

# 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 . . . . .	3
1.2.1 condition avec 1 <&> . . . . .	3
1.2.2 condition avec 2 <&> . . . . .	5
1.2.3 condition avec 3 <&> . . . . .	7
1.2.4 condition avec 2 <&> et 1 < > . . . . .	9
1.2.5 <&> avec des entiers . . . . .	11
1.2.6 <&> avec des chaînes . . . . .	13
<b>2 break</b>	<b>15</b>
2.1 KO . . . . .	15
2.1.1 <break> comme condition dans <if> . . . . .	15
2.1.2 <break> comme condition dans <while> . . . . .	17
2.1.3 appel de <break>, avec corps de programme contenant <break> . . . . .	19
2.2 OK . . . . .	21
2.2.1 <break> dans un <for>, après instruction . . . . .	21
2.2.2 <break> dans un <for>, sans instruction avant . . . . .	23
2.2.3 fonction exécutant seulement <break> . . . . .	25
2.2.4 appel de <break>, avec corps de programme ne contenant pas <break> . . . . .	27
2.2.5 <break> dans un <while>, après instruction . . . . .	29
2.2.6 <break> dans un <while>, sans instruction avant . . . . .	31
2.2.7 <break> dans un <if then>, après instruction . . . . .	33
2.2.8 <break> dans un <if then>, sans instruction avant . . . . .	35
2.2.9 <break> dans un <if then else>, après instruction . . . . .	37
2.2.10 <break> dans un <if then else>, sans instruction avant . . . . .	39
2.2.11 <break> seul . . . . .	41
2.2.12 <break> seul, après saut de ligne . . . . .	43
<b>3 calcul</b>	<b>45</b>
3.1 KO . . . . .	45
3.1.1 addition avec opérateur mal écrit . . . . .	45
3.1.2 soustraction avec opérateur mal écrit . . . . .	47
3.1.3 multiplication avec opérateur mal écrit . . . . .	49
3.1.4 division avec opérateur mal écrit . . . . .	51
3.1.5 moins unaire avec opérateur mal écrit . . . . .	53
3.1.6 calcul à 2 termes avec oubli de parenthèse gauche . . . . .	55
3.1.7 calcul à 2 termes avec oubli de parenthèse droite . . . . .	57
3.1.8 calcul à 3 termes avec oubli de parenthèse gauche . . . . .	59
3.1.9 calcul à 3 termes avec oubli de parenthèse droite . . . . .	61
3.1.10 calcul à 4 termes avec oubli de parenthèse gauche . . . . .	63
3.1.11 calcul à 4 termes avec oubli de parenthèse droite . . . . .	65
3.1.12 parenthesage sans contenu . . . . .	67
3.1.13 expression parenthesée avec <;> en trop . . . . .	69
3.1.14 expression parenthesée avec oubli de <;> . . . . .	71
3.1.15 expression non parenthesée avec oubli de <;> . . . . .	73
3.1.16 expression non parenthesée avec <;> en trop . . . . .	75
3.1.17 <;> sans instruction avant . . . . .	77
3.2 OK . . . . .	79
3.2.1 addition simple, à 2 termes . . . . .	79
3.2.2 soustraction simple, à 2 termes . . . . .	81
3.2.3 multiplication simple, à 2 termes . . . . .	83
3.2.4 division simple, à 2 termes . . . . .	85
3.2.5 addition à 3 termes . . . . .	87

3.2.6	addition suivie de soustraction . . . . .	89
3.2.7	addition suivie de multiplication . . . . .	91
3.2.8	addition suivie de division . . . . .	93
3.2.9	multiplication suivie d'addition . . . . .	95
3.2.10	multiplication suivie de soustraction . . . . .	97
3.2.11	multiplication a 3 termes . . . . .	99
3.2.12	multiplication suivie de division . . . . .	101
3.2.13	addition simple, a 2 termes, identifies par des variables . . . . .	103
3.2.14	soustraction simple, a 2 termes, identifies par des variables . . . . .	105
3.2.15	multiplication simple, a 2 termes, identifies par des variables . . . . .	107
3.2.16	division simple, a 2 termes, identifies par des variables . . . . .	109
3.2.17	addition a 3 termes, identifies par des variables . . . . .	111
3.2.18	addition suivie de soustraction, avec termes identifies par variables . . . . .	113
3.2.19	addition suivie de multiplication, avec termes identifies par variables . . . . .	115
3.2.20	addition suivie de division, avec termes identifies par variables . . . . .	117
3.2.21	multiplication suivie d'addition, avec termes identifies par variables . . . . .	119
3.2.22	multiplication suivie de soustraction, avec termes identifies par variables . . . . .	121
3.2.23	multiplication a 3 termes, identifies par des variables . . . . .	123
3.2.24	multliplication suivie de division, avec termes identifies par variables . . . . .	125
3.2.25	addition simple, a 2 termes, dont un ayant moins unaire . . . . .	127
3.2.26	soustraction simple, a 2 termes, dont ayant moins unaire . . . . .	129
3.2.27	multiplication simple, a 2 termes, dont un ayant moins unaire . . . . .	131
3.2.28	division simple, a 2 termes, dont un ayant moins unaire . . . . .	133
3.2.29	addition a 3 termes, dont un ayant moins unaire . . . . .	135
3.2.30	addition suivie de soustraction, avec un terme ayant moins unaire . . . . .	137
3.2.31	addition suivie de multiplication, avec un terme ayant moins unaire . . . . .	139
3.2.32	addition suivie de division, avec un terme ayant moins unaire . . . . .	141
3.2.33	multiplication suivie d'addition, dont un terme ayant moins unaire . . . . .	143
3.2.34	multiplication suivie de soustraction, dont un terme ayant moins unaire . . . . .	145
3.2.35	multiplication a 3 termes, dont un ayant moins unaire . . . . .	147
3.2.36	multiplication suivie de division, dont un terme ayant moins unaire . . . . .	149
3.2.37	addition simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	151
3.2.38	soustraction simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	153
3.2.39	multiplication simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	155
3.2.40	division simple, a 2 termes, identifies par des variables, dont une ayant moins unaire . . . . .	157
3.2.41	addition a 3 termes, identifies par des variables, dont une ayant moins unaire . . . . .	159
3.2.42	addition suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire . . . . .	161
3.2.43	addition suivie de multiplication, avec termes identifies par variables, dont une ayant moins unaire . . . . .	163
3.2.44	addition suivie de division, avec termes identifies par variables, dont une ayant moins unaire . . . . .	165
3.2.45	multiplication suivie d'addition, avec termes identifies par variables, dont une ayant moins unaire . . . . .	167
3.2.46	multiplication suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire . . . . .	169
3.2.47	multiplication a 3 termes, identifies par des variables, dont une ayant moins unaire . . . . .	171
3.2.48	multlication suivie de division, avec termes identifies par variables, dont une ayant moins unaire . . . . .	173
3.2.49	addition a 3 termes, avec parenthesage des 2 termes a droite . . . . .	175
3.2.50	addition suivie de soustraction, avec parenthesage des 2 termes a droite . . . . .	177
3.2.51	addition suivie de multiplication, avec parenthesage des 2 termes a droite . . . . .	179
3.2.52	addition suivie de division, avec parenthesage des 2 termes a droite . . . . .	181
3.2.53	multiplication suivie d'addition, avec parenthesage des 2 termes a droite . . . . .	183
3.2.54	multiplication suivie de soustraction, avec parenthesage des 2 termes a droite . . . . .	185
3.2.55	multiplication a 3 termes, avec parenthesage des 2 termes a droite . . . . .	187
3.2.56	multiplication suivie de vision, avec parenthesage des 2 termes a droite . . . . .	189
3.2.57	addition a 3 termes, avec parenthesage des 2 termes a gauche . . . . .	191

3.2.58 addition suivie de soustraction, avec parenthesage des 2 termes à gauche . . . . .	193
3.2.59 addition suivie de multiplication, avec parenthesage des 2 termes à gauche . . . . .	195
3.2.60 addition suivie de division, avec parenthesage des 2 termes à gauche . . . . .	197
3.2.61 multiplication suivie d'addition, avec parenthesage des 2 termes à gauche . . . . .	199
3.2.62 multiplication suivie de soustraction, avec parenthesage des 2 termes à gauche . . . . .	201
3.2.63 multiplication à 3 termes, avec parenthesage des 2 termes à gauche . . . . .	203
3.2.64 multiplication suivie de division, avec parenthesage des 2 termes à gauche . . . . .	205
3.2.65 multiplication de 2 additions parenthèses . . . . .	207
3.2.66 multiplication de 3 additions parenthèses . . . . .	209
3.2.67 division de 2 soustractions parenthèses . . . . .	211
3.2.68 division de 3 soustractions parenthèses . . . . .	213
3.2.69 addition de 2 multiplications parenthèses . . . . .	215
3.2.70 addition de 3 multiplications parenthèses . . . . .	217
3.2.71 soustraction de 2 divisions parenthèses . . . . .	219
3.2.72 soustraction de 3 divisions parenthèses . . . . .	221
3.2.73 addition à 3 termes identifiés par variables, avec parenthesage des 2 variables à droite . . . . .	223
3.2.74 addition suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	225
3.2.75 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	227
3.2.76 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	229
3.2.77 multiplication suivie d'addition, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	231
3.2.78 multiplication suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	233
3.2.79 multiplication à 3 termes identifiés par variables, avec parenthesage des 2 variables à droite . . . . .	235
3.2.80 multiplication suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite . . . . .	237
3.2.81 addition à 3 termes identifiés par variables, avec parenthesage des 2 variables à gauche . . . . .	239
3.2.82 addition suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	241
3.2.83 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	243
3.2.84 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	245
3.2.85 multiplication suivie d'addition, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	247
3.2.86 multiplication suivie de soustraction, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	249
3.2.87 multiplication à 3 termes identifiés par variables, avec parenthesage des 2 variables à gauche . . . . .	251
3.2.88 multiplication suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche . . . . .	253
3.2.89 multiplication de 2 additions parenthèses, avec termes identifiés par variables . . . . .	255
3.2.90 multiplication de 3 additions parenthèses, avec termes identifiés par variables . . . . .	257
3.2.91 division de 2 soustractions parenthèses, avec termes identifiés par variables . . . . .	259
3.2.92 division de 3 soustractions parenthèses, avec termes identifiés par variables . . . . .	261
3.2.93 addition de 2 multiplications parenthèses, avec termes identifiés par variables . . . . .	263
3.2.94 addition de 3 multiplications parenthèses, avec termes identifiés par variables . . . . .	265
3.2.95 soustraction de 2 divisions parenthèses, avec termes identifiés par variables . . . . .	267
3.2.96 soustraction de 3 divisions parenthèses, avec termes identifiés par variables . . . . .	269
3.2.97 parenthesage d'une instruction . . . . .	271
3.2.98 parenthesage de 2 instructions . . . . .	273
3.2.99 parenthesage de 3 instructions . . . . .	275
3.2.100 1 instruction sans parenthèses . . . . .	277
3.2.101 2 instructions sans parenthèses . . . . .	279

3.2.102	3 instructions sans parenthèses . . . . .	281
3.2.103	concatenation d'entier et de chaine . . . . .	283
3.2.104	concatenation de 2 chaines . . . . .	285
<b>4</b>	<b>commentaire</b>	<b>287</b>
4.1	KO . . . . .	287
4.1.1	imbrication . . . . .	287
4.1.2	oubli de </> en debut de commentaire . . . . .	289
4.1.3	oubli de <*> en debut de commentaire . . . . .	291
4.1.4	oubli de <*/> en debut de commentaire . . . . .	293
4.1.5	oubli de <*> en fin de commentaire . . . . .	295
4.1.6	oubli de </> en fin de commentaire . . . . .	297
4.1.7	oubli de <*> en fin de commentaire . . . . .	299
4.2	OK . . . . .	301
4.2.1	1 commentaire, sans instruction avant, sur 1 ligne . . . . .	301
4.2.2	1 commentaire, sans instruction avant, sur plusieurs lignes . . . . .	303
4.2.3	1 commentaire, apres instruction, sur 1 ligne . . . . .	305
4.2.4	1 commentaire, apres instruction, sur plusieurs lignes . . . . .	307
4.2.5	2 commentaires, sans instruction avant, sur 1 ligne . . . . .	309
4.2.6	2 commentaires, sans instruction avant, sur plusieurs lignes . . . . .	311
4.2.7	2 commentaires, apres instructions, sur 1 ligne . . . . .	313
4.2.8	2 commentaires, apres instructions, sur plusieurs lignes . . . . .	315
4.2.9	3 commentaires, sans instruction avant, sur 1 ligne . . . . .	317
4.2.10	3 commentaires, sans instruction avant, sur plusieurs lignes . . . . .	319
4.2.11	3 commentaires, apres instructions, sur 1 ligne . . . . .	321
4.2.12	3 commentaires, apres instructions, sur plusieurs lignes . . . . .	323
4.2.13	commentaire sans contenu sur 1 ligne . . . . .	325
4.2.14	commentaire sans contenu sur plusieurs lignes . . . . .	327
<b>5</b>	<b>comparaison</b>	<b>329</b>
5.1	KO . . . . .	329
5.1.1	comparaison avec oubli d'operande gauche . . . . .	329
5.1.2	comparaison avec oubli d'operator . . . . .	331
5.1.3	comparaison avec oubli d'operator droit . . . . .	333
5.2	OK . . . . .	335
5.2.1	comparaison simple d'entiers avec <"<"> . . . . .	335
5.2.2	comparaison simple d'entiers avec <">"> . . . . .	337
5.2.3	comparaison simple d'entiers avec <"<="> . . . . .	339
5.2.4	comparaison simple d'entiers avec <">="> . . . . .	341
5.2.5	comparaison simple d'entiers avec <==> . . . . .	343
5.2.6	comparaison simple d'entiers avec <"<>"> . . . . .	345
5.2.7	comparaison double d'entiers . . . . .	347
5.2.8	comparaison triple d'entiers . . . . .	349
5.2.9	comparaison simple de chaines avec <"<"> . . . . .	351
5.2.10	comparaison simple de chaines avec <">"> . . . . .	353
5.2.11	comparaison simple de chaines avec <"<="> . . . . .	355
5.2.12	comparaison simple de chaines avec <">="> . . . . .	357
5.2.13	comparaison simple de chaines avec <==> . . . . .	359
5.2.14	comparaison simple de chaines avec <"<>"> . . . . .	361
5.2.15	comparaison double de chaines . . . . .	363
5.2.16	comparaison triple de chaines . . . . .	365

<b>6 espace</b>	<b>367</b>
6.1 KO . . . . .	367
6.2 OK . . . . .	367
6.2.1 1 espace entre <var> et <i> . . . . .	367
6.2.2 1 tabulation entre <var> et <i> . . . . .	369
6.2.3 1 saut de ligne entre <var> et <i> . . . . .	371
6.2.4 2 espaces entre <var> et <i> . . . . .	373
6.2.5 2 tabulations entre <var> et <i> . . . . .	375
6.2.6 2 sauts de ligne entre <var> et <i> . . . . .	377
6.2.7 3 espaces entre <var> et <i> . . . . .	379
6.2.8 3 tabulations entre <var> et <i> . . . . .	381
6.2.9 3 sauts de ligne entre <var> et <i> . . . . .	383
<b>7 for</b>	<b>385</b>
7.1 KO . . . . .	385
7.1.1 <for> avec affectation de chaine pour le compteur . . . . .	385
7.1.2 <for> avec affectation de chaine pour la borne superieure de l'iteration . . . . .	387
7.1.3 <for> avec oubli de l'identifiant du compteur . . . . .	389
7.1.4 <for> avec oubli du <for> . . . . .	391
7.1.5 <for> "express" . . . . .	393
7.1.6 <for> avec affectation de chaines pour le compteur et la borne maximale de l'iteration . . . . .	395
7.1.7 <for> avec oubli du <do> . . . . .	397
7.1.8 <for> avec oubli de la borne maximale de l'iteration . . . . .	399
7.1.9 <for> avec oubli du <to> . . . . .	401
7.1.10 <for> avec oubli de la borne inferieure de l'iteration . . . . .	403
7.1.11 <for> avec oubli du <=> . . . . .	405
7.1.12 <for> avec oubli du < :> . . . . .	407
7.1.13 <for> avec oubli du < :=> . . . . .	409
7.2 OK . . . . .	411
7.2.1 <for> simple croissant . . . . .	411
7.2.2 <for> simple decroissant . . . . .	413
7.2.3 <for> sans instruction . . . . .	415
7.2.4 <for> avec ligne vide . . . . .	417
7.2.5 double-imbrication de <for> . . . . .	419
7.2.6 triple-imbrication de <for> . . . . .	421
7.2.7 <for> avec reutilisation de compteur . . . . .	423
7.2.8 <for> avec reutilisation de compteurs . . . . .	425
<b>8 function</b>	<b>427</b>
8.1 KO . . . . .	427
8.1.1 fonction identifiee par caractere special . . . . .	427
8.1.2 oubli de <function> . . . . .	429
8.1.3 fonction sans identifiant . . . . .	431
8.1.4 oubli de <(> . . . . .	433
8.1.5 oubli d'identifiant de parametre . . . . .	435
8.1.6 oubli de < :> pour parametre . . . . .	437
8.1.7 oubli de type de parametre . . . . .	439
8.1.8 oubli de <)> . . . . .	441
8.1.9 oubli de < :> pour type de fonction . . . . .	443
8.1.10 < :> pour type de fonction present mais pas de type . . . . .	445
8.1.11 oubli de <=> . . . . .	447
8.1.12 parametres mal separees . . . . .	449
8.1.13 fonction sans instruction . . . . .	451
8.1.14 <function> mal ecrit . . . . .	453
8.1.15 fonction avec identifiant seulement en parametre . . . . .	455
8.1.16 fonction avec entier directement en parametre . . . . .	457
8.1.17 fonction avec chaine directement en parametre . . . . .	459

8.2	OK . . . . .	461
8.2.1	fonction definissant un entier . . . . .	461
8.2.2	fonction simple, avec declaration du type de retour <int> et instruction de retour . . . . .	463
8.2.3	fonction simple, avec instruction simple et instruction retournant un entier . . . . .	465
8.2.4	fonction avec 1 parametre entier . . . . .	467
8.2.5	fonction avec 2 parametres entiers . . . . .	469
8.2.6	fonction avec 3 parametres entiers . . . . .	471
8.2.7	fonction avec 2 parametres entiers et 1 parametre de type <string> . . . . .	473
8.2.8	fonction definissant une chaine . . . . .	475
8.2.9	fonction simple, avec declaration du type de retour <string> et instruction de retour . . . . .	477
8.2.10	fonction simple, avec instruction simple et instruction retournant une chaine . . . . .	479
8.2.11	fonction avec 1 parametre de type <string> . . . . .	481
8.2.12	fonction avec 2 parametres de type <string> . . . . .	483
8.2.13	fonction avec 3 parametres de type <string> . . . . .	485
8.2.14	fonction avec 2 parametres de type <string> et 1 parametre entier . . . . .	487
8.2.15	double-imbrication de fonction . . . . .	489
8.2.16	triple-imbrication de fonction . . . . .	491
8.2.17	recursion finie . . . . .	493
8.2.18	recursion infinie . . . . .	495
<b>9</b>	<b>if</b> . . . . .	<b>497</b>
9.1	KO . . . . .	497
9.1.1	incrementation dans la condition <if then> . . . . .	497
9.1.2	affectation d'entier dans la condition <if then> . . . . .	499
9.1.3	affectation de chaine dans la condition <if then> . . . . .	501
9.1.4	<if then> avec oubli du <if> . . . . .	503
9.1.5	<if then> avec oubli du <then> . . . . .	505
9.1.6	<if then> avec oubli de la condition . . . . .	507
9.1.7	<if then else> avec oubli de la condition . . . . .	509
9.1.8	<if then else> sans instruction dans <else> . . . . .	511
9.1.9	<if then else> avec ligne vide dans <else> . . . . .	513
9.1.10	<if then> avec condition entiere . . . . .	515
9.2	OK . . . . .	517
9.2.1	<if then> sans instruction . . . . .	517
9.2.2	<if then> avec ligne vide . . . . .	519
9.2.3	<if then> avec simple condition . . . . .	521
9.2.4	<if then> avec double-condition . . . . .	523
9.2.5	<if then> avec triple-condition . . . . .	525
9.2.6	double-imbrication de <if then> . . . . .	527
9.2.7	triple-imbrication de <if then> . . . . .	529
9.2.8	double-imbrication de <if then else> . . . . .	531
9.2.9	triple-imbrication de <if then else> . . . . .	533
<b>10</b>	<b>let</b> . . . . .	<b>535</b>
10.1	KO . . . . .	535
10.1.1	<let> sans declaration . . . . .	535
10.1.2	<let> sans instruction . . . . .	537
10.1.3	<let> avec inversion de declaration et instruction . . . . .	539
10.1.4	<let> avec declaration vide . . . . .	541
10.1.5	<let> avec instruction vide . . . . .	543
10.2	OK . . . . .	545
10.2.1	<let> avec 1 declaration de fonction et 1 appel de fonction . . . . .	545
10.2.2	<let> avec 2 declarations de fonction et 2 appels de fonction . . . . .	547
10.2.3	<let> avec 3 declarations de fonction et 3 appels de fonction . . . . .	549
10.2.4	<let> avec 1 declaration de variable et 1 instruction . . . . .	551
10.2.5	<let> avec 2 declarations de variable et 2 instructions . . . . .	553
10.2.6	<let> avec 3 declarations de variable et 3 instructions . . . . .	555

10.2.7 <let> avec 3 declarations de variable et fonction . . . . .	557
10.2.8 <let> avec double-imbrication . . . . .	559
10.2.9 <let> avec triple-imbrication . . . . .	561
<b>11 or</b>	<b>563</b>
11.1 KO . . . . .	563
11.1.1 < > mal ecrit . . . . .	563
11.2 OK . . . . .	565
11.2.1 condition avec 1 < > . . . . .	565
11.2.2 condition avec 2 < > . . . . .	567
11.2.3 condition avec 3 < > . . . . .	569
11.2.4 condition avec 2 < > et 1 <&> . . . . .	571
11.2.5 < > avec des entiers . . . . .	573
11.2.6 < > avec des chaines . . . . .	575
<b>12 return</b>	<b>577</b>
12.1 KO . . . . .	577
12.1.1 <return> mal ecrit . . . . .	577
12.1.2 <return> sans valeur . . . . .	579
12.1.3 <return> avec oubli de parentheses . . . . .	581
12.1.4 2 <return> . . . . .	583
12.2 OK . . . . .	585
12.2.1 retour d'entier positif . . . . .	585
12.2.2 retour d'entier negatif . . . . .	587
12.2.3 retour de chaine . . . . .	589
12.2.4 retour de <nil> . . . . .	591
12.2.5 retour de variable . . . . .	593
12.2.6 retour de fonction . . . . .	595
12.2.7 retour d'expression . . . . .	597
<b>13 var</b>	<b>599</b>
13.1 KO . . . . .	599
13.1.1 declaration sans affectation . . . . .	599
13.1.2 declaration d'entier avec type sans affectation . . . . .	601
13.1.3 declaration de chaine avec type sans affectation . . . . .	603
13.1.4 affectation d'entier sans declaration . . . . .	605
13.1.5 affectation de chaine sans declaration . . . . .	607
13.1.6 affectation de <nil> . . . . .	609
13.1.7 double-declaration avec valeurs égales . . . . .	611
13.1.8 double-declaration avec valeurs différentes . . . . .	613
13.1.9 <var> mal ecrit . . . . .	615
13.1.10 declaration de caractere special . . . . .	617
13.1.11 affectation de caractere special . . . . .	619
13.1.12 affectation d'opération logique sur entiers . . . . .	621
13.1.13 affectation d'opération logique sur chaines . . . . .	623
13.1.14 chiffre comme nom de variable . . . . .	625
13.1.15 <_> comme nom de variable . . . . .	627
13.2 OK . . . . .	629
13.2.1 declaration d'entier simple . . . . .	629
13.2.2 declarations d'entier avec reutilisation de 2 autres entiers . . . . .	631
13.2.3 declarations d'entier avec reutilisation de 3 autres entiers . . . . .	633
13.2.4 declaration simple d'entier avec type . . . . .	635
13.2.5 declarations d'entier avec types et reutilisation de 2 autres entiers . . . . .	637
13.2.6 declarations d'entier avec types et reutilisation de 3 autres entiers . . . . .	639
13.2.7 declaration de chaine simple . . . . .	641
13.2.8 declarations de chaine avec reutilisation de 2 autres chaines . . . . .	643
13.2.9 declarations de chaine avec reutilisation de 3 autres chaines . . . . .	645

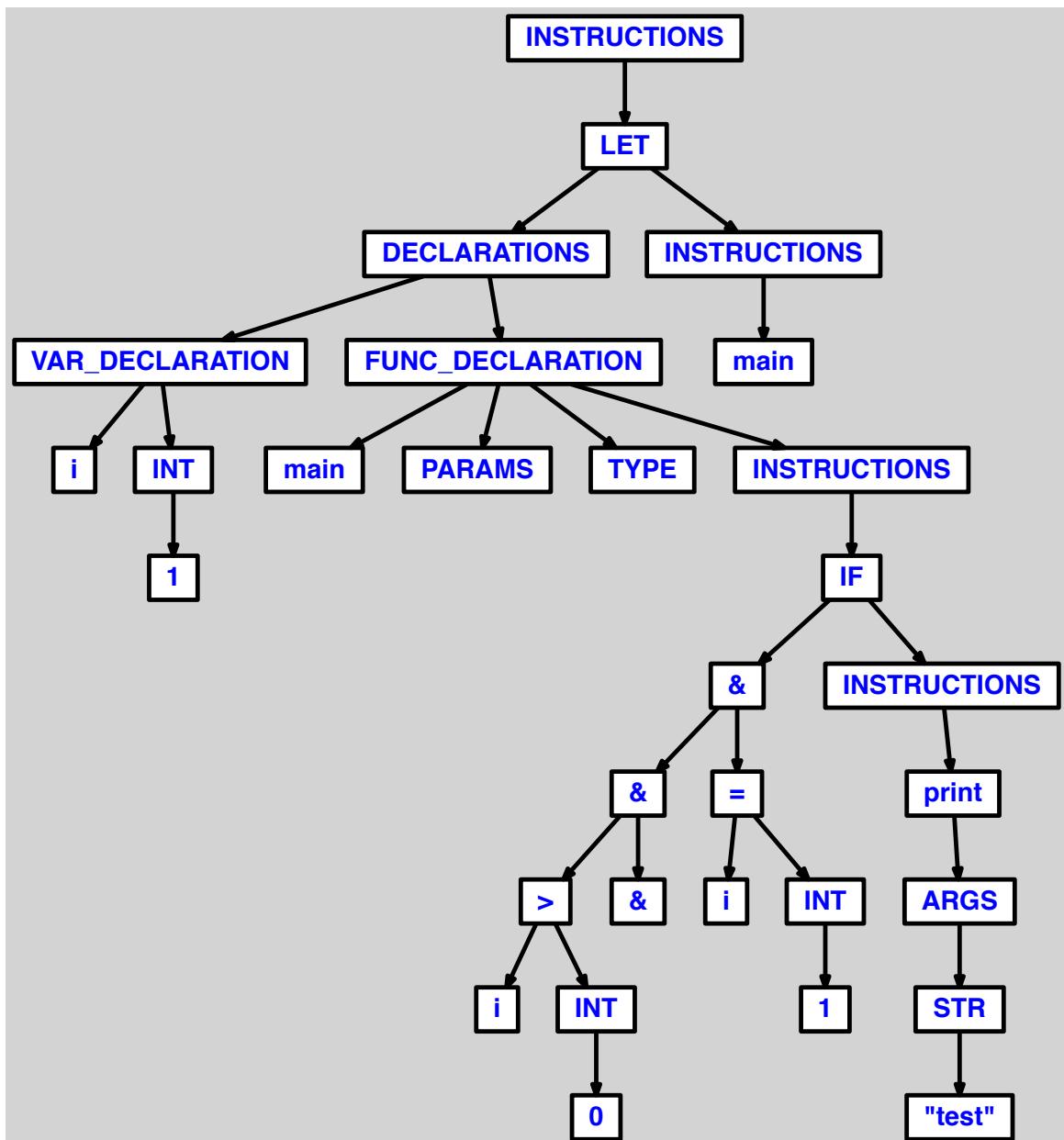
13.2.10 declaration de chaine avec type . . . . .	647
13.2.11 declarations de chaine avec types et reutilisation de 2 autres chaines . . . . .	649
13.2.12 declarations de chaine avec types et reutilisation de 3 autres chaines . . . . .	651
13.2.13 lettre minuscule comme nom de variable . . . . .	653
13.2.14 lettre majuscule comme nom de variable . . . . .	655
13.2.15 lettre minuscule et chiffre dans nom de variable . . . . .	657
13.2.16 lettre majuscule et chiffre dans nom de variable . . . . .	659
13.2.17 lettre minuscule et <_> dans nom de variable . . . . .	661
13.2.18 lettre majuscule et <_> dans nom de variable . . . . .	663
13.2.19 lettre minuscule, chiffre et <_> dans nom de variable . . . . .	665
13.2.20 lettre majuscule, chiffre et <_> dans nom de variable . . . . .	667
<b>14 while</b>	<b>669</b>
14.1 KO . . . . .	669
14.1.1 <while> avec affectation d'entier . . . . .	669
14.1.2 <while> avec affectation de chaine . . . . .	671
14.1.3 <while> avec oubli du compteur . . . . .	673
14.1.4 <while> avec oubli du <while> . . . . .	675
14.1.5 <while> avec oubli du <do> . . . . .	677
14.1.6 <while> avec oubli de la condition . . . . .	679
14.2 OK . . . . .	681
14.2.1 <while> avec iteration croissante . . . . .	681
14.2.2 <while> avec iteration decroissante . . . . .	683
14.2.3 <while> avec double-condition . . . . .	685
14.2.4 <while> avec triple-condition . . . . .	687
14.2.5 <while> sans instruction . . . . .	689
14.2.6 <while> avec ligne vide . . . . .	691
14.2.7 <while> avec condition entiere . . . . .	693
14.2.8 <while> avec double-imbrication . . . . .	695
14.2.9 <while> avec triple-imbrication . . . . .	697
14.2.10 <while> avec reutilisation de compteur . . . . .	699
14.2.11 <while> avec reutilisation de compteurs . . . . .	701
14.2.12 <while> avec iteration sur une chaine . . . . .	703

# 1 and

## 1.1 KO

### 1.1.1 <&> mal écrit

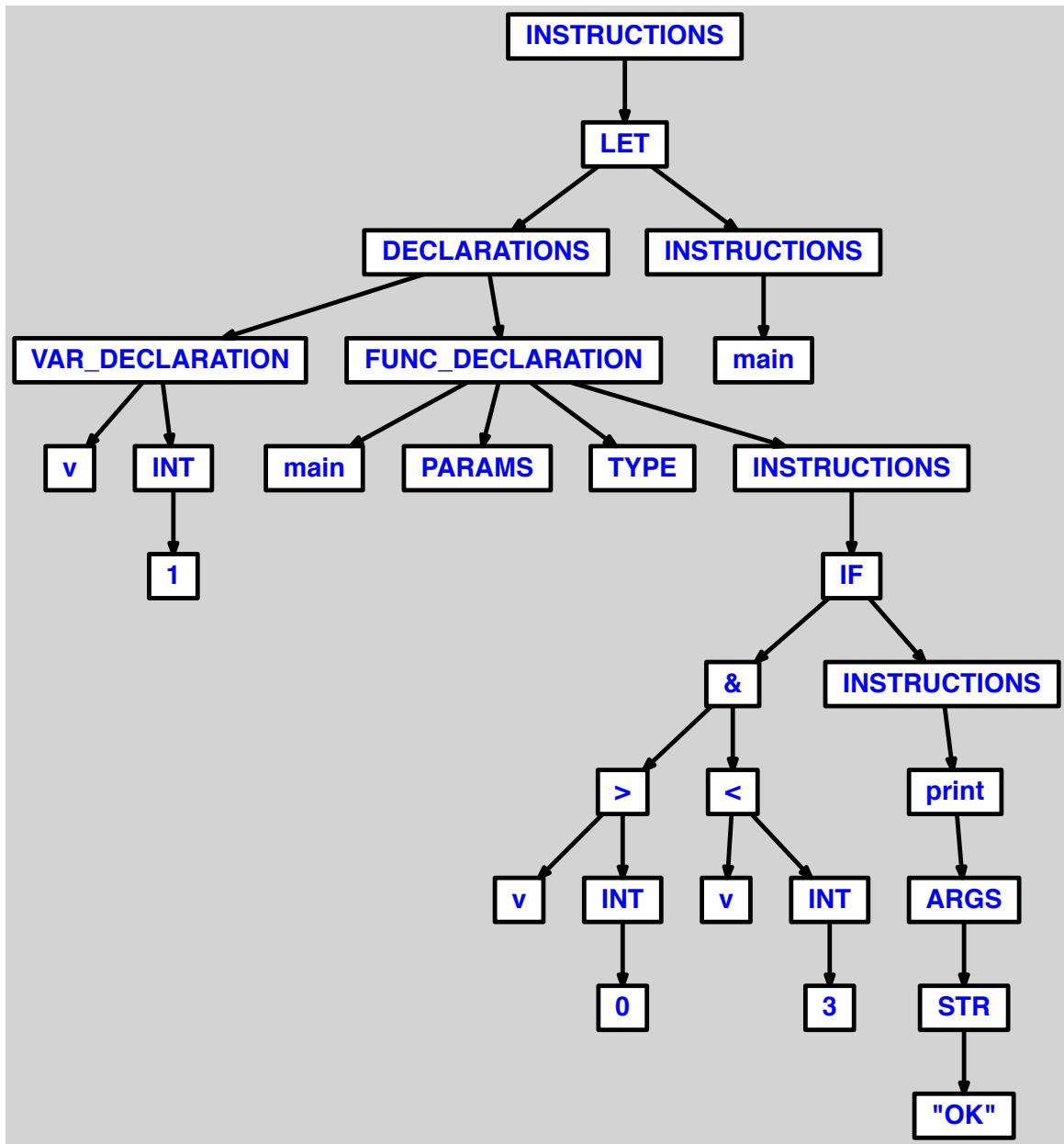
```
1 let
2   var i := 1
3
4   function main() =
5     if i > 0 && i = 1 then print("test")
6 in main() end
```



## 1.2 OK

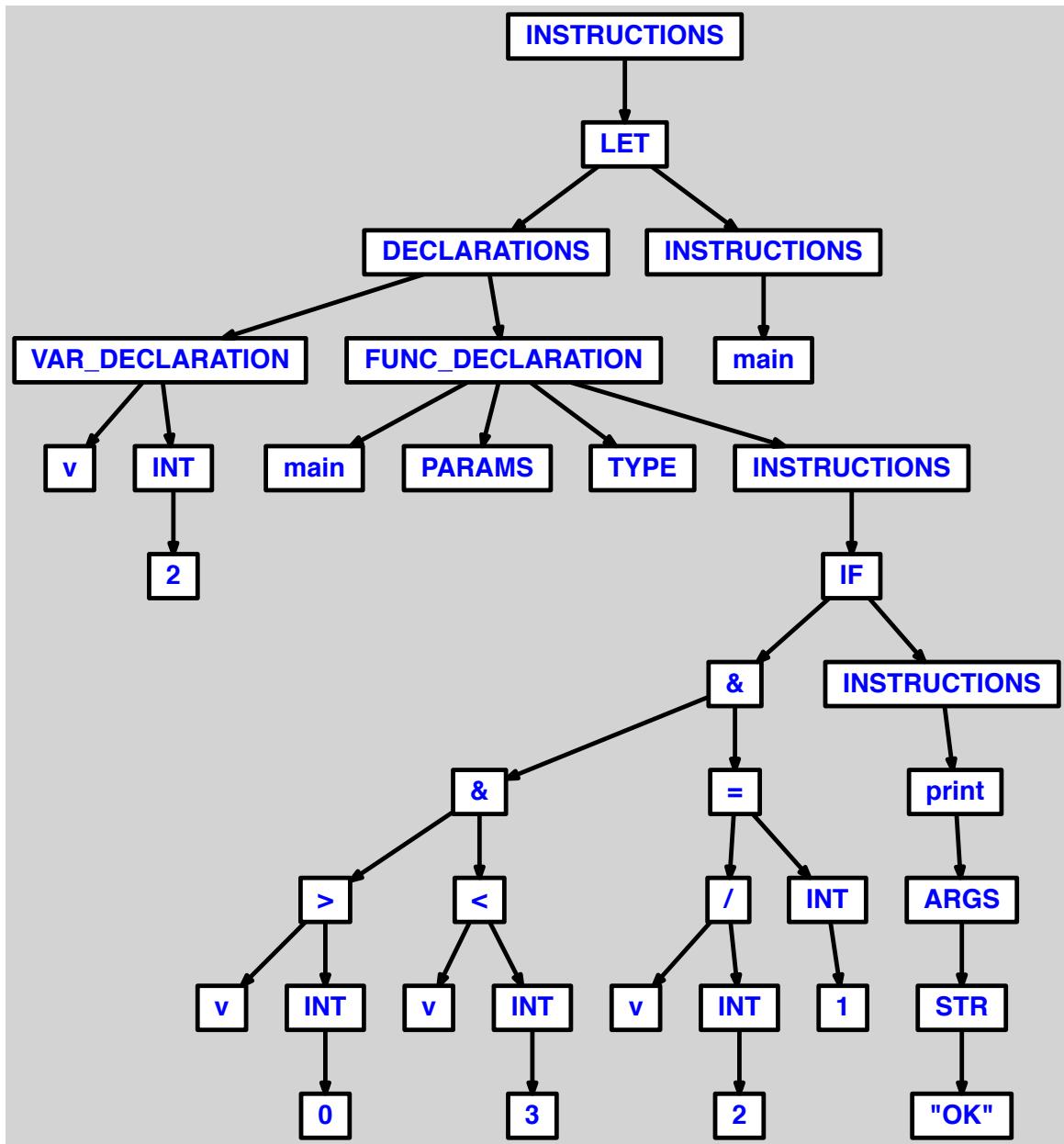
### 1.2.1 condition avec 1 <&>

```
1 let
2   var v := 1
3
4   function main() =
5     if v > 0 & v < 3 then print("OK")
6   in main() end
```



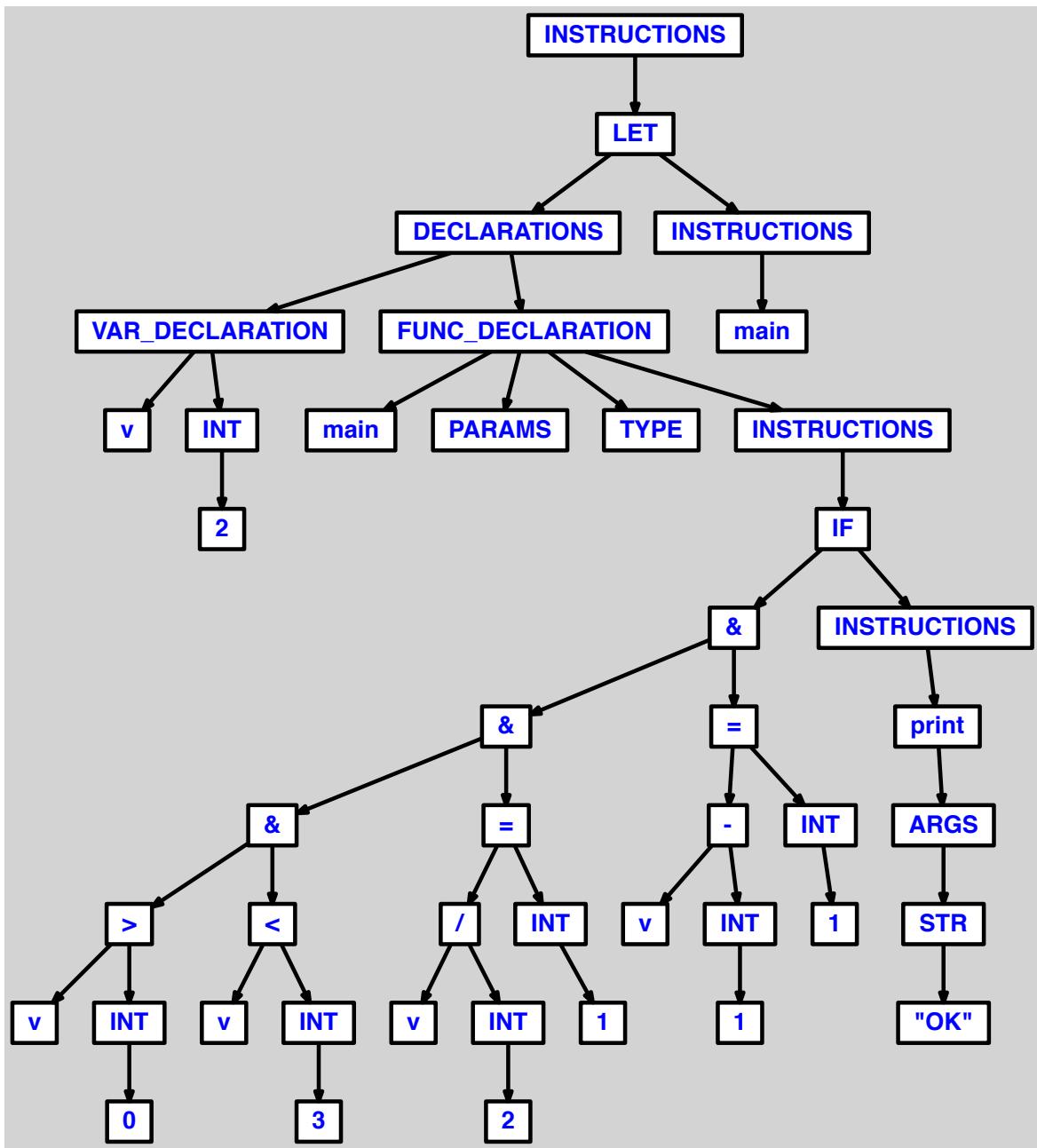
### 1.2.2 condition avec 2 <&>

```
1 let
2   var v := 2
3
4   function main() =
5     if v > 0 & v < 3 & v/2 = 1 then print("OK")
6 in main() end
```



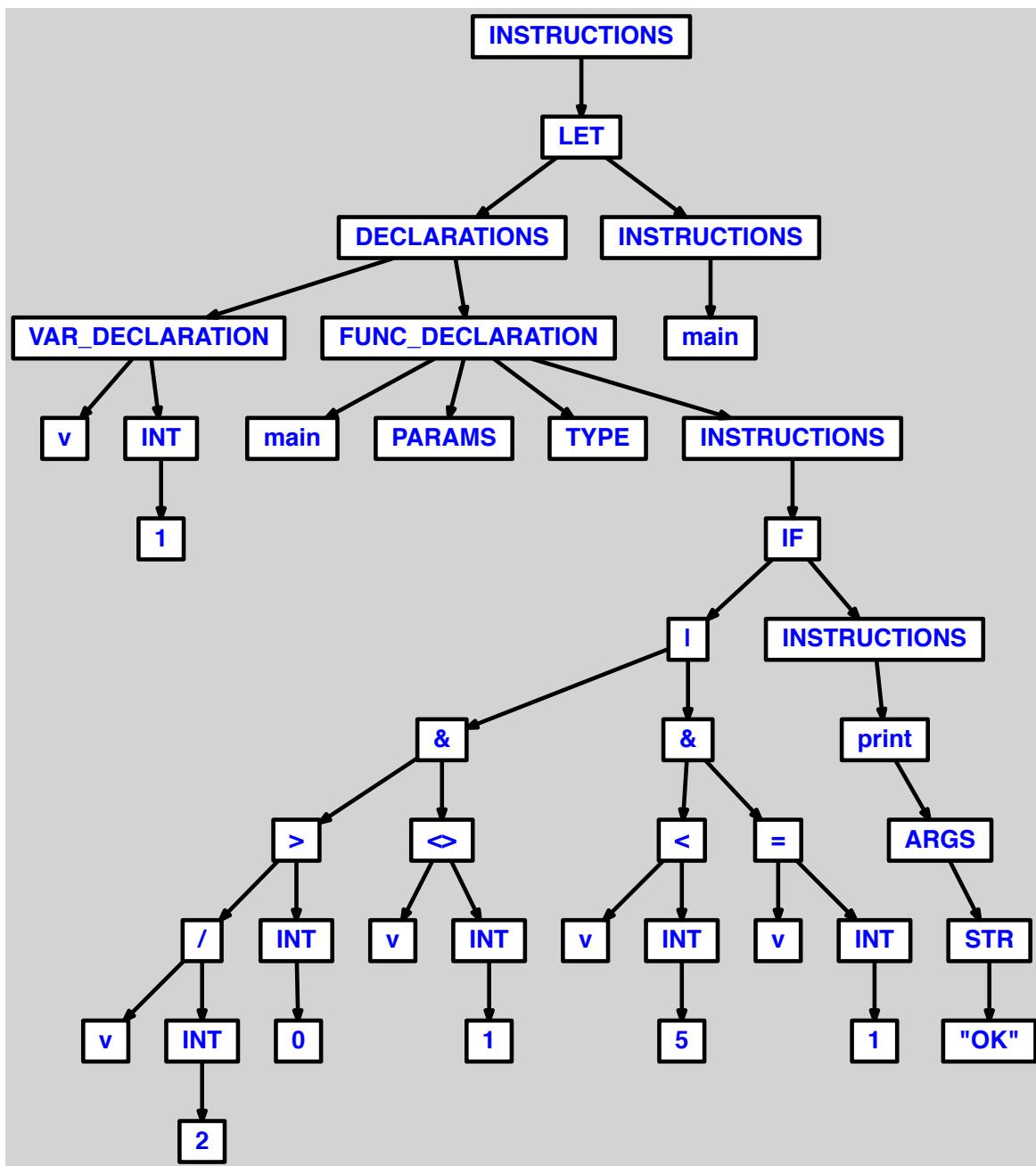
### 1.2.3 condition avec 3 <&>

```
1 let
2   var v := 2
3
4   function main() =
5     if v > 0 & v < 3 & v/2 = 1 & v-1 = 1 then print("OK")
6   in main() end
```



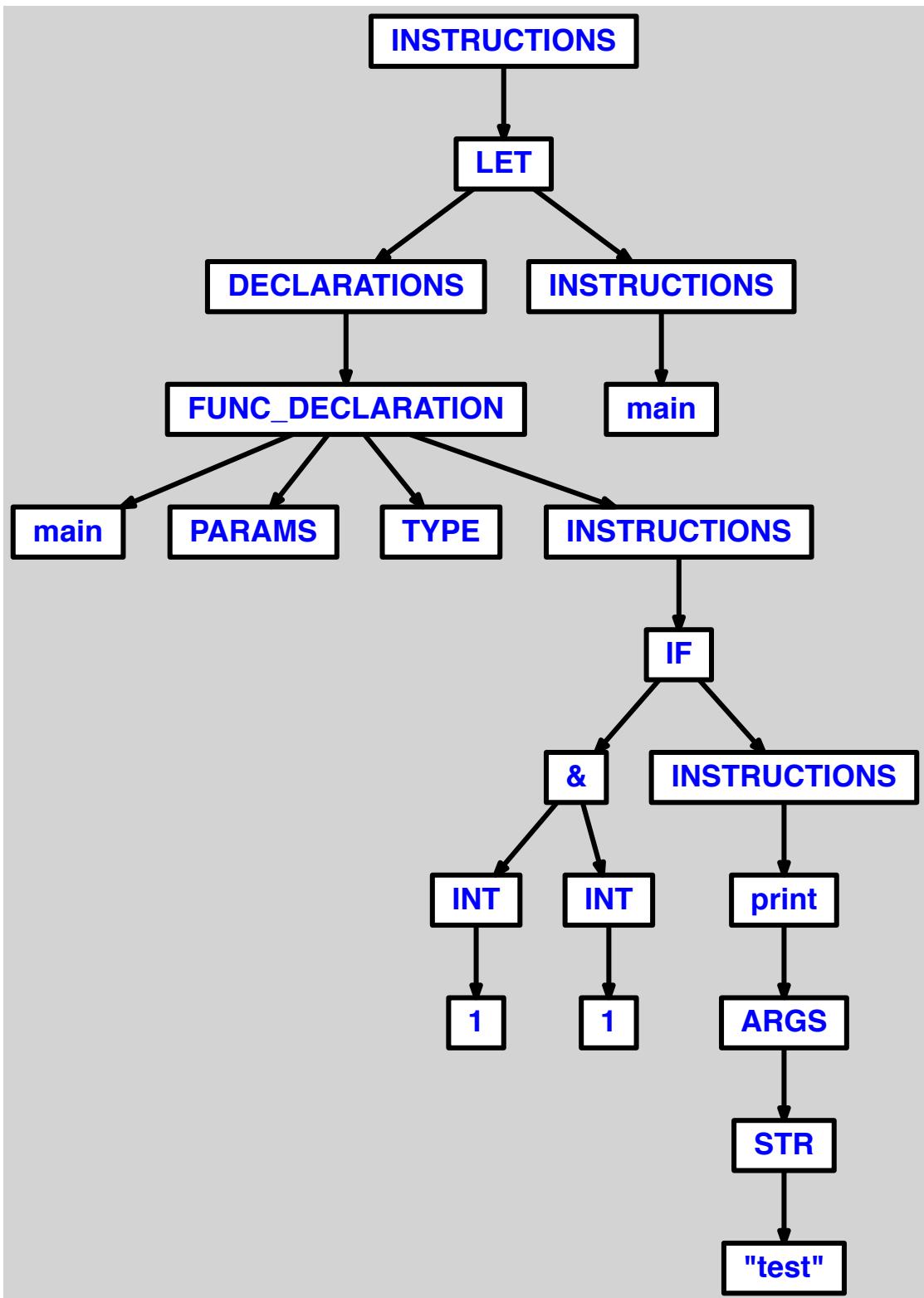
#### 1.2.4 condition avec 2 <&> et 1 <|>

```
1 let
2   var v := 1
3
4   function main() =
5     if v/2 > 0 & v <> 1 | v < 5 & v = 1 then print("OK")
6 in main() end
```



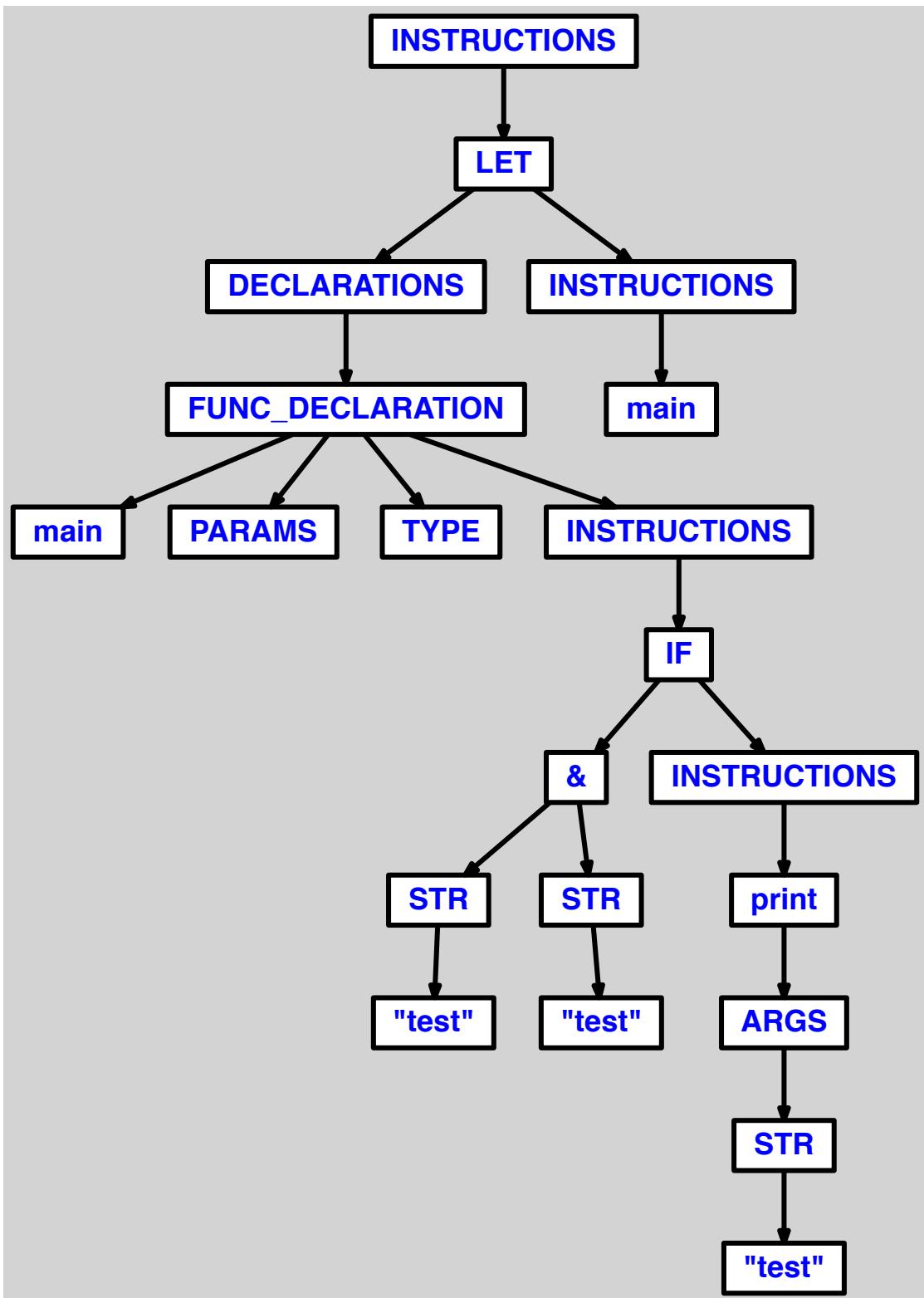
### 1.2.5 <&> avec des entiers

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



### 1.2.6 <&> avec des chaines

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

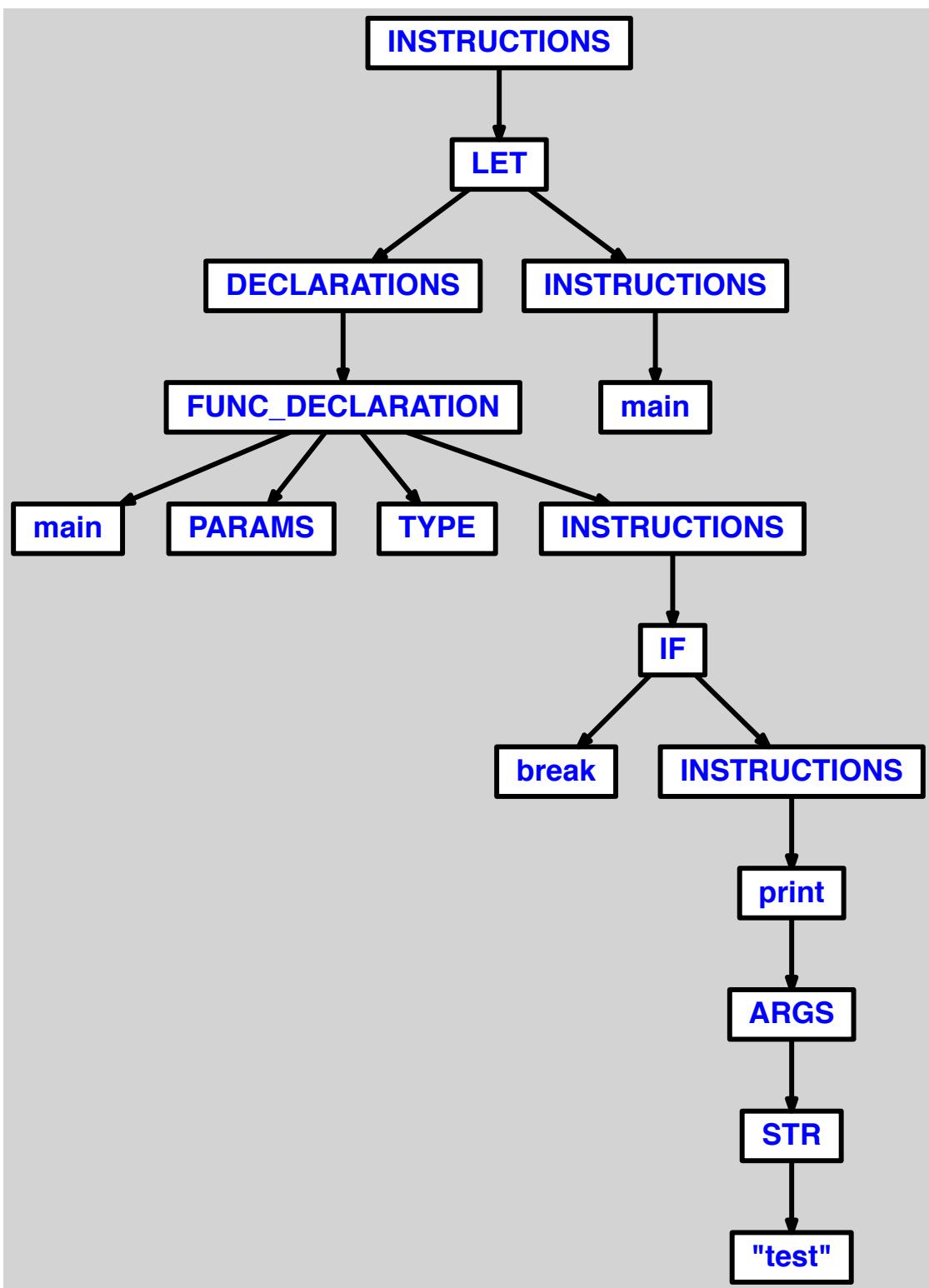


## 2 break

### 2.1 KO

#### 2.1.1 <break> comme condition dans <if>

```
1 let
2   function main() =
3     if break then
4       print("test")
5   in main() end
```



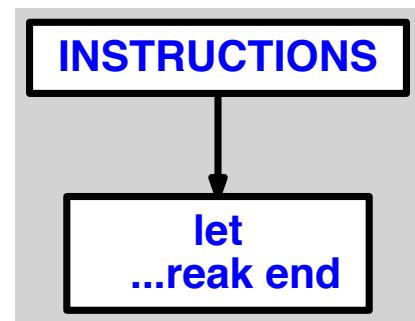
### 2.1.2 <break> comme condition dans <while>

```
1 let
2   function main() =
3     while break then
4       print("test")
5   in main() end
```

Pas d'AST, problème de syntaxe.

### 2.1.3 appel de <break>, avec corps de programme contenant <break>

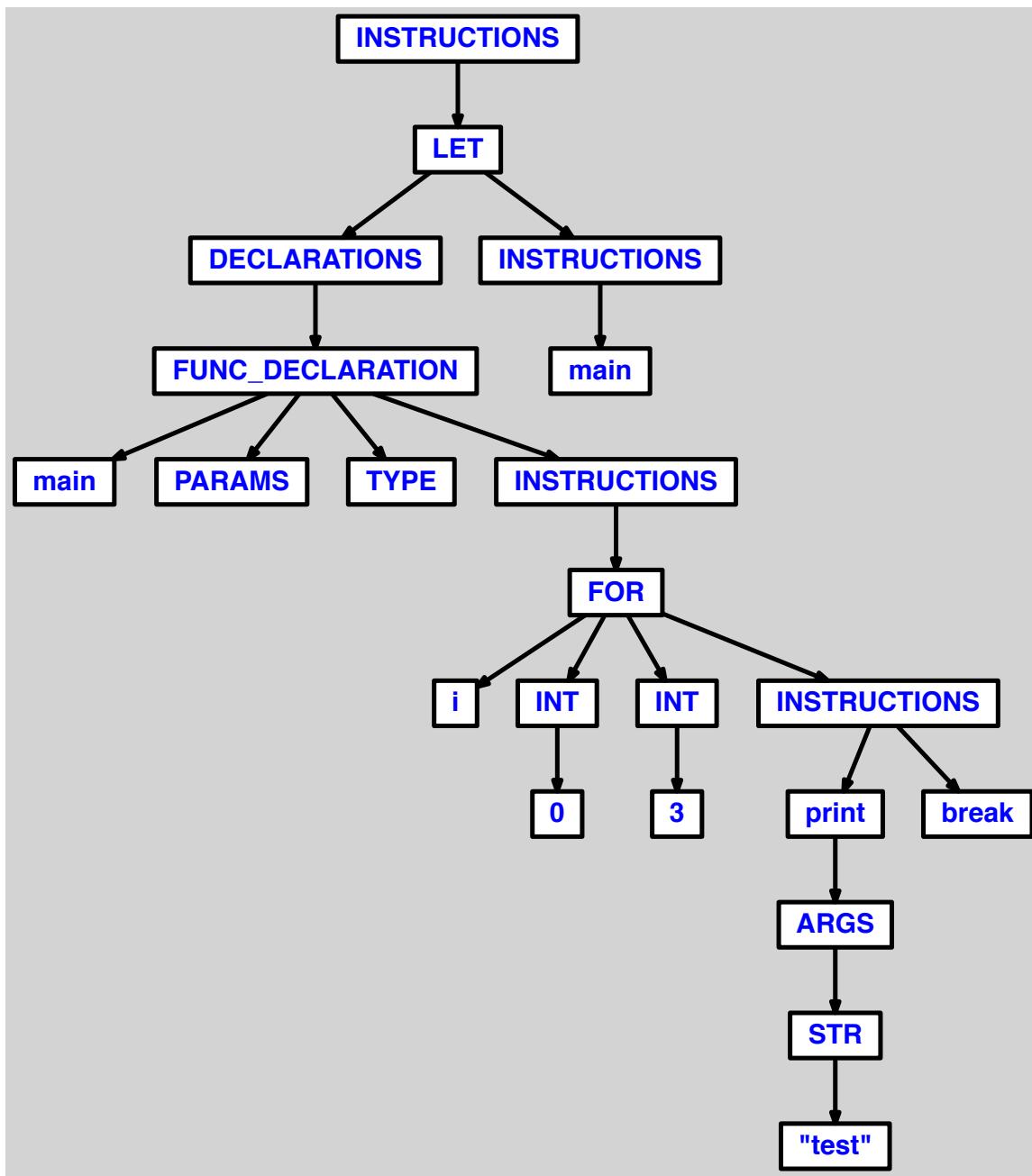
```
1 let
2   break
3 in break end
```



## 2.2 OK

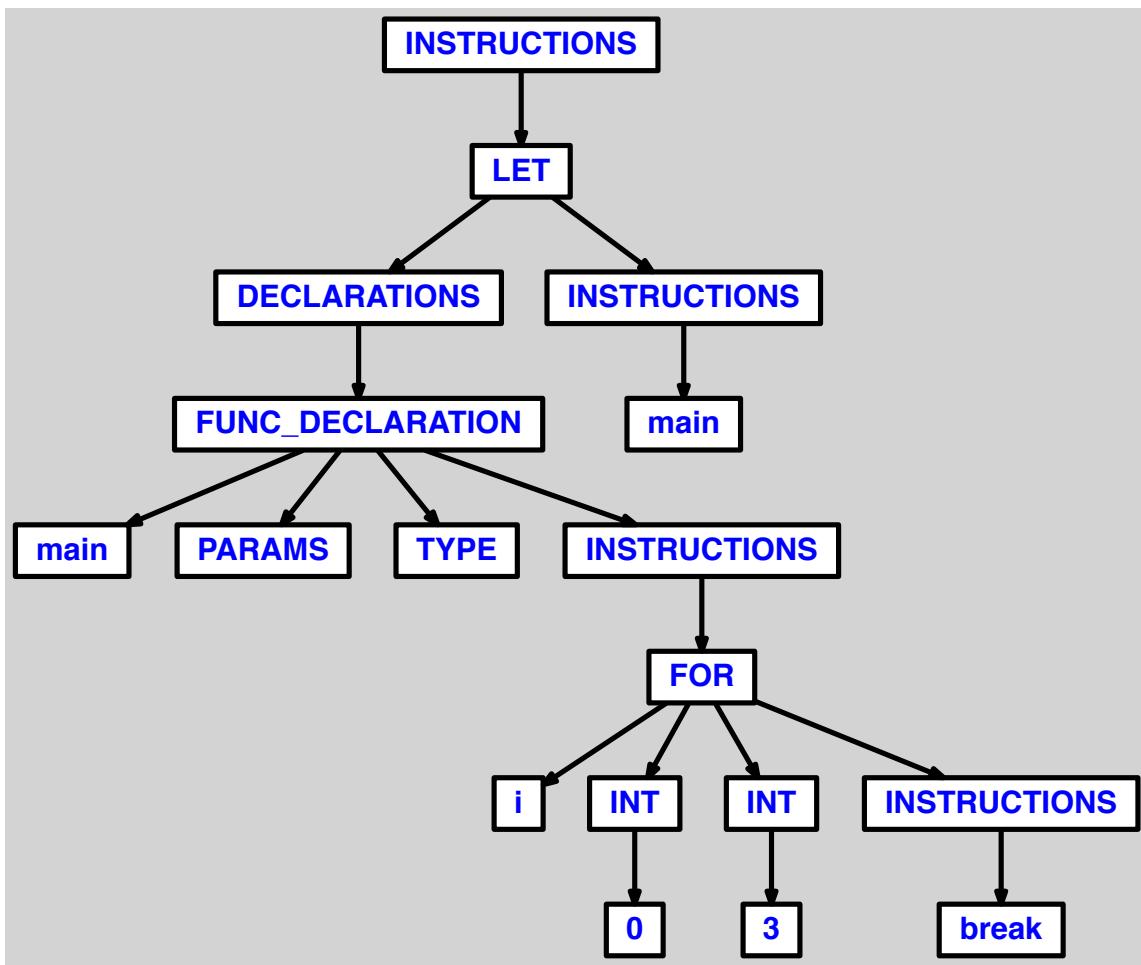
### 2.2.1 <break> dans un <for>, apres instruction

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



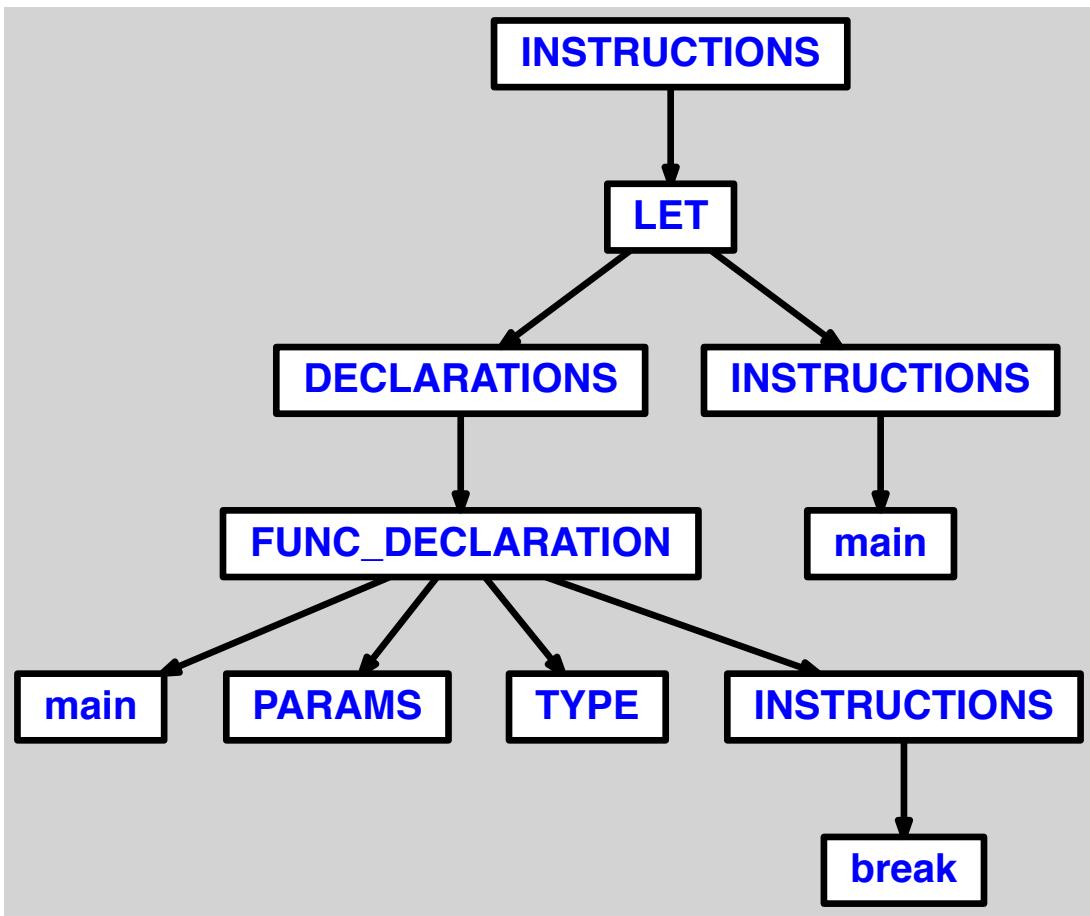
## 2.2.2 <break> dans un <for>, sans instruction avant

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



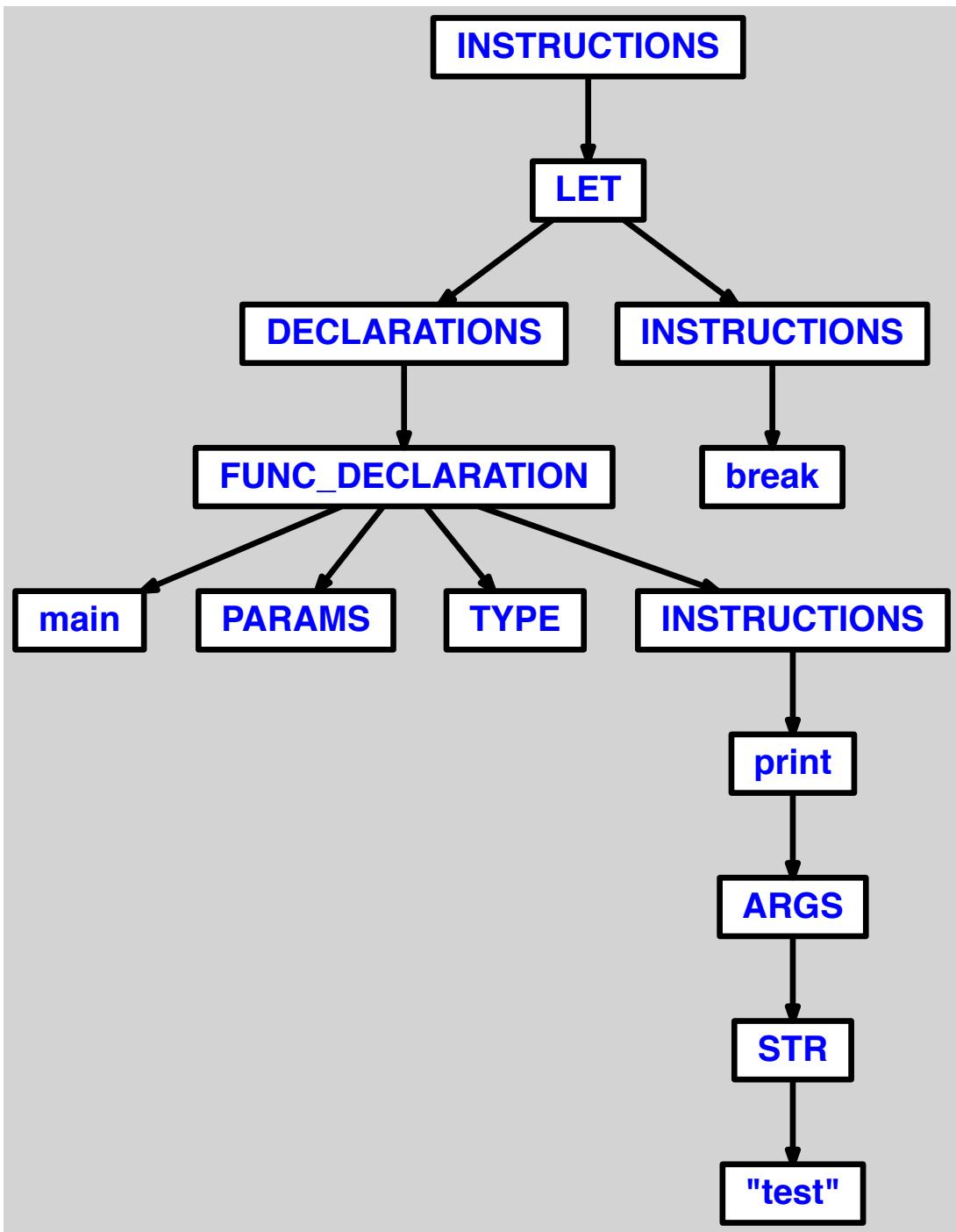
### 2.2.3 fonction executant seulement <break>

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



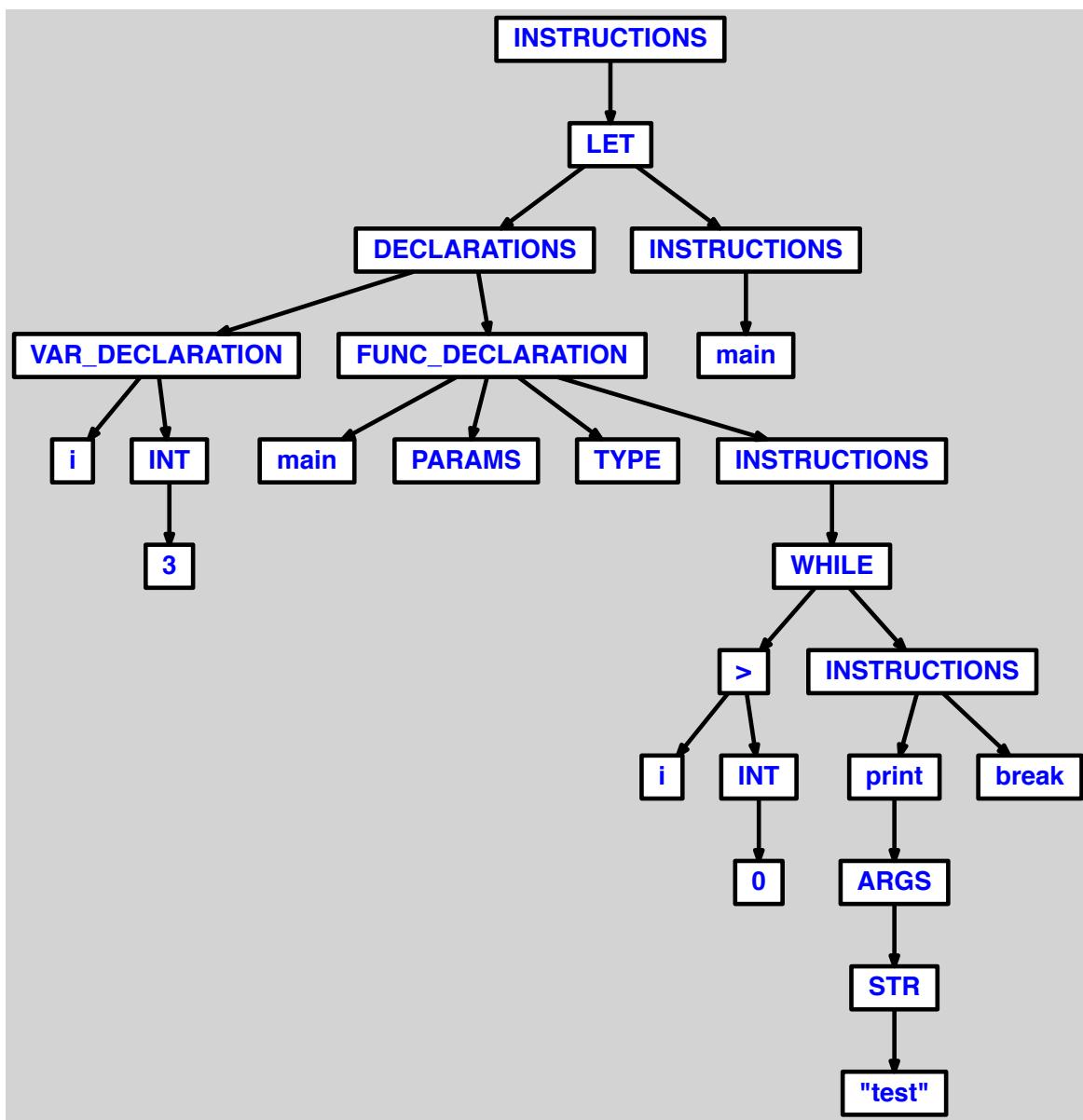
#### 2.2.4 appel de <break>, avec corps de programme ne contenant pas <break>

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



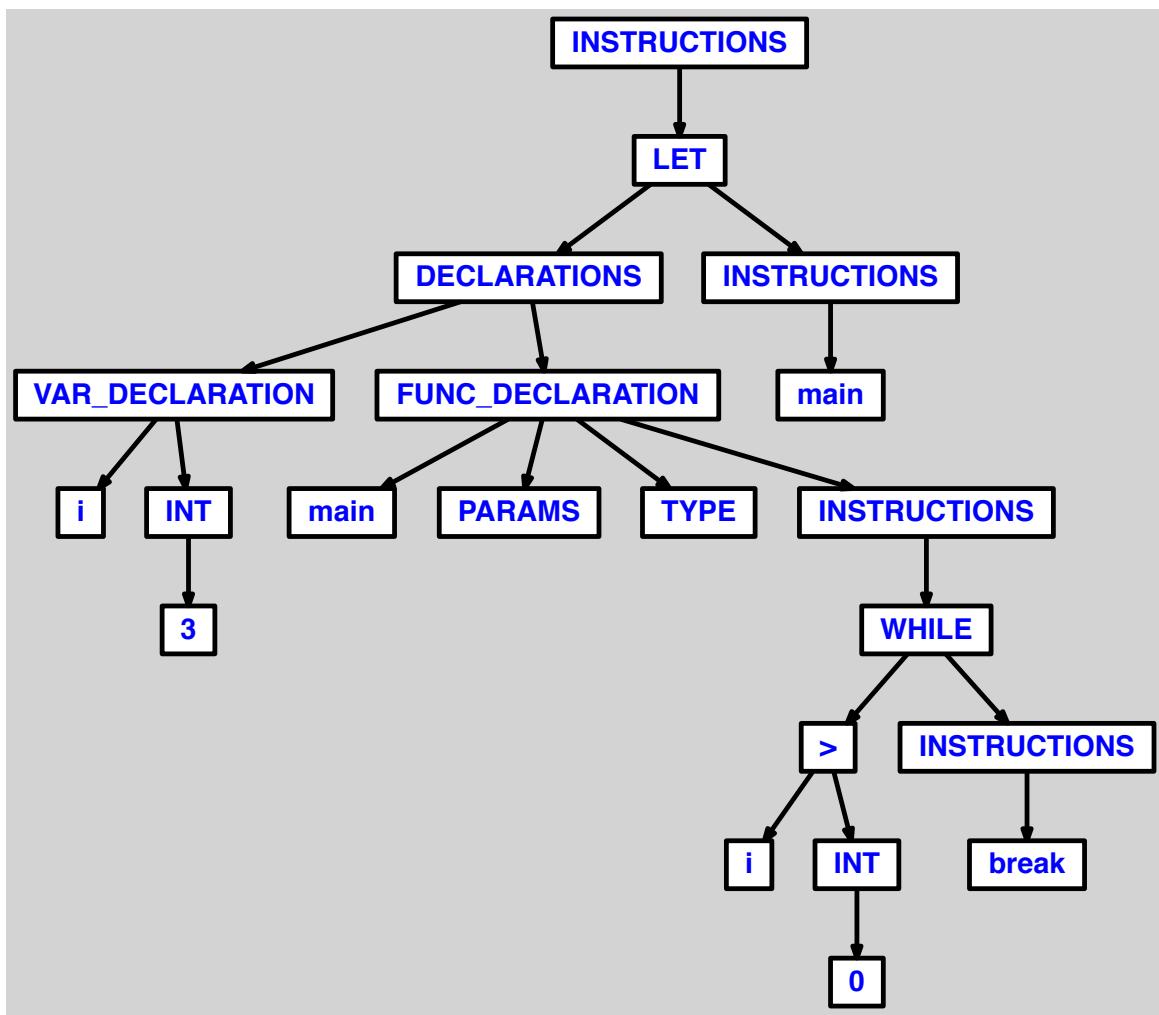
## 2.2.5 <break> dans un <while>, apres instruction

```
1 let
2   var i := 3
3
4   function main() =
5     while i > 0 do
6       (print("test"); break)
7   in main() end
```



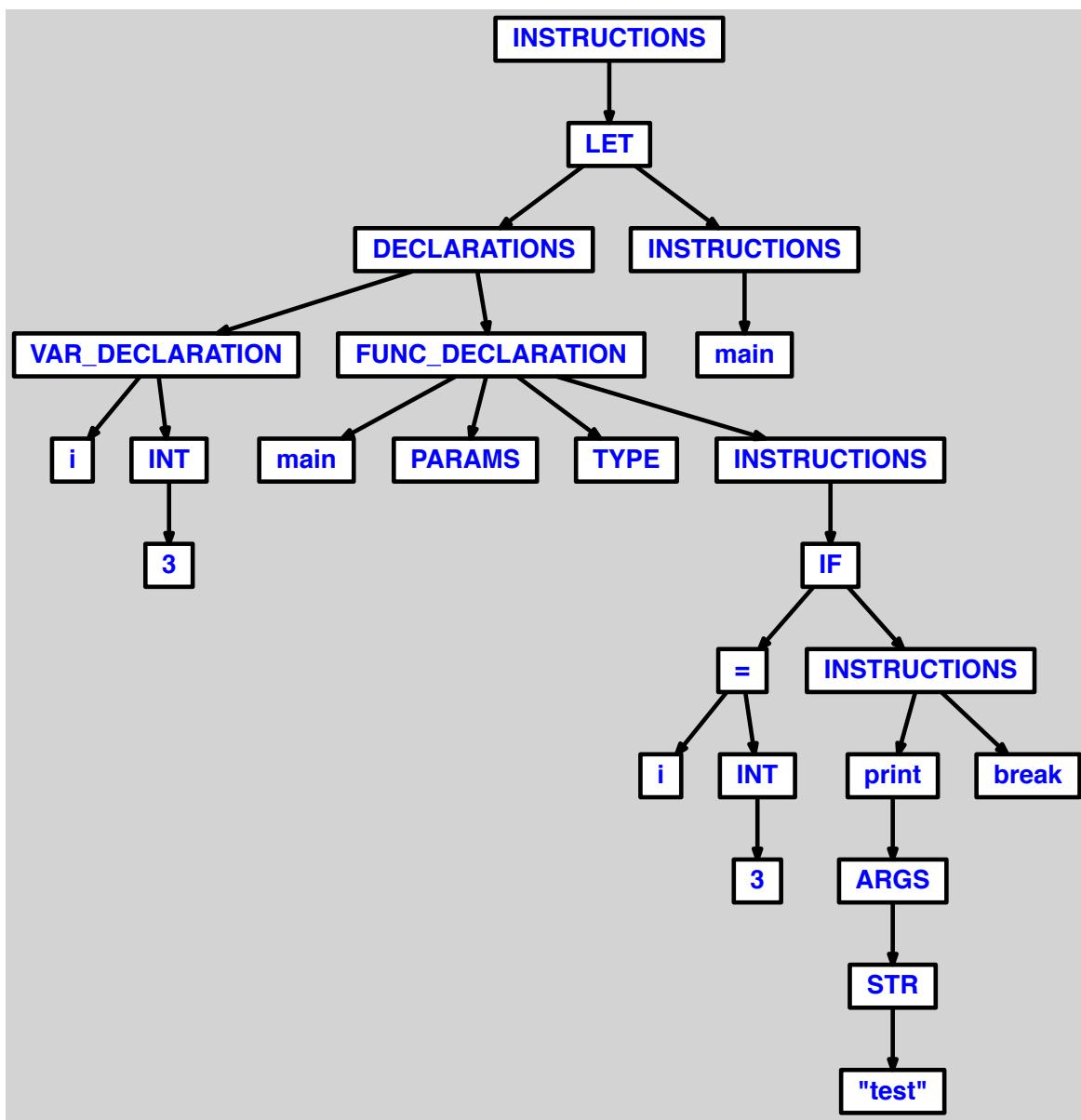
## 2.2.6 <break> dans un <while>, sans instruction avant

```
1 let
2   var i := 3
3
4   function main() =
5     while i > 0 do
6       break
7 in main() end
```



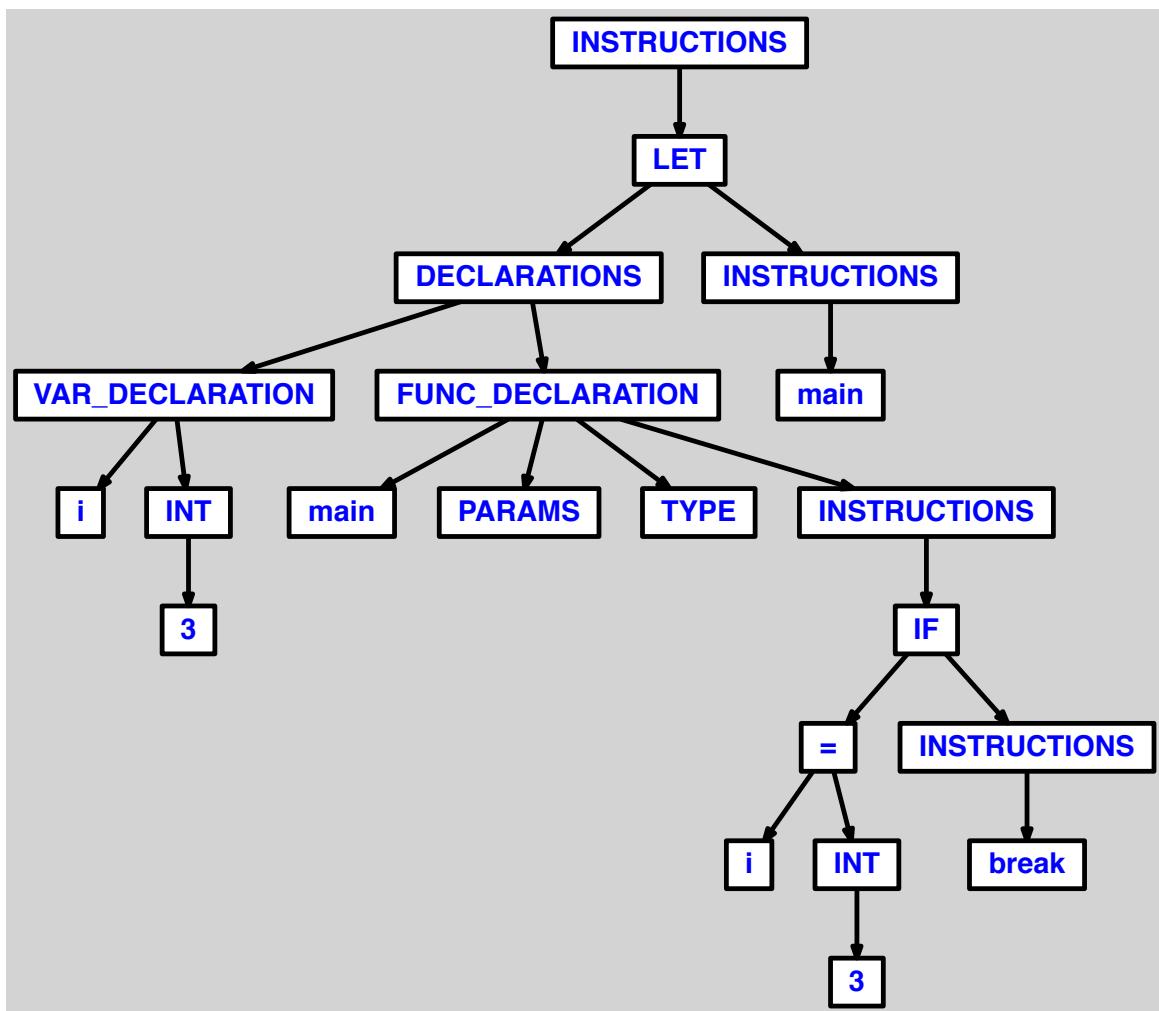
## 2.2.7 <break> dans un <if then>, apres instruction

```
1 let
2   var i := 3
3
4   function main() =
5     if i=3 then
6       (print("test"); break)
7 in main() end
```



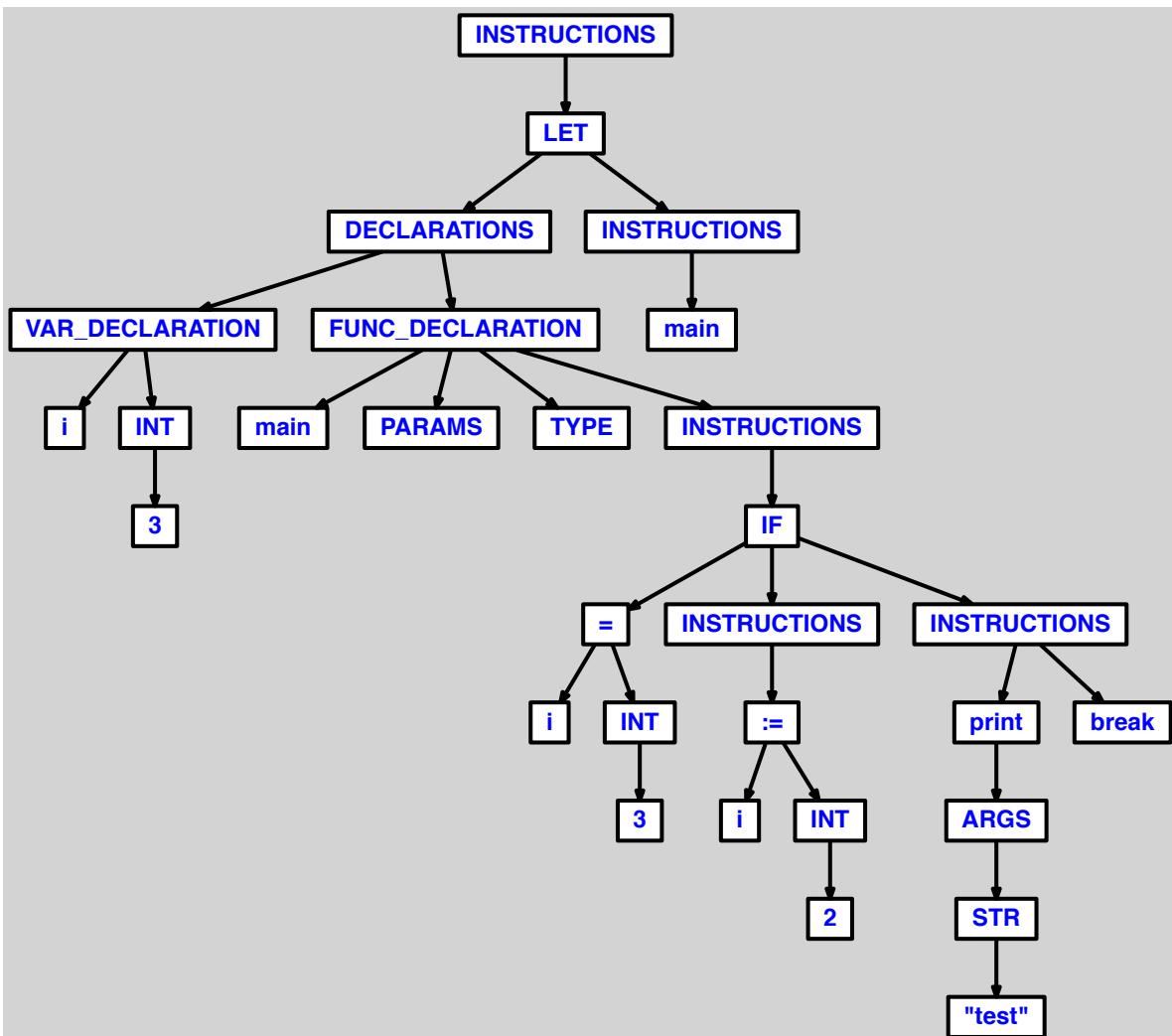
## 2.2.8 <break> dans un <if then>, sans instruction avant

```
1 let
2   var i := 3
3
4   function main() =
5     if i=3 then
6       break
7 in main() end
```



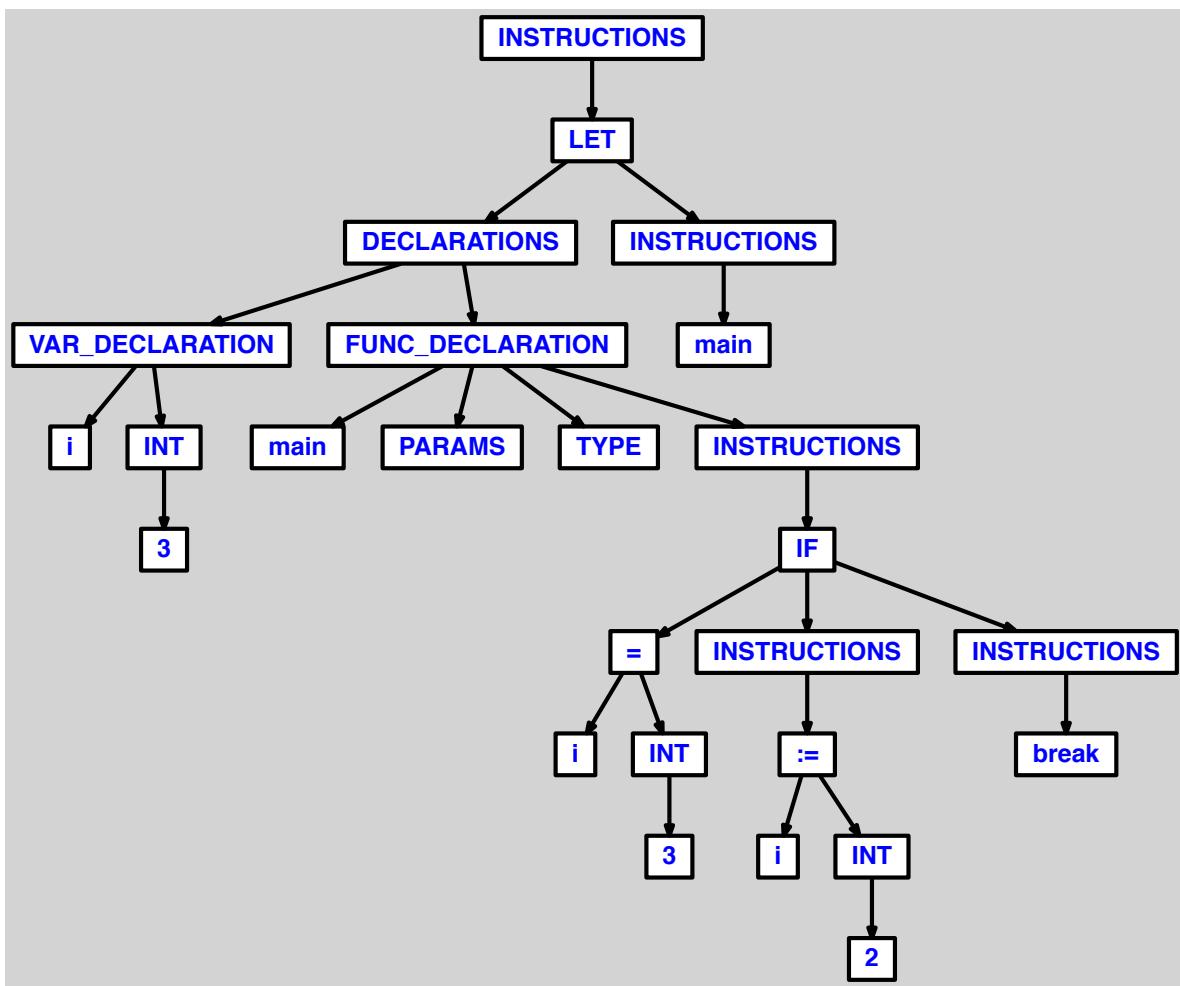
## 2.2.9 <break> dans un <if then else>, apres instruction

```
1 let
2   var i := 3
3
4   function main() =
5     if i=3 then
6       i := 2
7     else
8       (print("test"); break)
9 in main() end
```



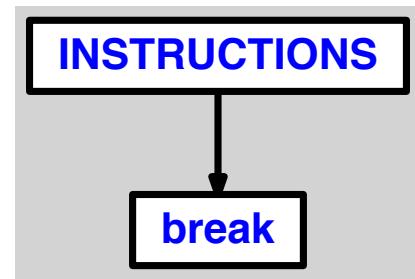
## 2.2.10 <break> dans un <if then else>, sans instruction avant

```
1 let
2   var i := 3
3
4   function main() =
5     if i=3 then
6       i := 2
7     else
8       break
9 in main() end
```



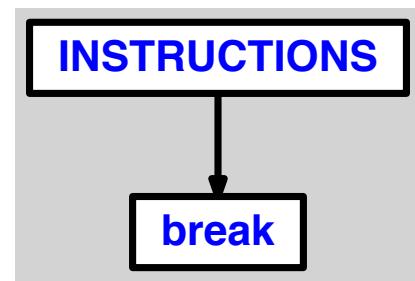
## 2.2.11 <break> seul

```
1 break
```



### 2.2.12 <break> seul, apres saut de ligne

```
1  
2 break
```

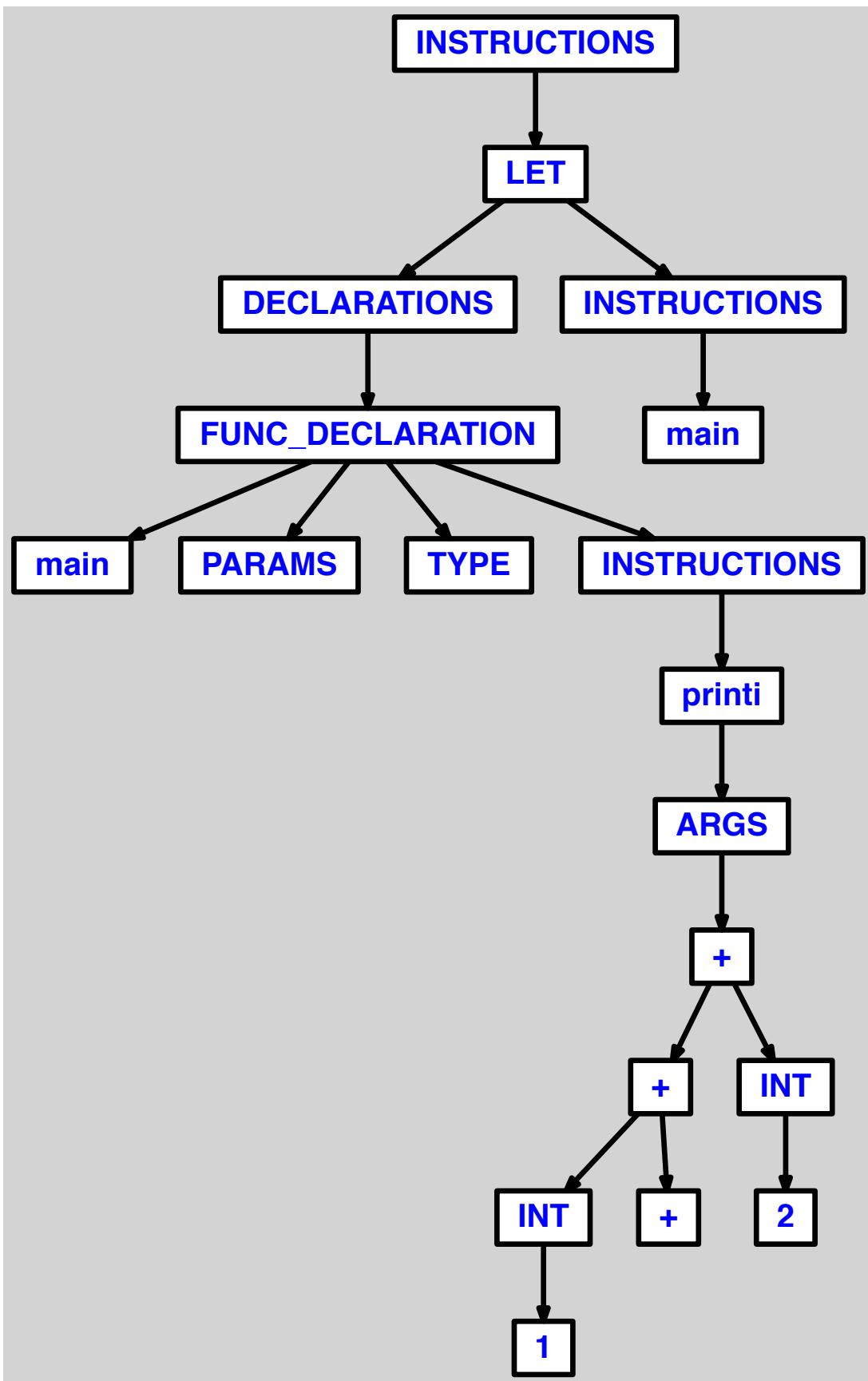


### 3 calcul

#### 3.1 KO

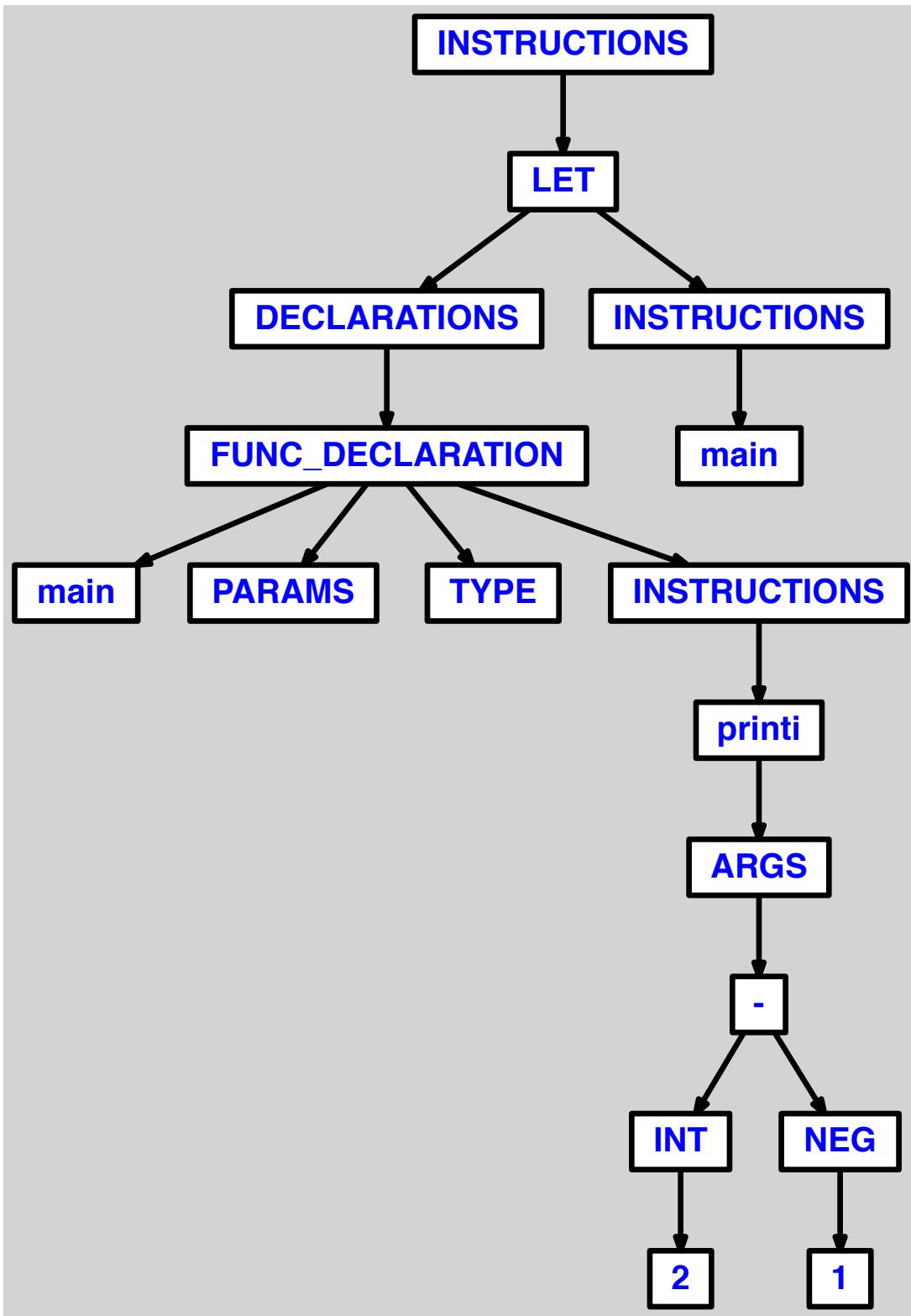
##### 3.1.1 addition avec operateur mal écrit

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



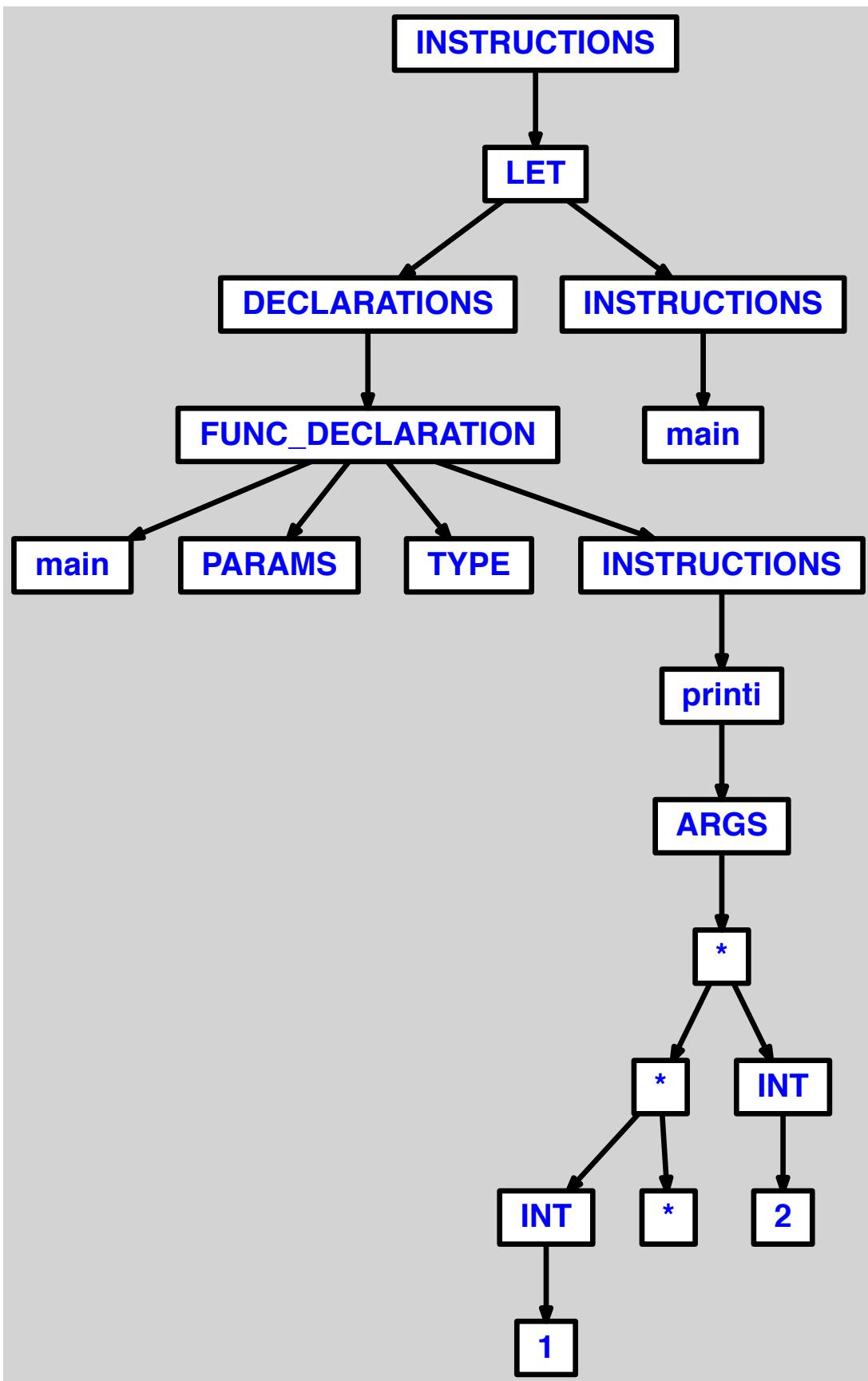
### 3.1.2 soustraction avec operateur mal écrit

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



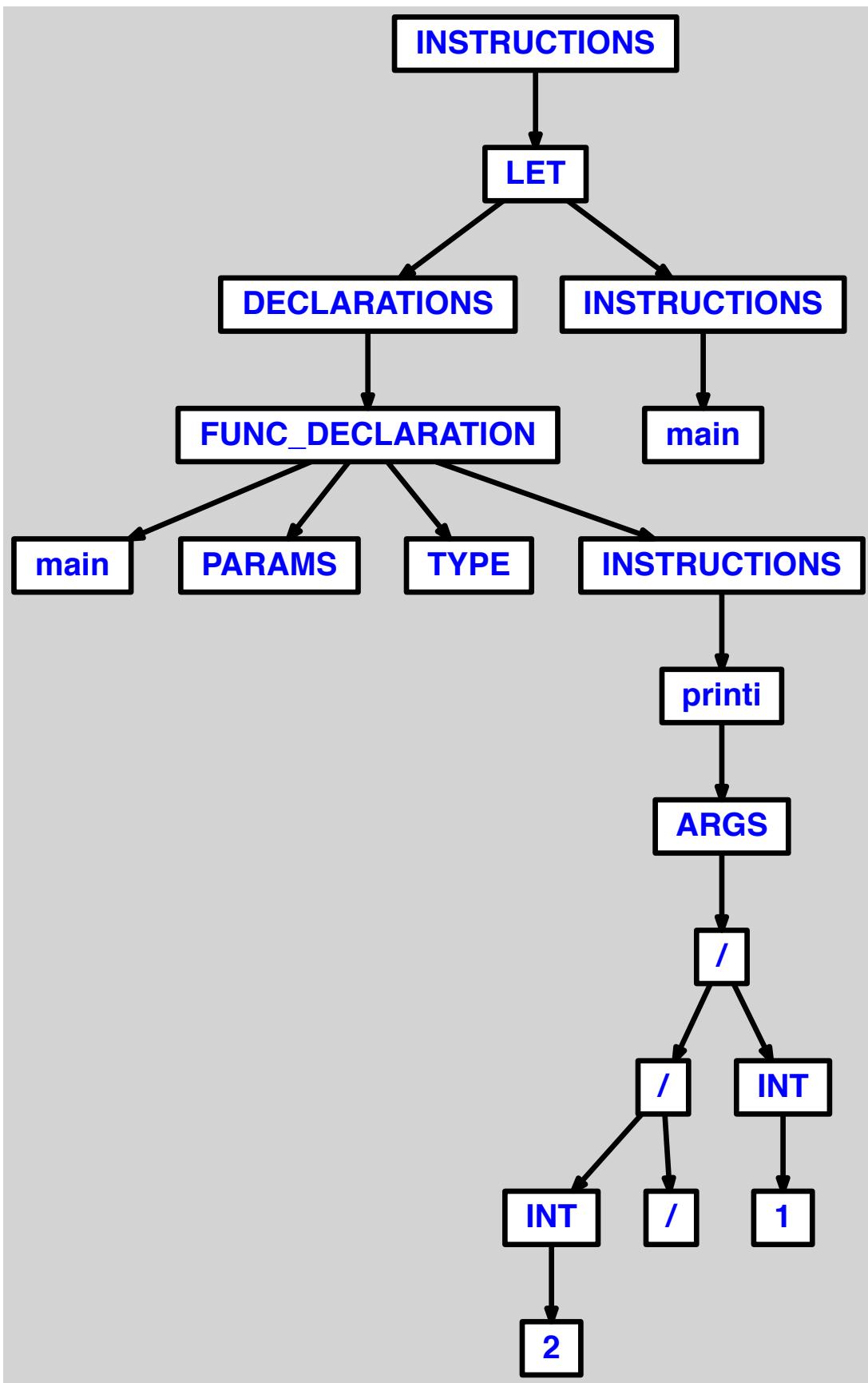
### 3.1.3 multiplication avec operateur mal ecrit

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



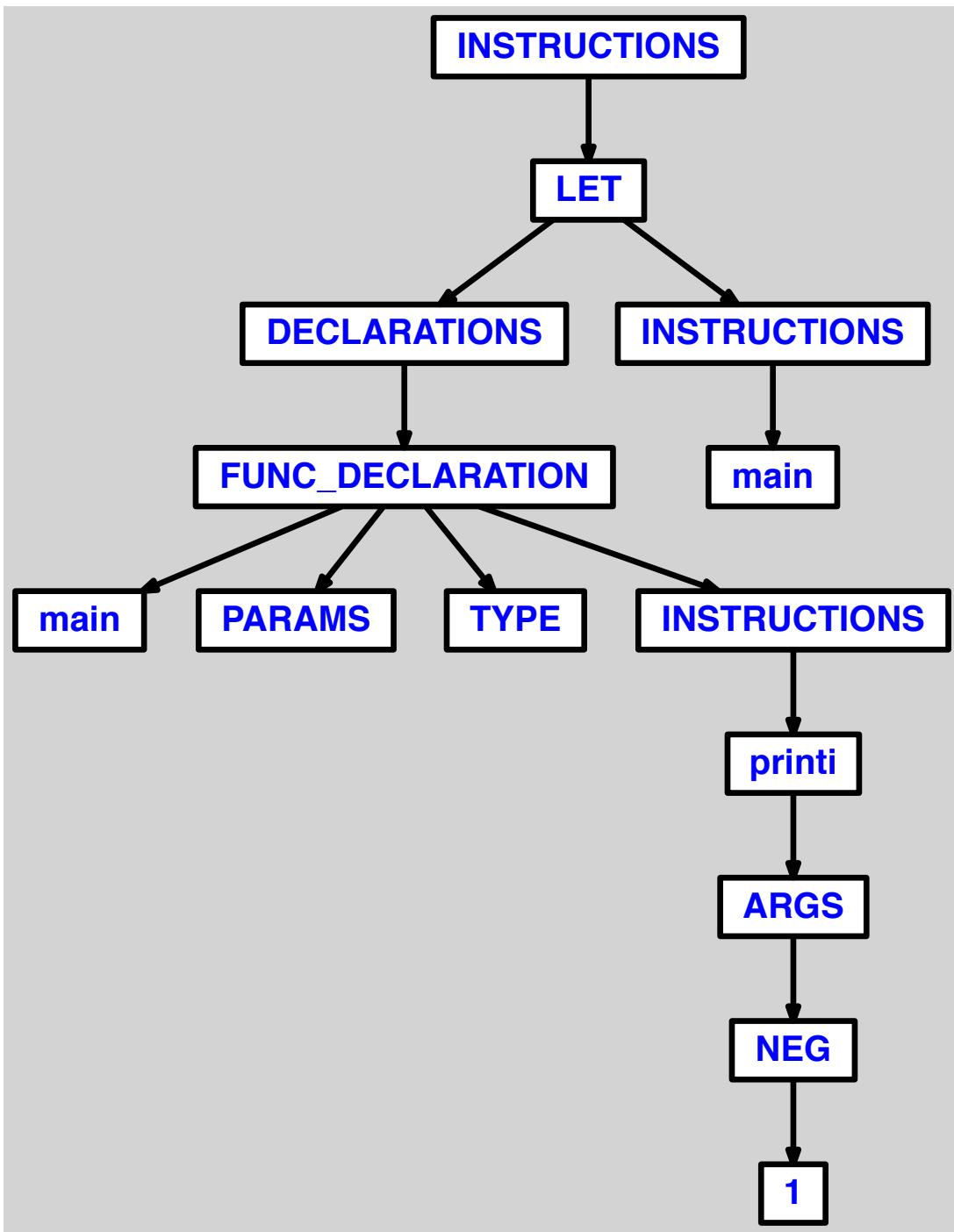
### 3.1.4 division avec operateur mal écrit

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



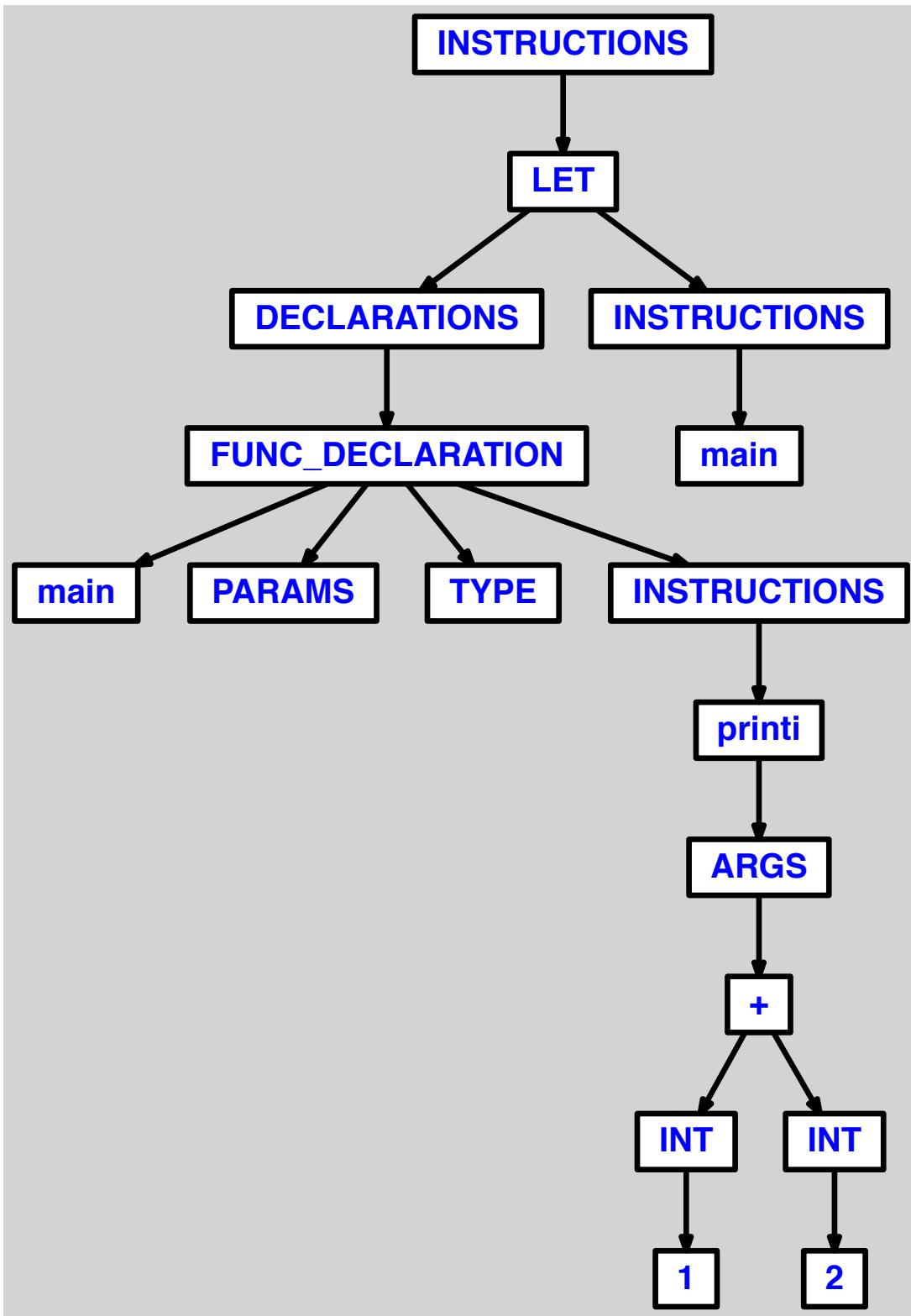
### 3.1.5 moins unaire avec operateur mal ecrit

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



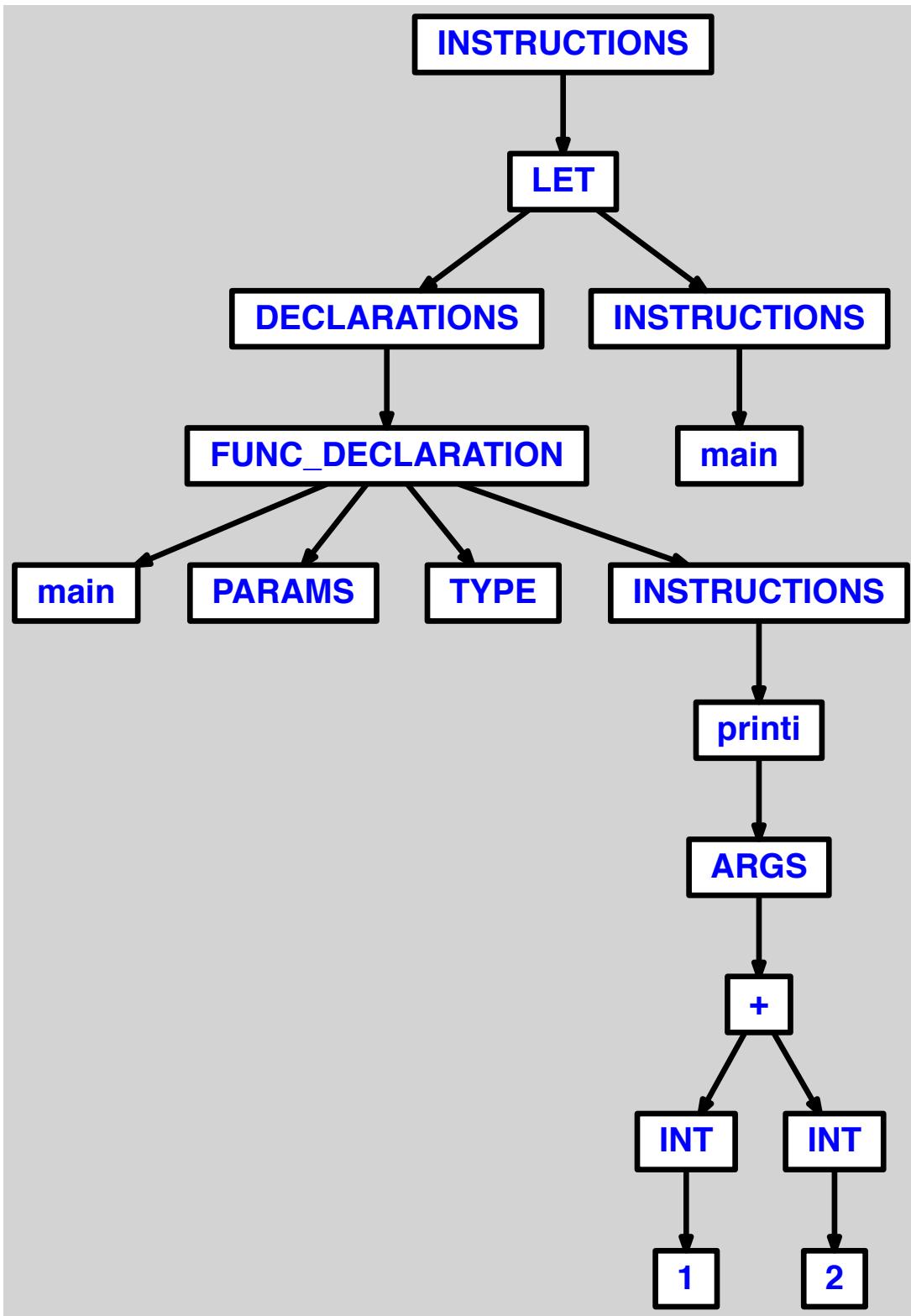
### 3.1.6 calcul à 2 termes avec oubli de parenthèse gauche

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



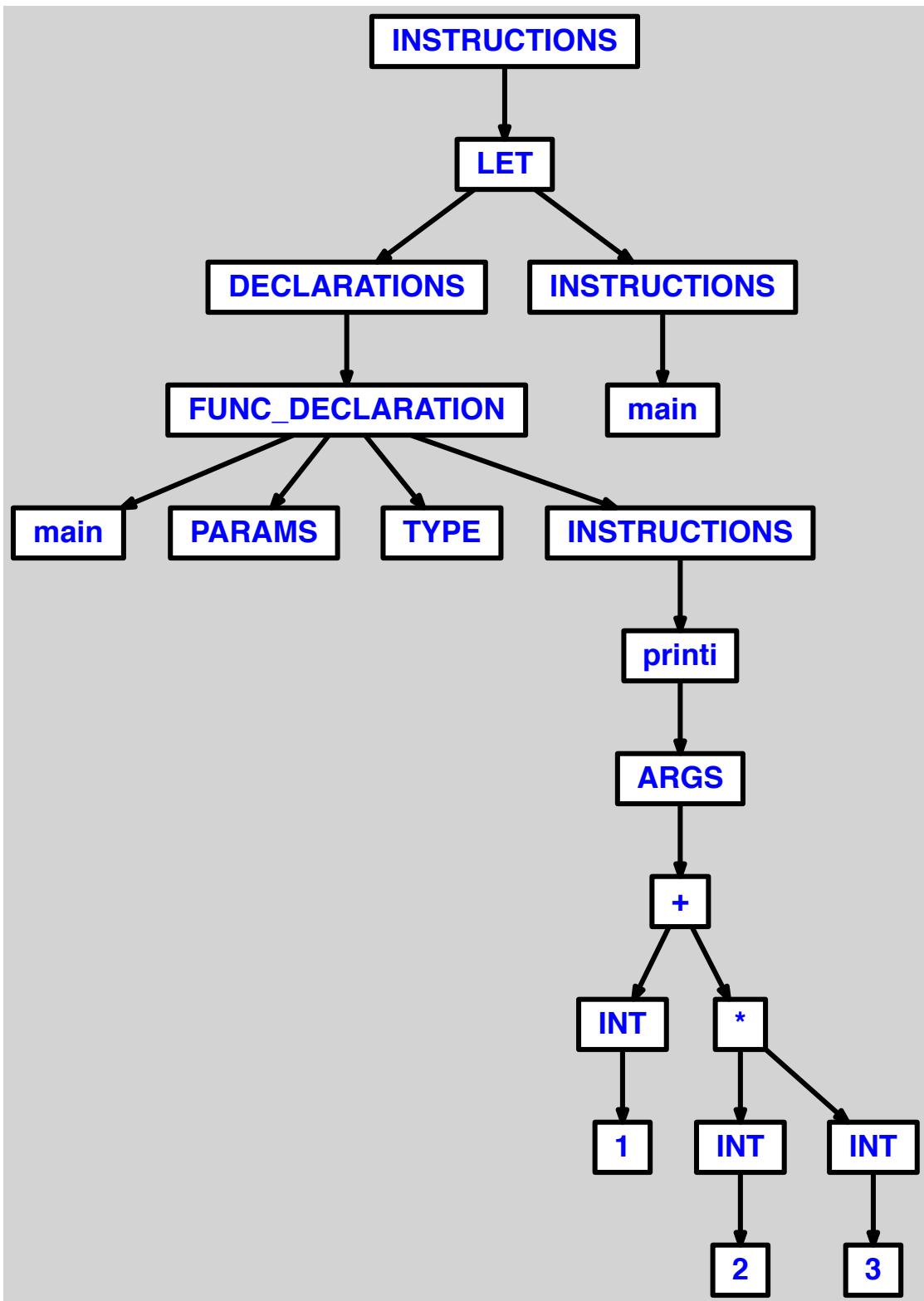
### 3.1.7 calcul à 2 termes avec oubli de parenthèse droite

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



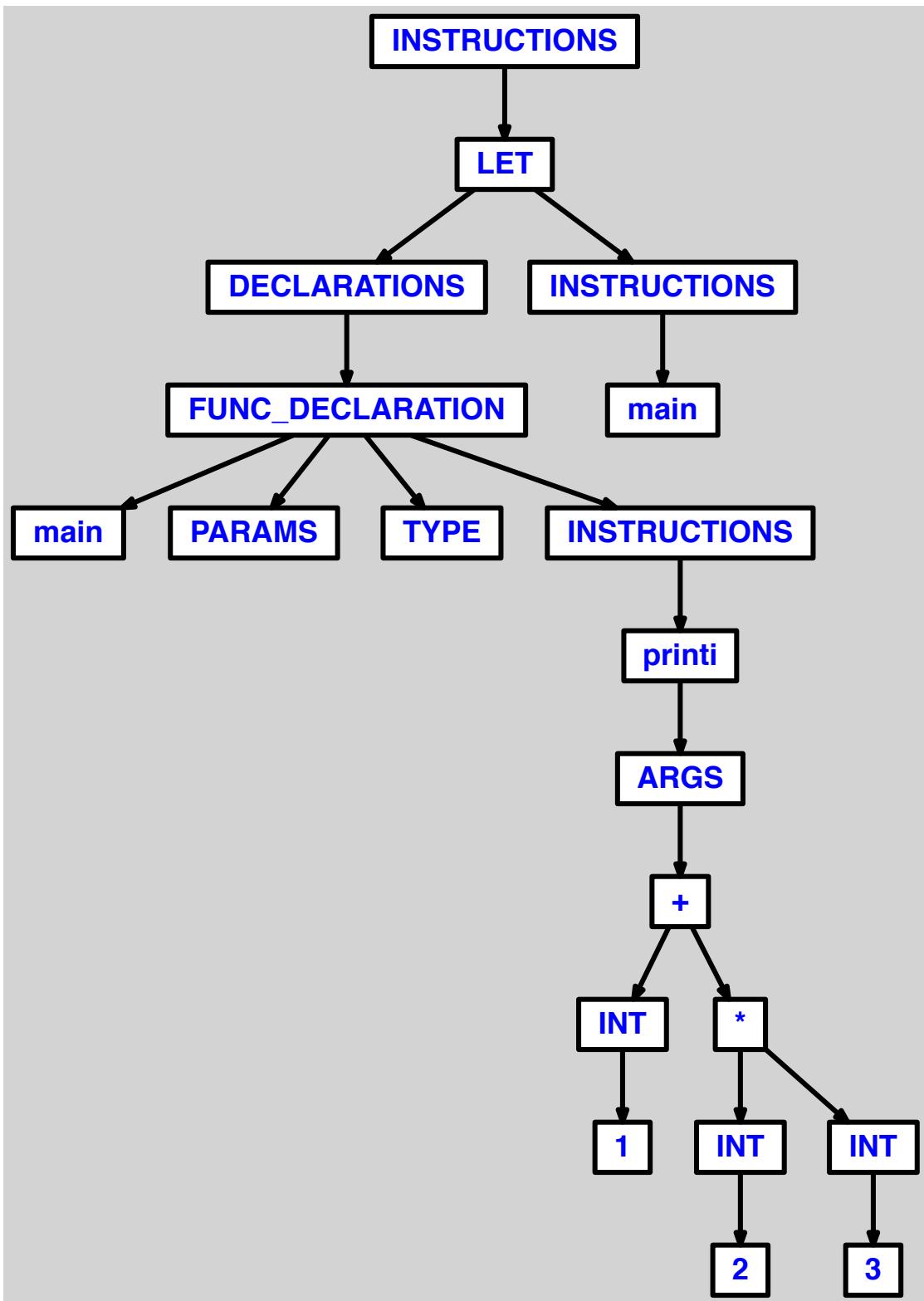
### 3.1.8 calcul à 3 termes avec oubli de parenthèse gauche

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



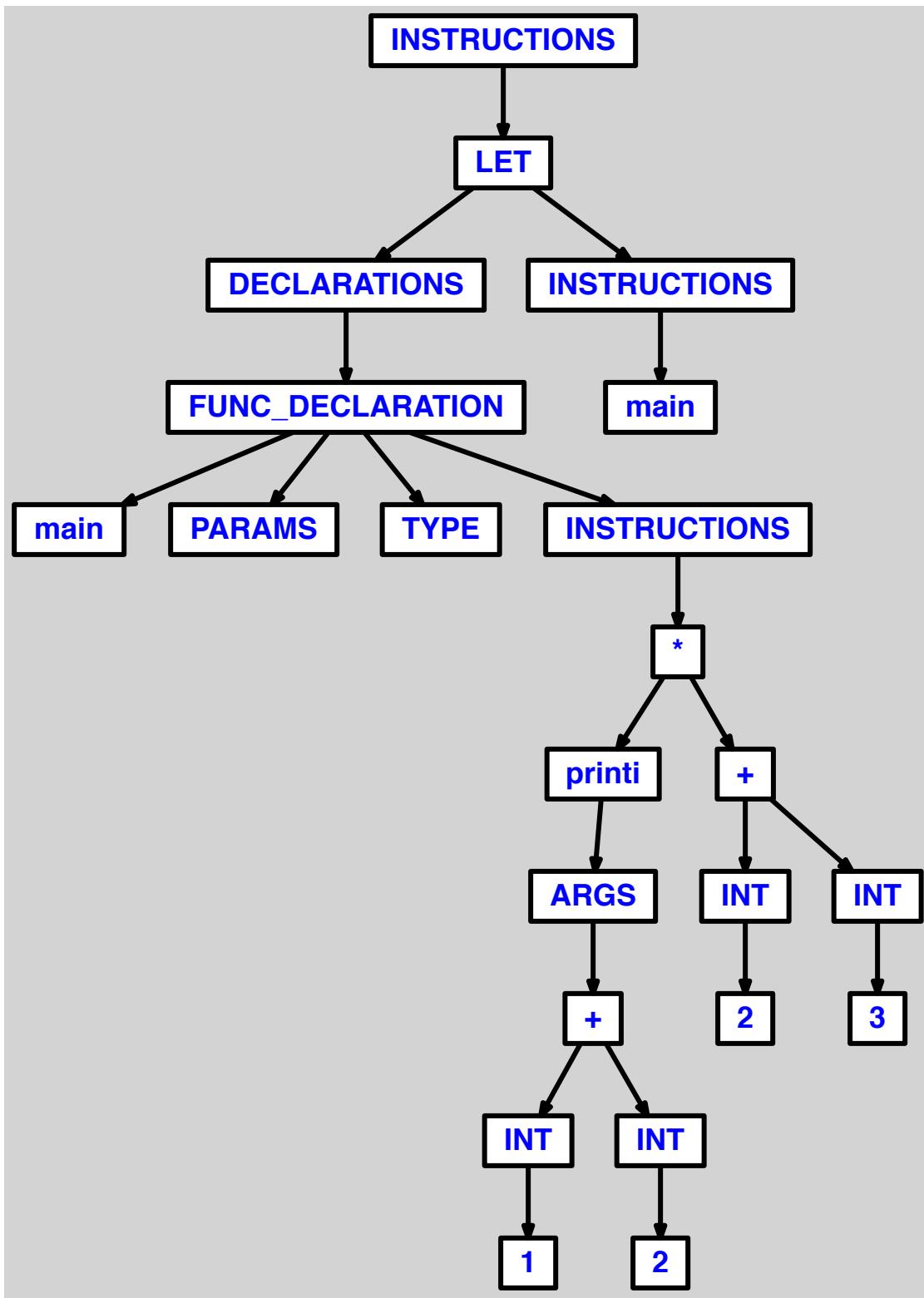
### 3.1.9 calcul à 3 termes avec oubli de parenthèse droite

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



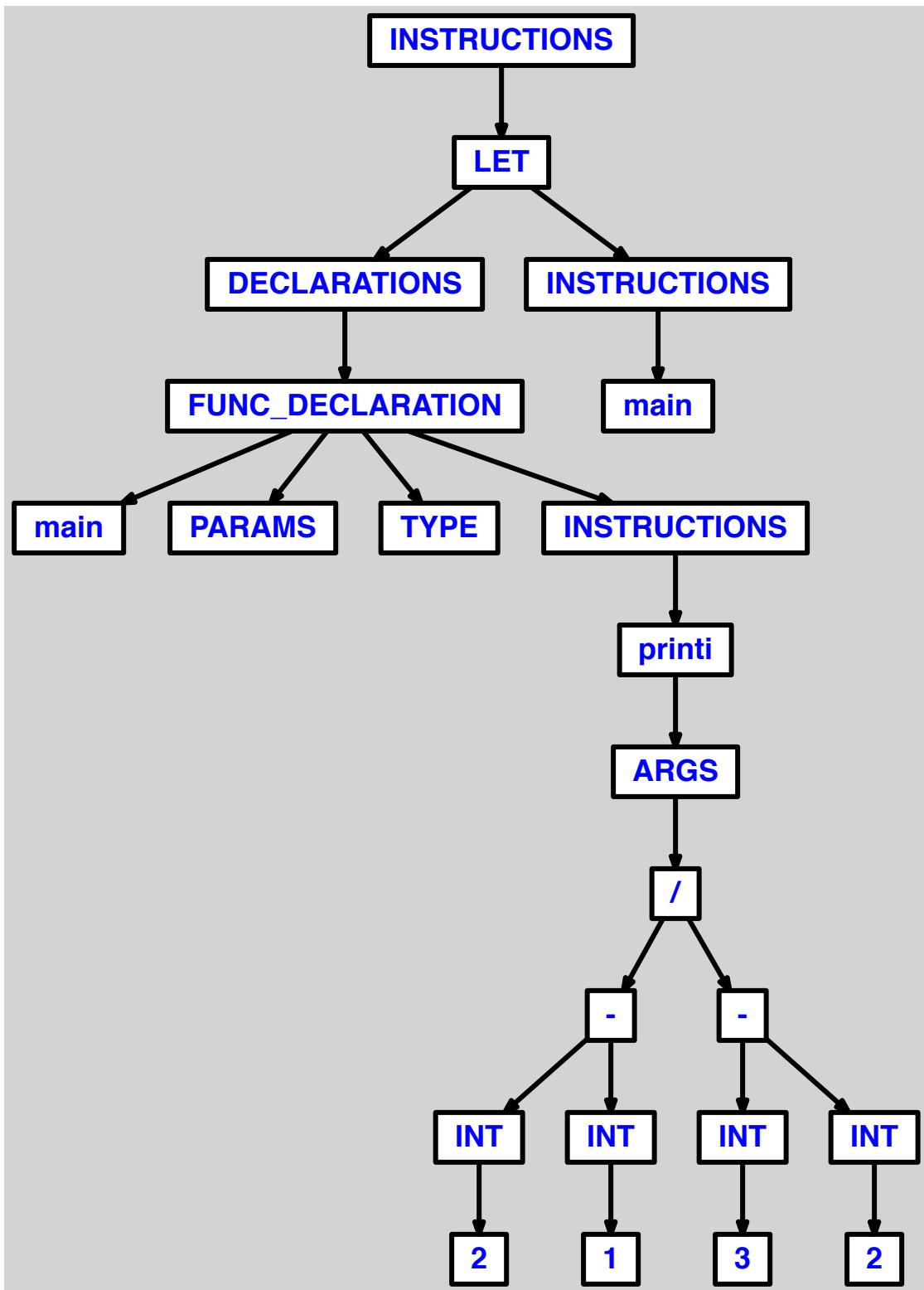
### 3.1.10 calcul a 4 termes avec oubli de parenthese gauche

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



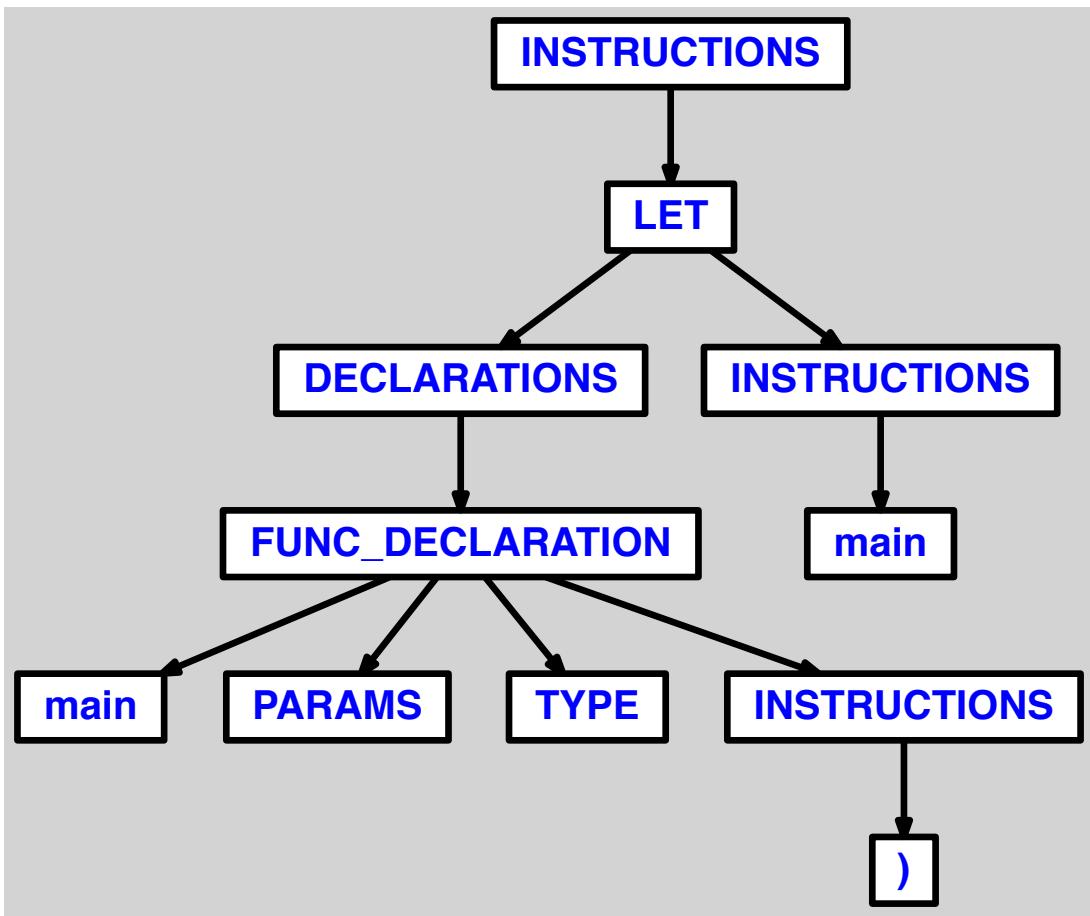
### 3.1.11 calcul a 4 termes avec oubli de parenthese droite

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



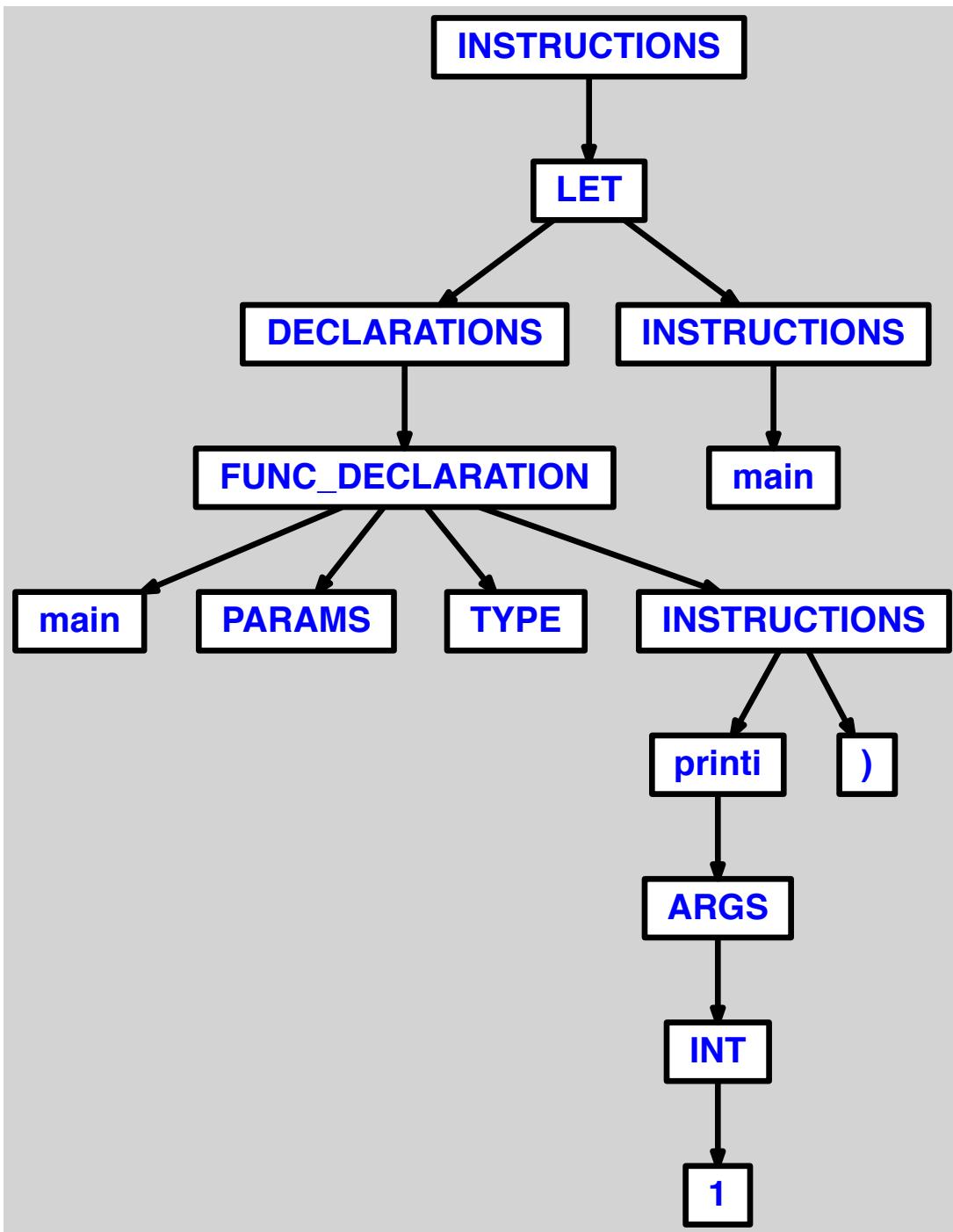
### 3.1.12 parenthesage sans contenu

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



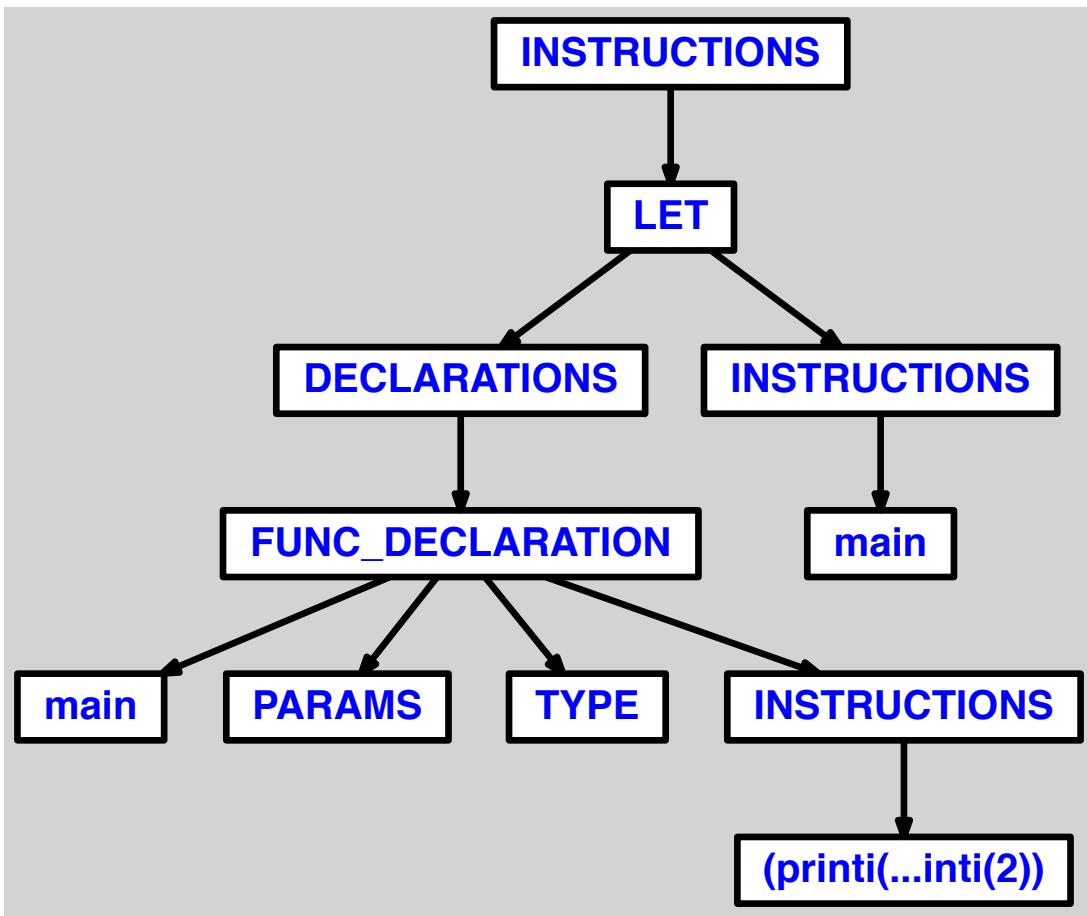
### 3.1.13 expression parenthesée avec <;> en trop

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



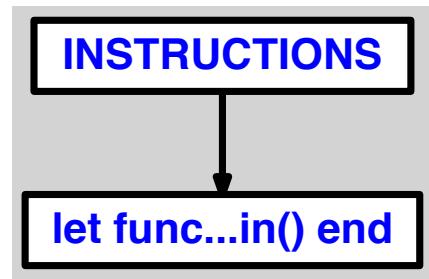
### 3.1.14 expression parenthesée avec oubli de <;>

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



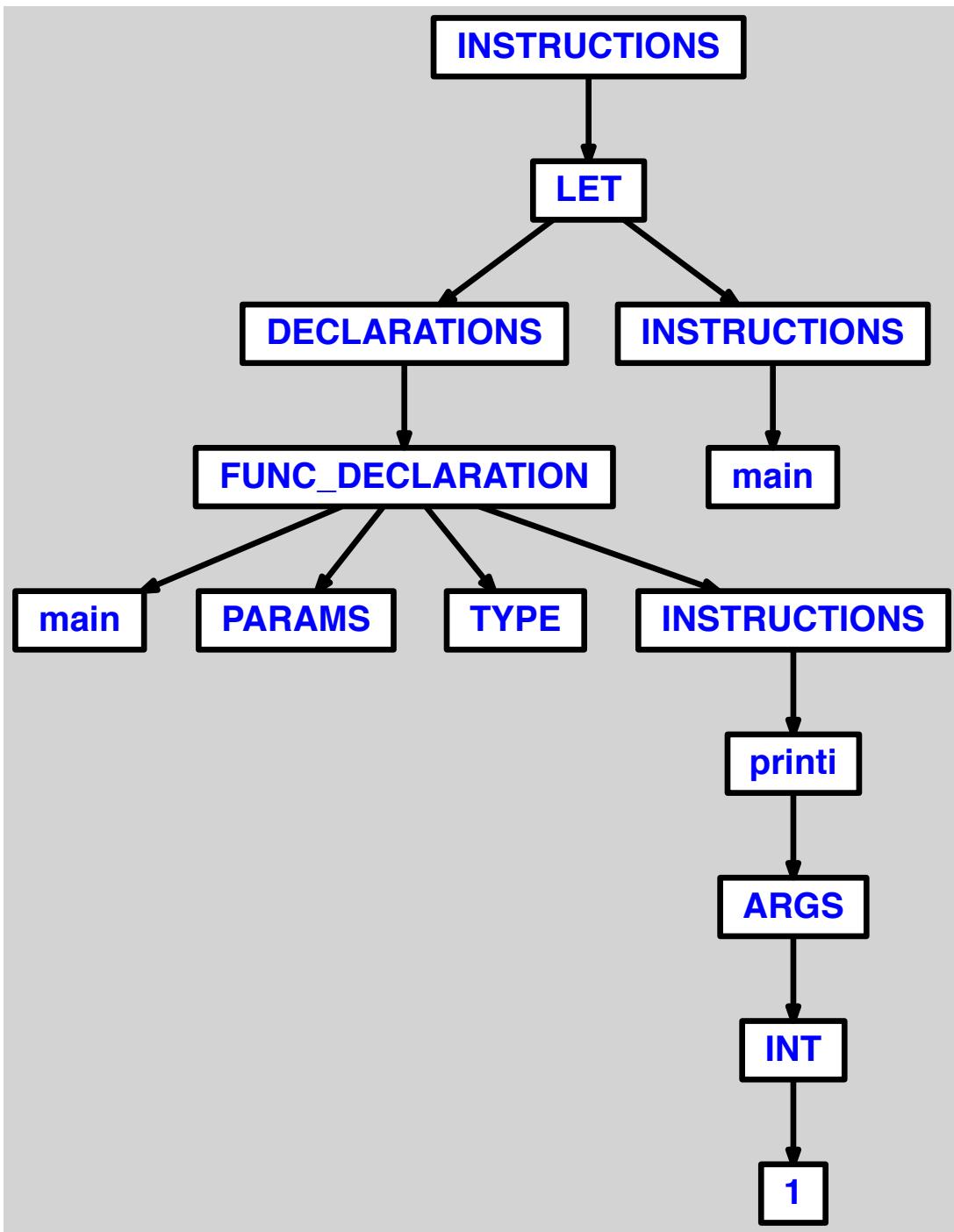
### 3.1.15 expression non parenthesee avec oubli de <;>

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



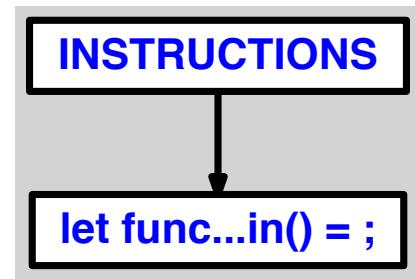
### 3.1.16 expression non parenthesée avec < ;> en trop

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



### 3.1.17 < ;> sans instruction avant

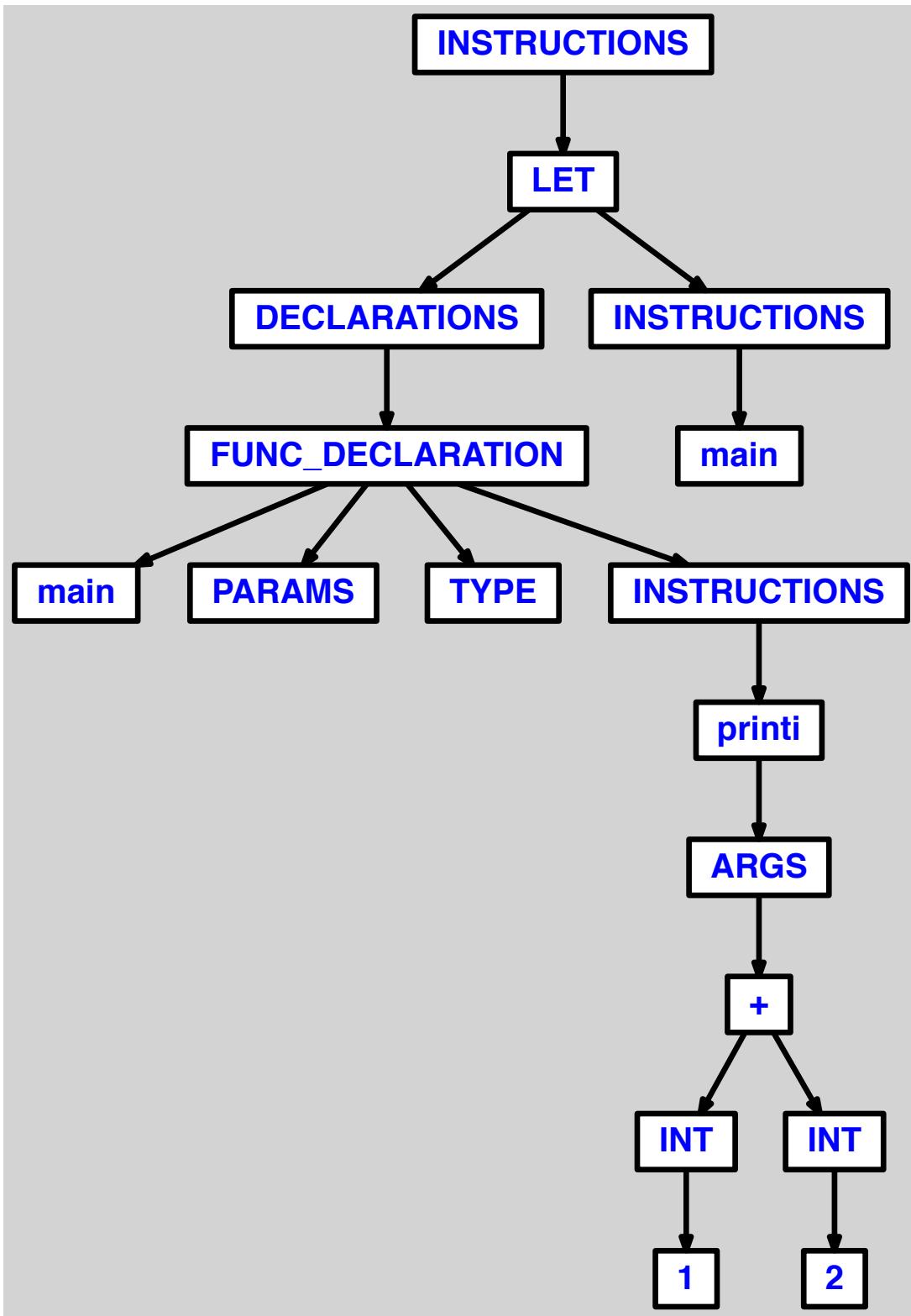
```
1 let function main() = ;
```



## 3.2 OK

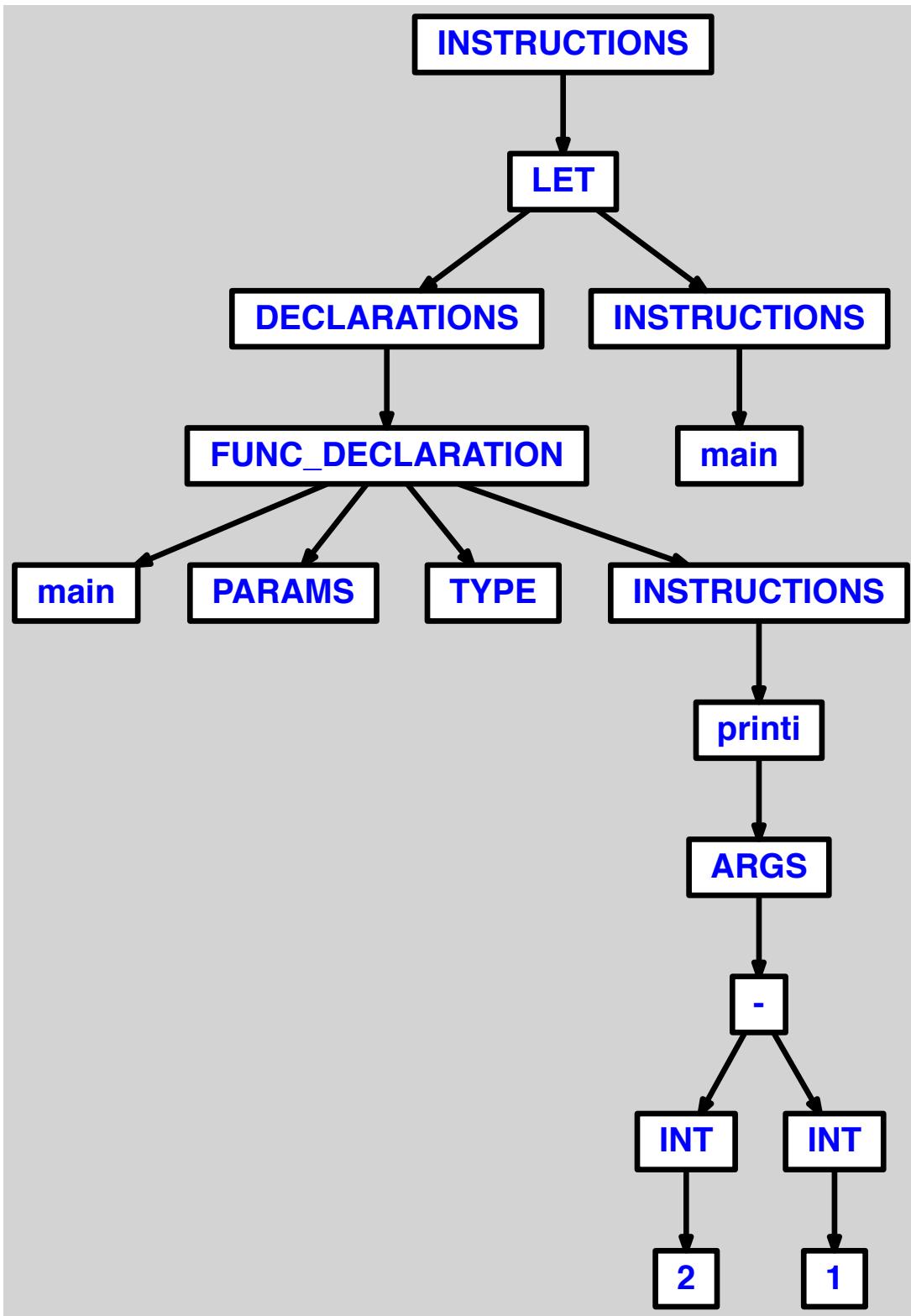
### 3.2.1 addition simple, a 2 termes

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



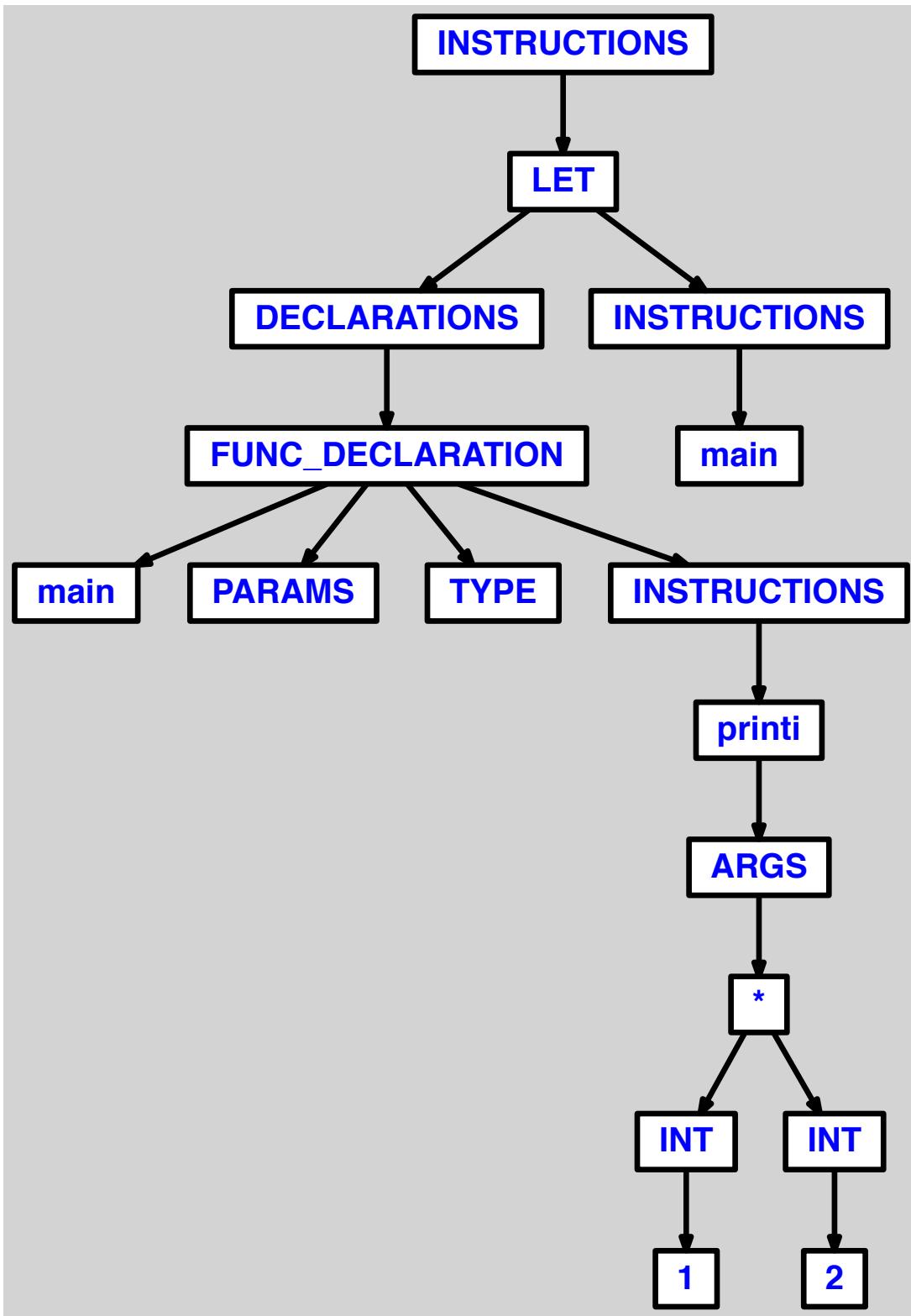
### 3.2.2 soustraction simple, à 2 termes

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



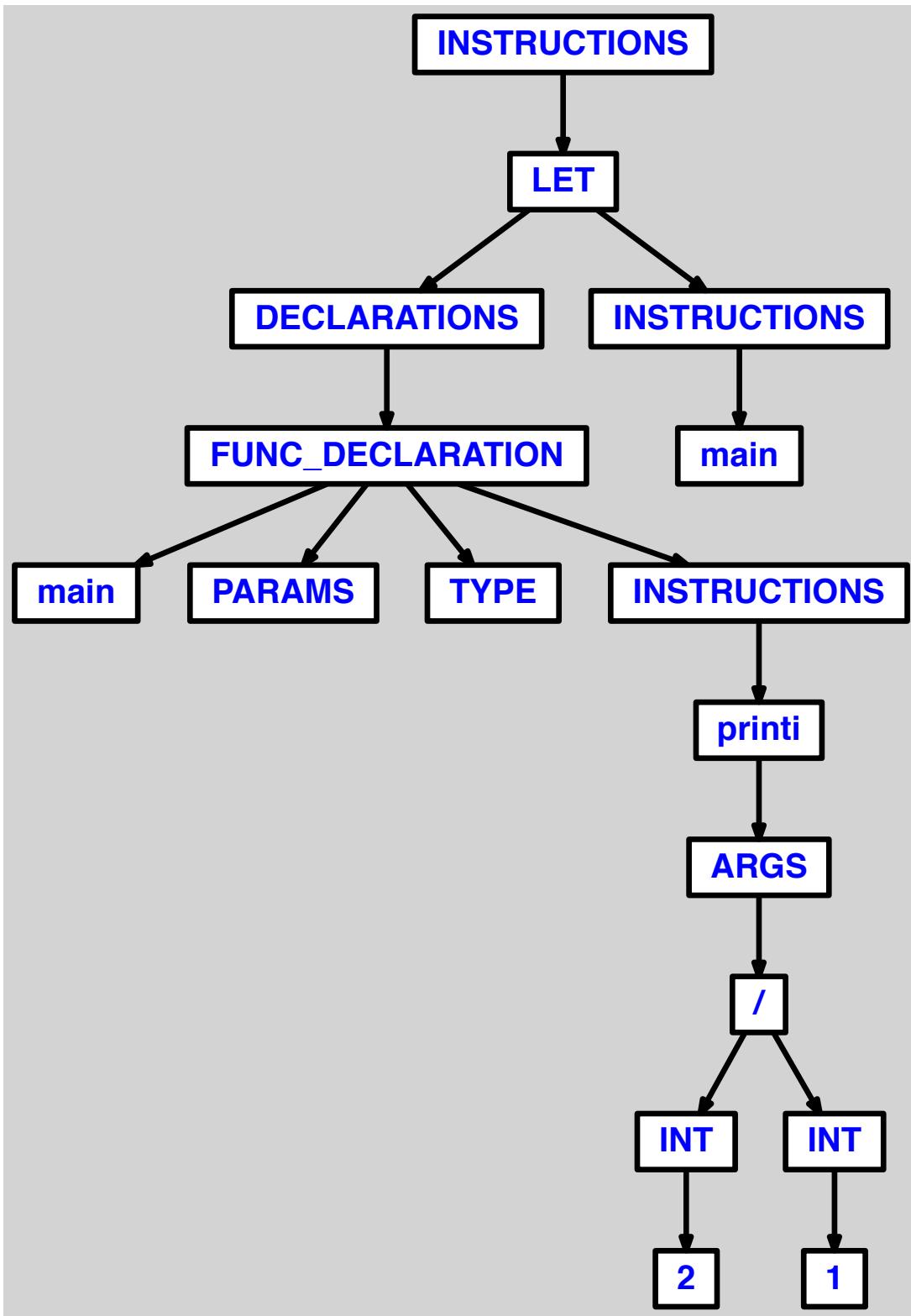
### 3.2.3 multiplication simple, a 2 termes

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



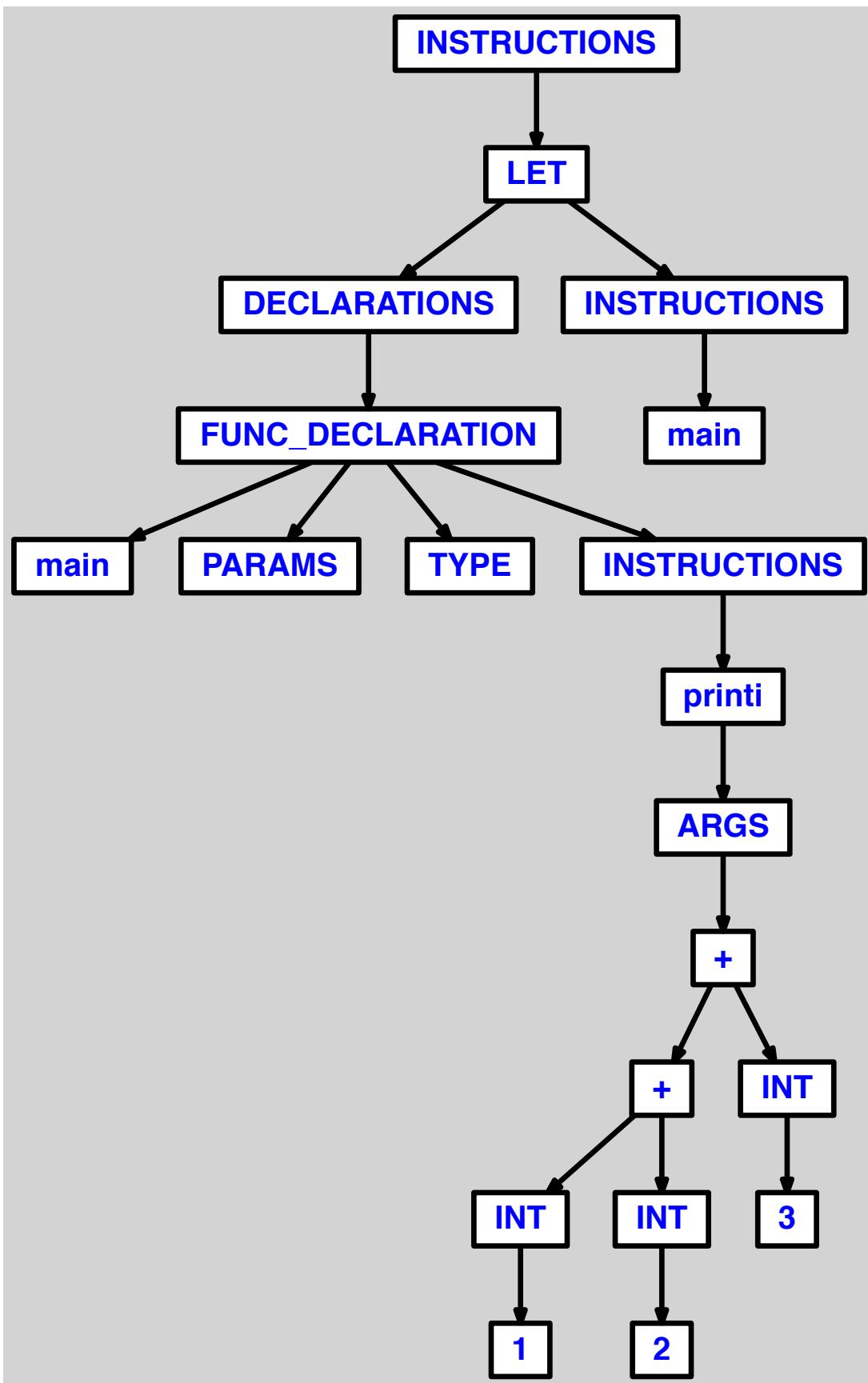
### 3.2.4 division simple, à 2 termes

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



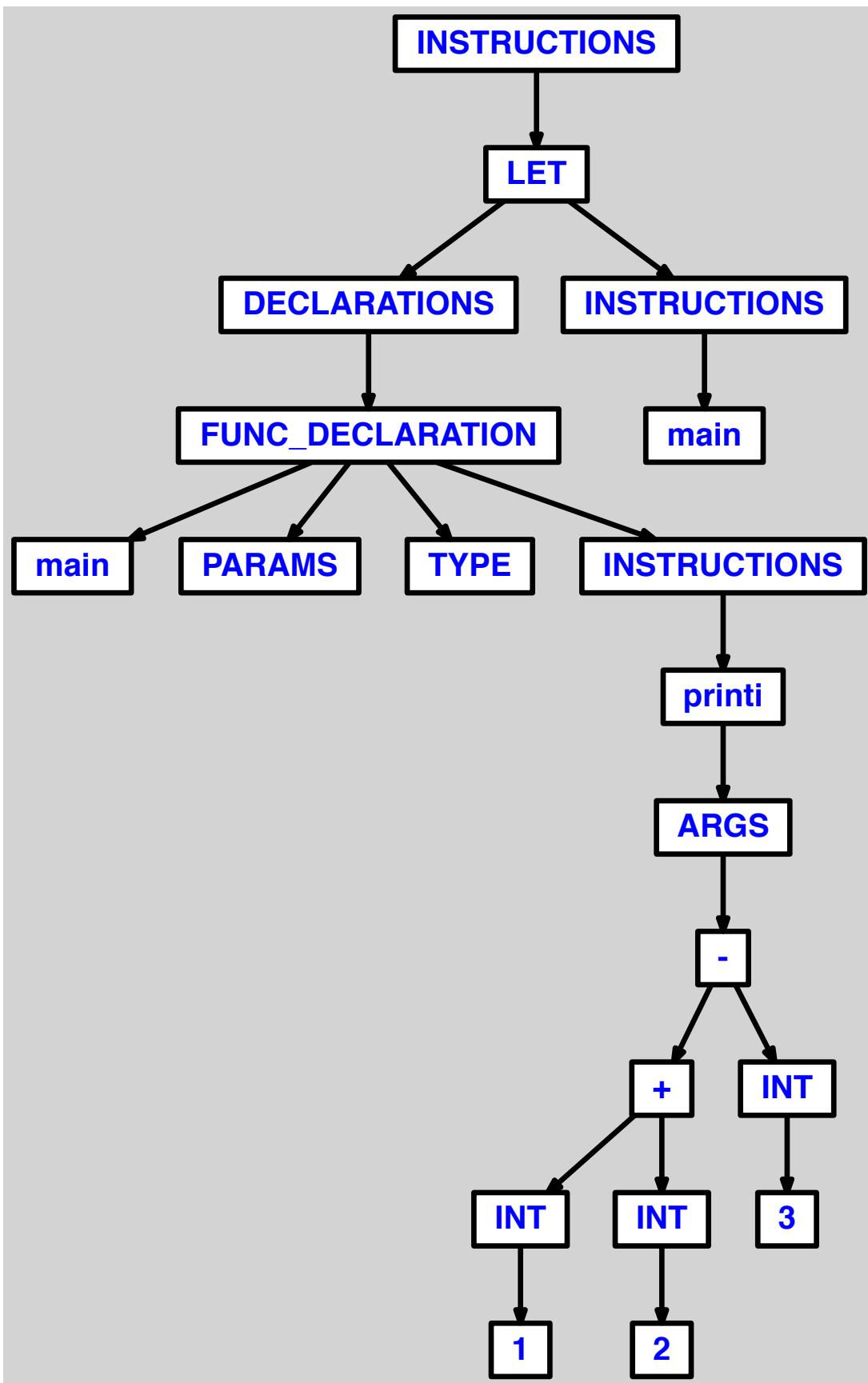
### 3.2.5 addition à 3 termes

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



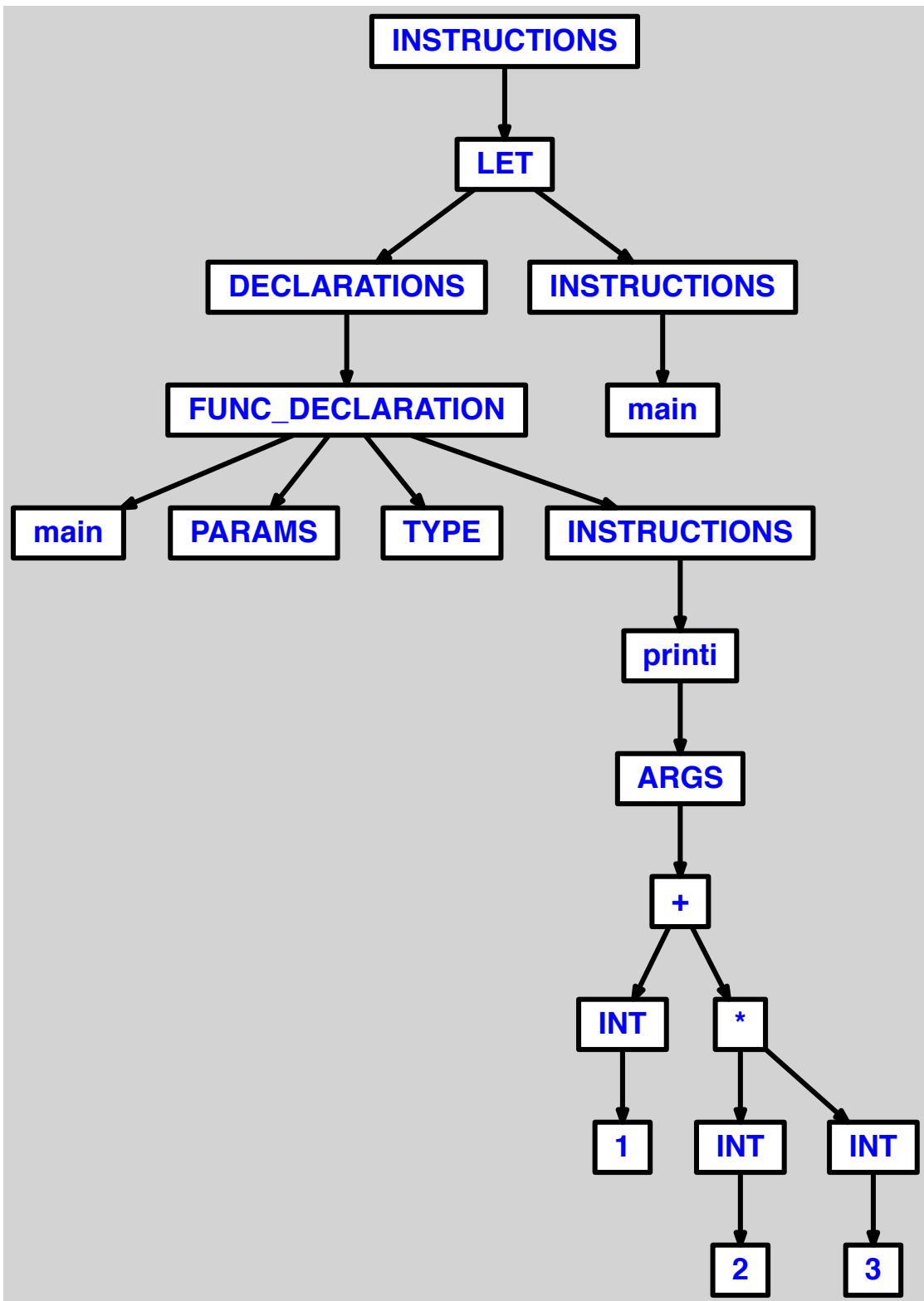
### 3.2.6 addition suivie de soustraction

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



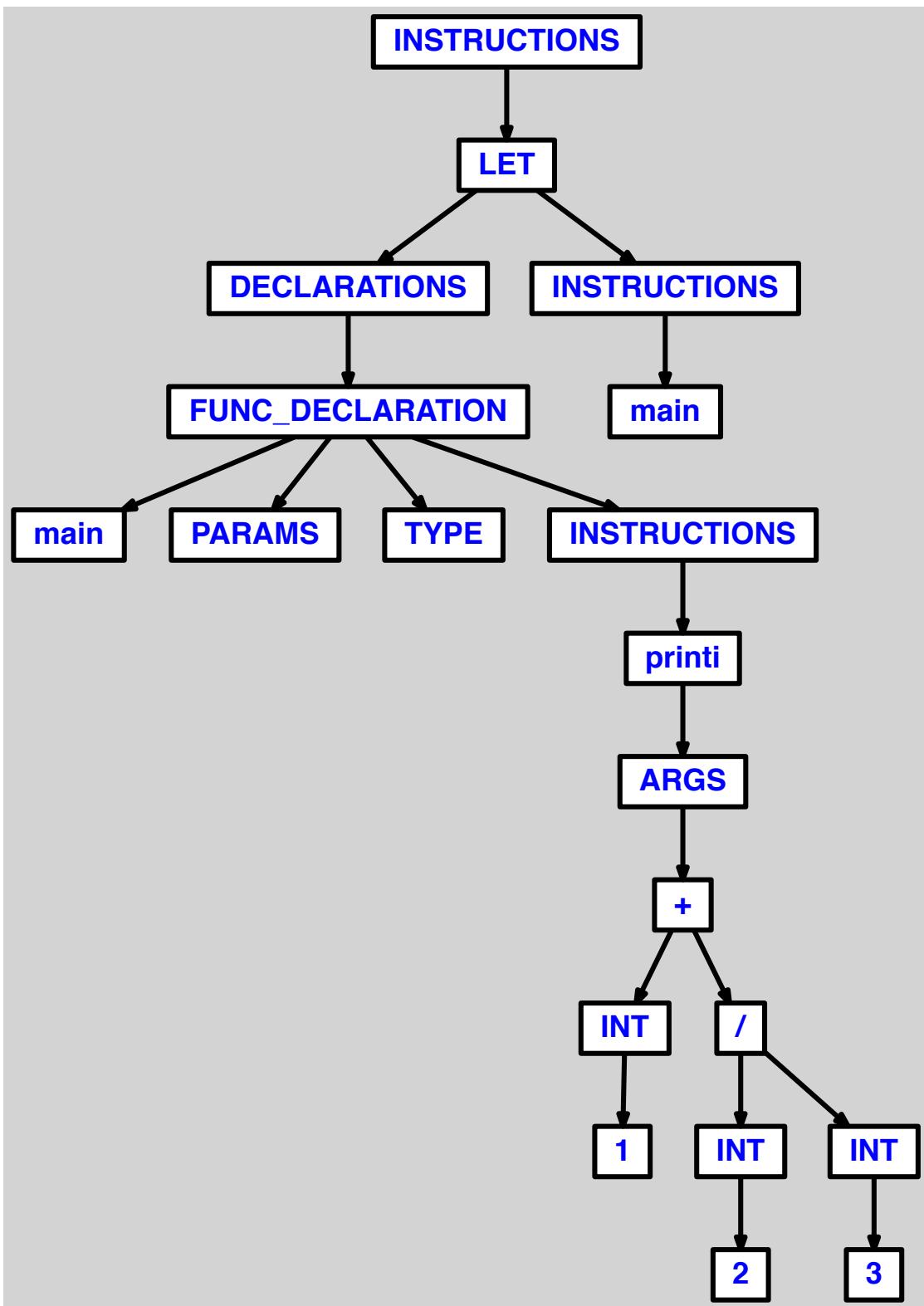
### 3.2.7 addition suivie de multiplication

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



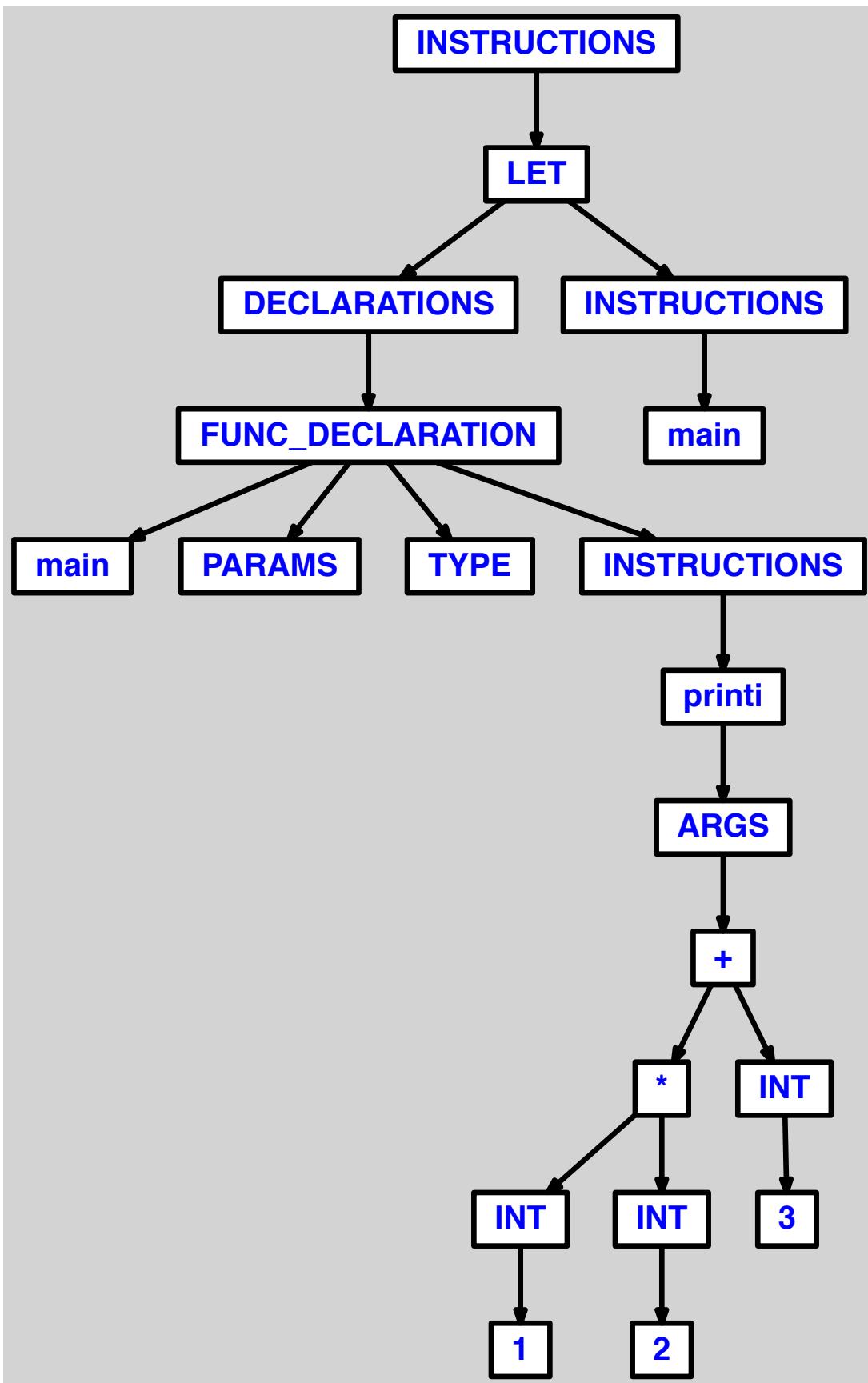
### 3.2.8 addition suivie de division

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



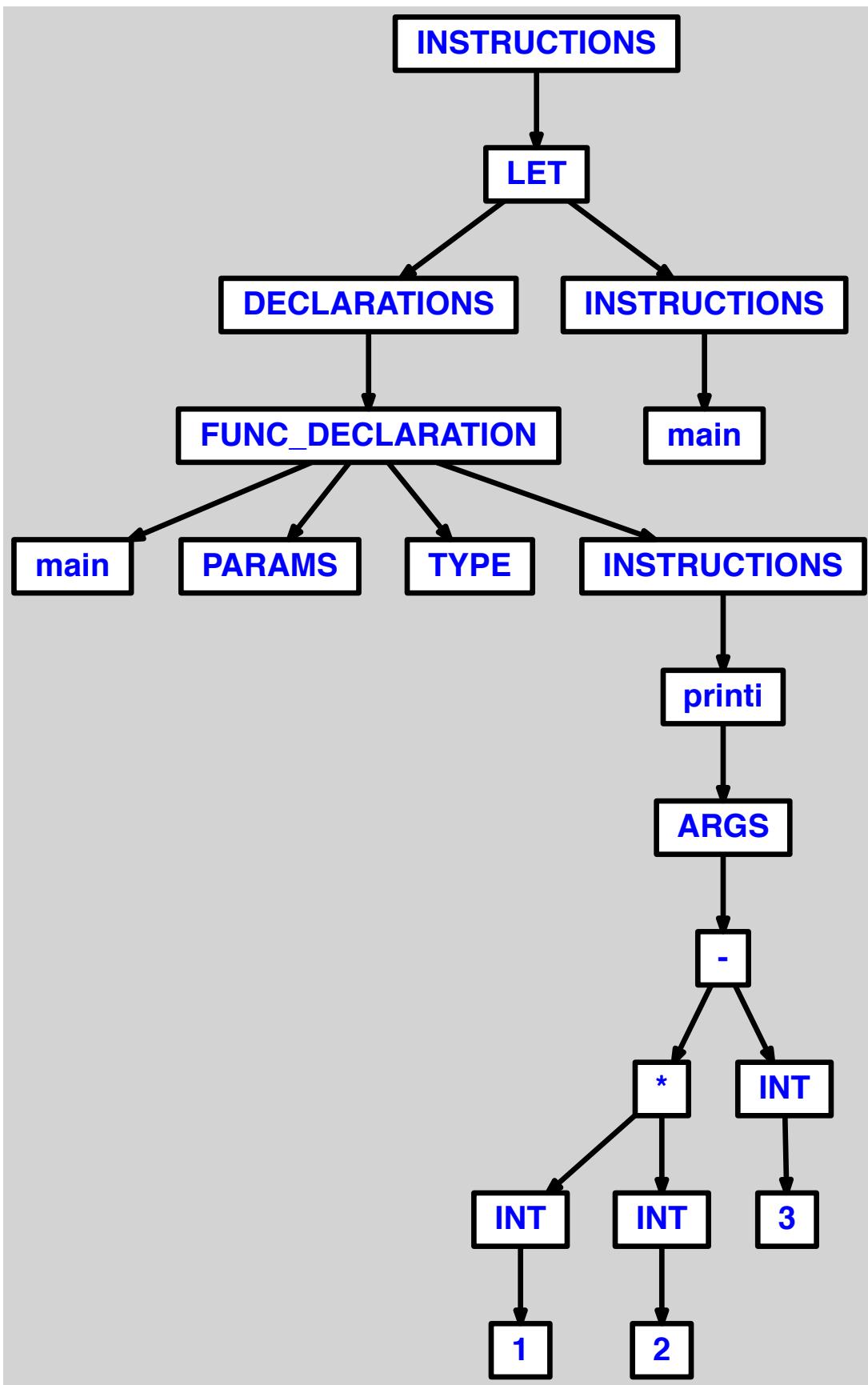
### 3.2.9 multiplication suivie d'addition

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



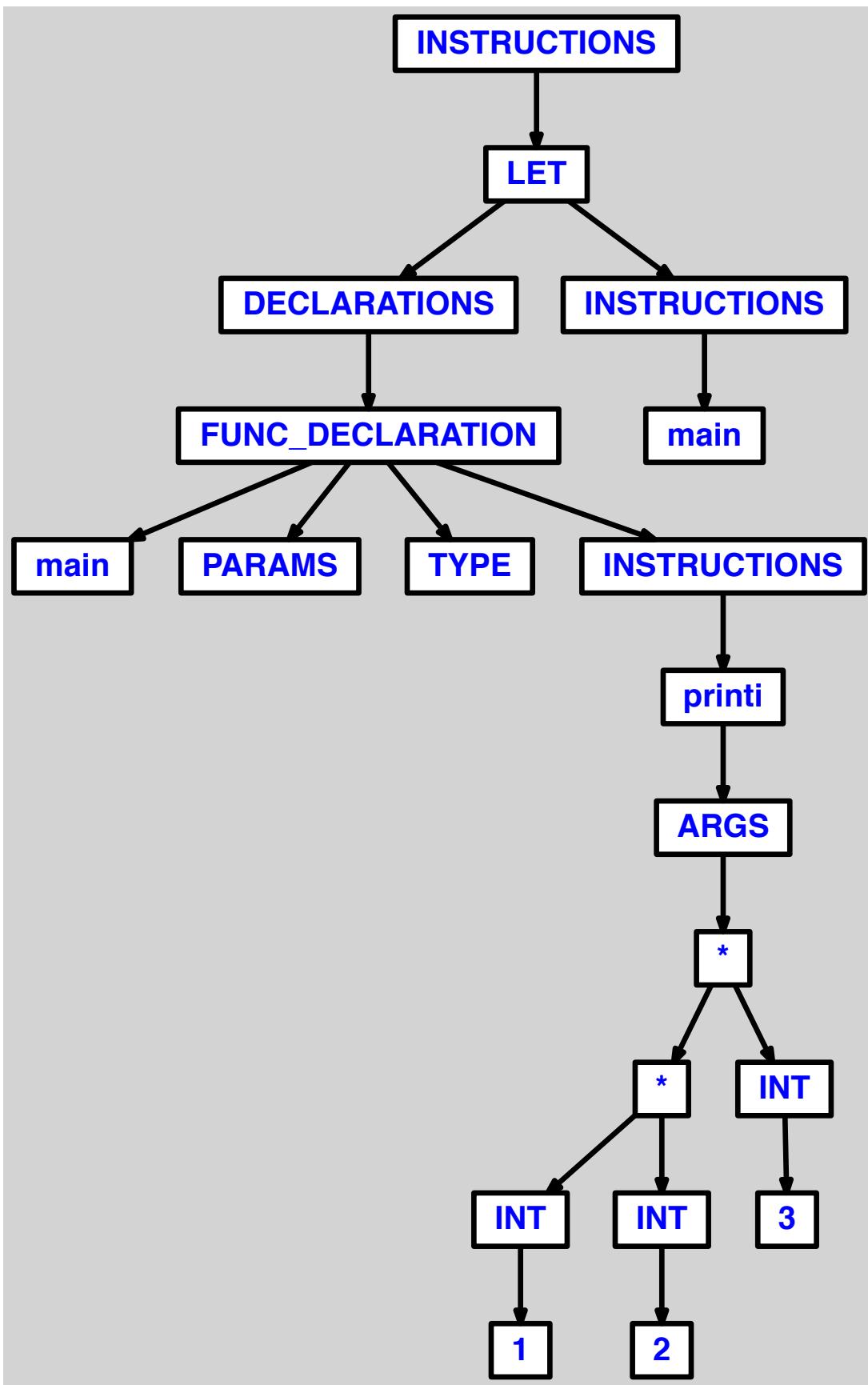
### 3.2.10 multiplication suivie de soustraction

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



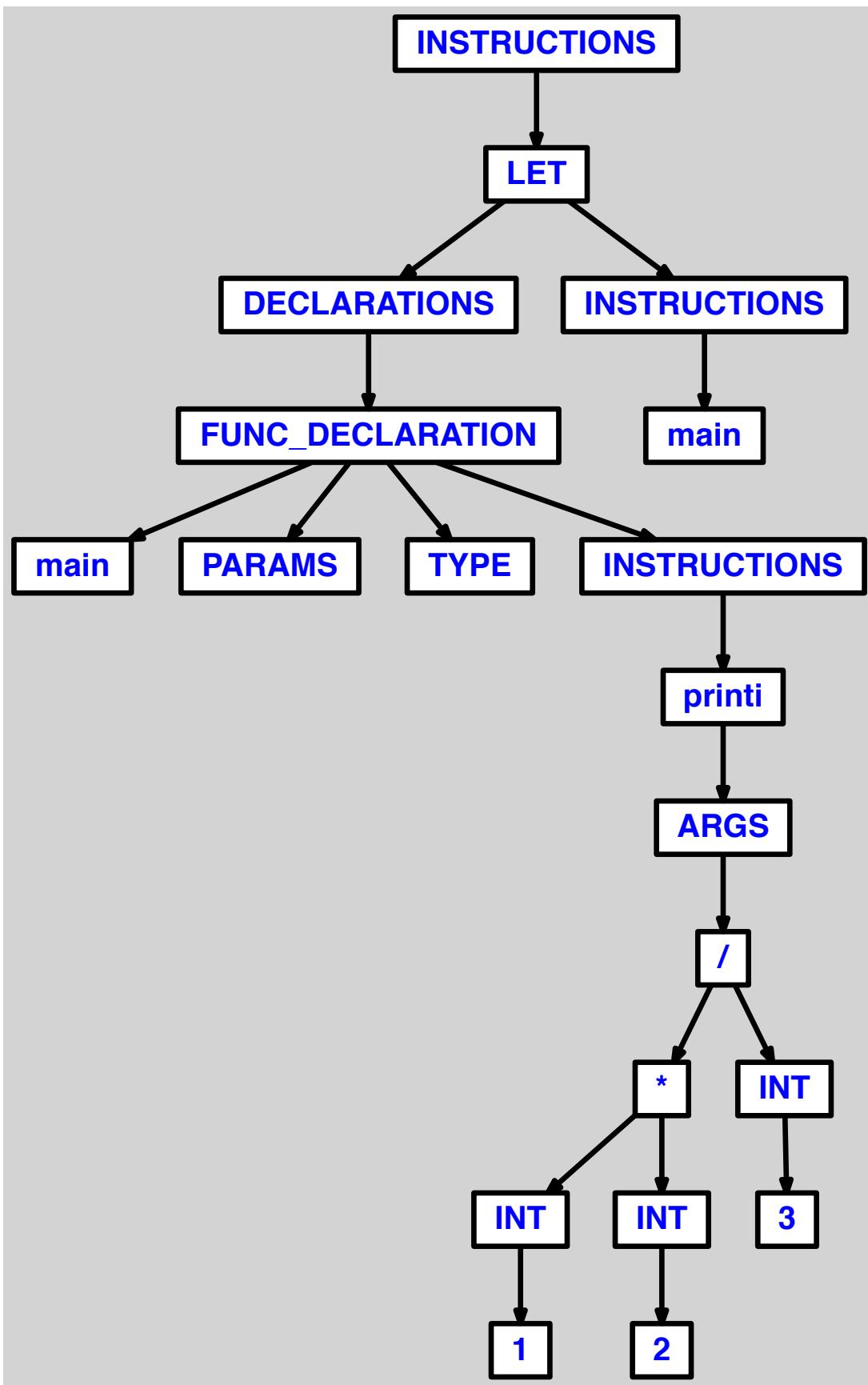
### 3.2.11 multiplication a 3 termes

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



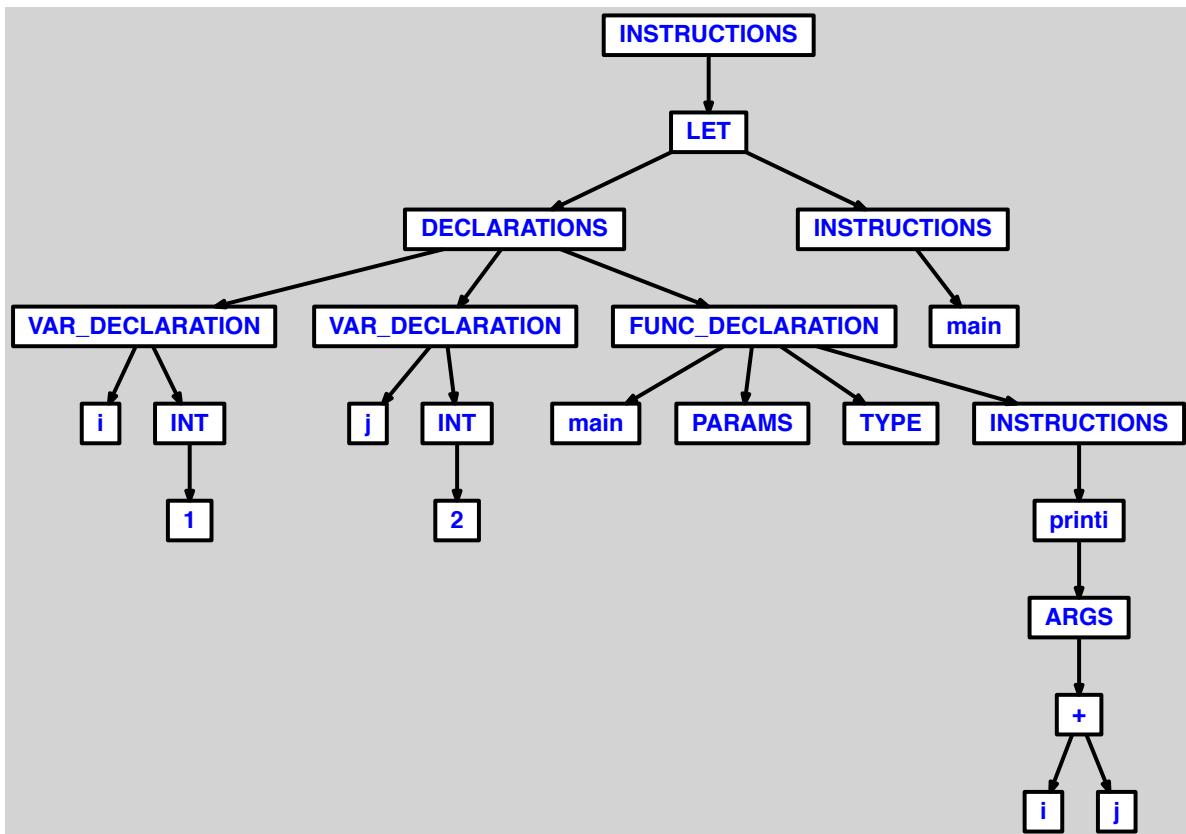
### 3.2.12 multiplication suivie de division

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



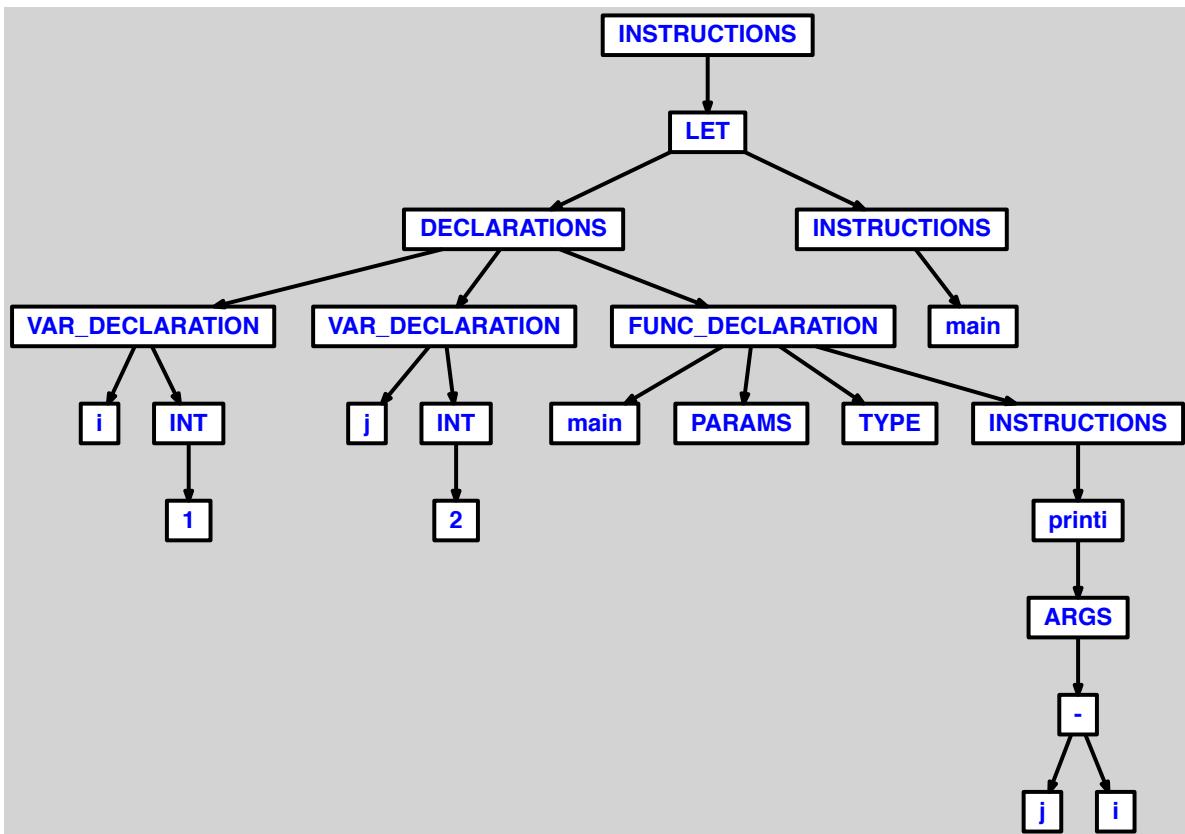
### 3.2.13 addition simple, à 2 termes, identifiés par des variables

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(i+j)
6 in main() end
```



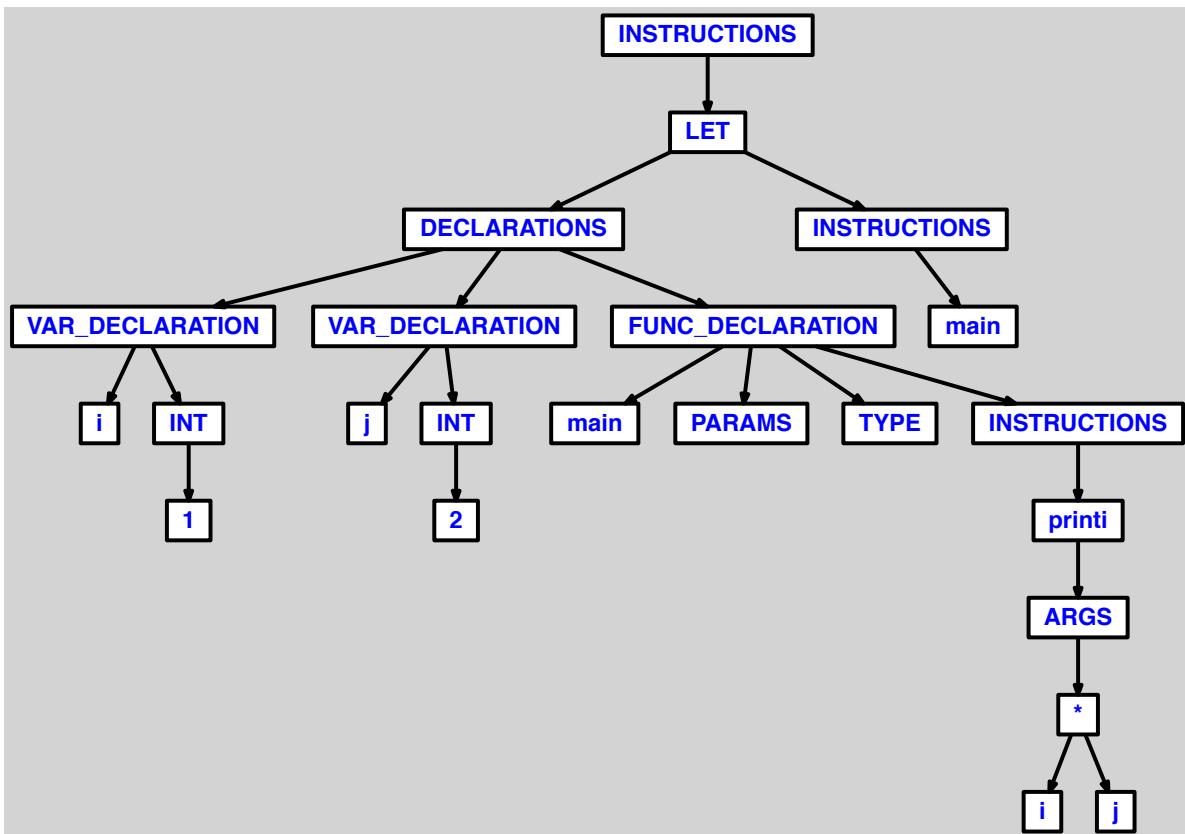
### 3.2.14 soustraction simple, à 2 termes, identifiés par des variables

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(j-i)
6 in main() end
```



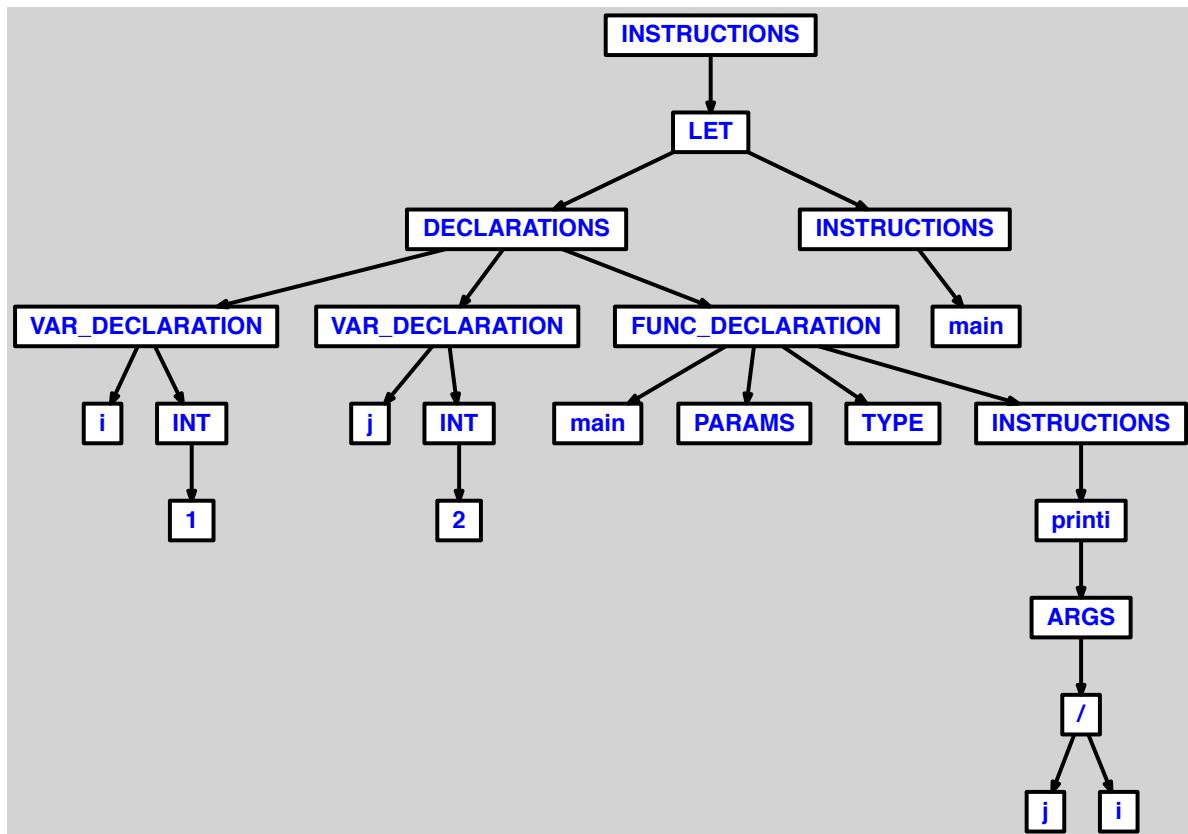
### 3.2.15 multiplication simple, à 2 termes, identifiés par des variables

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(i*j)
6 in main() end
```



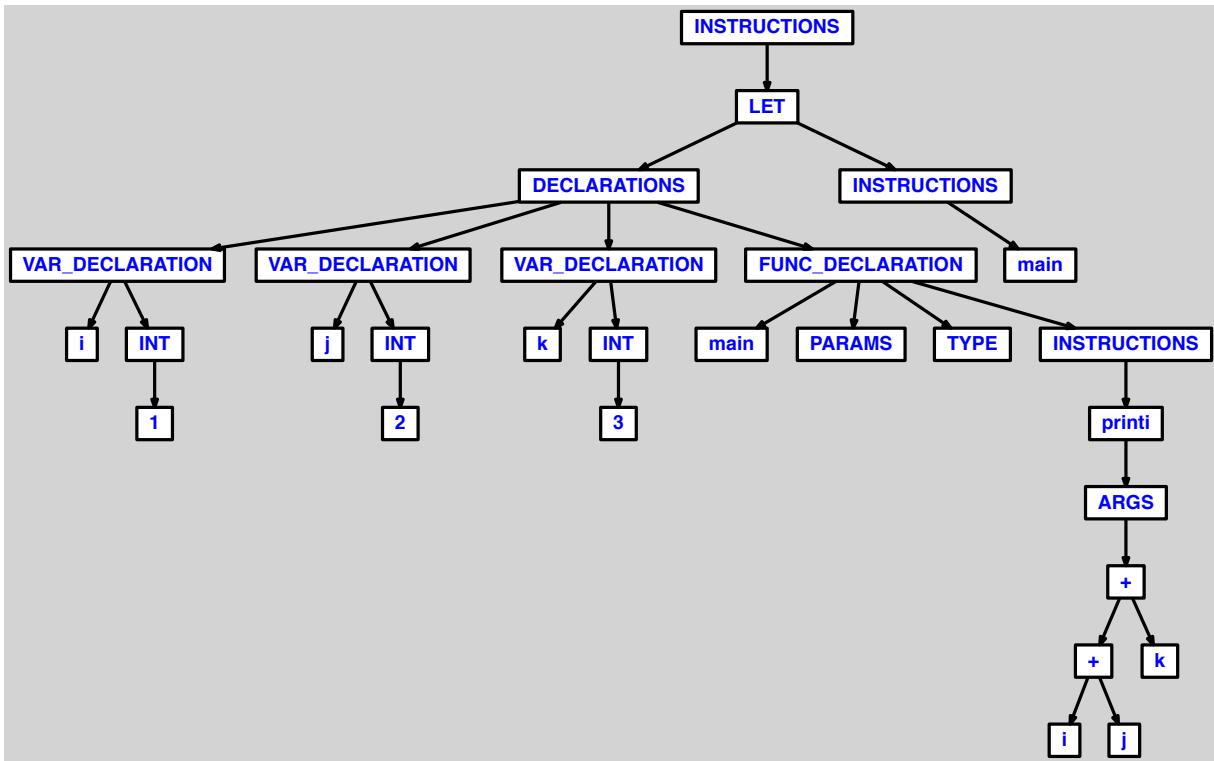
### 3.2.16 division simple, à 2 termes, identifiés par des variables

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(j/i)
6 in main() end
```



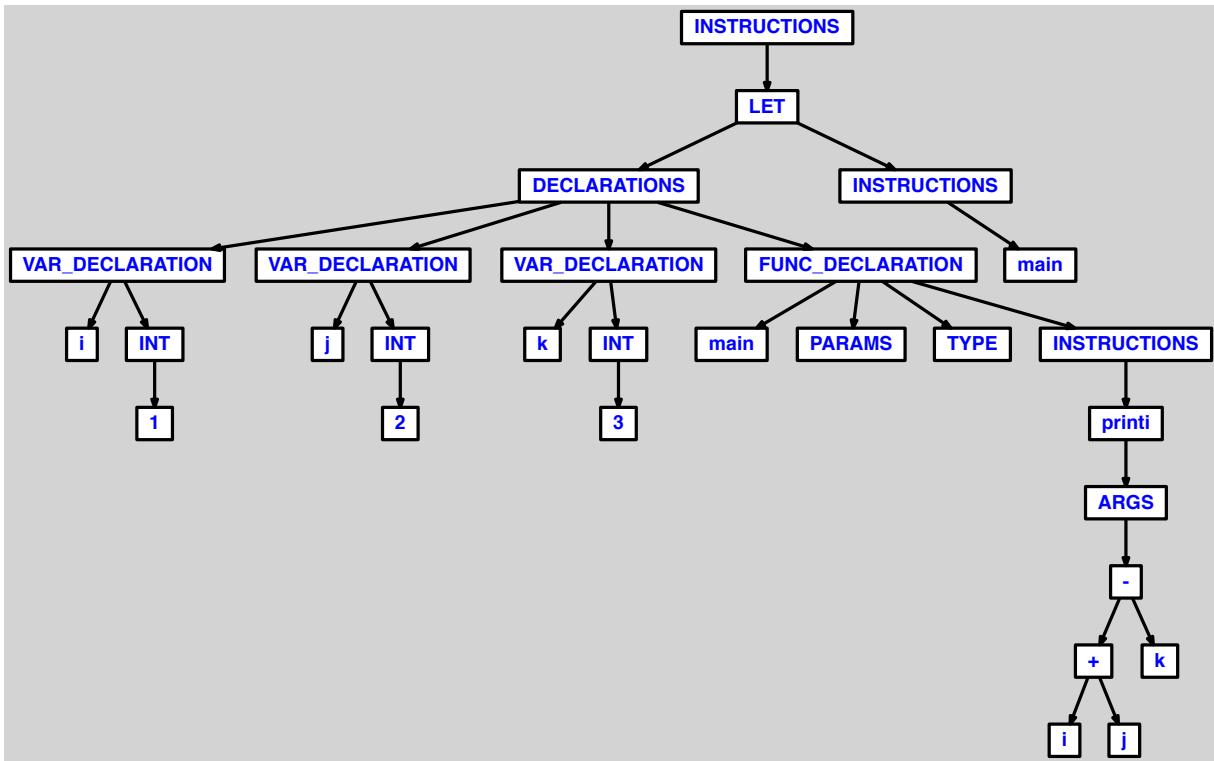
### 3.2.17 addition à 3 termes, identifiés par des variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+j+k)
7 in main() end
```



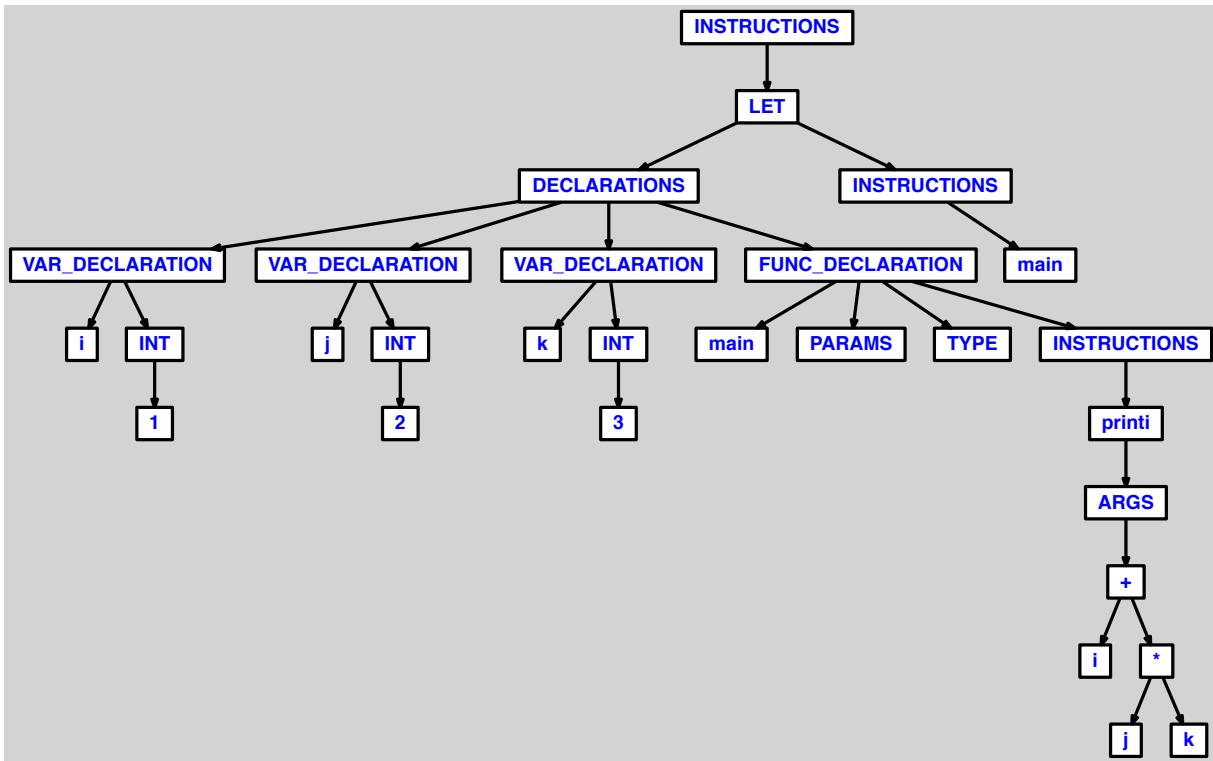
### 3.2.18 addition suivie de soustraction, avec termes identifiés par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+j-k)
7 in main() end
```



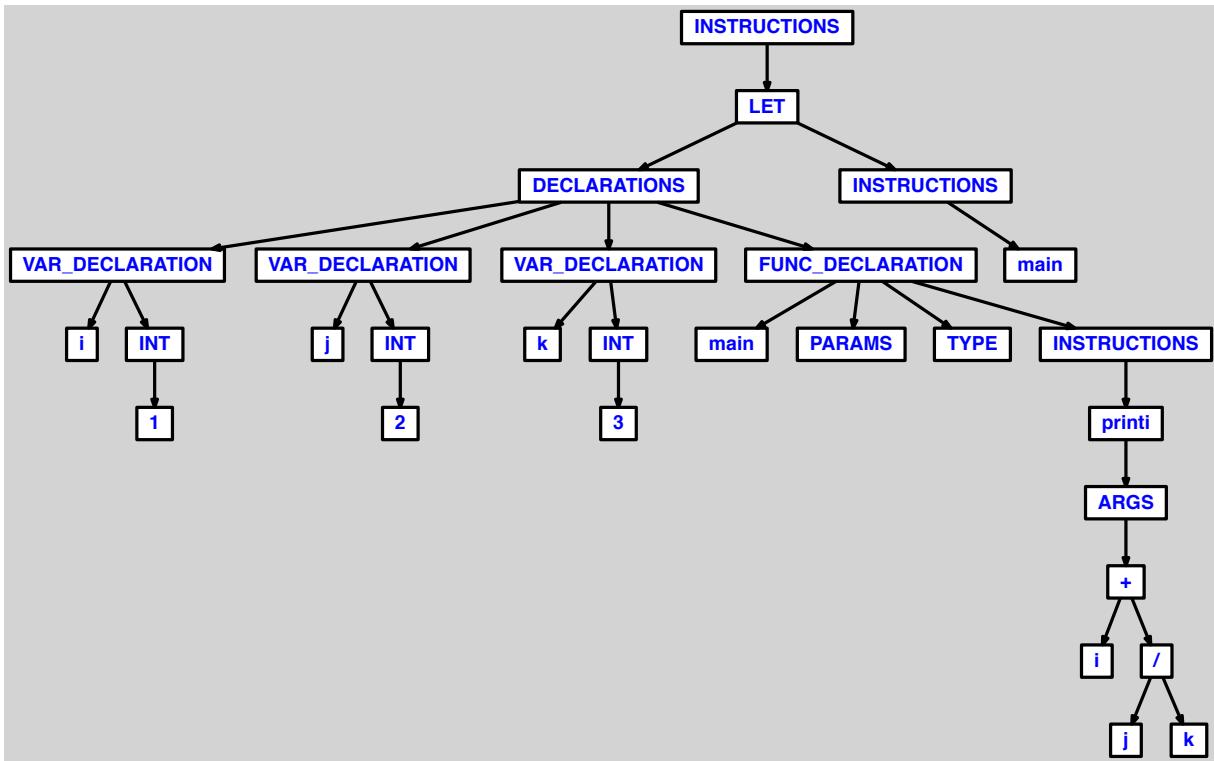
### 3.2.19 addition suivie de multiplication, avec termes identifiés par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+j*k)
7 in main() end
```



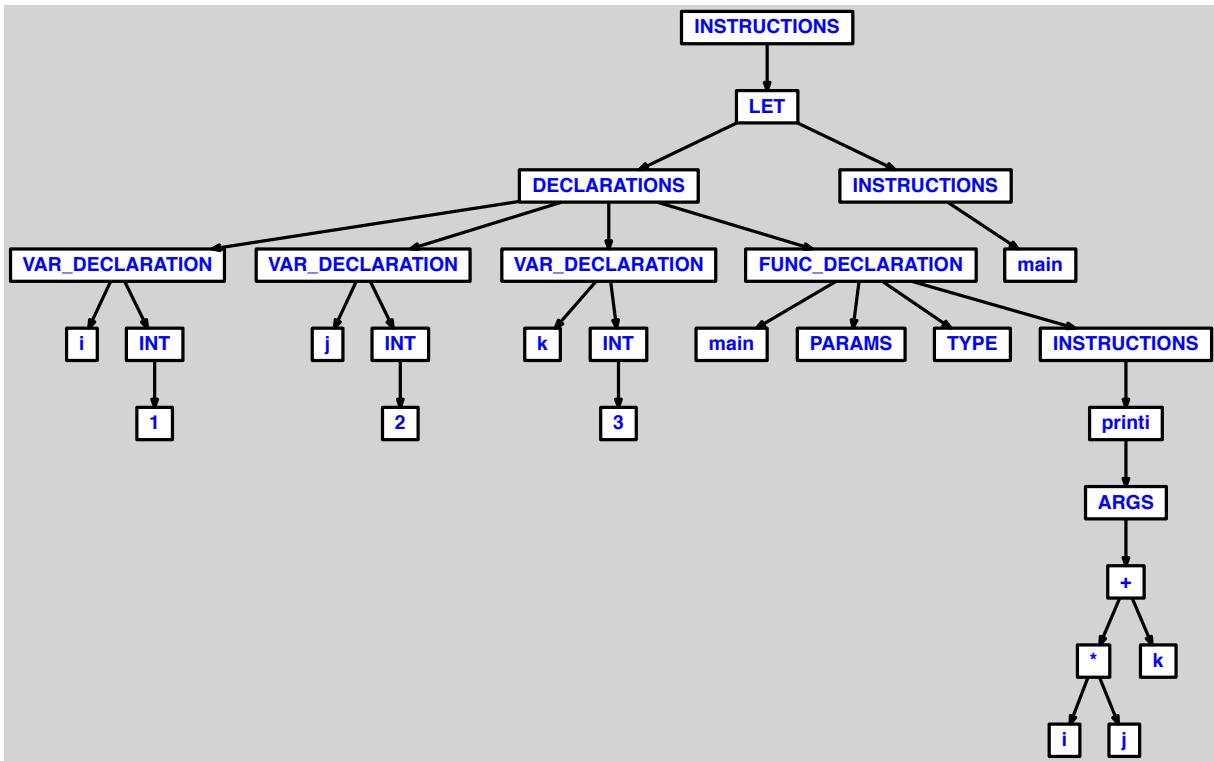
### 3.2.20 addition suivie de division, avec termes identifiés par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+j/k)
7 in main() end
```



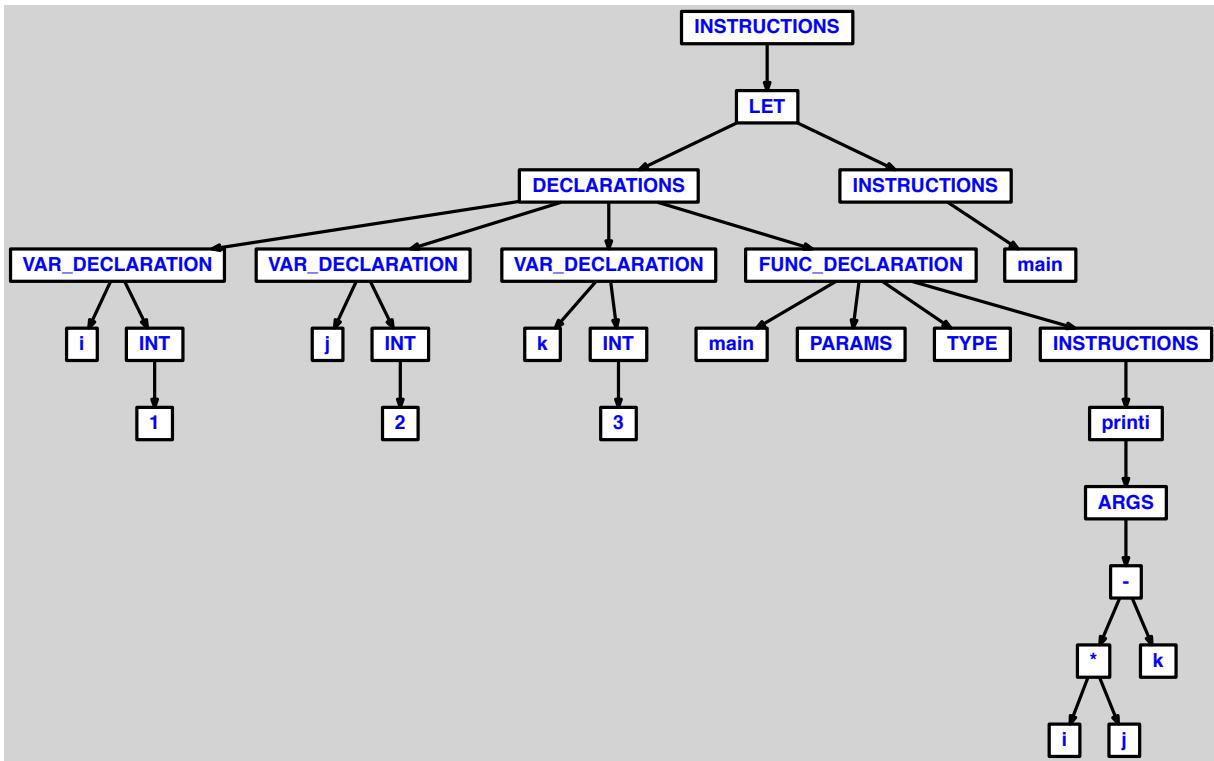
### 3.2.21 multiplication suivie d'addition, avec termes identifiés par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*j+k)
7 in main() end
```



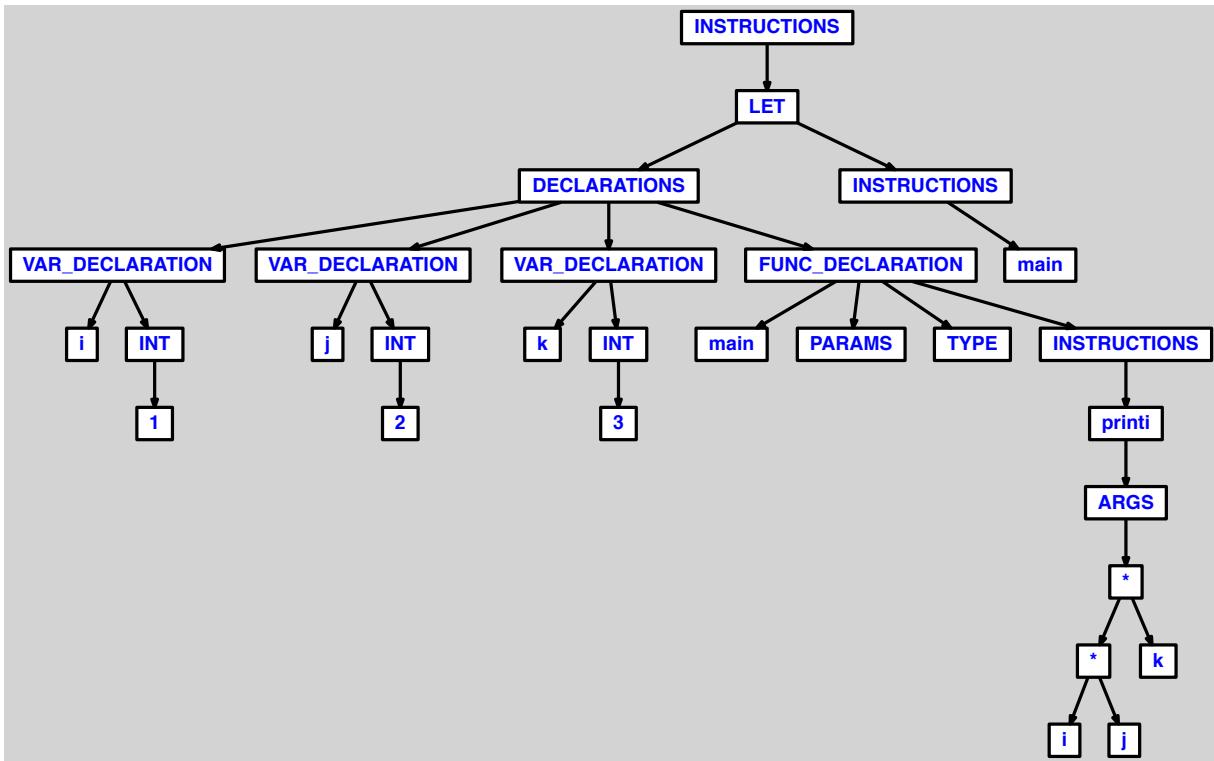
### 3.2.22 multiplication suivie de soustraction, avec termes identifiés par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*j-k)
7 in main() end
```



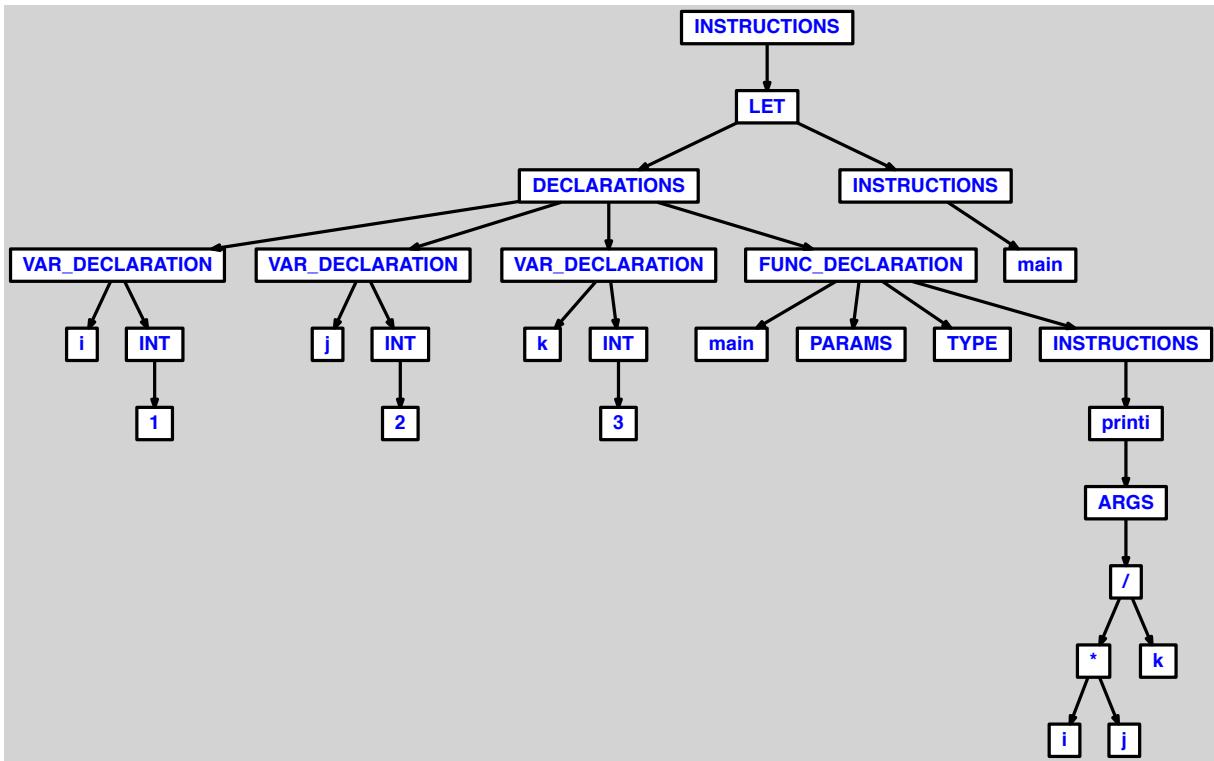
### 3.2.23 multiplication a 3 termes, identifies par des variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*j*k)
7 in main() end
```



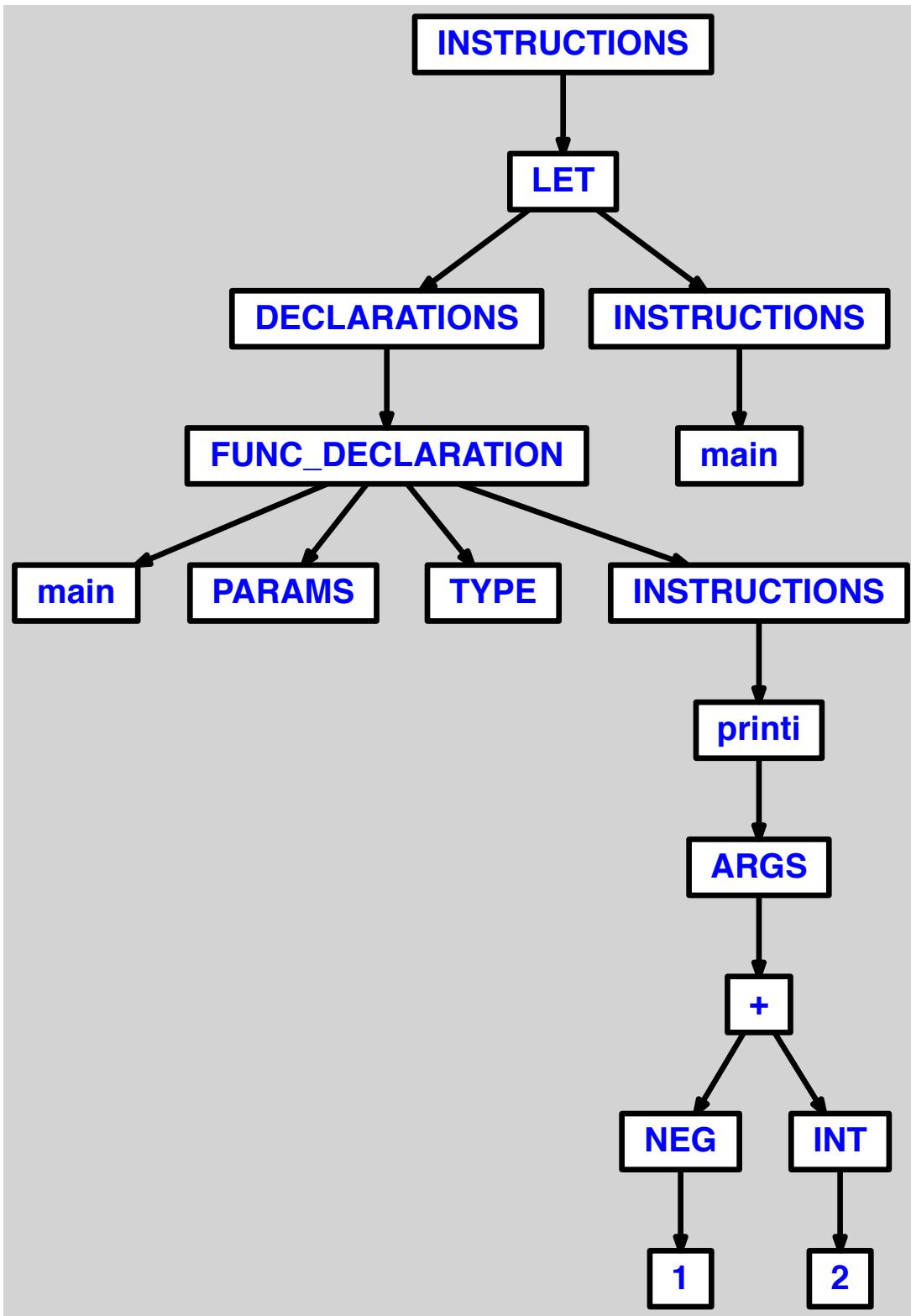
### 3.2.24 multiplication suivie de division, avec termes identifiés par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*j/k)
7 in main() end
```



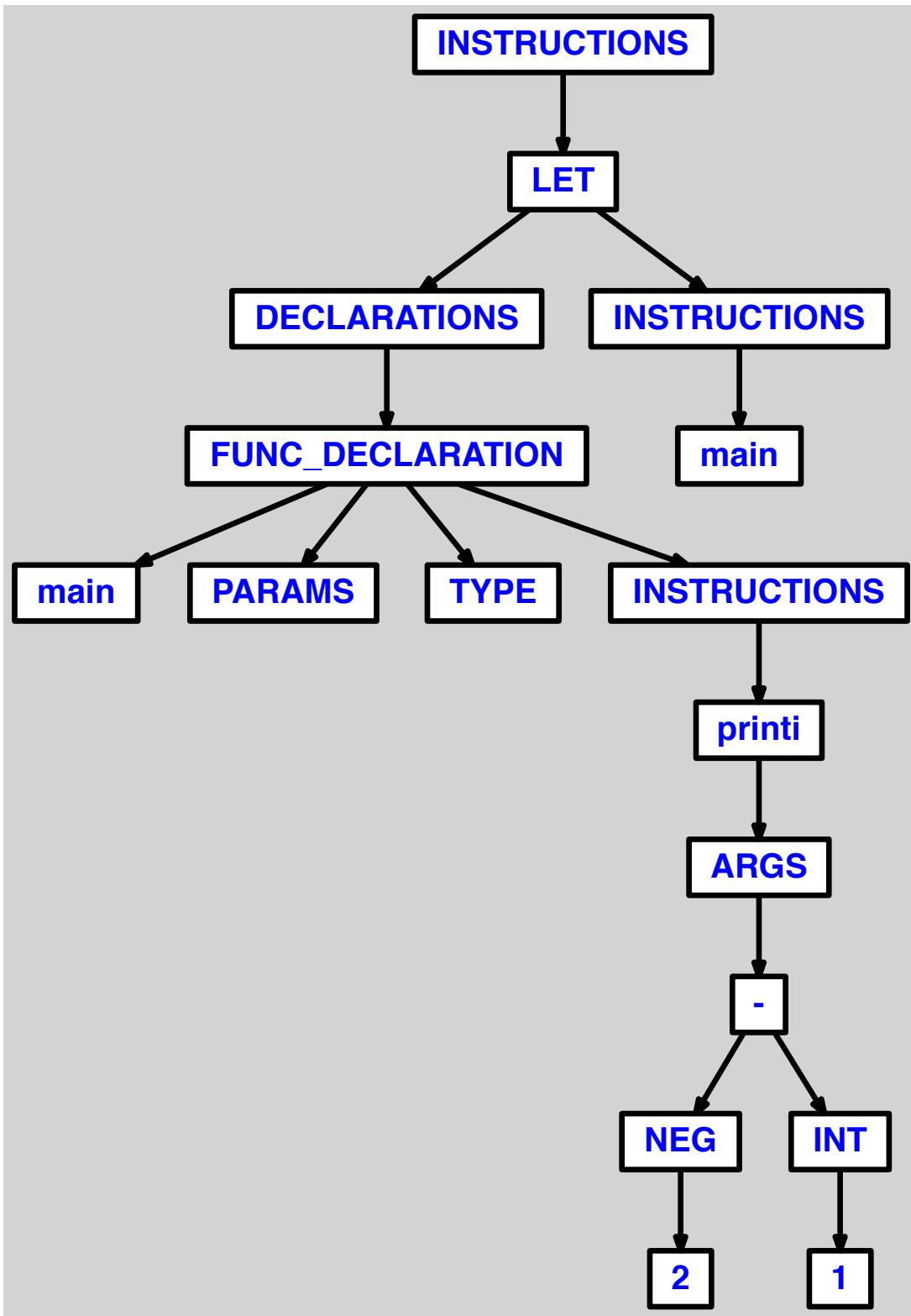
### 3.2.25 addition simple, a 2 termes, dont un ayant moins unaire

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



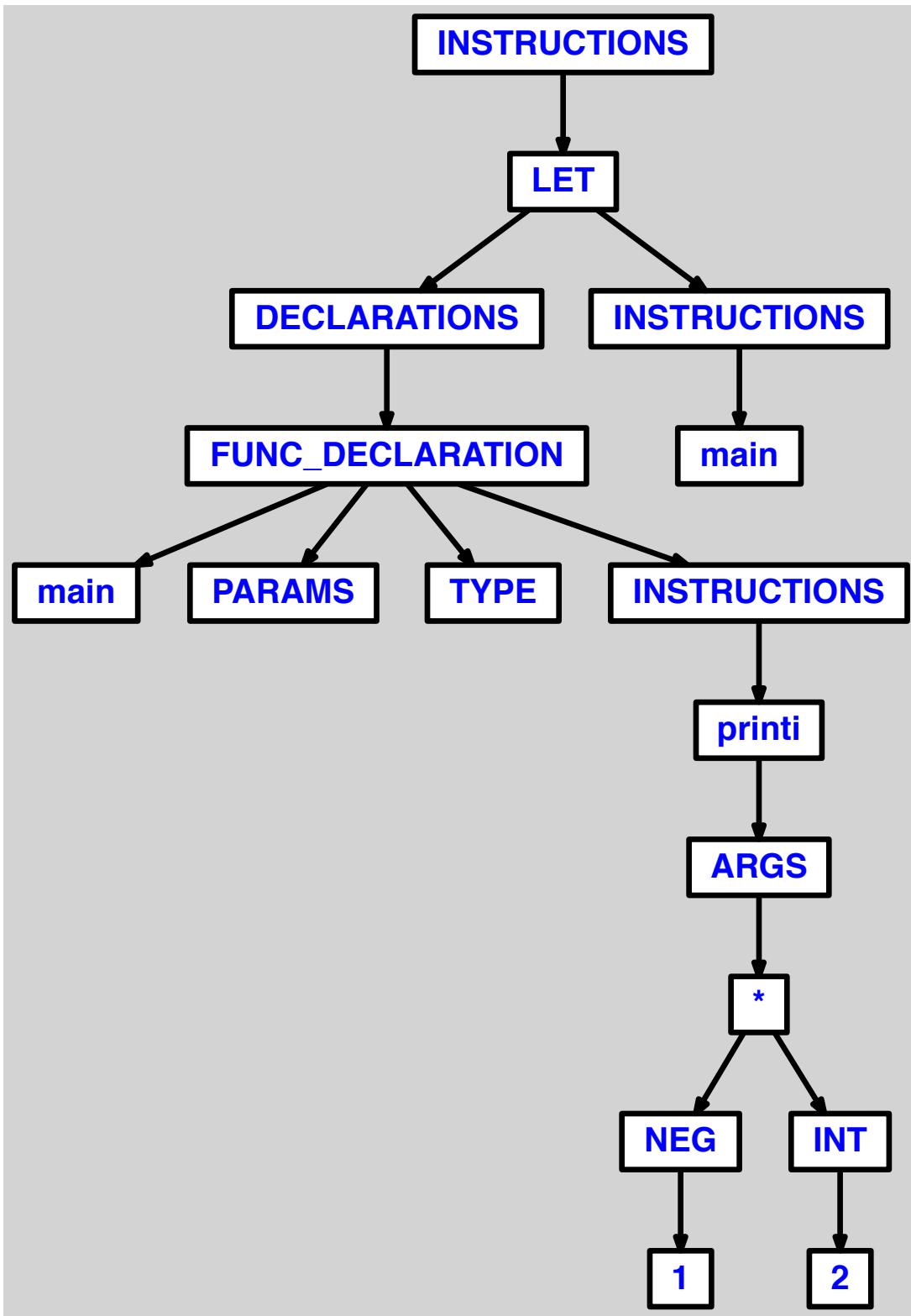
### 3.2.26 soustraction simple, a 2 termes, dont ayant moins unaire

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



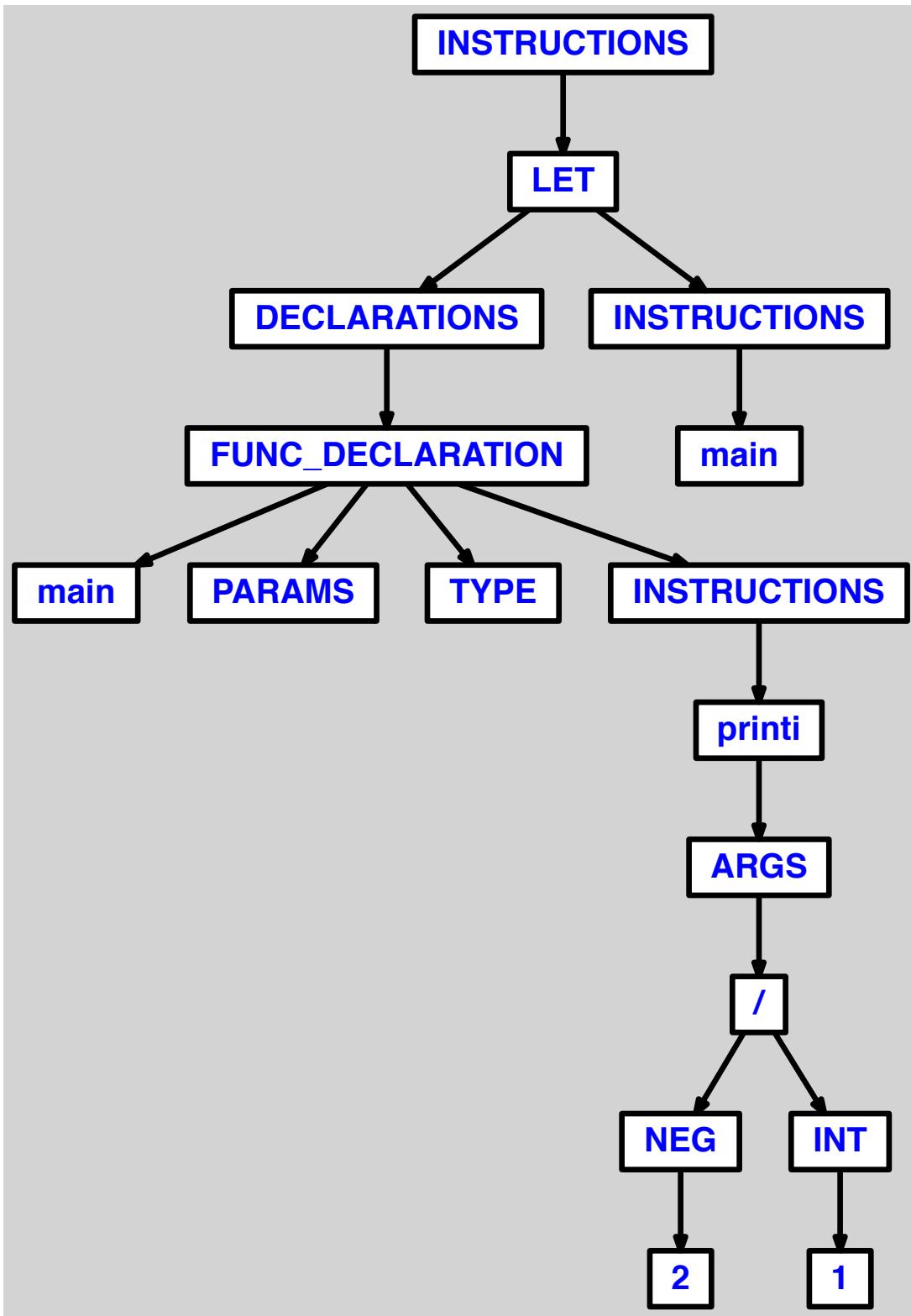
### 3.2.27 multiplication simple, a 2 termes, dont un ayant moins uneire

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



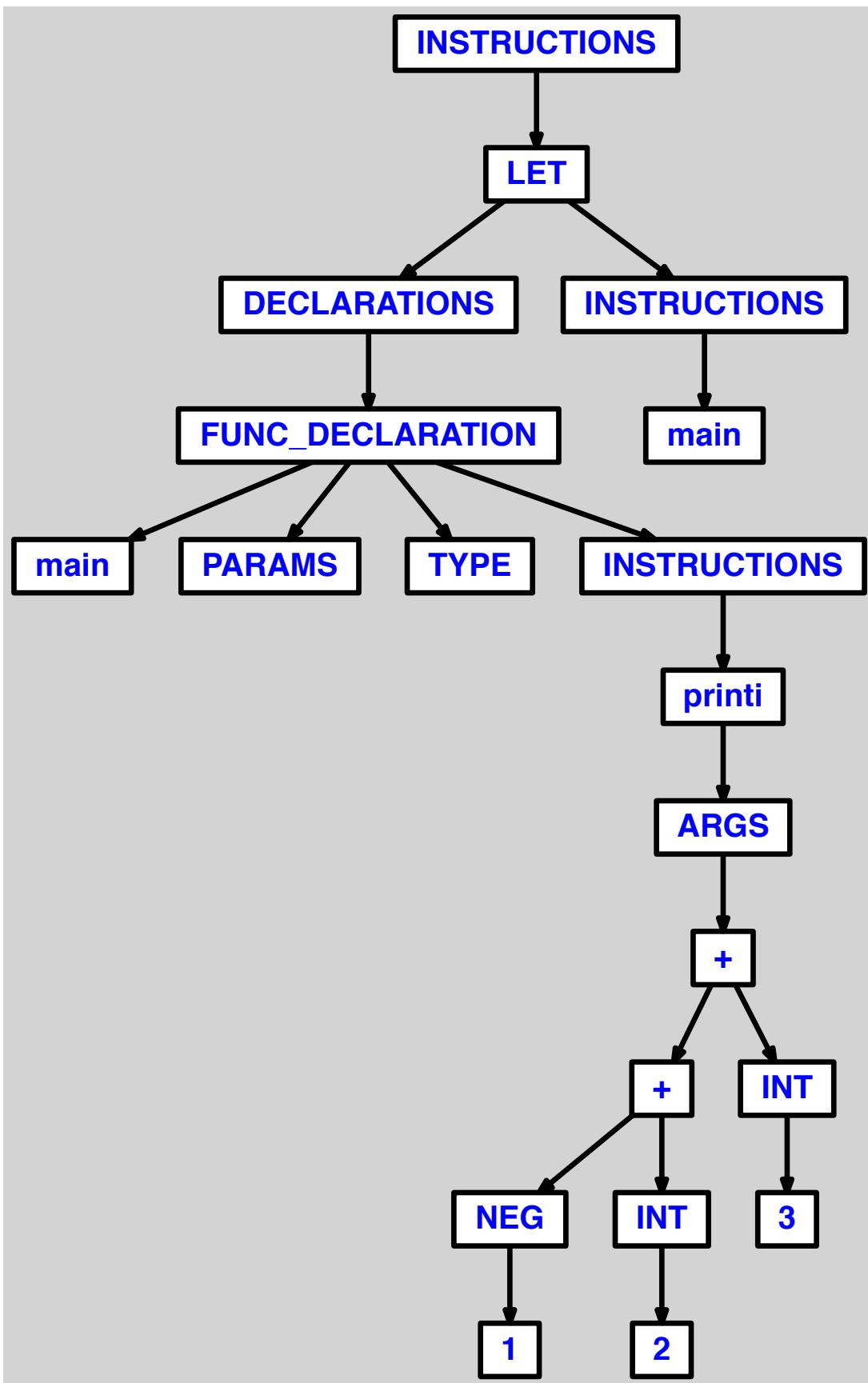
### 3.2.28 division simple, a 2 termes, dont un ayant moins unaire

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



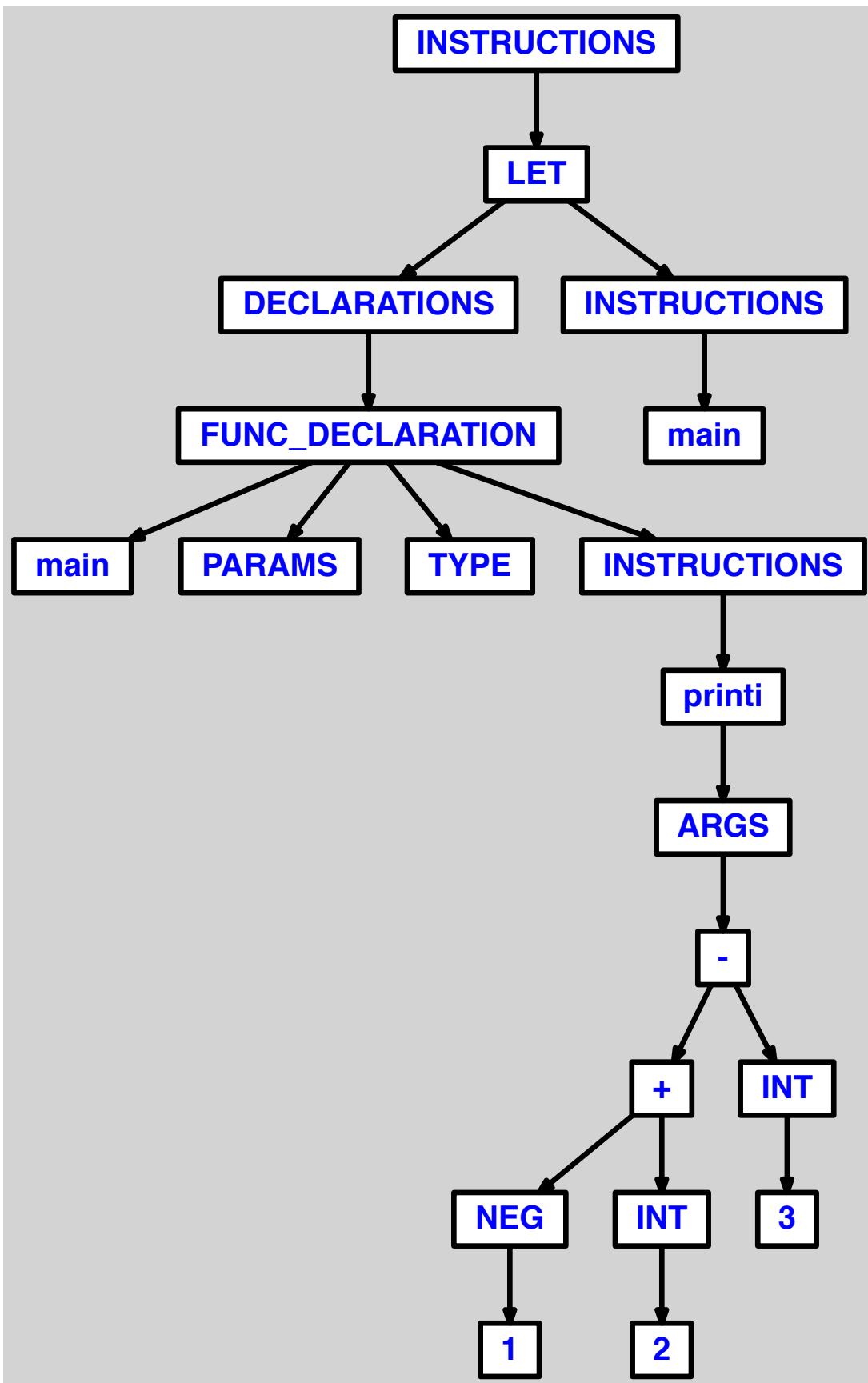
### 3.2.29 addition a 3 termes, dont un ayant moins unaire

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



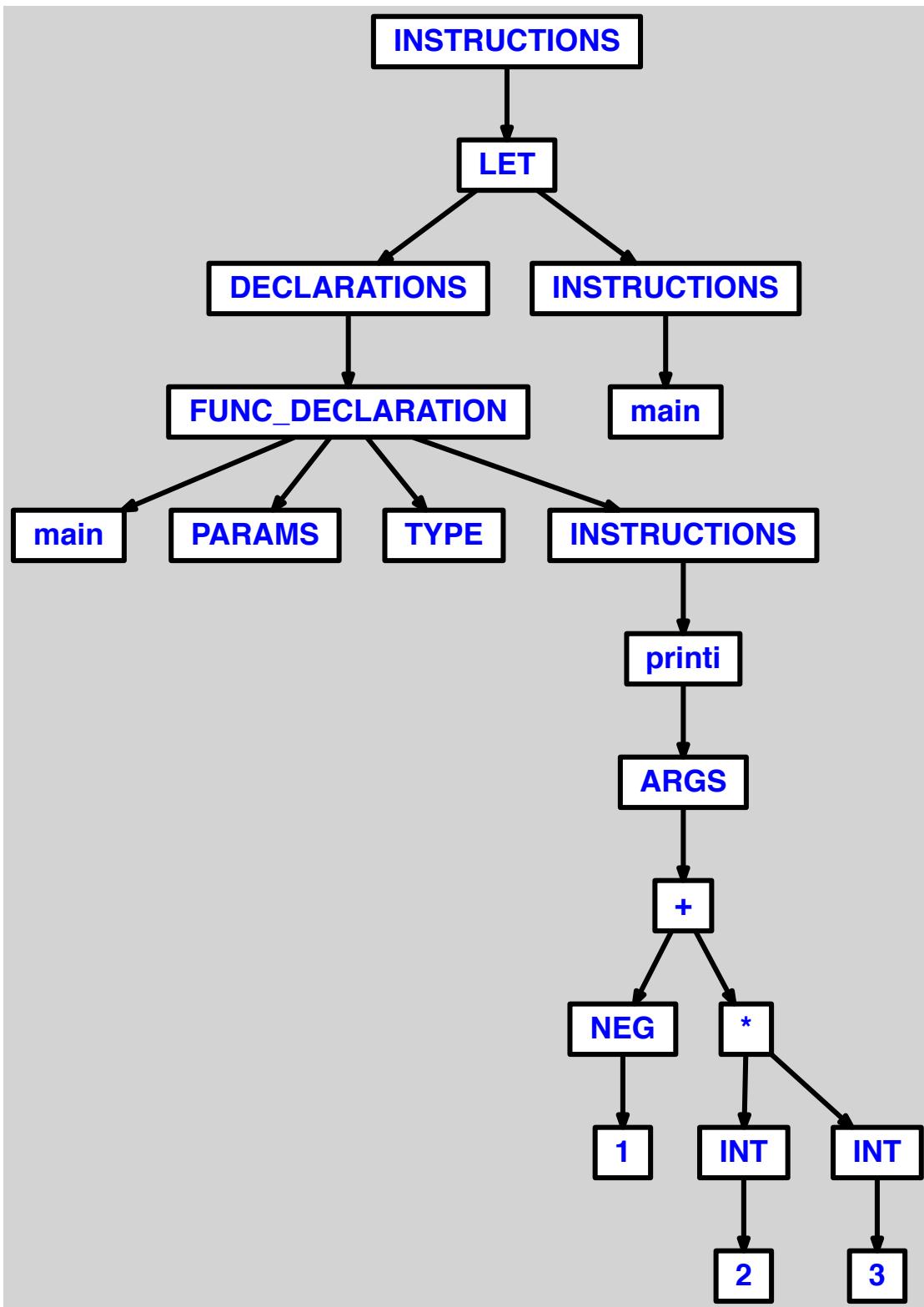
### 3.2.30 addition suivie de soustraction, avec un terme ayant moins unaire

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



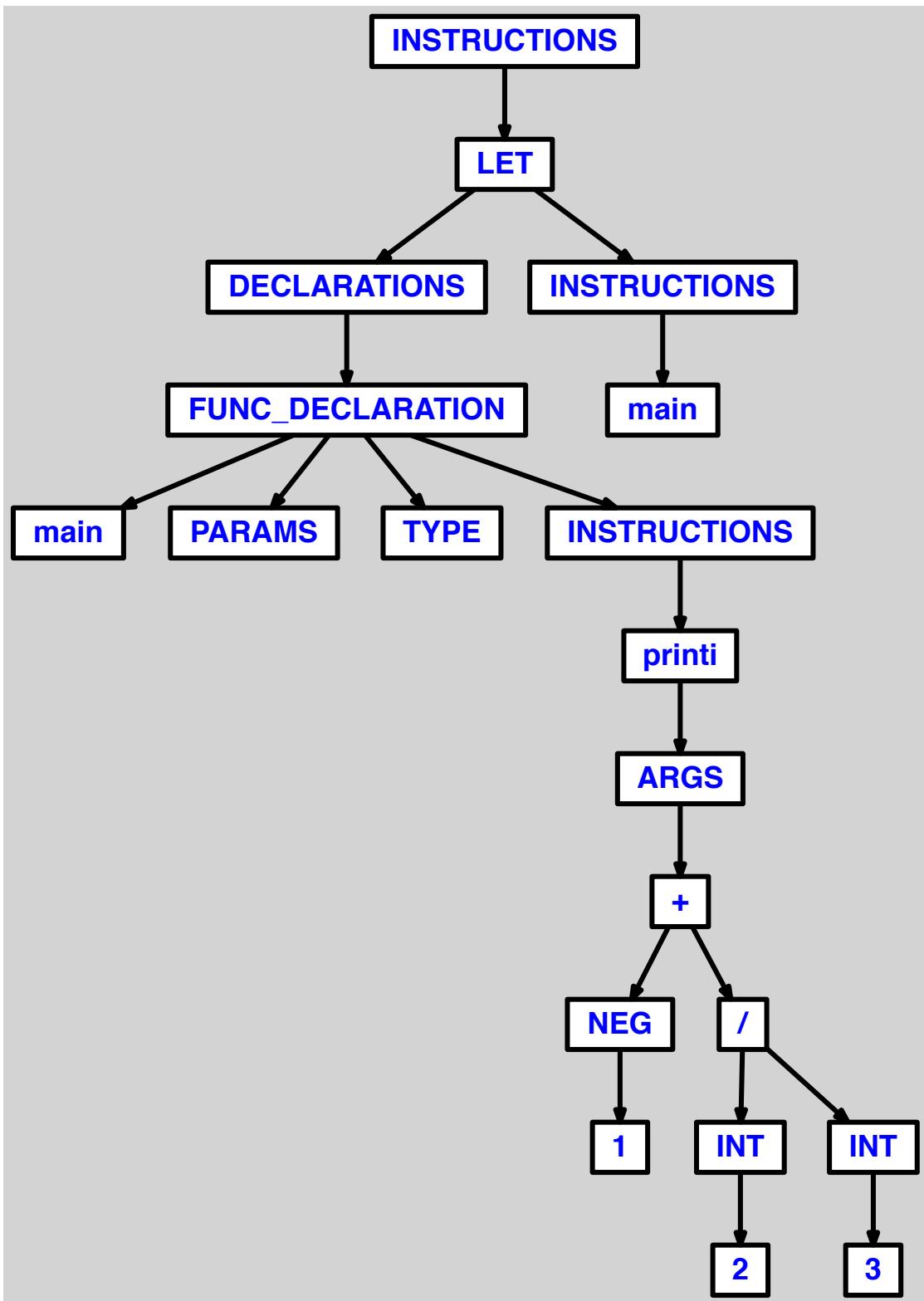
### 3.2.31 addition suivie de multiplication, avec un terme ayant moins unaire

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



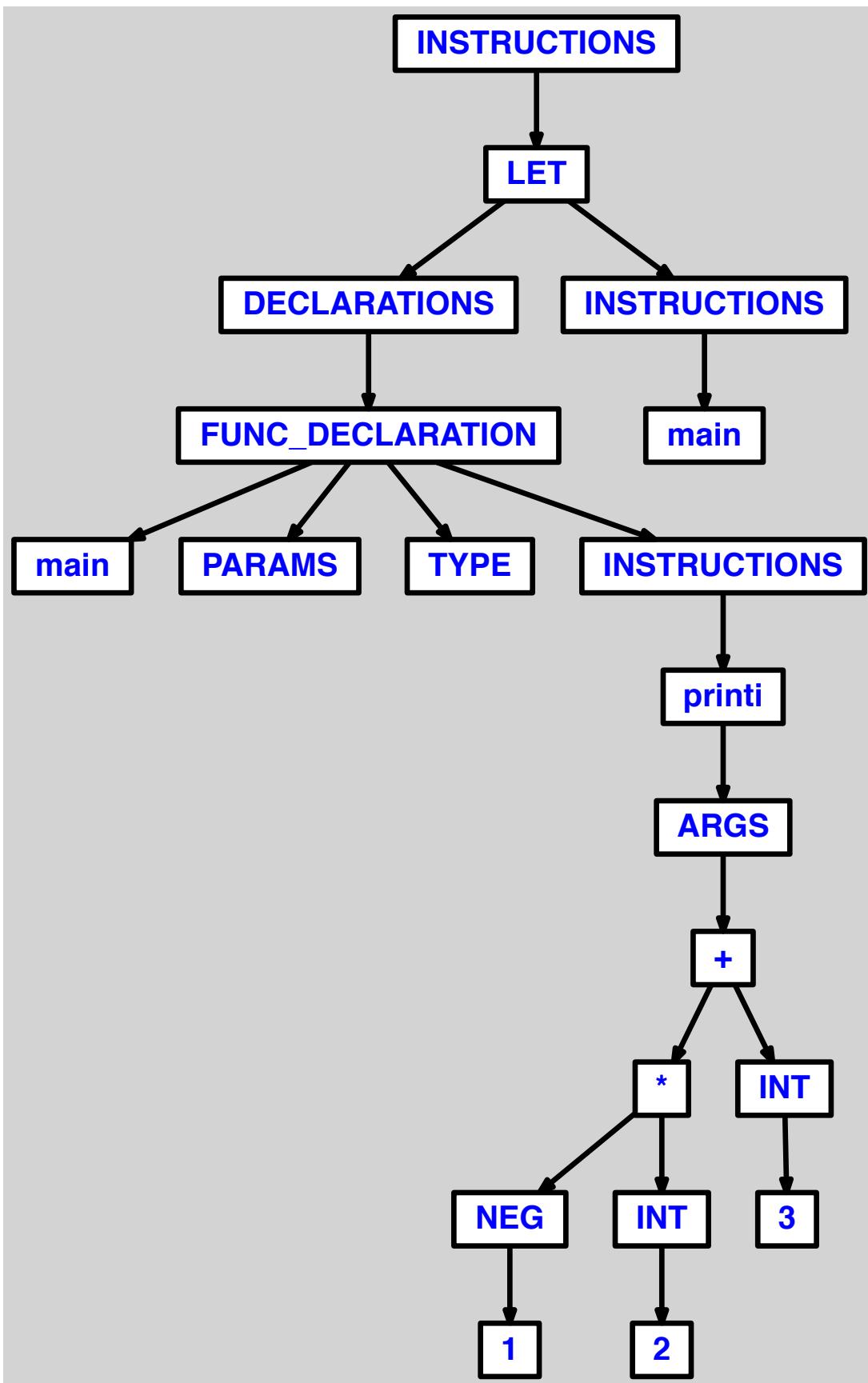
### 3.2.32 addition suivie de division, avec un terme ayant moins unaire

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



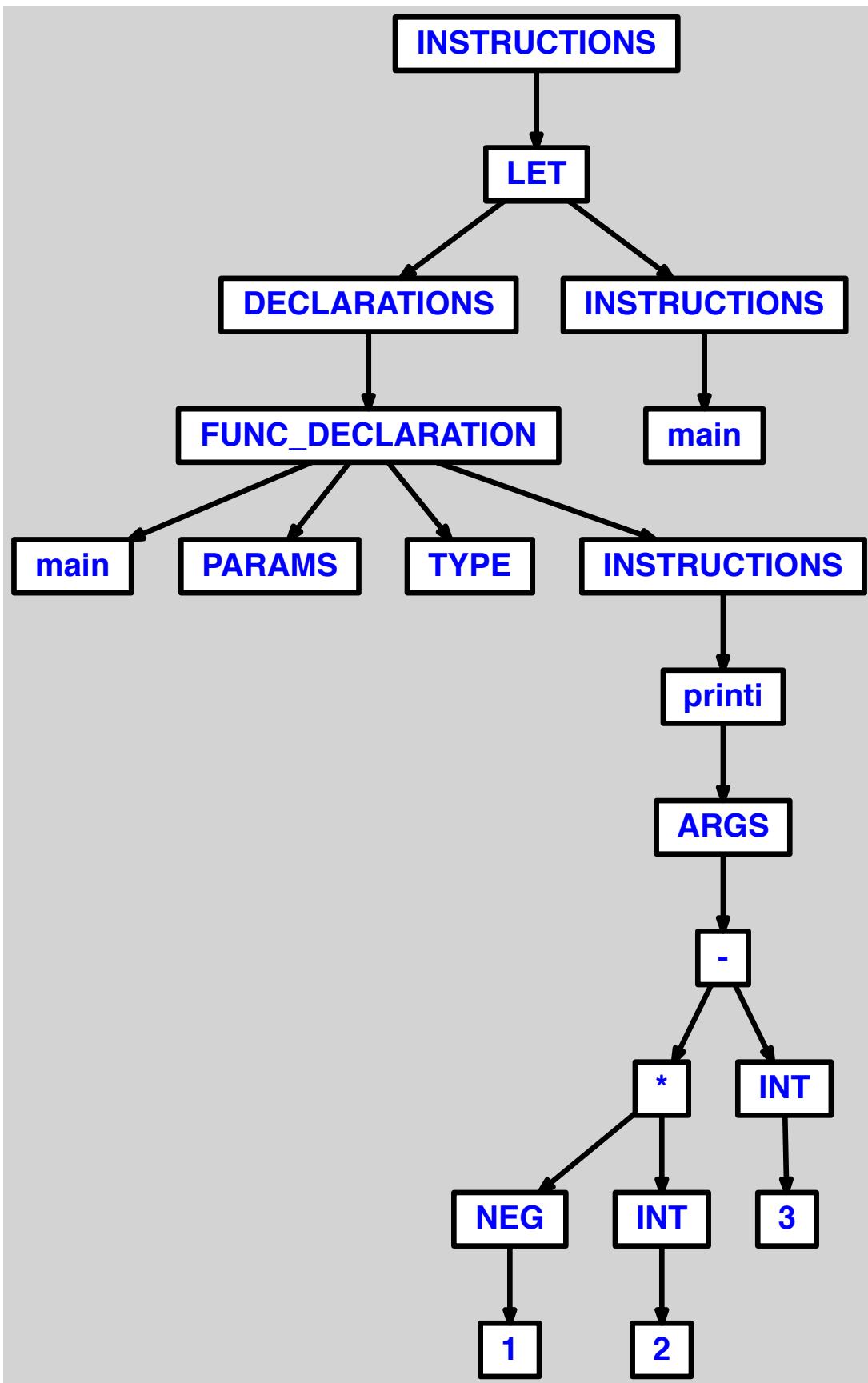
### 3.2.33 multiplication suivie d'addition, dont un terme ayant moins unaire

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



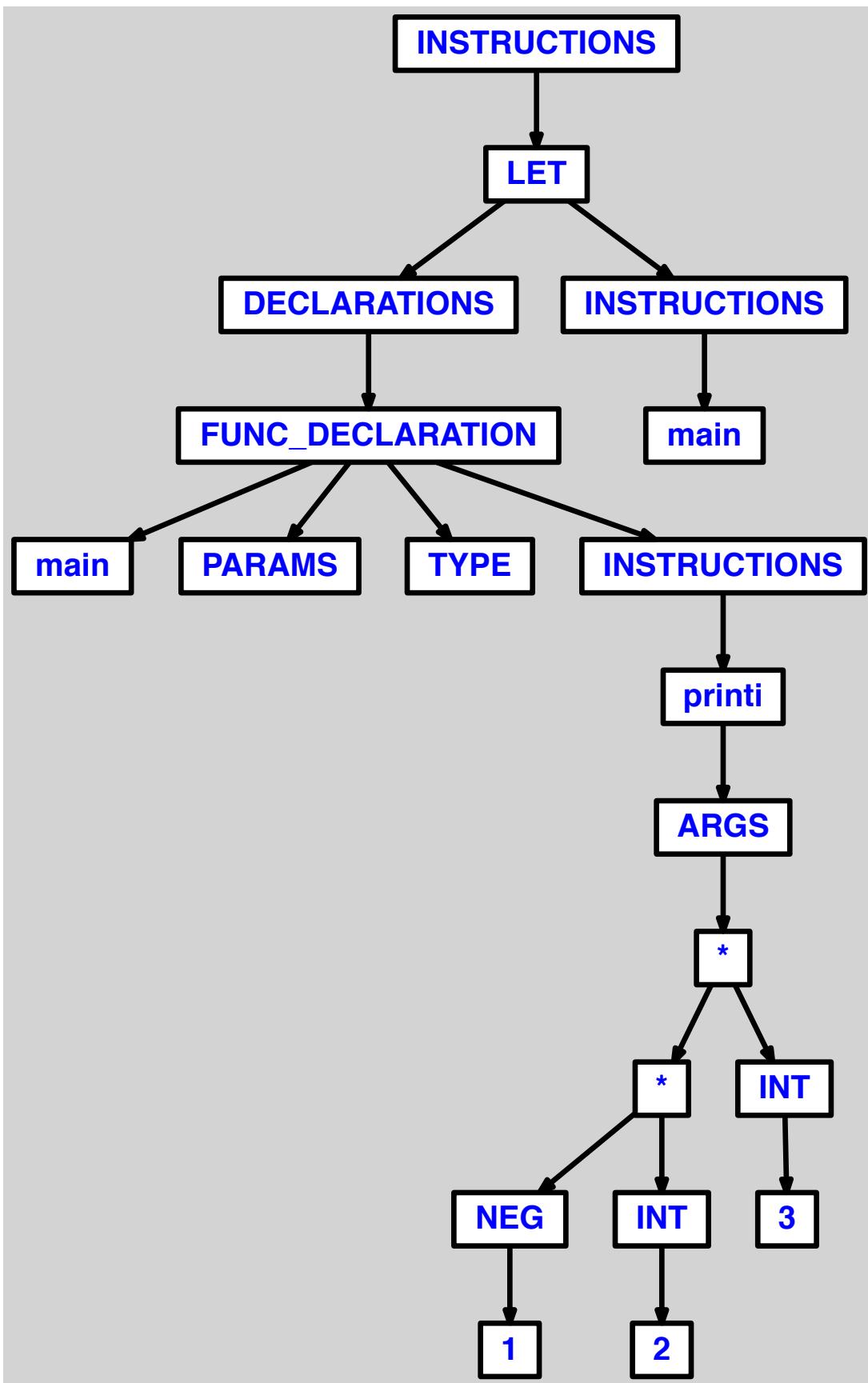
### 3.2.34 multiplication suivie de soustraction, dont un terme ayant moins unaire

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



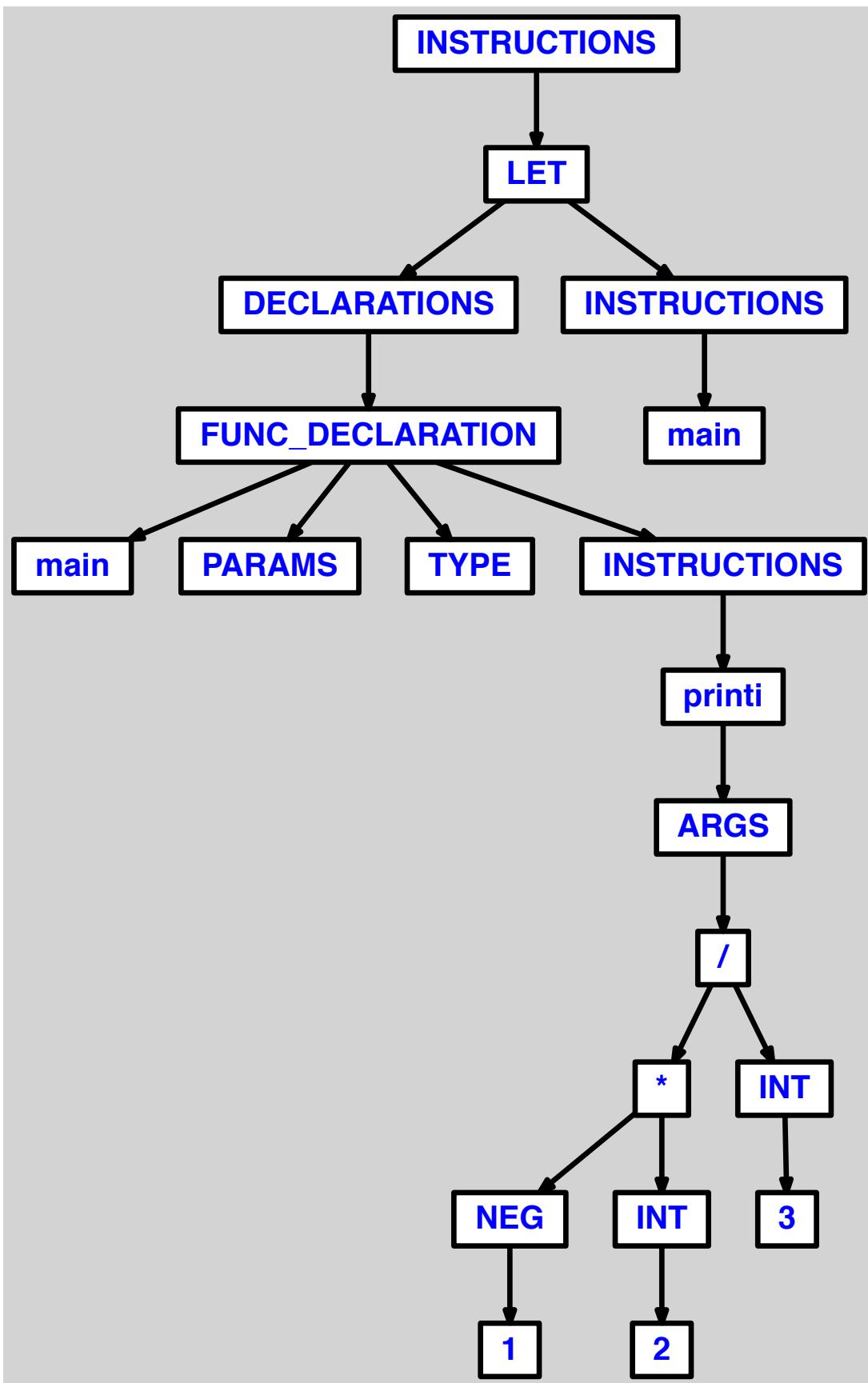
### 3.2.35 multiplication a 3 termes, dont un ayant moins unaire

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



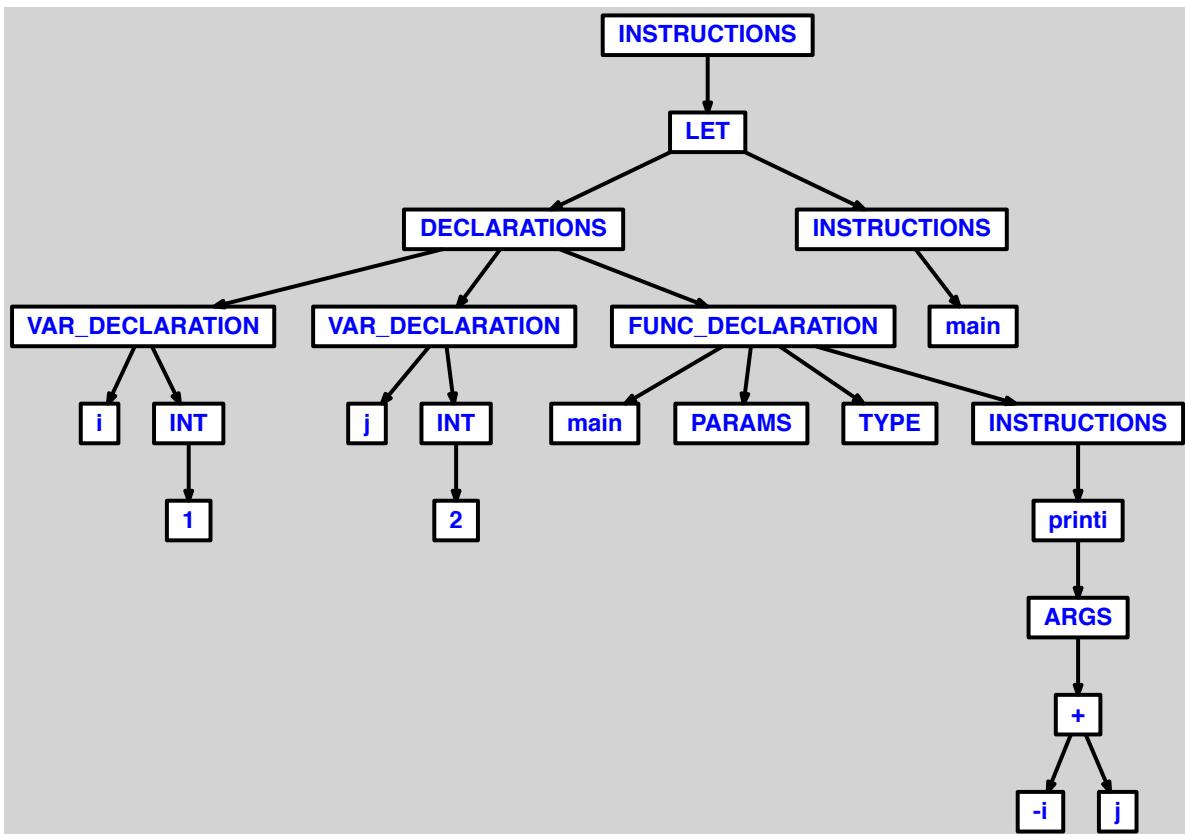
### 3.2.36 multiplication suivie de division, dont un terme ayant moins unaire

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



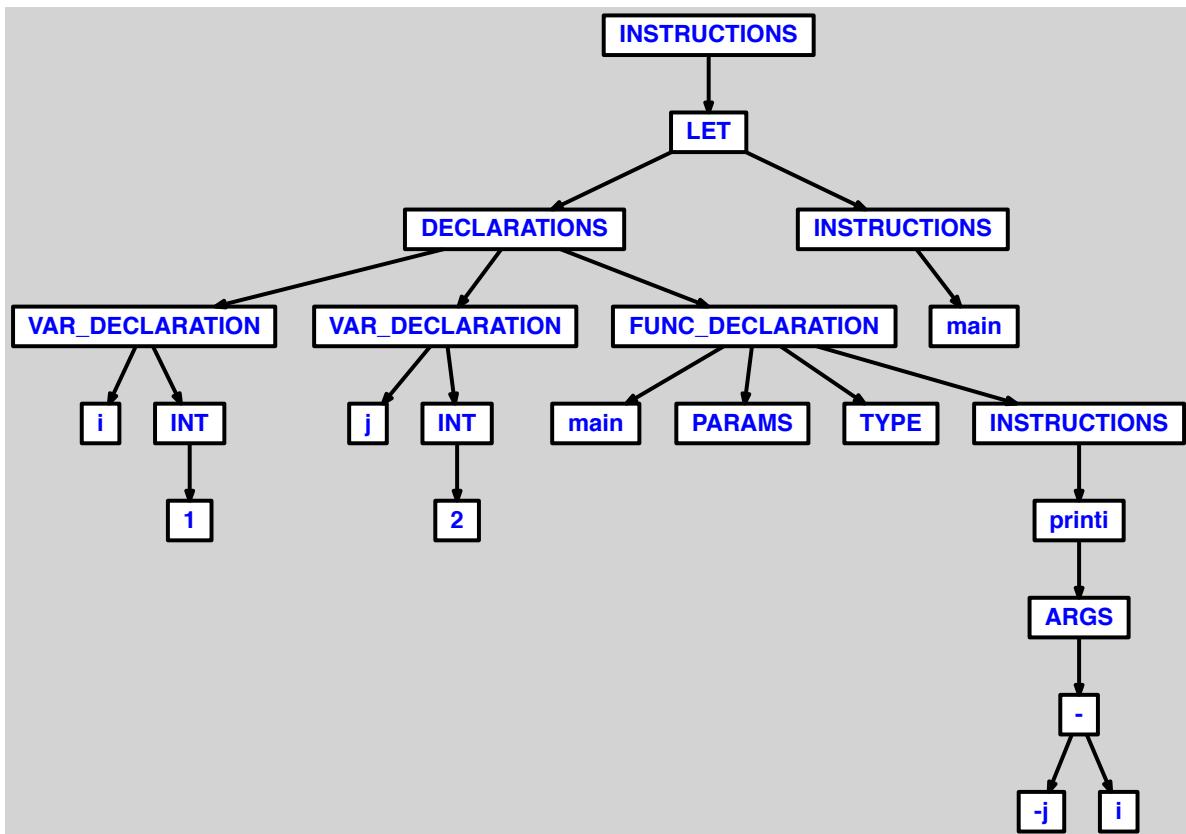
### 3.2.37 addition simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(-i+j)
6 in main() end
```



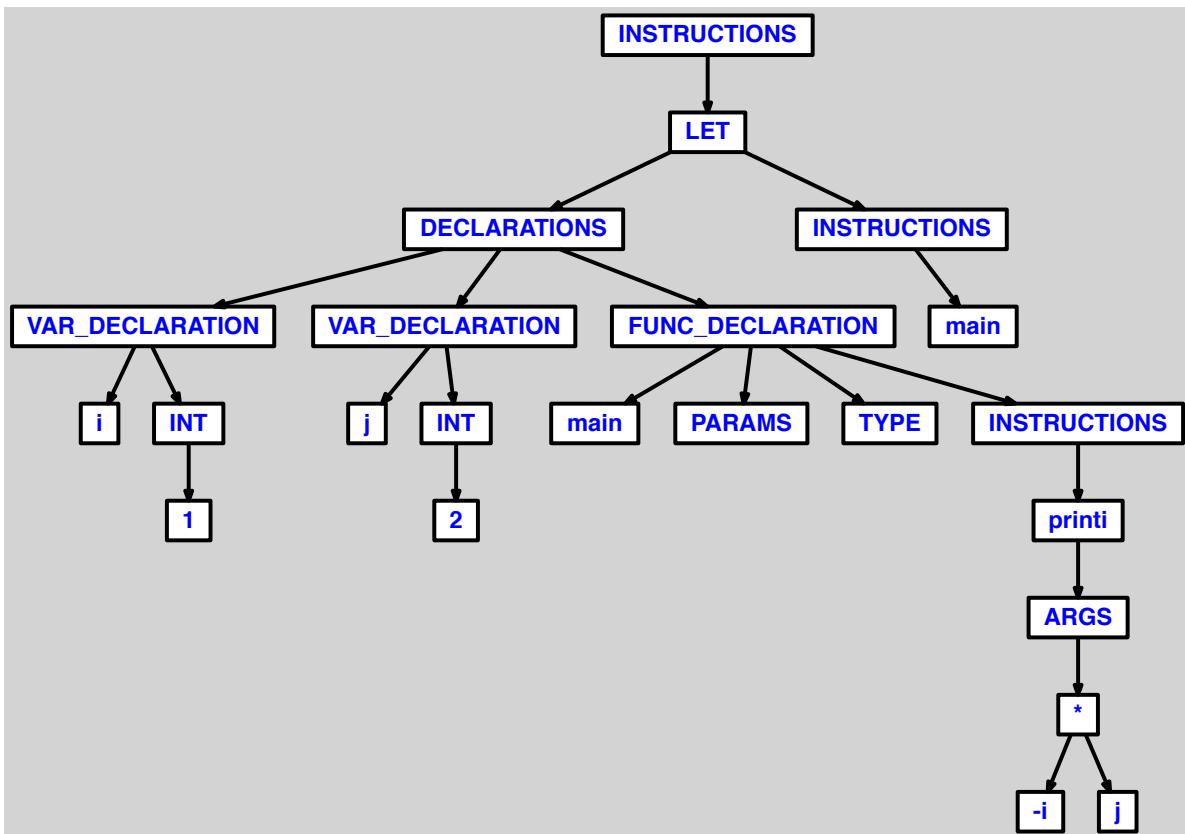
### 3.2.38 soustraction simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(-j-i)
6 in main() end
```



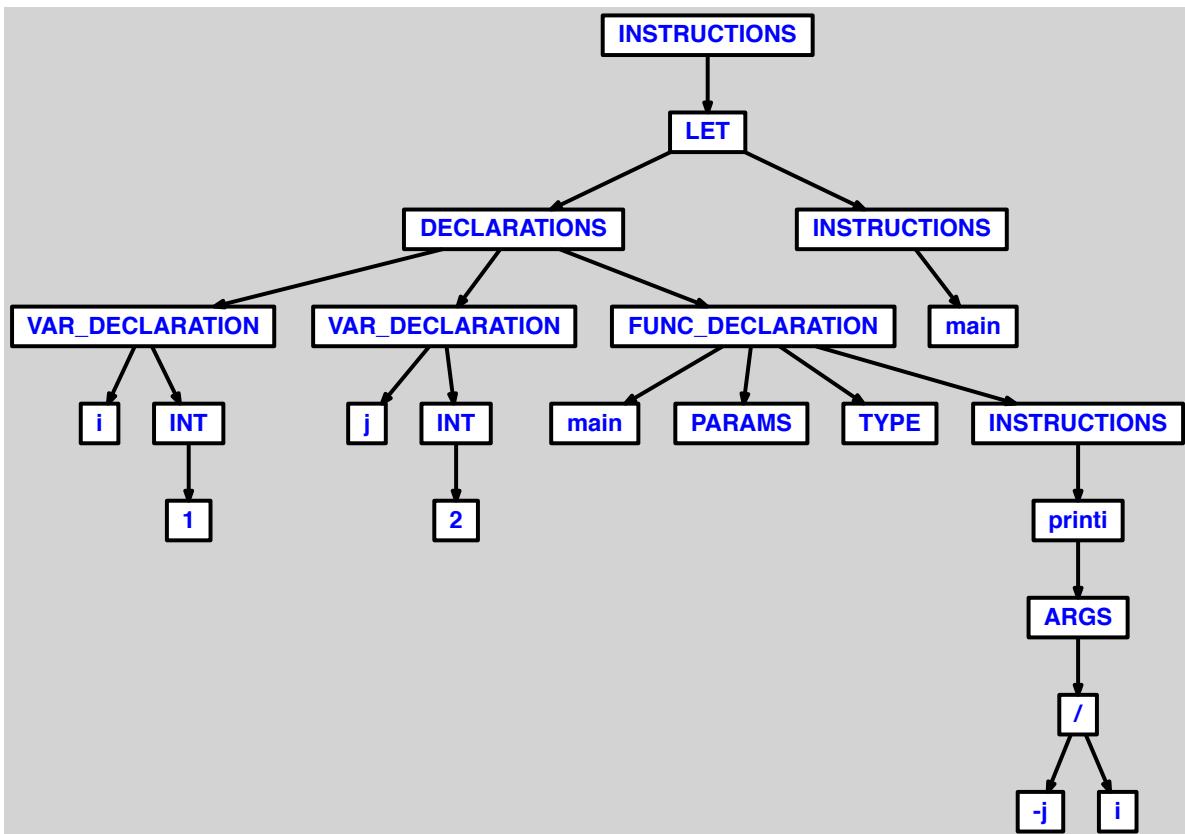
### 3.2.39 multiplication simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(-i*j)
6 in main() end
```



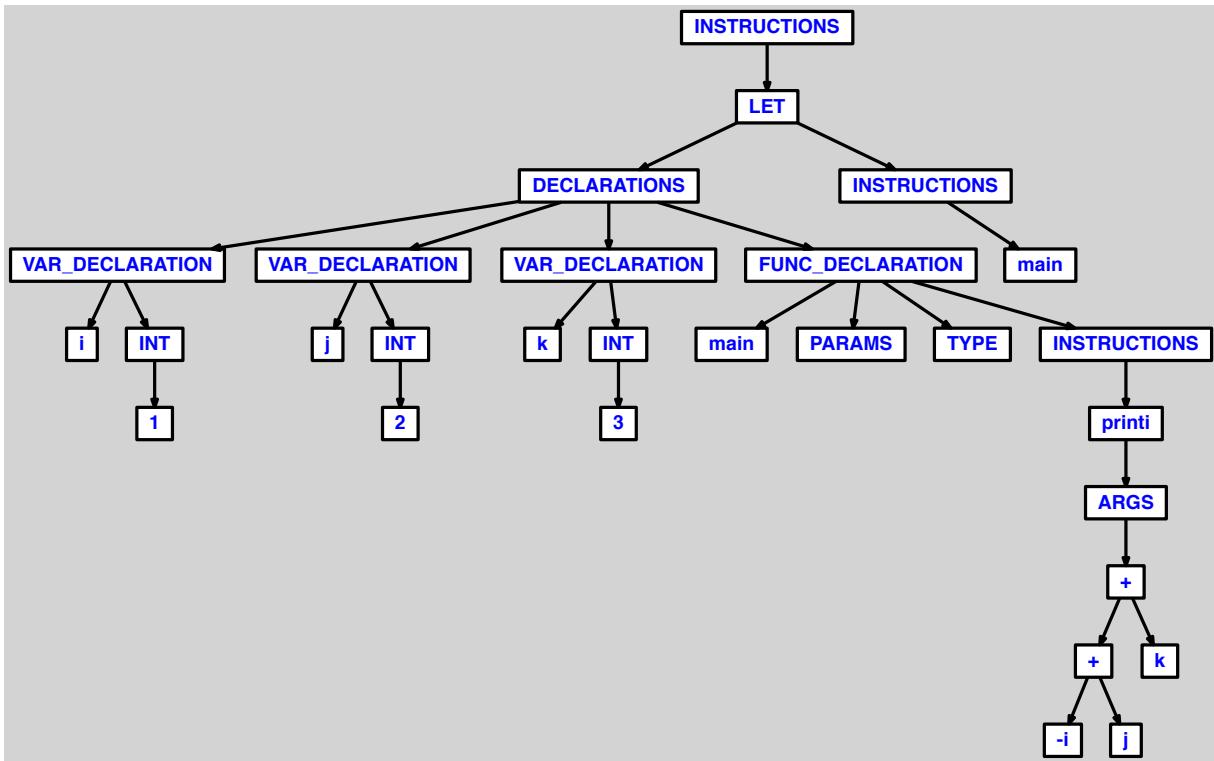
### 3.2.40 division simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4
5   function main() = printi(-j/i)
6 in main() end
```



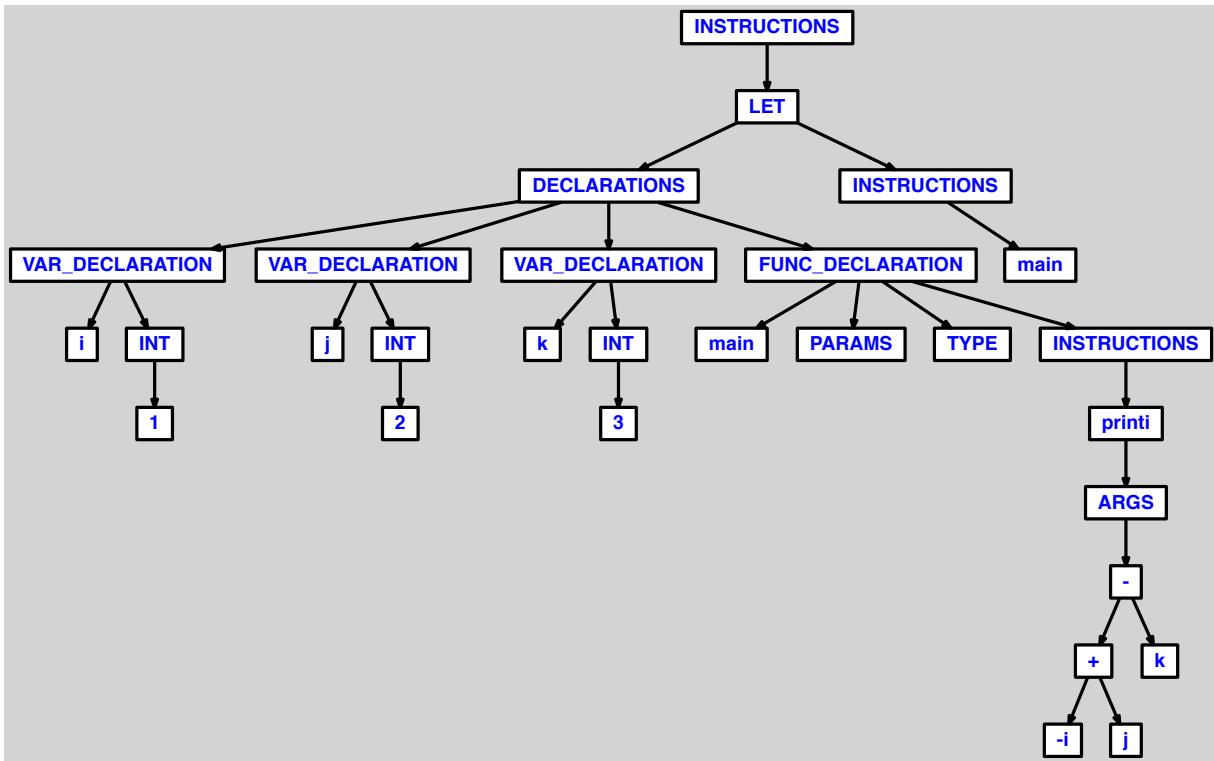
### 3.2.41 addition a 3 termes, identifies par des variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i+j+k)
7 in main() end
```



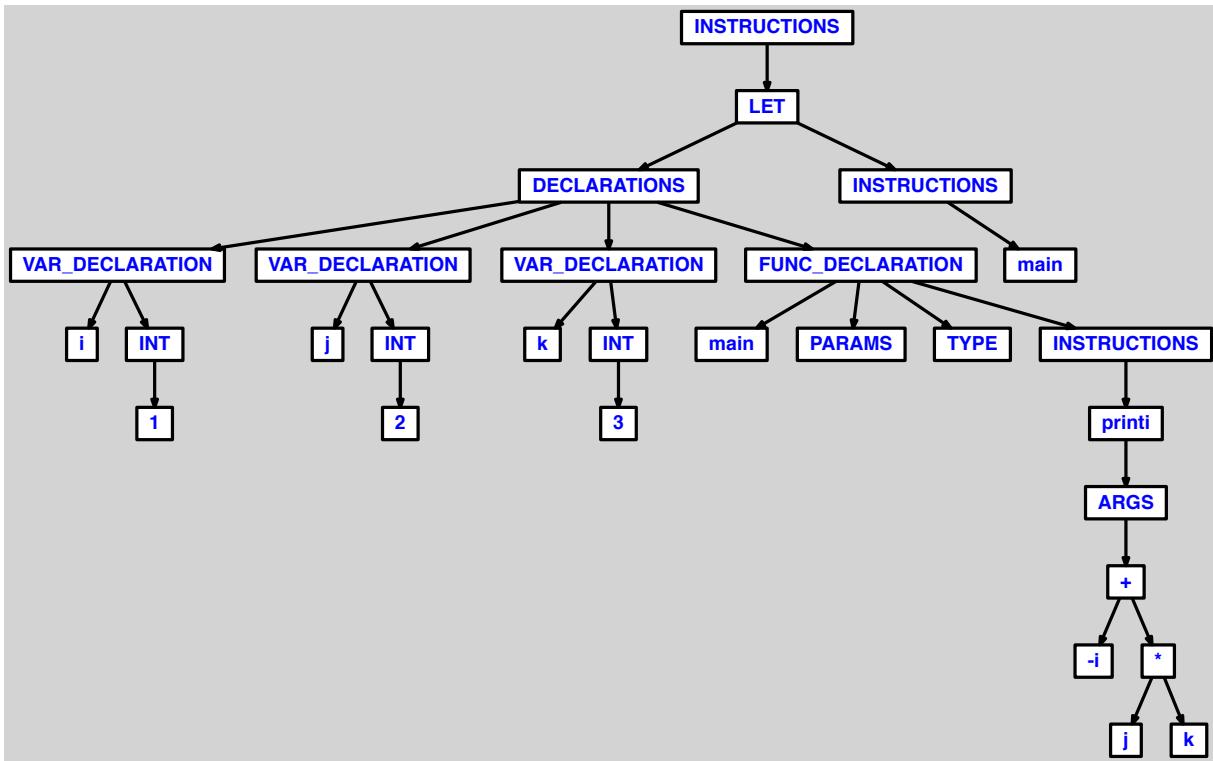
### 3.2.42 addition suivie de soustraction, avec termes identifiés par variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i+j-k)
7 in main() end
```



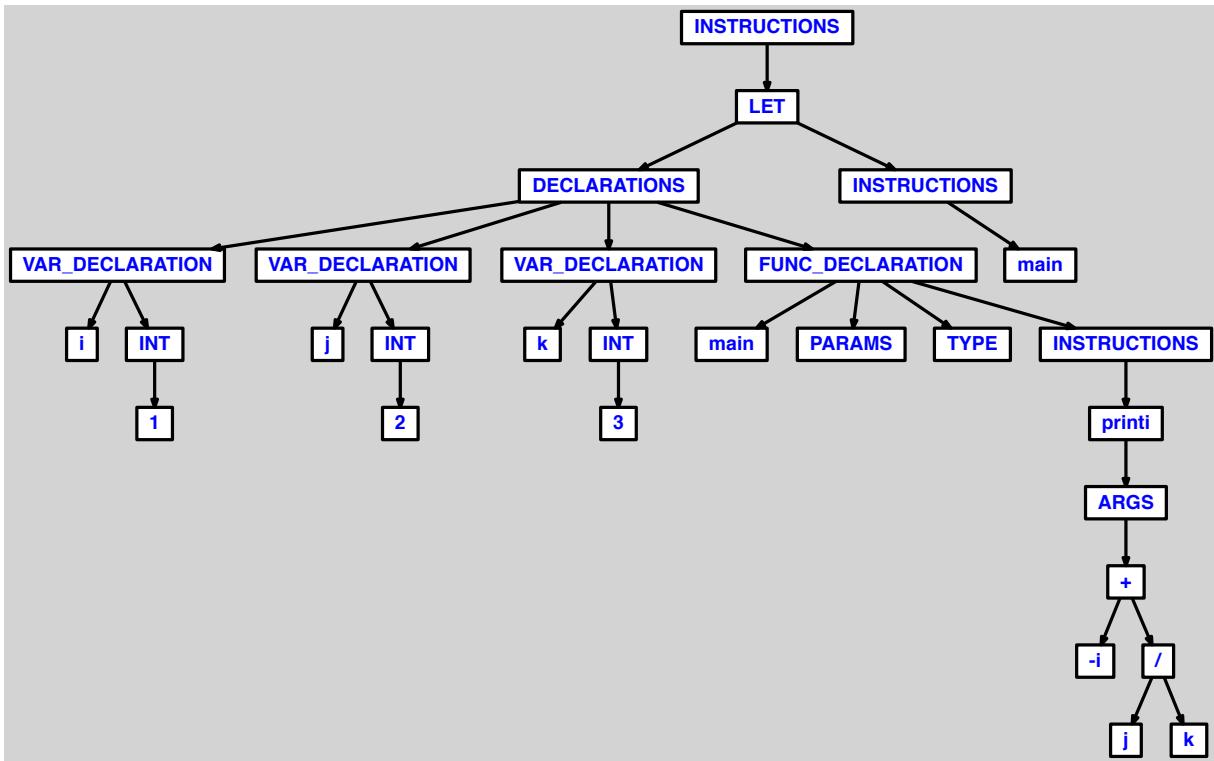
### 3.2.43 addition suivie de multiplication, avec termes identifies par variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i+j*k)
7 in main() end
```



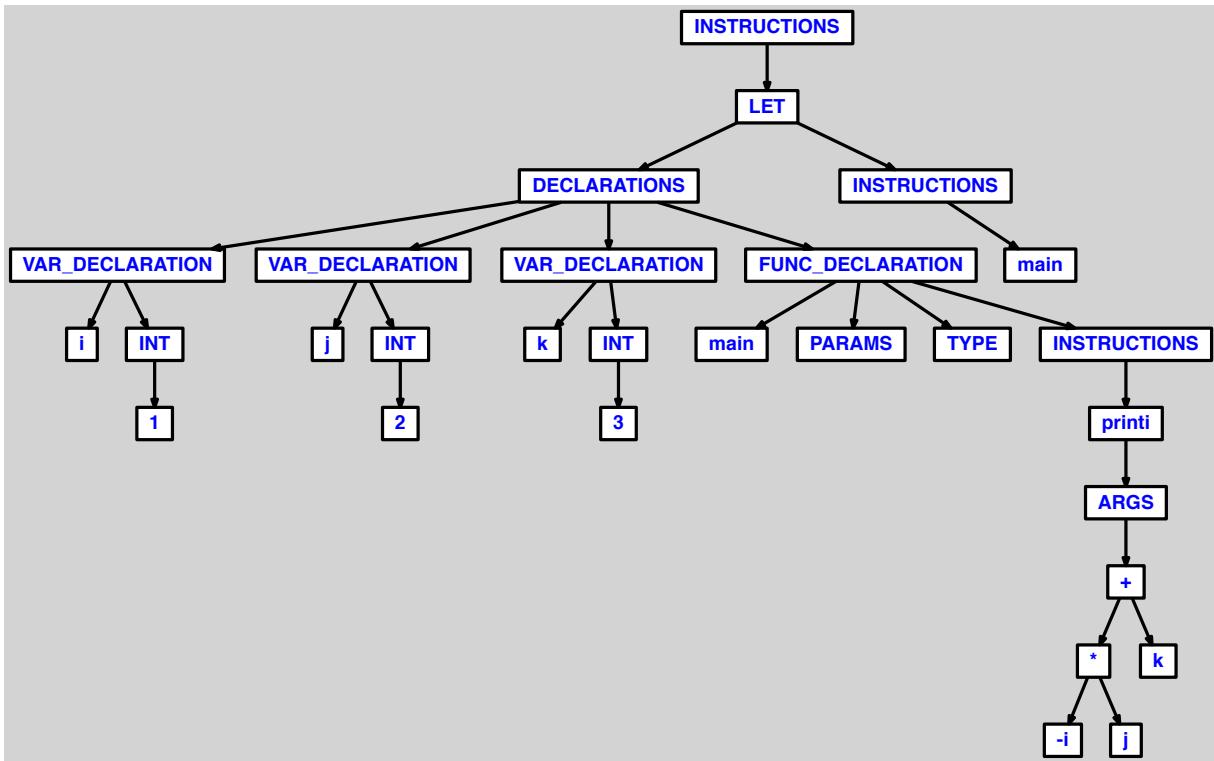
### 3.2.44 addition suivie de division, avec termes identifies par variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i+j/k)
7 in main() end
```



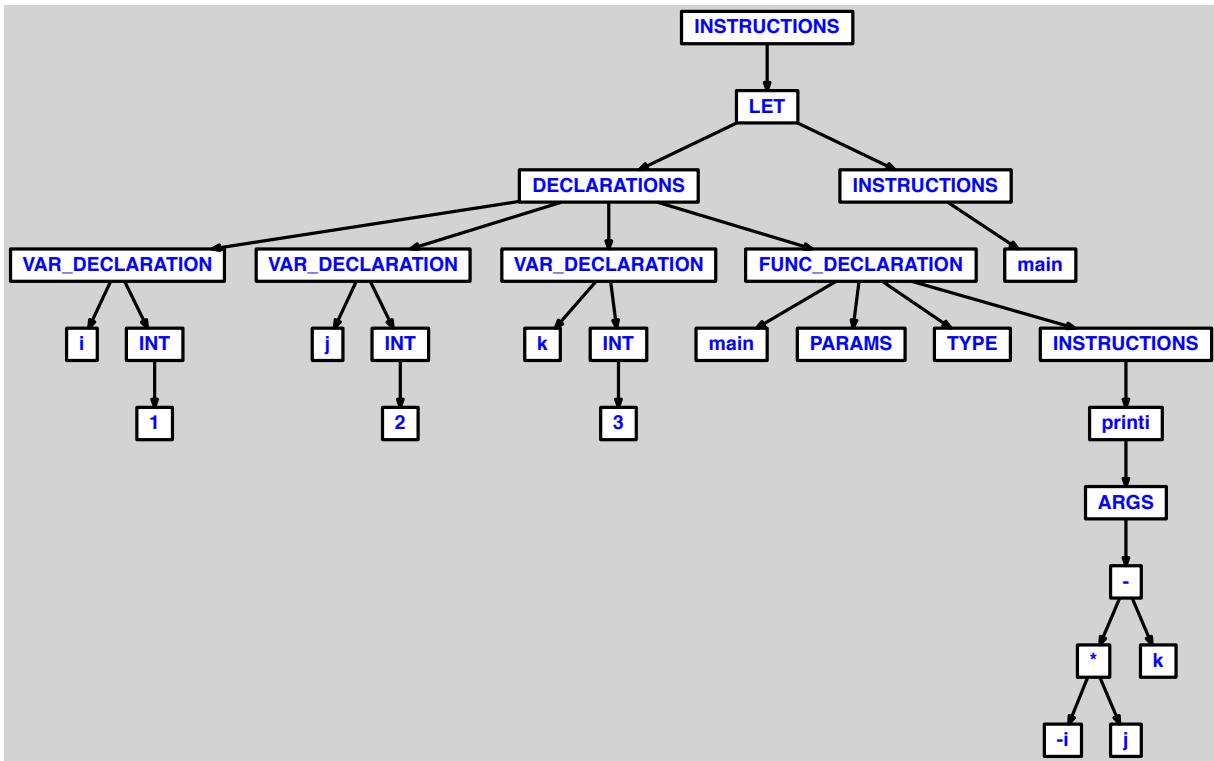
### 3.2.45 multiplication suivie d'addition, avec termes identifiés par variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i*j+k)
7 in main() end
```



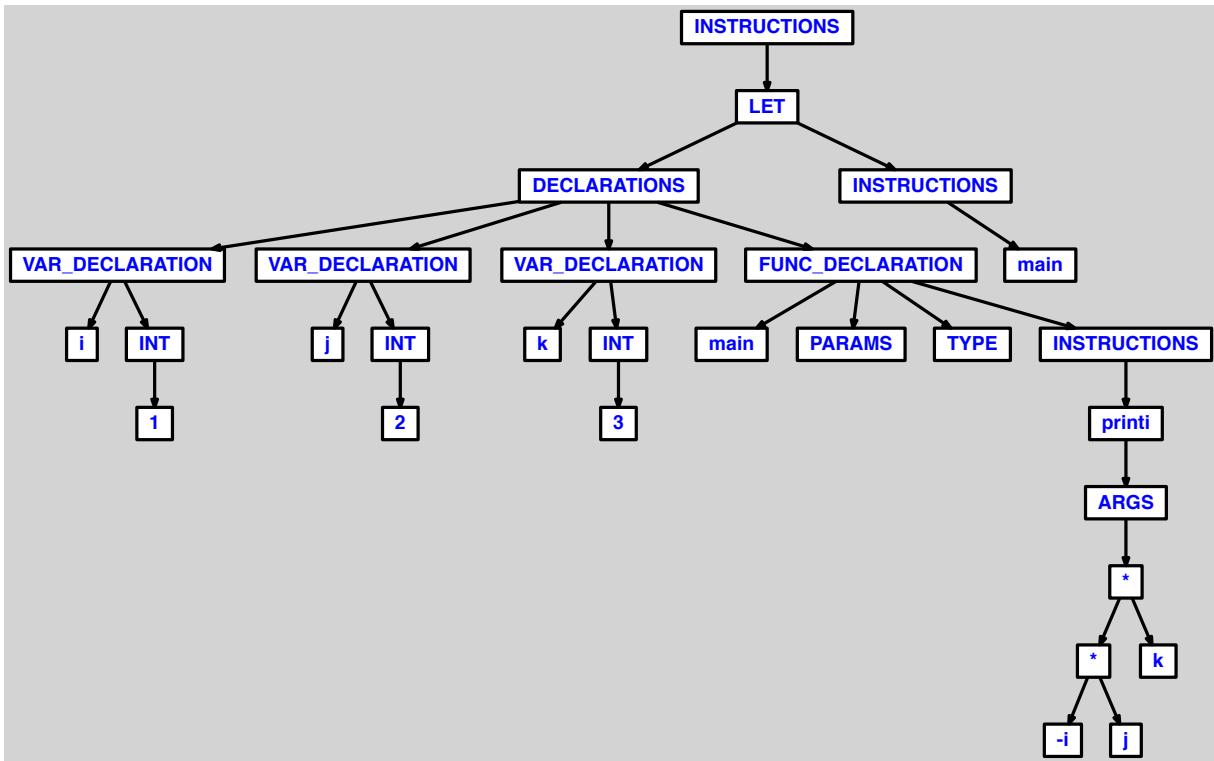
### 3.2.46 multiplication suivie de soustraction, avec termes identifies par variables, dont une ayant moins uneire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i*j-k)
7 in main() end
```



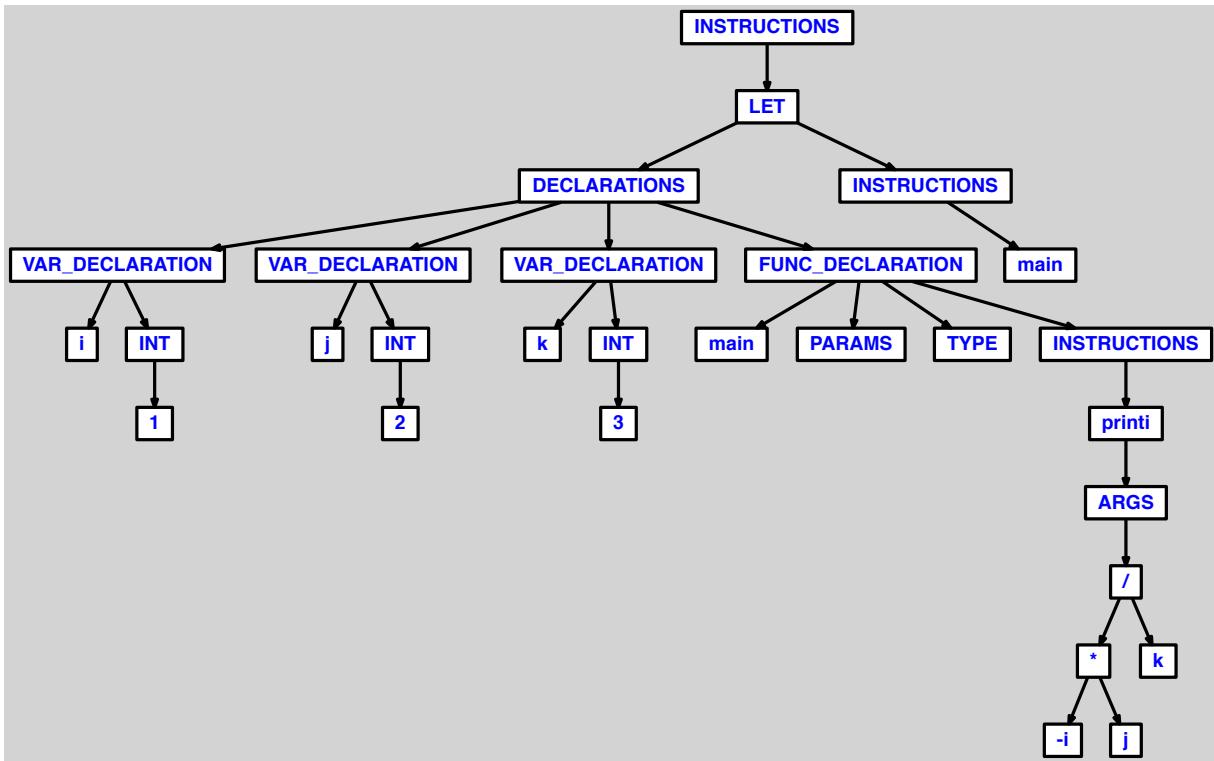
### 3.2.47 multiplication a 3 termes, identifies par des variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i*j*k)
7 in main() end
```



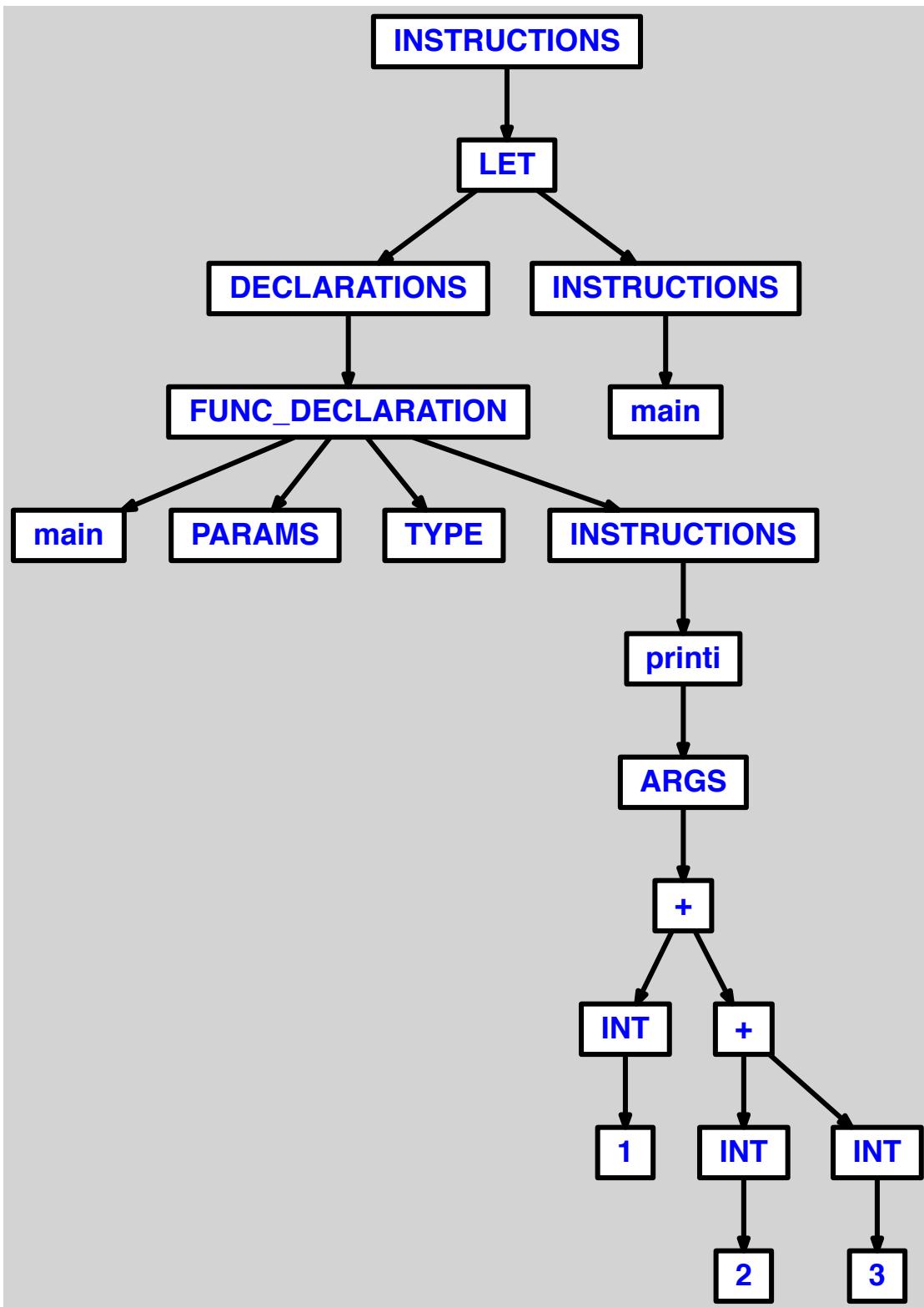
### 3.2.48 multiplication suivie de division, avec termes identifies par variables, dont une ayant moins unaire

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(-i*j/k)
7 in main() end
```



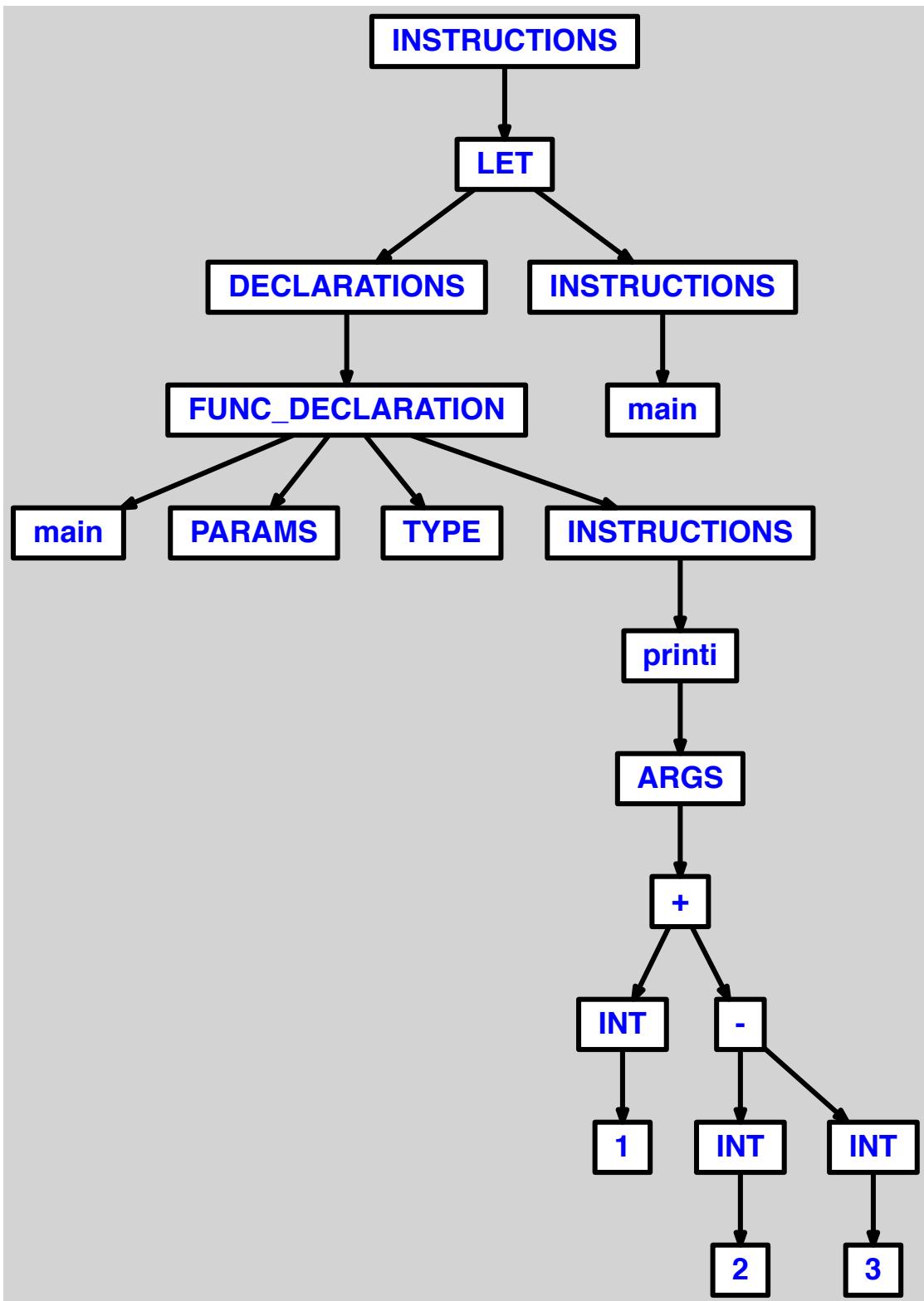
### 3.2.49 addition a 3 termes, avec parenthesage des 2 termes a droite

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



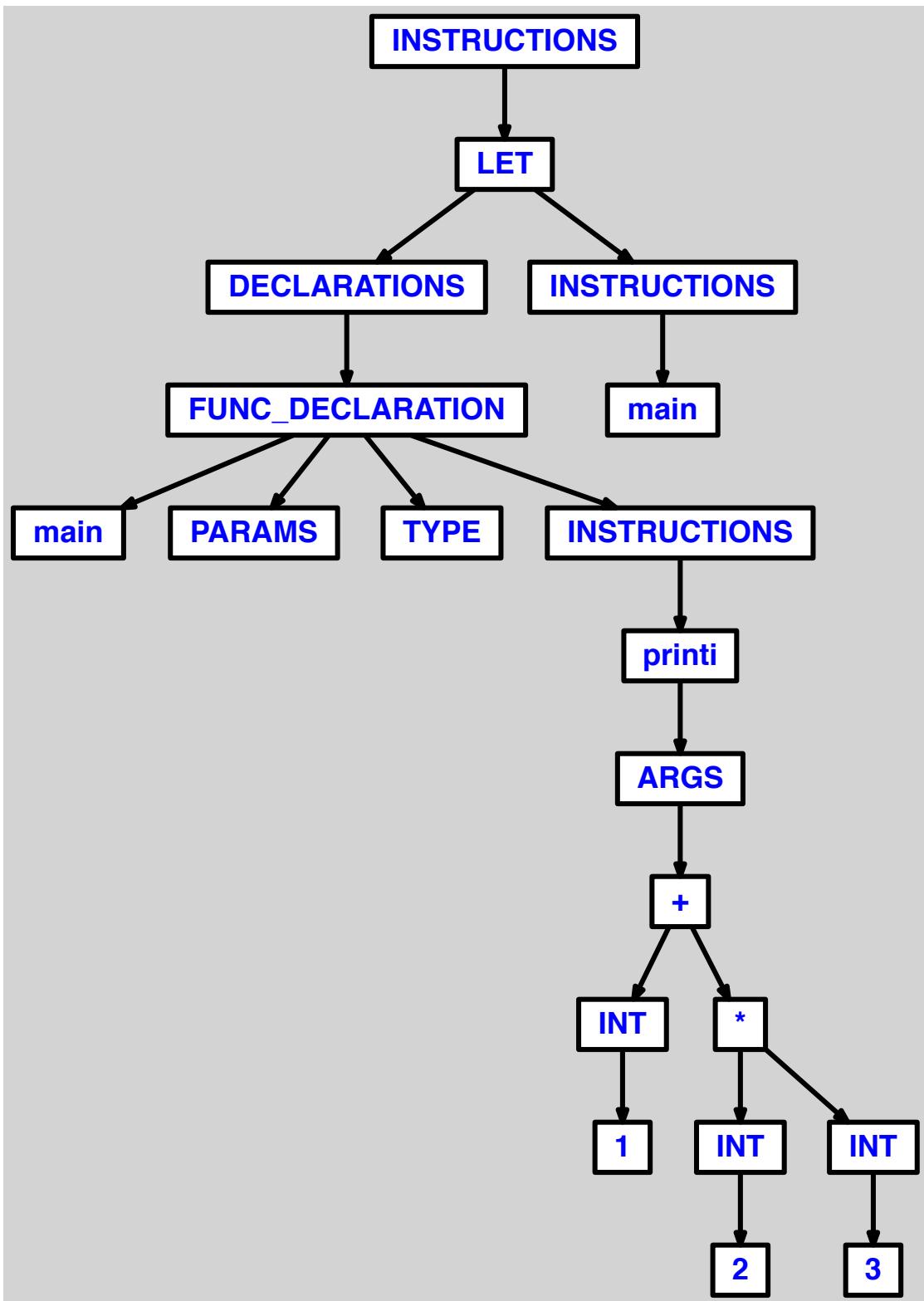
### 3.2.50 addition suivie de soustraction, avec parenthesage des 2 termes à droite

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



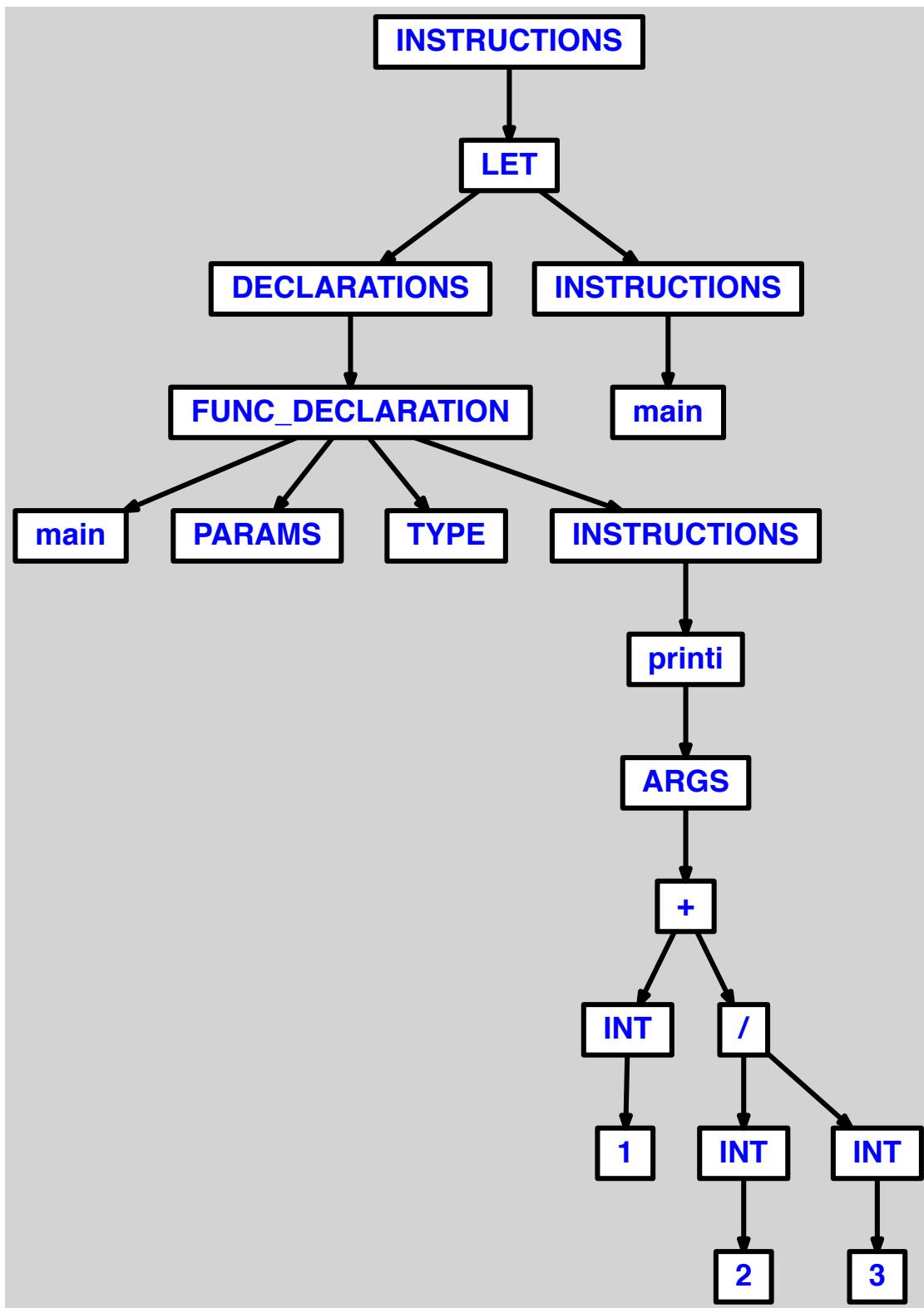
### 3.2.51 addition suivie de multiplication, avec parenthesage des 2 termes à droite

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



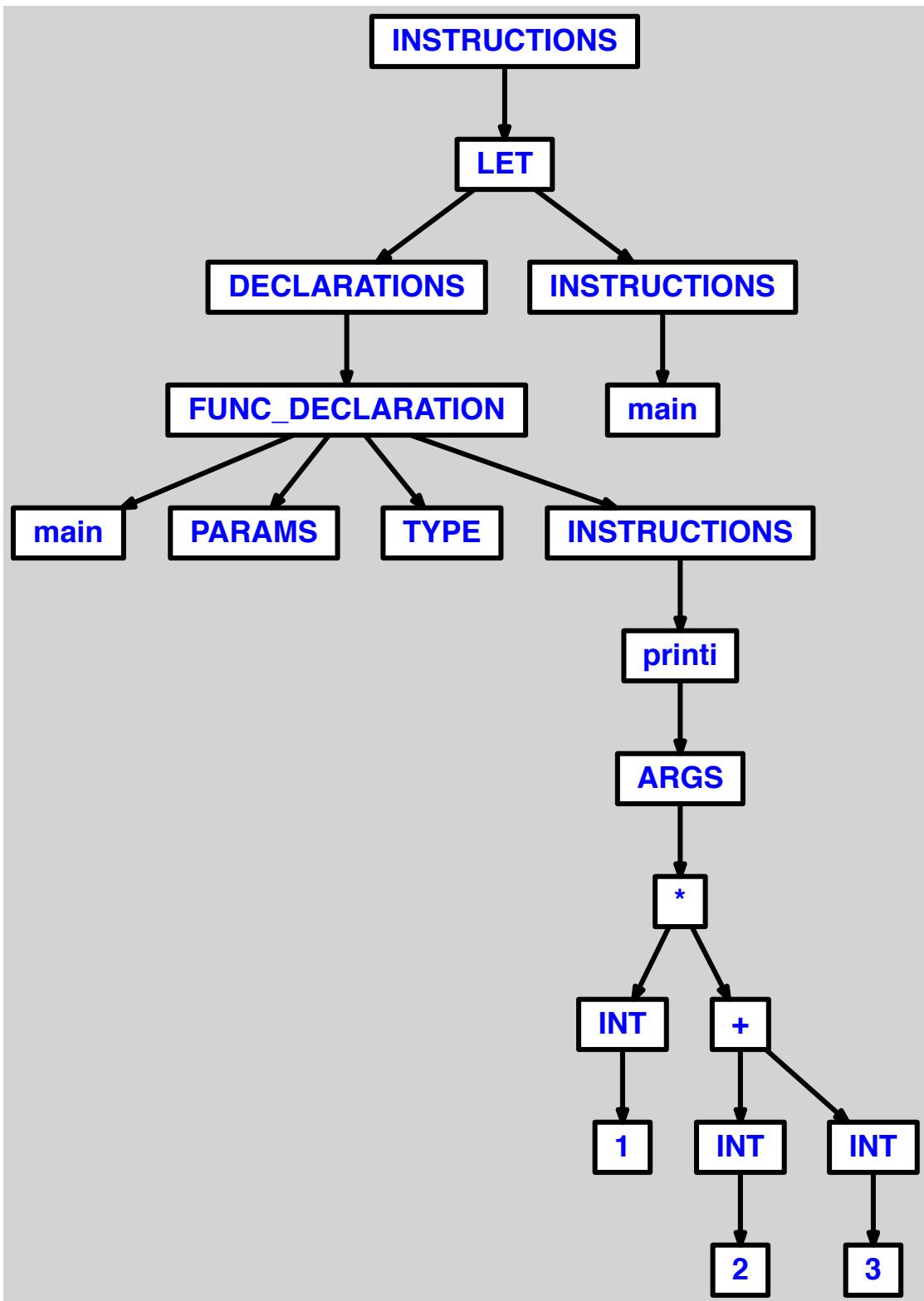
### 3.2.52 addition suivie de division, avec parenthesage des 2 termes à droite

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



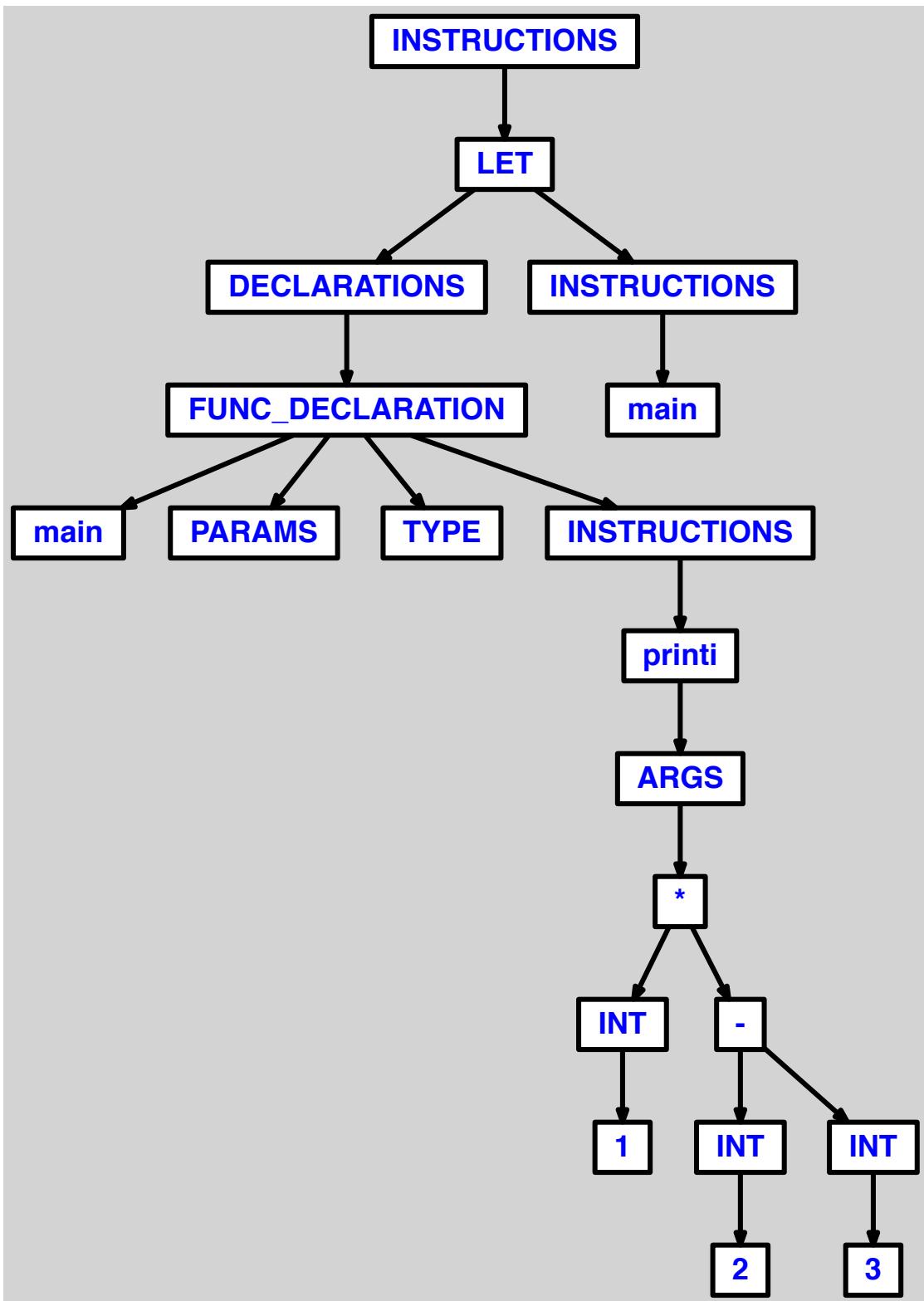
### 3.2.53 multiplication suivie d'addition, avec parenthesage des 2 termes à droite

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



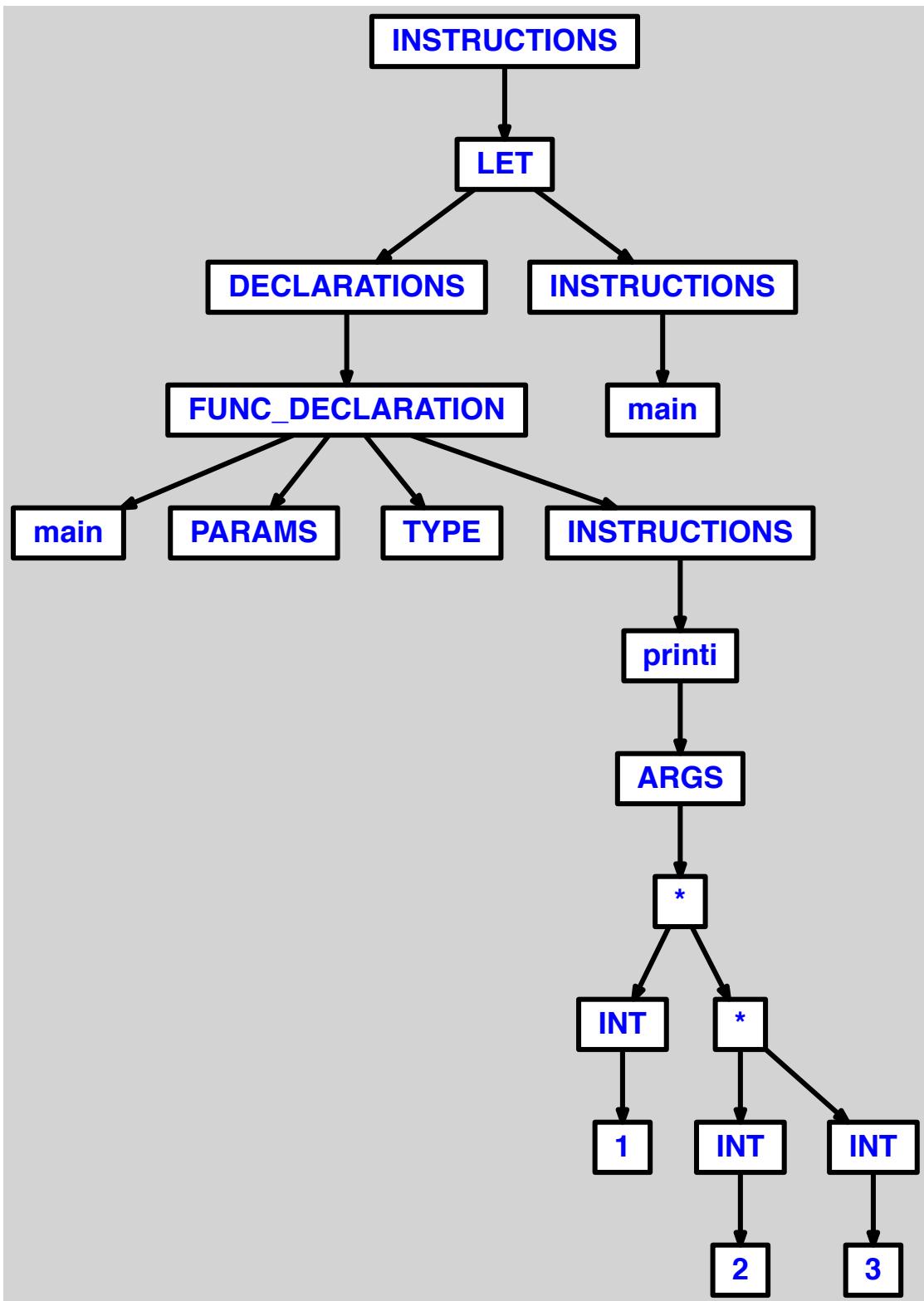
### 3.2.54 multiplication suivie de soustraction, avec parenthesage des 2 termes à droite

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



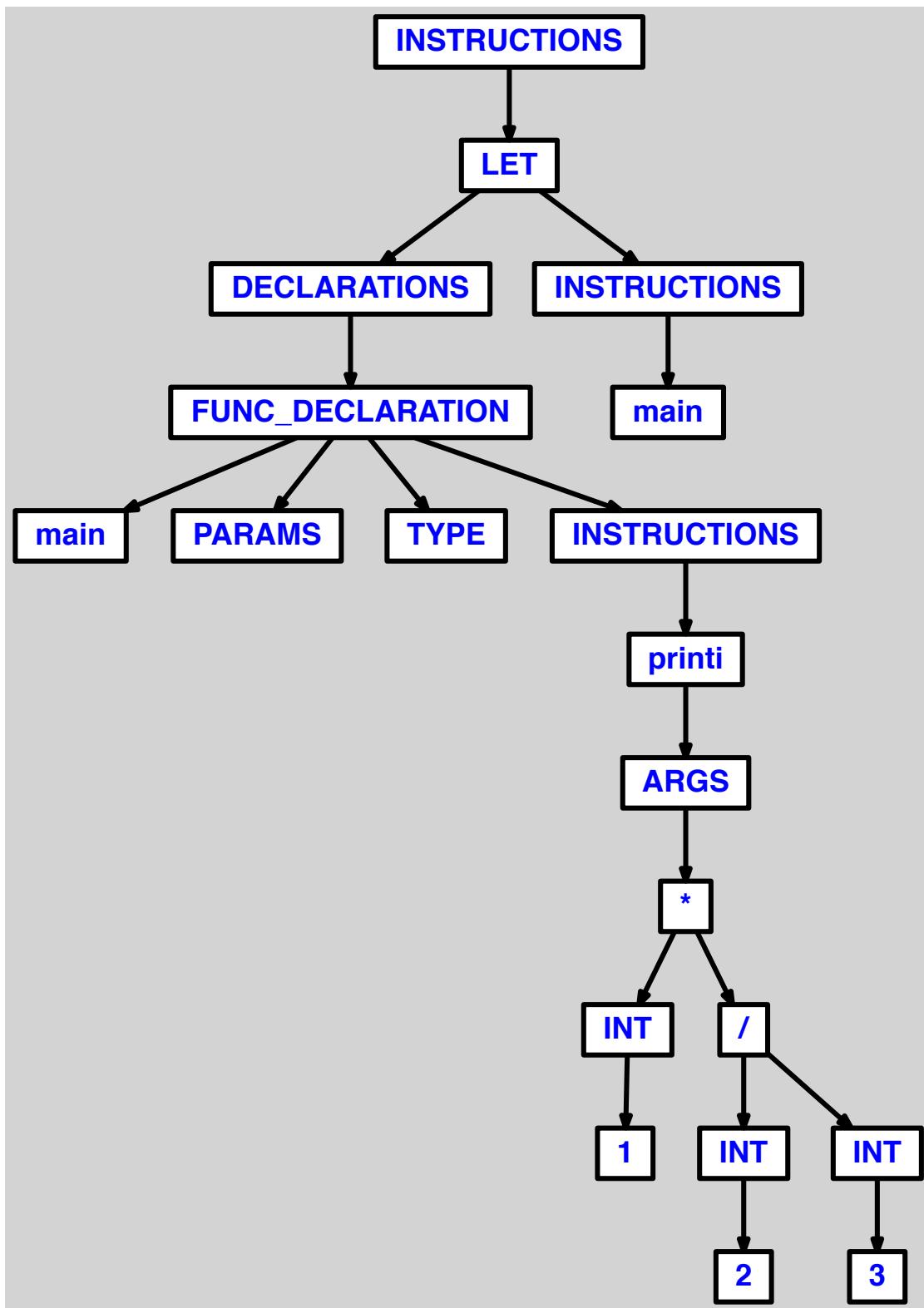
### 3.2.55 multiplication a 3 termes, avec parenthesage des 2 termes a droite

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



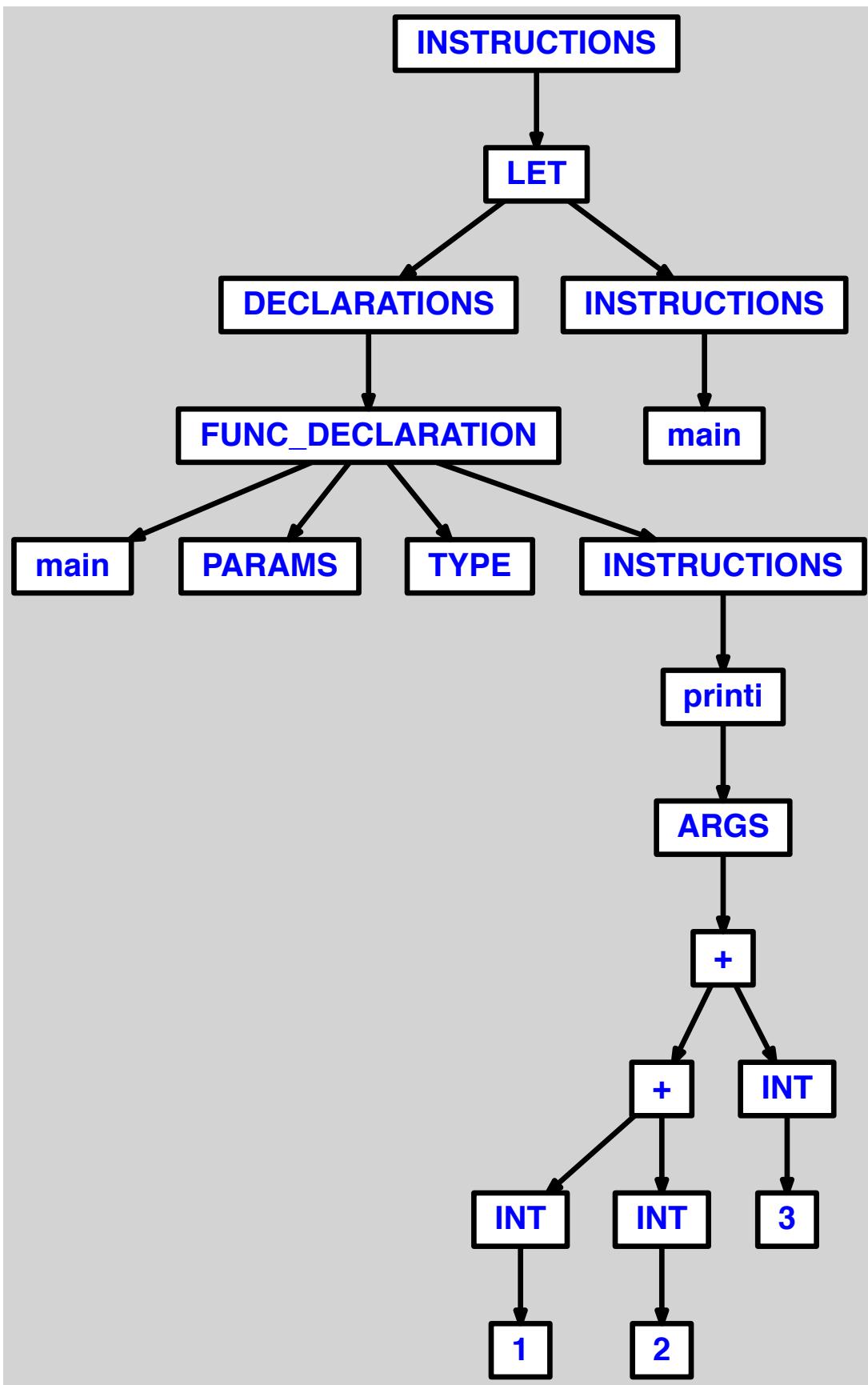
### 3.2.56 multiplication suivie de vision, avec parenthesage des 2 termes a droite

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



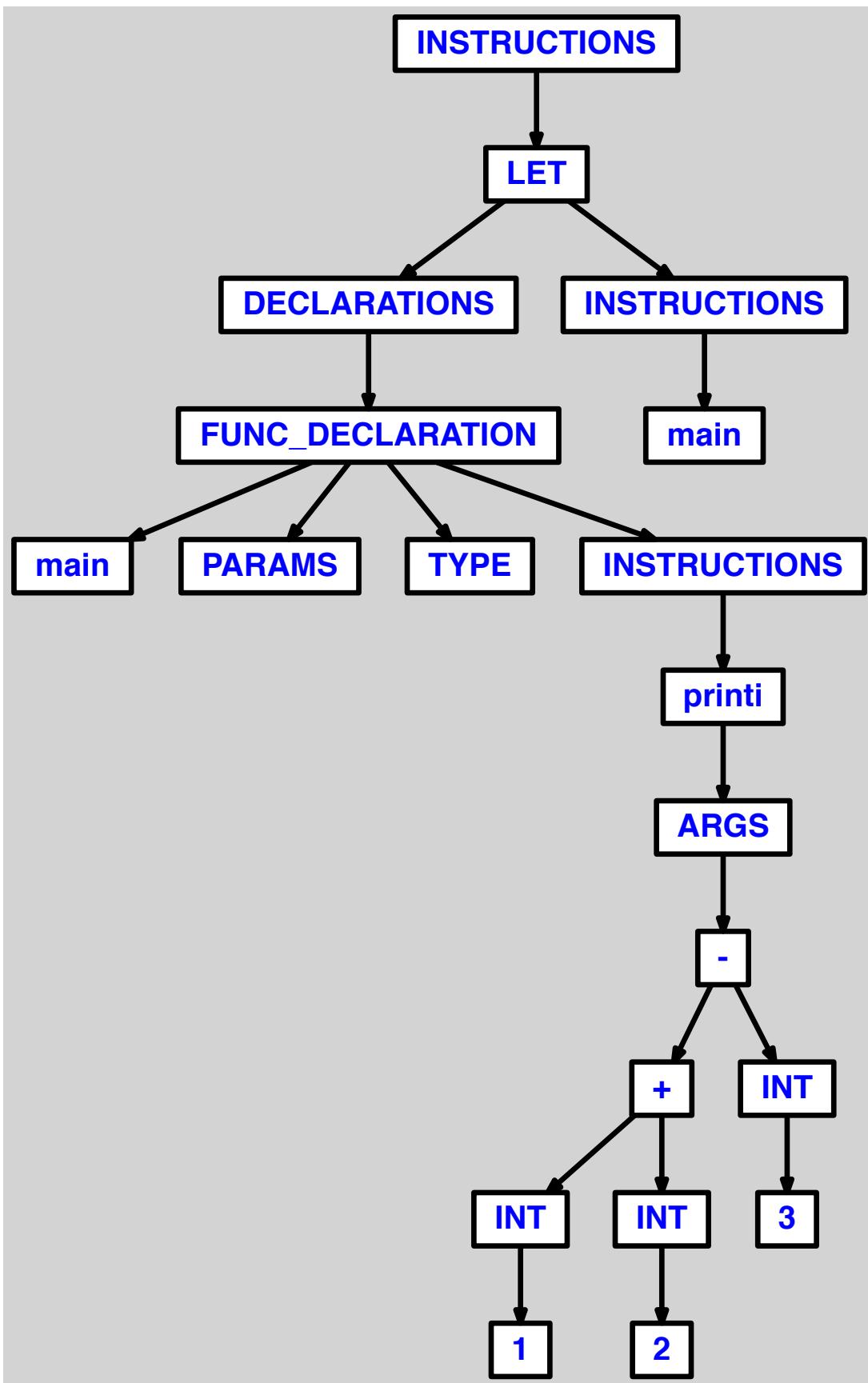
### 3.2.57 addition a 3 termes, avec parenthesage des 2 termes a gauche

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



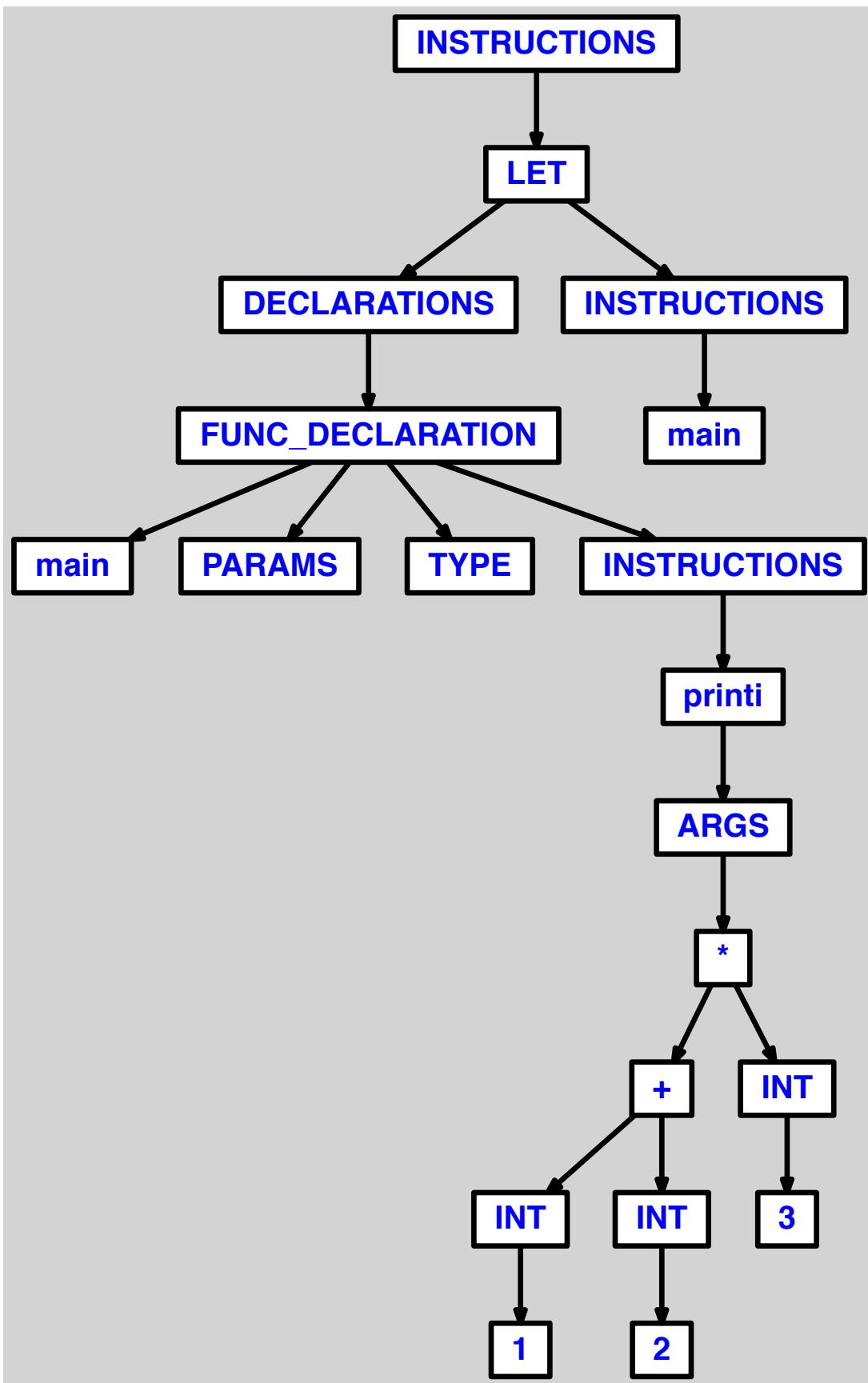
### 3.2.58 addition suivie de soustraction, avec parenthesage des 2 termes à gauche

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



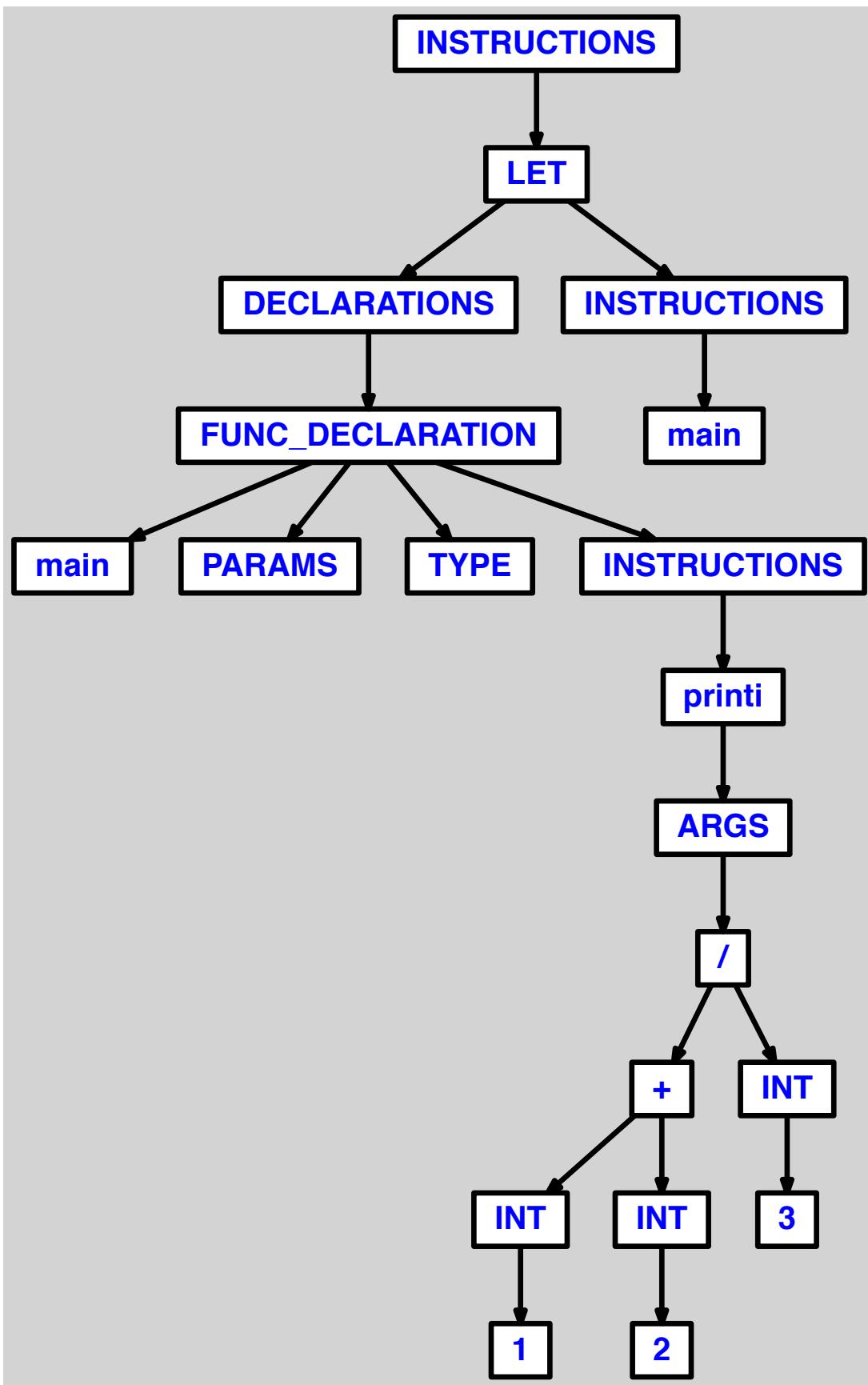
### 3.2.59 addition suivie de multiplication, avec parenthesage des 2 termes à gauche

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



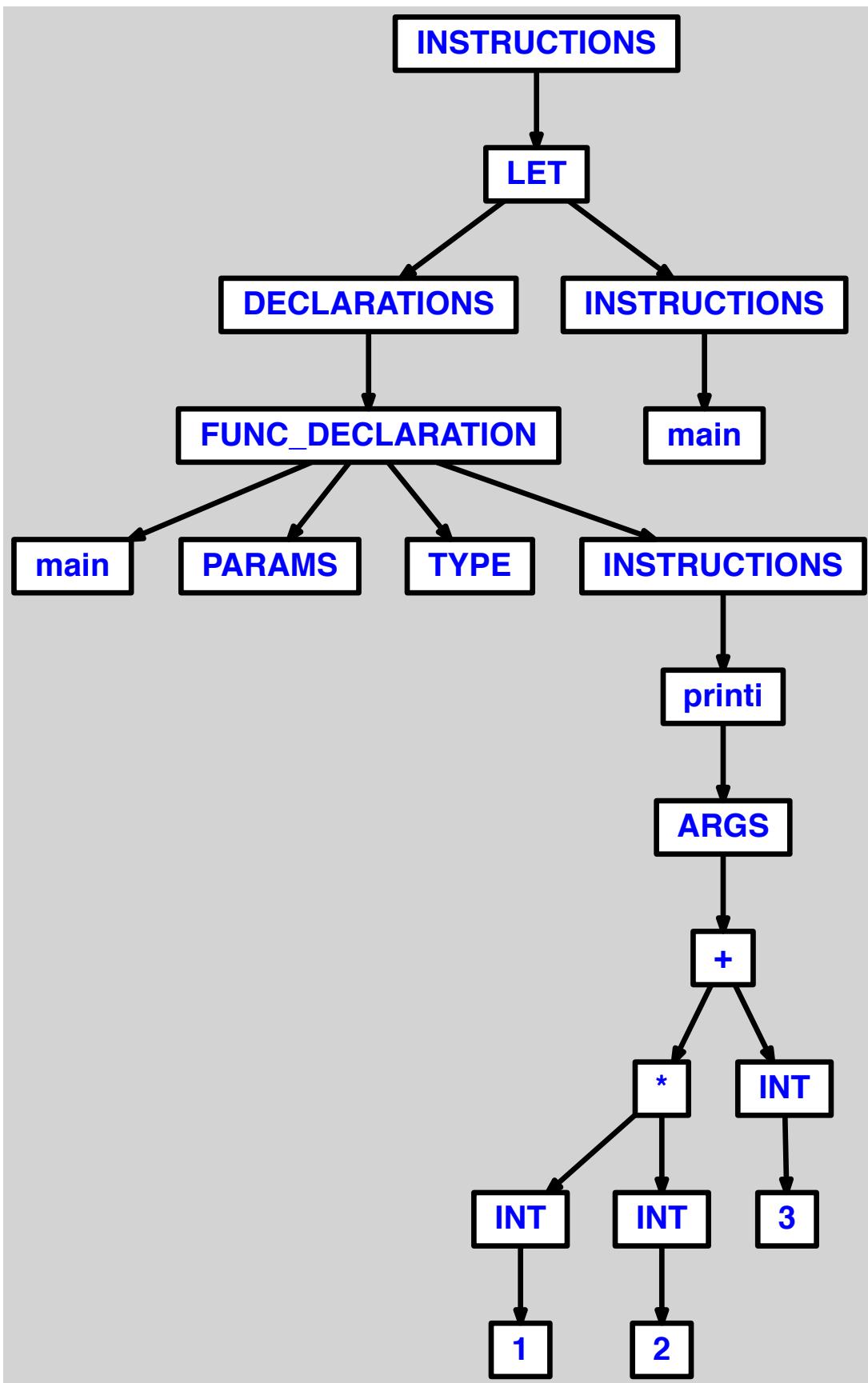
### 3.2.60 addition suivie de division, avec parenthesage des 2 termes à gauche

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



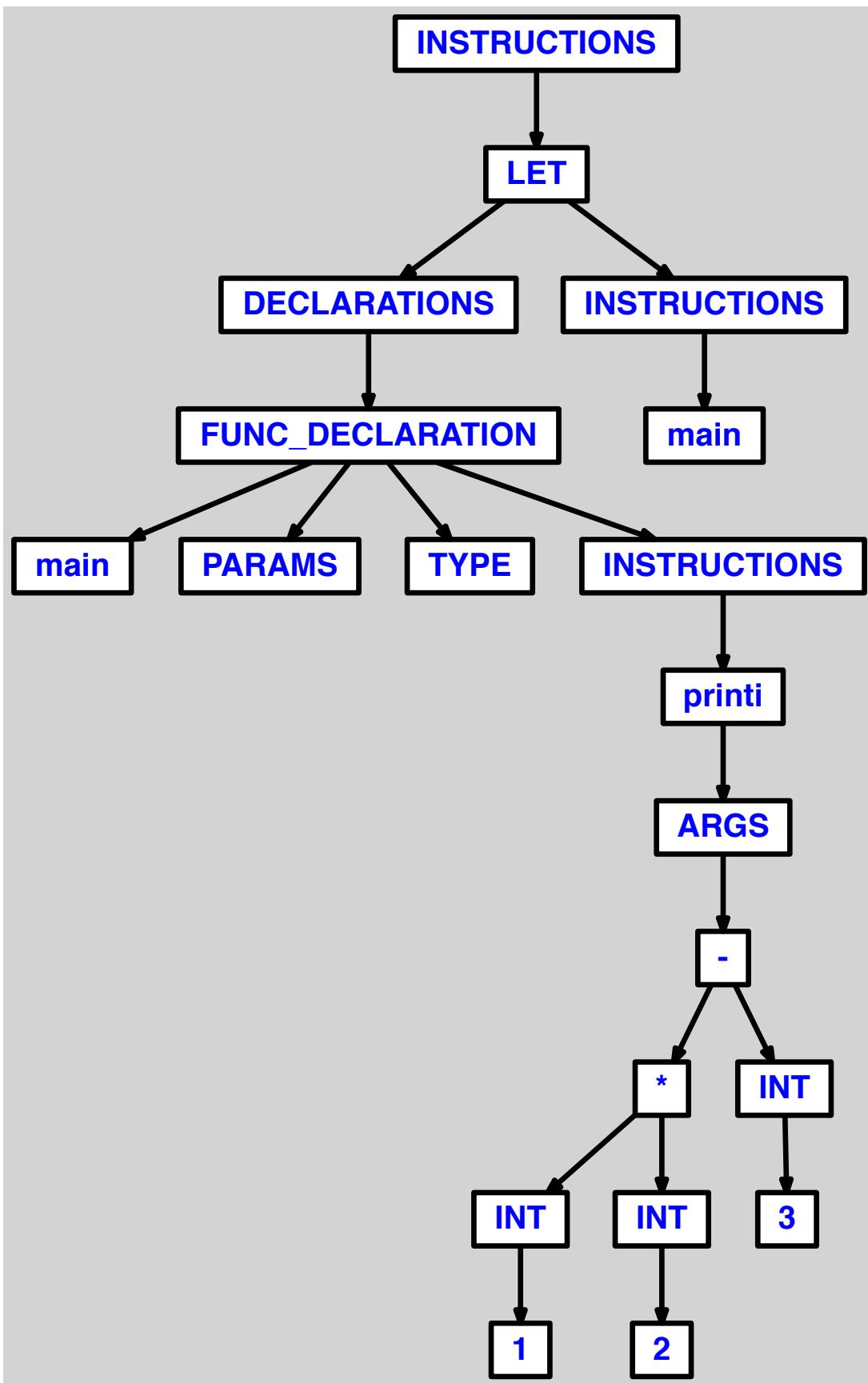
### 3.2.61 multiplication suivie d'addition, avec parenthesage des 2 termes à gauche

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



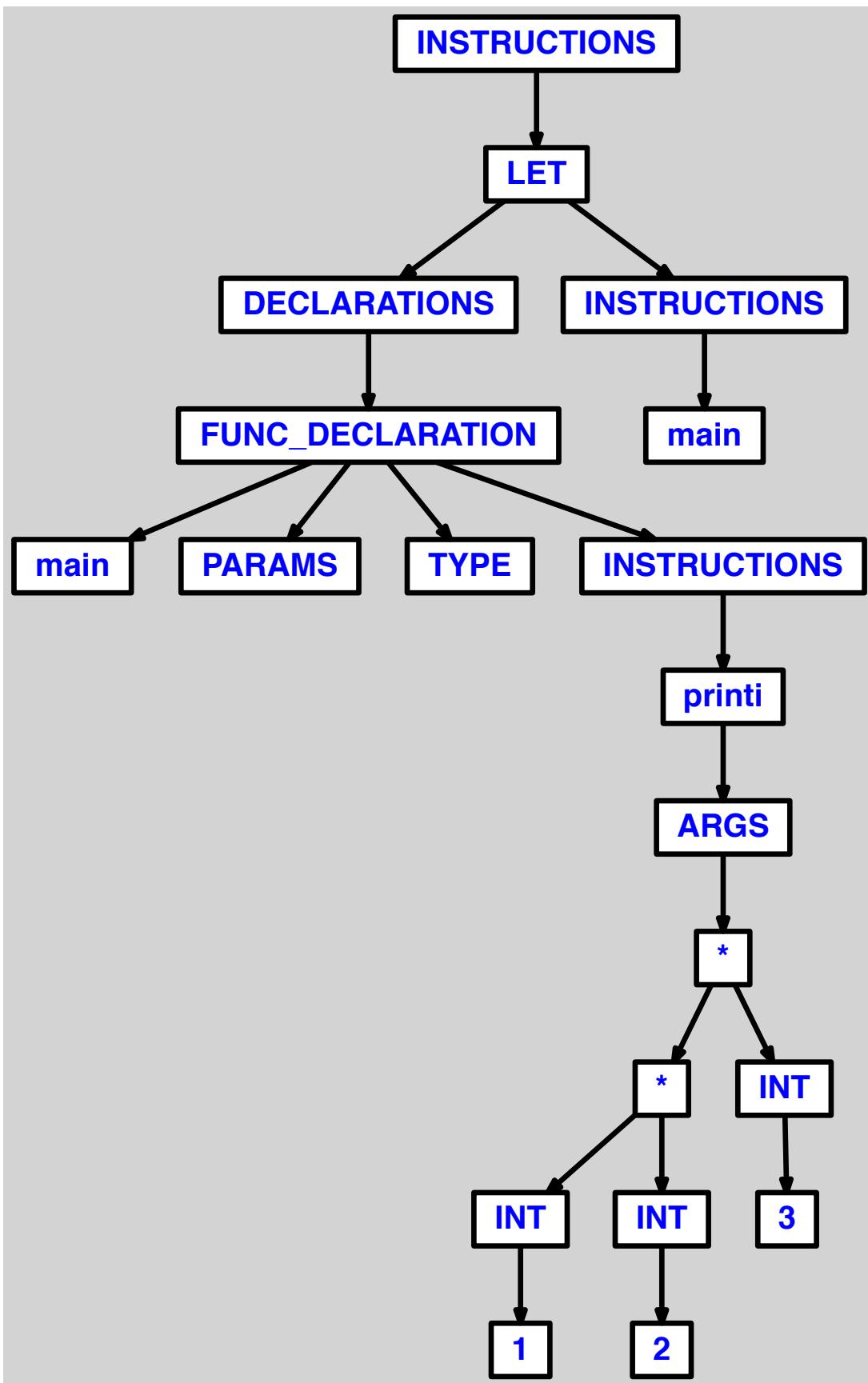
### 3.2.62 multiplication suivie de soustraction, avec parenthesage des 2 termes à gauche

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



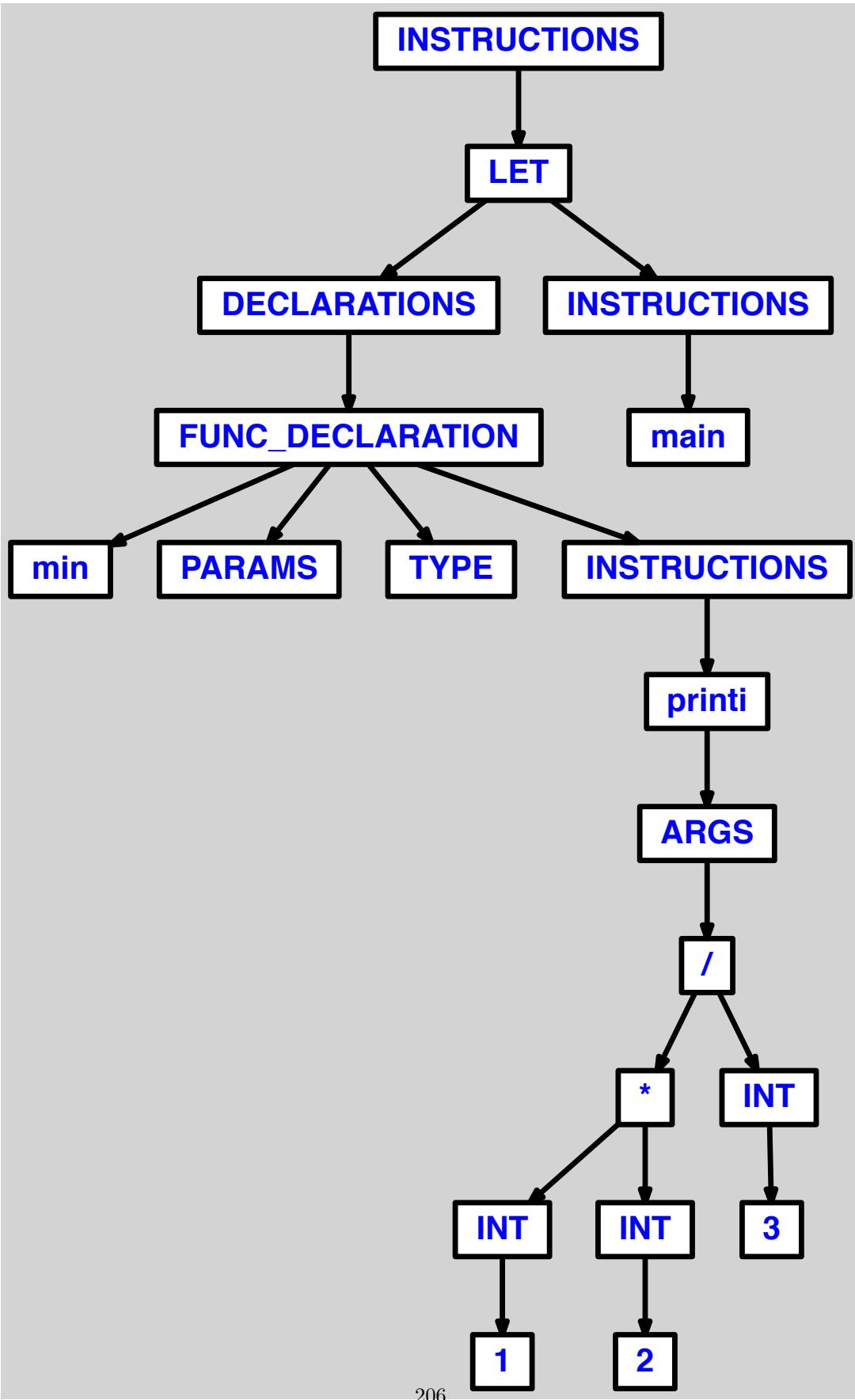
### 3.2.63 multiplication a 3 termes, avec parenthesage des 2 termes a gauche

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



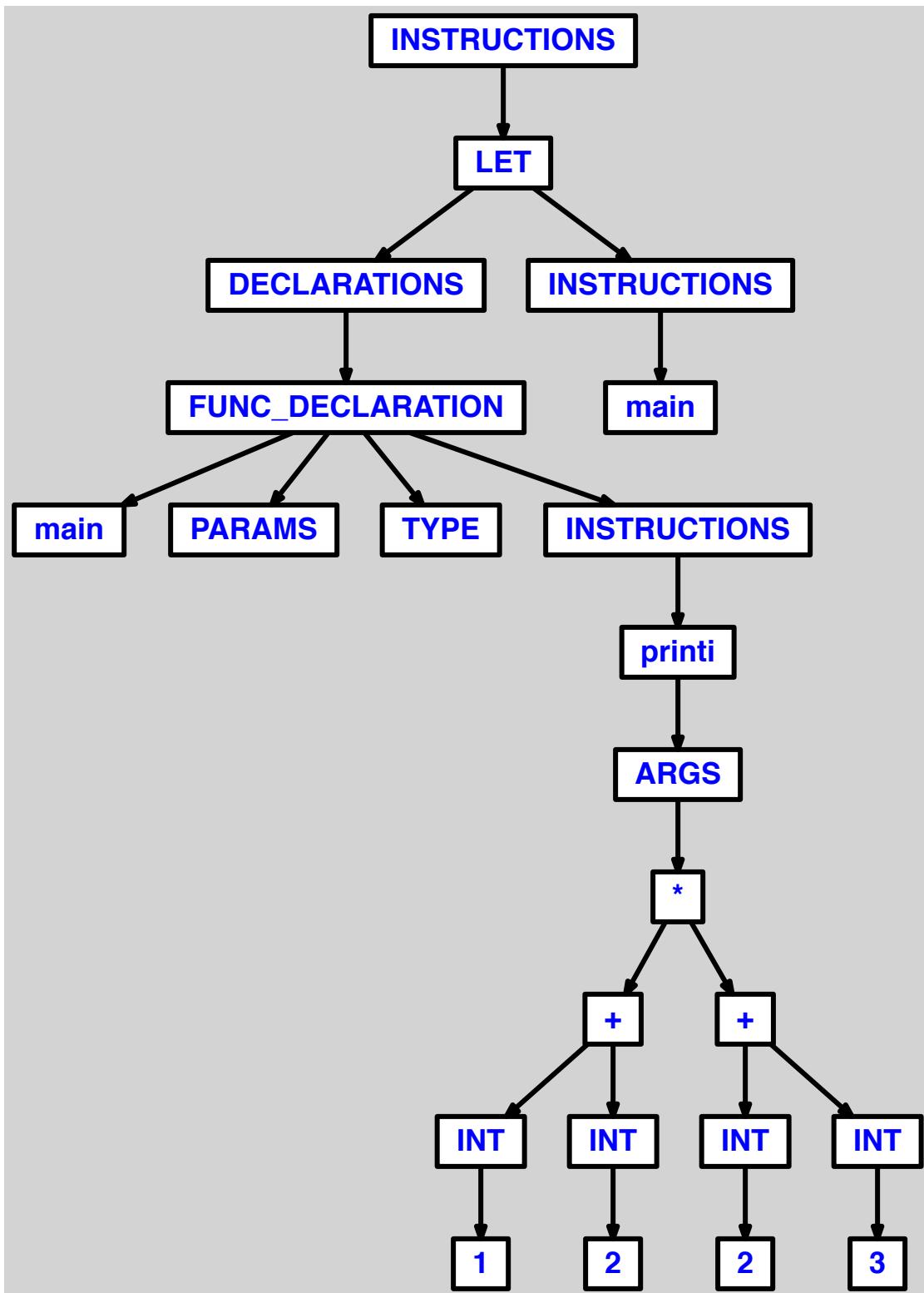
### 3.2.64 multiplication suivie de division, avec parenthesage des 2 termes à gauche

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



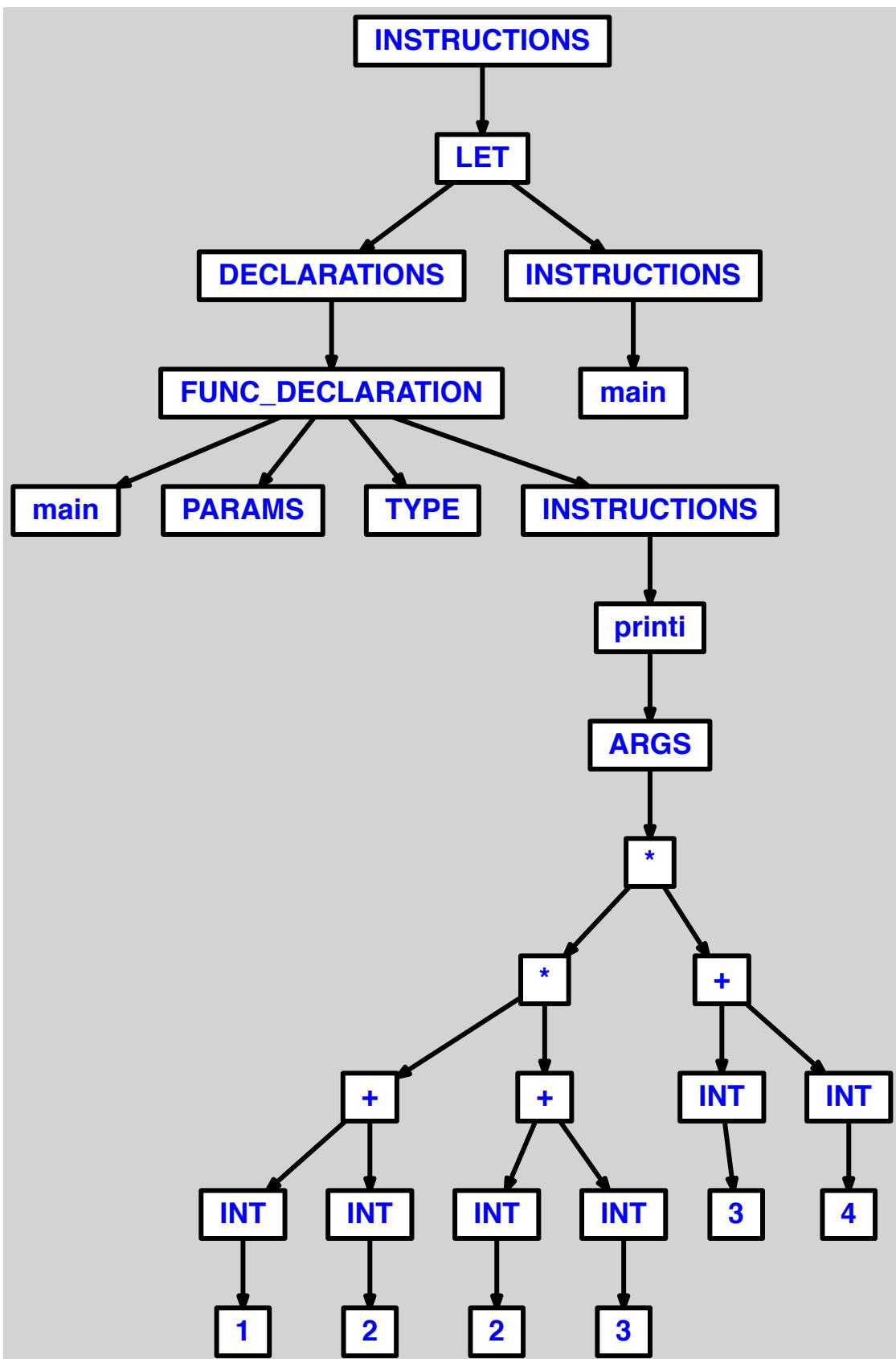
### 3.2.65 multiplication de 2 additions parenthesees

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



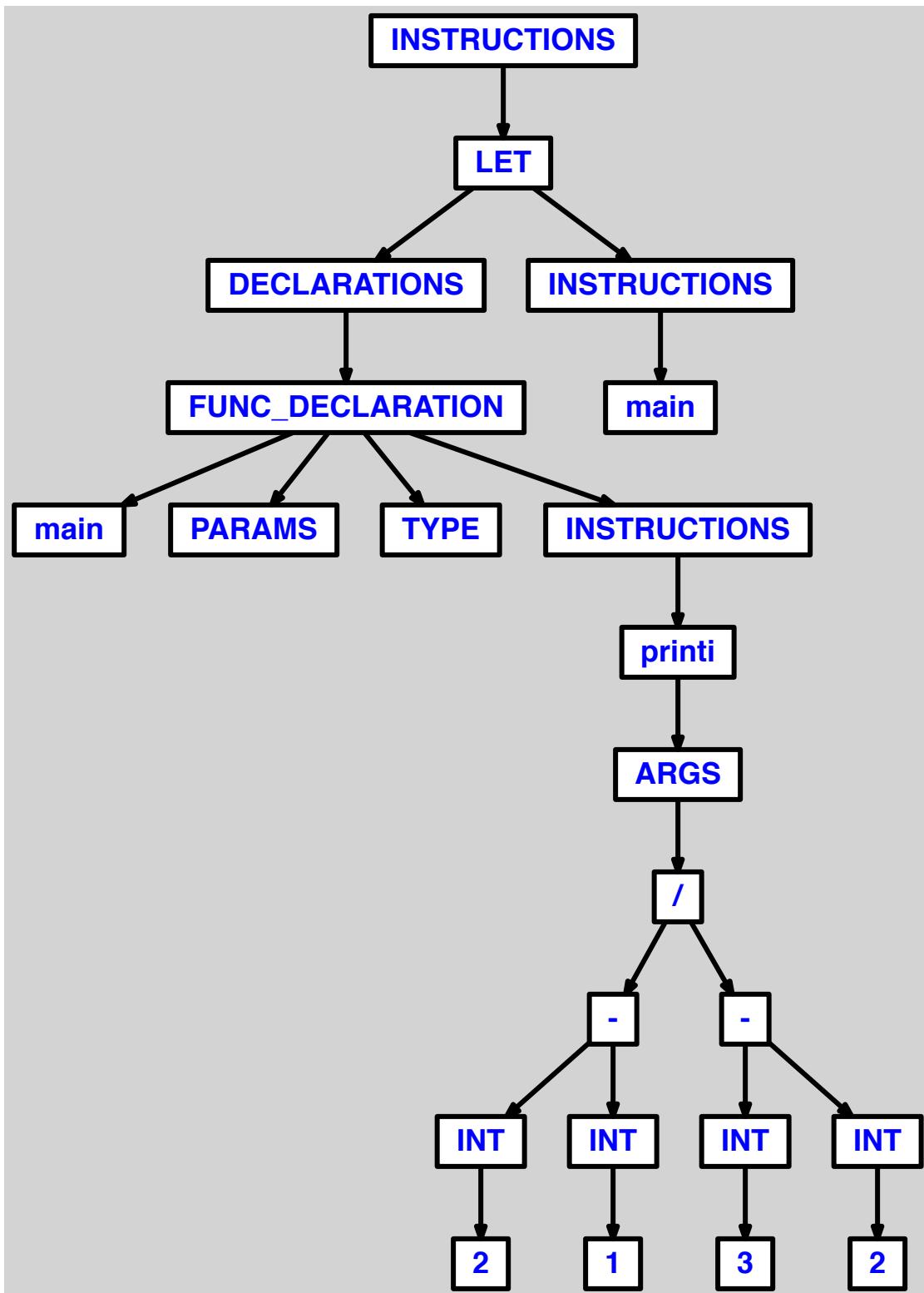
### 3.2.66 multiplication de 3 additions parenthesees

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



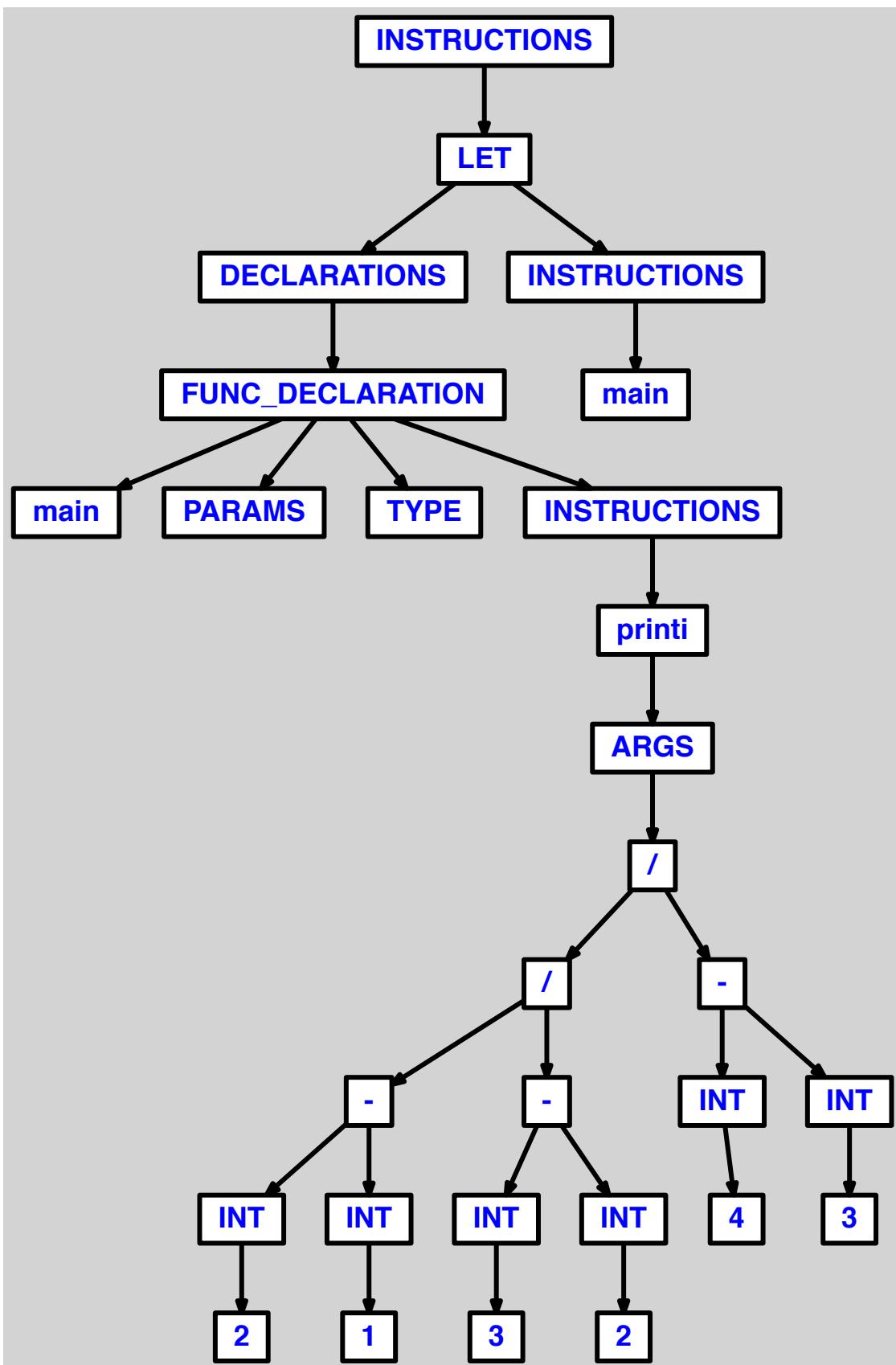
### 3.2.67 division de 2 soustractions parenthesees

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



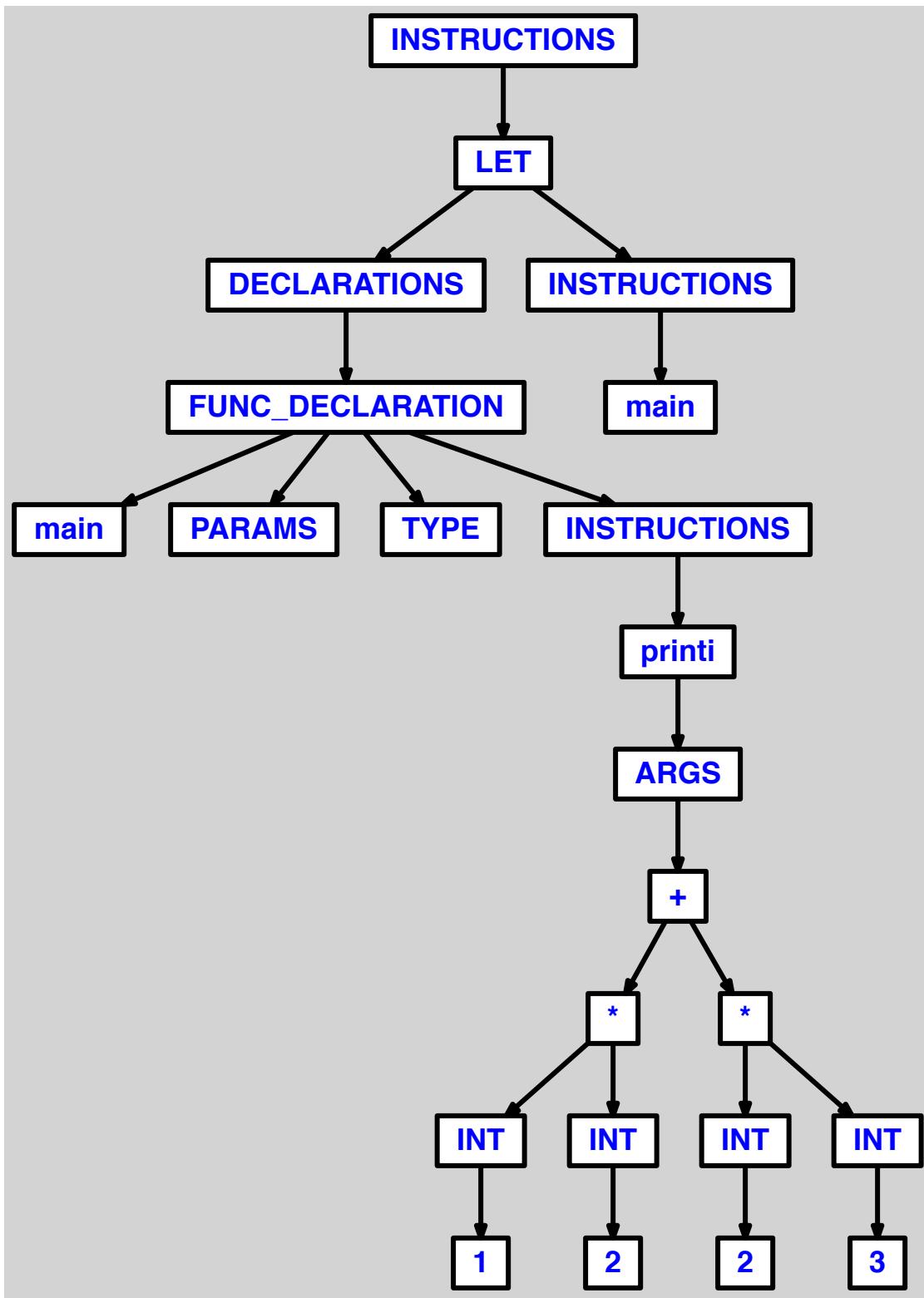
### 3.2.68 division de 3 soustractions parenthesees

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



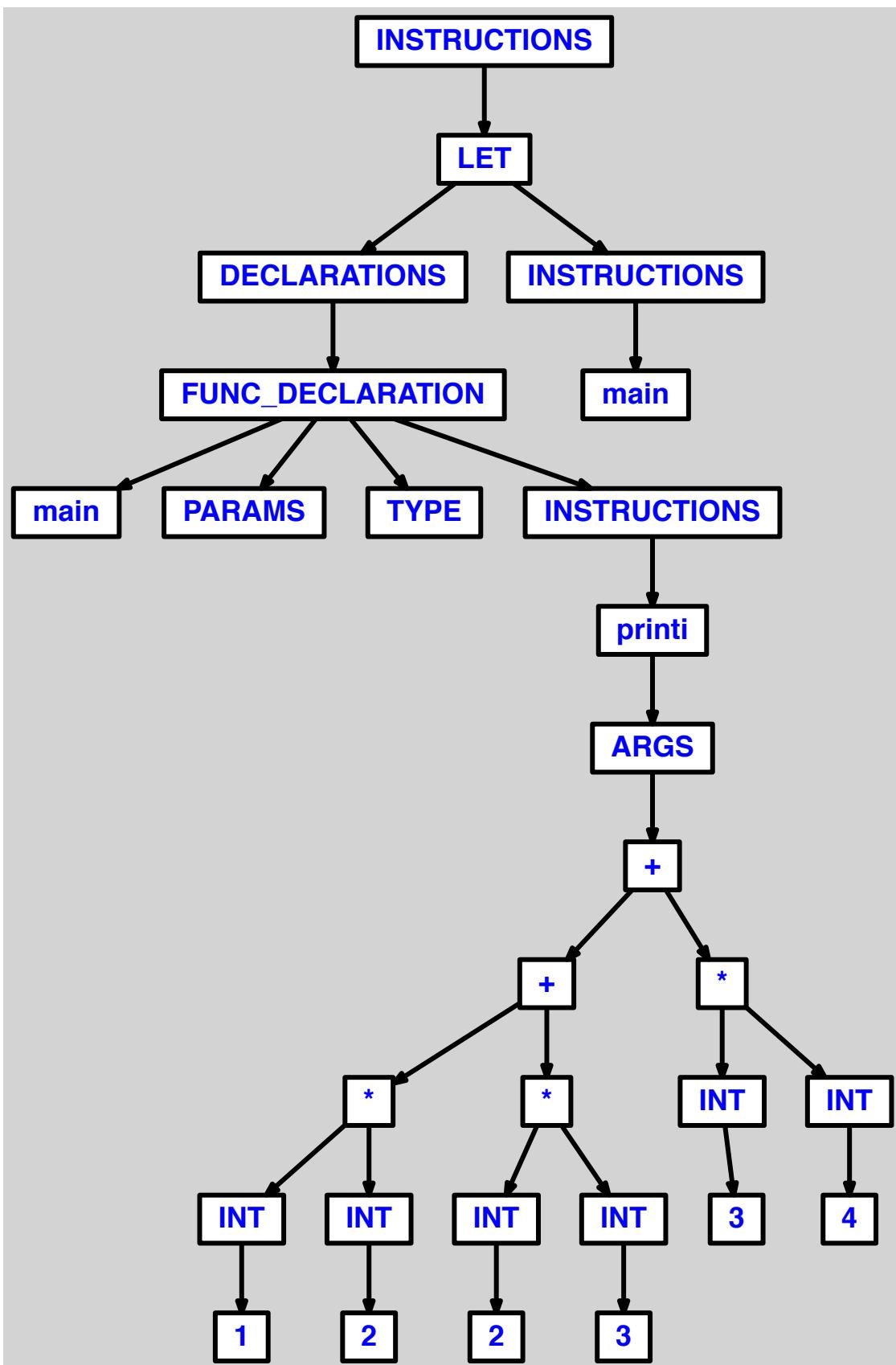
### 3.2.69 addition de 2 multiplications parenthesees

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



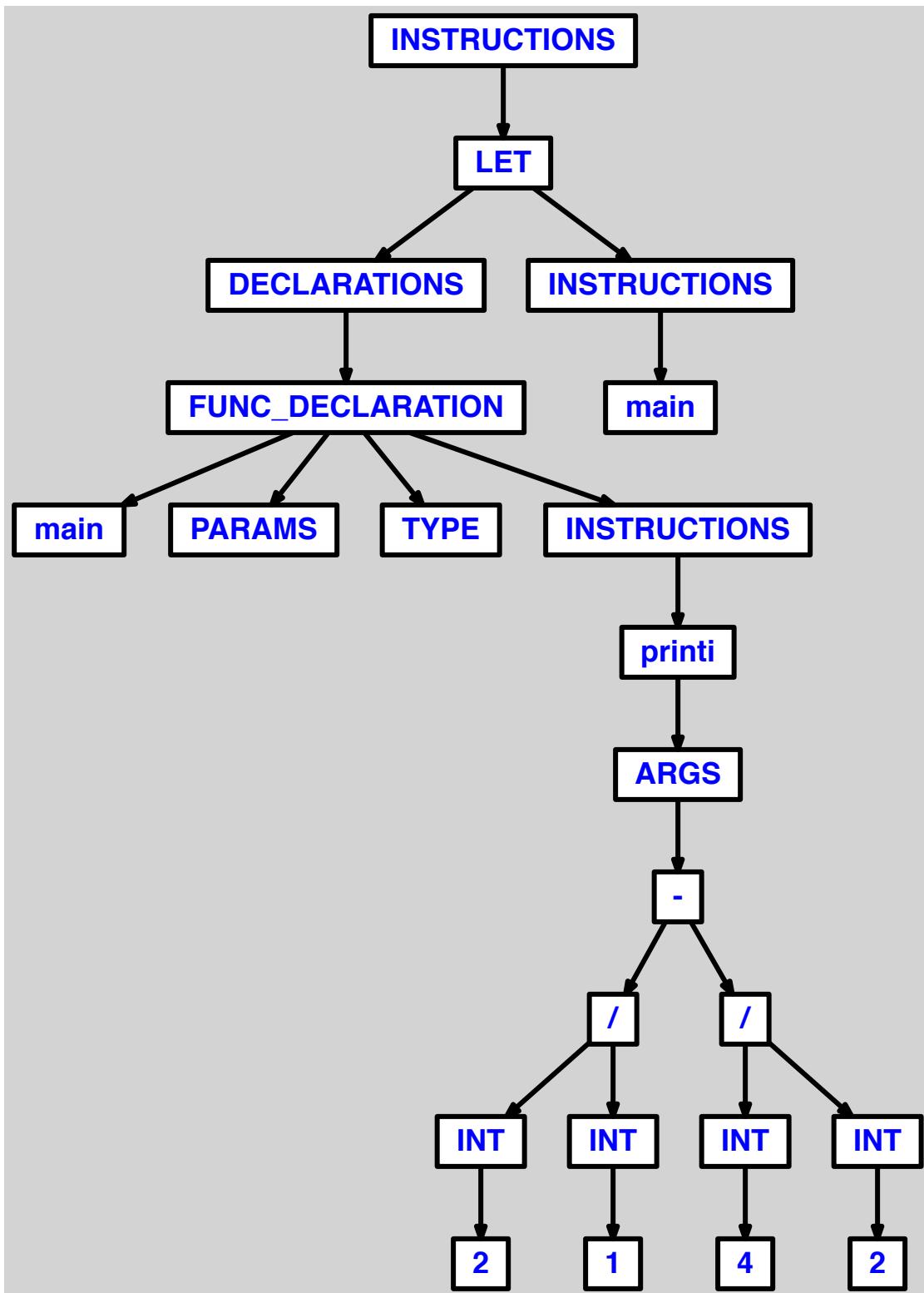
### 3.2.70 addition de 3 multiplications parenthesees

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



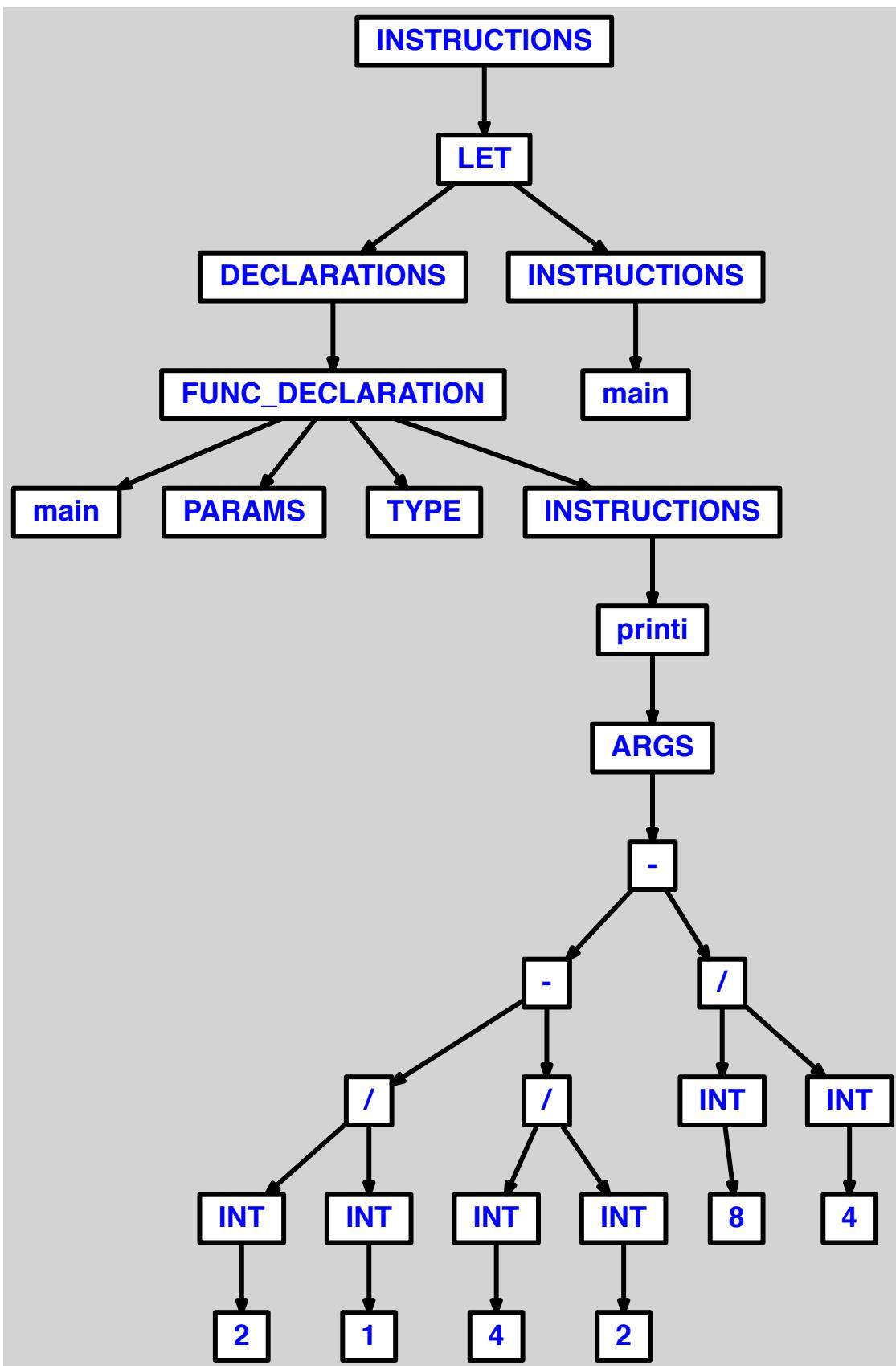
### 3.2.71 soustraction de 2 divisions parenthesees

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



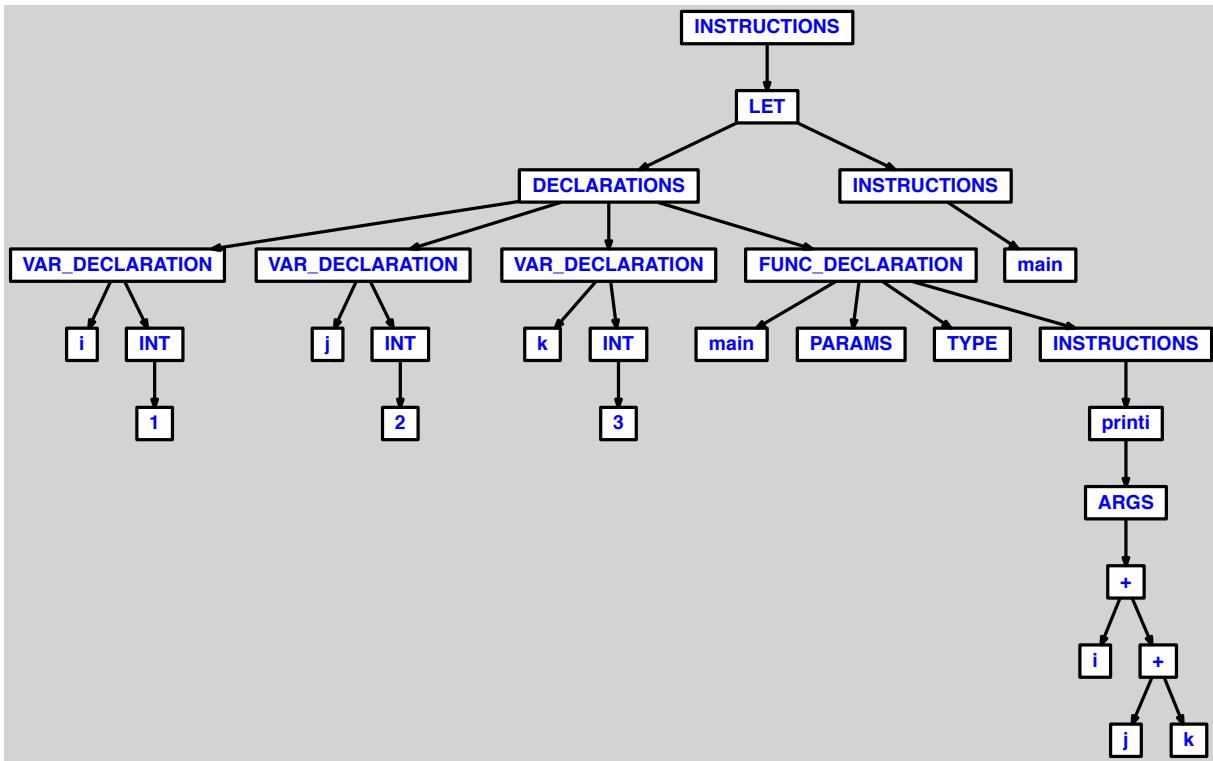
### 3.2.72 soustraction de 3 divisions parenthesees

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



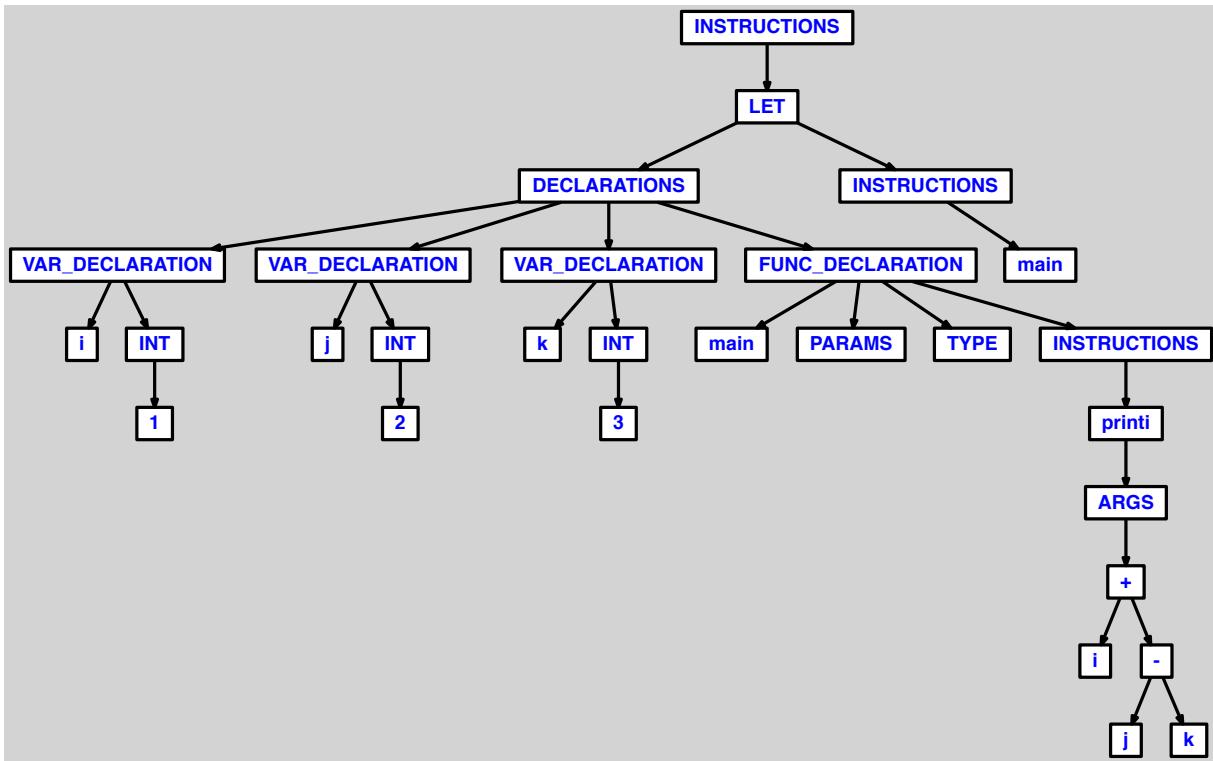
### 3.2.73 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+(j+k))
7 in main() end
```



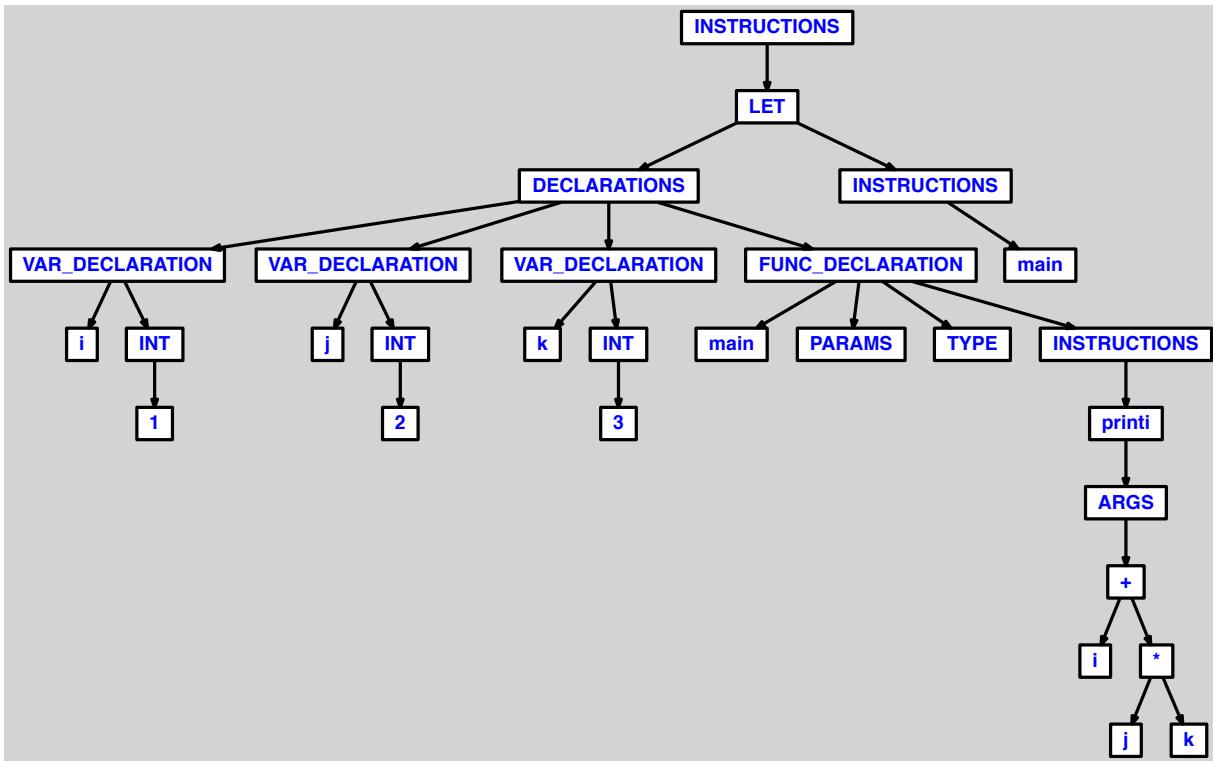
### 3.2.74 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+(j-k))
7 in main() end
```



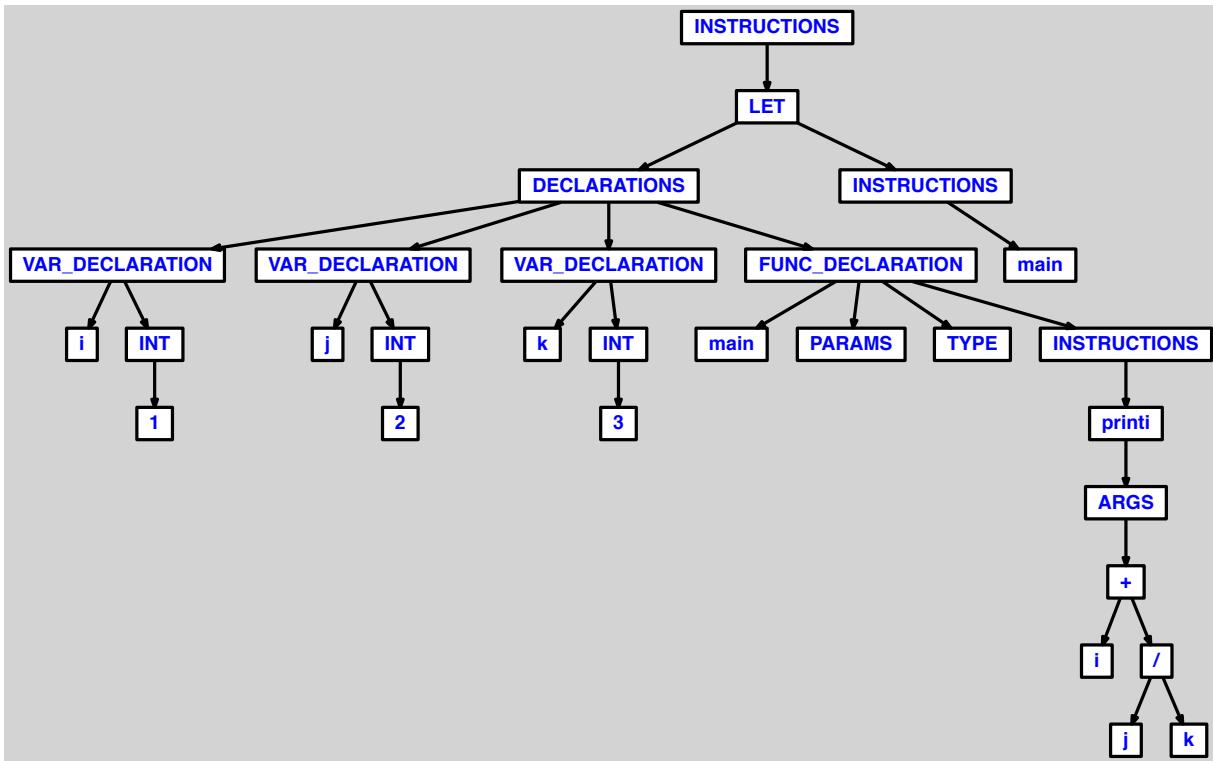
### 3.2.75 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+(j*k))
7 in main() end
```



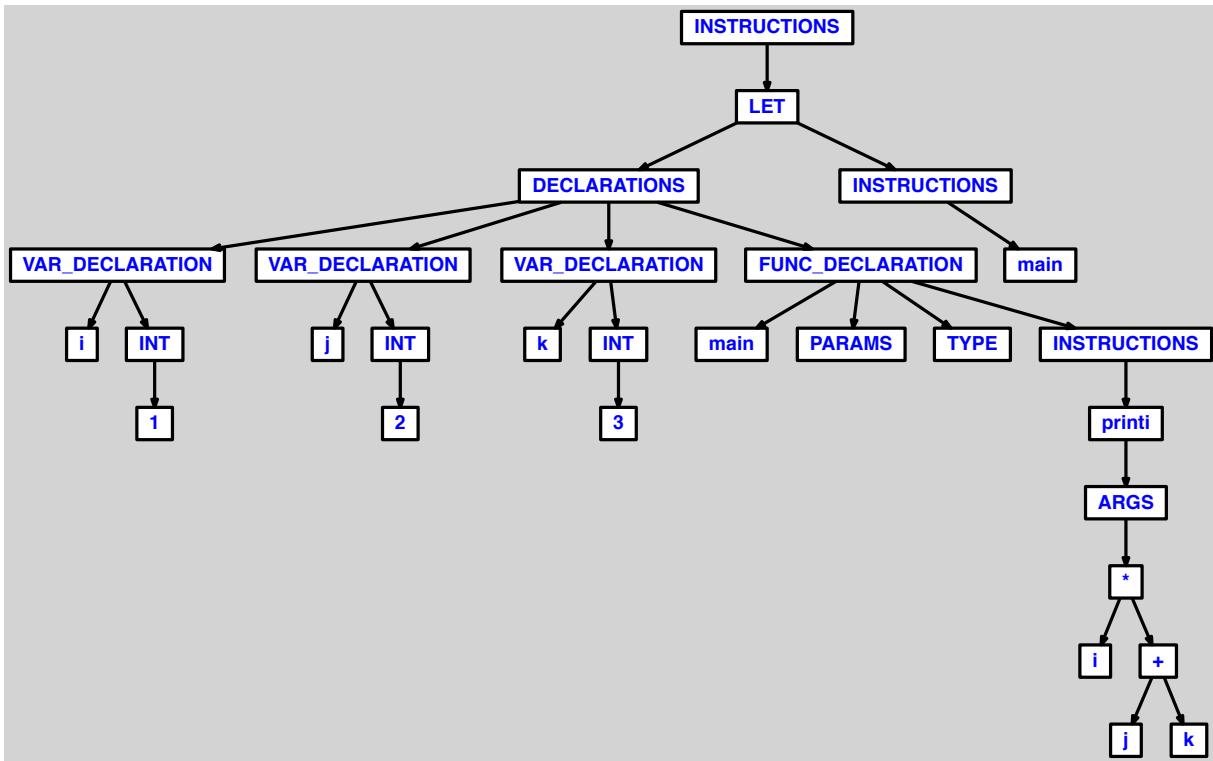
### 3.2.76 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i+(j/k))
7 in main() end
```



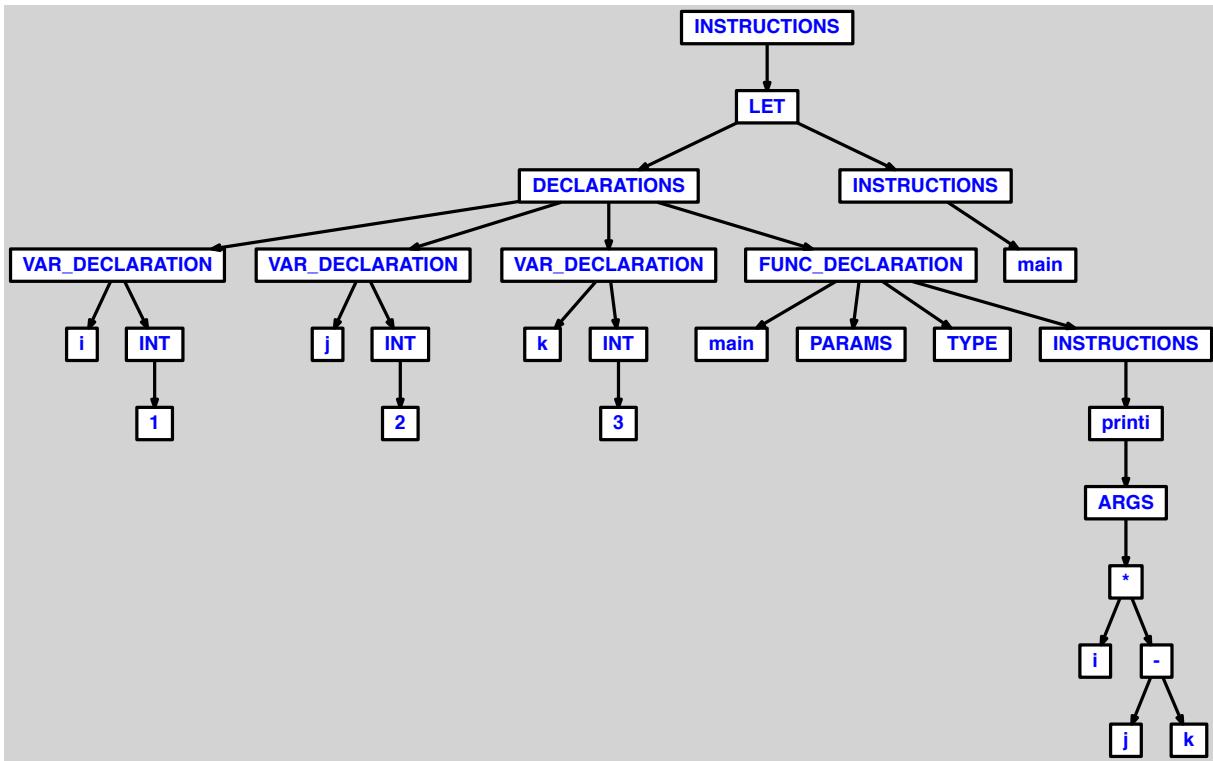
### 3.2.77 multiplication suivie d'addition, avec termes identifies par variables et parenthesage des 2 variables a droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*(j+k))
7 in main() end
```



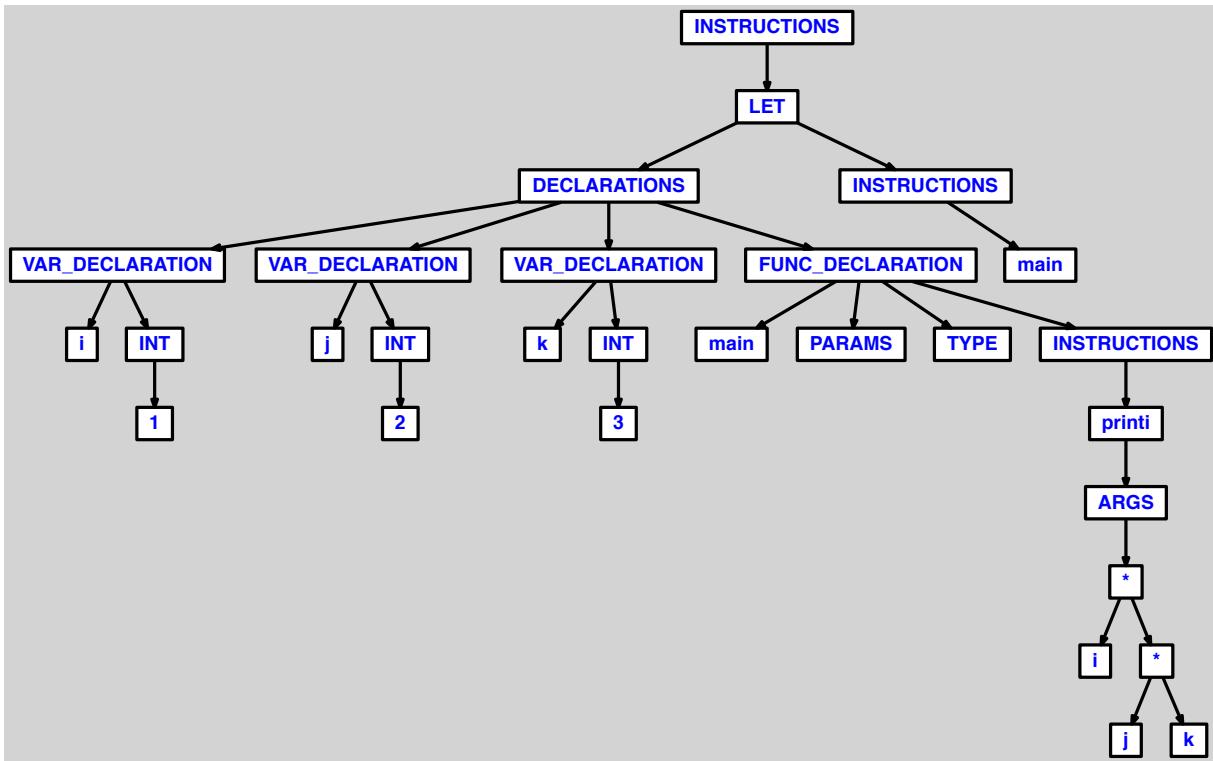
**3.2.78 multiplication suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite**

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*(j-k))
7 in main() end
```



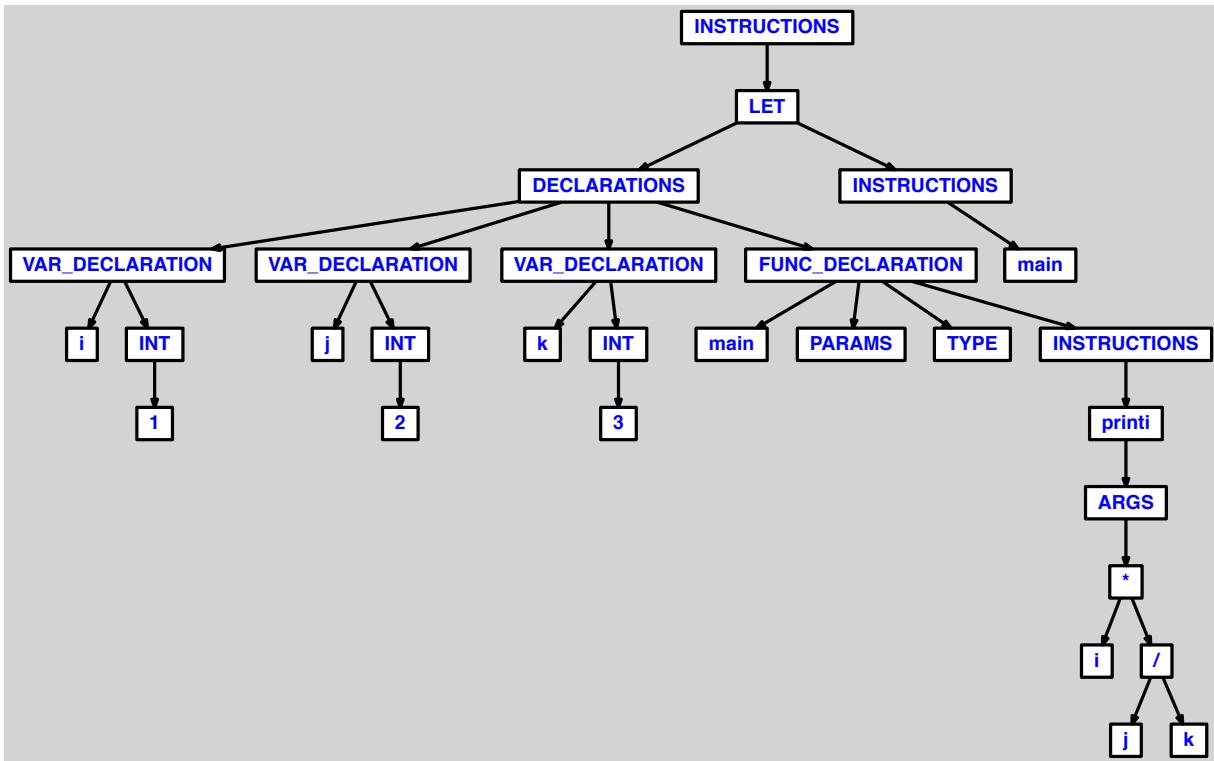
### 3.2.79 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*(j*k))
7 in main() end
```



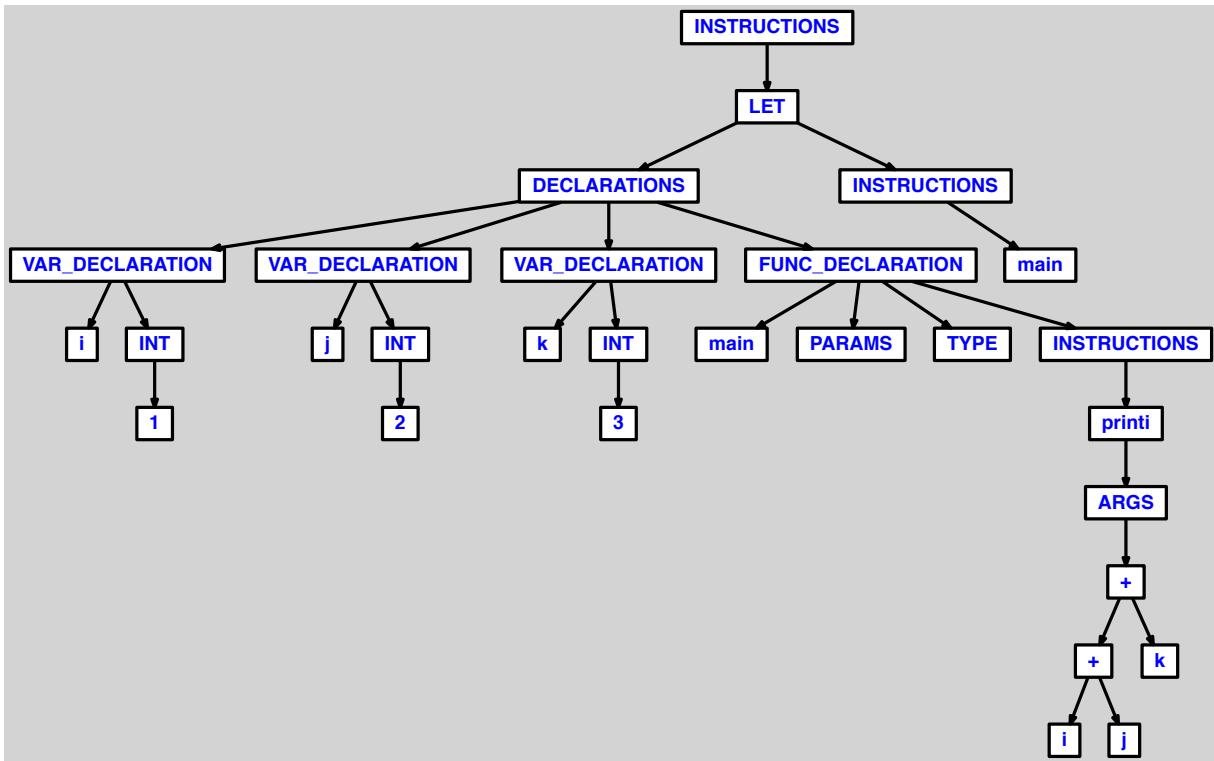
### 3.2.80 multiplication suivie de division, avec termes identifies par variables et parenthesage des 2 variables a droite

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi(i*(j/k))
7 in main() end
```



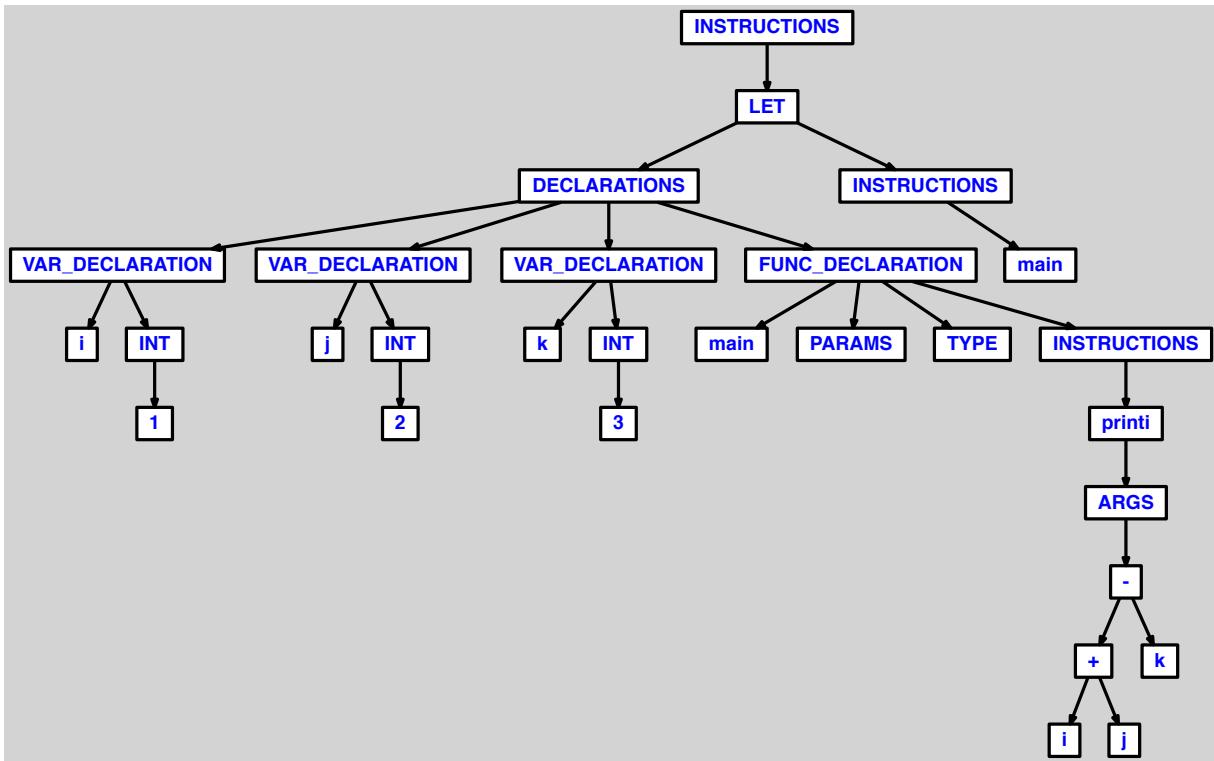
### 3.2.81 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i+j)+k)
7 in main() end
```



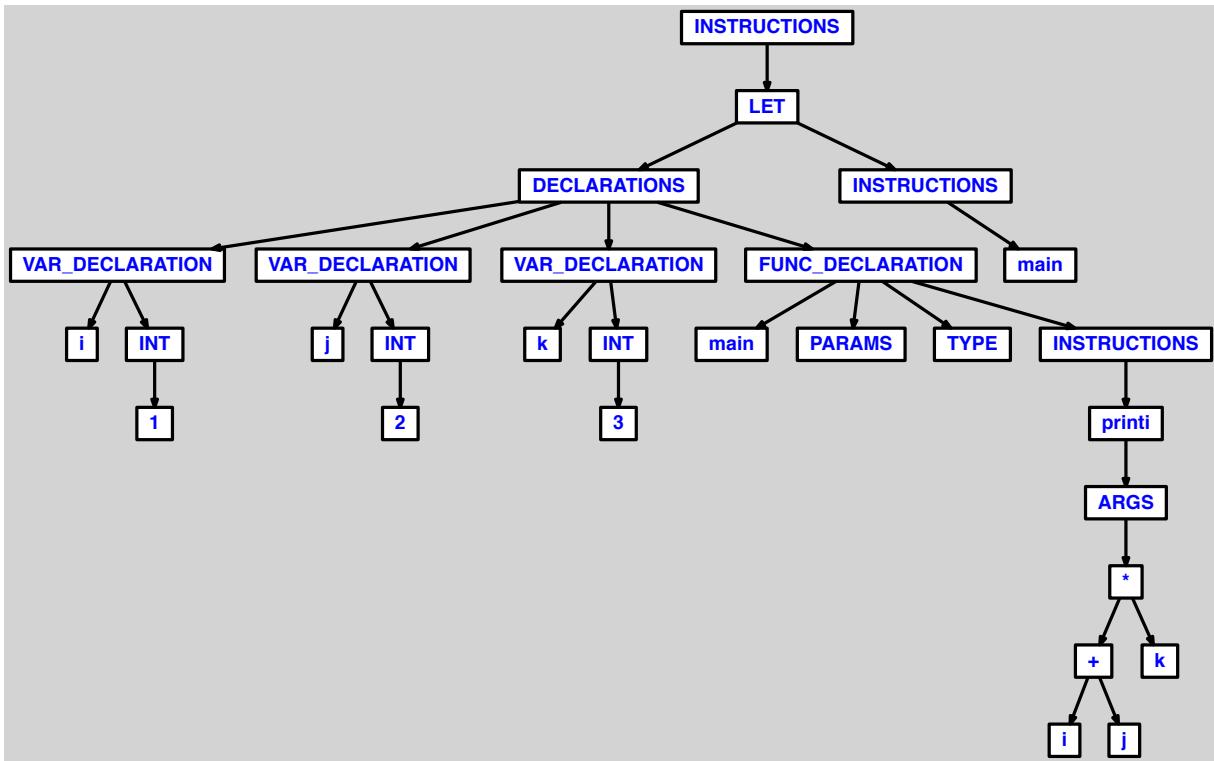
### 3.2.82 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i+j)-k)
7 in main() end
```



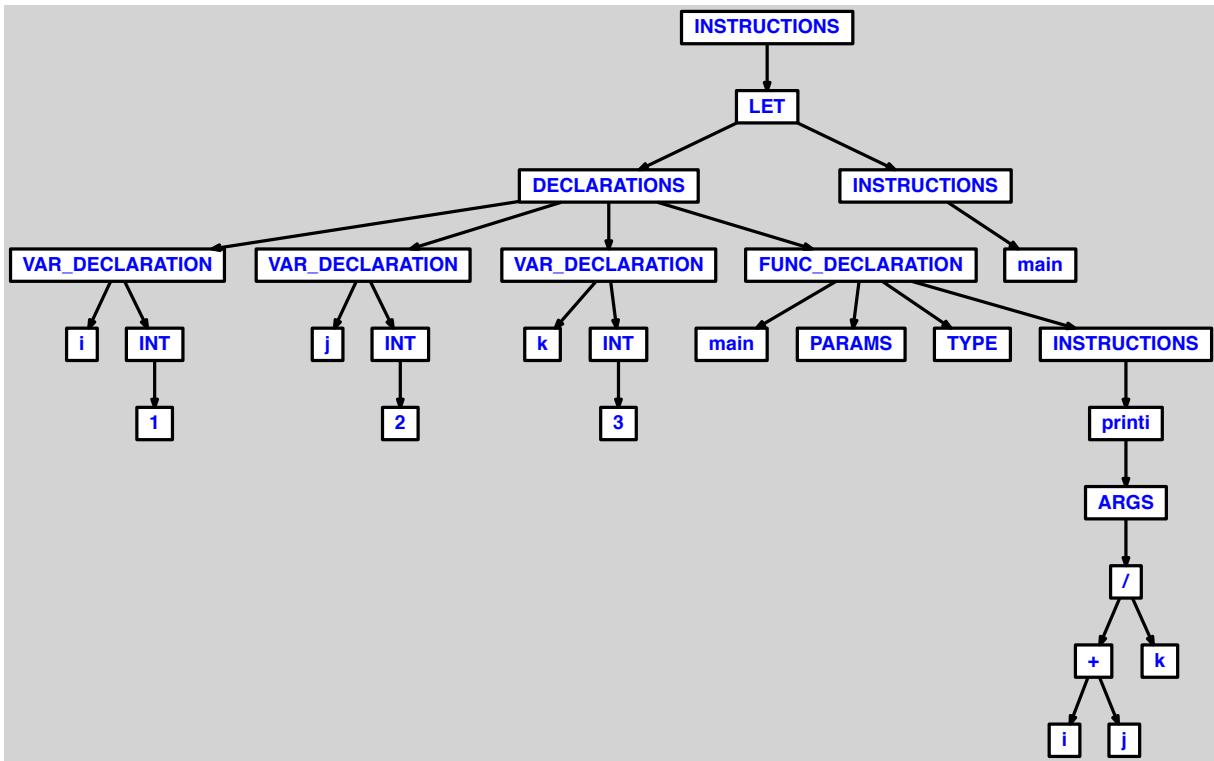
### 3.2.83 addition suivie de multiplication, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i+j)*k)
7 in main() end
```



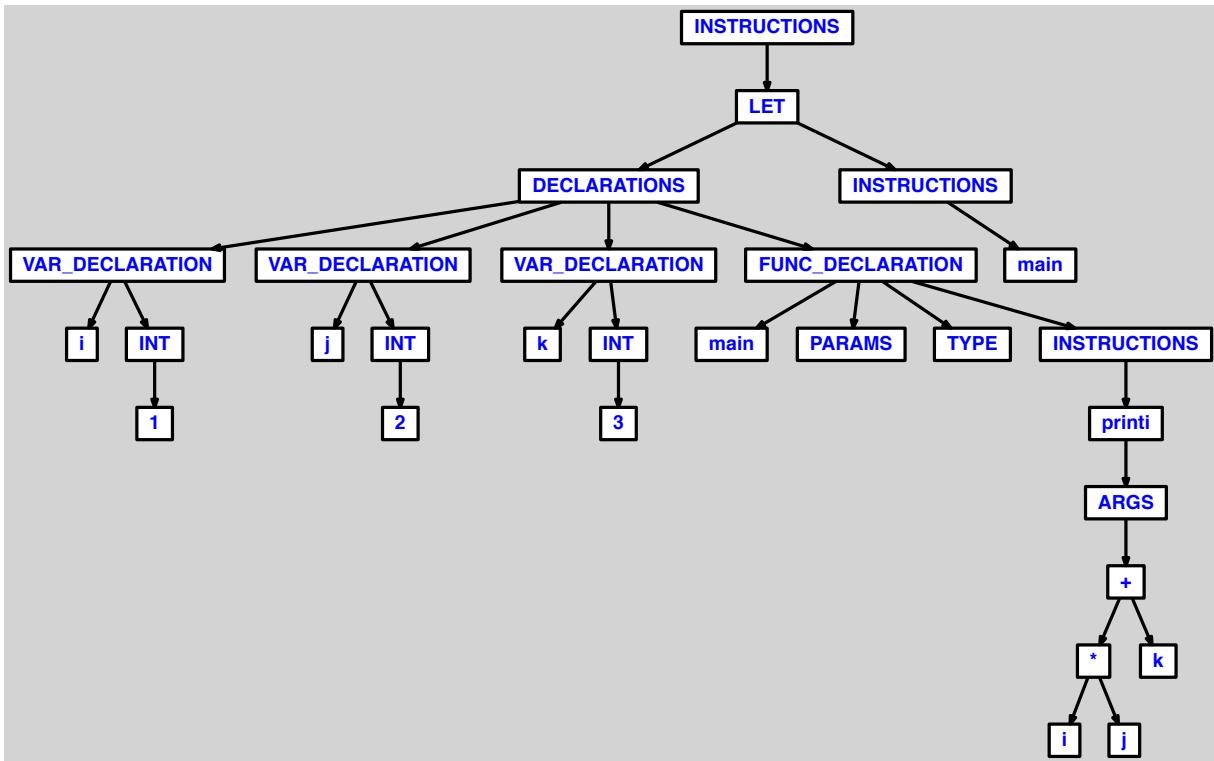
### 3.2.84 addition suivie de division, avec termes identifiés par variables et parenthesage des 2 variables à gauche

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = print((i+j)/k)
7 in main() end
```



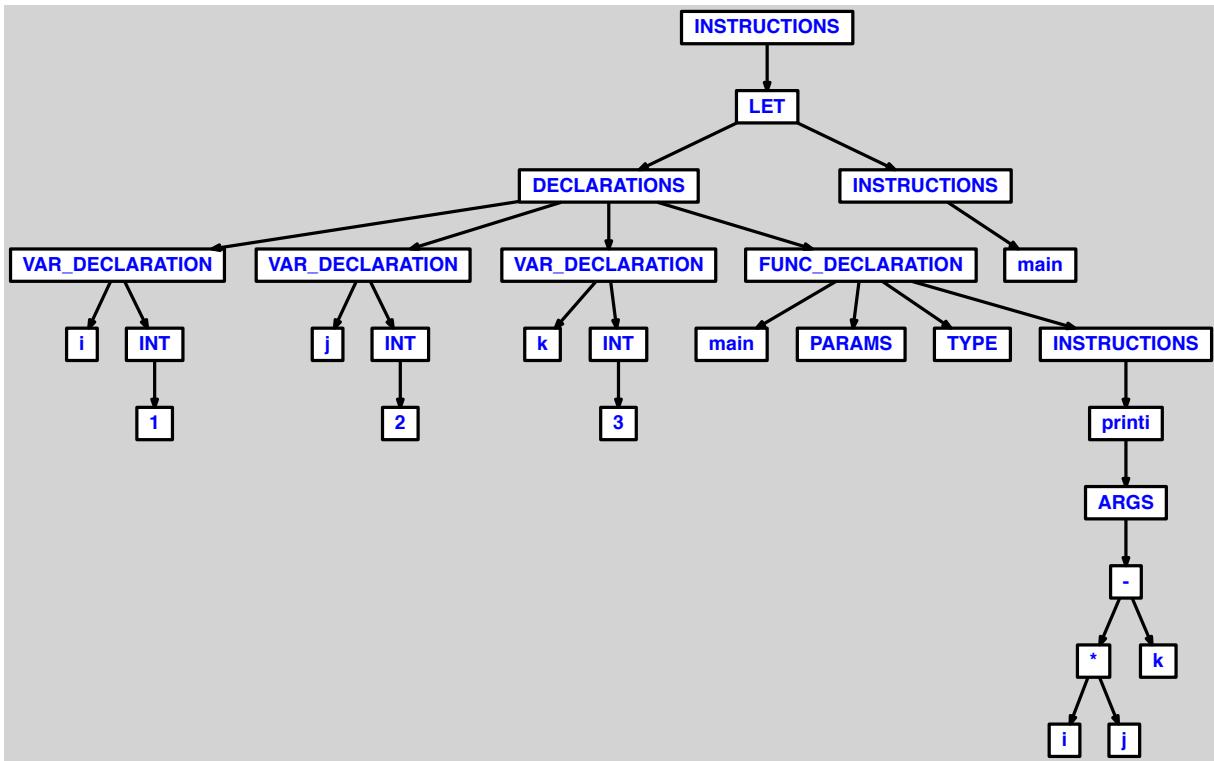
### 3.2.85 multiplication suivie d'addition, avec termes identifies par variables et parenthesage des 2 variables a gauche

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i*j)+k)
7 in main() end
```



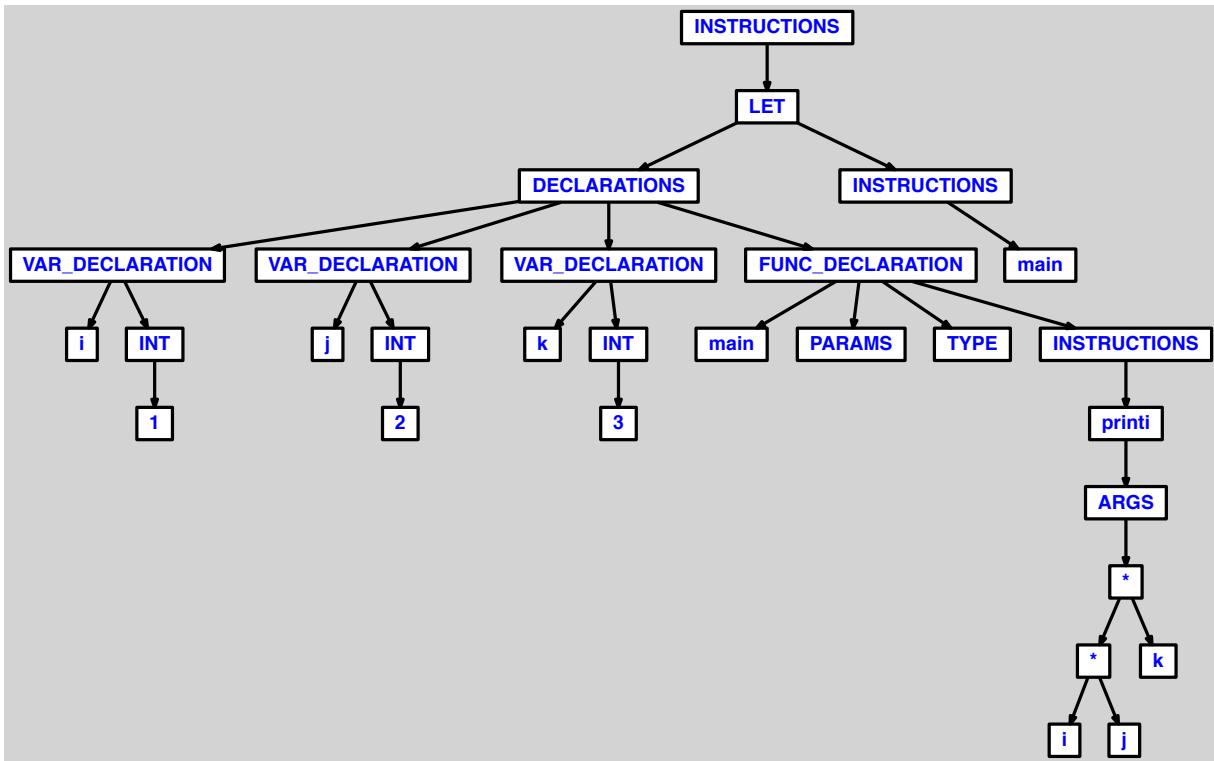
**3.2.86 multiplication suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a gauche**

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = print((i*j)-k)
7 in main() end
```



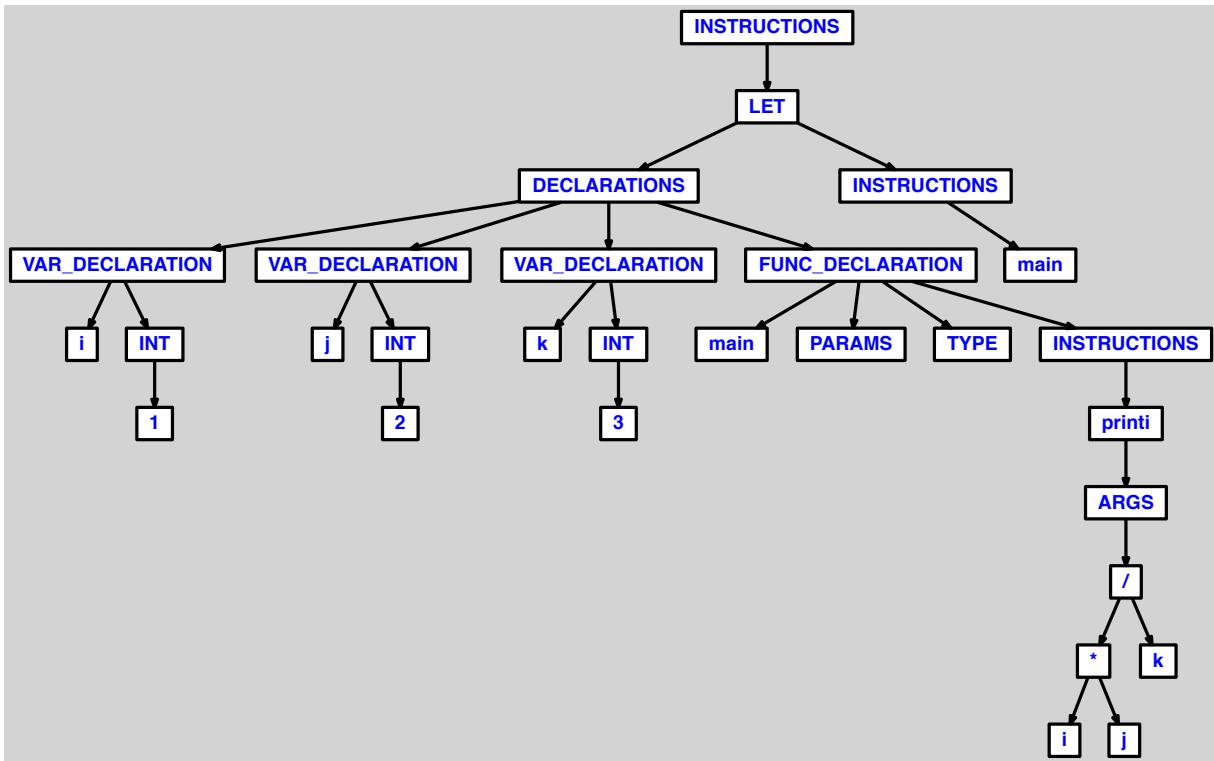
### 3.2.87 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i*j)*k)
7 in main() end
```



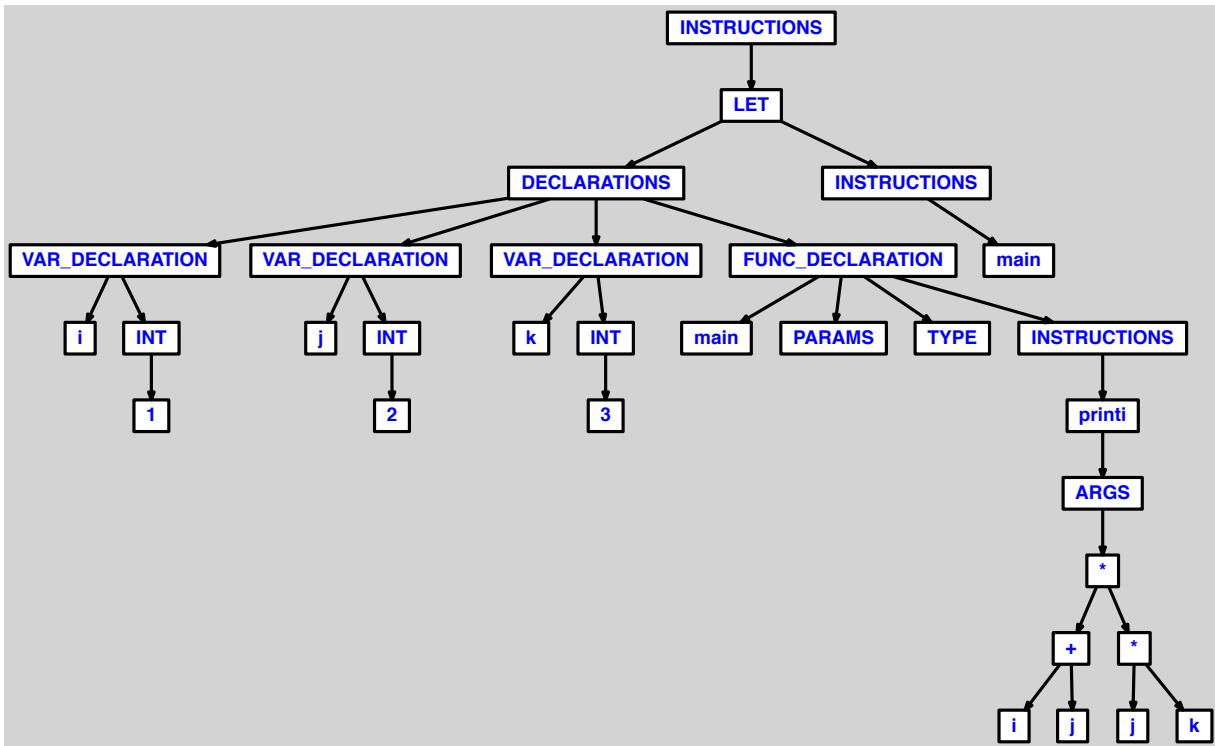
**3.2.88 multiplication suivie de division, avec termes identifies par variables et parenthesage des 2 variables a gauche**

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = print((i*j)/k)
7 in main() end
```



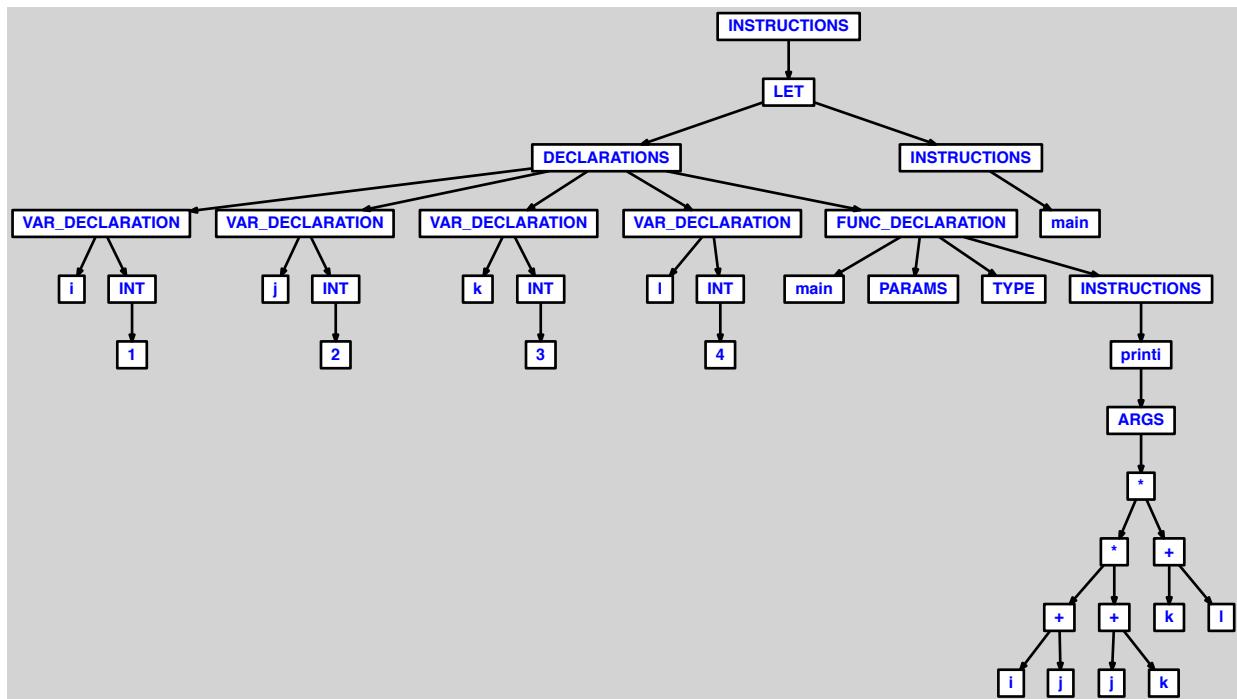
### 3.2.89 multiplication de 2 additions parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i+j)*(j*k))
7 in main() end
```



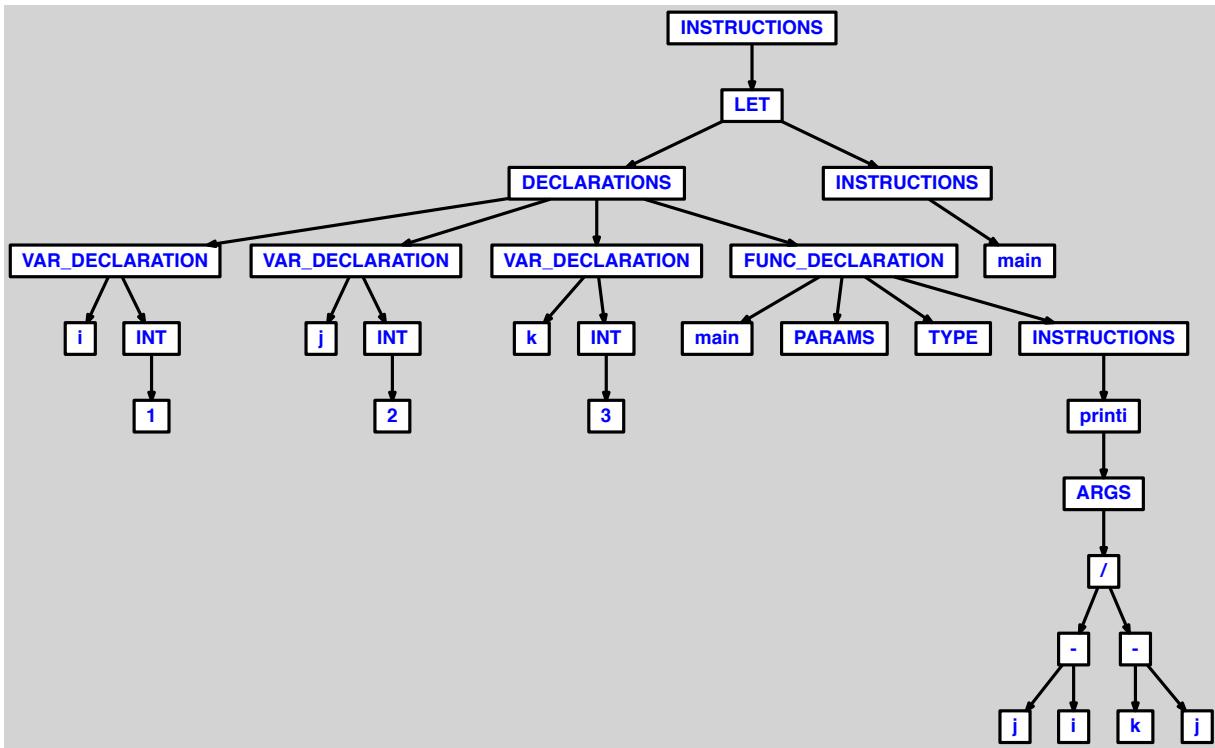
### 3.2.90 multiplication de 3 additions parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5   var l := 4
6
7   function main() = printi((i+j)*(j+k)*(k+l))
8 in main() end
```



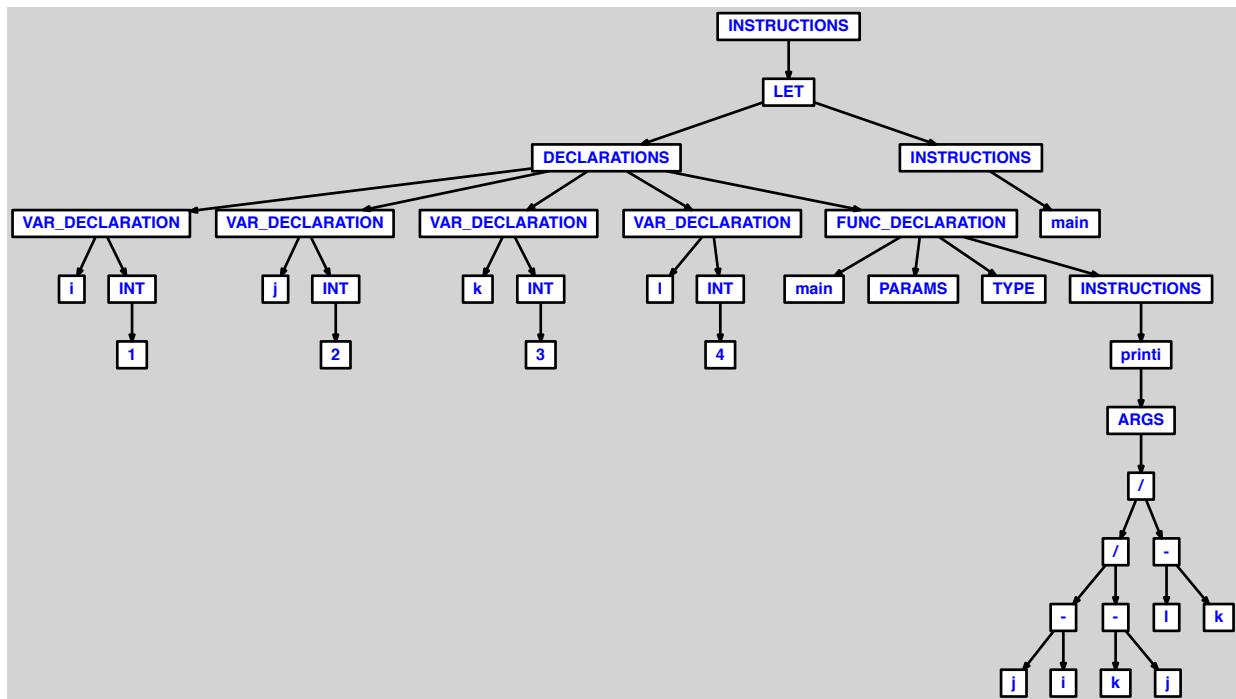
### 3.2.91 division de 2 soustractions parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((j-i)/(k-j))
7 in main() end
```



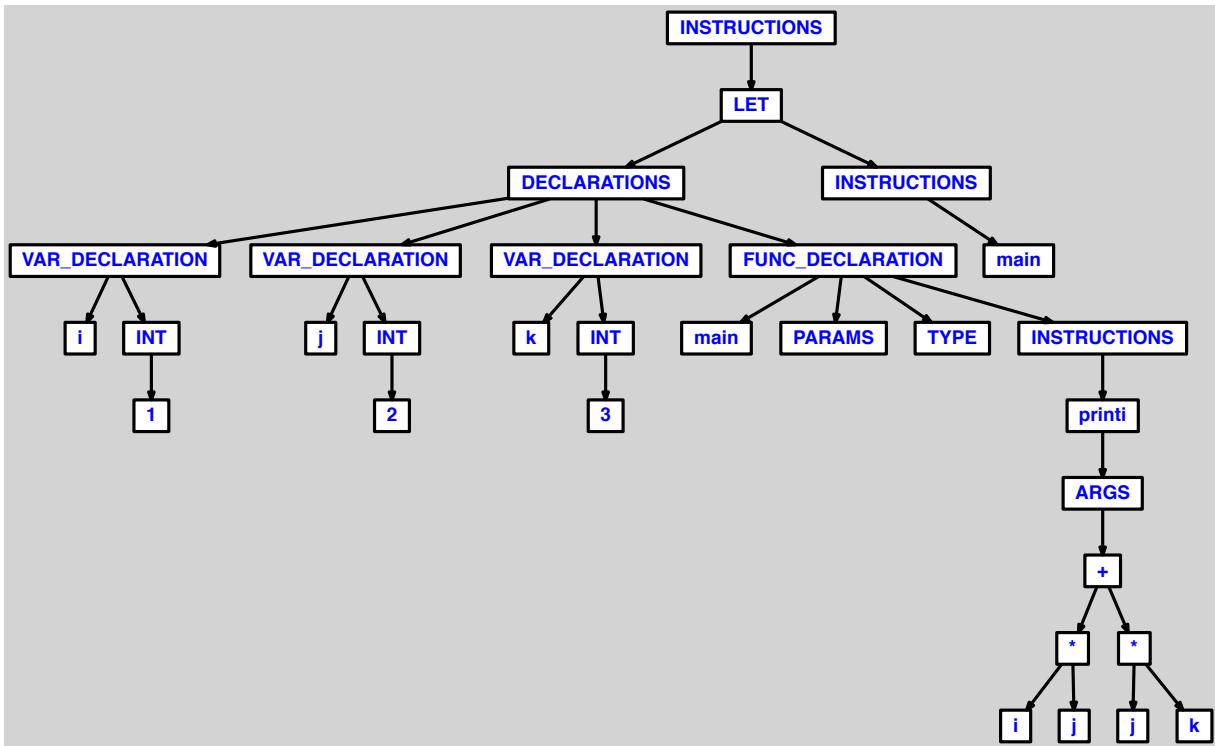
### 3.2.92 division de 3 soustractions parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5   var l := 4
6
7   function main() = printi((j-i)/(k-j)/(l-k))
8 in main() end
```



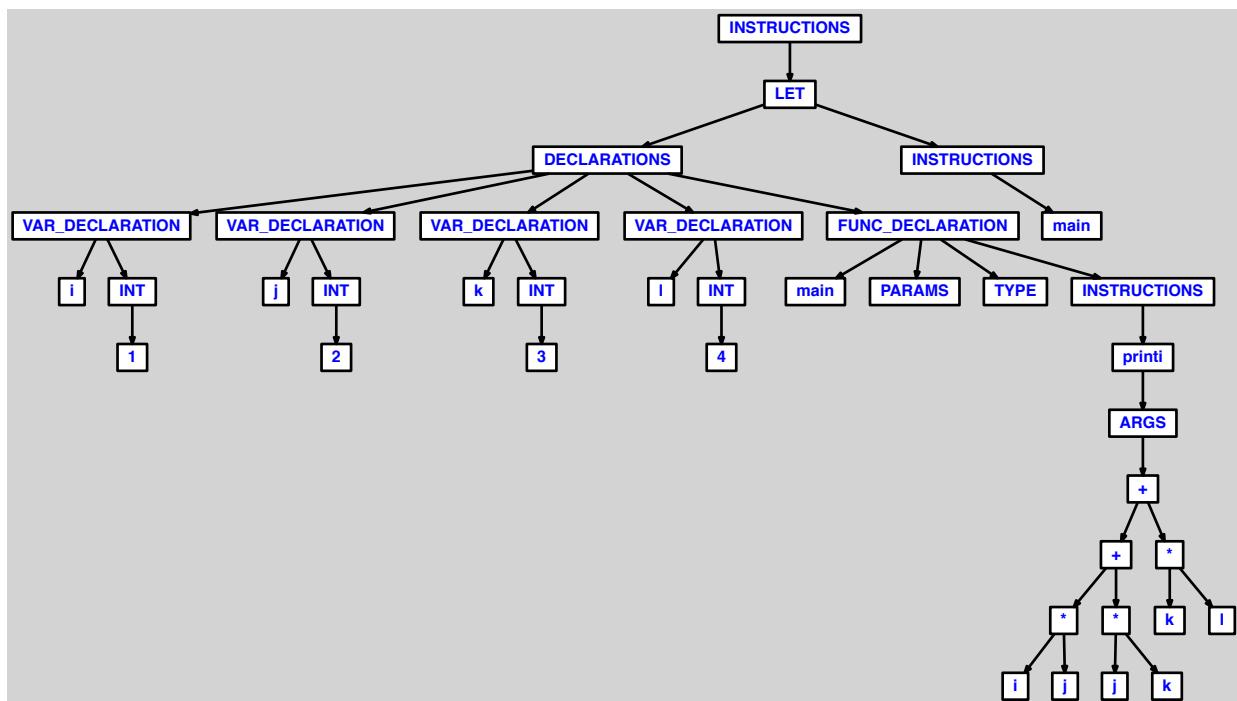
### 3.2.93 addition de 2 multiplications parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5
6   function main() = printi((i*j)+(j*k))
7 in main() end
```



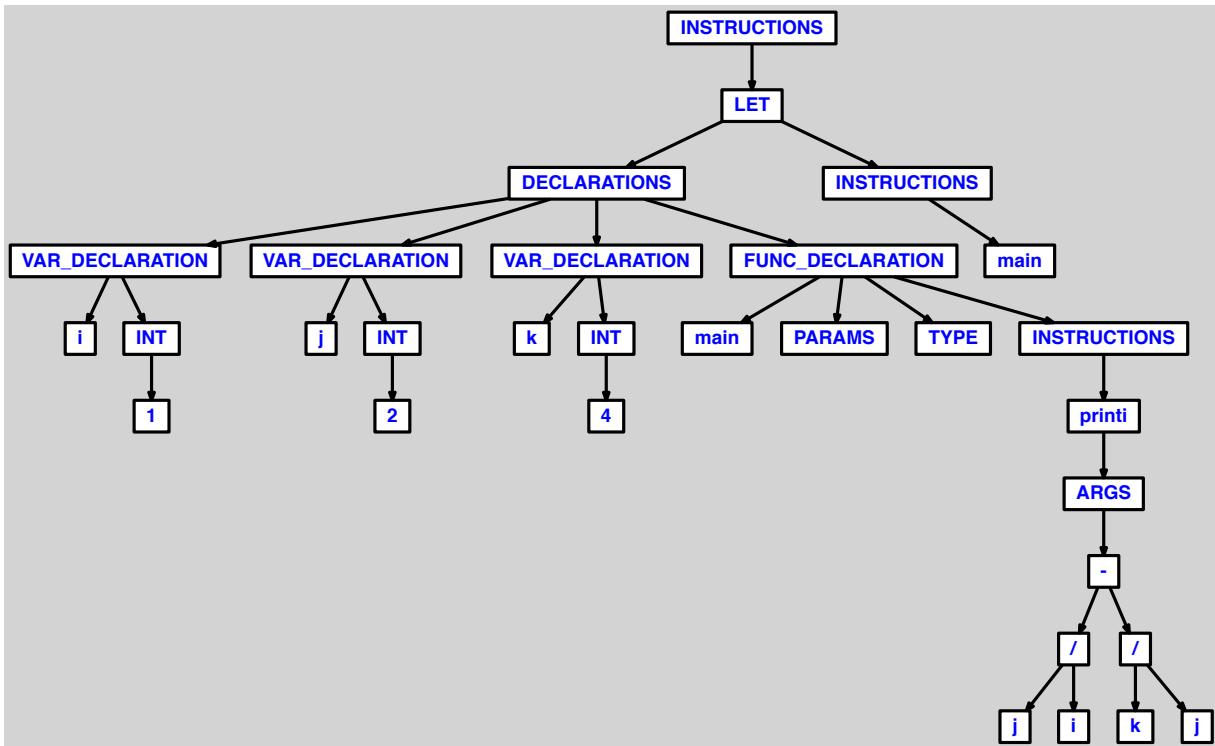
### 3.2.94 addition de 3 multiplications parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 3
5   var l := 4
6
7   function main() = printi((i*j)+(j*k)+(k*l))
8 in main() end
```



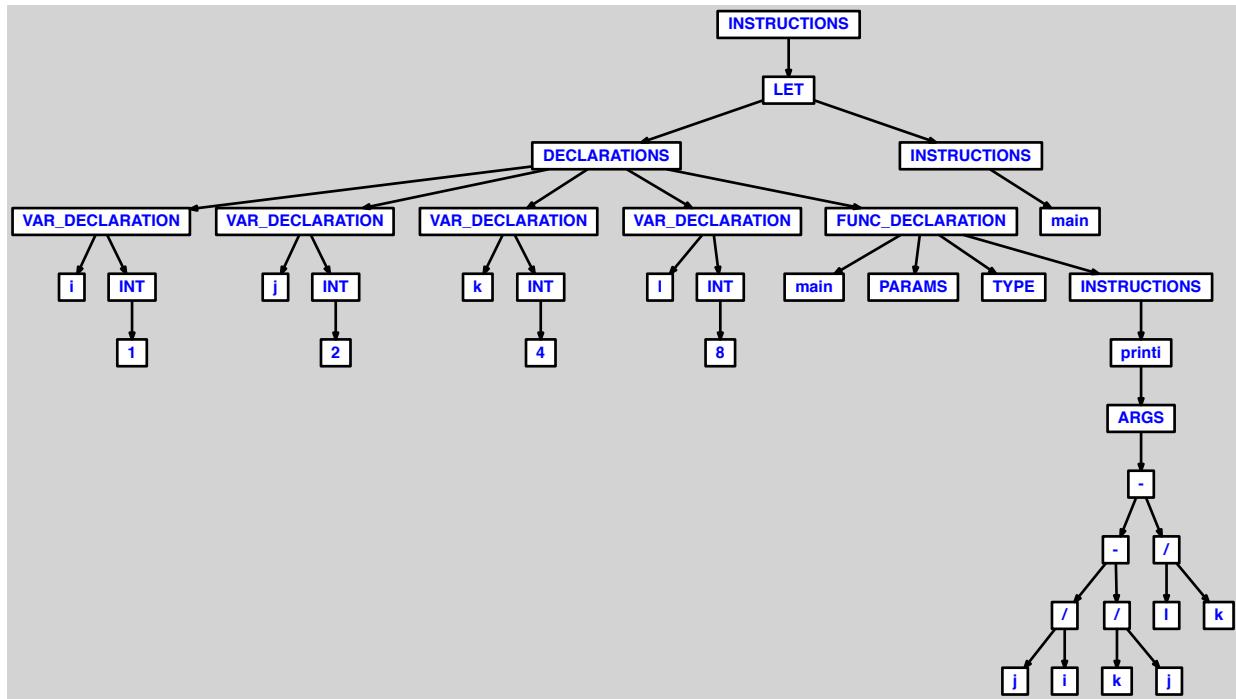
### 3.2.95 soustraction de 2 divisions parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 4
5
6   function main() = printi((j/i)-(k/j))
7 in main() end
```



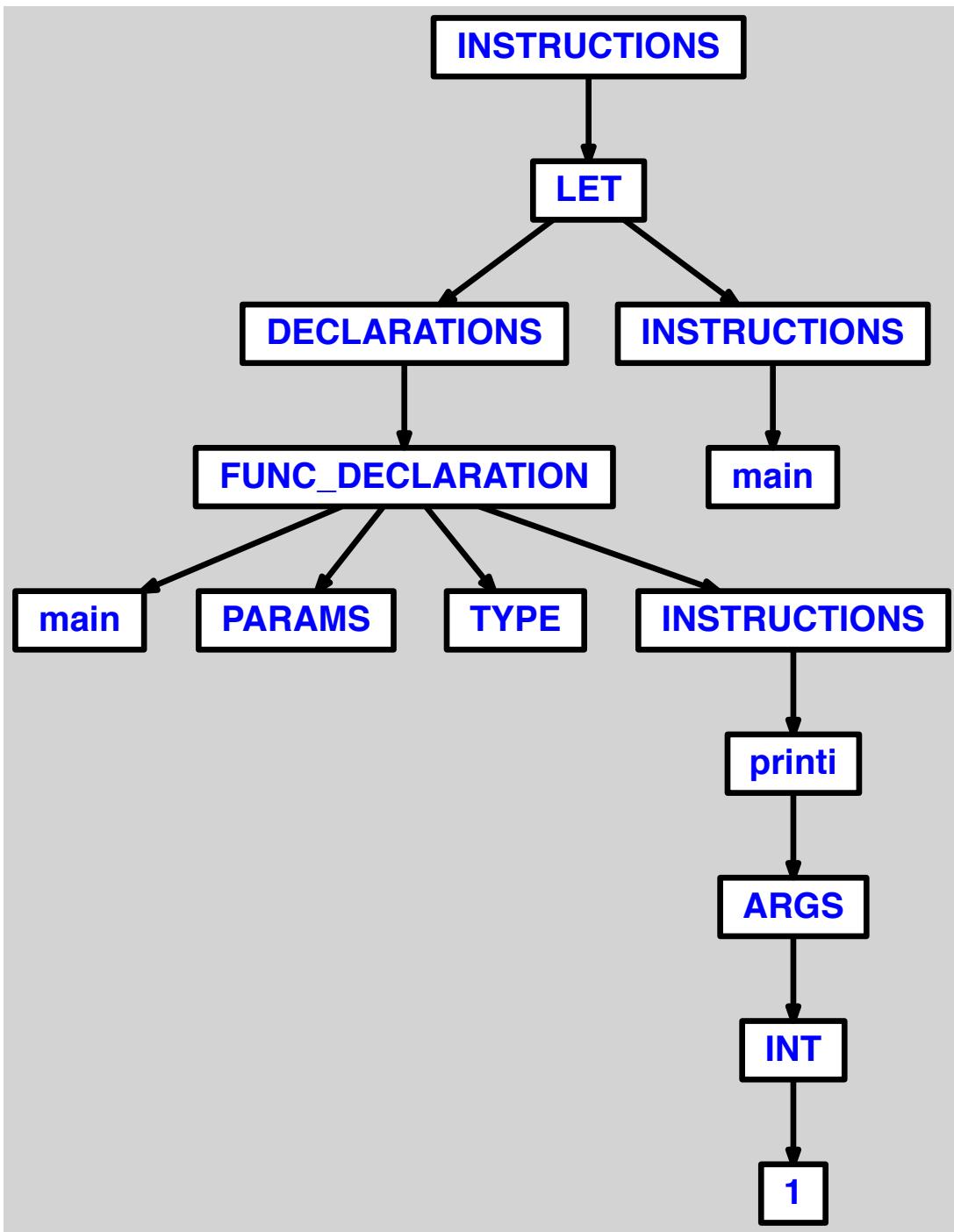
### 3.2.96 soustraction de 3 divisions parenthesees, avec termes identifies par variables

```
1 let
2   var i := 1
3   var j := 2
4   var k := 4
5   var l := 8
6
7   function main() = printi((j/i)-(k/j)-(l/k))
8 in main() end
```



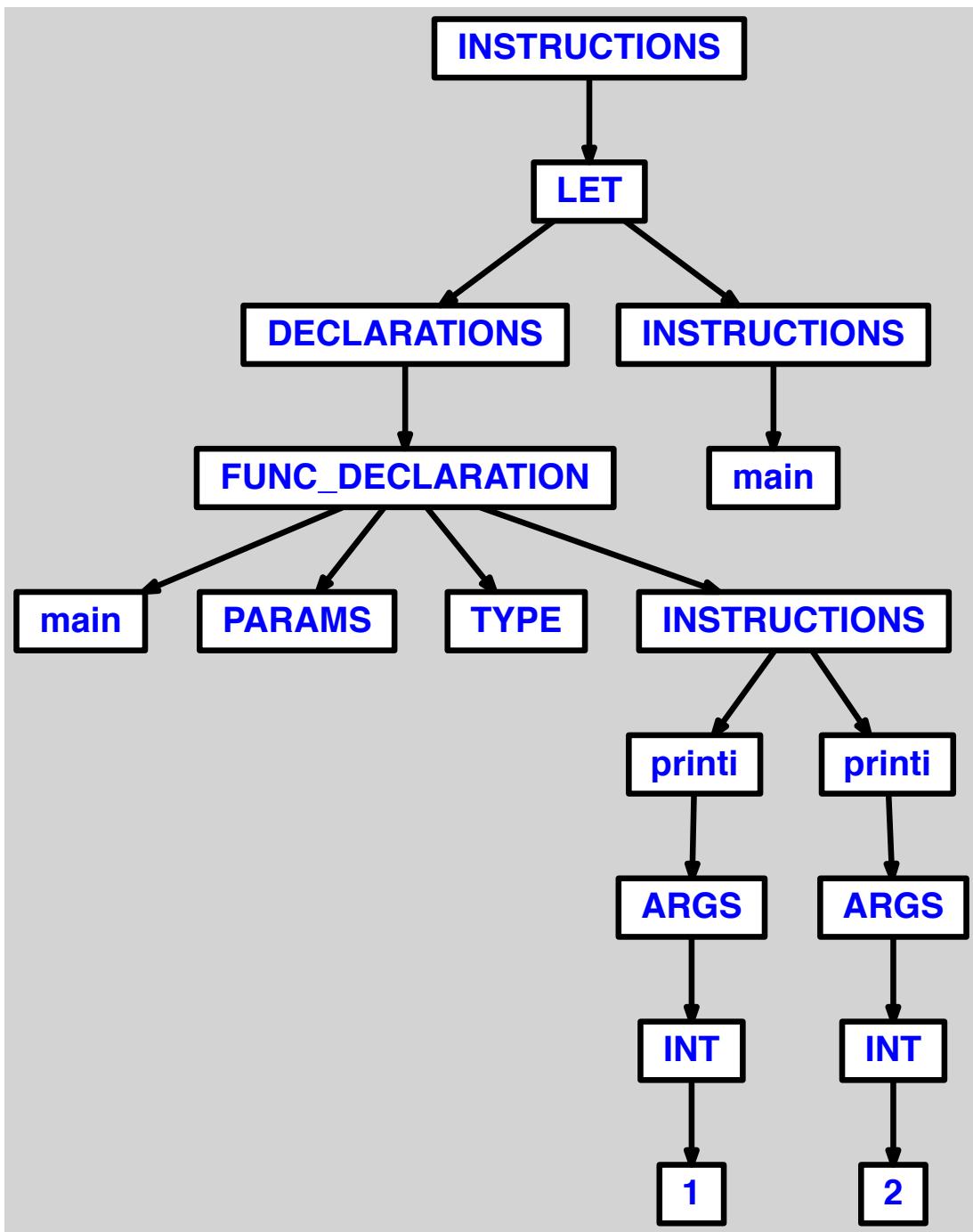
### 3.2.97 parenthesage d'une instruction

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



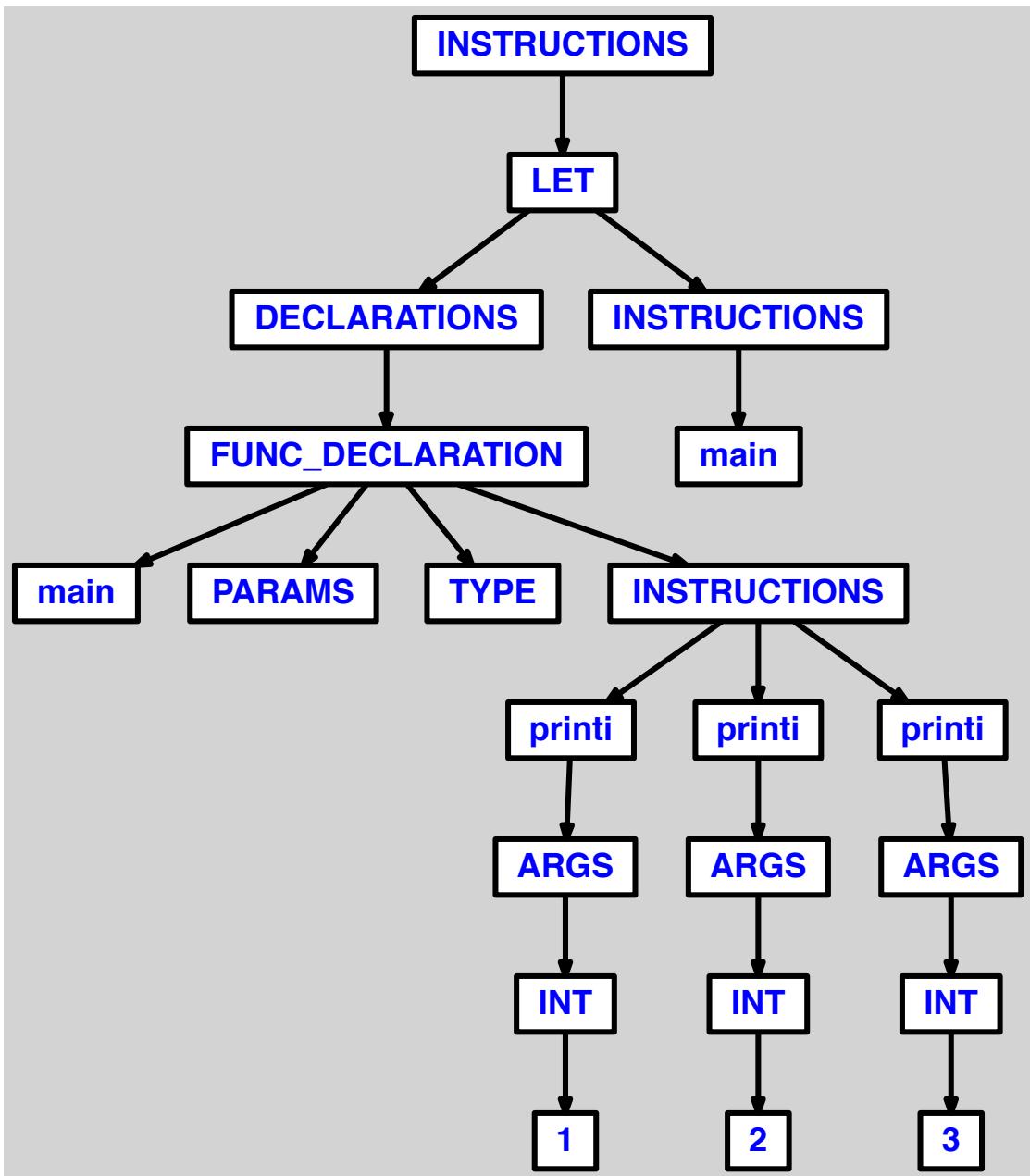
### 3.2.98 parenthesage de 2 instructions

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



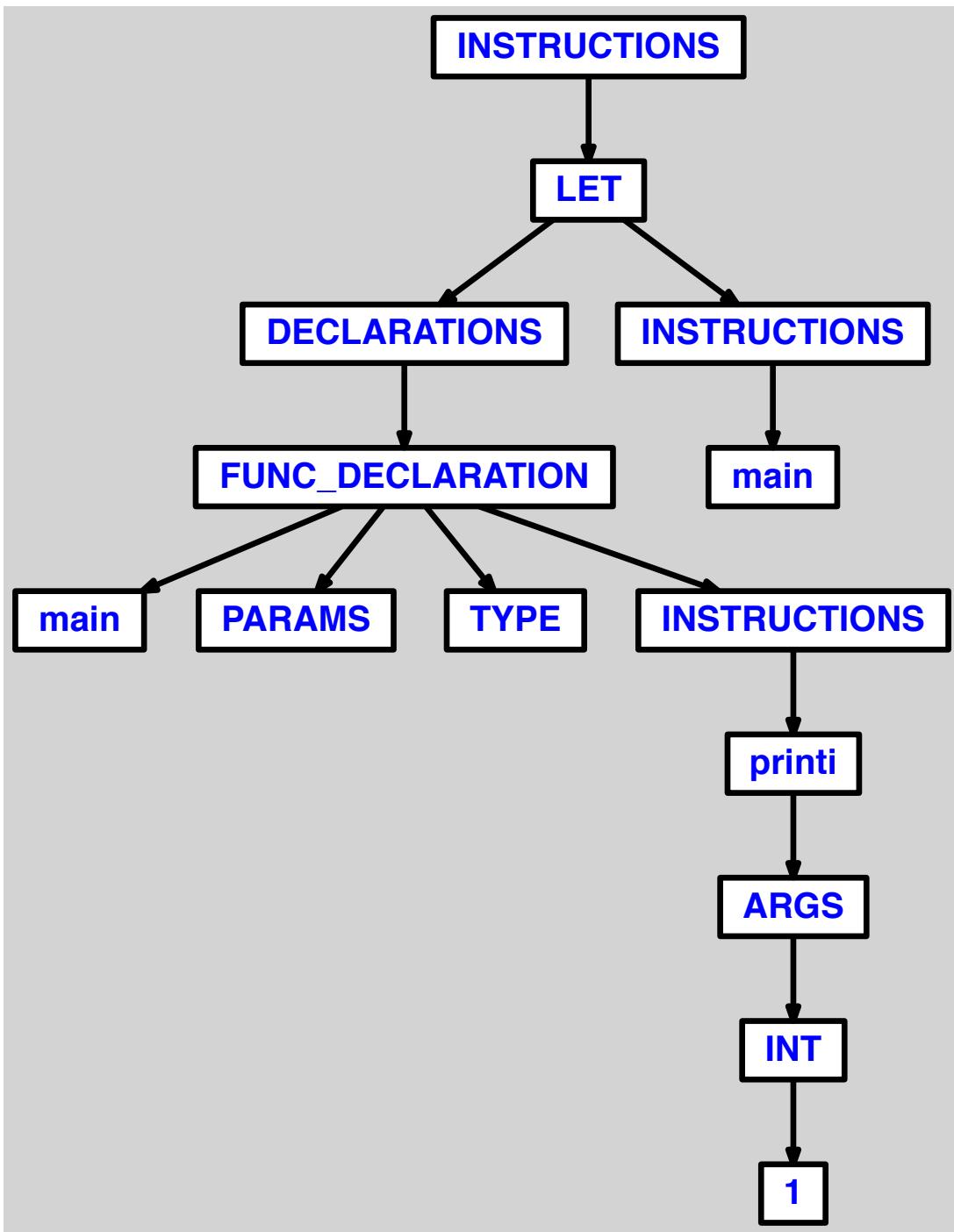
### 3.2.99 parenthesage de 3 instructions

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



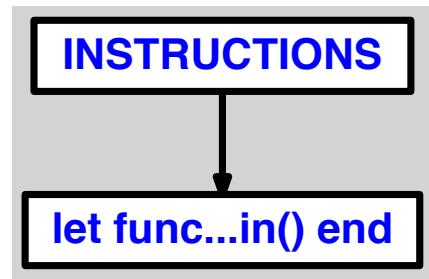
### 3.2.100 1 instruction sans parenthèses

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



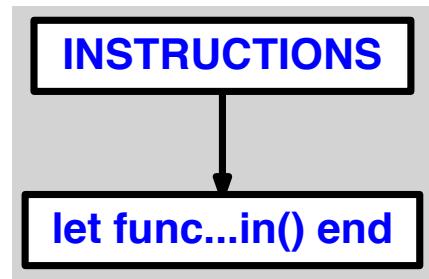
### 3.2.101 2 instructions sans parenthèses

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



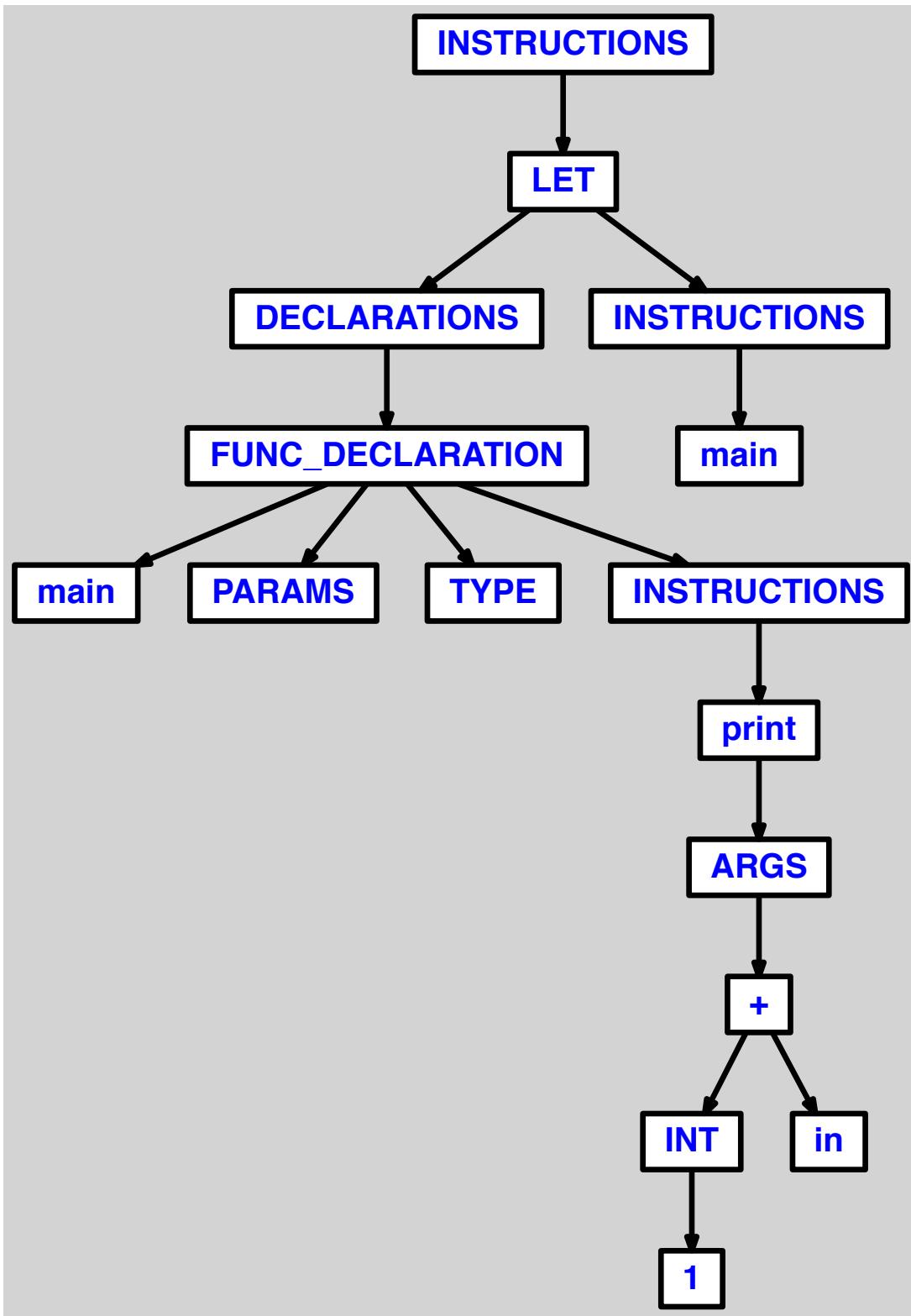
### 3.2.102 3 instructions sans parenthèses

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



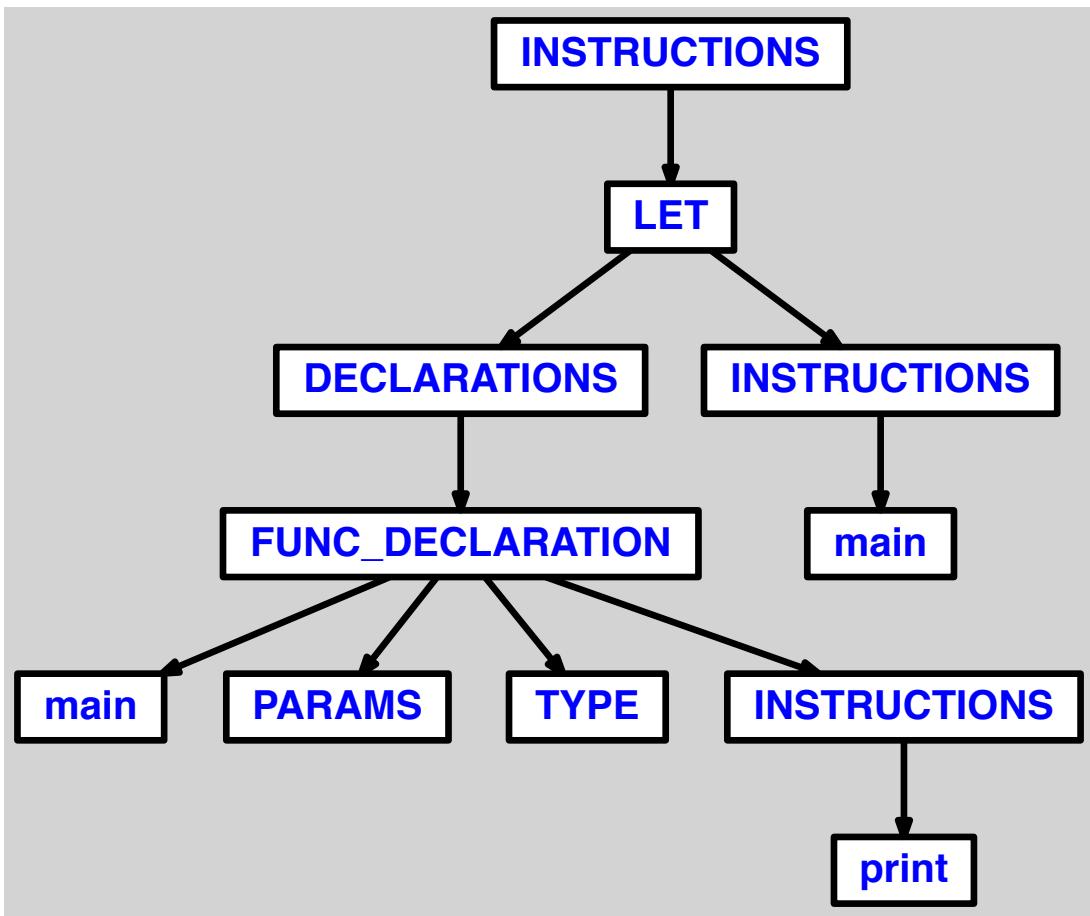
### 3.2.103 concatenation d'entier et de chaine

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



### 3.2.104 concatenation de 2 chaines

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

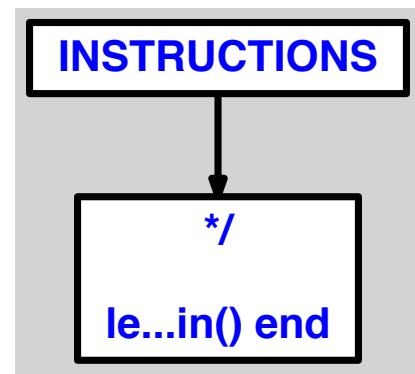


## 4 commentaire

### 4.1 KO

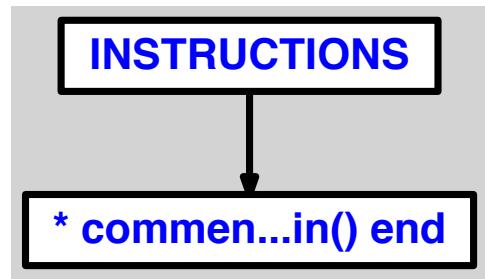
#### 4.1.1 imbrication

```
1  /*
2   * un commentaire
3   * /* imbrique dans ce commentaire */
4   */
5
6 let
7   function main() = print("test")
8 in main() end
```



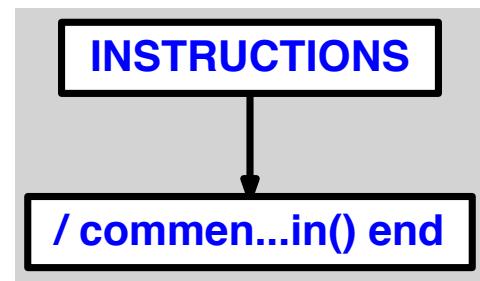
#### 4.1.2 oubli de </> en debut de commentaire

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



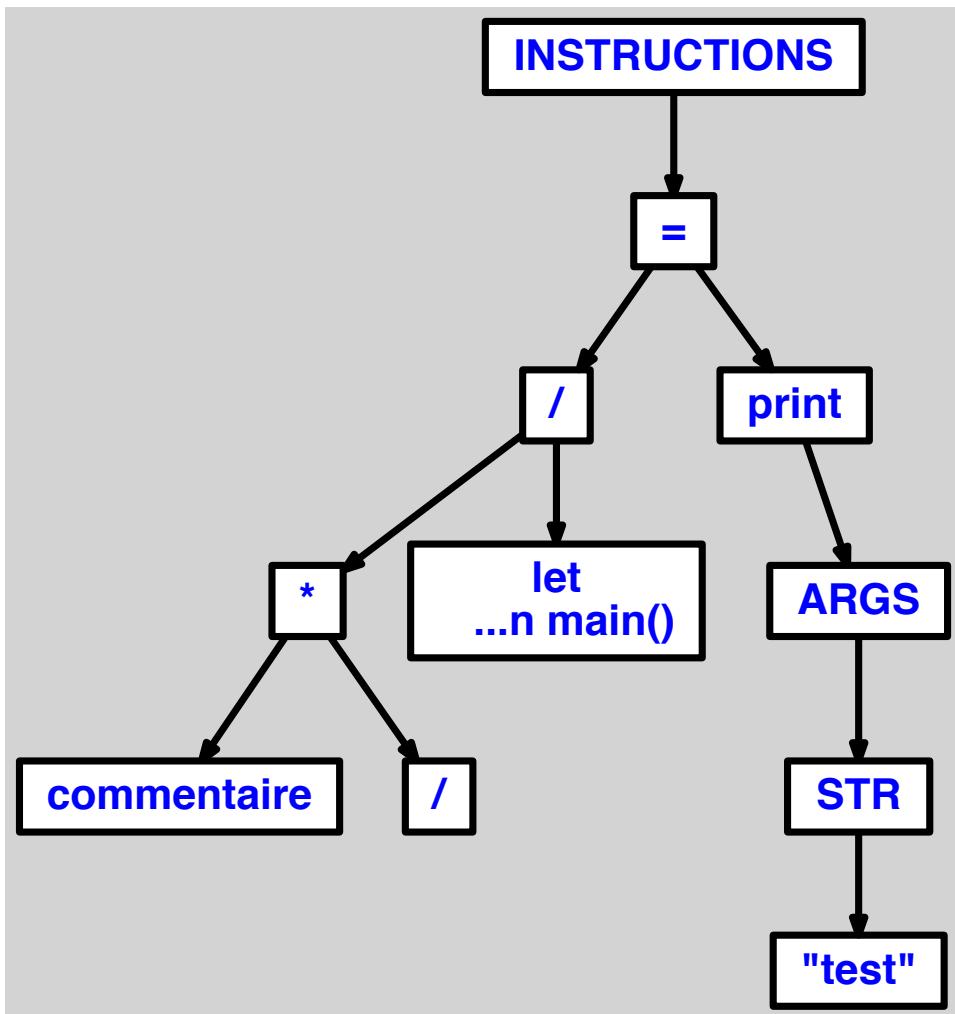
#### 4.1.3 oubli de <\*> en debut de commentaire

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



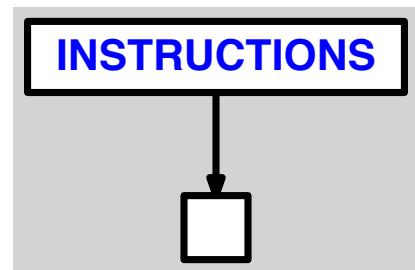
#### 4.1.4oubli de <\*> en debut de commentaire

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



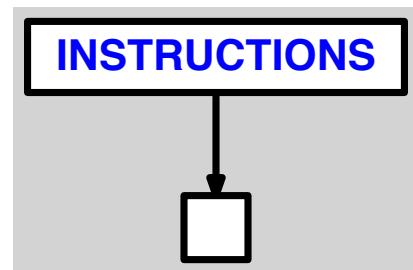
#### 4.1.5oubli de <\*> en fin de commentaire

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



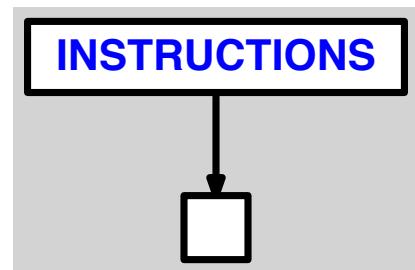
#### 4.1.6oubli de </> en fin de commentaire

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



#### 4.1.7oubli de <\*>/> en fin de commentaire

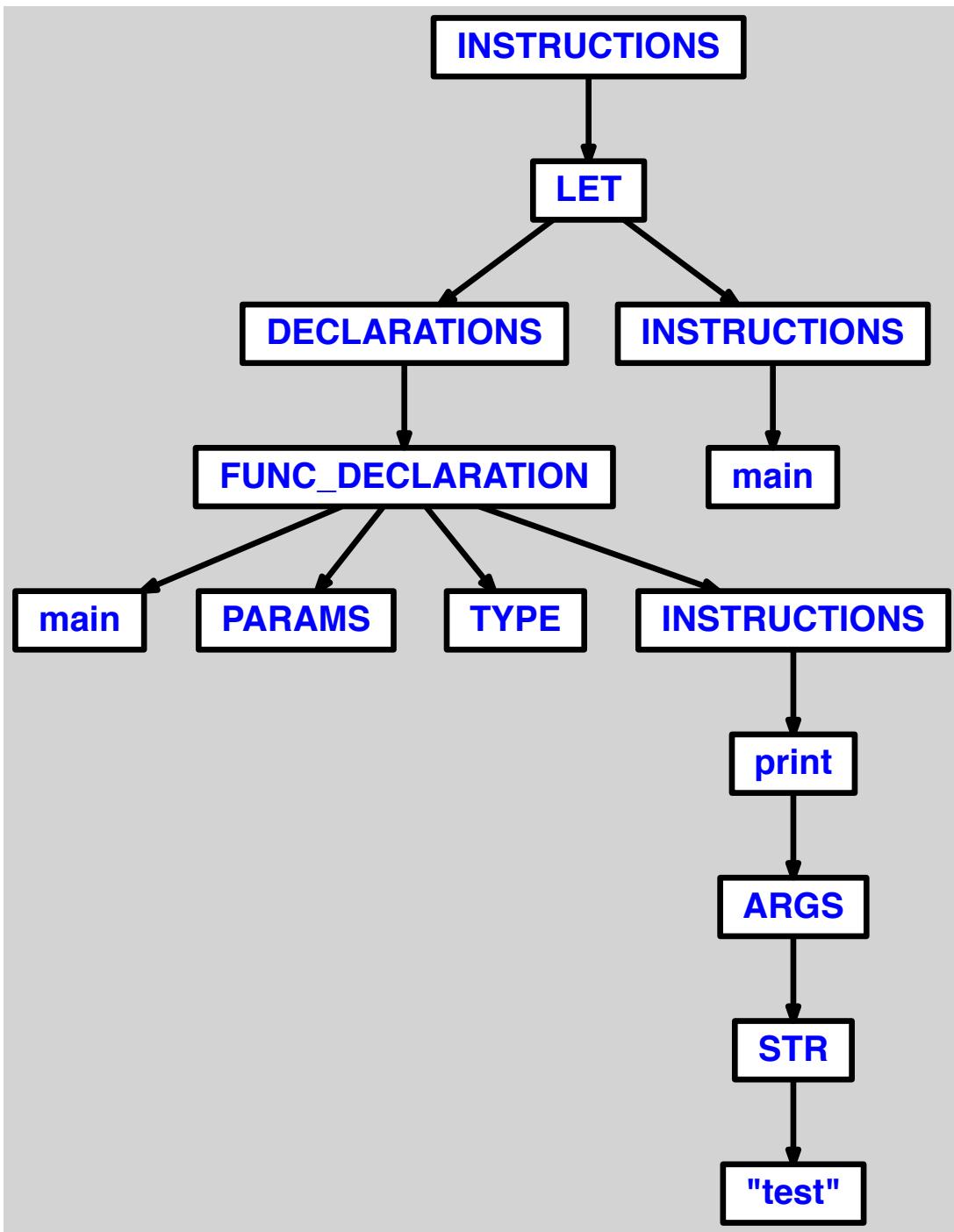
```
1 /* commentaire
2
3 let
4     function main() = print("test")
5 in main() end
```



## 4.2 OK

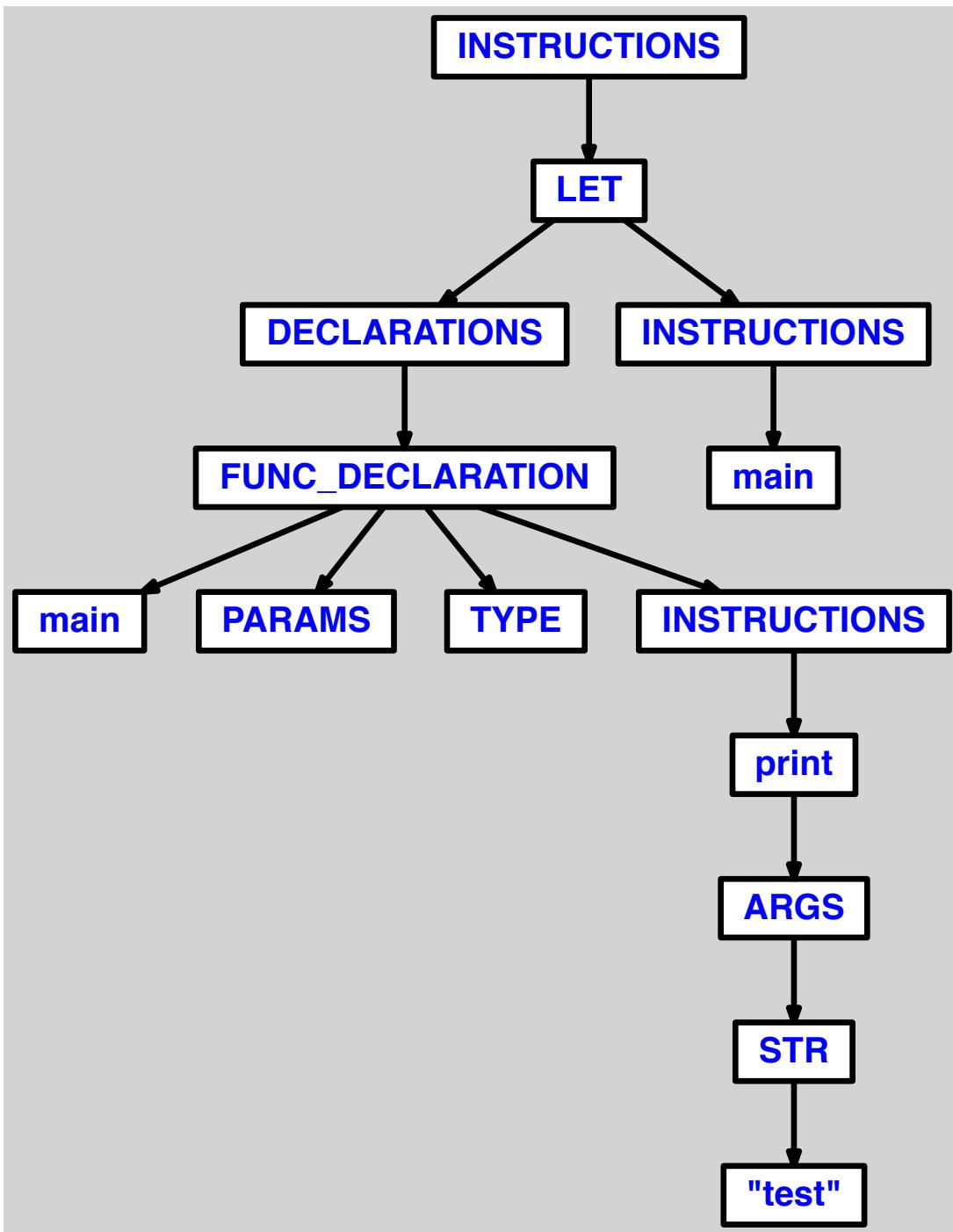
### 4.2.1 1 commentaire, sans instruction avant, sur 1 ligne

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



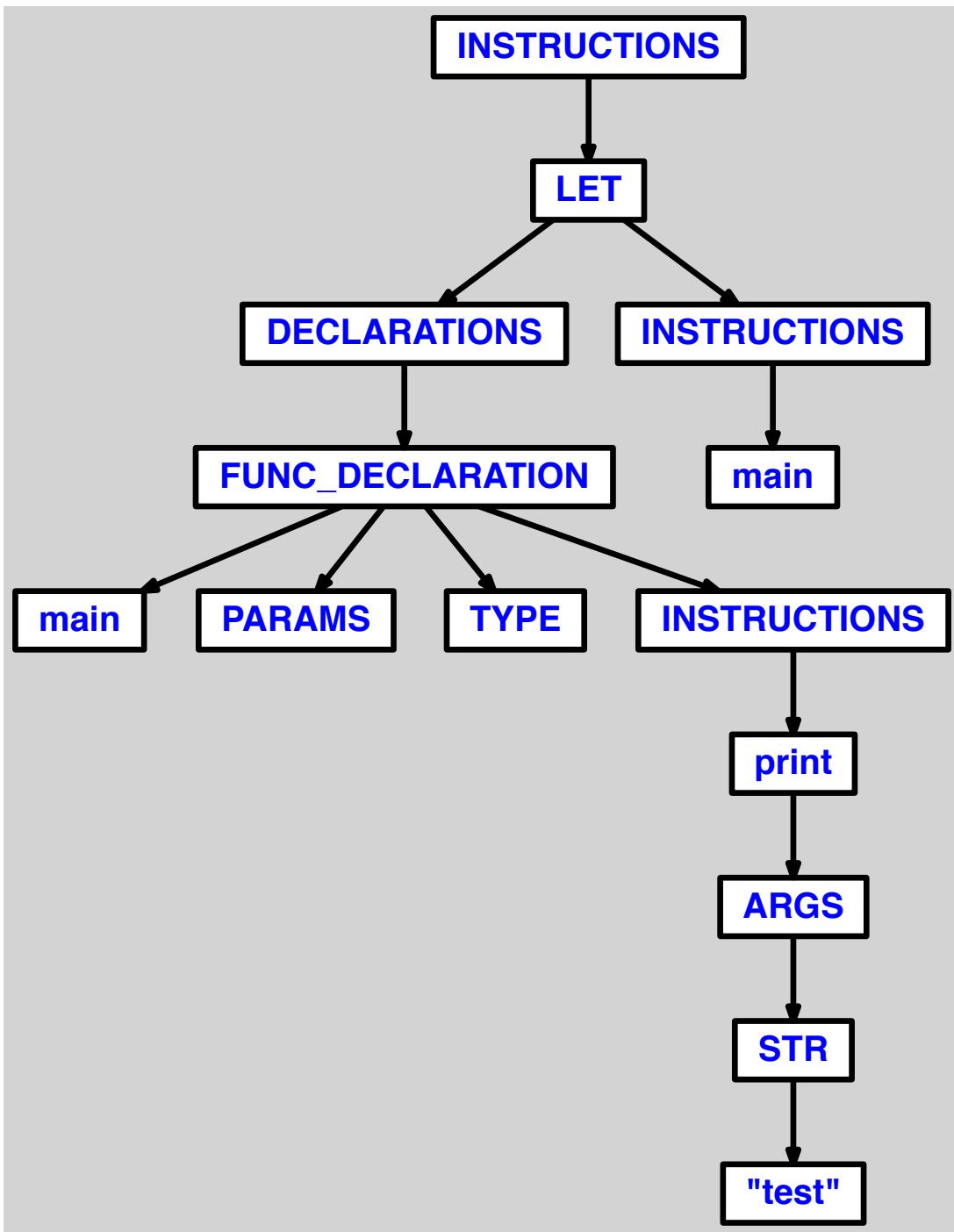
#### 4.2.2 1 commentaire, sans instruction avant, sur plusieurs lignes

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



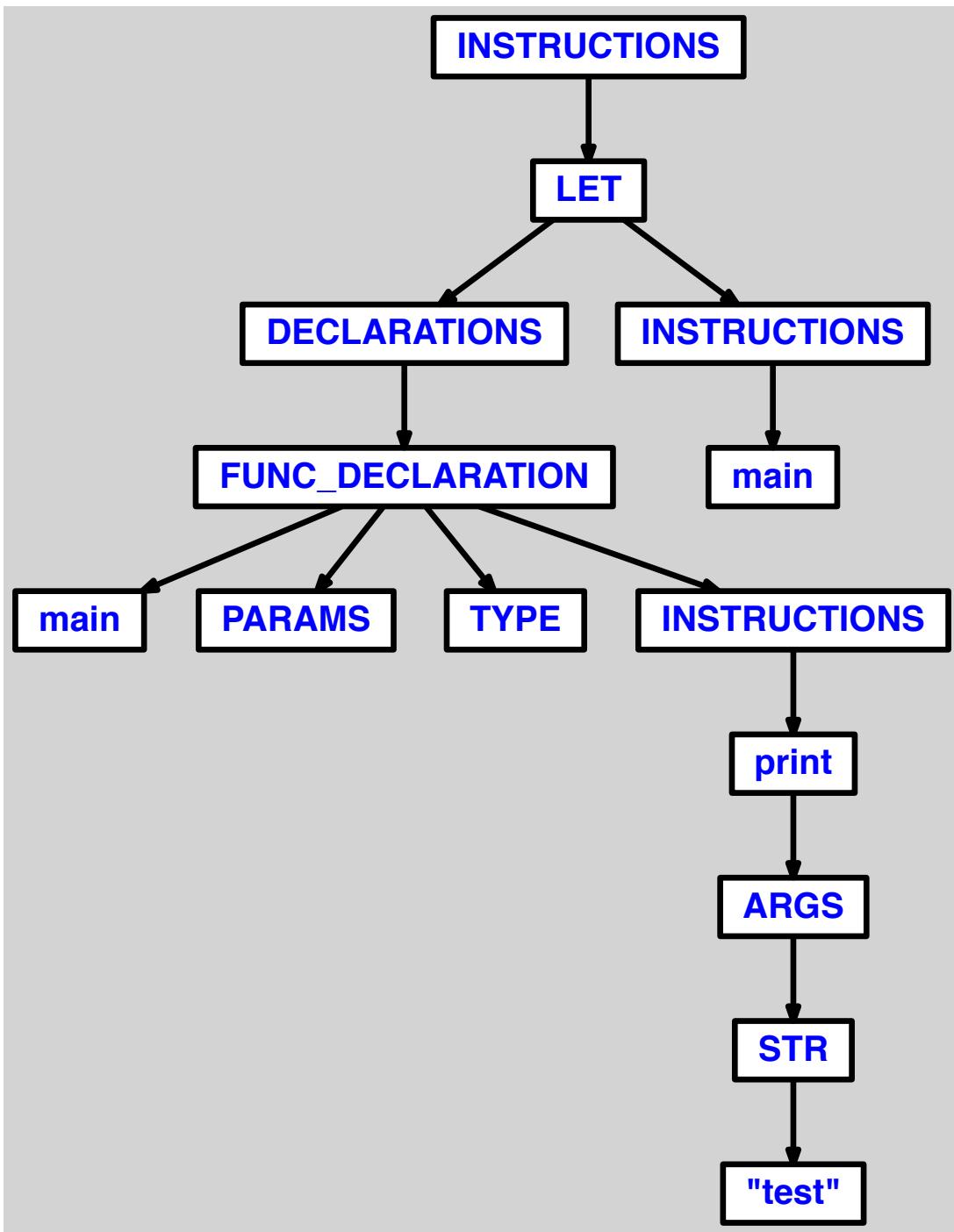
#### 4.2.3 1 commentaire, apres instruction, sur 1 ligne

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



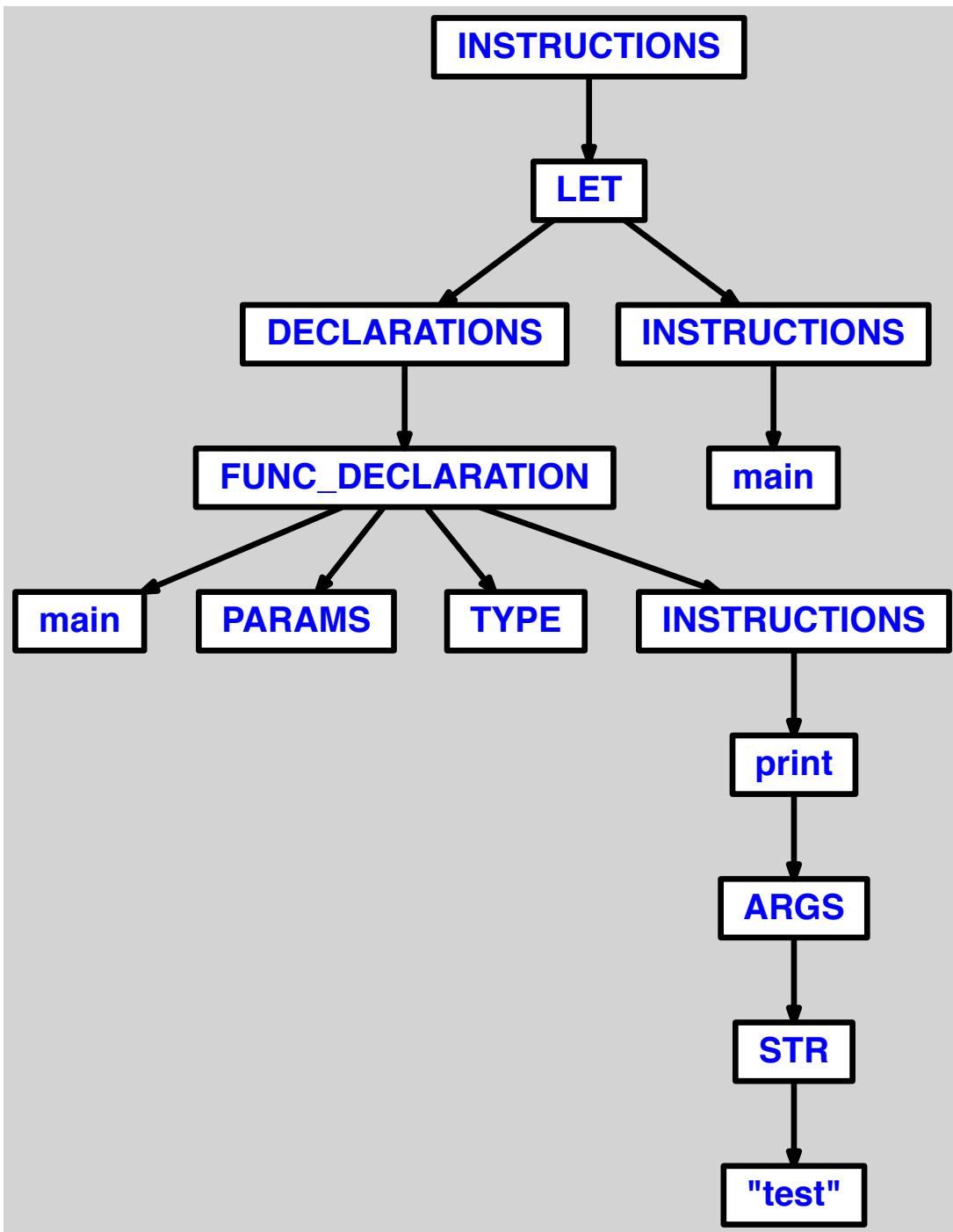
#### 4.2.4 1 commentaire, apres instruction, sur plusieurs lignes

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



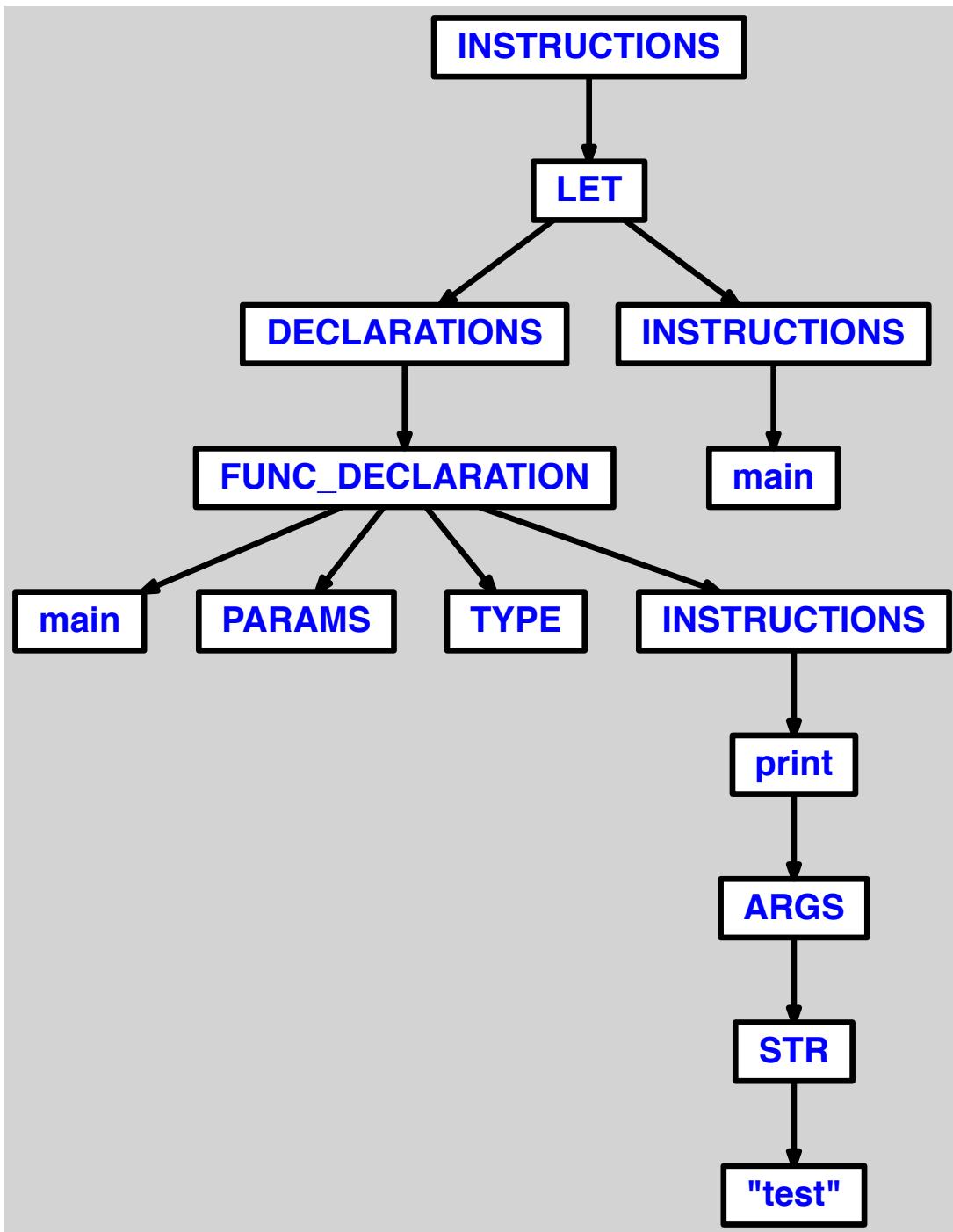
#### 4.2.5 2 commentaires, sans instruction avant, sur 1 ligne

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



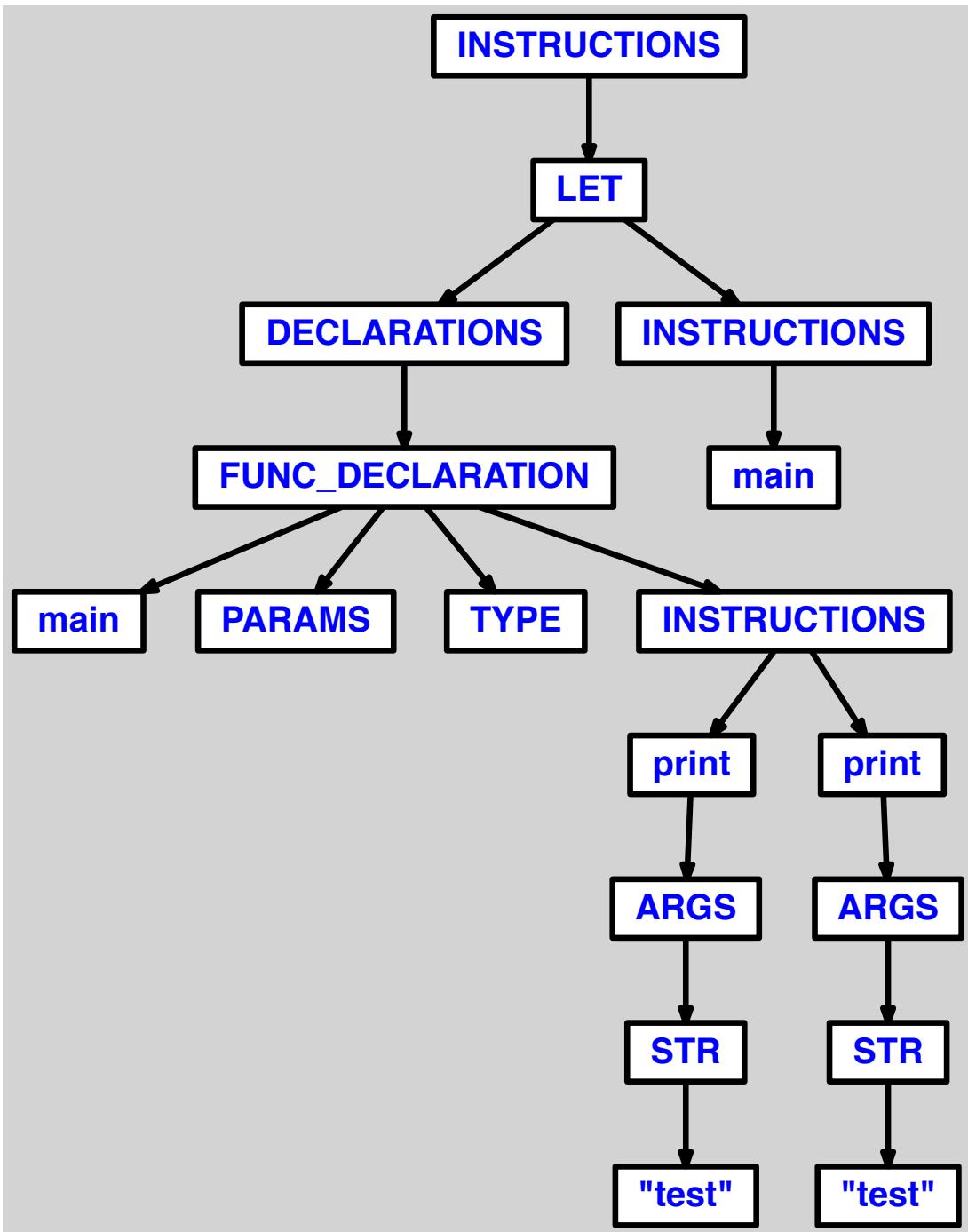
#### 4.2.6 2 commentaires, sans instruction avant, sur plusieurs lignes

```
1  /*
2  1
3  2
4  3
5  */
6
7  /*
8  4
9  5
10 6
11 */
12
13 let
14     function main() = print("test")
15 in main() end
```



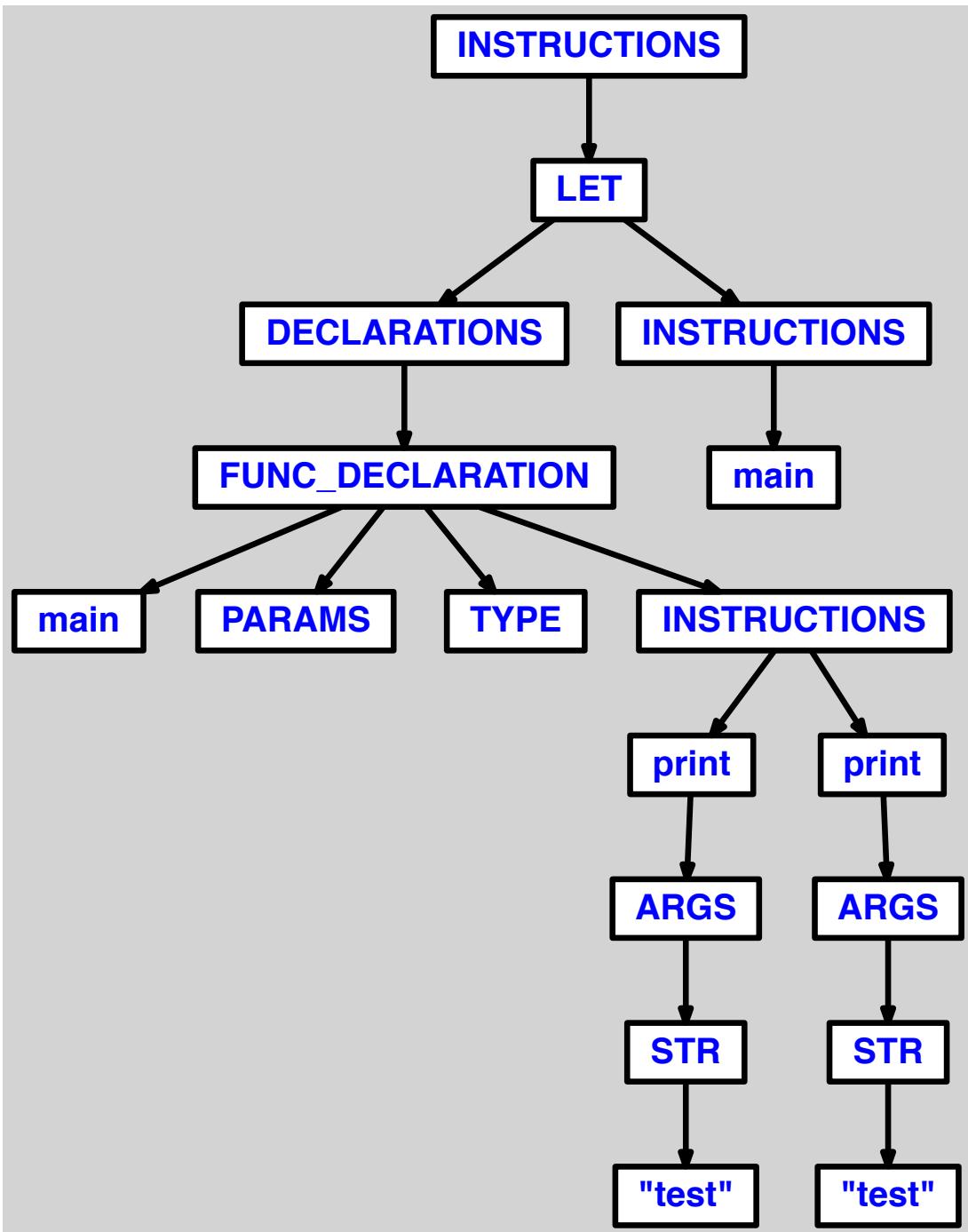
#### 4.2.7 2 commentaires, apres instructions, sur 1 ligne

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



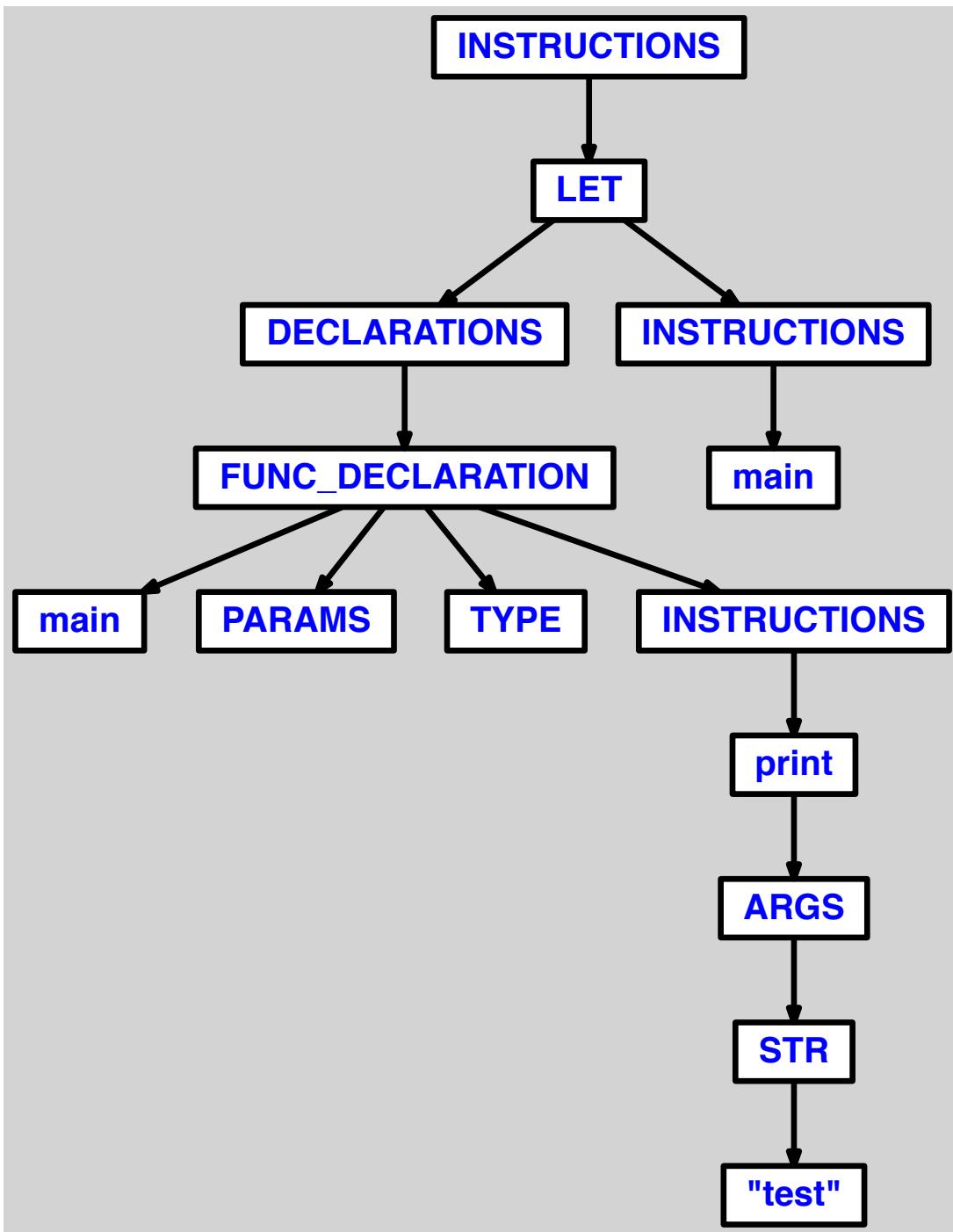
#### 4.2.8 2 commentaires, apres instructions, sur plusieurs lignes

```
1 let
2   function main() =
3     (print("test");
4     /*
5      1
6      2
7      3
8     */
9     print("test")
10    /*
11     4
12     5
13     6
14    */
15   )
16 in main() end
```



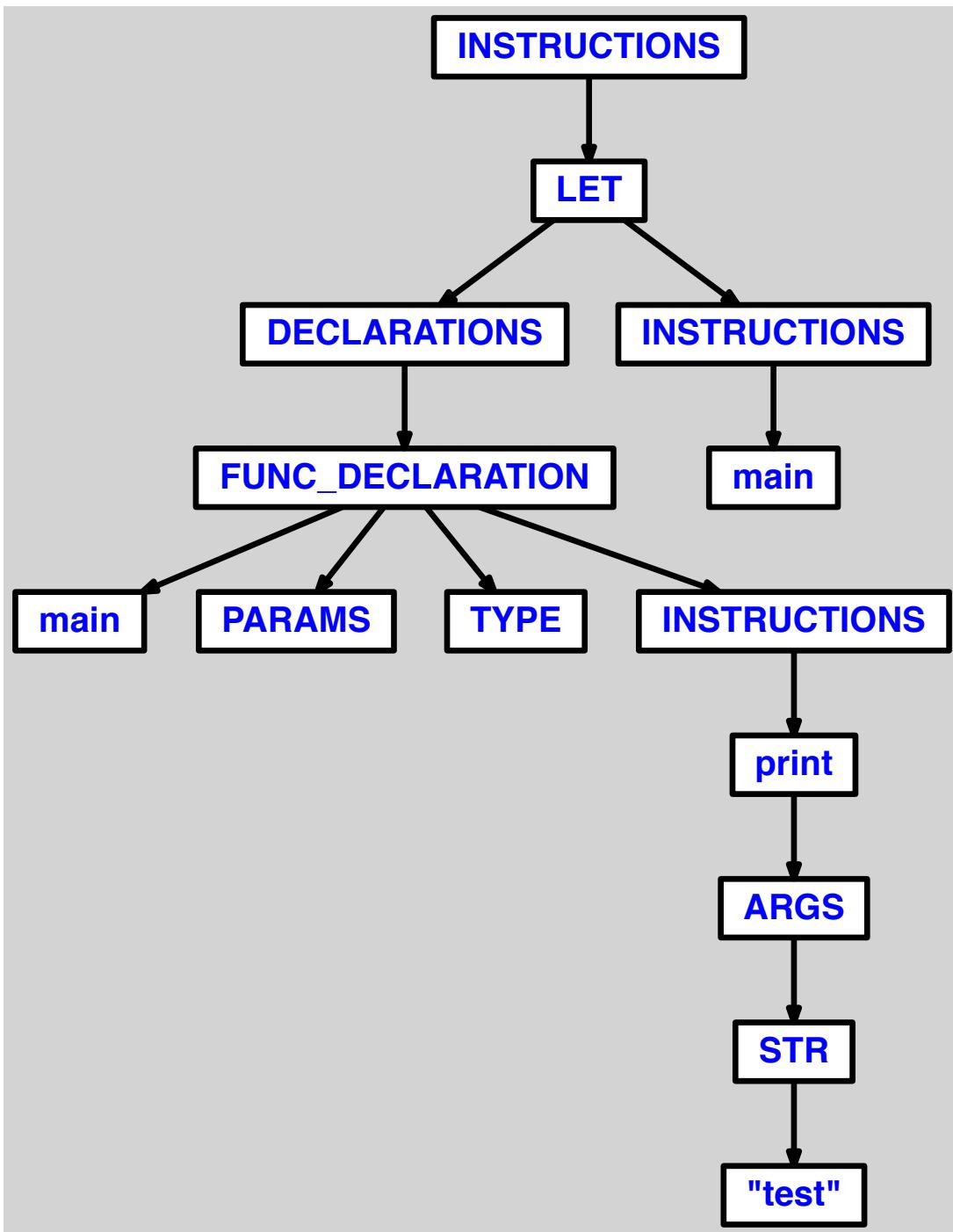
#### 4.2.9 3 commentaires, sans instruction avant, sur 1 ligne

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



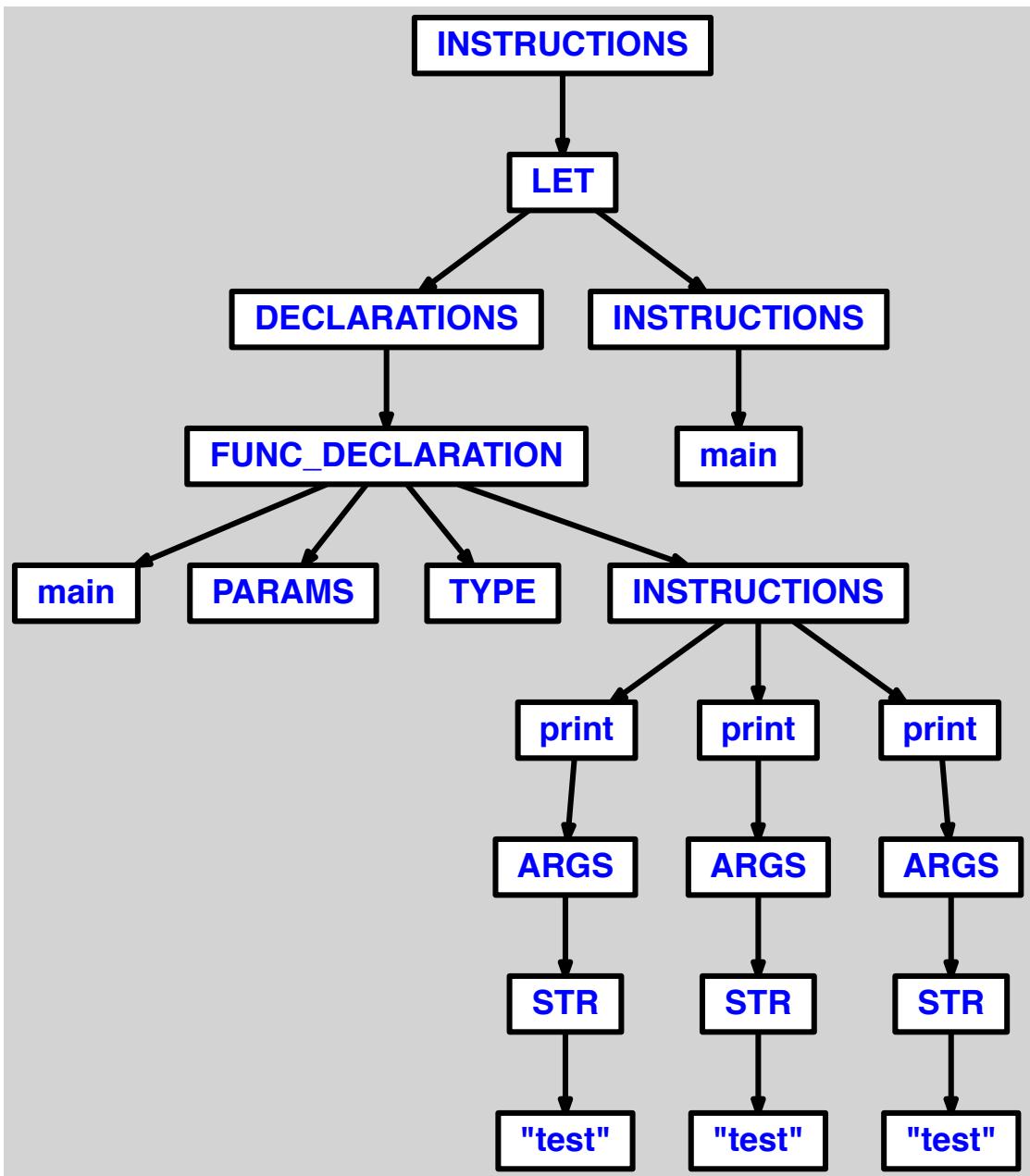
#### 4.2.10 3 commentaires, sans instruction avant, sur plusieurs lignes

```
1  /*
2   1
3   2
4   3
5   */
6
7  /*
8   4
9   5
10  6
11 */
12
13 /*
14  7
15  8
16  9
17 */
18
19 let
20     function main() = print("test")
21 in main() end
```



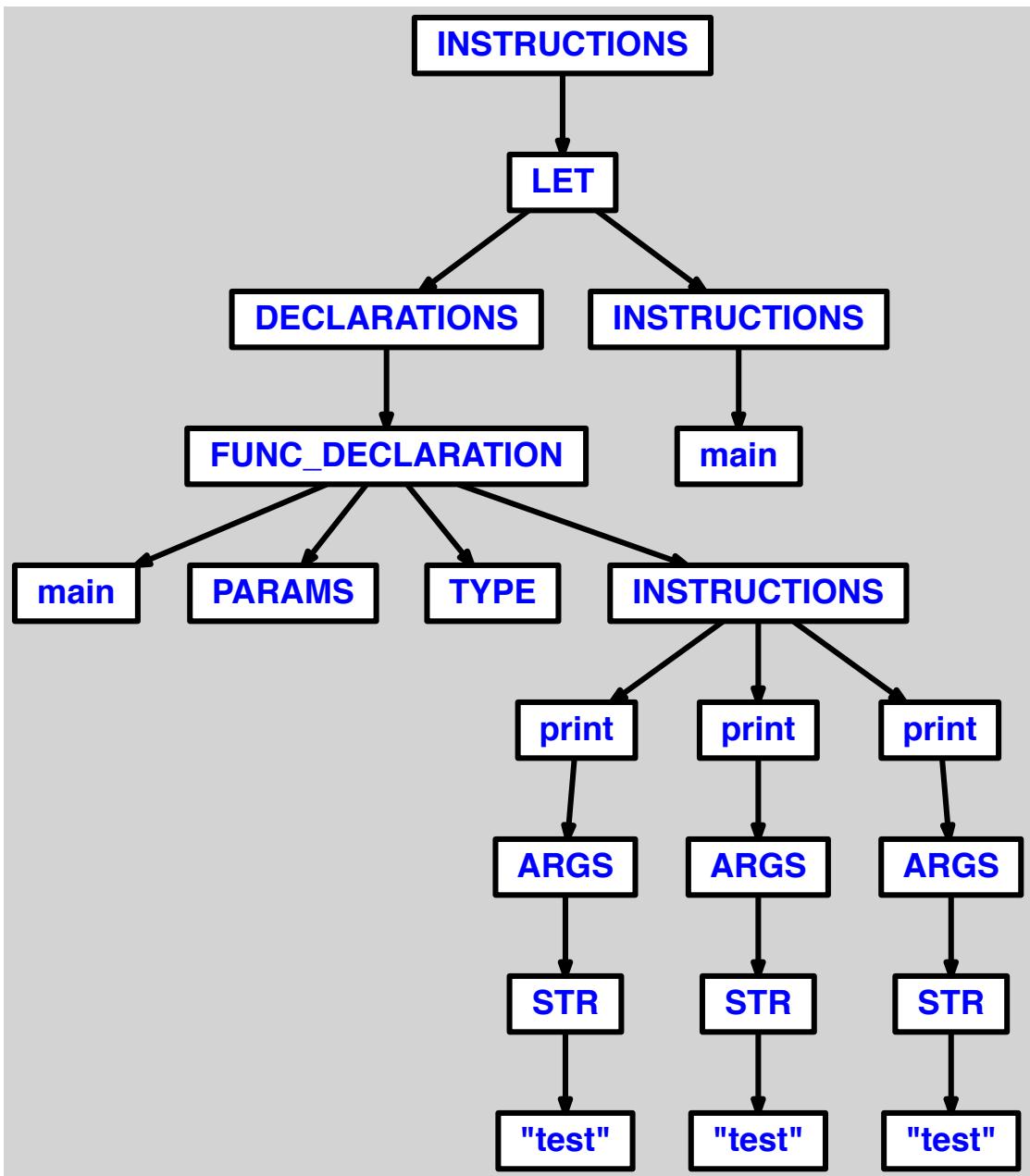
#### 4.2.11 3 commentaires, apres instructions, sur 1 ligne

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



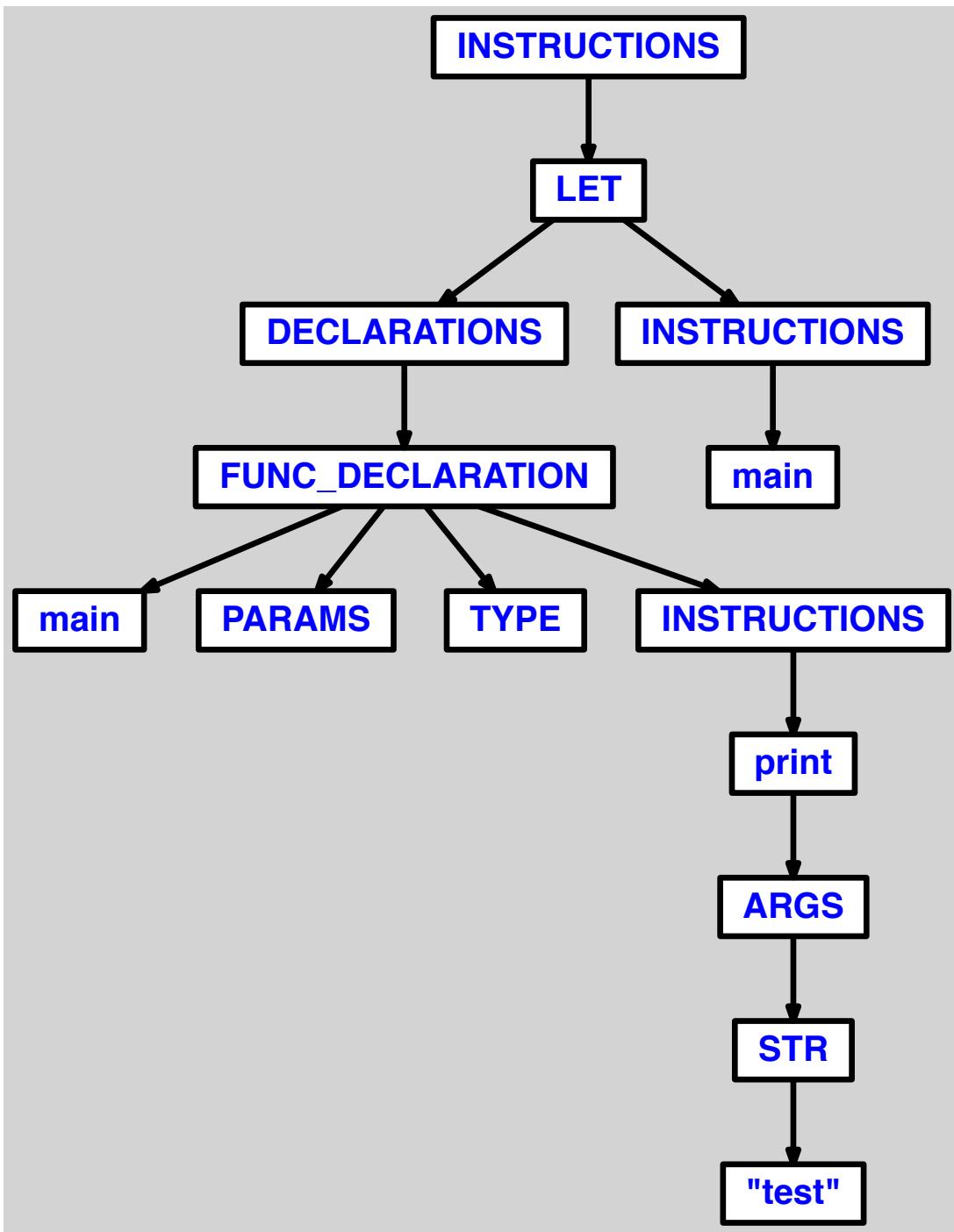
#### 4.2.12 3 commentaires, apres instructions, sur plusieurs lignes

```
1 let
2   function main() =
3     (print("test");
4     /*
5      1
6      2
7      3
8     */
9     print("test");
10    /*
11     4
12     5
13     6
14    */
15    print("test")
16    /*
17     7
18     8
19     9
20    */
21  )
22 in main() end
```



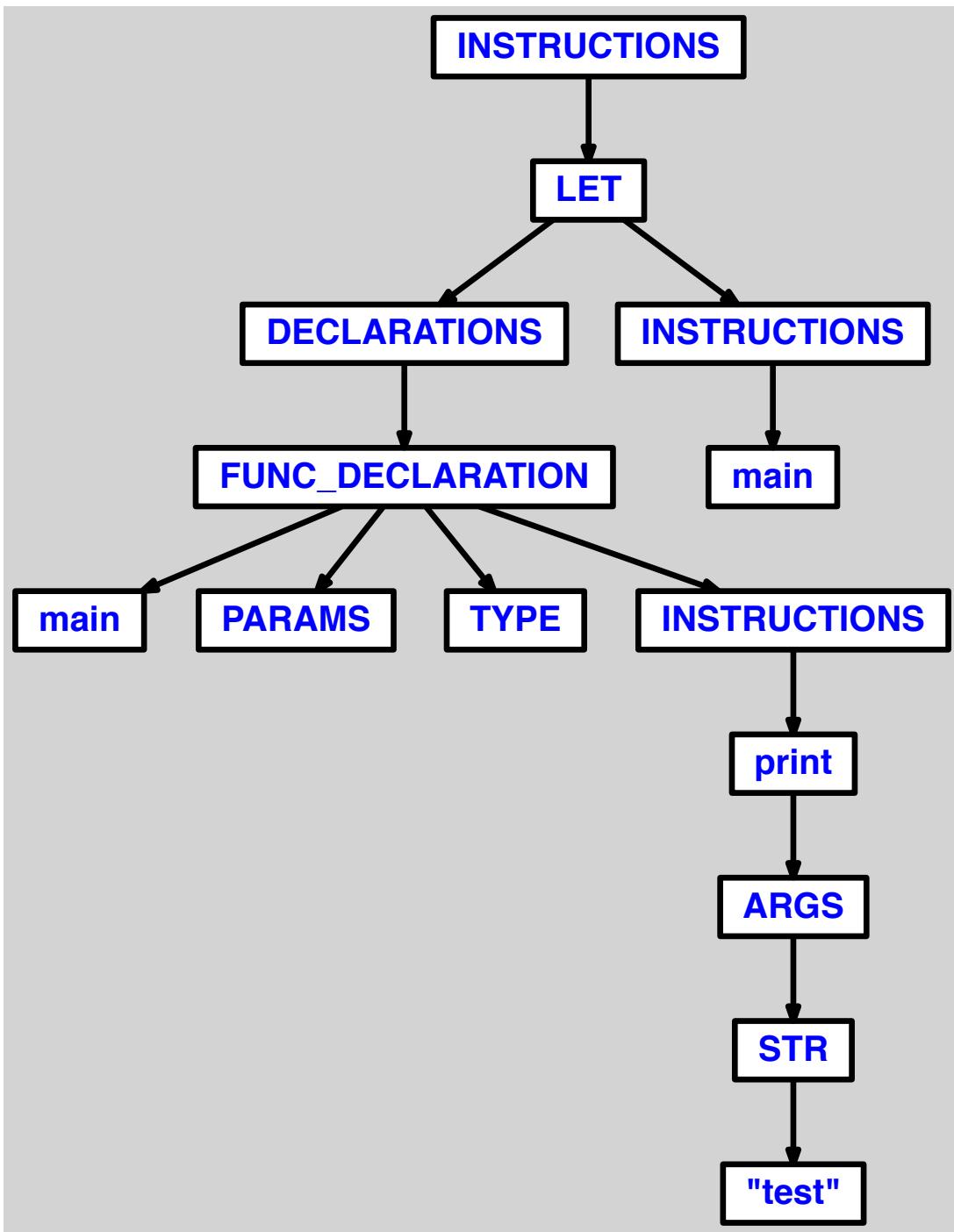
#### 4.2.13 commentaire sans contenu sur 1 ligne

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



#### 4.2.14 commentaire sans contenu sur plusieurs lignes

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

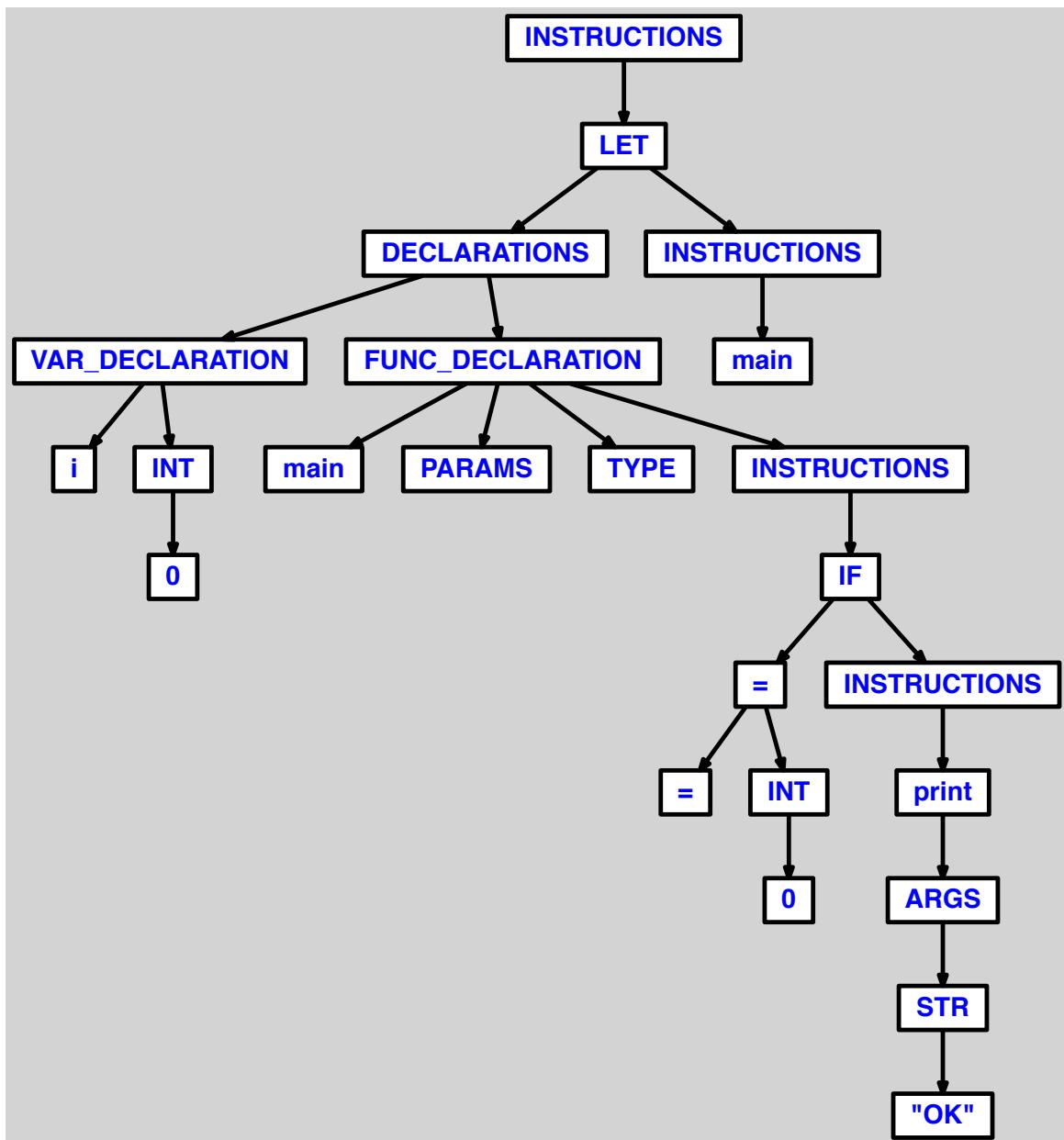


## 5 comparaison

### 5.1 KO

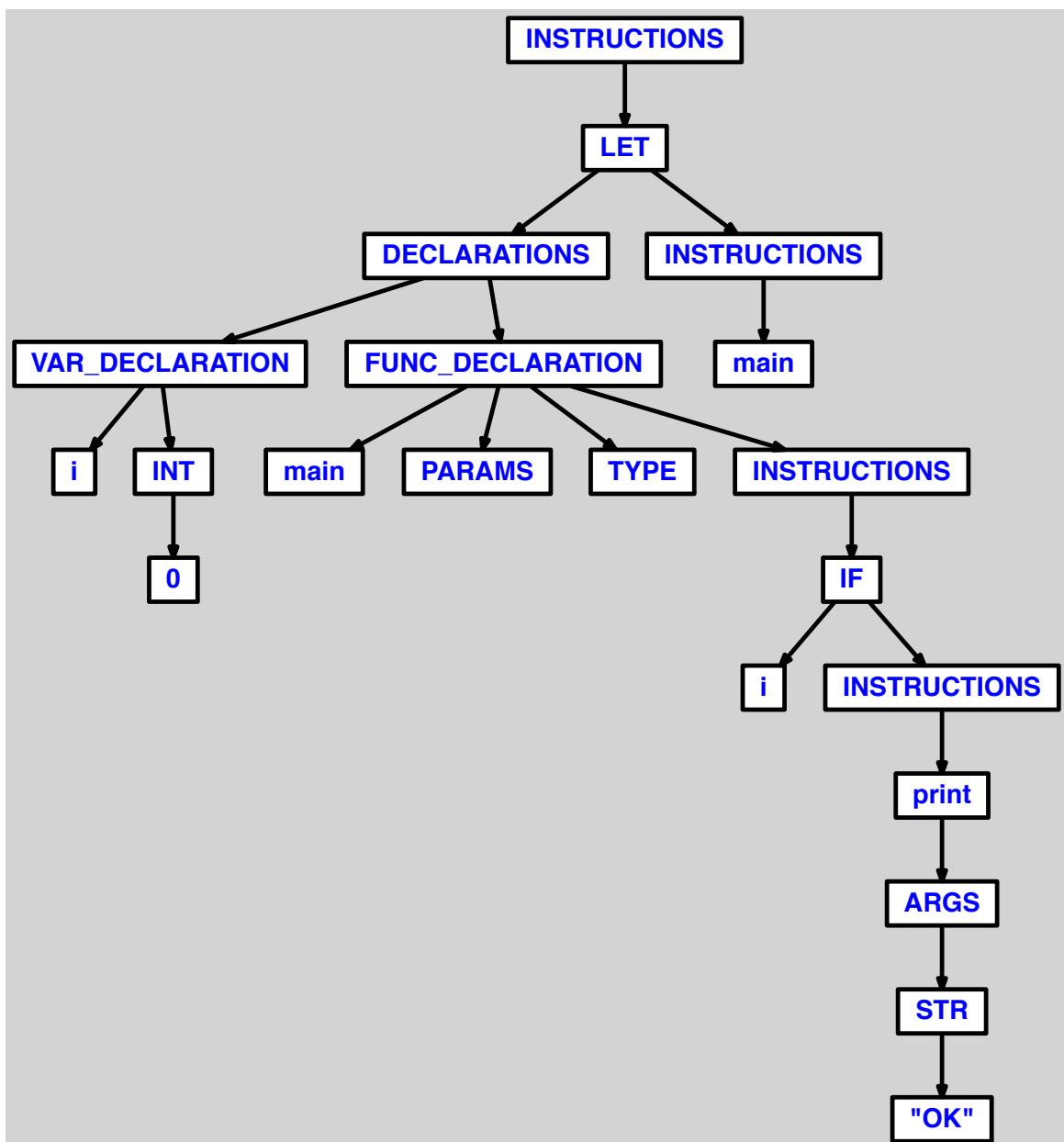
#### 5.1.1 comparaison avec oubli d'opérande gauche

```
1 let
2   var i := 0
3
4   function main() =
5     if = 0 then print("OK")
6   in main() end
```



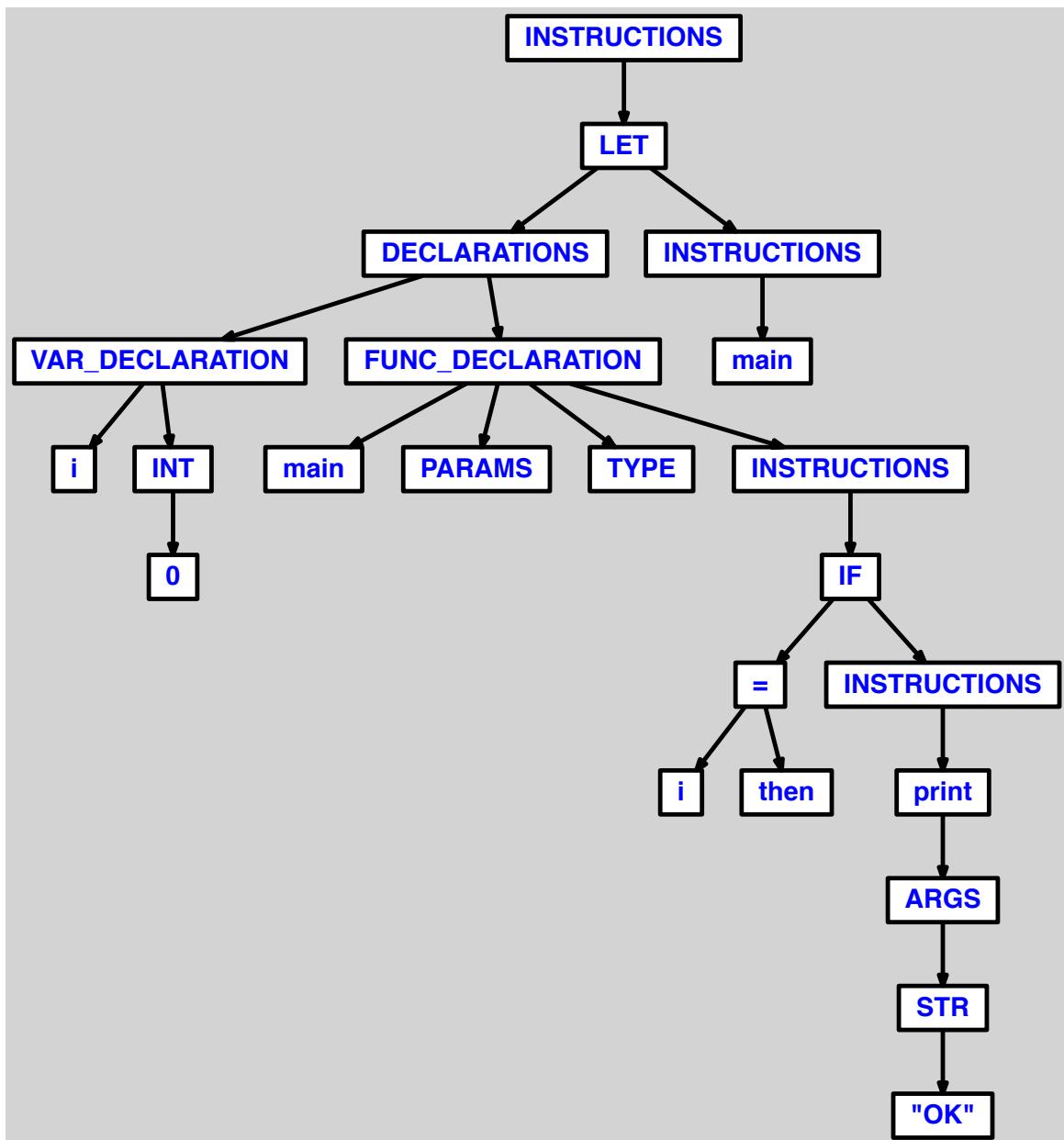
### 5.1.2 comparaison avec oubli d'opérateur

```
1 let
2   var i := 0
3
4   function main() =
5     if i 0 then print("OK")
6 in main() end
```



### 5.1.3 comparaison avec oubli d'opérateur droit

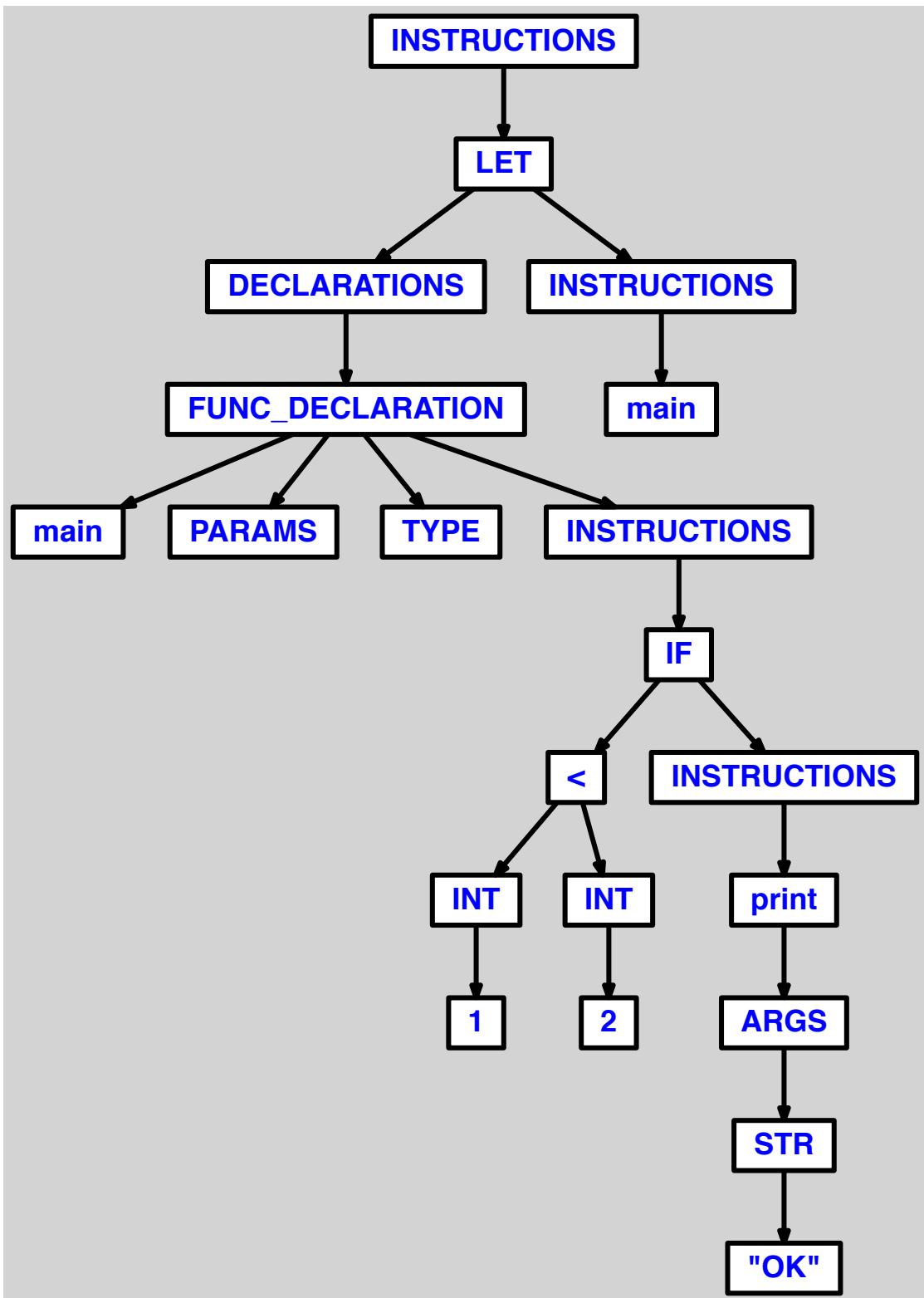
```
1 let
2   var i := 0
3
4   function main() =
5     if i = then print("OK")
6 in main() end
```



## 5.2 OK

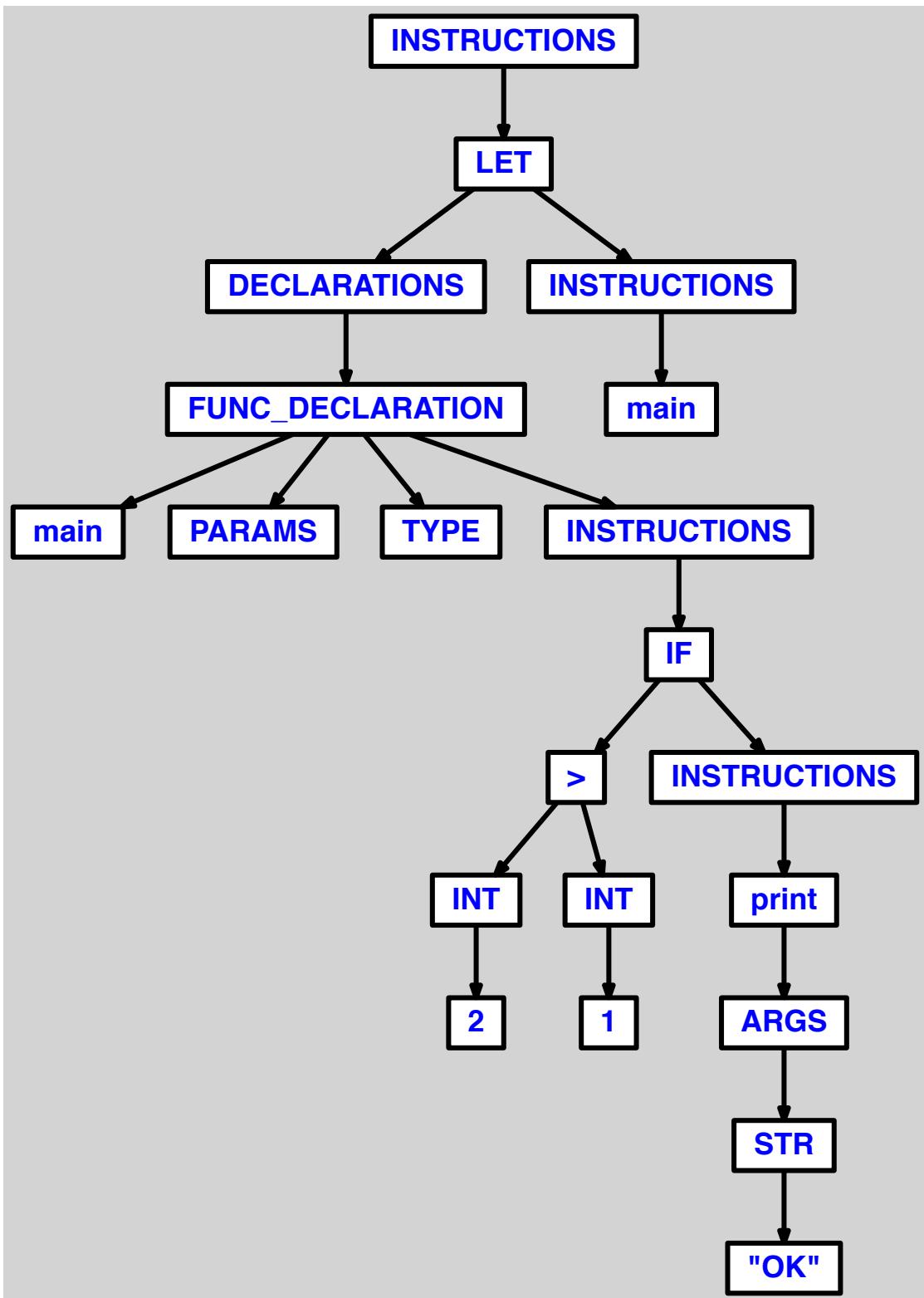
### 5.2.1 comparaison simple d'entiers avec <"<">

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



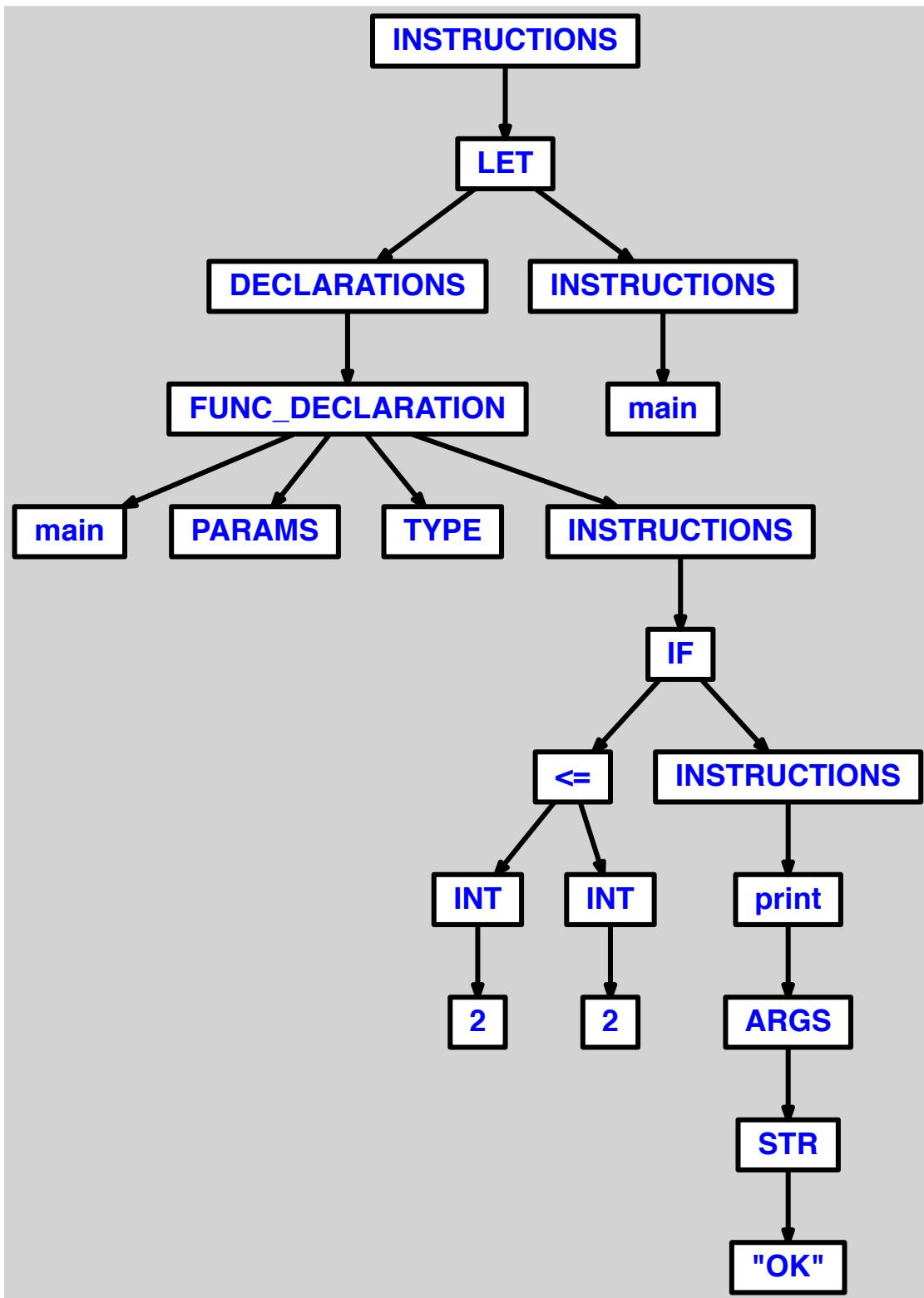
### 5.2.2 comparaison simple d'entiers avec <">>

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



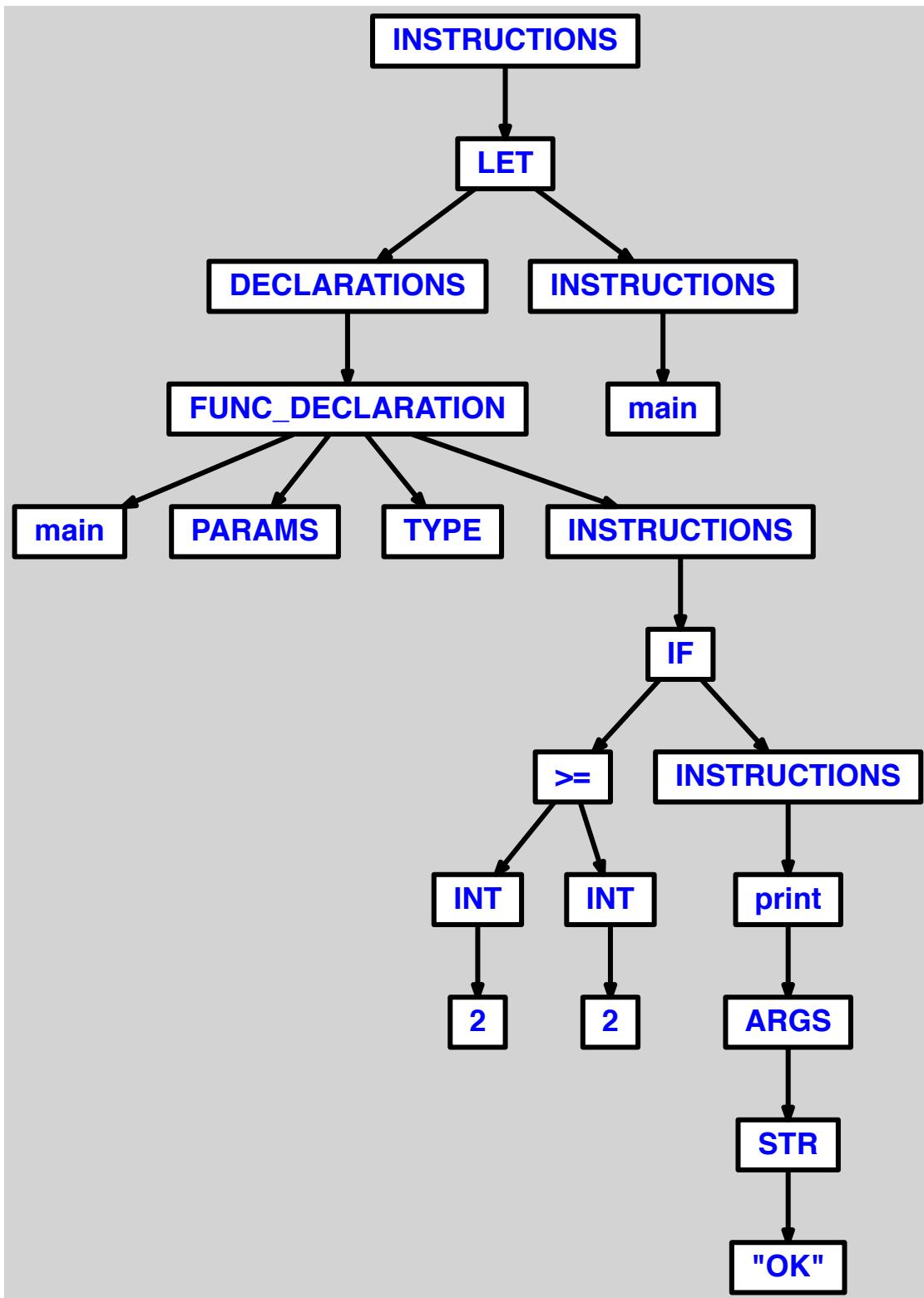
### 5.2.3 comparaison simple d'entiers avec <"<=>>

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



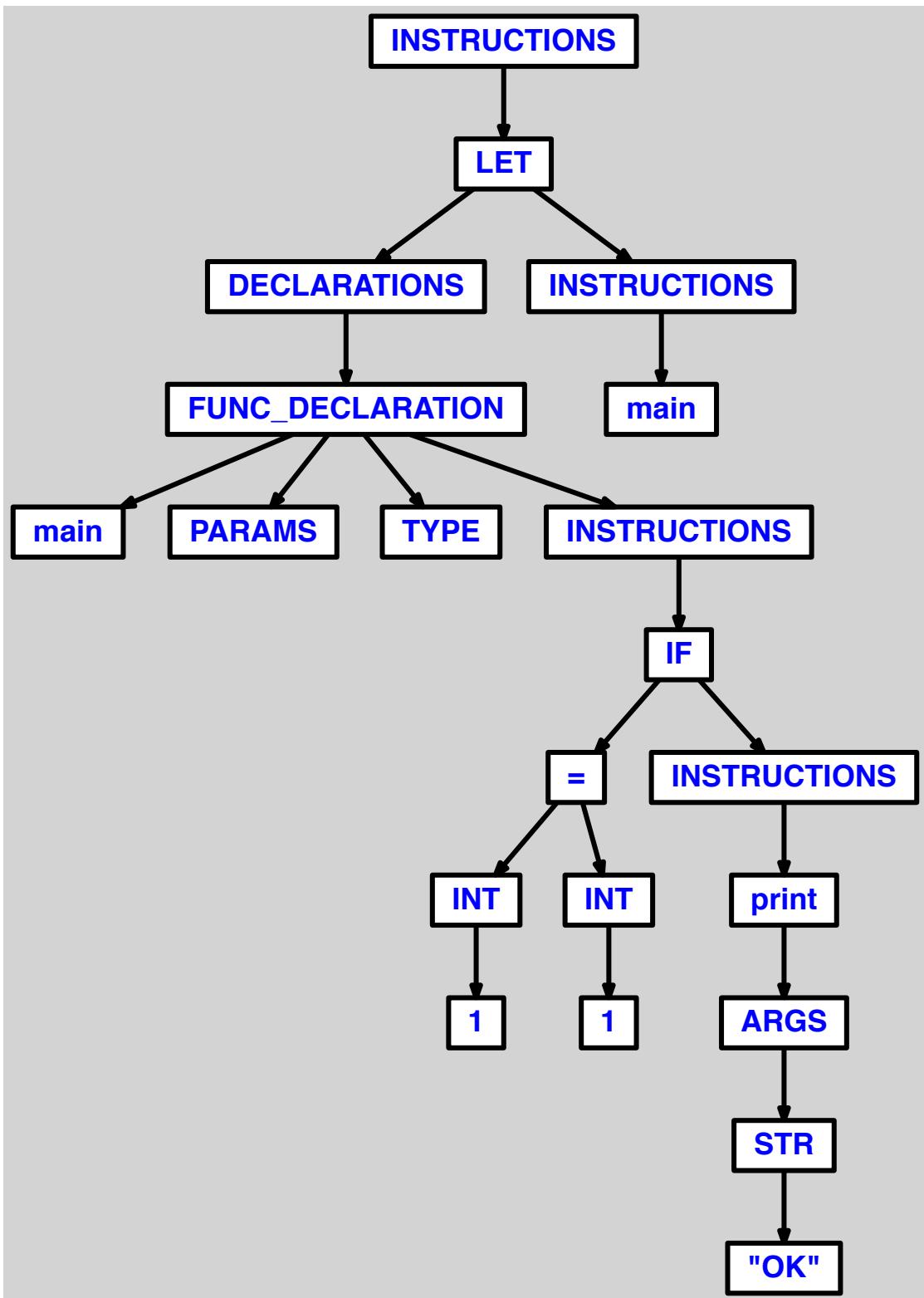
#### 5.2.4 comparaison simple d'entiers avec <">=>

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



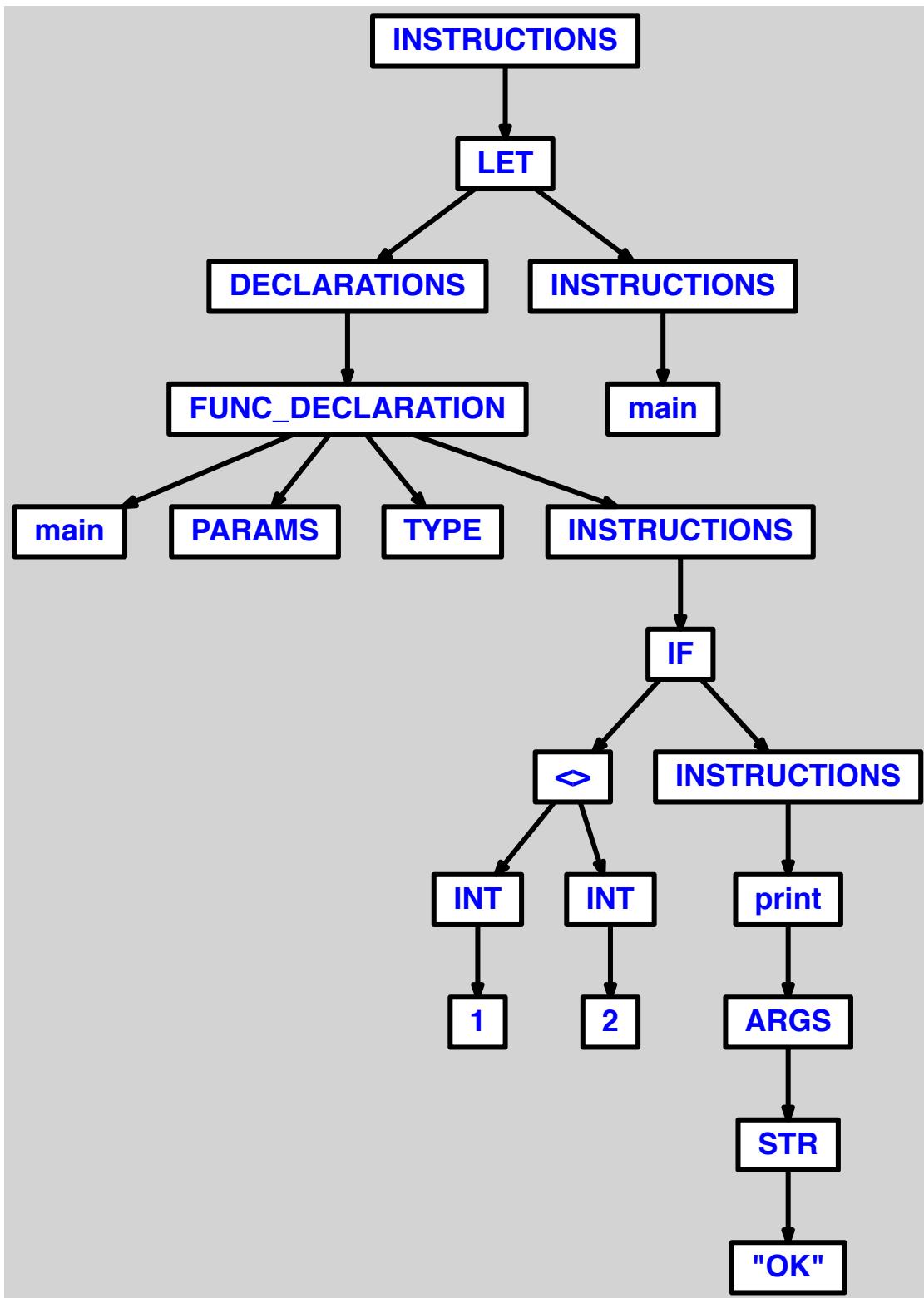
### 5.2.5 comparaison simple d'entiers avec <=>

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



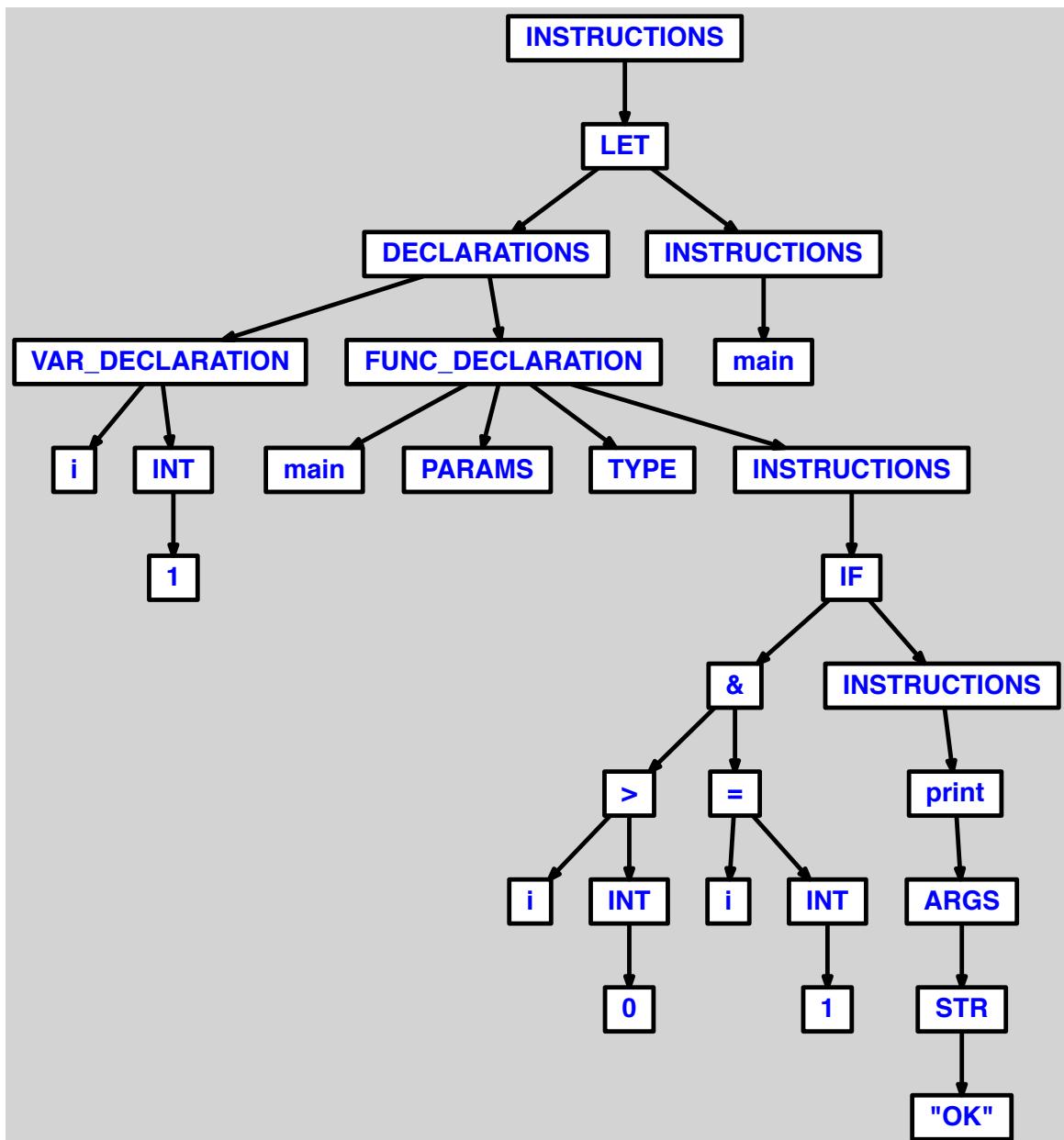
### 5.2.6 comparaison simple d'entiers avec <"<>">

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



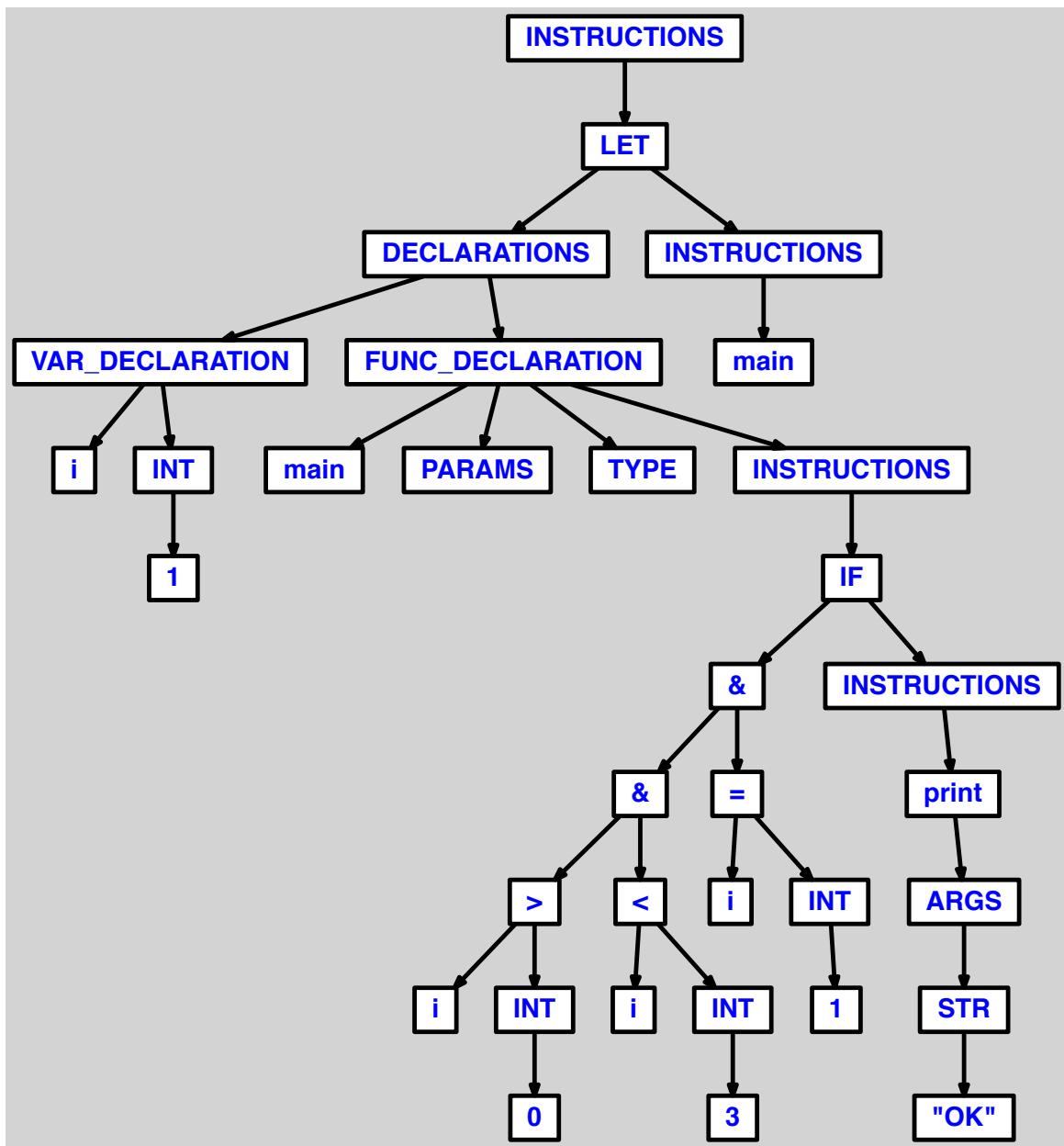
### 5.2.7 comparaison double d'entiers

```
1 let
2   var i := 1
3
4   function main() =
5     if i > 0 & i = 1 then print("OK")
6 in main() end
```



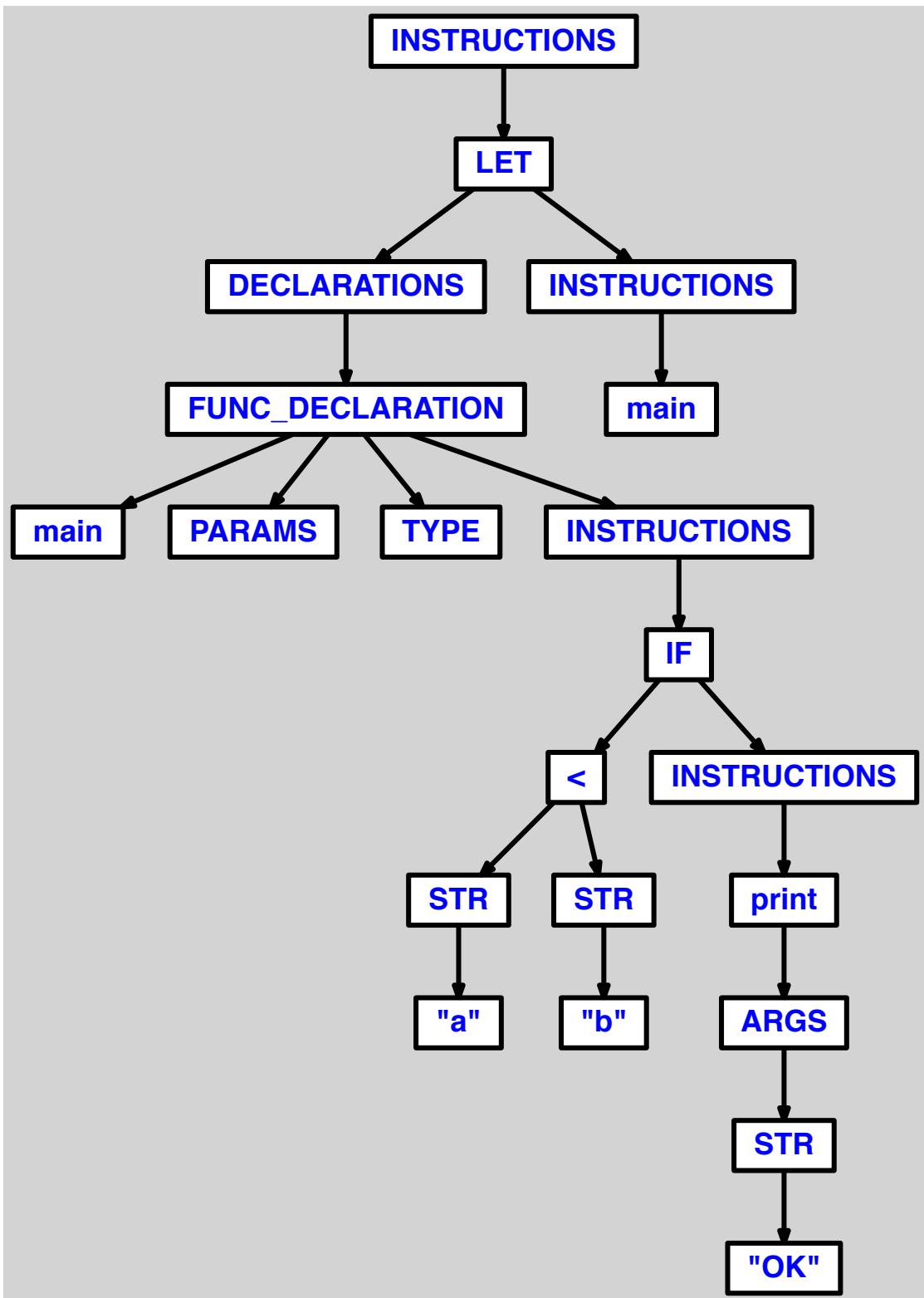
### 5.2.8 comparaison triple d'entiers

```
1 let
2   var i := 1
3
4   function main() =
5     if i > 0 & i < 3 & i = 1 then print("OK")
6 in main() end
```



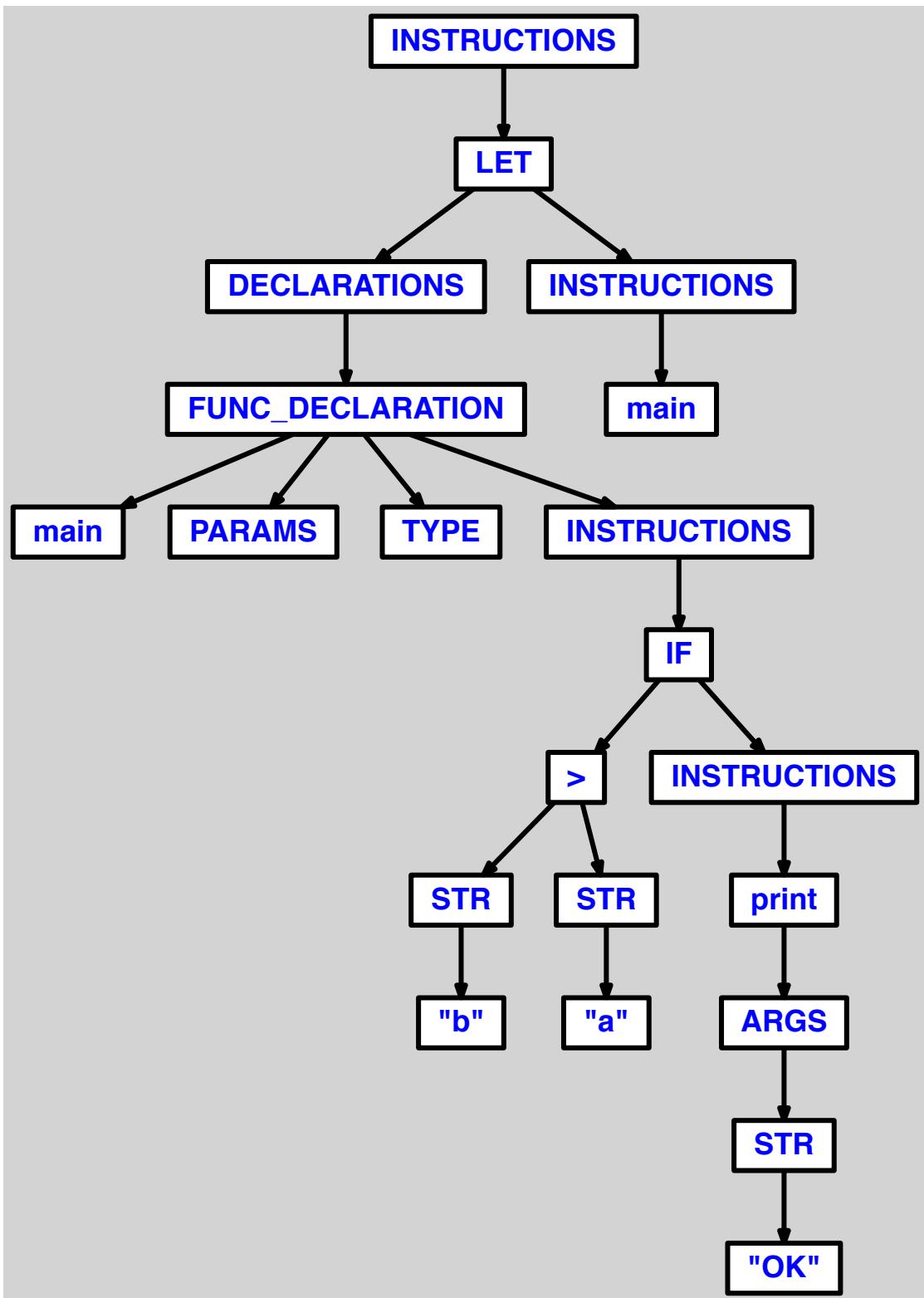
### 5.2.9 comparaison simple de chaines avec <"<">

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



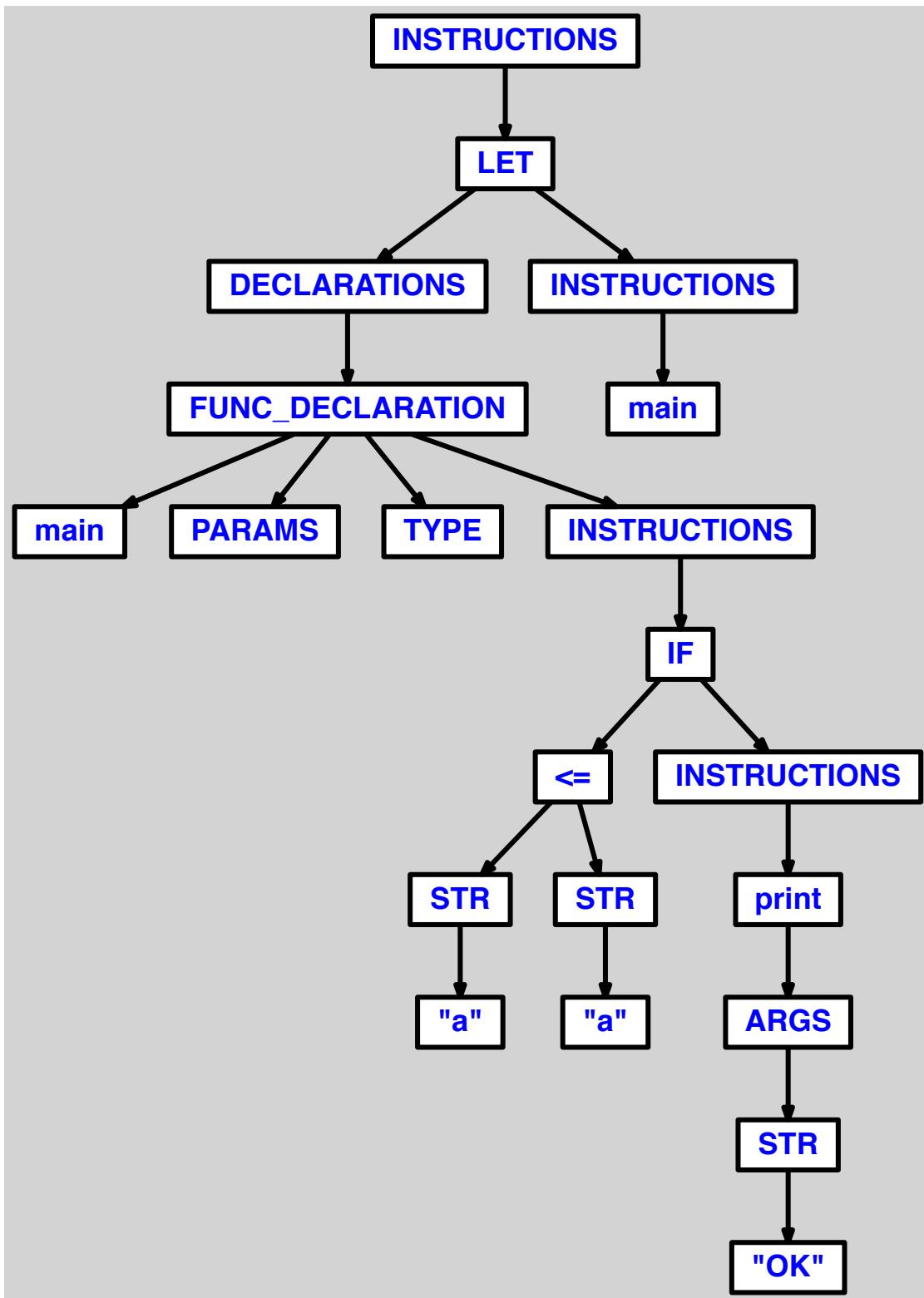
### 5.2.10 comparaison simple de chaines avec <">>

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



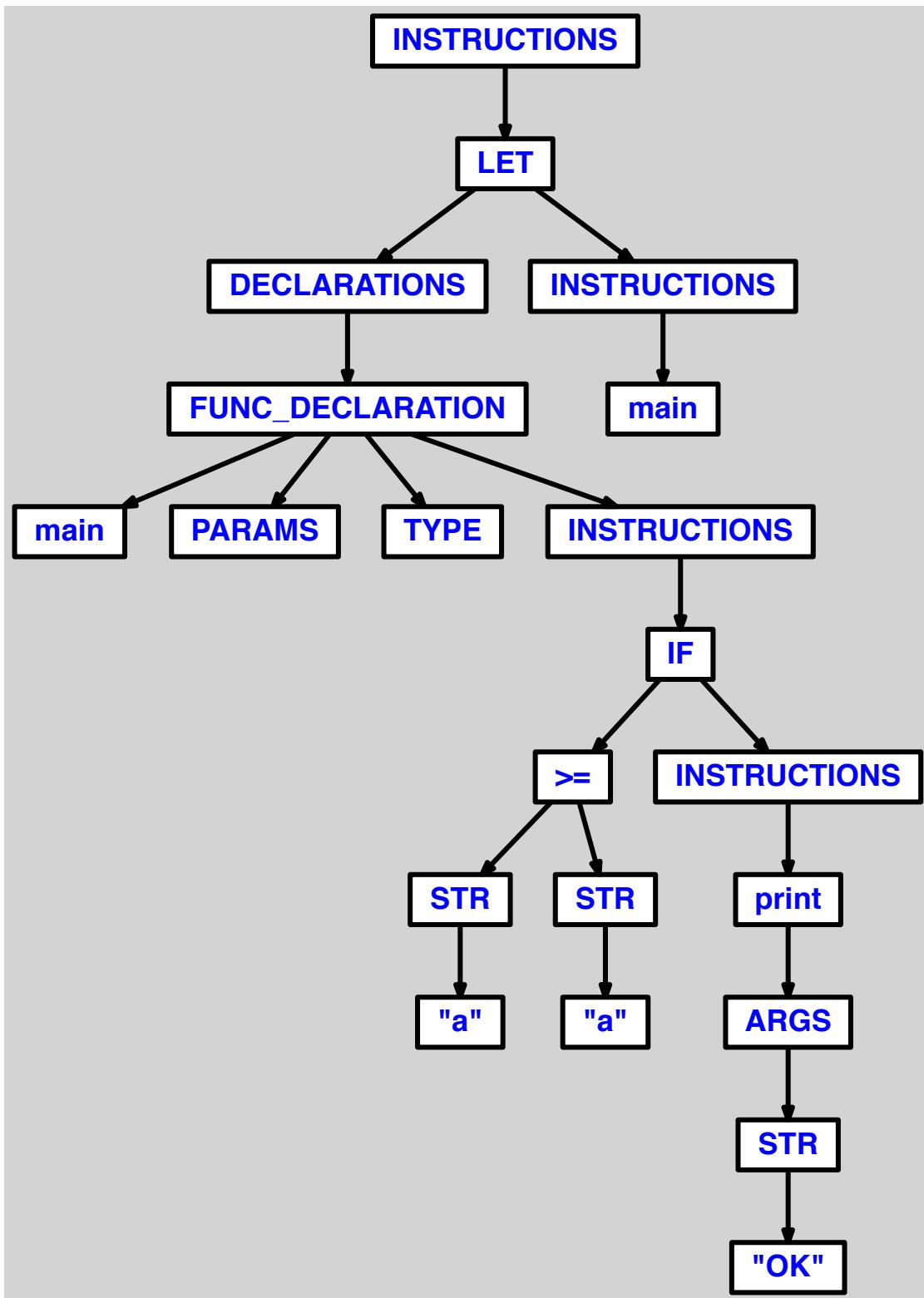
### 5.2.11 comparaison simple de chaines avec <"<=">

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



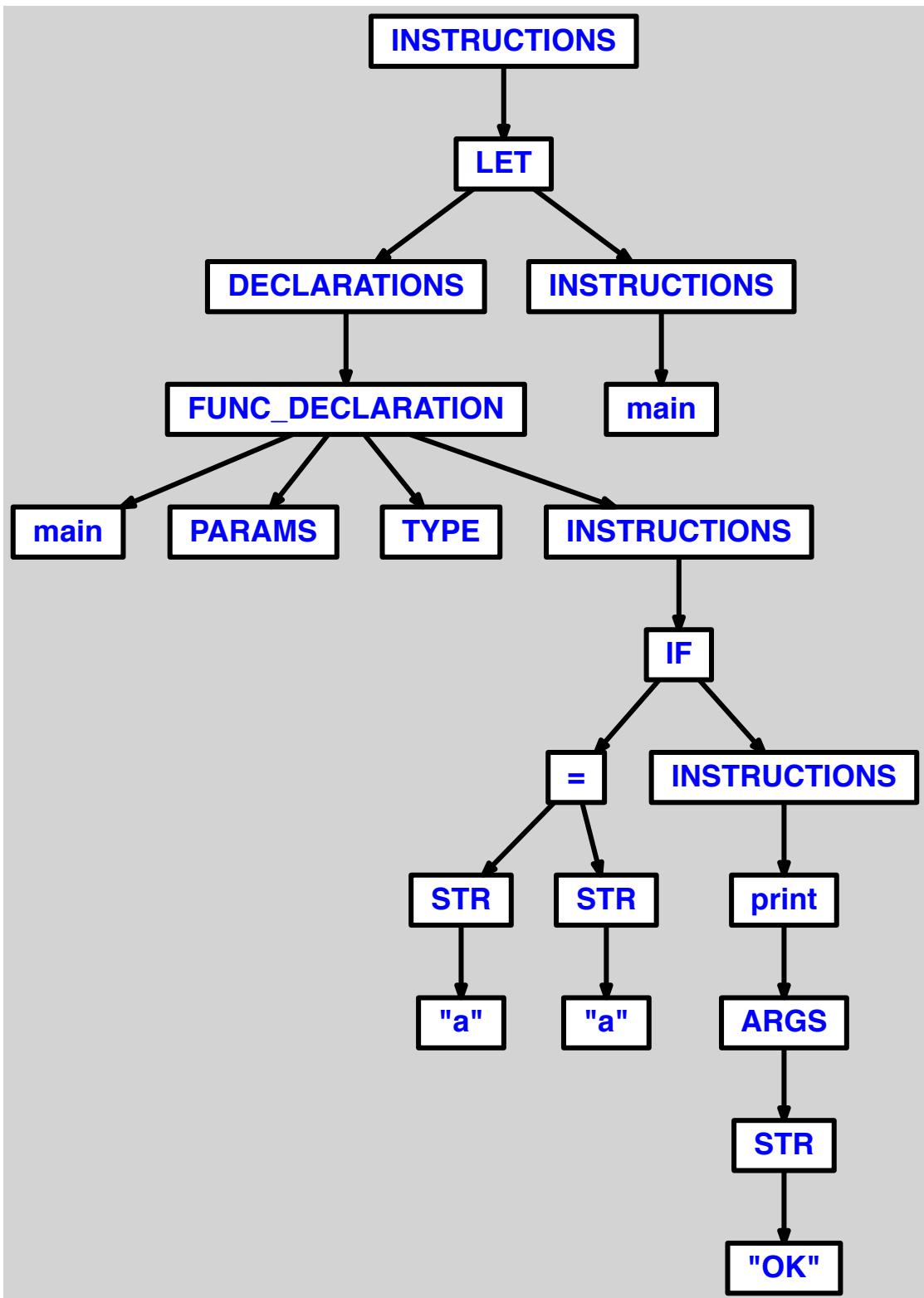
### 5.2.12 comparaison simple de chaines avec <">=>

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



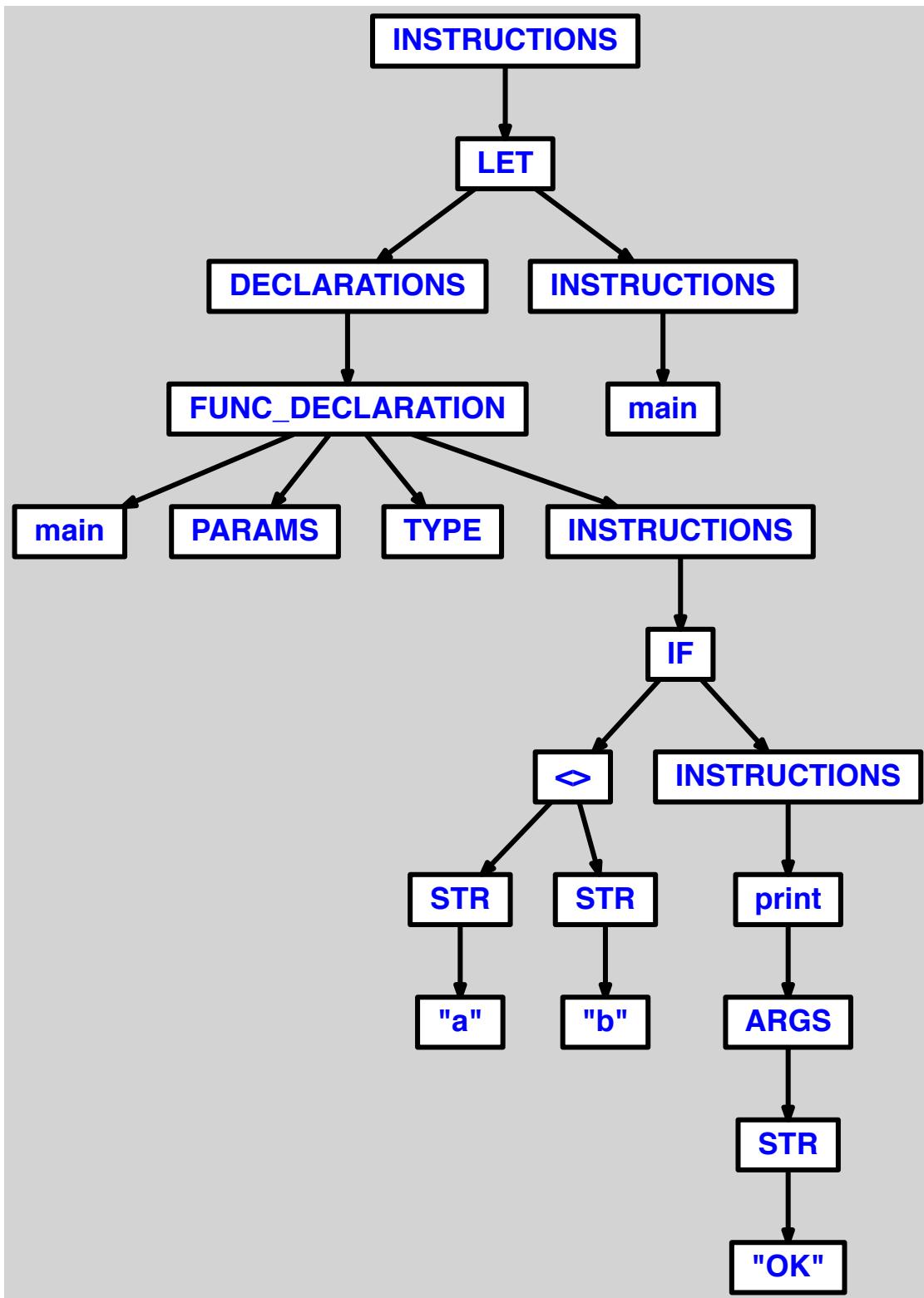
### 5.2.13 comparaison simple de chaines avec <=>

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



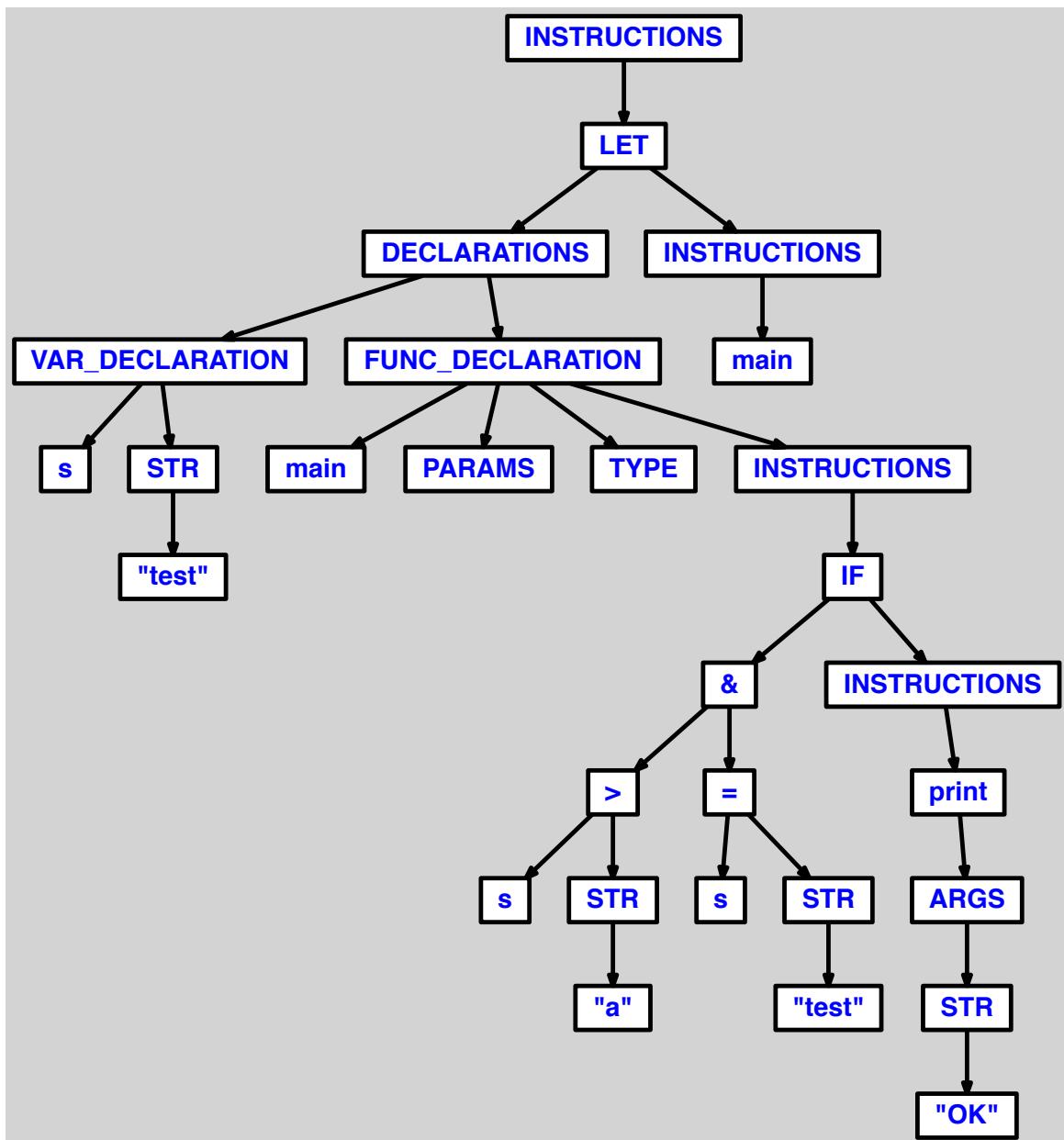
#### 5.2.14 comparaison simple de chaines avec <">>

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



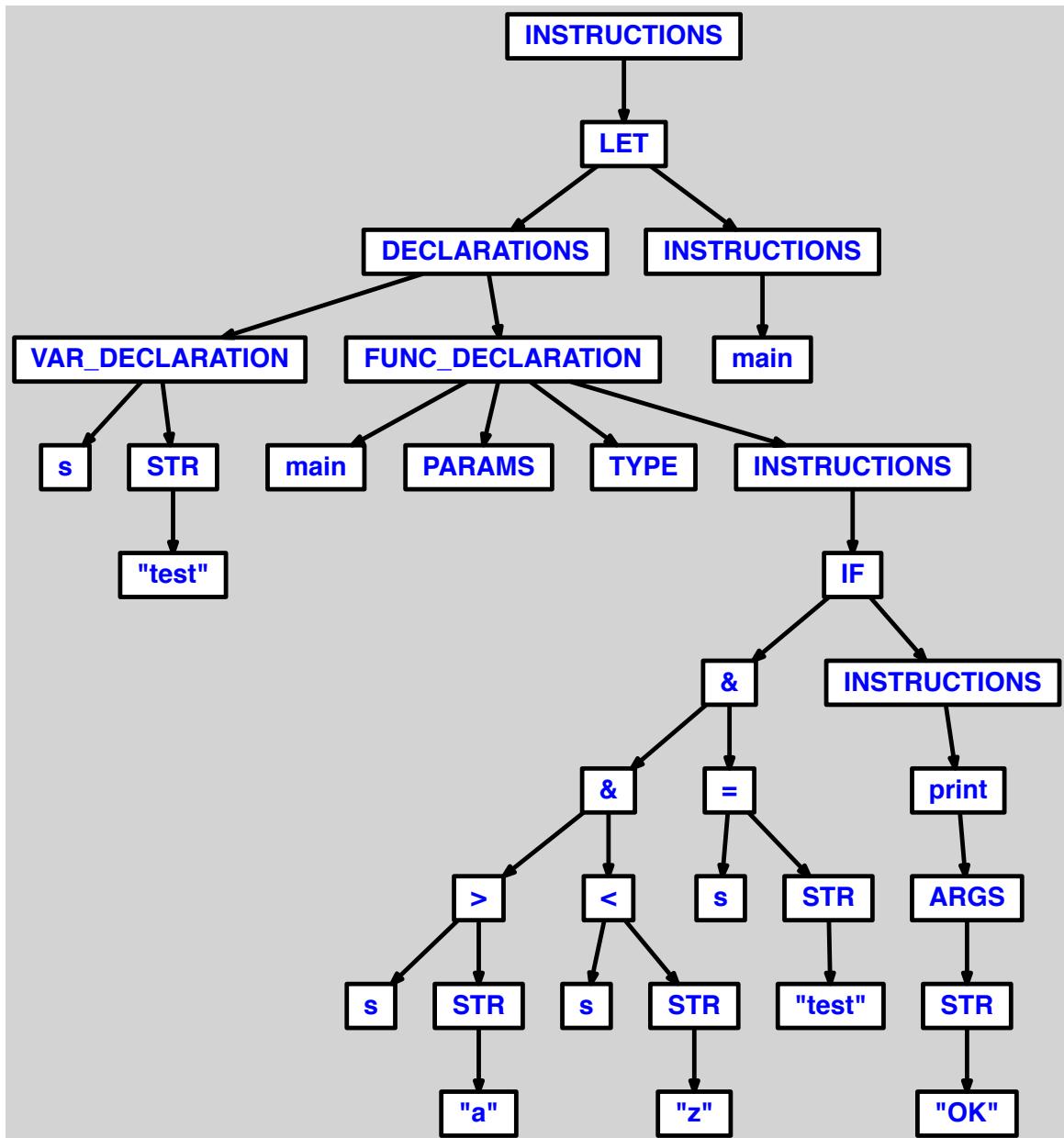
### 5.2.15 comparaison double de chaines

```
1 let
2   var s := "test"
3
4   function main() =
5     if s > "a" & s = "test" then print("OK")
6 in main() end
```



### 5.2.16 comparaison triple de chaines

```
1 let
2   var s := "test"
3
4   function main() =
5     if s > "a" & s < "z" & s = "test" then print("OK")
6 in main() end
```



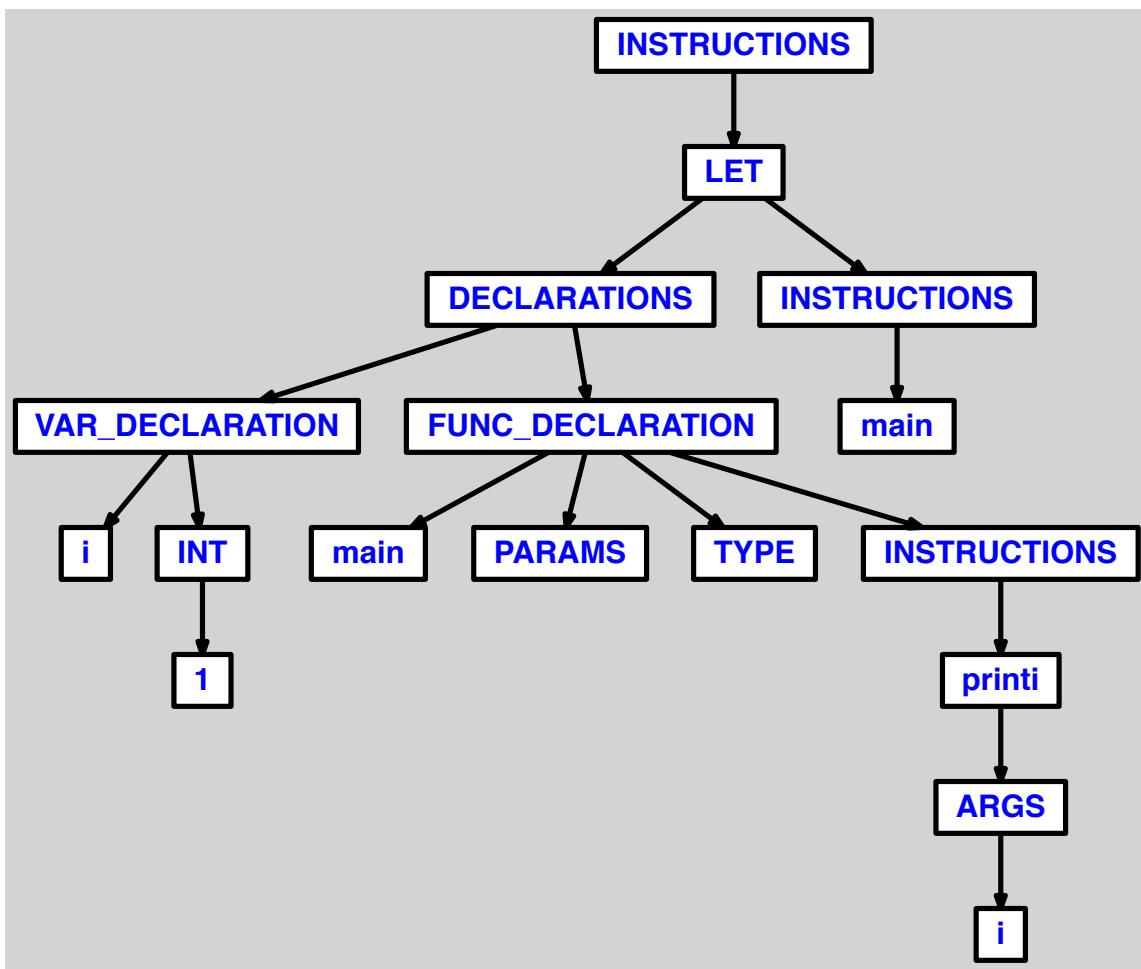
## 6 espace

### 6.1 KO

### 6.2 OK

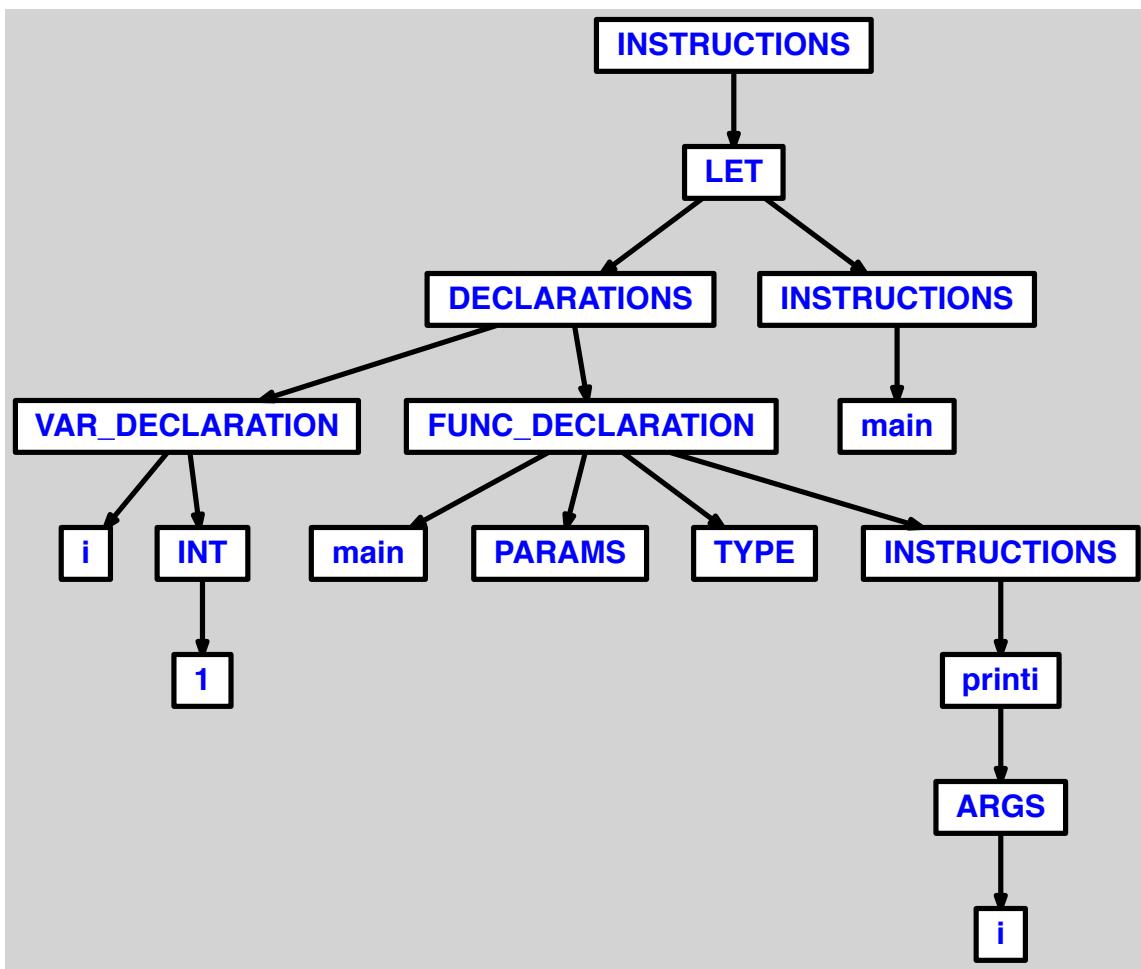
#### 6.2.1 1 espace entre <var> et <i>

```
1 let
2   var i := 1
3
4   function main() = printi(i)
5 in main() end
```



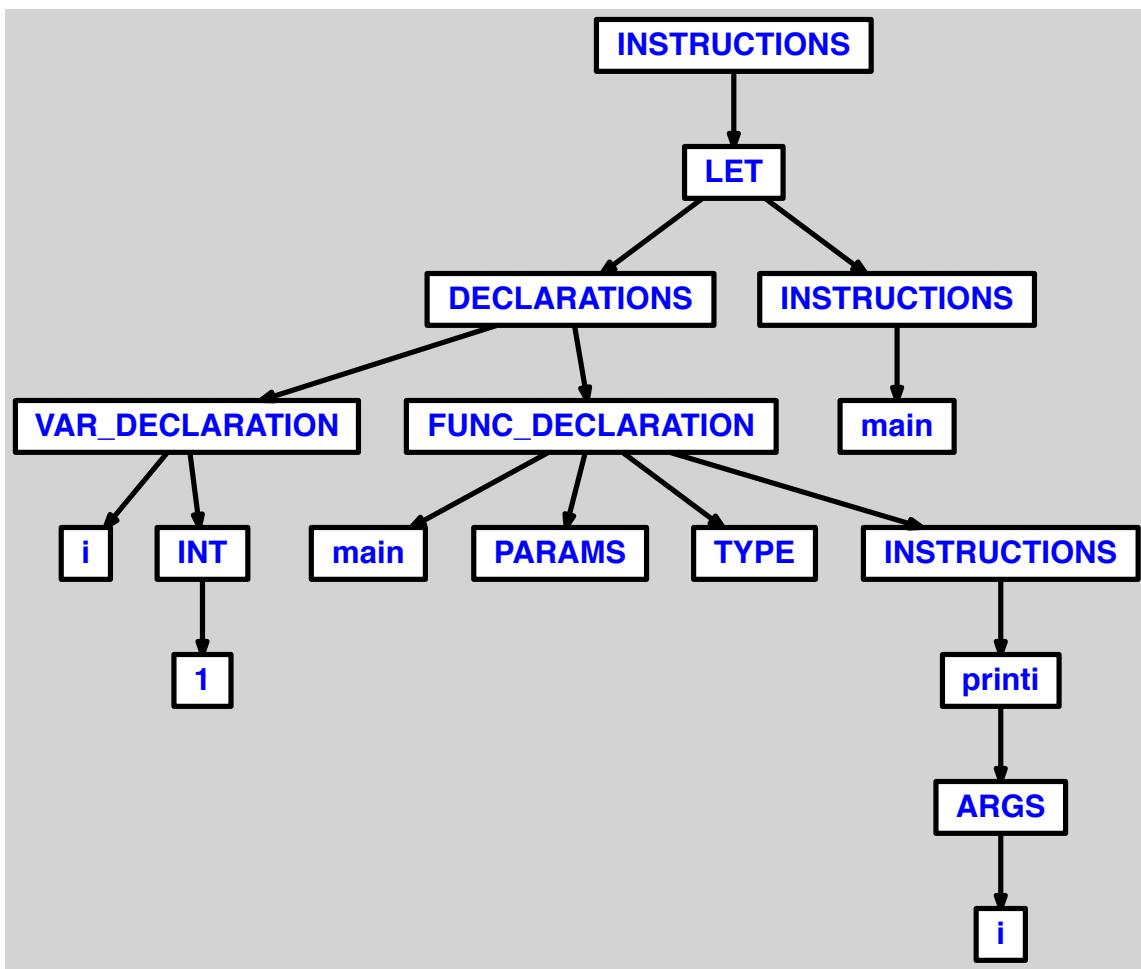
### 6.2.2 1 tabulation entre <var> et <i>

```
1 let
2   var i := 1
3
4   function main() = printi(i)
5 in main() end
```



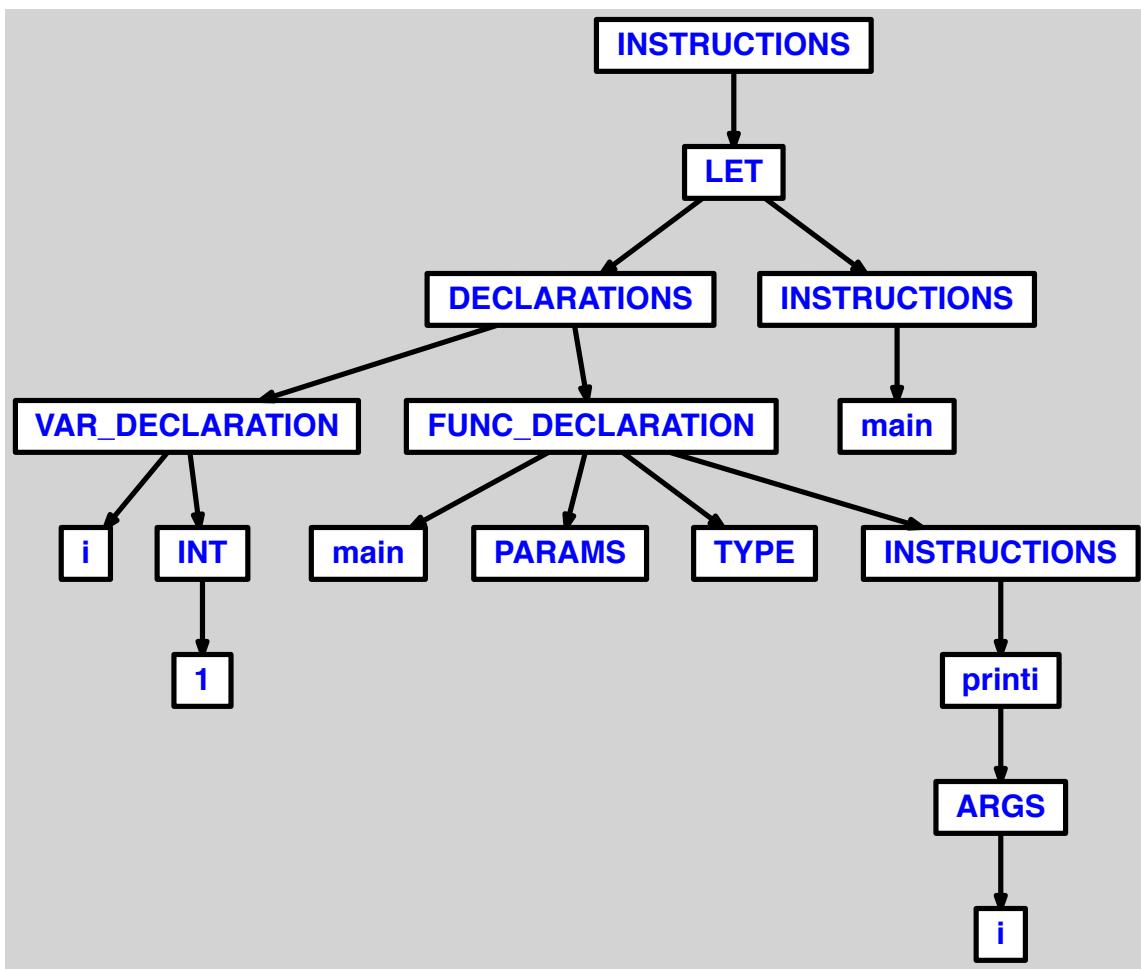
### 6.2.3 1 saut de ligne entre <var> et <i>

```
1 let
2   var
3   i := 1
4
5   function main() = printi(i)
6 in main() end
```



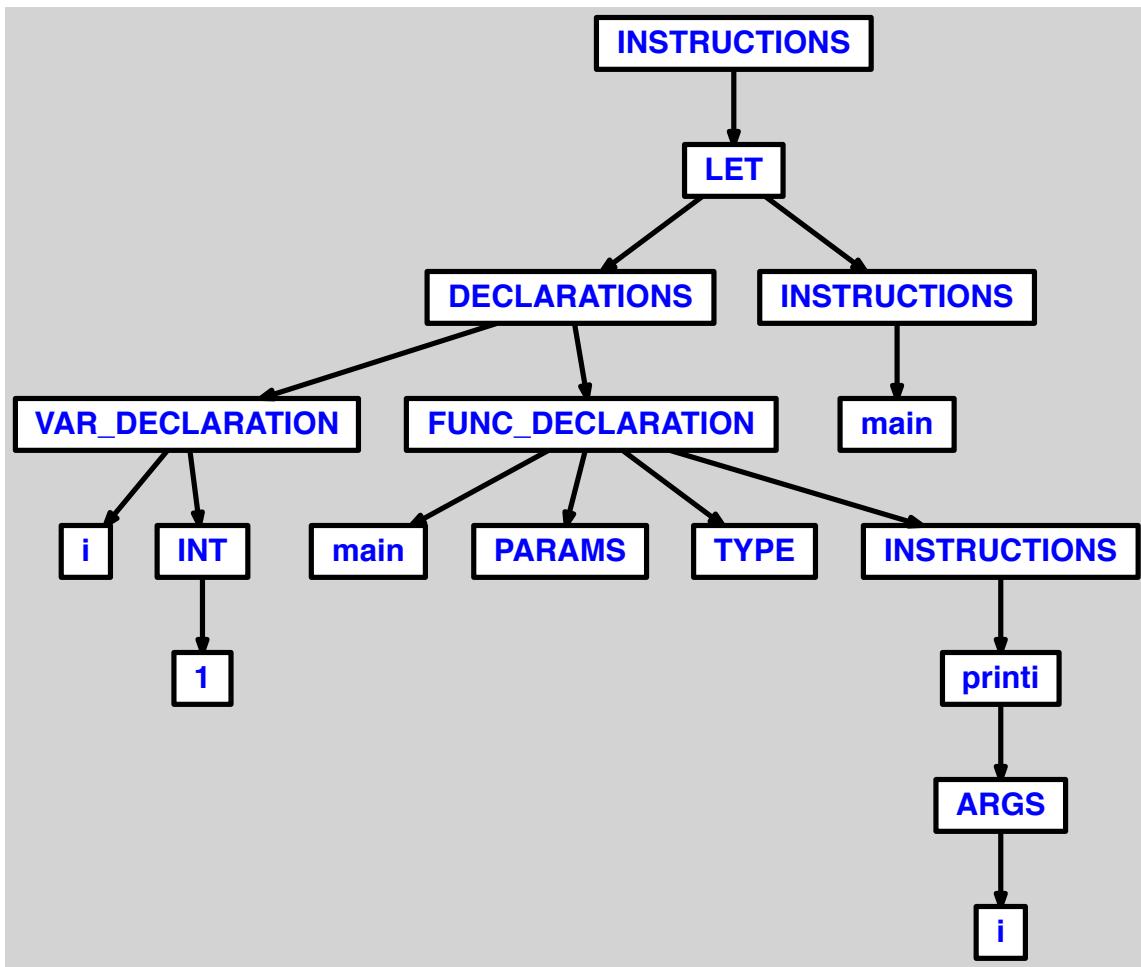
#### 6.2.4 2 espaces entre <var> et <i>

```
1 let
2     var   i  := 1
3
4     function main() = printi(i)
5 in main() end
```



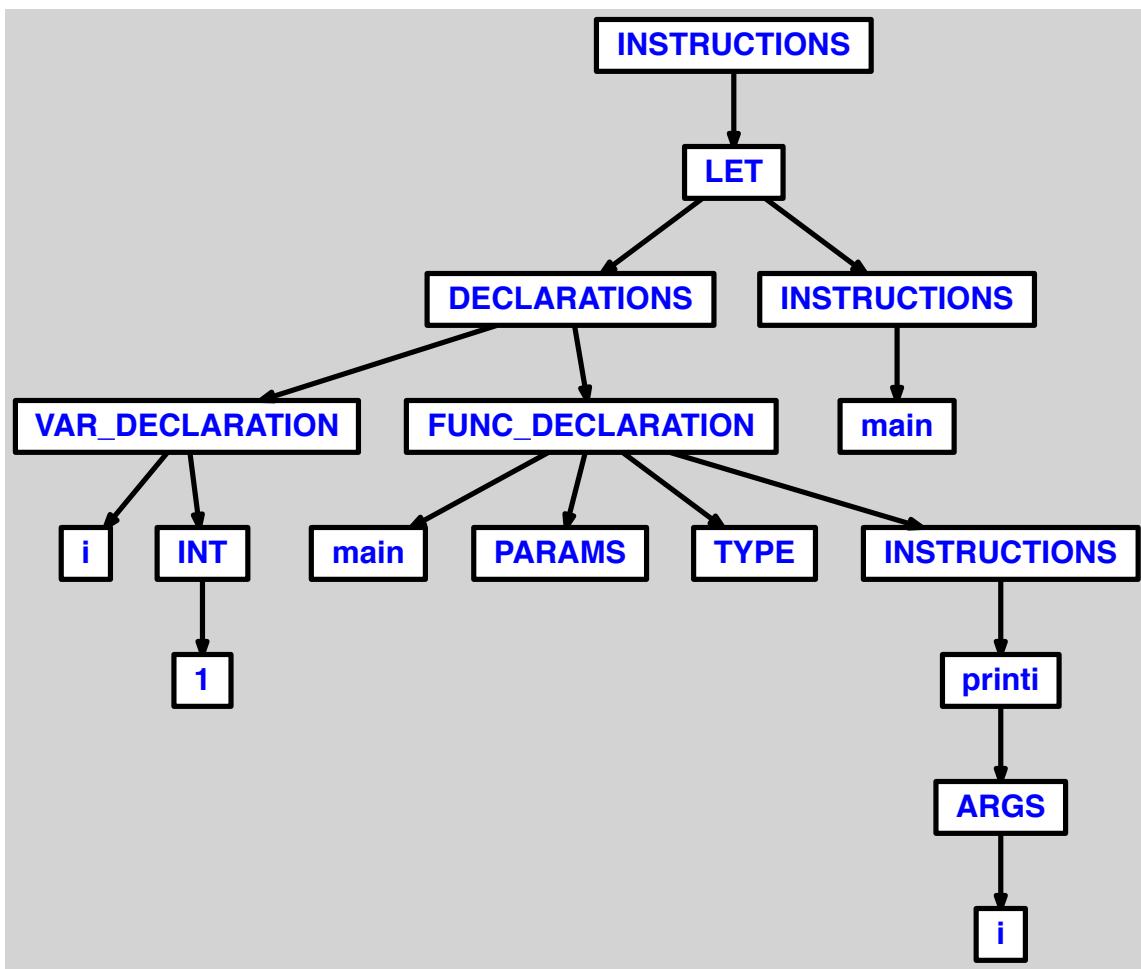
#### 6.2.5 2 tabulations entre <var> et <i>

```
1 let
2   var    i := 1
3
4   function main() = printi(i)
5 in main() end
```



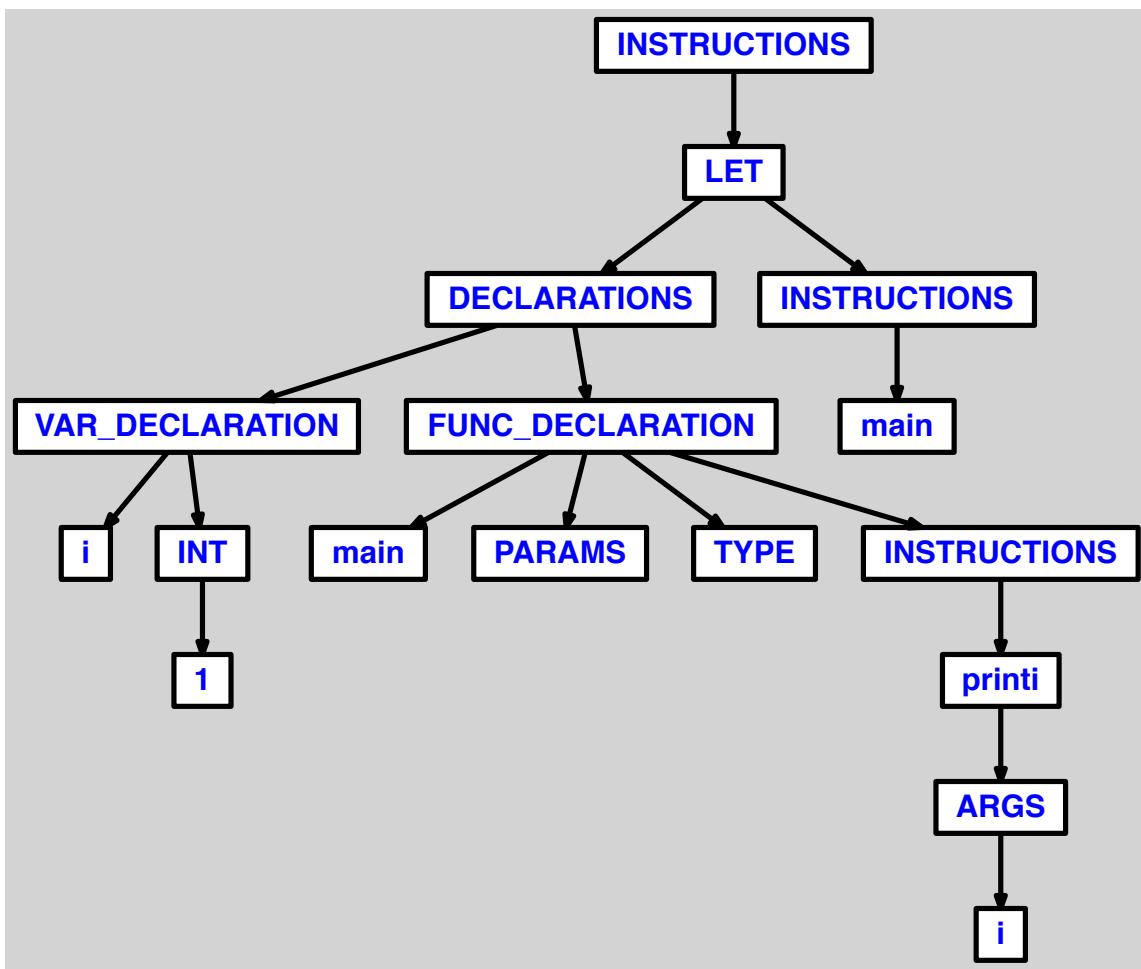
#### 6.2.6 2 sauts de ligne entre <var> et <i>

```
1 let
2   var
3
4   i := 1
5
6   function main() = printi(i)
7 in main() end
```



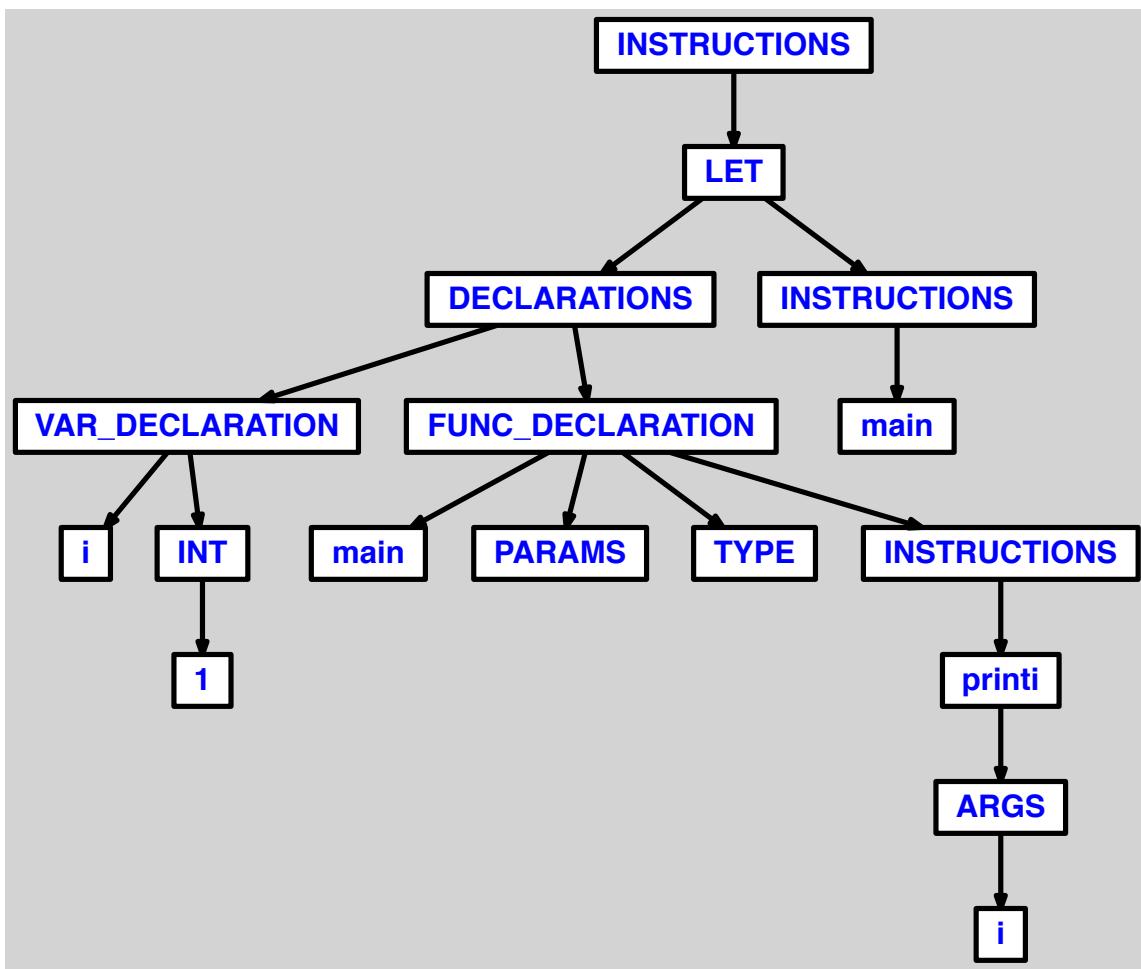
### 6.2.7 3 espaces entre <var> et <i>

```
1 let
2     var    i := 1
3
4     function main() = printi(i)
5 in main() end
```



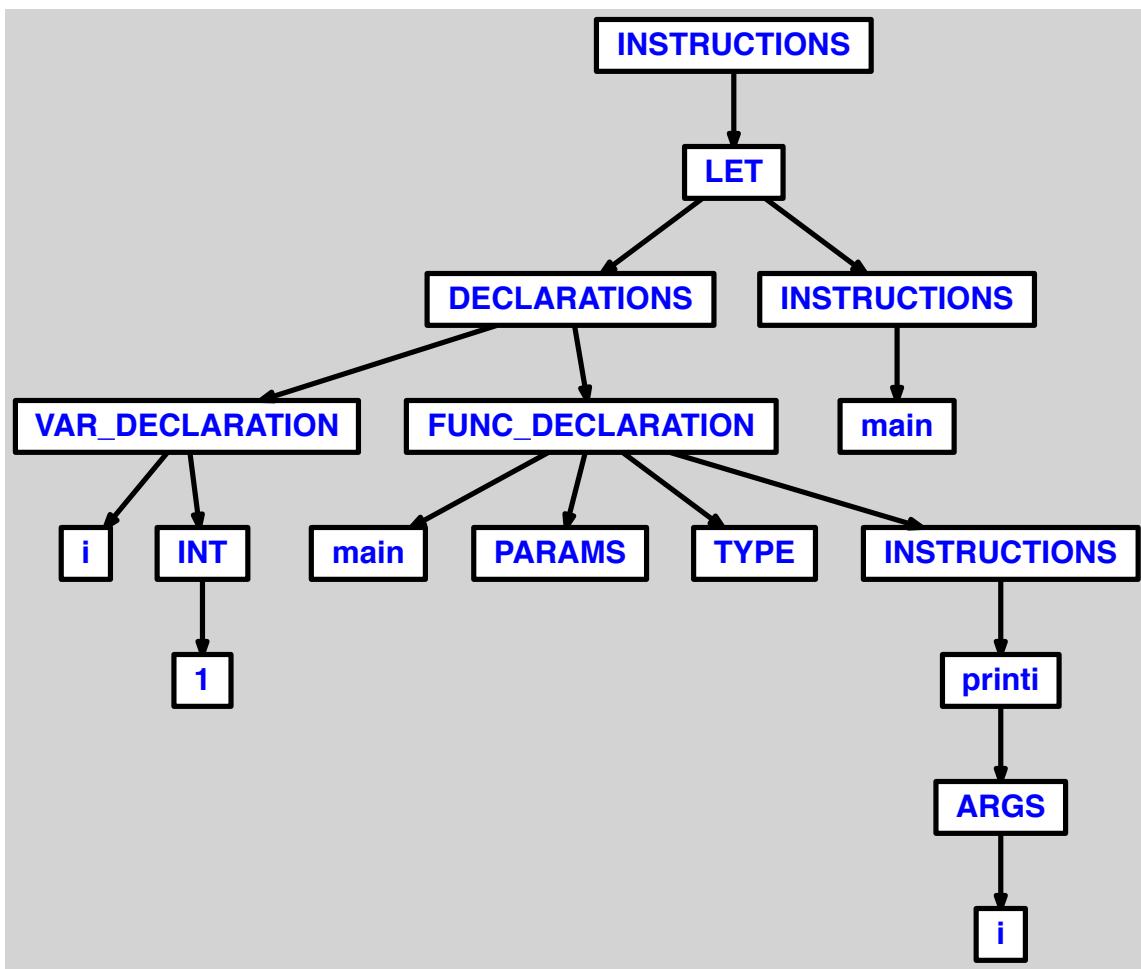
#### 6.2.8 3 tabulations entre <var> et <i>

```
1 let
2     var      i := 1
3
4     function main() = printi(i)
5 in main() end
```



### 6.2.9 3 sauts de ligne entre <var> et <i>

```
1 let
2     var
3
4
5     i := 1
6
7     function main() = printi(i)
8 in main() end
```

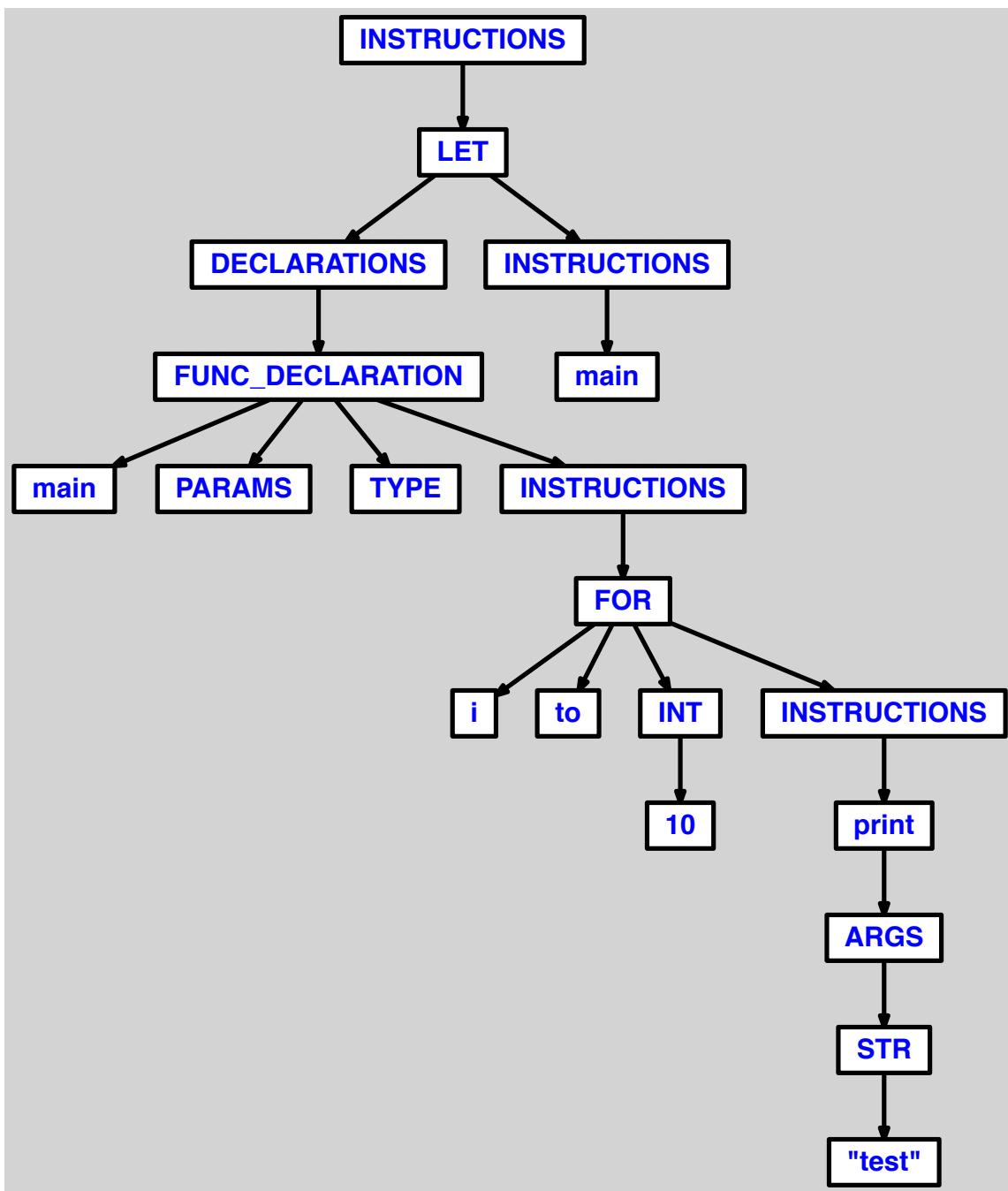


## 7 for

### 7.1 KO

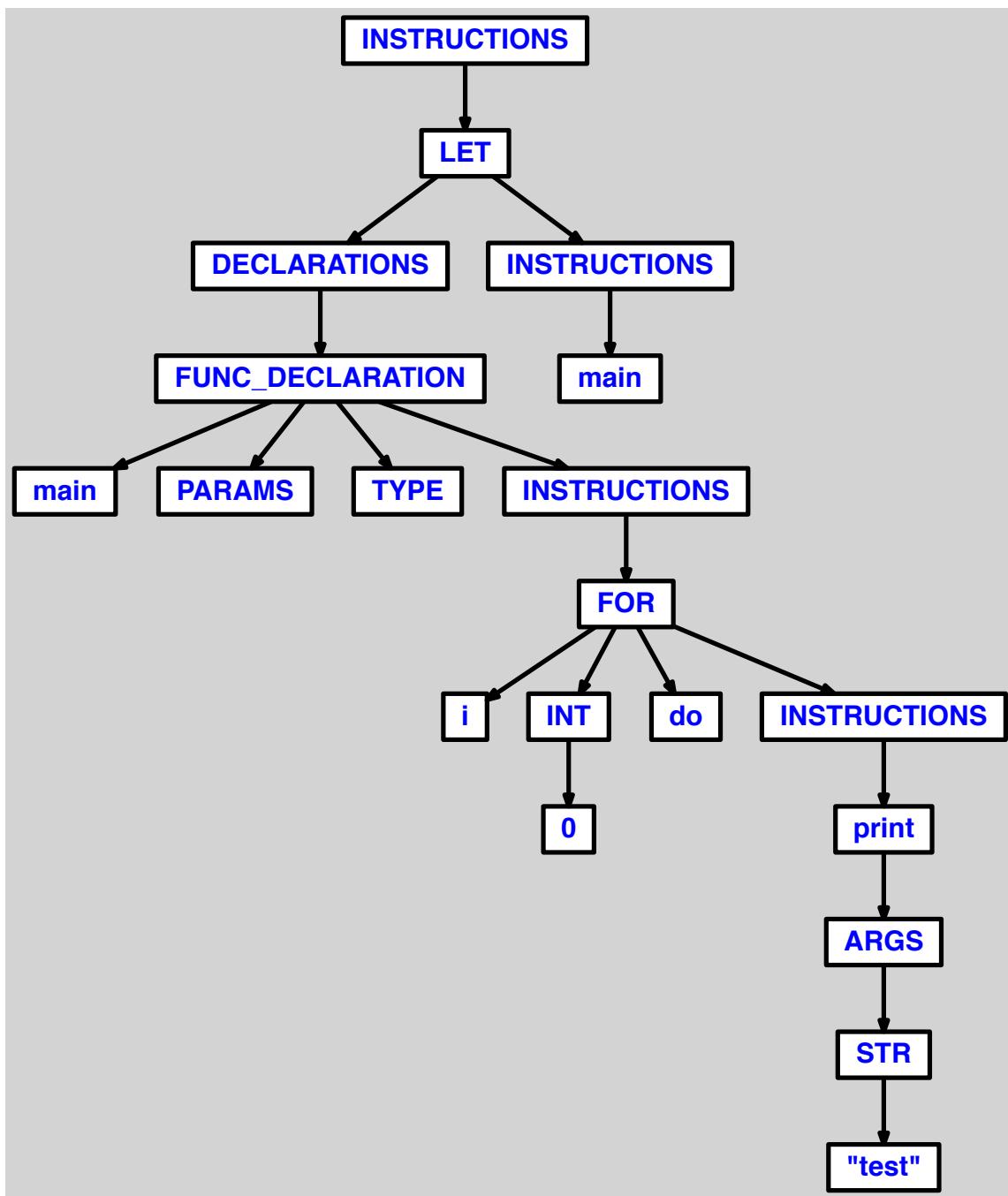
#### 7.1.1 <for> avec affectation de chaine pour le compteur

```
1 let
2   function main() =
3     for i := "3" to 10 do
4       print("test")
5   in main() end
```



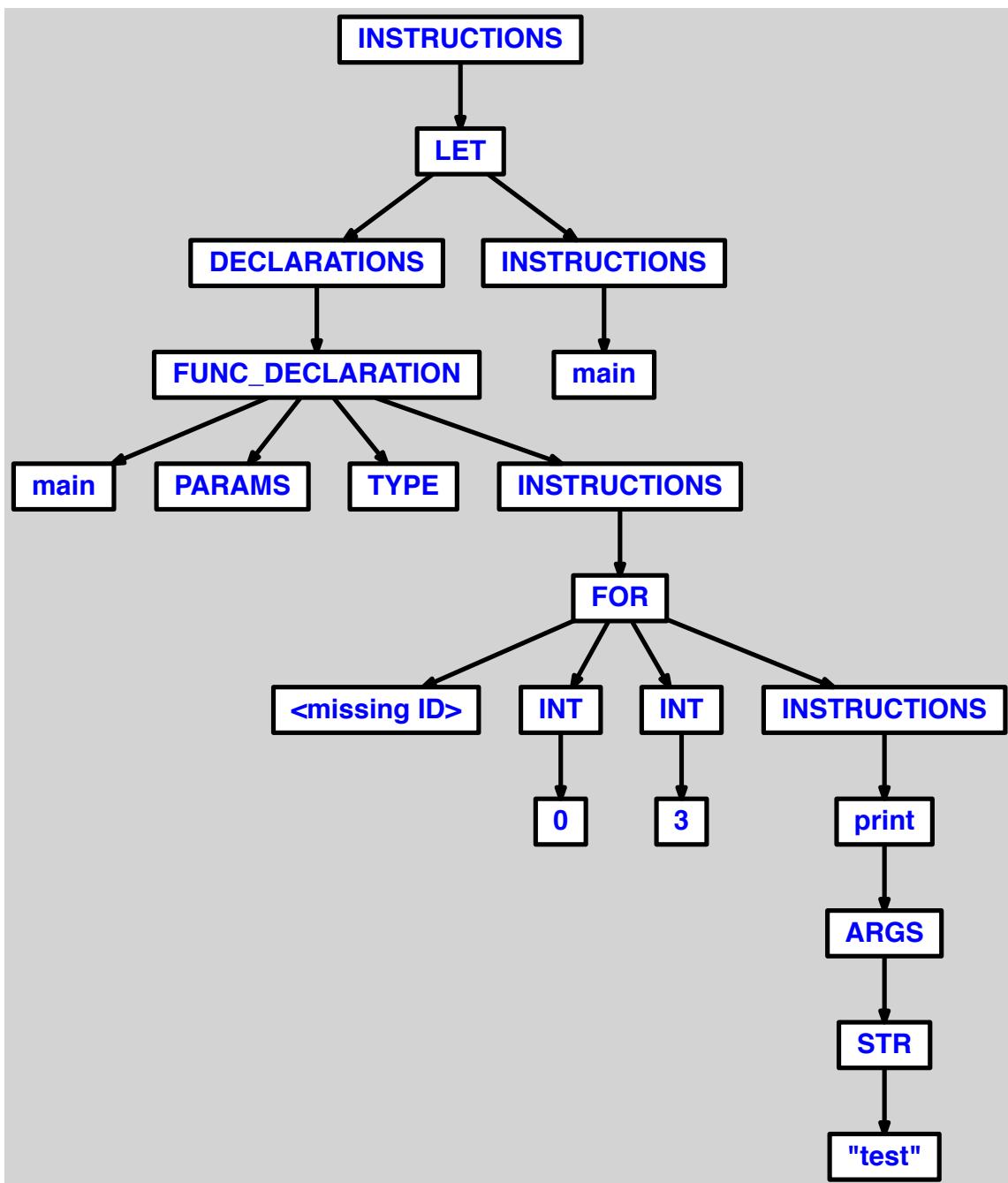
### 7.1.2 <for> avec affectation de chaine pour la borne superieure de l'iteration

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



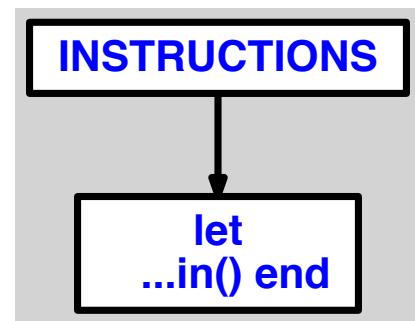
### 7.1.3 <for> avec oubli de l'identifiant du compteur

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



#### 7.1.4 <for> avec oubli du <for>

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



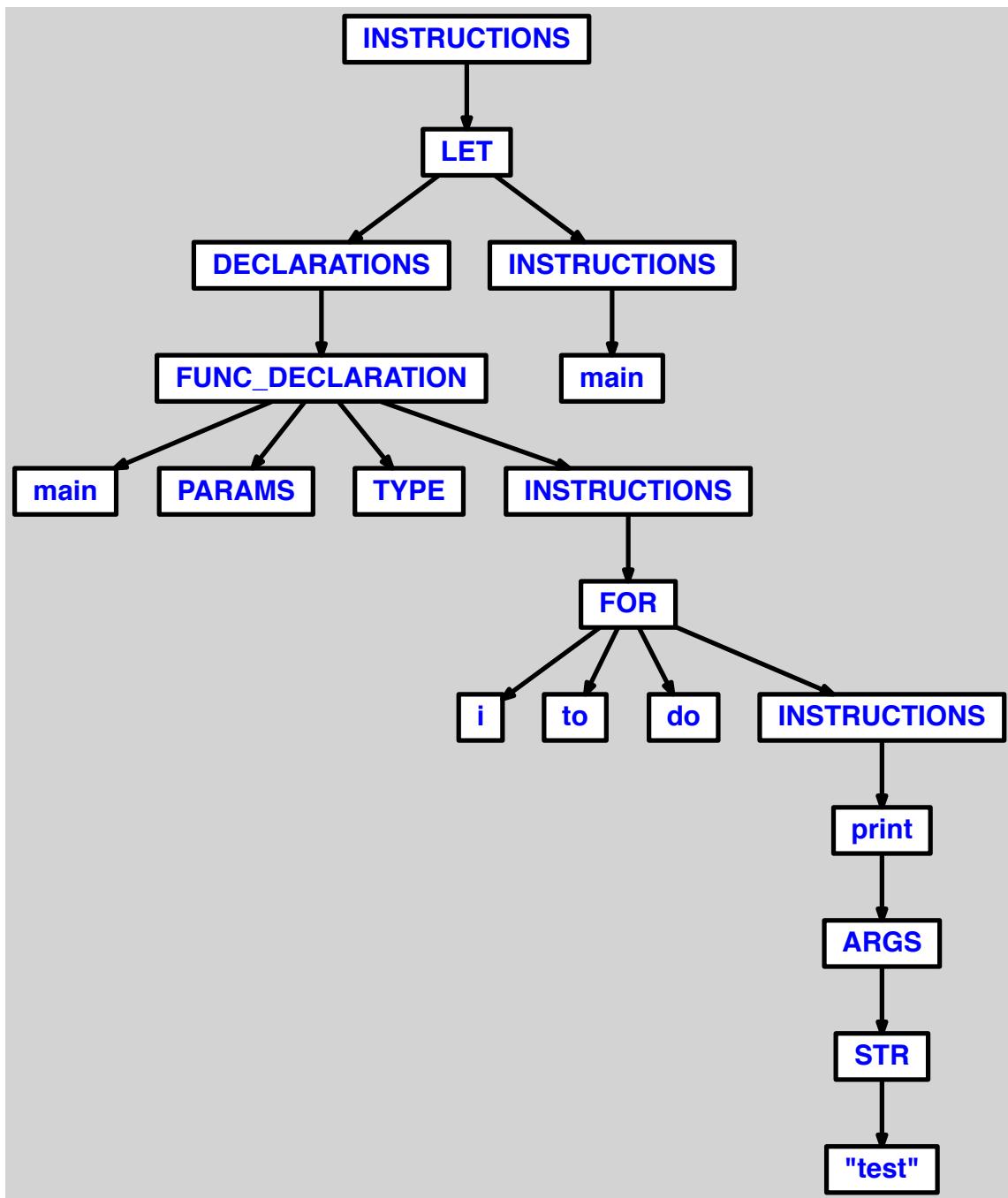
### 7.1.5 <for> "express"

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

Pas d'AST, problème de syntaxe.

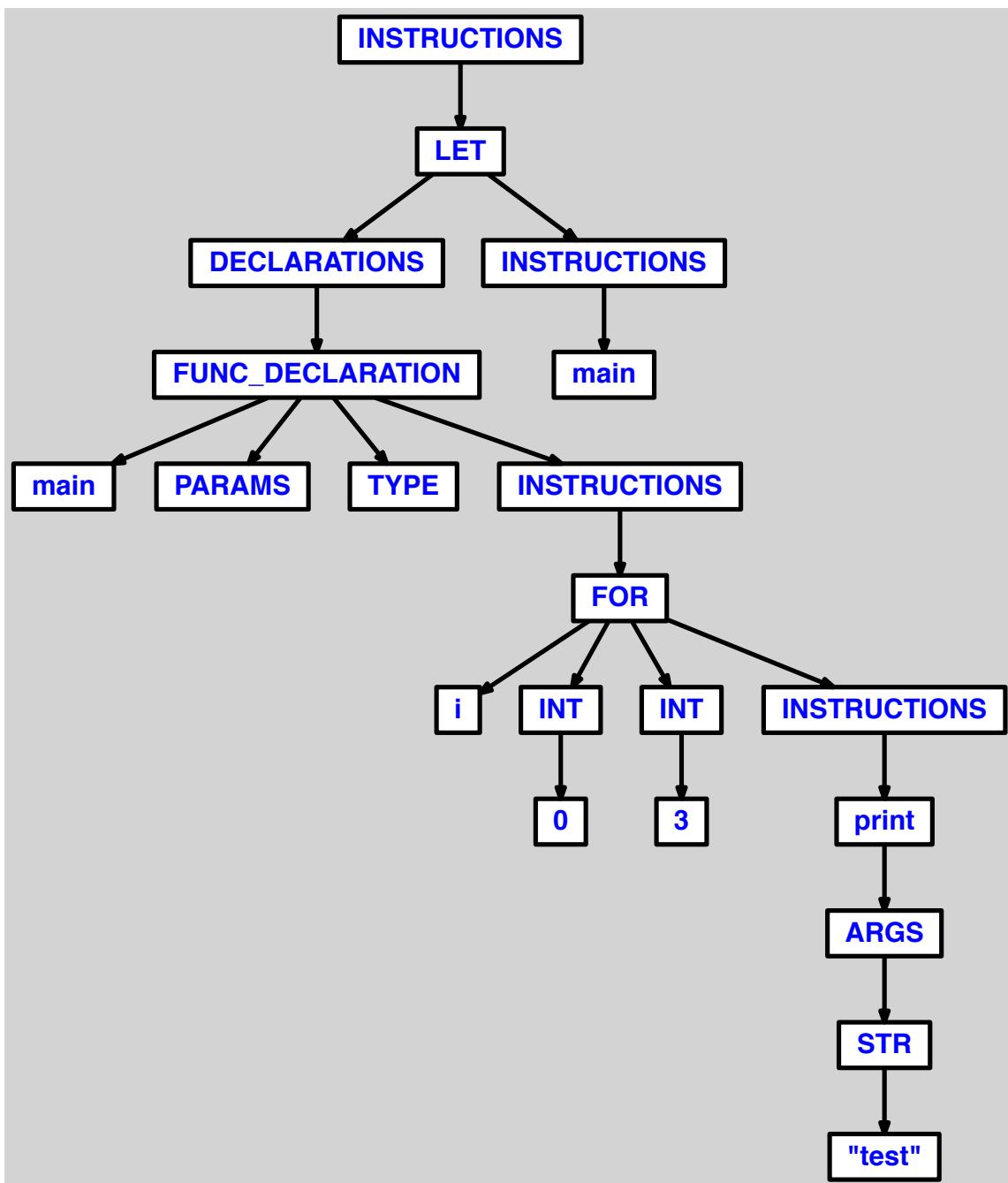
### 7.1.6 <for> avec affectation de chaines pour le compteur et la borne maximale de l'iteration

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



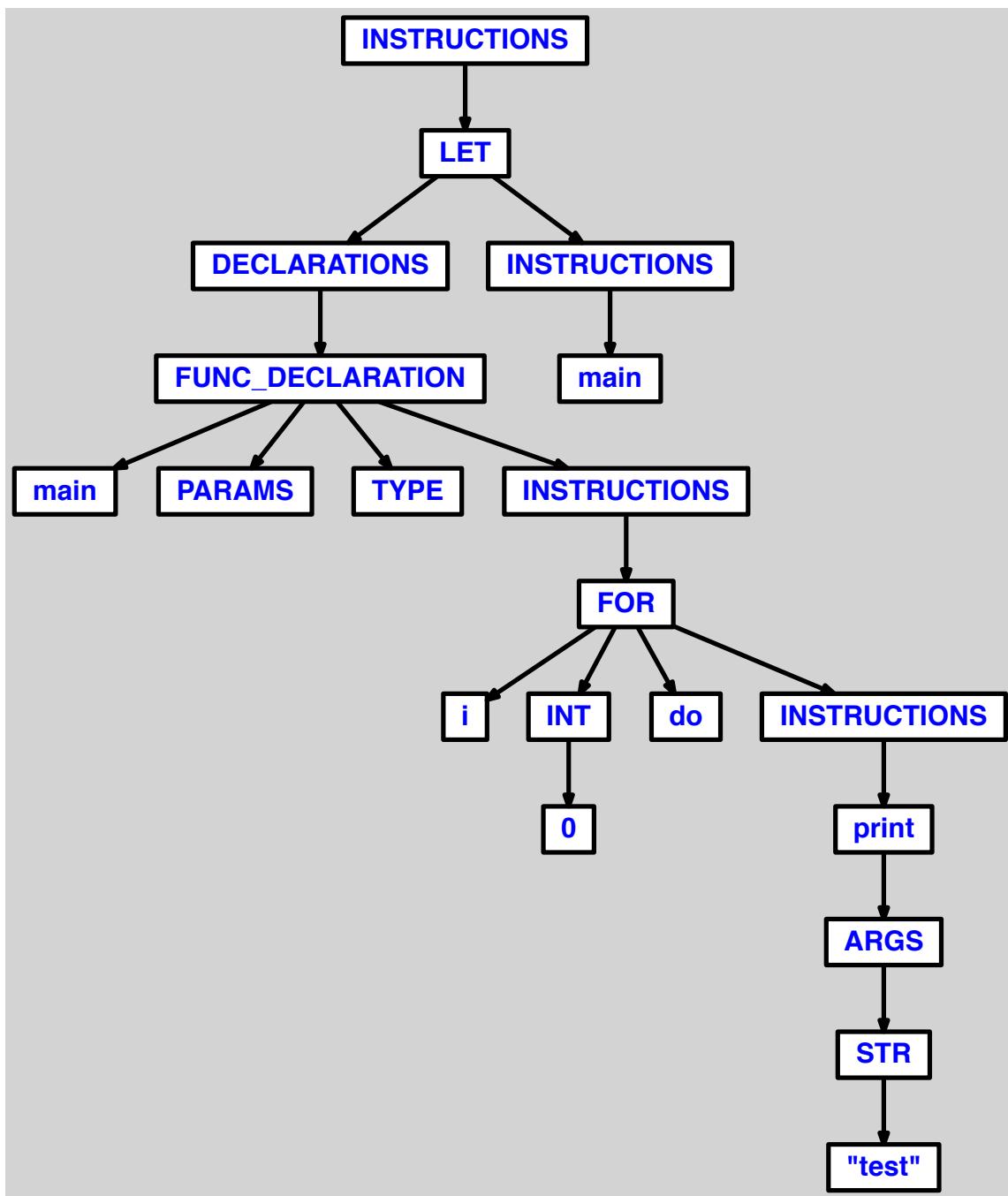
### 7.1.7 <for> avec oubli du <do>

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



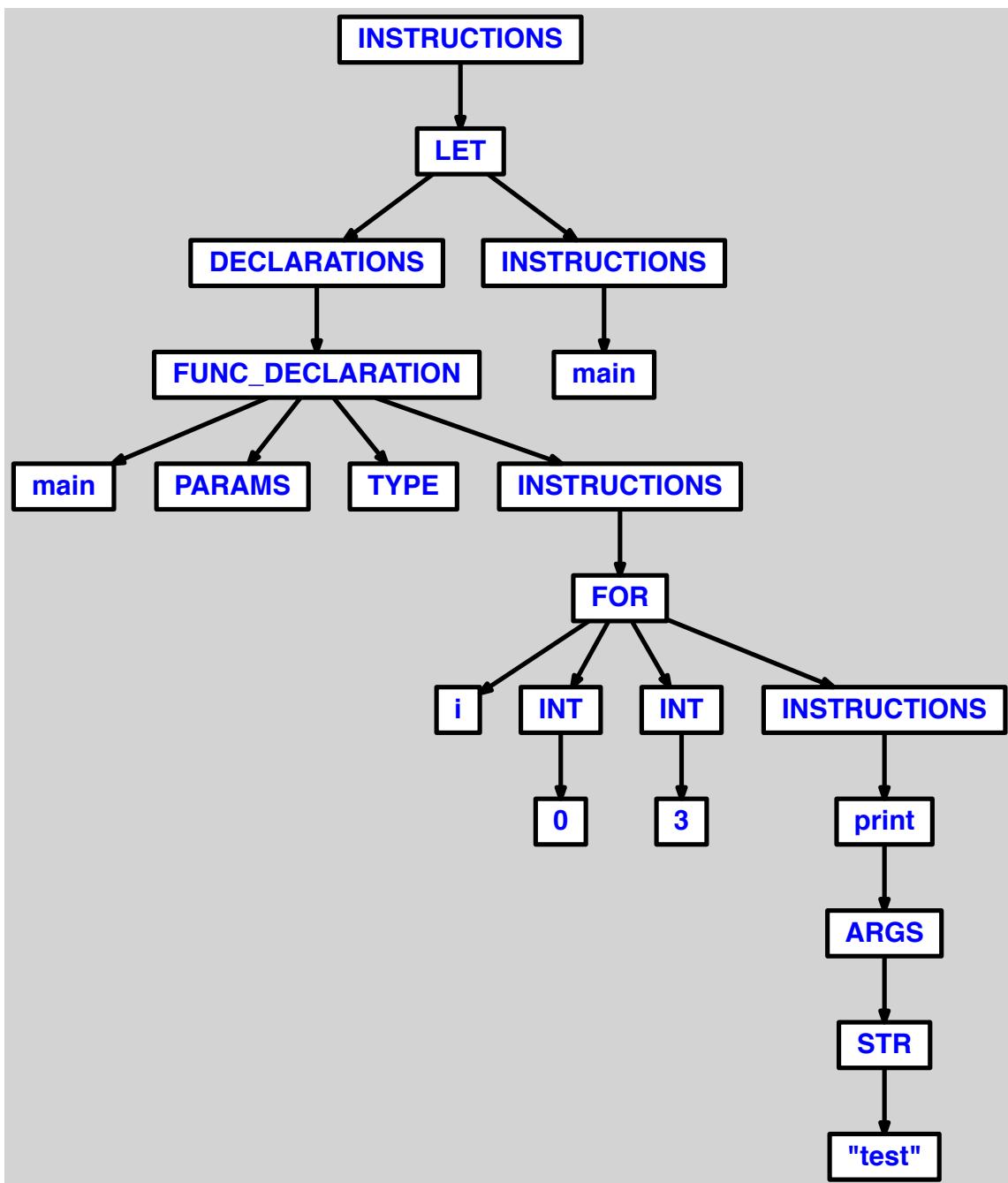
### 7.1.8 <for> avec oubli de la borne maximale de l'itération

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



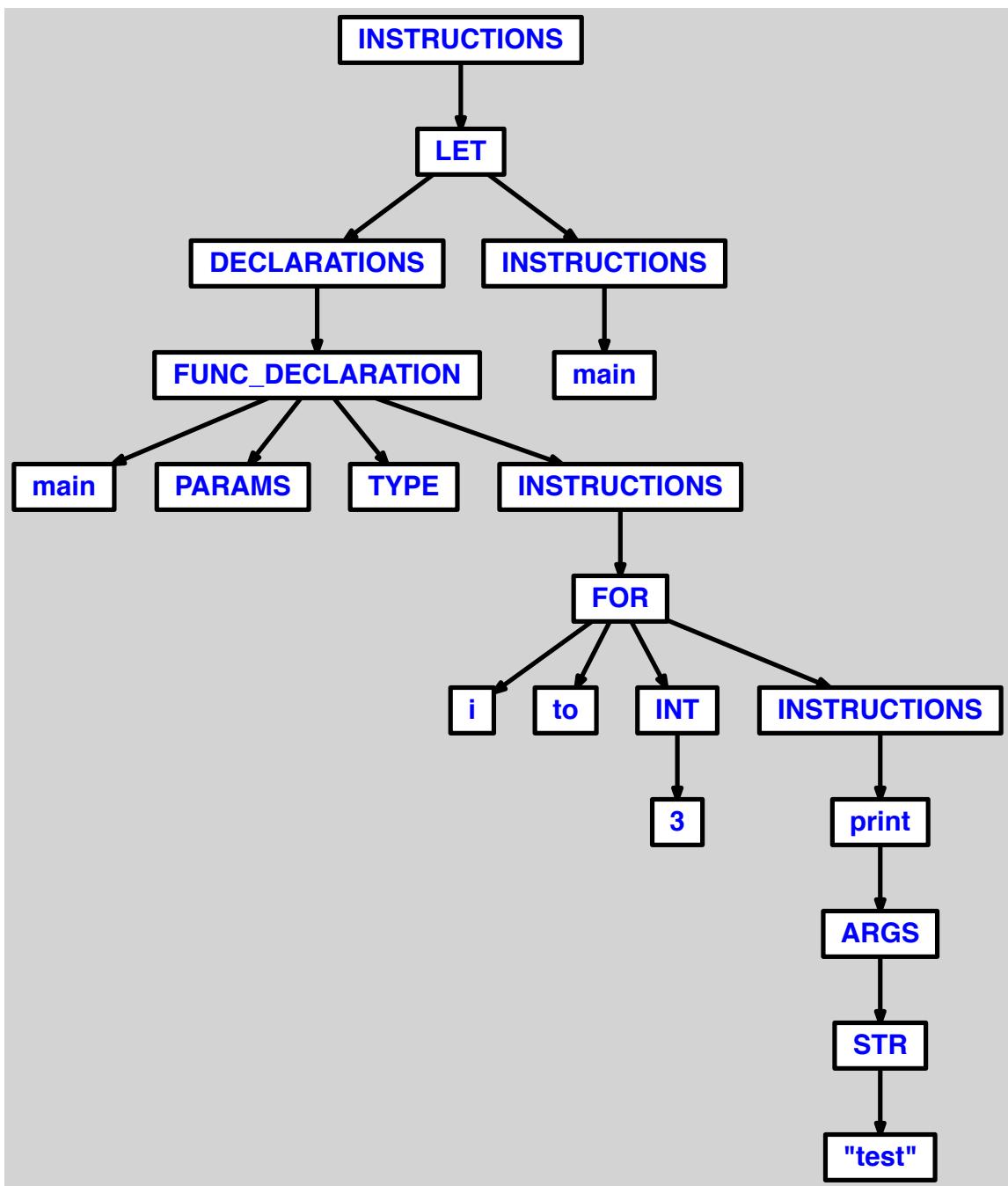
### 7.1.9 <for> avec oubli du <to>

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



### 7.1.10 <for> avec oubli de la borne inférieure de l'itération

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



### 7.1.11 <for> avec oubli du <=>

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

Pas d'AST, problème de syntaxe.

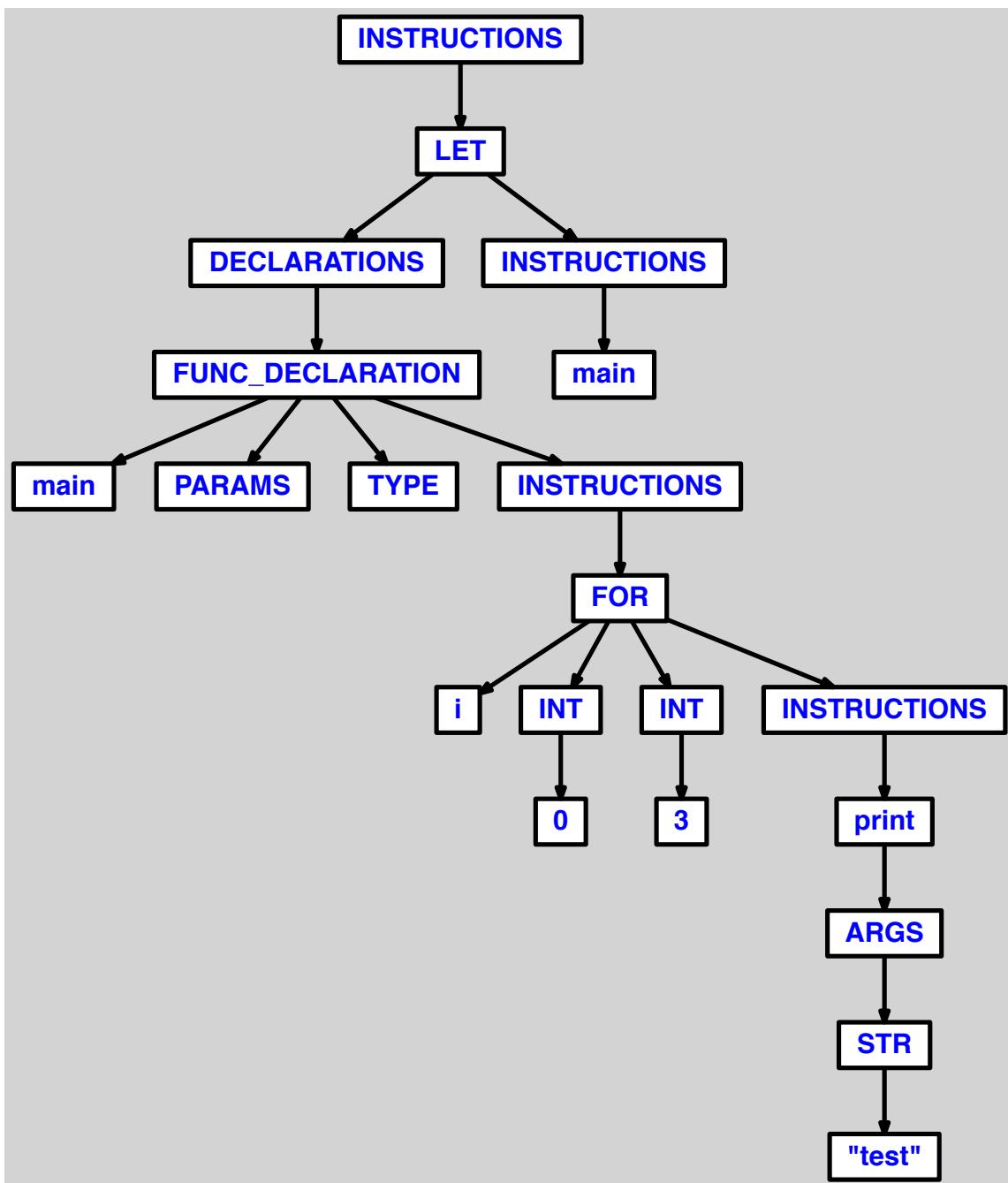
### 7.1.12 <for> avec oubli du <:>

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

Pas d'AST, problème de syntaxe.

### 7.1.13 <for> avec oubli du <:=>

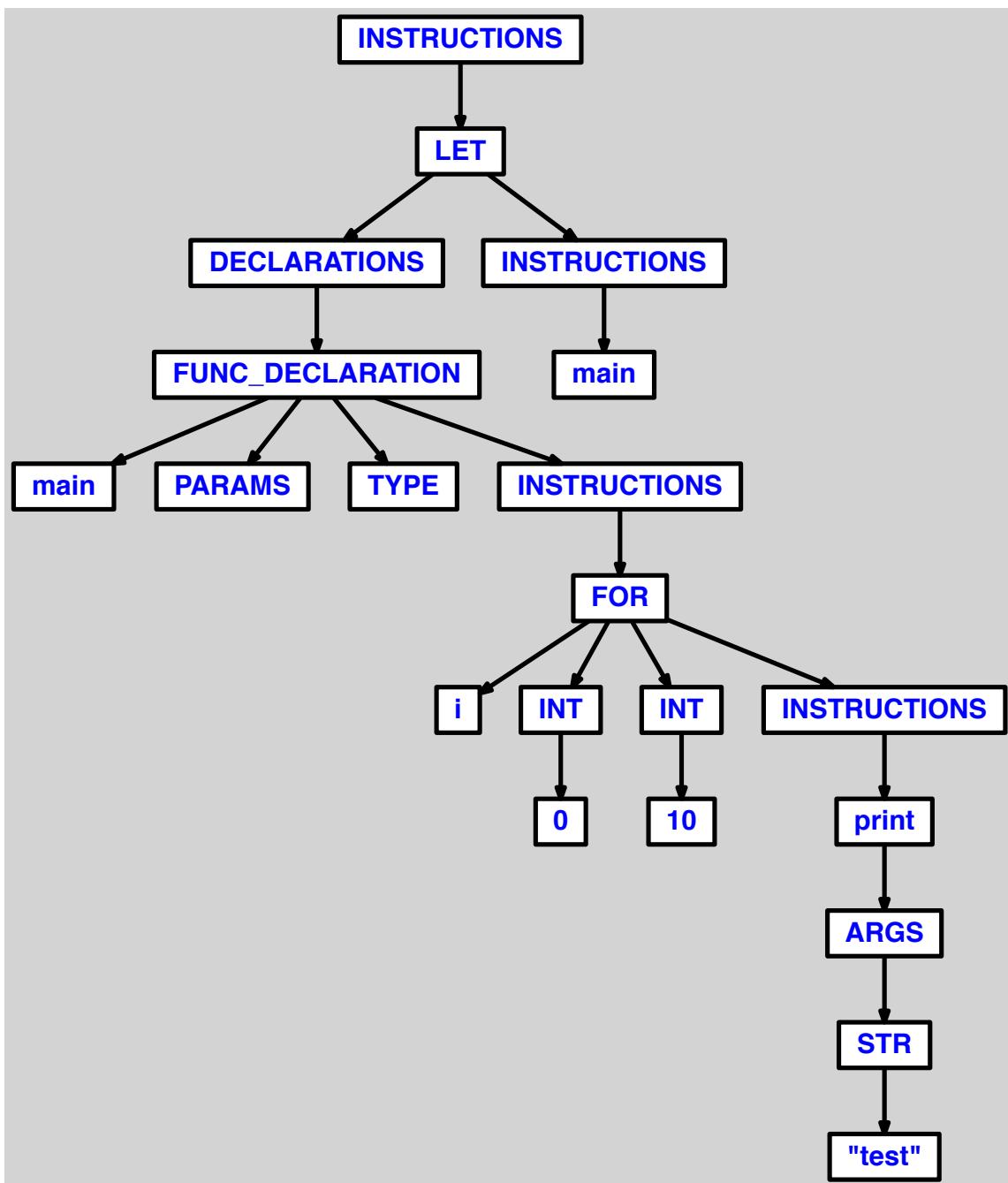
```
1 let
2   function main() =
3     for i 0 to 3 do
4       print("test")
5   in main() end
```



## 7.2 OK

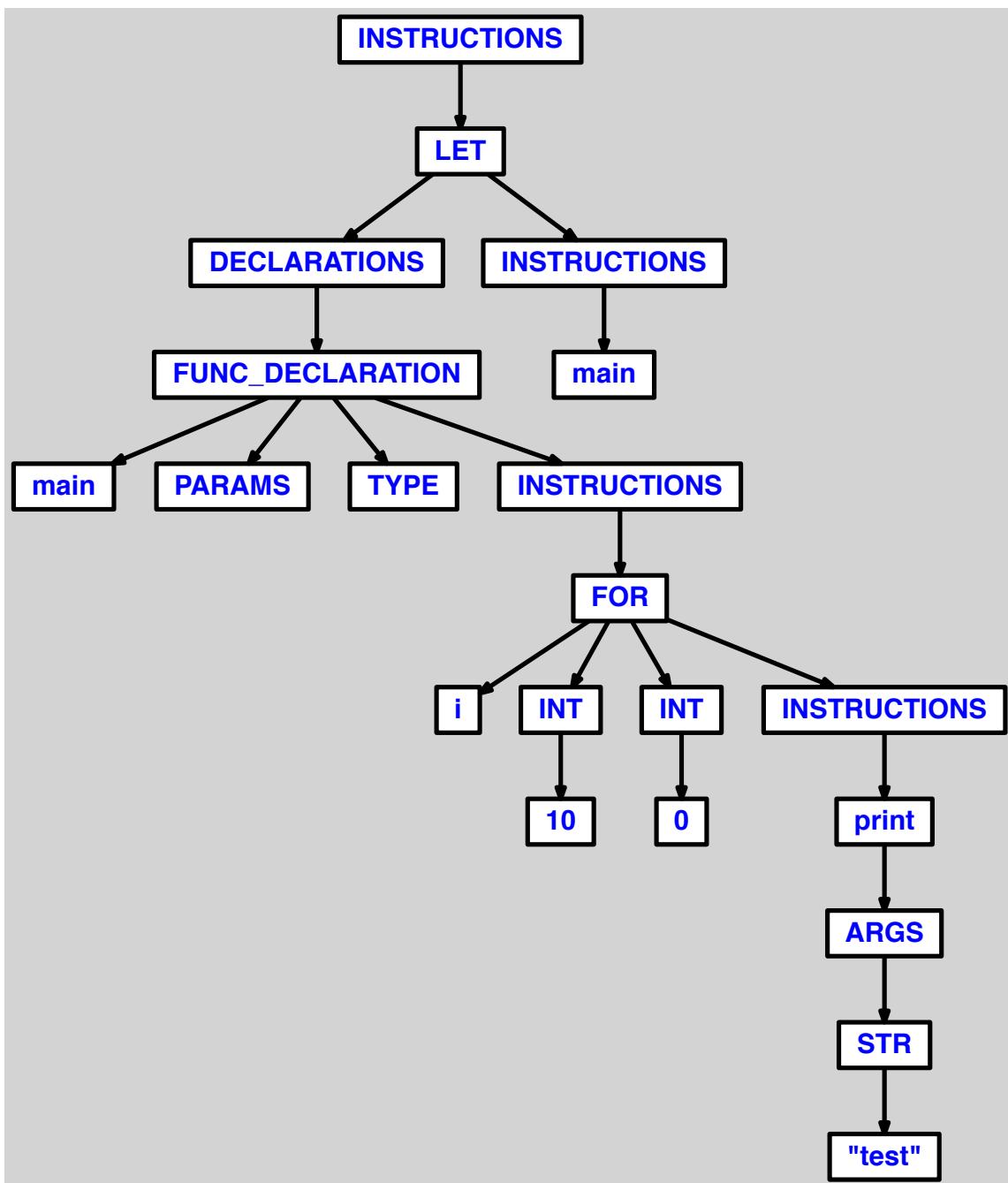
### 7.2.1 <for> simple croissant

```
1 let
2   function main() =
3     for i := 0 to 10 do
4       print("test")
5 in main() end
```



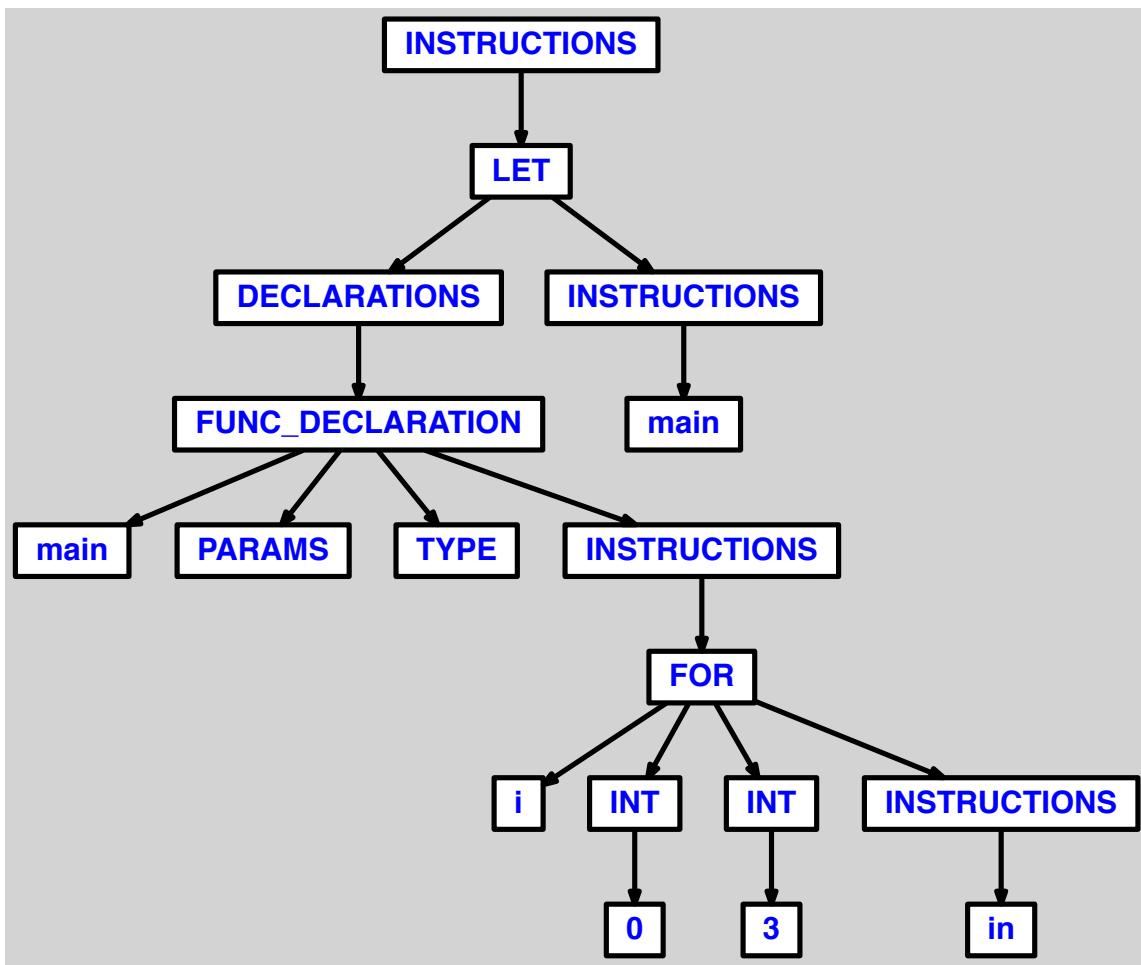
### 7.2.2 <for> simple decroissant

```
1 let
2   function main() =
3     for i := 10 to 0 do
4       print("test")
5 in main() end
```



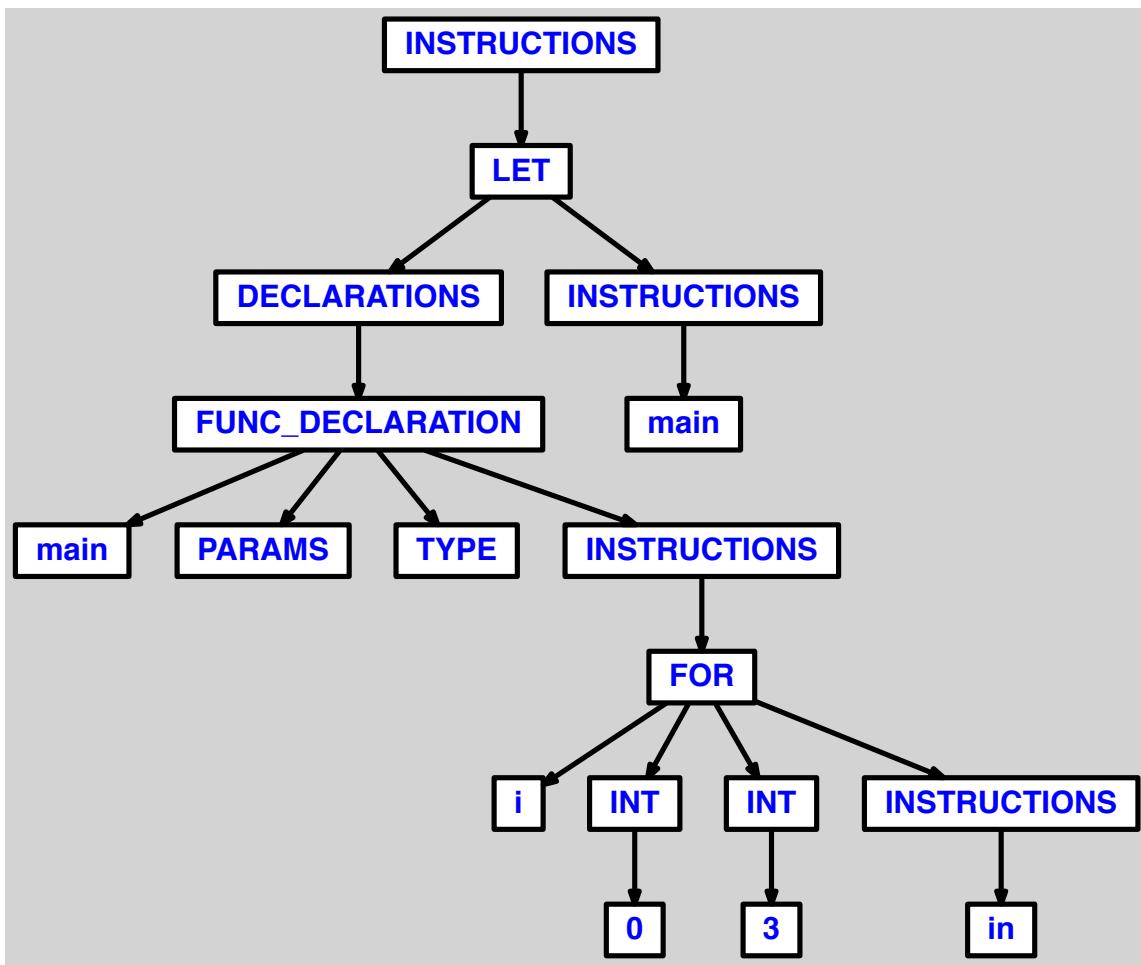
### 7.2.3 <for> sans instruction

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



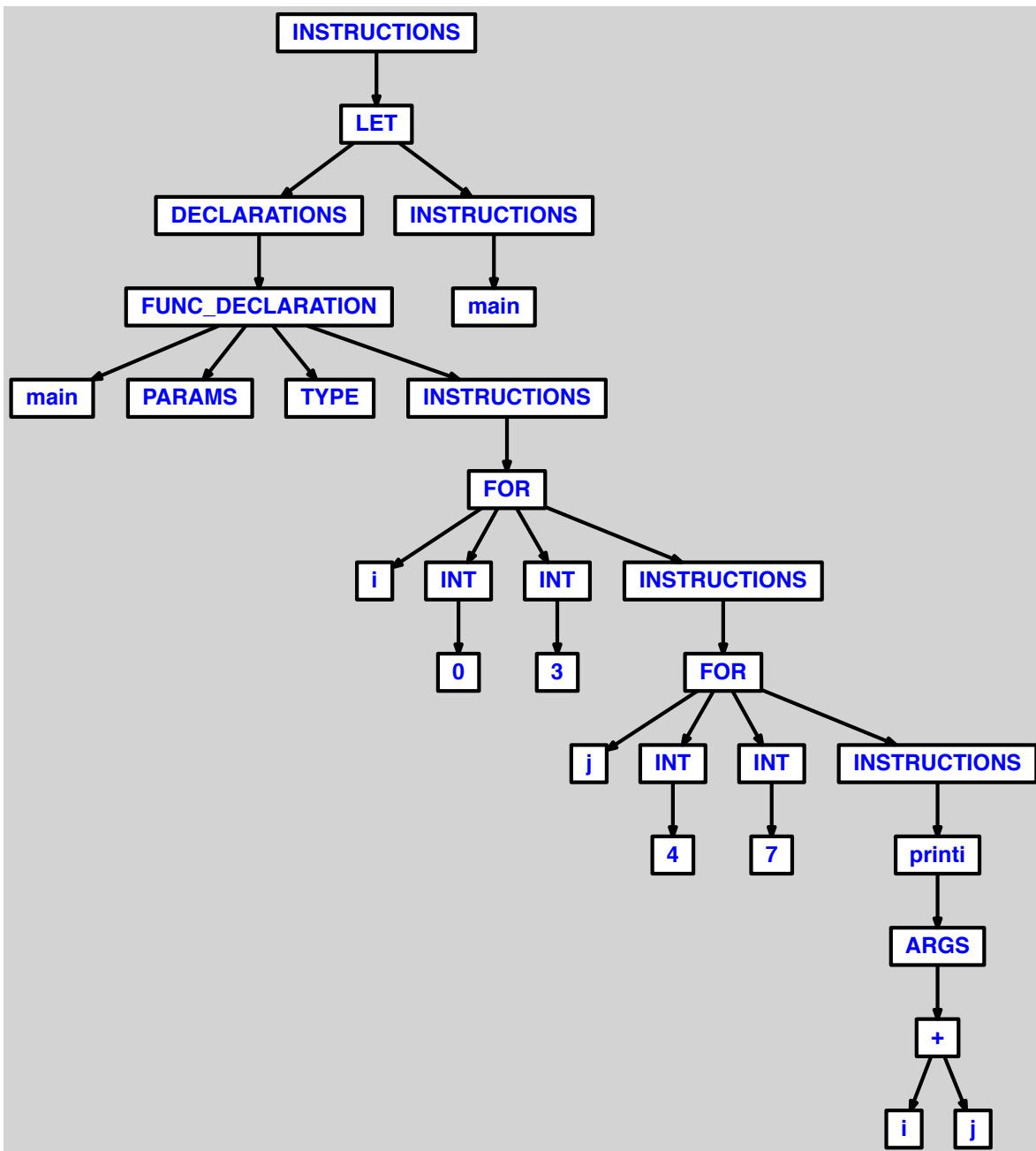
#### 7.2.4 <for> avec ligne vide

```
1 let
2   function main() =
3     for i := 0 to 3 do
4
5   in main() end
```



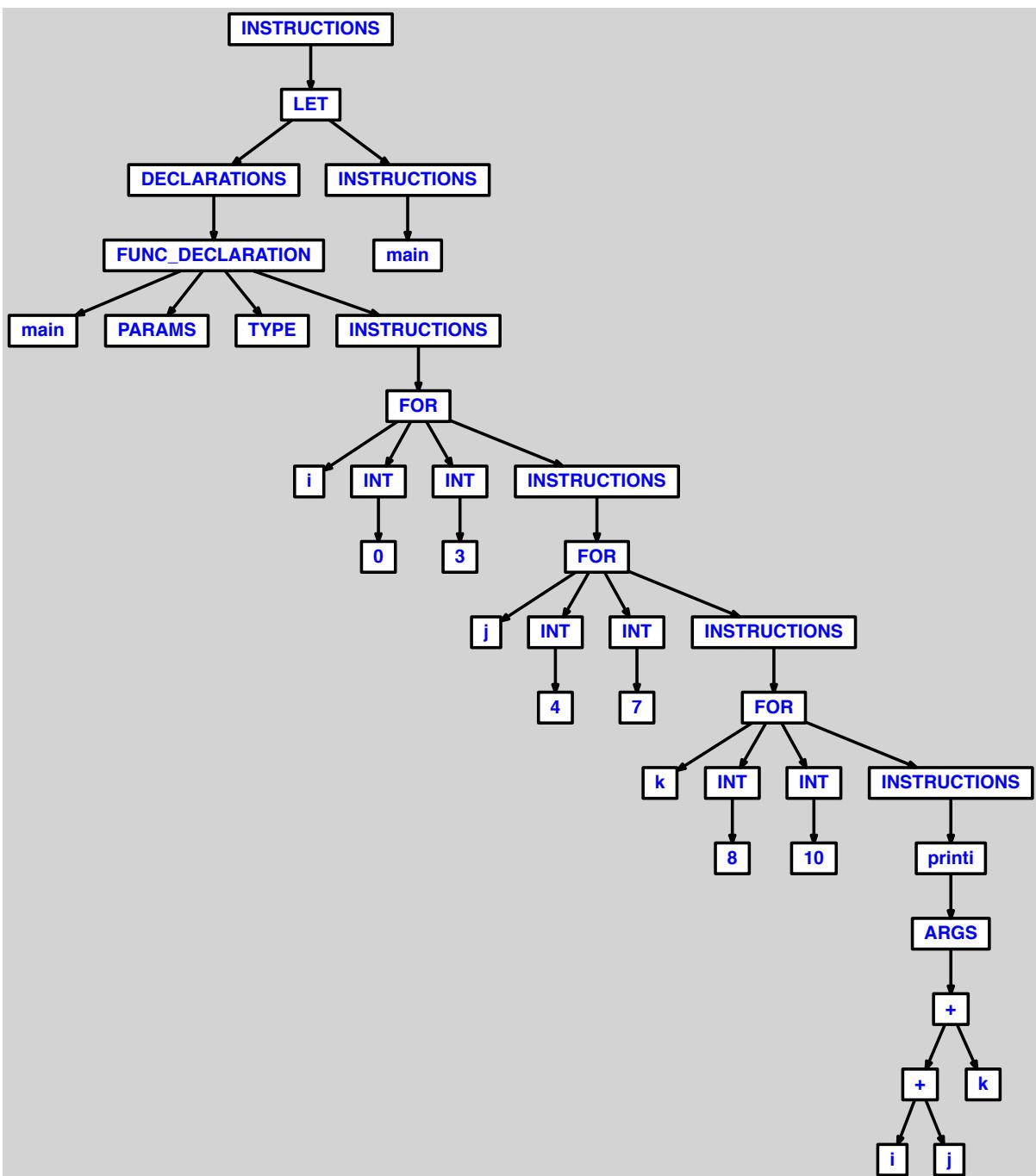
### 7.2.5 double-imbrication de <for>

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



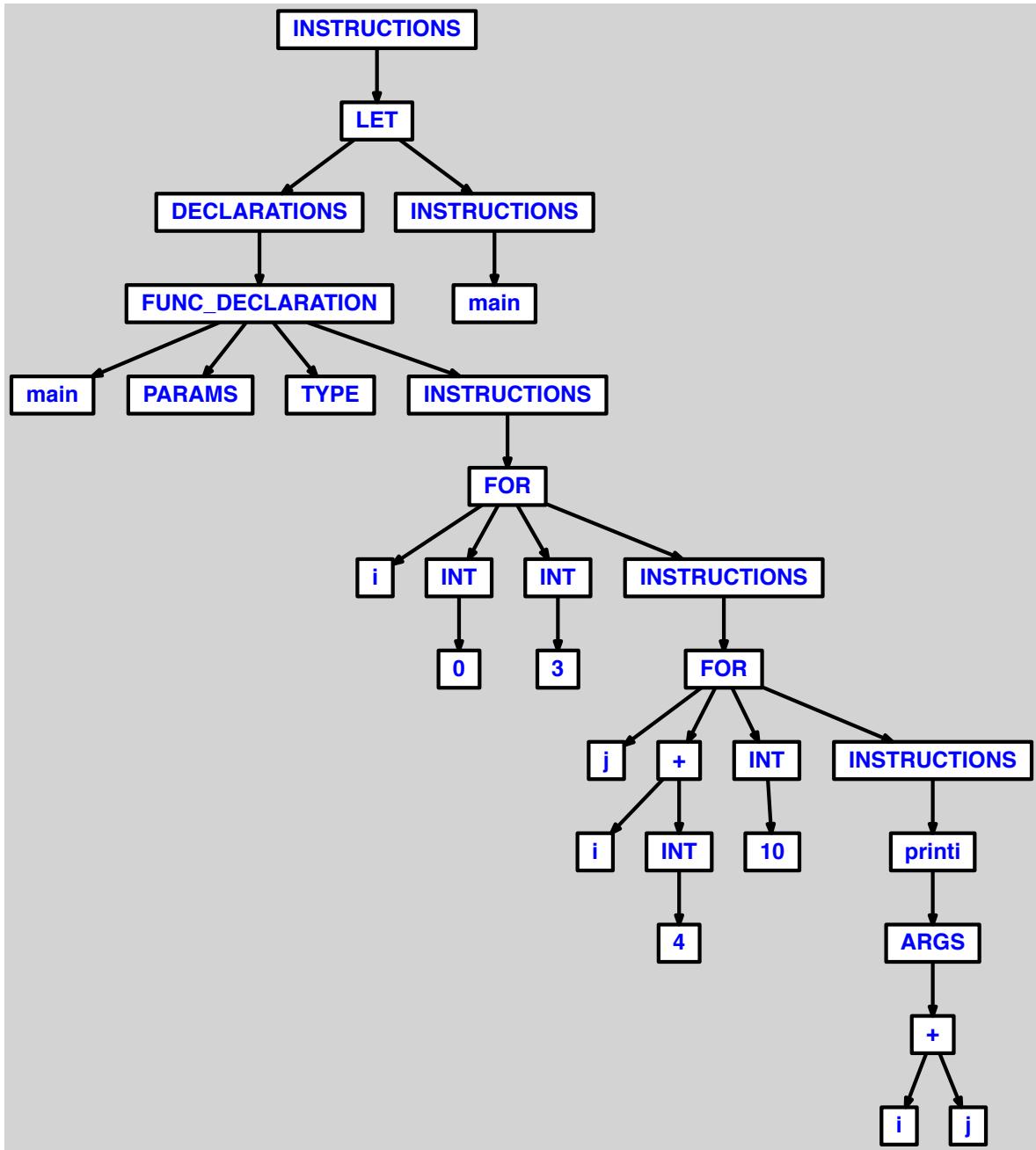
### 7.2.6 triple-imbrication de <for>

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



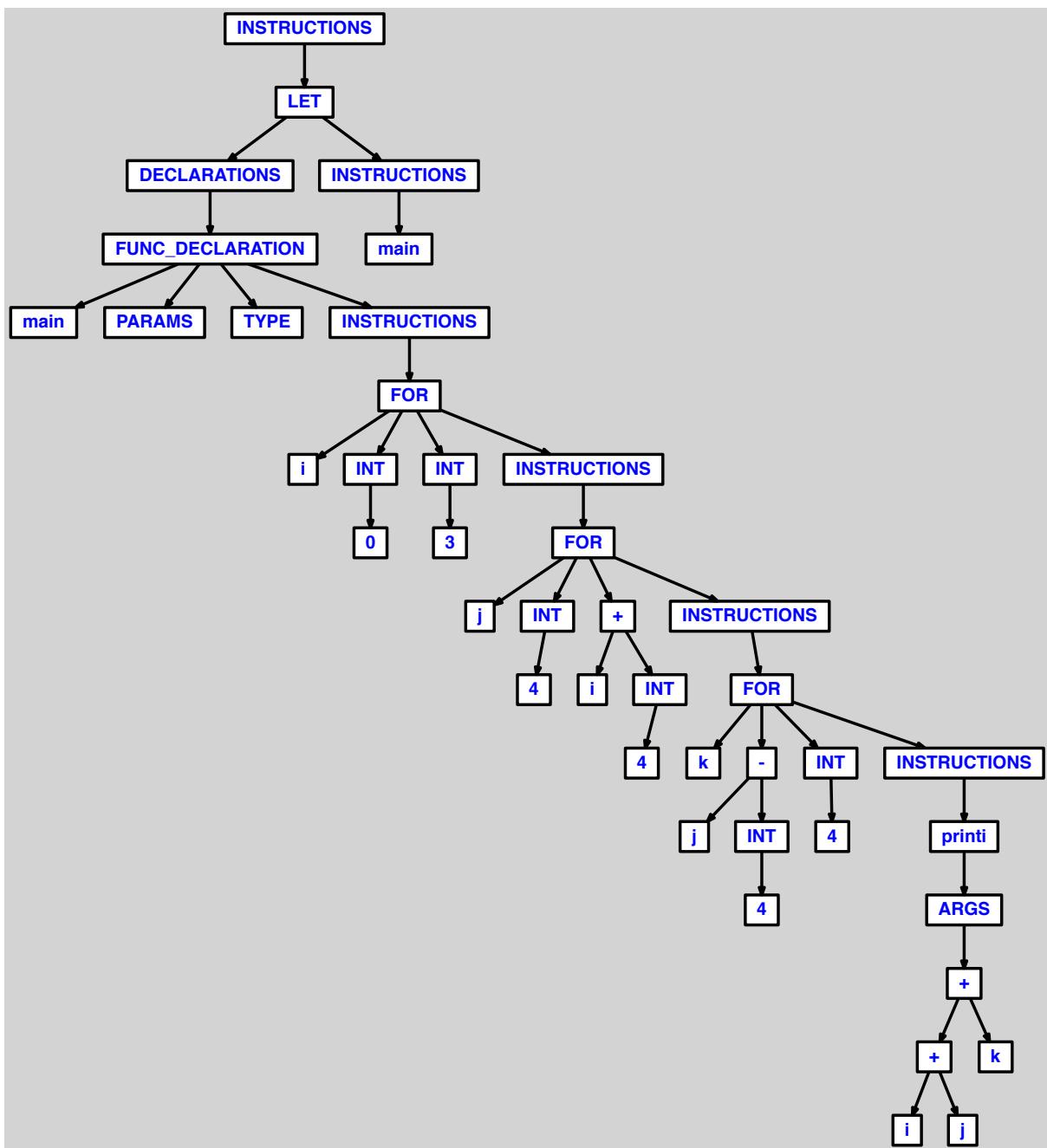
### 7.2.7 <for> avec reutilisation de compteur

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



### 7.2.8 <for> avec reutilisation de compteurs

```
1 let
2   function main() =
3     for i := 0 to 3 do
4       for j := 4 to i+4 do
5         for k := j-4 to 4 do
6           printi(i+j+k)
7 in main() end
```

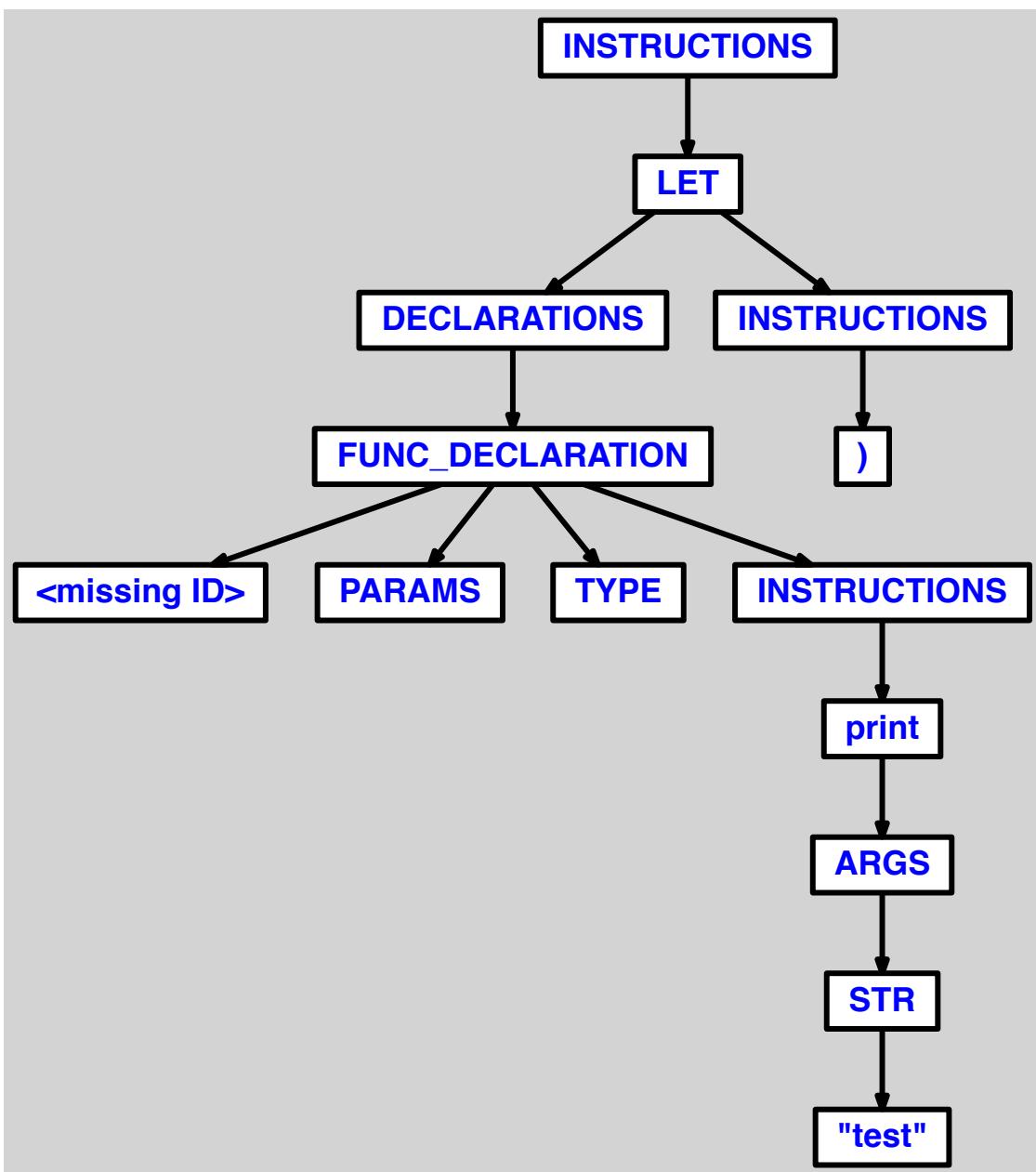


## 8 function

### 8.1 KO

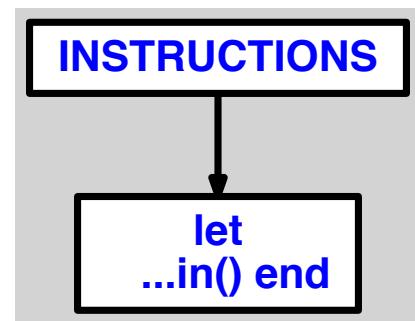
#### 8.1.1 fonction identifiee par caractere special

```
1 let
2   function \() = print("test")
3 in \() end
```



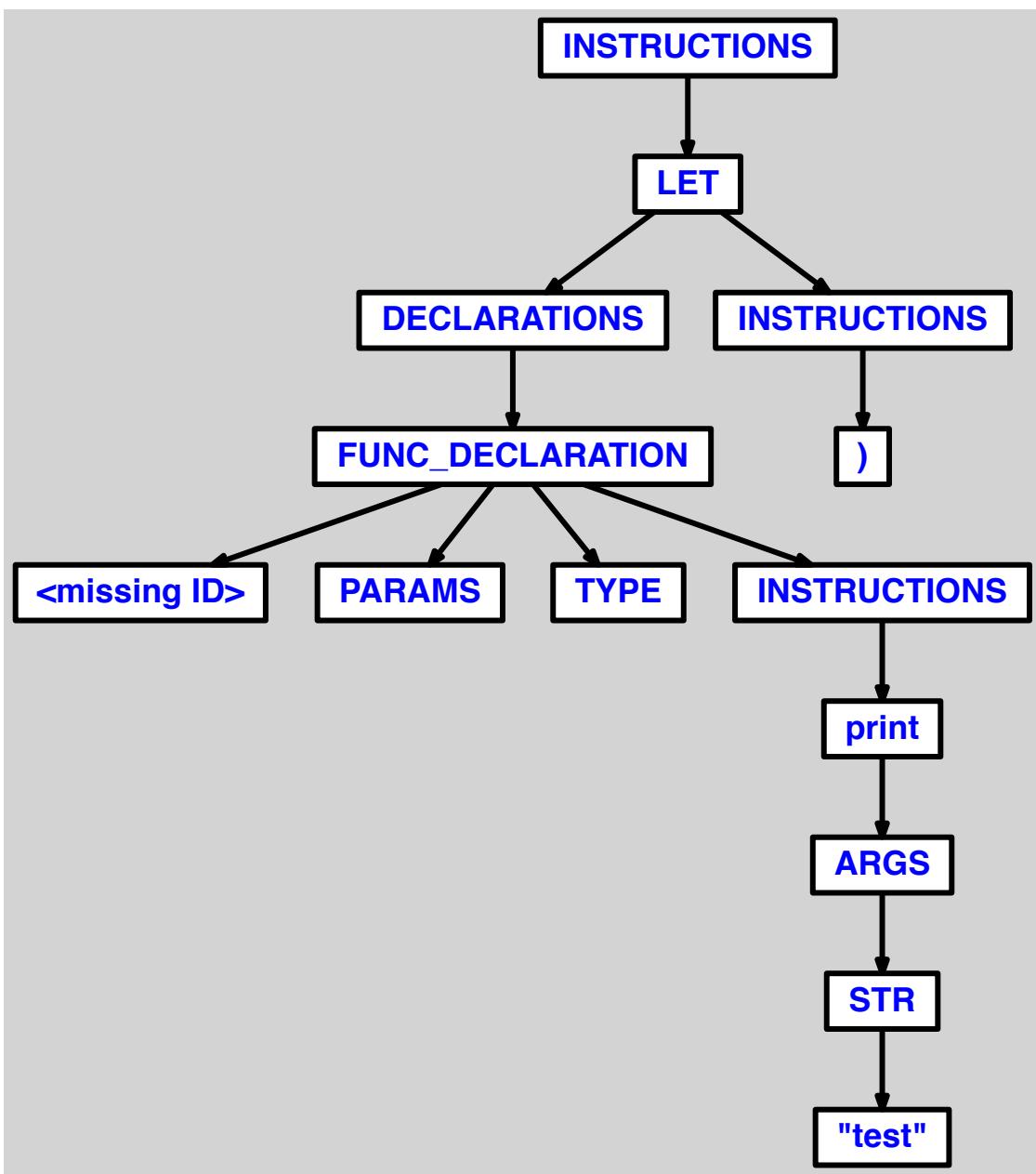
### 8.1.2 oubli de <function>

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



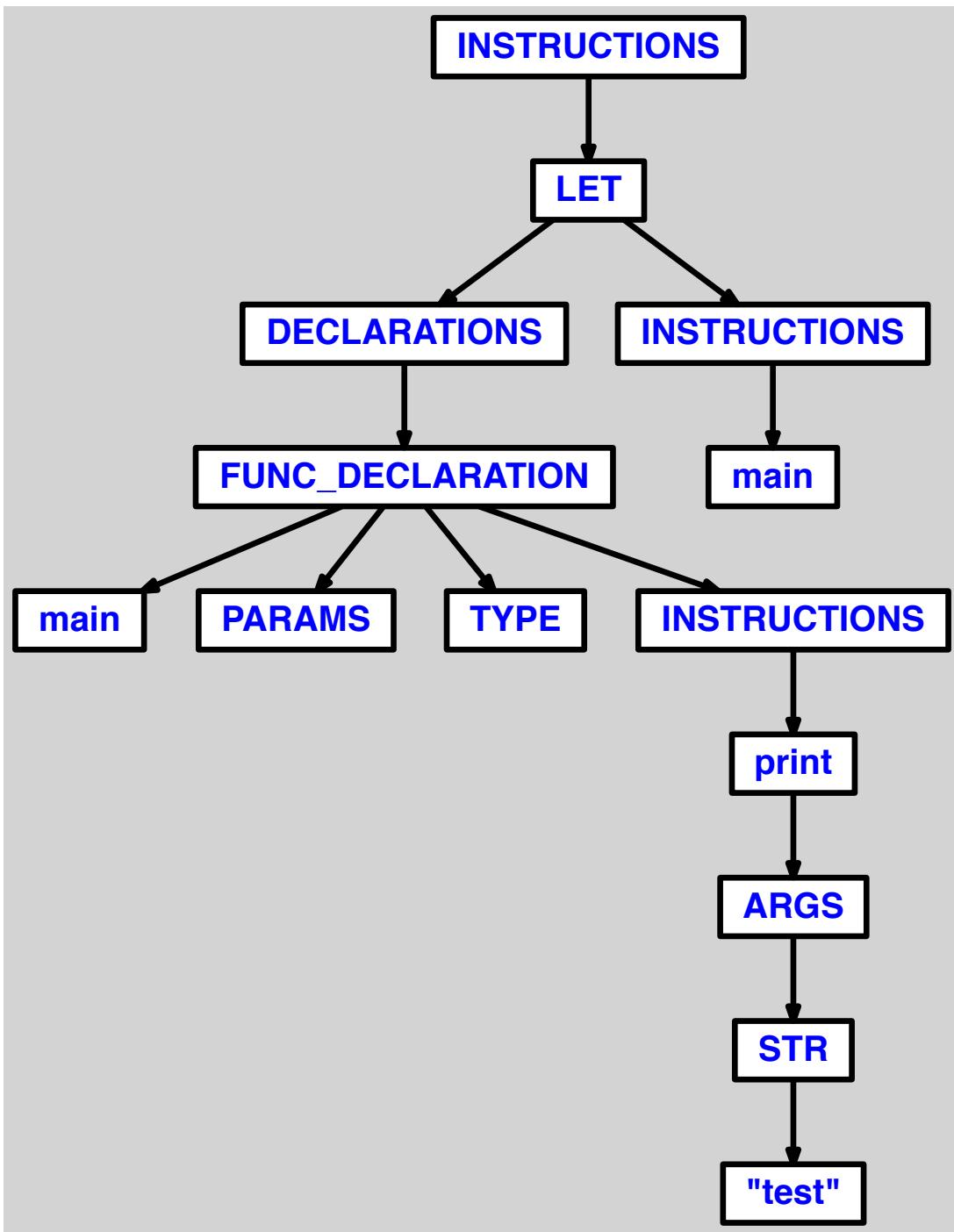
### 8.1.3 fonction sans identifiant

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



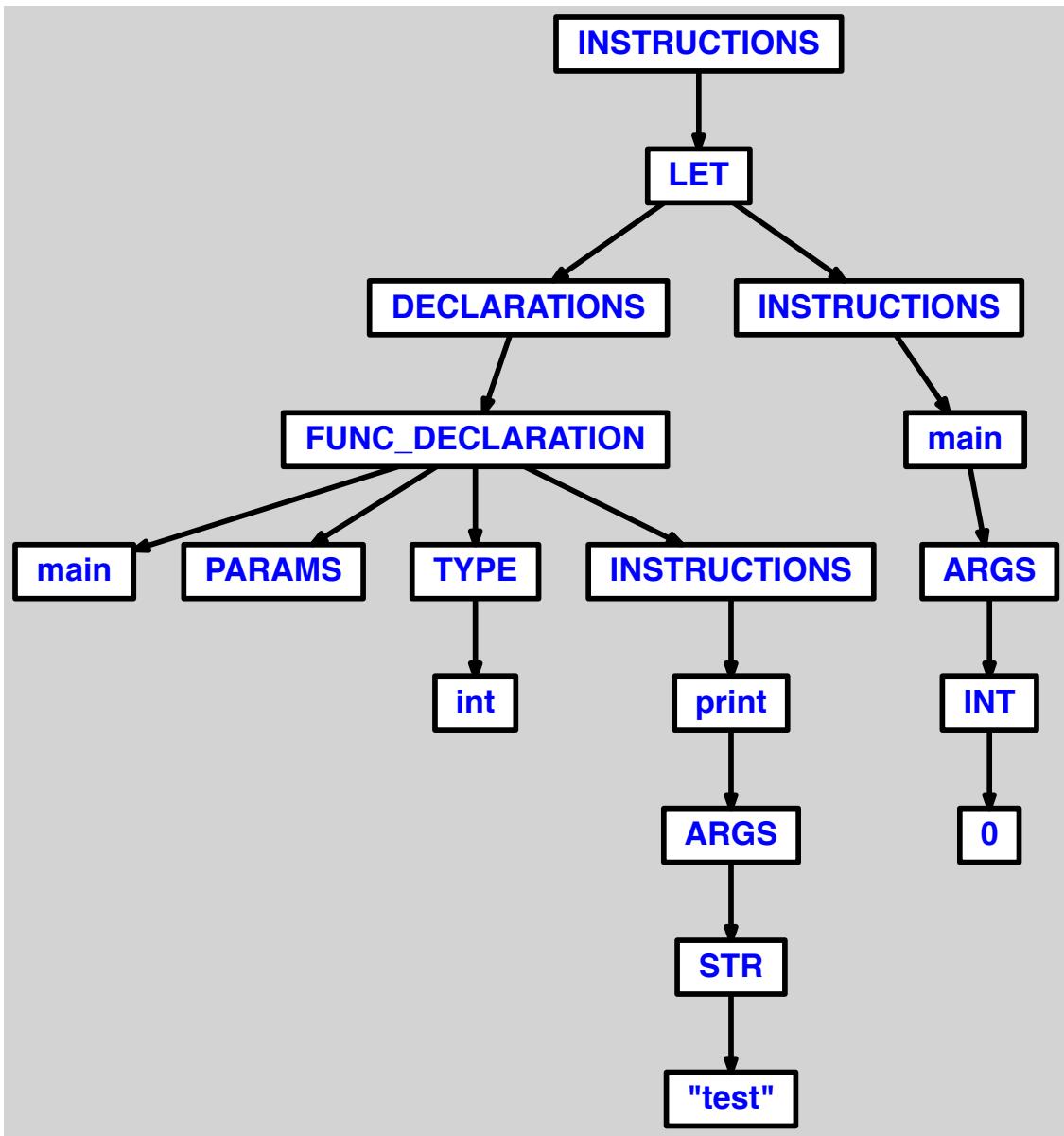
#### 8.1.4 oubli de <(>

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



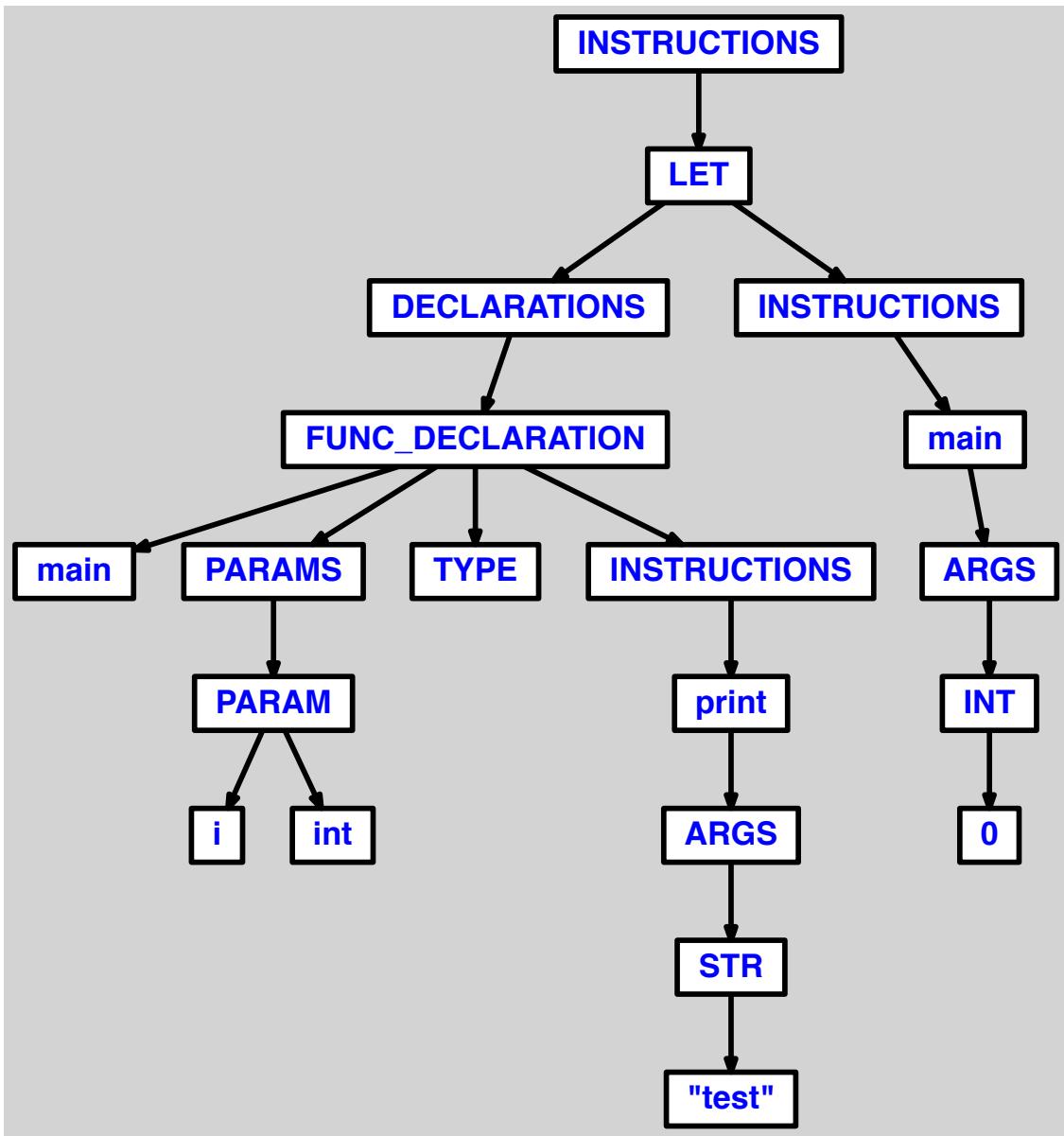
### 8.1.5oubli d'identifiant de parametre

```
1 let
2   function main(: int) = print("test")
3 in main(0) end
```



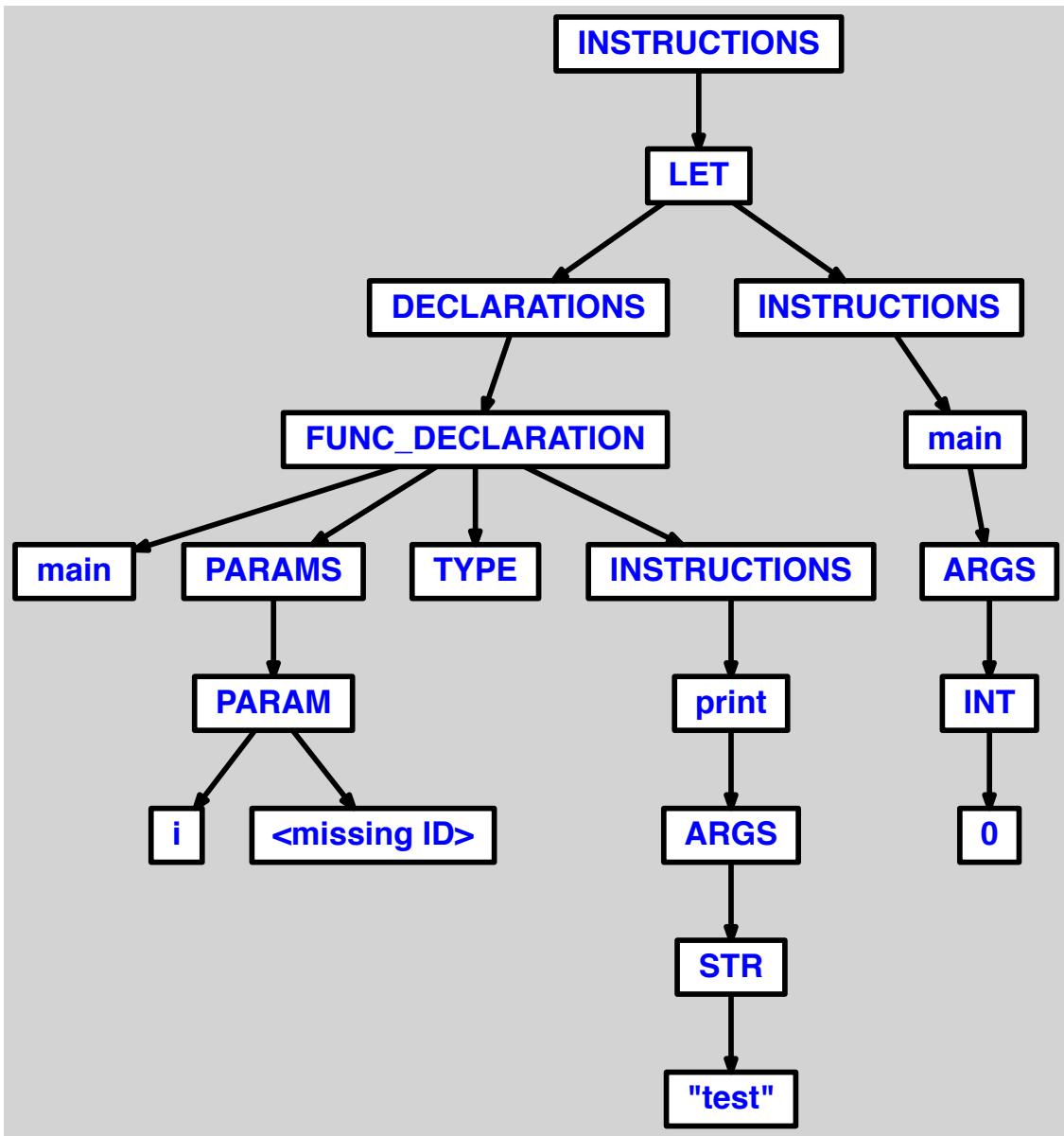
### 8.1.6 oubli de < :> pour parametre

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



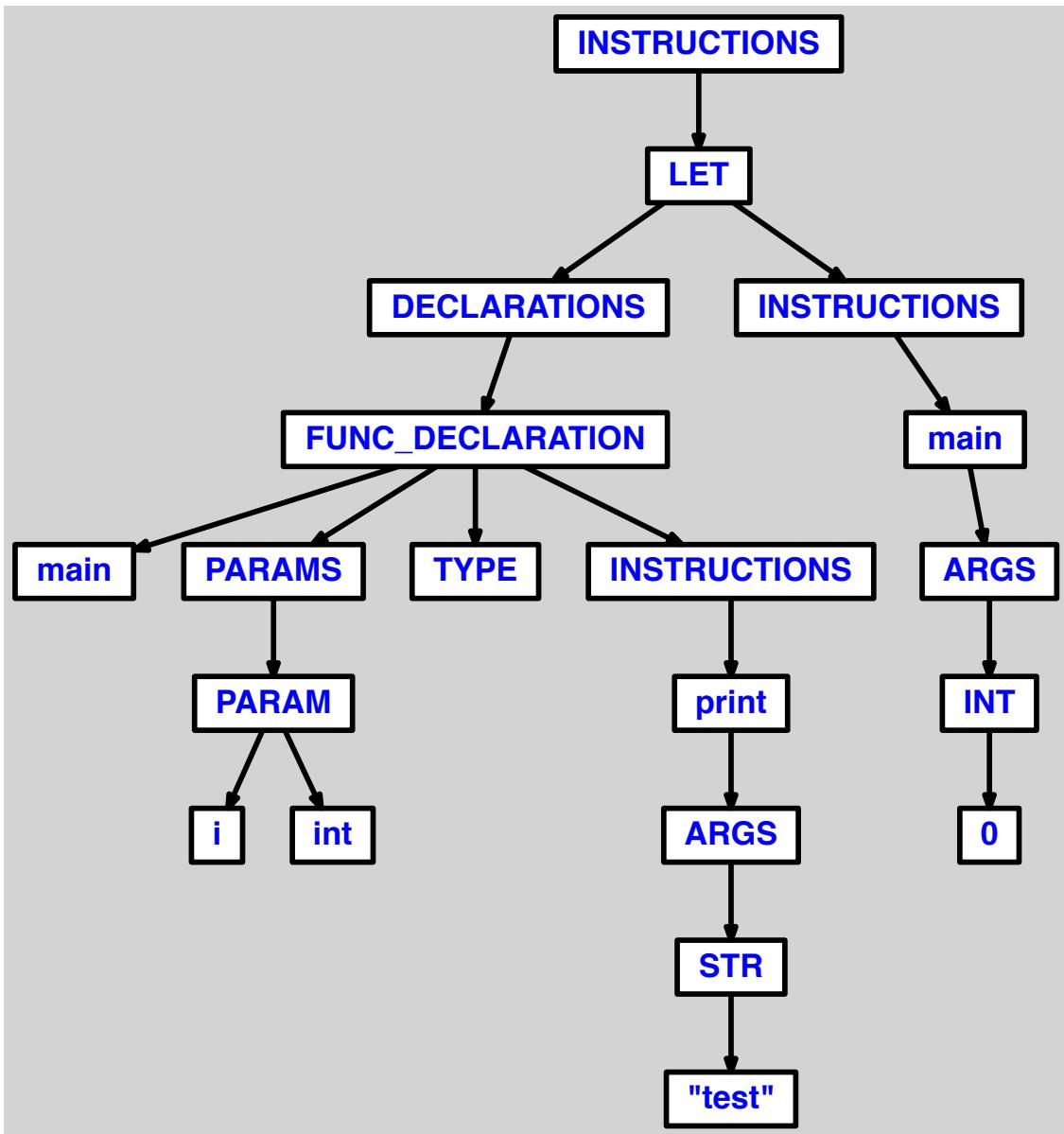
### 8.1.7 oubli de type de parametre

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



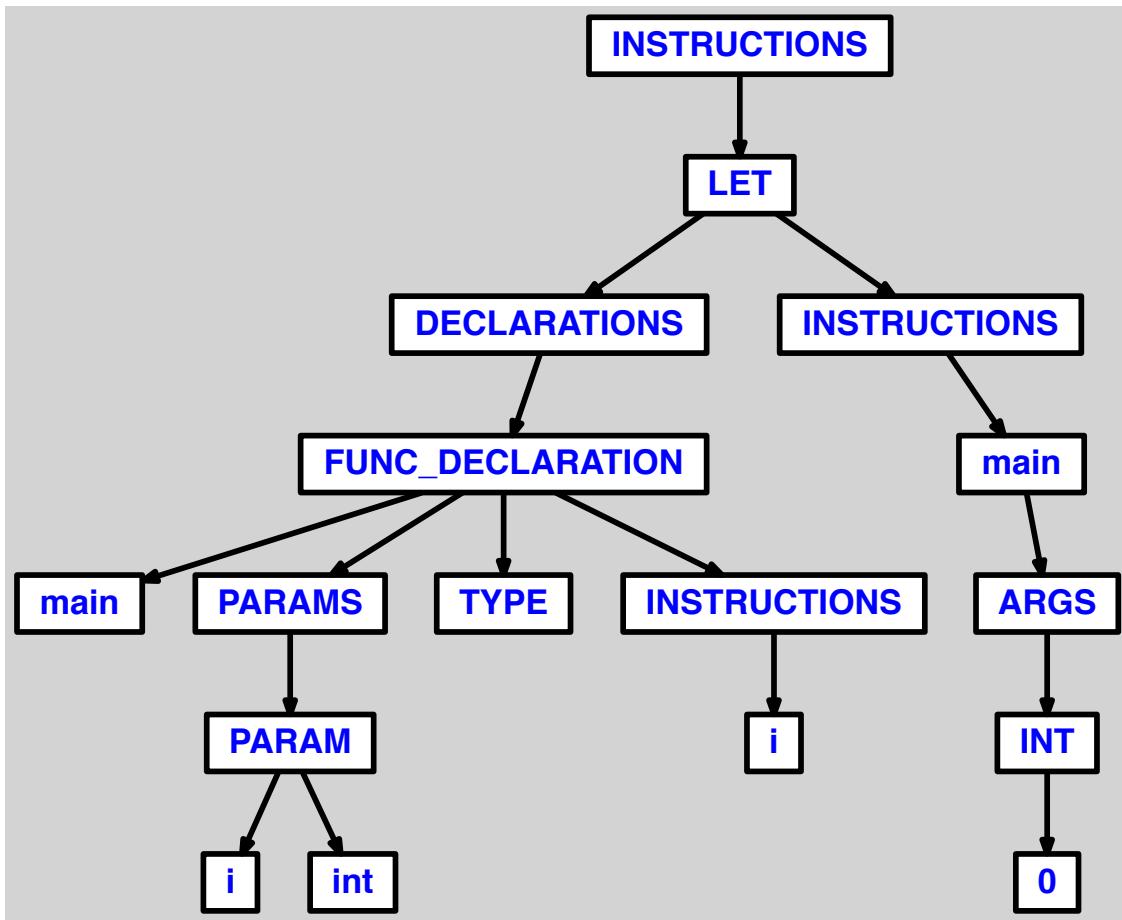
### 8.1.8 oubli de <)>

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



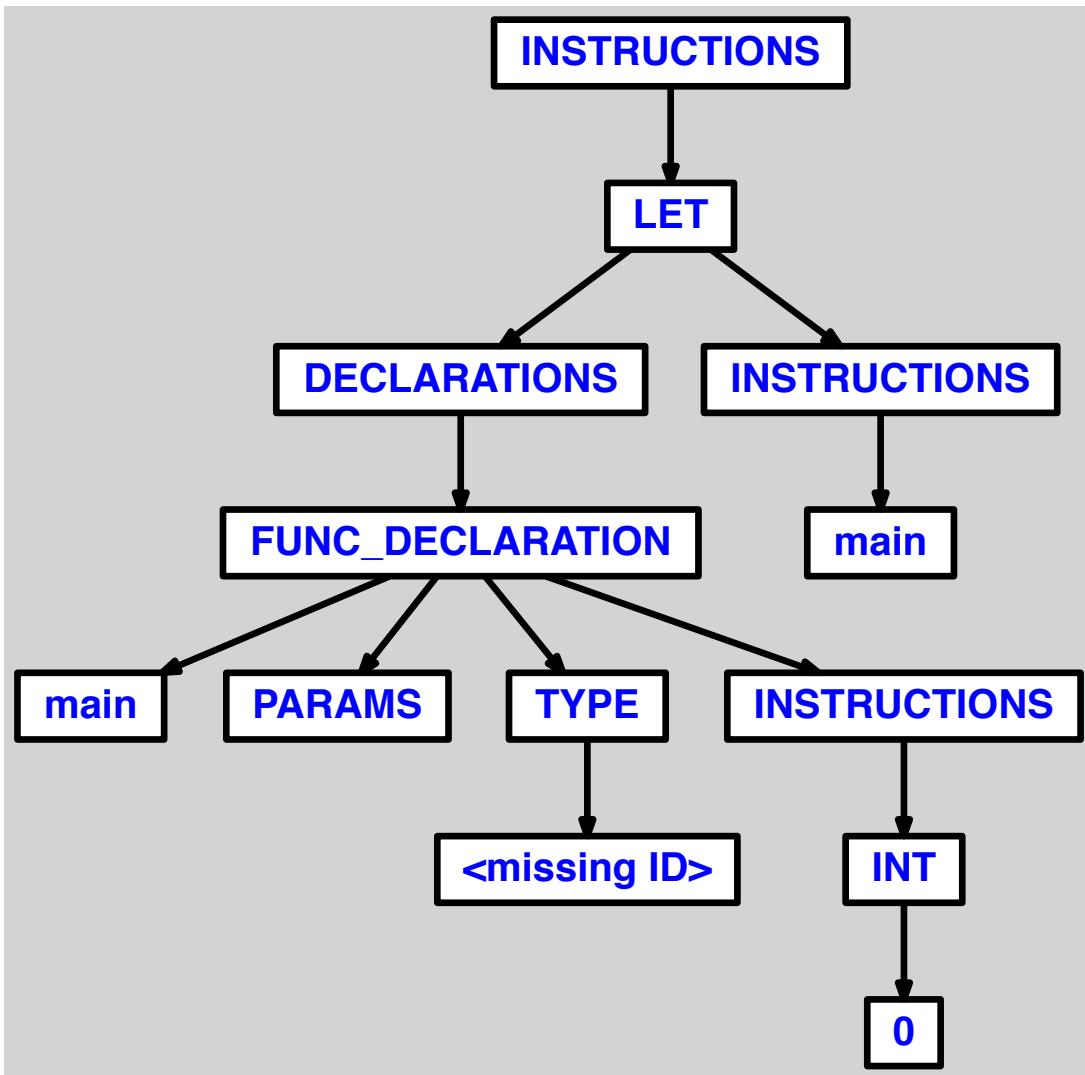
### 8.1.9 oubli de < :> pour type de fonction

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



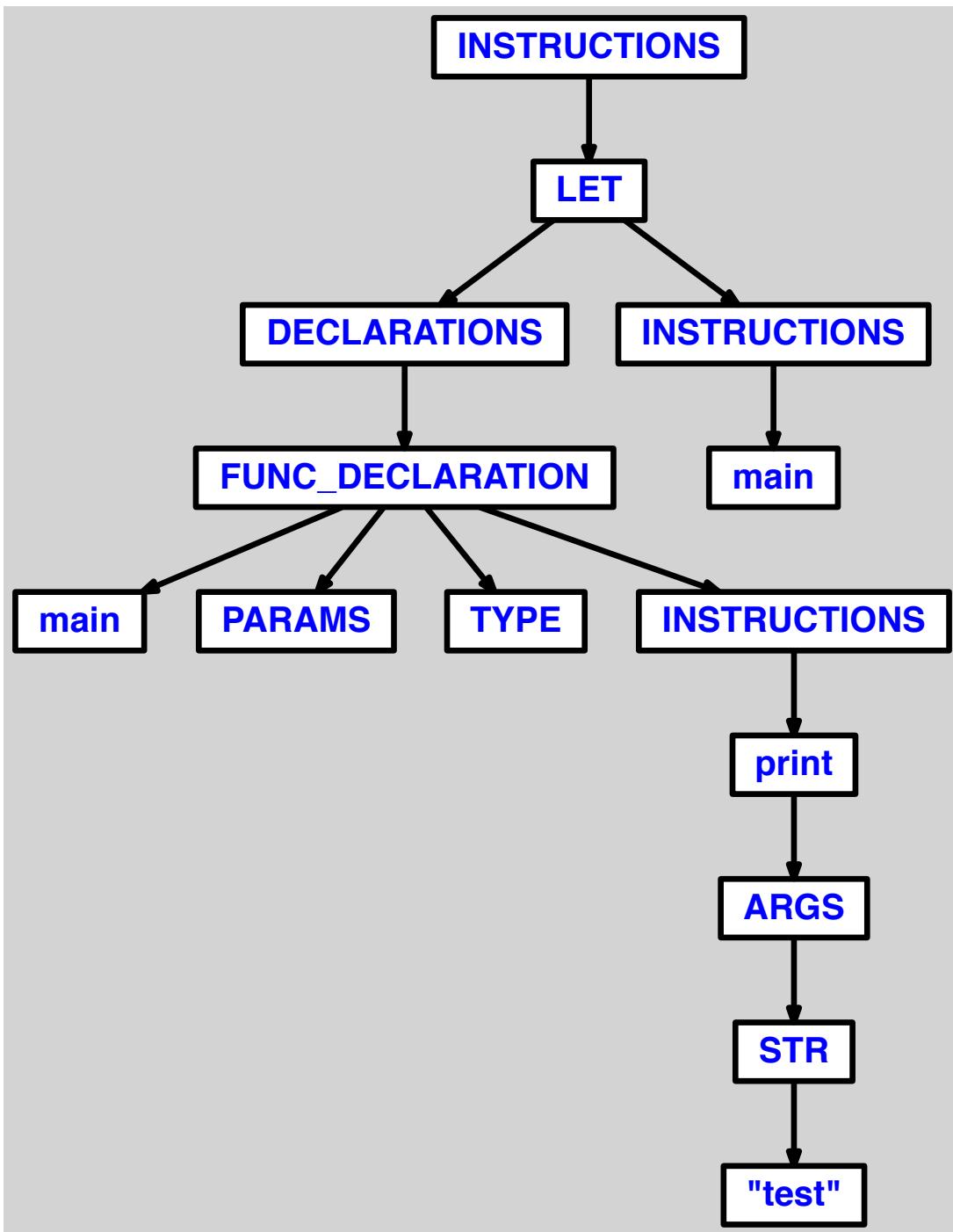
### 8.1.10 < :> pour type de fonction présent mais pas de type

```
1 let
2   function main(): = 0
3 in main() end
```



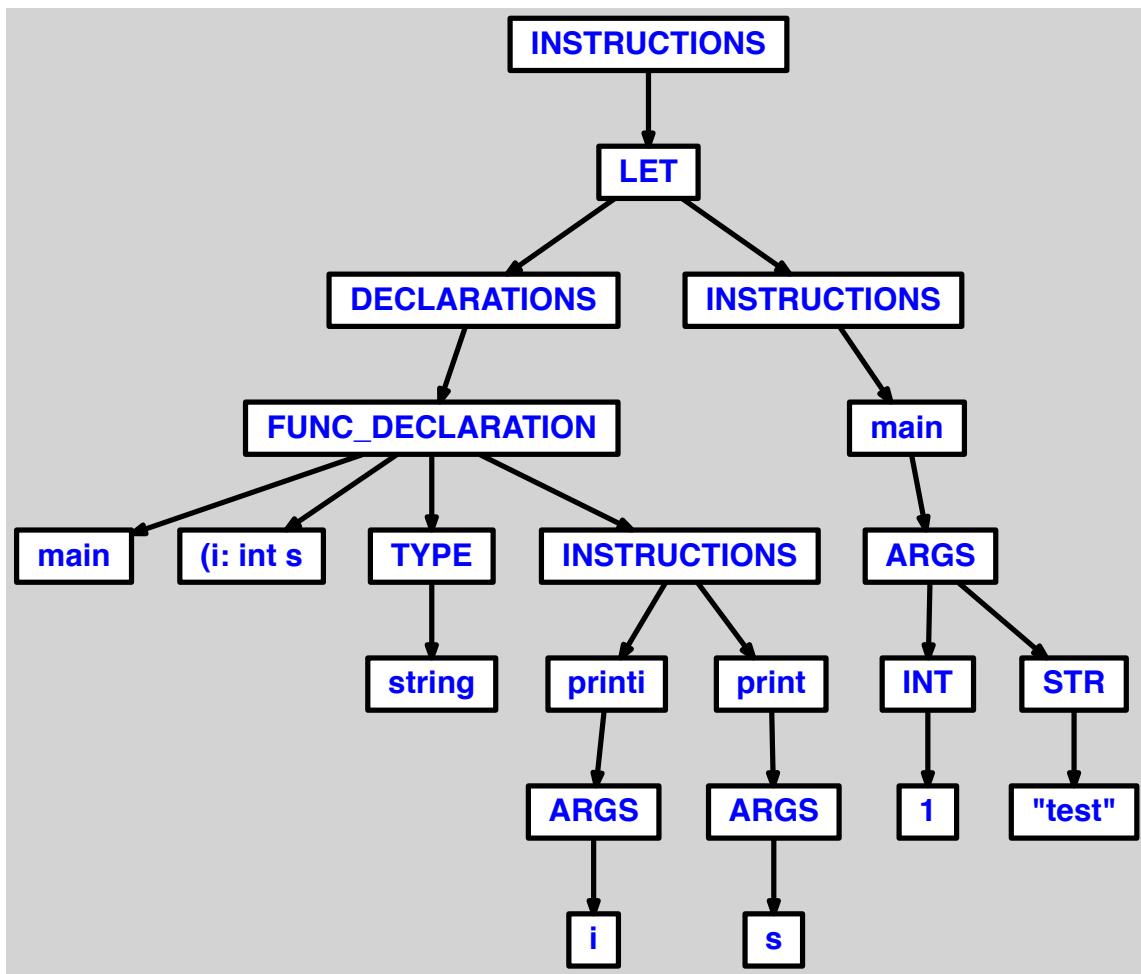
### 8.1.11 oubli de <=>

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



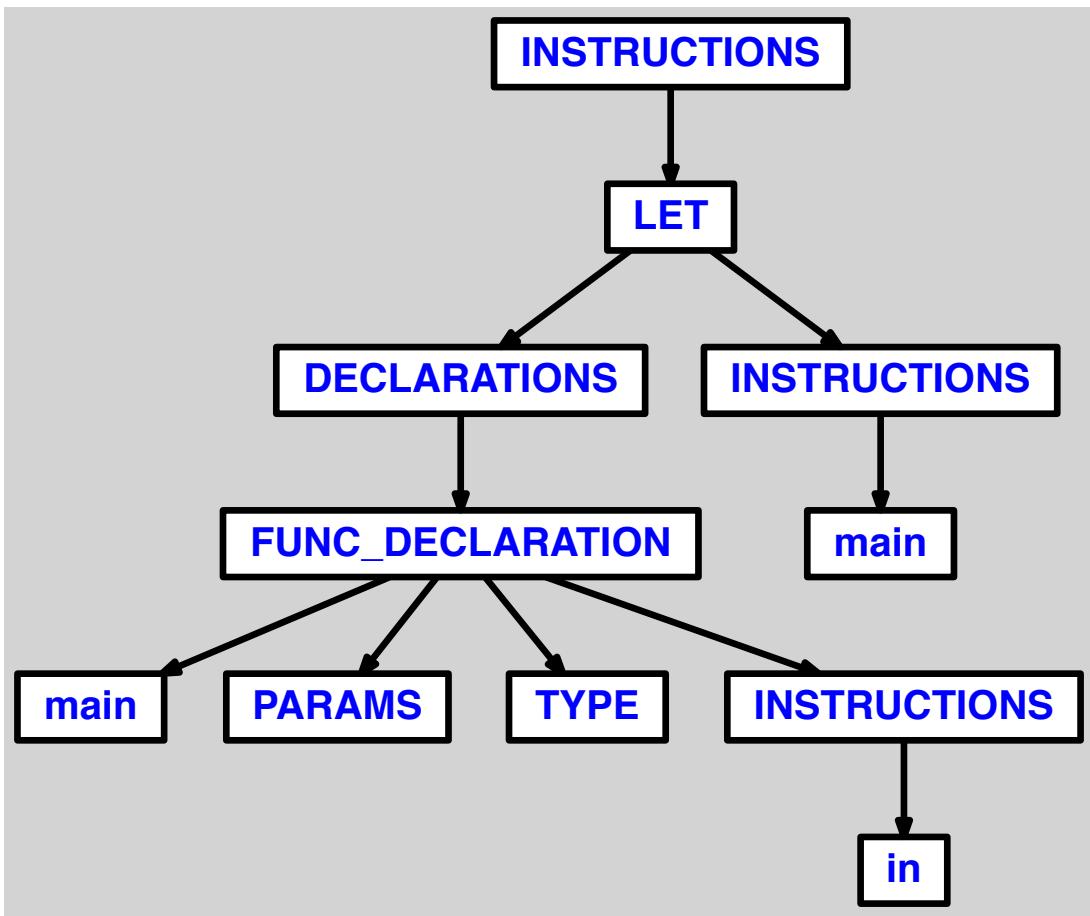
### 8.1.12 parametres mal separees

```
1 let
2   function main(i: int s: string) =
3     (printi(i); print(s))
4   in main(1, "test") end
```



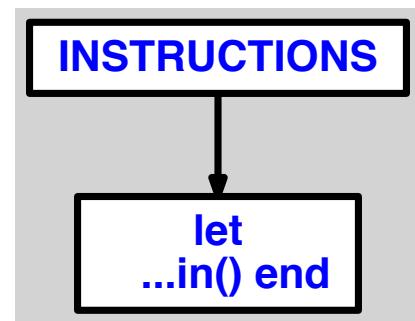
### 8.1.13 fonction sans instruction

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



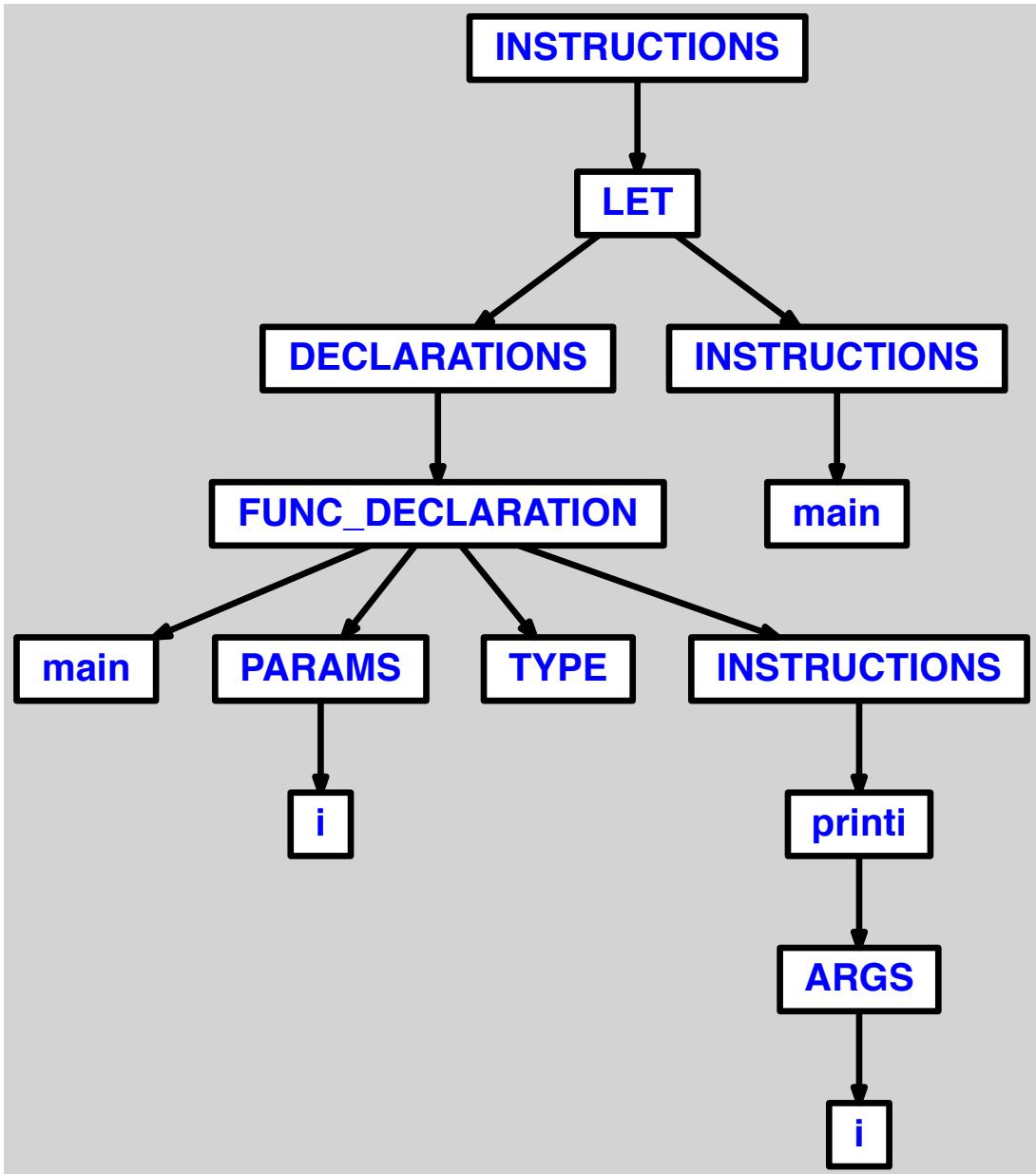
#### 8.1.14 <function> mal écrit

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



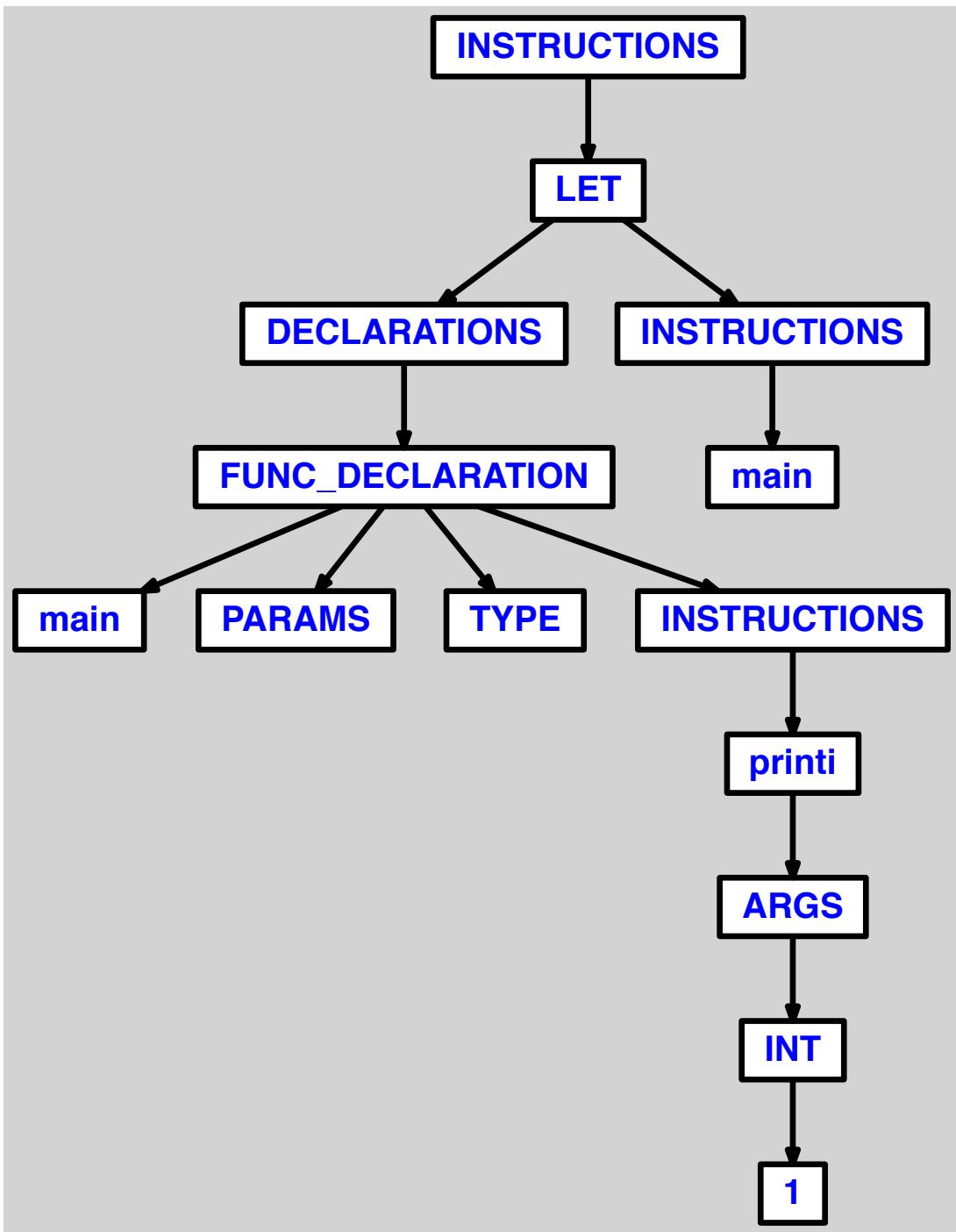
### 8.1.15 fonction avec identifiant seulement en parametre

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



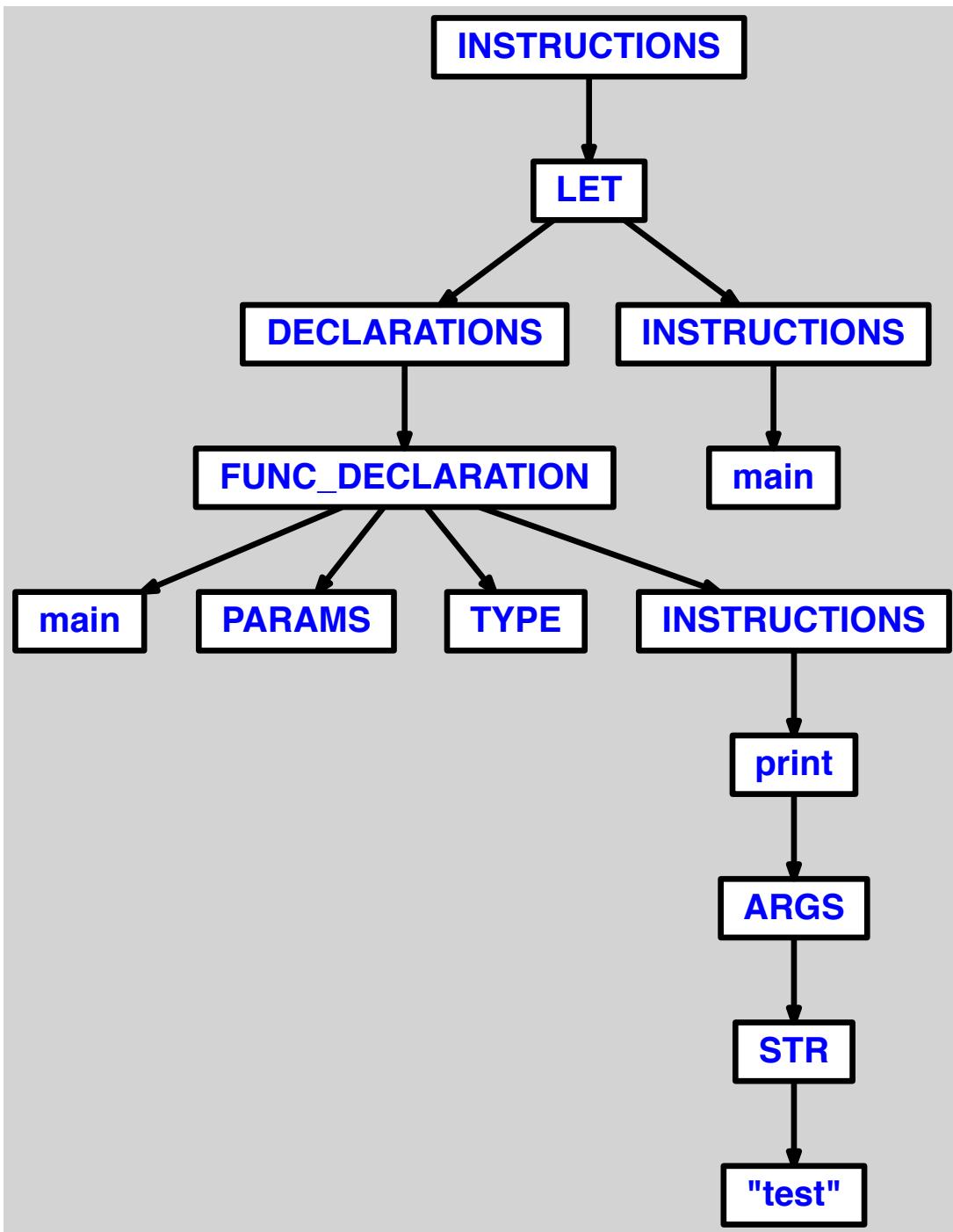
### 8.1.16 fonction avec entier directement en parametre

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



### 8.1.17 fonction avec chaine directement en parametre

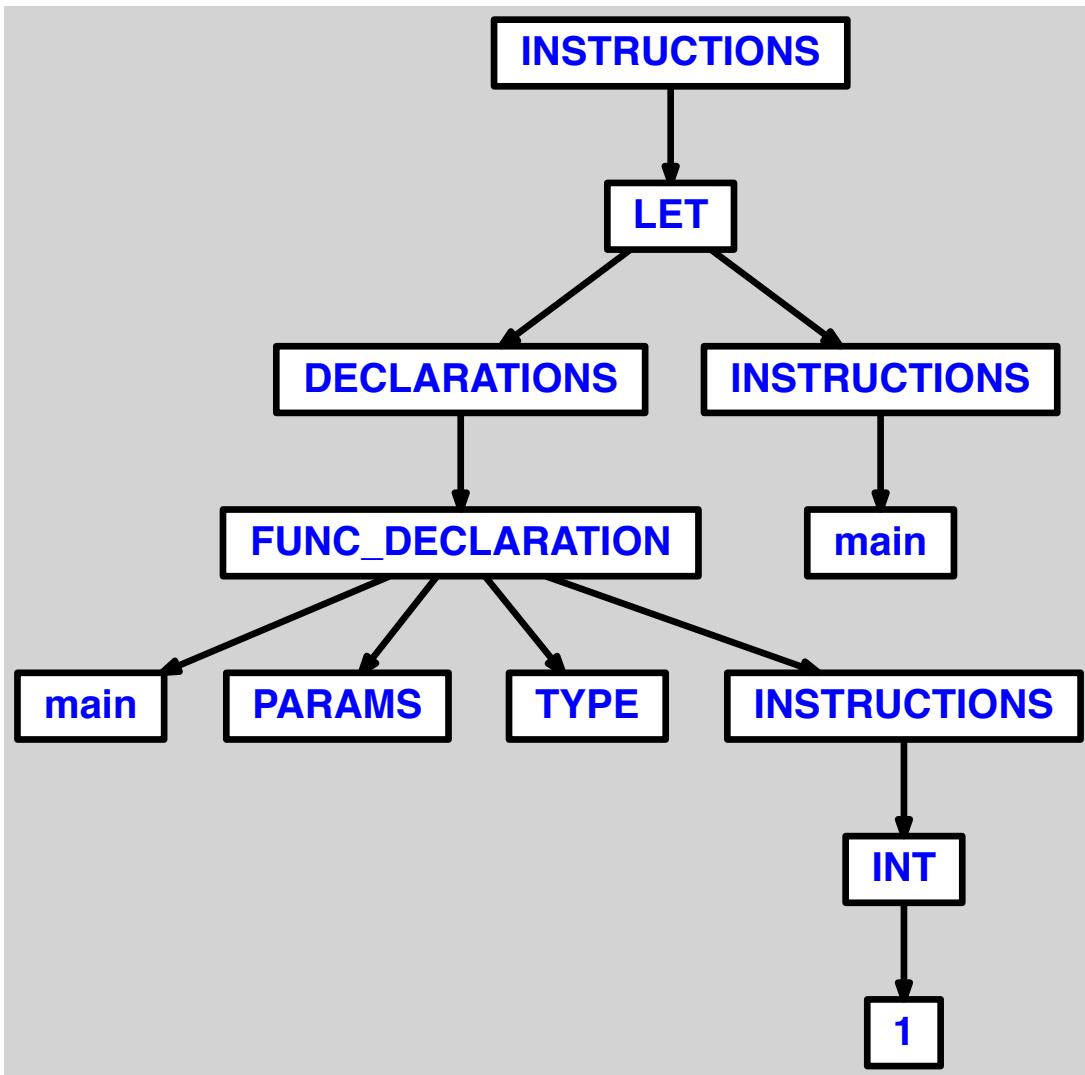
```
1 let
2   function main("test") = print("test")
3 in main() end
```



## 8.2 OK

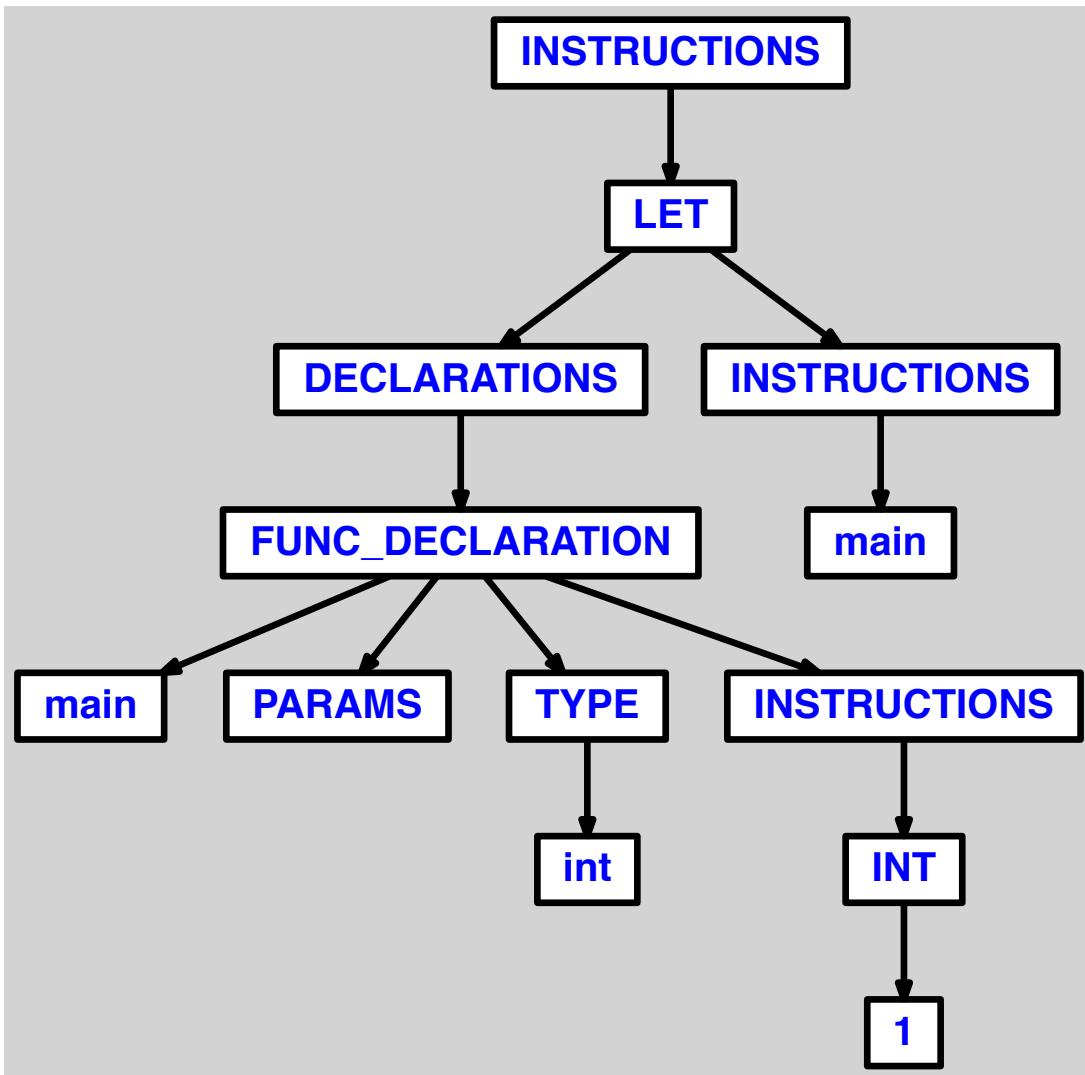
### 8.2.1 fonction definissant un entier

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



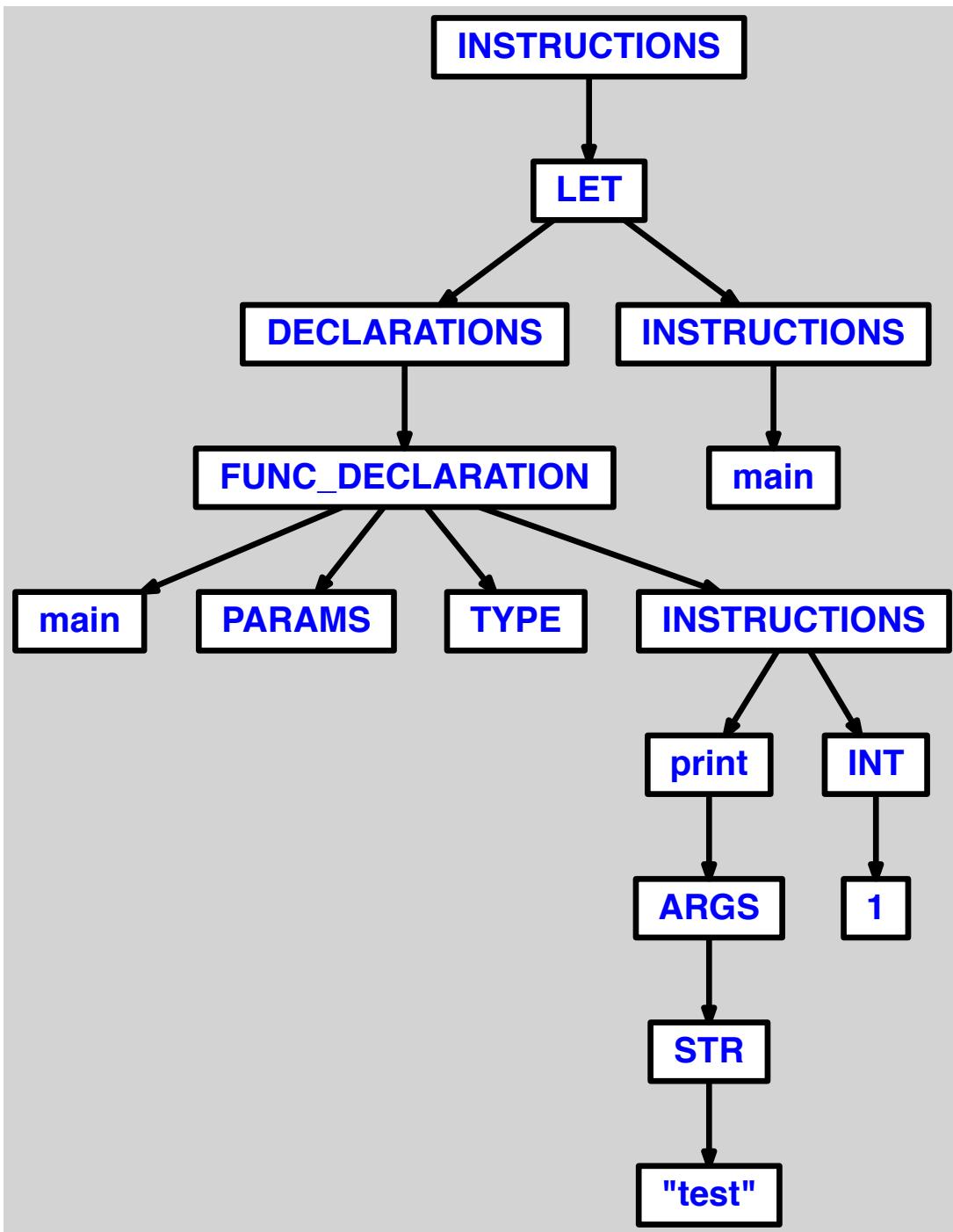
### 8.2.2 fonction simple, avec déclaration du type de retour <int> et instruction de retour

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



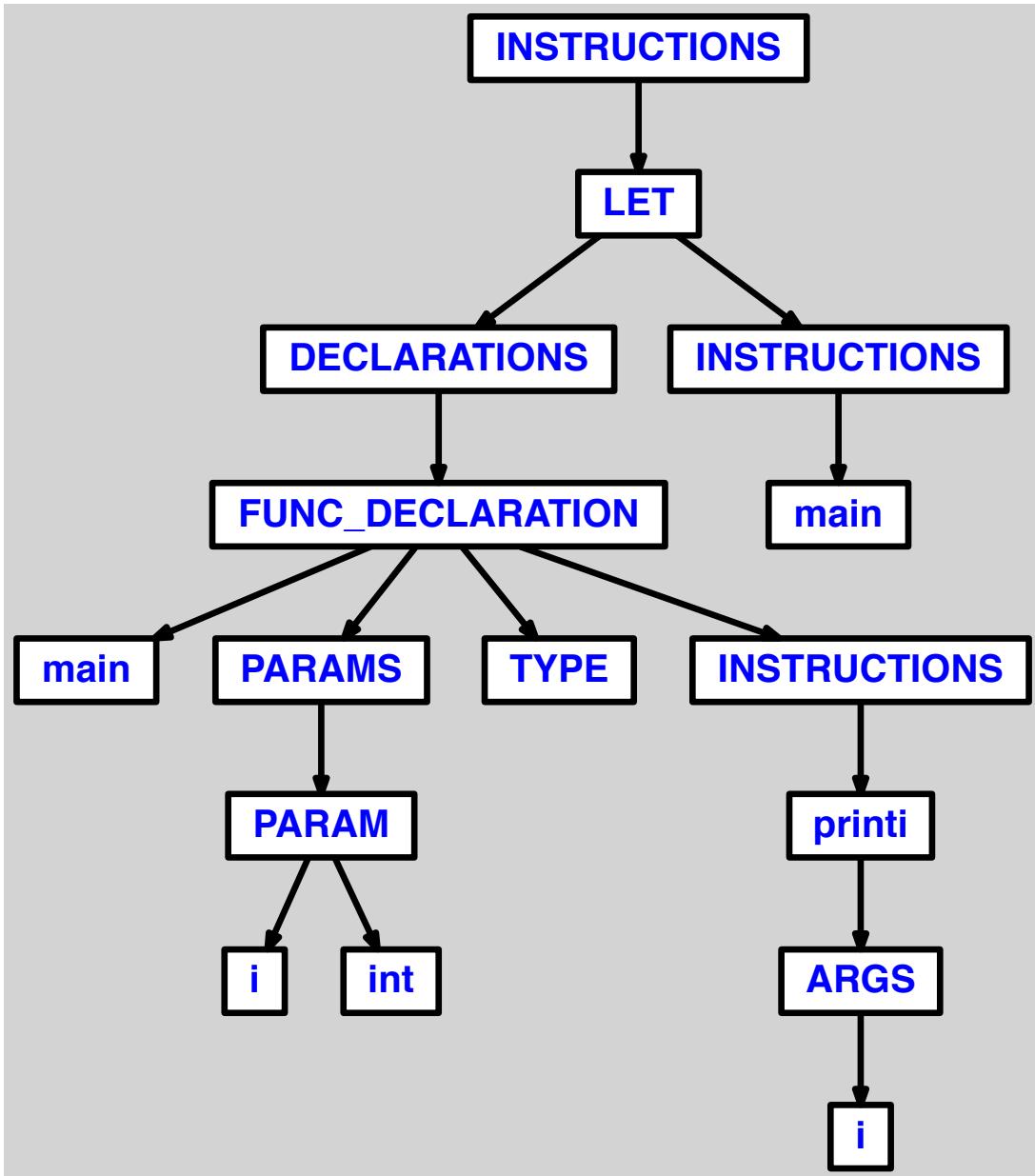
### 8.2.3 fonction simple, avec instruction simple et instruction retournant un entier

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



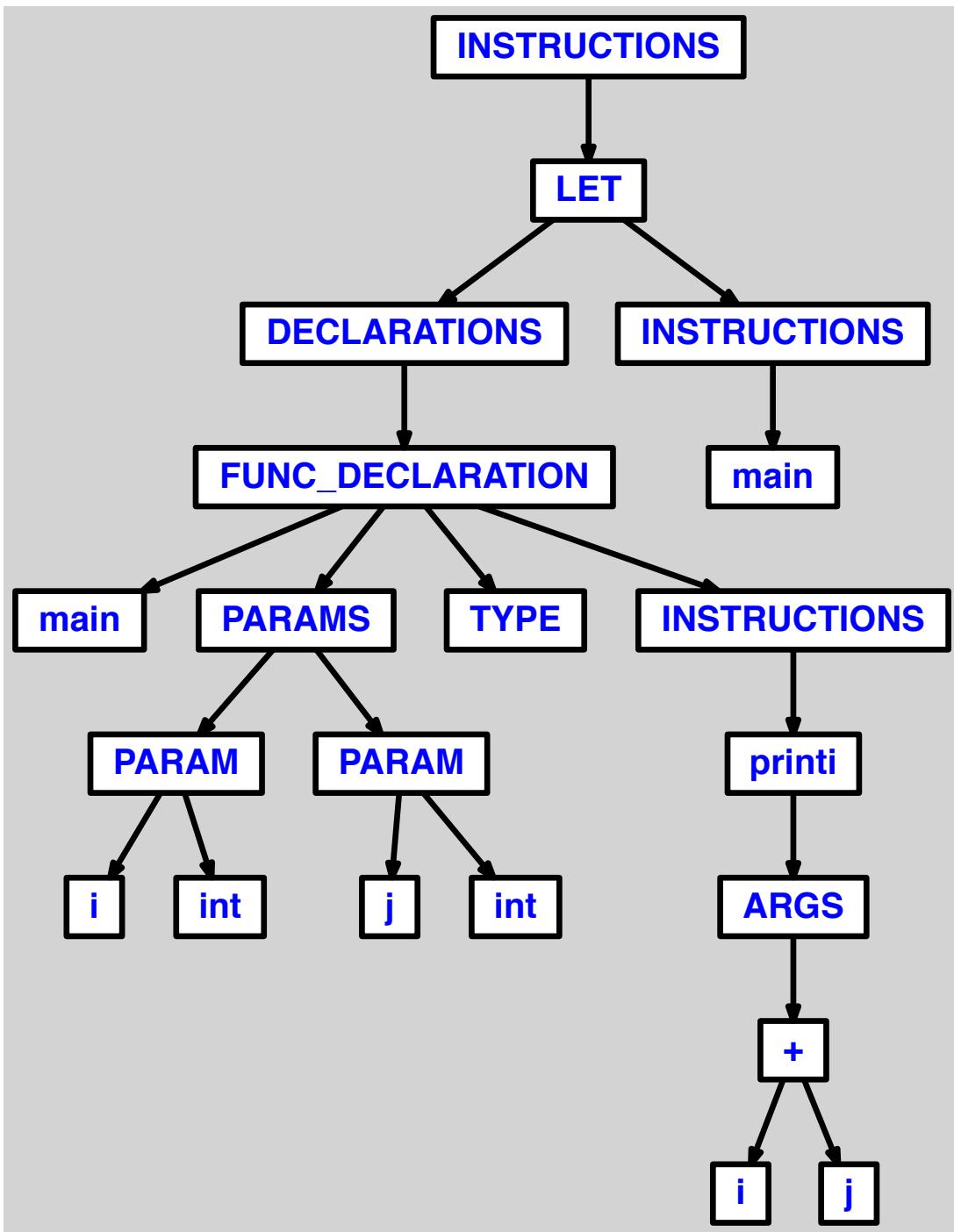
#### 8.2.4 fonction avec 1 parametre entier

```
1 let
2   function main(i: int) = printi(i)
3 in main() end
```



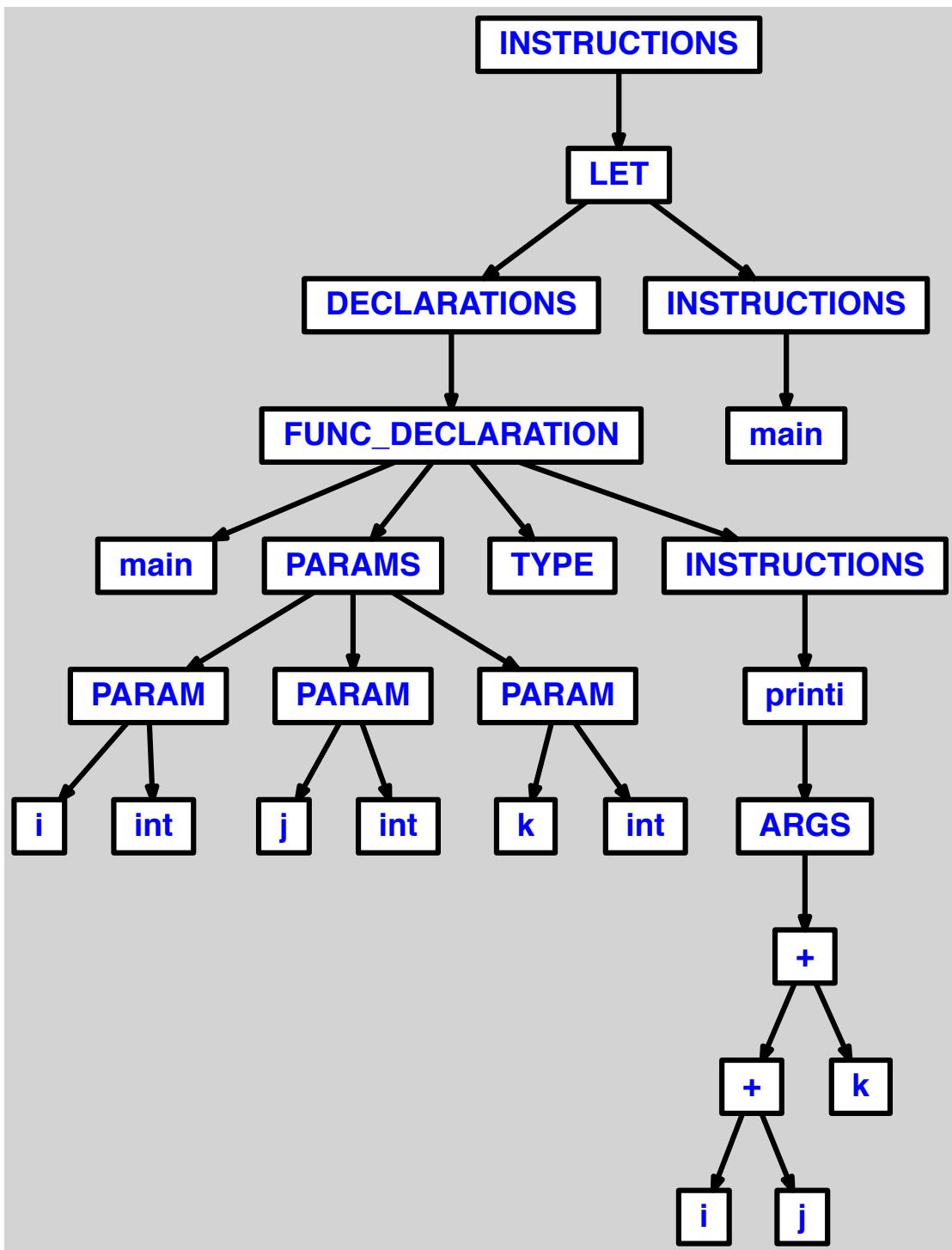
### 8.2.5 fonction avec 2 parametres entiers

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



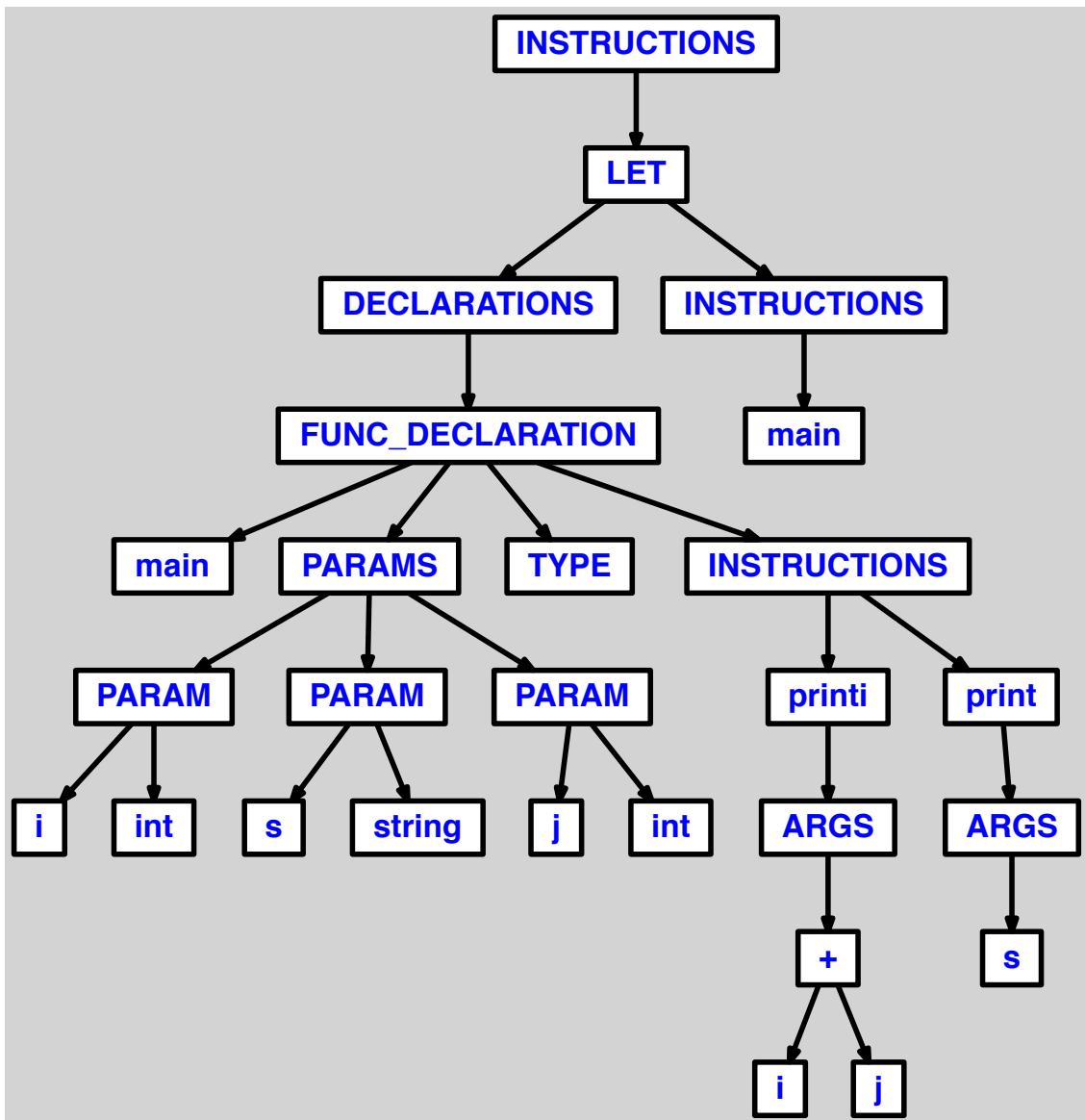
### 8.2.6 fonction avec 3 parametres entiers

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



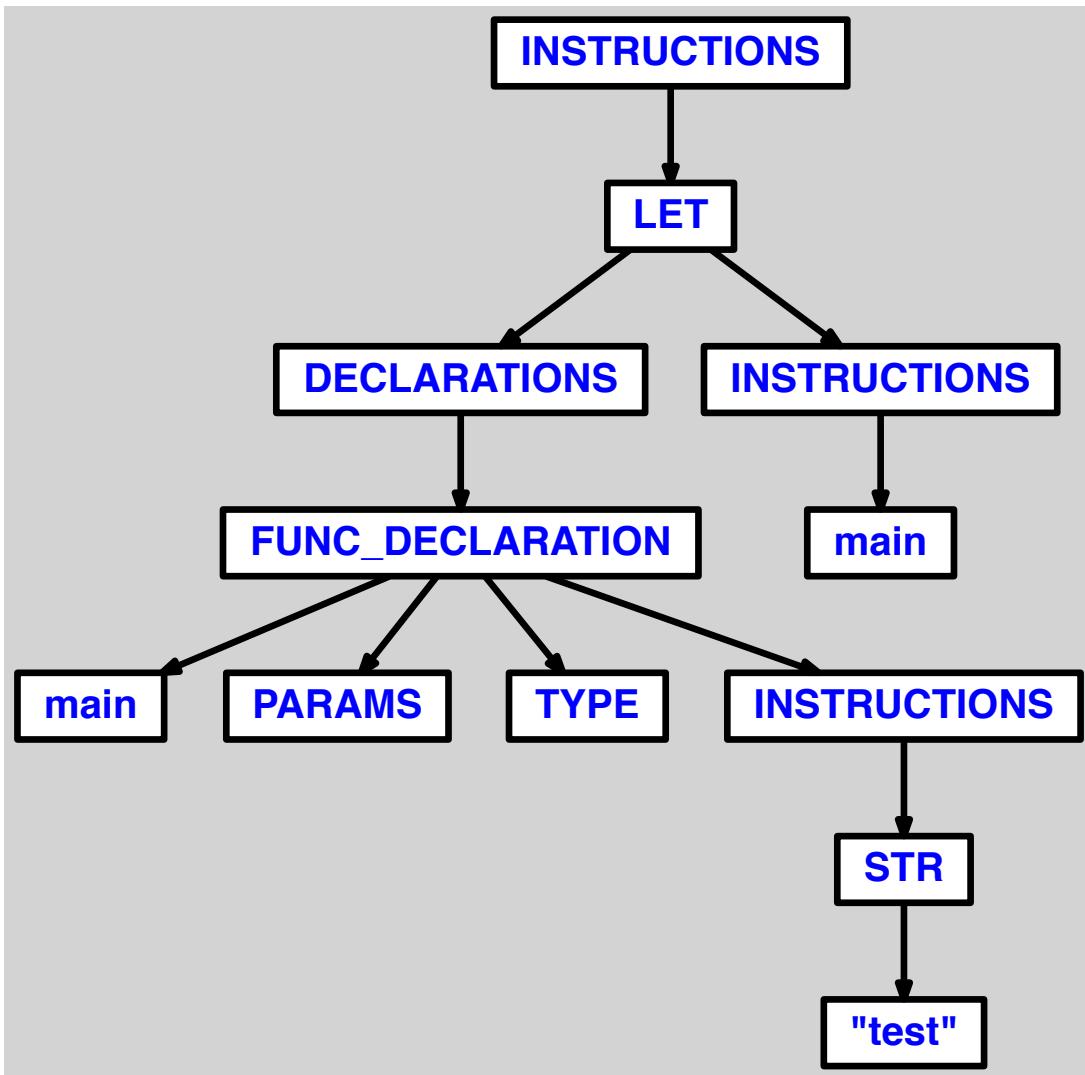
#### 8.2.7 fonction avec 2 parametres entiers et 1 parametre de type <string>

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



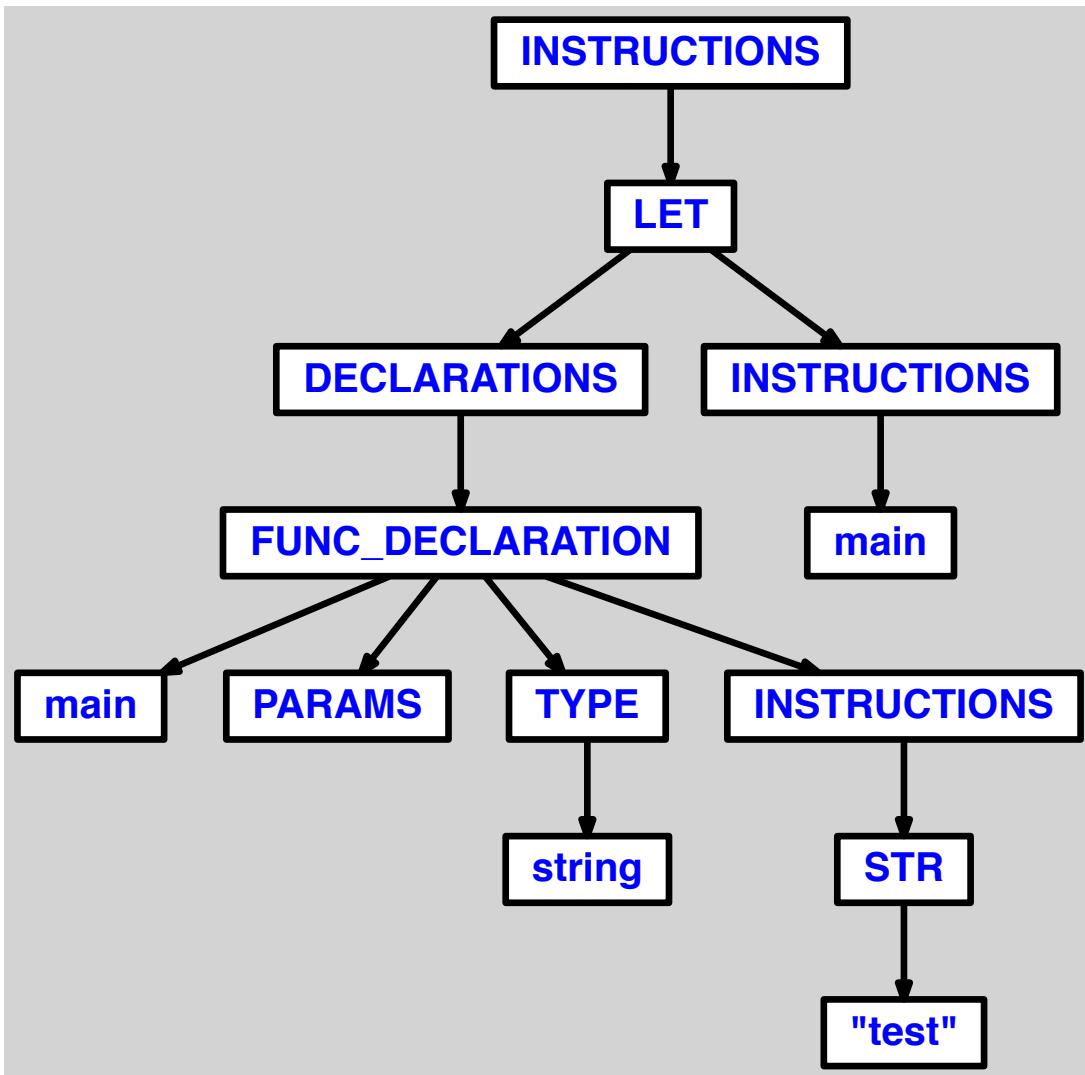
### 8.2.8 fonction definissant une chaine

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



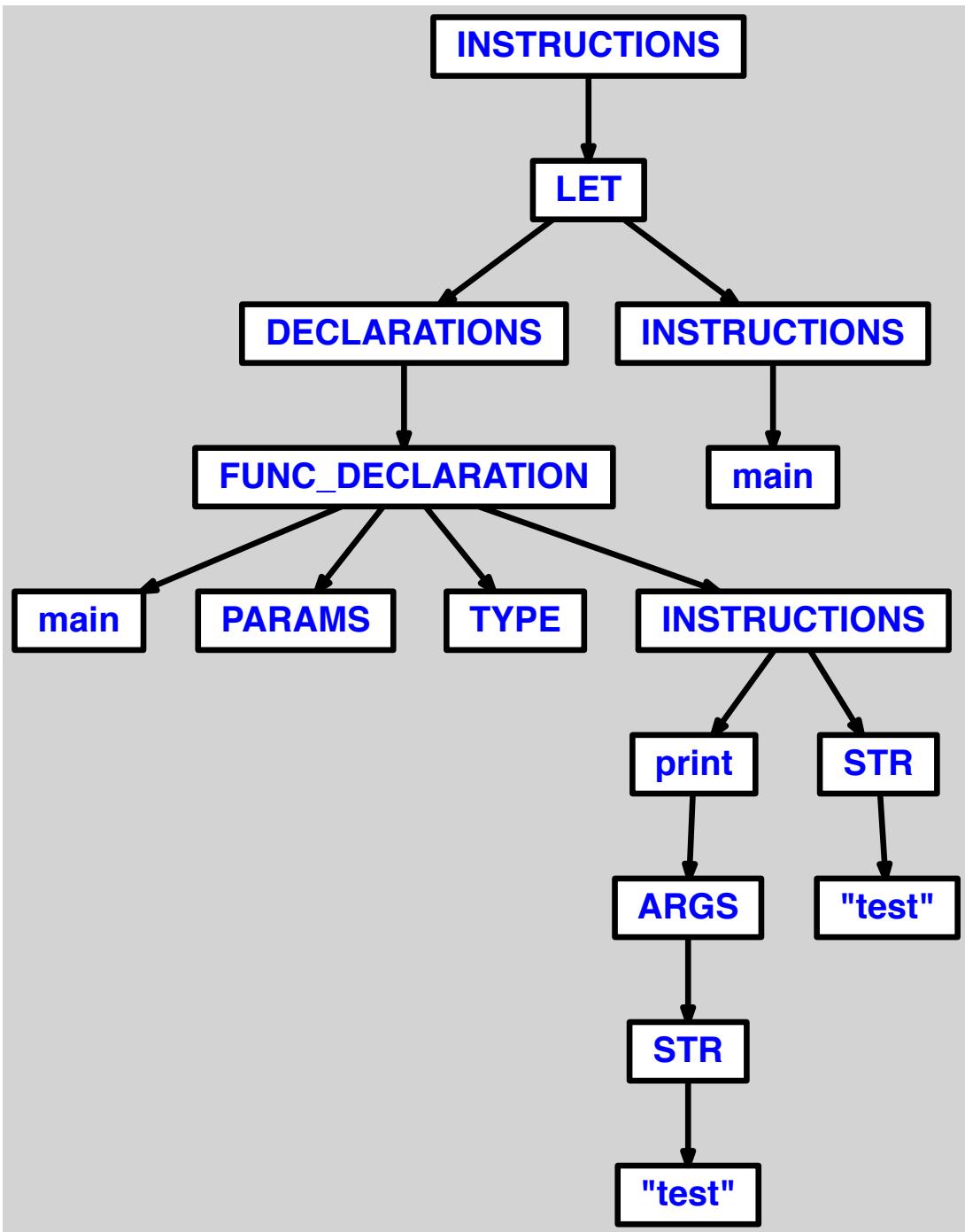
### 8.2.9 fonction simple, avec déclaration du type de retour <string> et instruction de retour

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



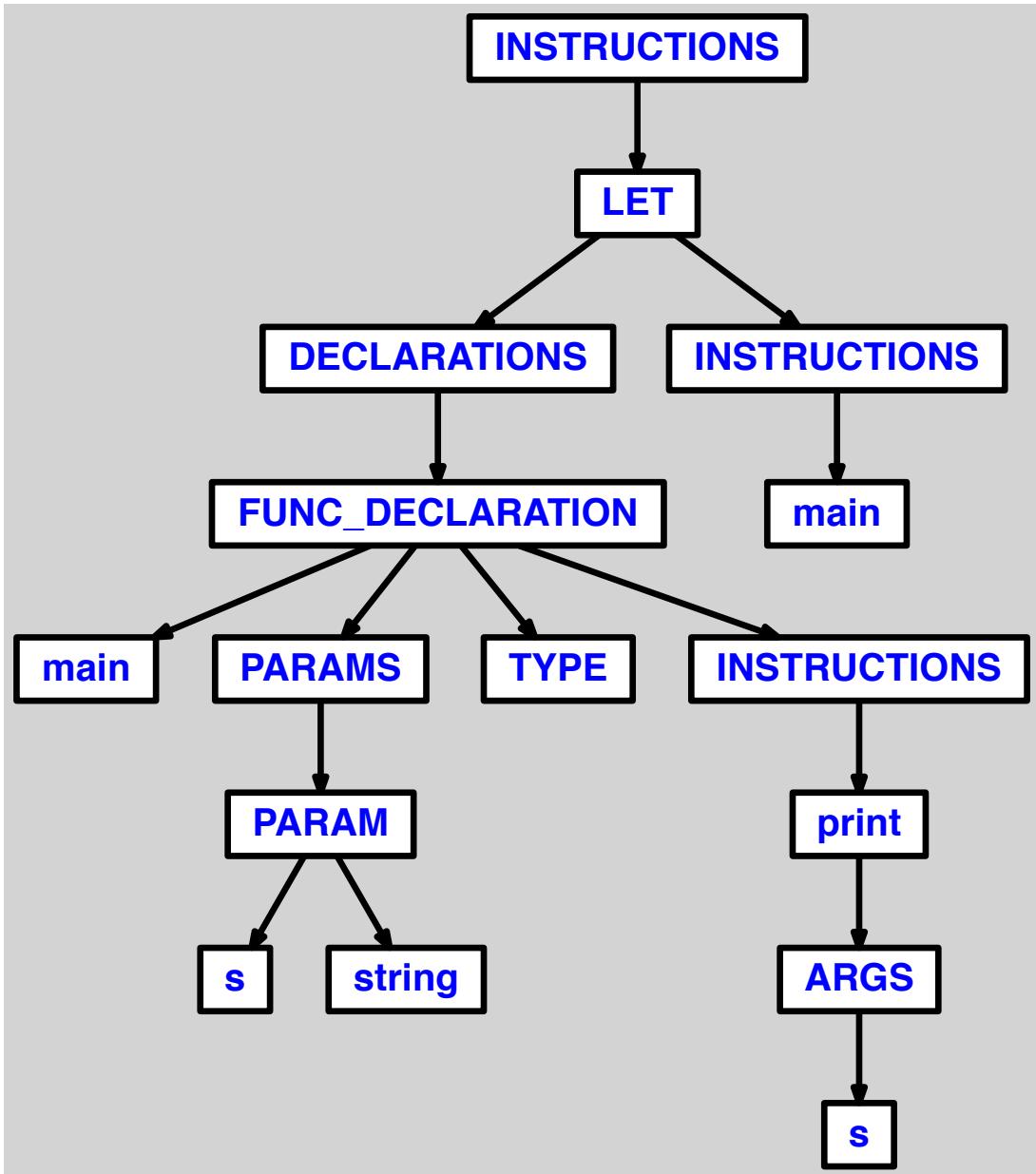
#### 8.2.10 fonction simple, avec instruction simple et instruction retournant une chaîne

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



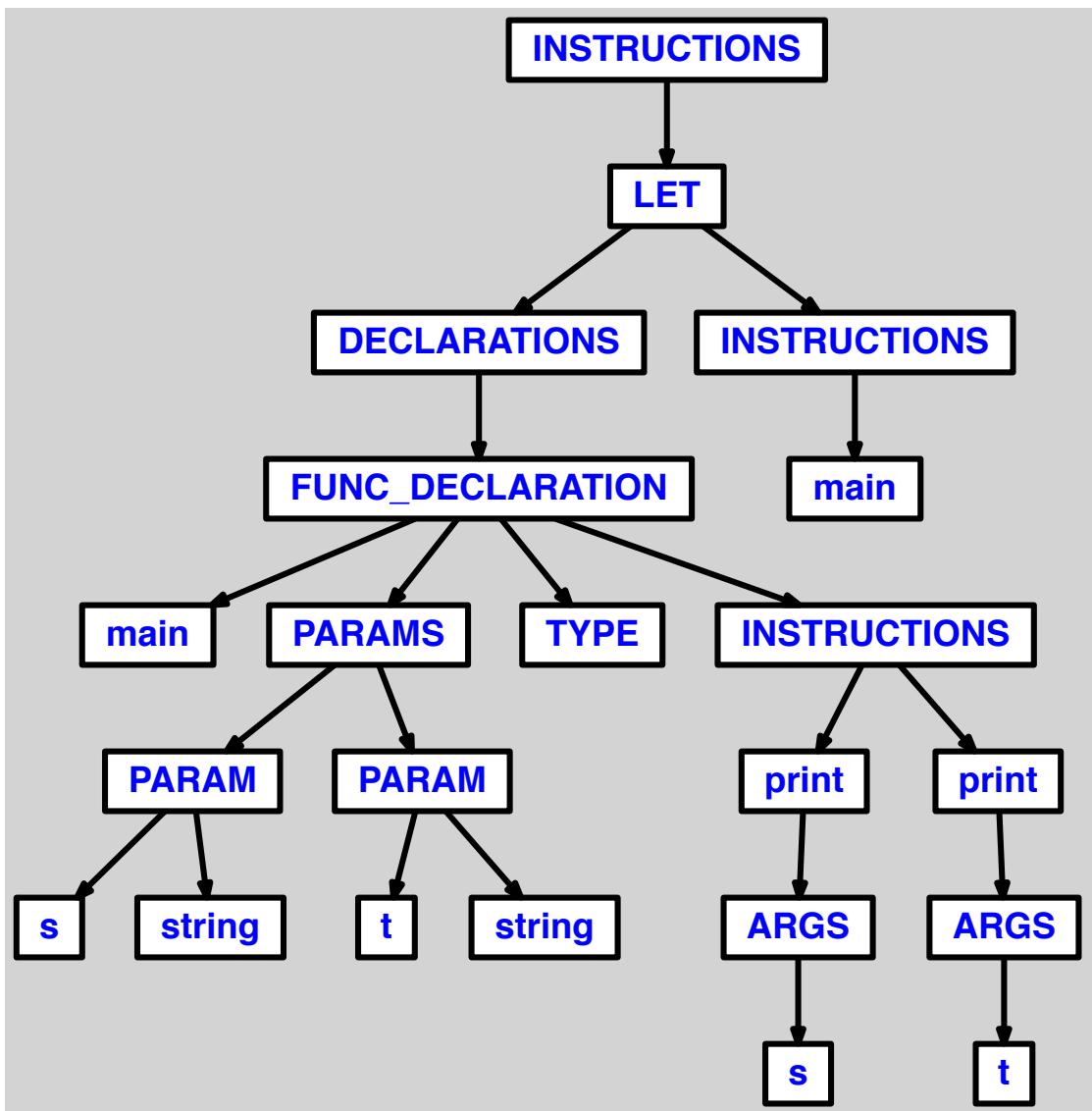
### 8.2.11 fonction avec 1 parametre de type <string>

```
1 let
2   function main(s: string) = print(s)
3 in main() end
```



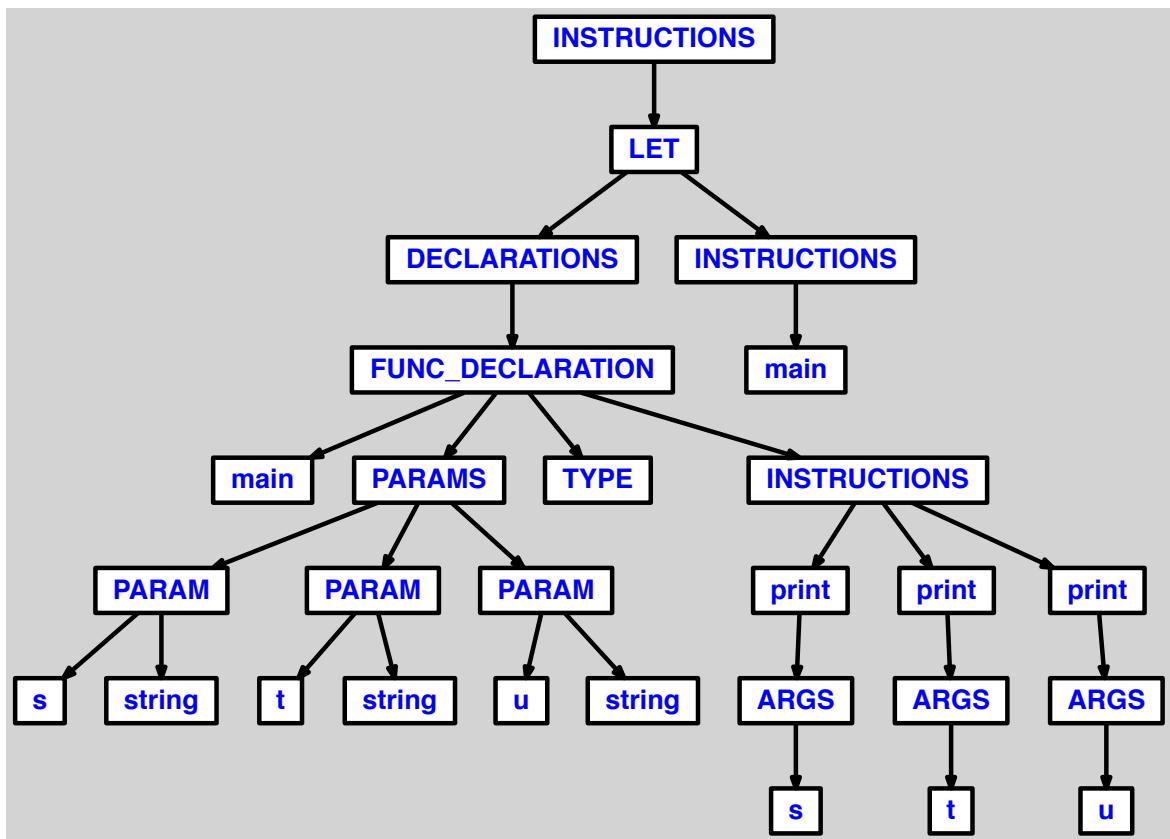
#### 8.2.12 fonction avec 2 parametres de type <string>

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



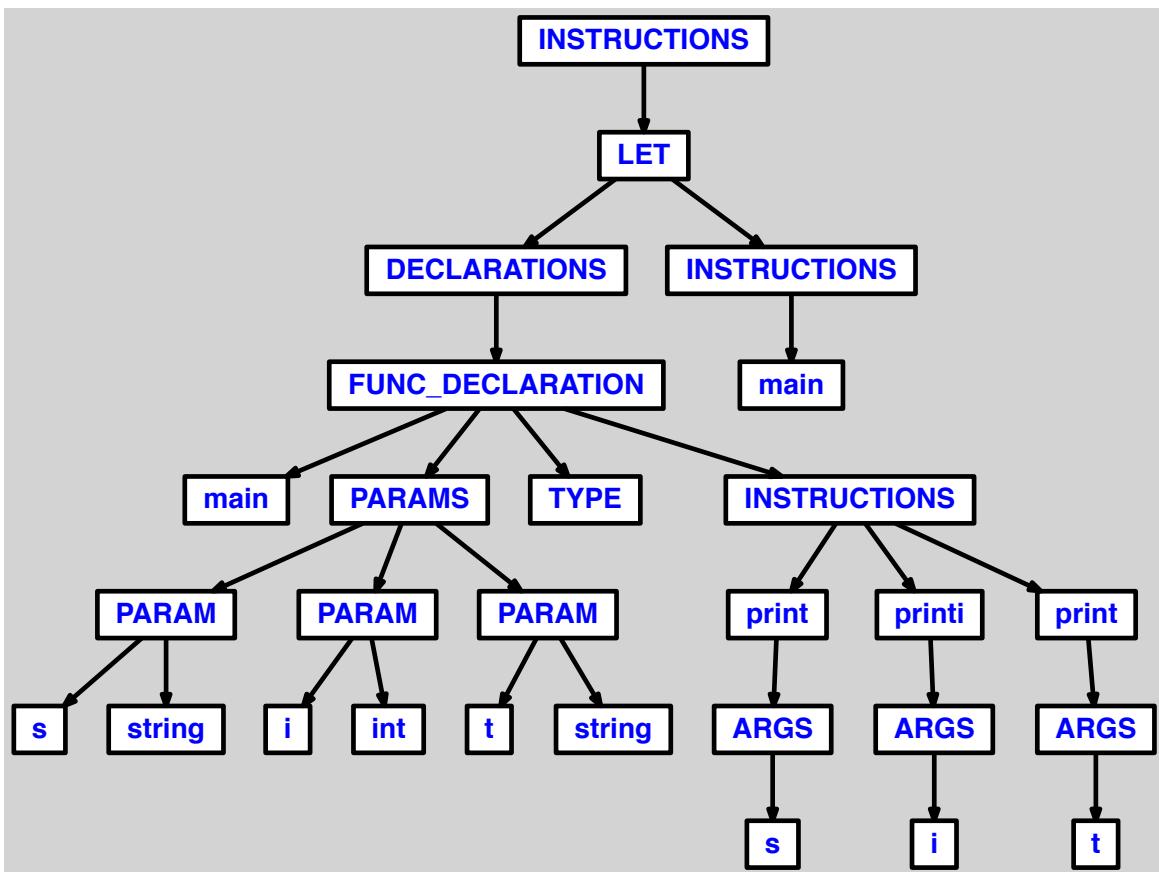
### 8.2.13 fonction avec 3 paramètres de type <string>

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



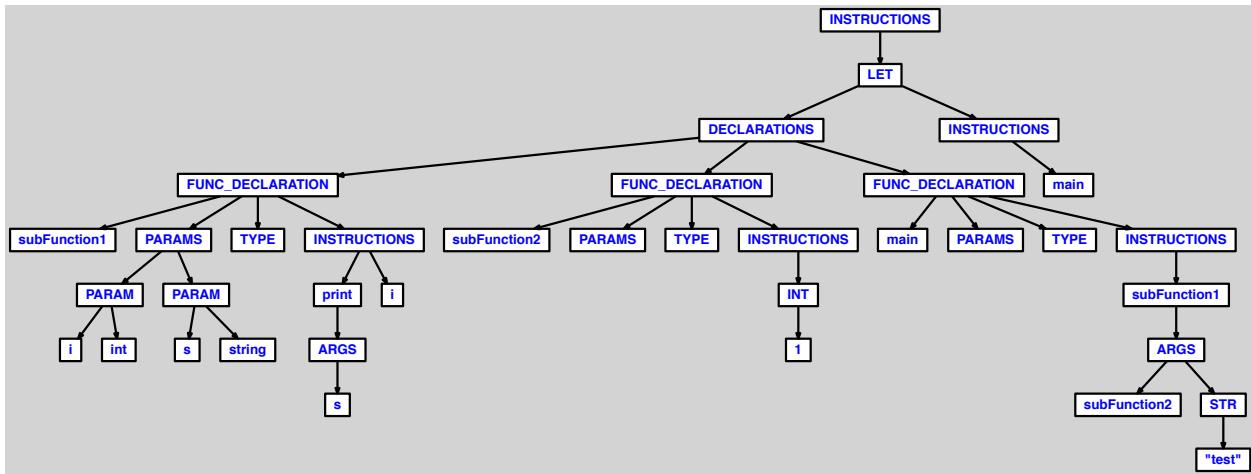
#### 8.2.14 fonction avec 2 parametres de type <string> et 1 parametre entier

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



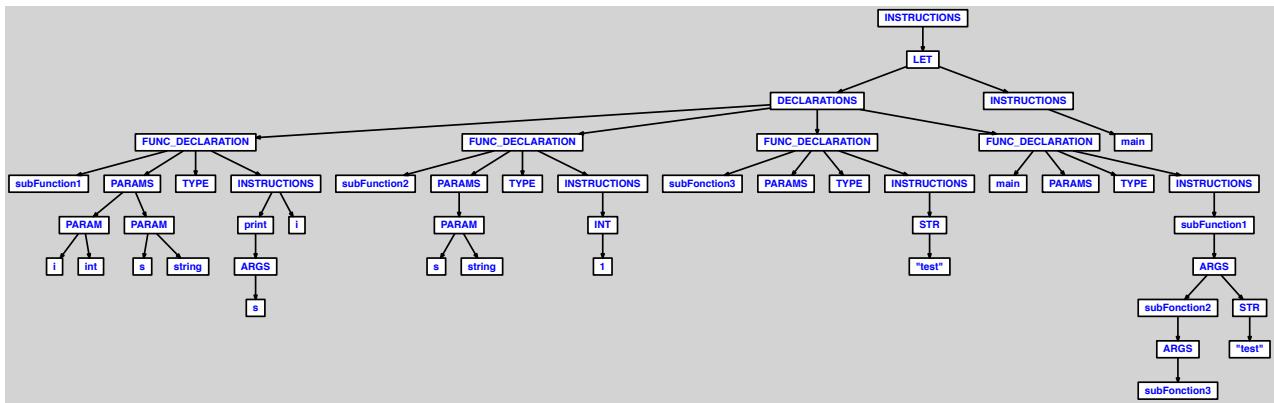
### 8.2.15 double-imbrication de fonction

```
1 let
2     function subFunction1(i: int, s: string) =
3         (print(s); i)
4
5     function subFunction2() = 1
6
7     function main() = subFunction1(subFunction2(), "test")
8 in main() end
```



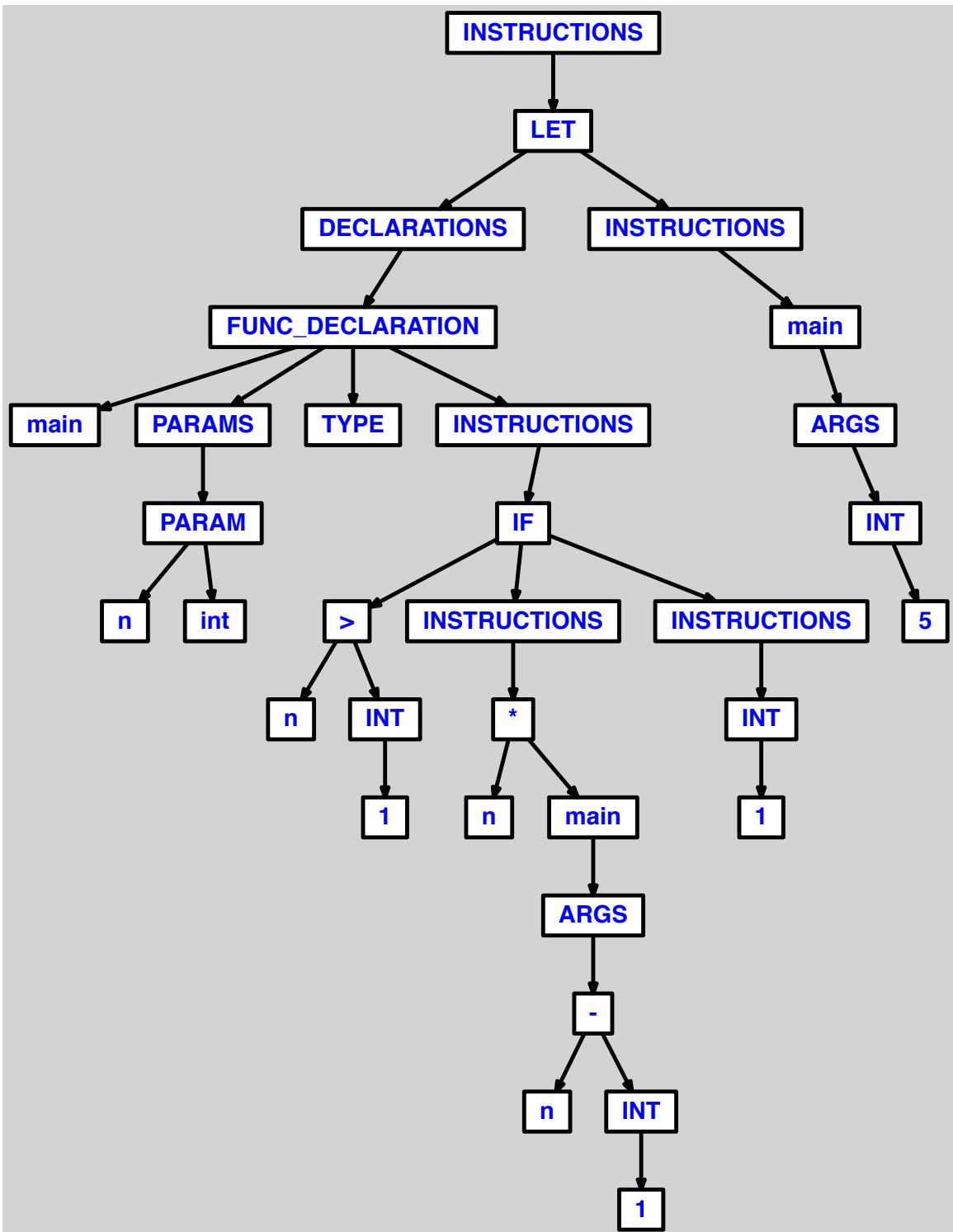
### 8.2.16 triple-imbrication de fonction

```
1 let
2   function subFunction1(i: int, s: string) =
3     (print(s); i)
4
5   function subFunction2(s: string) = 1
6
7   function subFonction3() = "test"
8
9   function main() = subFunction1(subFonction2(subFonction3()), "test")
10 in main() end
```



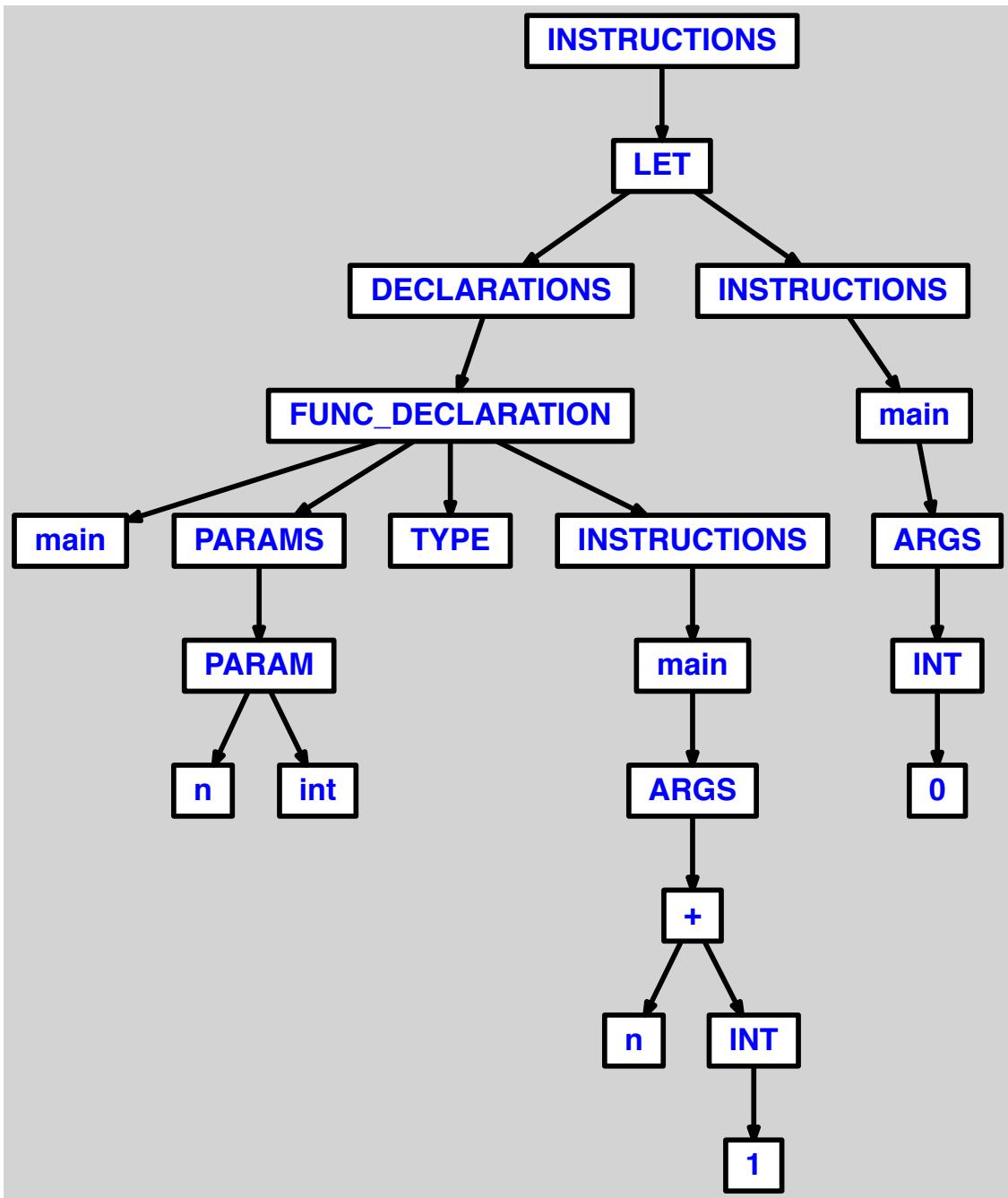
### 8.2.17 recursion finie

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



### 8.2.18 recursion infinie

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

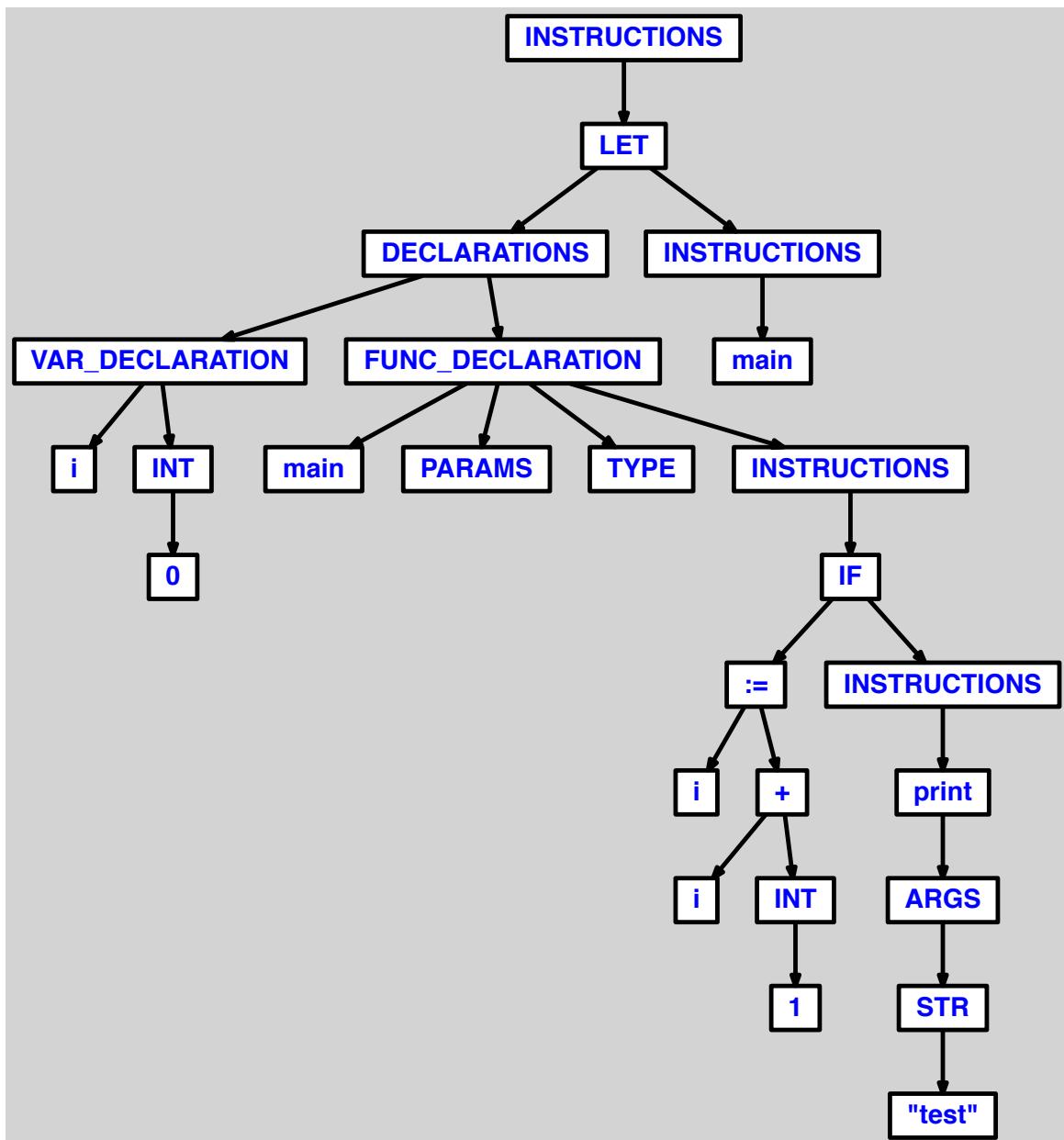


## 9 if

### 9.1 KO

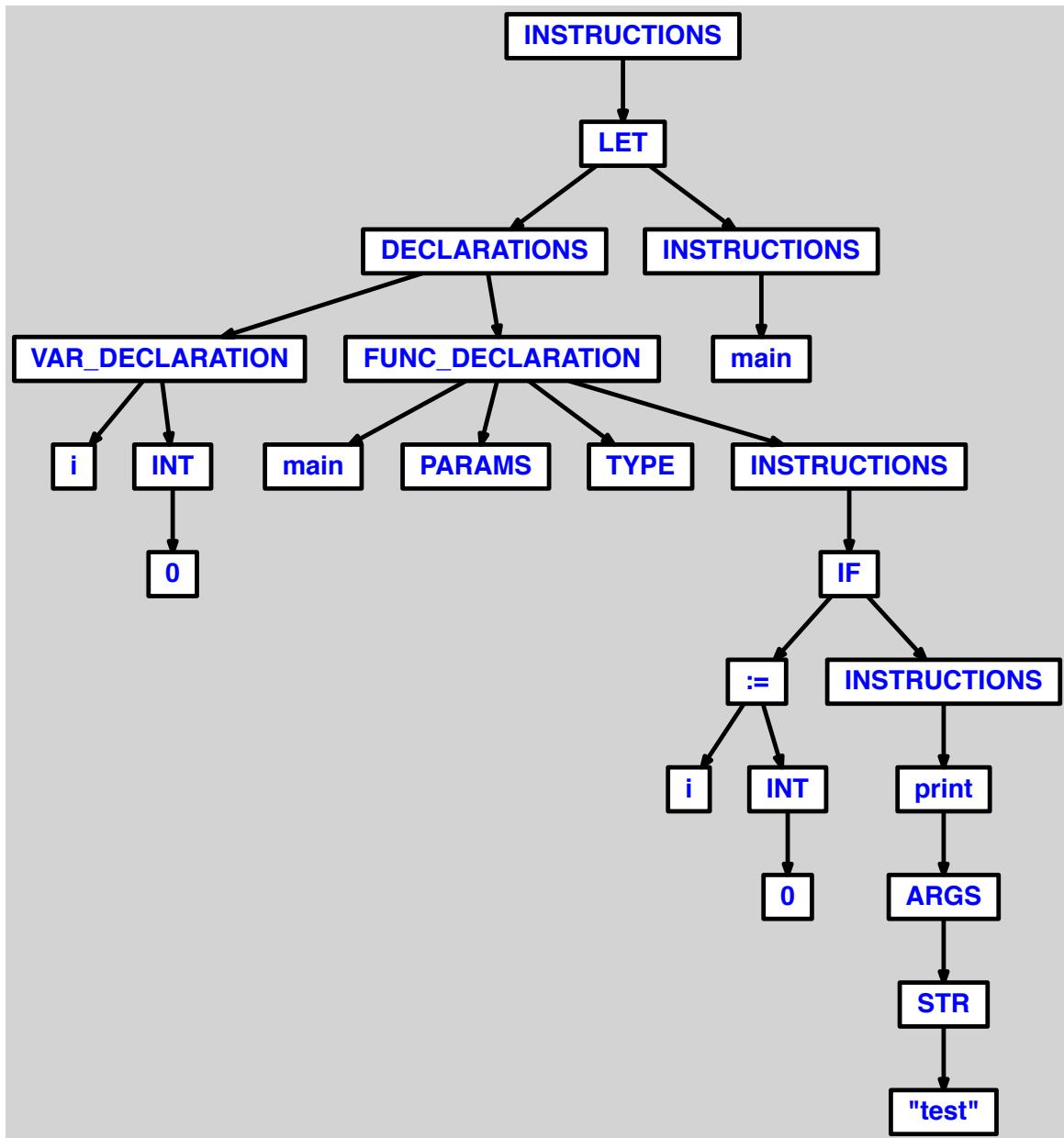
#### 9.1.1 incrementation dans la condition <if then>

```
1 let
2   var i := 0
3
4   function main() =
5     if i := i+1 then
6       print("test")
7   in main() end
```



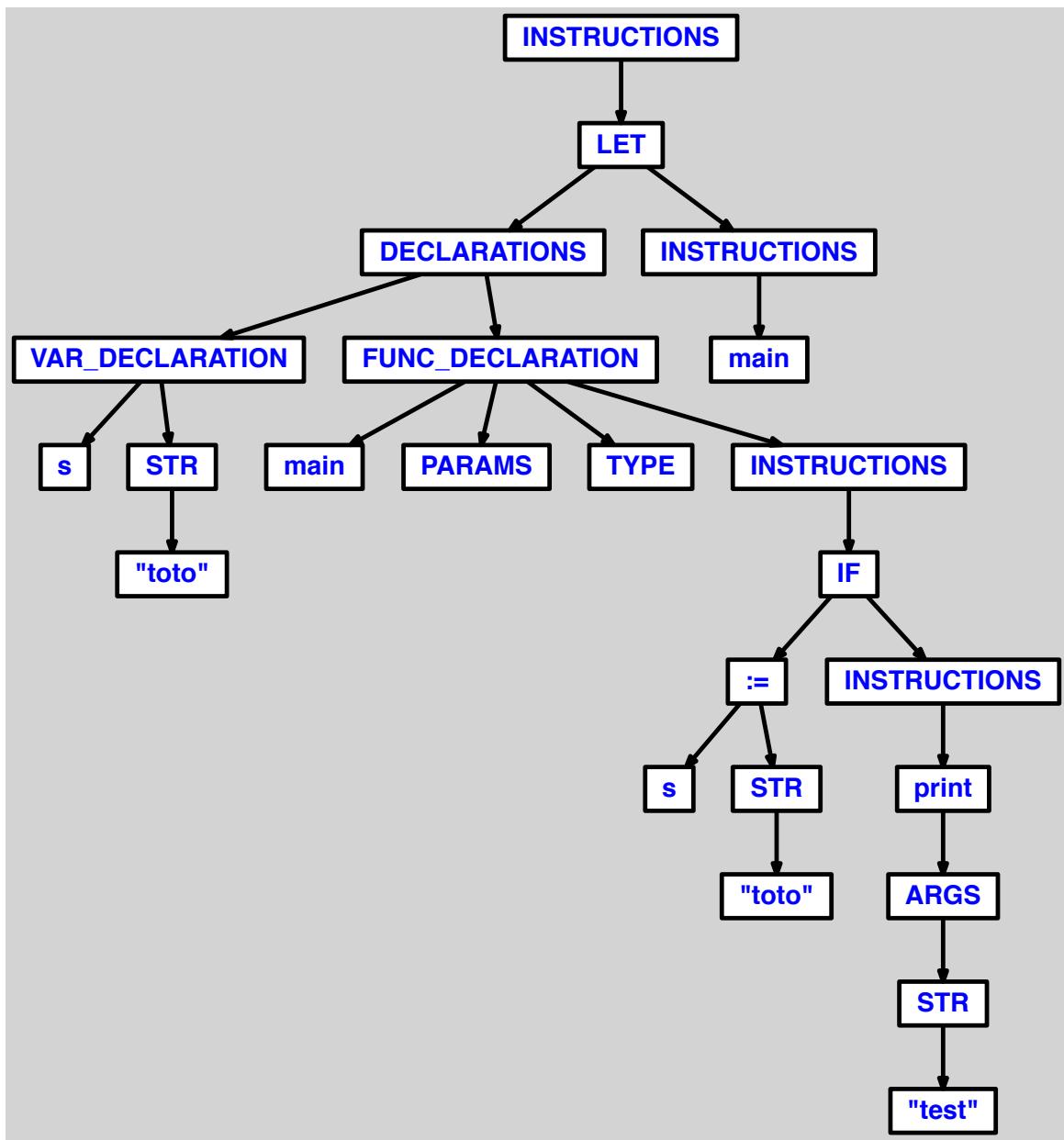
### 9.1.2 affectation d'entier dans la condition <if then>

```
1 let
2   var i := 0
3
4   function main() =
5     if i := 0 then
6       print("test")
7 in main() end
```



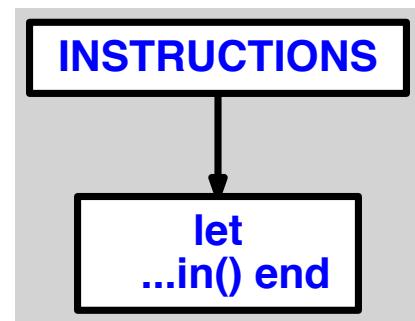
### 9.1.3 affectation de chaine dans la condition <if then>

```
1 let
2   var s := "toto"
3
4   function main() =
5     if s := "toto" then
6       print("test")
7   in main() end
```



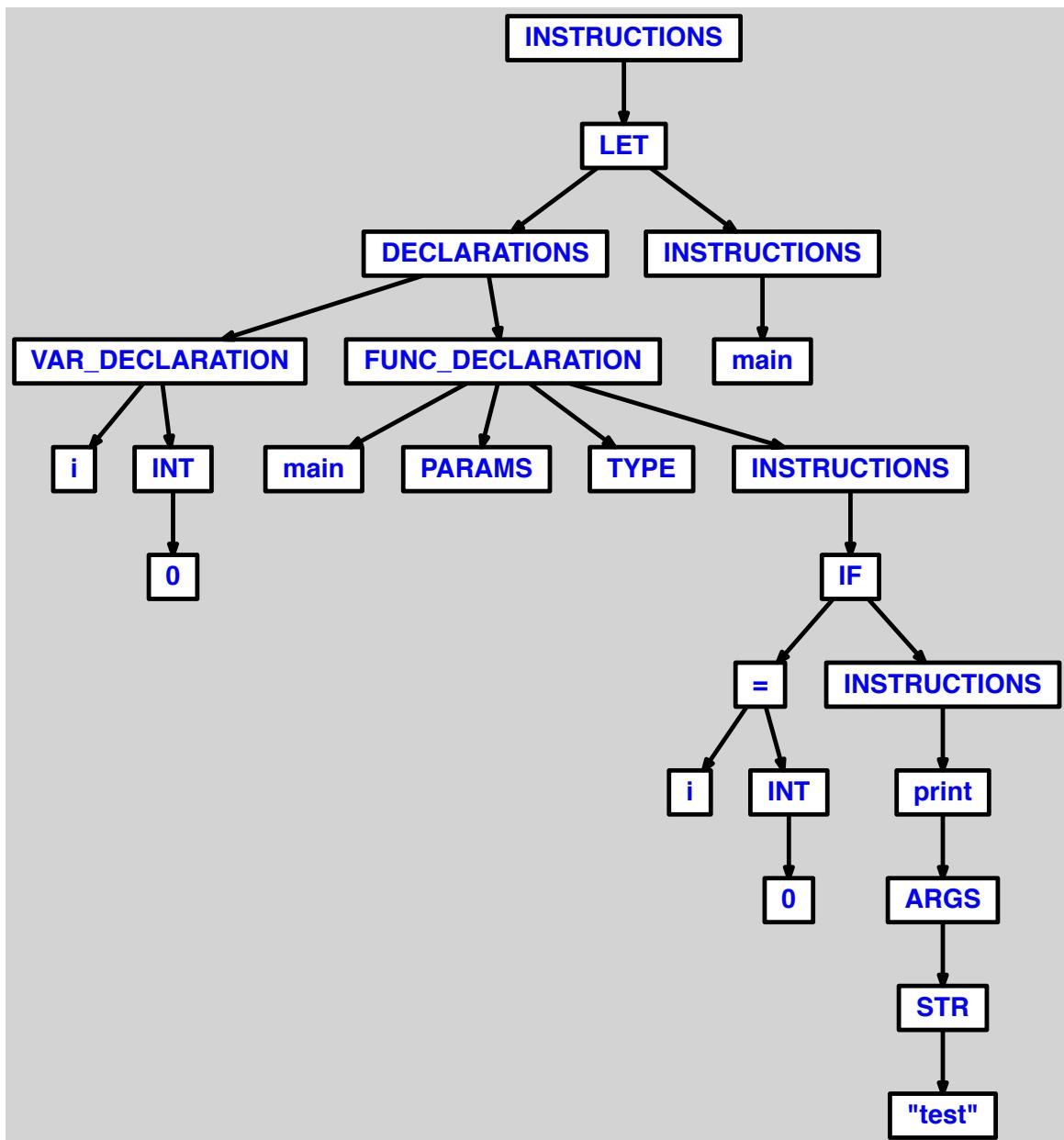
#### 9.1.4 <if then> avec oubli du <if>

```
1 let
2   var i := 0
3
4   function main() =
5     i = 0 then
6       print("test")
7 in main() end
```



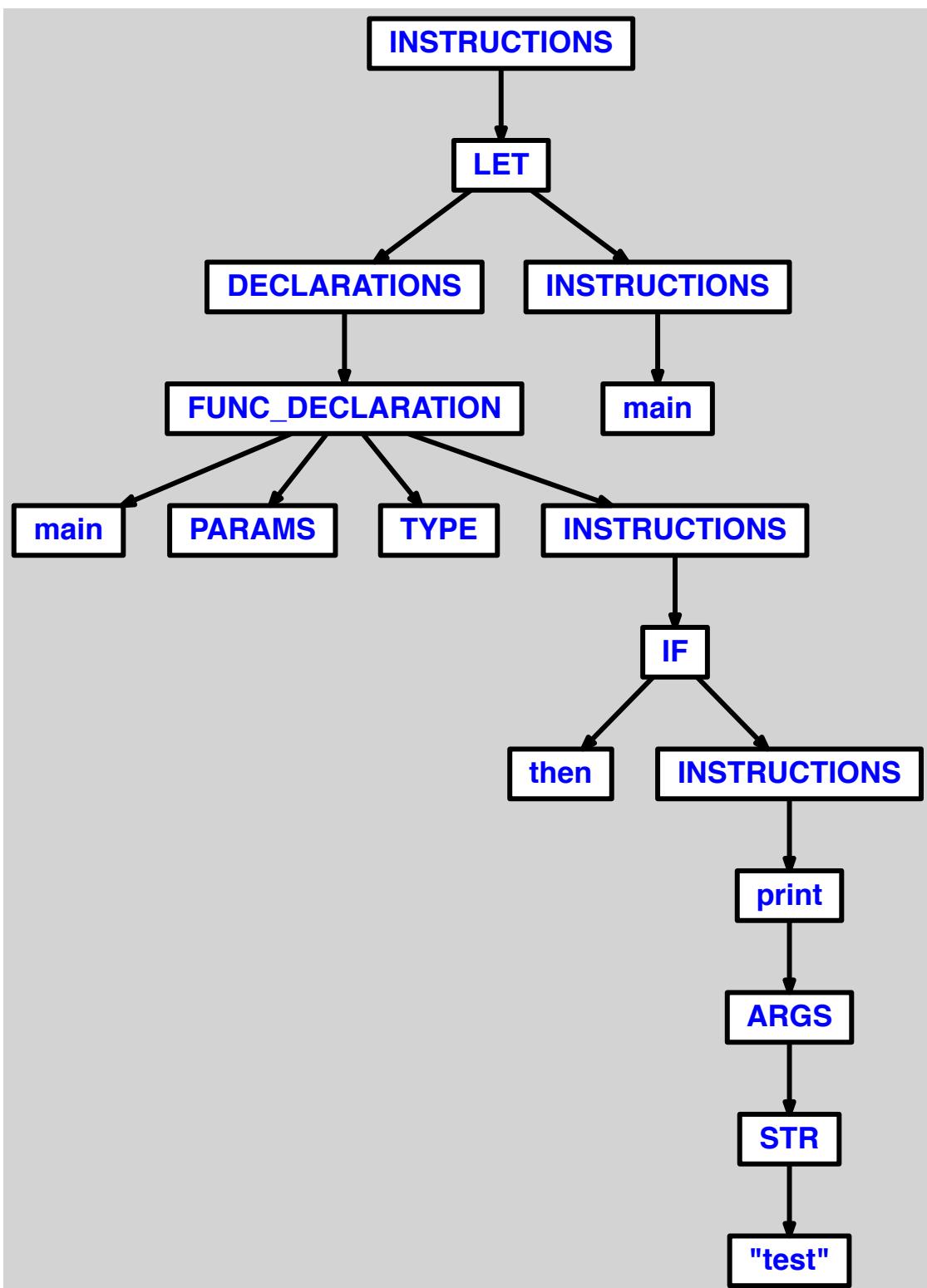
### 9.1.5 <if then> avec oubli du <then>

```
1 let
2   var i := 0
3
4   function main() =
5     if i = 0
6       print("test")
7 in main() end
```



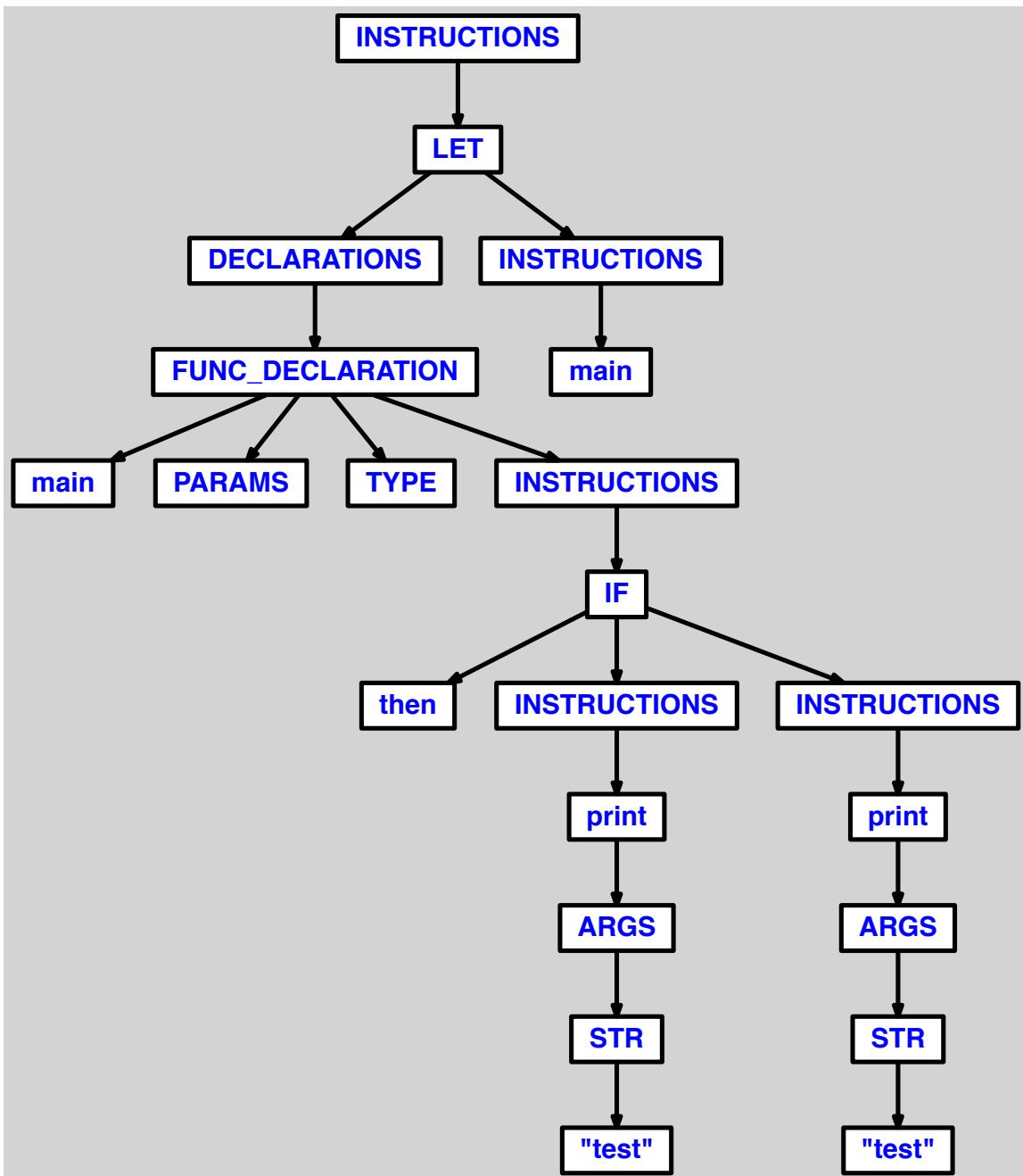
### 9.1.6 <if then> avec oubli de la condition

```
1 let
2   function main() =
3     if then
4       print("test")
5   in main() end
```



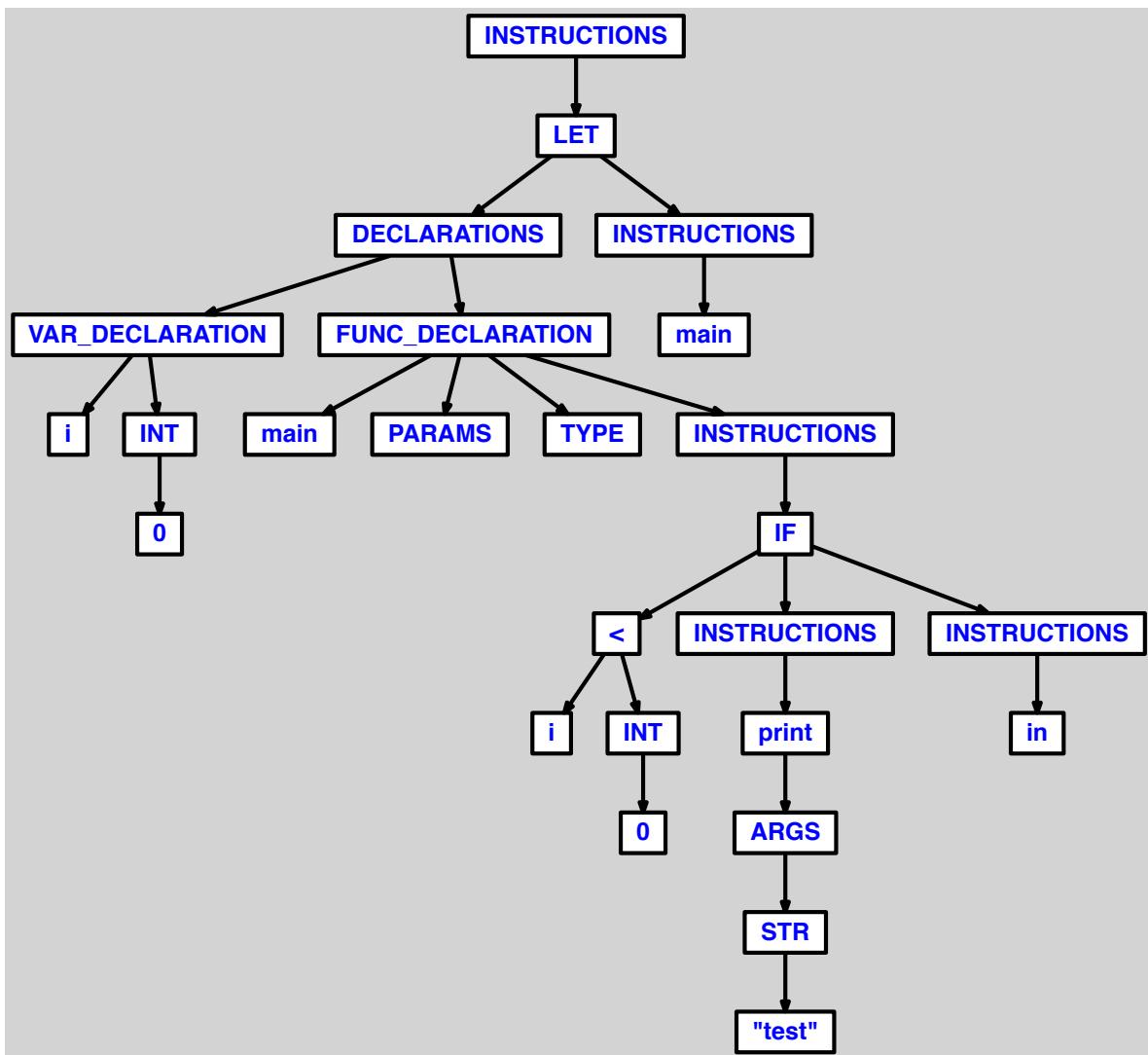
### 9.1.7 <if then else> avec oubli de la condition

```
1 let
2     function main() =
3         if then
4             print("test")
5         else
6             print("test")
7 in main() end
```



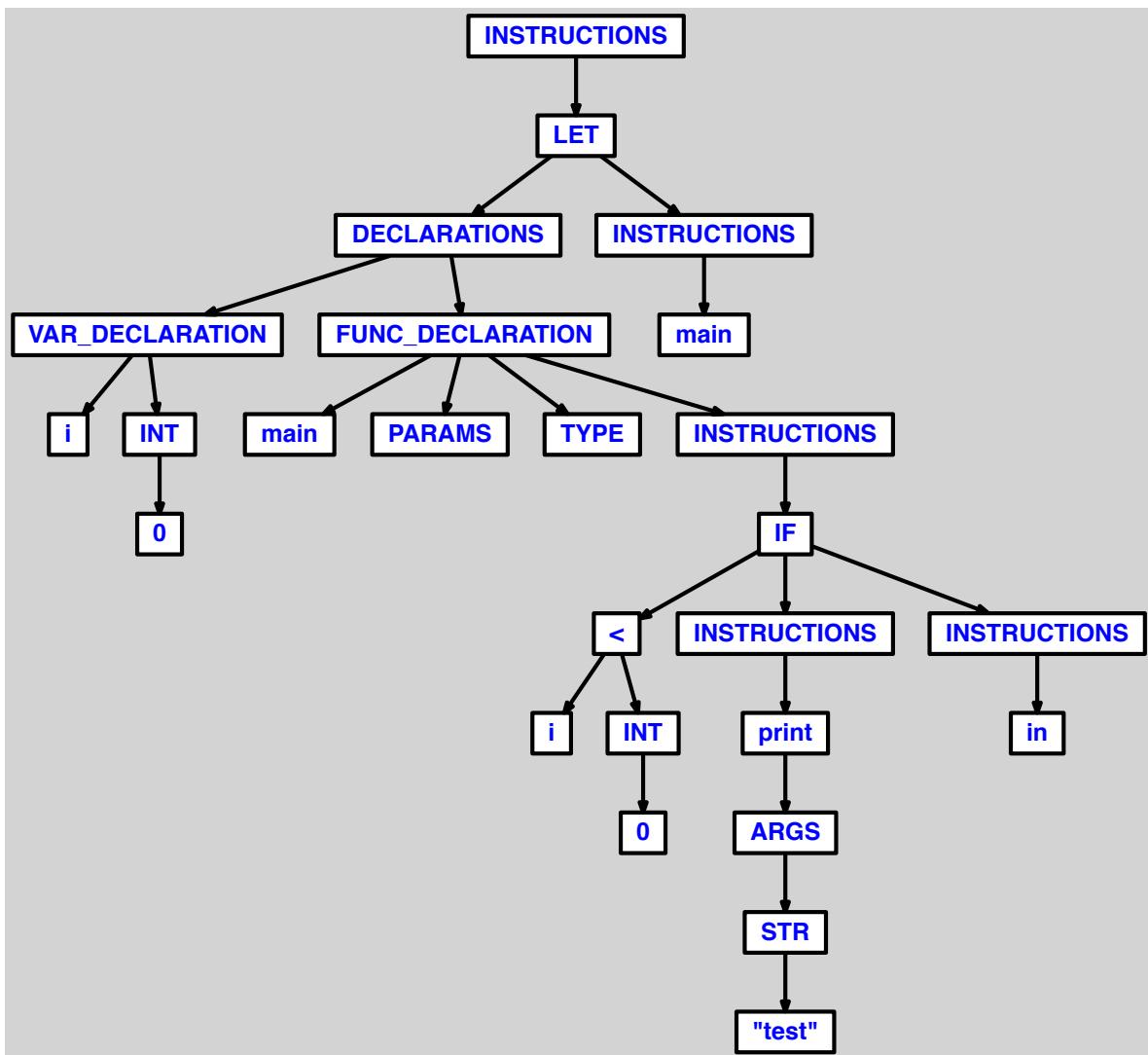
### 9.1.8 <if then else> sans instruction dans <else>

```
1 let
2   var i := 0
3
4   function main() =
5     if i < 0 then
6       print("test")
7     else
8   in main() end
```



### 9.1.9 <if then else> avec ligne vide dans <else>

```
1 let
2   var i := 0
3
4   function main() =
5     if i < 0 then
6       print("test")
7     else
8
9 in main() end
```



### 9.1.10 <if then> avec condition entiere

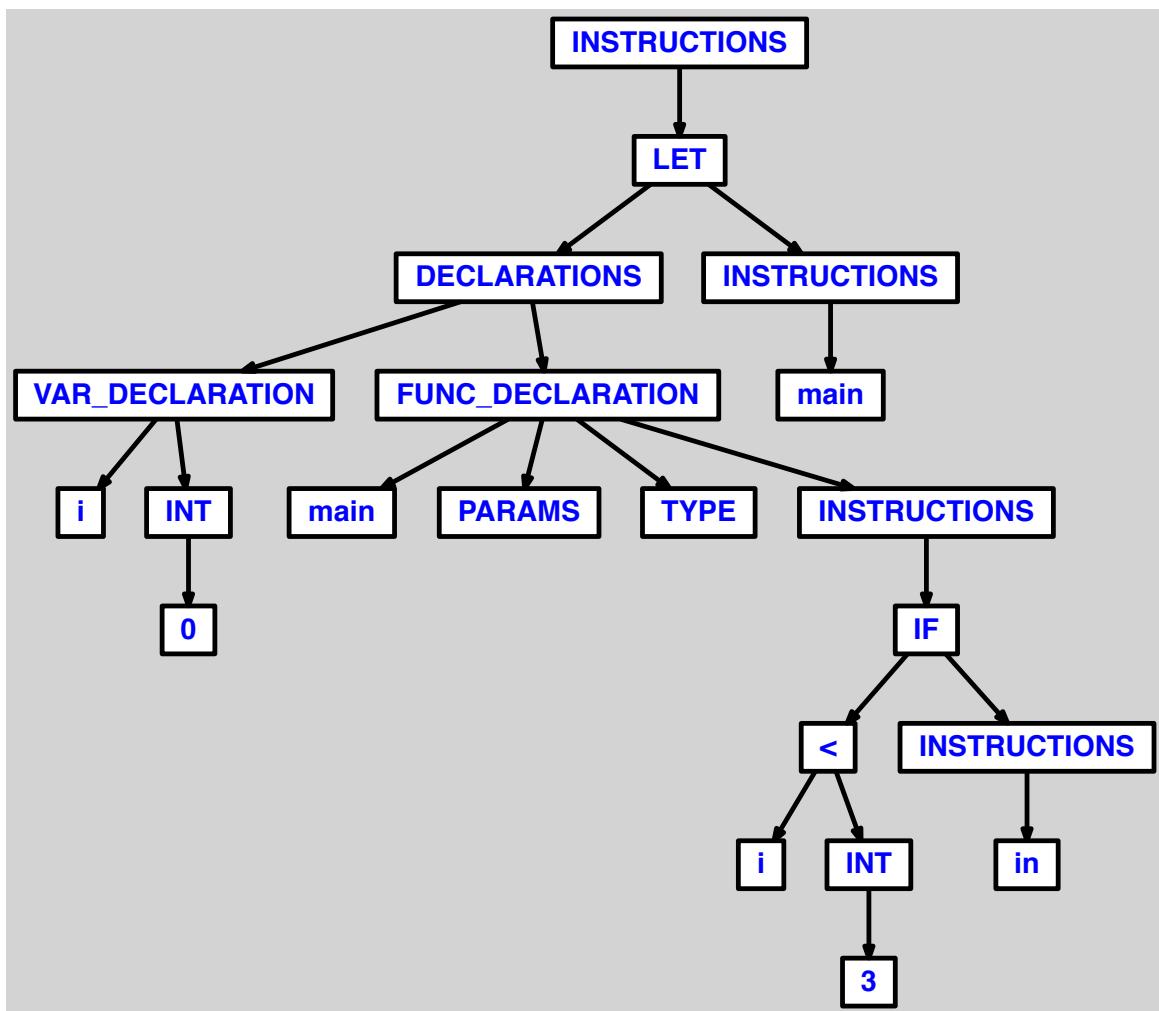
```
1 let
2   function main()() =
3     if 1 then
4       print("test")
5   in main() end
```

Pas d'AST, problème de syntaxe.

## 9.2 OK

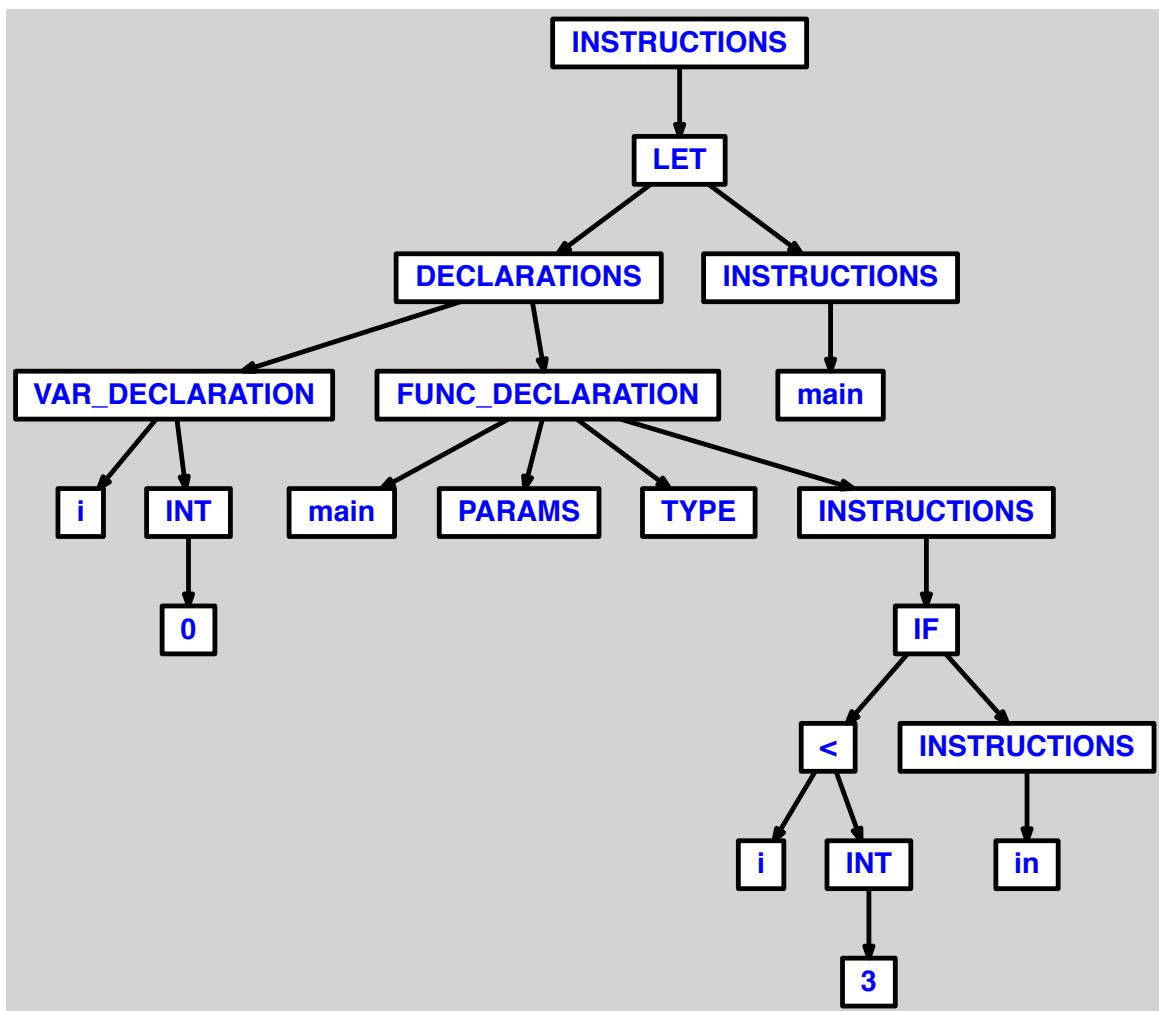
### 9.2.1 <if then> sans instruction

```
1 let
2   var i := 0
3
4   function main() =
5     if i < 3 then
6       in main() end
```



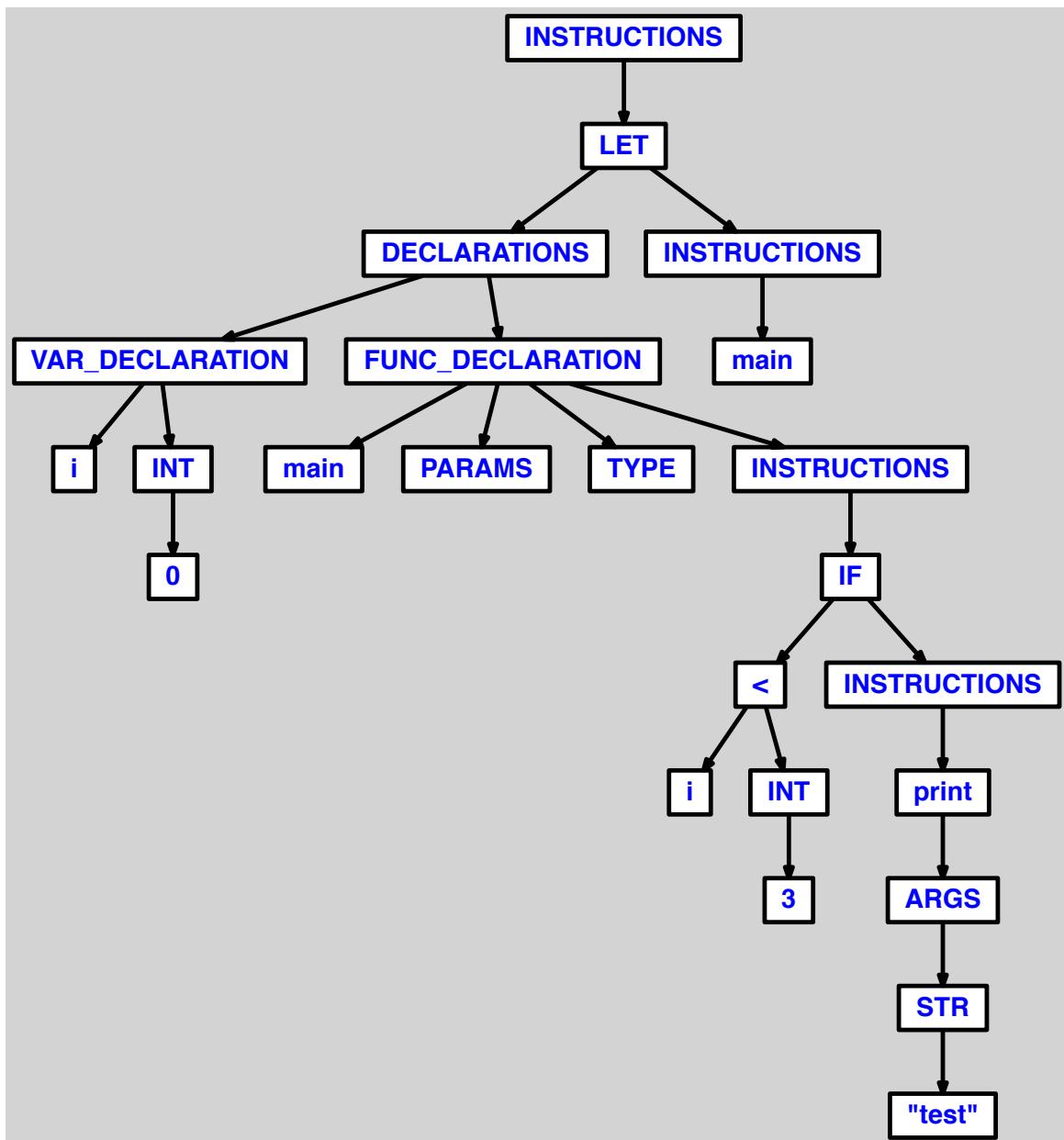
### 9.2.2 <if then> avec ligne vide

```
1 let
2   var i := 0
3
4   function main() =
5     if i < 3 then
6
7   in main() end
```



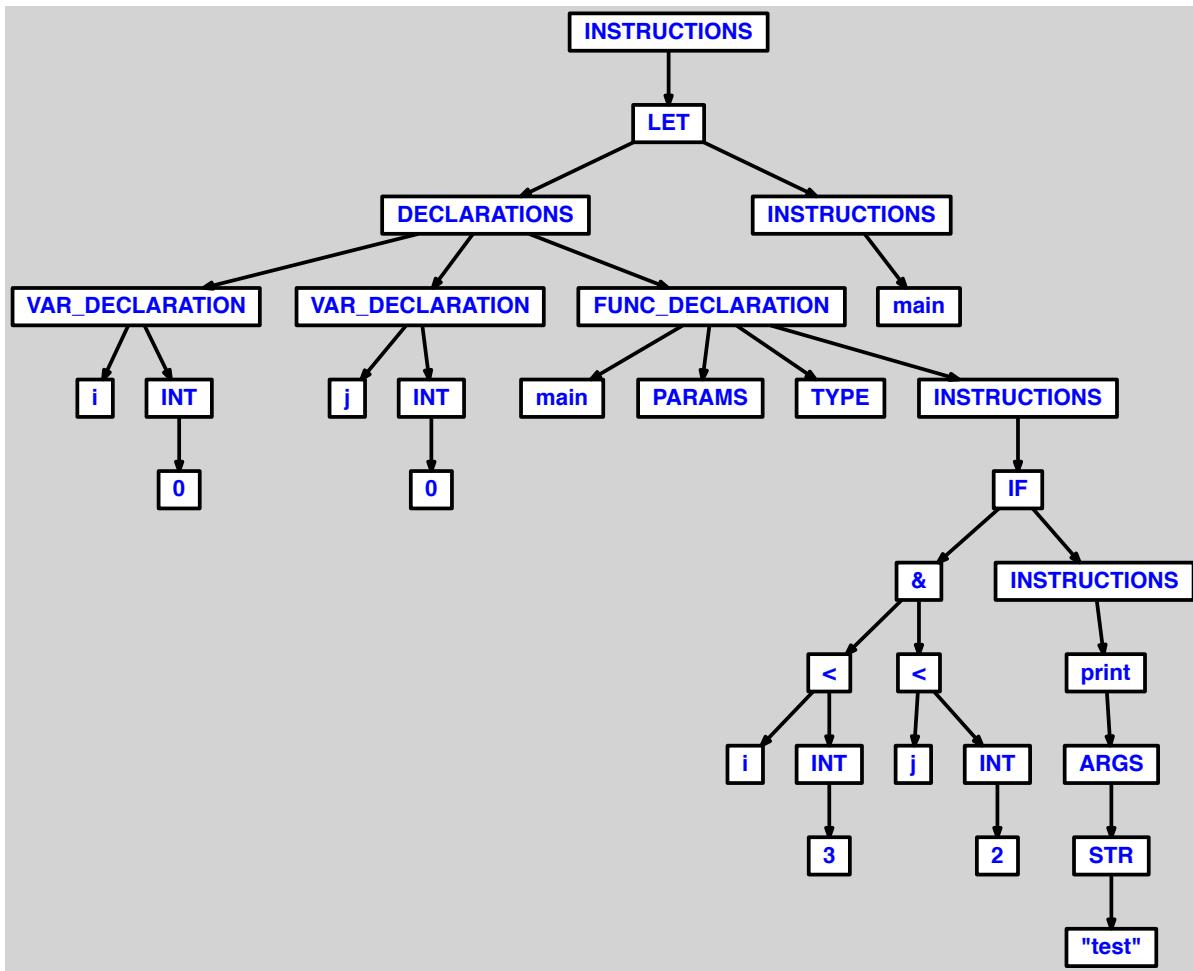
### 9.2.3 <if then> avec simple condition

```
1 let
2   var i := 0
3
4   function main() =
5     if i < 3 then
6       print("test")
7   in main() end
```



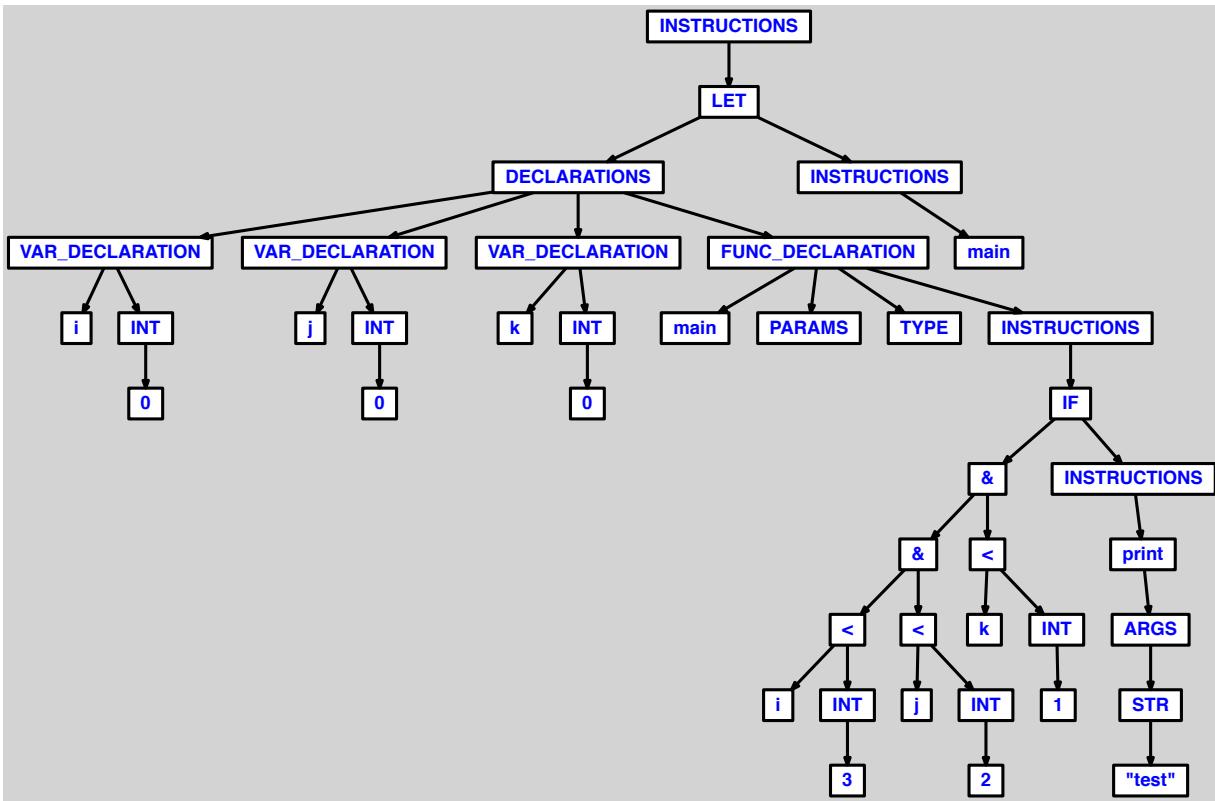
#### 9.2.4 <if then> avec double-condition

```
1 let
2   var i := 0
3   var j := 0
4
5   function main() =
6     if i < 3 & j < 2 then
7       print("test")
8 in main() end
```



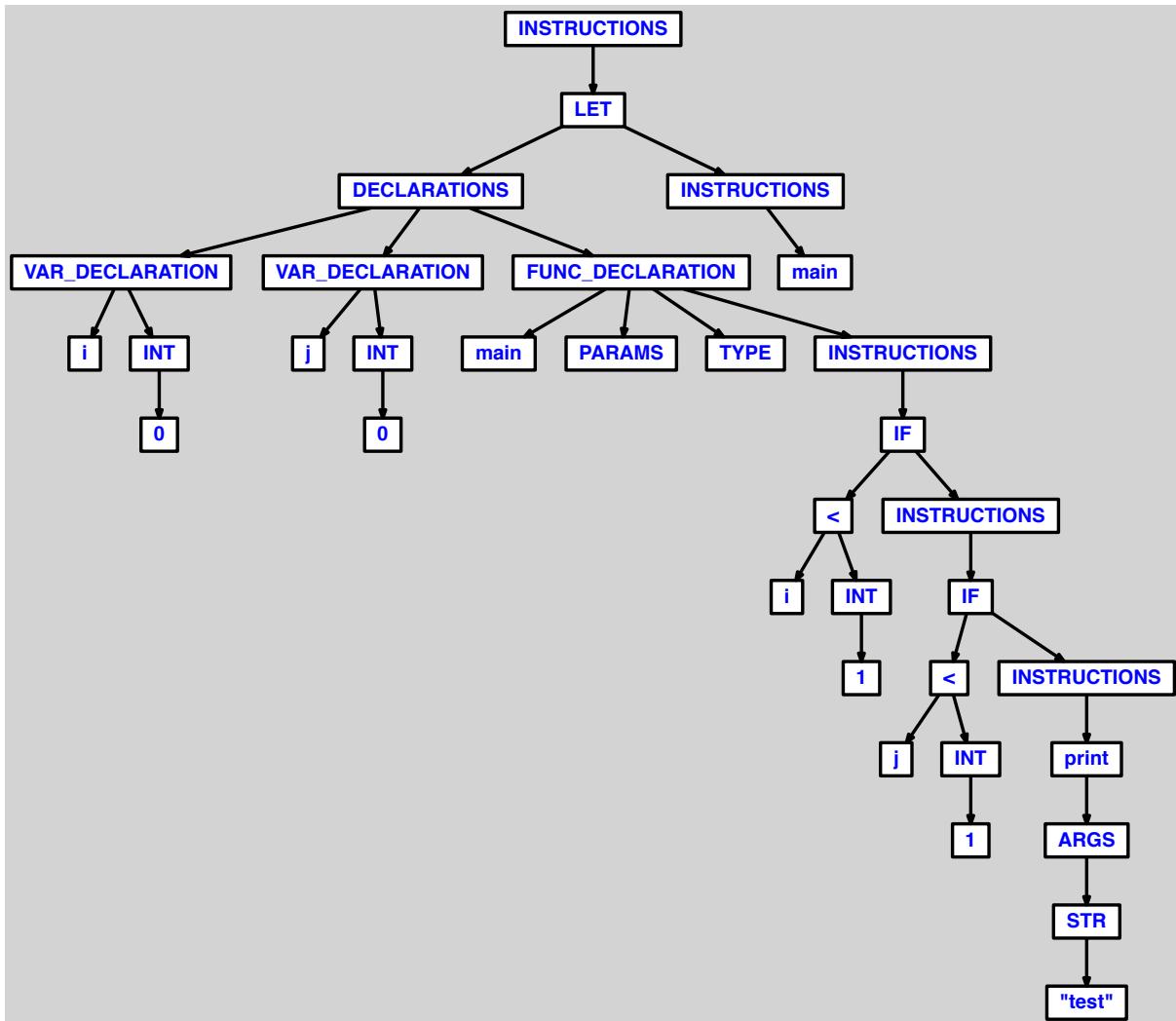
### 9.2.5 <if then> avec triple-condition

```
1 let
2     var i := 0
3     var j := 0
4     var k := 0
5
6     function main() =
7         if i < 3 & j < 2 & k < 1 then
8             print("test")
9     in main() end
```



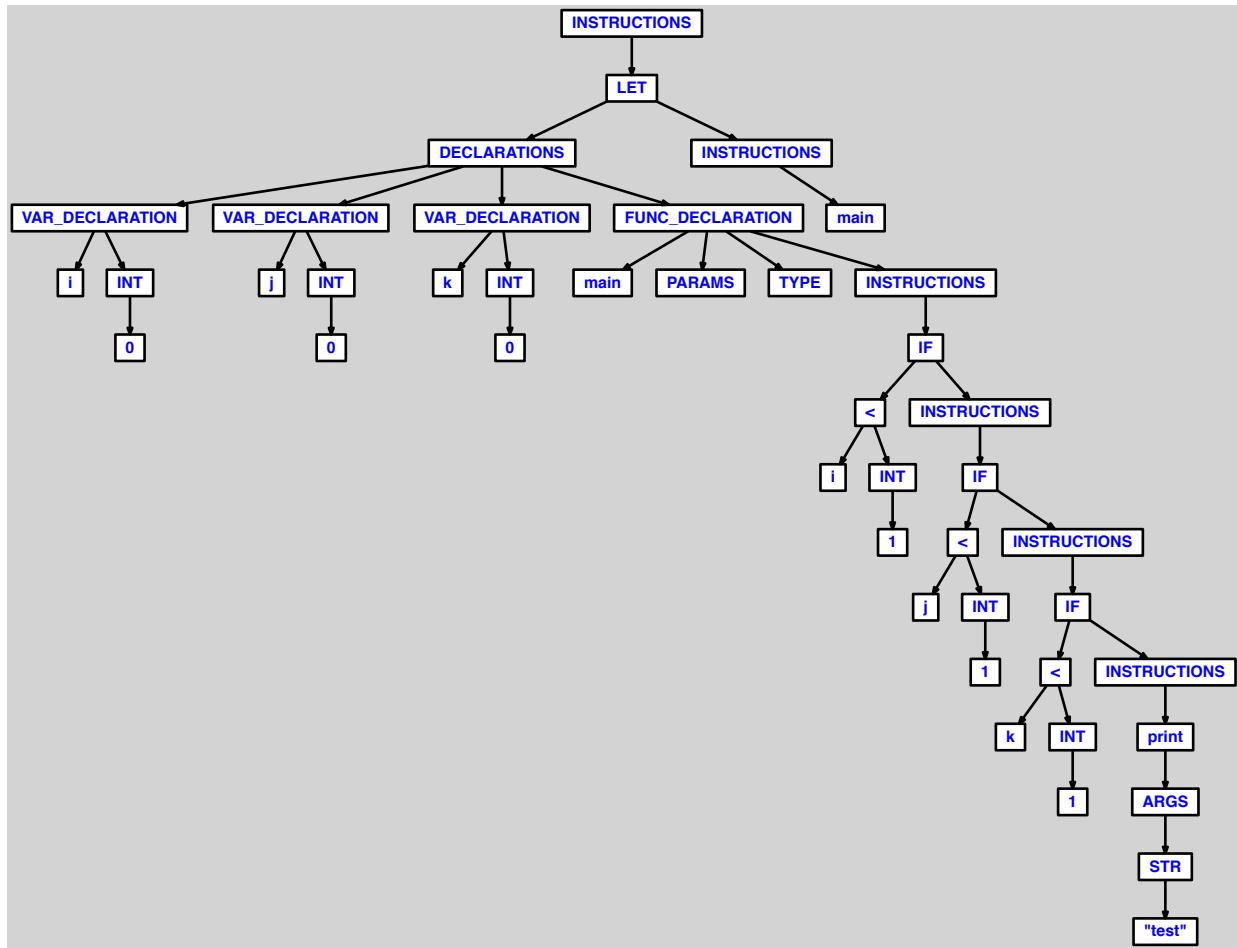
### 9.2.6 double-imbrication de <if then>

```
1 let
2     var i := 0
3     var j := 0
4
5     function main() =
6         if i < 1 then
7             if j < 1 then
8                 print("test")
9 in main() end
```



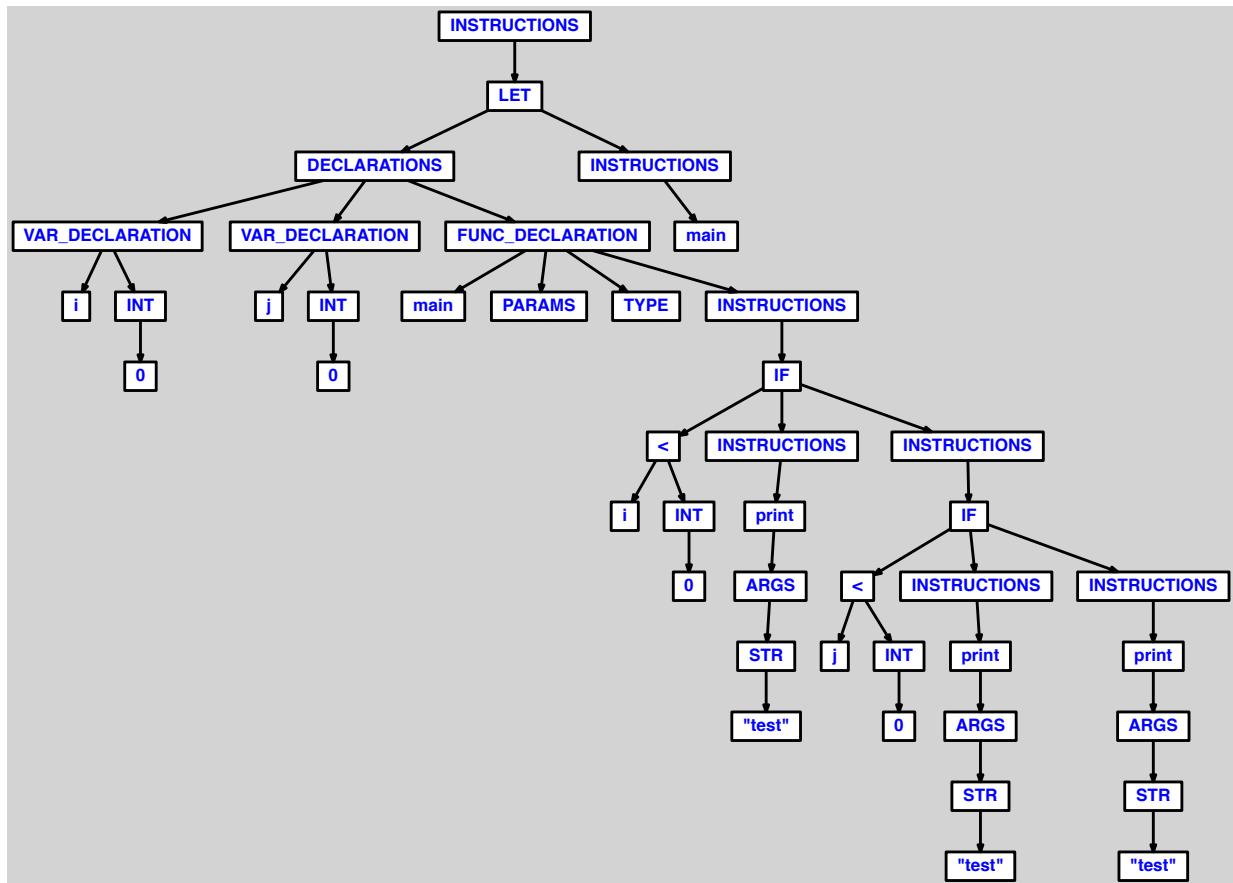
### 9.2.7 triple-imbrication de <if then>

```
1 let
2     var i := 0
3     var j := 0
4     var k := 0
5
6     function main() =
7         if i < 1 then
8             if j < 1 then
9                 if k < 1 then
10                    print("test")
11    in main() end
```



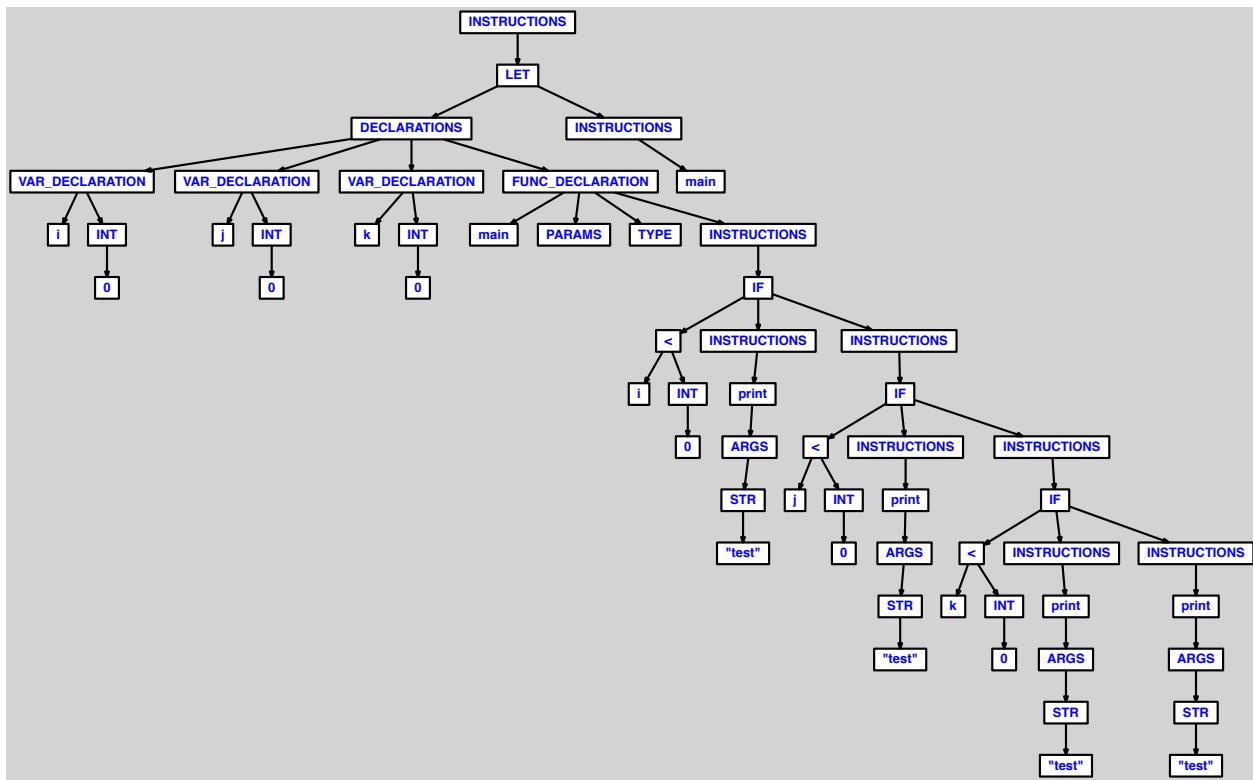
### 9.2.8 double-imbrication de <if then else>

```
1 let
2   var i := 0
3   var j := 0
4
5   function main() =
6     if i < 0 then
7       print("test")
8     else
9       if j < 0 then
10         print("test")
11       else
12         print("test")
13 in main() end
```



### 9.2.9 triple-imbrication de <if then else>

```
1 let
2   var i := 0
3   var j := 0
4   var k := 0
5
6   function main() =
7     if i < 0 then
8       print("test")
9     else
10      if j < 0 then
11        print("test")
12      else
13        if k < 0 then
14          print("test")
15        else
16          print("test")
17 in main() end
```

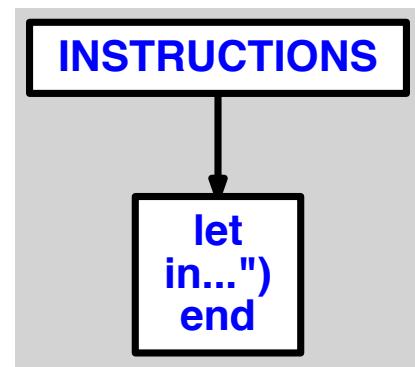


## 10 let

### 10.1 KO

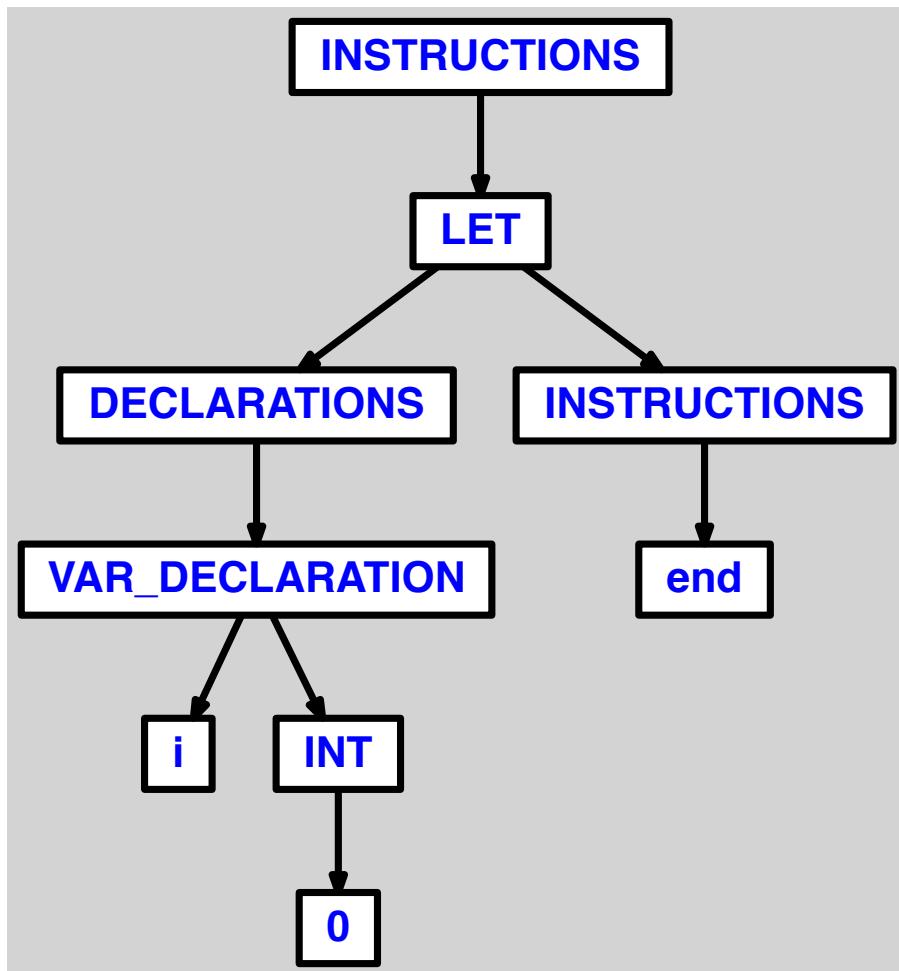
#### 10.1.1 <let> sans declaration

```
1 let
2 in
3   print("test")
4 end
```



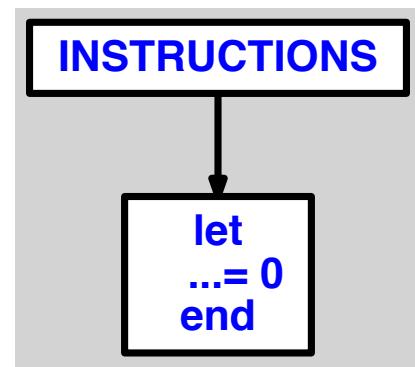
### 10.1.2 <let> sans instruction

```
1 let
2   var i := 0
3 in
4 end
```



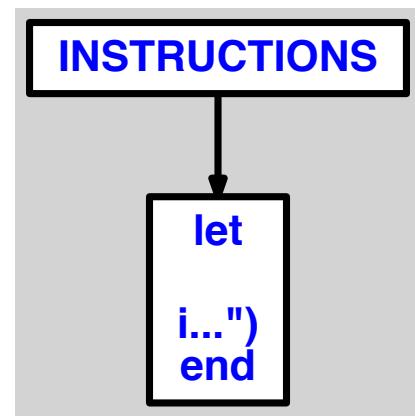
### 10.1.3 <let> avec inversion de declaration et instruction

```
1 let
2   printi(i)
3 in
4   var i := 0
5 end
```



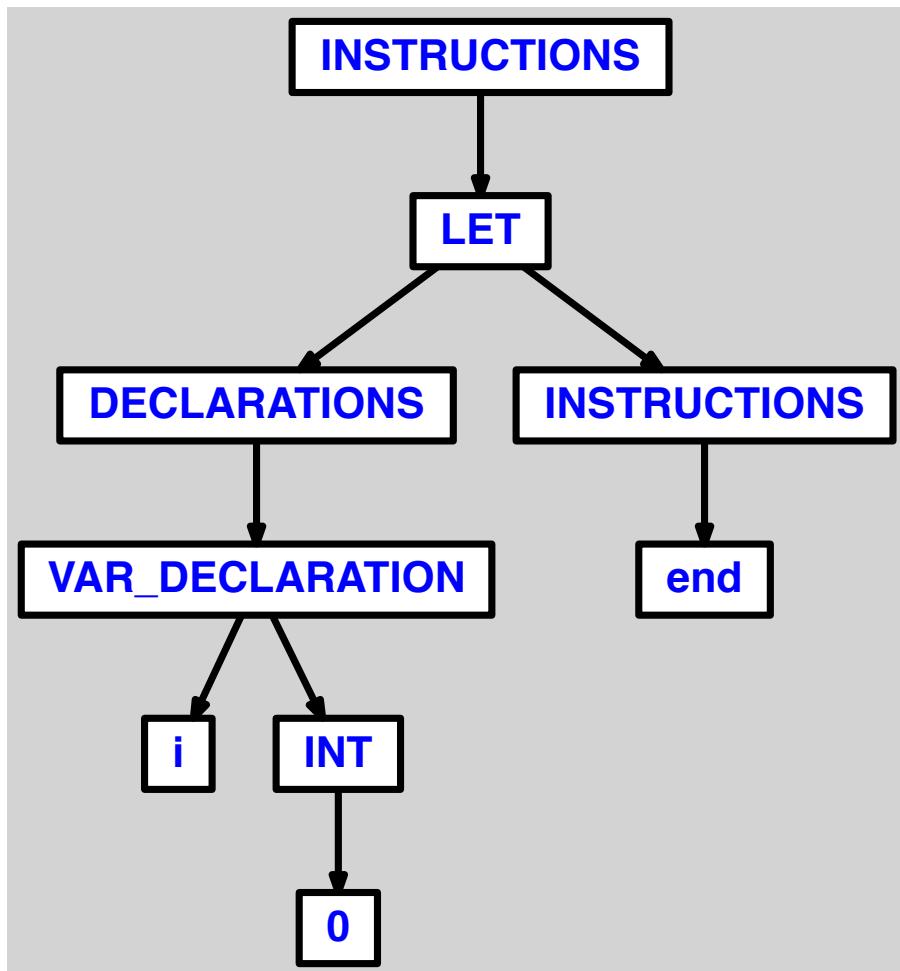
#### 10.1.4 <let> avec declaration vide

```
1 let
2
3 in
4     print("test")
5 end
```



### 10.1.5 <let> avec instruction vide

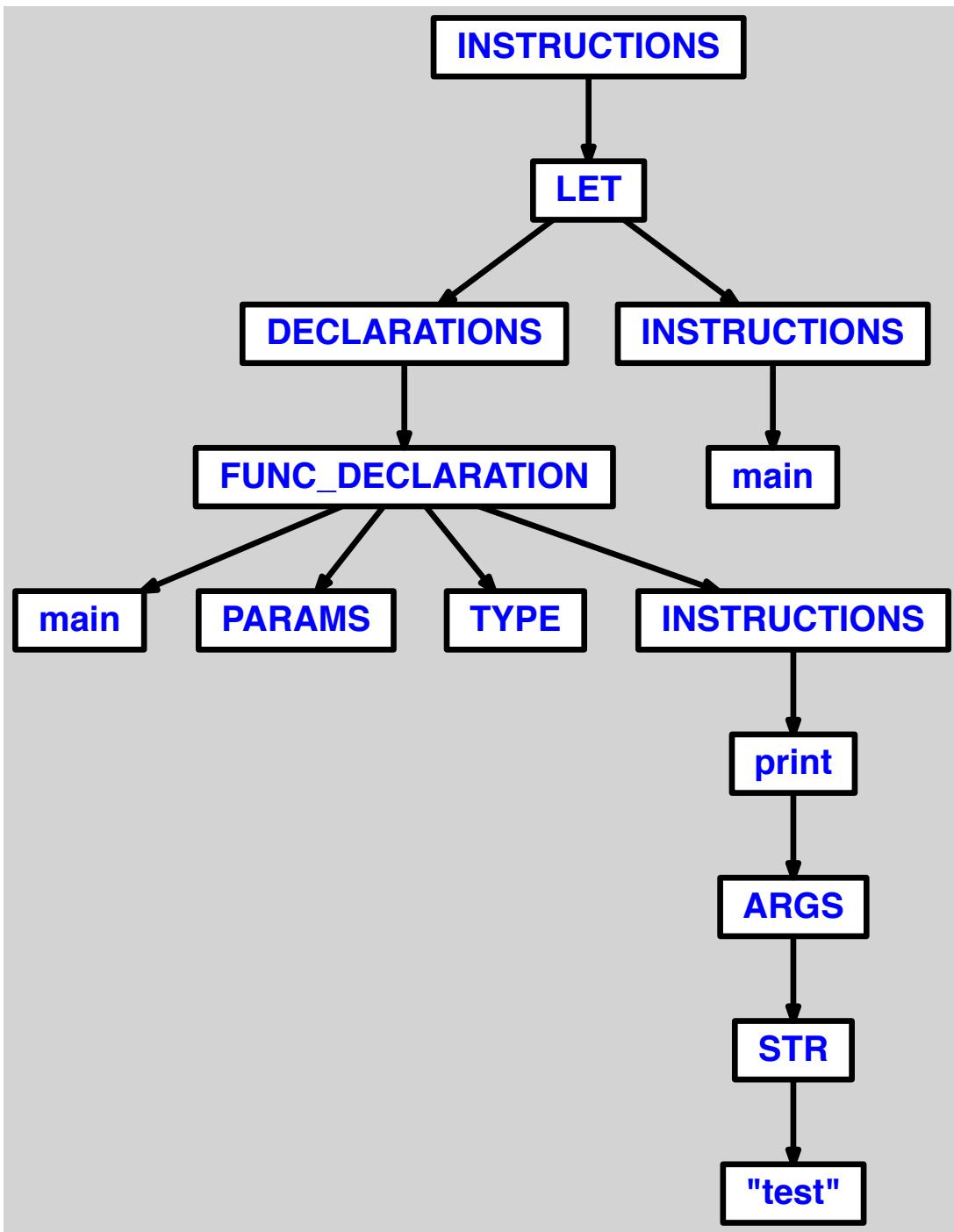
```
1 let
2   var i := 0
3 in
4
5 end
```



## 10.2 OK

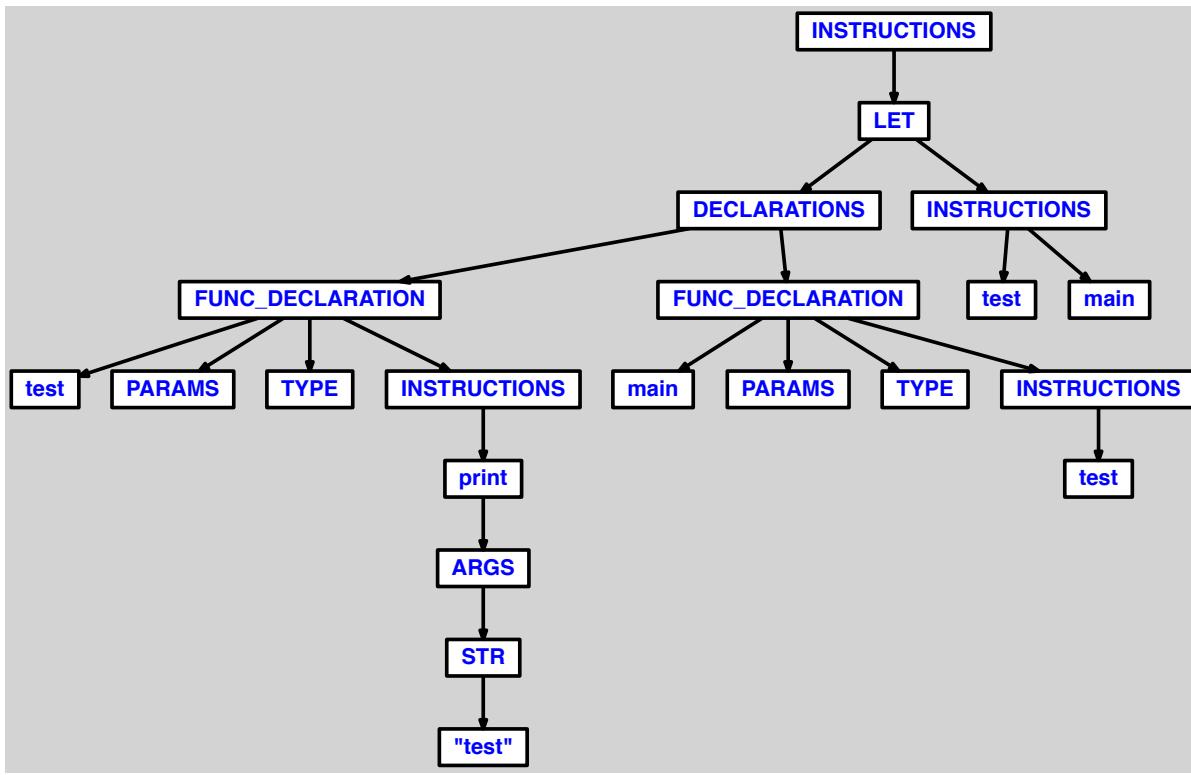
### 10.2.1 <let> avec 1 declaration de fonction et 1 appel de fonction

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



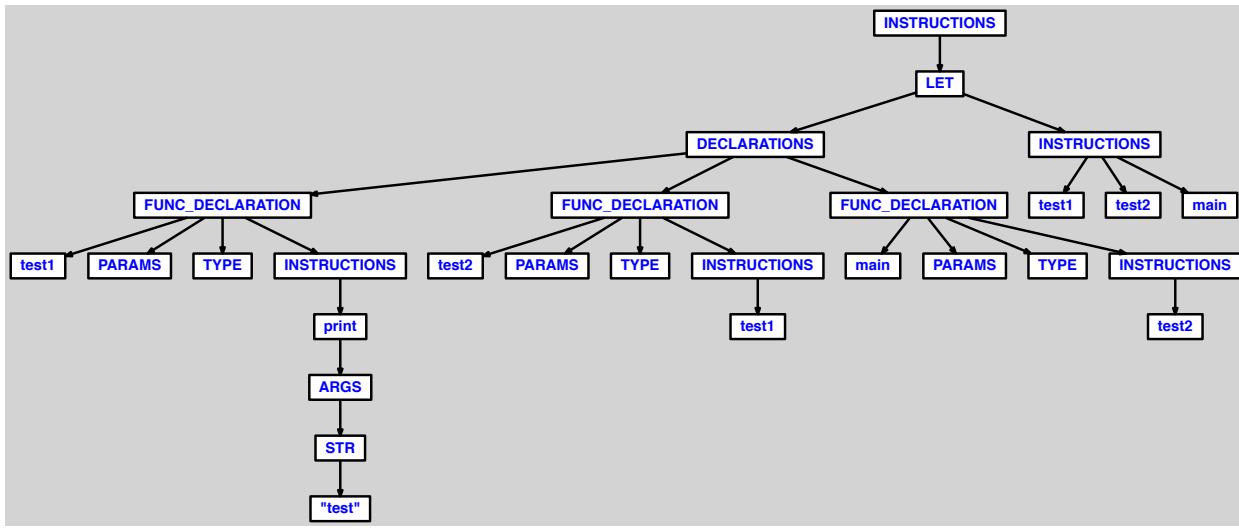
### 10.2.2 <let> avec 2 declarations de fonction et 2 appels de fonction

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



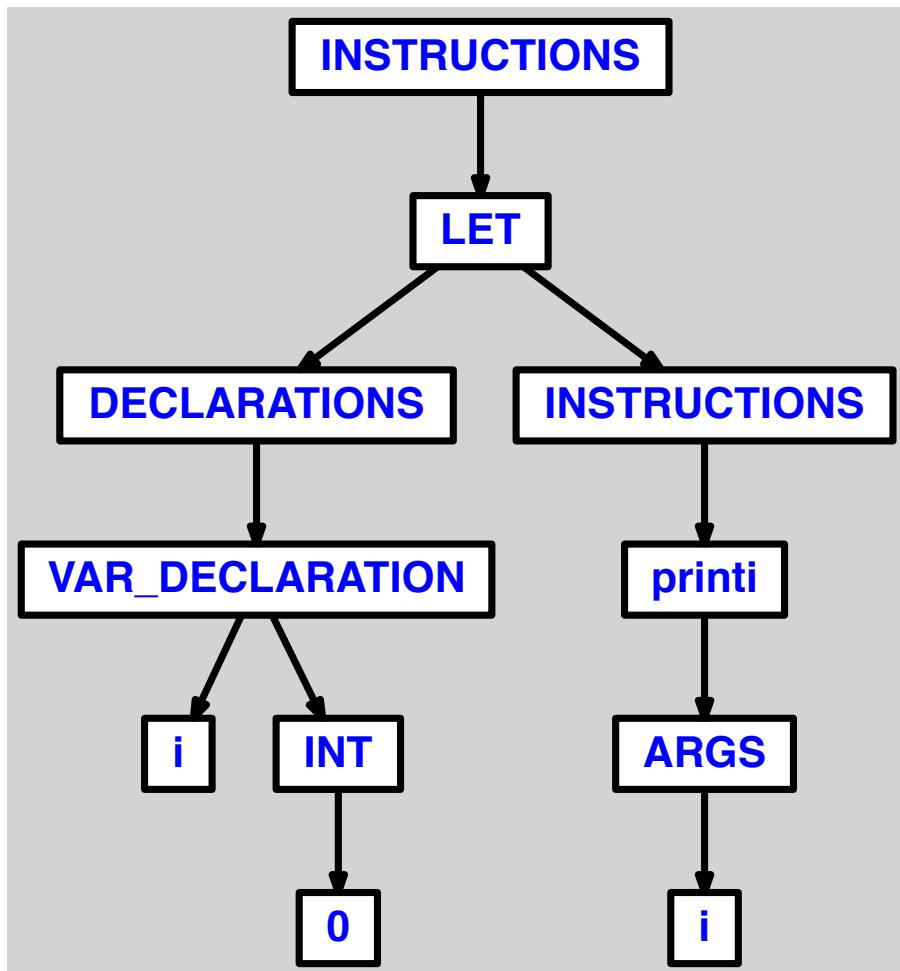
### 10.2.3 <let> avec 3 declarations de fonction et 3 appels de fonction

```
1 let
2     function test1() = print("test")
3
4     function test2() = test1()
5
6     function main() = test2()
7 in
8     (test1(); test2(); main())
9 end
```



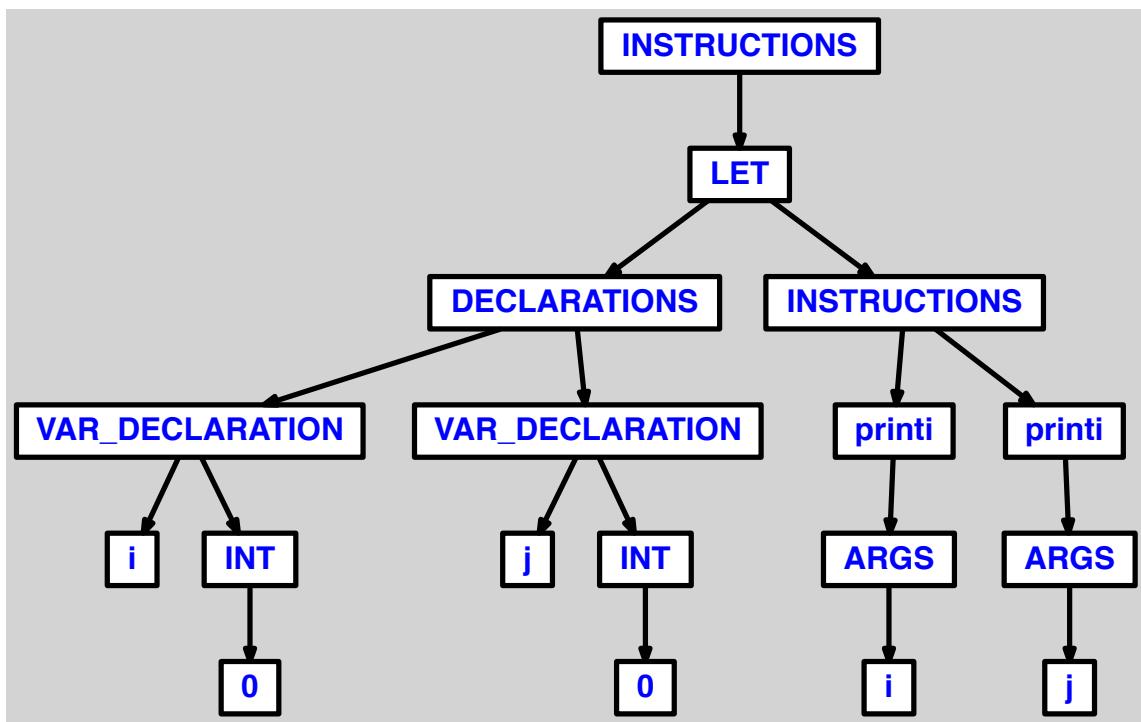
#### 10.2.4 <let> avec 1 déclaration de variable et 1 instruction

```
1 let
2   var i := 0
3 in
4   printi(i)
5 end
```



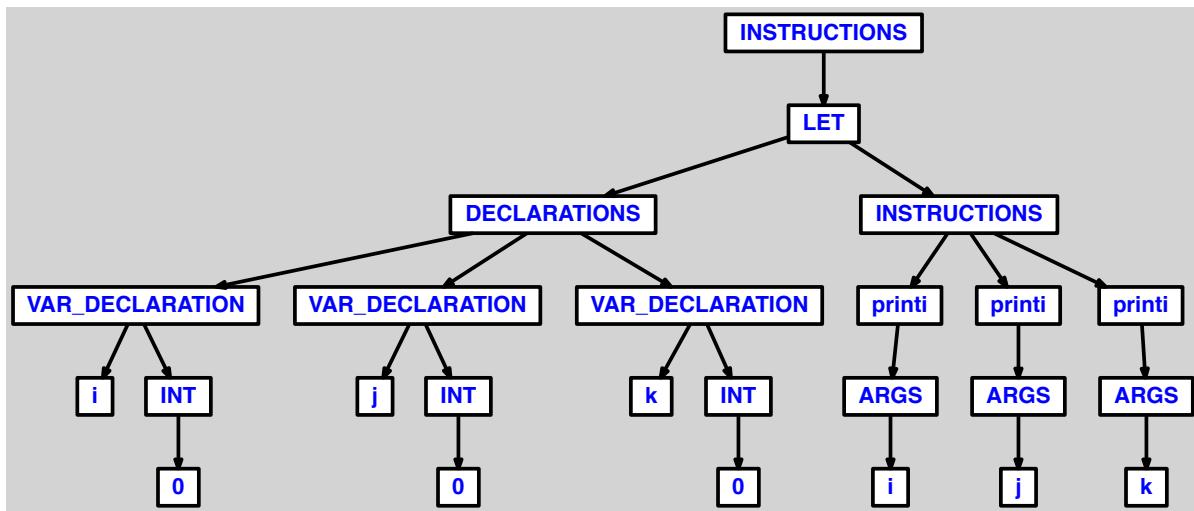
#### 10.2.5 <let> avec 2 declarations de variable et 2 instructions

```
1 let
2   var i := 0
3   var j := 0
4 in
5   (printi(i); printi(j))
6 end
```



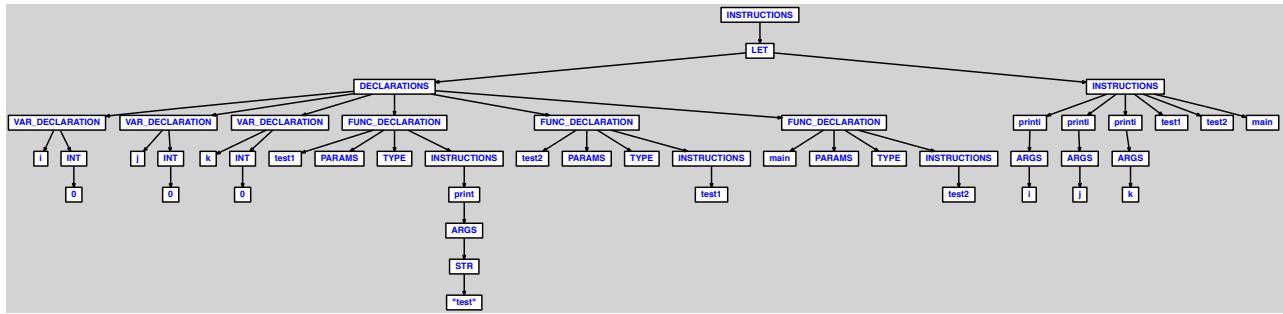
#### 10.2.6 <let> avec 3 declarations de variable et 3 instructions

```
1 let
2   var i := 0
3   var j := 0
4   var k := 0
5 in
6   (printi(i); printi(j); printi(k))
7 end
```



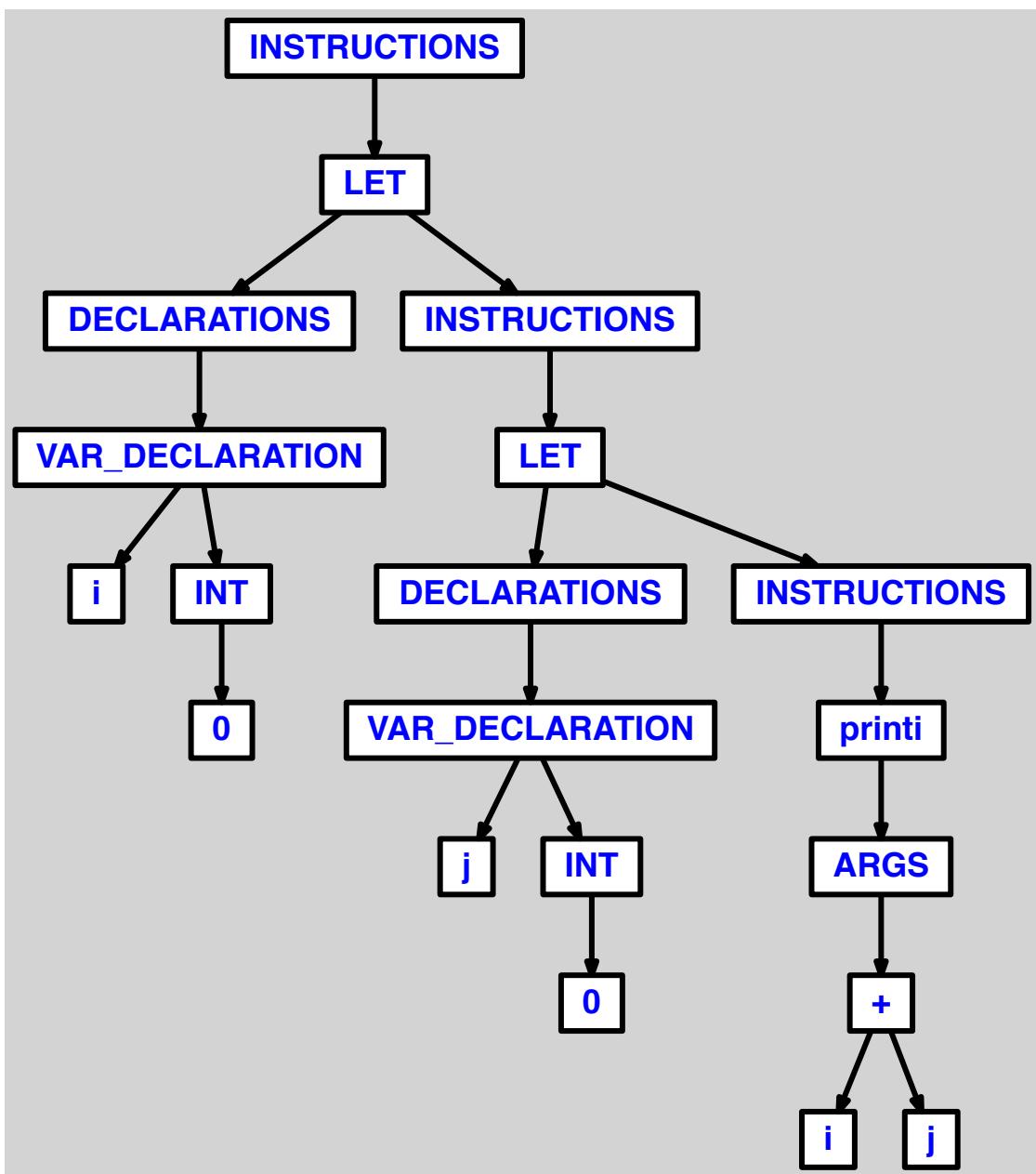
### 10.2.7 <let> avec 3 declarations de variable et fonction, et 3 instructions et appels de fonction

```
1 let
2   var i := 0
3   var j := 0
4   var k := 0
5
6   function test1() = print("test")
7
8   function test2() = test1()
9
10  function main() = test2()
11 in
12   (printi(i);
13   printi(j);
14   printi(k);
15   test1();
16   test2();
17   main())
18 end
```



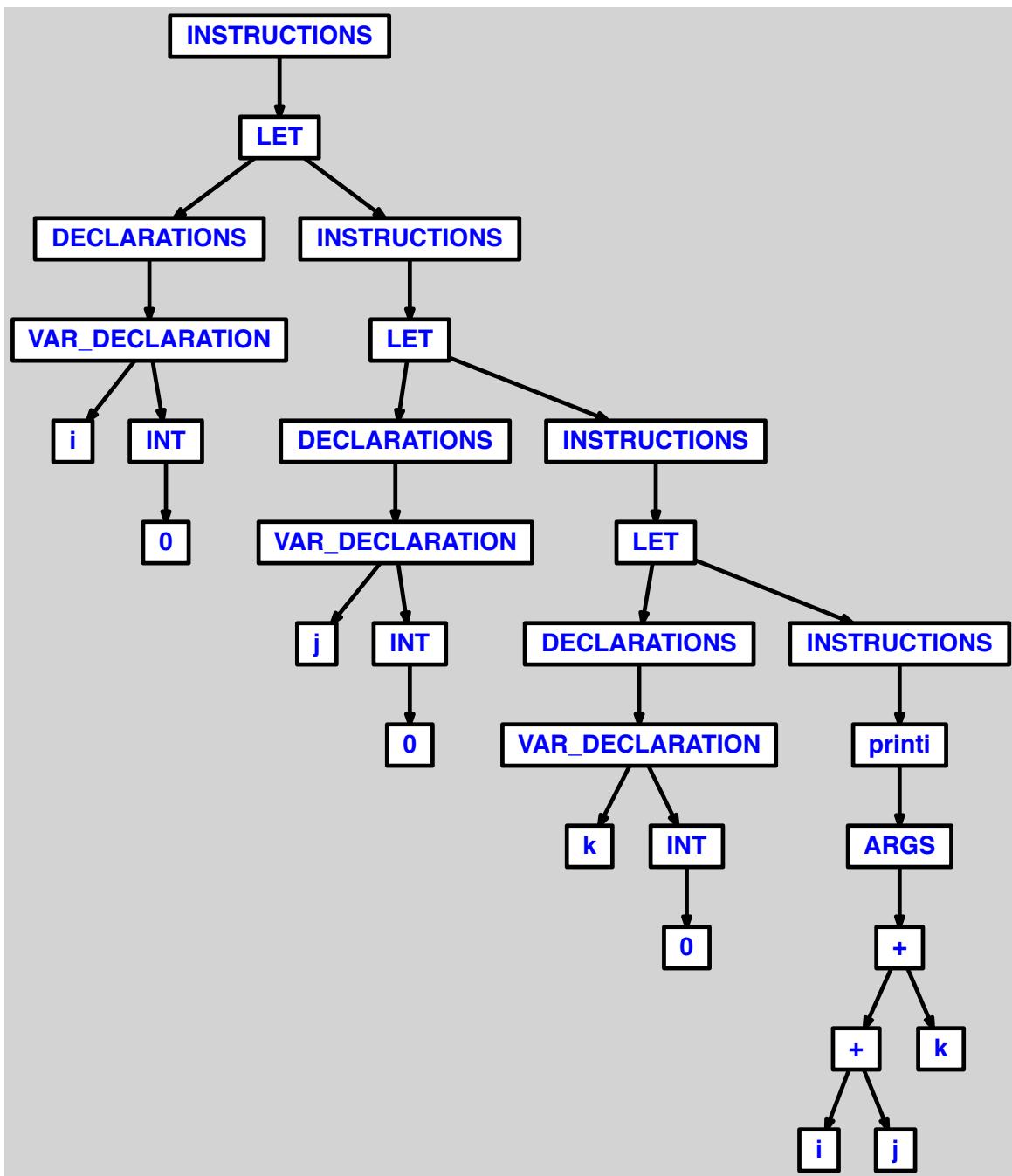
### 10.2.8 <let> avec double-imbrication

```
1 let
2   var i := 0
3 in
4   let
5     var j := 0
6   in
7     printi(i+j)
8   end
9 end
```



### 10.2.9 <let> avec triple-imbrication

```
1 let
2   var i := 0
3 in
4   let
5     var j := 0
6 in
7   let
8     var k := 0
9   in
10    printi(i+j+k)
11  end
12 end
13 end
```

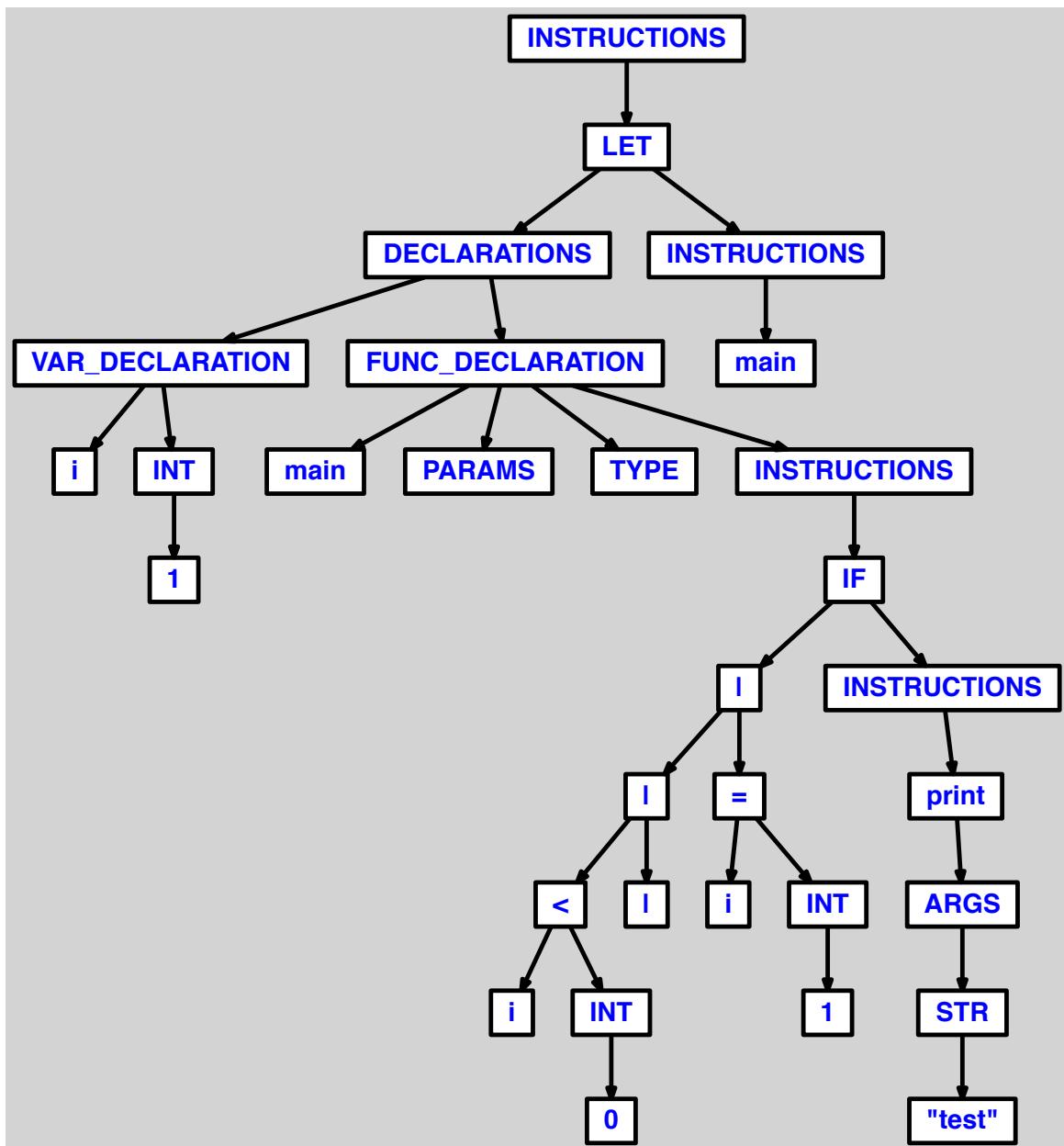


## 11 or

### 11.1 KO

#### 11.1.1 <|> mal écrit

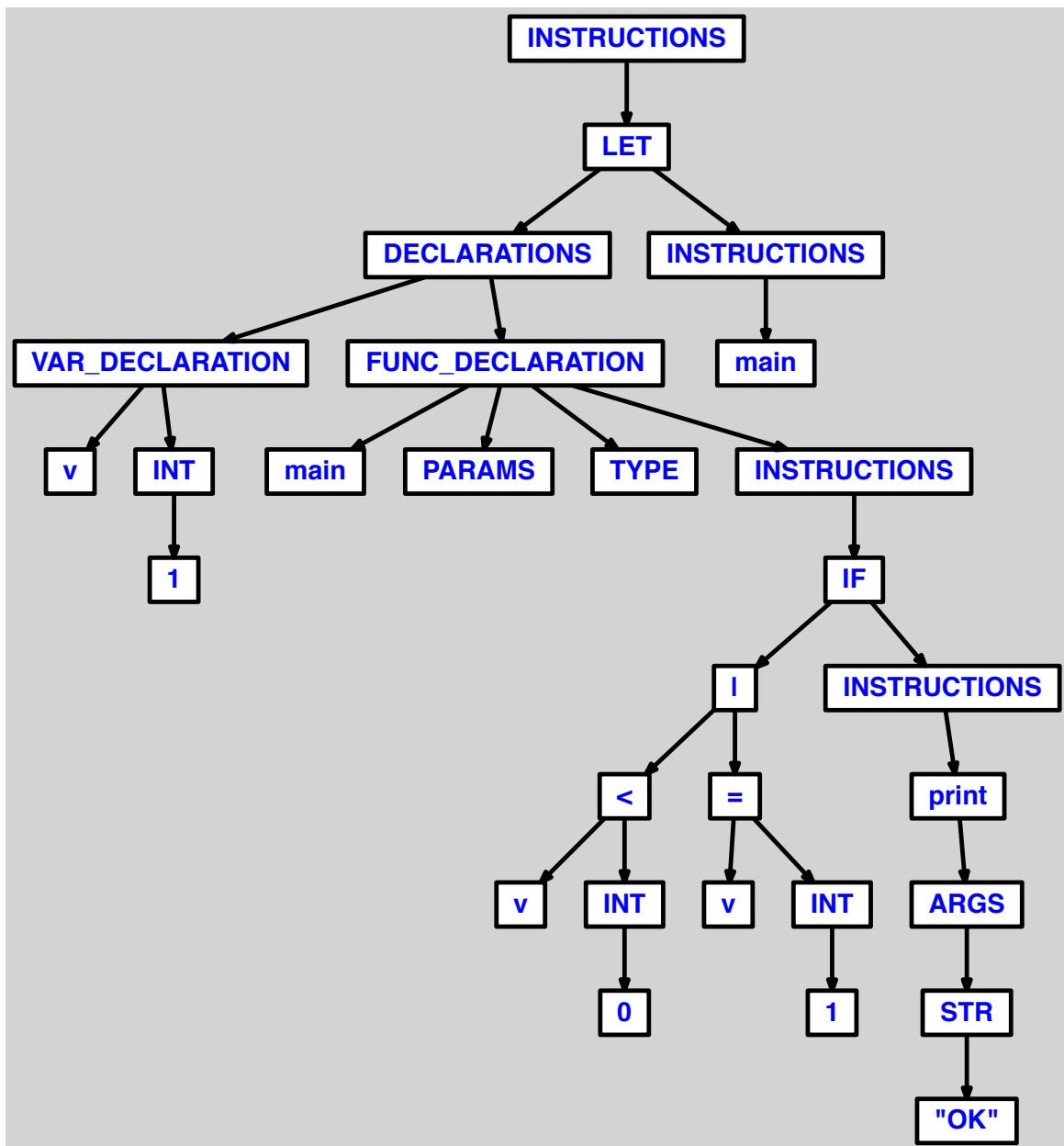
```
1 let
2   var i := 1
3
4   function main() =
5     if i < 0 || i = 1 then print("test")
6 in main() end
```



## 11.2 OK

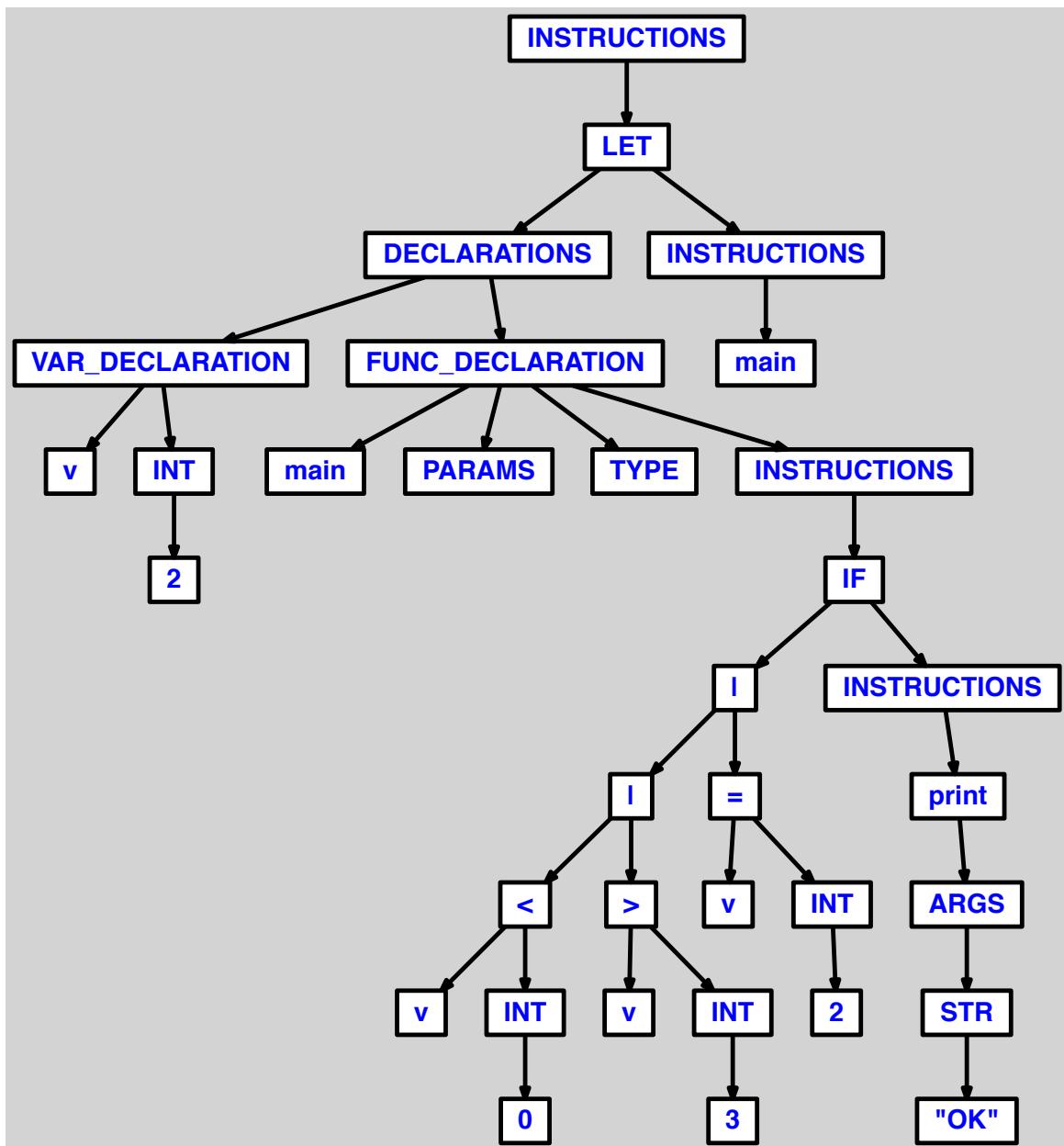
### 11.2.1 condition avec 1 <|>

```
1 let
2   var v := 1
3
4   function main() =
5     if v < 0 | v = 1 then print("OK")
6   in main() end
```



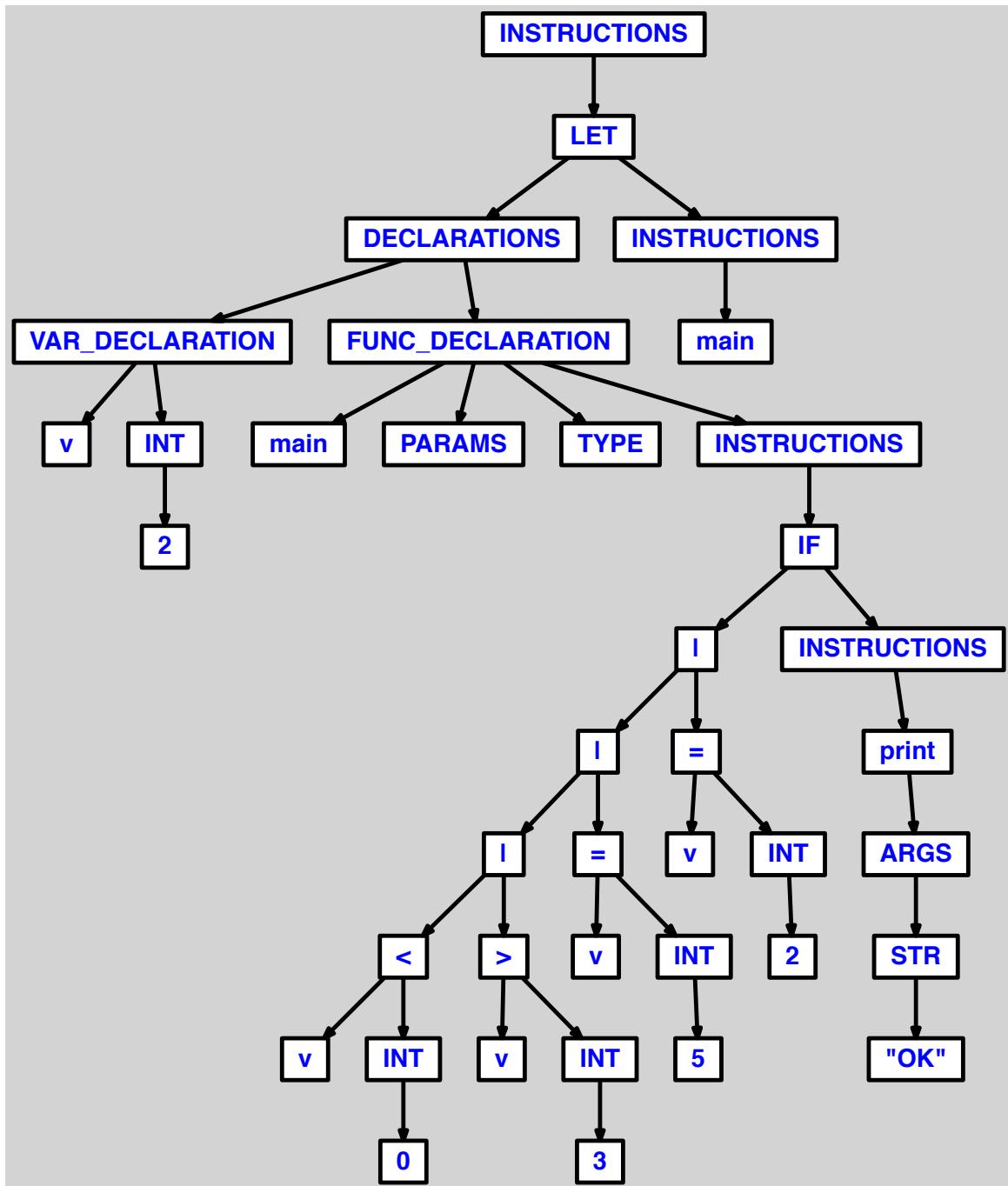
### 11.2.2 condition avec 2 <|>

```
1 let
2   var v := 2
3
4   function main() =
5     if v < 0 | v > 3 | v = 2 then print("OK")
6 in main() end
```



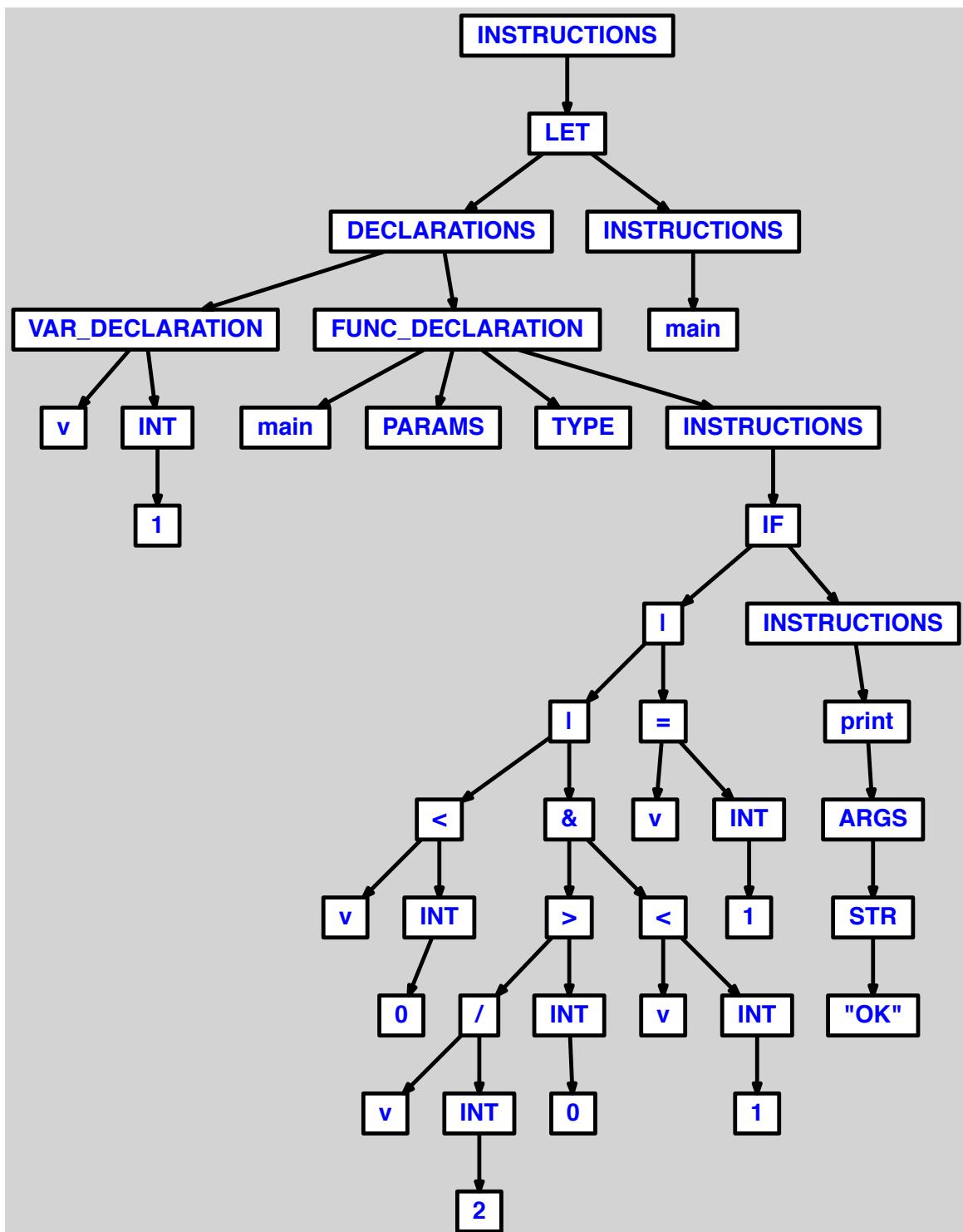
### 11.2.3 condition avec 3 <|>

```
1 let
2   var v := 2
3
4   function main() =
5     if v < 0 | v > 3 | v = 5 | v = 2 then print("OK")
6 in main() end
```



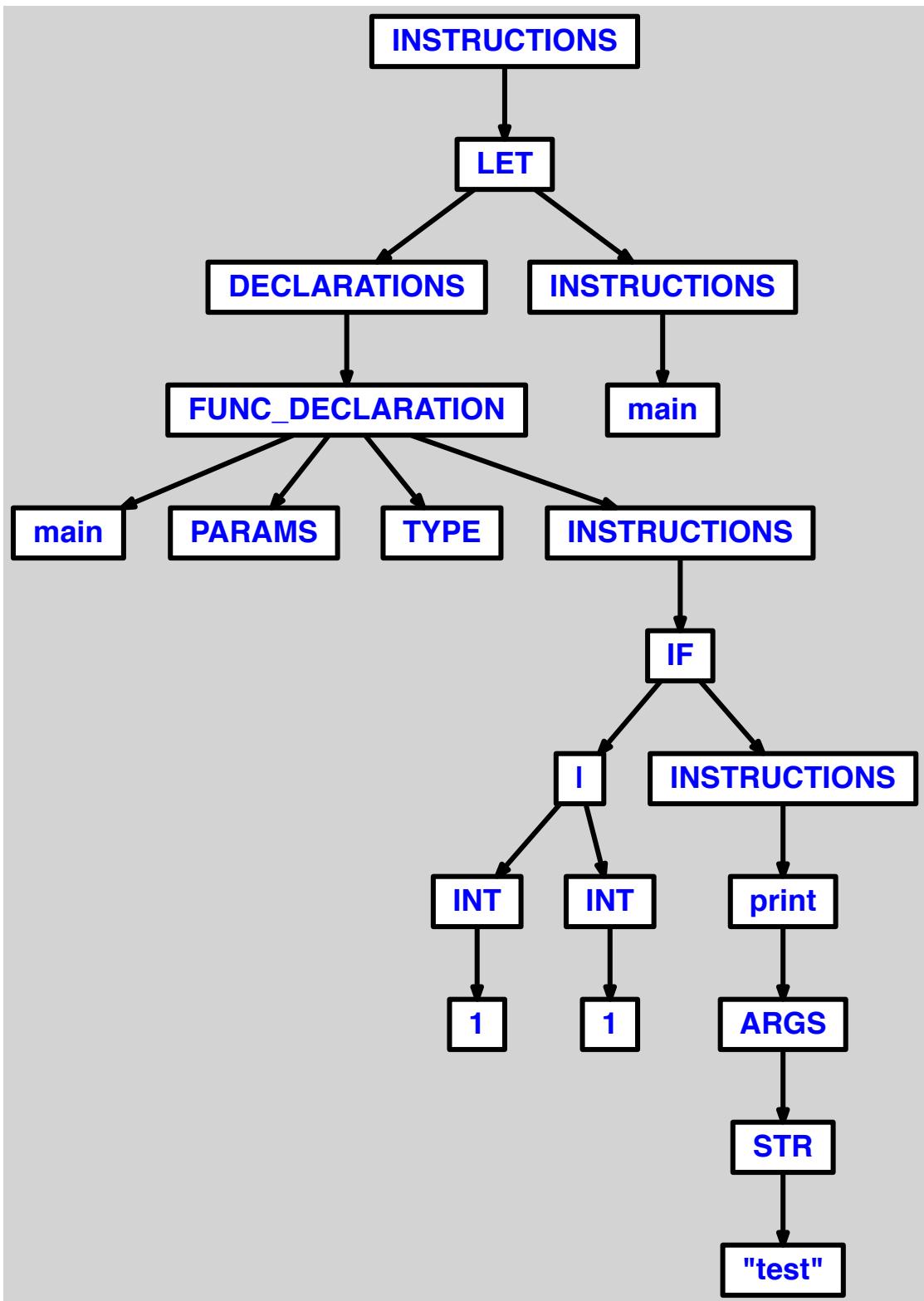
#### 11.2.4 condition avec 2 <|> et 1 <&>

```
1 let
2   var v := 1
3
4   function main() =
5     if v < 0 | v/2 > 0 & v < 1 | v = 1 then print("OK")
6 in main() end
```



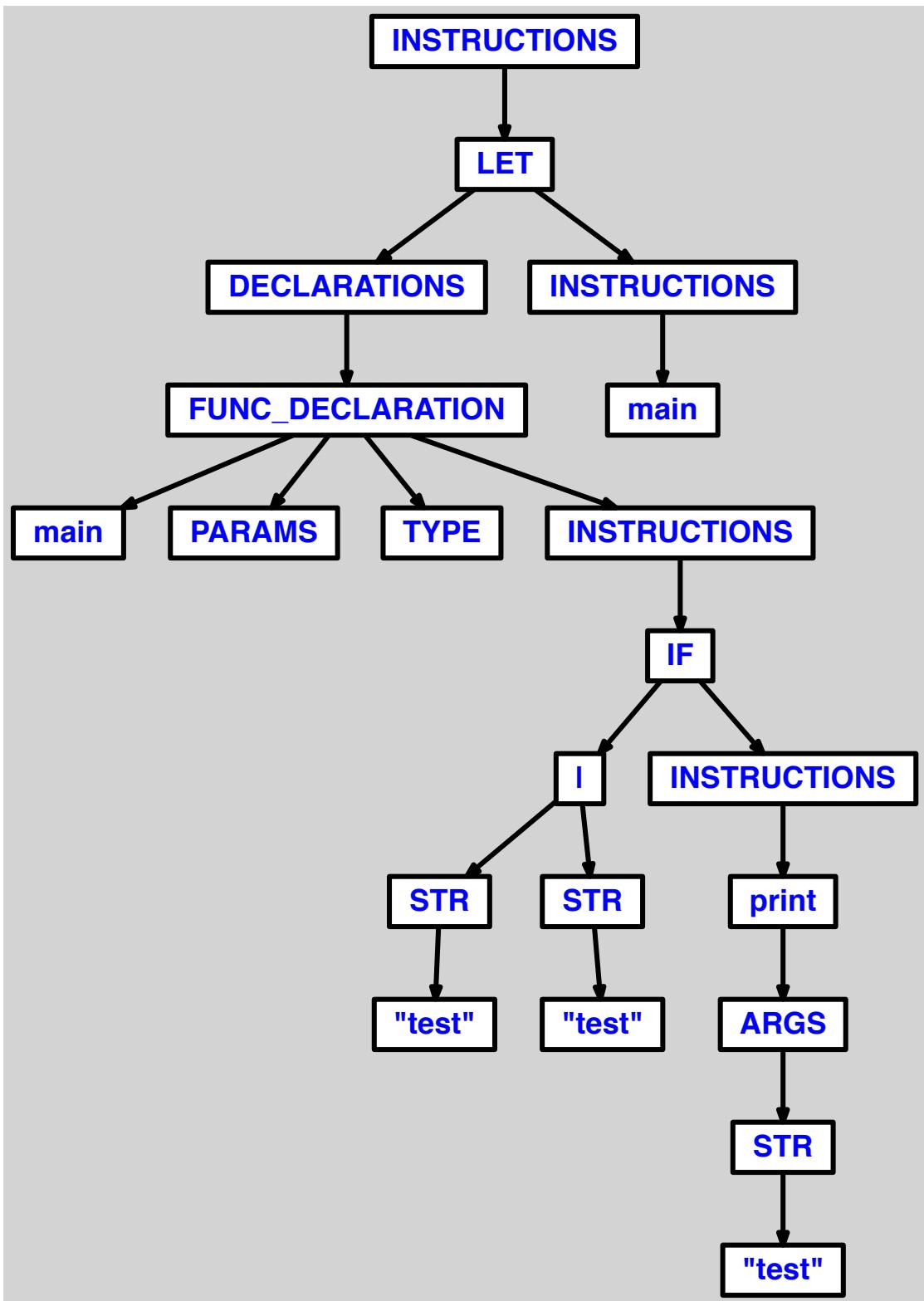
### 11.2.5 <|> avec des entiers

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



### 11.2.6 <|> avec des chaines

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

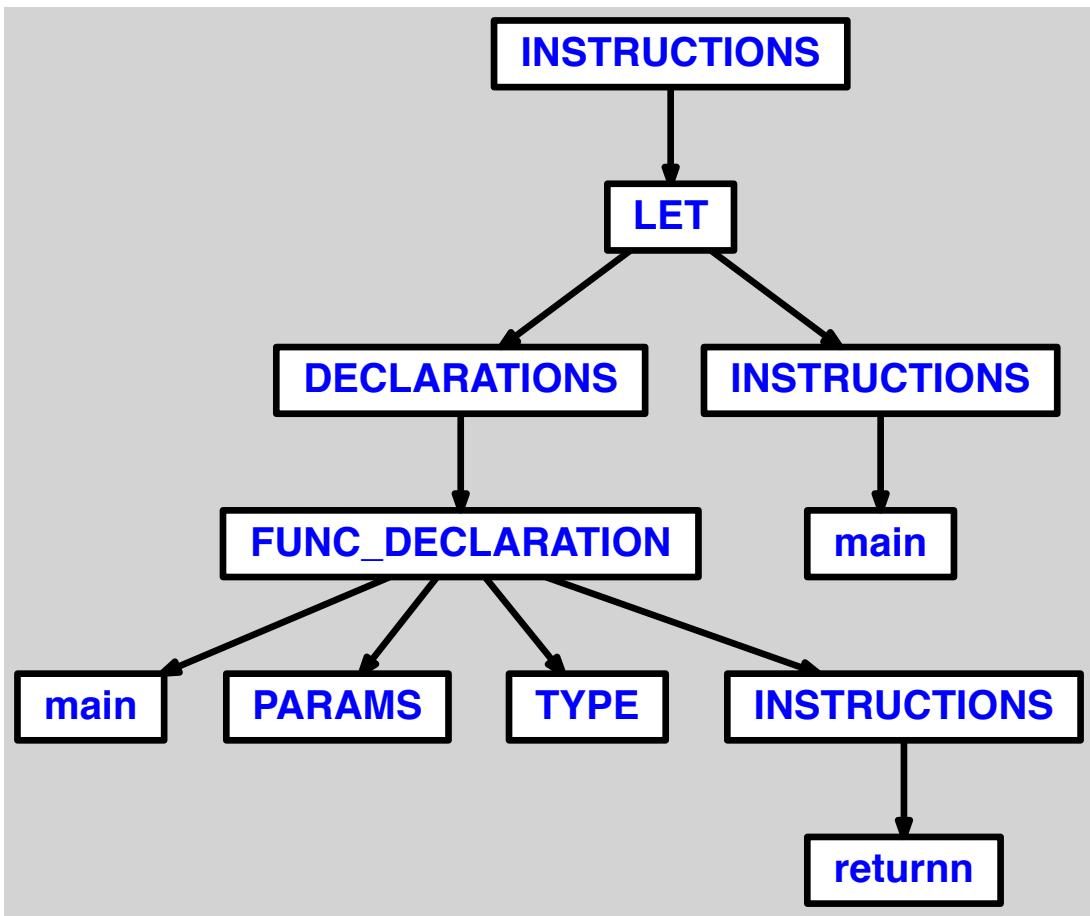


## 12 return

### 12.1 KO

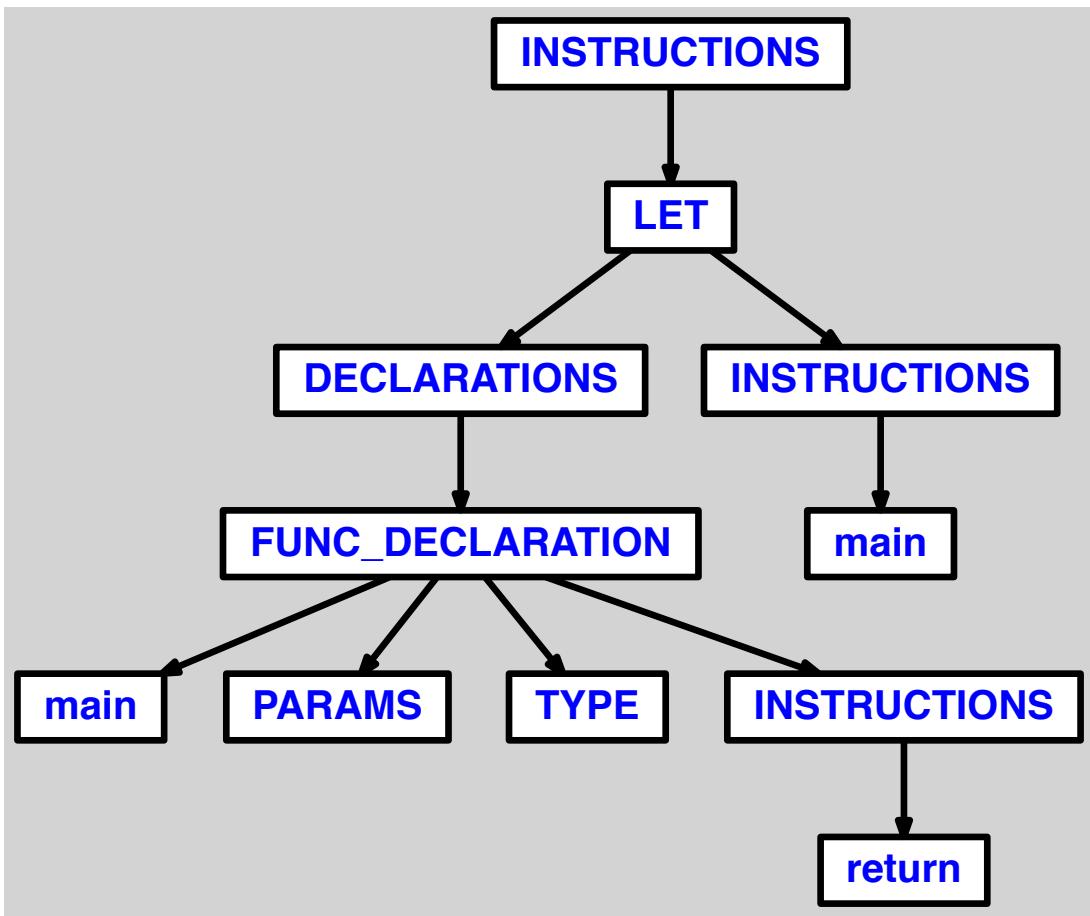
#### 12.1.1 <return> mal écrit

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



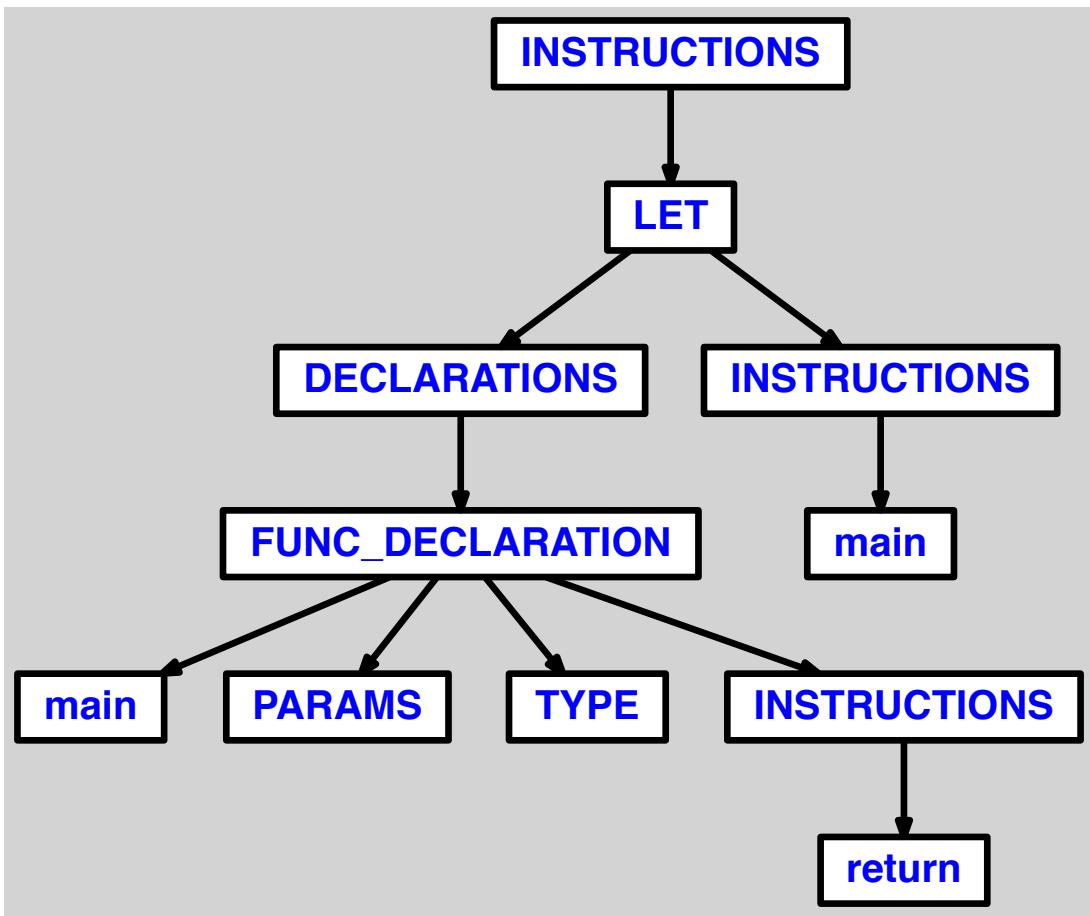
### 12.1.2 <return> sans valeur

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



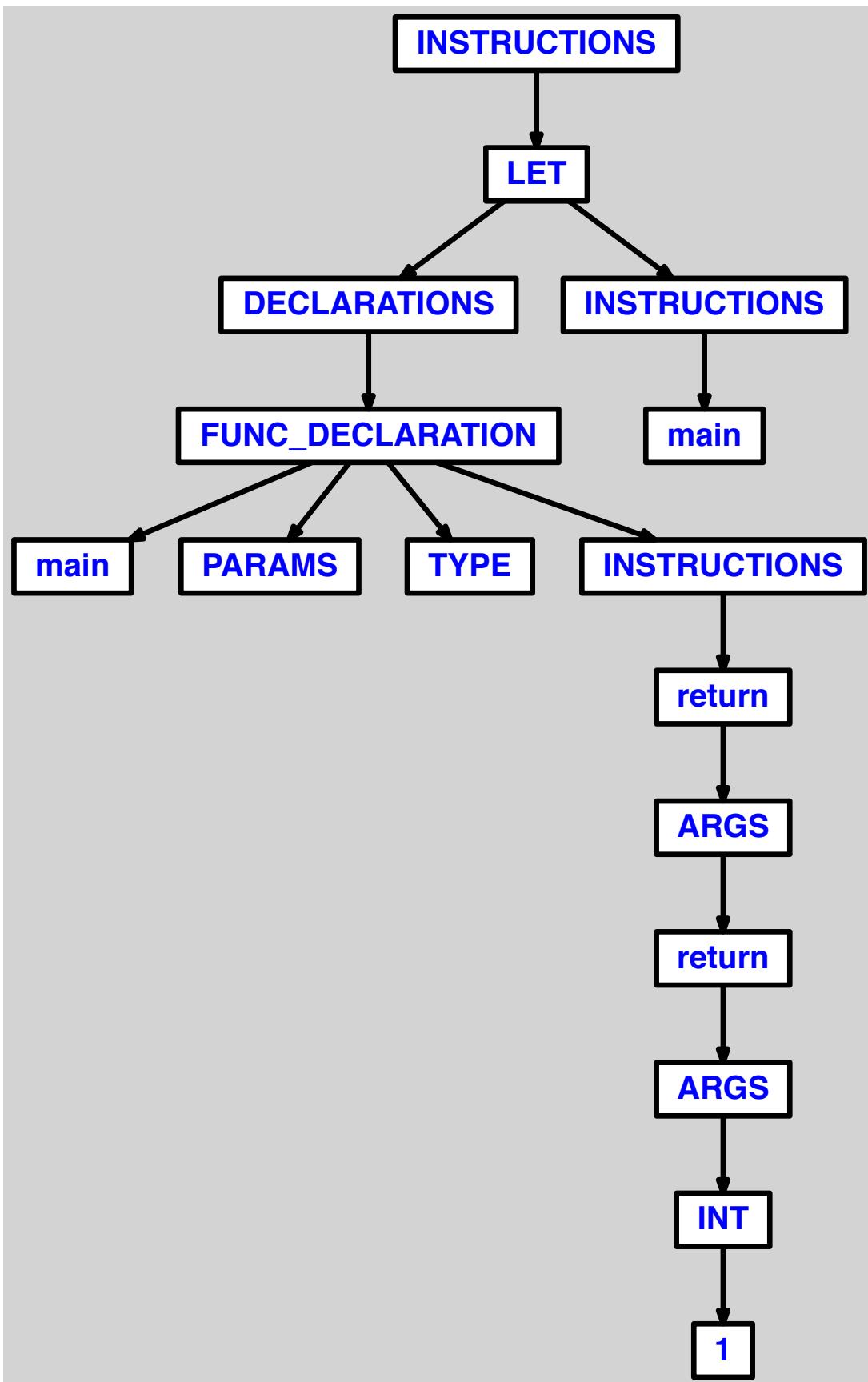
### 12.1.3 <return> avec oubli de parentheses

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



#### 12.1.4 2 <return>

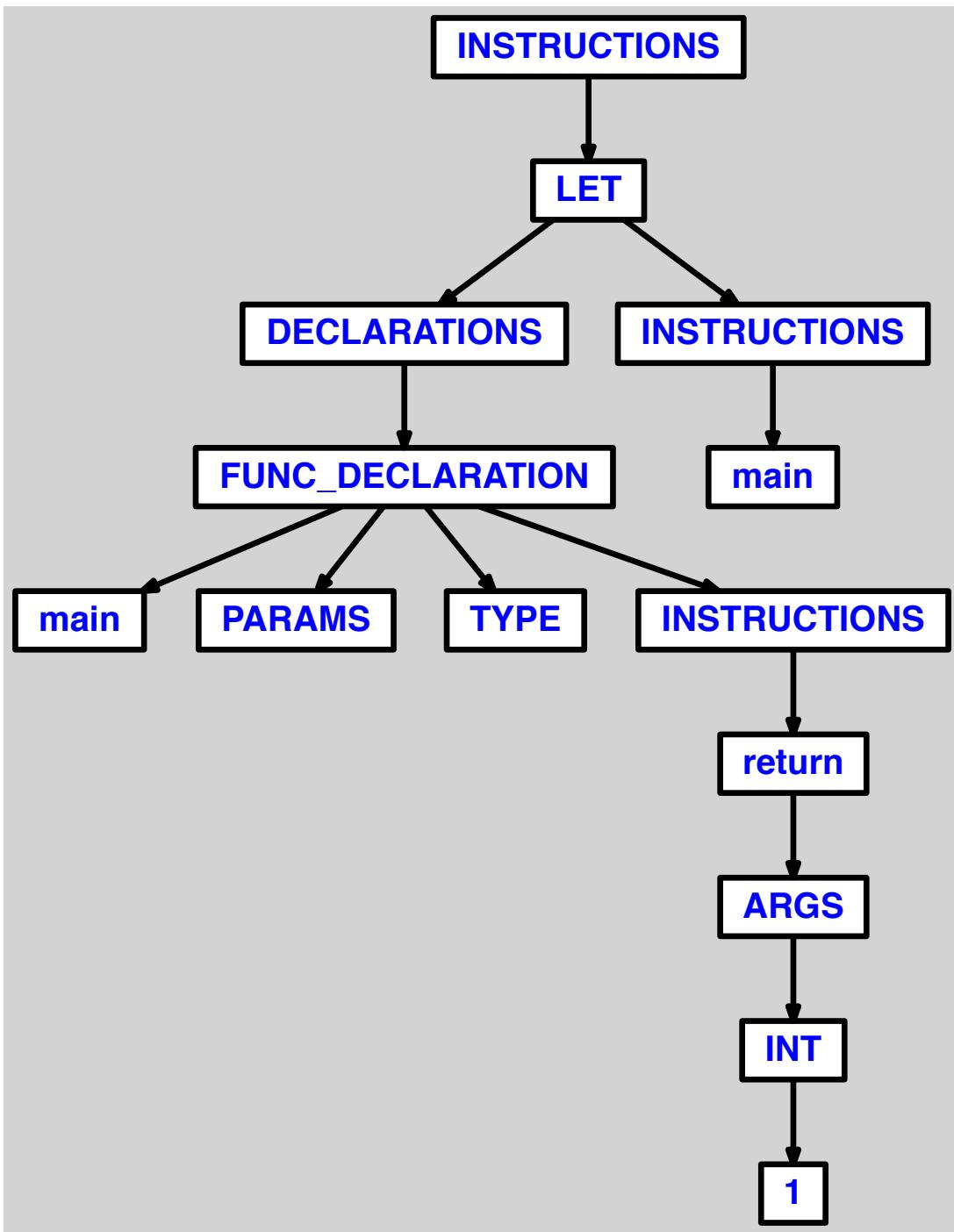
```
1 let
2   function main() = return (return (1))
3 in main() end
```



## 12.2 OK

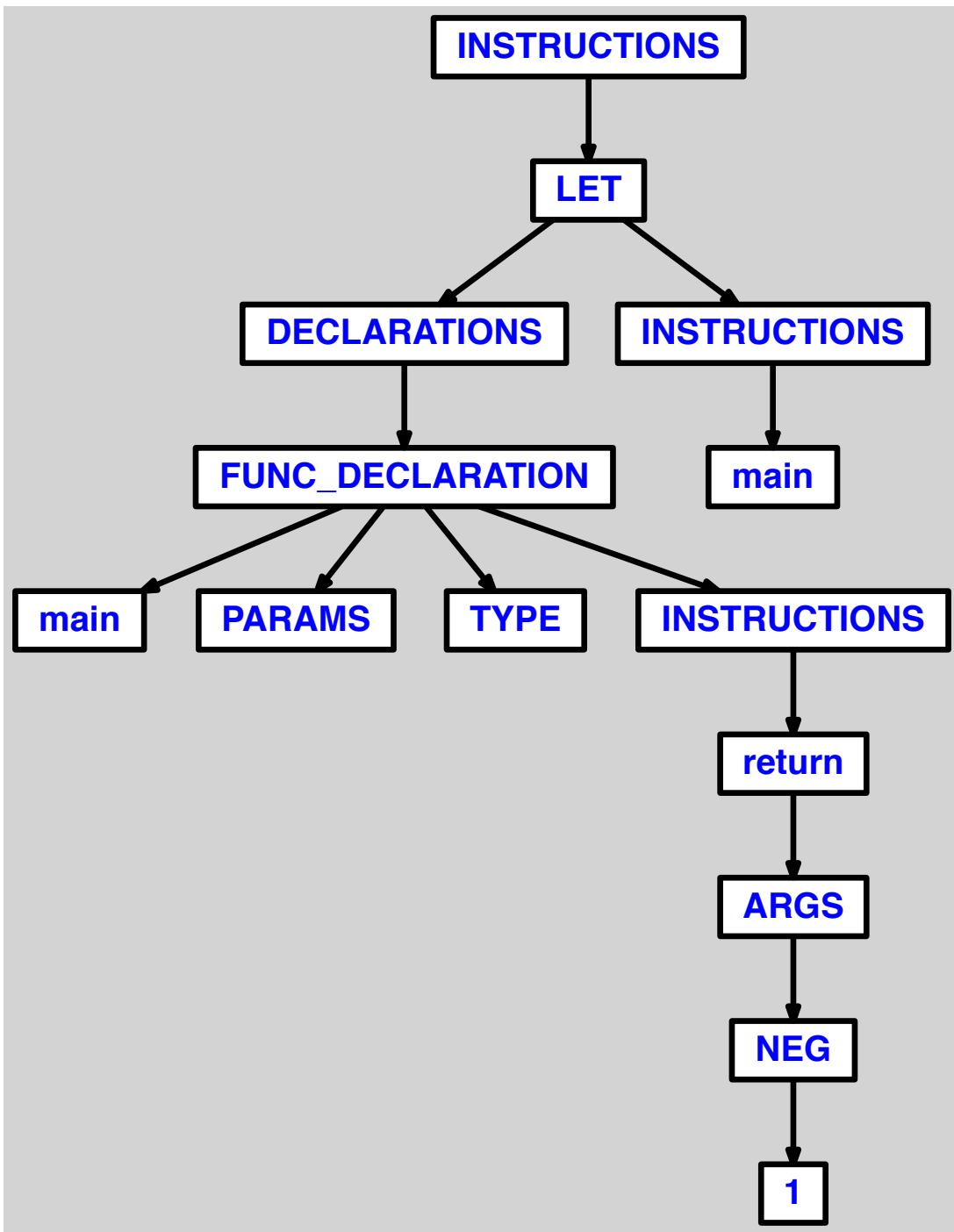
### 12.2.1 retour d'entier positif

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



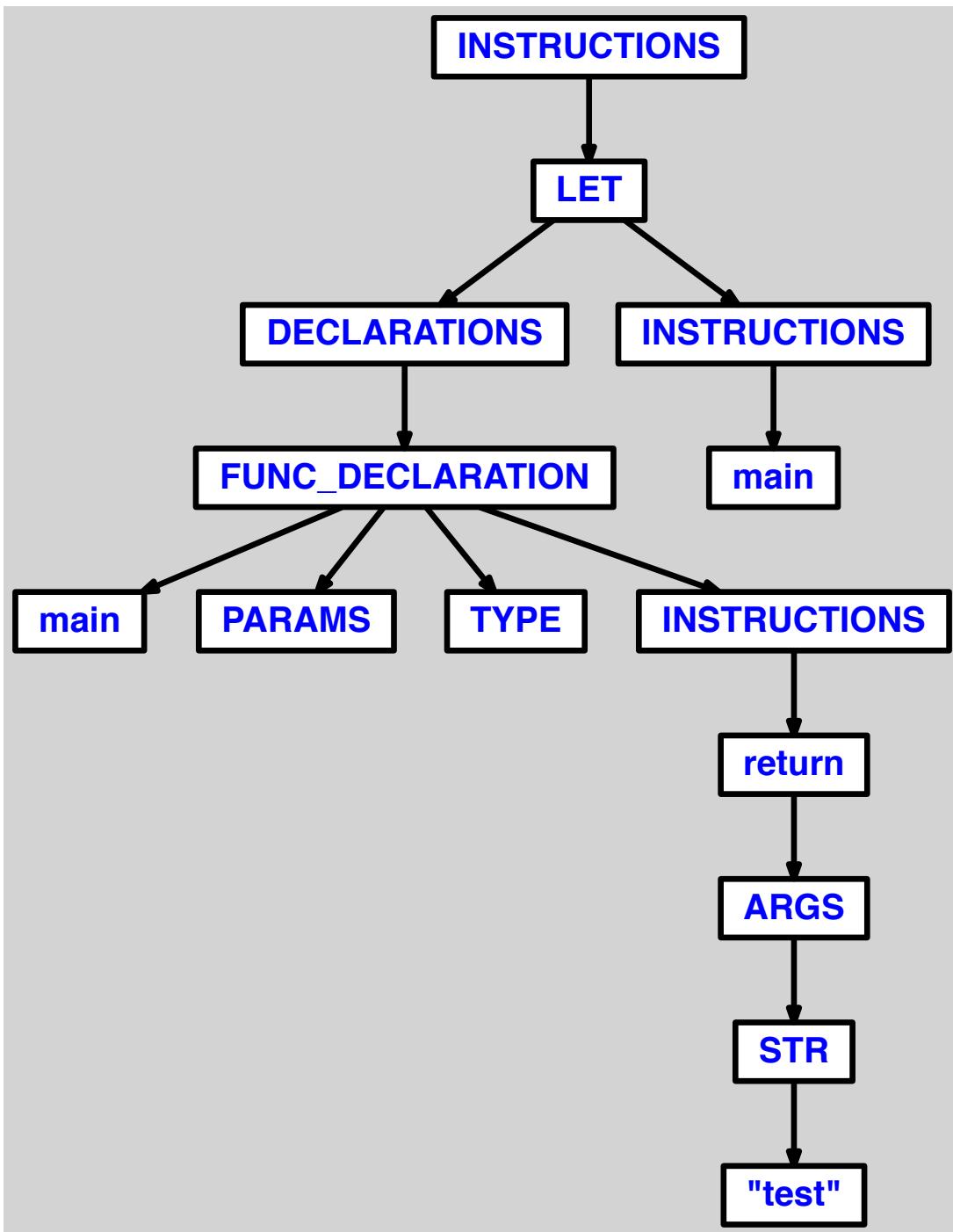
### 12.2.2 retour d'entier negatif

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



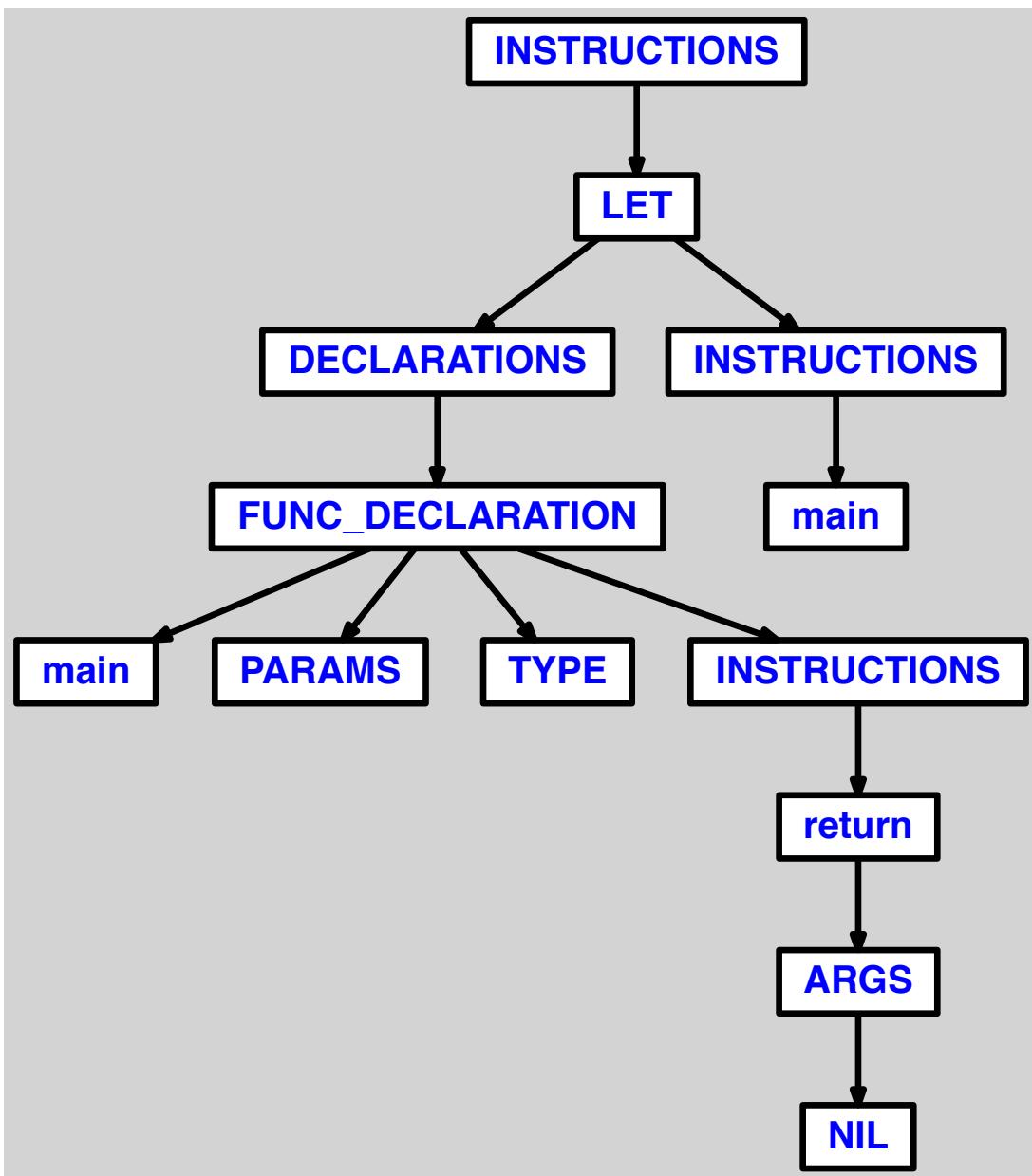
### 12.2.3 retour de chaine

```
1 let
2   function main() = return ("test")
3 in main() end
```



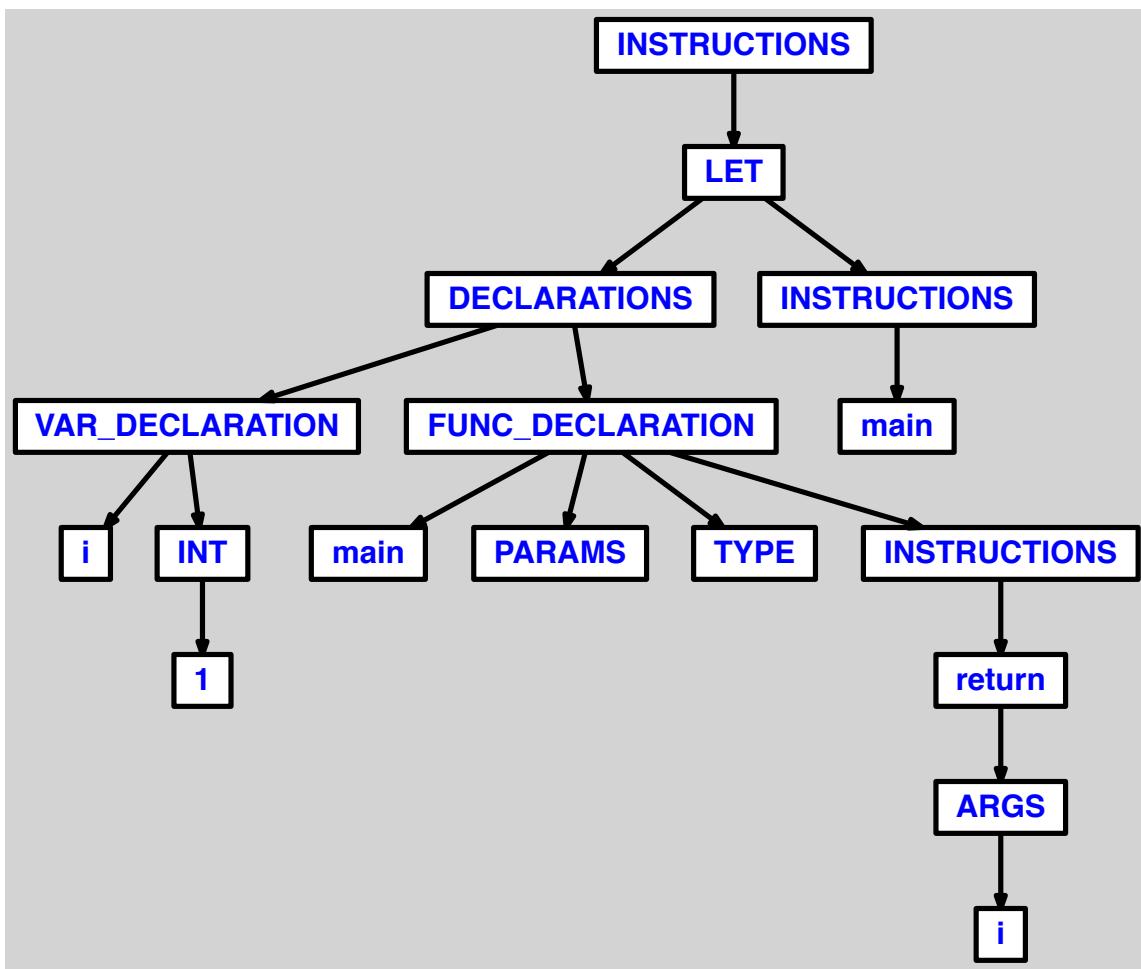
#### 12.2.4 retour de <nil>

```
1 let
2   function main() = return (nil)
3 in main() end
```



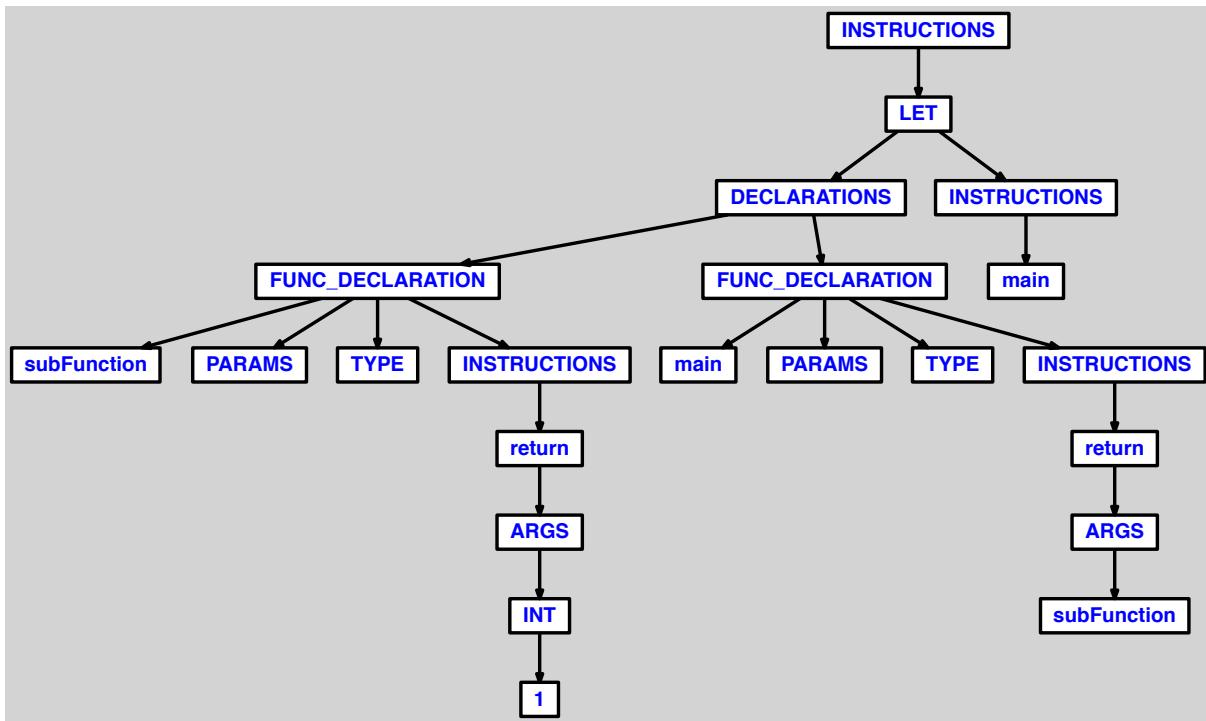
### 12.2.5 retour de variable

```
1 let
2   var i := 1
3
4   function main() = return (i)
5 in main() end
```



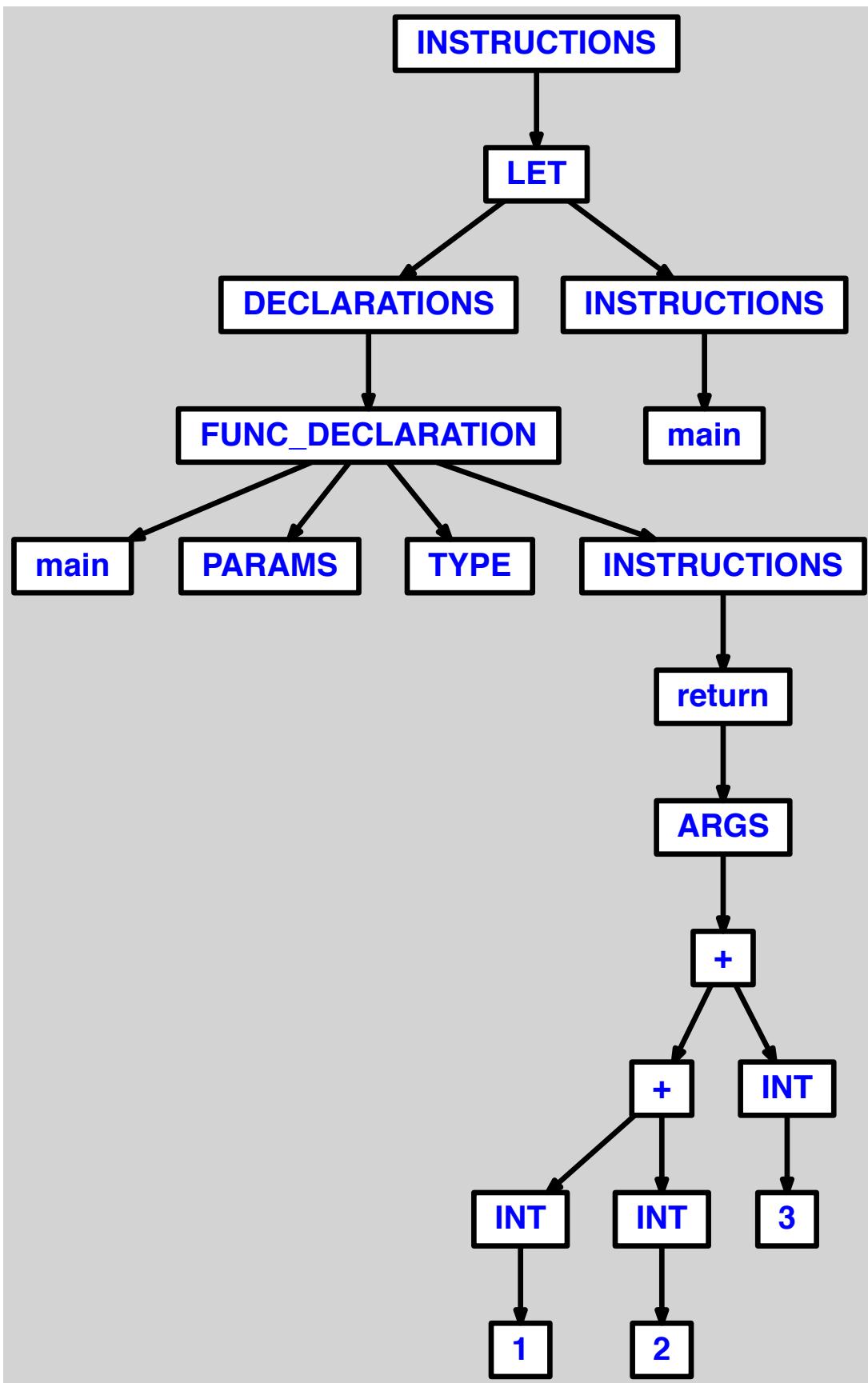
### 12.2.6 retour de fonction

```
1 let
2     function subFunction() = return (1)
3
4     function main() = return (subFunction())
5 in main() end
```



### 12.2.7 retour d'expression

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

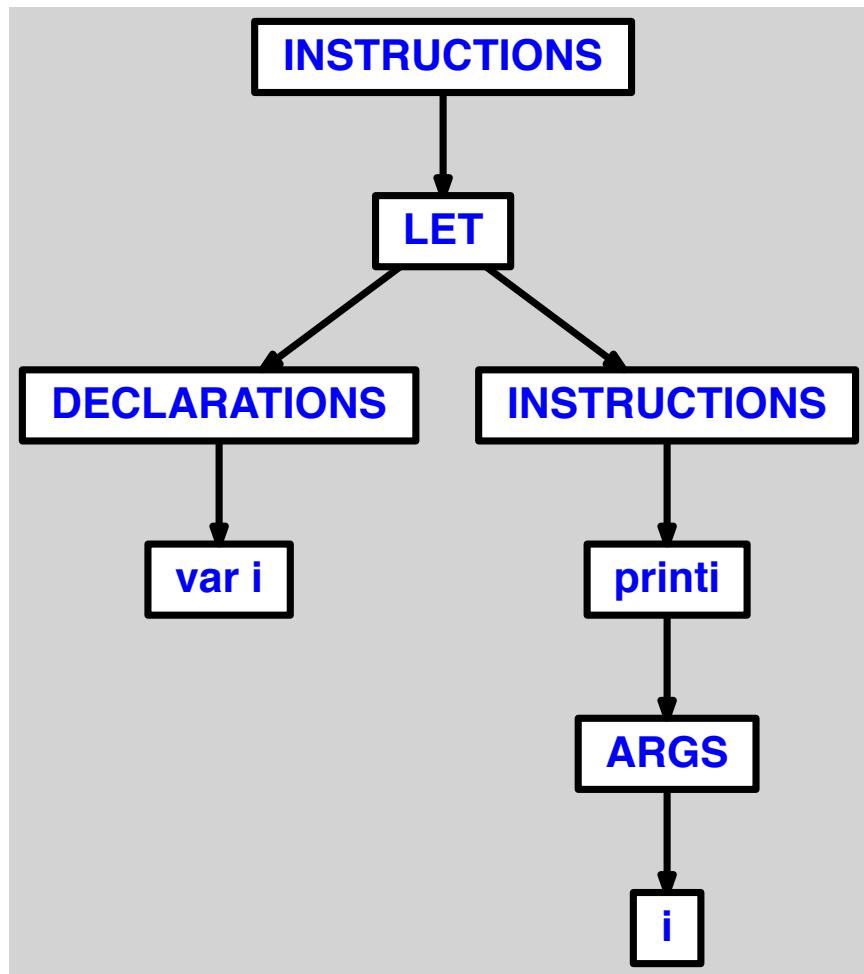


## 13 var

### 13.1 KO

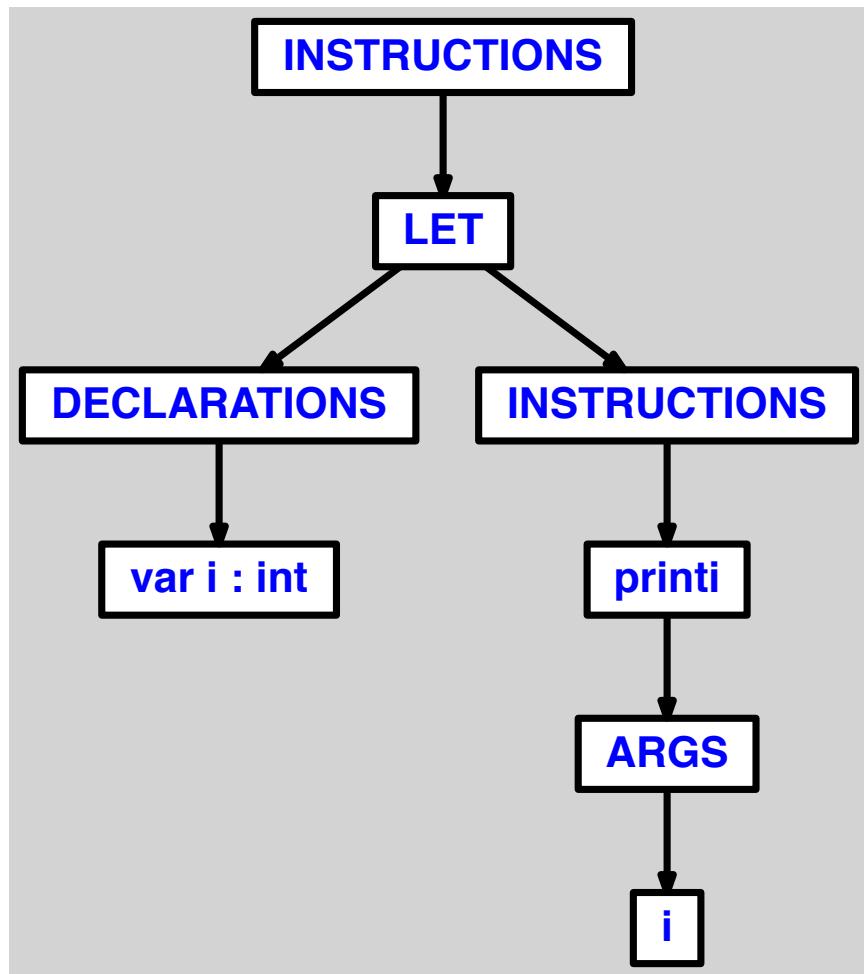
#### 13.1.1 declaration sans affectation

```
1 let
2   var i
3 in
4   printi(i)
5 end
```



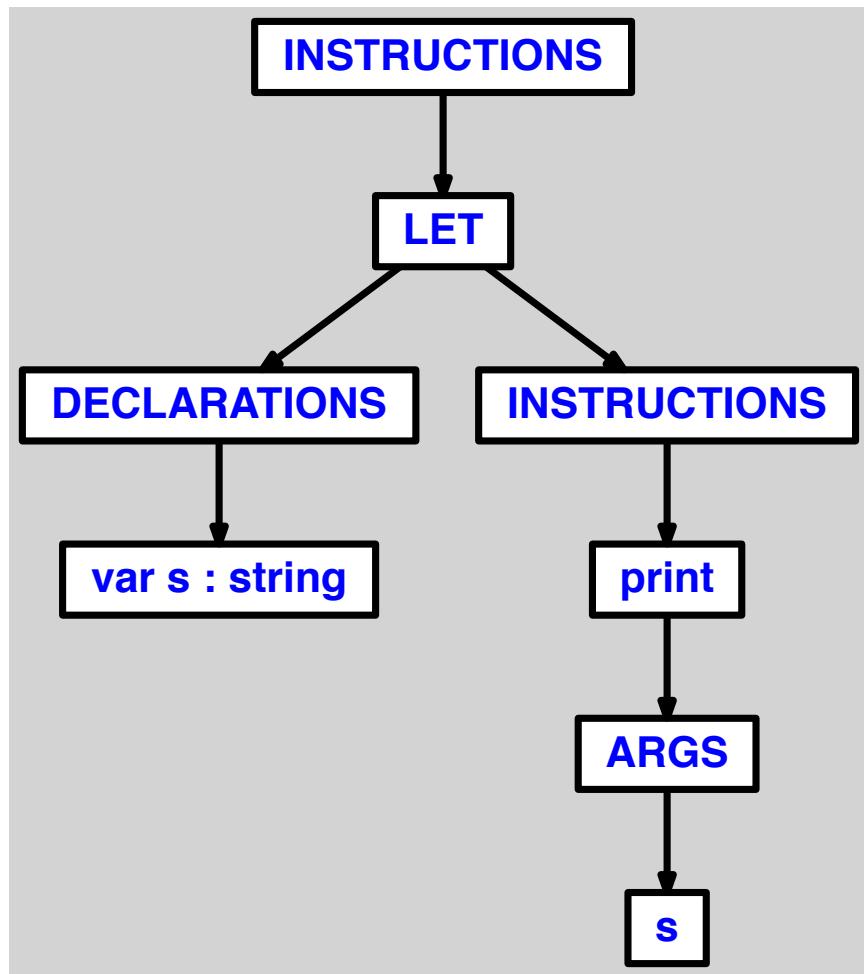
### 13.1.2 déclaration d'entier avec type sans affectation

```
1 let
2   var i : int
3 in
4   printi(i)
5 end
```



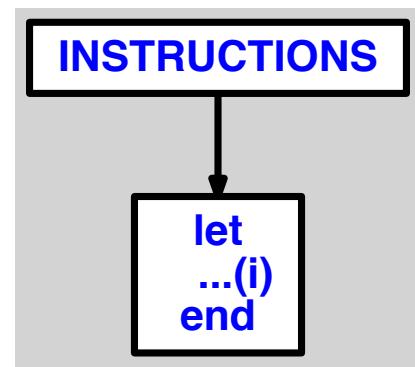
### 13.1.3 déclaration de chaîne avec type sans affectation

```
1 let
2   var s : string
3 in
4   print(s)
5 end
```



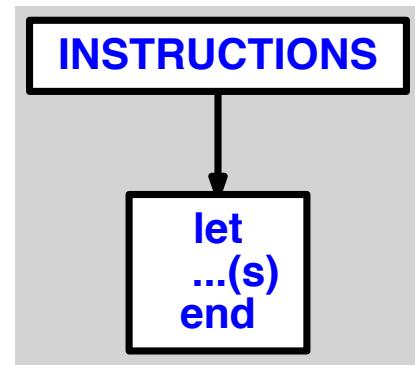
### 13.1.4 affectation d'entier sans declaration

```
1 let
2   i := 1
3 in
4   printi(i)
5 end
```



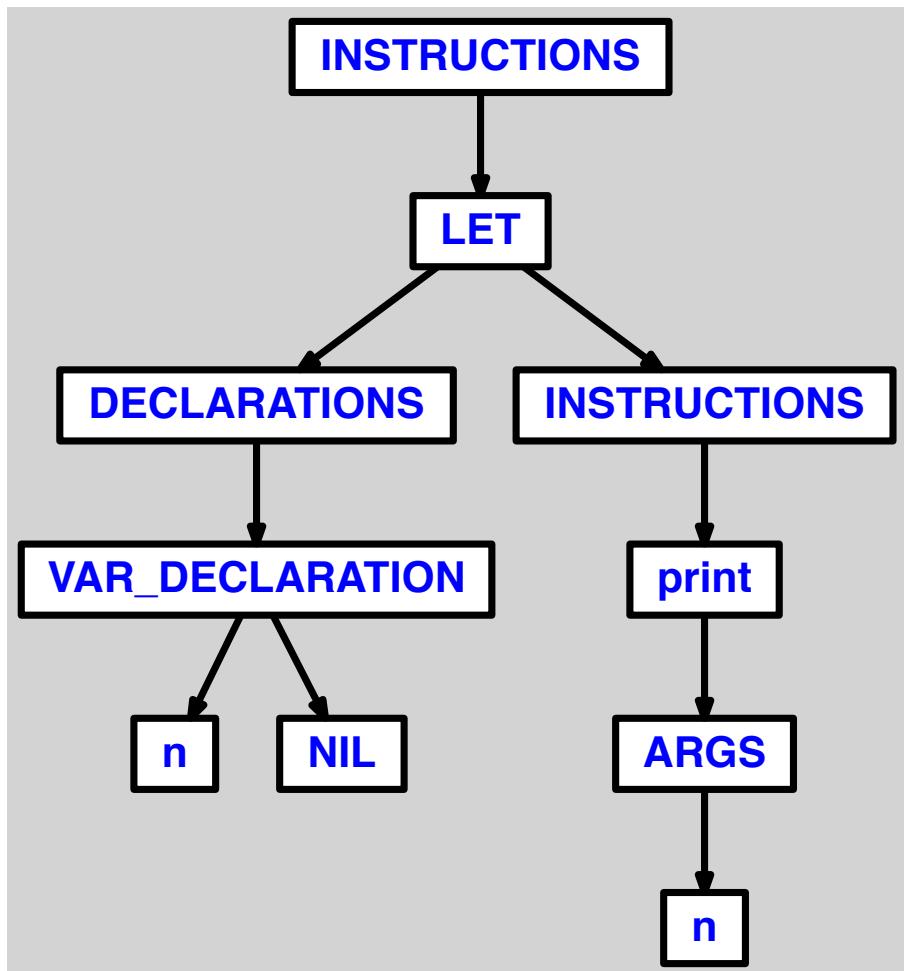
### 13.1.5 affectation de chaine sans declaration

```
1 let
2   s := "test"
3 in
4   print(s)
5 end
```



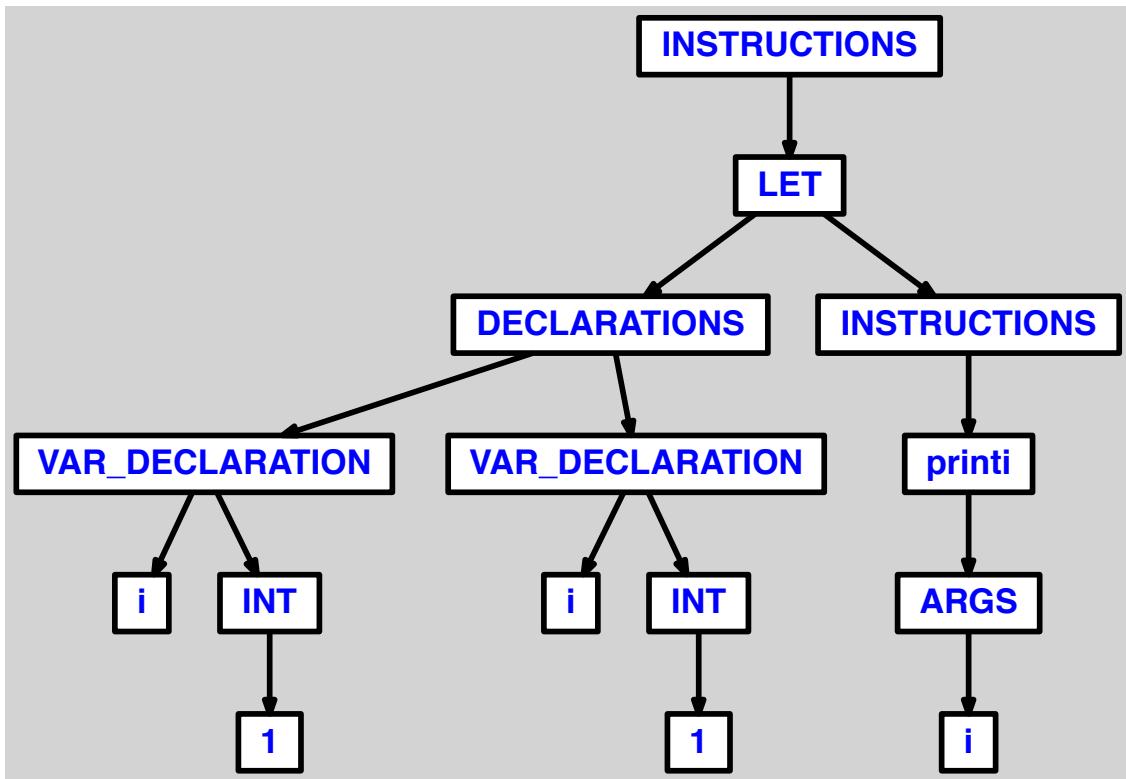
### 13.1.6 affectation de <nil>

```
1 let
2   var n := nil
3 in
4   print(n)
5 end
```



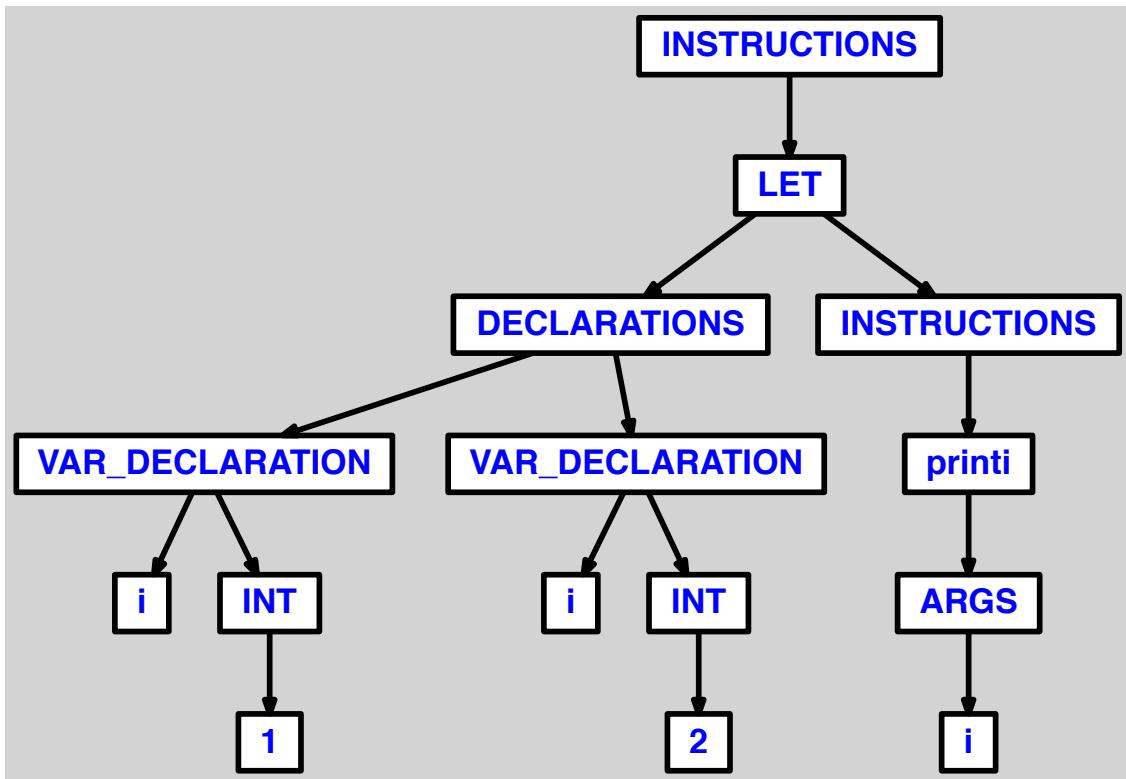
### 13.1.7 double-declaration avec valeurs égales

```
1 let
2   var i := 1
3   var i := 1
4 in
5   printi(i)
6 end
```



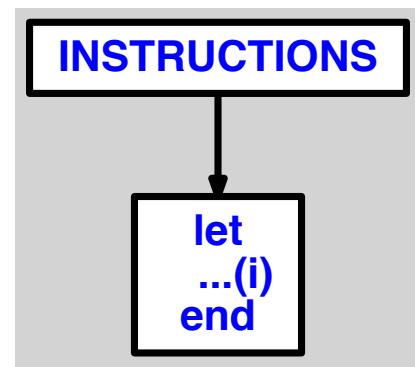
### 13.1.8 double-declaration avec valeurs differentes

```
1 let
2   var i := 1
3   var i := 2
4 in
5   printi(i)
6 end
```



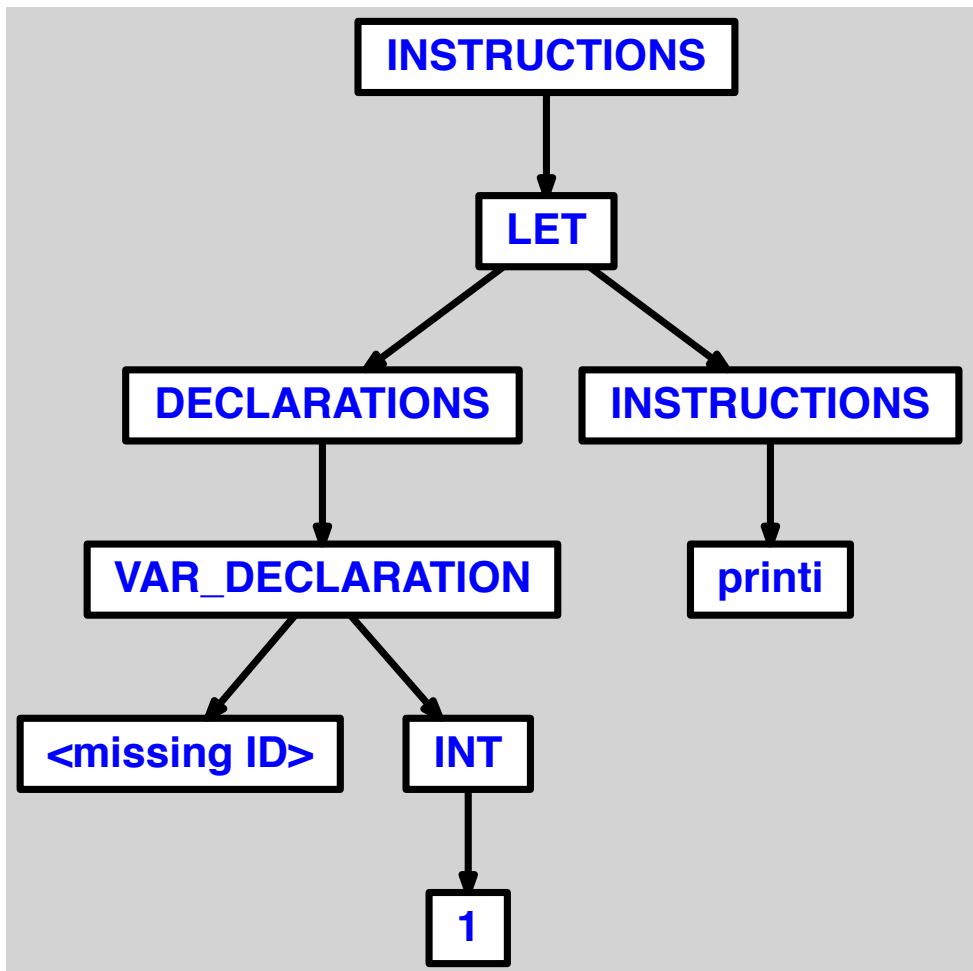
### 13.1.9 <var> mal écrit

```
1 let
2   varr i := 1
3 in
4   printi(i)
5 end
```



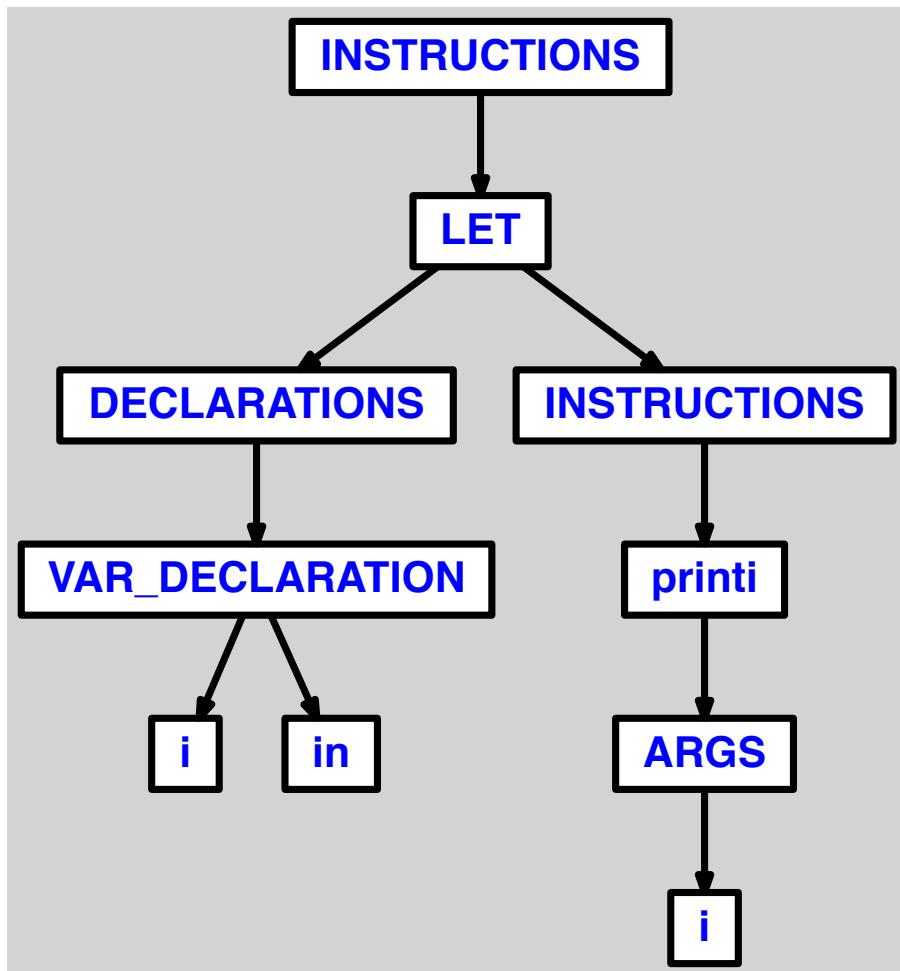
### 13.1.10 declaration de caractere special

```
1 let
2   var \ := 1
3 in
4   printi(\)
5 end
```



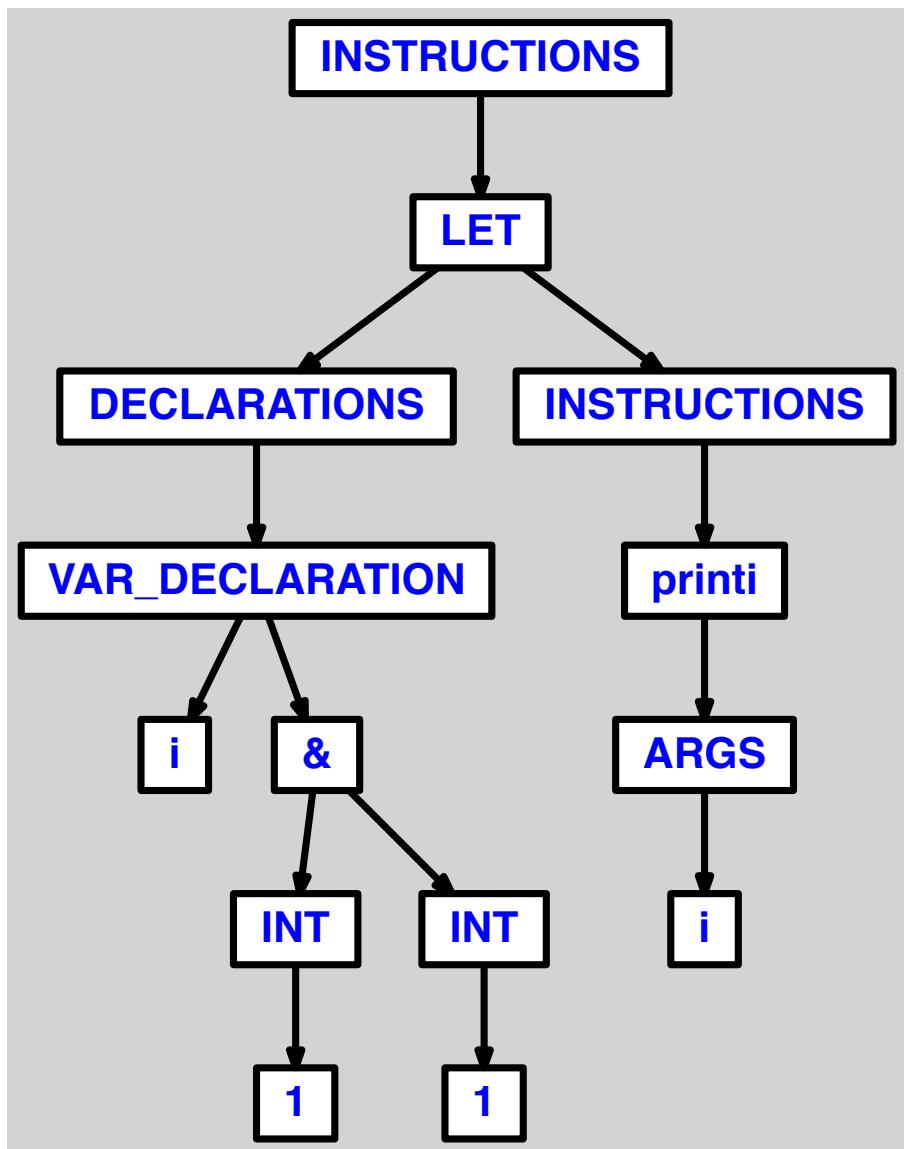
### 13.1.11 affectation de caractere special

```
1 let
2   var i := \
3 in
4   printi(i)
5 end
```



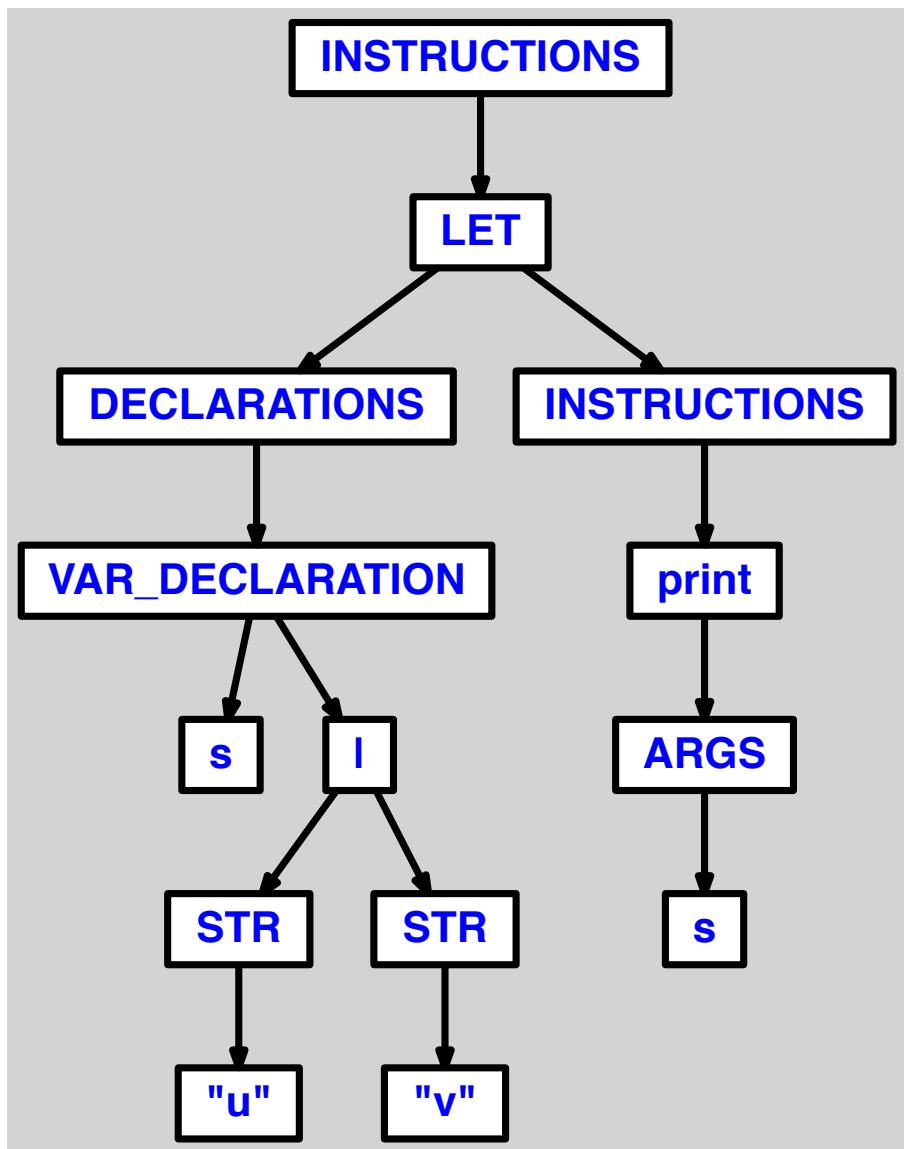
### 13.1.12 affectation d'opération logique sur entiers

```
1 let
2   var i := 1 & 1
3 in
4   printi(i)
5 end
```



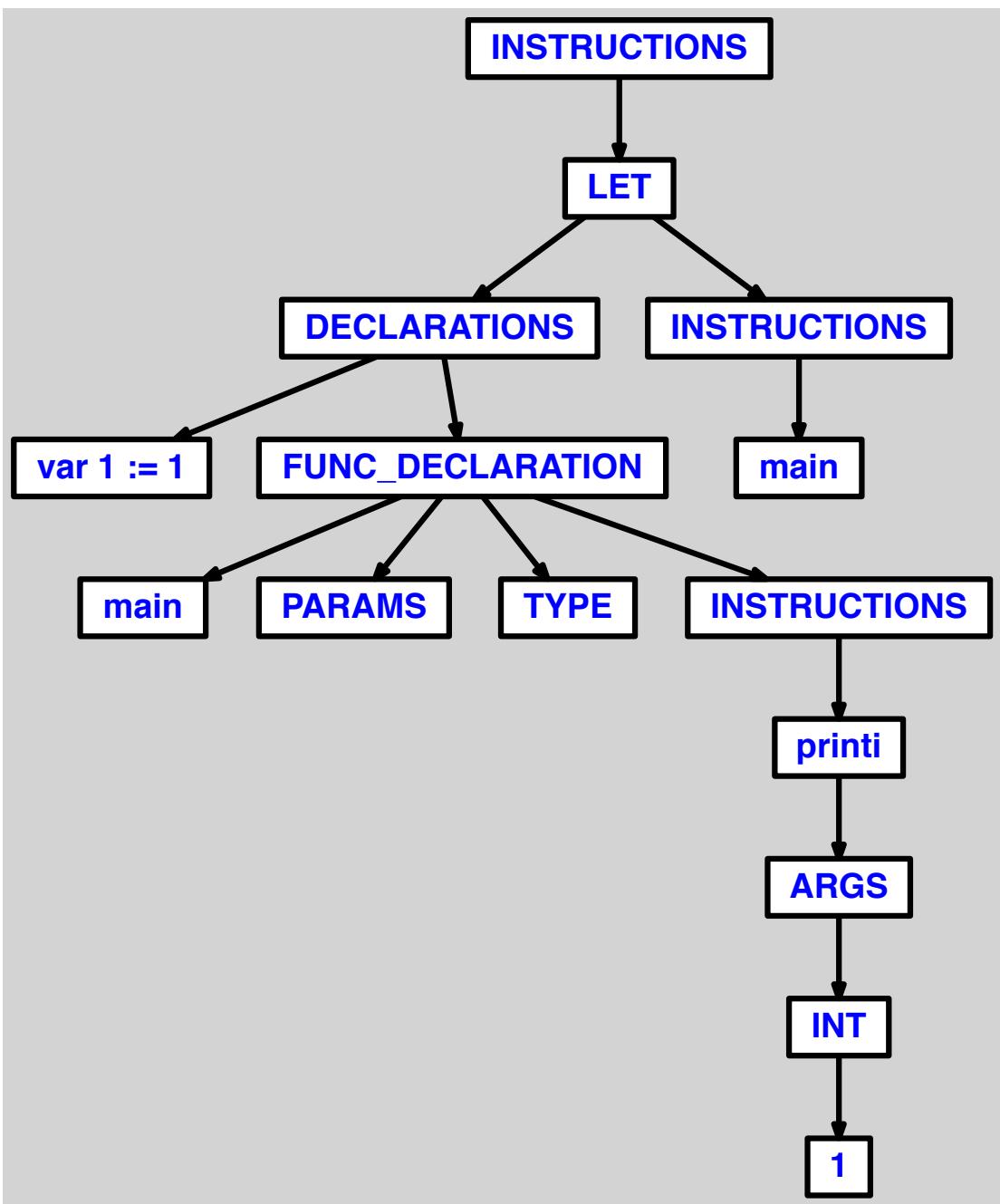
### 13.1.13 affectation d'opération logique sur chaînes

```
1 let
2   var s := "u" | "v"
3 in
4   print(s)
5 end
```



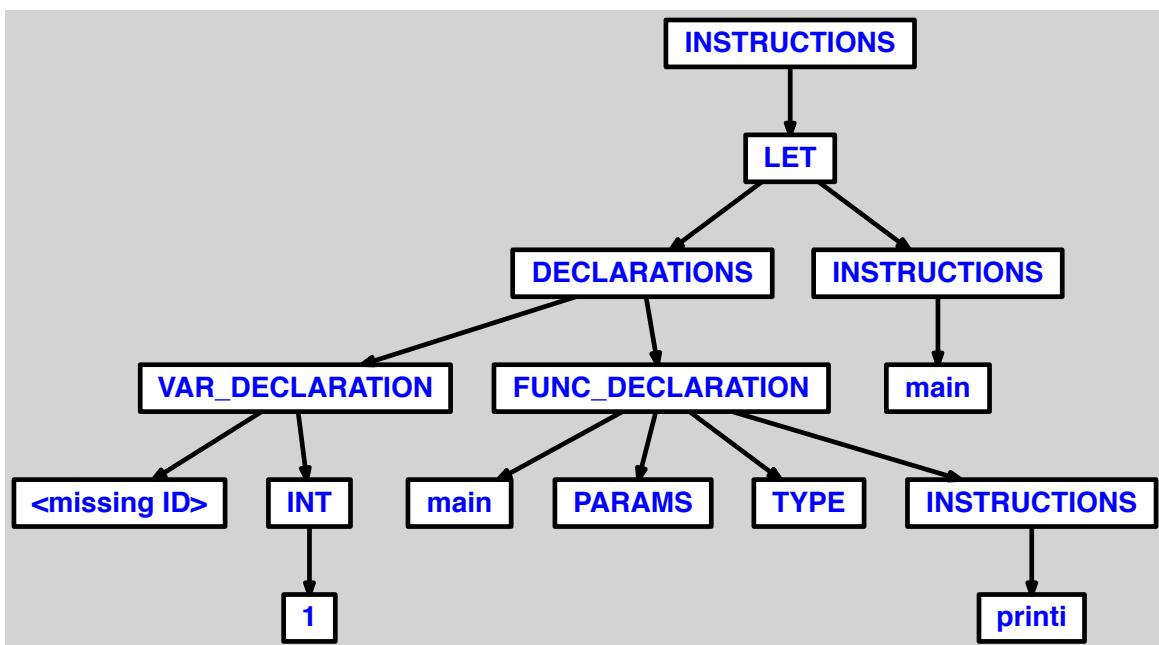
### 13.1.14 chiffre comme nom de variable

```
1 let
2   var 1 := 1
3
4   function main() = printi(1)
5 in main() end
```



### 13.1.15 <-> comme nom de variable

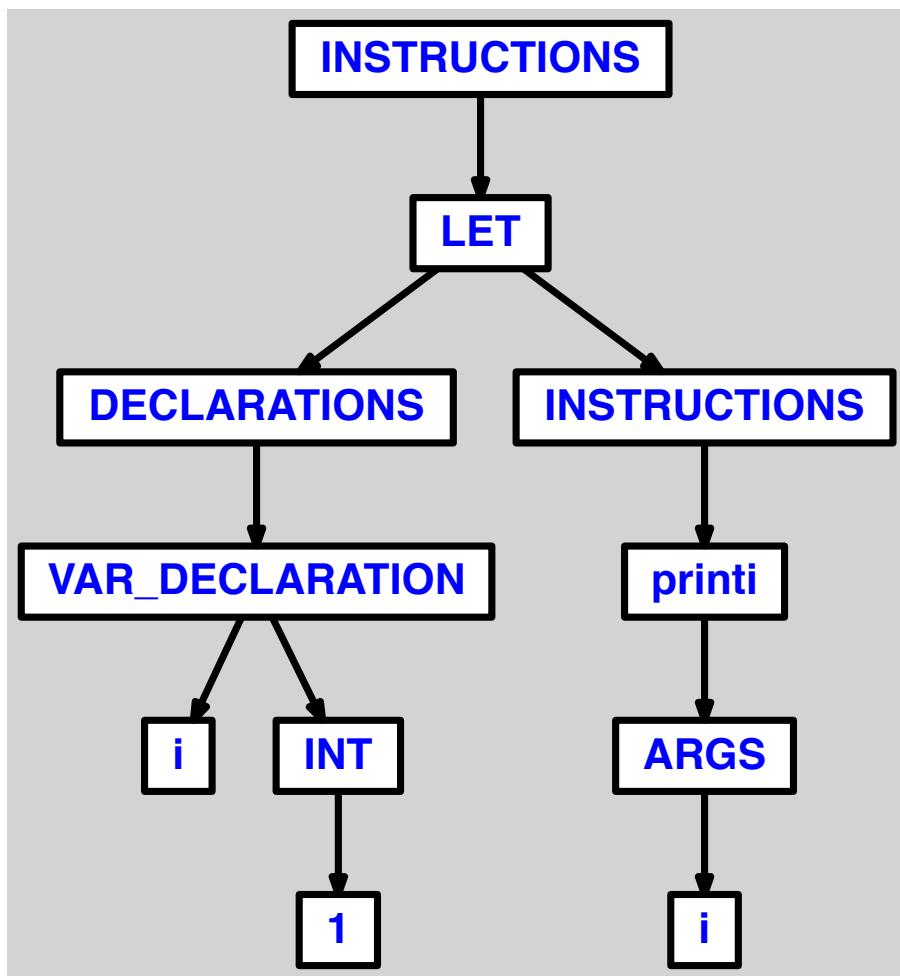
```
1 let
2   var _ := 1
3
4   function main() = printi(_)
5 in main() end
```



## 13.2 OK

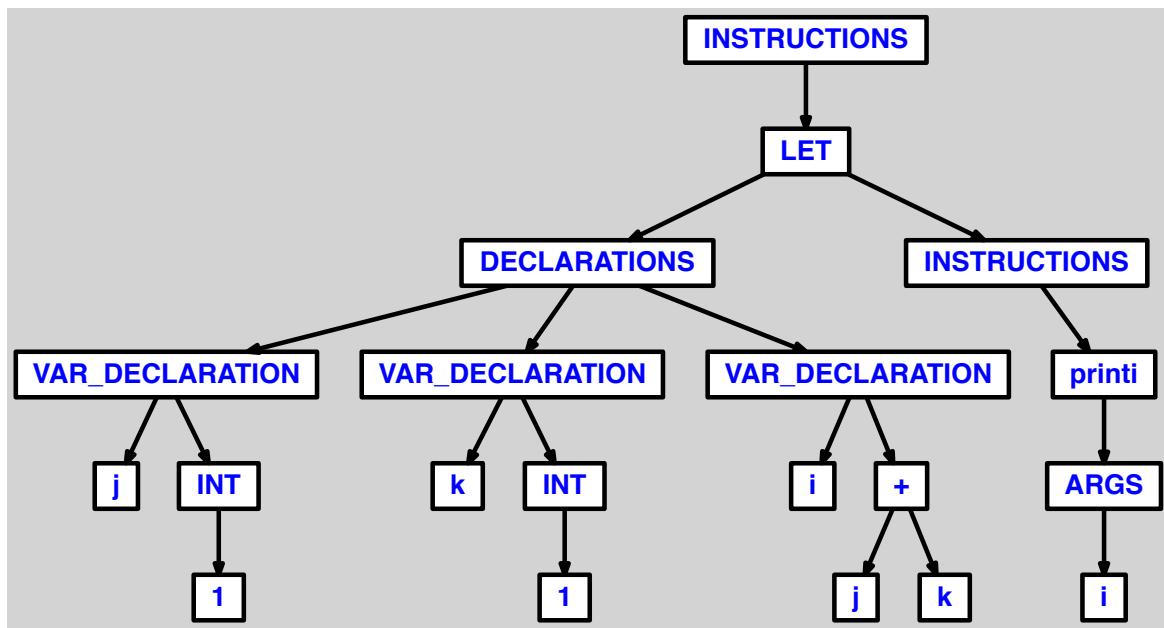
### 13.2.1 declaration d'entier simple

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



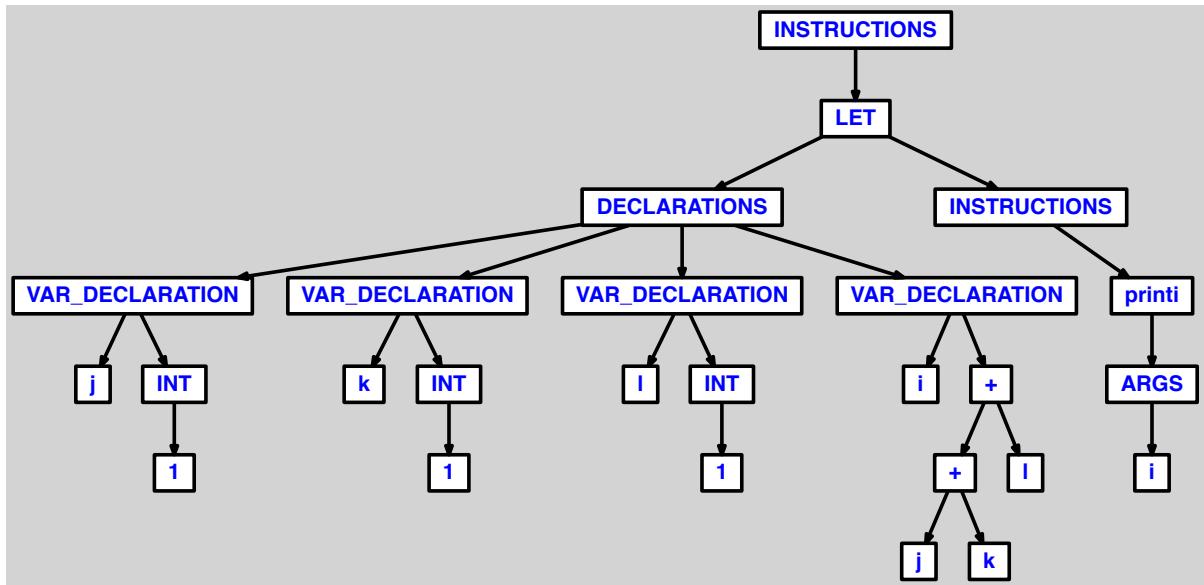
### 13.2.2 déclarations d'entier avec reutilisation de 2 autres entiers

```
1 let
2   var j := 1
3   var k := 1
4   var i := j+k
5 in
6   printi(i)
7 end
```



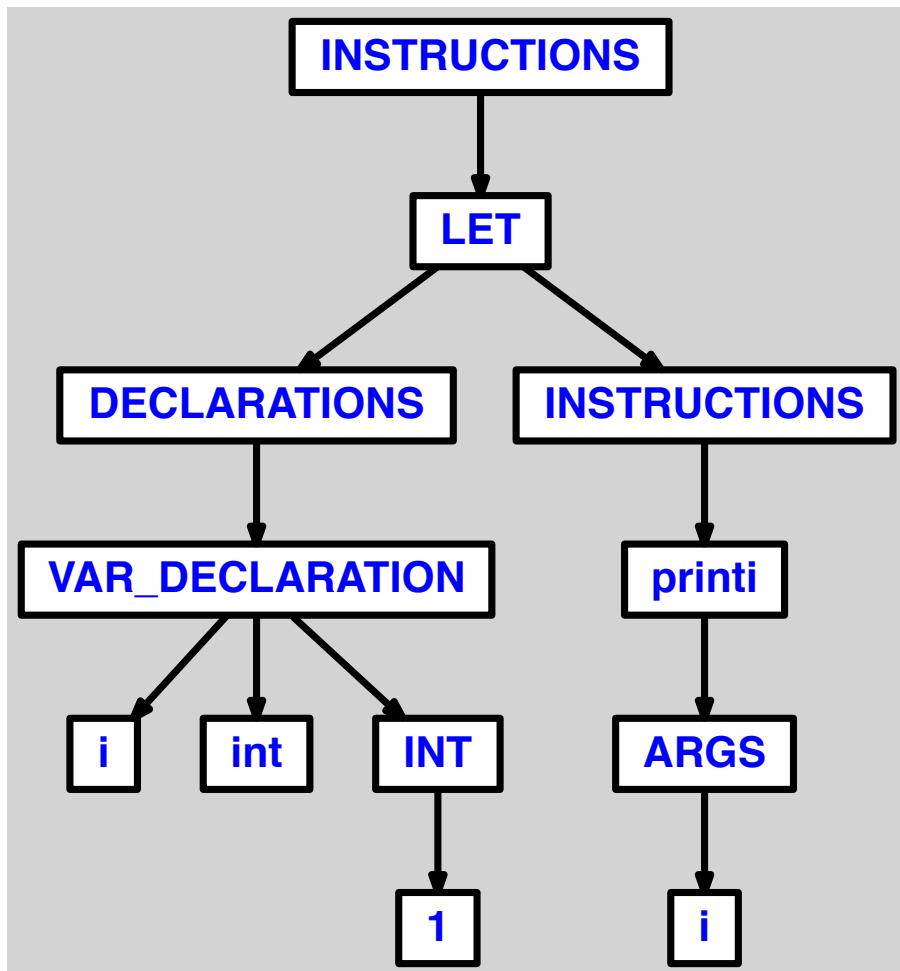
### 13.2.3 déclarations d'entier avec reutilisation de 3 autres entiers

```
1 let
2   var j := 1
3   var k := 1
4   var l := 1
5   var i := j+k+l
6 in
7   printi(i)
8 end
```



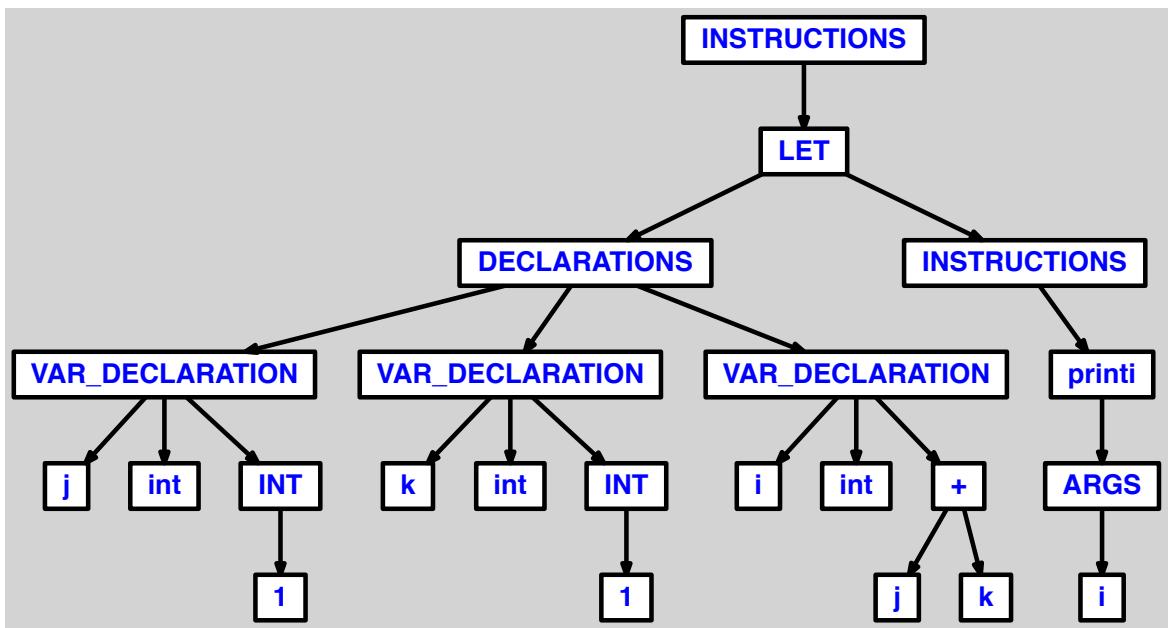
#### 13.2.4 déclaration simple d'entier avec type

```
1 let
2   var i : int := 1
3 in
4   printi(i)
5 end
```



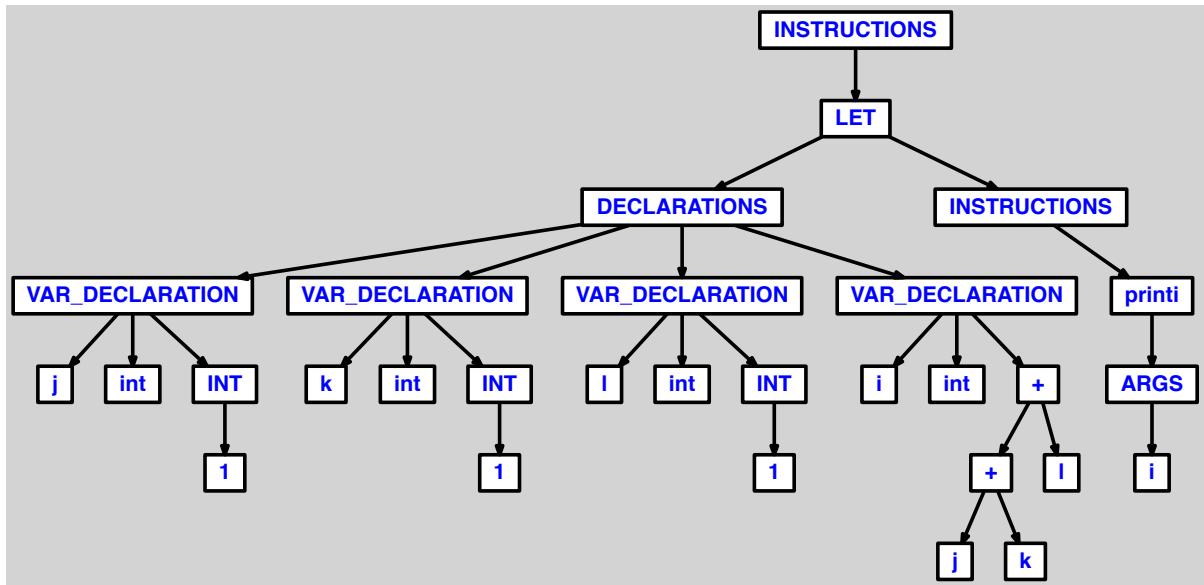
### 13.2.5 declarations d'entier avec types et reutilisation de 2 autres entiers

```
1 let
2     var j : int := 1
3     var k : int := 1
4     var i : int := j+k
5 in
6     printi(i)
7 end
```



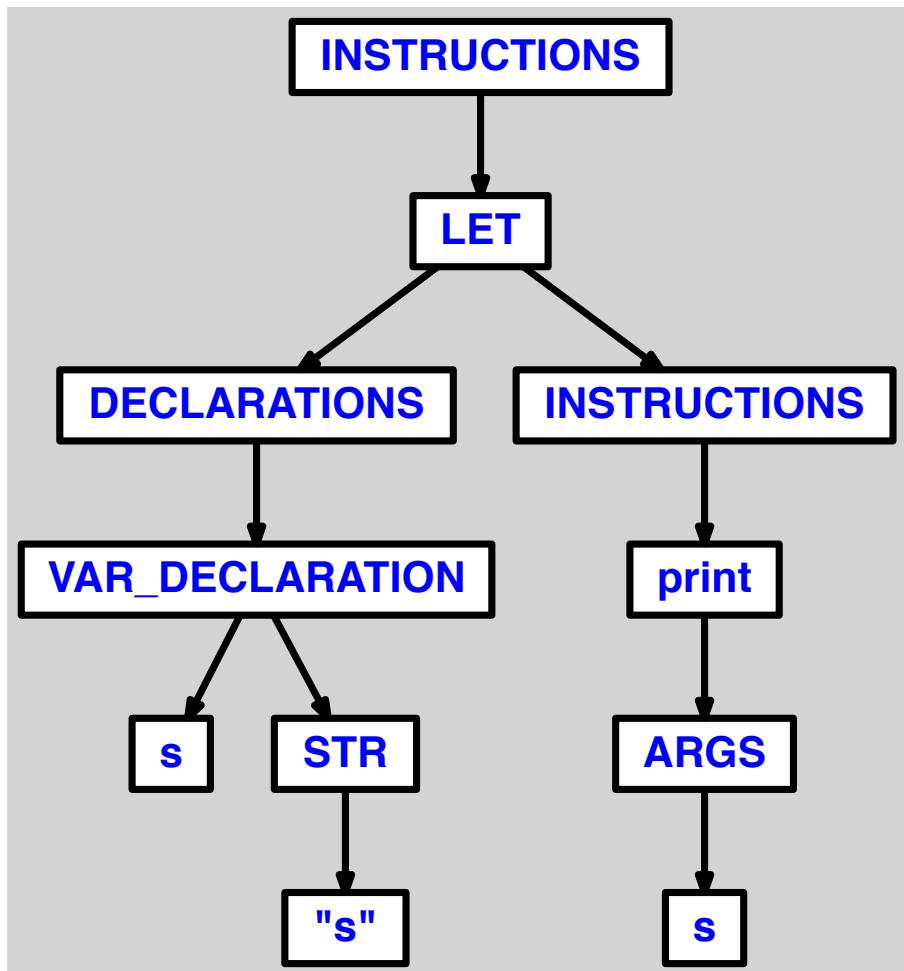
### 13.2.6 declarations d'entier avec types et reutilisation de 3 autres entiers

```
1 let
2     var j : int := 1
3     var k : int := 1
4     var l : int := 1
5     var i : int := j+k+l
6 in
7     printi(i)
8 end
```



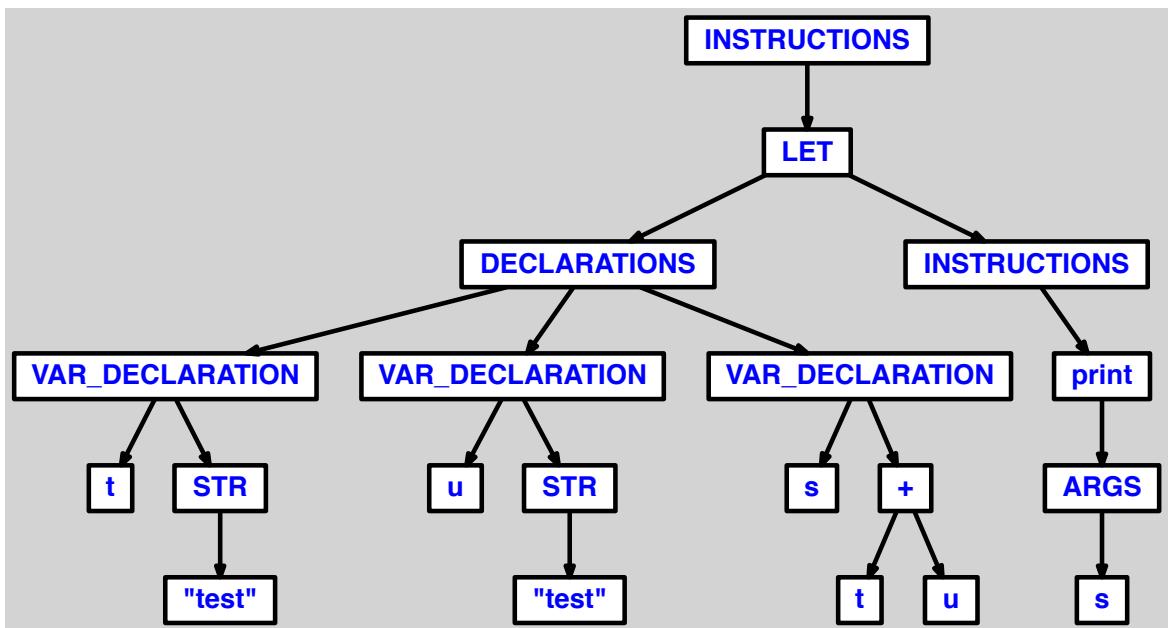
### 13.2.7 déclaration de chaîne simple

```
1 let
2   var s := "s"
3 in
4   print(s)
5 end
```



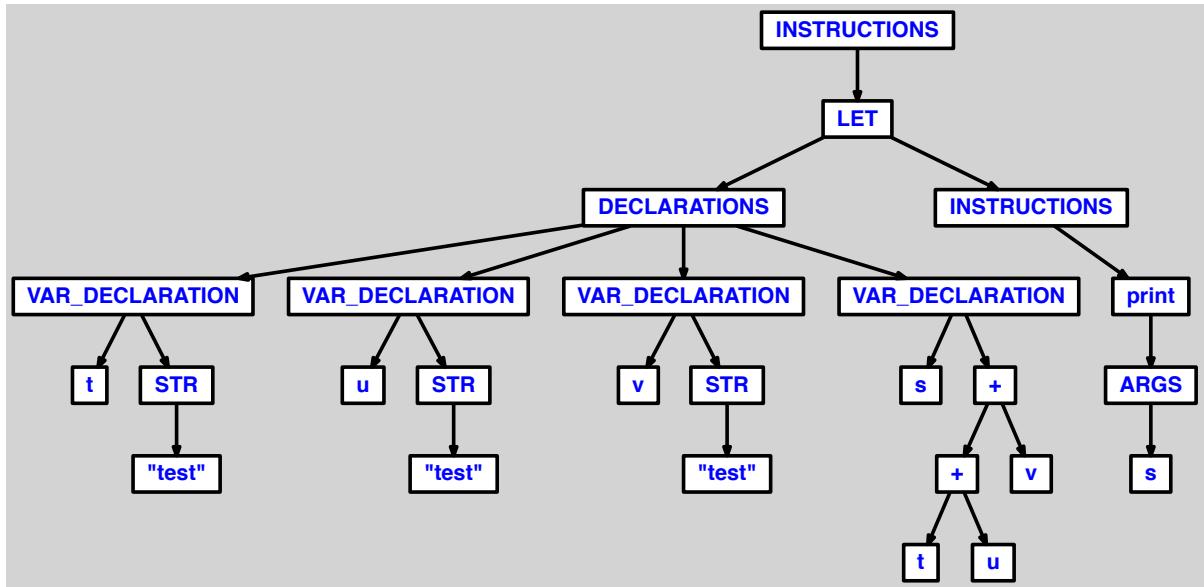
### 13.2.8 déclarations de chaîne avec réutilisation de 2 autres chaînes

```
1 let
2   var t := "test"
3   var u := "test"
4   var s := t+u
5 in
6   print(s)
7 end
```



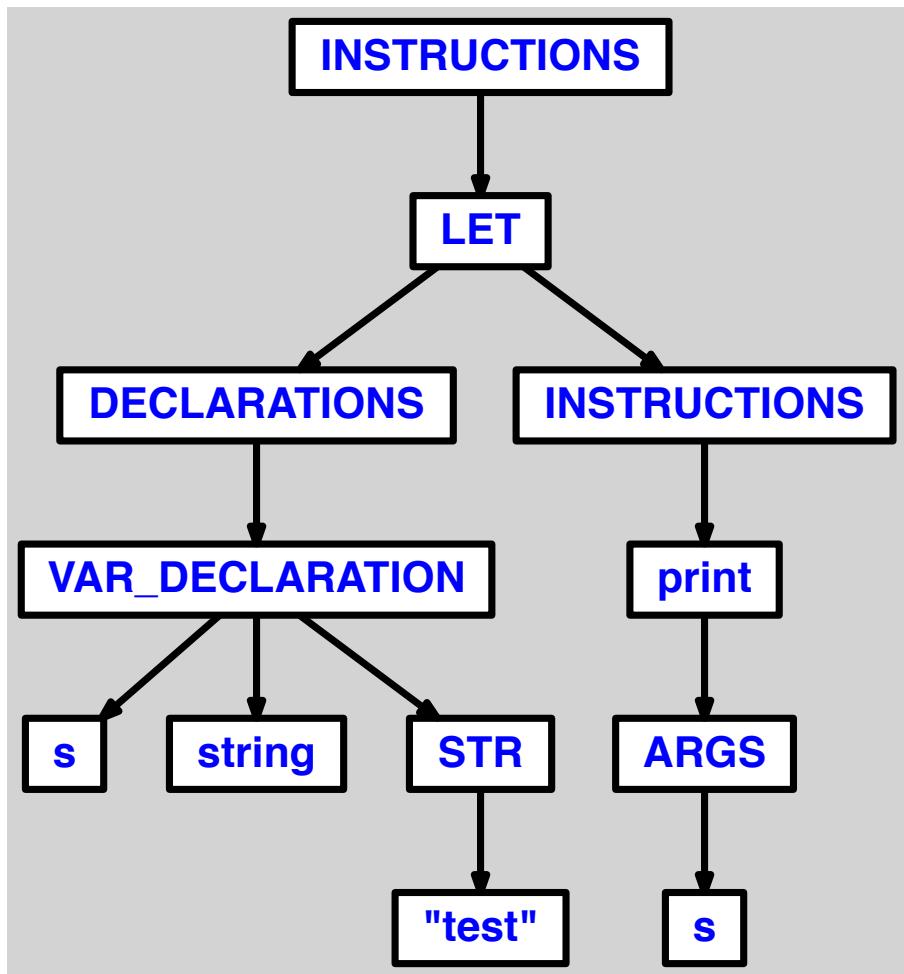
### 13.2.9 déclarations de chaîne avec réutilisation de 3 autres chaînes

```
1 let
2   var t := "test"
3   var u := "test"
4   var v := "test"
5   var s := t+u+v
6 in
7   print(s)
8 end
```



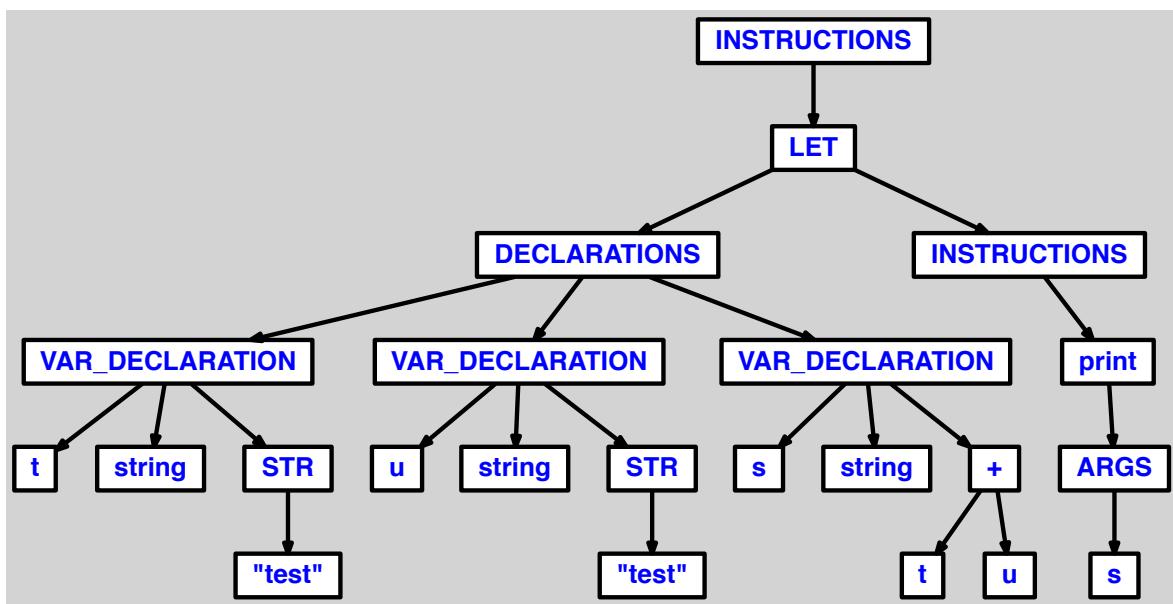
### 13.2.10 déclaration de chaîne avec type

```
1 let
2   var s : string := "test"
3 in
4   print(s)
5 end
```



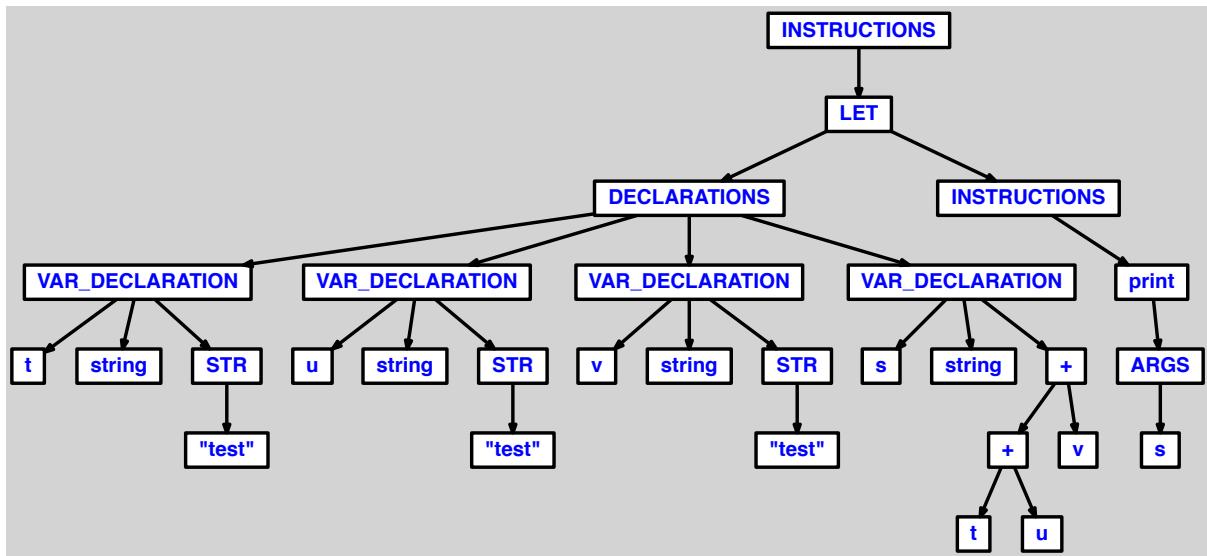
### 13.2.11 declarations de chaine avec types et reutilisation de 2 autres chaines

```
1 let
2   var t : string := "test"
3   var u : string := "test"
4   var s : string := t+u
5 in
6   print(s)
7 end
```



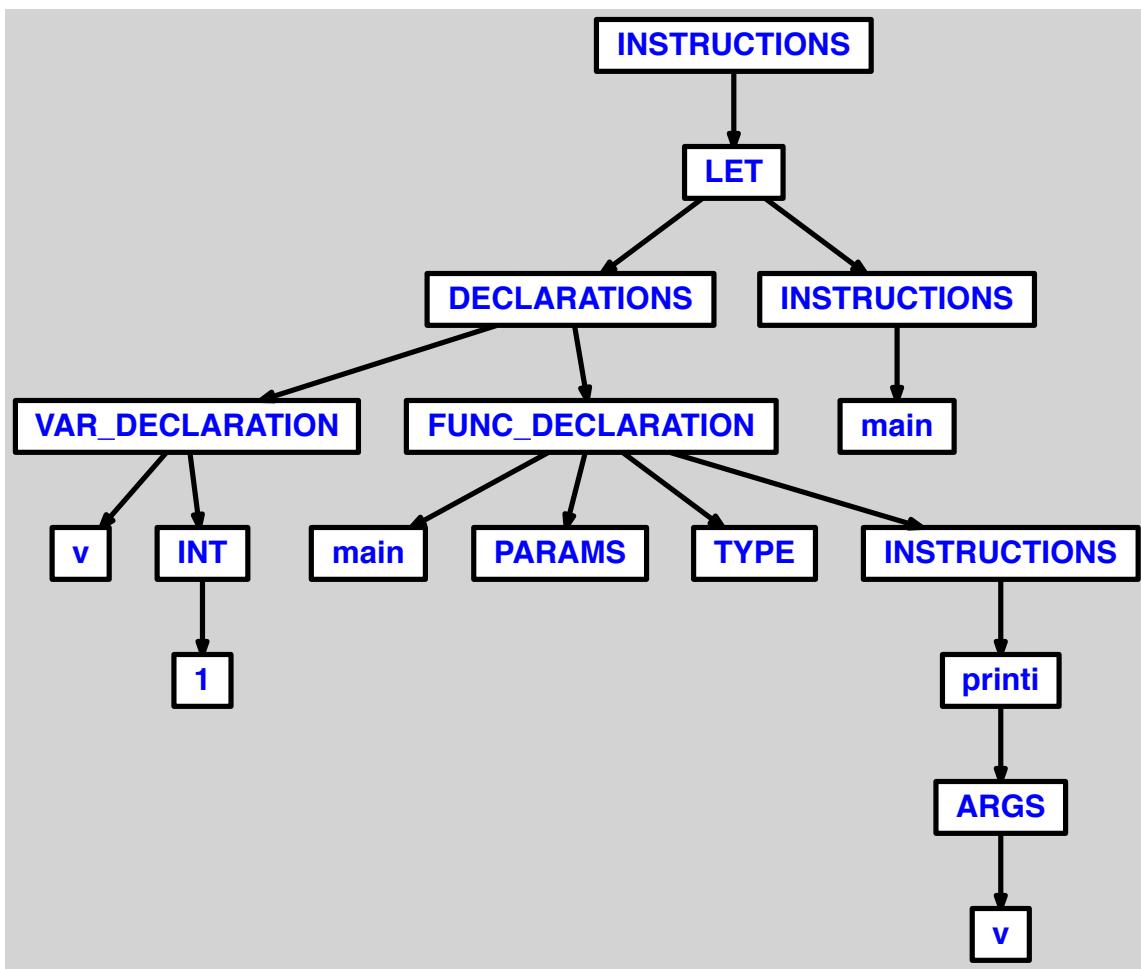
### 13.2.12 declarations de chaine avec types et reutilisation de 3 autres chaines

```
1 let
2   var t : string := "test"
3   var u : string := "test"
4   var v : string := "test"
5   var s : string := t+u+v
6 in
7   print(s)
8 end
```



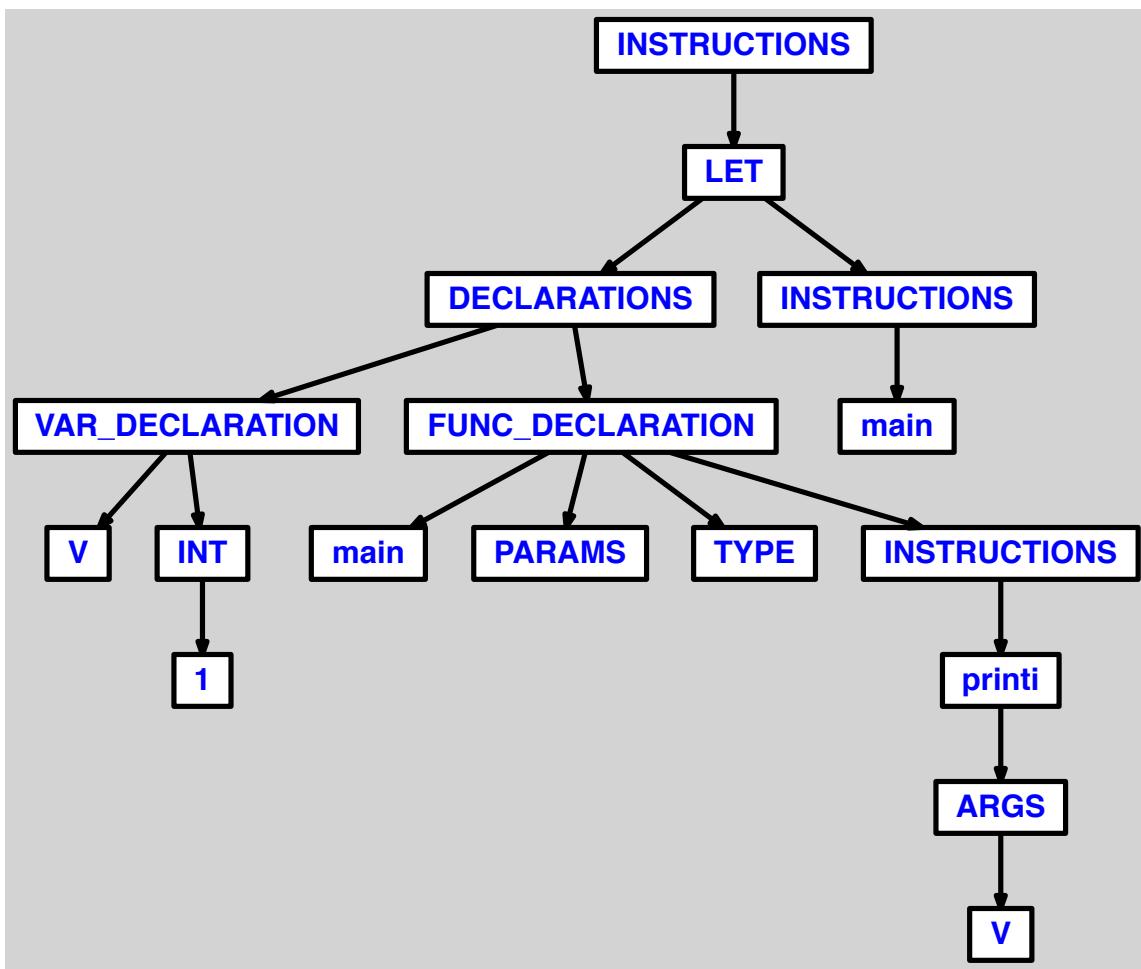
### 13.2.13 lettre minuscule comme nom de variable

```
1 let
2   var v := 1
3
4   function main() = printi(v)
5 in main() end
```



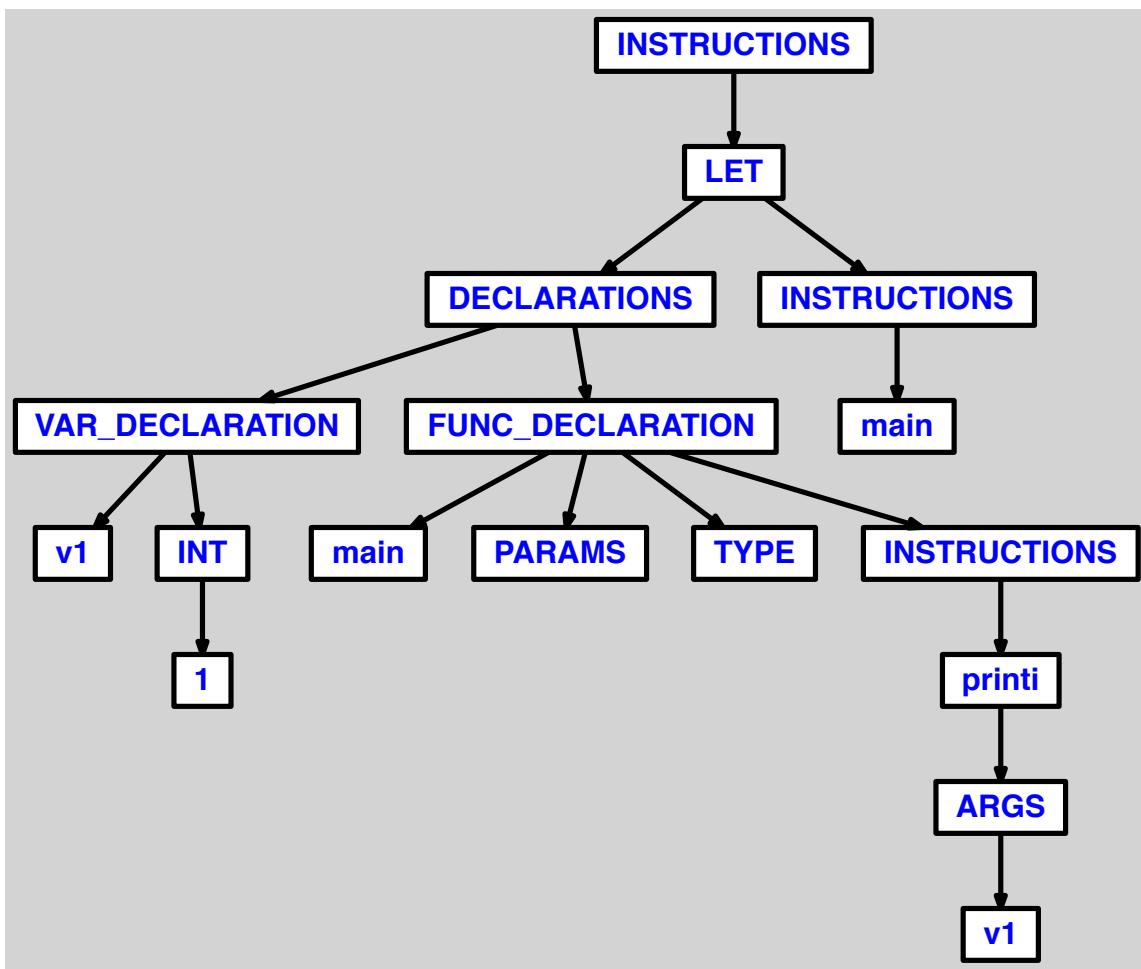
### 13.2.14 lettre majuscule comme nom de variable

```
1 let
2   var V := 1
3
4   function main() = printi(V)
5 in main() end
```



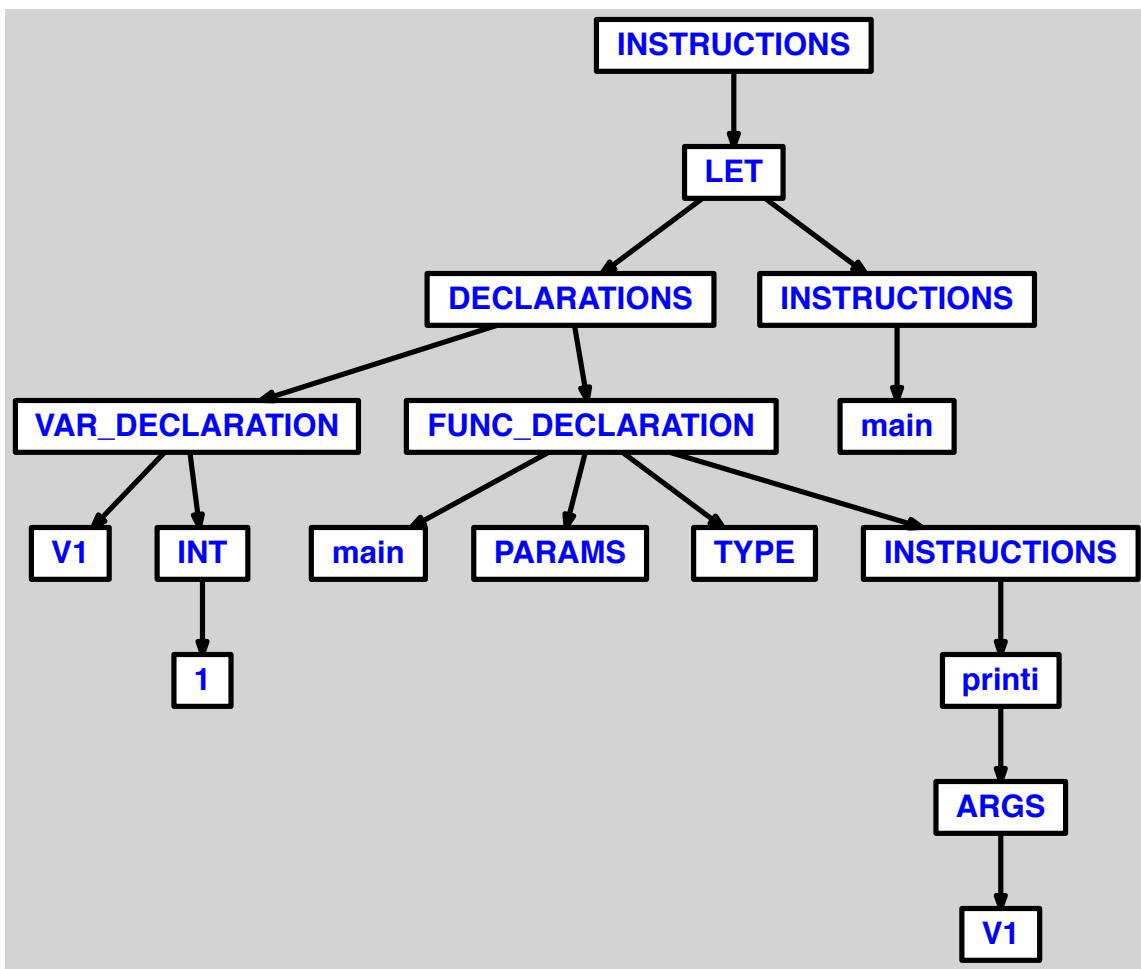
### 13.2.15 lettre minuscule et chiffre dans nom de variable

```
1 let
2   var v1 := 1
3
4   function main() = printi(v1)
5 in main() end
```



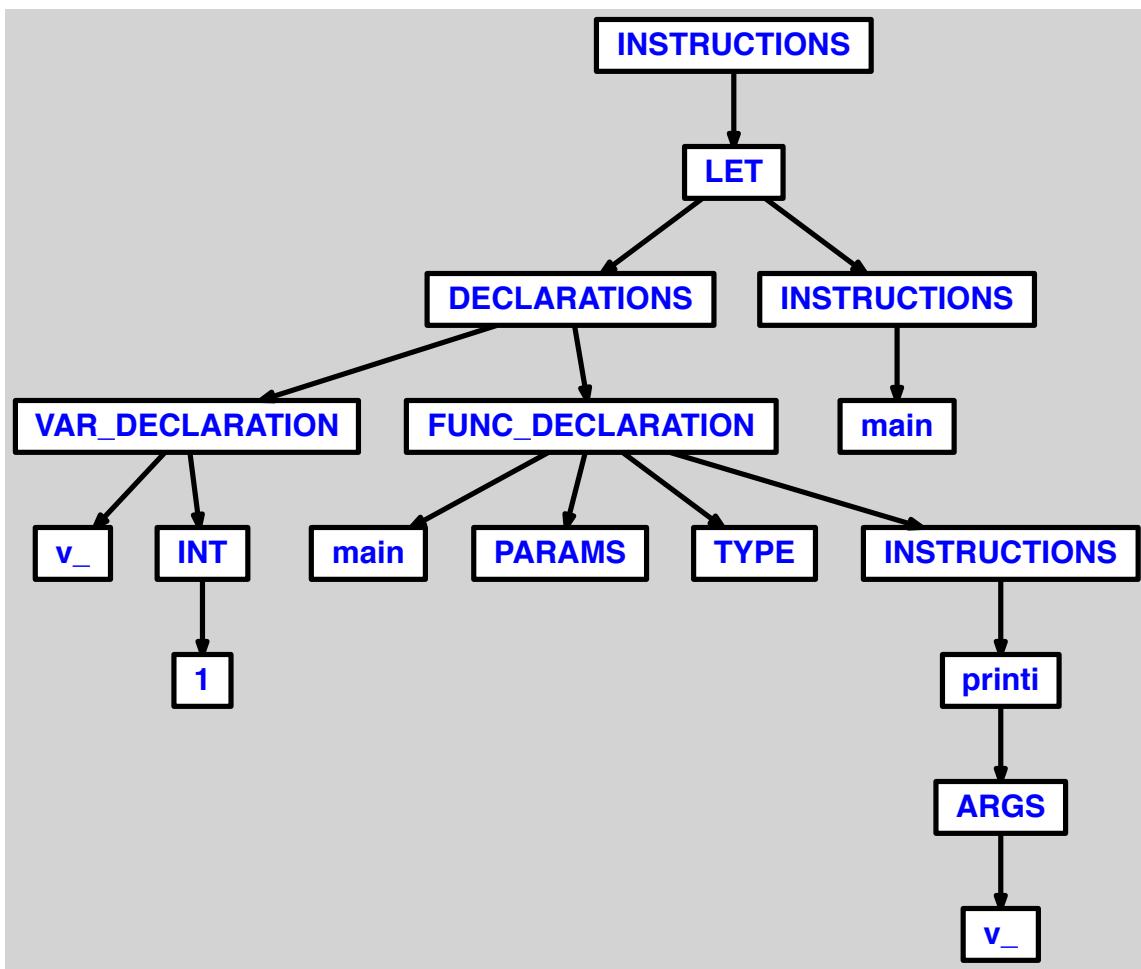
### 13.2.16 lettre majuscule et chiffre dans nom de variable

```
1 let
2   var V1 := 1
3
4   function main() = printi(V1)
5 in main() end
```



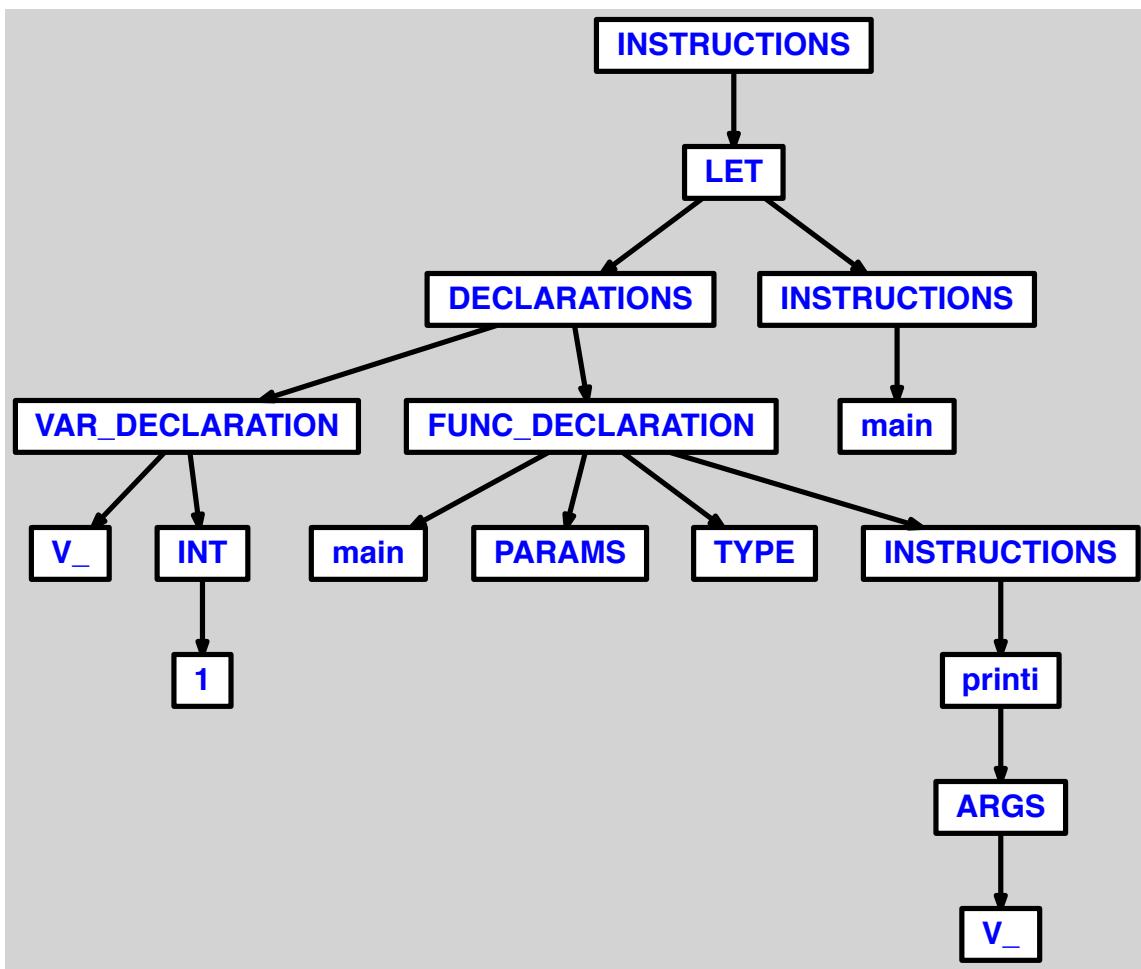
### 13.2.17 lettre minuscule et <\_> dans nom de variable

```
1 let
2   var v_- := 1
3
4   function main() = printi(v_)
5 in main() end
```



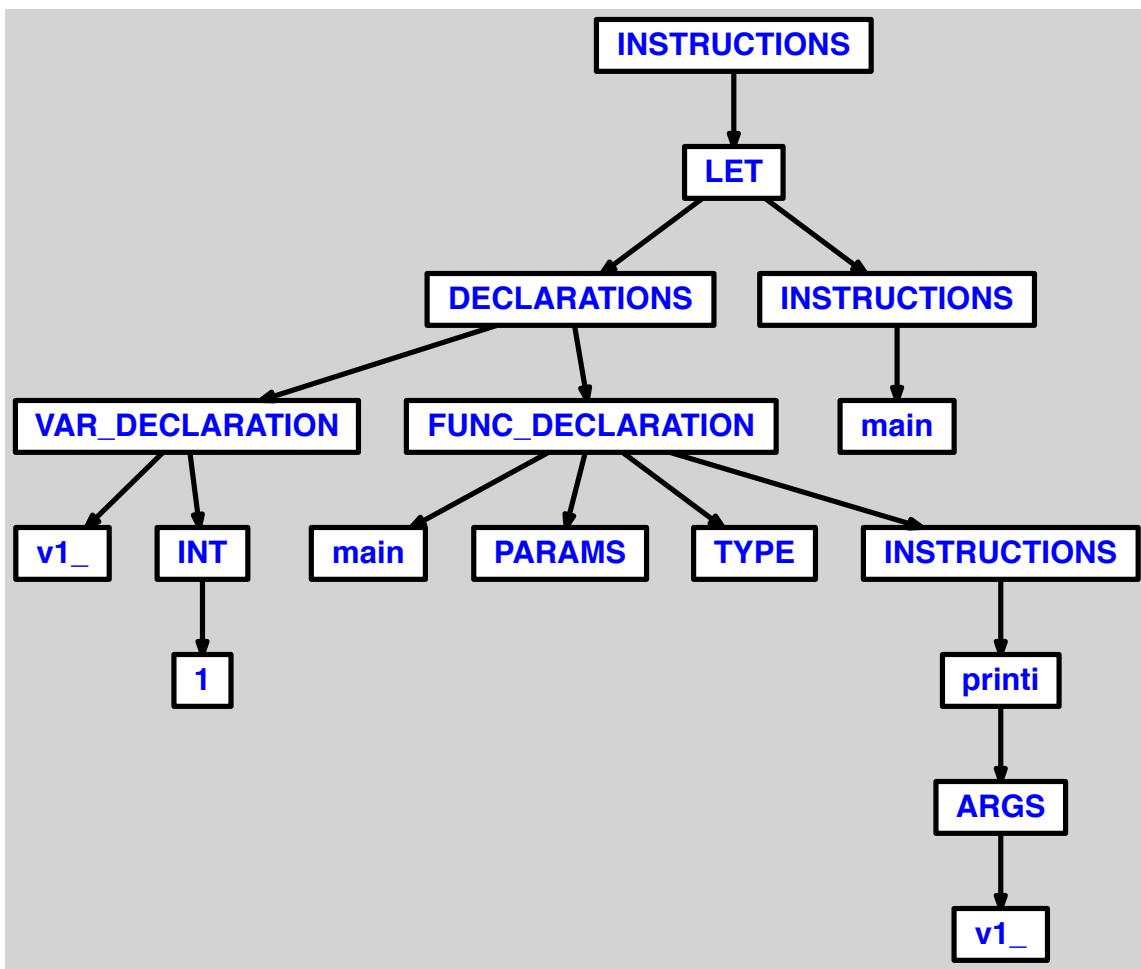
### 13.2.18 lettre majuscule et <\_> dans nom de variable

```
1 let
2   var V_ := 1
3
4   function main() = printi(V_)
5 in main() end
```



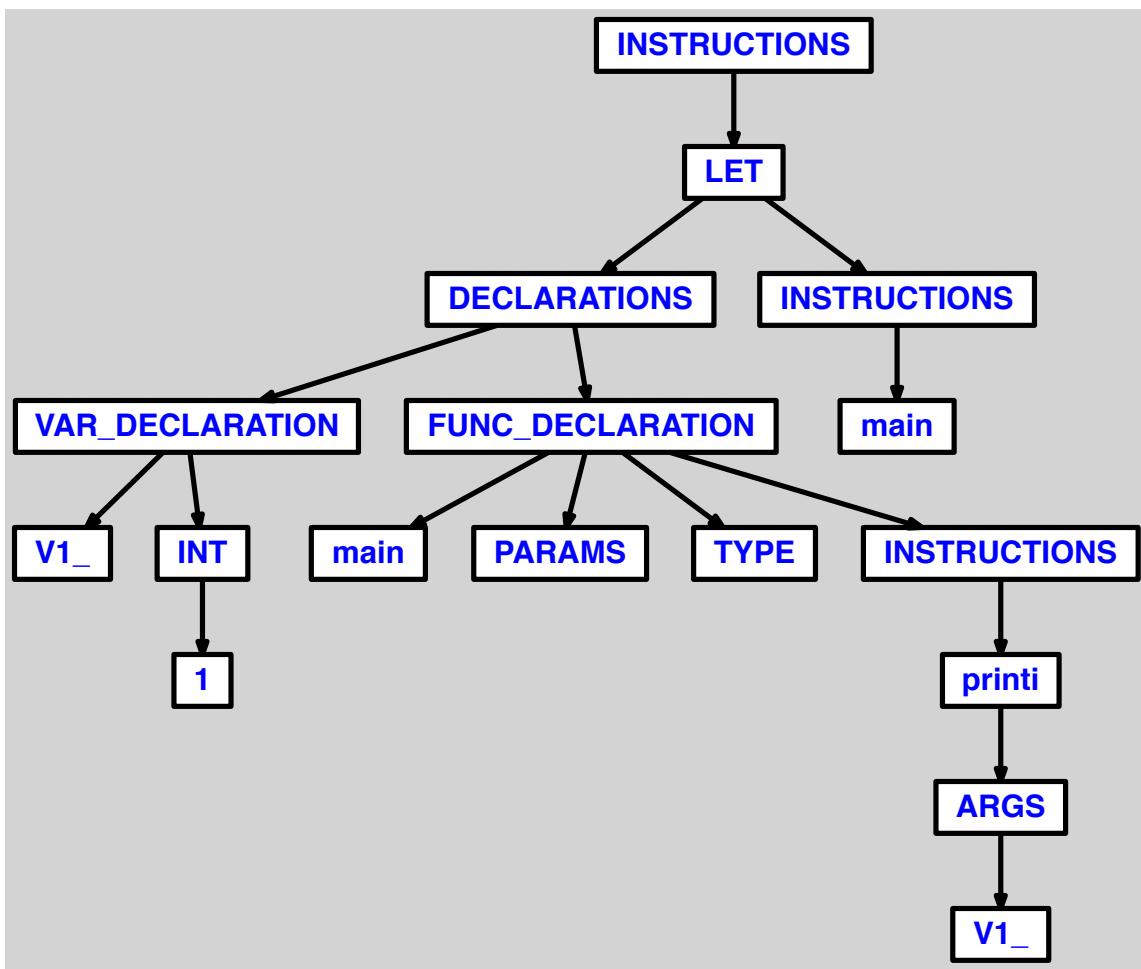
### 13.2.19 lettre minuscule, chiffre et <-> dans nom de variable

```
1 let
2   var v1_ := 1
3
4   function main() = printi(v1_)
5 in main() end
```



### 13.2.20 lettre majuscule, chiffre et <-> dans nom de variable

```
1 let
2   var V1_ := 1
3
4   function main() = printi(V1_)
5 in main() end
```

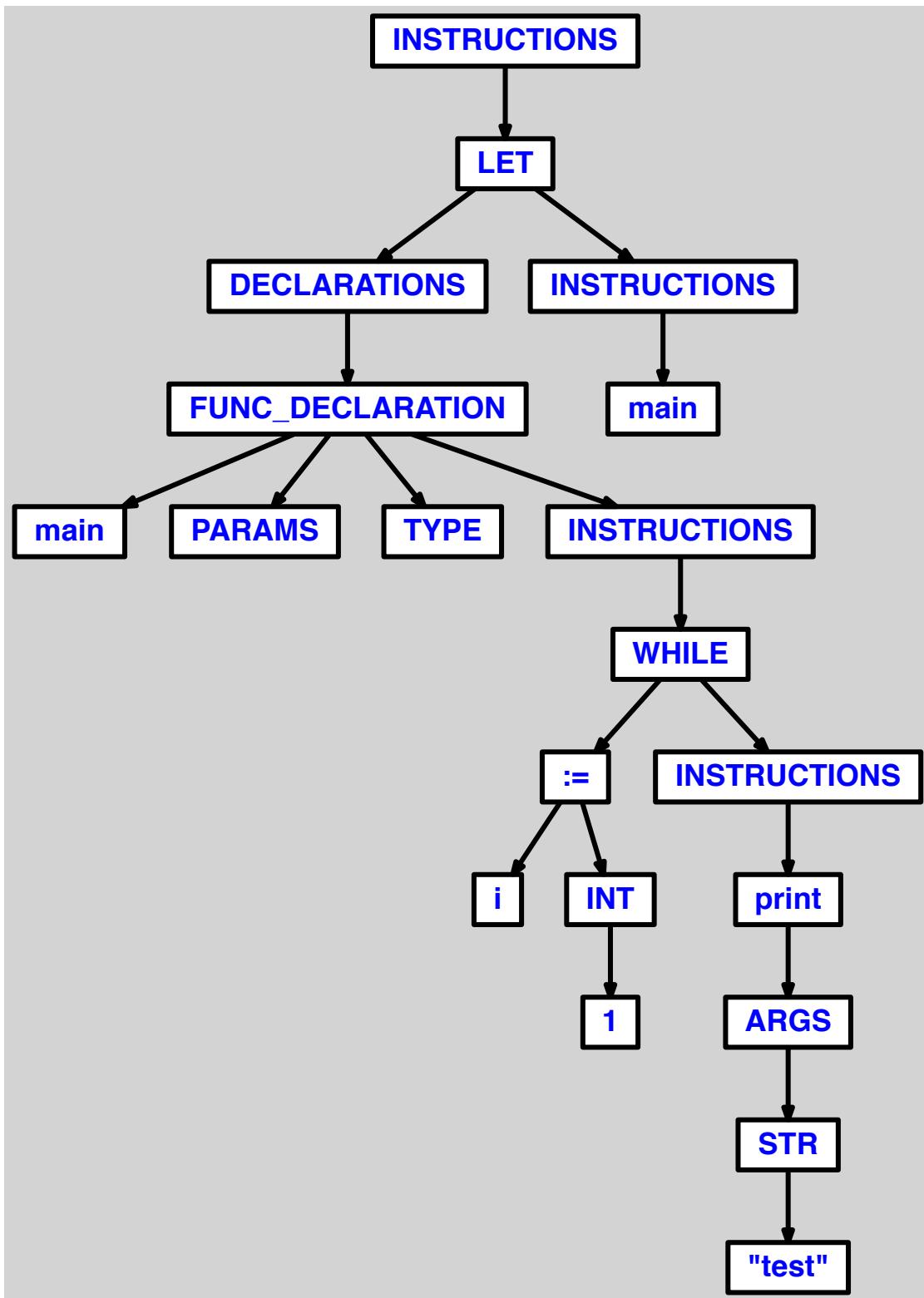


## 14 while

### 14.1 KO

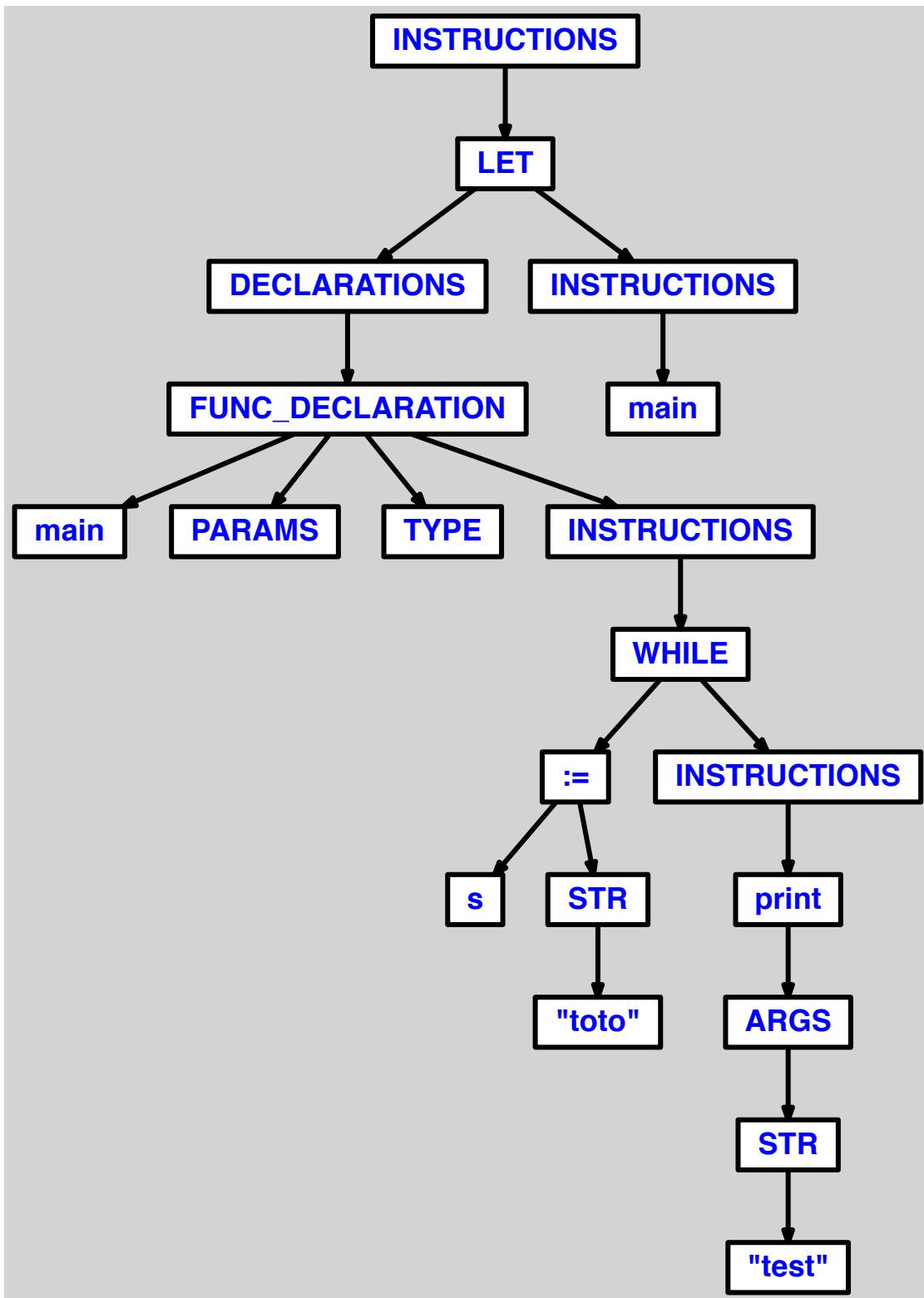
#### 14.1.1 <while> avec affectation d'entier

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



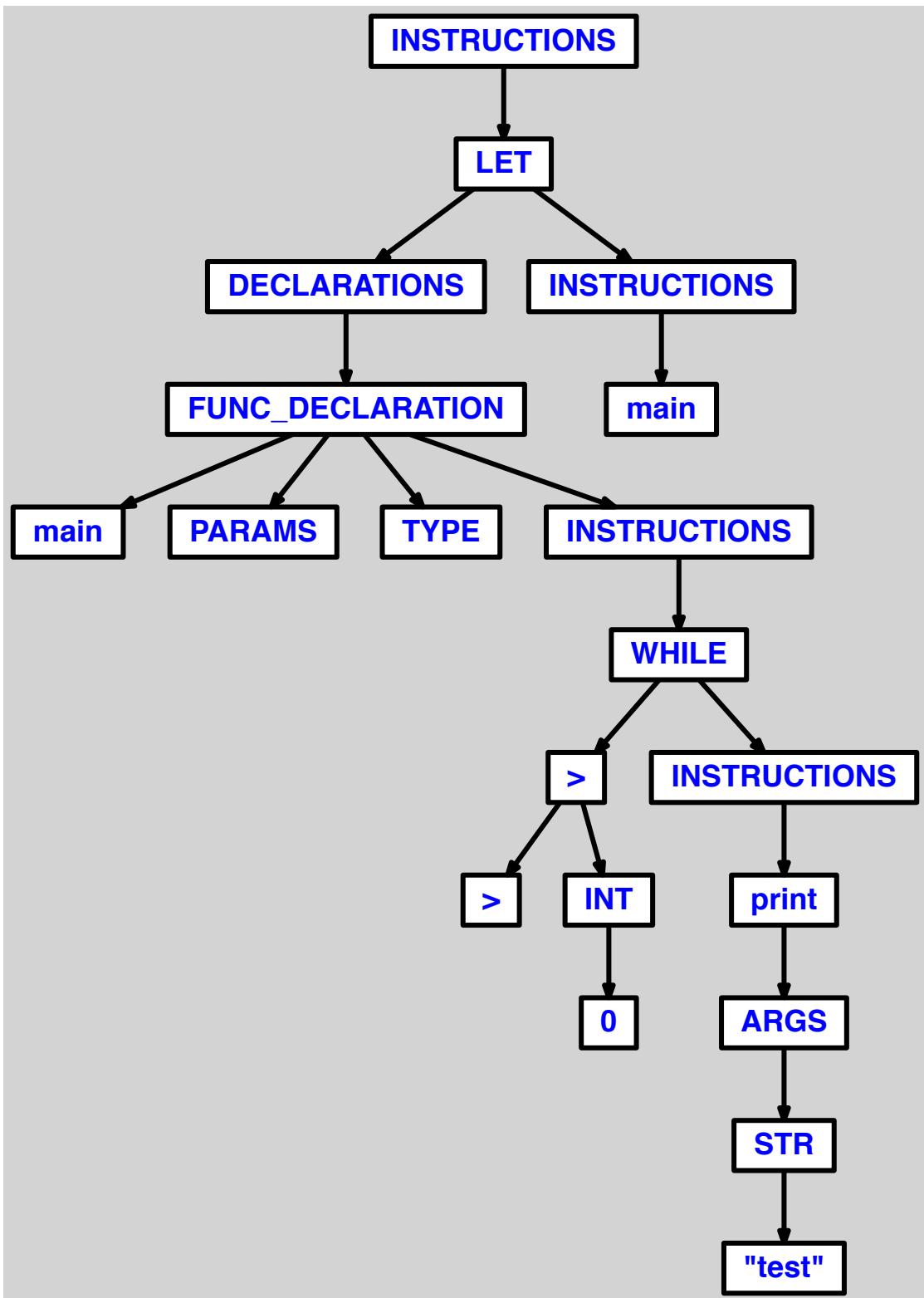
#### 14.1.2 <while> avec affectation de chaine

```
1 let
2   function main() =
3     while s := "toto" do
4       print("test")
5   in main() end
```



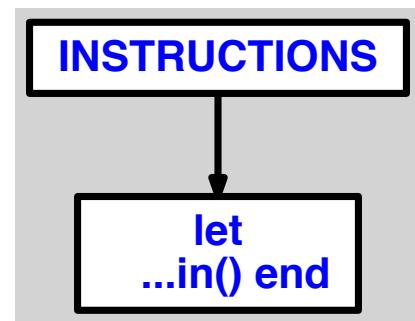
### 14.1.3 <while> avec oubli du compteur

```
1 let
2   function main() =
3     while > 0 do
4       print("test")
5   in main() end
```



#### 14.1.4 <while> avec oubli du <while>

```
1 let
2   var i := 0
3
4   function main() =
5     i >= 0 do
6       print("test")
7   in main() end
```



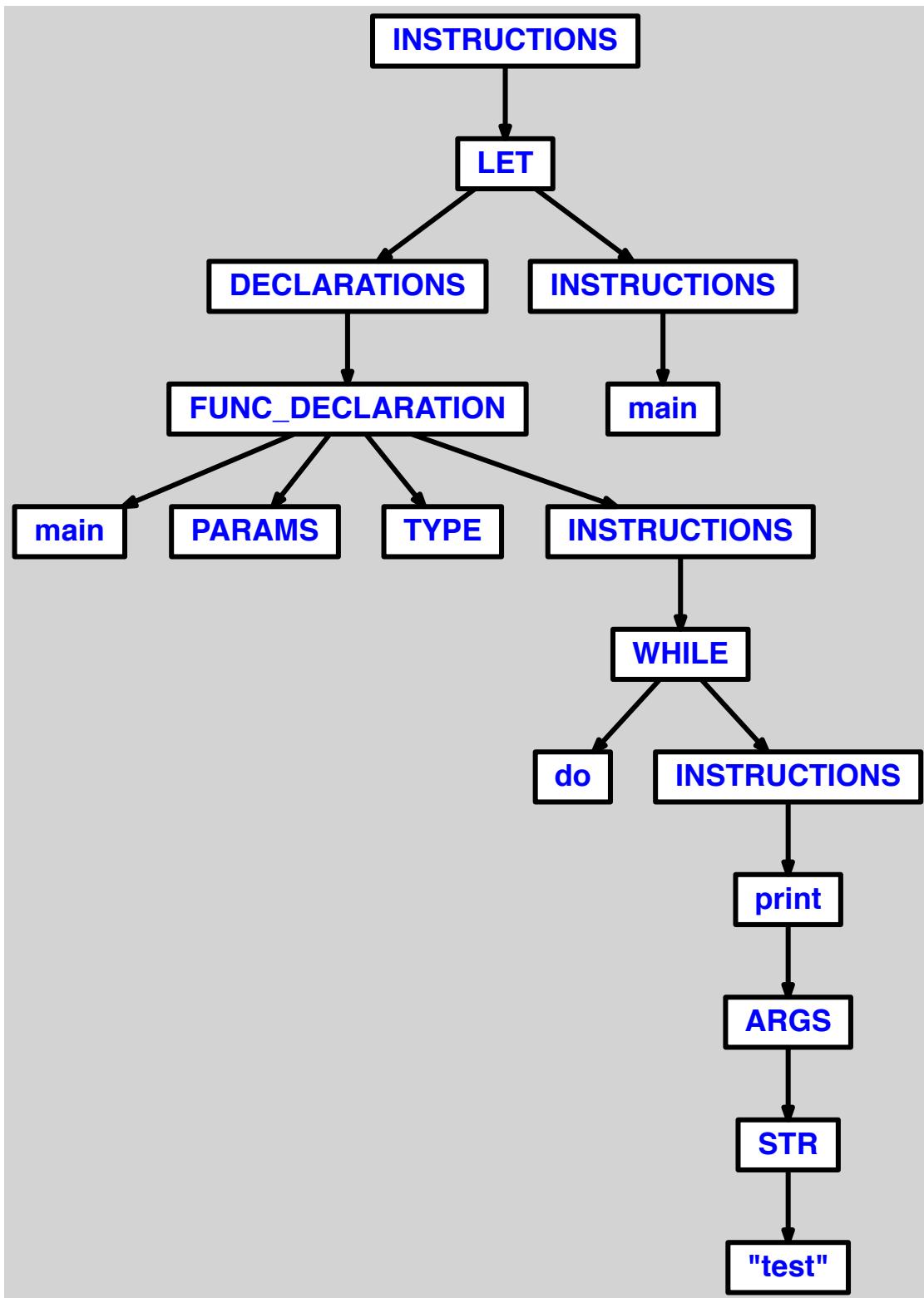
#### 14.1.5 <while> avec oubli du <do>

```
1 let
2   var i := 0
3
4   function main() :
5     while i >= 0
6       print("test")
7   in main() end
```

Pas d'AST, problème de syntaxe.

#### 14.1.6 <while> avec oubli de la condition

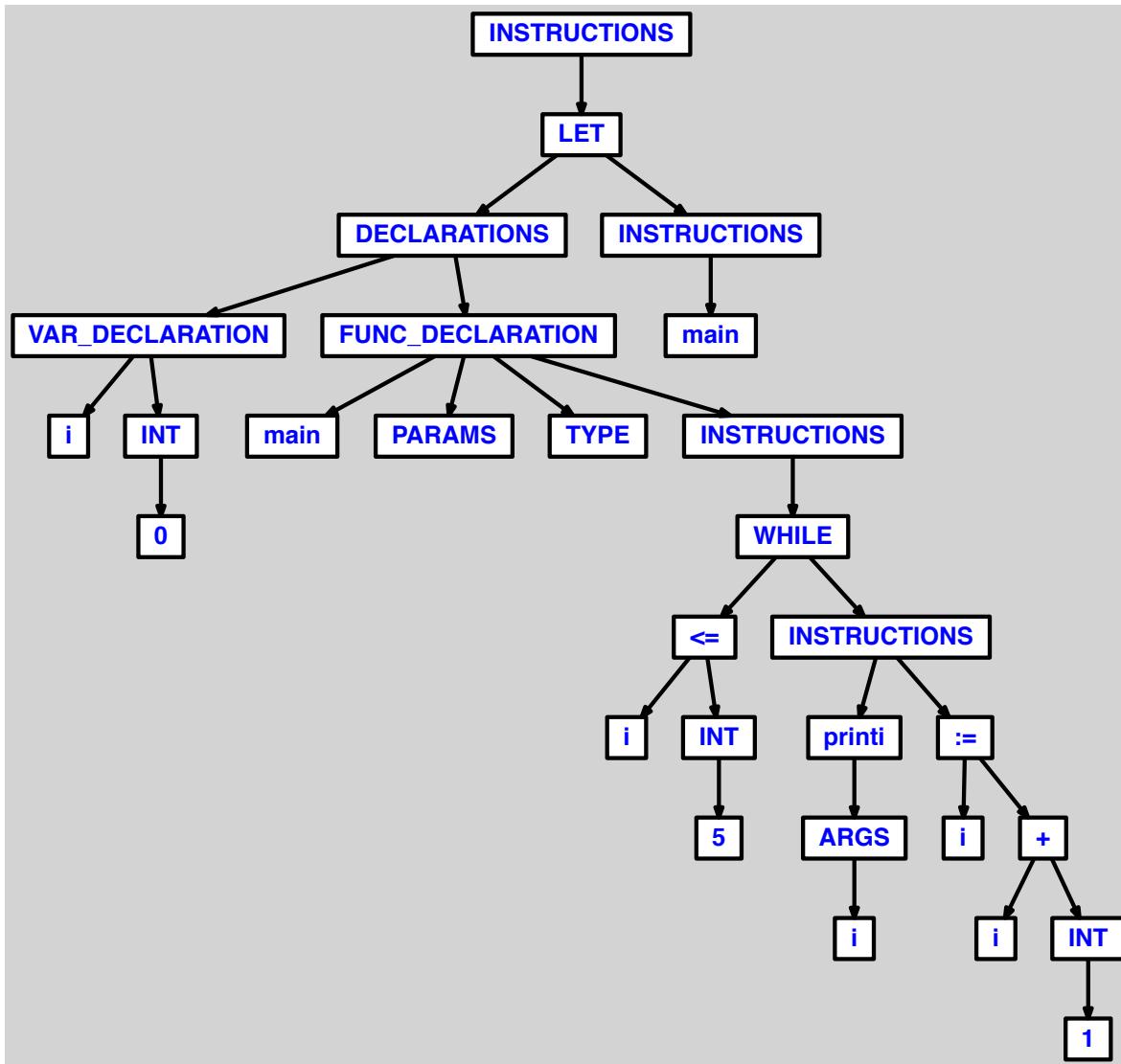
```
1 let
2   function main() =
3     while do
4       print("test")
5   in main() end
```



## 14.2 OK

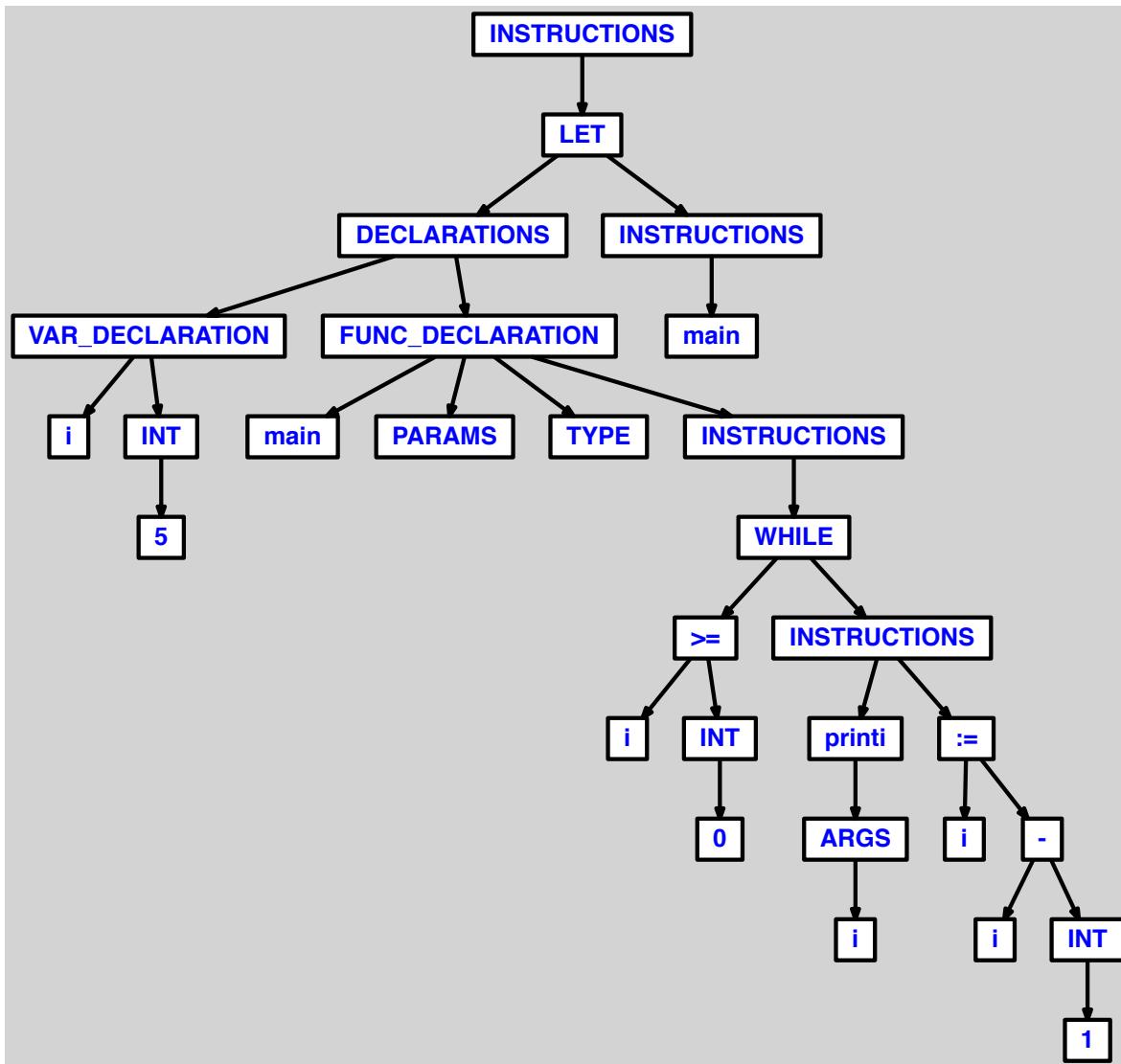
### 14.2.1 <while> avec iteration croissante

```
1 let
2   var i := 0
3
4   function main() =
5     while i <= 5 do
6       (printi(i);
7        i := i+1)
8 in main() end
```



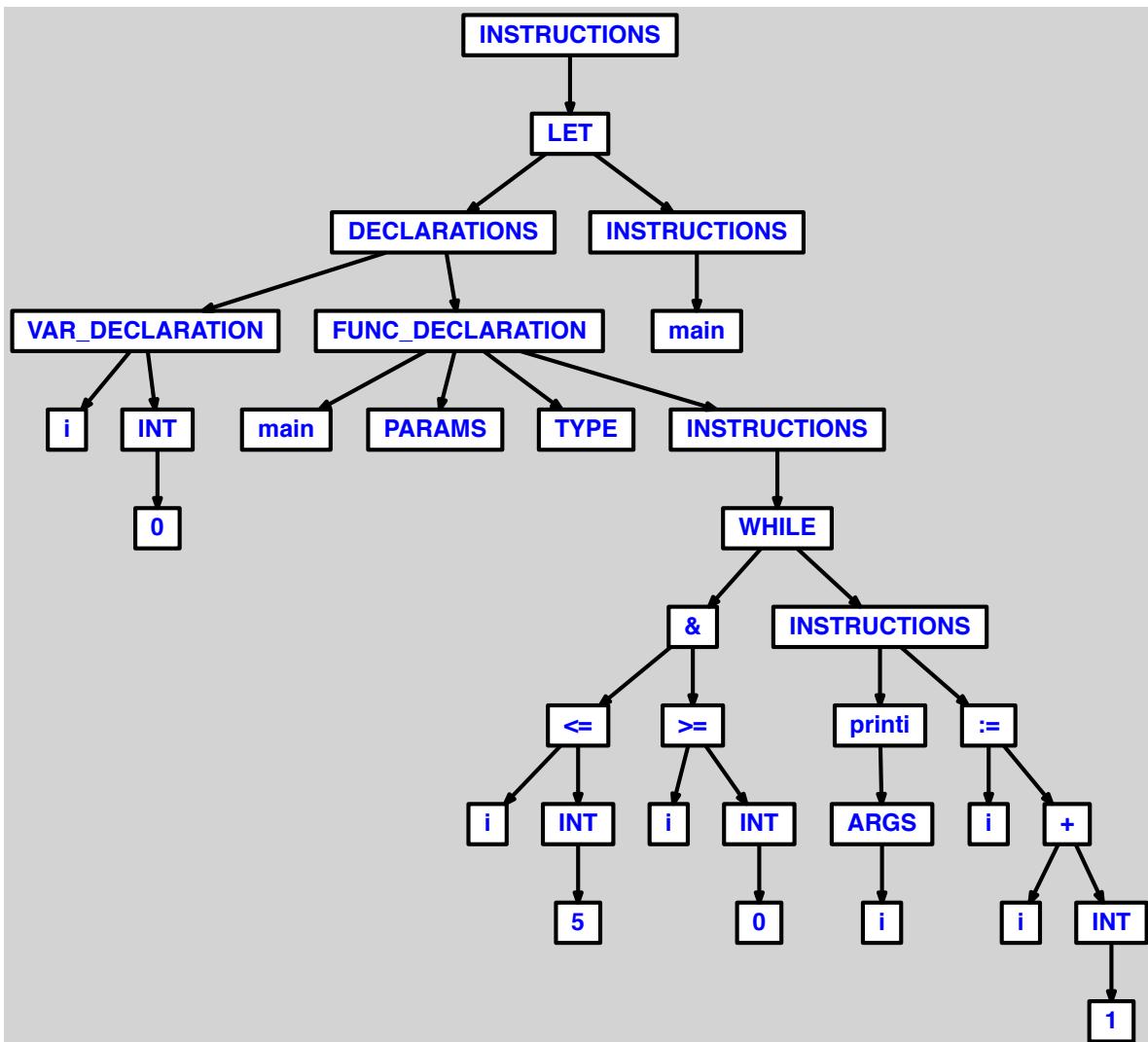
#### 14.2.2 <while> avec iteration decroissante

```
1 let
2   var i := 5
3
4   function main() =
5     while i >= 0 do
6       (printi(i);
7        i := i-1)
8 in main() end
```



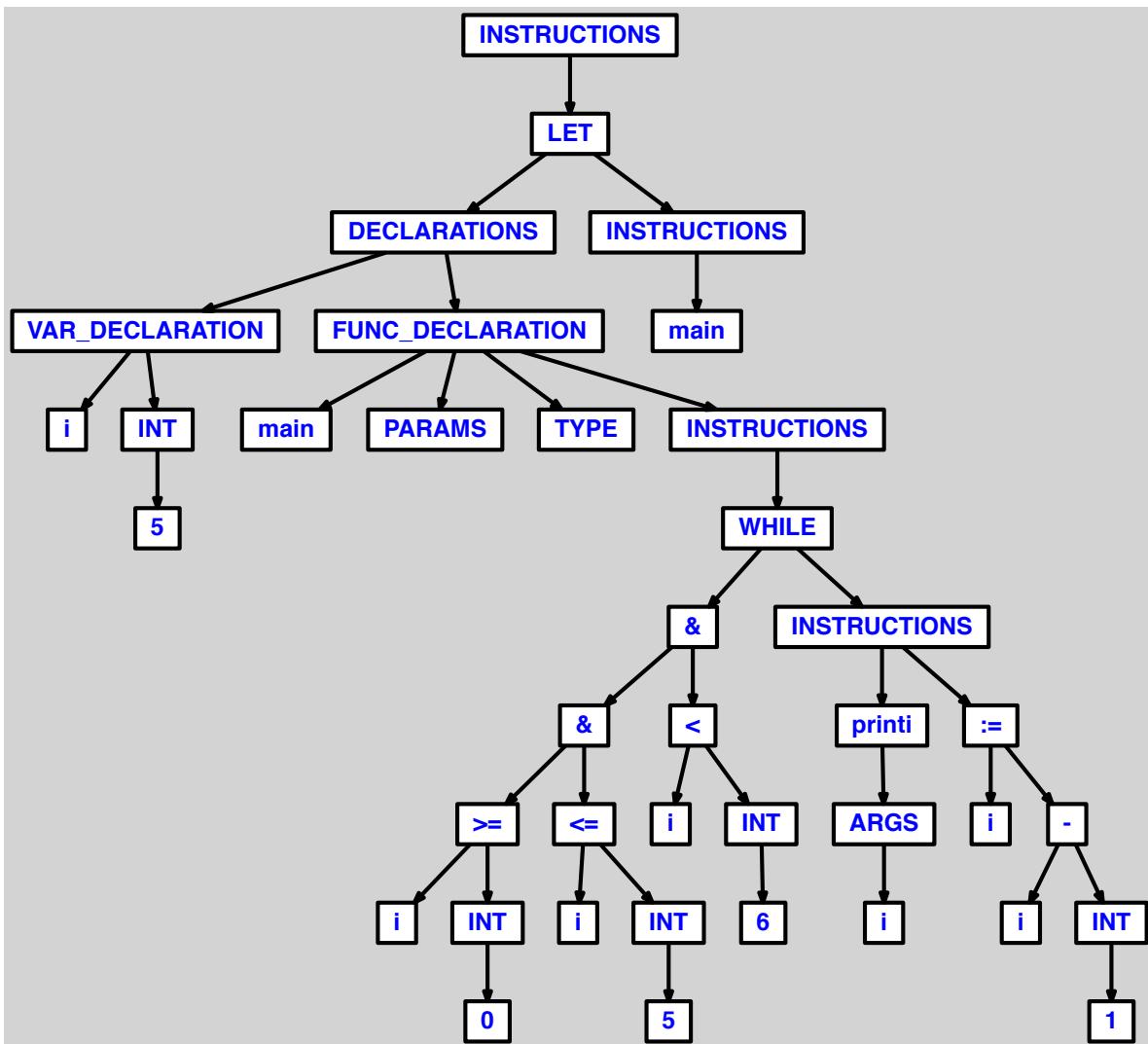
### 14.2.3 <while> avec double-condition

```
1 let
2   var i := 0
3
4   function main() =
5     while i <= 5 & i >= 0 do
6       (printi(i);
7        i := i+1)
8 in main() end
```



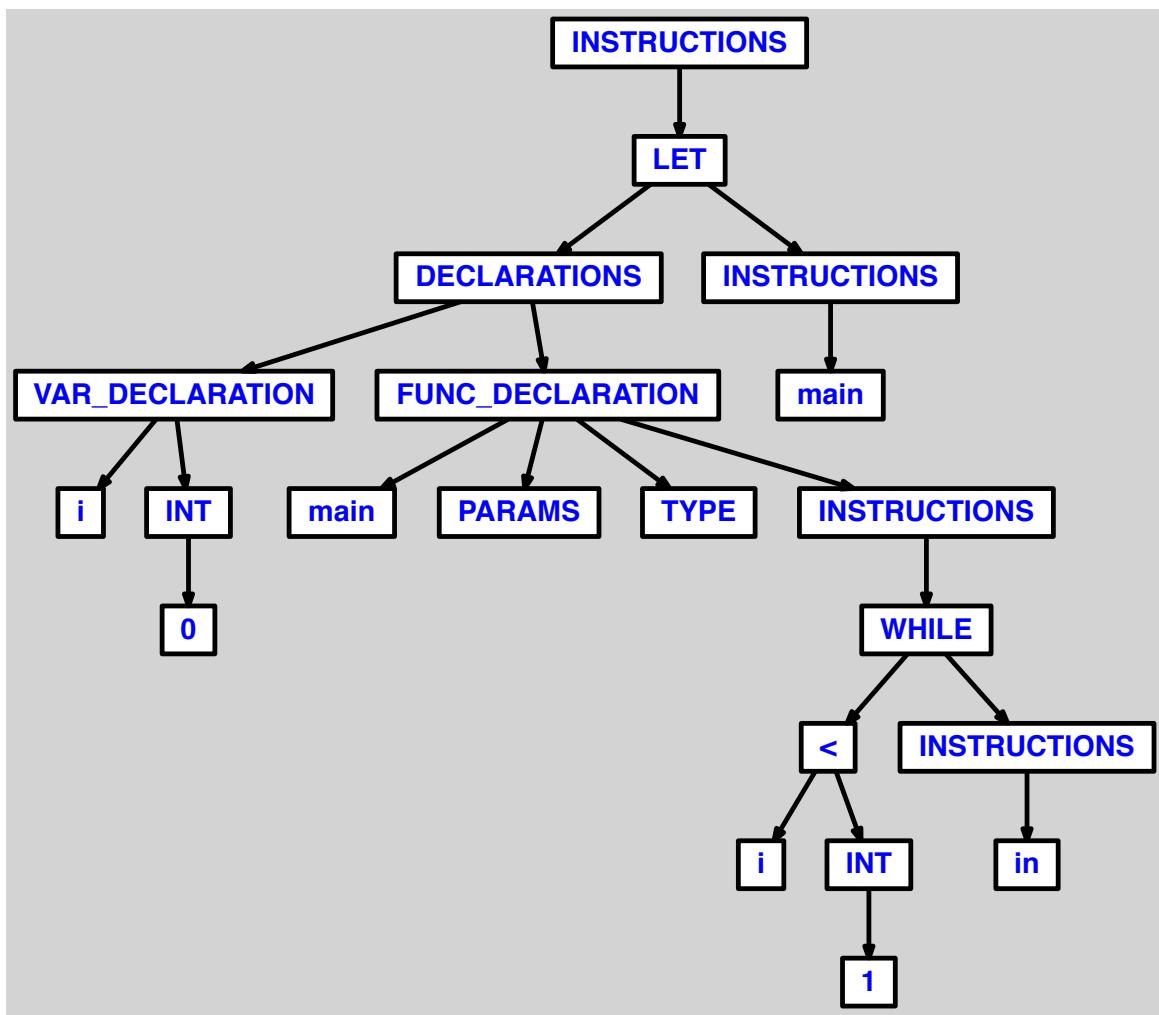
#### 14.2.4 <while> avec triple-condition

```
1 let
2   var i := 5
3
4   function main() =
5     while i >= 0 & i <= 5 & i < 6 do
6       (printi(i);
7        i := i-1)
8 in main() end
```



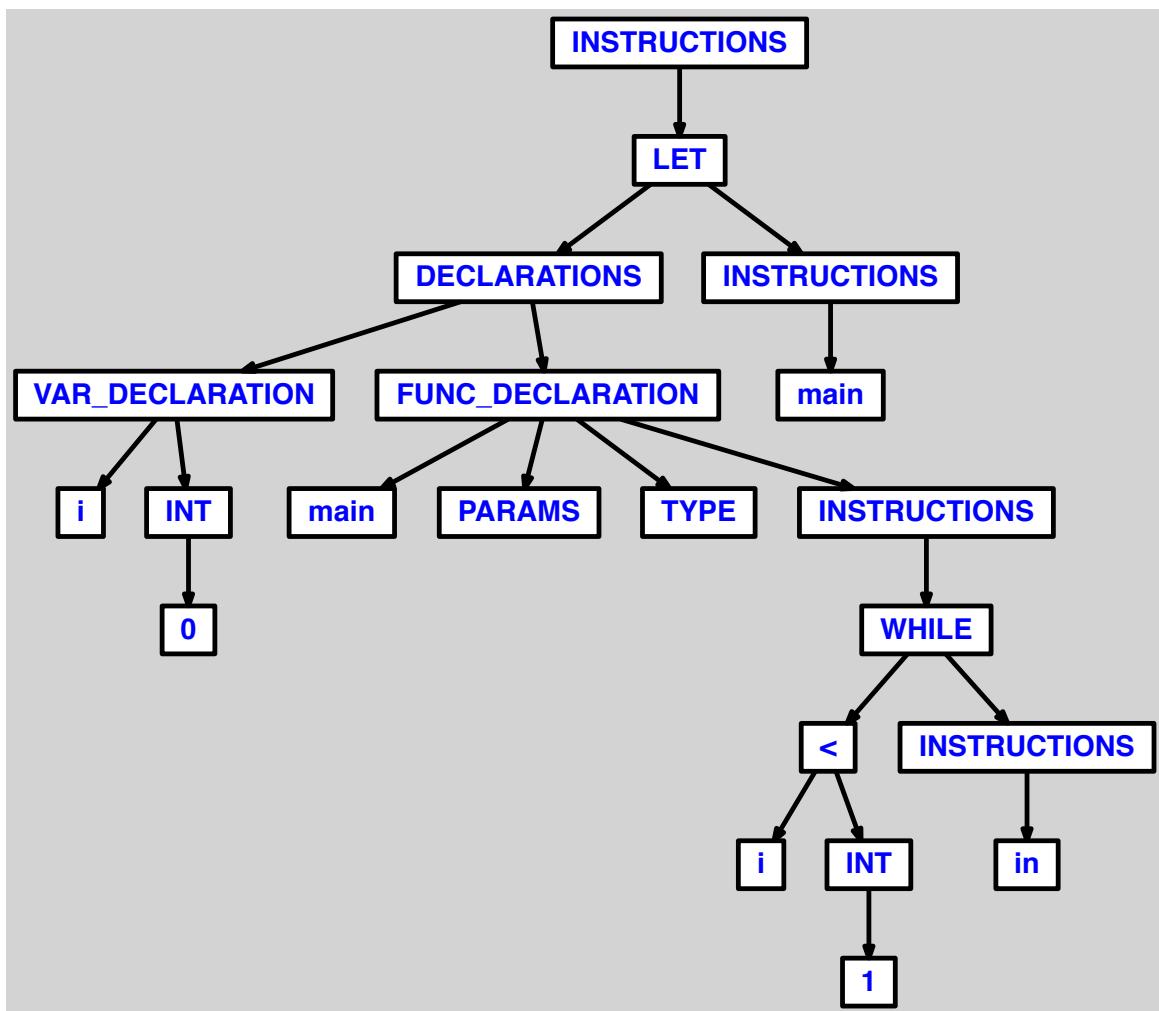
#### 14.2.5 <while> sans instruction

```
1 let
2   var i := 0
3
4   function main() =
5     while i < 1 do
6       in main() end
```



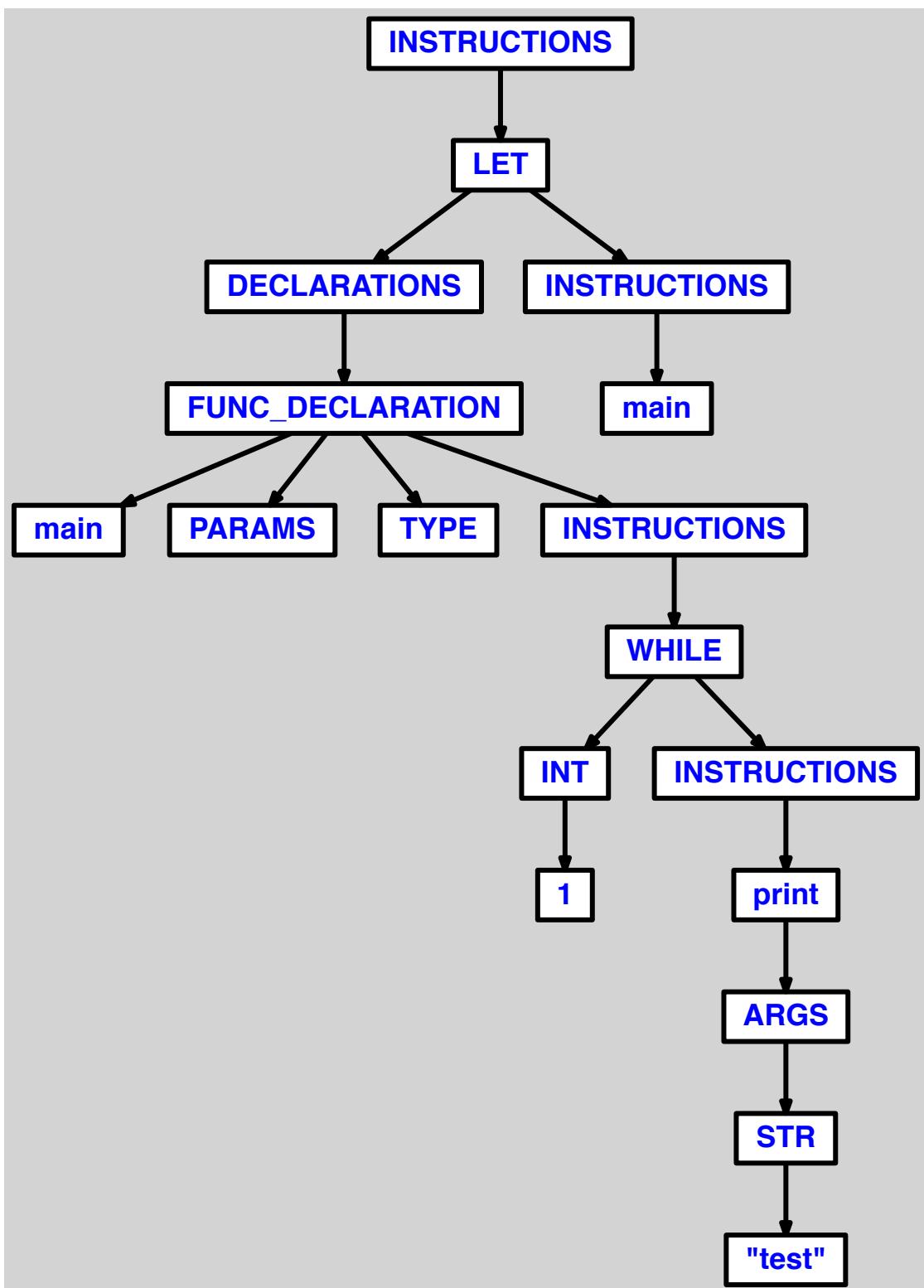
#### 14.2.6 <while> avec ligne vide

```
1 let
2   var i := 0
3
4   function main() =
5     while i < 1 do
6
7   in main() end
```



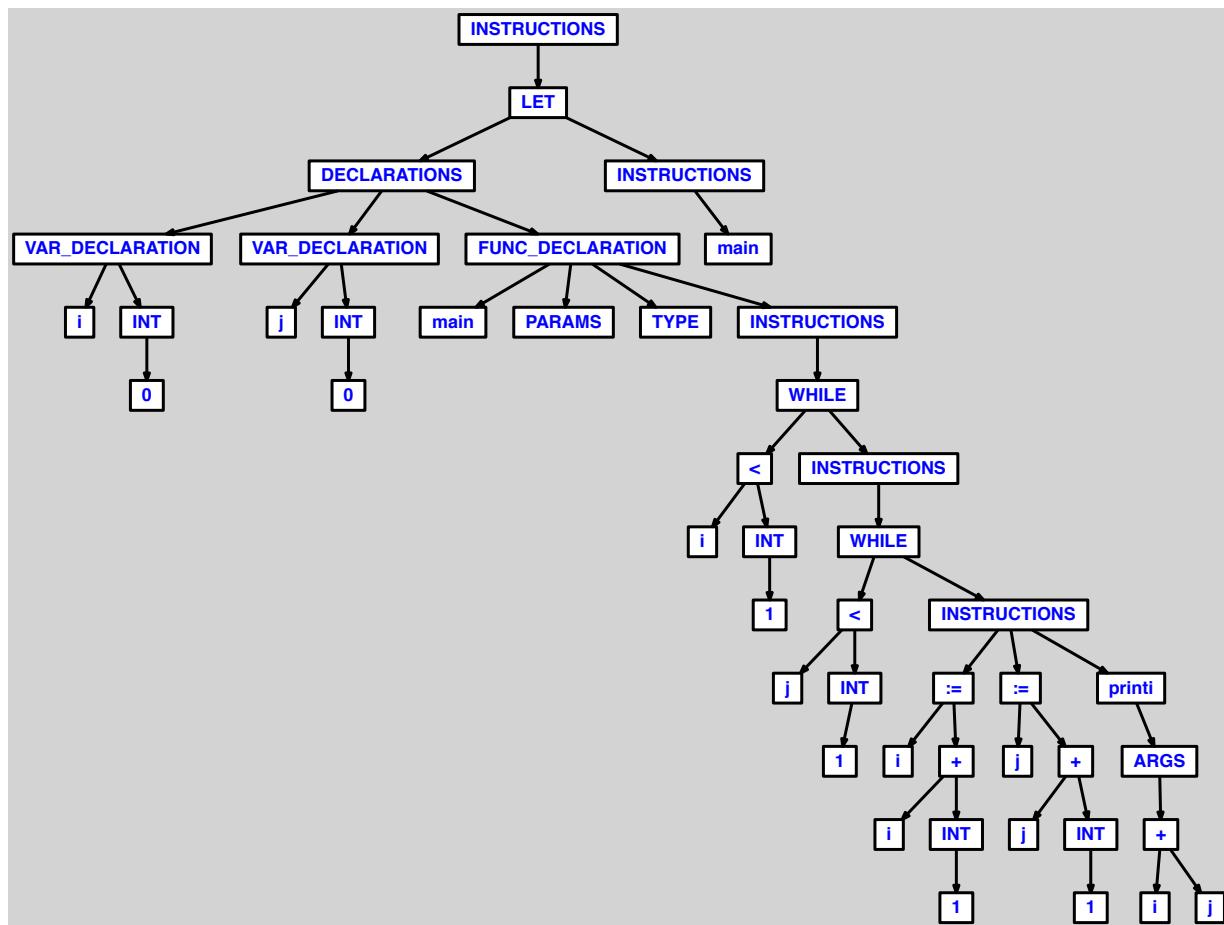
#### 14.2.7 <while> avec condition entiere

```
1 let
2   function main() =
3     while 1 do
4       print("test")
5   in main() end
```



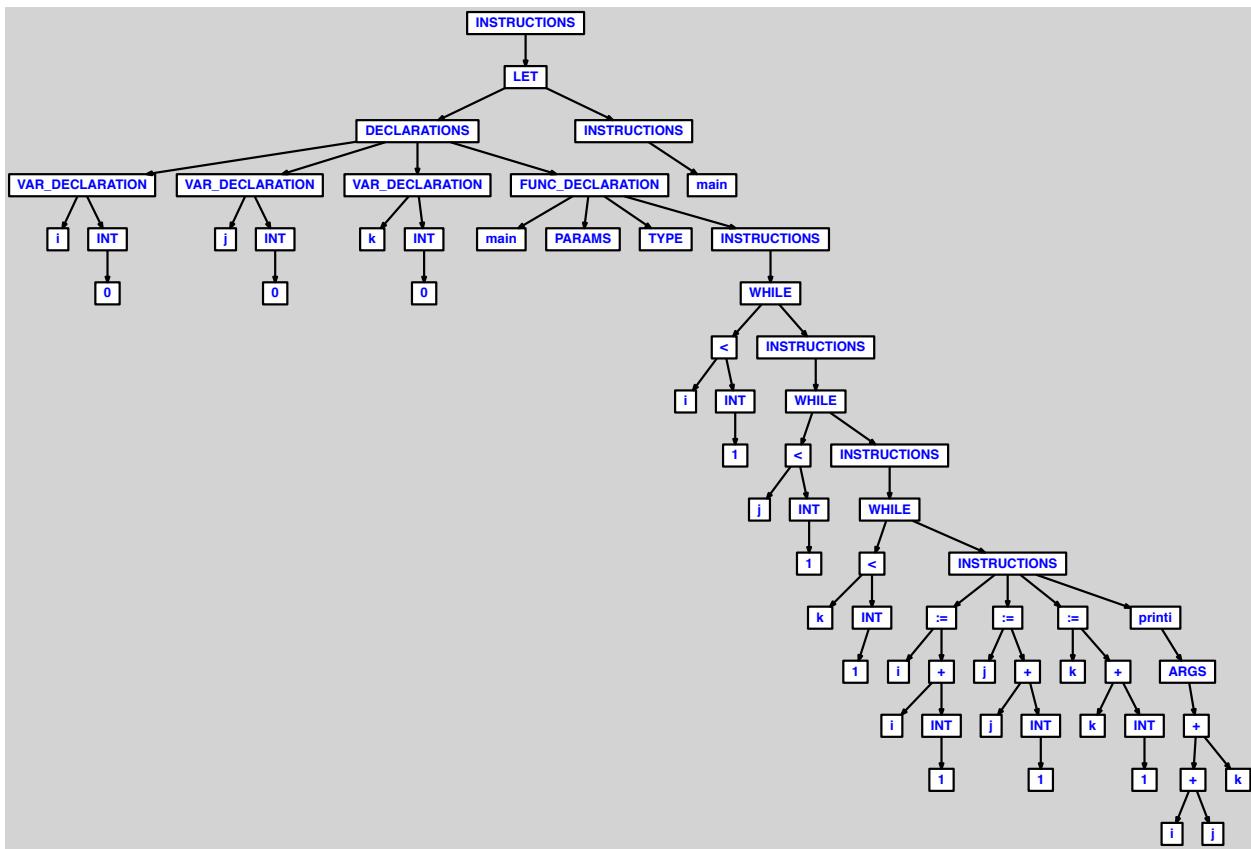
#### 14.2.8 <while> avec double-imbrication

```
1 let
2   var i := 0
3   var j := 0
4
5   function main() =
6     while i < 1 do
7       while j < 1 do
8         (i := i+1;
9          j := j+1;
10         printi(i+j))
11 in main() end
```



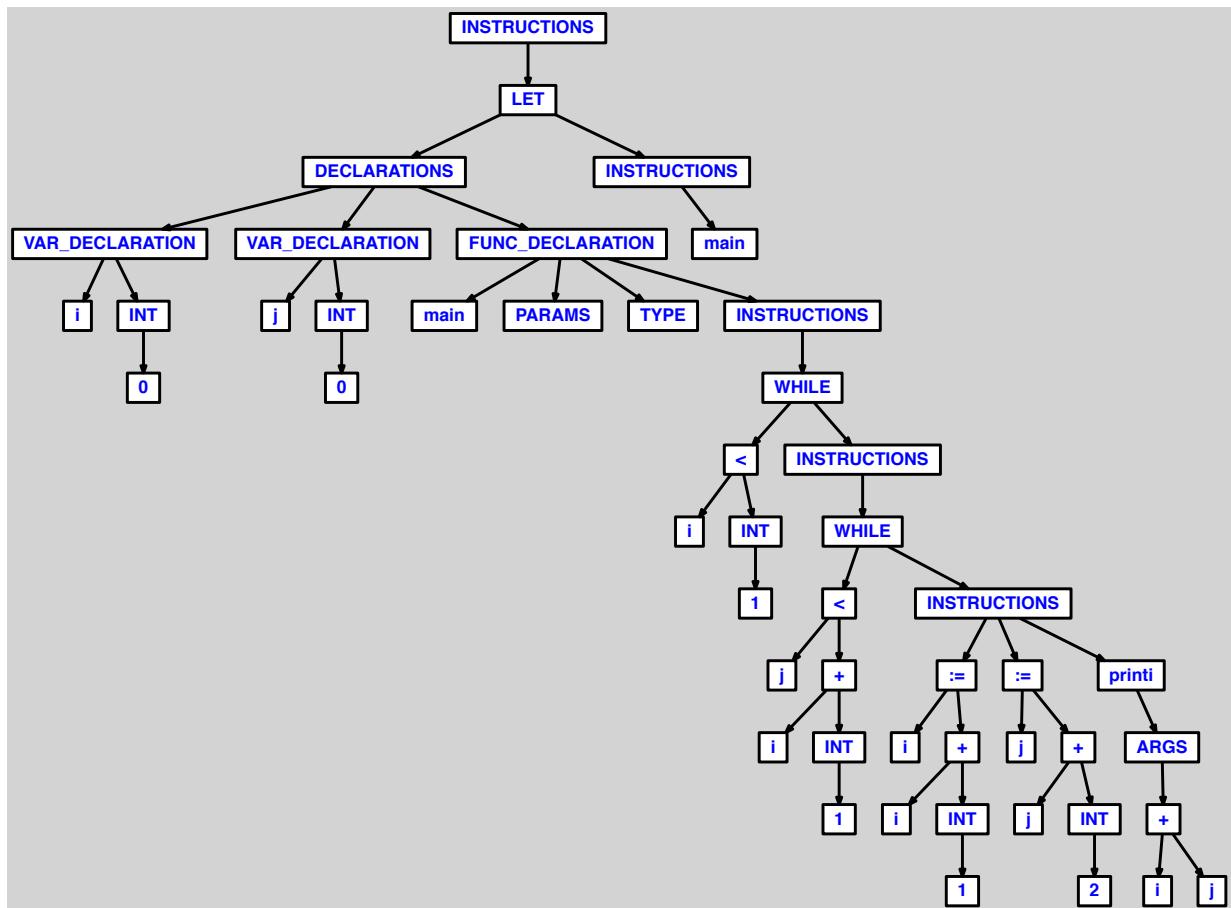
#### 14.2.9 <while> avec triple-imbrication

```
1 let
2     var i := 0
3     var j := 0
4     var k := 0
5
6     function main() =
7         while i < 1 do
8             while j < 1 do
9                 while k < 1 do
10                     (i := i+1;
11                     j := j+1;
12                     k := k+1;
13                     printi(i+j+k))
14     in main() end
```



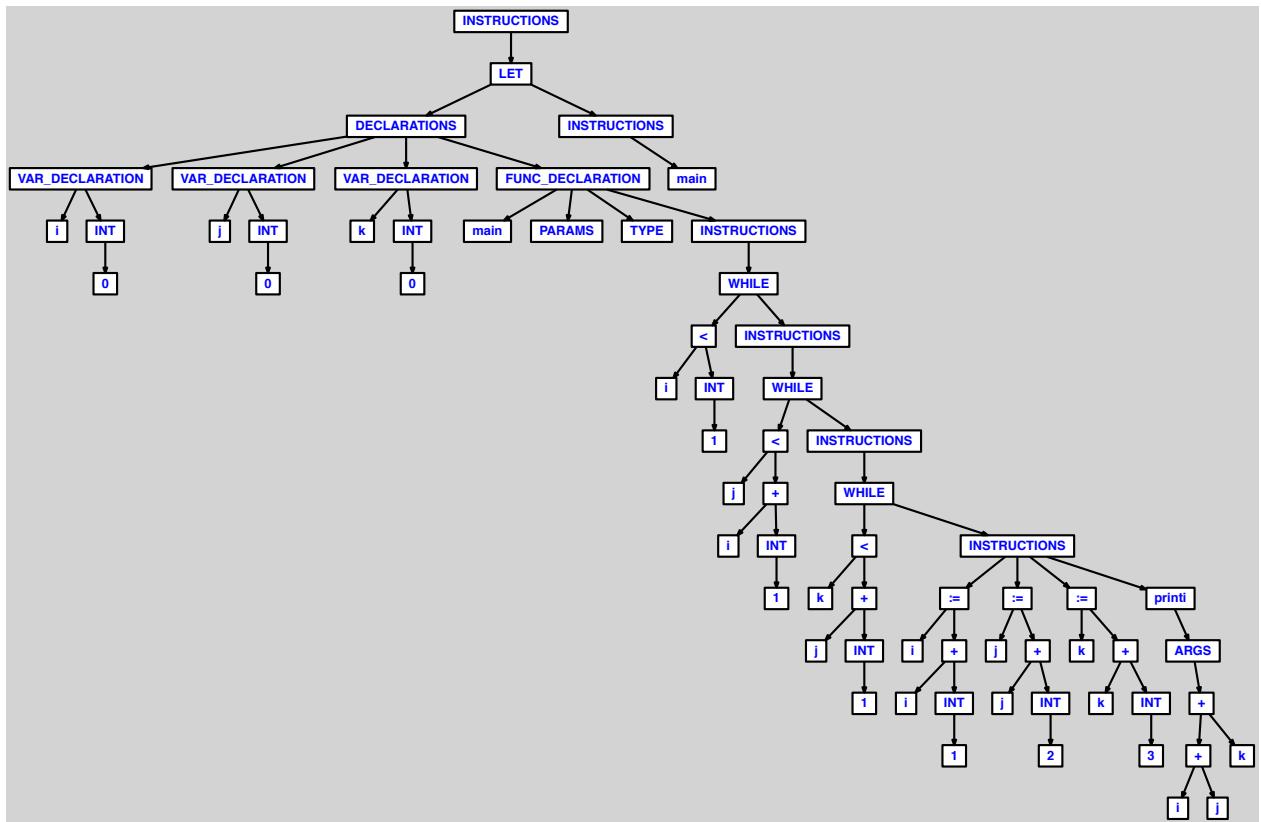
#### 14.2.10 <while> avec reutilisation de compteur

```
1 let
2   var i := 0
3   var j := 0
4
5   function main() =
6     while i < 1 do
7       while j < i+1 do
8         (i := i+1;
9          j := j+2;
10         printi(i+j))
11 in main() end
```



#### 14.2.11 <while> avec reutilisation de compteurs

```
1 let
2   var i := 0
3   var j := 0
4   var k := 0
5
6   function main() =
7     while i < 1 do
8       while j < i+1 do
9         while k < j+1 do
10           (i := i+1;
11            j := j+2;
12            k := k+3;
13            printi(i+j+k))
14 in main() end
```



#### 14.2.12 <while> avec iteration sur une chaine

```
1 let
2   var s := "t"
3
4   function main() =
5     while s <> "ttt" do
6       s := s+"t"
7   in main() end
```

