

Compilateur NASM pour language LL

v1.0

Généré par Doxygen 1.8.12

Table des matières

1 c-compilator-II-for-nasm	1
2 Liste des choses à faire	3
3 Index des modules	5
3.1 Modules	5
4 Index des structures de données	7
4.1 Structures de données	7
5 Index des fichiers	9
5.1 Liste des fichiers	9
6 Documentation des modules	11
6.1 Compilateur	11
6.1.1 Description détaillée	11
6.1.2 Documentation du type de l'énumération	11
6.1.2.1 ArgsSecond	11
6.1.3 Documentation des fonctions	12
6.1.3.1 afficherAideF()	12
6.1.3.2 main()	12
6.1.4 Documentation des variables	13
6.1.4.1 yytext	13
6.1.4.2 yyin	13
6.2 Génération du code assembleur	14
6.2.1 Description détaillée	15

6.3 Gestion des données	16
6.3.1 Description détaillée	16
6.3.2 Documentation des définitions de type	16
6.3.2.1 assembleur_entry_instr_circularAdjList	16
6.4 Énumération	17
6.4.1 Description détaillée	17
6.4.2 Documentation du type de l'énumération	17
6.4.2.1 pseudo_instruction_data	17
6.4.2.2 pseudo_instruction_bss	17
6.4.2.3 byte_flag	18
6.5 Tableau des correspondance enum->str	19
6.5.1 Description détaillée	19
6.5.2 Documentation des variables	19
6.5.2.1 pseudo_instruction_data_str	19
6.5.2.2 pseudo_instruction_bss_str	19
6.6 Fonction et variable pour la gestion des listes d'adjacence	20
6.6.1 Description détaillée	20
6.6.2 Documentation des fonctions	20
6.6.2.1 assembleur_add_commentary()	20
6.6.2.2 assembleur_commentary_next()	21
6.6.2.3 circularAdjListStruct_header() [1/2]	21
6.6.2.4 circularAdjListCopier_header() [1/6]	22
6.6.2.5 circularAdjListComparator_header() [1/3]	22
6.6.2.6 circularAdjListStaticDecl_struct() [1/3]	22
6.6.2.7 circularAdjListCopier_header() [2/6]	23
6.6.2.8 circularAdjListComparator_header() [2/3]	23
6.6.2.9 circularAdjListStaticDecl_struct() [2/3]	23
6.6.2.10 circularAdjListCopier_header() [3/6]	24
6.6.2.11 circularAdjListComparator_header() [3/3]	24
6.6.2.12 circularAdjListStaticDecl_struct() [3/3]	24

6.6.2.13	circularAdjListStruct_header() [2/2]	24
6.6.2.14	circularAdjListCopier_header() [4/6]	25
6.6.2.15	circularAdjListCopier_header() [5/6]	25
6.6.2.16	circularAdjListCopier_header() [6/6]	26
6.7	Variable de gestion de l'assembleur	27
6.8	Gestion des commentaires	28
6.8.1	Description détaillée	28
6.9	Fonction de gestion pour l'assembleur	29
6.9.1	Description détaillée	29
6.9.2	Documentation des macros	29
6.9.2.1	SECURE_MALLOC_STRUCT	29
6.9.3	Documentation des fonctions	30
6.9.3.1	assembleur_new_entryName()	30
6.9.3.2	assembleur_add_data_andComment()	30
6.9.3.3	assembleur_add_bss_andComment()	30
6.9.3.4	assembleur_add_entry_andComment()	31
6.9.3.5	assembleur_add_entryName_andComment()	31
6.9.3.6	assembleur_add_data()	32
6.9.3.7	assembleur_add_bss()	32
6.9.3.8	assembleur_add_entry()	33
6.9.3.9	assembleur_add_entryName()	33
6.10	Fonction de génération du flux de sortie	34
6.10.1	Description détaillée	34
6.10.2	Documentation des macros	34
6.10.2.1	SIMPLE_ERROR	34
6.10.2.2	printSpace	34
6.10.3	Documentation des fonctions	35
6.10.3.1	assembleur_dump_section_oneEntry()	35
6.10.3.2	assembleur_dump_section_import()	35
6.10.3.3	assembleur_dump_section_data()	36

6.10.3.4 <code>assembleur_dump_section_bss()</code>	36
6.10.3.5 <code>assembleur_dump_section_entry_instrs()</code>	36
6.10.3.6 <code>assembleur_dump_section_entry()</code>	37
6.10.3.7 <code>assembleur_dump()</code>	37
6.11 Fonction d'instruction	38
6.11.1 Description détaillée	39
6.11.2 Documentation des macros	39
6.11.2.1 <code>CHECK_ENTRY</code>	39
6.11.2.2 <code>SECURE_MALLOC_INST_STR</code>	39
6.11.2.3 <code>SECURE_INST_STR</code>	39
6.11.2.4 <code>ADD_ENTRY</code>	39
6.11.3 Documentation des fonctions	40
6.11.3.1 <code>printedSizeOfSigned()</code>	40
6.11.3.2 <code>printedSizeOfUnsigned()</code>	40
6.11.3.3 <code>assembleur_entry_0s_andComment()</code>	41
6.11.3.4 <code>assembleur_entry_1s_andComment()</code>	41
6.11.3.5 <code>assembleur_entry_2s_andComment()</code>	42
6.11.3.6 <code>assembleur_entry_add_subEntry_andComment()</code>	43
6.11.3.7 <code>assembleur_entry_add_entryPoint_andComment()</code>	44
6.11.3.8 <code>assembleur_entry_subEntryName_andComment()</code>	45
6.11.3.9 <code>assembleur_entry_0s()</code>	45
6.11.3.10 <code>assembleur_entry_1s()</code>	46
6.11.3.11 <code>assembleur_entry_2s()</code>	46
6.11.3.12 <code>assembleur_entry_add_subEntry()</code>	47
6.11.3.13 <code>assembleur_entry_add_entryPoint()</code>	47
6.11.3.14 <code>assembleur_entry_subEntryName()</code>	48
6.11.3.15 <code>assembleur_entry_getLastInstr()</code>	48
6.11.3.16 <code>assembleur_entry_getName()</code>	48
6.12 Conversion en str	49
6.12.1 Description détaillée	49

6.12.2 Documentation des macros	49
6.12.2.1 GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE	49
6.12.3 Documentation des fonctions	49
6.12.3.1 assembleur_make_tabVar()	49
6.12.3.2 assembleur_make_localVar()	50
6.12.3.3 assembleur_make_globalVar()	51
6.12.3.4 assembleur_make_constant()	51
6.12.3.5 assembleur_make_idSuffix()	52
6.12.3.6 assembleur_make_concat()	52
6.12.3.7 assembleur_entry_make_prefix()	53
6.12.3.8 assembleur_entry_make_suffix()	53
6.12.3.9 assembleur_entry_make_next()	54
6.13 Fonction de gestion mémoire	55
6.14 Configuration du module Assembleur	56
6.14.1 Description détaillée	56
6.14.2 Documentation des macros	56
6.14.2.1 GENASM_ENTRY_PRETTY_NAME	56
6.14.2.2 GENASM_ENTRY_ARE_INDENT	56
6.14.2.3 GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE	56
6.14.2.4 GENASM_ENTRYPOINT_SEPERATE_RETURNLINE	56
6.15 Parcours de l'arbres abstrait pour la génération	57
6.15.1 Description détaillée	57
6.16 Variable de gestion de la table des symboles	58
6.16.1 Description détaillée	58
6.16.2 Documentation des variables	58
6.16.2.1 adresseGlobalCourant	58
6.16.2.2 adresseLocaleCourante	58
6.16.2.3 adresseArgumentCourant	58
6.16.2.4 ce	59
6.16.2.5 topEntry	59

6.16.2.6	endEntry	59
6.17	Definition pour la gestion du parcours	60
6.17.1	Description détaillée	60
6.17.2	Documentation des macros	60
6.17.2.1	ErreurSiNulle	60
6.17.2.2	ErreurSiMauvaisType	60
6.18	Fonction de parcours pour la génération	61
6.18.1	Description détaillée	61
6.19	Entrées principales	62
6.19.1	Description détaillée	62
6.19.2	Documentation des fonctions	62
6.19.2.1	parcours_prog()	62
6.19.2.2	parcours_init_asm()	63
6.20	Fonction de gestion interne	64
6.20.1	Description détaillée	64
6.20.2	Documentation des fonctions	64
6.20.2.1	parcours_get_endEntryName()	64
6.21	Declaration	66
6.21.1	Description détaillée	66
6.21.2	Documentation des fonctions	66
6.21.2.1	parcours_prefixe_I_dec()	66
6.21.2.2	parcours_suffixe_I_dec()	67
6.21.2.3	parcours_dec()	68
6.21.2.4	parcours_dec_fonctions()	69
6.22	Instruction	71
6.22.1	Description détaillée	71
6.22.2	Documentation des fonctions	71
6.22.2.1	parcours_prefixe_I_instr()	71
6.22.2.2	parcours_suffixe_I_instr()	73
6.22.2.3	parcours_instr()	73

6.22.2.4 parcours_instr_ecrire()	75
6.22.2.5 parcours_instr_affect()	76
6.22.2.6 parcours_instr_incr()	77
6.22.2.7 parcours_instr_tantque()	78
6.22.2.8 parcours_instr_appel()	79
6.22.2.9 parcours_instr_si()	80
6.22.2.10 parcours_instr_retour()	81
6.23 Expression	83
6.23.1 Description détaillée	83
6.23.2 Documentation des fonctions	83
6.23.2.1 parcours_prefixe_l_exp()	83
6.23.2.2 parcours_suffixe_l_exp()	84
6.23.2.3 parcours_exp()	85
6.23.2.4 parcours_exp_comparaison()	86
6.23.2.5 parcours_exp_lire()	88
6.23.2.6 parcours_exp_var()	88
6.23.2.7 parcours_exp_int()	89
6.23.2.8 parcours_exp_appel()	90
6.23.2.9 parcours_exp_op()	91
6.24 Configuration du module parcours	93
6.24.1 Description détaillée	93
6.24.2 Documentation des macros	93
6.24.2.1 PARCOURS_STARTENTRY_ACCEPT_RETVALUE	93
6.24.2.2 PARCOURS_VAR_USE_DWORD	93
6.24.2.3 HAVE_LOGIQUE_COMPARAISSON	93
6.24.2.4 IO_USE_READLINE_ENTRY	93
6.25 Outils pour le parcours de l'arbres abstrait	94
6.25.1 Description détaillée	94
6.26 Expression	95
6.26.1 Description détaillée	95

6.26.2.2 Documentation des fonctions	95
6.26.2.1 parcoursTools_constexpr_operation()	95
6.26.2.2 parcoursTools_constexpr()	96
6.26.2.3 parcoursTools_computeOperation()	97
6.26.2.4 parcoursTools_comparaisonOperation()	98
6.27 Liste	99
6.27.1 Description détaillée	99
6.27.2 Documentation des fonctions	99
6.27.2.1 parcoursTools_taille_n_l()	99
6.28 Aquisition	101
6.28.1 Description détaillée	101
6.28.2 Documentation des fonctions	101
6.28.2.1 parcours_exp()	101
6.28.2.2 parcoursTools_accesVar()	102
6.28.3 Documentation des variables	103
6.28.3.1 ce	103
6.29 Bibliothèque	104
6.29.1 Description détaillée	104
6.29.2 Documentation des fonctions	104
6.29.2.1 parcoursTools_dump_asm_lib()	104
6.30 Bibliothèque comparatorExpr	106
6.30.1 Description détaillée	106
6.30.2 Documentation des macros	106
6.30.2.1 LIB_COMPARATOREXPR_PREFIX	106
6.30.2.2 LIB_COMPARATOREXPR_SIZE	106
6.30.3 Documentation des fonctions	107
6.30.3.1 parcoursTools_lib_comparatorExpr()	107
6.30.3.2 parcoursTools_genLib_comparatorExpr()	107
6.31 Bibliothèque comparaisonBool	108
6.31.1 Description détaillée	108

6.31.2 Documentation des macros	108
6.31.2.1 LIB_COMPARAISSONBOOL_PREFIX	108
6.31.2.2 LIB_COMPARAISSONBOOL_SIZE	108
6.31.3 Documentation des fonctions	109
6.31.3.1 parcoursTools_lib_comparaisonBool_getName()	109
6.31.3.2 parcoursTools_lib_comparaisonBool()	109
6.31.3.3 parcoursTools_genLib_comparaisonBool()	109
6.32 Gestion de la table des symboles	110
6.32.1 Description détaillée	110
6.32.2 Documentation des fonctions	111
6.32.2.1 tabsymb_setAffiche()	111
6.32.3 Documentation des variables	111
6.32.3.1 adresseGlobalCourant	111
6.32.3.2 adresseLocaleCourante	111
6.32.3.3 adresseArgumentCourant	111
6.33 Fonction de Gestion	112
6.33.1 Description détaillée	112
6.33.2 Documentation des fonctions	112
6.33.2.1 tabsymb_entreeFonction()	112
6.33.2.2 tabsymb_sortieFonction()	113
6.33.2.3 tabsymb_ajoutIdentificateur()	113
6.33.2.4 tabsymb_rechercheExecutable()	114
6.33.2.5 tabsymb_rechercheDeclarative()	115
6.33.2.6 tabsymb_getSymbole()	115
6.33.2.7 tabsymb_getSymboleFonctionCourante()	116
6.33.2.8 tabsymb_dump()	116
6.34 Definition pour la Gestion	117
6.34.1 Description détaillée	117
6.34.2 Documentation des macros	117
6.34.2.1 MAX_IDENTIF	117

6.35 Definition des portées	118
6.35.1 Description détaillée	118
6.35.2 Documentation des macros	118
6.35.2.1 P_VARIABLE_GLOBALE	118
6.35.2.2 P_VARIABLE_LOCALE	118
6.35.2.3 P_ARGUMENT	118
6.36 Definition des types	119
6.36.1 Description détaillée	119
6.36.2 Documentation des macros	119
6.36.2.1 T_ENTIER	119
6.36.2.2 T_TABLEAU_ENTIER	119
6.36.2.3 T_FONCTION	119
6.37 Analyseur lexical	120
6.37.1 Description détaillée	120
6.38 Test des catégories	122
6.38.1 Description détaillée	122
6.38.2 Documentation des macros	122
6.38.2.1 YYTEXT_MAX	122
6.38.2.2 is_num	122
6.38.2.3 is_maj	122
6.38.2.4 is_min	122
6.38.2.5 is_alpha	123
6.38.2.6 is_alpha_simple	123
6.38.2.7 is_alphanum	123
6.39 Table Opérateurs 1 Caractere	124
6.39.1 Description détaillée	124
6.39.2 Documentation des macros	124
6.39.2.1 MotsClefs_Operators_1Carac_SIZE	124
6.40 Table des Opérateurs	125
6.40.1 Description détaillée	125

6.40.2 Documentation des macros	125
6.40.2.1 MotsClefs_Operators_SIZE	125
6.41 Mots instructions	126
6.41.1 Description détaillée	126
6.41.2 Documentation des macros	126
6.41.2.1 MotsClefs_Instruction_SIZE	126
6.42 Mots fonction	127
6.42.1 Description détaillée	127
6.42.2 Documentation des macros	127
6.42.2.1 MotsClefs_Fonction_SIZE	127
6.43 Variables du parser yylex	128
6.43.1 Description détaillée	128
6.43.2 Documentation des variables	128
6.43.2.1 yytext	128
6.43.2.2 nb_ligne	128
6.44 Fonctions du parser yylex	129
6.44.1 Description détaillée	129
6.44.2 Documentation des fonctions	129
6.44.2.1 erreurLexical()	129
6.44.2.2 mangeEspaces()	130
6.44.2.3 lireCar()	130
6.44.2.4 delireCar()	130
6.44.2.5 initBuff()	131
6.44.2.6 estUneVariable()	131
6.44.2.7 estUneFonction()	131
6.44.2.8 estUnNombre()	131
6.44.2.9 yylex()	132
6.44.2.10 nom_token()	132
6.45 Script d'execution	133
6.45.1 Description détaillée	133

6.45.2 Documentation des fonctions	133
6.45.2.1 afficherLexique()	133
6.46 Analyse Syntaxique	134
6.46.1 Description détaillée	134
6.46.2 Documentation des fonctions	135
6.46.2.1 syntaxical_setAffiche()	135
6.46.2.2 syntaxical()	135
6.47 Define des fonctions	136
6.47.1 Description détaillée	136
6.47.2 Documentation des macros	136
6.47.2.1 Entre	136
6.47.2.2 Sortie	136
6.47.2.3 Erreur	137
6.48 Variables globales	138
6.48.1 Description détaillée	138
6.49 Gestion erreur et yylex	139
6.49.1 Description détaillée	139
6.49.2 Documentation des fonctions	139
6.49.2.1 uniteCouranteSuivante()	139
6.49.2.2 erreurSyntaxical()	139
6.50 Analyse	140
6.50.1 Description détaillée	140
6.51 Entrée, Variable globale, Fonction, Variable	141
6.51.1 Description détaillée	141
6.51.2 Documentation des fonctions	141
6.51.2.1 programme()	141
6.51.2.2 optDecVariables()	142
6.51.2.3 listeDecVariables()	142
6.51.2.4 listeDecVariablesBis()	143
6.51.2.5 declarationVariable()	143

6.51.2.6 <code>optTailleTableau()</code>	143
6.51.2.7 <code>listeDecFonctions()</code>	143
6.51.2.8 <code>declarationFonction()</code>	143
6.51.2.9 <code>listeParam()</code>	143
6.51.2.10 <code>optListeDecVariables()</code>	144
6.52 Instruction	145
6.52.1 Description détaillée	145
6.52.2 Documentation des fonctions	145
6.52.2.1 <code>instruction()</code>	145
6.52.2.2 <code>instructionAffect()</code>	146
6.52.2.3 <code>instructionBloc()</code>	146
6.52.2.4 <code>listInstructions()</code>	146
6.52.2.5 <code>instructionSi()</code>	146
6.52.2.6 <code>optSinon()</code>	147
6.52.2.7 <code>instructionTantque()</code>	147
6.52.2.8 <code>instructionFaire()</code>	147
6.52.2.9 <code>instructionAppel()</code>	147
6.52.2.10 <code>instructionRetour()</code>	147
6.52.2.11 <code>instructionEcriture()</code>	147
6.52.2.12 <code>instructionVide()</code>	148
6.52.2.13 <code>instructionIncrementer()</code>	148
6.53 Expression	149
6.53.1 Description détaillée	149
6.53.2 Documentation des fonctions	149
6.53.2.1 <code>expression()</code>	149
6.53.2.2 <code>expressionBis()</code>	150
6.53.2.3 <code>conjonction()</code>	150
6.53.2.4 <code>conjonctionBis()</code>	150
6.53.2.5 <code>comparaison()</code>	151
6.53.2.6 <code>comparaisonBis()</code>	151

6.53.2.7 expArith()	151
6.53.2.8 expArithBis()	151
6.53.2.9 terme()	152
6.53.2.10 termeBis()	152
6.53.2.11 negation()	152
6.53.2.12 facteur()	152
6.53.2.13 var()	153
6.53.2.14 optIndice()	153
6.53.2.15 appelFct()	153
6.53.2.16 listeExpressions()	153
6.53.2.17 listeExpressionsBis()	154
6.54 /Remplissage des premiers	155
6.54.1 Description détaillée	155
6.54.2 Documentation des fonctions	155
6.54.2.1 premier_valide()	155
6.54.2.2 premier_union()	156
6.54.2.3 initialise_premiers_terminaux()	157
6.54.2.4 initialise_premiers_nonterminaux()	158
6.54.2.5 suivant_valide()	160
6.54.2.6 suivant_union()	161
6.54.2.7 premier_union_suivant()	161
6.54.2.8 initialise_suivants_terminaux()	162
6.54.2.9 initialise_suivants_nonterminaux()	163

7 Documentation des structures de données	165
7.1 Référence de la structure <code>AdjList</code>	165
7.1.1 Description détaillée	165
7.1.2 Documentation des champs	166
7.1.2.1 <code>next</code>	166
7.1.2.2 <code>prev</code>	166
7.1.2.3 <code>isEnd</code>	166
7.2 Référence de la structure <code>assembleur_bss</code>	166
7.2.1 Description détaillée	167
7.2.2 Documentation des champs	167
7.2.2.1 <code>etiquette</code>	167
7.2.2.2 <code>pi</code>	167
7.2.2.3 <code>nb</code>	167
7.3 Référence de la structure <code>assembleur_data</code>	167
7.3.1 Description détaillée	168
7.3.2 Documentation des champs	168
7.3.2.1 <code>etiquette</code>	168
7.3.2.2 <code>pi</code>	168
7.3.2.3 <code>valeur</code>	168
7.4 Référence de la structure <code>assembleur_entry</code>	168
7.4.1 Description détaillée	169
7.4.2 Documentation des champs	169
7.4.2.1 <code>name</code>	169
7.4.2.2 <code>instrs</code>	169
7.4.2.3 <code>counter</code>	169
7.5 Référence de la structure <code>desc_identif</code>	169
7.5.1 Description détaillée	170
7.5.2 Documentation des champs	170
7.5.2.1 <code>identif</code>	170
7.5.2.2 <code>portee</code>	170

7.5.2.3	type	170
7.5.2.4	adresse	170
7.5.2.5	complement	171
7.6	Référence de la structure n_appel	171
7.6.1	Documentation des champs	171
7.6.1.1	fonction	171
7.6.1.2	args	171
7.7	Référence de la structure n_dec	172
7.7.1	Documentation des énumérations membres	173
7.7.1.1	anonymous enum	173
7.7.2	Documentation des champs	173
7.7.2.1	type [1/2]	173
7.7.2.2	nom	173
7.7.2.3	param	173
7.7.2.4	variables	173
7.7.2.5	corps	173
7.7.2.6	foncDec_	174
7.7.2.7	type [2/2]	174
7.7.2.8	varDec_	174
7.7.2.9	taille	174
7.7.2.10	tabDec_	174
7.7.2.11	u	174
7.8	Référence de la structure n_exp	174
7.8.1	Documentation des énumérations membres	175
7.8.1.1	anonymous enum	175
7.8.2	Documentation des champs	175
7.8.2.1	type	175
7.8.2.2	op	175
7.8.2.3	op1	175
7.8.2.4	op2	176

7.8.2.5	opExp_	176
7.8.2.6	var	176
7.8.2.7	entier	176
7.8.2.8	appel	176
7.8.2.9	u	176
7.9	Référence de la structure n_instr	176
7.9.1	Documentation des énumérations membres	178
7.9.1.1	anonymous enum	178
7.9.2	Documentation des champs	179
7.9.2.1	type	179
7.9.2.2	incr	179
7.9.2.3	test	179
7.9.2.4	alors	179
7.9.2.5	sinon	179
7.9.2.6	si_	179
7.9.2.7	faire	179
7.9.2.8	tantque_	179
7.9.2.9	faire_	179
7.9.2.10	appel	179
7.9.2.11	var	180
7.9.2.12	exp	180
7.9.2.13	affecte_	180
7.9.2.14	expression	180
7.9.2.15	retour_	180
7.9.2.16	ecrire_	180
7.9.2.17	liste	180
7.9.2.18	u	180
7.10	Référence de la structure n_l_dec	181
7.10.1	Documentation des champs	181
7.10.1.1	tete	181

7.10.1.2 queue	181
7.11 Référence de la structure n_l_exp	182
7.11.1 Documentation des champs	182
7.11.1.1 tete	182
7.11.1.2 queue	182
7.12 Référence de la structure n_l_instr	183
7.12.1 Documentation des champs	183
7.12.1.1 tete	183
7.12.1.2 queue	183
7.13 Référence de la structure n_prog	184
7.13.1 Documentation des champs	184
7.13.1.1 variables	184
7.13.1.2 fonctions	184
7.14 Référence de la structure n_var	184
7.14.1 Documentation des énumérations membres	185
7.14.1.1 anonymous enum	185
7.14.2 Documentation des champs	185
7.14.2.1 type	185
7.14.2.2 nom	185
7.14.2.3 indice	185
7.14.2.4 indicee_	185
7.14.2.5 u	186
7.15 Référence de la structure tabsymboles_	186
7.15.1 Description détaillée	186
7.15.2 Documentation des champs	187
7.15.2.1 tab	187
7.15.2.2 base	187
7.15.2.3 sommet	187

8 Documentation des fichiers	189
8.1 Référence du fichier afficheArbreAbstrait.c	189
8.1.1 Documentation des fonctions	190
8.1.1.1 afficheArbreAbstrait_n_prog()	190
8.1.1.2 afficheArbreAbstrait_l_instr()	190
8.1.1.3 afficheArbreAbstrait_instr()	191
8.1.1.4 afficheArbreAbstrait_instr_si()	192
8.1.1.5 afficheArbreAbstrait_instr_tantque()	193
8.1.1.6 afficheArbreAbstrait_instr_faire()	195
8.1.1.7 afficheArbreAbstrait_instr_affect()	196
8.1.1.8 afficheArbreAbstrait_instr_appel()	197
8.1.1.9 afficheArbreAbstrait_appel()	197
8.1.1.10 afficheArbreAbstrait_instr_retour()	198
8.1.1.11 afficheArbreAbstrait_instr_ecrire()	199
8.1.1.12 afficheArbreAbstrait_l_exp()	200
8.1.1.13 afficheArbreAbstrait_exp()	201
8.1.1.14 afficheArbreAbstrait_varExp()	202
8.1.1.15 afficheArbreAbstrait_opExp()	203
8.1.1.16 afficheArbreAbstrait_intExp()	204
8.1.1.17 afficheArbreAbstrait_lireExp()	205
8.1.1.18 afficheArbreAbstrait_appelExp()	206
8.1.1.19 afficheArbreAbstrait_l_dec()	206
8.1.1.20 afficheArbreAbstrait_dec()	207
8.1.1.21 afficheArbreAbstrait_foncDec()	208
8.1.1.22 afficheArbreAbstrait_varDec()	209
8.1.1.23 afficheArbreAbstrait_tabDec()	210
8.1.1.24 afficheArbreAbstrait_var()	211
8.1.1.25 afficheArbreAbstrait_var_simple()	211
8.1.1.26 afficheArbreAbstrait_var_indicee()	212
8.2 Référence du fichier afficheArbreAbstrait.h	213

8.3 Référence du fichier afficheXml.c	213
8.3.1 Documentation des macros	214
8.3.1.1 ErreurSiAucuneSortie	214
8.3.2 Documentation des fonctions	214
8.3.2.1 afficheXml_setOutput()	214
8.3.2.2 indent()	215
8.3.2.3 afficheXml_balise_ouvrante()	215
8.3.2.4 afficheXml_balise_fermante()	216
8.3.2.5 afficheXml_texte()	217
8.3.2.6 special_texte()	218
8.3.2.7 afficheXml_element()	218
8.4 Référence du fichier afficheXml.h	219
8.5 Référence du fichier analyseur_lexical.c	219
8.5.1 Documentation des macros	220
8.5.1.1 ELIF_TEST_TOKEN	220
8.6 Référence du fichier analyseur_lexical.h	221
8.7 Référence du fichier analyseur_syntactical.c	221
8.8 Référence du fichier analyseur_syntactical.h	223
8.9 Référence du fichier assembleur.c	223
8.10 Référence du fichier assembleur.h	226
8.11 Référence du fichier circularAdjList.c	227
8.11.1 Documentation des fonctions	227
8.11.1.1 circularAdjList_add()	227
8.11.1.2 circularAdjList_search()	228
8.11.1.3 circularAdjList_free()	229
8.12 Référence du fichier circularAdjList.h	230
8.12.1 Documentation des macros	231
8.12.1.1 circularAdjList_add	231
8.12.1.2 circularAdjList_search	232
8.12.1.3 circularAdjList_free	232

8.12.1.4	circularAdjListStruct_header	232
8.12.1.5	circularAdjListStruct_content	232
8.12.1.6	circularAdjListStaticDecl_struct	232
8.12.1.7	circularAdjListCopier_header	232
8.12.1.8	circularAdjListComparator_header	233
8.12.1.9	circularAdjListFreer_header	233
8.12.1.10	circularAdjListTools_headCast	233
8.12.1.11	circularAdjListTools_contentCast	233
8.12.1.12	circularAdjList_isEnd	233
8.12.1.13	circularAdjList_doWhileNext	233
8.12.2	Documentation des définitions de type	234
8.12.2.1	circularAdjListCopier_pf	234
8.12.2.2	circularAdjListComparator_pf	234
8.12.2.3	circularAdjListFreer_pf	234
8.12.3	Documentation des fonctions	234
8.12.3.1	circularAdjList_add_()	234
8.12.3.2	circularAdjList_search_()	235
8.12.3.3	circularAdjList_free_()	236
8.13	Référence du fichier compilo.c	236
8.14	Référence du fichier dump_symb.c	237
8.14.1	Documentation des fonctions	238
8.14.1.1	main()	238
8.15	Référence du fichier message.c	238
8.15.1	Documentation des fonctions	239
8.15.1.1	warningLigne()	239
8.15.1.2	erreurLigne()	239
8.15.1.3	erreur()	239
8.15.1.4	erreurArgs()	240
8.15.1.5	warning_1s()	240
8.15.1.6	erreur_1s()	241

8.16 Référence du fichier message.h	241
8.17 Référence du fichier parcours.c	241
8.18 Référence du fichier parcours.h	242
8.19 Référence du fichier parcoursTools.c	243
8.20 Référence du fichier parcoursTools.h	245
8.21 Référence du fichier premiers.c	245
8.21.1 Documentation des fonctions	246
8.21.1.1 est_premier()	246
8.21.1.2 initialise_premiers()	248
8.21.2 Documentation des variables	249
8.21.2.1 premiers	249
8.22 Référence du fichier premiers.h	249
8.22.1 Documentation des variables	250
8.22.1.1 premiers	250
8.23 Référence du fichier README.md	250
8.24 Référence du fichier suivants.c	250
8.24.1 Documentation des fonctions	251
8.24.1.1 est_suivant()	251
8.24.1.2 initialise_suivants()	252
8.24.2 Documentation des variables	253
8.24.2.1 suivants	253
8.25 Référence du fichier suivants.h	253
8.25.1 Documentation des variables	254
8.25.1.1 suivants	254
8.26 Référence du fichier symboles.h	254
8.26.1 Documentation des macros	255
8.26.1.1 EPSILON	255
8.26.1.2 NB_NON_TERMINAUX	255
8.26.1.3 _listeDecVariables_	255
8.26.1.4 _listeDecFonctions_	256

8.26.1.5 <code>_declarationVariable_</code>	256
8.26.1.6 <code>_declarationFonction_</code>	256
8.26.1.7 <code>_listeParam_</code>	256
8.26.1.8 <code>_listelInstructions_</code>	256
8.26.1.9 <code>_instruction_</code>	256
8.26.1.10 <code>_instructionAffect_</code>	256
8.26.1.11 <code>_instructionBloc_</code>	257
8.26.1.12 <code>_instructionSi_</code>	257
8.26.1.13 <code>_instructionTantque_</code>	257
8.26.1.14 <code>_instructionAppel_</code>	257
8.26.1.15 <code>_instructionRetour_</code>	257
8.26.1.16 <code>_instructionEcriture_</code>	257
8.26.1.17 <code>_instructionIncrementer_</code>	257
8.26.1.18 <code>_instructionVide_</code>	258
8.26.1.19 <code>_var_</code>	258
8.26.1.20 <code>_expression_</code>	258
8.26.1.21 <code>_appelFct_</code>	258
8.26.1.22 <code>_conjonction_</code>	258
8.26.1.23 <code>_negation_</code>	258
8.26.1.24 <code>_comparaison_</code>	258
8.26.1.25 <code>_expArith_</code>	259
8.26.1.26 <code>_terme_</code>	259
8.26.1.27 <code>_facteur_</code>	259
8.26.1.28 <code>_argumentsEffectifs_</code>	259
8.26.1.29 <code>_listeExpressions_</code>	259
8.26.1.30 <code>_listeExpressionsBis_</code>	259
8.26.1.31 <code>_conjonctionBis_</code>	259
8.26.1.32 <code>_optTailleTableau_</code>	260
8.26.1.33 <code>_expArithBis_</code>	260
8.26.1.34 <code>_optSinon_</code>	260

8.26.1.35 _comparaisonBis_	260
8.26.1.36 _optDecVariables_	260
8.26.1.37 _optIndice_	260
8.26.1.38 _listeDecVariablesBis_	260
8.26.1.39 _instructionPour_	260
8.26.1.40 _termeBis_	261
8.26.1.41 _expressionBis_	261
8.26.1.42 _instructionFaire_	261
8.26.1.43 _optListeDecVariables_	261
8.26.1.44 _programme_	261
8.26.1.45 NB_TERMINAUX	261
8.26.1.46 POINT_VIRGULE	261
8.26.1.47 PLUS	262
8.26.1.48 MOINS	262
8.26.1.49 FOIS	262
8.26.1.50 DIVISE	262
8.26.1.51 MODULO	262
8.26.1.52 PARENTHÈSE_OUVRANTE	262
8.26.1.53 PARENTHÈSE_FERMANTE	262
8.26.1.54 CROCHET_OUVRANT	262
8.26.1.55 CROCHET_FERMANT	263
8.26.1.56 ACCOLADE_OUVRANTE	263
8.26.1.57 ACCOLADE_FERMANTE	263
8.26.1.58 EGAL	263
8.26.1.59 DIFFERENT	263
8.26.1.60 INFÉRIEUR	263
8.26.1.61 SUPÉRIEUR	263
8.26.1.62 INFÉRIEUR_EGAL	263
8.26.1.63 SUPÉRIEUR_EGAL	264
8.26.1.64 ET	264

8.26.1.65 OU	264
8.26.1.66 NON	264
8.26.1.67 SI	264
8.26.1.68 ALORS	264
8.26.1.69 SINON	264
8.26.1.70 TANTQUE	264
8.26.1.71 FAIRE	265
8.26.1.72 ENTIER	265
8.26.1.73 RETOUR	265
8.26.1.74 LIRE	265
8.26.1.75 ECRIRE	265
8.26.1.76 INCR	265
8.26.1.77 ID_VAR	265
8.26.1.78 ID_FCT	265
8.26.1.79 NOMBRE	266
8.26.1.80 FIN	266
8.26.1.81 VIRGULE	266
8.27 Référence du fichier syntabs.c	266
8.27.1 Documentation des fonctions	267
8.27.1.1 cree_n_appel()	267
8.27.1.2 cree_n_prog()	267
8.27.1.3 cree_n_var_simple()	267
8.27.1.4 cree_n_var_indicee()	267
8.27.1.5 cree_n_exp_op()	268
8.27.1.6 cree_n_exp_appel()	268
8.27.1.7 cree_n_exp_var()	268
8.27.1.8 cree_n_exp_entier()	268
8.27.1.9 cree_n_exp_lire()	268
8.27.1.10 cree_n_l_exp()	268
8.27.1.11 cree_n_instr_incr()	269

8.27.1.12 cree_n_instr_si()	269
8.27.1.13 cree_n_instr_tantque()	269
8.27.1.14 cree_n_instr_faire()	269
8.27.1.15 cree_n_instr_affect()	269
8.27.1.16 cree_n_l_instr()	269
8.27.1.17 cree_n_instr_bloc()	270
8.27.1.18 cree_n_instr_appel()	270
8.27.1.19 cree_n_instr_ecrire()	270
8.27.1.20 cree_n_instr_retour()	270
8.27.1.21 cree_n_instr_vide()	271
8.27.1.22 cree_n_dec_var()	271
8.27.1.23 cree_n_dec_tab()	271
8.27.1.24 cree_n_dec_fonc()	271
8.27.1.25 cree_n_l_dec()	271
8.28 Référence du fichier syntabs.h	271
8.28.1 Documentation du type de l'énumération	272
8.28.1.1 operation	272
8.28.2 Documentation des fonctions	273
8.28.2.1 cree_n_prog()	273
8.28.2.2 cree_n_dec_var()	273
8.28.2.3 cree_n_dec_tab()	273
8.28.2.4 cree_n_dec_fonc()	273
8.28.2.5 cree_n_exp_op()	273
8.28.2.6 cree_n_exp_entier()	274
8.28.2.7 cree_n_exp_var()	274
8.28.2.8 cree_n_exp_appel()	274
8.28.2.9 cree_n_exp_lire()	274
8.28.2.10 cree_n_instr_incr()	274
8.28.2.11 cree_n_instr_si()	274
8.28.2.12 cree_n_instr_bloc()	275

8.28.2.13 cree_n_instr_tantque()	275
8.28.2.14 cree_n_instr_faire()	275
8.28.2.15 cree_n_instr_affect()	275
8.28.2.16 cree_n_instr_appel()	275
8.28.2.17 cree_n_instr_retour()	276
8.28.2.18 cree_n_instr_ecrire()	276
8.28.2.19 cree_n_instr_vide()	276
8.28.2.20 cree_n_appel()	276
8.28.2.21 cree_n_var_simple()	277
8.28.2.22 cree_n_var_indicee()	277
8.28.2.23 cree_n_l_exp()	277
8.28.2.24 cree_n_l_instr()	277
8.28.2.25 cree_n_l_dec()	277
8.29 Référence du fichier tabsymboles.c	277
8.30 Référence du fichier tabsymboles.h	278
9 Documentation des exemples	281
9.1 une	281
Index	283

Chapitre 1

c-compilator-II-for-nasm

Simple compilateur pour grammaire LL (prédéfinit) vers langage assembleur NASM.

Auteurs

- Yannick Robin
- Christophe-Alexandre Sonntag (<http://u4a.at>)

Tests

Les fichiers de test sont dans le dossier : ./generator

Documentation :

La documentation a été générée à l'aide de Doxygen.

- Fichier de configuration : ./Doxyfile
- Accéder à l'aide pdf : ./refman.pdf

Chapitre 2

Liste des choses à faire

Global(e) `assembleur_entry_make_next (assembleur_entry *entry)`

gerer la désallocation

Global(e) `assembleur_entry_make_prefix (assembleur_entry *entry, const char *str)`

gerer la désallocation

Global(e) `assembleur_entry_make_suffix (assembleur_entry *entry, const char *str)`

gerer la désallocation

Global(e) `assembleur_make_concat (const char *str1, const char *str2)`

gerer la désallocation

Global(e) `assembleur_make_constant (const int c)`

gerer la désallocation

Global(e) `assembleur_make_globalVar (const char *m)`

gerer la désallocation

Global(e) `assembleur_make_idSuffix (const char *prefix, const unsigned int id)`

gerer la désallocation

Global(e) `assembleur_make_localVar (const char *reg, const int adr)`

gerer la désallocation

Global(e) `assembleur_make_tabVar (const char *mTab, const char *reg)`

gerer la désallocation

Global(e) `circularAdjList_add_ (struct AdjList *circularAdjList, struct AdjList *previous_circularAdjList, const size_t sizeofList, void *dataToCopy, circularAdjListCopier_pf inserterFunction, circularAdjListComparator_pf comparatorFunction)`

gerer la désallocation

Global(e) `circularAdjList_add_ (struct AdjList *circularAdjList, struct AdjList *previous_circularAdjList, const size_t sizeofList, void *dataToCopy, circularAdjListCopier_pf inserterFunction, circularAdjListComparator_pf comparatorFunction)`

gerer la désallocation

Global(e) `main (int argc, char *argv[])`

terminer la désallocation

Global(e) `nom_token (int token, char *nom, char *valeur)`

voir pour remettre 'sys_instruction'

Global(e) `parcours_dec_fonctions (const n_dec *n)`

tester si cela ne fait pas de bug avec 'topEntry' plutot que 'ce'

Global(e) `parcours_exp_var (const n_var *variable)`

A Confirmer : PUSH direct de la variable, sans passer par MOV

Global(e) `parcours_instr (const n_instr *n)`

faire 'instruction faire'

voir pour 'instruction incrInst'

Global(e) **parcours_instr_si** (**const n_exp *test, const n_instr *alors, const n_instr *sinon**)

voir si utile de generer le bloc inutilisé dans le cas d'un constexpr par exemple, si la condition ne fait que valider, alors ne pas generer le sinon s'il existe ...

Global(e) **parcoursTools_computeOperation** (**const operation op, const char **reg, bool *oneOperand ↵ Destination, bool *isCall**)

Voir pour ajouter l'instruction initialisation de edx : mov edx, 0 ; initialise edx à zéro

Groupe **SyntAnalyseHead**

Ajouter opérateur 'neg' : '-'

Chapitre 3

Index des modules

3.1 Modules

Liste de tous les modules :

Compilateur	11
Génération du code assembleur	14
Gestion des données	16
Énumération	17
Tableau des correspondance enum->str	19
Fonction et variable pour la gestion des listes d'adjacence	20
Variable de gestion de l'assembleur	27
Gestion des commentaires	28
Fonction de gestion pour l'assembleur	29
Fonction de génération du flux de sortie	34
Fonction d'instruction	38
Conversion en str	49
Fonction de gestion mémoire	55
Configuration du module Assembleur	56
Parcours de l'arbres abstrait pour la génération	57
Variable de gestion de la table des symboles	58
Definition pour la gestion du parcours	60
Fonction de parcours pour la génération	61
Entrées principales	62
Fonction de gestion interne	64
Declaration	66
Instruction	71
Expression	83
Configuration du module parcours	93
Outils pour le parcours de l'arbres abstrait	94
Expression	95
Liste	99
Aquisition	101
Bibliothèque	104
Bibliothèque comparatorExpr	106
Bibliothèque comparaisonBool	108
Gestion de la table des symboles	110
Fonction de Gestion	112

Definition pour la Gestion	117
Definition des portées	118
Definition des types	119
Analyseur lexical	120
Test des catégories	122
Table Opérateurs 1 Caractere	124
Table des Opérateurs	125
Mots instructions	126
Mots fonction	127
Variables du parser yylex	128
Fonctions du parser yylex	129
Script d'execution	133
Ananlyse Syntaxique	134
Define des fonctions	136
Variables globales	138
Gestion erreur et yylex	139
Analyse	140
Entrée, Variable globale, Fonction, Variable	141
Instruction	145
Expression	149
/Remplissage des premiers	155

Chapitre 4

Index des structures de données

4.1 Structures de données

Liste des structures de données avec une brève description :

AdjList	165
///	
assembleur_bss	
Assembleur_bss : donnée pour section bss	166
assembleur_data	
Assembleur_data : donnée pour section data	167
assembleur_entry	
Assembleur_entry : donnée pour une entrée	168
desc_identif	169
n_appel	171
n_dec	172
n_exp	174
n_instr	176
n_l_dec	181
n_l_exp	182
n_l_instr	183
n_prog	184
n_var	184
tabsymboles	
Tabsymboles : Table des symboles (globale ET locale)	186

Chapitre 5

Index des fichiers

5.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

afficheArbreAbstrait.c	189
afficheArbreAbstrait.h	213
afficheXml.c	213
afficheXml.h	219
analyseur_lexical.c	219
analyseur_lexical.h	221
analyseur_syntactical.c	221
analyseur_syntactical.h	223
assembleur.c	223
assembleur.h	226
circularAdjList.c	227
circularAdjList.h	230
compilo.c	236
dump_symb.c	237
message.c	238
message.h	241
parcours.c	241
parcours.h	242
parcoursTools.c	243
parcoursTools.h	245
premiers.c	245
premiers.h	249
suivants.c	250
suivants.h	253
symboles.h	254
syntabs.c	266
syntabs.h	271
tabsymboles.c	277
tabsymboles.h	278

Chapitre 6

Documentation des modules

6.1 Compilateur

Compilateur.

Énumérations

- enum `ArgsSecond` { `ArgsSecond_None`, `ArgsSecond_outputPath` }
The ArgsSecond : enum nécessaire pour parser la seconde valeur d'un arguments.

Fonctions

- void `afficherAideF` ()
- int `main` (int argc, char *argv[])

Variables

- char `yytext` [100]
- FILE * `yyin` = NULL

6.1.1 Description détaillée

Compilateur.

Auteur

Alexis Nasr (<http://pageperso.lif.univ-mrs.fr/~alexis.nasr/>)
released by Yannick Robin
released by Christophe Sonntag (<http://u4a.at>)

6.1.2 Documentation du type de l'énumération

6.1.2.1 `ArgsSecond`

enum `ArgsSecond`

The ArgsSecond : enum nécessaire pour parser la seconde valeur d'un arguments.

Valeurs énumérées

ArgsSecond_None	
ArgsSecond_outputPath	

6.1.3 Documentation des fonctions

6.1.3.1 afficherAideF()

```
void afficherAideF ( )
```

Référencé par main().

Voici le graphe des appels de cette fonction :



6.1.3.2 main()

```
int main (
    int argc,
    char * argv[ ] )
```

A faire terminer la désallocation

Références afficherAideF(), ArgsSecond_None, ArgsSecond_outputPath, et yyin.

Voici le graphe d'appel pour cette fonction :



6.1.4 Documentation des variables

6.1.4.1 yytext

```
char yytext[100]
```

6.1.4.2 yyin

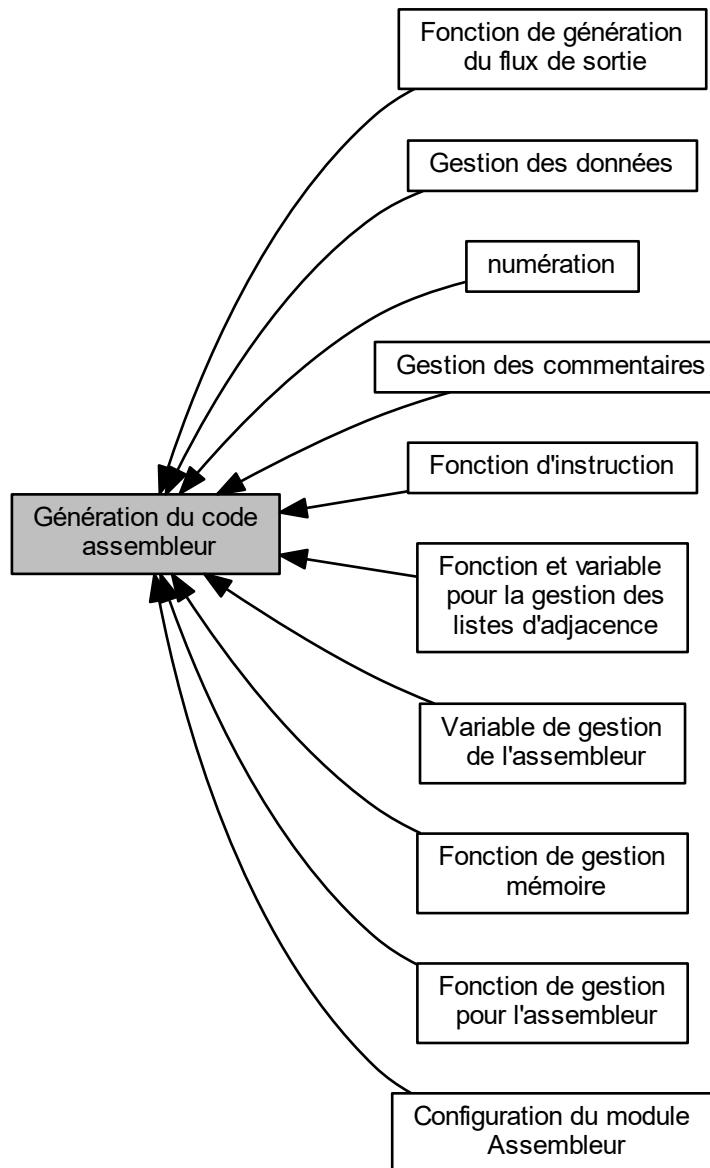
```
FILE* yyin = NULL
```

Référencé par main(), et mangeEspaces().

6.2 Génération du code assembleur

Partie génération code assembleur.

Graphe de collaboration de Génération du code assembleur :



Modules

- Gestion des données
- Enumération
- Fonction et variable pour la gestion des listes d'adjacence
- Variable de gestion de l'assembleur

- Gestion des commentaires
- Fonction de gestion pour l'assembleur
- Fonction de génération du flux de sortie
- Fonction d'instruction
- Fonction de gestion mémoire
- Configuration du module Assembleur

6.2.1 Description détaillée

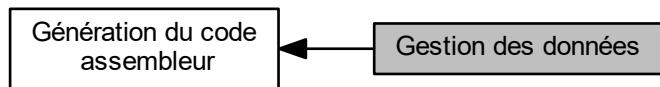
Partie génération code assembleur.

Auteur

released by Yannick Robin
released by Christophe Sonntag (<http://u4a.at>)

6.3 Gestion des données

Graphe de collaboration de Gestion des données :



Structures de données

- struct `assembleur_data`
assembleur_data : donnée pour section data
- struct `assembleur_bss`
assembleur_bss : donnée pour section bss
- struct `assembleur_entry`
assembleur_entry : donnée pour une entrée

Définitions de type

- typedef struct `assembleur_entry_instr_circularAdjList` `assembleur_entry_instr_circularAdjList`

6.3.1 Description détaillée

6.3.2 Documentation des définitions de type

6.3.2.1 `assembleur_entry_instr_circularAdjList`

```
typedef struct assembleur_entry_instr_circularAdjList assembleur_entry_instr_circularAdjList
```

6.4 Énumération

Graphe de collaboration de Énumération :



Modules

- Tableau des correspondance enum->str

Énumérations

- enum `pseudo_instruction_data` {

 db = 0, dw, dd, dq,

 dt }
- enum `pseudo_instruction_bss` {

 resb = 0, resw, resd, resq,

 rest }
- enum `byte_flag` { byte = 0, word, dword }

6.4.1 Description détaillée

6.4.2 Documentation du type de l'énumération

6.4.2.1 `pseudo_instruction_data`

`enum pseudo_instruction_data`

Valeurs énumérées

db	'define byte' declare un octet
dw	'define word' declare deux octets
dd	'define doubleword' declare quatre octets
dq	'define quadword' declare huit octets
dt	'define tenbytes' declare dix octets

6.4.2.2 `pseudo_instruction_bss`

`enum pseudo_instruction_bss`

Valeurs énumérées

resb	'reserve byte' declare un octet
resw	'reserve word' declare deux octets
resd	'reserve doubleword' declare quatre octets
resq	'reserve quadword' declare huit octets
rest	'reserve tenbytes' declare dix octets

6.4.2.3 byte_flag

```
enum byte_flag
```

Valeurs énumérées

byte	un octet
word	deux octets
dword	quatre octets

6.5 Tableau des correspondance enum->str

Graphe de collaboration de Tableau des correspondance enum->str :



Variables

- const char * [pseudo_instruction_data_str](#) []
- const char * [pseudo_instruction_bss_str](#) []

6.5.1 Description détaillée

6.5.2 Documentation des variables

6.5.2.1 [pseudo_instruction_data_str](#)

```
const char* pseudo_instruction_data_str[]
```

Valeur initiale :

```
=
{
  "db",
  "dw",
  "dd",
  "dq",
  "dt",
}
```

Référencé par `assembleur_dump_section_data()`.

6.5.2.2 [pseudo_instruction_bss_str](#)

```
const char* pseudo_instruction_bss_str[]
```

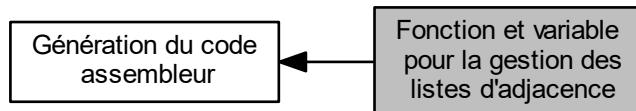
Valeur initiale :

```
=
{
  "resb",
  "resw",
  "resd",
  "resq",
  "rest",
}
```

Référencé par `assembleur_dump_section_bss()`.

6.6 Fonction et variable pour la gestion des listes d'adjacence

Graphe de collaboration de Fonction et variable pour la gestion des listes d'adjacence :



Fonctions

- void `assembleur_add_commentary` (const char *commentary)
 - assembleur_entry_lastInstr_setCommentary : ajouter un commentaire pour la prochaine instruction ajouté (si elle existe)*
- const char * `assembleur_commentary_next` ()
- `circularAdjListStruct_header` (assembleur_data_circularAdjList)
 - assembleur_data_circularAdjList Liste d'adjacence pour section data*
- `circularAdjListCopier_header` (data_circularAdjList_copier)
 - assembleur_data_circularAdjList Fonction de copie pour assembleur_data_circularAdjList*
- `circularAdjListComparator_header` (data_circularAdjList_comparator)
 - data_circularAdjList_comparator Fonction de comparaison pour assembleur_data_circularAdjList*
- `circularAdjListStaticDecl_struct` (assembleur_data_circularAdjList, data_circularAdjList)
 - data_circularAdjList Déclaration de la liste d'adjacence pour section data*
- `circularAdjListCopier_header` (bss_circularAdjList_copier)
 - bss_circularAdjList_copier Fonction de copie pour assembleur_bss_circularAdjList*
- `circularAdjListComparator_header` (bss_circularAdjList_comparator)
 - bss_circularAdjList_comparator Fonction de comparaison pour assembleur_bss_circularAdjList*
- `circularAdjListStaticDecl_struct` (assembleur_bss_circularAdjList, bss_circularAdjList)
 - bss_circularAdjList Déclaration de la liste d'adjacence pour section bss*
- `circularAdjListCopier_header` (entry_circularAdjList_copier)
 - entry_circularAdjList_copier Fonction de copie pour assembleur_entry_circularAdjList*
- `circularAdjListComparator_header` (entry_circularAdjList_comparator)
 - entry_circularAdjList_comparator Fonction de comparaison pour assembleur_entry_circularAdjList*
- `circularAdjListStaticDecl_struct` (assembleur_entry_circularAdjList, entry_circularAdjList)
 - entry_circularAdjList Déclaration de la liste d'adjacence pour section text*
- `circularAdjListStruct_header` (assembleur_entry_instr_circularAdjList)
 - assembleur_entry_instr_circularAdjList Liste d'adjacence pour les instructions/sous-entrées/points-entrées dans une d'une entrée*
- `circularAdjListCopier_header` (entry_instr_circularAdjList_copier)
 - entry_instr_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList*
- `circularAdjListCopier_header` (entry_subEntry_circularAdjList_copier)
 - entry_subEntry_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList*
- `circularAdjListCopier_header` (entry_entryPoint_circularAdjList_copier)
 - entry_entryPoint_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList*

6.6.1 Description détaillée

6.6.2 Documentation des fonctions

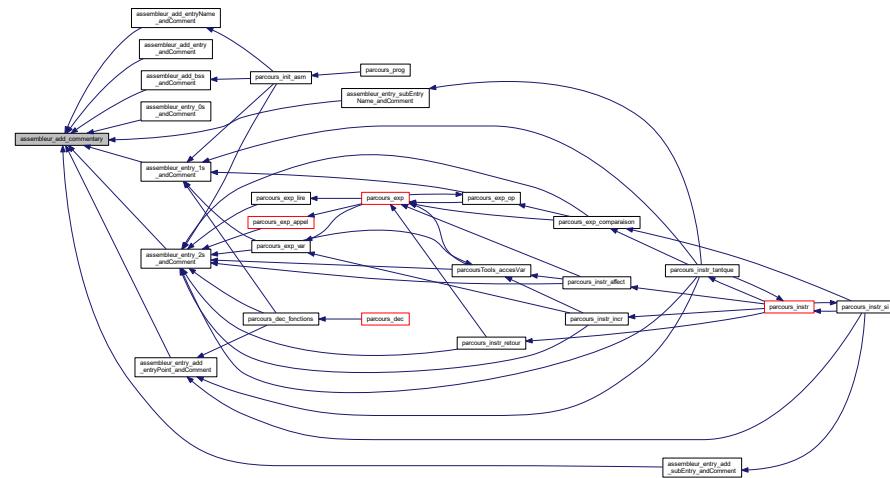
6.6.2.1 `assembleur_add_commentary()`

```
void assembleur_add_commentary (
    const char * commentary )
```

assembleur_entry_lastInstr_setCommentary : ajouter un commentaire pour la prochaine instruction ajouté (si elle existe)

Référencé par assembleur_add_bss_andComment(), assembleur_add_entry_andComment(), assembleur← add_entryName_andComment(), assembleur_entry_0s_andComment(), assembleur_entry_1s_andComment(), assembleur_entry_2s_andComment(), assembleur_entry_addEntryPoint_andComment(), assembleur_entry← add_subEntry_andComment(), et assembleur_entry_subEntryName_andComment().

Voici le graphe des appelants de cette fonction :

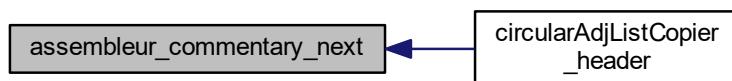


6.6.2.2 assembleur_commentary_next()

```
const char * assembleur_commentary_next ( )
```

Référencé par circularAdjListCopier_header().

Voici le graphe des appelants de cette fonction :



6.6.2.3 circularAdjListStruct_header() [1/2]

```
circularAdjListStruct_header (
    assembleur_data_circularAdjList )
```

assembleur_data_circularAdjList Liste d'adjacence pour section data

Références circularAdjListStruct_content.

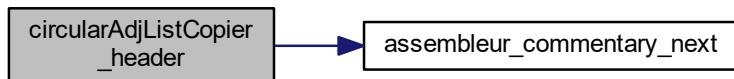
6.6.2.4 circularAdjListCopier_header() [1/6]

```
circularAdjListCopier_header (
    data_circularAdjList_copier )
```

assembleur_data_circularAdjList Fonction de copie pour assembleur_data_circularAdjList

Références assembleur_commentary_next(), et circularAdjListTools_contentCast.

Voici le graphe d'appel pour cette fonction :



6.6.2.5 circularAdjListComparator_header() [1/3]

```
circularAdjListComparator_header (
    data_circularAdjList_comparator )
```

data_circularAdjList_comparator Fonction de comparaison pour assembleur_data_circularAdjList

Références circularAdjListTools_contentCast.

6.6.2.6 circularAdjListStaticDecl_struct() [1/3]

```
circularAdjListStaticDecl_struct (
    assembleur_data_circularAdjList ,
    data_circularAdjList )
```

data_circularAdjList Déclaration de la liste d'adjacence pour section data

/// assembleur_bss_circularAdjList Liste d'adjacence pour section bss

Références circularAdjListStruct_content.

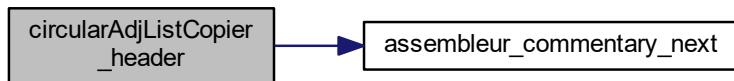
6.6.2.7 circularAdjListCopier_header() [2/6]

```
circularAdjListCopier_header (
    bss_circularAdjList_copier )
```

bss_circularAdjList_copier Fonction de copie pour assembleur_bss_circularAdjList

Références assembleur_commentary_next(), et circularAdjListTools_contentCast.

Voici le graphe d'appel pour cette fonction :



6.6.2.8 circularAdjListComparator_header() [2/3]

```
circularAdjListComparator_header (
    bss_circularAdjList_comparator )
```

bss_circularAdjList_comparator Fonction de comparaison pour assembleur_bss_circularAdjList

Références circularAdjListTools_contentCast.

6.6.2.9 circularAdjListStaticDecl_struct() [2/3]

```
circularAdjListStaticDecl_struct (
    assembleur_bss_circularAdjList ,
    bss_circularAdjList )
```

bss_circularAdjList Déclaration de la liste d'adjacence pour section bss

/// assembleur_entry_circularAdjList Liste d'adjacence pour section data

Références circularAdjListStruct_content.

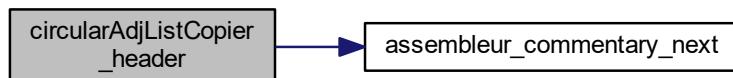
6.6.2.10 circularAdjListCopier_header() [3/6]

```
circularAdjListCopier_header (
    entry_circularAdjList_copier )
```

entry_circularAdjList_copier Fonction de copie pour assembleur_entry_circularAdjList

Références assembleur_commentary_next(), et circularAdjListTools_contentCast.

Voici le graphe d'appel pour cette fonction :



6.6.2.11 circularAdjListComparator_header() [3/3]

```
circularAdjListComparator_header (
    entry_circularAdjList_comparator )
```

entry_circularAdjList_comparator Fonction de comparaison pour assembleur_entry_circularAdjList

Références circularAdjListTools_contentCast.

6.6.2.12 circularAdjListStaticDecl_struct() [3/3]

```
circularAdjListStaticDecl_struct (
    assembleur_entry_circularAdjList ,
    entry_circularAdjList )
```

entry_circularAdjList Déclaration de la liste d'adjacence pour section text

//// The instrType enum : permet d'indiquer quel est le type d'ajout dans la liste d'instruction

6.6.2.13 circularAdjListStruct_header() [2/2]

```
circularAdjListStruct_header (
    assembleur_entry_instr_circularAdjList )
```

assembleur_entry_instr_circularAdjList Liste d'adjacence pour les instructions/sous-entrées/points-entrées dans une d'une entrée

Références circularAdjListStruct_content.

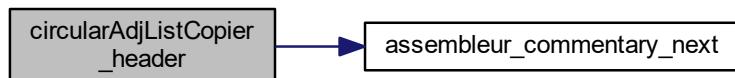
6.6.2.14 circularAdjListCopier_header() [4/6]

```
circularAdjListCopier_header (
    entry_instr_circularAdjList_copier )
```

entry_instr_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList

Références assembleur_commentary_next(), et circularAdjListTools_contentCast.

Voici le graphe d'appel pour cette fonction :



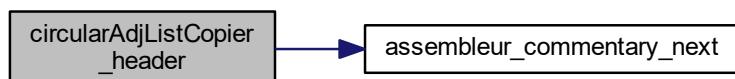
6.6.2.15 circularAdjListCopier_header() [5/6]

```
circularAdjListCopier_header (
    entry_subEntry_circularAdjList_copier )
```

entry_subEntry_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList

Références assembleur_commentary_next(), et circularAdjListTools_contentCast.

Voici le graphe d'appel pour cette fonction :



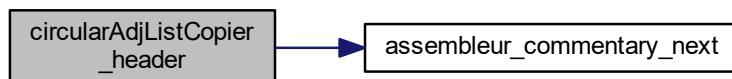
6.6.2.16 circularAdjListCopier_header() [6/6]

```
circularAdjListCopier_header (
    entry_entryPoint_circularAdjList_copier )
```

entry_entryPoint_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList

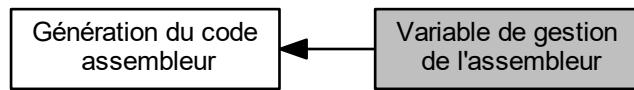
Références assembleur_commentary_next(), et circularAdjListTools_contentCast.

Voici le graphe d'appel pour cette fonction :



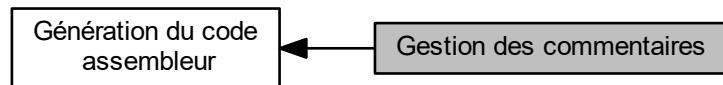
6.7 Variable de gestion de l'assembleur

Graphe de collaboration de Variable de gestion de l'assembleur :



6.8 Gestion des commentaires

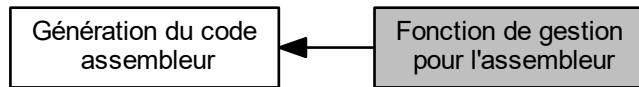
Graphe de collaboration de Gestion des commentaires :



6.8.1 Description détaillée

6.9 Fonction de gestion pour l'assembleur

Graphe de collaboration de Fonction de gestion pour l'assembleur :



Macros

- #define SECURE_MALLOC_STRUCT(dataType, varName)

Fonctions

- assembleur_entry * assembleur_new_entryName (const char *entryName)
- void assembleur_add_data_andComment (const char *etiquette, const pseudo_instruction_data pi, const char *valeur, const char *comment)
- void assembleur_add_bss_andComment (const char *etiquette, const pseudo_instruction_bss pi, const unsigned int nb, const char *comment)
- void assembleur_add_entry_andComment (assembleur_entry *entry, const char *comment)
- assembleur_entry * assembleur_add_entryName_andComment (const char *entryName, const char *comment)
 - assembleur_entry : généré une entrée entryName et l'ajoute dans la liste des entrées de la section data*
- void assembleur_add_data (const char *etiquette, const pseudo_instruction_data pi, const char *valeur)
 - // /
- void assembleur_add_bss (const char *etiquette, const pseudo_instruction_bss pi, const unsigned int nb)
- void assembleur_add_entry (assembleur_entry *entry)
- assembleur_entry * assembleur_add_entryName (const char *entryName)

6.9.1 Description détaillée

6.9.2 Documentation des macros

6.9.2.1 SECURE_MALLOC_STRUCT

```
#define SECURE_MALLOC_STRUCT(
    dataType,
    varName )
```

Valeur :

```
dataType * varName = ( dataType * ) malloc( sizeof( dataType ) ); \
if( varName == NULL ) { perror( "malloc assembleur struct" ); exit( EXIT_FAILURE ); }
```

Référencé par assembleur_add_bss(), et assembleur_new_entryName().

6.9.3 Documentation des fonctions

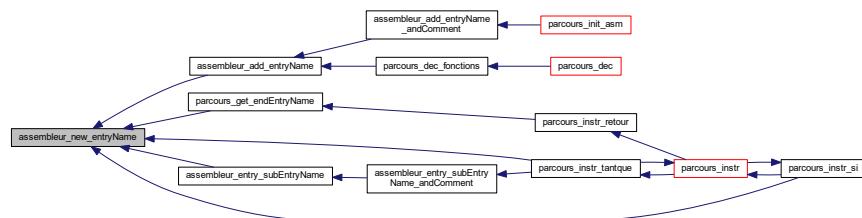
6.9.3.1 assembleur_new_entryName()

```
assembleur_entry* assembleur_new_entryName (
    const char * entryName )
```

Références SECURE_MALLOC_STRUCT.

Référencé par assembleur_add_entryName(), assembleur_entry_subEntryName(), parcours_get_endEntryName(), parcours_instr_si(), et parcours_instr_tantque().

Voici le graphe des appelleurs de cette fonction :



6.9.3.2 assembleur_add_data_andComment()

```
void assembleur_add_data_andComment (
    const char * etiquette,
    const pseudo_instruction_data pi,
    const char * valeur,
    const char * comment )
```

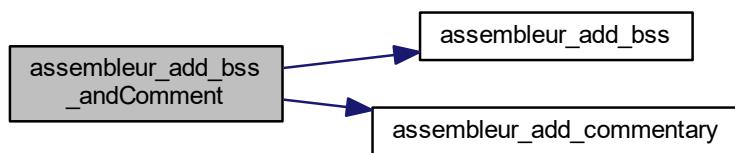
6.9.3.3 assembleur_add_bss_andComment()

```
void assembleur_add_bss_andComment (
    const char * etiquette,
    const pseudo_instruction_bss pi,
    const unsigned int nb,
    const char * comment )
```

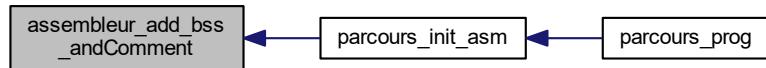
Références assembleur_add_bss(), et assembleur_add_commentary().

Référencé par parcours_init_asm().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :

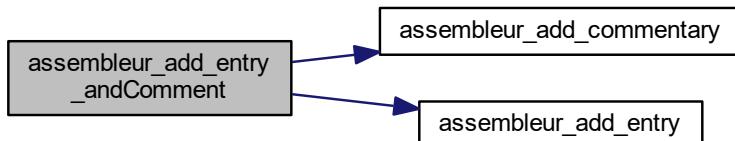


6.9.3.4 assemblleur_add_entry_andComment()

```
void assemblleur_add_entry_andComment (
    assemblleur_entry * entry,
    const char * comment )
```

Références assemblleur_add_commentary(), et assemblleur_add_entry().

Voici le graphe d'appel pour cette fonction :



6.9.3.5 assemblleur_add_entryName_andComment()

```
assemblleur_entry* assemblleur_add_entryName_andComment (
    const char * entryName,
    const char * comment )
```

assemblleur_entry : généré une entrée entryName et l'ajoute dans la liste des entrées de la section data

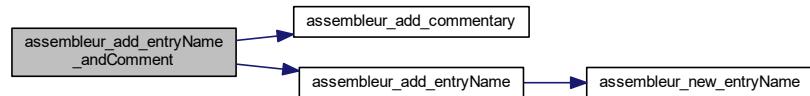
Paramètres

entry	
entryName	

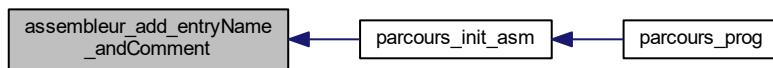
Références assemblleur_add_commentary(), et assemblleur_add_entryName().

Référencé par parcours_init_asm().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.9.3.6 assemblleur_add_data()

```

void assemblleur_add_data (
    const char * etiquette,
    const pseudo_instruction_data pi,
    const char * valeur )

///
```

6.9.3.7 assemblleur_add_bss()

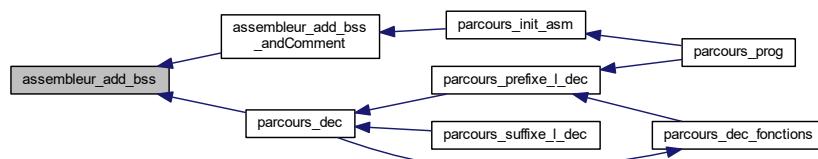
```

void assemblleur_add_bss (
    const char * etiquette,
    const pseudo_instruction_bss pi,
    const unsigned int nb )
```

Références circularAdjList_add, et SECURE_MALLOC_STRUCT.

Référencé par assemblleur_add_bss_andComment(), et parcours_dec().

Voici le graphe des appelants de cette fonction :



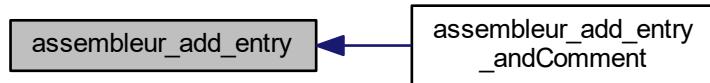
6.9.3.8 assembleur_add_entry()

```
void assembleur_add_entry (
    assembleur_entry * entry )
```

Références circularAdjList_add.

Référencé par assembleur_add_entry_andComment().

Voici le graphe des appelants de cette fonction :



6.9.3.9 assembleur_add_entryName()

```
assembleur_entry* assembleur_add_entryName (
    const char * entryName )
```

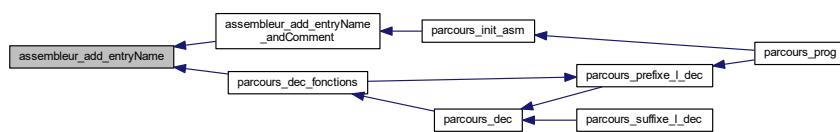
Références assembleur_new_entryName(), et circularAdjList_add.

Référencé par assembleur_add_entryName_andComment(), et parcours_dec_fonctions().

Voici le graphe d'appel pour cette fonction :

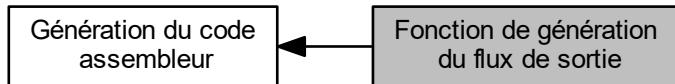


Voici le graphe des appelants de cette fonction :



6.10 Fonction de génération du flux de sortie

Graphe de collaboration de Fonction de génération du flux de sortie :



Macros

- #define SIMPLE_ERROR(errStr) { fprintf(stderr, "Erreur Assembleur : %s \n", errStr); exit(EXIT_FAILURE); }
- #define printSpace(space, out) for (int i = 0 ; i < space ; i++) {fputs(" ", out) ;}

Fonctions

- void assembleur_dump_section_oneEntry (FILE *out, assembleur_entry *entry, const char *commentary)
- void assembleur_dump_section_import (FILE *out)
- void assembleur_dump_section_data (FILE *out)
- void assembleur_dump_section_bss (FILE *out)
- void assembleur_dump_section_entry_instrs (FILE *out, struct assembleur_entry_instr_circularAdjList *instrs)
- void assembleur_dump_section_entry (FILE *out)
- void assembleur_dump (FILE *out)

6.10.1 Description détaillée

6.10.2 Documentation des macros

6.10.2.1 SIMPLE_ERROR

```
#define SIMPLE_ERROR(
    errStr ) { fprintf(stderr, "Erreur Assembleur : %s \n", errStr); exit( EXIT_FAILURE ); }
```

6.10.2.2 printSpace

```
#define printSpace(
    space,
    out ) for ( int i = 0; i < space; i++ ) {fputs( " ", out ) ;}
```

Référencé par assembleur_dump_section_entry_instrs(), et assembleur_dump_section_oneEntry().

6.10.3 Documentation des fonctions

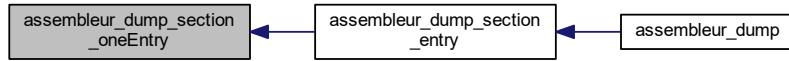
6.10.3.1 assembleur_dump_section_oneEntry()

```
void assembleur_dump_section_oneEntry (
    FILE * out,
    assembleur_entry * entry,
    const char * commentary )
```

Références printSpace.

Référencé par assembleur_dump_section_entry().

Voici le graphe des appelants de cette fonction :

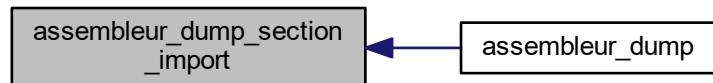


6.10.3.2 assembleur_dump_section_import()

```
void assembleur_dump_section_import (
    FILE * out )
```

Référencé par assembleur_dump().

Voici le graphe des appelants de cette fonction :



6.10.3.3 assembleur_dump_section_data()

```
void assembleur_dump_section_data (
    FILE * out )
```

Références circularAdjList_doWhileNext, et pseudo_instruction_data_str.

Référencé par assembleur_dump().

Voici le graphe des appelants de cette fonction :



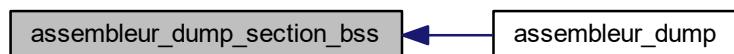
6.10.3.4 assembleur_dump_section_bss()

```
void assembleur_dump_section_bss (
    FILE * out )
```

Références circularAdjList_doWhileNext, et pseudo_instruction_bss_str.

Référencé par assembleur_dump().

Voici le graphe des appelants de cette fonction :



6.10.3.5 assembleur_dump_section_entry_instrs()

```
void assembleur_dump_section_entry_instrs (
    FILE * out,
    struct assembleur_entry_instr_circularAdjList * instrs )
```

Références assembleur_entry : :instrs, et printSpace.

6.10.3.6 assembleur_dump_section_entry()

```
void assembleur_dump_section_entry (
    FILE * out )
```

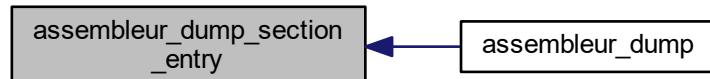
Références assembleur_dump_section_oneEntry(), et circularAdjList_doWhileNext.

Référencé par assembleur_dump().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

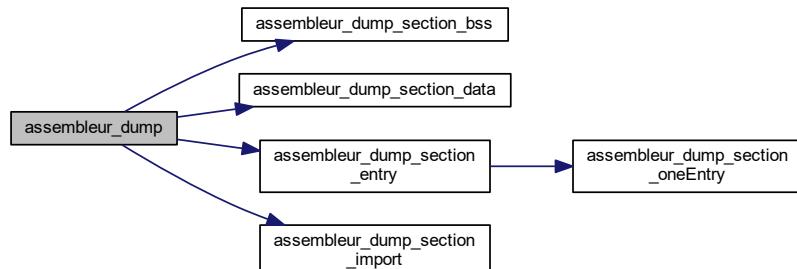


6.10.3.7 assembleur_dump()

```
void assembleur_dump (
    FILE * out )
```

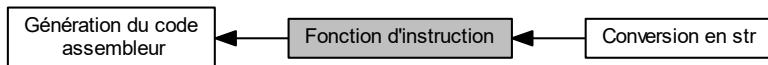
Références assembleur_dump_section_bss(), assembleur_dump_section_data(), assembleur_dump_section_entry(), et assembleur_dump_section_import().

Voici le graphe d'appel pour cette fonction :



6.11 Fonction d'instruction

Graphe de collaboration de Fonction d'instruction :



Modules

- Conversion en str

Macros

- #define **CHECK_ENTRY()** if(entry==NULL) { fprintf(stderr,"entry ne doit pas être nul\n"); exit(EXIT_FAILURE); }
- #define **SECURE_MALLOC_INST_STR(size)**
- #define **SECURE_INST_STR(size, pattern, args...)**
- #define **ADD_ENTRY(newStr)** entry->instrs = **circularAdjList_add(struct assembleur_entry_instr_circularAdjList, entry->instrs, newStr, entry_instr_circularAdjList_copier, NULL)**;

Fonctions

- unsigned int **printedSizeOfSigned** (const signed int val)
printedSizeOfSigned trouve la taille à afficher pour un signed int
- unsigned int **printedSizeOfUnsigned** (const unsigned int val)
printedSizeOfUnsigned trouve la taille à afficher pour un unsigned int
- void **assembleur_entry_0s_andComment** (assembleur_entry *entry, const char *instr, const char *comment)
- void **assembleur_entry_1s_andComment** (assembleur_entry *entry, const char *instr, const char *str, const char *comment)
- void **assembleur_entry_2s_andComment** (assembleur_entry *entry, const char *instr, const char *str1, const char *str2, const char *comment)
- void **assembleur_entry_add_subEntry_andComment** (assembleur_entry *entry, assembleur_entry *subEntry, const char *comment)
- void **assembleur_entry_addEntryPoint_andComment** (assembleur_entry *entry, assembleur_entry *entryPoint, const char *comment)
- assembleur_entry * **assembleur_entry_subEntryName_andComment** (assembleur_entry *entry, const char *subEntryName, const char *comment)
- void **assembleur_entry_0s** (assembleur_entry *entry, const char *instr)
///
- void **assembleur_entry_1s** (assembleur_entry *entry, const char *instr, const char *str)
- void **assembleur_entry_2s** (assembleur_entry *entry, const char *instr, const char *str1, const char *str2)
- void **assembleur_entry_add_subEntry** (assembleur_entry *entry, assembleur_entry *subEntry)
- void **assembleur_entry_addEntryPoint** (assembleur_entry *entry, assembleur_entry *entryPoint)
- assembleur_entry * **assembleur_entry_subEntryName** (assembleur_entry *entry, const char *subEntryName)
- const char * **assembleur_entry_getLastInstr** (assembleur_entry *entry)
///
- const char * **assembleur_entry_getName** (assembleur_entry *entry)

6.11.1 Description détaillée

6.11.2 Documentation des macros

6.11.2.1 CHECK_ENTRY

```
#define CHECK_ENTRY( ) if(entry==NULL) { fprintf(stderr,"entry ne doit pas être nul\n"); exit( EXIT_FAILURE ); }
```

Référencé par assembleur_entry_0s(), assembleur_entry_1s(), assembleur_entry_2s(), assembleur_entry_add_entryPoint(), assembleur_entry_add_subEntry(), et assembleur_entry_subEntryName().

6.11.2.2 SECURE_MALLOC_INST_STR

```
#define SECURE_MALLOC_INST_STR( size )
```

Valeur :

```
char * instStr = ( char * ) malloc( size ); \
if( instStr == NULL ) { perror( "malloc assembleur entry instr" ); exit( EXIT_FAILURE ); }
```

6.11.2.3 SECURE_INST_STR

```
#define SECURE_INST_STR( size, pattern, args... )
```

Valeur :

```
const unsigned int malloc_size = sizeof(char) * (size + 1); \
SECURE_MALLOC_INST_STR( malloc_size ) \
snprintf( instStr, malloc_size, pattern, args );
```

Référencé par assembleur_entry_1s(), assembleur_entry_2s(), assembleur_entry_make_next(), assembleur_entry_make_prefix(), assembleur_entry_make_suffix(), assembleur_make_concat(), assembleur_make_constant(), assembleur_make_globalVar(), assembleur_make_idSuffix(), assembleur_make_localVar(), et assembleur_make_tabVar().

6.11.2.4 ADD_ENTRY

```
#define ADD_ENTRY( newStr ) entry->instrs = circularAdjList_add( struct assembleur_entry_instr_circularAdjList, entry->instrs, newStr, entry_instr_circularAdjList_copier, NULL );
```

Référencé par assembleur_entry_0s(), assembleur_entry_1s(), et assembleur_entry_2s().

6.11.3 Documentation des fonctions

6.11.3.1 printedSizeOfSigned()

```
unsigned int printedSizeOfSigned (
    const signed int val )
```

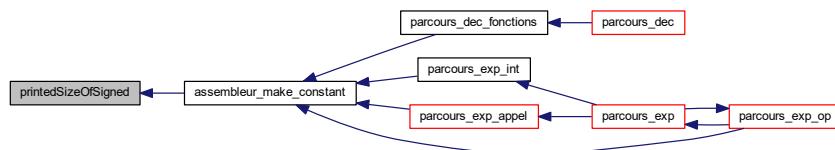
printedSizeOfSigned trouve la taille à afficher pour un signed int

Note

si val dépasse 10 alors $\log_{10}(p) + 1$
 si négatif alors doit prendre en compte le '-' (+1)

Référencé par assembleur_make_constant().

Voici le graphe des appels de cette fonction :



6.11.3.2 printedSizeOfUnsigned()

```
unsigned int printedSizeOfUnsigned (
    const unsigned int val )
```

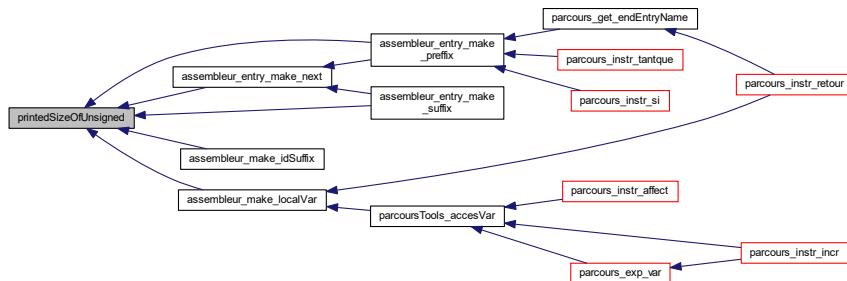
printedSizeOfUnsigned trouve la taille à afficher pour un unsigned int

Note

si val dépasse 10 alors $\log_{10}(p) + 1$

Référencé par assembleur_entry_make_next(), assembleur_entry_make_prefix(), assembleur_entry_make_← suffix(), assembleur_make_idSuffix(), et assembleur_make_localVar().

Voici le graphe des appels de cette fonction :

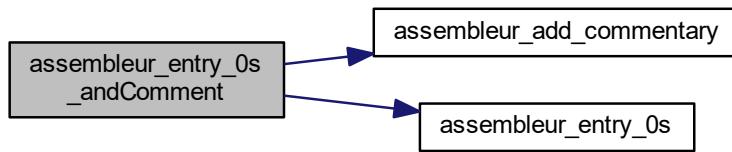


6.11.3.3 assembleur_entry_0s_andComment()

```
void assembleur_entry_0s_andComment (  
    assembleur_entry * entry,  
    const char * instr,  
    const char * comment )
```

Références assembleur_add_commentary(), et assembleur_entry_0s().

Voici le graphe d'appel pour cette fonction :



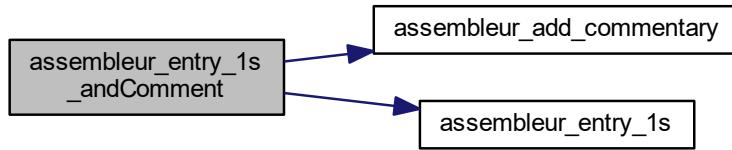
6.11.3.4 assembleur_entry_1s_andComment()

```
void assembleur_entry_1s_andComment (   
    assembleur_entry * entry,  
    const char * instr,  
    const char * str,  
    const char * comment )
```

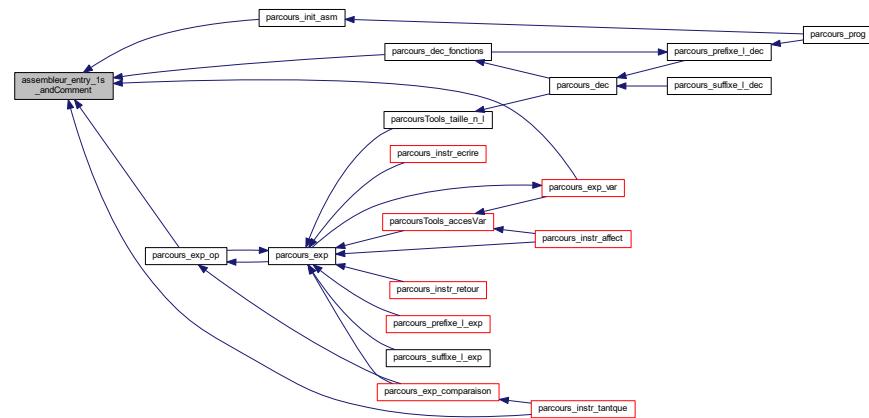
Références assembleur_add_commentary(), et assembleur_entry_1s().

Référencé par parcours_dec_fonctions(), parcours_exp_op(), parcours_exp_var(), parcours_init_asm(), et parcours_instr_tantque().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



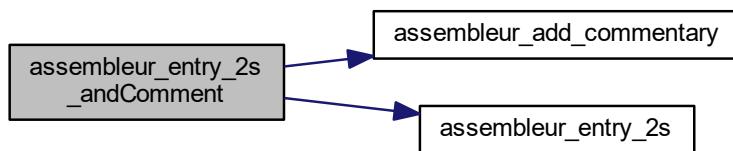
6.11.3.5 assembleur_entry_2s_andComment()

```
void assembleur_entry_2s_andComment (
    assembleur_entry * entry,
    const char * instr,
    const char * str1,
    const char * str2,
    const char * comment )
```

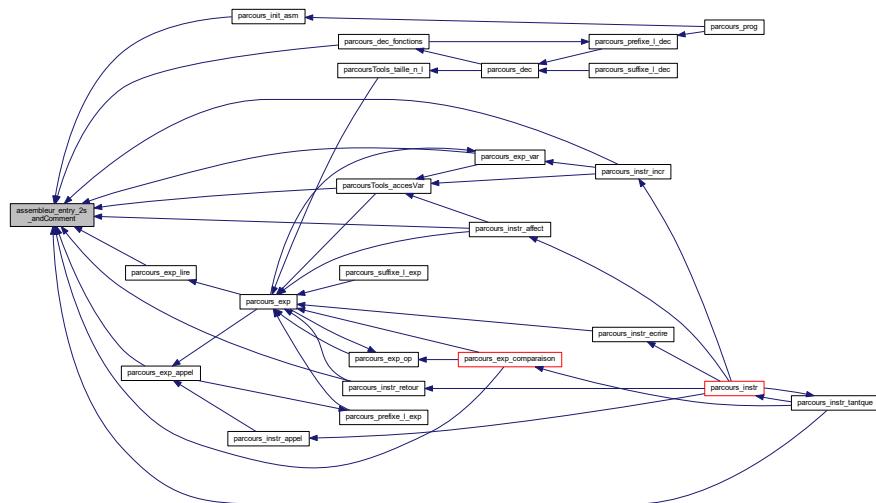
Références assembleur_add_commentary(), et assembleur_entry_2s().

Référencé par parcours_dec_fonctions(), parcours_exp_appel(), parcours_exp_comparaison(), parcours_exp_lire(), parcours_exp_var(), parcours_init_asm(), parcours_instr_affect(), parcours_instr_incr(), parcours_instr_retour(), parcours_instr_tantque(), et parcoursTools_accesVar().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.11.3.6 assembleur_entry_add_subEntry_andComment()

```
void assembleur_entry_add_subEntry_andComment (
```

`assembleur_entry * entry,`

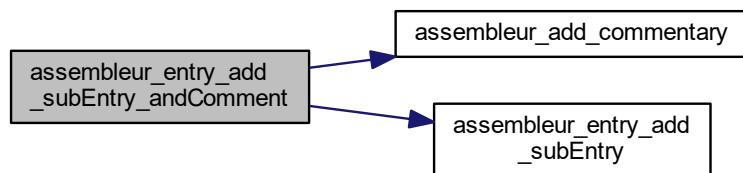
`assembleur_entry * subEntry,`

`const char * comment)`

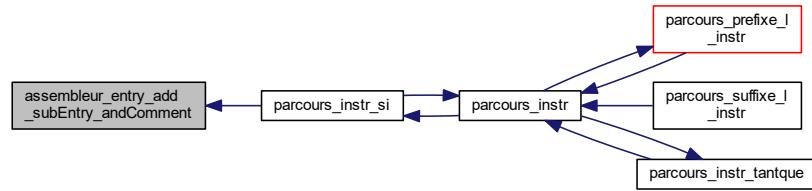
Références assembleur_add_commentary(), et assembleur_entry_add_subEntry().

Référencé par parcours_instr_si().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.11.3.7 assembleur_entry_addEntryPoint_andComment()

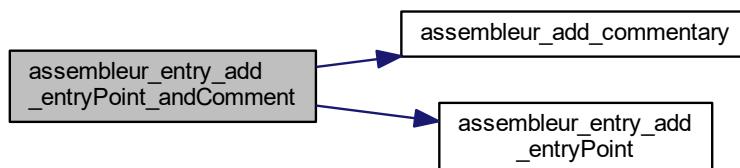
```

void assembleur_entry_addEntryPoint_andComment (
    assembleur_entry * entry,
    assembleur_entry * entryPoint,
    const char * comment )
  
```

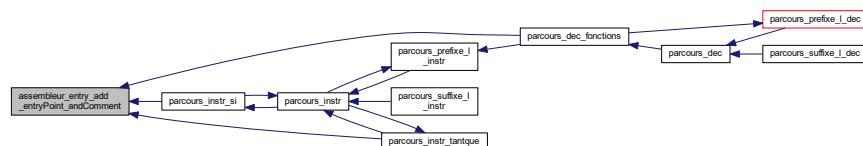
Références assembleur_add_commentary(), et assembleur_entry_addEntryPoint().

Référencé par parcours_dec_fonctions(), parcours_instr_si(), et parcours_instr_tantque().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



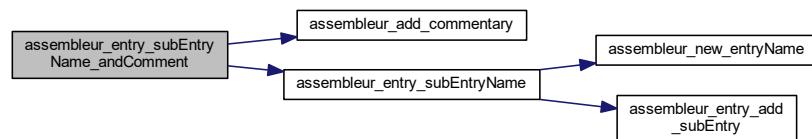
6.11.3.8 assembleur_entry_subEntryName_andComment()

```
assembleur_entry* assembleur_entry_subEntryName_andComment (
    assembleur_entry * entry,
    const char * subEntryName,
    const char * comment )
```

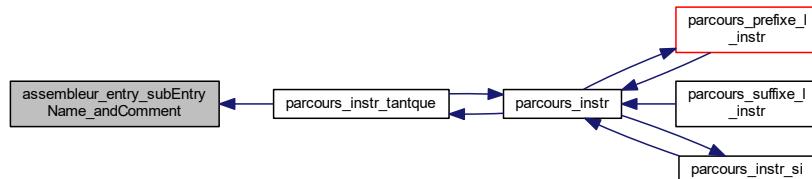
Références assembleur_add_commentary(), et assembleur_entry_subEntryName().

Référencé par parcours_instr_tantque().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



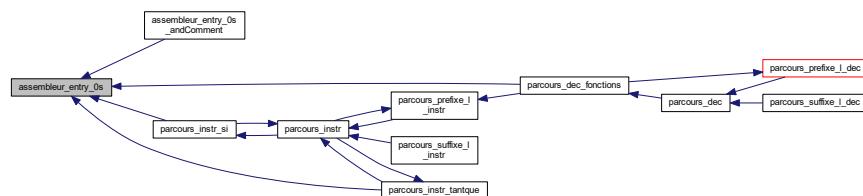
6.11.3.9 assembleur_entry_0s()

```
void assembleur_entry_0s (
    assembleur_entry * entry,
    const char * instr )
///
```

Références ADD_ENTRY, et CHECK_ENTRY.

Référencé par assembleur_entry_0s_andComment(), parcours_dec_fonctions(), parcours_instr_si(), et parcours_instr_tantque().

Voici le graphe des appelants de cette fonction :



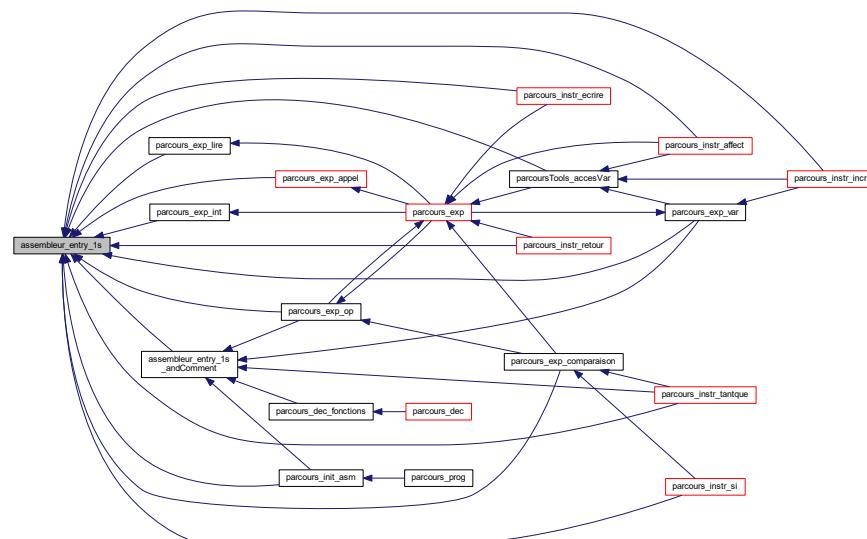
6.11.3.10 assembleur_entry_1s()

```
void assembleur_entry_1s (
    assembleur_entry * entry,
    const char * instr,
    const char * str )
```

Références ADD_ENTRY, CHECK_ENTRY, et SECURE_INST_STR.

Référencé par assembleur_entry_1s_andComment(), parcours_exp_appel(), parcours_exp_comparaison(), parcours_exp_int(), parcours_exp_lire(), parcours_exp_op(), parcours_exp_var(), parcours_init_asm(), parcours_instr_affect(), parcours_instr_ecrire(), parcours_instr_incr(), parcours_instr_retour(), parcours_instr_si(), parcours_instr_tantque(), et parcoursTools_accesVar().

Voici le graphe des appels de cette fonction :



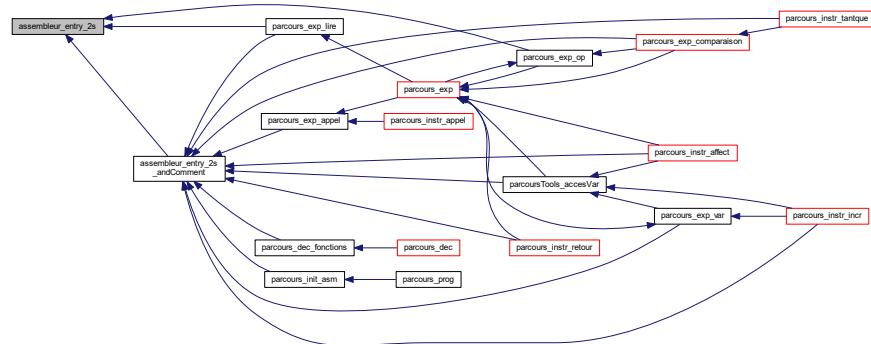
6.11.3.11 assembleur_entry_2s()

```
void assembleur_entry_2s (
    assembleur_entry * entry,
    const char * instr,
    const char * str1,
    const char * str2 )
```

Références ADD_ENTRY, CHECK_ENTRY, et SECURE_INST_STR.

Référencé par assembleur_entry_2s_andComment(), parcours_exp_lire(), et parcours_exp_op().

Voici le graphe des appelants de cette fonction :



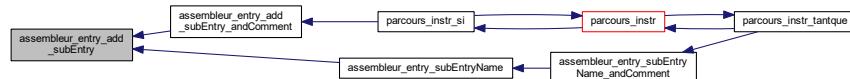
6.11.3.12 assembleur_entry_add_subEntry()

```
void assembleur_entry_add_subEntry (
    assembleur_entry * entry,
    assembleur_entry * subEntry )
```

Références CHECK_ENTRY, circularAdjList_add, et assembleur_entry ::instrs.

Référencé par assembleur_entry_add_subEntry_andComment(), et assembleur_entry_subEntryName().

Voici le graphe des appelants de cette fonction :



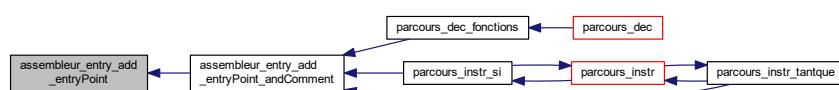
6.11.3.13 assembleur_entry_add_entryPoint()

```
void assembleur_entry_addEntryPoint (
    assembleur_entry * entry,
    assembleur_entry * entryPoint )
```

Références CHECK_ENTRY, circularAdjList_add, et assembleur_entry ::instrs.

Référencé par assembleur_entry_add_entryPoint_andComment().

Voici le graphe des appelants de cette fonction :



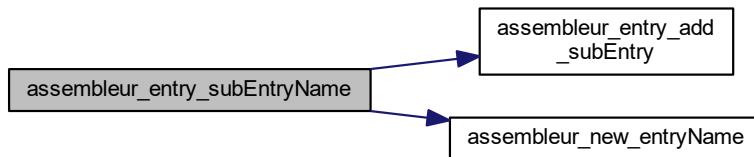
6.11.3.14 assembleur_entry_subEntryName()

```
assembleur_entry* assembleur_entry_subEntryName (
    assembleur_entry * entry,
    const char * subEntryName )
```

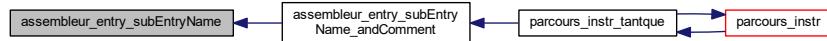
Références assembleur_entry_add_subEntry(), assembleur_new_entryName(), et CHECK_ENTRY.

Référencé par assembleur_entry_subEntryName_andComment().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.11.3.15 assembleur_entry_getLastInstr()

```
const char* assembleur_entry_getLastInstr (
    assembleur_entry * entry )
///
```

Références assembleur_entry : :instrs.

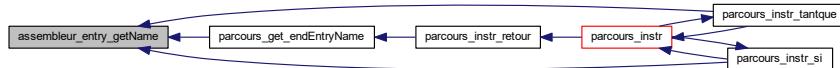
6.11.3.16 assembleur_entry_getName()

```
const char* assembleur_entry_getName (
    assembleur_entry * entry )
```

Références assembleur_entry : :name.

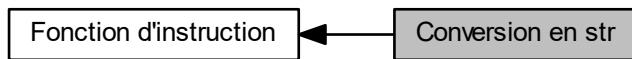
Référencé par parcours_get_endEntryName(), parcours_instr_si(), et parcours_instr_tantque().

Voici le graphe des appels de cette fonction :



6.12 Conversion en str

Graphe de collaboration de Conversion en str :



Macros

```
— #define GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE 0
```

Fonctions

```
— const char * assembleur_make_tabVar (const char *mTab, const char *reg)
— const char * assembleur_make_localVar (const char *reg, const int adr)
— const char * assembleur_make_globalVar (const char *m)
— const char * assembleur_make_constant (const int c)
— const char * assembleur_make_idSuffix (const char *prefix, const unsigned int id)
— const char * assembleur_make_concat (const char *str1, const char *str2)
— const char * assembleur_entry_make_prefix (assembleur_entry *entry, const char *str)
— const char * assembleur_entry_make_suffix (assembleur_entry *entry, const char *str)
— const char * assembleur_entry_make_next (assembleur_entry *entry)
```

6.12.1 Description détaillée

6.12.2 Documentation des macros

6.12.2.1 GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE

```
#define GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE 0
```

Référencé par assembleur_entry_make_next(), assembleur_entry_make_prefix(), et assembleur_entry_make_← suffix().

6.12.3 Documentation des fonctions

6.12.3.1 assembleur_make_tabVar()

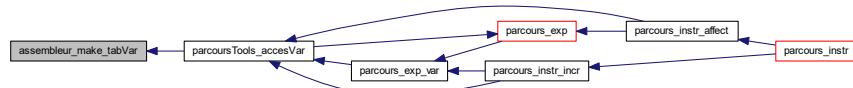
```
const char* assembleur_make_tabVar (
    const char * mTab,
    const char * reg )
```

A faire gerer la désallocation

Références SECURE_INST_STR.

Référencé par parcoursTools_accesVar().

Voici le graphe des appelants de cette fonction :



6.12.3.2 assembleur_make_localVar()

```
const char* assembleur_make_localVar (
    const char * reg,
    const int adr )
```

A faire gerer la désallocation

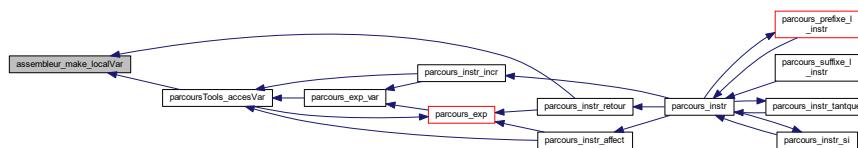
Références printedSizeOfUnsigned(), et SECURE_INST_STR.

Référencé par parcours_instr_retour(), et parcoursTools_accesVar().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.12.3.3 assembleur_make_globalVar()

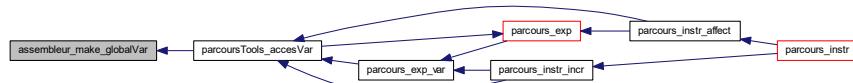
```
const char* assembleur_make_globalVar (
    const char * m )
```

A faire gerer la désallocation

Références SECURE_INST_STR.

Référencé par parcoursTools_accesVar().

Voici le graphe des appelants de cette fonction :



6.12.3.4 assembleur_make_constant()

```
const char* assembleur_make_constant (
    const int c )
```

A faire gerer la désallocation

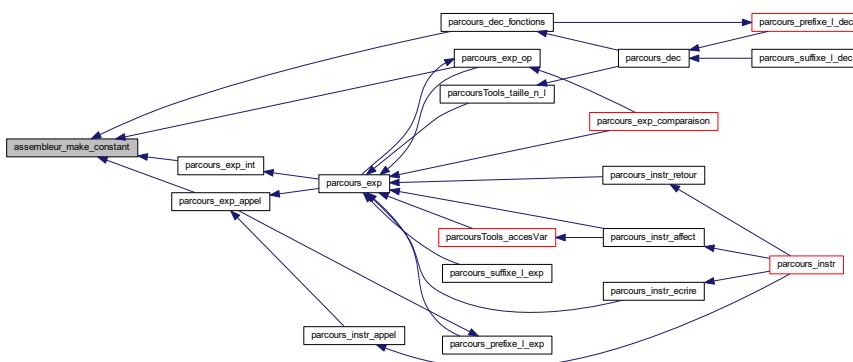
Références printedSizeOfSigned(), et SECURE_INST_STR.

Référencé par parcours_dec_fonctions(), parcours_exp_appel(), parcours_exp_int(), et parcours_exp_op().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



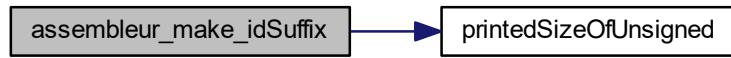
6.12.3.5 assembleur_make_idSuffix()

```
const char* assembleur_make_idSuffix (
    const char * prefix,
    const unsigned int id )
```

A faire gerer la désallocation

Références printedSizeOfUnsigned(), et SECURE_INST_STR.

Voici le graphe d'appel pour cette fonction :



6.12.3.6 assembleur_make_concat()

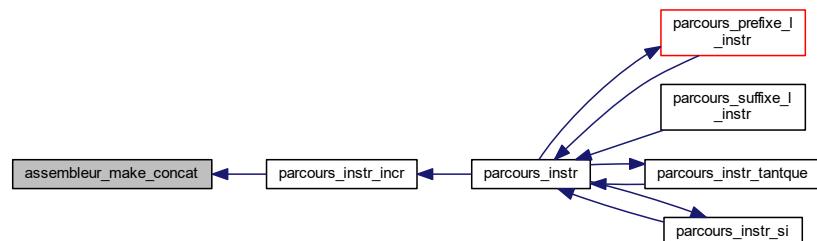
```
const char* assembleur_make_concat (
    const char * str1,
    const char * str2 )
```

A faire gerer la désallocation

Références SECURE_INST_STR.

Référencé par parcours_instr_incr().

Voici le graphe des appelants de cette fonction :



6.12.3.7 assembleur_entry_make_prefix()

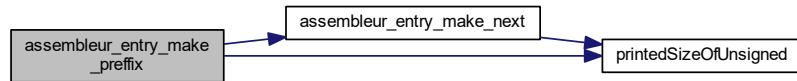
```
const char* assembleur_entry_make_prefix (
    assembleur_entry * entry,
    const char * str )
```

A faire gerer la désallocation

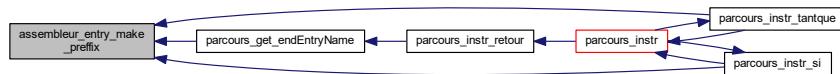
Références assembleur_entry_make_next(), assembleur_entry : :counter, GENASM_ENTRYCOUNTER_SEP←ERATE_BY_UNDERSCORE, GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE, assembleur_entry : :name, printedSizeOfUnsigned(), et SECURE_INST_STR.

Référencé par parcours_get_endEntryName(), parcours_instr_si(), et parcours_instr_tantque().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



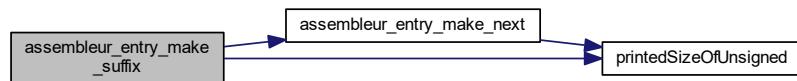
6.12.3.8 assembleur_entry_make_suffix()

```
const char* assembleur_entry_make_suffix (
    assembleur_entry * entry,
    const char * str )
```

A faire gerer la désallocation

Références assembleur_entry_make_next(), assembleur_entry : :counter, GENASM_ENTRYCOUNTER_SEP←ERATE_BY_UNDERSCORE, GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE, assembleur_entry : :name, printedSizeOfUnsigned(), et SECURE_INST_STR.

Voici le graphe d'appel pour cette fonction :



6.12.3.9 assembleur_entry_make_next()

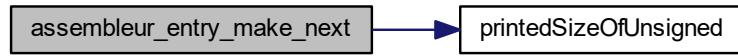
```
const char* assembleur_entry_make_next (
    assembleur_entry * entry )
```

A faire gerer la désallocation

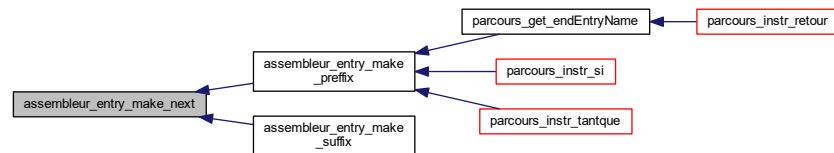
Références assembleur_entry : :counter, GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE, GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE, assembleur_entry : :name, printedSizeOfUnsigned(), et SECURE_INST_STR.

Référencé par assembleur_entry_make_prefix(), et assembleur_entry_make_suffix().

Voici le graphe d'appel pour cette fonction :

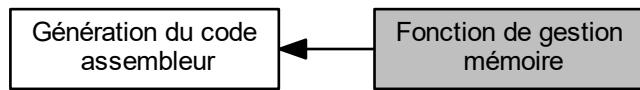


Voici le graphe des appels de cette fonction :



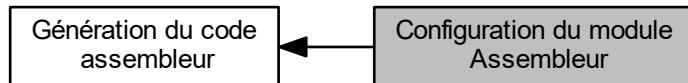
6.13 Fonction de gestion mémoire

Graphe de collaboration de Fonction de gestion mémoire :



6.14 Configuration du module Assembleur

Graphe de collaboration de Configuration du module Assembleur :



Macros

- #define GENASM_ENTRY_PRETTY_NAME 1
- #define GENASM_ENTRY_ARE_INDENT 1
- #define GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE 0
- #define GENASM_ENTRYPOINT_SEPERATE_RETURNLINE 0

6.14.1 Description détaillée

6.14.2 Documentation des macros

6.14.2.1 GENASM_ENTRY_PRETTY_NAME

```
#define GENASM_ENTRY_PRETTY_NAME 1
```

6.14.2.2 GENASM_ENTRY_ARE_INDENT

```
#define GENASM_ENTRY_ARE_INDENT 1
```

Note

se justifie esthétiquement...

6.14.2.3 GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE

```
#define GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE 0
```

Note

N'est pas très utile, mais peut se justifier esthétiquement...

Référencé par assembleur_entry_make_next(), assembleur_entry_make_prefix(), et assembleur_entry_make_← suffix().

6.14.2.4 GENASM_ENTRYPOINT_SEPERATE_RETURNLINE

```
#define GENASM_ENTRYPOINT_SEPERATE_RETURNLINE 0
```

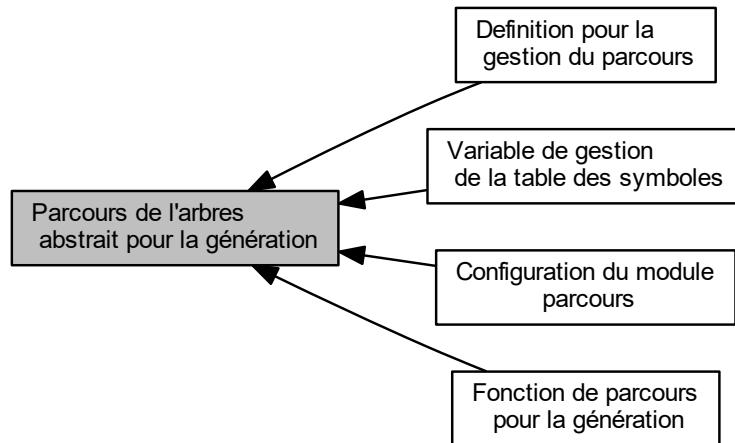
Note

N'est pas très utile, mais peut se justifier esthétiquement...

6.15 Parcours de l'arbres abstrait pour la génération

Partie création de la Table des symboles et de l'arbre abstrait.

Graphe de collaboration de Parcours de l'arbres abstrait pour la génération :



Modules

- Variable de gestion de la table des symboles
- Definition pour la gestion du parcours
- Fonction de parcours pour la génération
- Configuration du module parcours

6.15.1 Description détaillée

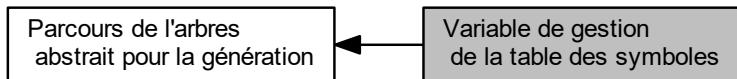
Partie création de la Table des symboles et de l'arbre abstrait.

Auteur

Alexis Nasr (<http://pageperso.lif.univ-mrs.fr/~alexis.nasr/>)
released by Yannick Robin
released by Christophe Sonntag (<http://u4a.at>)

6.16 Variable de gestion de la table des symboles

Graphe de collaboration de Variable de gestion de la table des symboles :



Variables

- int `adresseGlobalCourant`
- int `adresseLocaleCourante`
- int `adresseArgumentCourant`
- `assembleur_entry * ce` = NULL
- `assembleur_entry * topEntry` = NULL
- `assembleur_entry * endEntry` = NULL

6.16.1 Description détaillée

6.16.2 Documentation des variables

6.16.2.1 `adresseGlobalCourant`

```
int adresseGlobalCourant
```

Référencé par `parcours_dec()`, et `parcours_prog()`.

6.16.2.2 `adresseLocaleCourante`

```
int adresseLocaleCourante
```

Référencé par `parcours_dec()`, et `tabsymb_entreeFonction()`.

6.16.2.3 `adresseArgumentCourant`

```
int adresseArgumentCourant
```

Référencé par `parcours_dec()`, et `tabsymb_entreeFonction()`.

6.16.2.4 ce

```
assembleur_entry* ce = NULL
```

c'est l'entrée courante

Référencé par parcours_instr_si(), parcours_instr_tantque(), parcoursTools_accesVar(), et parcoursTools_taille←n_l().

6.16.2.5 topEntry

```
assembleur_entry* topEntry = NULL
```

entrée d'une fonction

Référencé par parcours_dec_fonctions().

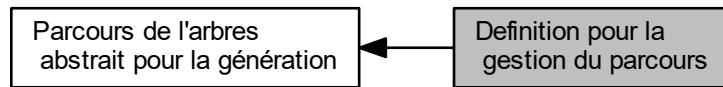
6.16.2.6 endEntry

```
assembleur_entry* endEntry = NULL
```

entrée de fin si un ou, plusieurs 'retour' ont eu lieu

6.17 Definition pour la gestion du parcours

Graphe de collaboration de Definition pour la gestion du parcours :



Macros

- #define `ErreursiNulle(var)` if(`var==NULL`) { `erreurArgs("La fonction '%s' n'accepte pas de valeur nulle", __FUNCTION__);` }
- #define `ErreursiMauvaisType(var, leTypeVoulut)` if(`var->type != leTypeVoulut`) { `erreurArgs("La fonction '%s' necessite le type '%s'", __FUNCTION__, #leTypeVoulut);` }

6.17.1 Description détaillée

6.17.2 Documentation des macros

6.17.2.1 ErreursiNulle

```
#define ErreursiNulle(
    var ) if (var==NULL) { erreurArgs ("La fonction '%s' n'accepte pas de valeur nulle",
    __FUNCTION__); }
```

Référencé par `parcours_dec()`, `parcours_dec_fonctions()`, `parcours_exp()`, `parcours_exp_appel()`, `parcours_exp_op()`, `parcours_exp_var()`, `parcours_get_endEntryName()`, `parcours_instr()`, `parcours_instr_affect()`, `parcours_instr_appel()`, `parcours_instr_ecrire()`, `parcours_instr_incr()`, `parcours_instr_retour()`, `parcours_instr_si()`, `parcours_instr_tantque()`, et `parcours_prog()`.

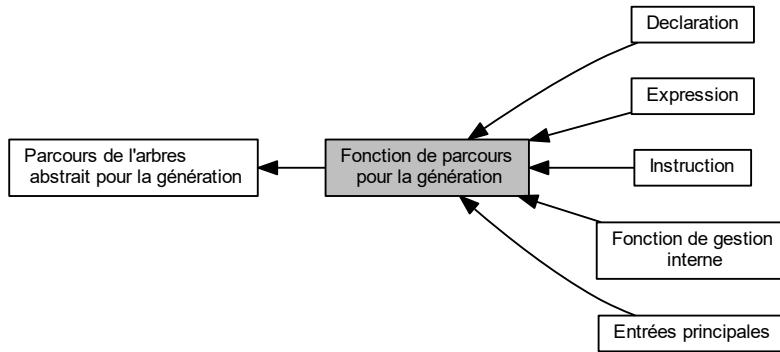
6.17.2.2 ErreursiMauvaisType

```
#define ErreursiMauvaisType(
    var,
    leTypeVoulut ) if (var->type!=leTypeVoulut) { erreurArgs ("La fonction '%s' necessite
    le type '%s'", __FUNCTION__, #leTypeVoulut ); }
```

Référencé par `parcours_dec_fonctions()`, et `parcours_exp_op()`.

6.18 Fonction de parcours pour la génération

Graphe de collaboration de Fonction de parcours pour la génération :



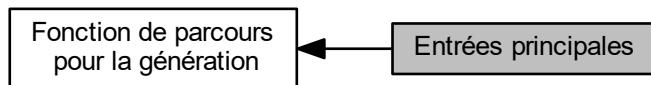
Modules

- Entrées principales
- Fonction de gestion interne
- Declaration
- Instruction
- Expression

6.18.1 Description détaillée

6.19 Entrées principales

Graphe de collaboration de Entrées principales :



Fonctions

- void [parcours_prog](#) (const n_prog *n)
parcours_prog : Génère la table des symbole et le code assembleur
- void [parcours_init_asm](#) ()

6.19.1 Description détaillée

6.19.2 Documentation des fonctions

6.19.2.1 parcours_prog()

```
void parcours_prog (
    const n_prog * n )
```

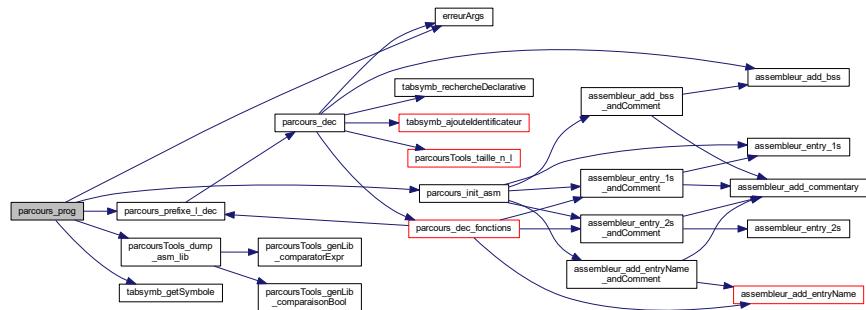
`parcours_prog` : Génère la table des symbole et le code assembleur

Paramètres

<i>n</i>	: un arbre abstrait
----------	---------------------

Références `adresseGlobalCourant`, `erreurArgs()`, `ErreursSiNulle`, `P_VARIABLE_GLOBALE`, `parcours_init_asm()`, `parcours_prefixe_l_dec()`, `parcoursTools_dump_asm_lib()`, `T_FONCTION`, `tabsymb_getSymbole()`, et `desc_← identif : type`.

Voici le graphe d'appel pour cette fonction :



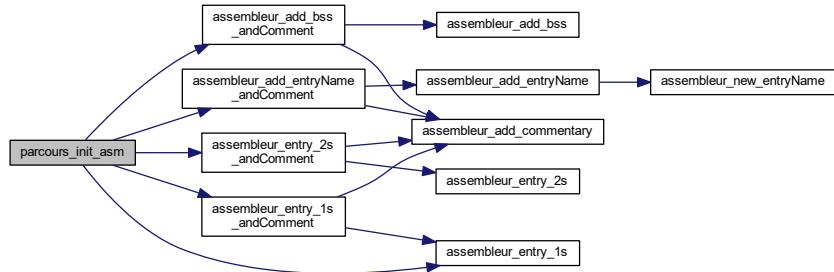
6.19.2.2 parcours_init_asm()

```
void parcours_init_asm ( )
```

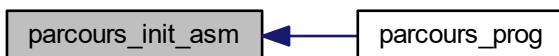
Références assembleur_add_bss_andComment(), assembleur_add_entryName_andComment(), assembleur__← entry_1s(), assembleur_entry_1s_andComment(), assembleur_entry_2s_andComment(), et resb.

Référencé par parcours_prog().

Voici le graphe d'appel pour cette fonction :

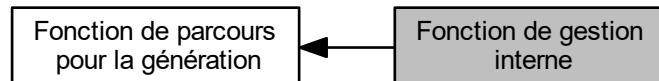


Voici le graphe des appels de cette fonction :



6.20 Fonction de gestion interne

Graphe de collaboration de Fonction de gestion interne :



Fonctions

- const char * **parcours_get_endEntryName ()**
parcours_get_endEntryName genère une entrée ('assembleur_entry') de type 'entryPoint' conçue pour être placée en fin de fonction.

6.20.1 Description détaillée

6.20.2 Documentation des fonctions

6.20.2.1 parcours_get_endEntryName()

```
const char* parcours_get_endEntryName ( )
```

parcours_get_endEntryName genère une entrée ('assembleur_entry') de type 'entryPoint' conçue pour être placée en fin de fonction.

Note

Nécessaire à l'implémantation de l'instruction 'retour' du langage L

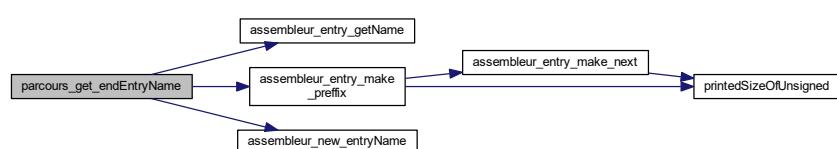
Renvoie

retourne le nom du 'entryPoint' disponible en fin de fonction.

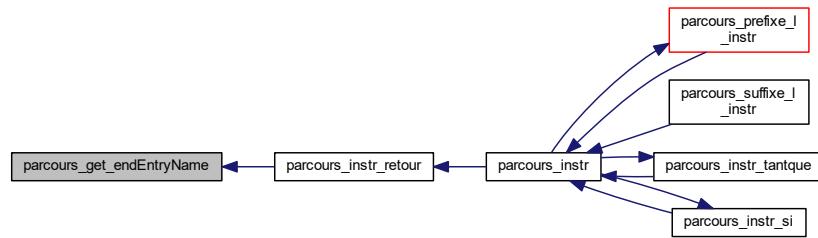
Références assembleur_entry_getName(), assembleur_entry_make_prefix(), assembleur_new_entryName(), et ErreurSiNulle.

Référencé par parcours_instr_retour().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.21 Declaration

Graphe de collaboration de Declaration :



Fonctions

- unsigned int [parcours_prefixe_l_dec](#) (const n_l_dec *n, int porte)
 - unsigned int [parcours_suffixe_l_dec](#) (const n_l_dec *n, int porte)
 - void [parcours_dec](#) (const n_dec *n, int porte)
- parcours_dec Pour le moment, les seules déclarations prises en compte sont les variables globales. Lors du parcours d'un noeud correspondant à une variable globale, il faut allouer de la place pour cette variable dans la région .bss du code machine, à l'aide des pseudo instructions resb, resw ..*
- void [parcours_dec_fonctions](#) (const n_dec *n)

6.21.1 Description détaillée

6.21.2 Documentation des fonctions

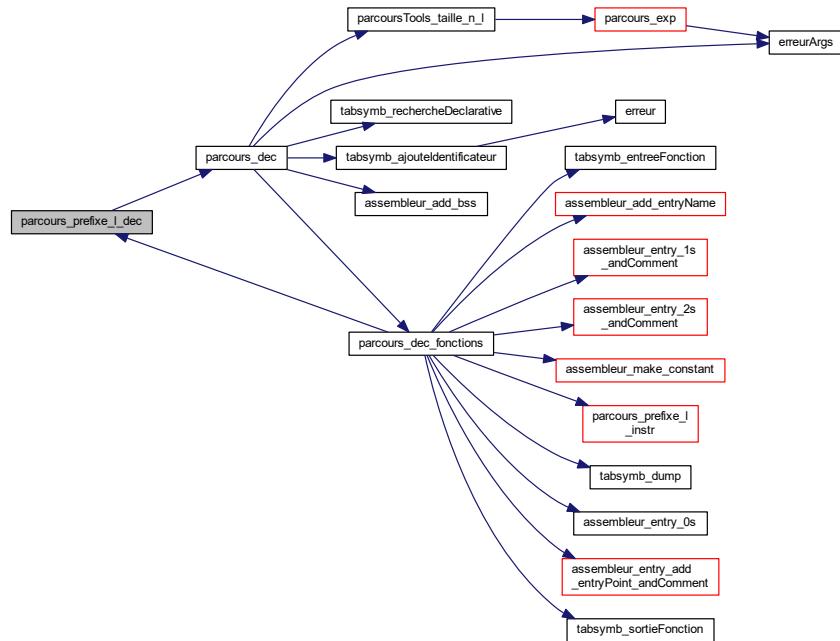
6.21.2.1 parcours_prefixe_l_dec()

```
unsigned int parcours_prefixe_l_dec (
    const n_l_dec * n,
    int porte )
```

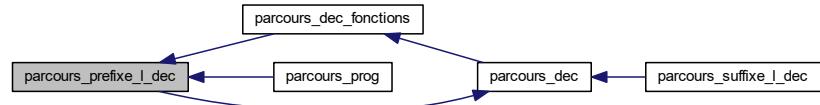
Références [parcours_dec\(\)](#).

Référencé par [parcours_dec_fonctions\(\)](#), et [parcours_prog\(\)](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



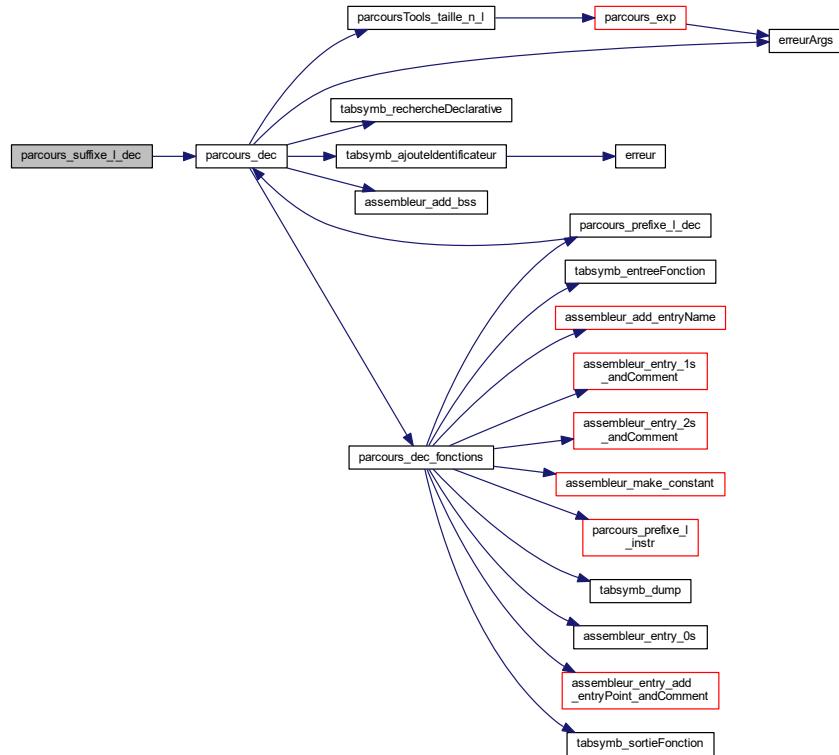
6.21.2.2 parcours_suffixe_l_dec()

```

unsigned int parcours_suffixe_l_dec (
    const n_l_dec * n,
    int porte )
  
```

Références parcours_dec().

Voici le graphe d'appel pour cette fonction :



6.21.2.3 parcours_dec()

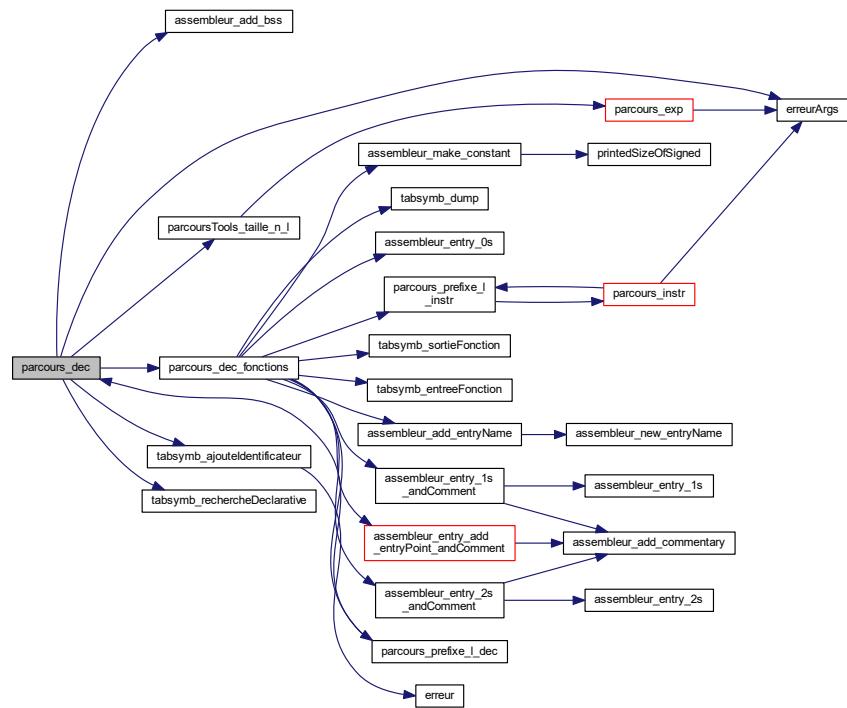
```
void parcours_dec (
    const n_dec * n,
    int porte )
```

parcours_dec Pour le moment, les seules déclarations prises en compte sont les variables globales. Lors du parcours d'un noeud correspondant à une variable globale, il faut allouer de la place pour cette variable dans la région .bss du code machine, à l'aide des pseudo instructions resb, resw ...

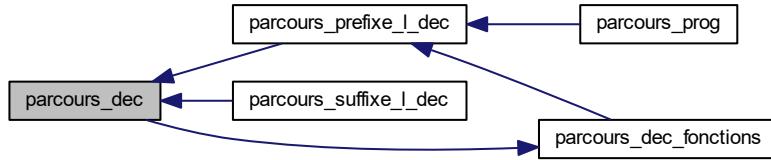
Références adresseArgumentCourant, adresseGlobalCourant, adresseLocaleCourante, assembleur_add_bss(), erreurArgs(), ErreurSiNulle, P_ARGUMENT, P_VARIABLE_GLOBALE, P_VARIABLE_LOCALE, parcours_dec_fonctions(), parcoursTools_taille_n_l(), rest, T_ENTIER, T_FONCTION, T_TABLEAU_ENTIER, tabsymb_ajouteIdentificateur(), et tabsymb_rechercheDeclarative().

Référencé par parcours_prefixe_l_dec(), et parcours_suffixe_l_dec().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.21.2.4 parcours_dec_fonctions()

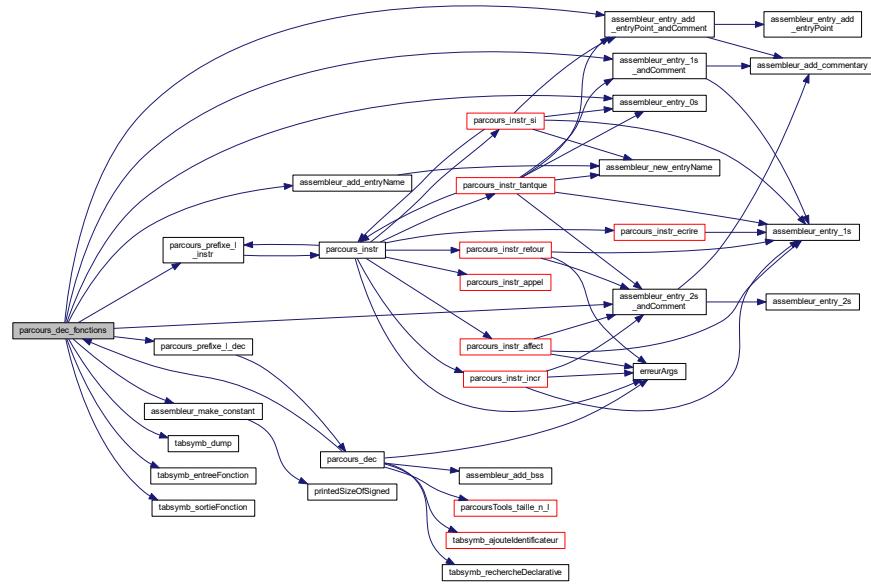
```
void parcours_dec_fonctions (
    const n_dec * n )
```

A faire tester si cela ne fait pas de bug avec 'topEntry' plutot que 'ce'

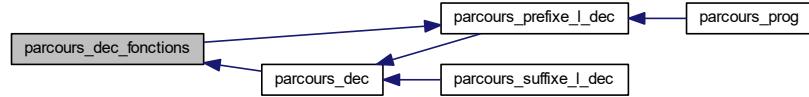
Références assembleur_add_entryName(), assembleur_entry_0s(), assembleur_entry_1s_andComment(), assembleur_entry_2s_andComment(), assembleur_entry_addEntryPoint_andComment(), assembleur_makeConstant(), ErreurSiMauvaisType, ErreurSiNulle, P_ARGUMENT, P_VARIABLE_LOCALE, parcours_prefixe_l_dec(), parcours_prefixe_l_instr(), tabsymb_dump(), tabsymb_entreeFonction(), tabsymb_sortieFonction(), et topEntry.

Référencé par parcours_dec().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.22 Instruction

Graphe de collaboration de Instruction :



Fonctions

- unsigned int `parcours_prefixe_l_instr` (const `n_l_instr` **n*)
- unsigned int `parcours_suffixe_l_instr` (const `n_l_instr` **n*)
- void `parcours_instr` (const `n_instr` **n*)

`parcours_instr` : Seules les instructions de type `ecrire` et `affect` seront traitées pendant ce TP. Une instruction `ecrire` doit être traduite par un appel système `int 0x80`, une affectation stocke la valeur d'un registre (qui contient le résultat de `parcours_exp`) dans une adresse mémoire. Les adresses des variables simples peuvent être représentées sous forme d'étiquette, tandis que les adresses des tableaux sont aussi des expressions dont la valeur doit être calculée.

- void `parcours_instr_ecrire` (const `n_exp` **n*)
- void `parcours_instr_affect` (const `n_var` **var*, const `n_exp` **exp*)
- void `parcours_instr_incr` (const `n_var` **var*)
- void `parcours_instr_tantque` (const `n_exp` **test*, const `n_instr` **faire*)
- void `parcours_instr_appel` (const `n_appel` **appel*)
- void `parcours_instr_si` (const `n_exp` **test*, const `n_instr` **alors*, const `n_instr` **sinon*)
- void `parcours_instr_retour` (const `n_exp` **expression*)

6.22.1 Description détaillée

6.22.2 Documentation des fonctions

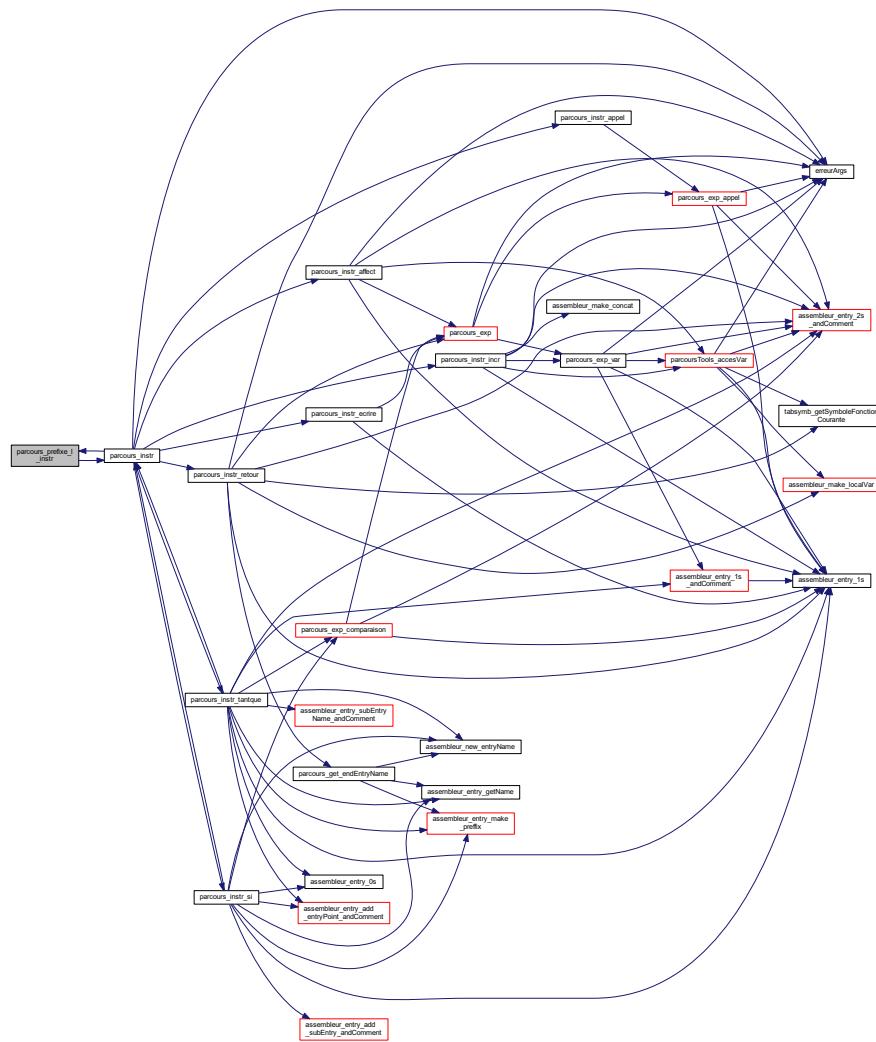
6.22.2.1 `parcours_prefixe_l_instr()`

```
unsigned int parcours_prefixe_l_instr (
    const n_l_instr * n )
```

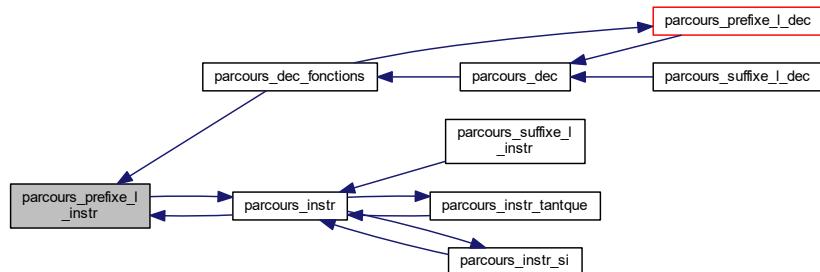
Références `parcours_instr()`.

Référencé par `parcours_dec_fonctions()`, et `parcours_instr()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :

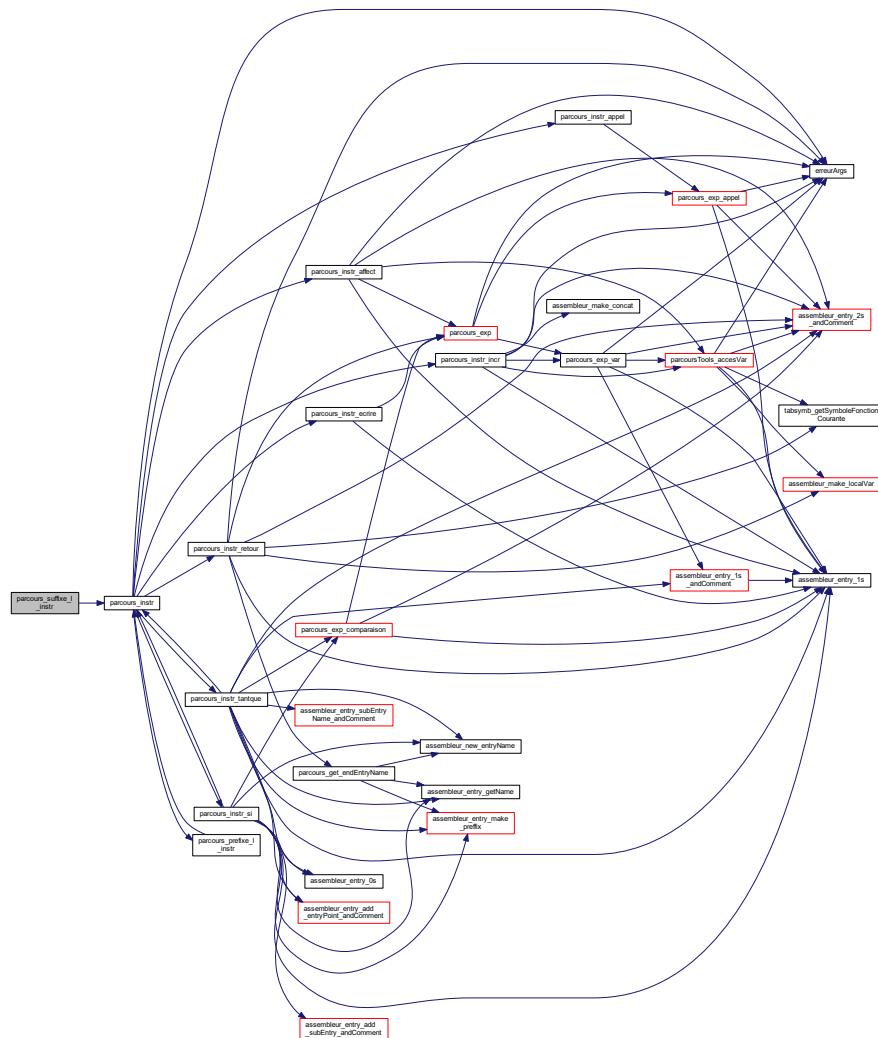


6.22.2.2 parcours_suffixe_l_instr()

```
unsigned int parcours_suffixe_l_instr (
```

Références parcours_instr().

Voici le graphe d'appel pour cette fonction :



6.22.2.3 parcours instr()

```
void parcours_instr (
```

`parcours_instr` : Seules les instructions de type écrire et affecter seront traitées pendant ce TP. Une instruction écrire doit être traduite par un appel système `int 0x80`, une affectation stocke la valeur d'un registre (qui contient le résultat de `parcours_exp`) dans une adresse mémoire. Les adresses des variables globales simples peuvent être représentées sous forme d'étiquette, tandis que les adresses des tableaux sont aussi des expressions dont la valeur doit être calculée.

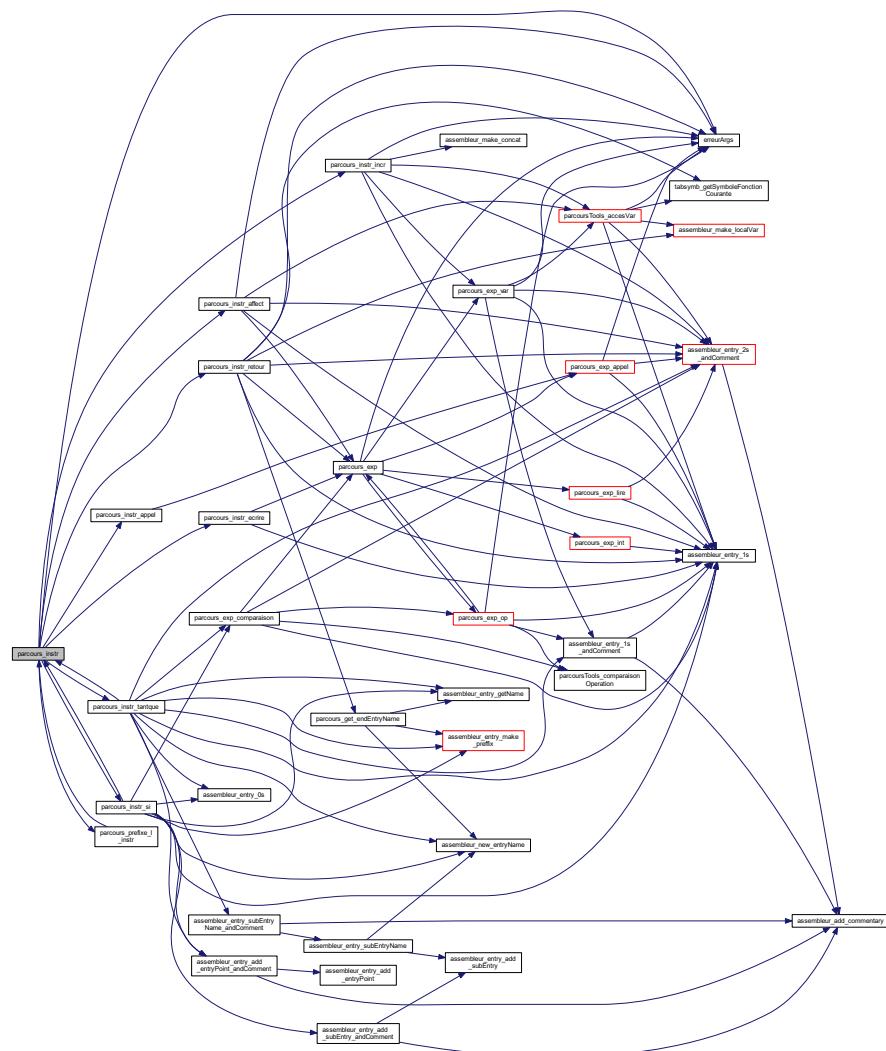
A faire faire 'instruction faire'

A faire voir pour 'instruction incrInst'

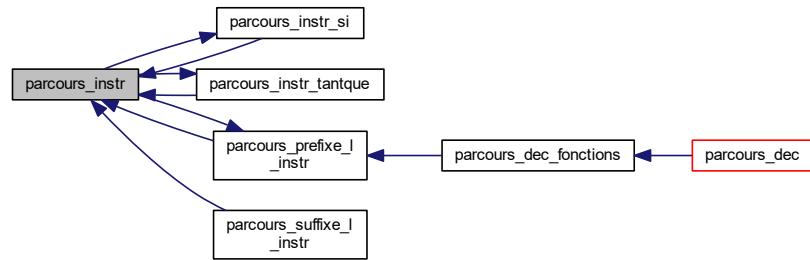
Références erreurArgs(), ErreurSiNulle, parcours_instr_affect(), parcours_instr_appel(), parcours_instr_ecrire(), parcours_instr_incr(), parcours_instr_retour(), parcours_instr_si(), parcours_instr_tantque(), et parcours_prefixe←_l_instr().

Référencé par parcours_instr_si(), parcours_instr_tantque(), parcours_prefixe_l_instr(), et parcours_suffixe_l←_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



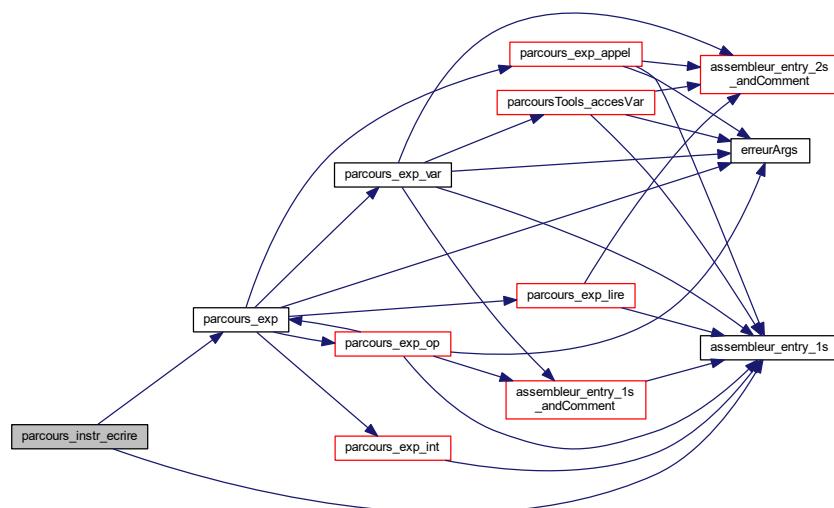
6.22.2.4 parcours_instr_ecrire()

```
void parcours_instr_ecrire (
    const n_exp * n )
```

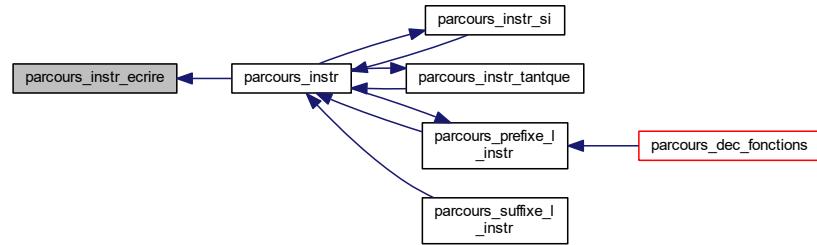
Références assembleur_entry_1s(), ErreurSiNulle, et parcours_exp().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



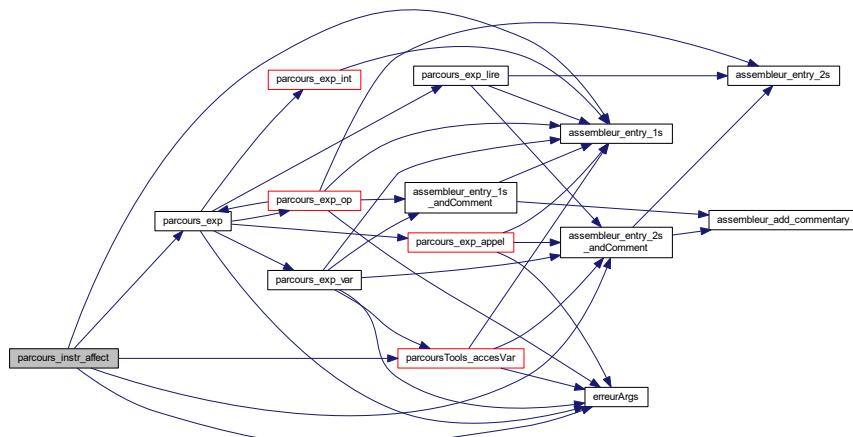
6.22.2.5 parcours_instr_affect()

```
void parcours_instr_affect (
    const n_var * var,
    const n_exp * exp )
```

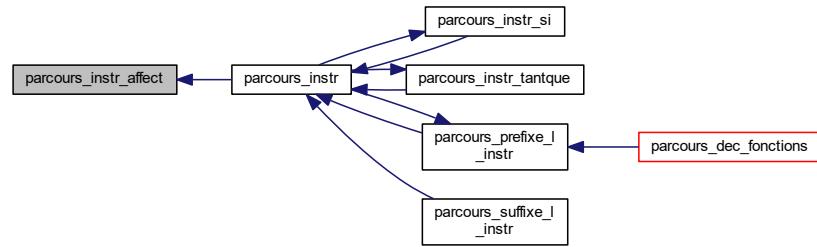
Références assembleur_entry_1s(), assembleur_entry_2s_andComment(), erreurArgs(), ErreurSiNulle, parcours_exp(), et parcoursTools_accesVar().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



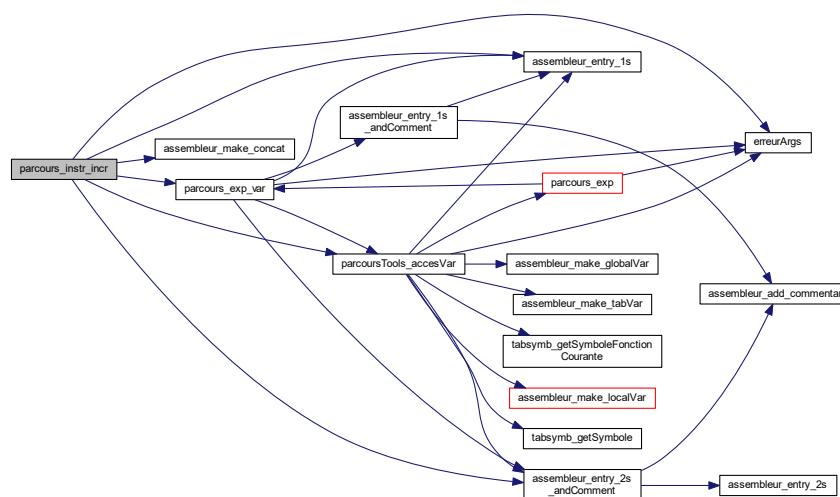
6.22.2.6 parcours_instr_incr()

```
void parcours_instr_incr (
    const n_var * var )
```

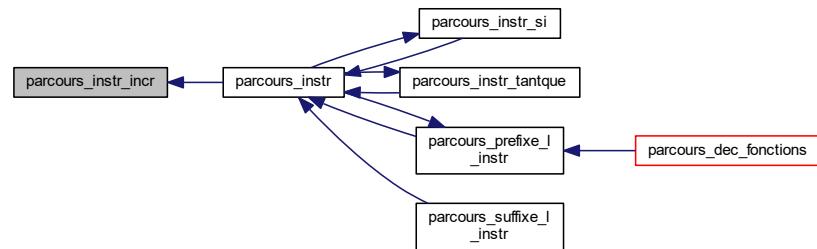
Références assembleur_entry_1s(), assembleur_entry_2s_andComment(), assembleur_make_concat(), erreurArgs(), ErreurSiNulle, parcours_exp_var(), et parcoursTools_accesVar().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.22.2.7 parcours_instr_tantque()

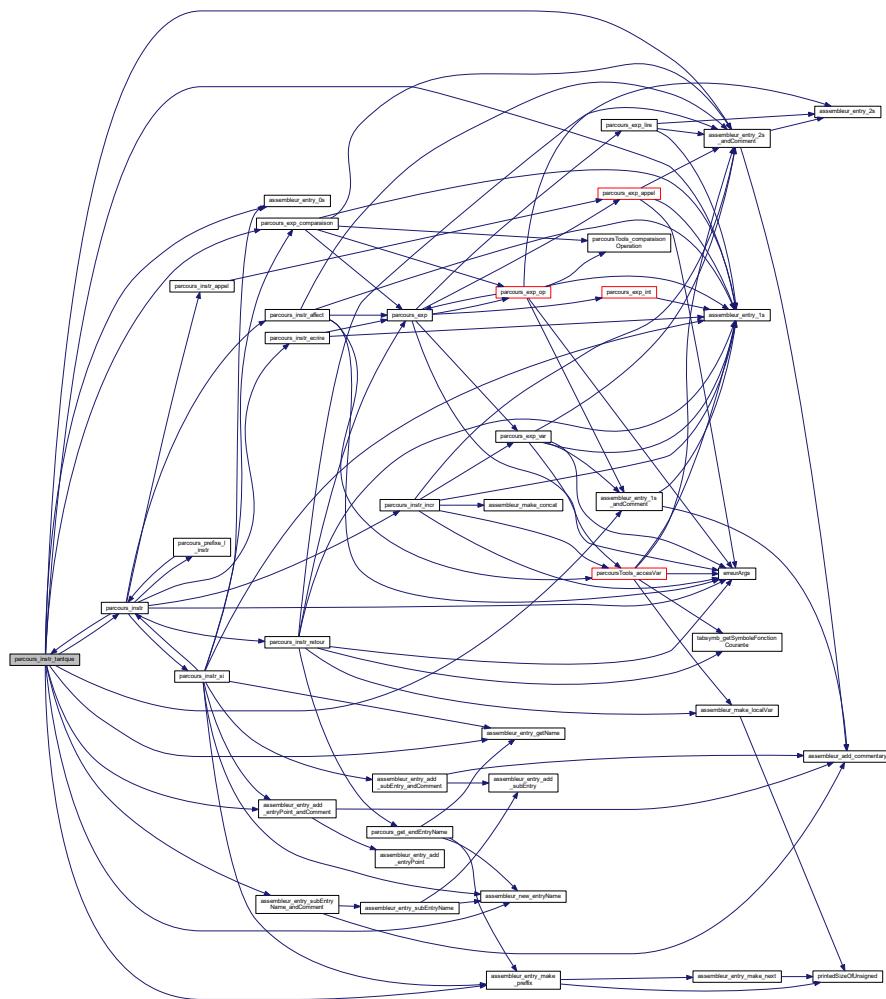
```

void parcours_instr_tantque (
    const n_exp * test,
    const n_instr * faire )
  
```

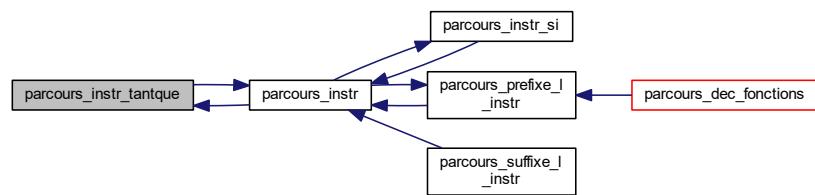
Références assembleur_entry_0s(), assembleur_entry_1s(), assembleur_entry_1s_andComment(), assembleur_entry_2s_andComment(), assembleur_entry_add_entryPoint_andComment(), assembleur_entry_getName(), assembleur_entry_make_prefix(), assembleur_entry_subEntryName_andComment(), assembleur_new_entryName(), ce, ErreurSiNulle, parcours_exp_comparaison(), et parcours_instr().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



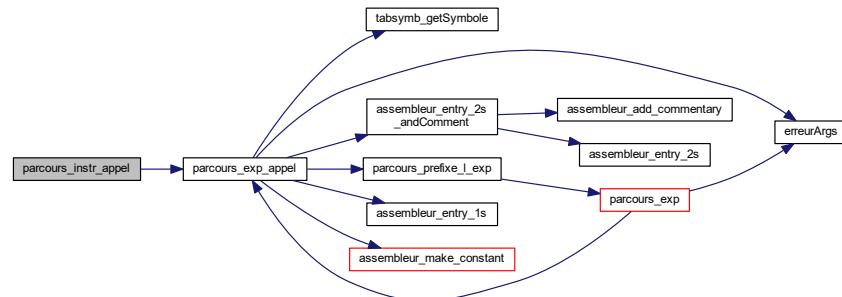
6.22.2.8 parcours_instr_appel()

```
void parcours_instr_appel (
```

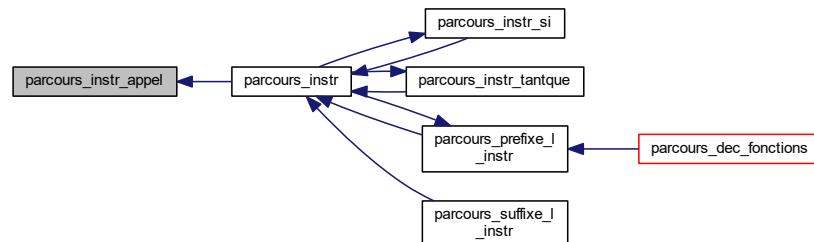
Références ErreurSiNulle, et parcours_exp_appel().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.22.2.9 parcours_instr_si()

```

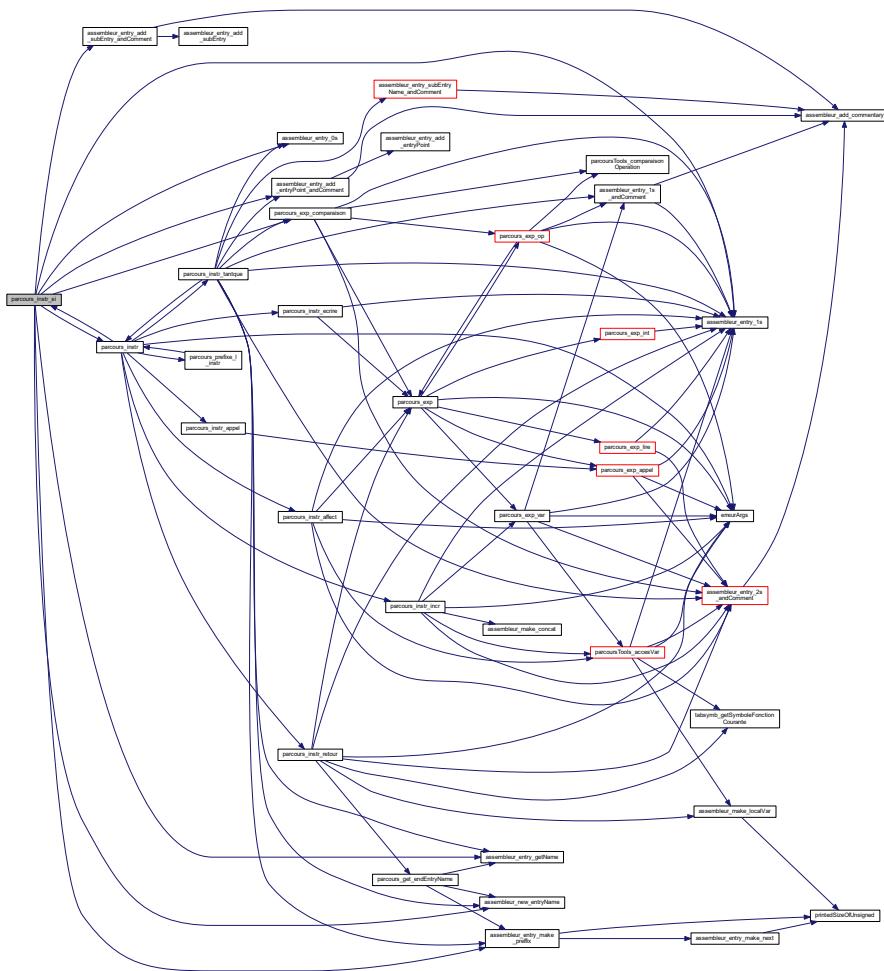
void parcours_instr_si (
    const n_exp * test,
    const n_instr * alors,
    const n_instr * sinon )
    
```

A faire voir si utile de generer le bloc inutilisé dans le cas d'un constexpr par exemple, si la condition ne fait que valider, alors ne pas generer le sinon s'il existe ...

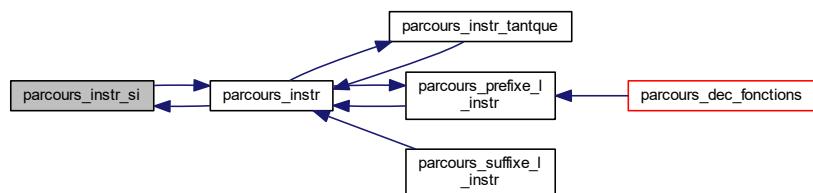
Références assembleur_entry_0s(), assembleur_entry_1s(), assembleur_entry_addEntryPoint_andComment(), assembleur_entry_add_subEntry_andComment(), assembleur_entry_getName(), assembleur_entry_make_prefix(), assembleur_new_entryName(), ce, ErreurSiNulle, parcours_exp_comparaison(), et parcours_instr().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



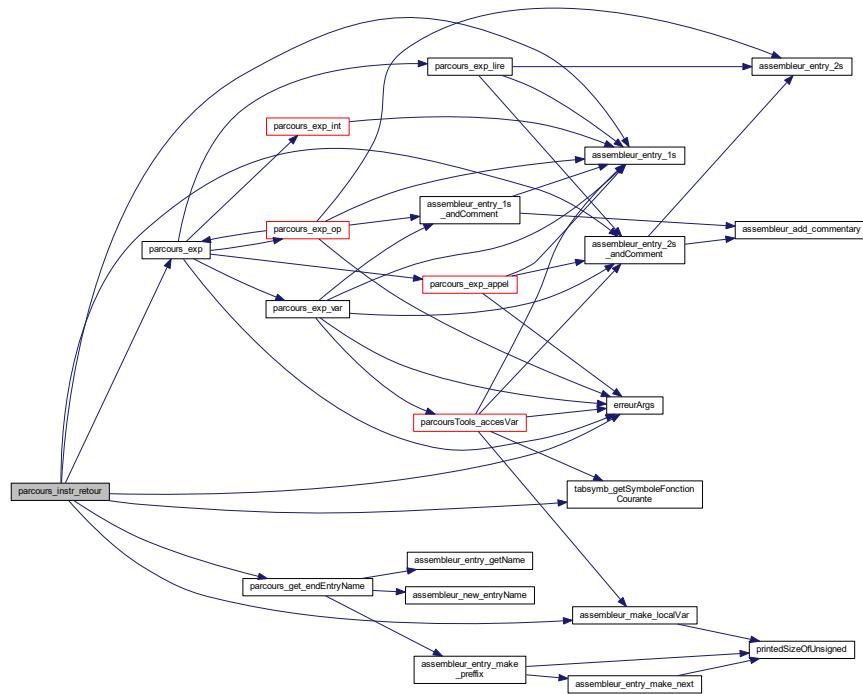
6.22.2.10 parcours instr retour()

```
void parcours_instr_retour (
```

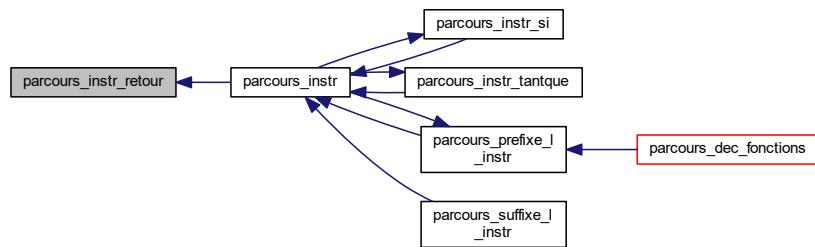
Références assembleur_entry_1s(), assembleur_entry_2s_andComment(), assembleur_make_localVar(), desc_← identif : :complement, erreurArgs(), ErreurSiNulle, parcours_exp(), parcours_get_endEntryName(), et tabsymb_← getSymboleFonctionCourante().

Référencé par parcours_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.23 Expression

Graphe de collaboration de Expression :



Fonctions

- unsigned int `parcours_prefixe_l_exp` (const `n_l_exp` *`n`)
- unsigned int `parcours_suffixe_l_exp` (const `n_l_exp` *`n`)
- void `parcours_exp` (const `n_exp` *`n`)

parcours_exp : Cette fonction génère les instructions correspondant au calcul de la valeur de l'expression et dépose le résultat du calcul dans la pile. La génération de code pour une opération de type arg1 op arg2 consiste à affecter à des registres les deux opérandes arg1 et arg2 puis à réaliser l'opération à l'aide de l'instruction correspondante et enfin à empiler le résultat de l'opération. Pour les instructions complexes, les valeurs des opérandes (sous expressions) seront prises dans la pile, c'est là qu'aura été déposé le résultat de ces sous expressions. Il faut aussi traiter les lectures de variables lire avec un appel système.
- int `parcours_exp_comparaison` (const `n_exp` *`n`, const char **`comparaisonOperationPtr`)

parcours_exp_comparaison : est executer a partir d'une fonction de comparaison (pas d'opération) permet de traduire n'importe quelle expression en comparaison (comparaison, test vrai...)
- void `parcours_exp_lire` ()
- void `parcours_exp_var` (const `n_var` *`variable`)
- void `parcours_exp_int` (const int entier)
- void `parcours_exp_appel` (const `n_appel` *`appel`, const bool `desallocReturnValue`)
- int `parcours_exp_op` (const `n_exp` *`n`, const bool `isComparaison`)

parcours_exp_op

6.23.1 Description détaillée

6.23.2 Documentation des fonctions

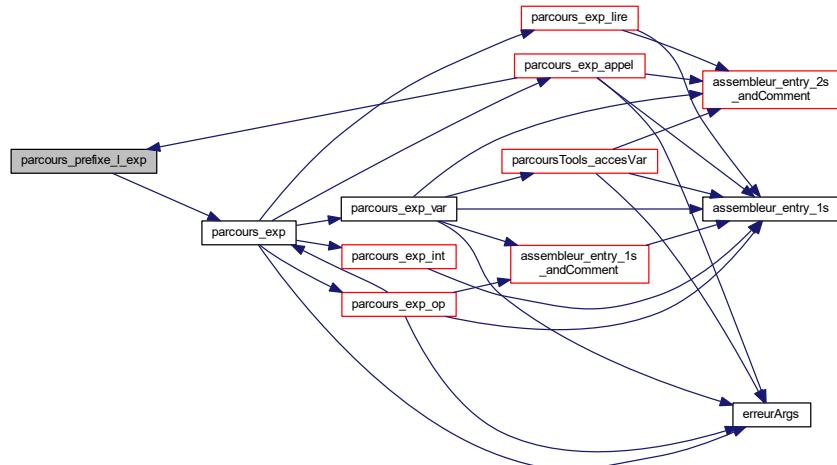
6.23.2.1 `parcours_prefixe_l_exp()`

```
unsigned int parcours_prefixe_l_exp (
    const n_l_exp * n )
```

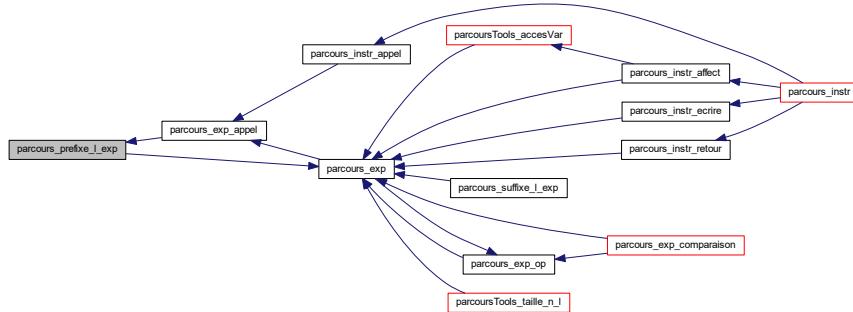
Références `parcours_exp()`.

Référencé par `parcours_exp_appel()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

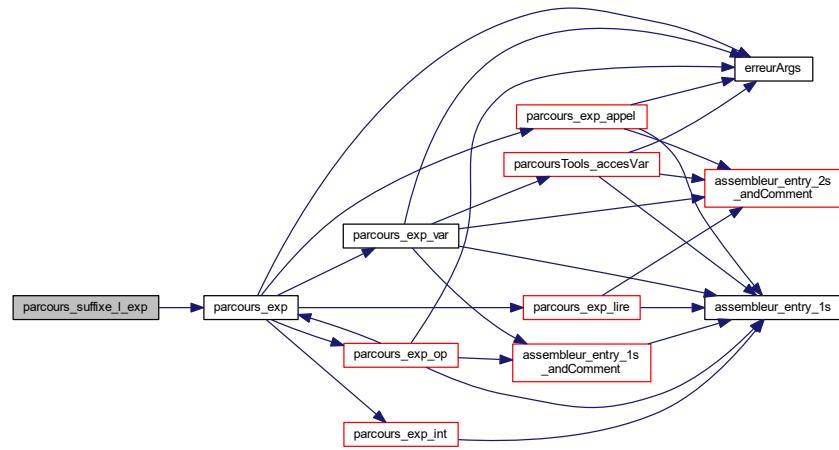


6.23.2.2 `parcours_suffixe_l_exp()`

```
unsigned int parcours_suffixe_l_exp (
    const n_l_exp * n )
```

Références `parcours_exp()`.

Voici le graphe d'appel pour cette fonction :



6.23.2.3 parcours_exp()

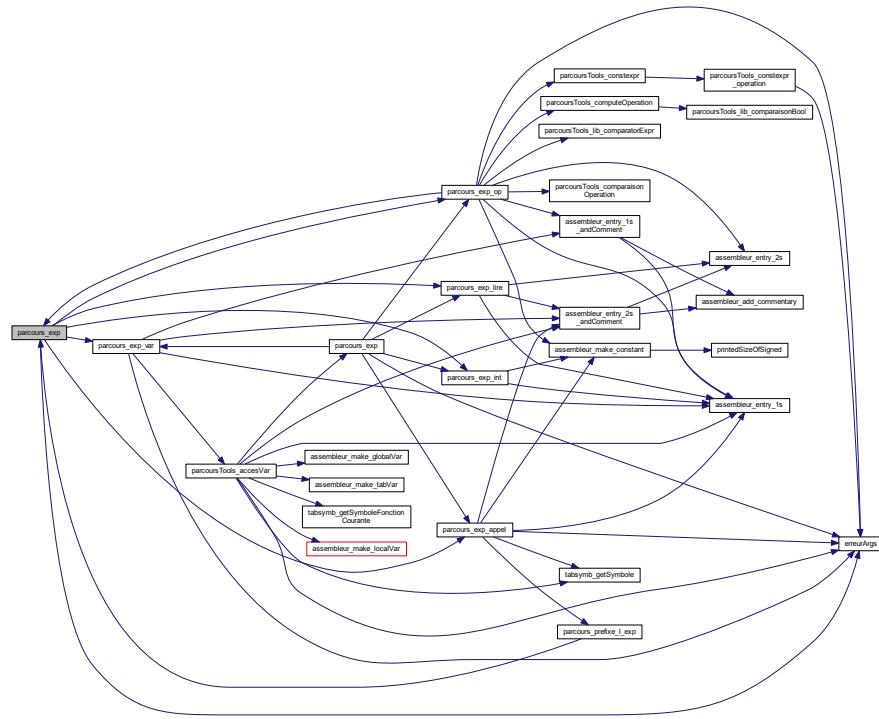
```
void parcours_exp (
    const n_exp * n )
```

parcours_exp : Cette fonction génère les instructions correspondant au calcul de la valeur de l'expression et dépose le résultat du calcul dans la pile. La génération de code pour une opération de type arg1 op arg2 consiste à affecter à des registres les deux opérandes arg1 et arg2 puis à réaliser l'opération à l'aide de l'instruction correspondante et enfin à empiler le résultat de l'opération. Pour les instructions complexes, les valeurs des opérandes (sous expressions) seront prises dans la pile, c'est là qu'aura été déposé le résultat de ces sous expressions. Il faut aussi traiter les lectures de variables lire avec un appel système.

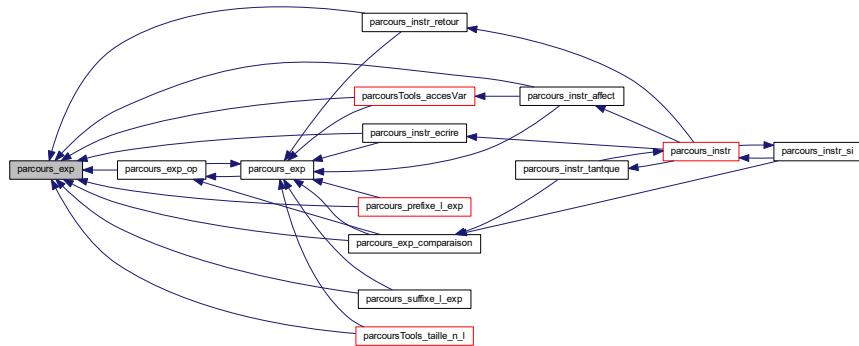
Références erreurArgs(), ErreurSiNulle, parcours_exp_appel(), parcours_exp_int(), parcours_exp_lire(), parcours←_exp_op(), et parcours_exp_var().

Référencé par parcours_exp_comparaison(), parcours_exp_op(), parcours_instr_affect(), parcours_instr←ecrire(), parcours_instr_retour(), parcours_prefixe_l_exp(), parcours_suffixe_l_exp(), parcoursTools_accesVar(), et parcoursTools_taille_n_l().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.23.2.4 parcours_exp_comparaison()

```
int parcours_exp_comparaison (
    const n_exp * n,
    const char ** comparaisonOperationPtr )
```

parcours_exp_comparaison : est executer a partir d'une fonction de comparaison (pas d'opération) permet de traduire n'importe quelle expression en comparaison (comparaison, test vrai...)

Paramètres

<i>comparaisonOperation</i>	: retourne dans un pointeur, l'instruction asm nécessaire, à valider la comparaison.
-----------------------------	--

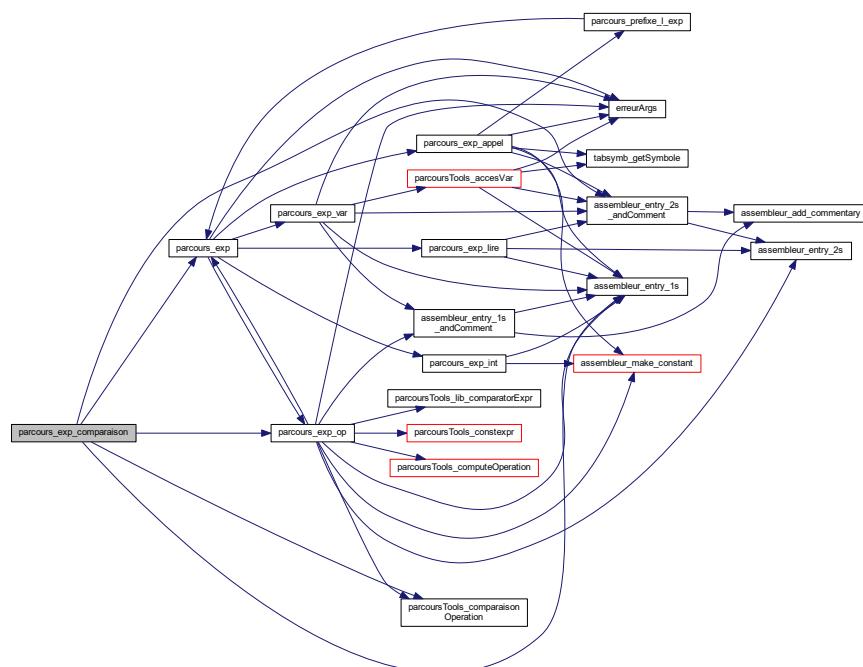
Renvoie

- 1 : comparaisonOperation
- 3 : constexpr vrai
- 4 : constexpr faux

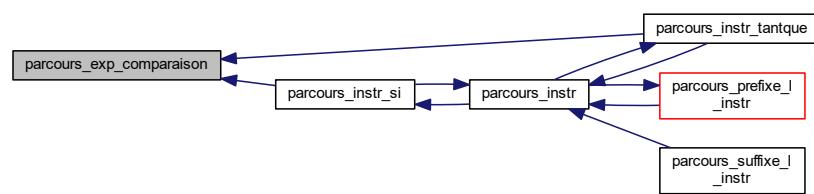
Références assembleur_entry_1s(), assembleur_entry_2s_andComment(), diff, parcours_exp(), parcours_exp←op(), et parcoursTools_comparaisonOperation().

Référencé par parcours_instr_si(), et parcours_instr_tantque().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



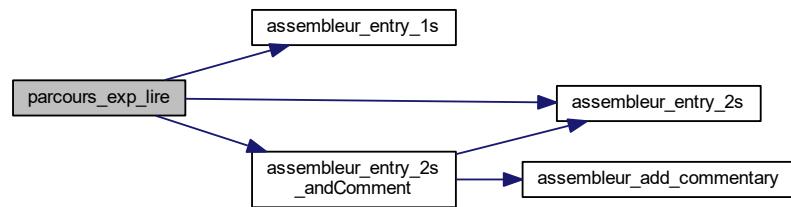
6.23.2.5 parcours_exp_lire()

```
void parcours_exp_lire ( )
```

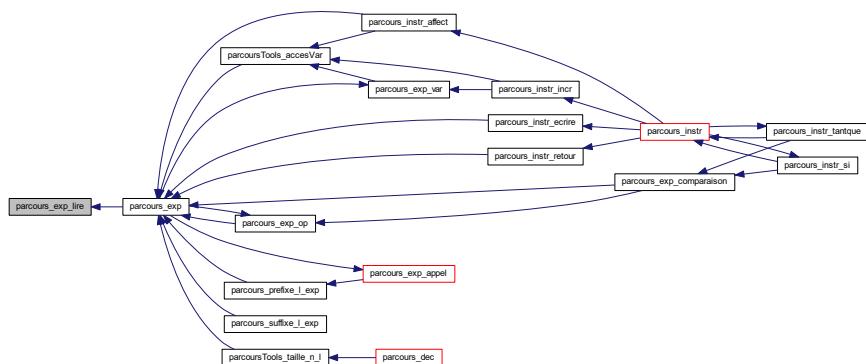
Références assembleur_entry_1s(), assembleur_entry_2s(), et assembleur_entry_2s_andComment().

Référencé par parcours_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.23.2.6 parcours_exp_var()

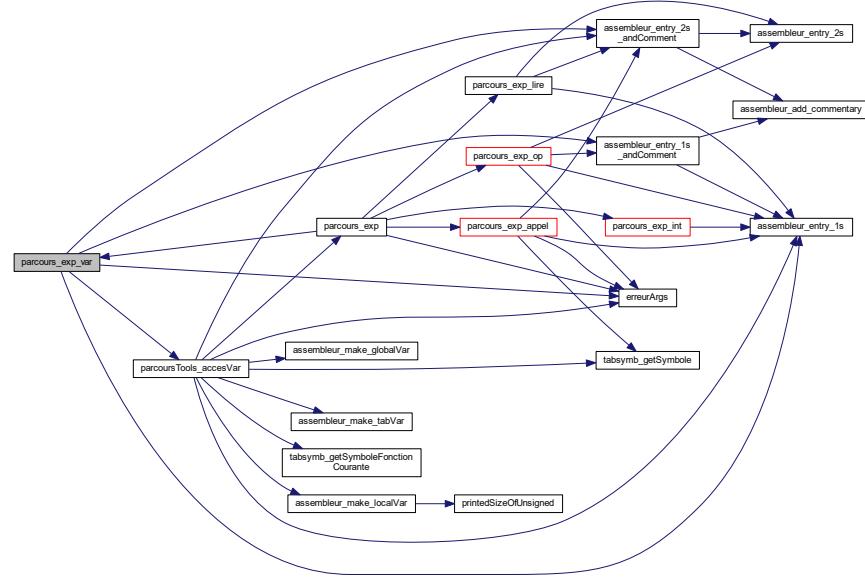
```
void parcours_exp_var (
```

A faire A Confirmer : PUSH direct de la variable, sans passer par MOV

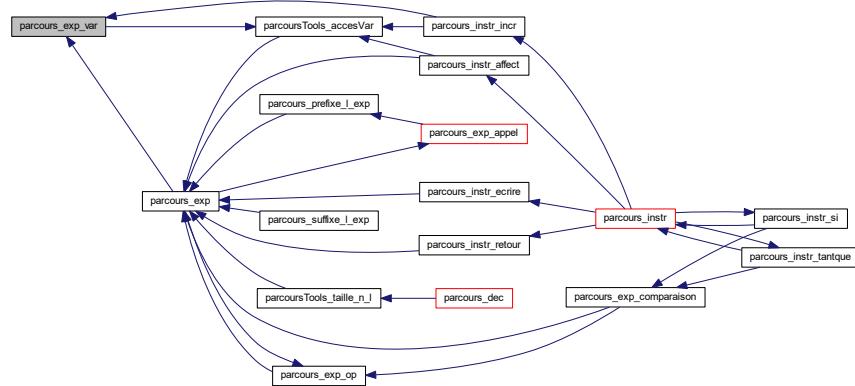
Références assembleur_entry_1s(), assembleur_entry_1s_andComment(), assembleur_entry_2s_andComment(), erreurArgs(), ErreurSiNulle, et parcoursTools_accesVar().

Référencé par parcours `exp()`, et parcours `instr incr()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



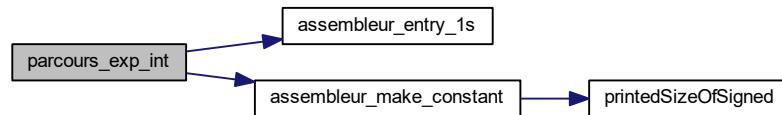
6.23.2.7 parcours_exp_int()

```
void parcours_exp_int (
```

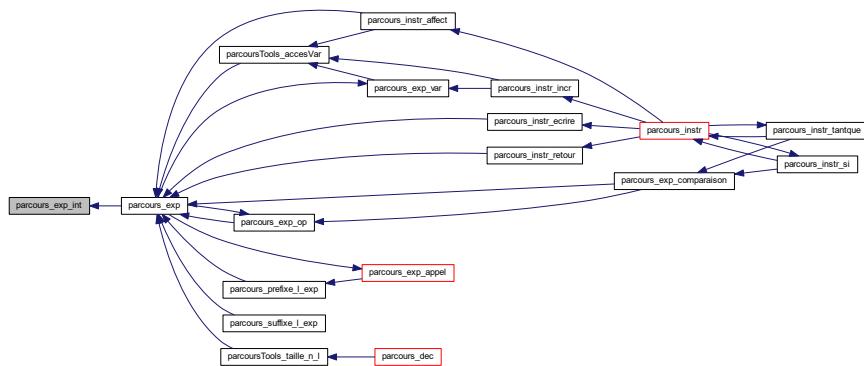
Références assemebleur_entry_1s(), et assemebleur_make_constant().

Référencé par parcours_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.23.2.8 parcours_exp_appel()

```
void parcours_exp_appel (
    const n_appel * appel,
    const bool desallocReturnValue )
```

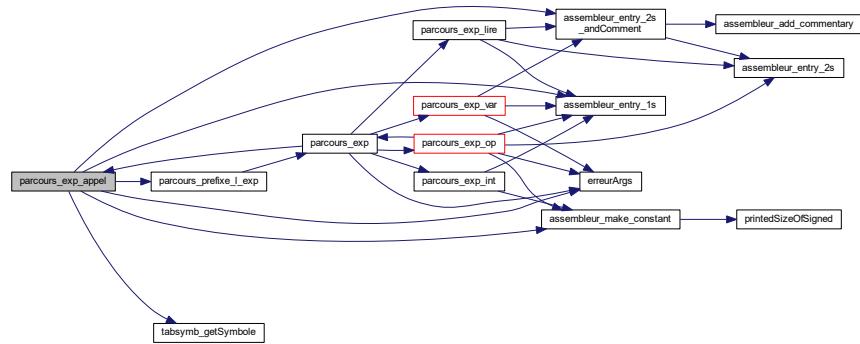
Paramètres

<code>desallocReturnValue</code>	cela désaloue automatiquement la variable de retour avec les arguments (si existant)
----------------------------------	--

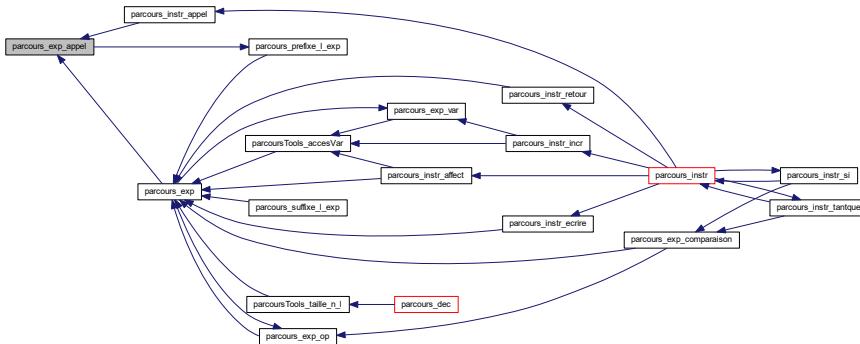
Références assembleur_entry_1s(), assembleur_entry_2s_andComment(), assembleur_make_constant(), desc_{_identif} : :complement, erreurArgs(), ErreurSiNulle, parcours_prefixe_l_exp(), T_FONCTION, tabsymb_get_{Symbole()}, et desc_{_identif} : :type.

Référencé par parcours_exp(), et parcours_instr_appel().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.23.2.9 parcours_exp_op()

```
int parcours_exp_op (
    const n_exp * n,
    const bool isComparaison )
```

parcours_exp_op

Paramètres

<i>isComparaison</i>	doit indiquer si l'on ressort avec un 'cmp' (comparaison) ou 'push' (expression)
----------------------	--

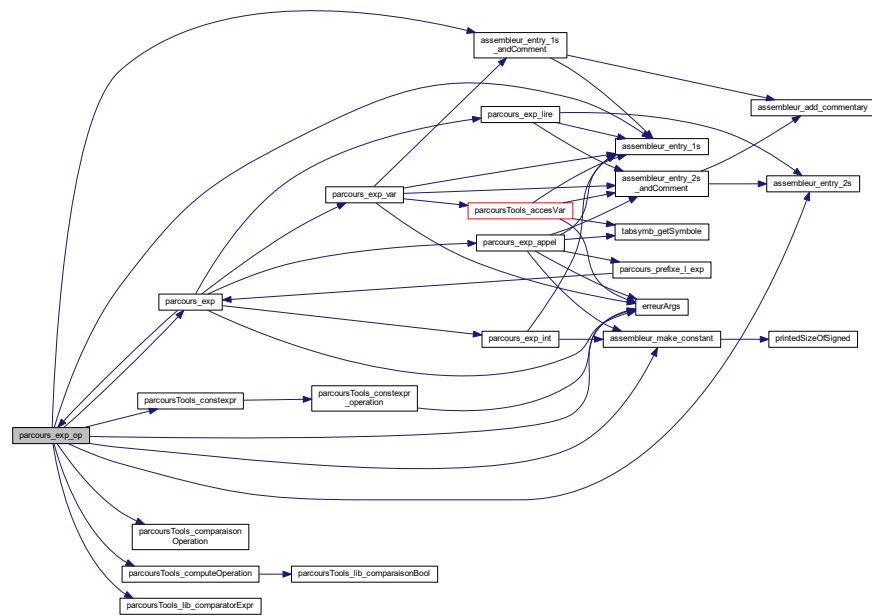
Renvoie

- 0 : computeOperation
- 1 : comparaisonOperation
- 2 : constexpr
- 3 : constexpr vrai
- 4 : constexpr faux

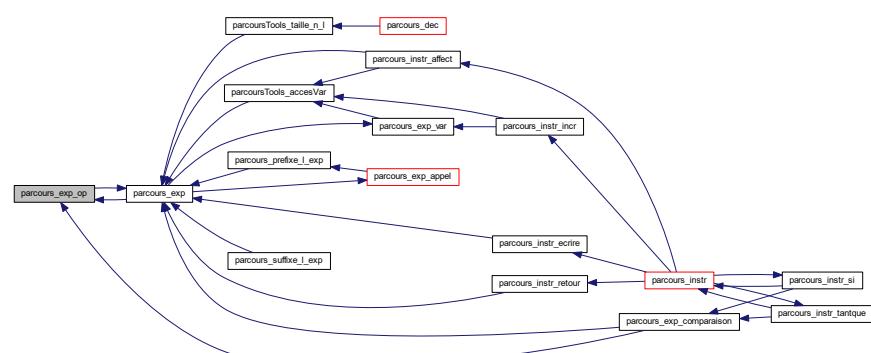
Références assembleur_entry_1s(), assembleur_entry_1s_andComment(), assembleur_entry_2s(), assembleur←_make_constant(), erreurArgs(), ErreurSiMauvaisType, ErreurSiNulle, parcours_exp(), parcoursTools←comparaisonOperation(), parcoursTools_computeOperation(), parcoursTools_constexpr(), et parcoursTools←lib_comparatorExpr().

Référencé par parcours_exp(), et parcours_exp_comparaison().

Voici le graphe d'appel pour cette fonction :

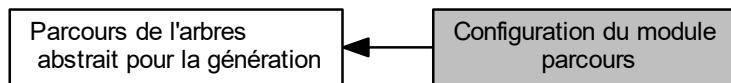


Voici le graphe des appels de cette fonction :



6.24 Configuration du module parcours

Graphe de collaboration de Configuration du module parcours :



Macros

- #define PARCOURS_STARTENTRY_ACCEPT_RETVALUE 1
- #define PARCOURS_VAR_USE_DWORD 1
- #define HAVE_LOGIQUE_COMPARAISSON 0
- #define IO_USE_READLINE_ENTRY 1

6.24.1 Description détaillée

6.24.2 Documentation des macros

6.24.2.1 PARCOURS_STARTENTRY_ACCEPT_RETVALUE

```
#define PARCOURS_STARTENTRY_ACCEPT_RETVALUE 1
```

permet au programme de renvoyer une valeur

6.24.2.2 PARCOURS_VAR_USE_DWORD

```
#define PARCOURS_VAR_USE_DWORD 1
```

voir dans le code, mais cela évite un instruction mov

6.24.2.3 HAVE_LOGIQUE_COMPARAISSON

```
#define HAVE_LOGIQUE_COMPARAISSON 0
```

Inverssement à booléenne

6.24.2.4 IO_USE_READLINE_ENTRY

```
#define IO_USE_READLINE_ENTRY 1
```

L'opération de lecture est réalisé par l'entrée 'readline' incluse dans le fichier 'io.asm'

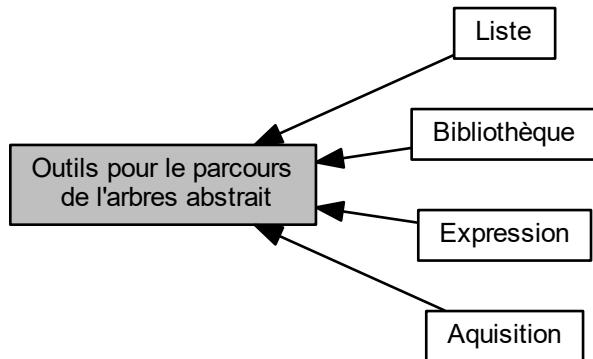
Note

Cela permet de corriger le problème de lecture d'une entrée d'utilisateur, notamment pour l'utilisation de cette valeur obtenu, dans une instruction de division et de modulo ce qui empêche le problème : 'Exception en point flottant (core dumped)'

6.25 Outils pour le parcours de l'arbres abstrait

Outils pour le parcours.

Graphe de collaboration de Outils pour le parcours de l'arbres abstrait :



Modules

- Expression
- Liste
- Aquisition
- Bibliothèque

6.25.1 Description détaillée

Outils pour le parcours.

Auteur

Yannick Robin
Christophe Sonntag (<http://u4a.at>)

6.26 Expression

Graphe de collaboration de Expression :



Fonctions

- int `parcoursTools_constexpr_operation` (const `operation` op, const int op1, const int op2, bool *compute ←
Comparaison)
calculer_exp_const permet de calculer les opérations constantes
- bool `parcoursTools_constexpr` (const n_exp *n, int *compute, bool *computeComparaison)
parcoursTools_constexpr : créer la valeur constante, si l'opération (recursive) ne contient que des entiers
- const char * `parcoursTools_computeOperation` (const `operation` op, const char **reg, bool *oneOperand ←
Destination, bool *isCall)
parcoursTools_computeOperation
- const char * `parcoursTools_comparaisonOperation` (const `operation` op)

6.26.1 Description détaillée

6.26.2 Documentation des fonctions

6.26.2.1 `parcoursTools_constexpr_operation()`

```
int parcoursTools_constexpr_operation (
    const operation op,
    const int op1,
    const int op2,
    bool * computeComparaison )
```

calculer_exp_const permet de calculer les opérations constantes

Paramètres

<code>computeComparaison</code>	: indique si le résultat est une comparaison
---------------------------------	--

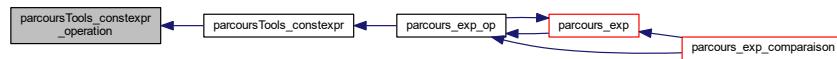
Références diff, divise, erreurArgs(), et, fois, inf, infeg, modulo, moins, negatif, non, ou, plus, sup, et supeg.

Référencé par `parcoursTools_constexpr()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.26.2.2 parcoursTools_constexpr()

```

bool parcoursTools_constexpr (
    const n_exp * n,
    int * compute,
    bool * computeComparaison )
  
```

`parcoursTools_constexpr` : créer la valeur constante, si l'opération (réursive) ne contient que des entiers

Paramètres

<code>compute</code>	: le résultat constant, si l'opération est valide
<code>computeComparaison</code>	: indique si le résultat est une comparaison

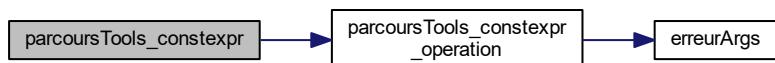
Renvoie

indique si l'opération est valide

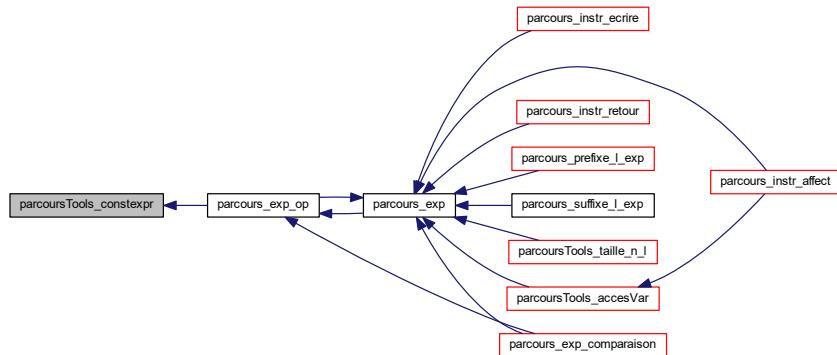
Références `parcoursTools_constexpr_operation()`.

Référencé par `parcours_exp_op()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.26.2.3 parcoursTools_computeOperation()

```
const char* parcoursTools_computeOperation (
    const operation op,
    const char ** reg,
    bool * oneOperandDestination,
    bool * isCall )
```

parcoursTools_computeOperation

Paramètres

<i>reg</i>	: le registre pour recuperer le resultat
<i>oneOperandDestination</i>	: ptr sur boolean, indique si l'opération ne prend pas le registre 'source' et donc seulement le registre 'destination'
<i>isCall</i>	: ptr sur boolean, indique si l'opération doit être appellée

Renvoie

la chaine de caractère corespondant à l'instruction de l'opération 'op'

A faire Voir pour ajouter l'instruction initialisation de edx : mov edx, 0 ; initialise edx à zéro

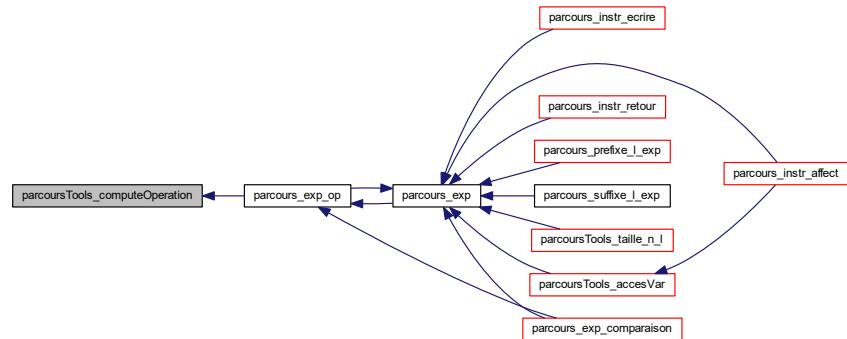
Références divise, et, fois, modulo, moins, non, ou, parcoursTools_lib_comparaisonBool(), et plus.

Référencé par parcours_exp_op().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



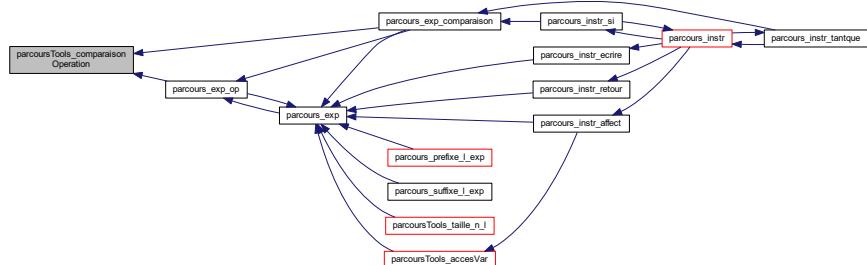
6.26.2.4 parcoursTools_comparaisonOperation()

```
const char* parcoursTools_comparaisonOperation (
    const operation op )
```

Références diff, egal, inf, infeg, sup, et supeg.

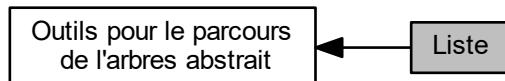
Référencé par parcours_exp_comparaison(), et parcours_exp_op().

Voici le graphe des appelants de cette fonction :



6.27 Liste

Graphe de collaboration de Liste :



Fonctions

- `size_t parcoursTools_taille_n_l (n_l_dec *n)`
`parcoursTools_taille_n_l` : renvoi la longeur d'une liste chainée

6.27.1 Description détaillée

6.27.2 Documentation des fonctions

6.27.2.1 `parcoursTools_taille_n_l()`

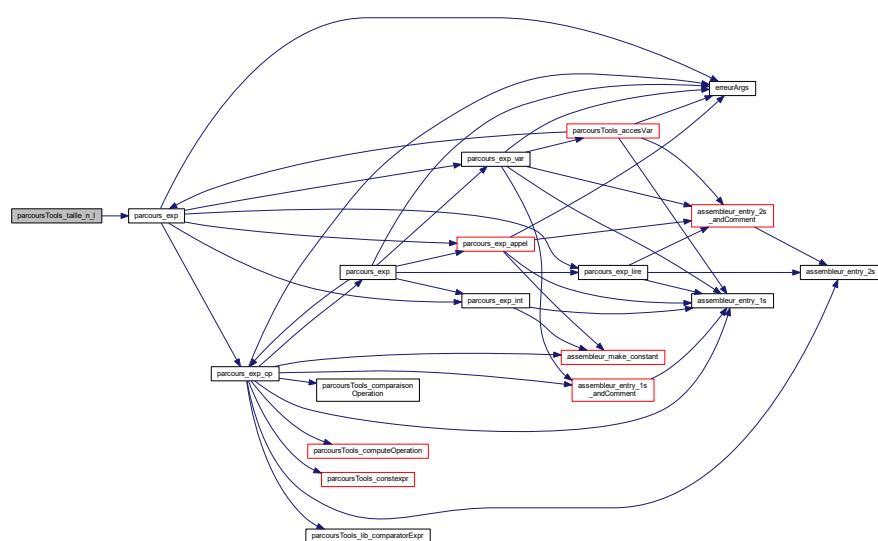
```
size_t parcoursTools_taille_n_l (
    n_l_dec * n )
```

`parcoursTools_taille_n_l` : renvoi la longeur d'une liste chainée

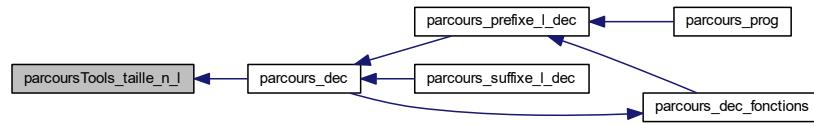
Références ce, et `parcours_exp()`.

Référencé par `parcours_dec()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.28 Aquisition

Graphe de collaboration de Aquisition :



Fonctions

- void [parcours_exp](#) (const n_exp *n)

parcours_exp : Cette fonction génère les instructions correspondant au calcul de la valeur de l'expression et dépose le résultat du calcul dans la pile. La génération de code pour une opération de type arg1 op arg2 consiste à affecter à des registres les deux opérandes arg1 et arg2 puis à réaliser l'opération à l'aide de l'instruction correspondante et enfin à empiler le résultat de l'opération. Pour les instructions complexes, les valeurs des opérandes (sous expressions) seront prises dans la pile, c'est là qu'aura été déposé le résultat de ces sous expressions. Il faut aussi traiter les lectures de variables lire avec un appel système.
- const char * [parcoursTools_accesVar](#) (const n_var *variable, const char **comment)

parcoursTools_accesVar

Variables

- [assembleur_entry](#) * ce

6.28.1 Description détaillée

6.28.2 Documentation des fonctions

6.28.2.1 parcours_exp()

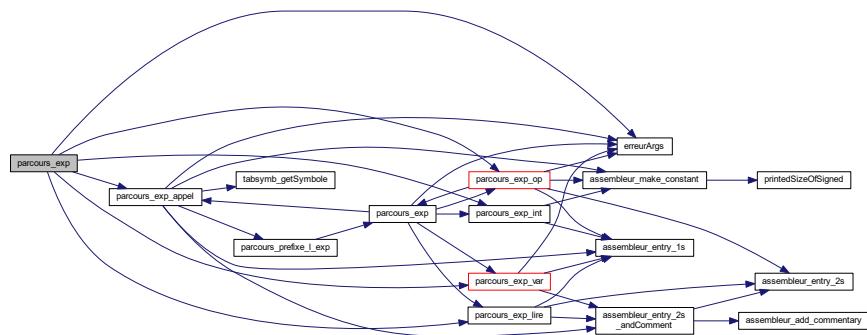
```
void parcours_exp (
    const n_exp * n )
```

parcours_exp : Cette fonction génère les instructions correspondant au calcul de la valeur de l'expression et dépose le résultat du calcul dans la pile. La génération de code pour une opération de type arg1 op arg2 consiste à affecter à des registres les deux opérandes arg1 et arg2 puis à réaliser l'opération à l'aide de l'instruction correspondante et enfin à empiler le résultat de l'opération. Pour les instructions complexes, les valeurs des opérandes (sous expressions) seront prises dans la pile, c'est là qu'aura été déposé le résultat de ces sous expressions. Il faut aussi traiter les lectures de variables lire avec un appel système.

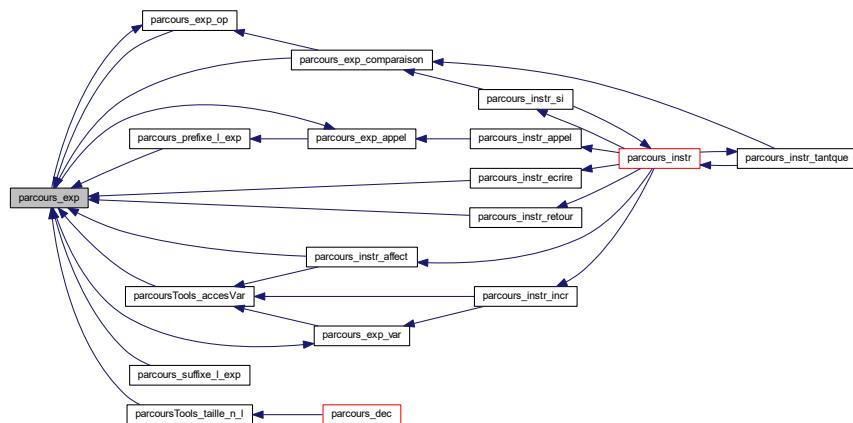
Références erreurArgs(), ErreurSiNulle, parcours_exp_appel(), parcours_exp_int(), parcours_exp_lire(), parcours_exp_op(), et parcours_exp_var().

Référencé par parcours_exp_comparaison(), parcours_exp_op(), parcours_instr_affect(), parcours_instr_ecrire(), parcours_instr_retour(), parcours_prefixe_l_exp(), parcours_suffixe_l_exp(), parcoursTools_accesVar(), et parcoursTools_taille_n_l().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.28.2.2 parcoursTools_accesVar()

```
const char* parcoursTools_accesVar (
    const n_var * variable,
    const char ** comment )
```

`parcoursTools_accesVar`

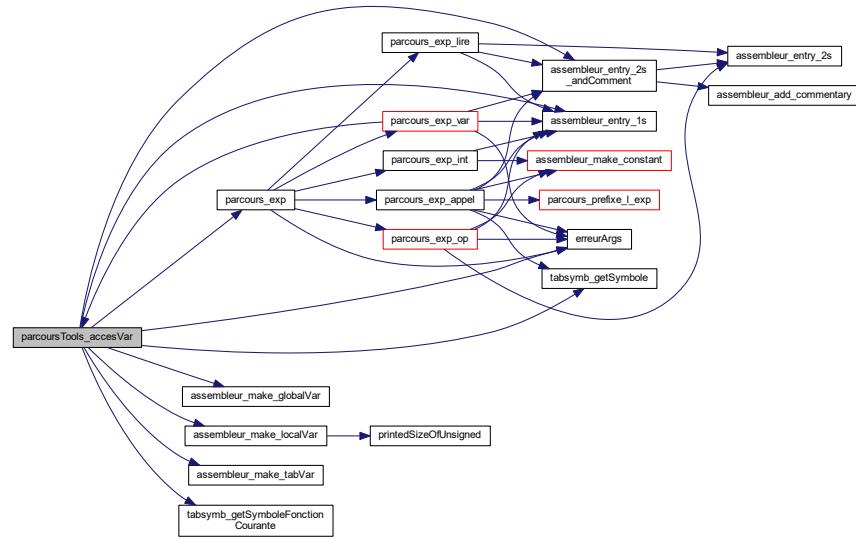
Avertissement

AccesVar peut écrire dans l'entrée courante 'ce'
le registre 'eax' peut être utilisé durant l'opération (cas tableau)

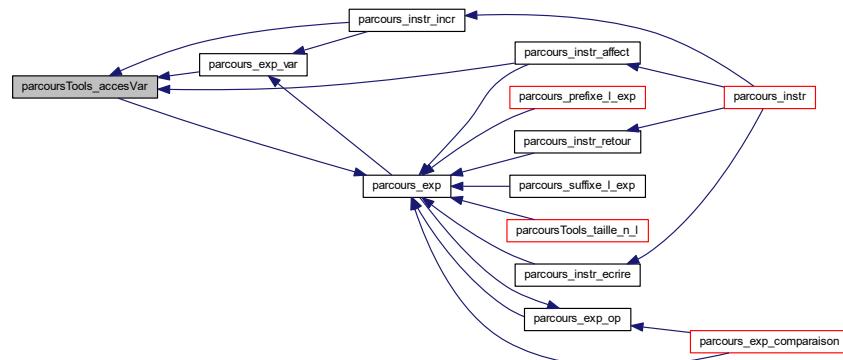
Références desc_identif : :adresse, assembleur_entry_1s(), assembleur_entry_2s_andComment(), assembleur_make_globalVar(), assembleur_make_localVar(), assembleur_make_tabVar(), ce, desc_identif : :complement, erreurArgs(), P_ARGUMENT, P_VARIABLE_GLOBALE, P_VARIABLE_LOCALE, parcours_exp(), desc_identif : :portee, T_ENTIER, T_TABLEAU_ENTIER, tabsymb_getSymbole(), tabsymb_getSymboleFonctionCourante(), et desc_identif : :type.

Référencé par parcours_exp_var(), parcours_instr_affect(), et parcours_instr_incr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.28.3 Documentation des variables

6.28.3.1 ce

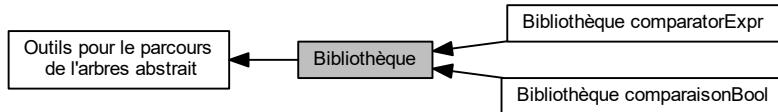
`assembleur_entry*` ce

c'est l'entrée courante

Référencé par parcours_instr_si(), parcours_instr_tantque(), parcoursTools_accesVar(), et parcoursTools_taille_←n_!().

6.29 Bibliothèque

Graphe de collaboration de Bibliothèque :



Modules

- Bibliothèque comparatorExpr
- Bibliothèque comparaisonBool

Fonctions

- void parcoursTools_dump_asm_lib ()

6.29.1 Description détaillée

6.29.2 Documentation des fonctions

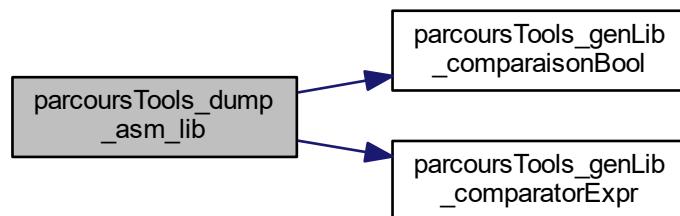
6.29.2.1 parcoursTools_dump_asm_lib()

```
void parcoursTools_dump_asm_lib ( )
```

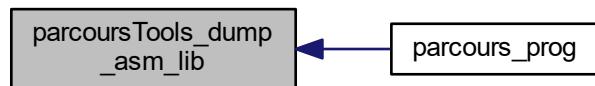
Références parcoursTools_genLib_comparaisonBool(), et parcoursTools_genLib_comparatorExpr().

Référencé par parcours_prog().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.30 Bibliothèque comparatorExpr

Graphe de collaboration de Bibliothèque comparatorExpr :



Macros

- #define LIB_COMPARATOREXPR_PREFIX "_cmpBin_"
- #define LIB_COMPARATOREXPR_SIZE 6

Fonctions

- const char * parcoursTools_lib_comparatorExpr (operation op)
- void parcoursTools_genLib_comparatorExpr ()

6.30.1 Description détaillée

6.30.2 Documentation des macros

6.30.2.1 LIB_COMPARATOREXPR_PREFIX

```
#define LIB_COMPARATOREXPR_PREFIX "_cmpBin_"
```

6.30.2.2 LIB_COMPARATOREXPR_SIZE

```
#define LIB_COMPARATOREXPR_SIZE 6
```

Référencé par parcoursTools_genLib_comparatorExpr(), et parcoursTools_lib_comparatorExpr().

6.30.3 Documentation des fonctions

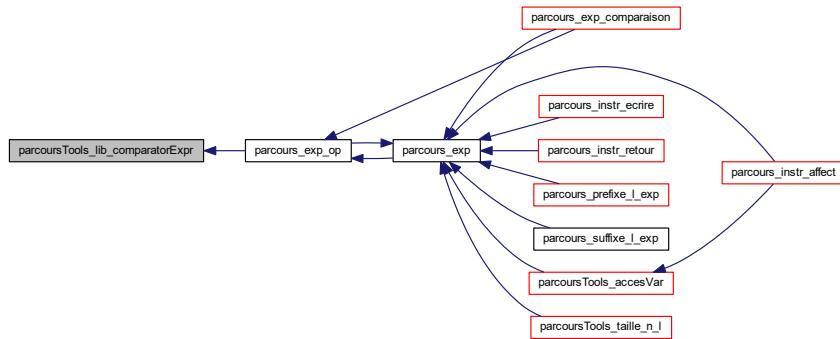
6.30.3.1 parcoursTools_lib_comparatorExpr()

```
const char* parcoursTools_lib_comparatorExpr (
    operation op )
```

Références LIB_COMPARATOREXPR_SIZE.

Référencé par parcours_exp_op().

Voici le graphe des appels de cette fonction :



6.30.3.2 parcoursTools_genLib_comparatorExpr()

```
void parcoursTools_genLib_comparatorExpr ( )
```

Références LIB_COMPARATOREXPR_SIZE.

Référencé par parcoursTools_dump_asm_lib().

Voici le graphe des appels de cette fonction :



6.31 Bibliothèque comparaisonBool

Graphe de collaboration de Bibliothèque comparaisonBool :



Macros

- `#define LIB_COMPARISONBOOL_PREFIX "_cmpBool_"`
- `#define LIB_COMPARISONBOOL_SIZE 3`

Fonctions

- `const char * parcoursTools_lib_comparaisonBool_getName(operation op)`
- `const char * parcoursTools_lib_comparaisonBool(operation op)`
- `void parcoursTools_genLib_comparaisonBool()`

6.31.1 Description détaillée

6.31.2 Documentation des macros

6.31.2.1 LIB_COMPARISONBOOL_PREFIX

```
#define LIB_COMPARISONBOOL_PREFIX "_cmpBool_"
```

Référencé par `parcoursTools_lib_comparaisonBool_getName()`.

6.31.2.2 LIB_COMPARISONBOOL_SIZE

```
#define LIB_COMPARISONBOOL_SIZE 3
```

Référencé par `parcoursTools_genLib_comparaisonBool()`, et `parcoursTools_lib_comparaisonBool()`.

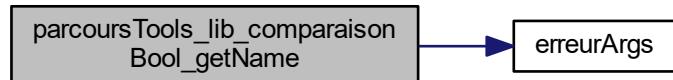
6.31.3 Documentation des fonctions

6.31.3.1 parcoursTools_lib_comparaisonBool_getName()

```
const char* parcoursTools_lib_comparaisonBool_getName (
    operation op )
```

Références erreurArgs(), et, LIB_COMPARISONBOOL_PREFIX, non, et ou.

Voici le graphe d'appel pour cette fonction :



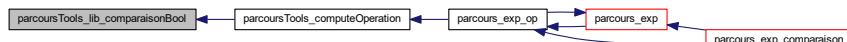
6.31.3.2 parcoursTools_lib_comparaisonBool()

```
const char* parcoursTools_lib_comparaisonBool (
    operation op )
```

Références LIB_COMPARISONBOOL_SIZE.

Référencé par parcoursTools_computeOperation().

Voici le graphe des appelants de cette fonction :



6.31.3.3 parcoursTools_genLib_comparaisonBool()

```
void parcoursTools_genLib_comparaisonBool ( )
```

Références LIB_COMPARISONBOOL_SIZE.

Référencé par parcoursTools_dump_asm_lib().

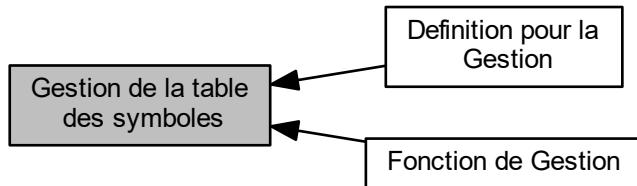
Voici le graphe des appelants de cette fonction :



6.32 Gestion de la table des symboles

Partie gestion de la Table des symboles.

Graphe de collaboration de Gestion de la table des symboles :



Modules

- Fonction de Gestion
- Definition pour la Gestion

Structures de données

- struct `tabsymboles`
`tabsymboles` : Table des symboles (globale ET locale)
- struct `desc_identif`

Fonctions

- void `tabsymb_setAffiche` (bool b)

Variables

- int `adresseGlobalCourant`
- int `adresseLocaleCourante`
- int `adresseArgumentCourant`

6.32.1 Description détaillée

Partie gestion de la Table des symboles.

Auteur

Alexis Nasr (<http://pageperso.lif.univ-mrs.fr/~alexis.nasr/>)
released by Yannick Robin
released by Christophe Sonntag (<http://u4a.at>)

6.32.2 Documentation des fonctions

6.32.2.1 tabsymb_setAffiche()

```
void tabsymb_setAffiche (
    bool b )
```

6.32.3 Documentation des variables

6.32.3.1 adresseGlobalCourant

```
int adresseGlobalCourant
```

Référencé par parcours_dec(), et parcours_prog().

6.32.3.2 adresseLocaleCourante

```
int adresseLocaleCourante
```

Référencé par parcours_dec(), et tabsymb_entreeFonction().

6.32.3.3 adresseArgumentCourant

```
int adresseArgumentCourant
```

Référencé par parcours_dec(), et tabsymb_entreeFonction().

6.33 Fonction de Gestion

Graphe de collaboration de Fonction de Gestion :



Fonctions

- void **tabsymb_entreeFonction** (void)

tabsymb_entreeFonction : Fonction qui bascule table globale -> table locale. À appeler lors qu'on parcourt une déclaration de fonction, juste avant la liste d'arguments.
- void **tabsymb_sortieFonction** (void)

tabsymb_sortieFonction : Fonction qui bascule table locale -> table globale. À appeler lors qu'on a fini de parcourir une déclaration de fonction, après le bloc d'instructions qui constitue le corps de la fonction
- int **tabsymb_ajouteIdentificateur** (char *identif, int portee, int type, size_t adresse, size_t complement)

tabsymb_ajouteIdentificateur : Ajoute un nouvel identificateur à la table des symboles courante.
- int **tabsymb_rechercheExecutable** (const char *identif)

tabsymb_rechercheExecutable : Recherche si un identificateur est présent dans la table LOCALE ou GLOBALE. Cette fonction doit être utilisée pour s'assurer que tout identificateur utilisé a été déclaré auparavant. Elle doit être utilisée lors de l'UTILISATION d'un identificateur, soit dans un appel à fonction, une partie gauche d'affectation ou une variable dans une expression. Si deux identificateurs ont le même nom dans des portées différentes (un global et un local), la fonction renvoie le plus proche, c-à-d le local.
- int **tabsymb_rechercheDeclarative** (const char *identif)

tabsymb_rechercheDeclarative : Recherche si un identificateur est présent dans la table LOCALE. Cette fonction doit être utilisée pour s'assurer qu'il n'y a pas deux identificateurs avec le même lors d'une DÉCLARATION d'un identificateur.
- const **desc_identif * tabsymb_getSymbole** (const char *nom)

tabsymb_getSymbole : Trouve le premier executable, voir tabsymb_rechercheExecutable pour plus d'information.
- const **desc_identif * tabsymb_getSymboleFonctionCourante** ()

tabsymb_getSymboleFonctionCourante : Trouve la fonction courante
- void **tabsymb_dump** (void)

tabsymb_dump : Fonction auxiliaire qui permet d'afficher le contenu actuel de la table des symboles.

6.33.1 Description détaillée

6.33.2 Documentation des fonctions

6.33.2.1 tabsymb_entreeFonction()

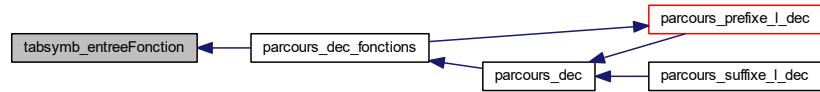
```
void tabsymb_entreeFonction (
    void )
```

tabsymb_entreeFonction : Fonction qui bascule table globale -> table locale. À appeler lors qu'on parcourt une déclaration de fonction, juste avant la liste d'arguments.

Références adresseArgumentCourant, adresseLocaleCourante, tabsymboles_ : :base, et tabsymboles_ : :sommet.

Référencé par parcours_dec_fonctions().

Voici le graphe des appelants de cette fonction :



6.33.2.2 tabsymb_sortieFonction()

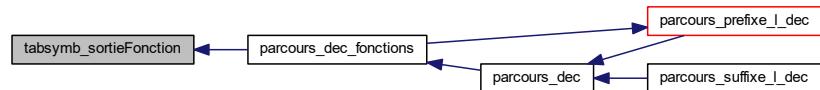
```
void tabsymb_sortieFonction (
    void )
```

tabsymb_sortieFonction : Fonction qui bascule table locale -> table globale. À appeler lors qu'on a fini de parcourir une déclaration de fonction, après le bloc d'instructions qui constitue le corps de la fonction

Références tabsymboles_ : :base, et tabsymboles_ : :sommet.

Référencé par parcours_dec_fonctions().

Voici le graphe des appelants de cette fonction :



6.33.2.3 tabsymb_ajouteidificateur()

```
int tabsymb_ajouteIdentificateur (
    char * identif,
    int portee,
    int type,
    size_t adresse,
    size_t complement )
```

tabsymb_ajouteidificateur : Ajoute un nouvel identificateur à la table des symboles courante.

Paramètres

<i>identif</i>	Nom du nouvel identificateur (variable ou fonction)
<i>portee</i>	Constante parmi P_VARIABLE_GLOBALE, P_ARGUMENT ou P_VARIABLE_LOCALE
<i>type</i>	Constante parmi T_ENTIER, T_TABLEAU_ENTIER et T_FONCTION
<i>adresse</i>	Nombre d'octets de décalage par rapport à la base de la zone mémoire des variables (\$fp pour locales/arguments, .data pour globales)
<i>complement</i>	Nombre de paramètres d'une fonction OU nombre de cases d'un tableau. Indéfini (0) quand type=T_ENTIER.

Renvoie

Numéro de ligne de l'entrée qui vient d'être rajoutée

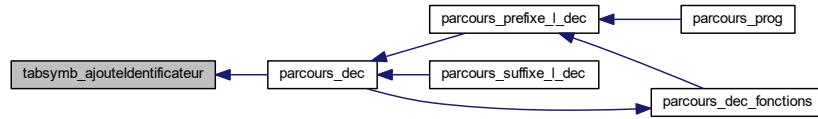
Références desc_identif : :adresse, desc_identif : :complement, erreur(), desc_identif : :identif, MAX_IDENTIF, desc_identif : :portee, tabsymboles_ : :sommet, tabsymboles_ : :tab, et desc_identif : :type.

Référencé par parcours_dec().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

**6.33.2.4 tabsymb_rechercheExecutable()**

```
int tabsymb_rechercheExecutable (
    const char * identif )
```

tabsymb_rechercheExecutable : Recherche si un identificateur est présent dans la table LOCALE ou GLOBALE. Cette fonction doit être utilisée pour s'assurer que tout identificateur utilisé a été déclaré auparavant. Elle doit être utilisée lors de l'UTILISATION d'un identificateur, soit dans un appel à fonction, une partie gauche d'affectation ou une variable dans une expression. Si deux identificateurs ont le même nom dans des portées différentes (un global et un local), la fonction renvoie le plus proche, c-à-d le local.

Paramètres

<i>identif</i>	Le nom de variable ou fonction que l'on cherche
----------------	---

Renvoie

Si oui, renvoie le numéro de ligne dans tabsymboles, si non -1

Références desc_identif : :identif, tabsymboles_ : :sommet, et tabsymboles_ : :tab.

6.33.2.5 tabsymb_rechercheDeclarative()

```
int tabsymb_rechercheDeclarative (
    const char * identif )
```

tabsymb_rechercheDeclarative : Recherche si un identificateur est présent dans la table LOCALE Cette fonction doit être utilisée pour s'assurer qu'il n'y a pas deux identificateurs avec le même lors d'une DÉCLARATION d'un identificateur.

Paramètres

<i>identif</i>	Le nom de variable ou fonction que l'on cherche
----------------	---

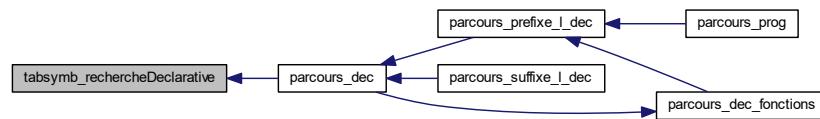
Renvoie

Si oui, renvoie le numéro de ligne dans tabsymboles, si non -1

Références tabsymboles_::base, desc_identif::identif, tabsymboles_::sommet, et tabsymboles_::tab.

Référencé par parcours_dec().

Voici le graphe des appelants de cette fonction :



6.33.2.6 tabsymb_getSymbole()

```
const desc_identif* tabsymb_getSymbole (
    const char * nom )
```

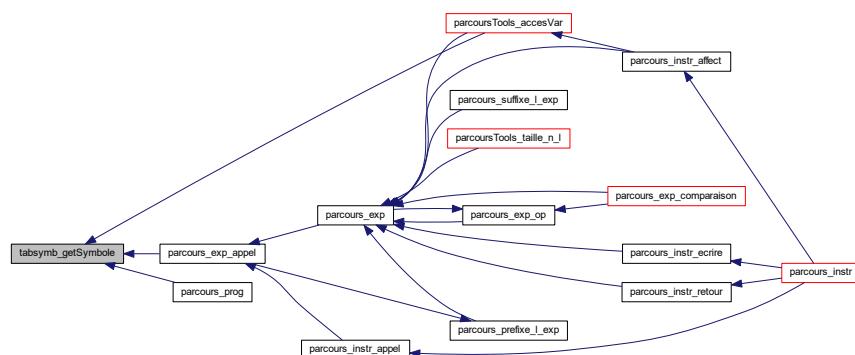
tabsymb_getSymbole : Trouve le premier exectuable, voir tabsymb_rechercheExecutable pour plus d'information.

Renvoie

si le symbole existe alors il est renvoyé, sinon NULL

Référencé par parcours_exp_appel(), parcours_prog(), et parcoursTools_accesVar().

Voici le graphe des appelants de cette fonction :



6.33.2.7 tabsymb_getSymboleFonctionCourante()

```
const desc_identif* tabsymb_getSymboleFonctionCourante ( )
```

tabsymb_getSymboleFonctionCourante : Trouve la fonction courante

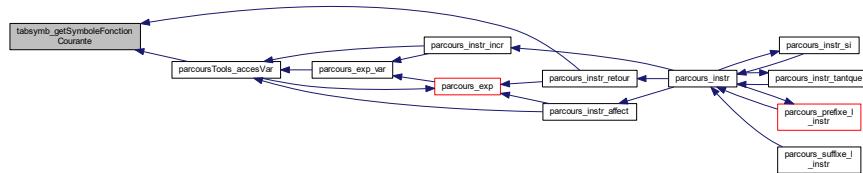
Renvoie

si la fonction est trouvé alors elle est renvoyé, sinon NULL

Références tabsymboles_ : :base, T_FONCTION, tabsymboles_ : :tab, et desc_identif : :type.

Référencé par parcours_instr_retour(), et parcoursTools_accesVar().

Voici le graphe des appelants de cette fonction :



6.33.2.8 tabsymb_dump()

```
void tabsymb_dump (
    void )
```

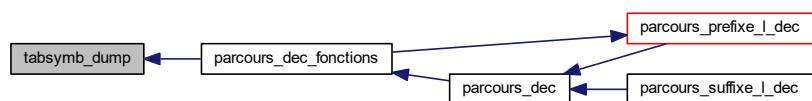
tabsymb_dump : Fonction auxiliaire qui permet d'afficher le contenu actuel de la table des symboles.

Note

Cette fonction est conditionné sur une variable booléenne qui contrôle si on veut ou pas afficher la table des symboles en fonction des options passées au compilateur.

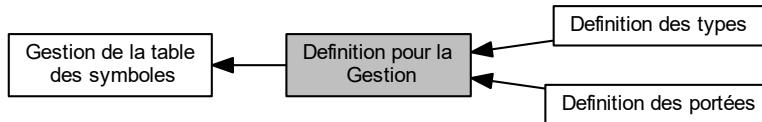
Référencé par parcours_dec_fonctions().

Voici le graphe des appelants de cette fonction :



6.34 Definition pour la Gestion

Graphe de collaboration de Definition pour la Gestion :



Modules

- [Definition des portées](#)
- [Definition des types](#)

Macros

- #define MAX_IDENTIF 1000

6.34.1 Description détaillée

6.34.2 Documentation des macros

6.34.2.1 MAX_IDENTIF

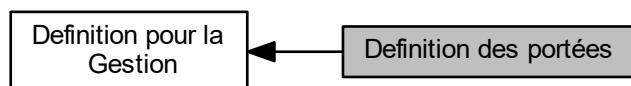
```
#define MAX_IDENTIF 1000
```

Nombre maximum d'identificateurs (globaux + locaux dans 1 fonction)

Référencé par tabsymb_ajoutelidentificateur().

6.35 Definition des portées

Graphe de collaboration de Definition des portées :



Macros

- #define P_VARIABLE_GLOBALE 1
- #define P_VARIABLE_LOCALE 2
- #define P_ARGUMENT 3

6.35.1 Description détaillée

6.35.2 Documentation des macros

6.35.2.1 P_VARIABLE_GLOBALE

```
#define P_VARIABLE_GLOBALE 1
```

Référencé par parcours_dec(), parcours_prog(), et parcoursTools_accesVar().

6.35.2.2 P_VARIABLE_LOCALE

```
#define P_VARIABLE_LOCALE 2
```

Référencé par parcours_dec(), parcours_dec_fonctions(), et parcoursTools_accesVar().

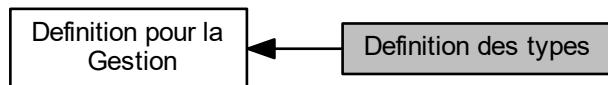
6.35.2.3 P_ARGUMENT

```
#define P_ARGUMENT 3
```

Référencé par parcours_dec(), parcours_dec_fonctions(), et parcoursTools_accesVar().

6.36 Definition des types

Graphe de collaboration de Definition des types :



Macros

```
— #define T_ENTIER 1
— #define T_TABLEAU_ENTIER 2
— #define T_FONCTION 3
```

6.36.1 Description détaillée

6.36.2 Documentation des macros

6.36.2.1 T_ENTIER

```
#define T_ENTIER 1
```

Référencé par parcours_dec(), et parcoursTools_accesVar().

6.36.2.2 T_TABLEAU_ENTIER

```
#define T_TABLEAU_ENTIER 2
```

Référencé par parcours_dec(), et parcoursTools_accesVar().

6.36.2.3 T_FONCTION

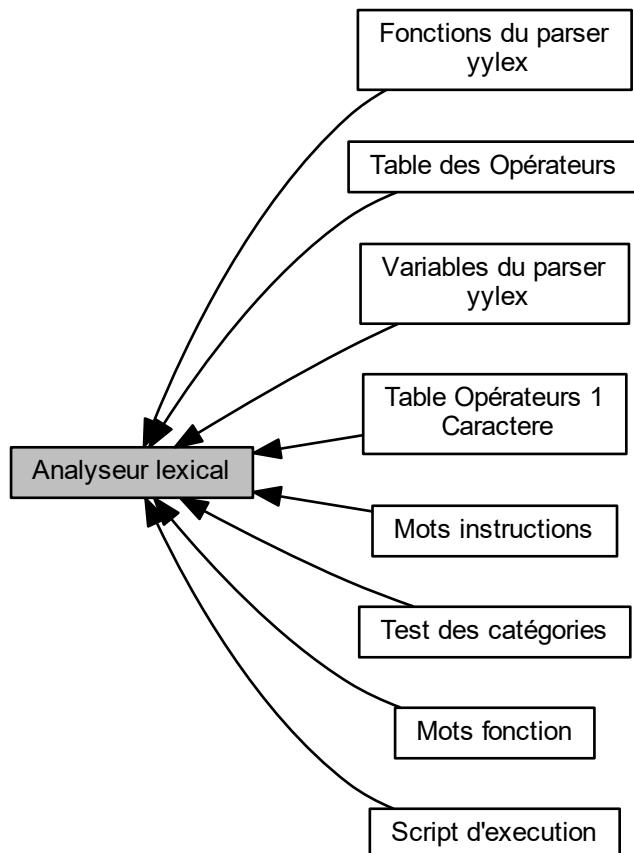
```
#define T_FONCTION 3
```

Référencé par parcours_dec(), parcours_exp_appel(), parcours_prog(), et tabsymb_getSymboleFonctionCourante().

6.37 Analyseur lexical

Partie création de la Table des symboles.

Graphe de collaboration de Analyseur lexical :



Modules

- Test des catégories
- Table Opérateurs 1 Caractere
- Table des Opérateurs
- Mots instructions
- Mots fonction
- Variables du parser yylex
- Fonctions du parser yylex
- Script d'exécution

6.37.1 Description détaillée

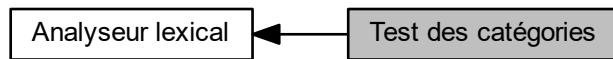
Partie création de la Table des symboles.

Auteur

Alexis Nasr (<http://pageperso.lif.univ-mrs.fr/~alexis.nasr/>)
released by Yannick Robin
released by Christophe Sonntag (<http://u4a.at>)

6.38 Test des catégories

Graphe de collaboration de Test des catégories :



Macros

```

— #define YYTEXT_MAX 100
— #define is_num(c) ((‘0’ <= (c)) && ((c) <= ‘9’))
— #define is_maj(c) ((‘A’ <= (c)) && ((c) <= ‘Z’))
— #define is_min(c) ((‘a’ <= (c)) && ((c) <= ‘z’))
— #define is_alpha(c) (is_maj(c) || is_min(c) || (c) == ‘ ’ || (c) == ‘$’)
— #define is_alpha_simple(c) (is_maj(c) || is_min(c) || (c) == ‘_’)
— #define is_alphanum(c) (is_num((c)) || is_alpha((c)))
  
```

6.38.1 Description détaillée

6.38.2 Documentation des macros

6.38.2.1 YYTEXT_MAX

```
#define YYTEXT_MAX 100
```

6.38.2.2 is_num

```
#define is_num(
    c ) ((‘0’ <= (c)) && ((c) <= ‘9’))
```

6.38.2.3 is_maj

```
#define is_maj(
    c ) ((‘A’ <= (c)) && ((c) <= ‘Z’))
```

6.38.2.4 is_min

```
#define is_min(
    c ) ((‘a’ <= (c)) && ((c) <= ‘z’))
```

6.38.2.5 `is_alpha`

```
#define is_alpha(  
    c ) (is_maj(c) || is_min(c) || (c) == '_' || (c) == '$')
```

6.38.2.6 `is_alpha_simple`

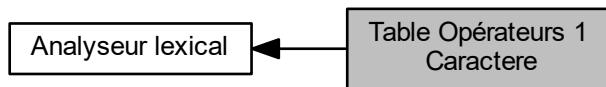
```
#define is_alpha_simple(  
    c ) (is_maj(c) || is_min(c) || (c) == '_')
```

6.38.2.7 `is_alphanum`

```
#define is_alphanum(  
    c ) (is_num((c)) || is_alpha((c)))
```

6.39 Table Opérateurs 1 Caractere

Graphe de collaboration de Table Opérateurs 1 Caractere :



Macros

— #define MotsClefs_Operators_1Carac_SIZE 15

6.39.1 Description détaillée

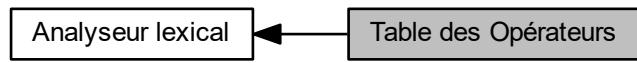
6.39.2 Documentation des macros

6.39.2.1 MotsClefs_Operators_1Carac_SIZE

```
#define MotsClefs_Operators_1Carac_SIZE 15
```

6.40 Table des Opérateurs

Graphe de collaboration de Table des Opérateurs :



Macros

— `#define MotsClefs_Operators_SIZE 7`

6.40.1 Description détaillée

6.40.2 Documentation des macros

6.40.2.1 MotsClefs_Operators_SIZE

```
#define MotsClefs_Operators_SIZE 7
```

6.41 Mots instructions

Graphe de collaboration de Mots instructions :



Macros

```
— #define MotsClefs_Instruction_SIZE 8
```

6.41.1 Description détaillée

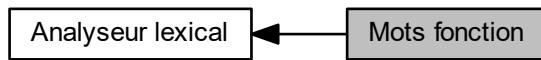
6.41.2 Documentation des macros

6.41.2.1 MotsClefs_Instruction_SIZE

```
#define MotsClefs_Instruction_SIZE 8
```

6.42 Mots fonction

Graphe de collaboration de Mots fonction :



Macros

```
— #define MotsClefs_Fonction_SIZE 2
```

6.42.1 Description détaillée

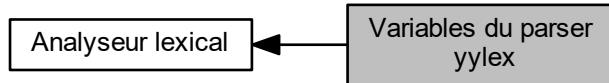
6.42.2 Documentation des macros

6.42.2.1 MotsClefs_Fonction_SIZE

```
#define MotsClefs_Fonction_SIZE 2
```

6.43 Variables du parser yylex

Graphe de collaboration de Variables du parser yylex :



Variables

- char `yytext` [100]
- int `nb_ligne` = 1

6.43.1 Description détaillée

6.43.2 Documentation des variables

6.43.2.1 yytext

```
char yytext[100]
```

Référencé par `delireCar()`, `erreurLexical()`, et `initBuff()`.

6.43.2.2 nb_ligne

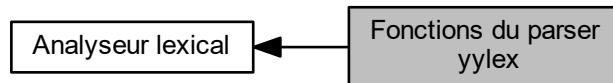
```
int nb_ligne = 1
```

Compter les lignes pour afficher les messages d'erreur avec numéro ligne

Référencé par `erreur_1s()`, `erreurLigne()`, `mangeEspaces()`, `warning_1s()`, et `warningLigne()`.

6.44 Fonctions du parser yylex

Graphe de collaboration de Fonctions du parser yylex :



Fonctions

- void **erreurLexical** (const char *typeErreur)
- int **mangeEspaces** ()
 - mangeEspaces : Fonction qui ignore les espaces et commentaires.*
- char **lireCar** (void)
 - lireCar : Lit un caractère et le stocke dans le buffer yytext*
- void **delireCar** ()
 - delireCar : Remet le dernier caractère lu au buffer clavier et enlève du buffer yytext*
- void **initBuff** ()
 - initBuff : Initialise le buffer yytext*
- bool **estUneVariable** ()
 - estUneVariable : test si la valeur courante correspond à un nom de variable*
- bool **estUneFonction** ()
 - estUneFonction : test si la valeur courante correspond à un nom de fonction*
- bool **estUnNombre** ()
 - estUnNombre : test si la valeur courante est nombre*
- int **yylex** (void)
 - yylex : Fonction principale de l'analyseur lexical, lit les caractères de yyin et renvoie les tokens sous forme d'entier. Le code de chaque unité est défini dans [symboles.h](#) sinon (mot clé, identifiant, etc.). Pour les tokens de type ID_FCT, ID_VAR et NOMBRE la valeur du token est dans yytext, visible dans l'analyseur syntaxique.*
- void **nom_token** (int token, char *nom, char *valeur)
 - nom_token : Fonction auxiliaire appelée par l'analyseur syntaxique tout simplement pour afficher des messages d'erreur et l'arbre XML*

6.44.1 Description détaillée

6.44.2 Documentation des fonctions

6.44.2.1 erreurLexical()

```
void erreurLexical (
    const char * typeErreur )
```

Références yytext.

6.44.2.2 mangeEspaces()

```
int mangeEspaces ( )
```

mangeEspaces : Fonction qui ignore les espaces et commentaires.

Renvoie

Renvoie -1 si arrivé à la fin du fichier, 0 si tout va bien

Références nb_ligne, et yyin.

Référencé par yylex().

Voici le graphe des appelants de cette fonction :



6.44.2.3 lireCar()

```
char lireCar (
    void )
```

lireCar : Lit un caractère et le stocke dans le buffer yytext

Référencé par yylex().

Voici le graphe des appelants de cette fonction :



6.44.2.4 delireCar()

```
void delireCar ( )
```

delireCar : Remet le dernier caractère lu au buffer clavier et enlève du buffer yytext

Références yytext.

6.44.2.5 initBuff()

```
void initBuff ( )
```

Références yytext.

Référencé par yylex().

Voici le graphe des appels de cette fonction :



6.44.2.6 estUneVariable()

```
bool estUneVariable ( )
```

estUneVariable : test si la valeur courante correspond à un nom de variable

Note

un test est effectuer dans yylex avant

6.44.2.7 estUneFonction()

```
bool estUneFonction ( )
```

estUneFonction : test si la valeur courante correspond à un nom de fonction

Note

un test est effectuer dans yylex avant

6.44.2.8 estUnNombre()

```
bool estUnNombre ( )
```

estUnNombre : test si la valeur courante est nombre

Note

un test est effectuer dans yylex avant
on accepte 45aa où l'analyseur renverra Nombre = 45 et fonction = aa

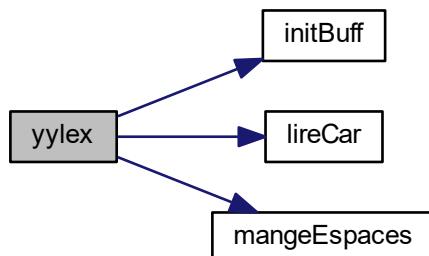
6.44.2.9 yylex()

```
int yylex (
    void )
```

yylex : Fonction principale de l'analyseur lexical, lit les caractères de yyin et renvoie les tokens sous forme d'entier. Le code de chaque unité est défini dans [symboles.h](#) sinon (mot clé, identifiant, etc.). Pour les tokens de type ID_FCT, ID_VAR et NOMBRE la valeur du token est dans yytext, visible dans l'analyseur syntaxique.

Références FIN, initBuff(), lireCar(), et mangeEspaces().

Voici le graphe d'appel pour cette fonction :



6.44.2.10 nom_token()

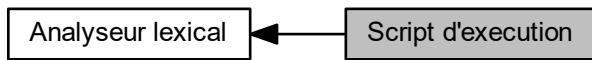
```
void nom_token (
    int token,
    char * nom,
    char * valeur )
```

nom_token : Fonction auxiliaire appelée par l'analyseur syntaxique tout simplement pour afficher des messages d'erreur et l'arbre XML

A faire voir pour remettre 'sys_instruction'

6.45 Script d'execution

Graphe de collaboration de Script d'execution :



Fonctions

— void [afficherLexique](#) (FILE *yyin, FILE *output)

afficherLexique : Fonction auxiliaire appelée par le compilo en mode -l, pour tester l'analyseur lexical et, étant donné un programme en entrée, afficher la liste des tokens.

6.45.1 Description détaillée

6.45.2 Documentation des fonctions

6.45.2.1 afficherLexique()

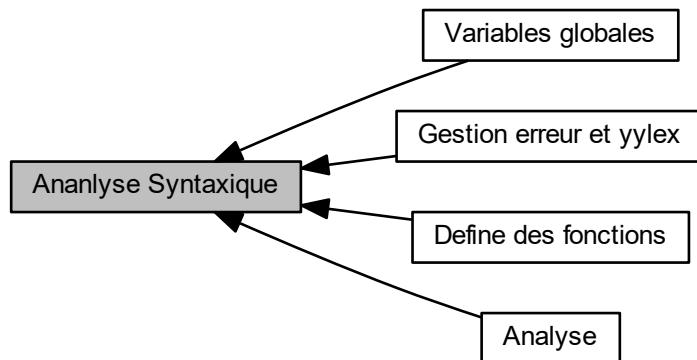
```
void afficherLexique (
    FILE * yyin,
    FILE * output )
```

afficherLexique : Fonction auxiliaire appelée par le compilo en mode -l, pour tester l'analyseur lexical et, étant donné un programme en entrée, afficher la liste des tokens.

6.46 Ananlyse Syntaxique

Partie syntaxique du compilateur.

Graphe de collaboration de Ananlyse Syntaxique :



Modules

- Define des fonctions
- Variables globales
- Gestion erreur et yylex
- Analyse

Fonctions

- void **syntaxical_setAffiche** (bool b)
syntaxicalAffiche : spécifier si l'arbre syntaxique doit s'afficher
- n_prog * **syntaxical** ()
syntaxical : appel du parseur syntaxique

6.46.1 Description détaillée

Partie syntaxique du compilateur.

Auteur

Alexis Nasr (<http://pageperso.lif.univ-mrs.fr/~alexis.nasr/>)
 released by Yannick Robin
 released by Christophe Sonntag (<http://u4a.at>)

6.46.2 Documentation des fonctions

6.46.2.1 syntaxical_setAffiche()

```
void syntaxical_setAffiche (
    bool b )
```

`syntaxicalAffiche` : spécifier si l'arbre syntaxique doit s'afficher

6.46.2.2 syntaxical()

```
n_prog* syntaxical ( )
```

`syntaxical` : appel du parseur syntaxique

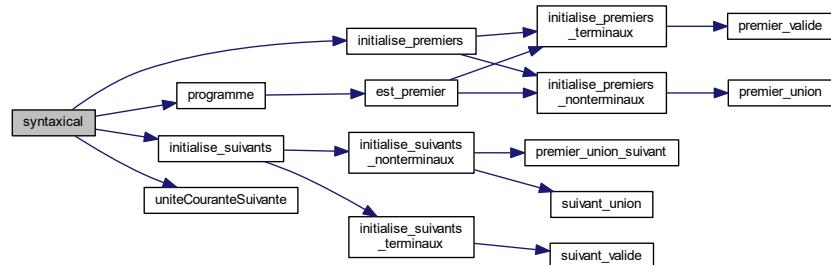
Paramètres

<code>yyin</code>	le fichier à parcourir
<code>afficherSynt</code>	

Renvoie

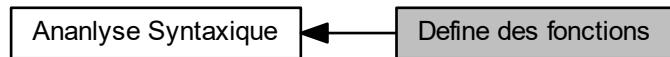
Références `initialise_premiers()`, `initialise_suivants()`, `programme()`, et `uniteCouranteSuivante()`.

Voici le graphe d'appel pour cette fonction :



6.47 Define des fonctions

Graphe de collaboration de Define des fonctions :



Macros

- #define Entre
- #define Sortie(retData)
 - necessite une variable préalablement définie 'retData'*
- #define Erreur

6.47.1 Description détaillée

6.47.2 Documentation des macros

6.47.2.1 Entre

```
#define Entre
```

Valeur :

```
if(afficheArbreSyntaxique) \
    afficheXml_balise_ouvrante(__FUNCTION__);
```

Référencé par appelFct(), comparaison(), comparaisonBis(), conjonction(), conjonctionBis(), declarationFonction(), declarationVariable(), expArith(), expArithBis(), expression(), expressionBis(), facteur(), instruction(), instruction← Affect(), instructionAppel(), instructionBloc(), instructionEcriture(), instructionFaire(), instructionIncrementer(), instructionRetour(), instructionSi(), instructionTantque(), instructionVide(), listeDecFonctions(), listeDecVariables(), listeDecVariablesBis(), listeExpressions(), listeExpressionsBis(), listeInstructions(), listeParam(), negation(), opt← DecVariables(), optIndice(), optListeDecVariables(), optSinon(), optTailleTableau(), programme(), terme(), terme← Bis(), et var().

6.47.2.2 Sortie

```
#define Sortie(
    retData )
```

Valeur :

```
if(afficheArbreSyntaxique) \
    afficheXml_balise_fermante(__FUNCTION__); \
return retData;
```

necessite une variable préalablement définie 'retData'

Avertissement

Sortie doit impérativement renvoyer une variable et non une fonction ! car sinon la fonction est exécutée lors du 'return' c'est à dire, après la fonction d'affichage 'afficheXml_balise_fermante'

6.47.2.3 Erreur

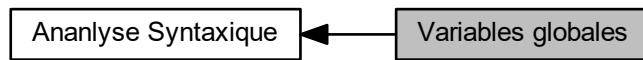
```
#define Erreur
```

Valeur :

```
erreurSyntaxical(__FUNCTION__); \
return NULL;
```

6.48 Variables globales

Graphe de collaboration de Variables globales :



6.48.1 Description détaillée

6.49 Gestion erreur et yylex

Graphe de collaboration de Gestion erreur et yylex :



Fonctions

- void `uniteCouranteSuivante ()`
- void `erreurSyntaxical (const char *fonctionName)`

6.49.1 Description détaillée

6.49.2 Documentation des fonctions

6.49.2.1 `uniteCouranteSuivante()`

```
void uniteCouranteSuivante ( )
```

Passe à l'uniteCourante suivante et affiche le lexique (si existant) dans la sortie (si ok)

Référencé par `syntaxical()`.

Voici le graphe des appels de cette fonction :

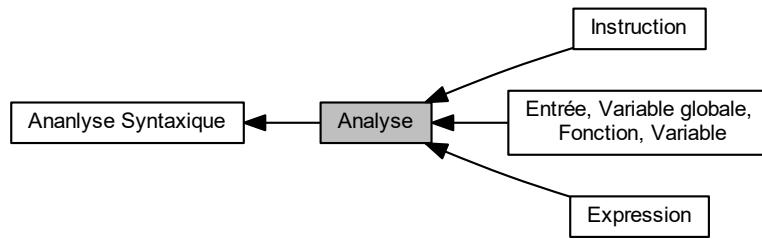


6.49.2.2 `erreurSyntaxical()`

```
void erreurSyntaxical (   
    const char * fonctionName )
```

6.50 Analyse

Graphe de collaboration de Analyse :



Modules

- Entrée, Variable globale, Fonction, Variable
- Instruction
- Expression

6.50.1 Description détaillée

6.51 Entrée, Variable globale, Fonction, Variable

Graphe de collaboration de Entrée, Variable globale, Fonction, Variable :



Fonctions

```

— n_prog * programme ()
— n_l_dec * optDecVariables ()
— n_l_dec * listeDecVariables ()
— n_l_dec * listeDecVariablesBis ()
— n_dec * declarationVariable ()
— n_dec * optTailleTableau (char *nomVariable)
— n_l_dec * listeDecFonctions ()
— n_dec * declarationFonction ()
— n_l_dec * listeParam ()
— n_l_dec * optListeDecVariables () 
  
```

6.51.1 Description détaillée

A faire Ajouter opérateur 'neg' : '-'

6.51.2 Documentation des fonctions

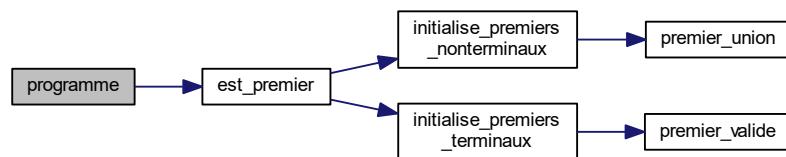
6.51.2.1 programme()

```
n_prog* programme ( )
```

Références Entre, et est_premier().

Référencé par syntaxical().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

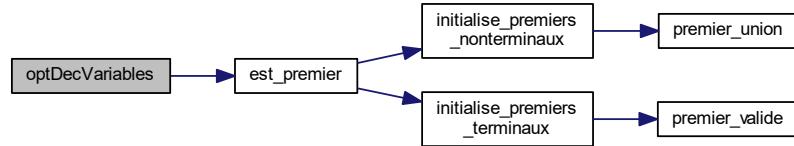


6.51.2.2 optDecVariables()

```
n_1_dec* optDecVariables ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :

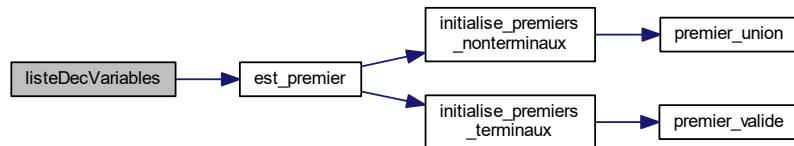


6.51.2.3 listeDecVariables()

```
n_1_dec* listeDecVariables ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.51.2.4 listeDecVariablesBis()

```
n_l_dec* listeDecVariablesBis ( )
```

Références Entre.

6.51.2.5 declarationVariable()

```
n_dec* declarationVariable ( )
```

Références Entre.

6.51.2.6 optTailleTableau()

```
n_dec* optTailleTableau (
    char * nomVariable )
```

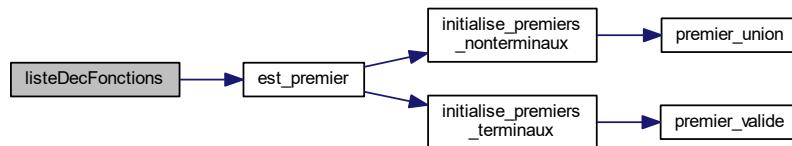
Références Entre.

6.51.2.7 listeDecFonctions()

```
n_l_dec* listeDecFonctions ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.51.2.8 declarationFonction()

```
n_dec* declarationFonction ( )
```

Références Entre.

6.51.2.9 listeParam()

```
n_l_dec* listeParam ( )
```

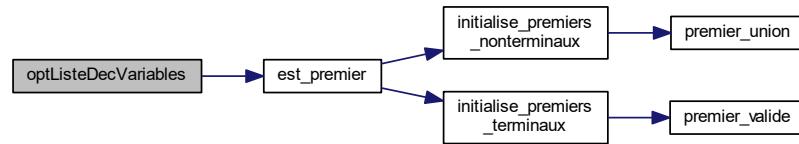
Références Entre.

6.51.2.10 optListeDecVariables()

```
n_1_dec* optListeDecVariables ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.52 Instruction

Graphe de collaboration de Instruction :



Fonctions

- n_instr * instruction ()
- n_instr * instructionAffect ()
- n_instr * instructionBloc ()
- n_l_instr * listeInstructions ()
- n_instr * instructionSi ()
- n_instr * optSinon ()
- n_instr * instructionTantque ()
- n_instr * instructionFaire ()
- n_instr * instructionAppel ()
- n_instr * instructionRetour ()
- n_instr * instructionEcriture ()
- n_instr * instructionVide ()
- n_instr * instructionIncrementer ()

6.52.1 Description détaillée

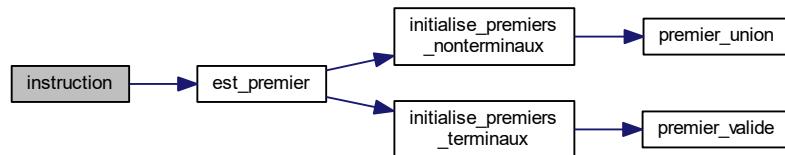
6.52.2 Documentation des fonctions

6.52.2.1 instruction()

n_instr* instruction ()

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :

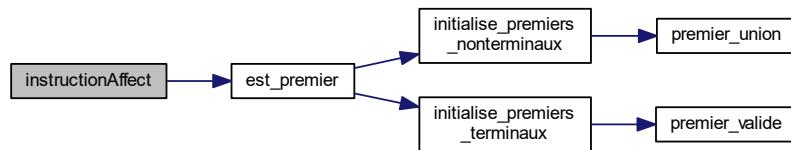


6.52.2.2 instructionAffect()

```
n_instr* instructionAffect ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.52.2.3 instructionBloc()

```
n_instr* instructionBloc ( )
```

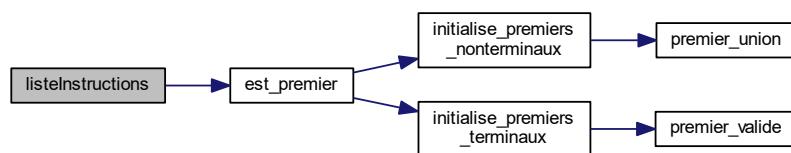
Références Entre.

6.52.2.4 listeInstructions()

```
n_l_instr* listeInstructions ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.52.2.5 instructionSi()

```
n_instr* instructionSi ( )
```

Références Entre.

6.52.2.6 optSinon()

```
n_instr* optSinon ( )
```

Références Entre.

6.52.2.7 instructionTantque()

```
n_instr* instructionTantque ( )
```

Références Entre.

6.52.2.8 instructionFaire()

```
n_instr* instructionFaire ( )
```

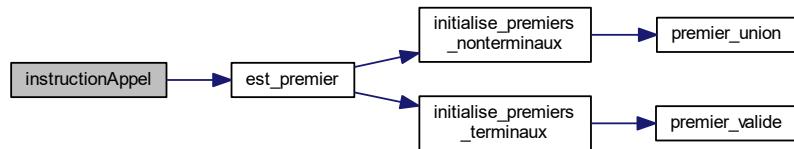
Références Entre.

6.52.2.9 instructionAppel()

```
n_instr* instructionAppel ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.52.2.10 instructionRetour()

```
n_instr* instructionRetour ( )
```

Références Entre.

6.52.2.11 instructionEcriture()

```
n_instr* instructionEcriture ( )
```

Références Entre.

6.52.2.12 instructionVide()

```
n_instr* instructionVide ( )
```

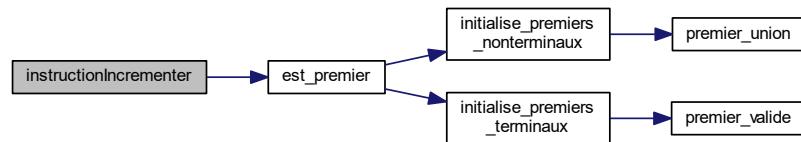
Références Entre.

6.52.2.13 instructionIncrementer()

```
n_instr* instructionIncrementer ( )
```

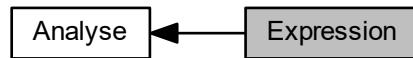
Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.53 Expression

Graphe de collaboration de Expression :



Fonctions

- n_exp * **expression ()**
- n_exp * **expressionBis** (n_exp *op1)
- n_exp * **conjonction ()**
- n_exp * **conjonctionBis** (n_exp *op1)
- n_exp * **comparaison ()**
- n_exp * **comparaisonBis** (n_exp *op1)
- n_exp * **expArith ()**
- n_exp * **expArithBis** (n_exp *op1)
- n_exp * **terme ()**
- n_exp * **termeBis** (n_exp *op1)
- n_exp * **negation ()**
- n_exp * **facteur ()**
- n_var * **var ()**
- n_var * **optIndice** (char *nomVariable)
- n_appel * **appelFct ()**
- n_l_exp * **listeExpressions ()**
- n_l_exp * **listeExpressionsBis ()**

6.53.1 Description détaillée

6.53.2 Documentation des fonctions

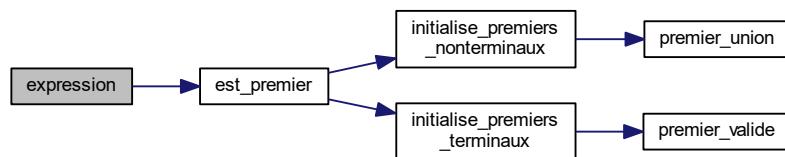
6.53.2.1 expression()

n_exp* expression ()

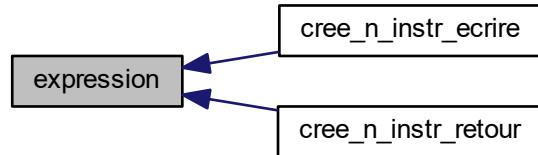
Références Entre, et est_premier().

Référencé par cree_n_instr_ecrire(), et cree_n_instr_retour().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.53.2.2 expressionBis()

```
n_exp* expressionBis (
    n_exp * op1 )
```

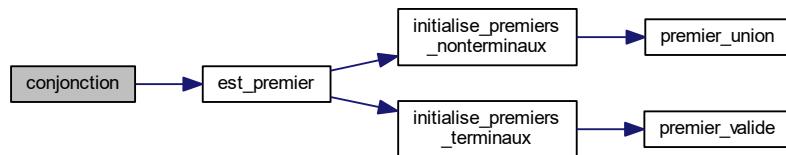
Références Entre.

6.53.2.3 conjonction()

```
n_exp* conjonction ()
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.53.2.4 conjonctionBis()

```
n_exp* conjonctionBis (
    n_exp * op1 )
```

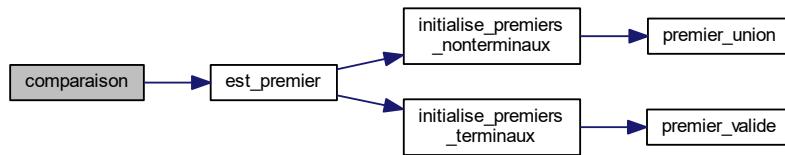
Références Entre.

6.53.2.5 comparaison()

```
n_exp* comparaison ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.53.2.6 comparaisonBis()

```
n_exp* comparaisonBis (
    n_exp * op1 )
```

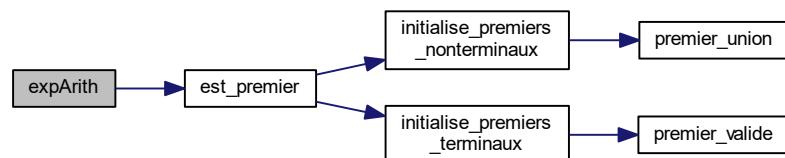
Références Entre.

6.53.2.7 expArith()

```
n_exp* expArith ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.53.2.8 expArithBis()

```
n_exp* expArithBis (
    n_exp * op1 )
```

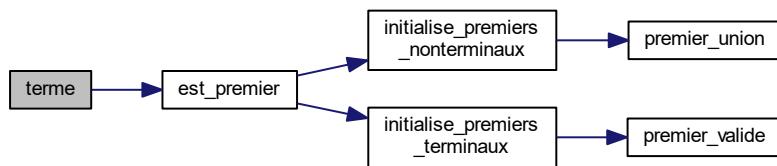
Références Entre.

6.53.2.9 terme()

```
n_exp* terme ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.53.2.10 termeBis()

```
n_exp* termeBis (
    n_exp * opl )
```

Références Entre.

6.53.2.11 negation()

```
n_exp* negation ( )
```

Références Entre.

6.53.2.12 facteur()

```
n_exp* facteur ( )
```

Références Entre.

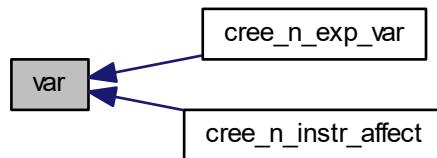
6.53.2.13 var()

```
n_var* var ( )
```

Références Entre.

Référencé par cree_n_exp_var(), et cree_n_instr_affect().

Voici le graphe des appelants de cette fonction :



6.53.2.14 optIndice()

```
n_var* optIndice (
    char * nomVariable )
```

Références Entre.

6.53.2.15 appelFct()

```
n_appel* appelFct ( )
```

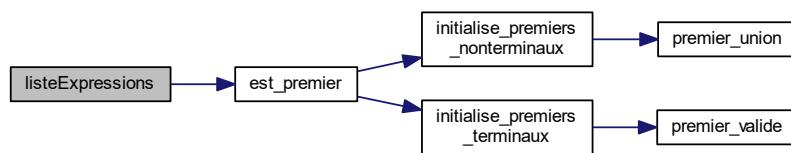
Références Entre.

6.53.2.16 listeExpressions()

```
n_l_exp* listeExpressions ( )
```

Références Entre, et est_premier().

Voici le graphe d'appel pour cette fonction :



6.53.2.17 listeExpressionsBis()

```
n_l_exp* listeExpressionsBis ( )
```

Références Entre.

6.54 /Remplissage des premiers

Fonctions

- void **premier_valide** (int non_terminal, int terminal)
premier_valide valide un terminal de la table 'premier' pour un 'non_terminal' donné
- void **premier_union** (int idSource, int idDestination)
premier_union permet de copier un groupe de terminaux de la table 'premier'.
- void **initialise_premiers_terminaux** ()
- void **initialise_premiers_nonterminaux** ()
- void **suivant_valide** (int non_terminal, int terminal)
suivant_valide valide un terminal de la table 'suivant' pour un 'non_terminal' donné
- void **suivant_union** (int idSource, int idDestination)
suivant_union permet de copier un groupe de terminaux de la table 'suivant'.
- void **premier_union_suivant** (int idSourcePremier, int idDestinationSuivant)
premier_union_suivant permet de copier un groupe de terminaux de la table 'premiers' vers la table 'suivant'.
- void **initialise_suivants_terminaux** ()
- void **initialise_suivants_nonterminaux** ()

6.54.1 Description détaillée

6.54.2 Documentation des fonctions

6.54.2.1 premier_valide()

```
void premier_valide (
    int non_terminal,
    int terminal )
```

premier_valide valide un terminal de la table 'premier' pour un 'non_terminal' donné

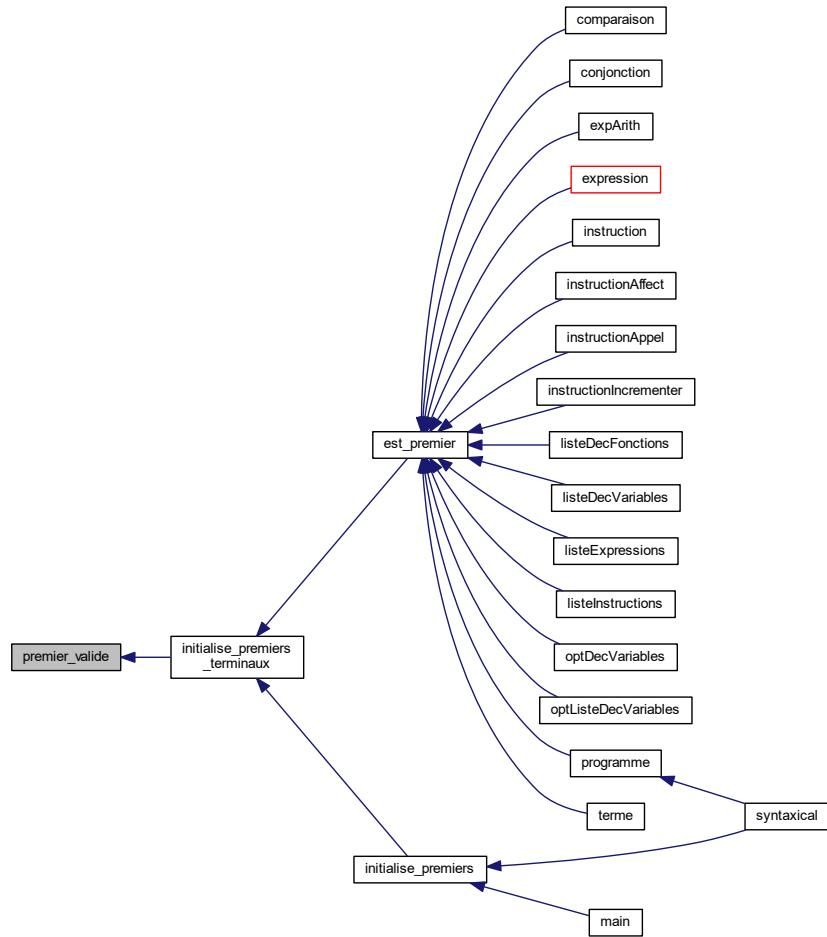
Paramètres

<i>non_terminal</i>	le 'non_terminal' qui reçoi le 'terminal' à valider
<i>terminal</i>	le 'terminal' à valider

Références premiers.

Référencé par initialise_premiers_terminaux().

Voici le graphe des appelants de cette fonction :



6.54.2.2 premier_union()

```
void premier_union (
    int idSource,
    int idDestination )
```

premier_union permet de copier un groupe de terminaux de la table 'premier'.

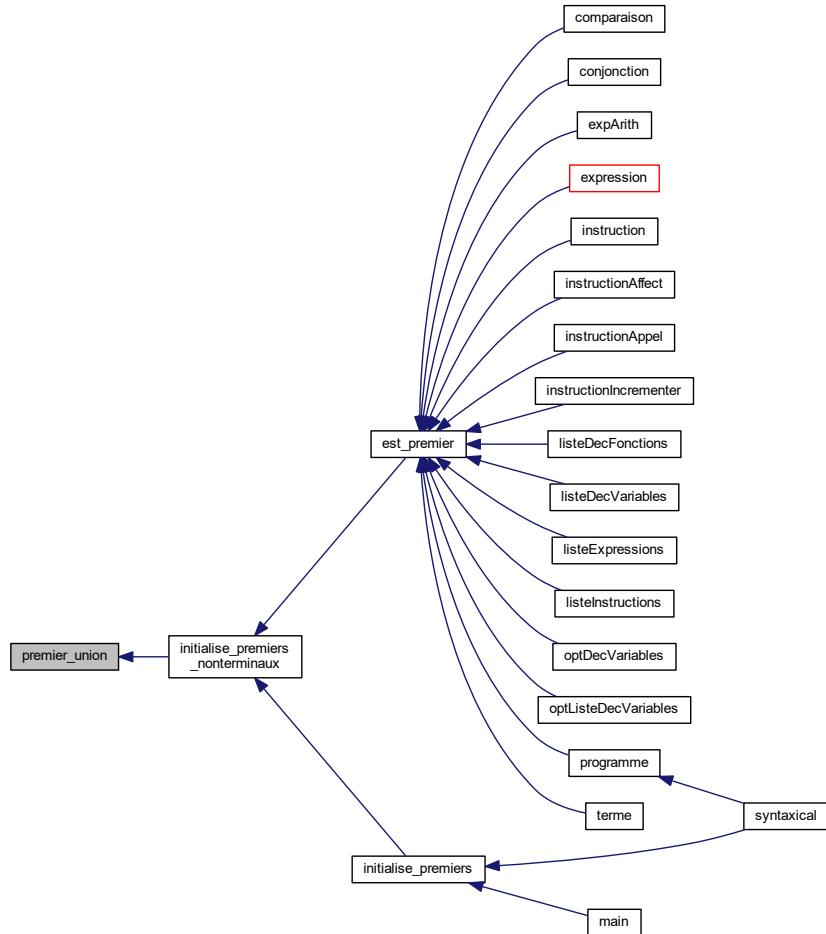
Paramètres

<i>idSource</i>	l'id de la table 'premier' à copier
<i>idDestination</i>	l'id de la table 'premier' qui reçoit

Références NB_TERMINAUX, et premiers.

Référencé par initialise_premiers_nonterminaux().

Voici le graphe des appels de cette fonction :



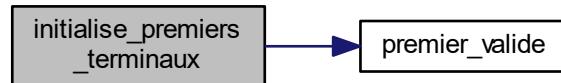
6.54.2.3 initialise_premiers_terminaux()

```
void initialise_premiers_terminaux ( )
```

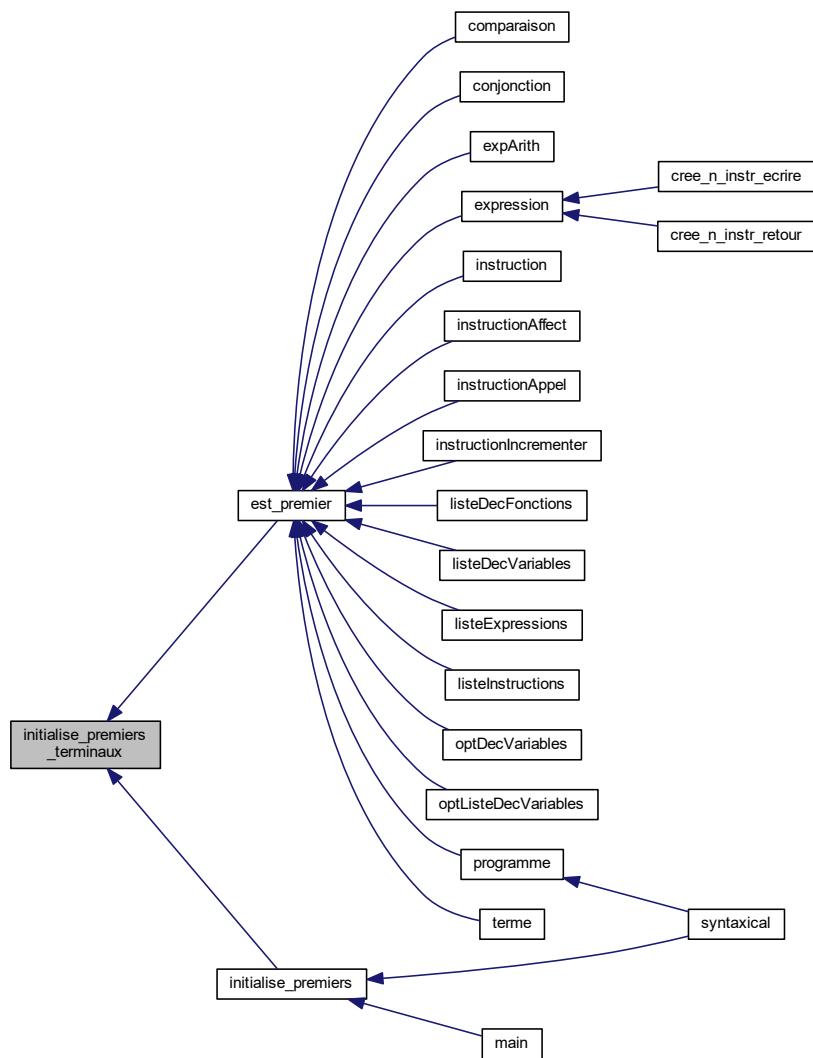
Références _appelFct_, _comparaisonBis_, _conjonctionBis_, _declarationFonction_, _declarationVariable_, _expArithBis_, _expressionBis_, _facteur_, _instructionBloc_, _instructionEcriture_, _instructionFaire_, _instructionIncrementer_, _instructionRetour_, _instructionSi_, _instructionTantque_, _instructionVide_, _listeDecFonctions_, _listeDecVariablesBis_, _listeExpressions_, _listeExpressionsBis_, _listInstructions_, _listeParam_, _negation_, _optDecVariables_, _optIndice_, _optListeDecVariables_, _optSinon_, _optTailleTableau_, _termeBis_, _var_, ACCOLADE_OUVRANTE, CROCHET_OUVRANT, DIFFERENT, DIVISE, ECRIRE, EGAL, ENTIER, EPSILON, ET, FAIRE, FOIS, ID_FCT, ID_VAR, INCR, INFERIEUR, INFERIEUR_EGAL, LIRE, MODULO, MOINS, NOMBRE, NON, OU, PARENTHESI_OUVRANTE, PLUS, POINT_VIRGULE, premier_valide(), RETOUR, SI, SINON, SUPERIEUR, SUPERIEUR_EGAL, TANTQUE, et VIRGULE.

Référencé par `est_premier()`, et `initialise_premiers()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



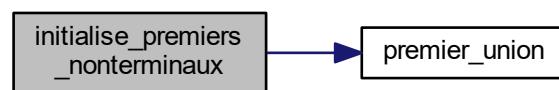
6.54.2.4 initialise_premiers_nonterminaux()

```
void initialise_premiers_nonterminaux ( )
```

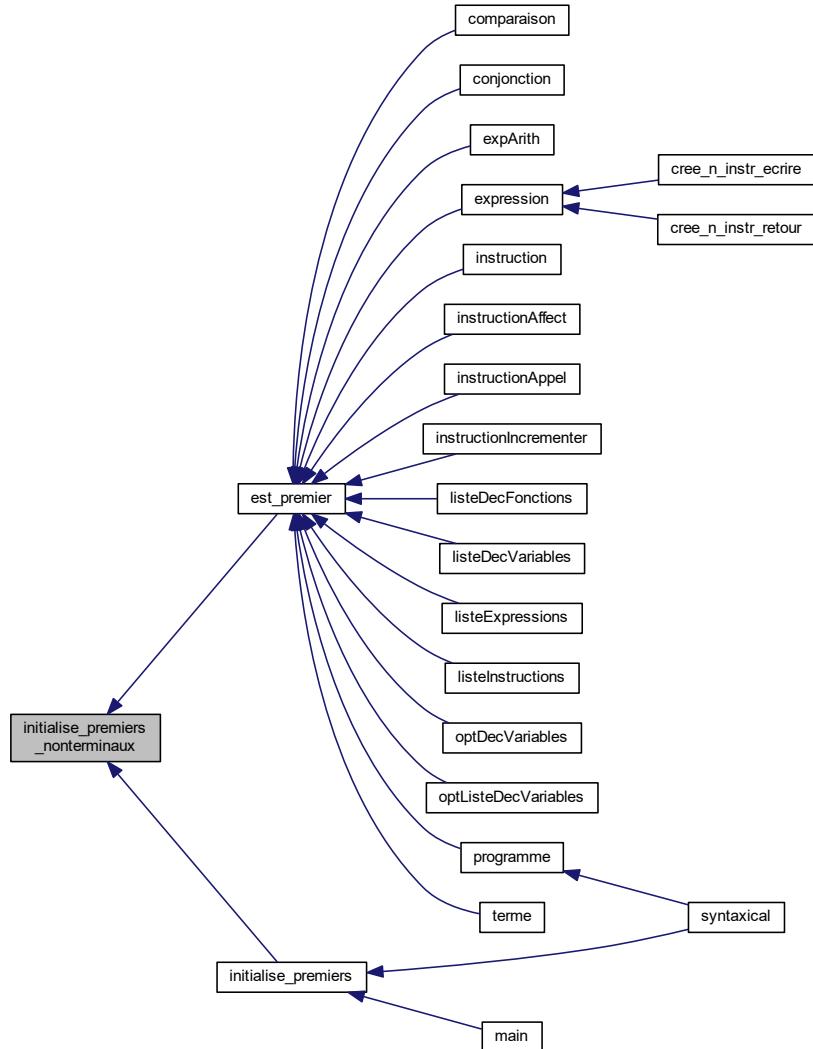
Références _appelFct_, _comparaison_, _conjonction_, _declarationFonction_, _declarationVariable_, _expArith←_, _expression_, _facteur_, _instruction_, _instructionAffect_, _instructionAppel_, _instructionBloc_, _instruction←Ecriture_, _instructionFaire_, _instructionIncrementer_, _instructionRetour_, _instructionSi_, _instructionTantque←_, _instructionVide_, _listeDecFonctions_, _listeDecVariables_, _listeExpressions_, _listeInstructions_, _←negation_, _optDecVariables_, _optListeDecVariables_, _programme_, _terme_, _var_, et premier_union().

Référencé par est_premier(), et initialise_premiers().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.54.2.5 suivant_valide()

```
void suivant_valide (
    int non_terminal,
    int terminal )
```

`suisant_valide` valide un terminal de la table 'suivant' pour un 'non_terminal' donné

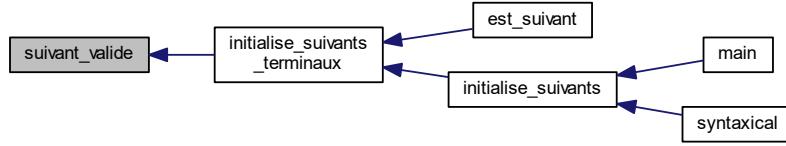
Paramètres

<code>non_terminal</code>	le 'non_terminal' qui reçoit le 'terminal' à valider
<code>terminal</code>	le 'terminal' à valider

Références suivants.

Référencé par initialise_suivants_terminaux().

Voici le graphe des appelants de cette fonction :



6.54.2.6 suivant_union()

```
void suivant_union (
    int idSource,
    int idDestination )
```

suivant_union permet de copier un groupe de terminaux de la table 'suivant'.

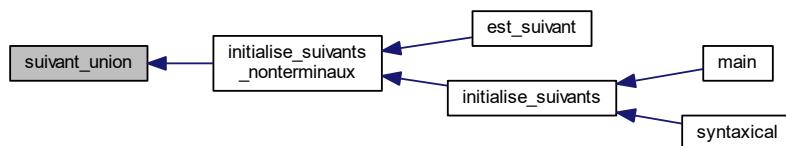
Paramètres

<i>idSource</i>	l'id de la table 'suivant' à copier
<i>idDestination</i>	l'id de la table 'suivant' qui reçoit

Références NB_TERMINAUX, et suivants.

Référencé par initialise_suivants_nonterminaux().

Voici le graphe des appelants de cette fonction :



6.54.2.7 premier_union_suivant()

```
void premier_union_suivant (
    int idSourcePremier,
    int idDestinationSuivant )
```

premier_union_suivant permet de copier un groupe de terminaux de la table 'premiers' vers la table 'suivant'.

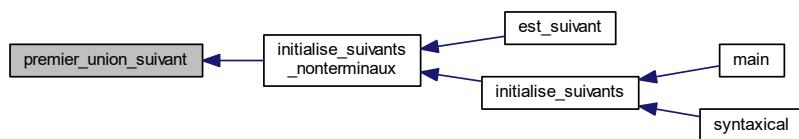
Paramètres

<i>idSourcePremier</i>	l'id de la table 'premier' à copier
<i>idDestinationSuivant</i>	l'id de la table 'suivant' qui reçoit

Références EPSILON, NB_TERMINAUX, premiers, et suivants.

Référencé par initialise_suivants_nonterminaux().

Voici le graphe des appels de cette fonction :



6.54.2.8 initialise_suivants_terminaux()

```
void initialise_suivants_terminaux ( )
```

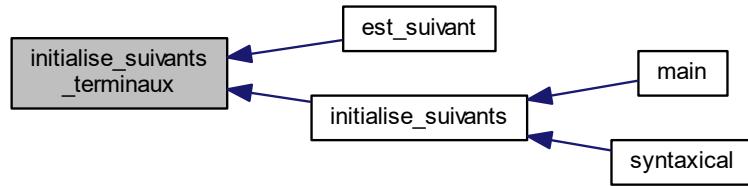
Références _appelFct_, _comparaison_, _conjonction_, _expArith_, _expression_, _instructionBloc_, _listeDec_, Fonctions_, _listeDecVariables_, _listeExpressions_, _listeExpressionsBis_, _listInstructions_, _optListeDec_, Variables_, _programme_, _terme_, _termeBis_, ACCOLADE_FERMANTE, ALORS, CROCHET_FERMANT, DIFFERENT, DIVISE, EGAL, ET, FAIRE, FIN, FOIS, INFERIEUR, INFERIEUR_EGAL, MODULO, MOINS, NON, OU, PARENTHESE_FERMANTE, PLUS, POINT_VIRGULE, suivant_valide(), SUPERIEUR, SUPERIEUR_EGAL, et TANTQUE.

Référencé par `est_suivant()`, et `initialise_suivants()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



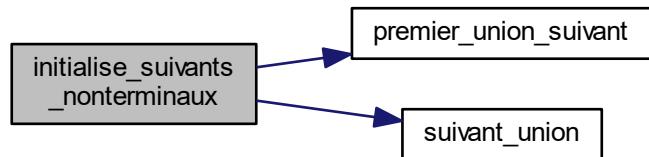
6.54.2.9 initialise_suivants_nonterminaux()

```
void initialise_suivants_nonterminaux ( )
```

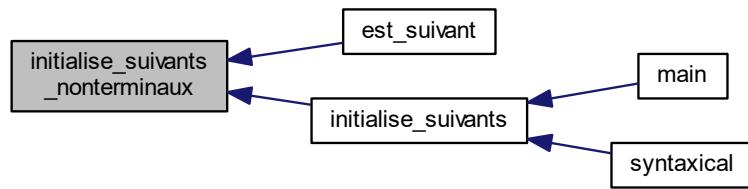
Références `_appelFct_`, `_comparaison_`, `_comparaisonBis_`, `_conjonction_`, `_conjonctionBis_`, `_declarationFonction_`, `_declarationVariable_`, `_expArith_`, `_expArithBis_`, `_expression_`, `_expressionBis_`, `_facteur_`, `_instruction_`, `_instructionAffect_`, `_instructionAppel_`, `_instructionBloc_`, `_instructionEcriture_`, `_instructionFaire_`, `_instructionRetour_`, `_instructionSi_`, `_instructionTantque_`, `_instructionVide_`, `_listeDecFonctions_`, `_listeDecVariables_`, `_listeDecVariablesBis_`, `_listeExpressionsBis_`, `_listelInstructions_`, `_listeParam_`, `_negation_`, `_optDecVariables_`, `_optIndice_`, `_optListeDecVariables_`, `_optSinon_`, `_optTailleTableau_`, `_terme_`, `_termeBis_`, `_var_`, `premier_union_suivant()`, et `suivant_union()`.

Référencé par `est_suivant()`, et `initialise_suivants()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



Chapitre 7

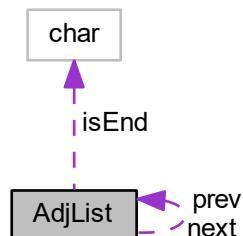
Documentation des structures de données

7.1 Référence de la structure AdjList

///

```
#include <circularAdjList.h>
```

Graphe de collaboration de AdjList :



Champs de données

- struct `AdjList` * `next`
- struct `AdjList` * `prev`
- char `isEnd`

7.1.1 Description détaillée

///

7.1.2 Documentation des champs

7.1.2.1 next

```
struct AdjList* next
```

Référencé par circularAdjList_add_(), circularAdjList_free_(), et circularAdjList_search_().

7.1.2.2 prev

```
struct AdjList * prev
```

Référencé par circularAdjList_add_().

7.1.2.3 isEnd

```
char isEnd
```

Référencé par circularAdjList_add_(), circularAdjList_free_(), et circularAdjList_search_().

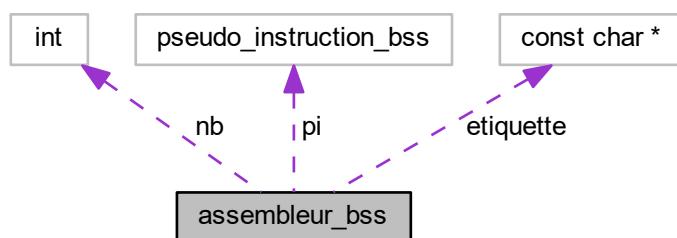
La documentation de cette structure a été générée à partir du fichier suivant :

- [circularAdjList.h](#)

7.2 Référence de la structure assembleur_bss

[assembleur_bss](#) : donnée pour section bss

Graphe de collaboration de assembleur_bss :



Champs de données

- const char * [etiquette](#)
- [pseudo_instruction_bss](#) pi
- unsigned int [nb](#)

7.2.1 Description détaillée

`assembleur_bss` : donnée pour section bss

7.2.2 Documentation des champs

7.2.2.1 `etiquette`

```
const char* etiquette
```

7.2.2.2 `pi`

```
pseudo_instruction_bss pi
```

7.2.2.3 `nb`

```
unsigned int nb
```

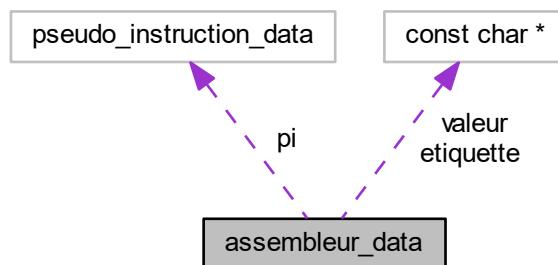
La documentation de cette structure a été générée à partir du fichier suivant :

— `assembleur.c`

7.3 Référence de la structure assembleur_data

`assembleur_data` : donnée pour section data

Graphe de collaboration de `assembleur_data` :



Champs de données

- `const char * etiquette`
- `pseudo_instruction_data pi`
- `const char * valeur`

7.3.1 Description détaillée

`assembleur_data` : donnée pour section data

7.3.2 Documentation des champs

7.3.2.1 étiquette

```
const char* étiquette
```

7.3.2.2 pi

```
pseudo_instruction_data pi
```

7.3.2.3 valeur

```
const char* valeur
```

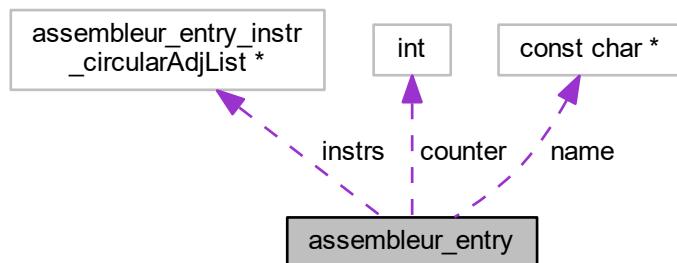
La documentation de cette structure a été générée à partir du fichier suivant :

— [assembleur.c](#)

7.4 Référence de la structure assembleur_entry

`assembleur_entry` : donnée pour une entrée

Graphe de collaboration de `assembleur_entry` :



Champs de données

- `const char * name`
- `assembleur_entry_instr_circularAdjList * instrs`
- `unsigned int counter`

7.4.1 Description détaillée

`assembleur_entry` : donnée pour une entrée

7.4.2 Documentation des champs

7.4.2.1 name

```
const char* name
```

Référencé par `assembleur_entry_getName()`, `assembleur_entry_make_next()`, `assembleur_entry_make_prefix()`, et `assembleur_entry_make_suffix()`.

7.4.2.2 instrs

```
assembleur_entry_instr_circularAdjList* instrs
```

Référencé par `assembleur_dump_section_entry_instrs()`, `assembleur_entry_addEntryPoint()`, `assembleur_entry_add_subEntry()`, et `assembleur_entry_getLastInstr()`.

7.4.2.3 counter

```
unsigned int counter
```

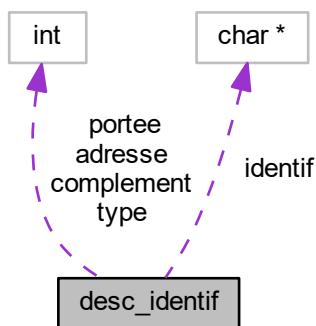
Référencé par `assembleur_entry_make_next()`, `assembleur_entry_make_prefix()`, et `assembleur_entry_make_suffix()`.

La documentation de cette structure a été générée à partir du fichier suivant :
— [assembleur.c](#)

7.5 Référence de la structure desc_identif

```
#include <tabsymboles.h>
```

Graphe de collaboration de `desc_identif` :



Champs de données

- char * **identif**
- int **portee**
- int **type**
- unsigned int **adresse**
- unsigned int **complement**

7.5.1 Description détaillée

une entrée de la table des symboles

7.5.2 Documentation des champs

7.5.2.1 **identif**

char* identif

Nom de l'identificateur

Référencé par tabsymb_ajoutelidentificateur(), tabsymb_rechercheDeclarative(), et tabsymb_rechercheExecutable().

7.5.2.2 **portee**

int portee

Valeurs possibles P_VARIABLE_GLOBALE, P_VARIABLE_LOCALE, P_ARGUMENT

Référencé par parcoursTools_accesVar(), et tabsymb_ajoutelidentificateur().

7.5.2.3 **type**

int type

Valeurs possibles T_ENTIER, T_TABLEAU_ENTIER et T_FONCTION

Référencé par parcours_exp_appel(), parcours_prog(), parcoursTools_accesVar(), tabsymb_ajoutelidentificateur(), et tabsymb_getSymboleFonctionCourante().

7.5.2.4 **adresse**

unsigned int adresse

décalage à partir de \$fp ou .data nombre d'octets

Référencé par parcoursTools_accesVar(), et tabsymb_ajoutelidentificateur().

7.5.2.5 complement

```
unsigned int complement
```

taille d'un tableau OU nombre d'arguments d'une fonction

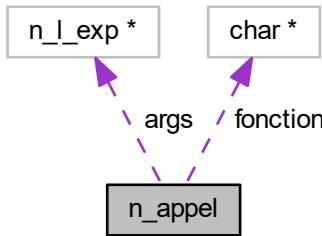
Référencé par parcours_exp_appel(), parcours_instr_retour(), parcoursTools_accesVar(), et tabsymb_ajouteIdentificateur().

La documentation de cette structure a été générée à partir du fichier suivant :
— [tabsymboles.h](#)

7.6 Référence de la structure n_appel

```
#include <syntabs.h>
```

Graphe de collaboration de n_appel :



Champs de données

- char * fonction
- n_l_exp * args

7.6.1 Documentation des champs

7.6.1.1 fonction

```
char* fonction
```

7.6.1.2 args

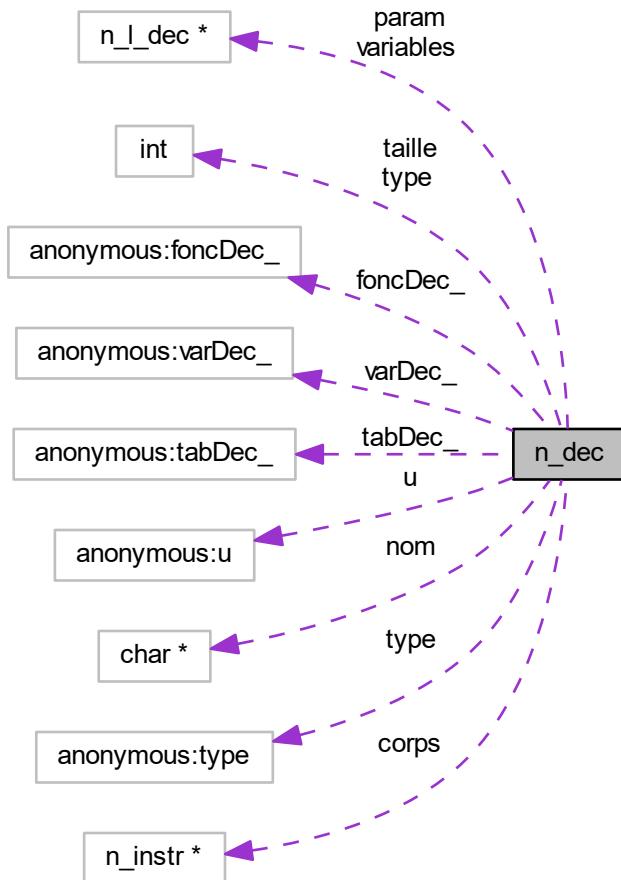
```
n_l_exp* args
```

La documentation de cette structure a été générée à partir du fichier suivant :
— [syntabs.h](#)

7.7 Référence de la structure n_dec

```
#include <syntabs.h>
```

Graphe de collaboration de n_dec :



Types publics

- enum { foncDec, varDec, tabDec }

Champs de données

```

— enum n_dec_:::{ ... } type
— char * nom
— union {
    struct {
        n_I_dec * param
        n_I_dec * variables
        n_instr * corps
    }
}
```

```
} fncDec_
struct {
    int type
} varDec_
struct {
    int taille
} tabDec_
} u
```

7.7.1 Documentation des énumérations membres

7.7.1.1 anonymous enum

anonymous enum

Valeurs énumérées

fncDec	
varDec	
tabDec	

7.7.2 Documentation des champs

7.7.2.1 type [1/2]

enum { ... } type

7.7.2.2 nom

char* nom

7.7.2.3 param

n_1_dec* param

7.7.2.4 variables

n_1_dec* variables

7.7.2.5 corps

n_instr* corps

7.7.2.6 fonzDec_

```
struct { ... } fonzDec_
```

7.7.2.7 type [2/2]

```
int type
```

7.7.2.8 varDec_

```
struct { ... } varDec_
```

7.7.2.9 taille

```
int taille
```

7.7.2.10 tabDec_

```
struct { ... } tabDec_
```

7.7.2.11 u

```
union { ... } u
```

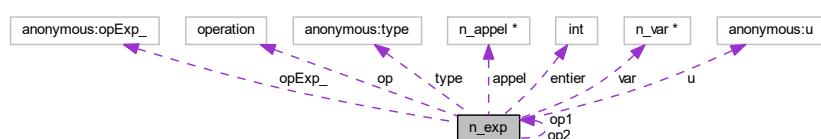
La documentation de cette structure a été générée à partir du fichier suivant :

— [syntabs.h](#)

7.8 Référence de la structure n_exp

```
#include <syntabs.h>
```

Graphe de collaboration de n_exp :



Types publics

```
— enum {  
    varExp, opExp, intExp, appelExp,  
    lireExp }
```

Champs de données

```
— enum n_exp_ :: { ... } type  
— union {  
    struct {  
        operation op  
        struct n_exp_* op1  
        struct n_exp_* op2  
    } opExp_  
    n_var * var  
    int entier  
    n_appel * appel  
} u
```

7.8.1 Documentation des énumérations membres

7.8.1.1 anonymous enum

anonymous enum

Valeurs énumérées

varExp	
opExp	
intExp	
appelExp	
lireExp	

7.8.2 Documentation des champs

7.8.2.1 type

enum { ... } type

7.8.2.2 op

operation op

7.8.2.3 op1

struct n_exp_* op1

7.8.2.4 op2

```
struct n_exp_* op2
```

7.8.2.5 opExp_

```
struct { ... } opExp_
```

7.8.2.6 var

```
n_var* var
```

7.8.2.7 entier

```
int entier
```

7.8.2.8 appel

```
n_appel* appel
```

7.8.2.9 u

```
union { ... } u
```

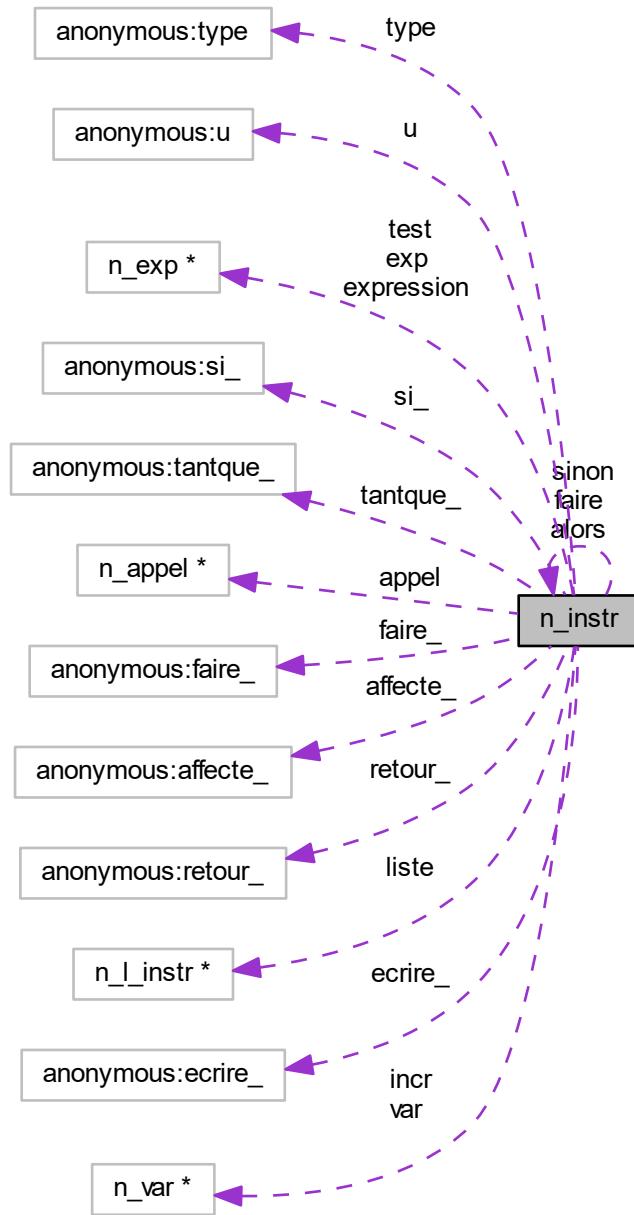
La documentation de cette structure a été générée à partir du fichier suivant :

— [syntabs.h](#)

7.9 Référence de la structure n_instr

```
#include <syntabs.h>
```

Graphe de collaboration de n_instr :



Types publics

```

— enum {
  incrInst, affecteInst, silInst, faireInst,
  tantqueInst, appelleInst, retourInst, ecrireInst,
  videlInst, blocInst
}
  
```

Champs de données

```

— enum n_instr_ :: { ... } type
— union {
    n_var * incr
    struct {
        n_exp * test
        struct n_instr_ * alors
        struct n_instr_ * sinon
    } si_
    struct {
        n_exp * test
        struct n_instr_ * faire
    } tantque_
    struct {
        n_exp * test
        struct n_instr_ * faire
    } faire_
    n_appel * appel
    struct {
        n_var * var
        n_exp * exp
    } affecte_
    struct {
        n_exp * expression
    } retour_
    struct {
        n_exp * expression
    } ecrire_
    n_l_instr * liste
} u

```

7.9.1 Documentation des énumérations membres

7.9.1.1 anonymous enum

anonymous enum

Valeurs énumérées

incrInst	
affecteInst	
silInst	
faireInst	
tantqueInst	
appellInst	
retourInst	
ecrireInst	
videInst	
blocInst	

7.9.2 Documentation des champs

7.9.2.1 type

```
enum { ... } type
```

7.9.2.2 incr

```
n_var* incr
```

7.9.2.3 test

```
n_exp* test
```

7.9.2.4 alors

```
struct n_instr_* alors
```

7.9.2.5 sinon

```
struct n_instr_* sinon
```

7.9.2.6 si_

```
struct { ... } si_
```

7.9.2.7 faire

```
struct n_instr_* faire
```

7.9.2.8 tantque_

```
struct { ... } tantque_
```

7.9.2.9 faire_

```
struct { ... } faire_
```

7.9.2.10 appel

```
n_appel* appel
```

7.9.2.11 var

```
n_var* var
```

7.9.2.12 exp

```
n_exp* exp
```

7.9.2.13 affecte_

```
struct { ... } affecte_
```

7.9.2.14 expression

```
n_exp* expression
```

7.9.2.15 retour_

```
struct { ... } retour_
```

7.9.2.16 ecrire_

```
struct { ... } ecrire_
```

7.9.2.17 liste

```
n_l_instr* liste
```

7.9.2.18 u

```
union { ... } u
```

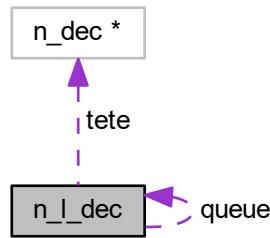
La documentation de cette structure a été générée à partir du fichier suivant :

— [syntabs.h](#)

7.10 Référence de la structure n_l_dec

```
#include <syntabs.h>
```

Graphe de collaboration de n_l_dec :



Champs de données

- n_dec * **tete**
- struct **n_l_dec_** * **queue**

7.10.1 Documentation des champs

7.10.1.1 tete

`n_dec* tete`

7.10.1.2 queue

`struct n_l_dec_* queue`

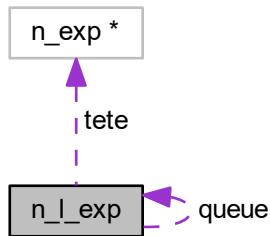
La documentation de cette structure a été générée à partir du fichier suivant :

- [syntabs.h](#)

7.11 Référence de la structure n_l_exp

```
#include <syntabs.h>
```

Graphe de collaboration de n_l_exp :



Champs de données

- n_exp * [tete](#)
- struct [n_l_exp_](#) * [queue](#)

7.11.1 Documentation des champs

7.11.1.1 [tete](#)

```
n_exp* tete
```

7.11.1.2 [queue](#)

```
struct n_l_exp_* queue
```

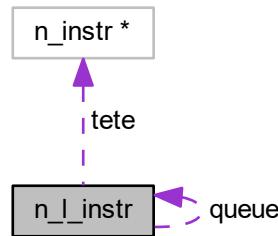
La documentation de cette structure a été générée à partir du fichier suivant :

- [syntabs.h](#)

7.12 Référence de la structure n_l_instr

```
#include <syntabs.h>
```

Graphe de collaboration de n_l_instr :



Champs de données

- n_instr * **tete**
- struct n_l_instr_ * **queue**

7.12.1 Documentation des champs

7.12.1.1 tete

```
n_instr* tete
```

7.12.1.2 queue

```
struct n_l_instr_* queue
```

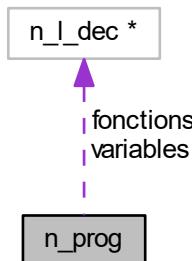
La documentation de cette structure a été générée à partir du fichier suivant :

- [syntabs.h](#)

7.13 Référence de la structure n_prog

```
#include <syntabs.h>
```

Graphe de collaboration de n_prog :



Champs de données

- n_l_dec * variables
- n_l_dec * fonctions

7.13.1 Documentation des champs

7.13.1.1 variables

n_l_dec* variables

7.13.1.2 fonctions

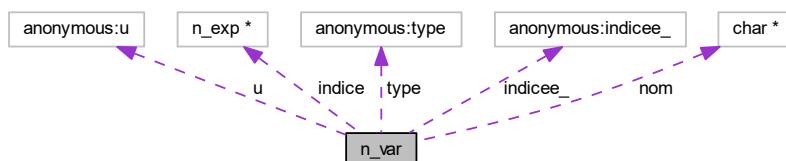
n_l_dec* fonctions

La documentation de cette structure a été générée à partir du fichier suivant : [syntabs.h](#)

7.14 Référence de la structure n_var

```
#include <syntabs.h>
```

Graphe de collaboration de n_var :



Types publics

— enum { simple, indicee }

Champs de données

```
— enum n_var_:::{ ... } type
— char * nom
— union {
    struct {
        n_exp * indice
    } indicee_
} u
```

7.14.1 Documentation des énumérations membres

7.14.1.1 anonymous enum

anonymous enum

Valeurs énumérées

simple	
indicee	

7.14.2 Documentation des champs

7.14.2.1 type

enum { ... } type

7.14.2.2 nom

char* nom

7.14.2.3 indice

n_exp* indice

7.14.2.4 indicee_

struct { ... } indicee_

7.14.2.5 u

```
union { ... } u
```

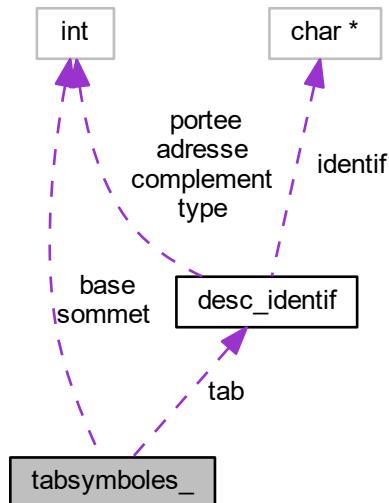
La documentation de cette structure a été générée à partir du fichier suivant :

- [syntabs.h](#)

7.15 Référence de la structure tabsymboles_

`tabsymboles` : Table des symboles (globale ET locale)

Graphe de collaboration de `tabsymboles_` :



Champs de données

- `desc_identif tab [1000]`
- `int base`
- `int sommet`

7.15.1 Description détaillée

`tabsymboles` : Table des symboles (globale ET locale)

des symboles' avec le tableau et deux indices pour le contexte

7.15.2 Documentation des champs

7.15.2.1 tab

```
desc_identif tab[1000]
```

Référencé par tabsymb_ajoutelidentificateur(), tabsymb_getSymboleFonctionCourante(), tabsymb_rechercheDeclarative(), et tabsymb_rechercheExecutable().

7.15.2.2 base

```
int base
```

base=0 : contexte global, base=sommet contexte local

Référencé par tabsymb_entreeFonction(), tabsymb_getSymboleFonctionCourante(), tabsymb_rechercheDeclarative(), et tabsymb_sortieFonction().

7.15.2.3 sommet

```
int sommet
```

pointe toujours vers la prochaine ligne disponible du tableau

Référencé par tabsymb_ajoutelidentificateur(), tabsymb_entreeFonction(), tabsymb_rechercheDeclarative(), tabsymb_rechercheExecutable(), et tabsymb_sortieFonction().

La documentation de cette structure a été générée à partir du fichier suivant :

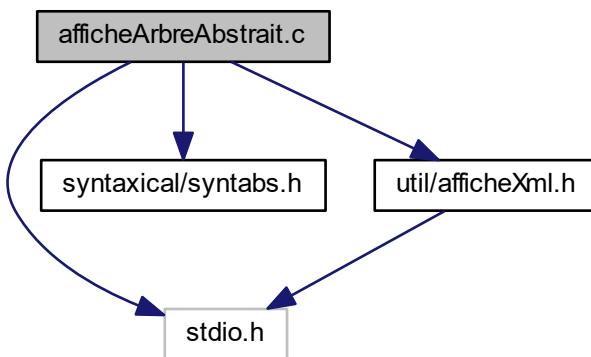
— [tabsymboles.c](#)

Chapitre 8

Documentation des fichiers

8.1 Référence du fichier afficheArbreAbstrait.c

```
#include <stdio.h>
#include "syntaxical/syntabs.h"
#include "util/afficheXml.h"
Graphe des dépendances par inclusion de afficheArbreAbstrait.c :
```



Fonctions

```
— void afficheArbreAbstrait_n_prog (n_prog *n)
— void afficheArbreAbstrait_l_instr (n_l_instr *n)
— void afficheArbreAbstrait_instr (n_instr *n)
— void afficheArbreAbstrait_instr_si (n_instr *n)
— void afficheArbreAbstrait_instr_tantque (n_instr *n)
— void afficheArbreAbstrait_instr_faire (n_instr *n)
— void afficheArbreAbstrait_instr_affect (n_instr *n)
— void afficheArbreAbstrait_instr_appel (n_instr *n)
— void afficheArbreAbstrait_appel (n_appel *n)
— void afficheArbreAbstrait_instr_retour (n_instr *n)
— void afficheArbreAbstrait_instr_ecrire (n_instr *n)
— void afficheArbreAbstrait_l_exp (n_l_exp *n)
```

```
— void afficheArbreAbstrait_exp (n_exp *n)
— void afficheArbreAbstrait_varExp (n_exp *n)
— void afficheArbreAbstrait_opExp (n_exp *n)
— void afficheArbreAbstrait_intExp (n_exp *n)
— void afficheArbreAbstrait_lireExp (n_exp *n)
— void afficheArbreAbstrait_appelExp (n_exp *n)
— void afficheArbreAbstrait_l_dec (n_l_dec *n)
— void afficheArbreAbstrait_dec (n_dec *n)
— void afficheArbreAbstrait_foncDec (n_dec *n)
— void afficheArbreAbstrait_varDec (n_dec *n)
— void afficheArbreAbstrait_tabDec (n_dec *n)
— void afficheArbreAbstrait_var (n_var *n)
— void afficheArbreAbstrait_var_simple (n_var *n)
— void afficheArbreAbstrait_var_indicee (n_var *n)
```

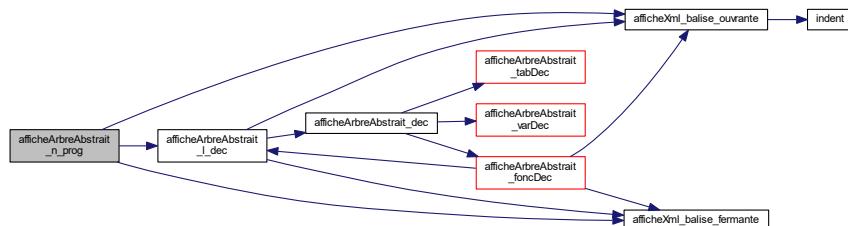
8.1.1 Documentation des fonctions

8.1.1.1 afficheArbreAbstrait_n_prog()

```
void afficheArbreAbstrait_n_prog (
    n_prog * n )
```

Références afficheArbreAbstrait_l_dec(), afficheXml_balise_ouverte(), et afficheXml_balise_ouvrante().

Voici le graphe d'appel pour cette fonction :



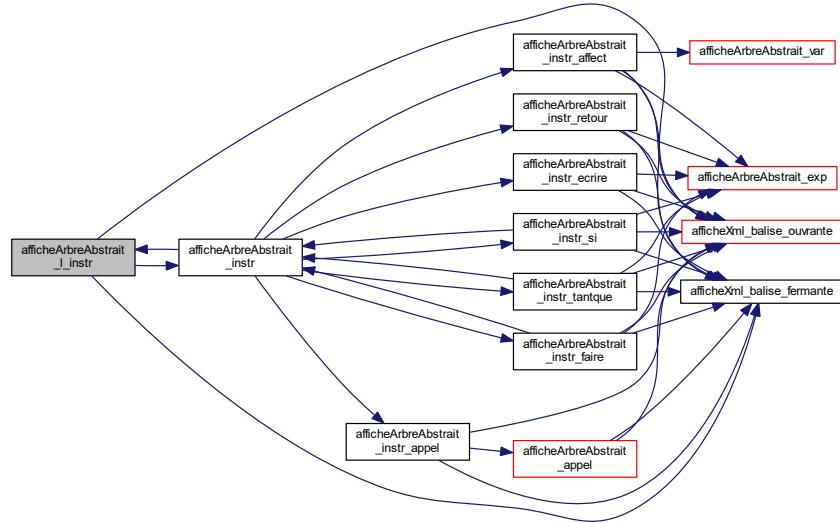
8.1.1.2 afficheArbreAbstrait_l_instr()

```
void afficheArbreAbstrait_l_instr (
    n_l_instr * n )
```

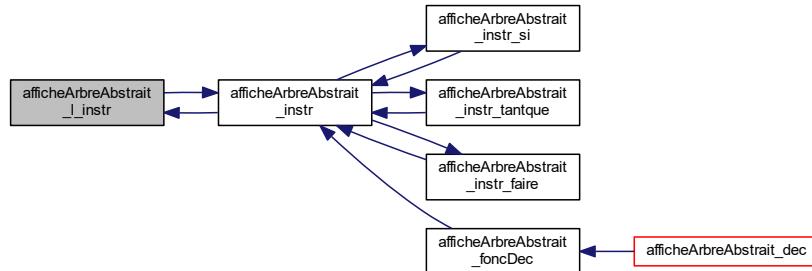
Références afficheArbreAbstrait_instr(), afficheXml_balise_ouverante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



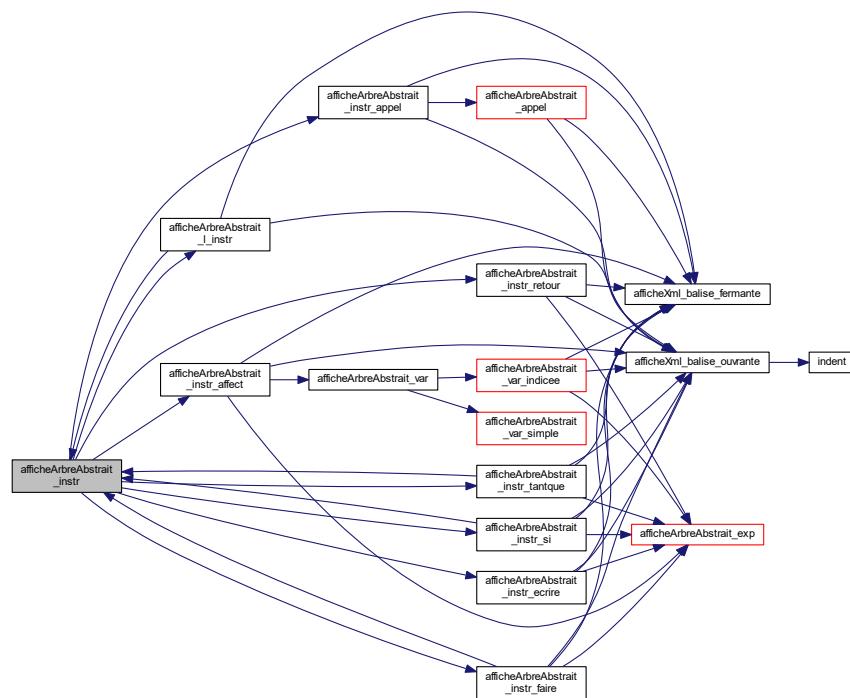
8.1.1.3 afficheArbreAbstrait_instr()

```
void afficheArbreAbstrait_instr (
    n_instr * n )
```

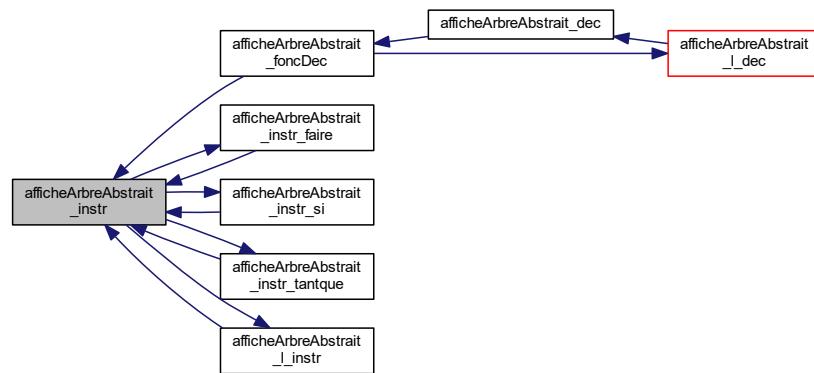
Références afficheArbreAbstrait_instr_affect(), afficheArbreAbstrait_instr_appel(), afficheArbreAbstrait_instr_ecrire(), afficheArbreAbstrait_instr_faire(), afficheArbreAbstrait_instr_retour(), afficheArbreAbstrait_instr_si(), afficheArbreAbstrait_instr_tantque(), et afficheArbreAbstrait_l_instr().

Référencé par afficheArbreAbstrait_foncDec(), afficheArbreAbstrait_instr_faire(), afficheArbreAbstrait_instr_si(), afficheArbreAbstrait_instr_tantque(), et afficheArbreAbstrait_l_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



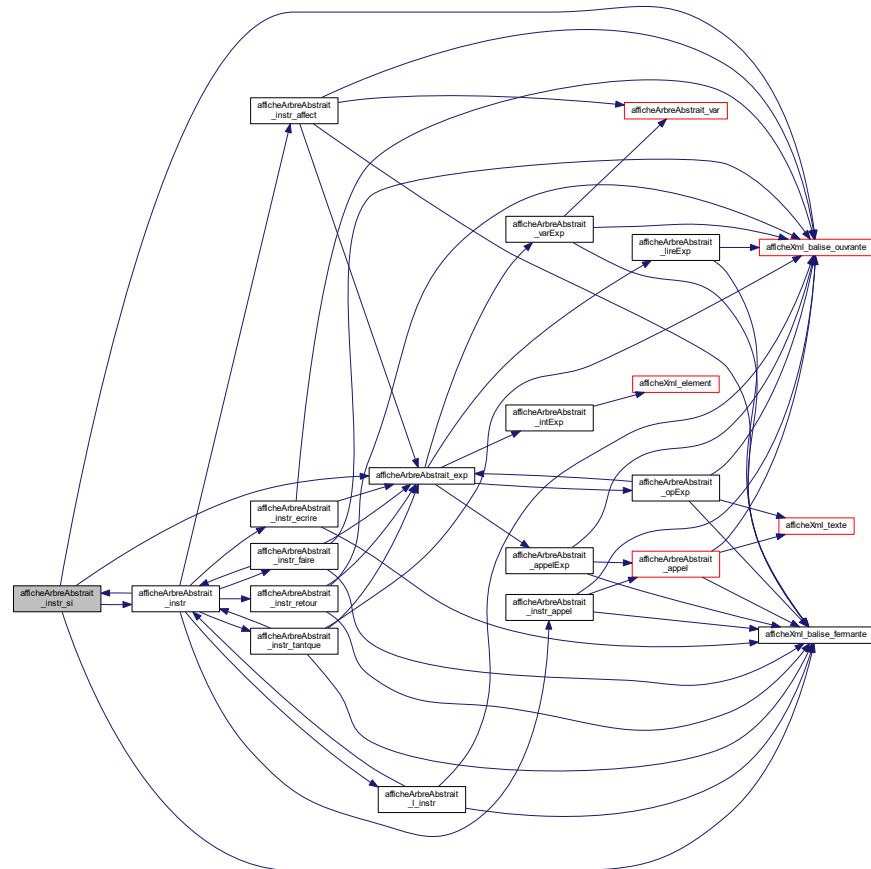
8.1.1.4 `afficheArbreAbstrait_instr_si()`

```
void afficheArbreAbstrait_instr_si (
    n_instr * n )
```

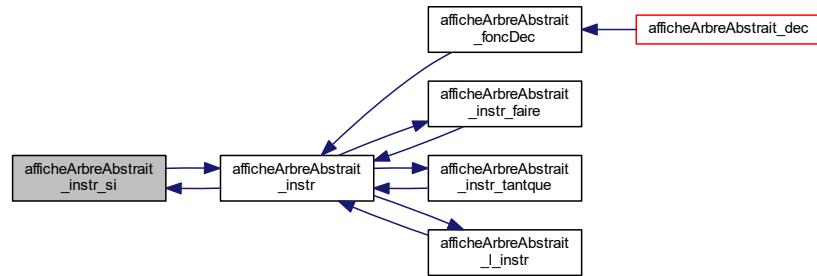
Références `afficheArbreAbstrait_exp()`, `afficheArbreAbstrait_instr()`, `afficheXml_balise_fermante()`, et `afficheXml_balise_ouvrante()`.

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



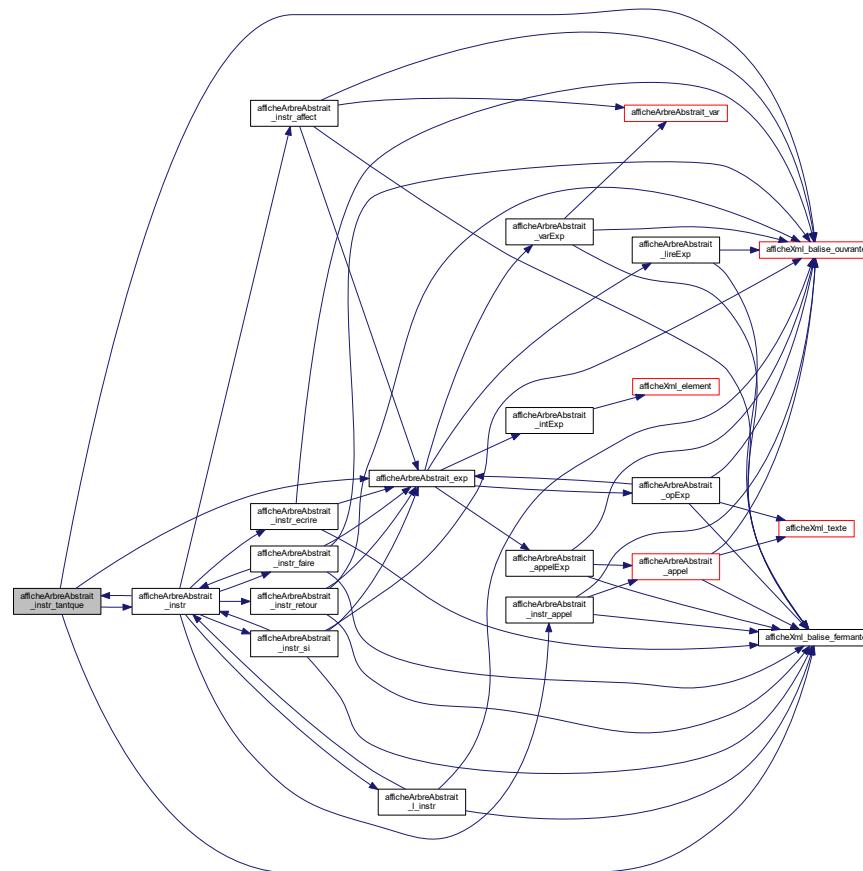
8.1.1.5 afficheArbreAbstrait instr tantque()

```
void afficheArbreAbstrait_instr_tantque (
```

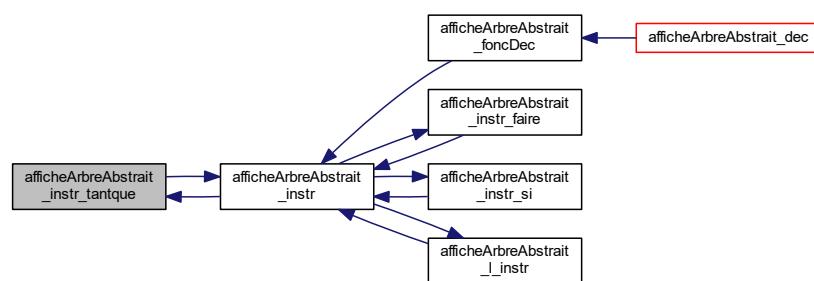
Références afficheArbreAbstrait_exp(), afficheArbreAbstrait_instr(), afficheXml_balise_fermante(), et afficheXml←_balise_ouvrante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



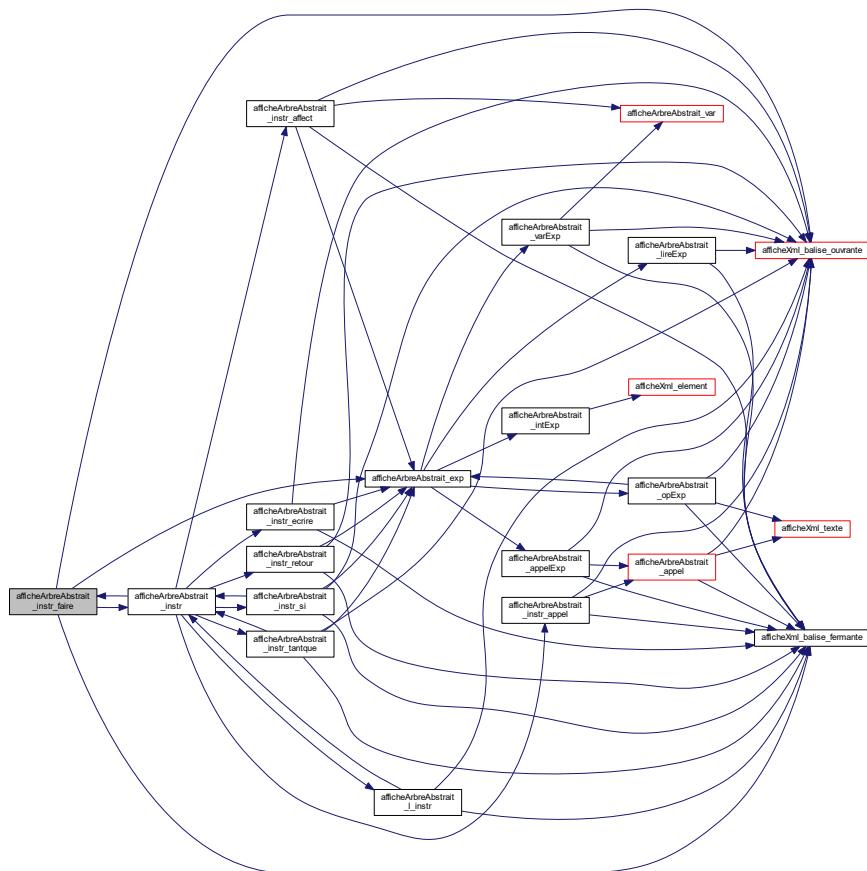
8.1.1.6 afficheArbreAbstrait_instr_faire()

```
void afficheArbreAbstrait_instr_faire (
    n_instr * n )
```

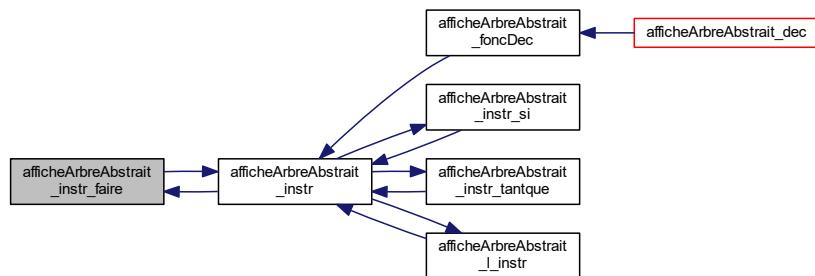
Références afficheArbreAbstrait_exp(), afficheArbreAbstrait_instr(), afficheXml_balise_fermante(), et afficheXml←_balise_ouvrante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



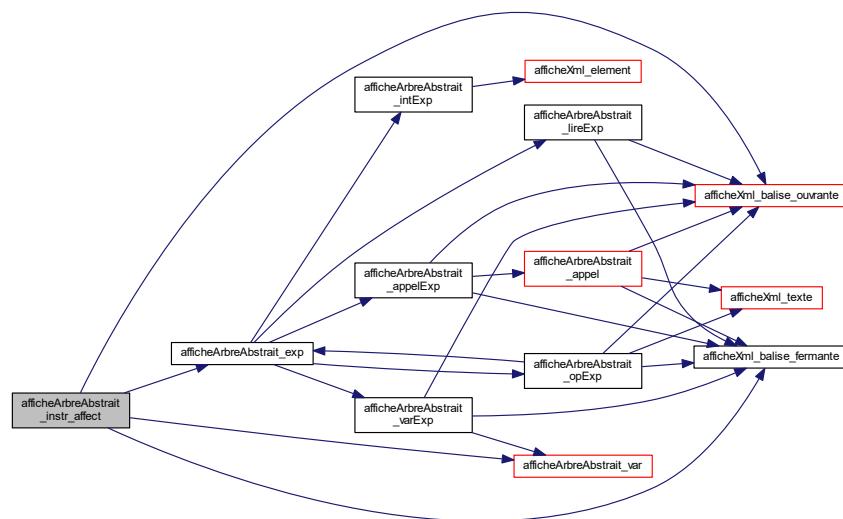
8.1.1.7 afficheArbreAbstrait_instr_affect()

```
void afficheArbreAbstrait_instr_affect (
    n_instr * n )
```

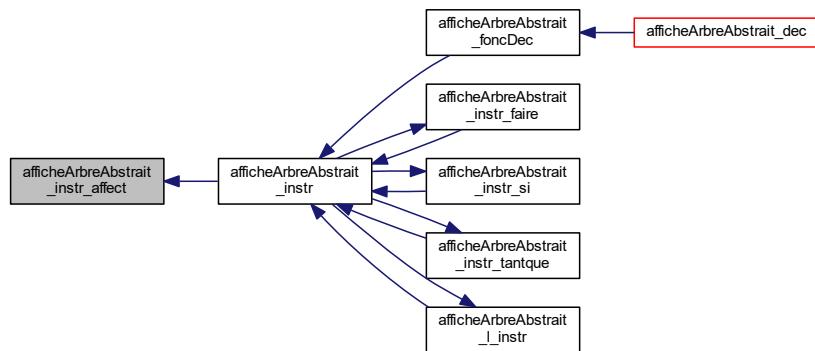
Références afficheArbreAbstrait_exp(), afficheArbreAbstrait_var(), afficheXml_balise_fermante(), et afficheXml_← balise_ouvante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



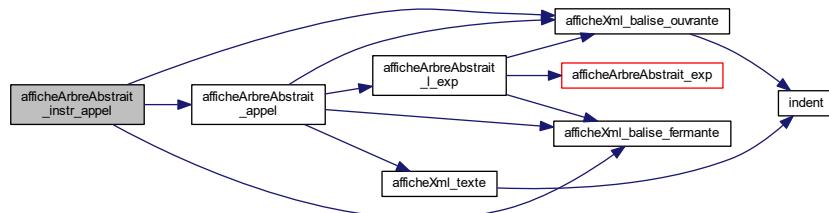
8.1.1.8 afficheArbreAbstrait_instr_appel()

```
void afficheArbreAbstrait_instr_appel (
    n_instr * n )
```

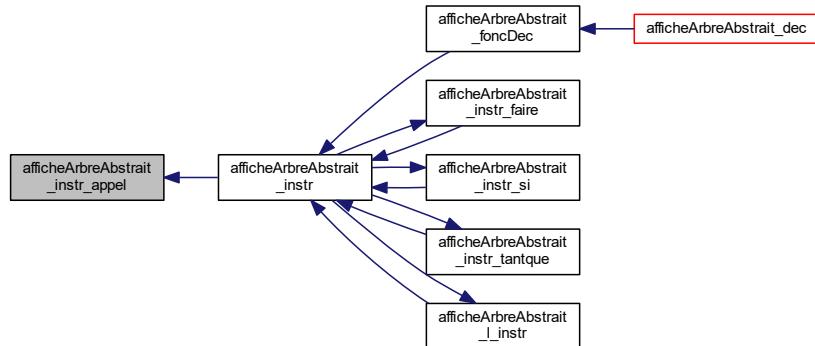
Références afficheArbreAbstrait_appel(), afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



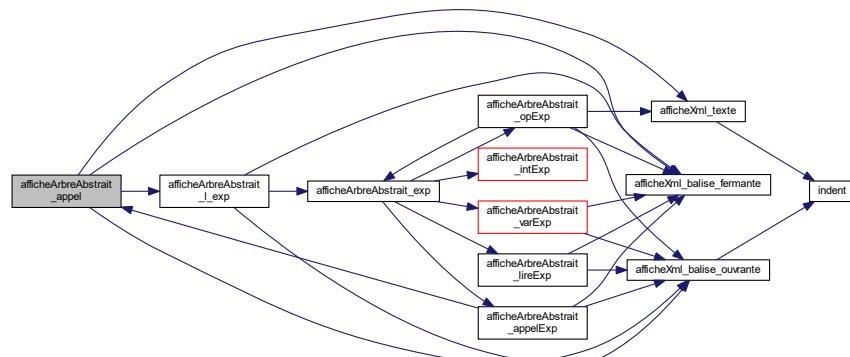
8.1.1.9 afficheArbreAbstrait_appel()

```
void afficheArbreAbstrait_appel (
    n_appel * n )
```

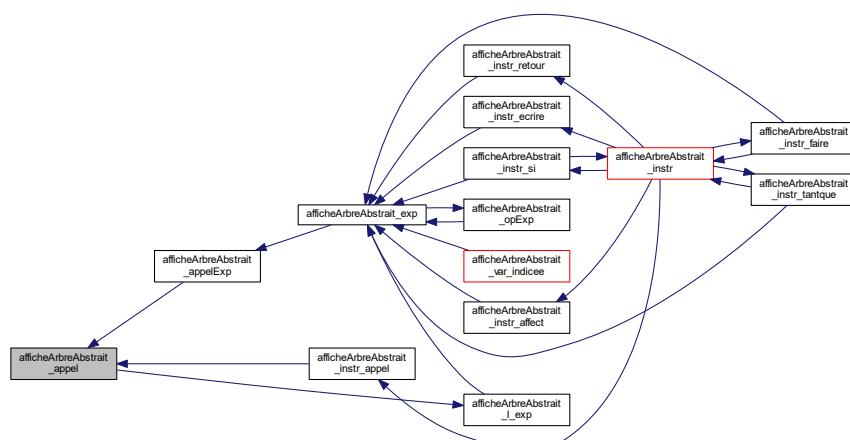
Références afficheArbreAbstrait_l_exp(), afficheXml_balise_fermante(), afficheXml_balise_ouvrante(), et afficheXml_texte().

Référencé par afficheArbreAbstrait_appelExp(), et afficheArbreAbstrait_instr_appel().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



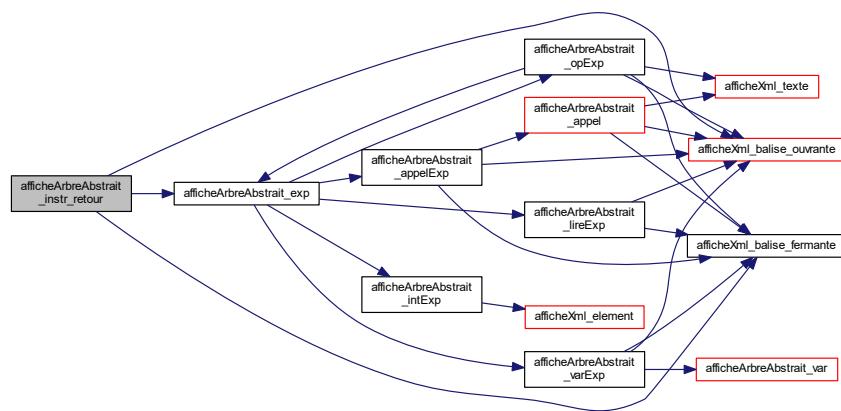
8.1.1.10 afficheArbreAbstrait_instr_retour()

```
void afficheArbreAbstrait_instr_retour (
    n_instr * n )
```

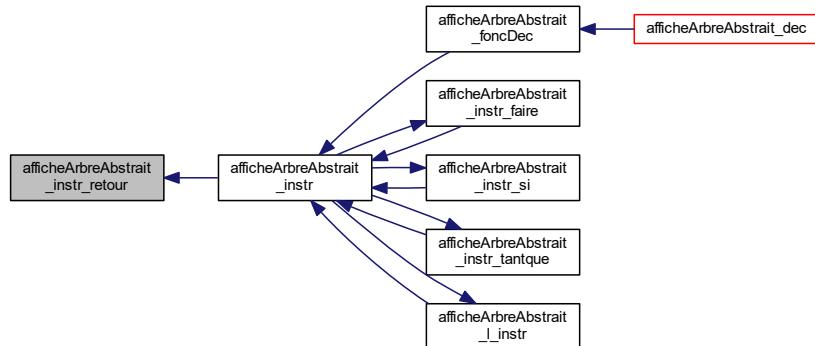
Références afficheArbreAbstrait_exp(), afficheXml_balise_fermante(), et afficheXml_balise_ouvante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



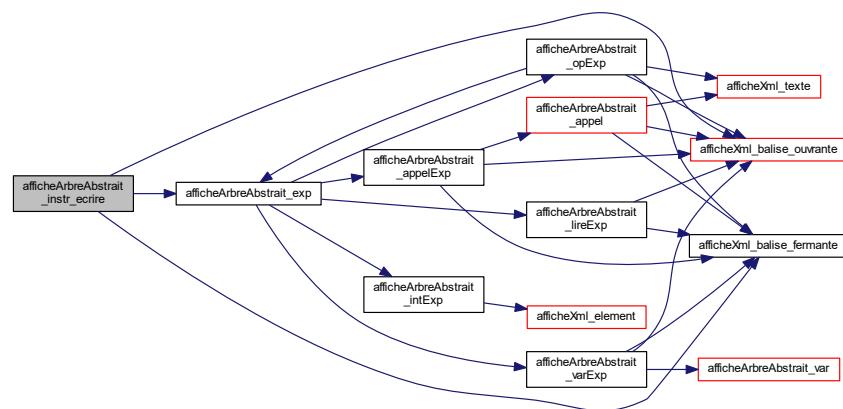
8.1.1.11 afficheArbreAbstrait_instr_ecrire()

```
void afficheArbreAbstrait_instr_ecrire (
    n_instr * n )
```

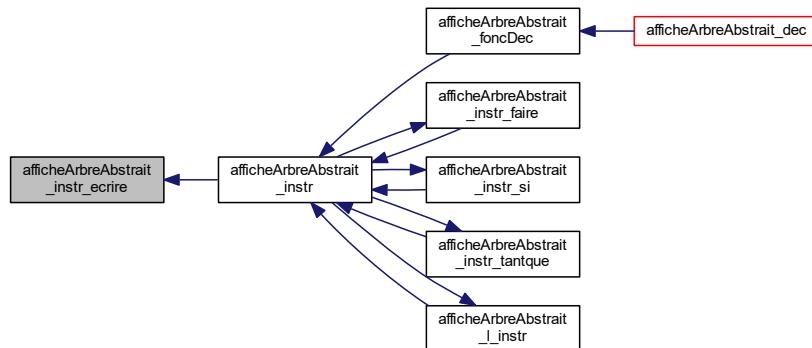
Références afficheArbreAbstrait_exp(), afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_instr().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



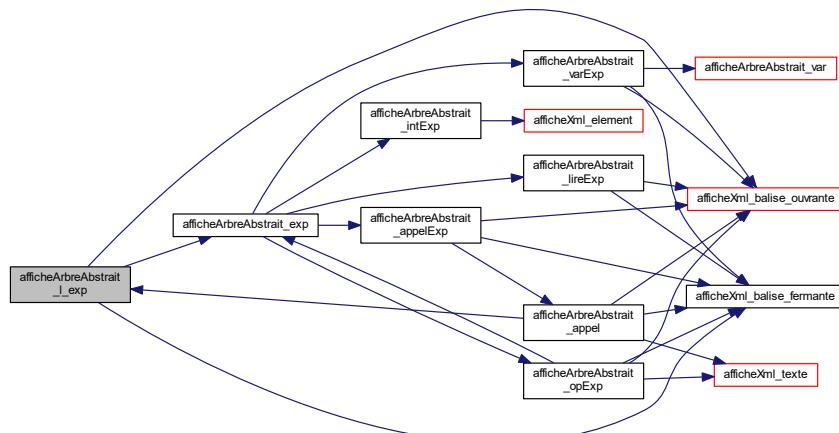
8.1.1.12 afficheArbreAbstrait_l_exp()

```
void afficheArbreAbstrait_l_exp (
    n_l_exp * n )
```

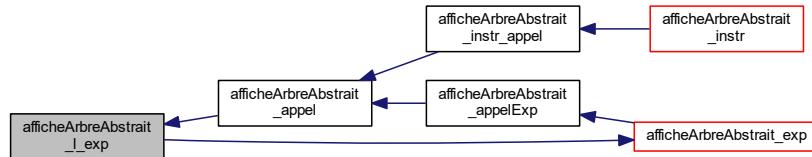
Références afficheArbreAbstrait_exp(), afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_appel().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



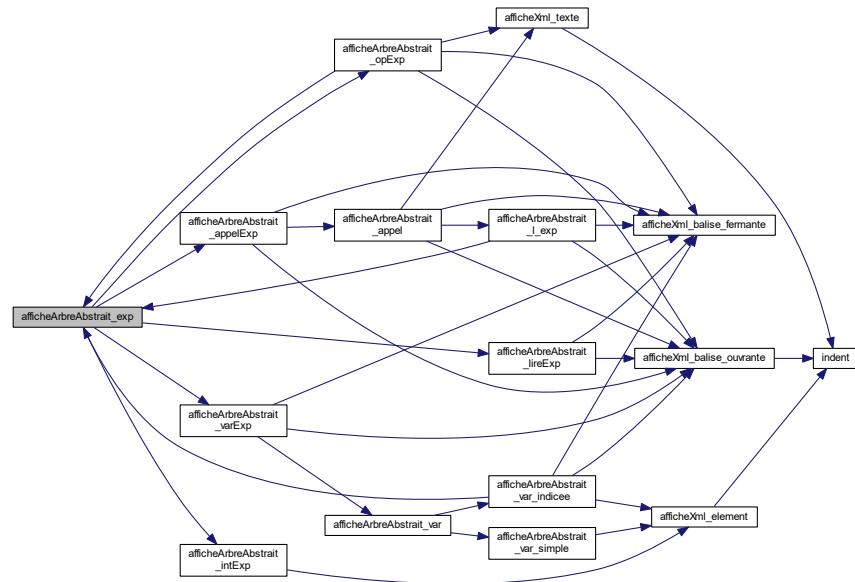
8.1.1.13 afficheArbreAbstrait_exp()

```
void afficheArbreAbstrait_exp (
    n_exp * n )
```

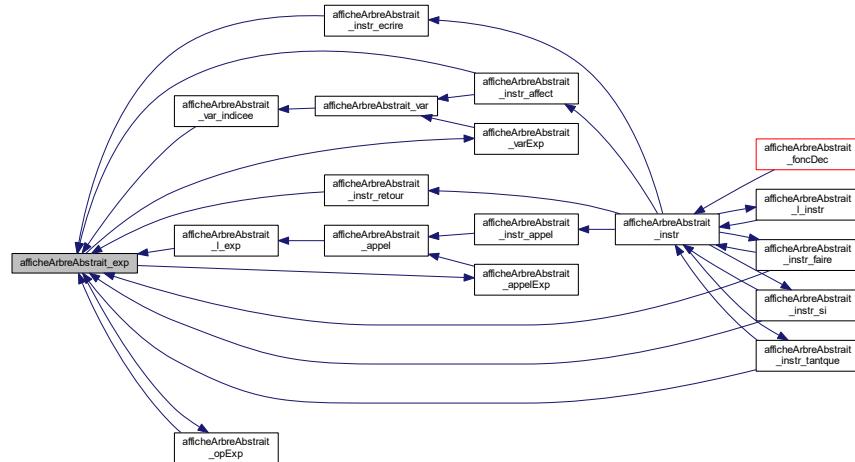
Références afficheArbreAbstrait_appeExp(), afficheArbreAbstrait_intExp(), afficheArbreAbstrait_lireExp(), afficheArbreAbstrait_opExp(), et afficheArbreAbstrait_varExp().

Référencé par afficheArbreAbstrait_instr_affect(), afficheArbreAbstrait_instr_ecrire(), afficheArbreAbstrait_instr_faire(), afficheArbreAbstrait_instr_retour(), afficheArbreAbstrait_instr_si(), afficheArbreAbstrait_instr_tantque(), afficheArbreAbstrait_l_exp(), afficheArbreAbstrait_opExp(), et afficheArbreAbstrait_var_indicee().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



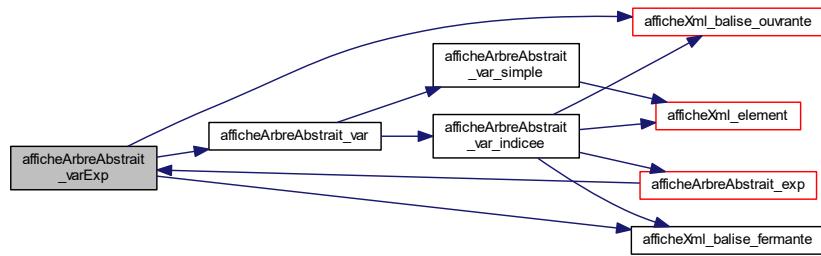
8.1.1.14 afficheArbreAbstrait_varExp()

```
void afficheArbreAbstrait_varExp (
```

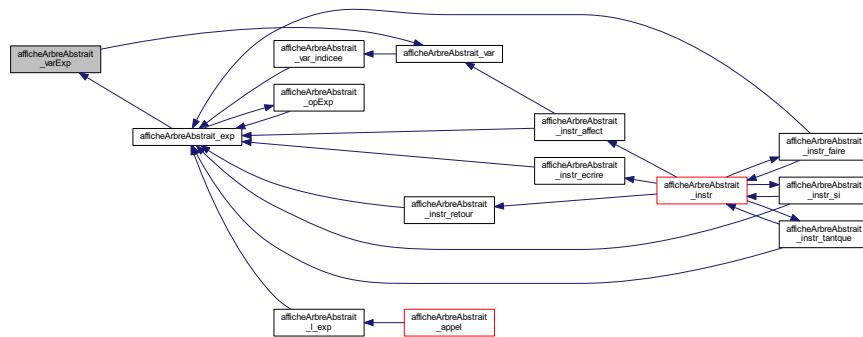
Références afficheArbreAbstrait_var(), afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



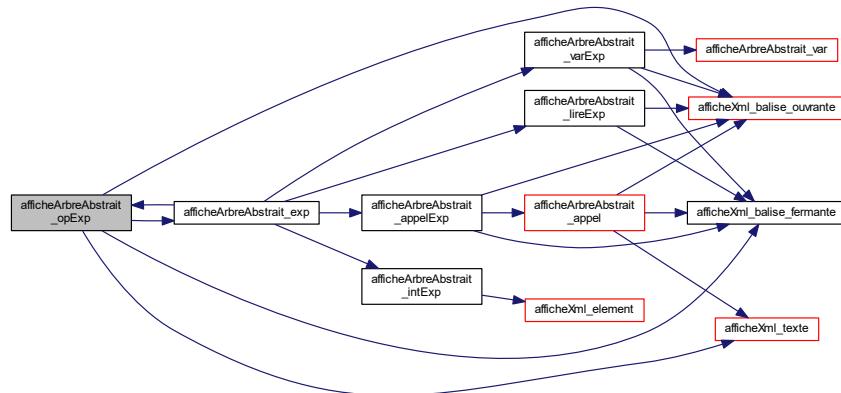
8.1.1.15 afficheArbreAbstrait_opExp()

```
void afficheArbreAbstrait_opExp (
    n_exp * n )
```

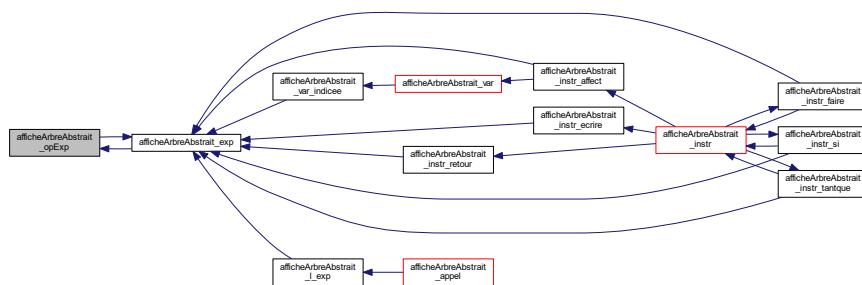
Références afficheArbreAbstrait_exp(), afficheXml_balise_fermante(), afficheXml_balise_ouvrante(), afficheXml←_texte(), diff, divise, égal, et, fois, inf, infeg, modulo, moins, non, ou, plus, sup, et supeg.

Référencé par afficheArbreAbstrait_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



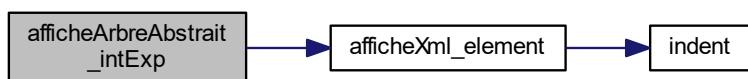
8.1.1.16 afficheArbreAbstrait_intExp()

```
void afficheArbreAbstrait_intExp (
    n_exp * n )
```

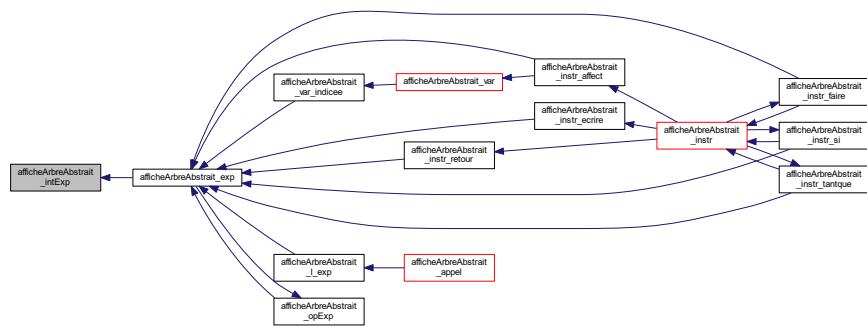
Références afficheXml_element().

Référencé par afficheArbreAbstrait_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



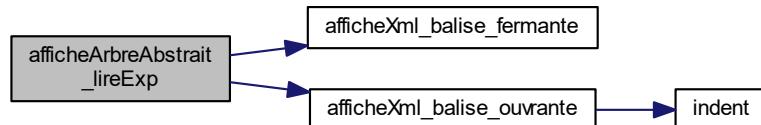
8.1.1.17 afficheArbreAbstrait_lireExp()

```
void afficheArbreAbstrait_lireExp (
    n_exp * n )
```

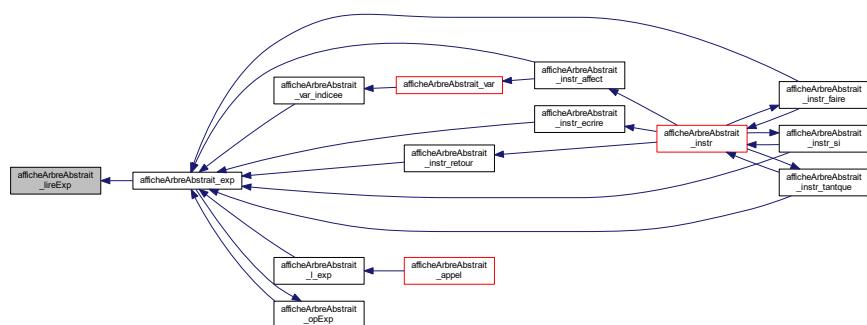
Références afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



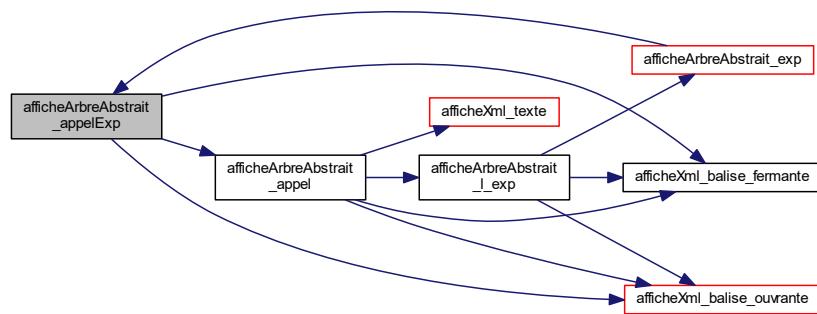
8.1.1.18 afficheArbreAbstrait_appelExp()

```
void afficheArbreAbstrait_appelExp (
    n_exp * n )
```

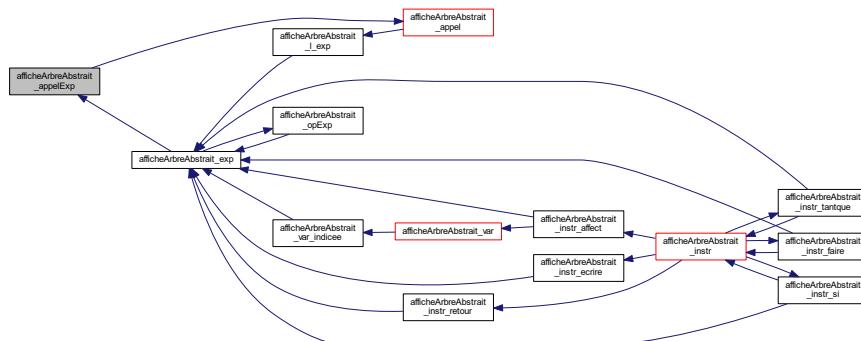
Références afficheArbreAbstrait_appel(), afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_exp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



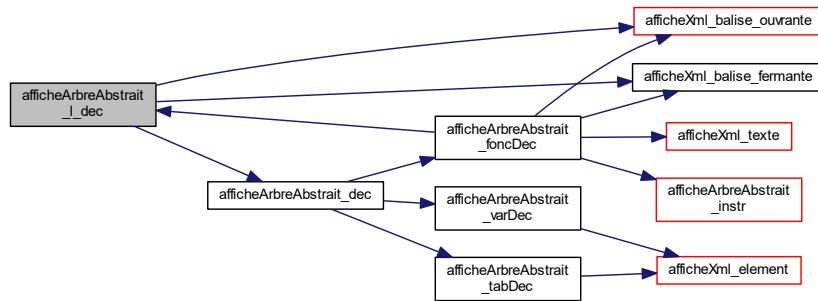
8.1.1.19 afficheArbreAbstrait_l_dec()

```
void afficheArbreAbstrait_l_dec (
    n_l_dec * n )
```

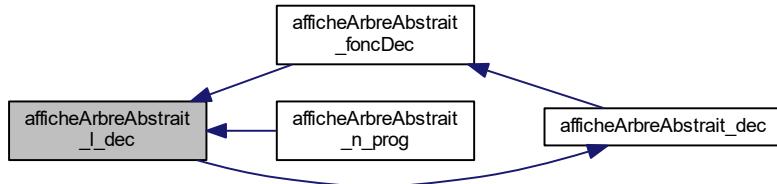
Références afficheArbreAbstrait_dec(), afficheXml_balise_fermante(), et afficheXml_balise_ouvrante().

Référencé par afficheArbreAbstrait_foncDec(), et afficheArbreAbstrait_n_prog().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



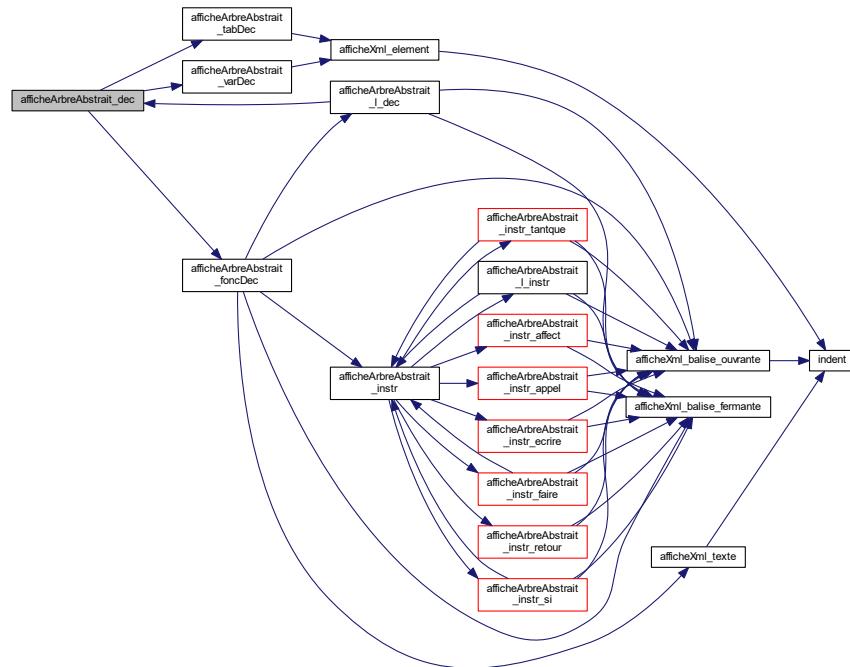
8.1.1.20 afficheArbreAbstrait_dec()

```
void afficheArbreAbstrait_dec (
    n_dec * n )
```

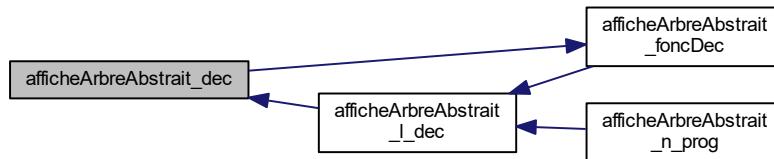
Références `afficheArbreAbstrait_foncDec()`, `afficheArbreAbstrait_tabDec()`, et `afficheArbreAbstrait_varDec()`.

Référencé par `afficheArbreAbstrait_l_dec()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



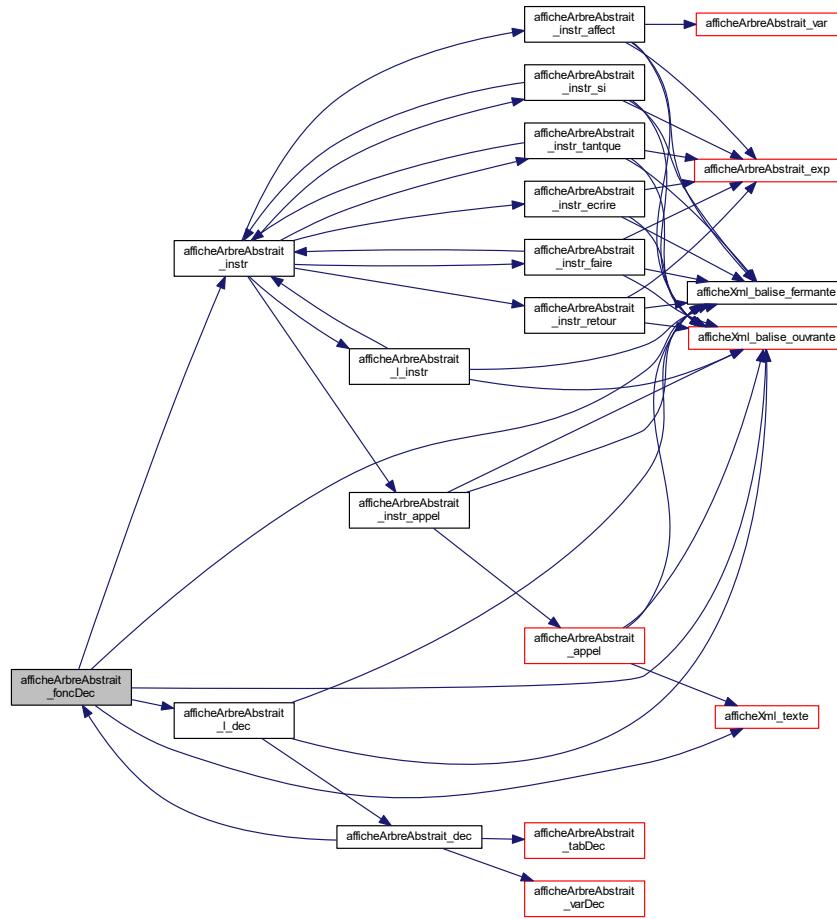
8.1.1.21 afficheArbreAbstrait_foncDec()

```
void afficheArbreAbstrait_foncDec (
    n_dec * n )
```

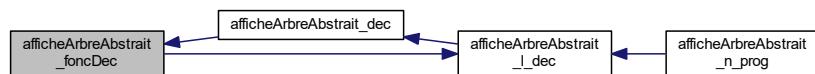
Références afficheArbreAbstrait_instr(), afficheArbreAbstrait_l_dec(), afficheXml_balise_fermante(), afficheXml_balise_ouvante(), et afficheXml_texte().

Référencé par afficheArbreAbstrait_dec().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



8.1.1.22 afficheArbreAbstrait_varDec()

```
void afficheArbreAbstrait_varDec (
    n_dec * n )
```

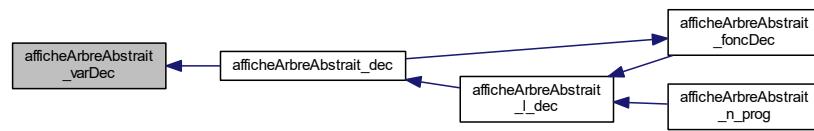
Références `afficheXml_element()`.

Référencé par `afficheArbreAbstrait_dec()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



8.1.1.23 afficheArbreAbstrait_tabDec()

```
void afficheArbreAbstrait_tabDec (
    n_dec * n )
```

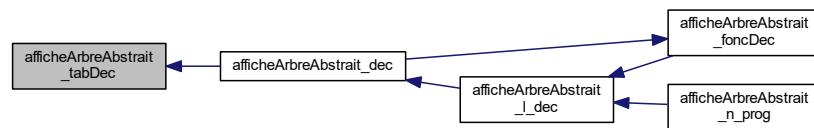
Références afficheXml_element().

Référencé par afficheArbreAbstrait_dec().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



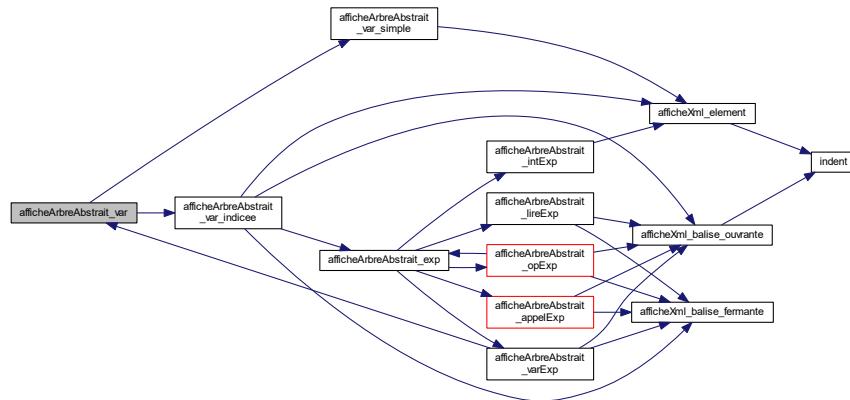
8.1.1.24 afficheArbreAbstrait_var()

```
void afficheArbreAbstrait_var (
    n_var * n )
```

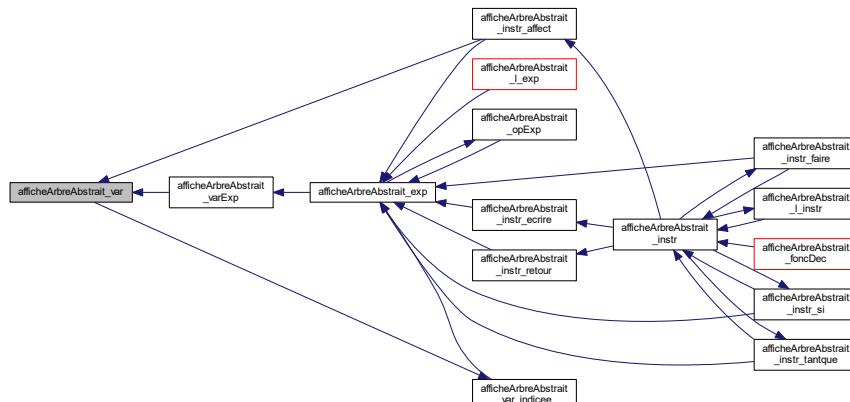
Références afficheArbreAbstrait_var_indicee(), et afficheArbreAbstrait_var_simple().

Référencé par afficheArbreAbstrait_instr_affect(), et afficheArbreAbstrait_varExp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



8.1.1.25 afficheArbreAbstrait_var_simple()

```
void afficheArbreAbstrait_var_simple (
    n_var * n )
```

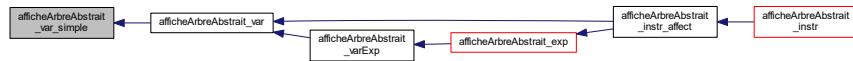
Références afficheXml_element().

Référencé par afficheArbreAbstrait_var().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



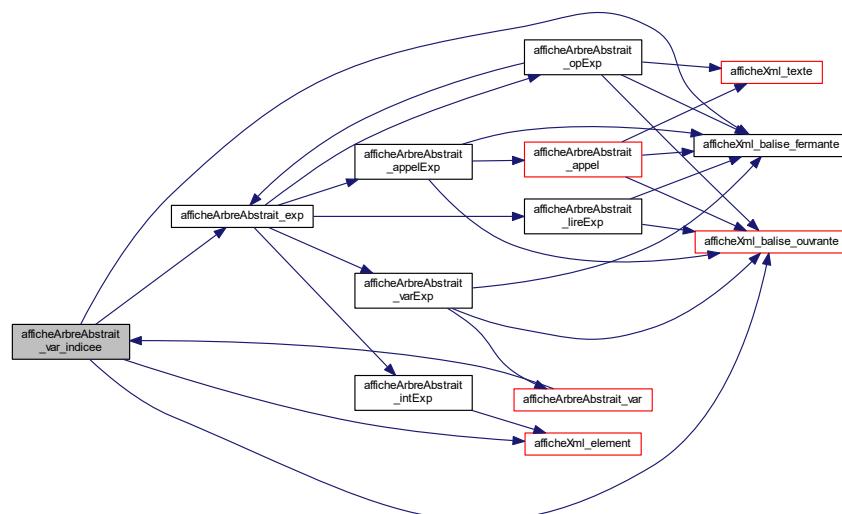
8.1.1.26 afficheArbreAbstrait_var_indicee()

```
void afficheArbreAbstrait_var_indicee (
    n_var * n )
```

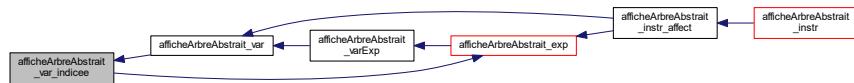
Références afficheArbreAbstrait_exp(), afficheXml_balise_fermante(), afficheXml_balise_ouvrante(), et afficheXml_element().

Référencé par afficheArbreAbstrait_var().

Voici le graphe d'appel pour cette fonction :



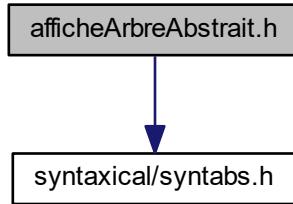
Voici le graphe des appelants de cette fonction :



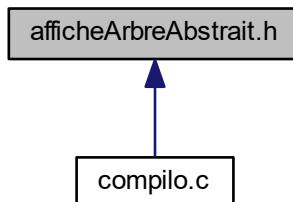
8.2 Référence du fichier afficheArbreAbstrait.h

#include "syntaxical/syntabs.h"

Graphe des dépendances par inclusion de afficheArbreAbstrait.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :

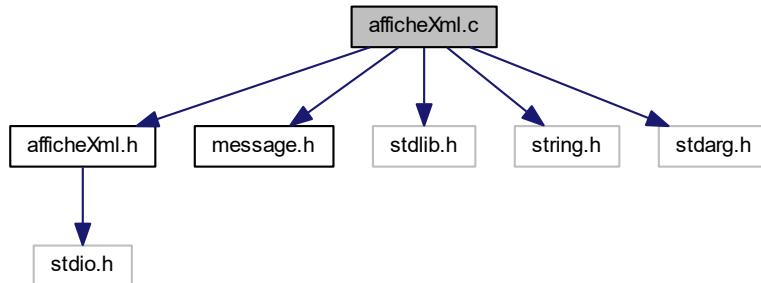


8.3 Référence du fichier afficheXml.c

```

#include "afficheXml.h"
#include "message.h"
#include <stdlib.h>
#include <string.h>
  
```

```
#include <stdarg.h>
Graphe des dépendances par inclusion de afficheXml.c :
```



Macros

- #define `ErreursSiAucuneSortie()` if(sortie_xml_file==NULL) { erreurArgs("Sortie XML nécessaire !"); }

Fonctions

- void `afficheXml_setOutput` (FILE *file)
- void `indent` ()
- void `afficheXml_balise_ouvante` (const char *fct_)
- void `afficheXml_balise_fermante` (const char *fct_)
- void `afficheXml_texte` (char *texte_)
- void `special_texte` (char *texte_)
- void `afficheXml_element` (char *fct_, char *texte_)

8.3.1 Documentation des macros

8.3.1.1 ErreursSiAucuneSortie

```
#define ErreursSiAucuneSortie( ) if(sortie_xml_file==NULL) { erreurArgs("Sortie XML nécessaire !"); }
```

Référencé par `afficheXml_balise_fermante()`, `afficheXml_balise_ouvante()`, `afficheXml_element()`, et `afficheXml_texte()`.

8.3.2 Documentation des fonctions

8.3.2.1 afficheXml_setOutput()

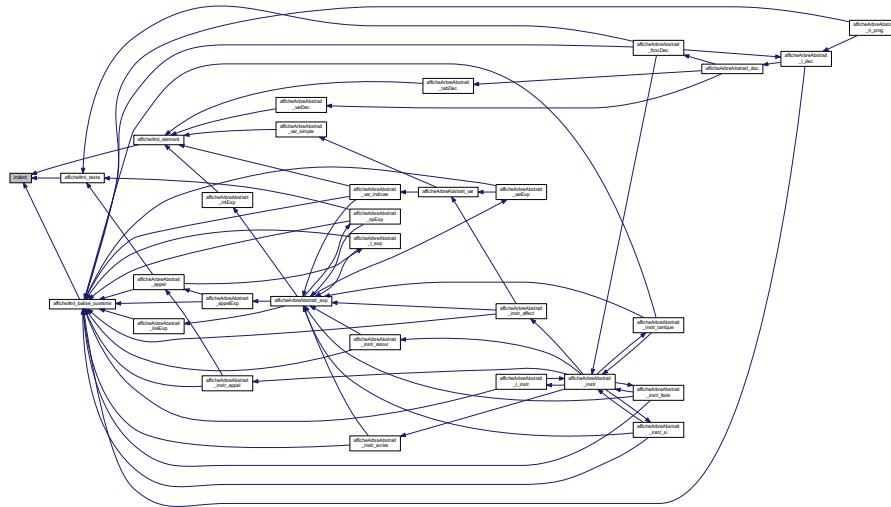
```
void afficheXml_setOutput (
    FILE * file )
```

8.3.2.2 indent()

```
void indent ( )
```

Référencé par afficheXml_balise_ouvrante(), afficheXml_element(), et afficheXml_texte().

Voici le graphe des appels de cette fonction :



8.3.2.3 afficheXml_balise_ouvrante()

```
void afficheXml_balise_ouvrante (
    const char * fct_ )
```

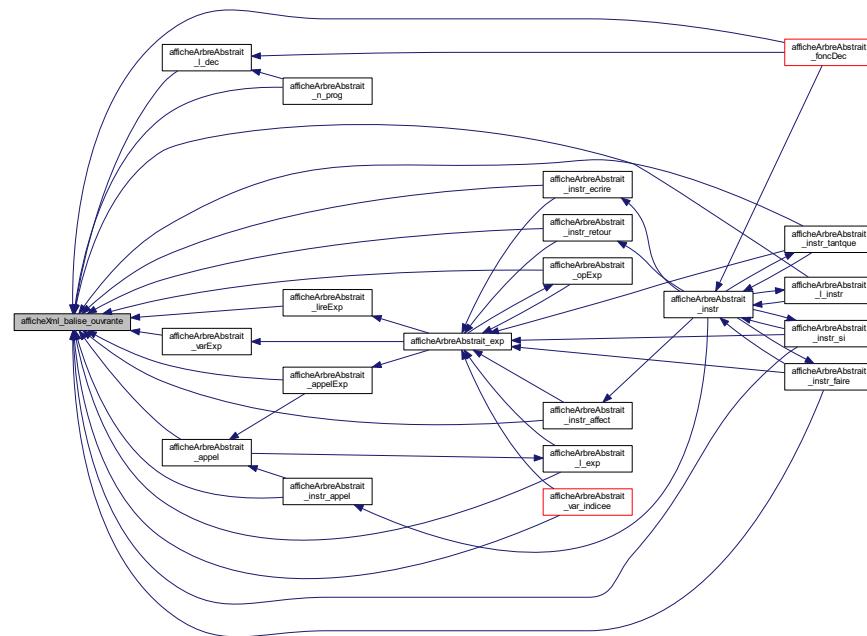
Références ErreurSiAucuneSortie, et indent().

Référencé par afficheArbreAbstrait_appel(), afficheArbreAbstrait_appelExp(), afficheArbreAbstrait_foncDec(), afficheArbreAbstrait_instr_affect(), afficheArbreAbstrait_instr_appel(), afficheArbreAbstrait_instr_ecrire(), afficheArbreAbstrait_instr faire(), afficheArbreAbstrait_instr_retour(), afficheArbreAbstrait_instr_si(), afficheArbreAbstrait_instr_tantque(), afficheArbreAbstrait_l_dec(), afficheArbreAbstrait_l_exp(), afficheArbreAbstrait_l_instr(), afficheArbreAbstrait_lireExp(), afficheArbreAbstrait_n_prog(), afficheArbreAbstrait_opExp(), afficheArbreAbstrait_var_indicee(), et afficheArbreAbstrait_varExp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



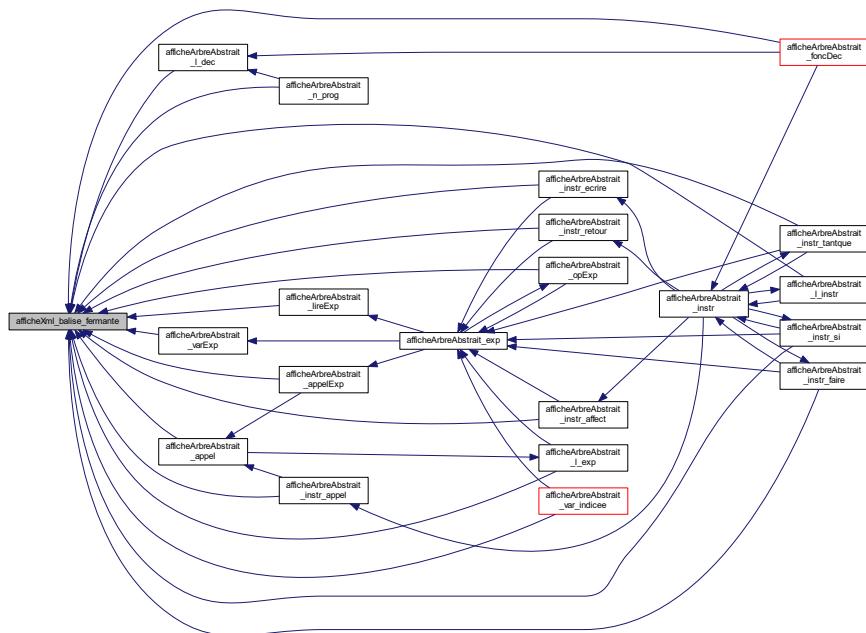
8.3.2.4 afficheXml_balise_fermante()

```
void afficheXml_balise_fermante (
    const char * fct_ )
```

Références ErreurSiAucuneSortie.

Référencé par afficheArbreAbstrait_appel(), afficheArbreAbstrait_appelExp(), afficheArbreAbstrait_foncDec(), afficheArbreAbstrait_instr_affect(), afficheArbreAbstrait_instr_appel(), afficheArbreAbstrait_instr_ecrire(), afficheArbreAbstrait_instr_faire(), afficheArbreAbstrait_instr_retour(), afficheArbreAbstrait_instr_si(), afficheArbreAbstrait_instr_tantque(), afficheArbreAbstrait_l_dec(), afficheArbreAbstrait_l_exp(), afficheArbreAbstrait_l_instr(), afficheArbreAbstrait_lireExp(), afficheArbreAbstrait_n_prog(), afficheArbreAbstrait_opExp(), afficheArbreAbstrait_var_indicee(), et afficheArbreAbstrait_varExp().

Voici le graphe des appelants de cette fonction :



8.3.2.5 afficheXml_texte()

```
void afficheXml_texte (
    char * texte_ )
```

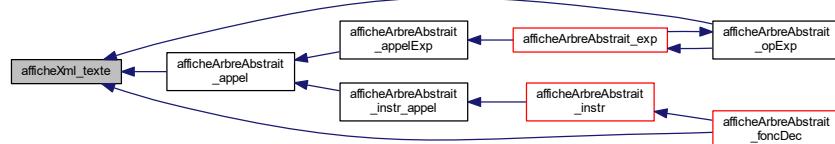
Références ErreurSiAucuneSortie, et indent().

Référencé par afficheArbreAbstrait_appel(), afficheArbreAbstrait_foncDec(), et afficheArbreAbstrait_opExp().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



8.3.2.6 special_texte()

```
void special_texte (
    char * texte_ )
```

8.3.2.7 afficheXml_element()

```
void afficheXml_element (
    char * fct_,
    char * texte_ )
```

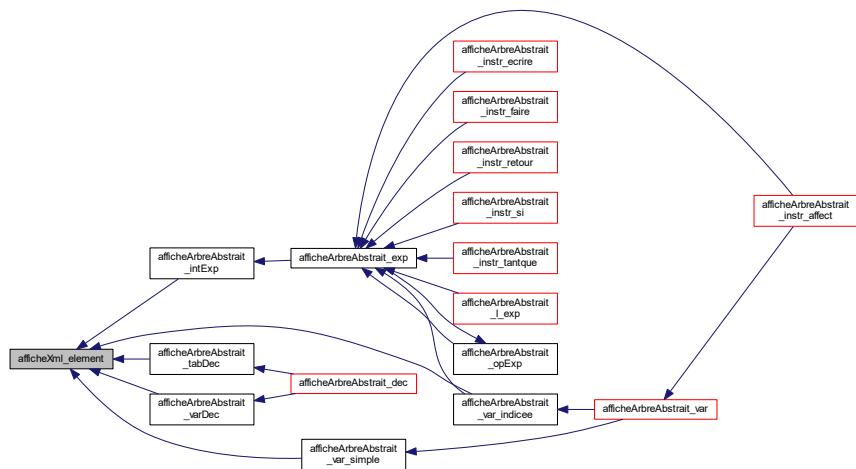
Références ErreurSiAucuneSortie, et indent().

Référencé par afficheArbreAbstrait_intExp(), afficheArbreAbstrait_tabDec(), afficheArbreAbstrait_var_indicee(), afficheArbreAbstrait_var_simple(), et afficheArbreAbstrait_varDec().

Voici le graphe d'appel pour cette fonction :



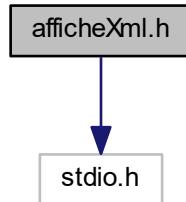
Voici le graphe des appels de cette fonction :



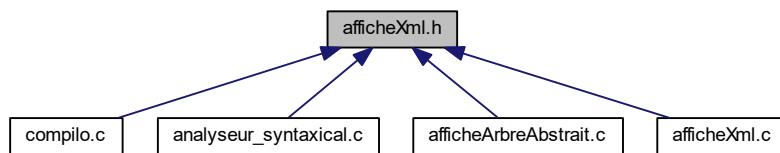
8.4 Référence du fichier afficheXml.h

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

Graphe des dépendances par inclusion de afficheXml.h :



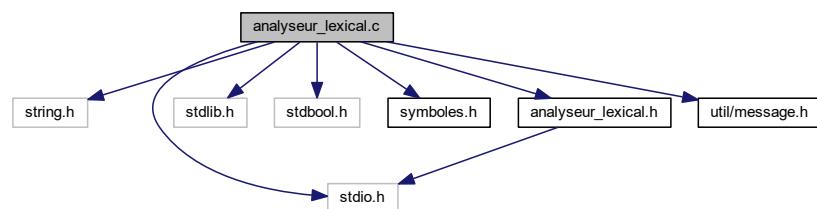
Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.5 Référence du fichier analyseur_lexical.c

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "symboles.h"
#include "analyseur_lexical.h"
#include "util/message.h"
```

Graphe des dépendances par inclusion de analyseur_lexical.c :



Macros

```
— #define YYTEXT_MAX 100
— #define is_num(c) ('0' <= (c)) && ((c) <= '9')
— #define is_maj(c) ('A' <= (c)) && ((c) <= 'Z')
— #define is_min(c) ('a' <= (c)) && ((c) <= 'z')
— #define is_alpha(c) (is_maj(c) || is_min(c) || (c) == ' ' || (c) == '$')
— #define is_alpha_simple(c) (is_maj(c) || is_min(c) || (c) == '_')
— #define is_alphanum(c) (is_num((c)) || is_alpha((c)))
— #define MotsClefs_Operators_1Carac_SIZE 15
— #define MotsClefs_Operators_SIZE 7
— #define MotsClefs_Instruction_SIZE 8
— #define MotsClefs_Fonction_SIZE 2
— #define ELIF_TEST_TOKEN(tok) else if( token == tok ) strcpy( valeur, #tok );
```

Fonctions

```
— void erreurLexical (const char *typeErreur)
— int mangeEspaces ()
    mangeEspaces : Fonction qui ignore les espaces et commentaires.
— char lireCar (void)
    lireCar : Lit un caractère et le stocke dans le buffer yytext
— void delireCar ()
    delireCar : Remet le dernier caractère lu au buffer clavier et enlève du buffer yytext
— void initBuff ()
— bool estUneVariable ()
    estUneVariable : test si la valeur courante correspond à un nom de variable
— bool estUneFonction ()
    estUneFonction : test si la valeur courante correspond à un nom de fonction
— bool estUnNombre ()
    estUnNombre : test si la valeur courante est nombre
— int yylex (void)
    yylex : Fonction principale de l'analyseur lexical, lit les caractères de yyin et renvoie les tokens sous forme d'entier.
    Le code de chaque unité est défini dans symboles.h sinon (mot clé, identifiant, etc.). Pour les tokens de type
    ID_FCT, ID_VAR et NOMBRE la valeur du token est dans yytext, visible dans l'analyseur syntaxique.
— void nom_token (int token, char *nom, char *valeur)
    nom_token : Fonction auxiliaire appelée par l'analyseur syntaxique tout simplement pour afficher des messages
    d'erreur et l'arbre XML
— void afficherLexique (FILE *yyin, FILE *output)
    afficherLexique : Fonction auxiliaire appelée par le compilo en mode -l, pour tester l'analyseur lexical et, étant donné
    un programme en entrée, afficher la liste des tokens.
```

Variables

```
— char yytext [100]
— int nb_ligne = 1
```

8.5.1 Documentation des macros

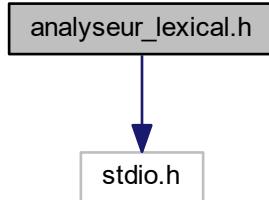
8.5.1.1 ELIF_TEST_TOKEN

```
#define ELIF_TEST_TOKEN(
    tok ) else if( token == tok ) strcpy( valeur, #tok );
```

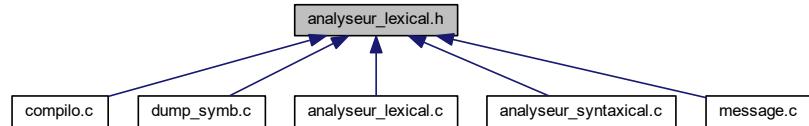
8.6 Référence du fichier analyseur_lexical.h

```
#include "stdio.h"
```

Graphe des dépendances par inclusion de analyseur_lexical.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



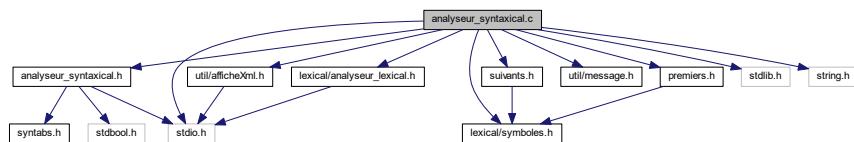
Variables

- char ***yytext*** [100]
- FILE * ***yyin***
- int ***nb_ligne***

8.7 Référence du fichier analyseur_syntactical.c

```
#include "analyseur_syntactical.h"
#include "premiers.h"
#include "suivants.h"
#include "lexical/symboles.h"
#include "lexical/analyseur_lexical.h"
#include "util/message.h"
#include "util/afficheXml.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
Graphe des dépendances par inclusion de analyseur_syntactical.c :
```



Macros

- #define Entre
- #define Sortie(retData)
 - nécessite une variable préalablement définie 'retData'*
- #define Erreur

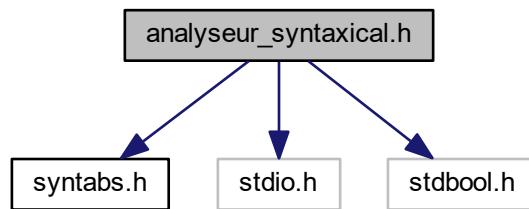
Fonctions

- void uniteCouranteSuivante ()
- void erreurSyntactical (const char *fonctionName)
- n_prog * programme ()
- n_l_dec * optDecVariables ()
- n_l_dec * listeDecVariables ()
- n_l_dec * listeDecVariablesBis ()
- n_dec * declarationVariable ()
- n_dec * optTailleTableau (char *nomVariable)
- n_l_dec * listeDecFonctions ()
- n_dec * declarationFonction ()
- n_l_dec * listeParam ()
- n_l_dec * optListeDecVariables ()
- n_instr * instruction ()
- n_instr * instructionAffect ()
- n_instr * instructionBloc ()
- n_l_instr * listeInstructions ()
- n_instr * instructionSi ()
- n_instr * optSinon ()
- n_instr * instructionTantque ()
- n_instr * instructionFaire ()
- n_instr * instructionAppel ()
- n_instr * instructionRetour ()
- n_instr * instructionEcriture ()
- n_instr * instructionVide ()
- n_instr * instructionIncrementer ()
- n_exp * expression ()
- n_exp * expressionBis (n_exp *op1)
- n_exp * conjonction ()
- n_exp * conjonctionBis (n_exp *op1)
- n_exp * comparaison ()
- n_exp * comparaisonBis (n_exp *op1)
- n_exp * expArith ()
- n_exp * expArithBis (n_exp *op1)
- n_exp * terme ()
- n_exp * termeBis (n_exp *op1)
- n_exp * negation ()
- n_exp * facteur ()
- n_var * var ()
- n_var * optIndice (char *nomVariable)
- n_appel * appelFct ()
- n_l_exp * listeExpressions ()
- n_l_exp * listeExpressionsBis ()
- void syntactical_setAffiche (bool b)
 - syntacticalAffiche : spécifier si l'arbre syntaxique doit s'afficher*
- n_prog * syntactical ()
 - syntactical : appel du parseur syntaxique*

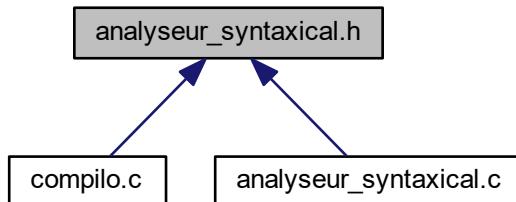
8.8 Référence du fichier analyseur_syntactical.h

```
#include "syntabs.h"
#include <stdio.h>
#include <stdbool.h>
```

Graphe des dépendances par inclusion de analyseur_syntactical.h :



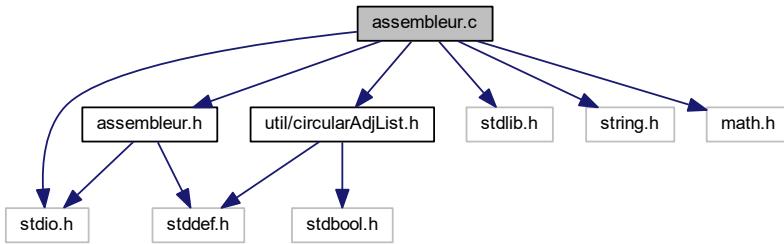
Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.9 Référence du fichier assembleur.c

```
#include "assembleur.h"
#include "util/circularAdjList.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
```

Graphe des dépendances par inclusion de assembleur.c :



Structures de données

- struct `assembleur_data`
`assembleur_data` : donnée pour section data
- struct `assembleur_bss`
`assembleur_bss` : donnée pour section bss
- struct `assembleur_entry`
`assembleur_entry` : donnée pour une entrée

Macros

- #define `SECURE_MALLOC_STRUCT`(dataType, varName)
- #define `SIMPLE_ERROR`(errStr) { fprintf(stderr, "Erreur Assembleur : %s \n", errStr); exit(EXIT_FAILURE); }
- #define `printSpace`(space, out) for (int i = 0 ; i < space ; i++) {fputs(" ", out) ;}
- #define `CHECK_ENTRY`() if(entry==NULL) { fprintf(stderr,"entry ne doit pas être nul\n"); exit(EXIT_FAILURE); }
- #define `SECURE_MALLOC_INST_STR`(size)
- #define `SECURE_INST_STR`(size, pattern, args...)
- #define `ADD_ENTRY`(newStr) entry->instrs = `circularAdjList_add`(struct `assembleur_entry_instr_circularAdjList`, entry->instrs, newStr, entry_instr_circularAdjList_copier, NULL);
- #define `GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE` 0

Définitions de type

- typedef struct `assembleur_entry_instr_circularAdjList` `assembleur_entry_instr_circularAdjList`

Fonctions

- void `assembleur_add_commentary` (const char *commentary)
`assembleur_entry_instr_setCommentary` : ajouter un commentaire pour la prochaine instruction ajouté (si elle existe)
- const char * `assembleur_commentary_next` ()
- `circularAdjList_struct_header` (`assembleur_data_circularAdjList`)
`assembleur_data_circularAdjList` Liste d'adjacence pour section data
- `circularAdjListCopier_header` (`data_circularAdjList_copier`)
`assembleur_data_circularAdjList` Fonction de copie pour `assembleur_data_circularAdjList`
- `circularAdjListComparator_header` (`data_circularAdjList_comparator`)
`data_circularAdjList_comparator` Fonction de comparaison pour `assembleur_data_circularAdjList`
- `circularAdjListStaticDecl_struct` (`assembleur_data_circularAdjList`, `data_circularAdjList`)
`data_circularAdjList` Déclaration de la liste d'adjacence pour section data

- `circularAdjListCopier_header` (`bss_circularAdjList_copier`)
 - bss_circularAdjList_copier Fonction de copie pour assembleur_bss_circularAdjList*
- `circularAdjListComparator_header` (`bss_circularAdjList_comparator`)
 - bss_circularAdjList_comparator Fonction de comparaison pour assembleur_bss_circularAdjList*
- `circularAdjListStaticDecl_struct` (`assembleur_bss_circularAdjList, bss_circularAdjList`)
 - bss_circularAdjList Déclaration de la liste d'adjacence pour section bss*
- `circularAdjListCopier_header` (`entry_circularAdjList_copier`)
 - entry_circularAdjList_copier Fonction de copie pour assembleur_entry_circularAdjList*
- `circularAdjListComparator_header` (`entry_circularAdjList_comparator`)
 - entry_circularAdjList_comparator Fonction de comparaison pour assembleur_entry_circularAdjList*
- `circularAdjListStaticDecl_struct` (`assembleur_entry_circularAdjList, entry_circularAdjList`)
 - entry_circularAdjList Déclaration de la liste d'adjacence pour section text*
- `circularAdjListStruct_header` (`assembleur_entry_instr_circularAdjList`)
 - assembleur_entry_instr_circularAdjList Liste d'adjacence pour les instructions/sous-entrées/points-entrées dans une d'une entrée*
- `circularAdjListCopier_header` (`entry_instr_circularAdjList_copier`)
 - entry_instr_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList*
- `circularAdjListCopier_header` (`entry_subEntry_circularAdjList_copier`)
 - entry_subEntry_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList*
- `circularAdjListCopier_header` (`entry_entryPoint_circularAdjList_copier`)
 - entry_entryPoint_circularAdjList_copier Fonction de copie pour assembleur_entry_instr_circularAdjList*
- `assembleur_entry * assembleur_new_entryName` (`const char *entryName`)
- `void assembleur_add_data_andComment` (`const char *etiquette, const pseudo_instruction_data pi, const char *valeur, const char *comment`)
- `void assembleur_add_bss_andComment` (`const char *etiquette, const pseudo_instruction_bss pi, const unsigned int nb, const char *comment`)
- `void assembleur_add_entry_andComment` (`assembleur_entry *entry, const char *comment`)
- `assembleur_entry * assembleur_add_entryName_andComment` (`const char *entryName, const char *comment`)
 - assembleur_entry : généré une entrée entryName et l'ajoute dans la liste des entrées de la section data*
- `void assembleur_add_data` (`const char *etiquette, const pseudo_instruction_data pi, const char *valeur`)
 - ///*
- `void assembleur_add_bss` (`const char *etiquette, const pseudo_instruction_bss pi, const unsigned int nb`)
- `void assembleur_add_entry` (`assembleur_entry *entry`)
- `assembleur_entry * assembleur_add_entryName` (`const char *entryName`)
- `void assembleur_dump_section_oneEntry` (`FILE *out, assembleur_entry *entry, const char *commentary`)
- `void assembleur_dump_section_import` (`FILE *out`)
- `void assembleur_dump_section_data` (`FILE *out`)
- `void assembleur_dump_section_bss` (`FILE *out`)
- `void assembleur_dump_section_entry_intrs` (`FILE *out, struct assembleur_entry_instr_circularAdjList *instrs`)
- `void assembleur_dump_section_entry` (`FILE *out`)
- `void assembleur_dump` (`FILE *out`)
- `unsigned int printedSizeOfSigned` (`const signed int val`)
 - printedSizeOfSigned trouve la taille à afficher pour un signed int*
- `unsigned int printedSizeOfUnsigned` (`const unsigned int val`)
 - printedSizeOfUnsigned trouve la taille à afficher pour un unsigned int*
- `const char * assembleur_make_tabVar` (`const char *mTab, const char *reg`)
- `const char * assembleur_make_localVar` (`const char *reg, const int adr`)
- `const char * assembleur_make_globalVar` (`const char *m`)
- `const char * assembleur_make_constant` (`const int c`)
- `const char * assembleur_make_idSuffix` (`const char *prefix, const unsigned int id`)
- `const char * assembleur_make_concat` (`const char *str1, const char *str2`)
- `const char * assembleur_entry_make_prefix` (`assembleur_entry *entry, const char *str`)
- `const char * assembleur_entry_make_suffix` (`assembleur_entry *entry, const char *str`)
- `const char * assembleur_entry_make_next` (`assembleur_entry *entry`)
- `void assembleur_entry_0s_andComment` (`assembleur_entry *entry, const char *instr, const char *comment`)
- `void assembleur_entry_1s_andComment` (`assembleur_entry *entry, const char *instr, const char *str, const char *comment`)
- `void assembleur_entry_2s_andComment` (`assembleur_entry *entry, const char *instr, const char *str1, const char *str2, const char *comment`)
- `void assembleur_entry_add_subEntry_andComment` (`assembleur_entry *entry, assembleur_entry *subEntry, const char *comment`)
- `void assembleur_entry_add_entryPoint_andComment` (`assembleur_entry *entry, assembleur_entry *entryPoint, const char *comment`)
- `assembleur_entry * assembleur_entry_subEntryName_andComment` (`assembleur_entry *entry, const char *subEntryName, const char *comment`)
- `void assembleur_entry_0s` (`assembleur_entry *entry, const char *instr`)
 - ///*
- `void assembleur_entry_1s` (`assembleur_entry *entry, const char *instr, const char *str`)
- `void assembleur_entry_2s` (`assembleur_entry *entry, const char *instr, const char *str1, const char *str2`)

```

— void assembleur_entry_add_subEntry (assembleur_entry *entry, assembleur_entry *subEntry)
— void assembleur_entry_add_entryPoint (assembleur_entry *entry, assembleur_entry *entryPoint)
— assembleur_entry * assembleur_entry_subEntryName (assembleur_entry *entry, const char *subEntryName)
— const char * assembleur_entry_getLastInstr (assembleur_entry *entry)
    /**
— const char * assembleur_entry_getName (assembleur_entry *entry)

```

Variables

```

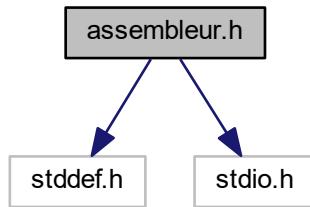
— const char * pseudo_instruction_data_str []
— const char * pseudo_instruction_bss_str []

```

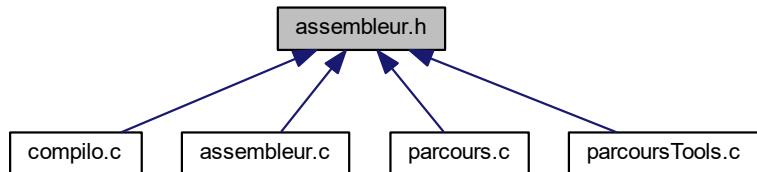
8.10 Référence du fichier assembleur.h

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

Graphe des dépendances par inclusion de assembleur.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

```

— #define GENASM_ENTRY_PRETTY_NAME 1
— #define GENASM_ENTRY_ARE_INDENT 1
— #define GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE 0
— #define GENASM_ENTRYPOINT_SEPERATE_RETURNLINE 0

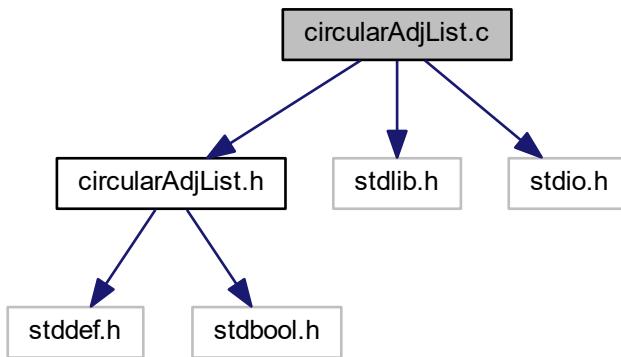
```

Énumérations

```
— enum pseudo_instruction_data {
    db = 0, dw, dd, dq,
    dt }
— enum pseudo_instruction_bss {
    resb = 0, resw, resd, resq,
    rest }
— enum byte_flag { byte = 0, word, dword }
```

8.11 Référence du fichier circularAdjList.c

```
#include "circularAdjList.h"
#include <stdlib.h>
#include <stdio.h>
Graphe des dépendances par inclusion de circularAdjList.c :
```



Fonctions

- struct `AdjList` * `circularAdjList_add_` (struct `AdjList` *circleAdjList, struct `AdjList` *previous_circleAdjList, const size_t sizeofList, void *dataToCopy, `circularAdjListCopier_pf` inserterFunction, `circularAdjListComparator_pf` comparatorFunction)
- struct `AdjList` * `circularAdjList_search_` (struct `AdjList` *circularAdjList, void *dataToSearch, `circularAdjListComparator_pf` comparatorFunction)
- struct `AdjList` * `circularAdjList_free_` (struct `AdjList` *circularAdjList, `circularAdjListFreer_pf` freerFunction)

8.11.1 Documentation des fonctions

8.11.1.1 `circularAdjList_add_()`

```
struct AdjList* circularAdjList_add_ (
    struct AdjList * circleAdjList,
    struct AdjList * previous_circleAdjList,
    const size_t sizeofList,
    void * dataToCopy,
    circularAdjListCopier_pf inserterFunction,
    circularAdjListComparator_pf comparatorFunction )
```

Auteur

Christophe Sonntag

A faire gerer la désallocation

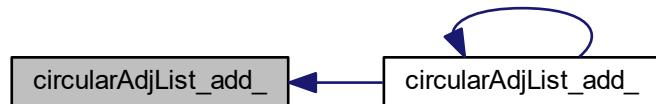
Références circularAdjList_add_(), AdjList ::isEnd, AdjList ::next, et AdjList ::prev.

Référencé par circularAdjList_add_().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



8.11.1.2 circularAdjList_search_()

```
struct AdjList* circularAdjList_search_ (
    struct AdjList * circularAdjList,
    void * dataToSearch,
    circularAdjListComparator_pf comparatorFunction )
```

Références circularAdjList_search_(), AdjList ::isEnd, et AdjList ::next.

Référencé par circularAdjList_search_().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



8.11.1.3 circularAdjList_free_()

```
struct AdjList* circularAdjList_free_ (
    struct AdjList * circularAdjList,
    circularAdjListFreer_pf freeFunction )
```

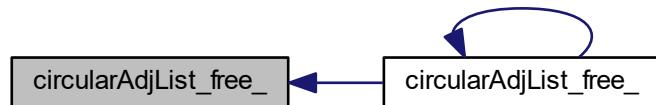
Références circularAdjList_free_(), AdjList ::isEnd, et AdjList ::next.

Référencé par circularAdjList_free_().

Voici le graphe d'appel pour cette fonction :



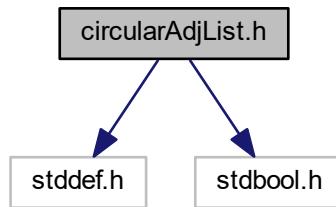
Voici le graphe des appelants de cette fonction :



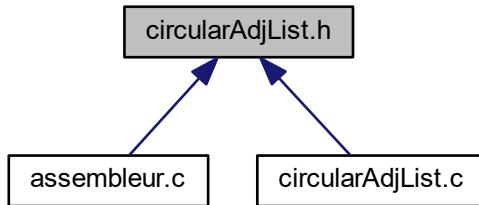
8.12 Référence du fichier circularAdjList.h

```
#include <stddef.h>
#include <stdbool.h>
```

Graphe des dépendances par inclusion de circularAdjList.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

- struct [AdjList](#)
- ///

Macros

```

— #define circularAdjList_add(type_in, circularAdjList, data, copier, comparator) ((type_in*) circularAdjList->
    add_( ( struct AdjList * ) circularAdjList, NULL, sizeof( type_in ), ( void * ) data, copier, comparator))
— #define circularAdjList_search(type_in, circularAdjList, data, comparator) ((type_in*) circularAdjList->
    search_( ( struct AdjList * ) circularAdjList, ( void * ) &data, comparator))
— #define circularAdjList_free(type_in, circularAdjList, freerFunction) circularAdjList = ((type_in*) circularAdjList->
    List_free_( ( struct AdjList * ) circularAdjList, freerFunction ))
— #define circularAdjListStruct_header(structName) struct structName
    /**
     */
— #define circularAdjListStruct_content(structName)
— #define circularAdjListStaticDecl_struct(structName, varName) static struct structName * varName = NULL;
    /**
     */
— #define circularAdjListCopier_header(functionName) void functionName( struct AdjList * circularAdjList, void
    * circularAdjListData )
    /**
     */
— #define circularAdjListComparator_header(functionName) bool functionName( struct AdjList * circularAdjList,
    void * circularAdjListData )
— #define circularAdjListFreer_header(functionName) void functionName( struct AdjList * circularAdjList )
— #define circularAdjListTools_headCast(listType) listType * head = ( listType * ) circularAdjList;
    /**
     */
— #define circularAdjListTools_contentCast(listType, dataType)
— #define circularAdjList_isEnd(circularAdjListPtrVar) (circularAdjListPtrVar->isEnd==1)
    /**
     */
— #define circularAdjList_doWhileNext(circularAdjList) (circularAdjList = circularAdjList_isEnd(circularAdjList) ?
    NULL : circularAdjList->next )

```

Définitions de type

```

— typedef void(* circularAdjListCopier_pf) (struct AdjList *circularAdjList, void *dataToCopy)
— typedef bool(* circularAdjListComparator_pf) (struct AdjList *circularAdjList, void *dataToCompare)
— typedef void(* circularAdjListFreer_pf) (struct AdjList *circularAdjList)

```

Fonctions

```

— struct AdjList * circularAdjList_add_ (struct AdjList *circularAdjList, struct AdjList *previous_circularAdjList,
    const size_t sizeOfList, void *dataToCopy, circularAdjListCopier_pf inserterFunction, circularAdjListComparator_pf
    comparatorFunction)
— struct AdjList * circularAdjList_search_ (struct AdjList *circularAdjList, void *dataToSearch, circularAdjListComparator_pf
    comparatorFunction)
— struct AdjList * circularAdjList_free_ (struct AdjList *circularAdjList, circularAdjListFreer_pf freerFunction)

```

8.12.1 Documentation des macros

8.12.1.1 circularAdjList_add

```
#define circularAdjList_add(
    type_in,
    circularAdjList,
    data,
    copier,
    comparator ) ((type_in*) circularAdjList_add_( ( struct AdjList * ) circularAdjList,
    NULL, sizeof( type_in ), ( void * ) data, copier, comparator))
```

Référencé par assembleur_add_bss(), assembleur_add_entry(), assembleur_add_entryName(), assembleur_entry_addEntryPoint(), et assembleur_entry_add_subEntry().

8.12.1.2 circularAdjList_search

```
#define circularAdjList_search(
    type_in,
    circularAdjList,
    data,
    comparator ) ((type_in*) circularAdjList_search_( ( struct AdjList * ) circularAdjList, ( void * ) &data, comparator))
```

8.12.1.3 circularAdjList_free

```
#define circularAdjList_free(
    type_in,
    circularAdjList,
    freerFunction ) circularAdjList = ((type_in*) circularAdjList_free_( ( struct AdjList * ) circularAdjList, freerFunction ))
```

8.12.1.4 circularAdjListStruct_header

```
#define circularAdjListStruct_header(
    structName ) struct structName

///
```

8.12.1.5 circularAdjListStruct_content

```
#define circularAdjListStruct_content(
    structName )
```

Valeur :

```
struct structName * next, * prev; \
char isEnd;
```

Référencé par circularAdjListStaticDecl_struct(), et circularAdjListStruct_header().

8.12.1.6 circularAdjListStaticDecl_struct

```
#define circularAdjListStaticDecl_struct(
    structName,
    varName ) static struct structName * varName = NULL;

///
```

8.12.1.7 circularAdjListCopier_header

```
#define circularAdjListCopier_header(
    functionName ) void functionName( struct AdjList * circularAdjList, void * circularAdjListData )
```

```
///
```

8.12.1.8 circularAdjListComparator_header

```
#define circularAdjListComparator_header(
    functionName ) bool functionName( struct AdjList * circularAdjList, void * circularAdjListData )
```

8.12.1.9 circularAdjListFreer_header

```
#define circularAdjListFreer_header(
    functionName ) void functionName( struct AdjList * circularAdjList )
```

8.12.1.10 circularAdjListTools_headCast

```
#define circularAdjListTools_headCast(
    listType ) listType * head = ( listType * ) circularAdjList;
///
```

8.12.1.11 circularAdjListTools_contentCast

```
#define circularAdjListTools_contentCast(
    listType,
    dataType )
```

Valeur :

```
circularAdjListTools_headCast(listType) \
    dataType * data = ( dataType * ) circularAdjListData;
```

Référencé par circularAdjListComparator_header(), et circularAdjListCopier_header().

8.12.1.12 circularAdjList_isEnd

```
#define circularAdjList_isEnd(
    circularAdjListPtrVar ) (circularAdjListPtrVar->isEnd==1)
///
```

8.12.1.13 circularAdjList_doWhileNext

```
#define circularAdjList_doWhileNext(
    circularAdjList ) (circularAdjList = circularAdjList_isEnd(circularAdjList) ? NULL : circularAdjList->next )
```

Référencé par assembleur_dump_section_bss(), assembleur_dump_section_data(), et assembleur_dump_section_entry().

8.12.2 Documentation des définitions de type

8.12.2.1 circularAdjListCopier_pf

```
typedef void( * circularAdjListCopier_pf) (struct AdjList *circularAdjList, void *dataToCopy)
```

8.12.2.2 circularAdjListComparator_pf

```
typedef bool( * circularAdjListComparator_pf) (struct AdjList *circularAdjList, void *dataToCompare)
```

8.12.2.3 circularAdjListFreer_pf

```
typedef void( * circularAdjListFreer_pf) (struct AdjList *circularAdjList)
```

8.12.3 Documentation des fonctions

8.12.3.1 circularAdjList_add_()

```
struct AdjList* circularAdjList_add_ (
    struct AdjList * circleAdjList,
    struct AdjList * previous_circleAdjList,
    const size_t sizeofList,
    void * dataToCopy,
    circularAdjListCopier_pf inserterFunction,
    circularAdjListComparator_pf comparatorFunction )
```

Auteur

Christophe Sonntag

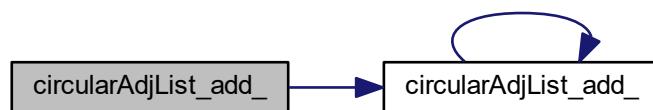
A faire

gerer la désallocation

Références circularAdjList_add_(), AdjList ::isEnd, AdjList ::next, et AdjList ::prev.

Référencé par circularAdjList_add_().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



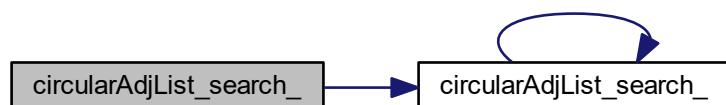
8.12.3.2 circularAdjList_search_()

```
struct AdjList* circularAdjList_search_ (
    struct AdjList * circularAdjList,
    void * dataToSearch,
    circularAdjListComparator_pf comparatorFunction )
```

Références circularAdjList_search_(), AdjList ::isEnd, et AdjList ::next.

Référencé par circularAdjList_search_().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



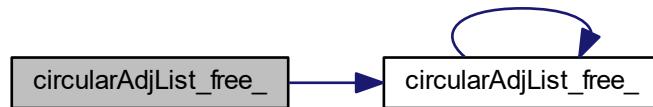
8.12.3.3 circularAdjList_free_()

```
struct AdjList* circularAdjList_free_ (
    struct AdjList * circularAdjList,
    circularAdjListFreeer_pf freerFunction )
```

Références circularAdjList_free_(), AdjList ::isEnd, et AdjList ::next.

Référencé par circularAdjList_free_().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :

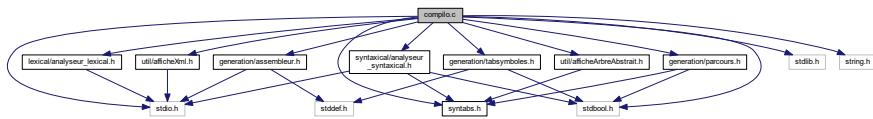


8.13 Référence du fichier compilo.c

```
#include "lexical/analyseur_lexical.h"
#include "syntaxical/analyseur_syntaxical.h"
#include "syntaxical/syntabs.h"
#include "util/afficheXml.h"
#include "util/afficheArbreAbstrait.h"
#include "generation/parcours.h"
#include "generation/tabsymboles.h"
#include "generation/assembleur.h"
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
```

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

Graphe des dépendances par inclusion de compilo.c :



Énumérations

- enum `ArgsSecond` { `ArgsSecond_None`, `ArgsSecond_outputPath` }
The `ArgsSecond` : enum nécessaire pour parser la seconde valeur d'un arguments.

Fonctions

- void `afficherAideF()`
- int `main` (int argc, char *argv[])

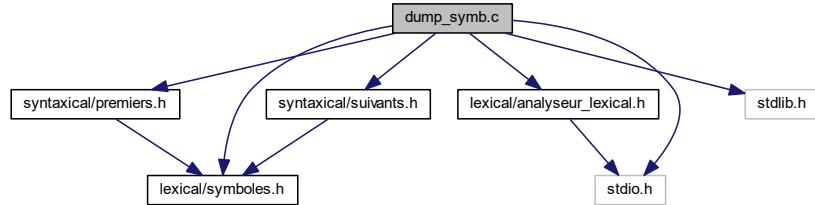
Variables

- char `ytext` [100]
- FILE * `yyin` = NULL

8.14 Référence du fichier dump_symb.c

```
#include "syntaxical/premiers.h"
#include "syntaxical/suivants.h"
#include "lexical/symboles.h"
#include "lexical/analyseur_lexical.h"
#include <stdio.h>
#include <stdlib.h>
```

Graphe des dépendances par inclusion de dump_symb.c :



Fonctions

- int `main` ()

Variables

- char `yytext` [100]
- FILE * `yyin`

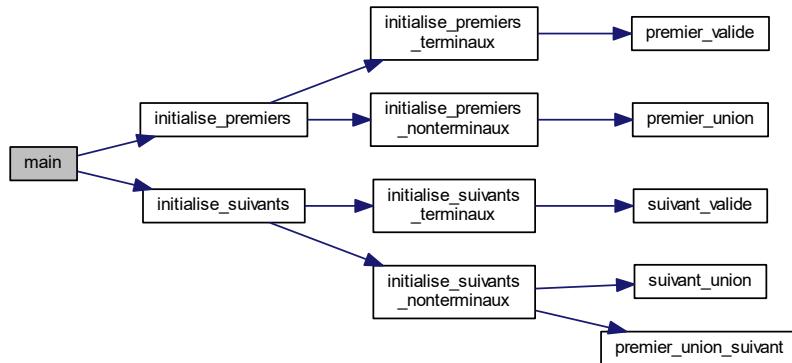
8.14.1 Documentation des fonctions

8.14.1.1 main()

```
int main ( )
```

Références `initialise_premiers()`, et `initialise_suivants()`.

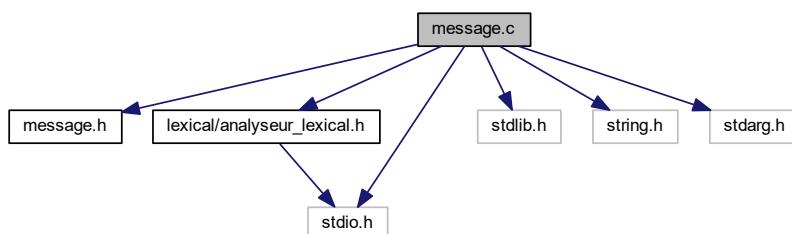
Voici le graphe d'appel pour cette fonction :



8.15 Référence du fichier message.c

```
#include "message.h"
#include "lexical/analyseur_lexical.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
```

Graphe des dépendances par inclusion de `message.c` :



Fonctions

- void **warningLigne** (char *message)

warningLigne : Affiche le message d'alerte donné en paramètre, avec le numéro de ligne
- void **erreurLigne** (char *message)

erreurLigne : Affiche le message d'erreur donné en paramètre, avec le numéro de ligne
- void **erreur** (char *message)

erreur : Affiche le message d'erreur donné en paramètre
- void **erreurArgs** (const char *fmt,...)

erreur : Affiche le message d'erreur donné en paramètre avec un Wrapper pour printf
- void **warning_1s** (char *message, char *s)

warning_1s : Affiche le message d'alerte donné en paramètre, avec le numéro de ligne. Le message d'alerte peut contenir un s variable, qui sera donné en argument s
- void **erreur_1s** (char *message, char *s)

erreur_1s : Affiche le message d'erreur donné en paramètre, avec le numéro de ligne. Le message d'erreur peut contenir un s variable, qui sera donné en argument s

8.15.1 Documentation des fonctions

8.15.1.1 warningLigne()

```
void warningLigne (
    char * message )
```

warningLigne : Affiche le message d'alerte donné en paramètre, avec le numéro de ligne

Références nb_ligne.

8.15.1.2 erreurLigne()

```
void erreurLigne (
    char * message )
```

erreurLigne : Affiche le message d'erreur donné en paramètre, avec le numéro de ligne

Références nb_ligne.

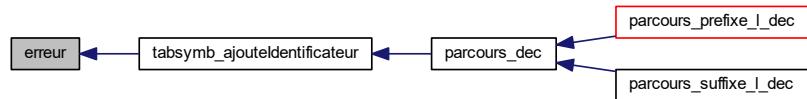
8.15.1.3 erreur()

```
void erreur (
    char * message )
```

erreur : Affiche le message d'erreur donné en paramètre

Référencé par tabsymb_ajoutelidentificateur().

Voici le graphe des appelants de cette fonction :



8.15.1.4 erreurArgs()

```
void erreurArgs (
    const char * fmt,
    ...
)
```

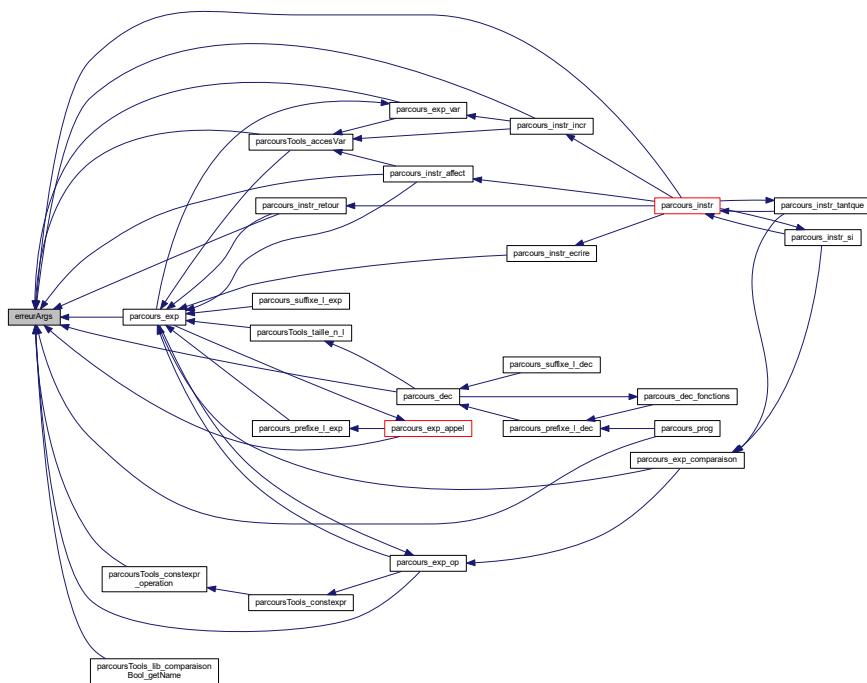
erreur : Affiche le message d'erreur donné en paramètre avec un Wrapper pour printf

Auteur

(<http://stackoverflow.com/a/14766163/7423251>)

Référencé par parcours_dec(), parcours_exp(), parcours_exp_appel(), parcours_exp_op(), parcours_exp←var(), parcours_instr(), parcours_instr_affect(), parcours_instr_incr(), parcours_instr_retour(), parcours_prog(), parcoursTools_accesVar(), parcoursTools_constexpr_operation(), et parcoursTools_lib_comparaisonBool_get←Name().

Voici le graphe des appels de cette fonction :



8.15.1.5 warning_1s()

```
void warning_1s (
    char * message,
    char * s )
```

`warning_1s` : Affiche le message d'alerte donné en paramètre, avec le numéro de ligne. Le message d'alerte peut contenir un `s` variable, qui sera donné en argument `s`

Références nb_ligne.

8.15.1.6 erreur_1s()

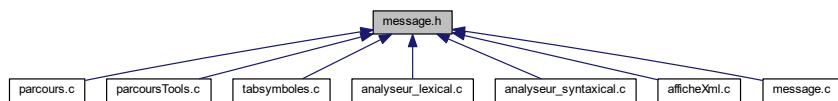
```
void erreur_1s (
    char * message,
    char * s )
```

erreur_1s : Affiche le message d'erreur donné en paramètre, avec le numéro de ligne. Le message d'erreur peut contenir un *s* variable, qui sera donné en argument *s*

Références nb_ligne.

8.16 Référence du fichier message.h

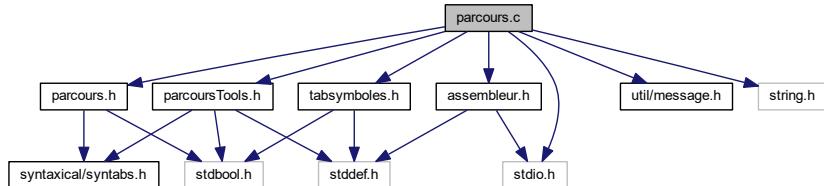
Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.17 Référence du fichier parcours.c

```
#include "parcours.h"
#include "parcoursTools.h"
#include "tabsymboles.h"
#include "assembleur.h"
#include "util/message.h"
#include <stdio.h>
#include <string.h>
```

Graphe des dépendances par inclusion de parcours.c :



Macros

- #define **ErreursSiNulle**(var) if(var==NULL) { **erreurArgs**("La fonction '%s' n'accepte pas de valeur nulle", __FUNCTION__); }
- #define **ErreursSiMauvaisType**(var, leTypeVoulut) if(var->type != leTypeVoulut) { **erreurArgs**("La fonction '%s' necessite le type '%s'", __FUNCTION__, #leTypeVoulut); }

Fonctions

- void `parcours_prog` (const `n_prog` **n*)

parcours_prog : Génère la table des symbole et le code assembleur
- void `parcours_init_asm` ()

`parcours_get_endEntryName` genère une entrée (`"assembleur_entry"`) de type 'entryPoint' conçue pour être placée en fin de fonction.
- unsigned int `parcours_prefixe_l_dec` (const `n_l_dec` **n*, int porte)

`parcours_dec` Pour le moment, les seules déclarations prises en compte sont les variables globales. Lors du parcours d'un noeud correspondant à une variable globale, il faut allouer de la place pour cette variable dans la région .bss du code machine, à l'aide des pseudo instructions resb, resw . .
- unsigned int `parcours_suffixe_l_dec` (const `n_l_dec` **n*, int porte)

`parcours_dec`
- void `parcours_dec` (const `n_dec` **n*, int porte)

`parcours_instr` : Seules les instructions de type ecrire et affect seront traitées pendant ce TP. Une instruction ecrire doit être traduite par un appel système int 0x80, une affectation stocke la valeur d'un registre (qui contient le résultat de `parcours_exp`) dans une adresse mémoire. Les adresses des variables simples peuvent être représentées sous forme d'étiquette, tandis que les adresses des tableaux sont aussi des expressions dont la valeur doit être calculée.
- void `parcours_instr_ecrire` (const `n_exp` **n*)

`parcours_exp` : Cette fonction génère les instructions correspondant au calcul de la valeur de l'expression et dépose le résultat du calcul dans la pile. La génération de code pour une opération de type arg1 op arg2 consiste à affecter à des registres les deux opérandes arg1 et arg2 puis à réaliser l'opération à l'aide de l'instruction correspondante et enfin à empiler le résultat de l'opération. Pour les instructions complexes, les valeurs des opérandes (sous expressions) seront prises dans la pile, c'est là qu'aura été déposé le résultat de ces sous expressions. Il faut aussi traiter les lectures de variables lire avec un appel système.
- void `parcours_instr_affect` (const `n_var` **var*, const `n_exp` **exp*)

`parcours_exp_comparaison` : est executer a partir d'une fonction de comparaison (pas d'opération) permet de traduire n'importe quelle expression en comparaison (comparaison, test vrai...)
- void `parcours_instr_incr` (const `n_var` **var*)

`parcours_exp_lire` ()
- void `parcours_instr_tantque` (const `n_exp` *test, const `n_instr` *faire)

`parcours_exp_var` (const `n_var` *variable)
- void `parcours_instr_appel` (const `n_appel` *appel)

`parcours_exp_int` (const int entier)
- void `parcours_instr_si` (const `n_exp` *test, const `n_instr` *alors, const `n_instr` *sinon)

`parcours_exp_appel` (const `n_appel` *appel, const bool desallocReturnValue)
- void `parcours_instr_retour` (const `n_exp` *expression)

`parcours_exp_op` (const `n_exp` **n*, const bool isComparaison)
- unsigned int `parcours_prefixe_l_exp` (const `n_l_exp` **n*)

`parcours_exp_op`
- unsigned int `parcours_suffixe_l_exp` (const `n_l_exp` **n*)

`parcours_exp_op`
- void `parcours_exp` (const `n_exp` **n*)

`parcours_exp_op`

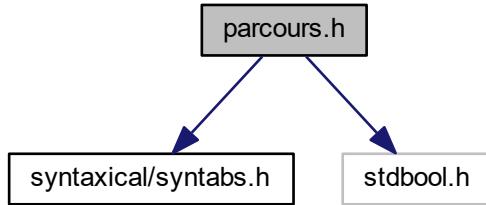
Variables

- int `adresseGlobalCourant`
- int `adresseLocaleCourante`
- int `adresseArgumentCourant`
- `assembleur_entry` * *ce* = NULL
- `assembleur_entry` * *topEntry* = NULL
- `assembleur_entry` * *endEntry* = NULL

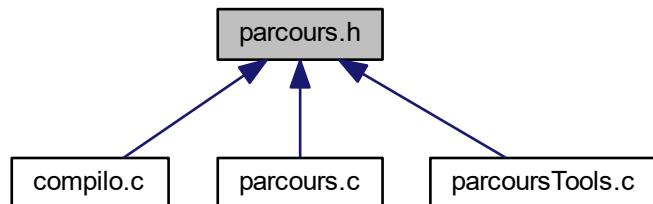
8.18 Référence du fichier parcours.h

```
#include "syntaxical/syntabs.h"
#include <stdbool.h>
```

Graphe des dépendances par inclusion de parcours.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

```

— #define PARCOURS_STARTENTRY_ACCEPT_RETVALUE 1
— #define PARCOURS_VAR_USE_DWORD 1
— #define HAVE_LOGIQUE_COMPARAISSON 0
— #define IO_USE_READLINE_ENTRY 1
  
```

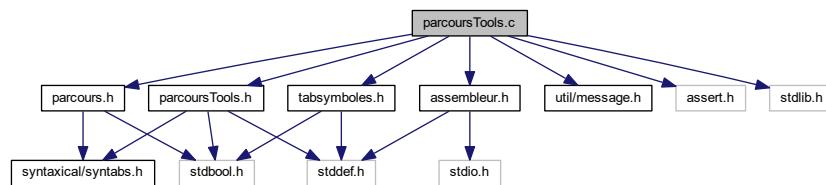
8.19 Référence du fichier parcoursTools.c

```

#include "parcoursTools.h"
#include "parcours.h"
#include "util/message.h"
#include "tabsymboles.h"
#include "assembleur.h"
#include <assert.h>
  
```

```
#include <stdlib.h>
```

Graphe des dépendances par inclusion de parcoursTools.c :



Macros

- #define LIB_COMPARATOREXPR_PREFIX "_cmpBin_"
- #define LIB_COMPARATOREXPR_SIZE 6
- #define LIB_COMPARAISONBOOL_PREFIX "_cmpBool_"
- #define LIB_COMPARAISONBOOL_SIZE 3

Fonctions

- int `parcoursTools_constexpr_operation` (const `operation` op, const int op1, const int op2, bool *compute ← Comparaison)
 - calculer_exp const permet de calculer les opérations constantes*
- bool `parcoursTools_constexpr` (const `n_expr` *n, int *compute, bool *computeComparaison)
 - parcoursTools_constexpr : créer la valeur constante, si l'opération (recursive) ne contient que des entiers*
- const char * `parcoursTools_computeOperation` (const `operation` op, const char **reg, bool *oneOperand ← Destination, bool *isCall)
 - parcoursTools_computeOperation*
- const char * `parcoursTools_comparaisonOperation` (const `operation` op)
 - size_t parcoursTools_taille_n_l (n_l_dec *n)*
 - parcoursTools_taille_n_l : renvoi la longueur d'une liste chainée*
- void `parcours_exp` (const `n_expr` *n)
 - parcours_exp : Cette fonction génère les instructions correspondant au calcul de la valeur de l'expression et dépose le résultat du calcul dans la pile. La génération de code pour une opération de type arg1 op arg2 consiste à affecter à des registres les deux opérandes arg1 et arg2 puis à réaliser l'opération à l'aide de l'instruction correspondante et enfin à empiler le résultat de l'opération. Pour les instructions complexes, les valeurs des opérandes (sous expressions) seront prises dans la pile, c'est là qu'aura été déposé le résultat de ces sous expressions. Il faut aussi traiter les lectures de variables lire avec un appel système.*
- const char * `parcoursTools_accesVar` (const `n_var` *variable, const char **comment)
 - parcoursTools_accesVar*
- const char * `parcoursTools_lib_comparatorExpr` (`operation` op)
 - `void parcoursTools_genLib_comparatorExpr ()`
- const char * `parcoursTools_lib_comparaisonBool_getName` (`operation` op)
 - `const char * parcoursTools_lib_comparaisonBool (operation op)`
- void `parcoursTools_genLib_comparaisonBool ()`
 - `void parcoursTools_dump_asm_lib ()`

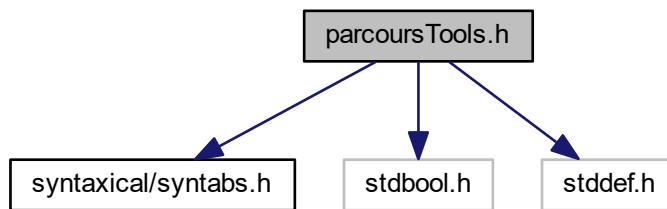
Variables

- `assembleur_entry` * ce

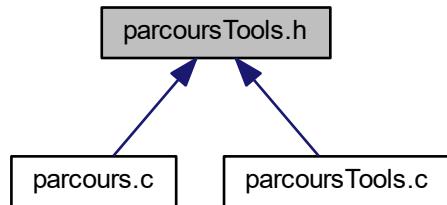
8.20 Référence du fichier parcoursTools.h

```
#include "syntaxical/syntabs.h"
#include <stdbool.h>
#include <stddef.h>
```

Graphe des dépendances par inclusion de parcoursTools.h :



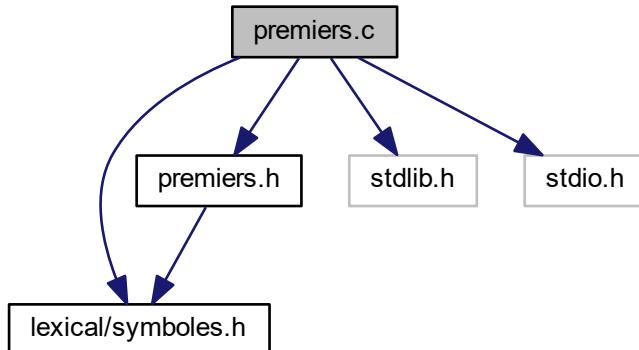
Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.21 Référence du fichier premiers.c

```
#include "lexical/symboles.h"
#include "premiers.h"
#include <stdlib.h>
#include <stdio.h>
```

Graphe des dépendances par inclusion de premiers.c :



Fonctions

- int `est_premier` (int non_terminal, int terminal)
- void `initialise_premiers_terminaux` ()
- void `initialise_premiers_nonterminaux` ()
- void `initialise_premiers` (void)
- void `premier_valide` (int non_terminal, int terminal)
 - premier_valide valide un terminal de la table 'premier' pour un 'non_terminal' donné*
- void `premier_union` (int idSource, int idDestination)
 - premier_union permet de copier un groupe de terminaux de la table 'premier'.*

Variables

- int `premiers` [NB_NON_TERMINAUX+1][NB_TERMINAUX+1]

8.21.1 Documentation des fonctions

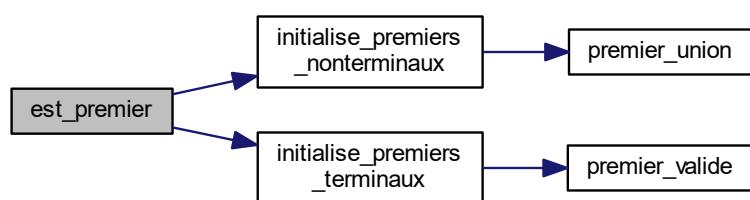
8.21.1.1 `est_premier()`

```
int est_premier (
    int non_terminal,
    int terminal )
```

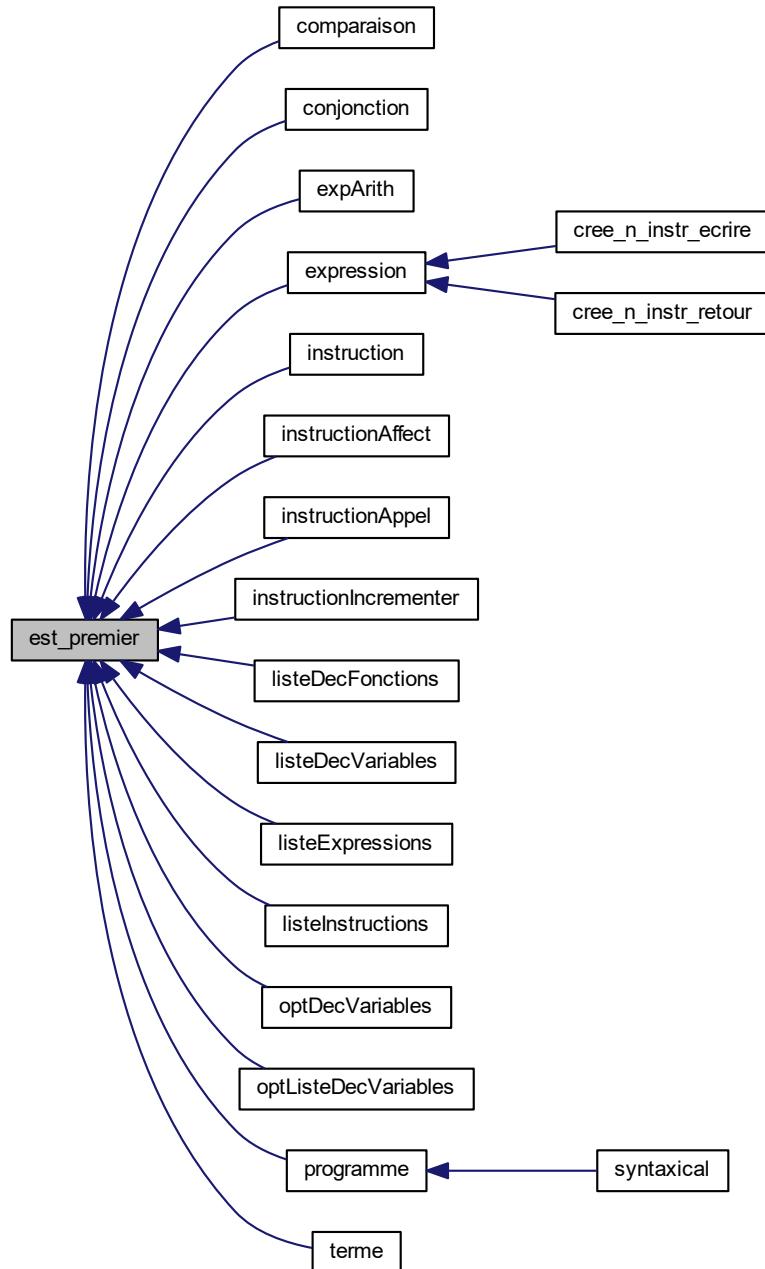
Références `initialise_premiers_nonterminaux()`, `initialise_premiers_terminaux()`, et `premiers`.

Référencé par `comparaison()`, `conjonction()`, `expArith()`, `expression()`, `instruction()`, `instructionAffect()`, `instruction←Appel()`, `instructionIncrementer()`, `listeDecFonctions()`, `listeDecVariables()`, `listeExpressions()`, `listInstructions()`, `optDecVariables()`, `optListeDecVariables()`, `programme()`, et `terme()`.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



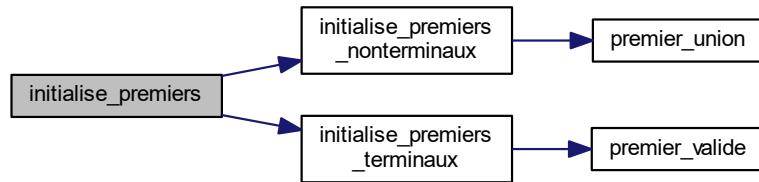
8.21.1.2 initialise_premiers()

```
void initialise_premiers (
    void )
```

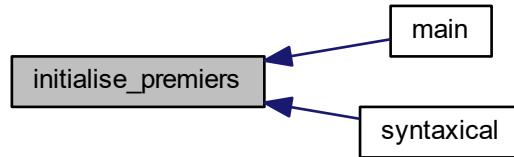
Références `initialise_premiers_nonterminaux()`, `initialise_premiers_terminaux()`, `NB_NON_TERMINAUX`, `NB_TERMINAUX`, et `premiers`.

Référencé par main(), et syntaxical().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



8.21.2 Documentation des variables

8.21.2.1 premiers

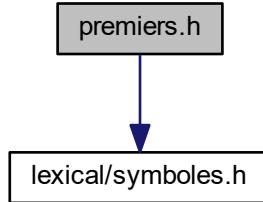
```
int premiers[NB_NON_TERMINAUX+1][NB_TERMINAUX+1]
```

Référencé par est_premier(), initialise_premiers(), premier_union(), premier_union_suivant(), et premier_valide().

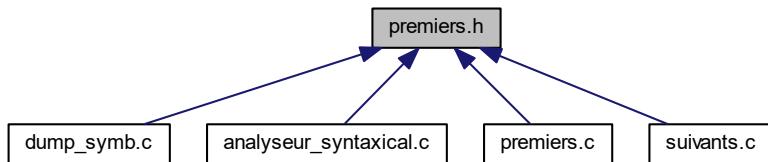
8.22 Référence du fichier premiers.h

```
#include "lexical/symboles.h"
```

Graphe des dépendances par inclusion de premiers.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Variables

— int `premiers [NB_NON_TERMINAUX+1][NB_TERMINAUX+1]`

8.22.1 Documentation des variables

8.22.1.1 premiers

`int premiers [NB_NON_TERMINAUX+1] [NB_TERMINAUX+1]`

Référencé par `est_premier()`, `initialise_premiers()`, `premier_union()`, `premier_union_suivant()`, et `premier_valide()`.

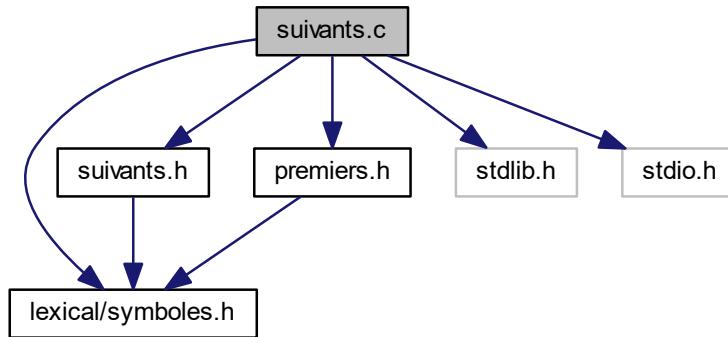
8.23 Référence du fichier README.md

8.24 Référence du fichier suivants.c

```
#include "lexical/symboles.h"
#include "suivants.h"
```

```
#include "premiers.h"
#include <stdlib.h>
#include <stdio.h>
```

Graphe des dépendances par inclusion de suivants.c :



Fonctions

- int [est_suivant](#) (int terminal, int non_terminal)
- void [initialise_suivants_terminaux](#) ()
- void [initialise_suivants_nonterminaux](#) ()
- void [initialise_suivants](#) (void)
- void [suivant_valide](#) (int non_terminal, int terminal)

suivant_valide valide un terminal de la table 'suivant' pour un 'non_terminal' donné
- void [suivant_union](#) (int idSource, int idDestination)

suivant_union permet de copier un groupe de terminaux de la table 'suivant'.
- void [premier_union_suivant](#) (int idSourcePremier, int idDestinationSuivant)

premier_union_suivant permet de copier un groupe de terminaux de la table 'premiers' vers la table 'suivant'.

Variables

- int [suivants](#) [NB_NON_TERMINAUX+1][NB_TERMINAUX+1]

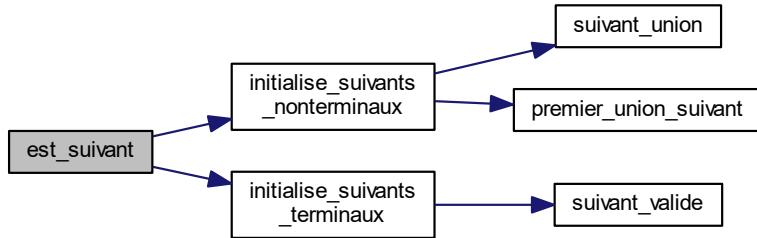
8.24.1 Documentation des fonctions

8.24.1.1 [est_suivant\(\)](#)

```
int est_suivant (
    int terminal,
    int non_terminal )
```

Références [initialise_suivants_nonterminaux\(\)](#), [initialise_suivants_terminaux\(\)](#), et [suivants](#).

Voici le graphe d'appel pour cette fonction :



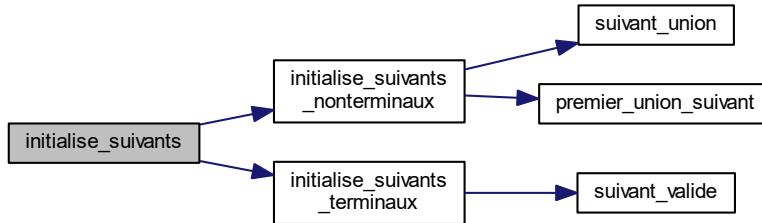
8.24.1.2 initialise_suivants()

```
void initialise_suivants (
    void )
```

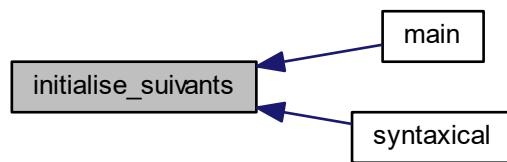
Références initialise_suivants_nonterminaux(), initialise_suivants_terminaux(), NB_NON_TERMINAUX, NB_TERMINAUX, et suivants.

Référencé par main(), et syntaxical().

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



8.24.2 Documentation des variables

8.24.2.1 suivants

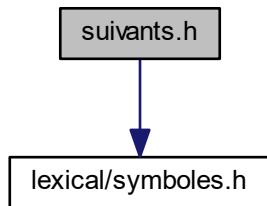
```
int suivants[NB_NON_TERMINAUX+1][NB_TERMINAUX+1]
```

Référencé par est_suivant(), initialise_suivants(), premier_union_suivant(), suivant_union(), et suivant_valide().

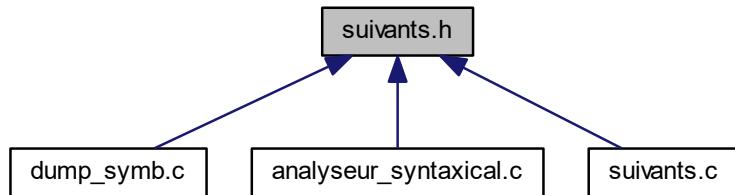
8.25 Référence du fichier suivants.h

```
#include "lexical/symboles.h"
```

Graphe des dépendances par inclusion de suivants.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Variables

- int suivants [NB_NON_TERMINAUX+1][NB_TERMINAUX+1]

8.25.1 Documentation des variables

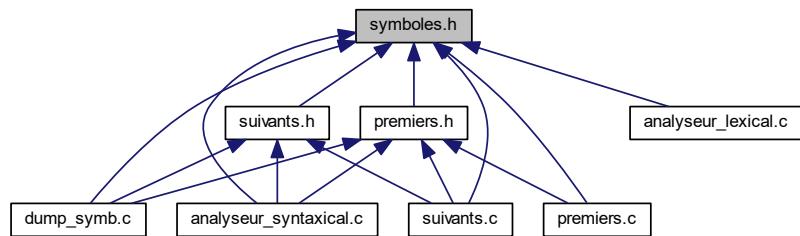
8.25.1.1 suivants

```
int suivants[NB_NON_TERMINAUX+1] [NB_TERMINAUX+1]
```

Référencé par est_suivant(), initialise_suivants(), premier_union_suivant(), suivant_union(), et suivant_valide().

8.26 Référence du fichier symboles.h

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

```
— #define EPSILON 0
— #define NB_NON_TERMINAUX 43
— #define _listeDecVariables_ 1
— #define _listeDecFonctions_ 2
— #define _declarationVariable_ 3
— #define _declarationFonction_ 4
— #define _listeParam_ 5
— #define _listelInstructions_ 6
— #define _instruction_ 8
— #define _instructionAffect_ 9
— #define _instructionBloc_ 10
— #define _instructionSi_ 11
— #define _instructionTantque_ 12
— #define _instructionAppel_ 13
— #define _instructionRetour_ 14
— #define _instructionEcriture_ 15
— #define _instructionIncrementer_ 16
— #define _instructionVide_ 17
— #define _var_ 18
— #define _expression_ 19
— #define _appelFct_ 20
— #define _conjonction_ 21
— #define _negation_ 22
— #define _comparaison_ 23
— #define _expArith_ 24
— #define _terme_ 25
— #define _facteur_ 26
— #define _argumentsEffectifs_ 27
— #define _listeExpressions_ 28
— #define _listeExpressionsBis_ 29
— #define _conjonctionBis_ 30
— #define _optTailleTableau_ 31
— #define _expArithBis_ 32
— #define _optSinon_ 33
```

```

— #define _comparaisonBis_ 34
— #define _optDecVariables_ 35
— #define _optIndice_ 36
— #define _listeDecVariablesBis_ 37
— #define _instructionPour_ 38
— #define _termeBis_ 39
— #define _expressionBis_ 40
— #define _instructionFaire_ 41
— #define _optListeDecVariables_ 42
— #define _programme_ 43
— #define NB_TERMINAUX 36
— #define POINT_VIRGULE 1
— #define PLUS 2
— #define MOINS 3
— #define FOIS 4
— #define DIVISE 5
— #define MODULO 6
— #define PARENTHESE_OUVRANTE 7
— #define PARENTHESE_FERMANTE 8
— #define CROCHET_OUVRANT 9
— #define CROCHET_FERMANT 10
— #define ACCOLADE_OUVRANTE 11
— #define ACCOLADE_FERMANTE 12
— #define EGAL 13
— #define DIFFERENT 14
— #define INFERIEUR 15
— #define SUPERIEUR 16
— #define INFERIEUR_EGAL 17
— #define SUPERIEUR_EGAL 18
— #define ET 19
— #define OU 20
— #define NON 21
— #define SI 22
— #define ALORS 23
— #define SINON 24
— #define TANTQUE 25
— #define FAIRE 26
— #define ENTIER 27
— #define RETOUR 28
— #define LIRE 29
— #define ECRIRE 30
— #define INCR 31
— #define ID_VAR 32
— #define ID_FCT 33
— #define NOMBRE 34
— #define FIN 35
— #define VIRGULE 36

```

8.26.1 Documentation des macros

8.26.1.1 EPSILON

```
#define EPSILON 0
```

Référencé par initialise_premiers_terminaux(), et premier_union_suivant().

8.26.1.2 NB_NON_TERMINAUX

```
#define NB_NON_TERMINAUX 43
```

Référencé par initialise_premiers(), et initialise_suivants().

8.26.1.3 _listeDecVariables_

```
#define _listeDecVariables_ 1
```

Référencé par initialise_premiers_nonterminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.4 listeDecFonctions

```
#define _listeDecFonctions_ 2
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.5 declarationVariable

```
#define _declarationVariable_ 3
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.6 declarationFonction

```
#define _declarationFonction_ 4
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.7 listeParam

```
#define _listeParam_ 5
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.8 listelInstructions

```
#define _listelInstructions_ 6
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.9 instruction

```
#define _instruction_ 8
```

Référencé par initialise_premiers_nonterminaux(), et initialise_suivants_nonterminaux().

8.26.1.10 instructionAffect

```
#define _instructionAffect_ 9
```

Référencé par initialise_premiers_nonterminaux(), et initialise_suivants_nonterminaux().

8.26.1.11 _instructionBloc_

```
#define _instructionBloc_ 10
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.12 _instructionSi_

```
#define _instructionSi_ 11
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.13 _instructionTantque_

```
#define _instructionTantque_ 12
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.14 _instructionAppel_

```
#define _instructionAppel_ 13
```

Référencé par initialise_premiers_nonterminaux(), et initialise_suivants_nonterminaux().

8.26.1.15 _instructionRetour_

```
#define _instructionRetour_ 14
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.16 _instructionEcriture_

```
#define _instructionEcriture_ 15
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.17 _instructionIncrementer_

```
#define _instructionIncrementer_ 16
```

Référencé par initialise_premiers_nonterminaux(), et initialise_premiers_terminaux().

8.26.1.18 _instructionVide_

```
#define _instructionVide_ 17
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_↔ nonterminaux().

8.26.1.19 _var_

```
#define _var_ 18
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_↔ nonterminaux().

8.26.1.20 _expression_

```
#define _expression_ 19
```

Référencé par initialise_premiers_nonterminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_↔ terminaux().

8.26.1.21 _appelFct_

```
#define _appelFct_ 20
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.22 _conjonction_

```
#define _conjonction_ 21
```

Référencé par initialise_premiers_nonterminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_↔ terminaux().

8.26.1.23 _negation_

```
#define _negation_ 22
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_↔ nonterminaux().

8.26.1.24 _comparaison_

```
#define _comparaison_ 23
```

Référencé par initialise_premiers_nonterminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_↔ terminaux().

8.26.1.25 _expArith_

```
#define _expArith_ 24
```

Référencé par initialise_premiers_nonterminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.26 _terme_

```
#define _terme_ 25
```

Référencé par initialise_premiers_nonterminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.27 _facteur_

```
#define _facteur_ 26
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.28 _argumentsEffectifs_

```
#define _argumentsEffectifs_ 27
```

8.26.1.29 _listeExpressions_

```
#define _listeExpressions_ 28
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.30 _listeExpressionsBis_

```
#define _listeExpressionsBis_ 29
```

Référencé par initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.31 _conjonctionBis_

```
#define _conjonctionBis_ 30
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.32 _optTailleTableau_

```
#define _optTailleTableau_ 31
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.33 _expArithBis_

```
#define _expArithBis_ 32
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.34 _optSinon_

```
#define _optSinon_ 33
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.35 _comparaisonBis_

```
#define _comparaisonBis_ 34
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.36 _optDecVariables_

```
#define _optDecVariables_ 35
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.37 _optIndice_

```
#define _optIndice_ 36
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.38 _listeDecVariablesBis_

```
#define _listeDecVariablesBis_ 37
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.39 _instructionPour_

```
#define _instructionPour_ 38
```

8.26.1.40 _termeBis_

```
#define _termeBis_ 39
```

Référencé par initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.41 _expressionBis_

```
#define _expressionBis_ 40
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.42 _instructionFaire_

```
#define _instructionFaire_ 41
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), et initialise_suivants_nonterminaux().

8.26.1.43 _optListeDecVariables_

```
#define _optListeDecVariables_ 42
```

Référencé par initialise_premiers_nonterminaux(), initialise_premiers_terminaux(), initialise_suivants_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.44 _programme_

```
#define _programme_ 43
```

Référencé par initialise_premiers_nonterminaux(), et initialise_suivants_terminaux().

8.26.1.45 NB_TERMINAUX

```
#define NB_TERMINAUX 36
```

Référencé par initialise_premiers(), initialise_suivants(), premier_union(), premier_union_suivant(), et suivant_union().

8.26.1.46 POINT_VIRGULE

```
#define POINT_VIRGULE 1
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.47 PLUS

```
#define PLUS 2
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.48 MOINS

```
#define MOINS 3
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.49 FOIS

```
#define FOIS 4
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.50 DIVISE

```
#define DIVISE 5
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.51 MODULO

```
#define MODULO 6
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.52 PARENTHÈSE_OUVRANTE

```
#define PARENTHÈSE_OUVRANTE 7
```

Référencé par initialise_premiers_terminaux().

8.26.1.53 PARENTHÈSE_FERMANTE

```
#define PARENTHÈSE_FERMANTE 8
```

Référencé par initialise_suivants_terminaux().

8.26.1.54 CROCHET_OUVRANT

```
#define CROCHET_OUVRANT 9
```

Référencé par initialise_premiers_terminaux().

8.26.1.55 CROCHET_FERMANT

```
#define CROCHET_FERMANT 10
```

Référencé par initialise_suivants_terminaux().

8.26.1.56 ACCOLADE_OUVRANTE

```
#define ACCOLADE_OUVRANTE 11
```

Référencé par initialise_premiers_terminaux().

8.26.1.57 ACCOLADE_FERMANTE

```
#define ACCOLADE_FERMANTE 12
```

Référencé par initialise_suivants_terminaux().

8.26.1.58 EGAL

```
#define EGAL 13
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.59 DIFFERENT

```
#define DIFFERENT 14
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.60 INFERIEUR

```
#define INFERIEUR 15
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.61 SUPERIEUR

```
#define SUPERIEUR 16
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.62 INFERIEUR_EGAL

```
#define INFERIEUR_EGAL 17
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.63 SUPERIEUR_EGAL

```
#define SUPERIEUR_EGAL 18
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.64 ET

```
#define ET 19
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.65 OU

```
#define OU 20
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.66 NON

```
#define NON 21
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.67 SI

```
#define SI 22
```

Référencé par initialise_premiers_terminaux().

8.26.1.68 ALORS

```
#define ALORS 23
```

Référencé par initialise_suivants_terminaux().

8.26.1.69 SINON

```
#define SINON 24
```

Référencé par initialise_premiers_terminaux().

8.26.1.70 TANTQUE

```
#define TANTQUE 25
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.71 FAIRE

```
#define FAIRE 26
```

Référencé par initialise_premiers_terminaux(), et initialise_suivants_terminaux().

8.26.1.72 ENTIER

```
#define ENTIER 27
```

Référencé par initialise_premiers_terminaux().

8.26.1.73 RETOUR

```
#define RETOUR 28
```

Référencé par initialise_premiers_terminaux().

8.26.1.74 LIRE

```
#define LIRE 29
```

Référencé par initialise_premiers_terminaux().

8.26.1.75 ECRIRE

```
#define ECRIRE 30
```

Référencé par initialise_premiers_terminaux().

8.26.1.76 INCR

```
#define INCR 31
```

Référencé par initialise_premiers_terminaux().

8.26.1.77 ID_VAR

```
#define ID_VAR 32
```

Référencé par initialise_premiers_terminaux().

8.26.1.78 ID_FCT

```
#define ID_FCT 33
```

Référencé par initialise_premiers_terminaux().

8.26.1.79 NOMBRE

```
#define NOMBRE 34
```

Référencé par initialise_premiers_terminaux().

8.26.1.80 FIN

```
#define FIN 35
```

Référencé par initialise_suivants_terminaux(), et yylex().

8.26.1.81 VIRGULE

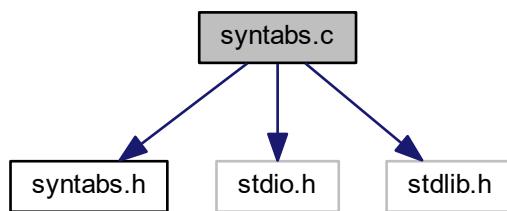
```
#define VIRGULE 36
```

Référencé par initialise_premiers_terminaux().

8.27 Référence du fichier syntabs.c

```
#include "syntabs.h"
#include <stdio.h>
#include <stdlib.h>
```

Graphe des dépendances par inclusion de syntabs.c :



Fonctions

```
— n_appel * cree_n_appel (char *fonction, n_l_exp *args)
— n_prog * cree_n_prog (n_l_dec *variables, n_l_dec *fonctions)
— n_var * cree_n_var_simple (char *nom)
— n_var * cree_n_var_indicee (char *nom, n_exp *indice)
— n_exp * cree_n_exp_op (operation op, n_exp *op1, n_exp *op2)
— n_exp * cree_n_exp_appel (n_appel *app)
— n_exp * cree_n_exp_var (n_var *var)
— n_exp * cree_n_exp_entier (int entier)
— n_exp * cree_n_exp_lire ()
— n_l_exp * cree_n_l_exp (n_exp *tete, n_l_exp *queue)
— n_instr * cree_n_instr_incr (n_var *incr)
— n_instr * cree_n_instr_si (n_exp *test, n_instr *alors, n_instr *sinon)
— n_instr * cree_n_instr_tantque (n_exp *test, n_instr *faire)
— n_instr * cree_n_instr_faire (n_instr *faire, n_exp *test)
— n_instr * cree_n_instr_affect (n_var *var, n_exp *exp)
— n_l_instr * cree_n_l_instr (n_instr *tete, n_l_instr *queue)
— n_instr * cree_n_instr_bloc (n_l_instr *liste)
— n_instr * cree_n_instr_appel (n_appel *app)
— n_instr * cree_n_instr_ecrire (n_exp *expression)
— n_instr * cree_n_instr_retour (n_exp *expression)
— n_instr * cree_n_instr_vide (void)
— n_dec * cree_n_dec_var (char *nom)
— n_dec * cree_n_dec_tab (char *nom, int taille)
— n_dec * cree_n_dec_fonc (char *nom, n_l_dec *param, n_l_dec *variables, n_instr *corps)
— n_l_dec * cree_n_l_dec (n_dec *tete, n_l_dec *queue)
```

8.27.1 Documentation des fonctions

8.27.1.1 **cree_n_appel()**

```
n_appel* cree_n_appel (
    char * fonction,
    n_l_exp * args )
```

8.27.1.2 **cree_n_prog()**

```
n_prog* cree_n_prog (
    n_l_dec * variables,
    n_l_dec * fonctions )
```

8.27.1.3 **cree_n_var_simple()**

```
n_var* cree_n_var_simple (
    char * nom )
```

8.27.1.4 **cree_n_var_indicee()**

```
n_var* cree_n_var_indicee (
    char * nom,
    n_exp * indice )
```

8.27.1.5 `cree_n_exp_op()`

```
n_exp* cree_n_exp_op (
    operation op,
    n_exp * op1,
    n_exp * op2 )
```

8.27.1.6 `cree_n_exp_appel()`

```
n_exp* cree_n_exp_appel (
    n_appel * app )
```

8.27.1.7 `cree_n_exp_var()`

```
n_exp* cree_n_exp_var (
    n_var * var )
```

Références `var()`.

Voici le graphe d'appel pour cette fonction :



8.27.1.8 `cree_n_exp_entier()`

```
n_exp* cree_n_exp_entier (
    int entier )
```

8.27.1.9 `cree_n_exp_lire()`

```
n_exp* cree_n_exp_lire (
    void )
```

8.27.1.10 `cree_n_l_exp()`

```
n_l_exp* cree_n_l_exp (
    n_exp * tete,
    n_l_exp * queue )
```

8.27.1.11 `cree_n_instr_incr()`

```
n_instr* cree_n_instr_incr (
    n_var * incr )
```

8.27.1.12 `cree_n_instr_si()`

```
n_instr* cree_n_instr_si (
    n_exp * test,
    n_instr * alors,
    n_instr * sinon )
```

8.27.1.13 `cree_n_instr_tantque()`

```
n_instr* cree_n_instr_tantque (
    n_exp * test,
    n_instr * faire )
```

8.27.1.14 `cree_n_instr_faire()`

```
n_instr* cree_n_instr_faire (
    n_instr * faire,
    n_exp * test )
```

8.27.1.15 `cree_n_instr_affect()`

```
n_instr* cree_n_instr_affect (
    n_var * var,
    n_exp * exp )
```

Références `var()`.

Voici le graphe d'appel pour cette fonction :

8.27.1.16 `cree_n_l_instr()`

```
n_l_instr* cree_n_l_instr (
    n_instr * tete,
    n_l_instr * queue )
```

8.27.1.17 cree_n_instr_bloc()

```
n_instr* cree_n_instr_bloc (
    n_l_instr * liste )
```

8.27.1.18 cree_n_instr_appel()

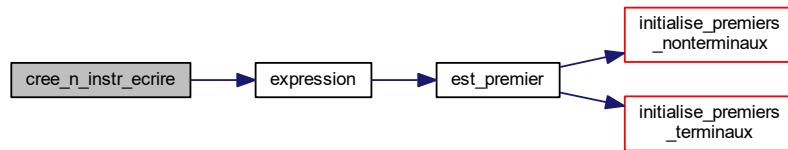
```
n_instr* cree_n_instr_appel (
    n_appel * app )
```

8.27.1.19 cree_n_instr_ecrire()

```
n_instr* cree_n_instr_ecrire (
    n_exp * expression )
```

Références `expression()`.

Voici le graphe d'appel pour cette fonction :

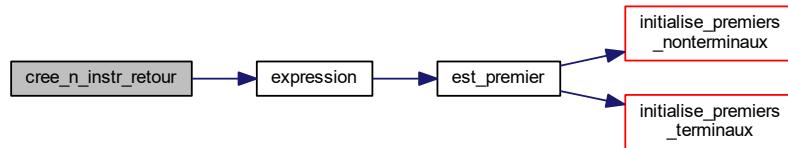


8.27.1.20 cree_n_instr_retour()

```
n_instr* cree_n_instr_retour (
    n_exp * expression )
```

Références `expression()`.

Voici le graphe d'appel pour cette fonction :



8.27.1.21 cree_n_instr_vide()

```
n_instr* cree_n_instr_vide (
    void )
```

8.27.1.22 cree_n_dec_var()

```
n_dec* cree_n_dec_var (
    char * nom )
```

8.27.1.23 cree_n_dec_tab()

```
n_dec* cree_n_dec_tab (
    char * nom,
    int taille )
```

8.27.1.24 cree_n_dec_fonc()

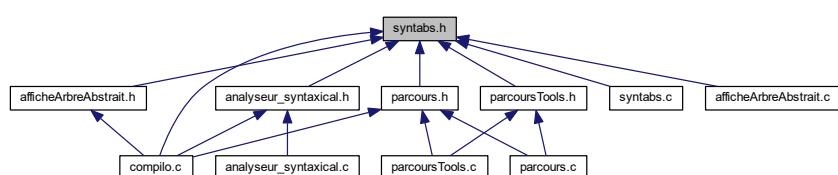
```
n_dec* cree_n_dec_fonc (
    char * nom,
    n_l_dec * param,
    n_l_dec * variables,
    n_instr * corps )
```

8.27.1.25 cree_n_l_dec()

```
n_l_dec* cree_n_l_dec (
    n_dec * tete,
    n_l_dec * queue )
```

8.28 Référence du fichier syntabs.h

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

```
— struct n_prog
— struct n_dec
— struct n_exp
— struct n_instr
— struct n_appel
— struct n_var
— struct n_l_exp
— struct n_l_instr
— struct n_l_dec
```

Énumérations

```
— enum operation {
    plus, moins, fois, divise,
    modulo, egal, diff, inf,
    sup, infeg, supeg, ou,
    et, non, negatif }
```

Fonctions

```
— n_prog * cree_n_prog (n_l_dec *variables, n_l_dec *fonctions)
— n_dec * cree_n_dec_var (char *nom)
— n_dec * cree_n_dec_tab (char *nom, int taille)
— n_dec * cree_n_dec_fonc (char *nom, n_l_dec *param, n_l_dec *variables, n_instr *corps)
— n_exp * cree_n_exp_op (operation type, n_exp *op1, n_exp *op2)
— n_exp * cree_n_exp_entier (int entier)
— n_exp * cree_n_exp_var (n_var *var)
— n_exp * cree_n_exp_appel (n_appel *app)
— n_exp * cree_n_exp_lire (void)
— n_instr * cree_n_instr_incr (n_var *incr)
— n_instr * cree_n_instr_si (n_exp *test, n_instr *alors, n_instr *sinon)
— n_instr * cree_n_instr_bloc (n_l_instr *liste)
— n_instr * cree_n_instr_tantque (n_exp *test, n_instr *faire)
— n_instr * cree_n_instr faire (n_instr *faire, n_exp *test)
— n_instr * cree_n_instr_affect (n_var *var, n_exp *exp)
— n_instr * cree_n_instr_appel (n_appel *appel)
— n_instr * cree_n_instr_retour (n_exp *expression)
— n_instr * cree_n_instr_ecrire (n_exp *expression)
— n_instr * cree_n_instr_vide (void)
— n_appel * cree_n_appel (char *fonction, n_l_exp *args)
— n_var * cree_n_var_simple (char *nom)
— n_var * cree_n_var_indicee (char *nom, n_exp *indice)
— n_l_exp * cree_n_l_exp (n_exp *tete, n_l_exp *queue)
— n_l_instr * cree_n_l_instr (n_instr *tete, n_l_instr *queue)
— n_l_dec * cree_n_l_dec (n_dec *tete, n_l_dec *queue)
```

8.28.1 Documentation du type de l'énumération

8.28.1.1 operation

```
enum operation
```

Valeurs énumérées

plus	
moins	
fois	
divise	
modulo	

Valeurs énumérées

egal	
diff	
inf	
sup	
infeg	
supeg	
ou	
et	
non	
negatif	

8.28.2 Documentation des fonctions

8.28.2.1 cree_n_prog()

```
n_prog* cree_n_prog (
    n_l_dec * variables,
    n_l_dec * fonctions )
```

8.28.2.2 cree_n_dec_var()

```
n_dec* cree_n_dec_var (
    char * nom )
```

8.28.2.3 cree_n_dec_tab()

```
n_dec* cree_n_dec_tab (
    char * nom,
    int taille )
```

8.28.2.4 cree_n_dec_fonc()

```
n_dec* cree_n_dec_fonc (
    char * nom,
    n_l_dec * param,
    n_l_dec * variables,
    n_instr * corps )
```

8.28.2.5 cree_n_exp_op()

```
n_exp* cree_n_exp_op (
    operation type,
    n_exp * op1,
    n_exp * op2 )
```

8.28.2.6 cree_n_exp_entier()

```
n_exp* cree_n_exp_entier (
    int entier )
```

8.28.2.7 cree_n_exp_var()

```
n_exp* cree_n_exp_var (
    n_var * var )
```

Références var().

Voici le graphe d'appel pour cette fonction :



8.28.2.8 cree_n_exp_appel()

```
n_exp* cree_n_exp_appel (
    n_appel * app )
```

8.28.2.9 cree_n_exp_lire()

```
n_exp* cree_n_exp_lire (
    void )
```

8.28.2.10 cree_n_instr_incr()

```
n_instr* cree_n_instr_incr (
    n_var * incr )
```

8.28.2.11 cree_n_instr_si()

```
n_instr* cree_n_instr_si (
    n_exp * test,
    n_instr * alors,
    n_instr * sinon )
```

8.28.2.12 cree_n_instr_bloc()

```
n_instr* cree_n_instr_bloc (
    n_l_instr * liste )
```

8.28.2.13 cree_n_instr_tantque()

```
n_instr* cree_n_instr_tantque (
    n_exp * test,
    n_instr * faire )
```

8.28.2.14 cree_n_instr_faire()

```
n_instr* cree_n_instr_faire (
    n_instr * faire,
    n_exp * test )
```

8.28.2.15 cree_n_instr_affect()

```
n_instr* cree_n_instr_affect (
    n_var * var,
    n_exp * exp )
```

Références var().

Voici le graphe d'appel pour cette fonction :

**8.28.2.16 cree_n_instr_appel()**

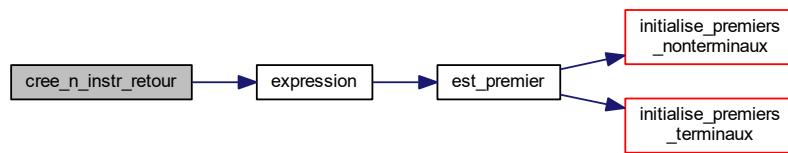
```
n_instr* cree_n_instr_appel (
    n_appel * appel )
```

8.28.2.17 cree_n_instr_retour()

```
n_instr* cree_n_instr_retour (
    n_exp * expression )
```

Références `expression()`.

Voici le graphe d'appel pour cette fonction :

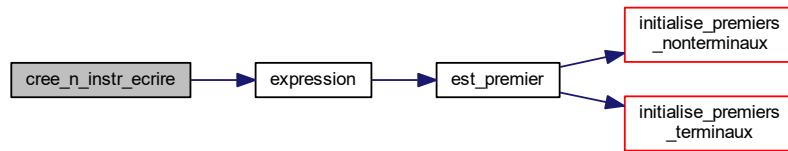


8.28.2.18 cree_n_instr_ecrire()

```
n_instr* cree_n_instr_ecrire (
    n_exp * expression )
```

Références `expression()`.

Voici le graphe d'appel pour cette fonction :



8.28.2.19 cree_n_instr_vide()

```
n_instr* cree_n_instr_vide (
    void )
```

8.28.2.20 cree_n_appel()

```
n_appel* cree_n_appel (
    char * fonction,
    n_l_exp * args )
```

8.28.2.21 cree_n_var_simple()

```
n_var* cree_n_var_simple (
    char * nom )
```

8.28.2.22 cree_n_var_indicee()

```
n_var* cree_n_var_indicee (
    char * nom,
    n_exp * indice )
```

8.28.2.23 cree_n_l_exp()

```
n_l_exp* cree_n_l_exp (
    n_exp * tete,
    n_l_exp * queue )
```

8.28.2.24 cree_n_l_instr()

```
n_l_instr* cree_n_l_instr (
    n_instr * tete,
    n_l_instr * queue )
```

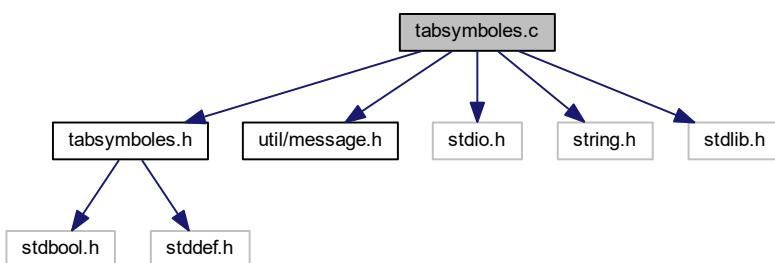
8.28.2.25 cree_n_l_dec()

```
n_l_dec* cree_n_l_dec (
    n_dec * tete,
    n_l_dec * queue )
```

8.29 Référence du fichier tabsymboles.c

```
#include "tabsymboles.h"
#include "util/message.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

Graphe des dépendances par inclusion de tabsymboles.c :



Structures de données

- struct **tabsymboles_**
tabsymboles : Table des symboles (globale ET locale)

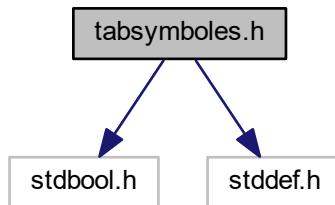
Fonctions

- void **tabsymb_setAffiche** (bool b)
- void **tabsymb_entreeFonction** (void)
tabsymb_entreeFonction : Fonction qui bascule table globale -> table locale. À appeler lors qu'on parcourt une déclaration de fonction, juste avant la liste d'arguments.
- void **tabsymb_sortieFonction** (void)
tabsymb_sortieFonction : Fonction qui bascule table locale -> table globale. À appeler lors qu'on a fini de parcourir une déclaration de fonction, après le bloc d'instructions qui constitue le corps de la fonction
- int **tabsymb_ajouteIdentificateur** (char *identif, int portee, int type, size_t adresse, size_t complement)
tabsymb_ajouteIdentificateur : Ajoute un nouvel identificateur à la table des symboles courante.
- int **tabsymb_rechercheExecutable** (const char *identif)
tabsymb_rechercheExecutable : Recherche si un identificateur est présent dans la table LOCALE ou GLOBALE Cette fonction doit être utilisée pour s'assurer que tout identificateur utilisé a été déclaré auparavant. Elle doit être utilisée lors de l'UTILISATION d'un identificateur, soit dans un appel à fonction, une partie gauche d'affectation ou une variable dans une expression. Si deux identificateurs ont le même nom dans des portées différentes (un global et un local), la fonction renvoie le plus proche, c-à-d le local.
- int **tabsymb_rechercheDeclarative** (const char *identif)
tabsymb_rechercheDeclarative : Recherche si un identificateur est présent dans la table LOCALE Cette fonction doit être utilisée pour s'assurer qu'il n'y a pas deux identificateurs avec le même lors d'une DÉCLARATION d'un identificateur.
- const **desc_identif * tabsymb_getSymbole** (const char *nom)
tabsymb_getSymbole : Trouve le premier executable, voir tabsymb_rechercheExecutable pour plus d'information.
- const **desc_identif * tabsymb_getSymboleFonctionCourante** ()
tabsymb_getSymboleFonctionCourante : Trouve la fonction courante
- void **tabsymb_dump** (void)
tabsymb_dump : Fonction auxiliaire qui permet d'afficher le contenu actuel de la table des symboles.

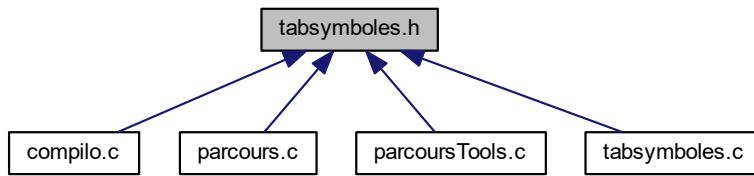
8.30 Référence du fichier tabsymboles.h

```
#include <stdbool.h>
#include <stddef.h>
```

Graphe des dépendances par inclusion de tabsymboles.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Structures de données

- struct `desc_identif`

Macros

- #define `MAX_IDENTIF` 1000
- #define `P_VARIABLE_GLOBALE` 1
- #define `P_VARIABLE_LOCALE` 2
- #define `P_ARGUMENT` 3
- #define `T_ENTIER` 1
- #define `T_TABLEAU_ENTIER` 2
- #define `T_FONCTION` 3

Variables

- int `adresseGlobalCourant`
- int `adresseLocaleCourante`
- int `adresseArgumentCourant`

Chapitre 9

Documentation des exemples

9.1 une

permet d'afficher les entrées en fonction de leur nom et sous instructions.. sous boucle d'une fonction main aura les instructions : main : ... [au lieu de] main : main0_TqT : ... e1 : ... main1_TqV : ... e2 : ... main2_Tql : ... e2 : ...

Note

se justifie esthétiquement...

Index

/Remplissage des premiers, 155
 initialise_premiers_nonterminaux, 158
 initialise_premiers_terminaux, 157
 initialise_suivants_nonterminaux, 163
 initialise_suivants_terminaux, 162
 premier_union, 156
 premier_union_suivant, 161
 premier_valide, 155
 suivant_union, 161
 suivant_valide, 160
appelFct
 symboles.h, 258
argumentsEffectifs
 symboles.h, 259
comparaisonBis
 symboles.h, 260
comparaison
 symboles.h, 258
conjonctionBis
 symboles.h, 259
conjonction
 symboles.h, 258
declarationFonction
 symboles.h, 256
declarationVariable
 symboles.h, 256
expArithBis
 symboles.h, 260
expArith
 symboles.h, 258
expressionBis
 symboles.h, 261
expression
 symboles.h, 258
facteur
 symboles.h, 259
instructionAffect
 symboles.h, 256
instructionAppel
 symboles.h, 257
instructionBloc
 symboles.h, 256
instructionEcriture
 symboles.h, 257
instructionFaire
 symboles.h, 261
instructionIncrementer
 symboles.h, 257
instructionPour
 symboles.h, 260
instructionRetour
 symboles.h, 257
instructionSi
 symboles.h, 257
instructionTantque
 symboles.h, 257
instructionVide
 symboles.h, 257
instruction
 symboles.h, 256
listeDecFonctions
 symboles.h, 255
listeDecVariablesBis
 symboles.h, 260
listeDecVariables
 symboles.h, 255
listeExpressionsBis
 symboles.h, 259
listeExpressions
 symboles.h, 259
listInstructions
 symboles.h, 256
listParam
 symboles.h, 256
negation
 symboles.h, 258
optDecVariables
 symboles.h, 260
optIndice
 symboles.h, 260
optListeDecVariables
 symboles.h, 261
optSinon
 symboles.h, 260
optTailleTableau
 symboles.h, 259
programme
 symboles.h, 261
termeBis
 symboles.h, 260
terme
 symboles.h, 259
var
 symboles.h, 258
Énumération, 17
 byte, 18
 byte_flag, 18
 db, 17

dd, 17
 dq, 17
 dt, 17
 dw, 17
 dword, 18
 pseudo_instruction_bss, 17
 pseudo_instruction_data, 17
 resb, 18
 resd, 18
 resq, 18
 rest, 18
 resw, 18
 word, 18

ACCOLADE_FERMANTE
 symboles.h, 263

ACCOLADE_OUVRANTE
 symboles.h, 263

ADD_ENTRY
 Fonction d'instruction, 39

ALORS
 symboles.h, 264

AdjList, 165
 isEnd, 166
 next, 166
 prev, 166

adresse
 desc_identif, 170

adresseArgumentCourant
 Gestion de la table des symboles, 111
 Variable de gestion de la table des symboles, 58

adresseGlobalCourant
 Gestion de la table des symboles, 111
 Variable de gestion de la table des symboles, 58

adresseLocaleCourante
 Gestion de la table des symboles, 111
 Variable de gestion de la table des symboles, 58

affecte_
 n_instr_, 180

afficheArbreAbstrait.c, 189
 afficheArbreAbstrait_appel, 197
 afficheArbreAbstrait_appelExp, 205
 afficheArbreAbstrait_dec, 207
 afficheArbreAbstrait_exp, 201
 afficheArbreAbstrait_foncDec, 208
 afficheArbreAbstrait_instr, 191
 afficheArbreAbstrait_instr_affect, 195
 afficheArbreAbstrait_instr_appel, 196
 afficheArbreAbstrait_instr_ecrire, 199
 afficheArbreAbstrait_instr_faire, 194
 afficheArbreAbstrait_instr_retour, 198
 afficheArbreAbstrait_instr_si, 192
 afficheArbreAbstrait_instr_tantque, 193
 afficheArbreAbstrait_intExp, 204
 afficheArbreAbstrait_l_dec, 206
 afficheArbreAbstrait_l_exp, 200
 afficheArbreAbstrait_l_instr, 190
 afficheArbreAbstrait_lireExp, 205
 afficheArbreAbstrait_n_prog, 190

afficheArbreAbstrait_opExp, 203
 afficheArbreAbstrait_tabDec, 210
 afficheArbreAbstrait_var, 210
 afficheArbreAbstrait_var_indicee, 212
 afficheArbreAbstrait_var_simple, 211
 afficheArbreAbstrait_varDec, 209
 afficheArbreAbstrait_varExp, 202

afficheArbreAbstrait.h, 213
 afficheArbreAbstrait_appel
 afficheArbreAbstrait.c, 197

afficheArbreAbstrait_appelExp
 afficheArbreAbstrait.c, 205

afficheArbreAbstrait_dec
 afficheArbreAbstrait.c, 207

afficheArbreAbstrait_exp
 afficheArbreAbstrait.c, 201

afficheArbreAbstrait_foncDec
 afficheArbreAbstrait.c, 208

afficheArbreAbstrait_instr
 afficheArbreAbstrait.c, 191

afficheArbreAbstrait_instr_affect
 afficheArbreAbstrait.c, 195

afficheArbreAbstrait_instr_appel
 afficheArbreAbstrait.c, 196

afficheArbreAbstrait_instr_ecrire
 afficheArbreAbstrait.c, 199

afficheArbreAbstrait_instr_faire
 afficheArbreAbstrait.c, 194

afficheArbreAbstrait_instr_retour
 afficheArbreAbstrait.c, 198

afficheArbreAbstrait_instr_si
 afficheArbreAbstrait.c, 192

afficheArbreAbstrait_instr_tantque
 afficheArbreAbstrait.c, 193

afficheArbreAbstrait_intExp
 afficheArbreAbstrait.c, 204

afficheArbreAbstrait_l_dec
 afficheArbreAbstrait.c, 206

afficheArbreAbstrait_l_exp
 afficheArbreAbstrait.c, 200

afficheArbreAbstrait_l_instr
 afficheArbreAbstrait.c, 190

afficheArbreAbstrait_lireExp
 afficheArbreAbstrait.c, 205

afficheArbreAbstrait_n_prog
 afficheArbreAbstrait.c, 190

afficheArbreAbstrait_opExp
 afficheArbreAbstrait.c, 203

afficheArbreAbstrait_tabDec
 afficheArbreAbstrait.c, 210

afficheArbreAbstrait_var
 afficheArbreAbstrait.c, 210

afficheArbreAbstrait_var_indicee
 afficheArbreAbstrait.c, 212

afficheArbreAbstrait_var_simple
 afficheArbreAbstrait.c, 211

afficheArbreAbstrait_varDec
 afficheArbreAbstrait.c, 209

afficheArbreAbstrait_varExp
afficheArbreAbstrait.c, 202

afficheXml.c, 213

- afficheXml_balise_fermante, 216
- afficheXml_balise_ouvrante, 215
- afficheXml_element, 218
- afficheXml_setOutput, 214
- afficheXml_texte, 217
- ErreursSiAucuneSortie, 214
- indent, 214
- special_texte, 217

afficheXml.h, 219

afficheXml_balise_fermante

- afficheXml.c, 216

afficheXml_balise_ouvrante

- afficheXml.c, 215

afficheXml_element

- afficheXml.c, 218

afficheXml_setOutput

- afficheXml.c, 214

afficheXml_texte

- afficheXml.c, 217

afficherAideF

- Compilateur, 12

afficherLexique

- Script d'execution, 133

alors

- n_instr_, 179

Analyse, 140

Analyseur lexical, 120

analyseur_lexical.c, 219

- ELIF_TEST_TOKEN, 220

analyseur_lexical.h, 221

analyseur_syntactical.c, 221

analyseur_syntactical.h, 223

Analyse Syntaxique, 134

- syntactical, 135
- syntactical_setAffiche, 135

appel

- n_exp_, 176
- n_instr_, 179

appelFct

- Expression, 153

Aquisition, 101

- ce, 103
- parcours_exp, 101
- parcoursTools_accesVar, 102

args

- n_appel_, 171

ArgsSecond

- Compilateur, 11

assembleur.c, 223

assembleur.h, 226

assembleur_add_bss

- Fonction de gestion pour l'assembleur, 32

assembleur_add_bss_andComment

- Fonction de gestion pour l'assembleur, 30

assembleur_add_commentary

Fonction et variable pour la gestion des listes d'adjacence, 20

assembleur_add_data

- Fonction de gestion pour l'assembleur, 32

assembleur_add_data_andComment

- Fonction de gestion pour l'assembleur, 30

assembleur_add_entry

- Fonction de gestion pour l'assembleur, 32

assembleur_add_entry_andComment

- Fonction de gestion pour l'assembleur, 31

assembleur_add_entryName

- Fonction de gestion pour l'assembleur, 33

assembleur_add_entryName_andComment

- Fonction de gestion pour l'assembleur, 31

assembleur_bss, 166

- etiquette, 167
- nb, 167
- pi, 167

assembleur_commentary_next

- Fonction et variable pour la gestion des listes d'adjacence, 21

assembleur_data, 167

- etiquette, 168
- pi, 168
- valeur, 168

assembleur_dump

- Fonction de génération du flux de sortie, 37

assembleur_dump_section_bss

- Fonction de génération du flux de sortie, 36

assembleur_dump_section_data

- Fonction de génération du flux de sortie, 35

assembleur_dump_section_entry

- Fonction de génération du flux de sortie, 36

assembleur_dump_section_entry_instrs

- Fonction de génération du flux de sortie, 36

assembleur_dump_section_import

- Fonction de génération du flux de sortie, 35

assembleur_dump_section_oneEntry

- Fonction de génération du flux de sortie, 35

assembleur_entry, 168

- counter, 169
- instrs, 169
- name, 169

assembleur_entry_0s

- Fonction d'instruction, 45

assembleur_entry_0s_andComment

- Fonction d'instruction, 40

assembleur_entry_1s

- Fonction d'instruction, 45

assembleur_entry_1s_andComment

- Fonction d'instruction, 41

assembleur_entry_2s

- Fonction d'instruction, 46

assembleur_entry_2s_andComment

- Fonction d'instruction, 42

assembleur_entry_addEntryPoint

- Fonction d'instruction, 47

assembleur_entry_addEntryPoint_andComment

Fonction d'instruction, 44
assembleur_entry_add_subEntry
 Fonction d'instruction, 47
assembleur_entry_add_subEntry_andComment
 Fonction d'instruction, 43
assembleur_entry_getLastInstr
 Fonction d'instruction, 48
assembleur_entry_getName
 Fonction d'instruction, 48
assembleur_entry_instr_circularAdjList
 Gestion des données, 16
assembleur_entry_make_next
 Conversion en str, 53
assembleur_entry_make_prefix
 Conversion en str, 52
assembleur_entry_make_suffix
 Conversion en str, 53
assembleur_entry_subEntryName
 Fonction d'instruction, 47
assembleur_entry_subEntryName_andComment
 Fonction d'instruction, 44
assembleur_make_concat
 Conversion en str, 52
assembleur_make_constant
 Conversion en str, 51
assembleur_make_globalVar
 Conversion en str, 50
assembleur_make_idSuffix
 Conversion en str, 51
assembleur_make_localVar
 Conversion en str, 50
assembleur_make_tabVar
 Conversion en str, 49
assembleur_new_entryName
 Fonction de gestion pour l'assembleur, 30

base
 tabsymboles_, 187
Bibliothèque, 104
 parcoursTools_dump_asm_lib, 104
Bibliothèque comparaisonBool, 108
 LIB_COMPARISONBOOL_PREFIX, 108
 LIB_COMPARISONBOOL_SIZE, 108
 parcoursTools_genLib_comparaisonBool, 109
 parcoursTools_lib_comparaisonBool, 109
 parcoursTools_lib_comparaisonBool_getName, 109
Bibliothèque comparatorExpr, 106
 LIB_COMPARATOREXPR_PREFIX, 106
 LIB_COMPARATOREXPR_SIZE, 106
 parcoursTools_genLib_comparatorExpr, 107
 parcoursTools_lib_comparatorExpr, 107
byte
 Énumération, 18
byte_flag
 Énumération, 18

CHECK_ENTRY
 Fonction d'instruction, 39

CROCHET_FERMANT
 symboles.h, 262
CROCHET_OUVRANT
 symboles.h, 262
ce
 Aquisition, 103
 Variable de gestion de la table des symboles, 58
circularAdjList.c, 227
 circularAdjList_add_, 227
 circularAdjList_free_, 229
 circularAdjList_search_, 228
circularAdjList.h, 230
 circularAdjList_add, 231
 circularAdjList_add_, 234
 circularAdjList_doWhileNext, 233
 circularAdjList_free, 232
 circularAdjList_free_, 235
 circularAdjList_isEnd, 233
 circularAdjList_search, 231
 circularAdjList_search_, 235
 circularAdjListComparator_header, 232
 circularAdjListComparator_pf, 234
 circularAdjListCopier_header, 232
 circularAdjListCopier_pf, 234
 circularAdjListFreer_header, 233
 circularAdjListFreer_pf, 234
 circularAdjListStaticDecl_struct, 232
 circularAdjListStruct_content, 232
 circularAdjListStruct_header, 232
 circularAdjListTools_contentCast, 233
 circularAdjListTools_headCast, 233
circularAdjList_add
 circularAdjList.h, 231
circularAdjList_add_
 circularAdjList.c, 227
 circularAdjList.h, 234
circularAdjList_doWhileNext
 circularAdjList.h, 233
circularAdjList_free
 circularAdjList.h, 232
circularAdjList_free_
 circularAdjList.c, 229
 circularAdjList.h, 235
circularAdjList_isEnd
 circularAdjList.h, 233
circularAdjList_search
 circularAdjList.h, 231
circularAdjList_search_
 circularAdjList.c, 228
 circularAdjList.h, 235
circularAdjListComparator_header
 circularAdjList.h, 232
 Fonction et variable pour la gestion des listes d'adjacence, 22–24
circularAdjListComparator_pf
 circularAdjList.h, 234
circularAdjListCopier_header
 circularAdjList.h, 232

Fonction et variable pour la gestion des listes d'adjacence, 21–25
circularAdjListCopier_pf
 circularAdjList.h, 234
circularAdjListFreer_header
 circularAdjList.h, 233
circularAdjListFreer_pf
 circularAdjList.h, 234
circularAdjListStaticDecl_struct
 circularAdjList.h, 232
 Fonction et variable pour la gestion des listes d'adjacence, 22–24
circularAdjListStruct_content
 circularAdjList.h, 232
circularAdjListStruct_header
 circularAdjList.h, 232
 Fonction et variable pour la gestion des listes d'adjacence, 21, 24
circularAdjListTools_contentCast
 circularAdjList.h, 233
circularAdjListTools_headCast
 circularAdjList.h, 233
comparaison
 Expression, 150
comparaisonBis
 Expression, 151
Compilateur, 11
 afficherAideF, 12
 ArgsSecond, 11
 main, 12
 yyin, 13
 yytext, 13
compilo.c, 236
complement
 desc_identif, 170
Configuration du module Assembleur, 56
 GENASM_ENTRY_ARE_INDENT, 56
 GENASM_ENTRY_PRETTY_NAME, 56
 GENASM_ENTRYCOUNTER_SEPERATE_BY←
 _UNDERSCORE, 56
 GENASM_ENTRYPOINT_SEPERATE_RETUR←
 NLIN, 56
Configuration du module parcours, 93
 HAVE_LOGIQUE_COMPARAISSON, 93
 IO_USE_READLINE_ENTRY, 93
 PARCOURS_STARTENTRY_ACCEPT_RETVA←
 LUE, 93
 PARCOURS_VAR_USE_DWORD, 93
conjonction
 Expression, 150
conjonctionBis
 Expression, 150
Conversion en str, 49
 assembleur_entry_make_next, 53
 assembleur_entry_make_prefix, 52
 assembleur_entry_make_suffix, 53
 assembleur_make_concat, 52
 assembleur_make_constant, 51
 assembleur_make_globalVar, 50
 assembleur_make_idSuffix, 51
 assembleur_make_localVar, 50
 assembleur_make_tabVar, 49
 GENASM_ENTRYCOUNTER_UNDERSCORE←
 SIZE, 49
corps
 n_dec_, 173
counter
 assembleur_entry, 169
cree_n_appel
 syntabs.c, 267
 syntabs.h, 276
cree_n_dec_fonc
 syntabs.c, 271
 syntabs.h, 273
cree_n_dec_tab
 syntabs.c, 271
 syntabs.h, 273
cree_n_dec_var
 syntabs.c, 271
 syntabs.h, 273
cree_n_exp_appel
 syntabs.c, 268
 syntabs.h, 274
cree_n_exp_entier
 syntabs.c, 268
 syntabs.h, 273
cree_n_exp_lire
 syntabs.c, 268
 syntabs.h, 274
cree_n_exp_op
 syntabs.c, 267
 syntabs.h, 273
cree_n_exp_var
 syntabs.c, 268
 syntabs.h, 274
cree_n_instr_affect
 syntabs.c, 269
 syntabs.h, 275
cree_n_instr_appel
 syntabs.c, 270
 syntabs.h, 275
cree_n_instr_bloc
 syntabs.c, 269
 syntabs.h, 274
cree_n_instr_ecrire
 syntabs.c, 270
 syntabs.h, 276
cree_n_instr_faire
 syntabs.c, 269
 syntabs.h, 275
cree_n_instr_incr
 syntabs.c, 268
 syntabs.h, 274
cree_n_instr_retour
 syntabs.c, 270
 syntabs.h, 275

cree_n_instr_si
 syntabs.c, 269
 syntabs.h, 274

cree_n_instr_tantque
 syntabs.c, 269
 syntabs.h, 275

cree_n_instr_vide
 syntabs.c, 270
 syntabs.h, 276

cree_n_l_dec
 syntabs.c, 271
 syntabs.h, 277

cree_n_l_exp
 syntabs.c, 268
 syntabs.h, 277

cree_n_l_instr
 syntabs.c, 269
 syntabs.h, 277

cree_n_prog
 syntabs.c, 267
 syntabs.h, 273

cree_n_var_indicee
 syntabs.c, 267
 syntabs.h, 277

cree_n_var_simple
 syntabs.c, 267
 syntabs.h, 276

DIFFERENT
 symboles.h, 263

DIVISE
 symboles.h, 262

db
 Énumération, 17

dd
 Énumération, 17

Declaration, 66
 parcours_dec, 68
 parcours_dec_fonctions, 69
 parcours_prefixe_l_dec, 66
 parcours_suffixe_l_dec, 67

declarationFonction
 Entrée, Variable globale, Fonction, Variable, 143

declarationVariable
 Entrée, Variable globale, Fonction, Variable, 143

Define des fonctions, 136
 Entre, 136
 Erreur, 136
 Sortie, 136

Definition des portées, 118
 P_ARGUMENT, 118
 P_VARIABLE_GLOBALE, 118
 P_VARIABLE_LOCALE, 118

Definition des types, 119
 T_ENTIER, 119
 T_FONCTION, 119
 T_TABLEAU_ENTIER, 119

Definition pour la Gestion, 117
 MAX_IDENTIFIER, 117

Definition pour la gestion du parcours, 60
 ErreurSiMauvaisType, 60
 ErreurSiNulle, 60

delireCar
 Fonctions du parser yylex, 130

desc_identif, 169
 adresse, 170
 complement, 170
 identif, 170
 portee, 170
 type, 170

dq
 Énumération, 17

dt
 Énumération, 17

dump_symb.c, 237
 main, 238

dw
 Énumération, 17

dword
 Énumération, 18

ECRIRE
 symboles.h, 265

EGAL
 symboles.h, 263

ELIF_TEST_TOKEN
 analyseur_lexical.c, 220

ENTIER
 symboles.h, 265

EPSILON
 symboles.h, 255

ecrire_
 n_instr_, 180

endEntry
 Variable de gestion de la table des symboles, 59

entier
 n_exp_, 176

Entrée, Variable globale, Fonction, Variable, 141
 declarationFonction, 143
 declarationVariable, 143
 listeDecFonctions, 143
 listeDecVariables, 142
 listeDecVariablesBis, 142
 listeParam, 143
 optDecVariables, 142
 optListeDecVariables, 143
 optTailleTableau, 143
 programme, 141

Entrées principales, 62
 parcours_init_asm, 63
 parcours_prog, 62

Entre
 Define des fonctions, 136

Erreur
 Define des fonctions, 136

erreur
 message.c, 239

erreur_1s

message.c, 240
erreurArgs
 message.c, 239
erreurLexical
 Fonctions du parser yylex, 129
erreurLigne
 message.c, 239
ErreurSiAucuneSortie
 afficheXml.c, 214
ErreurSiMauvaisType
 Definition pour la gestion du parcours, 60
ErreurSiNulle
 Definition pour la gestion du parcours, 60
erreurSyntaxical
 Gestion erreur et yylex, 139
est_premier
 premiers.c, 246
est_suivant
 suivants.c, 251
estUnNombre
 Fonctions du parser yylex, 131
estUneFonction
 Fonctions du parser yylex, 131
estUneVariable
 Fonctions du parser yylex, 131
ET
 symboles.h, 264
etiquette
 assembleur_bss, 167
 assembleur_data, 168
exp
 n_instr_, 180
expArith
 Expression, 151
expArithBis
 Expression, 151
Expression, 83, 95, 149
 appelFct, 153
 comparaison, 150
 comparaisonBis, 151
 conjonction, 150
 conjonctionBis, 150
 expArith, 151
 expArithBis, 151
 expression, 149
 expressionBis, 150
 facteur, 152
 listeExpressions, 153
 listeExpressionsBis, 153
 negation, 152
 optIndice, 153
 parcours_exp, 85
 parcours_exp_appel, 90
 parcours_exp_comparaison, 86
 parcours_exp_int, 89
 parcours_exp_lire, 87
 parcours_exp_op, 91
 parcours_exp_var, 88
parcours_prefixe_l_exp, 83
parcours_suffixe_l_exp, 84
parcoursTools_comparaisonOperation, 98
parcoursTools_computeOperation, 97
parcoursTools_constexpr, 96
parcoursTools_constexpr_operation, 95
terme, 151
termeBis, 152
var, 152
expression
 Expression, 149
 n_instr_, 180
expressionBis
 Expression, 150
FAIRE
 symboles.h, 264
FIN
 symboles.h, 266
FOIS
 symboles.h, 262
facteur
 Expression, 152
faire
 n_instr_, 179
faire_
 n_instr_, 179
foncDec_
 n_dec_, 173
fonction
 n_appel_, 171
Fonction d'instruction, 38
 ADD_ENTRY, 39
 assembleur_entry_0s, 45
 assembleur_entry_0s_andComment, 40
 assembleur_entry_1s, 45
 assembleur_entry_1s_andComment, 41
 assembleur_entry_2s, 46
 assembleur_entry_2s_andComment, 42
 assembleur_entry_addEntryPoint, 47
 assembleur_entry_addEntryPoint_andComment,
 44
 assembleur_entry_add_subEntry, 47
 assembleur_entry_add_subEntry_andComment,
 43
 assembleur_entry_getLastInstr, 48
 assembleur_entry_getName, 48
 assembleur_entry_subEntryName, 47
 assembleur_entry_subEntryName_andComment,
 44
 CHECK_ENTRY, 39
 printedSizeOfSigned, 40
 printedSizeOfUnsigned, 40
 SECURE_INST_STR, 39
 SECURE_MALLOC_INST_STR, 39
Fonction de génération du flux de sortie, 34
 assembleur_dump, 37
 assembleur_dump_section_bss, 36
 assembleur_dump_section_data, 35

assembleur_dump_section_entry, 36
assembleur_dump_section_entry_instrs, 36
assembleur_dump_section_import, 35
assembleur_dump_section_oneEntry, 35
printSpace, 34
SIMPLE_ERROR, 34
Fonction de Gestion, 112
 tabsymb_ajoutIdentificateur, 113
 tabsymb_dump, 116
 tabsymb_entreeFonction, 112
 tabsymb_getSymbole, 115
 tabsymb_getSymboleFonctionCourante, 116
 tabsymb_rechercheDeclarative, 114
 tabsymb_rechercheExecutable, 114
 tabsymb_sortieFonction, 113
Fonction de gestion interne, 64
 parcours_get_endEntryName, 64
Fonction de gestion mémoire, 55
Fonction de gestion pour l'assembleur, 29
 assembleur_add_bss, 32
 assembleur_add_bss_andComment, 30
 assembleur_add_data, 32
 assembleur_add_data_andComment, 30
 assembleur_add_entry, 32
 assembleur_add_entry_andComment, 31
 assembleur_add_entryName, 33
 assembleur_add_entryName_andComment, 31
 assembleur_new_entryName, 30
 SECURE_MALLOC_STRUCT, 29
Fonction de parcours pour la génération, 61
Fonction et variable pour la gestion des listes d'adjacence, 20
 assembleur_add_commentary, 20
 assembleur_commentary_next, 21
 circularAdjListComparator_header, 22–24
 circularAdjListCopier_header, 21–25
 circularAdjListStaticDecl_struct, 22–24
 circularAdjListStruct_header, 21, 24
fonctions
 n_prog_, 184
Fonctions du parser yylex, 129
 delireCar, 130
 erreurLexical, 129
 estUnNombre, 131
 estUneFonction, 131
 estUneVariable, 131
 initBuff, 130
 lireCar, 130
 mangeEspaces, 129
 nom_token, 132
 yylex, 131
GENASM_ENTRY_ARE_INDENT
 Configuration du module Assembleur, 56
GENASM_ENTRY_PRETTY_NAME
 Configuration du module Assembleur, 56
GENASM_ENTRYCOUNTER_SEPERATE_BY_UNDERSCORE
 Configuration du module Assembleur, 56
GENASM_ENTRYCOUNTER_UNDERSCORE_SIZE
 Conversion en str, 49
GENASM_ENTRYPOINT_SEPERATE_RETURNLINE
 Configuration du module Assembleur, 56
Génération du code assembleur, 14
Gestion de la table des symboles, 110
 adresseArgumentCourant, 111
 adresseGlobalCourant, 111
 adresseLocaleCourante, 111
 tabsymb_setAffiche, 111
Gestion des commentaires, 28
Gestion des données, 16
 assembleur_entry_instr_circularAdjList, 16
Gestion erreur et yylex, 139
 erreurSyntaxical, 139
 uniteCouranteSuivante, 139
HAVE_LOGIQUE_COMPARAISON
 Configuration du module parcours, 93
ID_FCT
 symboles.h, 265
ID_VAR
 symboles.h, 265
INCR
 symboles.h, 265
INFERIEUR_EGAL
 symboles.h, 263
INFERIEUR
 symboles.h, 263
IO_USE_READLINE_ENTRY
 Configuration du module parcours, 93
identif
 desc_identif, 170
incr
 n_instr_, 179
indent
 afficheXml.c, 214
indice
 n_var_, 185
indicee_
 n_var_, 185
initBuff
 Fonctions du parser yylex, 130
initialise_premiers
 premiers.c, 248
initialise_premiers_nonterminaux
 /Remplissage des premiers, 158
initialise_premiers_terminaux
 /Remplissage des premiers, 157
initialise_suivants
 suivants.c, 252
initialise_suivants_nonterminaux
 /Remplissage des premiers, 163
initialise_suivants_terminaux
 /Remplissage des premiers, 162
instrs
 assembleur_entry, 169
Instruction, 71, 145

instruction, 145
instructionAffect, 145
instructionAppel, 147
instructionBloc, 146
instructionEcriture, 147
instructionFaire, 147
instructionIncremente, 148
instructionRetour, 147
instructionSi, 146
instructionTantque, 147
instructionVide, 147
listInstructions, 146
optSinon, 146
parcours_instr, 73
parcours_instr_affect, 76
parcours_instr_appel, 79
parcours_instr_ecrire, 75
parcours_instr_incr, 77
parcours_instr_retour, 81
parcours_instr_si, 80
parcours_instr_tantque, 78
parcours_prefixe_l_instr, 71
parcours_suffixe_l_instr, 72
instruction
 Instruction, 145
instructionAffect
 Instruction, 145
instructionAppel
 Instruction, 147
instructionBloc
 Instruction, 146
instructionEcriture
 Instruction, 147
instructionFaire
 Instruction, 147
instructionIncremente
 Instruction, 148
instructionRetour
 Instruction, 147
instructionSi
 Instruction, 146
instructionTantque
 Instruction, 147
instructionVide
 Instruction, 147
is_alpha
 Test des catégories, 122
is_alpha_simple
 Test des catégories, 123
is_alphanum
 Test des catégories, 123
is_maj
 Test des catégories, 122
is_min
 Test des catégories, 122
is_num
 Test des catégories, 122
isEnd
 AdjList, 166
LIB_COMPARAISONBOOL_PREFIX
 Bibliothèque comparaisonBool, 108
LIB_COMPARAISONBOOL_SIZE
 Bibliothèque comparaisonBool, 108
LIB_COMPARATOREXPR_PREFIX
 Bibliothèque comparatorExpr, 106
LIB_COMPARATOREXPR_SIZE
 Bibliothèque comparatorExpr, 106
LIRE
 symboles.h, 265
lireCar
 Fonctions du parser yylex, 130
Liste, 99
 parcoursTools_taille_n_l, 99
liste
 n_instr_, 180
listeDecFonctions
 Entrée, Variable globale, Fonction, Variable, 143
listeDecVariables
 Entrée, Variable globale, Fonction, Variable, 142
listeDecVariablesBis
 Entrée, Variable globale, Fonction, Variable, 142
listeExpressions
 Expression, 153
listeExpressionsBis
 Expression, 153
listInstructions
 Instruction, 146
listeParam
 Entrée, Variable globale, Fonction, Variable, 143
MAX_IDENTIF
 Definition pour la Gestion, 117
MODULO
 symboles.h, 262
MOINS
 symboles.h, 262
main
 Compilateur, 12
 dump_symb.c, 238
mangeEspaces
 Fonctions du parser yylex, 129
message.c, 238
 erreur, 239
 erreur_1s, 240
 erreurArgs, 239
 erreurLigne, 239
 warning_1s, 240
 warningLigne, 239
message.h, 241
Mots fonction, 127
 MotsClefs_Fonction_SIZE, 127
Mots instructions, 126
 MotsClefs_Instruction_SIZE, 126
MotsClefs_Fonction_SIZE
 Mots fonction, 127
 MotsClefs_Instruction_SIZE

Mots instructions, 126
MotsClefs_Operateurs_1Carac_SIZE
Table Opérateurs 1 Caractere, 124
MotsClefs_Operateurs_SIZE
Table des Opérateurs, 125
n_appel, 171
n_appel_
args, 171
fonction, 171
n_dec, 172
n_dec_
corps, 173
fondDec_, 173
nom, 173
param, 173
tabDec_, 174
taille, 174
type, 173, 174
u, 174
varDec_, 174
variables, 173
n_exp, 174
n_exp_
appel, 176
entier, 176
op, 175
op1, 175
op2, 175
opExp_, 176
type, 175
u, 176
var, 176
n_instr, 176
n_instr_
affecte_, 180
alors, 179
appel, 179
ecrire_, 180
exp, 180
expression, 180
faire, 179
faire_, 179
incr, 179
liste, 180
retour_, 180
si_, 179
sinon, 179
tantque_, 179
test, 179
type, 179
u, 180
var, 179
n_l_dec, 181
n_l_dec_
queue, 181
tete, 181
n_l_exp, 182
n_l_exp_
queue, 182
tete, 182
n_l_instr, 183
n_l_instr_
queue, 183
tete, 183
n_prog, 184
n_prog_
fonctions, 184
variables, 184
n_var, 184
n_var_
indice, 185
indicee_, 185
nom, 185
type, 185
u, 185
NB_NON_TERMINAUX
symboles.h, 255
NB_TERMINAUX
symboles.h, 261
NOMBRE
symboles.h, 265
NON
symboles.h, 264
name
assembleur_entry, 169
nb
assembleur_bss, 167
nb_ligne
Variables du parser yylex, 128
negation
Expression, 152
next
AdjList, 166
nom
n_dec_, 173
n_var_, 185
nom_token
Fonctions du parser yylex, 132
op
n_exp_, 175
op1
n_exp_, 175
op2
n_exp_, 175
opExp
n_exp_, 176
operation
syntabs.h, 272
optDecVariables
Entrée, Variable globale, Fonction, Variable, 142
optIndice
Expression, 153
optListeDecVariables
Entrée, Variable globale, Fonction, Variable, 143
optSinon
Instruction, 146

optTailleTableau
 Entrée, Variable globale, Fonction, Variable, 143

OU
 symboles.h, 264

Outils pour le parcours de l'arbres abstrait, 94

P_ARGUMENT
 Definition des portées, 118

P_VARIABLE_GLOBALE
 Definition des portées, 118

P_VARIABLE_LOCALE
 Definition des portées, 118

PARCOURS_STARTENTRY_ACCEPT_RETVALUE
 Configuration du module parcours, 93

PARCOURS_VAR_USE_DWORD
 Configuration du module parcours, 93

PARENTHESE_FERMANTE
 symboles.h, 262

PARENTHESE_OUVRANTE
 symboles.h, 262

PLUS
 symboles.h, 261

POINT_VIRGULE
 symboles.h, 261

param
 n_dec_, 173

Parcours de l'arbres abstrait pour la génération, 57

parcours.c, 241

parcours.h, 242

parcours_dec
 Declaration, 68

parcours_dec_fonctions
 Declaration, 69

parcours_exp
 Aquisition, 101
 Expression, 85

parcours_exp_appel
 Expression, 90

parcours_exp_comparaison
 Expression, 86

parcours_exp_int
 Expression, 89

parcours_exp_lire
 Expression, 87

parcours_exp_op
 Expression, 91

parcours_exp_var
 Expression, 88

parcours_get_endEntryName
 Fonction de gestion interne, 64

parcours_init_asm
 Entrées principales, 63

parcours_instr
 Instruction, 73

parcours_instr_affect
 Instruction, 76

parcours_instr_appel
 Instruction, 79

parcours_instr_ecrire
 Instruction, 75

parcours_instr_incr
 Instruction, 77

parcours_instr_retour
 Instruction, 81

parcours_instr_si
 Instruction, 80

parcours_instr_tantque
 Instruction, 78

parcours_prefixe_l_dec
 Declaration, 66

parcours_prefixe_l_exp
 Expression, 83

parcours_prefixe_l_instr
 Instruction, 71

parcours_prog
 Entrées principales, 62

parcours_suffixe_l_dec
 Declaration, 67

parcours_suffixe_l_exp
 Expression, 84

parcours_suffixe_l_instr
 Instruction, 72

parcoursTools.c, 243

parcoursTools.h, 245

parcoursTools_accesVar
 Aquisition, 102

parcoursTools_comparaisonOperation
 Expression, 98

parcoursTools_computeOperation
 Expression, 97

parcoursTools_constexpr
 Expression, 96

parcoursTools_constexpr_operation
 Expression, 95

parcoursTools_dump_asm_lib
 Bibliothèque, 104

parcoursTools_genLib_comparaisonBool
 Bibliothèque comparaisonBool, 109

parcoursTools_genLib_comparatorExpr
 Bibliothèque comparatorExpr, 107

parcoursTools_lib_comparaisonBool
 Bibliothèque comparaisonBool, 109

parcoursTools_lib_comparaisonBool_getName
 Bibliothèque comparaisonBool, 109

parcoursTools_lib_comparatorExpr
 Bibliothèque comparatorExpr, 107

parcoursTools_taille_n_l
 Liste, 99

pi
 assembleur_bss, 167
 assembleur_data, 168

portee
 desc_identif, 170

premier_union
 /Remplissage des premiers, 156

premier_union_suivant
 /Remplissage des premiers, 161

premier_valide
 /Remplissage des premiers, 155
premiers
 premiers.c, 249
 premiers.h, 250
premiers.c, 245
 est_premier, 246
 initialise_premiers, 248
 premiers, 249
premiers.h, 249
 premiers, 250
prev
 AdjList, 166
printSpace
 Fonction de génération du flux de sortie, 34
printedSizeOfSigned
 Fonction d'instruction, 40
printedSizeOfUnsigned
 Fonction d'instruction, 40
programme
 Entrée, Variable globale, Fonction, Variable, 141
pseudo_instruction_bss
 Énumération, 17
pseudo_instruction_bss_str
 Tableau des correspondance enum->str, 19
pseudo_instruction_data
 Énumération, 17
pseudo_instruction_data_str
 Tableau des correspondance enum->str, 19
queue
 n_l_dec_, 181
 n_l_exp_, 182
 n_l_instr_, 183
README.md, 250
RETOUR
 symboles.h, 265
resb
 Énumération, 18
resd
 Énumération, 18
resq
 Énumération, 18
rest
 Énumération, 18
resw
 Énumération, 18
retour_
 n_instr_, 180
SECURE_INST_STR
 Fonction d'instruction, 39
SECURE_MALLOC_INST_STR
 Fonction d'instruction, 39
SECURE_MALLOC_STRUCT
 Fonction de gestion pour l'assembleur, 29
SIMPLE_ERROR
 Fonction de génération du flux de sortie, 34
SINON
 symboles.h, 264
SUPERIEUR_EGAL
 symboles.h, 263
SUPERIEUR
 symboles.h, 263
Script d'exécution, 133
 afficherLexique, 133
SI
 symboles.h, 264
si_
 n_instr_, 179
sinon
 n_instr_, 179
sommet
 tabsymboles_, 187
Sortie
 Define des fonctions, 136
special_texte
 afficheXml.c, 217
suivant_union
 /Remplissage des premiers, 161
suivant_valide
 /Remplissage des premiers, 160
suivants
 suivants.c, 253
 suivants.h, 254
suivants.c, 250
 est_suivant, 251
 initialise_suivants, 252
 suivants, 253
suivants.h, 253
 suivants, 254
symboles.h, 254
 appelFct, 258
 argumentsEffectifs, 259
 comparaisonBis, 260
 comparaison, 258
 conjonctionBis, 259
 conjonction, 258
 declarationFonction, 256
 declarationVariable, 256
 expArithBis, 260
 expArith, 258
 expressionBis, 261
 expression, 258
 facteur, 259
 instructionAffect, 256
 instructionAppel, 257
 instructionBloc, 256
 instructionEcriture, 257
 instructionFaire, 261
 instructionIncrementer, 257
 instructionPour, 260
 instructionRetour, 257
 instructionSi, 257
 instructionTantque, 257
 instructionVide, 257

instruction, 256
listeDecFonctions, 255
listeDecVariablesBis, 260
listeDecVariables, 255
listeExpressionsBis, 259
listeExpressions, 259
listInstructions, 256
listeParam, 256
negation, 258
optDecVariables, 260
optIndice, 260
optListeDecVariables, 261
optSinon, 260
optTailleTableau, 259
programme, 261
termeBis, 260
terme, 259
var, 258
ACCOLADE_FERMANTE, 263
ACCOLADE_OUVRANTE, 263
ALORS, 264
CROCHET_FERMANT, 262
CROCHET_OUVRANT, 262
DIFFERENT, 263
DIVISE, 262
Ecrire, 265
EGAL, 263
ENTIER, 265
EPSILON, 255
ET, 264
FAIRE, 264
FIN, 266
FOIS, 262
ID_FCT, 265
ID_VAR, 265
INCR, 265
INFERIEUR_EGAL, 263
INFERIEUR, 263
LIRE, 265
MODULO, 262
MOINS, 262
NB_NON_TERMINAUX, 255
NB_TERMINAUX, 261
NOMBRE, 265
NON, 264
OU, 264
PARENTHESE_FERMANTE, 262
PARENTHESE_OUVRANTE, 262
PLUS, 261
POINT_VIRGULE, 261
RETOUR, 265
SINON, 264
SUPERIEUR_EGAL, 263
SUPERIEUR, 263
SI, 264
TANTQUE, 264
VIRGULE, 266
syntabs.c, 266

cree_n_appel, 267
cree_n_dec_fonc, 271
cree_n_dec_tab, 271
cree_n_dec_var, 271
cree_n_exp_appel, 268
cree_n_exp_entier, 268
cree_n_exp_lire, 268
cree_n_exp_op, 267
cree_n_exp_var, 268
cree_n_instr_affect, 269
cree_n_instr_appel, 270
cree_n_instr_bloc, 269
cree_n_instr_ecrire, 270
cree_n_instr_faire, 269
cree_n_instr_incr, 268
cree_n_instr_retour, 270
cree_n_instr_si, 269
cree_n_instr_tantque, 269
cree_n_instr_vide, 270
cree_n_l_dec, 271
cree_n_l_exp, 268
cree_n_l_instr, 269
cree_n_prog, 267
cree_n_var_indicee, 267
cree_n_var_simple, 267

syntabs.h, 271

cree_n_appel, 276
cree_n_dec_fonc, 273
cree_n_dec_tab, 273
cree_n_dec_var, 273
cree_n_exp_appel, 274
cree_n_exp_entier, 273
cree_n_exp_lire, 274
cree_n_exp_op, 273
cree_n_exp_var, 274
cree_n_instr_affect, 275
cree_n_instr_appel, 275
cree_n_instr_bloc, 274
cree_n_instr_ecrire, 276
cree_n_instr_faire, 275
cree_n_instr_incr, 274
cree_n_instr_retour, 275
cree_n_instr_si, 274
cree_n_instr_tantque, 275
cree_n_instr_vide, 276
cree_n_l_dec, 277
cree_n_l_exp, 277
cree_n_l_instr, 277
cree_n_prog, 273
cree_n_var_indicee, 277
cree_n_var_simple, 276
operation, 272

syntactical

Ananalyse Syntaxique, 135

syntactical_setAffiche

Ananalyse Syntaxique, 135

T_ENTIER

Definition des types, 119

T_FONCTION
 Definition des types, 119

T_TABLEAU_ENTIER
 Definition des types, 119

TANTQUE
 symboles.h, 264

tab
 tabsymboles_, 187

tabDec_
 n_dec_, 174

Table des Opérateurs, 125
 MotsClefs_Operators_SIZE, 125

Table Opérateurs 1 Caractere, 124
 MotsClefs_Operators_1Carac_SIZE, 124

Tableau des corespondance enum->str, 19
 pseudo_instruction_bss_str, 19
 pseudo_instruction_data_str, 19

tabsymb ajoutelidentificateur
 Fonction de Gestion, 113

tabsymb_dump
 Fonction de Gestion, 116

tabsymb_entreeFonction
 Fonction de Gestion, 112

tabsymb_getSymbole
 Fonction de Gestion, 115

tabsymb_getSymboleFonctionCourante
 Fonction de Gestion, 116

tabsymb_rechercheDeclarative
 Fonction de Gestion, 114

tabsymb_rechercheExecutable
 Fonction de Gestion, 114

tabsymb_setAffiche
 Gestion de la table des symboles, 111

tabsymb_sortieFonction
 Fonction de Gestion, 113

tabsymboles.c, 277

tabsymboles.h, 278

tabsymboles_, 186
 base, 187
 sommet, 187
 tab, 187

taille
 n_dec_, 174

tantque_
 n_instr_, 179

terme
 Expression, 151

termeBis
 Expression, 152

test
 n_instr_, 179

Test des catégories, 122
 is_alpha, 122
 is_alpha_simple, 123
 is_alphanum, 123
 is_maj, 122
 is_min, 122
 is_num, 122

YYTEXT_MAX, 122

tete
 n_l_dec_, 181
 n_l_exp_, 182
 n_l_instr_, 183

topEntry
 Variable de gestion de la table des symboles, 59

type
 desc_identif, 170
 n_dec_, 173, 174
 n_exp_, 175
 n_instr_, 179
 n_var_, 185

u
 n_dec_, 174
 n_exp_, 176
 n_instr_, 180
 n_var_, 185

uniteCouranteSuivante
 Gestion erreur et yylex, 139

VIRGULE
 symboles.h, 266

valeur
 assembleur_data, 168

var
 Expression, 152
 n_exp_, 176
 n_instr_, 179

varDec_
 n_dec_, 174

Variable de gestion de l'assembleur, 27

Variable de gestion de la table des symboles, 58
 adresseArgumentCourant, 58
 adresseGlobalCourant, 58
 adresseLocaleCourante, 58
 ce, 58
 endEntry, 59
 topEntry, 59

variables
 n_dec_, 173
 n_prog_, 184

Variables du parser yylex, 128
 nb_ligne, 128
 yytext, 128

Variables globales, 138

warning_1s
 message.c, 240

warningLigne
 message.c, 239

word
 Énumération, 18

YYTEXT_MAX
 Test des catégories, 122

yyin
 Compilateur, 13

`yylex`

Fonctions du parser `yylex`, [131](#)

`yytext`

Compilateur, [13](#)

Variables du parser `yylex`, [128](#)