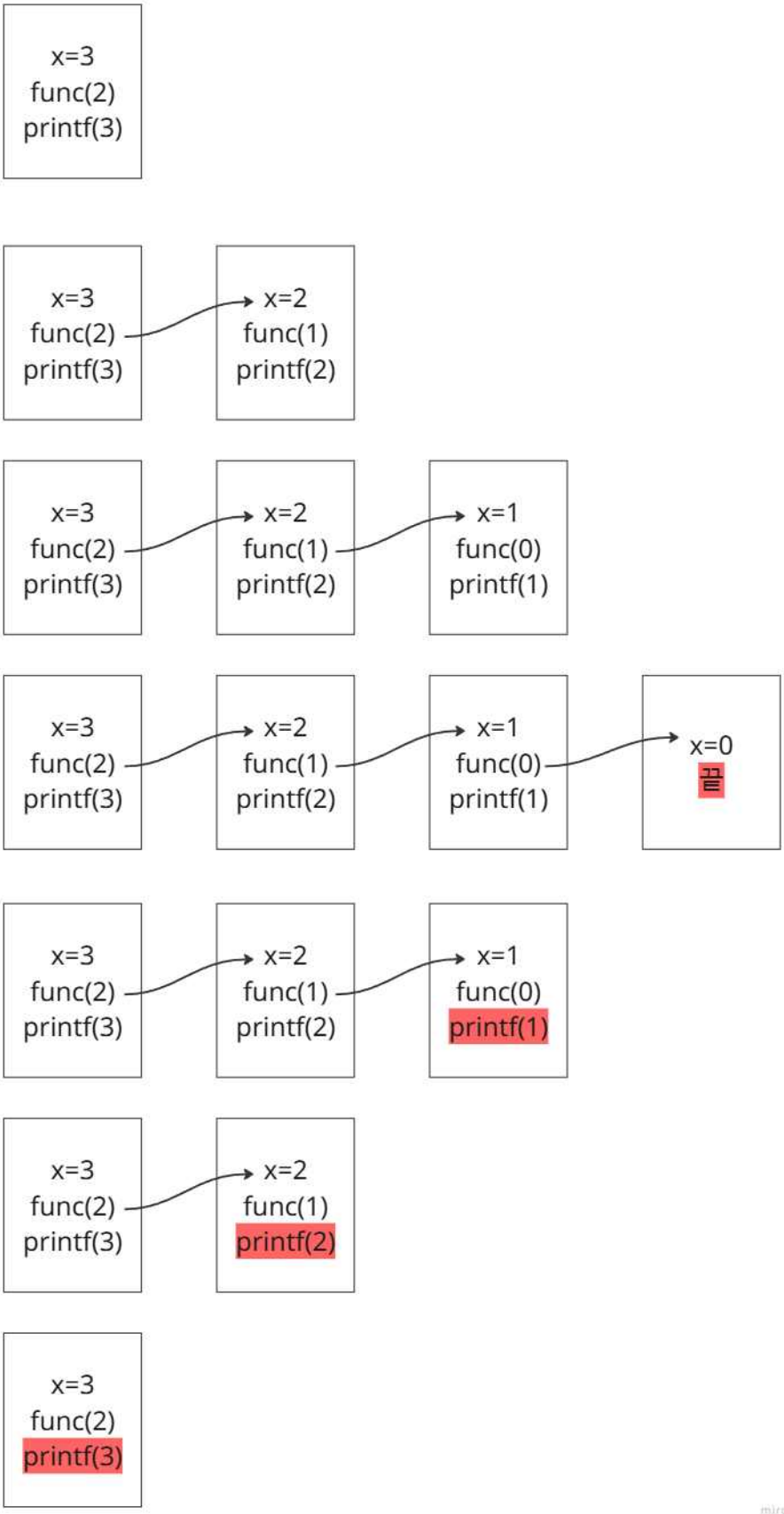
```c
#include<stdio.h>
void func(int x) {
	func(x);
	printf("A");
}
int main(void) {
	func(10);
} // 어떻게 될까요?
```

```c
#include<stdio.h>
void func(int x) {
	if (x == 0)return; //종단점이 있어야 하구요
	printf("%d ", x);
	func(x - 1); //재귀함수로 넘길 때 적어도 인자를 증감해야합니다.
}
int main(void) {
	func(3);
} // 어떻게 될까요?
```

```c
#include<stdio.h>
void func(int x) {
	if (x == 0)return; //종단점이 있어야 하구요
	func(x - 1); //재귀함수로 넘길 때 적어도 인자를 증감해야합니다.
	printf("%d ", x);
}
int main(void) {
	func(3);
} // 어떻게 될까요?
```

위의 출력 : 3 2 1
아래 출력 : 1 2 3
왜인지는 집에 가서 생각해 봅시다.

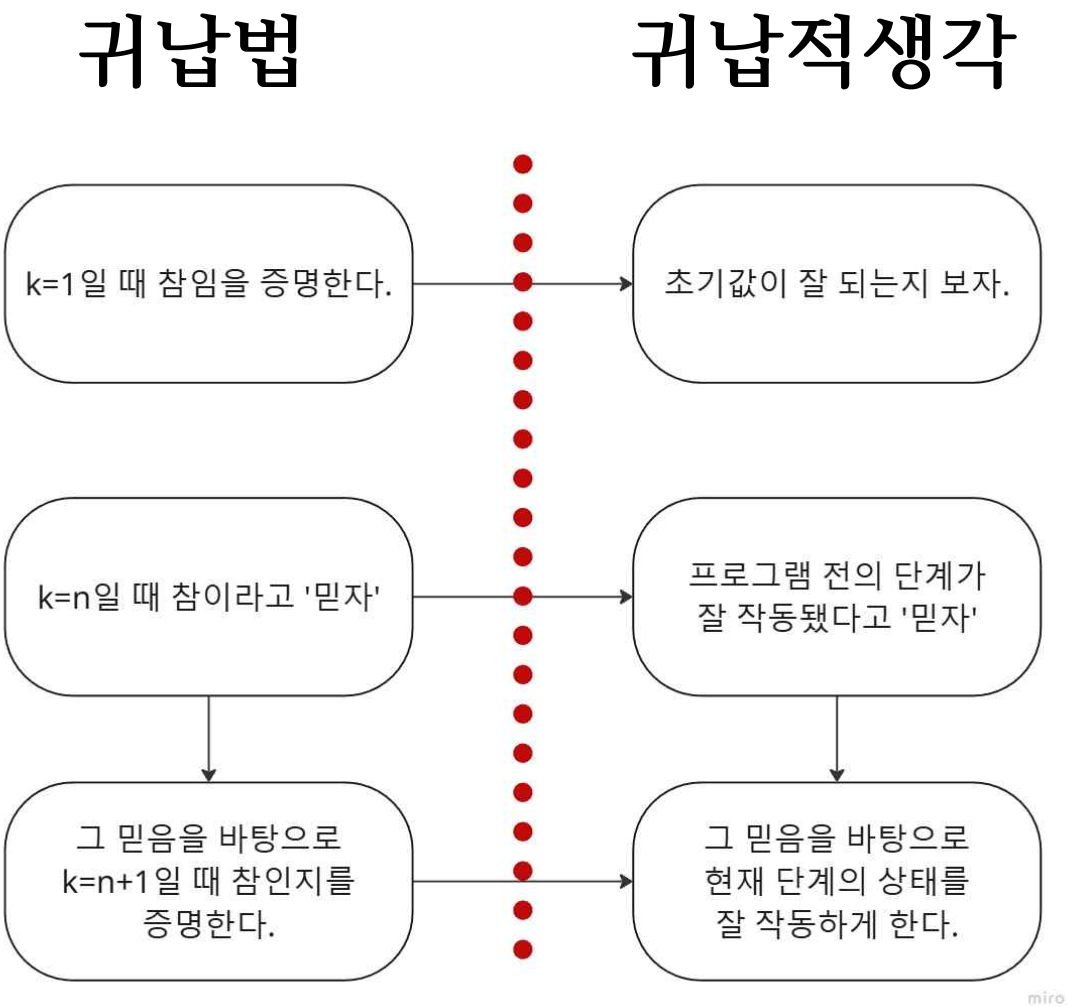

x=3
printf(3)
func(2)
x=3
printf(3)
func(2)
x=2
printf(2)
func(1)
x=3
printf(3)
func(2)
x=2
printf(2)
func(1)
x=1
printf(1)
func(0)
x=3
printf(3)
func(2)
x=2
printf(2)
func(1)
x=1
printf(1)
func(0)
x=0
끝
x=3
printf(3)
func(2)
x=2
printf(2)
func(1)
x=1
printf(1)
func(0)
x=3
printf(3)
func(2)
x=2
printf(2)
func(1)
x=3
printf(3)
func(2)
miro


x=3
func(2)
printf(3)
x=3
func(2)
printf(3)
x=2
func(1)
printf(2)
x=3
func(2)
printf(3)
x=2
func(1)
printf(2)
x=1
func(0)
printf(1)
x=3
func(2)
printf(3)
x=2
func(1)
printf(2)
x=1
func(0)
printf(1)
x=0
끝
x=3
func(2)
printf(3)
x=2
func(1)
printf(2)
x=1
func(0)
printf(1)
x=3
func(2)
printf(3)
x=2
func(1)
printf(2)
x=3
func(2)
printf(3)

귀납적 생각

연결리스트를 잘 하시게 된다면, 향후 다룰 재귀함수 등에 큰 도움이 될 겁니다. 아니? 사실 재귀함수를 잘하시면 연결리스트를 잘하게 될 겁니다. 아니? 사실 이 모든 걸 아우르는 하나의 능력이 있습니다.

연결리스트에서의 중요한 건 '귀납적인 생각'입니다. dp, 재귀 모두 '귀납적인 생각'이 중요하거든요. 사실 알고리즘을 배우실 때 가장 처음 만나고 가장 어려운 벽이 귀납적인 생각일 겁니다.



함수는 믿음입니다.

```
int main(void) {
        cout<<add(1,2);
} // 어떤 값이 나올 거 같으세요?
```

당연히 3이 나올 거라고 생각합니다.
왜냐하면 add라는 함수는 더하기를 한다고 "믿기" 때문입니다.

```
int main(void) {
        cout<<add(1,2);
} // 어떤 값이 나올 거 같으세요?
```

add의 입장에서도 마찬가지입니다.
x와 y가 더할 값으로 주겠지 라는 '믿음'으로 수를 받고,
더한 값을 잘 써 주겠지 라는 '믿음'으로 수를 뱉습니다.

```
int main(void) {
        cout<<factorial(4);
} // 어떤 값이 나올 거 같으세요?
```

아마 4!을 만들어 내는 함수인 거 같아요.
함수를 잘 짰을 테니, 아마 24를 뱉지 않을까요?
함수를 잘 만들었다는 믿음입니다.

```
int factorial(int x) {
        int tmp=factorial(x-1);
        return tmp*x;
}
```

이제 팩토리얼의 입장에서 봐 봅시다.
x!를 출력하고 싶으면, (x-1)!에 x를 출력합시다.
그럼 factorial(x-1)이 (x-1)!을 잘 뱉는다고 "믿어야"합니다.

```
#include<iostream>
using namespace std;
int factorial(int x) {
        int tmp=factorial(x-1);
        return tmp*x;
}
int main(void) {
        cout<<factorial(4);
} // 어떤 값이 나올 거 같으세요?
```

어때요 잘 돌아가나요?

아뇨 안 돌아갈 겁니다. 재귀함수에는 “종단점”이 중요해요

```
#include<iostream>
using namespace std;
int factorial(int x) {
        if(x==0)return 1;
        int tmp=factorial(x-1);
        return tmp*x;
}
int main(void) {
        cout<<factorial(4);
} // 어떤 값이 나올 거 같으세요?
```

네 이제 잘 나옵니다.

수학적 귀납법은

1. k=1일 때 가능한지 증명한다.
2. k=n일 때 가능하다고 ‘믿고’ k=n+1일 때 가능한지 증명한다.

입니다.

재귀함수는

1. 종단점을 잘 설정한다.
2. 자식 함수들이 가능하다고 ‘믿고’ 현재 함수를 잘 내뱉는다.

입니다.

귀납적인 생각은 훈련이 아주 많이 필요합니다. 가장 어려운 작업이 될 거예요.

그럴려면 문제를 많이 풀어 봐야 합니다. 파이팅.

문제

문제 두 개만 풀어봅시다.

https://www.acmicpc.net/problem/1991

```
#include<iostream>
using namespace std;
char arr[26][2]; // 인접 리스트입니다.
void preorder(int x); // 전위순회
void inorder(int x); // 중위순회
void postorder(int x); // 후위순회
int main(void) {
	ios::sync_with_stdio(0);
	cin.tie(0);
	cout.tie(0);
	for (int i = 0; i < 26; i++)for (int j = 0; j < 2; j++)arr[i][j] = '.';
	int N;cin >> N;
	for (int i = 0; i < N; i++) {
		char in;cin >> in;
		cin >> arr[in - 'A'][0] >> arr[in - 'A'][1];
	}
	preorder(0);
	cout << "\n";
	inorder(0);
	cout << "\n";
	postorder(0);
}
void preorder(int x) { // 전위순회
	cout << char(x + 'A'); // 앞에 있음
	if (arr[x][0] != '.')preorder(arr[x][0] - 'A'); //왼쪽
	if (arr[x][1] != '.')preorder(arr[x][1] - 'A'); //오른쪽
}
void inorder(int x) { // 중위순회
	if (arr[x][0] != '.')inorder(arr[x][0] - 'A');
	cout << char(x + 'A'); // 중간에 있음
	if (arr[x][1] != '.')inorder(arr[x][1] - 'A');
}
void postorder(int x) { // 후위순회
	if (arr[x][0] != '.')postorder(arr[x][0] - 'A');
	if (arr[x][1] != '.')postorder(arr[x][1] - 'A');
	cout << char(x + 'A'); // 뒤에 있음
}
```

https://www.acmicpc.net/problem/15649

```
#include<iostream>
using namespace std;
int N, M;    // dfs 함수에서 N과 M을 써야하기 때문에 전역변수로 합니다.
int vst[9]; // 쓴 숫자의 index에 표시해 줍니다.
int arr[9]; // 정답을 잘 저장해 줍시다.
void dfs(int h) {
    if (h == M) {
        for (int i = 0; i < M; i++)cout << arr[i] << " ";
        cout << "\n";
        return;
    }
    for (int i = 1; i <= N; i++) {
        if (vst[i])continue;
        vst[i] = 1;
        arr[h] = i;
        dfs(h + 1);
        vst[i] = 0;
    }
}
int main(void) {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cin >> N >> M;
    dfs(0);
}
```

위의 두 문제는 자력으로 다시한번 꼭 풀어주세요.

문제)

1. 27433 팩토리얼 2
   https://www.acmicpc.net/problem/27433
   https://github.com/sungjoonyoung/BOJ/blob/main/20000/27433.cpp
2. 1991 트리 순회
   https://www.acmicpc.net/problem/1991
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/1991.cpp
3. 15649 N과 M (1)
   https://www.acmicpc.net/problem/15649
   https://github.com/sungjoonyoung/BOJ/blob/main/10000/15649.cpp
4. 15651 N과 M (3)
   https://www.acmicpc.net/problem/15651

https://github.com/sungjoonyoung/BOJ/blob/main/10000/15651.cpp

5. 2251 물통 (이거 풀면 밥 사줌)
   https://www.acmicpc.net/problem/2251
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/2251.cpp
6. 2630 색종이 만들기 (이거 풀면 밥사줌)
   https://www.acmicpc.net/problem/2630
   https://github.com/sungjoonyoung/BOJ/blob/main/00000/2630.cpp