**자료구조**

자료구조란 프로그램에서 다룰 데이터를 정리하고, 하나의 단위로 묶는 다양한 구조를 말한다.

우리는 이전에 자료구조 중 하나를 배운 적이 있다. 배열이다.

같은 형태의 데이터들을 하나의 단위로 조직화한 것이 자료구조에 속한다.

배열을 통해 여러 문제들을 손쉽게 해결했을 것이다. 이처럼 알맞은 자료구조를 사용하면 문제를 해결하기 쉽다.

컴퓨터 프로그램이란 알맞은 자료구조를 선택하고 알고리즘을 구현한 것이라 할 정도로 자료구조는 중요하다.

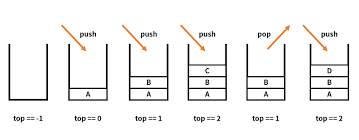
아래는 앞으로 배울 자료구조이다.

선형자료구조: 스택, 큐, 리스트 비선형 자료구조: 트리, 그래프

**스택(stack)**

스택(stack)의 사전적의미는 “쌓다”이다 사전적의미와 동일하게 스택은 데이터를 한 방향으로 쌓는 자료구조이다. 접시를 쌓아 올리는 것과 같다. 새로운 데이터를 추가하려면, 가장 위에 쌓으면 된다. 데이터 하나를 자료구조에서 삭제할 것이면 가장 위에 있는 데이터를 삭제하면 된다.

해당 구조는 후입선출구조이다. 가장 나중에 들어온 것이 먼저 나간다.



Top은 가장 위에 있는 데이터를 가리킨다. stack에서는 Top을 이용해 데이터를 관리한다.

이제 스택을 코드로 어떻게 구현하는 지 알아보자.

**<ADT>**

#include<stdio.h>

#define MAX\_STACK\_SIZE 100

int stack[MAX\_STACK\_SIZE];

int top = -1;

int IsEmpty() {//비었는 지 확인하는 함수

if (top < 0)

return true;

else

return false;

}

int IsFull() {//가득찼는 지 확인하는 함수

if (top >= MAX\_STACK\_SIZE - 1)

return true;

else

return false;

}

void push(int value) {//데이터 추가

if (IsFull() == true)

printf("스택이 가득 찼습니다.");

else

stack[++top] = value;

}

int pop() {//데이터 제거

if (IsEmpty() == true)

printf("스택이 비었습니다.");

else

return stack[top--];

}

int main() {

push(3);

push(5);

push(12);

printf("%d ", pop());

printf("%d ", pop());

printf("%d ", pop());

return 0;

}

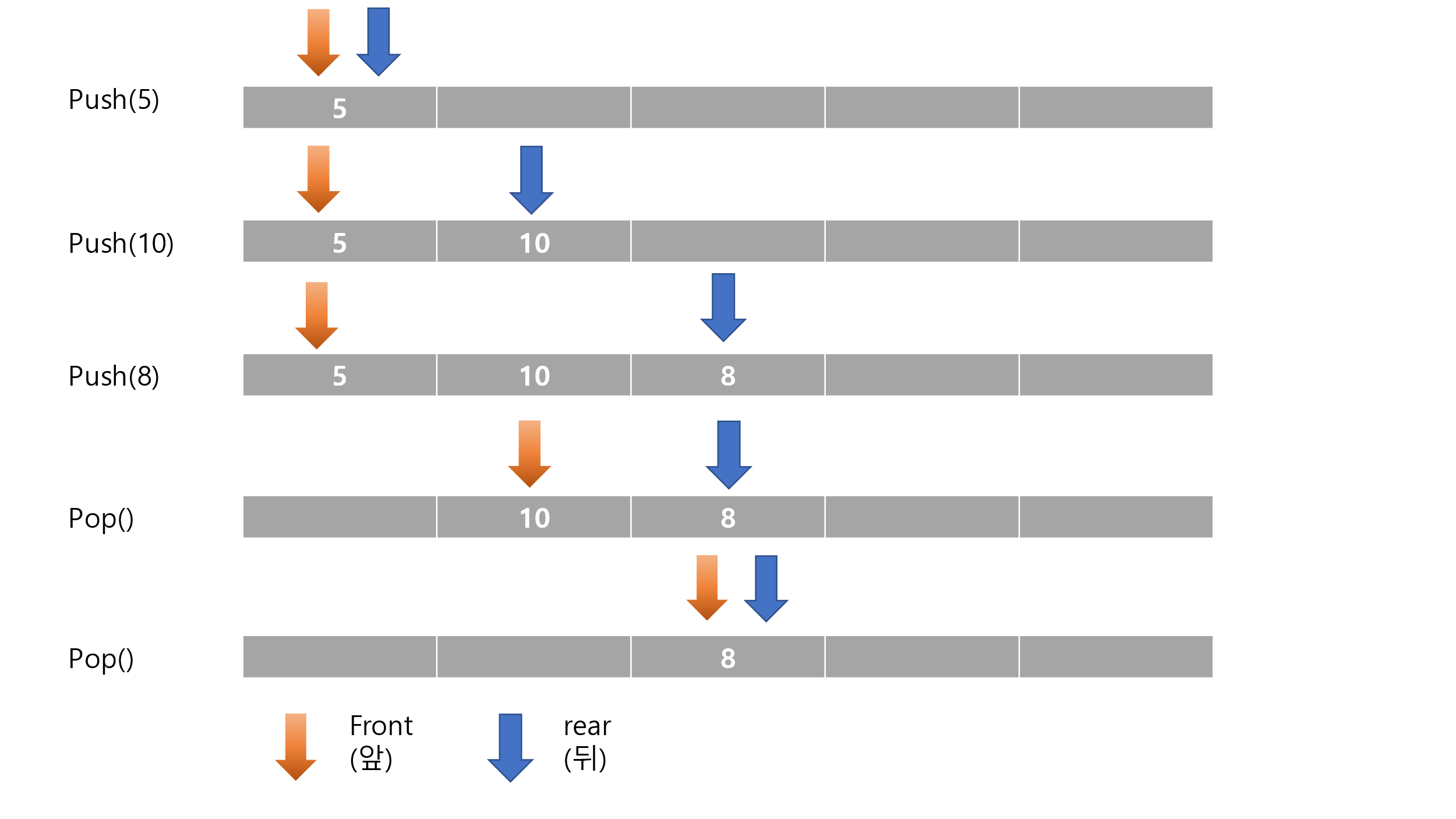
텍스트이(가) 표시된 사진

자동 생성된 설명

**큐(queue)**

스택과 달리, 큐는 먼저 들어온 데이터가 먼저 나간다. First in First out(선입선출) 구조이다.

줄을 서는 것과 같다. 데이터를 추가할 때는 맨 뒤에 추가되고, 데이터를 제거할 때는 앞에 있는 데이터부터 제거한다. 그래서 큐에서는 스택과 달리, 앞과 뒤를 가리키는 변수가 필요하다.



앞을 가리키는 front와 뒤를 가리키는 rear를 통해 queue 데이터를 관리한다.

하지만, queue는 단점이 있다. 데이터를 제거할 때마다. Front가 앞으로 증가하면서 데이터에 대한 공간이 하나씩 줄어든다. (queue는 front ~ rear 사이의 데이터만 취급하므로)

![텍스트, 시계이(가) 표시된 사진

자동 생성된 설명](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAYABgAAD/4RFKRXhpZgAATU0AKgAAAAgABwEaAAUAAAABAAAIgAEbAAUAAAABAAAIiAEoAAMAAAABAAIAAAE7AAIAAAAKAAAAYodpAAQAAAABAAAIkJydAAEAAAAIAAAAbOocAAcAAAgMAAAAdAAAAADsoITsoJXrr7wABMgVyPy7AAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGAAAAABAAAAYAAAAAEACpAAAAcAAAAEMDIxMJADAAIAAAAUAAAJDpAEAAIAAAAUAAAJIpKRAAIAAAADOTUAAJKSAAIAAAADOTUAAKAAAAcAAAAEMDEwMKABAAMAAAAB//8AAKACAAQAAAABAAABw6ADAAQAAAABAAABveocAAcAAAgMAAAJNgAAAAAyMDE4OjAxOjA1IDExOjIwOjA3ADIwMTg6MDE6MDUgMTE6MjA6MDcAHOoAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAD/4QscaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLwA8P3hwYWNrZXQgYmVnaW49J++7vycgaWQ9J1c1TTBNcENlaGlIenJlU3pOVGN6a2M5ZCc/Pg0KPHg6eG1wbWV0YSB4bWxuczp4PSJhZG9iZTpuczptZXRhLyI+PHJkZjpSREYgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyIvPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6eG1wPSJodHRwOi8vbnMuYWRvYmUuY29tL3hhcC8xLjAvIj48eG1wOkNyZWF0ZURhdGU+MjAxOC0wMS0wNVQxMToyMDowNy45NDk8L3htcDpDcmVhdGVEYXRlPjwvcmRmOkRlc2NyaXB0aW9uPjxyZGY6RGVzY3JpcHRpb24gcmRmOmFib3V0PSJ1dWlkOmZhZjViZGQ1LWJhM2QtMTFkYS1hZDMxLWQzM2Q3NTE4MmYxYiIgeG1sbnM6ZGM9Imh0dHA6Ly9wdXJsLm9yZy9kYy9lbGVtZW50cy8xLjEvIj48ZGM6Y3JlYXRvcj48cmRmOlNlcSB4bWxuczpyZGY9Imh0dHA6Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPjxyZGY6bGk+7KCE7KCV66+8PC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAb0BwwMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APpGiiikAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQB5t+0F/yQrxD/wBu3/pTFVfwF4Y3/BTRrbwubfQ7vVLG3a91G2hCzEFcs4IHzSckAk/Luz2wbH7QX/JCvEP/AG7f+lMVbPwvdYvhD4YeRgiLpcJLMcADYOTQB8//ABM8FX/wd8X6JrfgzXNTuZ9SmYbbmTfM8qlflYqB5ituxgj8816h+0VpOnz/AAmvNVudOtTqUDQJHcmNWliBlXKq+M45PHTmqq+IPBni/wAfR+LvEfiXRYdO0MPHo9jPfRb5Hzl7lk3ZGSAEXGflDYBxV7476hFqv7Pt1qFsHEV2LSaMOMMFaRGGR2ODQB2/w4/5JZ4U/wCwNZ/+iErpa5r4cf8AJLPCn/YGs/8A0QldLQAUUUUAFFFFABRRWJq/i/w5oDFNb1/TbBx/yzuLpEY/8BJyaANuivO7r46fD6CXyYNca+n7RWVpNKT9CFx+tQ/8LhN3/wAgXwH4wv17S/2Z5UZ+jM39KAPSqK81HxG8Zz/8enwr1Zh/03voYT+tZmsfFfxroX2A6n8L5IF1C9jsLbfrsJMk8mdiYCHGcHk4HqaAPXaK82PxA8fRc3HwovVH/TLV4JD+go/4WrrNvzqPwz8WRgfe+y26XGPphhmgD0mivNR8d/B8BxrcesaI3TGo6XKmP++Q1bmnfFPwLqpUWXivSyzfdSW5WJj+D4NAHXUVHFLHPCssLrJGwyrqcgj2NSUAFFFFABRRRQAV8+fstaPp8vhfVNVewtm1KG+MUV48IaSNfKX5VbqB8xyARnNfQdeDfstTR2/w51uaeRY4o9SZmdmwqgQoSST0FAFb4++E9A07wl/bd5q103ivz4/s0kl42ZyXAYLEW2xoF3EbQMYGSc893peg3Go/Ay0tPH8Frq2oRae8zNcKs+19j+W245+cIwG8HOc4PNQ/EL4feE/ib4VudWt3tJLxbdmtdYtZFYZQEgMynDr1BB6dsVifBI37/s73Lai0jR7bsWm85xCFIwPbdvoAl/Zg/wCSRSf9hKb/ANBSvY68c/Zg/wCSRSf9hKb/ANBSvY6ACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigDzb9oL/AJIV4h/7dv8A0pipnhnwzb+NPgF4f0a51G+srW406BZnsJVV5FC4KEsrfKT1GO2OmRXoGoabZaxYyWWqWkF7aSEeZb3MSyRvggjKsCDggH6iq+k+HtG0FZF0LSLDTFmIMi2VskIcjpnaBnGTQB5N/wAMq+CP+gpr/wD4EQf/ABmtf9oSKK1+BV/bxgIkcltHGvsJVwPyFerVj6r4V8P69cJNruh6bqUsa7EkvLOOZlXOcAsDgZPSgCl8OP8AklnhT/sDWf8A6ISulqvbW0Flaw21pFHBbwoI4ookCpGoGAqgcAADAAqtrOtab4f06TUNavoLG1j+9LO4UfQep9hyaANGobi4gtLd57uaOCKMZeSRgqqPUk8CvNP+FkeIvFzGH4Y+GpJ7Zjga3rAa3tQP7yJ9+QfTH0qW3+ER1q4S/wDiXrt14oulO5bQnyLKE/7MS9cep69xQBZ1L4zeFobtrHw+bzxNqI/5ddFgNx+JcfJj3yarC++LPibmx07SfB1m3R76T7ZdEdiFX5B9G5rv9N0rT9GsxaaTYW1jbr0itoljUfgABV2gDzT/AIVJfasN3jPx54g1jP37e2lFlbv9Y0/xra0n4T+BdEQfYfC+ns3/AD0uYvtD/XdJuNdjRQBXtbO2sofLtLaK3T+7FGFH5CrFFFABXm3xk/5kL/sc9O/9qV6TXm3xk/5kL/sc9O/9qUAek0UUUAIQGBBGQeCD3rC1LwT4X1gMNT8O6XdburS2cZb8GxkVvUUAebSfBHw3bSNN4XvdZ8MTk5LaTqDorH3Viwx7cUw+HviroPzaN4u07xFCvS11iz8l8eglj5J9zXplFAHmI+Lt1oBEfxF8I6r4fA4a+gT7ZaD3Micj6YJrtdB8WaB4ot/O8P6vaagoGWEEoZk/3l6r+IFbBAIIIyD1BriNf+EnhPXLgXsFi2jamp3R6jpL/Zpkb1yvBP1BoA7iivLi/wATvA4xKkPjrSE/ijAt9QjX6fdkx7fMfaui8KfEjw74vmezsLiW11SL/XaXfxmC5jPfKHrj/ZzigDr6+f8A9mWytdW+F3iPTL5BNb3V68M8W7G6N4VUjjkZGRX0BWFpvgzwxpF6t5pPhzSbC6QELPa2UUTqCMHDKoPIoA4nSPgB4X0YzRW+reIm0+4P7/TTqJS3mHo6oqlh9TXc6tbWum+C7+2tIo7a1t7CVI40UKkaCM4AHQACtqs7VdF0rXrVLXW9NtNSt0cSLFeQLMisAQGAYEZwSM+5oA8t/Zg/5JFJ/wBhKb/0FK9jqhpmj6bolp9k0XTrTTrbcX8m0gWJNx6naoAzwOav0AFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAU1mWNSzkKqjJJOABWD4r8Y6N4L0g6jrtyYkY7IYkG6Wd+yIvVj+g74rhxoHir4qOJ/GRn8O+F2OY9CgkK3N2vY3Dj7oP9wfoRmgC5qPxOvde1GXR/hbpQ127iby59UmYpp9qfd/+WhHov4E1Lo/wniudSj1v4ialJ4q1deY1uF22lt7Rw9PxPXrgGu50vS7HRtOisNJtYbS0hG2OGFAqqPp/XvV6gBqqFUKoAAGAB2p1FFABRRRQAUUUUAFFFFABXm3xk/5kL/sc9O/9qV6TXm3xk/5kL/sc9O/9qUAek0UUUAFFFFABRRRQAUUUUAFcv4u+H/h7xnGp1i0xdw829/bt5VxAR0KyDng84OR7V1FFAHlRv8Ax/8ADc41WGXxt4ej/wCXu2TbqFsv+2nSUAdxz1JIru/DPirRvF+kJqOgX0d3btw23ho2/uup5U+xrarz3xL8NPM1l/EngS//AOEe8RHmSSNf9GvO+2ePoc/3gM9+TigD0KivPvC3xIa61pPDPjfTjoHiXGI4nbNvej+9BJ0Of7vXtzg49BoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArjfG3xBtPCfkafZW0mr+Ib75LHSrY5kkP95j/AAIO7H0PocRePPHL+Hfs2j6DbDVPFOp/LYaep+6O8sn92NeTk4zj2JDvAfgOPwsLjVNXuf7U8S6kd+oalIOSf+ecf92McAAYzgegAAKPhLwBdf20vi3x/cx6r4lk/wBTGozb6av/ADzhU9x3br+pPodFFABRRRQAUUVSv7qeysZLi3sbi/kTG22tmjWSTJA4MjKvGc8sOB3PFAF2ivLdJ+Omk654pl8OaX4Z8RS6vC8iS2pitkKNGcOCzThRgjHWr3jj41eE/AOqf2bqr3V1fhQ0ltYxB2iB5G4syqMjnGc4wcc0AeiUVz3hDxnovjjRTqnh66M8AcxyIylXicDO1lPQ4I9q6GgAooooAK82+Mn/ADIX/Y56d/7Ur0mvNvjJ/wAyF/2Oenf+1KAPSaKKKACiiigAoorzvxv8YdN+H11FF4j8P65HFO7rb3MKW8kc23GSMTZHUcMAfagD0SiuWsvG9rc+F49evdN1PTra4Cm1gniSSe6DLuXy44XdiSP4Tg8E4xzWF4b+NXhnxH4sbw4bfVNK1XcUSDU7URGRgM4GGJBxzhsUAejUUUUAFFFFAHPeLvB+j+NdFfT9dty6g74ZkO2W3fs6N/CR+R75FcZpXi7Wvh5qUHh74lzi606VvL03xIRhJPSO4/uPj+I8Huepr1SqGq6VY65plxp2q2sd3aXClJIpVyrD/H0PUGgC6GDKCpBBGQR3p1eSabql78H9Yt9A8S3El34QupPL0vV5Wy1gx6W85/u/3W7D2zt9ZDBlBUggjII70AOooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArkPiD41XwfpES2cBvtb1F/s+mWCfenlPcj+6uQSfw71r+JfEen+E/Dl3rWryCO1tI9zY+8x6BVHckkAfWuQ8AeG9S1PWJPH3jSAJrV9HssLNuRplqfuoAejsDlj15I4yRQBpeA/A//CMxT6trlx/aXifU8PqGoPzz/wA8o/7sa8AAYzgewHa0UUAFFFFABRRRQAUUV5Z4t8VeKfD/AMY9EhuLq20/wVPAftN1coqw+YAxIeUj5GyFCjIBz35oA8w+F3/J2viX/r81P/0ca+gtW0zQNP8ADOstqVnbjTp45rnUDIoPm5BLMxPXjp6AADAArxH4NaDear8dfE/jKCCT+xHub0212UwlwZZsrsJ+8NuSSOnHrU3xi+NWlQ+Il8MxWDaxptnLu1SFbgQrcSKQRCW2tlAR8wA5IxnAOQDZ/Zj8K32i+C77WNQR4l1mWN7eJ+8SBgJMdtxc/UAHoRXt9eUfCP4p3nxN1XUTFo8ekabpcEaeSk3ml5HJ2nO1cALGRgD+Lntj1egAooooAK82+Mn/ADIX/Y56d/7Ur0mvNvjJ/wAyF/2Oenf+1KAPSaKKKACiiigAr54/a0/5BHhn/rvcf+gpXrvxFu/Een+ANUufBkXm6xFGGgURiRsbhuKqchmC5IBByex6V4B8UNQ1L4i+FPAulWkkeseKHSR7+2sQC1u7BBiVR/qj6g4AIboKAPoXwsAvw/0K5S2a4mh0qFo0QLvY+SvyqSQATjHJA9a8AbxTZah+09aan480S90NofLgsIZAPlfkRySn+IEsTlTgYHJAJr1bxd4/Hwi0bw3aXuiXeo2HkLbXN7btxb+WiqOMYYnnAJXoa5fxMNM+NnirwnP4RhuZbXTLk3F/qstq8KRRAq3lBnA3uSOAMgdemaAPdaKKKACiiigAooooAz9X0ix1zSrjTNVtY7q0uUKSxSjIYf0PcEcg815x4Y1C9+GfimHwT4hupbnQ79seH9SnOSh/59JG9R/Ce/QdQB6vWF4s8Lad4z8N3GjavGWhmGUkX78Mg+7Ih7MD/UdCaAN2ivPPh74n1OPUrrwR4zf/AIqHS0DxXPRdRts4Wdffsw9ffOPQ6ACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoorgfir4hvbHRrPw94ckxr3iKf7FZlesKf8ALWb2CL37Eg9qAMfA+K3xIzzJ4S8K3H1TUL8fzSMH8Se4NerV4/pWkeOvhHpqWOkWVv4v8NQFmWG3jFvfQAkk4A4l5J9WPtXZ+EviP4b8ZFodKvfKv48+dp12vlXMRHUFD1x3IyPegDraKKKACiiigAooooAKKKKACiiigAooooAKKKKACvNvjJ/zIX/Y56d/7Ur0mvNvjJ/zIX/Y56d/7UoA9JooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKy9c8Q6R4Z01r/X9Rt7C2XjzJ3xuPoB1Y+wyaAOX+Jfha+1WxtPEPhgBPEugubmyOP+PhMfvIG9Q44x6+mTW74O8U2PjLwrZ63p2VS4GJImPzQyDh429wePfg964r/hOvF/js+V8NdF/s/TH4PiDWYyiMP70MPV/YnjsQKzdE0y9+EPxAsYNR1WbU9I8WOY7q7ljWMR6lklW2rwokB247kZJ4oA9mooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAazLGpZyFVRkknAArzDwAreN/Guq/EO5XNku7TNBDdoEJ8yYf77Z56gAitP4wazc2Hgc6TpJ/4mviGdNKswDyDKcO3sAm7nsSK6vQNGtvD3h+w0exGLexgSCP3CjGT7nqfc0Aadcr4r+HXhrxkBJq+nhb1MeVqFsfKuYiOhEg547A5HtXVUUAeVlPiV8Pz+5ceO9CT+ByItRhX2PSXH4sfaum8JfEjw54wZrbTbw2+ox8S6beJ5NzER1BQ9ceq5FddXL+LPh74c8aRhtasFN2n+pvrc+VcREdCsg549Dke1AHUUV5Wtr8Sfh+T9km/4TrQ0/5Y3DCLUYV9n6S49+T2xXSeFfiX4b8Wym0sbprXU4+JdLv08m5jYdQUPXH+zmgDsKKKKACiiigAooooAKKKKACiiigArzb4yf8AMhf9jnp3/tSvSa82+Mn/ADIX/Y56d/7UoA9JooooAKKKKACiiigAooooAKKKKACiis/VdY07Q9Pe+1i+gsrWP701xIEUe2T39upoA0KztZ1zS/D+nNfa3qFvYWydZZ5Aoz6DPU+w5rz2T4jeIvGrm3+FWiGS0JKtr+rI0NqvbMafek/Lg9RV7SfhHp51JNa8c38/i3WV6S3wAt4T6RwD5VH1z68UAUT8QPFHjhjB8MdEMNix2/8ACQawhjgx6xR/ek9ieM9RV/Q/hFpkGpDWPGF7P4r1vqLrUeYoj6Rw/dUfnjtivQlUKoVQAAMAAdKdQAgAAAAwB0Fc5458KweNPB1/oszCN5k3W83OYZl5RwRzwwHTtkd66SigDjfhl4qn8U+EUbVFMWsadK1jqcLcFLiPhjj/AGuG/EjtXZV5g2PBnx8Vvuab4ztsH+6t7AOPpuQ/iTXp9ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFV726isLC4vLltkNvG0sjeiqMk/kKAPOYh/wl/wAf5pW+aw8G2YjjHY3lwMk++Ixj2NenV5x8EbSZvAT6/fLtvvEV9Pqc+e29yFH02qCPrXo9ABRRRQAUUUUAFcv4t+H3hvxrCBruno9xH/qryE+XPCR0KyDng84OR7V1FFAHlgsfiR8PudNuf+E60RP+Xa7YRahCv+zJ0lx78noAK6Lwt8TvDfiy4aytbmSw1WM4l0zUE8i5Q9xsP3v+Ak12Ncx4r8AeHPGkCpr+nRzTR/6q7T5J4vTbIORzzjp7UAdPRXlZ034j/D9c6Ndf8Jxoif8ALnfOI7+FfRZeknHrz2ArofCnxQ8OeLLg2EM0mnavGdsulainkXCN3G0/e/DPvigDs6KKKACiiigAorzTxn8aNI8O6q+haFaXHiPXhwbOyGVjb0kfnGO4AJHfFcPc6b8RfHJM3i7xK+g2UnTStHOwhfRnB5/EsPpQB7dqnifQtEbbrOtadp7YyBdXSRE/gxFeVfE3x54T1p/CS6V4k0y5On+JrO+ucXSARwR7975JwcbhwMn0FZOn/B/wZY/NJprXsvUyXczOT9QCB+lbC+A/CSDjw3pePe0Q/wAxTA9CsfiL4M1BlWy8V6PK7cBPt0YY/gTmujV1kQMjBlYZBByCK8PvPhh4MvVKzeH7VM94d0R/8dIrJi+GV34ec3Hw/wDFWqaFLnd5Bl82Bz6FO/47vpSA+iaK8O034zeIPCMsdl8VNDb7PuCLremLvibtl07evGD6LXsel6pY61pkOoaTdRXdnON0csLblYf54x2oAvUUUUAFFFVb/ULPS7GW91K6htLaEbpJp5AiIPcngUAWqp6nqljo9hJe6reQWdtEMvNPIEVfxNed3HxQ1XxTM1l8KNEfVgGKPrN8rQ2UJ74Jw0hHoMfjU2nfCSPUb+LV/iTqs3irUkO5IJRssoD6JCOD6Zbr3FAFab4l634wuHsvhVoRvYwSr67qatDZR9iUH3pD9PyIq1pXwjtbm/j1j4g6nP4u1ZTuQXY22sB9I4B8uPrweuBXokUMcEKRQoqRoAqoowFA6ADtUtAEcaJDGscSqiKAFVRgAegFSUUUAFFFFABRRRQBwXxf0SfVvAE9/pny6pocqarYuByJIfmI98ruGPXFdR4c1mDxF4a07WbUfur62SdQDnbuUEr9QePwrSdFkjZJFDIwIZSMgj0rzb4MM+maXr3hCViW8OatNbwbjkm3c+ZET9ctQB6ZRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFcB8bNTk0z4Ra39mybi9jSyiUdWMrhCB/wFmrv68z+LI+3674D0Pqt3r8dxIn99IFLEH25FAHe6Hpkei+H9P0qDHlWNtHbpj0RQv8ASr9FFABRRRQAUUUUAFFFFABRRRQAVzfirwH4d8Z2oi8QabHcSoP3VyPkmi9Csg5HPOOntXSUUAeVnSfiN8P2B0K+HjbRE/5cNQcR30S+iTdH/wCBfQCug8L/ABP8PeKLxtPE0ul6wh2yaVqUfkXCt6BTw34E/hXaVznirwN4d8a2gh8RaZFdFRiOcDbNF/uuPmH0zj1FAHR14h428e6x428R3Hg/4eXps7G1ymra5GM7T08qIjv15HJPQgAk4/jRfF/g2/tvAXh7xfcarFr0LLD9tXddabCGG5/OHJUqGA47HGCBnq/Dvh6w8L6JBpmmRbIYhlmP3pG7sx7k/wD1qYEHhfwjpPhDTfsmj2+0tgyzvzJMfVm/p0FbdFFABRRRQAUUUUAMmhiuIHhuI0likUq6OoZWB6gg9RXn8mm618KL2XxB4Dd7jRmfzNR0KRiUK/xPGf4SB+Ix3HFeh0UAdd4Z8T6Z4u8PW2s6JOJ7W4H/AAKNu6MOzDuP6c1pXl5a6fZyXV9cRWtvEu6SaZwiIPUk8CvnXUrzVfg94mk1bwpHbf2NrjCG4tLpytva3JPyy8fdXrnHbI/u49EtPhRceILuLUvinrT+IblDvj02HMVhbn2QYL/VsZ7g0gG3XxXvfEd7Jpvws0STXZVbY+rXAMNhCf8AfPLkegxntmpLH4TNrF9FqnxO1iXxPexndHZEeXY25/2Yh970y3XuK9FtbW3srWO2tII7eCJdscUSBVQegA4AqegCG3t4bW3SG2hjhhjUKkcahVUegA4AqaiigAooooAKKKKACiiigAooooAK8yswND/aT1CEfLD4i0SO5P8AtTwPsx/3xzXpteZ/ERfsHxP+HWsj5f8AT59Pc/3hPFgA/iDQB6ZRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHCaX8ULHWvHGo+F9K0XWLm702fyrydVgWGIbtu/LShivfhS2BwDWh4u+JHhXwRCT4h1aKGfblbWM+ZM/0ReR9Tge9ebfCX/k4P4k/9dz/6NNVP2pdMsbfwdpd5BZ28V1Lqf7ydIlDvmJs5YDJ+6PyHpQB7VFq/2rw7BrFhY3V2txbx3EVrH5ayuHAIHzuqA4POWxx1rB8EfEnSvHt9rNnpljqNncaNIkV0l6ka4Zi4wNjtnBjbP4da2PBn/Ih6B/2Dbb/0UteSfs+/8j/8T/8AsKR/+jbmgD3avNfFP+mftA+Brc8izsr+6x6FkCA16VXmt5+8/ab05T/yy8LyyD8bjFAHpVFFFABRRRQByXjfx3D4E06TUdU0TVrvTYwhlvbIQMkZZtoUq0qvnOOQuPmHOai8DfESz8fWL6hpOj6pa6cu4C8vVhSNmUjKgLKzZ5znbjg854rI/aC/5IV4h/7dv/SmKsr9nOCO4+CkME6LLFJdXCPG6gqyk4IIPUEUAdV/wtXwjN4ws/DOn6omoaldyNHttB5iRFVZjuf7v8JGASc9q7Svmi9srWw/bP063sreG2hGwiOGMIufsp7CvpegAooooAKKKKACiisLxrfyaZ4B8QX9u22W10y5mjOejLExH6igDxvwdP8A8Jb428T+OJSHS5ujY6ef7ttFgAj03YUn3B9a7muV+GNkth8M9EiUY32wmPuXJf8A9mrqqYBRRRQAUUUUAFFFFABRRRQBleJtDh8SeGb7SbgDbdRFVYj7j9Vb8GAP4VufBPxJN4j+F9h9uJN9pjNp10D1DxYAz6nYUJPqTUFY3wX3WXjz4haUp/cx31vdoo6Bpkdm/ktAHsNFFFIAooooAKKKKACiiigAooooAKKKKACvNfjb+58N+H9RHB07xHY3IPphyv8A7NXpVebfH4Y+Depy9DDPauG/u/6RGM/rQB6TVLU9Qj0rSbvUbhXaK0heeRUALFVUsQM45wKu1ieM/wDkQ9f/AOwbc/8AopqAMnwL8Qbb4g2Bv9H0bVbfT/mVbu8ECozqQCgCys+ec5244POcCtTxX4o03wb4cuNb1qVo7O2GSEALyMeAigkZYn/6+Bk1wH7NrqnwYt2chVW7nJJOABurh/2hbm68QeA7TxCZZI9KOqJbabCDgTxmKUtcsP8AbKgJ6IM/xkAA9qvvG8Fn4Ht/FcOkale6fNZLfuIBD5lvCY/M3OryLnCnkKWNWvBni+x8c+FrfXtIhuYbWdnVFuVVXBVipyFYjqPWuW/5tZ/7kz/2yqt+zp/yRHSv+u1x/wCjmoA9SooooAKKKKACiiigDwv4S/8AJwfxJ/67n/0aazv2pNe0q68M6dpNtqVrNqNvqIea0jlVpIl8puWUHK/eHX1roPBXhnxP4Y+M/irWLzw7czaXrdyRBdwXNsQimTO9laQNtwc8Atx0Ney0Acn8Nte0vXfAOjvpN/Bd/Z7GCKZYpAzQuIwCrgcqeOhrzX9n3/kf/if/ANhSP/0bc17XfXEtrZyT29lPeyIMrBbsgeTnoC7Kvvywryz4MeEPEfh3xL411LxHpLabHrF5HcWqvPFIzDfMxB8tmwR5i9fXigD16vNZ/l/aftCeA3hRwPc/as16VXmutf6L+0d4Ym/5/NFu7f67GD0AelUUUUAFQ3HnfZpTbhTMEPlhvulscZ9s1NRQB8y6/wCL/EOsfA7xPofj+6i/4SebUIYLPTDEsdy6LLE5IjUDcnDYYA5x1PFeqfBjQbvwb8IbGDxAhsJx5t1cJOQvkKzEjdn7uFAJz05zXo1FAHyxrHivQf8AhrWz11dVtJNKiaNHvY5Q0IP2cpneOMBiAT0FfT9tcwXtrDc2ksc9vMgkilicMkikZDKRwQQcgirFFABRRRQAUUUUAFcx8R4jL8LfFKLnJ0i6wB3PlMa6eq93bRX1lPaXK74Z42jkX1Vhgj8jQB478P5Vm+HWgshyBYRL+IUA/qK6KuF+FMktl4evvDV8w+2+H7+aylHqN5Ib6ElgPpXdUwCiiigAooooAKKKKACiiigArE+EmLj4t/ES5U5VDYRceojYH/0GtieeO2t5J53CRRIXdz0VQMk1R+AFpJN4S1TxNdJsl8QapNdJk8+Up2qPwIf8MUAes0UUUgCiiigAooooAKKKKACiiigAooooAK81/aAO74I63H/z1e2TPp/pMf8AhXpVea/HX998PILD/oIatZ22PXMoOP8Ax2gD0qsTxn/yIev/APYNuf8A0U1bdZPia0uNQ8J6vZWcfmXFzZTRRJkDczRkAZPA5PegDyX4KaM/iP8AZwutGS6Nm1+13bi4CbvL3cZxkZ69MivPvjN8N/EPg3wTZ3utePdS8RWzX6QR2V0JFSNjHIQ43SuMgKR06Mea9g+B2ieIPCfgtfD3iTQLmwlillmF0bi3kifcwwo2SFg3J6rjjr0Feo0AePeH9HvNH/ZY1Bb7Vp9TN34dnu4zNn/R45LTKwrlj8qDgYwPYVf/AGdP+SI6V/12uP8A0c1dT8Qk1K88DavpejaPc6pdalZT2iLBLDGIi8bKGcyOvGT2yfasn4MeG9W8J/C/T9I1+z+yX0MkzPF5iSYDSMRypI6Ed6AO/ooooAKKKKACiiigAooooAKKKKACvNPiL/oPxP8Ahxq3RU1C4sWPr58W0D/x2vS682+OcbQfD2HW4lLS6DqlpqSY6/JIFP6OaAPSaKjikSeFJYmDI6hlI7g8g1JQAUUUUAFFFFABRRRQAUUUUAFFFZOv+JdG8Lacb/xDqVvYW46NM+C59FXqx9gCaANasnX/ABLo3hbTjf8AiHUrewtx0aZ8Fz6KvVj7AE1wA8c+M/Hp8r4c6J/ZWlvx/b+sptDD1hh6t7E5HqBWpoHwj0Ww1Eax4lubjxTrp5N9qZ3qh/6ZxfdQenUjsaAPJ9V8Vpb/ABMbx5p2j6nZeE9Z8qxvL28h8tJpQCEnRc524UDOP73c4r1AEMoZSCCMgjvXba/oWn+JdButG1iDz7K7j8uRP1BB7EEAg9iBXhdld3/wt8Rx+D/GVyX0uUE6Lq8nCvGP+WTnoCuQPbjsVpgd/RQDkZHIooAKKKKACiiigAoormPGfjW28LWiQW6fbtZuyI7LT4ss8rscDIHOM/n0FAGP8T9Uvr22h8HeG4JbvWNWRi8MABZLdQS55IHIBAz1GfavRPh5488J3mn2fhixabRtQ0+FLYaRqieTcLtUDGDw54zxz3IFRfCv4d3PhiO51/xRKLvxTqq5upc5FunBEKdsDAzjjIAHABPS+KvA3h3xraCHxFpkV0VGI5wNs0X+64+YfTOPUUgOjoryv+xviL8PyG8OXw8Z6HH/AMw7UnCXsS+kc3R+P730Are8KfFPw94ovDprSzaTrSHbJpepp5E4b0APDfhz7CgDtqKKKACiiigAooooAKKKKACiiigArzT4r/6br3gHSBz9o8RRXTL/AHlgVmI+nNel15lq+dZ/aQ8P2f3o9B0e41BvQPM3kgfXABoA9NooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACsbxdoq+I/Bur6M4BN9aSwrnsxU7T+BwfwrZooA4r4Q60dd+E+hXEpPnw2wtJg33g8RMZz7nbn8a7WvMvAOPDXxP8YeEZPkhuZl1vT1/vJL8soHsrgDH1r02gAooooAKKKKACiisnX/EujeFtON/4h1K3sLcdGmfBc+ir1Y+wBNAGtWTr/iXRvC2nG/8Q6lb2FuOjTPgufRV6sfYAmuAHjnxn49PlfDnRP7K0t+P7f1lNoYesMPVvYnI9QK1NA+Eei2GojWPEtzceKddPJvtTO9UP/TOL7qD06kdjQBljxz4z8enyvhzon9laW/H9v6ym0MPWGHq3sTkeoFamgfCPRbDURrHiW5uPFOunk32pneqH/pnF91B6dSOxr0GigAooooAKyPEfhnSfFmjy6Zr9lHd2sgztccoezKeqsPUVr0UAfP934X8cfCp3XSLebxd4WQ5jRT/AKZaL/dwPvAewI46LWn4d+IHh3xMqrY36RXWcNaXJEcyn02nr+Ga9trjfFfwp8HeMpHm1vRYWumHN3ATFLn1LLjd/wACzQBk0Vz8vwCvtObHhT4ha3p0I6QXKi4UfkyAflXPeJvC/wAQ/Cf9lLd+P0nXVtTh0yAx6XESssu7azZ6L8pyRk+xpgeg1S1PWdN0W2Nxq19b2cX96aQLn6Z6n2FZK/Bjxtf/APIY+Jt0iHgpY2QjJH+8GXH5Vs6L+z94K066F3q0d7r93wfN1S4Mgz/urgEezZoA4hPG3iDxtPJp/wALtDmvFB2SaveL5dtD7jPU85wef9k16J8PvhPYeDpm1fVpzrXia45n1KcZ2EjlYgfujHGep9hwO+tbW3sbSO2s4Ire3iXbHFCgREHoAOAKnpAFFFFABXOeKvA3h3xraCHxFpkV0VGI5wNs0X+64+YfTOPUV0dFAHlf9jfEX4fkN4cvh4z0OP8A5h2pOEvYl9I5uj8f3voBW94U+Kfh7xReHTWlm0nWkO2TS9TTyJw3oAeG/Dn2FdtXOeKvA3h3xraCHxFpkV0VGI5wNs0X+64+YfTOPUUAdHRXlf8AY3xF+H5DeHL4eM9Dj/5h2pOEvYl9I5uj8f3voBW94U+Kfh7xReHTWlm0nWkO2TS9TTyJw3oAeG/Dn2FAHbUUUUAFFFFABRRRQAV5l8Nsa38Q/HnignfG9+mlWzHoEt0w+32LEH8K63xx4jj8JeBdX1yRgps7Zmiz/FIeI1/Fio/GqHwv8Ov4X+GujadcqwuzB9out33vOkO98+pBbH4UAdfRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHmXxXB8Oat4d8f26kDRrsW2o7f4rOc7GJ9drEED1NelqyyIGRgysMgg5BFZ2v6NbeIvD9/o98M299bvBJxyAwxke46j3Fch8I9aurjw3P4b1z5db8My/YLsf8APRAP3Uo9mQDnvgnvQB6FRRWTr/iXRvC2nG/8Q6lb2FuOjTPgufRV6sfYAmgDWrJ1/wAS6N4W043/AIh1K3sLcdGmfBc+ir1Y+wBNcAPHPjPx6fK+HOif2Vpb8f2/rKbQw9YYerexOR6gVqaB8I9FsNRGseJbm48U66eTfamd6of+mcX3UHp1I7GgDLHjnxn49PlfDnRP7K0t+P7f1lNoYesMPVvYnI9QK1NA+Eei2GojWPEtzceKddPJvtTO9UP/AEzi+6g9OpHY16DRQAUUUUAFFFFABRRRQAUUUUAFFFFABXm3xk/5kL/sc9O/9qV6TXm3xk/5kL/sc9O/9qUAek0UUUAFFFFABRRRQAUUUUAFFFFABXOeKvA3h3xraCHxFpkV0VGI5wNs0X+64+YfTOPUV0dFAHlf9jfEX4fkN4cvh4z0OP8A5h2pOEvYl9I5uj8f3voBW94U+Kfh7xReHTWlm0nWkO2TS9TTyJw3oAeG/Dn2FdtXOeKvA3h3xraCHxFpkV0VGI5wNs0X+64+YfTOPUUAdHRXlf8AY3xF+H5DeHL4eM9Dj/5h2pOEvYl9I5uj8f3voBW94U+Kfh7xReHTWlm0nWkO2TS9TTyJw3oAeG/Dn2FAHbUUVXvLu30+wnvbyVYbe3jaWWRuiIoySfoBQB5x8QP+Ks+I3hjwRH89rE/9taqo6eTEcRIfZn4I+hr0+vNfhPa3Gsz614/1OJop/EU4+xROPmhso/liHsW+8fXg969KoAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAK8u8fD/hB/HGl/EK1BFhLt0zXVUf8ALFj+7mPujYBPXBAr1GqGr6Xaa3o9zpupwrPaXUTRSxt/EpGPwPv2oA86Pjfxp46cw/DvQzpOmMcHXtajK5HrDD1b2JyPUCtPQPhHothqI1jxLc3HinXTyb7UzvVD/wBM4vuoPTqR2NVfhtql54f1a7+HPiOdpb3S083S7qTre2JOFP8AvJ90j274Jr0ugAooooAKKKKACiiigAooooAKKKKACiiigAooooAK82+Mn/Mhf9jnp3/tSvSa82+Mn/Mhf9jnp3/tSgD0miiigAooooAKKKKACiiigAooooAKKKKACiiigArnPFXgbw741tBD4i0yK6KjEc4G2aL/AHXHzD6Zx6iujooA8r/sb4i/D8hvDl8PGehx/wDMO1Jwl7EvpHN0fj+99AKxvEHjP/haz6b4F0iz1HTJb6cvr0N5AYpbS1iKll/4GxCgj6HrXqPizxPY+DfDF5reqk+TbJlUB+aVzwqL7k4H61zvw08M39la3nijxQufEmvsJ7kEf8esX/LO3X0CjGffrnFAHb21vDZ2sVtbRrFBCgjjjQYCKBgAewAqeiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA4f4leErvXdNtNX8OssHiXRJPtOnTHjef44W/2XHGPXHbNafgnxhZeNfDqahaK0FxGxhvLSTiS1nX70bD2PT1FdLXl/jbR77wT4kf4h+E7dpkKhdf02P/l7gH/LZR/z0Tr7j8cgHqFFZ2jaxY+INHtdV0mdbizuoxJFIh6g/wAiOhHUEYrRoAKKKKACiiigAooooAKKKKACiiigAooooAK82+Mn/Mhf9jnp3/tSvSa82+Mn/Mhf9jnp3/tSgD0miiigAooooAKKKKACiiigAooooAKKKKACiiigAprMsalnIVVGSScACnV5X4v1O8+IPiiX4f8Ahi4eHT7fB8RanEf9Uh/5dkP99uh9BkdmFAEOnk/Frx9DqwUt4O8OTn7FuHy6leDgy47onY9z9SB61VLS9Ns9G0y207TIEt7S2jEcUSDhVH+evertABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFIQCCCMg9RS0UAeR6npd98INZn8QeGbeS88H3UnmatpMQy1iT1uIB/d/vL6e2Cvp2lanZa1plvqOl3KXNpcIHiljOVYH/PTtVwqGUhgCCMEHvXlWpaFrPwr1O51zwRavqPhq4czal4fj+/bn+Ka2/mU/pjaAer0Vj+G/E2leLtFh1XQbtLm1l4yOGRu6sOqsPQ1sUAFFFFABRRRQAUUUUAFFFFABRRRQAV5t8ZP+ZC/7HPTv/alek15t8ZP+ZC/7HPTv/alAHpNFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFeXa14z1fxvqtx4Y+GMqpFC3l6l4iPMNqO6Q/wB+THccD9QAS+LPF2qeIPEMngX4ezBdRA/4murgbo9MjPUA95TyAO31yV67wl4U0zwZoEOk6PCVjUl5ZXOZJ5D96R27sf8A6w4FM8H+D9L8E6DFpejxEIDvmndsyXEh6yO3cn9Og4roaACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooA818Q+AtT0jXpfFnwykjs9UkO6+0qVttrqY75HRJPRuOeuMknX8G/EfTPFdzJplzDNpGv24/0nSL0bZUPcr/AH19x2wSBmuzrlfGPgDRvGlvGdRSS31C3O601K0by7i2bqCrjtnsePx5oA6qivKo/GHir4cn7J8RrSTVtHQ7YvEmnxbtq9vtEQ5U+rDj6nmvRNH1nTtf02O/0a+gvrWX7s0DhlPtx0PseRQBo0UUUAFFFFABRRRQAUUUUAFebfGT/mQv+xz07/2pXpNebfGT/mQv+xz07/2pQB6TRRRQAUUUUAFFFFABRRRQAUUUUAFZmt65pvh3SJdT1u8isbSEfPLK2B9B3JPYDk1yGvfFayh1N9C8G2M3inXhwbexI8mA9MyzfdQZ+vocVX0b4a3ms6tF4i+KF5HrWpxndbabGD9hsPZUP32/2m/XANAGY9x4k+MTCGzjuvDfglz+8uH+S81RP7qD/lnGfXuPXkD0vRdE07w9pEOmaLZx2dnAu2OKMYA9z3JPcnk1pAYGBRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRXm/gH4nal4917VrS18PW1pZaVc/Z7i5l1FmduWAKRiHB+7yCwxnqaAPSKK4K5+I8l1r+u6V4U0ddXfw/GH1GSS78hQxDHyo8I+9/lbg7RkEZrR0HxqvjHwVDr/gu3hvnkODbXlybfy2H3kZlR8MOO2DkHODmgDqmUMpVgCCMEEda851f4Tx2mpya38OtSfwtq7cyRwLus7r2kh6fiOnXBNXPhT8SP8AhZmgXmp/2V/Zn2W6+z+V9p87d8itnO1cfexjHau8oA8vT4oax4VZbf4oeG5tNQEL/bGmg3Nk/u2Pmj+hya7/AEfXNL1+wW90TULe/tm6S28gcZ9DjofY81edFdCjqGVhggjII9K8/wBX+D+izX7ar4WuLvwpqx5NzpL+Wkns8X3WHtxnvQB6HRXmH2n4s+FR/pNppfjWyT/lpbt9iu8epU/IfoOTVrT/AI1+FmuhY+JFvvC+of8APtrNs0OfcPyuPckUAei0VS07V9P1e38/Sb+1vof+eltMsi/mpNXaACiiigArzb4yf8yF/wBjnp3/ALUr0mvNvjJ/zIX/AGOenf8AtSgD0miiigAooooAKKazKilnIVVGSScAVx+ufFbwR4fyl/4is5JwcfZ7R/tEpPptjyQfrigDsqQkAEk4A6k15mPHvjXxN8vgfwPLawN93UfEb/Zox7iFcuw9waQ/DDXPExDfEnxhdalbHltJ0xfslqf9livzSD64NAF/XPi94fsL86V4fS58Uaz0FlpCedtP+3IPlUDvySO4rLfwr44+IDA+O78eHdEb/mB6RNummHpNP6eoXgj0Nd9onh7R/DWniy0DTbfT7cfwW8YXcfUnqx9zk1q0AZOg+G9H8L6amn+H9OgsLVf4IVxuPqx6sfckmtaiuF8B/FHT/iGZZNB0TV47OFzHLeXKwJGj7d23AlLkkY6KevOKAO6orzHxt8d/C3gTxNLoWq2uqXN3DGjyGzijZE3DIUlpFOcEHp3FaGgfFWy8UeELrxLofh7W7mztZmikj2W6SYVN7OA0wDKOBwc5PAPOADvqK5jwJ46034heHTrOiw3UFuJ2g2XaKr7lAJOFZhj5h3rp6ACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArwv9nH/kLePf+wqv/oUte6V4X+zj/wAhbx7/ANhVf/QpaAMD9mW7lv0+IF5ctumuBbyyH1ZvtBP6mrn7Jt3O+j+JbNt3kRT28qHtudXDfjhF/Stfwr4avPhT4n8bRHSNQvtP1tUm0t7G1ecEjzf3L7AfLIMoG5sLgZzXQ/Az4f3ngLwMyauqpqeoTfaJ0GD5IwAsZI6kck+7Eds0Acx+yl/yIGsf9hQ/+ikr3avCf2Uv+RA1j/sKH/0Ule7UAFFFFABVS906z1O2NvqVnb3kB6xXESyKfwIxVuigDz/UPgn4Gvbj7VZ6U+j3f8NxpVw9syfQKdv6VVHw+8a6Tx4Z+JeoGEdIdatI70n/ALaHDD8q9KooA81/4vNp3/Qo6xGP+u9vK3/stH/CcfEWy+XU/hfLNj/lrYavDIG/4CRkV6VRQB5sPitq0PF78M/FyH/p3tUm/kwrkfH3jbUPE3/CNCz+H3jSJtK1+11OfzNIJDRRbtwUqxy3zDAOB7ivd6KAPNf+Fu3rcRfDbxqWPTfpoUfnuob4j+MrjjTvhXrDk9PtV7Dbj8Sc4r0qigDzX+1/jBqP/Hp4Y8OaNn/oI6g9wV/79Cg+GvirqfGo+O9N0pG+8ul6UJDj0DSHI+telUUAeaD4JaLqDCTxdrOveJn6lNR1BxED/som3A9s112ieDPDfhvB0HQrCwcDHmQ26hz9XxuP4mt2igAooooAKKKKACvBv2V3Efw81pyGIXUiSFUkn90nQDrXvNeDfssuY/hzrbqjOV1JiEXq37lOBnvQBwFz8QL/AEPWr6/8dfCqxvIb69kn+0avpjR3JVmJVPMkVhhVwoAHQfjXvXhfX/DXiX4T3l94Ns4LCw+zTq9nDCsP2eXYSylV4B5ByOuQayfCnxg8N+OFvNC8U6eNE1Abo5dL1P51mT0BZQCf9kjPpmovh34LPhD4feLGWKS2ttRuLu4s7aUEPFbbSsW4HkMVGSDyOAeQaAKv7MH/ACSKT/sJTf8AoKV7HXjn7MH/ACSKT/sJTf8AoKV7HQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFeU/DDwD4o8BeIddkvBpF3p2sXn2gyRXkomhALkfIYcMTuHG4Y9TXq1FABVDVH1OKwZtFtrW6u8gLHd3DQx47ksqOfwxz6ir9FAHmnwT+Hmq/Djwvf6drdxZzzXN556NZu7KF2KuDuVTnKntXpdFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAV4z8A/DHifwLpN7o/iXw7c232q6Nwl2txbyRKPLAwwWUvnK8YU9ecV7NRQAVh+K5b2Pw3dw6ZpNzqtxcRvCsNvJEhUsjYZjK6DbnAOCTz0PNblFAHmfwJ8J614N+Hb6X4ksvsV4b6WUR+aknylVAOUJHY969MoooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAory62+Kuv+Ib29fwL4En17S7O5a1e/fU4bUO69diuPmHI5z37V6JpdzdXelW0+o2JsLqSMNLatKshhbuu5eGx6igC7RRRQAUUUUAFFFc5N4x0238f2/g+RLgalc2RvkYKPL8sMVwWznd8p7dutAHR0UUUAFFFFABRRRQAUVzHjnxrY+BdBXUL2Ga7mnmW3tbOBd0lzM33UUfh1/mcA4Xh74m3t34tt/DfjHwvc+GNSvYTNYeZdLcR3IUZZQ6gAOBzt/PBwCAeiUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABWN4l8PnxLox0/8AtbVNJy6v9p0u58iYY7bsHg9xWzWH4o0zW9V0lbfwzr/9gXglDm7Fkl1lADlNjkDkkHPt70AeHfBP4enxF4Ku7z/hL/FWleXqc0Xk6XqfkRNtC/MV2nLHPJ74FehfGjSrpPh7d6zp/iHW9LudItmeNbC9MKzklR+9wMt+Y6msrw18IvGfhCwex0D4mm2tZJ2neP8AsCF9ztjccs5Izge1dP8AETwRr3jeybT9O8WnRtMnhMV3af2bHcefznO8sGX0wDQBx3hbwZ458W+DNI1zVPiDqWmXkllbvYxWLFolj2KQ84Y5mdxycngk9elW/EGp+IfGXxck8BaXrtzoWnaZYLd393ZYW5uWbbgI38A+denv14rc8IeBvGfhmfTbe7+IH9o6Pp8SwLp/9iwxbo1Tai+aGLDGBzyTjmrHi74ax6/r8HiPQ9YuvD3iG3j8oX1qgdZU/uyRnhx+P1zgYAOVln174W/Erw3pUniPUPEOg+IpmtfK1SQS3FvKMYYSYGRl149M8ZwaxrfQNb1b48+IvCyeO/FFppcFol4RHfEyfOUOxGPEYBfjAzgYrutB+F00Hiy38TeMvEl14n1azUrZtLAlvDb56lYlyN3vn3xnBFfR/hjr+mfFO58ZT+NPtRvMx3Fp/ZcaeZAPuRbw/G3C/MFydvPU0AUvh9qGsaL8WfEngW+1q+1uxs7WK8tLjUpBJcR7gu5S/Vh846/3eAMmuR1vwJ5/7StlpX/CU+JYzeaPJefbI9RxcQZll/dRvt+WIY4XHc813ej/AAx1/TPinc+Mp/Gn2o3mY7i0/suNPMgH3It4fjbhfmC5O3nqa0PGPw5n8QeKbDxNoHiCfw/rtlAbZbpLdLhHiJJ2tGxAPLN+fsKANzwr4X/4Raxmtv7c1nWvNk3+brF59okTjG1TgYHtXQVx/hPwfrOianPqXiDxjqfiC5mi8rypUWC2TkHcsK8BuMZz0JFdhQBw3xS13+wPDlvc/wDCY/8ACJb7kJ9t/sv7d5nysdmzBx0zn2968m/4Wb/1cD/5Zf8A9hX0lRQBx/w01j+3PByXn/CU/wDCU5ndf7Q/s/7FnB+75eB09e9dhRRQB5H8Wv8Akqfwu87/AI9v7Um3bvu+Z+68v8c5xR8ZP+R5+GXkf8fP9vptx97y90fmfhjGa7Txz4KsPHegDTb6aa2mhlW4tLy3OJLaZfuuv6gj37HBGH4e+Gd7a+LrfxJ4x8U3PifUbKIw2HmWq28VsGGGYIpILEcbvzycEAHolFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUVieKPFui+DNKXU/El79is2lEIl8p5PnIJAwgJ7HtQBt0Vwtx8ZPANtolpqs/iKFLW8ZlgJhl3vtJUny9u8DIIyQB71t6j408OaX4ci1y+1q0g0ydQ0NyZMrLkZG3HLH2HNAG/RXHeHPit4J8W6gLDQdfguLtvuQSRyQu/GflEiru49M1Z8U/ETwr4Knii8T6vHYyzJvjjMTyMy5xnCKT1oA6iiuc8LePPDPjWOZvDOrxX/k48xAjI6A9CUcBse+MVRufip4Ns/Fv/CM3WsiPWPOWD7MbabAdsYG/Zt7jnOKAOxorkdF+J3g3xD4gfQ9E162u9RXOIUDAPgZOxiAr4AJ+UngV11ABRRRQAUVnarrelaFbLc61qdnp0DNsWW7uFhVm64BYgZ4PFZf/CxvBH/Q5eH/APwaQf8AxVAHS0VR0zV9O1q0F1o2oWuoWxYqJ7WZZUJHUblJGavUAFFcr450fxNrtnZWPhfXE0WN5/8AT7lVzN5OOkRwcNnHPH17HzyeTVvhr8VfC2i2PizVvEVprrtDd2Or3P2mWEDGJVbAKjknH+yevYA9tooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACsXxNrelaJos8ms6nZ6ek0bxxNd3CRB22n5QWIyfYVtVm6toOk67CkOuaXZ6lFG29EvLdJlVsYyAwODjvQB4f8AH8Aad4Ehm1O90SLXtQklgnjvLqLznQuVWMIxyFIxwBznvVX4x6ZLpXxV8KTtfQ+HvDsNn9ntLsaYlzbWUwLkgwt8gyNmD2wCPu5r2iL4f+DYJkmg8JaFFLGwZXTTYQykcggheDW5d2dtf2r219bRXMEgw8U0YdW+oPBoA+bvF9hpmrS6dc+IvjlpV5cWsySWk9locLzRPn5fmt33AZxweOAe1dp41+IOtt8T5/CNj4j0vwXZ2dqk76tqUaO1yWAOIhIQhAzjB5yrc9q9K0/wAGeGNKnE+meHNJs5h0ktrGKNh+KqKsar4d0TXRGNc0ew1IR/cF5apNs+m4HFAHz34A8R2o/aLnudT8Z2OuCTTGtf7U+zx2KTyb12xhRgO3AwRnOOM4ra+Oeo+GNe8ZeCtJv9WsJraDVJItUjW8QG3QmMMJCGzH0I5x0Nevv4L8LNcw3D+GtIaeAAQyGxiLRYORtO3IwSTxUd14E8I313LdXvhXRbm5mcvLNNp0LvIx5JLFckn1NAEXh7/hC9R+znwsdDvTpKeXC+ntDKbRXBGAUyU3AH0zg109ZWk+HdE0AS/2Fo9hpomx5os7VIfMxnG7aBnGT19TWrQAUUUUAcn4+0XVdc0OC30TTPDmpTrOGaPxFbvNAFwRlQoJ3cjn0zXn3/CuPG//AEJvwg/8Fc//AMTXttFAHN+B9J1HRfDSWmsafoOn3IldjDoELRWwB6EKwB3etdJRRQBzviLxj4Z8PXdrp3ibVYNPfUUcQi4JRHUYDZf7q/eHUivD77TvBnhb4qeE7r4P6hb3Wo3V8Le7sbS5+2RG3b/WPvJbYcZ/i9+MGvoTU9H0zWrcW+sadaX8OciO6gWVR+DAioNK8M6FoTM2h6Lp2nFxhjZ2iQlvrtAoA1aKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAP/9k=)이를 해결하기위해 우리는 queue의 구조를 원형구조로 만든다.

해당 그림과 같이 큐를 원형의 형태로 만든다면, 데이터를 제거할 때마다 큐의 길이가 줄어들지 않는다. 예를 들어, 데이터 1,2,3,4,5를 넣고, pop을 5번한다했을 때, 기존의 queue는 더이상 데이터를 추가할 수 없지만, 원형큐는 데이터를 추가할 수 있다.

원형큐를 구현하는 핵심은 ‘%’ 나머지 연산자이다.

다음과 같이 크기가 6이라면, [5]다음에 [6]이 아닌 [0]으로 가 원형형태를 만들어주면 되는데, 해당 연산은 나머지 연산으로 가능하다. 데이터를 추가하면 rear + 1이 아닌 (rear + 1)%6 을 하면 원의 형태를 구현할 수 있다.

**<ADT>**

int front = -1; int rear = -1;

int IsEmpty() {

if (front == rear)//front와 rear가 같으면 큐는 비어있는 상태

return 1;

return 0;

}

int IsFull() {

int tmp = (rear + 1) % MAX; //원형 큐에서 rear+1을 MAX로 나눈 나머지값이

if (tmp == front)//front와 같으면 큐는 가득찬 상태

return 1;

else

return 0;

}

void addq(int value) {

if (IsFull())

printf("Queue is Full.\n");

else {

rear = (rear + 1) % MAX;

queue[rear] = value;

}

}

int deleteq() {

if (IsEmpty())

printf("Queue is Empty.\n");

else {

front = (front + 1) % MAX;

return queue[front];

}

}

int main() {

addq(4);

addq(7);

addq(12);

printf("%d\n", deleteq());

printf("%d\n", deleteq());

printf("%d\n", deleteq());

deleteq();

}

텍스트이(가) 표시된 사진

자동 생성된 설명

**<문제풀이에서 사용법>**

#include<stdio.h>

#include<stack>

using namespace std;

int main()

{

stack<int> s;//stack<원하는 자료형> 이름;

//push, pop, empty등 기본적인 구현은 첨자를 사용해서 호출하면 된다.

for (int i = 1; i <= 5; i++) {

s.push(i \* 10);//push 기능, s에 값이 추가된다.

}

for (;!s.empty();s.pop()) {//(;스택이 빈 상태가 아니면 반복;한번 반복할 때마다 원소하나를 제거한다)

printf("%d ", s.top());//stack 에서 top이 가리키는 원소를 출력.

}

}

#include<stdio.h>

#include<utility>

#include<queue>

using namespace std;

int main()

{

//queue<자료형> 변수이름;

queue<int> q1;

for (int i = 1; i < 10; i++) {

q1.push(i);

}

for (;!q1.empty();q1.pop()) {

printf("size : %d , front : %d, rear : %d\n",q1.size() ,q1.front(), q1.back());

}

//queue 혹은 stack를 사용할 때 순서, 값을 모두 기억해야 하는 등, 두개의 원소를 한번에 저장해야 할때가 있다.

//그럴땐, pair를 이용해주면 여러개의 원소를 한번에 저장할 수 있다.

queue<pair<int, int>> q2;

int arr[10] = {2,7,8,1,3,5,10,9,4,6};

for (int i = 0; i < 10; i++) {

q2.push(make\_pair(i, arr[i]));

//q2.push({ i,temp }); 위와 동일한 코드, 편한 방법으로 사용하면 된다.

}

for (;!q2.empty(); q2.pop()) {

printf("(%d, %d) ", q2.front().first, q2.front().second);

}

}}

**<stack 문제풀이>**

**백준**

제로(10773)

#include<stdio.h>

#include<stack>

using namespace std;

int main() {

int n;

stack<int> s;

scanf("%d", &n);

for (int i = 0, temp; i < n;i++) {

scanf("%d", &temp);

if (temp) { //temp가 0보다 크면 push

s.push(temp);

}

else { //temp가 0이면 pop

s.pop();

}

}

int sum = 0;

for (;!s.empty();s.pop()) {

sum += s.top();

}

printf("%d", sum);

return 0;

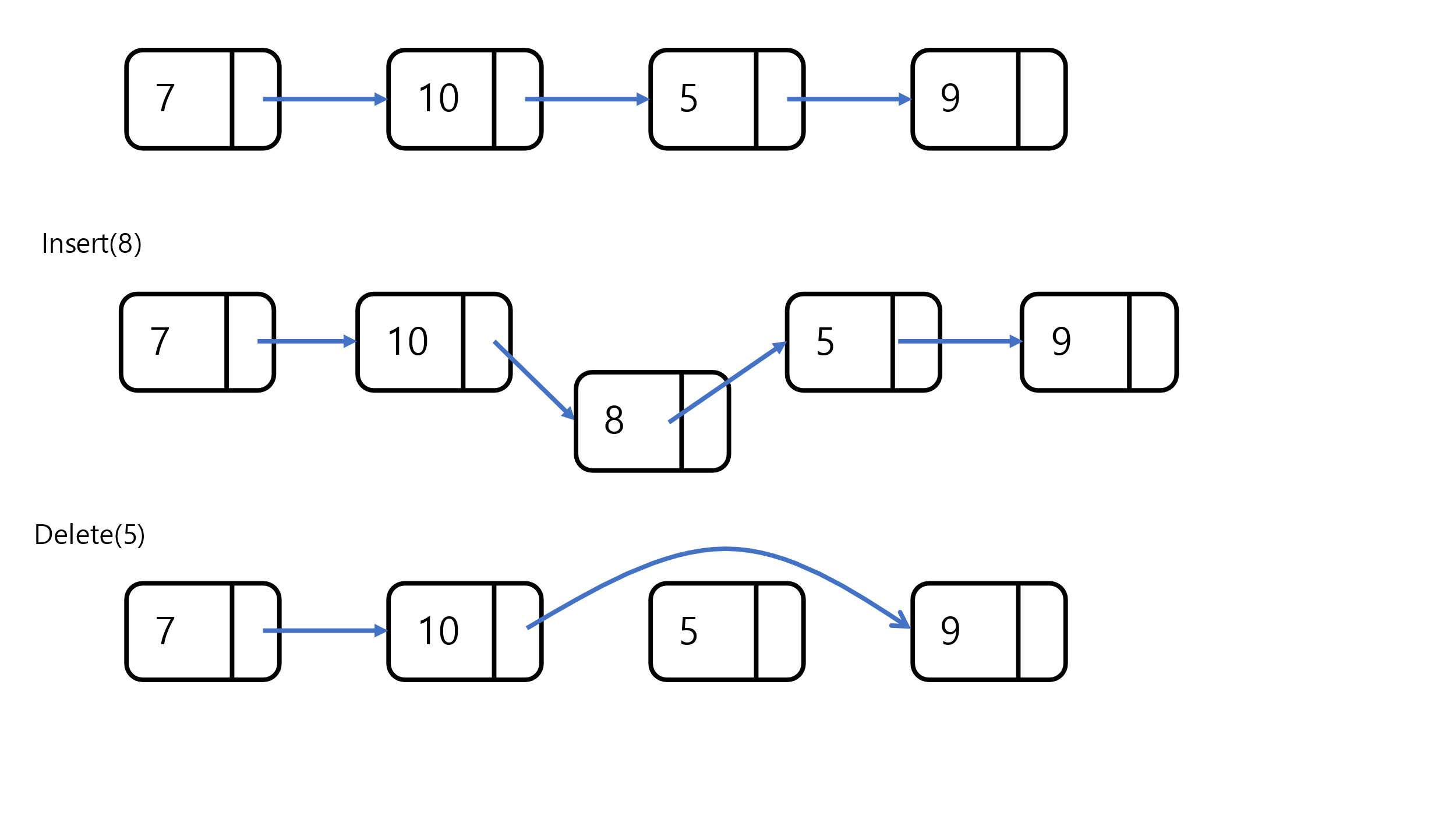
}

**List**

list는 stack, queue와 같이 선형구조를 가진 자료구조이다.

list는 노드로 구성 되어있고, 하나의 노드는 데이터와 다른 노드를 가리키는 포인터를 가지고 있다. 포인터를 통해 노드들이 한 줄로 연결된 구조이다. 정해진 위치에서만 데이터 삽입-삭제가 가능한 큐, 스택과 다르게 데이터의 위치에 상관없이 가능하다. 또한, 이 과정이 매우 편리하다.

만약 stack[5] = {1,2,3,4,5}에서 데이터 3을 삭제하려면, 5-4-3을 순서대로 삭제한 뒤, 다시 4-5를 넣어야 한다. 하지만, List는 연결된 순서만 바꾸면 되기에 이 과정이 매우 싶다.



**<linked list 구현>**

#include<stdio.h>

#include<stdlib.h>

struct Node {

int element;//data 부분

Node\* next;//다음 노드를 가리키는 포인터

};

Node\* list = NULL;

void add(int data) {

Node\* temp = (Node\*)malloc(sizeof(Node));//노드를 동적할당으로 생성

temp->element = data; temp->next = NULL;

//list에 노드가 없는 경우

if (list == NULL) { list = temp; }

//list에 노드가 있는 경우

else {

temp->next = list;

list = temp;

}

}

bool del(int data) {

if (list == NULL)

return false;

Node\* cur; Node\* pre = list;

//data와 같은 node를 찾는 과정

for (cur = list;cur != NULL && cur->element != data;pre = cur, cur = cur->next);

if (cur == NULL)

return false;

if (pre == cur) {list = cur->next;}

else {pre->next = cur->next;}

free(cur);

return true;

}

int main() {

int arr[5] = { 5,4,3,2,1 };

for (int i = 0; i < 5; i++) {

add(arr[i]);

}

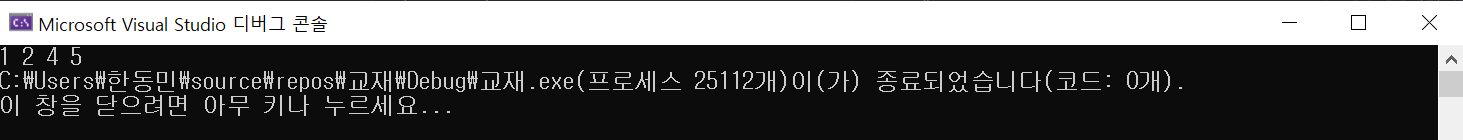
del(3);

for (Node\* temp = list; temp != NULL; temp = temp->next) {

printf("%d ", temp->element);

}

}



**<실전에서 사용>**

#include<stdio.h>

#include<list>

using namespace std;

int main() {

//list<자료형> list이름;

list<int> list1 = { 3,2,1,5,4 };

list<int> list2 = { 6,7,8,9,10 };

list1.push\_back(0);

//auto는 초기화값에 맞춰 자동으로 타입을 정해주는 키워드이다.

//아래에서는 자동으로 list의 위치에 대한 타입으로 정해줬다 생각하면 된다.(정확히는 iterator type)

for (auto it = list1.begin(); it != list1.end(); ++it)

printf("%d ", \*it);

printf("\n===========================\n");

list1.pop\_front();

list1.sort();

for (auto it = list2.begin(); it != list2.end(); ++it)

printf("%d ", \*it);

//merge 함수는 2개의 list를 합쳐준다.

list2.merge(list1);

printf("\n===========================\n");

for (auto it = list2.begin(); it != list2.end(); ++it)

printf("%d ", \*it);

}

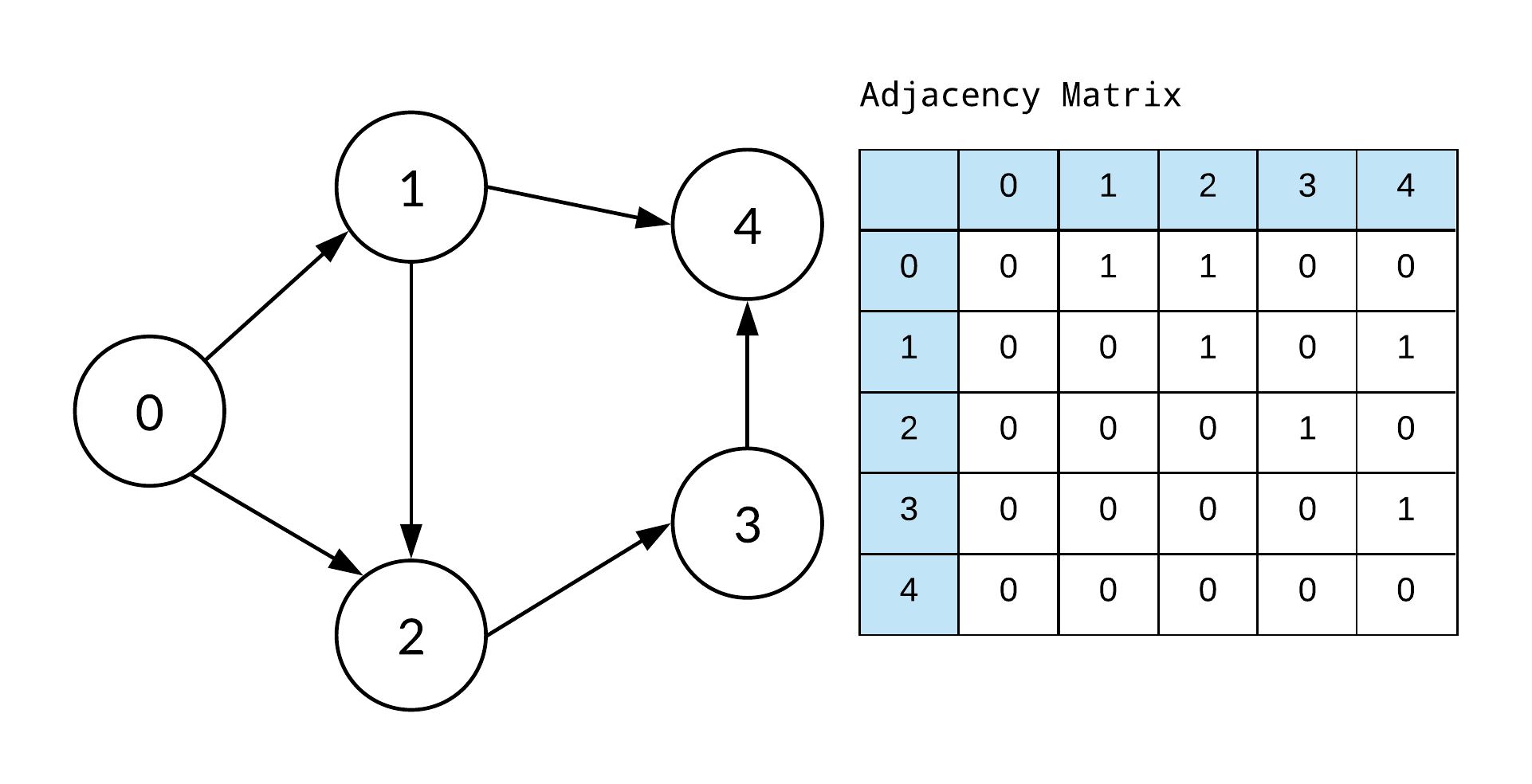
**Graph**

그래프란 정점(노드)와 간선의 집합을 의미한다. 객체사이의 관계를 표현하는 자료구조라고 생각하면 된다. 예를 들어 페이스북과 같은 SNS 관계망은 그래프로 표현된다. 아래의 그림에서 주황색 화살표로 표시된 사람이 노란색 화살표의 사람의 연락을 하고 싶으면 빨간색 화살표의 사람을 거쳐가야 한다. 위처럼 객체사이의 관계를 표현한 것이 그래프이다.



그렇다면 그래프를 어떻게 구현할 수 있을 까? 2차원 배열을 통해 그래프를 코드로 구현한다.

간단하게 정점이 연결되어 있으면 1, 아니면 0으로 표현하고, 이 행렬을 인접행렬이라 부른다.



**<예시 구현>**

#include<stdio.h>

#include<stdlib.h>

#define MAX\_VERTICES 5

int adj[MAX\_VERTICES][MAX\_VERTICES];

int main()

{

for (int i = 1, s, e; i <= 6; i++)

{

scanf("%d %d", &s, &e);

adj[s][e] = 1;

}

for (int i = 0; i <= 4; i++) {

printf("정점%d ->", i);

for (int j = 0; j <= 4; j++) {

if (i == j)

continue;

if (adj[i][j])

printf("%d ", j);

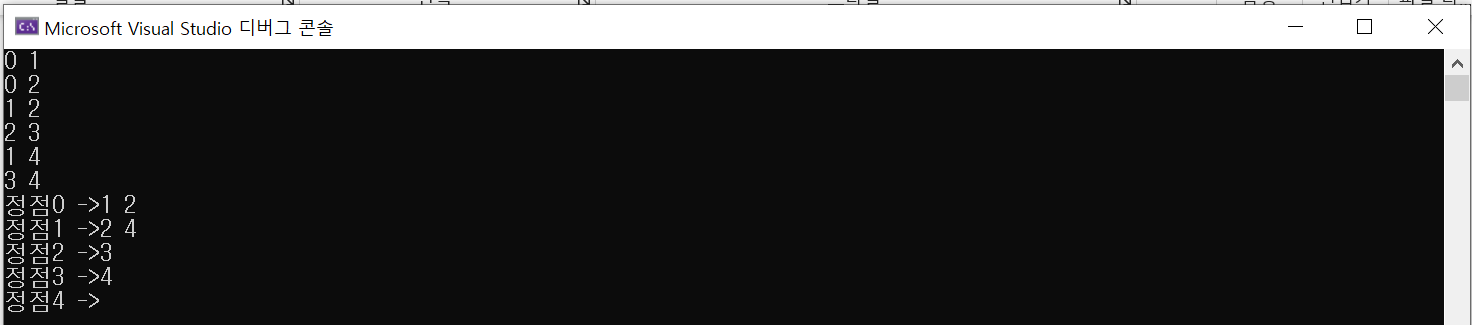
}

printf("\n");

}

return 0;

}



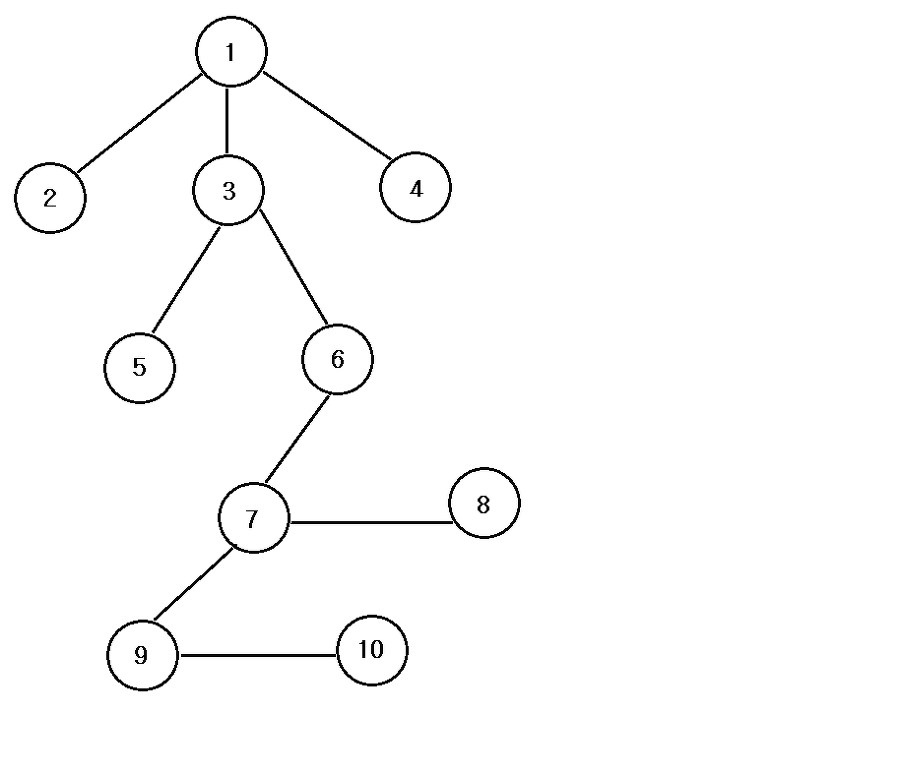
**<그래프 탐색>**

위의 예시에서 만약 정점3에서 정점1로 갈 수 있는 지 판단하고 싶다고 하자. 이를 알기 위해선 그래프를 탐색해야 한다. 즉, 그래프 탐색은 특정 노드에서 다른 노드들을 갈 수 있는 지 없는 지를 판단하기 위해 이뤄진다.

그래프 탐색의 종류는 2가지 있다. 깊이우선탐색(DFS)과 너비우선탐색(BFS)이 있다.

먼저, DFS는 노드를 탐색하는 데, 해당 노드와 연결된 노드들부터 검사하는 것이다. 전체적으로 위에서 아래로 깊게 탐색한다.

BFS는 이와 반대로, 자신과 가장 가까운 노드부터 탐색한다. 그림을 통해 이해해보자.

(DFS)

1과 연결된 노드는 2,3,4 여러 개가 있으니, 가장 왼쪽인 노드부터 탐색해보면 2를 먼저 검사하고 그 다음 3을 검사한다. 3과 연결된 5, 6을 순서대로 검사하고 4를 마지막으로 검사한다.

정리하면 1->2->3->5->6->4 순서이다.

(BFS)

가장 가까운 노드부터 방문하므로 거리가 1인 2, 3, 4를 순서대로 방문한 뒤, 거리가 2인 5와 6을 방문한다. 정리하면 1->2->3->4->5->6 순서이다.

2가지 탐색방법을 어떤 식으로 구현할 까? 인접행렬을 이용해 쉽게 구현할 수 있다.



#include<stdio.h>

#include<stack>

#include<queue>

bool visit[7];

bool adj[7][7];

using namespace std;

void DFS(int v){

stack<int> s;

s.push(v); visit[v] = true;

printf("DFS(탐색순서) : ");

while (!s.empty()) {

int cur = s.top();

printf("%d ", cur);

s.pop();

for (int i = 6; i >= 1; i--) {

if (adj[cur][i] && !visit[i]) {

s.push(i);

visit[i] = true;

}

}

}

printf("\n");

}

void BFS(int v) {

queue<int> q;

q.push(1); visit[1] = true;

printf("BFS(탐색 순서) : ");

while (!q.empty()) {

int cur = q.front();

printf("%d ", cur);

q.pop();

for (int i = 1; i <= 6; i++) {

if (adj[cur][i] && !visit[i]) {

q.push(i);

visit[i] = true;

}

}

}

printf("\n");

}

int main()

{

for (int i = 1, s, e; i <= 5; i++) {

scanf("%d %d", &s, &e);

adj[s][e] = adj[e][s] = true;

}

DFS(1);

for (int i = 1; i <= 6; i++)

visit[i] = false;

BFS(1);

return 0;

}

**<재귀함수를 통한 경로탐색> : 그래프는 인접행렬 예시그림으로 구현**

#include<stdio.h>

#include<stack>

#include<queue>

bool visit[5];

bool adj[5][5];

int cnt = 0;

using namespace std;

void DFS(int v, int end){

if (v == end) {

cnt++;

return;

}

for (int i = 1; i <= 6; i++)

{

if (adj[v][i] && !visit[i])

{

//정점이 곂지지 않게 visit = true

visit[i] = true;

DFS(i, end);

//한 가지의 경로탐색이 끝나면 원상태로 복구

visit[i] = false;

}

}

}

int main()

{

for (int i = 1, s, e; i <= 6; i++) {

scanf("%d %d", &s, &e);

adj[s][e] = true;

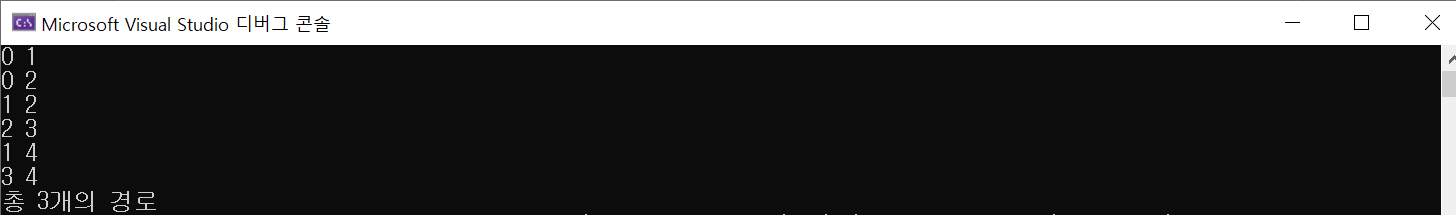
}

DFS(0, 4);

printf("총 %d개의 경로", cnt);

return 0;

}



Codeup

4503 : 바이러스

#include<stdio.h>

bool adj[101][101];

bool visit[101];

int n, m;

void DFS(int v)

{

for (int i = 1; i <= n; i++){

if (adj[v][i] == true && !visit[i]){

visit[i] = true;

DFS(i);

}

}

}

int main()

{

scanf("%d\n%d", &n, &m);

for (int i = 0, r, c; i < m;i++) {

scanf("%d %d", &r, &c);

adj[r][c] = adj[c][r] = true;

}

DFS(1);

int sum = 0;

for (int i = 2; i <= n;i++) {

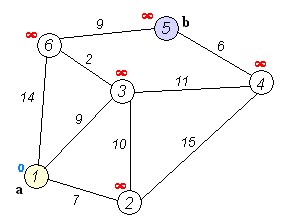
if (visit[i])

sum++;

}

printf("%d", sum);

}

**오른쪽의 그림은 앞으로 작성할 최단거리 알고리즘의 예시 그림이다.**

**도달할 수 없는 거리를 무한대로 표현한다.  
 <출처 : 위키백과>**

**<최단거리 알고리즘 : 다익스트라>**

**한 정점으로부터 나머지 정점에 대한 최단거리를 구하는 알고리즘**

//Dijkstra algorithm

//정점간의 최단경로를 구하는 알고리즘, 시간복잡도 O(n^2) n = 정점의 수

#include<stdio.h>

#define INF 100000000 //infinite(무한) - 도달할 수 없는 거리를 의미

int vertex[6][6] = {

{0,7,9,INF,INF,14},

{7,0,10,15,INF,INF},

{9,10,0,11,INF,2},

{INF,15,11,0,6,INF},

{INF,INF,INF,6,0,9},

{14,INF,2,INF,9,0}

};

bool visit[6];

int distance[6];

void Dijkstra(int start) {

for (int i = 0; i < 6;i++) {

distance[i] = vertex[start][i];

}

visit[start] = true;

for (int i = 0, cur; i < 6 - 2; i++) {//정점이 2개남았을 때까지 수행(2개남았으면 최소한의 간선만 존재 -> 거리check완료)

cur = 0;

for (int j = 0, min = INF; j < 6; j++) {//최소비용의 정점을 찾는 부분

if (distance[j] < min && !visit[j]) {

min = distance[j];

cur = j;

}

}

visit[cur] = true;

for (int k = 0; k < 6;k++) {

if (!visit[k] && distance[k] > distance[cur] + vertex[cur][k]) {

distance[k] = distance[cur] + vertex[cur][k];

}

}

}

}

int main(void) {

Dijkstra(0);

for (int i = 0; i < 6;i++) {

printf("%d : %d\n", i + 1, distance[i]);

}

return 0;

}



**<최단거리2 : Floyd\_Warshal algorithm>**

**정점간의 모든 최단경로를 구하는 알고리즘**

//Floyd\_Warshal algorithm

//정점간의 최단경로를 구하는 알고리즘, 시간복잡도 O(n^3) n = 정점의 수

#include<stdio.h>

#define INF 100000000

int vertex[6][6] = {

{0,7,9,INF,INF,14},

{7,0,10,15,INF,INF},

{9,10,0,11,INF,2},

{INF,15,11,0,6,INF},

{INF,INF,INF,6,0,9},

{14,INF,2,INF,9,0}

};

bool visit[6];

int distance[6];

void Floyd\_Warshal() {

for (int k = 0; k < 6;k++) {

for (int i = 0; i < 6;i++) {

for (int j = 0; j < 6;j++) {

if (vertex[i][k] == INF || vertex[k][j] == INF || i == j) {

continue;

}

if (vertex[i][j] > vertex[i][k] + vertex[k][j])

vertex[i][j] = vertex[i][k] + vertex[k][j];

}

}

}

}

int main(void) {

Floyd\_Warshal();

for (int i = 0; i < 6;i++) {

for (int j = 0; j < 6;j++) {

printf("%3d ", vertex[i][j]);

}

printf("\n");

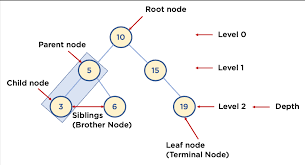
}

return 0;

}



**트리**

트리란 그래프의 한 종류로, 계층구조를 표현하다. 자신을 가리키는 노드가 하나여야 하고, 같은높이에 있는 왼쪽 노드와 오른쪽 노드를 구분한다.

오른쪽의 그림과 함께 보자. 최상위 계층에 있는 노드(시작하는 노드) 뿌리 노드, Root 노드라고 한다.

자신의 노드를 가리키는 노드를 부모노드라고 하며, 자식노드는 left, right로 구분한다.

Tree는 level로 높이를 따진다. Level은 0에서 시작하며 2로 끝나니 오른쪽 그림에서 높이는 3이다.

**<포인터를 통해 tree 구조 구현>**

#include<stdio.h>

#include<stdlib.h>

using namespace std;

struct node{

int node;

struct node\* left\_child;

struct node\* right\_child;

};

void preorder(node\* root) {

printf("%d", root->node);

if (root->left\_child != NULL)

preorder(root->left\_child);

if (root->right\_child != NULL)

preorder(root->right\_child);

}

int main() {

node\* n1, \* n2, \* n3, \* n4, \* n5, \* n6;

n1 = (node\*)malloc(sizeof(node)); n2 = (node\*)malloc(sizeof(node));

n3 = (node\*)malloc(sizeof(node)); n4 = (node\*)malloc(sizeof(node));

n5 = (node\*)malloc(sizeof(node)); n6 = (node\*)malloc(sizeof(node));

n1->node = 10; n2->node = 5;

n3->node = 15; n4->node = 3;

n5->node = 6; n6->node = 19;

n1->left\_child = n2; n1->right\_child = n3;

n2->left\_child = n4; n2->right\_child = n5;

n3->left\_child = NULL; n3->right\_child = n6;

n4->left\_child = NULL; n4->right\_child = NULL;

n5->left\_child = NULL; n5->right\_child = NULL;

n6->left\_child = NULL; n6->right\_child = NULL;

preorder(n1);

}

텍스트이(가) 표시된 사진

자동 생성된 설명

**<배열을 통한 트리구조 구현>**

#include<stdio.h>

int tree[10];

void preorder(int n) {

printf("%d번 노드 : %d\n",n, tree[n]);

if (tree[2 \* n])//left child

preorder(2 \* n);

if (tree[2 \* n + 1])//right child

preorder(2 \* n + 1);

}

int main() {

tree[1] = 10;

tree[2] = 5;

tree[3] = 15;

tree[4] = 3;

tree[5] = 6;

tree[7] = 19;

preorder(1);

}

텍스트이(가) 표시된 사진

자동 생성된 설명

**HEAP**

Binary tree의 한 종류로 자식노드가 2개 혹은 0개인 모양을 가진다 또한, 기존 tree에서 규칙이 생긴다. 보통 MIN or MAX와 같은 규칙을 따르며 MAX인 경우에는 부모노드가 자식노드보다 크거나 같아야 한다. ROOT 자리에는 당연히 가장 큰 값이 온다. heap구조에서 데이터의 삽입 제거는 모두 root 자리에서 일어나고, 데이터를 삽입 제거했을 때, 업데이트해서 heap구조를 완성해야 한다.

**<MAX HEAP>**

#include<stdio.h>

int heap\_size = 8;//높이 = 8

void push(int val, int heap[]) { //val을 넣을 노드를 찾는 과정

int index = ++heap\_size;

while (index > 1 && val > heap[index / 2]) {

heap[index] = heap[index / 2];

index /= 2;

}

heap[index] = val;

}

int pop(int heap[]) {//pop한 뒤 heap update

int re = heap[1];

heap[1] = heap[heap\_size--];

int p = 1, c = 2, temp;

while (c <= heap\_size) {

if (heap[c] < heap[c + 1]) {//c를 left, right 중에 더 큰 노드로 지정

c++;

}

if (heap[p] >= heap[c]) {

break;

}

temp = heap[p];

heap[p] = heap[c];

heap[c] = temp;

p = c;

c = c \* 2;

}

return re;

}

void preorder(int heap[]) {

int cur = 1;

for (int i = 1; i <= heap\_size; i++) {

if (i == cur) {

cur \*= 2;

printf("\n");

}

printf("%4d", heap[i]);

}

}

int main() {

int heap[16] = { 44, 42, 35, 33, 31, 19, 27, 10, 26 };

preorder(heap);

push(50,heap);

preorder(heap);

pop(heap);

preorder(heap);

}

텍스트이(가) 표시된 사진

자동 생성된 설명

**정렬**

효율적인 탐색, 데이터에 대한 여러 알고리즘을 사용하기 위해선 정렬시켜야 한다. 지금까지 배운 자료구조를 정렬시키는 방법에 대해 배워보자. 정렬을 할 때, 쓰는 알고리즘에 따라 시간복잡도와 공간복잡도의 차이가 발생하고, 정렬하는 알고리즘을 발전시키려고 많은 노력을 해왔다. 가장 쉬운 정렬방법부터 어려운 정렬알고리즘까지 살펴보자.

**<bubble sort>**

1. 왼쪽값과 오른쪽값을 비교한 뒤, 큰 값이 오른쪽에 위치하도록 바꾼다.

2. 해당 과정을 배열의 크기만큼 반복한다.

#include<stdio.h>

int main()

{

int arr[10] = { 1, 7, 5, 9 ,3 ,10, 2, 4, 8, 6 };

int max\_index, temp;

for (int i = 0; i <= 9; i++)

{

for (int j = 0; j < 9 - i; j++)//큰 값을 오른쪽으로 위치

{

if (arr[j] > arr[j + 1])

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

for (int i = 0; i < 10; i++)

{

printf("%d ", arr[i]);

}

}

**<Selection sort> : 내림차순**

1. 범위안에 가장 큰 값을 찾는다.

2. 해당 값을 범위에 가장 왼쪽과 바꾼다.

3. 데이터의 크기 만큼 1번 2번 과정을 반복한다.

#include<stdio.h>

int main()

{

int arr[10] = { 1, 7, 5, 9 ,3 ,10, 2, 4, 8, 6 };

int max\_index, temp;

for (int i = 0; i < 9; i++)

{

max\_index = i;

for (int j = i + 1; j < 10; j++)

{

if (arr[j] > arr[max\_index])

{

max\_index = j;

}

}

temp = arr[i];

arr[i] = arr[max\_index];

arr[max\_index] = temp;

}

for (int i = 0; i < 10; i++)

{

printf("%d ", arr[i]);

}

}

**<Quick sort> :** pivot을 통해 분할해가는 알고리즘

1. Pivot을 선택하고 pivot 보다 작은 값은 왼쪽으로, 큰 값은 오른쪽으로 이동시키고 pivot을 가운데로 이동시킨다.

2. Pivot 기준으로 왼쪽에 대해서도 quick sort 시키고, 오른쪽에 대해서도 quick sort를 시킨다.

3. 1번과 2번과정을 더 이상 분할이 불가능할 때까지 반복

#include<stdio.h>

int arr[1000000];

void quicksort(int left, int right)

{

if (left >= right)

return;

int l = left + 1; int r = right;

int pivot = left;

int temp;

while (l <= r)

{

for (;arr[l] < arr[pivot] && l <= right; l++);

for (;arr[r] > arr[pivot] && r > left; r--);

if (l < r) {

temp = arr[l];

arr[l] = arr[r];

arr[r] = temp;

l++;r--;

}

}

temp = arr[pivot];

arr[pivot] = arr[r];

arr[r] = temp;

pivot = r;

quicksort(left, pivot - 1);

quicksort(pivot + 1, right);

}

int main()

{

int n;

scanf("%d", &n);

for (int i = 0; i < n;i++)

scanf("%d", &arr[i]);

quicksort(0, n - 1);

for (int i = 0; i < n; i++)

printf("%d ", arr[i]);

}

**<Merge sort>**

1. 더 이상 쪼개지기 않을 때까지 분할

2. 2개를 하나의 단위로 병합시키면서 정렬시킴

3. 2번과정을 반복하여 원래 크기의 단위까지 병합

#include<stdio.h>

#include<stdlib.h>

void merge(int arr[], int left, int mid, int right) // 병합하는 함수

{

int size = right - left + 1;

int l\_ptr = left, r\_ptr = mid + 1;

int index = 0; //temp배열의 index

int\* ptr;

// 메모리 최소화를 위해 temp 배열을 동적할당

ptr = (int\*)malloc(sizeof(int) \* (right - left + 1));

//두 부분을 비교해서 작은 순으로 ptr에 정렬(둘중에 한 부분이 끝나면 끝)

while (l\_ptr <= mid && r\_ptr <= right) {

if (arr[l\_ptr] <= arr[r\_ptr]) ptr[index++] = arr[l\_ptr++];

else ptr[index++] = arr[r\_ptr++];

}

while (l\_ptr <= mid) ptr[index++] = arr[l\_ptr++];// 남은 부분 ptr에 집어넣기

while (r\_ptr <= right) ptr[index++] = arr[r\_ptr++];

for (int i = left, j = 0; i <= right; i++, j++)

arr[i] = ptr[j]; //배열에 정렬한 부분 집어넣기

}

void merge\_sort(int arr[], int left, int right)

{

int mid = (left + right) / 2;

if (left < right) {

merge\_sort(arr, left, mid);//분할하는 부분

merge\_sort(arr, mid + 1, right);

merge(arr, left, mid, right);//합병

}

}

int main()

{

int n;

scanf("%d", &n);

int\* arr = (int\*)malloc(sizeof(int) \* n); // 동적할당 (배열)

for (int i = 0; i < n; i++)

scanf("%d", &arr[i]);

merge\_sort(arr, 0, n - 1);

for (int i = 0; i < n; i++)

printf("%d ", arr[i]);

return 0;

}

**<Heap sort>**

1. root를 하나씩 가장 오른쪽 leaf랑 바꾸기

2. 다시 max heap 구조형성

3. 해당 과정을 root 까지 반복

#include<stdio.h>

int heap[1000001];

void insert(int v, int size)//새로운 값 삽입

{

int i;

i = size;

while ((i > 1) && (v > heap[i / 2])) {

//한 단계높은 level 위치로 이동하기 위해 2로 나눔 = v 노드의 자리를 한 칸 위로 올림

heap[i] = heap[i / 2];

i /= 2;

}

heap[i] = v;

}

void Hsort(int size)

{

int p, c, temp;

temp = heap[size];

heap[size--] = heap[1];

//후위연산자, 배열의 마지막(== 트리의 가장 오른쪽 leaf)에 root를 이동시킨 후, size를 감소시킴

heap[1] = temp;

p = 1; c = 2;//부모, 자식노드를 대신하는 변수생성

//root자리에 leaf의 값이 들어왔으니, max\_heap구조완성을 위해 update가 필요.

while (c <= size) {//max\_heap update 과정

//root부터 내려갈 건데, c중에 큰 값 이랑 p랑 비교를 위해

if (c < size&& heap[c + 1] >= heap[c])

c++;

if (temp >= heap[c])//p가 c보다 클 때, 즉, 맞는 위치일때

break;

heap[p] = heap[c];//c를 p로올린다.

p = c;

c \*= 2;

}

heap[p] = temp;

}

int main()

{

int n;

scanf("%d", &n);

for (int i = 1, temp; i <= n; i++) {

scanf("%d", &temp);

insert(temp, i);

}

for (int i = n; i > 1;i--)

Hsort(i);

for (int i = 1; i <= n; i++)

printf("%d ", heap[i]);

return 0;

}

**<STL SORT>**

정렬알고리즘을 직접 구현하지 말고, STL라이브러리에서 가져다 쓰면 된다.

#include<stdio.h>

#include<algorithm>

//내가 쓰기 편하게 사람이라는 데이터 타입을 만듬

struct human {

int age;//나이

int height;//키

};

//bool : 논리연산자 반환형 = true or false

bool compare(human a, human b)

{ //나이순으로 정렬시키고, 나이가 같으면 키순으로 정렬

if (a.age != b.age) {

if (a.age < b.age)

return true;

else

return false;

}

else {

if (a.height < b.height)

return true;

else

return false;

}

}

int main()

{

int num[10] = { 3,1,6,2,5,10,9,4,8,7 };

printf("=======================================\n");

for (int i = 0; i < 10;i++) {

printf("%d ", num[i]);

}

printf("\n=======================================\n");

//정렬할 시작위치, 끝 위치 + 1

//비교함수를 안두면 오름차순으로 정렬시킨다.

std::sort(num, num + 10);

for (int i = 0; i < 10;i++) {

printf("%d ", num[i]);

}

printf("\n=======================================\n");

human arr[5];

for (int i = 0; i < 5;i++) {

scanf("%d %d", &arr[i].age, &arr[i].height);

}

printf("=======================================\n");

//정렬하는 기준을 compare을 통해 정해줌

std::sort(arr, arr + 5, compare);

for (int i = 0; i < 5;i++) {

printf("%d %d\n", arr[i].age, arr[i].height);

}

return 0;

}

텍스트이(가) 표시된 사진

자동 생성된 설명

백준 : 좌표압축(18870)

#include<iostream>

#include<vector>

#include<algorithm>

using namespace std;

bool compare1(pair<int, int> a, pair<int, int> b) {

if (a.first != b.first)

return a.first < b.first;

return a.second < b.second;

}

bool compare2(pair<int, int> a, pair<int, int> b) {

return a.second < b.second;

}

int main() {

int n;

cin >> n;

vector<pair<int, int>> vec;

for (int i = 0, x; i < n;i++) {

cin >> x;

vec.push\_back(make\_pair(x, i));

}

sort(vec.begin(), vec.end(), compare1);

int k = vec[0].first;

for (int i = 1, c = 0; i < n;i++) {

if (k != vec[i].first) {

k = vec[i].first;

vec[i].first = ++c;

}

else {

vec[i].first = c;

}

}

vec[0].first = 0;

sort(vec.begin(), vec.end(), compare2);

for (int i = 0; i < n;i++) {

cout << vec[i].first << " ";

}

}

**번외)**

**<STL MAP>**

#include <iostream>

#include <map>

using namespace std;

int main() {

map<string, int> m;//pair 형태

string temp;

int n\_temp;

for (int i = 0; i < 5; i++) {

cin >> temp >> n\_temp;

m.insert(make\_pair(temp, n\_temp));

}

if (m.find("abcd") != m.end()) {

//이런식으로 사용가능

m["abcd"]++;

}

for (auto i = m.begin(); i != m.end(); i++) {

cout << i->first << " " << i->second << '\n';

}

for (auto iter : m) {

cout << iter.first << " " << iter.second << endl;

}

return 0;

}