

# Rapport de tests

# Table des matières

<b>1 and</b>	<b>1</b>
1.1 KO . . . . .	1
1.1.1 & mal écrit . . . . .	1
1.2 OK . . . . .	2
1.2.1 condition avec 1 & . . . . .	2
1.2.2 condition avec 2 & . . . . .	3
1.2.3 condition avec 3 & . . . . .	4
1.2.4 condition avec 2 & et 1   . . . . .	5
1.2.5 & avec des entiers . . . . .	6
1.2.6 & avec des chaînes . . . . .	8
<b>2 break</b>	<b>10</b>
2.1 KO . . . . .	10
2.1.1 break comme condition dans if . . . . .	10
2.1.2 break comme condition dans while . . . . .	12
2.2 OK . . . . .	12
2.2.1 break dans un for, après instruction . . . . .	12
2.2.2 break dans un for, sans instruction avant . . . . .	12
2.2.3 fonction exécutant seulement break . . . . .	13
2.2.4 appel de break, avec corps de programme contenant break . . . . .	14
2.2.5 appel de break, avec corps de programme ne contenant pas break . . . . .	15
2.2.6 break dans un while, après instruction . . . . .	16
2.2.7 break dans un while, sans instruction avant . . . . .	17
2.2.8 break dans un if then, après instruction . . . . .	18
2.2.9 break dans un if then, sans instruction avant . . . . .	19
2.2.10 break dans un if then else, après instruction . . . . .	20
2.2.11 break dans un if then else, sans instruction avant . . . . .	21
2.2.12 break seul . . . . .	22
2.2.13 break seul, après saut de ligne . . . . .	23
<b>3 calculation</b>	<b>23</b>
3.1 KO . . . . .	23
3.1.1 addition avec opérateur mal écrit . . . . .	23
3.1.2 soustraction avec opérateur mal écrit . . . . .	25
3.1.3 multiplication avec opérateur mal écrit . . . . .	27
3.1.4 division avec opérateur mal écrit . . . . .	29
3.1.5 moins unaire avec opérateur mal écrit . . . . .	31
3.1.6 calcul à 2 termes avec oubli de parenthèse gauche . . . . .	32
3.1.7 calcul à 2 termes avec oubli de parenthèse droite . . . . .	34
3.1.8 calcul à 3 termes avec oubli de parenthèse gauche . . . . .	36
3.1.9 calcul à 3 termes avec oubli de parenthèse droite . . . . .	38
3.1.10 calcul à 4 termes avec oubli de parenthèse gauche . . . . .	40
3.1.11 calcul à 4 termes avec oubli de parenthèse droite . . . . .	42
3.1.12 parenthesage sans contenu . . . . .	44
3.1.13 expression parenthésée avec ; en trop . . . . .	44
3.1.14 expression parenthésée avec oubli de ; . . . . .	45
3.1.15 expression non parenthésée avec oubli de ; . . . . .	46
3.1.16 expression non parenthésée avec ; en trop . . . . .	47
3.1.17 ; sans instruction avant . . . . .	48
3.2 OK . . . . .	49
3.2.1 addition simple, à 2 termes . . . . .	49
3.2.2 soustraction simple, à 2 termes . . . . .	51
3.2.3 multiplication simple, à 2 termes . . . . .	53
3.2.4 division simple, à 2 termes . . . . .	55
3.2.5 addition à 3 termes . . . . .	57

3.2.6	addition suivie de soustraction . . . . .	59
3.2.7	addition suivie de multiplication . . . . .	61
3.2.8	addition suivie de division . . . . .	63
3.2.9	multiplication suivie d'addition . . . . .	65
3.2.10	multiplication suivie de soustraction . . . . .	67
3.2.11	multiplication a 3 termes . . . . .	69
3.2.12	multiplication suivie de division . . . . .	71
3.2.13	addition simple, a 2 termes, identifies par des variables . . . . .	73
3.2.14	soustraction simple, a 2 termes, identifies par des variables . . . . .	73
3.2.15	multiplication simple, a 2 termes, identifies par des variables . . . . .	74
3.2.16	division simple, a 2 termes, identifies par des variables . . . . .	75
3.2.17	addition a 3 termes, identifies par des variables . . . . .	76
3.2.18	addition suivie de soustraction, avec termes identifies par variables . . . . .	77
3.2.19	addition suivie de multiplication, avec termes identifies par variables . . . . .	78
3.2.20	addition suivie de division, avec termes identifies par variables . . . . .	79
3.2.21	multiplication suivie d'addition, avec termes identifies par variables . . . . .	80
3.2.22	multiplication suivie de soustraction, avec termes identifies par variables . . . . .	81
3.2.23	multiplication a 3 termes, identifies par des variables . . . . .	82
3.2.24	multliplication suivie de division, avec termes identifies par variables . . . . .	83
3.2.25	addition simple, a 2 termes, dont un ayant moins unaire . . . . .	84
3.2.26	soustraction simple, a 2 termes, dont ayant moins unaire . . . . .	86
3.2.27	multiplication simple, a 2 termes, dont un ayant moins unaire . . . . .	88
3.2.28	division simple, a 2 termes, dont un ayant moins unaire . . . . .	90
3.2.29	addition a 3 termes, dont un ayant moins unaire . . . . .	92
3.2.30	addition suivie de soustraction, avec un terme ayant moins unaire . . . . .	94
3.2.31	addition suivie de multiplication, avec un terme ayant moins unaire . . . . .	96
3.2.32	addition suivie de division, avec un terme ayant moins unaire . . . . .	98
3.2.33	multiplication suivie d'addition, dont un terme ayant moins unaire . . . . .	100
3.2.34	multiplication suivie de soustraction, dont un terme ayant moins unaire . . . . .	102
3.2.35	multiplication a 3 termes, dont un ayant moins unaire . . . . .	104
3.2.36	multiplication suivie de division, dont un terme ayant moins unaire . . . . .	106
3.2.37	addition simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	108
3.2.38	soustraction simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	108
3.2.39	multiplication simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	109
3.2.40	division simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	110
3.2.41	addition a 3 termes, identifies par des variables, dont une ayant moins unaire . . . . .	111
3.2.42	addition suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire . . . . .	112
3.2.43	addition suivie de multiplication, avec termes identifies par variables, dont une ayant moins unaire . . . . .	113
3.2.44	addition suivie de division, avec termes identifies par variables, dont une ayant moins unaire . . . . .	114
3.2.45	multiplication suivie d'addition, avec termes identifies par variables, dont une ayant moins unaire . . . . .	115
3.2.46	multiplication suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire . . . . .	116
3.2.47	multiplication a 3 termes, identifies par des variables, dont une ayant moins unaire . . . . .	117
3.2.48	multliplication suivie de division, avec termes identifies par variables, dont une ayant moins unaire . . . . .	118
3.2.49	addition a 3 termes, avec parenthesage des 2 termes a droite . . . . .	119
3.2.50	addition suivie de soustraction, avec parenthesage des 2 termes a droite . . . . .	121
3.2.51	addition suivie de multiplication, avec parenthesage des 2 termes a droite . . . . .	123
3.2.52	addition suivie de division, avec parenthesage des 2 termes a droite . . . . .	125
3.2.53	multiplication suivie d'addition, avec parenthesage des 2 termes a droite . . . . .	127
3.2.54	multiplication suivie de soustraction, avec parenthesage des 2 termes a droite . . . . .	129
3.2.55	multiplication a 3 termes, avec parenthesage des 2 termes a droite . . . . .	131
3.2.56	multiplication suivie de vision, avec parenthesage des 2 termes a droite . . . . .	133
3.2.57	addition a 3 termes, avec parenthesage des 2 termes a gauche . . . . .	135

3.2.58 addition suivie de soustraction, avec parenthesage des 2 termes à gauche . . . . .	137
3.2.59 addition suivie de multiplication, avec parenthesage des 2 termes à gauche . . . . .	139
3.2.60 addition suivie de division, avec parenthesage des 2 termes à gauche . . . . .	141
3.2.61 multiplication suivie d'addition, avec parenthesage des 2 termes à gauche . . . . .	143
3.2.62 multiplication suivie de soustraction, avec parenthesage des 2 termes à gauche . . . . .	145
3.2.63 multiplication à 3 termes, avec parenthesage des 2 termes à gauche . . . . .	147
3.2.64 multiplication suivie de division, avec parenthesage des 2 termes à gauche . . . . .	149
3.2.65 multiplication de 2 additions parenthèses . . . . .	151
3.2.66 multiplication de 3 additions parenthèses . . . . .	153
3.2.67 division de 2 soustractions parenthèses . . . . .	155
3.2.68 division de 3 soustractions parenthèses . . . . .	157
3.2.69 addition de 2 multiplications parenthèses . . . . .	159
3.2.70 addition de 3 multiplications parenthèses . . . . .	161
3.2.71 soustraction de 2 divisions parenthèses . . . . .	163
3.2.72 soustraction de 3 divisions parenthèses . . . . .	165
3.2.73 addition à 3 termes identifiés par variables, avec parenthesage des 2 variables à droite . . . . .	167
3.2.74 addition suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	167
3.2.75 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	168
3.2.76 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	169
3.2.77 multiplication suivie d'addition, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	170
3.2.78 multiplication suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	171
3.2.79 multiplication à 3 termes identifiés par variables, avec parenthesage des 2 variables à droite . . . . .	172
3.2.80 multiplication suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	173
3.2.81 addition à 3 termes identifiés par variables, avec parenthesage des 2 variables à gauche . . . . .	174
3.2.82 addition suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	175
3.2.83 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	176
3.2.84 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	177
3.2.85 multiplication suivie d'addition, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	178
3.2.86 multiplication suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	179
3.2.87 multiplication à 3 termes identifiés par variables, avec parenthesage des 2 variables à gauche . . . . .	180
3.2.88 multiplication suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	181
3.2.89 multiplication de 2 additions parenthèses, avec termes identifiés par variables . . . . .	182
3.2.90 multiplication de 3 additions parenthèses, avec termes identifiés par variables . . . . .	183
3.2.91 division de 2 soustractions parenthèses, avec termes identifiés par variables . . . . .	184
3.2.92 division de 3 soustractions parenthèses, avec termes identifiés par variables . . . . .	185
3.2.93 addition de 2 multiplications parenthèses, avec termes identifiés par variables . . . . .	186
3.2.94 addition de 3 multiplications parenthèses, avec termes identifiés par variables . . . . .	187
3.2.95 soustraction de 2 divisions parenthèses, avec termes identifiés par variables . . . . .	188
3.2.96 soustraction de 3 divisions parenthèses, avec termes identifiés par variables . . . . .	189
3.2.97 parenthesage d'une instruction . . . . .	190
3.2.98 parenthesage de 2 instructions . . . . .	191
3.2.99 parenthesage de 3 instructions . . . . .	192
3.2.100 1 instruction sans parenthèses . . . . .	193
3.2.101 2 instructions sans parenthèses . . . . .	194

3.2.102	3 instructions sans parenthèses . . . . .	195
3.2.103	concatenation d'entier et de chaine . . . . .	195
3.2.104	concatenation de 2 chaines . . . . .	197
<b>4</b>	<b>comment</b>	<b>197</b>
4.1	KO . . . . .	197
4.1.1	imbrication . . . . .	197
4.1.2	oubli de / en debut de commentaire . . . . .	198
4.1.3	oubli de * en debut de commentaire . . . . .	198
4.1.4	oubli de en debut de commentaire . . . . .	199
4.1.5	oubli de * en fin de commentaire . . . . .	200
4.1.6	oubli de / en fin de commentaire . . . . .	201
4.1.7	oubli de en fin de commentaire . . . . .	201
4.2	OK . . . . .	202
4.2.1	1 commentaire, sans instruction avant, sur 1 ligne . . . . .	202
4.2.2	1 commentaire, sans instruction avant, sur plusieurs lignes . . . . .	203
4.2.3	1 commentaire, apres instruction, sur 1 ligne . . . . .	205
4.2.4	1 commentaire, apres instruction, sur plusieurs lignes . . . . .	207
4.2.5	2 commentaires, sans instruction avant, sur 1 ligne . . . . .	208
4.2.6	2 commentaires, sans instruction avant, sur plusieurs lignes . . . . .	210
4.2.7	2 commentaires, apres instructions, sur 1 ligne . . . . .	212
4.2.8	2 commentaires, apres instructions, sur plusieurs lignes . . . . .	213
4.2.9	3 commentaires, sans instruction avant, sur 1 ligne . . . . .	214
4.2.10	3 commentaires, sans instruction avant, sur plusieurs lignes . . . . .	215
4.2.11	3 commentaires, apres instructions, sur 1 ligne . . . . .	217
4.2.12	3 commentaires, apres instructions, sur plusieurs lignes . . . . .	218
4.2.13	commentaire sans contenu sur 1 ligne . . . . .	219
4.2.14	commentaire sans contenu sur plusieurs lignes . . . . .	220
<b>5</b>	<b>comparison</b>	<b>223</b>
5.1	KO . . . . .	223
5.1.1	comparaison avec oubli d'operande gauche . . . . .	223
5.1.2	comparaison avec oubli d'operator . . . . .	224
5.1.3	comparaison avec oubli d'operator droit . . . . .	225
5.2	OK . . . . .	226
5.2.1	comparaison simple d'entiers avec "" . . . . .	226
5.2.2	comparaison simple d'entiers avec "" . . . . .	228
5.2.3	comparaison simple d'entiers avec "=" . . . . .	230
5.2.4	comparaison simple d'entiers avec "=" . . . . .	232
5.2.5	comparaison simple d'entiers avec = . . . . .	234
5.2.6	comparaison simple d'entiers avec "" . . . . .	236
5.2.7	comparaison double d'entiers . . . . .	238
5.2.8	comparaison triple d'entiers . . . . .	239
5.2.9	comparaison simple de chaines avec "" . . . . .	240
5.2.10	comparaison simple de chaines avec "" . . . . .	242
5.2.11	comparaison simple de chaines avec "=" . . . . .	244
5.2.12	comparaison simple de chaines avec "=" . . . . .	246
5.2.13	comparaison simple de chaines avec = . . . . .	248
5.2.14	comparaison simple de chaines avec "" . . . . .	250
5.2.15	comparaison double de chaines . . . . .	252
5.2.16	comparaison triple de chaines . . . . .	253

<b>6 for</b>	<b>254</b>
6.1 KO . . . . .	254
6.1.1 for avec affectation de chaine pour le compteur . . . . .	254
6.1.2 for avec affectation de chaine pour la borne superieure de l'iteration . . . . .	255
6.1.3 for avec oubli de l'identifiant du compteur . . . . .	256
6.1.4 for avec oubli du for . . . . .	257
6.1.5 for "express" . . . . .	258
6.1.6 for avec affectation de chaines pour le compteur et la borne maximale de l'iteration . . . . .	258
6.1.7 for avec oubli du do . . . . .	259
6.1.8 for avec oubli de la borne maximale de l'iteration . . . . .	260
6.1.9 for avec oubli du to . . . . .	261
6.1.10 for avec oubli de la borne inferieure de l'iteration . . . . .	262
6.1.11 for avec oubli du = . . . . .	263
6.1.12 for avec oubli du : . . . . .	264
6.1.13 for avec oubli du := . . . . .	264
6.2 OK . . . . .	265
6.2.1 for simple croissant . . . . .	265
6.2.2 for simple decroissant . . . . .	266
6.2.3 for sans instruction . . . . .	267
6.2.4 for avec ligne vide . . . . .	268
6.2.5 double-imbrication de for . . . . .	269
6.2.6 triple-imbrication de for . . . . .	270
6.2.7 for avec reutilisation de compteur . . . . .	270
6.2.8 for avec reutilisation de compteurs . . . . .	271
<b>7 function</b>	<b>271</b>
7.1 KO . . . . .	271
7.1.1 fonction identifiee par caractere special . . . . .	271
7.1.2 oubli de function . . . . .	272
7.1.3 fonction sans identifiant . . . . .	273
7.1.4 oubli de ( . . . . .	274
7.1.5 oubli d'identifiant de parametre . . . . .	275
7.1.6 oubli de : pour parametre . . . . .	276
7.1.7 oubli de type de parametre . . . . .	277
7.1.8 oubli de ) . . . . .	278
7.1.9 oubli de : pour type de fonction . . . . .	279
7.1.10 : pour type de fonction present mais pas de type . . . . .	280
7.1.11 oubli de = . . . . .	281
7.1.12 parametres mal separates . . . . .	282
7.1.13 fonction sans instruction . . . . .	283
7.1.14 function mal ecrit . . . . .	284
7.1.15 fonction avec identifiant seulement en parametre . . . . .	285
7.1.16 fonction avec entier directement en parametre . . . . .	286
7.1.17 fonction avec chaine directement en parametre . . . . .	287
7.2 OK . . . . .	289
7.2.1 fonction definissant un entier . . . . .	289
7.2.2 fonction simple, avec declaration du type de retour int et instruction de retour . . . . .	289
7.2.3 fonction simple, avec instruction simple et instruction retournant un entier . . . . .	290
7.2.4 fonction avec 1 parametre entier . . . . .	291
7.2.5 fonction avec 2 parametres entiers . . . . .	292
7.2.6 fonction avec 3 parametres entiers . . . . .	293
7.2.7 fonction avec 2 parametres entiers et 1 parametre de type string . . . . .	295
7.2.8 fonction definissant une chaine . . . . .	295
7.2.9 fonction simple, avec declaration du type de retour string et instruction de retour . . . . .	296
7.2.10 fonction simple, avec instruction simple et instruction retournant une chaine . . . . .	297
7.2.11 fonction avec 1 parametre de type string . . . . .	298
7.2.12 fonction avec 2 parametres de type string . . . . .	299

7.2.13 fonction avec 3 parametres de type string . . . . .	300
7.2.14 fonction avec 2 parametres de type string et 1 parametre entier . . . . .	300
7.2.15 double-imbrication de fonction . . . . .	301
7.2.16 triple-imbrication de fonction . . . . .	301
7.2.17 recursion finie . . . . .	302
7.2.18 recursion infinie . . . . .	303
<b>8 if</b>	<b>305</b>
8.1 KO . . . . .	305
8.1.1 incrementation dans la condition if then . . . . .	305
8.1.2 affectation d'entier dans la condition if then . . . . .	306
8.1.3 affectation de chaine dans la condition if then . . . . .	307
8.1.4 if then avec oubli du if . . . . .	308
8.1.5 if then avec oubli du then . . . . .	309
8.1.6 if then avec oubli de la condition . . . . .	310
8.1.7 if then else avec oubli de la condition . . . . .	312
8.1.8 if then else sans instruction dans else . . . . .	313
8.1.9 if then else avec ligne vide dans else . . . . .	314
8.2 OK . . . . .	315
8.2.1 if then avec condition entiere . . . . .	315
8.2.2 if then sans instruction . . . . .	315
8.2.3 if then avec ligne vide . . . . .	316
8.2.4 if then avec simple condition . . . . .	317
8.2.5 if then avec double-condition . . . . .	318
8.2.6 if then avec triple-condition . . . . .	319
8.2.7 double-imbrication de if then . . . . .	320
8.2.8 triple-imbrication de if then . . . . .	321
8.2.9 double-imbrication de if then else . . . . .	322
8.2.10 triple-imbrication de if then else . . . . .	323
<b>9 let</b>	<b>324</b>
9.1 KO . . . . .	324
9.1.1 let sans declaration . . . . .	324
9.1.2 let sans instruction . . . . .	325
9.1.3 let avec inversion de declaration et instruction . . . . .	326
9.1.4 let avec declaration vide . . . . .	327
9.1.5 let avec instruction vide . . . . .	327
9.2 OK . . . . .	328
9.2.1 let avec 1 declaration de fonction et 1 appel de fonction . . . . .	328
9.2.2 let avec 2 declarations de fonction et 2 appels de fonction . . . . .	329
9.2.3 let avec 3 declarations de fonction et 3 appels de fonction . . . . .	330
9.2.4 let avec 1 declaration de variable et 1 instruction . . . . .	331
9.2.5 let avec 2 declarations de variable et 2 instructions . . . . .	332
9.2.6 let avec 3 declarations de variable et 3 instructions . . . . .	333
9.2.7 let avec 3 declarations de variable et fonction, et 3 instructions et appels de fonction . . . . .	333
9.2.8 let avec double-imbrication . . . . .	334
9.2.9 let avec triple-imbrication . . . . .	335
<b>10 or</b>	<b>337</b>
10.1 KO . . . . .	337
10.1.1   mal ecrit . . . . .	337
10.2 OK . . . . .	338
10.2.1 condition avec 1   . . . . .	338
10.2.2 condition avec 2   . . . . .	339
10.2.3 condition avec 3   . . . . .	340
10.2.4 condition avec 2   et 1 & . . . . .	341
10.2.5   avec des entiers . . . . .	343

10.2.6   avec des chaines . . . . .	345
<b>11 var</b>	<b>347</b>
11.1 KO . . . . .	347
11.1.1 declaration sans affectation . . . . .	347
11.1.2 declaration d'entier avec type sans affectation . . . . .	347
11.1.3 declaration de chaîne avec type sans affectation . . . . .	348
11.1.4 affectation d'entier sans declaration . . . . .	349
11.1.5 affectation de chaîne sans declaration . . . . .	350
11.1.6 affectation de nil . . . . .	350
11.1.7 double-declaration avec valeurs égales . . . . .	351
11.1.8 double-declaration avec valeurs différentes . . . . .	352
11.1.9 var mal écrit . . . . .	353
11.1.10 déclaration de caractère spécial . . . . .	354
11.1.11 affectation de caractère spécial . . . . .	355
11.1.12 affectation d'opération logique sur entiers . . . . .	356
11.1.13 affectation d'opération logique sur chaînes . . . . .	357
11.1.14 chiffre comme nom de variable . . . . .	358
11.1.15 _ comme nom de variable . . . . .	359
11.2 OK . . . . .	360
11.2.1 déclaration d'entier simple . . . . .	360
11.2.2 déclarations d'entier avec réutilisation de 2 autres entiers . . . . .	361
11.2.3 déclarations d'entier avec réutilisation de 3 autres entiers . . . . .	362
11.2.4 déclaration simple d'entier avec type . . . . .	363
11.2.5 déclarations d'entier avec types et réutilisation de 2 autres entiers . . . . .	364
11.2.6 déclarations d'entier avec types et réutilisation de 3 autres entiers . . . . .	365
11.2.7 déclaration de chaîne simple . . . . .	366
11.2.8 déclarations de chaîne avec réutilisation de 2 autres chaînes . . . . .	367
11.2.9 déclarations de chaîne avec réutilisation de 3 autres chaînes . . . . .	368
11.2.10 déclaration de chaîne avec type . . . . .	369
11.2.11 déclarations de chaîne avec types et réutilisation de 2 autres chaînes . . . . .	370
11.2.12 déclarations de chaîne avec types et réutilisation de 3 autres chaînes . . . . .	371
11.2.13 lettre minuscule comme nom de variable . . . . .	372
11.2.14 lettre majuscule comme nom de variable . . . . .	372
11.2.15 lettre minuscule et chiffre dans nom de variable . . . . .	373
11.2.16 lettre majuscule et chiffre dans nom de variable . . . . .	374
11.2.17 lettre minuscule et _ dans nom de variable . . . . .	375
11.2.18 lettre majuscule et _ dans nom de variable . . . . .	376
11.2.19 lettre minuscule, chiffre et _ dans nom de variable . . . . .	377
11.2.20 lettre majuscule, chiffre et _ dans nom de variable . . . . .	378
<b>12 while</b>	<b>379</b>
12.1 KO . . . . .	379
12.1.1 while avec affectation d'entier . . . . .	379
12.1.2 while avec affectation de chaîne . . . . .	380
12.1.3 while avec oubli du compteur . . . . .	380
12.1.4 while avec oubli du while . . . . .	382
12.1.5 while avec oubli du do . . . . .	382
12.1.6 while avec oubli de la condition . . . . .	382
12.2 OK . . . . .	384
12.2.1 while avec itération croissante . . . . .	384
12.2.2 while avec itération décroissante . . . . .	384
12.2.3 while avec double-condition . . . . .	385
12.2.4 while avec triple-condition . . . . .	386
12.2.5 while sans instruction . . . . .	386
12.2.6 while avec ligne vide . . . . .	387
12.2.7 while avec condition entière . . . . .	388

12.2.8 while avec double-imbrication . . . . .	388
12.2.9 while avec triple-imbrication . . . . .	389
12.2.10 while avec reutilisation de compteur . . . . .	390
12.2.11 while avec reutilisation de compteurs . . . . .	390
12.2.12 while avec iteration sur une chaine . . . . .	391
<b>12.3 OK . . . . .</b>	<b>392</b>
12.3.1 1 espace entre var et i . . . . .	392
12.3.2 1 tabulation entre var et i . . . . .	393
12.3.3 1 saut de ligne entre var et i . . . . .	394
12.3.4 2 espaces entre var et i . . . . .	395
12.3.5 2 tabulations entre var et i . . . . .	396
12.3.6 2 sauts de ligne entre var et i . . . . .	397
12.3.7 3 espaces entre var et i . . . . .	398
12.3.8 3 tabulations entre var et i . . . . .	399
12.3.9 3 sauts de ligne entre var et i . . . . .	400

1 and

## 1.1 KO

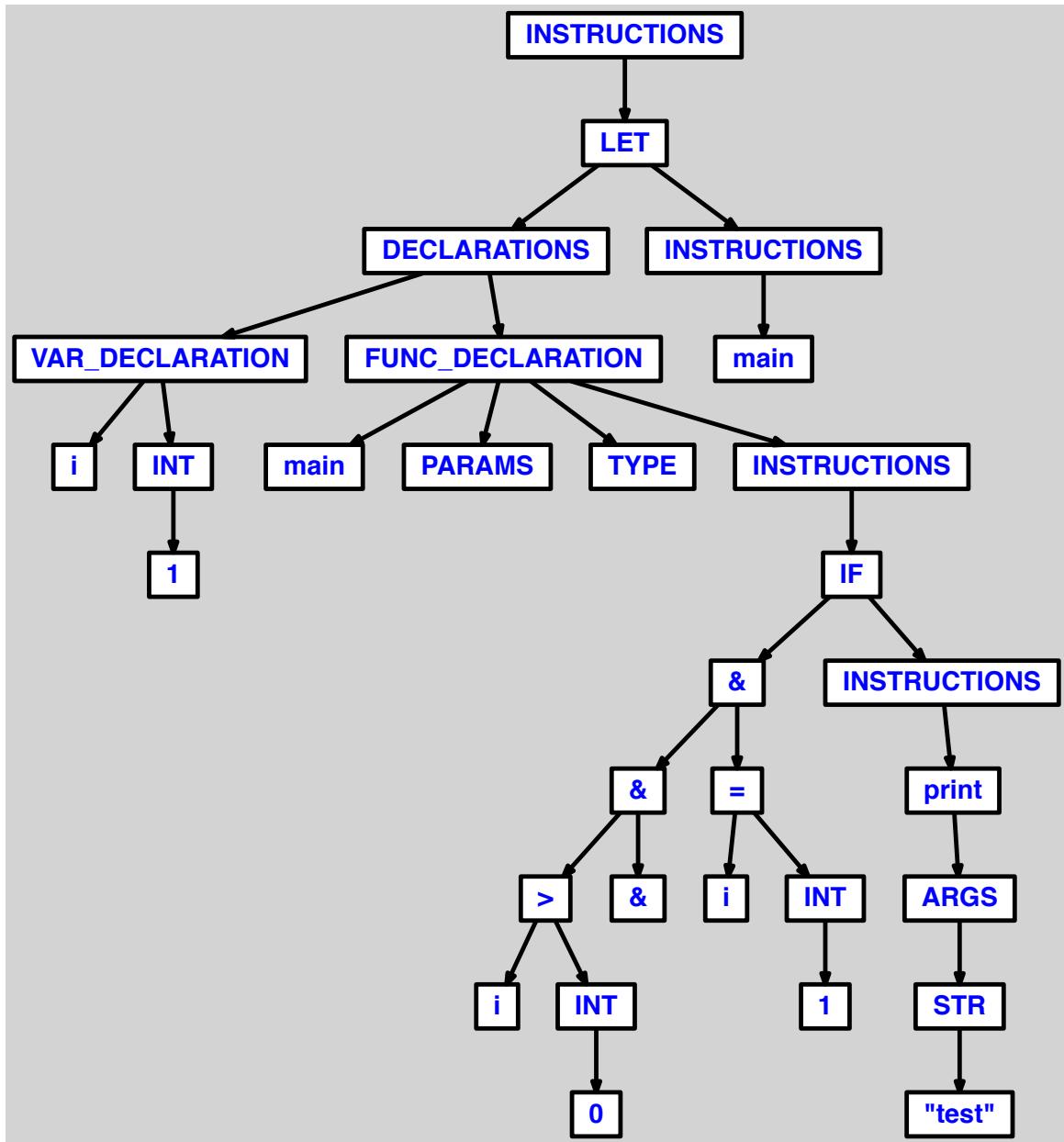
### 1.1.1 & mal écrit

let

```
var i := 1
```

```
function main() =  
    if i > 0 && i = 1 then print("test")
```

```
in main() end
```

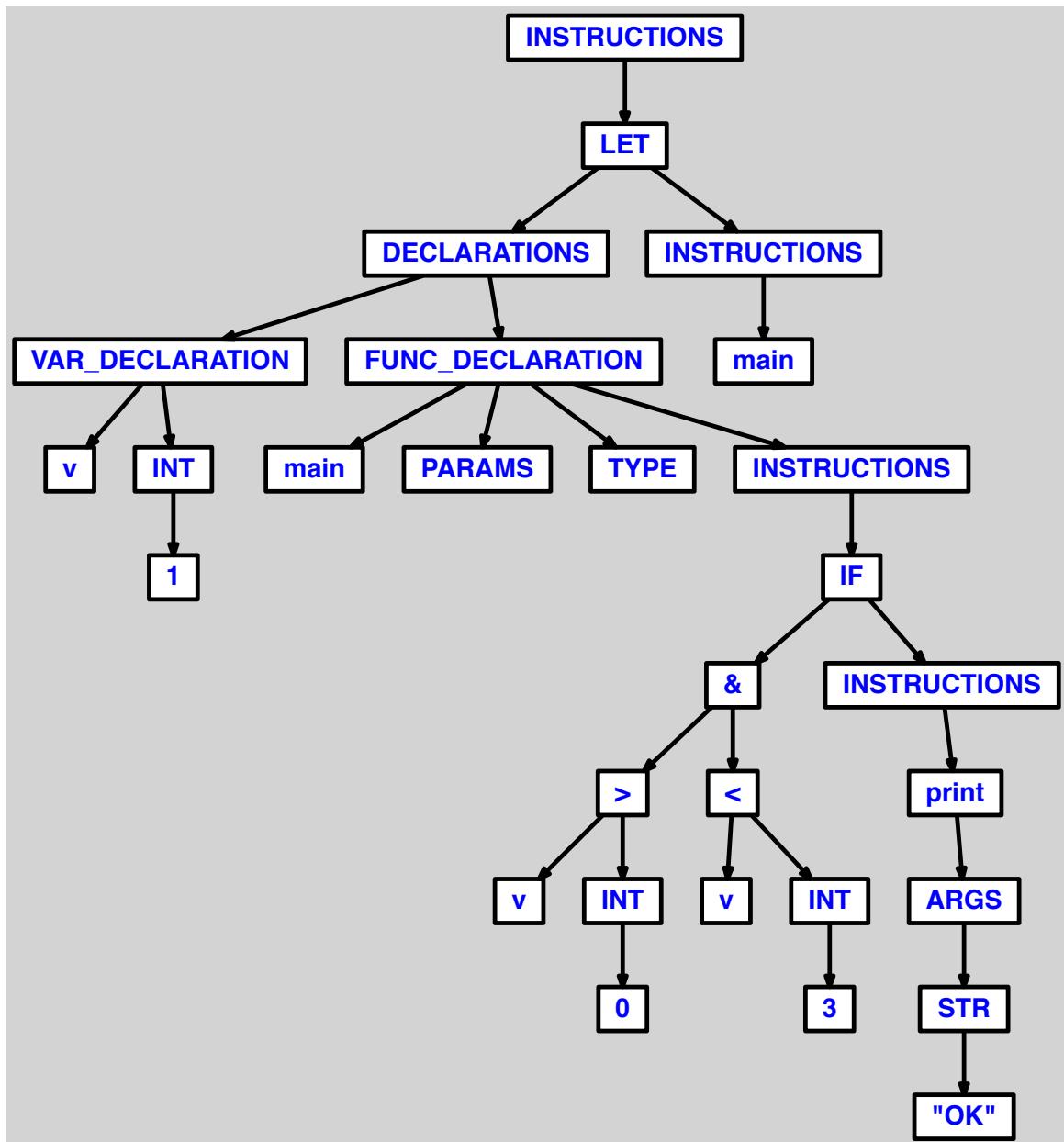


## 1.2 OK

### 1.2.1 condition avec 1 &

```
let
    var v := 1

    function main() =
        if v > 0 & v < 3 then print("OK")
in main() end
```



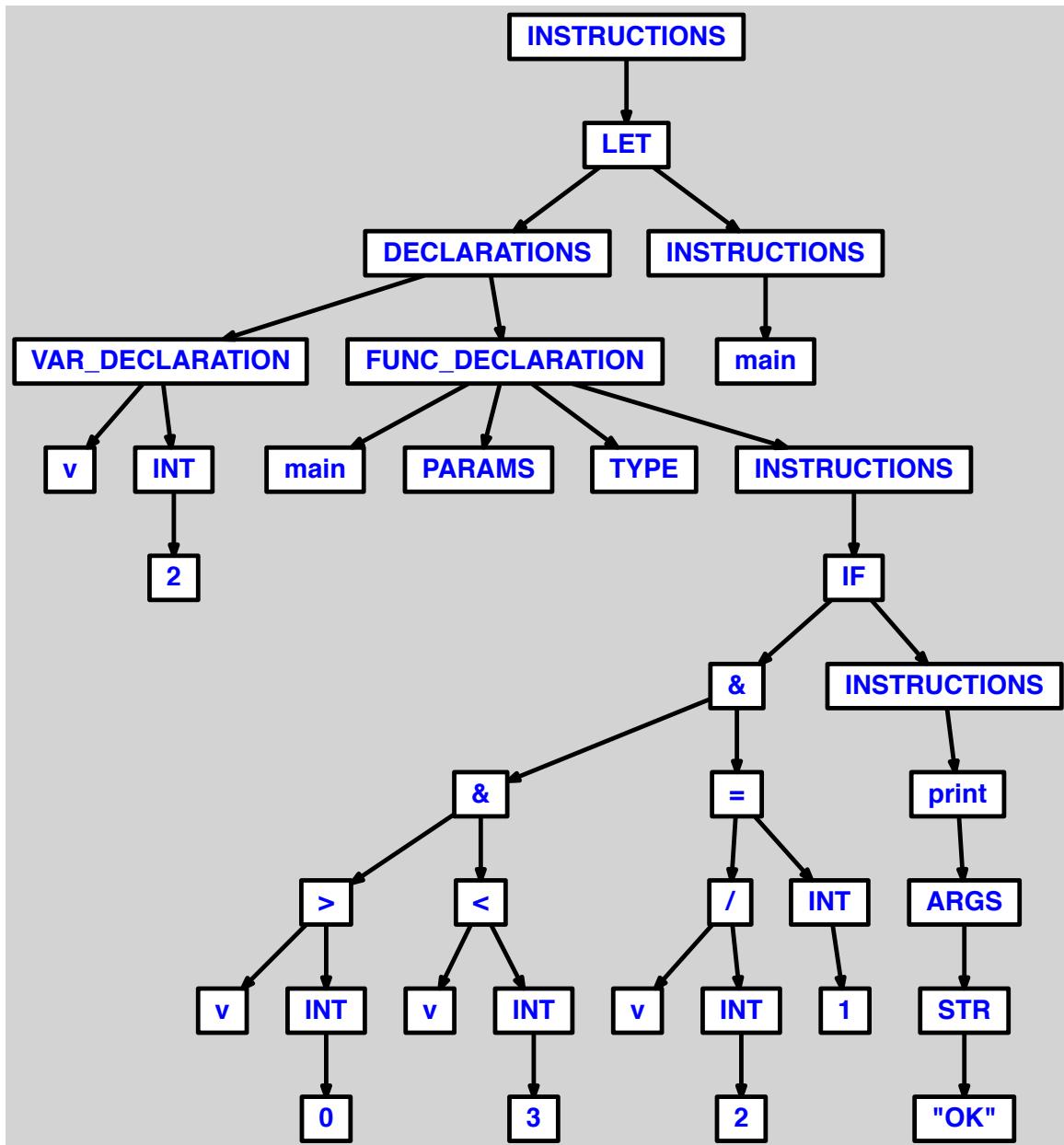
### 1.2.2 condition avec 2 &

let

```
var v := 2
```

```
function main() =  
    if v > 0 & v < 3 & v/2 = 1 then print("OK")
```

```
in main() end
```

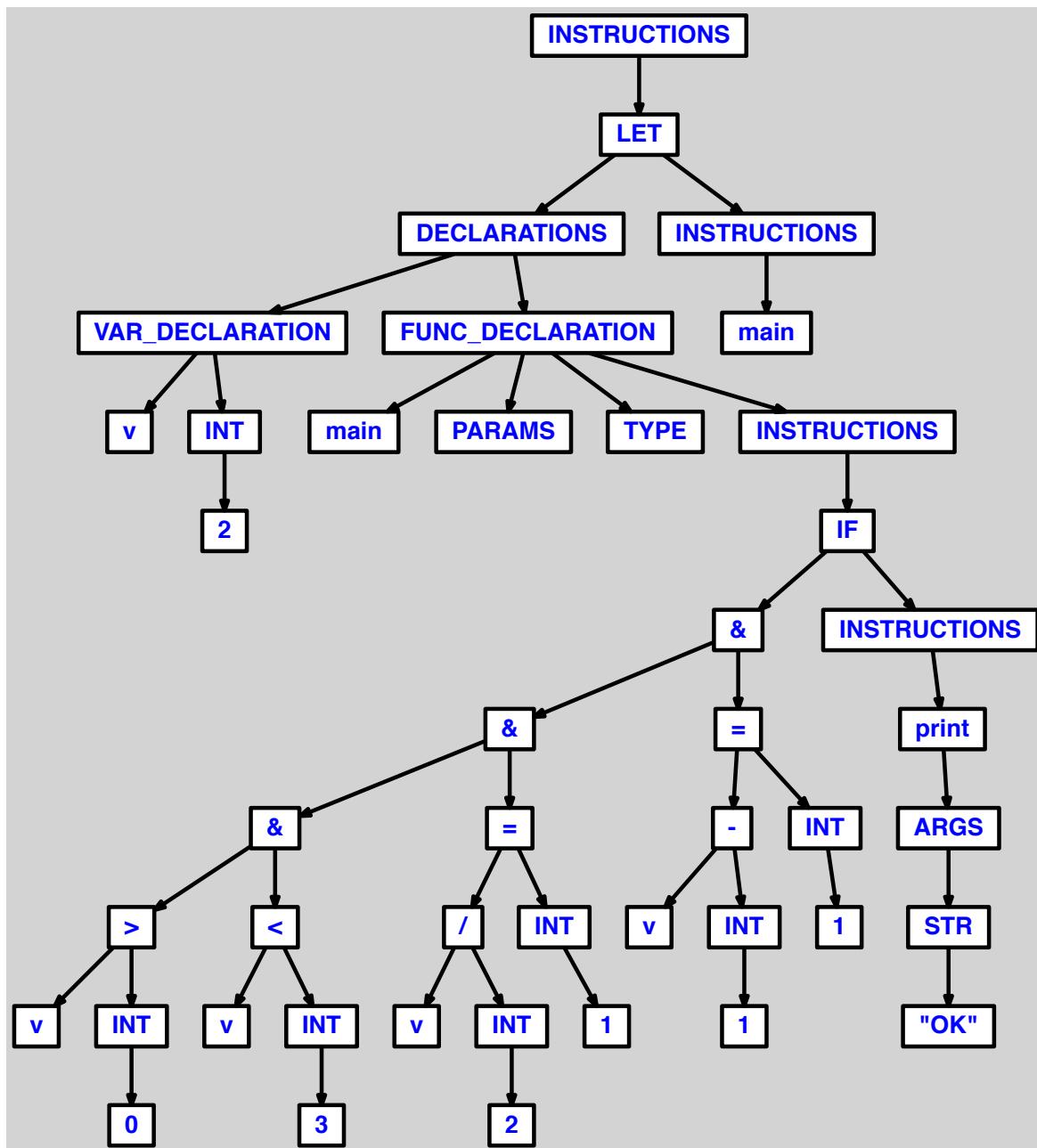


### 1.2.3 condition avec 3 &

let

```
var v := 2

function main() =
    if v > 0 & v < 3 & v/2 = 1 & v-1 = 1 then print("OK")
in main() end
```

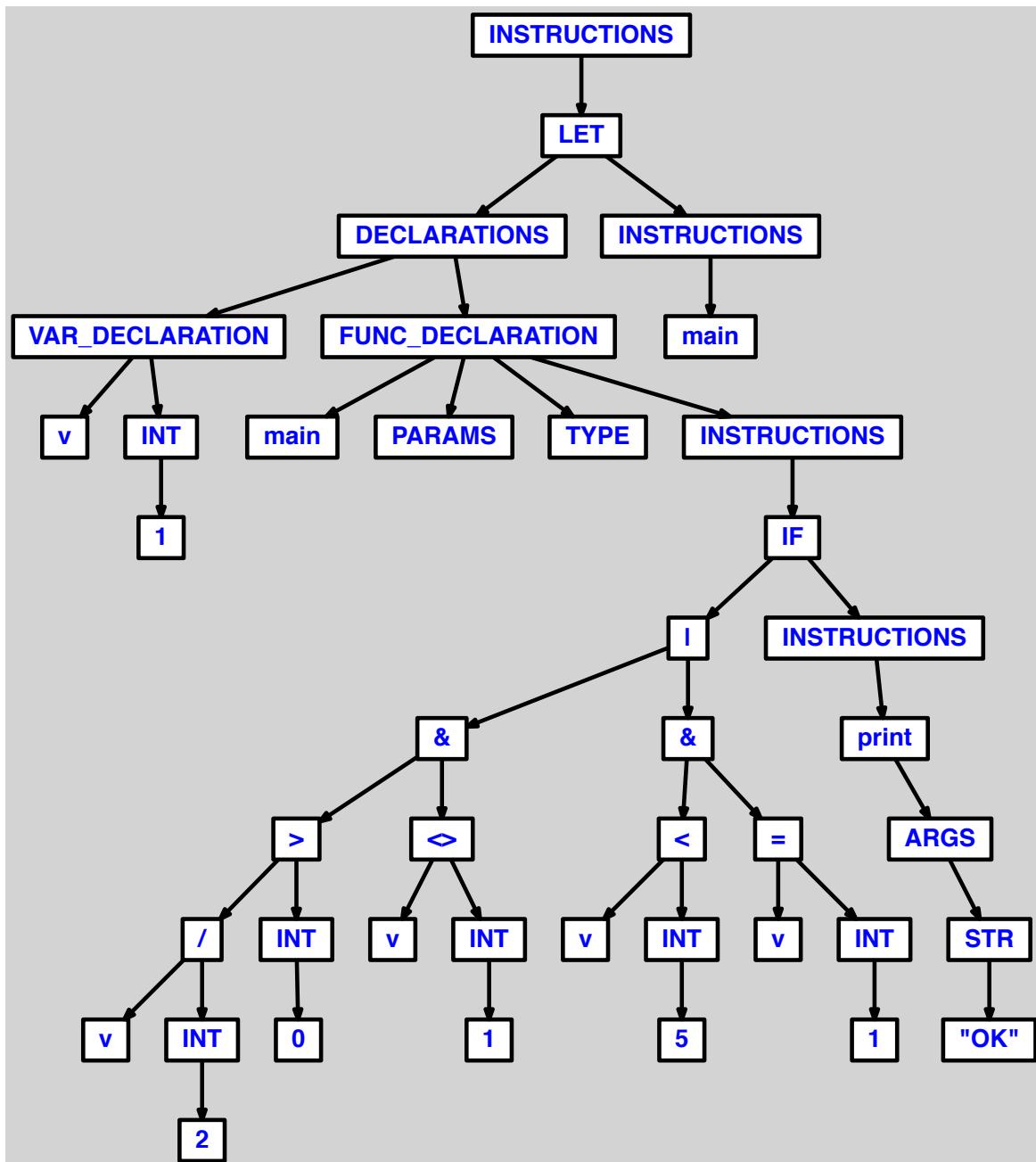


#### 1.2.4 condition avec 2 & et 1 |

let

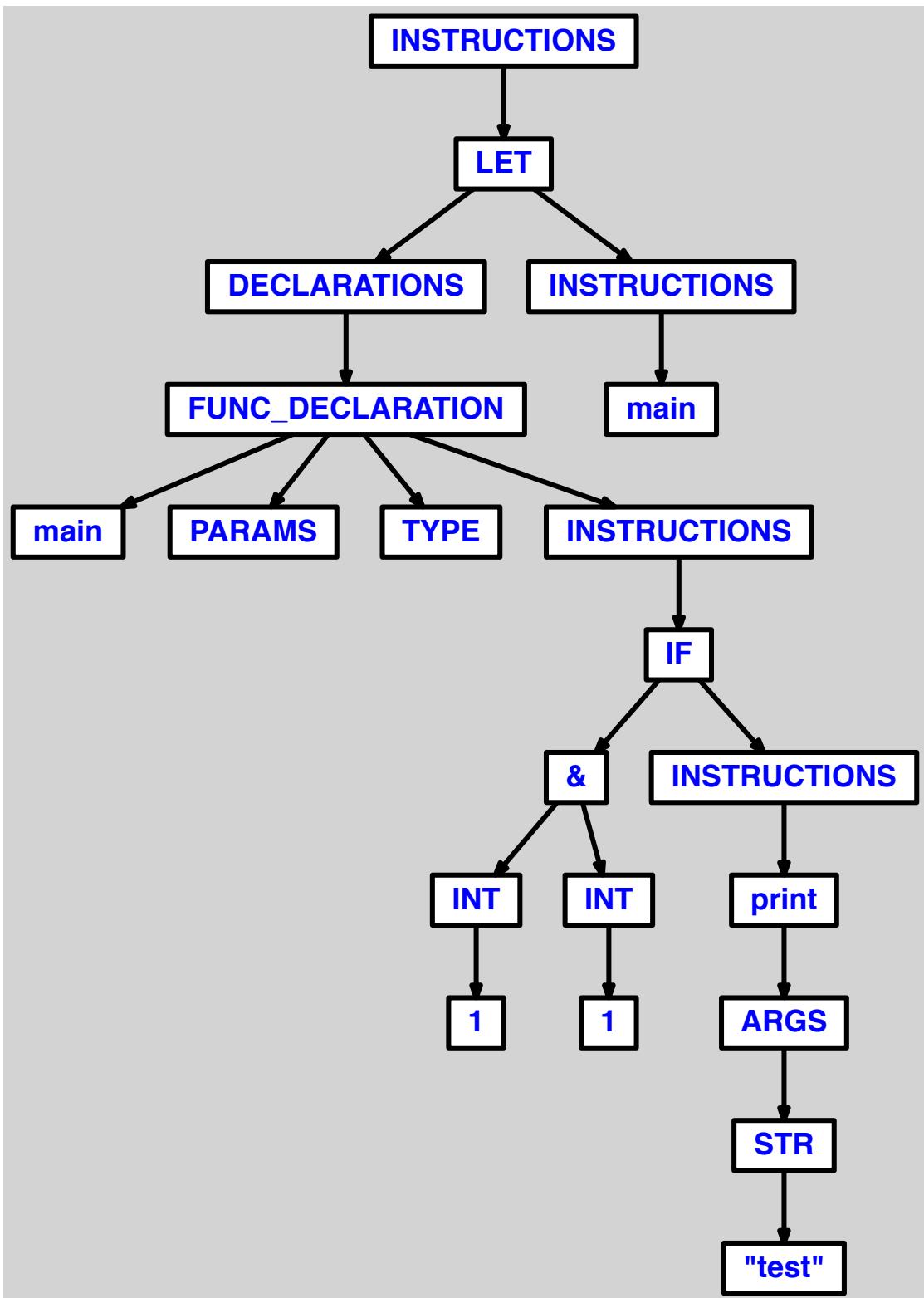
```
var v := 1

function main() =
    if v/2 > 0 & v <> 1 | v < 5 & v = 1 then print("OK")
in main() end
```



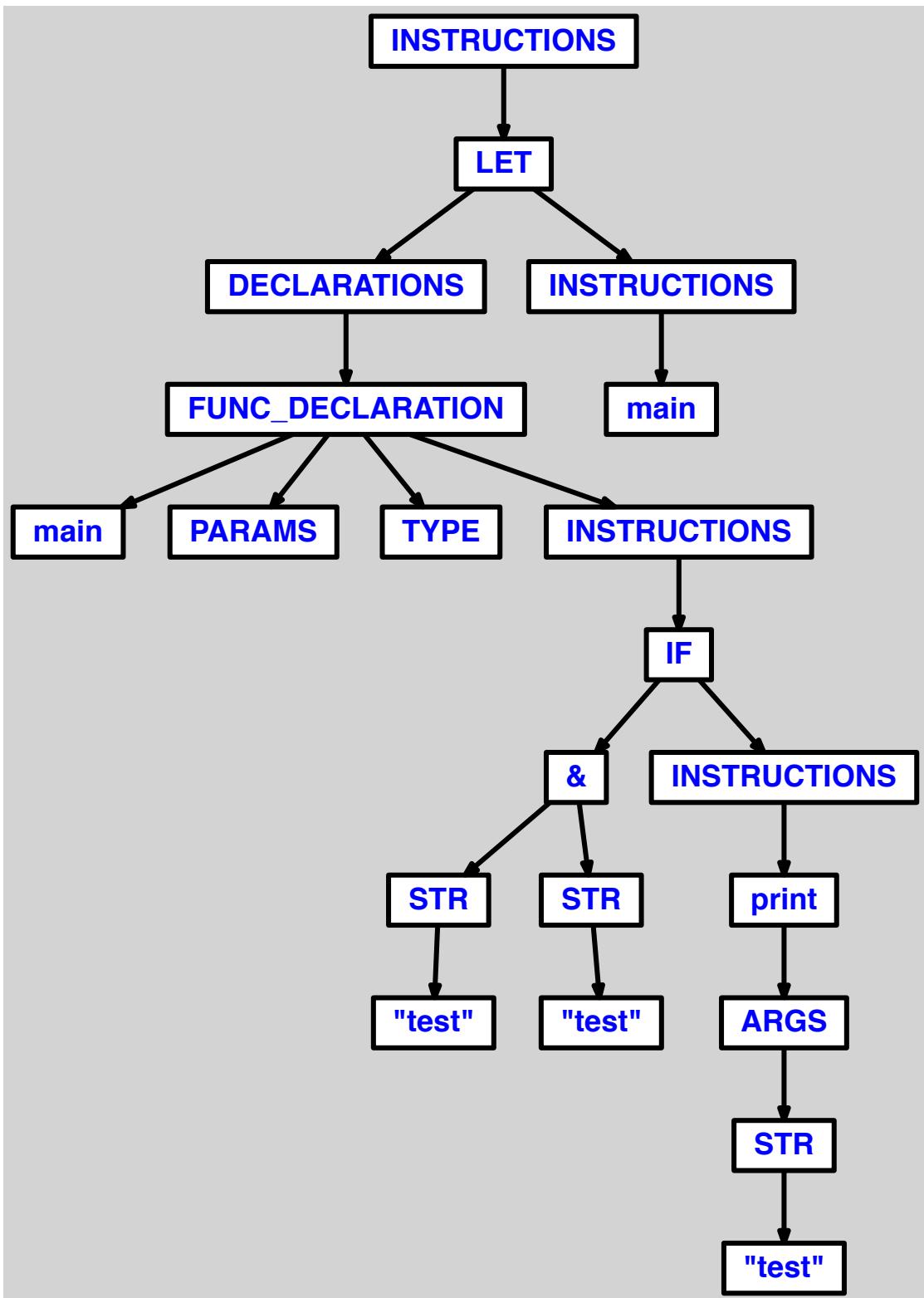
### 1.2.5 & avec des entiers

```
let
    function main() =
        if 1 & 1 then print("test")
in main() end
```



### 1.2.6 & avec des chaines

```
let
    function main() =
        if "test" & "test" then print("test")
in main() end
```

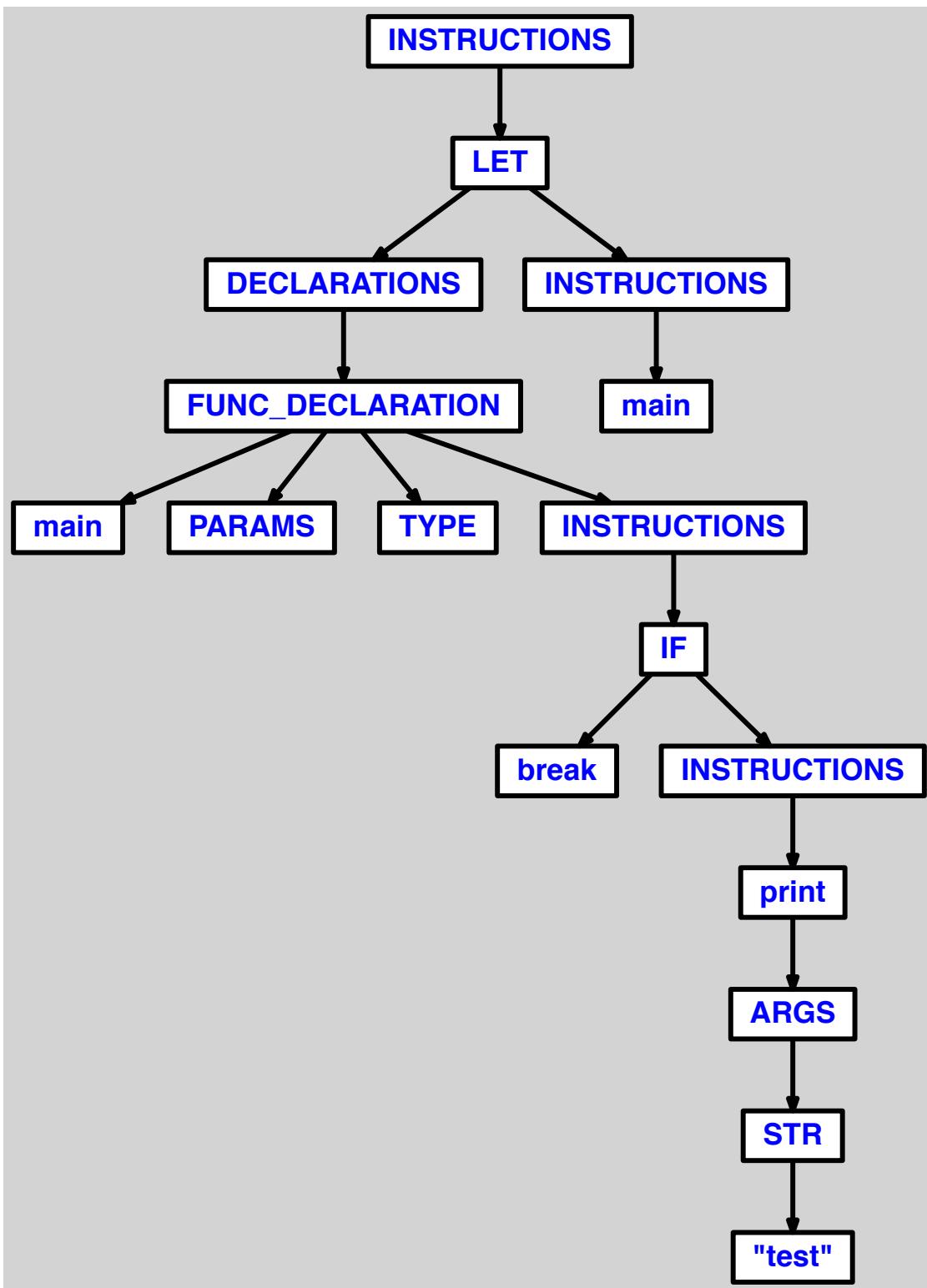


## 2 break

### 2.1 KO

#### 2.1.1 break comme condition dans if

```
let
    function main() =
        if break then
            print("test")
in main() end
```



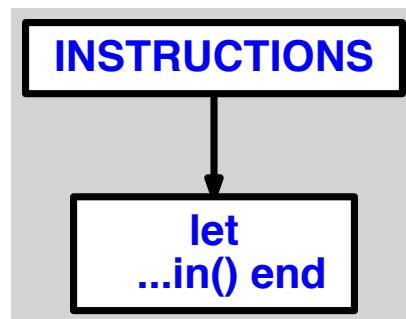
### 2.1.2 break comme condition dans while

```
let
    function main() =
        while break then
            print("test")
in main() end
```

## 2.2 OK

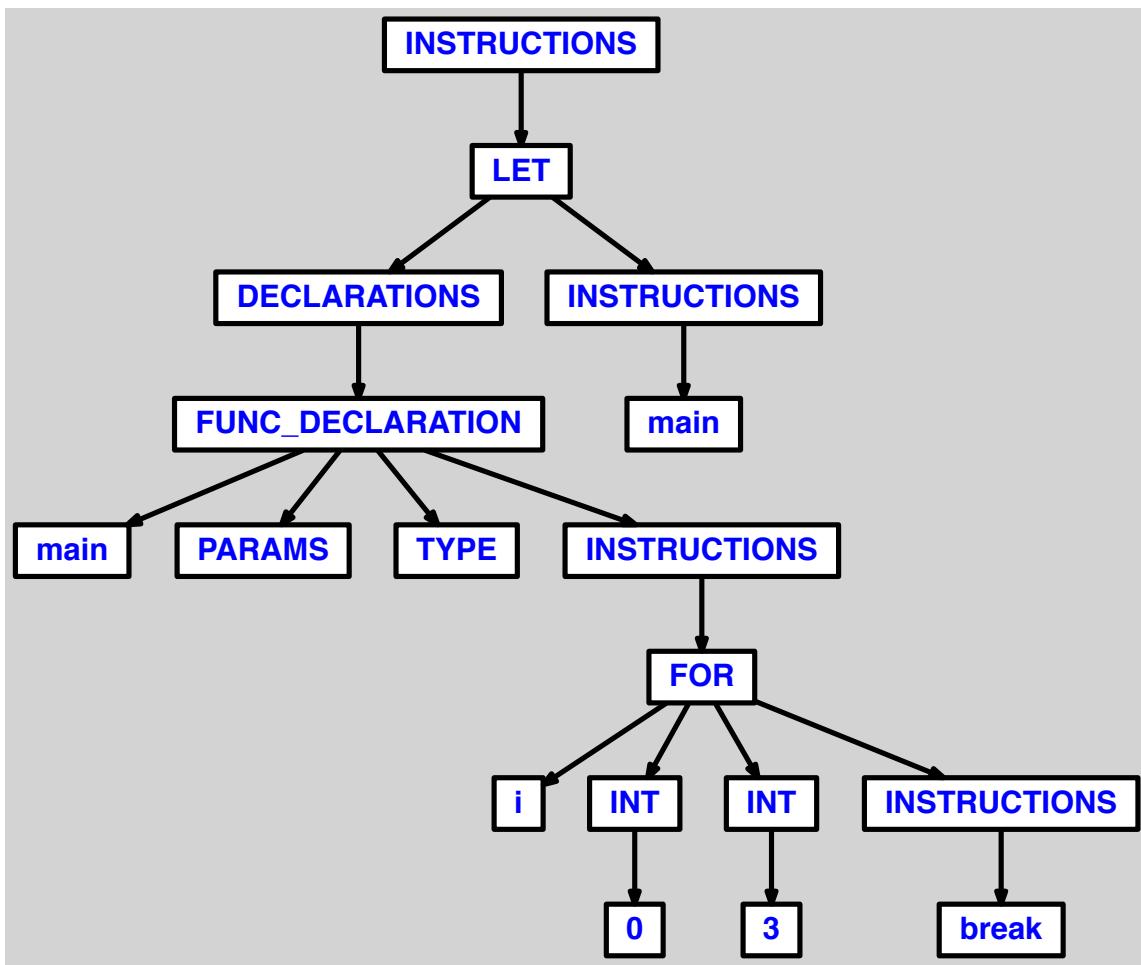
### 2.2.1 break dans un for, apres instruction

```
let
    function main() =
        for i := 0 to 3 do
            print("test");
            break
in main() end
```



### 2.2.2 break dans un for, sans instruction avant

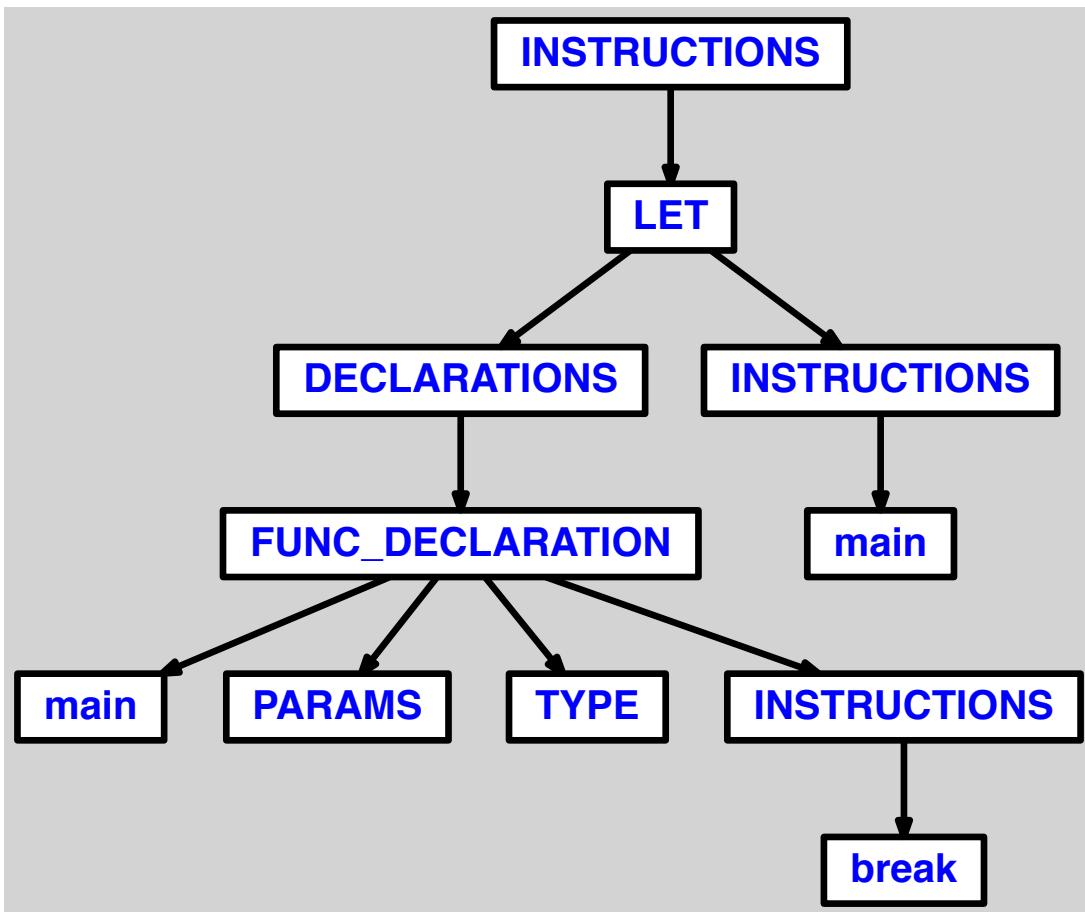
```
let
    function main() =
        for i := 0 to 3 do
            break
in main() end
```



### 2.2.3 fonction executant seulement break

```

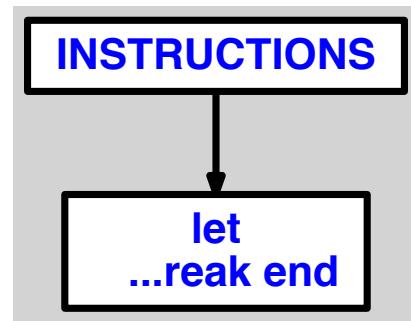
let
    function main() = break
in main() end
  
```



#### 2.2.4 appel de break, avec corps de programme contenant break

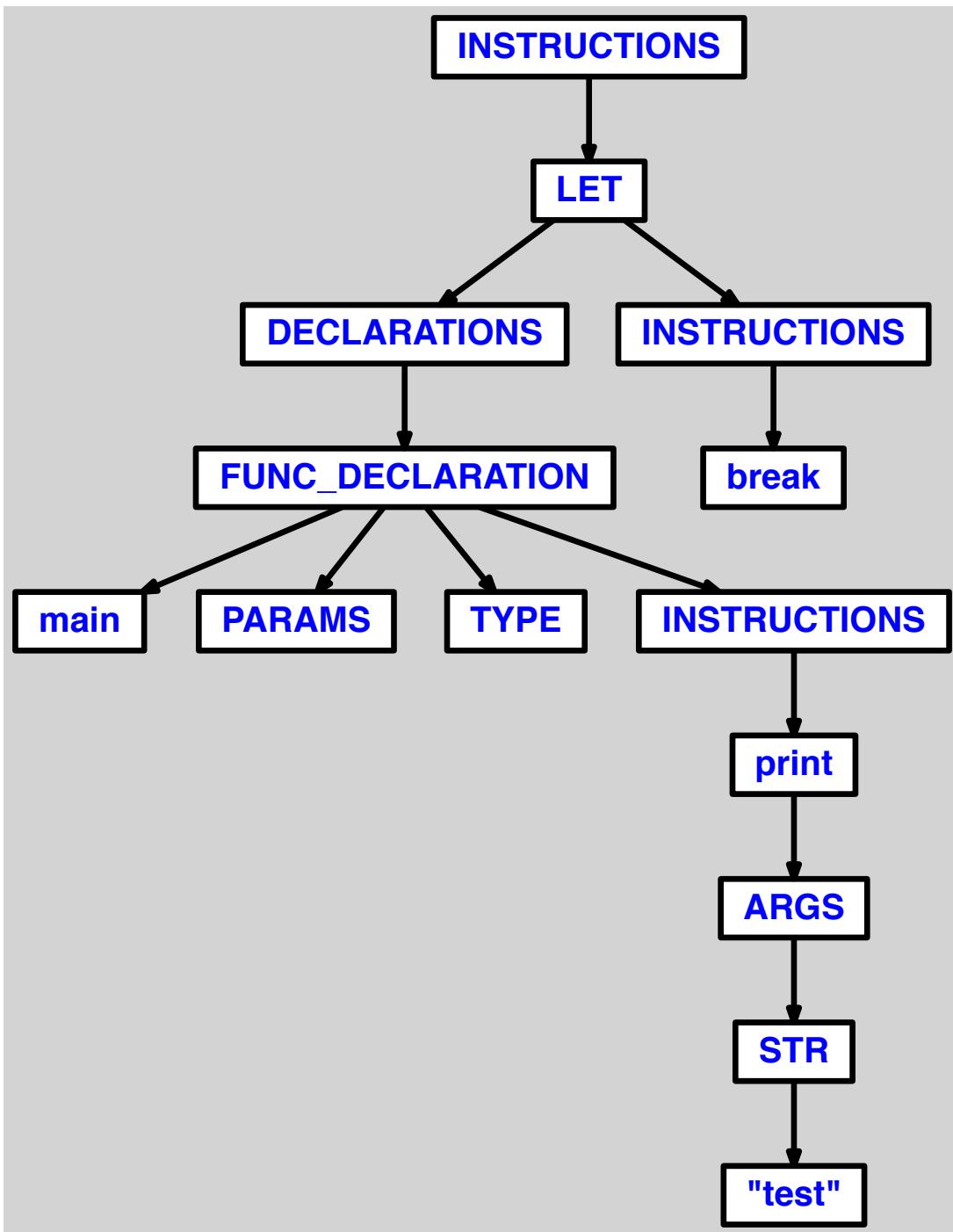
```

let
  break
in break end
  
```



#### 2.2.5 appel de break, avec corps de programme ne contenant pas break

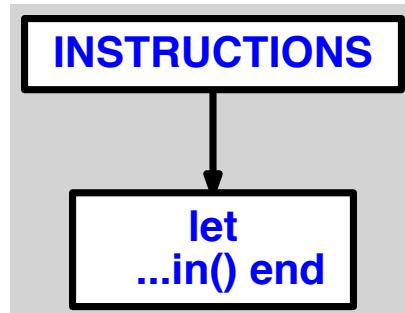
```
let
    function main() = print("test")
in break end
```



#### 2.2.6 break dans un while, apres instruction

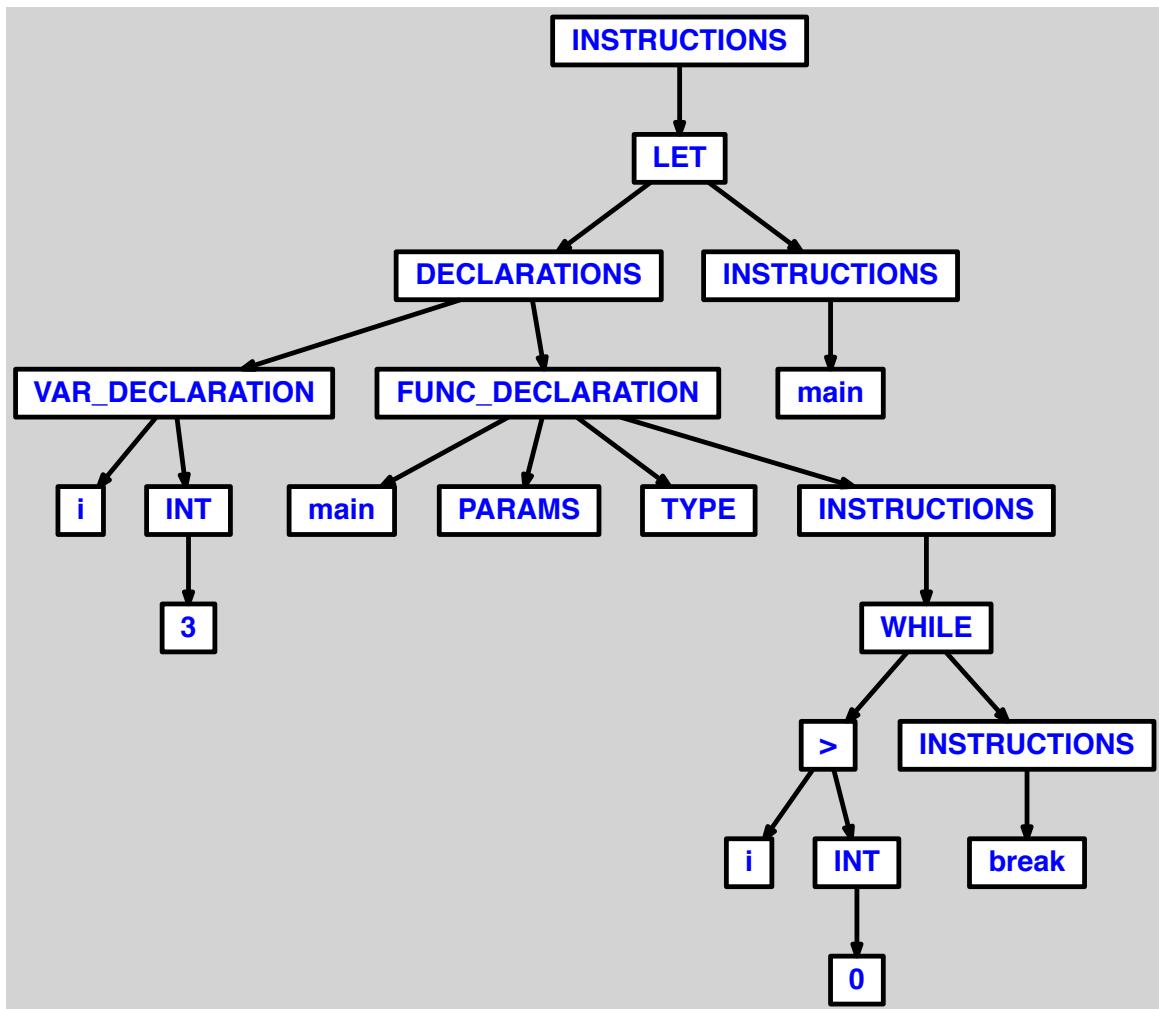
```
let
    var i := 3
```

```
function main() =  
    while i > 0 do  
        print("test");  
        break  
in main() end
```



### 2.2.7 break dans un while, sans instruction avant

```
let  
    var i := 3  
  
    function main() =  
        while i > 0 do  
            break  
    in main() end
```

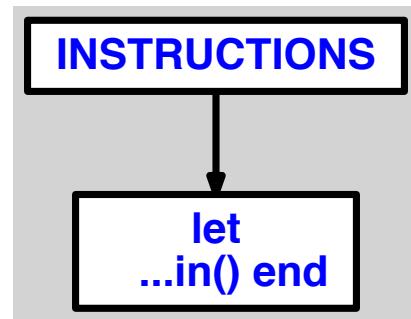


## 2.2.8 break dans un if then, apres instruction

```

let
  var i := 3

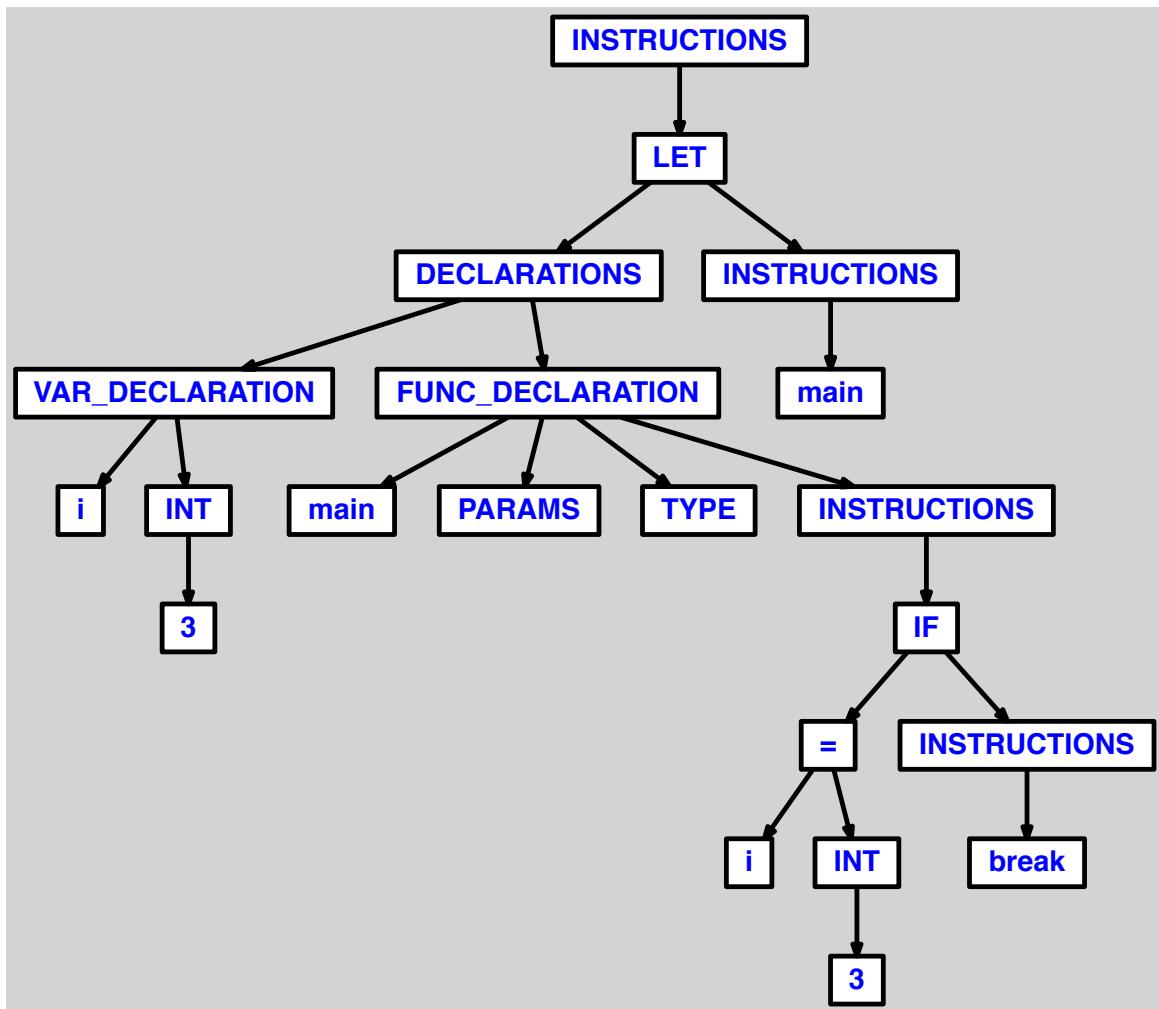
  function main() =
    if i=3 then
      print("test");
      break
in main() end
  
```



#### 2.2.9 break dans un if then, sans instruction avant

```
let
    var i := 3

    function main() =
        if i=3 then
            break
in main() end
```

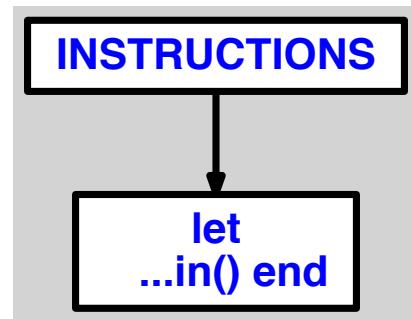


### 2.2.10 break dans un if then else, apres instruction

```

let
var i := 3

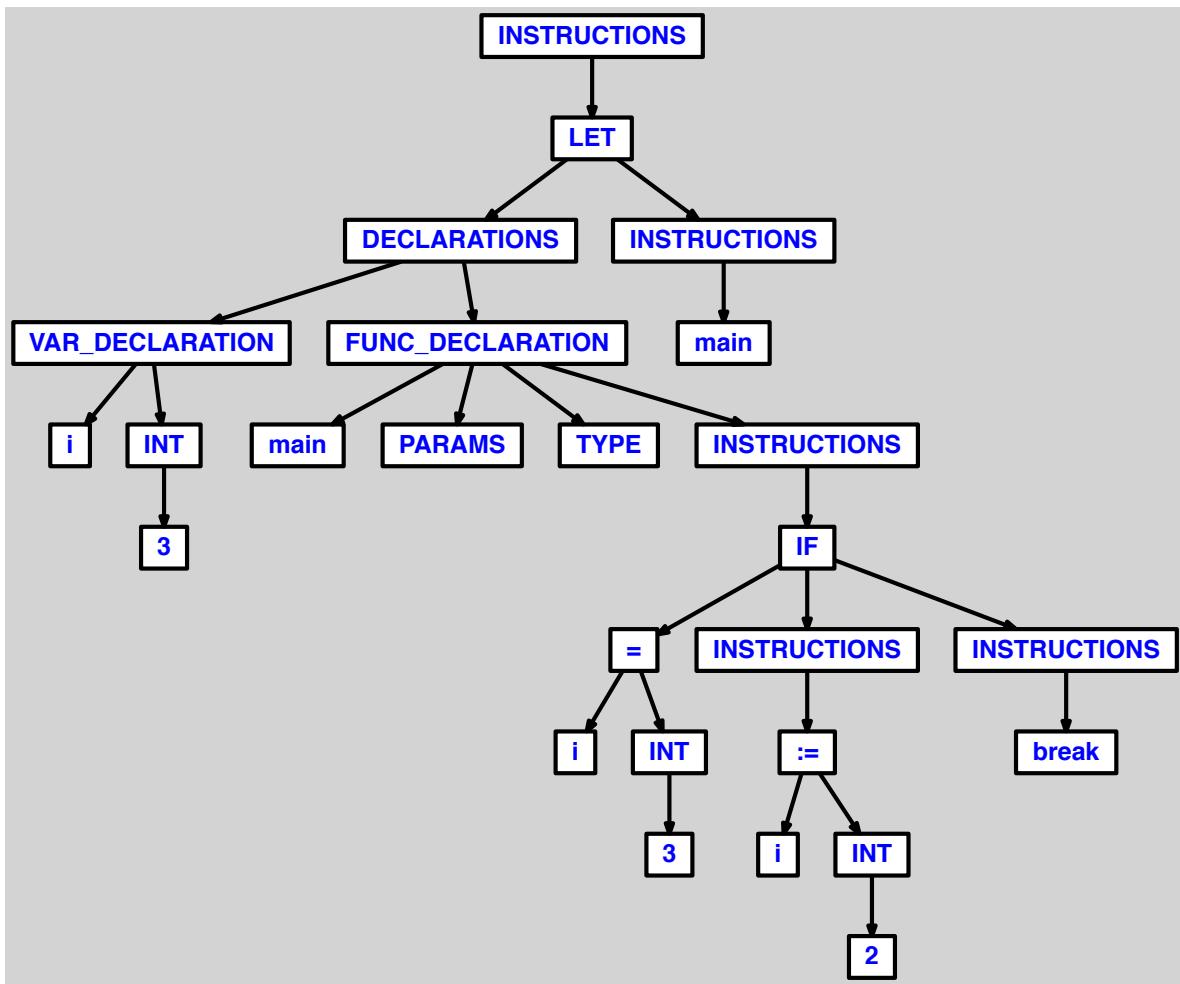
function main() =
    if i=3 then
        i := 2
    else
        print("test");
        break
in main() end
  
```



#### 2.2.11 break dans un if then else, sans instruction avant

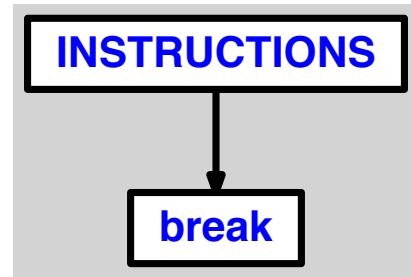
```
let
    var i := 3

    function main() =
        if i=3 then
            i := 2
        else
            break
    in main() end
```



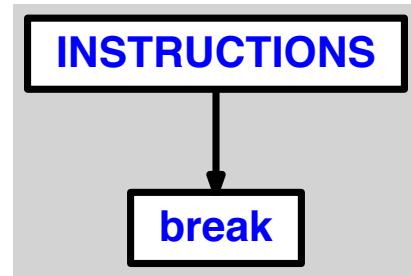
### 2.2.12 break seul

break



#### 2.2.13 break seul, apres saut de ligne

```
break
```

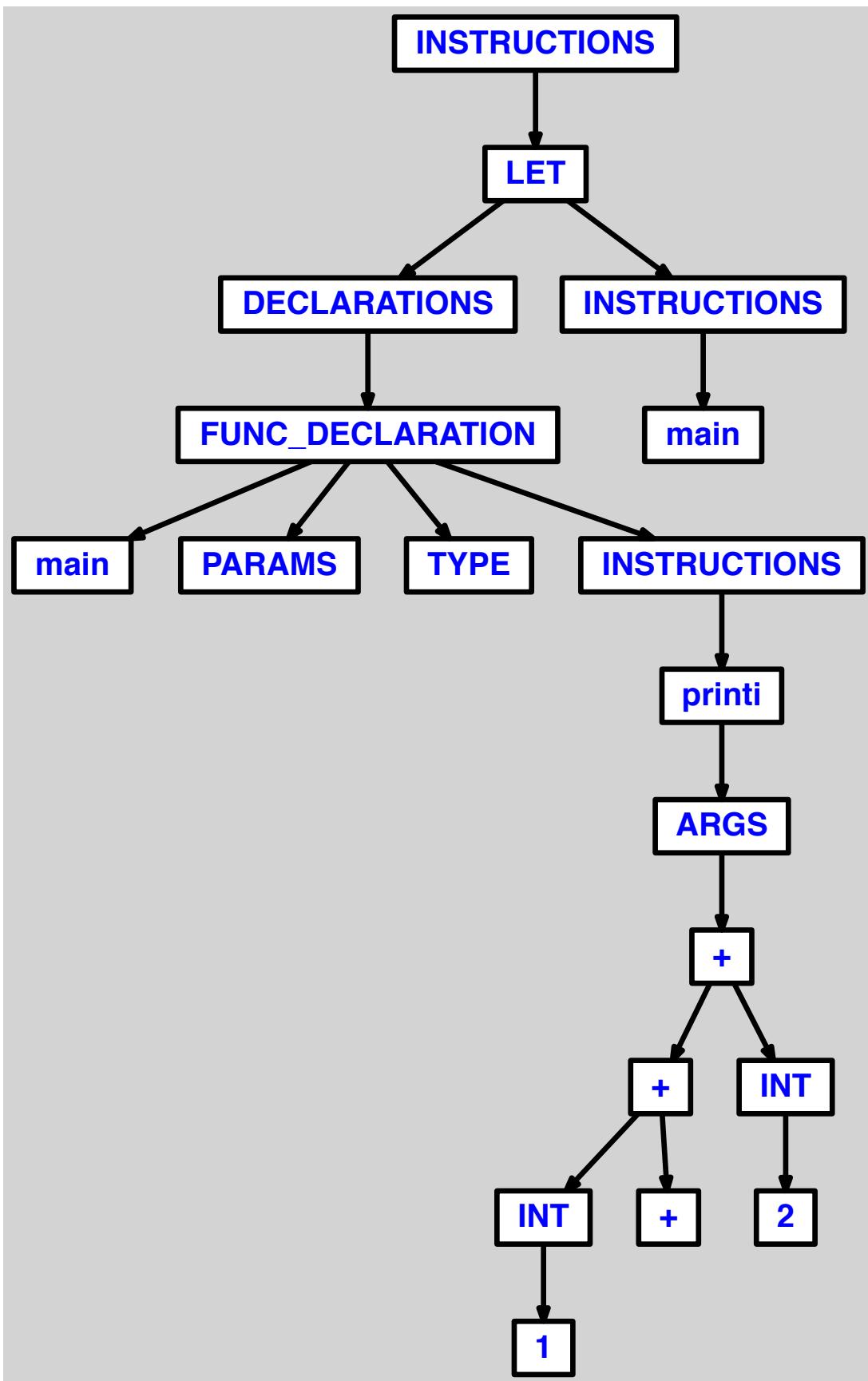


## 3 calculation

### 3.1 KO

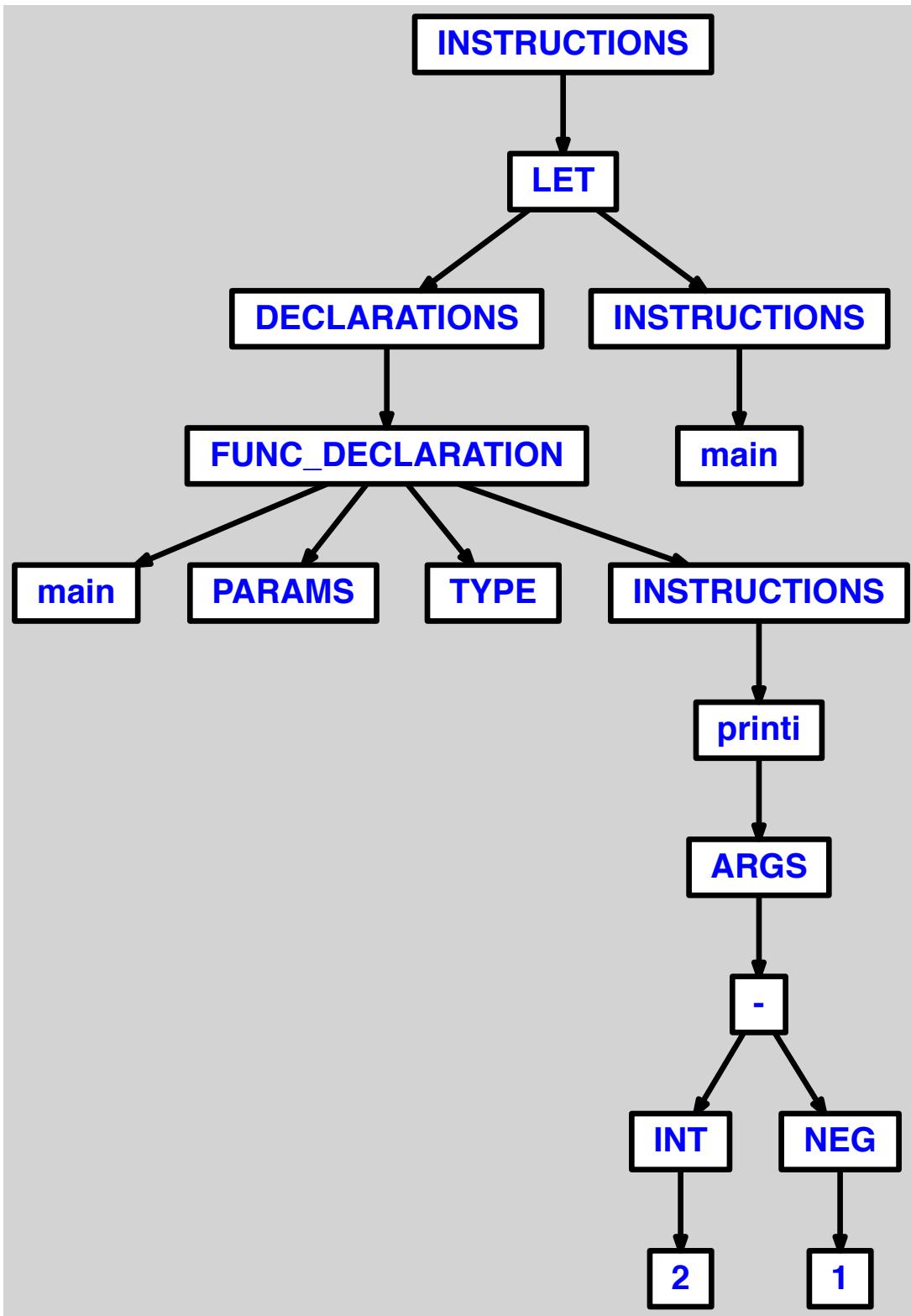
#### 3.1.1 addition avec operateur mal écrit

```
let
    function main() = printi(1++2)
in main() end
```



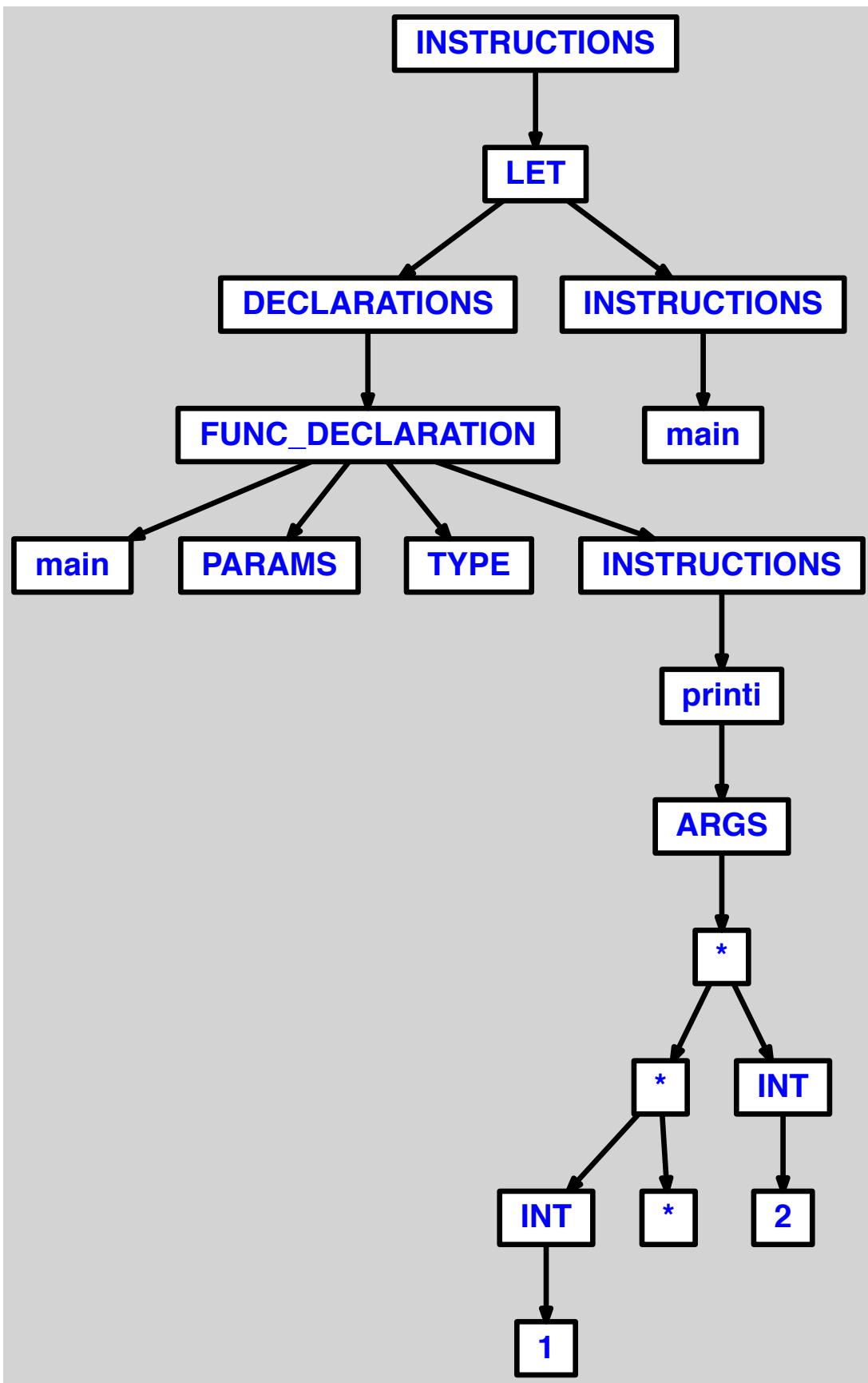
### 3.1.2 soustraction avec operateur mal écrit

```
let
    function main() = printi(2--1)
in main() end
```



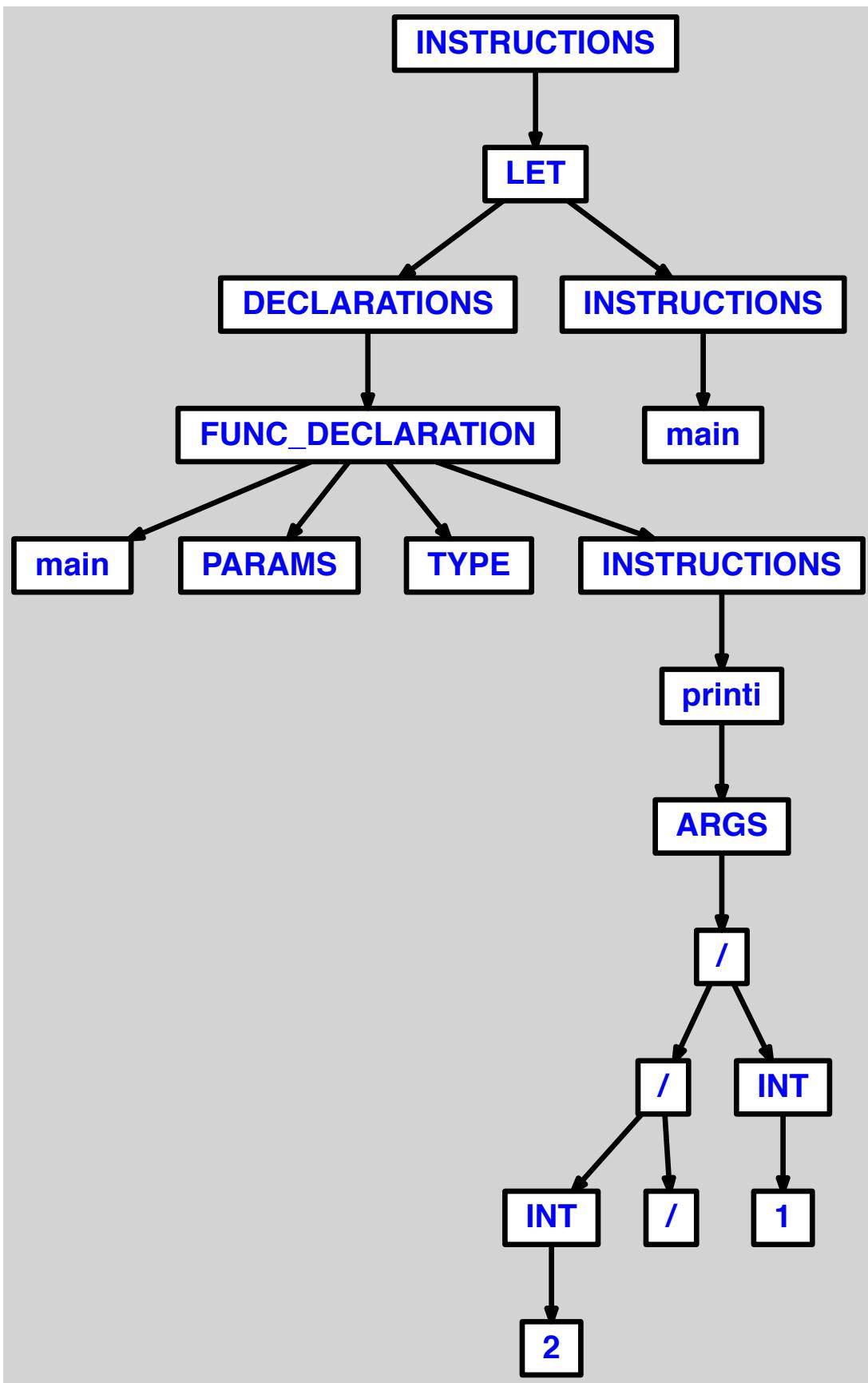
### 3.1.3 multiplication avec operateur mal ecrit

```
let
    function main() = printi(1**2)
in main() end
```



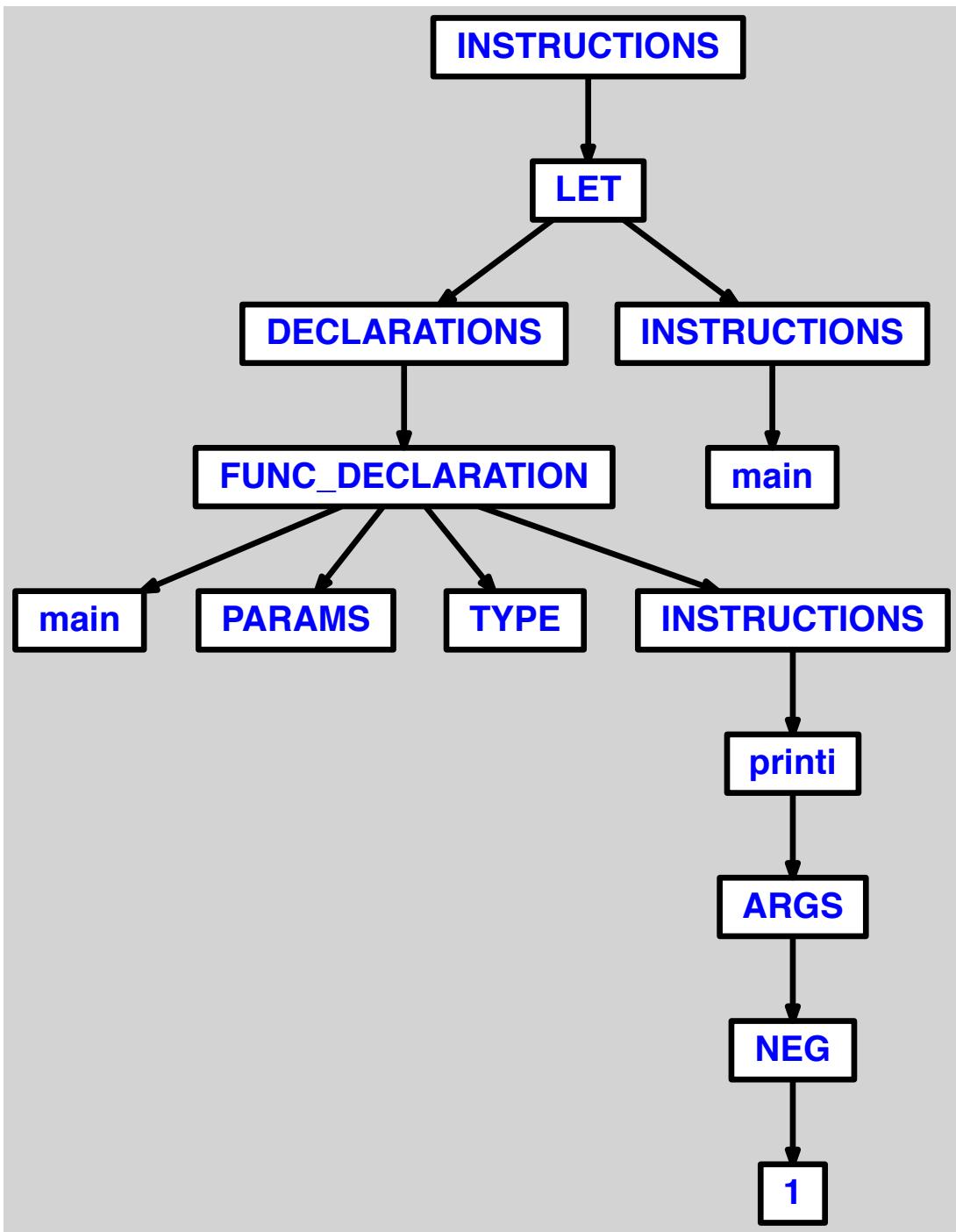
### 3.1.4 division avec operateur mal écrit

```
let
    function main() = printi(2//1)
in main() end
```



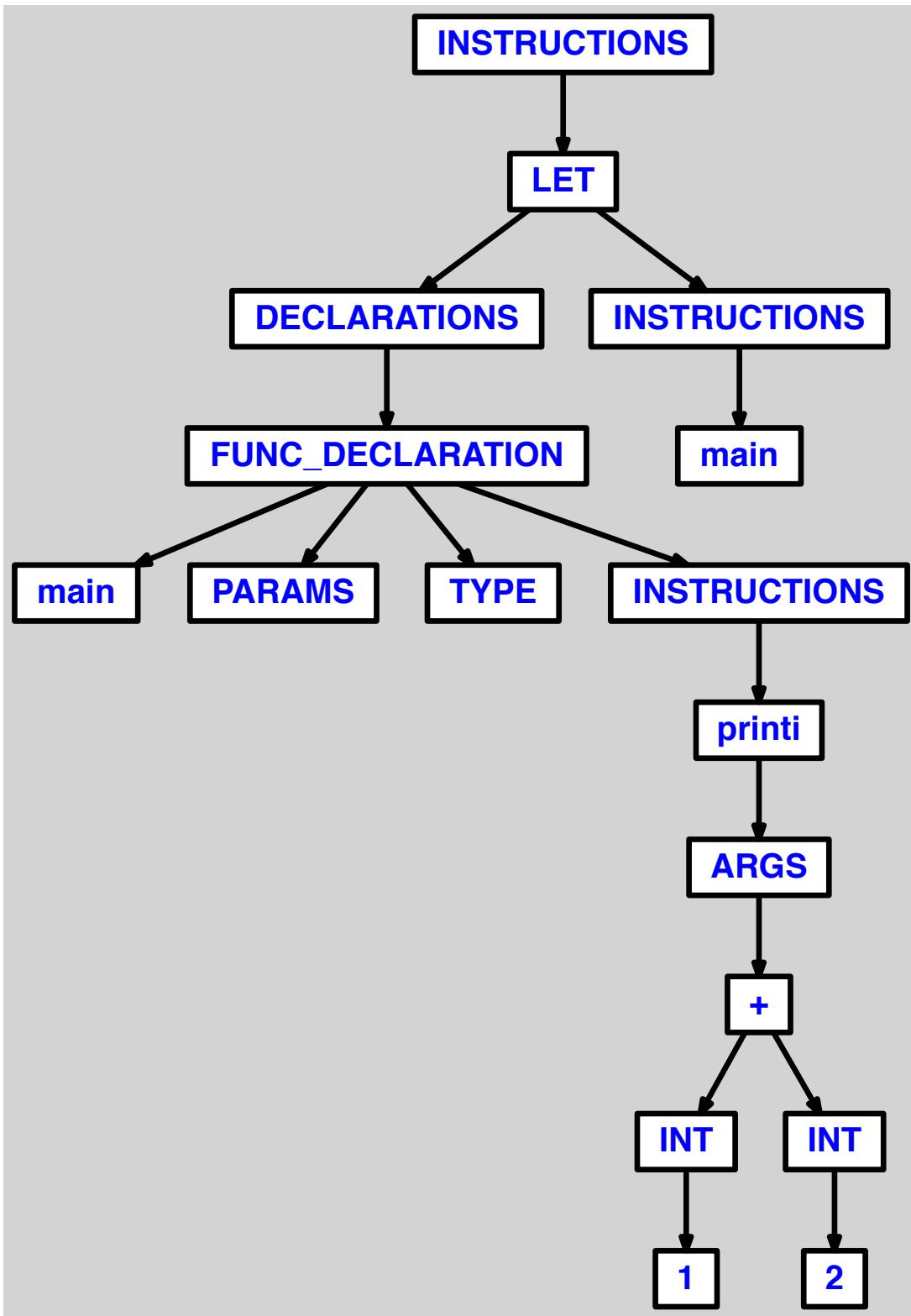
### 3.1.5 moins unaire avec operateur mal ecrit

```
let
    function main() = printi(--1)
in main() end
```



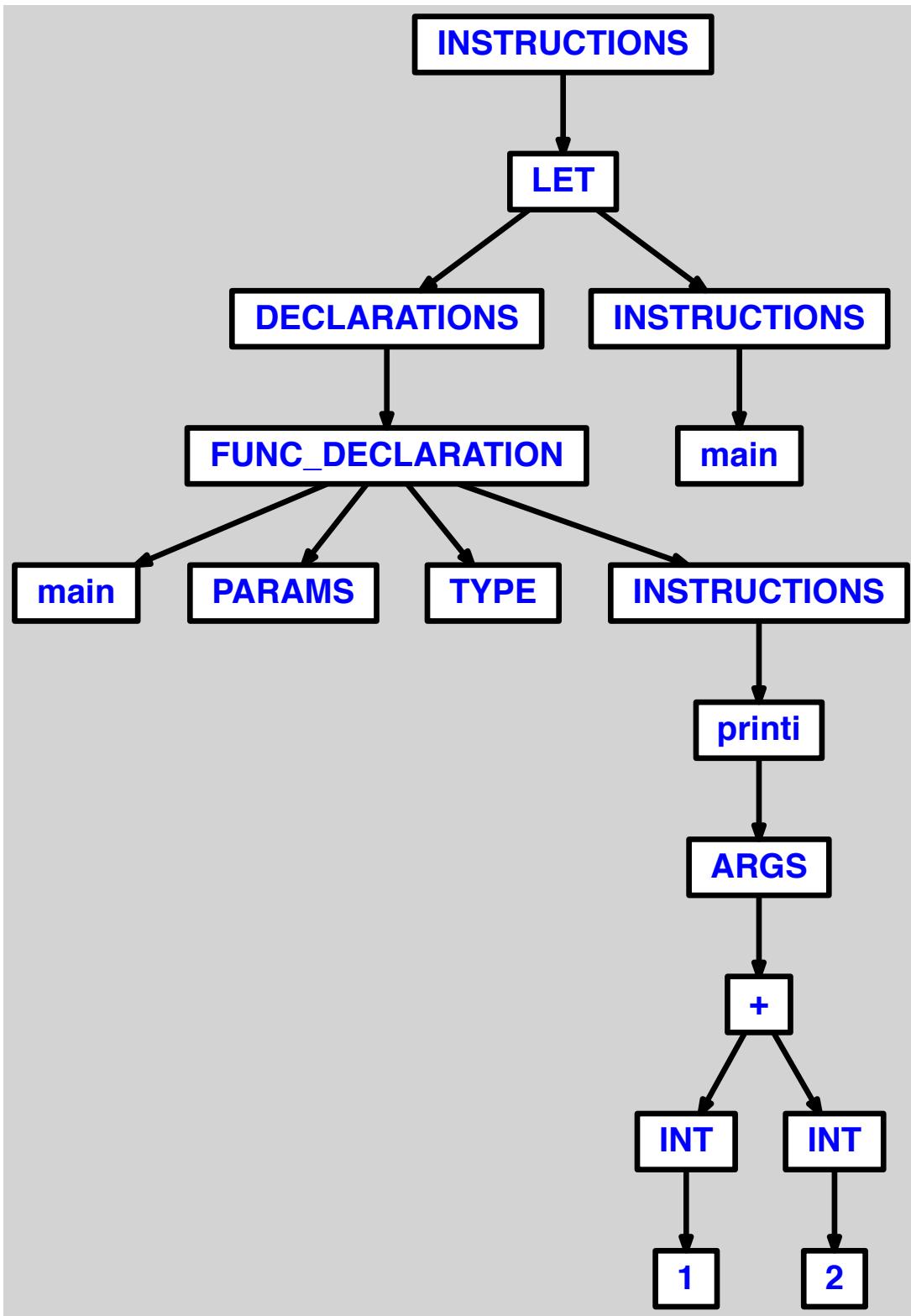
### 3.1.6 calcul a 2 termes avec oubli de parenthese gauche

```
let function main() = printi(1+2)) in main() end
```



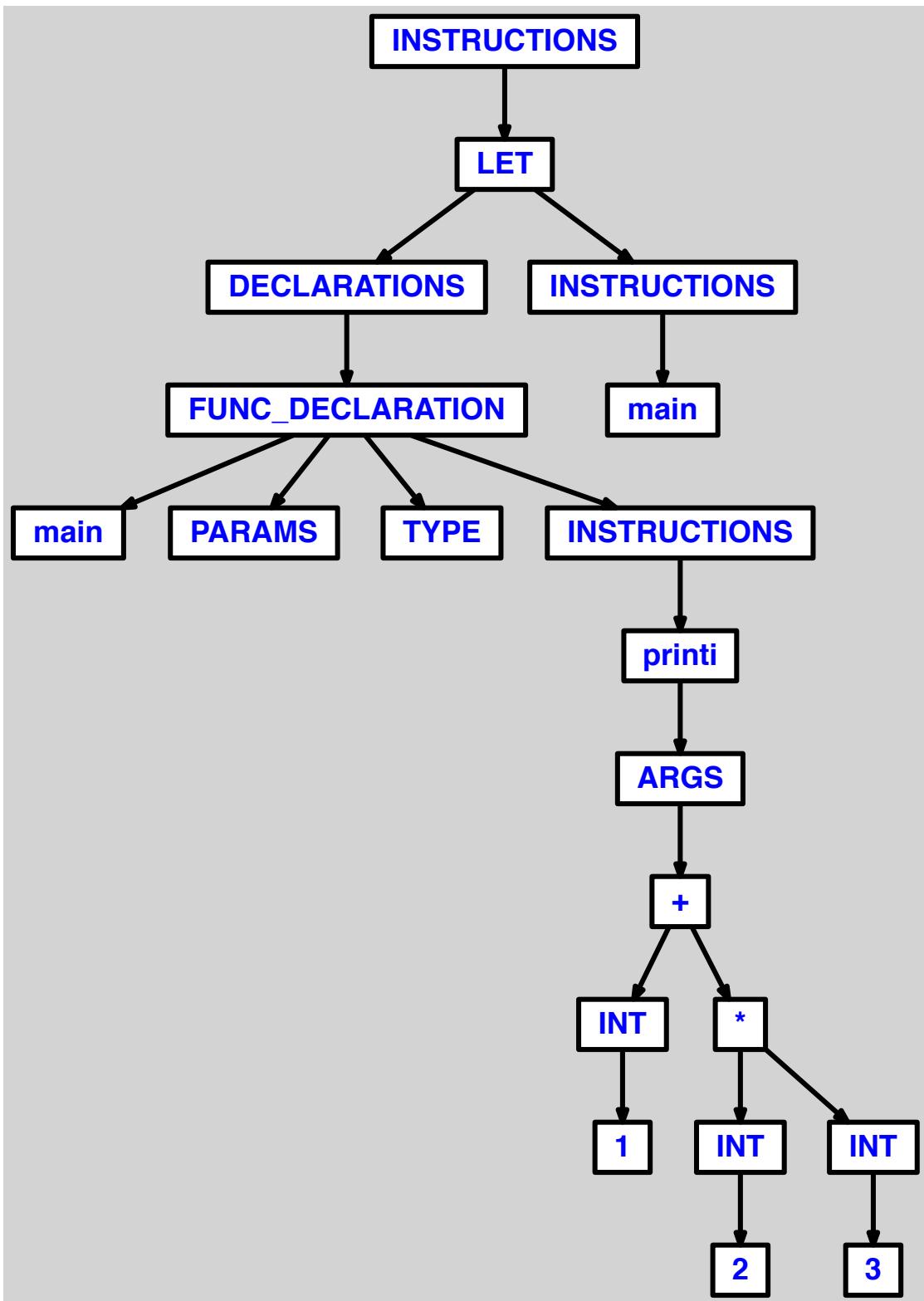
### 3.1.7 calcul a 2 termes avec oubli de parenthese droite

```
let function main() = printi((1+2) in main() end
```



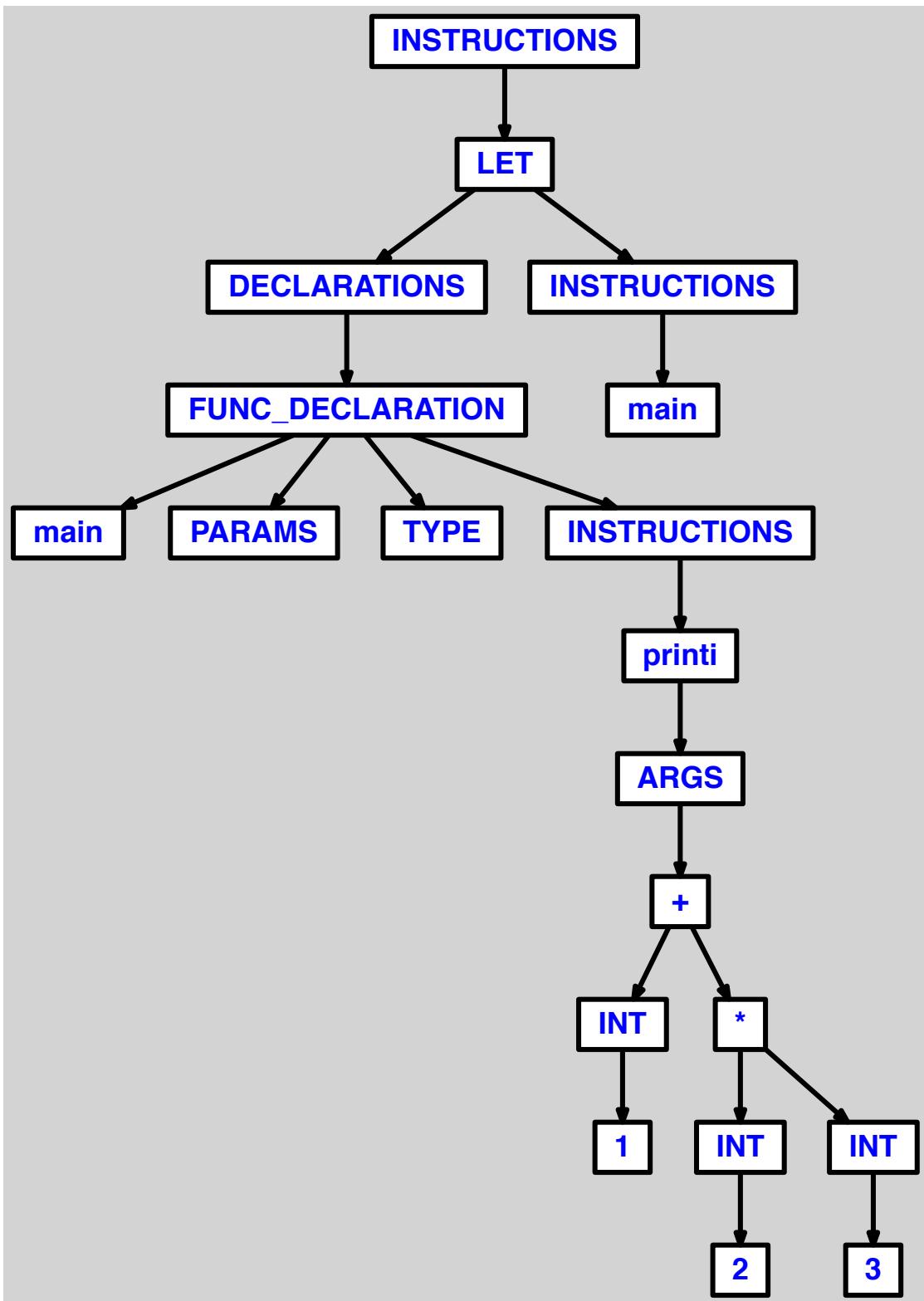
### 3.1.8 calcul a 3 termes avec oubli de parenthese gauche

```
let function main() = printi(1+(2*3)) in main() end
```



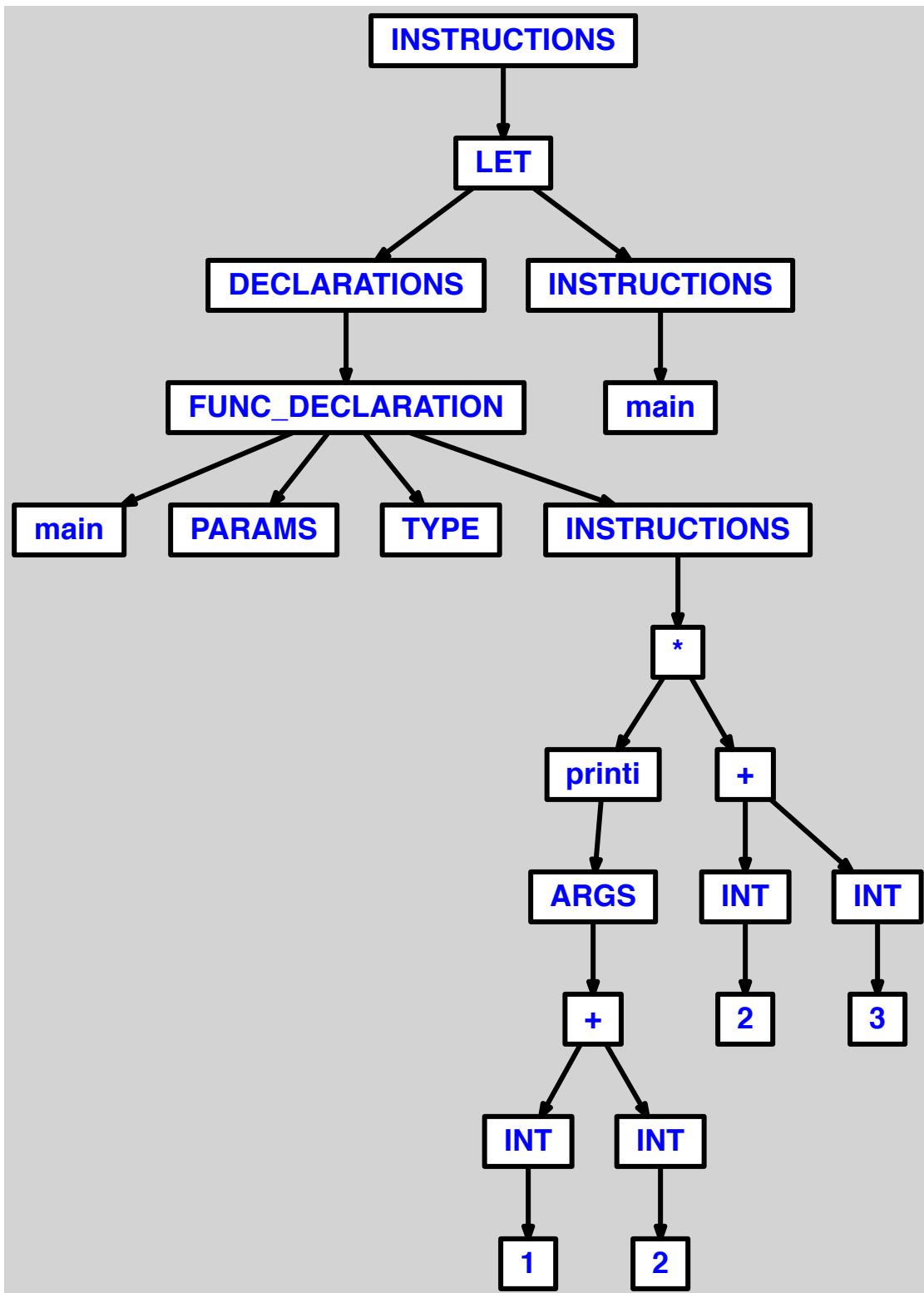
### 3.1.9 calcul a 3 termes avec oubli de parenthese droite

```
let function main() = printi((1+(2*3)) in main() end
```



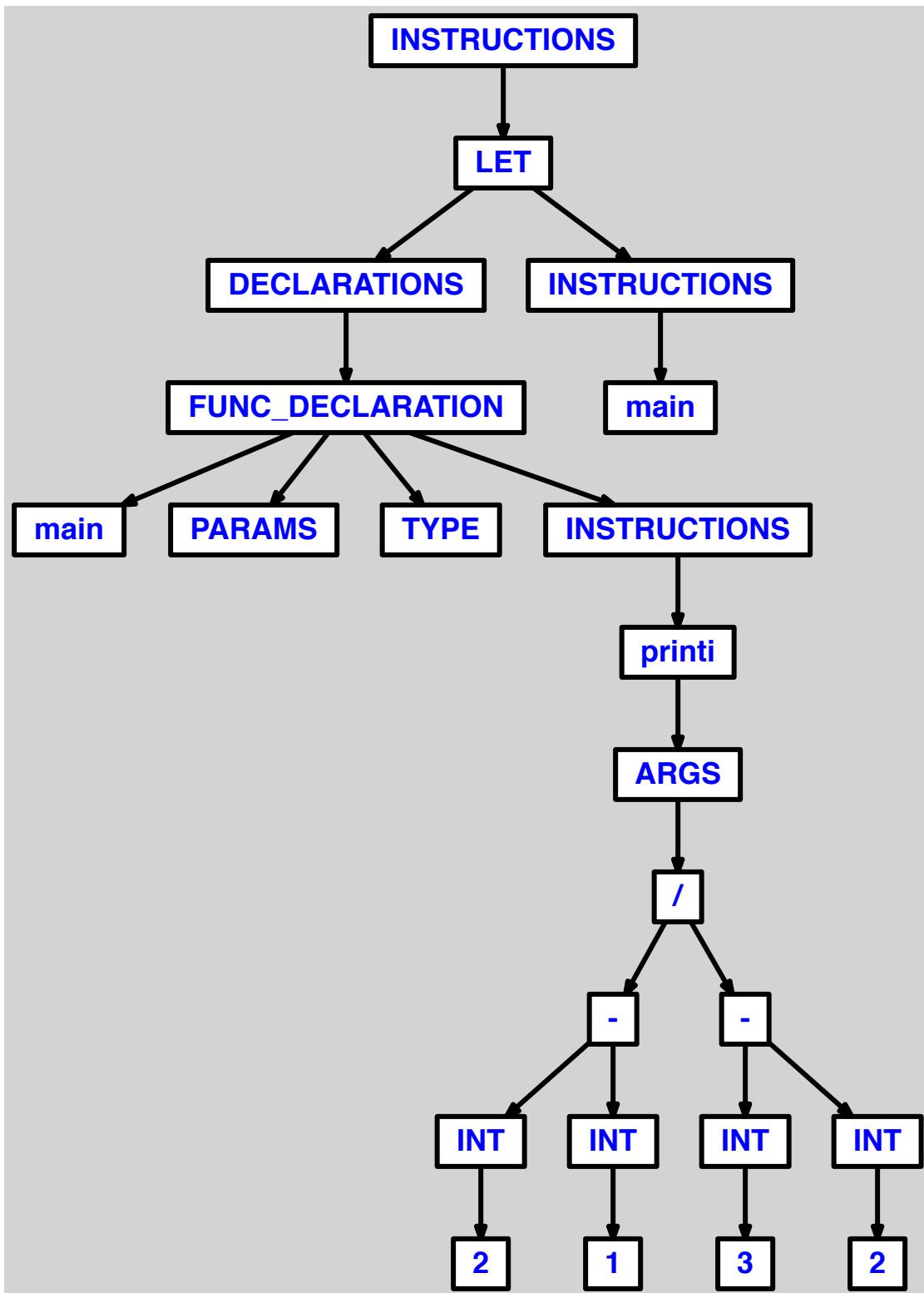
### 3.1.10 calcul a 4 termes avec oubli de parenthese gauche

```
let function main() = printi(1+2)*(2+3)) in main() end
```



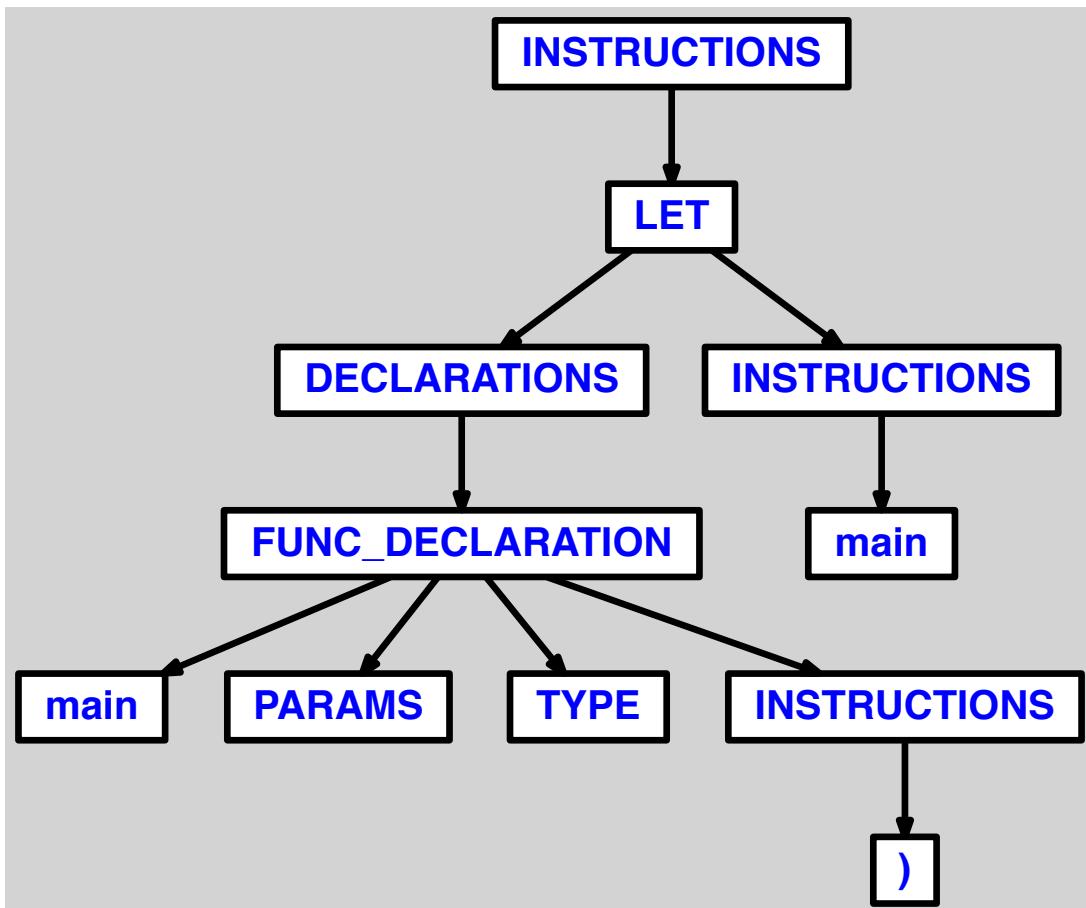
### 3.1.11 calcul a 4 termes avec oubli de parenthese droite

```
let function main() = printi((2-1)/(3-2) in main() end
```



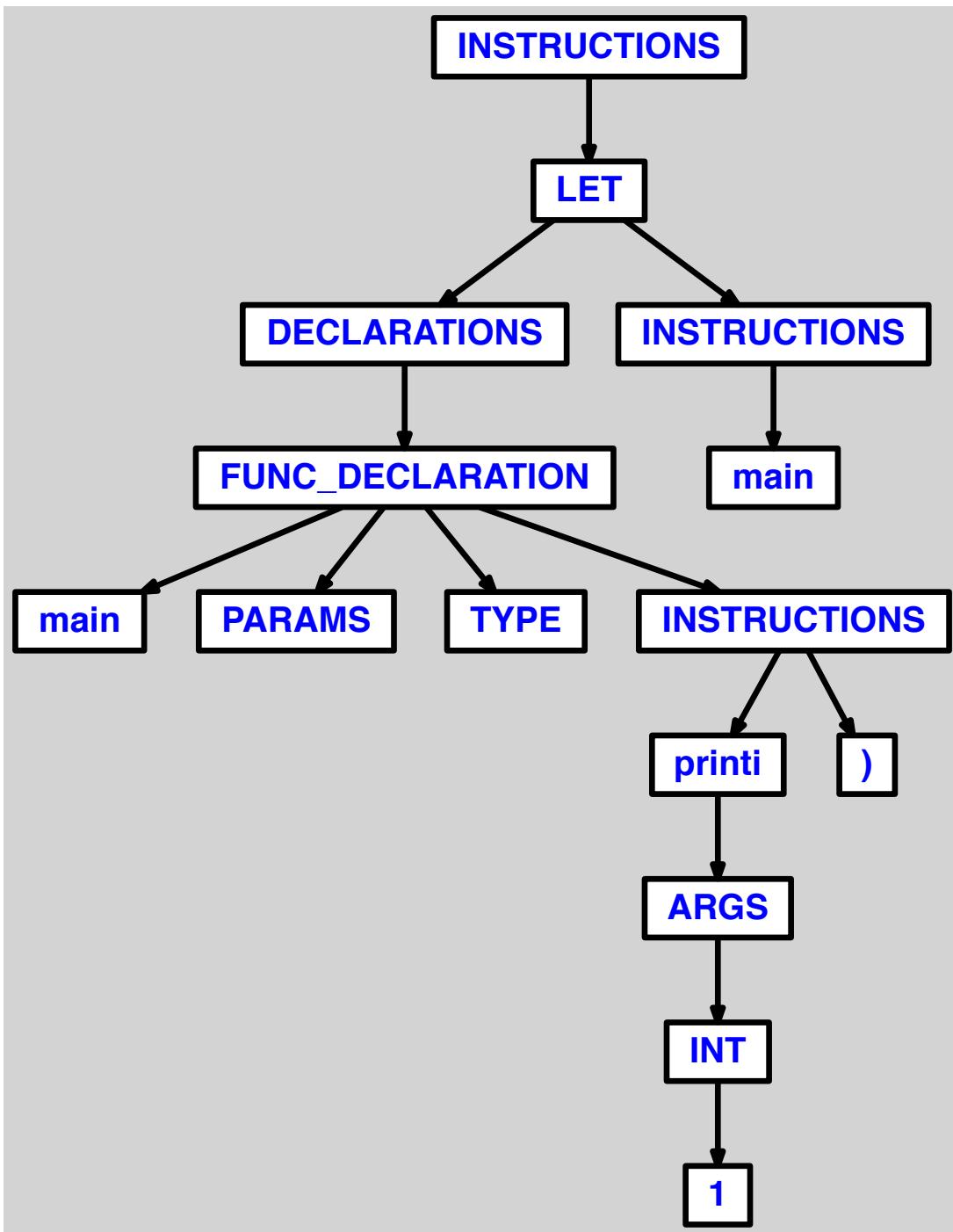
### 3.1.12 parenthesage sans contenu

```
let function main() = () in main() end
```



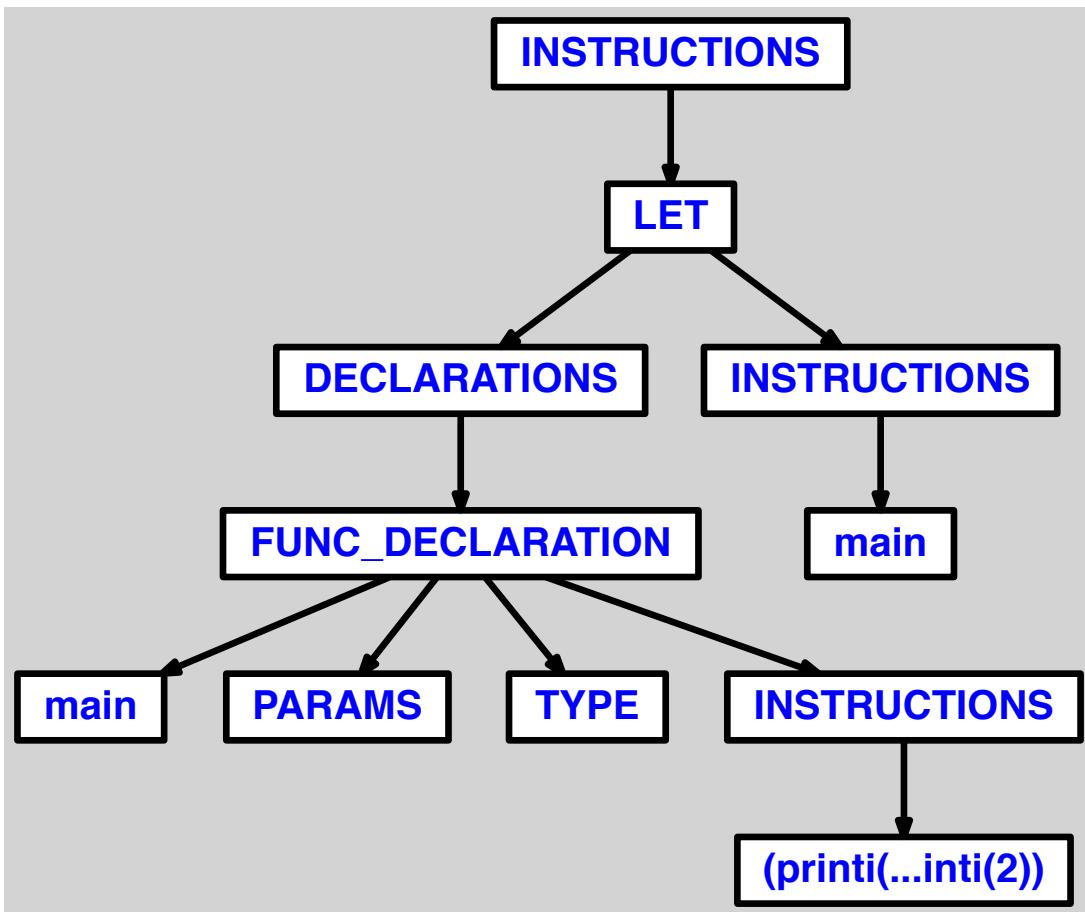
### 3.1.13 expression parenthessee avec ; en trop

```
let function main() = (printi(1);) in main() end
```



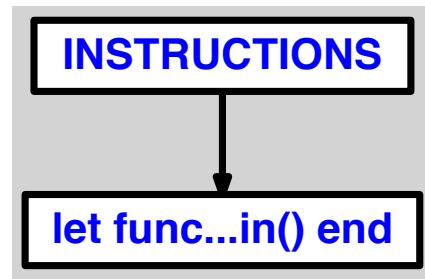
### 3.1.14 expression parenthesée avec oubli de ;

```
let function main() = (printi(1) printi(2)) in main() end
```



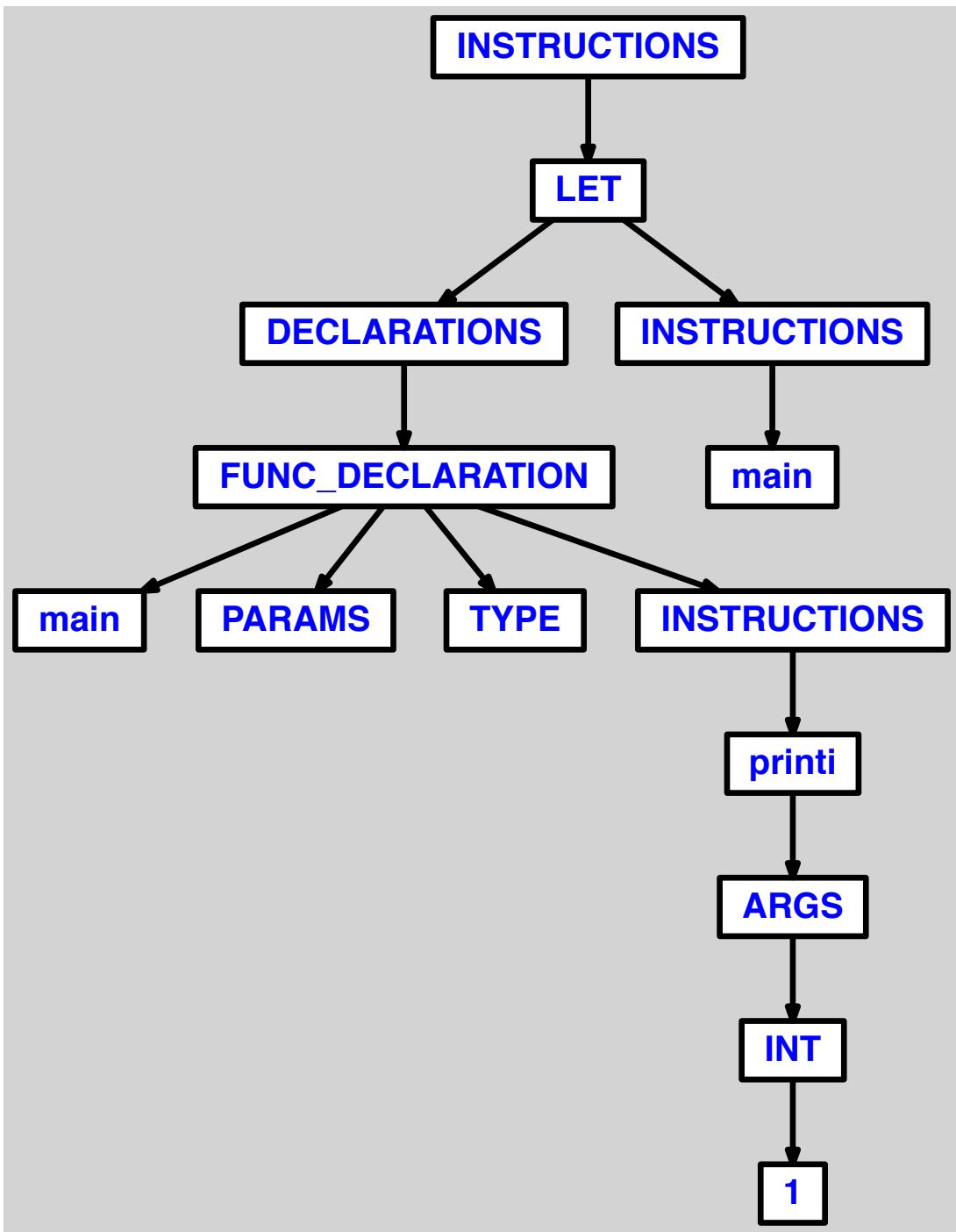
### 3.1.15 expression non parenthesee avec oubli de ;

```
let function main() = printi(1) printi(2) in main() end
```



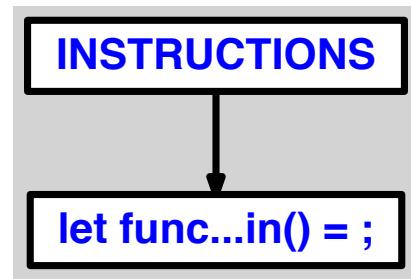
### 3.1.16 expression non parenthesee avec ; en trop

```
let function main() = printi(1); in main() end
```



### 3.1.17 ; sans instruction avant

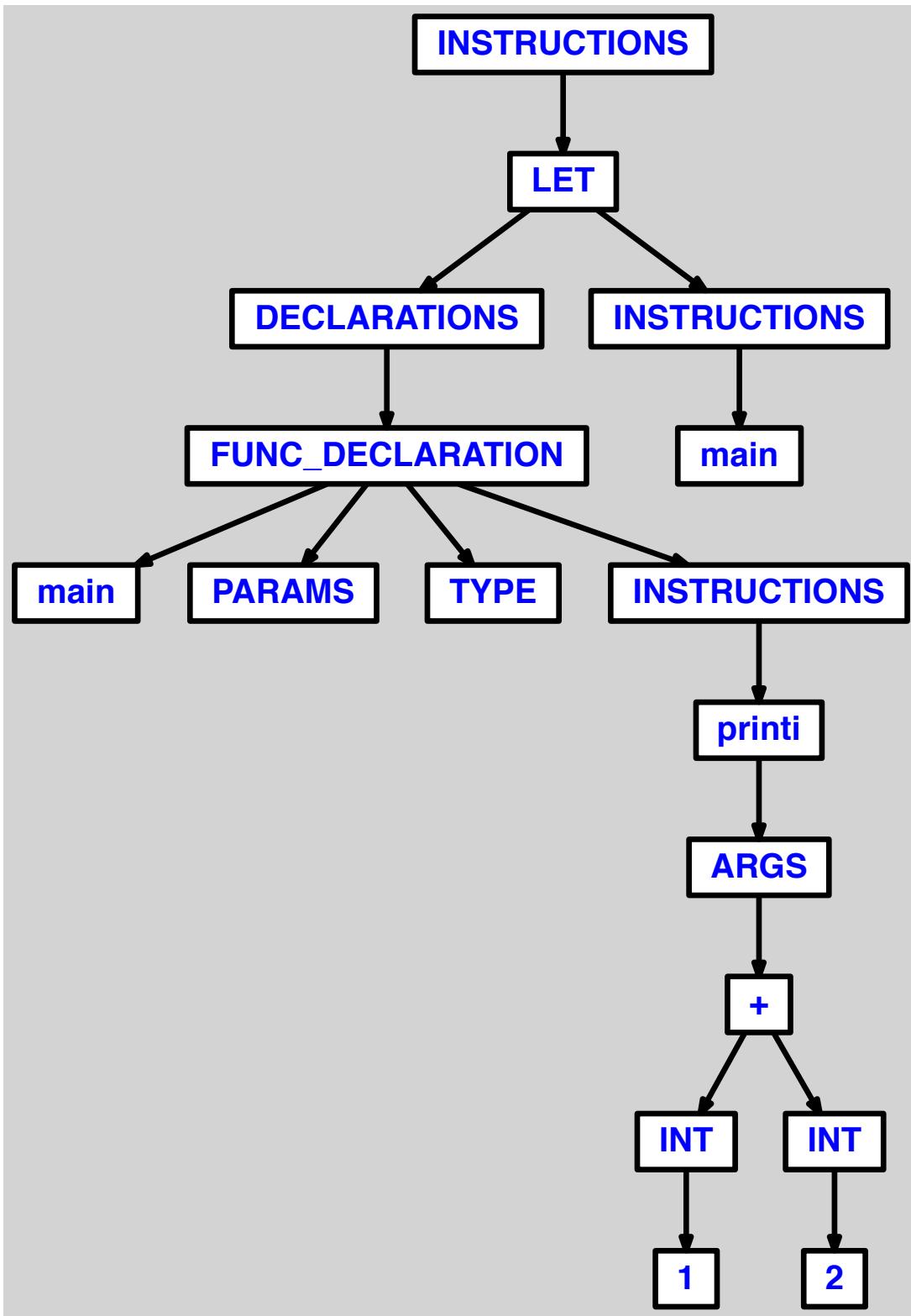
```
let function main() = ;
```



## 3.2 OK

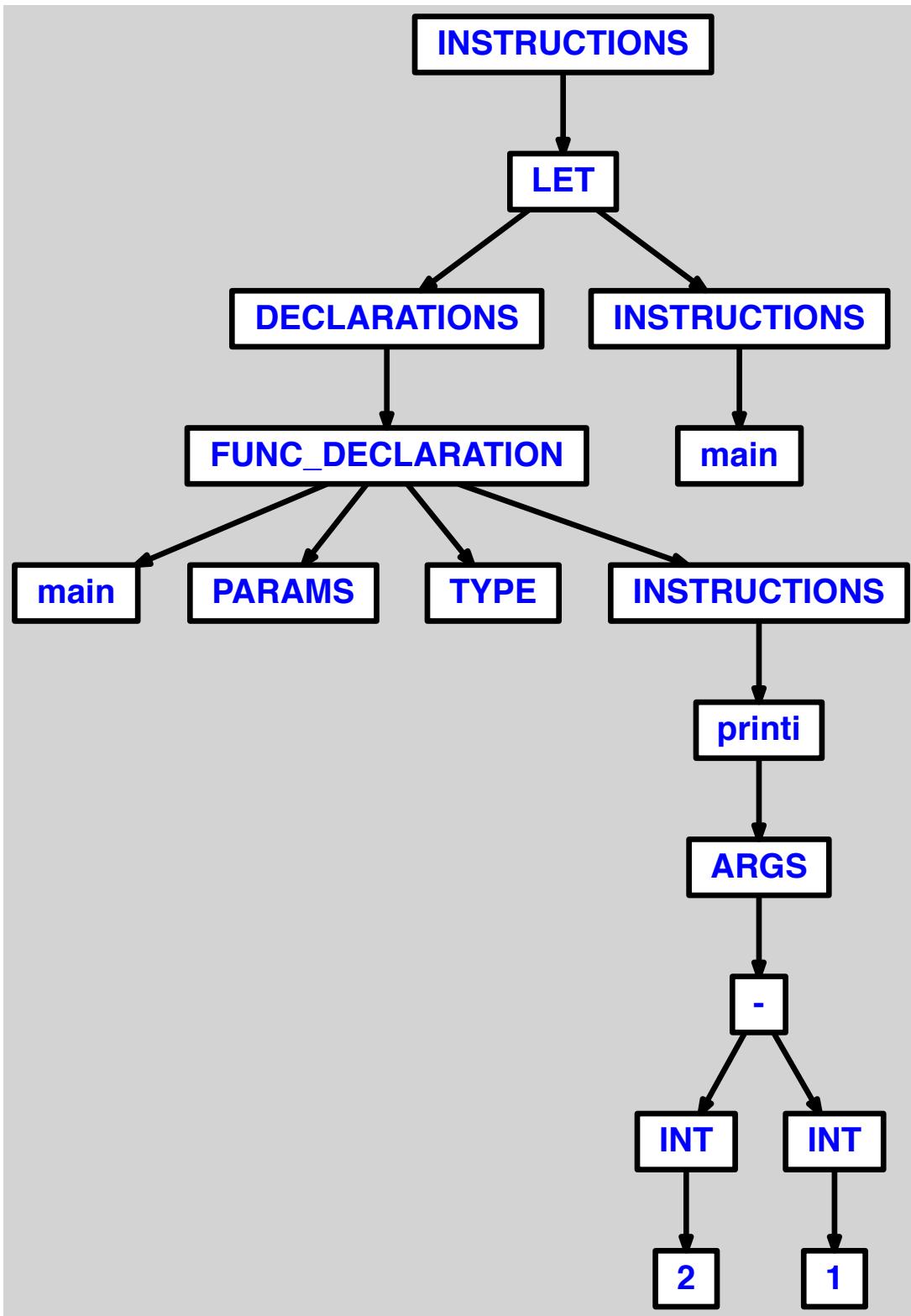
### 3.2.1 addition simple, a 2 termes

```
let
    function main() = printi(1+2)
in main() end
```



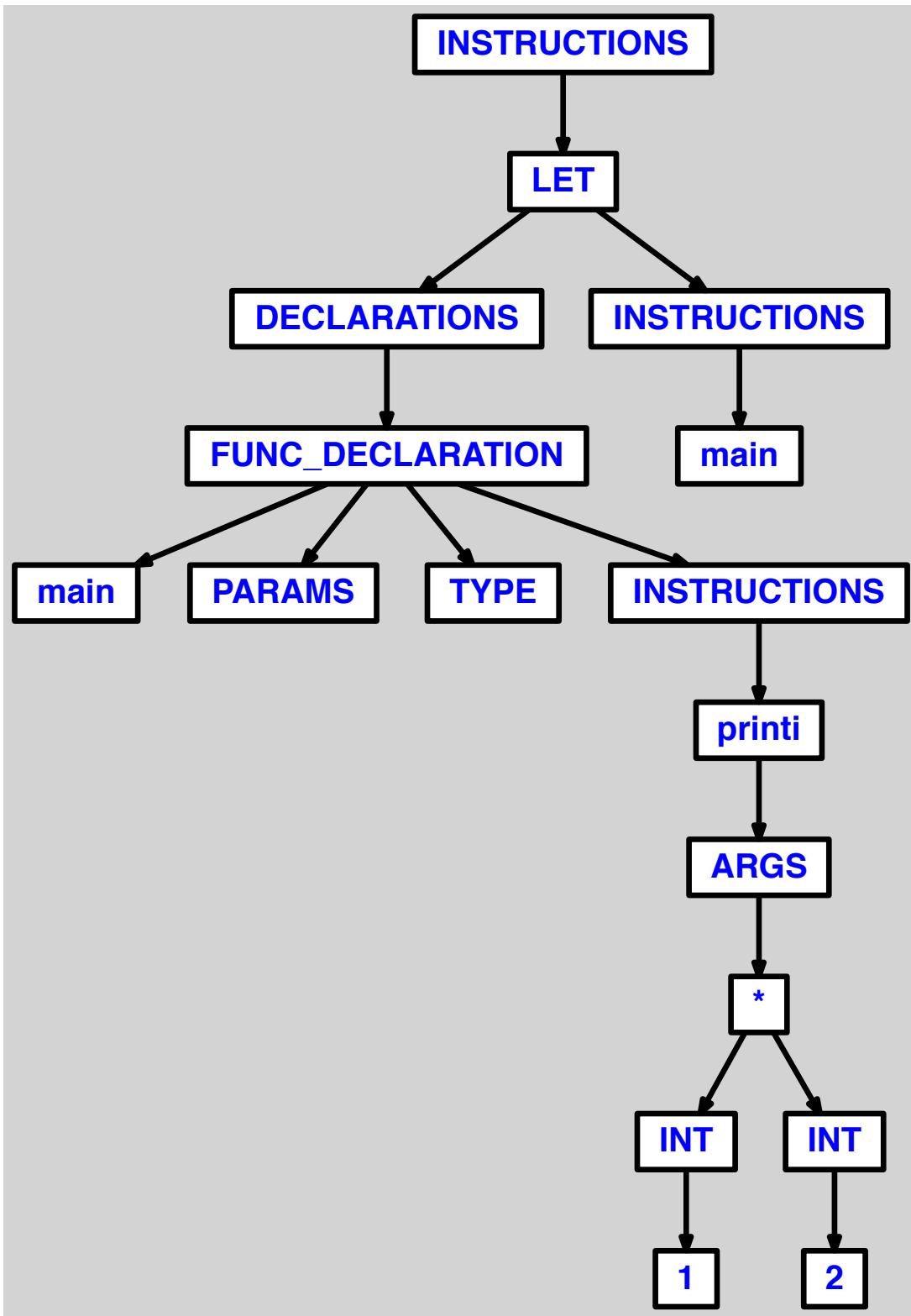
### 3.2.2 soustraction simple, à 2 termes

```
let
    function main() = printi(2-1)
in main() end
```



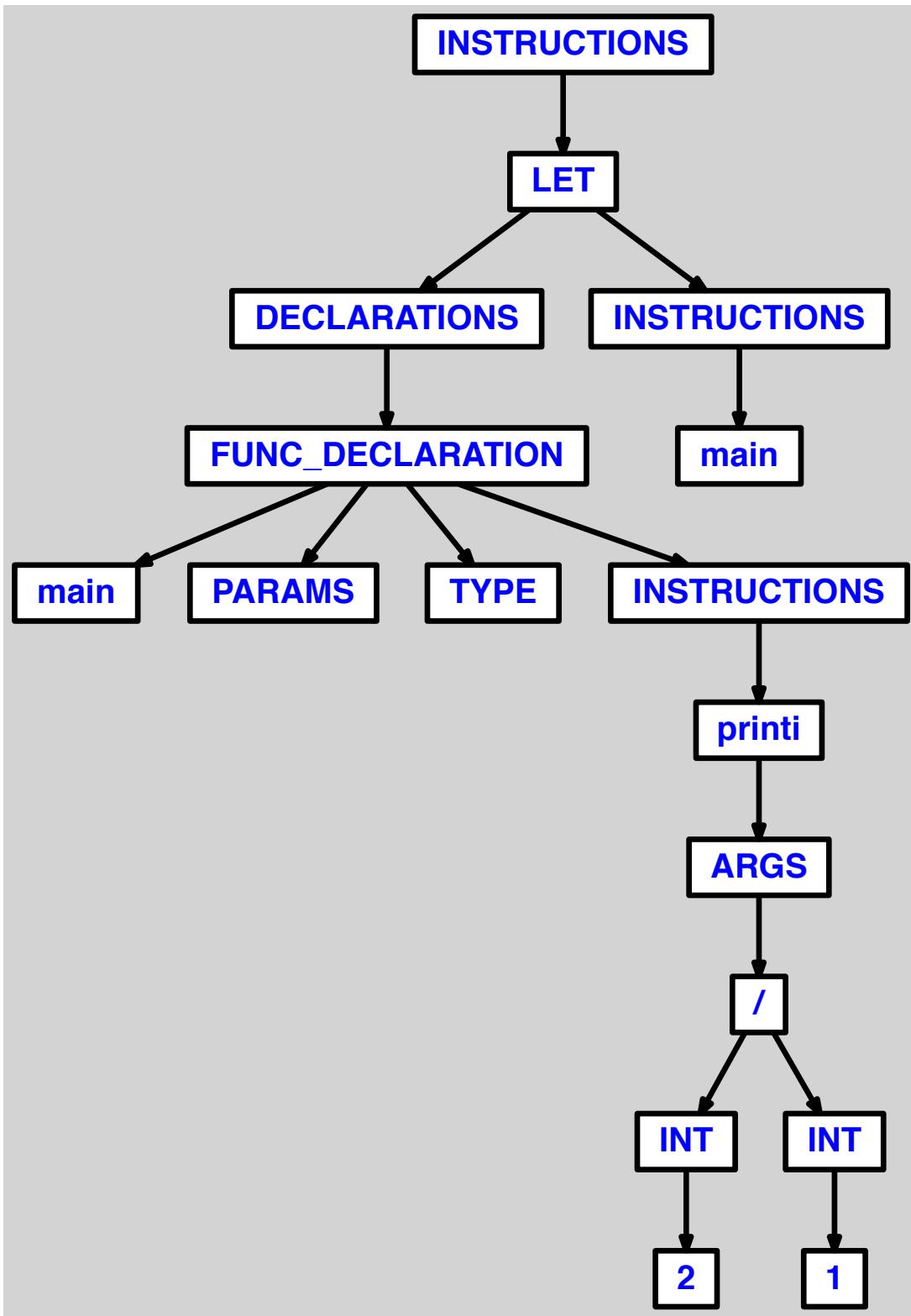
### 3.2.3 multiplication simple, a 2 termes

```
let
    function main() = printi(1*2)
in main() end
```



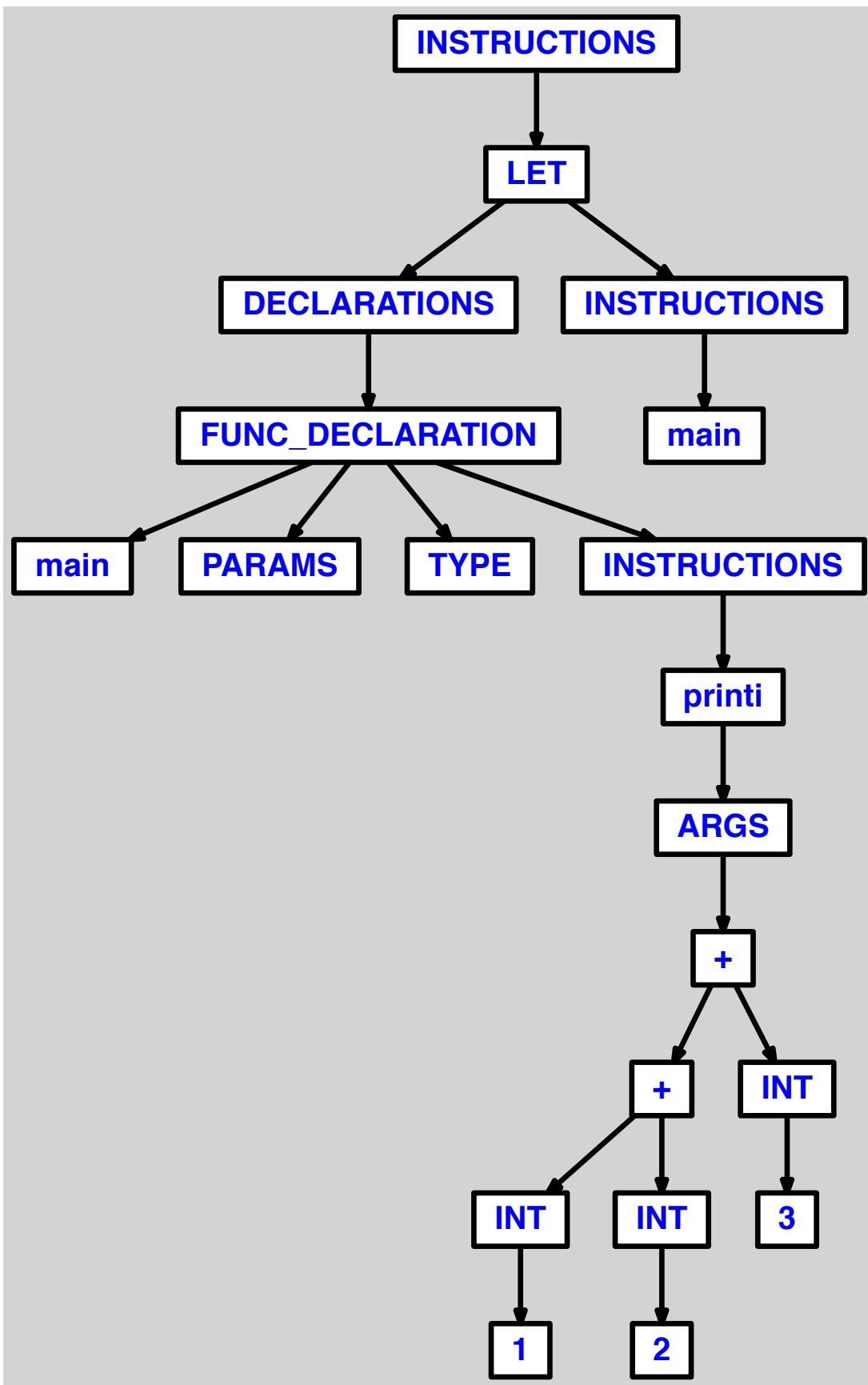
### 3.2.4 division simple, a 2 termes

```
let
    function main() = printi(2/1)
in main() end
```



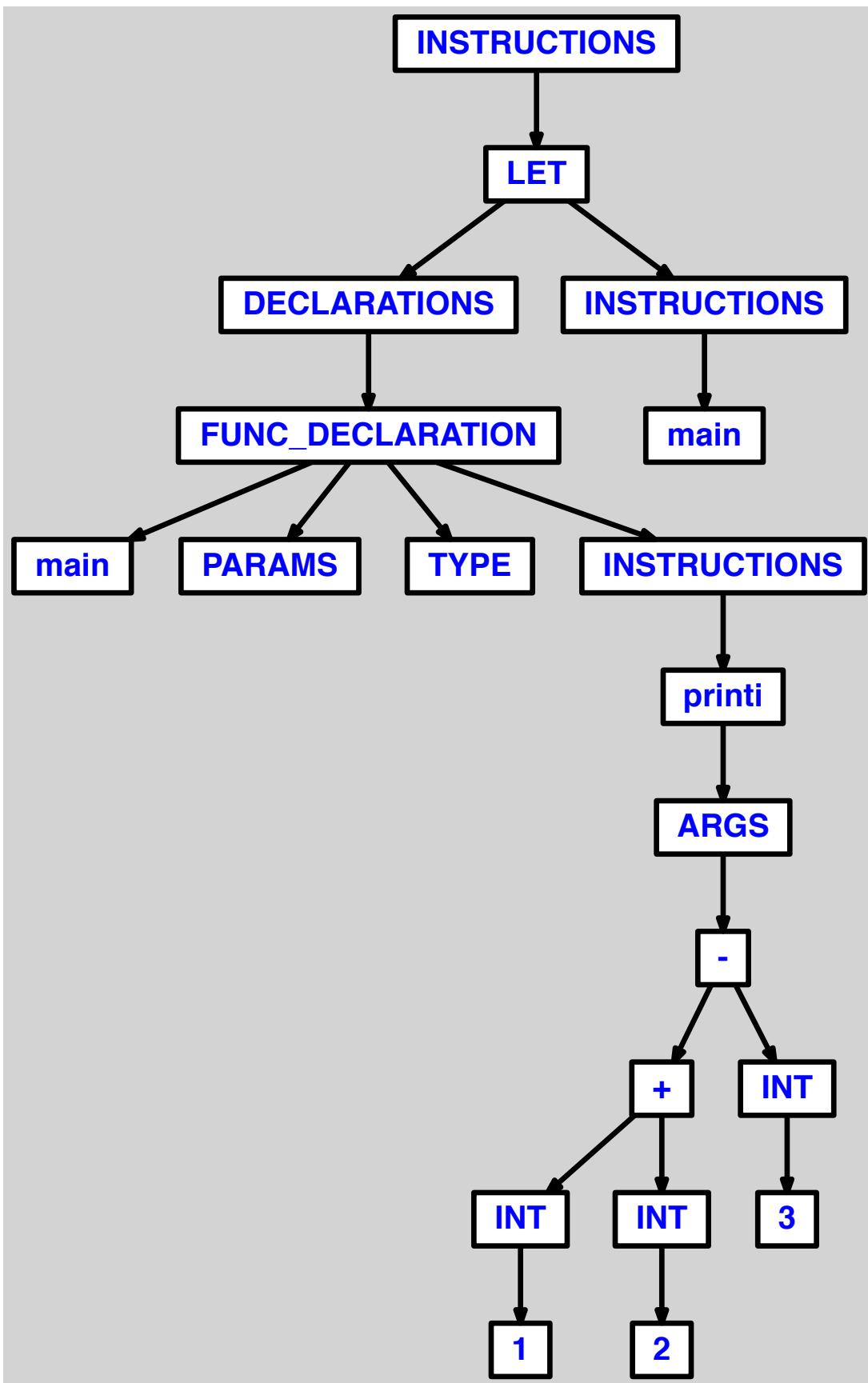
### 3.2.5 addition à 3 termes

```
let
    function main() = printi(1+2+3)
in main() end
```



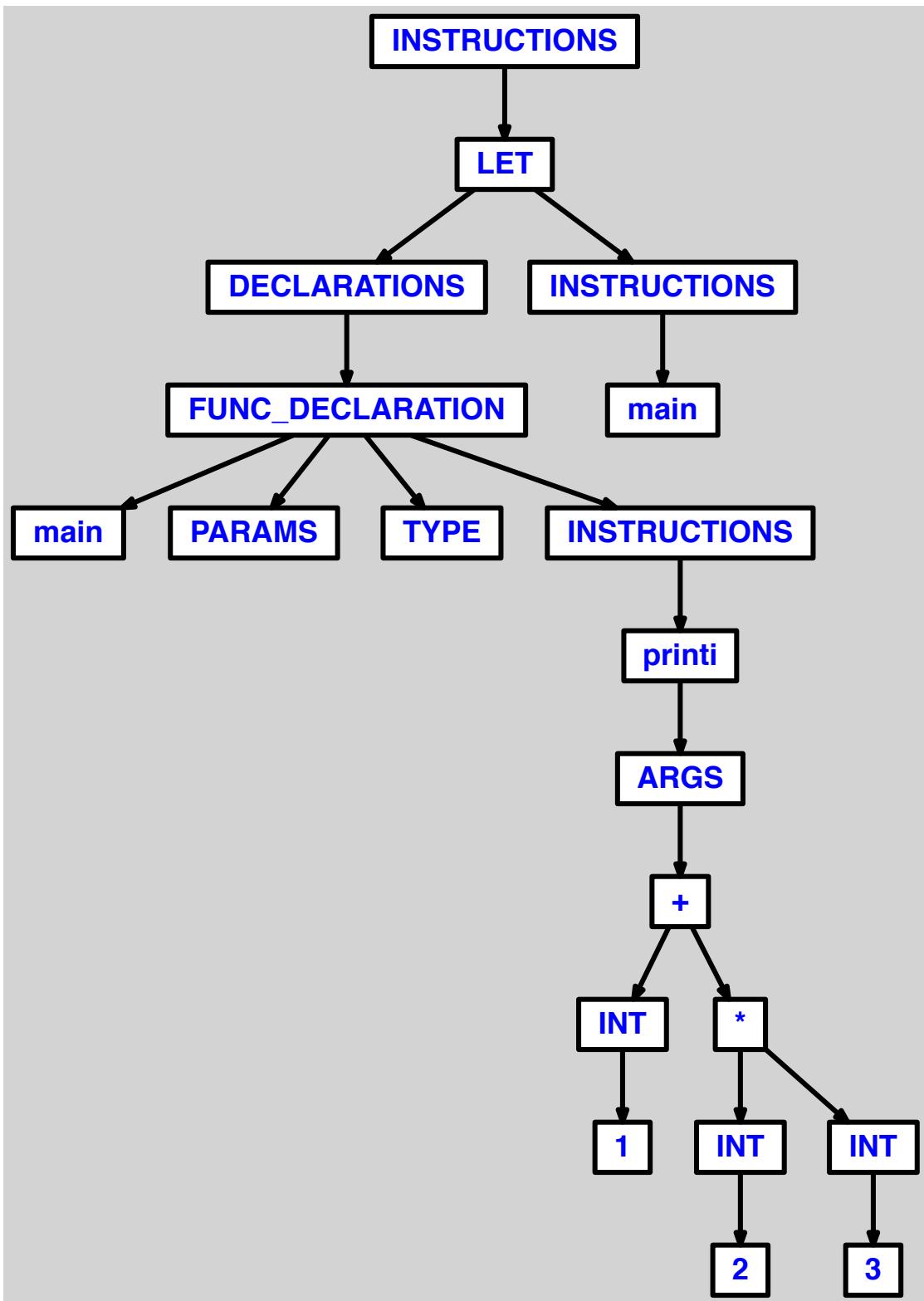
### 3.2.6 addition suivie de soustraction

```
let
    function main() = printi(1+2-3)
in main() end
```



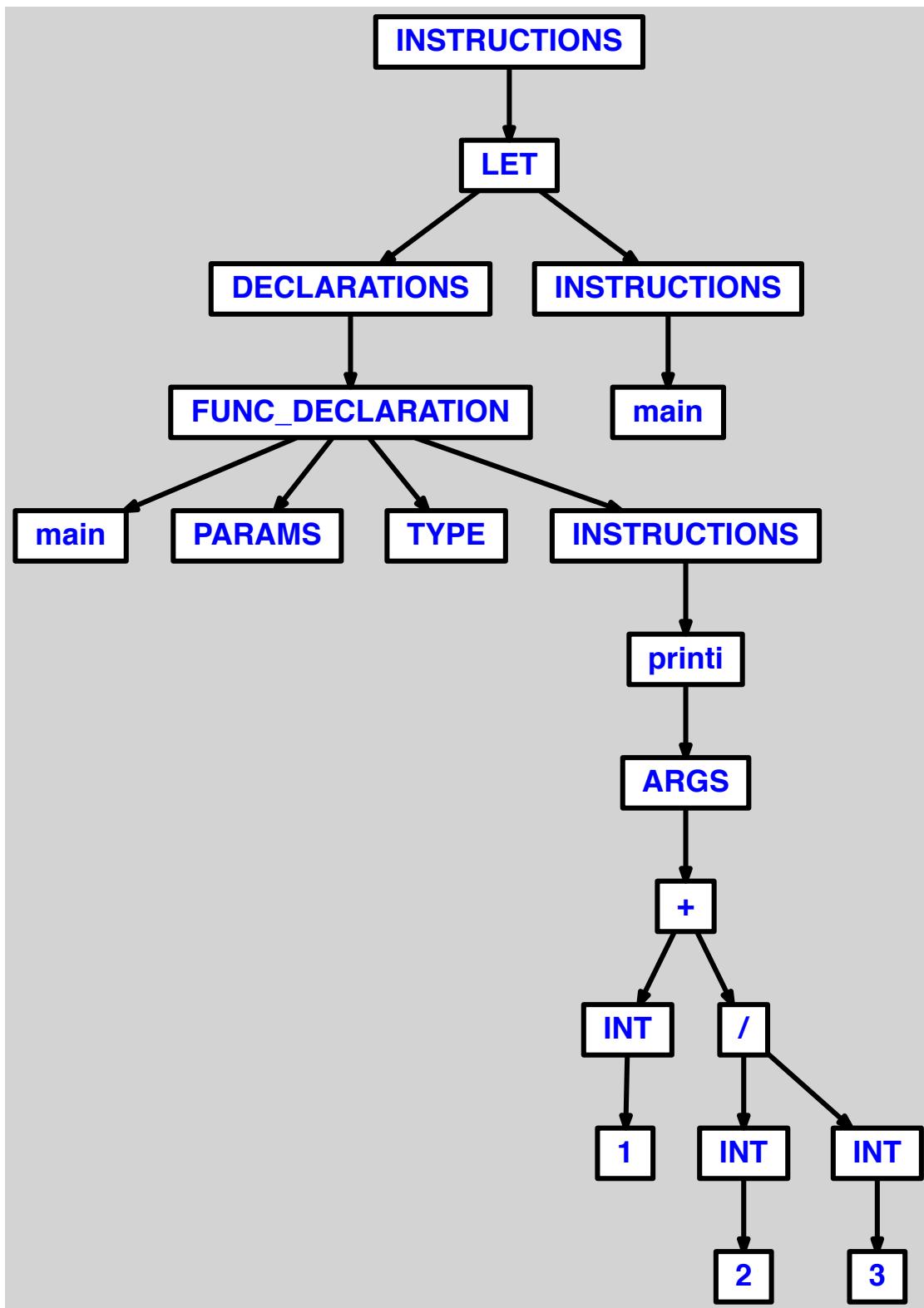
### 3.2.7 addition suivie de multiplication

```
let
    function main() = printi(1+2*3)
in main() end
```



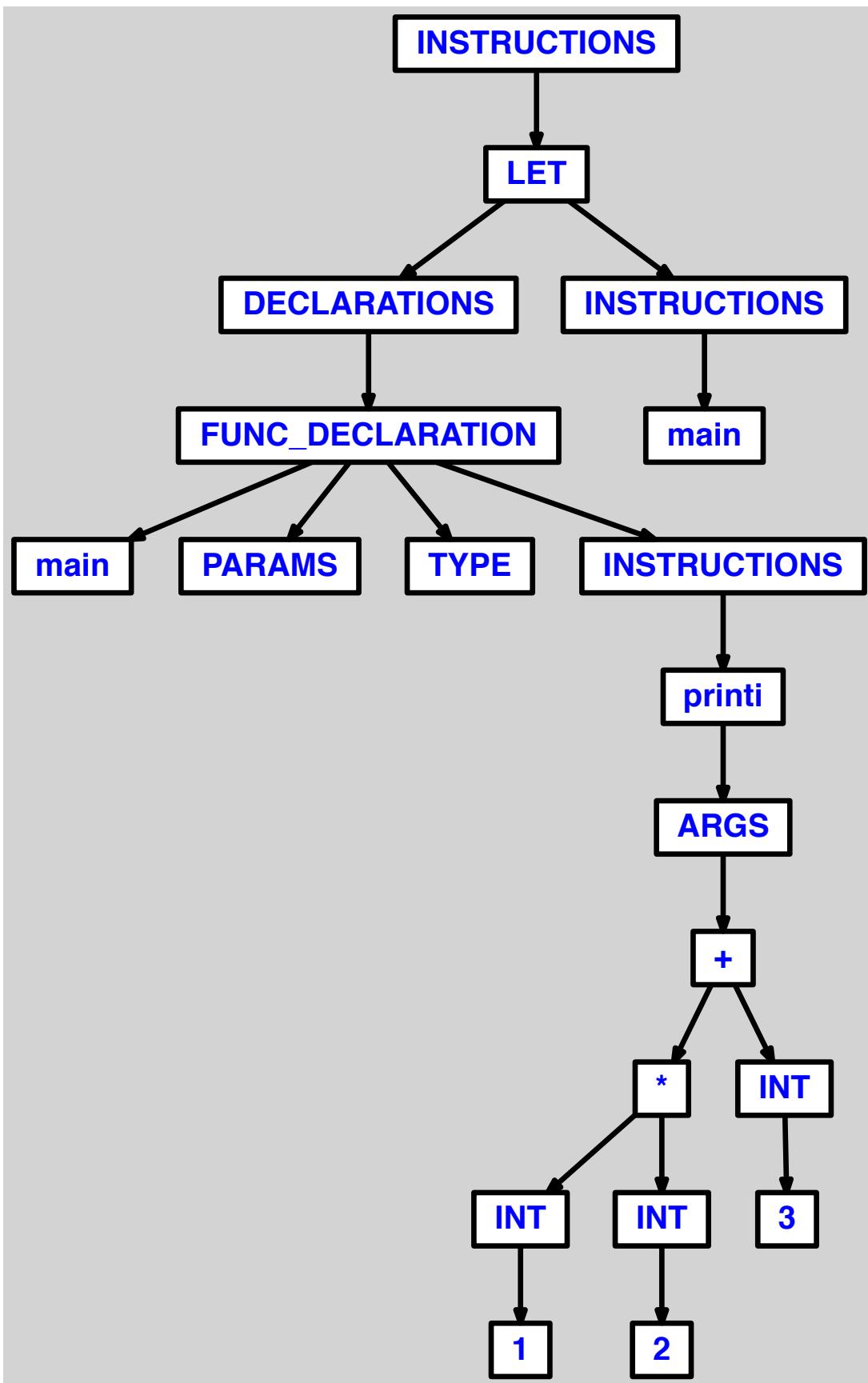
### 3.2.8 addition suivie de division

```
let
    function main() = printi(1+2/3)
in main() end
```



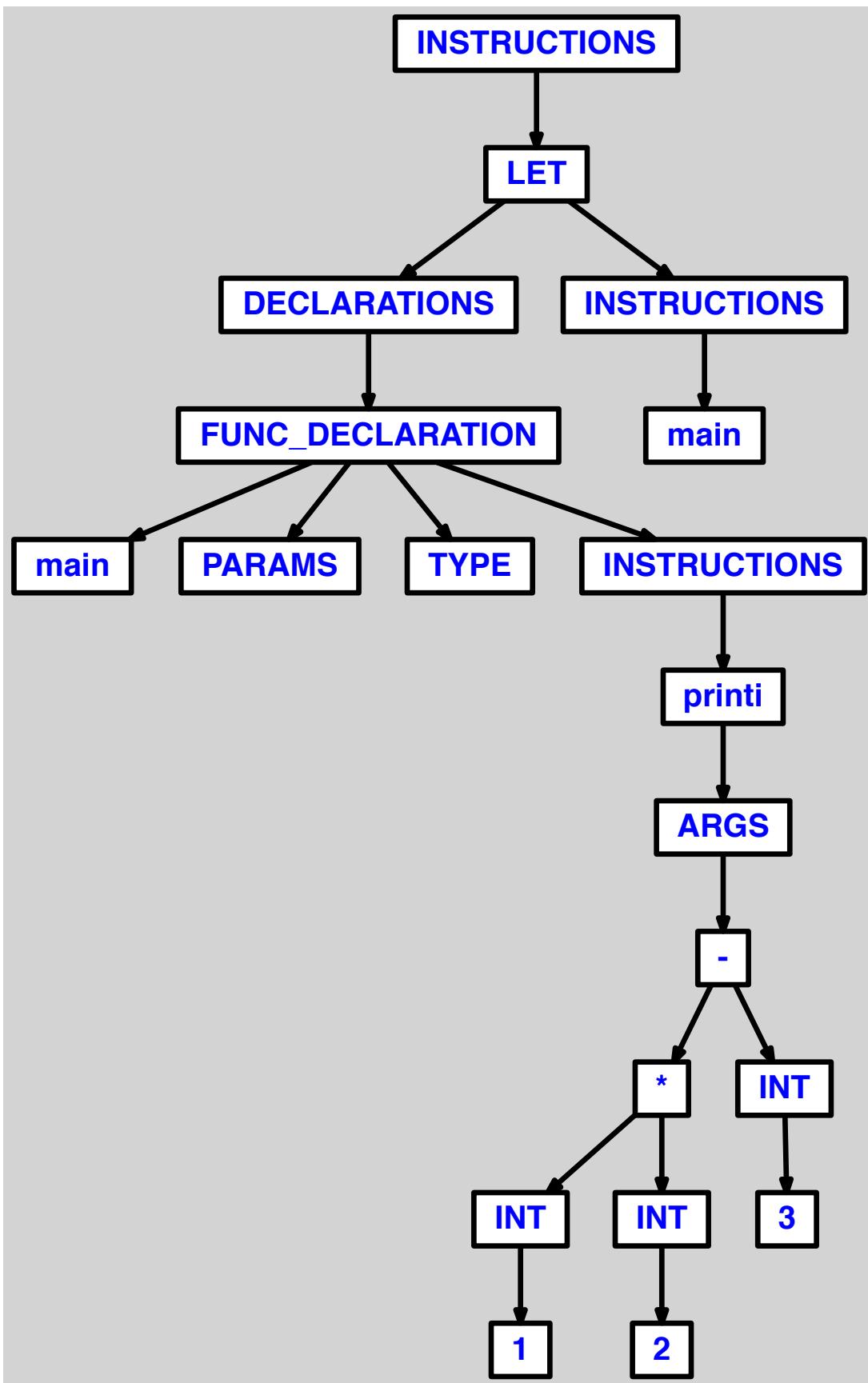
### 3.2.9 multiplication suivie d'addition

```
let
    function main() = printi(1*2+3)
in main() end
```



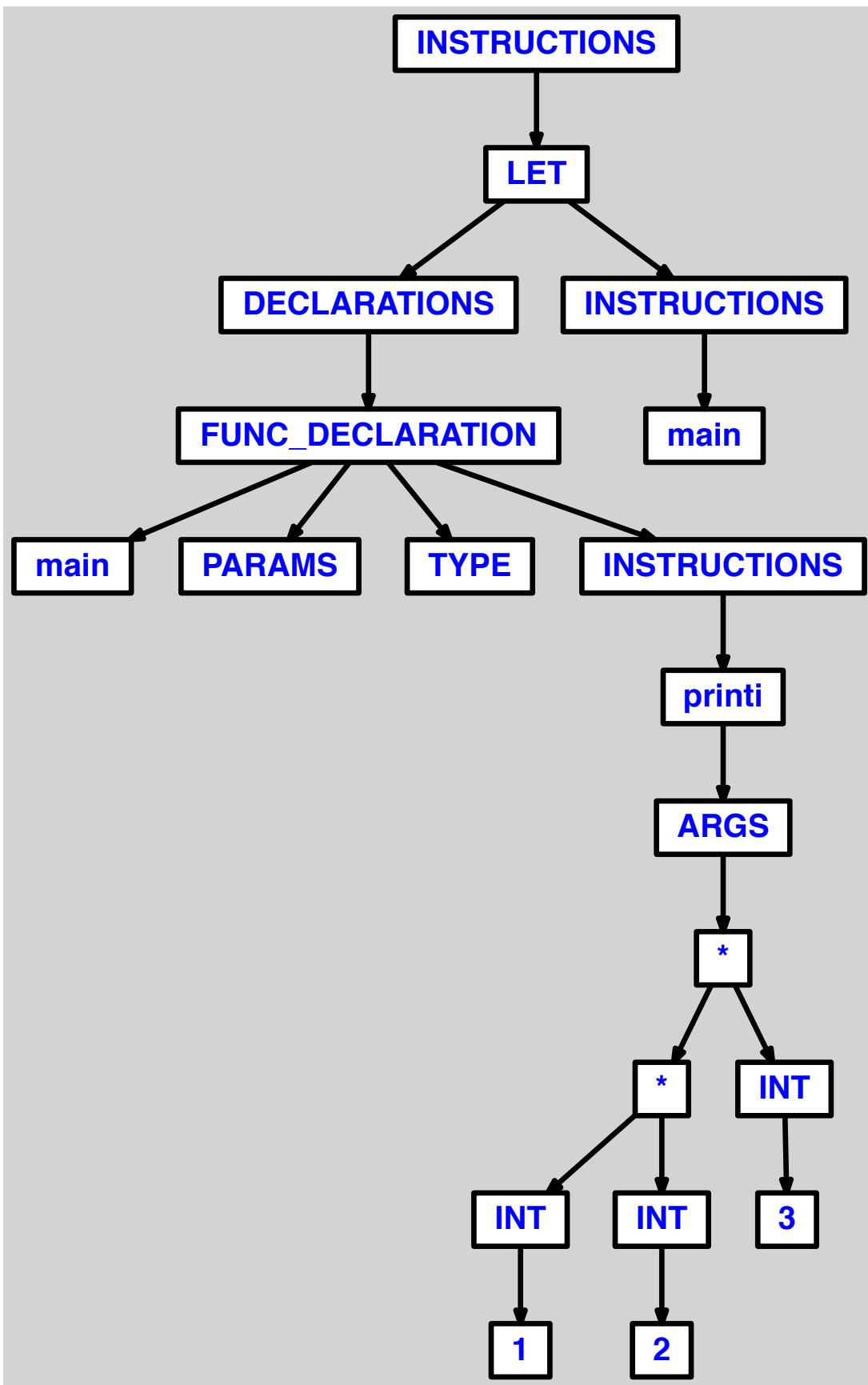
### 3.2.10 multiplication suivie de soustraction

```
let
    function main() = printi(1*2-3)
in main() end
```



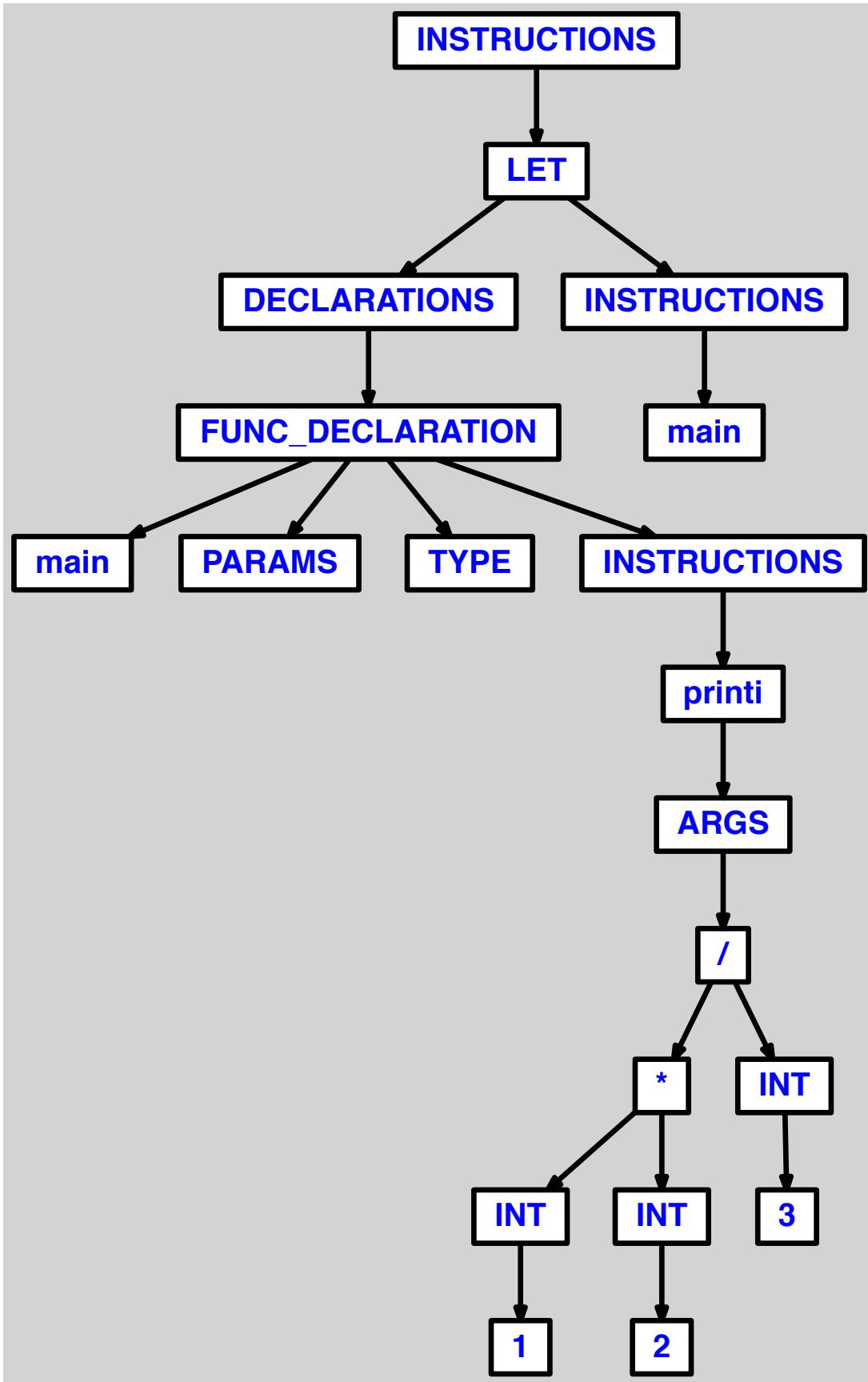
### 3.2.11 multiplication a 3 termes

```
let
    function main() = printi(1*2*3)
in main() end
```



### 3.2.12 multiplication suivie de division

```
let
    function main() = printi(1*2/3)
in main() end
```

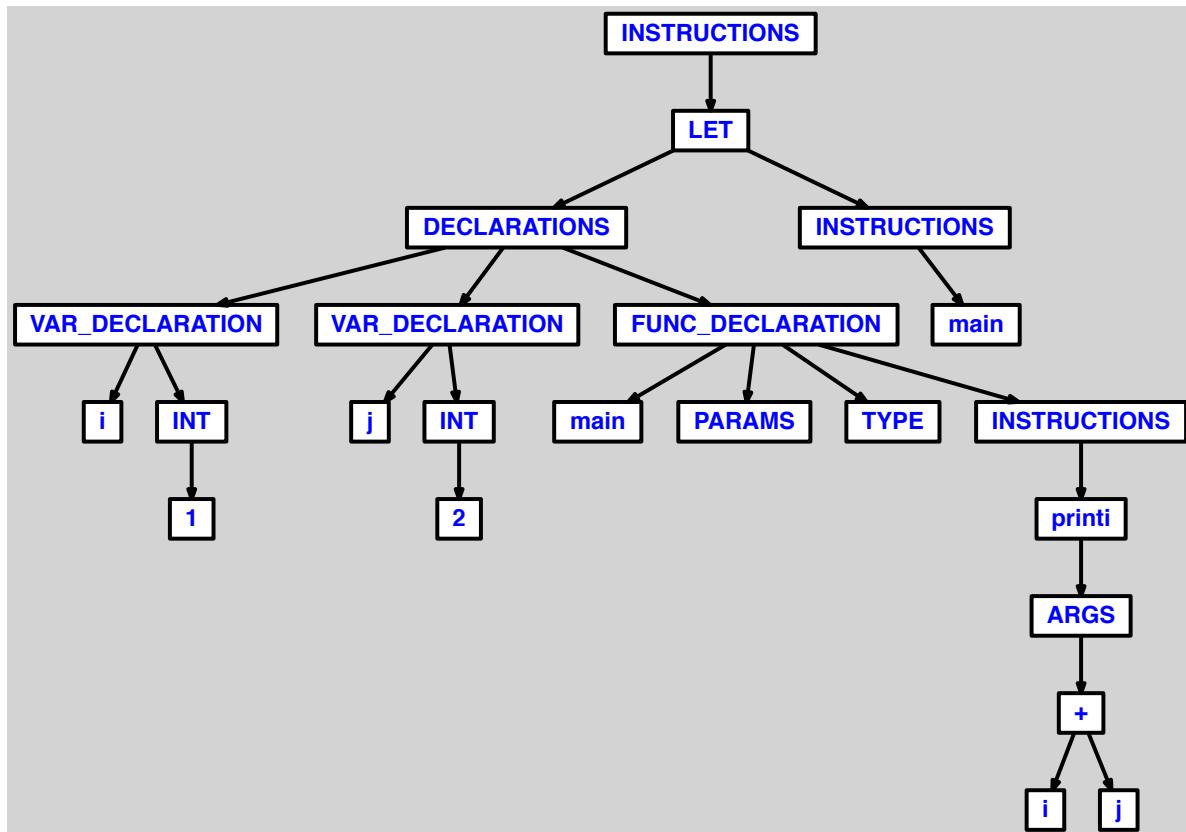


### 3.2.13 addition simple, à 2 termes, identifiés par des variables

let

```
var i := 1
var j := 2
```

```
function main() = printi(i+j)
in main() end
```

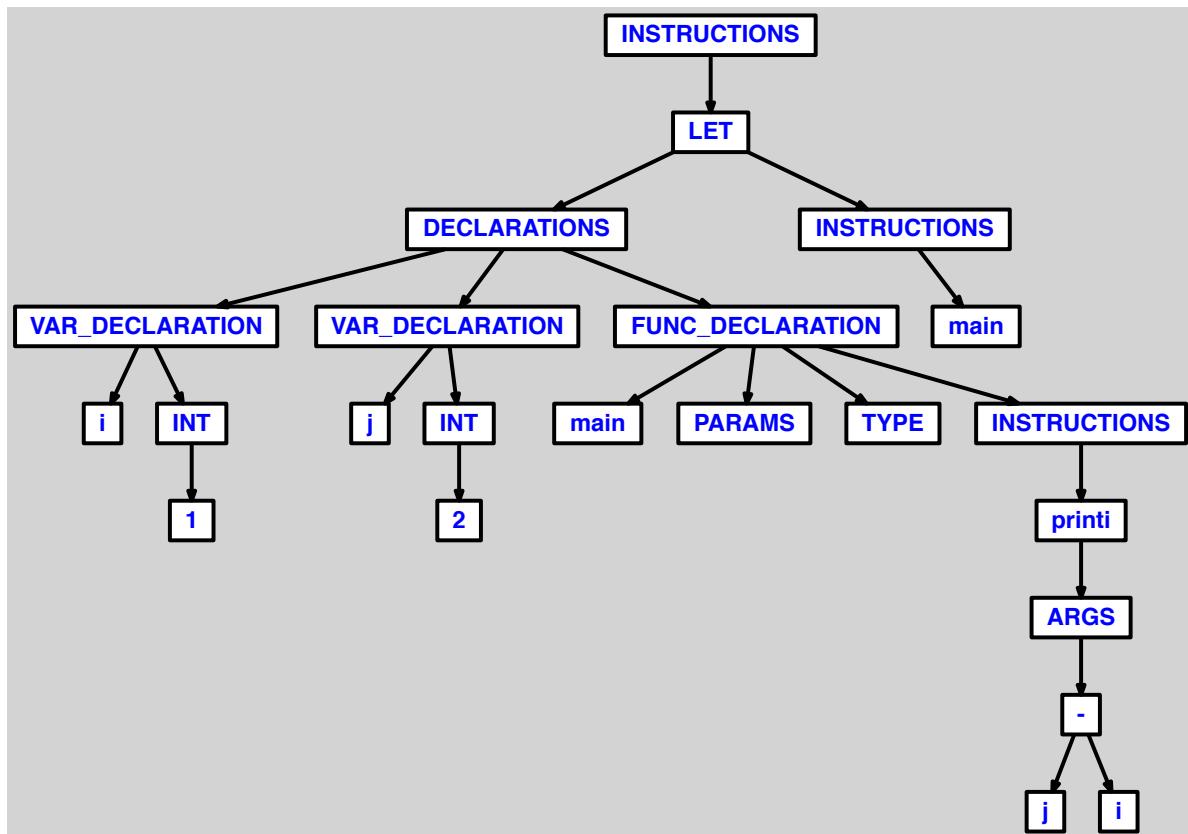


### 3.2.14 soustraction simple, à 2 termes, identifiés par des variables

let

```
var i := 1
var j := 2
```

```
function main() = printi(j-i)
in main() end
```

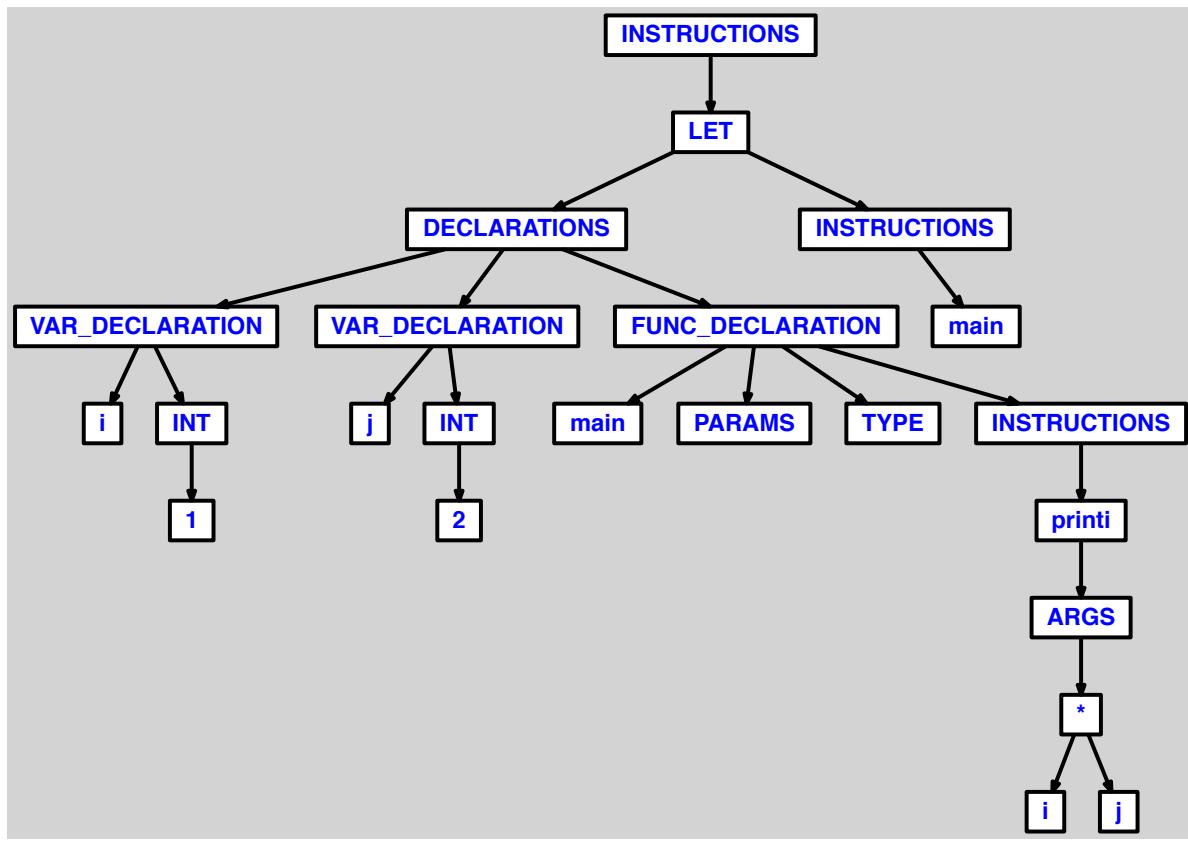


### 3.2.15 multiplication simple, à 2 termes, identifiés par des variables

```

let
  var i := 1
  var j := 2

  function main() = printi(i*j)
in main() end
  
```

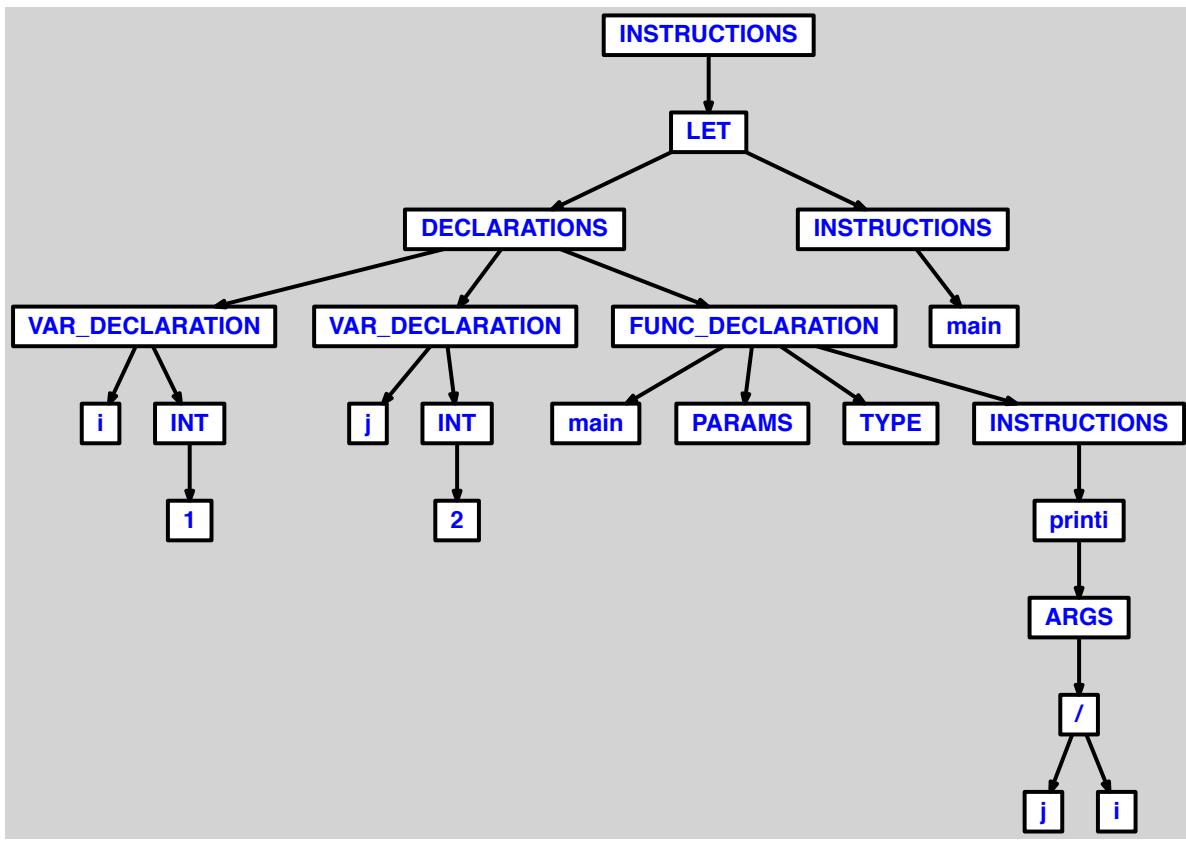


### 3.2.16 division simple, à 2 termes, identifiés par des variables

```

let
  var i := 1
  var j := 2

  function main() = printi(j/i)
in main() end
  
```

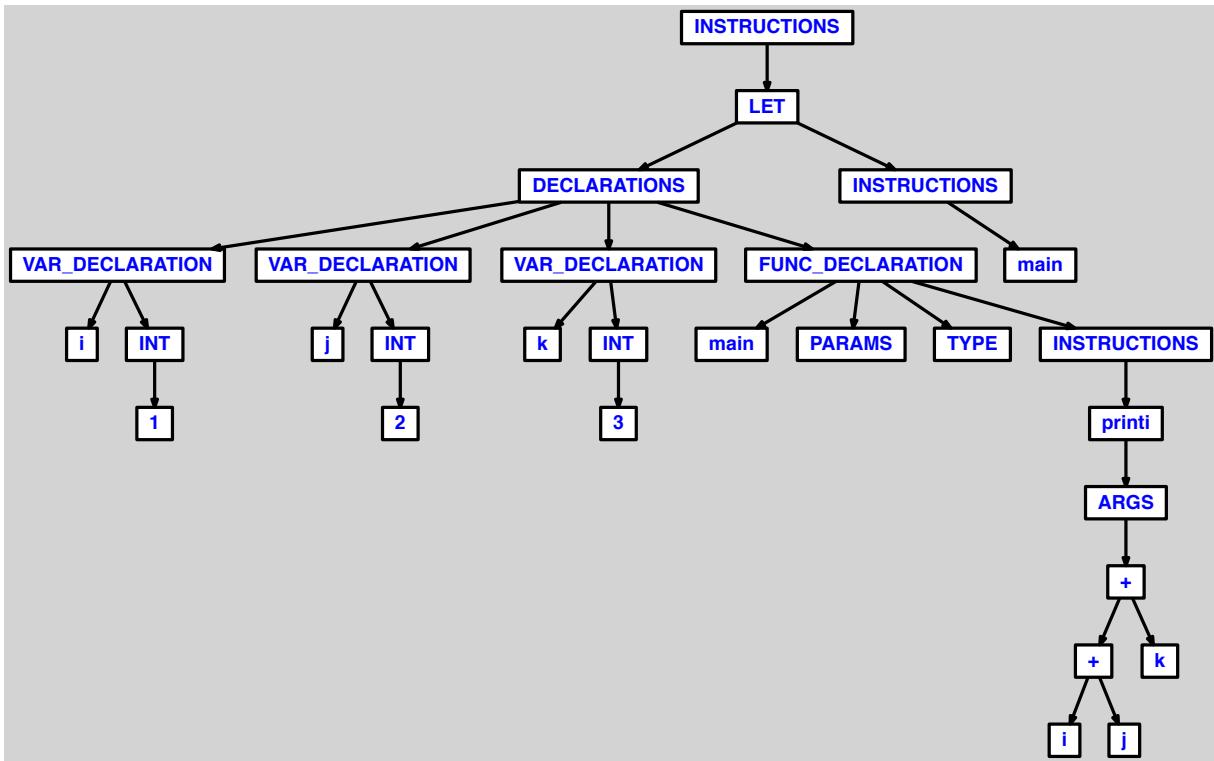


### 3.2.17 addition a 3 termes, identifies par des variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i+j+k)
in main() end
  
```



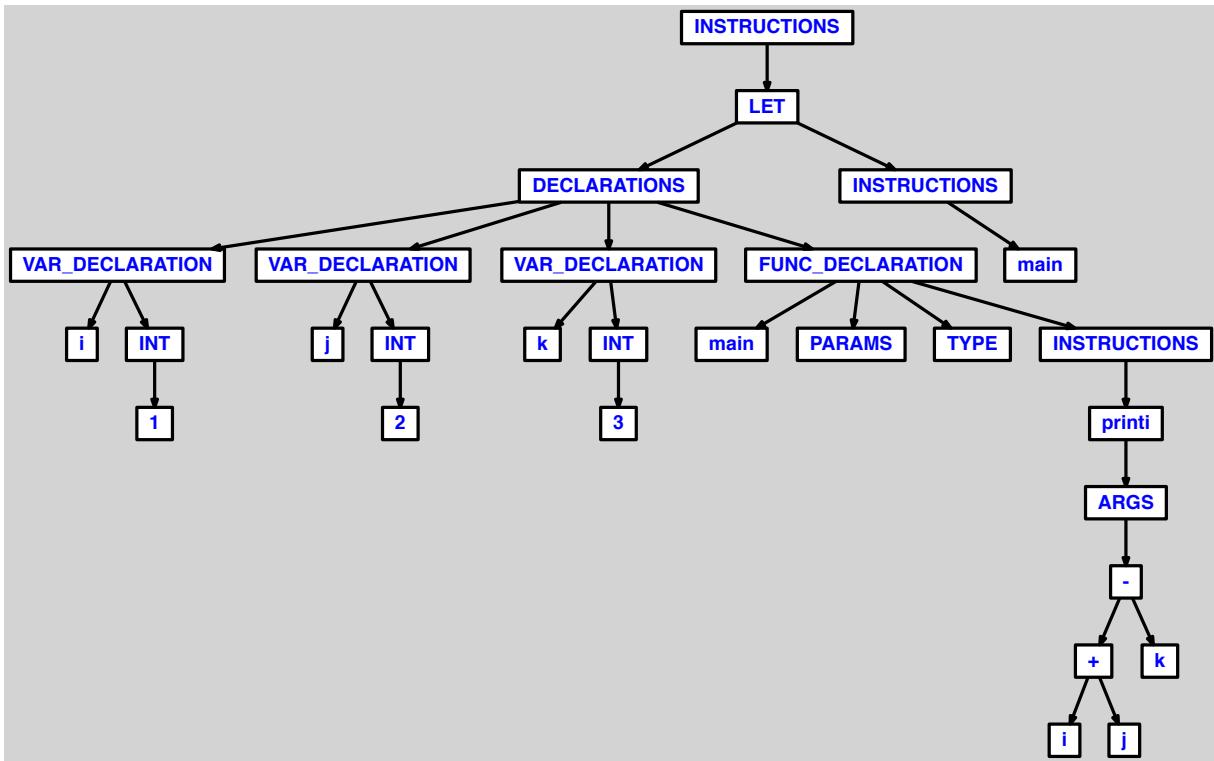
### 3.2.18 addition suivie de soustraction, avec termes identifiés par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i+j-k)
in main() end

```



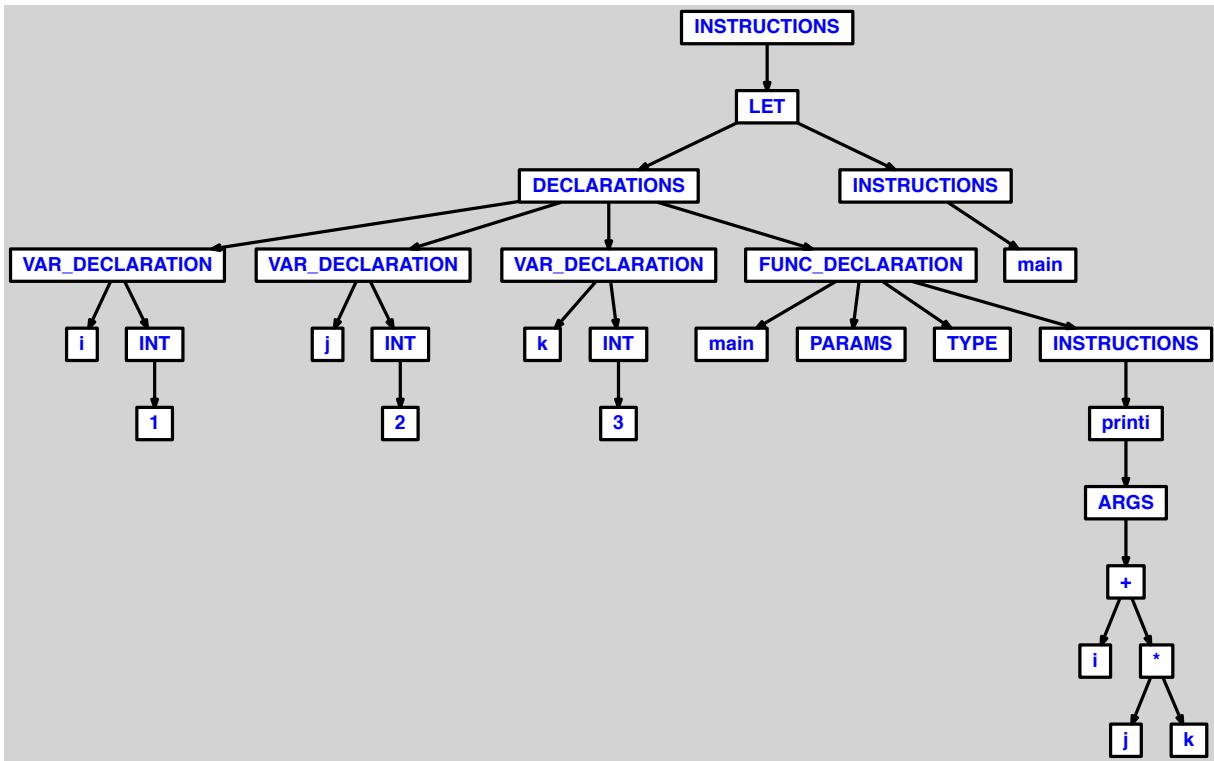
### 3.2.19 addition suivie de multiplication, avec termes identifiés par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i+j*k)
in main() end

```

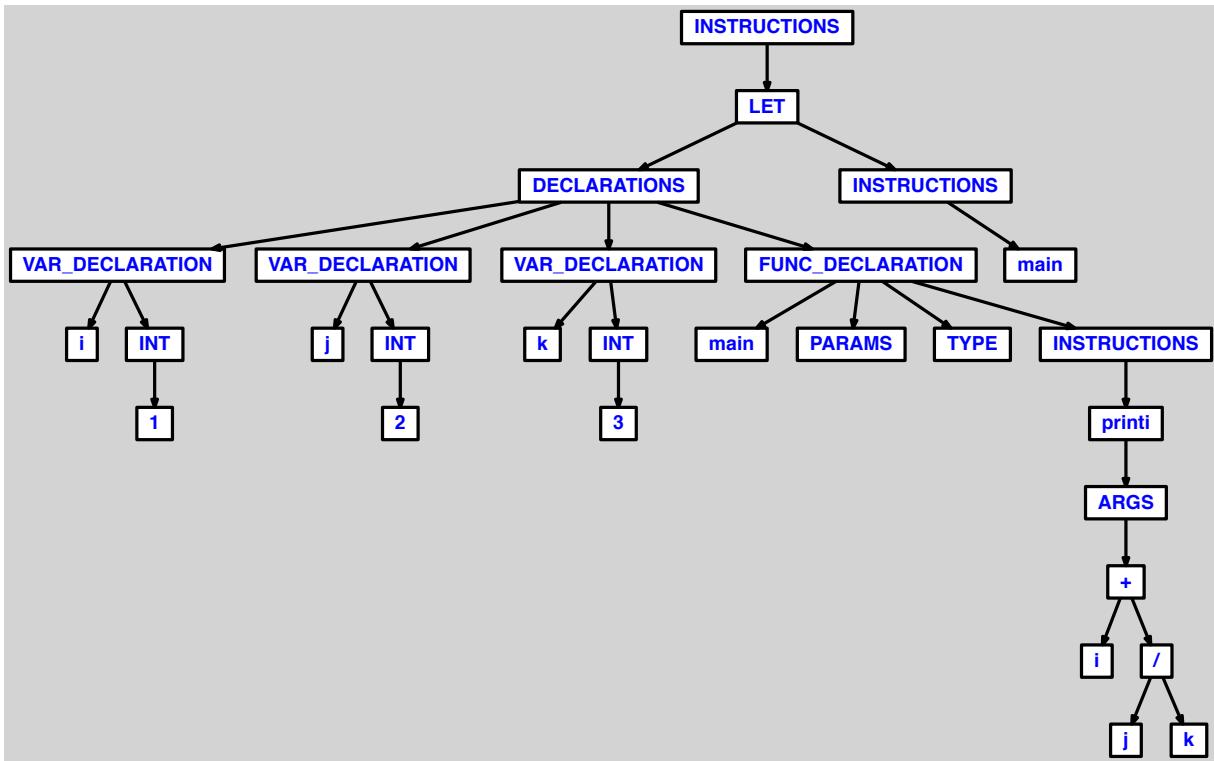


### 3.2.20 addition suivie de division, avec termes identifiés par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i+j/k)
in main() end
  
```

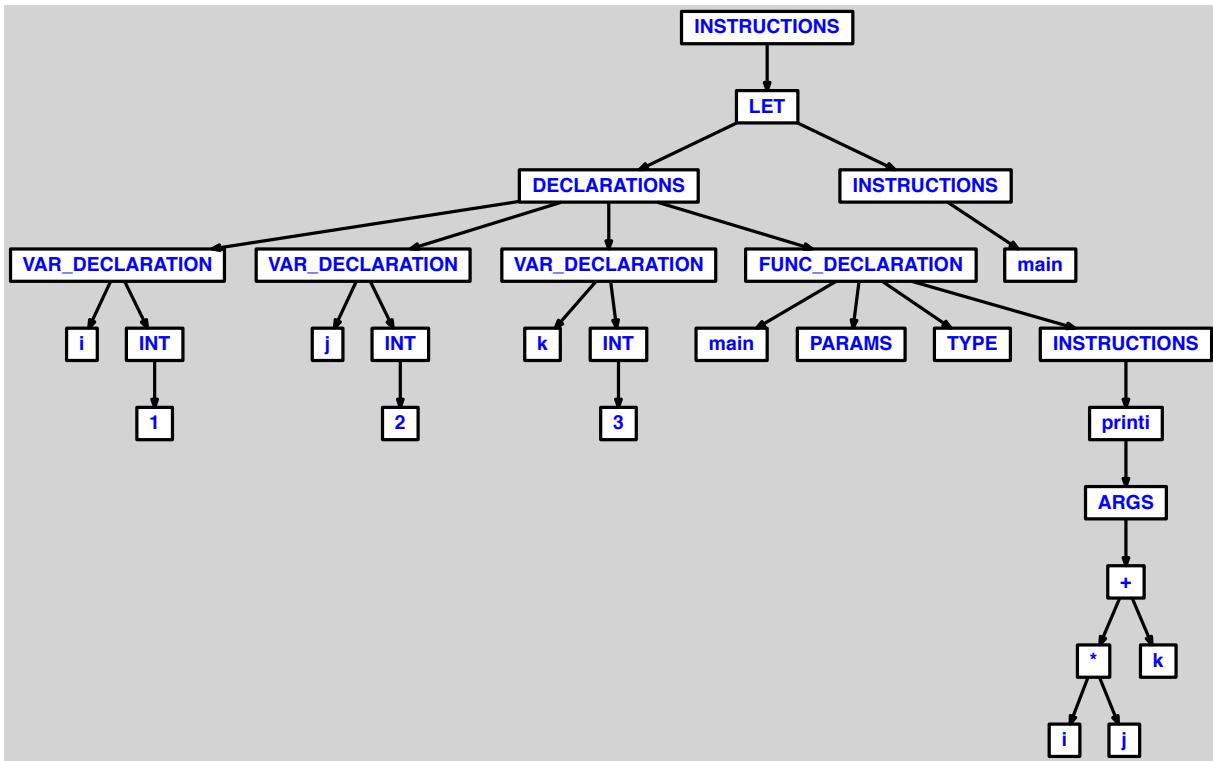


### 3.2.21 multiplication suivie d'addition, avec termes identifiés par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*j+k)
in main() end
  
```



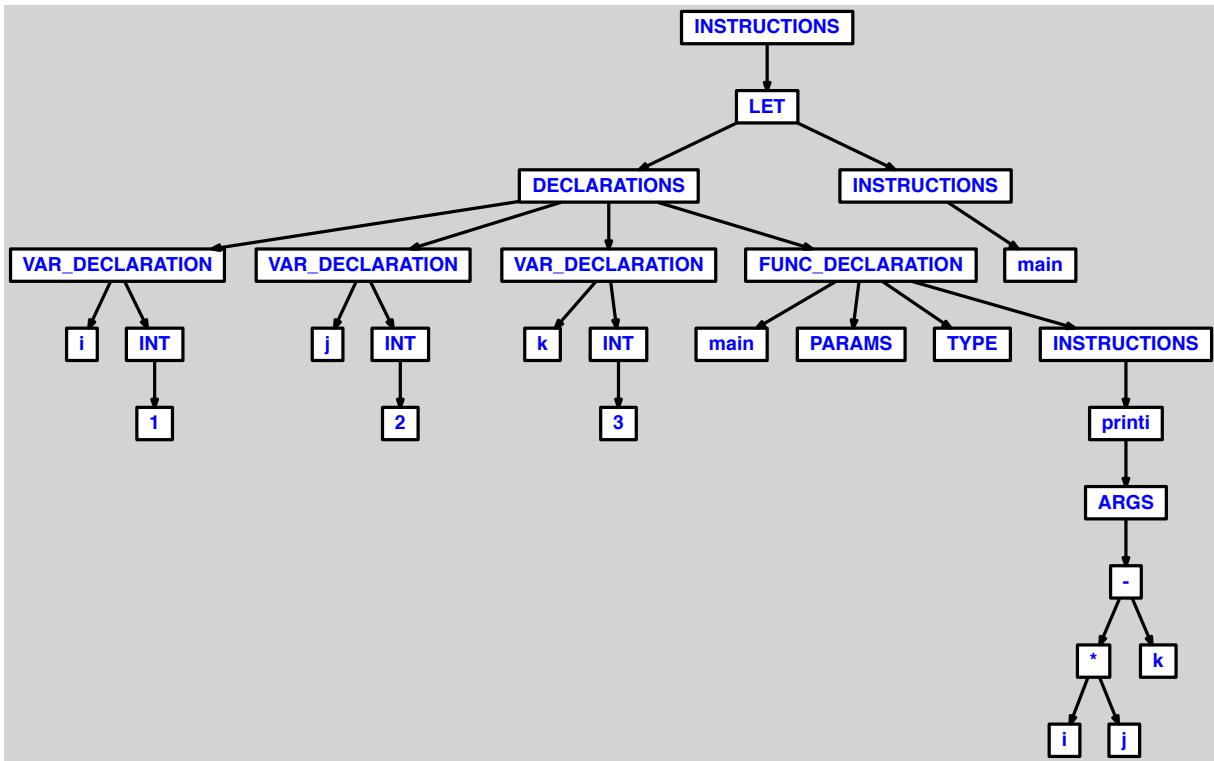
### 3.2.22 multiplication suivie de soustraction, avec termes identifiés par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*j-k)
in main() end

```

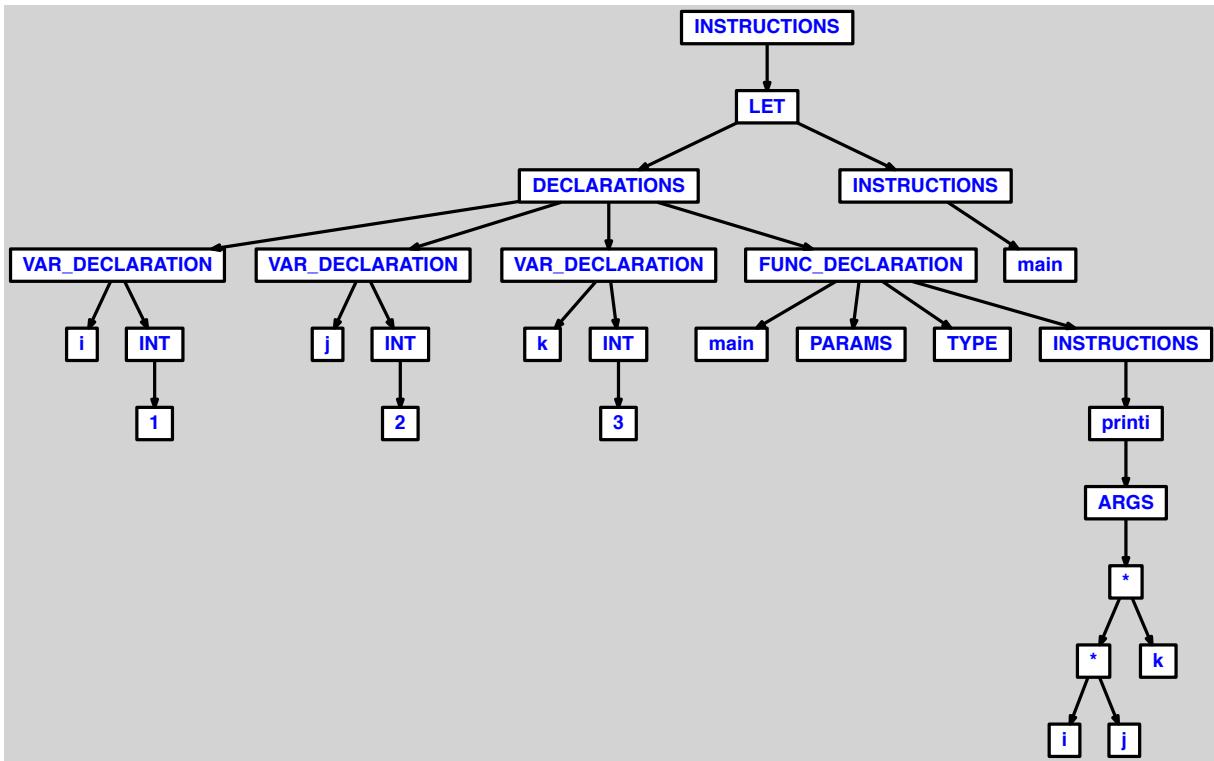


### 3.2.23 multiplication a 3 termes, identifiés par des variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*j*k)
in main() end
  
```



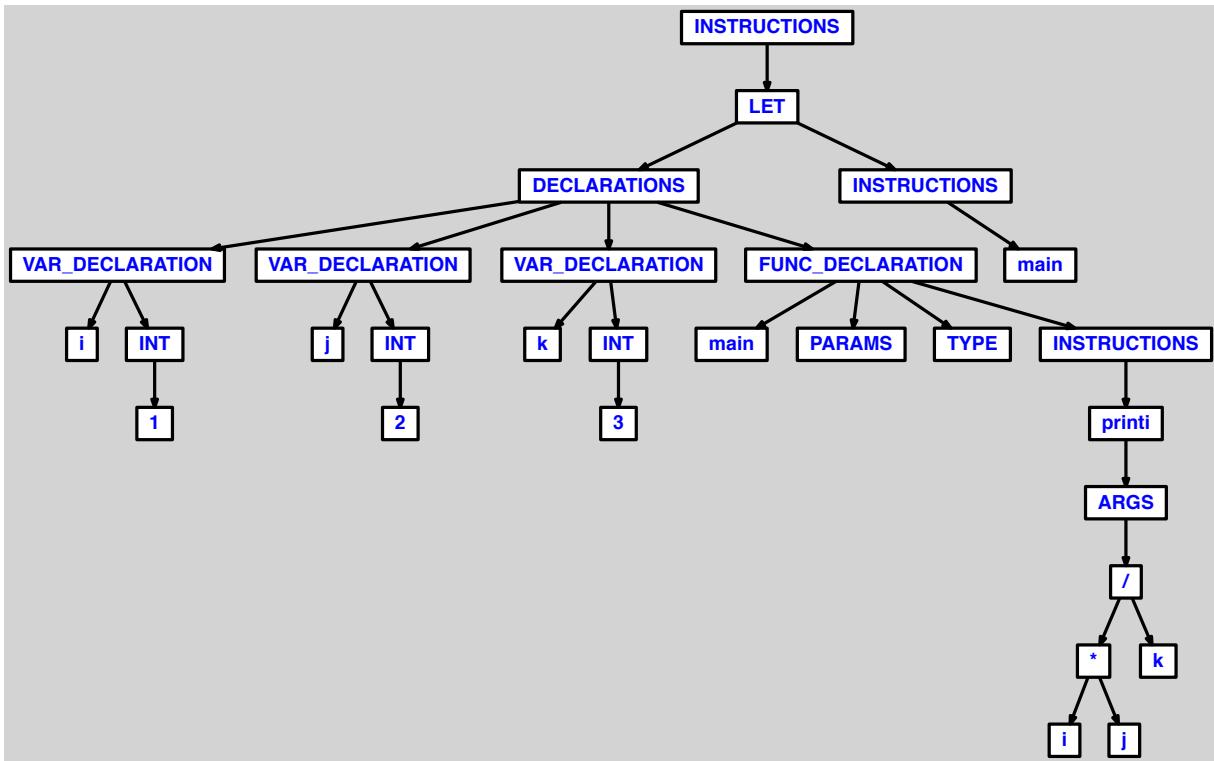
### 3.2.24 multiplication suivie de division, avec termes identifiés par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*j/k)
in main() end

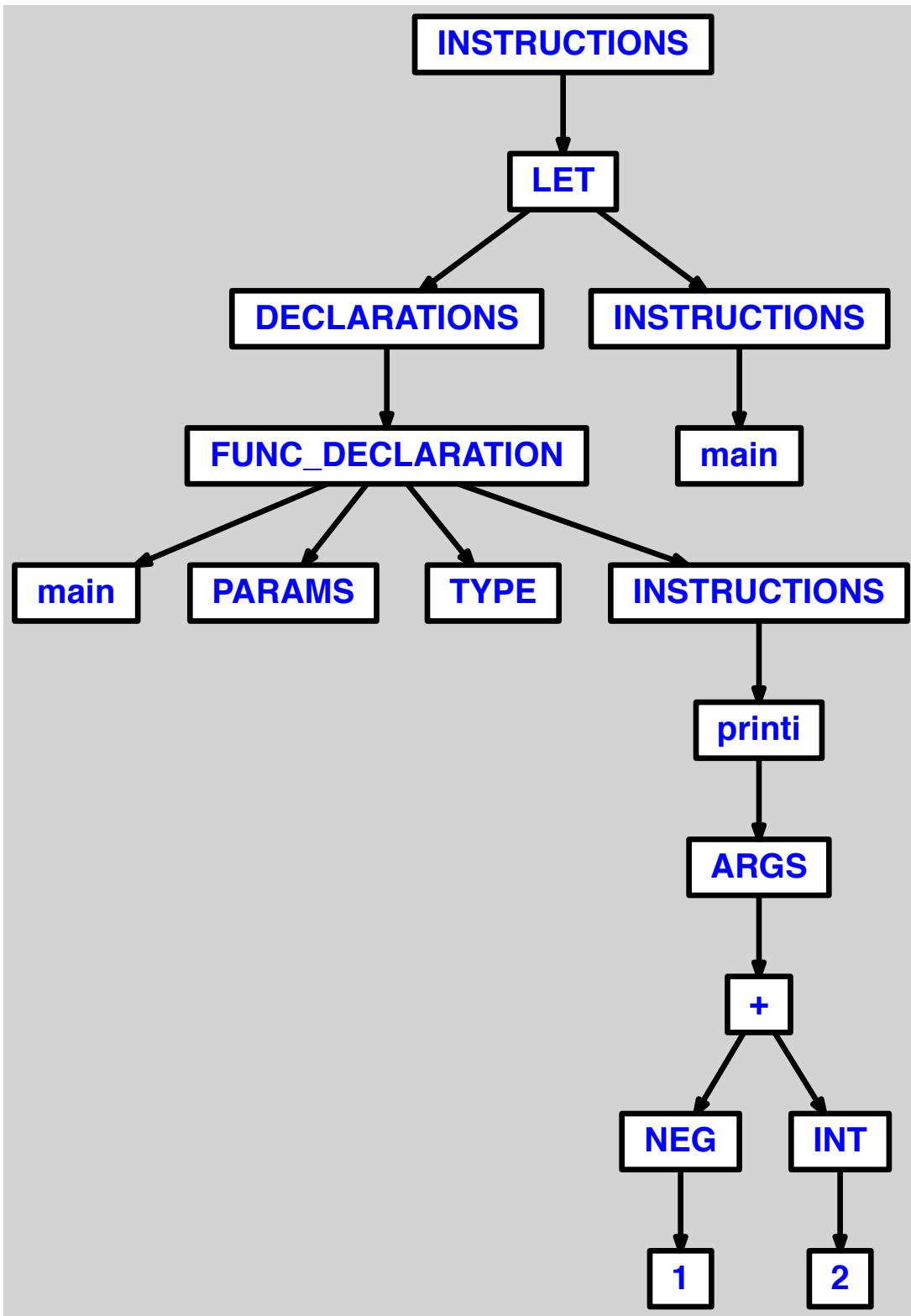
```



### 3.2.25 addition simple, à 2 termes, dont un ayant moins uneire

```

let
    function main() = printi(-1+2)
in main() end
  
```

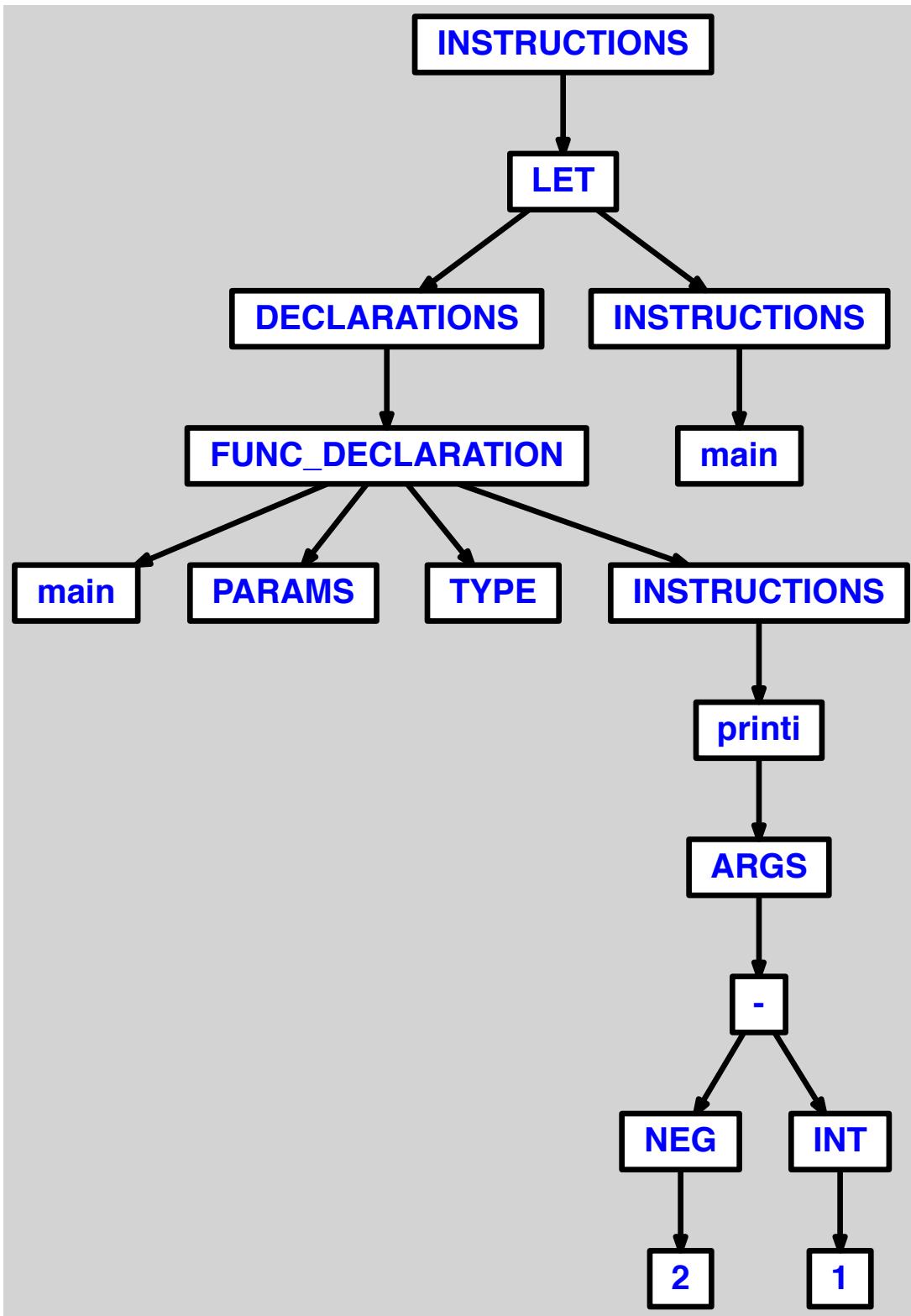


### 3.2.26 soustraction simple, a 2 termes, dont ayant moins unaire

let

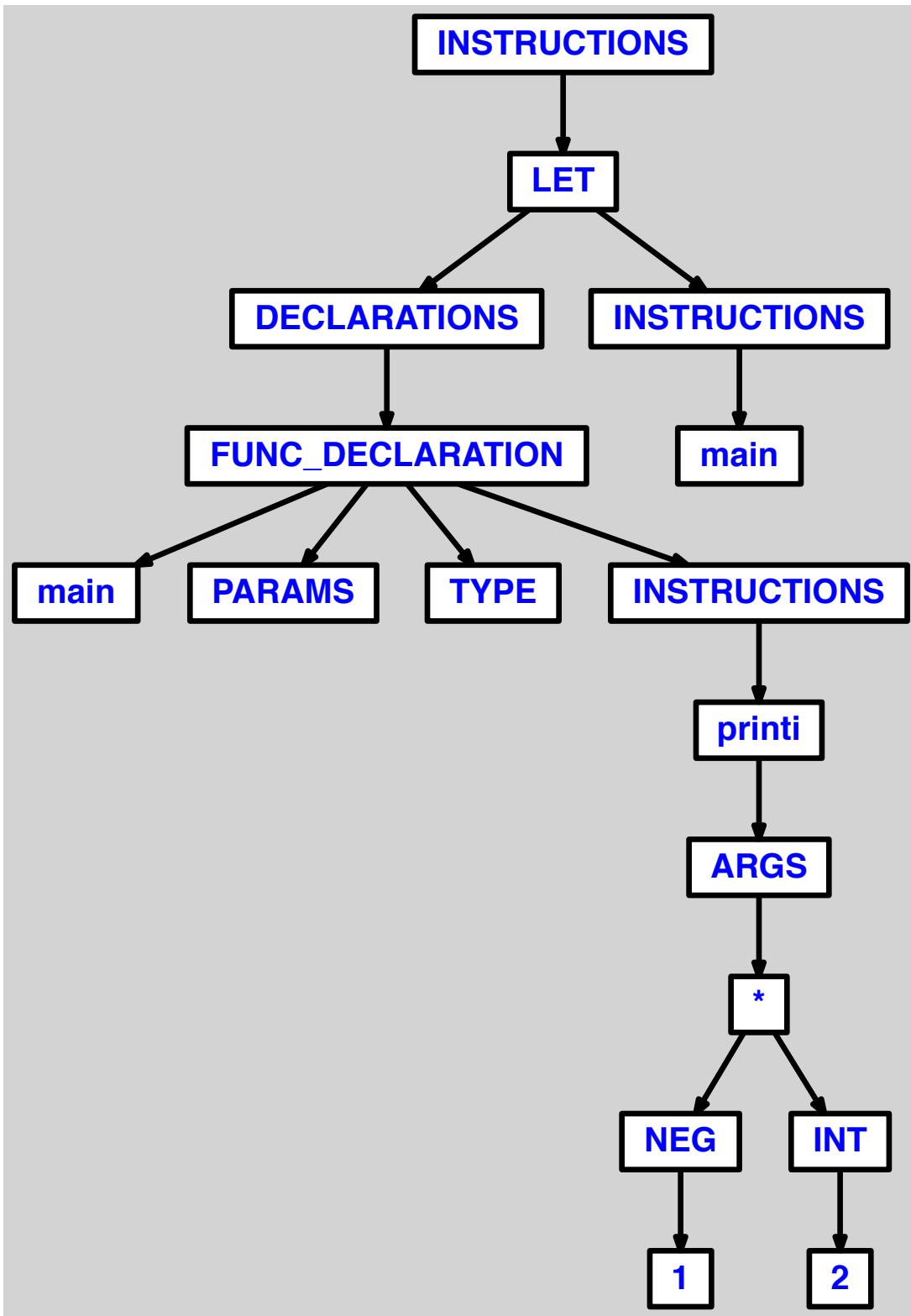
```
    function main() = printi(-2-1)
```

in main() end



### 3.2.27 multiplication simple, a 2 termes, dont un ayant moins unaire

```
let
    function main() = printi(-1*2)
in main() end
```

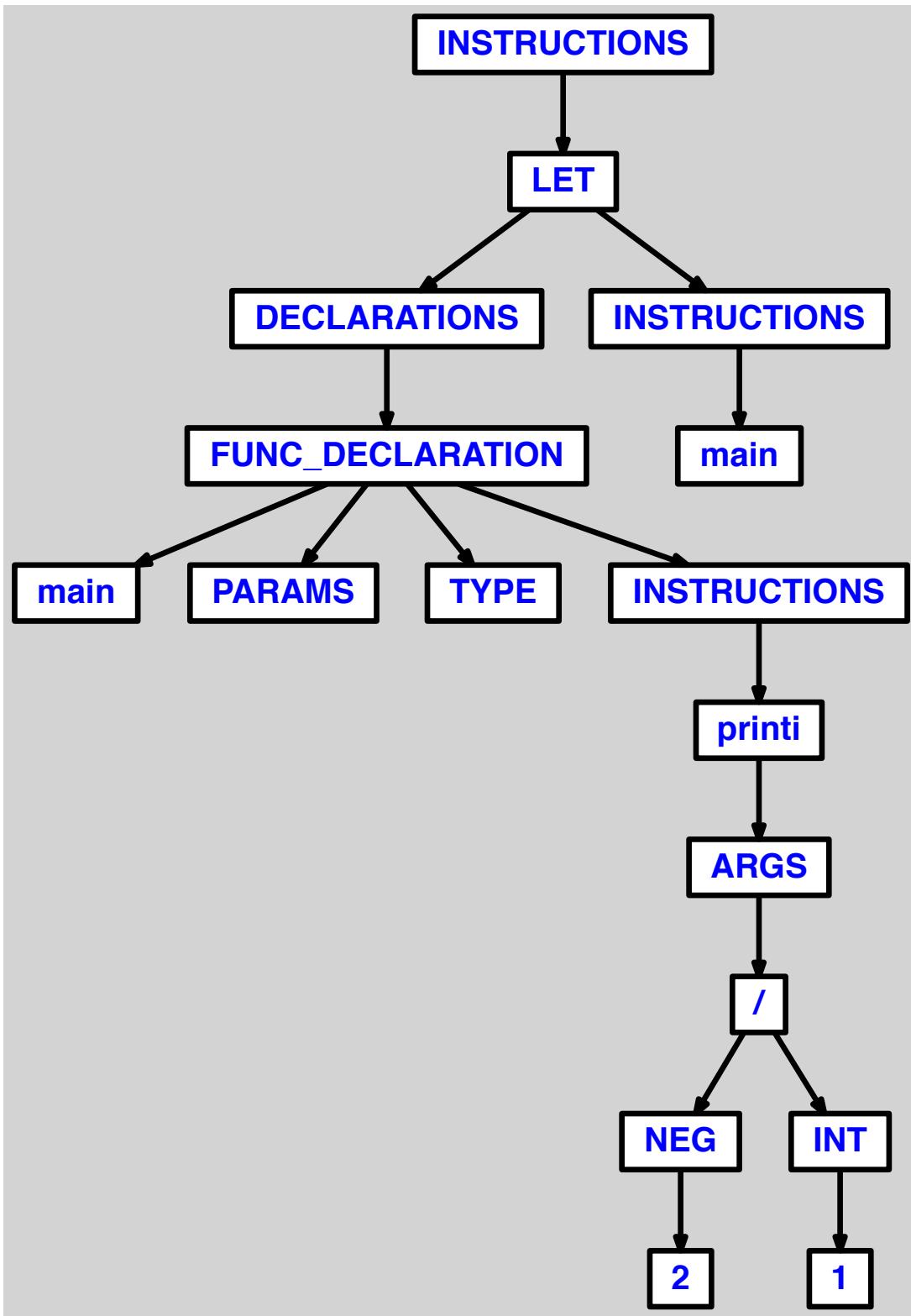


### 3.2.28 division simple, a 2 termes, dont un ayant moins uneire

let

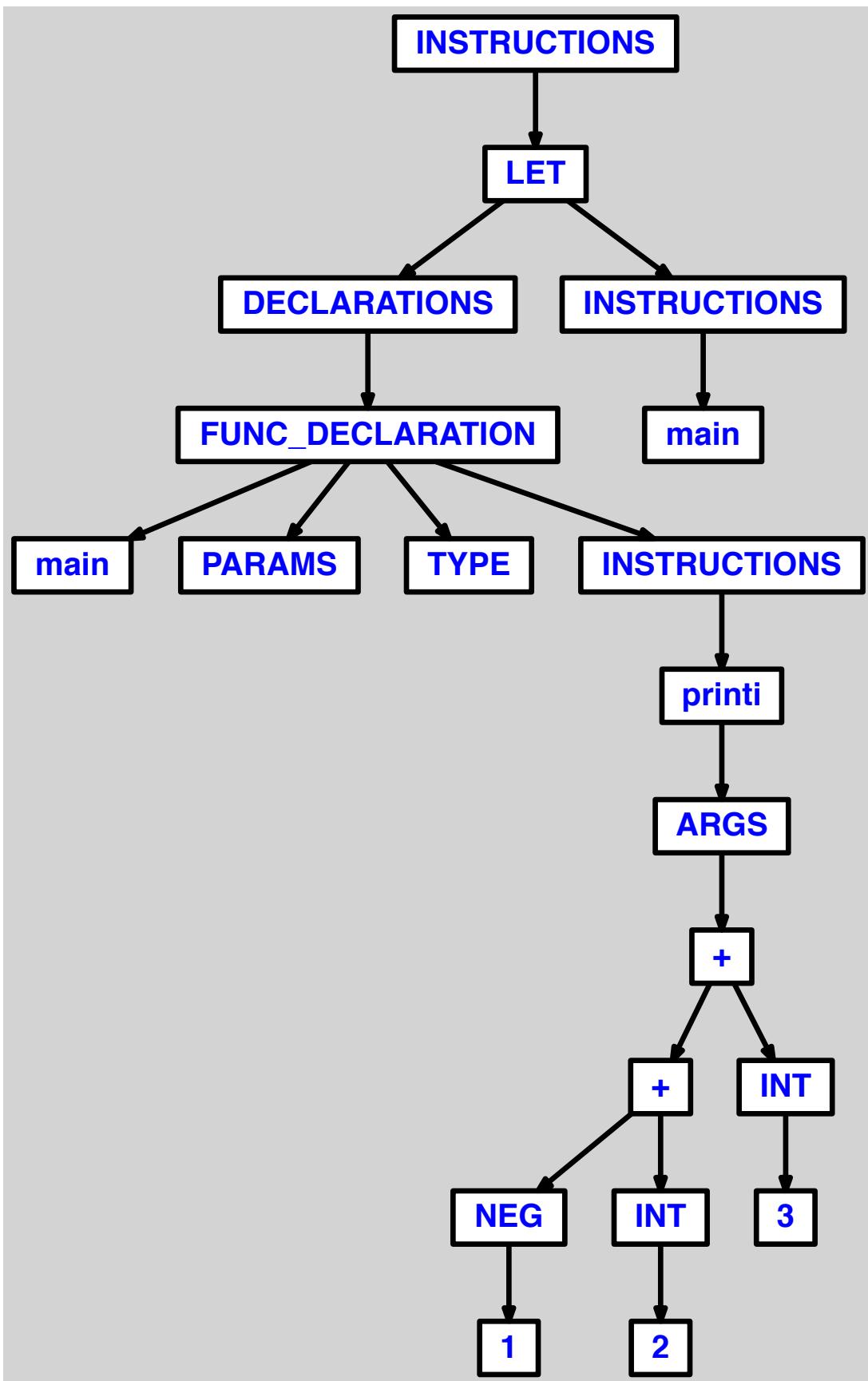
    function main() = printi(-2/1)

in main() end



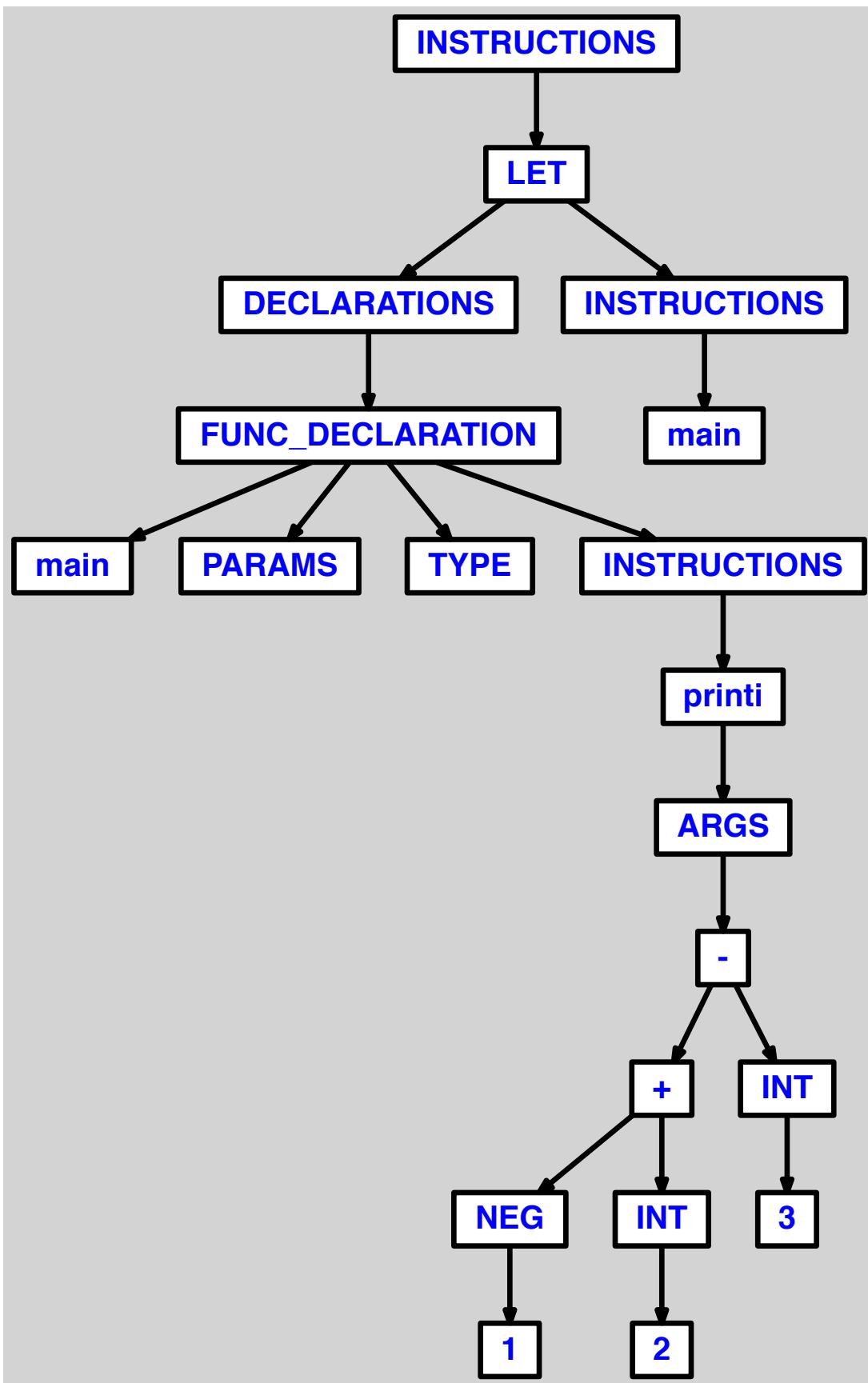
### 3.2.29 addition a 3 termes, dont un ayant moins unaire

```
let
    function main() = printi(-1+2+3)
in main() end
```



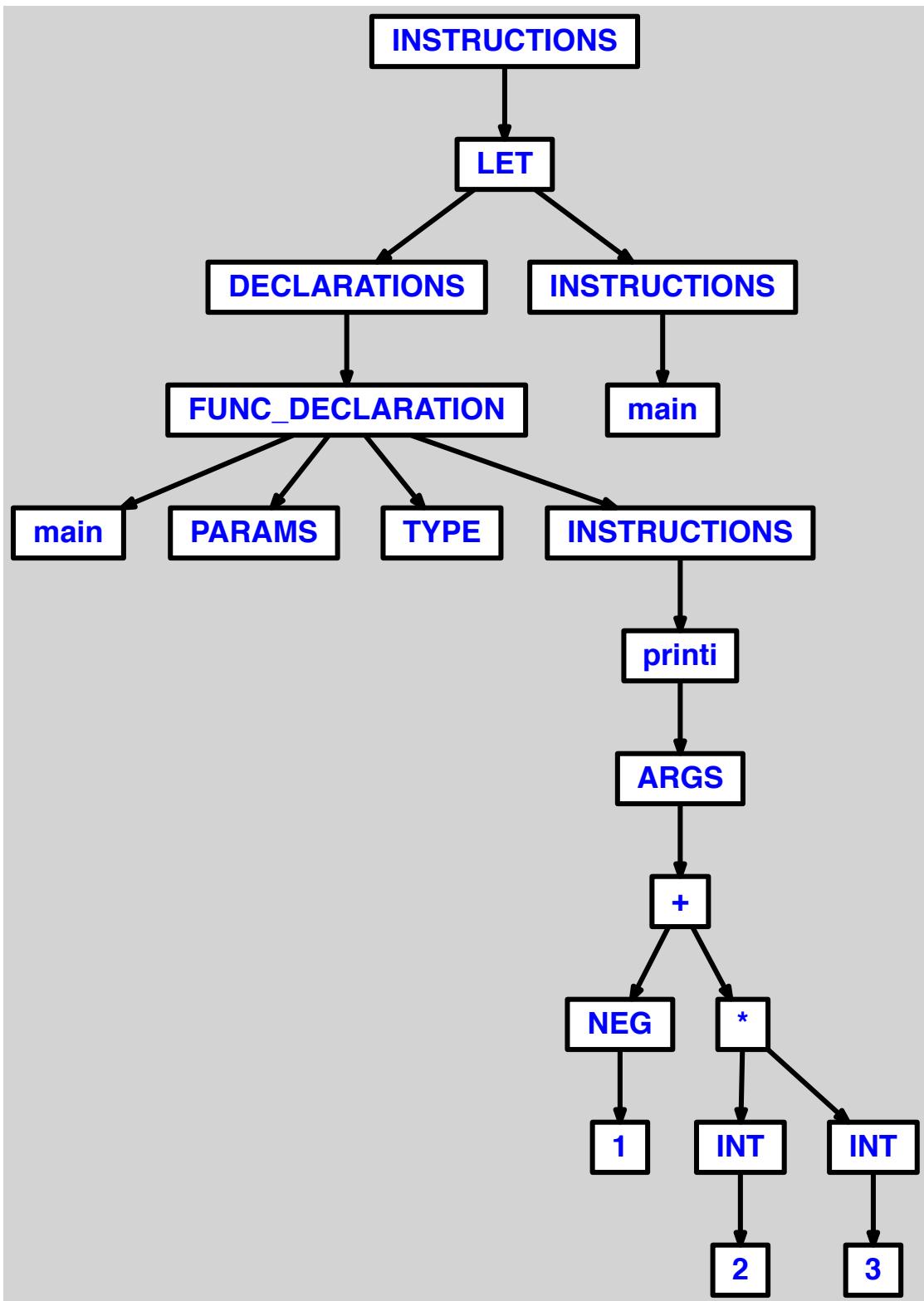
### 3.2.30 addition suivie de soustraction, avec un terme ayant moins unaire

```
let
    function main() = printi(-1+2-3)
in main() end
```



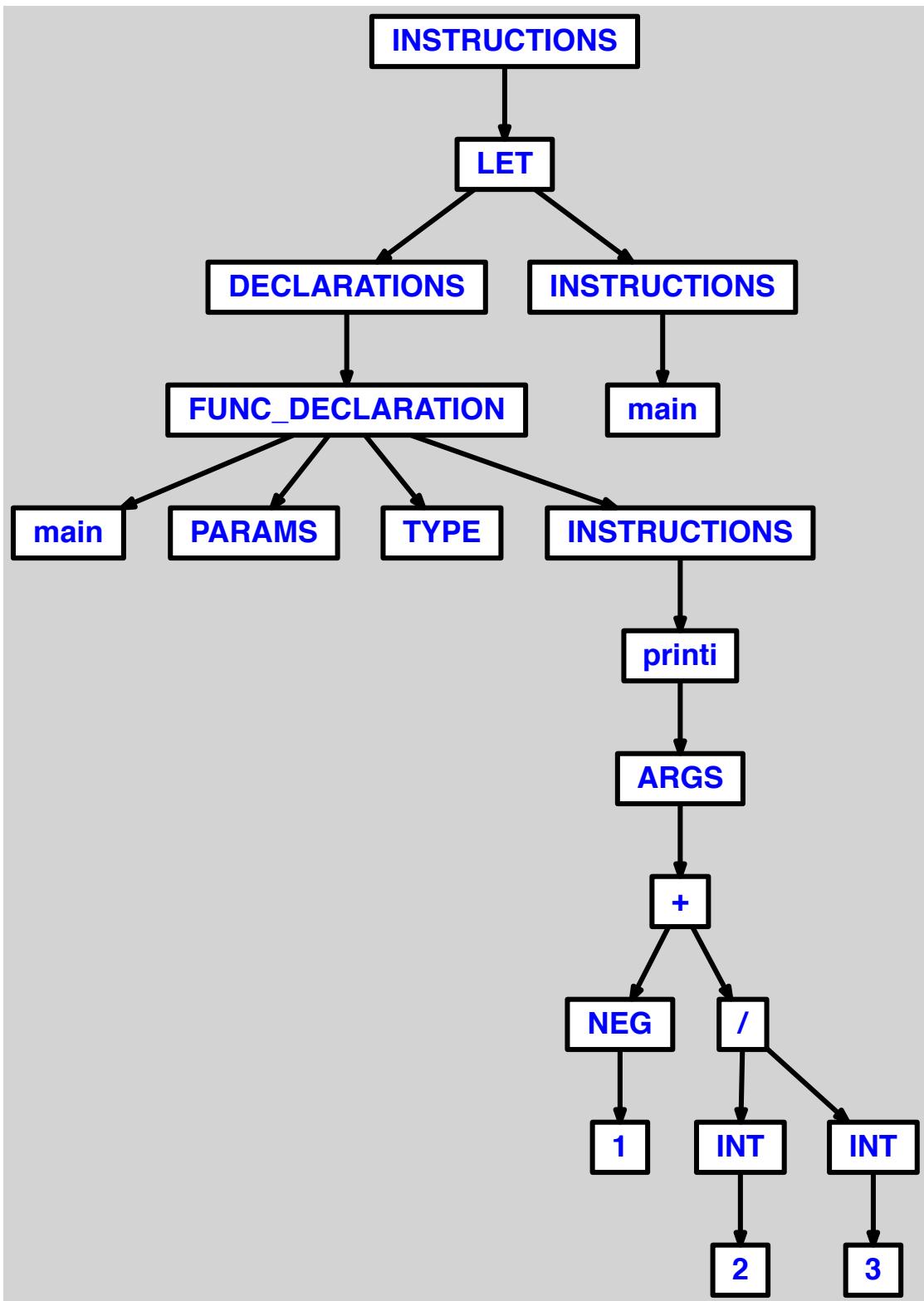
### 3.2.31 addition suivie de multiplication, avec un terme ayant moins unaire

```
let
    function main() = printi(-1+2*3)
in main() end
```



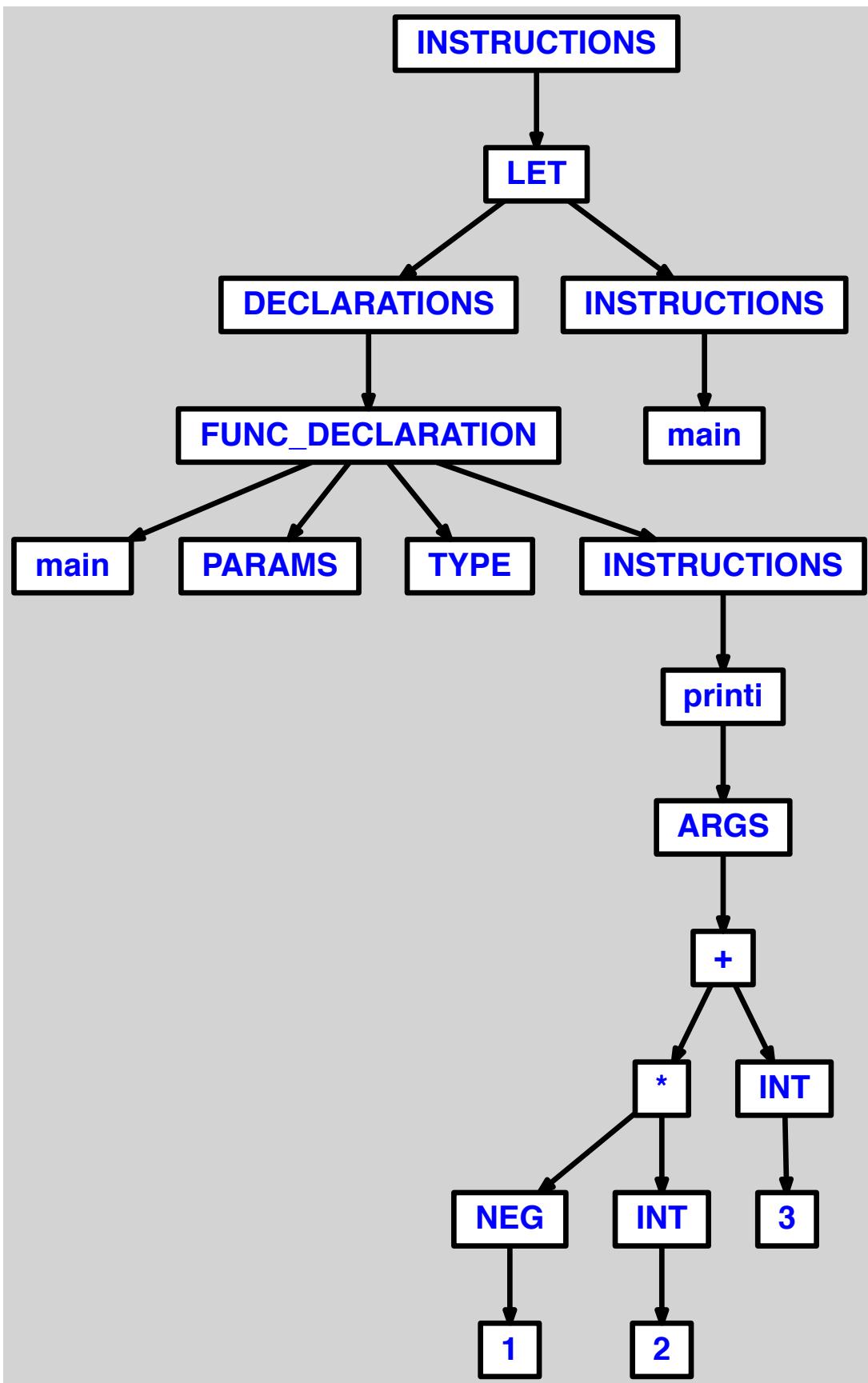
### 3.2.32 addition suivie de division, avec un terme ayant moins unaire

```
let
    function main() = printi(-1+2/3)
in main() end
```



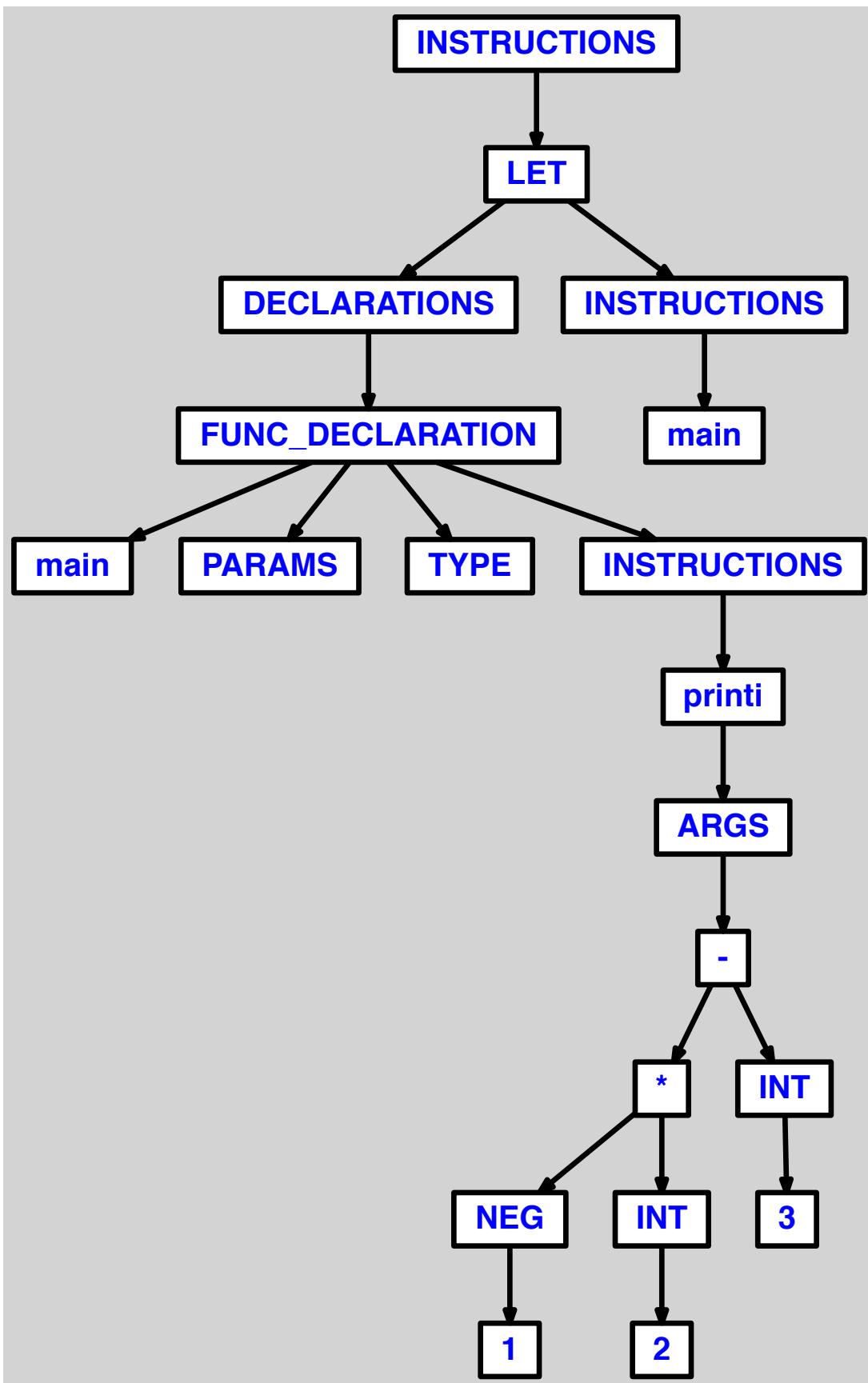
### 3.2.33 multiplication suivie d'addition, dont un terme ayant moins unaire

```
let
    function main() = printi(-1*2+3)
in main() end
```



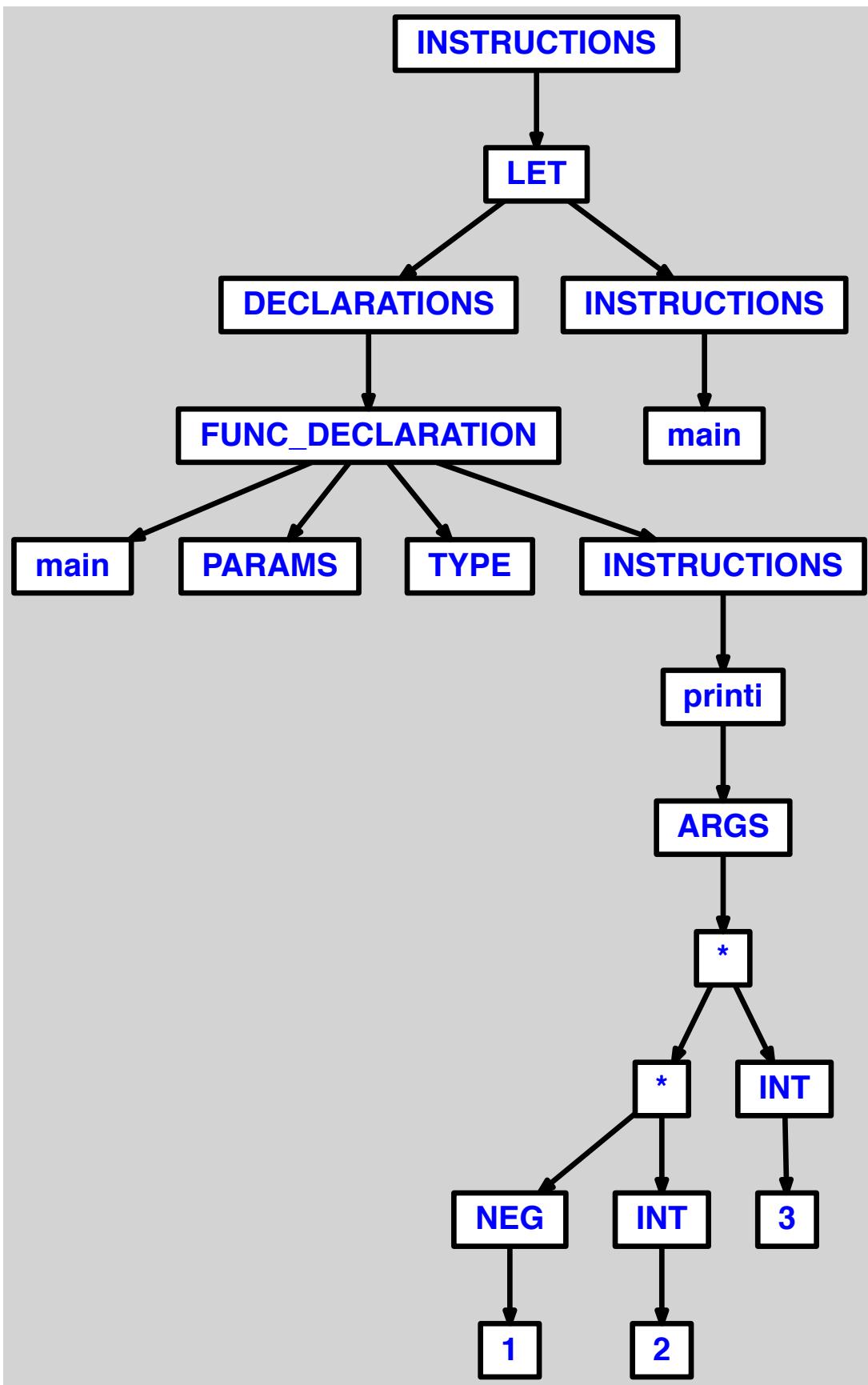
### 3.2.34 multiplication suivie de soustraction, dont un terme ayant moins unaire

```
let
    function main() = printi(-1*2-3)
in main() end
```



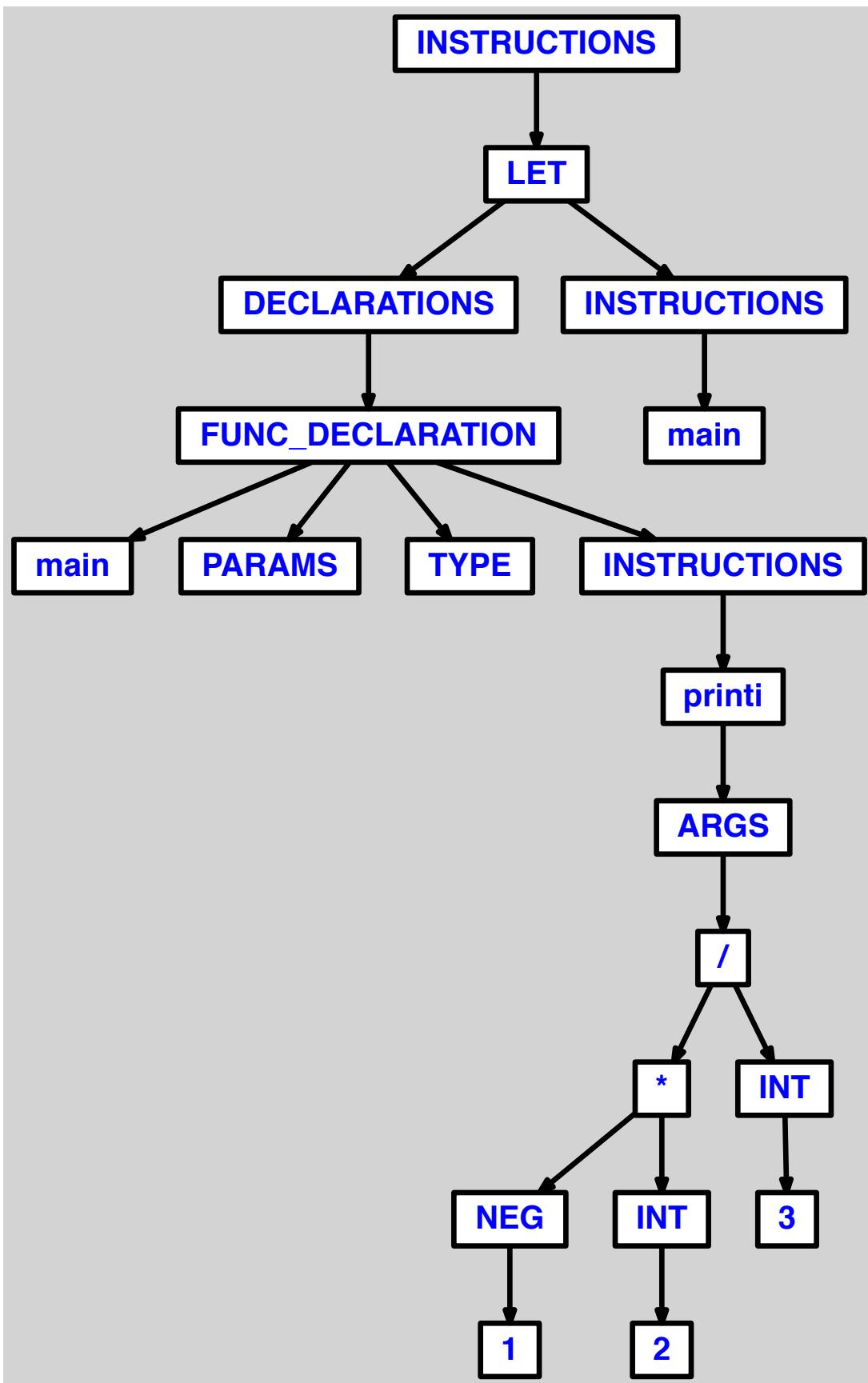
### 3.2.35 multiplication a 3 termes, dont un ayant moins unaire

```
let
    function main() = printi(-1*2*3)
in main() end
```



### 3.2.36 multiplication suivie de division, dont un terme ayant moins unaire

```
let
    function main() = printi(-1*2/3)
in main() end
```

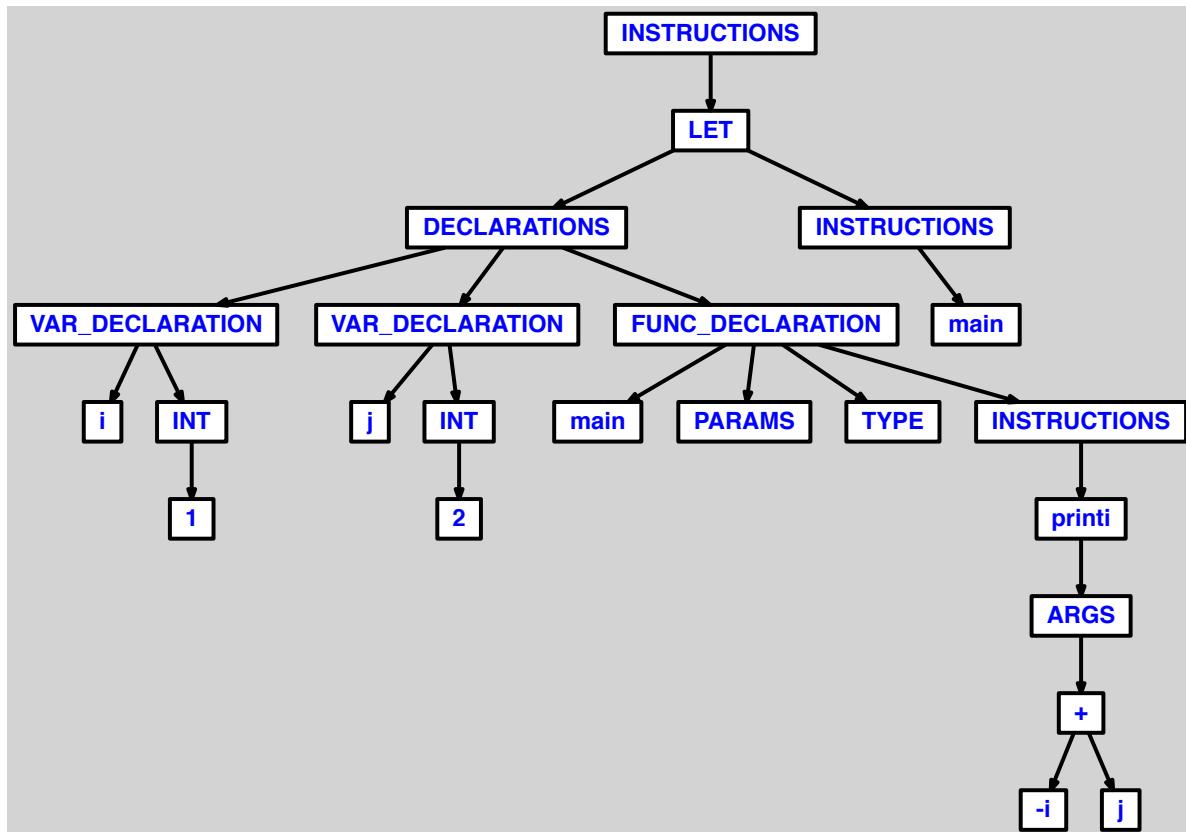


3.2.37 addition simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

let

```
var i := 1
var j := 2
```

```
function main() = printi(-i+j)
in main() end
```

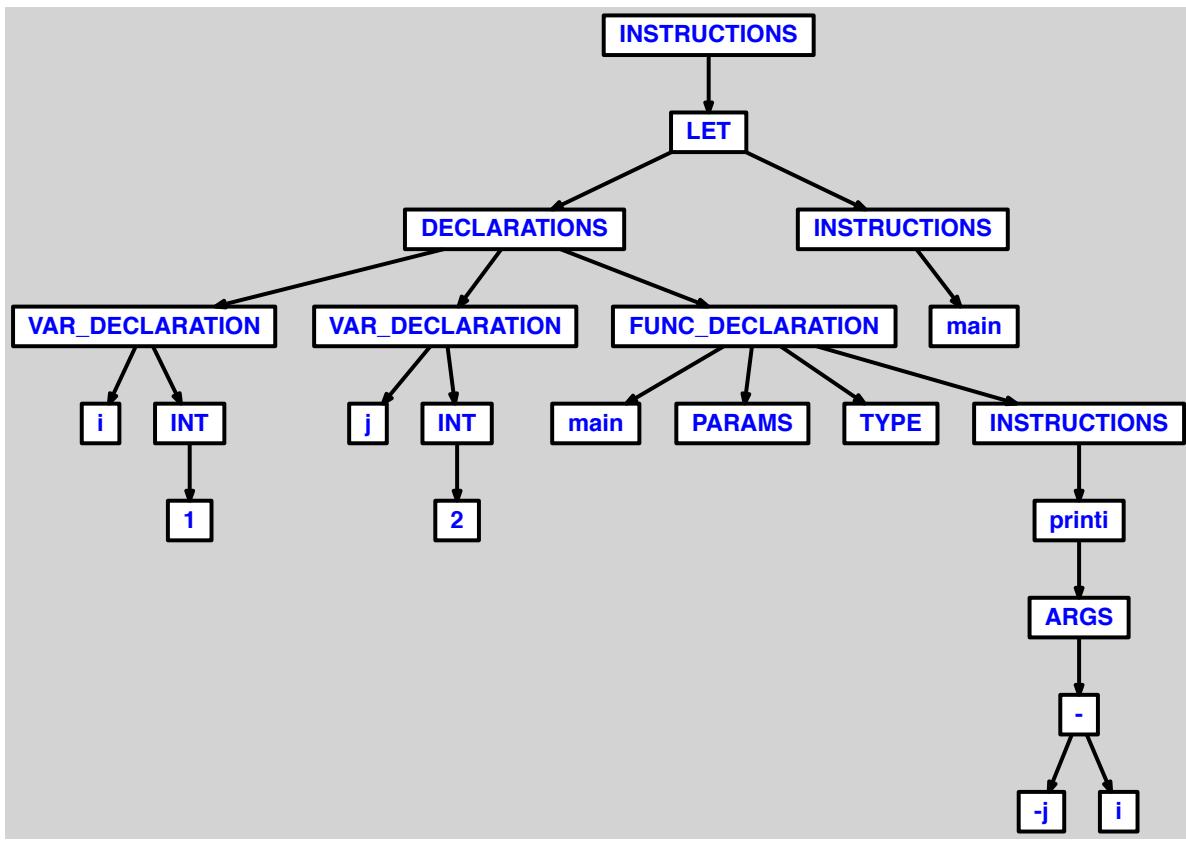


3.2.38 soustraction simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

let

```
var i := 1
var j := 2
```

```
function main() = printi(-j-i)
in main() end
```

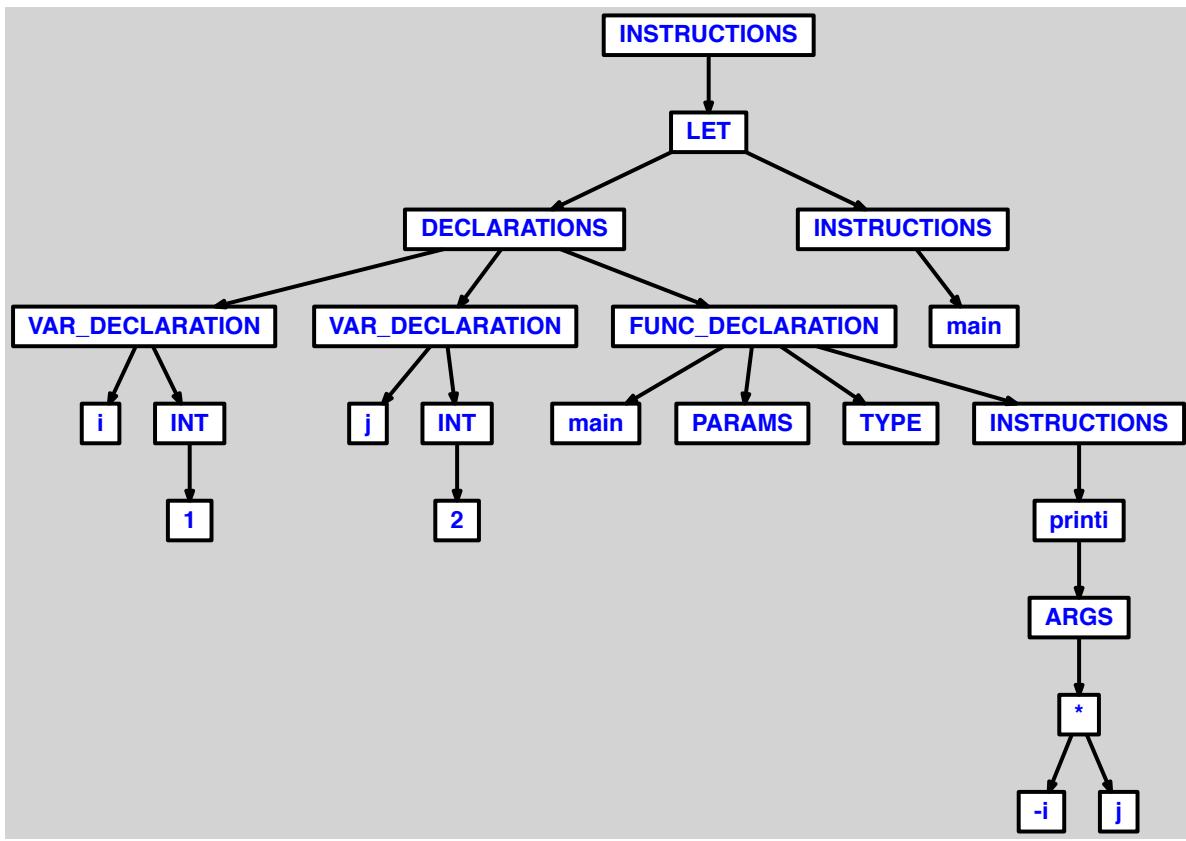


### 3.2.39 multiplication simple, à 2 termes, identifiés par des variables, dont une ayant moins uneire

let

```
var i := 1
var j := 2
```

```
function main() = printi(-i*j)
in main() end
```

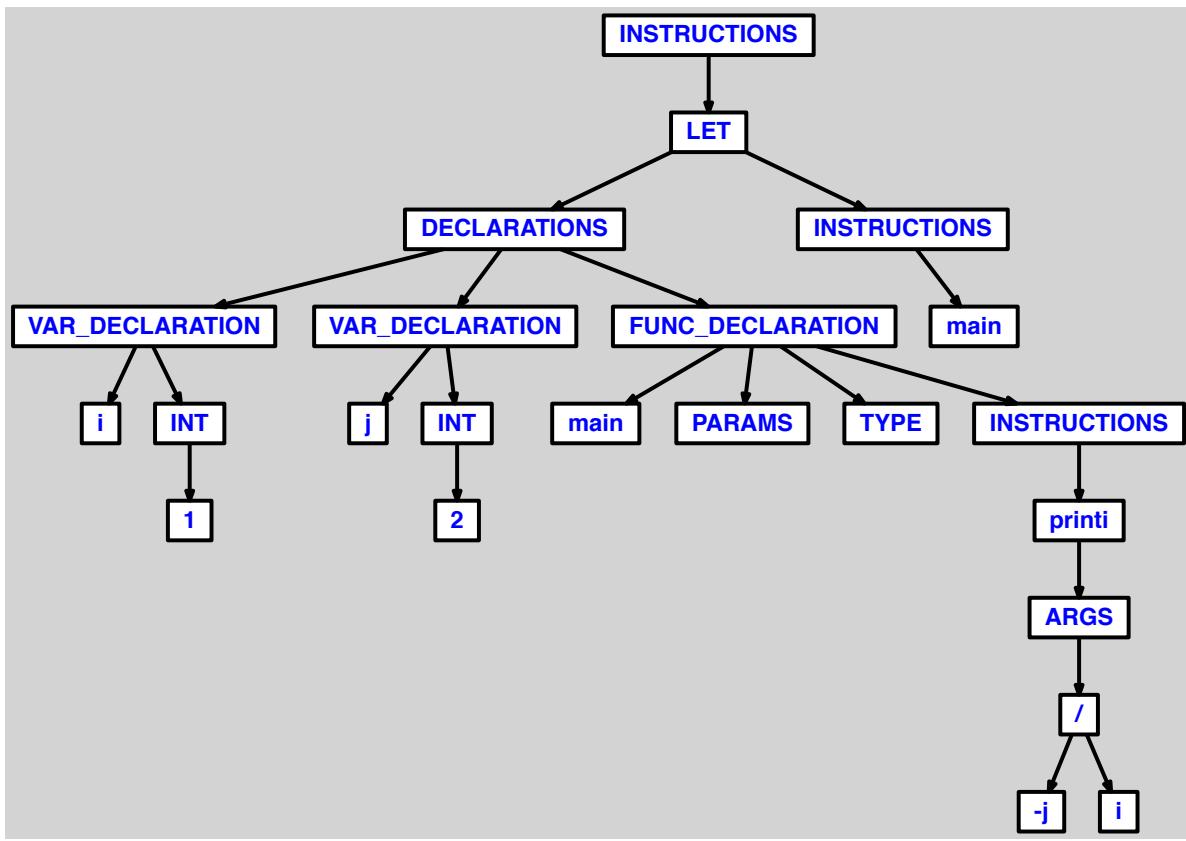


### 3.2.40 division simple, à 2 termes, identifiés par des variables, dont une ayant moins uneire

```

let
  var i := 1
  var j := 2

  function main() = printi(-j/i)
in main() end
  
```

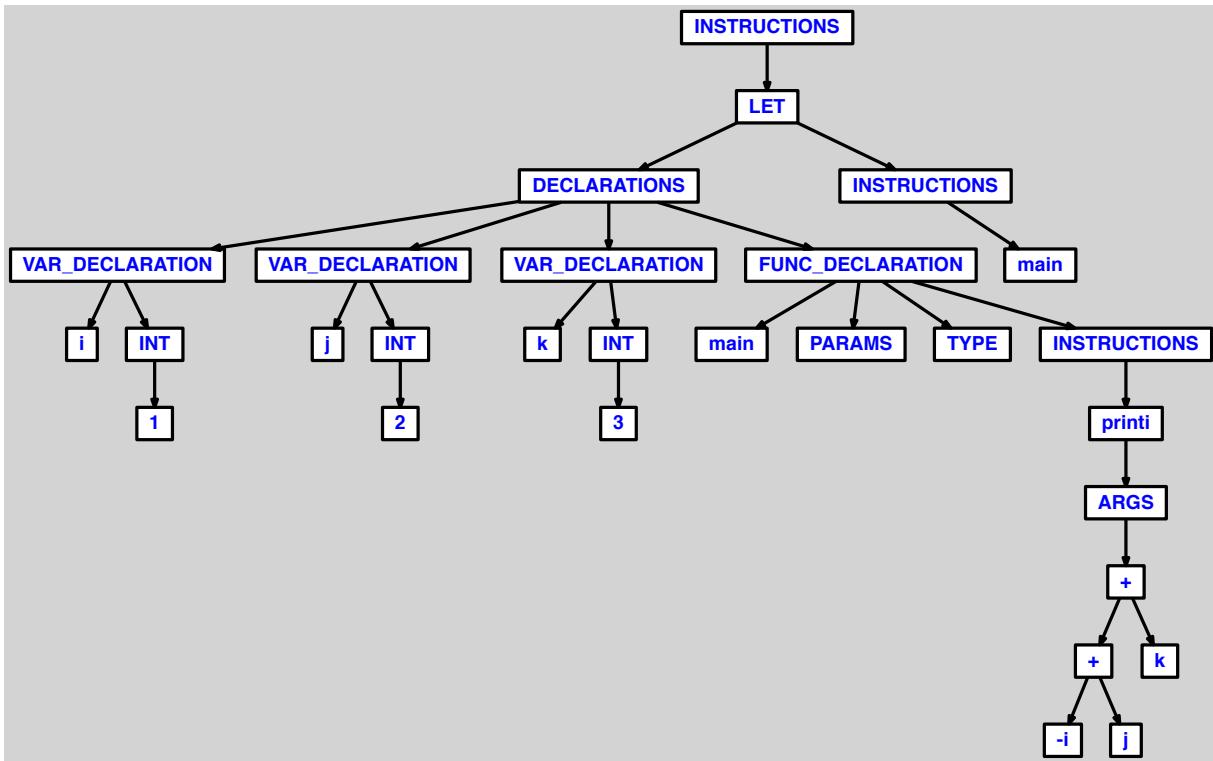


### 3.2.41 addition a 3 termes, identifies par des variables, dont une ayant moins uneire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i+j+k)
in main() end
  
```



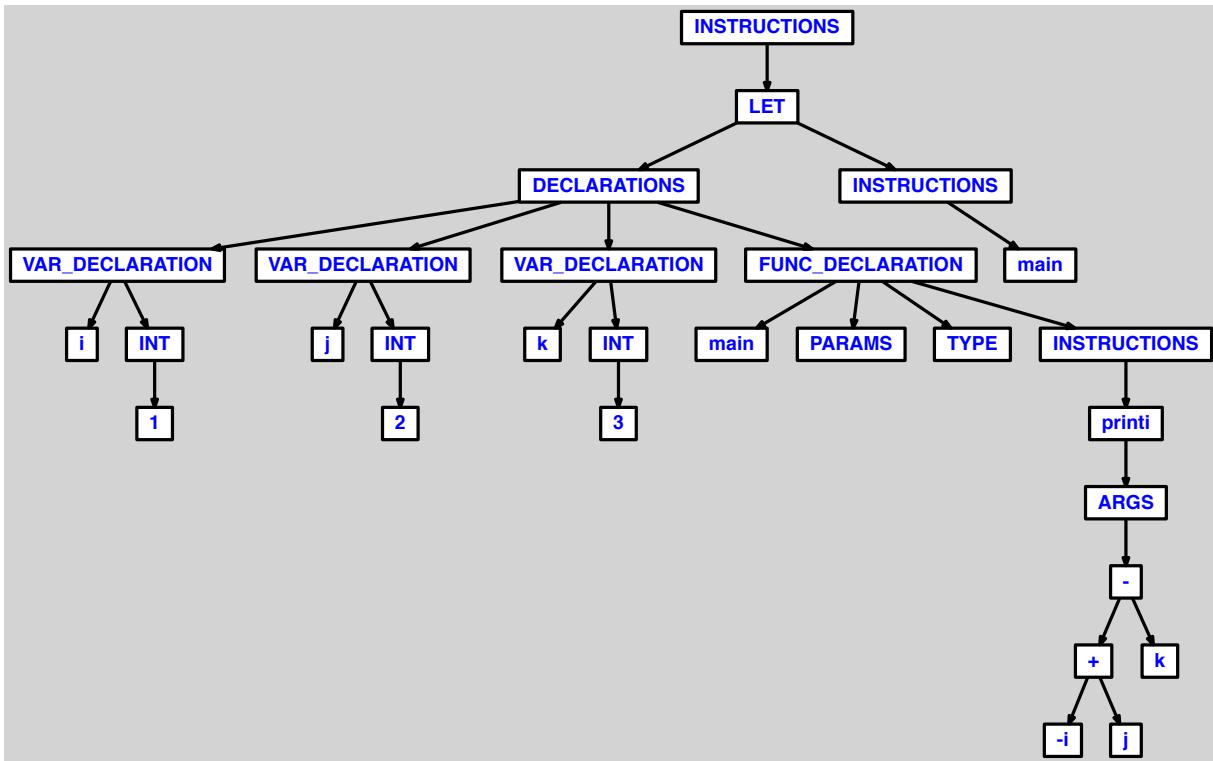
### 3.2.42 addition suivie de soustraction, avec termes identifiés par variables, dont une ayant moins unaire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i+j-k)
in main() end

```



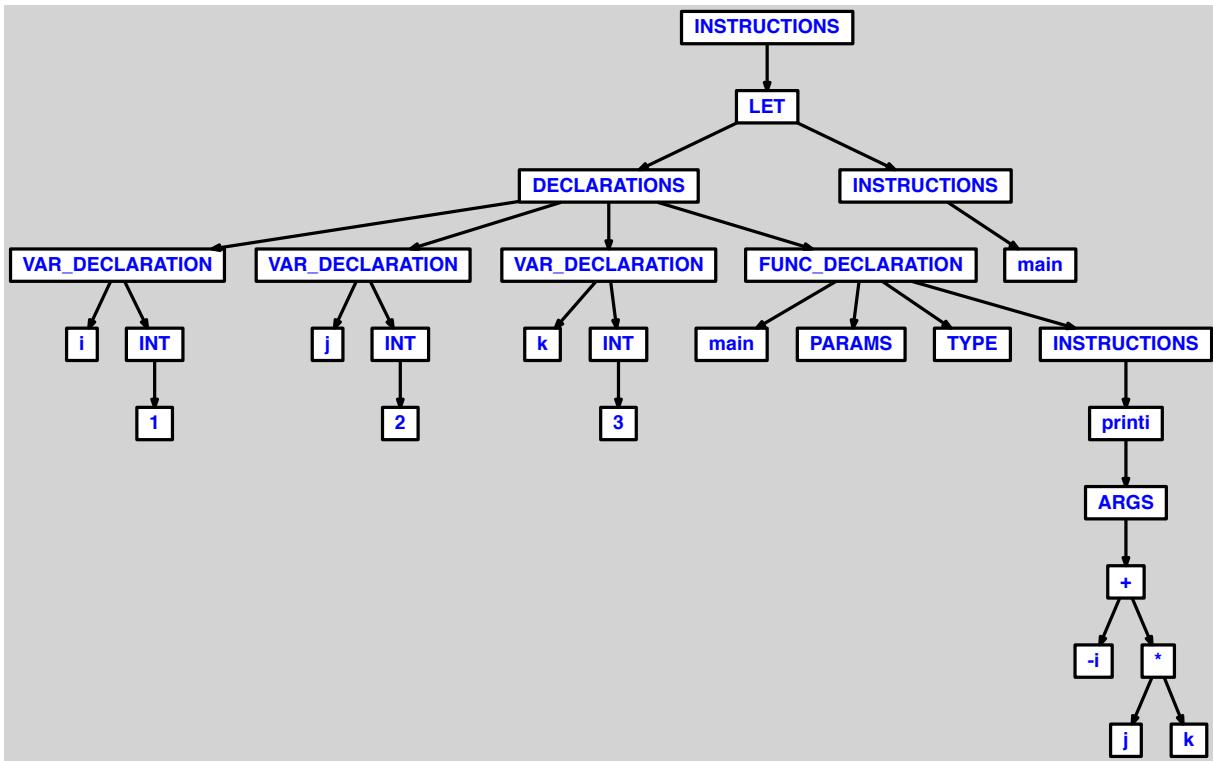
### 3.2.43 addition suivie de multiplication, avec termes identifies par variables, dont une ayant moins unaire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i+j*k)
in main() end

```

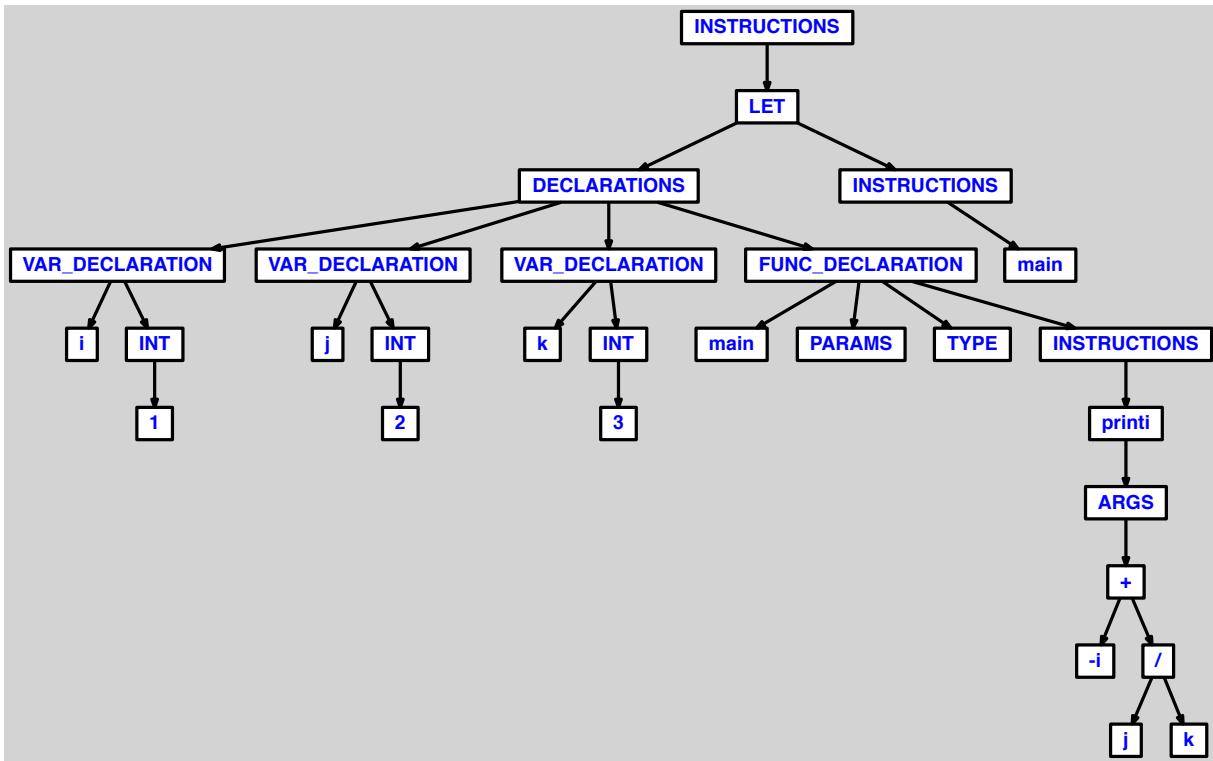


### 3.2.44 addition suivie de division, avec termes identifiés par variables, dont une ayant moins uneire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i+j/k)
in main() end
  
```

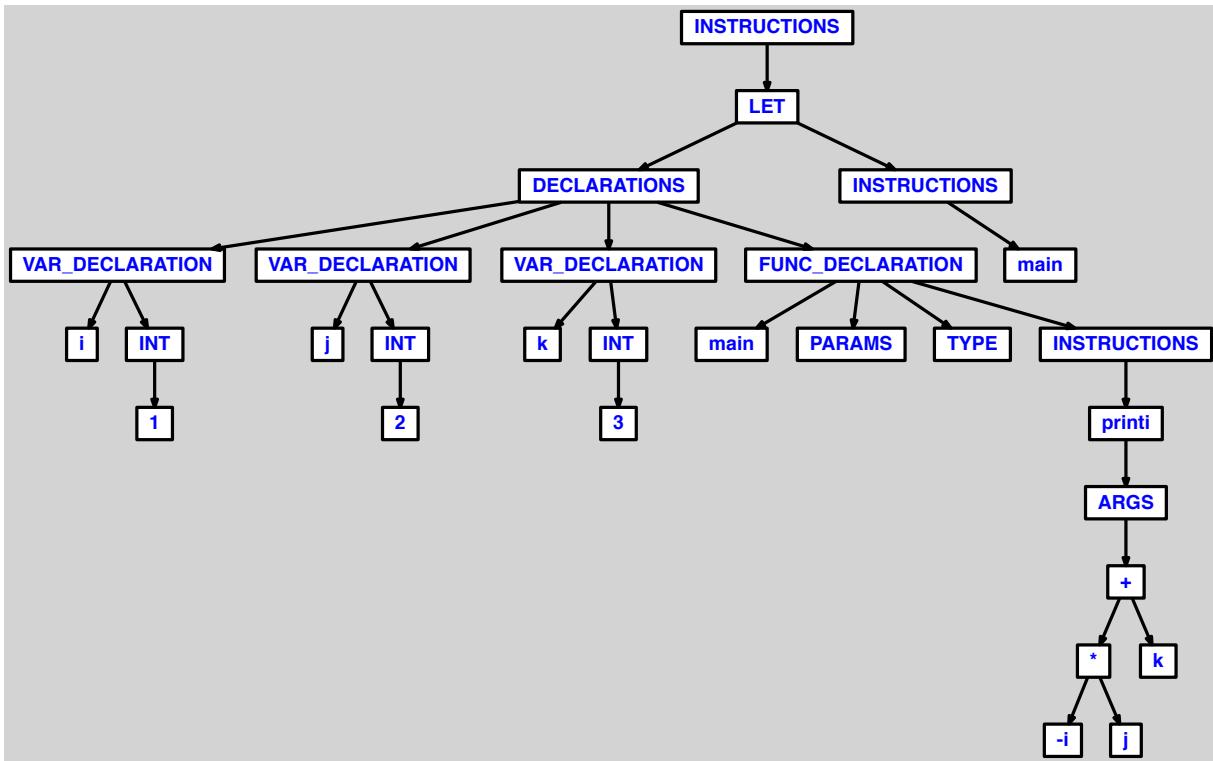


### 3.2.45 multiplication suivie d'addition, avec termes identifiés par variables, dont une ayant moins unaire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i*j+k)
in main() end
  
```



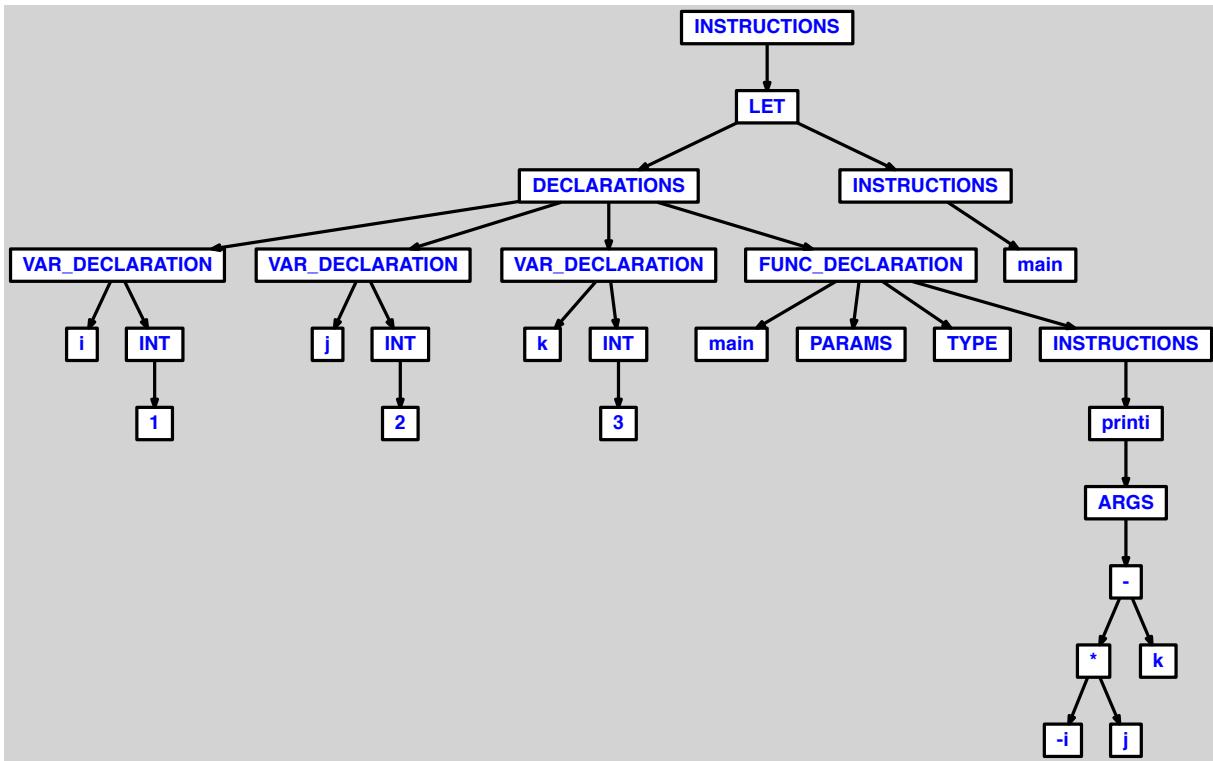
### 3.2.46 multiplication suivie de soustraction, avec termes identifiés par variables, dont une ayant moins uneire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i*j-k)
in main() end

```



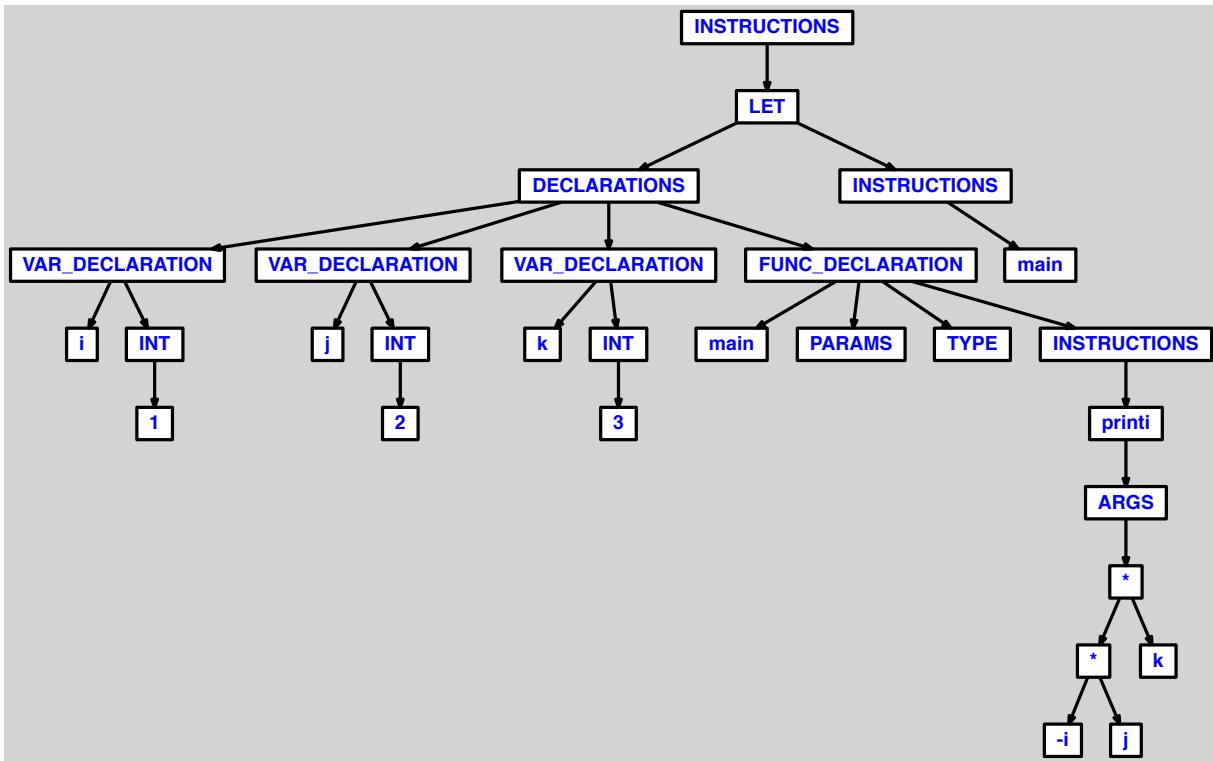
### 3.2.47 multiplication a 3 termes, identifiés par des variables, dont une ayant moins uneire

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(-i*j*k)
in main() end

```



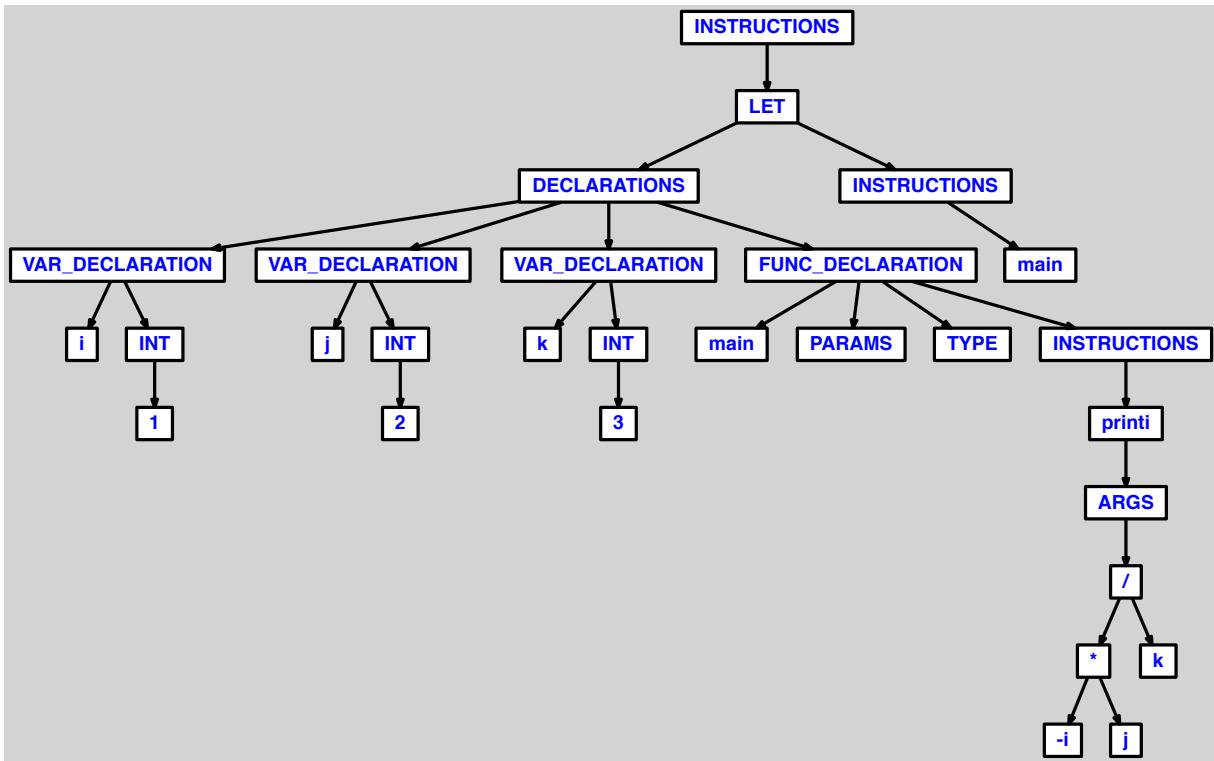
### 3.2.48 multiplication suivie de division, avec termes identifiés par variables, dont une ayant moins unaire

```

let
  var i := 1
  var j := 2
  var k := 3

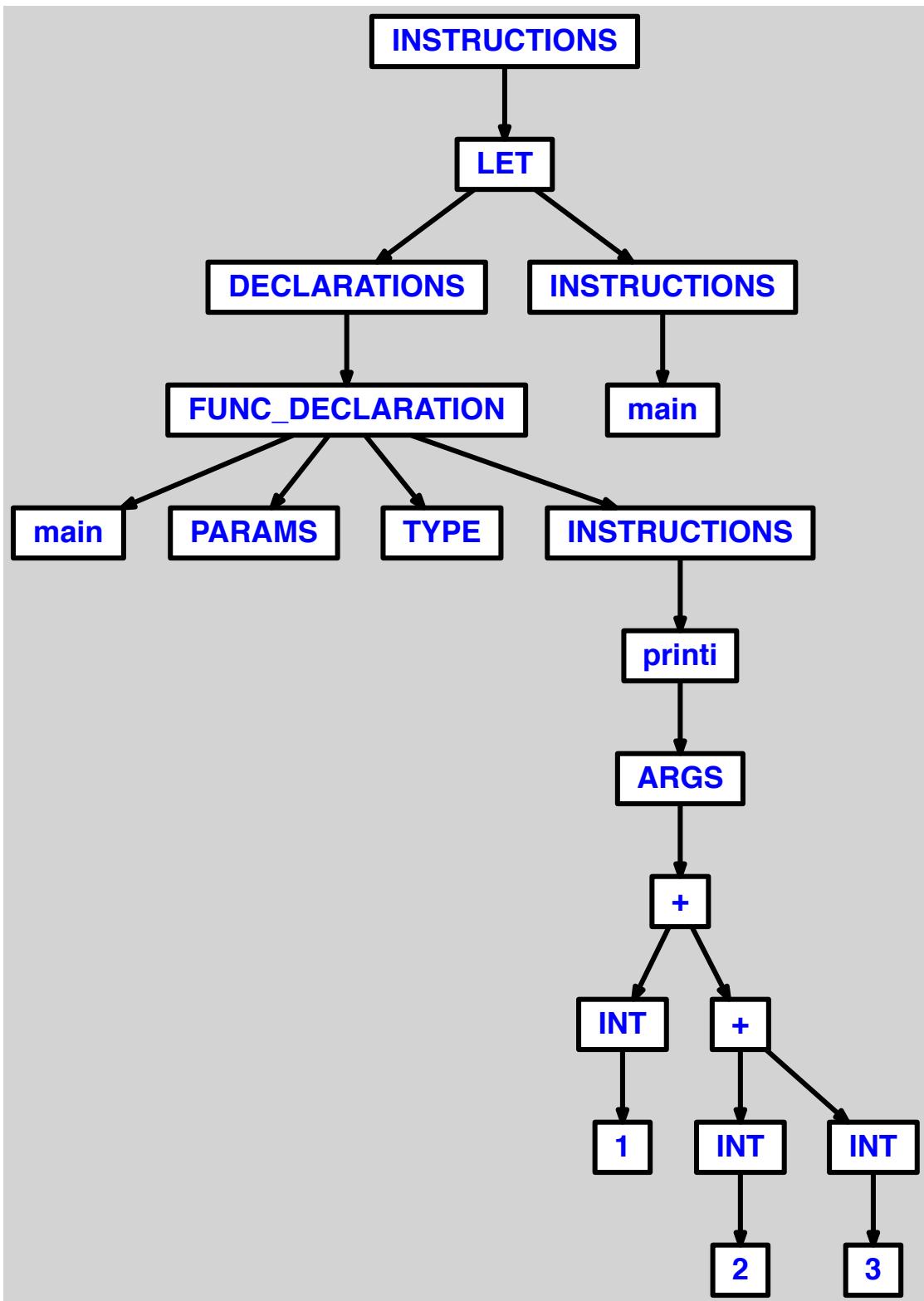
  function main() = printi(-i*j/k)
in main() end

```



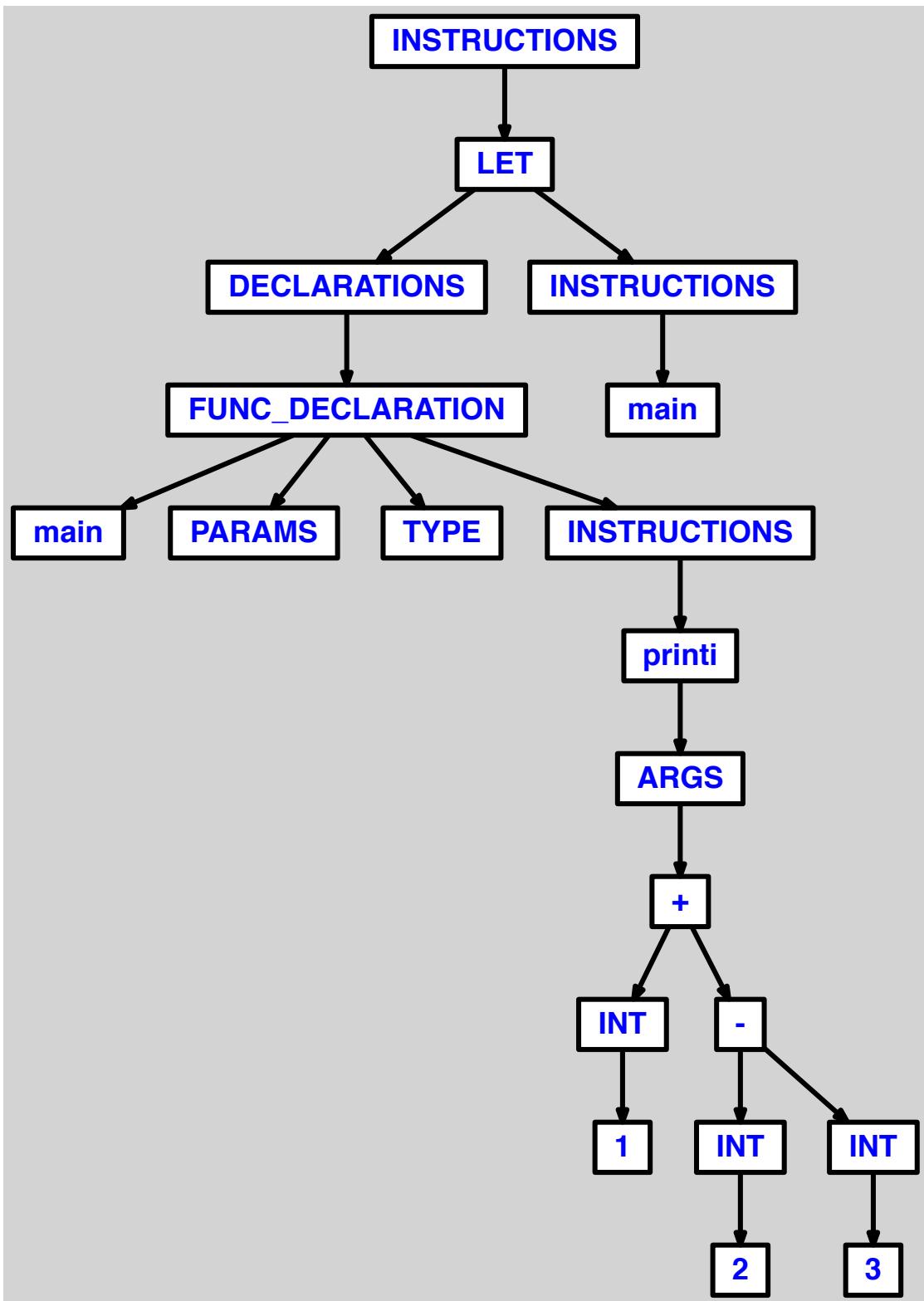
### 3.2.49 addition a 3 termes, avec parenthesage des 2 termes a droite

```
let function main() = printi(1+(2+3)) in main() end
```



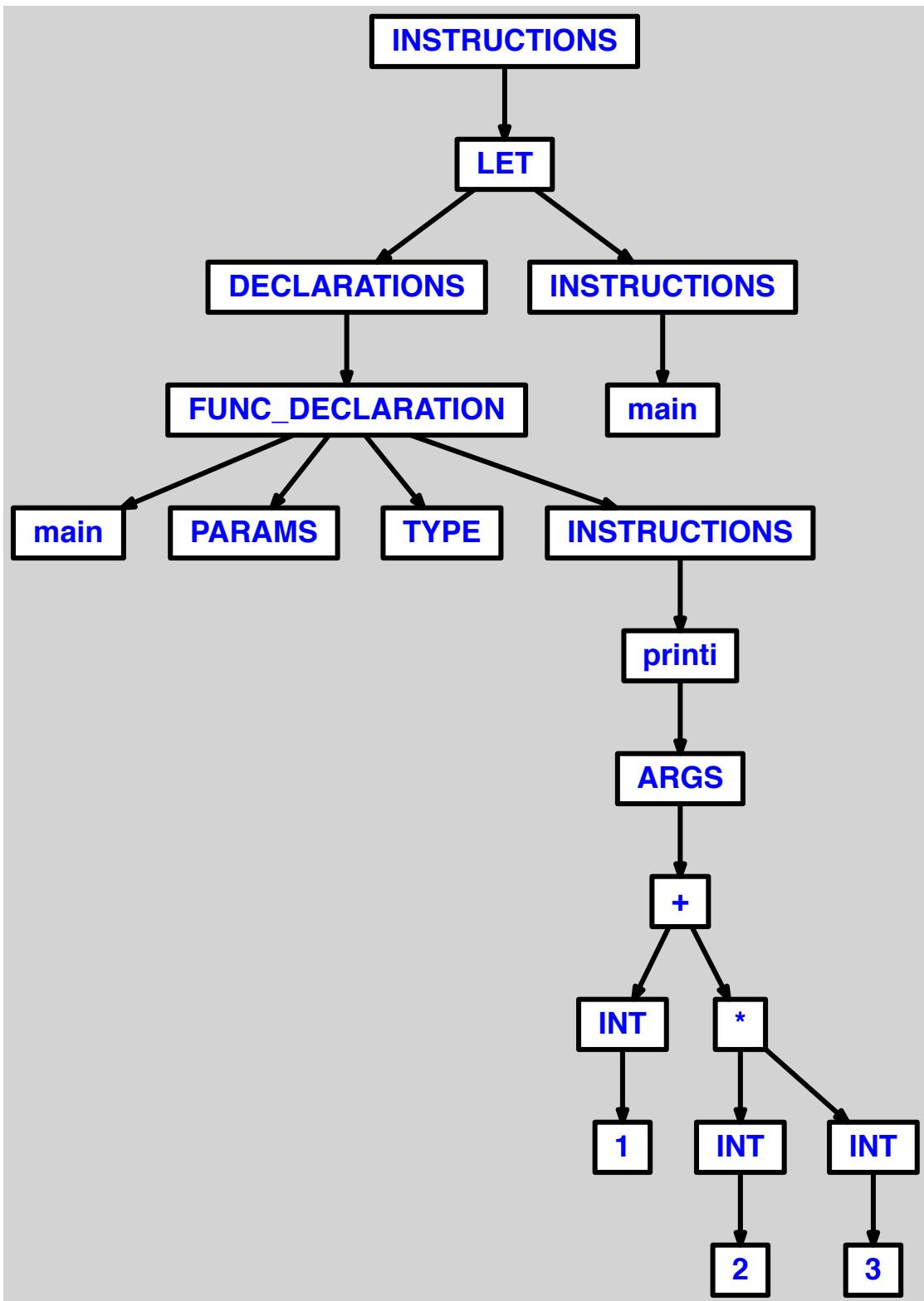
### 3.2.50 addition suivie de soustraction, avec parenthesage des 2 termes à droite

```
let function main() = printi(1+(2-3)) in main() end
```



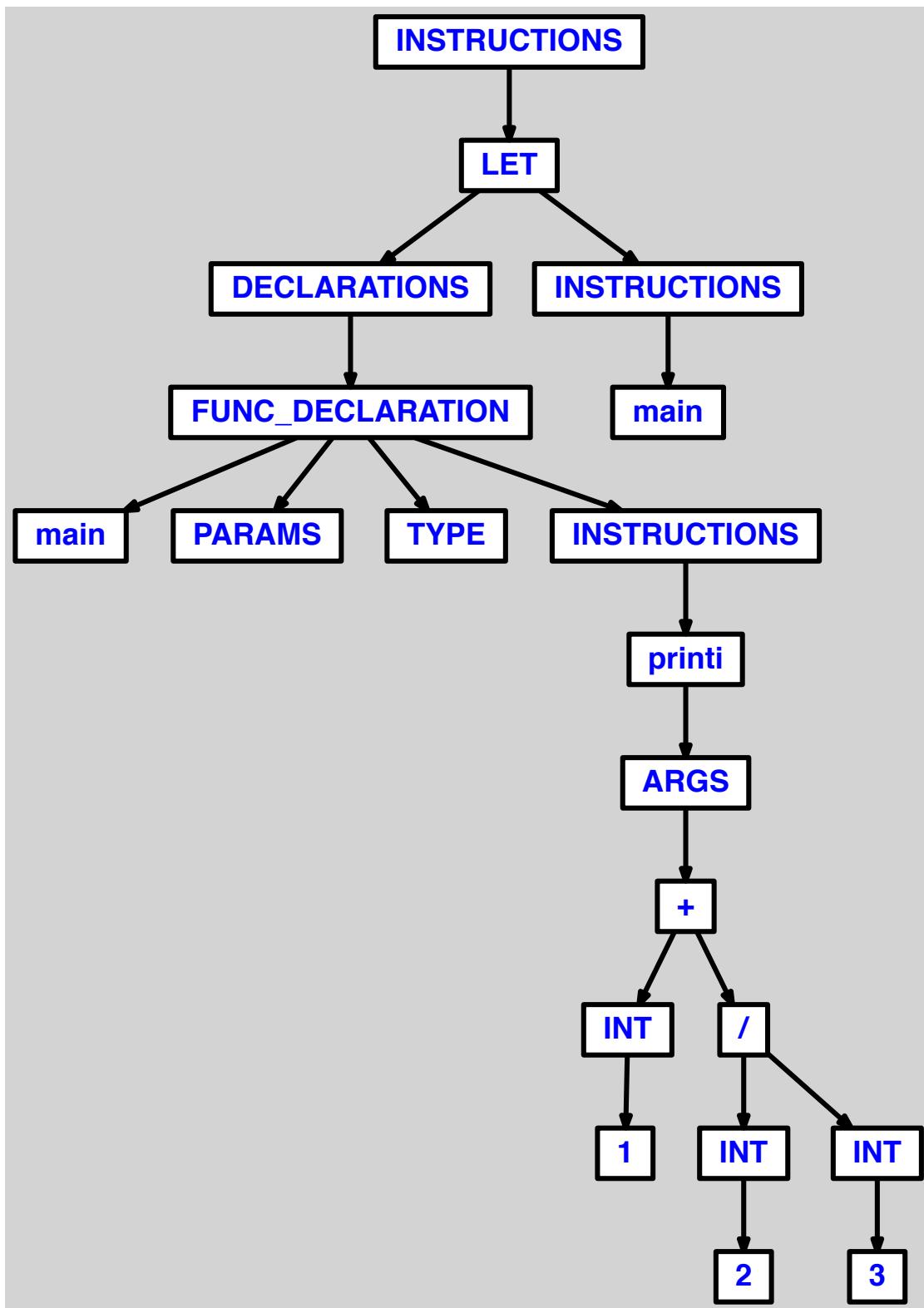
### 3.2.51 addition suivie de multiplication, avec parenthesage des 2 termes à droite

```
let function main() = printi(1+(2*3)) in main() end
```



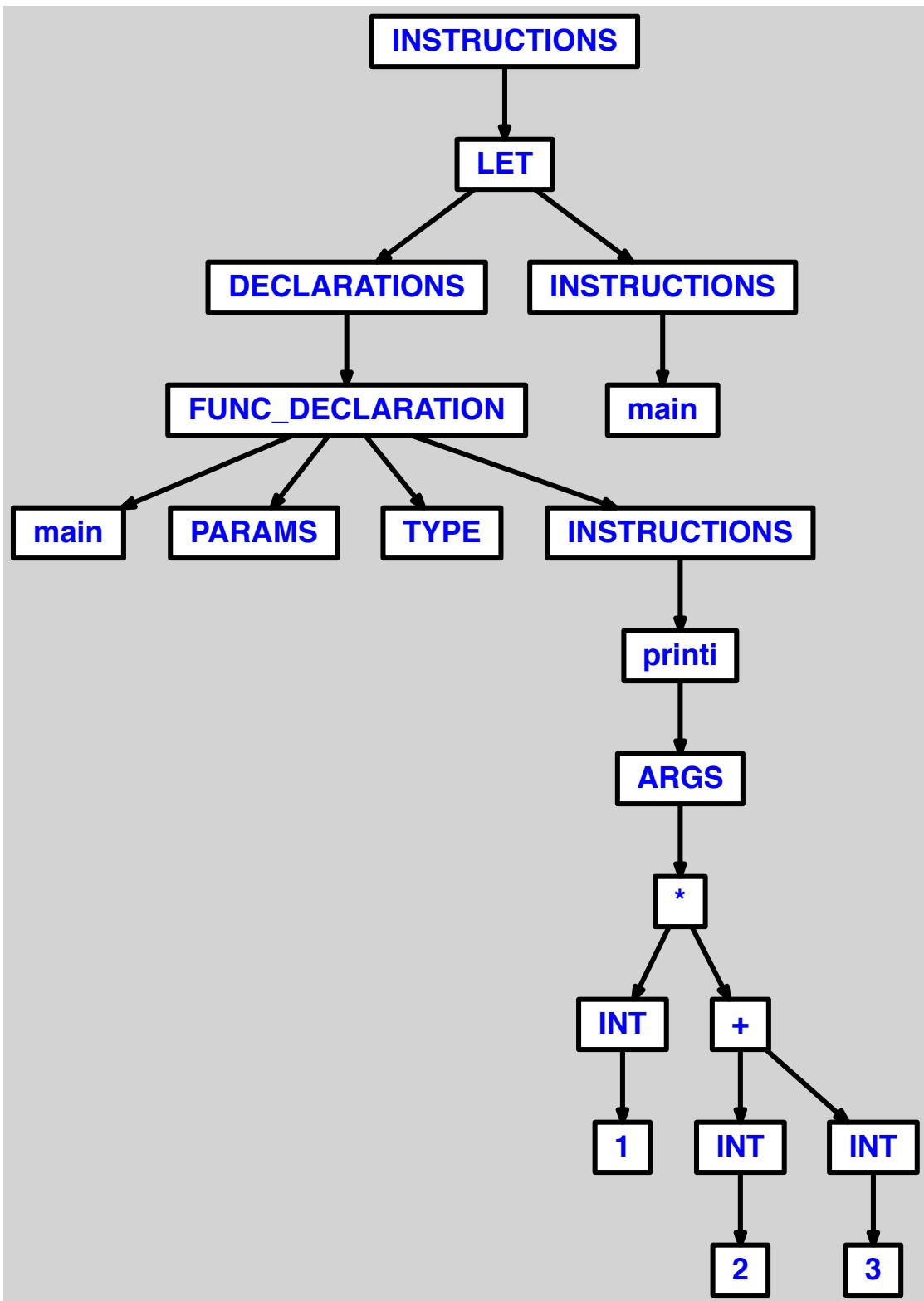
### 3.2.52 addition suivie de division, avec parenthesage des 2 termes à droite

```
let function main() = printi(1+(2/3)) in main() end
```



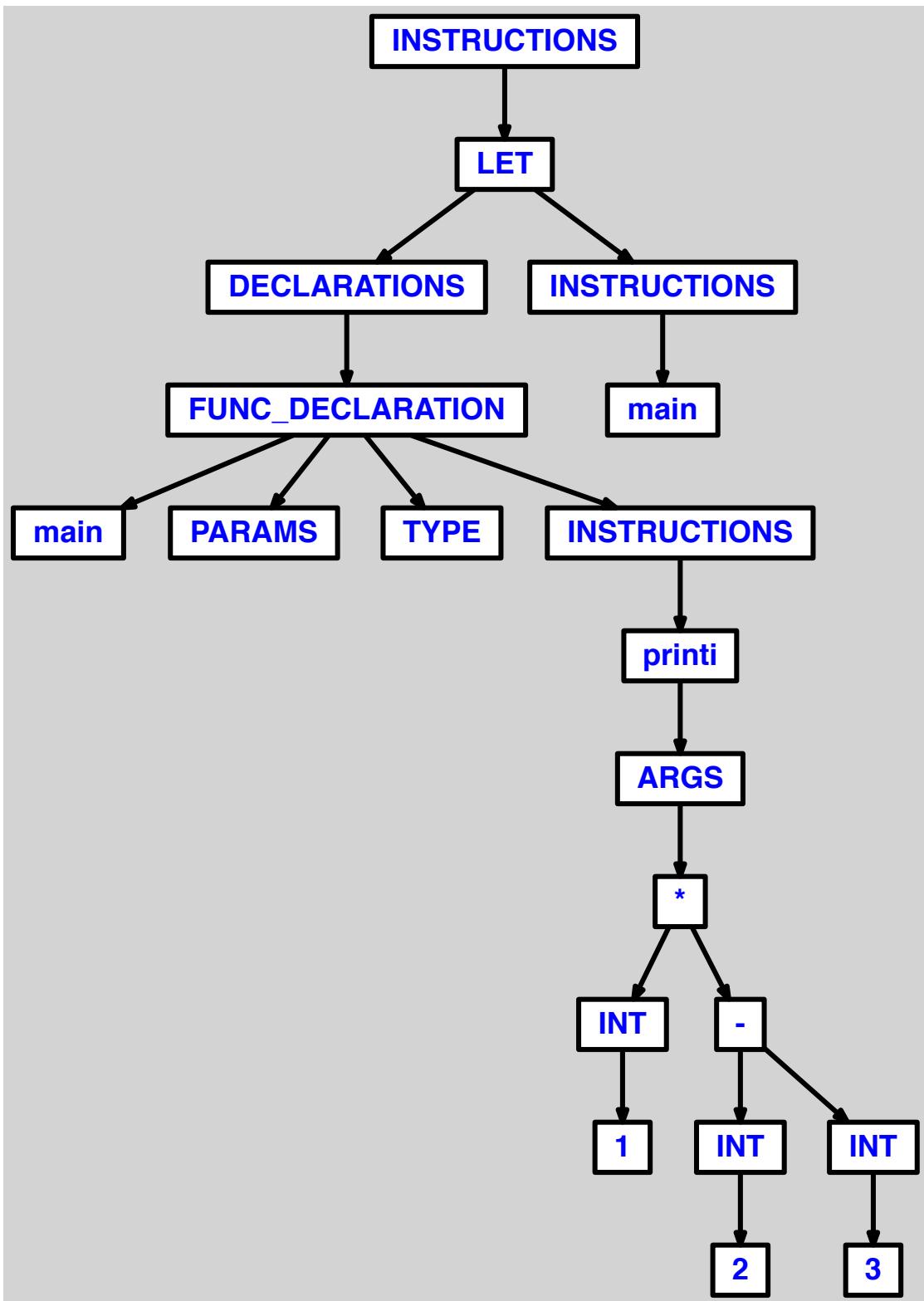
### 3.2.53 multiplication suivie d'addition, avec parenthesage des 2 termes à droite

```
let function main() = printi(1*(2+3)) in main() end
```



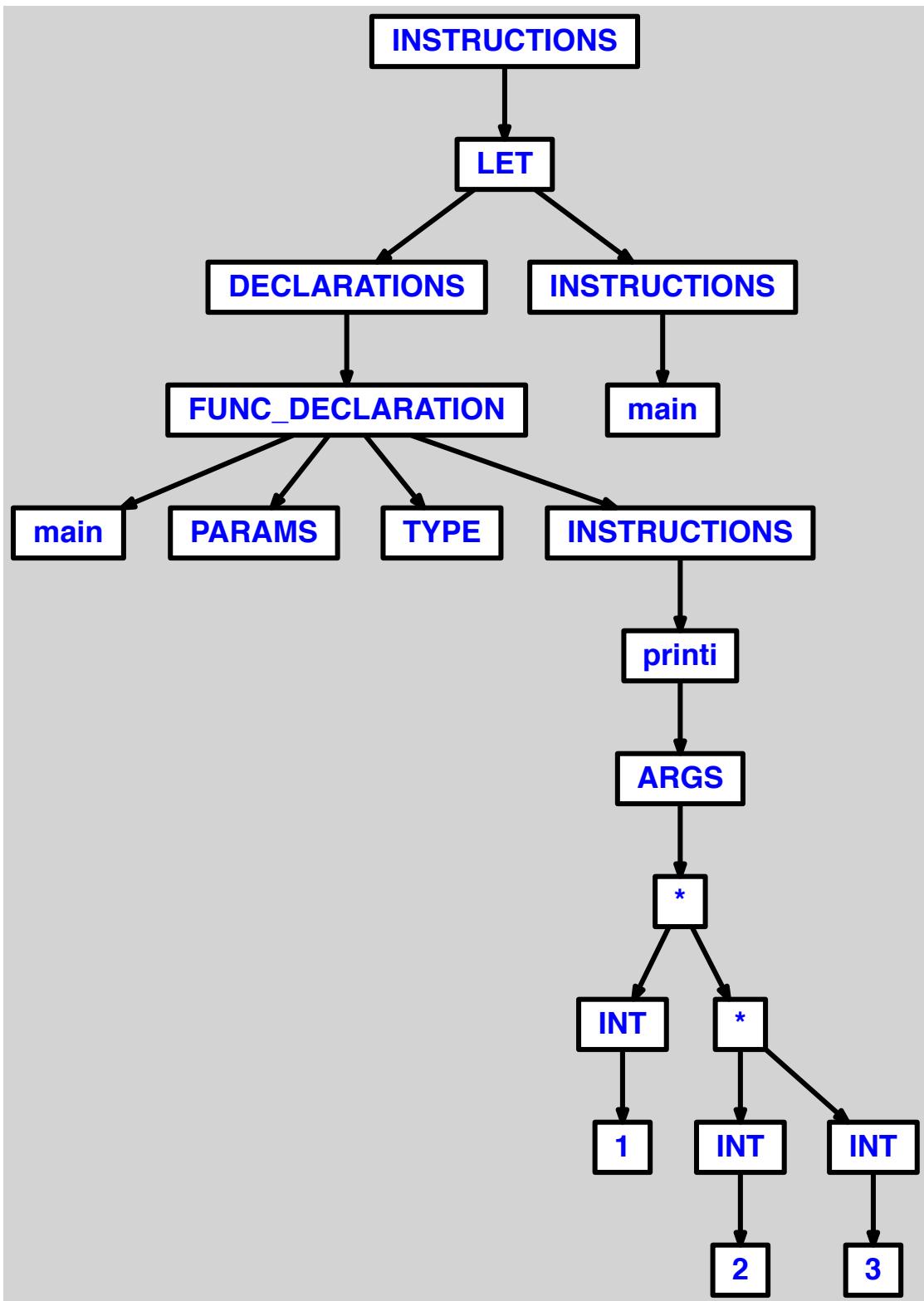
### 3.2.54 multiplication suivie de soustraction, avec parenthesage des 2 termes a droite

```
let function main() = printi(1*(2-3)) in main() end
```



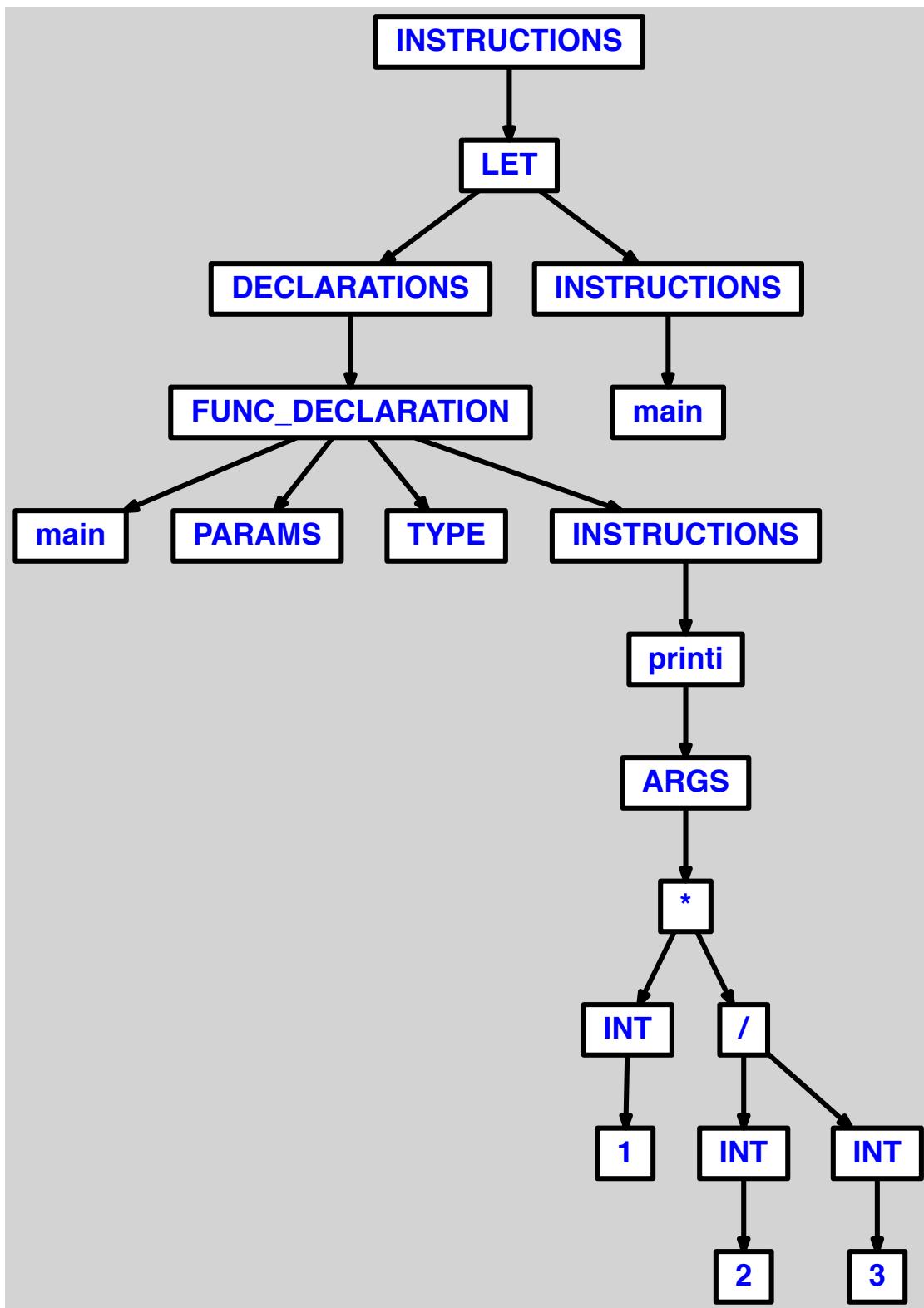
### 3.2.55 multiplication a 3 termes, avec parenthesage des 2 termes a droite

```
let function main() = printi(1*(2*3)) in main() end
```



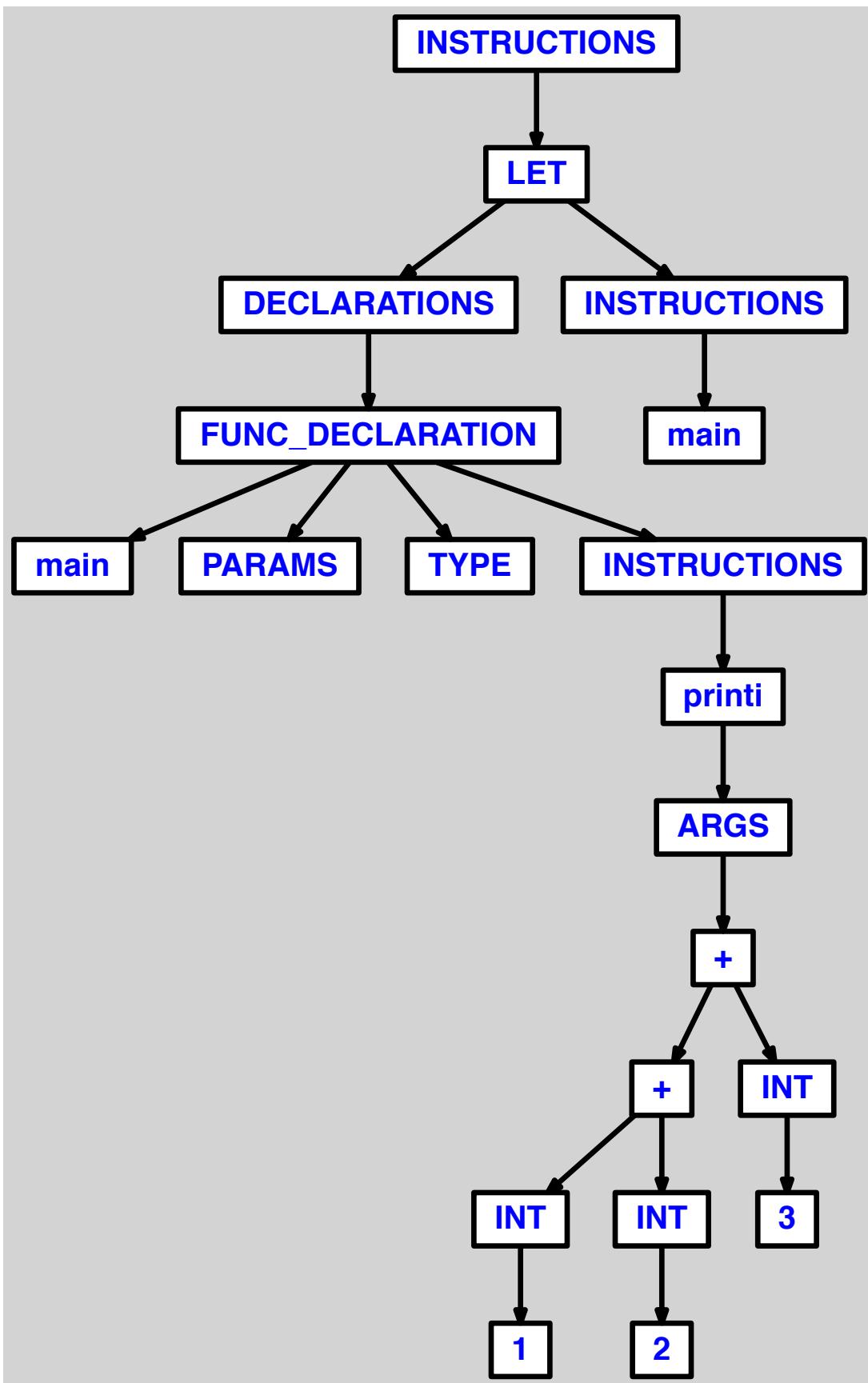
### 3.2.56 multiplication suivie de vision, avec parenthesage des 2 termes a droite

```
let function main() = printi(1*(2/3)) in main() end
```



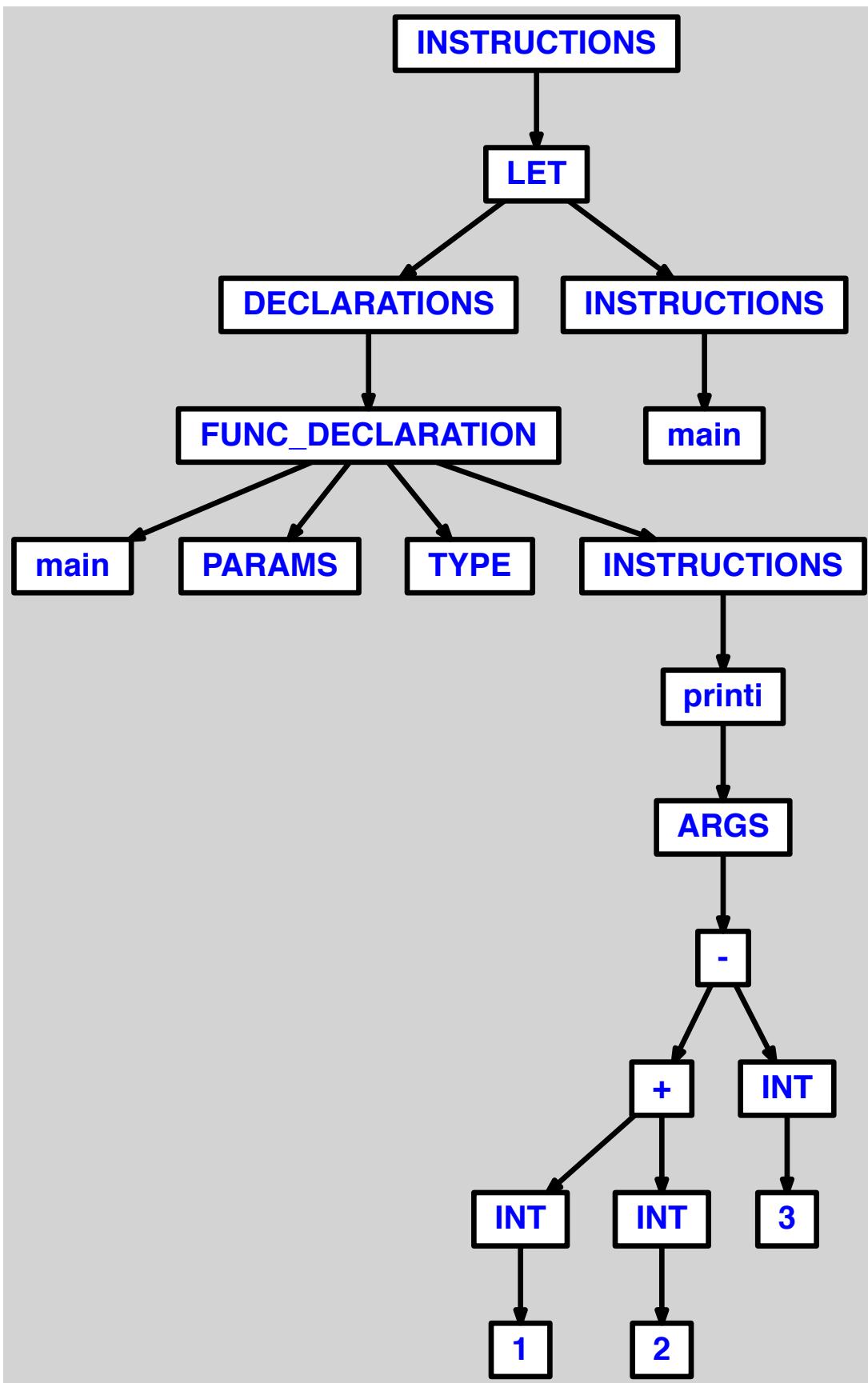
### 3.2.57 addition a 3 termes, avec parenthesage des 2 termes a gauche

```
let function main() = printi((1+2)+3) in main() end
```



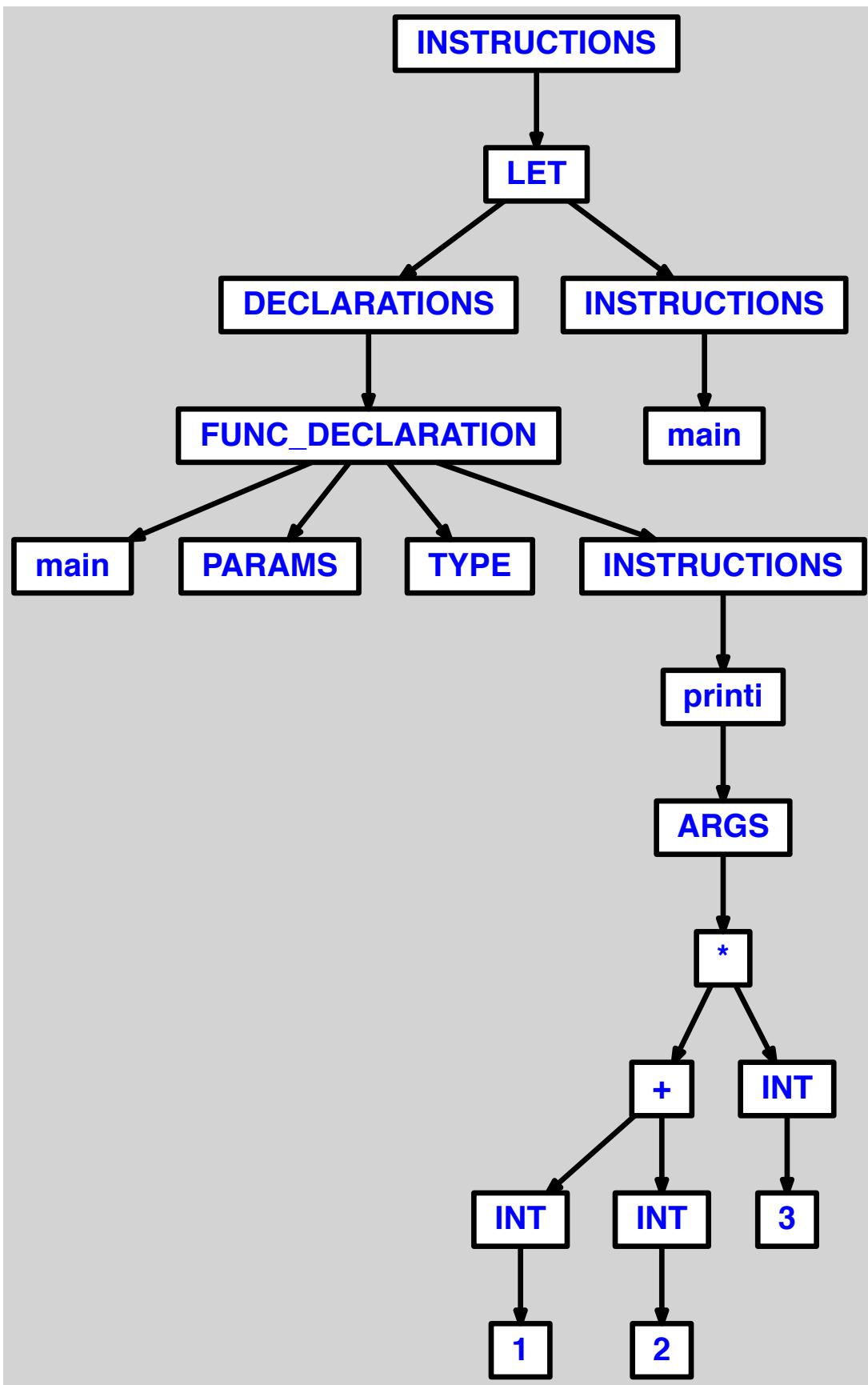
### 3.2.58 addition suivie de soustraction, avec parenthesage des 2 termes à gauche

```
let function main() = printi((1+2)-3) in main() end
```



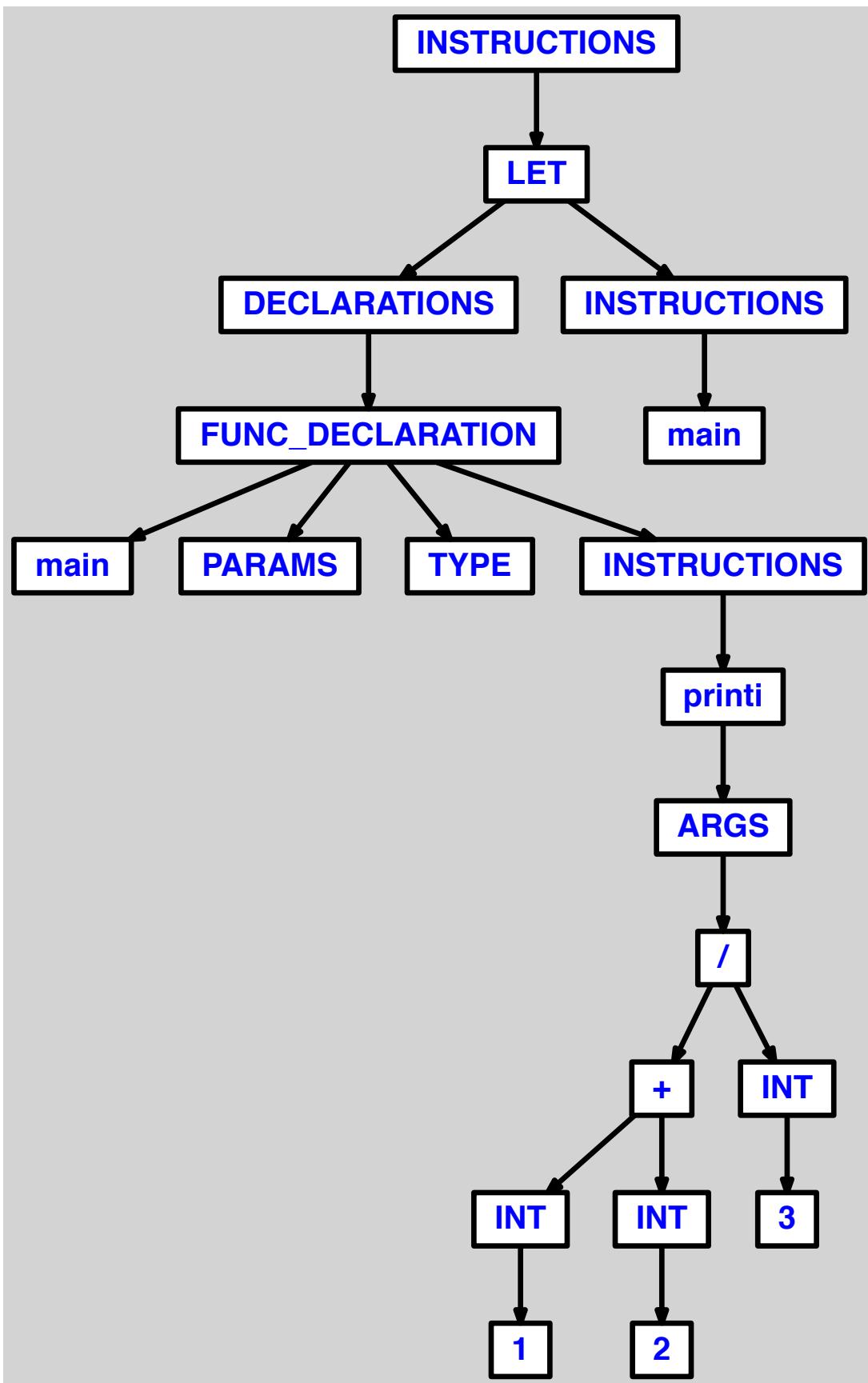
### 3.2.59 addition suivie de multiplication, avec parenthesage des 2 termes à gauche

```
let function main() = printi((1+2)*3) in main() end
```



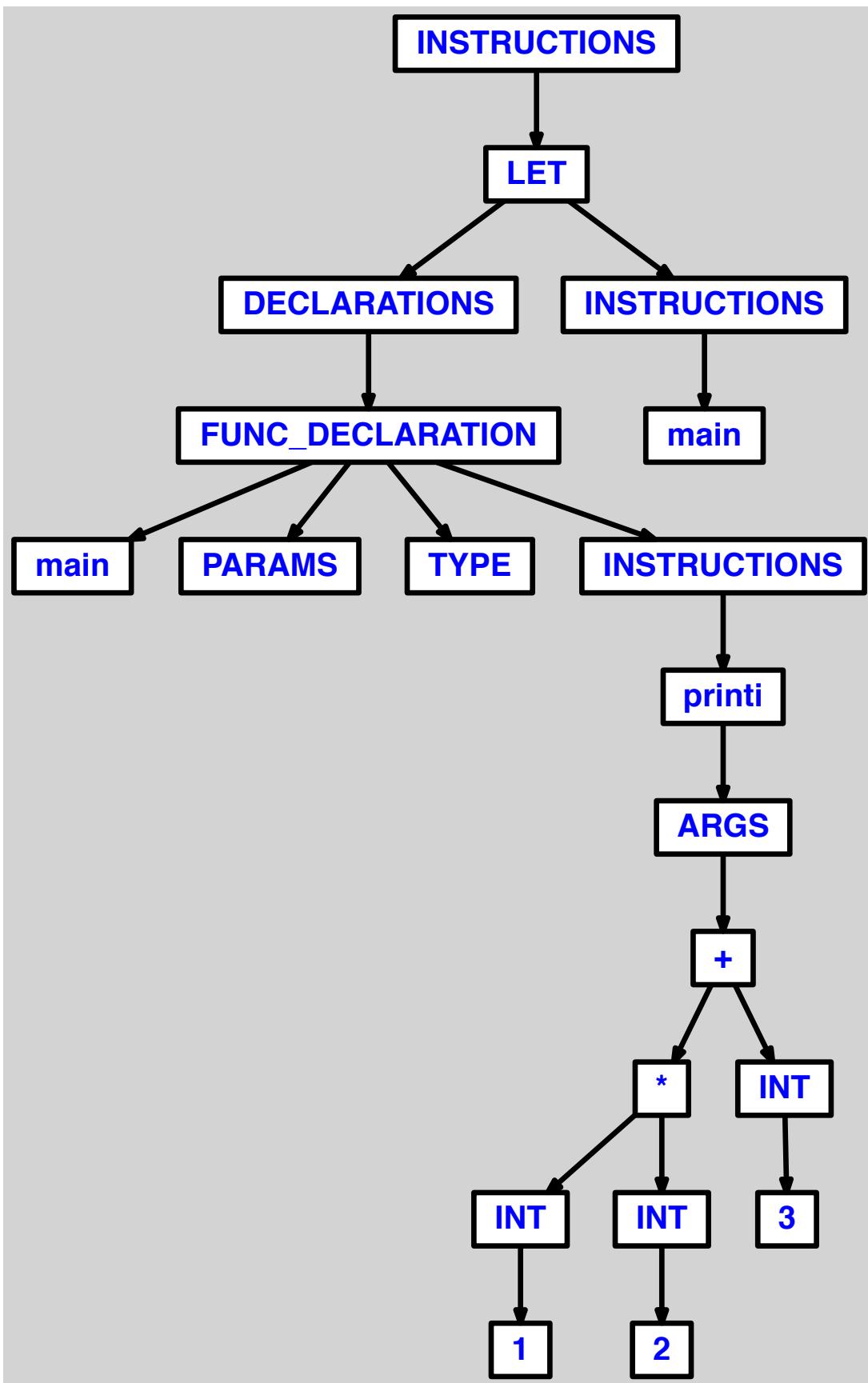
### 3.2.60 addition suivie de division, avec parenthesage des 2 termes a gauche

```
let function main() = printi((1+2)/3) in main() end
```



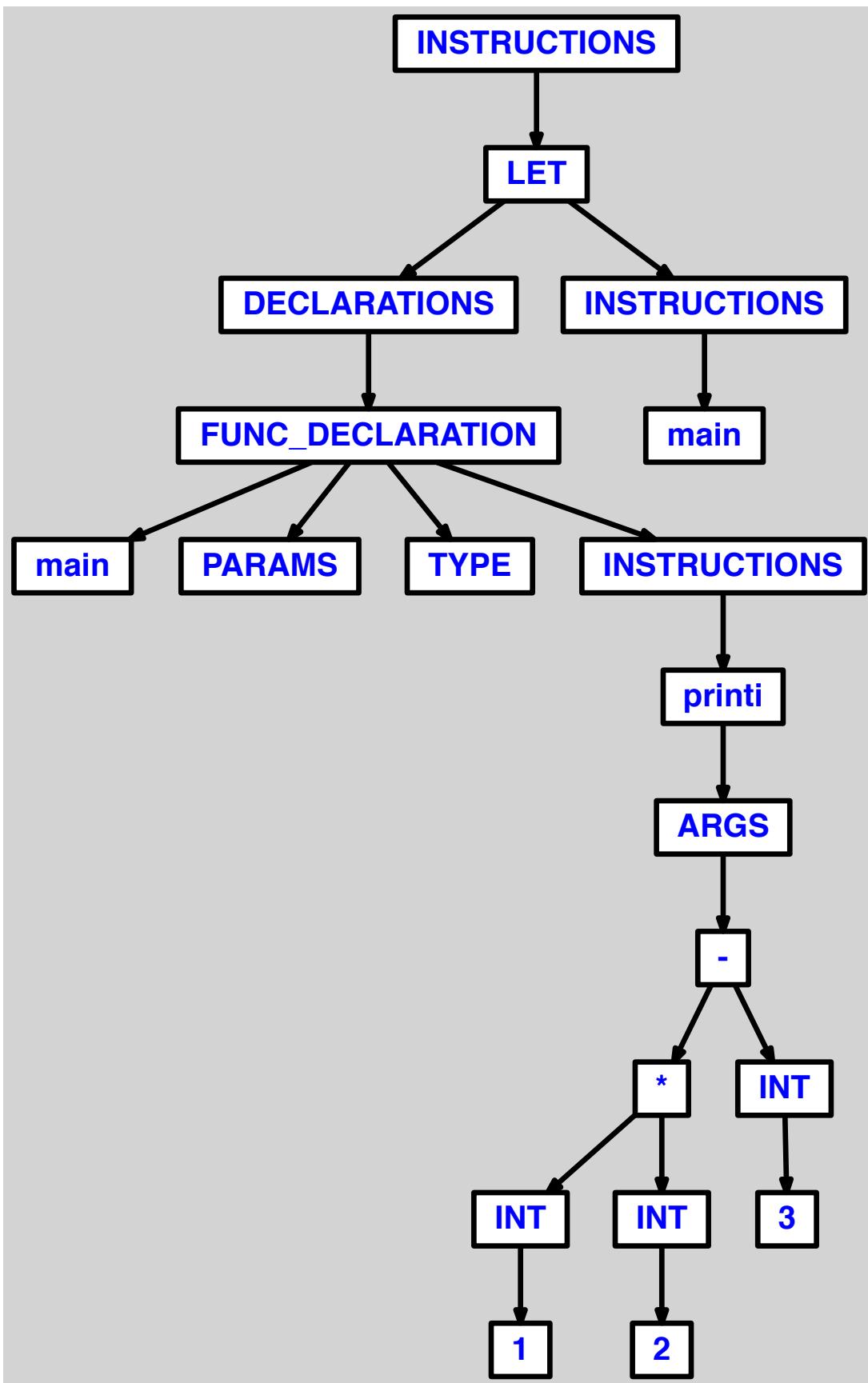
### 3.2.61 multiplication suivie d'addition, avec parenthesage des 2 termes à gauche

```
let function main() = printi((1*2)+3) in main() end
```



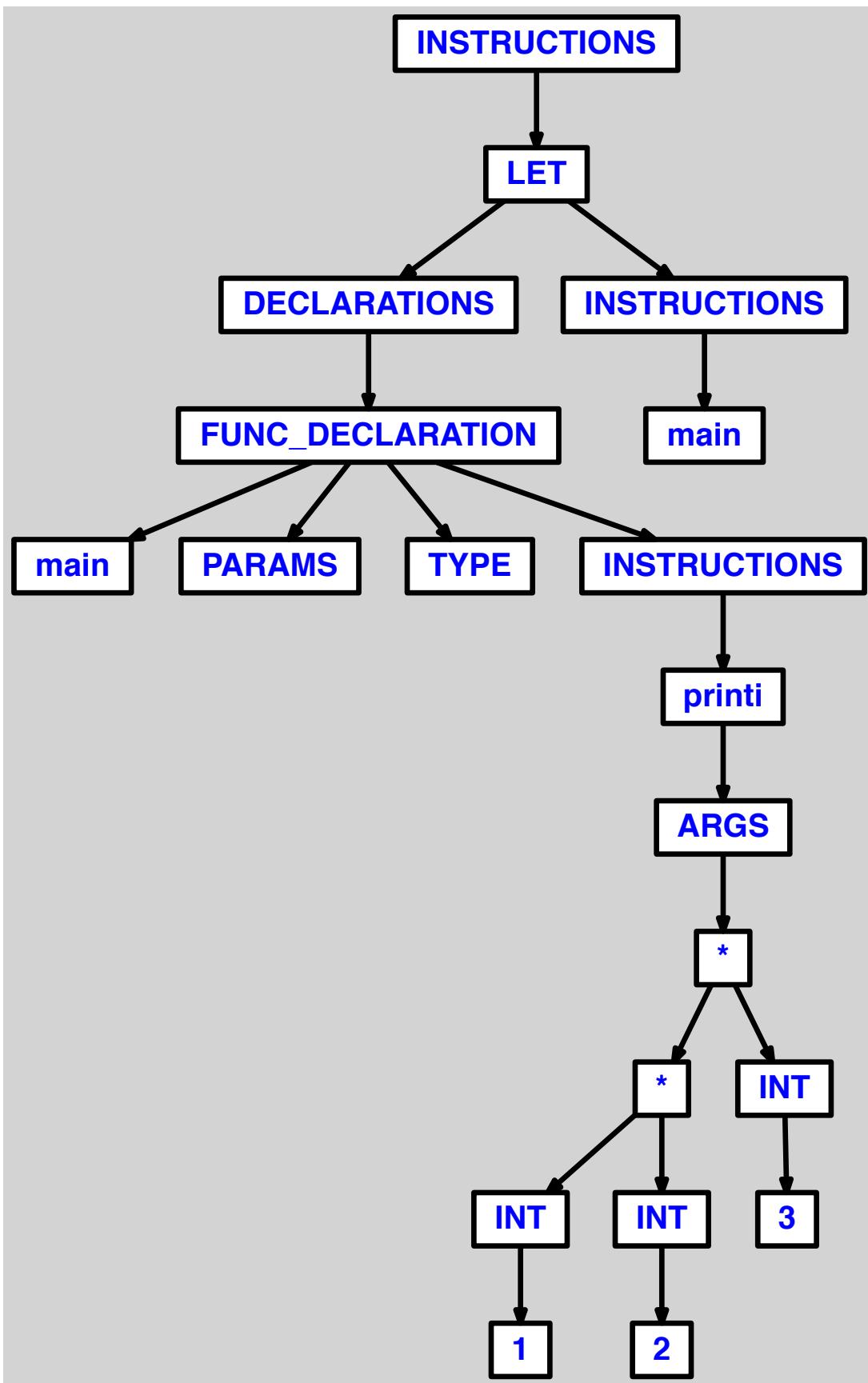
### 3.2.62 multiplication suivie de soustraction, avec parenthesage des 2 termes a gauche

```
let function main() = printi((1*2)-3) in main() end
```



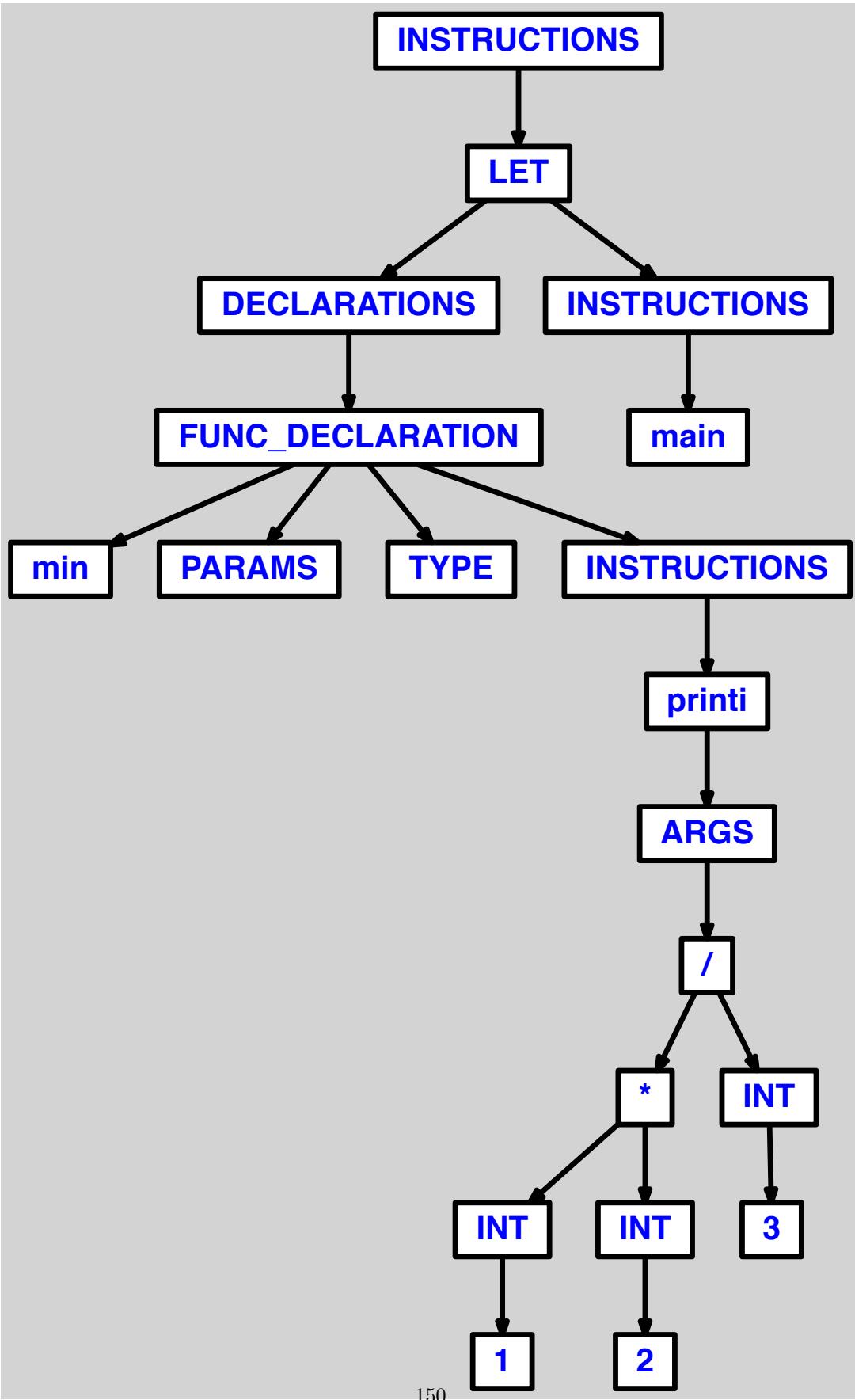
### 3.2.63 multiplication a 3 termes, avec parenthesage des 2 termes a gauche

```
let function main() = printi((1*2)*3) in main() end
```



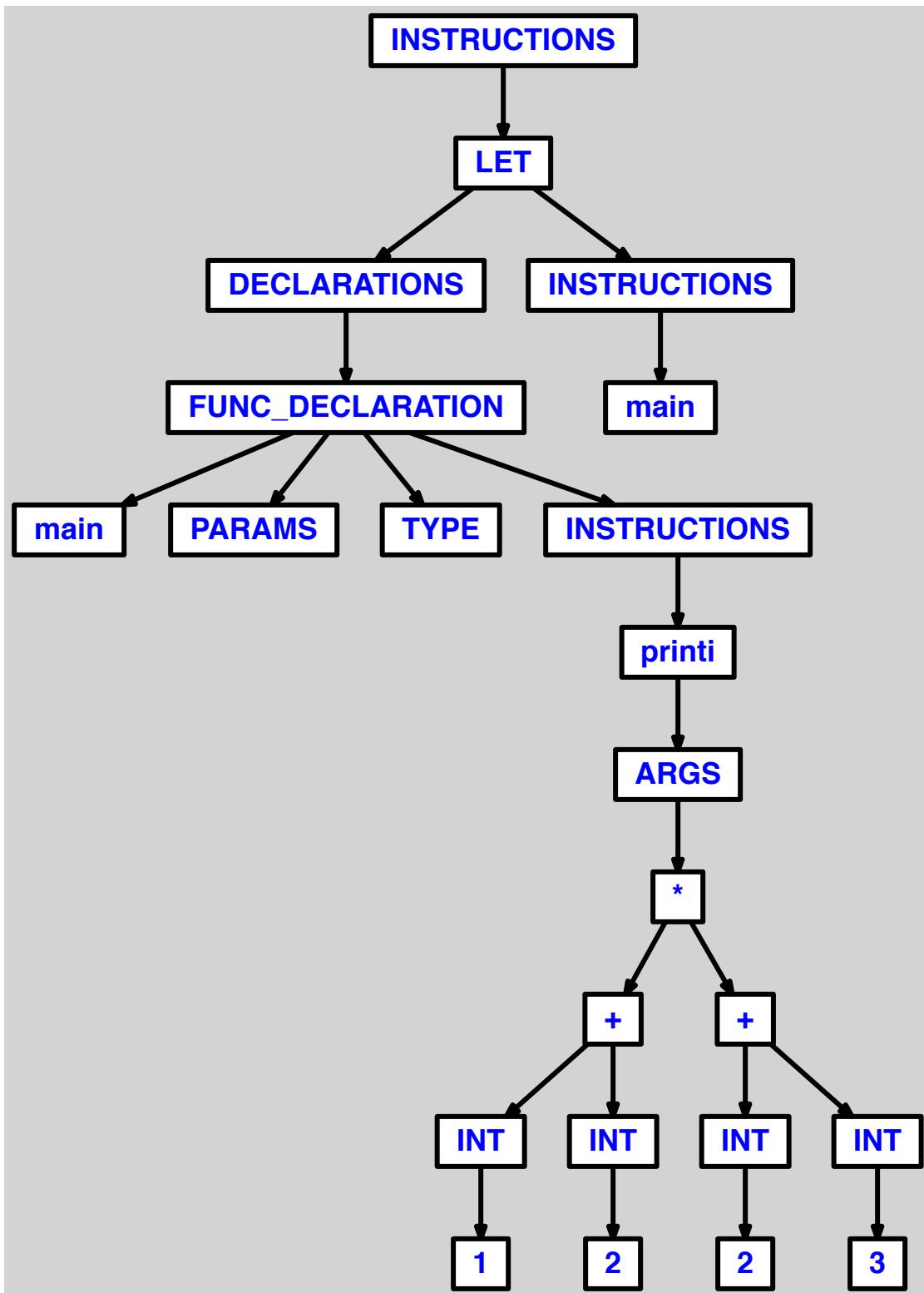
### 3.2.64 multiplication suivie de division, avec parenthesage des 2 termes a gauche

```
let function min() = printi((1*2)/3) in main() end
```



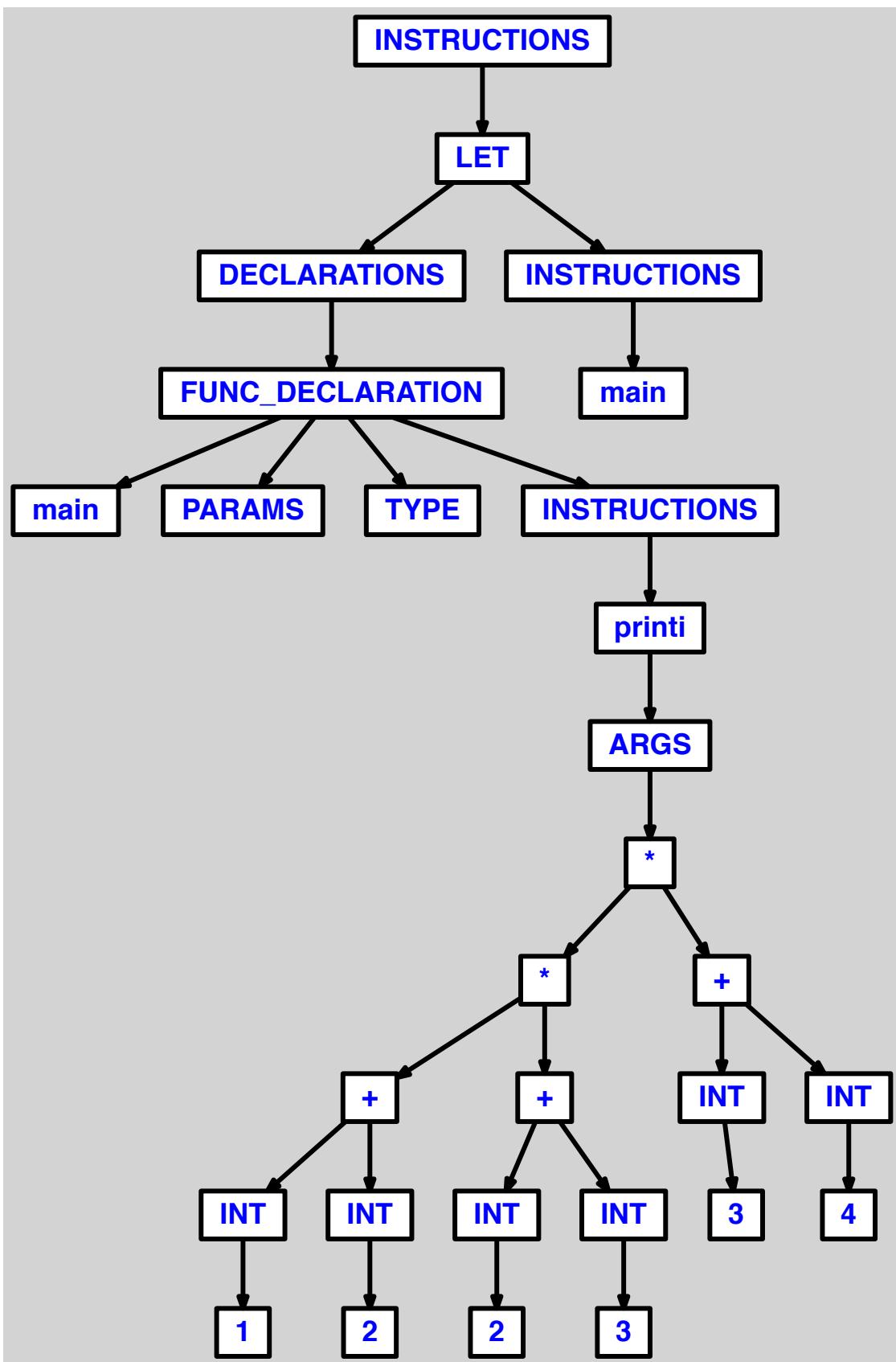
### 3.2.65 multiplication de 2 additions parenthesees

```
let function main() = printi((1+2)*(2+3)) in main() end */
```



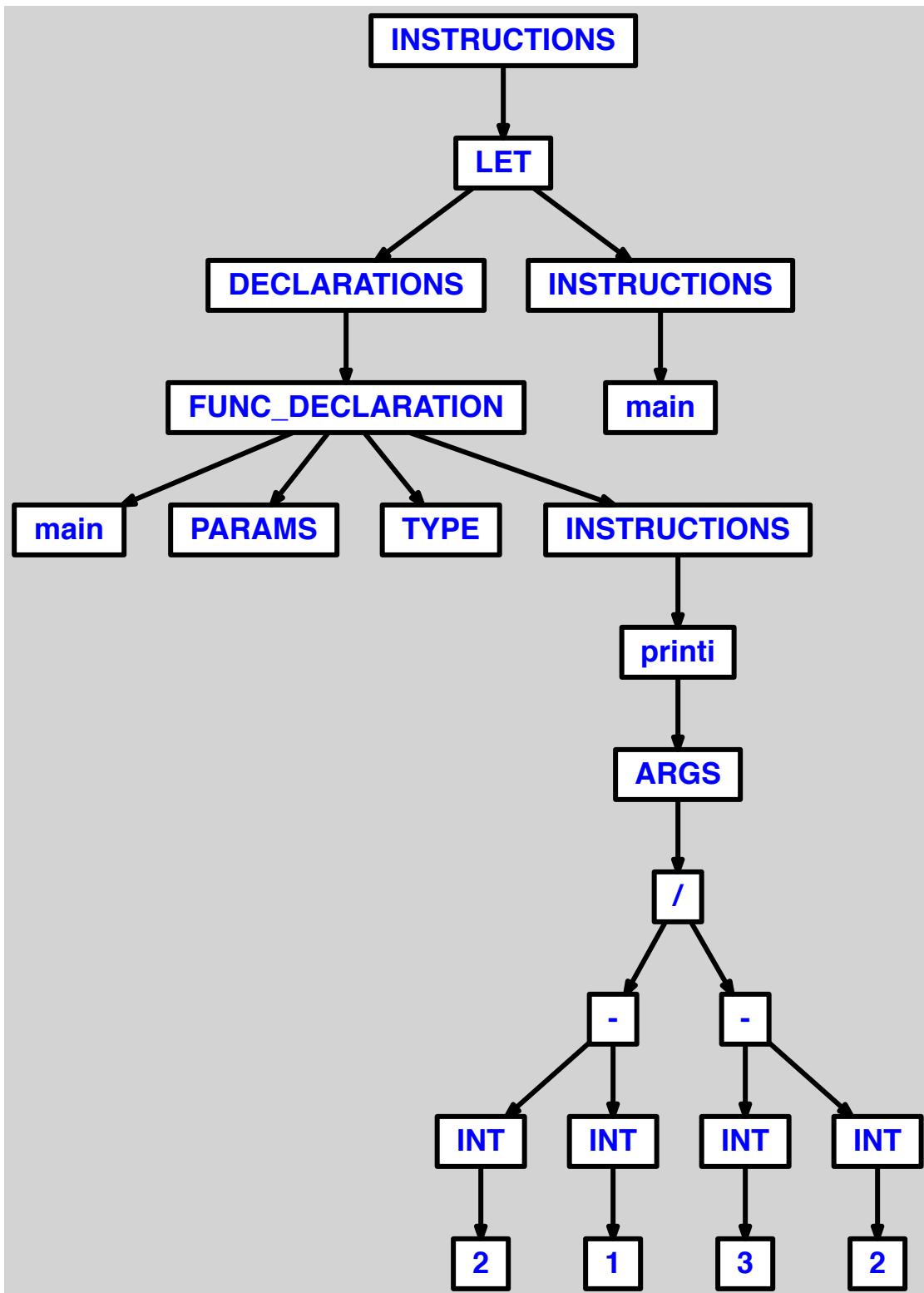
### 3.2.66 multiplication de 3 additions parenthesees

```
let function main() = printi((1+2)*(2+3)*(3+4)) in main() end
```



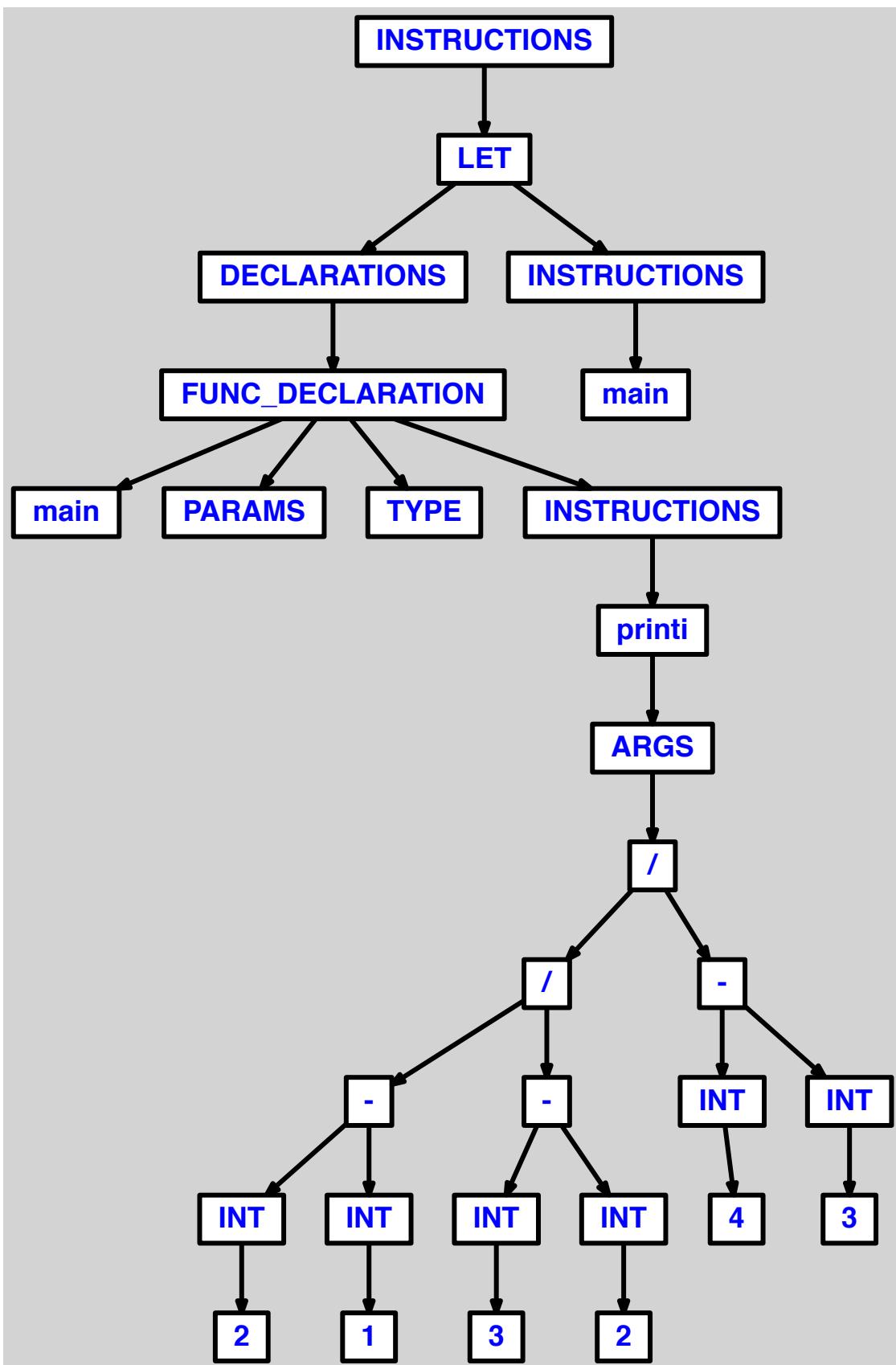
### 3.2.67 division de 2 soustractions parenthesees

```
let function main() = printi((2-1)/(3-2)) in main() end
```



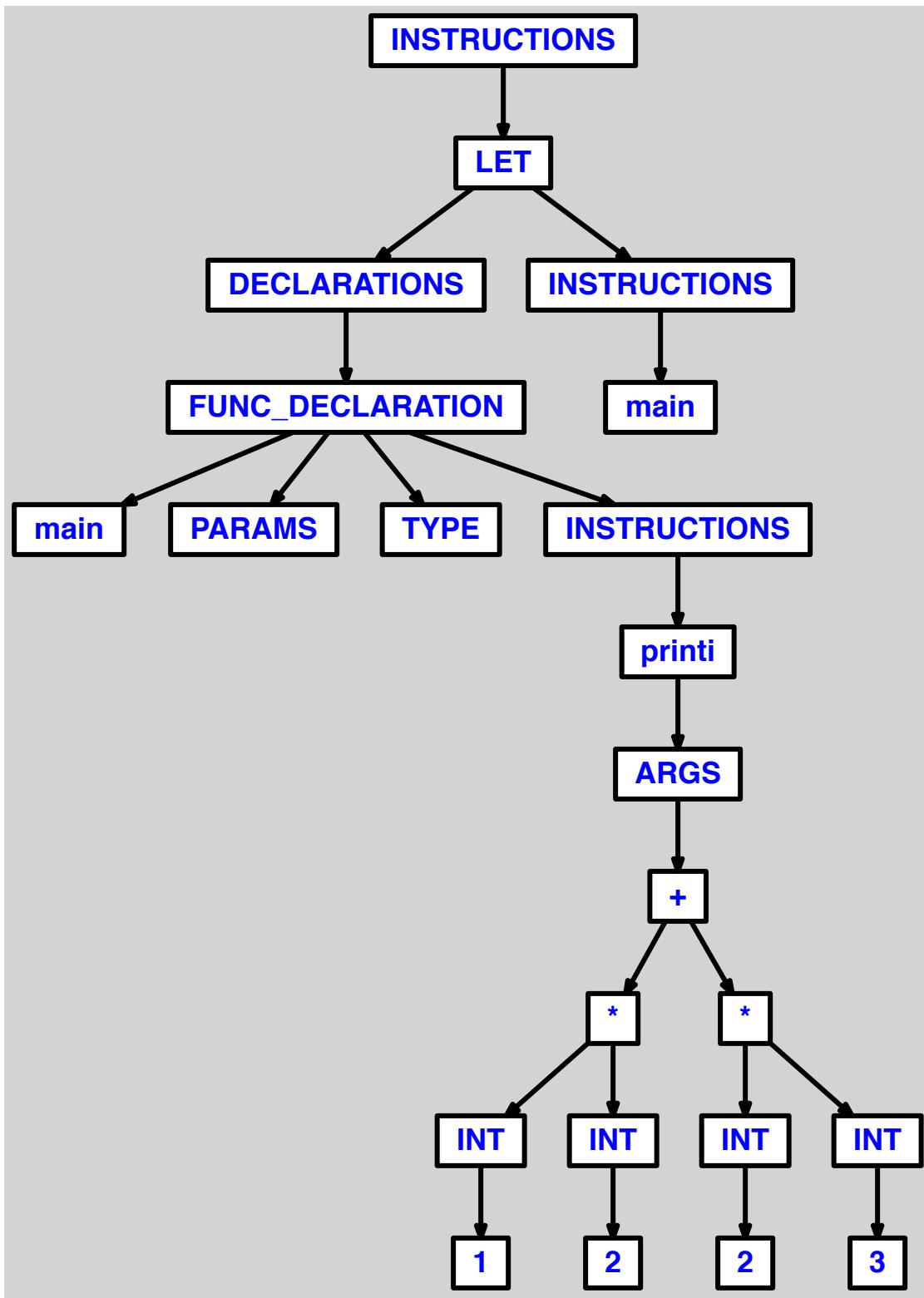
### 3.2.68 division de 3 soustractions parenthesees

```
let function main() = printi((2-1)/(3-2)/(4-3)) in main() end
```



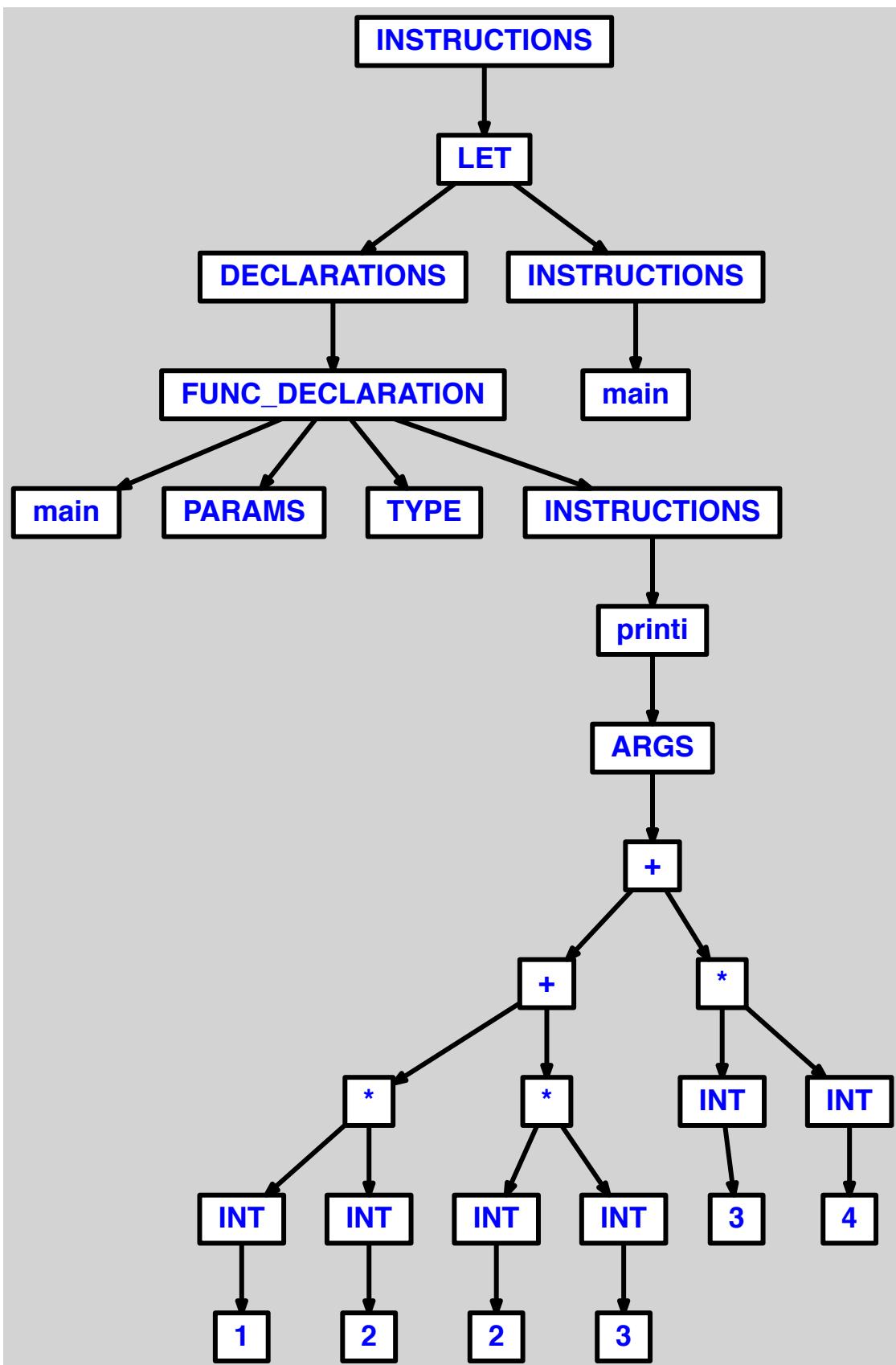
### 3.2.69 addition de 2 multiplications parenthesees

```
let function main() = printi((1*2)+(2*3)) in main() end
```



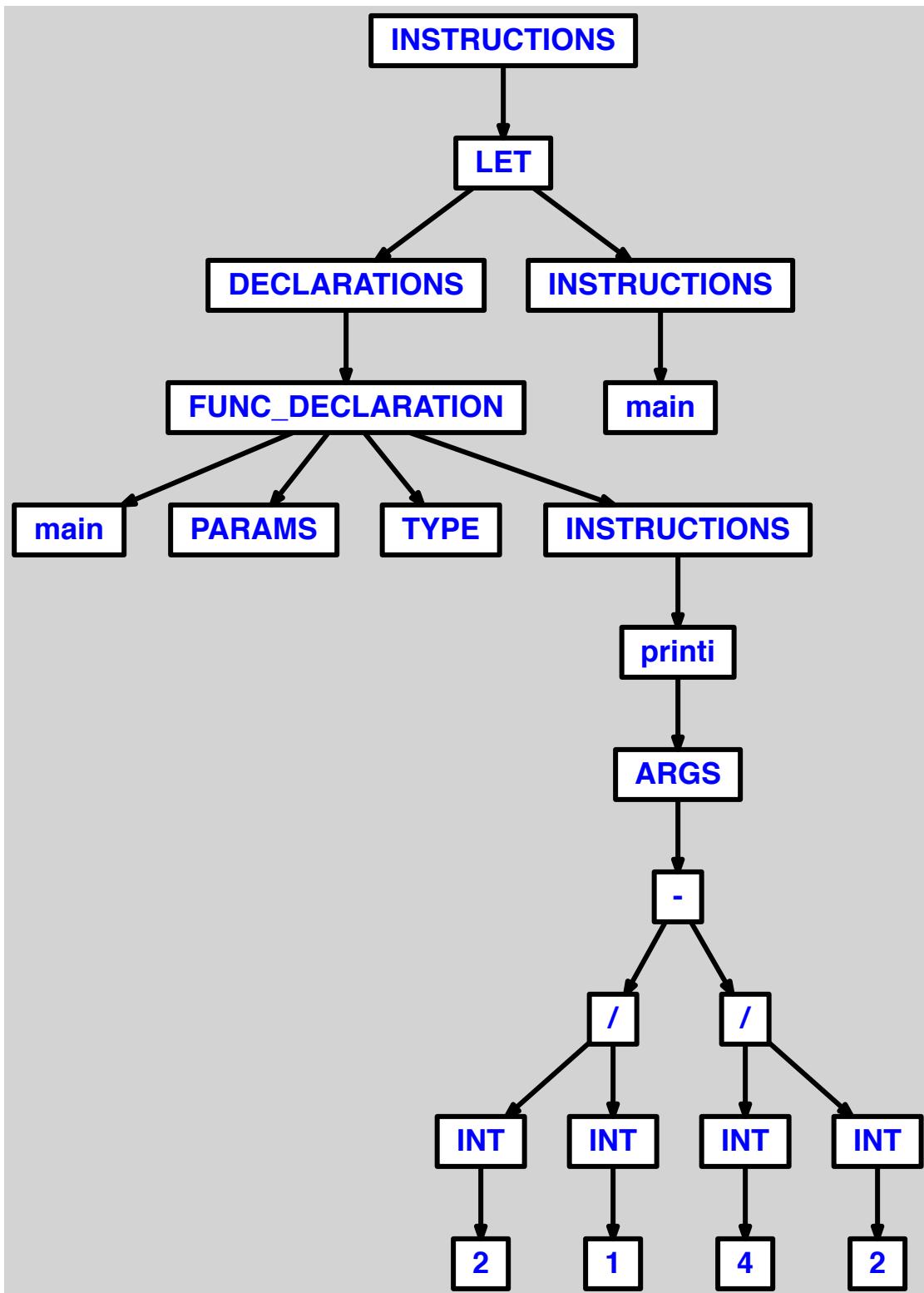
### 3.2.70 addition de 3 multiplications parenthesees

```
let function main() = printi((1*2)+(2*3)+(3*4)) in main() end
```



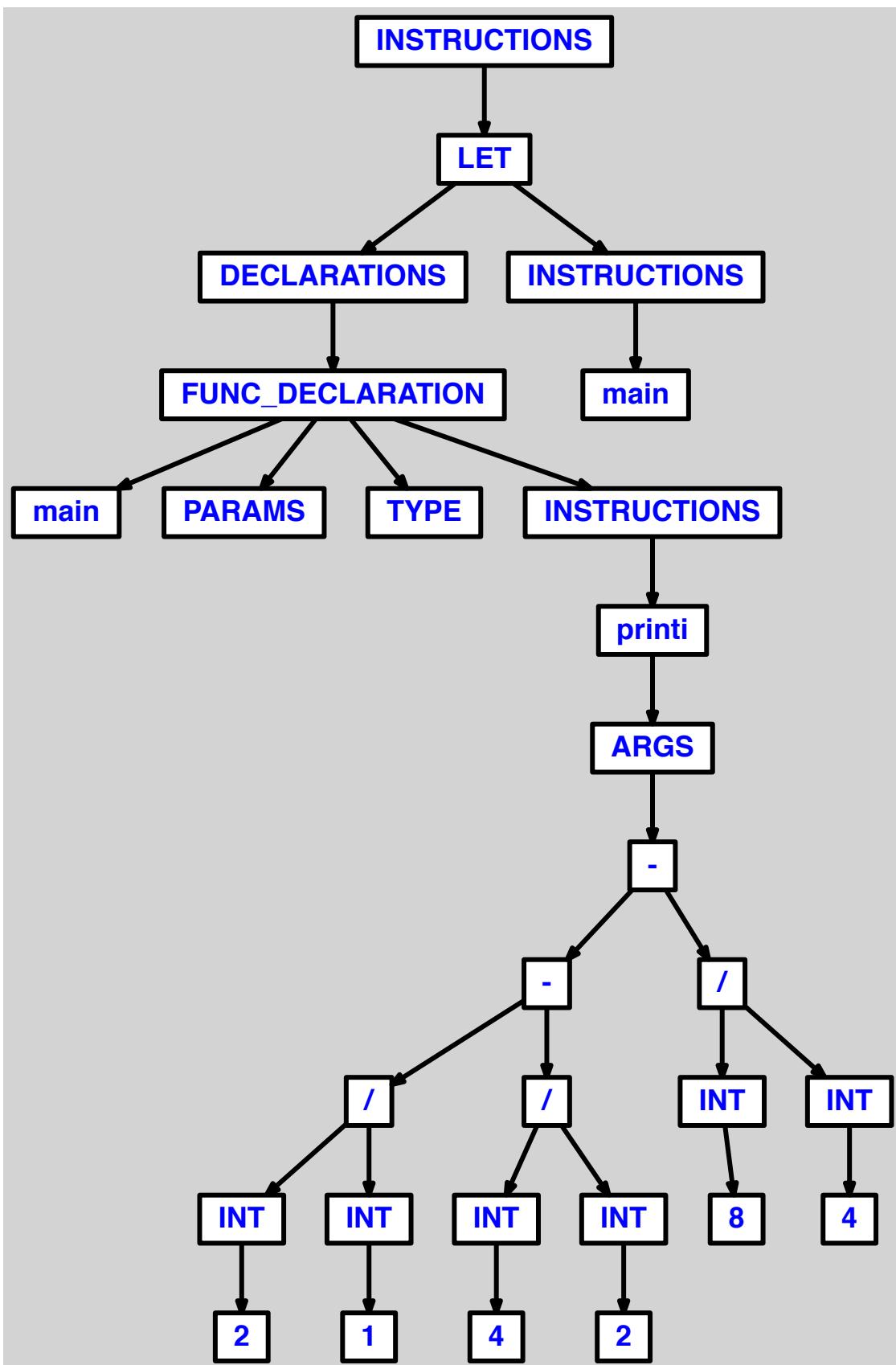
### 3.2.71 soustraction de 2 divisions parenthesees

```
let function main() = printi((2/1)-(4/2) in main() end
```



### 3.2.72 soustraction de 3 divisions parenthesees

```
let function main() = printi((2/1)-(4/2)-(8/4)) in main() end
```

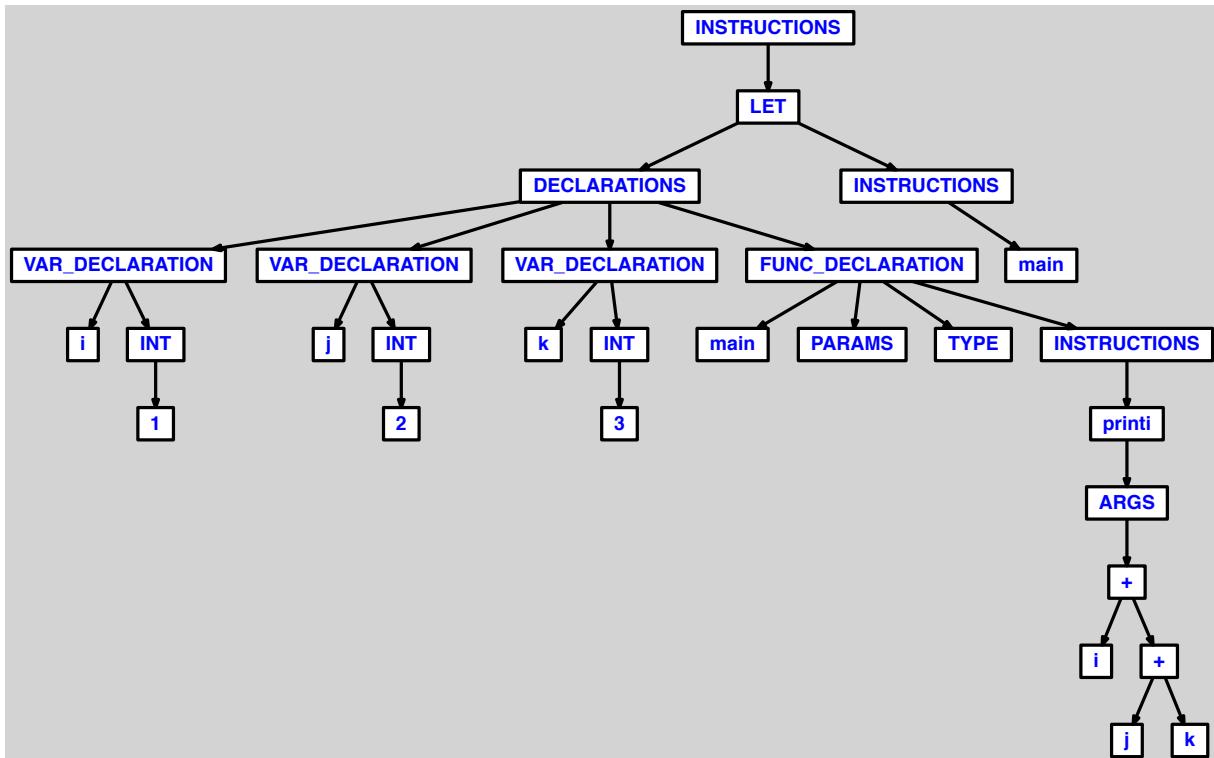


### 3.2.73 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

let

```
var i := 1
var j := 2
var k := 3
```

```
function main() = printi(i+(j+k))
in main() end
```

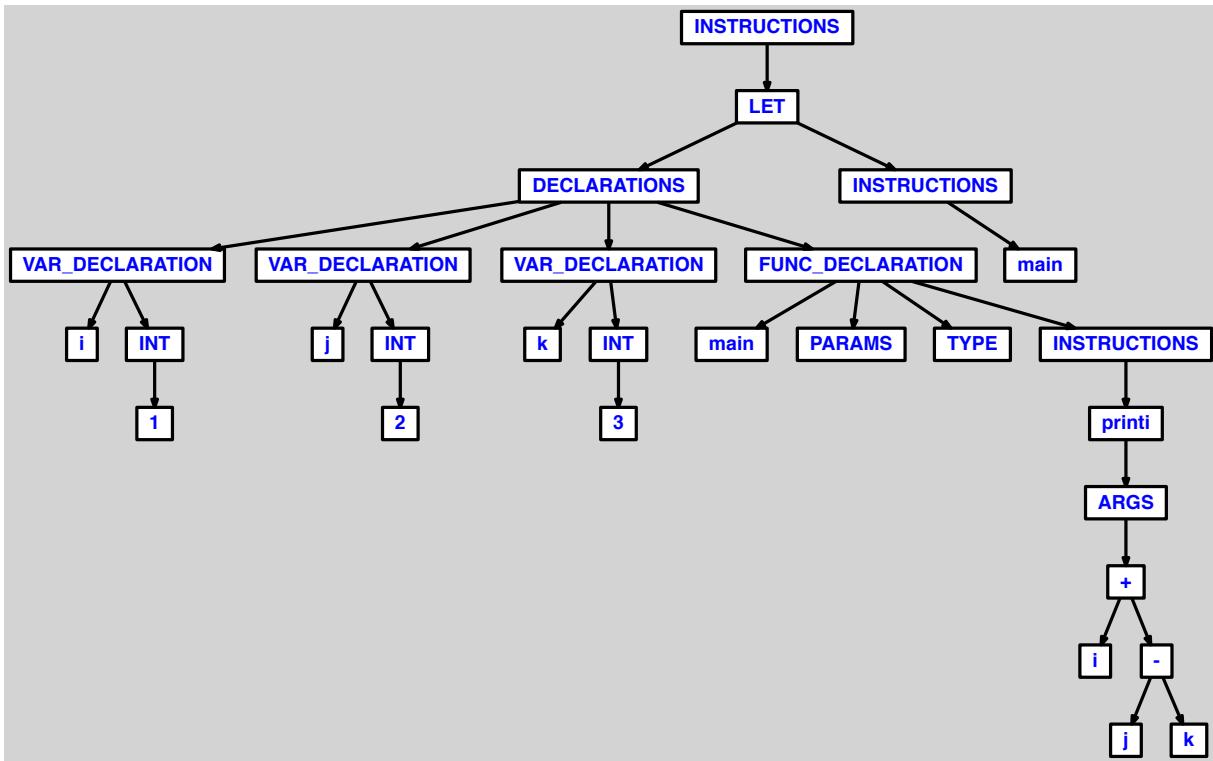


### 3.2.74 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite

let

```
var i := 1
var j := 2
var k := 3
```

```
function main() = printi(i+(j-k))
in main() end
```

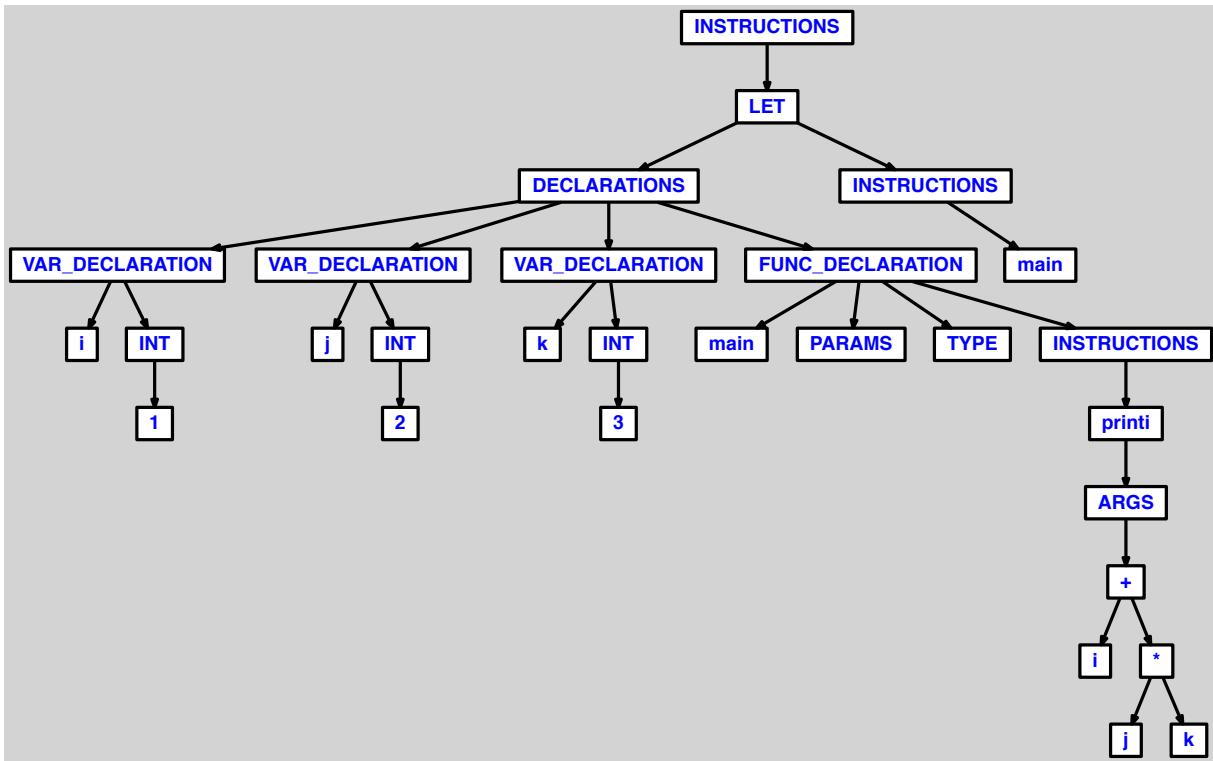


3.2.75 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à droite

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i+(j*k))
in main() end
  
```



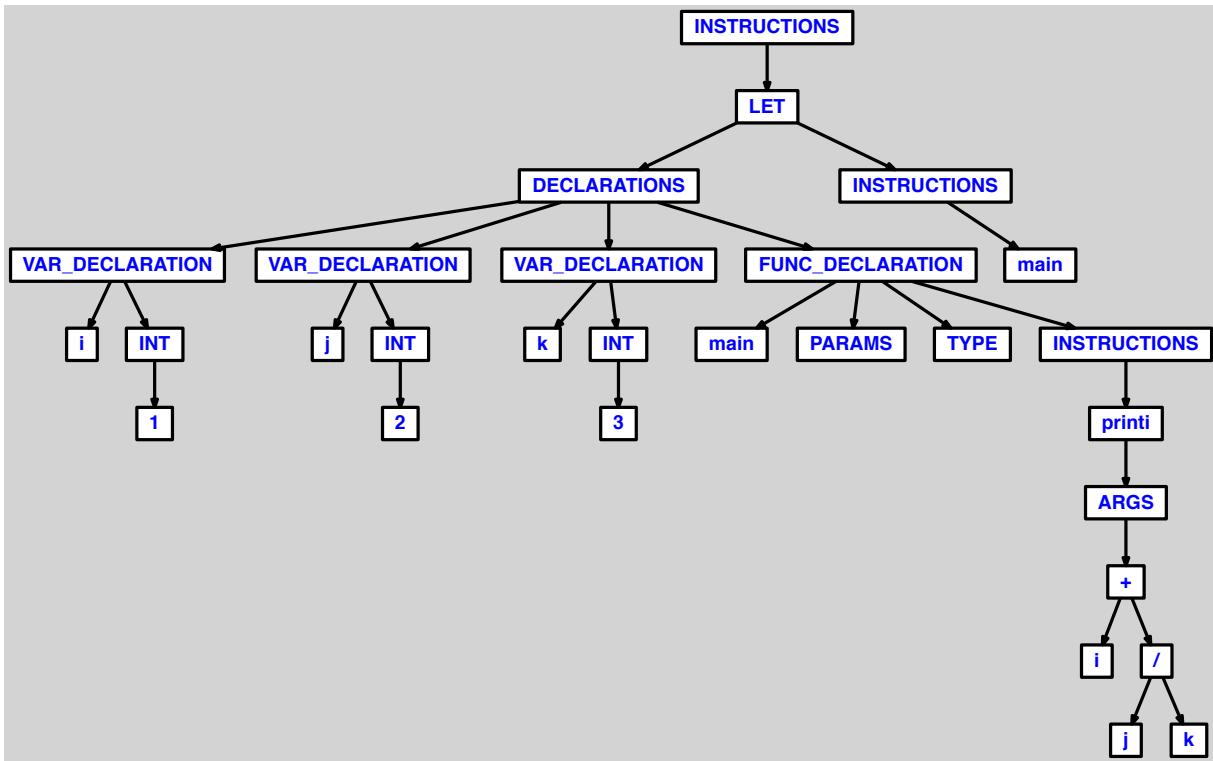
### 3.2.76 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i+(j/k))
in main() end

```



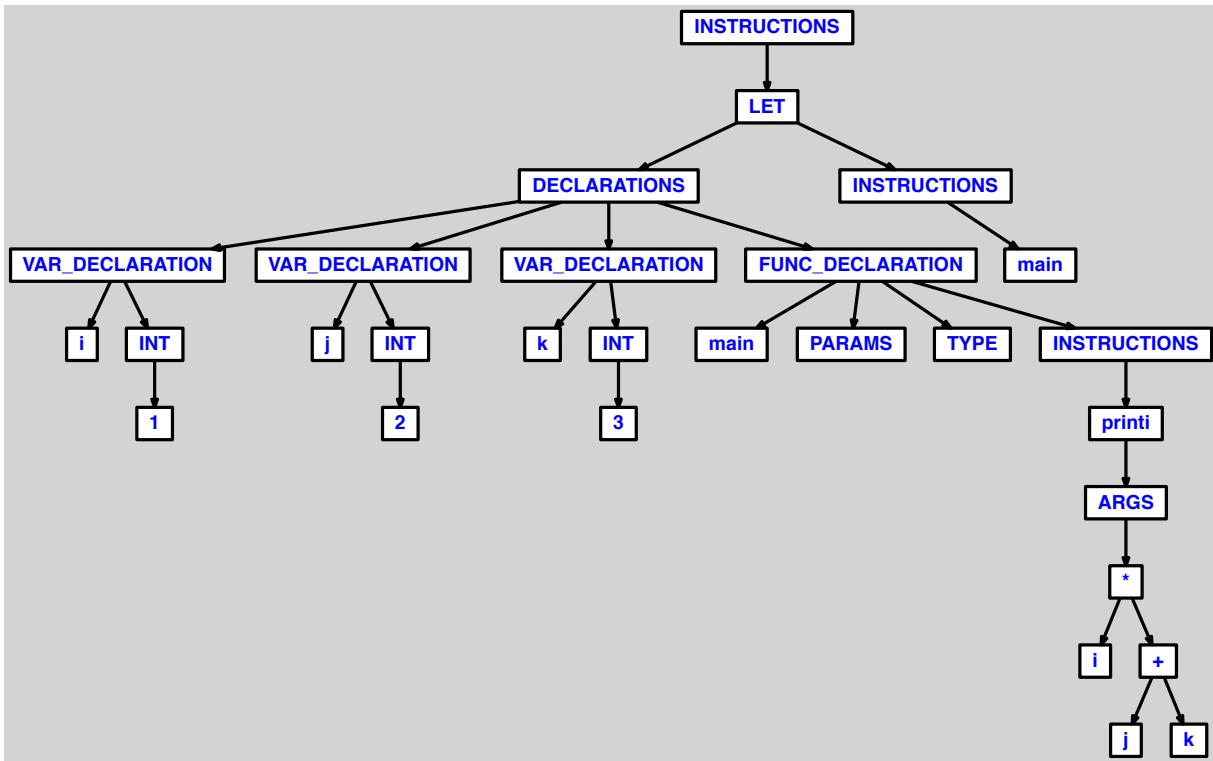
### 3.2.77 multiplication suivie d'addition, avec termes identifiés par variables et parenthesage des 2 variables à droite

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*(j+k))
in main() end

```

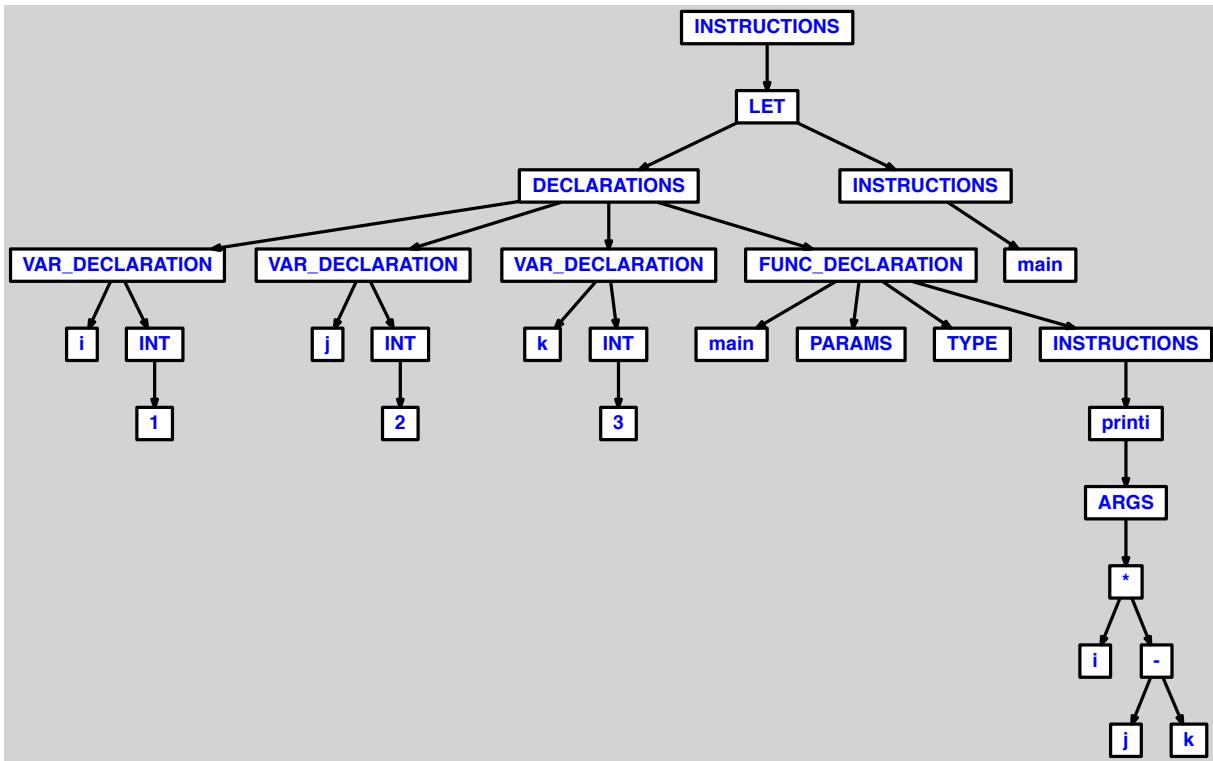


3.2.78 multiplication suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à droite

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*(j-k))
in main() end
  
```

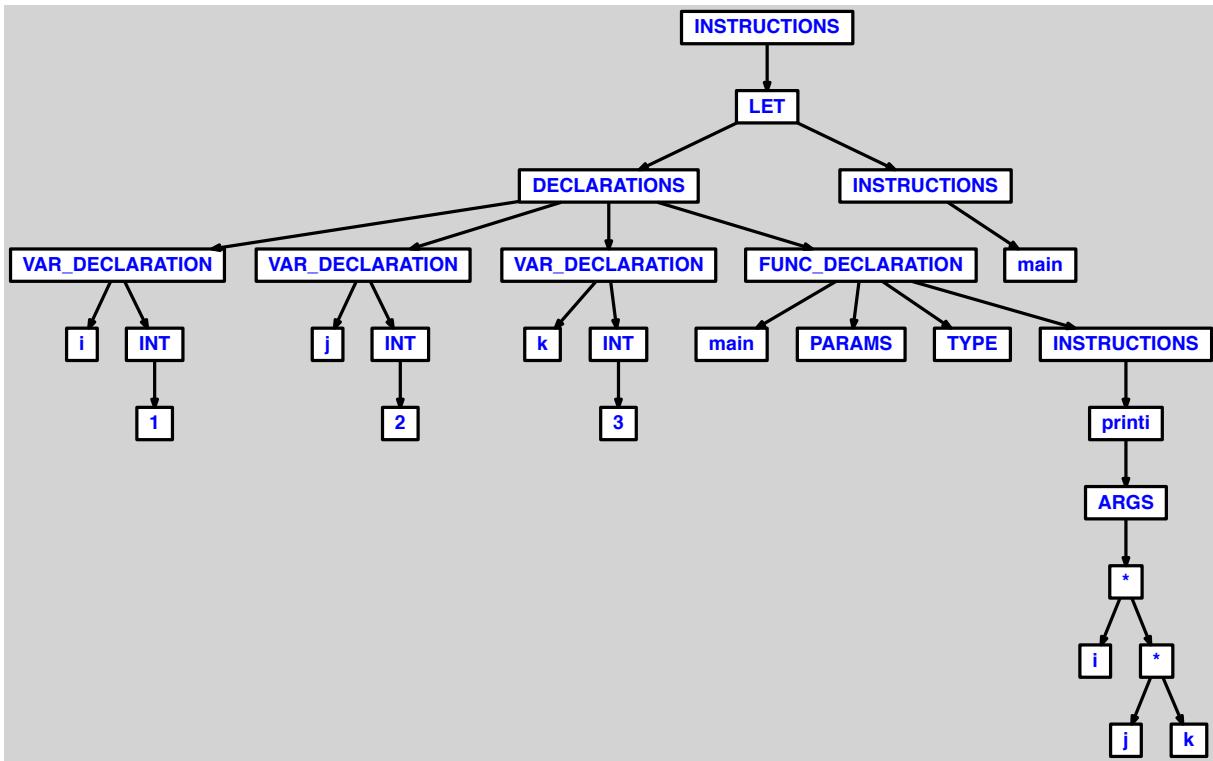


### 3.2.79 multiplication a 3 termes identifiés par variables, avec parenthesage des 2 variables à droite

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*(j*k))
in main() end
  
```



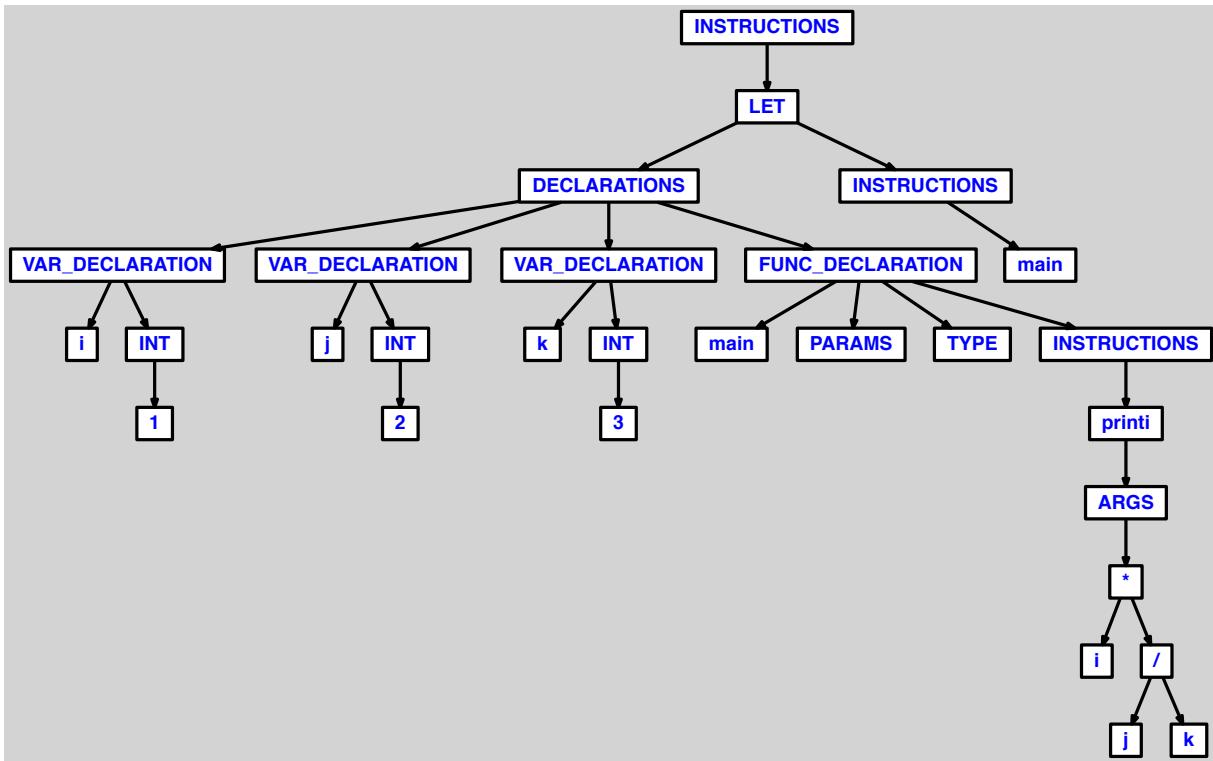
### 3.2.80 multiplication suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi(i*(j/k))
in main() end

```

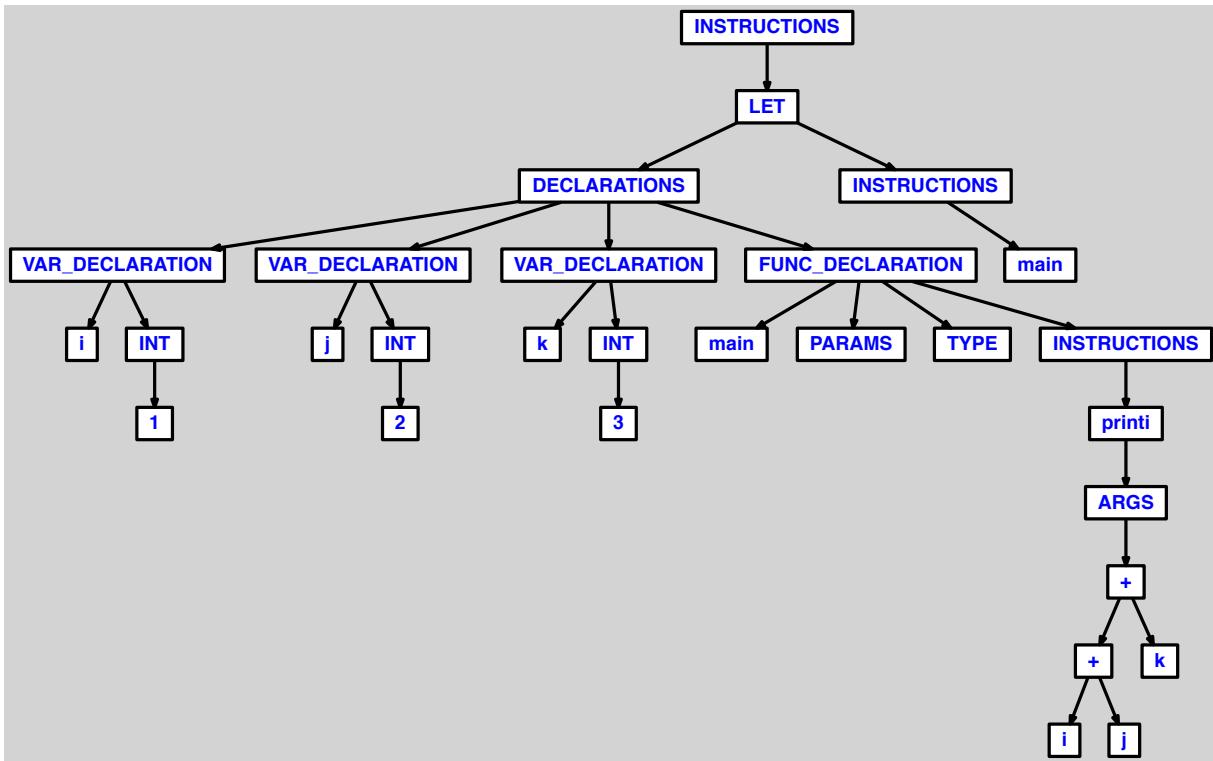


### 3.2.81 addition a 3 termes identifiés par variables, avec parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i+j)+k)
in main() end
  
```



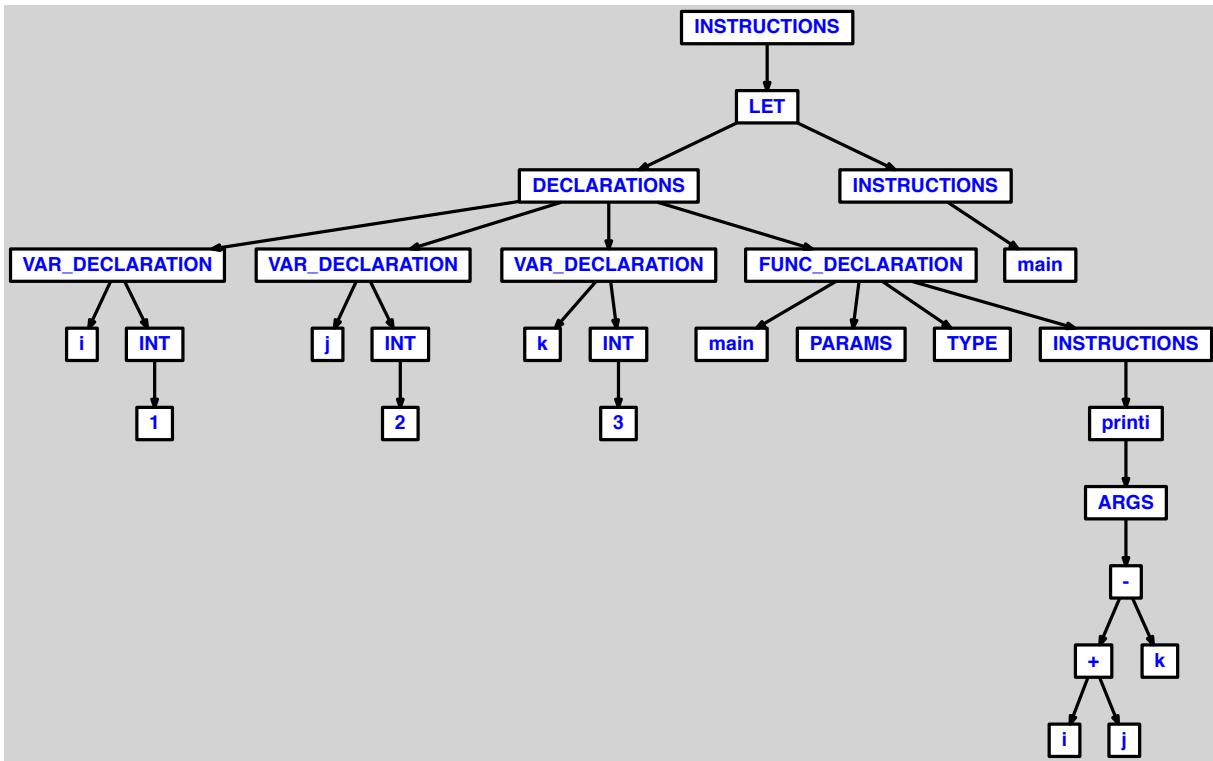
### 3.2.82 addition suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i+j)-k)
in main() end

```



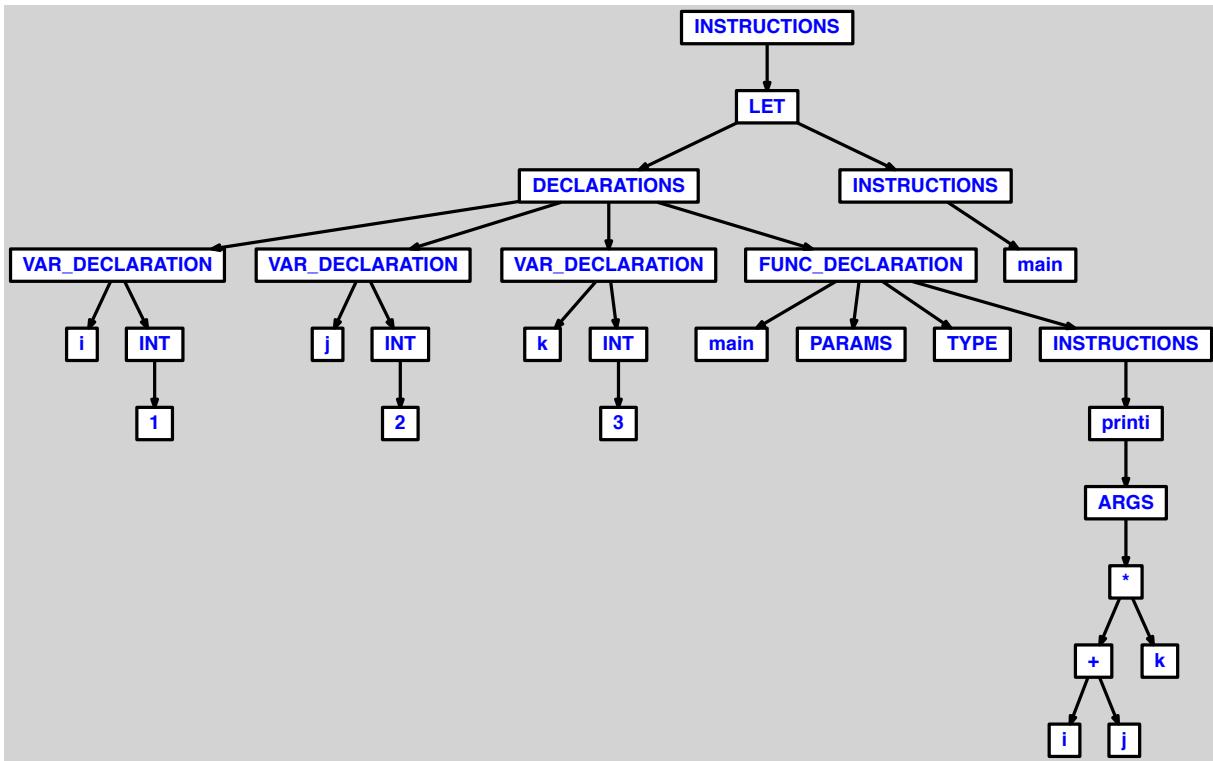
### 3.2.83 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i+j)*k)
in main() end

```

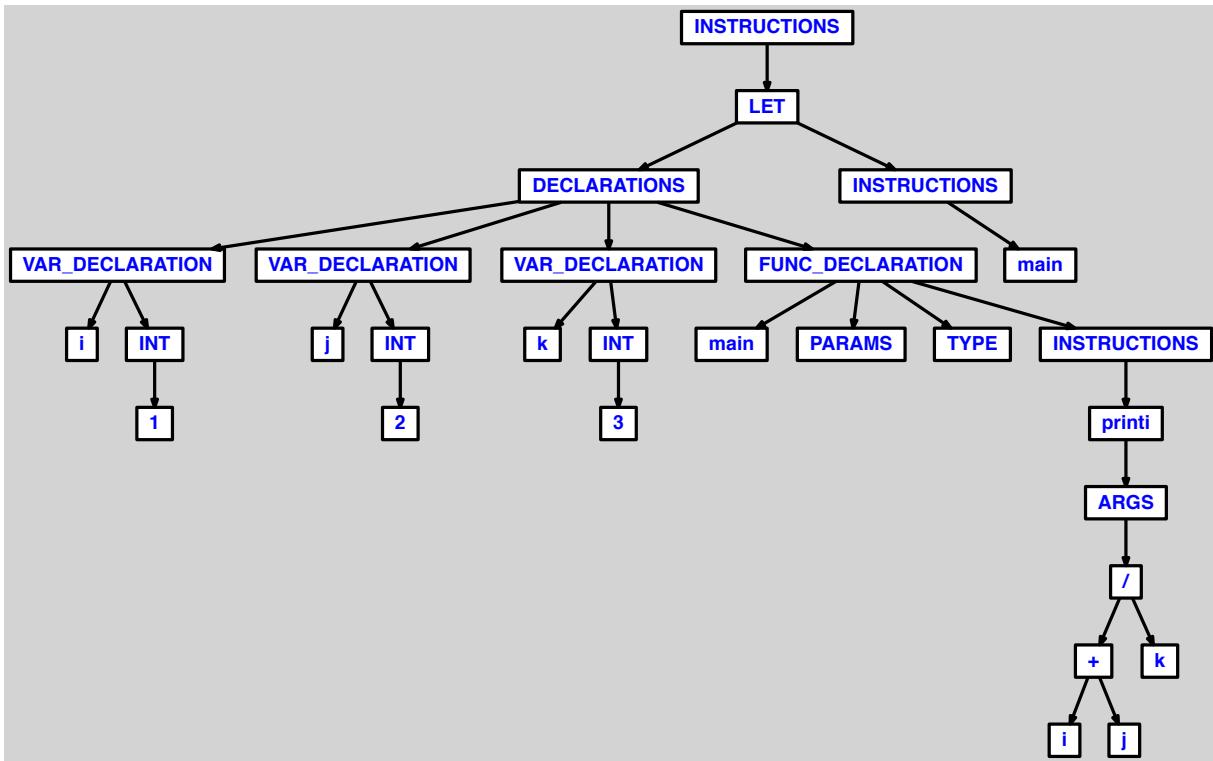


### 3.2.84 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i+j)/k)
in main() end
  
```



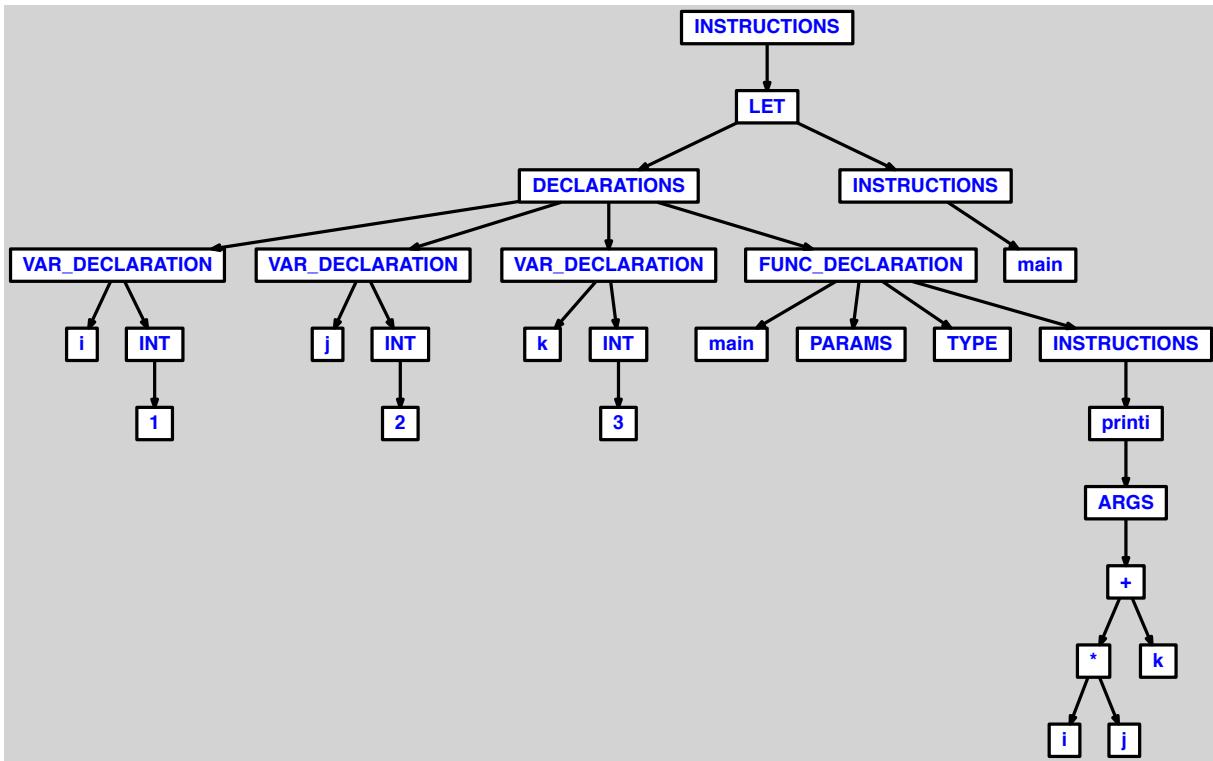
### 3.2.85 multiplication suivie d'addition, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i*j)+k)
in main() end

```



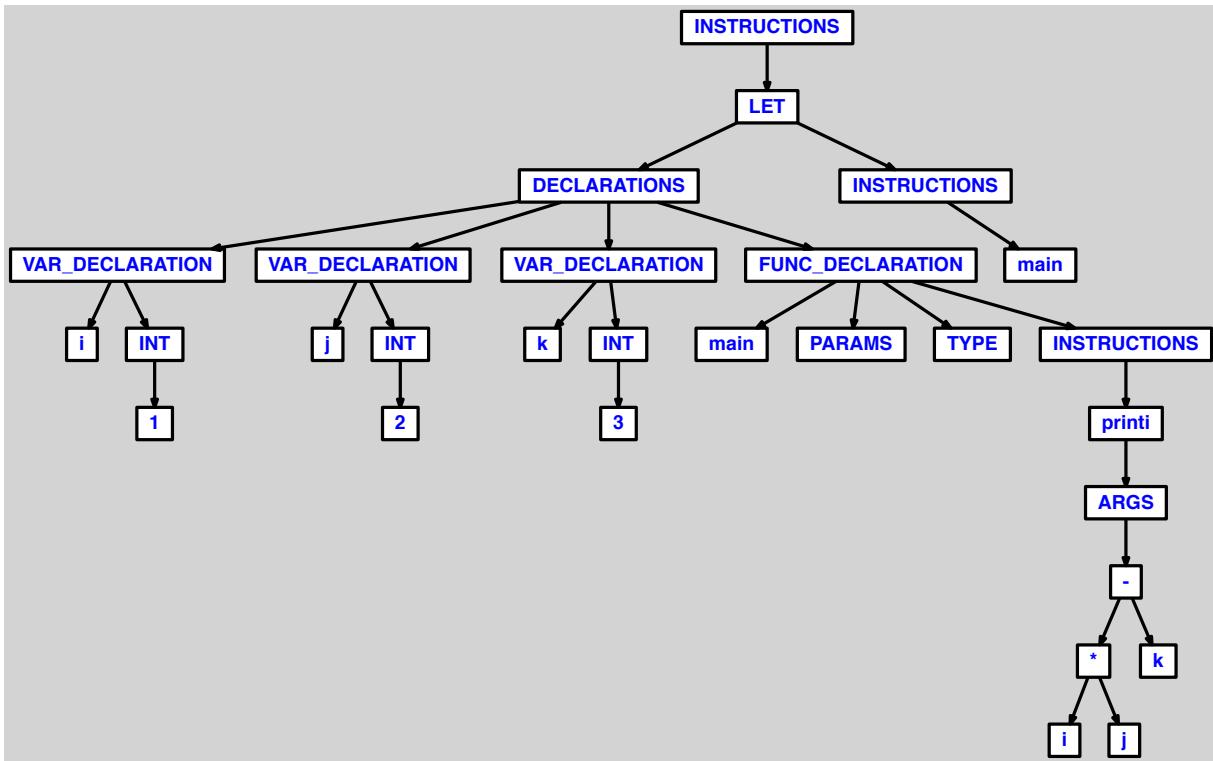
### 3.2.86 multiplication suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i*j)-k)
in main() end

```



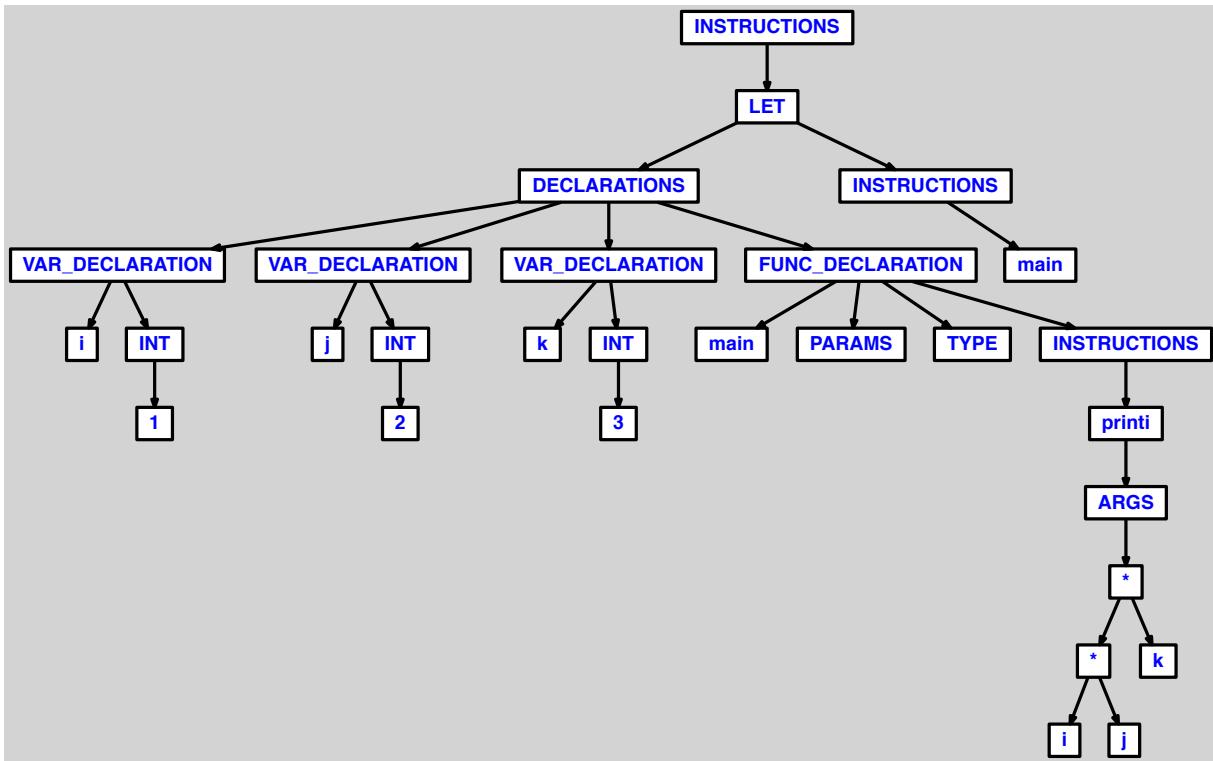
### 3.2.87 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i*j)*k)
in main() end

```



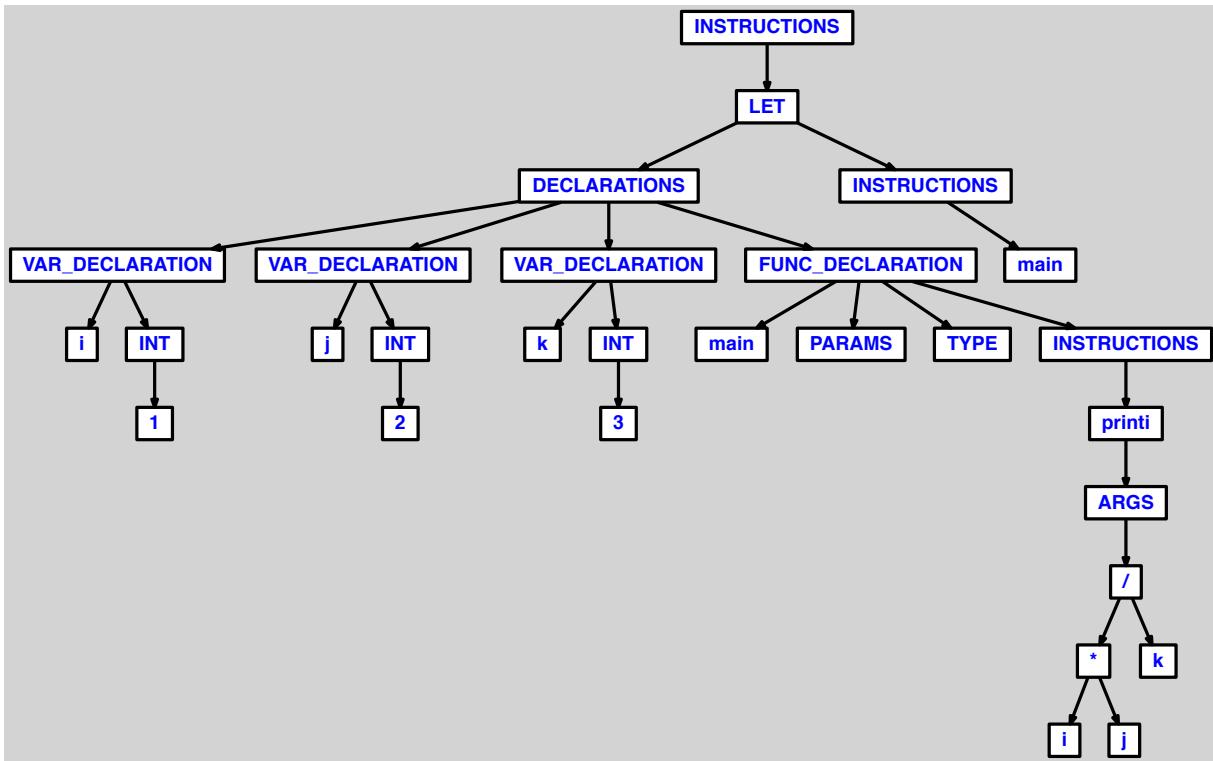
3.2.88 multiplication suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i*j)/k)
in main() end

```



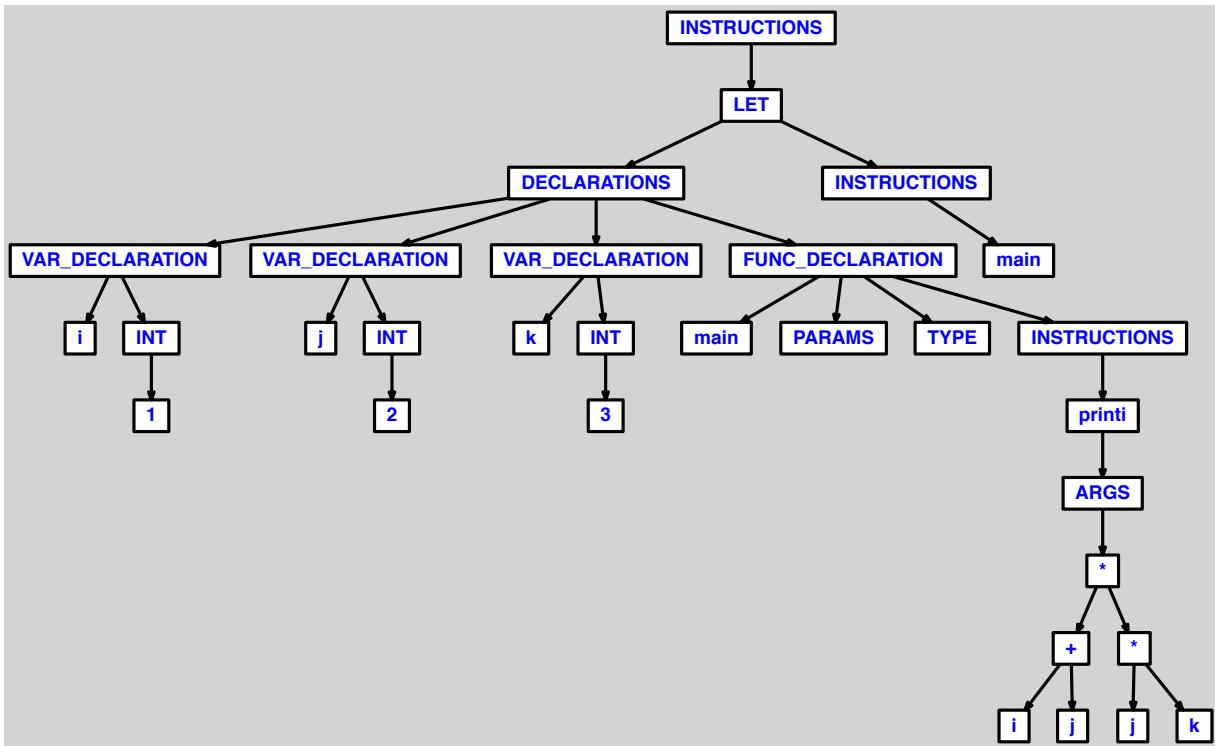
### 3.2.89 multiplication de 2 additions parenthesees, avec termes identifies par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i+j)*(j*k))
in main() end

```



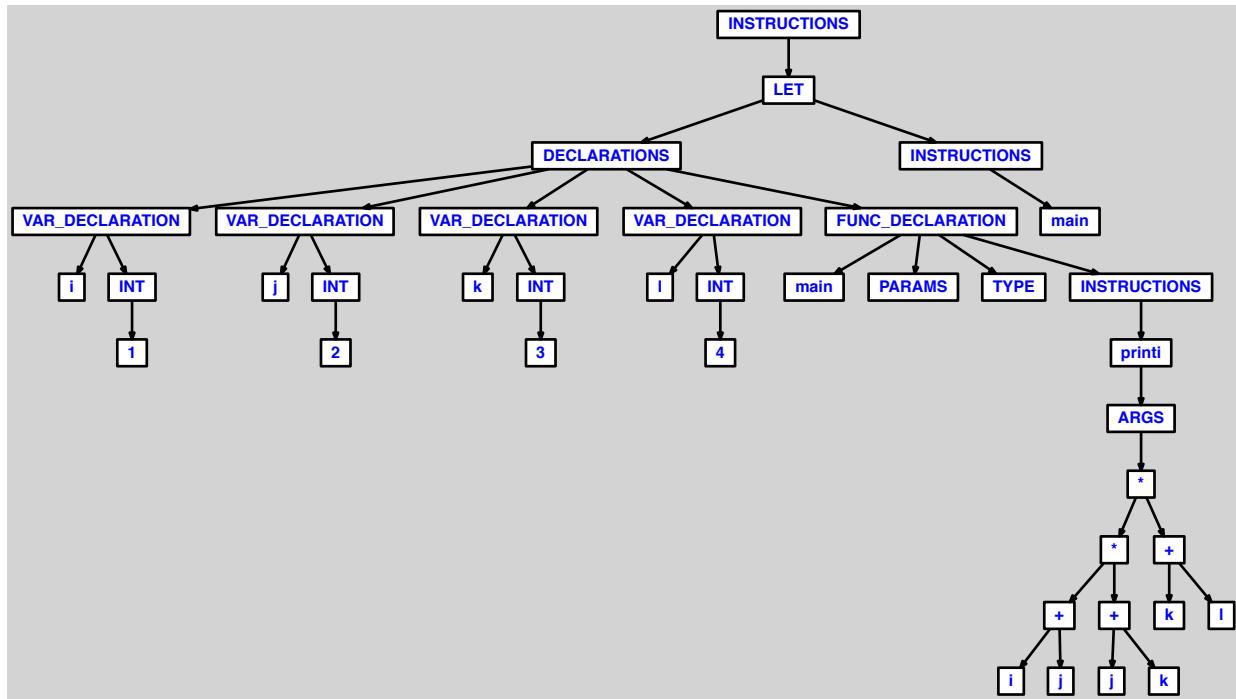
### 3.2.90 multiplication de 3 additions parenthesees, avec termes identifies par variables

```

let
    var i := 1
    var j := 2
    var k := 3
    var l := 4

    function main() = printi((i+j)*(j+k)*(k+l))
in main() end

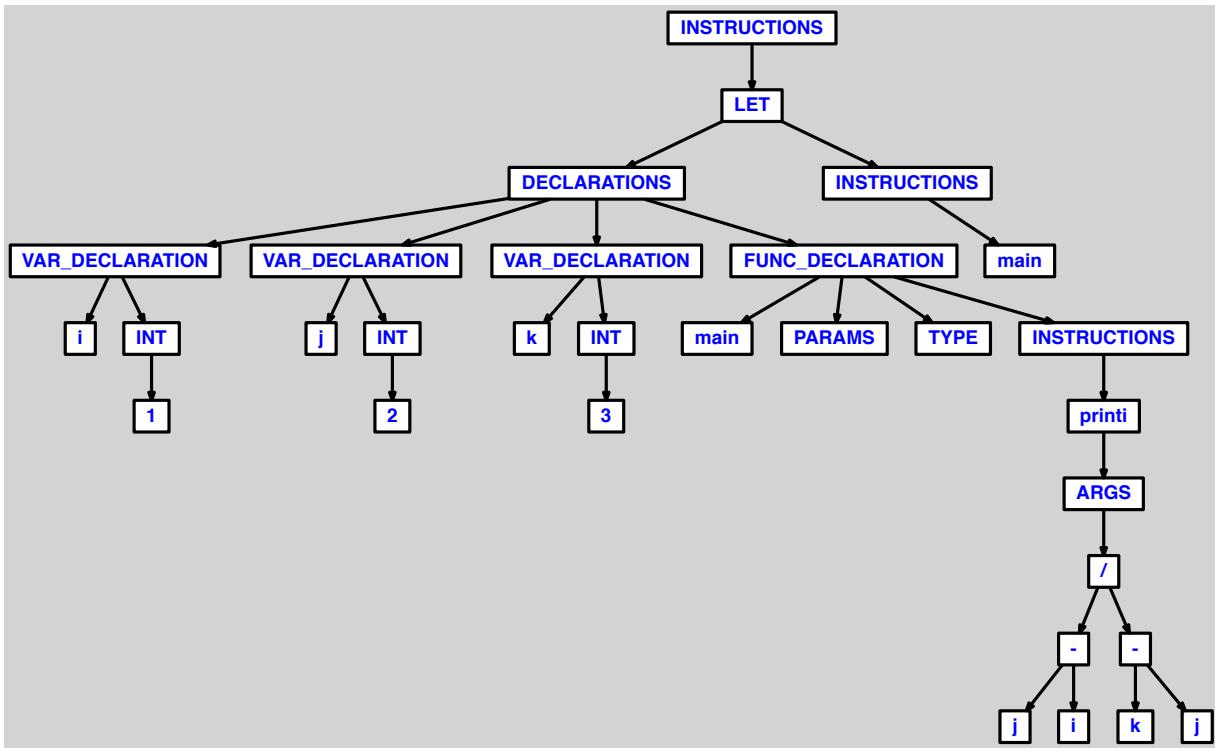
```



### 3.2.91 division de 2 soustractions parenthesees, avec termes identifies par variables

```
let
    var i := 1
    var j := 2
    var k := 3

    function main() = printi((j-i)/(k-j))
in main() end
```



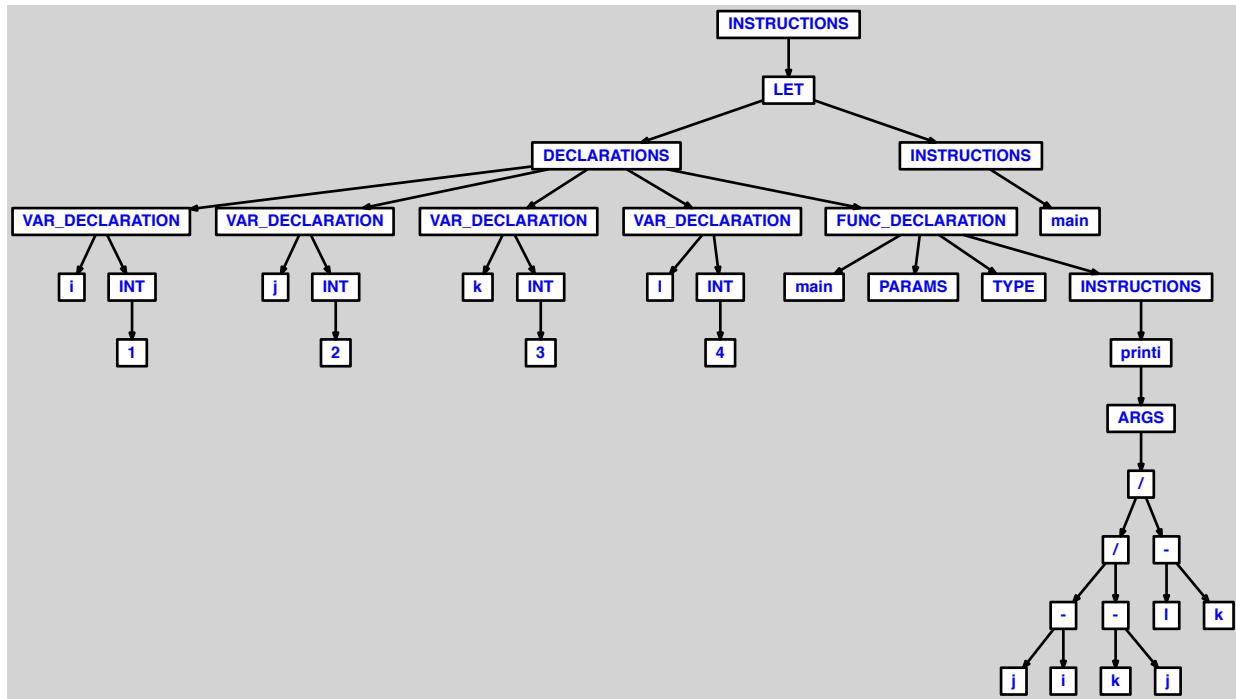
### 3.2.92 division de 3 soustractions parenthesees, avec termes identifies par variables

```

let
    var i := 1
    var j := 2
    var k := 3
    var l := 4

    function main() = printi((j-i)/(k-j)/(l-k))
in main() end

```



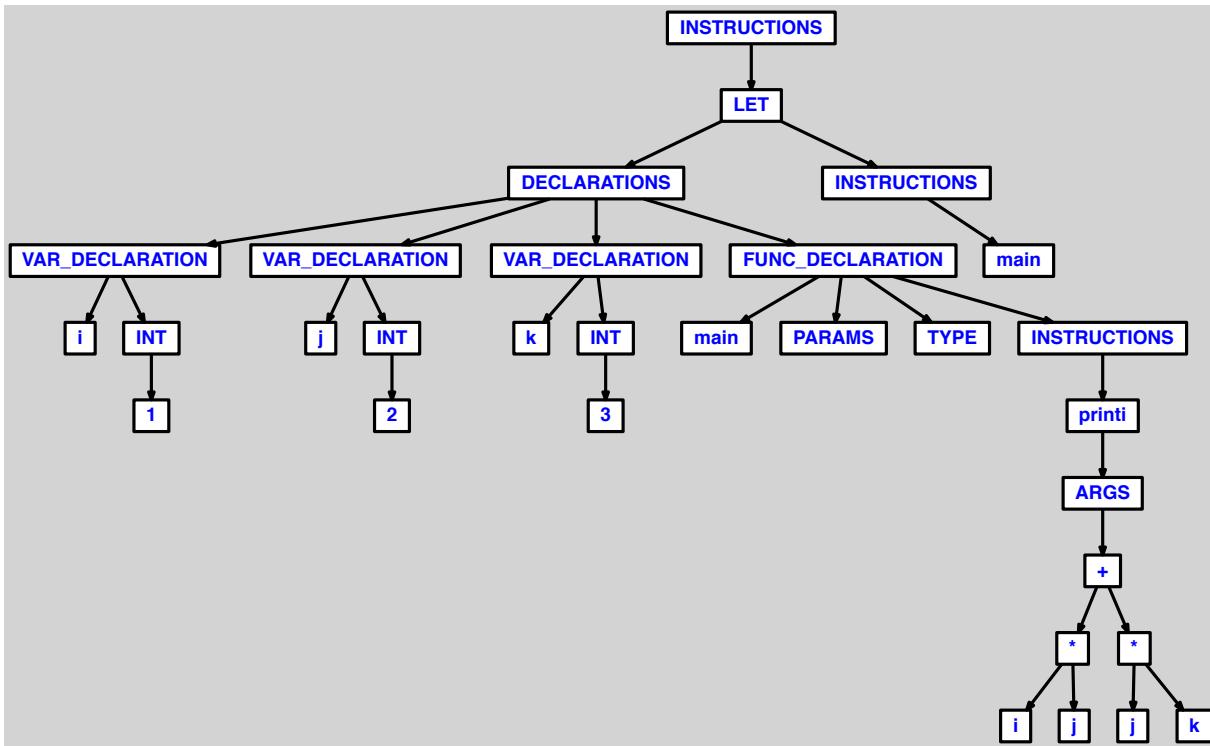
### 3.2.93 addition de 2 multiplications parenthesees, avec termes identifies par variables

```

let
  var i := 1
  var j := 2
  var k := 3

  function main() = printi((i*j)+(j*k))
in main() end

```

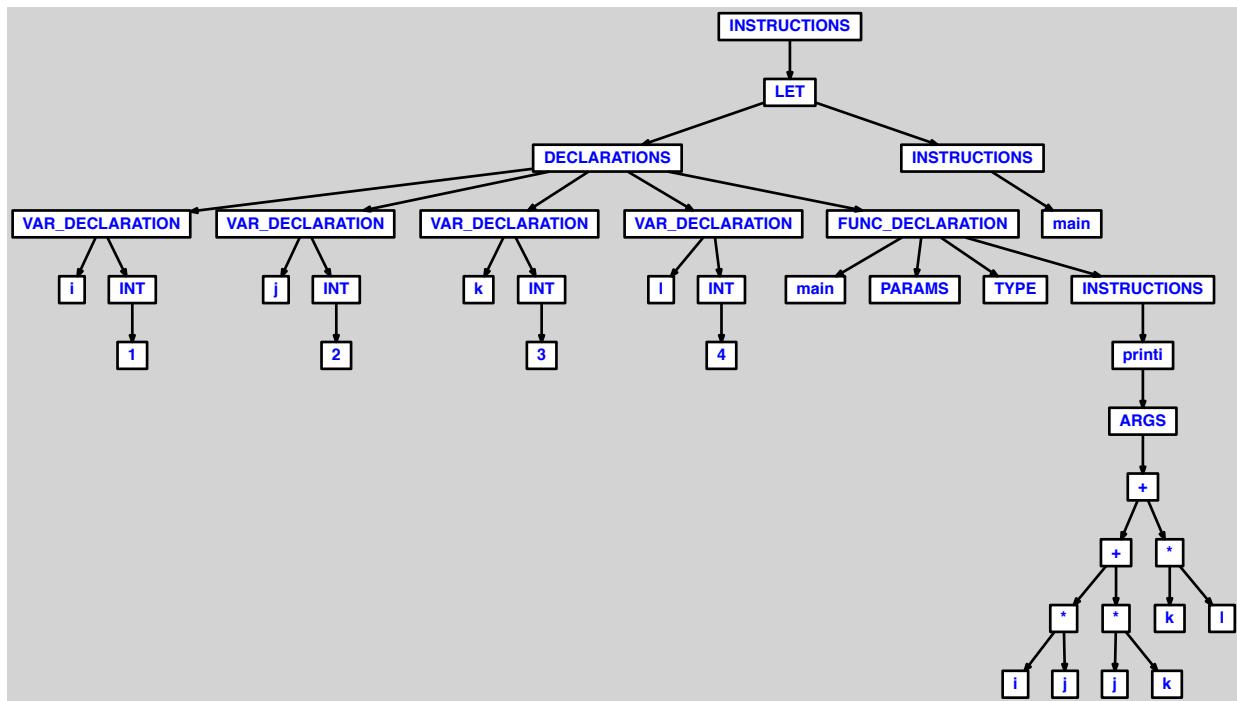


### 3.2.94 addition de 3 multiplications parenthesees, avec termes identifies par variables

```

let
  var i := 1
  var j := 2
  var k := 3
  var l := 4

  function main() = printi((i*j)+(j*k)+(k*l))
in main() end
  
```



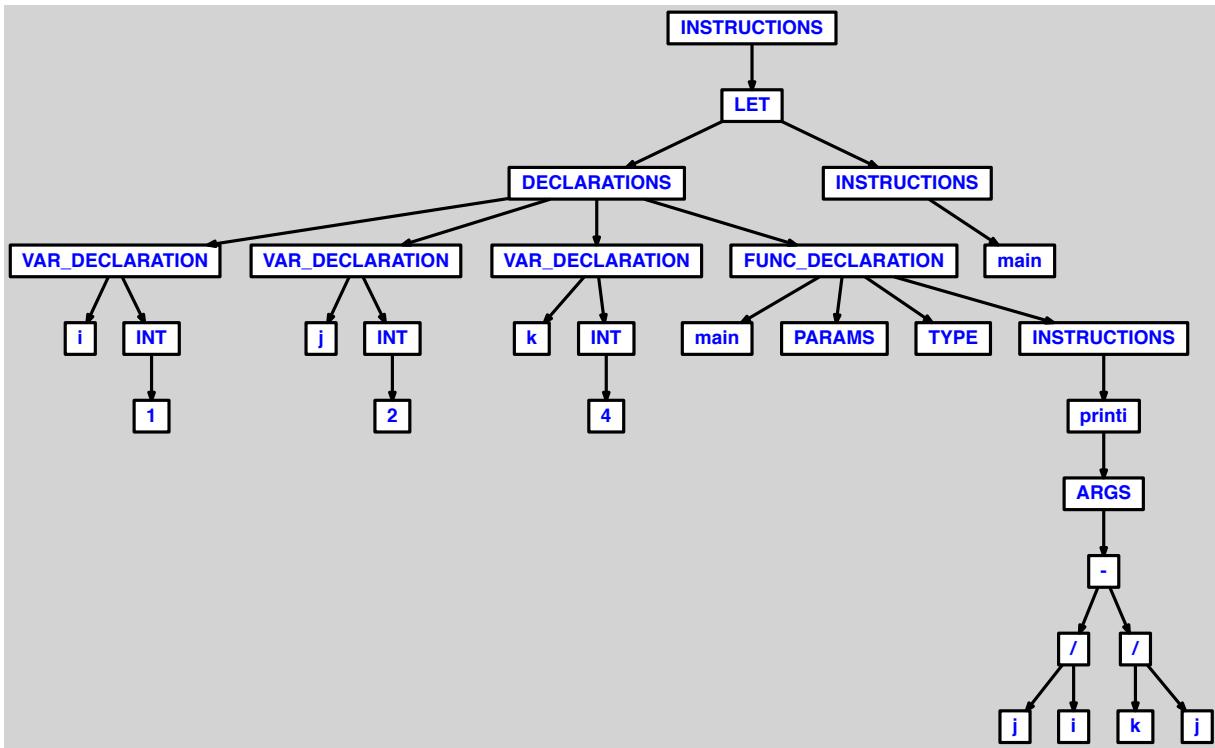
### 3.2.95 soustraction de 2 divisions parenthesees, avec termes identifies par variables

```

let
    var i := 1
    var j := 2
    var k := 4

    function main() = printi((j/i)-(k/j))
in main() end

```

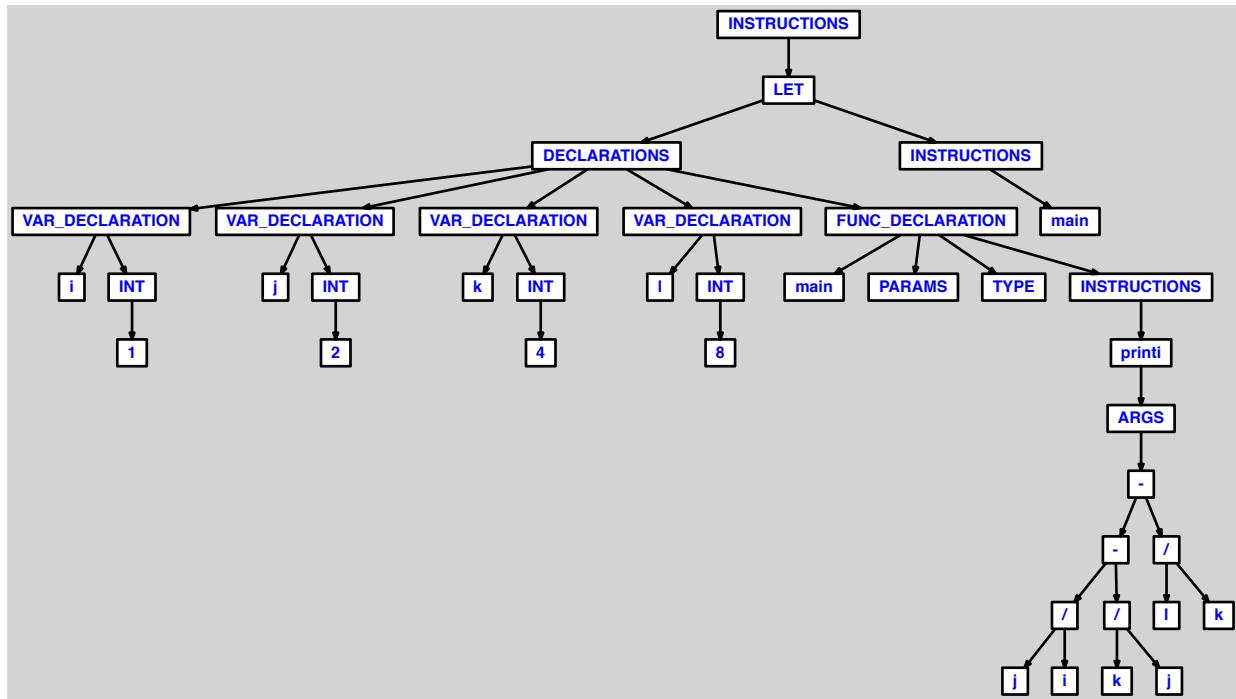


### 3.2.96 soustraction de 3 divisions parenthesees, avec termes identifies par variables

```

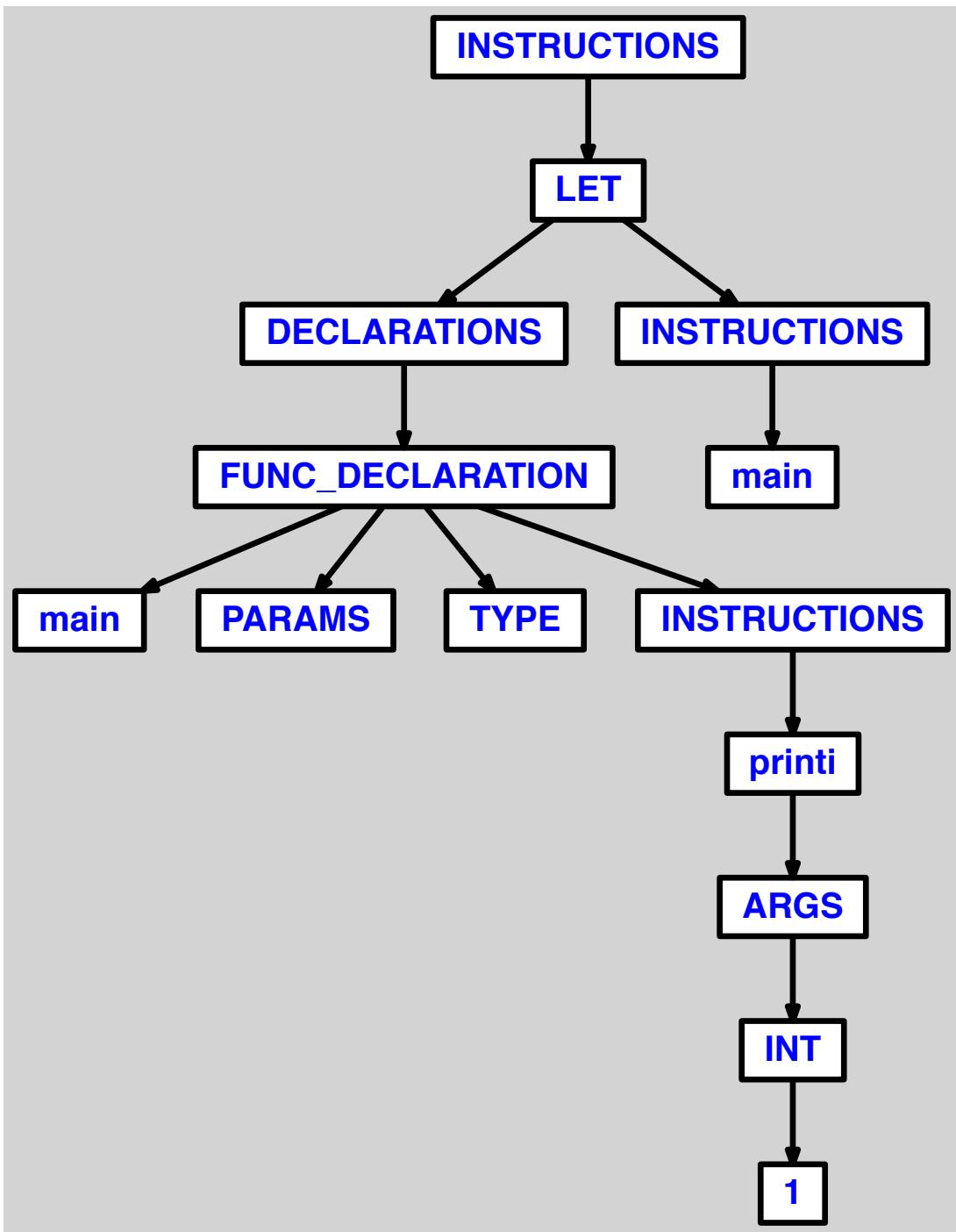
let
  var i := 1
  var j := 2
  var k := 4
  var l := 8

  function main() = printi((j/i)-(k/j)-(l/k))
in main() end
  
```



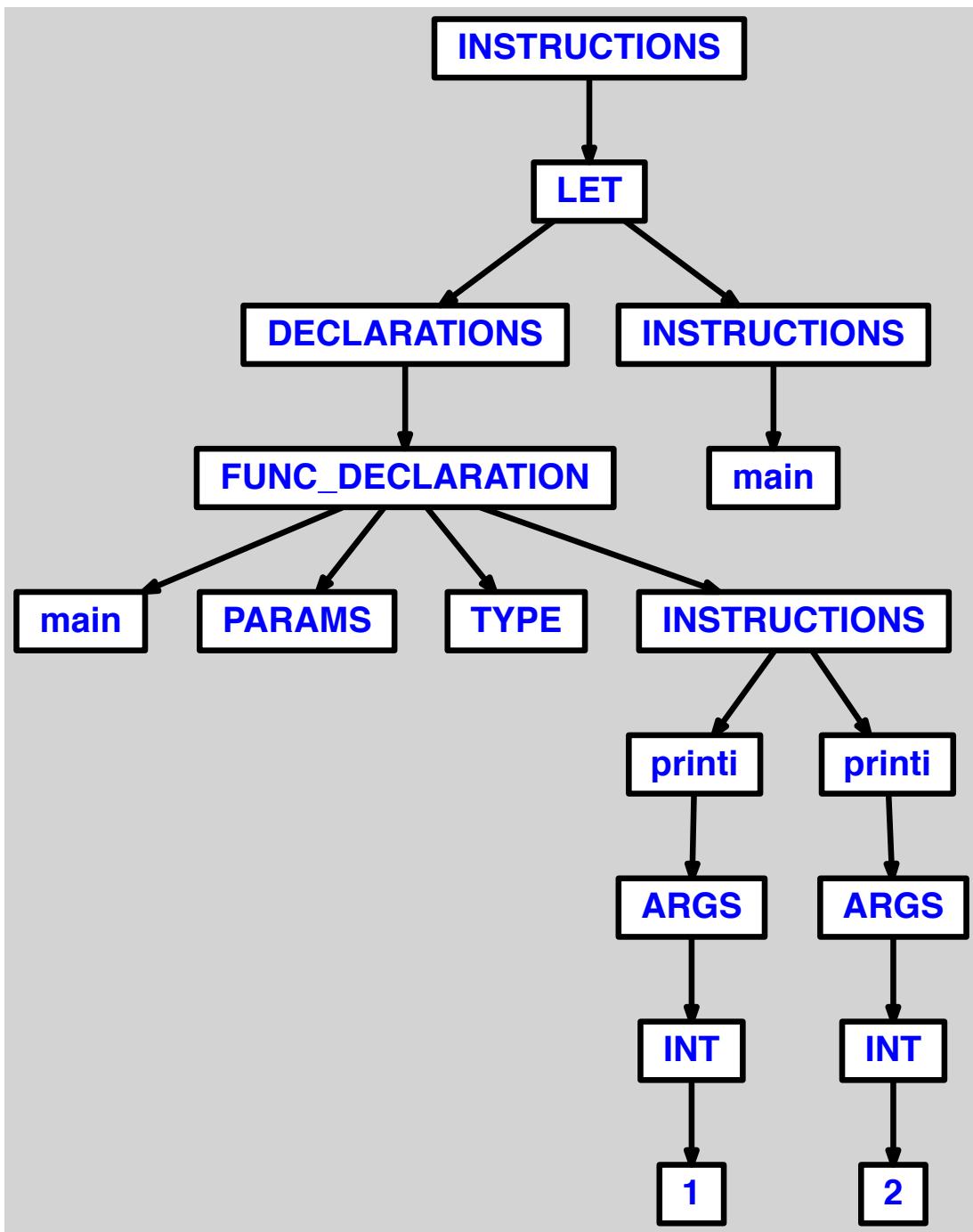
### 3.2.97 parenthesage d'une instruction

```
let function main() = (printi(1)) in main() end
```



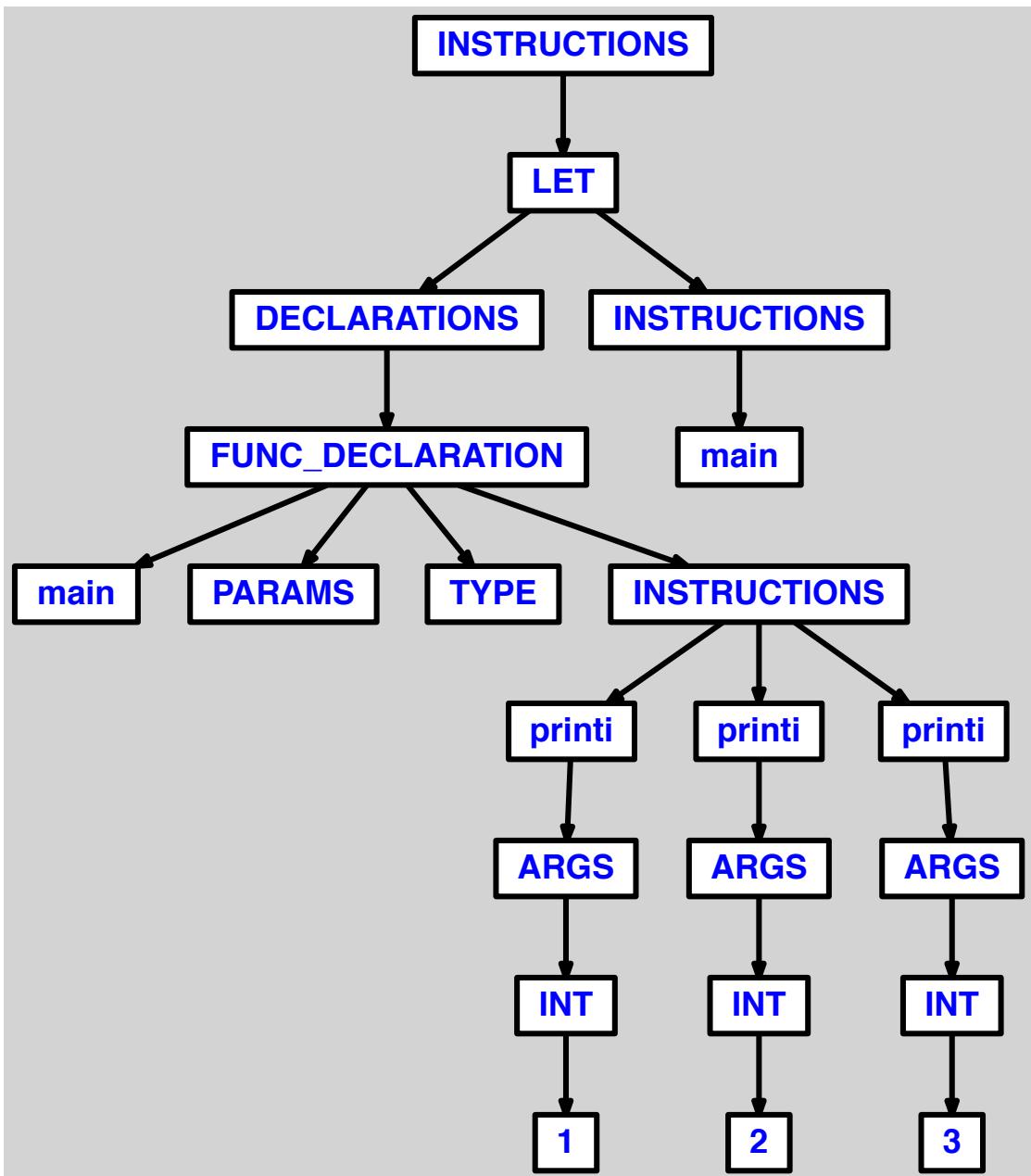
### 3.2.98 parenthesage de 2 instructions

```
let function main() = (printi(1); printi(2)) in main() end
```



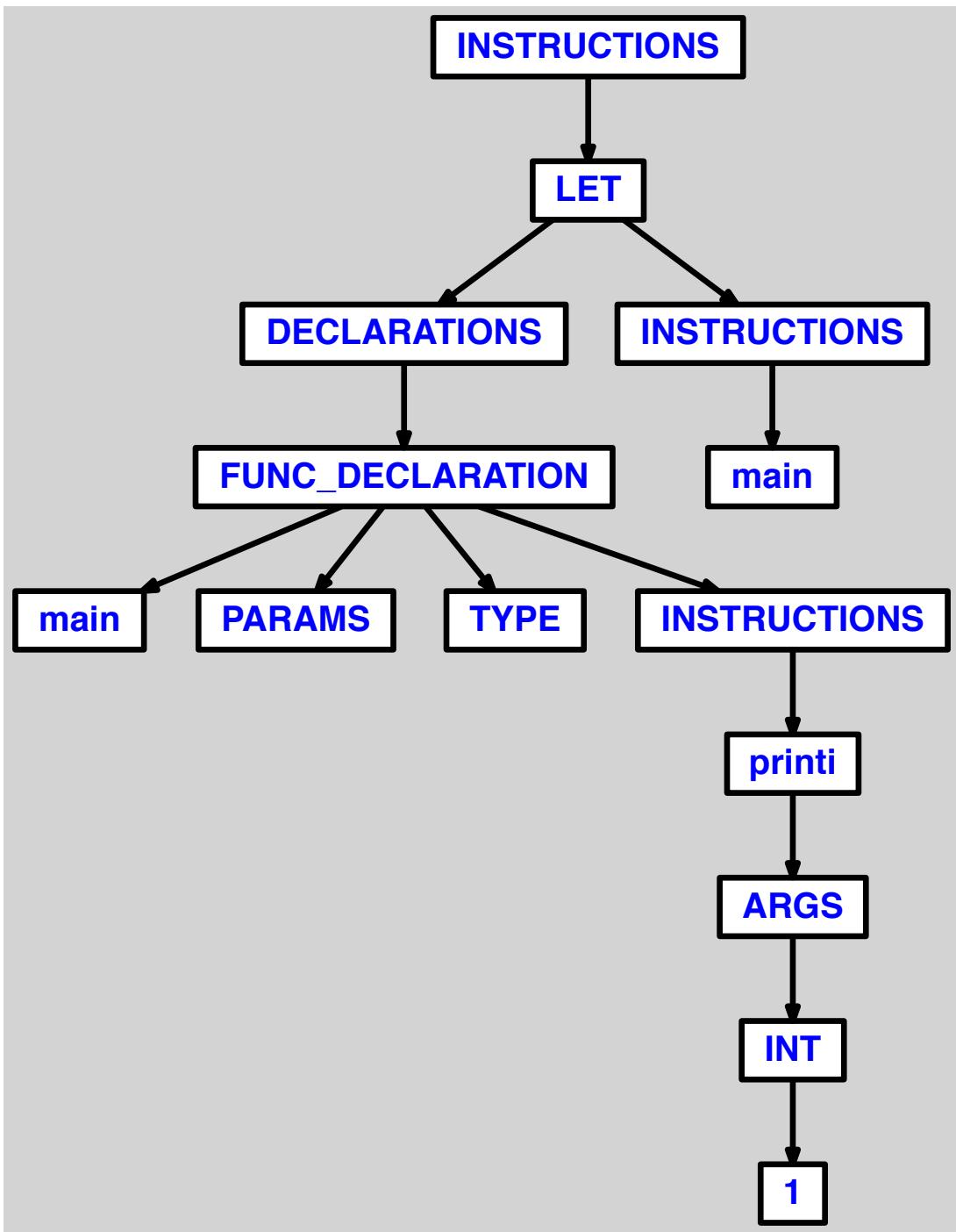
### 3.2.99 parenthesage de 3 instructions

```
let function main() = (printi(1); printi(2); printi(3)) in main() end
```



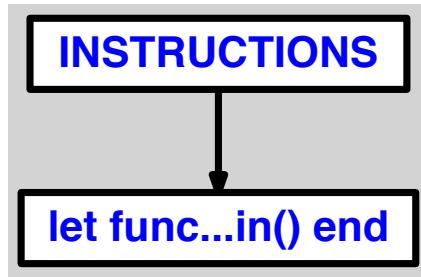
### 3.2.100 1 instruction sans parentheses

```
let function main() = printi(1) in main() end
```



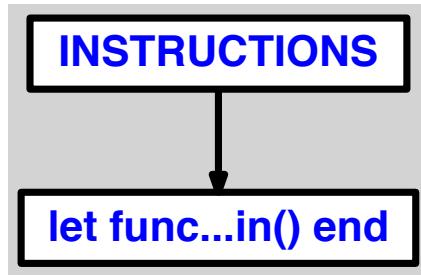
### 3.2.101 2 instructions sans parentheses

```
let function main() = printi(1); printi(2) in main() end
```



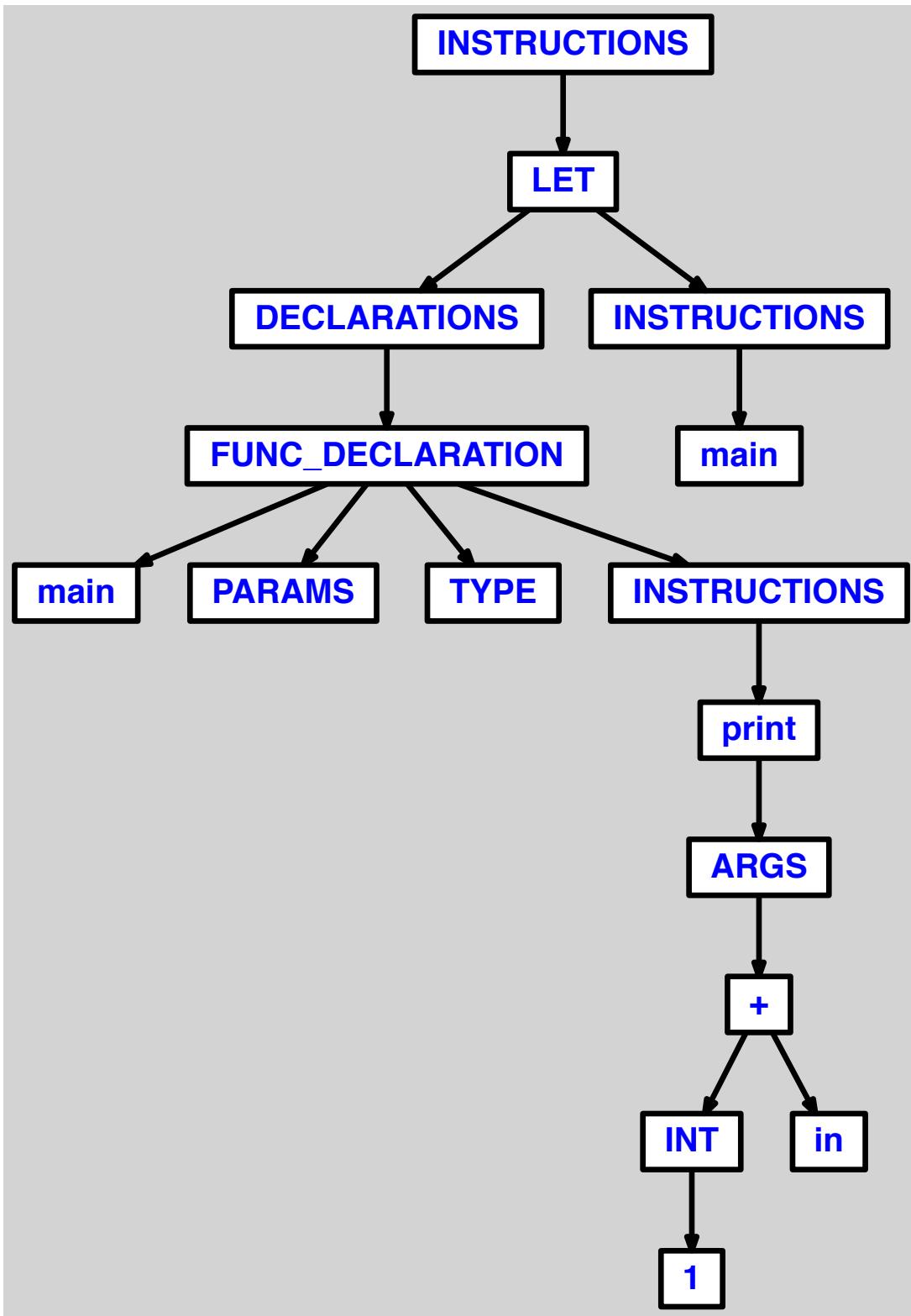
### 3.2.102 3 instructions sans parenthèses

```
let function main() = printi(1); printi(2); printi(3) in main() end
```



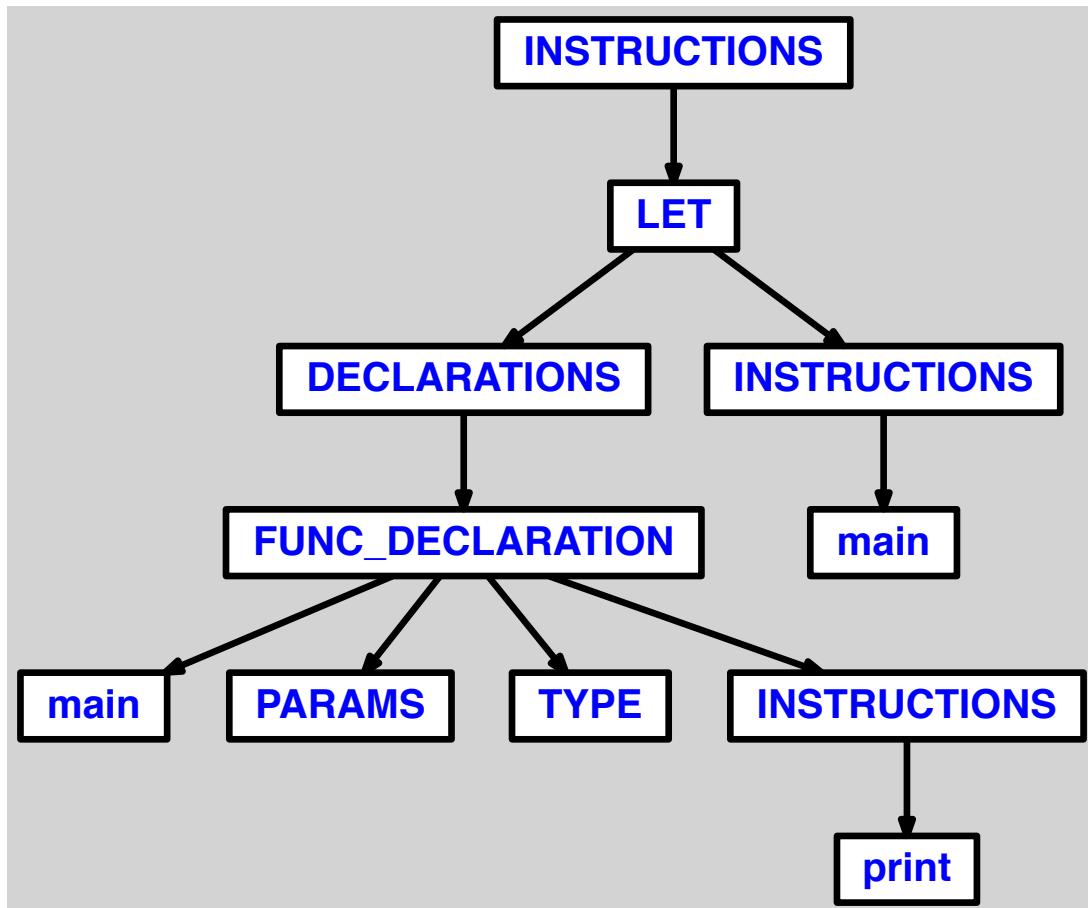
### 3.2.103 concatenation d'entier et de chaine

```
let
    function main() = print(1+"2")
in main() end
```



### 3.2.104 concatenation de 2 chaines

```
let
    function main() = print("1"+"2")
in main() end
```

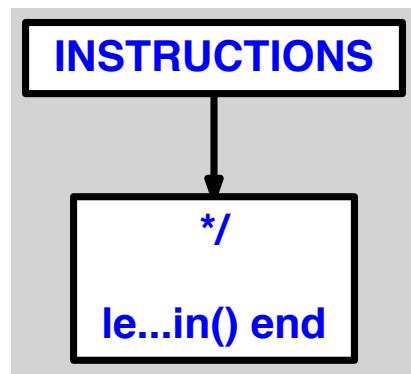


## 4 comment

### 4.1 KO

#### 4.1.1 imbrication

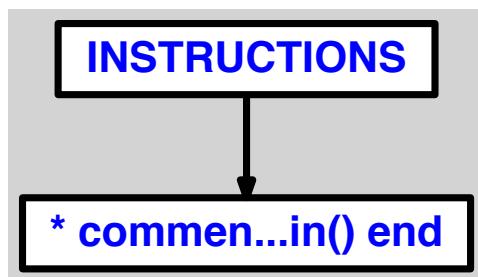
```
/*
    un commentaire
    /* imbrique dans ce commentaire */
*/
let
    function main() = print("test")
in main() end
```



#### 4.1.2 oubli de / en debut de commentaire

```
* commentaire */

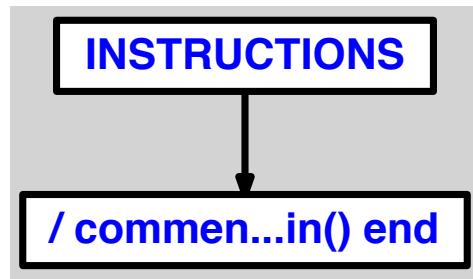
let
    function main() = print("test")
in main() end
```



#### 4.1.3 oubli de \* en debut de commentaire

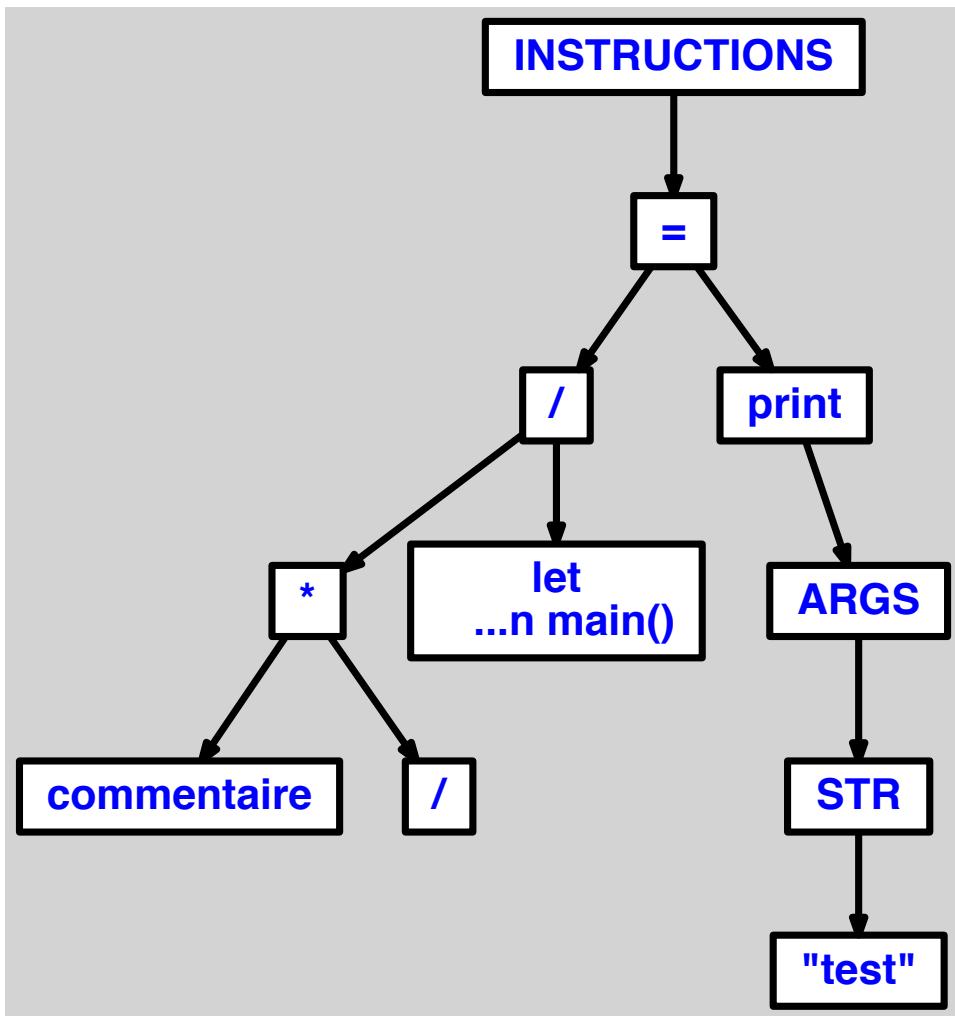
```
/ commentaire */

let
    function main() = print("test")
in main() end
```



#### 4.1.4 oubli de en debut de commentaire

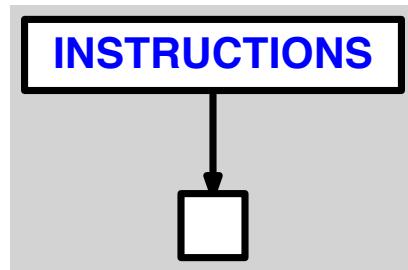
```
commentaire */  
  
let  
    function main() = print("test")  
in main() end
```



#### 4.1.5 oubli de \* en fin de commentaire

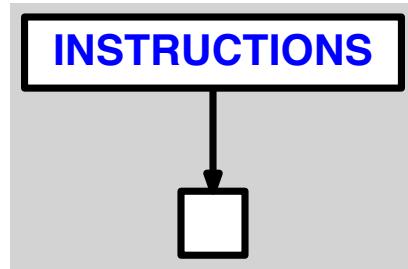
```

/* commentaire /
let
    function main() = print("test")
in main() end
  
```



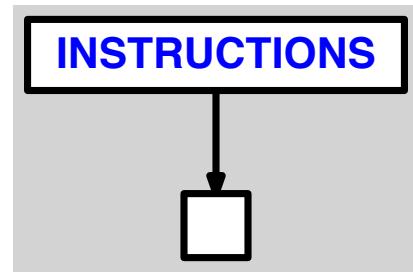
#### 4.1.6oubli de / en fin de commentaire

```
/* commentaire *  
  
let  
    function main() = print("test")  
in main() end
```



#### 4.1.7oubli de en fin de commentaire

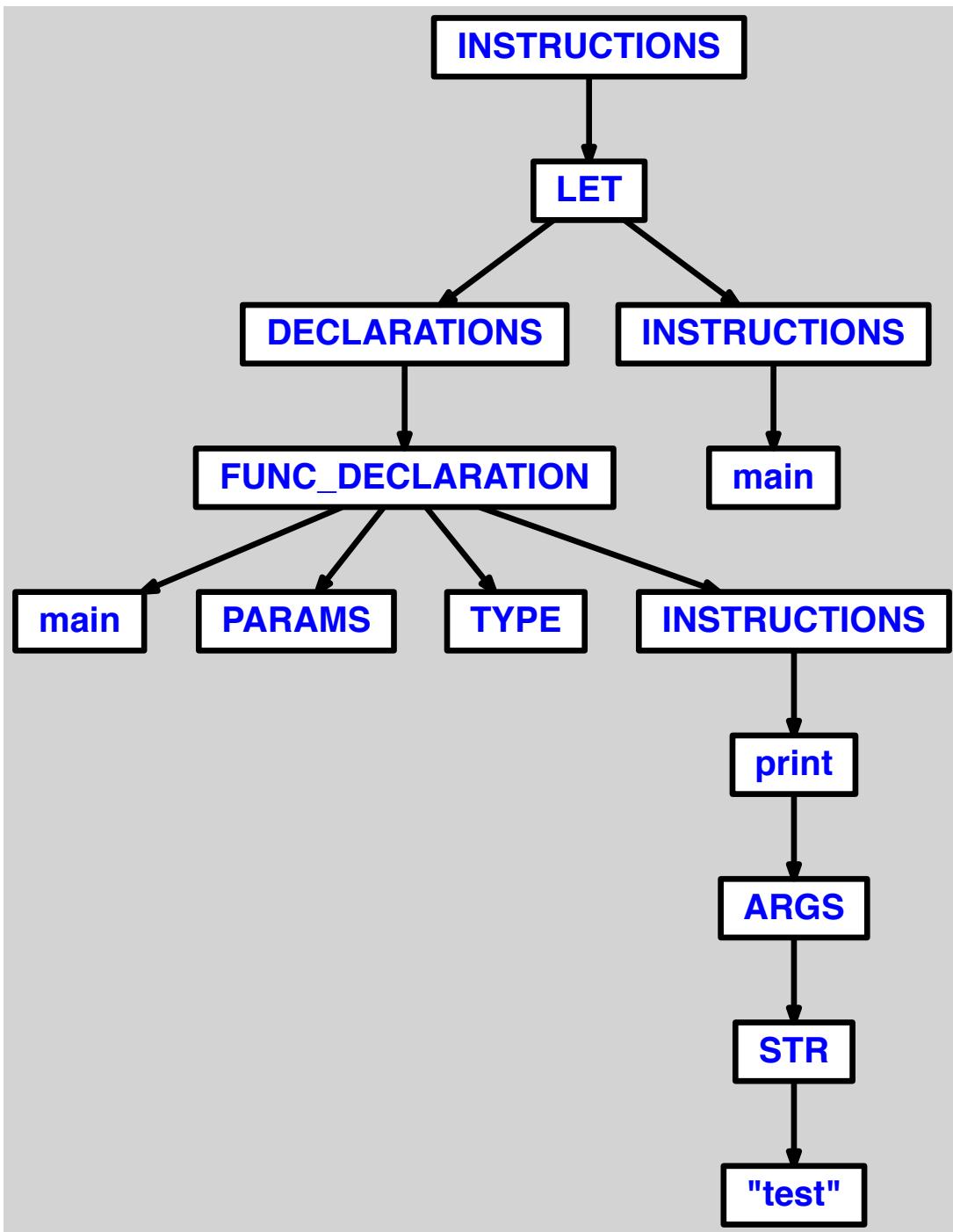
```
/* commentaire *  
  
let  
    function main() = print("test")  
in main() end
```



## 4.2 OK

### 4.2.1 1 commentaire, sans instruction avant, sur 1 ligne

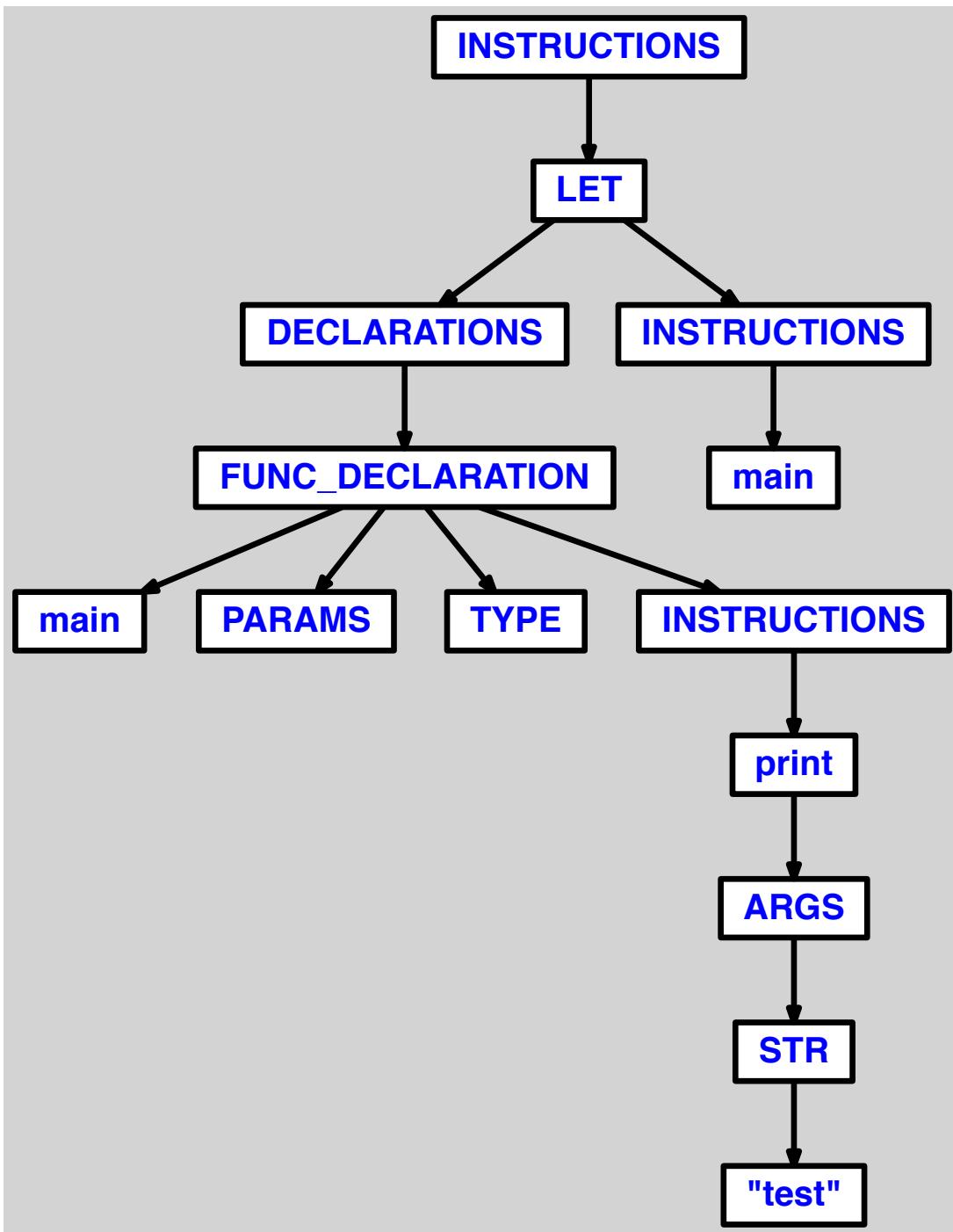
```
/* commentaire */  
  
let  
    function main() = print("test")  
in main() end
```



#### 4.2.2 1 commentaire, sans instruction avant, sur plusieurs lignes

```
/*  
1
```

```
2
3
*/
let
    function main() = print("test")
in main() end
```

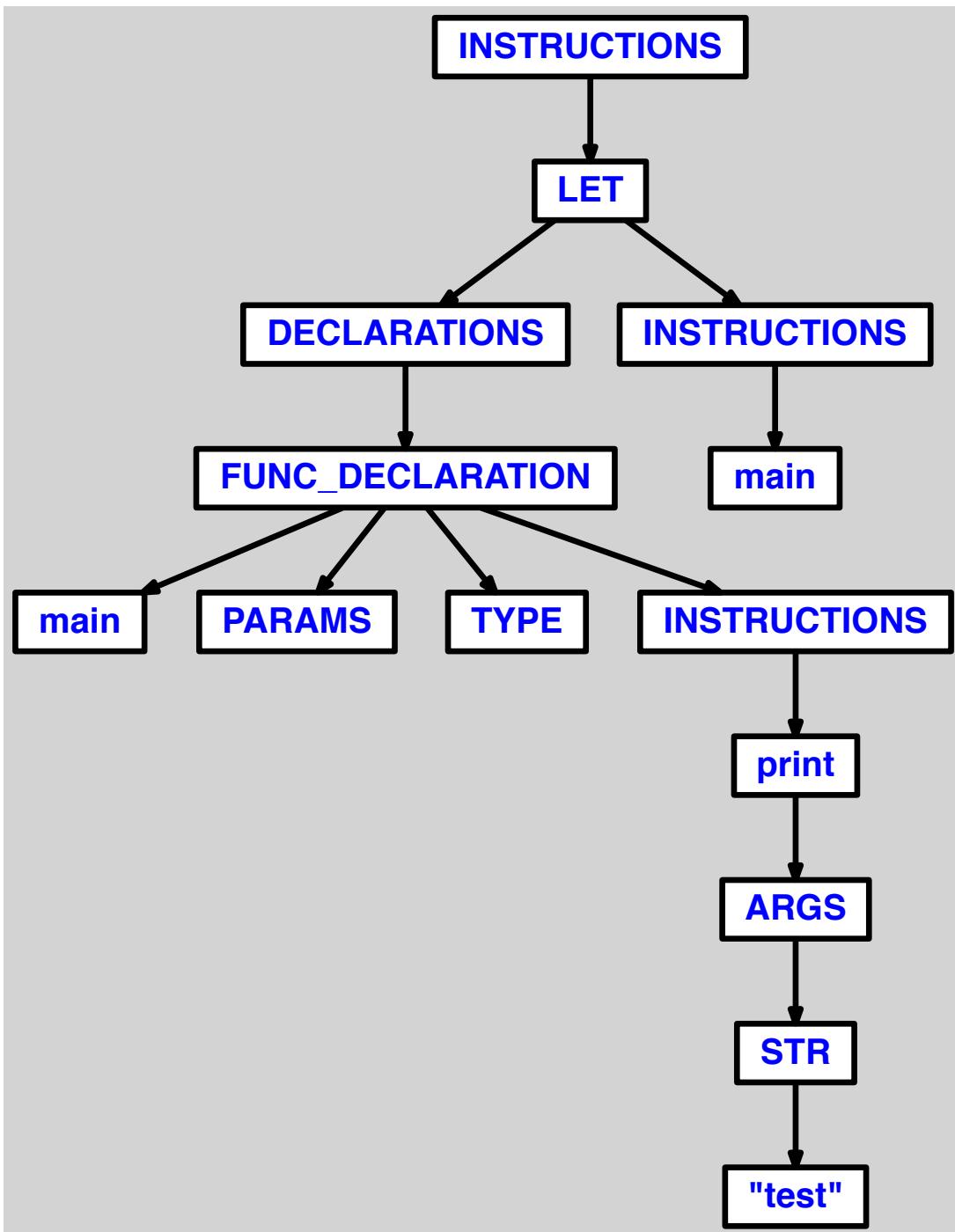


#### 4.2.3 1 commentaire, apres instruction, sur 1 ligne

```

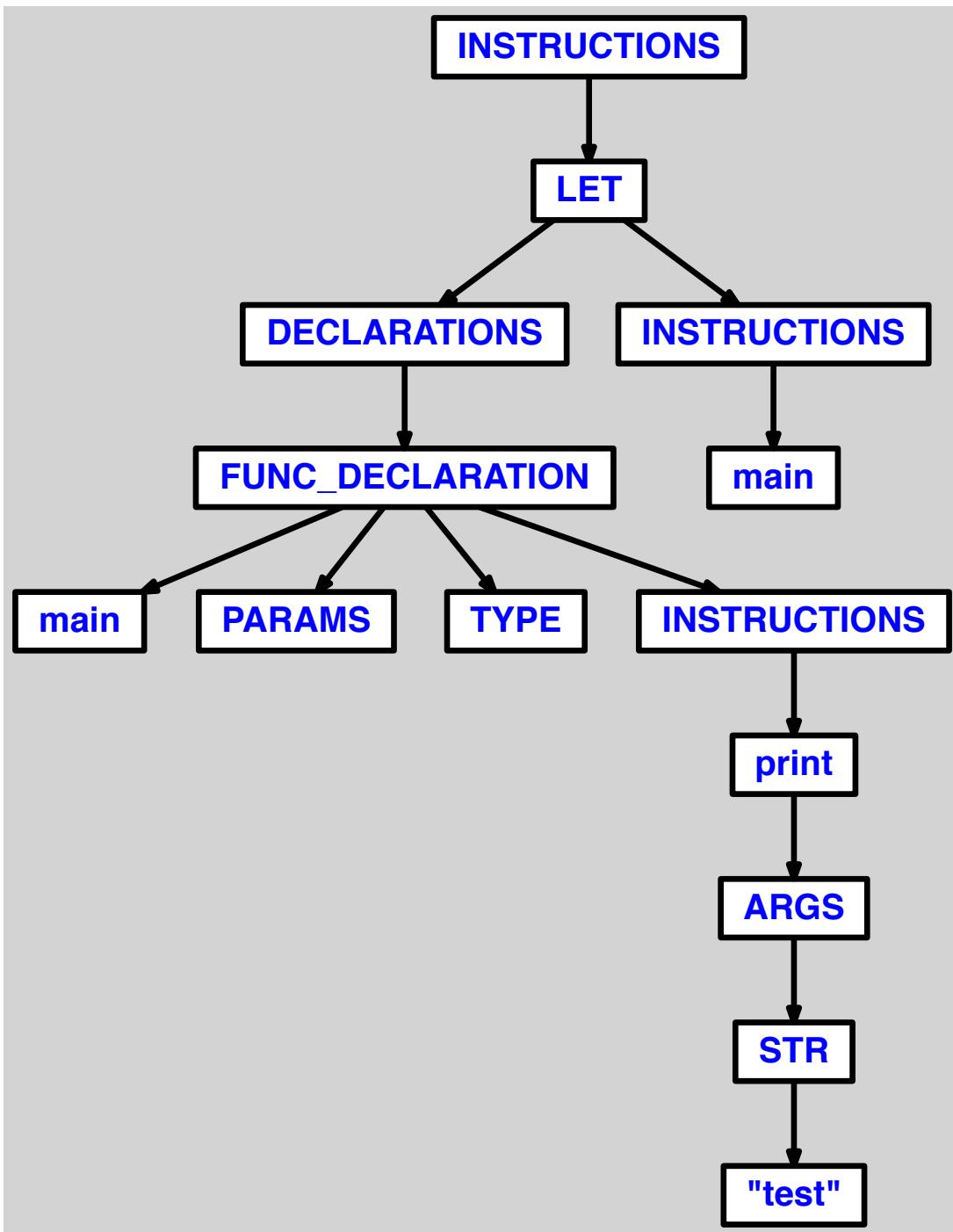
let
function main() = print("test")
  
```

```
/* commentaire */  
in main() end
```



#### 4.2.4 1 commentaire, apres instruction, sur plusieurs lignes

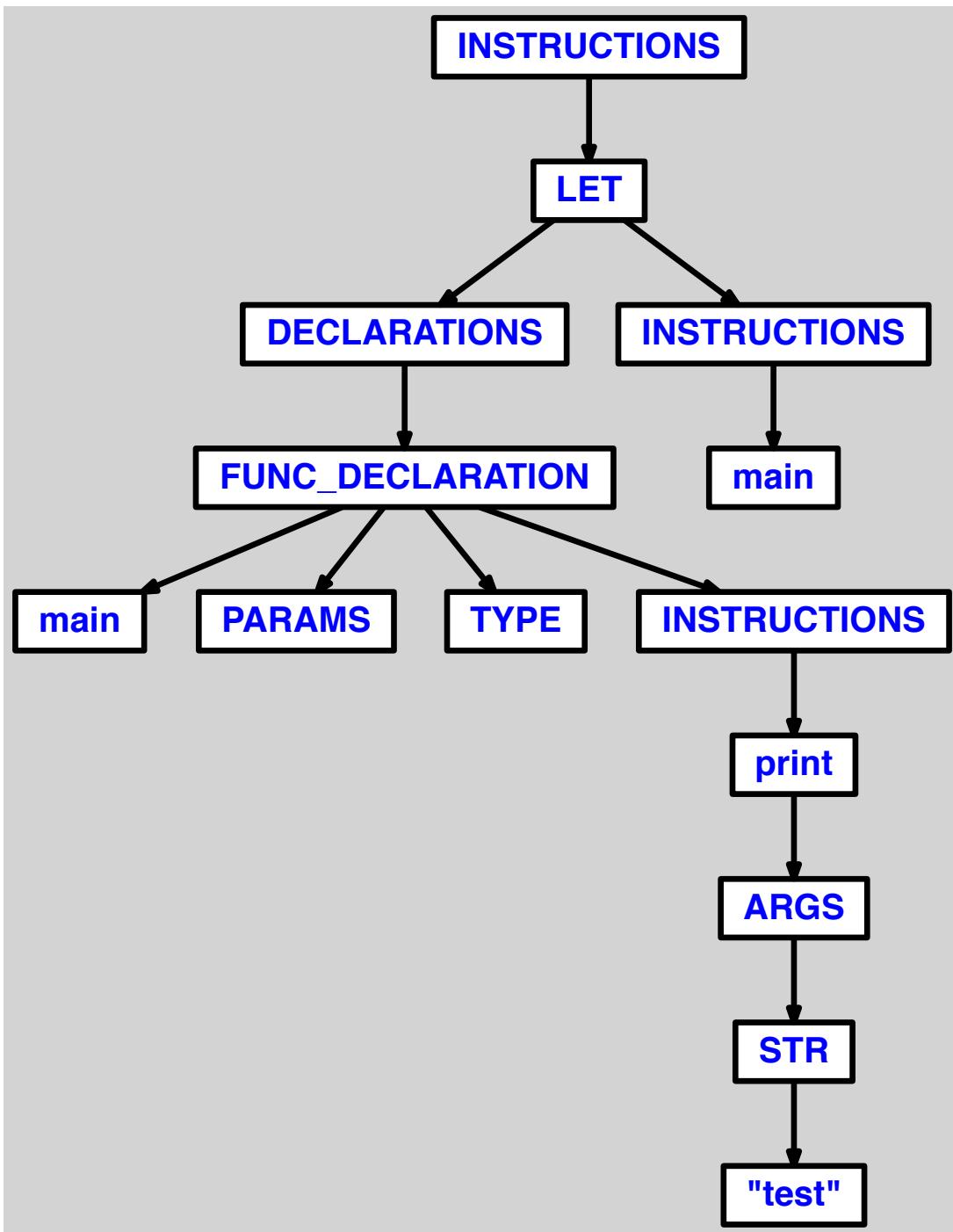
```
let
    function main() = print("test")
    /*
    1
    2
    3
    */
in main() end
```



#### 4.2.5 2 commentaires, sans instruction avant, sur 1 ligne

```
/* commentaire 1 */
```

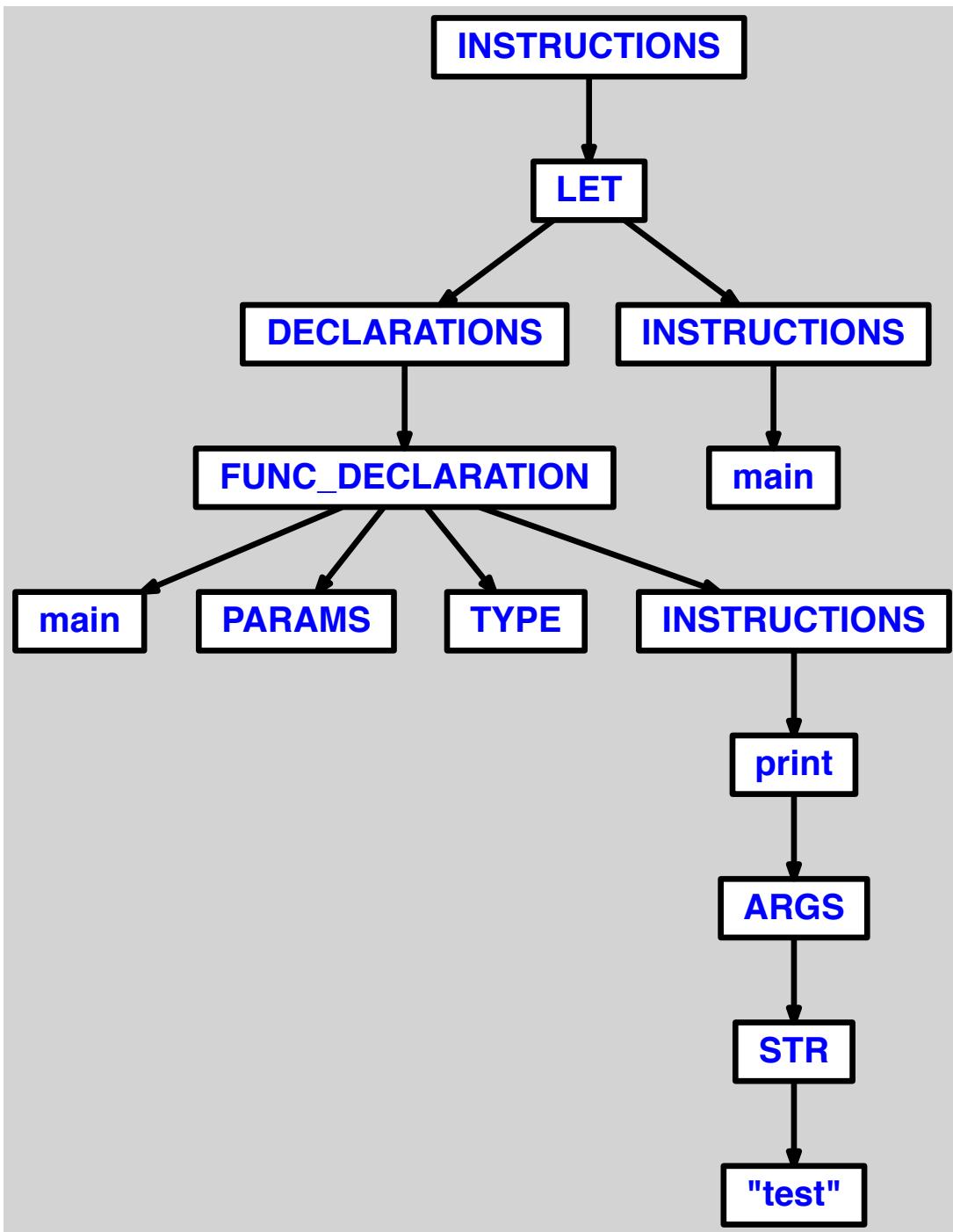
```
/* commentaire 2 */  
  
let  
    function main() = print("test")  
in main() end
```



#### 4.2.6 2 commentaires, sans instruction avant, sur plusieurs lignes

```
/*  
1
```

```
2
3
*/
/*
4
5
6
*/
let
    function main() = print("test")
in main() end
```

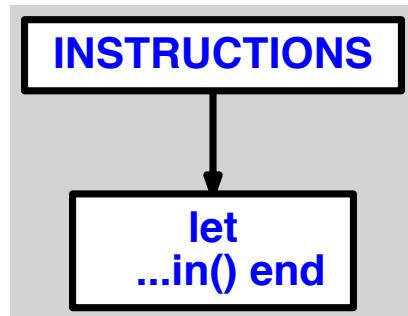


#### 4.2.7 2 commentaires, apres instructions, sur 1 ligne

```

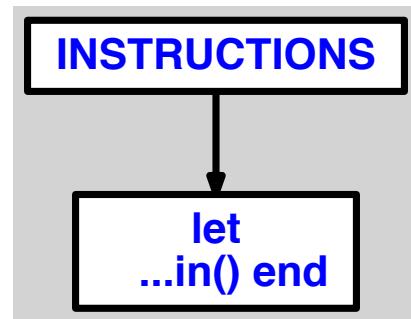
let
function main() =
  
```

```
print("test");
/* commentaire 1 */
print("test")
/* commentaire 2 */
in main() end
```



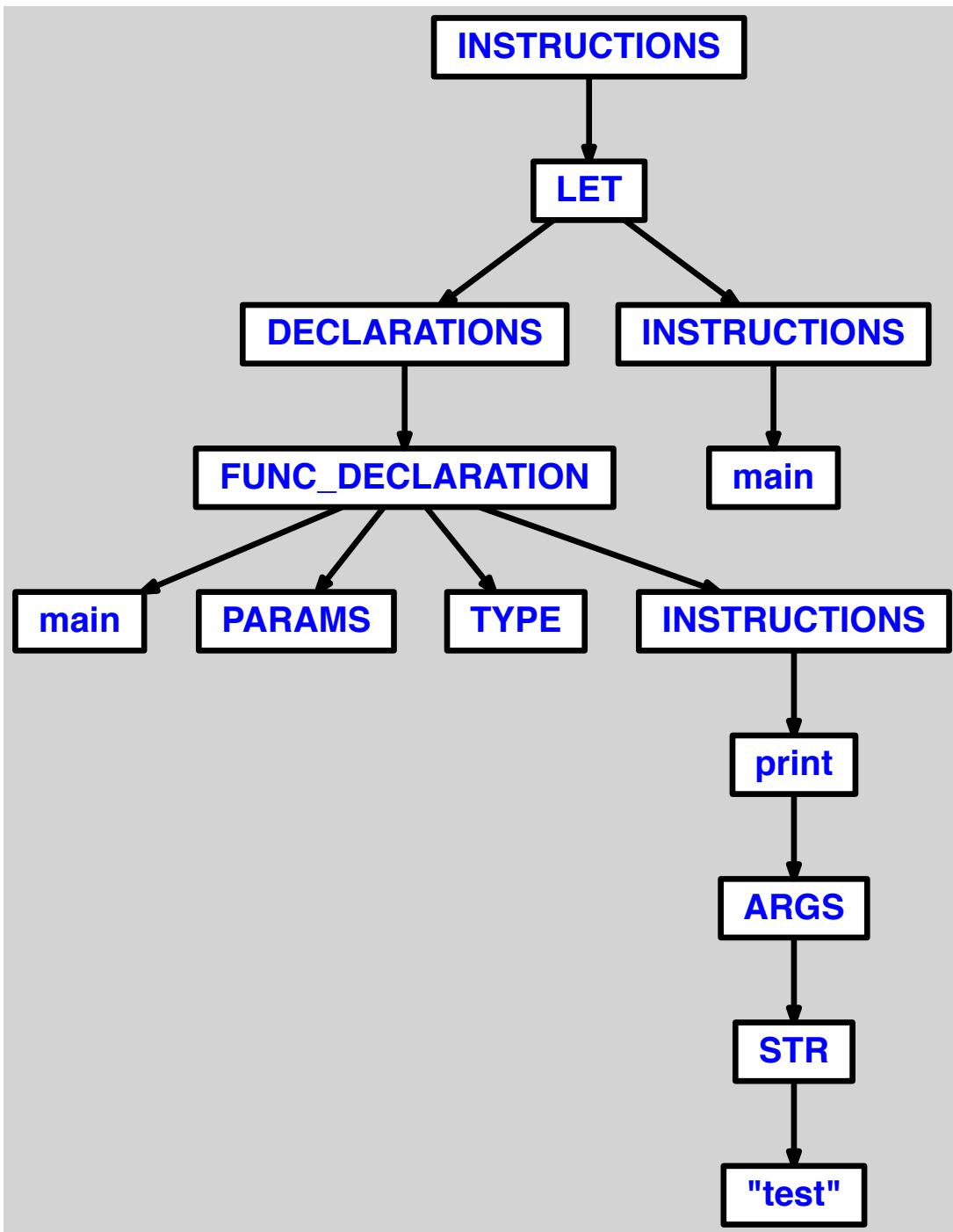
#### 4.2.8 2 commentaires, apres instructions, sur plusieurs lignes

```
let
    function main() =
        print("test");
        /*
        1
        2
        3
        */
        print("test")
        /*
        4
        5
        6
        */
in main() end
```



#### 4.2.9 3 commentaires, sans instruction avant, sur 1 ligne

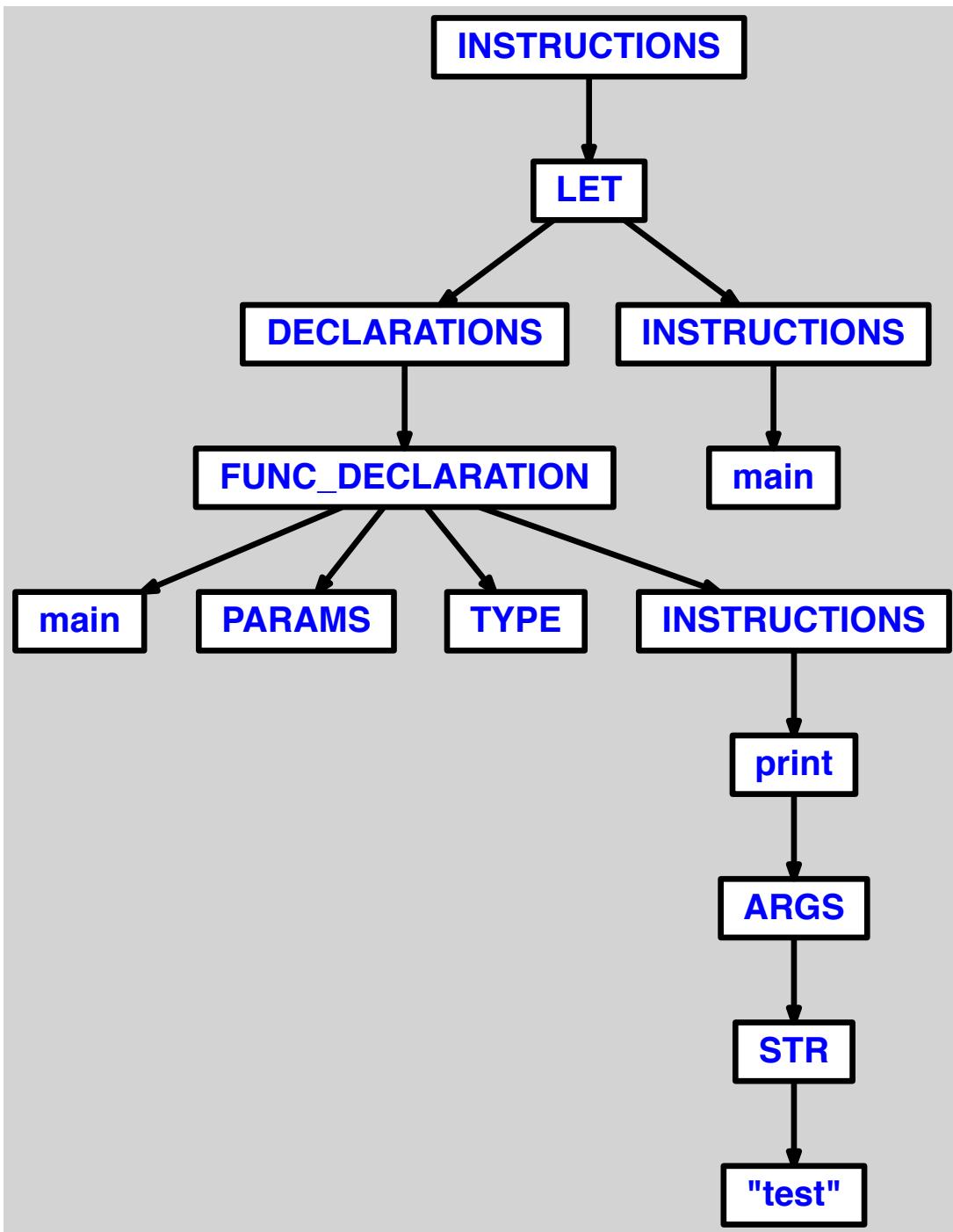
```
/* commentaire 1 */  
/* commentaire 2 */  
/* commentaire 3 */  
  
let  
    function main() = print("test")  
in main() end
```



#### 4.2.10 3 commentaires, sans instruction avant, sur plusieurs lignes

```
/*
1
```

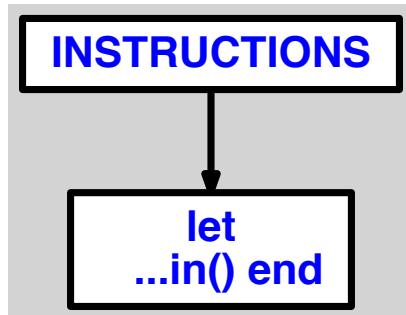
```
2
3
*/
/*
4
5
6
*/
/*
7
8
9
*/
let
    function main() = print("test")
in main() end
```



#### 4.2.11 3 commentaires, apres instructions, sur 1 ligne

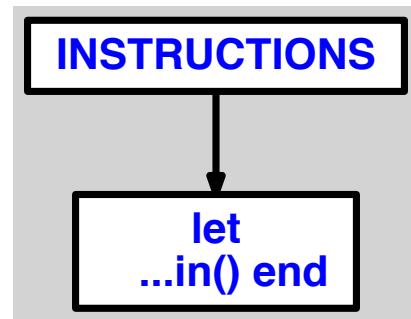
```
let
function main() =
```

```
print("test");
/* commentaire 1 */
print("test");
/* commentaire 2 */
print("test")
/* commentaire 3 */
in main() end
```



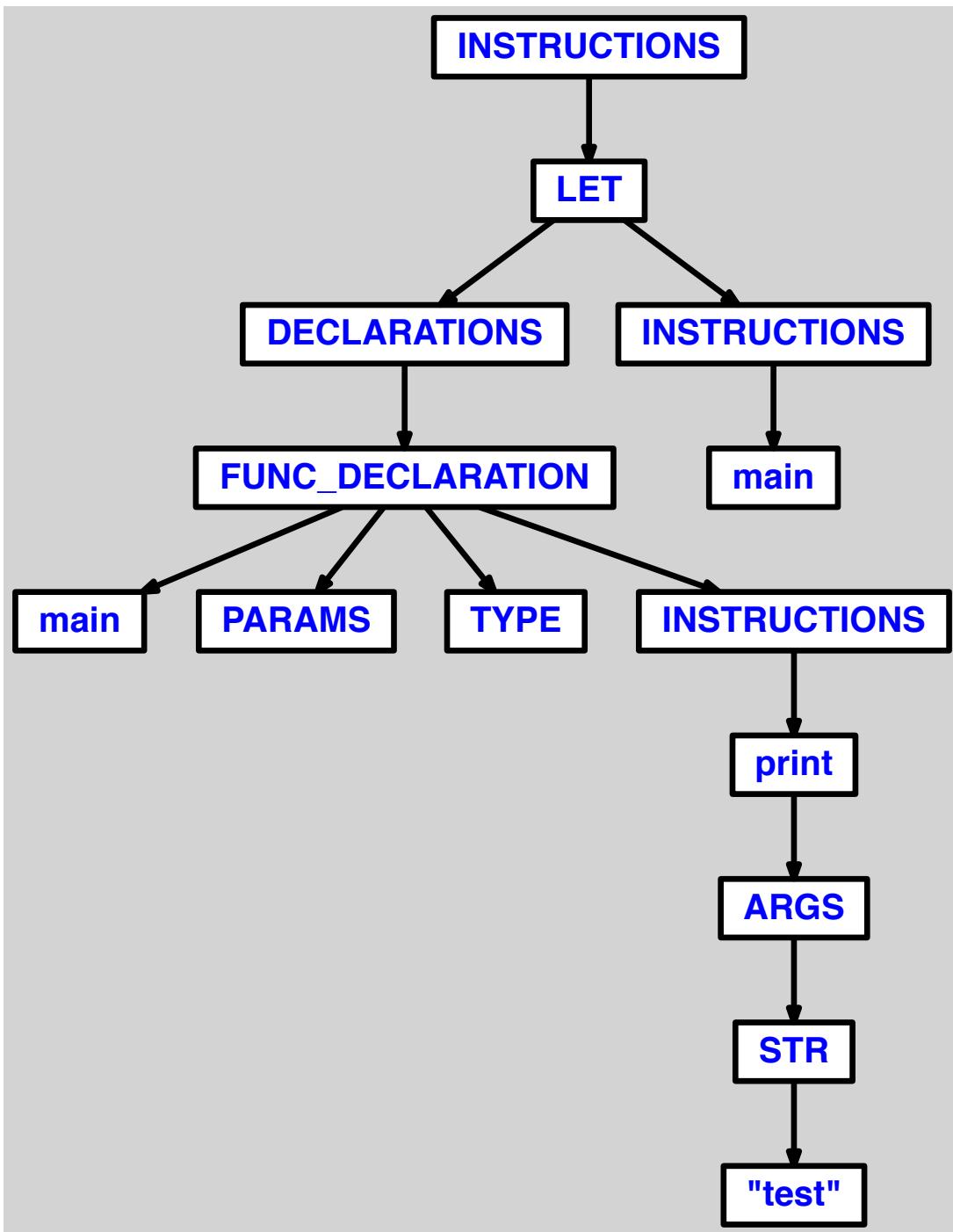
#### 4.2.12 3 commentaires, apres instructions, sur plusieurs lignes

```
let
    function main() =
        print("test");
        /*
        1
        2
        3
        */
        print("test");
        /*
        4
        5
        6
        */
        print("test")
        /*
        7
        8
        9
        */
in main() end
```



#### 4.2.13 commentaire sans contenu sur 1 ligne

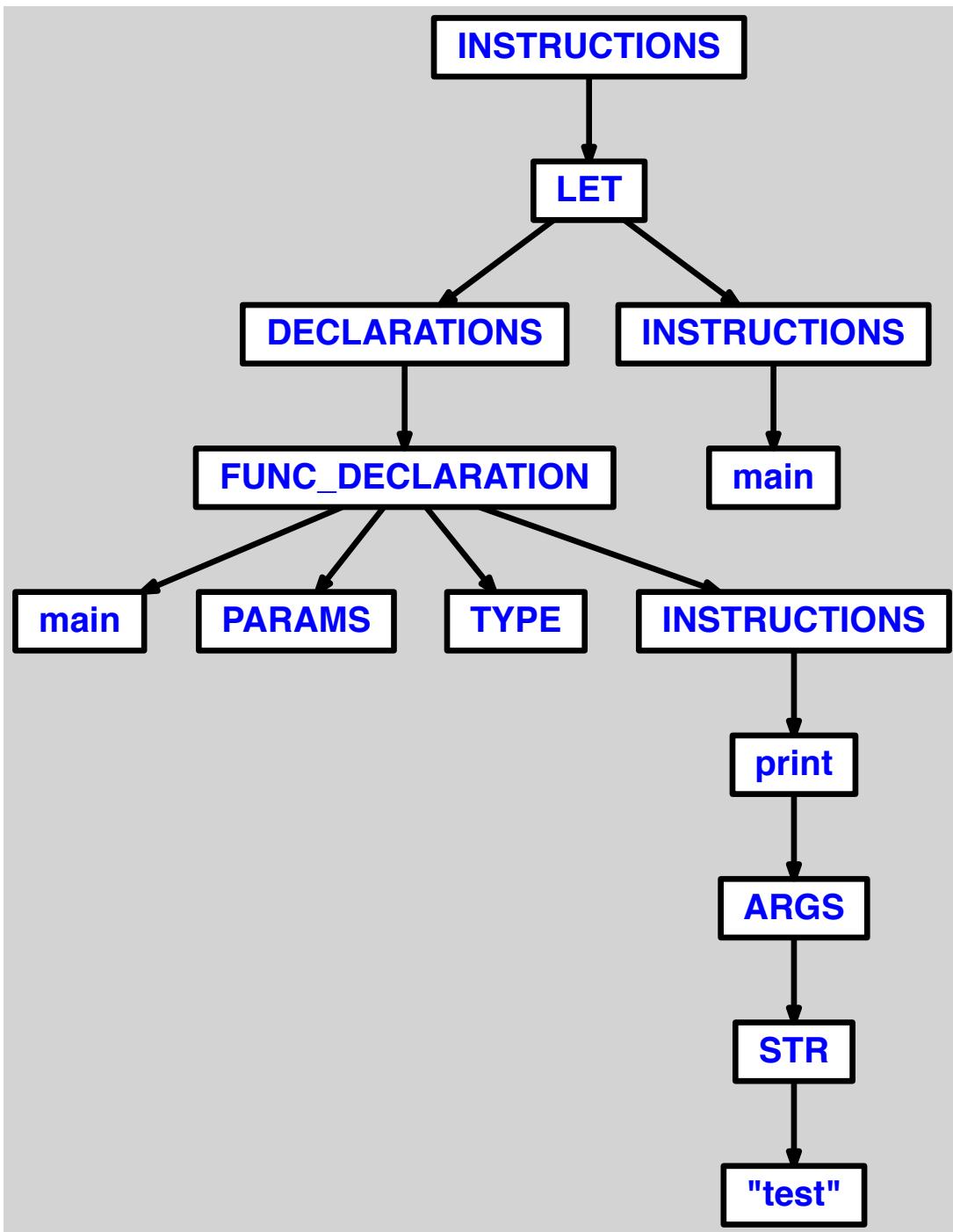
```
/* */  
  
let  
    function main() = print("test")  
in main() end
```



#### 4.2.14 commentaire sans contenu sur plusieurs lignes

/\*

```
*/  
let  
    function main() = print("test")  
in main() end
```



## 5 comparison

### 5.1 KO

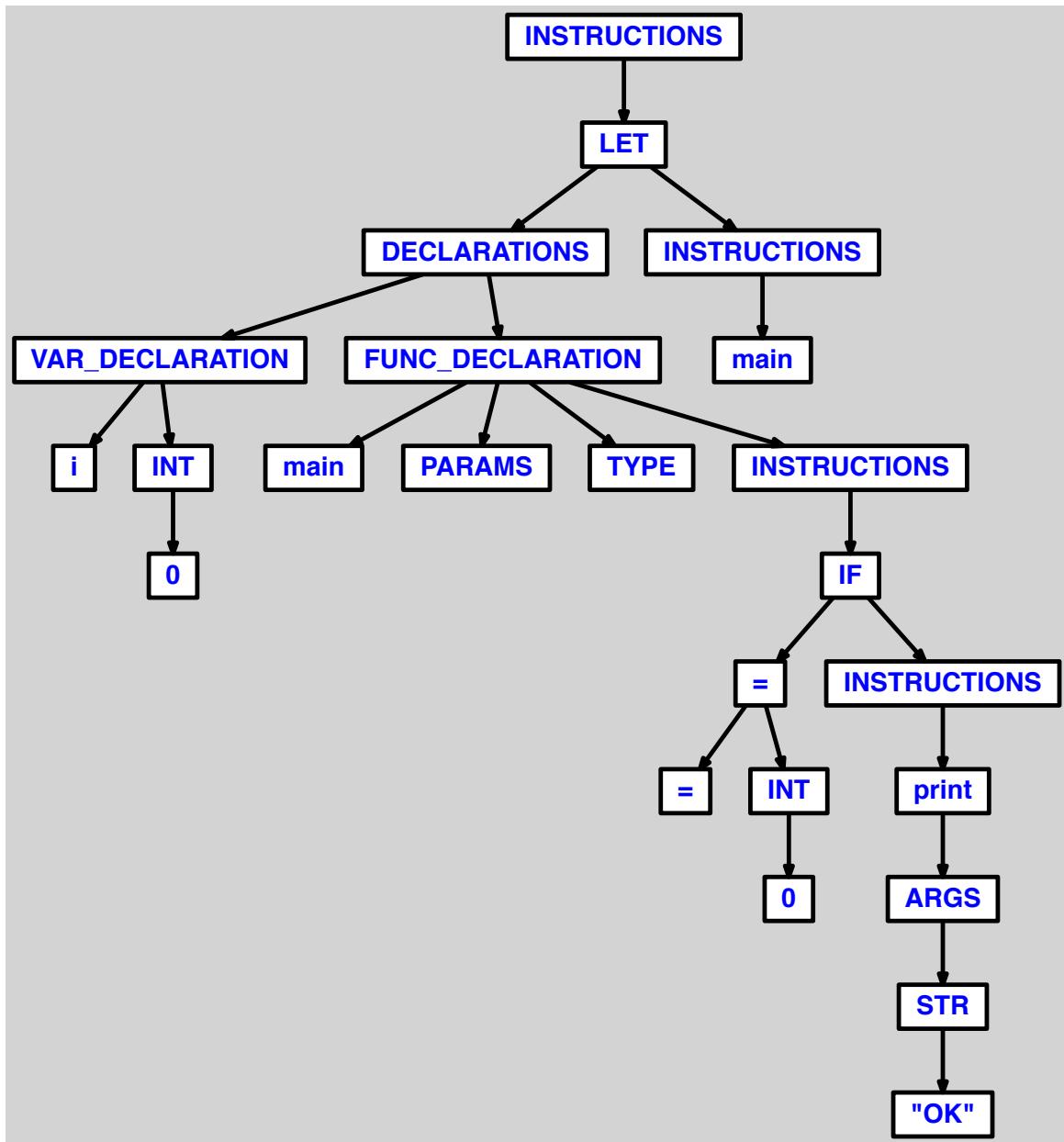
#### 5.1.1 comparaison avec oubli d'opérande gauche

let

    var i := 0

    function main() =  
        if = 0 then print("OK")

in main() end



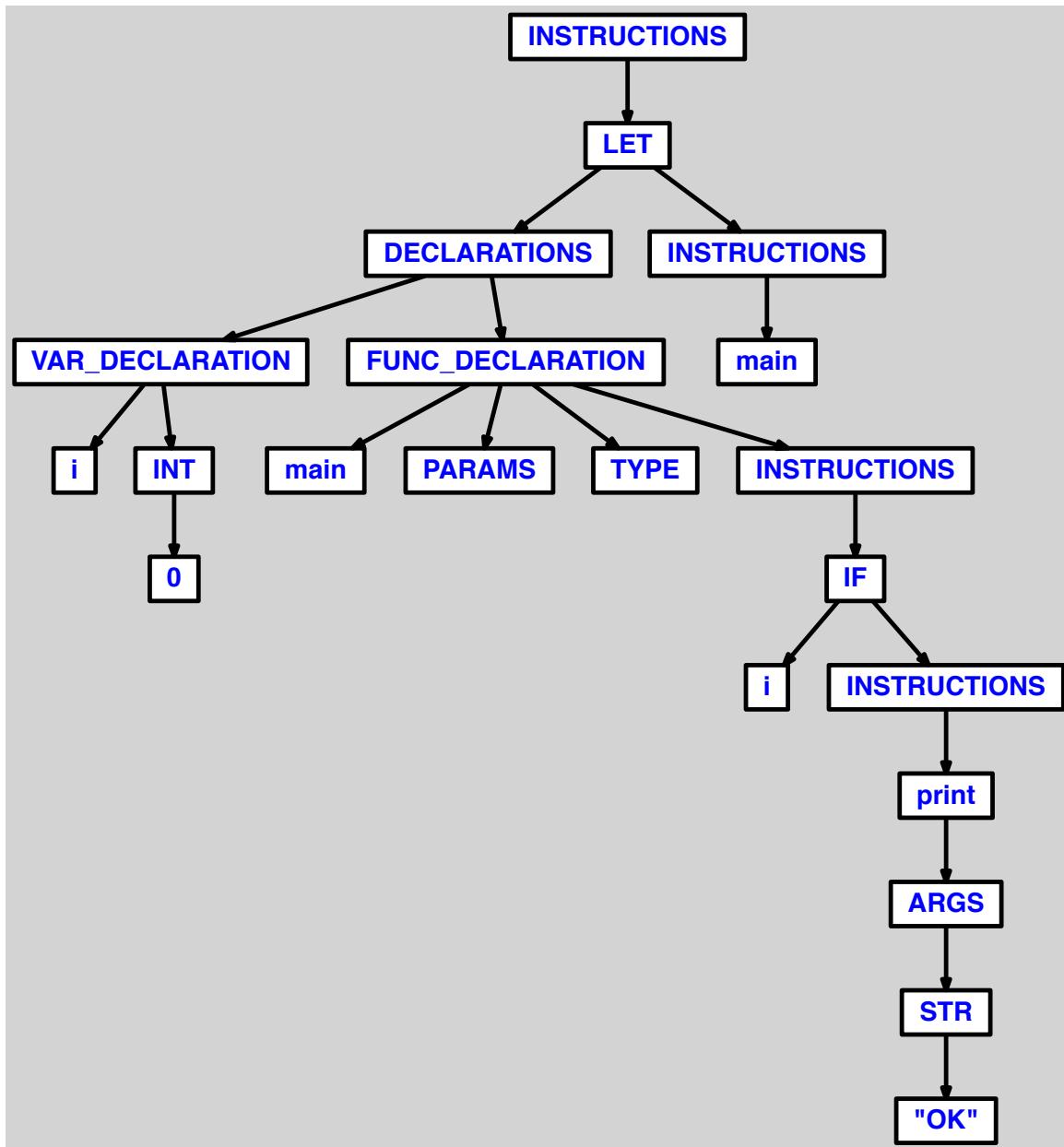
### 5.1.2 comparaison avec oubli d'opérateur

let

```
var i := 0
```

```
function main() =  
    if i 0 then print("OK")
```

```
in main() end
```

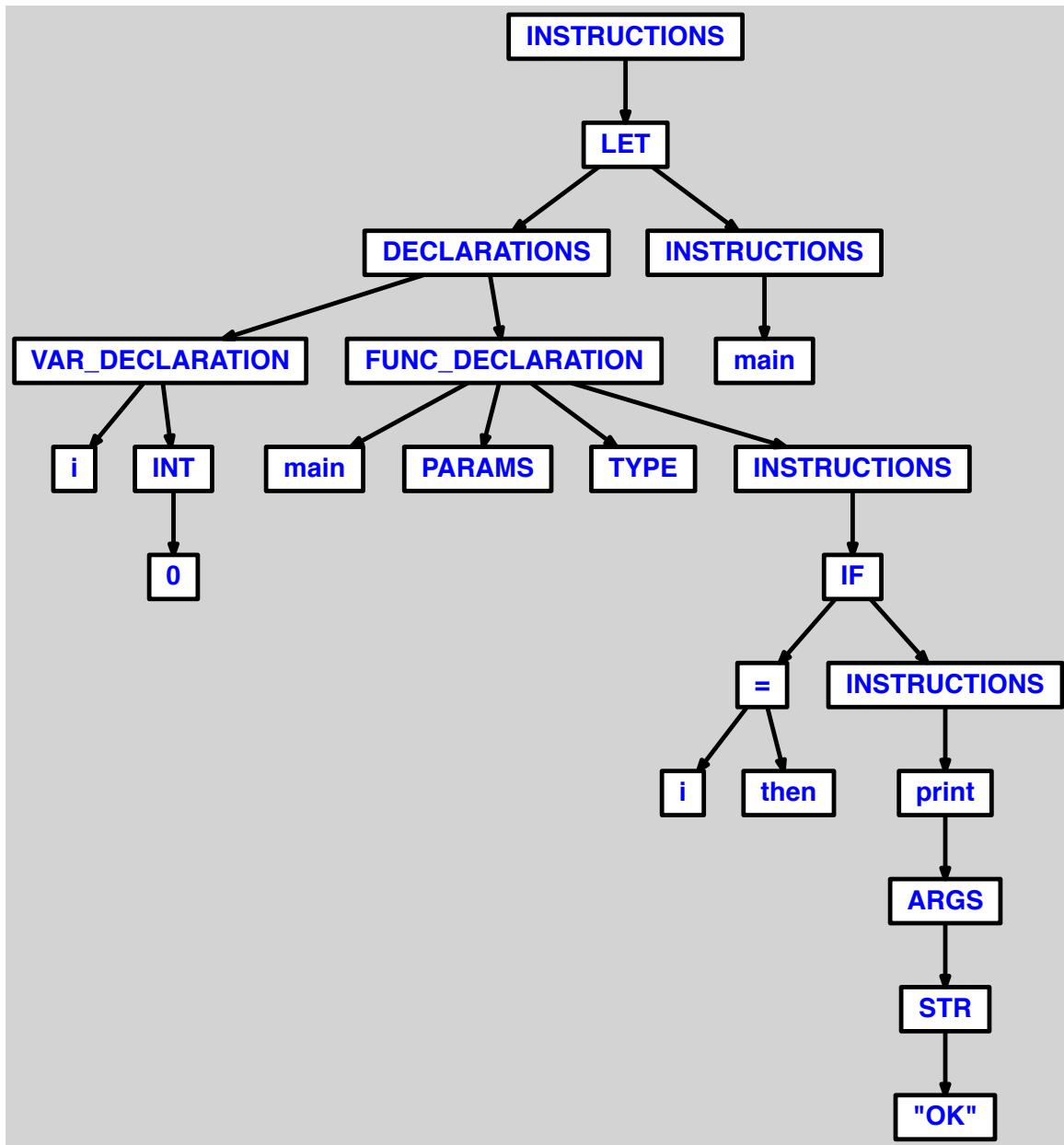


### 5.1.3 comparaison avec oubli d'opérateur droit

let

```
var i := 0

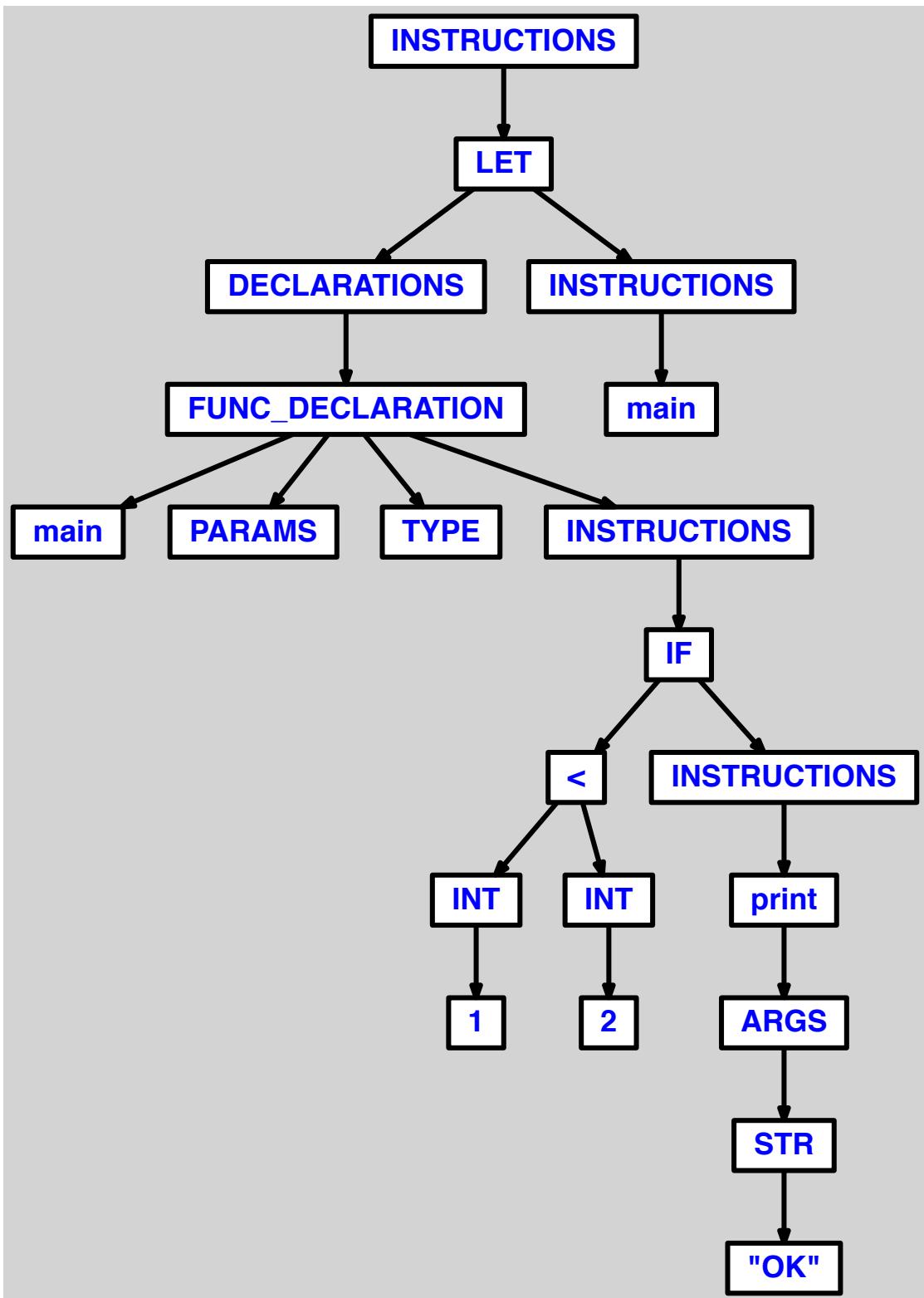
function main() =
    if i = then print("OK")
in main() end
```



## 5.2 OK

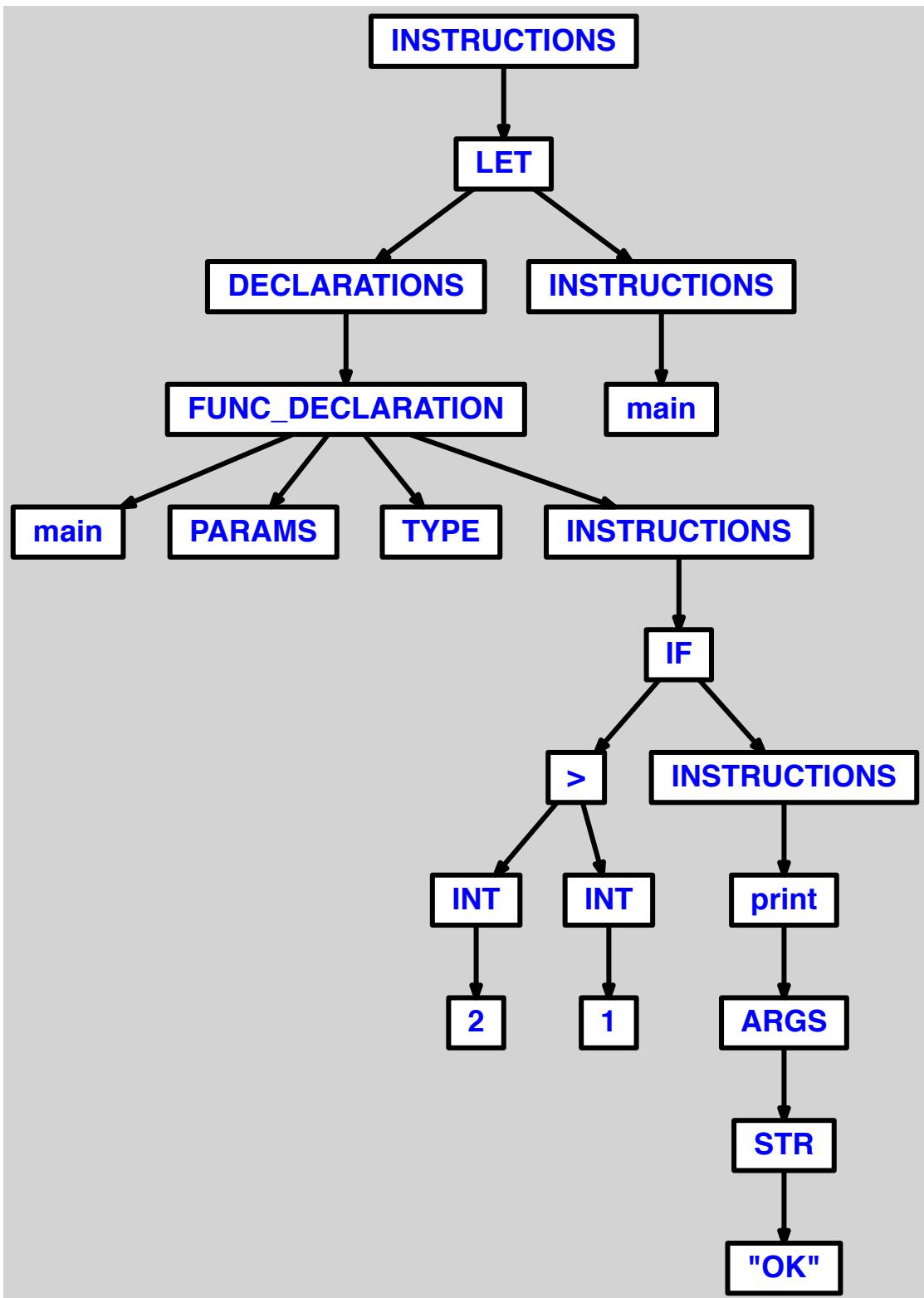
### 5.2.1 comparaison simple d'entiers avec """

```
let
    function main() =
        if 1 < 2 then print("OK")
in main() end
```



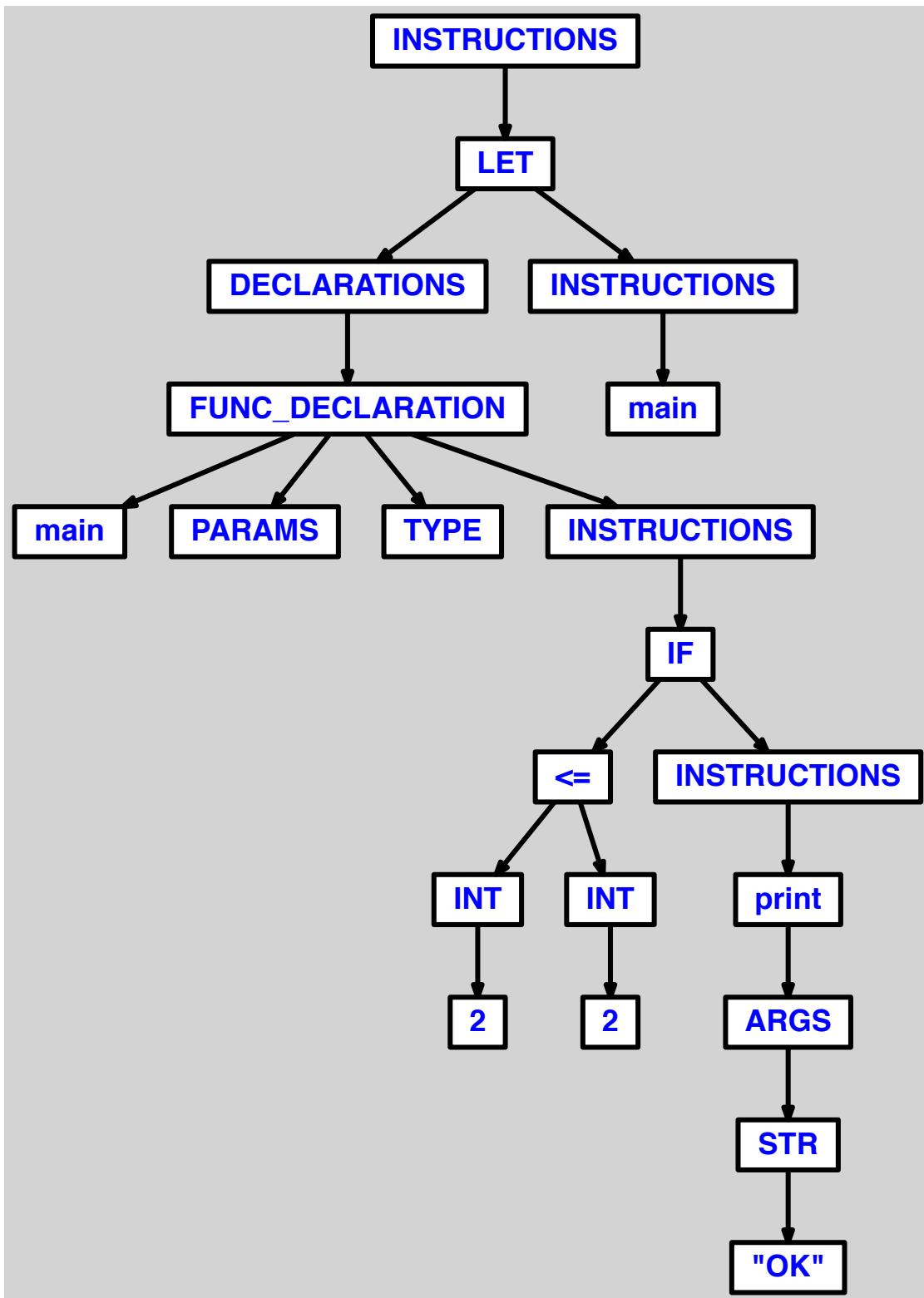
### 5.2.2 comparaison simple d'entiers avec ""

```
let
    function main() =
        if 2 > 1 then print("OK")
in main() end
```



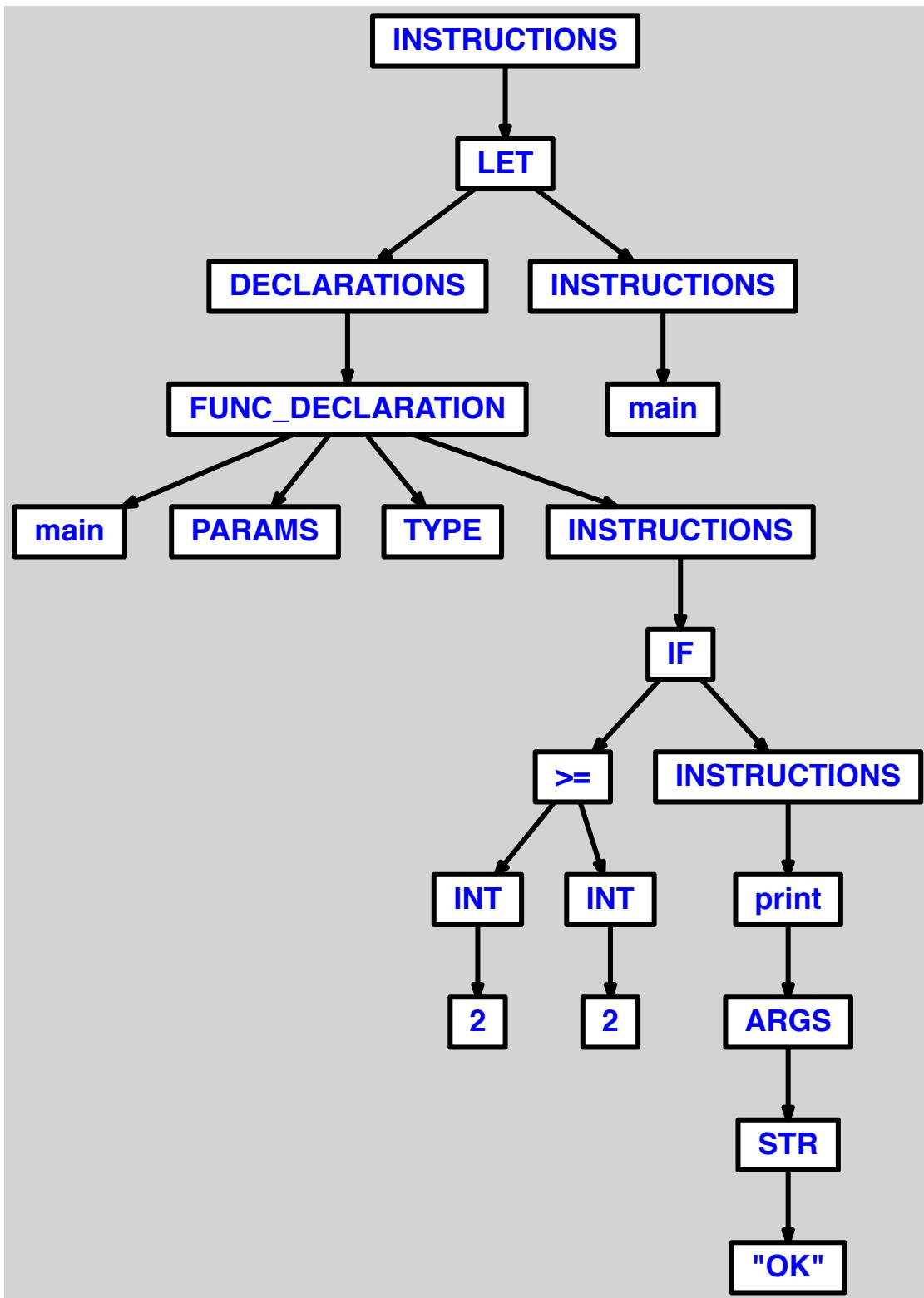
### 5.2.3 comparaison simple d'entiers avec "=="

```
let
    function main() =
        if 2 <= 2 then print("OK")
in main() end
```



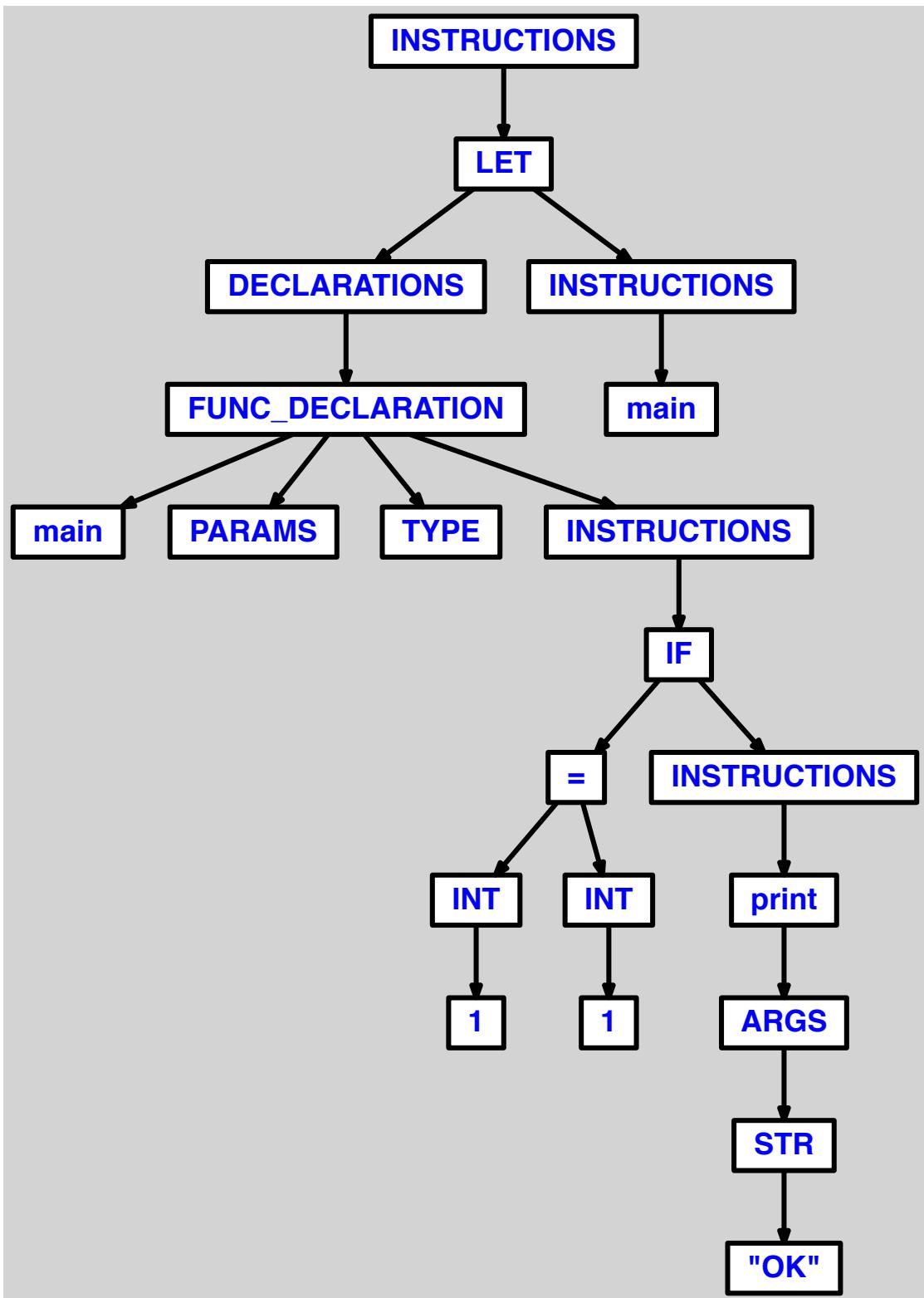
#### 5.2.4 comparaison simple d'entiers avec "=="

```
let
    function main() =
        if 2 >= 2 then print("OK")
in main() end
```



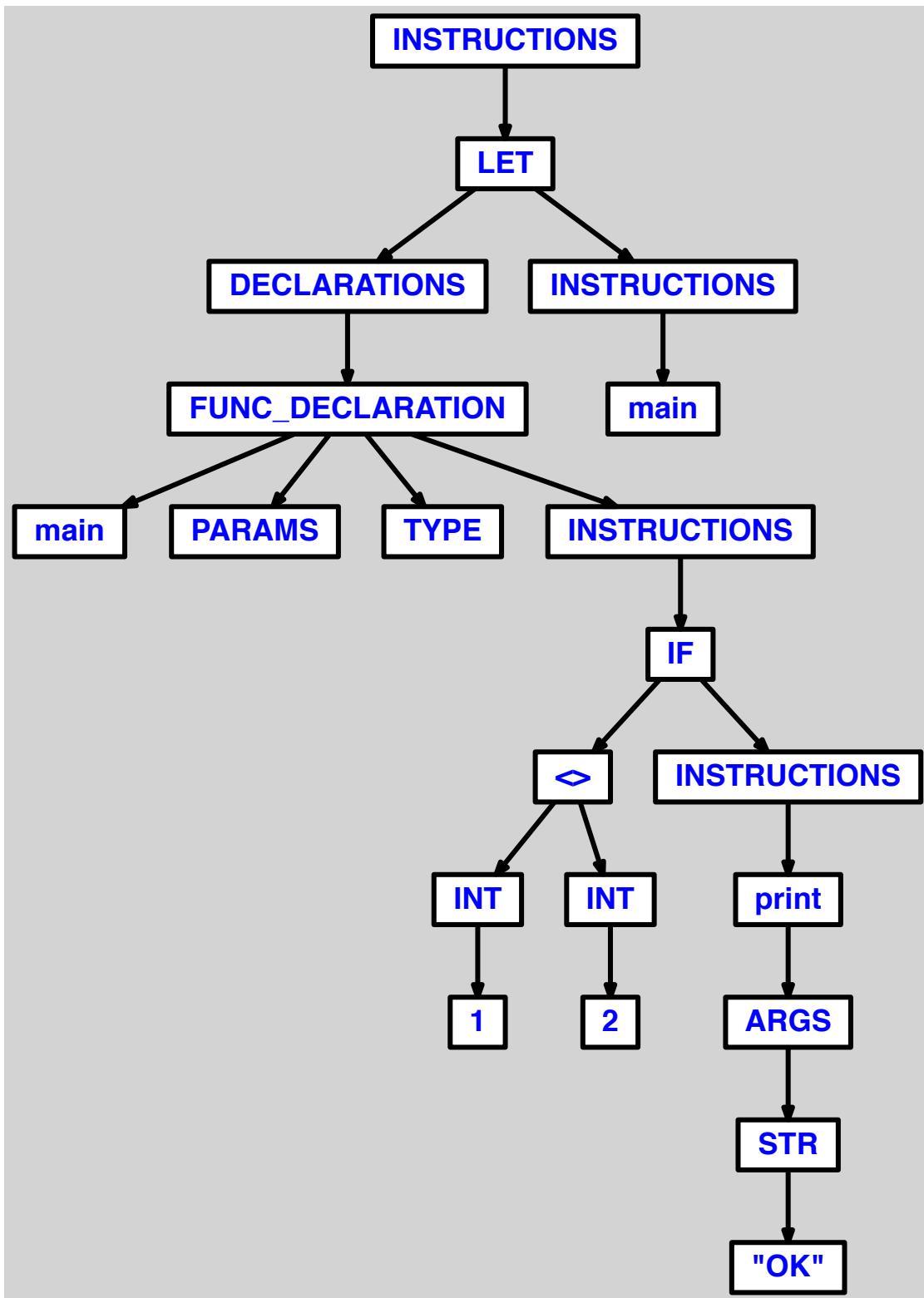
### 5.2.5 comparaison simple d'entiers avec =

```
let
    function main() =
        if 1 = 1 then print("OK")
in main() end
```



### 5.2.6 comparaison simple d'entiers avec ""

```
let
    function main() =
        if 1 <> 2 then print("OK")
in main() end
```

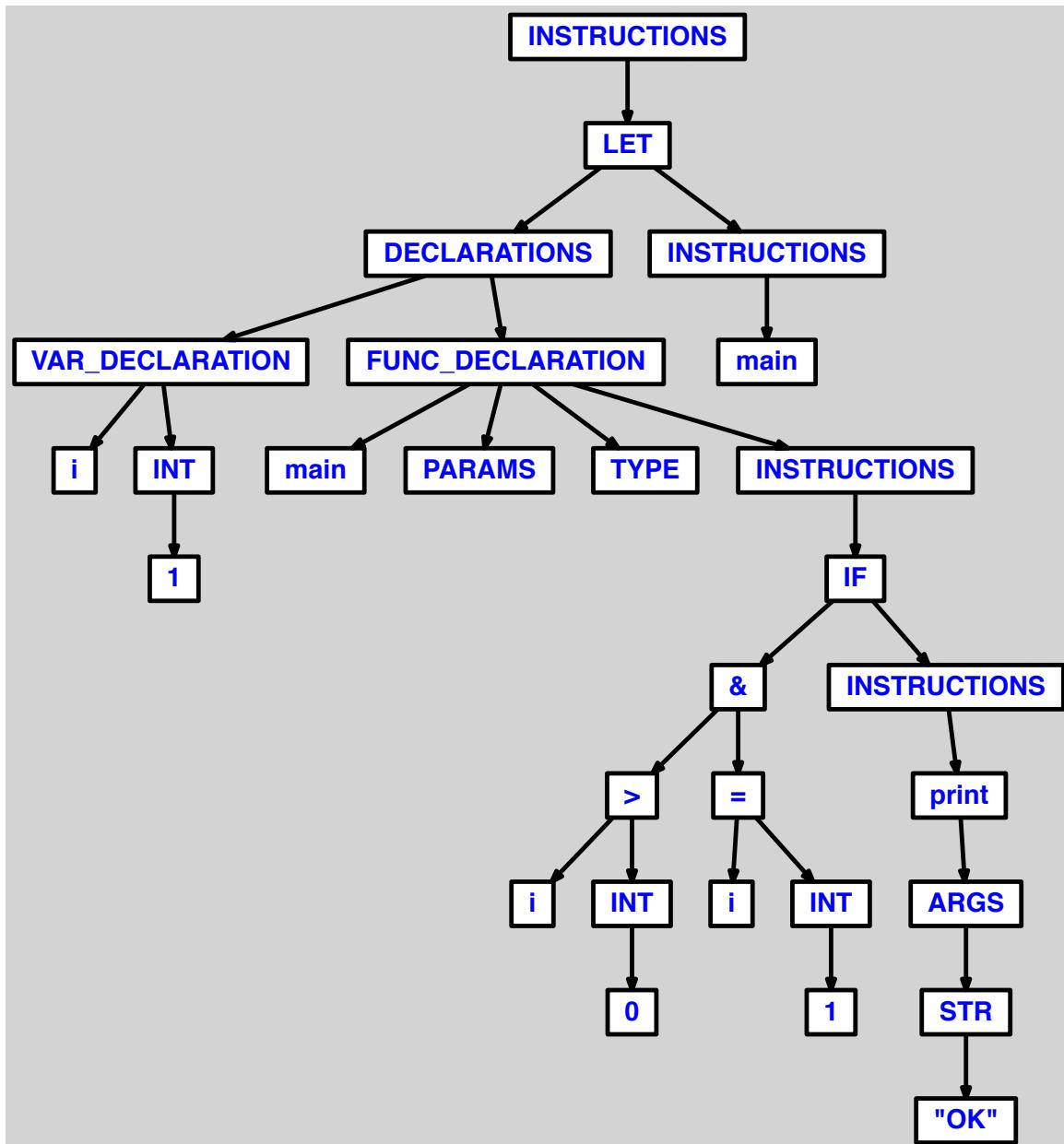


### 5.2.7 comparaison double d'entiers

let

```
var i := 1

function main() =
    if i > 0 & i = 1 then print("OK")
in main() end
```

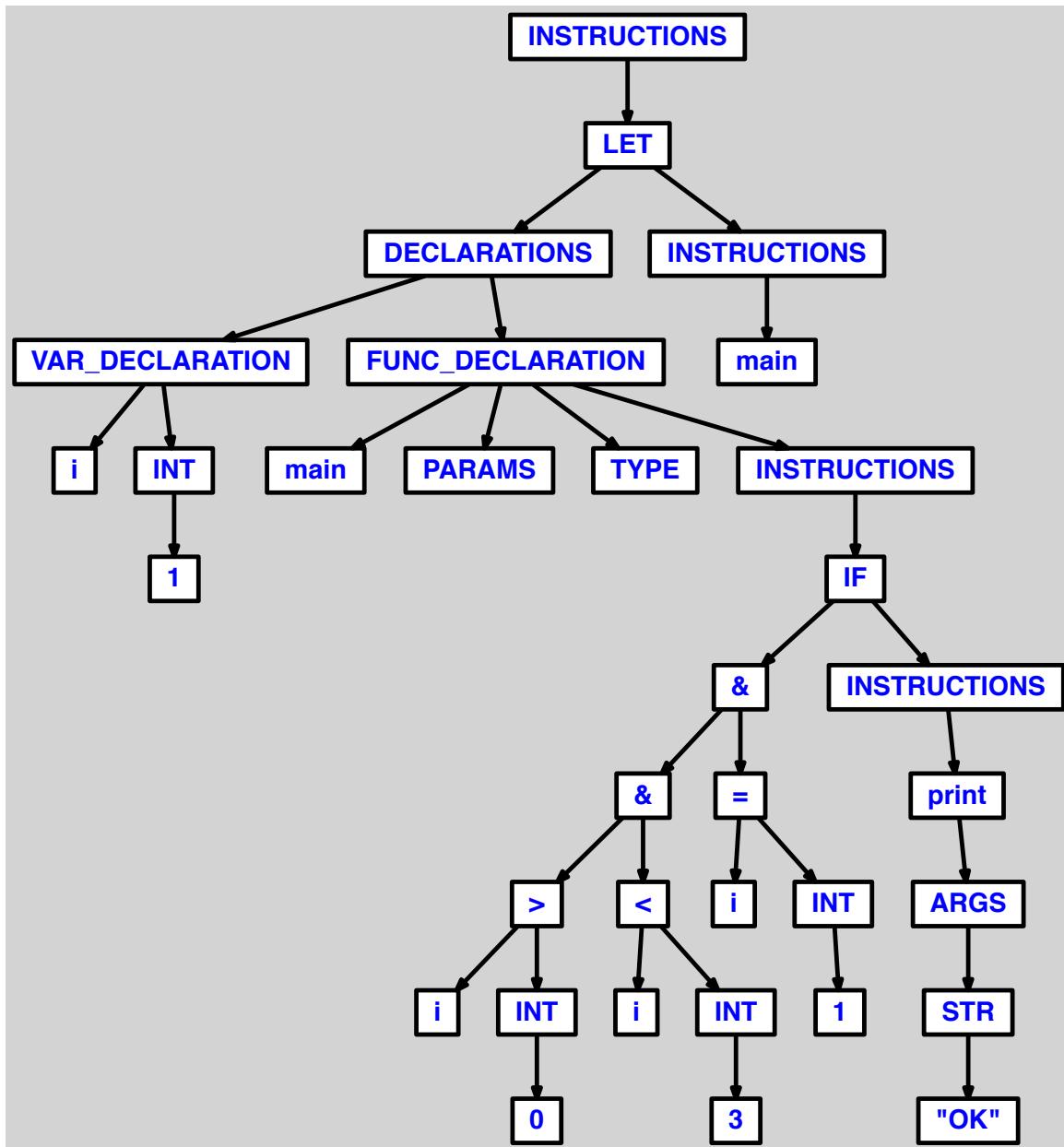


### 5.2.8 comparaison triple d'entiers

let

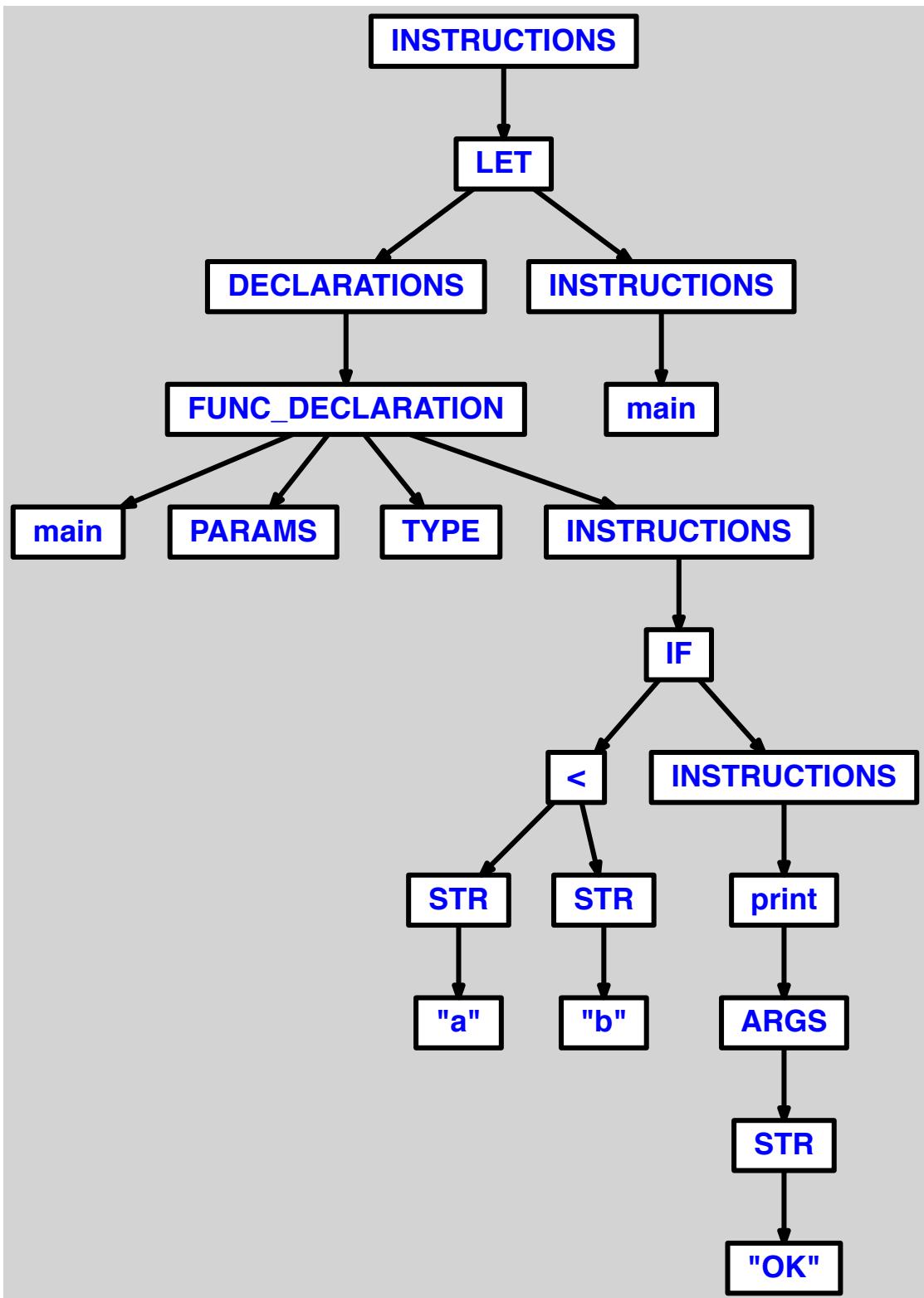
```
var i := 1

function main() =
    if i > 0 & i < 3 & i = 1 then print("OK")
in main() end
```



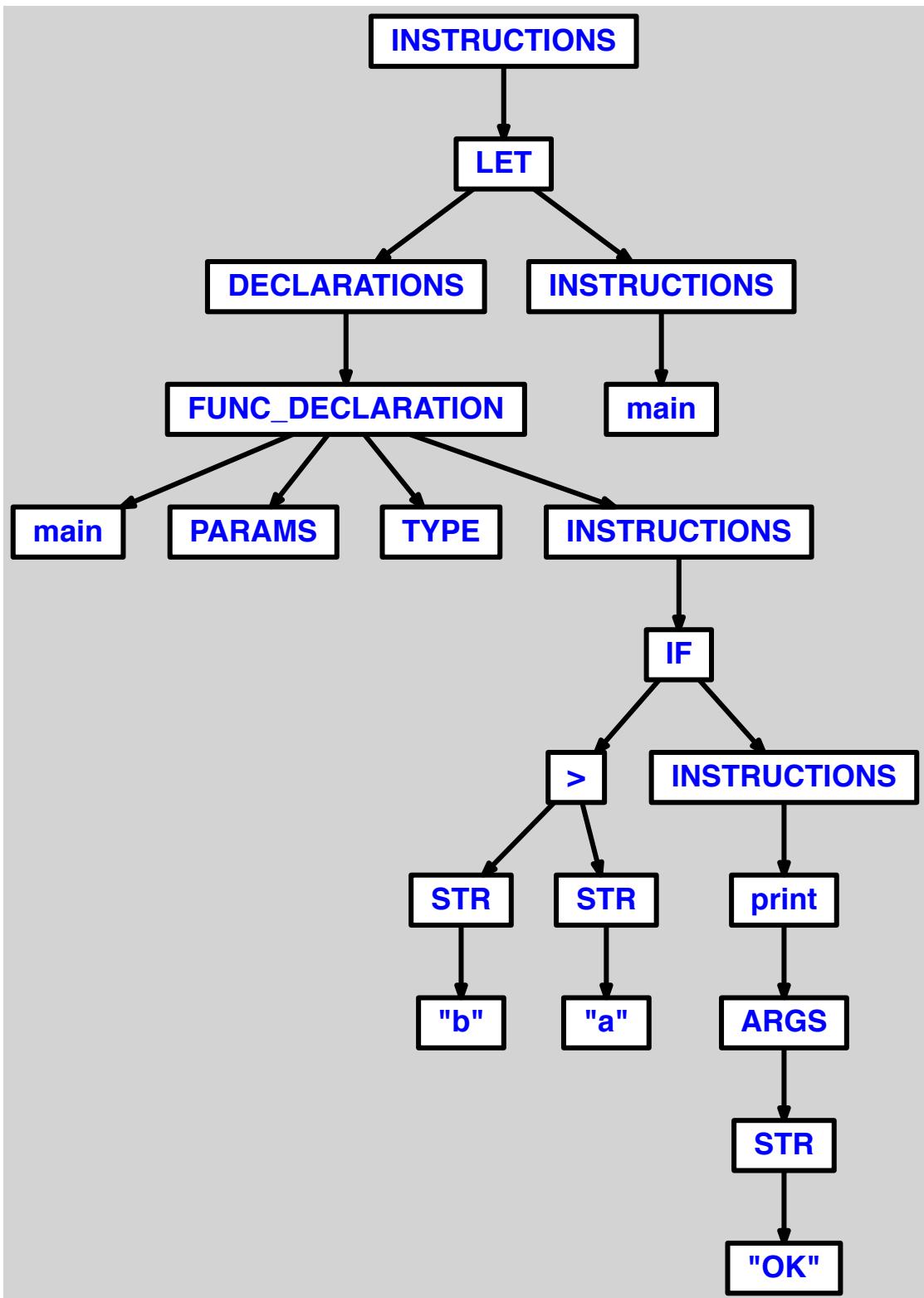
### 5.2.9 comparaison simple de chaines avec ""

```
let
    function main() =
        if "a" < "b" then print("OK")
in main() end
```



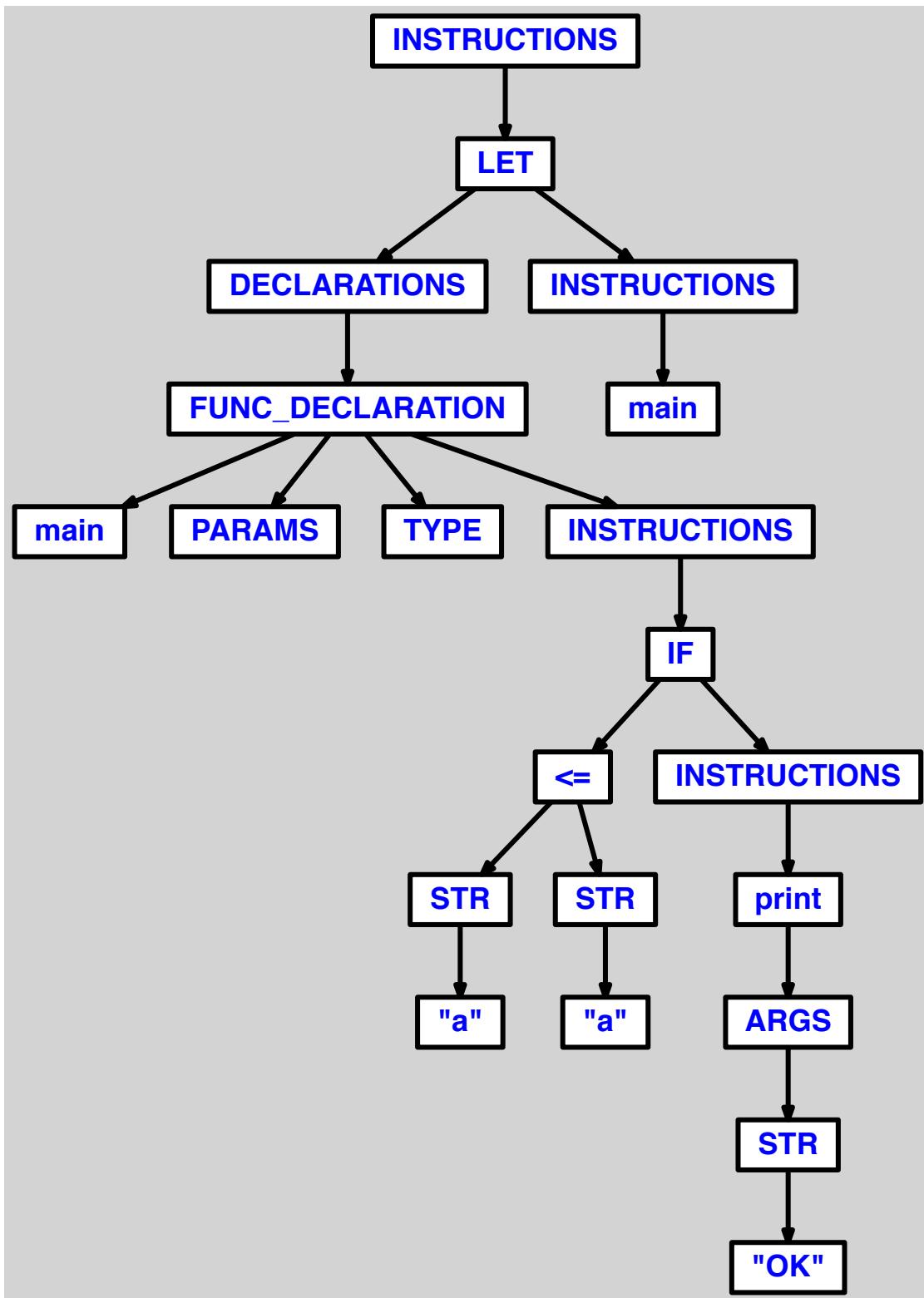
### 5.2.10 comparaison simple de chaines avec ""

```
let
    function main() =
        if "b" > "a" then print("OK")
in main() end
```



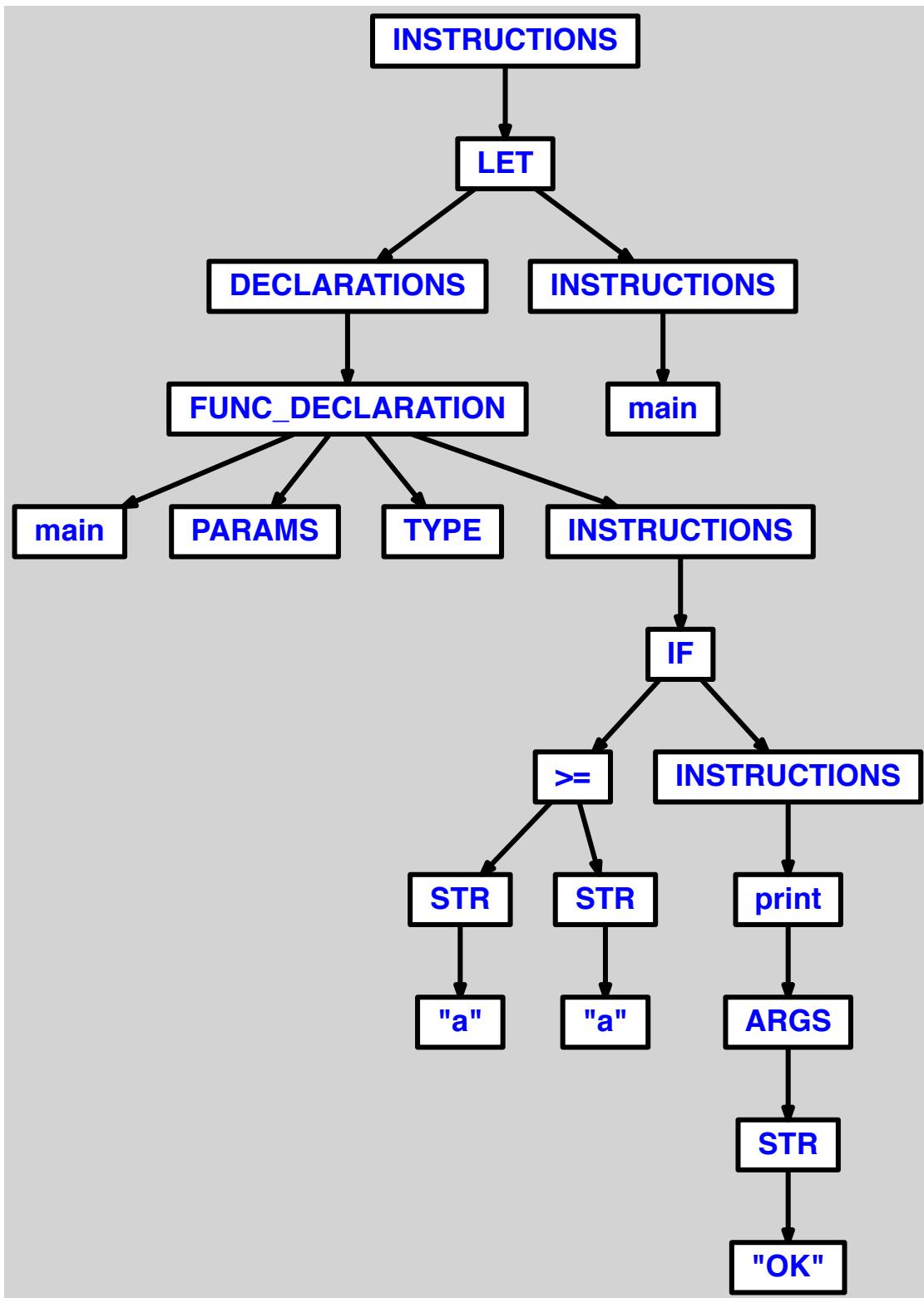
### 5.2.11 comparaison simple de chaines avec "=="

```
let
    function main() =
        if "a" <= "a" then print("OK")
in main() end
```



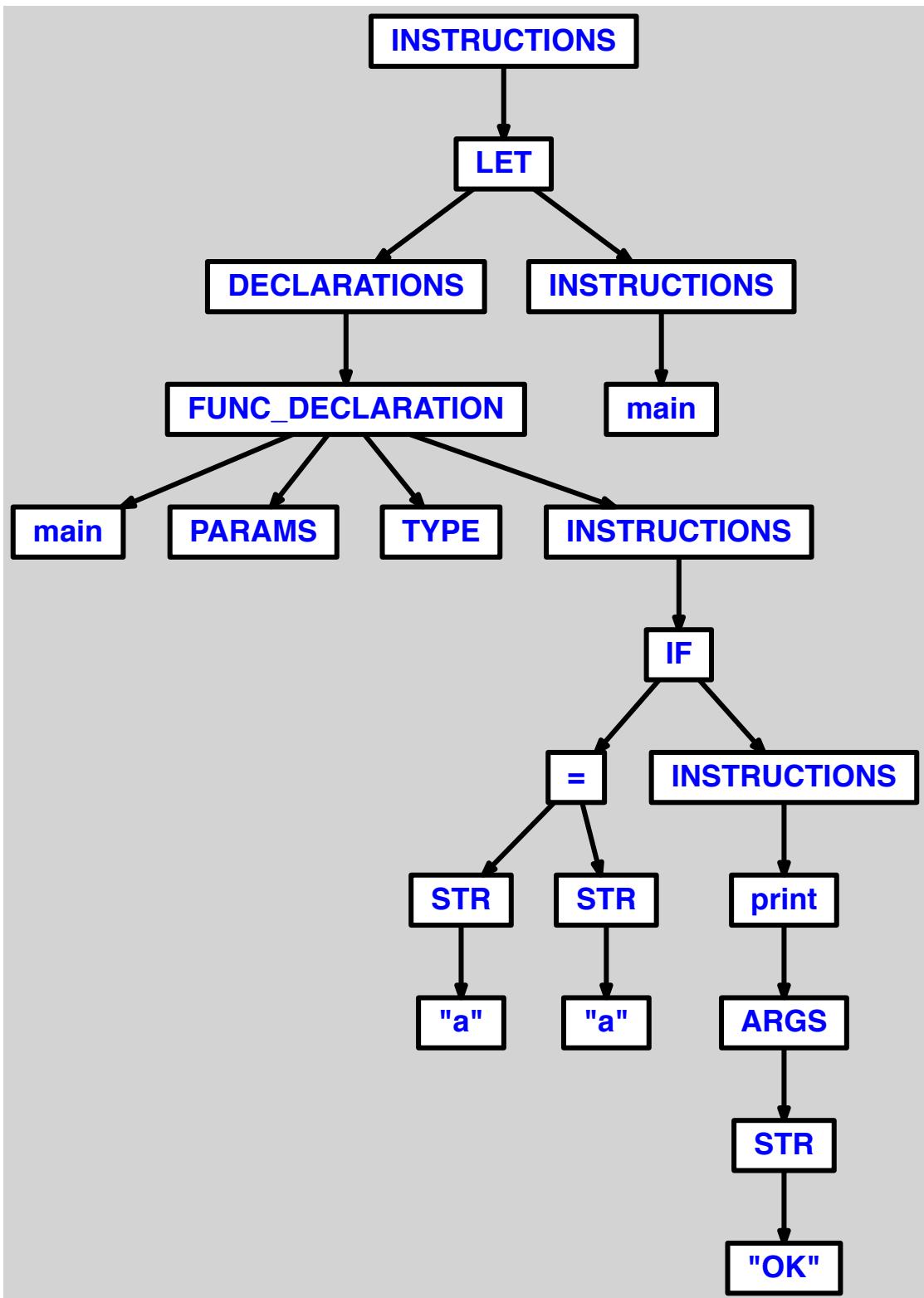
### 5.2.12 comparaison simple de chaines avec "=="

```
let
    function main() =
        if "a" >= "a" then print("OK")
in main() end
```



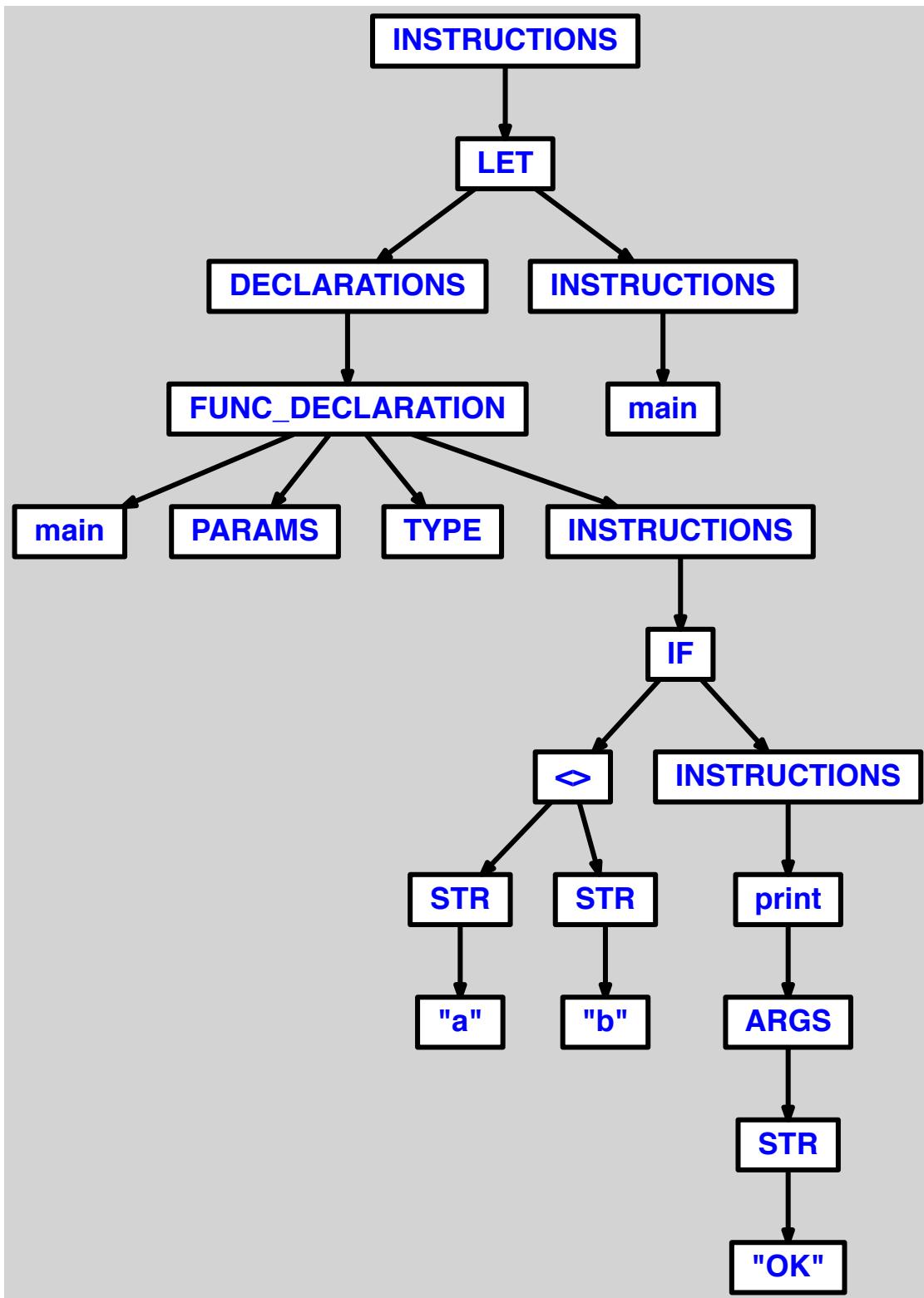
### 5.2.13 comparaison simple de chaines avec =

```
let
    function main() =
        if "a" = "a" then print("OK")
in main() end
```



#### 5.2.14 comparaison simple de chaines avec ""

```
let
    function main() =
        if "a" <> "b" then print("OK")
in main() end
```



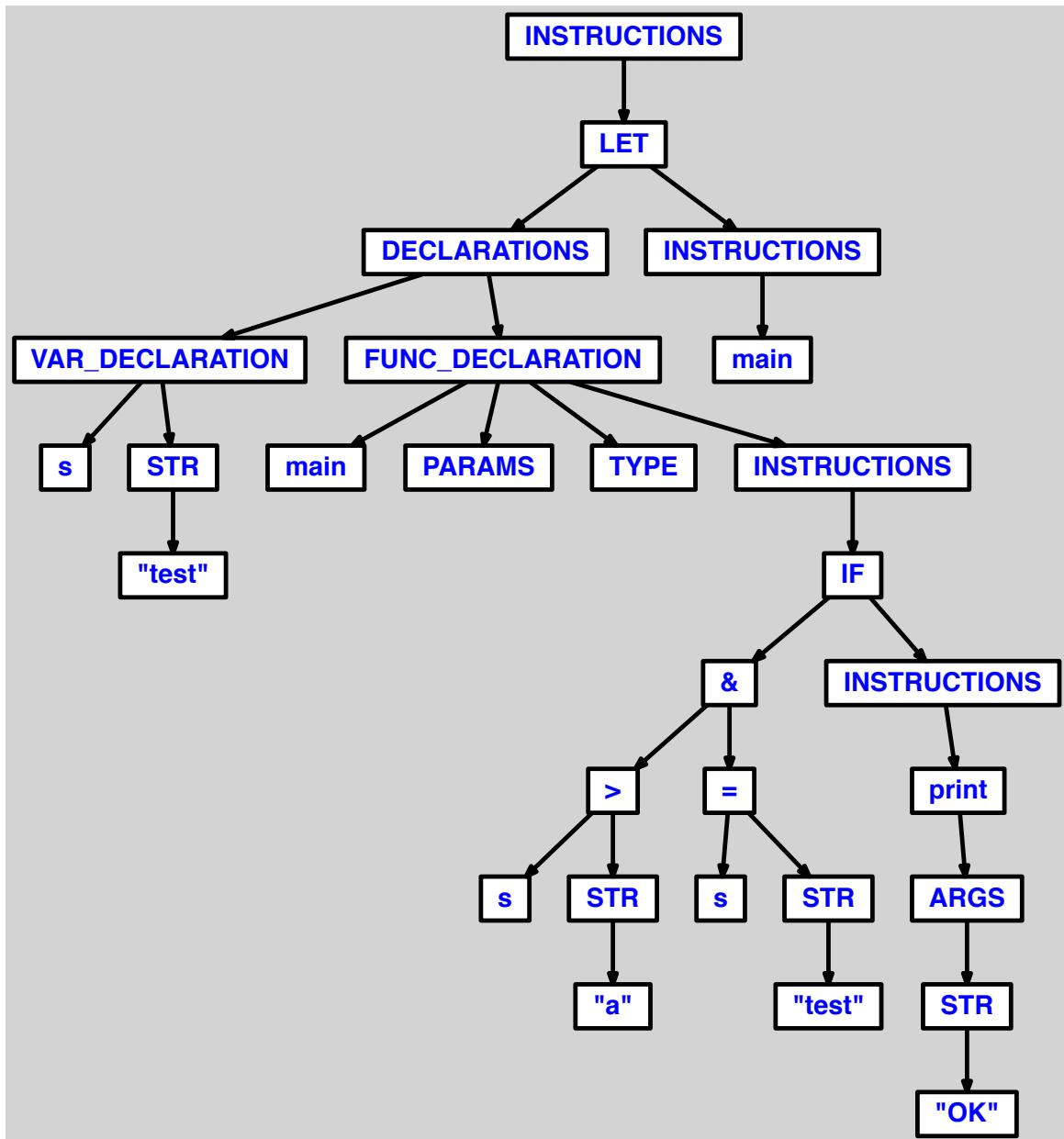
### 5.2.15 comparaison double de chaines

```

let
    var s := "test"

    function main() =
        if s > "a" & s = "test" then print("OK")
in main() end

```

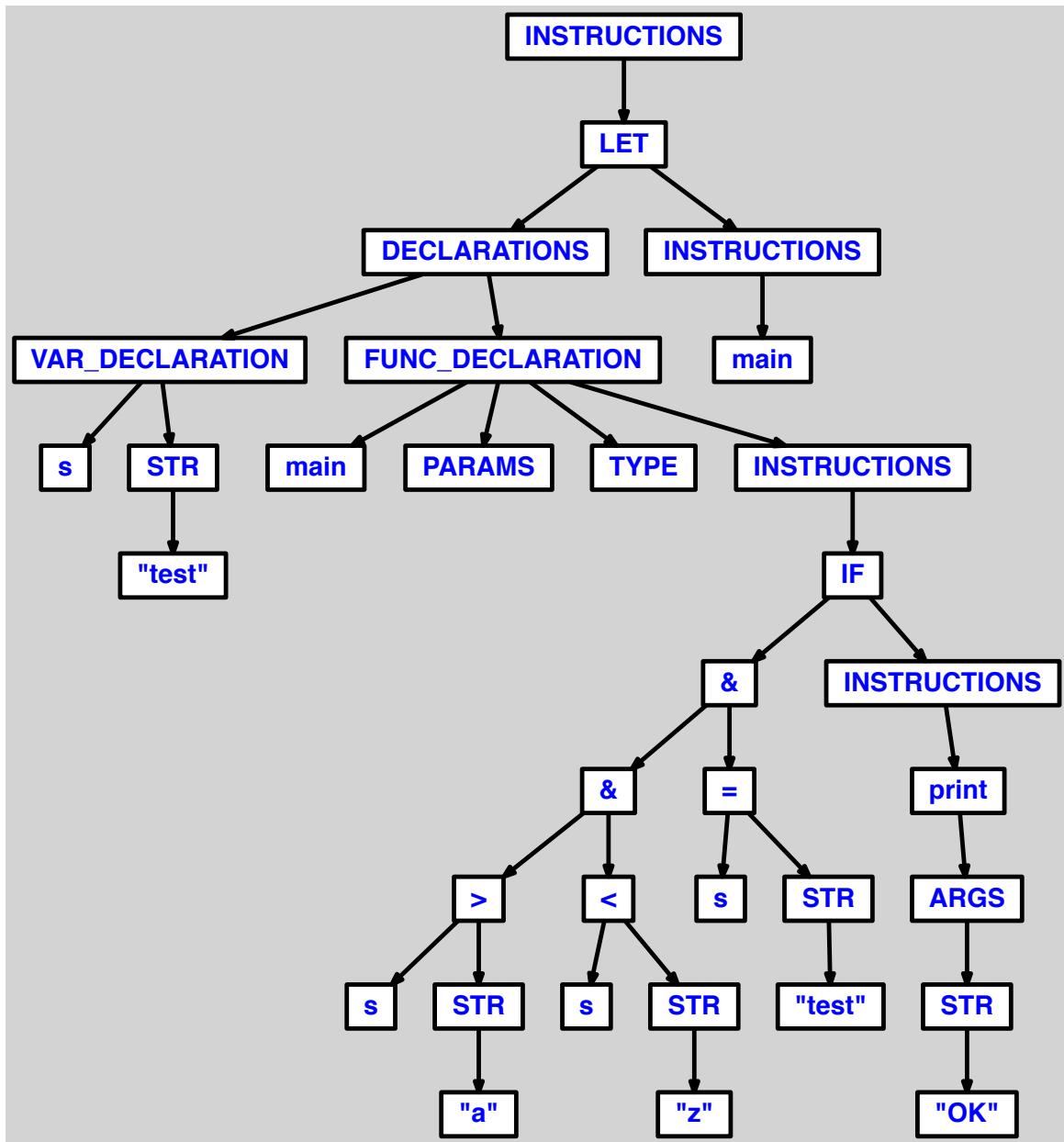


### 5.2.16 comparaison triple de chaines

let

```
var s := "test"

function main() =
    if s > "a" & s < "z" & s = "test" then print("OK")
in main() end
```



## 6 for

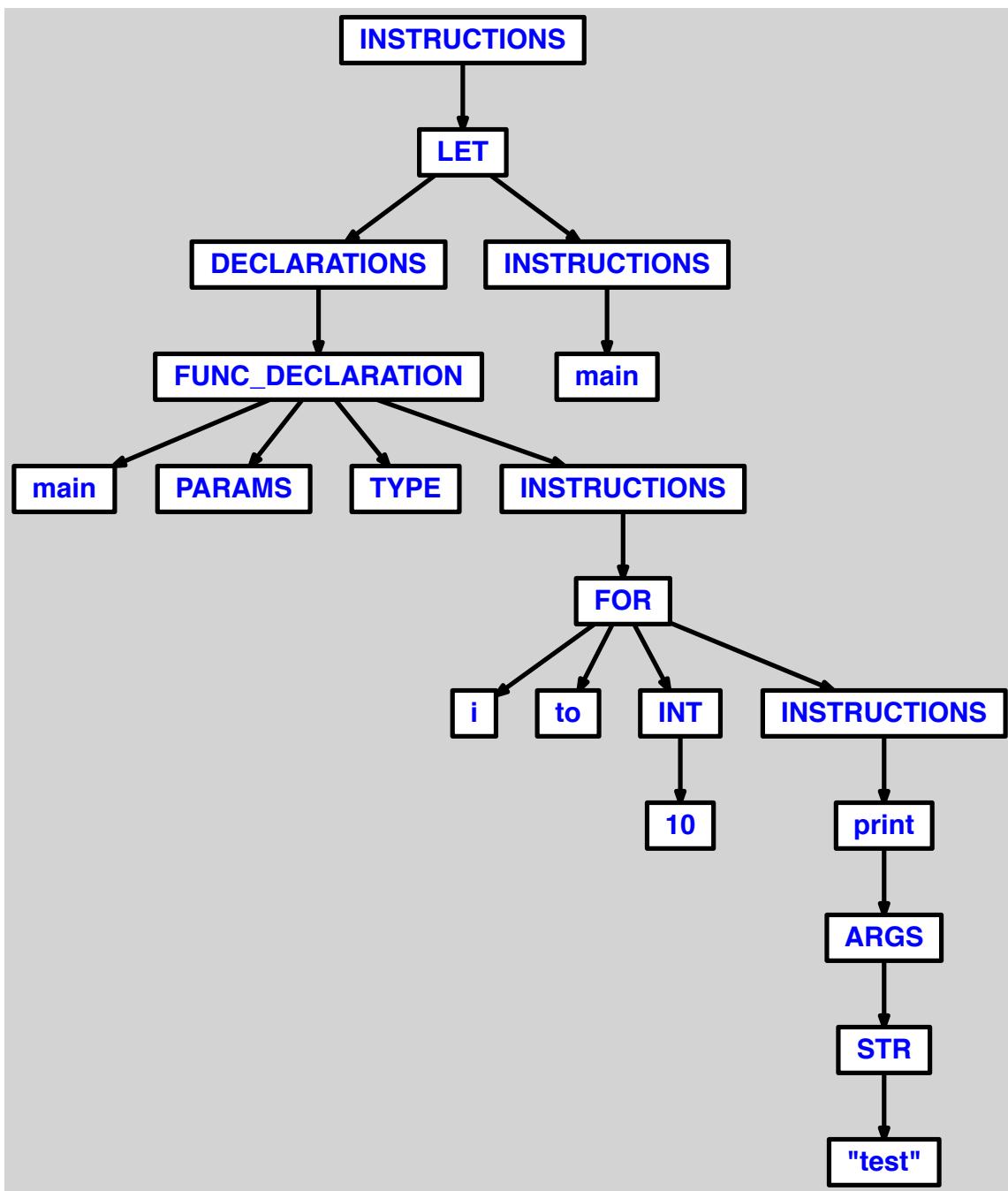
### 6.1 KO

#### 6.1.1 for avec affectation de chaine pour le compteur

```
let
```

```
    function main() =  
        for i := "3" to 10 do  
            print("test")
```

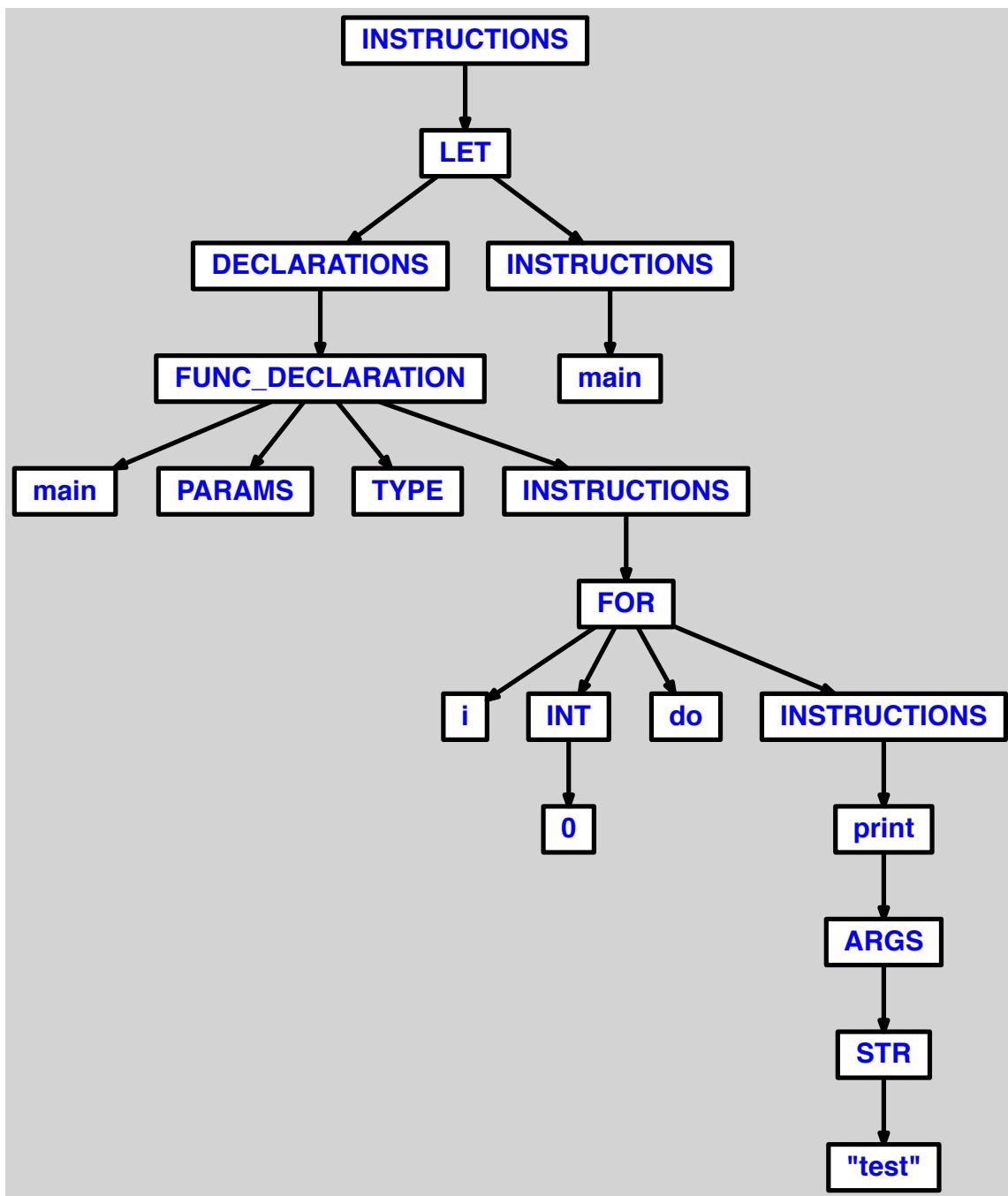
```
in main() end
```



#### 6.1.2 for avec affectation de chaine pour la borne superieure de l'iteration

```

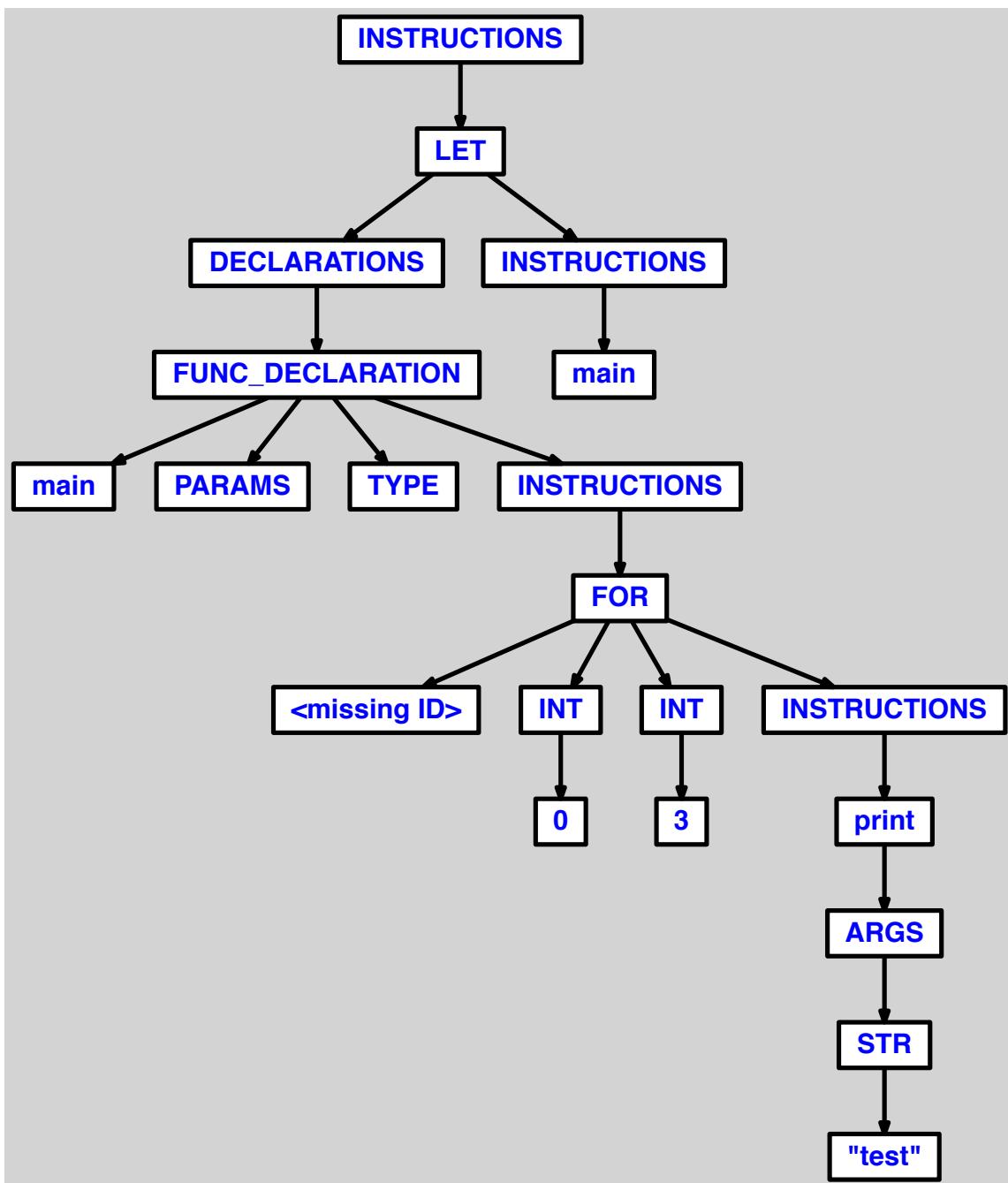
let
  function main() =
    for i := 0 to "3" do
      print("test")
in main() end
  
```



#### 6.1.3 for avec oubli de l'identifiant du compteur

```

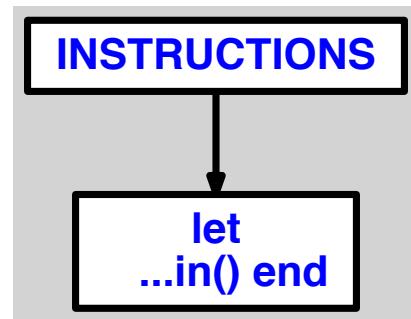
let
    function main() =
        for := 0 to 3 do
            print("test")
in main() end
  
```



#### 6.1.4 for avec oubli du for

```

let
    function main() =
        i := 0 to 3 do
            print("test")
in main() end
  
```

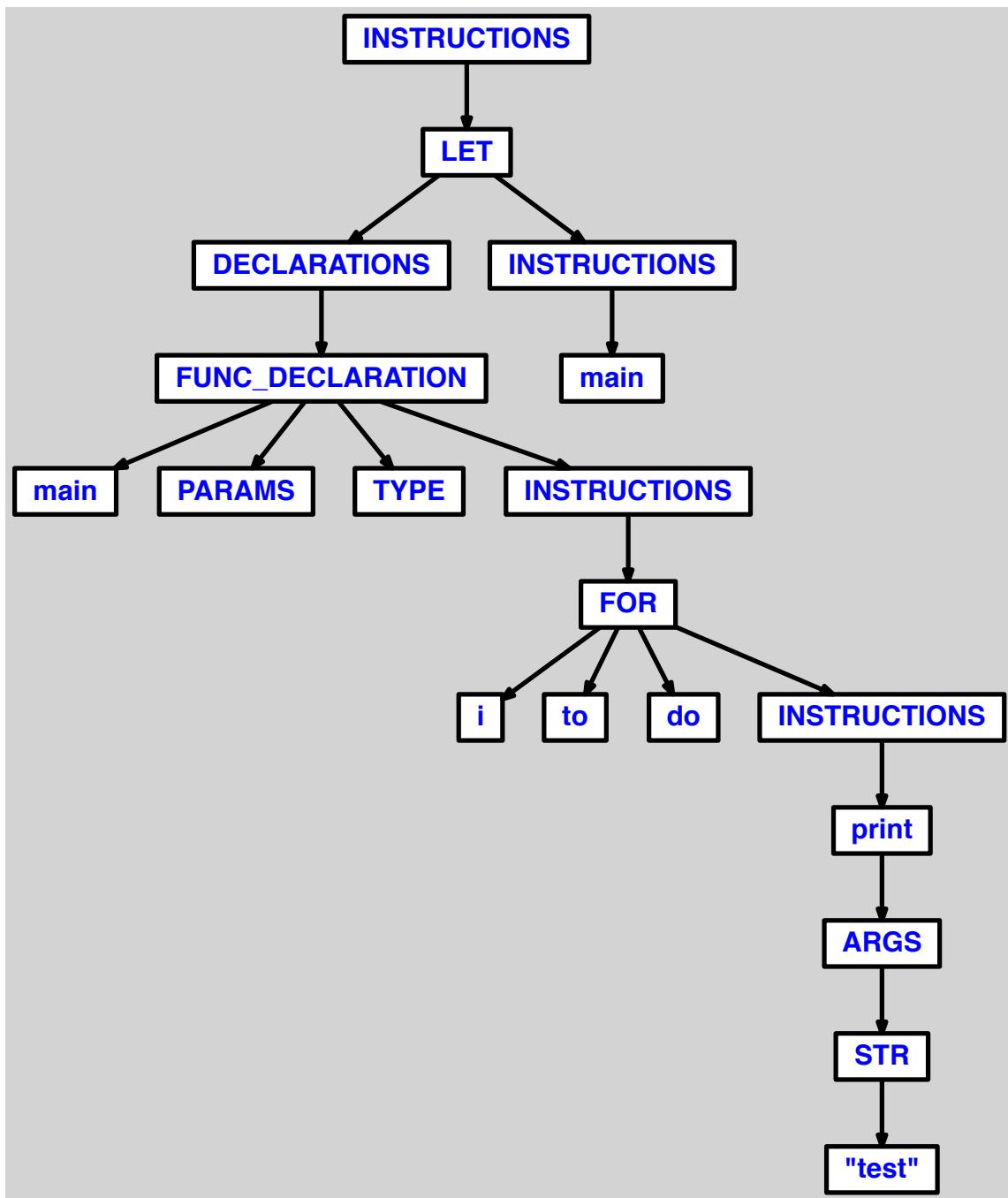


#### 6.1.5 for "express"

```
let
    function main() =
        for 3 to 5 do
            print("test")
in main() end
```

#### 6.1.6 for avec affectation de chaines pour le compteur et la borne maximale de l'itération

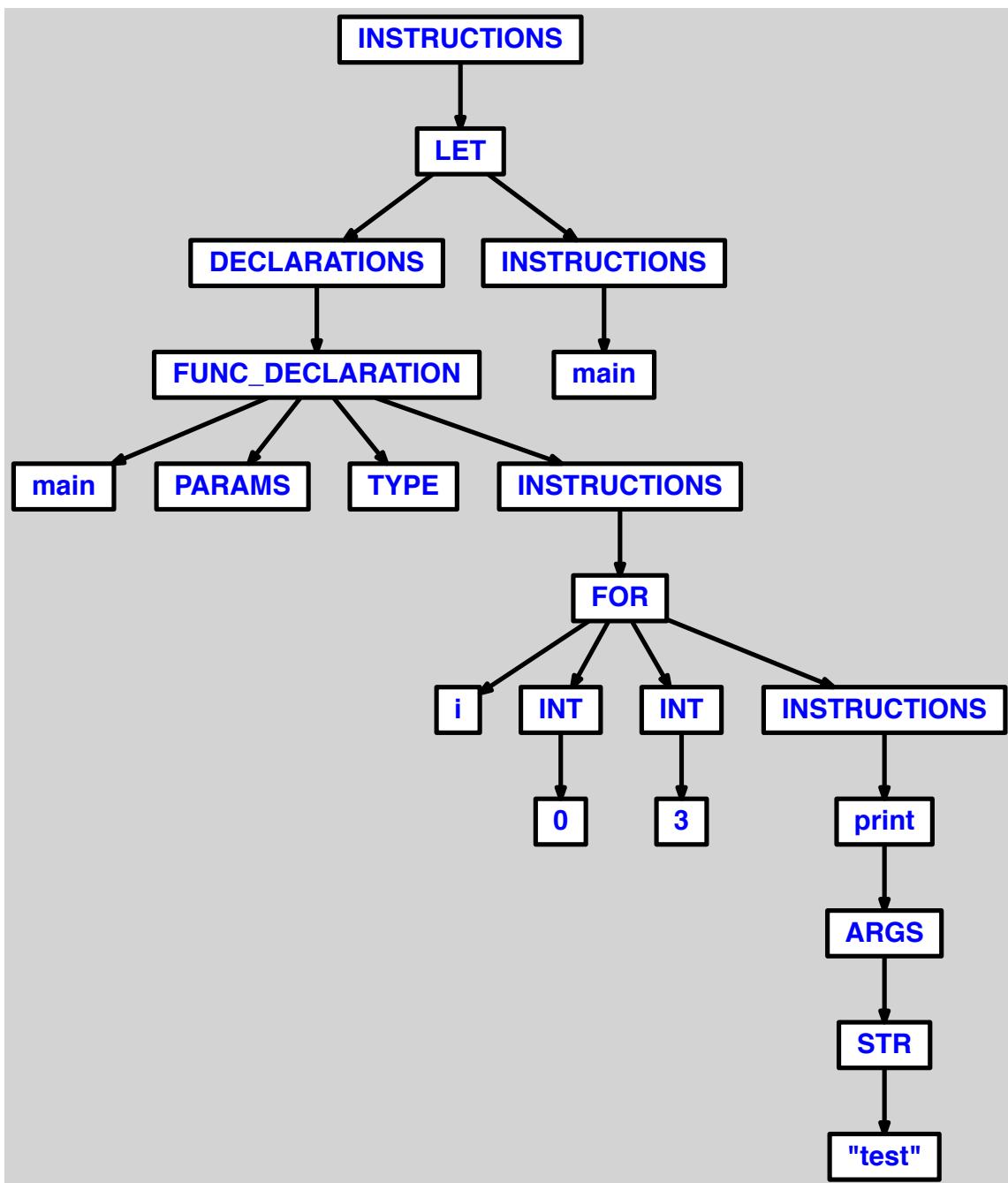
```
let
    function main() =
        for i := "0" to "3" do
            print("test")
in main() end
```



#### 6.1.7 for avec oubli du do

```

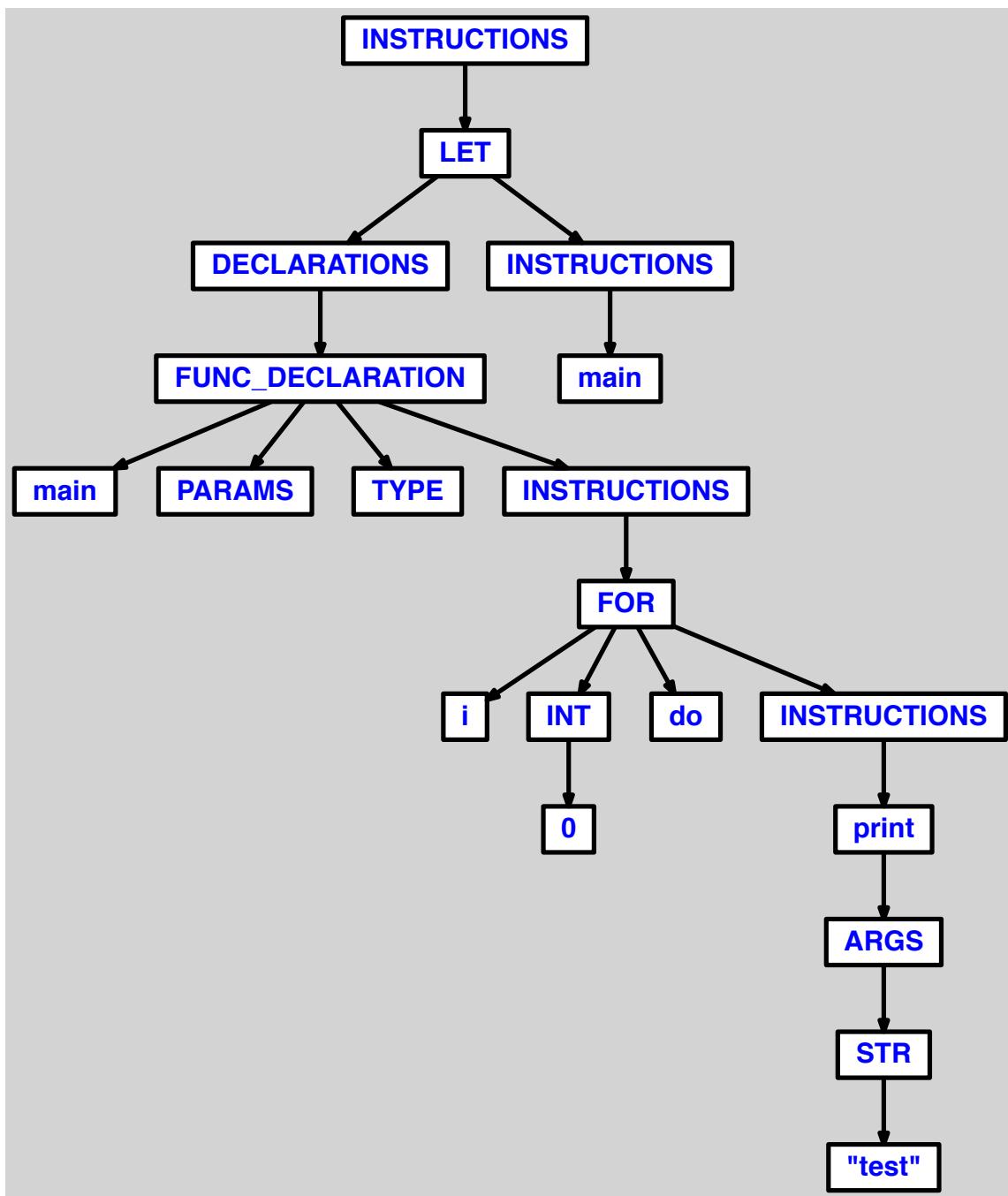
let
  function main() =
    for i := 0 to 3
      print("test")
in main() end
  
```



#### 6.1.8 for avec oubli de la borne maximale de l'itération

```

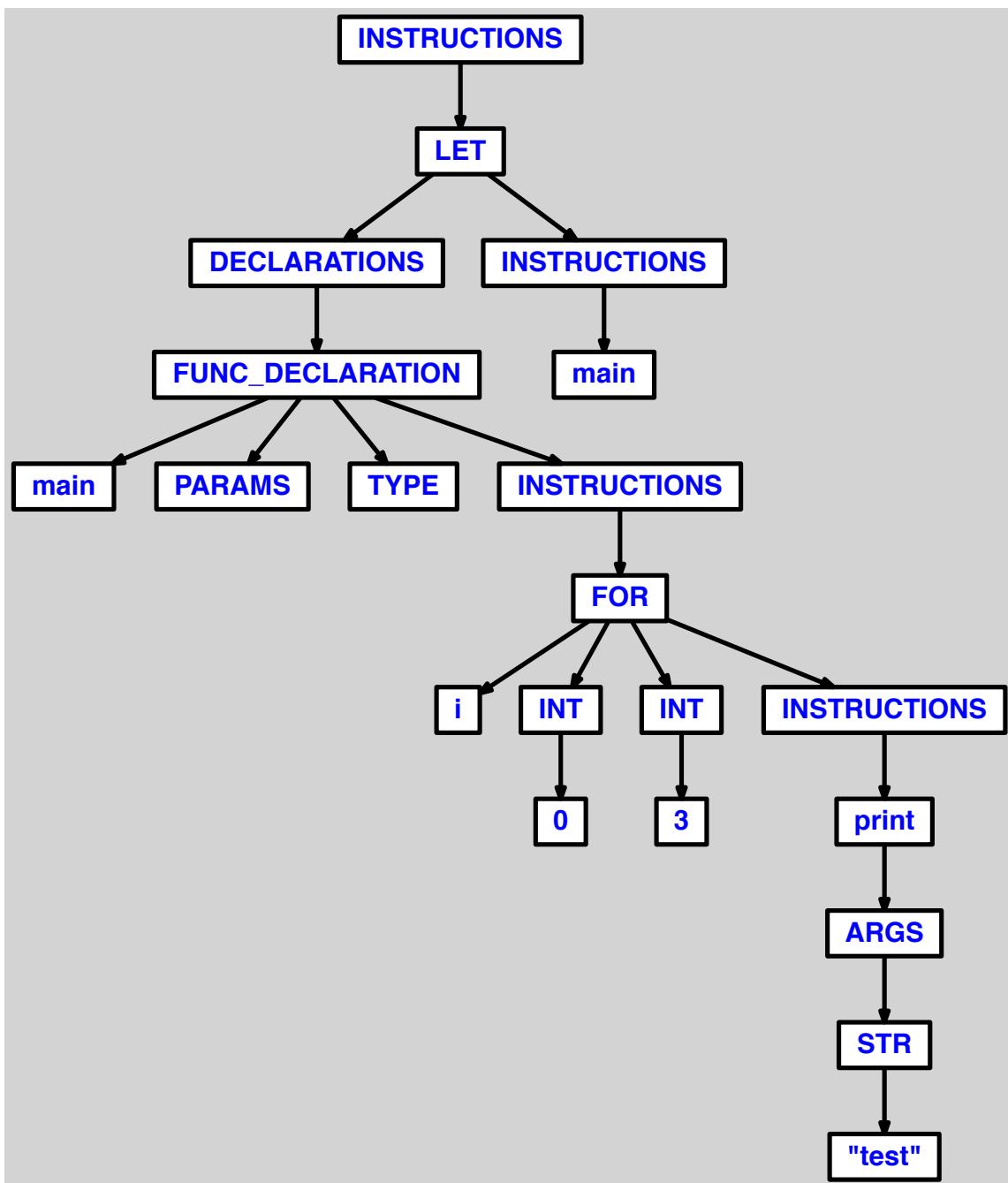
let
  function main() =
    for i := 0 to do
      print("test")
in main() end
  
```



#### 6.1.9 for avec oubli du to

```

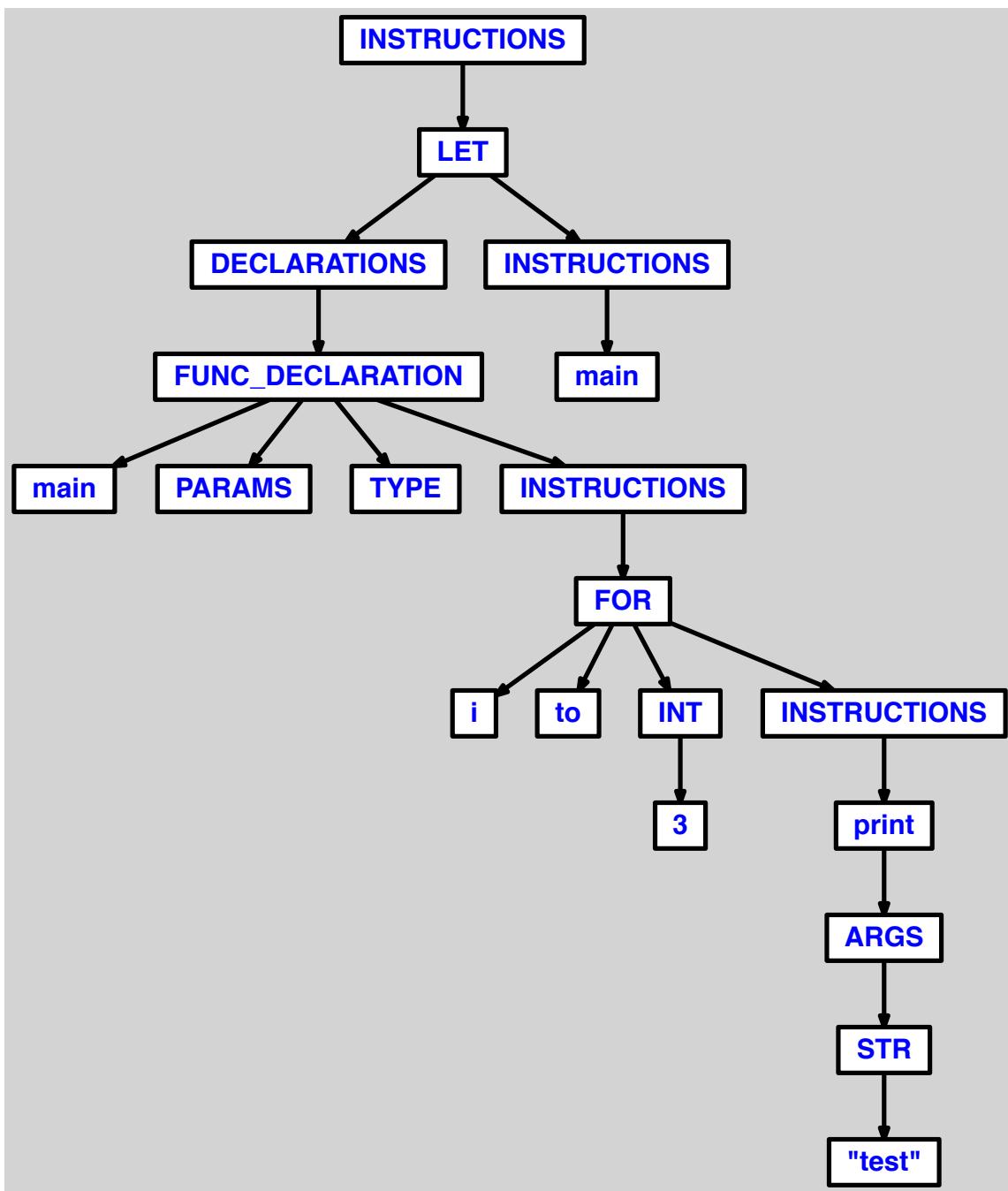
let
  function main() =
    for i := 0 3 do
      print("test")
in main() end
  
```



#### 6.1.10 for avec oubli de la borne inferieure de l'iteration

```

let
  function main() =
    for i := to 3 do
      print("test")
in main() end
  
```



#### 6.1.11 for avec oubli du =

```

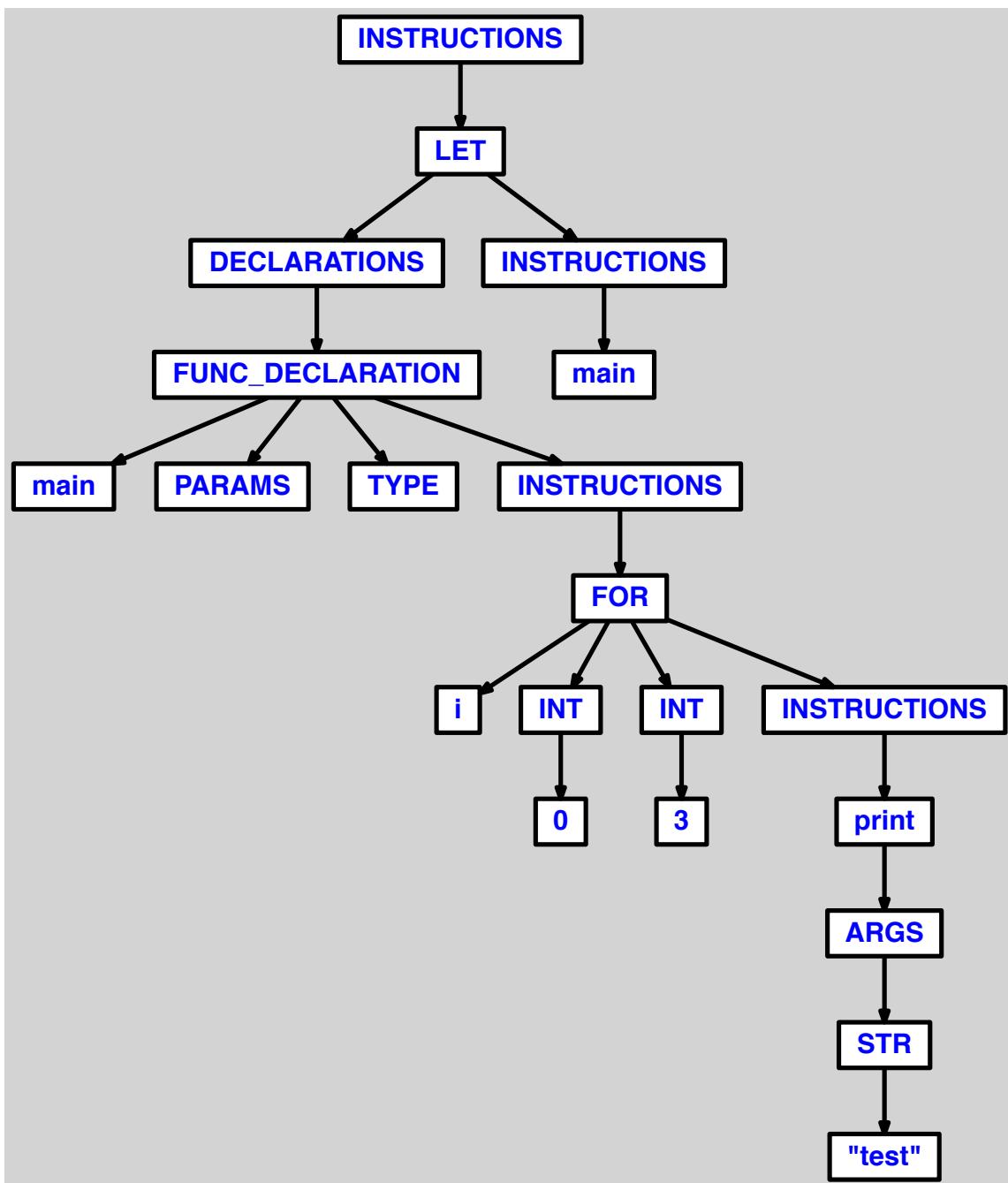
let
  function main() =
    for i : 0 to 3 do
      print("test")
in main() end
  
```

### 6.1.12 for avec oubli du :

```
let
    function main() =
        for i = 0 to 3 do
            print("test")
in main() end
```

### 6.1.13 for avec oubli du :=

```
let
    function main() =
        for i 0 to 3 do
            print("test")
in main() end
```



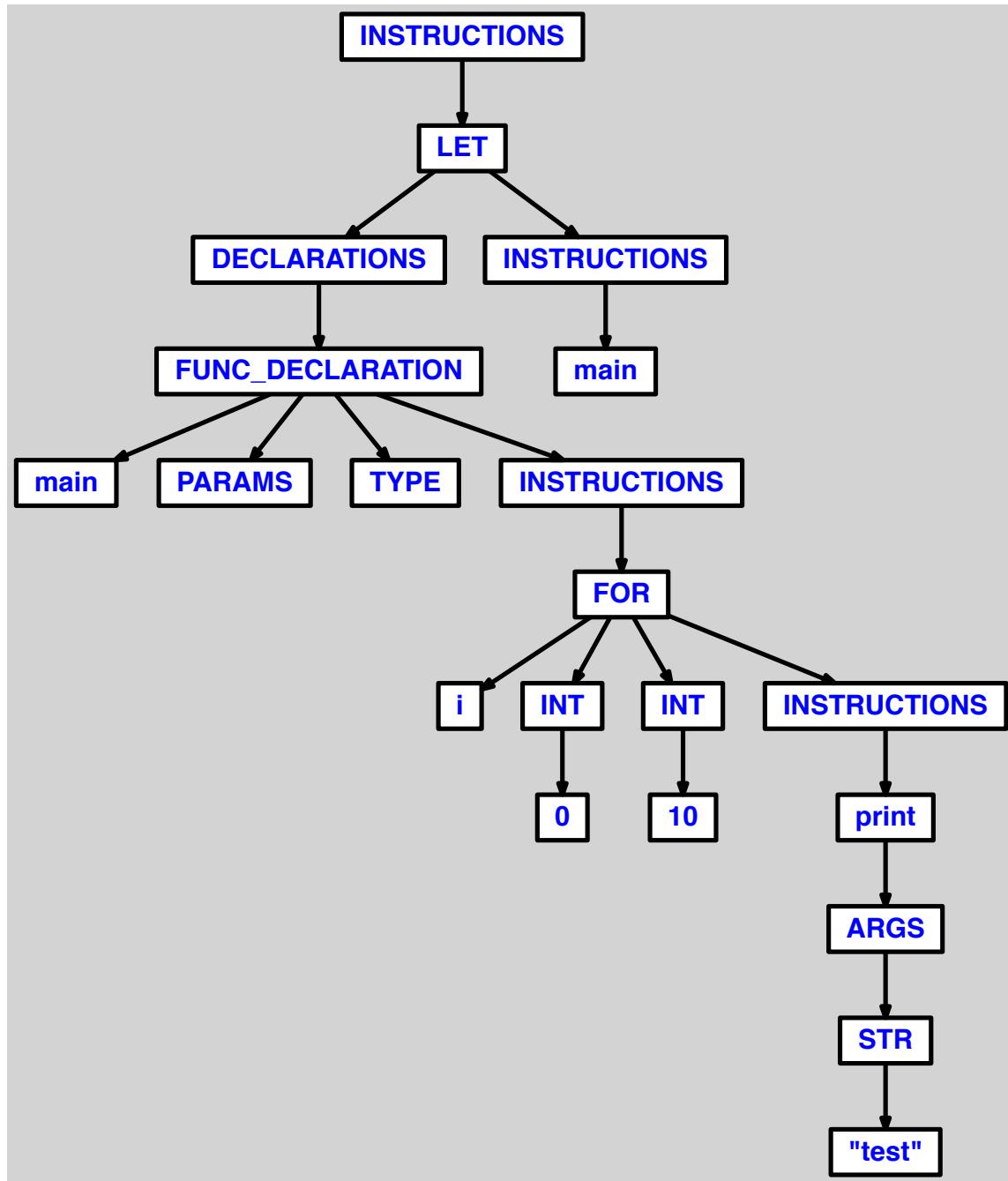
## 6.2 OK

### 6.2.1 for simple croissant

```

let
function main() =
  for i := 0 to 10 do
    print("test")
  
```

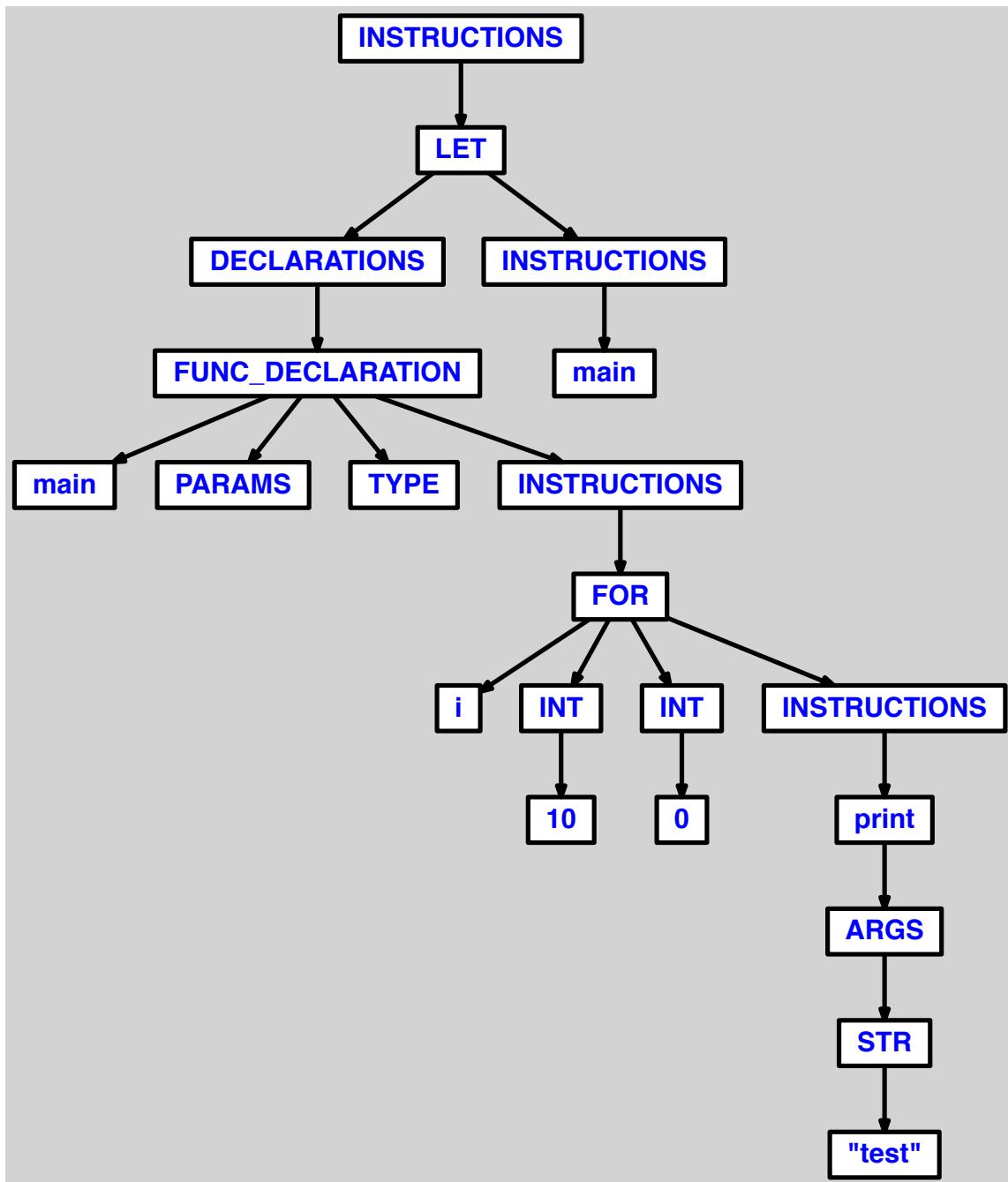
```
in main() end
```



### 6.2.2 for simple decroissant

```
let
    function main() =
        for i := 10 to 0 do
```

```
print("test")
in main() end
```



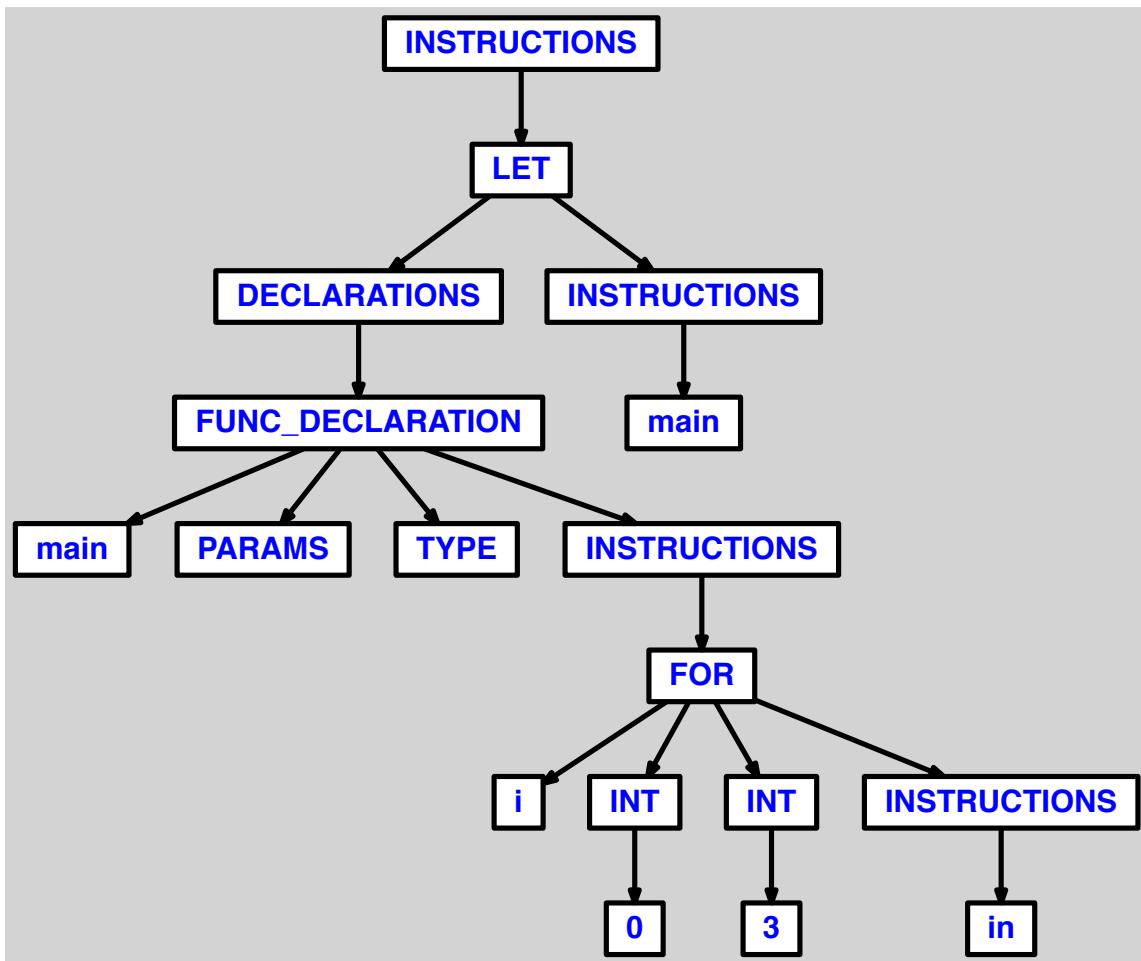
### 6.2.3 for sans instruction

```
let
function main() =
```

```

for i := 0 to 3 do
in main() end

```

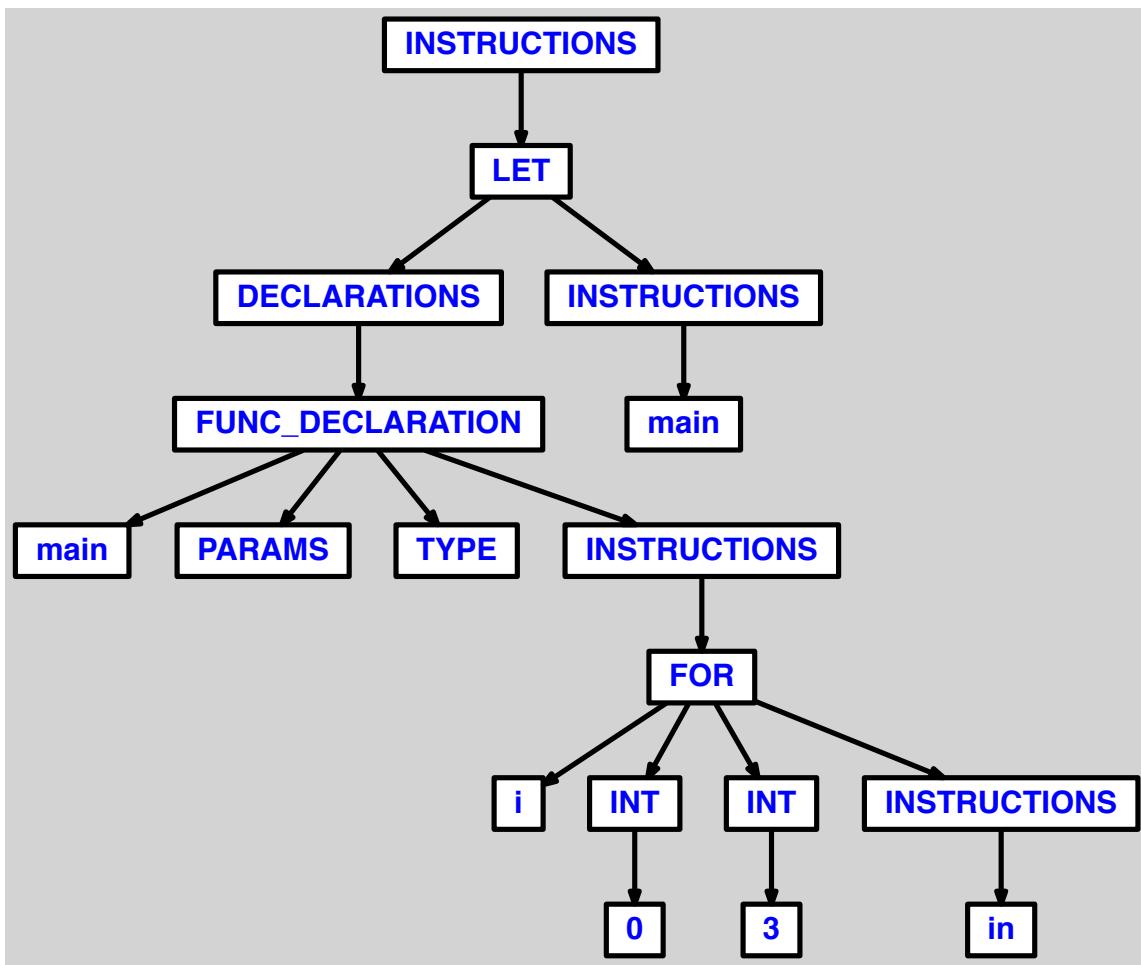


#### 6.2.4 for avec ligne vide

```

let
function main() =
  for i := 0 to 3 do
in main() end

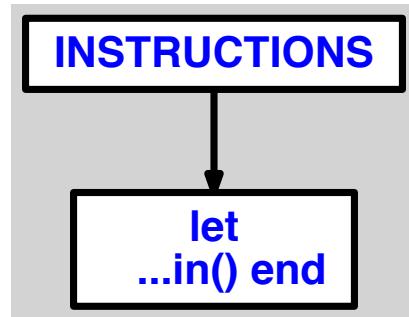
```



#### 6.2.5 double-imbrication de for

```

let
  function main()()
    for i := 0 to 3 do
      for j := 4 to 7 do
        printi(i+j)
  in main() end
  
```

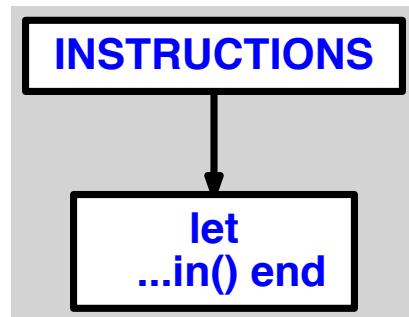


#### 6.2.6 triple-imbrication de for

```

let
    function main()()
        for i := 0 to 3 do
            for j := 4 to 7 do
                for k := 8 to 10 do
                    printi(i+j+k)
    in main() end

```

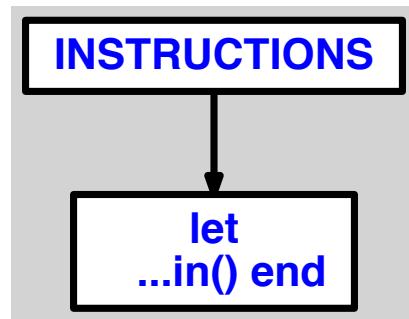


#### 6.2.7 for avec reutilisation de compteur

```

let
    function main()()
        for i := 0 to 3 do
            for j := i+4 to 10 do
                printi(i+j)
    in main() end

```

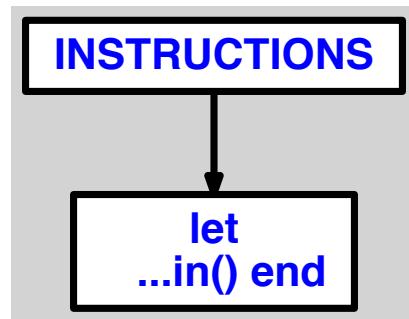


#### 6.2.8 for avec reutilisation de compteurs

```

let
    function main()()
        for i := 0 to 3 do
            for j := 4 to i+4 do
                for k := j-4 to 4 do
                    printi(i+j+k)
    in main() end

```



## 7 function

### 7.1 KO

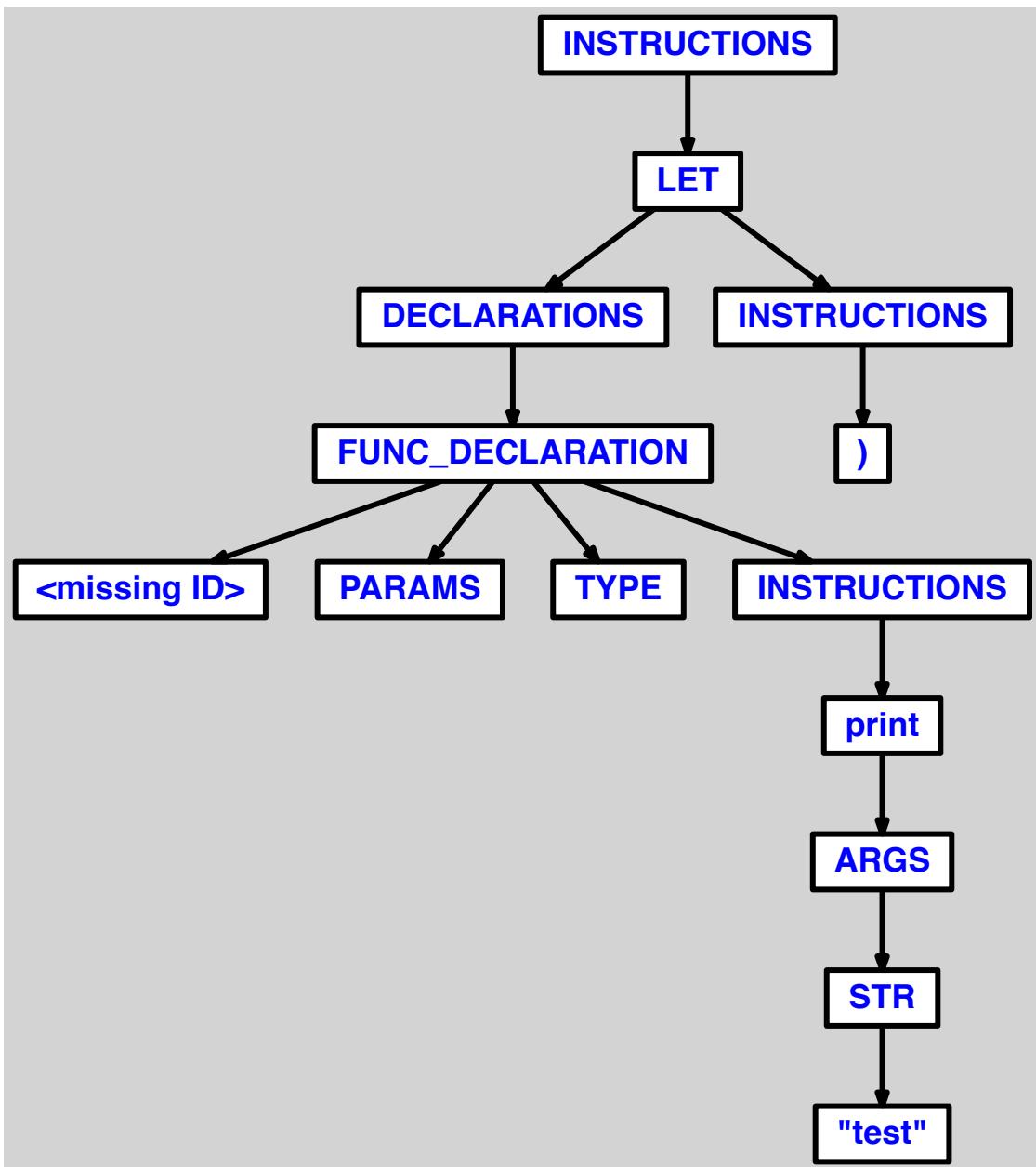
#### 7.1.1 fonction identifiee par caractere special

```

let
    function \() = print("test")

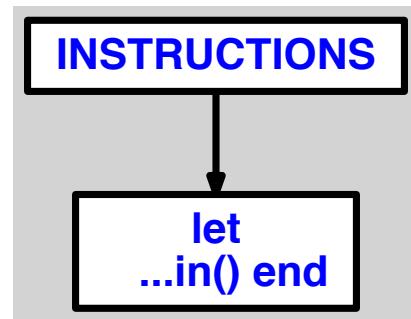
```

```
in \() end
```



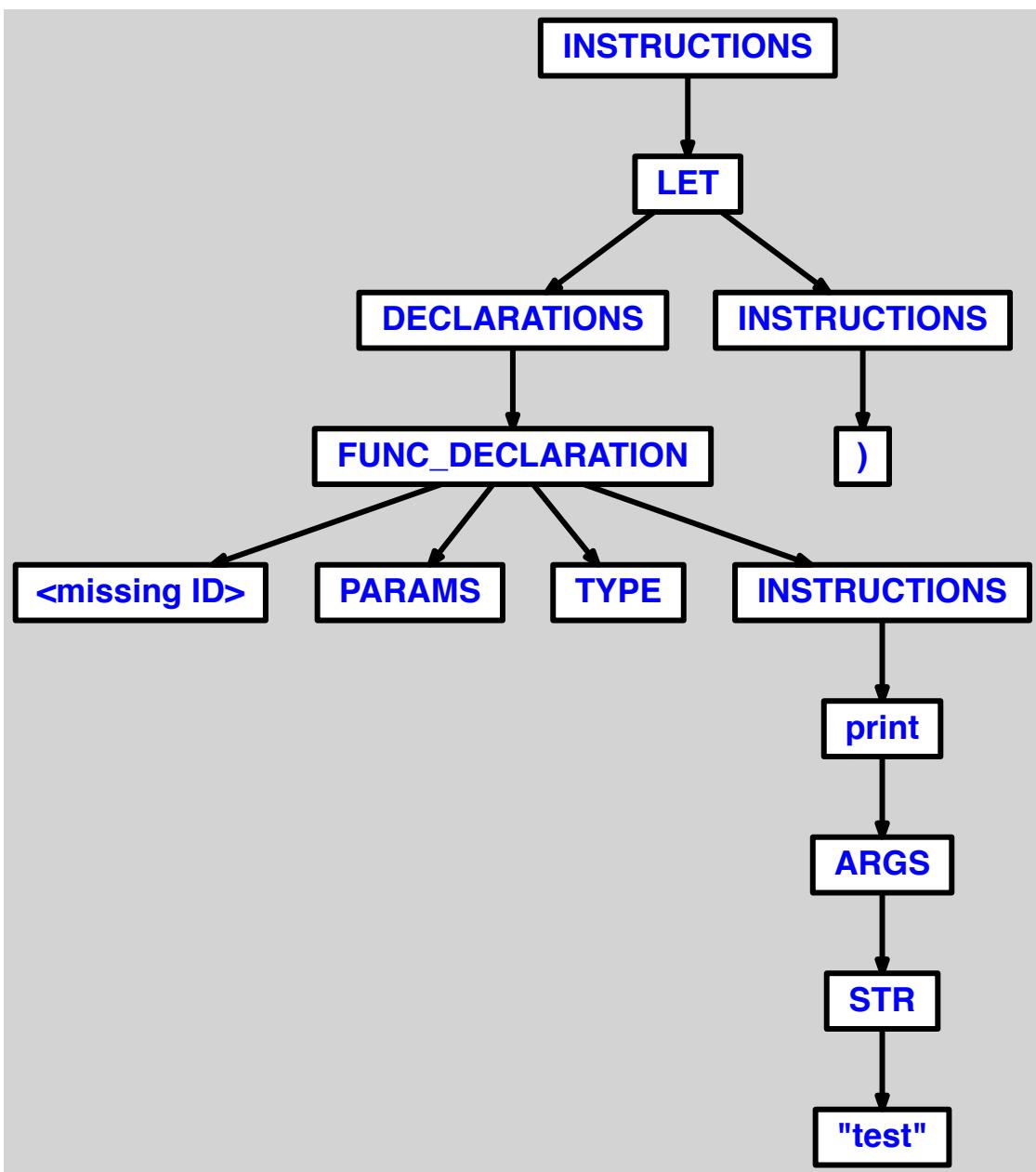
### 7.1.2oubli de function

```
let
    main() = print("test")
in main() end
```



#### 7.1.3 fonction sans identifiant

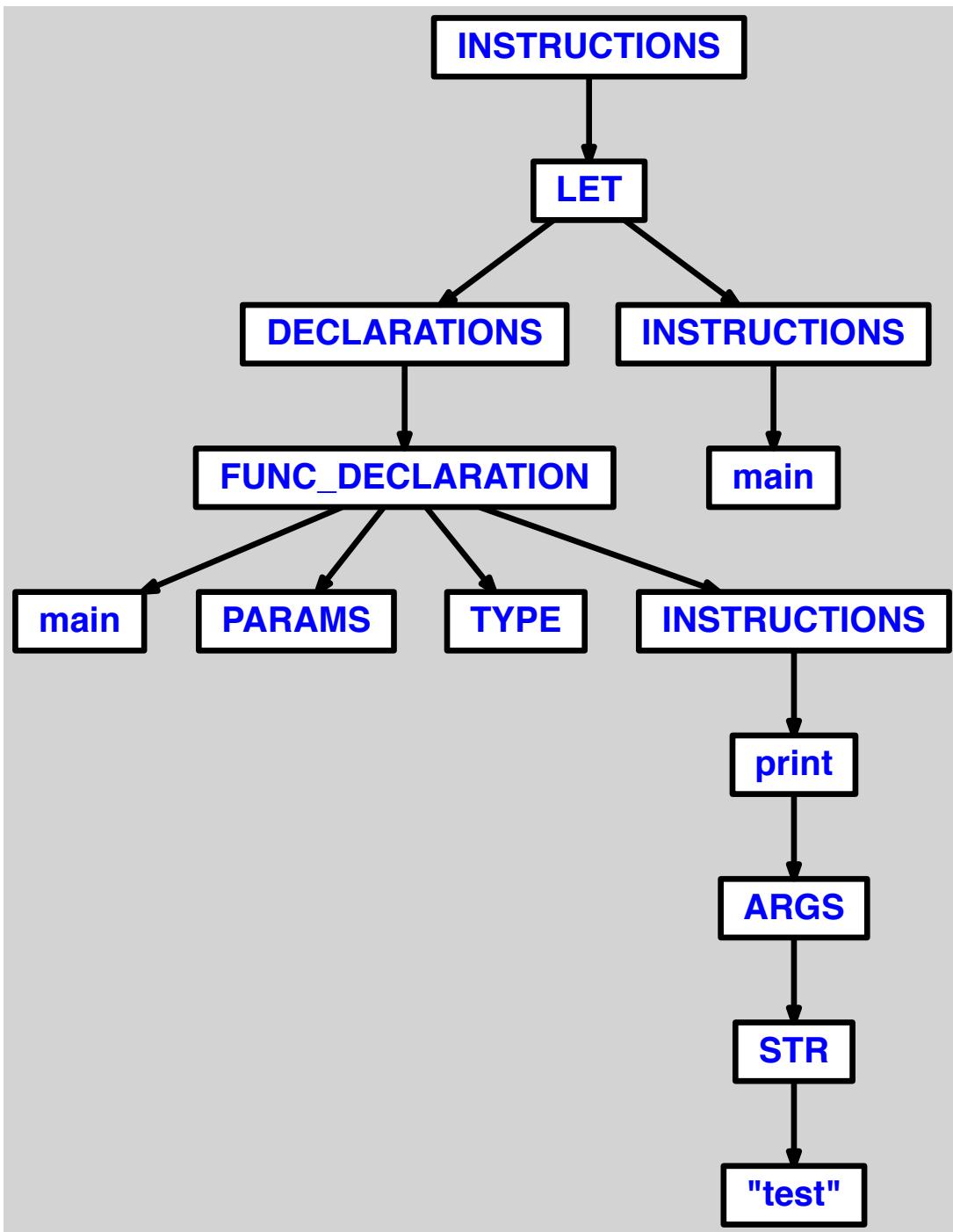
```
let
    function () = print("test")
in () end
```



#### 7.1.4 oubli de (

```

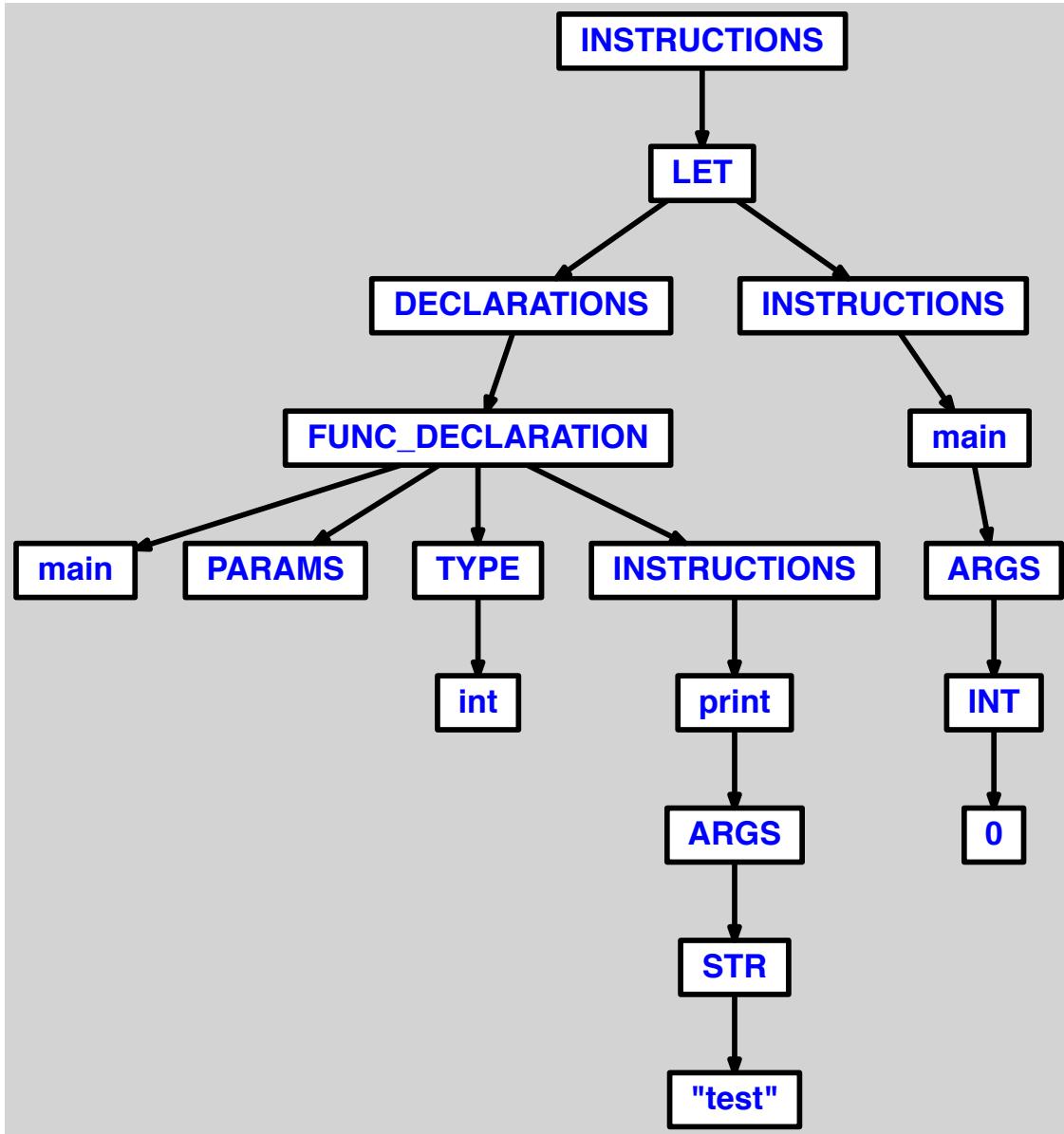
let
    function main() = print("test")
in main() end
  
```



#### 7.1.5oubli d'identifiant de parametre

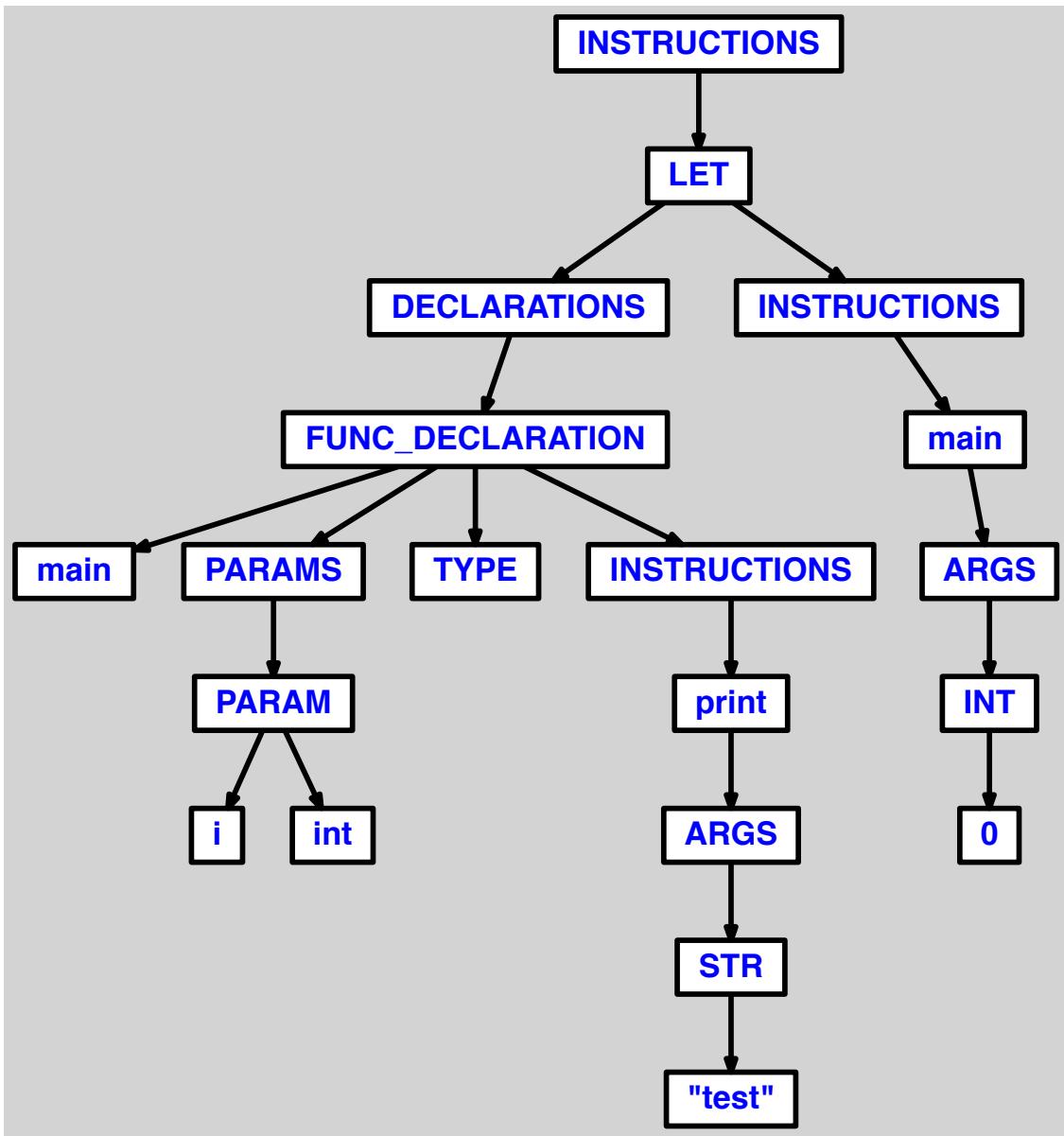
```
let
  function main(: int) = print("test")
```

```
in main(0) end
```



#### 7.1.6 oubli de : pour parametre

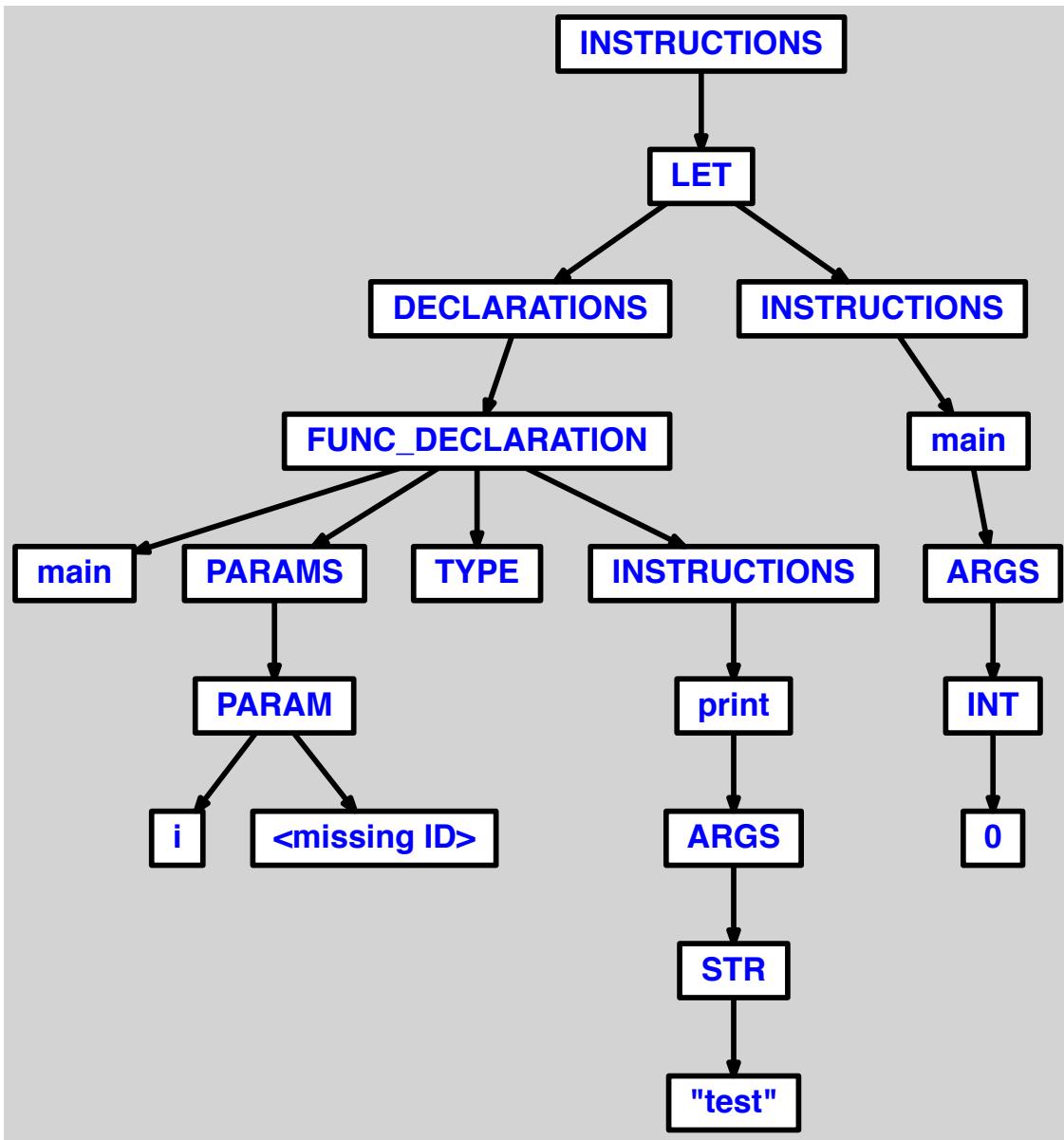
```
let
    function main(i int) = print("test")
in main(0) end
```



### 7.1.7oubli de type de parametre

```

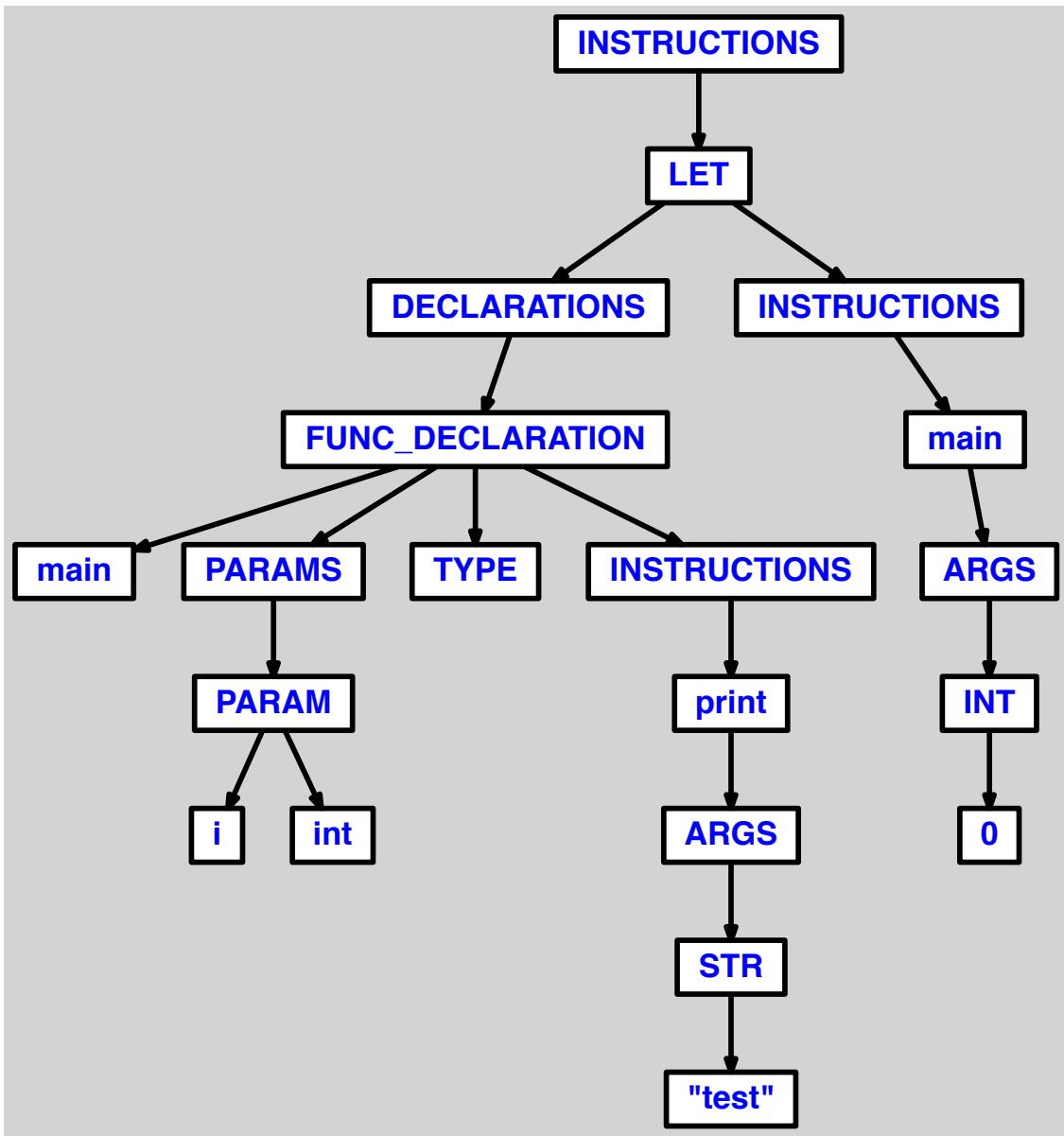
let
    function main(i: ) = print("test")
in main(0) end
  
```



### 7.1.8 oubli de )

```

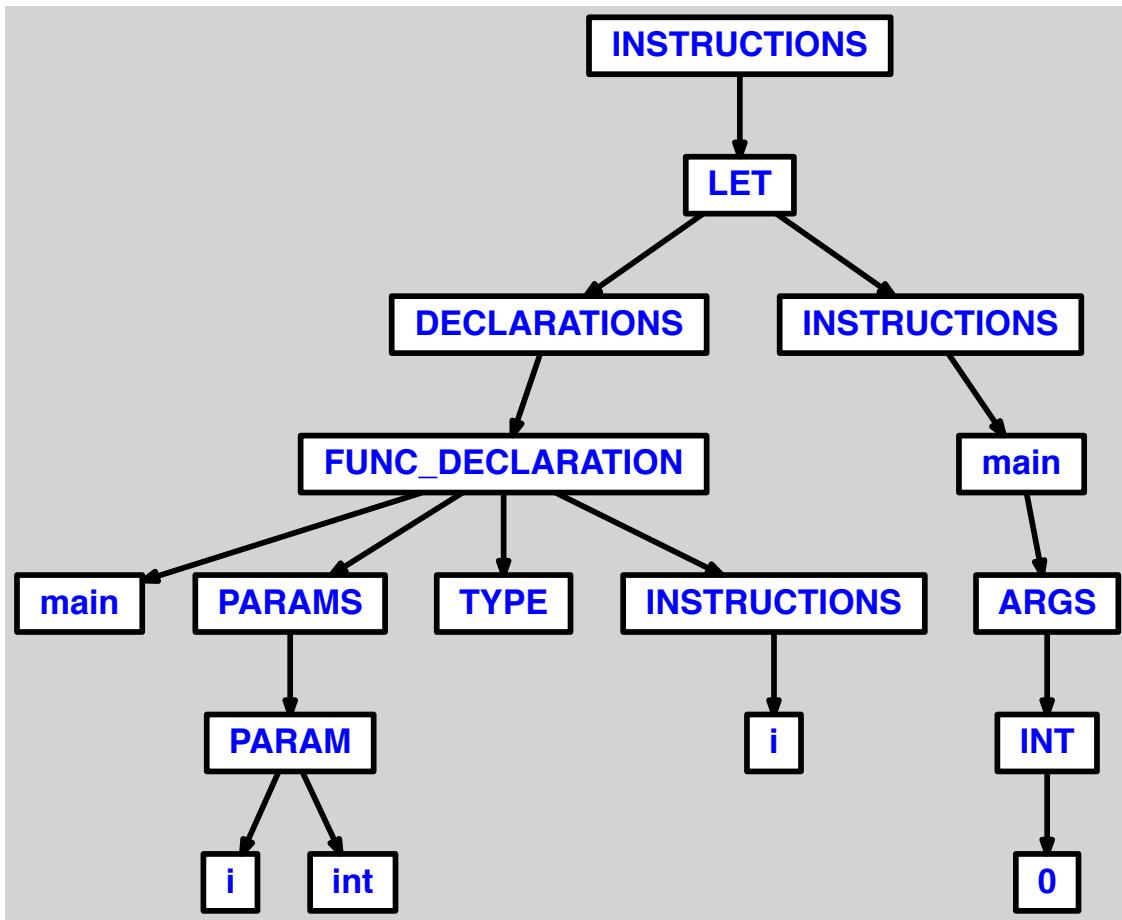
let
    function main(i: int = print("test"))
in main(0) end
  
```



### 7.1.9 oubli de : pour type de fonction

```

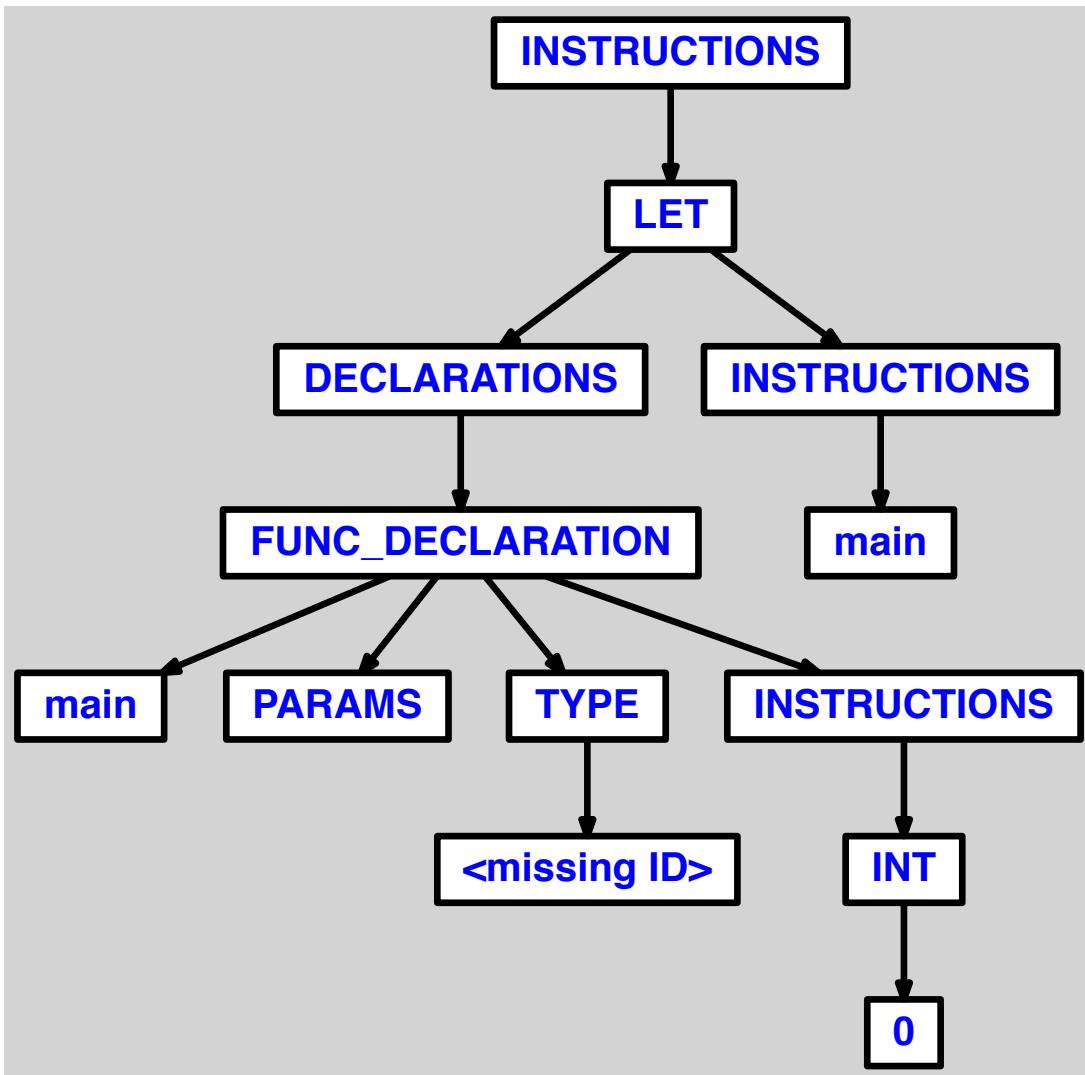
let
    function main(i: int) int = i
in main(0) end
  
```



7.1.10 : pour type de fonction présent mais pas de type

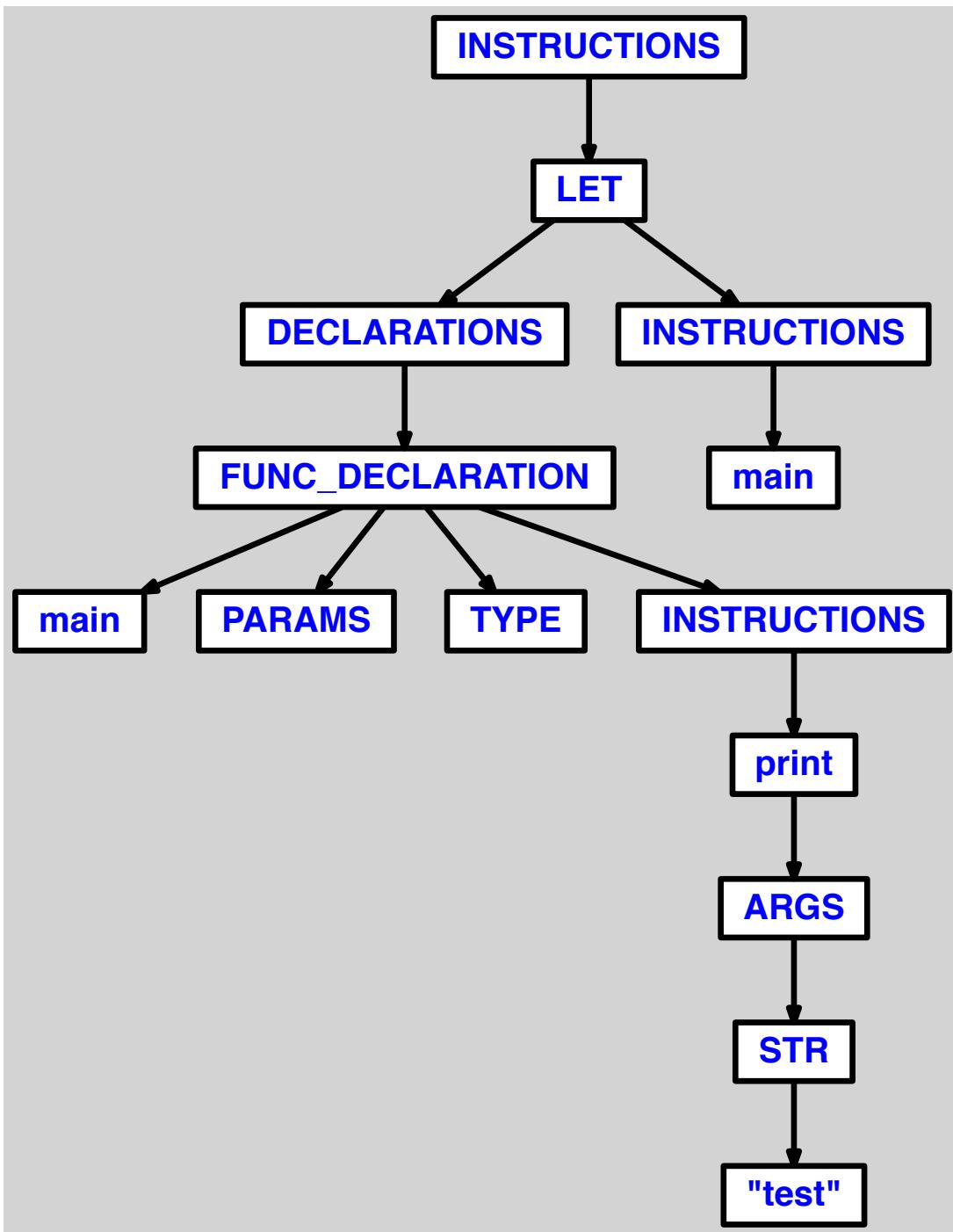
```

let
    function main(): = 0
in main() end
  
```



7.1.11 oubli de =

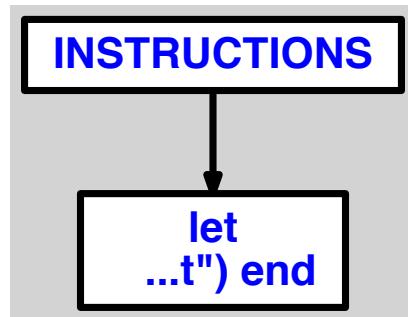
```
let
    function main() print("test")
in main() end
```



#### 7.1.12 parametres mal séparés

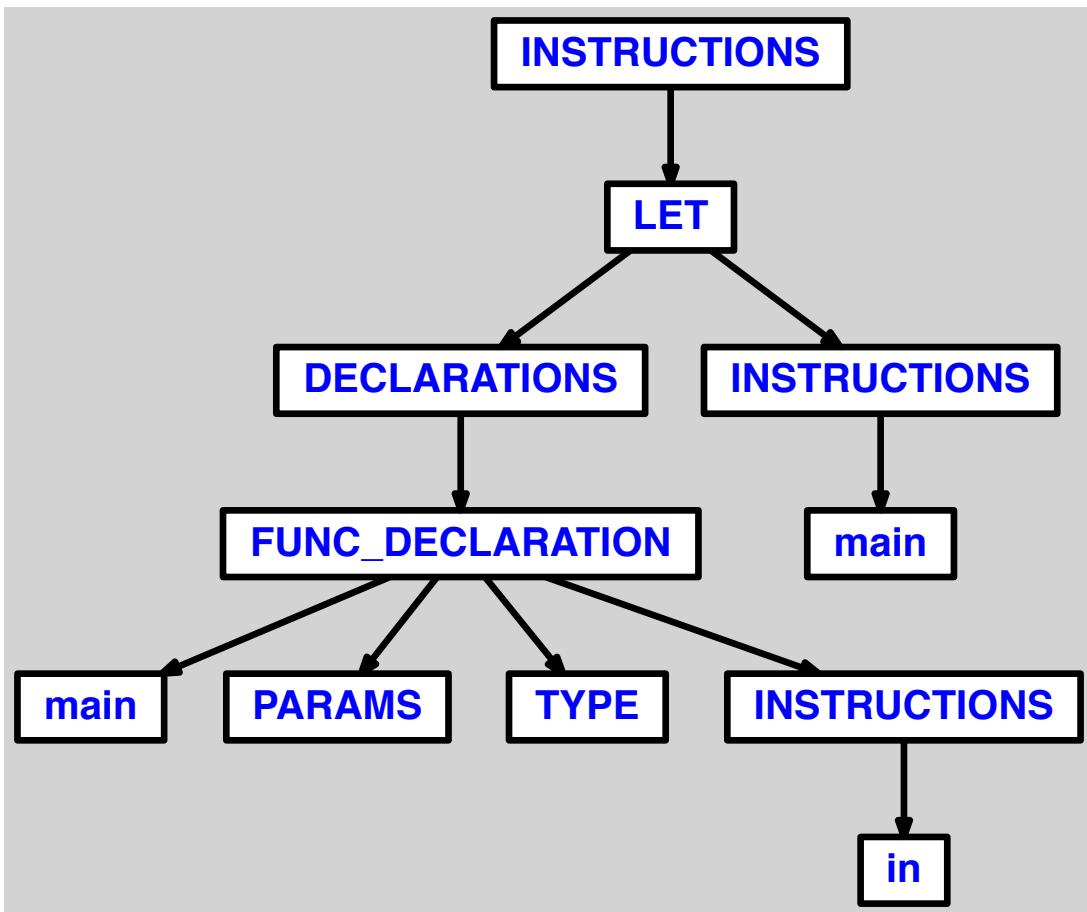
```
let
  function main(i: int s: string) =
```

```
printi(i);
print(s)
in main(1, "test") end
```



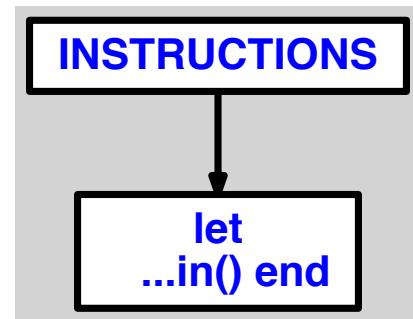
#### 7.1.13 fonction sans instruction

```
let
    function main() =
in main() end
```



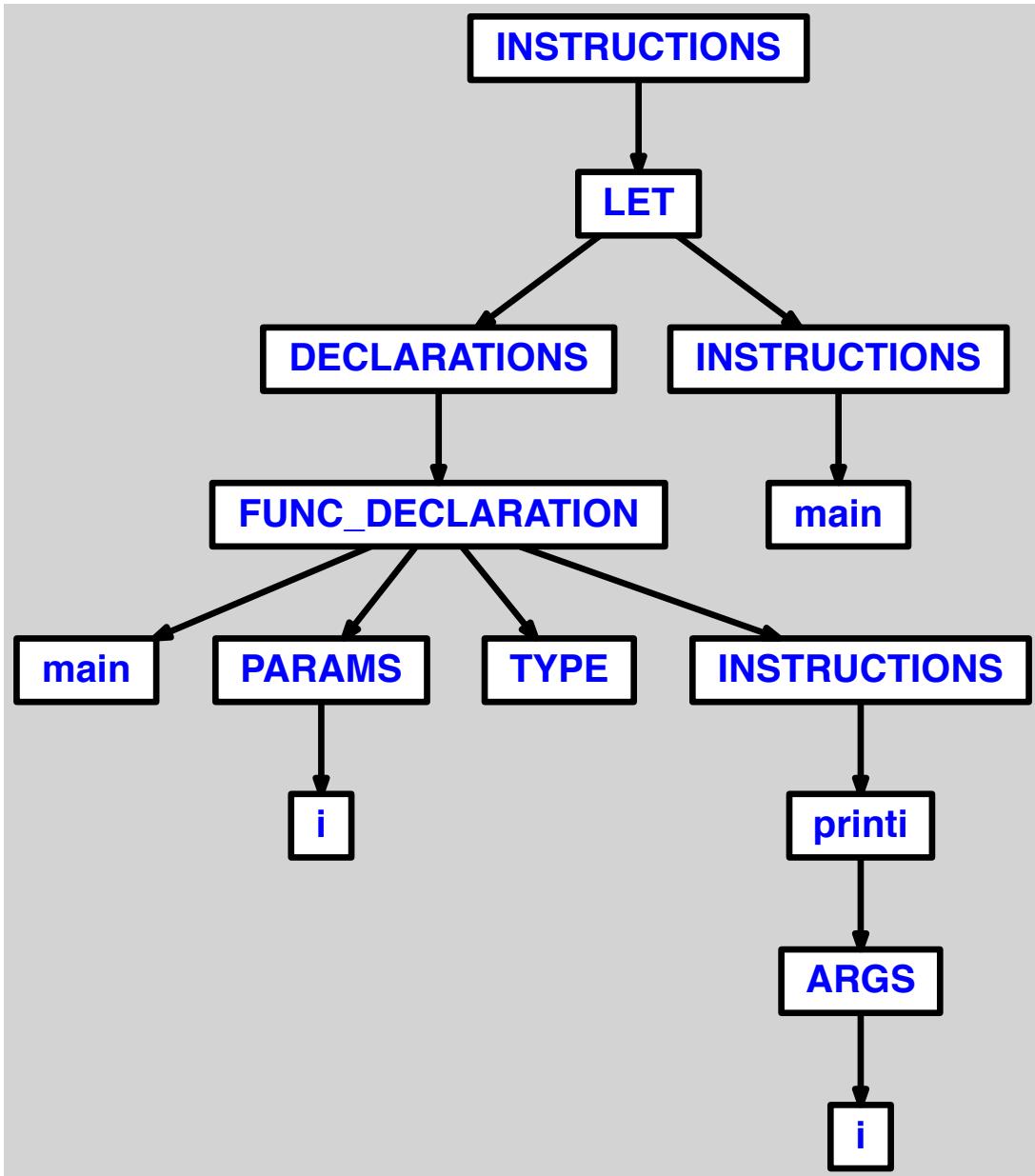
#### 7.1.14 function mal écrit

```
let
    functionn main() = print("test")
in main() end
```



#### 7.1.15 fonction avec identifiant seulement en parametre

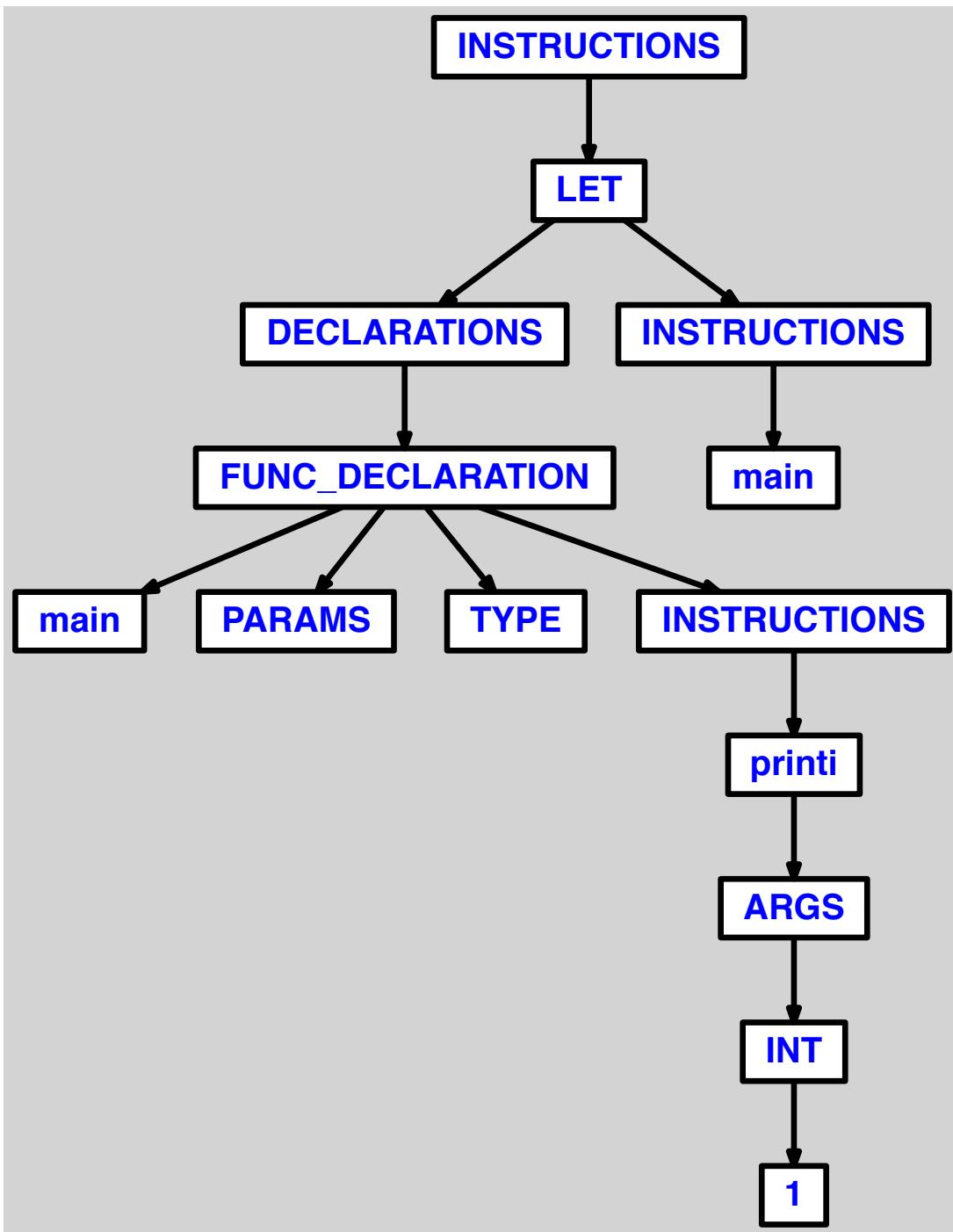
```
let
    function main(i) = printi(i)
in main() end
```



#### 7.1.16 fonction avec entier directement en parametre

```

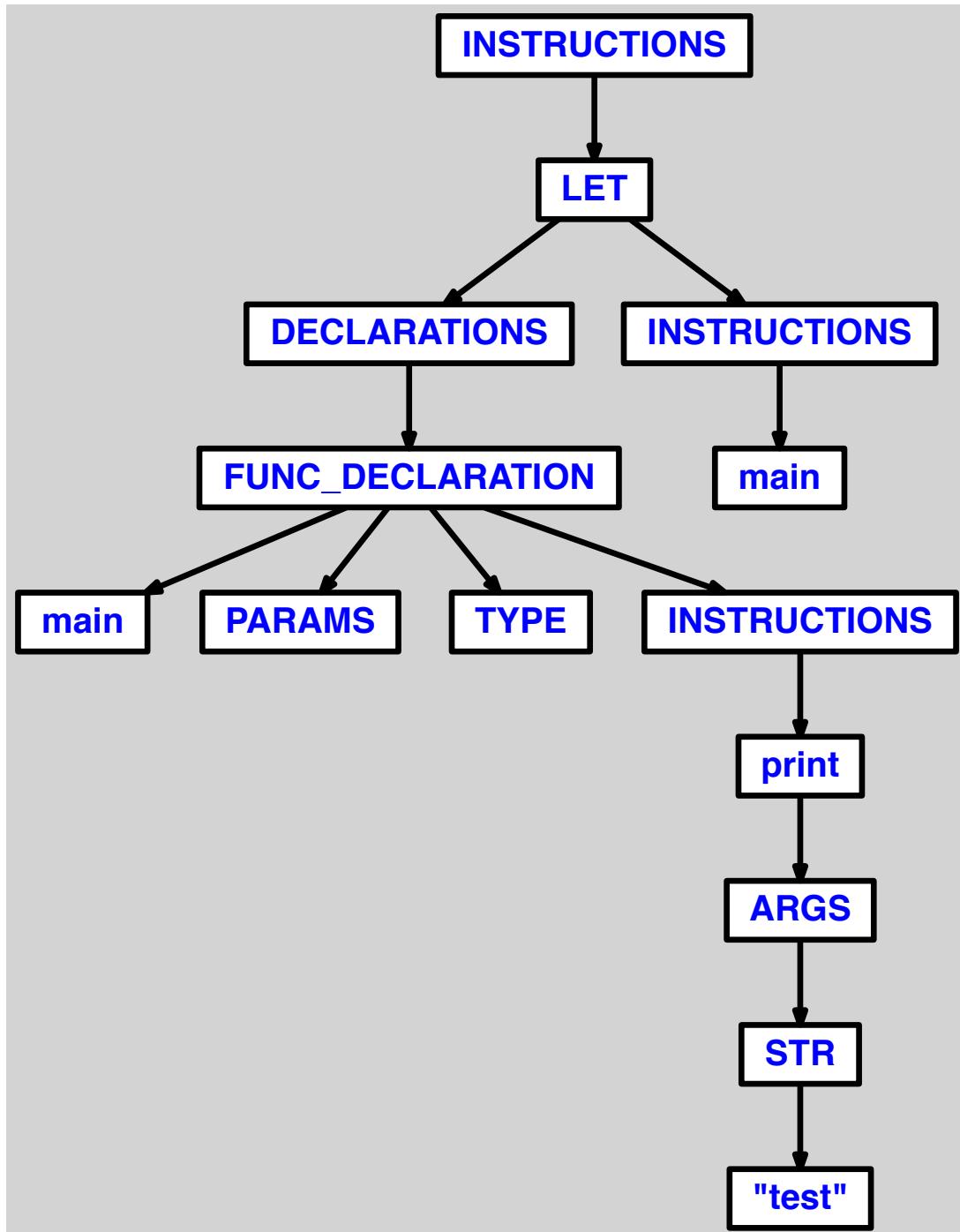
let
    function main(1) = printi(1)
in main() end
  
```



#### 7.1.17 fonction avec chaine directement en parametre

```
let
  function main("test") = print("test")
```

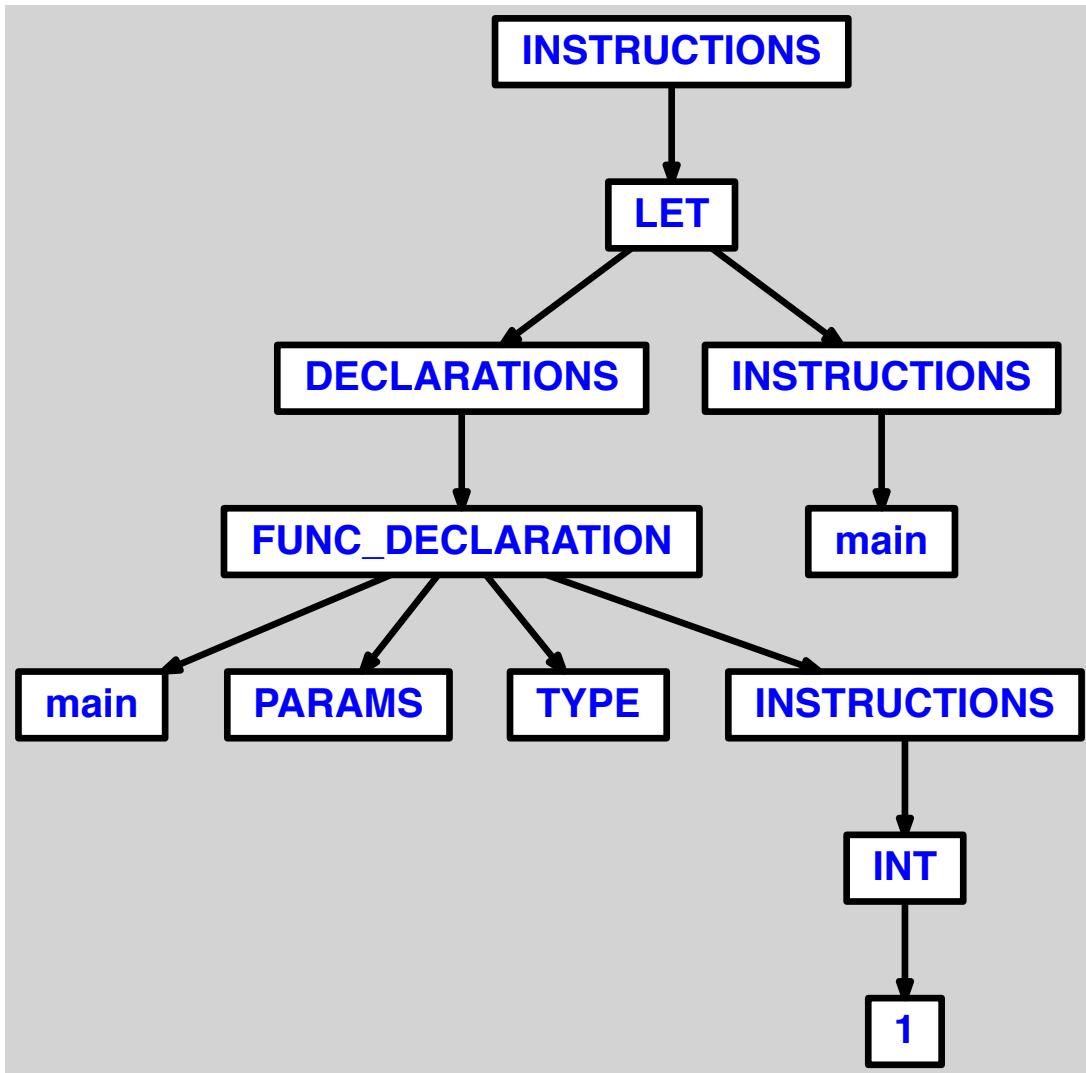
```
in main() end
```



## 7.2 OK

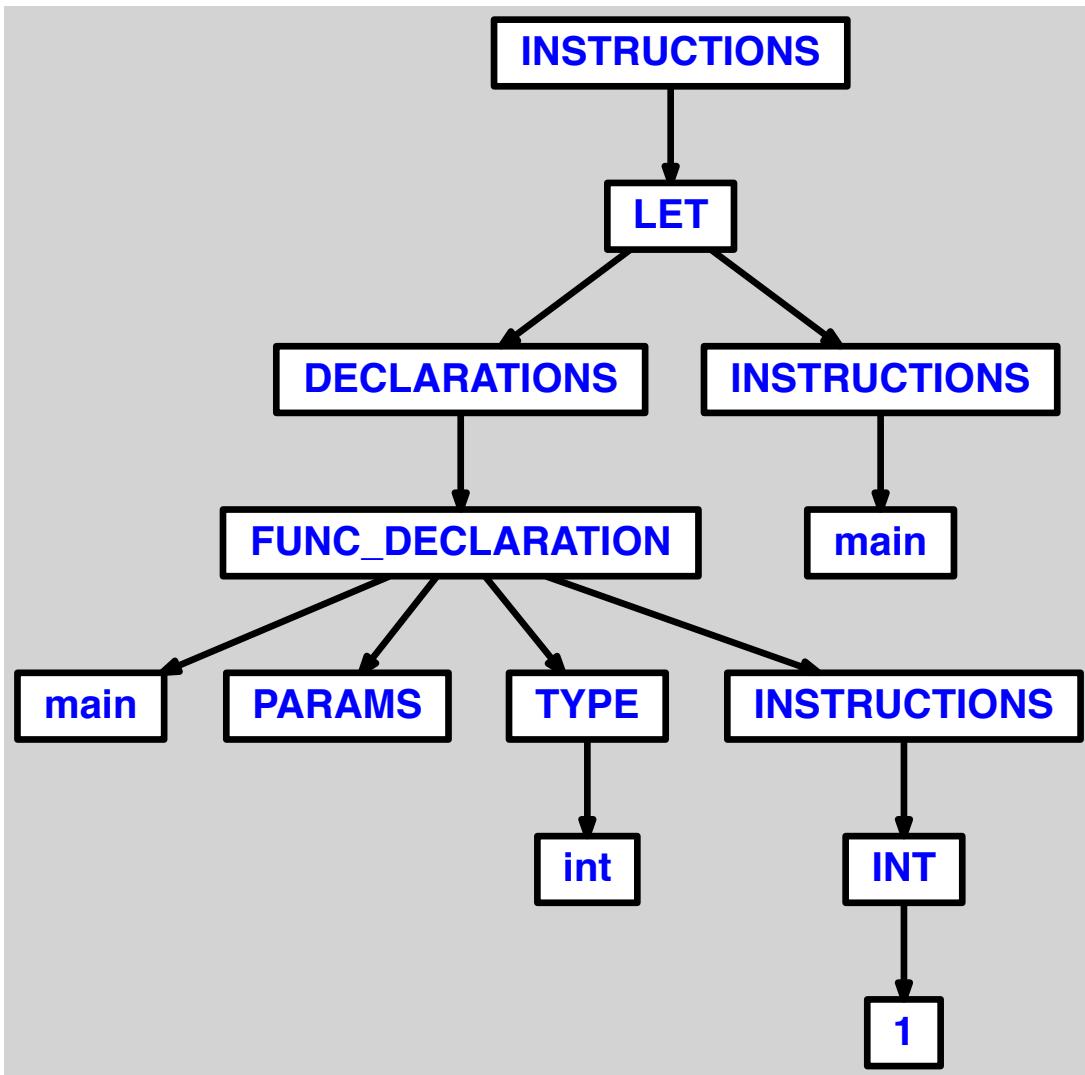
### 7.2.1 fonction definissant un entier

```
let
    function main() = 1
in main() end
```



### 7.2.2 fonction simple, avec declaration du type de retour int et instruction de retour

```
let
    function main() : int = 1
in main() end
```



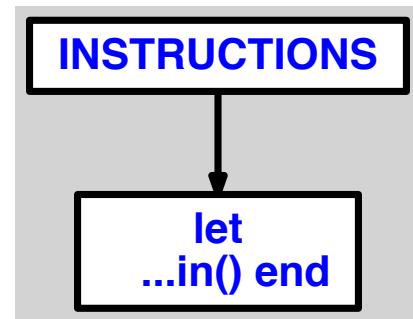
### 7.2.3 fonction simple, avec instruction simple et instruction retournant un entier

let

```

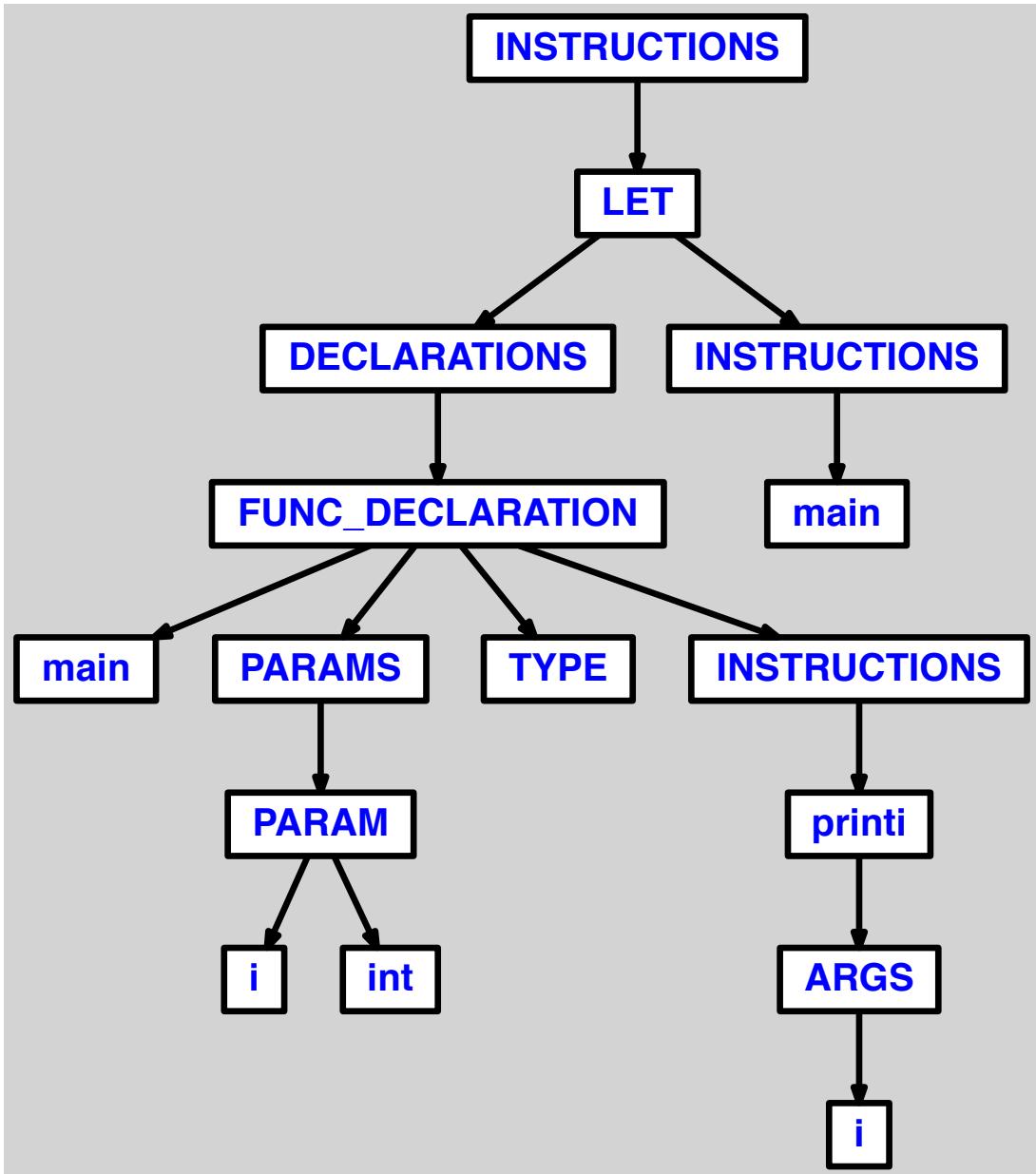
        function main() =
            print("test");
            1
  
```

in main() end



#### 7.2.4 fonction avec 1 parametre entier

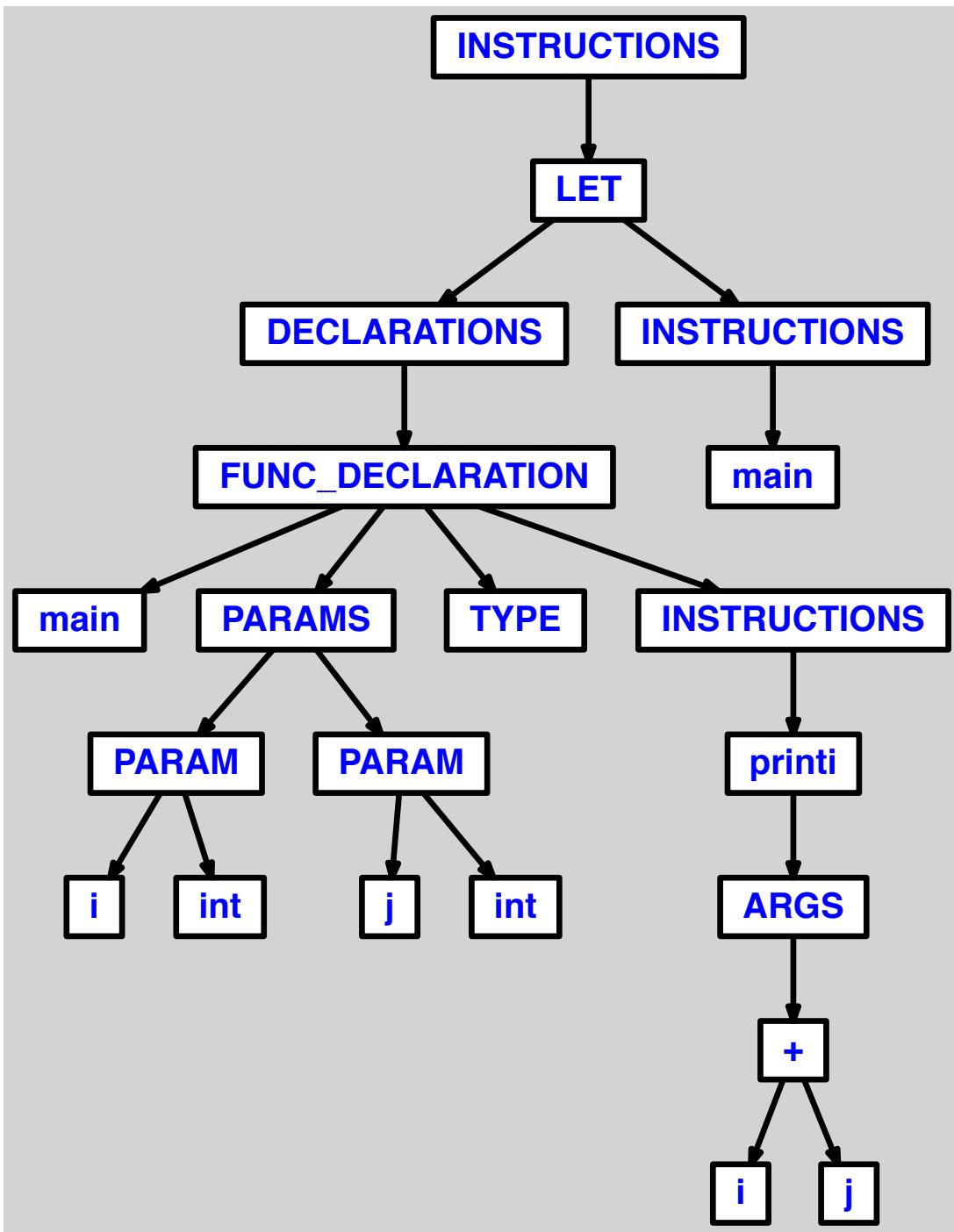
```
let
    function main(i: int) = printi(i)
in main()
```



#### 7.2.5 fonction avec 2 paramètres entiers

```

let
    function main(i: int, j: int) = printi(i+j)
in main() end
  
```

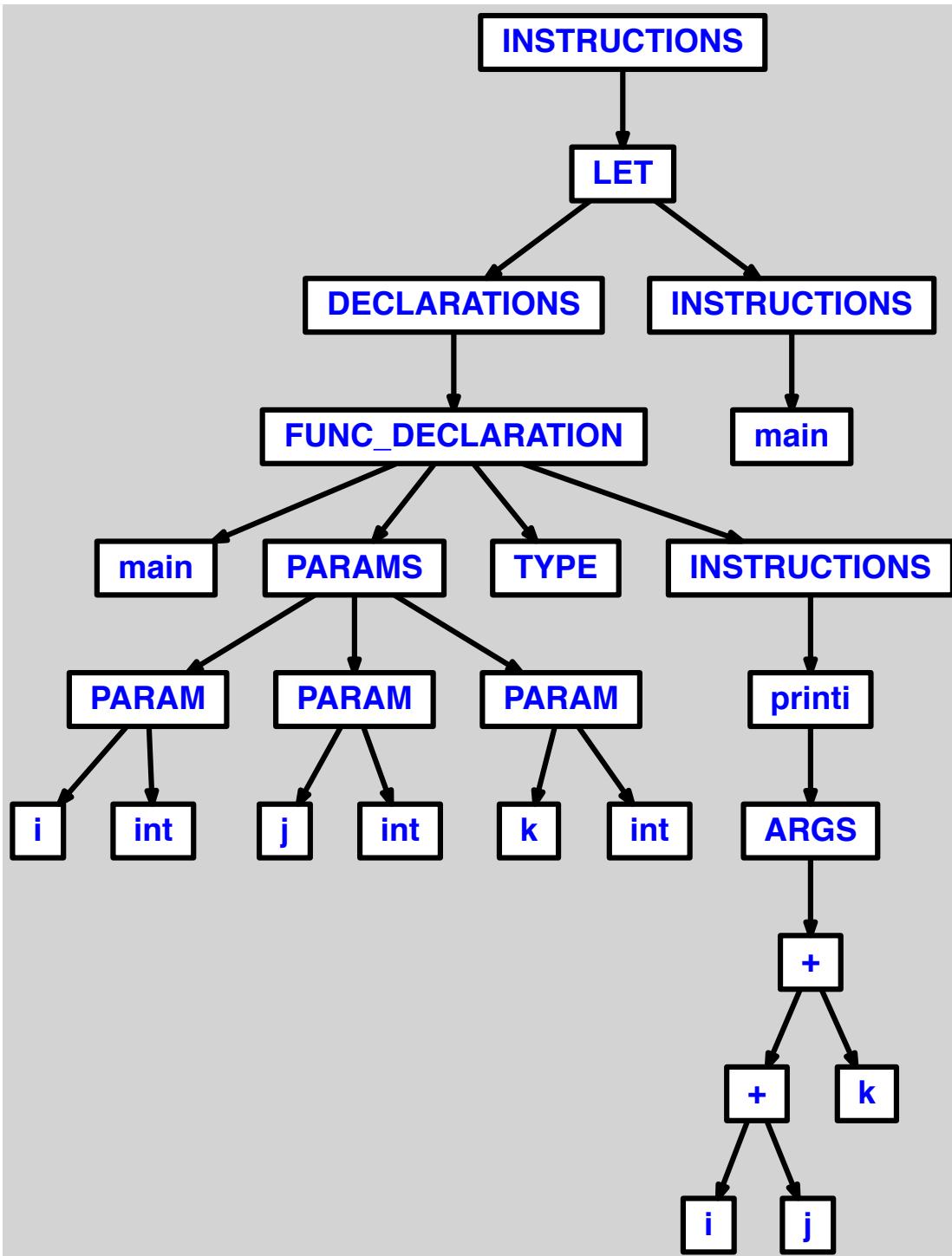


#### 7.2.6 fonction avec 3 paramètres entiers

```

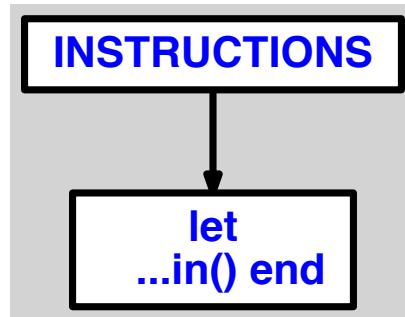
let
  function main(i: int, j: int, k: int) = printi(i+j+k)
  
```

in main() end



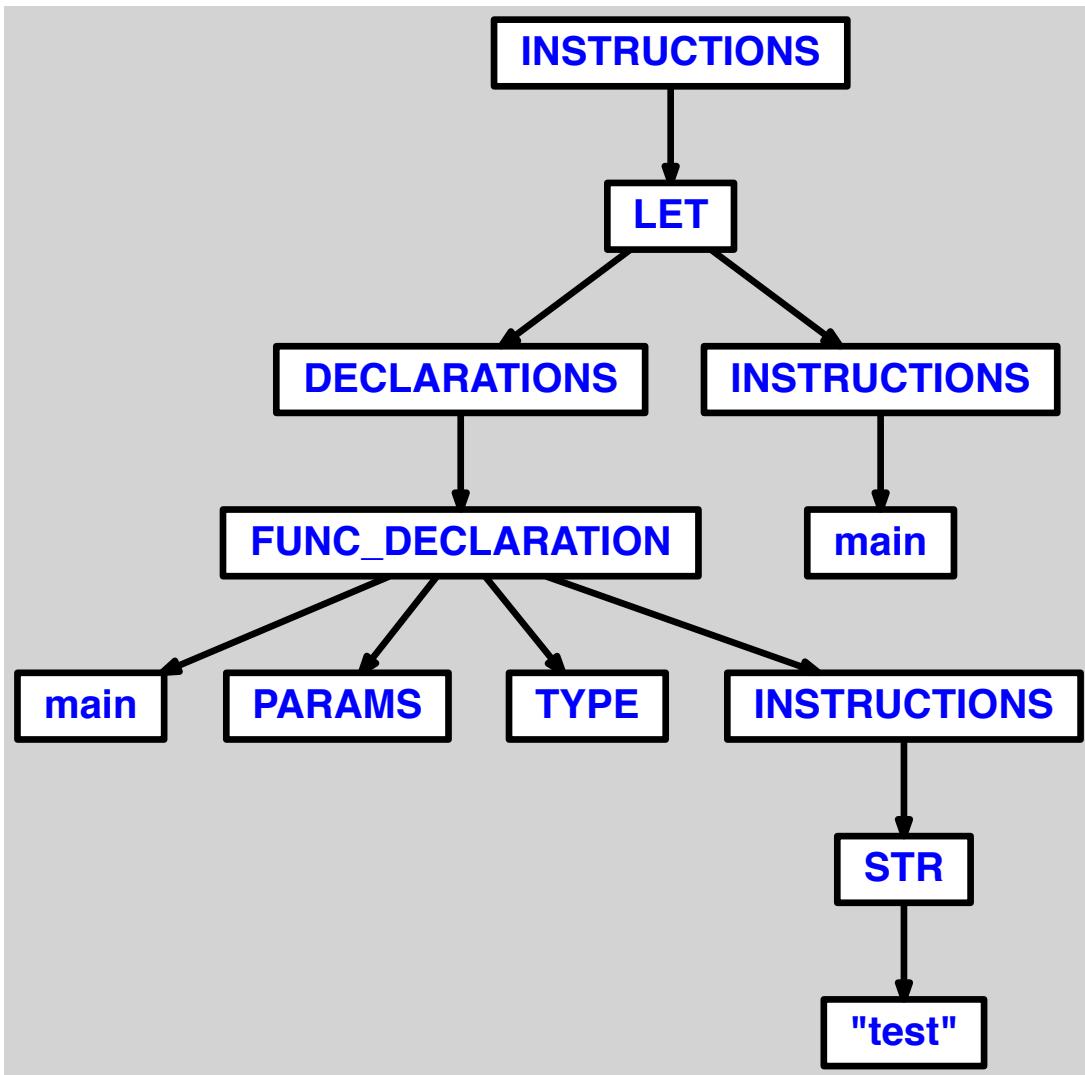
### 7.2.7 fonction avec 2 parametres entiers et 1 parametre de type string

```
let
    function main(i: int, s: string, j: int) =
        printi(i+j);
        print(s)
in main() end
```



### 7.2.8 fonction definissant une chaine

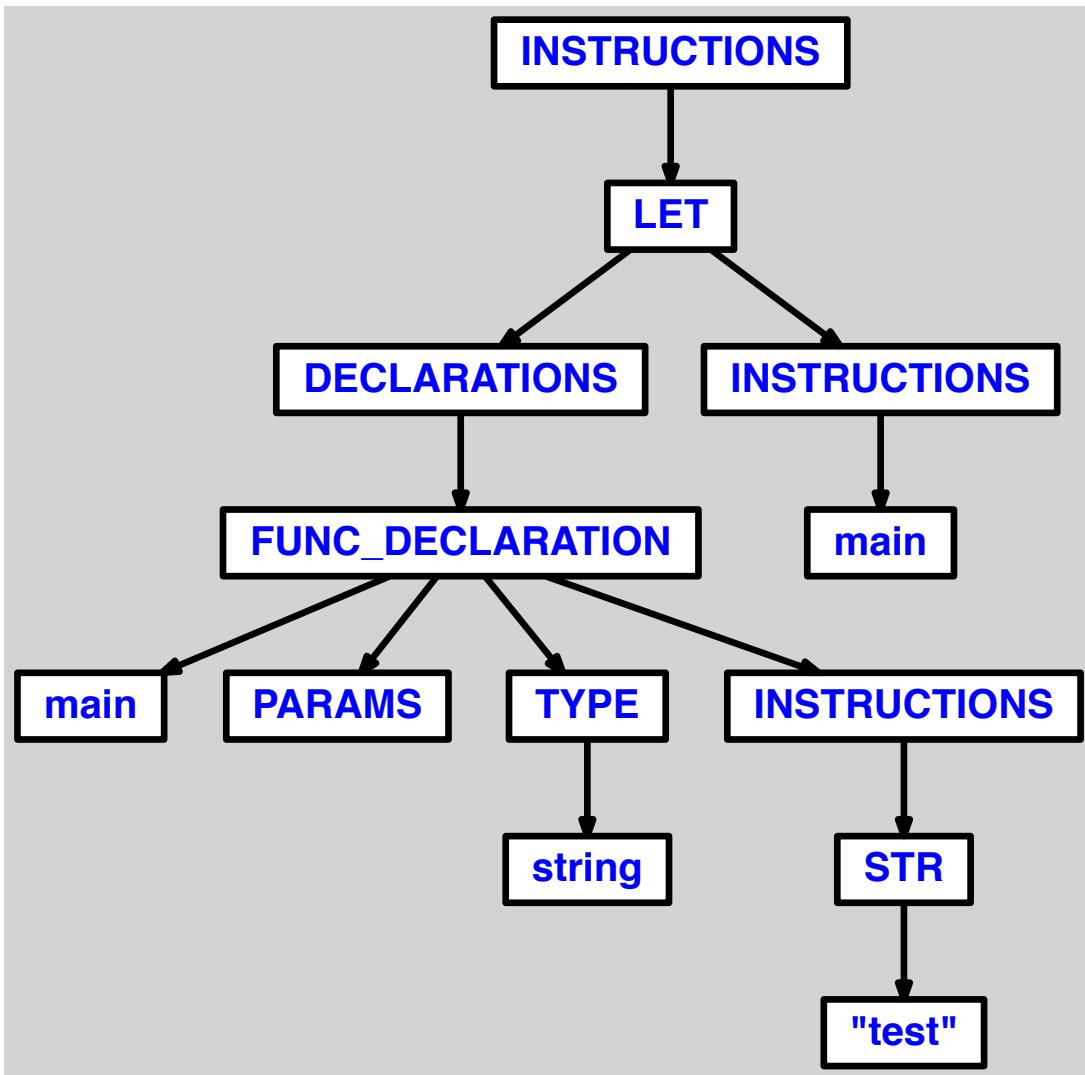
```
let
    function main() = "test"
in main() end
```



#### 7.2.9 fonction simple, avec déclaration du type de retour string et instruction de retour

```

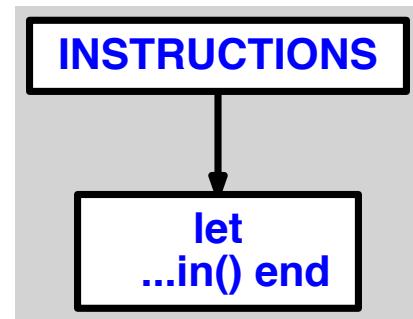
let
  function main() : string = "test"
in main() end
  
```



#### 7.2.10 fonction simple, avec instruction simple et instruction retournant une chaîne

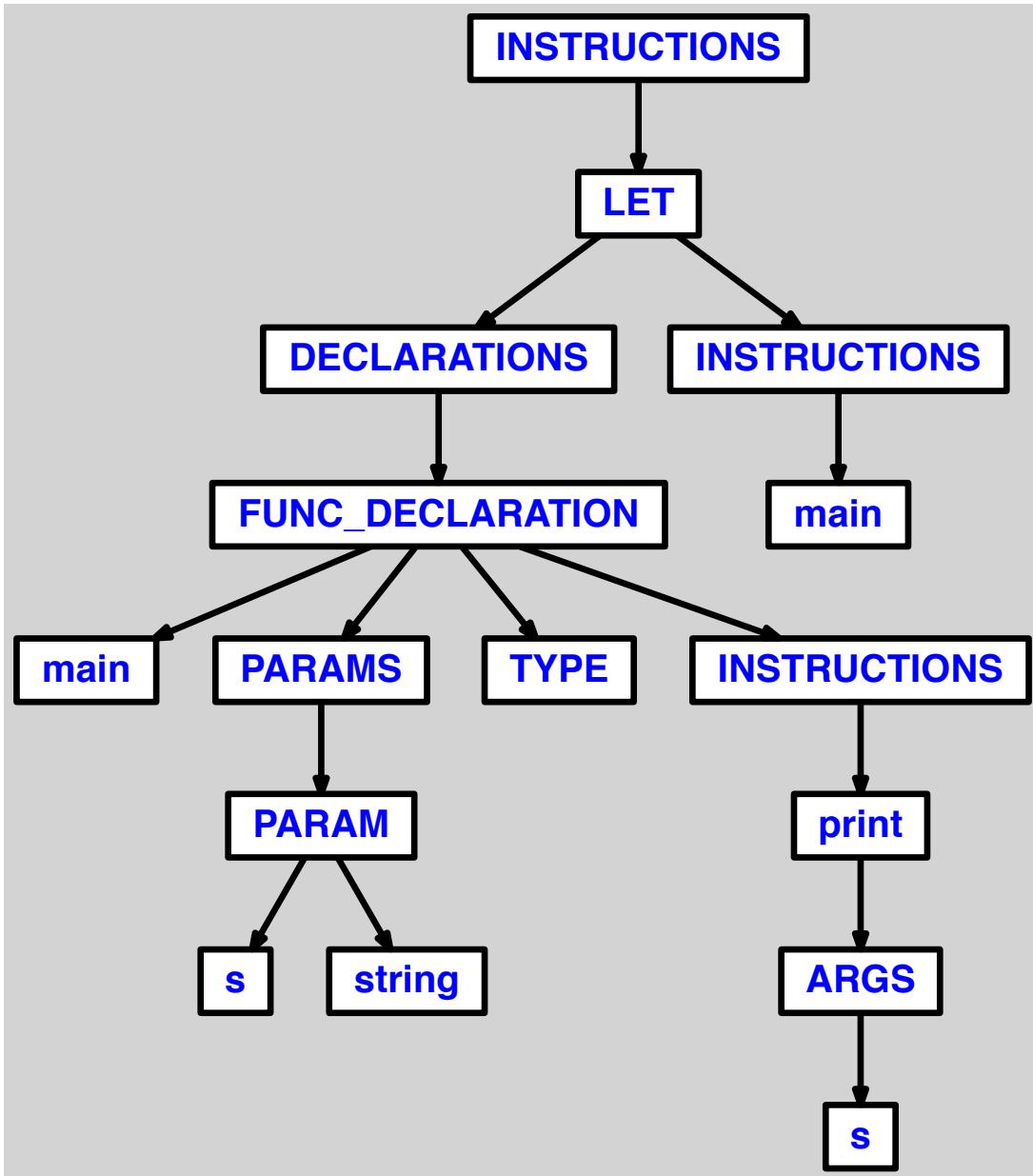
```

let
  function main() =
    print("test");
    "test"
in main() end
  
```



#### 7.2.11 fonction avec 1 parametre de type string

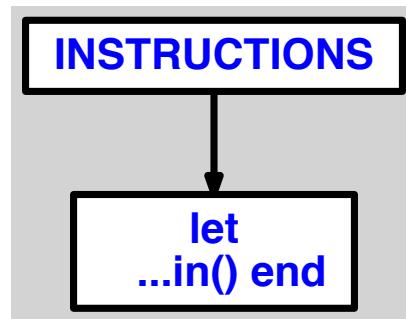
```
let
    function main(s: string) = print(s)
in main()
```



#### 7.2.12 fonction avec 2 paramètres de type string

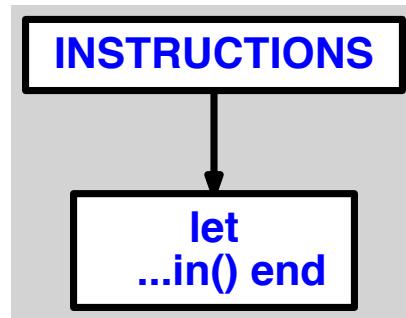
```

let
  function main(s: string, t: string) =
    print(s);
    print(t)
in main() end
  
```



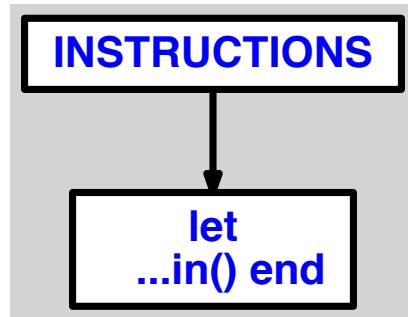
#### 7.2.13 fonction avec 3 parametres de type string

```
let
    function main(s: string, t: string, u: string) =
        print(s);
        print(t);
        print(u)
in main() end
```



#### 7.2.14 fonction avec 2 parametres de type string et 1 parametre entier

```
let
    function main(s: string, i: int, t: string) =
        print(s);
        printi(i);
        print(t)
in main() end
```



#### 7.2.15 double-imbrication de fonction

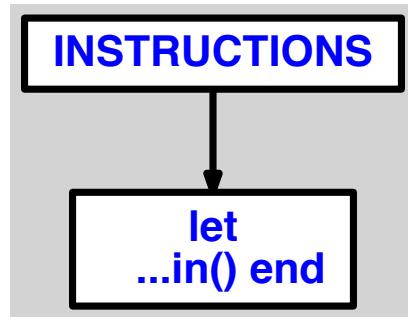
```

let
    function subFunction1(i: int, s: string) =
        print(s);
        i

    function subFunction2() = 1

    function main() = subFunction1(subFunction2(), "test")
in main() end

```



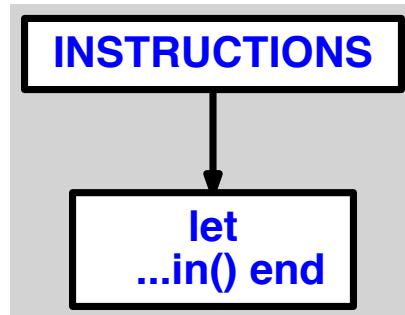
#### 7.2.16 triple-imbrication de fonction

```

let
    function subFunction1(i: int, s: string) =
        print(s);
        i

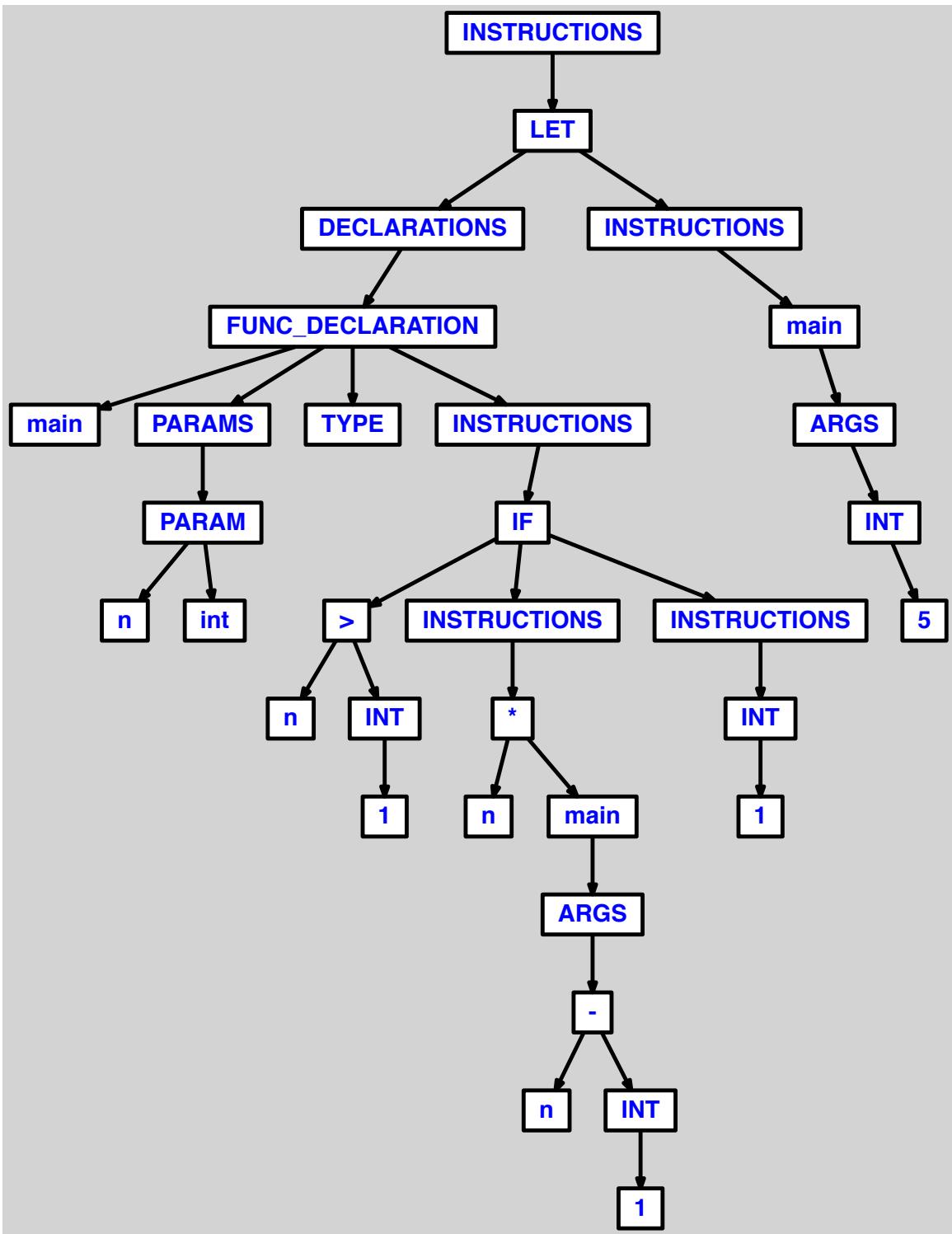
```

```
function subFunction2(s: string) = 1  
function subFonction3() = "test"  
  
function main() = subFunction1(subFonction2(subFonction3()), "test")  
in main() end
```



#### 7.2.17 recursion finie

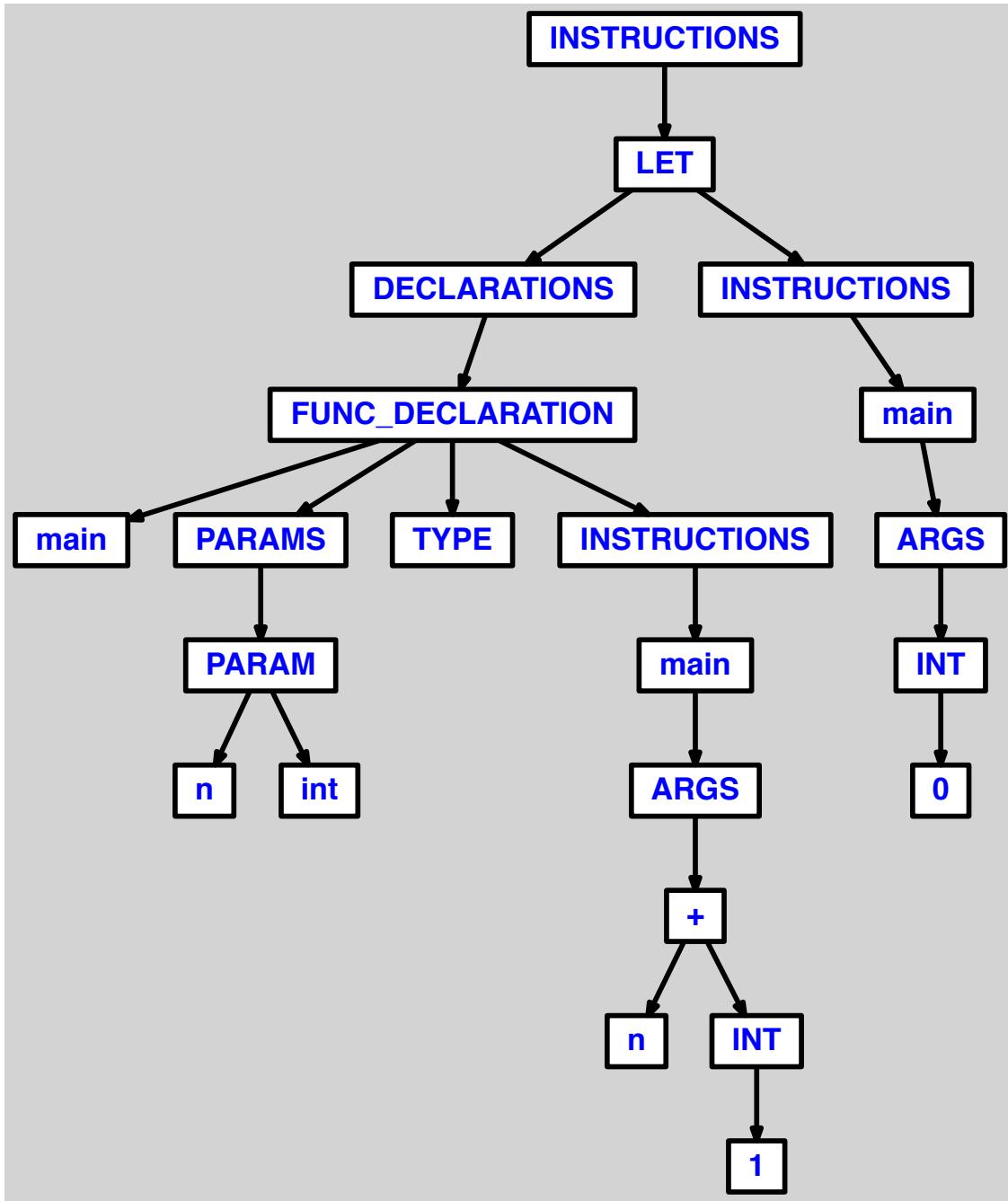
```
let  
    function main(n: int) =  
        if n > 1 then  
            n * main(n-1)  
        else 1  
in main(5) end
```



#### 7.2.18 recursion infinie

let

```
function main(n: int) = main(n+1)
in main(0) end
```



8 if

## 8.1 KO

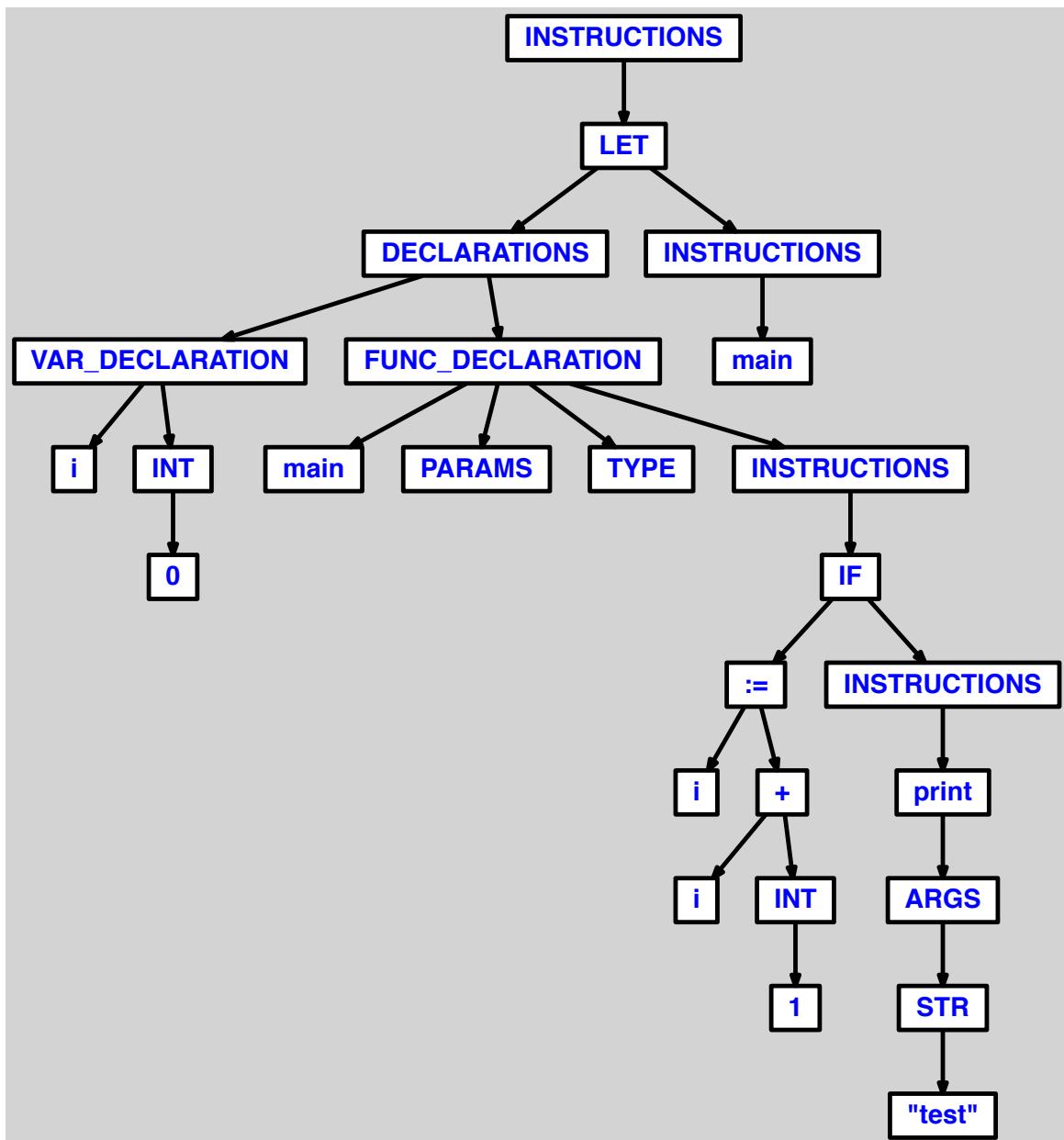
### 8.1.1 incrementation dans la condition if then

let

```
var i := 0
```

```
function main() =  
    if i := i+1 then  
        print("test")
```

```
in main() end
```

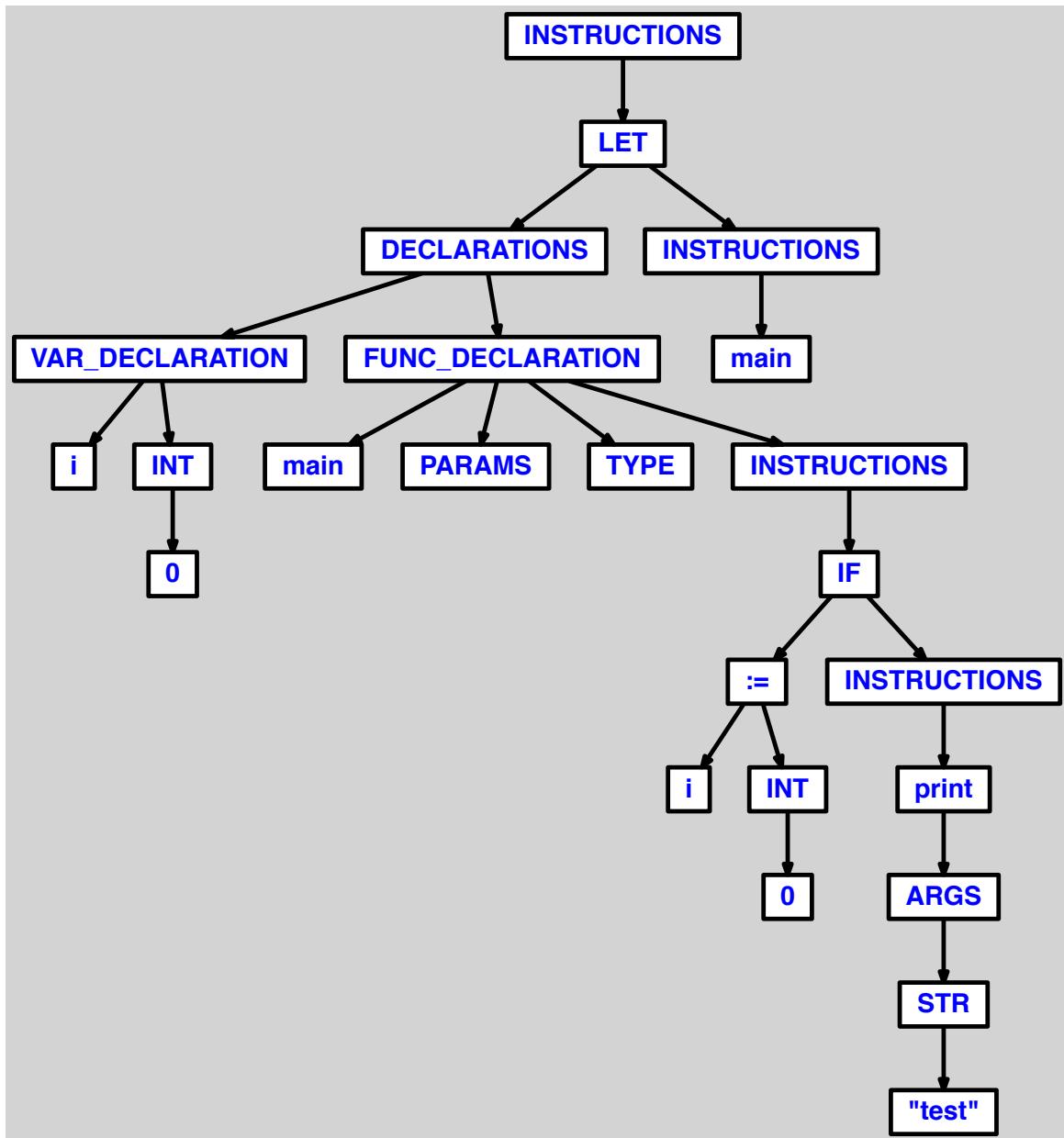


### 8.1.2 affectation d'entier dans la condition if then

```

let
  var i := 0

  function main() =
    if i := 0 then
      print("test")
in main() end
  
```

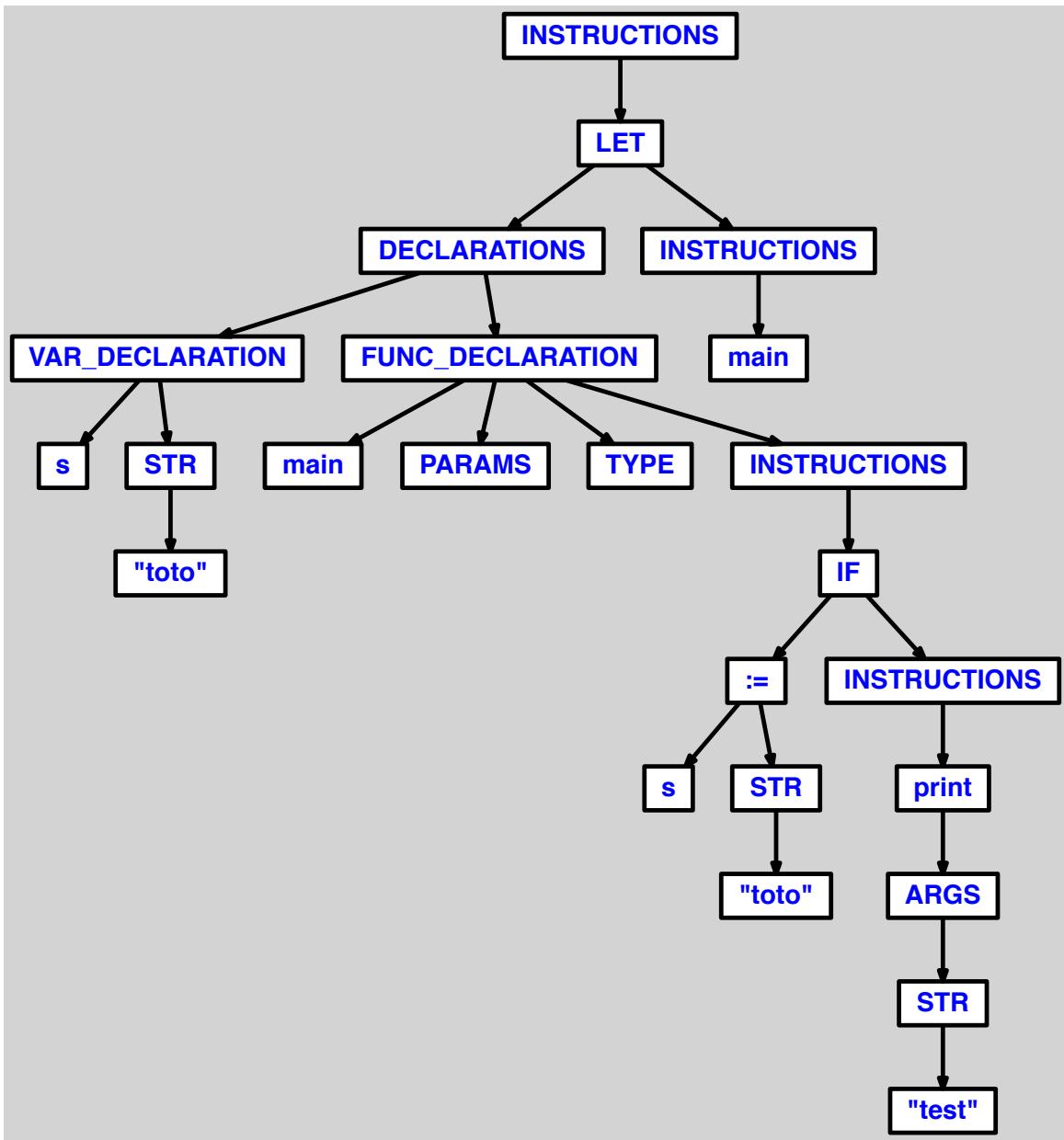


### 8.1.3 affectation de chaine dans la condition if then

```

let
  var s := "toto"

  function main() =
    if s := "toto" then
      print("test")
in main() end
  
```

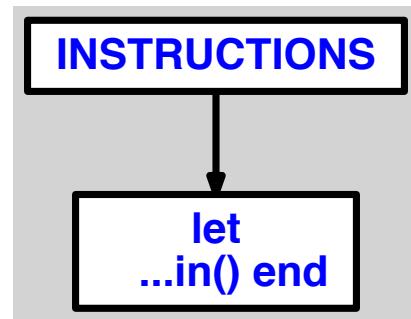


#### 8.1.4 if then avec oubli du if

```

let
  var i := 0

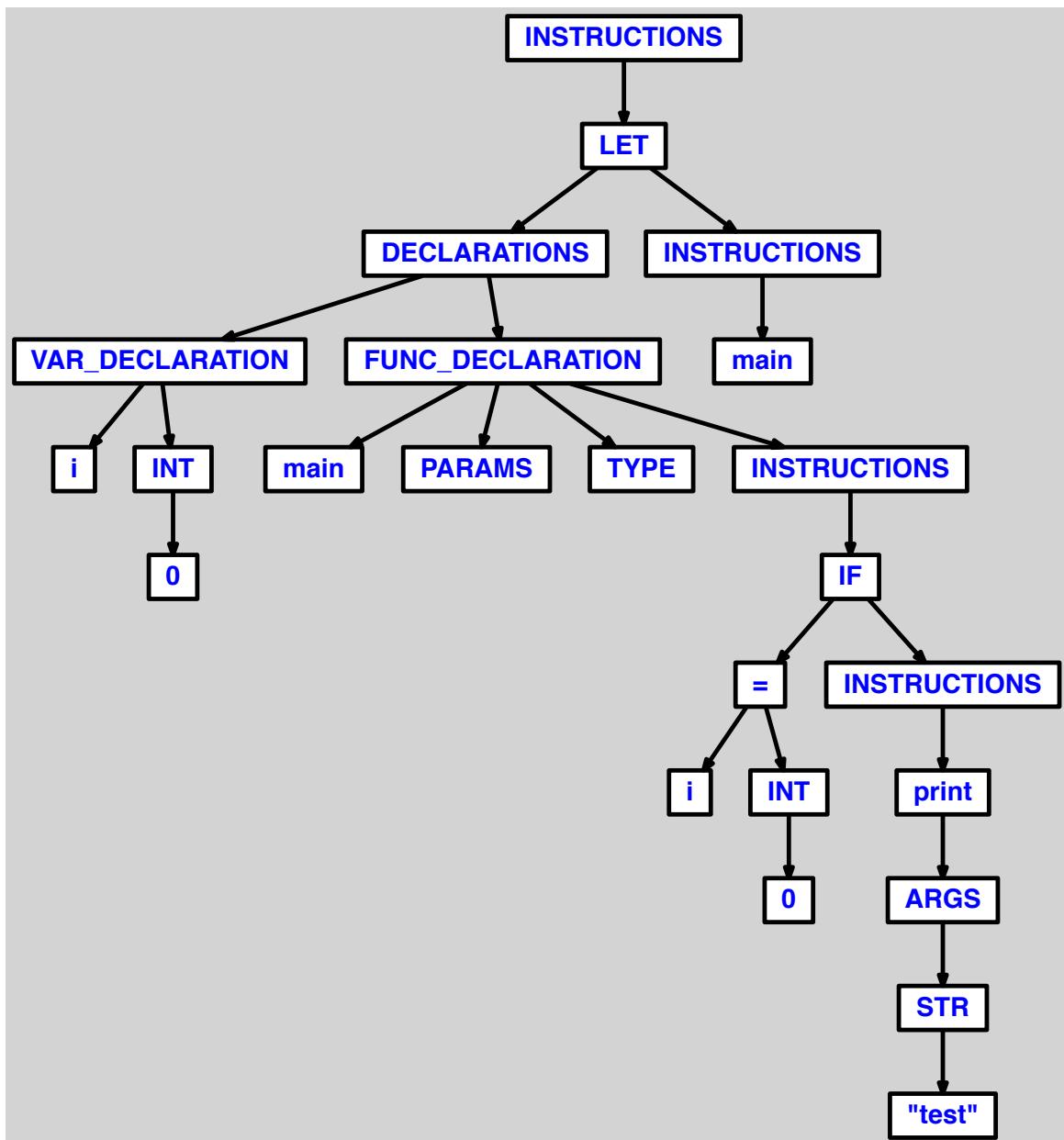
  function main() =
    i = 0 then
      print("test")
in main() end
  
```



#### 8.1.5 if then avec oubli du then

```
let
    var i := 0

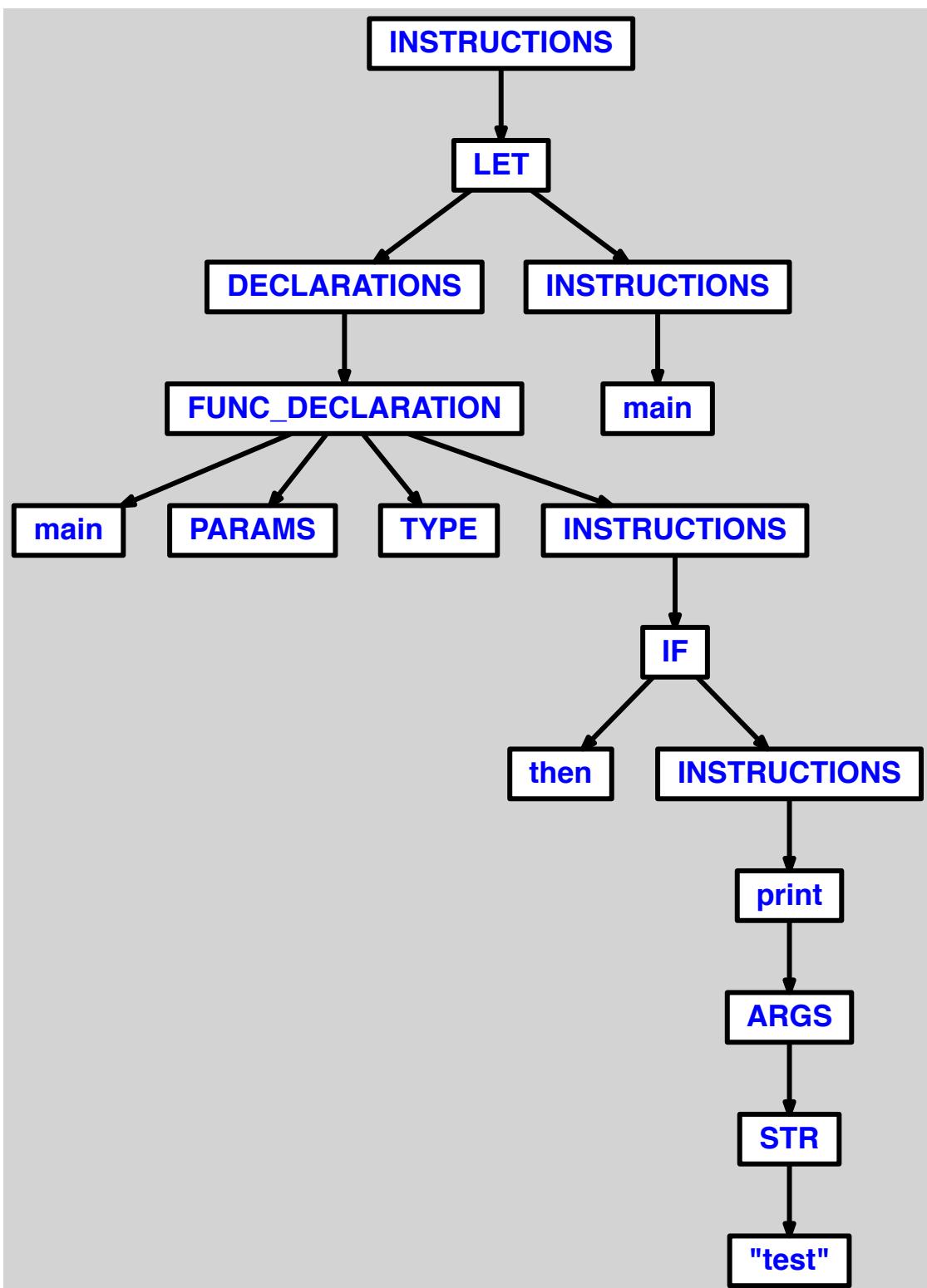
    function main() =
        if i = 0
            print("test")
in main() end
```



#### 8.1.6 if then avec oubli de la condition

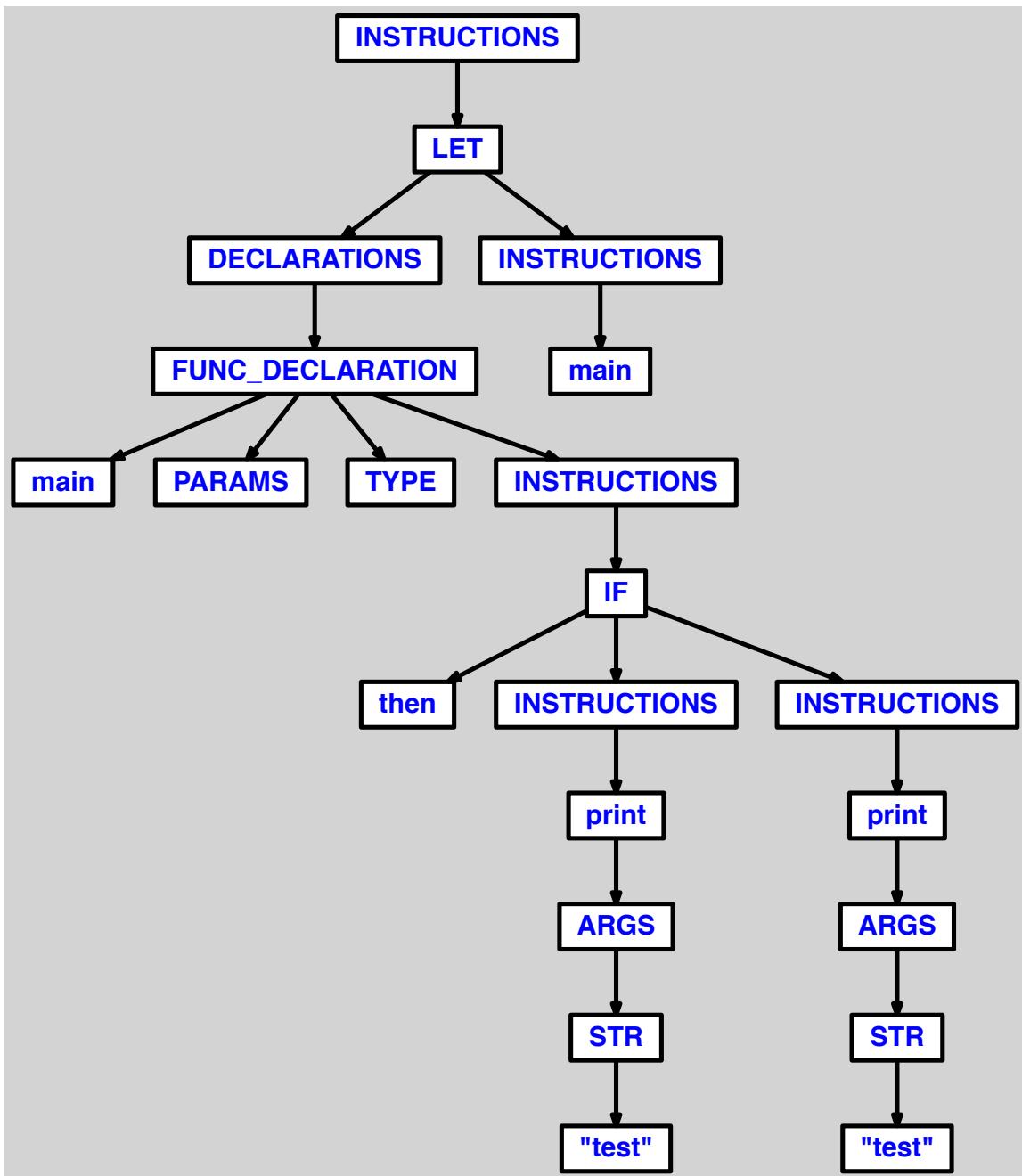
```

let
  function main() =
    if then
      print("test")
in main() end
  
```



### 8.1.7 if then else avec oubli de la condition

```
let
    function main() =
        if then
            print("test")
        else
            print("test")
in main() end
```



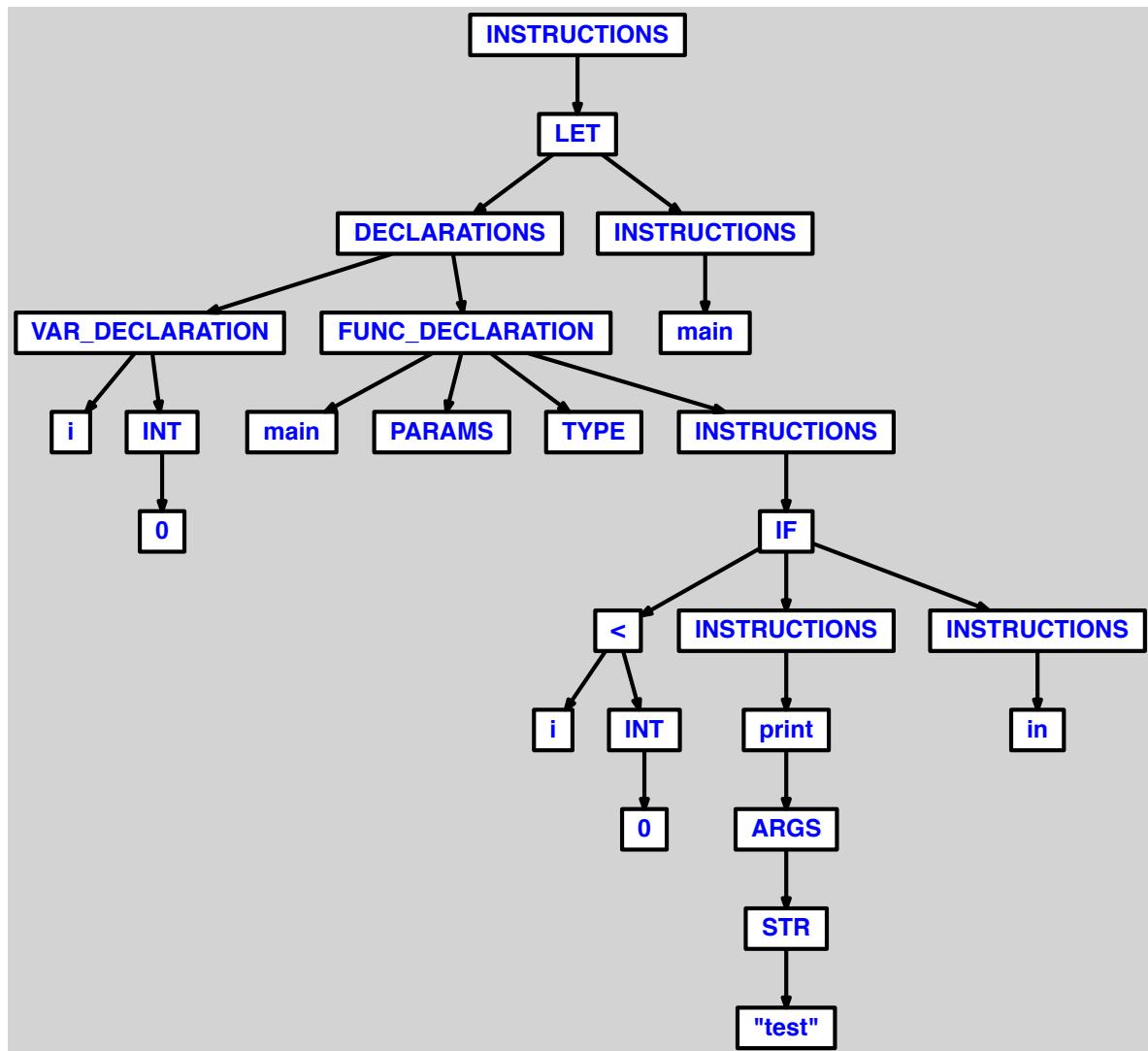
#### 8.1.8 if then else sans instruction dans else

```

let
  var i := 0

  function main() =
    if i < 0 then
      print("test")
    else
  
```

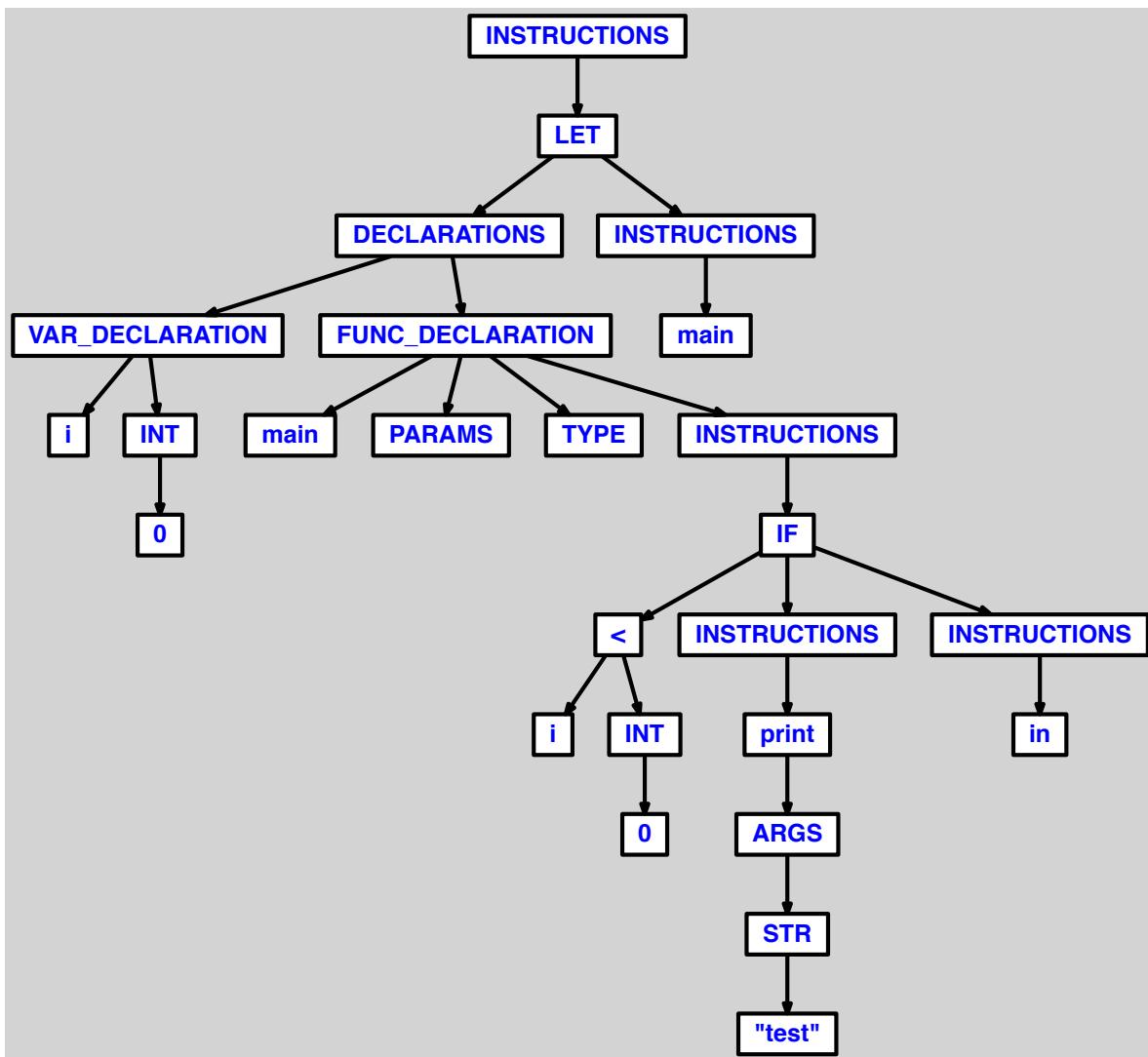
```
in main() end
```



#### 8.1.9 if then else avec ligne vide dans else

```
let
    var i := 0

    function main() =
        if i < 0 then
            print("test")
        else
            in main() end
```



## 8.2 OK

### 8.2.1 if then avec condition entiere

```

let
    function main()() =
        if 1 then
            print("test")
in main() end

```

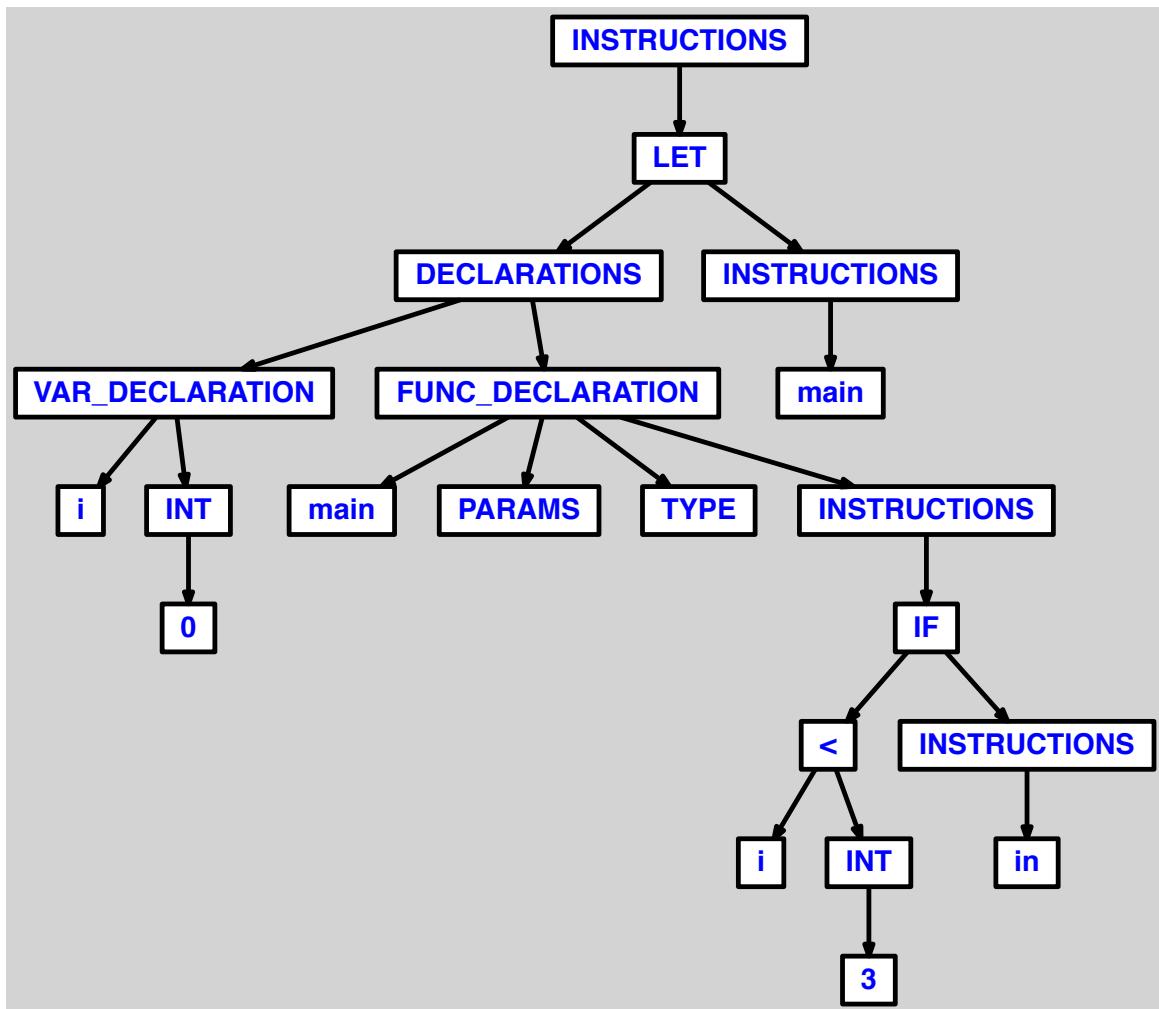
### 8.2.2 if then sans instruction

```

let
    var i := 0

    function main() =
        if i < 3 then
in main() end

```



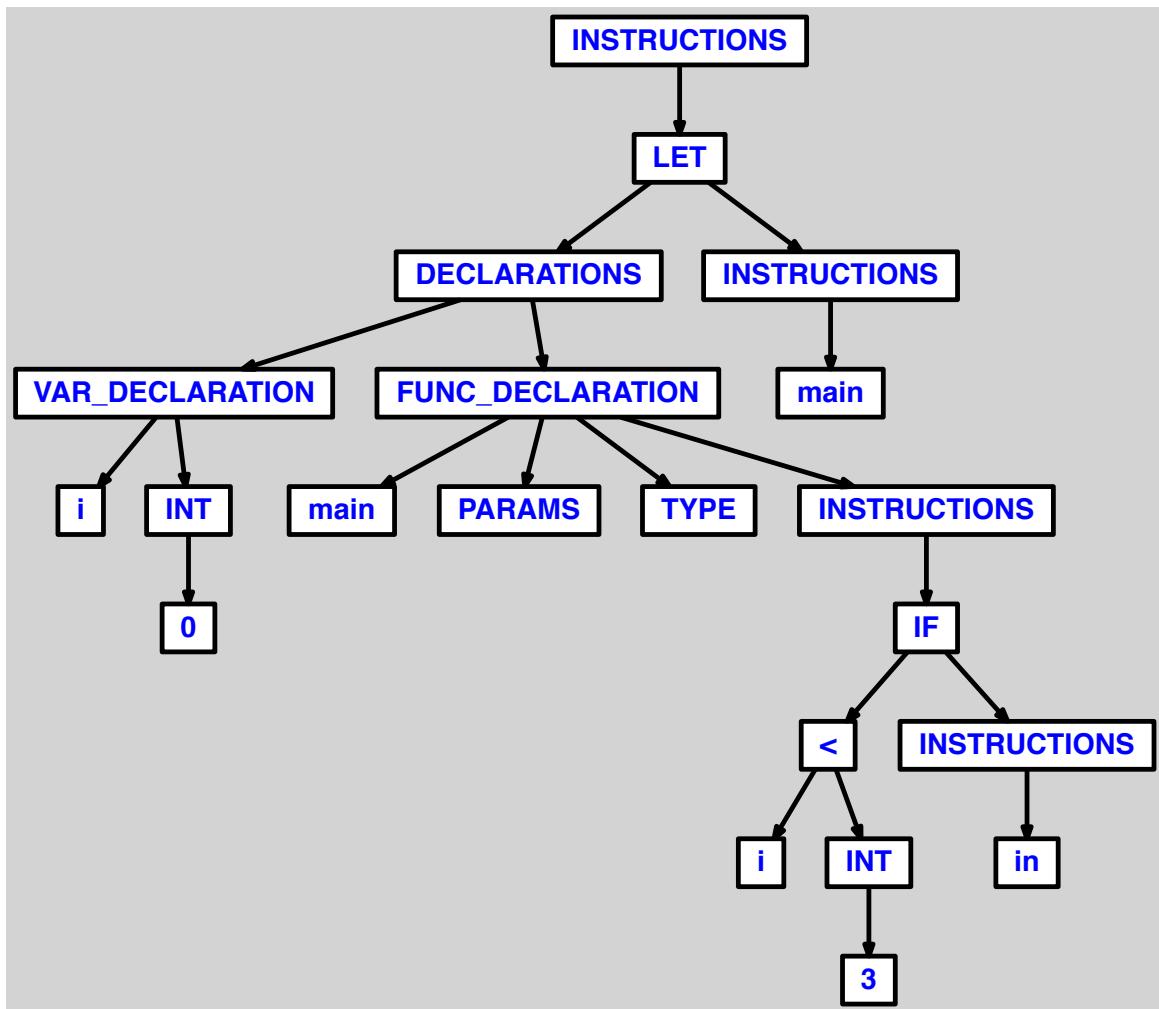
### 8.2.3 if then avec ligne vide

```

let
  var i := 0

  function main() =
    if i < 3 then

in main() end
  
```

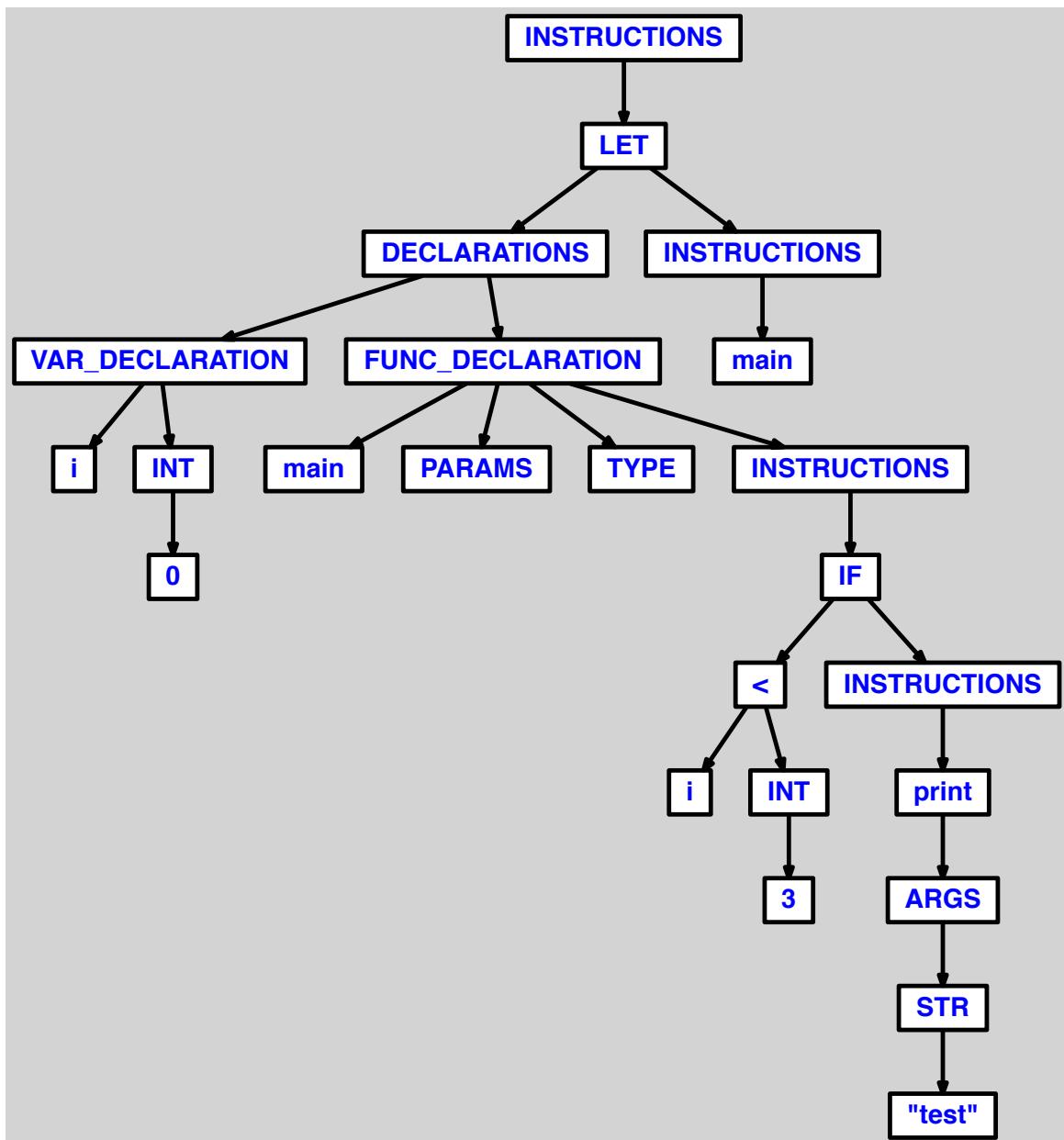


#### 8.2.4 if then avec simple condition

```

let
  var i := 0

  function main() =
    if i < 3 then
      print("test")
in main() end
  
```

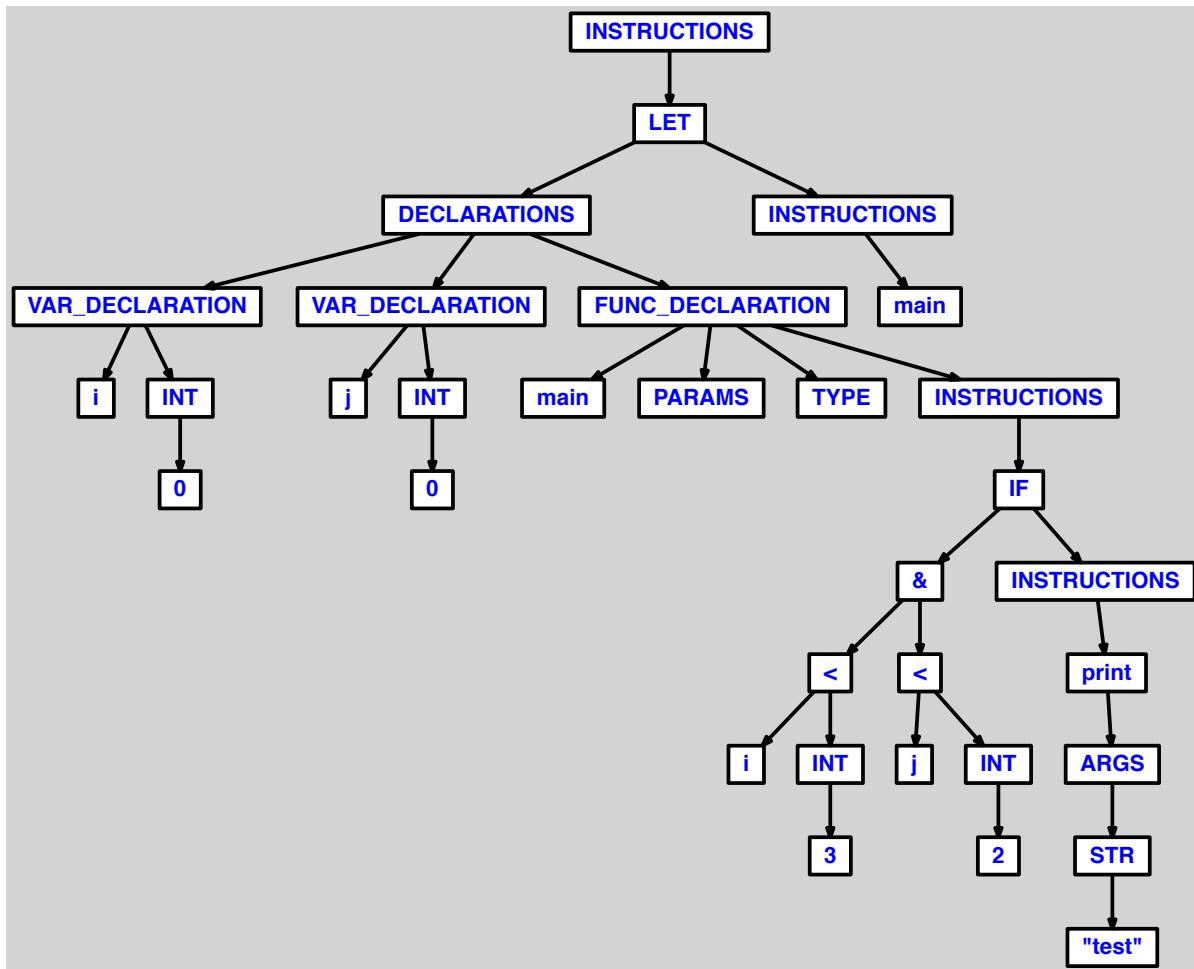


#### 8.2.5 if then avec double-condition

```

let
  var i := 0
  var j := 0

  function main() =
    if i < 3 & j < 2 then
      print("test")
in main() end
  
```

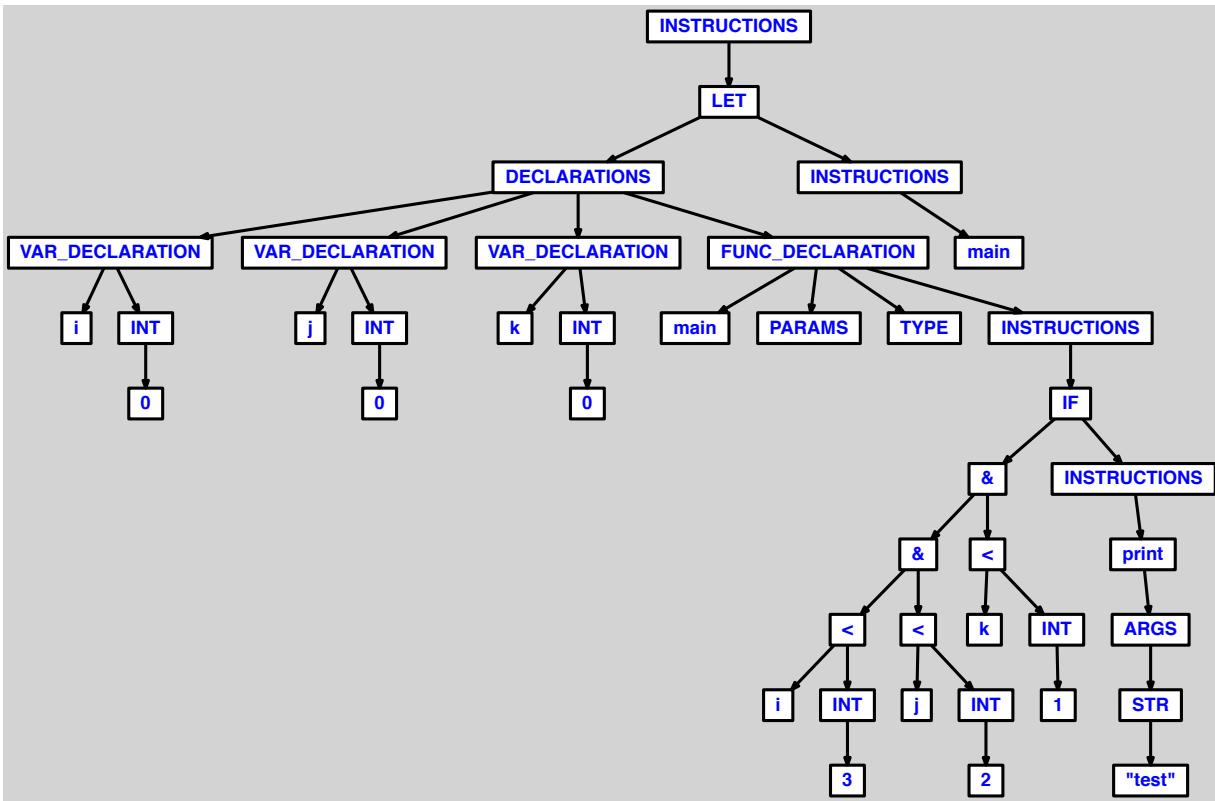


#### 8.2.6 if then avec triple-condition

```

let
  var i := 0
  var j := 0
  var k := 0

  function main() =
    if i < 3 & j < 2 & k < 1 then
      print("test")
in main() end
  
```



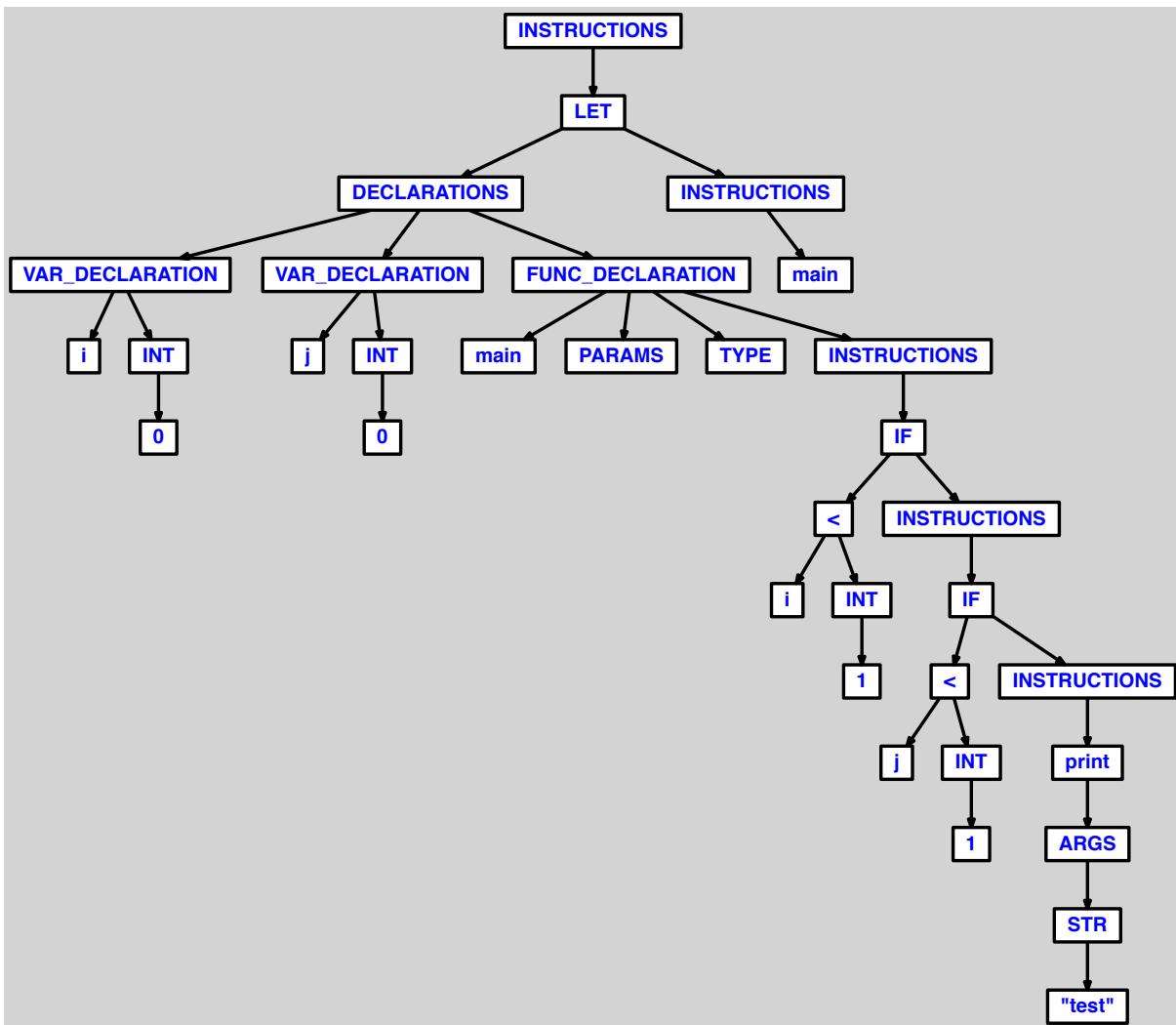
### 8.2.7 double-imbrication de if then

```

let
    var i := 0
    var j := 0

    function main() =
        if i < 1 then
            if j < 1 then
                print("test")
in main() end

```

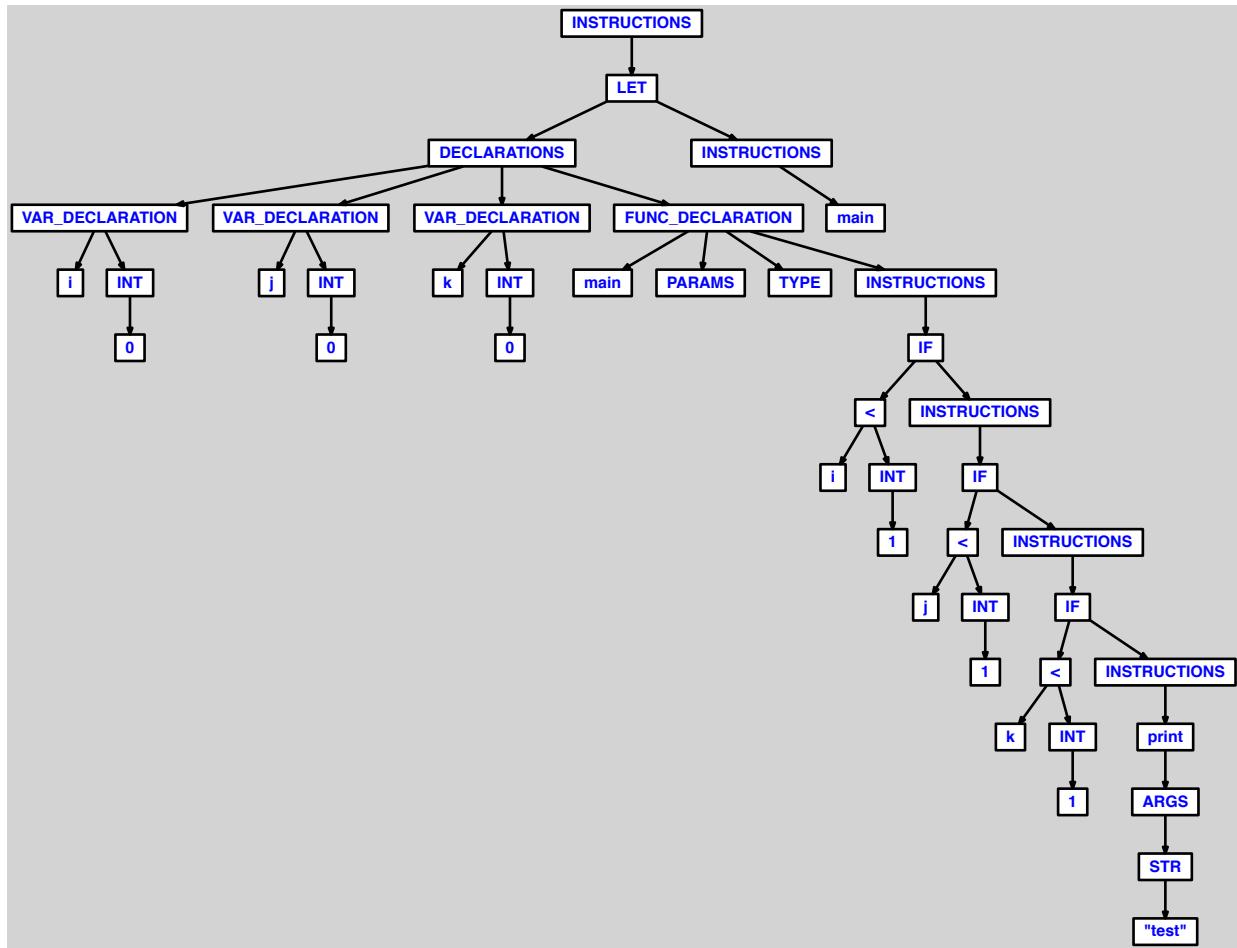


### 8.2.8 triple-imbrication de if then

```

let
  var i := 0
  var j := 0
  var k := 0

  function main() =
    if i < 1 then
      if j < 1 then
        if k < 1 then
          print("test")
in main() end
  
```



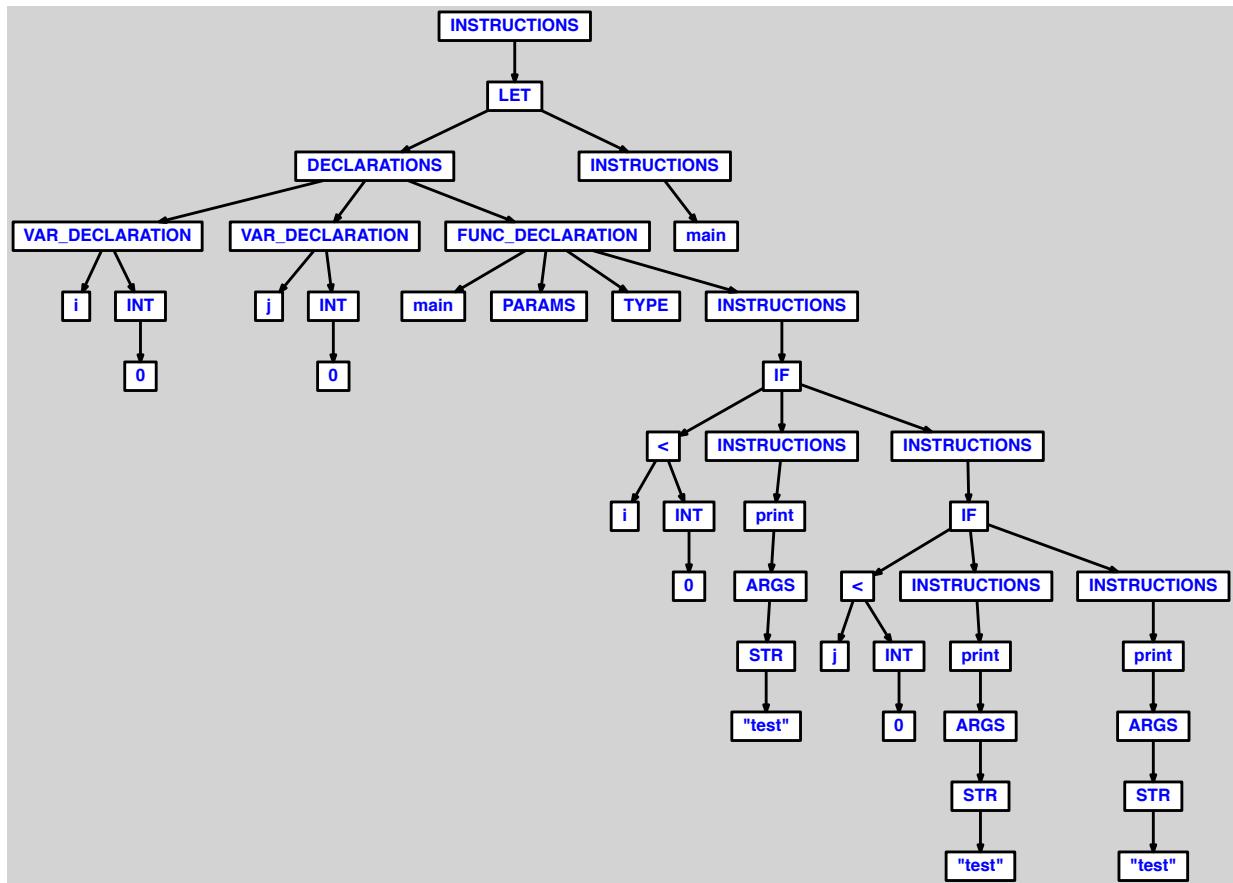
### 8.2.9 double-imbrication de if then else

```

let
    var i := 0
    var j := 0

    function main() =
        if i < 0 then
            print("test")
        else
            if j < 0 then
                print("test")
            else
                print("test")
in main() end

```



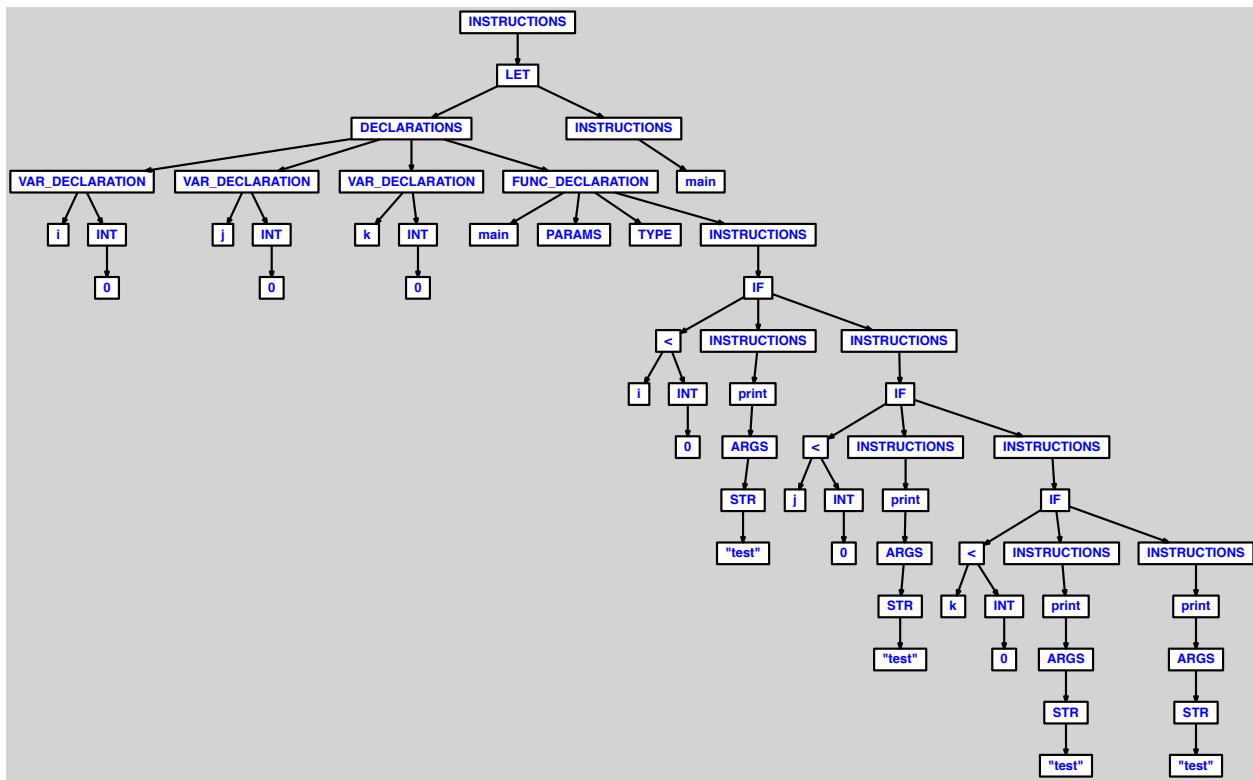
### 8.2.10 triple-imbrication de if then else

```

let
    var i := 0
    var j := 0
    var k := 0

    function main() =
        if i < 0 then
            print("test")
        else
            if j < 0 then
                print("test")
            else
                if k < 0 then
                    print("test")
                else
                    print("test")
in main() end

```



## 9 let

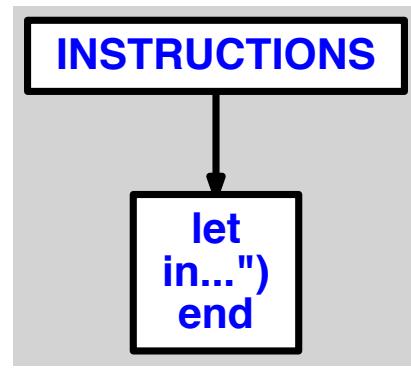
### 9.1 KO

#### 9.1.1 let sans declaration

```

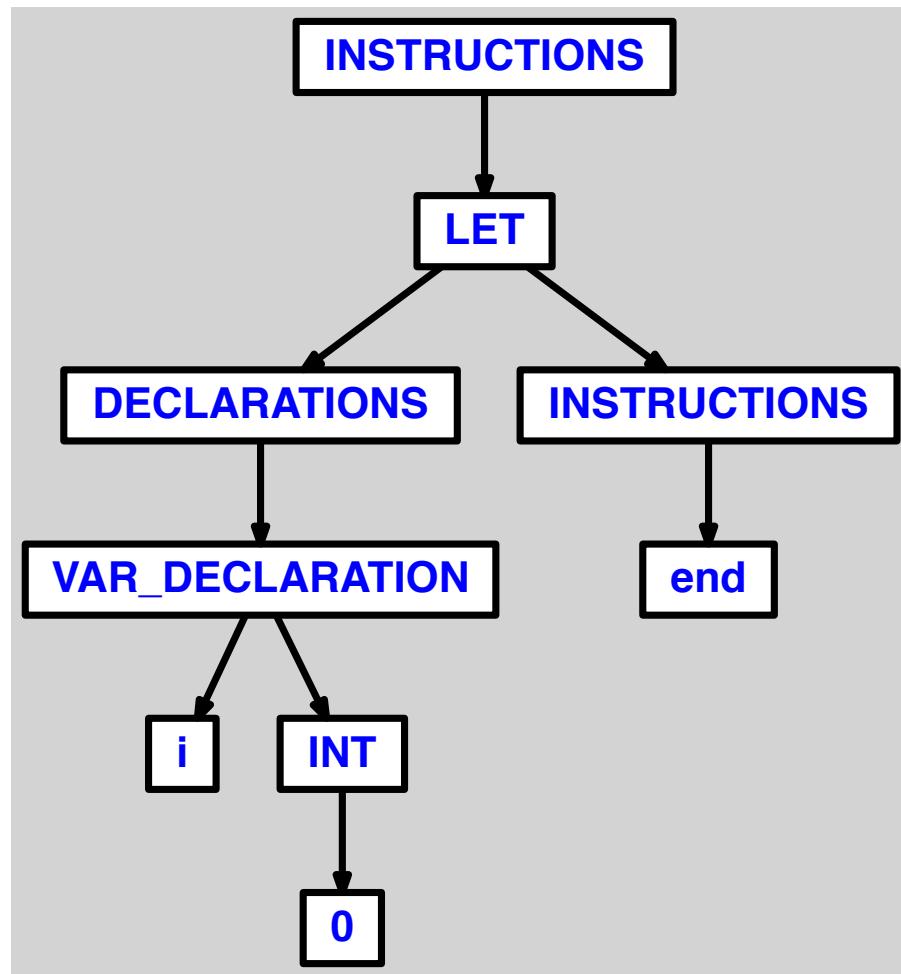
let
in
    print("test")
end

```



### 9.1.2 let sans instruction

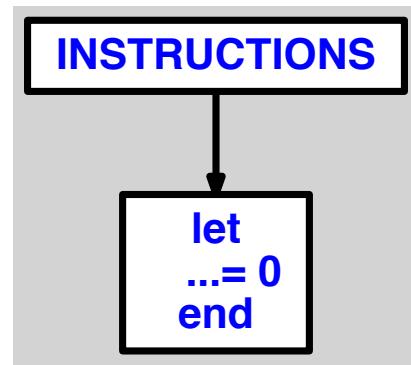
```
let
    var i := 0
in
end
```



#### 9.1.3 let avec inversion de declaration et instruction

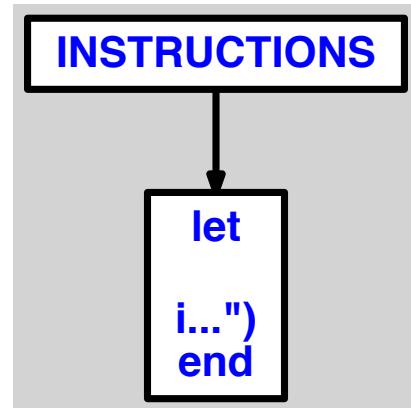
```

let
  print(i)
in
  var i := 0
end
  
```



#### 9.1.4 let avec déclaration vide

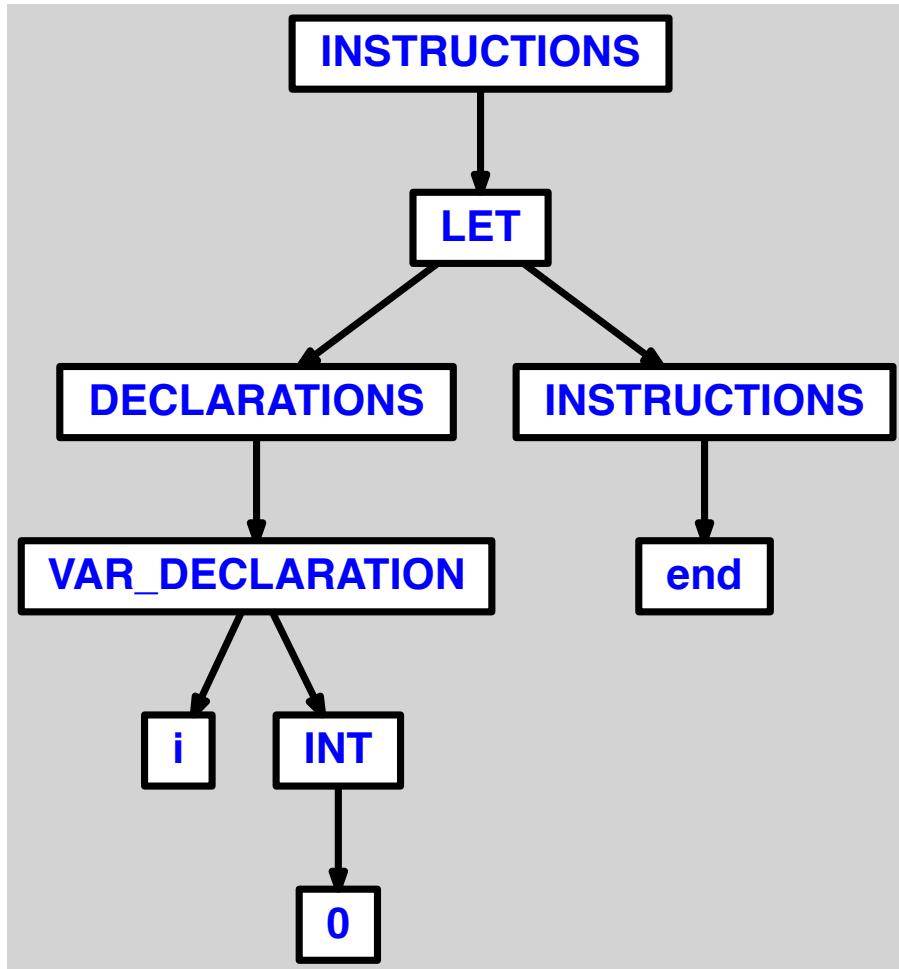
```
let  
in  
    print("test")  
end
```



#### 9.1.5 let avec instruction vide

```
let  
    var i := 0  
in
```

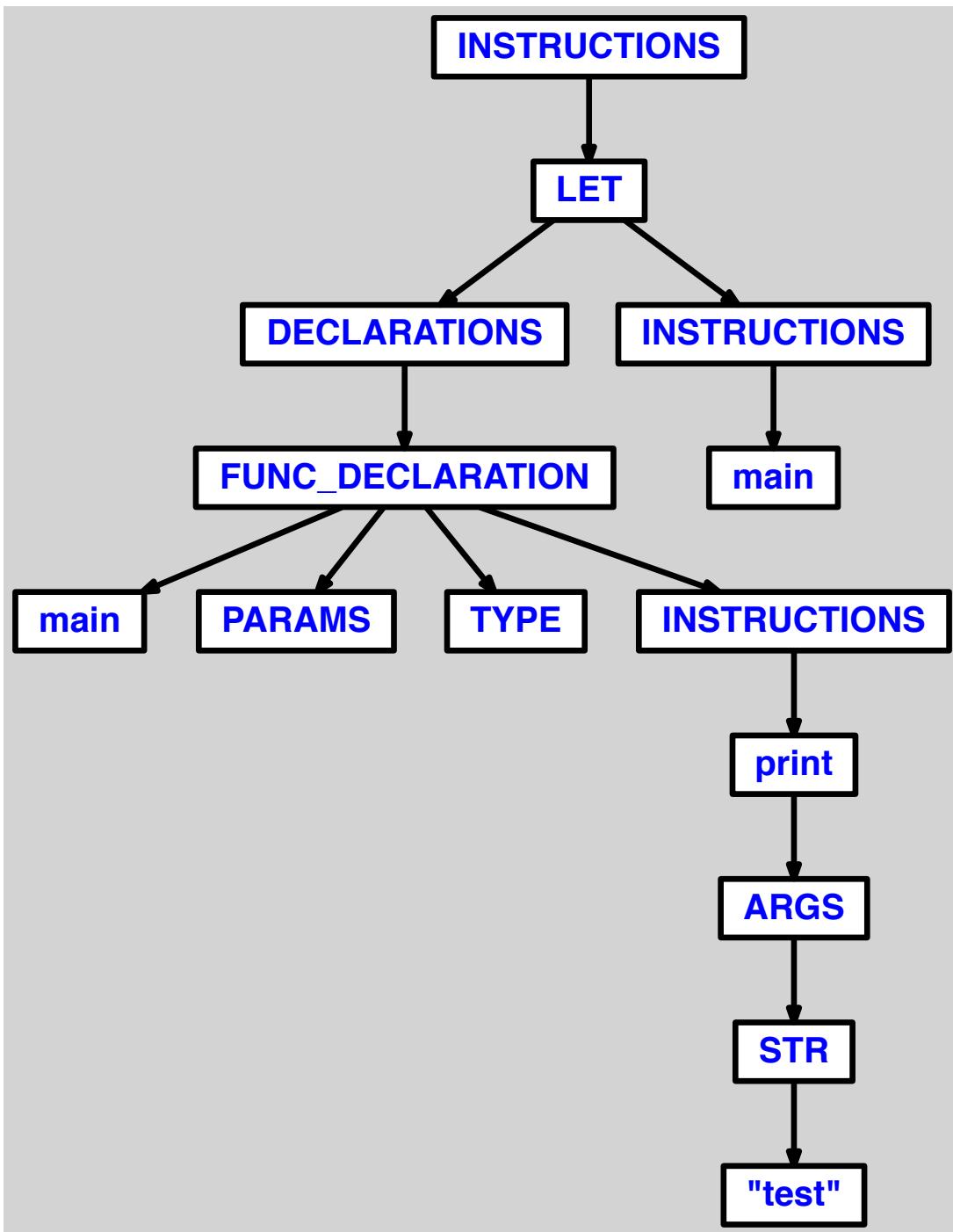
end



## 9.2 OK

### 9.2.1 let avec 1 declaration de fonction et 1 appel de fonction

```
let
    function main() = print("test")
in main() end
```

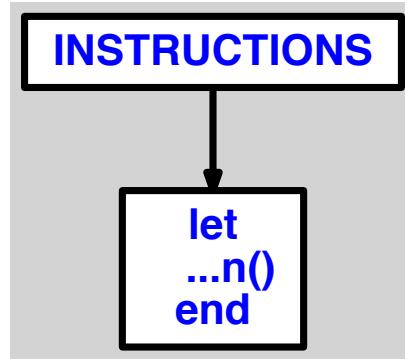


#### 9.2.2 let avec 2 déclarations de fonction et 2 appels de fonction

```

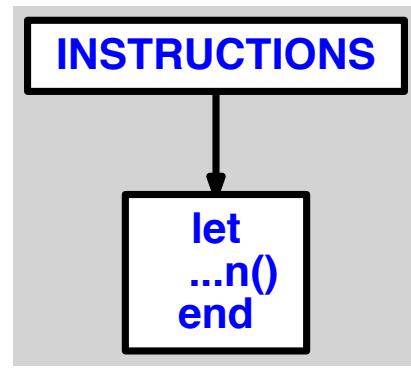
let
  function test() = print("test")
  
```

```
function main() = test()  
in  
    test()  
    main()  
end
```



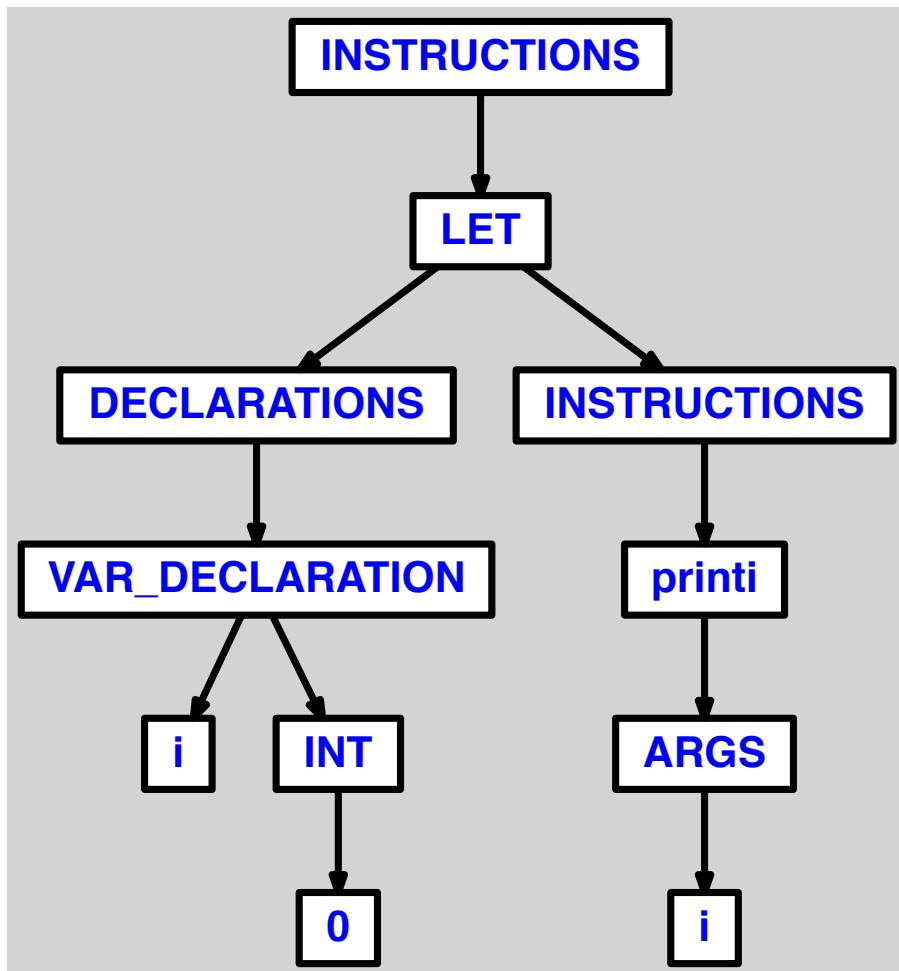
### 9.2.3 let avec 3 déclarations de fonction et 3 appels de fonction

```
let  
    function test1() = print("test")  
  
    function test2() = test1()  
  
    function main() = test2()  
in  
    test1()  
    test2()  
    main()  
end
```



#### 9.2.4 let avec 1 déclaration de variable et 1 instruction

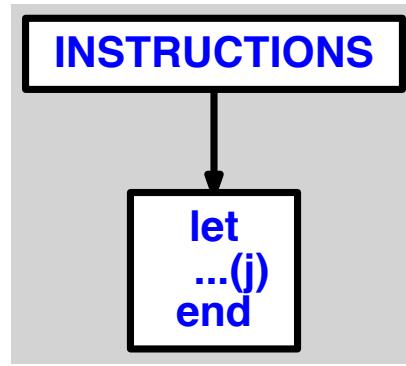
```
let
    var i := 0
in
    printi(i)
end
```



#### 9.2.5 let avec 2 declarations de variable et 2 instructions

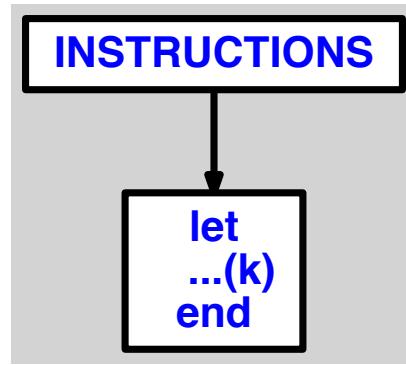
```

let
    var i := 0
    var j := 0
in
    printi(i)
    printi(j)
end
  
```



#### 9.2.6 let avec 3 déclarations de variable et 3 instructions

```
let
    var i := 0
    var j := 0
    var k := 0
in
    printi(i)
    printi(j)
    printi(k)
end
```



#### 9.2.7 let avec 3 déclarations de variable et fonction, et 3 instructions et appels de fonction

```
let
    var i := 0
```

```

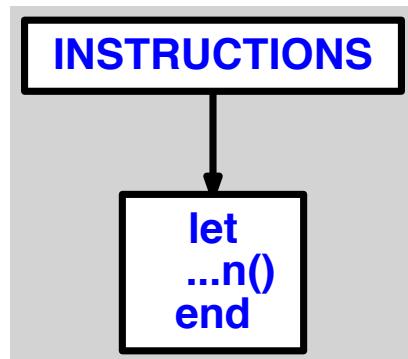
var j := 0
var k := 0

function test1() = print("test")

function test2() = test1()

function main() = test2()
in
printi(i)
printi(j)
printi(k)
test1()
test2()
main()
end

```

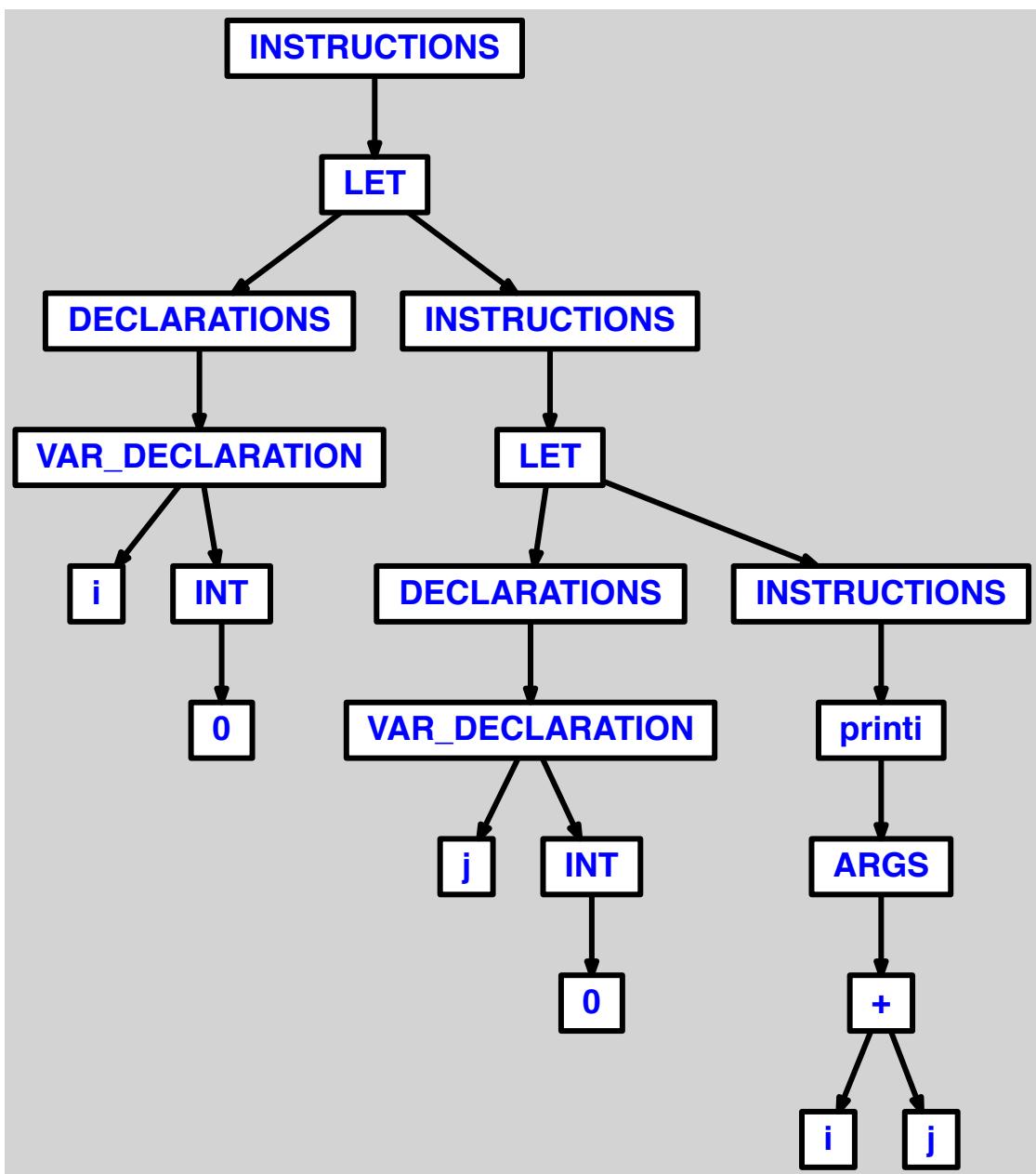


#### 9.2.8 let avec double-imbrication

```

let
    var i := 0
in
    let
        var j := 0
    in
        printi(i+j)
    end
end

```



#### 9.2.9 let avec triple-imbrication

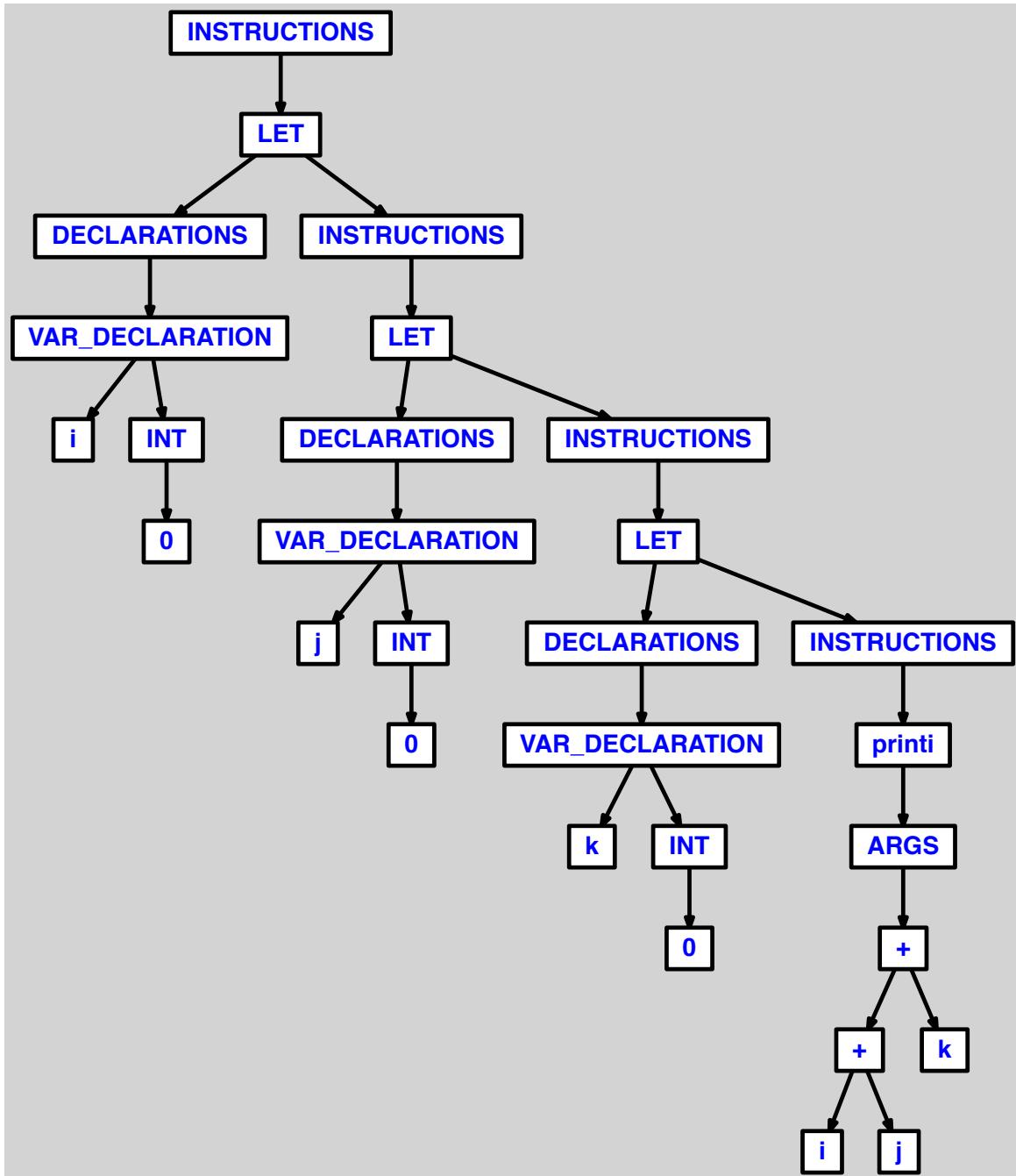
```

let
  var i := 0
in
let
  var j := 0
in
let
  
```

```

var k := 0
in
printi(i+j+k)
end
end

```



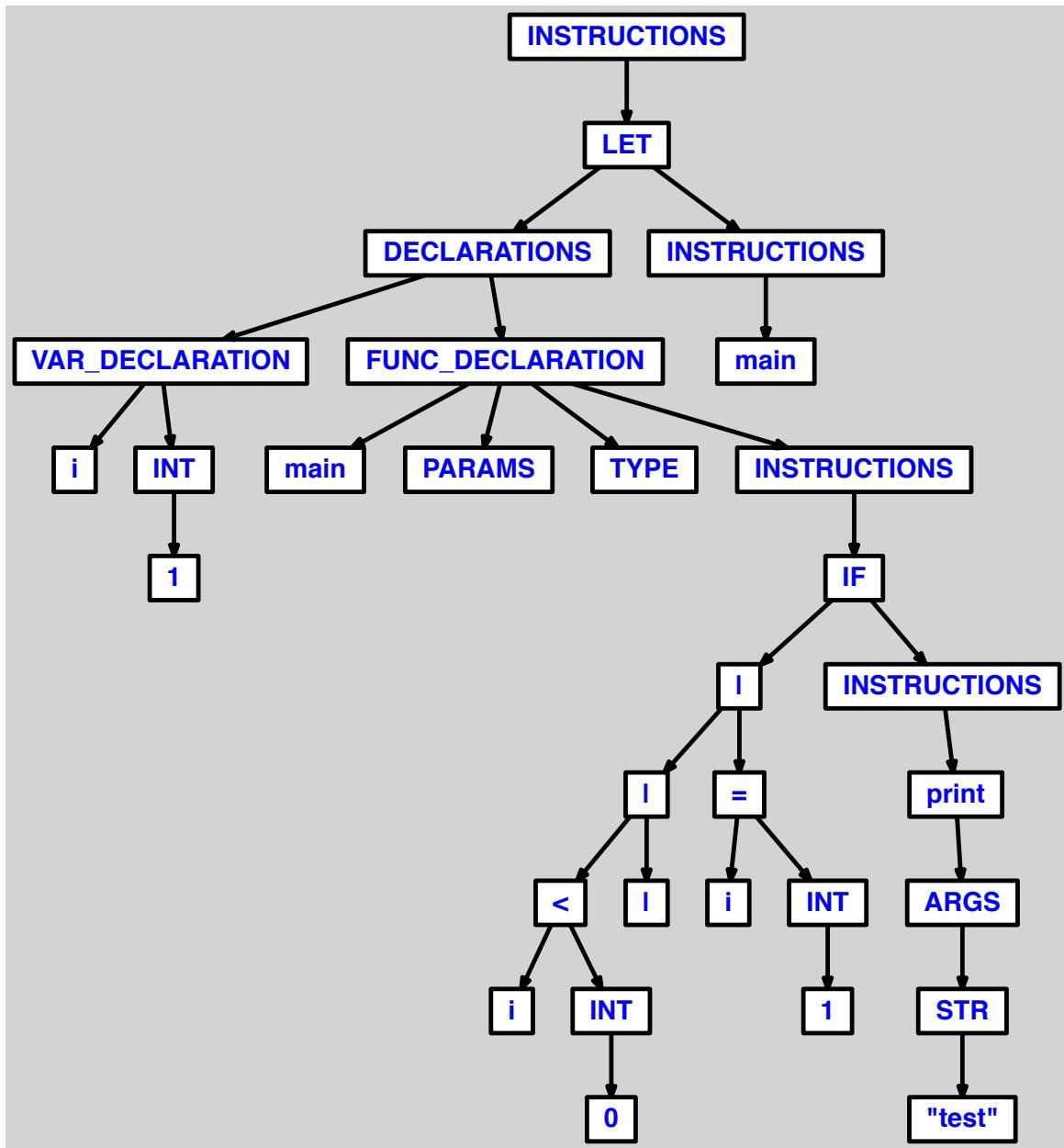
**10 or**

## 10.1 KO

### 10.1.1 | mal écrit

```
let
    var i := 1

    function main() =
        if i < 0 || i = 1 then print("test")
in main() end
```



## 10.2 OK

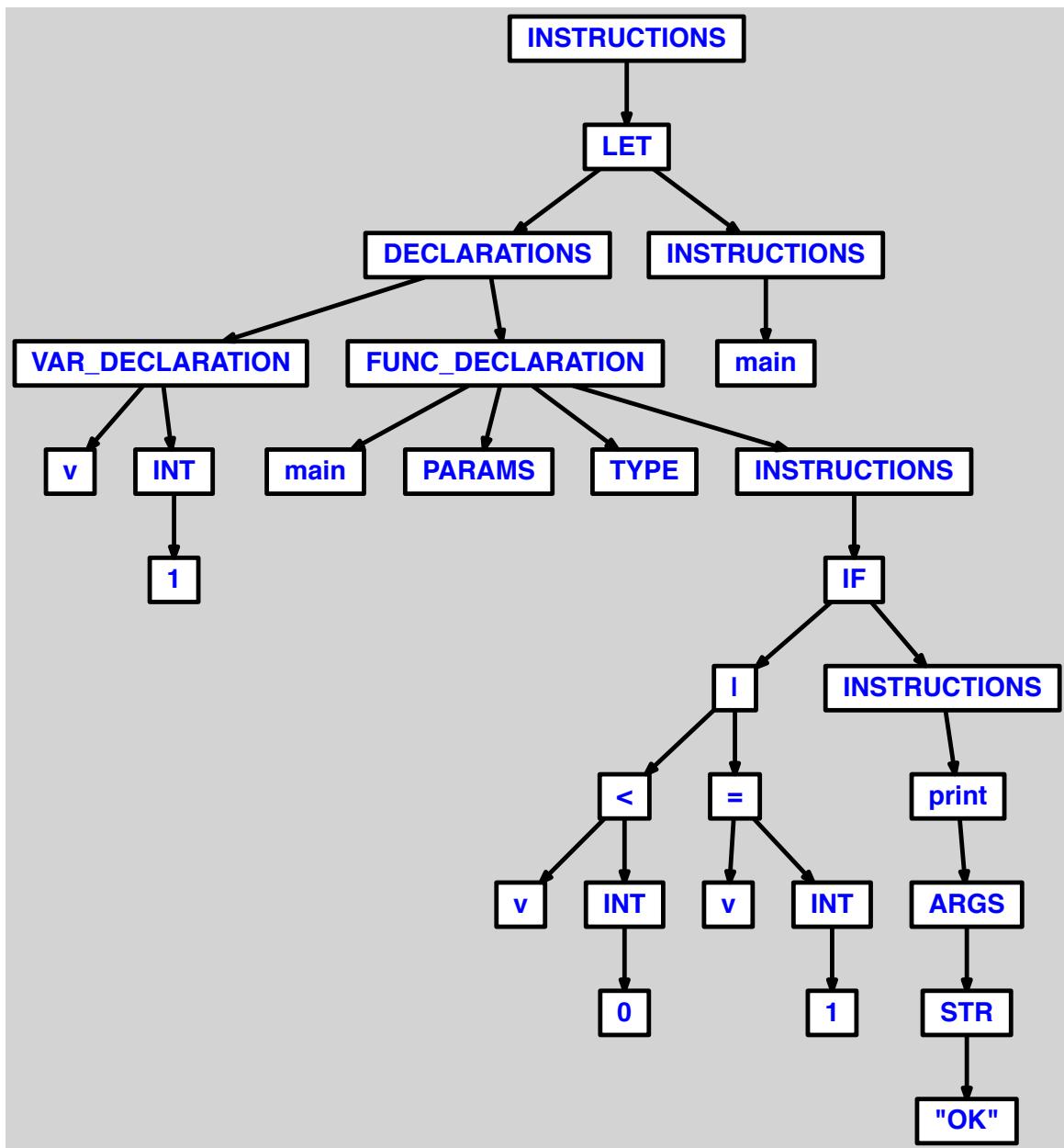
### 10.2.1 condition avec 1 |

```

let
    var v := 1

    function main() =
        if v < 0 | v = 1 then print("OK")
in main() end

```

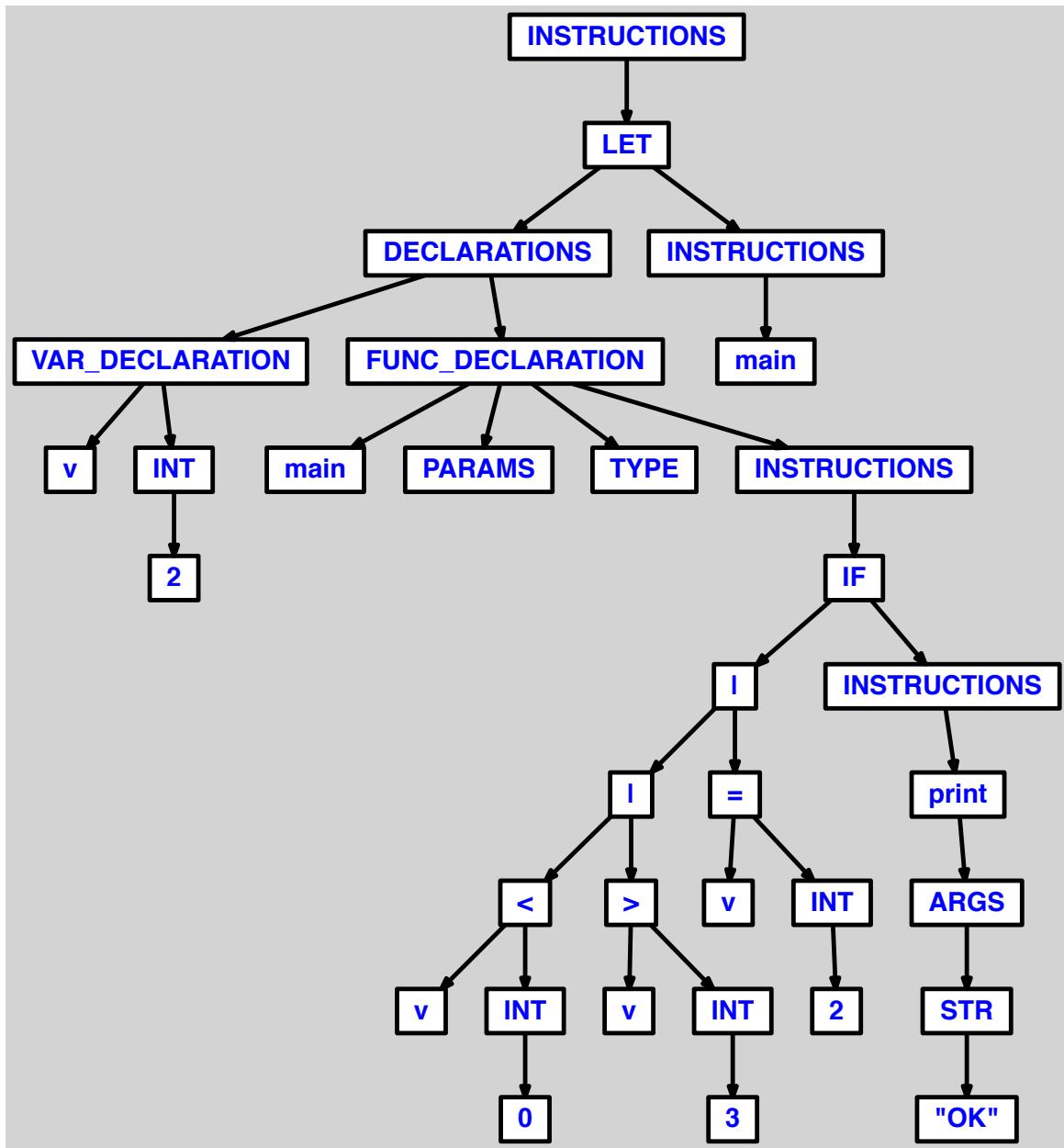


### 10.2.2 condition avec 2 |

```

let
  var v := 2

  function main() =
    if v < 0 | v > 3 | v = 2 then print("OK")
in main() end
  
```

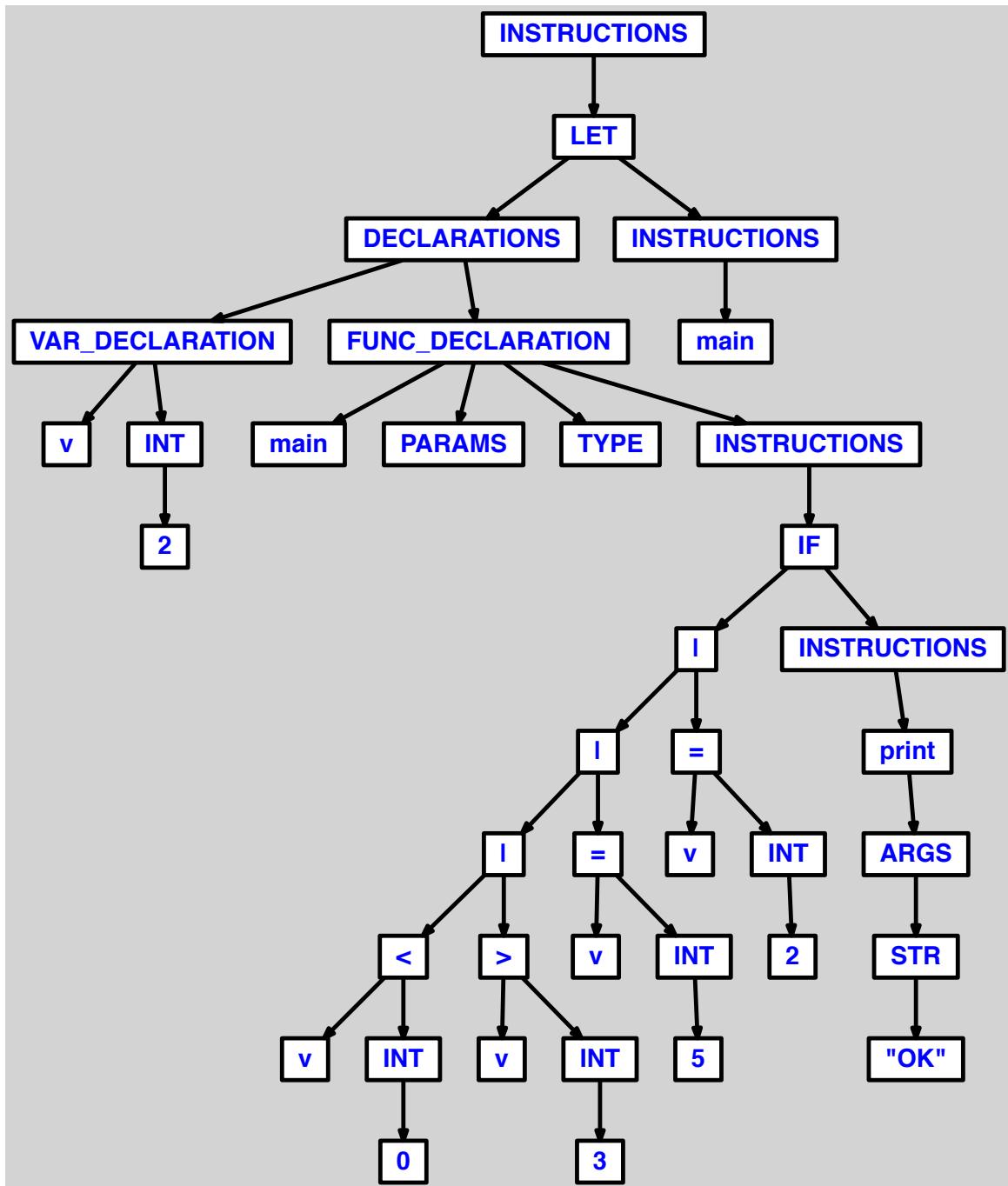


### 10.2.3 condition avec 3 |

```

let
  var v := 2

  function main() =
    if v < 0 | v > 3 | v = 5 | v = 2 then print("OK")
in main() end
  
```



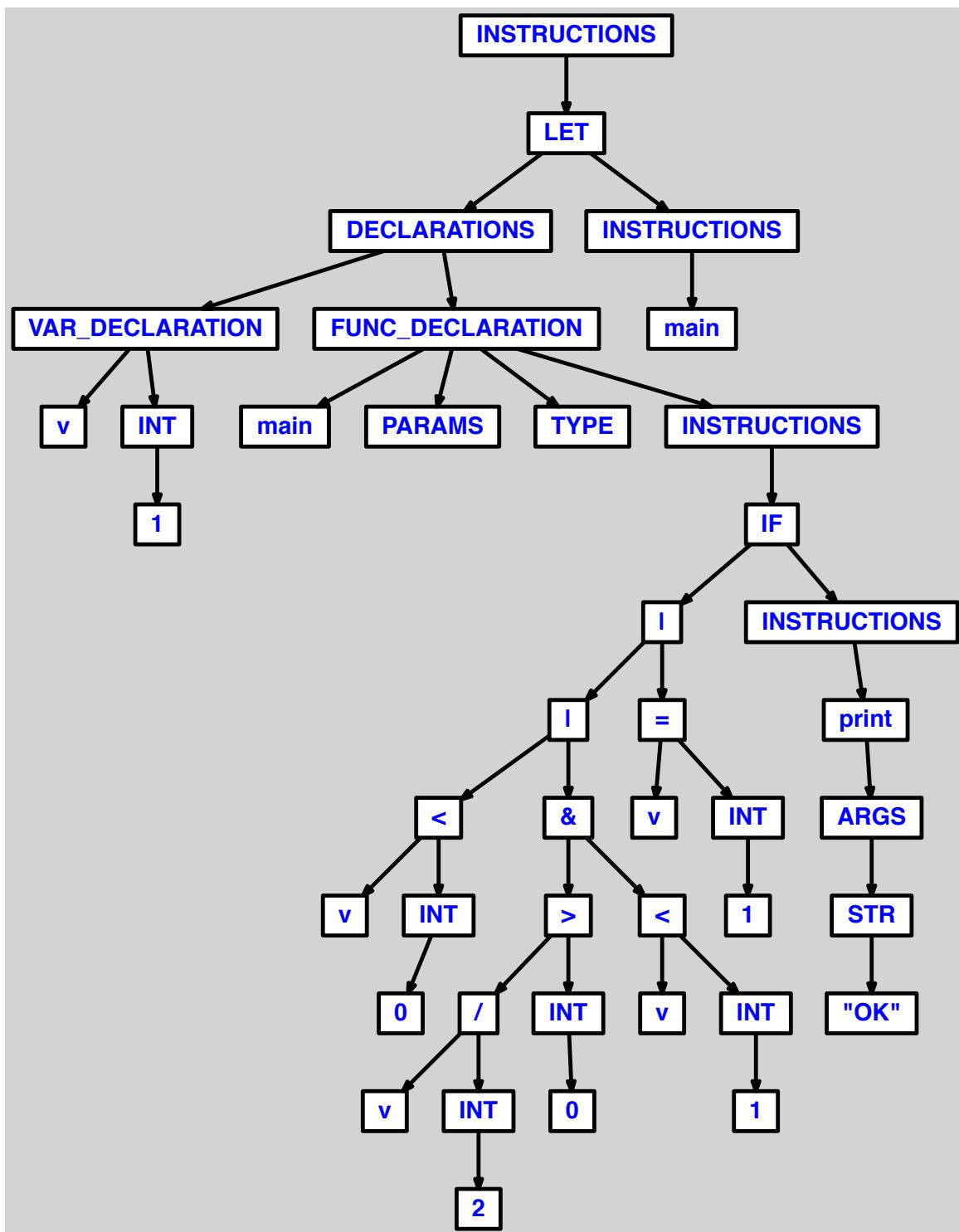
#### 10.2.4 condition avec 2 | et 1 &

let

```
var v := 1
```

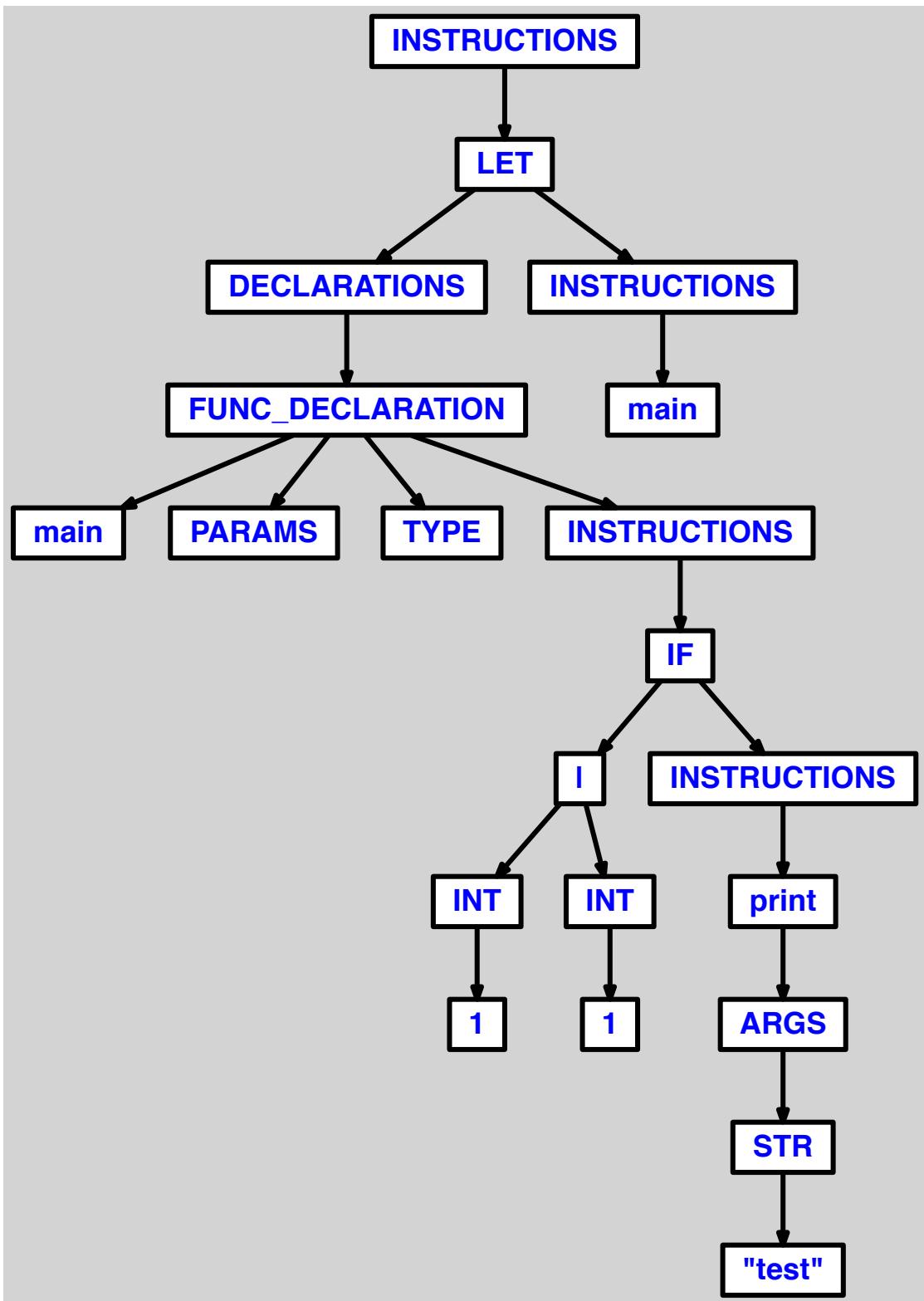
```
function main() =
    if v < 0 | v/2 > 0 & v < 1 | v = 1 then print("OK")
```

in main() end



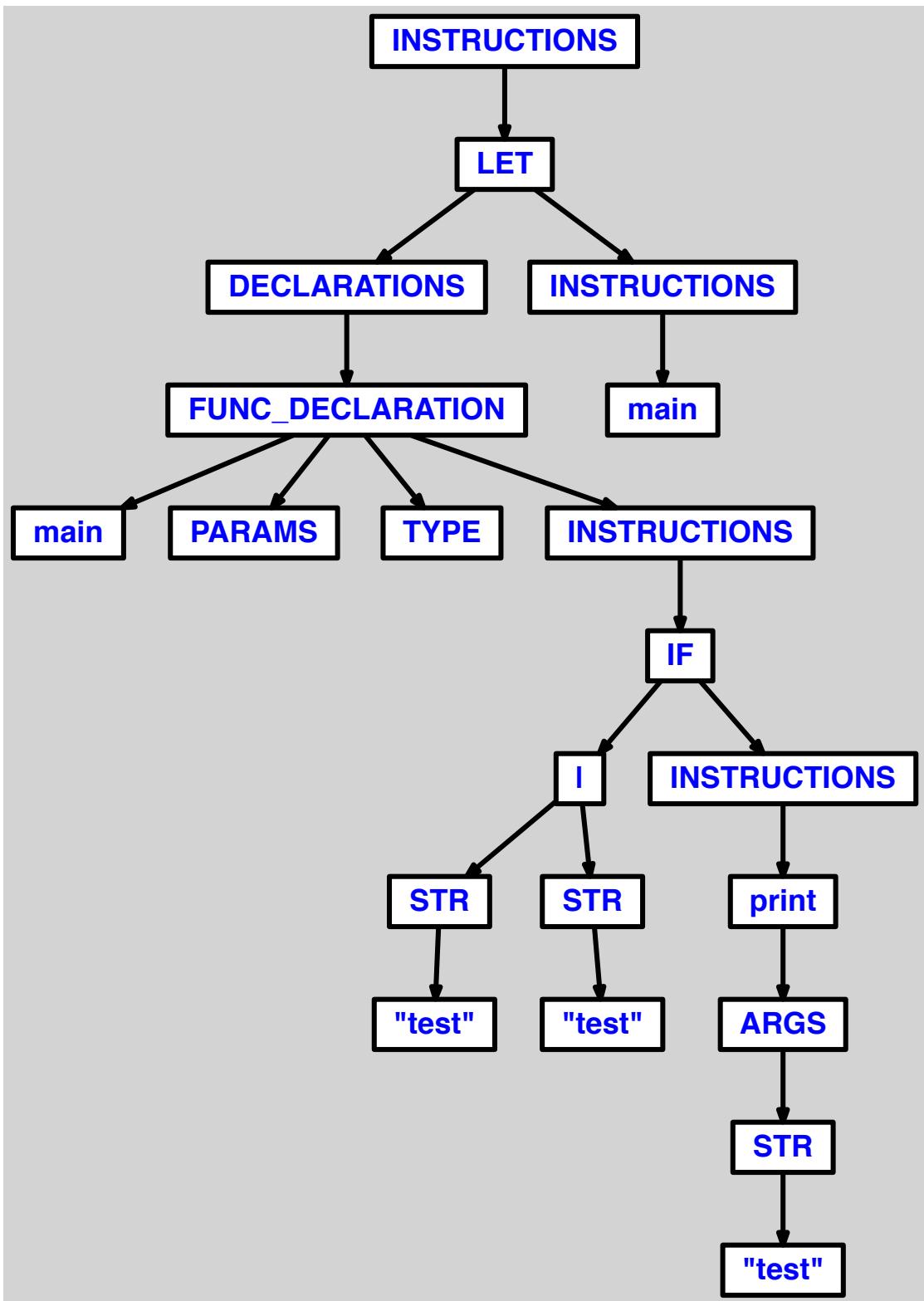
### 10.2.5 | avec des entiers

```
let
    function main() =
        if 1 | 1 then print("test")
in main() end
```



### 10.2.6 | avec des chaines

```
let
    function main() =
        if "test" | "test" then print("test")
in main() end
```

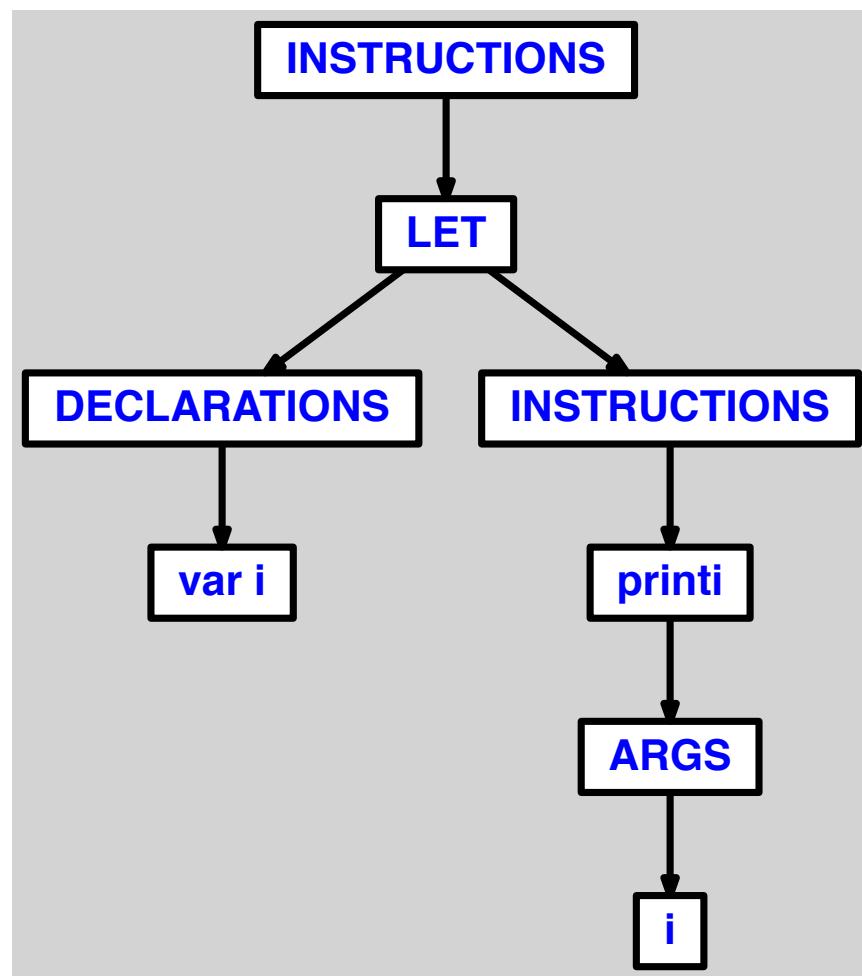


11 var

### 11.1 KO

#### 11.1.1 déclaration sans affectation

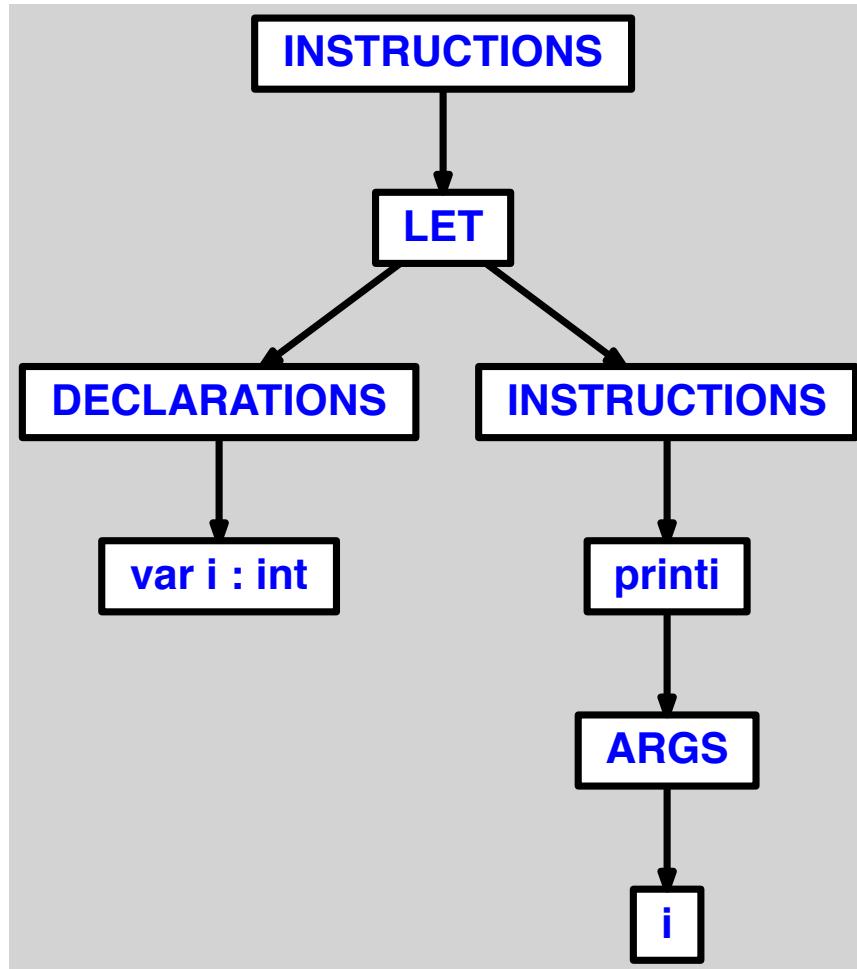
```
let
  var i
in
  printi(i)
end
```



#### 11.1.2 déclaration d'entier avec type sans affectation

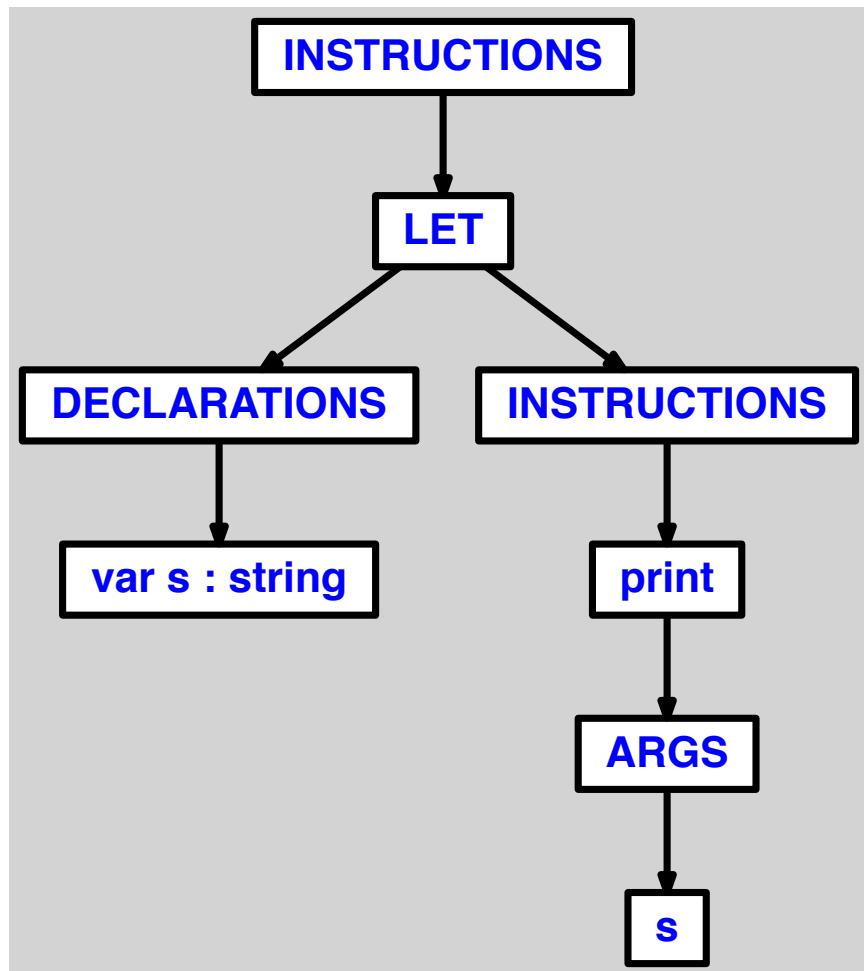
```
let
  var i : int
in
  printi(i)
```

end



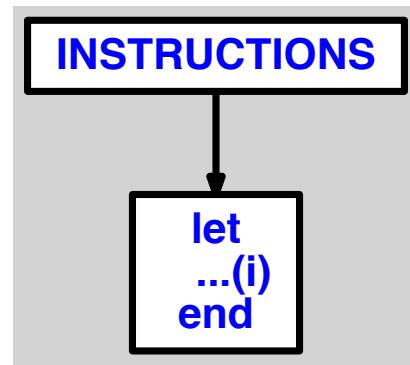
#### 11.1.3 déclaration de chaîne avec type sans affectation

```
let
    var s : string
in
    print(s)
end
```



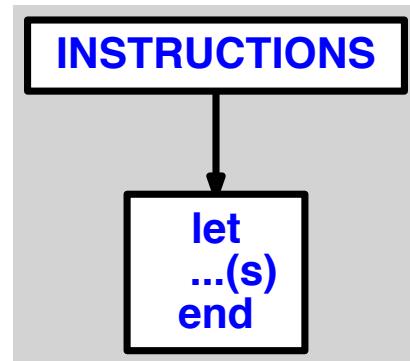
#### 11.1.4 affectation d'entier sans declaration

```
let
    i := 1
in
    print(i)
end
```



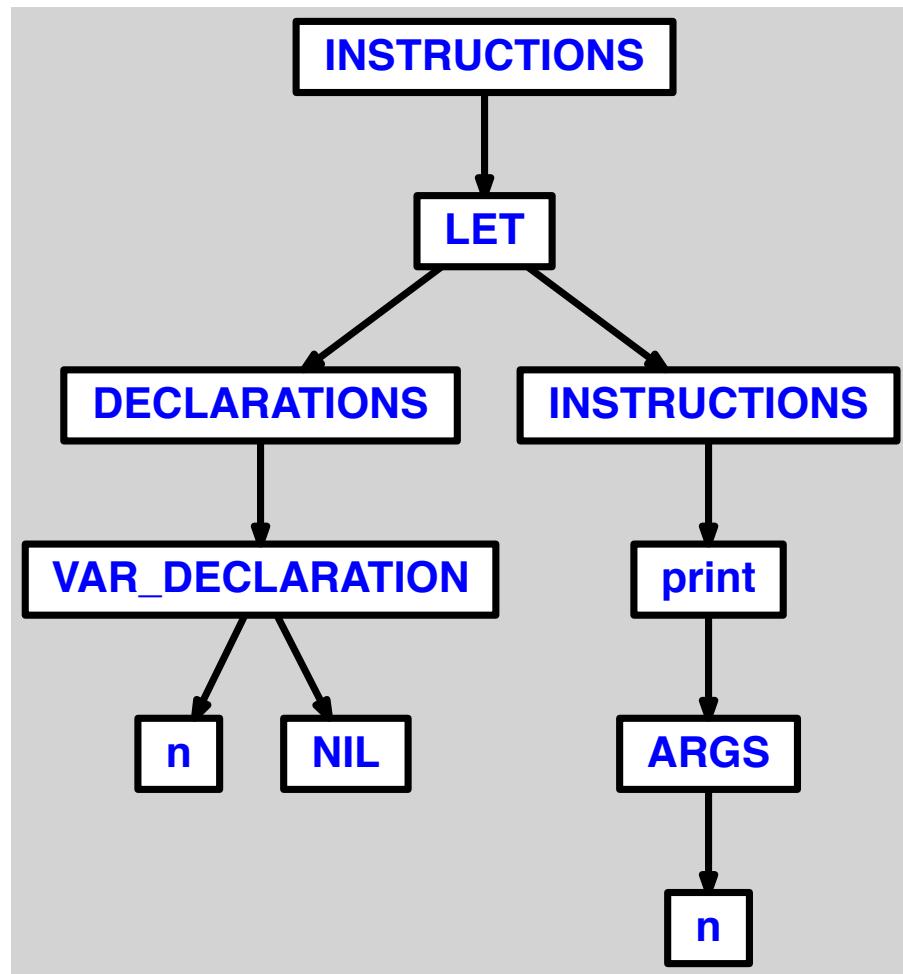
#### 11.1.5 affectation de chaine sans declaration

```
let
    s := "test"
in
    print(s)
end
```



#### 11.1.6 affectation de nil

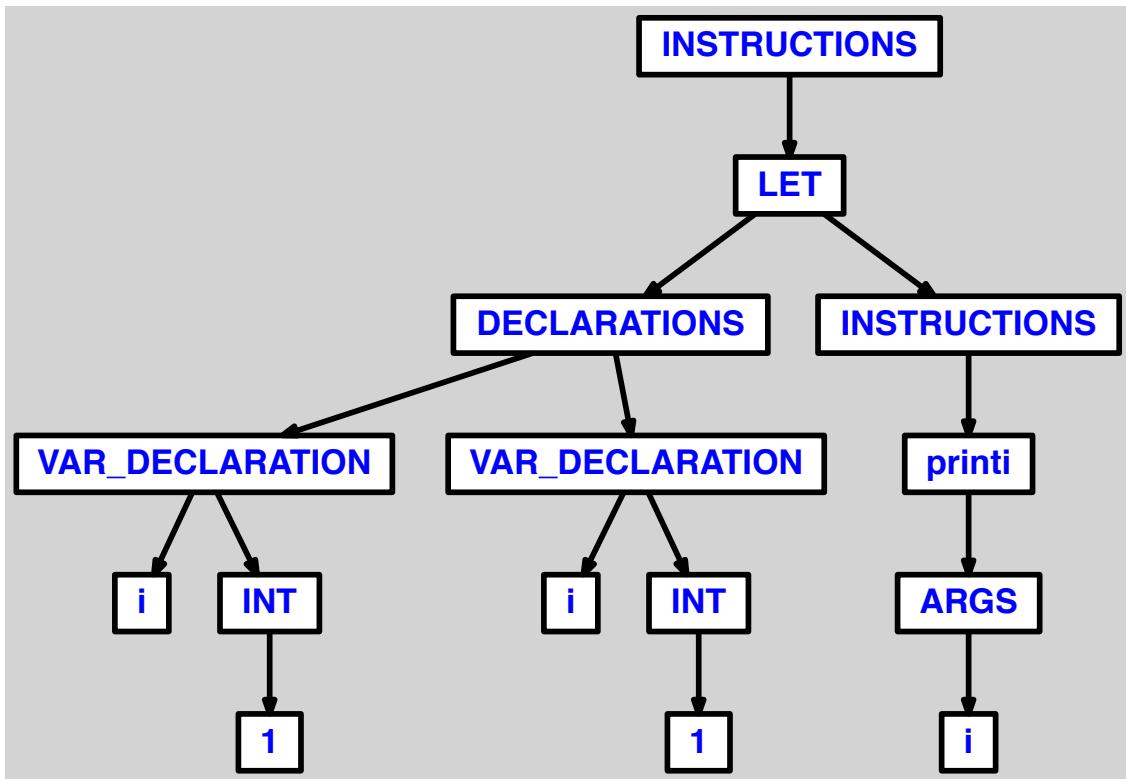
```
let
    var n := nil
in
    print(n)
end
```



#### 11.1.7 double-declaration avec valeurs égales

```

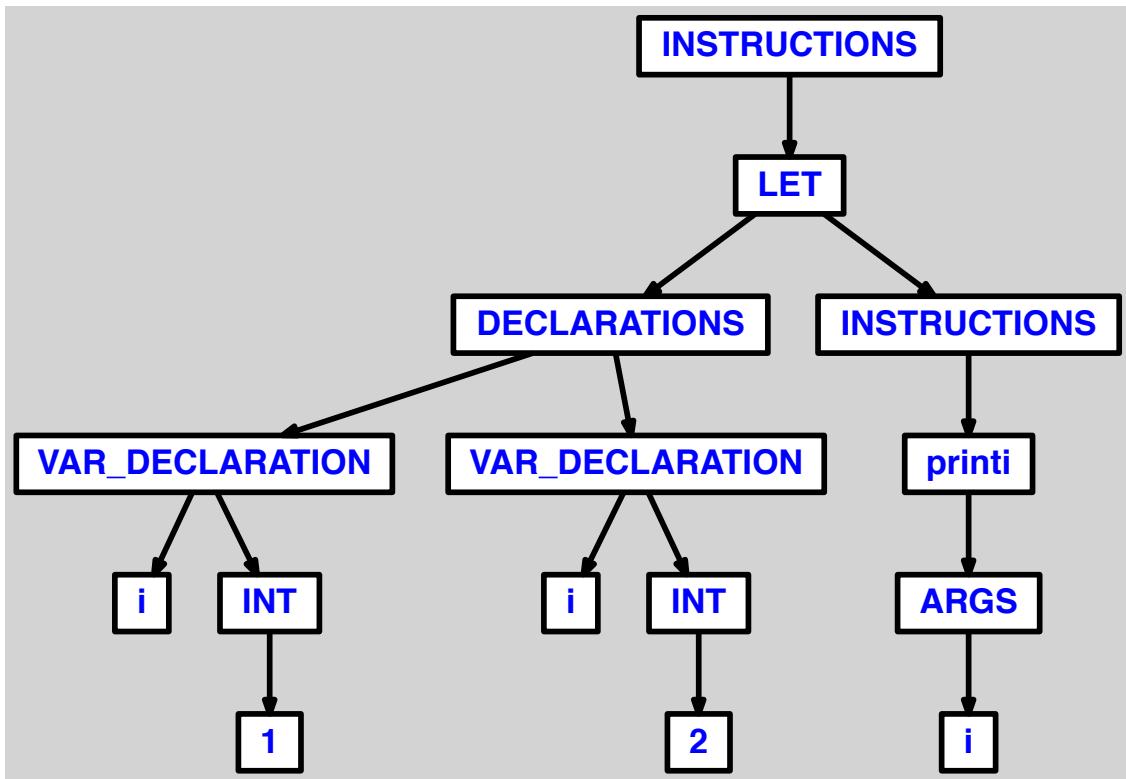
let
    var i := 1
    var i := 1
in
    print(i)
end
  
```



#### 11.1.8 double-declaration avec valeurs différentes

```

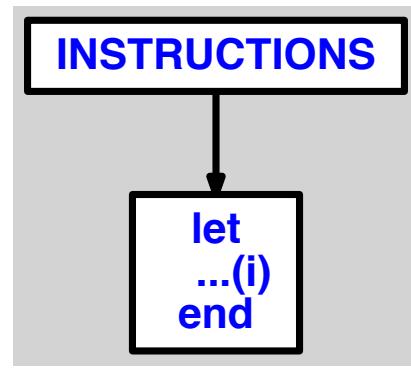
let
  var i := 1
  var i := 2
in
  printi(i)
end
  
```



### 11.1.9 var mal écrit

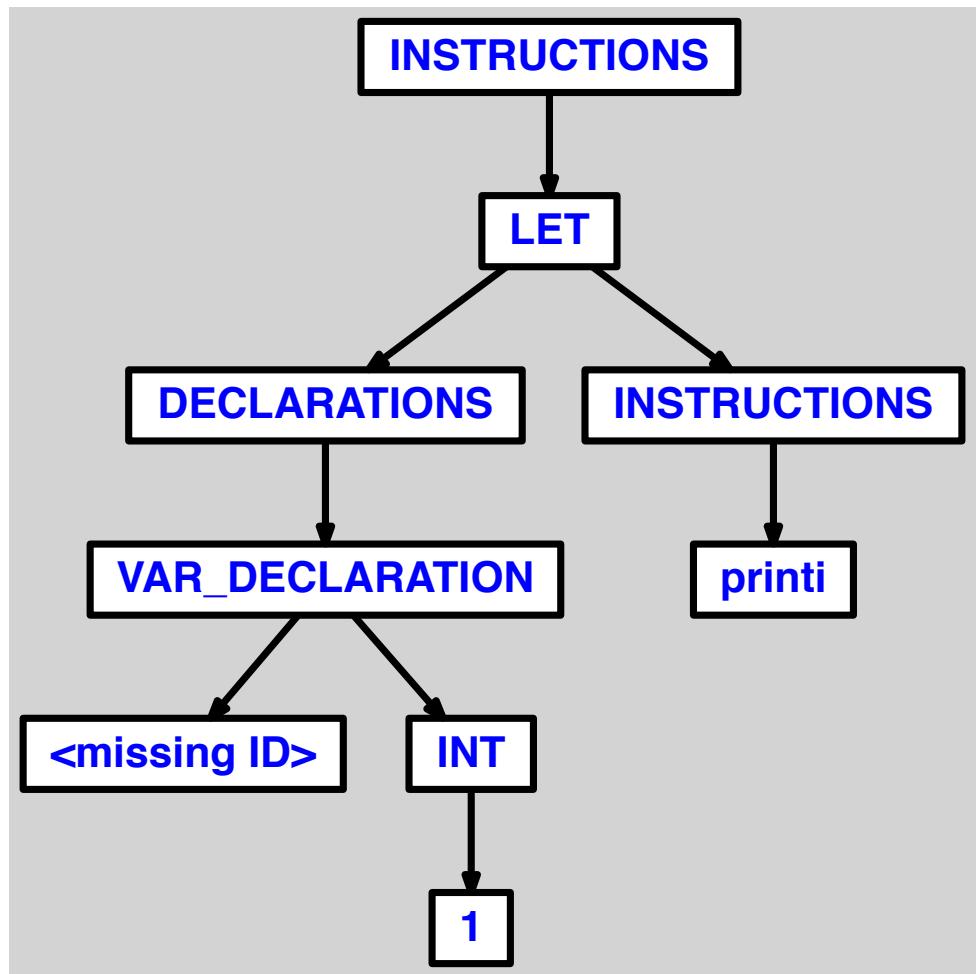
```

let
  varr i := 1
in
  printi(i)
end
  
```



#### 11.1.10 declaration de caractere special

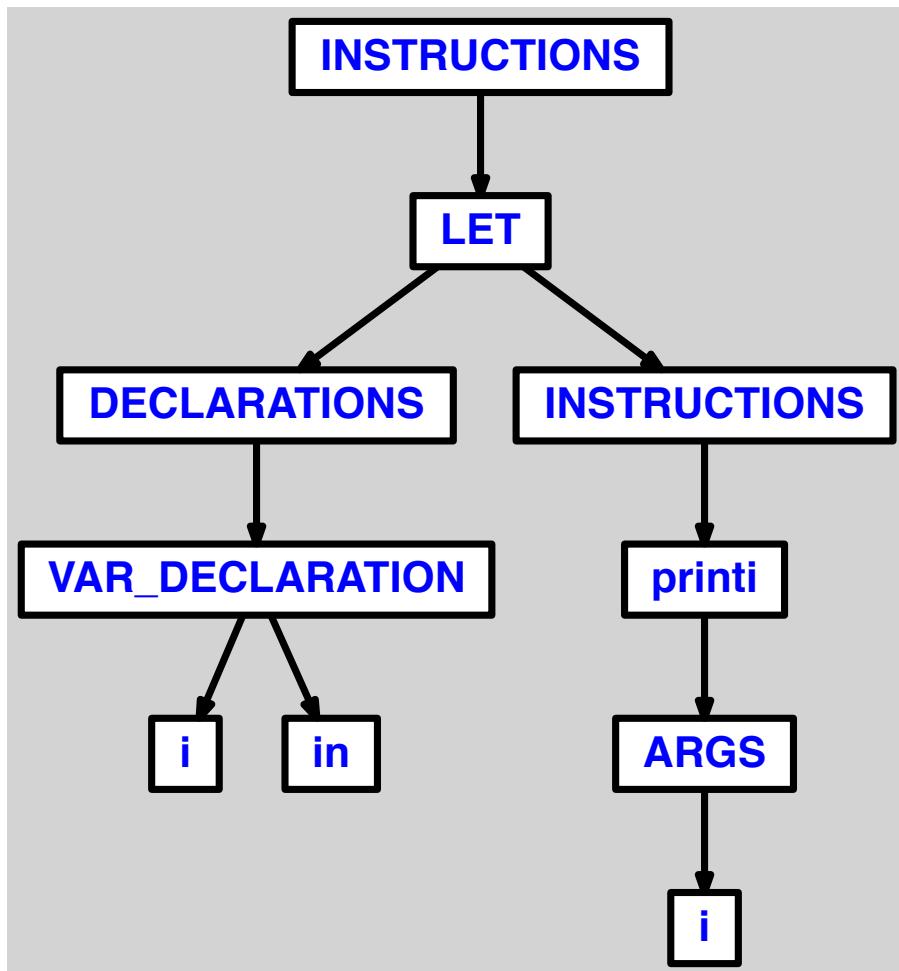
```
let
    var \ := 1
in
    printi(\)
end
```



#### 11.1.11 affectation de caractere special

```

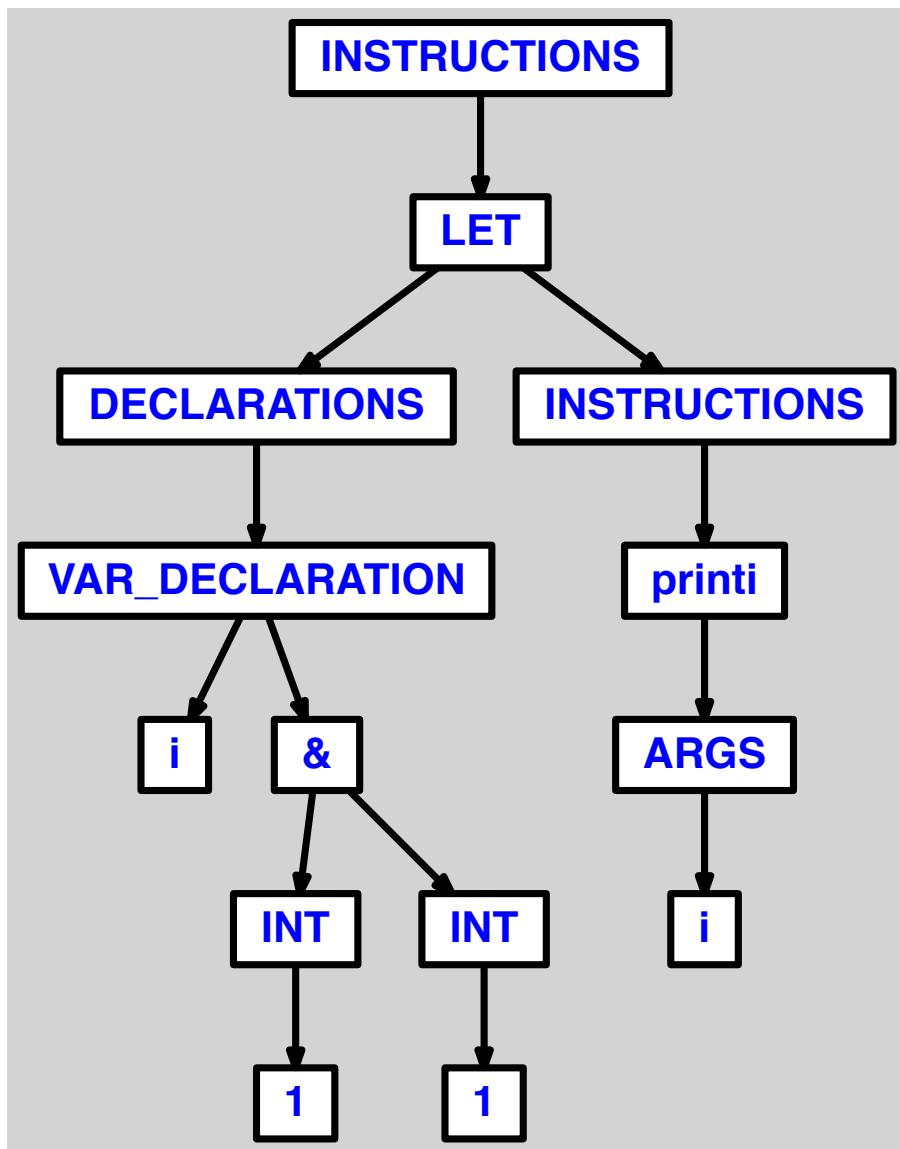
let
    var i := \
in
    printi(i)
end
  
```



#### 11.1.12 affectation d'opération logique sur entiers

```

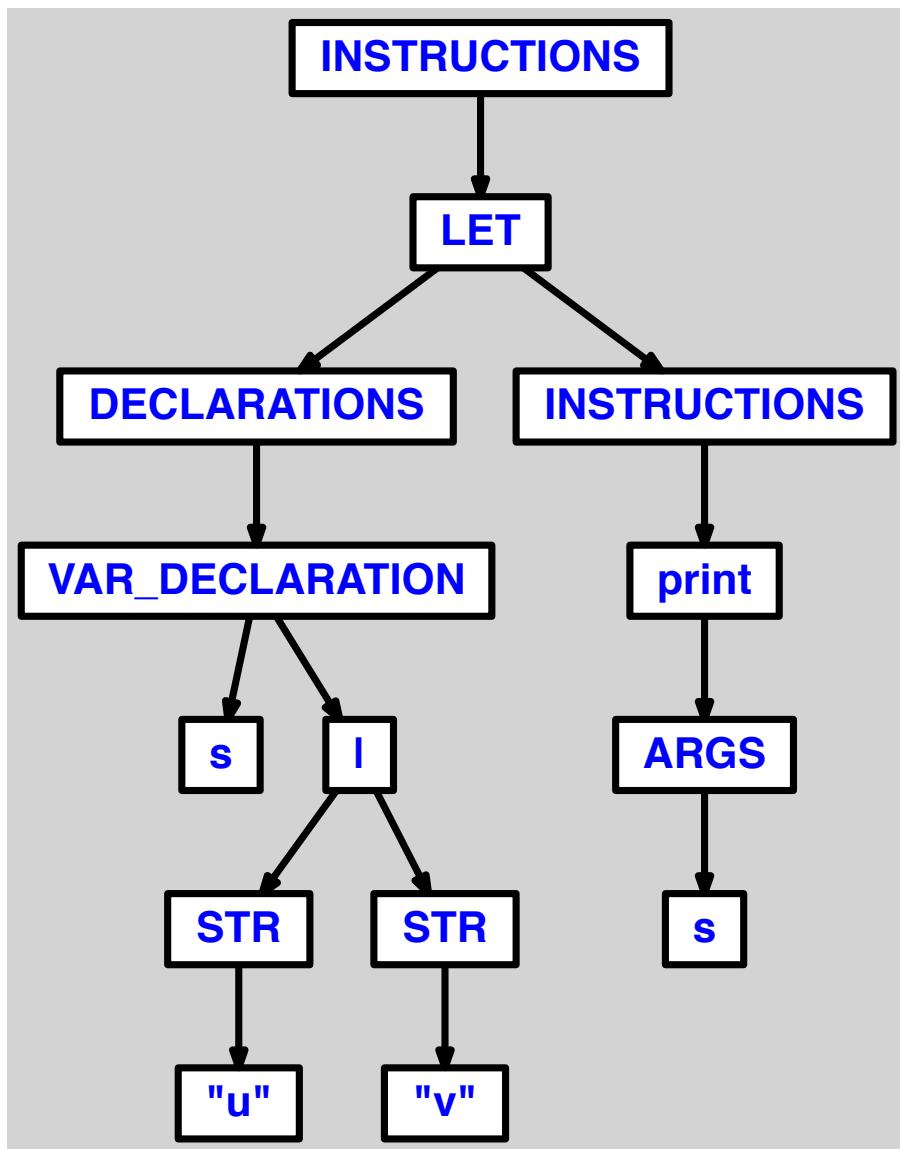
let
    var i := 1 & 1
in
    printi(i)
end
  
```



#### 11.1.13 affectation d'opération logique sur chaînes

```

let
  var s := "u" | "v"
in
  print(s)
end
  
```

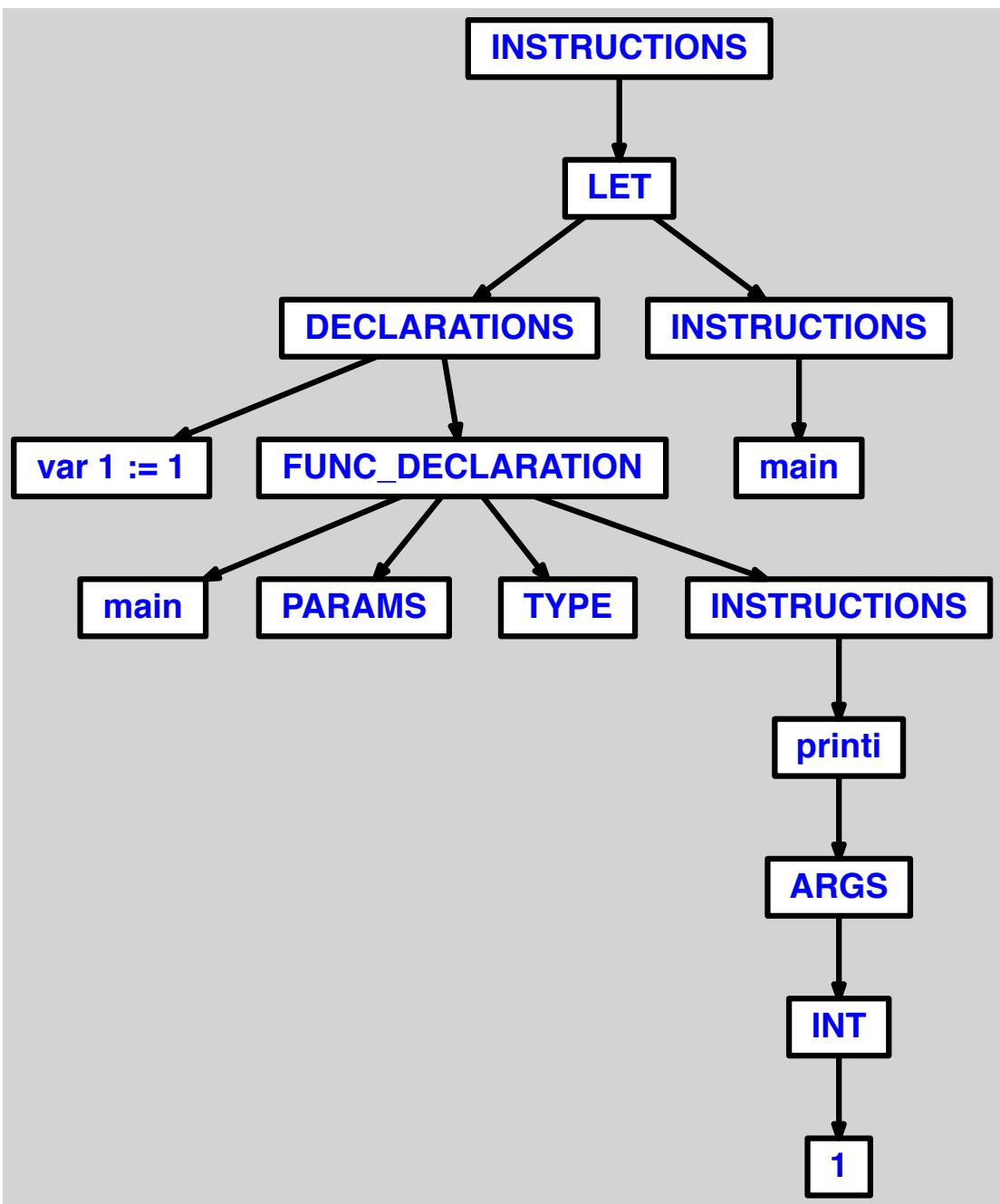


#### 11.1.14 chiffre comme nom de variable

```

let
  var 1 := 1

  function main() = printi(1)
in main() end
  
```



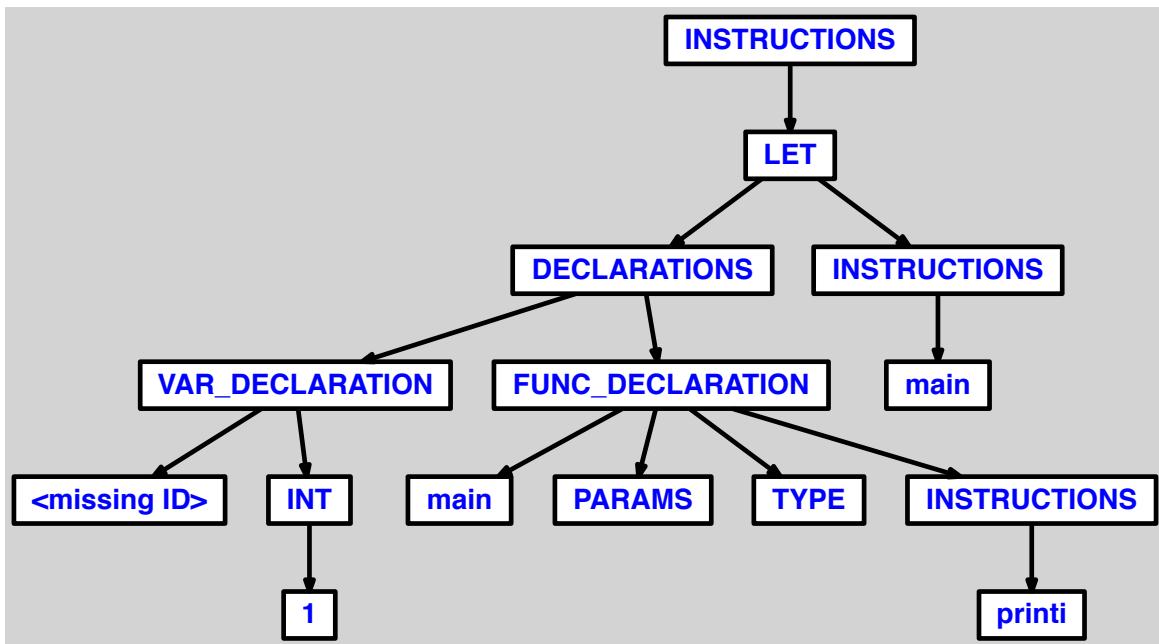
#### 11.1.15 \_ comme nom de variable

```

let
  var _ := 1

  function main() = printi(_)
in main() end

```

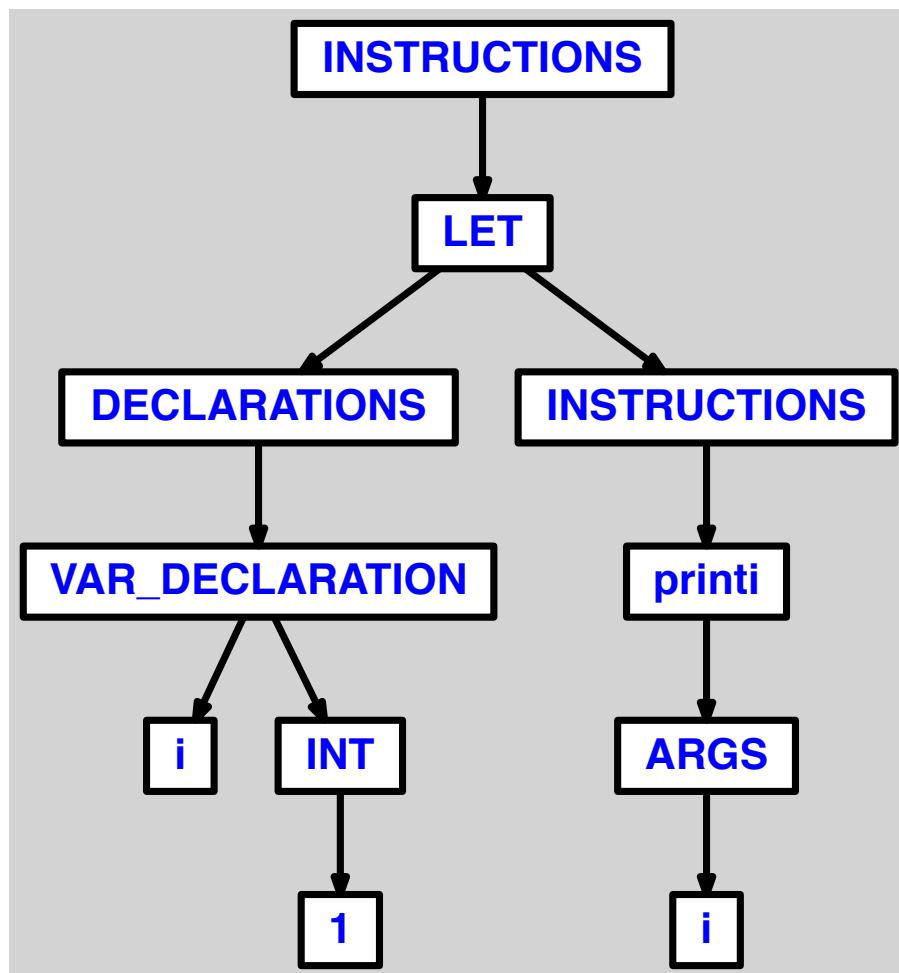


## 11.2 OK

### 11.2.1 declaration d'entier simple

```

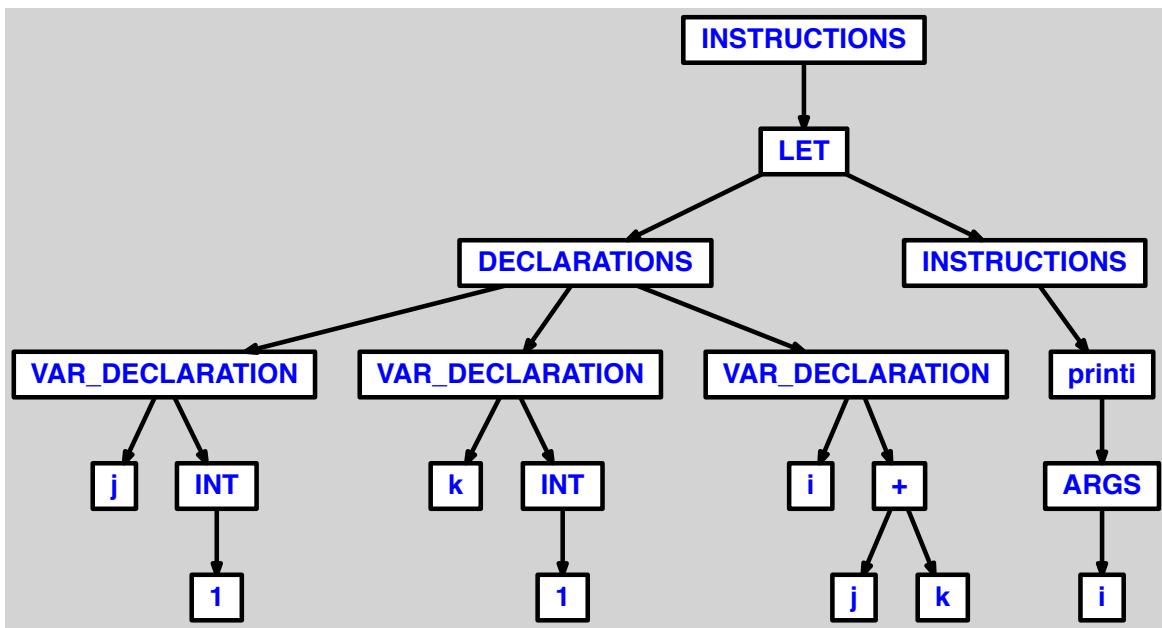
let
  var i := 1
in
  printi(i)
end
  
```



#### 11.2.2 déclarations d'entier avec reutilisation de 2 autres entiers

```

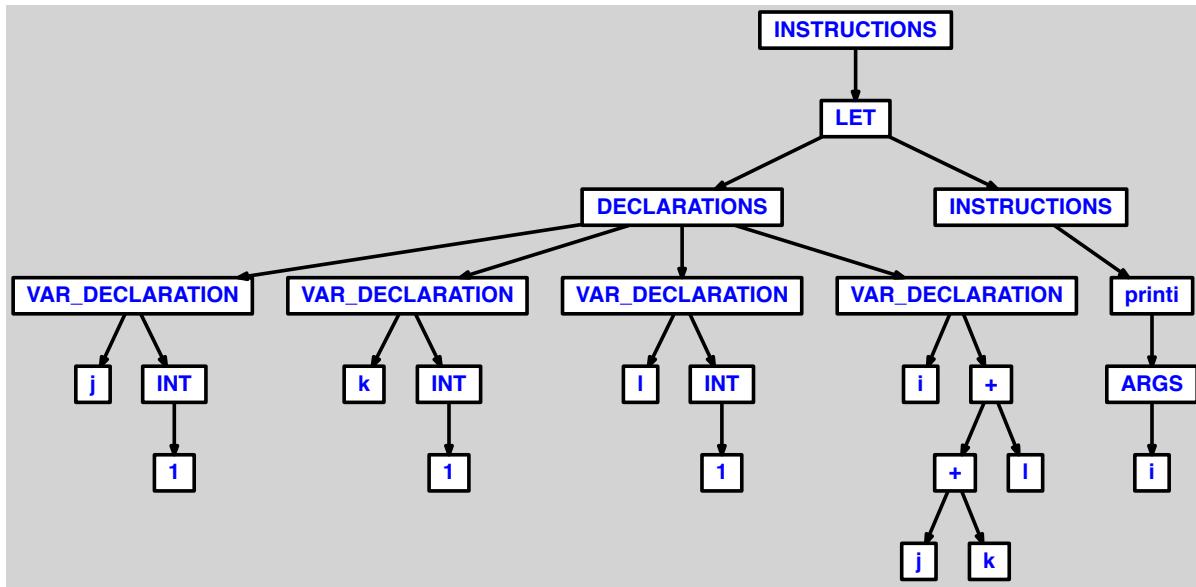
let
    var j := 1
    var k := 1
    var i := j+k
in
    printi(i)
end
  
```



### 11.2.3 déclarations d'entier avec réutilisation de 3 autres entiers

```

let
  var j := 1
  var k := 1
  var l := 1
  var i := j+k+l
in
  printi(i)
end
  
```

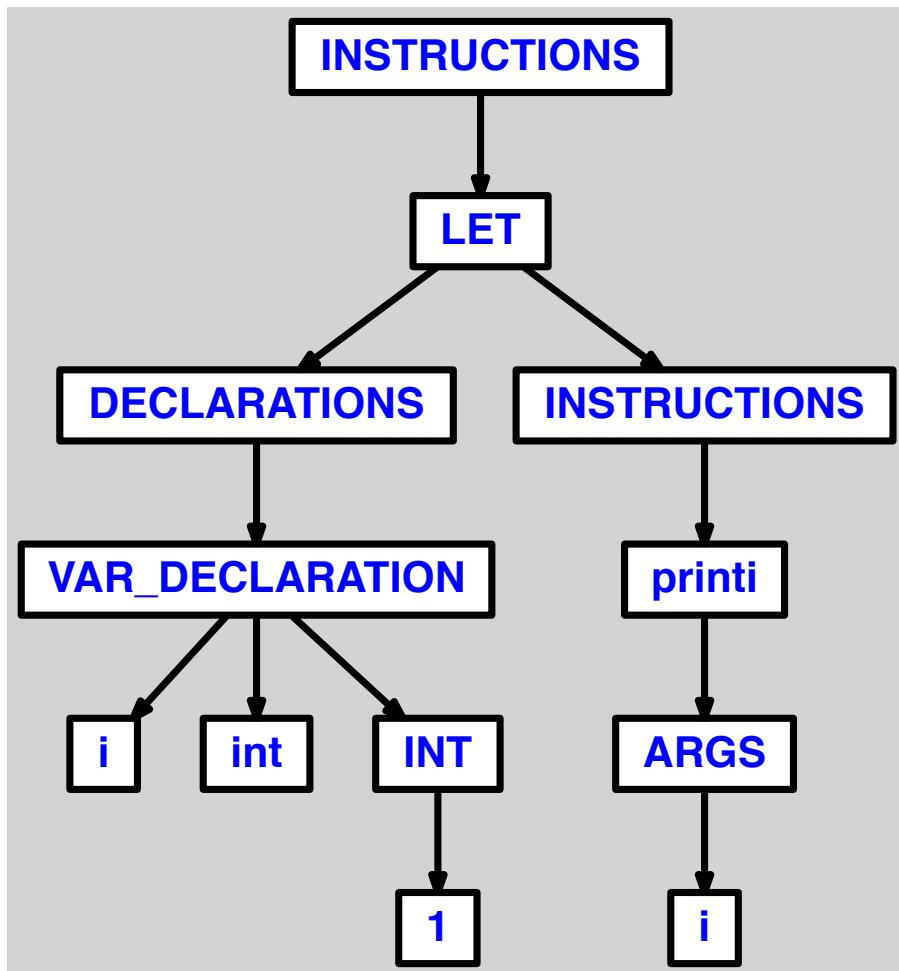


#### 11.2.4 declaration simple d'entier avec type

```

let
    var i : int := 1
in
    printi(i)
end

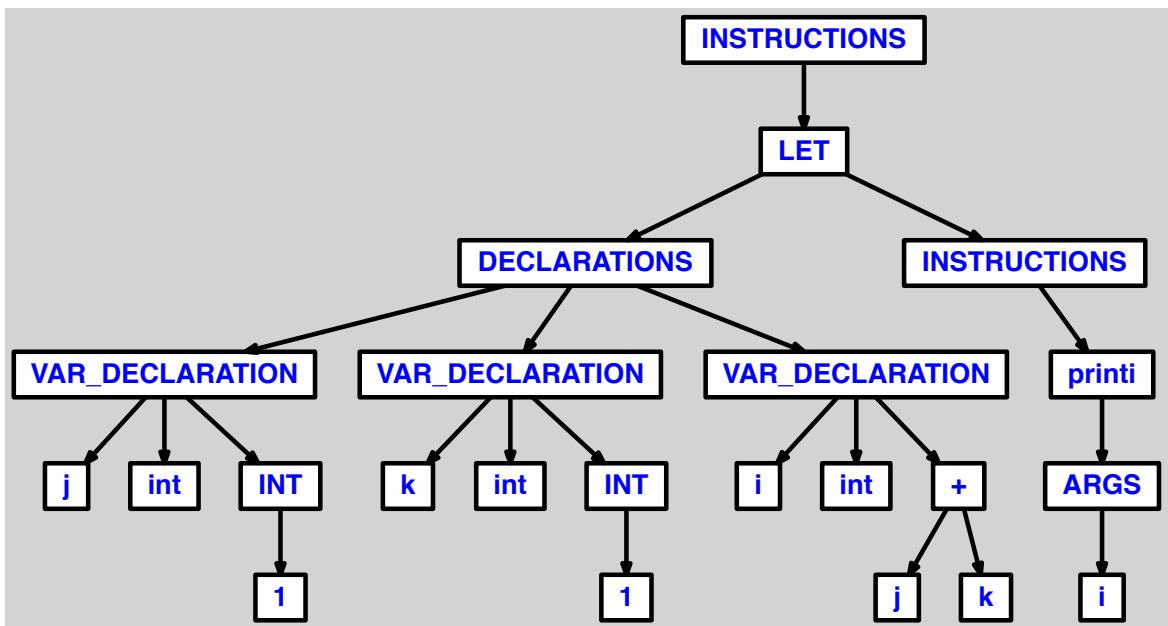
```



#### 11.2.5 déclarations d'entier avec types et reutilisation de 2 autres entiers

```

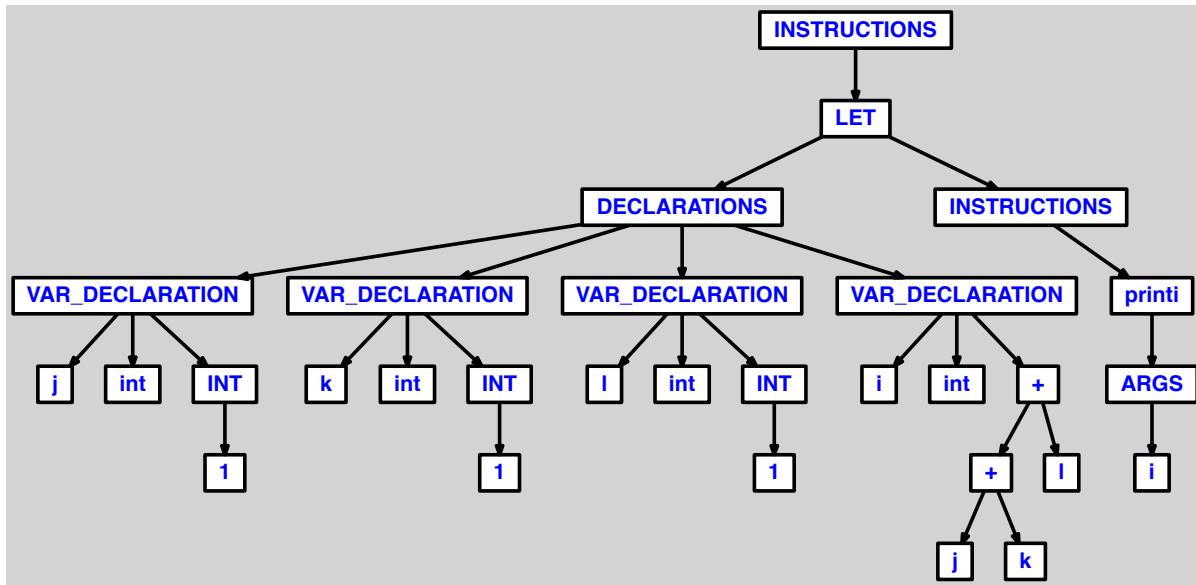
let
    var j : int := 1
    var k : int := 1
    var i : int := j+k
in
    printi(i)
end
  
```



#### 11.2.6 déclarations d'entier avec types et reutilisation de 3 autres entiers

```

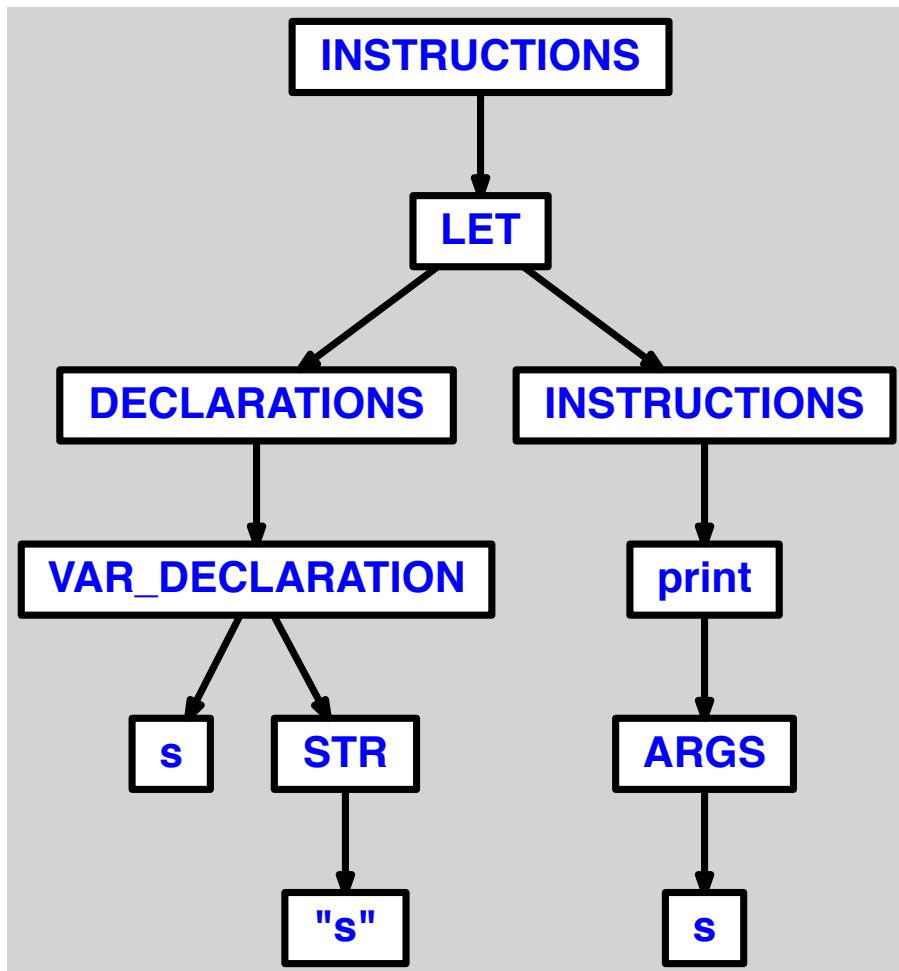
let
  var j : int := 1
  var k : int := 1
  var l : int := 1
  var i : int := j+k+l
in
  printi(i)
end
  
```



### 11.2.7 déclaration de chaîne simple

```

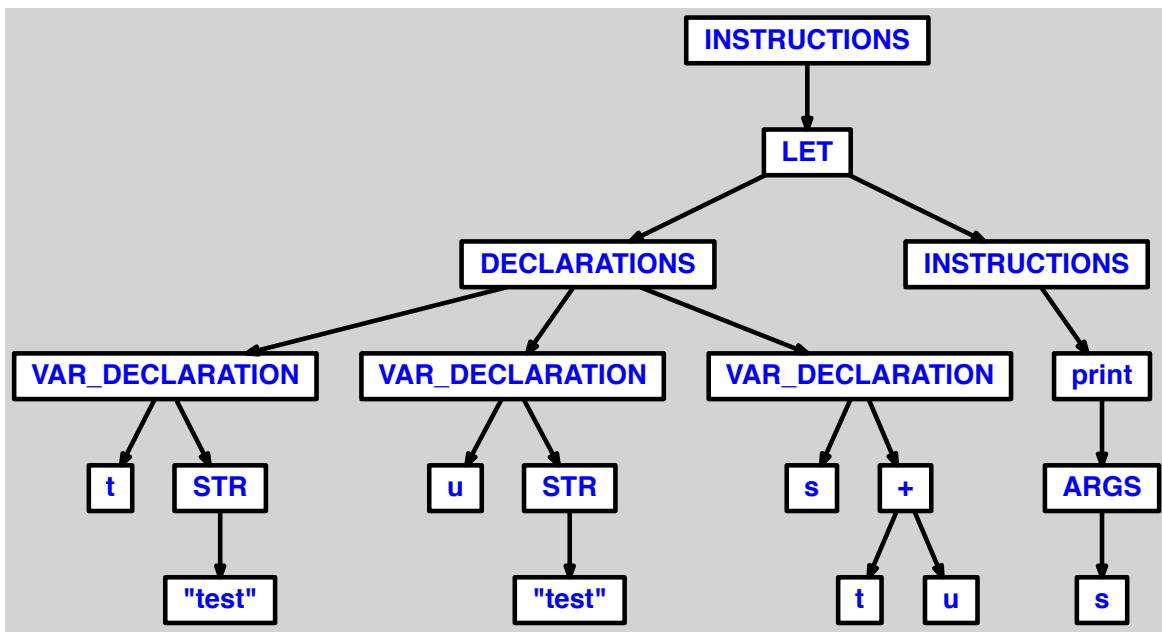
let
  var s := "s"
in
  print(s)
end
  
```



#### 11.2.8 déclarations de chaîne avec réutilisation de 2 autres chaînes

```

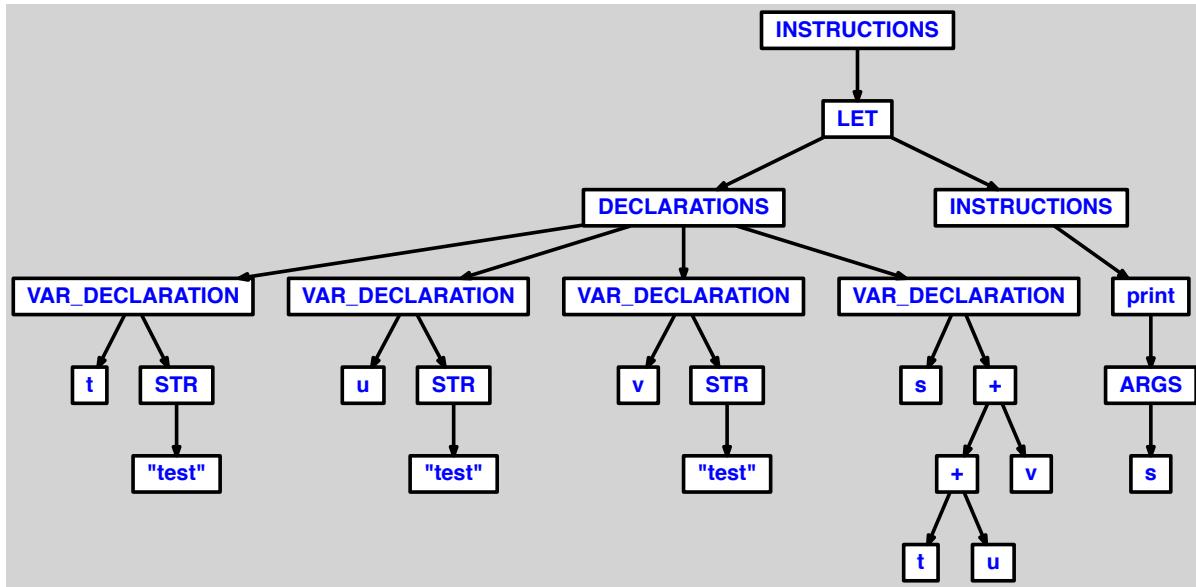
let
    var t := "test"
    var u := "test"
    var s := t+u
in
    print(s)
end
  
```



#### 11.2.9 déclarations de chaîne avec réutilisation de 3 autres chaînes

```

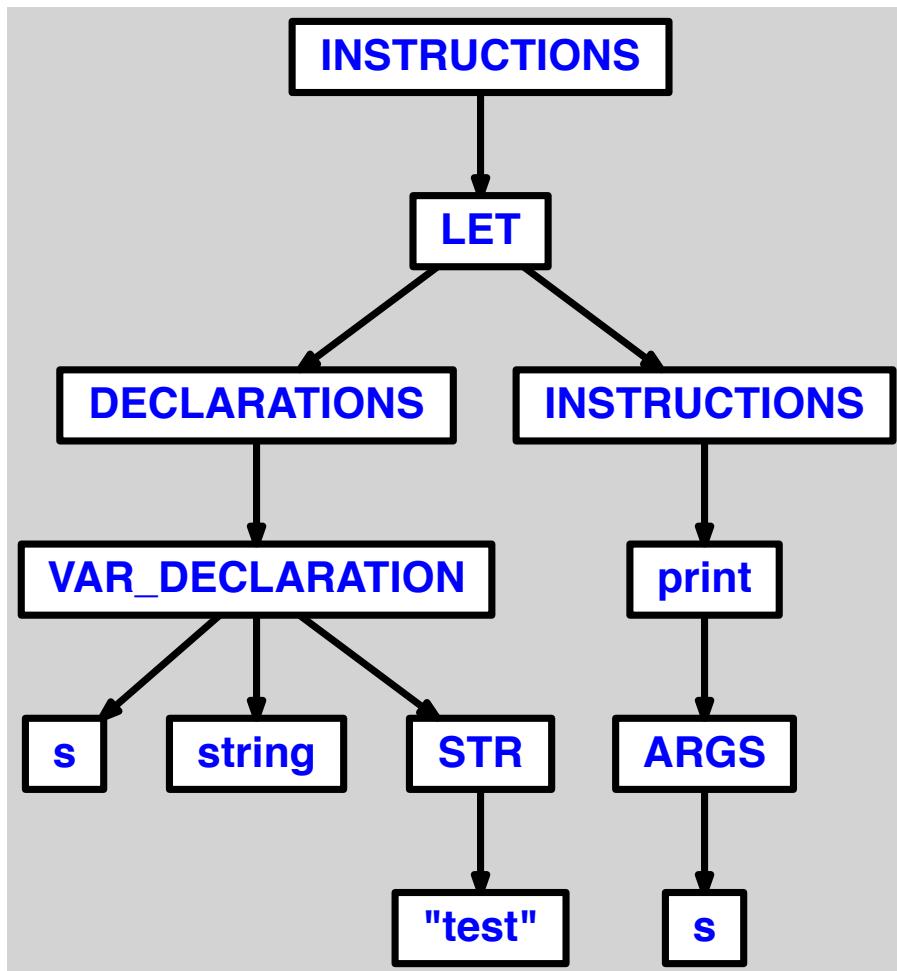
let
  var t := "test"
  var u := "test"
  var v := "test"
  var s := t+u+v
in
  print(s)
end
  
```



#### 11.2.10 déclaration de chaîne avec type

```

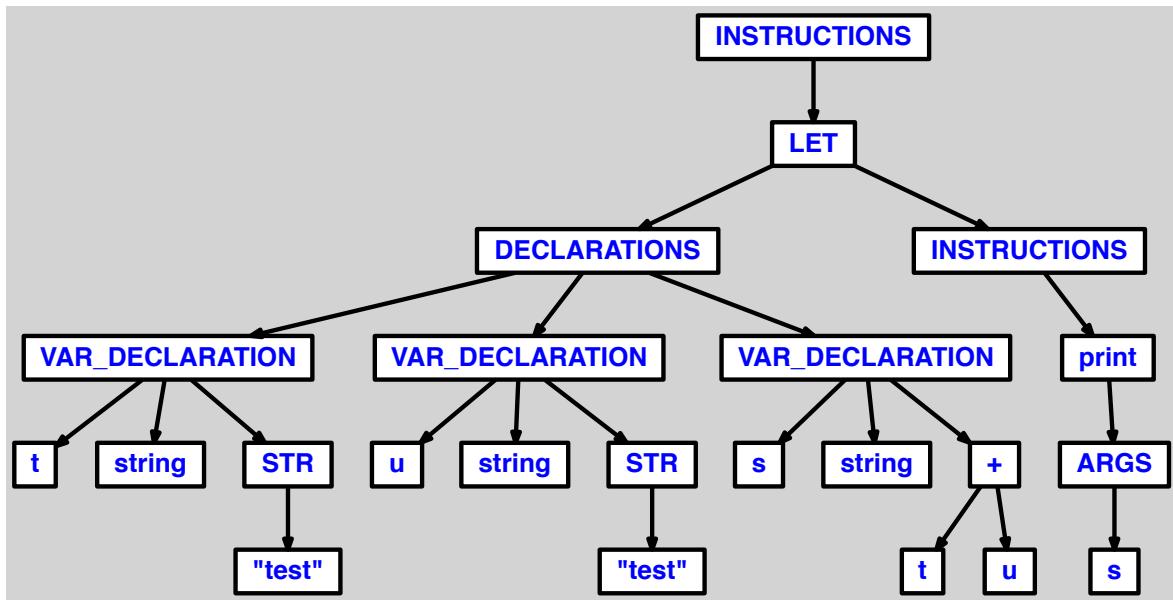
let
  var s : string := "test"
in
  print(s)
end
  
```



#### 11.2.11 declarations de chaine avec types et reutilisation de 2 autres chaines

```

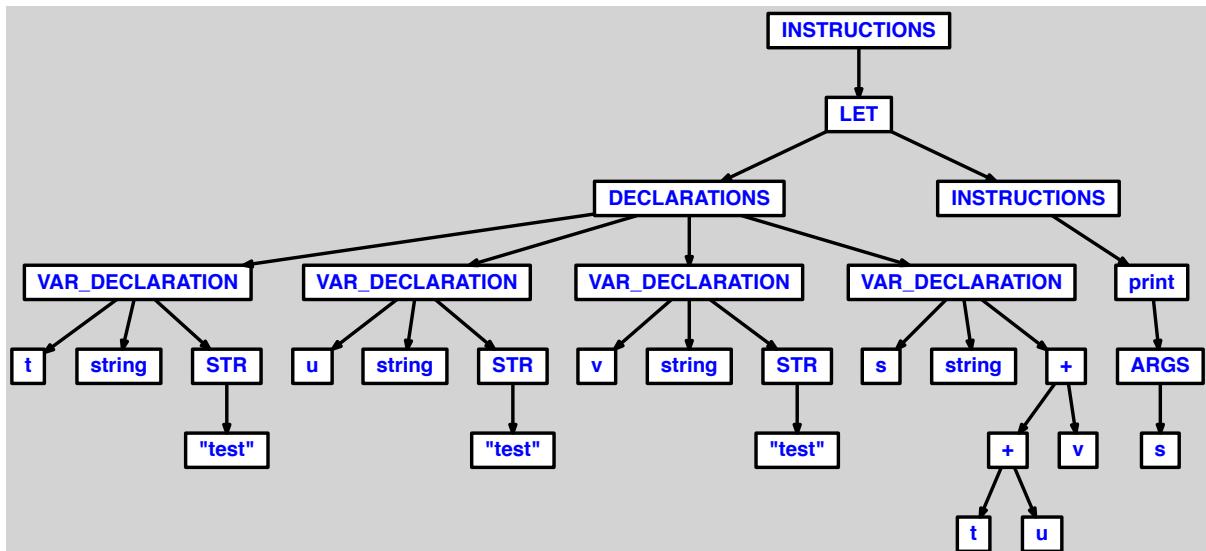
let
    var t : string := "test"
    var u : string := "test"
    var s : string := t+u
in
    print(s)
end
  
```



#### 11.2.12 declarations de chaine avec types et reutilisation de 3 autres chaines

```

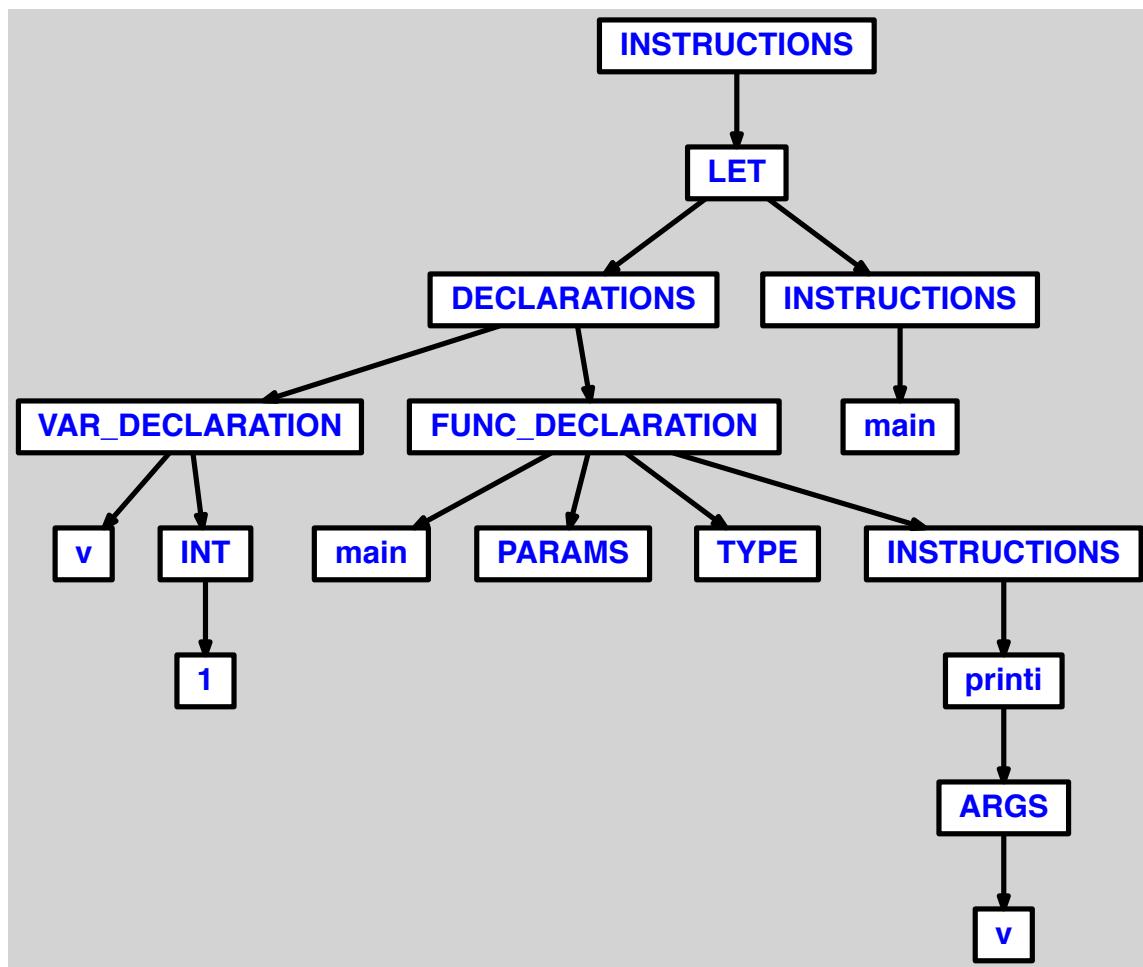
let
  var t : string := "test"
  var u : string := "test"
  var v : string := "test"
  var s : string := t+u+v
in
  print(s)
end
  
```



### 11.2.13 lettre minuscule comme nom de variable

let

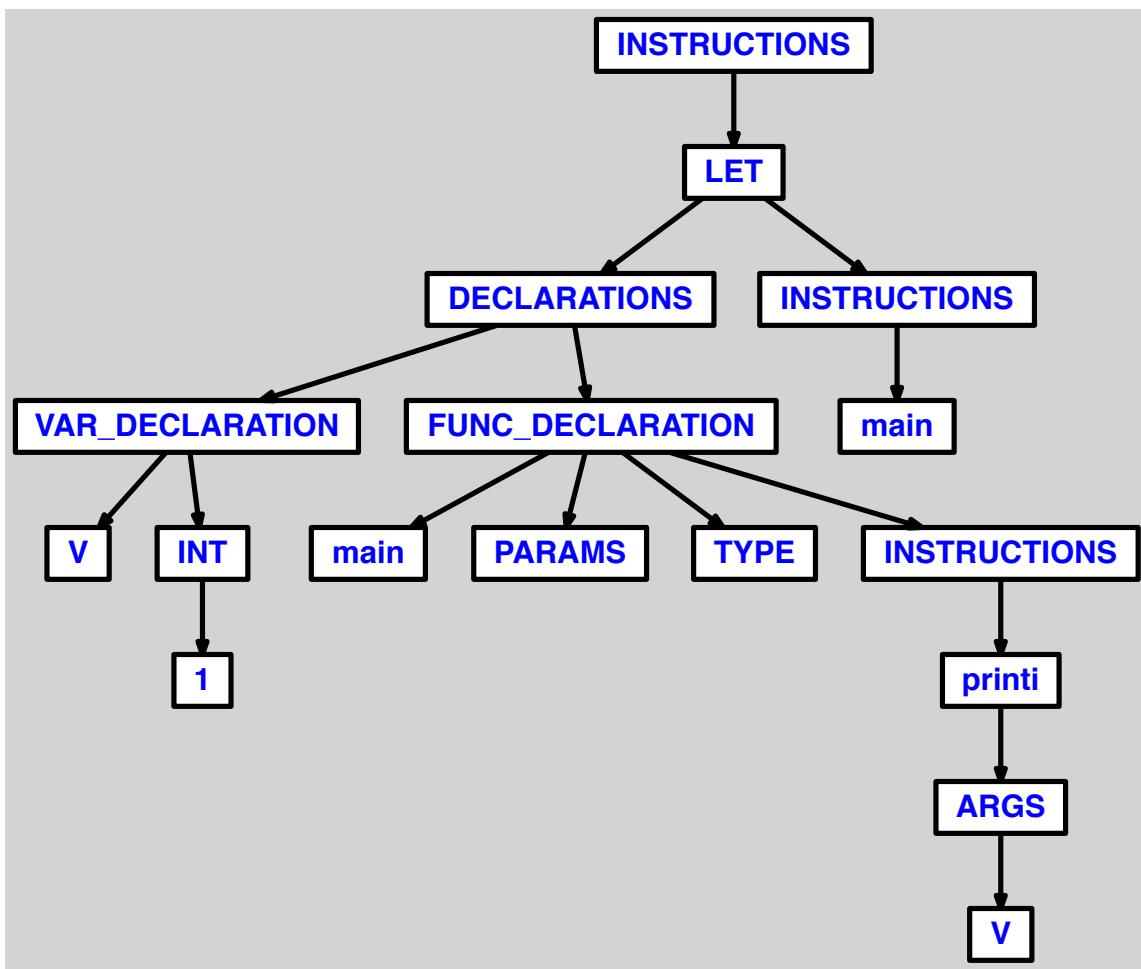
```
  var v := 1  
  
  function main() = printi(v)  
in main() end
```



### 11.2.14 lettre majuscule comme nom de variable

let

```
  var V := 1  
  
  function main() = printi(V)  
in main() end
```

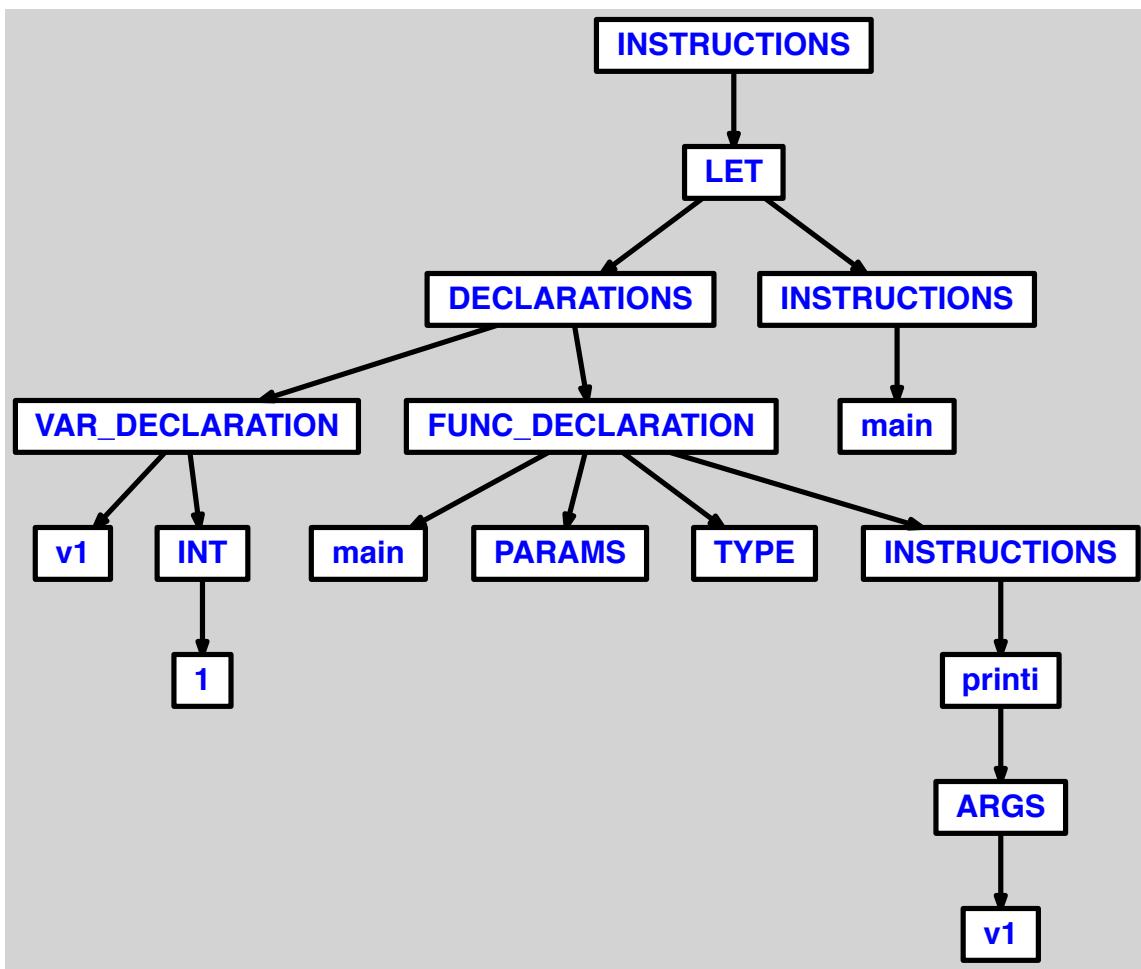


#### 11.2.15 lettre minuscule et chiffre dans nom de variable

```

let
  var v1 := 1

  function main() = printi(v1)
in main() end
  
```

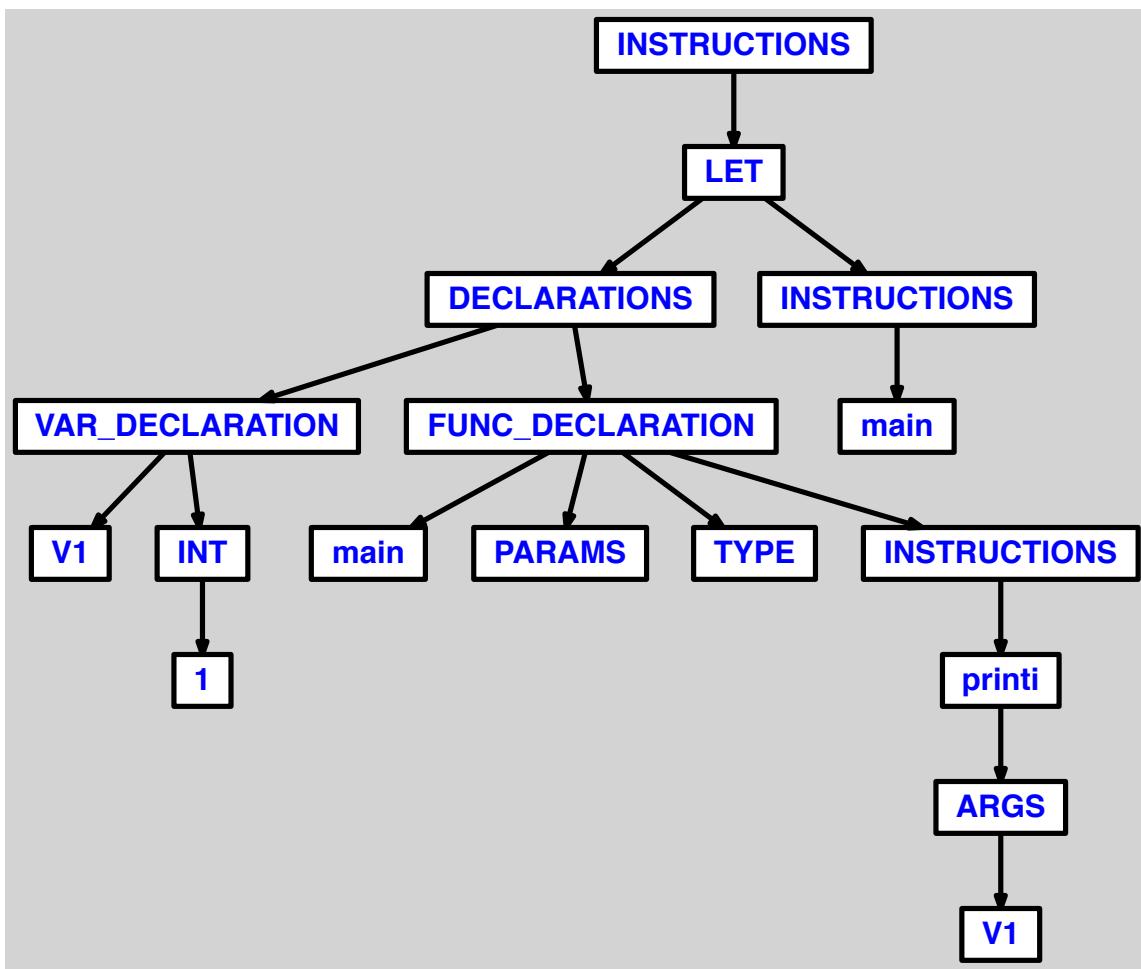


#### 11.2.16 lettre majuscule et chiffre dans nom de variable

```

let
  var V1 := 1

  function main() = printi(V1)
in main() end
  
```

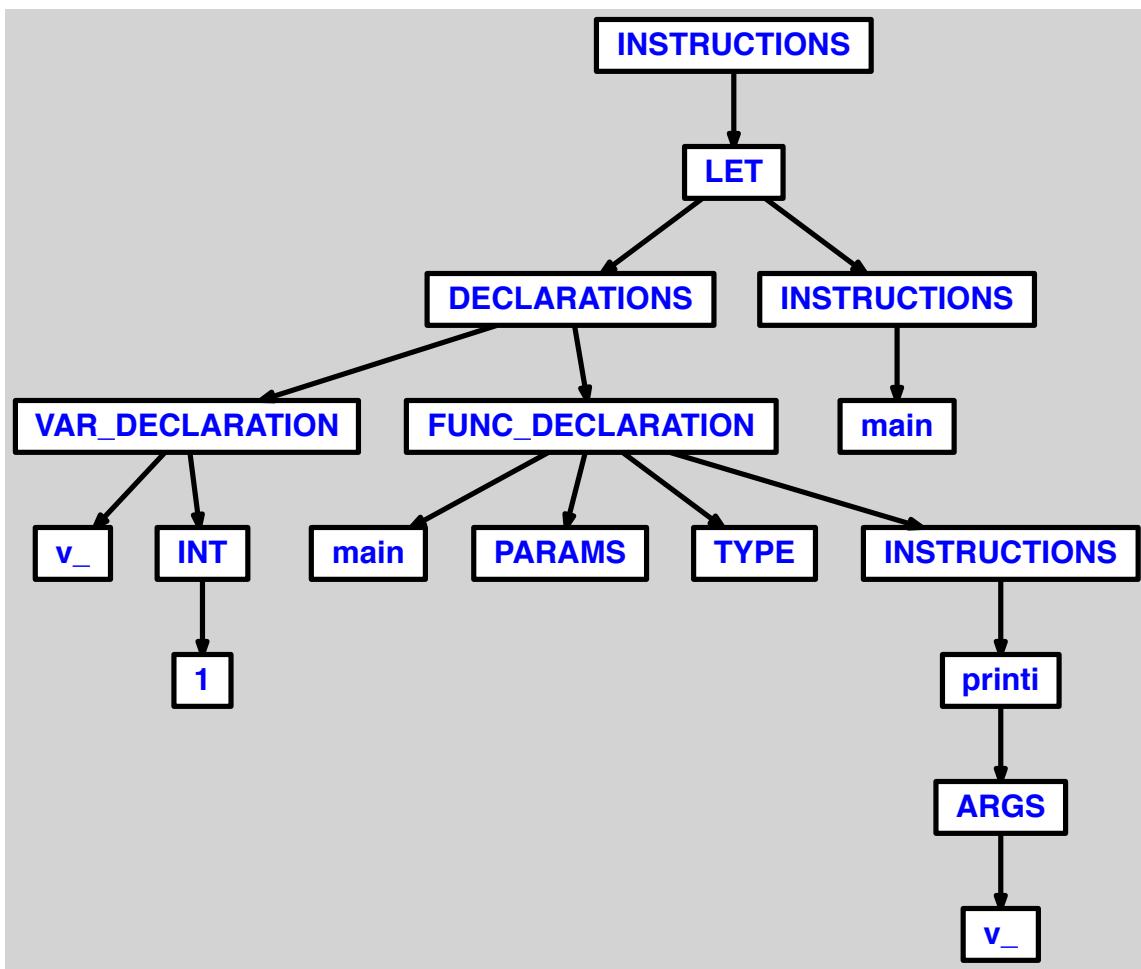


#### 11.2.17 lettre minuscule et \_ dans nom de variable

```

let
  var v_ := 1

  function main() = printi(v_)
in main() end
  
```

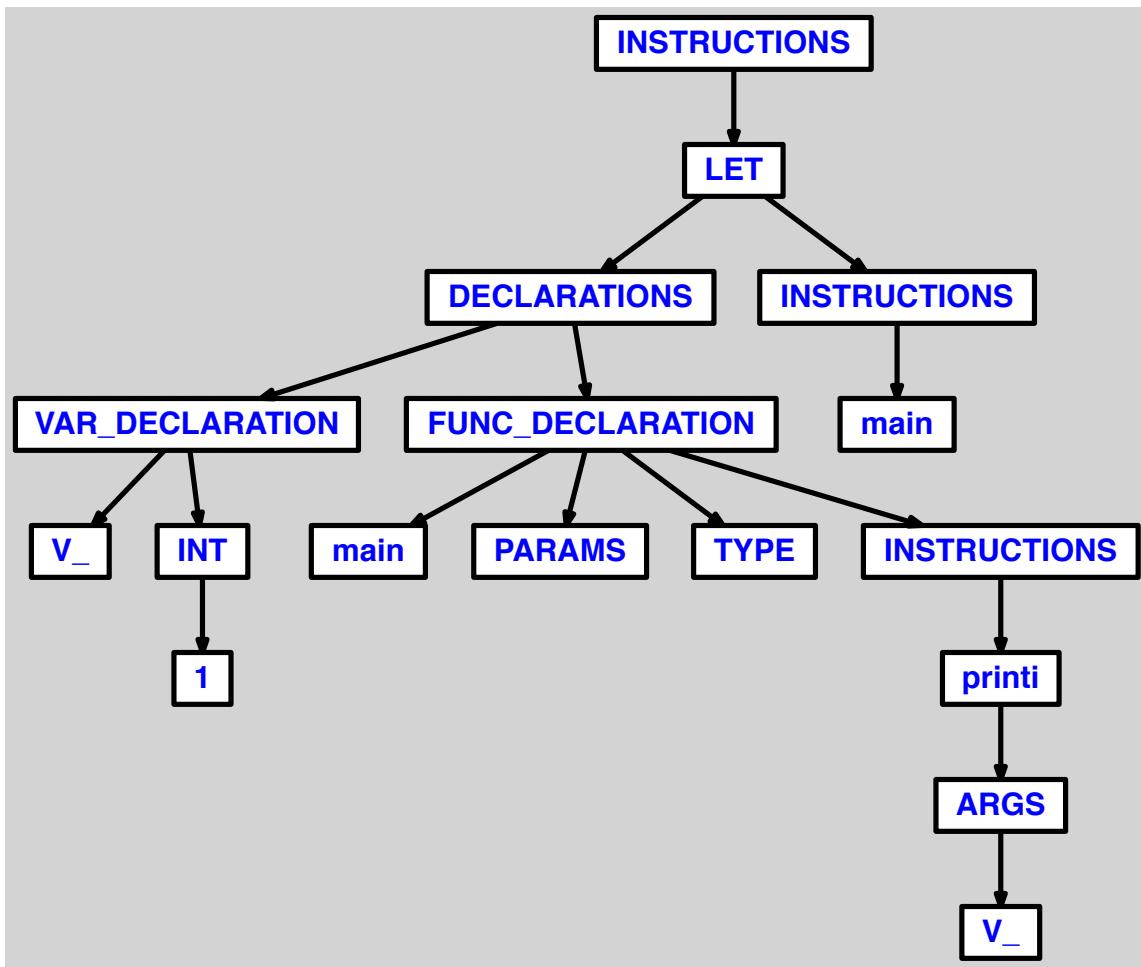


#### 11.2.18 lettre majuscule et \_ dans nom de variable

```

let
  var V_ := 1

  function main() = printi(V_)
in main() end
  
```



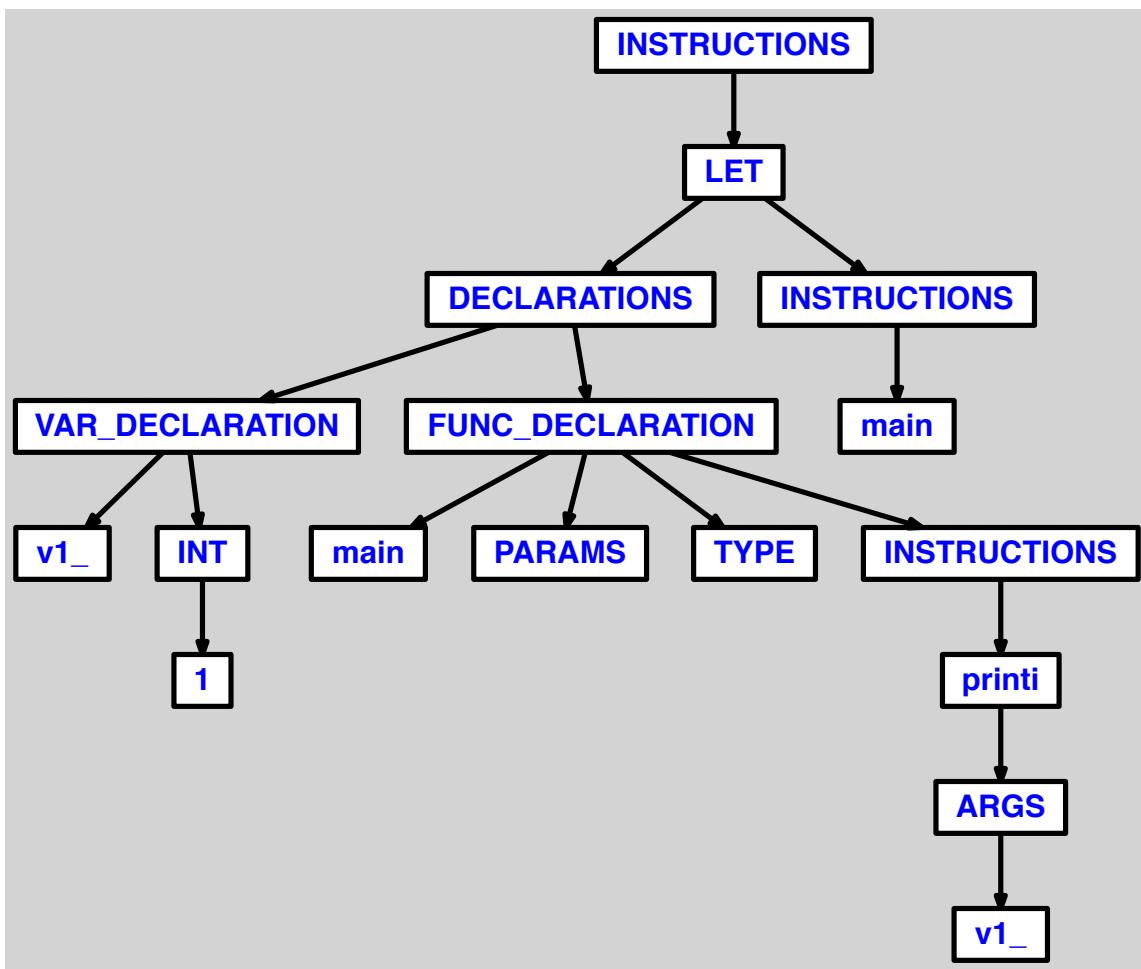
#### 11.2.19 lettre minuscule, chiffre et \_ dans nom de variable

```

let
    var v1_ := 1

    function main() = printi(v1_)
in main() end

```

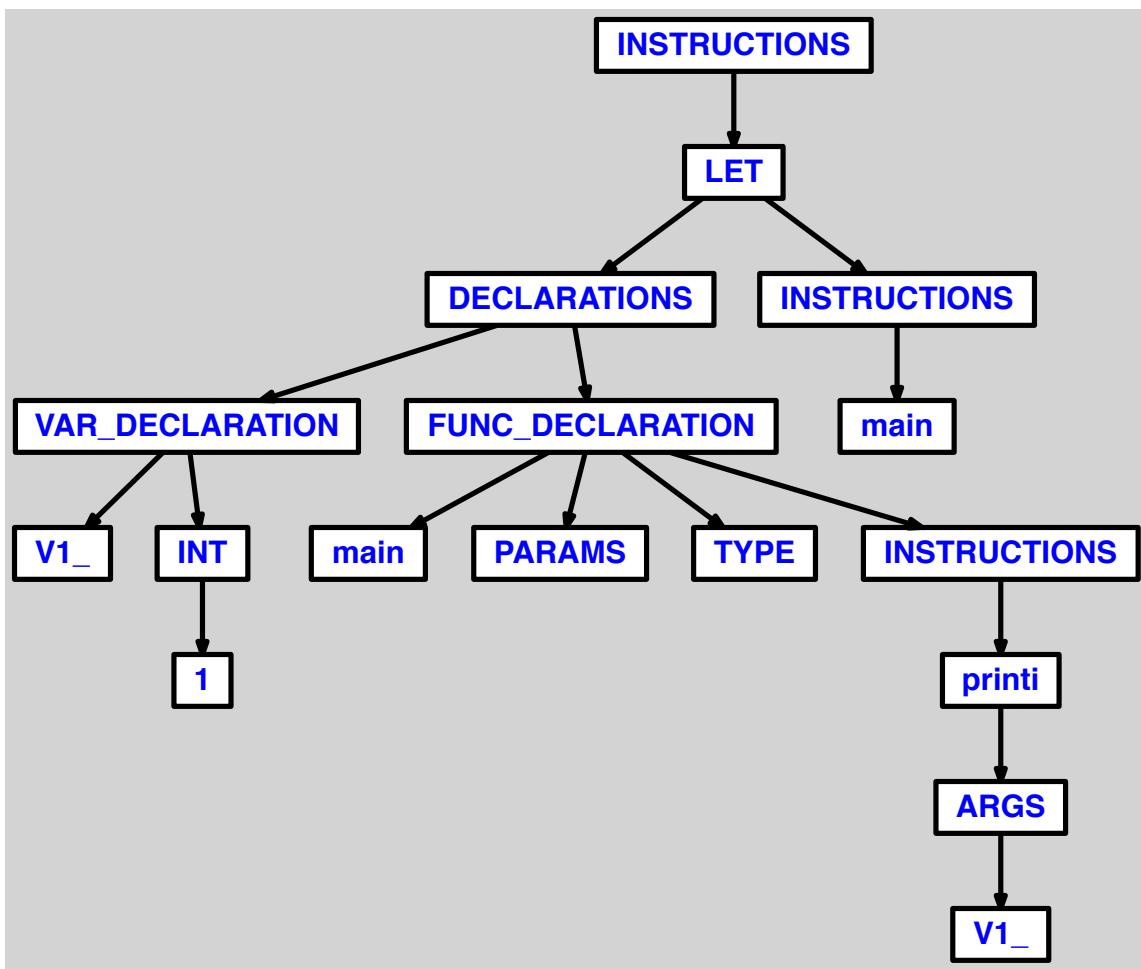


#### 11.2.20 lettre majuscule, chiffre et \_ dans nom de variable

```

let
  var V1_ := 1

  function main() = printi(V1_)
in main() end
  
```



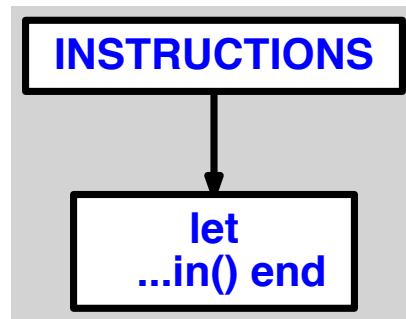
## 12 while

### 12.1 KO

#### 12.1.1 while avec affectation d'entier

```

let
    function main() =
        while i := 1 do
            print("test")
in main() end
  
```

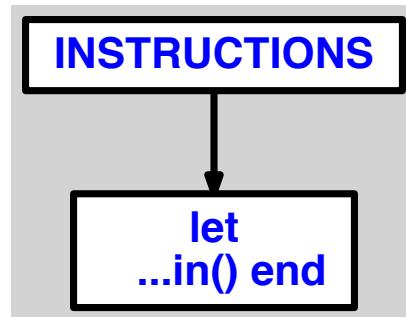


### 12.1.2 while avec affectation de chaine

```

let
    function main()()
        while s := "toto" do
            print("test")
    in main() end

```

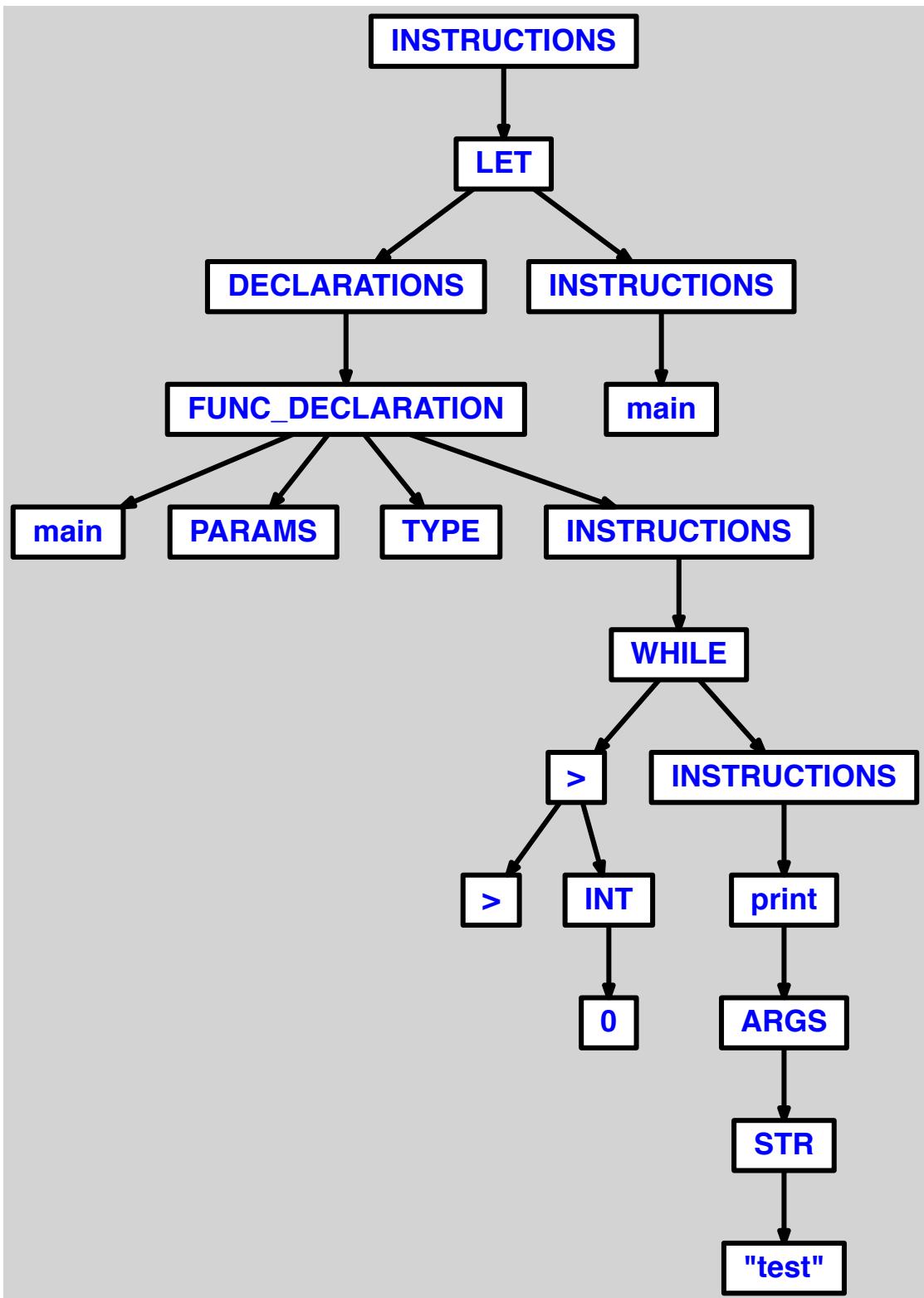


### 12.1.3 while avec oubli du compteur

```

let
    function main() =
        while > 0 do
            print("test")
    in main() end

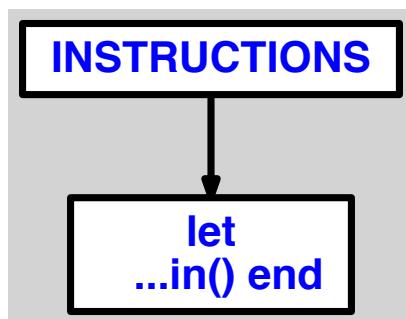
```



#### 12.1.4 while avec oubli du while

```
let
    var i := 0

    function main() =
        i >= 0 do
            print("test")
in main() end
```



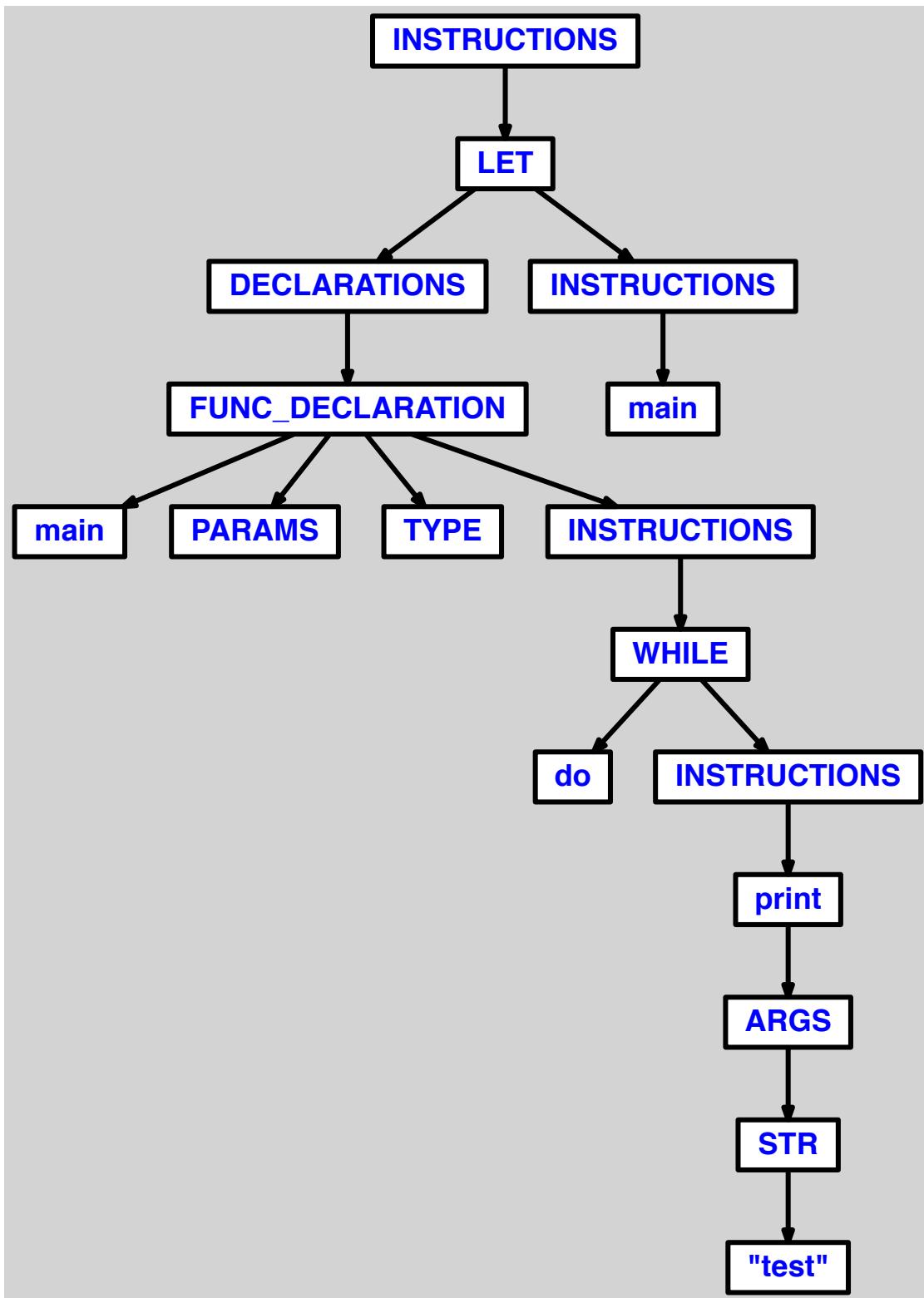
#### 12.1.5 while avec oubli du do

```
let
    var i := 0

    function main() :
        while i >= 0
            print("test")
in main() end
```

#### 12.1.6 while avec oubli de la condition

```
let
    function main() =
        while do
            print("test")
in main() end
```

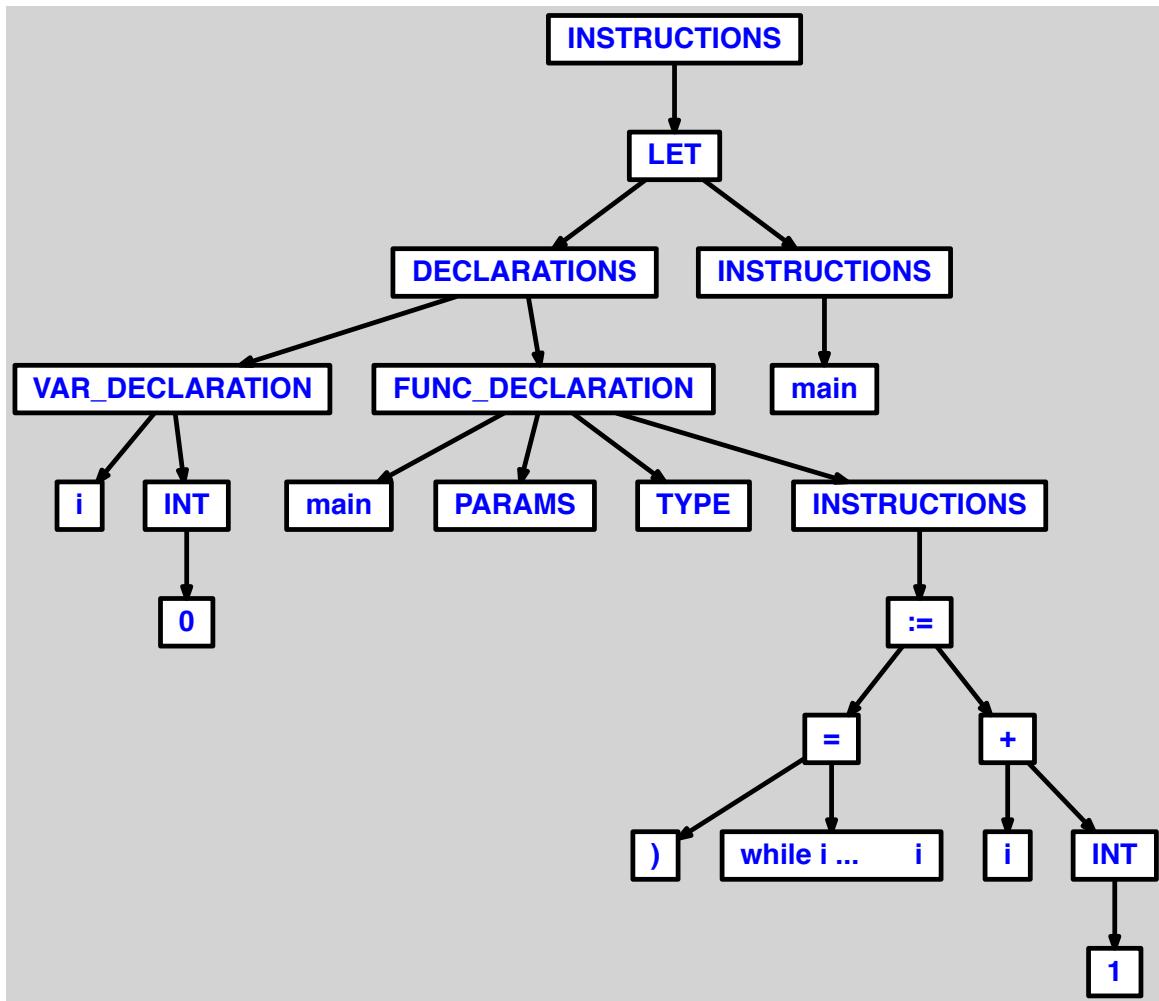


## 12.2 OK

### 12.2.1 while avec iteration croissante

```
let
    var i := 0

    function main()()
        while i <= 5 do
            printi(i);
            i := i+1
    in main() end
```



### 12.2.2 while avec iteration decroissante

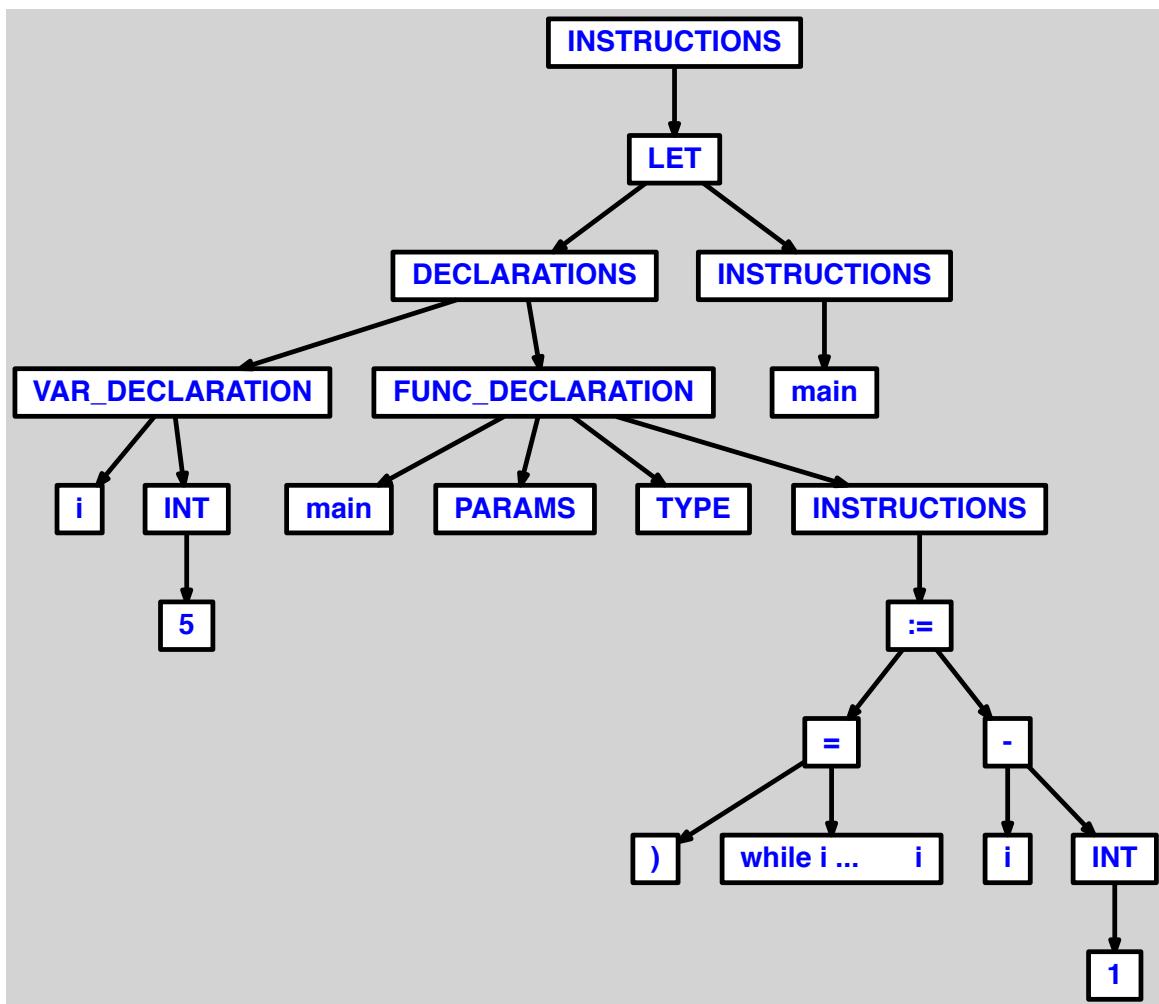
```
let
    var i := 5

    function main()()
        while i >= 0 do
            printi(i);
            i := i-1
    in main() end
```

```

while i >= 0 do
    printi(i);
    i := i-1
in main() end

```



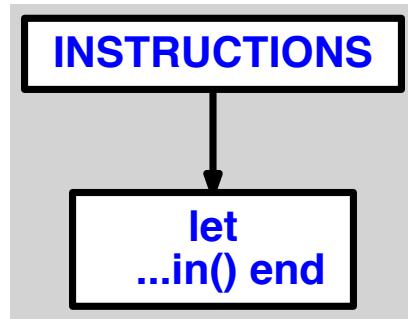
### 12.2.3 while avec double-condition

```

let
    var i := 0

    function main() =
        while i <= 5 & i >= 0 do
            printi(i);
            i := i+1
in main() end

```



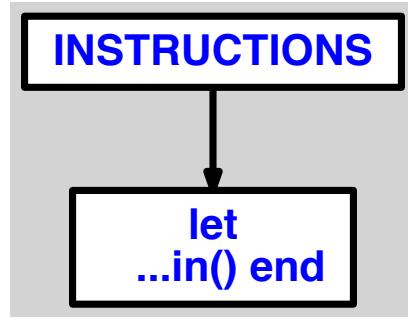
#### 12.2.4 while avec triple-condition

```

let
    var i := 5

    function main()()
        while i >= 0 & i <= 5 & i < 6 do
            printi(i);
            i := i-1
    in main() end

```



#### 12.2.5 while sans instruction

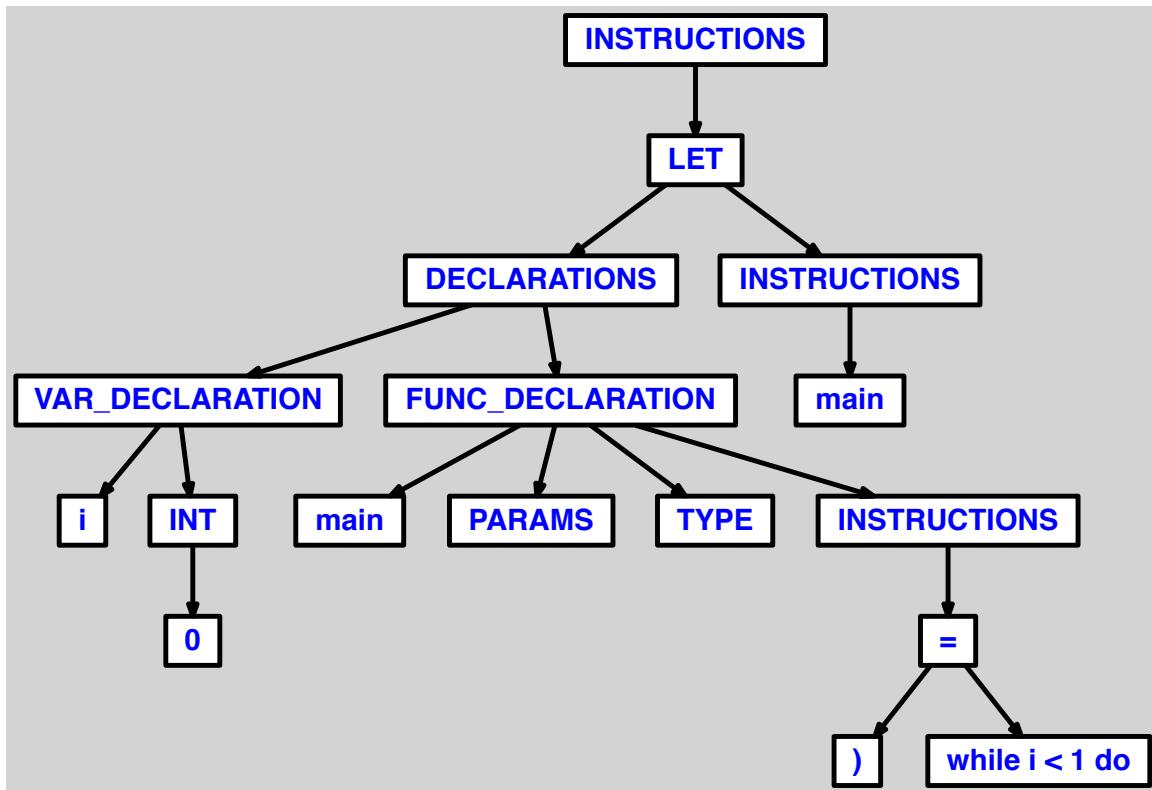
```

let
    var i := 0

    function main()()
        while i < 1 do

```

```
in main() end
```

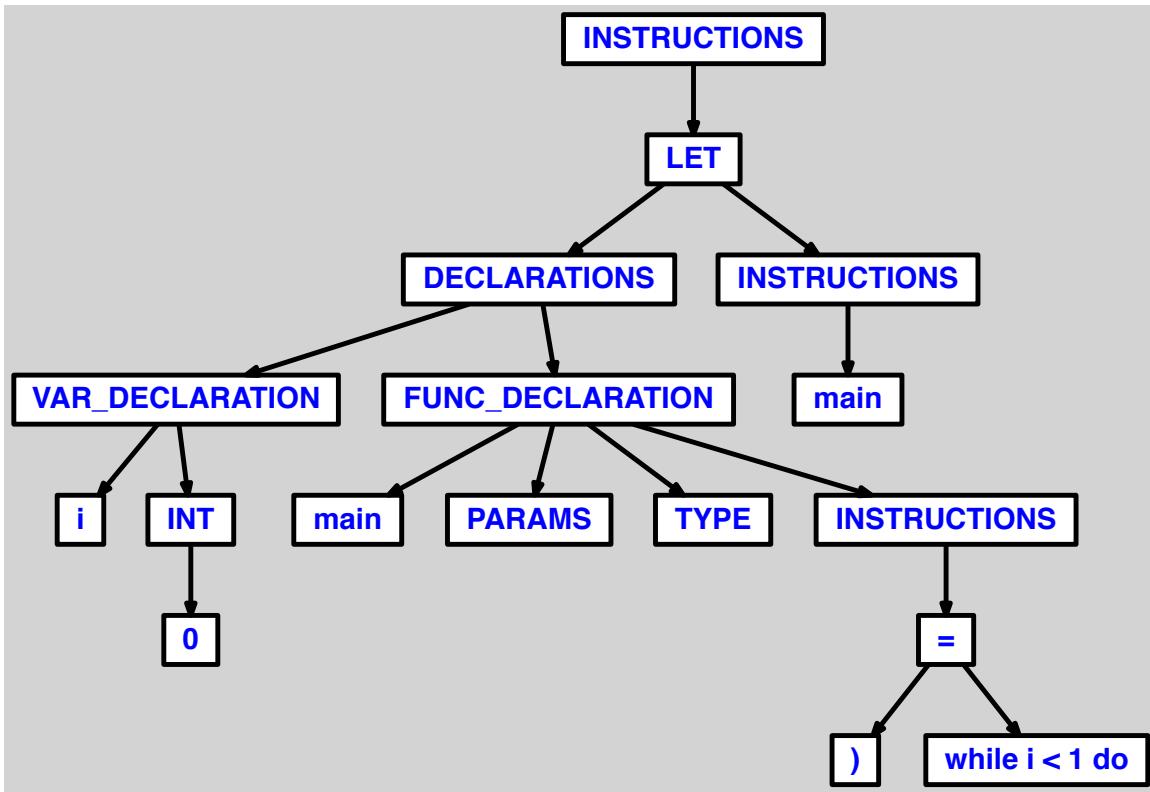


#### 12.2.6 while avec ligne vide

```
let
var i := 0

function main()()
    while i < 1 do

in main() end
```



### 12.2.7 while avec condition entière

```

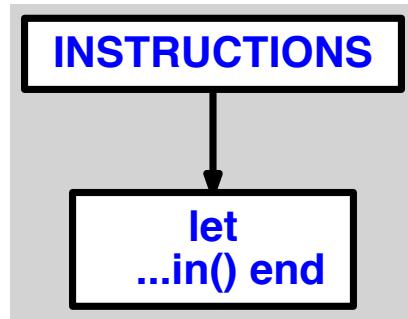
let
  function main()()
    while 1 do
      print("test")
in main() end
  
```

### 12.2.8 while avec double-imbrication

```

let
  var i := 0
  var j := 0

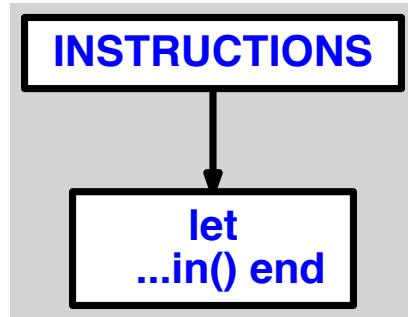
  function main()()
    while i < 1 do
      while j < 1 do
        i := i+1;
        j := j+1;
        print(i+j)
  in main() end
  
```



#### 12.2.9 while avec triple-imbrication

```
let
    var i := 0
    var j := 0
    var k := 0

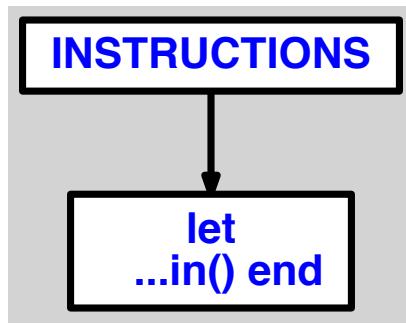
    function main()()
        while i < 1 do
            while j < 1 do
                while k < 1 do
                    i := i+1;
                    j := j+1;
                    k := k+1;
                    printi(i+j+k)
    in main() end
```



### 12.2.10 while avec reutilisation de compteur

```
let
    var i := 0
    var j := 0

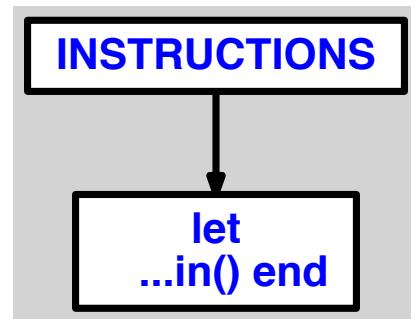
    function main()()
        while i < 1 do
            while j < i+1 do
                i := i+1;
                j := j+2;
                printi(i+j)
in main() end
```



### 12.2.11 while avec reutilisation de compteurs

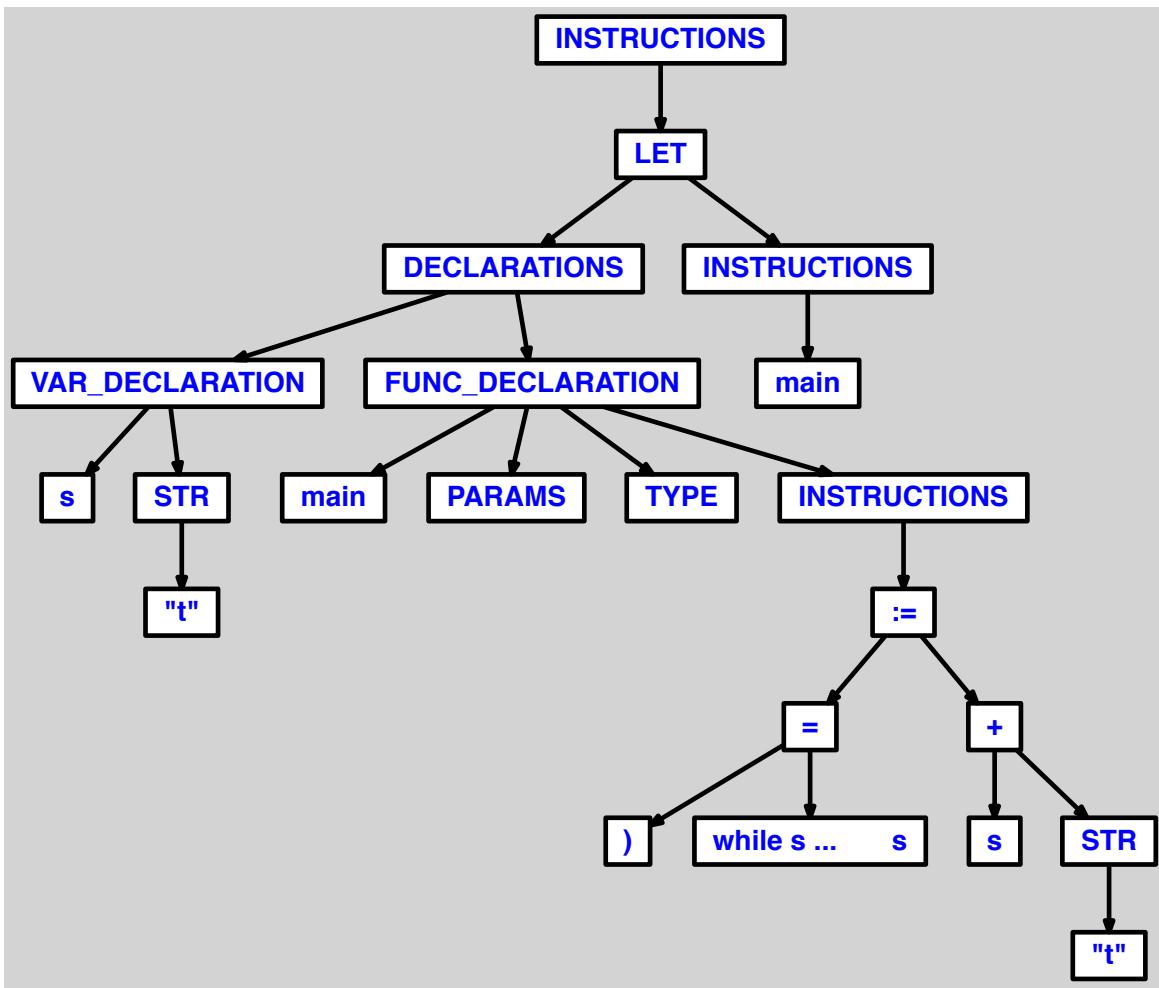
```
let
    var i := 0
    var j := 0
    var k := 0

    function main()()
        while i < 1 do
            while j < i+1 do
                while k < j+1 do
                    i := i+1;
                    j := j+2;
                    k := k+3;
                    printi(i+j+k)
in main() end
```



#### 12.2.12 while avec iteration sur une chaine

```
let
  var s := "t"
  function main()()
    while s <> "ttt" do
      s := s+"t"
  in main() end
```

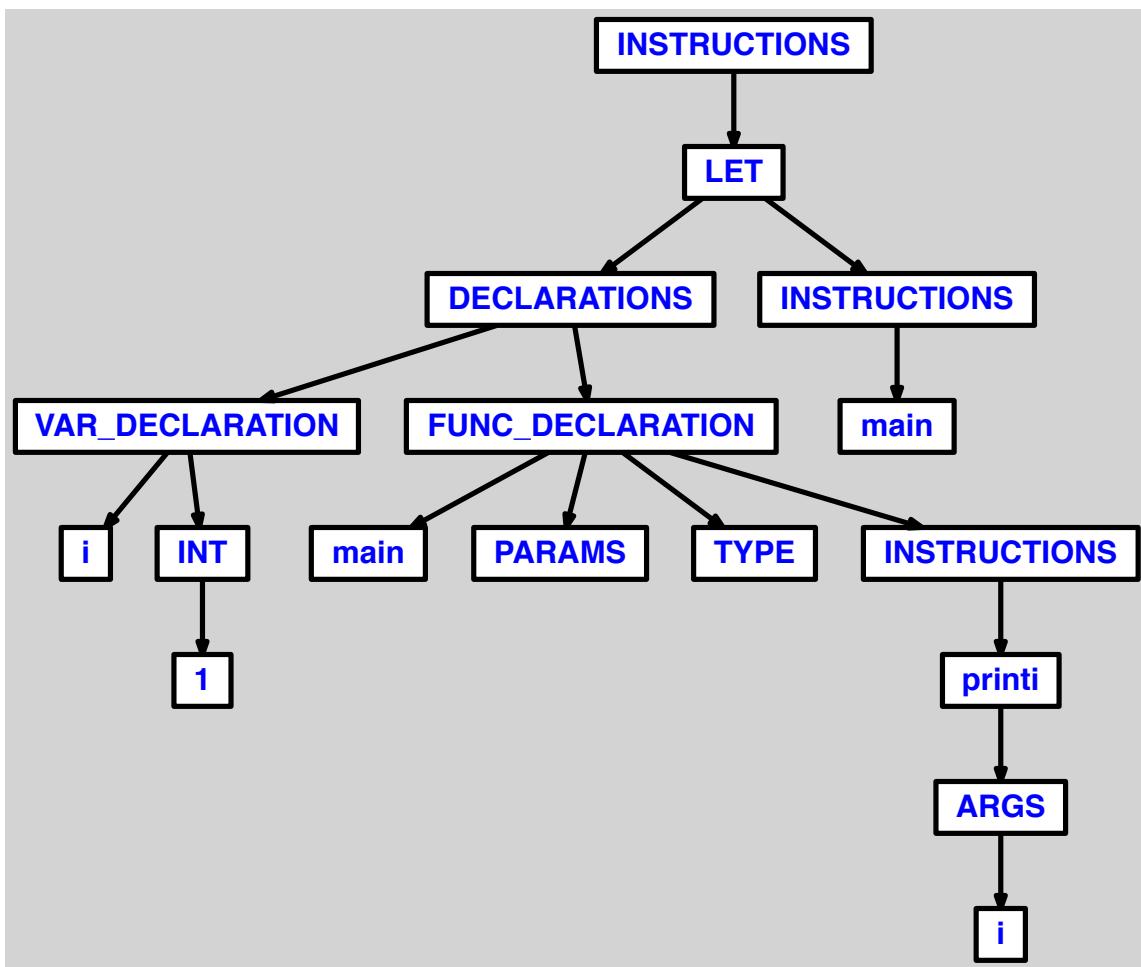


## 12.3 OK

### 12.3.1 1 espace entre var et i

```
let
  var i := 1
```

```
function main() = printi(i)
in main() end
```

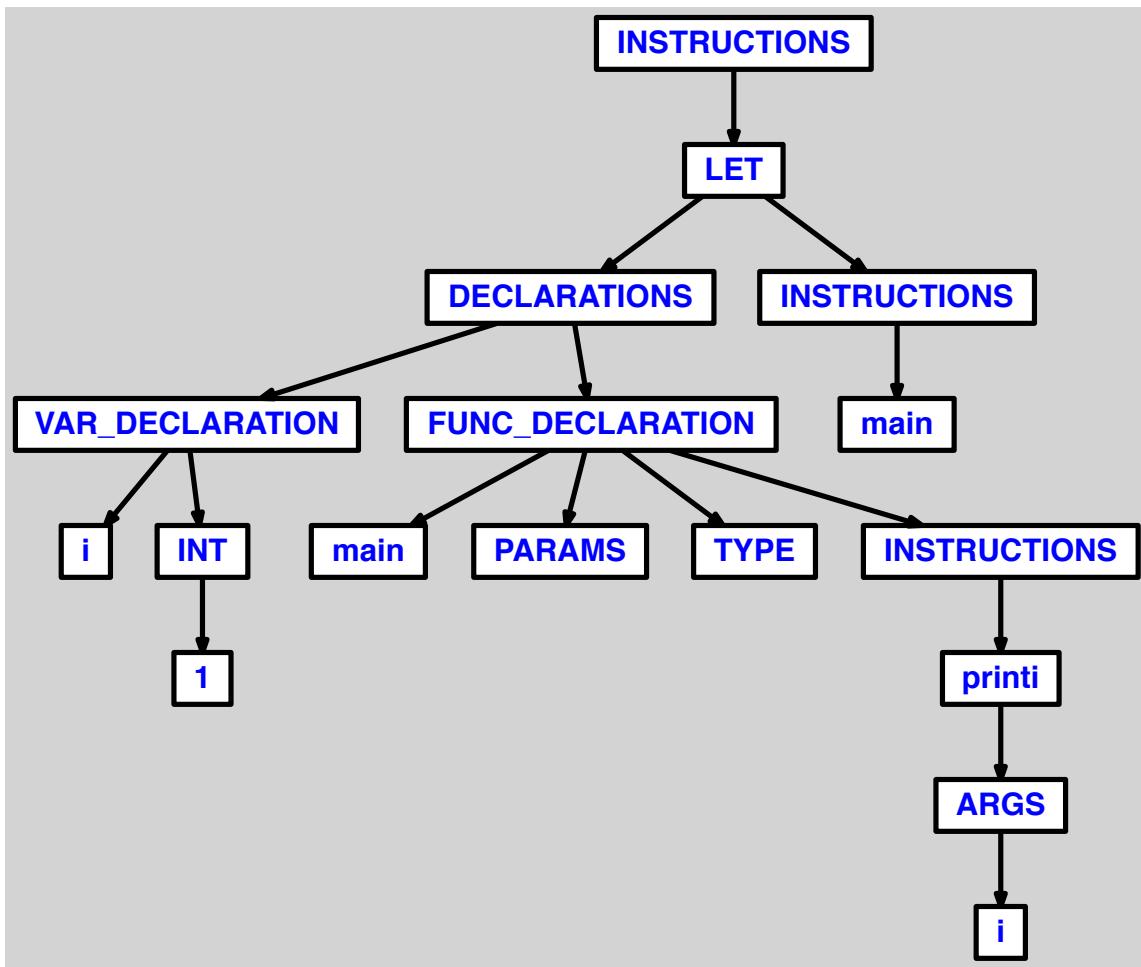


### 12.3.2 1 tabulation entre var et i

```

let
  var      i := 1

  function main() = printi(i)
in main() end
  
```

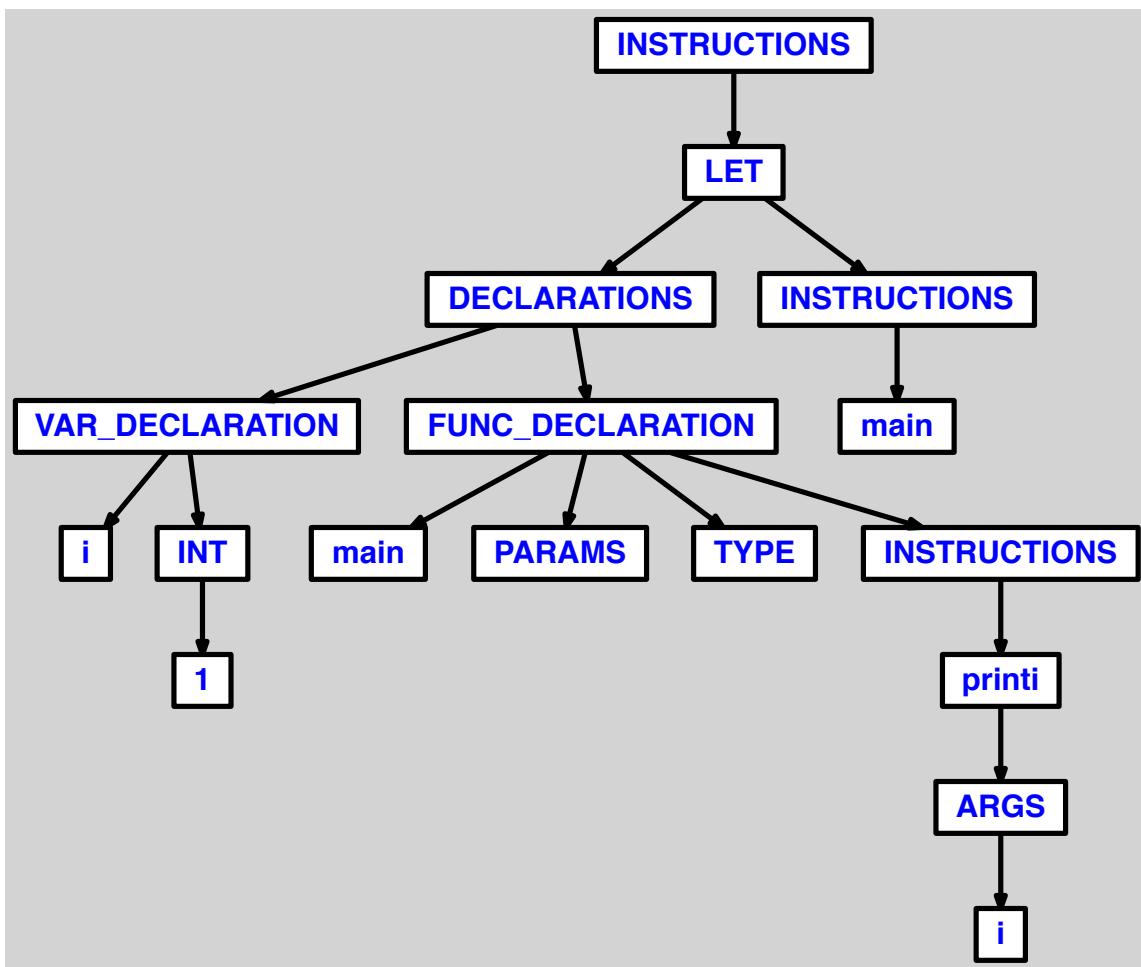


### 12.3.3 1 saut de ligne entre var et i

```

let
  var
  i := 1

  function main() = printi(i)
in main() end
  
```

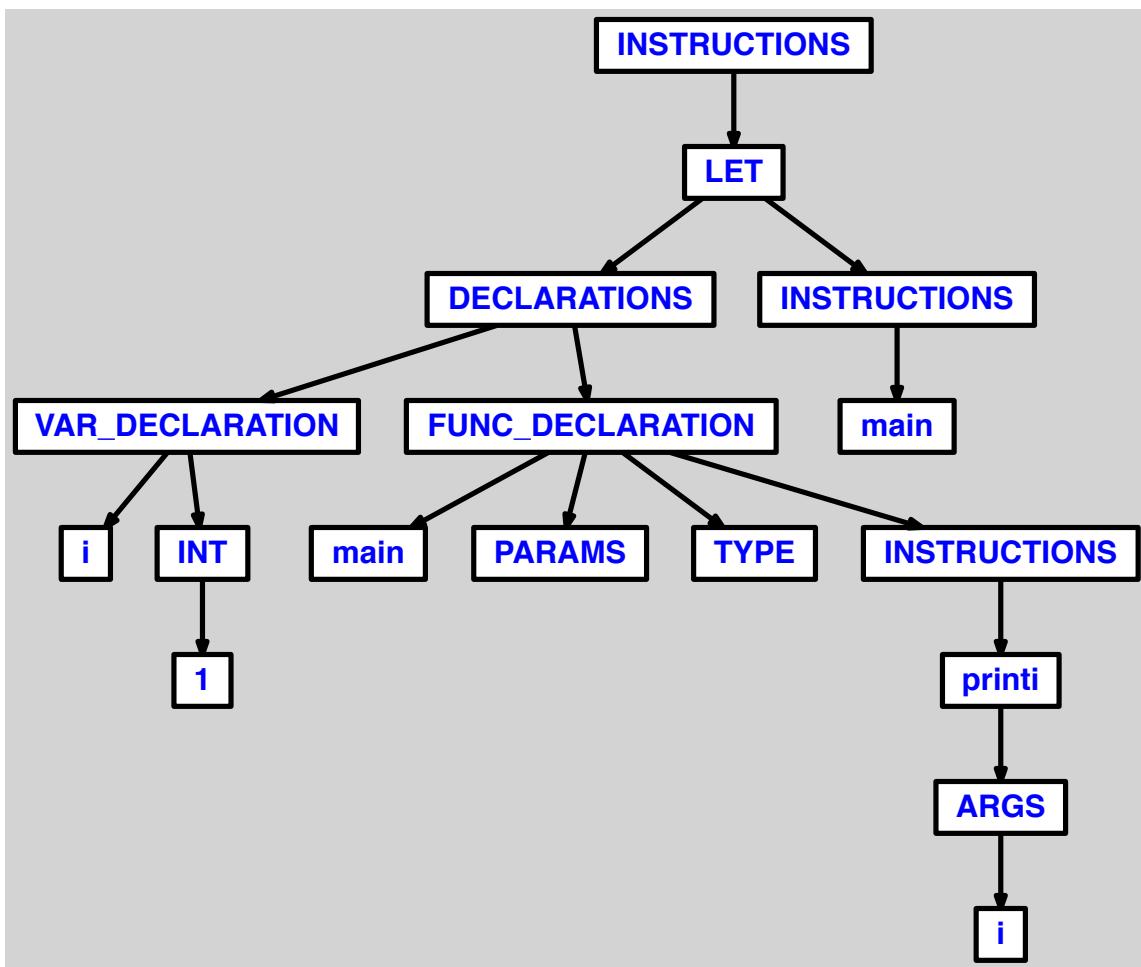


#### 12.3.4 2 espaces entre var et i

```

let
  var  i := 1

  function main() = printi(i)
in main() end
  
```

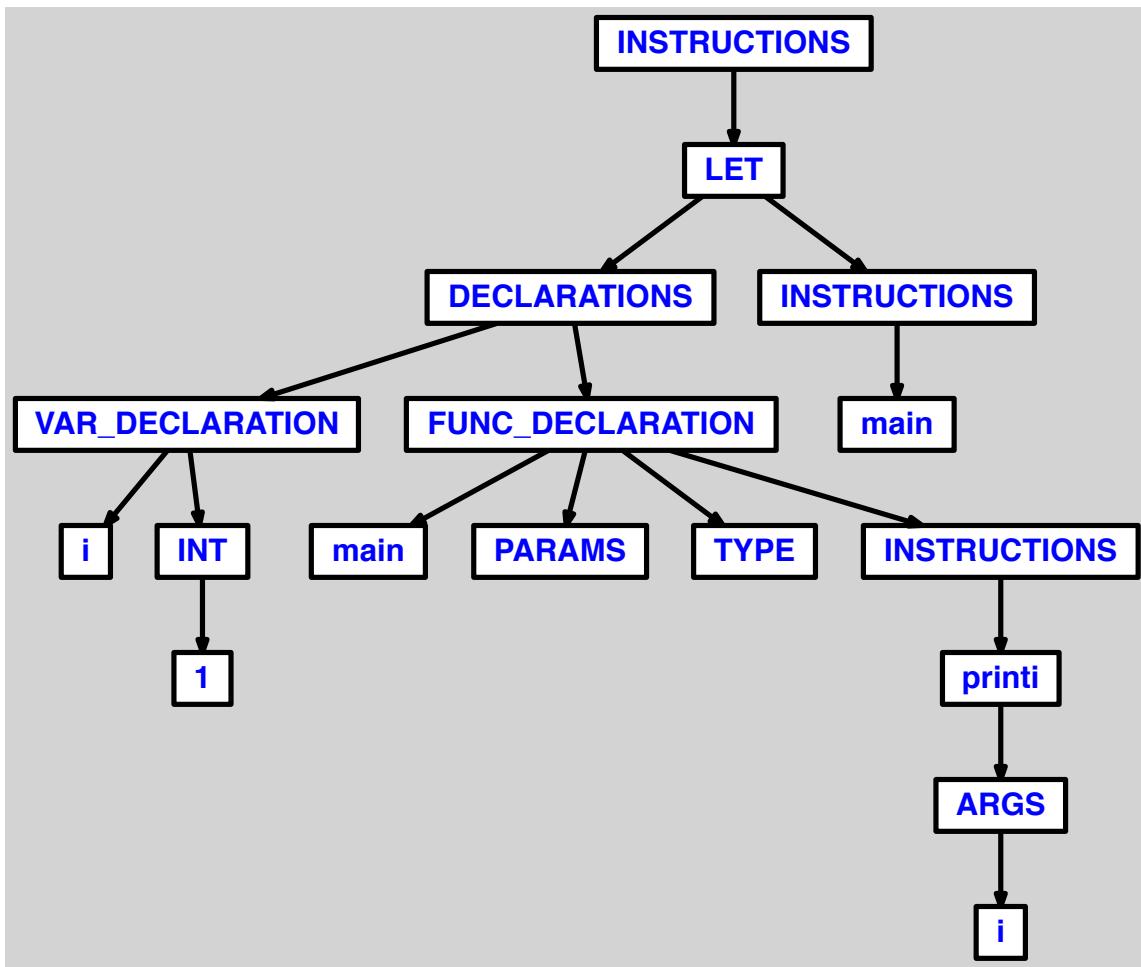


### 12.3.5 2 tabulations entre var et i

```

let
    var           i := 1

    function main() = printi(i)
in main() end
  
```

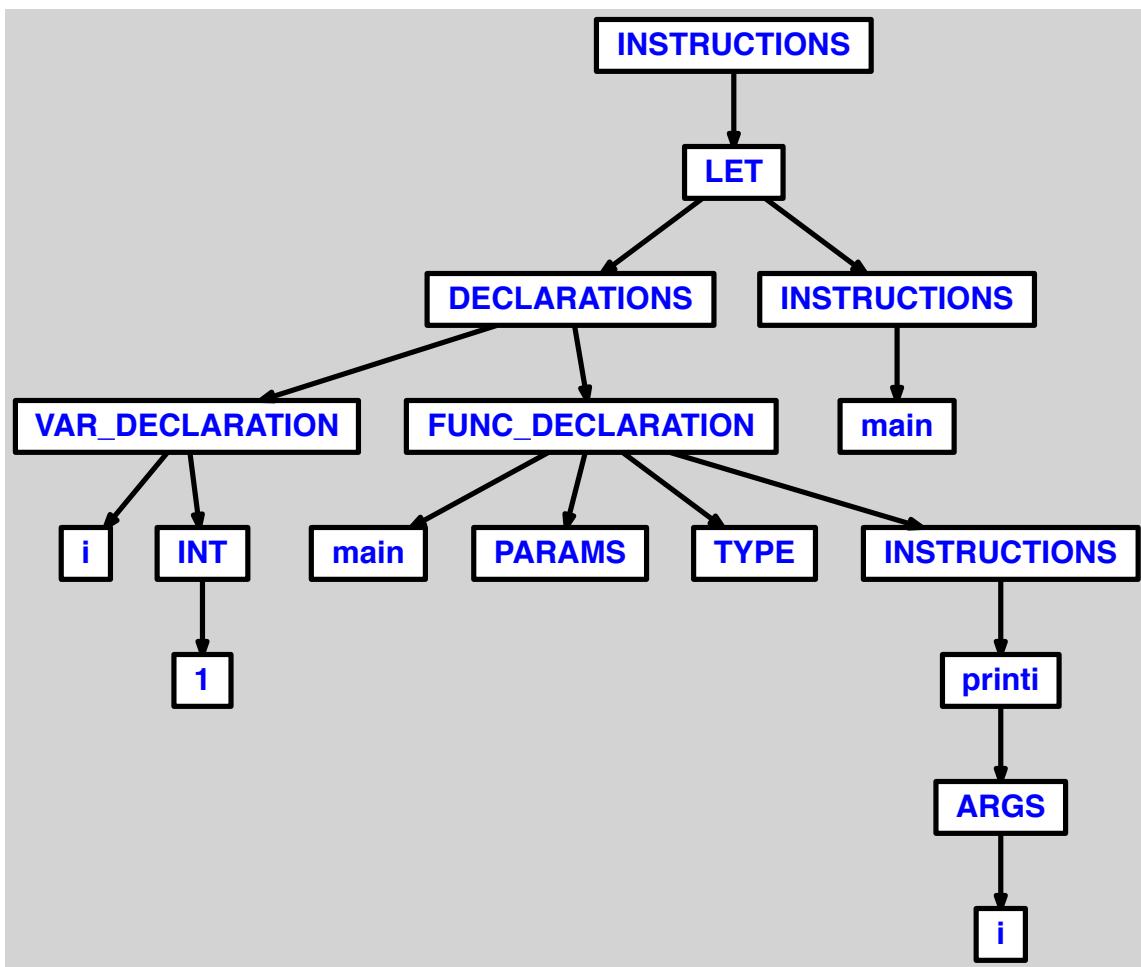


### 12.3.6 2 sauts de ligne entre var et i

```

let
  var
    i := 1

    function main() = printi(i)
in main() end
  
```

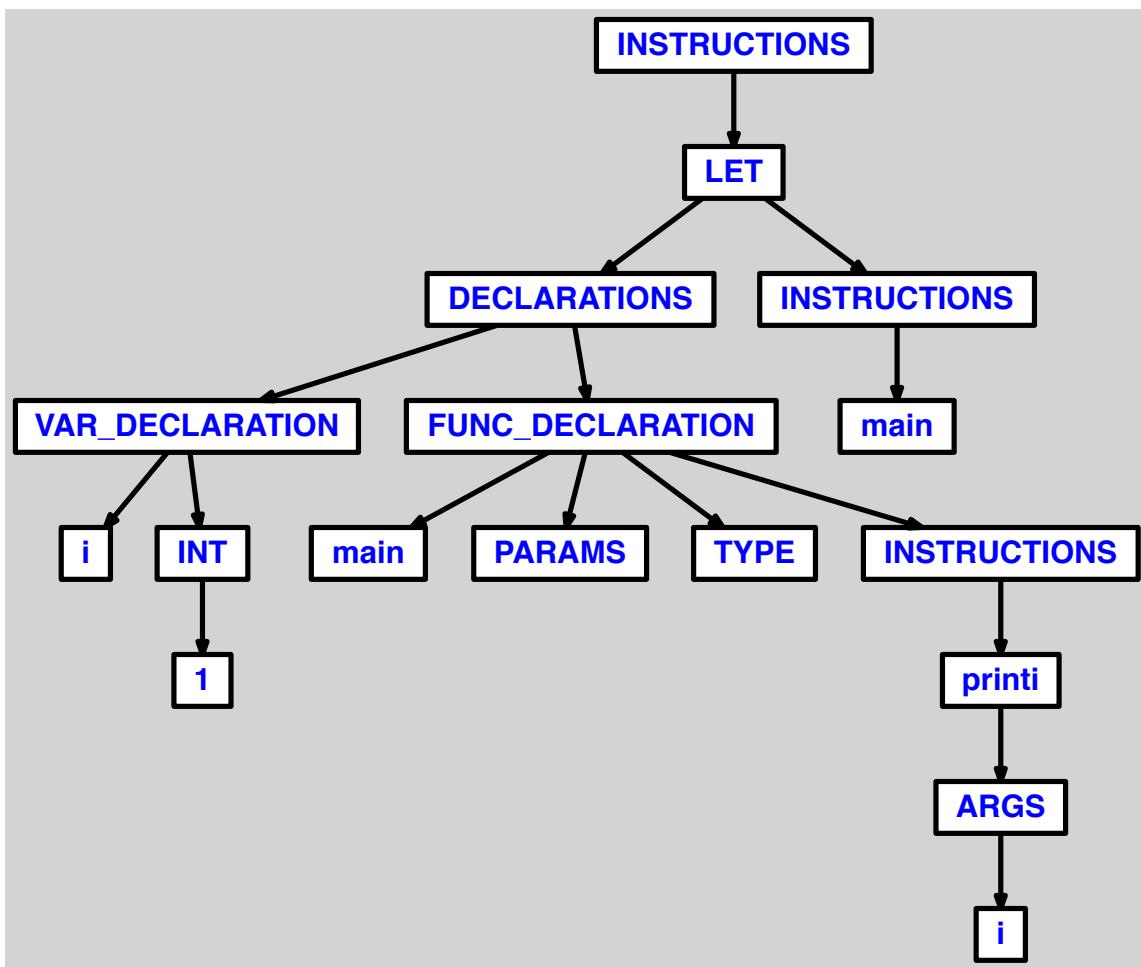


### 12.3.7 3 espaces entre var et i

```

let
  var    i := 1

  function main() = printi(i)
in main() end
  
```

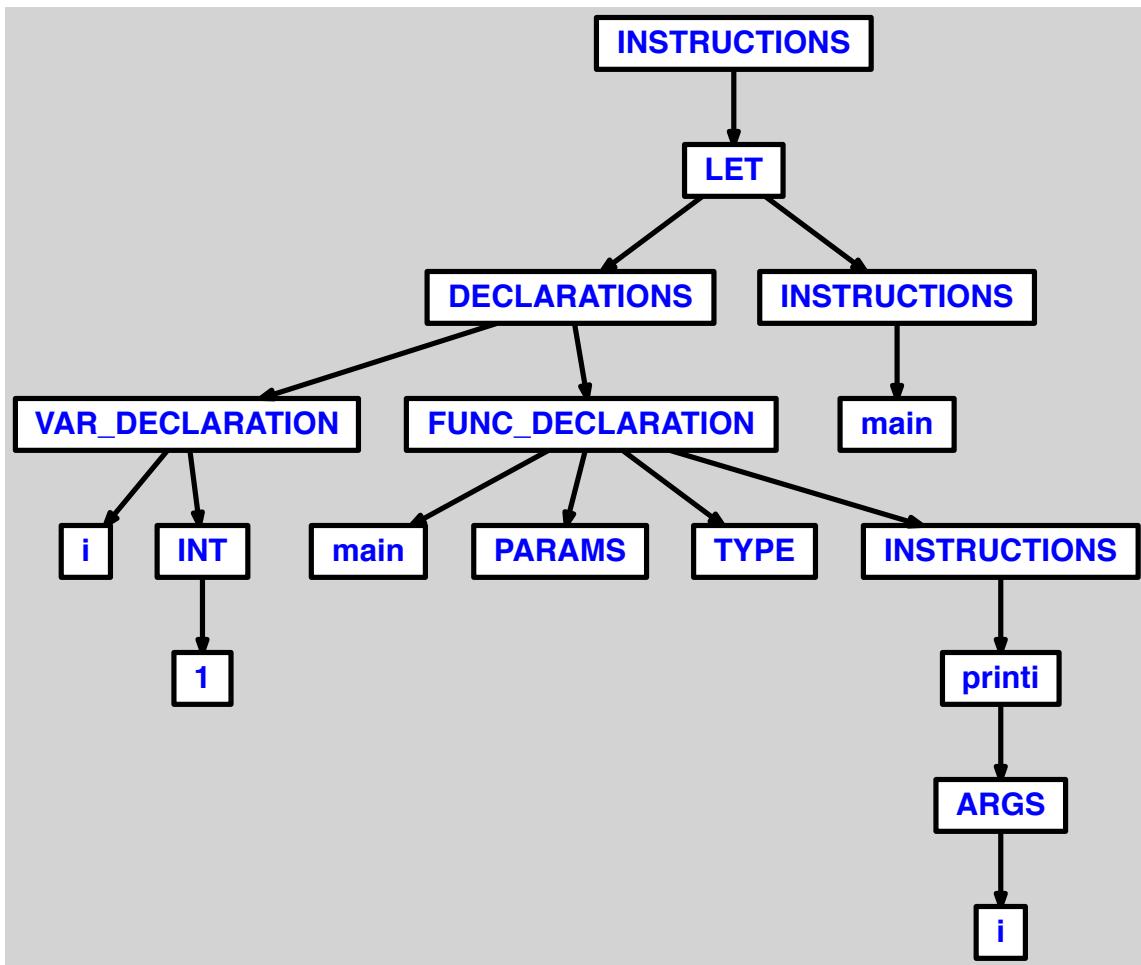


#### 12.3.8 3 tabulations entre var et i

```

let
    var           i := 1

        function main() = printi(i)
in main() end
  
```



### 12.3.9 3 sauts de ligne entre var et i

```

let
var

i := 1

function main() = printi(i)
in main() end
  
```

