(급조...하려고 했는데 생각보다 오래걸린)
(근데 필요가 없네. 문제 잘못 읽어서 만든!!)

restrict *pointer 이해하기

저자 minckim

Restrict_test.c

```c
#include <stdio.h>

void    f_normal(int *a, int *x)
{
    *a += *x;
    *a += *x;
    *a += *x;
    *a += *x;
}

void    f_restrict(int *restrict a, int *restrict x)
{
    *a += *x;
    *a += *x;
    *a += *x;
    *a += *x;
}

int     main(void)
{
    int     a_n = 1;
    int     a_r = 1;

    f_normal(&a_n, &a_n);
    f_restrict(&a_r, &a_r);
    printf("%d\n",a_n);
    printf("%d\n",a_r);
}
```

이번 과제에 restrict 포인터 어쩌고 저쩌고 하는 부분이 있죠.
몇몇 함수 프로토타입에 restrict가 있는데
쓰지 말라고 하죠.
바보같은 저는 이걸 다 만들고 나서야 쓰지 말라는 것을 이해했습니다——;
음. 그렇지만 언젠가는 도움이 되겠죠ㅎㅎㅎㅎ

~~쓸모없지만~~이것이 무엇인지를 설명해볼까 합니다.(고수님들의 설명 첨언 환영합니다ㅎㅎ)

결론부터 이야기 하자면 restrict는 최적화와 관련 있습니다.

일단은 코드를 따라서 적어봅시다.
두 함수가 거의 같은 코드인데,
위쪽 코드는 restrict 없는 일반적인 코드(대조군)
아래쪽 코드는 restrict 를 추가한 코드입니다.

피신을 통과한 여러분의 내공이라면, 두 함수가 어떻게 작동할지 금방 파악하셨을 거라고 생각합니다.

다만, 함수 인자를 다 같은 것으로 넣은 것이 좀 걸리네요ㅎㅎ

Restrict_test.c

```c
#include <stdio.h>

void    f_normal(int *a, int *x)
{
    *a += *x;
    *a += *x;
    *a += *x;
    *a += *x;
}

void    f_restrict(int *restrict a, int *restrict x)
{
    *a += *x;
    *a += *x;
    *a += *x;
    *a += *x;
}

int     main(void)
{
    int     a_n = 1;
    int     a_r = 1;

    f_normal(&a_n, &a_n);
    f_restrict(&a_r, &a_r);
    printf("%d\n",a_n);
    printf("%d\n",a_r);
}
```

컴파일 하고 실행해 봅시다ㅎㅎ

```
$ gcc -O3 restrict_test.c && ./a.out
16
5
$ gcc restrict_test.c && ./a.out
16
16
```

????

옵션에 따라 실행결과가 다릅니다. 무슨 차이인지를 알아보는 시간을 갖도록 합시다

일단 터미널에서 다음과 같이 입력해 봅시다:

```
$ gcc -g --std=c99 -O3 -c restrict_test.c && objdump -S restrict_test.o > disassembly
```

옵션 설명: (*파란색 이탤릭체*로 된 건 그냥 마음대로 지어도 되는 것들입니다.)
-g : 오브젝트 파일에 코드를 추가
-std=c99 : C 버전을 c99로 함(없어도 결과는 같네요)
-O3 : 어셈블리를 최적화 하는 옵션. 최적화 하는 옵션은 O1, O2, Os, O3가 있습니다. 여기서 O3는 코드 길이 상관없이 최적화를 최대한으로 하는 옵션.
-c : *.c인 파일을 *.o인 오브젝트 파일로 만드는 옵션

&& : (&& 기준)왼쪽의 명령이 성공적으로 수행되었을 때 오른쪽 명령을 수행

objdump : 바이너리 파일을 역어셈블 하라는 명령
-S : 코드와 함께 역어셈블 된 것을 보여줌(앞에서 컴파일 할 때 옵션에 –g를 추가해줬기 때문에 가능)

> : 이전 명령으로 출력될 것들을 *disassembly*라는 파일에 담아주라는 명령어

*하나씩 빼거나 바꿔가면서 해보면 이해가 더 잘 될 것 같아요!

*disassembly*를 열어서 확인해보면 다음과 같은 것들이 뜨는데(길어서 나눴습니다.), 우리에게 중요한 부분은 함수 부분이기때문에 나머지는 지우겠습니다.

```
void          f_normal(int *a, int *x)
{
              *a += *x;
   0:         8b 06                    mov    (%rsi),%eax
   2:         03 07                    add    (%rdi),%eax
   4:         89 07                    mov    %eax,(%rdi)
              *a += *x;
   6:         03 06                    add    (%rsi),%eax
   8:         89 07                    mov    %eax,(%rdi)
              *a += *x;
   a:         03 06                    add    (%rsi),%eax
   c:         89 07                    mov    %eax,(%rdi)
              *a += *x;
   e:         03 06                    add    (%rsi),%eax
  10:         89 07                    mov    %eax,(%rdi)
}
```

```
void          f_restrict(int *restrict a, int *restrict x)
{
              *a += *x;
  20:         8b 06                    mov    (%rsi),%eax
              *a += *x;
              *a += *x;
  22:         c1 e0 02                 shl    $0x2,%eax
              *a += *x;
  25:         01 07                    add    %eax,(%rdi)
}
```

```
void        f_normal(int *a, int *x)
{
            *a += *x;
   0:       8b 06                   mov    (%rsi),%eax
   2:       03 07                   add    (%rdi),%eax
   4:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   6:       03 06                   add    (%rsi),%eax
   8:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   a:       03 06                   add    (%rsi),%eax
   c:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   e:       03 06                   add    (%rsi),%eax
  10:       89 07                   mov    %eax,(%rdi)
}
```

9줄

```
void        f_restrict(int *restrict a, int *restrict x)
{
            *a += *x;
  20:       8b 06                   mov    (%rsi),%eax
            *a += *x;
            *a += *x;
  22:       c1 e0 02                shl    $0x2,%eax
            *a += *x;
  25:       01 07                   add    %eax,(%rdi)
}
```

3줄

훨씬 짧네요!

```
void        f_normal(int *a, int *x)
{
            *a += *x;
   0:       8b 06                   mov    (%rsi),%eax
   2:       03 07                   add    (%rdi),%eax
   4:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   6:       03 06                   add    (%rsi),%eax
   8:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   a:       03 06                   add    (%rsi),%eax
   c:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   e:       03 06                   add    (%rsi),%eax
  10:       89 07                   mov    %eax,(%rdi)
}
```

mov는 왼쪽의 값을 오른쪽에 저장합니다.

add는 왼쪽의 값만큼 오른쪽 값을 증가시킵니다.

()를 씌우면 그 주소에 있는 값을 참조한다는 의미입니다.

C언어처럼 풀어서 쓰면 다음과 같습니다.

(파란색은 그 단계에서 변하는 변수입니다)

	*a	*x(=*a)	tmp_x
	1	1	?
tmp_x = *x	1	1	1
*a += tmp_x	2	2	1
tmp_x = *x	2	2	2
*a += tmp_x	4	4	2
tmp_x = *x	4	4	4
*a += tmp_x	8	8	4
tmp_x = *x	8	8	8
*a += tmp_x	16	16	8
tmp_x = *x	16	16	16

restrict가 없는 함수에서는

*x를 호출할 때마다 다른 변수에 복사한 후, 갱신된 값을 토대로 계산합니다.

```
void        f_restrict(int *restrict a, int *restrict x)
{
            *a += *x;
  20:       8b 06                   mov    (%rsi),%eax
            *a += *x;
            *a += *x;
  22:       c1 e0 02                shl    $0x2,%eax
            *a += *x;
  25:       01 07                   add    %eax,(%rdi)
}
```

	*a	*x(=*a)	temp_x
	1	1	?
tmp_x = *x	1	1	1
tmp_x <<= 2	1	1	4
*a += tmp_x	5	5	4

restrict가 있는 함수에서는

놀랍게도 상당한 축약이 있었습니다.

1. *x는 *x가 좌변에 오지 않는 한 변하지 않는 값으로 인식
2. {*a += *x}가 4번 반복 된 것을 {*a += (*x) * 4}로 변환
3. * 4 를 <<2 (비트연산)으로 변환

```
void        f_normal(int *a, int *x)
{
            *a += *x;
   0:       8b 06                   mov    (%rsi),%eax
   2:       03 07                   add    (%rdi),%eax
   4:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   6:       03 06                   add    (%rsi),%eax
   8:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   a:       03 06                   add    (%rsi),%eax
   c:       89 07                   mov    %eax,(%rdi)
            *a += *x;
   e:       03 06                   add    (%rsi),%eax
  10:       89 07                   mov    %eax,(%rdi)
}
```

```
void        f_restrict(int *restrict a, int *restrict x)
{
            *a += *x;
  20:       8b 06                   mov    (%rsi),%eax
            *a += *x;
            *a += *x;
  22:       c1 e0 02                shl    $0x2,%eax
            *a += *x;
  25:       01 07                   add    %eax,(%rdi)
}
```

Restrict는 그 주소를 참조할 수 있는 포인터 변수가 한 개 뿐이라는 것을 나타냅니다.
(참고. alias : 같은 주소를 가리키는 포인터들이 있는 상태(?))
(참고. Dangling pointer : 할당 해제된 메모리 공간을 가리키는 포인터)

x라는 주소에 접근할 수 있는 것이 만약 포인터 변수 x 하나 뿐이라면 매번 x를 참조해서 *x를 갱신할 필요가 없습니다.

왜냐하면 *x는 {*x = a + 1}과 같은 경우에만 변할 것이기 때문입니다.
해당 주소에 있는 값을 참조만 하는 경우에는 변할 이유가 없습니다.

그래서 restrict를 붙여서 x를 참조하는 포인터가 하나라고 선언하면, 참조할 때마다 복사하면서 갱신하는 시간을 아낄 수 있기 때문에, restrict를 붙이지 않은 경우에 비해서 실행 속도가 향상됩니다.
게다가 갱신할 필요가 없는 것을 보장받자마자 상당한 축약이 이루어 졌군요!

그런데 사실은, 여기에서는 x에 접근 가능한 포인터가 무려 3개였죠.

저는 컴파일러에게 사기 쳤지만, 컴파일러는 눈치 못 채고 순진하게 최적화를 진행한 것을 볼 수 있습니다. 심지어 실행시간 에러를 출력 하지도 않네요.

Restrict는 gcc에 –O3같은 최적화 옵션을 추가했을 때에만 작동합니다. (테스트 해 본 결과 –O1,2,3,s 모두 비슷합니다.)
이번엔 –O3를 지우고 컴파일 해봅시다.

```
$ gcc -g --std=c99 -c restrict_test.c && objdump -S restrict_test.o > disassembly
```

*disassembly*를 뷰어로 보면 다음과 같습니다.

```
void            f_normal(int *a, int *x)
{
  0:            55              push   %rbp
  1:            48 89 e5        mov    %rsp,%rbp
  4:            48 89 7d f8     mov    %rdi,-0x8(%rbp)
  8:            48 89 75 f0     mov    %rsi,-0x10(%rbp)
                *a += *x;
  c:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 10:            8b 10           mov    (%rax),%edx
 12:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 16:            8b 00           mov    (%rax),%eax
 18:            01 c2           add    %eax,%edx
 1a:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 1e:            89 10           mov    %edx,(%rax)
                *a += *x;
 20:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 24:            8b 10           mov    (%rax),%edx
 26:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 2a:            8b 00           mov    (%rax),%eax
 2c:            01 c2           add    %eax,%edx
 2e:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 32:            89 10           mov    %edx,(%rax)
                *a += *x;
 34:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 38:            8b 10           mov    (%rax),%edx
 3a:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 3e:            8b 00           mov    (%rax),%eax
 40:            01 c2           add    %eax,%edx
 42:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 46:            89 10           mov    %edx,(%rax)
                *a += *x;
 48:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 4c:            8b 10           mov    (%rax),%edx
 4e:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 52:            8b 00           mov    (%rax),%eax
 54:            01 c2           add    %eax,%edx
 56:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 5a:            89 10           mov    %edx,(%rax)
}
```

```
void            f_restrict(int *restrict a, int *restrict x)
{
 5f:            55              push   %rbp
 60:            48 89 e5        mov    %rsp,%rbp
 63:            48 89 7d f8     mov    %rdi,-0x8(%rbp)
 67:            48 89 75 f0     mov    %rsi,-0x10(%rbp)
                *a += *x;
 6b:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 6f:            8b 10           mov    (%rax),%edx
 71:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 75:            8b 00           mov    (%rax),%eax
 77:            01 c2           add    %eax,%edx
 79:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 7d:            89 10           mov    %edx,(%rax)
                *a += *x;
 7f:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 83:            8b 10           mov    (%rax),%edx
 85:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 89:            8b 00           mov    (%rax),%eax
 8b:            01 c2           add    %eax,%edx
 8d:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 91:            89 10           mov    %edx,(%rax)
                *a += *x;
 93:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 97:            8b 10           mov    (%rax),%edx
 99:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 9d:            8b 00           mov    (%rax),%eax
 9f:            01 c2           add    %eax,%edx
 a1:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 a5:            89 10           mov    %edx,(%rax)
                *a += *x;
 a7:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 ab:            8b 10           mov    (%rax),%edx
 ad:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 b1:            8b 00           mov    (%rax),%eax
 b3:            01 c2           add    %eax,%edx
 b5:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 b9:            89 10           mov    %edx,(%rax)
}
```

최적화 옵션(-O3)을 제거했더니 전체적으로 길어졌고,
두 함수가 어셈블리어로 다른 게 없습니다!

즉 두 함수를 실행한 수 결괏값이 같은 이유입니다ㅎㅎ

```
void            f_normal(int *a, int *x)
{
  0:            55              push   %rbp
  1:            48 89 e5        mov    %rsp,%rbp
  4:            48 89 7d f8     mov    %rdi,-0x8(%rbp)
  8:            48 89 75 f0     mov    %rsi,-0x10(%rbp)
                *a += *x;
  c:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 10:            8b 10           mov    (%rax),%edx
 12:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 16:            8b 00           mov    (%rax),%eax
 18:            01 c2           add    %eax,%edx
 1a:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 1e:            89 10           mov    %edx,(%rax)
                *a += *x;
 20:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 24:            8b 10           mov    (%rax),%edx
 26:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 2a:            8b 00           mov    (%rax),%eax
 2c:            01 c2           add    %eax,%edx
 2e:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 32:            89 10           mov    %edx,(%rax)
                *a += *x;
 34:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 38:            8b 10           mov    (%rax),%edx
 3a:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 3e:            8b 00           mov    (%rax),%eax
 40:            01 c2           add    %eax,%edx
 42:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 46:            89 10           mov    %edx,(%rax)
                *a += *x;
 48:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 4c:            8b 10           mov    (%rax),%edx
 4e:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 52:            8b 00           mov    (%rax),%eax
 54:            01 c2           add    %eax,%edx
 56:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 5a:            89 10           mov    %edx,(%rax)
}
```

```
void            f_restrict(int *restrict a, int *restrict x)
{
 5f:            55              push   %rbp
 60:            48 89 e5        mov    %rsp,%rbp
 63:            48 89 7d f8     mov    %rdi,-0x8(%rbp)
 67:            48 89 75 f0     mov    %rsi,-0x10(%rbp)
                *a += *x;
 6b:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 6f:            8b 10           mov    (%rax),%edx
 71:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 75:            8b 00           mov    (%rax),%eax
 77:            01 c2           add    %eax,%edx
 79:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 7d:            89 10           mov    %edx,(%rax)
                *a += *x;
 7f:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 83:            8b 10           mov    (%rax),%edx
 85:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 89:            8b 00           mov    (%rax),%eax
 8b:            01 c2           add    %eax,%edx
 8d:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 91:            89 10           mov    %edx,(%rax)
                *a += *x;
 93:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 97:            8b 10           mov    (%rax),%edx
 99:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 9d:            8b 00           mov    (%rax),%eax
 9f:            01 c2           add    %eax,%edx
 a1:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 a5:            89 10           mov    %edx,(%rax)
                *a += *x;
 a7:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 ab:            8b 10           mov    (%rax),%edx
 ad:            48 8b 45 f0     mov    -0x10(%rbp),%rax
 b1:            8b 00           mov    (%rax),%eax
 b3:            01 c2           add    %eax,%edx
 b5:            48 8b 45 f8     mov    -0x8(%rbp),%rax
 b9:            89 10           mov    %edx,(%rax)
}
```

더 깊이 이해하려다 보니 어셈블리어를 찾고 있는 제 자신을 발견했습니다. 이건 아닌 것 같아서ㅋㅋㅋ
일단 여기까지ㅎㅎ

https://dojang.io/mod/page/view.php?id=760	명불허전 갓딩도장
https://wiki.kl에.org/wiki.php/GccOptimizationOptions	gcc 최적화 관련 설명
https://devanix.tistory.com/188	역어셈블에 관한 자료