4190.103A-001: Programming Practice Lab. 15

Euidong Lee
Junseok Lee
Minhyo Jung
(snucsl.ta@gmail.com)

Systems Software & Architecture Lab.

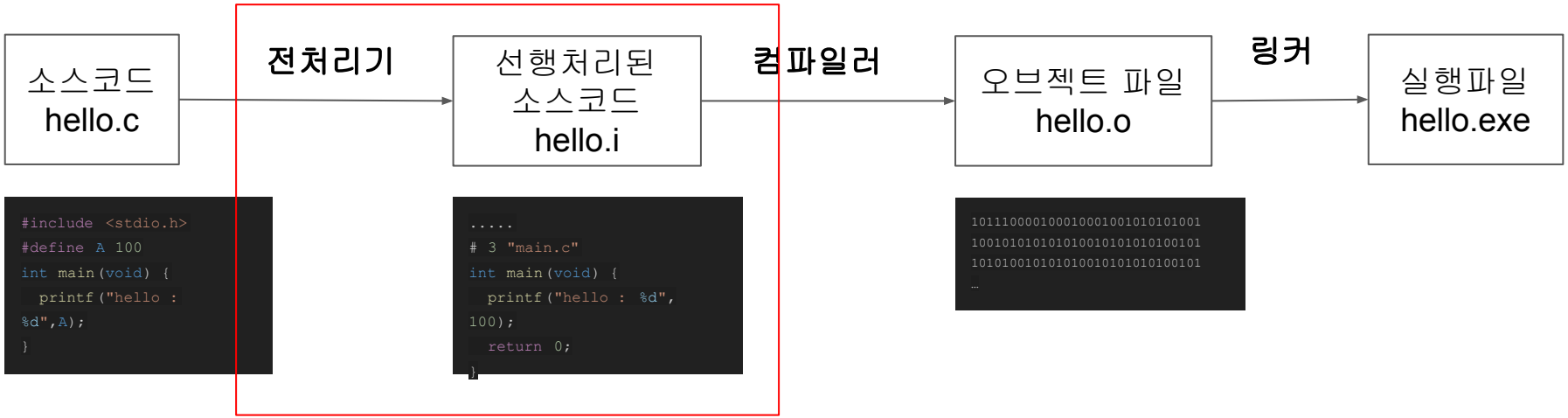
Seoul National University

Spring 2022

실습 1

실습 1 설명 - 전처리기

- 전처리기는 컴파일러 이전에 실행되어 텍스트 치환과 같은 소스 코드의 텍스트를 조작하는 일을 진행함



실습 1 설명 - Object-like Macro (#define)

- 단순한 치환과정을 진행함
- #define ARRAY_NAME “Scores” -> ARRAY_NAME를 “Scores”으로 치환

```c
#include <stdio.h>
#define ARRAY_NAME "Scores"
#define ARRAY_LENGTH 10

int main(void)
{
  int scores[ARRAY_LENGTH] = {0,};

  printf("%s\n", ARRAY_NAME);

  for (int i = 0; i < ARRAY_LENGTH; i++)
     printf("%d\n", scores[i]);
}
```

전처리

```c
int main(void)
{
  int scores[10] = {0,};

  printf("%s\n", "Scores");

  for (int i = 0; i < 10; i++)
     printf("%d\n", scores[i]);
}
```

실습 1 설명 - Function-like Macro (#define)

- #define SQUARE(X) X*X -> SQUARE(X) 패턴을 X*X로 치환

```c
#include <stdio.h>
#define SQUARE(X) X*X

int main(void)
{
  int n = 10;
  float f = 10.5;
  printf("%d*%d = %d\n", n, n, SQUARE(n));
  printf("%f*%f = %f\n", f, f, SQUARE(f));

  // correct?
  printf("%d*%d = %d\n", 10+5, 10+5,
                         SQUARE(10+5));
}
```

전처리

```c
int main(void)
{
  int n = 10;
  float f = 10.5;
  printf("%d*%d = %d\n", n, n, n*n);
  printf("%f*%f = %f\n", f, f, f*f);

  // correct?
  printf("%d*%d = %d\n", 10+5, 10+5,
                         10+5*10+5);
}
```

???

```
> make -s
> ./main
10*10 = 100
10.500000*10.500000 = 110.2500
15*15 = 65 ???
>
```

실습 1 설명 - SQUARE(X) X*X

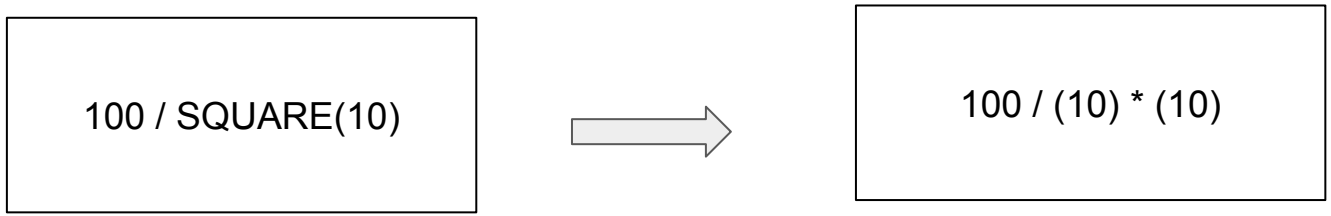
- SQUARE(10+5)가 올바른 값이 나오게 하려면 어떻게 해야될까요?
  - 소괄호를 이용해서 SQUARE(X)를 다시 정의해보세요

```c
#include <stdio.h>
#define SQUARE(X) /* write your code */

int main(void)
{
  int n = 10;
  float f = 10.5;
  printf("%d*%d = %d\n", n, n, SQUARE(n));
  printf("%f*%f = %f\n", f, f, SQUARE(f));

  // correct?
  printf("%d*%d = %d\n", 10+5, 10+5, SQUARE(10+5));
}
```

```
> make -s
> ./main
10*10 = 100
10.500000*10.500000 = 110.250000
15*15 = 225
```

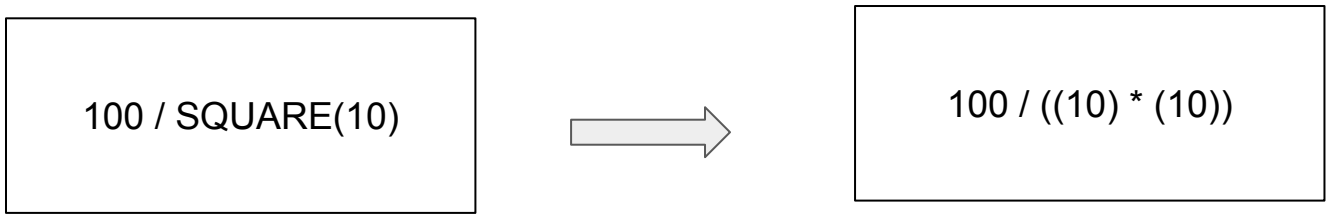
실습 1 설명 - SQUARE(X) (X) * (X)

- #define SQUARE(X) (X)*(X)
  - 괜찮을까요?



실습 1 설명 - SQUARE(X) ((X) * (X))

- #define SQUARE(X) ((X)*(X))



실습 1 설명 - 매크로 사용시 주의사항

- 매크로 사용시 소괄호를 사용하지 않으면 연산자 우선순위 때문에 의도한대로 프로그램이 동작하지 않을 수 있습니다.
- 매크로를 잘 못 사용했을시 디버깅이 쉽지 않습니다.
- 따라서 매크로를 정의할 때 소괄호를 많이 이용하여 의도한 순서대로 연산이 되도록 하는 것이 좋습니다

실습 1 설명 - 이미 정의된 매크로를 이용한 매크로

- 앞서 정의된 매크로를 사용해서 새로운 매크로를 생성할 수 있습니다

```c
#include <stdio.h>
#define KB(X) ((X) * 1024)
#define MB(X) (KB(X) * 1024)
#define GB(X) (MB(X) * 1024)
#define TB(X) (GB(X) * 1024)

int main(void) {
  long n = 1;
  printf("%ldKB : %ldByte\n", n, KB(n));
  printf("%ldMB : %ldByte\n", n, MB(n));
  printf("%ldGB : %ldByte\n", n, GB(n));
  printf("%ldTB : %ldByte\n", n, TB(n));
  return 0;
}
```

실습 1 - MIN(), MAX(), DIV_ROUND_UP()

- 아래 3개의 매크로를 작성해 보세요
  - #define MIN(A,B)
  - #define MAX(A,B)
  - #define DIV_ROUND_UP(A,B)
    - A,B 중 큰 수에서 작은 수로 나누어서 올림함

```c
#define MAX(A,B) /* write your code */
#define MIN(A,B) /* write your code */
#define DIV_ROUND_UP(A,B) /* write your code */

int main(void) {
  int a,b;
  scanf("%d %d",&a, &b);
  printf("DIV_ROUND_UP(%d,%d) = %d\n", a, b, DIV_ROUND_UP(a,b));
  return 0;
}
```

```
> ./main
10 100
DIV_ROUND_UP(10,100) = 10
> ./main
10 109
DIV_ROUND_UP(10,109) = 11
> ./main
109 10
DIV_ROUND_UP(109,10) = 11
```

실습 1 - 풀이

- Tip : 여러 줄에 걸쳐서 매크로를 정의하려면 줄 끝에 ₩(역 슬래시)를 삽입하면 됩니다.

```
#define MAX(A,B) (((A) > (B)) ? (A) : (B))
#define MIN(A,B) (((A) < (B)) ? (A) : (B))

#define DIV_ROUND_UP(A,B) ((MAX(A,B) / MIN(A,B)) \
                          + (MAX(A,B) % MIN(A,B) != 0))
```

실습 2

실습 2 설명 - #if

- 전처리과정에서 #if 조건이 참이라면 #endif까지 컴파일 대상에 포함.
  거짓이라면 컴파일 대상에 포함하지 않음

```c
#define DEBUG 1

int main(void) {
  int a,b;
  scanf("%d %d",&a, &b);

  #if DEBUG
  printf("a: %d b: %d\n", a, b);
  #endif

  return 0;
}
```

DEBUG가 1이라면 어떻게 될까요?

DEBUG가 0이라면 어떻게 될까요?

실습 2 설명 - #elif, #else

- #elif , #else 를 통해 분기를 만들어낼 수 있습니다.

```c
#define DEBUG_LEVEL 0
int main(void) {
  int a = 10, b = 20;

  #if DEBUG_LEVEL == 2
  printf("a: %d b: %d \n", a, b);
  #elif DEBUG_LEVEL == 1
  printf("a: %d\n", a);
  #else
  printf("nothing\n");
  #endif

  return 0;
}
```

DEBUG_LEVEL을 0,1,2로 변경하면서 프로그램을 실행해보세요

실습 2 - multiply()

- USE_ONLY_SUM_OPERATION에 따라 구현부가 다른 multiply 함수를 만들어보세요
  - USE_ONLY_SUM_OPERATION == 1 - > *연산 사용하지 않음
  - USE_ONLY_SUM_OPERATION == 0 -> * 연산 사용

```c
#define USE_ONLY_SUM_OPERATION 1

int main(void) {
  int a,b,result;
  a = 3;
  b = 5;
  result = multiply(a,b);
  printf("%d*%d=%d\n",a,b, result);
  return 0;
}
```

```
> ./main
multiply using sum operation
3*5=15
```

실습 2 - multiply() 풀이

```c
#define USE_ONLY_SUM_OPERATION 1

#if USE_ONLY_SUM_OPERATION
int multiply(int a, int b) {
  printf("multiply using sum operation\n");
  int sum = 0;
  for(int i = 0; i < b; i++)
    sum += a;
  return sum;
}

#else
int multiply(int a, int b) {
  printf("multiply\n");
  return a * b;
}
#endif
```

실습 2 - DEBUG_PRINT()

- 디버깅을 하기 위해서 여러가지 로그들을 프로그램 중간 중간에 삽입합니다.
- 삽입 한 디버깅 로그들은 실제 프로그램 동작하는데는 필요가 없지만 추후 문제가 발생했을시 해당 코드들을 다시 사용할 수도 있습니다.
- 즉 디버깅 로그들을 쉽게 끄고 킬 수 있는 기능을 이용하여, 작성된 디버깅로그들을 관리하는 것은 개발의 효율을 높여줍니다.
- 이러한 기능은 매크로를 통해 구현할 수 있습니다.

실습 2 - DEBUG_PRINT()

- 가변인자를 이용한 디버깅 출력 매크로를 만들어봅시다
  - ... : 가변인자
  - ##__VA_ARGS__ 은 ... (가변인자)에 대응되어 치환됩니다
- 인자의 수를 바꿔가며 실행해보세요

```
#define DEBUG_PRINT(msg, ...)  printf(msg, ##__VA_ARGS__)

int main(void) {
  int a = 10, b = 20;
  DEBUG_PRINT("a: %d b: %d\n", a, b);
  return 0;
}
```

msg ...

```
printf("a: %d b: %d\n", a, b);
```

실습 2 - DEBUG_PRINT()

- 해당 매크로를 호출한 Debugging정보(File, Line, Function)를 같이 출력해보세요
  - __FILE__ : 매크로가 호출된 파일
  - __LINE__ : 매크로가 호출된 줄
  - __FUNCTION__ : 매크로가 호출된 함수
- 연속된 문자열상수는 접합된다는 점을 이용해주세요
  - printf(“abc””def”) -> printf(“abcdef”)

```
#define DEBUG_PRINT(msg, ...) printf("File: %s Line: %d Func: %s\n" msg \
                                    ,__FILE__, __LINE__, __FUNCTION__, ##__VA_ARGS__)
```

실습 2 - DEBUG_PRINT()

```
#define DEBUG_PRINT(msg, ...) printf("File: %s Line: %d func: %s\n" msg \
                                     ,__FILE__, __LINE__, __FUNCTION__, ##__VA_ARGS__)
```

```
DEBUG_PRINT("a: %d b: %d\n", a, b);
```

msg ...

```
printf("File: %s Line: %d func: %s\n" "a: %d b: %d\n"
                                     ,__FILE__, __LINE__, __FUNCTION__, a, b)
```

```
printf("File: %s Line: %d func: %s\n a: %d b: %d\n"
                                     ,__FILE__, __LINE__, __FUNCTION__, a, b)
```

실습 2 - DEBUG_PRINT()

- 눈에 잘 보이도록 초록색으로 출력을 해봅시다
- DEBUG_PRINT()를 호출해도 출력이 안되게 하려면 어떻게 하면 될까요?

```
#define ANSI_COLOR_RED      "\x1b[31m"
#define ANSI_COLOR_GREEN    "\x1b[32m"
#define ANSI_COLOR_YELLOW   "\x1b[33m"
#define ANSI_COLOR_RESET    "\x1b[0m"
#define DEBUG_PRINT(msg, ...)  printf(ANSI_COLOR_GREEN "File: %s Line: %d func: %s\n" msg \
                                      ANSI_COLOR_RESET, __FILE__, __LINE__, __FUNCTION__, ##__VA_ARGS__)
int main(void) {
  int a = 10, b = 20;
  DEBUG_PRINT("DEBUG!! a: %d b: %d \n", a, b);
  return 0;
}
```

실습 2 - DEBUG_PRINT()

- DEBUG_PRINT가 치환되는 부분을 없애주면 됩니다

```c
#define DEBUG_PRINT(msg, ...)

int main(void) {

  int a = 10, b = 20;
  DEBUG_PRINT("DEBUG!! a: %d b: %d \n", a, b);
  return 0;
}
```

실습 2 - DEBUG_PRINT()

- ERROR_PRINT도 만들어봅시다. 빨간색으로 출력되게 해주세요
- DEBUG_OPTION을 통해 DEBUG_PRINT 출력 여부, ERROR_PRINT의 출력여부를 컨트롤해보세요
- DEBUG_OPTION의 0번 bit이 1이면 DEBUG_PRINT 출력, 그렇지않으면 DEBUG_PRINT 출력 안함
- DEBUG_OPTION의 1번 bit이 1이면 ERROR_PRINT 출력, 그렇지 않으면 ERROR_PRINT 출력안함

```c
#define BIT_DEBUG_PRINT  (1<<0)
#define BIT_ERROR_PRINT  (1<<1)
#define DEBUG_OPTION (BIT_DEBUG_PRINT | BIT_ERROR_PRINT)

int main(void) {
  int a = 10, b = 20;
  DEBUG_PRINT("DEBUG!! a: %d b: %d \n", a, b);
  ERROR_PRINT("ERROR!! a: %d b: %d \n", a, b);
  return 0;
}
```

실습 2 - DEBUG_PRINT()

```
#define BIT_DEBUG_PRINT  (1<<0)
#define BIT_ERROR_PRINT  (1<<1)
#define DEBUG_OPTION (BIT_DEBUG_PRINT | BIT_ERROR_PRINT)

#if DEBUG_OPTION & BIT_DEBUG_PRINT
#define DEBUG_PRINT(msg, ...)  printf(ANSI_COLOR_GREEN "File: %s Line: %d func: %s\n" msg \
                                ANSI_COLOR_RESET, __FILE__, __LINE__, __FUNCTION__, ##__VA_ARGS__)
#else
#define DEBUG_PRINT(msg, ...)
#endif

#if DEBUG_OPTION & BIT_ERROR_PRINT
#define ERROR_PRINT(msg, ...)  printf(ANSI_COLOR_RED "File: %s Line: %d func: %s\n" msg \
                                ANSI_COLOR_RESET, __FILE__, __LINE__, __FUNCTION__, ##__VA_ARGS__)
#else
#define ERROR_PRINT(msg, ...)
#endif
```

실습 3

실습 3 설명 - fgetc, fputc

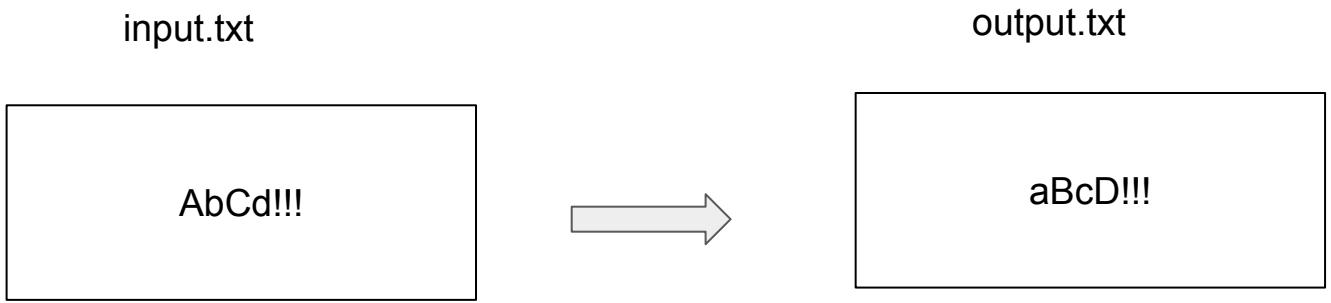
- fgetc : 인자로 받은 파일로 부터 문자 하나를 읽는 함수
- fputc : 인자로 받은 파일로 문자 하나를 쓰는 함수
- EOF : End of File
- 아래 프로그램이 의미하는 것이 무엇일까요?

```c
#include <stdio.h>
#define MAX_FILE_LEN 20
int main(void) {
  FILE * src;
  FILE * dest;
  char src_file_name[MAX_FILE_LEN];
  char dest_file_name[MAX_FILE_LEN];
  int c;
  printf("Source File: ");
  scanf("%s", src_file_name);
  printf("Dest File: ");
  scanf("%s", dest_file_name);
  src = fopen(src_file_name, "r");
  dest = fopen(dest_file_name, "w");

  while ((c = fgetc(src)) != EOF)
    fputc(c, dest);

  fclose(src);
  fclose(dest);

  return 0;
}
```

src file에서 문자 하나를 읽어
dest file에 문자 하나를 씀

실습 3 - 대소문자를 변환하여 저장하는 프로그램

- 텍스트 파일의 이름 두개를 입력받아, 첫 번째 파일의 내용에서 대소문자를 변환하여 두번 째 파일에 저장하는 프로그램을 작성해보세요
  - fgetc(), fputc()를 사용해주세요
  - 알파벳이 아니면 그대로 출력해주세요



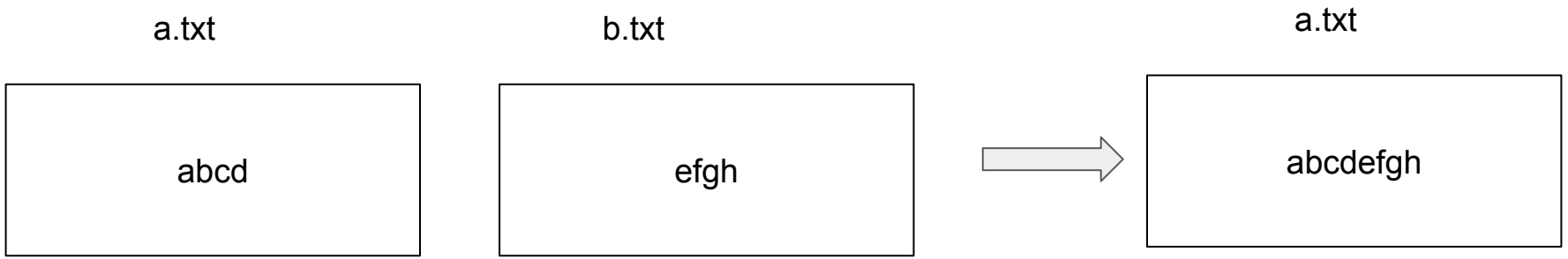
실습 3 - 풀이

```c
#define LOWER_TO_UPPER_OFFSET ('a' - 'A')

char convert_character(char c)
{
  if (c >= 'a' && c <= 'z')
    return c - LOWER_TO_UPPER_OFFSET;
  else if (c >= 'A' && c <= 'Z')
    return c + LOWER_TO_UPPER_OFFSET;

  return c;
}

int main(void) {
  FILE * src;
  FILE * dest;
  char src_file_name[MAX_FILE_LEN];
  char dest_file_name[MAX_FILE_LEN];
  int c;
```

```c
  printf("Source File: ");
  scanf("%s", src_file_name);

  printf("Dest File: ");
  scanf("%s", dest_file_name);
  src = fopen(src_file_name, "r");
  dest = fopen(dest_file_name, "w");

  while ((c = fgetc(src)) != EOF)
    fputc(convert_character(c), dest);

  fclose(src);
  fclose(dest);

  return 0;
}
```

실습 3 - 두 파일을 합치는 프로그램

- 텍스트 파일의 이름 두개를 입력받아, 첫 번째 파일의 내용 뒤에 두번째 파일을 추가해보세요
  - fgetc(), fputc()를 사용해주세요



실습 3 - 두 파일을 합치는 프로그램

- 기존 파일 내용은 유지하고 끝에 이어쓰려면 어떻게 해야될까요?
  - 파일 접근모드를 염두해주세요
  - w 모드로 파일을 열 경우에는 기존 파일 내용을 지우기 때문에 해당 모드를 사용하면 안됩니다
  - r+ 모드의 경우에는 기존의 내용에 덮어쓰는 모드입니다.
    - r+로 파일을 열 경우에는 fseek(fp, 0, SEEK_END)를 통해 파일 포인터를 EOF로 이동해주세요
  - a 모드의 경우에는 파일의 끝부터 데이터를 쓰는 모드입니다.
    - a로 파일을 열 경우에는 fseek()를 통해 파일 포인터를 옮기지 않아도 됩니다
  - r+모드, a모드 두 모두 실습해 보세요

실습 3 - 풀이

r+ 모드 사용

```
fp1 = fopen(file1, "r+");
fp2 = fopen(file2, "r");

fseek(fp1, 0, SEEK_END);

while ((c = fgetc(fp2)) != EOF)
  fputc(c, fp1);
```

a 모드 사용

```
fp1 = fopen(file1, "a");
fp2 = fopen(file2, "r");

while ((c = fgetc(fp2)) != EOF)
  fputc(c, fp1);
```

Q&A

Q & A

- 기말고사 및 텀 프로젝트 관련 내용 자유롭게 질문해주세요