GCC Trees
변환

정원교

2004년 5월 16일

목 차

제 1 절 24 주 문서를 시작하며

이 문서에서는 GCC 의 TREE 의 data type 을 변환을 위한 부분을 설명하도록 하겠습니다. 이 문서는 상당히 짧은 문서가 될 듯 합니다.

제 2 절 변환

우선 이 문서에서 사용하는 변환의 정의를 들면, TREE node 자체를 다른 것으로 변환한다는 의미가 아니라, TREE node 의 data type 을 변환하는 것을 의미한다. 여기서 data type 은 각 node 의 TREE_TYPE 을 가르키는 것이 일반적이다.

GCC 에서의 변환은 width 의 변경, 즉 integer 들 혹은 real 들의 truncation 과 extension, 은 NOP_EXPR 로 나타낸다. 그리고, integer 와 pointer 사이의 변환은 CONVERT_EXPR 로 나타난다. integer 를 real 로의 변환은 FLOAT_EXPR 를 사용하고 real 에서 integer 로는 FIX_TRUNC_EXPR 를 사용한다.

아래는 NOP_EXPR 를 가지고 widening 과 narrowing 이 항상 이루어진다고 가정하는 모든 함수들의 목록이다.

- convert.c 에서, convert_to_integer
- c-typeck.c 에서, build_binary_op (boolean ops) 와 truthvalue_conversion
- expr.c 에서, MULT_EXPR 의 operand 들을 위한 expand_expr.
- fold-const.c 에서, fold
- tree.c 에서, get_narrower 와 get_unwidened

GCC 에서의 TREE node 변환의 entry point 는 'convert' 함수이며, 이 함수는 $prefix/gcc/c-covert.c 파일에 선언되어 있다. GCC 의 모든 언어의 front end 는 'convert' 함수를 가지고 있어야 한다. 하지만 어떤 종류의 변환을 사용할 것인가는 그 언어에 따라 다를 것이다. 우선 이 convert 함수에 대해서 살펴 보도록 하자.

이 함수의 경우, Type 의 code 에 따라서, 내부적으로 호출하는 함수가 각각 달라지는데, 그 기준이 되는 type 으로는 다음과 같은 것이 있다.

- VOID_TYPE

  build1 () 함수를 사용하여, CONVERT_EXPR 를 생성한다.

- INTEGER_TYPE

  convert_to_integer () 함수를 이용하여 처리한다. 결과값에 대해 constant folding 적용.

- BOOLEAN_TYPE

  truthvalue_conversion () 함수를 통해서 expr 을 처리한 후, build1 함수를 통해 NOP_EXPR node 를 생성한다. 결과값에 대해 constant folding 적용.

- POINTER_TYPE

  convert_to_pointer () 함수를 이용하여 처리한다. 결과값에 대해 constant folding 적용.

- REAL_TYPE

  convert_to_real () 함수를 이용하여 처리한다. 결과값에 대해 constant folding 적용.

- COMPLEX_TYPE

  convert_to_complex () 함수를 이용하여 처리한다. 결과값에 대해 constant folding 적용.

- VECTOR_TYPE

  convert_to_vector () 함수를 이용하여 처리한다. 결과값에 대해 constant folding 적용.

각 code 의 이름이 알려주듯이, 해당 TYPE 으로의 변환시 이를 위한 함수를 각각 호출되게 되며, VOID_TYPE 을 제외한 나머지 모든 부분에서 constant folding 이 이루어진다는 것을 알 수 있다. constant folding 에 대한 문서는 다른 문서에서 후에 언급할 것이다.

위에서 언급한 convert_to_... () 함수들은 모두 $prefix/gcc/convert.c 파일에 선언되어 있는 함수들이다. truthvalue_conversion () 함수의 경우, $prefix/gcc/c-common.c 파일에 선언되어 있다.

그럼 아래의 부분에서는 각 변환 함수들에 대한 세부 내용을 살펴봄으로써 GCC 의 data type 을 구성하는 요소에는 어떤 것이 있는지 살펴보도록 하자.

제 3 절 truthvalue_conversion 함수

이 함수는 위의 convert 에서 BOOLEAN_TYPE 을 처리할 때 호출되는 함수이다. 물론 이곳에서만 호출되는 것이 아니라, 어떤 BOOLEAN 처리와 관련된 거의 모든 곳에서 호출된다. 이 함수는 expr 을 TRUTH_NOT_EXPR 의 argument 와 비교하거나, 'if' 혹은 'while' statement, ? .. : exp 에 대한 그것의 data type 유효성 검사를 하는데 사용된다.

준비는 expression expr 의 원래 representation 을 취하는 것과 expr 가 0 인지 아닌지를 표현하는 유효한 tree boolean expression 를 생성하는 것으로 이루어져 있다.

우리는 항상 단순히 build_binary_op (NE_EXPR, expr, boolean_false_node, 1), 를 실행할 수 있지만, 우리는 comparison 들과 &&, ||, ! 를 최적화 하려고 노력한다. 결과적으로 반환되는 type 은 항상 'boolean_type_node' 어여야 한다.

처리하고 하는 expr 의 TREE_CODE 가

- EQ_EXPR 혹은 NE_EXPR, LE_EXPR, GE_EXPR, LT_EXPR, GT_EXPR, TRUTH_ANDIF_EXPR, TRUTH_ORIF_EXPR, TRUTH_AND_EXPR, TRUTH_OR_EXPR, TRUTH_XOR_EXPR, TRUTH_NOT_EXPR 일 경우, expr 의 TYPE 을 boolean_type_node 로 변경하고 끝낸다.
- INTEGER_CST 일 경우, integer_zerop () 의 여부에 따라, boolean_false_node 혹은 boolean_true_node 가 설정된다.
- REAL_CST 일 경우, real_zerop () 의 여부에 따라, boolean_false_node 혹은 boolean_true_node 가 설정된다.
- ADDR_EXPR 일 경우, expr 이 external decl 의 주소일 경우, 최적화를 못하기 때문에 단순히 build_binary_op (NE_EXPR, expr, boolean_false_node, 1) 를 수행한다. 그렇지 않을 경우, TREE_SIDE_EFFECTS 를 살펴봄으로써, COMPOUND_EXPR node 를 생성해서 반환하거나, boolean_true_node 를 반환하게 된다.
- COMPLEX_EXPR 일 경우, TREE_SIDE_EFFECTS 의 여부에 따라 TRUTH_OR_EXPR 혹은 TRUTH_ORIF_EXPR node 가 생성되어 반환된다.
- NEGATE_EXPR, ABS_EXPR, FLOAT_EXPR, FFS_EXPR 일 경우, 단순히 operand 를 인자로 recursive 를 수행한다.
- LROTATE_EXPR, RROTATE_EXPR 일 경우 TREE_SIDE_EFFECTS 의 여부에 따라 COMPOUND_EXPR node 를 반환하거나, operand 를 인자로 recursive 를 수행한다.
- COND_EXPR 일 경우, COND_EXPR node 를 생성한다
- BIT_AND_EXPR 일 경우, 조건에 따라 NOP_EXPR node 를 수행한다.

그 외의 모든 TREE_CODE 나 위의 TREE_CODE 중 caller 에게 중간에 return 되지 않는 모든 TREE_CODE 들은

```
build_binary_op (NE_EXPR, expr, integer_zero_node, 1);
```

가 수행되어 반환되게 된다.

제 4 절 convert_to_... () 함수들

이 함수는 실제로 해당 expr 의 type 을 우리가 지정한 TYPE 으로 변경하여 그에 대한 tree node 를 반환하는 부분이다. 현재 GCC 에서는 이렇게 수행된 tree node 에 기본적으로 constant folding 이 수행되게 된다. 그럼 아래 부터는 각각의 함수에 대해서 살펴볼 텐데, convert_to_... () 함수를 살펴본 후 이러한 함수가 제대로 수행하기 위해 support 하는 함수들에 대해 살펴보도록 하겠다.

tree convert_to_pointer (tree type, tree expr)

EXPR 를 어떤 pointer 혹은 reference type TYPE 으로 변환한다.

EXPR 는 반드시 pointer 혹은 reference, integer, enumeral, literal zero 여야 하며, 아닐 경우 오류가 호출된다. 아래의 다른 함수들을 살펴보면 알겠지만, 대부분의 convert_to_... () 함수들은 recursive 를 이용한다. 이 함수는 각각에 대해 그에 알맞은, NOP_EXPR, CONVERT_EXPR tree node 를 생성하게 되는데, 이 함수는 어떠한 node 의 type 을 pointer 로 변환하는 것이 목적인데, 만약 같은 POINTER_TYPE 혹은 REFERENCE_TYPE 을 pointer type 으로 변경할 경우, NOP_EXPR node 를 생성하게 되며, 나머지의 경우, 해당 precision 과 pointer type 의 precision 이 맞을 경우, CONVERT_EXPR node 을 생성하게 된다.

tree convert_to_real (tree type, tree expr)

EXPR 을 어느 floating-point type TYPE 로 변환한다.

EXPR 은 반드시 float 혹은 integer, enumeral 여야 한다. 그렇지 않을 경우 오류가 호출된다.

변경하고 하는 expr 가

- REAL_TYPE 일 경우, 전역 변수 flag_float_store 에 따라 CONVERT_EXPR 혹은 NOP_EXPR node 가 생성된다.
- INTEGER_TYPE, ENUMERAL_TYPE, BOOLEAN_TYPE, CHAR_TYPE 일 경우, FLOAT_EXPR tree node 가 생성된다.
- COMPLEX_TYPE 일 경우,

tree convert_to_integer (tree type, tree expr)

EXPR 를 어떤 integer (혹은 enum) type TYPE 으로 변환한다. EXPR 는 pointer 혹은 integer, discrete (enum 혹은 char, bool), float, vector 임이 틀림없을 것이다; 그렇지 않을 경우, 오류가 호출된다. 이것의 결과는 존재하는 어떠한 구조체에서도 사용되지 않는 새로 생성된 tree node 이다는 것이 항상 가정된다

우선 이 함수가 수행되면서 중요한 정보가 될 수 있는 것들을 먼저 추출하게 되는데, 가장 중요한 것은 TYPE 과 EXPR 의 TYPE 이 각각 가지고 있는 TYPE_PRECISION 일 것이다. EXPR 의 TYPE 은 inprec 지역 변수에 저장되고, TYPE 은 outprec 에 저장되게 된다. 그리고 다른 것은 EXPR 의 TREE_CODE 정보 (아래에서 ex_form 으로 표현했다.) 이다. 물론 변환의 기준은 EXPR 의 TYPE 이 무엇이냐에 따라 나누어지지만, 나머지 부분에서의 수행은 TYPE_PRECISION 와 EXPR 의 TREE_CODE 정보에 달라지게 된다. Type 의 변환에서 많은 기준이 되는 것은 해당 input 및 output 의 precision 의 크기 변동일 것이다.

변경하고 하는 expr 가

- POINTER_TYPE, REFERENCE_TYPE 일 경우, CONVERT_EXPR node 를 새로 생성시켜서, 다시 recursive 수행을 한다.

- INTEGER_TYPE, ENUMERAL_TYPE, BOOLEAN_TYPE, CHAR_TYPE 일 경우, TREE_CODE_CLASS (ex_form) 가 '<' 이면 단순히 TREE_TYPE (expr) = type 를 수행한다. ex_form 이 TRUTH_AND_EXPR 혹은 TRUTH_ANDIF_EXPR, TRUTH_OR_EXPR, TRUTH_ORIF_EXPR, TRUTH_XOR_EXPR, TRUTH_NOT_EXPR 일 경우, EXPR 의 operand 들의 type 을 convert 함수를 이용해 변환한 후, 이에 대한 expr 를 반환한다. 만약 output precision 이 input presicion 보다 값이 클 경우, NOP_EXPR node 를 생성한다. 복잡해 지는 부분은 output precision 보다 input presicion 이 클 경우가 해당한다. 이럴 경우, 안전하다고 판단되는 TYPE 을 알아내야 하고 각 ex_form 마다 다른 수행을 해줘야 한다. 물론 처리해 줄 필요가 없는 ex_form 의 경우, NOP_EXPR node 를 생성해 주게 된다. 이러한 수행은 바로 적당한 node 를 만드는 것이 아닌, recursive 호출을 통해 다시 convert () 혹은 covert_to_... () 함수를 호출하게 되는 것이 일반적이다.
- REAL_TYPE 일 경우, FIX_TRUNC_EXPR node 를 생성한다.
- VECTOR_TYPE 일 경우, NOP_EXPR node 를 생성한다.

tree convert_to_complex (tree type, tree expr)

EXPR 를 일반적인 complex type TYPE 로 변환한다.

변경하고 하는 expr 가

- REAL_TYPE, INTEGER_TYPE, ENUMERAL_TYPE, BOOLEAN_TYPE, CHAR_TYPE 일 경우, COMPLEX_EXPR node 를 생성한다.
- COMPLEX_TYPE 일 경우, 조건에 따라 COMPLEX_EXPR node 를 생성하거나, expr 을 그대로 반환한다.

tree convert_to_vector (tree type, tree expr)

EXPR 를 일반적인 vector type TYPE 으로 변환한다.

변경하고 하는 expr 가

- INTEGER_TYPE 혹은 VECTOR_TYPE 일 경우, NOP_EXPR node 를 생성한다.

4.1 Support 함수들

아래에 정의된 함수들은 물론 convert_to_... () 함수들만 위해서 존재하는 것은 아니다. 하지만 이 함수들은 type 의 변환을 위해서 base 로 사용되는 함수이기 때문에, 여기서 언급하도록 한다.

- tree type_for_size (unsigned bits, int unsignedp)

  Precision 이 BITS bit 들로 구성되고, 만약 UNSIGNEDP 가 0 이 아닐 경우 unsigned 인 integer type 을 반환한다. 그렇지 않으면, signed 이다. 실제 파일을 해석하기 전에 GCC 의 초기화에서 미리 만들어둔 TREE node 들의 TYPE_PRECISION 를 비교함으로써 적당한 tree node 를 반환하게 된다.

- tree get_unwidened (tree op, tree for_type)

  안전한 만큼의 wider type 들로의 어떠한 변환들을 잘라낸, OP 를 반환한다. OP 의 type 으로 되돌리는 값의 변환은 값을 OP 와 같게 만든다.

  만약 FOR_TYPE 이 0 이 아니라면, 우리는, type FOR_TYPE 으로 변환되었다면, OP 를 type FOR_TYPE 으로 변환한 것과 같은 값을 반환할 것이다.

  만약 FOR_TYPE 가 0 이 아니라면, unaligned bit-field reference 들은, 그것이 알맞지 않을 수 있지만, 값을 잡고 있을 수 있는 가장 작은 type 으로 변경 될 수 있다. 그렇지 않을 경우, bit-field reference 들은 해당 type 내 memory 로부터 직접적으로 fetch 될 수 있을 경우에만 좀 더 작은 type 으로 변경될 것이다.

OP 는 반드시 정수, 실수, 혹은 enumeral type 를 가져야 한다. 포인터들은 허락하지 않는다.
우리가 반환할 수 있는 분명한 값이 OP 의 type 이 변환될 경우 OP 로 재 생성될 수 있는 몇몇 경우가 존재한다. 하지만 좀 더 넓은 type 으로 OP 를 확장하지는 않는다. 만약 FOR_TYPE 가 그러한 확장에 관해 심사숙고 하였음을 가르킨다면, 우리는 그러한 값들을 의도적으로 삼가한다. 예를 들어, 만약 OP 가 (unsigned short)(signed char)-1 라면, 우리는 만약 FOR_TYPE 이 int 이고, 그것을 unsigned short 로 확장하는 것이 OP 를 재생성하는 것이라면 (signed char)-1 를 반환하는 것을 피한다. 그렇기 때문에, (signed char)-1 를 (int) 로 확장한 결과는 (int) OP 와는 다르다.

- tree unsigned_type (tree type)

  다른 부분으로 TYPE 과 동일한 unsigned type 을 반환한다.

- tree signed_type (tree type)

  다른 부분으로 TYPE 과 동일한 signed type 을 반환한다.

- int integer_zerop (tree expr)

  만약 EXPR 이 정수 상수 0 혹은 복소수 상수 0 일 경우 1 을 반환한다.

- int real_zerop (tree expr)

  만약 EXPR 이 실수 상수 0 이면 1 을 반환한다.

- int tree_int_cst_lt (tree t1, tree t2)

  만약 정수 상수들 T1 과 T2 가 $<$ 를 만족하는 값들을 나타낸다면 0 이 아닌 값. 비교하는데 있어서 접근 방법은 그들의 data type 에 의존한다.

- int tree_int_cst_sgn (tree t)

  정수 상수 T 의 부호에 관해 반환한다. 반환값은 T $<$ 0 이면 -1, T $==$ 0 이면 0, T $>$ 0 이면 1 이다. T 의 type 이 unsign 일 때 절대 -1 이 반환되지 않음을 참고하라.