12 기타 기능

12-1 파일 입출력

12-1-1. 파일 불러오기

12-1-2. 파일로 저장하기

12-2 전처리기

12-2-1. 매크로와 매크로 함수

12-2-2. 조건부 컴파일

12-2-3. 기타 전처리기

12-3 표준 라이브러리

12-3-1. stdio.h

12-3-2. stdlib.h

12-3-3. math.h

12-3-4. string.h

11장까지 같이 열심히 달려온 독자 여러분 모두 수고하셨습니다~! 지금까지는 C언어를 이용해 프로그램을 작성하는 기본적인 방법들과 심화적인 활용법들에 대해 알아보았고, 이번 장에서는 C언어의 부가적인 기능들, 파일 입출력과 전처리기에 대해 간략히 살펴보고, 또 C언어의 내장 함수들을 모아놓은 라이브러리들을 소개하려고 합니다.

12-1 파일 입출력

지금까지 코딩할 때마다 stdio.h를 계속 불러왔고, stdio는 Standard Input/Output의 약자로써 표준 입출력에 관한 모든 걸 포함하는 헤더 파일이다. 우리는 지금까지 검은 콘솔 창에서 입력과 출력을 하는 방법만 공부했는데, 이번 장에서는 메모장에 적힌 데이터를 불러오고 또 메모장에 데이터를 저장하는 방법을 공부하기로 한다.

12-1-1. 파일 불러오기

**A** 파일 포인터의 개념 소개

파일포인터란 파일을 가리키는 포인터변수이다. FILE \*filept; 와 같이 포인터를 선언할 때, 변수형을 FILE로 지정한다. \*뒤에는 다른 포인터 선언과 마찬가지로 사용자의 임의로 포인터 이름을 설정한다.

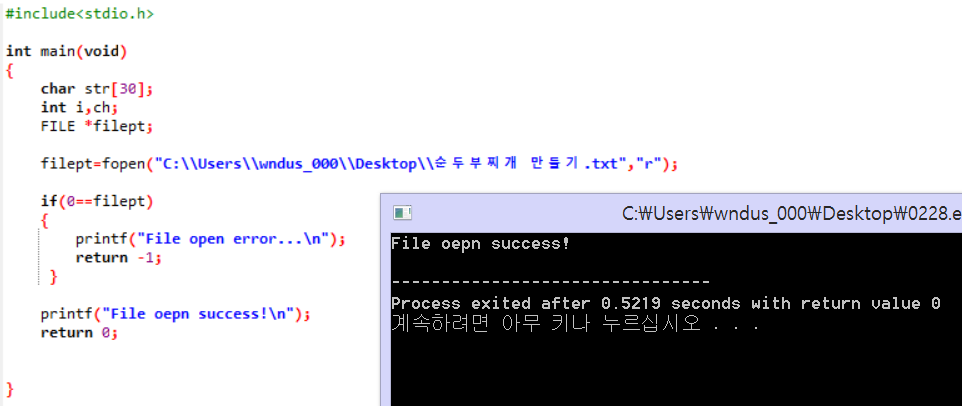
**B 파일을 불러오는 두가지 방법**

파일을 사용하기 위해서는 먼저 파일을 불러와야 한다. 파일을 불러오는 방법에는 프로젝트 외부파일을 불러오는 방법과, 프로젝트 내의 파일을 불러오는 방법이 있다.

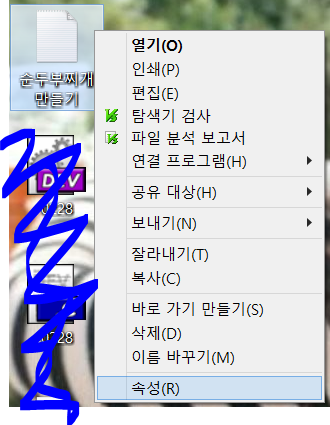
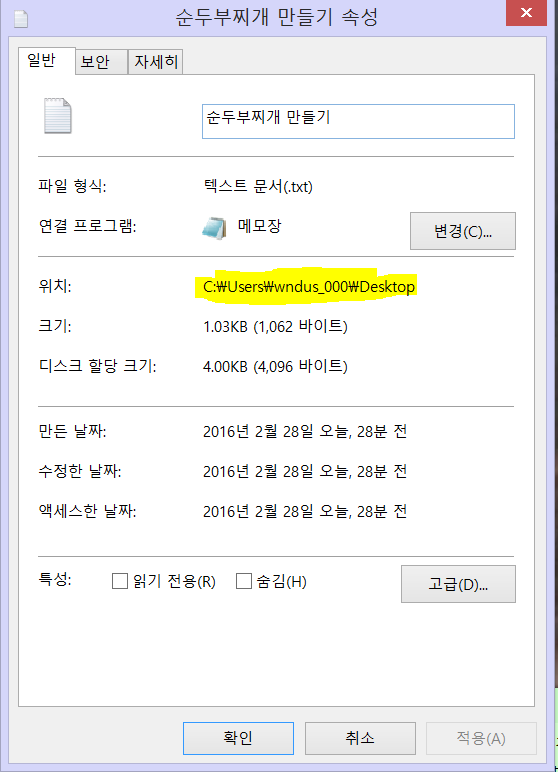
**프로젝트 외부파일 불러오기.**

filept=fopen("파일의 절대 경로\\파일이름.txt","r");

ex) filept=fopen("C:\\Users\\wndus\_000\\Desktop\\순두부찌개 만들기.txt","r");



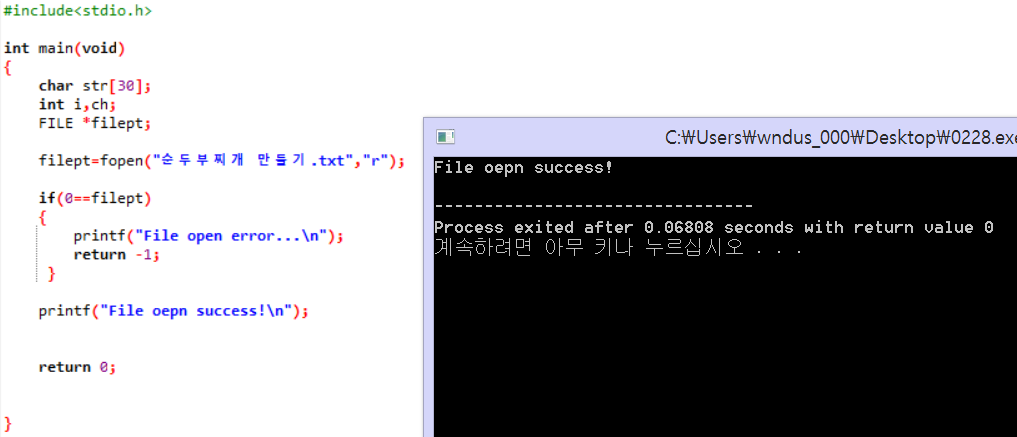
파일의 절대경로는 파일을 마우스 오른쪽 버튼으로 눌렀을 때,

속성 버튼을 누르면 나타나는 정보 중에 위치 항목을 통해 알 수 있다. \를 [\\로](file:///\\로) 바꾼 뒤 이 뒤에 [\\파일이름.확장자”](file:///\\파일이름.확장자)를 뒤에 붙이면 된다.

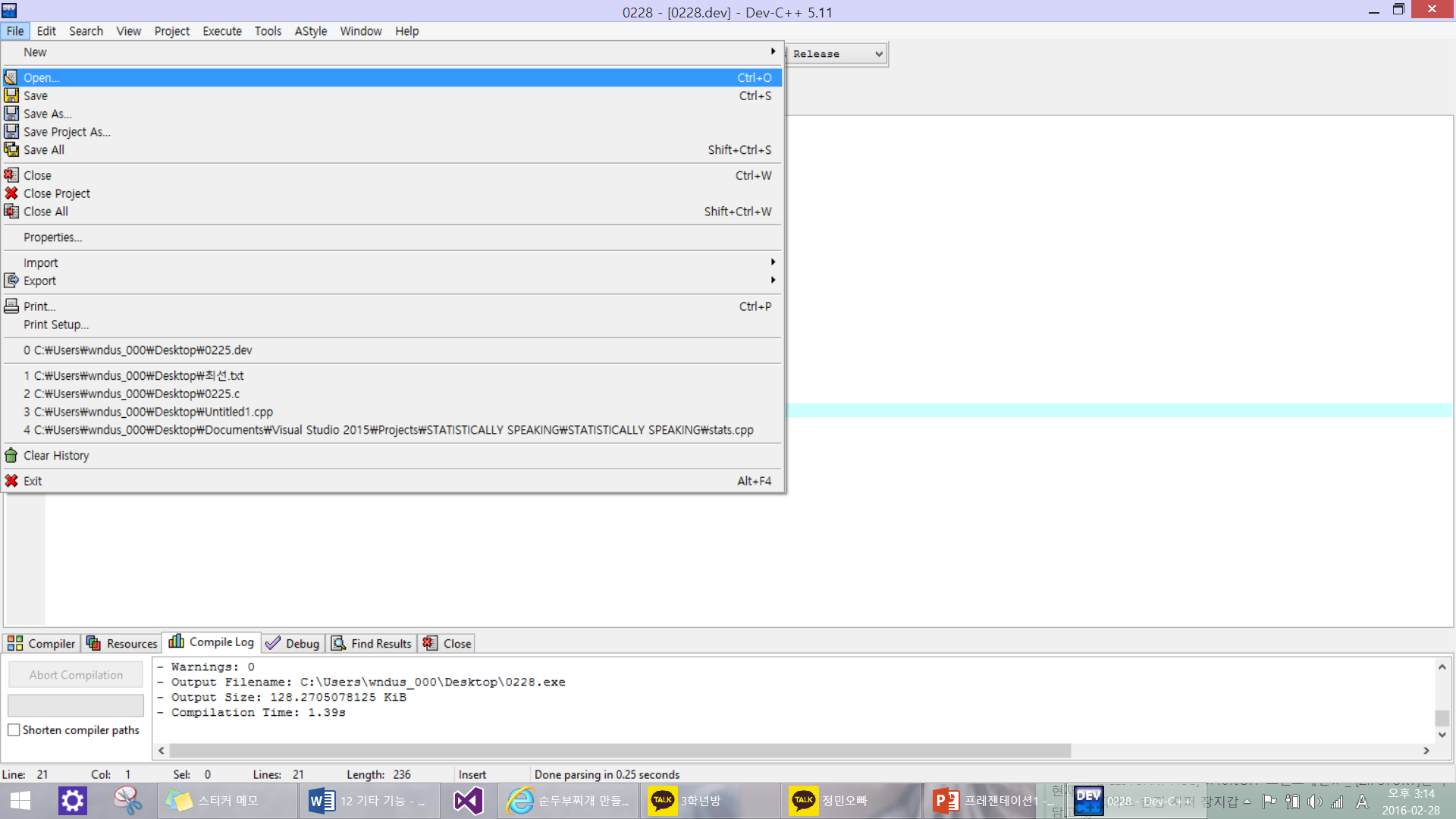
**프로젝트 내 파일 불러오기.**

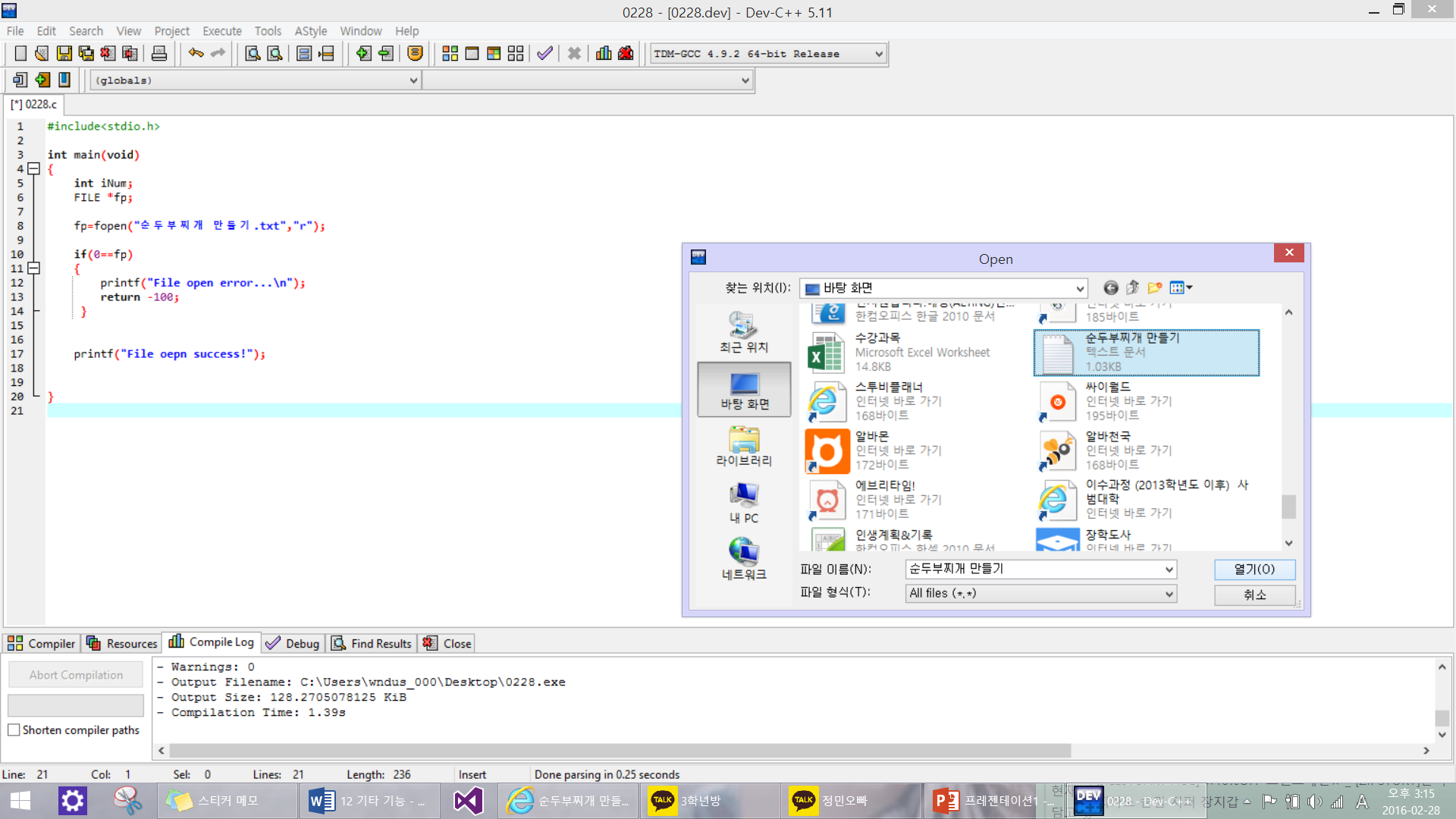
filept = fopen(“input.txt”, “r”);



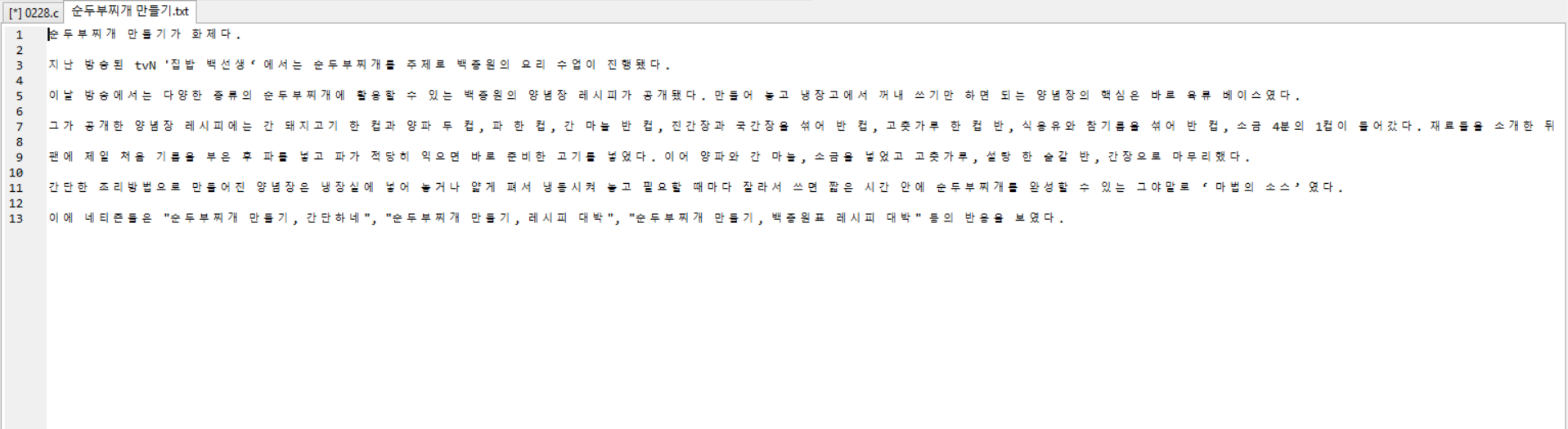
파일을 불러오기 위한 코드를 작성한 뒤

파일명에 절대경로가 아닌 “파일이름.확장자”만을 입력한다.

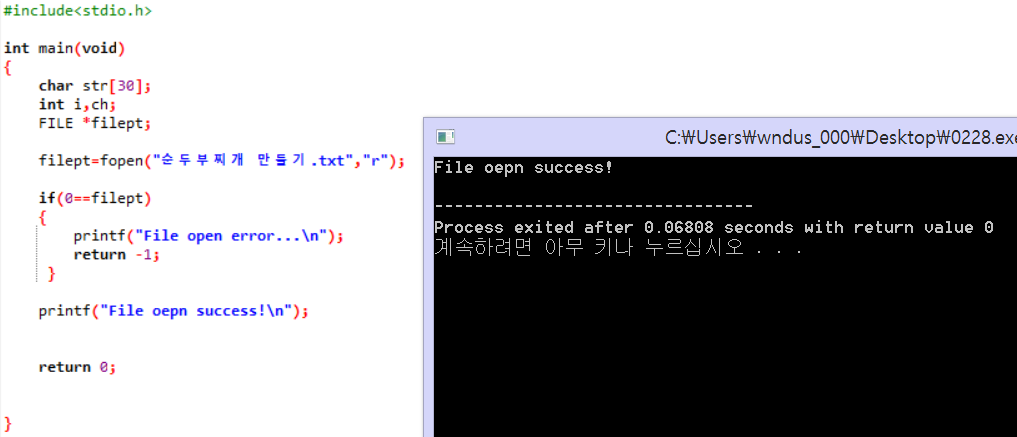




FILE-OPEN에 들어가서 코드작성에 사용한 파일을 선택한다.

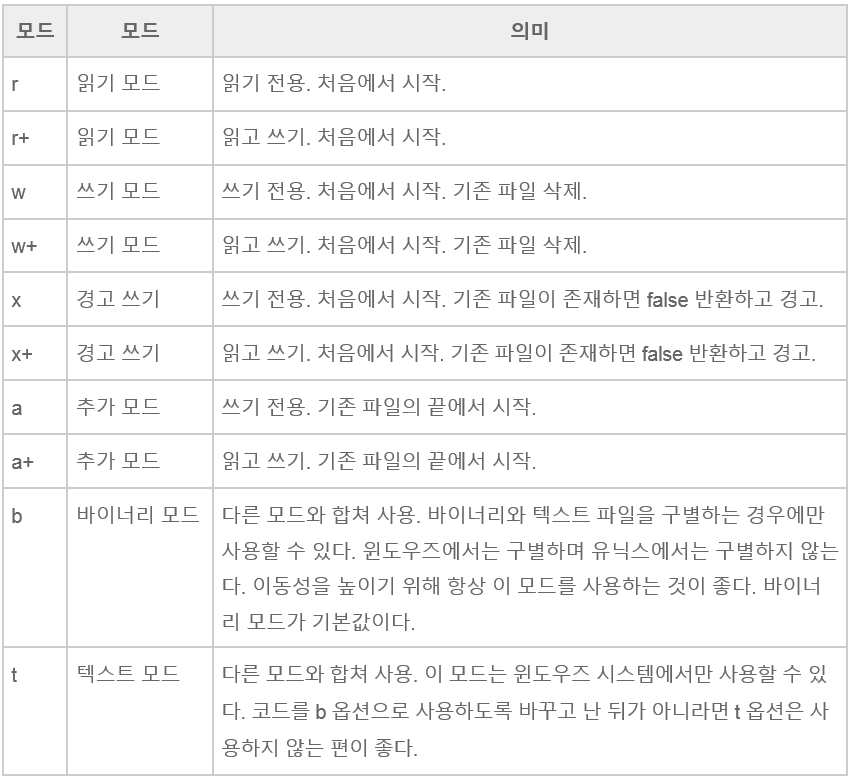


위와 같이 파일이 열린 것을 확인할 수 있다.



이 문법으로 input.txt 파일을 불러올 수 있는 filept 파일 포인터 초기화

파일 불러오기를 할 때 파일 이름 옆에 “r”에 대한 의문을 가졌을 것이다. “r”이 들어갔던 자리는 액세스 방법을 지정하는 역할을 하는 자리로서 “r”은 읽기모드, “w”는 쓰기모드, “r+”는 읽기와 쓰기모드 등의 역할을 지정할 수 있다. 이 외에도 다양한 모드를 지정할 수 있다.



**C** fscanf 구문 소개

fscanf라는 단어를 보면 scanf와 비슷하다는 생각을 했을 것이다. 맞다. fscanf는 파일 입출력에서 사용하는 scanf함수라고 생각하면 간단하다. 사용하는 방법은 다음과 같다.

첫번째 인수인 포인터가 가리키는 파일에서 fscanf함수 호출문의 형식에 맞게 데이터를 불러들인다.

fscanf(filept, ”%s %c %d”,name, &sex, &age);

fscanf함수 외에도 파일의 데이터를 불러들이는 함수가 있다. 그 내용에 대해서는 12-2에서 배우도록 하자.

**D** fclose 구문으로 파일 포인터 해제하기

fclose(filept);

fopen함수를 통해서 파일을 불러왔다면, 프로그램이 끝나기 전에 fclose함수를 이용해 파일을 닫아줘야 한다. 파일을 닫아 줘야 하는 이유는, 동적 할당에서 배운 이유와 같다. 우리는 간단하게 생각하면 된다. 파일을 열었으면 닫아줘야 하는 것이 당연하다.



12-1-2. 파일로 저장하기

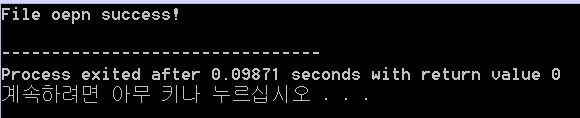
fscanf가 파일의 데이터를 읽어 들이는 함수였다면, fprintf함수는 어떨까? fprintf함수는 데이터를 파일에 저장하는 함수이다. 출력을 콘솔에 하는 것이 아니라 파일에 한다고 생각하면 된다.

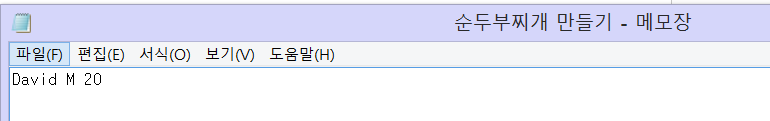
fprintf함수 호출문은 다음과 같다.

fprintf(filept, ”%s %c %d”, name, sex, age);

함수호출문은 fscanf와 유사하다. 첫번째 인수에 파일을 가리키는 포인터를 넣고, 나머지 호출문에 맞는 형식으로 파일에 데이터를 저장한다.







12-2 전처리기

전처리기란 말 그대로, 소스 코드를 처리하기 전에 미리 처리해주는 구문을 말한다. 1장에서 프로그래밍을 하는 것을 설계도를 갖고 하나의 가구를 만드는 것에 비유했었다. 하나의 예시를 들어 한 목수가 책장을 만든다고 생각하자. 목수는 책장, 의자, 책상, 침대 등 다양한 가구들을 만들 수 있는 지식이 있다. 하지만, 책장을 만들 때는 다른 지식이 필요 없고 딱 책장을 만들 지식만 필요하다. 마찬가지로 프로그래밍을 할 때도, 불필요한 것들을 제외한 프로그래밍에 필요한 것들만 필요하다. 이렇게 필요한 것들을 추려주는 것이 전처리기의 역할이다. 우리가 계속해서 봐왔던 대표적인 전처리기는 #include 이다. #include 는 외부에서 프로그래밍에 필요한 도구들을 가져와주는 역할을 한다. #include 외에도 #(전처리기 지시자)으로 시작하는 구문들은 전처리기이다. 좀 더 자세히 알아보도록 하자.

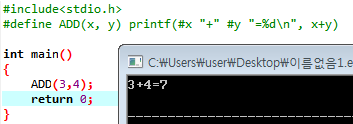
12-2-1. 매크로와 매크로 함수

매크로(Macro)라는 말은 자주 사용하는 여러 개의 명령들을 하나의 키 입력으로 묶은 것을 말한다. 매크로는 #define 문을 사용하는데 단순 매크로와 함수 매크로의 두 종류가 있다. 먼저 단순 매크로는 심볼릭 상수를 정의한 것을 말하는데, 변수와 상수 단원에서 한 번 학습했을 것이다.

(코드)

코드 설명. 이렇게 매크로를 이용하면, 반복적으로 사용되는 값을 바꿀 때 일일이 바꾸지 않고 한꺼번에 바꿀 수 있다는 장점이 있다. 매크로 선언 시 유의할 점은, 매크로 선언은 대부분 대개 대문자로 표기하고, 선언이 끝난 후에 ;(세미 콜론)을 붙이지 않는다는 것이다.

한편, 매크로 함수는 매크로와 동일하지만, 함수의 기능을 하는 매크로를 말한다. 바로 예시를 보자.



ADD라는 매크로 함수를 선언했는데, 이 함수는 x, y값을 받아서 더해주는 함수이다. 매번 printf 함수를 이용해서 더해주는 것보다 이렇게 ADD라는 매크로 함수를 만들어서 사용해주는 것이 훨씬 효율적일 것은 당연하다. 그런데, 매크로 함수를 선언하면서 printf 안의 내용이 우리가 원래 알던 방식과는 다르다. #x 와 #y는 은 매크로 함수를 통해 받은 값을 그대로 써주고, 큰따옴표 안에 들어가 있는 내용은 그대로 출력이 된다. #은 큰따옴표 밖에 위치하도록 한다.

이 외에도 다양한 매크로 함수가 있다. 아래 매크로 함수들을 한번씩 실행해보고 어떤 결과가 나오는지 확인해보자.

#define SUM(x,y) ((x)+(y))

#define AVERAGE(x,y,z) ((x)+(y)+(z)/3)

#define MIN(x,y) ((x)<(y)) ? (x) : (y)

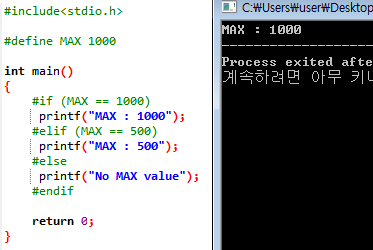
#define MAX(x,y) ((x)>(y)) ? (x) : (y)

한 가지 더 주의할 점은, 매크로 함수 선언 시에는 연산의 오류가 생길 가능성을 예방하기 위해 모든 전달 인자에 소괄호를 쳐줘야 한다.

12-2-2. 조건부 컴파일

12-2-2-1. #if, #else, #elif, #endif

메인 함수에서 if 와 else 문이 있듯이 특정 조건에 합당할 때 실행 시킬 수 있는 조건문을 전처리기에서도 사용할 수 있다. 바로 #if, #else, #elif, #endif 이다. if문과 별 차이가 없기 때문에 간단하게 코드를 보면서 학습해보자.

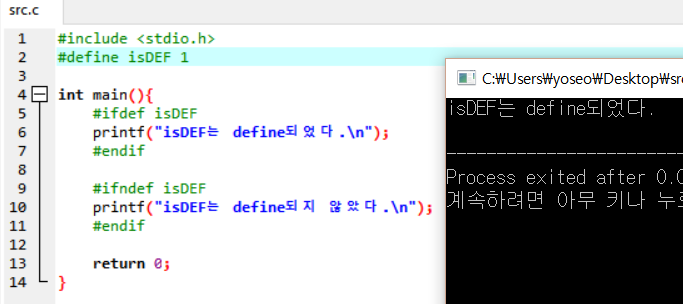


if 문과 거의 비슷하지만 매크로를 이용해서 조건문을 작성했을 때는 else if 대신에 elif를 써주고 if문이 끝났을 때 #endif 로 if 문이 끝났음을 알려주어야 한다.

12-2-2-2. #define 과 연결해서, #ifdef, #ifndef

앞의 매크로 상수와 함수 단원에서 독자 여러분들은 #define 구문을 활용한 매크로 상수와 함수에 대해 알아보았을 것이다. 그럼 이번 단원에서는 이에 연결하여 조건부 컴파일에 대해 조금 더 알아보도록 한다.

바로 위의 단원에서 #if 구문을 통한 조건부 컴파일에 대해 알아보았다. 이 구문의 경우 전처리기 문법이 아닌 if, else if, else 구문과 매우 유사해 굳이 왜 사용하는지에 대한 의문도 들었을 수도 있다. 그러나 일반적인 소스 코드와 전처리기 구문이 소스 코드를 실행 파일로 만드는 과정에서 다르게 처리되는 것을 바탕으로 하여 필요한 경우에 잘 활용할 수가 있는데, 그 예 중의 하나로써 #define 전처리기를 활용한 조건부 컴파일을 하는 방법을 소개하려고 한다. 소스코드 예제부터 살펴보면 다음과 같다.



stdio.h를 불러오고 바로 다음 줄에 isDEF 를 define하였고, main함수에서 #ifdef isDEF는 isDEF가 define되었느냐를 파악하는 전처리기 구문이고 #ifndef는 isDEF가 define되지 않았는지 여부를 파악하는 전처리기 구문이다. 위의 예제에서 isDEF는 잘 define 되었으므로 당연히 출력되는 결과는 #ifdef isDEF 아래의 구문이 실행된 것의 결과가 출력될 것이고, #define~을 주석처리한다면 아래의 #ifndef~ 이하의 구문이 실행될 것이다.

결국 #if를 배웠던 이유는 #ifdef, #ifndef를 사용하기 위함이고, 헤더 파일을 직접 작성하는 경우 혹은 윈도우 어플리케이션에 관련된 프로그래밍을 할 때 #define 여부를 파악하는 경우가 꽤 되는데 이러한 경우들을 위해 조건부 컴파일을 알아야 하는 것이다. 지금은 딱히 사용할 일이 없어 보이더라도 알아두고 손해볼 일은 없으니 잘 숙지하도록 하자.

12-2-3 기타 전처리기

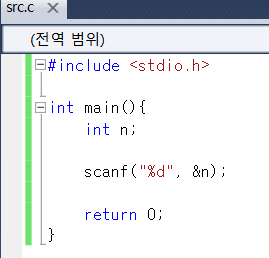
12-2-3-1 #include에서 <>와 “”의 차이

전처리기 구문 중 include 구문은 헤더 파일을 불러오는 것이라는 사실은 지금쯤 대부분의 독자 여러분들은 다 알고있는 사실일 것이다. Hello, world를 출력하였던 첫 코드부터, stdio.h를 불러오기 때문이다. 그리고 그 헤더 파일들을 불러 올 때 #include <stdio.h> 와 같은 형식으로 사용하였던 것을 기억할 것이다.

그렇지만 모든 헤더 파일이 <>모양의 꺽쇠를 이용하여 불러오는 것만은 아니다. IDE(Integrated development environment)에 내장되어 있는 헤더 파일의 경우(stdio.h, stdlib.h 등 뒤의 표준 라이브러리에서 더 다룰 수 많은 헤더들) <>를 이용하지만, 개인 프로그래머가 직접 작성한 헤더 파일의 경우, “” 쌍따옴표를 이용해 include한다. 유념해야 할 사실이다.

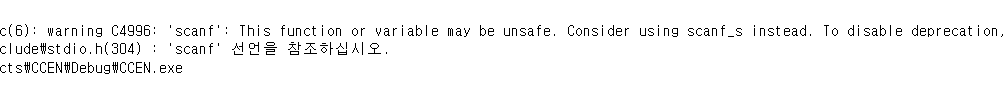
12-2-3-2 #pragma

지금까지 #include, #define과 조건부 컴파일을 위한 전처리기 등 수 많은 전처리기를 학습했다. 마지막으로, #pragma라는 전처리기에 대해 학습하며 이번 중단원을 마무리 지으려고 한다.

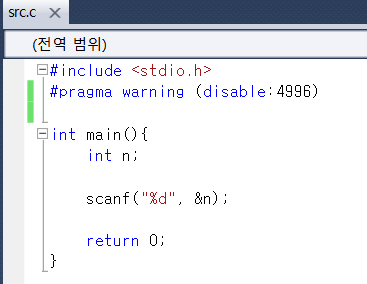


#pragma를 사용하는 경우는 거의 없다. 그래서 많은 사람들이 익숙하지 않은 전처리기일 것이다. 지금까지 온 책을 통틀어 우리는 Dev-C++라는 프로그램을 이용해 C언어를 학습하였다. 하지만 계중에는 조금 더 유명한 IDE인 Visual Studio 를 이용해 실습을 하였던 독자들도 있을 것이고, 또 지금 Dev를 이용해 학습한 독자들도 앞으로의 코딩에 있어 Visual Studio를 이용하는 독자들이 꽤나 많을 것이다.

Visual Studio를 이용해 위의 코드를 실행하면, 실행 버전에 따라 다음과 같은 경고문 혹은 컴파일 에러가 날 수 있다.



4996번 경고, scanf 함수가 안전하지 않다는 경고이며, scanf\_s를 대신 사용해 달라는 메시지를 가지고 있다. scanf 함수는 보안이 매우 취약하다는 것은 최근 널리 알려지고 있는 사실이며, 그래서 입력 버퍼를 직접 조절하는 scanf\_s를 사용해 달라는 경고를 띄우지만 이제 막 걸음마를 떼는 단계의 C언어 학습자 여러분들에게는 정말 귀찮고 또 헷갈리는 것이 될 수 있다. 그래서 이러한 경고(혹은 컴파일 에러 – 2013, 2015 버전에서)를 안 보이게 하는 방법이 하나 있는데,



바로 소스코드 상단에 #include와 #define을 쓰는 전처리기 부위에 #pragma warning (disable:4996)이라고 써 주면 된다. 4996번 경고를 disable 시켜 달라는 전처리기로 해석할 수 있다.

지금까지 12장 기타 기능 대단원의 큰 하나의 중단원이었던 전처리기에 대해 학습하였다. 전처리기의 경우 우리가 작성한 C언어 소스 코드가 어떻게 컴파일 되고, 링크되며 실행 파일로 만들어지는지 전 과정을 다 알아야만 해서 조금 많이 어렵게 느껴졌을 지도 모르겠다. 하지만 이 전처리기들을 잘 활용할 줄 안다면 C프로그래머로서 조금 더 테크니컬한 코딩을 할 수 있게 된 것이다. 마지막으로 대부분의 C언어 컴파일러에서 기본적으로 제공하는 표준 라이브러리에 대해 알아보며 이번 장을 마치려고 한다. 이 내용들은 두고두고 참고하여 쓰는 reference정도로 생각하면 될 것이다.

12-3 표준 라이브러리

12-3-1 stdio.h: Standard Input/Output

printf, scanf는 이미 잘 숙지하고 있으므로 생략한다.

int putchar(int c); 한 문자 출력

int getchar(void); 한 문자 입력

int puts(const char \*str); 한 줄 출력

char \*gets(char \*buffer); 한 줄 입력(Enter가 나올 때까지)

12-3-2 stdlib.h: Standard Library

int atoi(const char \*str); 문자 하나를 숫자로 변환

system(char \*str); 윈도우의 시스템 함수를 사용하고 싶을 때 이용

이외에도 stdlib.h가 다루는 함수는 매우 많다. (stdio.h에 이어 두 번째로 많이 쓰이는 헤더 파일)

12-3-3 math.h: 수학 관련

double pow(double x, double y); x의 y제곱을 구하는 함수이다. 주의해야 할 점은 x와 y가 모두 double형이라는 것이다. int형의 경우 명시 형 변환을 해주어야만 한다.

double sqrt(double x); x의 제곱근을 구하는 함수

int rand(void); 난수를 발생시키는 함수이다. 1~N 범위 내의 값에 대해 난수를 발생시키고 싶으면, rand%N 이렇게 사용하면 된다.

12-3-4 string.h: 문자열 관련

strcpy, strlen, strcmp, strcat는 문자열 장에서 다루었으므로 생략한다.

sscanf, sprintf라는 함수가 있는데 이들은 다른 문자열 변수(char\*)에서 값을 읽어오거나, 다른 문자열 변수에 값을 저장하는 함수이다.

이 외에도 훨씬 더 많은 라이브러리 함수가 존재한다. 이에 대해 관심이 많은 독자들은 <https://en.wikipedia.org/wiki/C_standard_library>를 참고하도록 하자.