참고자료: chapter 8-1.

전처리

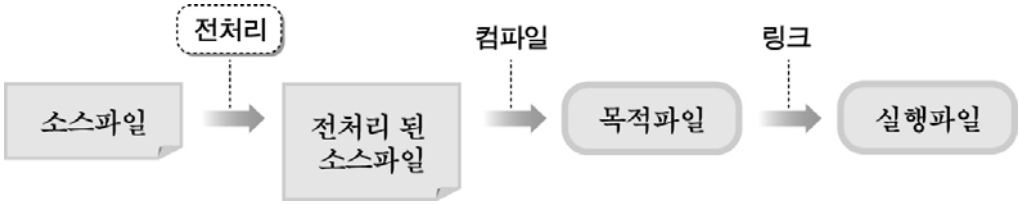
박 종 혁

Tel: 970-6702
Email: jhpark1@snut.ac.kr

출처: 뇌를 자극하는 C프로그래밍, 한빛미디어

❑ 전처리 명령어

- 컴파일 과정에는 전처리(preprocessing) 단계가 있다.
  - 컴파일러는 목적파일을 만들기 전에 전처리라고 하는 특별한 작업을 먼저 수행한다.



  - 전처리 과정에서 소스파일에 다른 파일의 텍스트를 포함시키거나 일부 문장을 다른 문장으로 바꾸는 작업 등을 수행한다.
  - 전처리 명령어는 '#'기호로 시작한다.
  - 전처리가 끝난 파일 역시 소스파일과 마찬가지로 텍스트 파일이다.

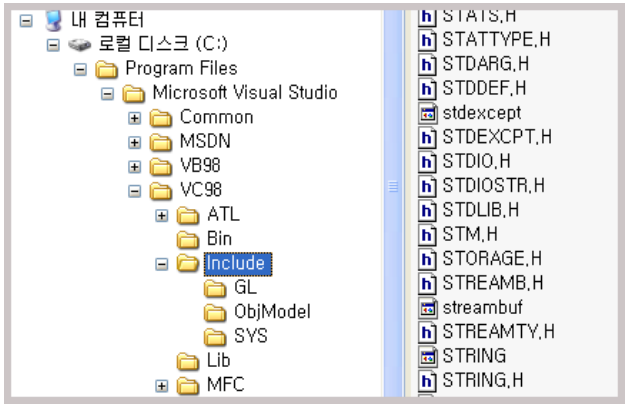
▶ include 명령으로 파일을 포함한다.

- include명령은 원하는 파일을 특정 디렉토리에서 찾아서 명령어가 사용된 곳에 포함시킨다.

```
#include <파일이름>      // 꺽쇠괄호를 사용하여 포함
#include "파일이름"      // 이중따옴표를 사용하여 포함
```

  - 꺽쇠괄호는 포함할 파일을 컴파일러에 설정되어 있는 특정 디렉토리에서 찾는다.

VC++6.0 컴파일러에 설정되어 있는 include 디렉토리



  - 큰따옴표는 포함할 파일을 우선 현재의 작업 디렉토리에서 찾고 해당 파일이 존재하지 않으면 컴파일러에 설정되어 있는 디렉토리에서 찾는다.

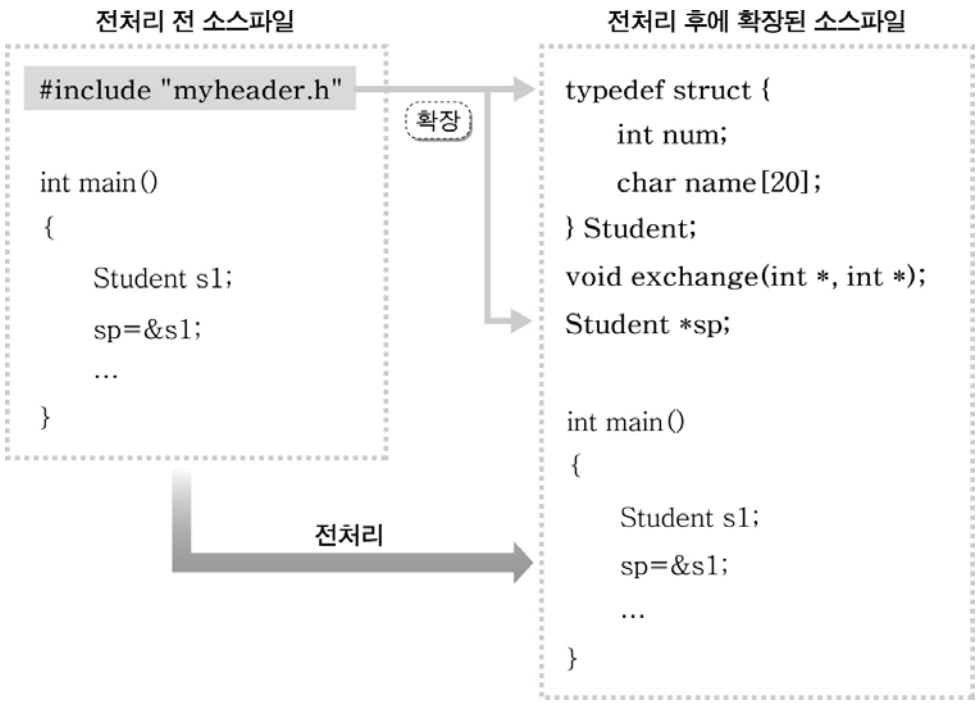
    #include “c:₩user₩myheader.h” // 포함할 파일의 위치를 직접 지정할 때 사용한다.

▶ include 명령으로 파일을 포함한다.

- 헤더파일을 사용하면 프로그램을 깔끔하고 편리하게 작성할 수 있다.
  - 여러 프로그램에서 공통적으로 사용하는 구조체나 함수 또는 외부변의 선언을 헤더파일로 만들어 놓고 각 프로그램에서 간단히 포함하여 사용한다.

```
typedef struct {                    // 구조체의 형 선언
    int num;
    char name[20];
} Student;
void exchange(int *, int *);        // 교환 함수의 선언
Student *sp;                        // 외부변수의 선언
```

⇨ myheader.h로 작성하여 현재의 작업 디렉토리에 저장해 둔다.



▶ define 명령어로 매크로상수를 만들자.

- define 명령어는 매크로명을 정의하여 복잡한 상수나 문장을 의미 있는 단어로 사용할 수 있도록 한다.

```
#define    매크로명    확장문자열
```

  - 컴파일러는 전처리 과정에서 프로그램에서 사용된 매크로명을 확장문자열로 치환한다.

```
#include <stdio.h>
#define PI 3.14159

int main()
{
    double radius;

    printf("원의 반지름을 입력하세요 : ");
    scanf("%lf", &radius);

    printf("원의 둘레 : %lf₩n", 2*PI*radius);
    printf("원의 면적 : %lf₩n", PI*radius*radius);
    return 0;
}
```

전처리 후

```
// 이곳에는 stdio.h 헤더파일의 내용이 포함된다.

int main()
{
    double radius;

    printf("원의 반지름을 입력하세요 : ");
    scanf("%lf", &radius);

    printf("원의 둘레 : %lf₩n", 2*3.14159*radius);
    printf("원의 면적 : %lf₩n", 3.14159*radius*radius);
    return 0;              // 매크로명이 상수값으로 치환
}
```

▶ 문장으로 치환되는 매크로명

- 매크로명은 복잡한 수식이나 문장에도 사용할 수 있다.

```
#define INTRO "Perfect C Language \
& Data Structure"

int main()
{
    printf("Introduction => %s", INTRO);
    ...
}
```

⇦ 여러 줄에 걸쳐 사용할 때는 백슬래시를 사용하여 연결시킨다.

전처리 후

```
int main()
{
    printf("Introduction => %s", "Perfect C Language & Data Structure");
    ...
}
```

– 매크로명은 복잡한 숫자나 문자열 등을 의미 있는 단어로 표현함으로써 프로그램을 읽기 쉽게 해주고 매크로명의 수정이 쉬우므로 유지보수에 도움이 된다.

▶ 여러 가지 매크로명을 사용한 프로그램

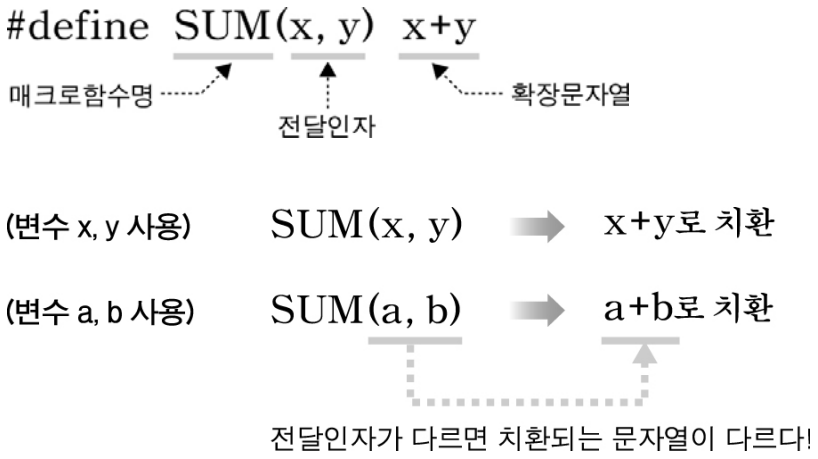
```
#include <stdio.h>
#define PI 3.14159              // 매크로상수 정의
#define LIMIT 100.0             // 매크로상수 정의
#define MSG "passed!"           // 문자열을 매크로명으로 정의
#define ERR_PRN printf("범위를 벗어났습니다!₩n")
                                // 출력문을 매크로명으로 정의
int main()
{
    double radius, area;

    printf("반지름을 입력하세요 : ");
    scanf("%lf", &radius);      // 반지름 입력
    area=PI*radius*radius;      // 면적 계산
    if(area>LIMIT){             // 면적이 한계값을 넘으면
        ERR_PRN;                // 에러 메시지를 출력한다.
    }
    else{
        printf("원의 면적 : %.2lf (%s)₩n", area, MSG);
    }                           // 그렇지 않으면 면적과 문자열 출력

    return 0;
}
```

▶ define 명령어로 매크로함수를 만들자.

- 동일한 수식이나 문장에 대하여 경우에 따라 각각 다른 값으로 확장될 수 있도록 만든 것이 매크로함수이다.

```
#define    매크로함수명(전달인자)    확장문자열
```

  - 전처리 과정에서 전달인자에 따라 서로 다른 문자열로 확장된다.




```
#define SUM(x, y) x+y
...
int x=10, y=20;
int a=100, b=200;
...
printf("x+y = %d₩n", SUM(x, y));
printf("a+b = %d₩n", SUM(a, b));
```

▶ 매크로함수 사용할 때 주의할 점

- 매크로함수는 확장 후에 다른 연산자와의 추가적인 연산에 의해서 부작용이 생길 수 있다.

```
#define MUL(x, y) x*y
printf("%d₩n", 30/MUL(2, 5));    // 전처리 되기 전의 코드

printf("%d₩n", 30/2*5);    // 전처리 된 이후의 코드
```

나눗셈이 먼저 연산 된다!

- 확장 후의 부작용을 막기 위해서 확장문자열에 괄호를 사용한다.

```
#define MUL(x, y) ((x)*(y))    // 확장문자열의 각 문자를 모두 괄호로 감싼다.
printf("%d₩n", 30/MUL(2, 2+3));
```

```
printf("%d₩n", 30/((2)*(2+3)));
```

▶ 매크로함수를 활용한 프로그램

```
#include <stdio.h>
#define PI 3.14159                    // 매크로상수 정의
#define SQUARE(x) ((x)*(x))           // 제곱을 구하는 매크로함수 정의

int main()
{
    double radius;

    printf("반지름을 입력하세요 : ");
    scanf("%lf", &radius);
    printf("원의 면적은 : %.2lf", PI*SQUARE(radius));
    return 0;          // 매크로상수와 매크로함수를 사용하여 원의 면적을 구한다.
}
```

▶ 조건부 컴파일 전처리 명령어

- 조건부 컴파일은 프로그램 코드를 조건에 따라 선택적으로 컴파일 할 수 있도록 전처리 단계에서 걸러준다.
- 명령어에는 if, else, elif, ifdef, ifndef, endif 등이 있다.

[형식 1]

```
#if          조건식1
    컴파일 할 문장 1
#elif        조건식2
    컴파일 할 문장 2
#else
    컴파일 할 문장 3
#endif
```

[형식 2]

```
#ifdef       매크로명
    컴파일 할 문장 1
#elif        조건식
    컴파일 할 문장 2
#else
    컴파일 할 문장 3
#endif
```

[형식 3]

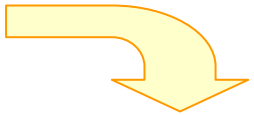
```
#ifndef      매크로명
    컴파일 할 문장 1
#elif        조건식
    컴파일 할 문장 2
#else
    컴파일 할 문장 3
#endif
```

- 조건식에 괄호가 필요 없으며 반드시 #endif로 끝을 표시해 준다.
- 형식1 : if선택문의 if~else if~else구문과 처리방식이 같다.
- 형식2 : 매크로명의 정의에 따라 문장을 선택적으로 컴파일 한다.
- 형식3 : 매크로명이 정의되지 않았을 때 선택적으로 컴파일 한다.

▶ 조건부 컴파일을 사용한 예

```
#include <stdio.h>
#define TC

#ifdef TC
#include <conio.h>
#define MAX_INT 32767
#else
#define MAX_INT 2147483647
#endif

int main()
{
    int m=MAX_INT;
#ifdef TC
    clrscr();
#endif
    printf("Maximum value => %d₩n", m);
#ifdef TC
    getch();
#endif
    return 0;
}
```

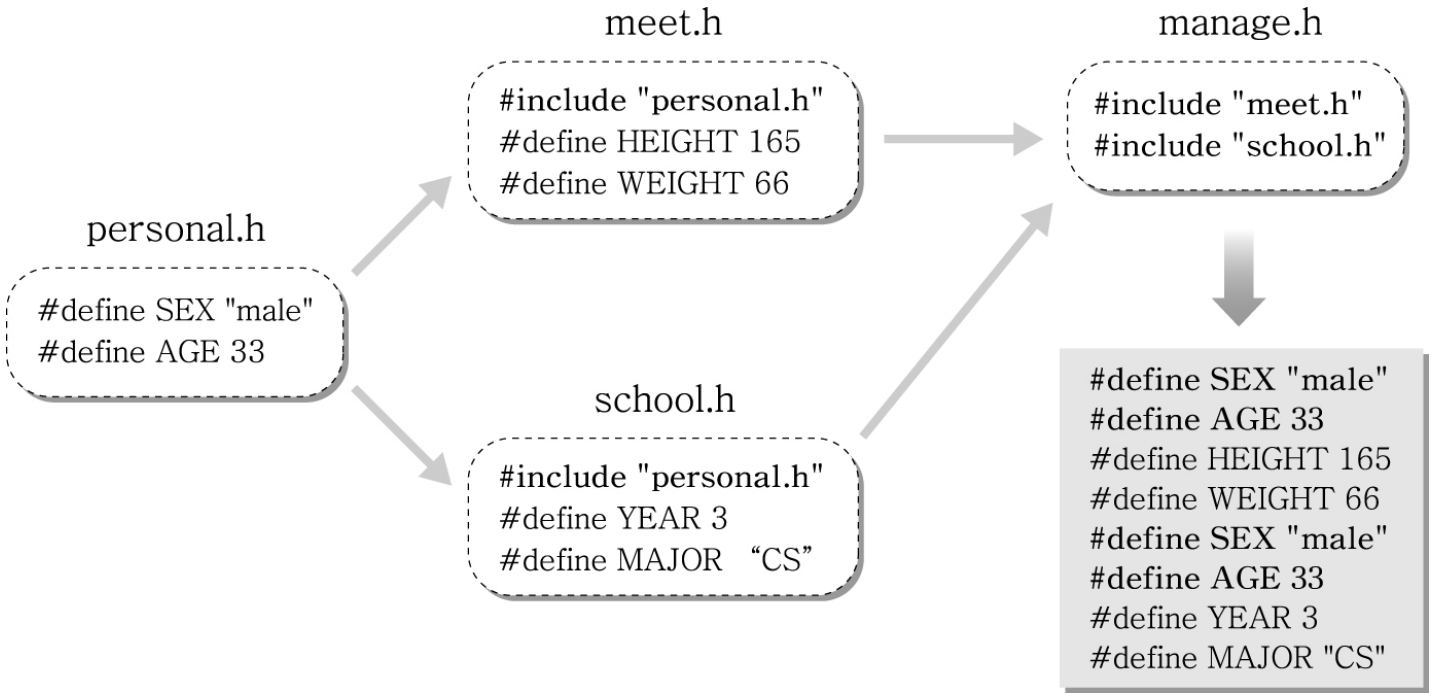
조건부 컴파일이 수행되면



```
#include <stdio.h>
#define TC

#include <conio.h>
#define MAX_INT 32767

int main()
{
    int m=MAX_INT;

    clrscr();
    printf("Maximum value => %d₩n", m);
    getch();

    return 0;
}
```

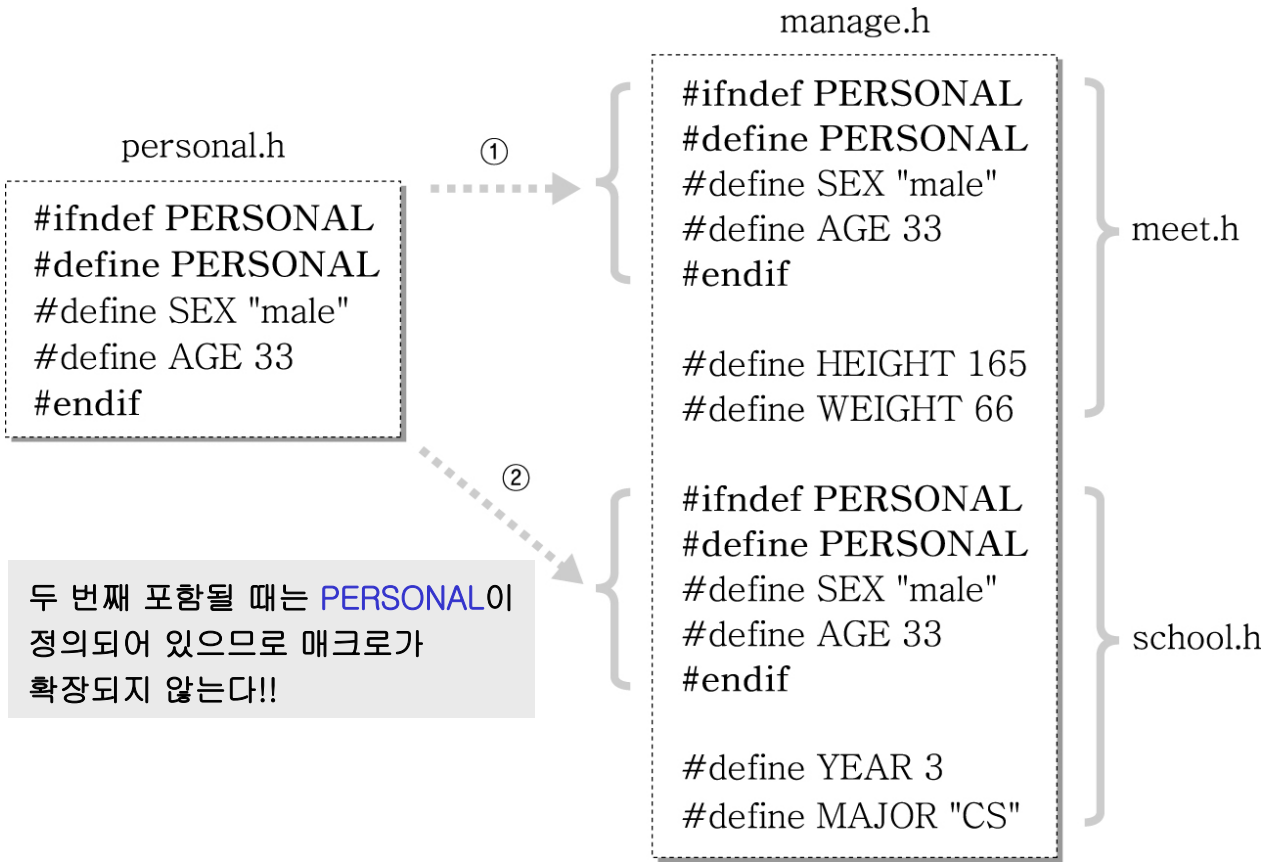
▶ 헤더파일의 중복포함을 해결하자.

- 조건부 컴파일을 사용하면 헤더파일의 포함 관계가 복잡해 질 때 자신의 헤더파일을 다시 포함하는 문제를 해결할 수 있다.
- personal.h 헤더파일이 중복되는 예



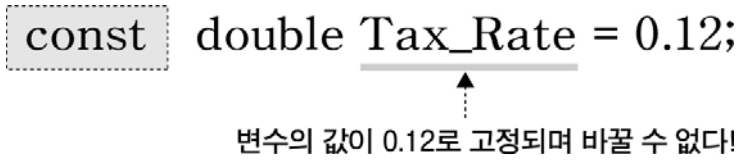
▶ 헤더파일의 중복포함을 해결하자.

- 헤더파일의 선두에 조건에 따라 특정 매크로명을 정의하여 같은 헤더파일이 두 번 이상 포함되지 못하도록 만든다.



❑ const를 사용한 기호상수

- 변수에 const를 사용하면 상수화되므로 변수를 기호상수처럼 사용할 수 있다.



- const를 사용하여 변수를 상수화시키면 그 값을 바꿀 수 없으므로 반드시 선언과 동시에 초기화 해야 한다.

```
#include <stdio.h>

int main()
{
    const double Tax_Rate=0.12;          // 세율을 12%로 고정시킨다.
    int income;                          // 소득액을 저장할 변수
    printf("소득액을 입력하세요 : ");
    scanf("%d", &income);
    printf("세금은 : %.1lf입니다.₩n", income*Tax_Rate);   // Tax_Rate는 상수와 같이 사용된다.
    return 0;
}
```