week 03

GCC 의 전체적인 구조 및 구성

정원교

2004년 2월 23일

목 차

제 1 절 시작하며

3 주째 강의를 시작하겠습니다.
1 주와 2 주에서는 GCC 디렉토리의 구성과 GCC 각 파일들의 역활에 대해서 설명하였습니다. 그래서 각 파일들에 대해서 열거한 형태를 하였기 때문에 조금은 지루하지 않았나 생각합니다. :) 물론 저의 엉터리 문장력도 영향을 미쳤다고 생각합니다.
이번 주 부터는 조금씩 내부 구조와 기능에 대해서 설명을 하도록 하겠습니다. 이제부터는 각 함수들의 기능과 전역 변수들의 기능들에 대해서 많이 다루게 될 것이며, 함수 및 변수들의 기능 및 역활에 대해서 일일이 말씀드릴 수 없을 것입니다. 문장의 흐름을 끊을 수 있기에 정리된 '번외편 - GCC 내부 구성요소 참고서' 를 따로 작성하며 함께 글을 써 나가겠습니다.

제 2 절 GCC 의 시작

2.1 main () 에서 시작을

모든 C 프로그램이 그렇듯 GCC 도 main () 함수를 가지고 있습니다. C 언어를 컴파일하는 compiler 가 C 언어로 구성되어 있다고 생각하니 조금 아이러니하게 보일 수도 있을 것 같습니다. 이것은 'bootstraping' 의 개념으로 이해할 수 있는데, 더 자세한 사항은 이번 주 강의와 연관이 없는 내용이므로 나중을 기약하겠습니다.

GCC 의 main () 함수는 $prefix/gcc/main.c 파일에 존재합니다. 두근두근한 마음에 열여 보았을 독자들이 계실테지만, 안에는 단지 toplev_main () 함수를 부르는 것만 존재할 뿐입니다. 그렇습니다. GCC 의 main () 함수는 toplev_main () 함수인 것입니다.

2.2 하지만 toplev_main () 가 진짜다.

toplev_main () 는 $prefix/gcc/toplev.c 에 선언되어 있는데, 이제 본격적으로 GCC 에 대해서 보게 된다는 기분이 드실 것입니다.
GCC 의 메인 함수는 아래와 같은 3 단계로 구성되어 있습니다.

1 단계 command line option 들을 해석하기 전에 front end 환경을 초기화하고 Signal handler 들과 internationalization 등등을 다루는 단계입니다.

2 단계 이 단계에서 다루는 일은 아래와 같습니다.
command line option 들을 해석하고 default falg 값들을 설정합니다. 이 함수는 language-independent option-independent 초기화후에 호출됩니다. 그리고 최소 option processing 을 합니다. Outputting diagnostics 는 초기화 되었지만 GC 와 identifier hashtable은 아직 초기화 되지 않았습니다.

3 단계 compiler 를 초기화하고, 입력 파일을 compile 하는 단계입니다. 실제로 compiler 로써 수행은 대부분 이 부분에서 이루어 집니다.

이제 조금씩 각 단계에 대해 조금만 더 깊이 들어 가도록 하자. 하지만 이번 주의 주제가 “전체적인 구조 및 구성” 이기 때문에 어떻게 프로그래밍되어 있는지만 보는 수준에서 머물도록 합시다.

제 3 절 GCC 프로그램 각 단계

이제 위에서 언급한 단계 별로 조금씩 그것이 어떻게 구성되어 있는가에 대해서 알아 보게 될 텐데, 마지막 3 단계의 경우 프로그램으로써의 GCC 가 아닌 compiler 로써 GCC 를 담고 있기 때문에 다른 단계보다 좀 더 깊이 들어가도 무방할 것 같습니다.

3.1 1 단계

이 단계의 경우 수행하는 일은 아래와 같습니다.

1. Front-end 환경의 초기화
2. 국제화에 따른 locale 메시지 설정
3. Signal handler 의 설정
4. diagnostic 초기화

여기에서는 Compiler 로써의 작동보다는 하나의 프로그램으로써 수행이 되기 위해서 그리고 어떤 기반을 잡기위해서 행하는 것들이 대다수입니다. 비록 위에서 단계에서의 일을 나누어 놓았지만, 각각이 수행하는 code 의 양은 아주 짧습니다. 이 단계에서는 더 깊숙히 들어가는 것은 멈추고 각각의 것에 대한 설명으로 마무리 짓겠습니다.

`Front-end 환경의 초기화`

이름은 거창하나 내용은 없다. 하는 일은 프로그램 이름을 argv 인자로 부터 구분해서 전역 변수에 저장하는 것이 전부다.

`국제화에 따른 locale 메시지 설정`

이것 또한 이름은 거창하다. 하지만 하는 일은 setlocale, bindtextdomain, textdomain 을 차례대로 호출하는 일이다. 물론 함수들에게 GCC 관련이라고 말하는 것은 빼먹지 않는다.

`Signal handler 의 설정`

GCC 프로그램을 수행하다가 다음과 같은 Exception, Fault 가 발생하였는 경우 호출된다.

SIGSEGV, SIGILL, SIGBUS, SIGABRT, SIGIOT

이러한 signal 이 발생하였을 경우 각각 crash_signal () 함수 혹은 float_signal () 함수가 호출되는데, 부동 소수점 연산에 관한 것은 float_handler 가 처리하며, 내부 오류의 경우 crash_signal () 함수가 처리한다. 대부분 경우 GCC 의 실행이 멈추게 된다.

`diagnostic 초기화`

diagnostic 메세지 출력 부분을 초기화하는 단계입니다. 내부적으로 diagnostic_context 라는 구조체를 가지고 있는데 이를 구성하는 요소들을 설정하는 부분이 전부입니다. 하지만 이 구조체가 상당히 방대한 크기를 자랑하기 때문에 이에 대한 더 자세한 설명이 필요할 듯합니다. 다른 회에서 이에 대해 설명하도록 하겠습니다.

3.2 2 단계

이제 이 단계로 접어들면서 프로그램으로써의 역활을 또 다시 수행해야 합니다. 이제 GCC 를 사용하여 compile 할 때 사용자가 입력한 option 들을 하나 하나 살펴봐서 이 옵션이 다른 옵션과 사용할 수 있는지, 이 옵션을 지원하고 있는지에 대해서 살펴 보며, 특정 옵션을 선택했을 때 내부적으로 어떤 다른 옵션들에게 영향을 미칠지에 대해 결정을 하게 됩니다. 크게 아래와 같은 기능들이 수행하게 됩니다.

1. 레지스터 사용에 대한 초기화
2. 언어-독립적 parameter 들의 등록
3. Language-specific option 들의 초기화
4. 옵션 scan 및 타당성 검증
5. 각 Front end 들의 option 무결성 검사

마찬가지로 각각에 대해 부연 설명을 하겠습니다.

`레지스터 사용에 대한 초기화`

여기에서는 각 해당 CPU 에 사용되는 레지스터 class 들을 초기화하는 단계를 거치게 되는데, 사실 이 부분은 “GCC option 해석”이 주제인 이 단계에서는 어울리지 않습니다. 이 부분이 여기에 들어오게 된 것은 아마도 이 단계를 수행함으로써 일어나는 각 switch 들의 변경때문이라고 여겨집니다.

`언어-독립적 parameter 들의 등록`

이 parameter 들은 모두 $prefix/gcc/params.def 파일에 선언되어 있으며 이를 등록하는 과정입니다. 등록된 것은 전역 변수 compiler_params 에 저장되게 됩니다.

`Language-specific option 들의 초기화`

GCC 에서는 Language Hook(lang_hooks) 이라는 것을 제공하는데, “GNU C Compiler” 라고 불리우던 처음 버전에서 “GNU Compiler Collection”으로 바뀌면서 C 외의 다른 많은 언어들을 지원하게 되었는데 하나의 프로그램에 여러개의 compiler 를 지원하게 되다 보니 또한 여러개의 Front-end 들을 가지게 되었습니다. 그리고 각 Front-end 마다 원하는 option 의 형태 또한 다르게 되어 이러한 고민을 해결하기 위해서 lang_hooks 을 사용하여 각각의 Front-end 가 원하는 Option 초기화를 따로 할 수 있도록 하였는데, 이 부분은 각 언어마다 다른 Option 설정을 지원하도록 하는 부분입니다.

`옵션 scan 및 타당성 검증`

여기에서는 실제로 main () 함수로 부터 넘어온 argc 와 argv 를 사용하여 옵션들을 훑게 됩니다. 이 과정에서도 마찬가지로 lang_hooks 의 설정에 따라 각기 다르게 해석하게 됩니다.

각 Front end 들의 option 무결성 검사

이 부분까지 프로그램이 도착하게 되면 모든 command line option 들의 해석이 끝난 시점입니다. 이제 각 Front-end 들에게 무결성 검사 등등과 같은 일을 하도록 배려합니다.

3.3 3 단계

이제 여러가지 기본 준비가 모두 끝나고 실제로 컴파일러로써 역활을 하는 부분입니다. 수행하는 것은 아래와 같습니다. 하지만 여기에서 모든 것을 설명하기엔 GCC 의 내용이 너무 많아 행복하군요. :)

1. 해석한 option 들을 처리
2. Timer 를 초기화
3. 언어-독립적인 부분을 초기화
4. 언어-의존적인 부분을 초기화
5. Input 파일을 compile.
6. 마무리하기 :: 열었던 파일 등등을 close

세부 설명은 아래에서 하겠습니다.

해석한 option 들을 처리

사용자 입력으로 들어온 GCC option 을 앞에서 설정을 했는데, 여기에서는 그 option 들의 조합이 정확한지를 검사하는 부분입니다.

Timer 를 초기화

당연한 이야기겠지만 각종 수행 속도를 재기 위해서 timer 를 사용합니다. 하지만 현재의 대부분 시분활 운영체제의 한계로 인해 0.01 초 이하의 속도를 재지 못합니다.

언어-독립적인 부분을 초기화

언어-독립적인 부분에 대한 초기화가 이루어 집니다. 또한 Garbage-collector, String pool, identifier hash, Register mode 초기화 등등을 설정합니다. 내부적으로 수행되는 자세한 사항은 아래에서 따로 설명하겠습니다.

언어-의존적인 부분을 초기화

언어-의존적인 부분에 대한 초기화가 이루어 집니다. Front-end 초기화, Assembler Output 초기화 등등이 이루어지게 됩니다. 내부적으로 수행되는 자세한 사항은 아래에서 따로 설명하겠습니다.

Input 파일을 compile.

전체 번역해야 할 unit 을 컴파일합니다. 어셈블리 output 과 여러 debugging dump 들의 파일을 작성합니다. 그리고 아직 초기화 되지 않은 다른 컴파일러 단계를 초기화하고 yyparse () 함수를 호출하여 실제 문법 해석에 들어가게 됩니다. 그리고 약간의 마무리 단계를 수행하게 됩니다. 내부적으로 수행되는 자세한 사항은 아래에서 따로 설명하겠습니다.

마무리하기 :: 열었던 파일 등등을 close

Dump 파일들을 닫고, 메모리를 free 하는 과정을 거친후, 각 언어가 원하는 마무리 과정을 수행할 수 있도록 lang_hooks 의 finish 과정을 수행합니다.

제 4 절 3 단계의 부연 설명

GCC 가 compiler 로써 역활을 하는 부분은 거의 3 단계에 집중되어 있습니다. 그래서 다른 단계보다 조금 더 깊게 설명해야 할 듯합니다.

4.1 언어-독립적인 부분을 초기화

compiler 에서 이 부분은 컴파일을 하기 위해서 필요한 요소들을 초기화하는 곳입니다. 그래서 이 부분에서는 compiler 의 여러 단계들이 초기화 되는데, 여기서 초기화 되는 부분은 아래와 같은 것이 존재하며 이에 대한 설명도 붙이겠습니다.

`Garbage-collector 초기화`

여기에서는 ggc-mmap allocator 를 초기화하는데, Paging 기법을 사용합니다. 이 부분에 대해서는 나중에 자세히 기술하도록 하겠습니다.

`String Pool 초기화`

Identifer 가 저장될 장소를 할당받습니다.

`Obstacks 초기화`

GCC 내에서 사용되는 library 중 obstack 관련 부분을 초기화합니다. 이 부분에 대해서도 나중에 기술하도록 하겠습니다.

`Unique rtl object 생성`

모든 함수사이에 공유되는 몇몇 unique rtl object 를 생성합니다. 만약 line number 가 생성되었다면 LINE_NUMBERS 는 0 이 아닐 것입니다.

`Register mode 초기화`

register mode 들의 테이블을 계산합니다. 그러한 값들은 개별적인 register 들에 대한 death 정보를 기록하는데 사용됩니다. (multi-register mode 와는 반대)

`Alias 관련 부분 초기화`

Alist set 관련 부분을 초기화 합니다.

`Statement 관련 초기화`

전역 변수 stmt_obstack 를 위한 obstack 을 초기화합니다.

`Loop 관련 초기화`

전역 변수 reg_address_cost 와 copy_cost 를 설정합니다.

`Reload 단계 초기화`

컴파일당 한번 reload pass 를 초기화합니다.

`Fuction 관련 부분 초기화`

초기화때 function.c 를 초기화하기 위해 한번 호출됩니다.

`Stor-Layout 관련 부분 초기화`

이 함수는 stor-layout.c 파일을 초기화하기 위해 한번 실행됩니다.

`Varasm 관련 부분 초기화`

관련 여러 전역 변수 const_str_htab, in_named_htab, const_alias_set 등등을 초기화합니다.

`EXPR_INSN cache 관련 초기화`

전역 변수 unused_expr_list 를 GGC 에 등록합니다.

`Dummy function 시작`

RTL expansion mechanism 를 초기화합니다. 이렇게 함으로써 sequence 생성과 같은 간단한 것을 할 수 있습니다. 이것은 몇몇 단계들에서 global initialization 동안 context 를 제공하는데 사용됩니다.

`Expmed 관련 초기화`

내용 없음.

`Expr 관련 부분 초기화`

이 함수는 메모리내에서 직접적으로 사용될 수 있는 어떤 mode 들을 설정하고 block mov optab 을 초기화하기 위해 컴파일당 한번만 수행됩니다

`Caller-save 초기화 (선택사항)`

호출시 사용되는 모든 Hard register 들을 살펴봅니다.

`Dummy function 없애기`

함수 init_dummy_function_start 의 영향력을 없앱니다.

4.2 언어-의존적인 부분을 초기화

각 언어마다 다르게 설정할 수 있도록 배려한 lang_hooks 부분을 사용하여 여러가지 언어 의존적인 부분을 초기화합니다.

`Front-end 초기화`

각 언어의 Front-end 초기화 부분을 담당하는 lang_hooks.init () 함수를 호출함으로써 초기화가 이루어진다.

`Assembler output 초기화`

어셈블리 code output 파일을 엽니다. 이것은 -fsyntax-only 가 활성화되어 있어도 수행하게 됩니다.

`Exception Handler 초기화`

여러 Exception 에 관한 사항을 다루게 됩니다.

`Optabs 초기화`

현재 target machine 을 위한 적당한 optabs 내용을 초기화하기 위해 한번 호출됩니다.

`Debug output 초기화`

이제 우리는 올바른 원래 파일이름을 가지고 있으므로 debug output 을 초기화 할 수 있습니다. debug_hooks 을 사용하게 됩니다.

4.3 Input 파일을 compile.

이 부분에서 실제 원시 파일을 parsing 하고 output 을 생성하게 됩니다.

`마지막 단계에서의 여러 DATA 초기화`

컴파일의 시작점에서 마지막 pass 에서의 data 를 초기화합니다.

`Branch-prob processing`

branch-prob processing 을 위한 file-level 초기화를 수행합니다.

`Parsing`

전체 파일을 해석하는 parser 를 호출합니다. (각 함수마다 rest_of_compilation 를 호출)

`Binding stack 청소`

각 언어마다의 Binding stack 관련 부분을 처리.

`나머지 (보충 부분)`

아직 이 부분에 대한 설명은 제가 자세히 말씀을 못드릴 것 같아 이렇게 마무리합니다. 저도 공부하여 나중에 이 부분을 채우도록 하겠습니다.

제 5 절 마치며...

3 주 강의도 이렇게 마치는 군요. 이번에 전체적인 구조에 대해서 자세히 말하고 싶었는데, 생각보다 저의 지식에 대한 바닥이 일찍들어나다 보니 많은 설명을 못한 것 같습니다. 아쉽군요. 하지만 제가 생각 날때마다 계속 문서 업데이트를 하기 때문에 자주 지켜봐 주시기 바랍니다.