최근 동적할당에 대한 글들이 몇건이 올라오는 것 같아

동적할당 대신 제가 주로 사용하는 배열에 대해 알려드립니다.

쓱 한번 읽어 보시고, 본인 스타일에 잘 맞다고 생각하시는 분만 사용하시는 것을 추천 드립니다.

한글보다 쉬운 C 언어로 동적할당 대신 사용할 코드를 보여드리면.

```
int arr_idx = 0;


struct NODE {
    int v;
} a[10000];


NODE * myalloc(void)
{
    return &a[arr_idx++];


}
```

매 case 가 시작할 때마다 arr_idx = 0 으로 초기화만으로 충분합니다.

당연히 NODE 구조체에 변수를 추가하여 여러가지 사항을 만들어서 사용하실 수도 있습니다.

가장 많이 사용하는 single linked list 는 아래와 같이 사용 할 수 있습니다.

```
int arr_idx = 0;


struct NODE {

    int v;
    NODE * prev;  //싱글 리스트를 위해 추가.
} a[10000];

NODE * myalloc(void)

{

    return &a[arr_idx++];

}

void main(void)
{
    NODE * pList = NULL; // 싱글 링크드 리스트의 시작
    NODE * p;

    arr_idx = 0;  // 배열 초기화

    //첫번째 노드(1) 추가

    p = myalloc();
    p->v = 1;
    p->prev = pList;

    pList = p;

    //두번째 노드(2) 추가
    p = myalloc();
    p->v = 2;
    p->prev = pList;
    pList = p;

    //추가된 노드 확인

    for(NODE * pp = pList; pp != NULL; pp = pp->prev)
    {
        cout << pp->v << " ";
    }

```

```
57
58}
59
60
```

코드에서 노드를 추가하는 두개의 코드가 값(v)을 넣는 것을 빼고는 동일한 것을 보셨을 껍니다.

C++ 의 생성자를 활용해서 이부분을 한출로 줄이는 법도 있습니다만, 이건 다음기회로~~

그럼 malloc 과 array 의 속도를 비교해보면 어떨까요?

당연히 array 를 사용하는 것이 현저히 빠릅니다.

1000000 번 정도의 반복 테스트를 해본 결과

array 가 debug 에서는 약 5 배, release 에서는 2 배 정도 빠르네요.

속도 테스트 코드는 아래와 같습니다.

```
 NODE * p;
 clock_t start;


 cout << "USING ARRAY : ";
 start = clock();
 for (int i = 0; i < 1000000; i++)
 {
  p = myalloc();
  p->v = i;
  p->prev = pList;
  pList = p;
 }
 cout << clock() - start << endl;


 pList = NULL;
 cout << "USING ALLOC : ";
 start = clock();
 for (int i = 0; i < 1000000; i++)
 {
  p = (NODE*) malloc(sizeof(NODE));
  p->v = i;
  p->prev = pList;
  pList = p;
 }
 cout << clock() - start << endl;

```