**CSED211 컴퓨터SW시스템개론**

**Lab Assignment #5: Cache Lab:**

**Understanding Cache Memories**

**20180551**

**컴퓨터공학과**

**이준석**

**[Part A]**

**캐시 시뮬레이터를 만드는 부분이다. csim.c 파일을 수정하여 만든다.**

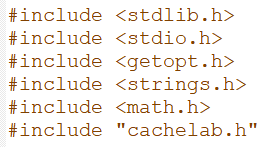


그림 1

포함되어 있는 헤더 파일들이다.

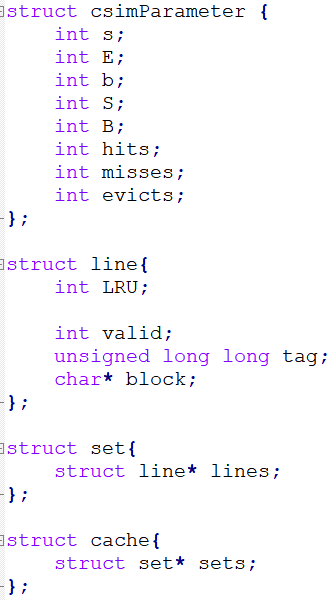
stdlib.h : 동적 할당

stdio.h : 입출력

getopt.h : getopt 함수를 호출하여 커맨드 라인에 입력된 것을 파싱한다.

strings.h : string을 int로 바꿔주는 atoi 함수를 호출하여 사용한다.

math.h : pow 함수를 호출하여 사용한다. set bit로 set의 개수를 구할 때 사용된다.



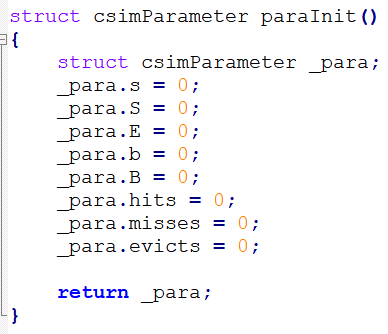
4개의 구조체를 선언하였다.

csimParameter : 캐시 크기와 관련된 5개의 문자( s, E, b, S, B )와 hit, miss, evict 횟수를 세기 위한 변수 hits, misses, evicts를 포함하였다.

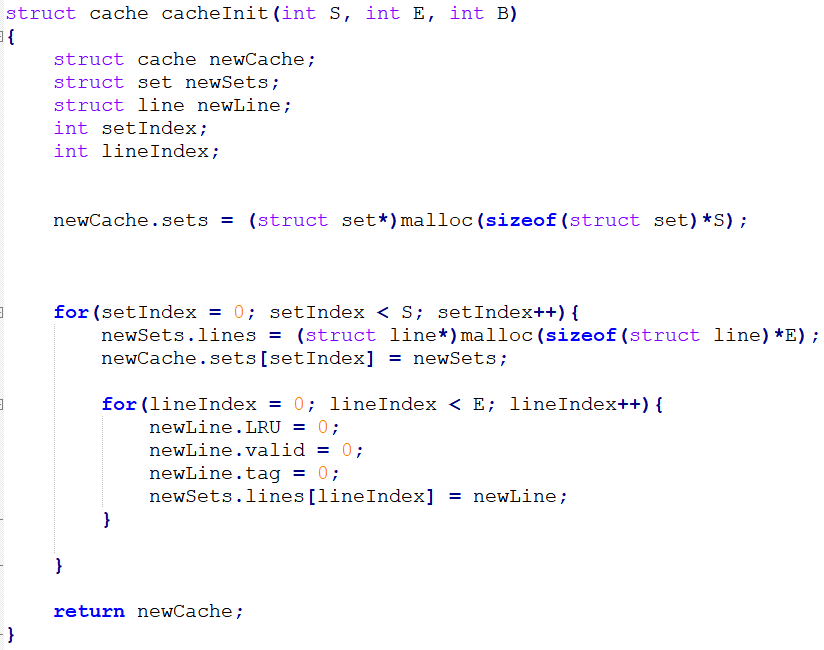
line : LRU에 LRU를 위한 카운트를 기록한다. valid, tag, block은 각각 캐시의 그것에 대응한다. 실제로는 캐시의 hit과 miss를 판단할 때 block은 사용되지 않는다.

set : line들을 여러 개 포함하고 있다.

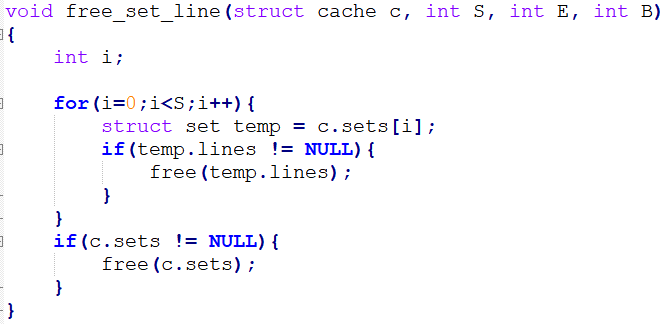
cache : set들을 여러 개 포함하고 있다.



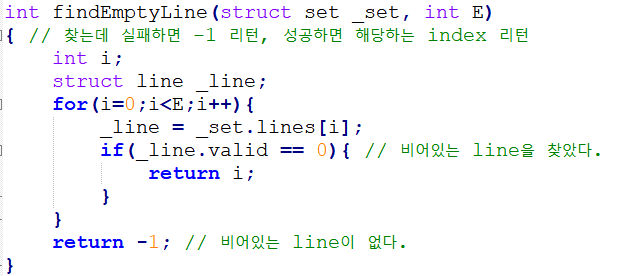
csimParameter를 초기화하는 함수이다.



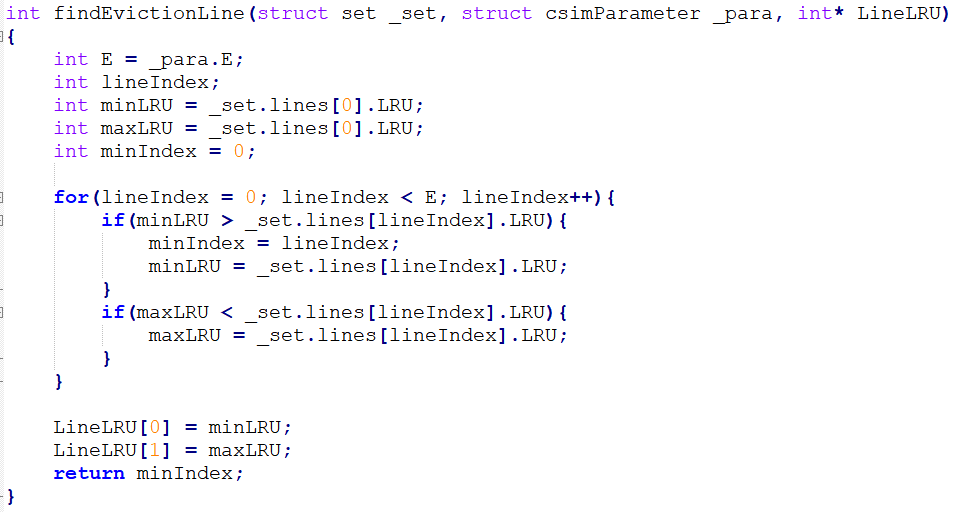
cache를 초기화하는 함수이다. 동적 할당과 값의 초기화가 이루어진다.



동적 할당한 것을 초기화하는 함수이다.

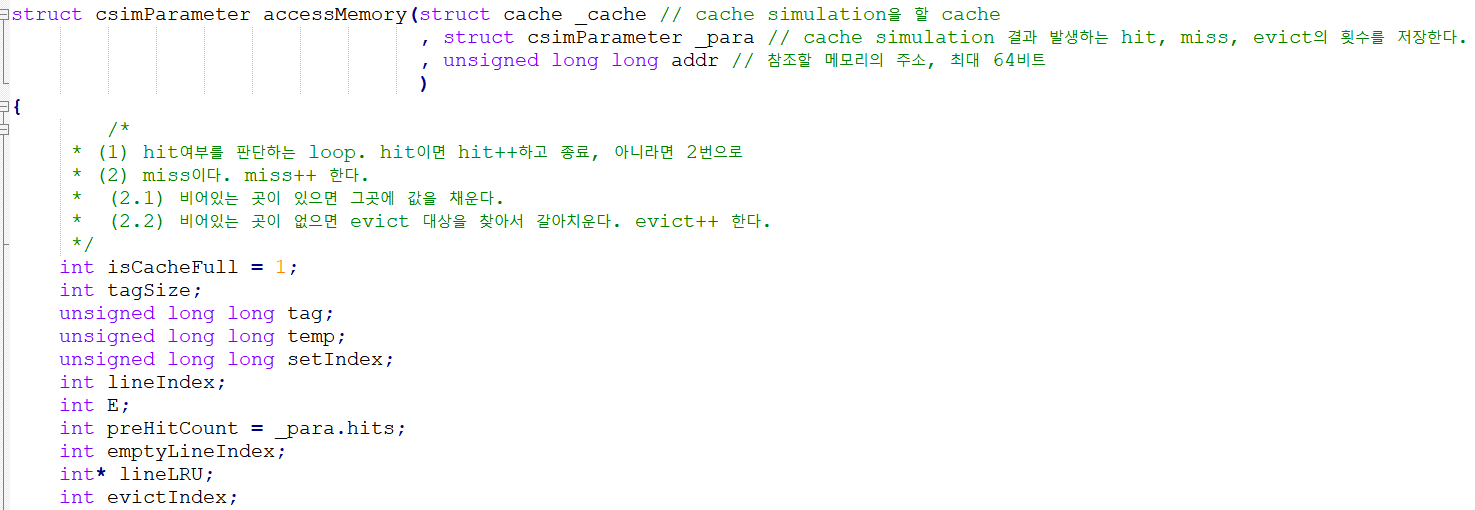


miss가 발생했을 때, 비어 있는 line이 있다면 그곳에 그대로 채워 넣으면 된다. 그래서 그 비어 있는 line을 찾기 위한 함수이다.

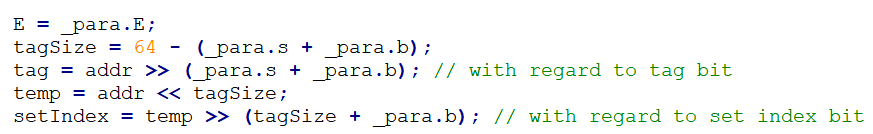


miss가 발생했을 때, set이 가득 차 있다면 evict할 line을 선별해야 한다. 그것을 위한 함수이다.

이것이 실현되는 이유는 LRU라는 변수가 단순하게 line이 하나 추가 될 때 하나 증가하는 것이 아니기 때문이다.

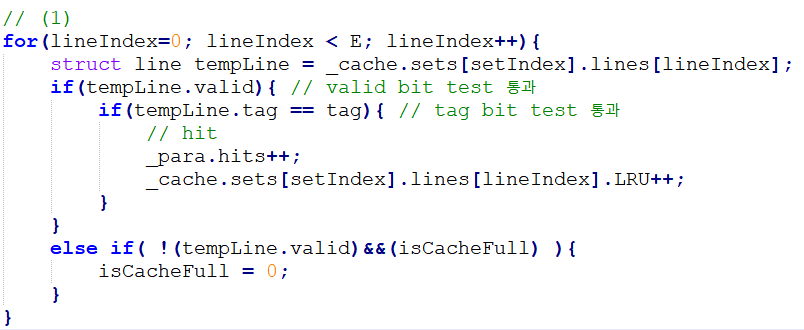


메모리에 접근하는 함수와 그것에서 사용한 지역 변수들이다.



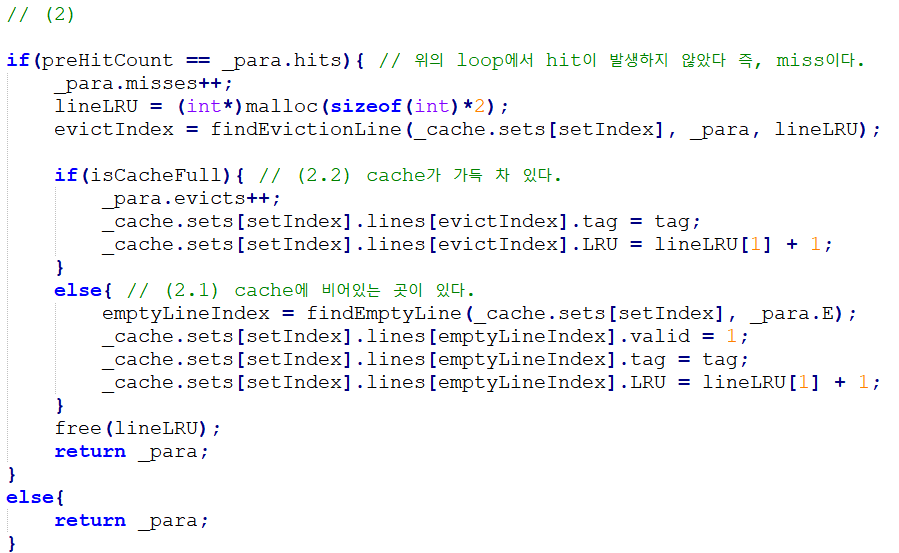
캐시의 hit과 miss는 set을 결정하는 단계와 tag의 일치 여부를 판단하는 단계에 의해서 결정된다.

그러기 위해서 set과 tag를 구했다.



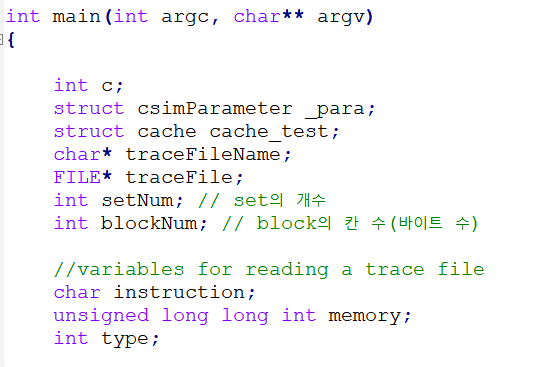
hit인지를 확인하는 단계이다.

set 안에 포함되어 있는 모든 line들을 순회하면서 일치하는 tag가 있는지 확인한다. 물론 valid bit도 켜져(값이 1) 있어야 한다.

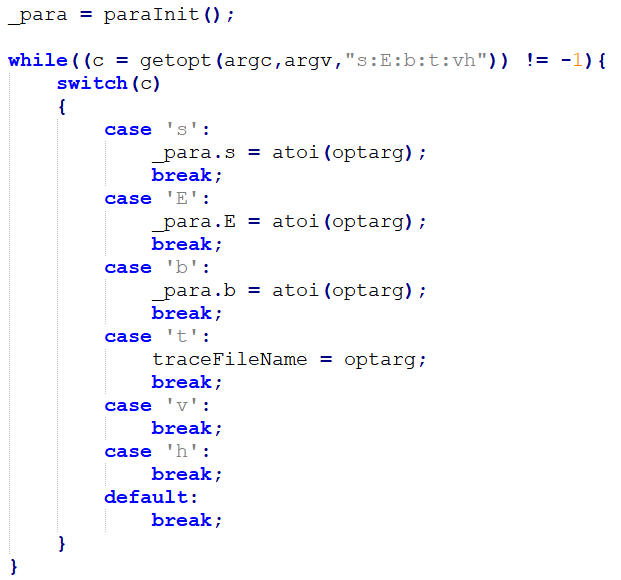


miss일 때, 캐시의 set이 가득 차 있는 경우와 그렇지 않은 경우로 나누어서 생각한다.

가득 차 있지 않으면 비어 있는 그곳에 값을 저장하면 된다. 가득 차 있다면 LRU 방식으로 evict할 line을 선택해서 값을 바꿔 넣는다.

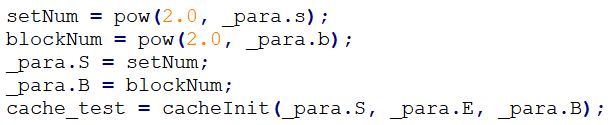


main에서 사용한 변수들이다.

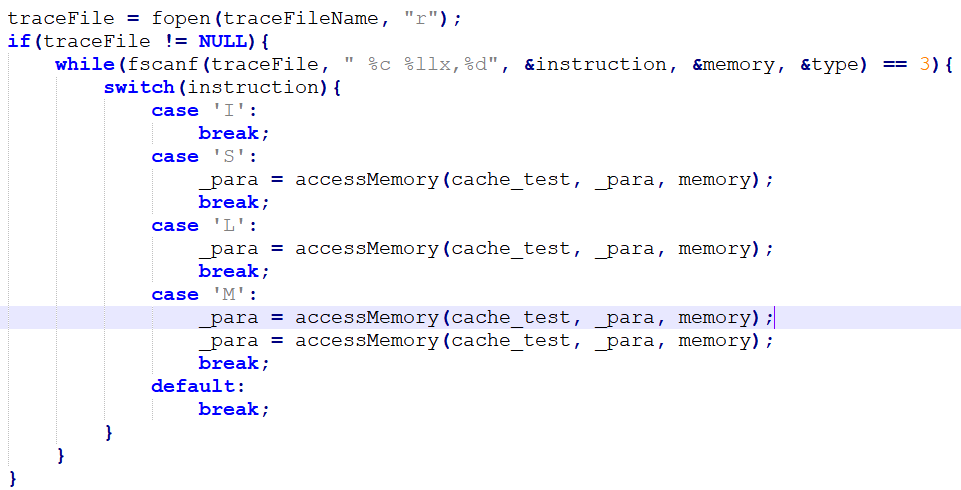


처음에 커맨드라인에 입력한 것을 파싱하는 단계이다. 옵션 파싱을 위해 getopt 함수를 사용하였고 atoi 함수를 사용하여 문자열을 한 번에 정수형으로 바꿔서 저장한다.

트레이스 파일의 이름은 문자열인 상태 그대로 변수에 저장해 놓는다.



캐시의 S와 B를 저장해 놓는 단계이다.



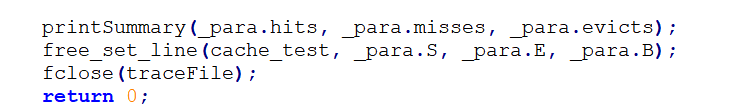
앞서 입력받은 트레이스 파일의 이름을 토대로 트레이스 파일을 읽어오는 단계이다.

참조하는 메모리 주소를 16진수 형태로 받아서 이용하기 쉽도록 하였다.

캐시의 hit과 miss를 판단할 때, hit 발생했을 때 블록을 어떻게 읽느냐 하는 type은 중요하지 않다. 받아만 놓고 사용하지는 않는다.

accessMemory 함수의 실행의 결과 변경된 hit/miss/evict가 \_para에 저장된다.

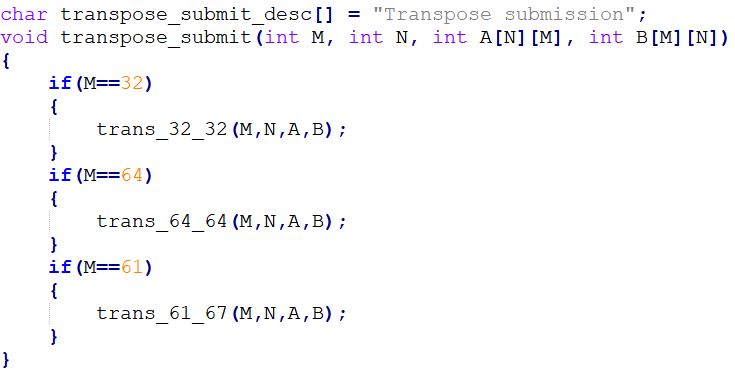
M은 메모리 참조를 2번하므로 accessMemory라는 함수를 두 번 사용하였다.



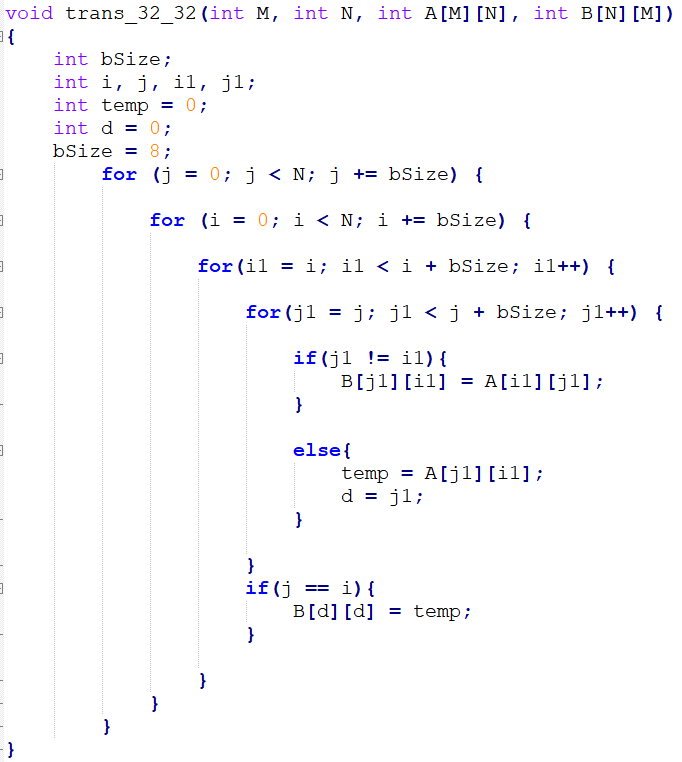
마지막으로 결과를 출력하고 동적 할당한 것을 해제하고 열었던 파일을 닫은 후에 종료한다.

**[Part B]**

**matrix transpose를 cache의 miss가 최대한 적게 발생하도록 구현하는 부분이다.**

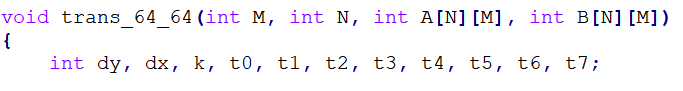


evaluation은 32x32, 64x64, 61x67에 대해 수행된다. 따라서 세 가지 경우로 나누어 구현하였다.

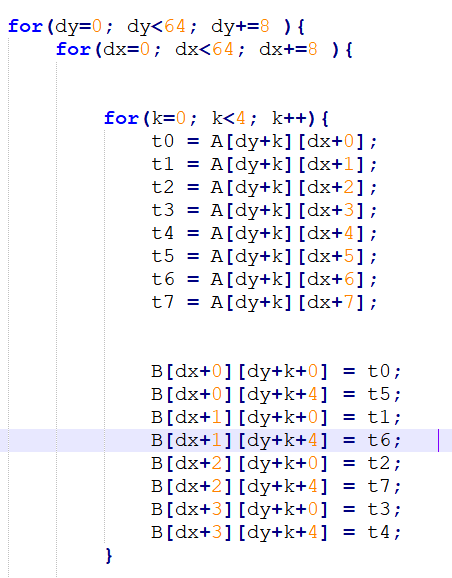


32x32에 대한 함수이다. 8x8 행렬 4개로 나누면 miss를 최소화할 수 있다. 8x8 내부의 원소들이 전부 다른 set에 대응되어 conflict miss가 발생하지 않기 때문이다.

일반적인 transpose를 단순히 8x8 블록화해서 수행하였다. miss를 추가적으로 줄이기 위해 diagonal 부분을 따로 처리하였다.



64x64문제를 해결하기 위한 함수이다. 그리고 사용한 변수들의 모습이다.



그림

8x8 행렬 블록들로 나누어서 생각하였다.

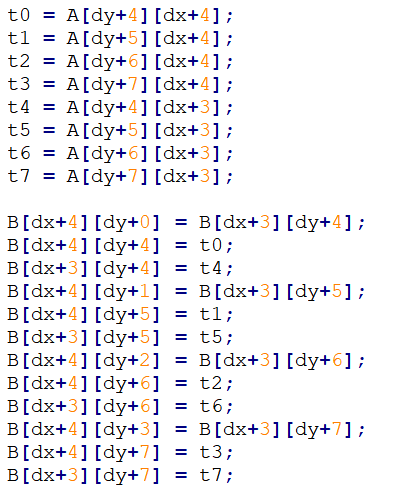
처음에는 64x64를 32x32 4개로 나누어서 위의 함수를 적절히 바꿔서 이용하면 될 것이라고 생각하였으나 set이 겹쳐지는 범위가 그와 달라서 다른 방법을 이용해야 했다.

8x8 행렬 블록 내에서 추가적으로 4x4가 4개가 있다고 간주한다. 그래서 각각의 사분면에 대해 transpose를 수행하게 된다.

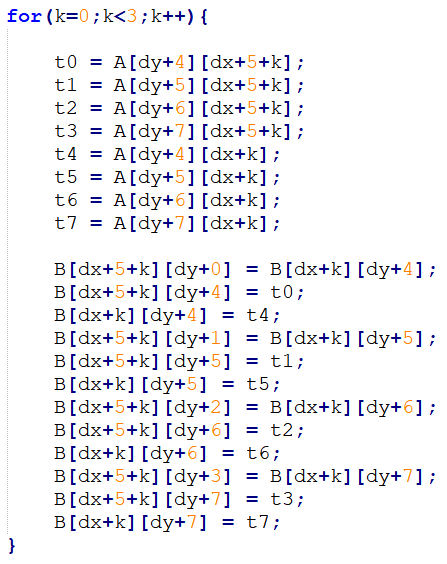
각각의 4x4를 다음과 같이 번호를 매기도록 하자.

1 2

3 4



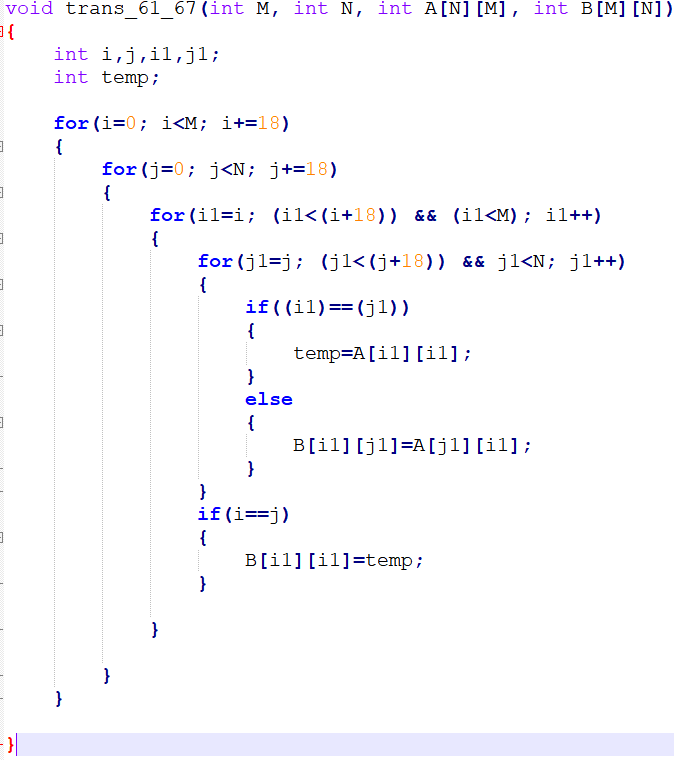
그림



그림

그림 2와 3의 과정을 통해서 1의 전부, 2의 맨 아래, 3과 4의 맨 위 부분이 transpose가 완료된 채로 채워진다.

그림 4의 과정을 통해 남은 2,3,4번 영역의 남은 3개의 행에 대해 transpose가 완료된 것들이 채워진다.



18이라는 블록의 크기는 실험적으로 찾아 내었다. 18개로 나누었을 때 miss의 개수가 조건을 통과하게 된다. 문제를 해결하는 아이디어는 32x32와 같다.