전처리기와 헤더

01

전처리기란

02

지시자 종류

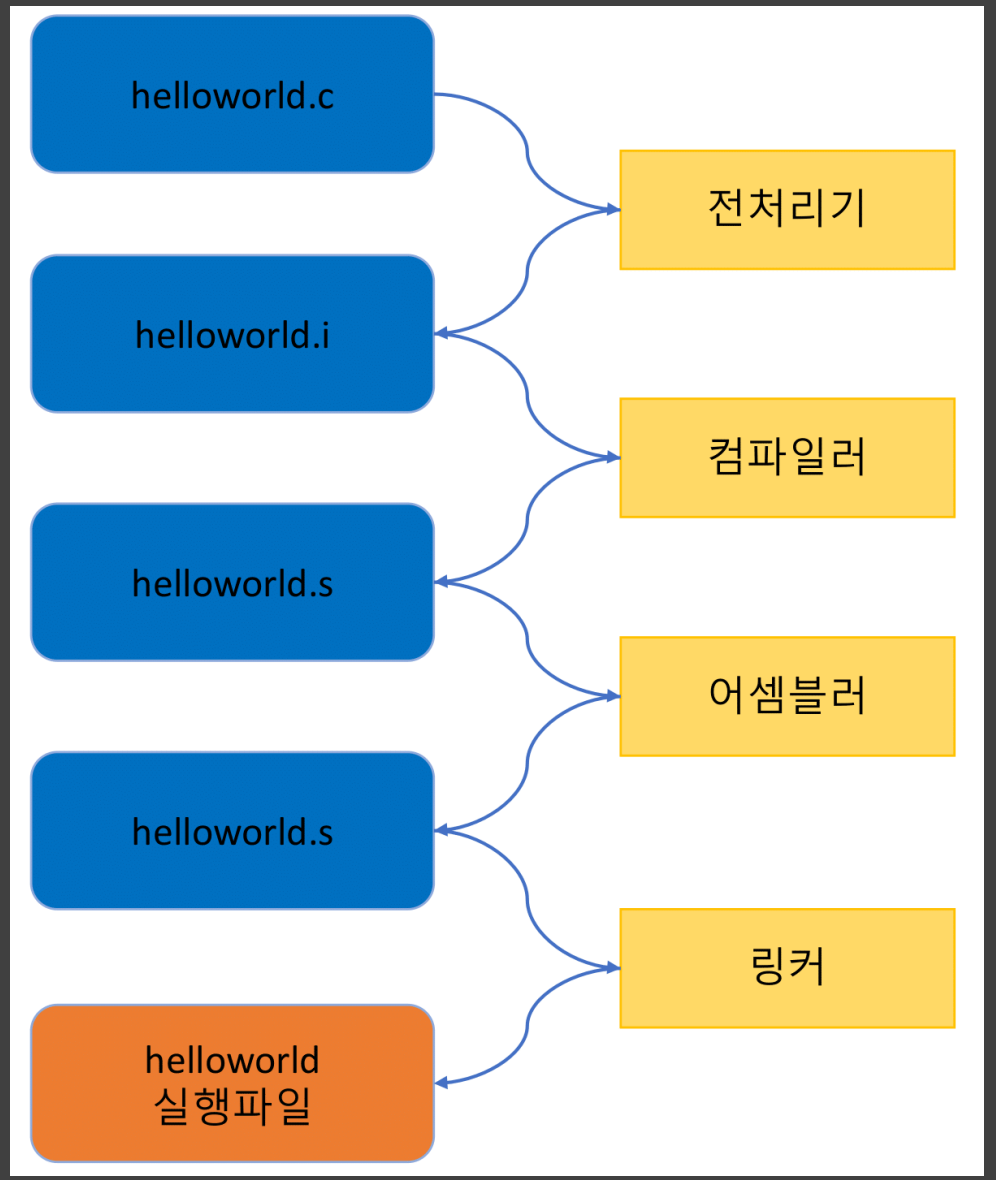
03

헤더파일

04

파일 입출력

C언어 컴파일


helloworld.c
helloworld.i
helloworld.s
helloworld.s
helloworld
실행파일
전처리기
컴파일러
어셈블러
링커

전처리기란(pre-processor)

실행 파일을 생성하는 과정에서

소스 파일 내에 존재하는 전처리 지시문을 처리하는 작업

코드를 생성하는 것이 아니라, 컴파일러가 컴파일하기 좋도록 소스를 재구성해 주는 역할

전처리기란(pre-processor)

특징 및 작성법

1. 전처리문자(#)로 시작
2. 코드 내에서 하나의 라인을 모두 차지
3. 전처리문 뒤에 C언어 코드를 추가하여 같이 사용할 수 없음
4. 맨 뒤에 세미콜론(;)을 붙이지 않음
5. 소스 파일 어디에나 위치 가능
6. 선행처리문이 위치한 곳에서부터 파일의 끝까지 영향

```c
#define PI 3.14
#include <stdio.h>
#define RADIUS 12.0

void square();
void square()
{
    printf("정사각형의 면적은 %.2f * %.2f = %.2f입니다.\n", RADIUS, RADIUS, RADIUS * RADIUS);
}

int main() {
    printf("원주율을 나타내는 PI의 값은 %.2f입니다.\n", PI);

    printf("원의 면적은 %.2f * %.2f * %.2f = %.2f입니다.\n", PI, RADIUS, RADIUS, PI *
    RADIUS * RADIUS);
    square();

    return 0;
}
```

지시자 종류

전처리 지시자	설명
#include	외부에 선언된 함수나 상수 등을 사용하기 위해, 함수나 상수가 포함된 외부 파일을 현재 파일에 포함할 때 사용함.
#define	함수나 상수를 단순화해주는 매크로를 정의할 때 사용함.
#undef	#define 지시자로 이미 정의된 매크로를 삭제할 때 사용함.
#line	__LINE__ 매크로와 __FILE__ 매크로를 재정의할 때 사용함.
#error	지정한 오류 메시지를 출력하고, 컴파일 과정을 중단하고자 할 때 사용함.
#pragma	프로그램의 이식성을 위해 운영체제별로 달라지는 지시사항을 컴파일러에 전달할 때 사용함.
#if, #ifdef, #ifndef, #elif, #else, #endif	조건부 컴파일 지시자

지시자 종류

#define + [매크로]

- 함수나 상수를 단순화해주는 매크로를 정의할 때 사용
- 코드의 가독성을 위해 사용

```c
#include <stdio.h>

void square();
void square()
{
    printf("정사각형의 면적은 %.2f * %.2f = %.2f입니다.\n", 12.0, 12.0, 12.0 * 12.0);
}

int main() {
    printf("원주율을 나타내는 PI의 값은 %.2f입니다.\n", 3.14);

    printf("원의 면적은 %.2f * %.2f * %.2f = %.2f입니다.\n", 3.14, 12.0, 12.0, 3.14 *
    12.0 * 12.0);
    square();

    return 0;
}
```

```c
#define PI 3.14
#define RADIUS 12.0
#include <stdio.h>
#define OUTPUT1 "정사각형의 면적은 %.2f * %.2f = %.2f입니다.\n"
#define OUTPUT2 "원주율을 나타내는 PI의 값은 %.2f입니다.\n"
#define OUTPUT3 "원의 면적은 %.2f * %.2f * %.2f = %.2f입니다.\n"

void square();
void square()
{
    printf(OUTPUT1, RADIUS, RADIUS, RADIUS * RADIUS);
}

int main() {
    printf(OUTPUT2, PI);

    printf(OUTPUT3, PI, RADIUS, RADIUS, PI * RADIUS * RADIUS);
    square();

    return 0;
}
```

지시자 종류

조건부 컴파일 지시자

- 종류
    1. #if
    2. #ifdef / #ifndef
    3. #elif / #else
    4. #endif
- 활용
    1. 디버깅
    2. 다양한 소프트웨어 간의 호환성을 위한 코드 수정 ex) OS, 컴파일러
    3. 매크로 기본 정의

지시자 종류

조건부 컴파일 지시자

작성법

- #if [조건식] -- > 조건식의 결과에 따라 명령문 실행
- #elif [조건식] -- > “else + if”의 의미, 조건식의 결과에 따라 명령문 실행
- #else -- > 조건식의 결과에 따라 명령문 실행
- #ifdef [매크로] -- > “if + defined”의 의미, 정의 여부에 따라 명령문 실행
- #ifndef [매크로] -- > “if + !defined”의 의미, 정의 여부에 따라 명령문 실행
- #endif -- > 조건부 컴파일 종료 지시자, 필수 작성

지시자 종류

#if ~> #elif ~> #else ~> #endif

```c
#include <stdio.h>
#define ERROR 0

int main()
{
    #if ERROR == 1
        printf("오류코드 :  %d", ERROR);
    #elif ERROR == 2
        printf("오류코드 :  %d", ERROR);
    #else
        printf("오류 없음");
    #endif

    return 0;
}
```

#ifdef ~> #endif

```c
#include <stdio.h>
#define PI 3.14
#define RADIUS 0


int main()
{
    #ifdef PI
    {
        printf("PI가 정의되어있으며 PI값은 %.2f입니다.", PI);
    }
    #endif

    return 0;
}
```

지시자 종류

#include

- 헤더 파일을 현재 파일에 포함할 때 사용
- 파일 이름을 보고 해당 파일을 찾아서 그 내용을 현재 파일에 포함
- 표기법
    1. #include <stdio.h> -- > C언어에서 제공하는 표준 헤더 파일을 포함할 때
    2. #include "myStdio.h" -- > 사용자가 직접 작성한 헤더 파일을 포함할 때

헤더 파일

헤더 파일의 정의 및 특징

- 대상을 미리 정의해 놓은 파일
- 한군데로 모아 관리하고 편하게 사용하기 위해 사용
- 대상은 함수, 구조체, 전역변수, 매크로, 다른 헤더파일들 등
- 두 종류의 헤더 파일이 존재
    1. 표준 헤더 파일 --- > c언어 표준라이브러리 내 정의된 대상들의 모음
    2. 커스텀 헤더 파일 --- > 사용자가 정의한 대상들의 모음

표준 헤더 파일
https://www.ibm.com/docs/ko/i/7.3?topic=extensions-standard-c-library-functions-table-by-name

헤더 파일

커스텀 헤더 파일 작성법

- 헤더 파일과 구현 파일로 구분해서 작성
  1. 헤더 파일 --- > 전처리문들 작성, 대상 선언 등으로 구성 (확장자 : .h)
  2. 구현 파일 --- > 전처리문들 작성, 헤더 파일에 선언된 대상 정의 등으로 구성(확장자 : .c)
- 헤더 파일과 구현 파일의 이름은 같아야 함 (hello.h & hello.c)

getPi.h

```h
#include <stdio.h>

void pi();
```

getPi.c

```c
#include "area.h"
#define PI 3.14
#define OUTPUT "원주율을 나타내는 PI의 값은 %.2f입니다.\n"

void pi()
{
    printf(OUTPUT, PI);
}
```

파일 입출력

파일 유형

- 텍스트 파일, 바이너리 파일만 가능

파일 처리 방식

- 파일 열기 -> 읽기/쓰기 -> 파일 닫기(필수)

작성법

1. File 포인터 선언 (File * [변수 명])
2. [변수] = fopen(“해당 파일”, “모드”); --- > 파일 열기
3. 함수 작성
4. fclose(변수); --- > 파일 닫기

모드	설명
"r"	읽기 모드
"w"	쓰기 모드(파일 생성) 파일 존재 시, 기존 내용 삭제 후 작성됨
"a"	추가 모드(파일생성) 파일 존재 시, 데이터가 끝에 추가됨
"r+"	읽기와 쓰기 모드 파일 존재 필수, 기존 내용에 추가됨
"w+"	읽기와 쓰기 모드 파일 존재 시, 기존 내용 삭제 후 작성됨
"a+"	읽기와 추가 모드 파일 존재 시, 데이터가 파일 끝에 추가됨
"b+"	이진 파일 모드
“NULL”	기본값으로 사용

파일 입출력 관련 함수

종류	입력	출력
문자 단위	int fgetc(FILE *fp)	int fputc(int c, FILE *fp)
문자열 단위	char *fgets(char *buf, int n, FILE *fp)	int fputs(const char *buf, FILE *fp)
타입지정 입 출력	int fscanf(FILE *fp, ...)	int fprintf(FILE *fp, ...)
이진 데이터	fread(char *buf, int size, int count, FILE *fp)	fwrite(char *buf, int size, int count, FILE *fp)

출처

IBM : https://www.ibm.com/docs/ko/i/7.3?topic=languages-c-c

TCP School : http://www.tcpschool.com/

FRAIS GOUT : https://fraisgout.tistory.com/

옷 좋아하는 개발자. : https://m.blog.naver.com/PostList.naver?blogId=vjhh0712v