2009년 06월 12일

Preprocss (-E option)와 #Include ("" <>)

앞에서 했던 말을 다시 말하면, armcc, tcc가 *.c file을 *.s로 만들기 전에 Preprocessor(전처리기)라고 불리 우는 일을 합니다. 일단 C processor와 lint Processor라는 걸 이용해서 웬만한 syntax적인 것들을 정리해 둡니다. 선언된 macro나 define을 compile하기 전에 모두 바꿔 치기 해 놓는다거나, syntax error가 있는지 등을 점검한 후, armcc나 tcc가 제대로 컴파일 할 수 있는 단계까지 만들어 놓습니다.

이렇게 얘기하면, 조금 뭔가 찔끔찔끔 하는 거 같아서 답답하시겠지만, 작은 예를 들면, #으로 시작 하는 녀석들이 대표선수들입니다. #include, #define, #if #ifdef 등등. 이 녀석들에 대한 용법은, C에 관련된 책자를 열어보시면 무궁무진하게 설명해 놓았으니, 참고하시기 바랍니다 - 라고 말하는 이유는, 어물쩡 거리지 말고 좀더 파고 들어가 깊숙한 이야기를 구체적인 예를 들면서 좀더 자세히 설명하고 싶지만, 하염없이 길어져 감당하지 못할 지경에 이르게 되면, 곤란하게 되니까 그렇습니다. 죄송 -

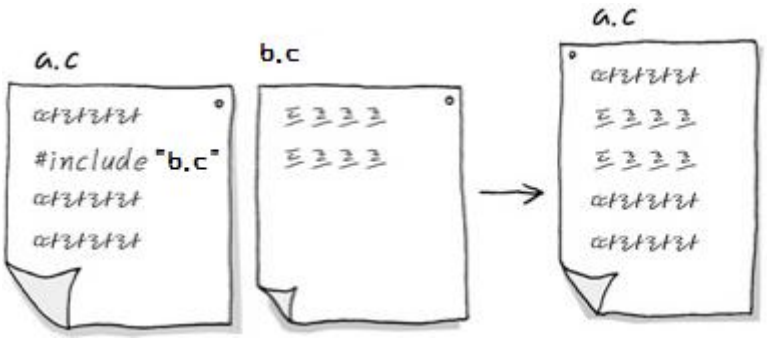
일단은 Preprocessing을 직접 해보는 것이, 엉정쩡한 설명을 길게 하는 것 보다는 나을 거라는 생각이 듭니다. 자, 원하는 컴파일을 해보자 편에서 일단 예를 들어놓은 code를 Preprocessing해 보겠습니다. Preprocess를 하려면 tcc에는 -E option을 이용해서 아래와 같이 하게 됩니다.

```
    tcc -E spaghetti.c > spaghetti.i
spaghetti.i ------------------------------------

#line 1 "spaghetti.c" #line 1 "spaghetti.h"

typedef struct {  bool memberBool;  int memberInt;  word memberWord;  }structure;
#line 2 "spaghetti.c"

int zi = 0; int rw = 3;
extern int relocate = 3;
extern structure recipes [3];
int add (int a, int b);
int main () {  int stack;  volatile int local,local2,loca3;    local = 3;  localI = 4;    localIII = add (local, local2);  stack += local3;
 return stack;   }
int add (int a, int b) {  return (a+b); }
```

자, 어떻습니까? 요리재료를 -E option으로 compile하니 어떤 일이 벌어졌는지 냄새를 맡으셨는지요? 자, spaghetti.h에 있던 #define EQAUL = 이 모두 처리 되어, spaghetti.c에 있던 모든 EQUAL이 =으로 바뀌었습니다. 편리하네요. 그리고 spaghetti.h와 spaghetti.c가 spaghetti.i 라는 파일에 모두 합쳐져 버렸습니다. 이렇게 한 개의 파일이 탄생하게 되는 거죠. 이걸 전처리라고 하며, 이렇게 만들어 놓아야 다음에 C compiler가 Assembly로 바꾸기 쉽겠죠. (일단 이 단계는 tcc나 armcc가 알아서 다 처리해 주니까, 우리가 볼 수 있는 파일은 아닙니다) 이 Preprocessing 단계를 할 줄 안다면, 우리가 흔히 저지르는 #define에 의한 또는 #include에 의해서 저지르는 실수들로 인하여 compile error가 났을 때, 찾기 힘든 문제를 쉽게 찾아갈 수 있습니다. 뭐 저 같은 경우에는 #ifdef과 #end로 묶여서 엄청나게 지저분하게 된 source code를 볼 때, 머리 쓰기 귀찮으니까 이런 식으로 Preprocessing하면 깔끔하게 정리되어 실제로는 결국에 이런 코드가 컴파일 되는 구나... 하는 걸 확인 가능하게 해주니까 그럴 때 사용하면 편리합니다.

전처리 후에는 뭔가 기계어 세상이 아니고, 아직까지는 txt중에 #으로 된 녀석들을 처리해 주는 과정이니까 그저 여전히 source code의 세계입니다. 혼동하지 마세요~!.

잠시, Preprocessing한다고 하니까, 기본적 인 게 하나 생각나네요.
#include하면 header file을 include한다고 생각하시는데, c file도 include할 수 있다는 거 아시나요? #include는 무조건 #include자리에 include한 file이 들어간다는 사실인 게죠.



혹시나 그런 걸 생각해 보신적 있는지 모르겠습니다. #include "" 와 #include <>의 차이를 요. #include 하는 파일을 찾기 위해서 #include<>는 Compiler에게 predefine되어 있는 path에서부터 header를 찾아가고요, #include""는 .c가 있는 해당 directory에서부터 시작하여 Search Path로 등록된 path를 따라서 찾아 갑니다.

예를 들어, ads의 경우에는 ADS가 설치된 directory의 include directory가 predefine path입니다. 그러니까 보통 C에서 제공하는 standard library들은 ADS의 include에 들어 있으니까
#include <stdio.h>,
#include <string.h>
뭐 이런 거 많이 보셨죠? 이런 식으로 include해서 쓴답니다.

대신, 보통 우리가 만든 header를 include할 때는 #include "spaghetti.h" 뭐 이런 식으로 quotation mark로 묶습니다. 그러면, spaghetti.h를 해당 directory에서 찾아서 include하게 됩니다요.

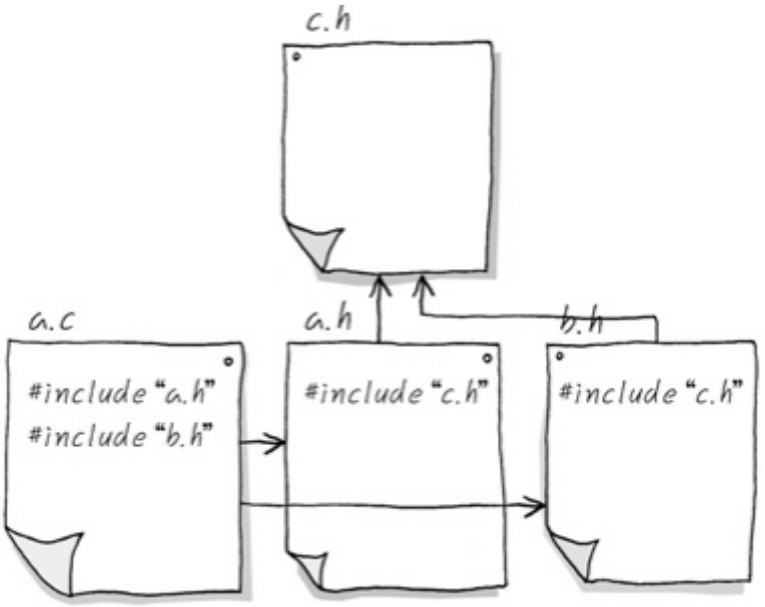
자, 그럼 test를 해볼까요?

#include ""를 테스트 해 봐요~

ⓐ 우리 작업 directory에 spaghetti.h와 spaghetti.c를 모두 한데 넣고서 컴파일 해보니까
잘 되었죠?
→ #include "spaghetti.h"
ⓑ 그럼 spaghetti.c와 spaghetti.h를 다른 directory에 넣게 된다면 어떻게 해야 할까요?
→ #include "spaghetti.h" 하면 뜨아~ 씨리어스 에라네요.
"spaghetti.c", line 2: Serious error: C2857E: #include file "spaghetti.h" wouldn't open
이럴 땐 -I optoin으로 Include path를 더 정해줄 수 있습니다. 만약에 spaghetti.h가 c:₩tempinclude 라는 directory에 따로 있다고 치면 tcc -E -Ic:₩tempinclude spaghetti.c > spaghetti.i 라고 해보면~ 순탄하게 컴파일 되는걸 알 수 있겠습니다.
ⓒ 그러면 spaghetti.h가 ADS의 default directory에 들어 있다면 어떻게 될까요? 아~무 문제 없이 컴파일 잘됩니다.
ⓓ 그러면 거꾸로, stdio.h 가 ADS default include path에 안 들어 있으면 어떤 일이 벌어질까요? 멀쩡히 잘 있는 stdio.h를 ADS default include path에서 다른 곳 (c:₩adsinclude)으로 옮기고 그냥 억지로 spaghetti.c 에 #include <stdio.h>를 해 볼 게요. 어헛! 또 에라네요. 이럴 때 어떤 게 있느냐!! 바로 compiler에 -J option이 있어요 -J option은 Compiler Default Include path를 바꿔줄 수 있어요. 다시 말해 <>의 path를 바꿔 주는 거죠. -J option으로 default directory를 바꾸어서 standard library를 바꿔 치기 할 수도 있다는 말이죠.

그러면, 무조건 include path대로 header를 찾아서 include하면 되지 뭐 이렇게 복잡하게 하느냐는 의문이 생기죠. 그 이유는 컴파일 속도 개선에 그 의미가 있습니다. 어디부터 찾느냐가 컴파일 속도에 큰 영향을 주죠. 파일 한두개 찾는 건 문제가 아닙니다만 보통 큰 Project를 하게 되면 header include가 이만 저만이 아닙니다. ㅎㅎ

include 하니까 한가지 더. 한번 include된 header를 두 번 다시 include 안하고 싶을 때 어떻게 할까요? header가 엉키고 엉키다 보면 우연찮게 같은 header가 두 번 include 될 수 도 있어요 그런 경우에는 두 번 선언되었다고 컴파일러가 에라를 낼 텐데, 그걸 어떻게 막느냐의 문제인데요. 그건 header의 맨 위와 맨 뒤에 아래와 같이 선언하면 두 번 include 안 하게 되죠.



자 어떻게? 좋은 방법이 있지요.

#ifndef __SPAGHETTI_H__
#define __SPAGHETTI_H__

.....

#endif /- __SPAGHETTI_H__ *-

이런 식으로 header를 구성하게 되면 난생처음 파일이 include되었을 때는 __SPAGHETTI_H__ 요게 define이 안되어 있으니까 define하면서 들어가지만, 두 번째 include가 또 된다면 __SPAGHETTI_H__가 define되어 있으니까
아예 본문을 include하지 못하게 됩니다. 요거 요거 참 좋죠?
  이번엔 사족이 본문보다 200만배 더 길었어요. 이게 무슨 사족이람.

compile, include, preprocessing

Linked at at 2009/06/12 23:06

... ; ⓒ 컴파일 공장 이야기 ⓓ 원하는 컴파일을 해보자 ⓔ Preprocess (-E option)과 #include <> "" ⓕ Assembly로 만드는 방법 &n ... more

Commented by 주니 at 2009/07/09 10:55

^^ 읽고 가요 ^^

Commented by 히언 at 2009/07/09 22:58

얌냠~ 감사합니다~