

adder  
1.0

作成 : Doxygen 1.7.3

Wed Feb 16 2011 22:08:29



# Contents

<b>1</b>	<b>データ構造索引</b>	<b>1</b>
1.1	データ構造	1
<b>2</b>	<b>ファイル索引</b>	<b>3</b>
2.1	ファイル一覧	3
<b>3</b>	<b>データ構造</b>	<b>7</b>
3.1	構造体 abstract_syntax_tree	7
3.1.1	説明	8
3.2	構造体 array_offset	8
3.2.1	説明	8
3.3	構造体 divition_information	9
3.3.1	説明	9
3.4	構造体 for_information	9
3.4.1	説明	10
3.5	構造体 freemem_info	10
3.5.1	説明	10
3.6	構造体 function_information	10
3.6.1	説明	10
3.7	構造体 include_data	11
3.7.1	説明	11
3.8	構造体 memory_allocation_info	11
3.8.1	説明	11
3.9	構造体 param_information	11
3.9.1	説明	12
3.10	構造体 return_info	12
3.10.1	説明	12
3.11	構造体 struct_table	12
3.11.1	説明	13
3.12	構造体 typedef_table	13
3.12.1	説明	13
3.13	構造体 validate_statement	13
3.13.1	説明	14
3.14	構造体 validate_variable	14
3.14.1	説明	15
3.15	構造体 variable_table	15
3.15.1	説明	16
<b>4</b>	<b>ファイル</b>	<b>17</b>

4.1	ANSICInformation/AST.h	17
4.1.1	説明	18
4.1.2	型定義	18
4.1.2.1	AST	18
4.1.3	関数	18
4.1.3.1	deleteAST	18
4.1.3.2	findASTAddress	19
4.1.3.3	fprintDataFromAST	19
4.1.3.4	fprintfStatement	19
4.1.3.5	getArgumentAST	20
4.1.3.6	getArgumentString	20
4.1.3.7	getArgumentStringEnableExcept	21
4.1.3.8	getASTWithString	21
4.1.3.9	getStringFromAST	22
4.1.3.10	getStringFromASTEnableExcept	22
4.1.3.11	getStringReplaceASTtoString	22
4.1.3.12	multi_push_back_childrenAST	23
4.1.3.13	new_AST	23
4.1.3.14	printDataFromAST	23
4.1.3.15	printTargetASTNode	23
4.1.3.16	push_back_childrenAST	24
4.1.3.17	same_new_AST	24
4.1.3.18	setASTBlocklevelAndId	24
4.1.3.19	setASTReturnType	25
4.1.3.20	traverseAST	25
4.1.3.21	traverseASTwithXML	25
4.2	ANSICInformation/DivitionDeclarator.h	26
4.2.1	説明	26
4.2.2	関数	26
4.2.2.1	OutputSourceAfterDivitionDeclarator	26
4.3	ANSICInformation/DivitionInformation.h	27
4.3.1	説明	27
4.3.2	型定義	27
4.3.2.1	DIVITION_INFORMATION	27
4.3.3	関数	28
4.3.3.1	getDIVITION_INFORMATION_LIST	28
4.3.3.2	new_DIVITION_INFORMATION	28
4.3.3.3	new_DIVITION_INFORMATION_char	29
4.3.3.4	printDIVITION_INFORMATION_LIST	29
4.4	ANSICInformation/FreeMemInfo.h	29
4.4.1	説明	30
4.4.2	型定義	30
4.4.2.1	FREEMEMINFO	30
4.4.3	関数	30
4.4.3.1	getFreememInfo	30
4.4.3.2	new_FREEMEMINFO	31
4.4.3.3	printFREEMEMINFO	31
4.5	ANSICInformation/FunctionInformation.h	31
4.5.1	説明	32
4.5.2	型定義	32

4.5.2.1	FUNCTION_INFORMATION	32
4.5.2.2	PARAM_INFORMATION	33
4.5.3	関数	33
4.5.3.1	deleteParameterDefine	33
4.5.3.2	getFunctionInformation	33
4.5.3.3	getFunctionInformationFromFile	33
4.5.3.4	getIN_OUT_FLAG	34
4.5.3.5	getParamInformationFromFunctionDefinition	34
4.5.3.6	getPointerLevelFromFUNCTION_INFORMATION_LIST	34
4.5.3.7	new_FUNCTION_INFORMATION	35
4.5.3.8	new_PARAM_INFORMATION	35
4.5.3.9	printFUNCTION_INFORMATION_LIST	36
4.5.3.10	searchFUNCTION_INFORMATION	36
4.6	ANSICInformation/MemallocInfo.h	36
4.6.1	説明	37
4.6.2	型定義	37
4.6.2.1	MEMALLOC_INFO	37
4.6.3	関数	38
4.6.3.1	getCallocInformation	38
4.6.3.2	getMallocInformation	38
4.6.3.3	getMallocMaxsize	38
4.6.3.4	getReallocInformation	39
4.6.3.5	memoryAllocationAnarysis	39
4.6.3.6	new_MEMALLOC_INFO	39
4.6.3.7	new_MEMALLOC_INFO_char	40
4.6.3.8	searchSizeof	40
4.7	ANSICInformation/PointerArrayControl.h	40
4.7.1	説明	42
4.7.2	型定義	43
4.7.2.1	ARRAY_OFFSET	43
4.7.3	関数	43
4.7.3.1	ARRAY_OFFSET_LIST_push_back_ref_not_dup	43
4.7.3.2	checkCallFunction	43
4.7.3.3	checkIdentifierPointerArrayLevel	44
4.7.3.4	checkIgnoreASTList	44
4.7.3.5	copyArrayOffsetList	45
4.7.3.6	createArrayExpression	45
4.7.3.7	createValidateVariableArrayExpression	45
4.7.3.8	deleteOFFSET_LIST	46
4.7.3.9	deletePointer	46
4.7.3.10	deletePointerAndArraySynbol	46
4.7.3.11	get_ARRAY_OFFSET_LISTIgnoreASTNAME	47
4.7.3.12	getArgumentOffsetInfo	47
4.7.3.13	getARRAY_OFFSET_LIST	48
4.7.3.14	getArrayOffsetInAnpasandInfo	48
4.7.3.15	getArrayOffsetInIncDecInfo	49
4.7.3.16	getDeclaratorArrayOffset	49
4.7.3.17	getExpressionOffsetInfo	50
4.7.3.18	getOFFSET_LISTFromVariableTable	50
4.7.3.19	getOffsetLevelFromArrayOffset	51

4.7.3.20	getPointerAccessOrIdentifierList . . . . .	51
4.7.3.21	getPointerArrayOffset . . . . .	52
4.7.3.22	getSingleExpressionOffsetInfo . . . . .	52
4.7.3.23	getUpperExpressionRelationNode . . . . .	53
4.7.3.24	maxOffsetLevelAddressFromArrayOffsetList . . . . .	54
4.7.3.25	maxOffsetLevelFromArrayOffsetList . . . . .	54
4.7.3.26	minusArrayOffsetList . . . . .	54
4.7.3.27	moveArrayOffsetList . . . . .	55
4.7.3.28	new_ARRAY_OFFSET . . . . .	55
4.7.3.29	new_ARRAY_OFFSET_char . . . . .	56
4.7.3.30	OFFSET_LIST_push_back_alloc . . . . .	56
4.7.3.31	printASTPOINTER_LIST . . . . .	56
4.7.3.32	searchARRAY_OFFSET_LIST . . . . .	57
4.7.3.33	searchExpressionOrPointeArrayOrIden . . . . .	57
4.7.3.34	searchOffsetLevelAddressFromArrayOffsetList . . . . .	58
4.7.3.35	searchPointerAccessOrIdentifier . . . . .	58
4.8	ANSICInformation/PreProcess.h . . . . .	58
4.8.1	説明 . . . . .	59
4.8.2	型定義 . . . . .	59
4.8.2.1	INCLUDE_DATA . . . . .	59
4.8.3	関数 . . . . .	59
4.8.3.1	addIncludeDataFromFile . . . . .	59
4.8.3.2	adjustProgramStart . . . . .	60
4.8.3.3	includeComment . . . . .	60
4.8.3.4	new_INCLUDE_DATA . . . . .	60
4.8.3.5	preProcessor . . . . .	61
4.8.3.6	readIncludeDataFromFile . . . . .	61
4.9	ANSICInformation/Return_Info.h . . . . .	61
4.9.1	説明 . . . . .	62
4.9.2	型定義 . . . . .	62
4.9.2.1	RETURN_INFO . . . . .	62
4.9.3	関数 . . . . .	62
4.9.3.1	new_RETURN_INFO . . . . .	62
4.10	ANSICInformation/SubEffectCheck.h . . . . .	62
4.10.1	説明 . . . . .	63
4.10.2	関数 . . . . .	63
4.10.2.1	checkContainSubEffectStatement . . . . .	63
4.10.2.2	getAssignment_TYPE . . . . .	63
4.11	ANSICInformation/Synbol.h . . . . .	63
4.11.1	説明 . . . . .	65
4.11.2	型定義 . . . . .	65
4.11.2.1	STRUCT_TABLE . . . . .	65
4.11.2.2	TYPDEF_TABLE . . . . .	65
4.11.2.3	VARIABLE_TABLE . . . . .	65
4.11.3	関数 . . . . .	66
4.11.3.1	deletePointer . . . . .	66
4.11.3.2	deletePointerAndArraySynbol . . . . .	66
4.11.3.3	find_STRUCT_TABLE_DATA . . . . .	66
4.11.3.4	getDeclaratorFromAST . . . . .	66
4.11.3.5	getMemberList . . . . .	67

4.11.3.6	getParameterData	67
4.11.3.7	getParameterVARIABLE_TABLE_LIST	68
4.11.3.8	getPointerLevelAndArrayLevel	68
4.11.3.9	getPointerLevelAndArrayLevelFromVARIABLE_TABLE	69
4.11.3.10	getSTRUCT_DATA	69
4.11.3.11	getSTRUCT_TABLE_DATA	70
4.11.3.12	getTYPEDEF_TABLE_DATA	70
4.11.3.13	getTYPEDEFfromAST	70
4.11.3.14	getVARIABLE_TABLE_LIST	71
4.11.3.15	new_STRUCT_TABLE	71
4.11.3.16	new_STRUCT_TABLE_with_char	72
4.11.3.17	new_TYPEDEF_TABLE	72
4.11.3.18	new_TYPEDEF_TABLE_with_char	72
4.11.3.19	new_VARIABLE_TABLE	73
4.11.3.20	new_VARIABLE_TABLE_with_char	73
4.11.3.21	printSTRUCT_TABLE_LIST	74
4.11.3.22	printTYPEDEF_TABLE_LIST	74
4.11.3.23	printVARIABLE_TABLE_LIST	75
4.11.3.24	searchVARIABLE_TABLE_LIST	75
4.12	ANSICInformation/Varidate_statement.h	75
4.12.1	説明	79
4.12.2	型定義	79
4.12.2.1	VALIDATE_STATEMENT	79
4.12.2.2	VALIDATE_VARIABLE	79
4.12.3	関数	79
4.12.3.1	ArrayOffsetToValidateStatement	79
4.12.3.2	createCheckUnboundAndUndefineOperationCheck	80
4.12.3.3	createValidateStatemenFromIncDecExpr	81
4.12.3.4	createValidateStatement	81
4.12.3.5	createValidateStatementAdderFileEachCheck	83
4.12.3.6	createValidateStatementForFreeAction	83
4.12.3.7	createValidateStatementForMallocAction	84
4.12.3.8	createValidateStatementFromArrayDefine	85
4.12.3.9	createVaridateStatementFromPointerDefine	86
4.12.3.10	createViolentFreeOperation	86
4.12.3.11	createZeroDivitionCheck	87
4.12.3.12	fprintProgramDataWithPSIVaridateStatement	88
4.12.3.13	fprintProgramDataWithValidateStatement	89
4.12.3.14	fprintValidateStatement	90
4.12.3.15	fprintValidateStatement_not_assert	91
4.12.3.16	getASTList_FromVALIDATE_STATEMENT_LIST	91
4.12.3.17	getBasisLocationFromAssignmentExpression	91
4.12.3.18	getBasisLocationFromExpression	92
4.12.3.19	getLeftAssignmentInfo	92
4.12.3.20	getNewValidateStatementID	93
4.12.3.21	getRightAssignmentInfo	94
4.12.3.22	getValidate_Variable	94
4.12.3.23	getValidateStatementFromAssignStatement	95
4.12.3.24	getValidateStatementFromCallFunction	96
4.12.3.25	getValidateStatementFromForIteration	96

4.12.3.26	getValidateStatementFromInitializer	97
4.12.3.27	getValidateStatementFromMallocNumber	98
4.12.3.28	getValidateStatementFromPointerOperator	99
4.12.3.29	initVALIDATE_STATEMENT_flag	100
4.12.3.30	new_VALIDATE_STATEMENT	101
4.12.3.31	new_VALIDATE_STATEMENT_char	101
4.12.3.32	new_VALIDATE_VARIABLE	102
4.12.3.33	new_VALIDATE_VARIABLE_with_char	102
4.12.3.34	printProgramDataWithValidateStatement	103
4.12.3.35	printVALIDATE_VARIABLE_LIST	104
4.12.3.36	setValidateVariableFromExprSlicing	104
4.12.3.37	VALIDATE_STATEMENT_LIST_sort_ast	104
4.13	Library/CharStringExtend.h	105
4.13.1	説明	105
4.13.2	関数	105
4.13.2.1	isExpression	105
4.13.2.2	str_extract	106
4.14	Library/CSTLString.h	106
4.14.1	説明	106
4.14.2	関数	107
4.14.2.1	CSTLString_compare_with_char	107
4.14.2.2	CSTLString_delete_tail_str	107
4.14.2.3	CSTLString_ltrim	107
4.14.2.4	CSTLString_printf	107
4.14.2.5	CSTLString_replace_string	108
4.15	Library/FlagDatabase.h	108
4.15.1	説明	108
4.15.2	関数	109
4.15.2.1	getFlagDatabase	109
4.15.2.2	isArrayUnboundCheckMode	109
4.15.2.3	isFreeViolationCheckMode	109
4.15.2.4	isHelpMode	110
4.15.2.5	isProgramSlicingMode	110
4.15.2.6	isUndefineControlCheckMode	110
4.15.2.7	isXmlMode	111
4.15.2.8	isZeroDivitionCheckMode	111
4.16	Library/IdList.h	111
4.16.1	説明	111
4.16.2	関数	112
4.16.2.1	IDLIST_compare_with	112
4.16.2.2	printIDLIST	112
4.16.2.3	SET_STACK_INTToIDLIST	112
4.17	Library/Stack_int.h	113
4.17.1	説明	113
4.17.2	関数	113
4.17.2.1	STACK_INT_at_and_alloc	113
4.17.2.2	STACK_INT_inclement_at	113
4.18	Main/Help.h	114
4.18.1	説明	114
4.18.2	関数	114



4.18.2.1	viewHelp	114
4.19	ProgramSlicing/DeclarationPSI.h	114
4.19.1	説明	115
4.19.2	関数	115
4.19.2.1	getDeclarationtPSI	115
4.20	ProgramSlicing/FunctionPSI.h	115
4.20.1	説明	116
4.20.2	関数	116
4.20.2.1	getFunctionPSI	116
4.20.2.2	getParameterPSI	116
4.21	ProgramSlicing/IfStatementPSI.h	117
4.21.1	説明	117
4.21.2	関数	117
4.21.2.1	getIfStatementPSI	117
4.22	ProgramSlicing/JumpStatementPSI.h	118
4.22.1	説明	119
4.22.2	関数	119
4.22.2.1	getJumpStatementPSI	119
4.23	ProgramSlicing/LabeledStatementPSI.h	119
4.23.1	説明	120
4.23.2	関数	120
4.23.2.1	getLabeledStatementPSI	120
4.24	ProgramSlicing/ProgramSlicing.h	120
4.24.1	説明	121
4.24.2	関数	121
4.24.2.1	createDD_list	121
4.24.2.2	createDD_list_in_argument	122
4.24.2.3	createDD_list_in_Function	122
4.24.2.4	createDD_list_in_global	122
4.24.2.5	createDD_listAll	123
4.24.2.6	createStatementNodeList	123
4.24.2.7	startStaticSlicing	124
4.24.2.8	staticSlicing	124
4.25	ProgramSlicing/ReturnStatementPSI.h	125
4.25.1	説明	125
4.25.2	関数	125
4.25.2.1	getReturnStatementPSI	125
4.26	ProgramSlicing/SwitchStatementPSI.h	126
4.26.1	説明	126
4.26.2	関数	126
4.26.2.1	getSwicthStatementPSI	126



## Chapter 1

# データ構造索引

### 1.1 データ構造

データ構造の説明です。

<a href="#">abstract_syntax_tree</a>	7
<a href="#">array_offset</a>	8
<a href="#">division_information</a>	9
<a href="#">for_information</a>	9
<a href="#">freemem_info</a>	10
<a href="#">function_information</a>	10
<a href="#">include_data</a>	11
<a href="#">memory_allocation_info</a>	11
<a href="#">param_information</a>	11
<a href="#">return_info</a>	12
<a href="#">struct_table</a>	12
<a href="#">typedef_table</a>	13
<a href="#">validate_statement</a>	13
<a href="#">validate_variable</a>	14
<a href="#">variable_table</a>	15



## Chapter 2

# ファイル索引

## 2.1 ファイル一覧

これはファイル一覧です。

ANSICInformation/ <a href="#">ANSIC_CODE.h</a> (このファイルは、ANSIC のソースコードを省略させるためのマクロ文が含まれている) . . . . .	??
ANSICInformation/ <a href="#">AST.h</a> (このファイルは、構文解析によって通された C 言語プログラムから、抽象構文木 (AST) を生成させるためのものです。また生成させるほかに、実際に抽象構文木からソースファイルを生成しなおしたり、ある部分のノードから文字列を生成させたりといったことができます。) . . . . .	17
ANSICInformation/ <a href="#">DivitionDeclarator.h</a> (これは簡潔化のために変数定義を分割させるためのものである 例: int a,b = 1; int a; int b = 1;) . . . . .	26
ANSICInformation/ <a href="#">DivitionInformation.h</a> (これは除算・剰余算を検出するために使用するのに必要な情報を確保する) . . . . .	27
ANSICInformation/ <a href="#">ForInformation.h</a> (これは for ループに関する情報を集めるためのファイルである) . . . . .	??
ANSICInformation/ <a href="#">FreeMemInfo.h</a> (このファイルは、メモリ解放関係の関数から、メモリ解放関係の情報を取得します。具体的には、C 言語プログラム上にある free 関数から、どの変数が解放されているかどうかについて取得します。) . . . . .	29
ANSICInformation/ <a href="#">FunctionInformation.h</a> (このファイルは関数に関する情報を取得するためのファイルである。) . . . . .	31
ANSICInformation/ <a href="#">MallocNumber.h</a> (これはどの malloc 用識別番号を付加する関数を生成するためのものである) . . . . .	??
ANSICInformation/ <a href="#">MemallocInfo.h</a> (このファイルはメモリ確保関係の関数から、メモリ確保関係に関する情報を格納するためのものです。具体的には、C 言語プログラム上にある、malloc・calloc・realloc などといった関数から、確保している sizeof の型・realloc 使用時に対象としている変数・確保しているサイズを取得する。このファイルでできることとしては、::include をコメントアウトすることで省いて、プリプロセスを掛けることができます。) . . . . .	36

ANSICInformation/ <a href="#">PointerArrayControl.h</a> (このファイルは、C 言語プログラム上の複雑な配列参照および直接参照による演算を各次元のオフセットとして格納します。各次元のオフセットとは動的配列や静的配列などにおいて、配列のどの部分を指しているかという式のことです。)	40
ANSICInformation/ <a href="#">PreProcess.h</a> (このファイルは構文解析を通せるように、Gcc で C 言語にプリプロセス (前処理) をさせるものです。このファイルでできることとしては、 <code>::include</code> をコメントアウトすることで省いて、プリプロセスを掛けることができます。)	58
ANSICInformation/ <a href="#">Return_Info.h</a> (これはリターン命令に関する情報を取得するためのものである)	61
ANSICInformation/ <a href="#">SubEffectCheck.h</a> (このファイルには副作用式のチェックする関数や代入式のタイプを求める関数が含まれています)	62
ANSICInformation/ <a href="#">Synbol.h</a> (このファイルは、構文解析によって生成された抽象構文木 (AST) から、変数・typedef テーブル・構造体テーブルを生成させることができます。また、typedef テーブルの生成は、C 言語の構文解析では必須な処理です。)	63
ANSICInformation/ <a href="#">Varidate_statement.h</a> (これは C 言語プログラム上から、不具合を検証するための検証式や検証用を使用する変数などを追加するためのものである)	75
ANSICInformation/ <a href="#">y.tab.h</a>	??
Library/ <a href="#">CharStringExtend.h</a> (このファイルは、 <code>string.h</code> にないような処理を実現させるためのものです。具体的には文字列からある場所からある場所までを抽出させたりといったことができます。)	105
Library/ <a href="#">CSTLString.h</a> (このファイルは、CSTL ライブラリを用いた文字列型 <code>CSTLString</code> を生成したり、それに関する操作を行います。)	106
Library/ <a href="#">FlagDatabase.h</a> (このファイルは、本体に関するフラグ設定を格納するためのものです。たとえば、 <code>xml</code> として出力するのかなどといった設定はこちらに入ります。)	108
Library/ <a href="#">IdList.h</a> (このファイルは、この変数は C 言語プログラム上のどのブロックに宣言しているかどうかを確認するための情報を生成します。)	111
Library/ <a href="#">Stack_int.h</a> (このファイルは整数スタックに関する操作をするためのものです。このファイルで使用される主な用途として、C 言語ソースファイルにおけるブロックレベルを設定するのに使用されます。)	113
Library/ <a href="#">StoreInformation.h</a> (これはファイル名などといったプログラム内で共通する部分を確保するためのものです。主にファイル名などを確保します。)	??
Main/ <a href="#">Help.h</a> (このファイルはヘルプ出力関係のファイルである)	114
ProgramSlicing/ <a href="#">DeclarationPSI.h</a> (このファイルは Declaration Statement Program Slicing Information の略である。宣言式から、プログラムスライシングに関する情報を抽出するための操作が含まれている)	114
ProgramSlicing/ <a href="#">ExpressionStatementPSI.h</a> (このファイルは Expression Statement Program Slicing Information の略である。式全般から、プログラムスライシングに関する情報を抽出するための操作が含まれている)	??

ProgramSlicing/ <a href="#">ForStatementPSI.h</a> (このファイルは For Statement Program Slicing Information の略である。for 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . ??	
ProgramSlicing/ <a href="#">FunctionPSI.h</a> (このファイルは Function Program Slicing Information の略である。関数定義から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	115
ProgramSlicing/ <a href="#">IfStatementPSI.h</a> (このファイルは If Statement Program Slicing Information の略である。if 文もしくは、if と else 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	117
ProgramSlicing/ <a href="#">JumpStatementPSI.h</a> (このファイルは Jump Statement Program Slicing Information の略である。GOTO、continue、break 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	118
ProgramSlicing/ <a href="#">LabeledStatementPSI.h</a> (このファイルは Labeled Statement Program Slicing Information の略である。goto ラベル文、case ラベル文、default ラベル文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	119
ProgramSlicing/ <a href="#">ProgramSlicing.h</a> (このファイルはプログラムスライシングを行うためのものである。指定した識別子名および行数を入れることで、それに基づいてプログラムスライシングを行う) .	120
ProgramSlicing/ <a href="#">ProgramSlicingInformation.h</a> (このファイルはプログラムスライシングに関する情報を追加のみが含まれている) . . . . .	??
ProgramSlicing/ <a href="#">ReturnStatementPSI.h</a> (このファイルは Return Statement Program Slicing Information の略である。return 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	125
ProgramSlicing/ <a href="#">SwitchStatementPSI.h</a> (このファイルは Switch Statement Program Slicing Information の略である。switch 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	126
ProgramSlicing/ <a href="#">WhileStatementPSI.h</a> (このファイルは While Statement Program Slicing Information の略である。while 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている) . . . . .	??





## Chapter 3

# データ構造

### 3.1 構造体 `abstract_syntax_tree`

```
#include <AST.h>
```

#### 変数

- `CSTLString * content`  
ノード名
- `int line`  
ノードの内容
- `CSTLString * return_type`  
対象のノードの行数
- `int block_level`  
返却値のタイプ (*void*、*int*、*double* などが入る)
- `int block_id`  
ブロックの階層レベル (グローバルの場合は0、何らかの関数内であれば1となる)
- `IDLIST * idlist`  
各々のブロックを識別するためのもの。0から順に設定される
- `ASTList * children`  
ブロックの階層および識別子を同時に識別するためのもの。変数定義に関して確認するのに用いる

### 3.1.1 説明

C 言語プログラムの抽象構文木に関する情報が含まれる  
この構造体の説明は次のファイルから生成されました:

- [ANSICInformation/AST.h](#)

## 3.2 構造体 `array_offset`

```
#include <PointerArrayControl.h>
```

### 変数

- [AST \\* target\\_statement](#)  
変数名
- [AST \\* variable\\_address](#)  
ターゲットの *statement*(ここでは *expression\_statement* などが入る)
- [int pointer\\_level](#)  
この変数名が指している AST アドレス
- [int array\\_level](#)  
この変数のポインタレベル
- [int anpasand\\_flag](#)  
この変数の配列レベル
- [int inc\\_dec\\_flag](#)  
この変数はアンパサンドを挟んでいるかどうかのフラグ 1 : 挟んでいる 0 : 挟んでいない
- [OFFSET\\_LIST \\* offset\\_list](#)  
この変数はインクリメントおよびデクリメントが含まれているかどうかのフラグ 0 : 含んでいない 1 : インクリメントが含まれている 2 : デクリメントが含まれている

### 3.2.1 説明

配列やポインタの各次元のオフセット関係を格納するための構造体である。配列オフセットと呼ばれる

この構造体の説明は次のファイルから生成されました:

- [ANSICInformation/PointerArrayControl.h](#)

### 3.3 構造体 `divition_information`

```
#include <DivitionInformation.h>
```

#### 変数

- `int type`  
対象の式
- `CSTLString * statement`  
除算か剰余かどうかのタイプ 0 : 除算式 1 : 剰余式
- `ARRAY_OFFSET_LIST * identifiers`  
除算および剰余以下の式

#### 3.3.1 説明

除算および剰余算に関する情報を格納する

この構造体の説明は次のファイルから生成されました:

- `ANSICInformation/DivitionInformation.h`

### 3.4 構造体 `for_information`

```
#include <ForInformation.h>
```

#### 変数

- `AST * init_expression`  
この `for` 文自体への `AST` ノード
- `AST * continue_condition`  
初期式への `AST` ノード
- `AST * inc_dec_expression`  
継続式への `AST` ノード
- `AST * statement`  
増分式への `AST` ノード

### 3.4.1 説明

for 文の初期式・継続式・増分式および for 文内全体の式へのノードが格納される。これは、検証式付加ツールで for 文を while 文に使用される。

この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[ForInformation.h](#)

## 3.5 構造体 freemem\_info

```
#include <FreeMemInfo.h>
```

### 3.5.1 説明

メモリ確保関係に関係する変数が含まれる。これは、メモリ解放関数の挙動に合わせて検証式を追加するために用いる。

この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[FreeMemInfo.h](#)

## 3.6 構造体 function\_information

```
#include <FunctionInformation.h>
```

### 変数

- CStdString \* [return\\_type](#)  
関数定義の AST ノードへのアドレス
- CStdString \* [function\\_name](#)  
返却値の型 ( *const int* などと表記される )
- PARAM\_INFORMATION\_LIST \* [param\\_information\\_list](#)  
関数名 (*\*func* などと表記される )

### 3.6.1 説明

関数に関する情報。関数からの検証式の生成などに用いる。

この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[FunctionInformation.h](#)

### 3.7 構造体 `include_data`

```
#include <PreProcess.h>
```

#### 変数

- `int line`  
*include* ファイルの名前

#### 3.7.1 説明

インクルードファイルに関する情報が格納される。これは、検証式付加時にインクルードファイルを付加するのに使用する。

この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[PreProcess.h](#)

### 3.8 構造体 `memory_allocation_info`

```
#include <MemallocInfo.h>
```

#### 変数

- `ARRAY_OFFSET_LIST * realloc_target`  
*sizeof* の型名
- `CSTLString * size`  
*realloc* 時のターゲット変数の配列オフセットリスト

#### 3.8.1 説明

メモリ割り当てに関する情報を格納するための構造体

この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[MemallocInfo.h](#)

### 3.9 構造体 `param_information`

```
#include <FunctionInformation.h>
```

## 変数

- `CSTLString * param_name`  
パラメータの型
- `int array_level`  
パラメータの名前
- `int pointer_level`  
配列のレベル
- `int in_out_flag`  
ポインタのレベル

### 3.9.1 説明

引数に関する情報

この構造体の説明は次のファイルから生成されました:

- `ANSICInformation/FunctionInformation.h`

## 3.10 構造体 `return_info`

```
#include <Return_Info.h>
```

## 変数

- `ARRAY_OFFSET_LIST * return_array_offset_list`  
リターン命令自体へのノードへのアドレス

### 3.10.1 説明

リターン命令に関する情報

この構造体の説明は次のファイルから生成されました:

- `ANSICInformation/Return_Info.h`

## 3.11 構造体 `struct_table`

```
#include <Synbol.h>
```

## 変数

- CStdString \* [type](#)  
行数
- CStdString \* [struct\\_name](#)  
構造体のタイプ *union*: 共同体 *struct*: 構造体
- VARIABLE\_TABLE\_LIST \* [member\\_list](#)  
構造体の名前

### 3.11.1 説明

構造体に関する情報であり、プログラム中の構造体の識別するのに用いられる。  
この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[Symbol.h](#)

## 3.12 構造体 typedef\_table

```
#include <Symbol.h>
```

## 変数

- CStdString \* [change\\_type](#)  
対象の型定義

### 3.12.1 説明

型定義に関する情報で、BISON による構文解析時の型定義の認識に用いられる。  
この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[Symbol.h](#)

## 3.13 構造体 validate\_statement

```
#include <Varidate_statement.h>
```

## 変数

- int [check\\_or\\_modify](#)  
この検証式の識別 ID(どの順序でこの検証式を入れていくかを確認するための ID)
- int [used](#)  
検証式をチェックするタイプか、プログラムを元に編集するタイプかを判断するフラグ。0 : チェックするタイプ、1 : 編集するタイプ
- CString \* [statement](#)  
この検証式は使用しているかどうかのフラグ 1:使用 0:未使用
- AST \* [target\\_statement](#)  
この検証式の内容

### 3.13.1 説明

実際に検証式として挿入するための情報

この構造体の説明は次のファイルから生成されました:

- ANSICInformation/[Varidate\\_statement.h](#)

## 3.14 構造体 `validate_variable`

```
#include <Varidate_statement.h>
```

## 変数

- int [enable\\_start](#)  
この検証用の変数が使用しているかどうかのフラグ 1:使用 0:未使用
- int [enable\\_end](#)  
この変数の有効範囲 (ブロックでの) の開始行数を示す
- int [declaration\\_location](#)  
この変数の有効範囲 (ブロックでの) の終了行数を示す
- int [block\\_level](#)  
この変数を宣言する行数を示す
- int [block\\_id](#)  
この変数を宣言するブロックレベルを示す (0 ならグローバル変数・1 なら関数内と示す)



- `CSTLString * type`  
この変数を宣言するブロック *ID* を示す (同ブロックレベルで別のブロックを識別するための番号)
- `CSTLString * variable_name`  
この変数の型
- `CSTLString * target_variable_name`  
この変数名
- `int offset_level`  
この変数の検証対象となる変数名

#### 3.14.1 説明

ポインタや配列変数に対する検証用の変数リストを作成するための構造体  
この構造体の説明は次のファイルから生成されました:

- `ANSICInformation/Varidate_statement.h`

### 3.15 構造体 `variable_table`

```
#include <Symbol.h>
```

#### 変数

- `int enable_end`  
この変数が有効である開始の行数
- `AST * declaration_location_address`  
この変数が有効である終了の行数
- `int block_level`  
この変数の宣言をしている *AST* ノードへのアドレス
- `int block_id`  
この変数が宣言しているブロックレベル (グローバル変数なら 0、関数内のローカル変数なら 1 となる)
- `IDLIST * idlist`  
この変数が宣言している各々のブロックの識別子

- CStdString \* [type](#)  
この変数が宣言しているブロックレベルおよび識別子の両方に関する情報で、変数スコープを識別するのに使用される。
- CStdString \* [variable\\_name](#)  
この変数の型
- [AST](#) \* [initiarizer](#)  
この変数の名前

### 3.15.1 説明

プログラム中の変数に関する情報であり、検証式生成時に変数を識別するのに用いられる。

この構造体の説明は次のファイルから生成されました:

- [ANSICInformation/Synbol.h](#)

## Chapter 4

# ファイル

### 4.1 ANSICInformation/AST.h

このファイルは、構文解析によって通された C 言語プログラムから、抽象構文木 (AST) を生成させるためのものです。また生成させるほかに、実際に抽象構文木からソースファイルを生成しなおしたり、ある部分のノードから文字列を生成させたりといったことができます。

```
#include <cstl/string.h>
#include <cstl/list.h>
#include <stdio.h>
#include "../Library/CSTLString.h"
#include "../Library/IdList.h"
```

#### データ構造

- struct [abstract\\_syntax\\_tree](#)

#### 型定義

- typedef struct [abstract\\_syntax\\_tree](#) AST

#### 関数

- [AST \\* new\\_AST](#) (char \*new\_name, char \*new\_content, int new\_line)
- [AST \\* same\\_new\\_AST](#) (char \*new\_name, int new\_line)
- void [traverseAST](#) (AST \*root, int tab\_level)
- void [traverseASTwithXML](#) (AST \*root, int tab\_level)
- void [fprintDataFromAST](#) (FILE \*output, AST \*root, int \*line)
- void [printDataFromAST](#) (AST \*root, int \*line)

- void `getStringFromAST` (CSTLString \*output, AST \*root)
- void `getStringFromASTEnableExcept` (CSTLString \*output, AST \*root, int num, char except\_list[ ][256])
- void `deleteAST` (AST \*root)
- void `push_back_childrenAST` (AST \*parent, AST \*child)
- void `multi_push_back_childrenAST` (AST \*parent, int num,...)
- void `printTargetASTNode` (AST \*root, char \*target, int traverse\_flag, int xml\_flag)
- int `getArgumentString` (CSTLString \*output, AST \*call\_function, int arg\_num)
- int `getArgumentStringEnableExcept` (CSTLString \*output, AST \*call\_function, int arg\_num, int num, char except\_list[ ][256])
- int `getArgumentAST` (AST \*\*output, AST \*call\_function, int arg\_num)
- AST \* `getASTwithString` (AST \*root, char \*name, int depth)
- void `getStringReplaceASTtoString` (CSTLString \*output, AST \*root, AST \*target, char \*replace\_string)
- void `setASTReturnType` (AST \*root)
- void `setASTBlocklevelAndId` (AST \*root)
- int `findASTAddress` (AST \*root, AST \*target)
- void `fprintfStatement` (FILE \*output, AST \*root, int \*line, int output\_subnode\_num, AST \*\*out\_block\_start, AST \*\*out\_block\_end)

#### 4.1.1 説明

このファイルは、構文解析によって通された C 言語プログラムから、抽象構文木 (AST) を生成させるためのものです。また生成させるほかに、実際に抽象構文木からソースファイルを生成しなおしたり、ある部分のノードから文字列を生成させたりといったことができます。

作者

faithnh

#### 4.1.2 型定義

##### 4.1.2.1 typedef struct abstract\_syntax\_tree AST

C 言語プログラムの抽象構文木に関する情報が含まれる

#### 4.1.3 関数

##### 4.1.3.1 void deleteAST ( AST \* root )

指定した AST ノード以下を全て開放させる

引数

<i>output</i>	出力対象の文字列データ
---------------	-------------

戻り値

なし

#### 4.1.3.2 int findASTAddress ( AST \* root, AST \* target )

探す対象の AST ノード root から、指定した AST ノードのアドレス target が存在するかどうか調べる。みつければ、1 を返し、そうでなければ 0 を返す。

引数

<i>root</i>	探す対象の AST ノード
<i>target</i>	指定した AST ノードのアドレス

戻り値

指定した AST ノード target を見つけたら 1 を返し、そうでなければ 0 を返す。

#### 4.1.3.3 void fprintfDataFromAST ( FILE \* output, AST \* root, int \* line )

指定された AST ノード以下の内容を、指定されたファイルに対して出力させる

引数

<i>output</i>	出力先のファイル構造体
<i>root</i>	指定されたノード名
<i>line</i>	出力に利用する行数

戻り値

なし

#### 4.1.3.4 void fprintfStatement ( FILE \* output, AST \* root, int \* line, int output\_subnode\_num, AST \*\* out\_block\_start, AST \*\* out\_block\_end )

指定されたノードに対して、statement の直前まで出力させる。また、出力対象となっているノード番号が来るまでは出力の対象としない。ブロックとして表記されている場合は始まりのブロックや終わりのブロックへの AST ノードのアドレスを out\_block\_start や out\_block\_end に設定する。そうでなければ、NULL で設定される。

引数

<i>output</i>	出力先のファイル構造体
<i>root</i>	指定された AST ノード
<i>line</i>	出力に利用する行数

<i>output_subnode_num</i>	出力対象となるノード番号 1 ~ n ( 1 に指定すると、すべての子ノードが出力の対象となる )
<i>out_block_start</i>	ブロックとして表記されている場合は、始まりのブロックへの AST ノードのアドレスを返却する
<i>out_block_end</i>	ブロックとして表記されている場合は、終わりブのロックへの AST ノードのアドレスを返却する。

戻り値

なし

#### 4.1.3.5 int getArgumentsAST ( AST \*\* output, AST \* call\_function, int arg\_num )

指定された call\_function に相当する関数から、任意の引数目のへのノードを取得する。このとき、指定されたノード名は call\_function でなければならない

引数

<i>output</i>	取得する対象のノード
<i>call_function</i>	名前が call_function である AST ノード
<i>arg_num</i>	何番目の引数

戻り値

成功したかどうかを示す。成功したならば 1、失敗した場合は 0 を返す

#### 4.1.3.6 int getArgumentsString ( CString \* output, AST \* call\_function, int arg\_num )

指定された call\_function に相当する関数から、任意の引数目の情報を取得する。このとき、指定されたノード名は call\_function でなければならない

引数

<i>output</i>	出力される引数の内容
<i>call_function</i>	名前が call_function である AST ノード
<i>arg_num</i>	何番目の引数

戻り値

成功したかどうかを示す。成功したならば 1、失敗した場合は 0 を返す

**4.1.3.7** `int getArgumentsStringEnableExcept ( CSTLString * output, AST * call_function, int arg_num, int num, char except_list[][256] )`

`getArgumentEnableExcept` の拡張版。出力対象にしない文字列を除いた引数の情報をすべて出力させる

引数

<i>output</i>	出力される引数の内容
<i>call_function</i>	名前が <code>call_function</code> である AST ノード
<i>arg_num</i>	何番目の引数
<i>num</i>	除外したい AST ノードリストの数
<i>except_list</i>	除外したい AST ノード

戻り値

成功したかどうかを示す。成功したならば 1、失敗した場合は 0 を返す

`getArgumentString` の拡張版。出力対象にしない文字列を除いた引数の情報をすべて出力させる

引数

<i>output</i>	出力される引数の内容
<i>call_function</i>	名前が <code>call_function</code> である AST ノード
<i>arg_num</i>	何番目の引数
<i>num</i>	除外したい AST ノードリストの数
<i>except_list</i>	除外したい AST ノード

戻り値

成功したかどうかを示す。成功したならば 1、失敗した場合は 0 を返す

**4.1.3.8** `AST* getASTwithString ( AST * root, char * name, int depth )`

指定した AST ノード `root` から、指定した名前 `name` の AST ノードを探す。見つけた AST ノードへのアドレスを返す。

引数

<i>root</i>	指定した AST ノード
<i>name</i>	指定した名前
<i>depth</i>	探索する子ノードの深さ（無限にする場合は -1 を指定、子ノードは探索しない場合は 0 にする）

戻り値

見つけた AST ノードへのアドレス

#### 4.1.3.9 void getStringFromAST ( CSTLString \* output, AST \* root )

指定した AST ノード以下の内容を出力対象の文字列データに出力させる。

引数

<i>output</i>	出力対象の文字列データ
<i>root</i>	新しいノード名

戻り値

なし

#### 4.1.3.10 void getStringFromASTEnableExcept ( CSTLString \* output, AST \* root, int num, char except\_list[][256] )

getStringFromAST の拡張版。出力対象から除外した AST ノード名を指定できる。

引数

<i>output</i>	出力対象の文字列データ
<i>root</i>	新しいノード名
<i>num</i>	除外したい AST ノードリストの数
<i>except_list</i>	除外したい AST ノード

戻り値

なし

#### 4.1.3.11 void getStringReplaceASTToString ( CSTLString \* output, AST \* root, AST \* target, char \* replace\_string )

指定した AST ノード root に対しての情報を文字列 output を出力させる。そのとき、AST ノードの target のアドレス値と一致するノードを見つけた（アドレスとして全く同じ AST ノード）場合、そのノードだけ replace\_string に変換して出力させる。

引数

<i>root</i>	指定した AST ノード
<i>output</i>	出力する情報
<i>target</i>	変換対象の AST ノードのアドレス
<i>replace_string</i>	変換先の文字列

戻り値

なし



**4.1.3.12 void multi\_push\_back\_childrenAST ( AST \* parent, int num, ... )**

指定された親 AST ノードに任意の子 AST ノードを追加する

引数

<i>parent</i>	指定された親 AST ノード
<i>num</i>	追加する子 AST ノードの数
<i>...</i>	追加する任意の子 AST ノード

戻り値

なし

**4.1.3.13 AST\* new\_AST ( char \* new\_name, char \* new\_content, int new\_line )**

新しい AST ノードを生成する

引数

<i>newname</i>	新しいノード名
<i>new_content</i>	新しい内容
<i>new_line</i>	新しい行数

戻り値

新しく生成された AST ノードへのアドレスが返される

**4.1.3.14 void printDataFromAST ( AST \* root, int \* line )**

指定した AST ノード以下の内容を出力させる。AST ノードの行数にしたがって出力される。

引数

<i>root</i>	新しいノード名
<i>line</i>	指定する行数 (基本的に 1 を入力する )

戻り値

なし

**4.1.3.15 void printTargetASTNode ( AST \* root, char \* target, int traverse\_flag, int xml\_flag )**

指定された親 AST ノードから、指定したノード名より下位についての情報をすべて表示させる また、traverse\_flag を 1 にすることで、指定したノード名についてのすべての情報をツリー構造で表示させることもできる。さらに、xml\_flag を

1 にすることで、ツリー構造で表示させる内容が XML として出力できるようになる (traverse\_flag が 1 になっていることが前提である)。

引数

<i>parent</i>	指定された親 AST ノード
<i>target</i>	指定した文字列
<i>traverse_flag</i>	指定したノード名以下をツリー構造で表示させるかどうかのフラグ。1 なら表示させ、0 なら表示させない
<i>xml_flag</i>	XML として出力するかどうかのフラグ。1 なら XML として出力させる

戻り値

なし

#### 4.1.3.16 void push\_back\_childrenAST ( AST \* parent, AST \* child )

指定された親 AST ノードに指定された子 AST ノードを追加する

引数

<i>parent</i>	指定された親 AST ノード
<i>child</i>	指定された子 AST ノード

戻り値

なし

#### 4.1.3.17 AST\* same\_new\_AST ( char \* new\_name, int new\_line )

名前と内容が同じである AST ノードを生成する

引数

<i>newname</i>	新しいノード名
<i>new_line</i>	新しい行数

戻り値

新しく生成された AST ノードへのアドレスが返される

#### 4.1.3.18 void setASTBlocklevelAndId ( AST \* root )

対象の AST ノードに、ブロック ID およびブロックレベルを付加する

引数

<i>root</i>	対象の AST ノード
-------------	-------------

戻り値

なし

**4.1.3.19 void setASTReturnType ( AST \* root )**

対象の AST ノードの root 以下にあるに、関数の返却値のタイプを指定する。

引数

<i>root</i>	対象の AST ノード
-------------	-------------

戻り値

なし

**4.1.3.20 void traverseAST ( AST \* root, int tab\_level )**

AST のノードをたどり、root ノード以下のノードの名前・内容・行数を表示させる。また、下位レベルのノードはタブを挿入し表示する。例えば、1 レベル下位のノードはタブが 1 つ挿入した上で表示される。

引数

<i>root</i>	新しいノード名
<i>tab_level</i>	タブレベル(この数分タブが挿入される)

戻り値

なし

**4.1.3.21 void traverseASTwithXML ( AST \* root, int tab\_level )**

AST のノードをたどり、root ノード以下のノードの名前・内容・行数を XML 形式で出力する。(デバッグ用) また、下位レベルのノードはタブを挿入し出力する。例えば、1 レベル下位のノードはタブが 1 つ挿入した上でタグが出力される。XML については次の形式で出力する (a はルートノード、a\_sub はリーフノードであるとする) <leaf name="a\_sub" content="+" line="1">

引数

<i>root</i>	新しいノード名
<i>tab_level</i>	タブレベル(この数分タブが挿入される)

戻り値

なし

## 4.2 ANSICInformation/DivitionDeclarator.h

これは簡潔化のために変数定義を分割させるためのものである 例：int a,b = 1;  
int a; int b = 1;

```
#include "AST.h"
```

```
#include "Synbol.h"
```

関数

- void [OutputSourceAfterDivitionDeclarator](#) (FILE \*output, [AST](#) \*programAST, VARIABLE\_TABLE\_LIST \*variable\_table\_list)

### 4.2.1 説明

これは簡潔化のために変数定義を分割させるためのものである 例：int a,b = 1;  
int a; int b = 1;

作者

faithnh

### 4.2.2 関数

4.2.2.1 void [OutputSourceAfterDivitionDeclarator](#) ( FILE \* *output*, [AST](#) \* *programAST*,  
VARIABLE\_TABLE\_LIST \* *variable\_table\_list* )

変数定義を分割させた状態で出力させる 例：int a,b = 1;    int a; int b = 1;

引数

<i>output</i>	出力先のファイル構造体
<i>programAST</i>	プログラムに対する AST
<i>variable_ - table_list</i>	変数テーブル

戻り値

なし

## 4.3 ANSICInformation/DivitionInformation.h

これは除算・剰余算を検出するために使用するのに必要な情報を確保する

```
#include <cstl/list.h>
#include "AST.h"
#include "Synbol.h"
#include "../Library/CSTLString.h"
#include "PointerArrayControl.h"
#include "FunctionInformation.h"
```

### データ構造

- struct [divition\\_information](#)

### 型定義

- typedef struct [divition\\_information](#) [DIVITION\\_INFORMATION](#)

### 関数

- [DIVITION\\_INFORMATION](#) \* [new\\_DIVITION\\_INFORMATION](#) ([AST](#) \*target\_expression, int type, CSTLString \*statement, ARRAY\_OFFSET\_LIST \*identifiers)
- [DIVITION\\_INFORMATION](#) \* [new\\_DIVITION\\_INFORMATION\\_char](#) ([AST](#) \*target\_expression, int type, char \*statement, ARRAY\_OFFSET\_LIST \*identifiers)
- void [getDIVITION\\_INFORMATION\\_LIST](#) ([AST](#) \*expression\_ast, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, DIVITION\_INFORMATION\_LIST \*divition\_information\_list, [AST](#) \*target\_expression, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list)
- void [printDIVITION\\_INFORMATION\\_LIST](#) ([DIVITION\\_INFORMATION\\_LIST](#) \*divition\_information\_list)

#### 4.3.1 説明

これは除算・剰余算を検出するために使用するのに必要な情報を確保する

作者

faithnh

#### 4.3.2 型定義

##### 4.3.2.1 typedef struct [divition\\_information](#) [DIVITION\\_INFORMATION](#)

除算および剰余算に関する情報を格納する

### 4.3.3 関数

**4.3.3.1** void getDIVISION\_INFORMATION\_LIST ( AST \* *expression\_ast*,  
FUNCTION\_INFORMATION\_LIST \* *function\_information\_list*, VARIABLE\_TABLE\_LIST  
\* *vtlist*, DIVISION\_INFORMATION\_LIST \* *division\_information\_list*, AST \*  
*target\_expression*, ASTPOINTER\_LIST \* *ignore\_ast\_list* )

指定された式から、div\_expr および mod\_expr を見つけ、それ以下の式を除算および剰余残に関する情報に格納する

#### 引数

<i>expression_ast</i>	指定された式への AST ノードのアドレス
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	変数テーブルリスト
<i>division_information_list</i>	除算および剰余算に関する情報を格納するためのリスト
<i>target_expression</i>	対象の式への AST ノードのアドレス
<i>ignore_ast_list</i>	重複防止のために使用する AST ノードへのリスト

#### 戻り値

なし

**4.3.3.2** DIVISION\_INFORMATION\* new\_DIVISION\_INFORMATION ( AST \*  
*target\_expression*, int *type*, CSTLString \* *statement*, ARRAY\_OFFSET\_LIST \* *identifiers*  
)

DIVISION\_INFORMATION のリスト 除算および剰余算に関する情報を生成する

#### 引数

<i>target_expression</i>	対象の式への AST ノードのアドレス
<i>type</i>	除算か剰余かどうかのタイプ 0 : 除算式 1 : 剰余式
<i>statement</i>	除算および剰余以下の式
<i>identifiers</i>	式内の識別子一覧

#### 戻り値

生成された情報へのアドレスが返される

**4.3.3.3 DIVITION\_INFORMATION**\* new\_DIVITION\_INFORMATION\_char ( AST \*  
target\_expression, int type, char \* statement, ARRAY\_OFFSET\_LIST \* identifiers )

除算および剰余算に関する情報を生成する

引数

target_ expression	対象の式への AST ノードのアドレス
type	除算か剰余かどうかのタイプ 0 : 除算式 1 : 剰余式
statement	除算および剰余以下の式
identifiers	式内の識別子一覧

戻り値

生成された情報へのアドレスが返される

**4.3.3.4 void printDIVITION\_INFORMATION\_LIST** ( DIVITION\_INFORMATION\_LIST \*  
divition\_information\_list )

除算および剰余算に関する情報のリストの内容を出力させる

引数

divition_ information_ list	除算および剰余算に関する情報のリスト
-----------------------------------	--------------------

戻り値

なし

## 4.4 ANSICInformation/FreeMemInfo.h

このファイルは、メモリ解放関係の関数から、メモリ解放関係の情報を取得します。具体的には、C 言語プログラム上にある free 関数から、どの変数が解放されているかどうかについて取得します。

```
#include "PointerArrayControl.h"
#include "FunctionInformation.h"
```

### データ構造

- struct [freemem\\_info](#)

## 型定義

- typedef struct [freemem\\_info](#) FREEMEMINFO

## 関数

- [FREEMEMINFO](#) \* [new\\_FREEMEMINFO](#) (ARRAY\_OFFSET\_LIST \*free\_variable)
- void [getFreememInfo](#) ([FREEMEMINFO](#) \*\*freememinfo, [AST](#) \*call\_function, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, [AST](#) \*target\_statement)
- void [printFREEMEMINFO](#) ([FREEMEMINFO](#) \*freememinfo)

### 4.4.1 説明

このファイルは、メモリ解放関係の関数から、メモリ解放関係の情報を取得します。具体的には、C 言語プログラム上にある free 関数から、どの変数が解放されているかどうかについて取得します。

作者

faithnh

### 4.4.2 型定義

#### 4.4.2.1 typedef struct freemem\_info FREEMEMINFO

メモリ確保関係に関係する変数が含まれる。これは、メモリ解放関数の挙動に合わせて検証式を追加するために用いる。

### 4.4.3 関数

#### 4.4.3.1 void getFreememInfo ( FREEMEMINFO \*\* freememinfo, AST \* call\_function, FUNCTION\_INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, AST \* target\_statement )

対象の関数への AST ノードから free 関数を見つけ、それに関する情報を取得する。

引数

<i>freememinfo</i>	取得先のメモリ解放関係の情報
<i>call_function</i>	対象の関数への AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	変数テーブルの情報



<i>target_statement</i>	対象の関数が位置している式への AST ノード
-------------------------	-------------------------

戻り値

なし

#### 4.4.3.2 FREEMEMINFO\* new\_FREEMEMINFO ( ARRAY\_OFFSET\_LIST \* *free\_variable* )

新しいメモリ解放関係の情報を生成する

引数

<i>freememinfo</i>	新しい解放先の変数に関するオフセットリスト
--------------------	-----------------------

戻り値

新しく生成されたメモリ解放関係の情報へのアドレスを返す。

#### 4.4.3.3 void printFREEMEMINFO ( FREEMEMINFO \* *freememinfo* )

指定したメモリ解放関係の情報を出力する

引数

<i>freememinfo</i>	出力するメモリ解放関係の情報
--------------------	----------------

戻り値

なし

## 4.5 ANSICInformation/FunctionInformation.h

このファイルは関数に関する情報を取得するためのファイルである。

```
#include <cstl/list.h>
#include "../Library/CSTLString.h"
#include "AST.h"
```

### データ構造

- struct [param\\_information](#)
- struct [function\\_information](#)

## 型定義

- typedef struct `param_information` `PARAM_INFORMATION`
- typedef struct `function_information` `FUNCTION_INFORMATION`

## 関数

- `FUNCTION_INFORMATION * new_FUNCTION_INFORMATION` (`AST *function_node`, `CSTLString *return_type`, `CSTLString *function_name`, `PARAM_INFORMATION_LIST *param_informaiton_list`)
- `PARAM_INFORMATION * new_PARAM_INFORMATION` (`CSTLString *param_type`, `CSTLString *param_name`, `int array_level`, `int pointer_level`, `int in_out_flag`)
- void `getFunctionInformation` (`FUNCTION_INFORMATION_LIST *function_information_list`, `AST *root`)
- void `getFunctionInformationFromFile` (`FUNCTION_INFORMATION_LIST *function_information_list`, `char *file_name`)
- void `deleteParameterDefine` (`CSTLString *target_string`)
- void `printFUNCTION_INFORMATION_LIST` (`FUNCTION_INFORMATION_LIST *function_information_list`)
- `FUNCTION_INFORMATION * searchFUNCTION_INFORMATION` (`CSTLString *target_function_name`, `FUNCTION_INFORMATION_LIST *function_information_list`)
- int `getPointerLevelFromFUNCTION_INFORMATION_LIST` (`CSTLString *target_function_name`, `FUNCTION_INFORMATION_LIST *function_information_list`)
- void `getParamInformationFromFunctionDifinition` (`AST *param_info_node`, `PARAM_INFORMATION_LIST *param_information_list`)
- int `getIN_OUT_FLAG` (`CSTLString *param_type`, `int pointer_level`, `int array_level`)

### 4.5.1 説明

このファイルは関数に関する情報を取得するためのファイルである。

作者

faithnh

### 4.5.2 型定義

#### 4.5.2.1 typedef struct `function_information` `FUNCTION_INFORMATION`

関数に関する情報。関数からの検証式の生成などに用いる。

#### 4.5.2.2 typedef struct param\_information PARAM\_INFORMATION

引数に関する情報

#### 4.5.3 関数

##### 4.5.3.1 void deleteParameterDefine ( CSTLString \* *target\_string* )

指定した関数の定義自体から、パラメータ定義などを削除する

引数

<i>target_string</i>	対象の関数の定義自体の文字列
----------------------	----------------

戻り値

なし

##### 4.5.3.2 void getFunctionInformation ( FUNCTION\_INFORMATION\_LIST \* *function\_information\_list*, AST \* *root* )

指定したプログラムの AST ノードから関数定義を探し、それに基づいて、関数に関する情報を生成するし、それらのリスト *function\_information\_list* に登録する。

引数

<i>function_information_list</i>	登録先の関数に関する情報のリスト
<i>root</i>	指定したプログラムの AST ノード

戻り値

なし

##### 4.5.3.3 void getFunctionInformationFromFile ( FUNCTION\_INFORMATION\_LIST \* *function\_information\_list*, char \* *file\_name* )

指定したファイル名 *file\_name* で定義された関数に関する情報を関数に関する情報のリスト *function\_information\_list* に設定する

引数

<i>function_information_list</i>	登録先の関数に関する情報のリスト
<i>file_name</i>	指定したファイル名

戻り値

なし

#### 4.5.3.4 int getIN\_OUT\_FLAG ( CSTLString \* *param\_type*, int *pointer\_level*, int *array\_level* )

パラメータのポインタレベル・配列レベルおよび型名から、このパラメータは入力型か出力型か判定する

引数

<i>param_type</i>	パラメータの型名
<i>pointer_level</i>	ポインタのレベル
<i>array_level</i>	配列のレベル

戻り値

入力型ならば 1、出力型ならば 0 を返す。

#### 4.5.3.5 void getParamInformationFromFunctionDifinition ( AST \* *param\_info\_node*, PARAM\_INFORMATION\_LIST \* *param\_information\_list* )

パラメータリストを示すノード *param\_info\_node* から、パラメータの情報を取得し、*param\_information\_list* に登録する

引数

<i>param_info_node</i>	パラメータリストを示すノード
<i>param_information_list</i>	登録先のパラメータ情報リスト

戻り値

なし

#### 4.5.3.6 int getPointerLevelFromFUNCTION\_INFORMATION\_LIST ( CSTLString \* *target\_function\_name*, FUNCTION\_INFORMATION\_LIST \* *function\_information\_list* )

指定した関数名が関数に関する情報リストから探し、ポインタレベルを返す

引数

<i>target_function_name</i>	指定した関数名
-----------------------------	---------

<i>function_</i> - <i>information_</i> - <i>list</i>	検索対象の関数に関する情報リスト
--	------------------

#### 戻り値

見つかった場合はその関数のポインタレベルを返す、そうでない場合は-1 を返す

**4.5.3.7 FUNCTION\_INFORMATION\* new\_FUNCTION\_INFORMATION ( AST  
\* *function\_node*, CSTLString \* *return\_type*, CSTLString \* *function\_name*,  
PARAM\_INFORMATION\_LIST \* *param\_informaiton\_list* )**

関数に関する情報を生成する。

#### 引数

<i>function_</i> - <i>node</i>	対象の関数へのノード
<i>return_type</i>	返却値のタイプ
<i>function_</i> - <i>name</i>	関数名
<i>param_</i> - <i>information_</i> - <i>list</i>	パラメータに関する情報

#### 戻り値

生成された関数に関する情報へのアドレスを返す

**4.5.3.8 PARAM\_INFORMATION\* new\_PARAM\_INFORMATION ( CSTLString \*  
*param\_type*, CSTLString \* *param\_name*, int *array\_level*, int *pointer\_level*, int *in\_out\_flag*  
)**

パラメータに関する情報を生成する

#### 引数

<i>param_type</i>	パラメータの型
<i>param_</i> - <i>name</i>	パラメータの名前
<i>array_level</i>	配列のレベル
<i>pointer_</i> - <i>level</i>	ポインタのレベル
<i>in_out_flag</i>	入力型か出力型かの判定 1 : 入力 0 : 出力 2 : 入出力

#### 4.5.3.9 void printFUNCTION\_INFORMATION\_LIST ( FUNCTION\_INFORMATION\_LIST \* *function\_information\_list* )

関数に関する情報リストの内容を出力させる

引数

<i>function_information_list</i>	出力対象の関数に関する情報リスト
----------------------------------	------------------

戻り値

なし

#### 4.5.3.10 FUNCTION\_INFORMATION\* searchFUNCTION\_INFORMATION ( CSTLString \* *target\_function\_name*, FUNCTION\_INFORMATION\_LIST \* *function\_information\_list* )

関数に関する情報リストから、指定した関数名を探し、それに関する構造体へのアドレスを返す。

引数

<i>target_function_name</i>	指定した関数名
<i>function_information_list</i>	検索対象の関数に関する情報リスト

戻り値

見つかった場合はその関数に関する構造体へのアドレスを返す。みつからなければ、NULL を返す。

## 4.6 ANSICInformation/MemallocInfo.h

このファイルはメモリ確保関係の関数から、メモリ確保関係に関する情報を格納するためのものです。具体的には、C 言語プログラム上にある、malloc・calloc・realloc などといった関数から、確保している sizeof の型・realloc 使用時で対象としている変数・確保しているサイズを取得する。このファイルでできることとしては、`::include` をコメントアウトすることで省いて、プリプロセスを掛けることができます。

```
#include "PointerArrayControl.h"
```

```
#include "AST.h"
```

```
#include "Synbol.h"
```

## データ構造

- struct [memory\\_allocation\\_info](#)

## 型定義

- typedef struct [memory\\_allocation\\_info](#) MEMALLOC\_INFO

## 関数

- [MEMALLOC\\_INFO](#) \* [new\\_MEMALLOC\\_INFO\\_char](#) (char \*sizeof\_type, ARRAY\_OFFSET\_LIST \*realloc\_target, char \*size)
- [MEMALLOC\\_INFO](#) \* [new\\_MEMALLOC\\_INFO](#) (CSTLString \*sizeof\_type, ARRAY\_OFFSET\_LIST \*realloc\_target, CSTLString \*size)
- [MEMALLOC\\_INFO](#) \* [memoryAllocationAnarysis](#) (AST \*root, VARIABLE\_TABLE\_LIST \*vtlist)
- void [getMallocInformation](#) (AST \*root, CSTLString \*sizeof\_type, CSTLString \*constant)
- void [getCallocInformation](#) (AST \*root, CSTLString \*sizeof\_type, CSTLString \*constant)
- void [getReallocInformation](#) (AST \*root, VARIABLE\_TABLE\_LIST \*vtlist, CSTLString \*sizeof\_type, CSTLString \*constant, ARRAY\_OFFSET\_LIST \*realloc\_target)
- int [searchSizeof](#) (CSTLString \*sizeof\_type, AST \*root)
- void [getMallocMaxsize](#) (CSTLString \*constant, AST \*root)

### 4.6.1 説明

このファイルはメモリ確保関係の関数から、メモリ確保関係に関する情報を格納するためのものです。具体的には、C 言語プログラム上にある、malloc・calloc・realloc などといった関数から、確保している sizeof の型・realloc 使用時で対象としている変数・確保しているサイズを取得する。このファイルでできることとしては、`::include` をコメントアウトすることで省いて、プリプロセスを掛けることができます。

作者

faithnh

### 4.6.2 型定義

#### 4.6.2.1 typedef struct memory\_allocation\_info MEMALLOC\_INFO

メモリ割り当てに関する情報を格納するための構造体

### 4.6.3 関数

#### 4.6.3.1 void getCallocInformation ( AST \* root, CStdString \* sizeof\_type, CStdString \* constant )

calloc 関数に関する情報を取得する

引数

<i>root</i>	
<i>sizeof_type</i>	sizeof での型
<i>constant</i>	sizeof 以外での式 ( すなわち確保している型に対するサイズに相当する )

戻り値

なし

#### 4.6.3.2 void getMallocInformation ( AST \* root, CStdString \* sizeof\_type, CStdString \* constant )

malloc 関数に関する情報を取得する

引数

<i>root</i>	
<i>sizeof_type</i>	sizeof での型
<i>constant</i>	sizeof 以外での式 ( すなわち確保している型に対するサイズに相当する )

戻り値

なし

#### 4.6.3.3 void getMallocMaxsize ( CStdString \* constant, AST \* root )

malloc などの関数内の式から、確保したサイズを示す式を取得する。これは、sizeof(型) を 1 に書き直しながら、式を出力させることで行っている。

引数

<i>constant</i>	取得する sizeof 以外での式 ( すなわち確保している型に対するサイズに相当する )
<i>root</i>	malloc などの関数内の式への AST ノード

戻り値

なし



**4.6.3.4** void getReallocInformation ( AST \* root, VARIABLE\_TABLE\_LIST \* vtlist, CSTLString \* sizeof\_type, CSTLString \* constant, ARRAY\_OFFSET\_LIST \* realloc\_target )

realloc 関数に関する情報を取得する

引数

<i>root</i>	対象の AST ノード
<i>vtlist</i>	対象のプログラムの変数リスト
<i>sizeof_type</i>	sizeof での型
<i>constant</i>	sizeof 以外での式 ( すなわち確保している型に対するサイズに相当する )
<i>realloc</i>	realloc が対象とするオフセット情報

戻り値

なし

**4.6.3.5** MEMALLOC\_INFO\* memoryAllocationAnarysis ( AST \* root, VARIABLE\_TABLE\_LIST \* vtlist )

指定された AST ノードからメモリ確保関係の関数に関する情報を取得する

引数

<i>root</i>	指定された AST ノード
<i>vtlist</i>	realloc で realloc のターゲット情報のオフセット情報を取得するのに必要なプログラムの変数リスト

戻り値

メモリ確保関係の構造体へのアドレスを返す

**4.6.3.6** MEMALLOC\_INFO\* new\_MEMALLOC\_INFO ( CSTLString \* sizeof\_type, ARRAY\_OFFSET\_LIST \* realloc\_target, CSTLString \* size )

メモリ割り当てに関する情報を格納するための構造体のデータを生成させる

引数

<i>sizeof_type</i>	sizeof の型名
<i>realloc_target</i>	realloc 時のターゲットタイプ
<i>size</i>	malloc 時のサイズ

戻り値

メモリ割り当てに関する情報を格納するための構造体へのアドレスを返す

#### 4.6.3.7 MEMALLOC\_INFO\* new\_MEMALLOC\_INFO(char ( char \* sizeof\_type, ARRAY\_OFFSET\_LIST \* realloc\_target, char \* size )

メモリ割り当てに関する情報を格納するための構造体のデータを生成させる

引数

<i>sizeof_type</i>	sizeof の型名
<i>realloc_target</i>	realloc 時のターゲットタイプ
<i>size</i>	malloc 時のサイズ

戻り値

メモリ割り当てに関する情報を格納するための構造体へのアドレスを返す

#### 4.6.3.8 int searchSizeof ( CSTLString \* sizeof\_type, AST \* root )

指定した式の AST に対して、sizeof を探索し、その型名を sizeof\_type で取得する。もし、式に異なる 2 種類の sizeof 定義があれば、探索フラグは失敗する。また、見つからなければ、sizeof\_type は何も指定していないままの状態である。

引数

<i>sizeof_type</i>	出力する sizeof の型名
<i>root</i>	探索対象の式への AST ノード

戻り値

なし

## 4.7 ANSICInformation/PointerArrayControl.h

このファイルは、C 言語プログラム上の複雑な配列参照および直接参照による演算を各次元のオフセットとして格納します。各次元のオフセットとは動的配列や静的配列などにおいて、配列のどの部分を指しているかという式のことです。

```
#include <cstl/list.h>
#include "../Library/CSTLString.h"
#include "AST.h"
#include "Synbol.h"
#include "FunctionInformation.h"
```

データ構造

- struct [array\\_offset](#)

## 型定義

- typedef struct [array\\_offset](#) [ARRAY\\_OFFSET](#)

## 関数

- [ARRAY\\_OFFSET](#) \* [new\\_ARRAY\\_OFFSET\\_char](#) (char \*variable\_name, [AST](#) \*target\_statement, [AST](#) \*variable\_address, [OFFSET\\_LIST](#) \*offset\_list, int pointer\_level, int array\_level, int anpasand\_flag, int inc\_dec\_flag)
- [ARRAY\\_OFFSET](#) \* [new\\_ARRAY\\_OFFSET](#) (CSTLString \*variable\_name, [AST](#) \*target\_statement, [AST](#) \*variable\_address, [OFFSET\\_LIST](#) \*offset\_list, int pointer\_level, int array\_level, int anpasand\_flag, int inc\_dec\_flag)
- void [OFFSET\\_LIST\\_push\\_back\\_alloc](#) ([OFFSET\\_LIST](#) \*offset\_list, char \*string)
- int [getPointerArrayOffset](#) ([AST](#) \*root, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list, [VARIABLE\\_TABLE\\_LIST](#) \*vtlist, int offset\_level, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list, [ARRAY\\_OFFSET](#) \*\*array\_offset, [AST](#) \*target\_statement, int anpasand\_flag, int inc\_dec\_flag)
- void [getARRAY\\_OFFSET\\_LIST](#) ([AST](#) \*root, [ARRAY\\_OFFSET\\_LIST](#) \*array\_offset\_list, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list, [VARIABLE\\_TABLE\\_LIST](#) \*vtlist, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list, [AST](#) \*target\_statement)
- void [get\\_ARRAY\\_OFFSET\\_LISTIgnoreASTNAME](#) ([AST](#) \*root, [ARRAY\\_OFFSET\\_LIST](#) \*array\_offset\_list, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list, [VARIABLE\\_TABLE\\_LIST](#) \*vtlist, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list, [AST](#) \*target\_statement, CSTLString \*ignore\_ast\_name)
- void [getArrayOffsetInAnpasandInfo](#) ([AST](#) \*root, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list, [VARIABLE\\_TABLE\\_LIST](#) \*vtlist, [ARRAY\\_OFFSET\\_LIST](#) \*array\_offset\_list, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list, [AST](#) \*target\_statement)
- void [getArrayOffsetInIncDecInfo](#) ([AST](#) \*root, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list, [VARIABLE\\_TABLE\\_LIST](#) \*vtlist, [ARRAY\\_OFFSET\\_LIST](#) \*array\_offset\_list, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list, [AST](#) \*target\_statement, int inc\_dec\_flag)
- int [getUpperExpressionRelationNode](#) ([AST](#) \*target, [AST](#) \*root, [AST](#) \*\*output, [AST](#) \*\*output2)
- void [deletePointerAndArraySynbol](#) (CSTLString \*target)
- void [deletePointer](#) (CSTLString \*target)
- int [searchExpressionOrPointeArrayOrIden](#) ([AST](#) \*root, [AST](#) \*\*output)
- void [searchPointerAccessOrIdentifier](#) ([AST](#) \*root, [AST](#) \*\*output, int \*getSize)
- void [getPointerAccessOrIdentifierList](#) ([AST](#) \*root, [AST](#) \*\*\*output, int \*getSize)
- int [checkIdentifierPointerArrayLevel](#) ([AST](#) \*identifier, int offset\_level, [VARIABLE\\_TABLE\\_LIST](#) \*variable\_table\_list, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list, int \*pointer\_level, int \*array\_level)
- void [checkCallFunction](#) ([AST](#) \*call\_function, int offset\_level, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list)
- int [checkIgnoreASTList](#) ([AST](#) \*ast\_data, [ASTPOINTER\\_LIST](#) \*ignore\_ast\_list)
- void [getOFFSET\\_LISTFromVariableTable](#) ([OFFSET\\_LIST](#) \*offset\_list, [VARIABLE\\_TABLE](#) \*variable\_table)

- void `deleteOFFSET_LIST` (OFFSET\_LIST \*offset\_list)
- int `getOffsetLevelFromArrayOffset` (ARRAY\_OFFSET \*array\_offset)
- ARRAY\_OFFSET \* `maxOffsetLevelAddressFromArrayOffsetList` (ARRAY\_OFFSET\_LIST \*array\_offset\_list)
- ARRAY\_OFFSET \* `searchOffsetLevelAddressFromArrayOffsetList` (AST \*root, ARRAY\_OFFSET\_LIST \*array\_offset\_list, int pointer\_level, int array\_level)
- int `maxOffsetLevelFromArrayOffsetList` (ARRAY\_OFFSET\_LIST \*array\_offset\_list)
- void `getExpressionOffsetInfo` (AST \*expression, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, ASTPOINTER\_LIST \*ignore\_ast\_list, ARRAY\_OFFSET\_LIST \*array\_offset\_list, AST \*target\_expression, int \*switch\_mode, int allow\_subeffect)
- void `getArgumentOffsetInfo` (AST \*argument, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, ASTPOINTER\_LIST \*ignore\_ast\_list, ARRAY\_OFFSET\_LIST \*array\_offset\_list, AST \*target\_expression, int \*switch\_mode)
- void `getSingleExpressionOffsetInfo` (AST \*expression, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, ASTPOINTER\_LIST \*ignore\_ast\_list, ARRAY\_OFFSET\_LIST \*array\_offset\_list, AST \*target\_expression, int \*switch\_mode)
- void `createArrayExpression` (CSTLString \*output, ARRAY\_OFFSET \*array\_offset, int output\_level)
- int `createValidateVariableArrayExpression` (CSTLString \*output, ARRAY\_OFFSET \*array\_offset)
- void `moveArrayOffsetList` (ARRAY\_OFFSET\_LIST \*fromlist, ARRAY\_OFFSET\_LIST \*tolist, int move\_start)
- void `copyArrayOffsetList` (ARRAY\_OFFSET\_LIST \*fromlist, ARRAY\_OFFSET\_LIST \*tolist, int move\_start)
- void `getDeclaratorArrayOffset` (ARRAY\_OFFSET\_LIST \*declarator\_array\_offset\_list, AST \*declarator\_expression, AST \*target\_expression, VARIABLE\_TABLE\_LIST \*vtlist)
- ARRAY\_OFFSET \* `searchARRAY_OFFSET_LIST` (ARRAY\_OFFSET\_LIST \*array\_offset\_list, CSTLString \*name)
- void `minusArrayOffsetList` (ARRAY\_OFFSET\_LIST \*target\_array\_offset\_list, ARRAY\_OFFSET\_LIST \*delete\_array\_offset\_list)
- void `ARRAY_OFFSET_LIST_push_back_ref_not_dup` (ARRAY\_OFFSET\_LIST \*target\_array\_offset\_list, ARRAY\_OFFSET \*array\_offset)
- void `printASTPOINTER_LIST` (ASTPOINTER\_LIST \*astpointer\_list)

#### 4.7.1 説明

このファイルは、C 言語プログラム上の複雑な配列参照および直接参照による演算を各次元のオフセットとして格納します。各次元のオフセットとは動的配列や静的配列などにおいて、配列のどの部分を指しているかという式のことです。

作者

faithnh

## 4.7.2 型定義

### 4.7.2.1 typedef struct array\_offset ARRAY\_OFFSET

配列やポインタの各次元のオフセット関係を格納するための構造体である。配列オフセットと呼ばれる

## 4.7.3 関数

### 4.7.3.1 void ARRAY\_OFFSET\_LIST\_push\_back\_ref\_not\_dup ( ARRAY\_OFFSET\_LIST \* target\_array\_offset\_list, ARRAY\_OFFSET \* array\_offset )

配列オフセットリストに配列オフセット情報を追加する。ただし、変数名が重複するのであれば、追加しない。

引数

<i>target_array_offset_list</i>	追加先の配列オフセットリスト
<i>array_offset</i>	追加する配列オフセット情報

戻り値

なし

### 4.7.3.2 void checkCallFunction ( AST \* call\_function, int offset\_level, FUNCTION\_INFORMATION\_LIST \* function\_information\_list )

関数呼び出しを示す AST ノードが、登録されている関数に関する情報に含まれているかどうかを調べ、もし、その関数のポインタレベルがオフセットレベルと一致した場合は、エラーを出力し、強制終了させる。

引数

<i>call_function</i>	関数呼び出しを示す AST ノード
<i>offset_level</i>	オフセットレベル
<i>function_information_list</i>	関数に関する情報のリスト

戻り値

なし

**4.7.3.3** `int checkIdentifierPointerArrayLevel ( AST * identifier, int offset_level, VARIABLE_TABLE_LIST * variable_table_list, ASTPOINTER_LIST * ignore_ast_list, int * pointer_level, int * array_level )`

識別子の名前を一致する変数を変数リストから探す。このとき、一致する変数を調べたら、ポインタと配列の次元も調べ、オフセットレベル以上であれば、見つけたことになり、1を返す。そうでなければ、0を返す。また、`ignore_ast_list`は無視する IDENTIFIER の AST のアドレスリストを見つけるたびに登録される。もし、`ignore_ast_list`に登録されているノードなら、それは無視される。また、見つけるのに成功した場合、その該当する変数の配列レベルやポインタのレベルも返す。

#### 引数

<code>identifier</code>	識別子の名前
<code>offset_level</code>	オフセットレベル
<code>variable_table_list</code>	変数テーブルリスト
<code>ignore_ast_list</code>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<code>pointer_level</code>	変数リストから見つけた変数のポインタレベル
<code>array_level</code>	変数リストから見つけた変数の配列レベル

#### 戻り値

識別子の名前およびオフセットレベルが条件を満たしていれば1を返し、そうでなければ0を返す。

**4.7.3.4** `int checkIgnoreASTList ( AST * ast_data, ASTPOINTER_LIST * ignore_ast_list )`

指定した AST ノード `ast_data` が AST アドレスリスト `ignore_ast_list` に存在するかどうかを調べる。存在する場合は1をかえす。存在しない場合は、`ast_data` のアドレスを `ignore_ast_list` に追加した上、0を返す。

#### 引数

<code>ast_data</code>	指定した AST ノード
<code>ignore_ast_list</code>	調べる対象の AST アドレスリスト
<code>ignore_ast_list</code> に存在する AST アドレスの場合1を返し、そうでない場合は0を返す。	

**4.7.3.5** void copyArrayOffsetList ( ARRAY\_OFFSET\_LIST \* *fromlist*, ARRAY\_OFFSET\_LIST \* *tolist*, int *move\_start* )

配列オフセットリスト *fromlist* 内の *move\_start* 以降のデータをすべて、もう一方の配列オフセットリスト *tolist* にコピーさせる

引数

<i>fromlist</i>	コピー元の配列オフセットリスト
<i>tolist</i>	コピー先の配列オフセットリスト
<i>move_start</i>	移動させたいデータの位置 (先頭から 0 番目とする)

戻り値

なし

**4.7.3.6** void createArrayExpression ( CSTLString \* *output*, ARRAY\_OFFSET \* *array\_offset*, int *output\_level* )

配列オフセット情報から、任意の次元までの配列式を生成する

引数

<i>output</i>	配列式を生成される文字列
<a href="#"><i>array_offset</i></a>	対象の配列オフセット
<i>output_level</i>	出力したい次元 (このとき、配列オフセットを超える値を入れた場合は、配列オフセットが次元までの配列式を出力する)

戻り値

なし

**4.7.3.7** int createValidateVariableArrayExpression ( CSTLString \* *output*, ARRAY\_OFFSET \* *array\_offset* )

配列オフセット情報から、検証用変数に使われる配列式を生成し、オフセットレベルを返す

引数

<i>output</i>	生成先の文字列
<a href="#"><i>array_offset</i></a>	対象の配列オフセット

戻り値

配列オフセット情報から生成されたオフセットレベルを返す

**4.7.3.8 void deleteOFFSET\_LIST ( OFFSET\_LIST \* *offset\_list* )**

オフセットリスト *offset\_list* の中身を完全に解放させる

引数

<i>offset_list</i>	
--------------------	--

戻り値

なし

オフセットリスト *offset\_list* の中身を完全に解放させる

引数

<i>offset_list</i>	解放させる <i>offset_list</i>
--------------------	--------------------------

戻り値

なし

**4.7.3.9 void deletePointer ( CString \* *target* )**

変数名からポインタを示す記号のみ全て削除する

引数

<i>target</i>	変更対象の変数名
---------------	----------

戻り値

なし

**4.7.3.10 void deletePointerAndArraySymbol ( CString \* *target* )**

変数名から配列およびポインタを示す記号を全て削除する

引数

<i>target</i>	変更対象の変数名
---------------	----------

戻り値

なし



```
4.7.3.11 void get_ARRAY_OFFSET_LISTIgnoreASTNAME ( AST * root, ARRAY_OFFSET_LIST
* array_offset_list, FUNCTION_INFORMATION_LIST * function_information_list,
VARIABLE_TABLE_LIST * vtlist, ASTPOINTER_LIST * ignore_ast_list, AST *
target_statement, CSTLString * ignore_ast_name )
```

ポインタおよび配列変数の各次元のオフセットリストを取得する。また、無視をする対象のノードを設定可能である。

#### 引数

<i>root</i>	オフセットリストに該当する AST ノード
<i>array_offset_list</i>	ポインタおよび配列のオフセット情報のリスト
<i>function_information_list</i>	関数に関する情報リスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>target_statement</i>	検証式の対象となるステートメント
<i>ignore_ast_name</i>	無視をする A S T 名

#### 戻り値

なし

```
4.7.3.12 void getArgumentOffsetInfo ( AST * argument, FUNCTION_INFORMATION_LIST
* function_information_list, VARIABLE_TABLE_LIST * vtlist, ASTPOINTER_LIST *
ignore_ast_list, ARRAY_OFFSET_LIST * array_offset_list, AST * target_expression,
int * switch_mode )
```

指定した引数から、必要なオフセット情報を取得する

#### 引数

<i>argument</i>	指定した引数に関する AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>array_offset_list</i>	各ポインタおよび配列ごとのオフセットのリスト
<i>target_expression</i>	この左辺式の上位に位置する AST ノード

<i>switch_mode</i>	直接アクセスおよび配列アクセスを探すか、IDENTIFIER を探すかどうかのスイッチフラグ 0 : 両方さがす 1 : direct_ref や array_access のみ探す
--------------------	---

戻り値

なし

**4.7.3.13** void getARRAY\_OFFSET\_LIST ( AST \* root, ARRAY\_OFFSET\_LIST \* array\_offset\_list, FUNCTION\_INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, ASTPOINTER\_LIST \* ignore\_ast\_list, AST \* target\_statement )

ポインタおよび配列変数の各次元のオフセットリストを取得する

引数

<i>root</i>	オフセットリストに該当する AST ノード
<i>array_offset_list</i>	ポインタおよび配列のオフセット情報のリスト
<i>function_information_list</i>	関数に関する情報リスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>target_statement</i>	検証式の対象となるステートメント

戻り値

なし

**4.7.3.14** void getArrayOffsetInAnpasandInfo ( AST \* root, FUNCTION\_INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, ARRAY\_OFFSET\_LIST \* array\_offset\_list, ASTPOINTER\_LIST \* ignore\_ast\_list, AST \* target\_statement )

address\_ref であるノード内を探索し、それに対するアドレス参照や、識別子を探し出し、見つけたら配列オフセットリスト array\_offset\_list へ入れる。

引数

<i>root</i>	右辺式に関する AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	メモリ確保情報を取得するのに必要なプログラム変数リスト
<i>array_offset_list</i>	左辺式上にあるポインタ参照に対するオフセットリスト

<i>ignore_ast_list</i>	同じ位置のポインタが来ても無視するためのリスト
<i>target_statement</i>	この計算式を属している AST ノードへのアドレス ( 基本的に expression_statement であるノードが入る )

戻り値

なし

**4.7.3.15** void `getArrayOffsetIncDecInfo ( AST * root, FUNCTION_INFORMATION_LIST * function_information_list, VARIABLE_TABLE_LIST * vtlist, ARRAY_OFFSET_LIST * array_offset_list, ASTPOINTER_LIST * ignore_ast_list, AST * target_statement, int inc_dec_flag )`

inc\_expr や dec\_expr などのインクリメントやデクリメント式であるノード内を探索し、それに対するアドレス参照や、識別子を探し出し、見つけたら配列オフセットリスト array\_offset\_list へ入れる。

引数

<i>root</i>	inc_expr や dec_expr などのインクリメントやデクリメント式に関する AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	メモリ確保情報を取得するのに必要なプログラム変数リスト
<i>array_offset_list</i>	左辺式上にあるポインタ参照に対するオフセットリスト
<i>ignore_ast_list</i>	同じ位置のポインタが来ても無視するためのリスト
<i>target_statement</i>	この計算式を属している AST ノードへのアドレス ( 基本的に expression_statement であるノードが入る )
<i>inc_dec_flag</i>	インクリメントおよびデクリメントが含まれているかどうかのフラグ 1 : インクリメントが含まれている 2 : デクリメントが含まれている

戻り値

なし

**4.7.3.16** void `getDeclaratorArrayOffset ( ARRAY_OFFSET_LIST * declarator_array_offset_list, AST * declarator_expression, AST * target_expression, VARIABLE_TABLE_LIST * vtlist )`

変数テーブルから、変数の定義に対するノードに該当する情報を探し、それに対する配列オフセット情報を取得する。

## 引数

<i>declarator_-array_-offset_list</i>	取得先の配列オフセット情報
<i>declarator_-expression</i>	変数定義までの AST アドレス
<i>target_-expression</i>	対象の <i>declarator_with_init</i> への AST アドレス
<i>vtlist</i>	調べる先の変数テーブル

## 戻り値

なし

**4.7.3.17** void `getExpressionOffsetInfo` ( AST \* *expression*, FUNCTION\_INFORMATION\_LIST \* *function\_information\_list*, VARIABLE\_TABLE\_LIST \* *vtlist*, ASTPOINTER\_LIST \* *ignore\_ast\_list*, ARRAY\_OFFSET\_LIST \* *array\_offset\_list*, AST \* *target\_expression*, int \* *switch\_mode*, int *allow\_subeffect* )

指定した式から、必要なオフセット情報を取得する

## 引数

<i>expression</i>	指定した式に関する AST ノード
<i>function_-information_-list</i>	関数に関する情報のリスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>ignore_ast_-list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>array_-offset_list</i>	各ポインタおよび配列ごとのオフセットのリスト
<i>target_-expression</i>	この左辺式の上位に位置する AST ノード
<i>switch_-mode</i>	直接アクセスおよび配列アクセスを探すか、IDENTIFIER を探すかどうかのスイッチフラグ 0 : 両方さがす 1 : <i>direct_ref</i> や <i>array_access</i> のみ探す
<i>allow_-subeffect</i>	副作用の式を許すかどうかのフラグ 1 : 許す 0 : 許さない

## 戻り値

なし

**4.7.3.18** void `getOFFSET_LISTFromVariableTable` ( OFFSET\_LIST \* *offset\_list*, VARIABLE\_TABLE \* *variable\_table* )

変数テーブルデータ *variable\_table* からオフセットリスト *offset\_list* を生成する

## 引数

<i>offset_list</i>	生成先のオフセットリスト
<i>variable_table</i>	変数テーブルデータ

## 戻り値

なし

## 4.7.3.19 int getOffsetLevelFromArrayOffset ( ARRAY\_OFFSET \* array\_offset )

指定した配列オフセットから、演算後のポインタレベルを求める 演算後のポインタレベル = この変数の配列とポインタレベルの合計値 + アンパサンドフラグ ( 挟んでいるなら 1、そうでない場合は 0 ) - この配列オフセット内のオフセットリスト

## 引数

<i>array_offset</i>	指定した配列オフセット
---------------------	-------------

## 戻り値

求めた演算後のポインタレベルを返す

## 4.7.3.20 void getPointerAccessOrIdentifierList ( AST \* root, AST \*\*\* output, int \* getSize )

direct\_ref として指定した AST ノード root から、以下のノードを探しだし、それを AST リスト output として取得する

IDENTIFIER array\_access, direct\_ref, IDENTIFIER, primary\_expression

なお、output の内容を NULL にすることで、root より下位のノードからが検索の対象となる。見つからない場合は 0 である。

## 引数

<i>root</i>	指定した AST ノード
<i>output</i>	上記の見つけたノードへのアドレス

## 戻り値

上記の条件で値を返却する。

**4.7.3.21** `int getPointerArrayOffset ( AST * root, FUNCTION_INFORMATION_LIST * function_information_list, VARIABLE_TABLE_LIST * vtlist, int offset_level, ASTPOINTER_LIST * ignore_ast_list, ARRAY_OFFSET ** array_offset, AST * target_statement, int anpasand_flag, int inc_dec_flag )`

ポインタおよび配列変数の各次元のオフセットとなる式を求める

引数

<i>root</i>	左辺値に関する AST ノード
<i>function_information_list</i>	関数に関する情報リスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>offset_level</i>	オフセットレベルを計算するためのところ。基本的に 0 を入力する。1 以上入力すれば、それが最下位レベルとなる。
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>array_offset</i>	ポインタおよび配列のオフセット情報
<i>target_statement</i>	検証式の対象となるステートメント
<i>anpasand_flag</i>	アンパサンドを挟んでいるかどうかのフラグ 1 : 挟んでいる 0 : 挟んでいない
<i>inc_dec_flag</i>	インクリメントおよびデクリメントが含まれているかどうかのフラグ 0 : 含んでいない 1 : インクリメントが含まれている 2 : デクリメントが含まれている

戻り値

なし

**4.7.3.22** `void getSingleExpressionOffsetInfo ( AST * expression, FUNCTION_INFORMATION_LIST * function_information_list, VARIABLE_TABLE_LIST * vtlist, ASTPOINTER_LIST * ignore_ast_list, ARRAY_OFFSET_LIST * array_offset_list, AST * target_expression, int * switch_mode )`

指定した式から、必要なオフセット情報を取得する。これは副作用の式を許す

引数

<i>expression</i>	指定した式に関する AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>array_offset_list</i>	各ポインタおよび配列ごとのオフセットのリスト

<i>target_expression</i>	この左辺式の上位に位置する AST ノード
<i>switch_mode</i>	直接アクセスおよび配列アクセスを探すか、IDENTIFIER を探すかどうかのスイッチフラグ 0 : 両方さがす 1 : direct_ref や array_access のみ探す

戻り値

なし

**4.7.3.23** int getUpperExpressionRelationNode ( AST \* *target*, AST \* *root*, AST \*\* *output*, AST \*\* *output2* )

ポインタのオフセットの検証対象となっている変数を示す AST ノード *target* から、間接的にどの関係の中に位置しているかどうかを調べ、そのノードのアドレス *output* として返す。このとき、*target* より明らかに上位である AST ノード *root* を設定しなければならない。また、*output* が *minus\_expr* の場合はそのポインタよりひとつ下が左辺か右辺かどうかを調べるために、そのポインタの一つ下のノードを *output2* へ代入する。

引数

<i>target</i>	検証対象となっている変数
<i>root</i>	検証対象のノード
<i>output</i>	出力される間接的に関係しているノードへのアドレス
<i>output2</i>	<i>output</i> が <i>minus_expr</i> の場合、 <i>minus_expr</i> より 1 つ下のノードがここに代入される

戻り値

検索が成功したかどうかのフラグ。成功した場合は 1、そうでない場合は 0 を返す。  
なし

ポインタのオフセットの検証対象となっている変数を示す AST ノード *target* から、間接的にどの関係の中に位置しているかどうかを調べ、そのノードのアドレス *output* として返す。このとき、*target* より明らかに上位である AST ノード *root* を設定しなければならない。また、*output* が *minus\_expr* の場合はそのポインタよりひとつ下が左辺か右辺かどうかを調べるために、そのポインタの一つ下のノードを *output2* へ代入する。

引数

<i>target</i>	検証対象となっている変数
<i>root</i>	検証対象のノード
<i>output</i>	出力される間接的に関係しているノードへのアドレス
<i>output2</i>	<i>output</i> が <i>minus_expr</i> の場合、 <i>minus_expr</i> より 1 つ下のノードがここに代入される

## 戻り値

検索が成功したかどうかのフラグ。成功した場合は 1、そうでない場合は 0 を返す。

#### 4.7.3.24 `ARRAY_OFFSET* maxOffsetLevelAddressFromArrayOffsetList ( ARRAY_OFFSET_LIST * array_offset_list )`

指定した配列オフセットリストでの演算後のポインタレベルの最大レベルである配列オフセットのアドレスを求める。配列オフセットが空の場合は NULL を代入する。

## 引数

<code>array_offset_list</code>	指定した配列オフセットリスト
--------------------------------	----------------

## 戻り値

求めた演算後のポインタレベルが最大である配列オフセットのアドレスを返す

#### 4.7.3.25 `int maxOffsetLevelFromArrayOffsetList ( ARRAY_OFFSET_LIST * array_offset_list )`

指定した配列オフセットリストでの演算後のポインタレベルの最大レベルを求める。配列オフセットが空の場合は 0 を代入する。

## 引数

<code>array_offset_list</code>	指定した配列オフセットリスト
--------------------------------	----------------

## 戻り値

求めた演算後のポインタレベルを返す

#### 4.7.3.26 `void minusArrayOffsetList ( ARRAY_OFFSET_LIST * target_array_offset_list, ARRAY_OFFSET_LIST * delete_array_offset_list )`

対象の配列オフセットリスト `target_array_offset_list` に対して、対象から取り除きたい配列オフセットリスト `delete_array_offset_list` の名前に該当する配列オフセット情報を削除する。

対象の配列オフセットリスト 対象から取り除きたい配列オフセットリスト

## 戻り値

なし



**4.7.3.27** void moveArrayOffsetList ( ARRAY\_OFFSET\_LIST \* *fromlist*, ARRAY\_OFFSET\_LIST \* *tolist*, int *move\_start* )

配列オフセットリスト *fromlist* 内の *move\_start* 以降のデータをすべて、もう一方の配列オフセットリスト *tolist* に移動させる

引数

<i>fromlist</i>	移動もとの配列オフセットリスト
<i>tolist</i>	移動先の配列オフセットリスト
<i>move_start</i>	移動させたいデータの位置 (先頭から 0 番目とする)

戻り値

なし

**4.7.3.28** ARRAY\_OFFSET\* new\_ARRAY\_OFFSET ( CSTLString \* *variable\_name*, AST \* *target\_statement*, AST \* *variable\_address*, OFFSET\_LIST \* *offset\_list*, int *pointer\_level*, int *array\_level*, int *anpasand\_flag*, int *inc\_dec\_flag* )

配列やポインタの各次元のオフセット関係を格納するための構造体のデータを生成させる

引数

<i>variable_name</i>	変数名
<i>target_statement</i>	ターゲットの statement
<i>variable_address</i>	この変数名が指している AST アドレス
<i>offset_list</i>	各次元のオフセット
<i>pointer_level</i>	この変数のポインタレベル
<i>array_level</i>	この変数の配列レベル
<i>anpasand_flag</i>	この変数はアンパサンドを挟んでいるかどうかのフラグ 1 : 挟んでいる 0 : 挟んでいない
<i>inc_dec_flag</i>	インクリメントおよびデクリメントが含まれているかどうかのフラグ 0 : 含んでいない 1 : インクリメントが含まれている 2 : デクリメントが含まれている

戻り値

配列やポインタの各次元のオフセットに関する構造体へのアドレスを返す

**4.7.3.29** `ARRAY_OFFSET* new_ARRAY_OFFSET_char ( char * variable_name, AST * target_statement, AST * variable_address, OFFSET_LIST * offset_list, int pointer_level, int array_level, int anpasand_flag, int inc_dec_flag )`

配列やポインタの各次元のオフセット関係を格納するための構造体のデータを生成させる

引数

<i>variable_name</i>	変数名
<i>target_statement</i>	ターゲットの statement
<i>variable_address</i>	この変数名が指している AST アドレス
<i>offset_list</i>	各次元のオフセット
<i>pointer_level</i>	この変数のポインタレベル
<i>array_level</i>	この変数の配列レベル
<i>anpasand_flag</i>	アンパサンドを挟んでいるかどうかのフラグ 1 : 挟んでいる 0 : 挟んでいない
<i>inc_dec_flag</i>	インクリメントおよびデクリメントが含まれているかどうかのフラグ 0 : 含んでいない 1 : インクリメントが含まれている 2 : デクリメントが含まれている

戻り値

配列やポインタの各次元のオフセットに関する構造体へのアドレスを返す

**4.7.3.30** `void OFFSET_LIST_push.back.alloc ( OFFSET_LIST * offset_list, char * string )`

任意の文字列を、動的変数としてオフセットリストに追加する。

引数

<i>offset_list</i>	対象のオフセットリスト
<i>string</i>	任意の文字列

戻り値

なし

**4.7.3.31** `void printASTPOINTER_LIST ( ASTPOINTER_LIST * astpointer_list )`

AST のポインタリストの内容を出力させる

引数

<i>astpointer_-list</i>	AST のポインタリスト
-------------------------	--------------

戻り値

なし

#### 4.7.3.32 **ARRAY\_OFFSET\*** `searchARRAY_OFFSET_LIST ( ARRAY_OFFSET_LIST * array_offset_list, CSTLString * name )`

配列オフセットリスト `array_offset_list` から、指定した変数名を探索し、見つければその変数名へのアドレスを返す。

引数

<i>array_-offset_list</i>	探索対象の配列オフセットリスト
<i>name</i>	探索したい変数名

戻り値

見つければ変数名へのアドレスを返す。そうでなければ NULL を返す。

#### 4.7.3.33 **int** `searchExpressionOrPointeArrayOrIden ( AST * root, AST ** output )`

`primary_expression` として指定した AST ノード `root` から、その次の下位である次のノード名を探し出し、そのアドレスを `output` へ出力させ、1 を返す。

`minus_expr`, `plus_expr`, `array_access`, `direct_ref`, `IDENTIFIER`, `primary_expression`

なお、`output` の内容を NULL にすることで、`root` より下位のノードからが検索の対象となる。また、ポインタ計算の関係上、+-演算演算子を示すようなものや `CONSTANT`(定数) が来た場合のみ、-1 を返す。見つからない場合は 0 である。

引数

<i>root</i>	指定した AST ノード
<i>output</i>	上記の見つけたノードへのアドレス

戻り値

上記の条件で値を返却する。

#### 4.7.3.34 `ARRAY_OFFSET* searchOffsetLevelAddressFromArrayOffsetList ( AST * root, ARRAY_OFFSET_LIST * array_offset_list, int pointer_level, int array_level )`

対象の AST ノードから、演算後のポインタレベルが指定されたポインタレベルと配列レベルの合計と一致するような変数の配列オフセットを指定された配列オフセットリストから探し出し、見つかったらアドレスを取得する

##### 引数

<i>root</i>	対象の AST ノード
<i>array_offset_list</i>	対象の配列オフセットリスト
<i>pointer_level</i>	指定するポインタレベル
<i>array_level</i>	指定する配列レベル

##### 戻り値

演算後のポインタレベルと指定されたポインタレベルと配列レベルの合計が一致するような変数を返す。失敗した場合は NULL を返す。

#### 4.7.3.35 `void searchPointerAccessOrIdentifier ( AST * root, AST ** output, int * getSize )`

`primary_expression` として指定した AST ノード `root` から、その次の下位である次のノード名を探し出し、そのアドレスを `output` へ出力させ、1 を返す。

`minus_expr`, `plus_expr`, `array_access`, `direct_ref`, `IDENTIFIER`, `primary_expression`

なお、`output` の内容を NULL にすることで、`root` より下位のノードからが検索の対象となる。また、ポインタ計算の関係上、`++` 演算演算子を示すようなものや `CONSTANT`(定数) が来た場合のみ、-1 を返す。見つからない場合は 0 である。

##### 引数

<i>root</i>	指定した AST ノード
<i>output</i>	上記の見つけたノードへのアドレス

##### 戻り値

上記の条件で値を返却する。

## 4.8 ANSICInformation/PreProcess.h

このファイルは構文解析を通せるように、Gcc で C 言語にプリプロセス（前処理）をさせるものです。このファイルでできることとしては、`::include` をコメントアウトすることで省いて、プリプロセスを掛けることができます。

```
#include <cstl/list.h>
```

```
#include "../Library/CSTLString.h"
```

## データ構造

- struct [include\\_data](#)

## 型定義

- typedef struct [include\\_data](#) [INCLUDE\\_DATA](#)

## 関数

- [INCLUDE\\_DATA](#) \* [new\\_INCLUDE\\_DATA](#) (CSTLString \*[include\\_data](#), int line)
- int [preProcessor](#) (char \*source\_file\_name)
- int [includeComment](#) (char \*source\_file\_name)
- int [adjustProgramStart](#) (char \*source\_file\_name)
- void [readIncludeDataFromFile](#) (char \*file\_name, [INCLUDE\\_LIST](#) \*[include\\_list](#))
- void [addIncludeDataFromFile](#) (char \*file\_name, [INCLUDE\\_LIST](#) \*[include\\_list](#))

### 4.8.1 説明

このファイルは構文解析を通せるように、Gcc で C 言語にプリプロセス（前処理）をさせるものです。このファイルでできることとしては、`::include` をコメントアウトすることで省いて、プリプロセスを掛けることができます。

作者

faithnh

### 4.8.2 型定義

#### 4.8.2.1 typedef struct [include\\_data](#) [INCLUDE\\_DATA](#)

インクルードファイルに関する情報が格納される。これは、検証式付加時にインクルードファイルを付加するのに使用する。

### 4.8.3 関数

#### 4.8.3.1 void [addIncludeDataFromFile](#) ( char \* *file\_name*, [INCLUDE\\_LIST](#) \* *include\_list* )

インクルードリストを基に、対象のファイルにインクルードを追加する

引数

<i>file_name</i>	開くファイル名
<i>include_list</i>	インクルードリスト

戻り値

なし

#### 4.8.3.2 int adjustProgramStart ( char \* source\_file\_name )

プログラムの始まりを示す位置まですべて削除する

引数

<i>source_file_name</i>	対象のソースファイル名
-------------------------	-------------

戻り値

成功したかどうかを示す 成功した場合は 1 をかえし、そうでない場合は 0 を返す

#### 4.8.3.3 int includeComment ( char \* source\_file\_name )

#include にコメントを付けておく。コメントアウトをかけた結果はファイル名\_out.c として出力される

引数

<i>source_file_name</i>	対象のソースファイル名
-------------------------	-------------

戻り値

成功したかどうかを示す 成功した場合は 1 をかえし、そうでない場合は 0 を返す

#### 4.8.3.4 INCLUDE\_DATA\* new\_INCLUDE\_DATA ( CString \* include\_data, int line )

新しいインクルードファイルを生成する。

引数

<i>include_data</i>	include ファイルの名前
<i>line</i>	その行数

## 4.8.3.5 int preProcessor ( char \* source\_file\_name )

実行させたいソースにプリプロセッサをかける。プリプロセッサのかけたファイルはファイル名\_out\_pre.c として出力される

## 引数

<i>source_file_name</i>	実行させたいソースファイル名
-------------------------	----------------

## 戻り値

成功したかどうかを示す 成功した場合は 1 をかえし、そうでない場合は 0 を返す

## 4.8.3.6 void readIncludeDataFromFile ( char \* file\_name, INCLUDE\_LIST \* include\_list )

#include にコメントをかけたもののみを取り出し、取りだしたものをインクルードファイルのリスト include\_list にいれる

## 引数

<i>file_name</i>	開くファイル名
<i>include_list</i>	インクルードリスト

## 戻り値

なし

## 4.9 ANSICInformation/Return\_Info.h

これはリターン命令に関する情報を取得するためのものである

```
#include "PointerArrayControl.h"
```

## データ構造

- struct [return\\_info](#)

## 型定義

- typedef struct [return\\_info](#) RETURN\_INFO

## 関数

- [RETURN\\_INFO](#) \* [new\\_RETURN\\_INFO](#) ([AST](#) \*target\_expression, ARRAY\_OFFSET\_LIST \*return\_array\_offset\_list)

### 4.9.1 説明

これはリターン命令に関する情報を取得するためのものである

作者

faithnh

### 4.9.2 型定義

#### 4.9.2.1 typedef struct return\_info RETURN\_INFO

リターン命令に関する情報

### 4.9.3 関数

#### 4.9.3.1 RETURN\_INFO\* new\_RETURN\_INFO ( AST \* target\_expression, ARRAY\_OFFSET\_LIST \* return\_array\_offset\_list )

リターン命令に関する情報を生成させる

引数

<i>target_expression</i>	リターン命令自体へのノードへのアドレス
<i>return_array_offset_list</i>	リターン命令で表記された式の配列オフセットリスト

戻り値

生成されたリターン命令に関する情報へのアドレスを返す

## 4.10 ANSICInformation/SubEffectCheck.h

このファイルには副作用式のチェックする関数や代入式のタイプを求める関数が含まれています

```
#include "AST.h"
```

関数

- int [checkContainSubEffectStatement](#) (AST \*node)
- int [getAssignment\\_TYPE](#) (AST \*assignment\_expression\_list)



### 4.10.1 説明

このファイルには副作用式のチェックする関数や代入式のタイプを求める関数が含まれています

作者

faithnh

### 4.10.2 関数

#### 4.10.2.1 int checkContainSubEffectStatement ( AST \* node )

指定された AST ノード node から、副作用式 (インクリメント式・デクリメント式・代入式) が含まれているかどうかチェックする。

引数

<i>node</i>	対象の AST ノード
-------------	-------------

戻り値

含まれていた式がインクリメント式の場合 1、デクリメント式の場合 2、代入式の場合 3 とする。含まれていない場合は 0 を返す。

#### 4.10.2.2 int getAssignment\_TYPE ( AST \* assignment\_expression\_list )

代入式のタイプを出力させる

引数

<i>assignment_expression_list</i>	代入式に関する AST ノードのリスト
-----------------------------------	---------------------

戻り値

代入式のタイプに応じた値を返却する 0:=,1:+=,2:-=,3:\*=,4:/=,5:=,6:<<=,7:>>=,8:&=,9:|=,10:^=

## 4.11 ANSICInformation/Synbol.h

このファイルは、構文解析によって生成された抽象構文木 (AST) から、変数・typedef テーブル・構造体テーブルを生成させることができます。また、typedef テーブルの生成は、C 言語の構文解析では必須な処理です。

```
#include <stdio.h>
```

```
#include <cstl/list.h>
#include "../Library/CSTLString.h"
#include "../Library/IdList.h"
#include "AST.h"
```

## データ構造

- struct [typedef\\_table](#)
- struct [variable\\_table](#)
- struct [struct\\_table](#)

## 型定義

- typedef struct [typedef\\_table](#) [TYPEDEF\\_TABLE](#)
- typedef struct [variable\\_table](#) [VARIABLE\\_TABLE](#)
- typedef struct [struct\\_table](#) [STRUCT\\_TABLE](#)

## 関数

- [TYPEDEF\\_TABLE](#) \* [new\\_TYPEDEF\\_TABLE](#) (CSTLString \*target\_type, CSTLString \*change\_type)
- [VARIABLE\\_TABLE](#) \* [new\\_VARIABLE\\_TABLE](#) (int enable\_start, int enable\_end, [AST](#) \*declaration\_location\_address, int block\_level, int block\_id, IDLIST \*idlist, CSTLString \*type, CSTLString \*variable\_name, [AST](#) \*initarizer)
- [STRUCT\\_TABLE](#) \* [new\\_STRUCT\\_TABLE\\_with\\_char](#) (int line, char \*type, char \*struct\_name, [VARIABLE\\_TABLE\\_LIST](#) \*member\_list)
- [TYPEDEF\\_TABLE](#) \* [new\\_TYPEDEF\\_TABLE\\_with\\_char](#) (char \*target\_type, char \*change\_type)
- [VARIABLE\\_TABLE](#) \* [new\\_VARIABLE\\_TABLE\\_with\\_char](#) (int enable\_start, int enable\_end, [AST](#) \*declaration\_location\_address, int block\_level, int block\_id, IDLIST \*idlist, char \*type, char \*variable\_name, [AST](#) \*initarizer)
- [STRUCT\\_TABLE](#) \* [new\\_STRUCT\\_TABLE](#) (int line, CSTLString \*type, CSTLString \*struct\_name, [VARIABLE\\_TABLE\\_LIST](#) \*member\_list)
- void [getTYPEDEF\\_TABLE\\_DATA](#) ([TYPEDEF\\_TABLE\\_LIST](#) \*typedef\_table\_list, [AST](#) \*typelist, [AST](#) \*identifier)
- [AST](#) \* [getTYPEDEFfromAST](#) ([TYPEDEF\\_TABLE\\_LIST](#) \*typedef\_table\_list, char \*token, int line)
- void [printTYPEDEF\\_TABLE\\_LIST](#) ([TYPEDEF\\_TABLE\\_LIST](#) \*typedef\_table\_list)
- void [getSTRUCT\\_TABLE\\_DATA](#) ([STRUCT\\_TABLE\\_LIST](#) \*struct\_table\_list, [AST](#) \*ast\_data)
- int [find\\_STRUCT\\_TABLE\\_DATA](#) ([STRUCT\\_TABLE\\_LIST](#) \*struct\_table\_list, CSTLString \*target)
- void [getSTRUCT\\_DATA](#) ([AST](#) \*ast\_data, [STRUCT\\_TABLE\\_LIST](#) \*struct\_table\_data)

- void [getMemberList](#) (VARIABLE\_TABLE\_LIST \*member\_list, [AST](#) \*ast\_data)
- void [getDeclaratorFromAST](#) (char const \*type, [AST](#) \*ast\_data, VARIABLE\_TABLE\_LIST \*member\_list, int enable\_start, int enable\_end, int block\_level, int block\_id, [AST](#) \*declaration\_location\_address)
- void [printSTRUCT\\_TABLE\\_LIST](#) (STRUCT\_TABLE\_LIST \*struct\_table\_list)
- void [getVARIABLE\\_TABLE\\_LIST](#) (VARIABLE\_TABLE\_LIST \*variable\_table\_list, [AST](#) \*ast\_data)
- void [getParameterData](#) (VARIABLE\_TABLE\_LIST \*variable\_table\_list, [AST](#) \*ast\_data, [AST](#) \*enable\_start, [AST](#) \*enable\_end)
- void [getParameterVARIABLE\\_TABLE\\_LIST](#) (VARIABLE\_TABLE\_LIST \*variable\_table\_list, [AST](#) \*ast\_data)
- void [printVARIABLE\\_TABLE\\_LIST](#) (VARIABLE\_TABLE\_LIST \*variable\_table\_list)
- void [getPointerLevelAndArrayLevelFromVARIABLE\\_TABLE](#) (VARIABLE\_TABLE \*variable\_table\_data, int \*output\_pointer\_level, int \*output\_array\_level)
- void [getPointerLevelAndArrayLevel](#) (CSTLString \*target\_identifier, int \*output\_pointer\_level, int \*output\_array\_level)
- [VARIABLE\\_TABLE](#) \* [searchVARIABLE\\_TABLE\\_LIST](#) (IDLIST \*target\_idlist, CSTLString \*target\_string, VARIABLE\_TABLE\_LIST \*variable\_table\_list)
- void [deletePointerAndArraySynbol](#) (CSTLString \*target)
- void [deletePointer](#) (CSTLString \*target)

#### 4.11.1 説明

このファイルは、構文解析によって生成された抽象構文木 (AST) から、変数・typedef テーブル・構造体テーブルを生成させることができます。また、typedef テーブルの生成は、C 言語の構文解析では必須な処理です。

作者

faithnh

#### 4.11.2 型定義

##### 4.11.2.1 typedef struct struct\_table STRUCT\_TABLE

構造体に関する情報であり、プログラム中の構造体の識別するのに用いられる。

##### 4.11.2.2 typedef struct typedef\_table TYPEDEF\_TABLE

型定義に関する情報で、BISON による構文解析時の型定義の認識に用いられる。

##### 4.11.2.3 typedef struct variable\_table VARIABLE\_TABLE

プログラム中の変数に関する情報であり、検証式生成時に変数を識別するのに用いられる。

### 4.11.3 関数

#### 4.11.3.1 void deletePointer ( CString \* *target* )

変数名からポインタを示す記号のみ全て削除する

引数

<i>target</i>	変更対象の変数名
---------------	----------

戻り値

なし

#### 4.11.3.2 void deletePointerAndArraySynbol ( CString \* *target* )

変数名から配列およびポインタを示す記号を全て削除する

引数

<i>target</i>	変更対象の変数名
---------------	----------

戻り値

なし

#### 4.11.3.3 int find\_STRUCT\_TABLE\_DATA ( STRUCT\_TABLE\_LIST \* *struct\_table\_list*, CString \* *target* )

構造体テーブルリストに同じ定義がないかどうかを調べる

引数

<i>struct_table_list</i>	検索対象の構造体テーブルリスト
<i>target</i>	検索する文字列

戻り値

見つけれたら、1 を返し、そうでなければ 0 を返す。

#### 4.11.3.4 void getDeclaratorFromAST ( char const \* *type*, AST \* *ast\_data*, VARIABLE\_TABLE\_LIST \* *member\_list*, int *enable\_start*, int *enable\_end*, int *block\_level*, int *block\_id*, AST \* *declaration\_location\_address* )

指定された AST ノードから、declarator を探し、それを見つけたら指定した型の変数として変数リストに登録する

## 引数

<i>type</i>	指定した型
<i>ast_data</i>	指定された AST ノード
<i>member_list</i>	登録先の変数リスト
<i>enable_start</i>	変数スコープの有効範囲の開始
<i>enable_end</i>	変数スコープの有効範囲の終わり
<i>block_level</i>	この変数のブロックレベル
<i>block_level</i>	ブロックを識別するための識別番号
<i>declaration_location_address</i>	この宣言自体への AST アドレス

指定された AST ノードから、declarator を探し、それを見つけたら指定した型の変数として変数リストに登録する

## 引数

<i>type</i>	指定した型
<i>ast_data</i>	指定された AST ノード
<i>member_list</i>	登録先の変数リスト
<i>enable_start</i>	変数スコープの有効範囲の開始
<i>enable_end</i>	変数スコープの有効範囲の終わり
<i>block_level</i>	この変数のブロックレベル
<i>block_level</i>	ブロックを識別するための識別番号
<i>declaration_location_address</i>	この宣言自体の AST へのアドレス (検証式の生成に必要)

4.11.3.5 void getMemberList ( VARIABLE\_TABLE\_LIST \* *member\_list*, AST \* *ast\_data* )

指定された AST ノードから、メンバリストを生成する

## 引数

<i>member_list</i>	登録対象のメンバリスト
<i>ast_data</i>	指定された AST ノード

## 戻り値

作成された構造体データへのアドレスを返却する

4.11.3.6 void getParameterData ( VARIABLE\_TABLE\_LIST \* *variable\_table\_list*, AST \* *ast\_data*, AST \* *enable\_start*, AST \* *enable\_end* )

関数のパラメータリストを示す AST ノードから、parameter\_declaration を見つけ、そこから変数テーブルのリストに登録させる

## 引数

<i>variable_table_list</i>	変数テーブルのリスト
<i>ast_data</i>	対象の AST ノード
<i>enable_start</i>	有効範囲の開始を示す AST ノードのアドレス
<i>enable_end</i>	有効範囲の終了を示す AST ノードのアドレス

## 戻り値

なし

関数のパラメータリストを示す AST ノードから、`parameter_declaration` を見つけ、そこから変数テーブルのリストに登録させる

## 引数

<i>variable_table_list</i>	変数テーブルのリスト
<i>ast_data</i>	対象の AST ノード
<i>enable_start</i>	有効範囲の開始
<i>enable_end</i>	有効範囲の終了

## 戻り値

なし

**4.11.3.7** `void getParameterVARIABLE_TABLE_LIST ( VARIABLE_TABLE_LIST *  
variable_table_list, AST * ast_data )`

対象の AST ノードから関数を探し、関数内の引数を変数テーブルのリストに登録する

## 引数

<i>variable_table_list</i>	変数テーブルのリスト
<i>ast_data</i>	対象の AST ノード

## 戻り値

なし

**4.11.3.8** `void getPointerLevelAndArrayLevel ( CSTLString * target_identifier, int *  
output_pointer_level, int * output_array_level )`

対象の識別子のポインタの次元および配列の次元を取得する

## 引数

<i>target_ - identifier</i>	対象の識別子
<i>output_ - pointer_ - level</i>	出力されるポインタレベル
<i>output_ - array_level</i>	出力される配列レベル

## 戻り値

なし

**4.11.3.9 void getPointerLevelAndArrayLevelFromVARIABLE\_TABLE ( VARIABLE\_TABLE \* variable\_table\_data, int \* output\_pointer\_level, int \* output\_array\_level )**

変数テーブルから、ポインタの次元および配列の次元を取得する

## 引数

<i>variable_ - table_data</i>	変数テーブルのリスト
<i>output_ - pointer_ - level</i>	出力されるポインタレベル
<i>output_ - array_level</i>	出力される配列レベル

## 戻り値

なし

**4.11.3.10 void getSTRUCT\_DATA ( AST \* ast\_data, STRUCT\_TABLE\_LIST \* struct\_table\_data )**

指定された AST ノードから、構造体データを作成させ、構造体テーブルのリストへ登録させる

## 引数

<i>ast_data</i>	指定された AST ノード
<i>struct_ - table_data</i>	登録先の構造体テーブルリスト

## 戻り値

なし

**4.11.3.11 void getSTRUCT\_TABLE\_DATA ( STRUCT\_TABLE\_LIST \* *struct\_table\_list*, AST \* *ast\_data* )**

指定された AST ノードから、構造体テーブルリストに構造体データを登録させる

引数

<i>struct_table_list</i>	登録先の構造体テーブルリスト
<i>ast_data</i>	指定された AST ノード

戻り値

なし

**4.11.3.12 void getTYPEDEF\_TABLE\_DATA ( TYPEDEF\_TABLE\_LIST \* *typedef\_table\_list*, AST \* *typelist*, AST \* *identifier* )**

指定した AST ノードから参照し、もし typedef 宣言の場合は、typedef テーブルに入れる

引数

<i>typedef_table_list</i>	typedef テーブル
<i>typelist</i>	型リストへの AST ノード
<i>identifier</i>	識別への AST ノード

戻り値

なし

**4.11.3.13 AST\* getTYPEDEFfromAST ( TYPEDEF\_TABLE\_LIST \* *typedef\_table\_list*, char \* *token*, int *line* )**

指定した typedef テーブルのリストから参照し、指定されたトークンに一致するような typedef テーブルデータが存在するかどうか調べる。もし、見つければ、内容が指定されたトークンで、名前が TYPE\_NAME である AST ノードを生成し、それへのアドレスを返す。

引数

<i>typedef_table_list</i>	指定した typedef テーブルのリスト
<i>token</i>	指定されたトークン

戻り値

生成された AST ノードへのアドレスを返す



指定した typedef テーブルのリストから参照し、指定されたトークンに一致するような typedef テーブルデータが存在するかどうか調べる。もし、見つければ、内容が指定されたトークンで、名前が TYPE\_NAME である AST ノードを生成し、それへのアドレスを返す。見つけれなければ、名前が IDENTIFIER である AST ノードを生成し、それへのアドレスを返す。

引数

<i>typedef_</i> - <i>table_list</i>	指定した typedef テーブルのリスト
<i>token</i>	指定されたトークン
<i>line</i>	指定された行数

戻り値

生成された AST ノードへのアドレスを返す

**4.11.3.14** void getVARIABLE\_TABLE\_LIST ( VARIABLE\_TABLE\_LIST \* *variable.table\_list*, AST \* *ast.data* )

対象の AST ノードから変数テーブルのリストを登録する

引数

<i>variable_</i> - <i>table_list</i>	変数テーブルのリスト
<i>ast_data</i>	対象の AST ノード

戻り値

なし

**4.11.3.15** STRUCT\_TABLE\* new\_STRUCT\_TABLE ( int *line*, CSTLString \* *type*, CSTLString \* *struct\_name*, VARIABLE\_TABLE\_LIST \* *member\_list* )

新しい構造体テーブルのデータを生成させる

引数

<i>line</i>	行数
<i>type</i>	型名 (struct か union のいずれか)
<i>struct_name</i>	構造体の名前
<i>member_list</i>	メンバリスト (変数テーブルより)

戻り値

新しく生成された構造体テーブルのデータへのアドレスが返される

新しい構造体テーブルのデータを生成させる

## 引数

<i>line</i>	行数
<i>type</i>	型名
<i>struct_name</i>	構造体名 (struct か union のいずれか)
<i>member_list</i>	メンバリスト (変数テーブルより)

## 戻り値

新しく生成された構造体テーブルのデータへのアドレスが返される

**4.11.3.16** **STRUCT\_TABLE\*** **new\_STRUCT\_TABLE\_with\_char** ( int *line*, char \* *type*, char \* *struct\_name*, VARIABLE\_TABLE\_LIST \* *member\_list* )

新しい構造体テーブルのデータを生成させる (char 文字列対応)

## 引数

<i>line</i>	行数
<i>type</i>	型名 (struct か union のいずれか)
<i>struct_name</i>	構造体の名前
<i>member_list</i>	メンバリスト (変数テーブルより)

## 戻り値

新しく生成された構造体テーブルのデータへのアドレスが返される

**4.11.3.17** **TYPEDEF\_TABLE\*** **new\_TYPEDEF\_TABLE** ( CSTLString \* *target\_type*, CSTLString \* *change\_type* )

新しい typedef テーブルのデータを生成させる

## 引数

<i>target_type</i>	typedef の対象の型
<i>change_type</i>	typedef で割り当てた後の新しい型名

## 戻り値

新しく生成された typedef テーブルのデータへのアドレスが返される

**4.11.3.18** **TYPEDEF\_TABLE\*** **new\_TYPEDEF\_TABLE\_with\_char** ( char \* *target\_type*, char \* *change\_type* )

新しい typedef テーブルのデータを生成させる (char 文字列対応)

## 引数

<i>target_type</i>	typedef の対象の型
<i>change_type</i>	typedef で割り当てた後の新しい型名

戻り値

新しく生成された typedef テーブルのデータへのアドレスが返される

**4.11.3.19 VARIABLE\_TABLE\*** **new\_VARIABLE\_TABLE** ( int *enable\_start*, int *enable\_end*,  
AST \* *declaration\_location\_address*, int *block\_level*, int *block\_id*, IDLIST \* *idlist*,  
CSTLString \* *type*, CSTLString \* *variable\_name*, AST \* *initarizer* )

新しい変数テーブルのデータを生成させる

引数

<i>enable_start</i>	この変数の有効範囲の始まりの行数
<i>enable_end</i>	この変数の有効範囲の終わりの行数
<i>declaration_location_address</i>	この変数を宣言した場所を示す AST のアドレス
<i>block_level</i>	この変数のブロックレベル ( グローバル変数なら 0 とし、関数の中での定義なら 1、その関数内の for 文などのブロック文ないでの宣言なら 2 とする )
<i>block_id</i>	ブロックごとの ID ( 基本的には 0 から始まり、ブロックレベル 2 が 2 回目にくると、1 となる )
<i>idlist</i>	ブロックごとの ID ( これは変数スコープを識別するために使用する )
<i>type</i>	型名
<i>variable_name</i>	変数名
<i>initiarizer</i>	初期定義式への AST ノード

戻り値

新しく生成された変数テーブルのデータへのアドレスが返される

**4.11.3.20 VARIABLE\_TABLE\*** **new\_VARIABLE\_TABLE\_with\_char** ( int *enable\_start*, int *enable\_end*, AST \* *declaration\_location\_address*, int *block\_level*, int *block\_id*,  
IDLIST \* *idlist*, char \* *type*, char \* *variable\_name*, AST \* *initarizer* )

新しい変数テーブルのデータを生成させる (char 文字列対応)

引数

<i>enable_start</i>	この変数の有効範囲の始まりの行数
<i>enable_end</i>	この変数の有効範囲の終わりの行数

<i>declaration_location_address</i>	この変数を宣言した場所を示す AST のアドレス
<i>block_level</i>	この変数のブロックレベル (グローバル変数なら 0 とし、関数の中での定義なら 1、その関数内の for 文などのブロック文ないでの宣言なら 2 とする)
<i>block_id</i>	ブロックごとの ID (基本的には 0 から始まり、ブロックレベル 2 が 2 回目にくると、1 となる)
<i>idlist</i>	ブロックごとの ID (これは変数スコープを識別するために使用する)
<i>type</i>	型名
<i>variable_name</i>	変数名
<i>initializer</i>	初期定義式への AST ノード

#### 戻り値

新しく生成された変数テーブルのデータへのアドレスが返される

#### 4.11.3.21 void printSTRUCT\_TABLE\_LIST ( STRUCT\_TABLE\_LIST \* *struct\_table\_list* )

構造体テーブルのリストの内容を出力させる

#### 引数

<i>struct_table_list</i>	出力対象の構造体テーブルのリスト
--------------------------	------------------

#### 戻り値

なし

#### 4.11.3.22 void printTYPEDEF\_TABLE\_LIST ( TYPEDEF\_TABLE\_LIST \* *typedef\_table\_list* )

typedef テーブルのリストに登録されているものを、次のような形式で出力させる。

target\_type change\_type

#### 引数

<i>typedef_table_list</i>	出力対象の typedef テーブルのリスト
---------------------------	------------------------

#### 戻り値

なし

typedef テーブルのリストに登録されているものを出力させる。

## 引数

<i>typedef_-table_list</i>	出力対象の typedef テーブルのリスト
----------------------------	------------------------

## 戻り値

なし

4.11.3.23 void printVARIABLE\_TABLE\_LIST ( VARIABLE\_TABLE\_LIST \* *variable\_table\_list* )

変数テーブルのリストの内容を出力させる

## 引数

<i>variable_-table_list</i>	出力対象の変数テーブルのリスト
-----------------------------	-----------------

## 戻り値

なし

4.11.3.24 VARIABLE\_TABLE\* searchVARIABLE\_TABLE\_LIST ( IDLIST \* *target\_idlist*, CSTLString \* *target\_string*, VARIABLE\_TABLE\_LIST \* *variable\_table\_list* )

変数テーブルリスト *variable\_table\_list* から、指定した変数スコープの IDLIST *target\_idlist* と *target\_string* に該当ような変数テーブルへのアドレスを返す。

## 引数

<i>target_idlist</i>	指定した変数スコープの IDLIST
<i>target_-string</i>	対象の変数名
<i>variable_-table_list</i>	変数テーブルリスト

## 戻り値

上記の処理から見つけた変数テーブルへのアドレスを返す。見つからなければ NULL を返す。

## 4.12 ANSICInformation/Varidate\_statement.h

これは C 言語プログラム上から、不具合を検証するための検証式や検証用使用する変数などを追加するためのものである

```
#include <cstl/list.h>
```

```
#include "Synbol.h"
#include "PointerArrayControl.h"
#include "MemallocInfo.h"
#include "FreeMemInfo.h"
#include "ForInformation.h"
#include "DivitionInformation.h"
#include "PreProcess.h"
#include "../ProgramSlicing/ProgramSlicingInformation.h"
```

## データ構造

- struct [validate\\_variable](#)
- struct [validate\\_statement](#)

## 型定義

- typedef struct [validate\\_variable](#) [VALIDATE\\_VARIABLE](#)
- typedef struct [validate\\_statement](#) [VALIDATE\\_STATEMENT](#)

## 関数

- [VALIDATE\\_STATEMENT](#) \* [new\\_VALIDATE\\_STATEMENT\\_char](#) (int target\_id, int check\_or\_modify, int used, char \*statement, [AST](#) \*target\_statement)
- [VALIDATE\\_STATEMENT](#) \* [new\\_VALIDATE\\_STATEMENT](#) (int target\_id, int check\_or\_modify, int used, [CSTLString](#) \*statement, [AST](#) \*target\_statement)
- [VALIDATE\\_VARIABLE](#) \* [new\\_VALIDATE\\_VARIABLE](#) (int used, int enable\_start, int enable\_end, int declaration\_location, int block\_level, int block\_id, [CSTLString](#) \*type, [CSTLString](#) \*variable\_name, [CSTLString](#) \*target\_variable\_name, int offset\_level)
- [VALIDATE\\_VARIABLE](#) \* [new\\_VALIDATE\\_VARIABLE\\_with\\_char](#) (int used, int enable\_start, int enable\_end, int declaration\_location, int block\_level, int block\_id, char \*type, char \*variable\_name, char \*target\_variable\_name, int offset\_level)
- void [initVALIDATE\\_STATEMENT\\_flag](#) ([VALIDATE\\_STATEMENT\\_LIST](#) \*validate\_statement\_list)
- void [getValidate\\_Variable](#) ([VARIABLE\\_TABLE\\_LIST](#) \*variable\_table\_list, [VALIDATE\\_VARIABLE\\_LIST](#) \*validate\_variable)
- void [printVALIDATE\\_VARIABLE\\_LIST](#) ([VALIDATE\\_VARIABLE\\_LIST](#) \*validate\_variable\_list)
- void [createValidateStatement](#) ([AST](#) \*root, [FUNCTION\\_INFORMATION\\_LIST](#) \*function\_information\_list, [VARIABLE\\_TABLE\\_LIST](#) \*vtlist, [VALIDATE\\_VARIABLE\\_LIST](#) \*validate\_variable\_list, [VALIDATE\\_STATEMENT\\_LIST](#) \*validate\_statement\_list, [FOR\\_INFORMATION\\_LIST](#) \*for\_information\_list, int undefined\_control\_

- check, int zero\_divition\_check, int array\_unbound\_check, int free\_violation\_check)
- void [getValidateStatementFromInitializer](#) (AST \*root, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ASTPOINTER\_LIST \*ignore\_ast\_list, AST \*target\_expression, int undefined\_control\_check, int zero\_divition\_check, int array\_unbound\_check)
  - void [getValidateStatementFromMallocNumber](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, AST \*call\_function, ARRAY\_OFFSET\_LIST \*right\_array\_offset\_list, MEMALLOC\_INFO \*memalloc\_info)
  - void [getValidateStatementFromForIteration](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, FOR\_INFORMATION\_LIST \*for\_information\_list, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, ASTPOINTER\_LIST \*ignore\_ast\_list, int undefined\_control\_check, int zero\_divition\_check, int array\_unbound\_check, int free\_violation\_check)
  - void [getValidateStatementFromAssignStatement](#) (AST \*root, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ASTPOINTER\_LIST \*ignore\_ast\_list, AST \*target\_expression, int undefined\_control\_check, int zero\_divition\_check, int array\_unbound\_check, int free\_violation\_check)
  - void [getValidateStatementFromPointerOperator](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ARRAY\_OFFSET\_LIST \*left\_array\_offset\_list, ARRAY\_OFFSET\_LIST \*right\_array\_offset\_list, AST \*right\_expression, int a\_op\_flag)
  - void [getValidateStatementFromCallFunction](#) (AST \*root, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ARRAY\_OFFSET\_LIST \*left\_array\_offset\_list, ARRAY\_OFFSET\_LIST \*right\_array\_offset\_list, FUNCTION\_INFORMATION\_LIST \*function\_information\_list)
  - void [getBasisLocationFromAssignmentExpression](#) (CSTLString \*output, ARRAY\_OFFSET\_LIST \*left\_array\_offset\_list, ARRAY\_OFFSET\_LIST \*right\_array\_offset\_list, AST \*right\_expression\_ast, int a\_op\_flag)
  - void [getBasisLocationFromExpression](#) (CSTLString \*output, ARRAY\_OFFSET\_LIST \*array\_offset, AST \*expression\_ast)
  - void [createValidateStatementForMallocAction](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, MEMALLOC\_INFO \*memalloc\_info, ARRAY\_OFFSET\_LIST \*array\_offset\_list, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list)
  - void [createValidateStatementFromIncDecExpr](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ARRAY\_OFFSET\_LIST \*array\_offset\_list)
  - void [createValidateStatementForFreeAction](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, FREEMEMINFO \*freememinfo, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list)
  - void [createValidateStatementFromArrayDefine](#) (VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, VARIABLE\_TABLE\_LIST \*variable\_table\_list, FUNCTION\_INFORMATION\_LIST \*function\_information\_list)
  - void [createVaridateStatementFromPointerDefine](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, VARIABLE\_TABLE\_LIST \*variable\_table\_list, FUNCTION\_INFORMATION\_LIST \*function\_information\_list)

- void [ArrayOffsetToValidateStatement](#) (CSTLString \*output, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, VARIABLE\_TABLE \*variable\_table, OFFSET\_LIST \*offset\_list)
- void [createCheckUnboundAndUndefineOperationCheck](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ARRAY\_OFFSET\_LIST \*array\_offset\_list, int array\_unbound\_check, int undefined\_control\_check)
- void [createViolentFreeOperation](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, FREEMEMINFO \*freememinfo)
- void [createZeroDivitionCheck](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, DIVITION\_INFORMATION\_LIST \*divition\_information\_list)
- int [getNewValidateStatementID](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, AST \*target\_statement)
- void [getLeftAssignmentInfo](#) (AST \*left\_expression, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, ASTPOINTER\_LIST \*ignore\_ast\_list, ARRAY\_OFFSET\_LIST \*array\_offset\_list, AST \*target\_expression, int \*switch\_mode)
- void [getRightAssignmentInfo](#) (AST \*root, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, VARIABLE\_TABLE\_LIST \*vtlist, MEMALLOC\_INFO \*\*memalloc\_info, ARRAY\_OFFSET\_LIST \*array\_offset\_list, ASTPOINTER\_LIST \*ignore\_ast\_list, AST \*target\_statement)
- void [printProgramDataWithValidateStatement](#) (AST \*root, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, FOR\_INFORMATION\_LIST \*for\_information\_list)
- void [fprintfProgramDataWithValidateStatement](#) (FILE \*output, AST \*root, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, FOR\_INFORMATION\_LIST \*for\_information\_list)
- void [fprintfValidateStatement](#) (FILE \*output, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, AST \*target\_ast, int check\_or\_modify, int allow\_output\_used\_statement)
- void [fprintfValidateStatement\\_not\\_assert](#) (FILE \*output, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, AST \*target\_ast, int check\_or\_modify, int allow\_output\_used\_statement)
- void [setValidateVariableFromExprSlicing](#) (VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, EXPR\_SLICING\_LIST \*expr\_slicing\_list)
- void [fprintfProgramDataWithPSIVaridateStatement](#) (FILE \*output, EXPR\_SLICING\_LIST \*expr\_slicing\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, FOR\_INFORMATION\_LIST \*for\_information\_list, AST \*check\_target\_ast)
- void [createValidateStatementAdderFileEachCheck](#) (EXPR\_SLICING\_LIST \*expr\_slicing\_list, VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, VALIDATE\_VARIABLE\_LIST \*validate\_variable\_list, FOR\_INFORMATION\_LIST \*for\_information\_list, INCLUDE\_LIST \*include\_list)
- void [VALIDATE\\_STATEMENT\\_LIST\\_sort\\_ast](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list)
- void [getASTList\\_FromVALIDATE\\_STATEMENT\\_LIST](#) (VALIDATE\_STATEMENT\_LIST \*validate\_statement\_list, ASTPOINTER\_LIST \*ast\_node\_list)



### 4.12.1 説明

これはC言語プログラム上から、不具合を検証するための検証式や検証用に変数などを追加するためのものである

作者

faithnh

### 4.12.2 型定義

#### 4.12.2.1 typedef struct validate\_statement VALIDATE\_STATEMENT

実際に検証式として挿入するための情報

#### 4.12.2.2 typedef struct validate\_variable VALIDATE\_VARIABLE

ポインタや配列変数に対する検証用の変数リストを作成するための構造体

### 4.12.3 関数

#### 4.12.3.1 void ArrayOffsetToValidateStatement ( CSTLString \* output, VALIDATE\_VARIABLE\_LIST \* validate\_variable\_list, VARIABLE\_TABLE \* variable\_table, OFFSET\_LIST \* offset\_list )

配列のオフセットリストを基に、検証式を作成する

引数

<i>output</i>	出力する検証式
<i>VALIDATE_VARIABLE_LIST</i>	検証用変数リスト
<i>variable_table</i>	対象の変数データ
<i>offset_list</i>	オフセットリスト

戻り値

なし

配列のオフセットリストを基に、検証式を作成する

引数

<i>output</i>	出力する検証式
<i>validate_variable_list</i>	検証用変数リスト

<a href="#">variable_table</a>	対象の変数データ
<a href="#">offset_list</a>	オフセットリスト

戻り値  
なし

4.12.3.2 void createCheckUnboundAndUndefineOperationCheck ( VALIDATE\_STATEMENT\_LIST \* validate\_statement\_list, ARRAY\_OFFSET\_LIST \* array\_offset\_list, int array\_unbound\_check, int undefined\_control\_check )

配列やポインタなどのオフセット情報のリスト array\_offset\_list から、配列の範囲外参照のチェックをするための検証式 や、未定義状態で処理をチェックするための検証式を生成し、VALIDATE\_STATEMENT\_LIST へ追加する。

引数

<a href="#">VALIDATE_STATEMENT_LIST</a>	追加先の検証式リスト
<a href="#">array_offset_list</a>	配列やポインタなどのオフセット情報のリスト
<a href="#">array_unbound_check</a>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<a href="#">undefined_control_check</a>	未定義な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない

戻り値  
なし

ArrayOffsetToVaridateStatement の内部処理

引数

<a href="#">output</a>	出力する検証式
<a href="#">validate_variable_list</a>	検証用変数リスト
<a href="#">variable_table</a>	対象の変数データ
<a href="#">offset_list</a>	オフセットリスト
<a href="#">offset_level</a>	今参照しているオフセットレベル

戻り値

なし

**4.12.3.3** void createValidateStatementFromIncDecExpr ( VALIDATE\_STATEMENT\_LIST \*  
validate\_statement\_list, ARRAY\_OFFSET\_LIST \* array\_offset\_list )

指定した配列オフセット array\_offset\_list から、インクリメントおよびデクリメント式を含んでいたら、それに応じて basis\_location に反映させるための検証式を生成し、VALIDATE\_STATEMENT\_LIST に追加する。

引数

VALIDATE_ STATEMENT_ LIST	追加先の検証式
array_ offset_list	配列オフセットリスト

戻り値

なし

指定した配列オフセットリスト array\_offset\_list から、インクリメントおよびデクリメント式を含んでいたら、それに応じて basis\_location に反映させるための検証式を生成し、validate\_statement\_list に追加する。

引数

validate_ statement_ list	追加先の検証式
array_ offset_list	配列オフセットリスト

戻り値

なし

**4.12.3.4** void createValidateStatement ( AST \* root, FUNCTION\_INFORMATION\_LIST \*  
function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, VALIDATE\_VARIABLE\_LIST  
\* validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \* validate\_statement\_list,  
FOR\_INFORMATION\_LIST \* for\_information\_list, int undefined\_control\_check, int  
zero\_divition\_check, int array\_unbound\_check, int free\_violation\_check )

基本的な検証式の生成を行う

## 引数

<i>root</i>	検証式生成対象の AST ノード
<i>function_- informaiton_- list</i>	関数に関する情報のリスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>VALIDATE_- STATEMENT_- LIST</i>	検証用変数リスト
<i>VALIDATE_- VARIABLE_- LIST</i>	取得した検証式が格納するところ
<i>for_- information_- list</i>	for 文に関する情報
<i>undefined_- control_- check</i>	未定義な処理（未定義ポインタの参照など）を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>zero_- divition_- check</i>	0 で割っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>array_- unbound_- check</i>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>free_- violation_- check</i>	メモリ解放関係で不正な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない

## 戻り値

なし

## 基本的な検証式の生成を行う

## 引数

<i>root</i>	検証式生成対象の AST ノード
<i>function_- informaiton_- list</i>	関数に関する情報のリスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>validate_- statement_- list</i>	検証用変数リスト
<i>validate_- variable_list</i>	取得した検証式が格納するところ
<i>for_- information_- list</i>	for 文に関する情報

<i>undefined_-control_-check</i>	未定義な処理（未定義ポインタの参照など）を行っていないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない
<i>zero_-division_-check</i>	0 で割っていないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない
<i>array_-unbound_-check</i>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない
<i>free_-violation_-check</i>	メモリ解放関係で不正な処理を行っていないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない

戻り値

なし

```
4.12.3.5 void createValidateStatementAdderFileEachCheck ( EXPR_SLICING_LIST
* expr_slicing_list, VALIDATE_STATEMENT_LIST * validate_statement_list,
VALIDATE_VARIABLE_LIST * validate_variable_list, FOR_INFORMATION_LIST *
for_information_list, INCLUDE_LIST * include_list )
```

プログラムをチェック式ごとにプログラムスライシングと検証式付加を行ったファイルを生成する

引数

<i>expr_-slicing_list</i>	プログラムスライシング情報
<i>validate_-variable_list</i>	検証用変数リスト
<i>validate_-statement_list</i>	検証式リスト
<i>for_-information_list</i>	for 文に関する情報
<i>include_list</i>	インクルードリスト

```
4.12.3.6 void createValidateStatementForFreeAction ( VALIDATE_STATEMENT_LIST
* validate_statement_list, FREEMEMINFO * freememinfo,
VALIDATE_VARIABLE_LIST * validate_variable_list )
```

メモリ解放関係の情報 *freememinfo* から検証式を生成し、VALIDATE\_STATEMENT\_LIST に追加する

## 引数

<i>variable_-statement_-list</i>	生成先の検証式リスト
<i>freememinfo</i>	メモリ確保関係の情報
<i>VALIDATE_-VARIABLE_-LIST</i>	検証用変数リスト

## 戻り値

なし

メモリ解放関係の情報 *freememinfo* から検証式を生成し、*validate\_statement\_list* に追加する

## 引数

<i>variable_-statement_-list</i>	生成先の検証式リスト
<i>freememinfo</i>	メモリ確保関係の情報
<i>validate_-variable_list</i>	検証用変数リスト

## 戻り値

なし

```
4.12.3.7 void createValidateStatementForMallocAction ( VALIDATE_STATEMENT_LIST
* validate_statement_list, MEMALLOC_INFO * memalloc_info,
ARRAY_OFFSET_LIST * array_offset_list, VALIDATE_VARIABLE_LIST *
validate_variable_list )
```

メモリ確保関係の情報 *memalloc\_info* や配列やポインタのオフセット情報 *array\_offset\_list* から検証式を生成し、*VALIDATE\_STATEMENT\_LIST* に追加する

## 引数

<i>variable_-statement_-list</i>	生成先の検証式リスト
<i>memalloc_-info</i>	メモリ確保関係の情報
<i>array_-offset_list</i>	配列やポインタのオフセット情報

<i>VALIDATE_- VARIABLE_- LIST</i>	検証用変数リスト
---	----------

戻り値

なし

メモリ確保関係の情報 memalloc\_info や配列やポインタのオフセット情報 array\_offset\_list から検証式を生成し、validate\_statement\_list に追加する

引数

<i>variable_- statement_- list</i>	生成先の検証式リスト
<i>memalloc_- info</i>	メモリ確保関係の情報
<i>array_- offset_list</i>	配列やポインタのオフセット情報
<i>validate_- variable_list</i>	検証用変数リスト

戻り値

なし

**4.12.3.8 void createValidateStatementFromArrayDefine ( VALIDATE\_VARIABLE\_LIST \*  
validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \* validate\_statement\_list,  
VARIABLE\_TABLE\_LIST \* variable\_table\_list, FUNCTION\_INFORMATION\_LIST \*  
function\_information\_list )**

変数定義リストで、配列生成時の検証用変数の更新するための検証式を作成し、validate\_statement\_list に追加する また、検証用変数リスト validate\_variable\_list に、配列のオフセットを生成するのに使用する変数 vviterator\_2 ~ vviterator\_n ( n は配列の最大次元数 ) を生成し、追加する。

引数

<i>validate_- statement_- list</i>	追加先の検証式
<i>validate_- table_list</i>	変数テーブルリスト
<i>function_- information_- list</i>	関数定義に関する情報リスト

戻り値  
なし

4.12.3.9 void createVaridateStatementFromPointerDefine ( VALIDATE\_STATEMENT\_LIST  
\* validate\_statement\_list, VARIABLE\_TABLE\_LIST \* variable\_table\_list,  
FUNCTION\_INFORMATION\_LIST \* function\_information\_list )

変数定義リストでポインタ変数に対する検証用変数を更新するための検証式を作成し、validate\_statement\_list に追加する また、検証用変数リスト validate\_variable\_list に、配列のオフセットを生成するのに使用する変数 vviterator\_2 ~ vviterator\_n ( n は配列の最大次元数 ) を生成し、追加する。

引数

validate_statement_list	追加先の検証式
validate_table_list	変数テーブルリスト
function_information_list	関数定義に関する情報リスト

戻り値  
なし

4.12.3.10 void createViolentFreeOperation ( VALIDATE\_STATEMENT\_LIST \*  
validate\_statement\_list, FREEMEMINFO \* freememinfo )

メモリ解放関係の情報 freememinfo から、free 関数に関する違反行為を行っていないかどうかをチェックするための検証式を生成し、VALIDATE\_STATEMENT\_LIST へ追加する。

引数

VALIDATE_STATEMENT_LIST	追加先の検証式リスト
freememinfo	メモリ解放関係の情報

戻り値  
なし

配列やポインタなどのオフセット情報のリスト array\_offset\_list から、配列の範囲外参照のチェックをするための検証式 や、未定義状態で処理をチェックするため



の検証式を生成し、`validate_statement_list` へ追加する。

引数

<code>validate_statement_list</code>	追加先の検証式リスト
<code>array_offset_list</code>	配列やポインタなどのオフセット情報のリスト
<code>array_unbound_check</code>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<code>undefined_control_check</code>	未定義な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない

戻り値

なし メモリ解放関係の情報 `freememinfo` から、`free` 関数に関する違反行為を行っていないかどうかをチェックするための検証式を生成し、`validate_statement_list` へ追加する。

引数

<code>validate_statement_list</code>	追加先の検証式リスト
<code>freememinfo</code>	メモリ解放関係の情報

戻り値

なし

**4.12.3.11** `void createZeroDivisionCheck ( VALIDATE_STATEMENT_LIST * validate_statement_list, DIVISION_INFORMATION_LIST * division_information_list )`

除算および剰余式の情報から、ゼロ除算および剰余になっていないかどうかの検証式を生成する

引数

<code>VALIDATE_STATEMENT_LIST</code>	格納先の検証式リスト
<code>division_information_list</code>	対象の除算および剰余式の情報

戻り値

なし

除算および剰余式の情報から、ゼロ除算および剰余になっていないかどうかの検証式を生成する

引数

<i>validate_statement_list</i>	格納先の検証式リスト
<i>division_information_list</i>	対象の除算および剰余式の情報

戻り値

なし

```
4.12.3.12 void fprintProgramDataWithPSIVaridateStatement ( FILE * output,
EXPR_SLICING_LIST * expr_slicing_list, VALIDATE_STATEMENT_LIST *
validate_statement_list, VALIDATE_VARIABLE_LIST * validate_variable_list,
FOR_INFORMATION_LIST * for_information_list, AST * check_target_ast )
```

プログラムスライシング情報をもとに検証式を追加しながら出力させる

引数

<i>output</i>	出力する先のファイル
<i>expr_slicing_list</i>	プログラムスライシング情報
<i>validate_variable_list</i>	検証用変数リスト
<i>validate_statement_list</i>	検証式リスト
<i>for_information_list</i>	for 文に関する情報
<i>check_target_ast</i>	チェック検証式の対象への AST ノード

```
4.12.3.13 void fprintfProgramDataWithValidateStatement ( FILE * output, AST * root,
VALIDATE_VARIABLE_LIST * validate_variable_list, VALIDATE_STATEMENT_LIST *
validate_statement_list, FOR_INFORMATION_LIST * for_information_list )
```

検証式リストや検証用変数をもとにプログラムデータを生成し、指定したファイル *output* に出力する。

引数

<i>output</i>	出力先のファイル構造体
<i>root</i>	プログラムへの AST ノード
<i>VALIDATE_VARIABLE_LIST</i>	検証用変数リスト
<i>VALIDATE_STATEMENT_LIST</i>	検証式リスト
<i>for_information_list</i>	for 文に関する情報のリスト

戻り値

なし

検証式リストや検証用変数をもとにプログラムデータを生成し、指定したファイル *output* に出力する。

引数

<i>output</i>	出力先のファイル構造体
<i>root</i>	プログラムへの AST ノード
<i>validate_variable_list</i>	検証用変数リスト
<i>validate_statement_list</i>	検証式リスト
<i>for_information_list</i>	for 文に関する情報のリスト

戻り値

なし

```
4.12.3.14 void fprintfValidateStatement ( FILE * output, VALIDATE_STATEMENT_LIST
        * validate_statement_list, AST * target_ast, int check_or_modify, int
        allow_output_used_statement )
```

式に対応する検証式を出力させる

引数

<i>output</i>	出力先のファイル構造体
<i>VALIDATE_STATEMENT_LIST</i>	出力対象の検証式リスト
<i>target_ast</i>	対象の AST ノード
<i>check_or_modify</i>	検証式をチェックするタイプか、プログラムを元に編集するタイプかを判断するフラグ。0：チェックするタイプ、1：編集するタイプ
<i>allow_output_used_statement</i>	使用済みの検証式も含めて出力するかどうかのフラグ 0：出力しない 1：出力する

戻り値

なし

式に対応する検証式を出力させる

引数

<i>output</i>	出力先のファイル構造体
<i>validate_statement_list</i>	出力対象の検証式リスト
<i>target_ast</i>	対象の AST ノード
<i>check_or_modify</i>	検証式をチェックするタイプか、プログラムを元に編集するタイプかを判断するフラグ。0：チェックするタイプ、1：編集するタイプ
<i>allow_output_used_statement</i>	使用済みの検証式も含めて出力するかどうかのフラグ 0：出力しない 1：出力する

戻り値

なし

```
4.12.3.15 void fprintfValidateStatement_not_assert ( FILE * output, VALIDATE_STATEMENT_LIST
            * validate_statement_list, AST * target_ast, int check_or_modify, int
            allow_output_used_statement )
```

式に対応する検証式を `assert(0);` を削除したうえ出力させる。

引数

<i>output</i>	出力先のファイル構造体
<i>validate_statement_list</i>	出力対象の検証式リスト
<i>target_ast</i>	対象の AST ノード
<i>check_or_modify</i>	検証式をチェックするタイプか、プログラムを元に編集するタイプかを判断するフラグ。0 : チェックするタイプ、1 : 編集するタイプ
<i>allow_output_used_statement</i>	使用済みの検証式も含めて出力するかどうかのフラグ 0 : 出力しない 1 : 出力する

戻り値

なし

```
4.12.3.16 void getASTList_FromVALIDATE_STATEMENT_LIST ( VALIDATE_STATEMENT_LIST *
            validate_statement_list, ASTPOINTER_LIST * ast_node_list )
```

検証式リストのチェック式から、AST ノードを取り出し、AST リストとしてまとめる

引数

<i>validate_statement_list</i>	取り出し先の検証式リスト
<i>ast_node_list</i>	まとめる先の AST ノードリスト

戻り値

なし

```
4.12.3.17 void getBasisLocationFromAssignmentExpression ( CSTLString * output,
            ARRAY_OFFSET_LIST * left_array_offset_list, ARRAY_OFFSET_LIST *
            right_array_offset_list, AST * right_expression_ast, int a_op_flag )
```

ポインタ演算式に対して、ポインタ演算における基本的な位置の式を文字列として求め、`output` に入れる。

## 引数

<i>output</i>	出力先の CSTL 文字列
<i>left_array_offset_list</i>	左辺値の配列オフセットリスト
<i>right_array_offset_list</i>	右辺式の配列オフセットリスト
<i>right_expression_ast</i>	右辺式への AST アドレス
<i>a_op_flag</i>	代入演算子が何かを示すフラグ      0:=,1:+=,2:-=,3:*=,4:/=,5:=,6:<=<=,7:>>=,8:&=,9: =,10:^=

## 戻り値

なし

#### 4.12.3.18 void getBasisLocationFromExpression ( CSTLString \* output, ARRAY\_OFFSET \* array\_offset, AST \* expression\_ast )

指定した式 *expression\_ast* から、ポインタ演算における基本的な位置の式を文字列として求め、*output* に入れる。このとき、*array\_offset* を見つけたらそれに該当する式を 0 に変換する。

## 引数

<i>output</i>	出力先の CSTL 文字列
<i>array_offset</i>	指定した識別子の配列オフセット
<i>expression_ast</i>	指定した式への AST アドレス

#### 4.12.3.19 void getLeftAssignmentInfo ( AST \* left\_expression, FUNCTION\_INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, ASTPOINTER\_LIST \* ignore\_ast\_list, ARRAY\_OFFSET\_LIST \* array\_offset\_list, AST \* target\_expression, int \* switch\_mode )

代入式の左辺値について、検証式に必要な情報を取得する

## 引数

<i>left_expression</i>	左辺値に関する AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	検証対象の式をマークするための変数リスト
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト

<i>array_offset_list</i>	各ポインタおよび配列ごとのオフセットのリスト
<i>target_expression</i>	この左辺式の上位に位置する AST ノード
<i>switch_mode</i>	直接アクセスおよび配列アクセスを探すか、IDENTIFIER を探すかどうかのスイッチフラグ 0 : 両方さがす 1 : direct_ref や array_access のみ探す

戻り値

なし

**4.12.3.20** `int getNewValidateStatementID ( VALIDATE_STATEMENT_LIST *  
validate_statement_list, AST * target_statement )`

検証式リスト VALIDATE\_STATEMENT\_LIST から、target\_statement と同じ AST のアドレスを持ったものを探し出し、それを基に重複しないようにするための新しい検証式の識別番号を取得する。新しい検証式を作るにはこの関数から新しい識別番号を取得すること。

引数

<i>VALIDATE_STATEMENT_LIST</i>	対象の検証式リスト
<i>target_statement</i>	対象の検証式から確認するための AST のアドレス

戻り値

新しい識別番号を出力する。すでに同じ AST のアドレスを持っている検証式がなければ 0 を返す。

検証式リスト validate\_statement\_list から、target\_statement と同じ AST のアドレスを持ったものを探し出し、それを基に重複しないようにするための新しい検証式の識別番号を取得する。新しい検証式を作るにはこの関数から新しい識別番号を取得すること。

引数

<i>validate_statement_list</i>	対象の検証式リスト
<i>target_statement</i>	対象の検証式から確認するための AST のアドレス

## 戻り値

新しい識別番号を出力する。すでに同じ AST のアドレスを持っている検証式がなければ 0 を返す。

**4.12.3.21** void getRightAssignmentInfo ( AST \* root, FUNCTION\_INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, MEMALLOC\_INFO \*\* memalloc\_info, ARRAY\_OFFSET\_LIST \* array\_offset\_list, ASTPOINTER\_LIST \* ignore\_ast\_list, AST \* target\_statement )

代入式の右辺式について、検証式に必要な情報を取得する

## 引数

<i>root</i>	右辺式に関する AST ノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	メモリ確保情報を取得するのに必要なプログラム変数リスト
<i>memalloc_info</i>	malloc 関係の情報が出力される
<i>array_offset_list</i>	左辺式上にあるポインタ参照に対するオフセットリスト
<i>ignore_ast_list</i>	同じ位置のポインタが来ても無視するためのリスト
<i>target_statement</i>	この計算式を属している AST ノードへのアドレス ( 基本的に expression_statement であるノードが入る )

## 戻り値

なし

**4.12.3.22** void getValidate\_Variable ( VARIABLE\_TABLE\_LIST \* variable\_table\_list, VALIDATE\_VARIABLE\_LIST \* validate\_variable )

プログラムの変数リストをもとにプログラムの検証用の変数を設定する

## 引数

<i>variable_table_list</i>	プログラムの変数リスト
<i>struct_table_list</i>	構造体リスト
<i>typedef_table_list</i>	typedef リスト
<i>validate_variable</i>	検証用の変数リスト



戻り値

なし

**4.12.3.23** void getValidateStatementFromAssignStatement ( AST \* root,  
FUNCTION\_INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST  
\* vtlist, VALIDATE\_VARIABLE\_LIST \* validate\_variable\_list,  
VALIDATE\_STATEMENT\_LIST \* validate\_statement\_list, ASTPOINTER\_LIST \*  
ignore\_ast\_list, AST \* target\_expression, int undefined\_control\_check, int  
zero\_divition\_check, int array\_unbound\_check, int free\_violation\_check )

指定した AST ノード root から、assign\_expression を探しだし、そこから VARIDATE\_STATEMENT に関する情報を取得する。

引数

<i>root</i>	指定したノード
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	対象の変数リスト
<i>validate_variable_list</i>	検証用変数リスト
<i>validate_statement_list</i>	取得した検証式が格納するところ
<i>ignore_ast_list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>target_expression</i>	assign_expression が属している expression_statement
<i>undefined_control_check</i>	未定義な処理（未定義ポインタの参照など）を行っていないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない
<i>zero_divition_check</i>	0 で割っていないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない
<i>array_unbound_check</i>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない
<i>free_violation_check</i>	メモリ解放関係で不正な処理を行っていないかどうかを検証するための式を生成するかどうか 1：生成する 0：生成しない

戻り値

なし

```

4.12.3.24 void getValidateStatementFromCallFunction ( AST * root,
VALIDATE_STATEMENT_LIST * validate_statement_list, ARRAY_OFFSET_LIST
* left_array_offset_list, ARRAY_OFFSET_LIST * right_array_offset_list,
FUNCTION_INFORMATION_LIST * function_information_list )

```

式から、関数呼出を探しだし、関数呼出に対する検証式を追加する

引数

<i>root</i>	探索対象の AST ノード
<i>validate_statement_list</i>	検証式リスト
<i>left_array_offset_list</i>	左辺値の配列オフセットリスト
<i>right_array_offset_list</i>	右辺式の配列オフセットリスト
<i>function_information_list</i>	関数に関する情報リスト

戻り値

なし

```

4.12.3.25 void getValidateStatementFromForIteration ( VALIDATE_STATEMENT_LIST *
validate_statement_list, FOR_INFORMATION_LIST * for_information_list,
FUNCTION_INFORMATION_LIST * function_information_list, VARIABLE_TABLE_LIST
* vtlist, VALIDATE_VARIABLE_LIST * validate_variable_list, ASTPOINTER_LIST
* ignore_ast_list, int undefined_control_check, int zero_divition_check, int
array_unbound_check, int free_violation_check )

```

for 文の末尾の情報から、検証式を取得し、検証式リストに入れる

引数

<i>validate_statement_list</i>	取得した検証式が格納するところ
<i>for_information_list</i>	for 文に関する情報のリスト
<i>function_information_list</i>	関数に関する情報のリスト
<i>vtlist</i>	対象の変数リスト
<i>validate_variable_list</i>	検証用変数リスト

<i>ignore_ast_-list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>undefined_-control_-check</i>	未定義な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>zero_-division_-check</i>	0 で割っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>array_-unbound_-check</i>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>free_-violation_-check</i>	メモリ解放関係で不正な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない

戻り値

なし

**4.12.3.26** void getValidateStatementFromInitializer ( AST \* root, FUNCTION\_-INFORMATION\_LIST \* function\_information\_list, VARIABLE\_TABLE\_LIST \* vtlist, VALIDATE\_VARIABLE\_LIST \* validate\_variable\_list, VALIDATE\_STATEMENT\_LIST \* validate\_statement\_list, ASTPOINTER\_LIST \* ignore\_ast\_list, AST \* target\_expression, int undefined\_control\_check, int zero\_divition\_check, int array\_unbound\_check )

指定した AST ノード root から、init\_declarator を探しだし、そこから VARIDATE\_-STATEMENT に関する情報を取得する。

引数

<i>root</i>	指定したノード
<i>function_-information_-list</i>	関数に関する情報のリスト
<i>vtlist</i>	対象の変数リスト
<i>VALIDATE_-VARIABLE_-LIST</i>	検証用変数リスト
<i>VALIDATE_-STATEMENT_-LIST</i>	取得した検証式が格納するところ
<i>ignore_ast_-list</i>	ポインタでの位置が検証済みである、IDENTIFIER を無視するための AST のアドレスリスト
<i>target_-expression</i>	assign_expression が属している expression_statement

<i>undefined_-control_-check</i>	未定義な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>zero_-division_-check</i>	0 で割っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>array_-unbound_-check</i>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない

指定した AST ノード *root* から、*init\_declarator* を探しだし、そこから *VARIDATE\_STATEMENT* に関する情報を取得する。

#### 引数

<i>root</i>	指定したノード
<i>function_-information_-list</i>	関数に関する情報のリスト
<i>vtlist</i>	対象の変数リスト
<i>validate_-variable_-list</i>	検証用変数リスト
<i>validate_-statement_-list</i>	取得した検証式が格納するところ
<i>ignore_ast_-list</i>	ポインタでの位置が検証済みである、 <i>IDENTIFIER</i> を無視するための AST のアドレスリスト
<i>target_-expression</i>	<i>assign_expression</i> が属している <i>expression_statement</i>
<i>undefined_-control_-check</i>	未定義な処理を行っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>zero_-division_-check</i>	0 で割っていないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない
<i>array_-unbound_-check</i>	配列が範囲外を参照していないかどうかを検証するための式を生成するかどうか 1 : 生成する 0 : 生成しない

**4.12.3.27 void getValidateStatementFromMallocNumber ( VALIDATE\_STATEMENT\_LIST \* *validate\_statement\_list*, AST \* *call\_function*, ARRAY\_OFFSET\_LIST \* *right\_array\_offset\_list*, MEMALLOC\_INFO \* *memalloc\_info* )**

malloc 用識別番号を付加するための関数に変換するための検証式を追加させる

#### 引数

<i>VALIDATE_- STATEMENT_ LIST</i>	追加先の検証式リスト
<i>call_- function</i>	関数呼び出しに対するノード
<i>right_- array_- offset_list</i>	左辺値に関する配列オフセットリスト
<i>memalloc_- info</i>	メモリ確保情報

戻り値

なし

malloc 用識別番号を付加するための関数に変換するための検証式を追加させる

引数

<i>validate_- statement_- list</i>	追加先の検証式リスト
<i>call_- function</i>	関数呼び出しに対するノード
<i>right_- array_- offset_list</i>	左辺値に関する配列オフセットリスト
<i>memalloc_- info</i>	メモリ確保情報

戻り値

なし

```
4.12.3.28 void getValidateStatementFromPointerOperator ( VALIDATE_STATEMENT_LIST
* validate_statement_list, ARRAY_OFFSET_LIST * left_array_offset_list,
ARRAY_OFFSET_LIST * right_array_offset_list, AST * right_expression, int a_op_flag
)
```

ポインタ演算式後の内容を検証用変数に反映するための検証式を追加する

引数

<i>VALIDATE_- STATEMENT_ LIST</i>	検証式リスト
---	--------

<i>left_array_offset_list</i>	左辺値の配列オフセットリスト
<i>right_array_offset_list</i>	右辺式の配列オフセットリスト
<i>right_expression</i>	右辺式へのASTアドレス
<i>a_op_flag</i>	代入演算子が何かを示すフラグ0: =, 1: +=, 2: -=, 3: *=, 4: /=, 5: =, 6: <=, 7: >=, 8: &=, 9:  =, 10: ^=

戻り値  
なし

ポインタ演算式後の内容を検証用変数に反映するための検証式を追加する

引数

<i>validate_statement_list</i>	検証式リスト
<i>left_array_offset_list</i>	左辺値の配列オフセットリスト
<i>right_array_offset_list</i>	右辺式の配列オフセットリスト
<i>right_expression</i>	右辺式へのASTアドレス
<i>a_op_flag</i>	代入演算子が何かを示すフラグ0: =, 1: +=, 2: -=, 3: *=, 4: /=, 5: =, 6: <=, 7: >=, 8: &=, 9:  =, 10: ^=

戻り値  
なし

4.12.3.29 void initVALIDATE\_STATEMENT\_flag ( VALIDATE\_STATEMENT\_LIST \* validate\_statement\_list )

検証式リストの使用フラグを未使用状態に初期化する

引数

<i>validate_statement_list</i>	* 初期化対象の検証式リスト
--------------------------------	----------------

戻り値

なし

**4.12.3.30** `VALIDATE_STATEMENT* new_VALIDATE_STATEMENT ( int target_id, int check_or_modify, int used, CSTLString * statement, AST * target_statement )`

実際に検証式として挿入するための情報を生成する

引数

<i>target_id</i>	この検証式の識別 ID(どの順序でこの検証式を入れていくかを確認するための ID)
<i>check_or_modify</i>	検証式をチェックするタイプか、プログラムを元に編集するタイプかを判断するフラグ。 0 : チェックするタイプ、 1 : 編集するタイプ
<i>used</i>	この検証式は使用しているかどうかのフラグ 1:使用 0:未使用
<i>statement</i>	この検証式の内容
<i>target_statement</i>	この検証式のターゲットとなる AST ノードへのアドレス

戻り値

実際に検証式として挿入するための情報へのアドレスを返す

**4.12.3.31** `VALIDATE_STATEMENT* new_VALIDATE_STATEMENT_char ( int target_id, int check_or_modify, int used, char * statement, AST * target_statement )`

実際に検証式として挿入するための情報を生成する

引数

<i>target_id</i>	この検証式の識別 ID(どの順序でこの検証式を入れていくかを確認するための ID)
<i>check_or_modify</i>	検証式をチェックするタイプか、プログラムを元に編集するタイプかを判断するフラグ。 0 : チェックするタイプ、 1 : 編集するタイプ
<i>used</i>	この検証式は使用しているかどうかのフラグ 1:使用 0:未使用
<i>statement</i>	この検証式の内容
<i>target_statement</i>	この検証式のターゲットとなる AST ノードへのアドレス

戻り値

実際に検証式として挿入するための情報へのアドレスを返す

**4.12.3.32 VALIDATE\_VARIABLE\*** `new_VALIDATE_VARIABLE ( int used, int enable_start, int enable_end, int declaration_location, int block_level, int block_id, CSTLString * type, CSTLString * variable_name, CSTLString * target_variable_name, int offset_level )`

新しい検証用変数テーブルのデータを生成させる

引数

<i>used</i>	この検証用変数テーブルを使用したかどうか
<i>enable_start</i>	この変数の有効範囲の始まりの行数
<i>enable_end</i>	この変数の有効範囲の終わりの行数
<i>declaration_location</i>	この変数を宣言した場所の行数
<i>block_level</i>	この変数のブロックレベル (グローバル変数なら 0 とし、関数の中での定義なら 1、その関数内の for 文などのブロック文ないでの宣言なら 2 とする)
<i>block_id</i>	ブロックごとの ID (基本的には 0 から始まり、ブロックレベル 2 が 2 回目になると、1 となる)
<i>type</i>	型名
<i>variable_name</i>	変数名
<i>target_variable_name</i>	検証対象の変数名
<i>offset_level</i>	この変数の配列やポインタの次元レベル

戻り値

新しく生成された検証用変数のデータへのアドレスが返される

**4.12.3.33 VALIDATE\_VARIABLE\*** `new_VALIDATE_VARIABLE_with_char ( int used, int enable_start, int enable_end, int declaration_location, int block_level, int block_id, char * type, char * variable_name, char * target_variable_name, int offset_level )`

新しい検証用変数テーブルのデータを生成させる (char 対応版)

引数

<i>used</i>	この検証用変数テーブルを使用したかどうか
<i>enable_start</i>	この変数の有効範囲の始まりの行数
<i>enable_end</i>	この変数の有効範囲の終わりの行数
<i>declaration_location</i>	この変数を宣言した場所の行数
<i>block_level</i>	この変数のブロックレベル (グローバル変数なら 0 とし、関数の中での定義なら 1、その関数内の for 文などのブロック文ないでの宣言なら 2 とする)



<i>block_id</i>	ブロックごとの ID ( 基本的には 0 から始まり、ブロックレベル 2 が 2 回目にくると、1 となる )
<i>type</i>	型名
<i>variable_name</i>	変数名
<i>target_variable_name</i>	検証対象の変数名
<i>offset_level</i>	この変数の配列やポインタの次元レベル

戻り値

新しく生成された検証用変数のデータへのアドレスが返される

```
4.12.3.34 void printProgramDataWithValidateStatement ( AST * root,
VALIDATE_VARIABLE_LIST * validate_variable_list, VALIDATE_STATEMENT_LIST *
validate_statement_list, FOR_INFORMATION_LIST * for_information_list )
```

検証式リストとともにプログラムデータを出力する

引数

<i>root</i>	プログラムへの AST ノード
<i>VALIDATE_VARIABLE_LIST</i>	検証用変数リスト
<i>VALIDATE_STATEMENT_LIST</i>	検証式リスト
<i>for_information_list</i>	for 文に関する情報のリスト

戻り値

なし

検証式リストとともにプログラムデータを出力する

引数

<i>root</i>	プログラムへの AST ノード
<i>validate_variable_list</i>	検証用変数リスト
<i>validate_statement_list</i>	検証式リスト

<i>for_- information_ list</i>	for 文に関する情報のリスト
--	-----------------

戻り値  
なし

4.12.3.35 void printVALIDATE\_VARIABLE\_LIST ( VALIDATE\_VARIABLE\_LIST \*  
validate\_variable\_list )

検証用変数テーブルのリストの内容を出力させる

引数

<i>variable_- table_list</i>	出力対象の検証用変数テーブルのリスト
----------------------------------	--------------------

戻り値  
なし

4.12.3.36 void setValidateVariableFromExprSlicing ( VALIDATE\_VARIABLE\_LIST \*  
validate\_variable\_list, EXPR\_SLICING\_LIST \* expr\_slicing\_list )

プログラムスライシング情報の変数定義をもとに、検証用変数リストの出力を設定する

引数

<i>validate_- variable_list</i>	検証用変数リスト
<i>expr_- slicing_list</i>	プログラムスライシング情報のリスト

戻り値  
なし

4.12.3.37 void VALIDATE\_STATEMENT\_LIST\_sort\_ast ( VALIDATE\_STATEMENT\_LIST \*  
validate\_statement\_list )

検証式リストを AST ノードごとにソートする

引数

<i>validate_- statement_- list</i>	ソート対象の検証式リスト
--	--------------

戻り値

なし

## 4.13 Library/CharStringExtend.h

このファイルは、string.h にないような処理を実現させるためのものです。具体的には文字列からある場所からある場所までを抽出させたりといったことができます。

### 関数

- int [str\\_extract](#) (char \*dest, char \*source, int start, int str\_length)
- int [isExpression](#) (char \*source)

#### 4.13.1 説明

このファイルは、string.h にないような処理を実現させるためのものです。具体的には文字列からある場所からある場所までを抽出させたりといったことができます。

作者

faithnh

#### 4.13.2 関数

##### 4.13.2.1 int isExpression ( char \* source )

文字列 source は式であるかどうかを調べる

引数

<i>source</i>	対象の文字列
---------------	--------

戻り値

式である場合は 1、そうでない場合は 0 を返す

文字列 source は式または識別子であるかどうかを調べる

## 引数

<i>source</i>	対象の文字列
---------------	--------

## 戻り値

式または識別子である場合は 1、そうでない場合は 0 を返す

4.13.2.2 int str\_extract ( char \* *dest*, char \* *source*, int *start*, int *str\_length* )

文字列 *source* から開始文字数 *start* から指定された文字数 *strlen* 分の文字列を抽出し、その結果を文字列 *dest* へ入れる。また終端子も付くので、抜き出した文字 + 1 個分の文字列が必要

## 引数

<i>dest</i>	抽出結果を入れる文字列
<i>source</i>	抽出対象の文字列
<i>start</i>	抽出の開始位置
<i>strlen</i>	抽出する文字数

## 戻り値

成功したかどうかのフラグ 成功した場合は 1 途中で抽出に失敗した場合は 0 を返す

## 4.14 Library/CSTLString.h

このファイルは、CSTL ライブラリを用いた文字列型 CSTLString を生成したり、それに関する操作を行います。

```
#include <cstl/string.h>
```

## 関数

- void [CSTLString\\_replace\\_string](#) (CSTLString \*target, char \*search, char \*replace)
- int [CSTLString\\_compare\\_with\\_char](#) (CSTLString \*target1, char \*target2)
- int [CSTLString\\_delete\\_tail\\_str](#) (CSTLString \*target1, char \*del\_str)
- void [CSTLString\\_printf](#) (CSTLString \*target, int add\_flag, char \*source,...)
- void [CSTLString\\_ltrim](#) (CSTLString \*target)

## 4.14.1 説明

このファイルは、CSTL ライブラリを用いた文字列型 CSTLString を生成したり、それに関する操作を行います。

作者

faithnh

#### 4.14.2 関数

##### 4.14.2.1 int CSTLString\_compare\_with\_char ( CSTLString \* *target1*, char \* *target2* )

指定した CSTL 文字列 *target1* と比較対象の普通の文字列 *target2* と比較する。

引数

<i>target1</i>	指定した CSTL 文字列
<i>target2</i>	比較対象の普通の文字列

戻り値

比較した結果を返す、一致した場合は 0、そうでない場合は 0 以外の値を返す

##### 4.14.2.2 int CSTLString\_delete\_tail\_str ( CSTLString \* *target1*, char \* *del\_str* )

指定した CSTL 文字列 *target1* から、指定した文字列 *del\_str* の最初に出現する箇所以降を全て削除する。

引数

<i>target1</i>	指定した CSTL 文字列
<i>del_str</i>	削除する場所を指定するための文字列

戻り値

成功したかどうか否かを返す。成功した場合は 1、そうでない場合は 0 を返す。

##### 4.14.2.3 void CSTLString\_ltrim ( CSTLString \* *target* )

指定した文字列の前の半角スペース・改行文字・タブを削除する

引数

<i>target</i>	指定した文字列
---------------	---------

##### 4.14.2.4 void CSTLString\_printf ( CSTLString \* *target*, int *add\_flag*, char \* *source*, ... )

指定した文字列 *source* を簡易版の書式フォーマット形式で指定した CSTL 文字列 *target* に入力する。簡易版なので、d、f、s、d、%にしか対応していない。

は可能。また、*add\_flag* を指定することで、追加・代入の選択もできる。

## 引数

<i>target</i>	入力先の指定した CSTL 文字列（あらかじめ初期化する必要あり）
<i>add_flag</i>	追加挿入するかどうかのフラグ 1：追加 0：代入
<i>source</i>	指定した文字列（書式フォーマット形式）
...	書式に対応した任意の引数

## 戻り値

なし

4.14.2.5 void CSTLString\_replace\_string ( CSTLString \* *target*, char \* *search*, char \* *replace* )

指定した CSTL 文字列 *target* に対して、置換対象の文字列 *search* から置換したい文字列 *replace* に置換を行う

## 引数

<i>target</i>	指定した CSTL 文字列
<i>search</i>	置換対象の文字列
<i>teplace</i>	置換したい文字列

## 戻り値

なし

## 4.15 Library/FlagDatabase.h

このファイルは、本体に関するフラグ設定を格納するためのものです。たとえば、xml として出力するのかなどといった設定はこちらに入ります。

## 関数

- int [getFlagDatabase](#) (int argc, char \*argv[])
- int [isXmlMode](#) (int flag\_database)
- int [isHelpMode](#) (int flag\_database)
- int [isArrayUnboundCheckMode](#) (int flag\_database)
- int [isUndefineControlCheckMode](#) (int flag\_database)
- int [isZeroDivitionCheckMode](#) (int flag\_database)
- int [isFreeViolationCheckMode](#) (int flag\_database)
- int [isProgramSlicingMode](#) (int flag\_database)

## 4.15.1 説明

このファイルは、本体に関するフラグ設定を格納するためのものです。たとえば、xml として出力するのかなどといった設定はこちらに入ります。

作者

faithnh

## 4.15.2 関数

### 4.15.2.1 int getFlagDatabase ( int argc, char \* argv[] )

プログラムの引数からフラグデータベースを取得する。ここで不正にフラグが設定されていた場合（存在しないフラグがある・フラグが二重に定義されている場合）エラーを返し、強制終了される。

引数

<i>argc</i>	引数フラグの数
<i>argv</i>	引数フラグの文字列

戻り値

引数から取得したフラグを取得する

### 4.15.2.2 int isArrayUnboundCheckMode ( int flag\_database )

フラグデータベースから、ARRAY\_UNBOUND\_CHECK\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

ARRAY\_UNBOUND\_CHECK\_MODE が含まれていたら 1 を返し、そうでなければ 0 を返す

### 4.15.2.3 int isFreeViolationCheckMode ( int flag\_database )

フラグデータベースから、FREE\_VIOLATION\_CHECK\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

FREE\_VIOLATION\_CHECK\_MODE が含まれていたら 1 を返し、そうでなけ

れば 0 を返す

#### 4.15.2.4 int isHelpMode ( int *flag\_database* )

フラグデータベースから、HELP\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

HELP\_MODE が含まれていたら 1 を返し、そうでなければ 0 を返す

#### 4.15.2.5 int isProgramSlicingMode ( int *flag\_database* )

フラグデータベースから、PROGRAM\_SLICING\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

PROGRAM\_SLICING\_MODE が含まれていたら 1 を返し、そうでなければ 0 を返す

#### 4.15.2.6 int isUndefineControlCheckMode ( int *flag\_database* )

フラグデータベースから、UNDEFINE\_CONTROL\_CHECK\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

UNDEFINE\_CONTROL\_CHECK\_MODE が含まれていたら 1 を返し、そうでなければ 0 を返す



4.15.2.7 int isXmlMode ( int *flag\_database* )

フラグデータベースから、XML\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

XML\_MODE が含まれていたなら 1 を返し、そうでなければ 0 を返す

4.15.2.8 int isZeroDivitionCheckMode ( int *flag\_database* )

フラグデータベースから、ZERO\_DIVITION\_CHECK\_MODE が含まれているかどうか確認する

引数

<i>flag_database</i>	フラグデータベース
----------------------	-----------

戻り値

ZERO\_DIVITION\_CHECK\_MODE が含まれていたなら 1 を返し、そうでなければ 0 を返す

## 4.16 Library/IdList.h

このファイルは、この変数は C 言語プログラム上のどのブロックに宣言しているかどうかを確認するための情報を生成します。

```
#include "Stack_int.h"
#include <cstl/list.h>
```

関数

- int [IDLIST\\_compare\\_with](#) (IDLIST \*target1, IDLIST \*target2)
- void [SET\\_STACK\\_INTToIDLIST](#) (IDLIST \*dest, STACK\_INT \*source, int num)
- void [printIDLIST](#) (IDLIST \*idlist, int new\_line\_flag)

### 4.16.1 説明

このファイルは、この変数は C 言語プログラム上のどのブロックに宣言しているかどうかを確認するための情報を生成します。

作者

faithnh

#### 4.16.2 関数

##### 4.16.2.1 int IDLIST\_compare\_with ( IDLIST \* *target1*, IDLIST \* *target2* )

二つの IDLIST *target1*, *target2* を比較し、次の状態であれば、1 を返す。そうでなければ 0 を返す。 *target2* の一番最初のリストの値が 0 である。 *target2* のリスト内の値が全て、*target1* のリスト内の値と一致する場合

引数

<i>target1</i>	比較対象の IDLIST 1
<i>target2</i>	比較対象の IDLIST 2

戻り値

比較して上記道理になると 1 を返し、そうでなければ、0 を返す。

##### 4.16.2.2 void printIDLIST ( IDLIST \* *idlist*, int *new\_line\_flag* )

IDLIST の内容を出力する

引数

<i>idlist</i>	出力対象の IDLIST
<i>new_line_flag</i>	改行するかどうかのフラグ 1 : 改行する 0 : 改行しない

戻り値

なし

##### 4.16.2.3 void SET\_STACK\_INTtoIDLIST ( IDLIST \* *dest*, STACK\_INT \* *source*, int *num* )

整数スタックの内容 *source* を入れたい対象の IDLIST *dest* へ入れる。

引数

<i>dest</i>	対象の IDLIST
<i>source</i>	入れる整数スタック内容
<i>num</i>	整数スタックに入れる数

戻り値

なし

## 4.17 Library/Stack\_int.h

このファイルは整数スタックに関する操作をするためのものです。このファイルで使用される主な用途として、C 言語ソースファイルにおけるブロックレベルを設定するのに使用されます。

```
#include <cstl/vector.h>
```

### 関数

- int [STACK\\_INT\\_at\\_and\\_alloc](#) (STACK\_INT \*stack\_int, int index)
- int [STACK\\_INT\\_inclement\\_at](#) (STACK\_INT \*stack\_int, int index)

#### 4.17.1 説明

このファイルは整数スタックに関する操作をするためのものです。このファイルで使用される主な用途として、C 言語ソースファイルにおけるブロックレベルを設定するのに使用されます。

作者

faithnh

#### 4.17.2 関数

##### 4.17.2.1 int STACK\_INT\_at\_and\_alloc ( STACK\_INT \* stack\_int, int index )

対象の整数スタック stack\_int から、指定したインデックスの内容を出力する。インデックスが範囲外である場合は、そのサイズ分まで確保し、0 を返す。

引数

<i>stack_int</i>	対象の整数スタック
<i>index</i>	対象の座標

戻り値

指定したインデックスでの整数スタックの値を出力する。インデックスが範囲外である場合は 0 を返す。

##### 4.17.2.2 int STACK\_INT\_inclement\_at ( STACK\_INT \* stack\_int, int index )

対象の整数スタック stack\_int から、指定したインデックスの内容をインクリメントする。

引数

<i>stack_int</i>	対象の整数スタック
------------------	-----------

戻り値

インクリメントに成功したかどうかを示す。成功した場合は 1、そうでない場合は 0 を返す。

## 4.18 Main/Help.h

このファイルはヘルプ出力関係のファイルである

関数

- void [viewHelp](#) (void)

### 4.18.1 説明

このファイルはヘルプ出力関係のファイルである

作者

faithnh

### 4.18.2 関数

#### 4.18.2.1 void viewHelp ( void )

ANSIC Varidate Statement Adder tool ヘルプを出力させる

戻り値

なし

## 4.19 ProgramSlicing/DeclarationPSI.h

このファイルは Declaration Statement Program Slicing Information の略である。宣言式から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Synbol.h"
```

関数

- void [getDeclarationtPSI](#) (EXPR\_SLICING\_LIST \*expr\_slicing\_list, [AST](#) \*declaration, EXPR\_SLICING \*parent, VARIABLE\_TABLE\_LIST \*vtlist, FUNCTION\_INFORMATION\_

```
LIST *function_information_list, ASTPOINTER_LIST *ignore_ast_list, AST
*target_statement)
```

#### 4.19.1 説明

このファイルは Declaration Statement Program Slicing Information の略である。宣言式から、プログラムスライシングに関する情報を抽出するための操作が含まれている

作者

faithnh

#### 4.19.2 関数

**4.19.2.1 void getDeclarationtPSI ( EXPR\_SLICING\_LIST \* *expr\_slicing\_list*, AST \* *declaration*, EXPR\_SLICING \* *parent*, VARIABLE\_TABLE\_LIST \* *vtlist*, FUNCTION\_INFORMATION\_LIST \* *function\_information\_list*, ASTPOINTER\_LIST \* *ignore\_ast\_list*, AST \* *target\_statement* )**

宣言式の AST ノード *declaration* から、関数に対するプログラムスライシングを抽出し、[ プログラムスライシングリスト *expr\_slicing\_list* に追加する。

引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <i>expr_slicing_list</i>
<i>declaration</i>	式全般に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード
<i>vtlist</i>	変数テーブルリスト
<i>function_information_list</i>	関数に関する情報リスト
<i>ignore_ast_list</i>	重複防止のために無視するノードリスト
<i>target_statement</i>	<i>expression_statement</i> を指し示すノード

## 4.20 ProgramSlicing/FunctionPSI.h

このファイルは Function Program Slicing Information の略である。関数定義から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
```

## 関数

- `EXPR_SLICING * getFunctionPSI (EXPR_SLICING_LIST *expr_slicing_list, AST *function_definition, EXPR_SLICING *parent)`
- `void getParameterPSI (EXPR_SLICING *expr_slicing, AST *parameter_node, AST *basis_parameter_node)`

## 4.20.1 説明

このファイルは Function Program Slicing Information の略である。関数定義から、プログラムスライシングに関する情報を抽出するための操作が含まれている

作者

faithnh

## 4.20.2 関数

4.20.2.1 `EXPR_SLICING* getFunctionPSI ( EXPR_SLICING_LIST * expr_slicing_list, AST * function_definition, EXPR_SLICING * parent )`

関数定義の `AST` ノード `function_definition` から、関数に対するプログラムスライシングを抽出し、[ プログラムスライシングリスト `expr_slicing_list` に追加する。

引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <code>expr_slicing_list</code>
<i>function_definition</i>	関数定義に関する <code>AST</code> ノード
<i>parent</i>	ノードを追加するときの親ノード

戻り値

追加した関数のスライシングへのアドレスを返す

4.20.2.2 `void getParameterPSI ( EXPR_SLICING * expr_slicing, AST * parameter_node, AST * basis_parameter_node )`

パラメータリストに関する `AST` ノード `parameter_node` から関数定義のプログラムスライシングリスト `expr_slicing_list` に格納していく

引数

<i>expr_slicing_list</i>	関数定義のプログラムスライシング
<i>parameter_node</i>	各パラメータの定義リスト

<i>basis_- parameter_- node</i>	パラメータを指す AST ノード
---	------------------

## 4.21 ProgramSlicing/IfStatementPSI.h

このファイルは If Statement Program Slicing Information の略である。if 文もしくは、if と else 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Synbol.h"
```

### 関数

- `EXPR_SLICING * getIfStatementPSI (EXPR_SLICING_LIST *expr_slicing_list, AST *if_statement, EXPR_SLICING *parent, VARIABLE_TABLE_LIST *vtlist, FUNCTION_INFORMATION_LIST *function_information_list, ASTPOINTER_LIST *ignore_ast_list)`

#### 4.21.1 説明

このファイルは If Statement Program Slicing Information の略である。if 文もしくは、if と else 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

作者

faithnh

#### 4.21.2 関数

**4.21.2.1** `EXPR_SLICING* getIfStatementPSI ( EXPR_SLICING_LIST * expr_slicing_list,  
AST * if_statement, EXPR_SLICING * parent, VARIABLE_TABLE_LIST * vtlist,  
FUNCTION_INFORMATION_LIST * function_information_list, ASTPOINTER_LIST *  
ignore_ast_list )`

if 文もしくは、if と else 文の AST ノード `if_statement` から、関数に対するプログラムスライシングを抽出し、プログラムスライシングリスト `expr_slicing_list` に追加する。

引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <i>expr_slicing_list</i>
<i>if_statement</i>	if 文もしくは、if と else 文に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード
<i>vtlist</i>	変数テーブルリスト
<i>function_information_list</i>	関数に関する情報リスト
<i>ignore_ast_list</i>	重複防止のために無視するノードリスト

if 文もしくは、if と else 文の AST ノード *if\_statement* から、関数に対するプログラムスライシングを抽出し、プログラムスライシングリスト *expr\_slicing\_list* に追加する。

#### 引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト
<i>if_statement</i>	if 文もしくは、if と else 文に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード
<i>vtlist</i>	変数テーブルリスト
<i>function_information_list</i>	関数に関する情報リスト
<i>ignore_ast_list</i>	重複防止のために無視するノードリスト

## 4.22 ProgramSlicing/JumpStatementPSI.h

このファイルは Jump Statement Program Slicing Information の略である。GOTO、continue、break 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Symbol.h"
```

#### 関数

- void [getJumpStatementPSI](#) (EXPR\_SLICING\_LIST \**expr\_slicing\_list*, AST \**jump\_statement*, EXPR\_SLICING \**parent*)



### 4.22.1 説明

このファイルは Jump Statement Program Slicing Information の略である。GOTO、continue、break 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

作者

faithnh

### 4.22.2 関数

4.22.2.1 void getJumpStatementPSI ( EXPR\_SLICING\_LIST \* *expr\_slicing\_list*, AST \* *jump\_statement*, EXPR\_SLICING \* *parent* )

GOTO、continue、break 文の AST ノード *expression\_statement* から、関数に対するプログラムスライシングを抽出し、プログラムスライシングリスト *expr\_slicing\_list* に追加する。

引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <i>expr_slicing_list</i>
<i>jump_statement</i>	GOTO、continue、break 文に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード

## 4.23 ProgramSlicing/LabeledStatementPSI.h

このファイルは Labeled Statement Program Slicing Information の略である。goto ラベル文、case ラベル文、default ラベル文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Synbol.h"
```

関数

- EXPR\_SLICING \* [getLabeledStatementPSI](#) (EXPR\_SLICING\_LIST \**expr\_slicing\_list*, AST \**labeled\_statement*, EXPR\_SLICING \**parent*, VARIABLE\_TABLE\_LIST \**vtlist*, FUNCTION\_INFORMATION\_LIST \**function\_information\_list*, ASTPOINTER\_LIST \**ignore\_ast\_list*)

### 4.23.1 説明

このファイルは Labeled Statement Program Slicing Information の略である。 goto ラベル文、case ラベル文、default ラベル文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

作者

faithnh

### 4.23.2 関数

**4.23.2.1** `EXPR_SLICING* getLabeledStatementPSI ( EXPR_SLICING_LIST * expr_slicing_list, AST * labeled_statement, EXPR_SLICING * parent, VARIABLE_TABLE_LIST * vtlist, FUNCTION_INFORMATION_LIST * function_information_list, ASTPOINTER_LIST * ignore_ast_list )`

goto ラベル文、case ラベル文、default ラベル文の AST ノード `expression_statement` から、関数に対するプログラムスライシングを抽出し、プログラムスライシングリスト `expr_slicing_list` に追加する。

引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <code>expr_slicing_list</code>
<i>labeled_statement</i>	goto ラベル文、case ラベル文、default ラベル文に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード
<i>vtlist</i>	変数テーブルリスト
<i>function_information_list</i>	関数に関する情報リスト
<i>ignore_ast_list</i>	重複防止のために無視するノードリスト

## 4.24 ProgramSlicing/ProgramSlicing.h

このファイルはプログラムスライシングを行うためのものである。指定した識別子名および行数を入れることで、それに基づいてプログラムスライシングを行う

```
#include "../ANSICInformation/AST.h"
#include "../ANSICInformation/PointerArrayControl.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Synbol.h"
#include "ProgramSlicingInformation.h"
#include <cstl/list.h>
```

## 関数

- void \* [createStatementNodeList](#) (AST \*root, EXPR\_SLICING\_LIST \*expr\_slicing\_list, EXPR\_SLICING \*parent, VARIABLE\_TABLE\_LIST \*vtlist, FUNCTION\_INFORMATION\_LIST \*function\_information\_list)
- void [createDD\\_list\\_in\\_global](#) (EXPR\_SLICING \*expr\_slicing, EXPR\_SLICING\_LIST \*program\_expr\_slicing\_list)
- void [createDD\\_list](#) (EXPR\_SLICING \*expr\_slicing)
- void [createDD\\_listAll](#) (EXPR\_SLICING\_LIST \*expr\_slicing\_list, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, EXPR\_SLICING\_LIST \*program\_expr\_slicing\_list)
- void [createDD\\_list\\_in\\_Function](#) (EXPR\_SLICING \*expr\_slicing, EXPR\_SLICING\_LIST \*expr\_slicing\_list, FUNCTION\_INFORMATION\_LIST \*function\_information\_list)
- void [createDD\\_list\\_in\\_argument](#) (EXPR\_SLICING \*expr\_slicing, CSTLString \*argument\_name, EXPR\_SLICING \*function\_definition\_expr\_slicing)
- void [staticSlicing](#) (EXPR\_SLICING \*expr\_slicing)
- int [startStaticSlicing](#) (AST \*inst\_ast, EXPR\_SLICING\_LIST \*expr\_slicing\_list)

## 4.24.1 説明

このファイルはプログラムスライシングを行うためのものである。指定した識別子名および行数を入れることで、それに基づいてプログラムスライシングを行う

作者

faithnh

## 4.24.2 関数

4.24.2.1 void createDD\_list ( EXPR\_SLICING \* *expr\_slicing* )

指定した対象の命令に対して、データ依存関係の命令を探し、その命令を結びつける。

引数

<i>expr_slicing</i>	指定した対象の命令
---------------------	-----------

戻り値

なし

指定した対象の命令に対して、データ依存関係の命令を探し、その命令を結びつける。

引数

<i>expr_slicing</i>	指定した対象の命令
---------------------	-----------

**4.24.2.2** void createDD\_list\_in\_argument ( EXPR\_SLICING \* *expr\_slicing*, CStdString \* *argument\_name*, EXPR\_SLICING \* *function\_definition\_expr\_slicing* )

関数へのスライシング情報に対して、仮引数名が依存関係として用いている命令を探し、それを指定したプログラムスライシングのデータ依存関係として追加する

引数

<i>expr_slicing</i>	関数呼び出しのプログラムスライシング
<i>argument_name</i>	仮引数名
<i>function_definition_expr_slicing</i>	関数宣言へのプログラムスライシング

戻り値

なし

**4.24.2.3** void createDD\_list\_in\_Function ( EXPR\_SLICING \* *expr\_slicing*, EXPR\_SLICING\_LIST \* *expr\_slicing\_list*, FUNCTION\_INFORMATION\_LIST \* *function\_information\_list* )

関数呼び出しから、呼び出している関数へのデータ依存関係を生成する

引数

<i>expr_slicing</i>	関数呼び出しが含まれている処理名
<i>expr_slicing_list</i>	プログラム全体のプログラムスライシングリスト
<i>function_information_list</i>	関数に関する情報リスト

戻り値

なし

**4.24.2.4** void createDD\_list\_in\_global ( EXPR\_SLICING \* *expr\_slicing*, EXPR\_SLICING\_LIST \* *program\_expr\_slicing\_list* )

指定した対象の命令とグローバル変数に関してデータ依存関係があるかどうか調べる。存在すれば、グローバル変数宣言に対して、データ依存関係として結びつける

引数

<i>expr_slicing</i>	指定した対象の命令
---------------------	-----------

<i>program_ - expr_ - slicing_list</i>	プログラム全体のスライシングリスト
--	-------------------

指定した対象の命令とグローバル変数宣言に関してデータ依存関係があるかどうか調べる。存在すれば、グローバル変数宣言に対して、データ依存関係として結びつける

引数

<i>expr_slicing</i>	指定した対象の命令
<i>program_ - expr_ - slicing_list</i>	プログラム全体のスライシングリスト

```
4.24.2.5 void createDD_listAll ( EXPR_SLICING_LIST * expr_slicing_list,
    FUNCTION_INFORMATION_LIST * function_information_list, EXPR_SLICING_LIST *
    program_expr_slicing_list )
```

プログラムスライシングリスト中のすべての命令に対し、データ依存関係の命令を探し、その命令を結びつける。

引数

<i>expr_ - slicing_list</i>	指定した対象のプログラムスライシングリスト
<i>function_ - information_ - list</i>	関数に関する情報リスト
<i>program_ - expr_ - slicing_list</i>	プログラム全体へのスライシングリスト

戻り値

なし

```
4.24.2.6 void* createStatementNodeList ( AST * root, EXPR_SLICING_LIST *
    expr_slicing_list, EXPR_SLICING * parent, VARIABLE_TABLE_LIST * vtlist,
    FUNCTION_INFORMATION_LIST * function_information_list )
```

指定したプログラムASTノードから次のようなノードを取得し、そこからスライシングを行うための構造体リスト *expr\_slicing\_list* を生成する *function\_definition\_type\_a*、*function\_definition\_type\_b*、*expression\_statement*、*declaration\_with\_init*、*if\_statement*、*ifelse\_statement*、*switch\_statement*、*while\_statement*、*dowhile\_statement*、*for\_statement\_type\_a*、*for\_statement\_type\_b*、*for\_statement\_type\_c*、*for\_statement\_type\_d*、*goto\_statement*、*continue\_statement*、*break\_statement*、*return\_statement*、

return\_expr\_statement goto\_labeled\_statement、 case\_labeled\_statement、 default\_labeled\_statement

#### 引数

<i>root</i>	指定したプログラムASTノード
<i>expr_slicing_list</i>	スライシングを行うための構造体
<i>parent</i>	親のスライシングデータ
<i>vtlist</i>	変数テーブルリスト
<i>function_information_list</i>	関数に関する情報のリスト

#### 戻り値

なし

#### 4.24.2.7 int startStaticSlicing ( AST \* *inst\_ast*, EXPR\_SLICING\_LIST \* *expr\_slicing\_list* )

指定した命令へのASTノードに基づいて *expr\_slicing\_list* に対してスタティックスライシング処理を行う

#### 引数

<i>inst_ast</i>	指定した命令へのASTノード
<i>expr_slicing_list</i>	スタティックスライシング処理を行うための構造体リスト

#### 戻り値

成功したかどうかを返却する 1 : 成功 0 : 失敗

#### 4.24.2.8 void staticSlicing ( EXPR\_SLICING \* *expr\_slicing* )

指定した識別子の命令に対してスタティックスライシング処理を行う

#### 引数

<i>identifiers</i>	指定した識別子の配列オフセットリスト
<i>line</i>	指定した行数
<i>expr_slicing_list</i>	スタティックスライシング処理を行うための構造体リスト

#### 戻り値

なし

## 4.25 ProgramSlicing/ReturnStatementPSI.h

このファイルは Return Statement Program Slicing Information の略である。return 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Synbol.h"
```

### 関数

- void [getReturnStatementPSI](#) (EXPR\_SLICING\_LIST \*expr\_slicing\_list, AST \*return\_statement, EXPR\_SLICING \*parent, VARIABLE\_TABLE\_LIST \*vtlist, FUNCTION\_INFORMATION\_LIST \*function\_information\_list, ASTPOINTER\_LIST \*ignore\_ast\_list)

#### 4.25.1 説明

このファイルは Return Statement Program Slicing Information の略である。return 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

作者

faithnh

#### 4.25.2 関数

**4.25.2.1** void [getReturnStatementPSI](#) ( EXPR\_SLICING\_LIST \* *expr\_slicing\_list*, AST \* *return\_statement*, EXPR\_SLICING \* *parent*, VARIABLE\_TABLE\_LIST \* *vtlist*, FUNCTION\_INFORMATION\_LIST \* *function\_information\_list*, ASTPOINTER\_LIST \* *ignore\_ast\_list* )

return 文の AST ノード *expression\_statement* から、関数に対するプログラムスライシングを抽出し、プログラムスライシングリスト *expr\_slicing\_list* に追加する。

引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <i>expr_slicing_list</i>
<i>return_statement</i>	GOTO、continue、break 文に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード
<i>vtlist</i>	変数テーブルリスト
<i>function_information_list</i>	関数に関する情報リスト

<i>ignore_ast_list</i>	重複防止のために無視するノードリスト
------------------------	--------------------

## 4.26 ProgramSlicing/SwitchStatementPSI.h

このファイルは Switch Statement Program Slicing Information の略である。switch 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

```
#include "ProgramSlicingInformation.h"
#include "../ANSICInformation/FunctionInformation.h"
#include "../ANSICInformation/Symbol.h"
```

### 関数

- `EXPR_SLICING * getSwithStatementPSI (EXPR_SLICING_LIST *expr_slicing_list, AST *switch_statement, EXPR_SLICING *parent, VARIABLE_TABLE_LIST *vtlist, FUNCTION_INFORMATION_LIST *function_information_list, ASTPOINTER_LIST *ignore_ast_list)`

### 4.26.1 説明

このファイルは Switch Statement Program Slicing Information の略である。switch 文から、プログラムスライシングに関する情報を抽出するための操作が含まれている

#### 作者

faithnh

### 4.26.2 関数

**4.26.2.1** `EXPR_SLICING* getSwithStatementPSI ( EXPR_SLICING_LIST * expr_slicing_list, AST * switch_statement, EXPR_SLICING * parent, VARIABLE_TABLE_LIST * vtlist, FUNCTION_INFORMATION_LIST * function_information_list, ASTPOINTER_LIST * ignore_ast_list )`

switch 文の AST ノード `switch_statement` から、関数に対するプログラムスライシングを抽出し、プログラムスライシングリスト `expr_slicing_list` に追加する。

#### 引数

<i>expr_slicing_list</i>	追加先のプログラムスライシングリスト <code>expr_slicing_list</code>
--------------------------	---



<i>switch_-statement</i>	switch 文に関する AST ノード
<i>parent</i>	ノードを追加するときの親ノード
<i>vtlist</i>	変数テーブルリスト
<i>function_-information_-list</i>	関数に関する情報リスト
<i>ignore_ast_-list</i>	重複防止のために無視するノードリスト

# Index

abstract\_syntax\_tree, [7](#)  
addIncludeDataFromFile  
    PreProcess.h, [59](#)  
adjustProgramStart  
    PreProcess.h, [60](#)  
ANSICInformation/AST.h, [17](#)  
ANSICInformation/DivitionDeclarator.h, [26](#)  
ANSICInformation/DivitionInformation.h,  
    [27](#)  
ANSICInformation/FreeMemInfo.h, [29](#)  
ANSICInformation/FunctionInformation.h,  
    [31](#)  
ANSICInformation/MemallocInfo.h, [36](#)  
ANSICInformation/PointerArrayControl.h, [40](#)  
ANSICInformation/PreProcess.h, [58](#)  
ANSICInformation/Return\_Info.h, [61](#)  
ANSICInformation/SubEffectCheck.h, [62](#)  
ANSICInformation/Synbol.h, [63](#)  
ANSICInformation/Varidate\_statement.h, [75](#)  
ARRAY\_OFFSET  
    PointerArrayControl.h, [43](#)  
array\_offset, [8](#)  
ARRAY\_OFFSET\_LIST\_push\_back\_ref\_-  
    not\_dup  
    PointerArrayControl.h, [43](#)  
ArrayOffsetToValidateStatement  
    Varidate\_statement.h, [79](#)  
AST  
    AST.h, [18](#)  
AST.h  
    AST, [18](#)  
    deleteAST, [18](#)  
    findASTAddress, [19](#)  
    fprintDataFromAST, [19](#)  
    fprintfStatement, [19](#)  
    getArgumentAST, [20](#)  
    getArgumentString, [20](#)  
    getArgumentStringEnableExcept, [20](#)  
    getASTwithString, [21](#)  
    getStringFromAST, [22](#)  
    getStringFromASTEnableExcept, [22](#)  
    getStringReplaceASTtoString, [22](#)  
    multi\_push\_back\_childrenAST, [22](#)  
    new\_AST, [23](#)  
    printDataFromAST, [23](#)  
    printTargetASTNode, [23](#)  
    push\_back\_childrenAST, [24](#)  
    same\_new\_AST, [24](#)  
    setASTBlocklevelAndId, [24](#)  
    setASTReturntype, [25](#)  
    traverseAST, [25](#)  
    traverseASTwithXML, [25](#)  
CharStringExtend.h  
    isExpression, [105](#)  
    str\_extract, [106](#)  
checkCallFunction  
    PointerArrayControl.h, [43](#)  
checkContainSubEffectStatement  
    SubEffectCheck.h, [63](#)  
checkIdentifierPointerArrayLevel  
    PointerArrayControl.h, [43](#)  
checkIgnoreASTList  
    PointerArrayControl.h, [44](#)  
copyArrayOffsetList  
    PointerArrayControl.h, [44](#)  
createArrayExpression  
    PointerArrayControl.h, [45](#)  
createCheckUnboundAndUndefineOperationCheck  
    Varidate\_statement.h, [80](#)  
createDD\_list  
    ProgramSlicing.h, [121](#)  
createDD\_list\_in\_argument  
    ProgramSlicing.h, [122](#)  
createDD\_list\_in\_Function  
    ProgramSlicing.h, [122](#)  
createDD\_list\_in\_global  
    ProgramSlicing.h, [122](#)  
createDD\_listAll  
    ProgramSlicing.h, [123](#)  
createStatementNodeList

- ProgramSlicing.h, 123
- createValidateStatementFromIncDecExpr
  - Varidate\_statement.h, 81
- createValidateStatement
  - Varidate\_statement.h, 81
- createValidateStatementAdderFileEachCheck
  - Varidate\_statement.h, 83
- createValidateStatementForFreeAction
  - Varidate\_statement.h, 83
- createValidateStatementForMallocAction
  - Varidate\_statement.h, 84
- createValidateStatementFromArrayDefine
  - Varidate\_statement.h, 85
- createValidateVariableArrayExpression
  - PointerArrayControl.h, 45
- createVaridateStatementFromPointerDefine
  - Varidate\_statement.h, 86
- createViolentFreeOperation
  - Varidate\_statement.h, 86
- createZeroDivitionCheck
  - Varidate\_statement.h, 87
- CSTLString.h
  - CSTLString\_compare\_with\_char, 107
  - CSTLString\_delete\_tail\_str, 107
  - CSTLString\_ltrim, 107
  - CSTLString\_printf, 107
  - CSTLString\_replace\_string, 108
- CSTLString\_compare\_with\_char
  - CSTLString.h, 107
- CSTLString\_delete\_tail\_str
  - CSTLString.h, 107
- CSTLString\_ltrim
  - CSTLString.h, 107
- CSTLString\_printf
  - CSTLString.h, 107
- CSTLString\_replace\_string
  - CSTLString.h, 108
- DeclarationPSI.h
  - getDeclarationtPSI, 115
- deleteAST
  - AST.h, 18
- deleteOFFSET\_LIST
  - PointerArrayControl.h, 45
- deleteParameterDefine
  - FunctionInformation.h, 33
- deletePointer
  - PointerArrayControl.h, 46
  - Synbol.h, 66
- deletePointerAndArraySynbol
  - PointerArrayControl.h, 46
  - Synbol.h, 66
- DIVITION\_INFORMATION
  - DivitionInformation.h, 27
- divition\_information, 9
- DivitionDeclarator.h
  - OutputSourceAfterDivitionDeclarator, 26
- DivitionInformation.h
  - DIVITION\_INFORMATION, 27
  - getDIVITION\_INFORMATION\_LIST, 28
  - new\_DIVITION\_INFORMATION, 28
  - new\_DIVITION\_INFORMATION\_char, 28
  - printDIVITION\_INFORMATION\_LIST, 29
- find\_STRUCT\_TABLE\_DATA
  - Synbol.h, 66
- findASTAddress
  - AST.h, 19
- FlagDatabase.h
  - getFlagDatabase, 109
  - isArrayUnboundCheckMode, 109
  - isFreeViolationCheckMode, 109
  - isHelpMode, 110
  - isProgramSlicingMode, 110
  - isUndefineControlCheckMode, 110
  - isXmlMode, 110
  - isZeroDivitionCheckMode, 111
- for\_information, 9
- fprintDataFromAST
  - AST.h, 19
- fprintfStatement
  - AST.h, 19
- fprintProgramDataWithPSIVaridateStatement
  - Varidate\_statement.h, 88
- fprintProgramDataWithValidateStatement
  - Varidate\_statement.h, 88
- fprintValidateStatement
  - Varidate\_statement.h, 89
- fprintValidateStatement\_not\_assert
  - Varidate\_statement.h, 90
- freemem\_info, 10
- FREEMEMINFO
  - FreeMemInfo.h, 30
- FreeMemInfo.h
  - FREEMEMINFO, 30
  - getFreememInfo, 30

- new\_FREEMEMINFO, 31
- printFREEMEMINFO, 31
- FUNCTION\_INFORMATION
  - FunctionInformation.h, 32
- function\_information, 10
- FunctionInformation.h
  - deleteParameterDefine, 33
  - FUNCTION\_INFORMATION, 32
  - getFunctionInformation, 33
  - getFunctionInformationFromFile, 33
  - getIN\_OUT\_FLAG, 34
  - getParamInformationFromFunctionDefinition, 34
  - getPointerLevelFromFUNCTION\_INFORMATION\_LIST, 34
  - new\_FUNCTION\_INFORMATION, 35
  - new\_PARAM\_INFORMATION, 35
  - PARAM\_INFORMATION, 32
  - printFUNCTION\_INFORMATION\_LIST, 35
  - searchFUNCTION\_INFORMATION, 36
- FunctionPSI.h
  - getFunctionPSI, 116
  - getParameterPSI, 116
- get\_ARRAY\_OFFSET\_LISTIgnoreASTNAME
  - PointerArrayControl.h, 46
- getArgumentAST
  - AST.h, 20
- getArgumentOffsetInfo
  - PointerArrayControl.h, 47
- getArgumentString
  - AST.h, 20
- getArgumentStringEnableExcept
  - AST.h, 20
- getARRAY\_OFFSET\_LIST
  - PointerArrayControl.h, 48
- getArrayOffsetInAnpasandInfo
  - PointerArrayControl.h, 48
- getArrayOffsetInIncDecInfo
  - PointerArrayControl.h, 49
- getAssignment\_TYPE
  - SubEffectCheck.h, 63
- getASTList\_FromVALIDATE\_STATEMENT\_LIST
  - Varidate\_statement.h, 91
- getASTwithString
  - AST.h, 21
- getBasisLocationFromAssignmentExpression
  - Varidate\_statement.h, 91
- getBasisLocationFromExpression
  - Varidate\_statement.h, 92
- getCallocInformation
  - MemallocInfo.h, 38
- getDeclarationtPSI
  - DeclarationPSI.h, 115
- getDeclaratorArrayOffset
  - PointerArrayControl.h, 49
- getDeclaratorFromAST
  - Synbol.h, 66
- getDIVITION\_INFORMATION\_LIST
  - DivisionInformation.h, 28
- getExpressionOffsetInfo
  - PointerArrayControl.h, 50
- getFlagDatabase
  - FlagDatabase.h, 109
- getFreememInfo
  - FreeMemInfo.h, 30
- getFunctionInformation
  - FunctionInformation.h, 33
- getFunctionInformationFromFile
  - FunctionInformation.h, 33
- getFunctionPSI
  - FunctionPSI.h, 116
- getIfStatementPSI
  - IfStatementPSI.h, 117
- getIN\_OUT\_FLAG
  - FunctionInformation.h, 34
- getJumpStatementPSI
  - JumpStatementPSI.h, 119
- getLabeledStatementPSI
  - LabeledStatementPSI.h, 120
- getLeftAssignmentInfo
  - Varidate\_statement.h, 92
- getMallocInformation
  - MemallocInfo.h, 38
- getMallocMaxsize
  - MemallocInfo.h, 38
- getMemberList
  - Synbol.h, 67
- getNewValidateStatementID
  - Varidate\_statement.h, 93
- getOFFSET\_LISTFromVariableTable
  - PointerArrayControl.h, 50
- getOffsetLevelFromArrayOffset
  - PointerArrayControl.h, 51
- getParameterData
  - Synbol.h, 67

- getParameterPSI
  - FunctionPSI.h, [116](#)
- getParameterVARIABLE\_TABLE\_LIST
  - Synbol.h, [68](#)
- getParamInformationFromFunctionDifinition
  - FunctionInformation.h, [34](#)
- getPointerAccessOrIdentifierList
  - PointerArrayControl.h, [51](#)
- getPointerArrayOffset
  - PointerArrayControl.h, [51](#)
- getPointerLevelAndArrayLevel
  - Synbol.h, [68](#)
- getPointerLevelAndArrayLevelFromVARIABLEviewHelp, [114](#)
- TABLE
  - Synbol.h, [69](#)
- getPointerLevelFromFUNCTION\_INFORMATION\_LIST
  - FunctionInformation.h, [34](#)
- getReallocInformation
  - MemallocInfo.h, [38](#)
- getReturnStatementPSI
  - ReturnStatementPSI.h, [125](#)
- getRightAssignmentInfo
  - Varidate\_statement.h, [94](#)
- getSingleExpressionOffsetInfo
  - PointerArrayControl.h, [52](#)
- getStringFromAST
  - AST.h, [22](#)
- getStringFromASTEnableExcept
  - AST.h, [22](#)
- getStringReplaceASTtoString
  - AST.h, [22](#)
- getSTRUCT\_DATA
  - Synbol.h, [69](#)
- getSTRUCT\_TABLE\_DATA
  - Synbol.h, [69](#)
- getSwicthStatementPSI
  - SwitchStatementPSI.h, [126](#)
- getTYPEDEF\_TABLE\_DATA
  - Synbol.h, [70](#)
- getTYPEDEFfromAST
  - Synbol.h, [70](#)
- getUpperExpressionRelationNode
  - PointerArrayControl.h, [53](#)
- getValidate\_Variable
  - Varidate\_statement.h, [94](#)
- getValidateStatementFromAssignStatement
  - Varidate\_statement.h, [95](#)
- getValidateStatementFromCallFunction
  - Varidate\_statement.h, [95](#)
- getValidateStatementFromForIteration
  - Varidate\_statement.h, [96](#)
- getValidateStatementFromInitializer
  - Varidate\_statement.h, [97](#)
- getValidateStatementFromMallocNumber
  - Varidate\_statement.h, [98](#)
- getValidateStatementFromPointerOperator
  - Varidate\_statement.h, [99](#)
- getVARIABLE\_TABLE\_LIST
  - Synbol.h, [71](#)
- Help.h
- IdList.h
- IDLIST\_compare\_with, [112](#)
- printIDLIST, [112](#)
- SET\_STACK\_INTToIDLIST, [112](#)
- IDLIST\_compare\_with
  - IdList.h, [112](#)
- IfStatementPSI.h
  - getIfStatementPSI, [117](#)
- INCLUDE\_DATA
  - PreProcess.h, [59](#)
- include\_data, [11](#)
- includeComment
  - PreProcess.h, [60](#)
- initVALIDATE\_STATEMENT\_flag
  - Varidate\_statement.h, [100](#)
- isArrayUnboundCheckMode
  - FlagDatabase.h, [109](#)
- isExpression
  - CharStringExtend.h, [105](#)
- isFreeViolationCheckMode
  - FlagDatabase.h, [109](#)
- isHelpMode
  - FlagDatabase.h, [110](#)
- isProgramSlicingMode
  - FlagDatabase.h, [110](#)
- isUndefineControlCheckMode
  - FlagDatabase.h, [110](#)
- isXmlMode
  - FlagDatabase.h, [110](#)
- isZeroDivitionCheckMode
  - FlagDatabase.h, [111](#)
- JumpStatementPSI.h
  - getJumpStatementPSI, [119](#)
- LabeledStatementPSI.h

- getLabeledStatementPSI, 120
- Library/CharStringExtend.h, 105
- Library/CSTLString.h, 106
- Library/FlagDatabase.h, 108
- Library/IdList.h, 111
- Library/Stack\_int.h, 113
- Main/Help.h, 114
- maxOffsetLevelAddressFromArrayOffsetList
  - PointerArrayControl.h, 54
- maxOffsetLevelFromArrayOffsetList
  - PointerArrayControl.h, 54
- MEMALLOC\_INFO
  - MemallocInfo.h, 37
- MemallocInfo.h
  - getCallocInformation, 38
  - getMallocInformation, 38
  - getMallocMaxsize, 38
  - getReallocInformation, 38
  - MEMALLOC\_INFO, 37
  - memoryAllocationAnarysis, 39
  - new\_MEMALLOC\_INFO, 39
  - new\_MEMALLOC\_INFO\_char, 39
  - searchSizeof, 40
- memory\_allocation\_info, 11
- memoryAllocationAnarysis
  - MemallocInfo.h, 39
- minusArrayOffsetList
  - PointerArrayControl.h, 54
- moveArrayOffsetList
  - PointerArrayControl.h, 54
- multi\_push\_back\_childrenAST
  - AST.h, 22
- new\_ARRAY\_OFFSET
  - PointerArrayControl.h, 55
- new\_ARRAY\_OFFSET\_char
  - PointerArrayControl.h, 55
- new\_AST
  - AST.h, 23
- new\_DIVITION\_INFORMATION
  - DivitionInformation.h, 28
- new\_DIVITION\_INFORMATION\_char
  - DivitionInformation.h, 28
- new\_FREEMEMINFO
  - FreeMemInfo.h, 31
- new\_FUNCTION\_INFORMATION
  - FunctionInformation.h, 35
- new\_INCLUDE\_DATA
  - PreProcess.h, 60
- new\_MEMALLOC\_INFO
  - MemallocInfo.h, 39
- new\_MEMALLOC\_INFO\_char
  - MemallocInfo.h, 39
- new\_PARAM\_INFORMATION
  - FunctionInformation.h, 35
- new\_RETURN\_INFO
  - Return\_Info.h, 62
- new\_STRUCT\_TABLE
  - Synbol.h, 71
- new\_STRUCT\_TABLE\_with\_char
  - Synbol.h, 72
- new\_TYPEDEF\_TABLE
  - Synbol.h, 72
- new\_TYPEDEF\_TABLE\_with\_char
  - Synbol.h, 72
- new\_VALIDATE\_STATEMENT
  - Varidate\_statement.h, 101
- new\_VALIDATE\_STATEMENT\_char
  - Varidate\_statement.h, 101
- new\_VALIDATE\_VARIABLE
  - Varidate\_statement.h, 101
- new\_VALIDATE\_VARIABLE\_with\_char
  - Varidate\_statement.h, 102
- new\_VARIABLE\_TABLE
  - Synbol.h, 73
- new\_VARIABLE\_TABLE\_with\_char
  - Synbol.h, 73
- OFFSET\_LIST\_push\_back\_alloc
  - PointerArrayControl.h, 56
- OutputSourceAfterDivitionDeclarator
  - DivitionDeclarator.h, 26
- PARAM\_INFORMATION
  - FunctionInformation.h, 32
- param\_information, 11
- PointerArrayControl.h
  - ARRAY\_OFFSET, 43
  - ARRAY\_OFFSET\_LIST\_push\_back\_ref\_not\_dup, 43
  - checkCallFunction, 43
  - checkIdentifierPointerArrayLevel, 43
  - checkIgnoreASTList, 44
  - copyArrayOffsetList, 44
  - createArrayExpression, 45
  - createValidateVariableArrayExpression, 45
  - deleteOFFSET\_LIST, 45
  - deletePointer, 46

- deletePointerAndArraySynbol, [46](#)
- get\_ARRAY\_OFFSET\_LISTIgnoreASTName, [46](#)
- getArgumentOffsetInfo, [47](#)
- getARRAY\_OFFSET\_LIST, [48](#)
- getArrayOffsetInAnpasandInfo, [48](#)
- getArrayOffsetInIncDecInfo, [49](#)
- getDeclaratorArrayOffset, [49](#)
- getExpressionOffsetInfo, [50](#)
- getOFFSET\_LISTFromVariableTable, [50](#)
- getOffsetLevelFromArrayOffset, [51](#)
- getPointerAccessOrIdentifierList, [51](#)
- getPointerArrayOffset, [51](#)
- getSingleExpressionOffsetInfo, [52](#)
- getUpperExpressionRelationNode, [53](#)
- maxOffsetLevelAddressFromArrayOffset, [54](#)
- maxOffsetLevelFromArrayOffsetList, [54](#)
- minusArrayOffsetList, [54](#)
- moveArrayOffsetList, [54](#)
- new\_ARRAY\_OFFSET, [55](#)
- new\_ARRAY\_OFFSET\_char, [55](#)
- OFFSET\_LIST\_push\_back\_alloc, [56](#)
- printASTPOINTER\_LIST, [56](#)
- searchARRAY\_OFFSET\_LIST, [57](#)
- searchExpressionOrPointerArrayOrIdentifier, [57](#)
- searchOffsetLevelAddressFromArrayOffsetList, [57](#)
- searchPointerAccessOrIdentifier, [58](#)
- PreProcess.h
  - addIncludeDataFromFile, [59](#)
  - adjustProgramStart, [60](#)
  - INCLUDE\_DATA, [59](#)
  - includeComment, [60](#)
  - new\_INCLUDE\_DATA, [60](#)
  - preProcessor, [60](#)
  - readIncludeDataFromFile, [61](#)
- preProcessor
  - PreProcess.h, [60](#)
- printASTPOINTER\_LIST
  - PointerArrayControl.h, [56](#)
- printDataFromAST
  - AST.h, [23](#)
- printDIVITION\_INFORMATION\_LIST
  - DivitionInformation.h, [29](#)
- printFREEMEMINFO
  - FreeMemInfo.h, [31](#)
- printFUNCTION\_INFORMATION\_LIST
  - FunctionInformation.h, [35](#)
- printIDLIST
  - IdList.h, [112](#)
- printProgramDataWithValidateStatement
  - Varidate\_statement.h, [103](#)
- printSTRUCT\_TABLE\_LIST
  - Synbol.h, [74](#)
- printTargetASTNode
  - AST.h, [23](#)
- printTYPEDEF\_TABLE\_LIST
  - Synbol.h, [74](#)
- printVALIDATE\_VARIABLE\_LIST
  - Varidate\_statement.h, [104](#)
- printVARIABLE\_TABLE\_LIST
  - Synbol.h, [75](#)
- ProgramSlicing.h
  - createDD\_list, [121](#)
  - createDD\_list\_in\_argument, [122](#)
  - createDD\_list\_in\_Function, [122](#)
  - createDD\_list\_in\_global, [122](#)
  - createDD\_listAll, [123](#)
  - createStatementNodeList, [123](#)
  - startStaticSlicing, [124](#)
  - staticSlicing, [124](#)
- ProgramSlicing/DeclarationPSI.h, [114](#)
- ProgramSlicing/FunctionPSI.h, [115](#)
- ProgramSlicing/IfStatementPSI.h, [117](#)
- ProgramSlicing/JumpStatementPSI.h, [118](#)
- ProgramSlicing/LabeledStatementPSI.h, [119](#)
- ProgramSlicing/ProgramSlicing.h, [120](#)
- ProgramSlicing/ReturnStatementPSI.h, [125](#)
- ProgramSlicing/SwitchStatementPSI.h, [126](#)
- push\_back\_childrenAST
  - AST.h, [24](#)
- readIncludeDataFromFile
  - PreProcess.h, [61](#)
- RETURN\_INFO
  - Return\_Info.h, [62](#)
- return\_info, [12](#)
- Return\_Info.h
  - new\_RETURN\_INFO, [62](#)
  - RETURN\_INFO, [62](#)
- ReturnStatementPSI.h
  - getReturnStatementPSI, [125](#)
- same\_new\_AST
  - AST.h, [24](#)
- searchARRAY\_OFFSET\_LIST

- PointerArrayControl.h, [57](#)
- searchExpressionOrPointerArrayOrIden
  - PointerArrayControl.h, [57](#)
- searchFUNCTION\_INFORMATION
  - FunctionInformation.h, [36](#)
- searchOffsetLevelAddressFromArrayOffsetList
  - PointerArrayControl.h, [57](#)
- searchPointerAccessOrIdentifier
  - PointerArrayControl.h, [58](#)
- searchSizeof
  - MemallocInfo.h, [40](#)
- searchVARIABLE\_TABLE\_LIST
  - Synbol.h, [75](#)
- SET\_STACK\_INTToIDLIST
  - IdList.h, [112](#)
- setASTBlocklevelAndId
  - AST.h, [24](#)
- setASTReturnType
  - AST.h, [25](#)
- setValidateVariableFromExprSlicing
  - Varidate\_statement.h, [104](#)
- Stack\_int.h
  - STACK\_INT\_at\_and\_alloc, [113](#)
  - STACK\_INT\_inclement\_at, [113](#)
- STACK\_INT\_at\_and\_alloc
  - Stack\_int.h, [113](#)
- STACK\_INT\_inclement\_at
  - Stack\_int.h, [113](#)
- startStaticSlicing
  - ProgramSlicing.h, [124](#)
- staticSlicing
  - ProgramSlicing.h, [124](#)
- str\_extract
  - CharStringExtend.h, [106](#)
- STRUCT\_TABLE
  - Synbol.h, [65](#)
- struct\_table, [12](#)
- SubEffectCheck.h
  - checkContainSubEffectStatement, [63](#)
  - getAssignment\_TYPE, [63](#)
- SwitchStatementPSI.h
  - getSwicthStatementPSI, [126](#)
- Synbol.h
  - deletePointer, [66](#)
  - deletePointerAndArraySynbol, [66](#)
  - find\_STRUCT\_TABLE\_DATA, [66](#)
  - getDeclaratorFromAST, [66](#)
  - getMemberList, [67](#)
  - getParameterData, [67](#)
  - getParameterVARIABLE\_TABLE\_LIST, [68](#)
  - getPointerLevelAndArrayLevel, [68](#)
  - getPointerLevelAndArrayLevelFromVARIABLE\_TABLE, [69](#)
  - getSTRUCT\_DATA, [69](#)
  - getSTRUCT\_TABLE\_DATA, [69](#)
  - getTYPEDEF\_TABLE\_DATA, [70](#)
  - getTYPEDEFfromAST, [70](#)
  - getVARIABLE\_TABLE\_LIST, [71](#)
  - new\_STRUCT\_TABLE, [71](#)
  - new\_STRUCT\_TABLE\_with\_char, [72](#)
  - new\_TYPEDEF\_TABLE, [72](#)
  - new\_TYPEDEF\_TABLE\_with\_char, [72](#)
  - new\_VARIABLE\_TABLE, [73](#)
  - new\_VARIABLE\_TABLE\_with\_char, [73](#)
  - printSTRUCT\_TABLE\_LIST, [74](#)
  - printTYPEDEF\_TABLE\_LIST, [74](#)
  - printVARIABLE\_TABLE\_LIST, [75](#)
  - searchVARIABLE\_TABLE\_LIST, [75](#)
  - STRUCT\_TABLE, [65](#)
  - TYPEDEF\_TABLE, [65](#)
  - VARIABLE\_TABLE, [65](#)
- traverseAST
  - AST.h, [25](#)
- traverseASTwithXML
  - AST.h, [25](#)
- TYPEDEF\_TABLE
  - Synbol.h, [65](#)
- typedef\_table, [13](#)
- VALIDATE\_STATEMENT
  - Varidate\_statement.h, [79](#)
- validate\_statement, [13](#)
- VALIDATE\_STATEMENT\_LIST\_sort\_ast
  - Varidate\_statement.h, [104](#)
- VALIDATE\_VARIABLE
  - Varidate\_statement.h, [79](#)
- validate\_variable, [14](#)
- VARIABLE\_TABLE
  - Synbol.h, [65](#)
- variable\_table, [15](#)
- Varidate\_statement.h
  - ArrayOffsetToValidateStatement, [79](#)
  - createCheckUnboundAndUndefineOperationCheck, [80](#)



- createValidateStatementFromIncDec-Expr, [81](#)
- createValidateStatement, [81](#)
- createValidateStatementAdderFileEachCheck, [104](#)
- createValidateStatementForFreeAction, [83](#)
- createValidateStatementForMallocAction, [84](#)
- createValidateStatementFromArrayDefine, [85](#)
- createValidateStatementFromPointerDefine, [86](#)
- createViolentFreeOperation, [86](#)
- createZeroDivisionCheck, [87](#)
- fprintProgramDataWithPSIVaridateStatement, [88](#)
- fprintProgramDataWithValidateStatement, [88](#)
- fprintValidateStatement, [89](#)
- fprintValidateStatement\_not\_assert, [90](#)
- getASTList\_FromVALIDATE\_STATEMENT\_LIST, [91](#)
- getBasisLocationFromAssignmentExpression, [91](#)
- getBasisLocationFromExpression, [92](#)
- getLeftAssignmentInfo, [92](#)
- getNewValidateStatementID, [93](#)
- getRightAssignmentInfo, [94](#)
- getValidate\_Variable, [94](#)
- getValidateStatementFromAssignStatement, [95](#)
- getValidateStatementFromCallFunction, [95](#)
- getValidateStatementFromForIteration, [96](#)
- getValidateStatementFromInitializer, [97](#)
- getValidateStatementFromMallocNumber, [98](#)
- getValidateStatementFromPointerOperator, [99](#)
- initVALIDATE\_STATEMENT\_flag, [100](#)
- new\_VALIDATE\_STATEMENT, [101](#)
- new\_VALIDATE\_STATEMENT\_char, [101](#)
- new\_VALIDATE\_VARIABLE, [101](#)
- new\_VALIDATE\_VARIABLE\_with\_char, [102](#)
- printProgramDataWithValidateStatement, [103](#)
- printVALIDATE\_VARIABLE\_LIST, [104](#)
- setValidateVariableFromExprSlicing, [104](#)
- VALIDATE\_STATEMENT, [79](#)
- VALIDATE\_STATEMENT\_LIST\_sort\_ast, [104](#)
- VALIDATE\_VARIABLE, [79](#)
- viewHelpHelp.h, [114](#)