컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 18강 - 전처리기

학습 목표

전처리기

1) 전처리기를 사용하여 효과적으로 프로그램을 작성하는 방법에 대해서 학습합니다.

전처리기

전처리기

1) 전처리기 구문은 다른 프로그램 영역과 독립적으로 처리됩니다.
2) 전처리기 구문은 소스코드 파일 단위로 효력이 존재합니다.

전처리기

파일 포함 전처리기

1) #include는 전처리기에서 가장 많이 사용되는 문법입니다.
2) 특정한 파일을 라이브러리로서 포함시키기 위해 사용합니다.
3) #include 구문으로 가져 올 수 있는 파일에는 제약이 없습니다.

```
#include "파일 이름"
#include <파일 이름>
```

전처리기

#include 〈파일 이름〉

1) 이와 같이 선언하면 시스템 디렉토리에서 파일을 검색합니다.
2) 운영체제마다 시스템 디렉토리가 존재하는 경로가 다를 수 있습니다.
3) 대표적으로 stdio.h와 같은 헤더 파일 등이 시스템 디렉토리에 존재합니다.

전처리기

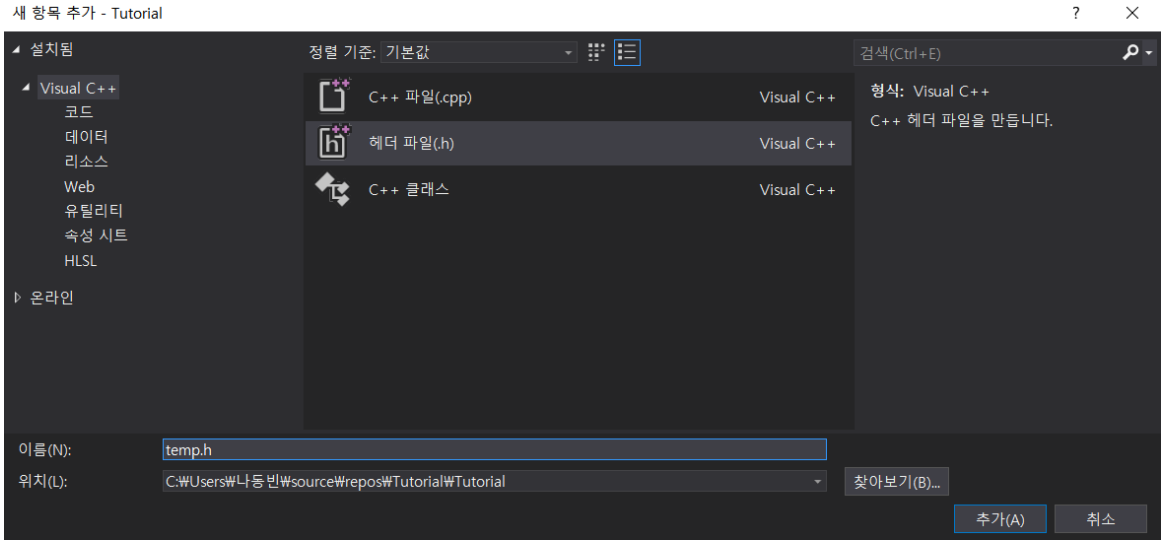
#include "파일 이름"

1) 이와 같이 선언하면 현재 폴더에서 파일을 먼저 검색합니다.
2) 만약 현재 폴더에 파일이 없다면 시스템 디렉토리에서 파일을 검색합니다.

전처리기

나만의 헤더를 만드는 방법

1) Visual Studio에서 C언어 소스코드를 생성할 때와 마찬가지로 헤더 파일을 생성할 수 있습니다.



전처리기

나만의 헤더 파일 작성하기

1) #include를 이용해 가져오는 파일은 말 그대로 파일의 소스코드 자체를 다 가져오는 것입니다.
2) 따라서 의도치 않게 한 번 참조한 헤더 파일을 여러 번 참조하지 않도록 유의해야 합니다.

```
int add(int a, int b) {
  return a + b;
}
```

전처리기

내가 정의한 헤더 파일 사용하기

1) #include를 이용해 가져오는 파일은 말 그대로 파일의 소스코드 자체를 다 가져오는 것입니다.
2) 따라서 의도치 않게 한 번 참조한 헤더 파일을 여러 번 참조하지 않도록 유의해야 합니다.

```c
#include <stdio.h>
#include "temp.h"

int main(void) {
  int a = 10, b = 20;
  printf("%d\n", add(a, b));
  system("pause");
  return 0;
}
```

전처리기

매크로 전처리기

1) 프로그램 내에서 사용되는 상수나 함수를 매크로 형태로 저장하기 위해 사용합니다.
2) #define 문법을 사용해 정의할 수 있습니다.

```c
#include <stdio.h>
#define PI 3.1415926535

int main(void) {
  int r = 10;
  printf("원의 둘레: %.2f\n", 2 * PI * r);
  system("pause");
  return 0;
}
```

전처리기

인자를 가지는 매크로 전처리기

1) #define 문법에는 인자가 포함될 수 있습니다.

```c
#include <stdio.h>
#define POW(x) (x * x)

int main(void) {
  int x = 10;
  printf("x의 제곱: %d\n", POW(x));
  system("pause");
  return 0;
}
```

전처리기

인자를 가지는 매크로 전처리기

1) #define 문법은 소스코드의 양을 획기적으로 줄이는 데 도움을 줍니다.

```c
#include <stdio.h>
#define ll long long
#define ld long double

int main(void) {
  ll a = 987654321000;
  ld b = 100.5054;
  printf("%.1f\n", a * b);
  system("pause");
  return 0;
}
```

전처리기

조건부 컴파일

1) 조건부 컴파일은 컴파일이 이루어지는 영역을 지정하는 기법입니다.
2) 일반적으로 디버깅과 소스코드 이식을 목적으로 하여 작성됩니다.
3) C언어로 시스템 프로그램을 작성할 때에는 운영체제에 따라서 소스코드가 달라질 수 있습니다.
4) 이 때 운영체제에 따라서 컴파일이 수행되는 소스코드를 다르게 할 수 있습니다.

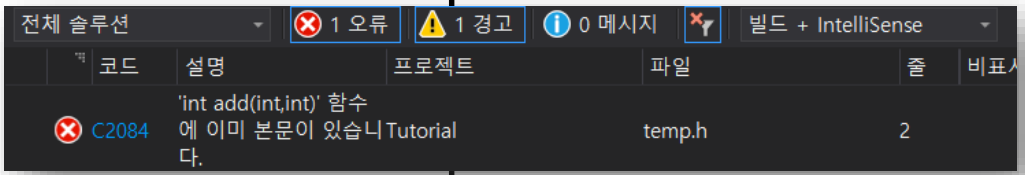
전처리기

조건부 컴파일

1) #ifndef ~ #endif 문법은 대표적인 조건부 컴파일 문법입니다.
2) 흔히 헤더 파일의 내용이 중복되어 사용되지 않도록 하기 위해 적용합니다.

```c
#include <stdio.h>
#include "temp.h"
#include "temp.h"

int main(void) {
  printf("%d\n", add(3, 5));
  system("pause");
  return 0;
}
```



전처리기

조건부 컴파일

1) #ifndef ~ #endif 문법은 대표적인 조건부 컴파일 문법입니다.
2) 흔히 헤더 파일의 내용이 중복되어 사용되지 않도록 하기 위해 적용합니다.
3) 특정한 매크로가 이미 선언이 되어 있지 않을 때에만 특정 구문을 컴파일 합니다.

```
#ifndef _TEMP_H_
#define _TEMP_H_
int add(int a, int b) {
  return a + b;
}
#endif
```

전처리기

파일 분할 컴파일

1) 일반적으로 자신이 직접 라이브러리를 만들 때에는 C언어 파일과, 헤더 파일을 모두 작성해야 합니다.

```c
#include <stdio.h>
#include "temp.h"

int main(void) {
  printf("%d\n", add(3, 5));
  system("pause");
  return 0;
}
```

main.c

```c
#ifndef _TEMP_H_
#define _TEMP_H_
int add(int a, int b);
#endif
```

temp.h

```c
#include "temp.h"

int add(int a, int b) {
  return a + b;
}
```

temp.c

배운 내용 정리하기

전처리기

1) 전처리기의 사용은 필수는 아니지만 때에 따라서 소스코드를 획기적으로 줄일 수 있습니다.