GCC TREE
..._DECL 과 ..._TYPE 의 layout

정원교

2004년 7월 11일

목 차

제 1 절 27 주 문서를 시작하며

이 문서에서는 TREE 의 ...._DECL 과 ...._TYPE 들의 layout 을 수행하는 과정에 대해서 살펴보자.

제 2 절 Layout

2.1 layout_decl

layout_decl () 함수는 $prefix/gcc/stor-layout.c 파일에 선언되어 있는데, VAR_DECL 와 PARM_DECL, RESULT_DECL, TYPE_DECL, FIELD_DECL 만 처리하는 함수이다. 이 함수의 목적은 ..._DECL node 들의 size 와 mode, alignment 를 설정하기 위해서 이다. LABEL_DECL 와 CONST_DECL node 들에 대해서는 layout 이 필요하지 않으며, FUNCTION_DECL node 들은 이 함수가 아닌 다른 방법으로 설정하게 된다.

보통 size 와 mode 는 변경없이 data type 으로부터 전해지며, alignment 또한 대개 data type 으로부터 전해진다

2.2 layout_type

이 함수는 ...._TYPE 을 위한 mode 와 size, alignment 를 계산하는데, 배열 type 에 대해서 element 분할 또한 계산한다. TYPE 의 계산은 TYPE_SIZE 가 설정되어 있는지 검사를 통해서 이루어지며, 만약 설정되어 있다면, layout_type () 함수는 type 에 대해 아무것도 하지 않는다.

...._TYPE 들 또한 mode 와 size 의 경우, data type 에 따라 그 값이 결정이 되는데, 이러한 값들을 구성하기 위해 사용되는 함수들로는 대부분 smallest_mode_for_size () 함수와 mode_for_size () 함수, bitsize_int (), size_int () 등등이 존재한다. 결과적으로 TYPE 의 크기는 machine mode 와 밀접한 관계를 맺는다. 아래의 함수 설명을 참조하기 바란다.

제 3 절 Layout 함수들

```
void
internal_reference_types ()
```

REFERENCE_TYPES 이 internal 이고 Pmode 여야 함을 보인다. 오직 front end 에서만 호출된다.

```
tree
get_pending_sizes ()
```

pending sizes list 상에 있는 모든 object 들의 list 를 얻는다.

```
int
is_pending_size (expr)
     tree expr;
```

만약 EXPR 이 pending sizes list 상에 존재한다면 0 이 아닌 값을 반환.

```
void
put_pending_size (expr)
     tree expr;
```

EXPR 을 pending sizes list 에 추가합니다.

```
void
put_pending_sizes (chain)
     tree chain;
```

pending sizes list 에 object 들의 chain 을 놓어야 하는데, 이 list 는 반드시 비어있어야 한다.

```
tree
variable_size (size)
     tree size;
```

주어진 size SIZE, 아마 constant 는 아닐 것이다, 는 type 혹은 decl 을 위한 실제 size-expression 으로 제공할 수 있는 SAVE_EXPR 를 반환한다.

```
enum machine_mode
mode_for_size (size, class, limit)
     unsigned int size;
     enum mode_class class;
     int limit;
```

SIZE bit 들의 nonscalar 에 사용할 machine mode 를 반환. 그 mode 는 반드시 class CLASS 내에 속한 것이어야 하며 정확히 얼마의 bit 를 가지는지 기술되어 있어야 한다. 만약 LIMIT 가 0 이 아니라면 MAX_FIXED_MODE_SIZE 보다 넓은 mode 들은 사용되지 않을 것이다.

```
enum machine_mode
mode_for_size_tree (size, class, limit)
     tree size;
     enum mode_class class;
     int limit;
```

비슷하지만 여기선 tree node 가 전달됩니다.

```
enum machine_mode
smallest_mode_for_size (size, class)
     unsigned int size;
     enum mode_class class;
```

비슷하지만, 절대 BLKmode 를 반환하지 않는다; 적어도 요청된 bit 의 수를 포함할 수 있는 가진 근접한 mode 를 반환한다.

```
enum machine_mode
int_mode_for_mode (mode)
     enum machine_mode mode;
```

정확히 같은 크기의 정수 mode 를 찾는다. 실패시 BLKmode 이다.

```
tree
round_up (value, divisor)
     tree value;
     int divisor;
```

DIVISOR 의 배수로 round up 된 VALUE 의 값을 반환한다. 이것은 단지 sizetype 의 object 에만 적용될 수 있다.

```
tree
round_down (value, divisor)
     tree value;
     int divisor;
```

비슷하지만 round down 되었다.

```
void
layout_decl (decl, known_align)
     tree decl;
     unsigned int known_align;
```

....DECL node 의 size 와 mode, alignment 를 설정합니다. TYPE_DECL 는 C++ 때문에 이것이 필요하다. LABEL_DECL 와 CONST_DECL node 들은 이것이 필요하지 않으며, FUNCTION_DECL node 들은 특별한 (혹은 간단한) 방법으로 설정한다. 이것들을 위해 layout_decl 함수를 호출하지 마라.

KNOWN_ALIGN 는 이 decl 이 특별한 노력없이 어느 정도인지 가정할 수 있는 있는 alignment 의 amount 이다. 이것은 오직 FIELD_DECL 들과 관련 있으며 이전 field 들에 의존한다. KNOWN_ALIGN 와 관련된 모든 것은 2 의 제곱으로 나눌 수 있다. 만약 KNOWN_ALIGN 가 0 일 경우, 그것은 "너가 바라는 만큼의 충분한 alignment" 를 의미하며 record 는 알맞게 할당되어질 것이다.

```
void
set_lang_adjust_rli (f)
     void (*f) PARAMS ((record_layout_info));
```

Finalize 되기 전에 record layout 에 대한 수정이 필요할 때 즉각적으로 실행되는 front-end function 로의 hook 을 설정한다.

```
record_layout_info
start_record_layout (t)
     tree t;
```

Type T 에 대한 lay out 을 실시하는데, T 는 RECORD_TYPE 혹은 UNION_TYPE, QUAL_UNION_TYPE 일 것이다. struct record_layout_info 로의 pointer 를 반환하는데, struct record_layout_info 는 이 record 에 대한 모든 다른 layout function 들에게 건네진다. 반환된 storage 에 대해 'free' 를 호출하는 것은 호출자의 책임이다. 우리가 record 에 대한 layout 을 끝나기 전에 garbage collection 를 실시하는 것은 허락되지 않음을 참조하라.

```
tree
bit_from_pos (offset, bitpos)
     tree offset, bitpos;
```

offset/bitpos form 과 byte 와 bit offset 사이의 변환을 실제 수행하는 함수이다.

```
tree
byte_from_pos (offset, bitpos)
     tree offset, bitpos;
```

offset/bitpos form 과 byte 와 bit offset 사이의 변환을 실제 수행하는 함수이다.

```
void
pos_from_byte (poffset, pbitpos, off_align, pos)
     tree *poffset, *pbitpos;
     unsigned int off_align;
     tree pos;
```

offset/bitpos form 과 byte 와 bit offset 사이의 변환을 실제 수행하는 함수이다.

```
void
pos_from_bit (poffset, pbitpos, off_align, pos)
     tree *poffset, *pbitpos;
     unsigned int off_align;
     tree pos;
```

offset/bitpos form 과 byte 와 bit offset 사이의 변환을 실제 수행하는 함수이다.

```
void
normalize_offset (poffset, pbitpos, off_align)
     tree *poffset, *pbitpos;
     unsigned int off_align;
```

주어진 bit 와 byte offset 로의 pointer 와 offset alignment 에 대한 포인터로 offset 들을 normalize 해서 그것이 alignment 내에 존재하도록 한다.

```
void
debug_rli (rli)
     record_layout_info rli;
```

RLI 내 정보에 관한 debugging information 를 출력한다.

```
void
normalize_rli (rli)
     record_layout_info rli;
```

Possibly-incremented BITPOS 를 가진 주어진 RLI 로 OFFSET_ALIGN 아래인 BITPOS 를 유지할 필요성이 있을 경우 OFFSET 과 BITPOS 를 맞춘다.

```
tree
rli_size_unit_so_far (rli)
     record_layout_info rli;
```

지금까지 할당된 크기를 바이트 단위로 반환한다.

```
tree
rli_size_so_far (rli)
     record_layout_info rli;
```

지금까지 할당된 크기를 비트 단위로 반환한다

```
static void
place_union_field (rli, field)
     record_layout_info rli;
     tree field;
```

Union 들을 다루기 위해 place_field 로부터 호출된다.

```
void
place_field (rli, field)
     record_layout_info rli;
     tree field;
```

RLI 은 RECORD_TYPE 의 layout 에 관한 정보를 포함한다. FIELD 는 T 내에 이미 존재하는 해당 field 들 뒤에 추가될 FIELD_DECL 이다. (FIELD 는 실제로 여기 TYPE_FIELDS list 에 추가되지 않는다; 그러한 행동을 요구하는 호출자는 그러한 단계를 수동으로 반드시 수행해야 한다.)

```
static void
finalize_record_size (rli)
     record_layout_info rli;
```

모든 field 들이 lay out 이 완료되었다는 가정하에, 이 함수는 RLI 에 의해 가르켜지는 type 에 대한 마지막 TYPE_SIZE, TYPE_ALIGN 등등을 계산하는데 RLI 를 사용한다.

```
void
compute_record_mode (type)
     tree type;
```

(RECORD_TYPE 인) TYPE 을 위한 TYPE_MODE 를 계산한다.

```
static void
finalize_type_size (type)
     tree type;
```

한번 레이아웃이 끝난 TYPE 을 위해 TYPE_SIZE 와 TYPE_ALIGN 을 계산한다.

```
void
finish_record_layout (rli)
     record_layout_info rli;
```

RLI 에 의해 가르켜지는 type 을 layout 하기 위해 요구되는 모든 작업을 수행한다. 그런 후 field 는 layout 되어질 것이다. 이 함수는 RLI 를 위해 'free' 를 호출할 것이다.

```
void
layout_type (type)
     tree type;
```

TYPE 을 위한 mode 와 size, alignment 를 계산한다. 배열 type 에 대해서 element 분할 또한 계산한다. TYPE 을 영원한 혹은 임시의 type 들의 연결 고리에 기록을 하는데, 이러면 dbxout 이 이것을 찾을 것이다.

type 의 TYPE_SIZE 는 이미 type 이 레이아웃이 되어 있을 경우 0 이 아닌 값이다. 그럴 경우 layout_type 는 type 에 대해 아무것도 하지 않는다.

만약 type 이 아직 완전하지 않다면, 그것의 TYPE_SIZE 는 여전히 0 으로 되어 있다.

```
tree
make_signed_type (precision)
     int precision;
```

PRECISION bit 들의 signed 정수들을 위한 type 을 생성하여 반환한다.

```
tree
make_unsigned_type (precision)
     int precision;
```

PRECISION bit 들의 unsigned 정수들을 위한 type 을 생성하여 반환한다.

```
void
initialize_sizetypes ()
```

생성되어야 하는 정수 type 들을 활성화하기 위해 sizetype 와 bitsizetype 를 어떤 근거있는 임의의 값들로 초기화한다.

```
void
set_sizetype (type)
     tree type;
```

TYPE 에 sizetype 을 설정하고 각자 *sizetype 을 초기화한다. 또한 지금까지 만들어진 어떤 표준 type 들의 type 을 update 한다.

```
void
fixup_signed_type (type)
     tree type;
```

bit 크기의 precision 에 기반하여 TYPE 의 나머지 값들을 설정하고, 그런 후 그것을 레이아웃 한다. Tree code 가 INTEGER_TYPE 이 아니기 때문에 make_signed_type 가 수행될 수 없을 경우 사용된다. 예를 들면, Pascal 에서는 -fsigned-char option 이 주어졌을 때 사용된다.

```
void
fixup_unsigned_type (type)
     tree type;
```

bit 크기의 precision 에 기반하여 TYPE 의 나머지 값들을 설정하고, 그런 후 그것을 레이아웃 한다. 이것은 'make_unsigned_type' 내에서 와 enumeral type 들을 위해 사용된다.

```
enum machine_mode
get_best_mode (bitsize, bitpos, align, largest_mode, volatilep)
     int bitsize, bitpos;
     unsigned int align;
     enum machine_mode largest_mode;
     int volatilep;
```

BITPOS 에서 시작하는 length BITSIZE bits 의 bit field 를 참조할 때 사용할 가장 적합한 machine mode 를 찾는다.

Underlaying object 는 ALIGN bits 의 경계에 align 된것으로 알려진다. 만약 LARGEST_MODE 가 VOIDmode 가 아닐 경우 우리가 LARGEST_MODE (보통 SImode) 보다 더 큰 mode 를 사용하면 안된다는 것을 의미한다.

만약 아무런 mode 가 모든 조건에 맞지 않을 경우, VOIDmode 를 반환한다. 그렇지 않고, VOLATILEP 가 true 이고 SLOW_BYTE_ACCESS 가 false 이면 해당 조건에 맞는 가장 작은 mode 를 반환한다.

그렇지 않을 경우 (VOLATILEP 가 false 이고 SLOW_BYTE_ACCESS 가 true 인), 우리는 모든 조건에 맞는 가장 큰 mode (하지만 UNITS_PER_WORD 보다 큰 mode 는 아닌) 를 반환한다.

```
void
init_stor_layout_once ()
```

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