

## 6. 기억류

### 🔄 외부파일의 포함

선처리기에 의해 컴파일러를 만든 회사에서 제공하는 헤더파일이나 사용자가 직접 만든 헤더파일을 프로그램에 포함시키는 작업입니다.

#### 외부파일 포함 형식

- ① `#include <경로명\파일명.확장자>`

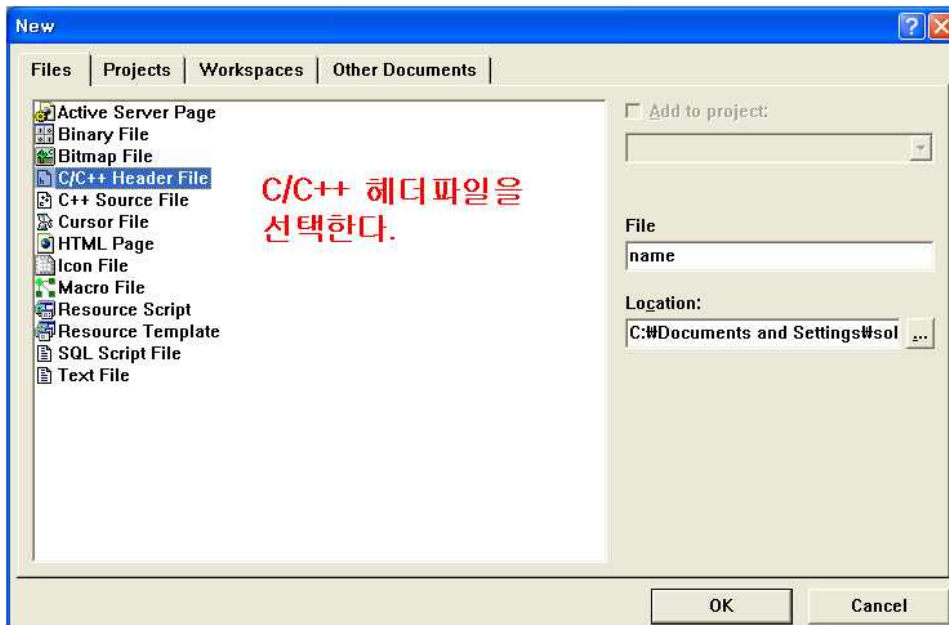
위의 방식은 경로명을 생략할 경우 컴파일러 제작회사에서 설정한 특정한 환경의 경로에서 파일을 찾게 된다.

- ② `#include "경로명\파일명.확장자"`

위의 방식은 경로명을 생략할 경우 사용자 폴더에서 파일을 찾게 된다.

#### 사용자 헤더파일

컴파일러를 사용하는 사용자가 직접 만든 헤더파일을 말한다. 프로그래머가 자주 사용하는 함수 등을 하나의 헤더파일로 만들어 두면 보다 편하고 빠르게 해당 함수를 프로그램에 포함 시킬 수 있게 된다. 사용자 헤더파일을 소스 파일과 동일한 방법으로 만들되 확장자를 .h 로 저장하면 된다. 확장자가 .h인 파일은 헤더파일이며 #include를 통해 프로그램에 포함 시킬 수 있다. 헤더 파일은 별도 실행이 불가능하며 main함수가 포함되어 있을 수 없다. visual studio에서는 새로 만들 파일의 종류를 선택할 때 Files탭에 있는 C/C++ Header Files를 선택한다.

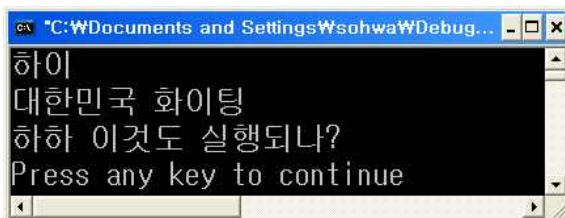


예제 6-1 : 사용자 헤더파일을 만들어 include 해보기

```
#include <stdio.h>
void hi( ) {
    printf("하이\n");
}
void fi( ) {
    printf("대한민국 화이팅\n");
}
```

위의 소스를 입력한 뒤 c:\w에 myheader.h 라는 이름으로 저장한다.

```
#include "c:\wmyheader.h"
void main ( ) {
    hi( );
    fi( );
    printf("하하 이것도 실행되나?\n");
}
```



설명 : 아래쪽의 소스 파일을 만들면서 위에서 만든 myheader.h 파일을 include 했으므로 실제 컴파일 될 때 소스파일 위쪽에 헤더파일의 내용이 포함되어 컴파일 된다. 따라서 실제 컴파일 되는 코드는 아래와 같다.

```
#include <stdio.h>                // #include "c:\wmyheader.h"
void hi( ) {                      // 의 내용이 소스파일 위쪽에
    printf("하이\n");             // 붙어 컴파일 된다.
}
void fi( ) {
    printf("대한민국 화이팅\n");
}

void main ( ) {
    hi( );
    fi( );
    printf("하하 이것도 실행되나?\n");
}
```

따라서 stdio.h도 함께 include 됨으로 main함수에서 printf 함수를 사용할 수 있는 것이다.

## ♻️ 조건부 컴파일

헤더파일에 있는 내용은 `include`를 통해 프로그램에 포함시킬 수 있지만 시스템의 환경에 따라 헤더 파일의 내용이 바뀌어야 할 경우가 있다. 이러한 경우 시스템의 호환성을 높이기 위하여 특정 조건을 만족했을 경우에만 컴파일이 되거나 원하는 내용을 선택해 컴파일 할 수 있도록 하는 것을 조건부 컴파일이라 한다.

### 조건부 컴파일 문의 종류

이름	#if문	#ifdef	#ifndef
용법	<pre>#if 조건식1     명령1.....; #elif 조건식2     명령2.....; #else     명령3.....; #endif</pre>	<pre>#ifdef 매크로명     명령1.....; #else     명령2.....; #endif</pre>	<pre>#ifndef 매크로명     명령1.....; #else     명령2.....; #endif</pre>
설명	조건식 1이 참이라면 명령1을 컴파일한다. 조건식 1이 거짓이라면 조건이 2를 판단해 참이라면 명령2를 컴파일한다. 모든 조건식이 거짓이라면 <code>#else</code> 의 명령3을 컴파일한다.	매크로명에 해당하는 매크로가 <code>define</code> 되어 있다면 명령 1을 컴파일 하고 그렇지 않다면 명령 2를 컴파일 한다.	매크로명에 해당하는 매크로가 <code>define</code> 되어 있지 않다면 명령 1을 컴파일 하고 그렇지 않다면 명령 2를 컴파일 한다.

예제 6-2 : 조건부 컴파일의 사용 예 - 1

```
.....
#define num 5
#if num > 0
    int a = num;           // #if의 조건식이 참 임으로 컴파일 된다.
#elif num < 0
    int a = -num;         // 컴파일되지 않는다.
#else
    int a = 0;            // 컴파일되지 않는다.
#endif

#ifdef num                // num이 define되어 있음으로
    int data;             // 컴파일 된다.
#else
    double data;          // 컴파일 되지 않는다.
#endif
.....
```

```
#include <iostream.h>
#ifndef MSDOS
    #include "MSDOS.h"
    define PI 3.1415
#else
    #include "WINDOWS.h"
    define PI 3.141592
#endif
.....
```

## 🔄 기억장소의 종류

프로그램이 실행되면 운영체제는 자신이 가지고 있는 메모리 중 일부의 제어권을 프로그램에게 넘겨 준다. 프로그램은 이렇게 넘겨받은 메모리를 4가지 영역으로 나누어 사용한다.

1. 코드영역 : 프로그램의 소스코드가 보관되는 영역이다. 함수도 소스코드임으로 이곳에 보관된다.
2. data영역 : 프로그램 종료 시 까지 사라지지 않는 변수들이 이곳에 보관된다. data영역에 보관 되는 변수에는 전역변수, 정적변수, 외부변수 등이 있다.
3. heap영역 : data영역과 stack영역 사이에 있는 여유공간이 heap영역이다. heap 영역은 동적 메모리 할당 시 사용된다. 정적영역인 data영역과는 달리 heap영역에는 사용자가 필요한 만큼 메모리를 할당 받고 필요시점에 할당을 해제 할 수 있다.
4. stack영역 : 함수의 활성화 레코드가 보관된다. 할당받은 메모리의 가장 상부에 있으며 모든 함수의 매개변수, 지역변수 등이 이곳에 보관된다.



## ♻ 변수의 종류

변수는 크게 전역변수, 지역변수, 외부변수로 나누어 진다. 변수에 종류에 따라 메모리에 보관되는 위치가 달라지게 된다. 전역변수와 외부변수는 data영역에 만들어 지고 지역변수는 stack영역에 만들어 진다. 단, 정적변수는 지역변수로 선언되었다 할 지라도 data영역에 만들어 진다.

**전역변수** : 함수 밖에 선언된 변수를 말하며 해당 소스 파일에 있는 모든 함수에서 사용가능한 변수이다. 모든 함수에서 사용할 수 있음으로 편하게 프로그램을 작성할 수 있게 해주지만 전역변수를 사용해 얻는 이득보다 손실이 더 클 수 있음으로 사용을 자제하는 것이 좋다. 전역변수는 초기화 작업을 하지 않아도 자동으로 0 또는 NULL로 초기화 된다.

※전역변수의 단점 : ①한 번 만들면 프로그램 종료시 까지 사라지지 않기 때문에 메모리의 낭비가 심하다.

②모든 함수에서 사용가능하다는 것은 바꾸어 말해 전역변수를 수정하면 모든 함수를 수정해야 할 경우도 있다는 말임으로 프로그램을 수정하기가 어려워진다.

**지역변수** : 함수 내부에 선언된 변수를 말하며 함수 내부에서만 사용이 가능하다. 지역변수는 자신이 만들어진 블록을 벗어나거나 함수가 종료되면 사라진다. 지역변수는 초기화 작업을 하지 않으면 쓰레기 값이 들어있게 된다. (단, 정적변수가 지역변수로 만들어 졌을 경우는 예외이다.)

**외부변수** : 다른 파일에 존재하는 변수와 그 변수에 담긴 값을 이용하기 위하여 사용하는 변수이다. (파일공유변수). 다른 파일에 변수나 함수가 존재하는 것을 알려주기 위한 것으로 실제 메모리에 변수를 만드는 것이 아니라 다른 파일에서 만들어진 변수를 사용만 한다. 변수를 선언 할 때 앞쪽에 extern 키워드를 붙여서 선언 한다.

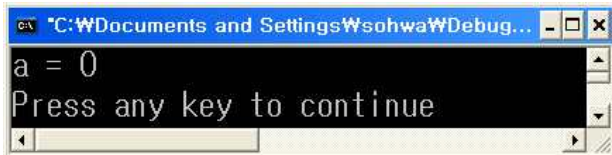
	전역변수	지역변수	외부변수
선언위치	함수 외부	함수 내부	함수 외부
만들어지는 위치	data영역	stack영역 (단, 정적변수는 data영역)	data영역
사용범위	프로그램 전체	함수 내부	다른 프로그램에서 사용가능
소멸시기	프로그램 종료 시	함수나 블록 종료 시	프로그램 종료 시
종 류	전역변수 정적변수(함수외부)	auto변수 정적변수(함수내부) register변수	extern 변수

## ♻ 변수의 종류와 사용 방법

1. 전역변수 : 함수 밖에 아무런 키워드 없이 변수를 만들면 전역 변수가 된다.

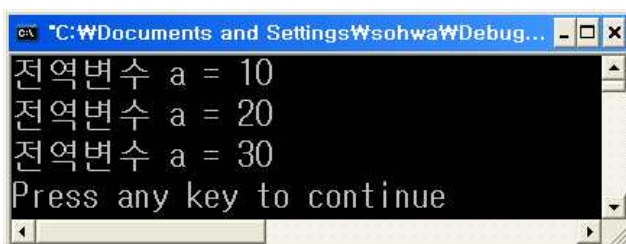
예제 6-4 : 전역변수를 만드는 방법과 전역변수의 사용방법

```
#include <stdio.h>
int a;                                // 전역변수를 만드는 방법
void main ( ) {
    printf("a = %d\n", a);            // 전역변수는 모든 함수에서 사용 할 수 있다.
}                                     // 전역변수는 자동으로 0으로 초기화 된다.
```



예제 6-5 : 전역변수는 모든 함수에서 사용할 수 있다.

```
#include <stdio.h>
int a = 10;                           // 전역변수 선언
void func ( ) {
    printf("전역변수 a = %d\n", a);    // main에서 변경한 a 값 20이 보인다.
    a = 30;
}
void main ( ) {
    printf("전역변수 a = %d\n", a);    // 10 이 출력된다.
    a = 20;
    func( );
    printf("전역변수 a = %d\n", a);    // func에서 변경한 a의 값 30이 보인다.
}
```



예제 6-6 : 전역변수를 사용한 덧셈 프로그램

```
#include <stdio.h>
int sum;                              // 전역변수 선언
void inc ( ) {
    sum = sum + 10;
}
void main ( ) {
    int a;
    for(a=1; a<=5; a++)
        inc( );
    printf("sum = %d\n", sum);
}
```

```
C:\ *C:\Documents and Settings\sohwa\Debug...
sum = 50
Press any key to continue
```

2. 외부변수 : 함수 밖에 변수를 만들면서 extern 키워드를 붙이면 외부변수가 된다.  
extern 자료형 변수명; ⇨ 다른 파일에 존재하는 변수를 사용하겠다는 뜻이다.  
extern 자료형 변수명 = 초기값; ⇨ 변수를 만들면서 초기값을 넣으라는 뜻이다.

예제 6-7 : 외부변수의 사용 예

<<< 헤더파일 c:\w에 ex1.h 로 저장한다. >>>

```
#include <stdio.h>
int a = 10;
void s( ) {
    int a = 100;
    ++a;
    printf("a = %d\n", a);
}
```

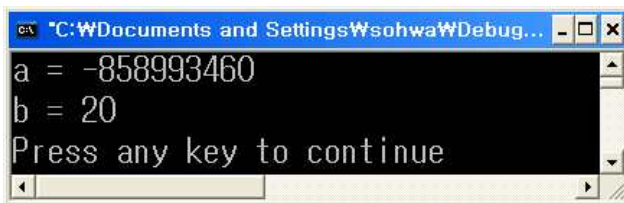
```
#include "c:\w\ex1.h"
extern int a; // 외부변수 선언
void main ( ) {
    int x;
    for(x=0; x<=5; x++) {
        a++; // 외부변수의 사용
        printf("a = %d\n", a);
        s( );
    }
}
```

```
C:\ *C:\Documents and Settings\sohwa\Debug...
a = 11
a = 101
a = 12
a = 101
a = 13
a = 101
a = 14
a = 101
a = 15
a = 101
a = 16
a = 101
Press any key to continue
```

3. auto변수 : 자동변수라 하며 함수의 매개변수 부분이나 함수의 본체 안쪽, 또는 block( { } ) 안에서 만들어 진다. 교제에서 5챕터까지 사용한 변수가 모두 auto변수이다. auto 변수는 만들어진 함수 내에서만 사용가능하며 함수가 종료되면 사라진다. 함수 종료 시 자동으로 소멸됨으로 전역변수에 비해 메모리를 효과적으로 사용 할 수 있다. 또한 auto변수는 초기화를 시키지 않으면 쓰레기 값이 들어 있다. auto라는 키워드는 붙여도 되지만 생략해도 된다.

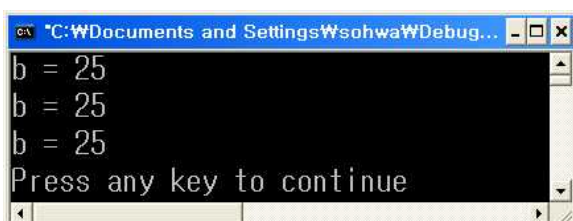
예제 6-8 : auto변수의 사용 예 - 1

```
#include <stdio.h>
void main ( ) {
    auto int a;                // auto변수의 선언 auto는 생략 할 수 있다.
    printf("a = %d\n", a);     // 쓰레기 값이 출력된다.
    {
        int b=20;             // 지역변수이면서 auto변수가 된다.
        printf("b = %d\n", b); // 20
    }
    // printf("b = %d\n", b); 를 하면 오류가 발생한다. b는 auto 변수임으로 선언된 { }
    // 를 벗어나면 사라진다.
}
```



예제 6-9 : auto변수의 사용 예 - 2

```
#include <stdio.h>
void func( ) {
    auto int b = 20;
    b = b + 5;
    printf("b = %d\n", b);    // 25가 출력된다.
}
void main ( ) {
    func( );
    func( );
    func( );
}
```





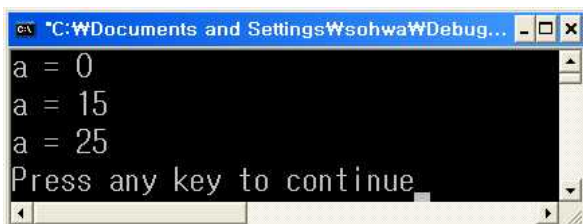
3. 정적변수(static 변수) : 변수를 만들면서 앞쪽에 static키워드를 붙으면 정적변수가 된다. 정적변수는 함수외부와 내부에 모두 만들 수 있으며 만들어진 정적 변수는 전역변수와 같이 자동으로 0으로 초기화되며 메모리의 data영역에 보관되기 때문에 프로그램 종료시까지 사라지지 않는다.

전역변수를 함수외부에 만들면 정적변수는 전역변수와 동일한 특성을 가지게 된다.

전역변수를 함수내부에 만들면 지역변수와 같이 함수 내부에서만 사용 할 수 있지만 함수가 끝나도 사라지지 않고 남아 있게되며 생성 시 자동으로 0으로 초기화 된다.

예제 6-10 : 함수 밖에 정적변수를 만들면 전역변수와 동일하다는 것을 확인하는 예제

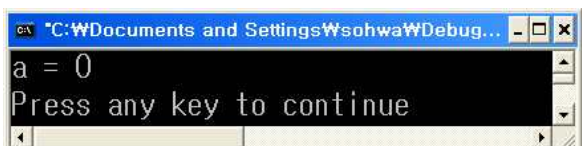
```
#include <stdio.h>
static int a;                // 정적 변수 a의 선언
void func( ) {
    printf("a = %d\n", a);    // 15가 출력된다.
    a = 25;
}
void main ( ) {
    printf("a = %d\n", a);    // 정적변수도 자동초기화 됨으로 0이 나온다.
    a = 15;
    func( );
    printf("a = %d\n", a);    // 25가 출력된다.
}
```



```
C:\Windows\Settings\Wsohwa\Debug...
a = 0
a = 15
a = 25
Press any key to continue
```

예제 6-11 : 함수 안에 정적변수를 만들어 사용한 예제

```
#include <stdio.h>
void func( ) {
    static int a;
    printf("a = %d\n", a);    // 자동 초기화 되어 0이 출력된다.
    // printf("x = %d\n", x); // 함수내에 만든 정적 변수는 다른 함수에서 볼수 없다.
}
void main ( ) {
    static int x;
    func( );
    // printf("a = %d\n", a); // 정적변수가 func함수 안에 만들어 졌음으로 지역변수와
    // 같이 다른 함수에서는 호출 할 수 없다.
}
```



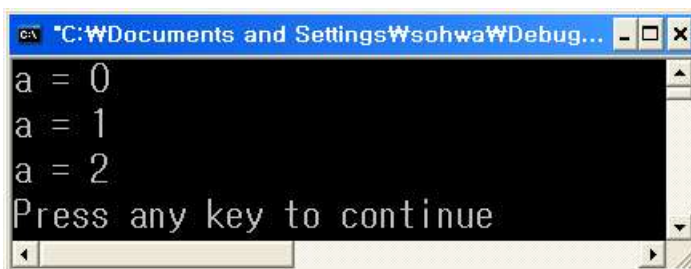
```
C:\Windows\Settings\Wsohwa\Debug...
a = 0
Press any key to continue
```

예제 6-12 : 정적 변수는 함수가 종료되어도 사라지지 않는다는 것을 확인하는 예제

```
#include <stdio.h>

void func( ) {
    static int a;
    printf("a = %d\n", a);    // 자동 초기화 되어 0이 출력된다.
    a = a + 1;
}

void main ( ) {
    func( );
    func( );
    func( );
}
```



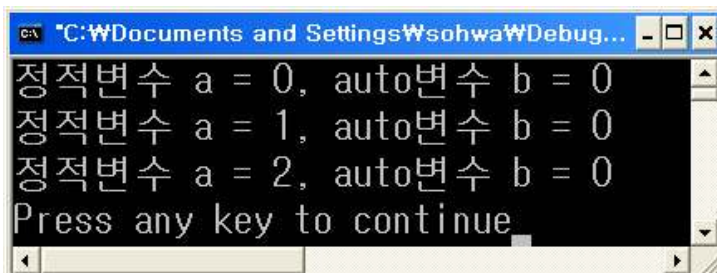
```
C:\Documents and Settings\Wsohwa\Debug...
a = 0
a = 1
a = 2
Press any key to continue
```

예제 6-13 : 함수 내의 정적 변수와 auto변수와의 차이를 보여주는 예제

```
#include <stdio.h>

void func( ) {
    static int a = 0;
    int b = 0;
    printf("정적변수 a = %d, auto변수 b = %d\n", a, b);
    a = a + 1;
    b = b + 1;
}

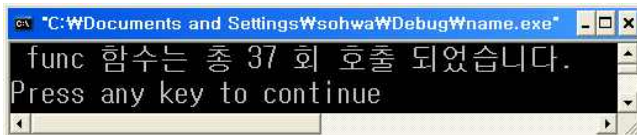
void main ( ) {
    func( );
    func( );
    func( );
}
```



```
C:\Documents and Settings\Wsohwa\Debug...
정적변수 a = 0, auto변수 b = 0
정적변수 a = 1, auto변수 b = 0
정적변수 a = 2, auto변수 b = 0
Press any key to continue
```

예제 6-14 : 정적변수를 이용해 함수의 호출 횟수를 세는 예제

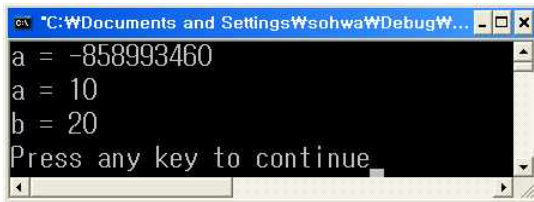
```
#include <stdio.h>
int func( ) {
    static int cnt = 0;
    cnt++;
    return cnt;
}
void main ( ) {
    int data;
    func( );func( );func( );func( );func( );func( );func( );func( );func( );
    func( );func( );func( );func( );func( );func( );func( );func( );func( );
    func( );func( );func( );func( );func( );func( );func( );func( );func( );
    func( );func( );func( );func( );func( );func( );func( );func( );func( );
    data = func( );
    printf(" func 함수는 총 %d 회 호출 되었습니다.\n", data);
}
```



4. register 변수 : register 변수는 변수에 대한 접근속도를 높일 목적으로 CPU안에 있는 register에 만드는 변수를 말한다. register 변수는 변수를 만들 때 앞쪽에 register 키워드를 붙여서 만들어야 하며 지역변수로만 선언 가능하다. 만들어진 register 변수는 auto변수와 모든 특성이 동일하다. (초기값은 쓰레기값, 블록내 선언가능, 함수가 끝나면 사라짐) 단, register 변수는 메모리에 만들어 지는 것이 아님으로 주소값을 구할 수 없다. (c언어에서는 register 변수의 주소값을 구할 경우 에러가 발생한다. C++에서는 register 변수의 주소값을 구하면 register 변수가 auto 변수로 변경된다.)

예제 6-15 : register 변수 사용

```
#include <stdio.h>
void main ( ) {
    register int a;
    printf("a = %d\n", a);    // 레지스터변수는 처음 만들면 쓰레기 값을 가지고 있다.
    a = 10;
    printf("a = %d\n", a);    // 10
    {
        register int b = 20; // 내부 블록이 시작하면 register 변수를 만들 수 있다.
        printf("b = %d\n", b); // 20
    }
    // printf("b = %d\n", b); // 에러, 내부 블록이 종료하면 레지스터 변수는 사라진다.
}
```



예제 6-15 : register변수의 주소값을 출력해 보기

```
#include <stdio.h>
void main ( ) {
    register int a;
    printf("&a = %p\n", &a);    // 레지스터변수의 주소값을 구하면 에러가 발생한다.
                                // C++에서는 레지스터 변수가 auto변수로 변경되어
}                               // 주소값이 출력된다.
```