GCC StringPool

정원교

2004년 3월 6일

목 차

제 1 절 19 주 문서를 시작하며

이번 주 문서는 GCC 에서 indentifier 들을 어떻게 관리하는지에 대해서 알아 볼 것이다.

제 2 절 StringPool 이란 무엇인가?

StringPool 은 컴파일러에서 Symbol Table 이라고 생각하면 된다. GCC 에서는 그냥 StringPool 이라고 쓴다. "문자열 집합소?" 정도로 해석될 수 있겠다. 다음 주제에서는 어떻게 이것을 초기화하는지에 대해서 간략히 알아 보도록 하자.

제 3 절 StringPool 의 초기화

실제 Symbol Table 을 초기화하는 부분은 lang_independent_init () 함수 부분에 정의되어 있는 init_stringpool 함수에서 이루어지게 된다. 이 함수는 아주 간단하게 구성되어 있으며, 결론적으로 이야기해서 아래의 두 전역 변수를 세팅하는 것으로 마무리 짓게 된다.

```
struct ht *ident_hash;
static struct obstack string_stack;
```

일단 struct obstack 구조체의 경우 이미 다른 문서에서 다룬 적이 있기 때문에 이 문서에서는 다루지 않겠지만, struct ht 구조체는 잠시 보고 넘어 가도록 하겠다.

제 4 절 struct ht 구조체

이 구조체도 이미 앞의 문서에서 업근한 적이 있다. 06 주 문서 "기반 작업 (1) struct cpp_reader 란" 의 2.14 섹션에서 언급을 하였다. 비록 중복이 되더라도 다시 한번 더 나열해 보고 지나가도록 하겠다.

4.1 구조체 설명

이 구조체는 $prefix/gcc/hashtable.h 에 선언되어 있다. 이것은 Cpplib 와 Front end 들을 위한 식별자 hash table 을 나타내는데, 원형은 아래와 같다.

```
struct ht
{
  struct obstack stack;

  hashnode *entries;
  hashnode (*alloc_node) PARAMS ((hash_table *));

  unsigned int nslots;
  unsigned int nelements;

  struct cpp_reader *pfile;

  unsigned int searches;
  unsigned int collisions;
};
```

각 구성요소의 쓰임새에 대해서 알아보자

- `stack`
  식별자들은 여기서 할당된다.

- `entries`
  아직 정확한 설명이 없다. 추후 UPDATE 하겠다.
- `alloc_node`
  이 구조체와 관련한 Call Back 함수이다.
- `nslots`
  Entry 배열내의 총 slot 수.
- `nelements`
  (살아있는—활동하는) element 들의 수.
- `pfile`
  만약을 위한 reader 에 대한 링크. cpplib 의 효율적 이득을 위해 사용될 수 있다.
- `searches, collisions`
  테이블 사용 통계치에 대해 조사를 할 때 사용된다.

struct cpp_reader 구조체의 경우 이미 앞에서 언급했기 때문에, 그냥 넘어 가도록 하자. 여기에서 볼 수 있는 entries, alloc_node 등등의 구조체 변수들은 hashnode 타입으로 선언되어 있는 것을 알 수 있는데, 이 hashnode 는 아래의 구조체 포인터라는 것을 알기 바란다.

```
struct ht_identifier {
    unsigned int len;
    const unsigned char *str;
}
```

여기서 할 수 있듯이, struct ht 구조체가 어떤 문자열을 가질 수 있도록 하는 요소들을 가지고 있음을 알 수 있을 것이다.

제 5 절 초기화가 끝난 후의 변화

앞에서 init_stringpool () 함수에서는 두 개의 전역 변수가 설정된다고 했으니, 그것에 대해서 이제 알아보자.

5.1 struct ht *ident_hash

init_stringpool () 함수에서 설정이 끝난 후 이 구조체의 변화에 대해서 살펴보도록 하자. 실제 이 구조체를 가지고 있는 전역 변수가 ident_hash 이기 때문에 이 변수의 내용을 살펴봄으로써 알 수 있을 것이다.

ident_hash 구조체의 각 요소에 어떤 값이 들어 있는지를 자세히 살펴보도록 하자. 위에서부터 아래로 언급하도록 하겠다.

```
struct obstack stack;
  {
    long int chunk_size = 4072,
    struct _obstack_chunk *chunk = 0x83ce3e8,
    char *object_base = 0x83ce3f0 "",
    char *next_free = 0x83ce3f0 "",
    char *chunk_limit = 0x83cf3d0 "",
    int temp = 0,
    int alignment_mask = 0,
    struct _obstack_chunk *(*chunkfun)(void *, long int) = 0x827df50 <xmalloc>,
    void (*freefun)(void *, struct _obstack_chunk *) = 0x8049640,
    void *extra_arg = 0x0,
    unsigned int use_extra_arg : 1 = 0,
```

```
    unsigned int maybe_empty_object : 1 = 0,
    unsigned int alloc_failed : 1 = 0
  }
```

위의 내용중 요소 chunk 의 경우, 내용이 모두 아직 초기화되지 않은 값을 가지고 있다.

```
hashnode *entries = (struct ht_identifier **) 0x83cf3d8;
```

이 entries 또한 공간만 활당되어 있고, 아직 내용이 존재하지 않는다.

```
hashnode (*alloc_node) PARAMS ((hash_table *))
  = (struct ht_identifier *(*)(struct ht *)) 0x81d6b80 <alloc_node>
```

alloc_node () 함수 포인터로 설정되어 있다.

```
unsigned int nslots = 16384;
```

entries 내에 16384 개의 slot 이 존재함을 알 수 있으며, 이를 십육진수로 변환시 0x4000 개라는 사실을 알 수 있다.

```
unsigned int nelements = 0;
```

아직 한번도 identifier 를 parsing 한 적이 없기 때문에 element 들의 수는 0 이다.

```
struct cpp_reader *pfile = 0;
```

아직 0x0 으로 설정되어 있다.

```
unsigned int searches = 0;
unsigned int collisions = 0;
```

각각 아직 한번도 찾은 적이 없고, 충돌을 일으킨 적도 없으므로 모두 0 으로 설정되어 있음을 확인할 수 있다.

5.2 struct obstack string_stack

string_stack 구조체는 위에서 언급했듯이, init_stringpool () 함수에서 수행이 된다. _obstack_begin () 함수에서 obstack 이 설정이 된 후에 아래와 같은 구조로 초기화됨을 알수 있다.

```
struct obstack stack;
  {
    long int chunk_size = 4072,
    struct _obstack_chunk *chunk = 0xa05abc0,
    char *object_base = 0xa05abc8 "",
    char *next_free = 0xa05abc8 "",
    char *chunk_limit = 0xa05bba8 "",
    int temp = 0,
    int alignment_mask = 7,
    struct _obstack_chunk *(*chunkfun)(void *, long int) = 0x827df50 <xmalloc>,
    void (*freefun)(void *, struct _obstack_chunk *) = 0x6b4050 <free>,
    void *extra_arg = 0x0,
    unsigned int use_extra_arg : 1 = 0,
    unsigned int maybe_empty_object : 1 = 0,
    unsigned int alloc_failed : 1 = 0
  }
```

제 6 절 어떻게 사용되는가?

위의 섹션까지 StringPool 을 구성하는 두 전역변수가 어떻게 초기화가 되는지에 대해서 알아 보았는데, 이제부터는 실제로 GCC 내에서 어떻게 사용되는지에 대해서 알아보도록 하겠다.

6.1 struct ht *ident_hash

전역변수 ident_hash 는 GCC 에서 해석되는 모든 identifier 들을 가지고 있는 Hash Table 이다. 예약어를 제외한 모든 단어들이 이곳에서 관리되어 진다고 할 수 있다. ident_hash 가 가르키는 상징성은 $prefix/gcc/stringpool.c 에서 관리한다고 할 수 있지만, 실제로 Hash Table 의 개념을 통해서 내부적으로 구조체를 관리하는 것은 $prefix/gcc/hashtable.c 에서 관리한다.

이 전역변수는 큰 저장소의 개념이기 때문에 직접적으로 다루지 않으며, 이것을 관리하는 함수들이 존재하는데, 다른 C source 에서는 아래와 같은 함수들을 호출함으로써 identifier 를 찾거나, 삽입하거나 한다.

```
get_identifier
get_identifier_with_length
maybe_get_identifier
```

다른 섹션에서 String Pool 과 관련된 모든 함수의 기능에 대한 설명을 하겠지만, 앞서 약간 설명하면 위의 세 함수 모두 identifier 들을 찾는데 사용된다. 만약 찾고자 하는 문자열이 Hash Table 에 등록되어 있다면 등록하게 된다.

String Pool 의 개념에서 전역함수 ident_hash 가 어떻게 사용되는지에 대해 알아 보려고 하기 때문에 우리는 위의 세 함수를 누가 사용하는지 알아보면 될 것이다. 왜냐하면 전역변수 ident_hash 에 직접적으로 접근하여 사용하는 다른 함수는 존재하지 않기 때문이다.

전역 변수 ident_hash 의 사용 용도는 아래와 같다.

1. “예약어”들을 저장하기 위해
2. Builtin Function 들 및 Type 들, Attribute 들을 저장하기 위해
3. Optab 의 이름들을 저장하기 위해
4. C source 들을 해석하는 과정에서 새롭게 입수된 identifier 들을 기록하기 위해

위의 세개는 GCC 가 실제 source 들을 해석하기 전, 즉 compile_file () 함수 전에 모두 초기화된다. 예약어의 경우, $prefix/gcc/c-parse.in 에 정의되어 있는 예약어 목록 값들이 등록된게 된다. Builtin 함수, type, attr 들은 $prefix/gcc/builtin-attrs.def 와 $prefix/gcc/builtin-types.def 에 등록된 값들이 들어가게 되며, Optab 이름들은 init_optabs () 함수에 정의된 값들이 들어가게 된다.

이제 실제로 C source 를 해석하는 과정에서 어떻게 identifier 가 등록되는지에 대해서 살펴보도록 하자.

컴파일러에서 실제 token 을 추출해서 DFA (결정적 유한 오토마타) 에 적용하기까지 하나의 token 을 추출하는 것은 c_lex 가 담당한다. 간략하게 요약해서 설명한다면 아래와 같이 설명할 수 있을 것이다.

1. 이제 실제로 하나의 source 를 해석한다고 하자.
2. c-parse.in 에 설정되어 있는 LALR 문법에 의거하여 yyparse_1 가 시작되면서 하나의 token 을 추출하여 그 분류에 따라 순서도가 진행되기 시작한다.
3. 하나의 token 은 c_lex () 함수에 의해 해석되는데, 내부 함수인 _cpp_lex_token () 함수에서 각 token 의 분류에 따라 그 해석 함수들이 호출된다.
4. 만약 identifier 일 경우, parse_identifier 가 해석을 담당한다. 이 함수의 경우, 바로 위에서 말한 세 개의 함수 (즉, get_identifier, ...) 를 이용하지 않고, 전역 변수 parse_in 의 구성 요소 hash_table 를 이용한다. 하지만 이 hash_table 이 위의 전역 변수 ident_hash 와 다른 것이 아닌 같은 것임을 알아야 할 것이다.

```
init_c_lex ();

->  cpp_read_main_file (parse_in, filename, ident_hash)

    ->  _cpp_init_hashtable (pfile, table);

        ->  pfile->hash_table = table;
```

위의 예제는 ident_hash 가 parse_in 의 hash_table 에 들어가는 과정을 요약한 것이다. 위에서 아래로 차례 수행된다.

6.2 struct obstack string_stack

이제 string_stack 에 대해서 알아보도록 하겠다. 이 전역변수를 다루는 곳은 오직 $prefix/gcc/stringpool.c 에 선언되어 있는 ggc_alloc_string () 함수에서만 이다. 또한 이 함수는 $prefix/gcc/ggc.h 에 아래와 같이 선언되어 있어 ggc_strdup 라는 이름으로 사용되기도 한다.

```
#define ggc_strdup(S) ggc_alloc_string((S), -1)
```

전역변수 string_stack 의 용도는 아래와 같다고 할 수 있다.

1. C 언어상에서 정의되는 문자열을 저장하기 위해. 예를 들면 아래와 같은 예제를 말한다.

   ```
   int *a = "test";
   ```

   build_string () 함수를 살펴보면 자세히 알 것이다.

2. Optab 내 entry 들의 libfunc 구성요소를 초기화하기 위해. init_optabs () 함수를 살펴보기 바란다.

3. RTX 의 구성요소 중 문자열을 포함해야 하는 것들을 위해. 예를 들면 SYMBOL_REF 나 ASM_INPUT 에 포함되는 문자열

4. 구조체 struct constant_descriptor 의 구성요소 label 이나, 구조체 struct in_named_entry 의 구성요소 name 을 담기 위해

제 7 절 StringPool 을 위한 함수들

간단하게 나마 String Pool 을 구성하는 함수들에 대해서 알아보도록 하자.

7.1 stringpool.c

void init_stringpool ()

string pool 을 초기화합니다.

static hashnode alloc_node (hash_table **table*);

hash node 를 할당합니다.

const char * ggc_alloc_string (const char **contents*, int *length*);

길이가 LENGTH 이고 CONTENTS 를 포함하는 문자열 constant 를 할당하여 반환합니다. 만약 LENGTH 가 -1 이면 CONTENTS 는 null-terminated 문자열로 가정합니다. 길이는 strlen 를 사용하여 계산됩니다. 만약 같은 문자열 constant 가 이전에 할당되었다면 그 복사본은 이 때 역시 반환됩니다.

tree get_identifier (const char *_text_);

이름이 TEXT (NULL 로 끝나는 문자열) 인 IDENTIFIER_NODE 를 되돌려줍니다. 만약 그 이름을 같은 identifier 가 이전에 참고되었다면 같은 node 를 되돌려줍니다.

tree get_identifier_with_length (const char *_text_, unsigned int _length_);

길이가 정의된다는 것을 제외하고 get_identifier 함수와 동일.

tree maybe_get_identifier (const char *_text_);

만약 이름이 TEXT (null 로 끝나는 문자열) 로 선언된 것이 이전에 있다면 그 node 를 반환하고 그렇지 않으면 NULL_TREE 를 반환합니다.

void stringpool_statistics ()

String pool 에 관한 몇몇 기본적인 통계치를 report 합니다.

static int mark_ident (struct cpp_reader *_pfile_, hashnode _h_, const PTR _v_);

GC 용 하나의 식별자를 mark 합니다.

static void mark_ident_hash (PTR _arg_);

GC 용 모든 식별자를 mark 합니다.

7.2 hashtable.c

void gcc_obstack_init PARAMS ((struct obstack *_obstack_));

Obstack 을 초기화합니다.

static unsigned int calc_hash (const unsigned char *_str_, unsigned int _len_);

길이가 LEN 인 문자열 STR 의 hash 를 계산합니다.

hash_table *ht_create PARAMS ((unsigned int _order_));

식별자 hashtable 을 초기화합니다.

void ht_destroy PARAMS ((hash_table *_table_));

Hash table 과 관련해 할당된 모든 memory 를 free 한다.

hashnode ht_lookup PARAMS ((hash_table *_table_, const unsigned char *_str_, unsigned int _len_, enum ht_lookup_option _insert_));

길이 LEN 의 STR 을 위한 hash entry 를 반환합니다. 만약 그 문자열이 이미 테이블내에 존재한다면 그 entry 를 반환합니다. 만약 INSERT 가 CPP_ALLOCED 라면 마지막 obstack object 를 free 합니다. 만약 식별자가 지금까지 보이지 않은 것이고 INSERT 가 CPP_NO_INSER 라면 NULL 을 반환합니다. 그렇지 않으면 삽입하고 새로운 entry 를 반환합니다. 만약 INSERT 가 CPP_ALLOC 이라면 새로운 문자열이 할당되지만 INSERT 가 CPP_ALLOCED 라면 item 은 obstack 의 꼭대기에 있는 것으로 가정합니다.

static void ht_expand (hash_table *_table_);

Hash table 의 크기를 두배로 늘리고 entry 들을 re-hashing 합니다.

void ht_forall PARAMS ((hash_table *_table_ , ht_cb _cb_, const PTR _v_));

TABLE 내 모든 node 들을 위한, 매개변수 TABLE→PFILE 와 node, V 를 가진 CB 를 callback 한다.

void ht_dump_statistics PARAMS ((hash_table *_table_));

Stderr 로 할당 통계치를 dump 한다.

double approx_sqrt PARAMS ((double _x_));

숫자 N 의 적당한 양의 square root 를 반환한다. 이것은 통계 보고서를 위해서 이고 code 생성을 위해서는 아니다.

제 8 절 19 주 문서를 마치며

이 정도에서 StringPool 에 대한 문서를 마치고, 다음 문서에서는 struct function 에 대해서 알아보도록 하겠습니다.