해설서

Preprocessor(전처리기)

전처리기는 컴파일 전 소스 파일에 있는 전처리 지시문을 처리하는 작업으로, 복잡한 상수나 함수를 간단하게 치환하고 컴파일 환경 또는 버전을 관리하기 위해 사용됩니다.

공통 서클 과제를 진행하면서 상수를 define하거나, 헤더 가드를 구현할 때 사용하게 되지만, 노미넷 규칙과 클러스터 맥이라는 고정된 환경으로 인해 매크로 함수와 버전 관리를 위한 지시문 사용은 경험하기 힘듭니다.

하지만, 42서울에서 사용하는 C와 CPP는 자바 등 최신 언어와 달리 **운영체제 종속적인 언어**로 각 OS에 맞는 환경을 모두 고려하여야 합니다.

또한, 과제와 평가를 진행하면서 여러 환경에서도 내 코드가 동일하게 작동을 하는 지, 나의 환경과 다른 환경에서 입력되는 데이터는 어떻게 처리해야 하는 지 등 생각은 해봤지만 어떤 방식으로 해야할 지 몰랐거나 방식은 알지만 어디부터 접근해야 할 지 막막했던 경험이 있을 거라 생각됩니다.

Preprocessor 대쉬를 통해 전처리기에 딥 다이브하여 42 과제에서 해왔던 방식과 다르게 접근하여 전처리기를 사용할 수 있습니다.

ex00 Get_File_Name

출제 의도

여러 플랫폼을 지원하는 라이브러리 또는 프로그램인 경우, 전처리 과정에서 특정 플랫폼 환경에 맞는 헤더 파일과 라이브러리를 사용해야 합니다.

전처리 지시자를 사용한 조건부 컴파일을 통해 운영체제를 확인하는 방식을 배우고, 각 운영체제에서 주어진 경로 에 있는 파일의 이름을 가져오는 과제입니다.

학습 포인트

- 여러가지 전처리 지시어를 찾아보고 사용할 수 있게 됩니다. (`#if defined`, `#elif defined`, `#ifdef`, `#elifdef`, `#ifndef` 등)
- `Predefined macro`에 대해 찾아보고, 각 OS의 컴파일러가 가지는 상수를 사용할 수 있게 됩니다.
- 각 플랫폼에서 사용 가능한 헤더와 함수가 다름을 인식하고 각 환경에서 사용 가능한 헤더와 함수를 사용하게 됩니다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h> // 헤더 추가(윈도우 & 맥 지원)

// Predefined macro인 _WIN32, __APPLE__ 또는 __MACH__ 를 익혀서 지시문을 통해 분기
#if defined _WIN32
#define DELIMITER '\\' // Windows 환경에서는 백슬래시('\')를 구분자로 설정
#elif defined __APPLE__
// #elif defined __MACH__
#define DELIMITER '/' // MacOS 환경에서는 슬래시('/')를 구분자로 설정
#else
#define DELIMITER -1 // 과제에서 주어진 두 OS가 아닌 경우, NULL을 반환하도록 -1 설정
#endif

char* ft_get_filename(const char* path) {
  char*  filename = NULL;
  char*  last_delimiter = NULL;
  size_t filename_len;

  if (path == NULL) {
    return NULL;
  }
  last_delimiter = strrchr(path, DELIMITER); // 경로의 뒤에서 부터 구분자를 확인
  if (last_delimiter == NULL) { // 구분자가 없는 경우 NULL을 반환
    return NULL;
  }
  filename_len = strlen(last_delimiter);
  if (filename_len == 1) { // 구분자로 경로가 끝나는 경우(끝이 폴더인 경우) NULL을 반환
    return NULL;
  }
  filename = malloc(filename_len + 1);
  if (filename == NULL) {
    return NULL;
  }
  strcpy(filename, last_delimiter + 1); // 파일 이름을 복사하여 반환
  return filename;
}
```

ex01 FT_HOSTCMP

출제 의도

네트워크(빅 엔디안 데이터 입력) 통신 시에 프로그램 실행 환경 메모리 저장 방식에 따라 다른 방식으로 받은 데이터를 처리합니다.

매크로 함수 사용을 강제하여 매크로 함수의 장/단점을 배우고, 두 가지 메모리 저장 방식(엔디안)을 익혀 그에 맞는 처리를 배울 수 있습니다.

학습 포인트

- 매크로 함수의 장/단점을 알아보고 사용 여부를 판단할 수 있게 됩니다.
- 두 가지 메모리 저장 방식(엔디안)을 익히고 그에 맞는 처리 방식을 사용할 수 있게 됩니다.

```
#define FT_COMPARE(A, B) ((A) == (B))

#if defined(FT_USE_32_BIT)
#define FT_HOST_CMP_IMPL(A, B)                                          \
  (FT_COMPARE((A)&0xff, ((B) & (0xff << 24)) >> 24) &&                  \
   FT_COMPARE(((A) & (0xff << 8)) >> 8, ((B) & (0xff << 16)) >> 16) && \
   FT_COMPARE(((A) & (0xff << 16)) >> 16, ((B) & (0xff << 8)) >> 8) && \
   FT_COMPARE(((A) & (0xff << 24)) >> 24, (B) & 0xff))
#else
#define FT_HOST_CMP_IMPL(A, B)                                                \
  (FT_COMPARE((A)&0xffUL, ((B) & (0xffUL << 56)) >> 56) &&                    \
   FT_COMPARE(((A) & (0xffUL << 8)) >> 8, ((B) & (0xffUL << 48)) >> 48) &&    \
   FT_COMPARE(((A) & (0xffUL << 16)) >> 16, ((B) & (0xffUL << 40)) >> 40) && \
   FT_COMPARE(((A) & (0xffUL << 24)) >> 24, ((B) & (0xffUL << 32)) >> 32) && \
   FT_COMPARE(((A) & (0xffUL << 32)) >> 32, ((B) & (0xffUL << 24)) >> 24) && \
   FT_COMPARE(((A) & (0xffUL << 40)) >> 40, ((B) & (0xffUL << 16)) >> 16) && \
   FT_COMPARE(((A) & (0xffUL << 48)) >> 48, ((B) & (0xffUL << 8)) >> 8) &&    \
   FT_COMPARE(((A) & (0xffUL << 56)) >> 56, (B) & 0xffUL))
#endif

#if defined(FT_USE_LITTLE_ENDIAN)
#define FT_HOSTCMP(A, B) (FT_HOST_CMP_IMPL((A), (B)))
#else
#define FT_HOSTCMP(A, B) ((A) == (B))
#endif
```