개발자측면에서 바라본 GCC 속의 option 들

정원교

2004년 2월 23일

목 차

제 1 절 5 주째 강의를 시작하며 **1**

제 2 절 GCC 옵션에는 어떤 것이 있을까? **1**

제 3 절 최적화 옵션에 따른 영향 **12**

제 4 절 각 옵션이 미치는 전역 변수 **13**

제 5 절 C 언어 lang_hook 에서의 옵션 처리 부분 **21**

제 6 절 5 주째 강의를 마치며 **22**

제 1 절 5 주째 강의를 시작하며

5 주째 강의의 문이 열렸습니다. 환영합니다.
이번 주 강의의 제목은 "개발자측면에서 바라본 GCC 속의 option 들"입니다. 이게 무슨 말인고 하니 GCC 는 compiler 로써 여러 architecture 로 porting 되어 있으며 많은 기능을 포함하고 그에 따른 의존성도 있습니다. 한마디로 아주 옵션이 많습니다. 물론 이 많은 옵션에 대한 설명을 할려면 많은 시간이 걸릴 것은 자명합니다. 모든 옵션에 대한 설명을 할 수 없지만 중요한 것에 대해서는 모두 설명을 하고 그에 따른 개발자 입장에서 GCC 를 바라볼 수 있도록 해보자.

제 2 절 GCC 옵션에는 어떤 것이 있을까?

GCC 는 많은 옵션을 가지고 있습니다. 이에 대한 설명만으로 충분히 사람의 머리를 아프게 만들 수 있을 정도입니다. 사용자가 GCC 를 사용할 때 바라보는

```
# gcc -O2 -Wall -DGCC_MANSE test.c
```

와 같은 모습으로 각 -O2 옵션, -Wall 옵션과 같이 이 옵션에 대해서 설명을 하는 것이 아니라, GCC source 파일속에 있는 이 option 에 해당하는 전역 변수가 어떻게 변화되고 설정되는지에 대해 볼 것이다.

먼저 GCC 에서 옵션에 해당하는 전역 변수에는 어떤 것이 있는지 알아 봅시다. 이 강의에서 즐겨 사용하는 열거형으로 나열해 봅니다. 알파벳 순으로 정리되어 있습니다. 분량이 상당히 많으니, 참조형식으로 봐주시기 바랍니다. :) 물론 직접 하나 하나 읽어 보실 분도 환영합니다.

```
align_jumps
align_jumps_log
align_jumps_max_skip
align_functions
align_functions_log
align_labels
```

`align_labels_log`
`align_labels_max_skip`
`align_loops`
`align_loops_log`
`align_loops_max_skip`

-falign-* flag 의 값들 : Code 내에서 각 라벨들을 얼마나 align 할 것인가. 0 은 '기본값 사용', 1 은 'align 하지 않음' 을 의미합니다. 각각의 변수에는 .align output 을 위한 _log 별형이 따로 존재하는데, 이는 변수보다 작지 않을 경우 제곱의 값을 갖습니다.

`dollars_in_ident`

0 이 아닌 경우 '$' 가 식별자(identifier)내에서 사용될 수 있습니다.

`exit_after_options`

값이 0 이 아닐 경우 우리는 option 을 해석한 후 끝마칩니다.

`flag_allow_single_precision`

값이 0 이 아닐 경우 우리가 일반적으로 traditional 로 하고 있을 지라도 single precision math 를 허락하는 것을 의미합니다.

`flag_argument_noalias`

만약 포인터 argument 들이 각각 다른 것을 alias 하고 있다면 0. C 에서 true.
만약 포인터 argument 들이 각각 다른 것에는 alias 하지 않지만 전역 변수에게는 alias 하고 있다면 1.
만약 포인터 argument 들이 각각 다른 것에도 alias 하지 않고 전역 변수에게도 alias 하고 있지 않다면 2. Fortran 에서 true.
이 값은 C 를 위해 기본값 0 으로 되어 있음.

`flag_asynchronous_unwind_tables`

값이 0 이 아닐 각 insn boundary 에 정확한 frame unwind info table 을 생성함을 의미합니다.

`flag_bounded_pointers`

-fbounded-pointers 는 gcc 로 하여금 세가지 의미를 가지는 composit pointer 로써 compiler 하게 합니다 : 1) Pointer 값, 2) Referent object 의 base address, 3) Referent object 의 바로 끝 뒤의 address. Base 와 Extent 는 runtime bounds checking 을 수행할 수 있도록 허락합니다. -fbounded-pointers 는 -fcheck-bounds 에 영향을 미칩니다.

`flag_bounds_check`

-fcheck-bounds 는 gcc 로 하여금 배열 bounds check 를 생성하도록 합니다.
C, C++ 용: 기본값은 flag_bounded_pointers 의 값.
For ObjC: 기본값은 off.
For Java: 기본값은 on.
For Fortran: 기본값은 off.
For CHILL: 기본값은 off.

`flag_branch_on_count_reg`

flag_branch_on_count_reg 는 count register 상에 cheaper branch 로써 add-1, compare, branch tupple 로 교체를 시도함을 의미합니다.

`flag_caller_saves`

-fcaller-saves 가 설정될 경우: 만약 레지스터내의 값을 할당하는 것이 더 좋은 코드를 생성해낸다면 함수 호출사이에 저장될 필요가 있는 값들을 할당한다. 선택적으로 이제 사람들이 그것에 대해 검사를 할 수 있게 되었다. toplev.c 파일을 보면 이 전역 변수의 경우 DEFAULT_CALLER_SAVES 가 정의되어 있는가에 따라 기본값이 변경이 됩니다. 저의 환경에서는 이것이 정의되어 있지 않군요.

`flag_complex_divide_method`

0 은 complex divide 의 straightforward implementation 를 받아들일 수 있음을 의미.
1 은 input 들의 wide ranges 는 complex divide 용이여야 함을 의미.
2 는 complex divide 용으로 C99-같은 요구 사항을 의미 (아직 구형되지 않음).

`flag_cond_mismatch`

값이 0 이 아닐 경우 Conditional expression 에서의 type 이 맞지 않을 경우를 허락합니다. 단순히 그들의 값을 'void' 로 만듭니다.

`flag_const_strings`

값이 0 이 아닐 경우 문자열 constant 들을 type 'const char *' 로 부여합니다. 이것을 그들로 부터 기타 다른 경고를 얻기 위해서 입니다. 이러한 경고문들은 전체적으로 ANSI 로 규격화되어 있는 프로그램들을 제외하고는 매우 유용하게 사용될 수 있을 것입니다.

`flag_cprop_registers`

아직 정확한 설명이 없음.
Copy propagation registers 의 준말입니다.

`flag_cse_follow_jumps`

-fcse-follow-jumps 가 설정될 경우: 더 큰(많은) 일을 하기 위해 cse follow jump 들을 가지고 있습니다.

`flag_cse_skip_blocks`

-fcse-skip-block 가 설정될 경우: Block 과 관련되는 brach(분기) 에 따른 cse 를 가지고 있습니다.

`flag_data_sections`

값이 0 이 아닐 경우 임의의 section 이름의 지정을 지원하며 section 의 수 또한 제한이 없는 그러한 platform 상에서 그 자신의 section 에 각 data 를 넣을 수 있게 함을 의미합니다.

`flag_debug_asm`

-dA 는 컴파일러에 의해 생성되는 어셈블리 코드내의 (가독성을 높일 수 있는) commentary infomation 를 debug 할 수 있게 합니다. 이 옵션은 일반적으로 컴파일러가 생성해낸 어셈블리 코드를 (컴파일러 그 자체를 debugging 해야하는) 실제적으로 읽을 필요가 있는 사람들을 위해 사용됩니다. 현재 이 스위치는 오직 dwarfout.c 에서만 사용됩니다; 하지만 이것은 어셈블러 파일내에 debug infomation 을 출력하기 위한 포괄적인 내용을 포함할려고 한다.

`flag_defer_pop`

-fdefer-pop 이 설정되었을 경우: 각 function call 후에 args 들을 pop 하지 않습니다. – 한 명령어(one insns)로 많은 call 들의 args 들을 pop 할 때까지 그것들을 저장해 놓습니다.

`flag_delayed_branch`

값이 0 이 아닐 경우 delayed brach slots 속에 schedule 함을 의미합니다. (지원할 경우)

`flag_delete_null_pointer_checks`

설정될 경우 사용되지 않는 null pointer 를 검사하는데
global dataflow analysis 를 사용함을 의미합니다.

`flag_dump_rtl_in_asm`

-dP 는 어셈블리에 comment 로써 rtl 을 삽입하도록 합니다.

`flag_eliminate_dwarf2_dups`

값이 0 이 아닐 경우 dwarf2 duplicate elimination 을 수행합니다.

`flag_errno_math`

값이 0 이 아닐 경우 front end 가 built-in SQRT 와 같이 일반적으로 math operation 들에 의해 유지되는 'errno' 를 요구하는 것을 의미합니다.

`flag_exceptions`

값이 0 이 아닐 경우 exception handling 을 위한 extra code 를 생성하고 exception handling 을 활성화함을 의미합니다.

`flag_expensive_optimizations`

-fexpensive-optimizations 가 설정될 경우: 비교적 비용이 많이 드는 기타 다른 optimization 를 수행합니다.

`flag_float_store`

-ffloat-store 가 설정되었을 경우: Extended-precision register 내에 float 들과 double 들을 할당하지 않는다.

`flag_force_addr`

-fforce-addr 가 설정되어 있을 경우: 메모리를 참조하기 전에 메모리 주소를 레지스터에 load 합니다. 이것은 cse 보다 더 낳은 결과를 낳지만 compilation 이 느려집니다.

`flag_force_mem`

-fforce-mem 가 설정되어 있을 경우: 메모리의 값을 사칙연산을 수행하기 전에 레지스터로 메모리값을 load 합니다. 이것은 cse 보다 더 낳은 결과를 낳지만 compilation 이 느려집니다.

`flag_function_sections`

값이 0 이 아닐 경우 임의의 section 이름의 지정을 지원하며 section 의 수 또한 제한이 없는 그러한 platform 상에서 그 자신의 section 에 각 함수를 넣을 수 있게 함을 의미합니다.

`flag_gcse`

설정될 경우 global cse 를 수행함을 의미합니다.

`flag_gcse_lm`

값이 0 이 아닐 경우 gcse 의 수행동안 enhanced load motion 을 실행함을 의미합니다. 이것은 같은 장소(same location)에 저장하는 것이 발견될 때 그들을 죽이지(killing) 않음으로써 load 를 천천히 감아 올리도록 합니다.

`flag_gnu_linker`

-fgnu-linker 는 초기화를 위해 GNU linker 의 사용을 할 것인가를 지정합니다. (혹은 더 일반적으로 초기화를 다루는 것을 linker 라고 합니다.). -fno-gnu-linker 는 collect2 를 사용할 것임을 말합니다.

`flag_guess_branch_prob`

이것이 설정되어 있다면 분기 가능성(branch probability)들을 추측하길 시도합니다.

`flag_hosted`

값이 0 이 아닐 경우 우리는 내장(builtin) 함수들을 가지고 있고 main 의 반환형은 int 인 것을 의미합니다.

`flag_no_ident`

값이 0 이 아닌 경우 '#ident' directive 들을 무시함을 의미합니다. 0 은 그것들을 다룬다는 의미입니다. SVR4 target 상에서는 컴파일러를 인식하게끔 하는 문자열을 내보낼지 안할 지를 제어하는데 또한 사용됩니다.

`flag_inhibit_size_directive`

-finhibit-size-directive 는 ELF 용 .size 관련 output 을 금한다. 이것은 crtstuff.c 를 컴파일하기 위해서만 사용됩니다. 그리고 다른 시스템상에서 crtstuff.c 를 위해 필요한 다른 효과(영향력)로 확장될 것입니다.

`flag_inline_functions`

-finline-functions 가 설정될 경우: Inline 함수로 정의하는 것이 더 좋아 보이는 함수의 경우 inline 하는 것을 승인할 때 사용.

`flag_inline_trees`

만약 우리가 inlining 을 수행해서는 않된다면 0 으로 설정.
만약 우리가 tree level 에서의 function call 로 확대해야 한다면 1 로 설정.
만약 우리가 *모든* 함수들을 inline 하는 것을 고려해야 한다면 2 로 설정.

`flag_instrument_function_entry_exit`

Entry 와 exit 에서의 call 들을 가지는 profiling 을 위한 instrument function 들.

`flag_isoc94`

값이 0 이 아닐 경우 C89 Amendment 1 기능들을 활성화함을 의미합니다.

`flag_isoc99`

값이 0 이 아닐 경우 C 의 다른 사촌인 ISO C99 를 사용함을 의미합니다.

`flag_gcse_sm`

값이 0 이 아닐 경우 gcse 수행 후 store motion 을 수행함을 의미합니다. 이것은 exit block 으로 stores closer 를 이동시키도록 시도합니다. 이것은 flag_gcse_lm 의 사용 없이는 매우 효율적이지 못합니다.

`flag_gen_aux_info`

값이 0 이 아닐 경우 우리는 .X 파일에 declaration info 를 저장함을 의미합니다.

`flag_keep_inline_functions`

-fkeep-inline-functions 가 설정될 경우: 우리가 함수를 어느 곳에도 inline 할 수 있도록 만든다 하더라도 Debugging 목적을 위해 그것의 정의 부분은 계속 유지합니다.

`flag_keep_static_consts`

값이 0 이 아닐 경우 우리는 최적화 옵션이 활성화되어 있든 아니든 상관없이 static const 변수들을 내보내야 합니다.

`flag_move_all_movables`

값이 0 이 아닐 경우 loop 내에 존재하는 모든 invariant computation 들을 loop 밖으로 이동시킵니다.

`flag_merge_constants`

이것은 constant section 의 constant 들을 merge 하도록 합니다. 만약 값이 1 이라면 string contant 들과 contant pool 에서의 constant 를 merge 하고 값이 2 라면 추가적으로 constant variable 들도 merge 합니다.

`flag_no_asm`

0 이 아닐 경우 키워드 'asm' 을 인식하지 않습니다.

`flag_no_common`

값이 0 이 아닐 경우 기본으로 common storage 내에 초기화되지 않은 global data 를 놓지 않음을 의미합니다.

`flag_no_function_cse`

값이 0 이 아닐 경우 constant function 들의 주소를 레지스터에 넣지 않습니다. 이 기능은 assembly output 이 상이한 교환(strange substitution)들이 일어나는 Unix kernel 을 컴파일시 사용됩니다.

`flag_no_inline`

값이 0 이 아닐 경우 함수들은 inline 되어지지 않을 것을 의미합니다.

`flag_no_peephole`

값이 0 이 아닐 경우 define_optimization peephole opt 들의 사용을 금한다.

`flag_noniso_default_format_attributes`

값이 0 이 아닐 경우 ISO C 에서 지정하지 않은 함수들을 위한 기본 format_arg attribute 들을 추가하는 것을 의미합니다.

`flag_omit_frame_pointer`

-fomit-frame-pointer 가 설정되어 있을 경우: 원하지 않는다면 간단한 함수속에 frame pointer 를 만들지 않습니다.

`flag_optimize_sibling_calls`

설정될 경우 GCC 가 sibling 와 tail recursive call 들을 최적화하도록 허락합니다.

`flag_pack_struct`

모든 구조체에 __attribute__(packed) tag 를 붙입니다.

`flag_pcc_struct_return`

-fpcc-struct-return 가 설정되어 있을 경우: PCC 와 같은 방식으로 값을 반환합니다.

`flag_pedantic_errors`

값이 0 이 아닐 경우 특정 경고들을 오류로 변경함을 의미합니다. 보통 몇몇 표준에 적합하게 하기 위한 실패에 관한 경고들입니다.

`flag_peephole2`

이것은 sched2 를 수행하기 전에 peephole 단계를 수행합니다.

`flag_pic`

만약 우리가 pure (sharable) code 를 컴파일하는 중이라면 0 이 아닌 값을 가집니다.
만약 우리가 reasonable (예를 들면 offset table 내에서의 간단한 offset) pic 을 하고 있다면 값이 1 입니다. 만약 우리가 단순히 register offset 들을 수행할 수 있다면 값은 2 입니다.

`flag_prefetch_loop_arrays`

값이 0 이 아닐 경우 loop 내에서의 배열에 대해 prefetch optimization 을 활성화합니다.

`flag_pretend_float`

값이 0 이 아닌 겨우 target float 의 bit 들의 시험에서 그것의 결과가 true 가 아니더라도 그것이 OK 라고 말해버립니다. 결과치 code 는 잘못된 constants 들을 가지게 되겠지만 명령어들의 집합들은 원래 컴파일러가 만들게 되는 것과 같은 것을 가지게 됩니다.

`flag_really_no_inline`

값이 0 이 아닐 경우 우리는 -fno-inline 의 효능으로 inlining 을 원하지 않습니다. 단순히 tree inliner 가 우리를 꺼놓았기 때문이 아닙니다.

`flag_reduce_all_givs`

값이 0 이 아닐 경우 loop 내에 존재하는 모든 일반적인 induction variable 들을 strength reduce 합니다.

`flag_regmove`

설정될 경우 full register move optimization pass 를 수행합니다. 이것은 -O2 최적화의 기본 부분입니다.

`flag_rename_registers`

만약 register 들이 재명명(rename) 해야 한다면 0 이 아닌 값을 가집니다.

`flag_reorder_blocks`

값이 0 이 아닐 경우 basic block 들은 재정렬되어야 합니다.

`flag_rerun_cse_after_loop`

설정될 경우 loop optimization 후에 cse 를 재실행함을 의미합니다. 이 수행은 compilation 시간을 약 20% 정도 증가시키고 몇몇 공통 표현식을 잡아냅니다.

`flag_rerun_loop_opt`

설정될 경우 loop optimization 를 두번 수행함을 의미합니다.

`flag_schedule_insns`

기본 block 내에 있는 insn 들을 schedule 함을 의미합니다.
(local_alloc 를 수행하기 전에)

`flag_schedule_insns_after_reload`

global_alloc 후에 insn 들을 schedule 함을 의미합니다.

`flag_schedule_interblock`
`flag_schedule_speculative`
`flag_schedule_speculative_load`
`flag_schedule_speculative_load_dangerous`

다음의 flag 들은 단지 레지스터 할당 전에의 스케줄링시에만 효력을 가지고 있습니다:

flag_schedule_interblock 는 basic block 사이로의 (across) insn 들을 스케줄함을 의미합니다.
flag_schedule_speculative 는 non-load insn 들의 speculative motion 을 허락함을 의미합니다.
flag_schedule_speculative_load 는 몇몇 load insn 들의 speculative motion 을 허락함을 의미합니다.
flag_schedule_speculative_load_dangerous 는 더 많은 load insn 들의 speculative motion 을 허락함을 의미합니다.

`flag_shared_data`

값이 0 이 아닐 경우 text 공유(the text shared)를 지원할 경우 그렇게 만듭니다.

`flag_short_enums`

값이 0 이 아닐 경우 요구하는 크기의 바이트만 enum type 에게 부여하는 것을 의미함.

`flag_signed_bitfields`

값이 0 이 아닐 경우 bitfields 를 'unsigned' 으로 한다고 말하지 않을 경우 signed 으로 취급함을 의미합니다.

`flag_signed_char`

'char' 가 sign 이여야 한다면 값이 0 이 아님.

`flag_single_precision_constant`

아직 정확한 설명이 없음.

`flag_ssa`

SSA 를 활성화합니다.

`flag_ssa_ccp`

ssa conditional constant propagation 을 활성화합니다.

`flag_ssa_dce`

ssa aggressive dead code elimination 을 활성화합니다

`flag_stack_check`

Stack overflow 를 검사하는 code 를 추가; 또한 동적으로 할당되는 큰 object 를 만들어 낼 수 있음.

`flag_strength_reduce`

설정될 경우 loop.c 내에서의 strength-reduction 을 활성화하도록 한다.

`flag_strict_aliasing`

만약 우리가 (언어-의존적인) alias analysis 를 해야한다면 설정합니다. 보편적으로 이 analysis 는 특정 type 들의 표현식은 특정 다른 type 들의 표현식에 alias 되어 있지 않다고 가정을 합니다. 오직 (일반 모드의) alias analysis 가 활성화되어 있을 때만 사용됩니다.

`flag_syntax_only`

값이 0 이 아닐 경우 단지 문법 검사만 합니다; 아무 output 이 없음.

`flag_thread_jumps`

-fthread-jumps 가 설정되었을 경우: loop 의 jump optimize output 을 가지고 있습니다.

`flag_traditional`

0 이 아닐 경우 PCC 가 수행한 것 같이 몇몇 부분을 똑같이 수행합니다.

`flag_trapping_math`

0 은 floating-point math operations 가 (user-visible) trap 을 생성할 수 없음을 의미합니다. 예를들면 이 경우는 nonstop IEEE 754 arithmetic 에 해당합니다.

`flag_unroll_all_loops`

값이 0 이 아닐 경우 unroll.c 내에서의 loop unrolling 을 활성화합니다. 모든 loop 들이 unroll 됩니다. 하지만 이것은 일반적으로 성공적인 컴파일의 조건이 아닙니다.

`flag_unroll_loops`

값이 0 이 아닐 경우 unroll.c 내에서의 loop unrolling 을 활성화합니다. 기능은 compile-time (UNROLL_COMPLETELY, UNROLL_MODULO) 시 반복(iteration) 횟수가 계산가능한 것이나 run-time (UNROLL_MODULO 로 전제 조건으로 되어 있고) 시 unroll 되어 있는 loop 들에 한해서만 수행됩니다.

`flag_unsafe_math_optimizations`

값이 0 이 아닐 경우 속도의 이익을 위해 불안전한 floating-point math optimization 의 수행을 허락합니다. IEEE compliance 는 이에 대한 보장을 하지 않으며 operation 은 그들의 argument 들과 result 들이 “normal” 이라는 가정 하에 허락합니다. (예를 들면 SQRT 의 nonegative)

`flag_unwind_tables`

값이 0 이 아닌 경우 frame unwind info table 을 지원시 생성함을 의미합니다.

`flag_writable_strings`

-fwritable-strings 가 설정될 경우: String constant 들을 data segment 에 저장하고 그들을 uniquize 하지 않는다.

`flag_verbose_asm`

-fverbose-asm 는 컴파일로써 생성되는 어셈블리 코드내에 (가독성을 높일 수 있는) 다른 commentary infomation 을 생산해 냅니다. 이 옵션은 일반적으로 컴파일러가 생성해낸 어셈블리 코드를 (컴파일러 그 자체를 debugging 해야하는) 실제적으로 읽을 필요가 있는 사람들을 위해 사용됩니다. 기본으로 -fno-verbose-asm 는 다른 infomation 들을 생략하도록 하는데, 이것은 두개의 어셈블러 파일들을 비교할 때 사용됩니다.

`flag_volatile`

값이 0 이 아닐 경우 포인터를 통한 모든 reference 들은 volatile 임을 의미합니다.

`flag_volatile_global`

값이 0 이 아닐 경우 모든 global 과 extern 변수들을 volatile 로써 취급합니다.

`flag_volatile_static`

값이 0 이 아닐 경우 모든 static 변수들을 volatile 로 취급합니다.

`mesg_implicit_function_declaration`

값이 0 이 아닐 경우 묵시적 함수 선언의 사용에 관한 메제시를 의미하는데; 값 1 은 경고를, 값 2 는 오류를 뜻합니다.

`optimize`

0 이 아닐 경우 최적화를 수행합니다. -O. 특정 수의 값은 최적화의 수행 정도를 나타내는데 사용됩니다. 그래서 -O2 는 여기서 2 를 저장합니다. 하지만 최적화는 이 변수를 통해서 직접적으로 제어되는 것이 아니며 대신에 그들은 개별적인 'flag_...' 변수에 의해 제어됩니다. 하지만 이 변수에 기본값들이 변경되게 됩니다.

`optimize_size`

0 이 아닐 경우 size 를 위한 최적화를 수행합니다. -Os. 이 값은 단지 0 혹은 1 만 유효합니다. optimize_size 가 0 이 아닐 경우 optimize 변수는 기본값을 2 를 가지지만 특정 individual code bloating 최적화는 disable 됩니다.

`pedantic`

0 이 아닐 경우 -pedantic 스위치 활성화 : 표준 스펙에 맞지 않는 것을 할 경우 어떤 것이든 경고합니다.

`profile_arc_flag`

값이 0 이 아닐 경우 program flow graph arc 들을 profile 하기위한 code 를 생성합니다.

`profile_flag`

값이 0 이 아닐 경우 profiling 을 하기위한 code 를 생성합니다.

`warn_about_return_type`

Return type 이 기본(defaluted)으로 설정된 함수를 'grokdeclarator'가 처리할 때 이에 대한 경고 메세지의 발생이 요구된다면 값이 0 이 아닌 것으로 설정합니다.

`warn_bad_function_cast`

값이 0 이 아닐 경우 return type 과 맞지 않는 type 에 함수 호출을 casting 할 때 경고함을 의미합니다. 예를 들어 설명하면 sqrt 혹은 malloc 함수에 대해 이전에 선언되어 있지 않을 때, (float) sqrt() 혹은 (anythiing *) malloc() 와 같이 사용할 때를 가르킵니다.

`warn_cast_qual`

값이 0 이 아닐 경우 pointer target type 에 의해 type qualifier 를 없앨 소지가 있는 pointer cast 들에 관해 경고함을 의미합니다.

`warn_char_subscripts`

Type char 를 가지는 subscript 에 관해 경고합니다.

`warn_conversion`

만약 type 변환이 수행될 때 결과를 혼동할 수 있을 경우 경고합니다.

`warn_float_equal`

Floating point number 의 equality 테스팅에 관해 경고합니다.

`warn_format`

Fomatted I/O 함수들의 호출에서 format/argument 의 형태가 변태(-_-;)적일 때 경고합니다. (*printf, *scanf, strftime, strfmon, 기타 등등.).

`warn_format_extra_args`

Format 에서 argument 의 수가 초과했을 때 경고합니다.

`warn_format_nonliteral`

Non-literal format argument 들에 관해 경고합니다.

`warn_format_security`

Format 함수들의 호출 부분에서 보안 문제가 발생할 수 있는 것에 관해 경고합니다.

`warn_format_y2k`

strftime format 에서의 Y2K 문제점들에 관해 경고합니다.

`warn_implicit_int`

값이 0 이 아닐 경우 묵시적 int 의 사용에 관해 경고함을 의미합니다.

`warn_inline`

값이 0 이 아닐 경우 만약 inline 함수가 너무 크면 경고합니다.

`warn_long_long`

값이 0 이 아닐 경우 '-pedantic' 이 설정되었을 때 long long 의 사용에 관해 경고함을 의미합니다.

`warn_main`

만약 main 함수가 의심스러운 형태이면 경고합니다.

`warn_missing_braces`

만약 초기화자(initializer) 가 완전히 괄호 처리되지 않았다면 경고합니다.

`warn_missing_declarations`

값이 0 이 아닐 경우 분리된 이전 prototype decl 가 없는 어떠한 전역 함수 def 에 대해 경고함을 의미합니다.

`warn_missing_format_attribute`

Format attribute 에 관해 빠져 있는 함수들에 관해 경고합니다.

`warn_missing_prototypes`

값이 0 이 아닐 경우 분리된 이전 prototype decl 가 없는 어떠한 전역 함수 def 에 대해 경고함을 의미합니다.

`warn_multichar`

값이 0 이 아닐 경우 multicharacter literal 들의 사용에 관해 경고합니다.

`warn_nested_externs`

값이 0 이 아닐 경우 file-scope level 에서가 아닌 오브젝트들의 외부 선언들에 관한 경고와 file-scope level 에서가 아닌 모든 함수(extern 이든 static 이든)의 선언에 관한 경고함을 의미합니다. 우리가 묵시적 함수 선언들을 제외하였다는 것을 참고하십시오. 그러한 것에 관한 경고 메세지를 얻기 위해서는 -Wimplicit 를 사용하십시오.

`warn_parentheses`

만약 () 를 추가할 필요가 있다면 경고합니다.

`warn_pointer_arith`

값이 0 이 아닐 경우 sizeof(function) 혹은 함수 포인터들의 더하기/빼기에 관한 경고 발생을 의미합니다.

`warn_redundant_decls`

값이 0 이 아닐 경우 같은 단일 변수 혹은 함수에 대해 다수의(중복의) decl 들에 대한 경고함을 의미합니다.

`warn_sign_compare`

Signed 와 unsigned 값들의 비교에 관해 경고합니다. 만약 -1 이면 -Wsign-compare 도 -Wno-sign-compare 도 지정되지 않았다는 뜻입니다.

`warn_strict_prototypes`

값이 0 이 아닐 경우 이전의 prototype 없이 선언한 non-prototype function decl 들 혹은 non-prototyped def 들에 대해 경고함을 의미합니다.

`warn_traditional`

ANSI C 내에서 변경된 것을 의미하는 traditional construct 들에 관해 경고합니다.

`warn_uninitialized`

값이 0 이 아닐 경우 변수들이 초기화되기 전에 사용되었을 경우에 대해 경고합니다.

`warn_unknown_pragmas`

#pragma directive 들이 인식이 되지 않는다면 경고합니다.

지금까지 GCC 내에서 옵션과 관련되어 사용되는 변수에 대해서 보셨습니다. 분량이 상당히 많이 있습니다. 비록 제가 정리한다고 했지만, 빼먹은 부분도 조금 있을 듯합니다. 그 부분에 대해서는 피드백 혹은 저의 개인적인 노력으로 보충하겠습니다.

제 3 절 최적화 옵션에 따른 영향

GCC 는 다른 컴파일러와 마찬가지로 최적화 기능 수행을 합니다. 그리고 최적화를 어느 정도로 하느냐에 따라서 선택되어지고 설정되어 지는 option 들이 달라지게 됩니다. 이 section 에서는 최적화가 어떻게 설정되느냐에 따라서 어떻게 설정이 선택되어지고 설정되어 지는지에 대해서 알아 봅니다.

- `optimize 의 값이 0 일 경우`

  최적화 기능을 수행을 하지 않습니다. 이의 경우
  flag_merge_constants 를 0 으로 설정합니다. 위에서 이 옵션에 대해서 설명하였으니 참고하시기 바랍니다. 그리고 아래와 같이 설정합니다.

  flag_no_inline = 1; warn_inline = 0;

- `optimize 의 값이 1 이상일 경우`

  최적화 level 이 1 이상일 경우 아래와 같은 옵션이 설정이 됩니다.

  - flag_defer_pop = 1;
  - flag_thread_jumps = 1;

  - flag_delayed_branch = 1;
    DELAY_SLOTS 가 정의되어 있어야 설정된다.
  - flag_omit_frame_pointer = 1;
    CAN_DEBUG_WITHOUT_FP 가 정의되어 있어야 설정된다.
  - flag_guess_branch_prob = 1;
  - flag_cprop_registers = 1;

- `optimize 의 값이 2 이상일 경우`

  최적화 level 이 2 이상일 경우 아래와 같은 옵션이 설정이 됩니다.
  - flag_optimize_sibling_calls = 1;
  - flag_cse_follow_jumps = 1;
  - flag_cse_skip_blocks = 1;
  - flag_gcse = 1;
  - flag_expensive_optimizations = 1;
  - flag_strength_reduce = 1;
  - flag_rerun_cse_after_loop = 1;
  - flag_rerun_loop_opt = 1;
  - flag_caller_saves = 1;
  - flag_force_mem = 1;
  - flag_peephole2 = 1;
  - flag_schedule_insns = 1;
    INSN_SCHEDULING 가 정의되어 있어야 수행됩니다.
  - flag_schedule_insns_after_reload = 1;
    INSN_SCHEDULING 가 정의되어 있어야 수행됩니다.
  - flag_regmove = 1;
  - flag_strict_aliasing = 1;
  - flag_delete_null_pointer_checks = 1;
  - flag_reorder_blocks = 1;

- `optimize 의 값이 3 이상일 경우`

  최적화 level 이 3 이상일 경우 아래와 같은 옵션이 설정됩니다.
  - flag_inline_functions = 1;
  - flag_rename_registers = 1;

제 4 절 각 옵션이 미치는 전역 변수

이 부분에서는 GCC 에 존재하는 많은 옵션들이 어떻게 설정되는냐에 따라서 컴파일러내에 존재하는 전역 변수가 어떻게 설정되는지 살펴보도록 하자.

`-ftraditional / -traditional`

```
flag_traditional = 1;
flag_writable_strings = 1;
```

`-fallow-single-precision`

```
flag_allow_single_precision = 1;
```

`-fhosted / -fno-freestanding`

```
  flag_hosted = 1;
  flag_no_builtin = 0;
```

-ffreestanding / -fno-hosted

```
  flag_hosted = 0;
  flag_no_builtin = 1;

  if (warn_main == 2)
    warn_main = 0;
```

-fnotraditional / -fno-traditional

```
  flag_traditional = 0;
  flag_writable_strings = 0;
```

-std=

iso9899:1990 / c89

```
      flag_isoc94 = 0;
      flag_traditional = 0;
      flag_writable_strings = 0;
      flag_no_asm = 1;
      flag_no_nonansi_builtin = 1;
      flag_noniso_default_format_attributes = 0;
      flag_isoc99 = 0;
```

iso9899:199409

```
      flag_isoc94 = 1;
      flag_traditional = 0;
      flag_writable_strings = 0;
      flag_no_asm = 1;
      flag_no_nonansi_builtin = 1;
      flag_noniso_default_format_attributes = 0;
      flag_isoc99 = 0;
```

iso9899:199x / iso9899:1999 / c9x / c99

```
      flag_traditional = 0;
      flag_writable_strings = 0;
      flag_no_asm = 1;
      flag_no_nonansi_builtin = 1;
      flag_noniso_default_format_attributes = 0;
      flag_isoc99 = 1;
      flag_isoc94 = 1;
```

gnu89

```
      flag_traditional = 0;
      flag_writable_strings = 0;
      flag_no_asm = 0;
      flag_no_nonansi_builtin = 0;
      flag_noniso_default_format_attributes = 1;
      flag_isoc99 = 0;
      flag_isoc94 = 0;
```

```
  gnu9x / gnu99

      flag_traditional = 0;
      flag_writable_strings = 0;
      flag_no_asm = 0;
      flag_no_nonansi_builtin = 0;
      flag_noniso_default_format_attributes = 1;
      flag_isoc99 = 1;
      flag_isoc94 = 1;

-fdollars-in-identifiers

  dollars_in_ident = 1;

-fno-dollars-in-identifiers

  dollars_in_ident = 0;

-fsigned-char

  flag_signed_char = 1;

-funsigned-char

  flag_signed_char = 0;

-fno-signed-char

  flag_signed_char = 0;

-fno-unsigned-char

  flag_signed_char = 1;

-fsigned-bitfields / -fno-unsigned-bitfields

  flag_signed_bitfields = 1;
  explicit_flag_signed_bitfields = 1;

-fshort-enums

  flag_short_enums = 1;

-fno-short-enums

  flag_short_enums = 0;

-fshort-wchar

  flag_short_wchar = 1;

-fno-short-wchar

  flag_short_wchar = 0;

-fcond-mismatch

  flag_cond_mismatch = 1;

-fno-cond-mismatch

  flag_cond_mismatch = 0;
```

```
-fshort-double

  flag_short_double = 1;

-fno-short-double

  flag_short_double = 0;

-fasm

  flag_no_asm = 0;

-fno-asm

  flag_no_asm = 1;

-fbuiltin

  flag_no_builtin = 0;

-fno-builtin

  flag_no_builtin = 1;

-ansi

  flag_isoc94 = 0;
  flag_traditional = 0;
  flag_writable_strings = 0;
  flag_no_asm = 1;
  flag_no_nonansi_builtin = 1;
  flag_noniso_default_format_attributes = 0;
  flag_isoc99 = 0;

-Werror-implicit-function-declaration

  mesg_implicit_function_declaration = 2;

-Wimplicit-function-declaration

  mesg_implicit_function_declaration = 1;

-Wno-implicit-function-declaration

  mesg_implicit_function_declaration = 0;

-Wimplicit-int

  warn_implicit_int = 1;

-Wno-implicit-int

  warn_implicit_int = 0;

-Wimplicit

  warn_implicit_int = 1;

  if (mesg_implicit_function_declaration != 2)
    mesg_implicit_function_declaration = 1;

-Wno-implicit
```

```
  warn_implicit_int = 0;
  mesg_implicit_function_declaration = 0;
-Wlong-long
  warn_long_long = 1;
-Wno-long-long
  warn_long_long = 0;
-Wwrite-strings
  flag_const_strings = 1;
-Wno-write-strings
  flag_const_strings = 0;
-Wcast-qual
  warn_cast_qual = 1;
-Wno-cast-qual
  warn_cast_qual = 0;
-Wbad-function-cast
  warn_bad_function_cast = 1;
-Wno-bad-function-cast
  warn_bad_function_cast = 0;
-Wno-missing-noreturn
  warn_missing_noreturn = 0;
-Wmissing-format-attribute
  warn_missing_format_attribute = 1;
-Wno-missing-format-attribute
  warn_missing_format_attribute = 0;
-Wpointer-arith
  warn_pointer_arith = 1;
-Wno-pointer-arith
  warn_pointer_arith = 0;
-Wstrict-prototypes
  warn_strict_prototypes = 1;
-Wno-strict-prototypes
  warn_strict_prototypes = 0;
-Wmissing-prototypes
```

```
  warn_missing_prototypes = 1;
-Wno-missing-prototypes
  warn_missing_prototypes = 0;
-Wmissing-declarations
  warn_missing_declarations = 1;
-Wno-missing-declarations
  warn_missing_declarations = 0;
-Wredundant-decls
  warn_redundant_decls = 1;
-Wno-redundant-decls
  warn_redundant_decls = 0;
-Wnested-externs
  warn_nested_externs = 1;
-Wno-nested-externs
  warn_nested_externs = 0;
-Wtraditional
  warn_traditional = 1;
-Wno-traditional
  warn_traditional = 0;
-Wformat=숫 자
  warn_format = 숫 자;
  warn_format_y2k = 숫 자;
  warn_format_extra_args = 숫 자;
  if (숫 자 != 1)
    {
      warn_format_nonliteral = 숫 자;
      warn_format_security = 숫 자;
    }
-Wformat
  warn_format = 1;
  warn_format_y2k = 1;
  warn_format_extra_args = 1;
-Wno-format
  warn_format = 0;
  warn_format_y2k = 0;
  warn_format_extra_args = 0;
  warn_format_nonliteral = 0;
  warn_format_security = 0;
```

```
-Wformat-y2k
  warn_format_y2k = 1;
-Wno-format-y2k
  warn_format_y2k = 0;
-Wformat-extra-args
  warn_format_extra_args = 1;
-Wno-format-extra-args
  warn_format_extra_args = 0;
-Wformat-nonliteral
  warn_format_nonliteral = 1;
-Wno-format-nonliteral
  warn_format_nonliteral = 0;
-Wformat-security
  warn_format_security = 1;
-Wno-format-security
  warn_format_security = 0;
-Wchar-subscripts
  warn_char_subscripts = 1;
-Wno-char-subscripts
  warn_char_subscripts = 0;
-Wconversion
  warn_conversion = 1;
-Wno-conversion
  warn_conversion = 0;
-Wparentheses
  warn_parentheses = 1;
-Wno-parentheses
  warn_parentheses = 0;
-Wreturn-type
  warn_return_type = 1;
-Wno-return-type
  warn_return_type = 0;
```

-Wsequence-point

  warn_sequence_point = 1;

-Wno-sequence-point

  warn_sequence_point = 0;

-Wcomment
-Wno-comment
-Wcomments
-Wno-comments
-Wtrigraphs
-Wno-trigraphs
-Wundef
-Wno-undef
-Wimport
-Wno-import

  이 옵션의 경우 CPP 에서 다루게 됩니다.

-Wmissing-braces

  warn_missing_braces = 1;

-Wno-missing-braces

  warn_missing_braces = 0;

-Wmain

  warn_main = 1;

-Wno-main

  warn_main = -1;

-Wsign-compare

  warn_sign_compare = 1;

-Wno-sign-compare

  warn_sign_compare = 0;

-Wfloat-equal

  warn_float_equal = 1;

-Wno-float-equal

  warn_float_equal = 0;

-Wmultichar

  warn_multichar = 1;

-Wno-multichar

  warn_multichar = 0;

-Wdiv-by-zero

```
  warn_div_by_zero = 1;

-Wno-div-by-zero

  warn_div_by_zero = 0;

-Wunknown-pragmas

  warn_unknown_pragmas = 2;

-Wno-unknown-pragmas

  warn_unknown_pragmas = 0;

-Wall

  if (warn_uninitialized != 1)
    warn_uninitialized = 2;
  warn_implicit_int = 1;
  mesg_implicit_function_declaration = 1;
  warn_return_type = 1;
  warn_unused_function = 1;
  warn_unused_label = 1;
  warn_unused_parameter = -1;
  warn_unused_variable = 1;
  warn_unused_value = 1;
  warn_switch = 1;
  warn_format = 1;
  warn_format_y2k = 1;
  warn_format_extra_args = 1;
  warn_char_subscripts = 1;
  warn_parentheses = 1;
  warn_sequence_point = 1;
  warn_missing_braces = 1;
  warn_main = 2;
  warn_unknown_pragmas = 1;
```

제 5 절 C 언어 lang_hook 에서의 옵션 처리 부분

C 언어가 사용하는 lang_hook 에 대한 설명은 이전 강의에서 언급하였으니 더 자세한 내용은 필요하지 않을 듯 싶습니다. 이 부분에서는 C 언어의 lang_hook 에서 옵션과 관련되어 있는 것만 살펴보도록 하겠습니다. GCC 에서의 옵션 처리는 앞에서 언급한 대로

```
static void parse_options_and_default_flags (int argc, char **argv);
```

함수에 선언되어 있습니다. 여러분들도 GCC code 를 보시면 아시게 되겠지만 이 함수내에서 사용하는 lang_hook 들로는 3 개가 있습니다.

- (*lang_hooks.init_options) ();
- (*lang_hooks.decode_option) (argc - i, argv + i);
- (*lang_hooks.post_options) ();

여기서 각각의 hook 마다 연결되어 있는 함수와 그에 대한 설명은 이전 강의에서 하였습니다. 여기에서는 이 함수내에서 GCC 옵션 전역변수가 어떤 것이 영향을 받는지에 대해서 간략하게 설명하겠습니다.

```
(*lang_hooks.init_options) ();
```

이 함수에서는 GCC 옵션을 초기화하는 부분이라고 생각할 수 있으나, 예상과는 약간 거리가 멉니다. 다른 전역 변수인 c_language 와 parse_in 변수를 여기서 설정하게 되며, flag_bounds_check 전역 변수를 -1 로 설정하는 것이 GCC 옵션 전역변수에 영향을 미치는 전부입니다.

```
(*lang_hooks.decode_option) (argc - i, argv + i);
```

이 함수에서 GCC 에서 C 언어에 대한 거의 모든 옵션들을 다룹니다. 방대한 량의 strncmp 함수를 보실 수 있습니다. 여기서 유심히 보셔야 할 부분은 이 옵션으로 인해 어떤 전역 변수들이 영향을 받게 되는 가에 대한 것입니다.

```
(*lang_hooks.post_options) ();
```

이 함수가 호출될 시점은 command line 으로 부터 입력받은 모든 옵션이 해석된 후에 호출되게 됩니다. 여기서는 위에서 처리못한 cpp 옵션들을 처리하게 되면 다른 무결성 검사를 하게 됩니다.

비록 여기에서는 간단하게 설명하였지만, 내부에 선언되어 있는 함수들과 그를 구성하는 구조체들과 전역 변수들의 의미를 열거할려면 한줄의 코드로도 한 주의 강의를 채울 수 있을 정도 입니다. 그것까지 포함하기에는 주제와 맞지 않는 성격이 많기 때문에 다음 주 혹은 다다음 주 강의 주제로 잡도록 하겠습니다. 여기에서는 간단하게 언급하였습니다.

제 6 절 5 주째 강의를 마치며

이번 강의는 상당히 시간이 많이 소유된 것 같습니다. 이것 저것 조사할 것도 많았지만, -_-; 제가 직장인이 되다 보니까 회사일을 하는데, 많은 시간을 보내게 됩니다. 나중 강의도 조금 늦어질 수 있으나, 글을 쓰다가 피를 토하는 일이 끝까지 GCC 에 대한 글을 완성하겠습니다. 앞으로의 강의도 재미있게 읽어 주시기 바랍니다.