Lang hooks 에 대하여...

정원교

2004년 2월 23일

목 차

제 1 절 4 주째 강의를 시작하며

안녕하세요. 정원교입니다.
4 주차 강의를 시작합니다. 지금까지 강의를 시작하면서 GCC 에 대한 디렉토리 구조, 파일들의 역활, 그리고 3 주째에는 전체적인 구조 및 구성에 대해서 언급하였는데 이제부터는 조금씩 GCC 내부적인 구성요소에 대해서 알아가 보자 합니다. 먼저 첫번째로 구조체 전역 변수인 lang_hooks 에 대해서 알아 보겠습니다. 이번 주도 재미있게 읽어 주시기 바랍니다.

제 2 절 Lang hooks

2.1 Lang hooks 의 역활

Lang hooks 은 각 front-end 들이 다른 front-end 와는 별개로 자신 고유의 설정이나, 행동사항들이 필요할 경우 GCC 컴파일러가 그에 대한 수행을 할 수 있도록 마련한 인터페이스이다. 각 세부적인 사항과 역활은 아래에서 말할 것이다.

2.2 Lang hooks 의 구조

Lang hooks 은 각 front-end 들이 모두 자신의 고유 lang_hooks 들을 가지고 있습니다. 그에 대한 구조체는 $prefix/gcc/langhooks.h 에 선언되어 있으며, 그림 1 는 그에 대한 구조체의 요소가 어떻게 이루어져 있는지 그리고 그에 대한 구조체 구조를 나열한 것입니다.

이 강의는 C 언어를 기준으로 작성해 나가는데, 우리가 사용하는 C 언어가 사용하는 struct lang_hooks lang_hooks 은 $prefix/gcc/c-lang.c 에 선언되어 있습니다.

C 에서 선언한 lang_hooks 함수들의 기능과 역활에 알아보기 이전에 우선 struct lang_hooks 을 구성하는 요소들의 기능에 대해서 먼저 알아보자.

`name`

Front end 를 식별하기 위한 문자열. 예) “GNU C++”.

```
struct lang_hooks
{
  const char *name;

  size_t identifier_size;

  void (*init_options) PARAMS ((void));
  int (*decode_option) PARAMS ((int, char **));
  void (*post_options) PARAMS ((void));

  const char * (*init) PARAMS ((const char *));
  void (*finish) PARAMS ((void));

  void (*clear_binding_stack) PARAMS ((void));

  HOST_WIDE_INT (*get_alias_set) PARAMS ((tree));

  tree (*expand_constant) PARAMS ((tree));

  int (*safe_from_p) PARAMS ((rtx, tree));

  int (*staticp) PARAMS ((tree));

  bool honor_readonly;

  void (*print_statistics) PARAMS ((void));

  lang_print_tree_hook print_xnode;

  lang_print_tree_hook print_decl;
  lang_print_tree_hook print_type;
  lang_print_tree_hook print_identifier;

  void (*set_yydebug) PARAMS ((int));

  struct lang_hooks_for_tree_inlining tree_inlining;

  struct lang_hooks_for_tree_dump tree_dump;

  /* 여러분이 추가할 entry 들을 여기에 넣으십시오. 그리고 꼭
     langhooks-def.h 와 langhooks.c 를 맞게 조정해 주어야
     합니다.  */
};
```

그림 1: struct lang_hooks 의 구성

`identifier_size`

sizeof (struct lang_identifier), 이것은 make_node() 함수가 language-specific slot 들을 위한 충분히 큰 identifier node 들을 만드는데 사용됩니다.

`*init_options`

Front-end 를 구성하는데 사용되는 최초의 callback 으로써
decode_option 과 같은 다른 call 들이 호출되기 이전에 필요한 간단한 초기화를 수행합니다.

`*decode_option`

단일 option (보통 -f 혹은 -W 혹은 + 로 시작하는 것들) 을 해석하는데 호출되는 함수로써 인자로 option vector 를 가진다. 이것은 만약 어떤 option 을 해석하였다면 그것이 사용한 command-line 인자들의 번호를 반환해야 하며 만약 인식한 option 이 없다면 0 을 반환한다. 만약 이 함수가 음수를 반환하면 그의 절대값은 사용된 command-line 의 번호를 나타내지만 추가적으로 언어-독립적인 option 처리는 이 option 에 대해서 수행되면 안된다.

`*post_options`

모든 command line option 들이 해석된 후 호출됩니다. 요청된 무결성 체크, 수정 관련 부분을 처리해야 합니다. Complex 초기화는 "init" callback 을 위해 남겨두어야 하는데 GC 와 identifier hash 들이 여기와 그때 사이에 구성되기 때문이다.

`*init`

Front end 를 초기화하기 위해 post_options 이 수행된 후 호출됩니다. main 입력 파일이름이 인자값으로 건네지며 NULL 일 가능성도 있습니다; front end 는 원래 파일을 반환해야 합니다(예를 들면 foo.i -¿ foo.c). NULL 을 반환하는 경우 몇몇 정렬 부분에서 심각한 오류가 있었다는 것을 가르키며 더이상 compilation 을 수행할수 없어 즉시 finish hook 이 호출됩니다.

`*finish`

finalizer 로써 컴파일 끝에 호출됩니다.

`*clear_binding_stack`

binding stack 을 깨끗히 하기 위해 parsing 후에 즉시 호출됩니다.

`*get_alias_set`

Expression 혹은 type 으로 사용되는 alias set 을 인자로 하여 호출됩니다. 만약 언어가 특별히 수행하는 것이 없다면 -1 을 반환합니다.

`*expand_constant`

Constant 로써 처리되는 expression 을 인자로 하여 호출됩니다. 같은 expression 을 반환하거나 입력값과 같은 역활을 하는 언어-독립적인 contant 를 반환합니다.

`*safe_from_p`

Language-specific tree code 들을 위해서 safe_from_p 에 의해 호출되는 Hook. 언어의 front-end 는 만약 safe_from_p 가 알기를 원하는 어떠한 code 들을 가지고 있다면 hook 을 설치해야 한다. same_from_p 가 expression 의 TREE_OPERAND 들을 recursive 하게 탐색할 것이기 때문에 이 hook 는 그러한 자식들을 재시험해서는 안됩니다. 이 routine 은 아마 recursive 하게 safe_from_p 를 호출할 것입니다; 그것은 항상 TOP_P 매개변수로 '0' 을 건내주어야 합니다.

`*staticp`

Language-specific tree code 들을 위해 staticp 에 의해 호출되는 hook.

`honor_readonly`

만약 TYPE_READONLY 와 TREE_READONLY 가 항상 honor 되어야 한다면 0 이 아닌 값을 가짐.

`*print_statistics`

Front-end 는 이 hook 을 가지고 -fmem-report 와 같은 자신의 통계자료를 더할 수 있습니다. 이에 대한 output 은 stderr 이여야 합니다.

`print_xnode`

어떻게 표현해야 할지 알지 못하는 'x' 분류의 tree 가 존재할 때 print_tree 에 의해 호출됩니다.

`print_decl`
`print_type`
`print_identifier`

분류 (class) 'd' 와 분류 't', IDENTIFIER_NODE node 들의 언어-의존적인 부분을 출력하기 위해 호출됩니다.

`*set_yydebug`

Command line 상에 -dy 와 같은 옵션이 주어 졌을 때 bison 기반의 parser 들을 위한 yydebug 를 설정합니다. 기본적으로 만약 매개변수가 0 이 아니라면 front-end 가 parsr 같은 것을 사용하지 않는다는 경고를 출력합니다.

`tree_inlining`

Tree 의 inlining 과 관련있는 hook 을 가지고 있습니다.

`tree_dump`

Tree node 들의 language-specific 부분들을 dump 합니다.

위에서 지금까지 보았던 것이 struct lang_hooks 을 구성하는 요소들의 설명이 있다. 하지만 아직 좀 더 세부적으로 살펴 봐야 할 부분이 존재하는데 tree_inlining 과 tree_dump 가 그것이다. 이 두가지는

```
struct lang_hooks_for_tree_inlining tree_inlining;
struct lang_hooks_for_tree_dump tree_dump;
```

각각 struct lang_hooks_for_tree_inlining 와 struct lang_hooks_for_tree_dump 구조체로 선언되어 있다. 이에 대해서 세부적으로 살펴 보도록 하자.
우선 lang_hooks_for_tree_inlining 에 대해서 알아 보자.

2.2.1 struct lang_hooks_for_tree_inlining

Tree 의 inlining 을 담당하는 구조체에 대해서 알아 봅시다.

struct lang_hooks_for_tree_inlining 는 $prefix/gcc/langhooks.h 파일에 선언되어 있으며 아래와 같은 구조체 요소들을 갖는데 그림 2 은 구조체의 모습을 보여주고 있다.

이제 이 구조체를 이루는 구성 요소가 어떤 기능을 하는지 한번 알아봅시다.

`*walk_subtrees`

lang_hooks.tree_inlining.walk_subtrees 는 common case 들을 다룬 후에 walk_tree() 에의해 호출되지만 code-specific sub-tree 들을 살펴보기 전에 호출됩니다. 만약 이 hook 이 언어에 의해 설정된다면 이것은 language-specific tree code 들을 다루는 것은 물론 common tree code 들로 구성된 language-specific infomation 도 다루어야 합니다. 만약 tree node 가 이 함수내에서 완전하게 다루어진다면 *SUBTREES 들을 0 으로 설정해야 합니다. 그래야 일반 handling 이 시도되지 않습니다. Language-secific tree code 들을 위해 일반 handling 은 abort() 할것이며 그것이 설정되었음을 확실히 할 것입니다. SUBTREES 와 *SUBTREES 들은 함수가 호출될 때 0 이 아님을 보장받을 것입니다.

```
struct lang_hooks_for_tree_inlining
{
  union tree_node *(*walk_subtrees)
                   PARAMS ((union tree_node **, int *,
                            union tree_node *(*)
                            (union tree_node **,
                            int *, void *),
                            void *, void *));
  int (*cannot_inline_tree_fn) PARAMS ((union tree_node **));
  int (*disregard_inline_limits) PARAMS ((union tree_node *));
  union tree_node *(*add_pending_fn_decls)
                   PARAMS ((void *,
                            union tree_node *));
  int (*tree_chain_matters_p) PARAMS ((union tree_node *));
  int (*auto_var_in_fn_p)
       PARAMS ((union tree_node *, union tree_node *));
  union tree_node *(*copy_res_decl_for_inlining)
                   PARAMS ((union tree_node *,
                            union tree_node *,
                            union tree_node *,
                            void *, int *,
                            void *));
  int (*anon_aggr_type_p) PARAMS ((union tree_node *));
  int (*start_inlining) PARAMS ((union tree_node *));
  void (*end_inlining) PARAMS ((union tree_node *));
  union tree_node *(*convert_parm_for_inlining)
                   PARAMS ((union tree_node *,
                   union tree_node *,
                   union tree_node *));
};
```

그림 2: struct lang_hooks_for_tree_inlining

`*cannot_inline_tree_fn`

lang_hooks.tree_inlining.cannot_inline_tree_fn 는 주어진 함수가 inlining 하지 못하는 language-secific 한 이유들이 존재하는지를 결정하기 위해 호출됩니다.

`*disregard_inline_limits`

lang_hooks.tree_inlining.disregard_inline_limits 는 함수가
inlining limit 를 초과하였더라도 inlining 을 고려해야 할지에 대해 결정하기 위해 호출됩니다.

`*add_pending_fn_decls`

lang_hooks.tree_inlining.add_pending_fn_decls 는 함수를 inline 하기 시작전에 호출되는데, 이러게 하는 이유는 현재 함수에 inline 되어서는 안되는 것이나 VAFNP 내에 inline 되어서는 안되는 어떤 language-specific 함수들을 push 하기 위해서 입니다. PFN 는 varray 의 top 으로써 만약 함수가 VAFNP 내로 push 되지 않는다면 반환되어야 합니다. varray 의 top 은 반드시 반환되어야 합니다. (여기서 VAFNP 는 add_pending_fn_decls 함수의 첫번째 인자의 이름입니다.)

`*tree_chain_matters_p`

lang_hooks.tree_inlining.tree_chain_matters_p 는
language-specific tree node 의 TREE_CHAIN 이 적절한(상응하는) 것인지를 가르킵니다. 예를 들면 이것이 살펴보아야(walk) 할것인지, 복사되어야(copy) 할것인지, 복사에 대해 보존해야(preserved accros copies) 하는지를 말입니다.

`*auto_var_in_fn_p`

lang_hooks.tree_inlining.auto_var_in_fn_p 는 VT 가 함수 FT 내에 정의된 자동 변수인지 결정하기 위해 호출됩니다. (여기서 VT 는 변수 tree 이고, FT 는 함수 tree 이며 각각 *auto_var_in_fn_p 한수의 첫번째, 두번째 인자입니다.)

`*copy_res_decl_for_inlining`

lang_hooks.tree_inlining.copy_res_decl_for_inlining 는 CALLER 내에 inline 되어야 하는 함수 FN 의 결과 RES 를 위한 선언을 반환해야 합니다. NDP 는 새로운 선언이 생성되지 않았을 경우 반드시 설정되어야 하는 정수를 가르킵니다. (생각컨데 TARGET_EXPR 이 결과(result)에 사용되는 것 같은 맥락으로 RES 가 aggregate type 의 것이기 때문인 것 같다). TEXPS 는 함수(fn)가 caller 내로 inlining 되는 동안 볼 수 있는 TARGET_EXPR 들의 stack 을 가지는 varray 를 가르키는 포인터입니다 ; TEXPS 의 top 은 RES 와 같다고 가정합니다. (여기서 사용한 RES, FN, CALLER, NDP, TEXPS 등은 모두 이 함수의 인자들입니다.)

`*anon_aggr_type_p`

lang_hooks.tree_inlining.anon_aggr_type_p 는 T 가 anonymous aggregate (union, struct 등등) 를 나타내는 type node 인지를 결정합니다. 예를 들면 union 으로써 그 자체로 같은 범위(scope) 내에 존재하는 member 들중 하나를 말함.

`*start_inlining`
`*end_inlining`

lang_hooks.tree_inlining.start_inlining 와 end_inlining 는 FN 을 처리하기 위해 어떤 language-specific 의 bookkeping necessary (부기 필요성?) 을 수행합니다. start_inlining 은 inlining 이 처리되어야 한다면 0 이 아닌 값을 반환하며 그렇지 않을 경우 0 을 반환합니다.

예를 들면, C++ 버전은 무한 recursion 을 피하기 위해서 template instatiation 들에 관한 track 을 유지해야 합니다.

`*convert_parm_for_inlining`

lang_hooks.tree_inlining.convert_parm_for_inlining 는
VALUE 를 PARM 으로 assign 하기 전에 어떤 language-secific 변환을 수행합니다. (여기서 VALUE 와 PARM 은 함수의 인자 이름입니다.)

```
struct lang_hooks_for_tree_dump
{
  int (*dump_tree) PARAMS ((void *, tree));

  int (*type_quals) PARAMS ((tree));
};
```

그림 3: struct lang_hooks_for_tree_dump

2.2.2 struct lang_hooks_for_tree_dump

Tree 의 dump 를 담당하는 구조체에 대해서 알아 봅시다.

struct lang_hooks_for_tree_dump 는 $prefix/gcc/langhooks.h 파일에 선언되어 있으며 아래와 같은 구조체 요소들을 갖는데 그림 3 은 구조체의 모습을 보여주고 있다.

이제 이 구조체를 이루는 구성 요소가 어떤 기능을 하는지 한번 알아봅시다. (위의 절에서 Copy & Paste. 귀차니즘 발동 ;))

`*dump_tree`

Tree node 들의 language-specific 부분들을 dump 합니다. 만약 두번째 인자에 대한 dump 하는 것을 원하지 않는다면 0 이 아닌 값을 반환합니다.

`*type_quals`

Language-specific 방식에서의 type qualifier 들을 결정합니다.

2.3 C 언어를 위한 Language hook

앞에서 말했지만 C 언어를 위한 Language hook 은 $prefix/gcc/c-lang.c 에 선언되어 있습니다.

그림 4 는 c-lang.c 파일에 선언되어 있는 실제 모습을 나타냅니다. 보시면 아시겠지만 lang_hooks 전역변수는 LANG_HOOKS_INITIALIZER 로 설정되며 이는 $prefix/gcc/langhooks-def.h 에 정의(define)되어 있습니다. 아래가 그 원형입니다.

```
#define LANG_HOOKS_INITIALIZER { \
  LANG_HOOKS_NAME, \
  LANG_HOOKS_IDENTIFIER_SIZE, \
  LANG_HOOKS_INIT_OPTIONS, \
  LANG_HOOKS_DECODE_OPTION, \
  LANG_HOOKS_POST_OPTIONS, \
  LANG_HOOKS_INIT, \
  LANG_HOOKS_FINISH, \
  LANG_HOOKS_CLEAR_BINDING_STACK, \
  LANG_HOOKS_GET_ALIAS_SET, \
  LANG_HOOKS_EXPAND_CONSTANT, \
  LANG_HOOKS_SAFE_FROM_P, \
  LANG_HOOKS_STATICP, \
  LANG_HOOKS_HONOR_READONLY, \
  LANG_HOOKS_PRINT_STATISTICS, \
  LANG_HOOKS_PRINT_XNODE, \
  LANG_HOOKS_PRINT_DECL, \
  LANG_HOOKS_PRINT_TYPE, \
  LANG_HOOKS_PRINT_IDENTIFIER, \
  LANG_HOOKS_SET_YYDEBUG, \
  LANG_HOOKS_TREE_INLINING_INITIALIZER, \
  LANG_HOOKS_TREE_DUMP_INITIALIZER \
```

```
#undef LANG_HOOKS_NAME
#define LANG_HOOKS_NAME "GNU C"
#undef LANG_HOOKS_INIT
#define LANG_HOOKS_INIT c_init
#undef LANG_HOOKS_FINISH
#define LANG_HOOKS_FINISH c_common_finish
#undef LANG_HOOKS_INIT_OPTIONS
#define LANG_HOOKS_INIT_OPTIONS c_init_options
#undef LANG_HOOKS_DECODE_OPTION
#define LANG_HOOKS_DECODE_OPTION c_decode_option
#undef LANG_HOOKS_POST_OPTIONS
#define LANG_HOOKS_POST_OPTIONS c_post_options
#undef LANG_HOOKS_GET_ALIAS_SET
#define LANG_HOOKS_GET_ALIAS_SET c_common_get_alias_set
#undef LANG_HOOKS_SAFE_FROM_P
#define LANG_HOOKS_SAFE_FROM_P c_safe_from_p
#undef LANG_HOOKS_STATICP
#define LANG_HOOKS_STATICP c_staticp
#undef LANG_HOOKS_PRINT_IDENTIFIER
#define LANG_HOOKS_PRINT_IDENTIFIER c_print_identifier
#undef LANG_HOOKS_SET_YYDEBUG
#define LANG_HOOKS_SET_YYDEBUG c_set_yydebug

#undef LANG_HOOKS_TREE_INLINING_CANNOT_INLINE_TREE_FN
#define LANG_HOOKS_TREE_INLINING_CANNOT_INLINE_TREE_FN \
  c_cannot_inline_tree_fn
#undef LANG_HOOKS_TREE_INLINING_DISREGARD_INLINE_LIMITS
#define LANG_HOOKS_TREE_INLINING_DISREGARD_INLINE_LIMITS \
  c_disregard_inline_limits
#undef LANG_HOOKS_TREE_INLINING_ANON_AGGR_TYPE_P
#define LANG_HOOKS_TREE_INLINING_ANON_AGGR_TYPE_P \
  anon_aggr_type_p
#undef LANG_HOOKS_TREE_INLINING_CONVERT_PARM_FOR_INLINING
#define LANG_HOOKS_TREE_INLINING_CONVERT_PARM_FOR_INLINING \
  c_convert_parm_for_inlining

const struct lang_hooks lang_hooks = LANG_HOOKS_INITIALIZER;
```

그림 4: C 언어의 lang_hooks

```
}
```

위의 모습을 보시면서 C 언어의 경우 이러한 language hook 들을 사용하는 구나 라고 생각하시면 됩니다. 나중에 GCC 의 다른 소스를 보시게 될 경우 lang_hook 관련 부분이 나오시면 이 문서를 참조해 가며 유도해 나가실 수 있을 것입니다.

제 3 절 4 주 강의를 마치며

후하- 드디어 4 주 강의를 마무리 지었습니다. 이번에 한것은 GCC 에서 각 front-end 들에게 제공하는 인터페이스 부분이었는데, 작성하고 나보니 그렇게 간단하게 끝나지가 않는 군요. 그래도 lang_hook 과 관련되는 대부분의 설명은 하지 않았나 생각합니다. 물론 GCC 내부적으로 달려져 있는 주석이 가장 큰 저의 source 이죠. 아마 그것이 없으면 저도 이렇게 문서화를 못 시킬 것입니다. 1987 년부터 지금까지 만들면서 작성한 소스 내부의 문서들도 결코 만만히 볼 수 없는 수준이죠. :) 참 즐겁습니다. 여러분도 저와 같은 행복을 맛보시기 바랍니다.

제가 4 주 강의까지 쓰고 있지만 불행하게도 저에게 질문하는 사람은 아직 한사람도 없군요. 물론 한국에서 컴파일러에 관심있어 하는 사람들은 극소수라는 사실은 알고 있지만 컴파일러를 입문할려고 하는 대학생분들도 많을 듯한데... 오쩜 저에게 질문이 없음을 다행으로 생각해야 할지도 모르죠. '요즘은 OS 를 공부하는 사람은 많아 졌지만 Compiler 를 공부하는 사람들은 옛날이나 지금이나 한국에서 천대받나 봅니다. Embedded 관련으로 시스템 프로그래머의 수요가 늘어나 그나마 시스템 쪽으로만 보았을 때 좋아 보입니다.

질문이 있으신 분들은 아래의 저의 홈페이지를 이용하시거나 저의 E 메일 주소로 보내주시기 바랍니다.

http://weongyo.org
weongyo@hotmail.com