Rapport de tests

Table des matières

1 and **1**
1.1 KO . . . 1
1.1.1 <&> mal ecrit . . . 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 chaines . . . 13

2 break **15**
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
2.1.4 appel de <break>, avec corps de programme ne contenant pas <break> . . . 21
2.2 OK . . . 23
2.2.1 <break> dans un <for>, apres instruction . . . 23
2.2.2 <break> dans un <for>, sans instruction avant . . . 25
2.2.3 fonction executant seulement <break> . . . 27
2.2.4 <break> dans un <while>, apres instruction . . . 29
2.2.5 <break> dans un <while>, sans instruction avant . . . 31
2.2.6 <break> dans un <if then>, apres instruction . . . 33
2.2.7 <break> dans un <if then>, sans instruction avant . . . 35
2.2.8 <break> dans un <if then else>, apres instruction . . . 37
2.2.9 <break> dans un <if then else>, sans instruction avant . . . 39
2.2.10 <break> seul . . . 41
2.2.11 <break> seul, apres saut de ligne . . . 43

3 calcul **45**
3.1 KO . . . 45
3.1.1 addition avec operateur mal ecrit . . . 45
3.1.2 soustraction avec operateur mal ecrit . . . 47
3.1.3 multiplication avec operateur mal ecrit . . . 49
3.1.4 division avec operateur mal ecrit . . . 51
3.1.5 moins unaire avec operateur mal ecrit . . . 53
3.1.6 calcul a 2 termes avec oubli de parenthese gauche . . . 55
3.1.7 calcul a 2 termes avec oubli de parenthese droite . . . 57
3.1.8 calcul a 3 termes avec oubli de parenthese gauche . . . 59
3.1.9 calcul a 3 termes avec oubli de parenthese droite . . . 61
3.1.10 calcul a 4 termes avec oubli de parenthese gauche . . . 63
3.1.11 calcul a 4 termes avec oubli de parenthese droite . . . 65
3.1.12 parenthesage sans contenu . . . 67
3.1.13 expression parenthesee avec <;> en trop . . . 69
3.1.14 expression parenthesee avec oubli de <;> . . . 71
3.1.15 expression non parenthesee avec oubli de <;> . . . 73
3.1.16 expression non parenthesee avec <;> en trop . . . 75
3.1.17 <;> sans instruction avant . . . 77
3.1.18 2 instructions sans parentheses . . . 79
3.1.19 3 instructions sans parentheses . . . 81
3.1.20 concatenation d'entier et de chaine . . . 83
3.1.21 concatenation de 2 chaines . . . 85
3.2 OK . . . 87
3.2.1 addition simple, a 2 termes . . . 87

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

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

3.2.98 parenthesage de 2 instructions . . . 281
3.2.99 parenthesage de 3 instructions . . . 283
3.2.100 1 instruction sans parentheses . . . 285

4 commentaire **287**
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

5 comparaison **329**
5.1 KO . . . 329
5.1.1 comparaison avec oubli d'operande gauche . . . 329
5.1.2 comparaison avec oubli d'operateur . . . 331
5.1.3 comparaison avec oubli d'operateur 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

6 espace **367**
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

7 for **385**
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

8 function **427**
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 separes . . . 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

9 if **497**
9.1 KO . . . 497
9.1.1 <if then> sans instruction . . . 497
9.1.2 <if then> avec ligne vide . . . 499
9.1.3 incrementation dans la condition <if then> . . . 501
9.1.4 affectation d'entier dans la condition <if then> . . . 503
9.1.5 affectation de chaine dans la condition <if then> . . . 505
9.1.6 <if then> avec oubli du <if> . . . 507
9.1.7 <if then> avec oubli du <then> . . . 509
9.1.8 <if then> avec oubli de la condition . . . 511
9.1.9 <if then else> avec oubli de la condition . . . 513
9.1.10 <if then else> sans instruction dans <else> . . . 515
9.1.11 <if then else> avec ligne vide dans <else> . . . 517
9.2 OK . . . 519
9.2.1 <if then> avec condition entiere . . . 519
9.2.2 <if then> avec simple condition . . . 521
9.2.3 <if then> avec double-condition . . . 523
9.2.4 <if then> avec triple-condition . . . 525
9.2.5 double-imbrication de <if then> . . . 527
9.2.6 triple-imbrication de <if then> . . . 529
9.2.7 double-imbrication de <if then else> . . . 531
9.2.8 triple-imbrication de <if then else> . . . 533

10 let **535**
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, et 3 instructions et appels de fonction . . . 557
10.2.8 <let> avec double-imbrication . . . 559
10.2.9 <let> avec triple-imbrication . . . 561

11 or **563**
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

12 return **577**
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

13 var **599**
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 egales . . . 611
13.1.8 double-declaration avec valeurs differentes . . . 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'operation logique sur entiers . . . 621
13.1.13 affectation d'operation 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

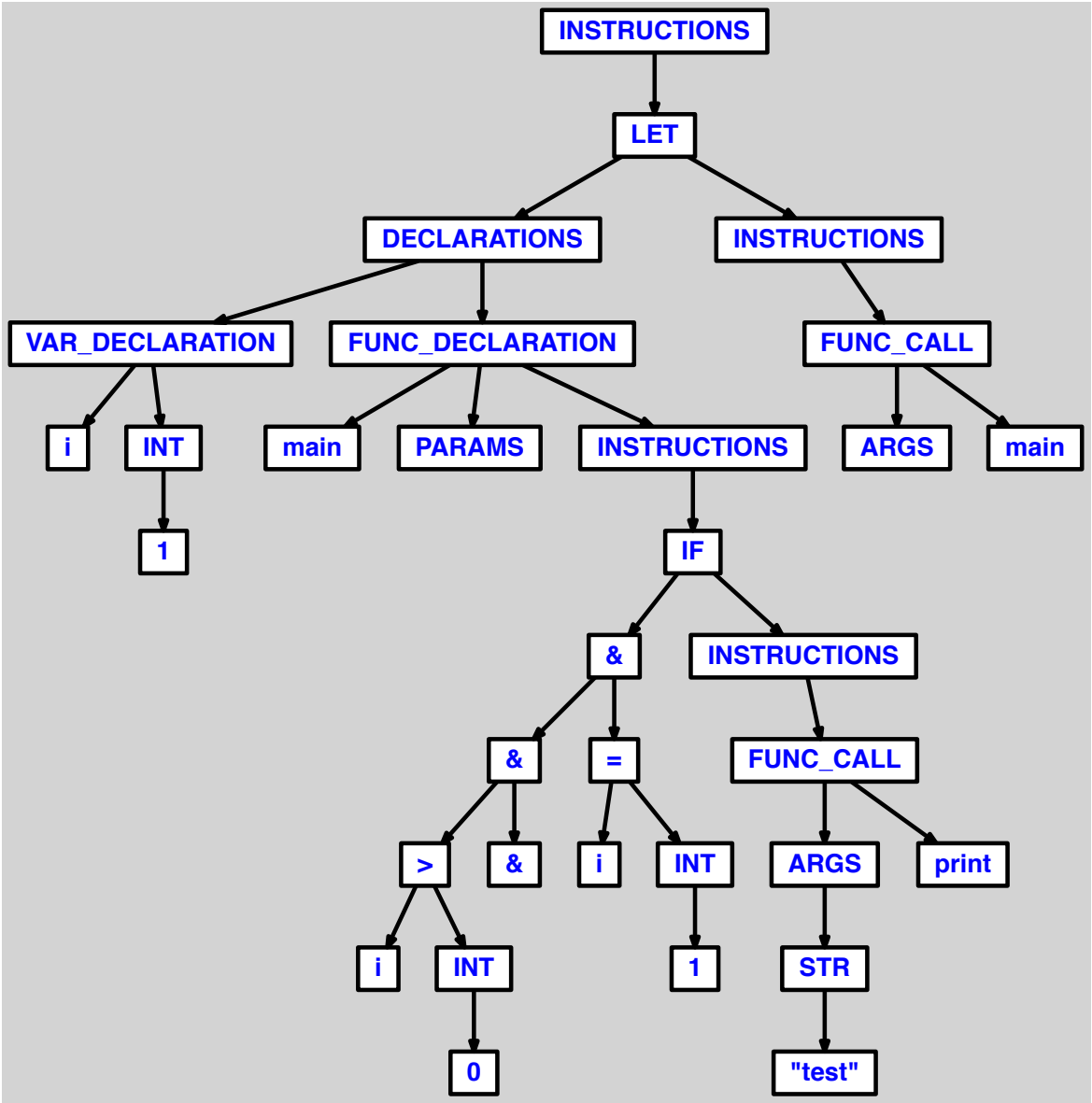
14 while **669**
14.1 KO . . . 669
14.1.1 <while> sans instruction . . . 669
14.1.2 <while> avec ligne vide . . . 671
14.1.3 <while> avec affectation d'entier . . . 673
14.1.4 <while> avec affectation de chaine . . . 675
14.1.5 <while> avec oubli du compteur . . . 677
14.1.6 <while> avec oubli du <while> . . . 679
14.1.7 <while> avec oubli du <do> . . . 681
14.1.8 <while> avec oubli de la condition . . . 683
14.2 OK . . . 685
14.2.1 <while> avec iteration croissante . . . 685
14.2.2 <while> avec iteration decroissante . . . 687
14.2.3 <while> avec double-condition . . . 689
14.2.4 <while> avec triple-condition . . . 691
14.2.5 <while> avec condition entiere . . . 693
14.2.6 <while> avec double-imbrication . . . 695
14.2.7 <while> avec triple-imbrication . . . 697
14.2.8 <while> avec reutilisation de compteur . . . 699
14.2.9 <while> avec reutilisation de compteurs . . . 701
14.2.10 <while> avec iteration sur une chaine . . . 703

1 and

1.1 KO

1.1.1 <&> mal ecrit

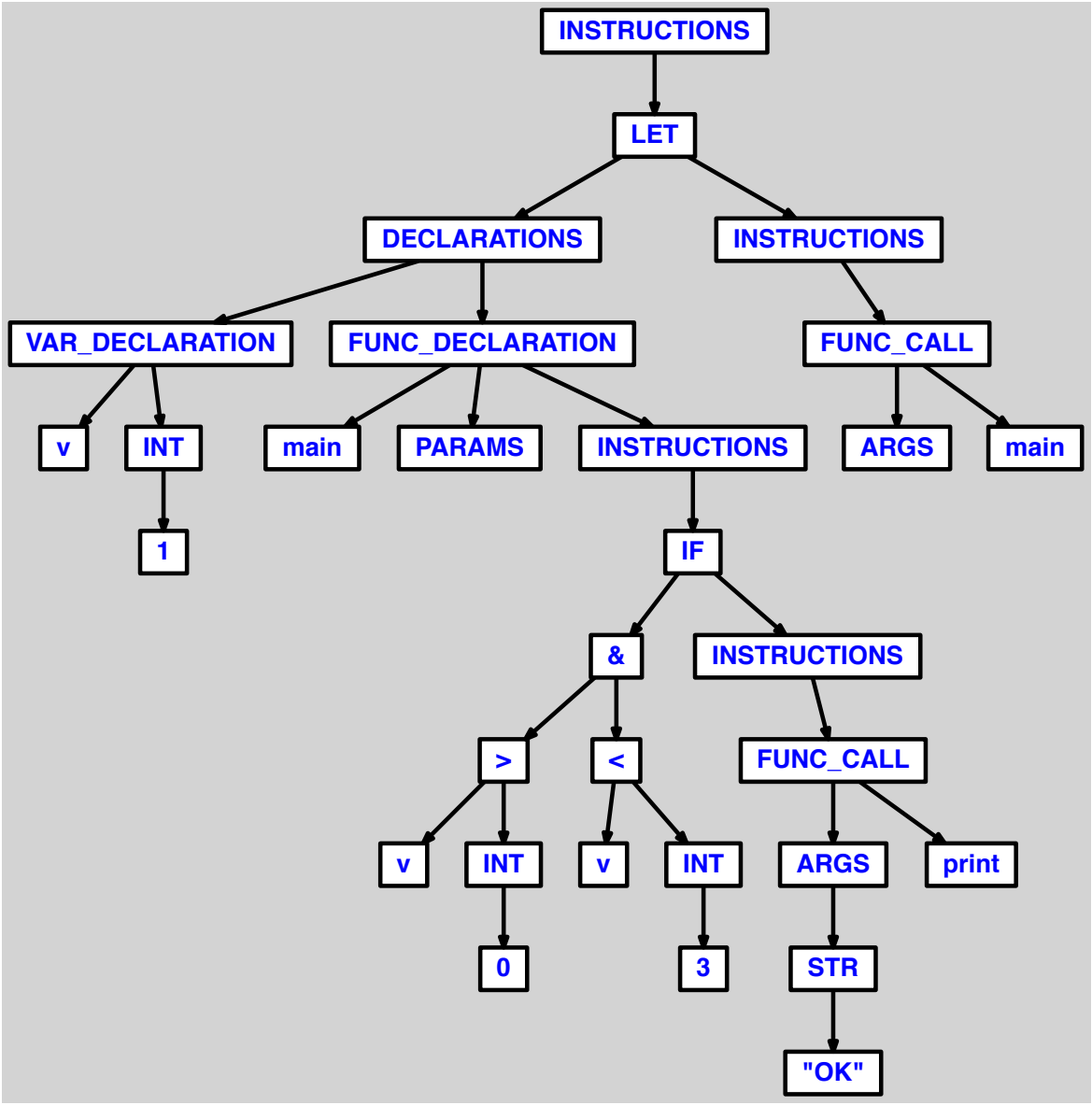
```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
&
INSTRUCTIONS
&
=
FUNC_CALL
>
&
i
INT
ARGS
print
i
INT
1
STR
0
"test"

1.2 OK

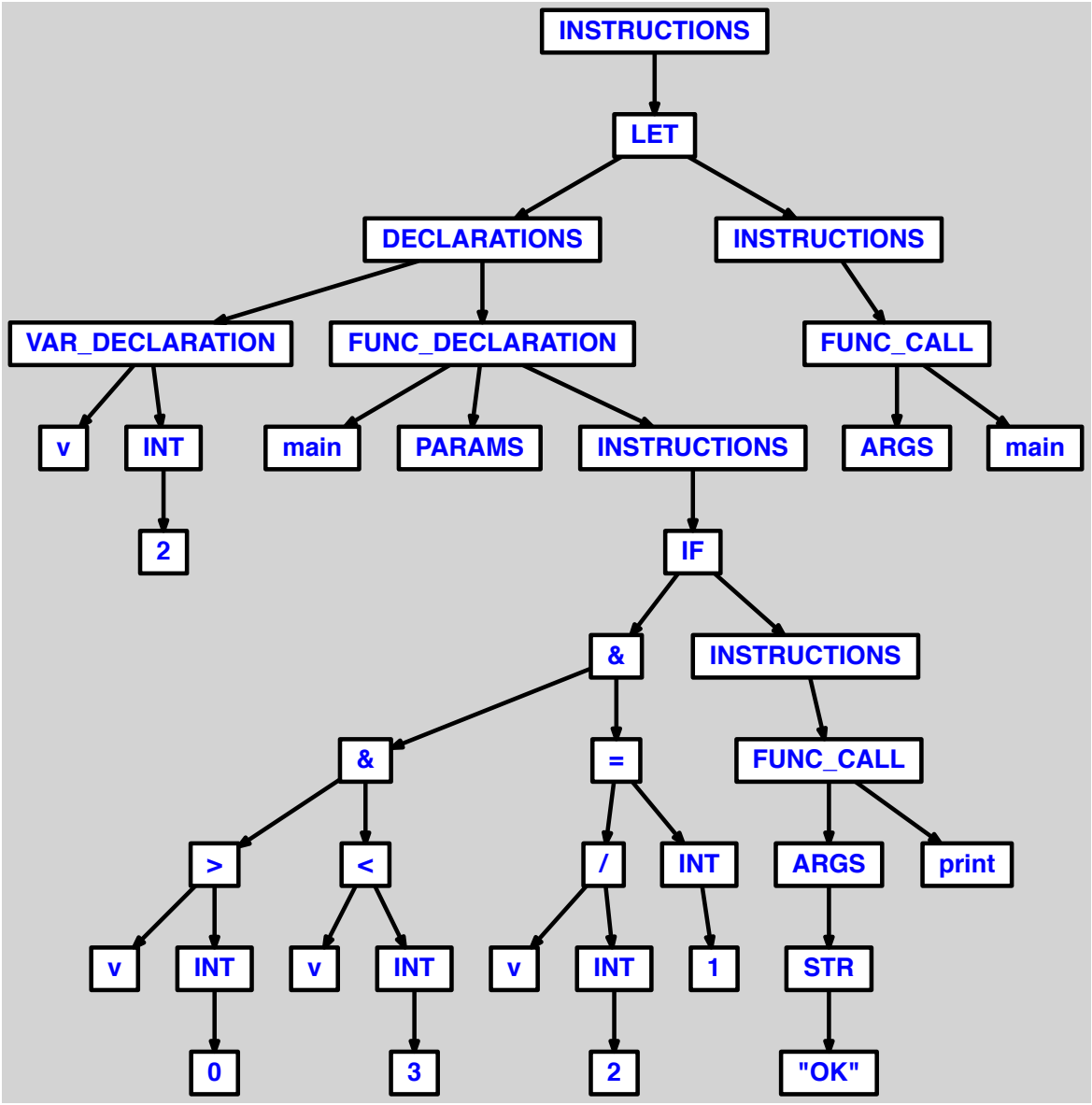
1.2.1 condition avec 1 <&>

```
let
  var v := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
&
INSTRUCTIONS
>
<
FUNC_CALL
v
INT
v
INT
ARGS
print
0
3
STR
"OK"

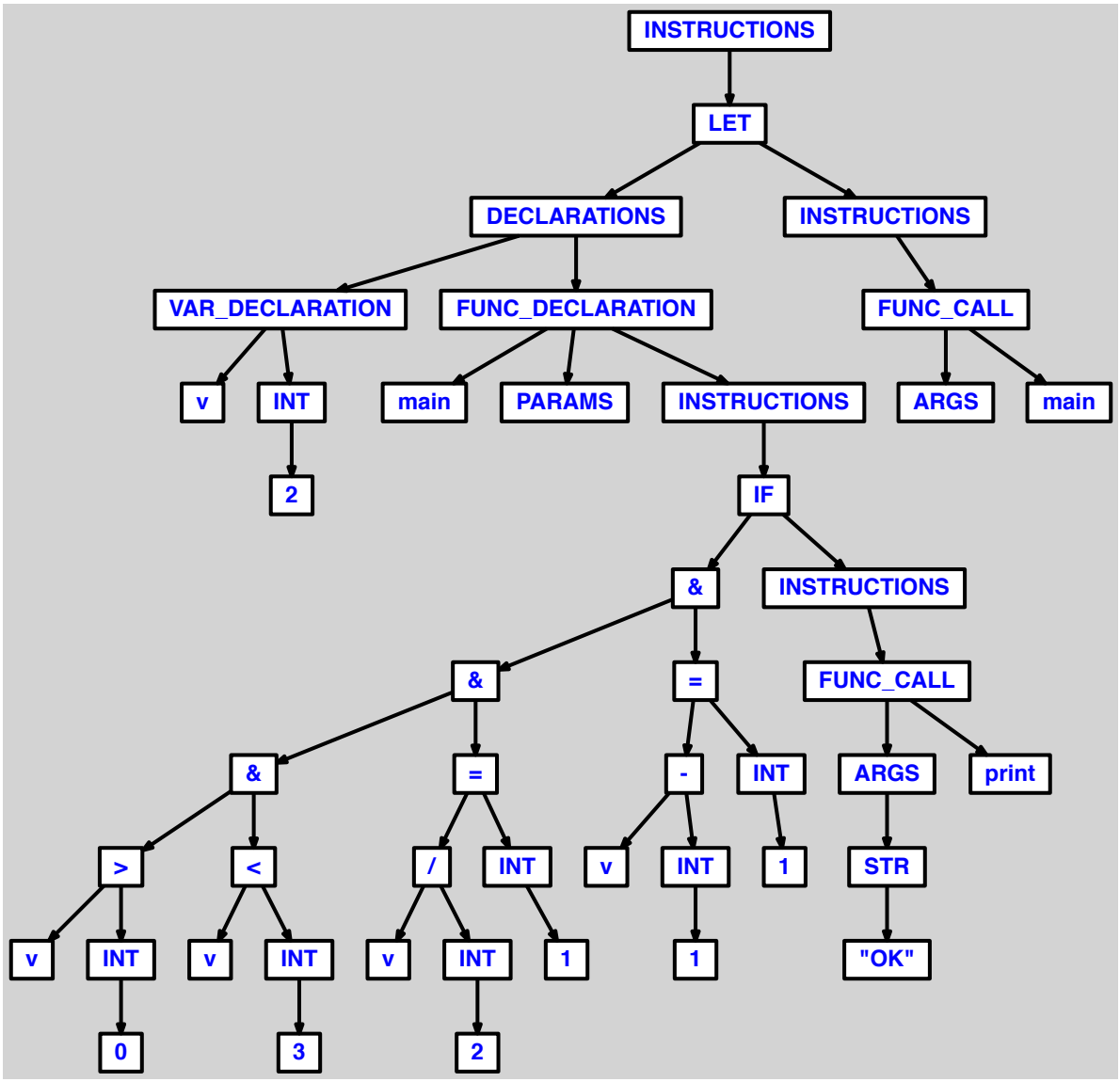
1.2.2 condition avec 2 <&>

```
let
  var v := 2

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
2
IF
&
INSTRUCTIONS
&
=
FUNC_CALL
>
<
/
INT
ARGS
print
v
INT
v
INT
v
INT
1
STR
0
3
2
"OK"

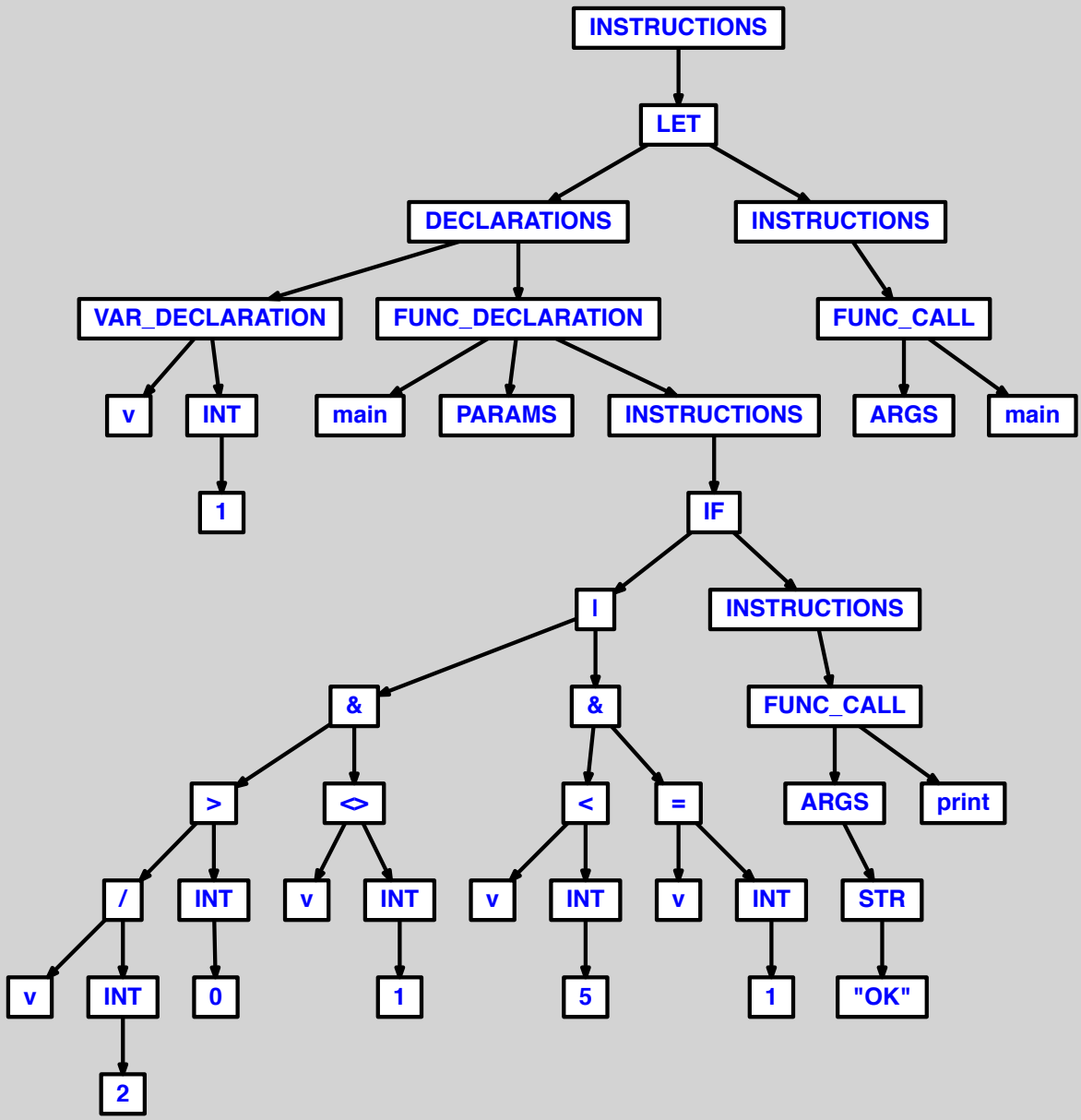
1.2.3 condition avec 3 <&>

```
let
  var v := 2

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
2
IF
&
INSTRUCTIONS
&
=
FUNC_CALL
&
=
-
INT
ARGS
print
>
<
/
INT
v
INT
1
STR
v
INT
v
INT
v
INT
1
1
"OK"
0
3
2

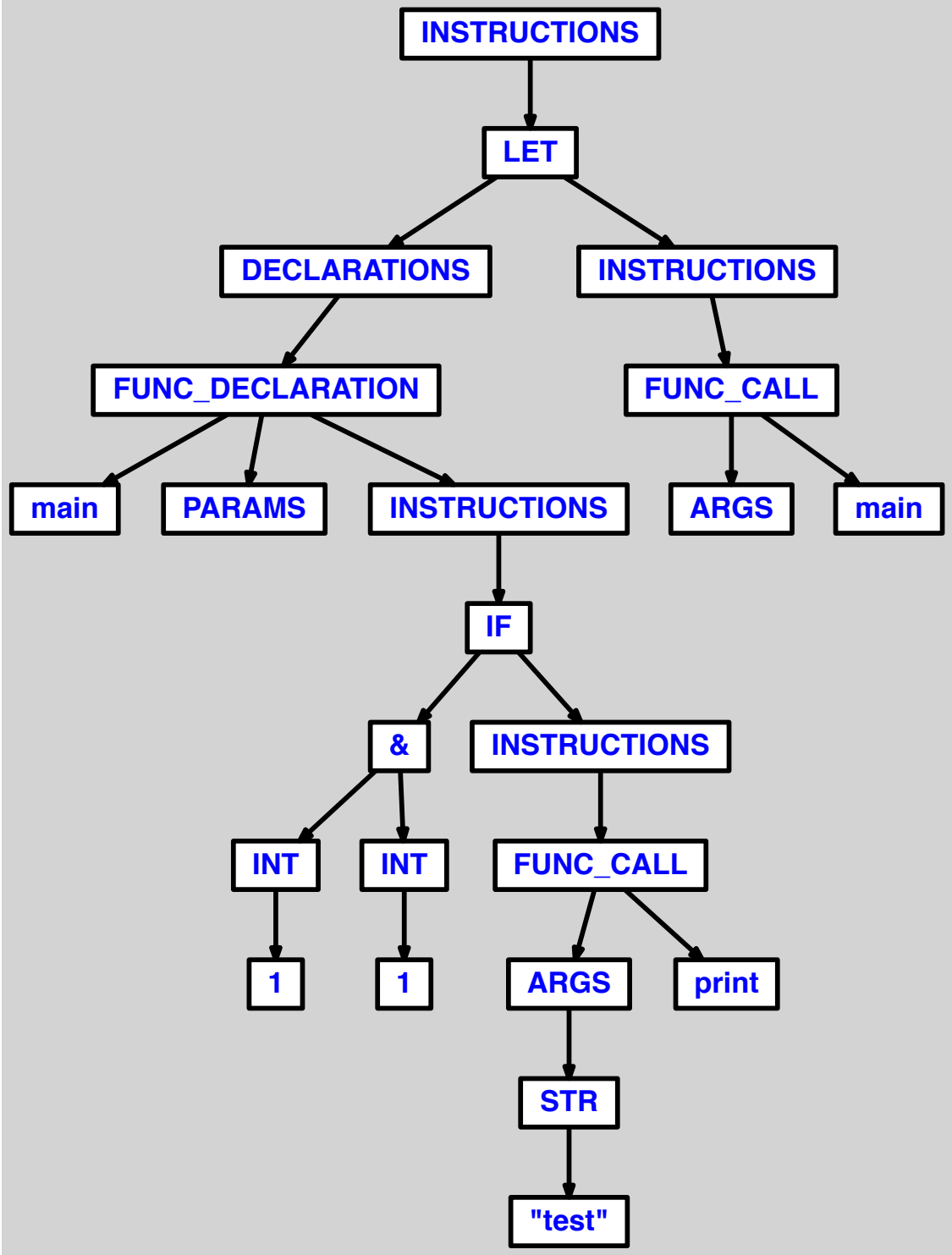
1.2.4 condition avec 2 <&> et 1 <|>

```
let
  var v := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
|
INSTRUCTIONS
&
&
FUNC_CALL
>
<>
<
=
ARGS
print
/
INT
v
INT
v
INT
v
INT
STR
v
INT
0
1
5
1
"OK"
2

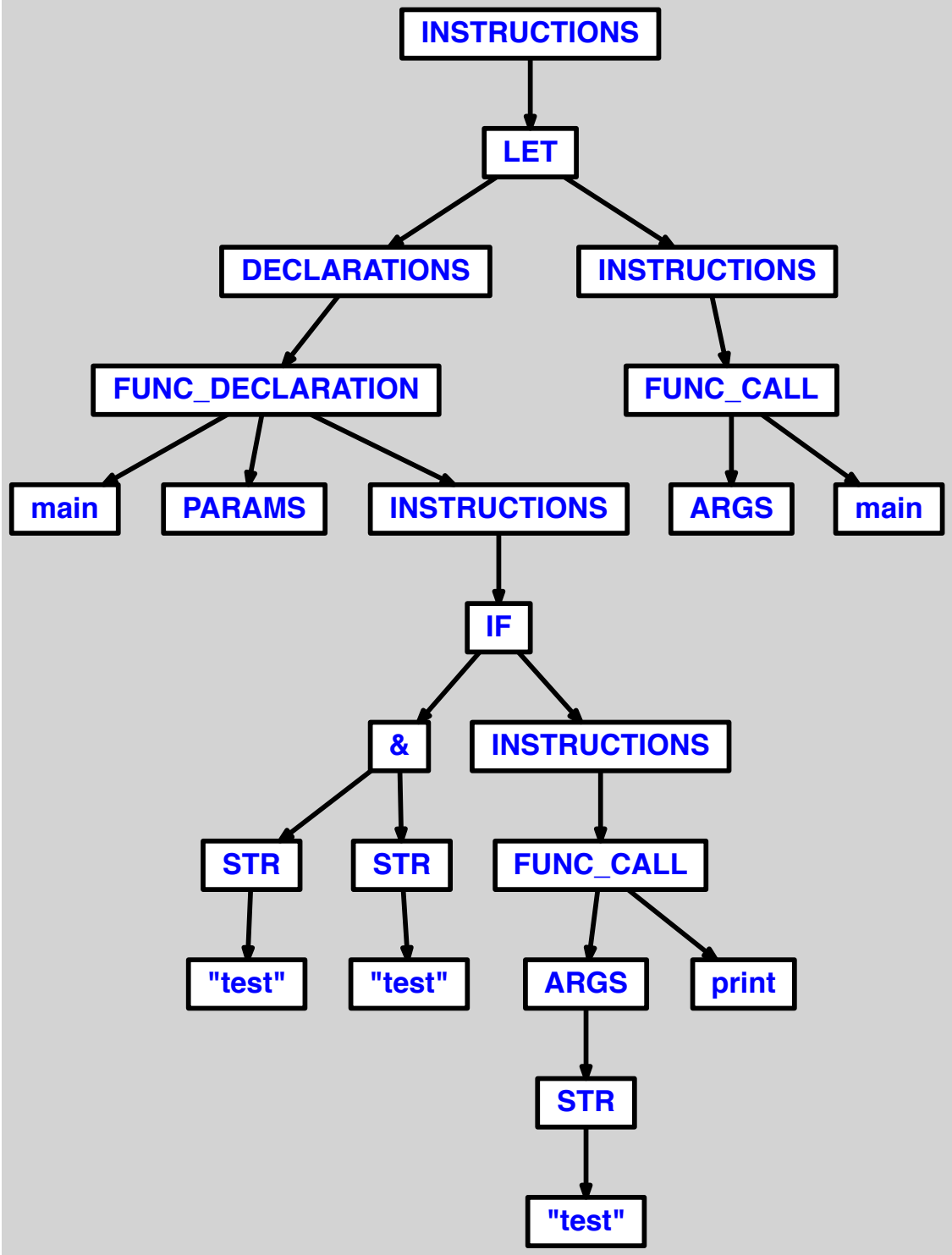
1.2.5 <&> avec des entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
&
INSTRUCTIONS
INT
INT
FUNC_CALL
1
1
ARGS
print
STR
"test"

1.2.6 <&> avec des chaines

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

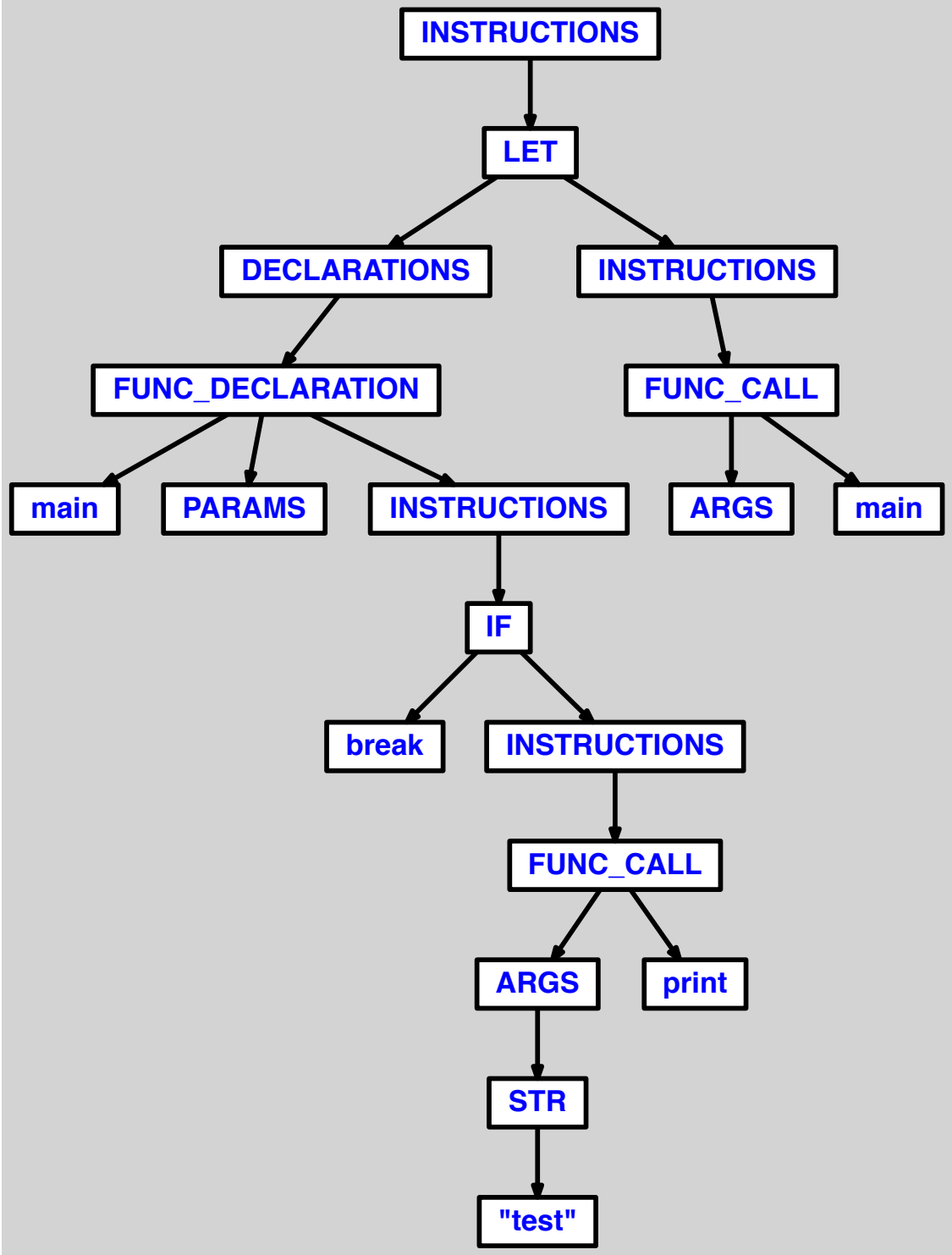

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
&
INSTRUCTIONS
STR
STR
FUNC_CALL
"test"
"test"
ARGS
print
STR
"test"

2 break

2.1 KO

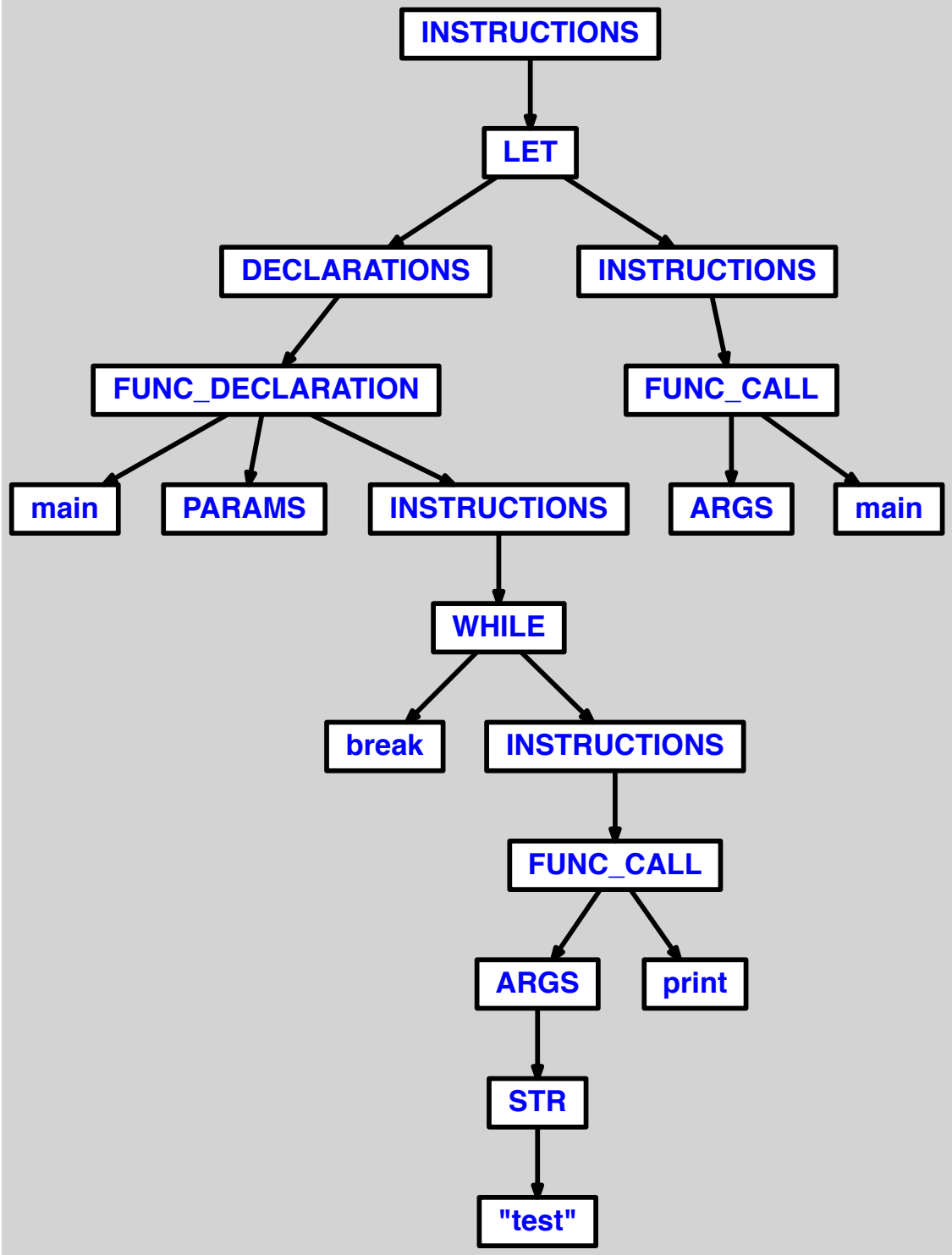
2.1.1 <break> comme condition dans <if>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
break
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"test"

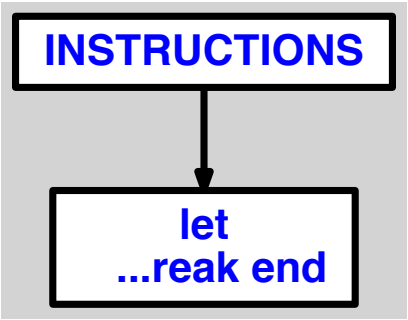
2.1.2 <break> comme condition dans <while>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
break
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"test"

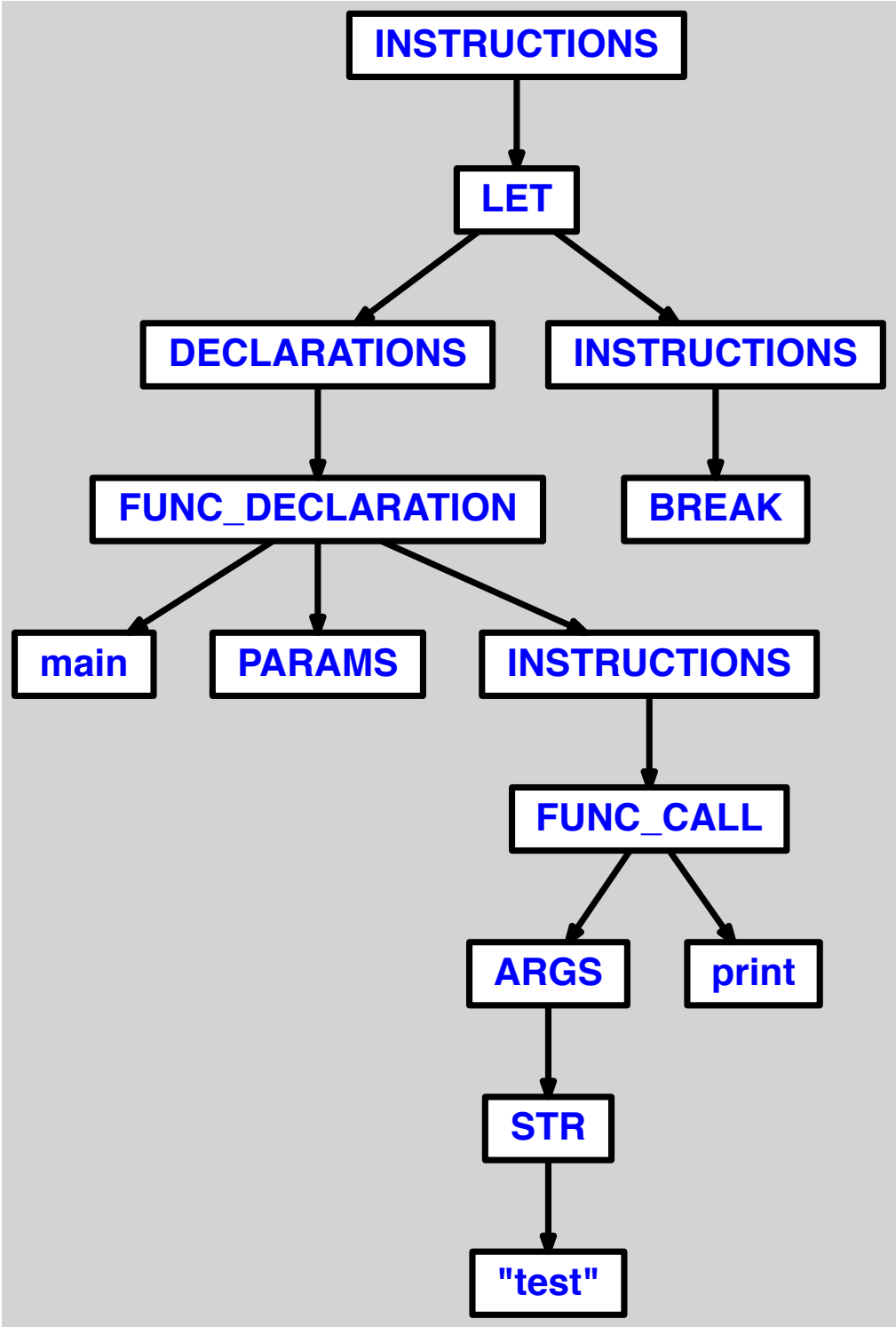
2.1.3 appel de <break>, avec corps de programme contenant <break>

```
let
  break
in break end
```


INSTRUCTIONS
let
...reak end

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

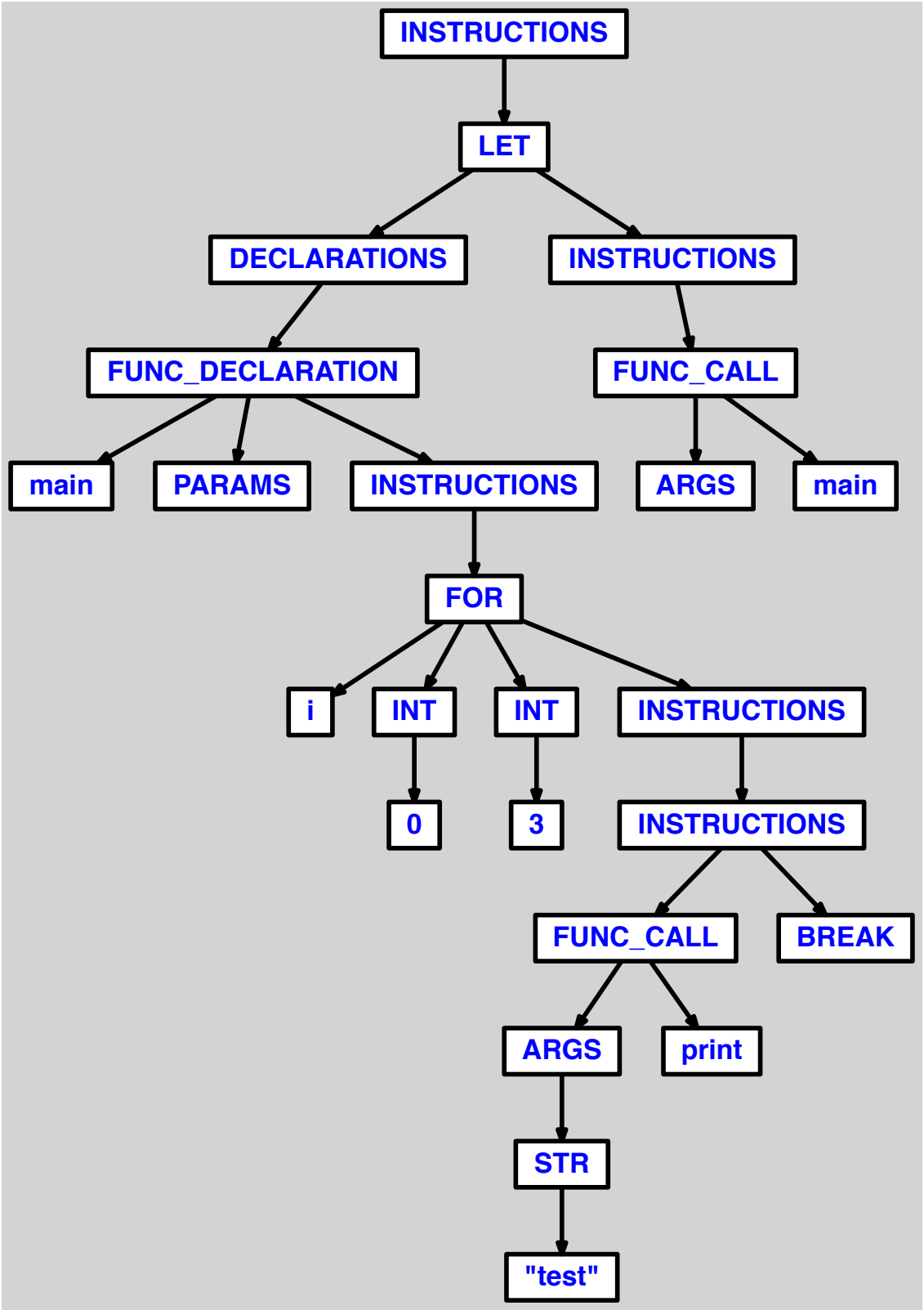
```
let
  function main() = print("test")
in break end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
BREAK
main
PARAMS
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"test"

2.2 OK

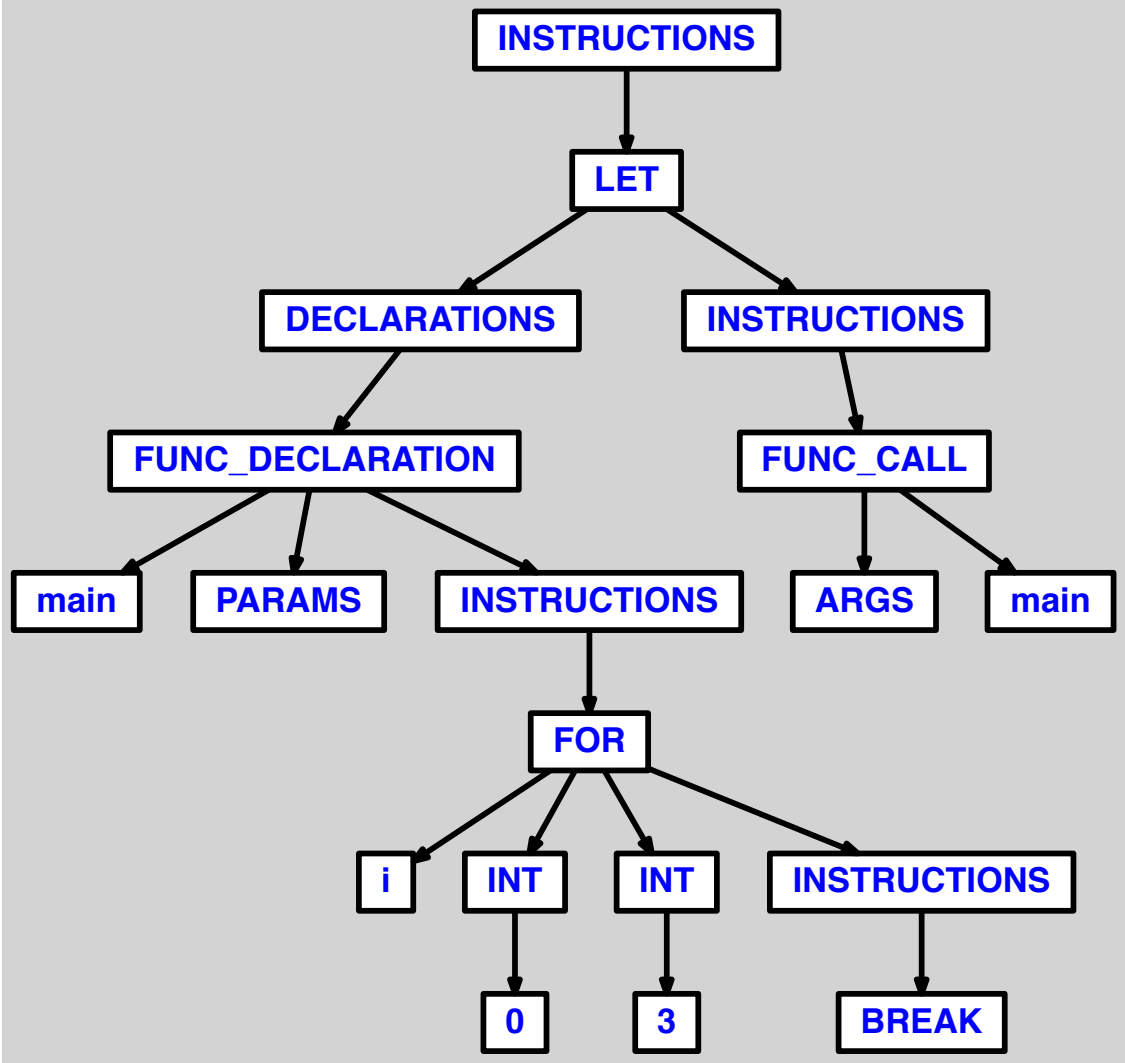
2.2.1 <break> dans un <for>, apres instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
INSTRUCTIONS
FUNC_CALL
BREAK
ARGS
print
STR
"test"

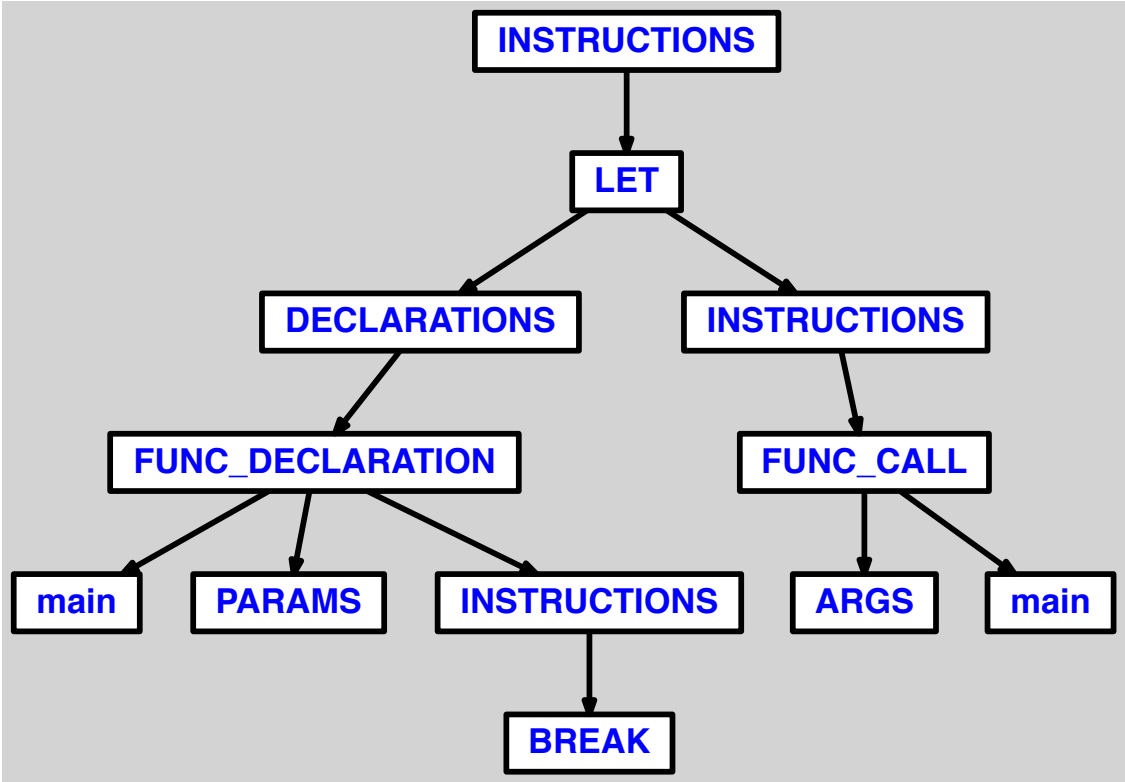
2.2.2 <break> dans un <for>, sans instruction avant

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
BREAK

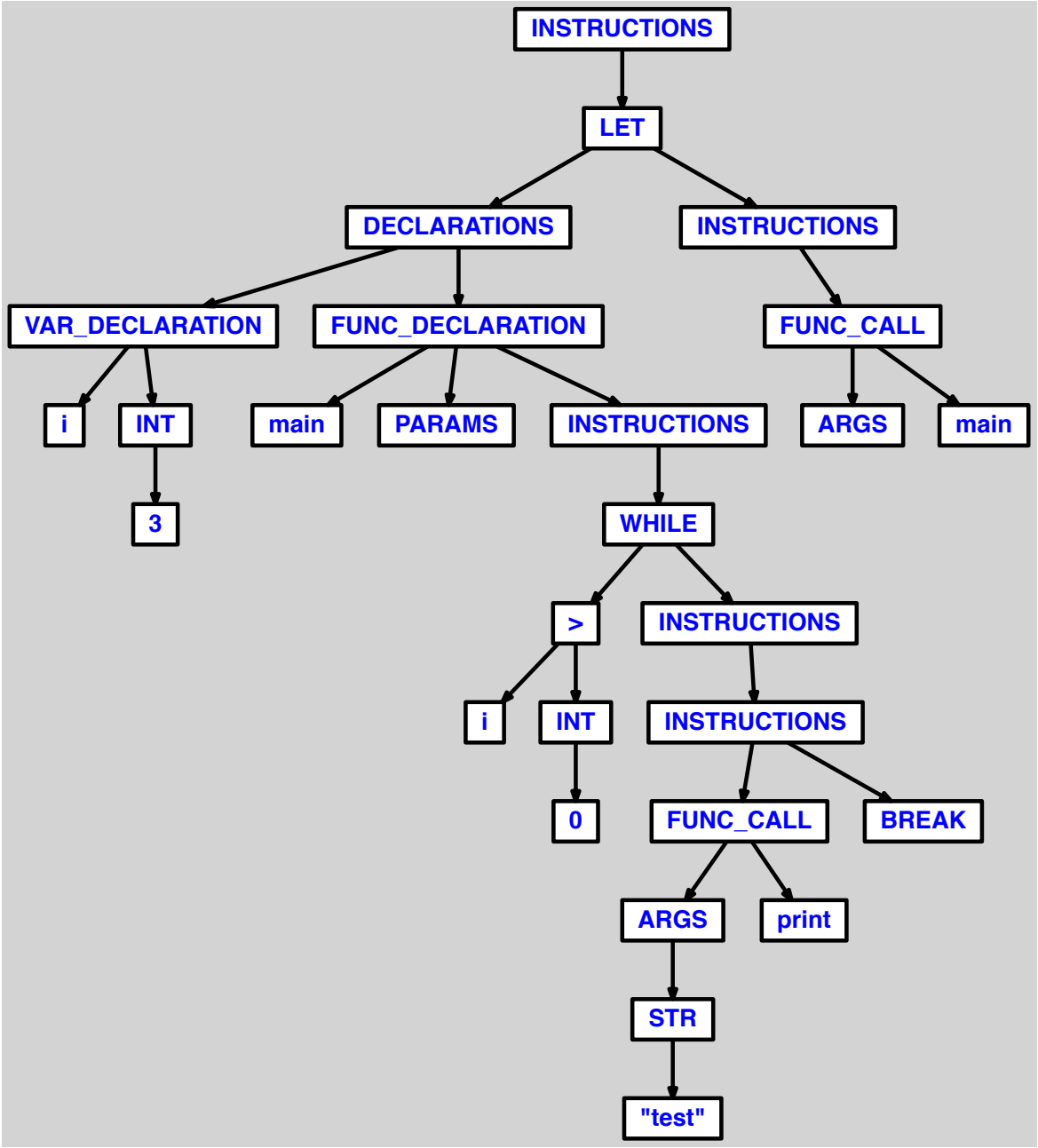
2.2.3 fonction executant seulement <break>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
BREAK

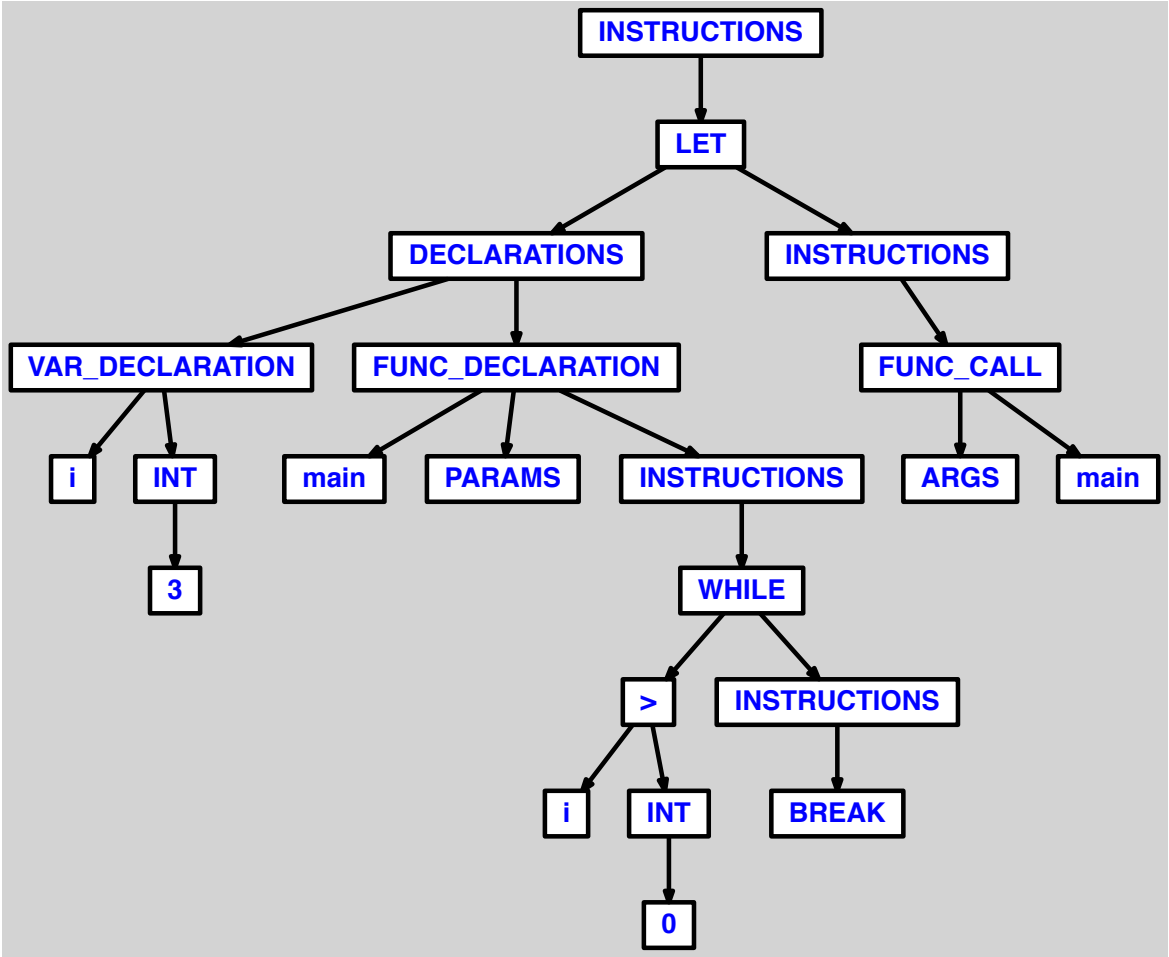
2.2.4 <break> dans un <while>, apres instruction

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
3
WHILE
>
INSTRUCTIONS
i
INT
INSTRUCTIONS
0
FUNC_CALL
BREAK
ARGS
print
STR
"test"

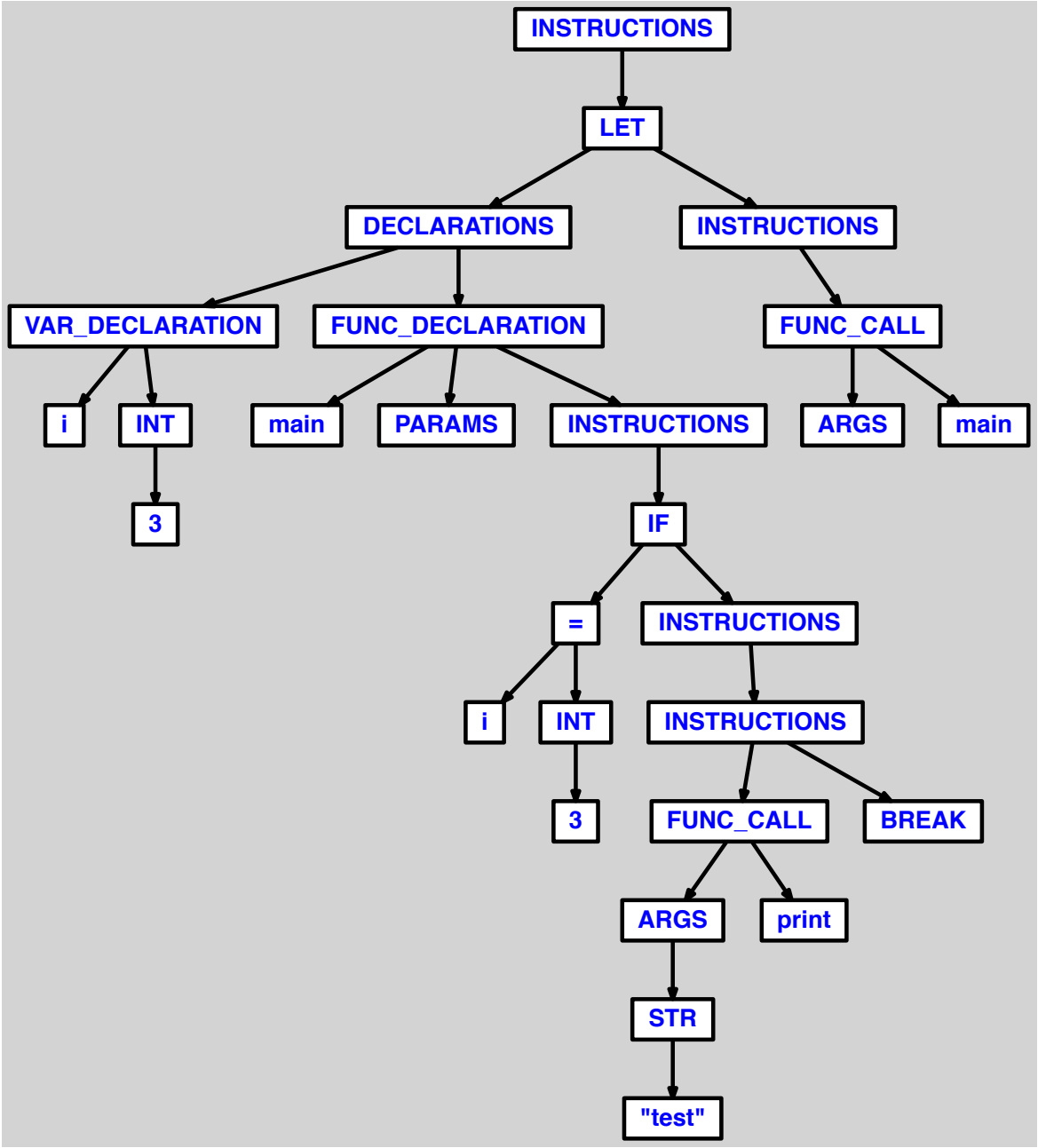
2.2.5 <break> dans un <while>, sans instruction avant

```
let
  var i := 3

  function main() =
    while i > 0 do
      break
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
3
WHILE
>
INSTRUCTIONS
i
INT
BREAK
0

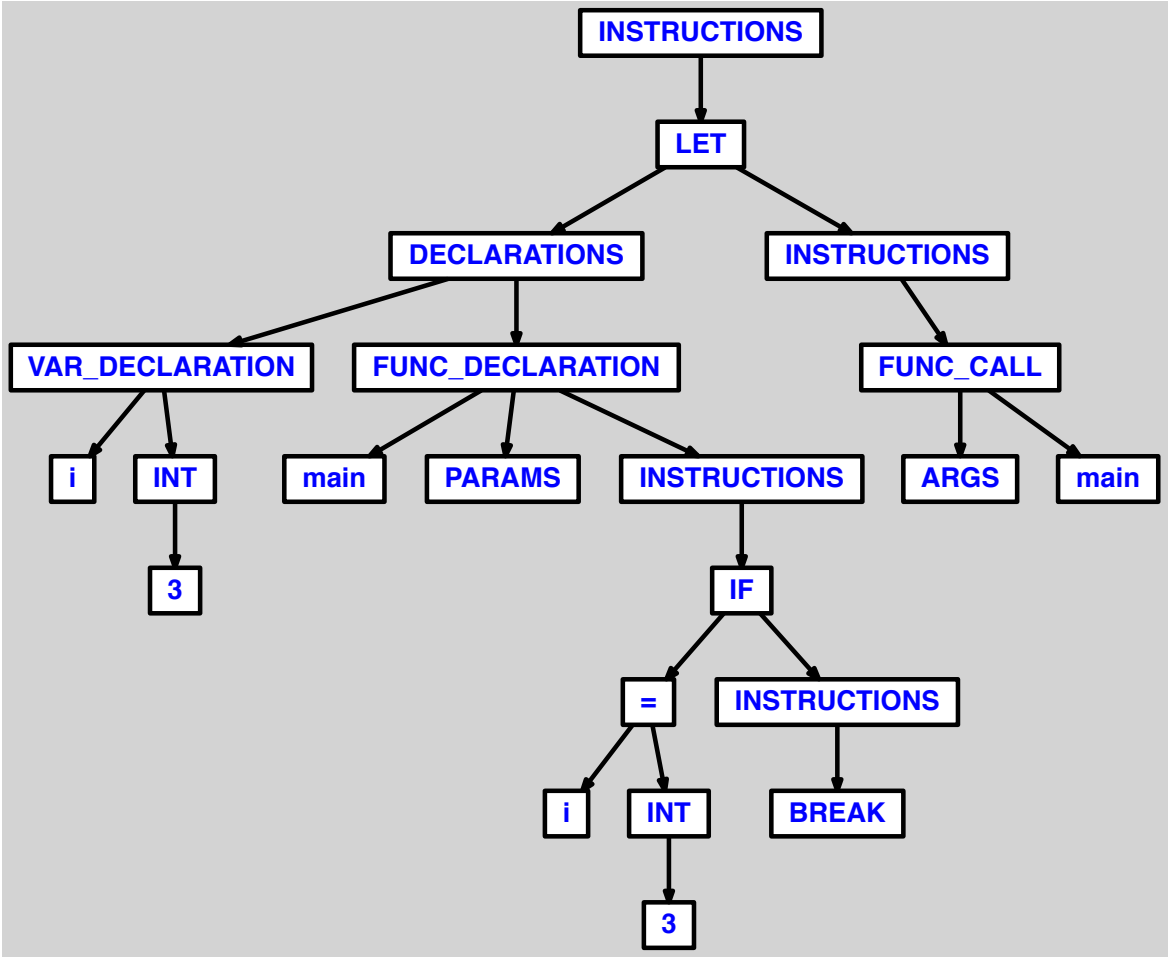
2.2.6 <break> dans un <if then>, apres instruction

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
3
IF
=
INSTRUCTIONS
i
INT
INSTRUCTIONS
3
FUNC_CALL
BREAK
ARGS
print
STR
"test"

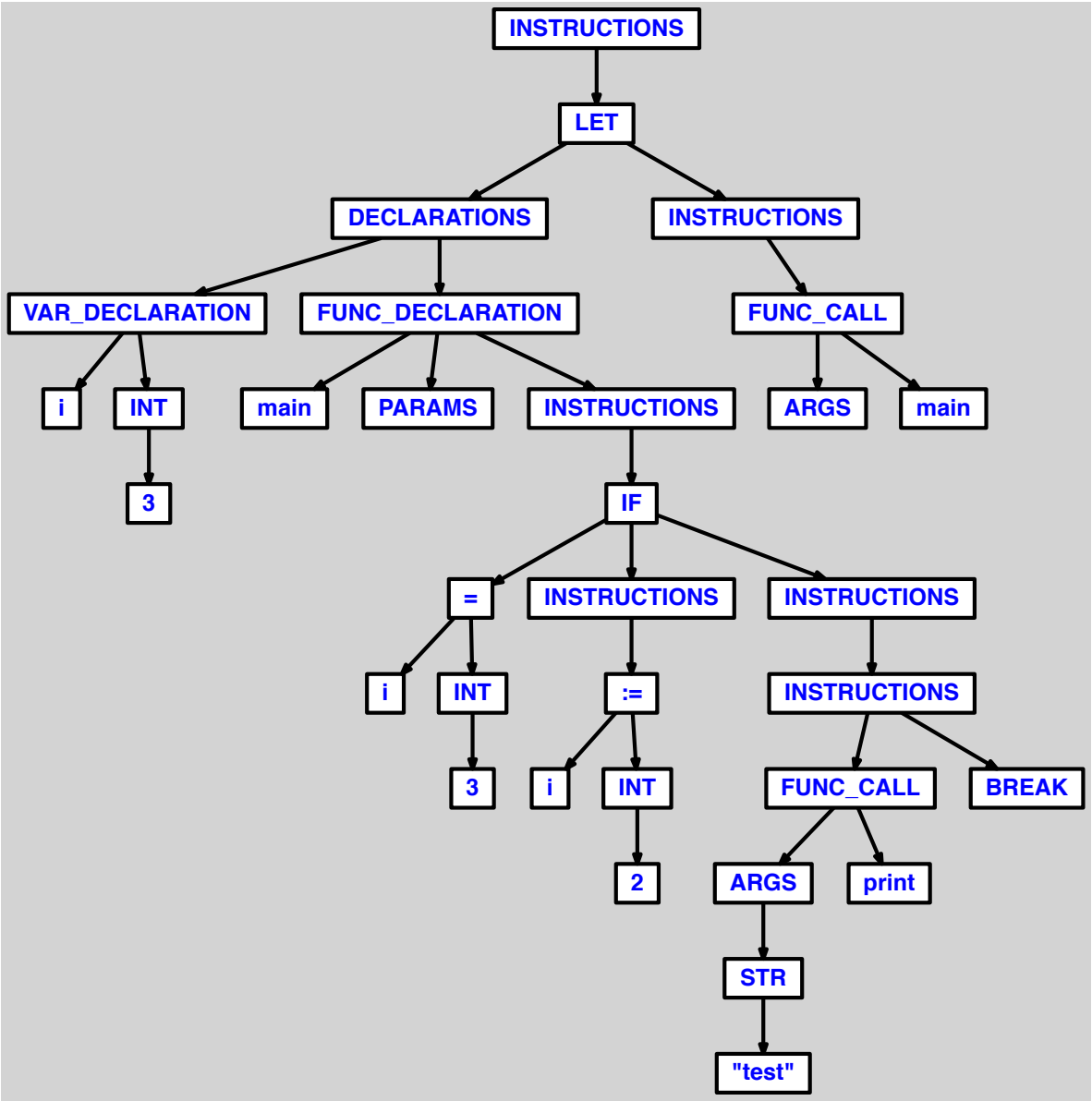
2.2.7 <break> dans un <if then>, sans instruction avant

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
3
IF
=
INSTRUCTIONS
i
INT
BREAK
3

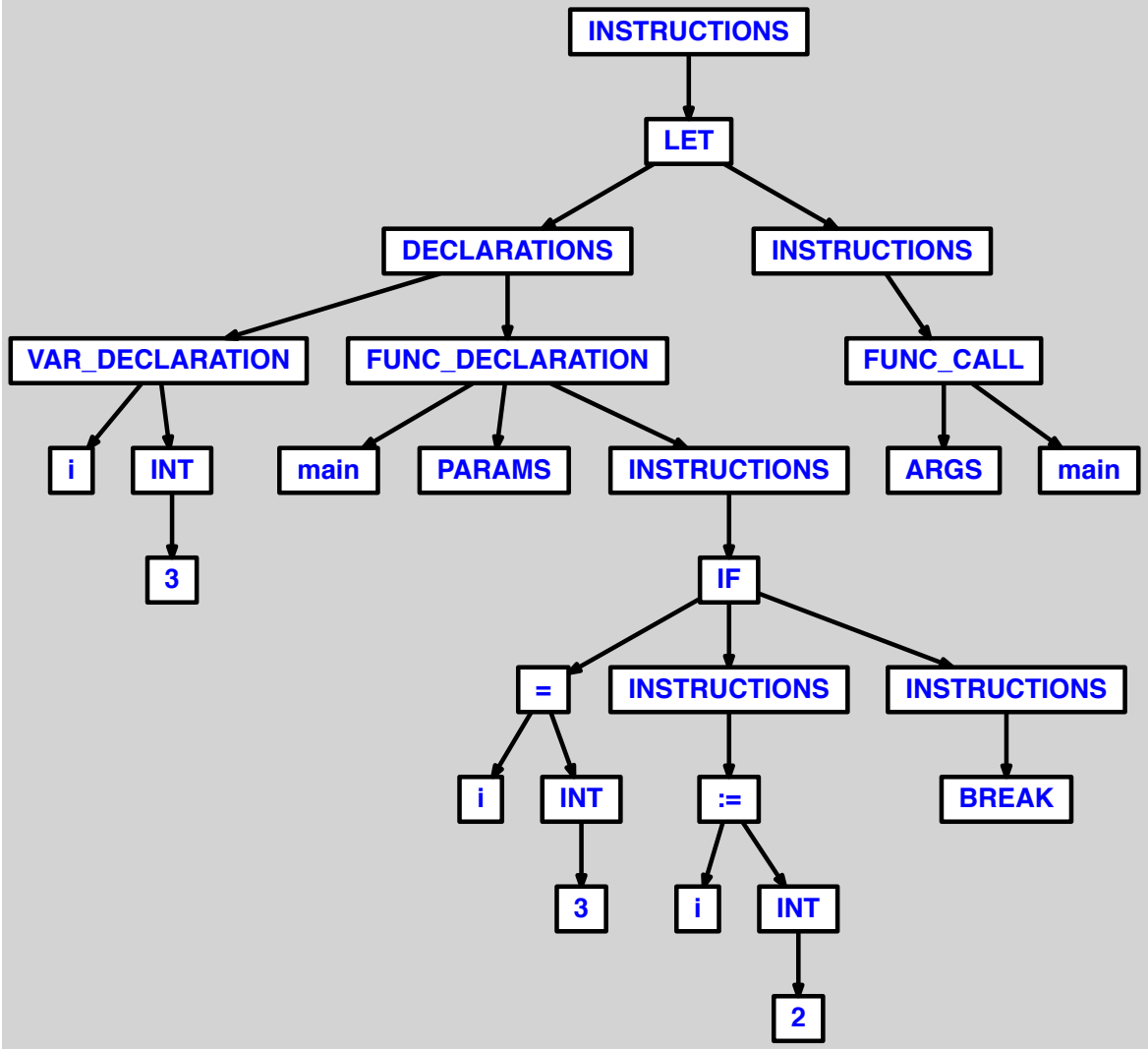
2.2.8 <break> dans un <if then else>, apres instruction

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
3
IF
=
INSTRUCTIONS
INSTRUCTIONS
i
INT
:=
INSTRUCTIONS
3
i
INT
FUNC_CALL
BREAK
2
ARGS
print
STR
"test"

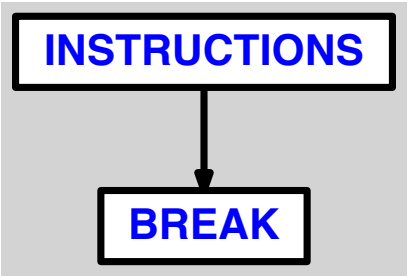
2.2.9 <break> dans un <if then else>, sans instruction avant

```
let
  var i := 3

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
3
IF
=
INSTRUCTIONS
INSTRUCTIONS
i
INT
:=
BREAK
3
i
INT
2

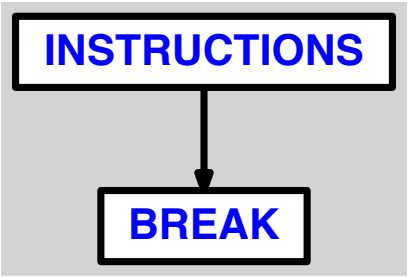
2.2.10 <break> seul

```
break
```


INSTRUCTIONS
BREAK

2.2.11 <break> seul, apres saut de ligne

```

break
```

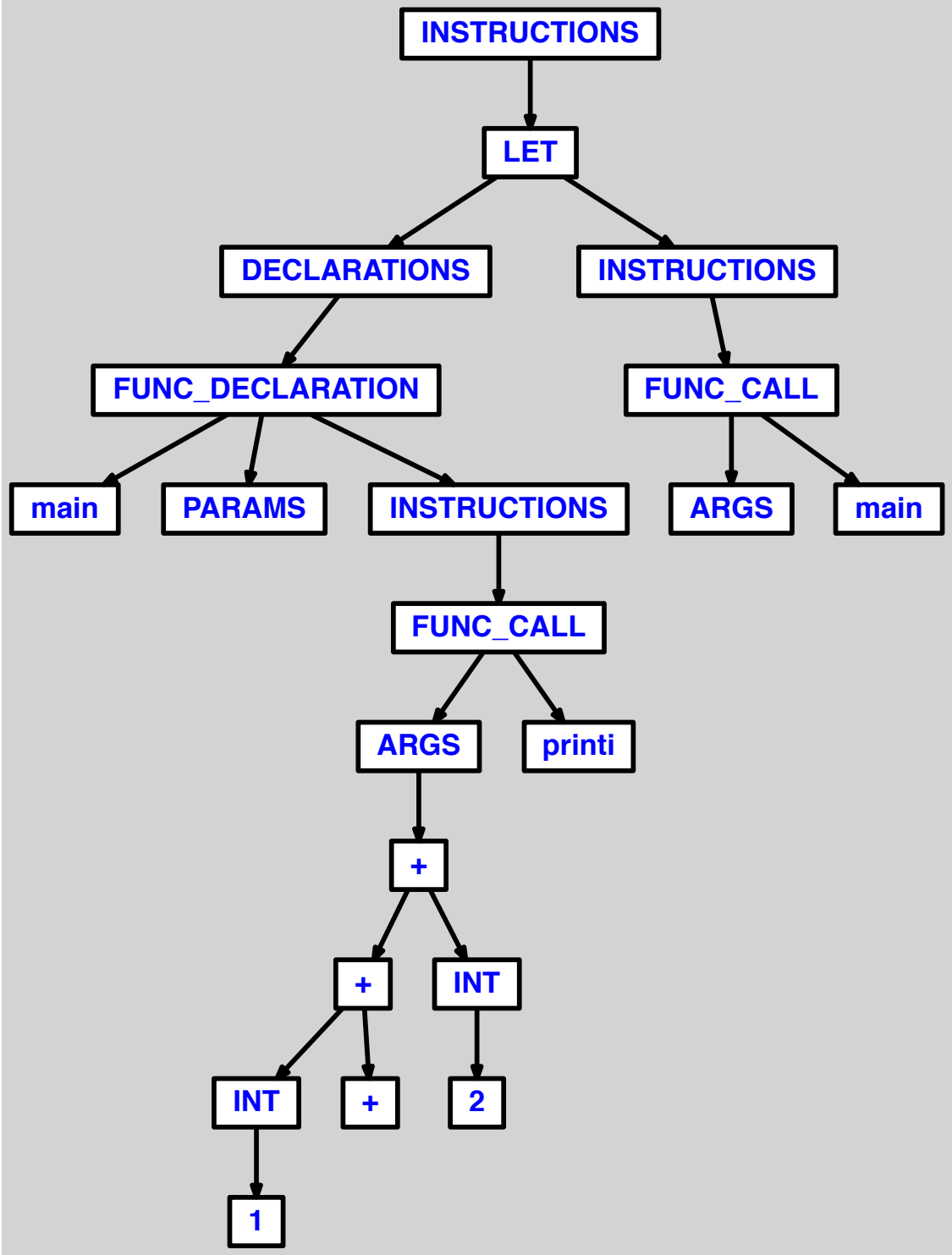

INSTRUCTIONS
BREAK

3 calcul

3.1 KO

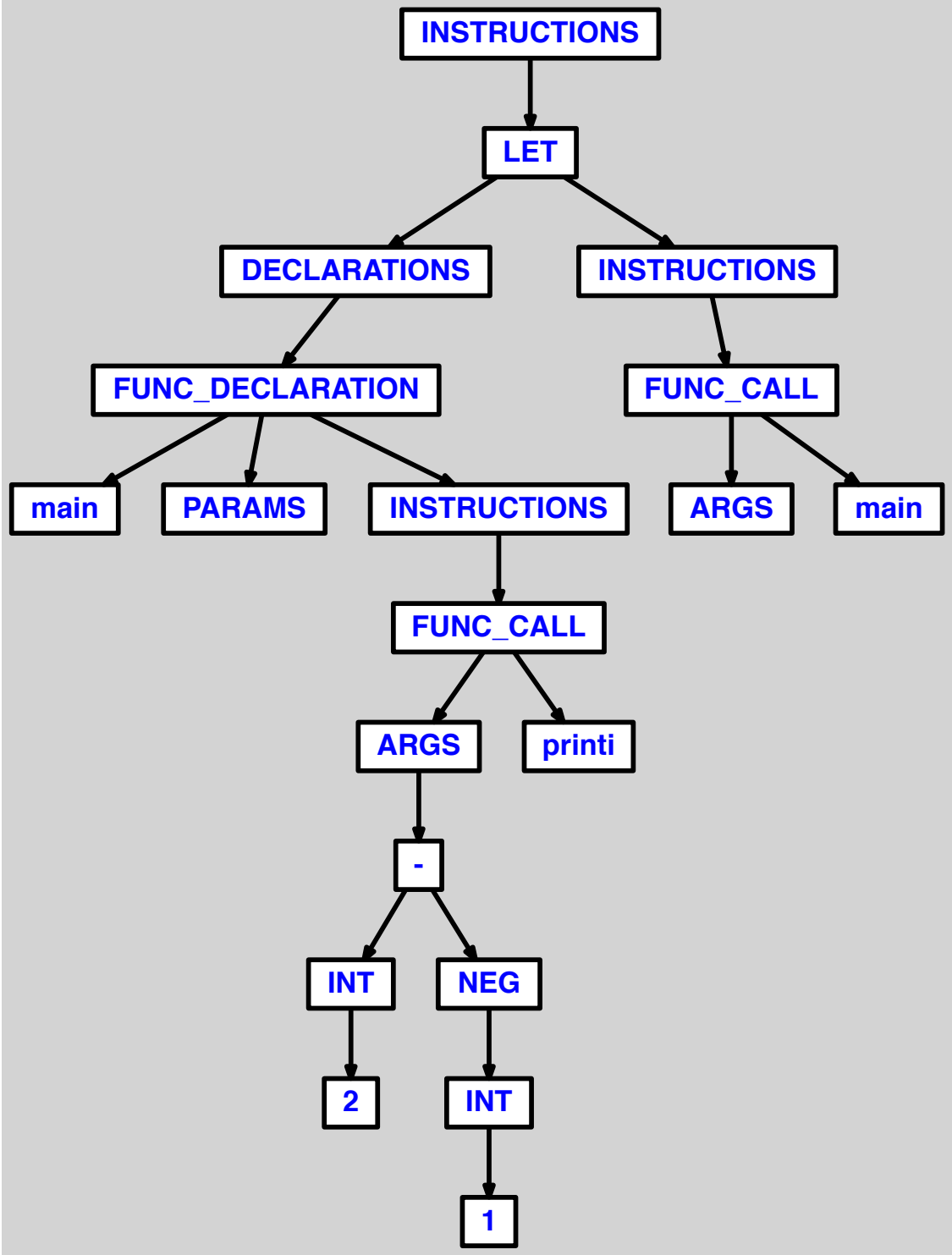
3.1.1 addition avec operateur mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
+
INT
INT
+
2
1

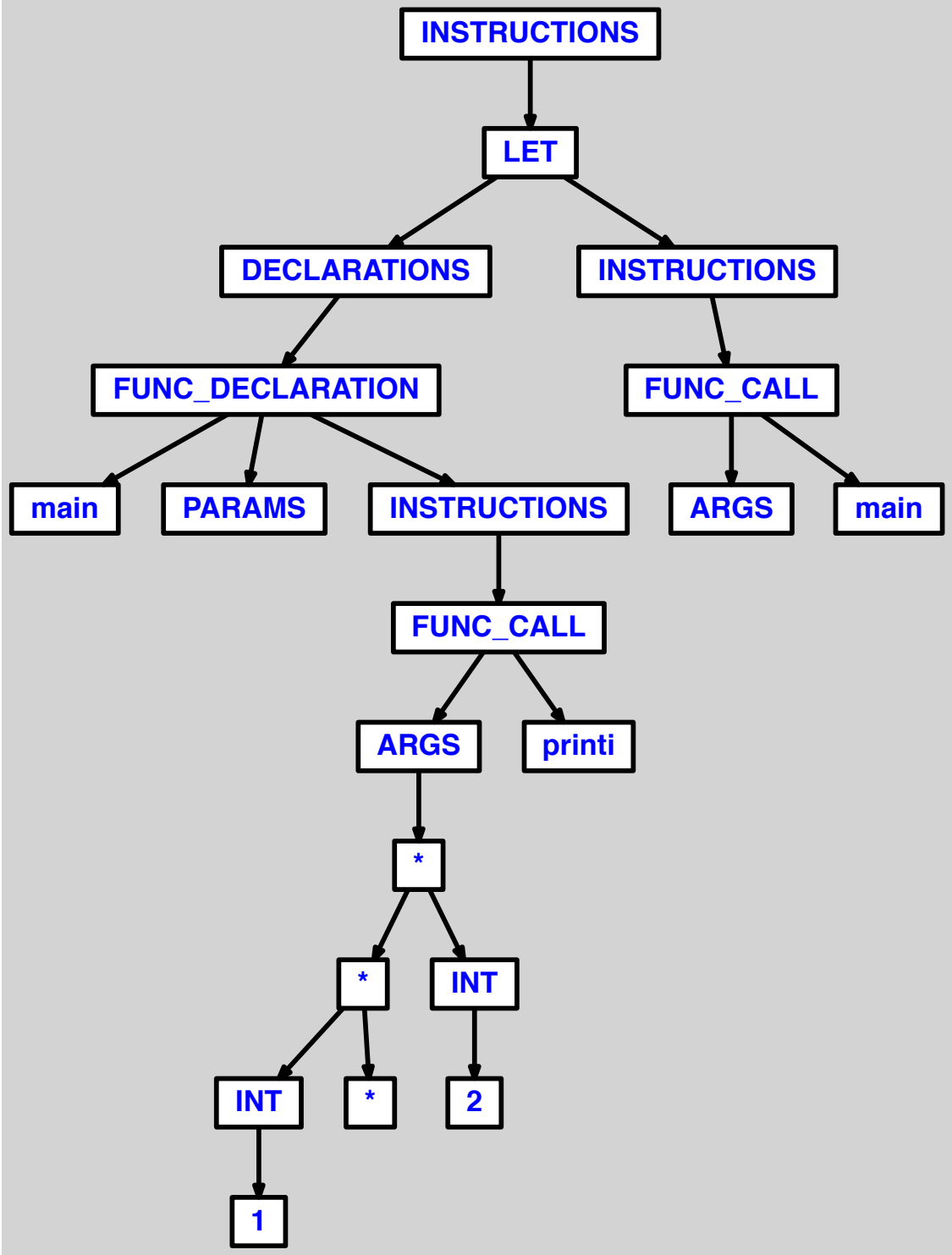
3.1.2 soustraction avec operateur mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
INT
NEG
2
INT
1

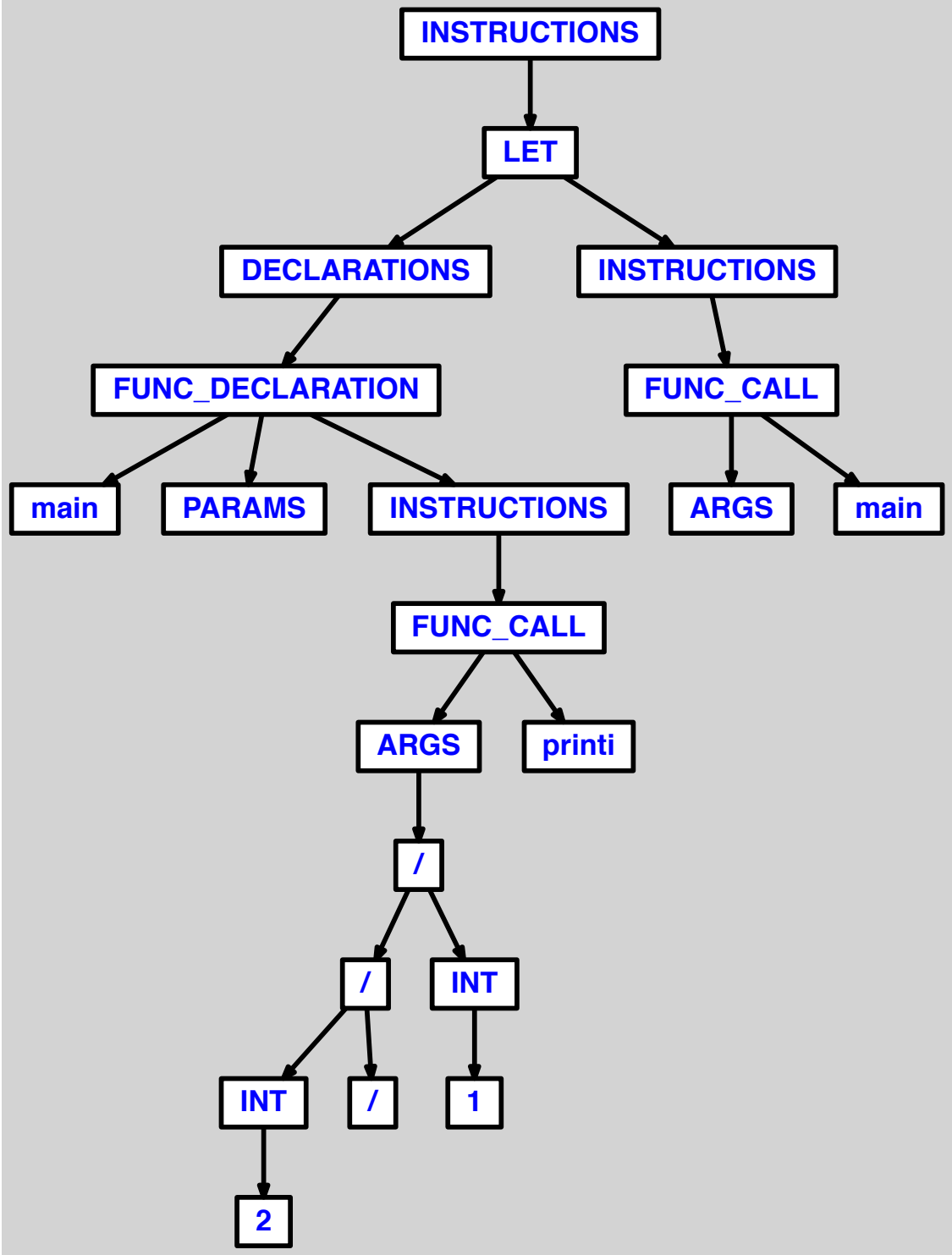
3.1.3 multiplication avec operateur mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
*
INT
INT
*
2
1

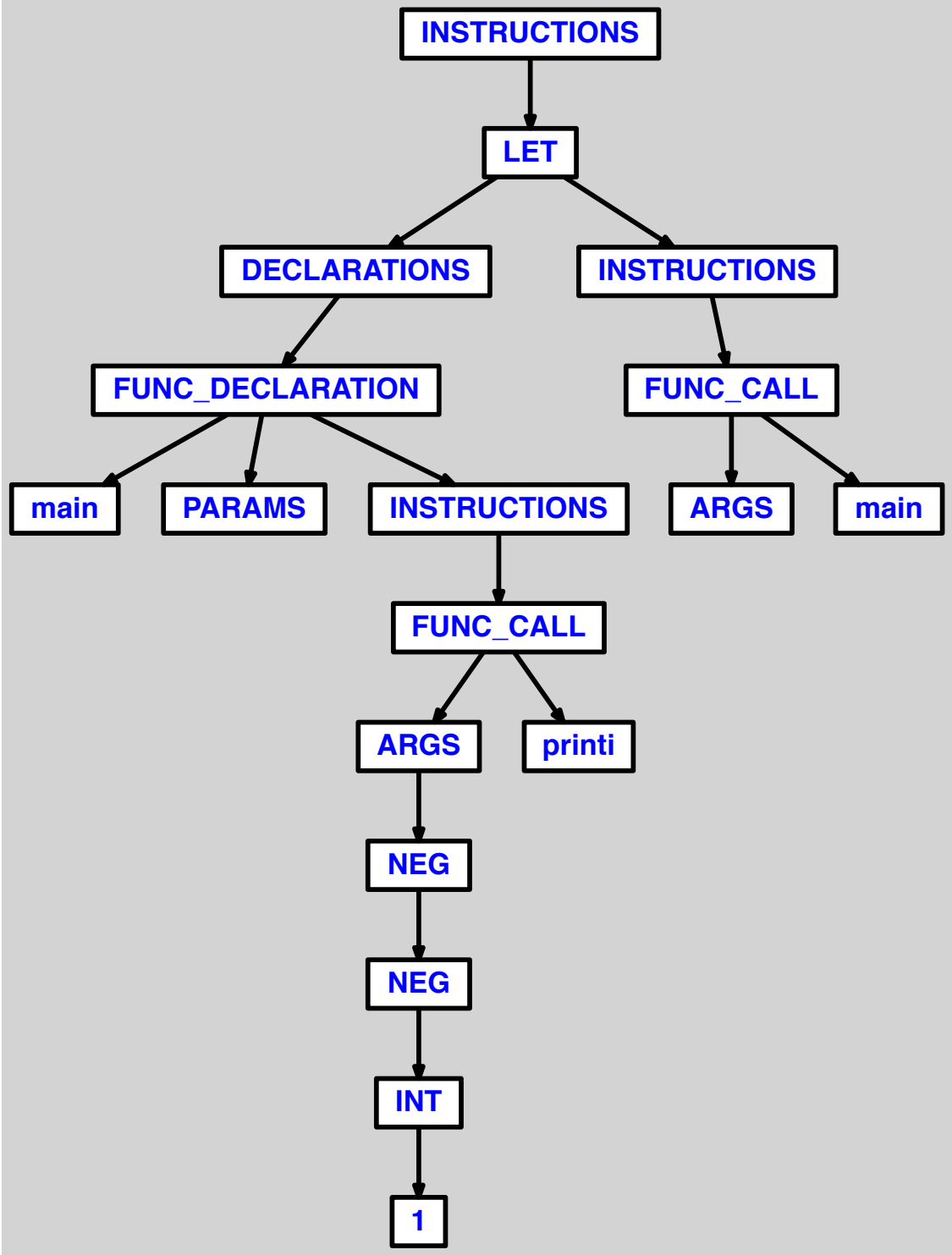
3.1.4 division avec operateur mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
/
INT
INT
/
1
2

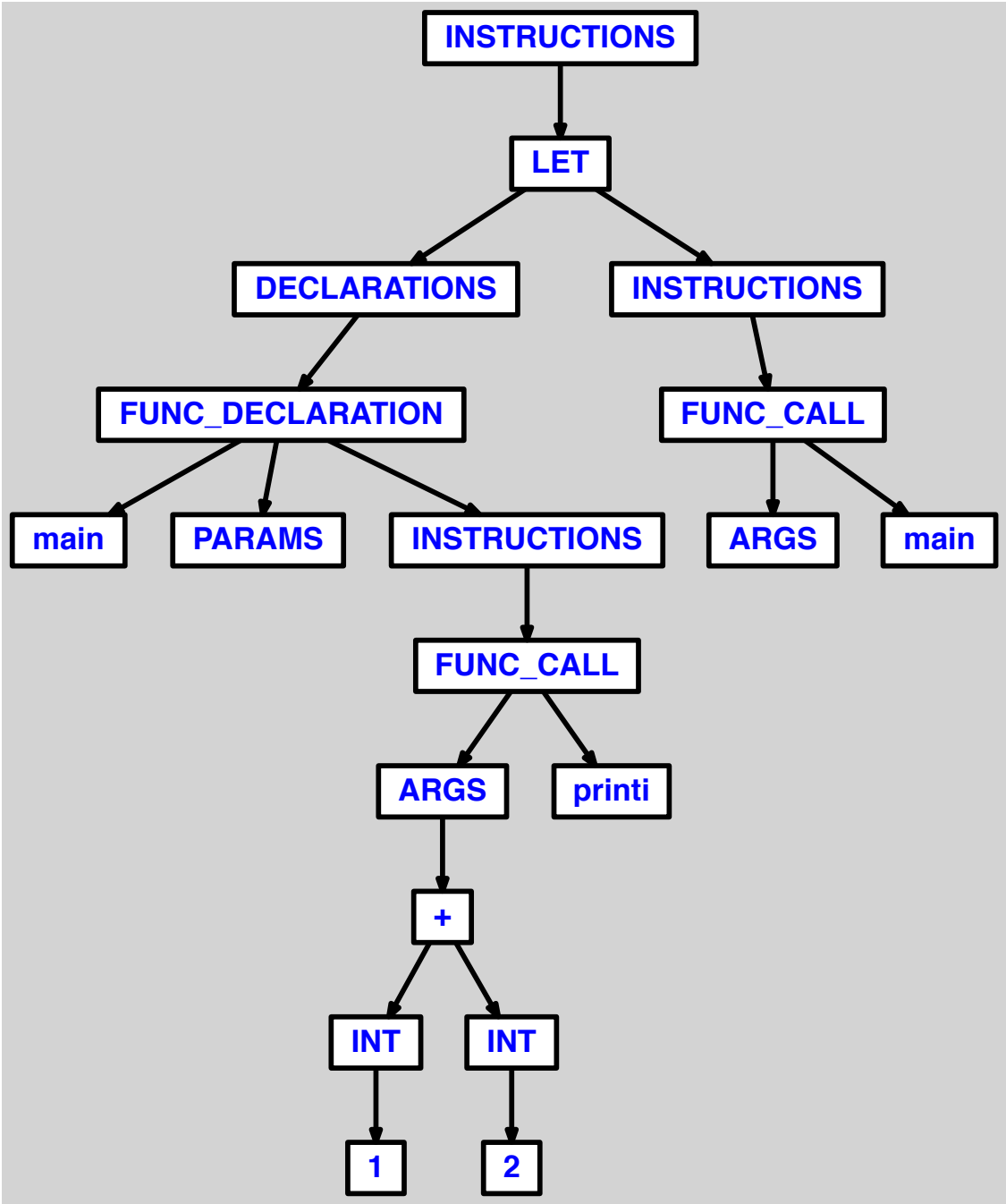
3.1.5 moins unaire avec operateur mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
NEG
NEG
INT
1

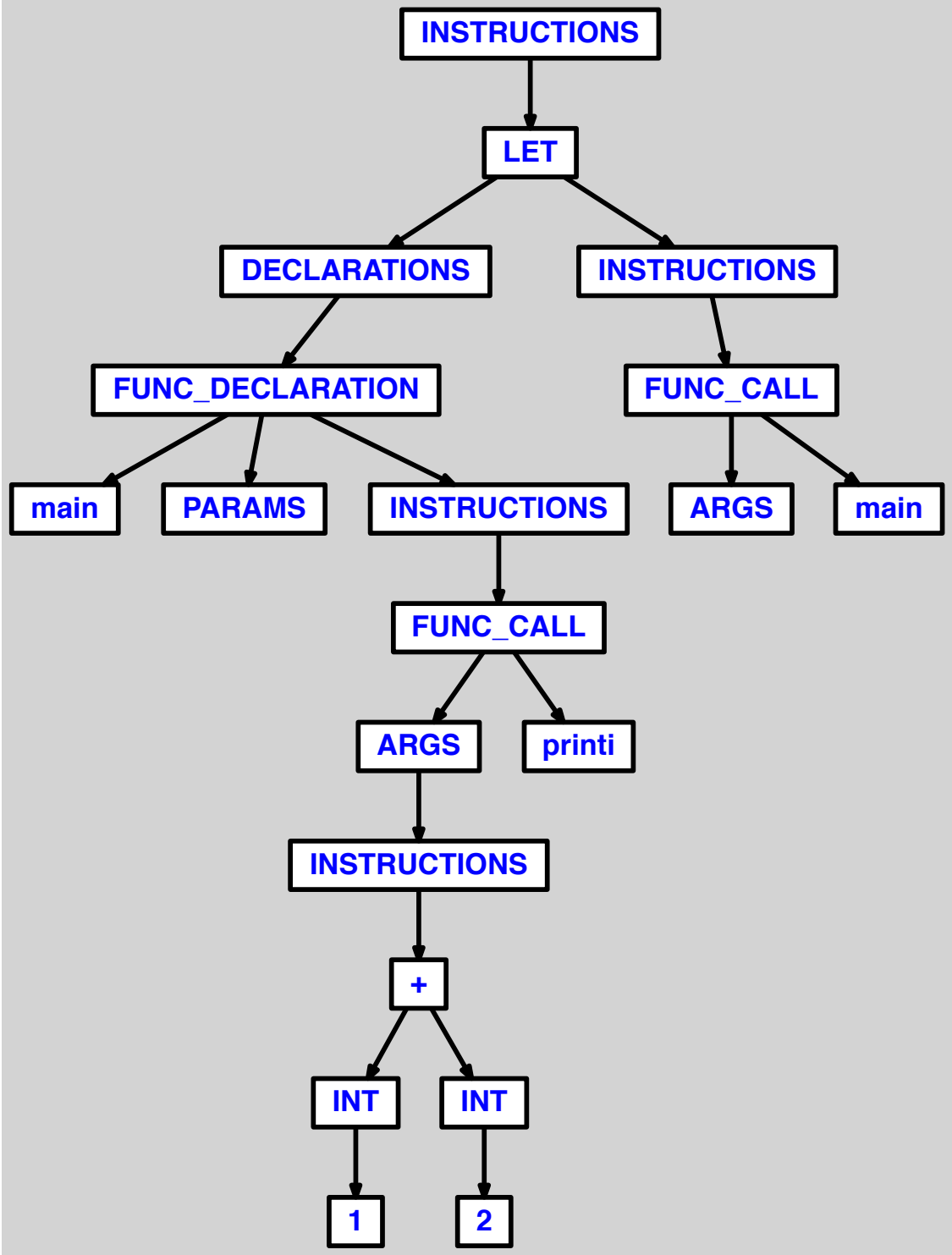
3.1.6 calcul a 2 termes avec oubli de parenthese gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INT
1
2

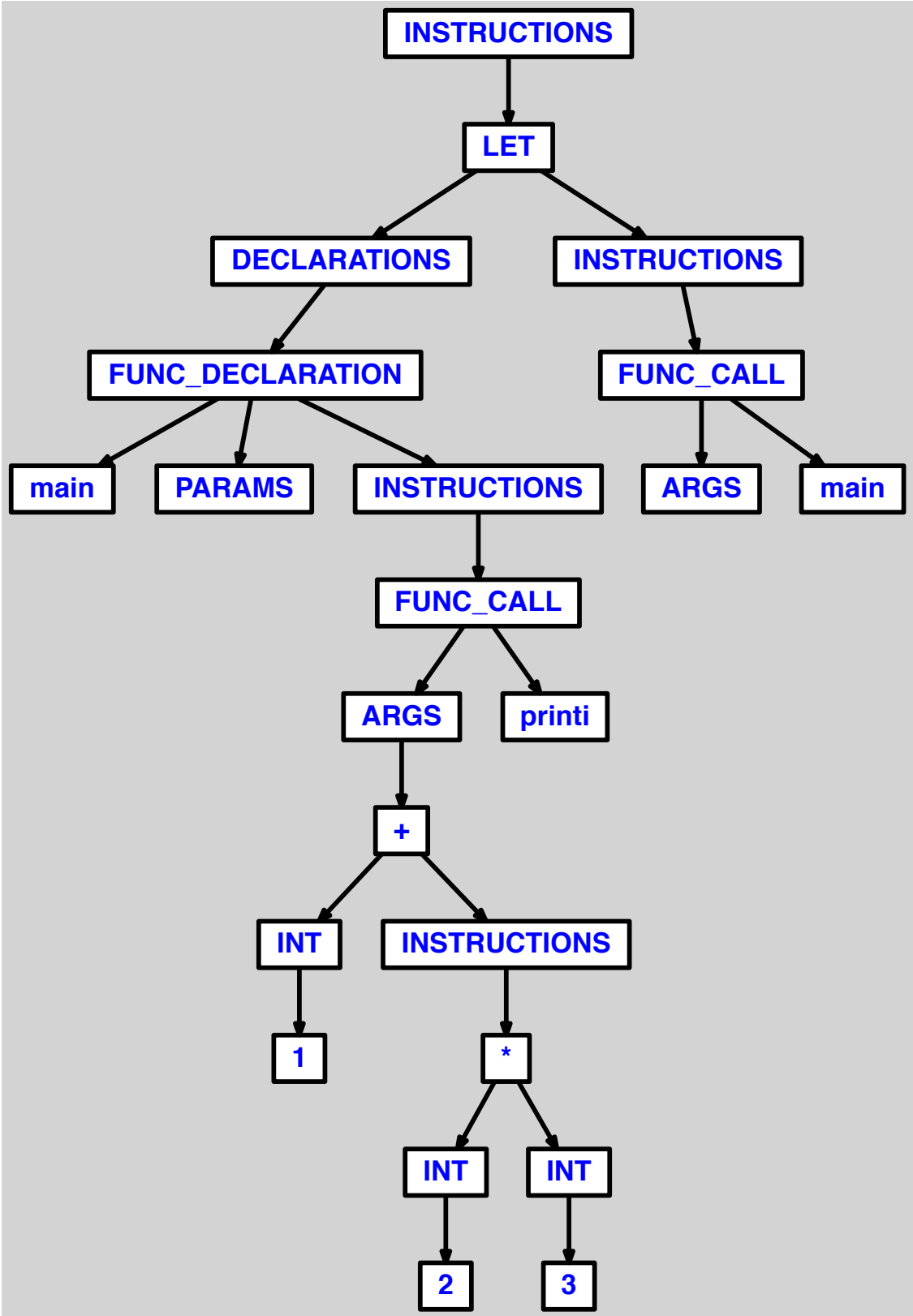
3.1.7 calcul a 2 termes avec oubli de parenthese droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
INSTRUCTIONS
+
INT
INT
1
2

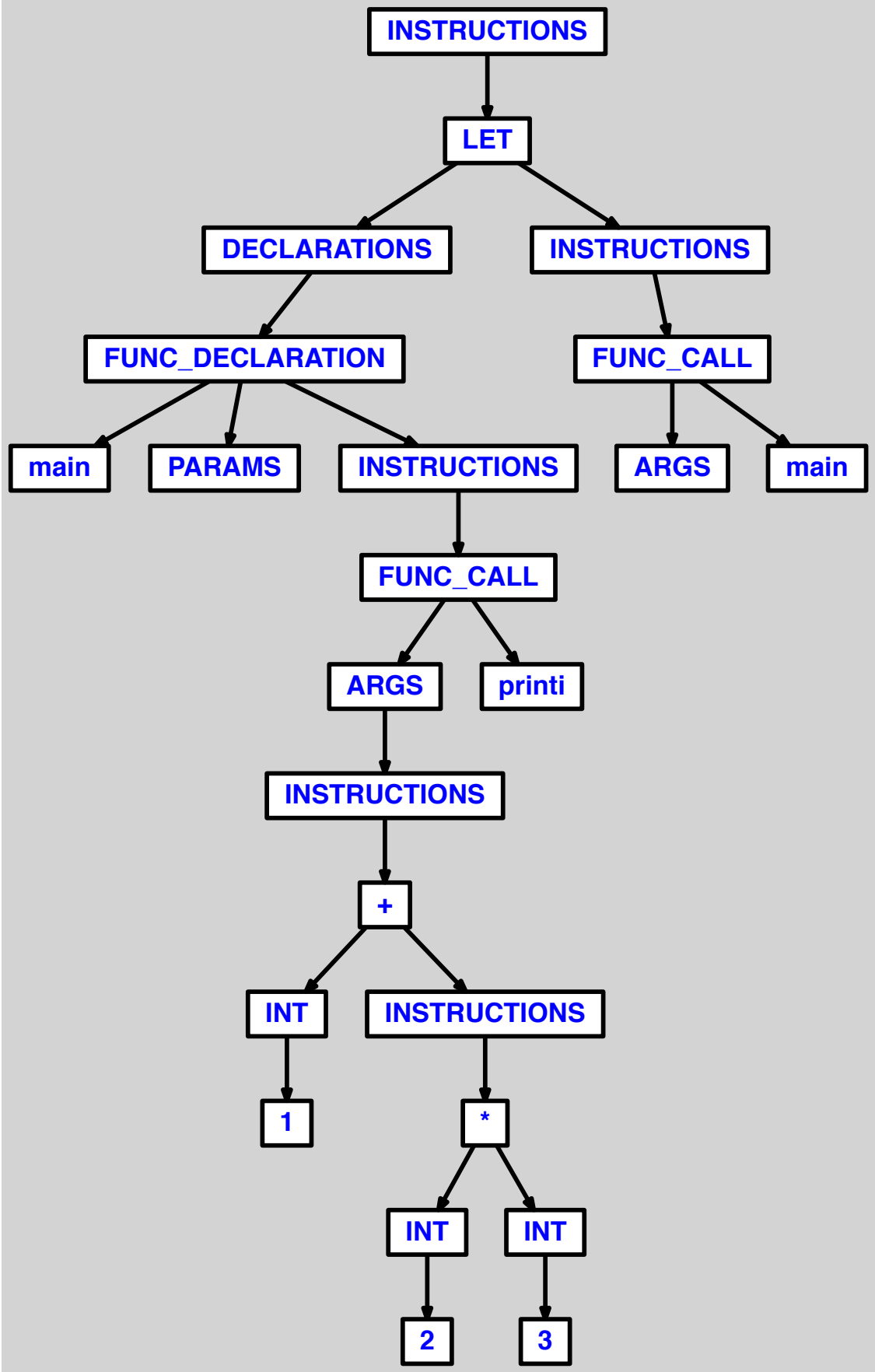
3.1.8 calcul a 3 termes avec oubli de parenthese gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INSTRUCTIONS
1
*
INT
INT
2
3

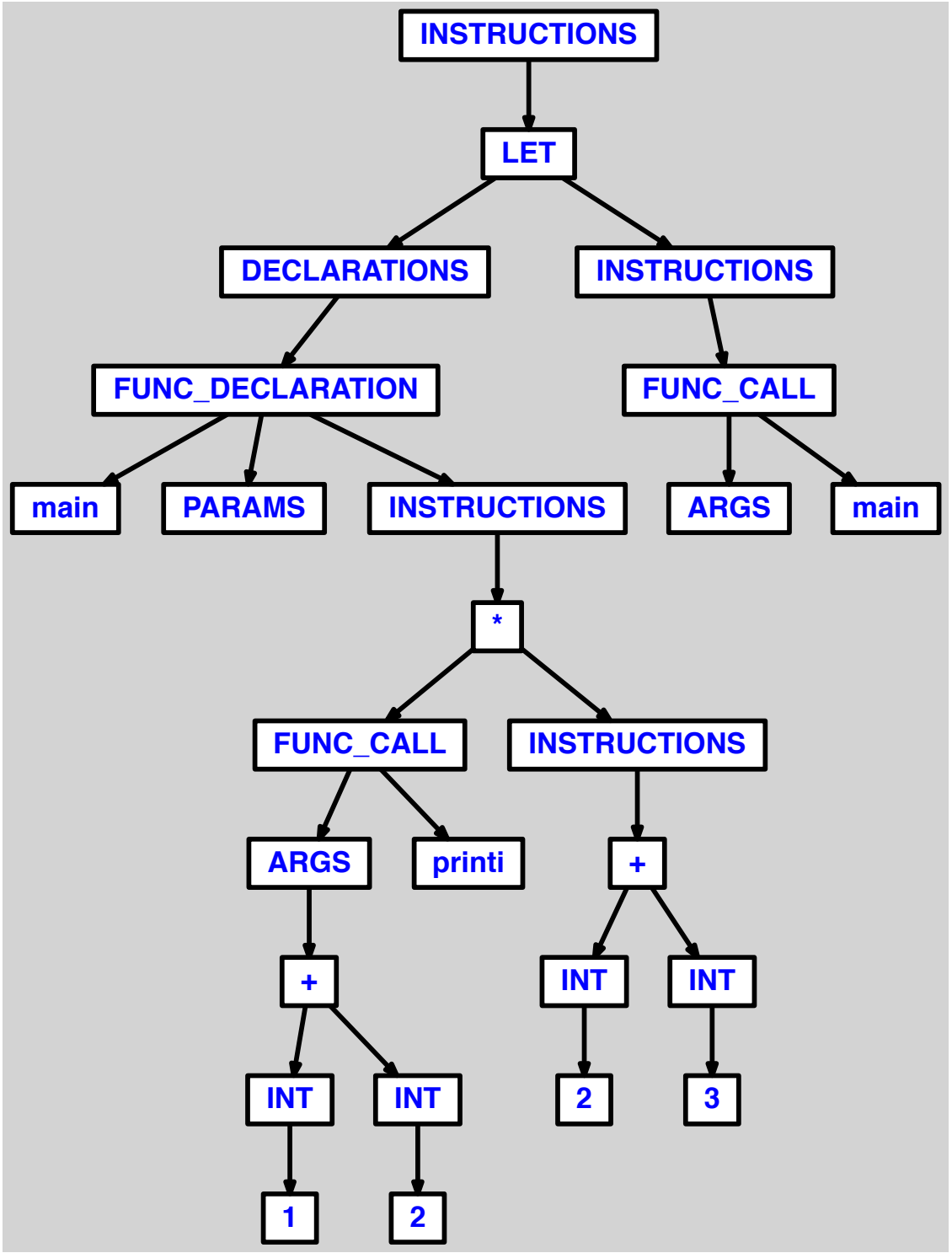
3.1.9 calcul a 3 termes avec oubli de parenthese droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
INSTRUCTIONS
+
INT
INSTRUCTIONS
1
*
INT
INT
2
3

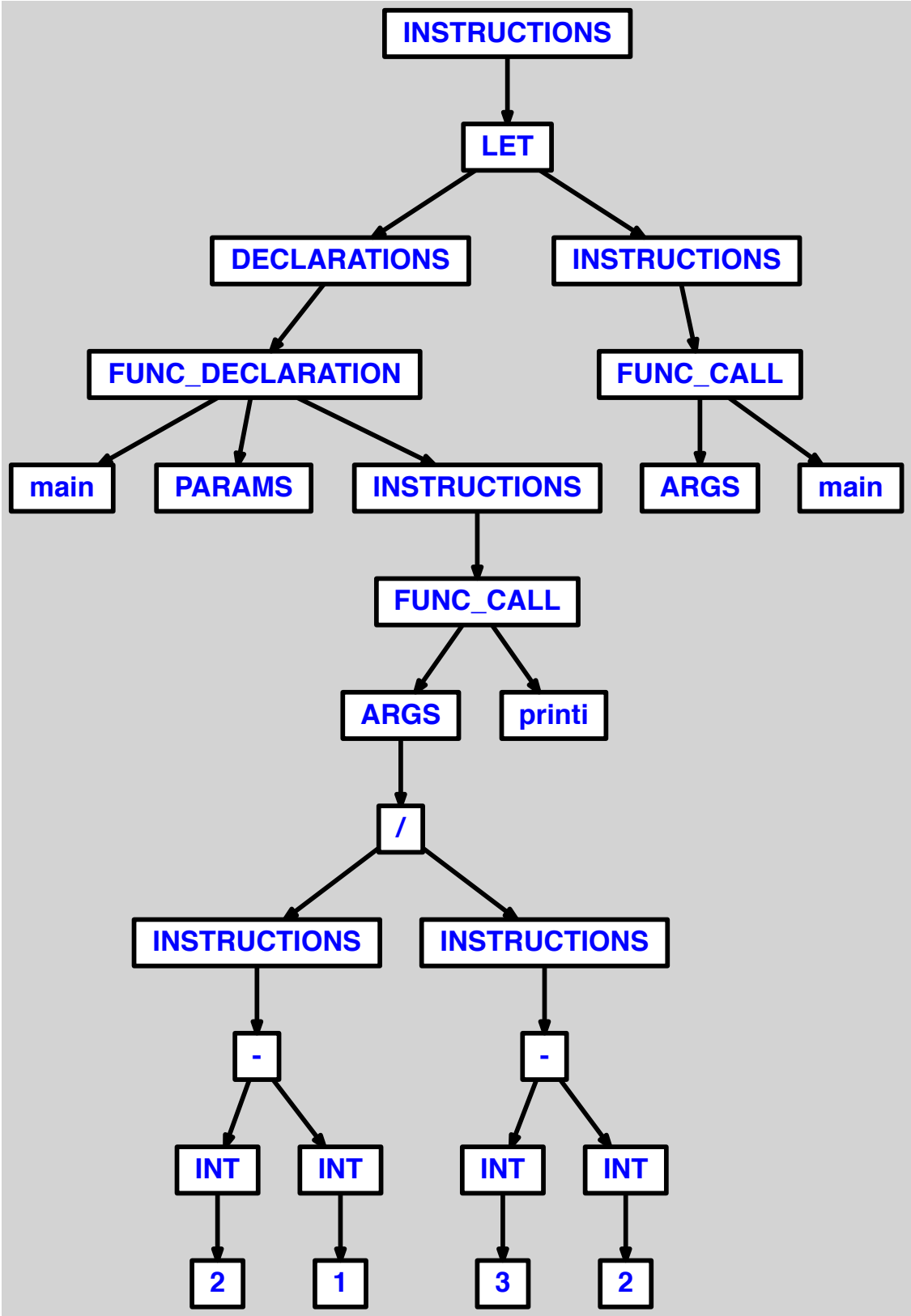
3.1.10 calcul a 4 termes avec oubli de parenthese gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
*
FUNC_CALL
INSTRUCTIONS
ARGS
printi
+
+
INT
INT
INT
INT
2
3
1
2

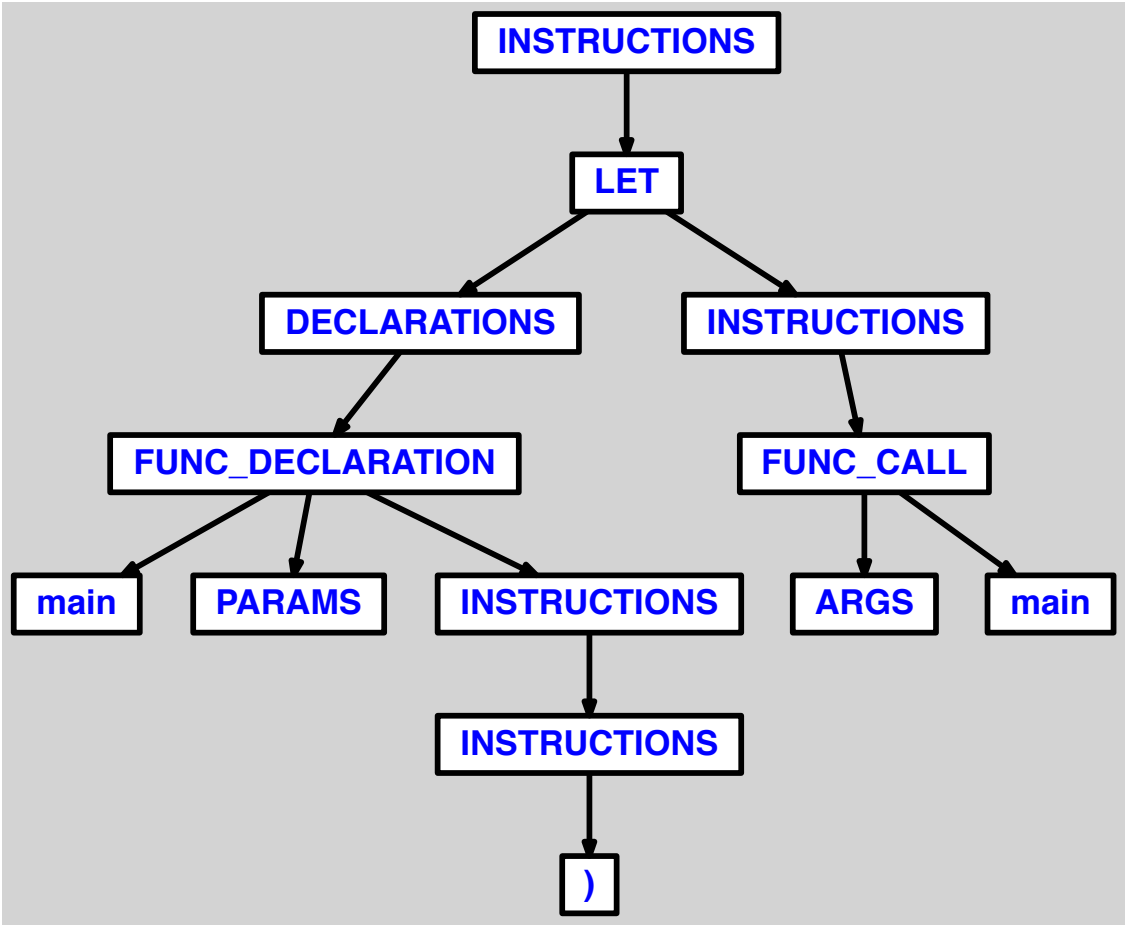
3.1.11 calcul a 4 termes avec oubli de parenthese droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
INSTRUCTIONS
-
-
INT
INT
INT
INT
2
1
3
2

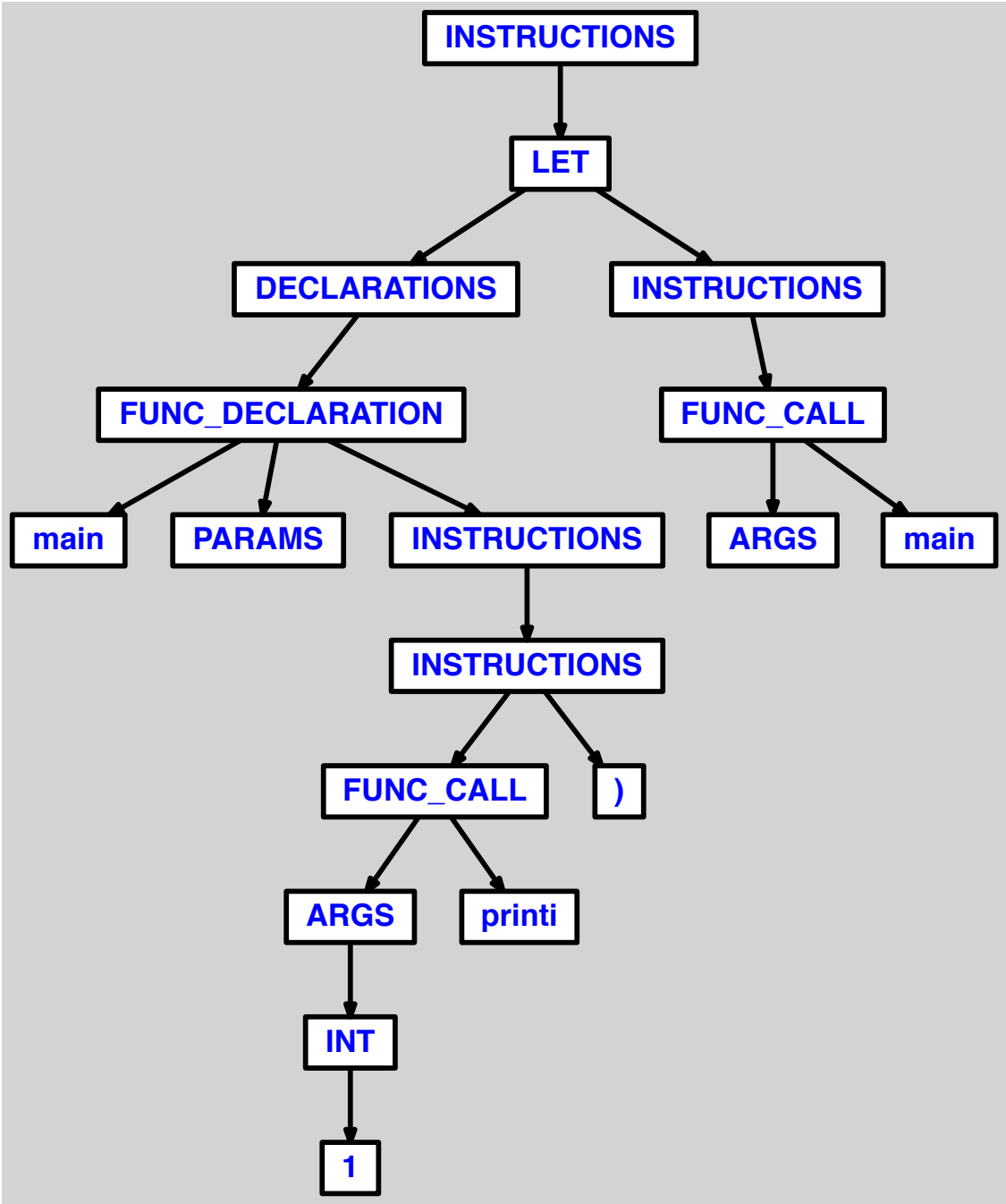
3.1.12 parenthesage sans contenu

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
)

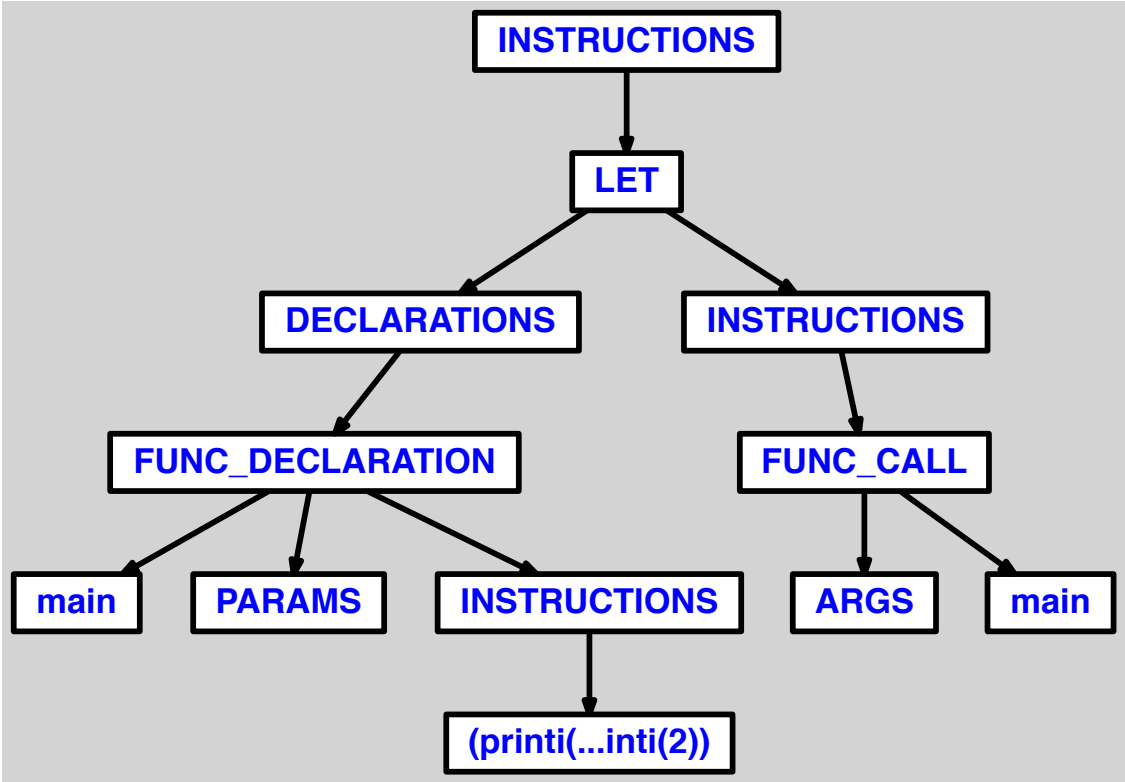
3.1.13 expression parenthesee avec <;> en trop

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
)
ARGS
printi
INT
1

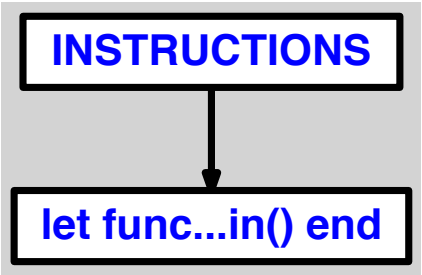
3.1.14 expression parenthesee avec oubli de <;>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
(printi(...inti(2))

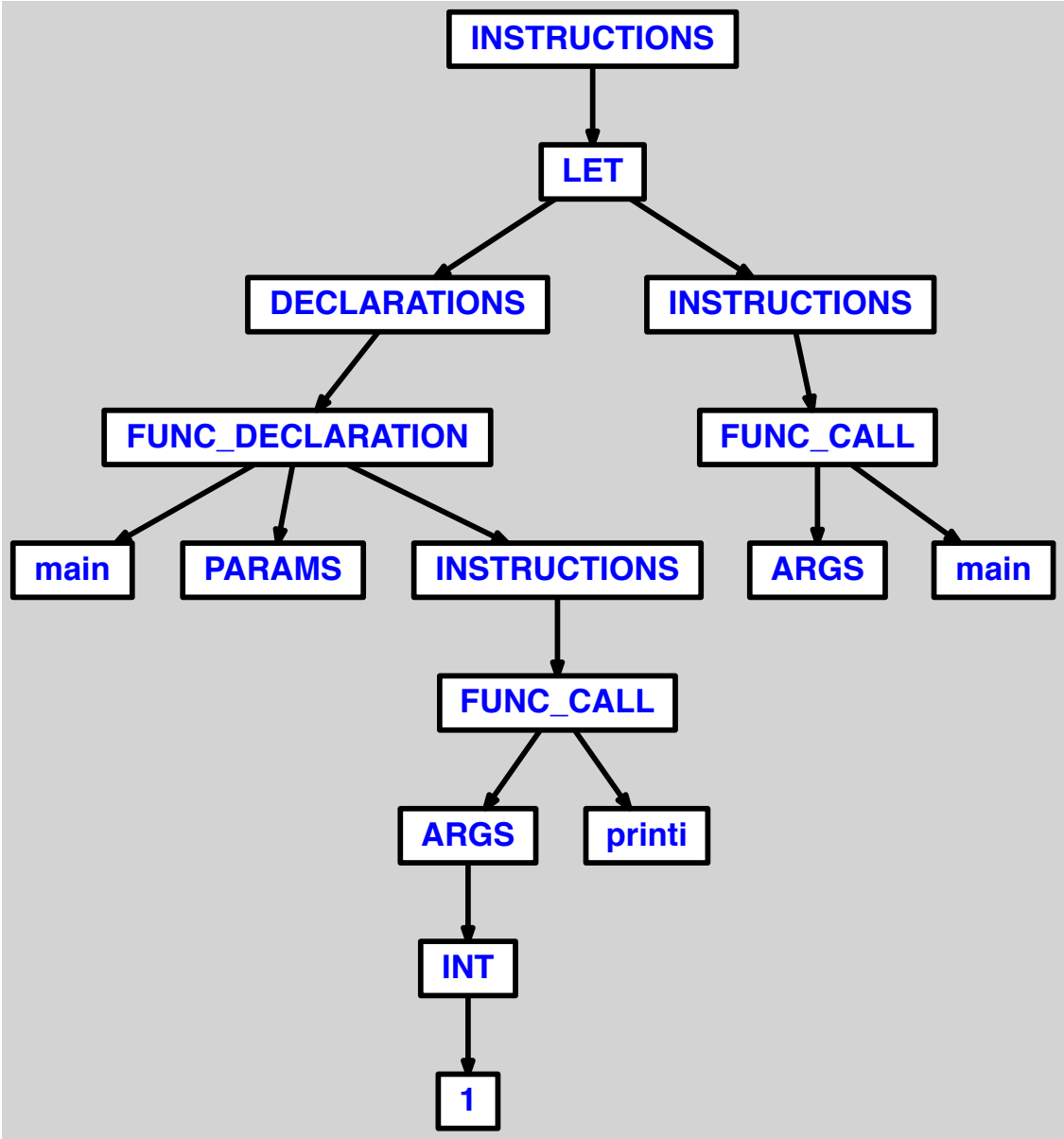
3.1.15 expression non parenthesee avec oubli de <;>

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


INSTRUCTIONS
let func...in() end

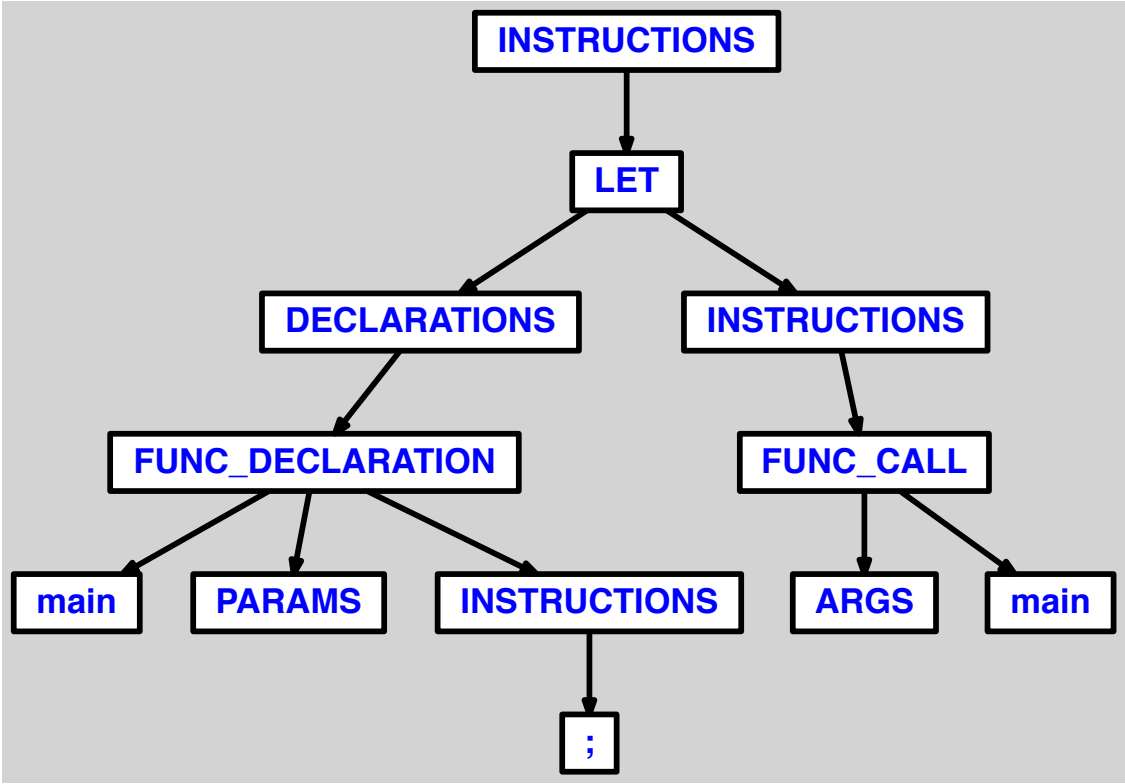
3.1.16 expression non parenthesee avec <;> en trop

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
INT
1

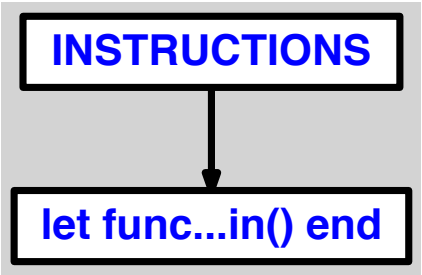
3.1.17 <;> sans instruction avant

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
;

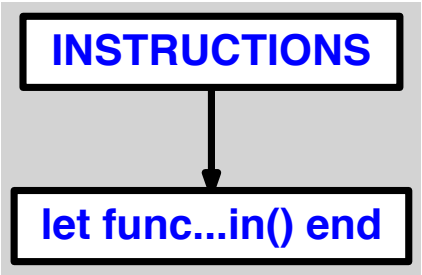
3.1.18 2 instructions sans parentheses

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


INSTRUCTIONS
let func...in() end

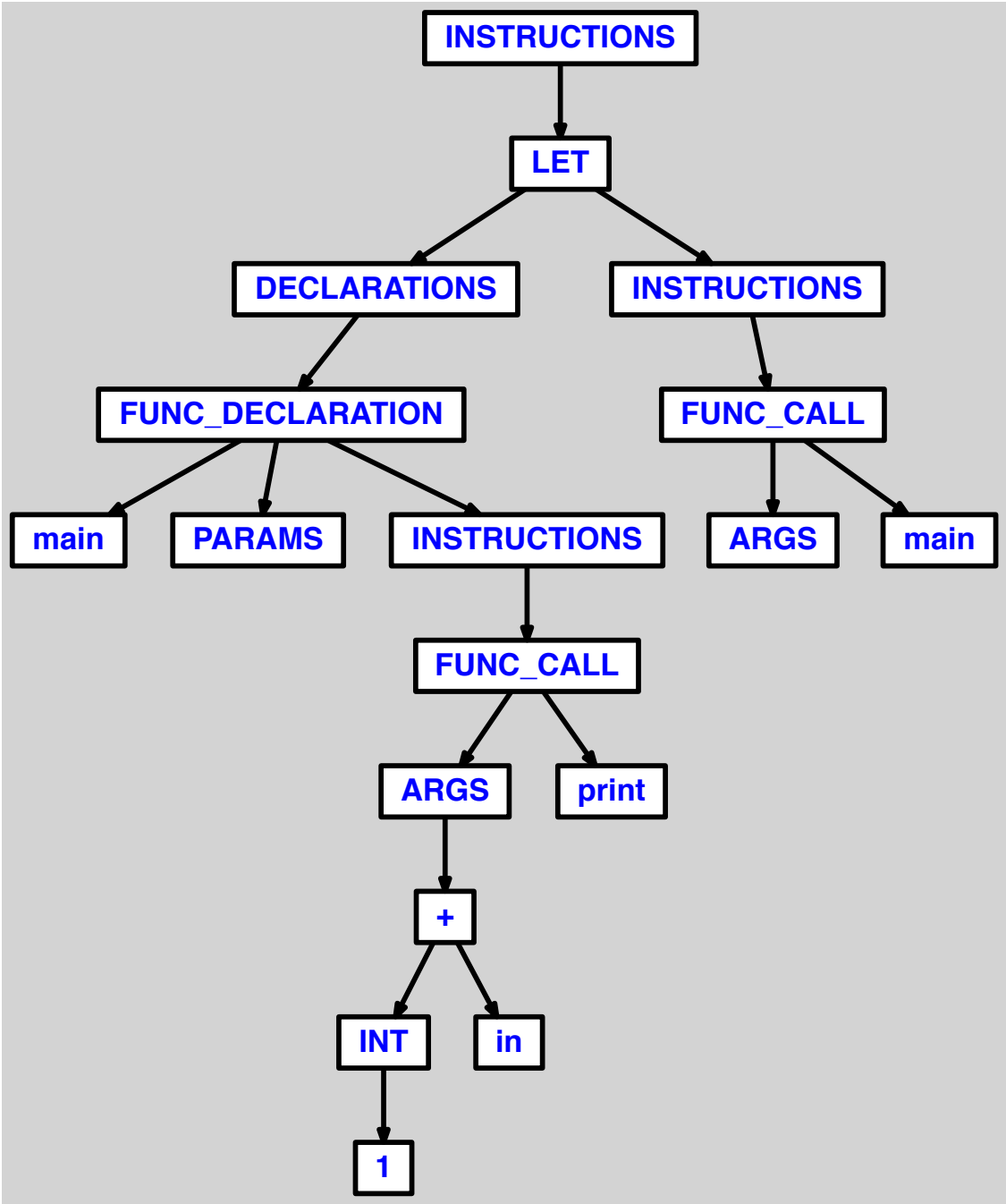
3.1.19 3 instructions sans parentheses

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


INSTRUCTIONS
let func...in() end

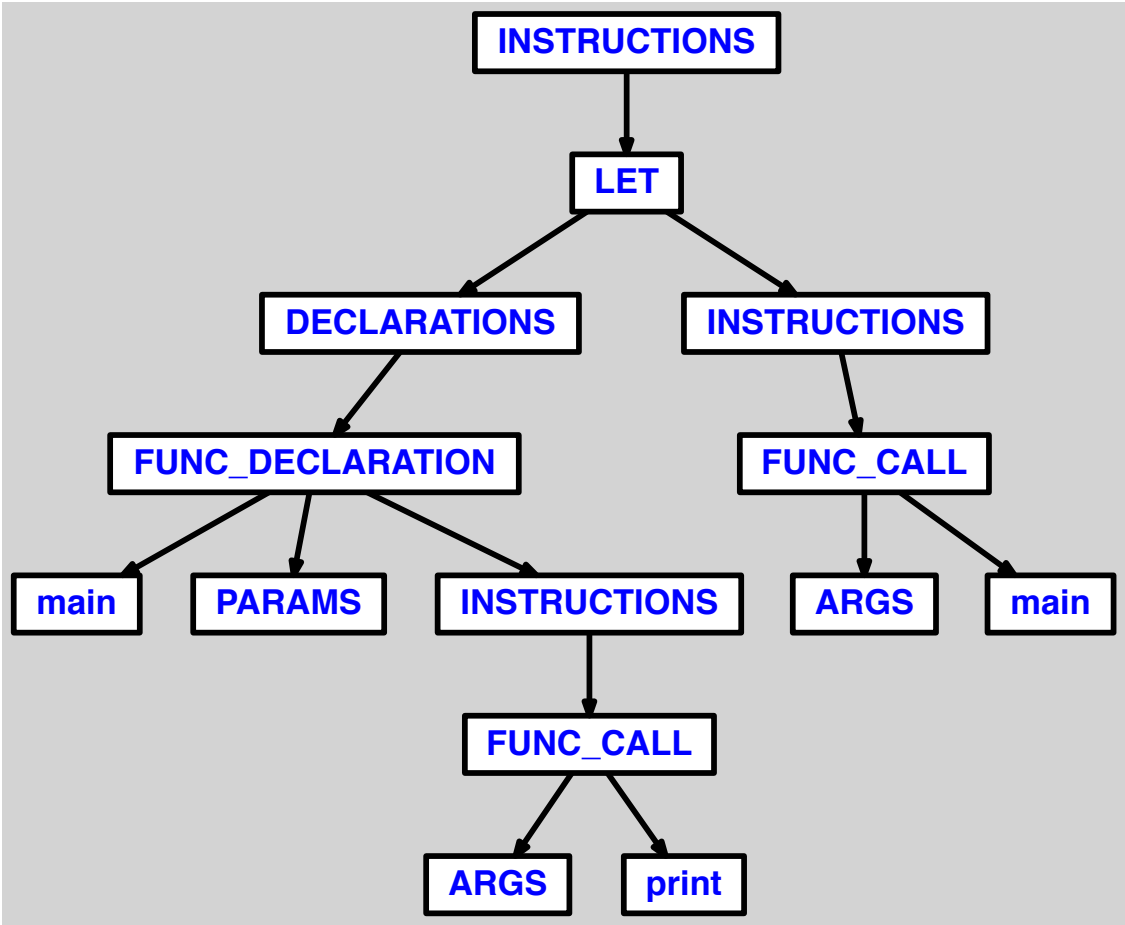
3.1.20 concatenation d'entier et de chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
+
INT
in
1

3.1.21 concatenation de 2 chaines

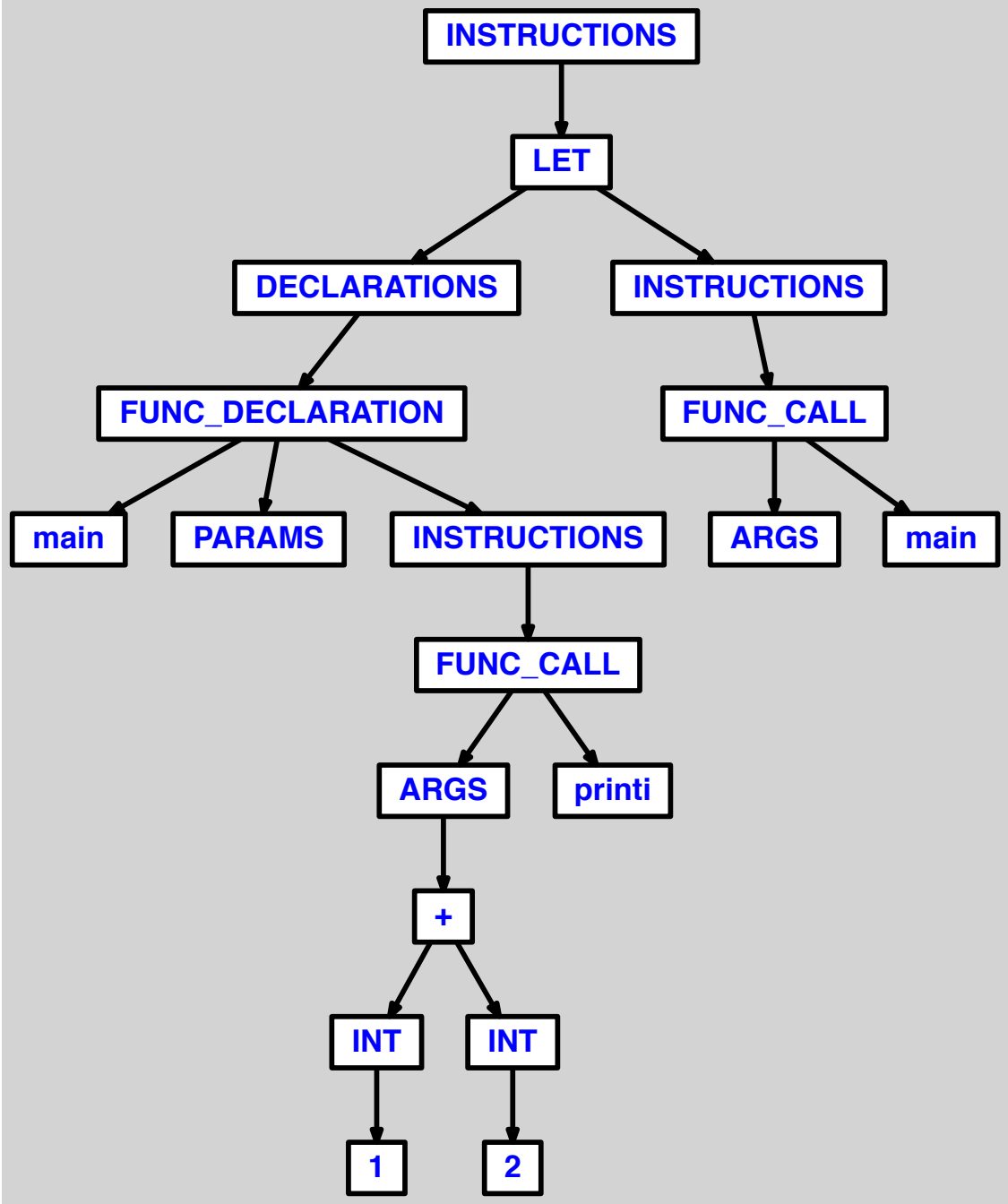
```
let
  function main() = print("1"+"2")
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print

3.2 OK

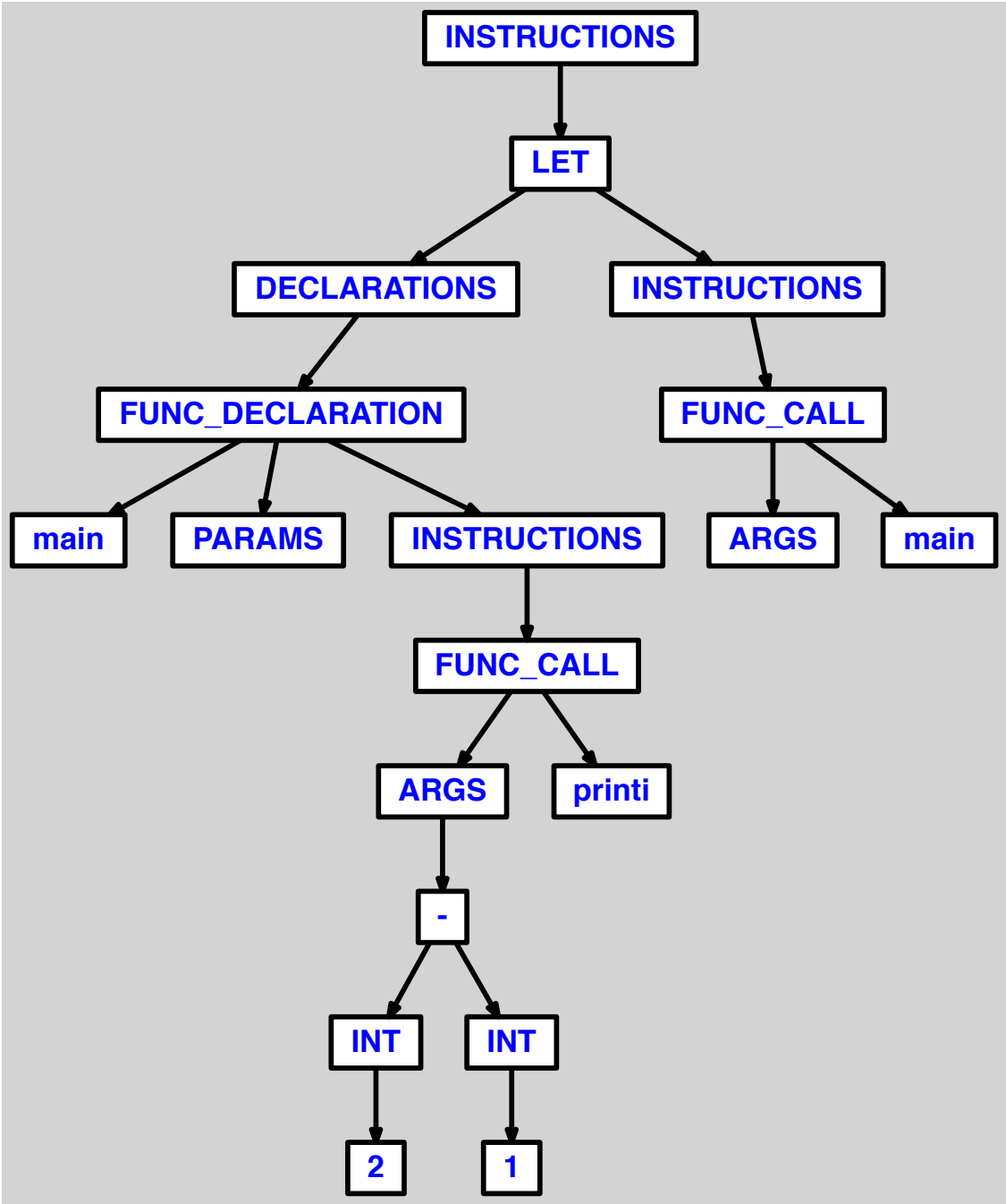
3.2.1 addition simple, a 2 termes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INT
1
2

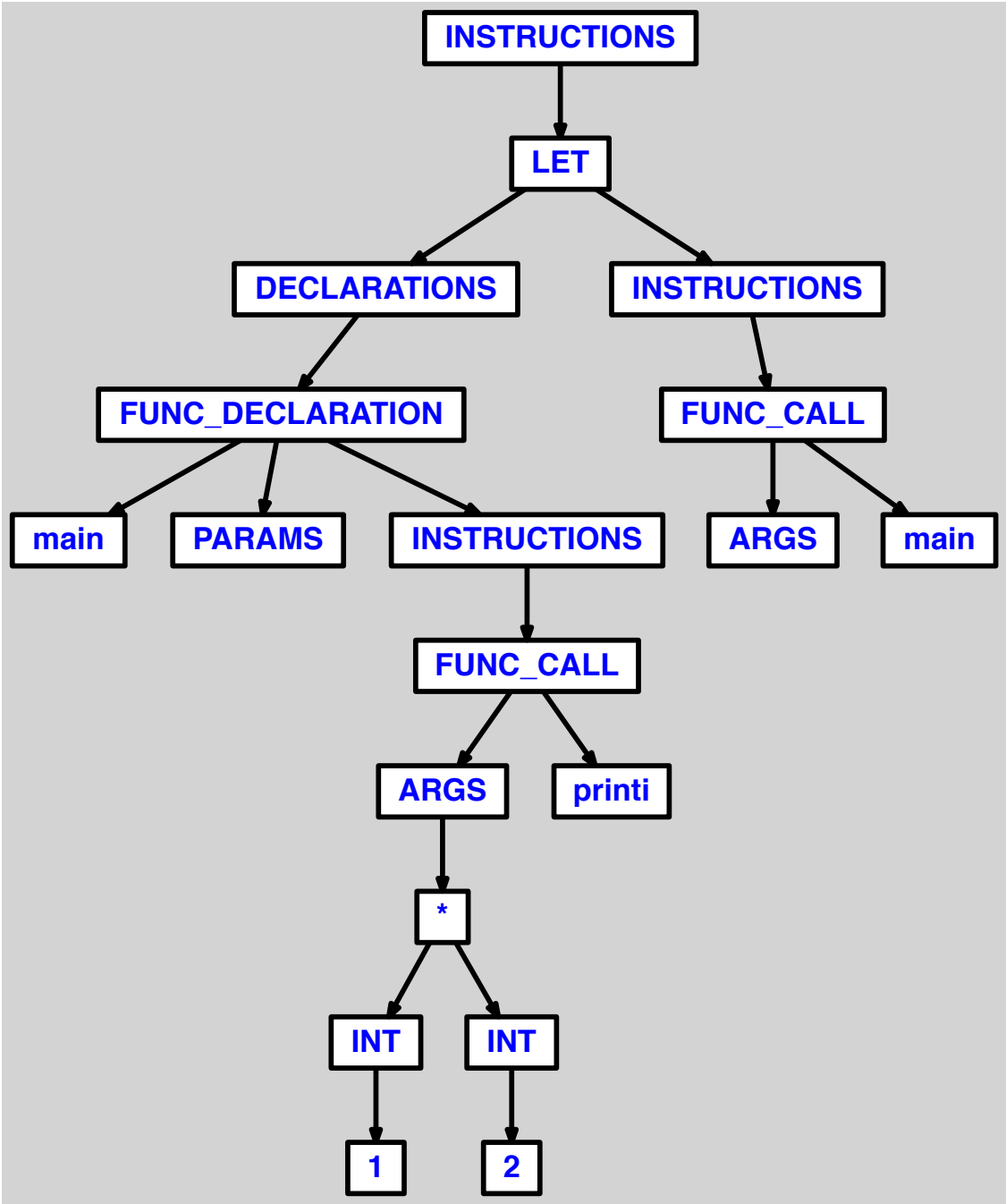
3.2.2 soustraction simple, a 2 termes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
INT
INT
2
1

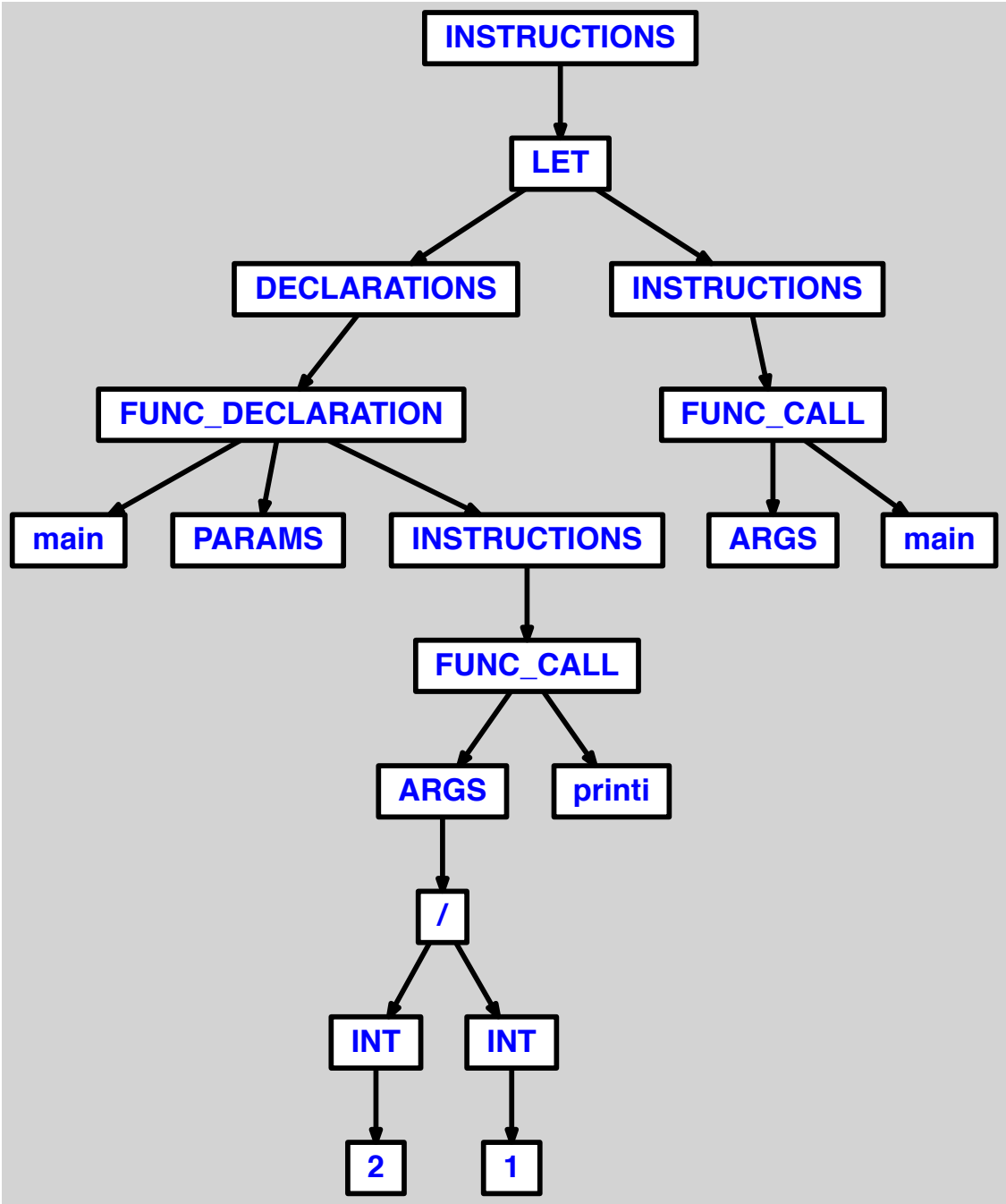
3.2.3 multiplication simple, a 2 termes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INT
INT
1
2

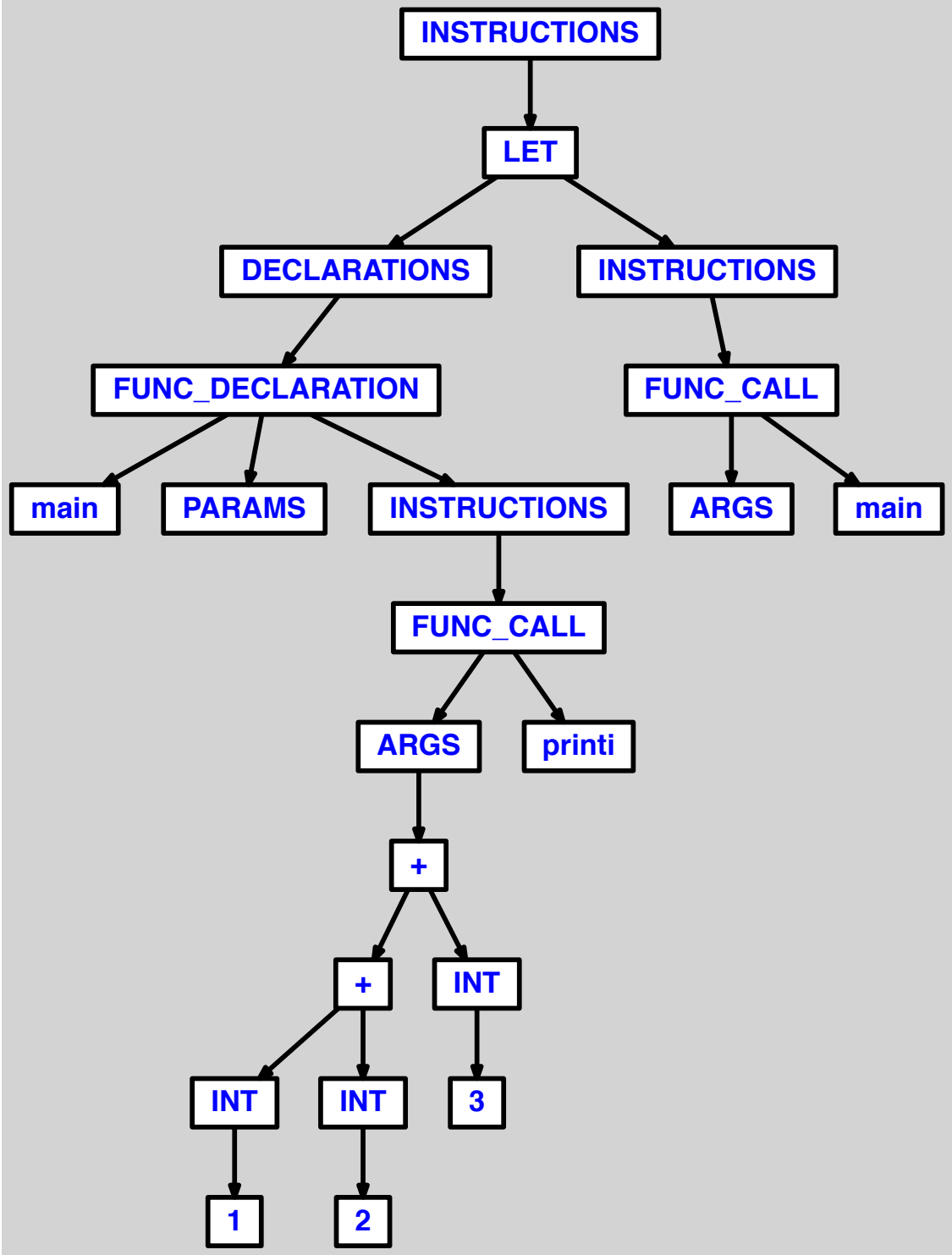
3.2.4 division simple, a 2 termes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
INT
INT
2
1

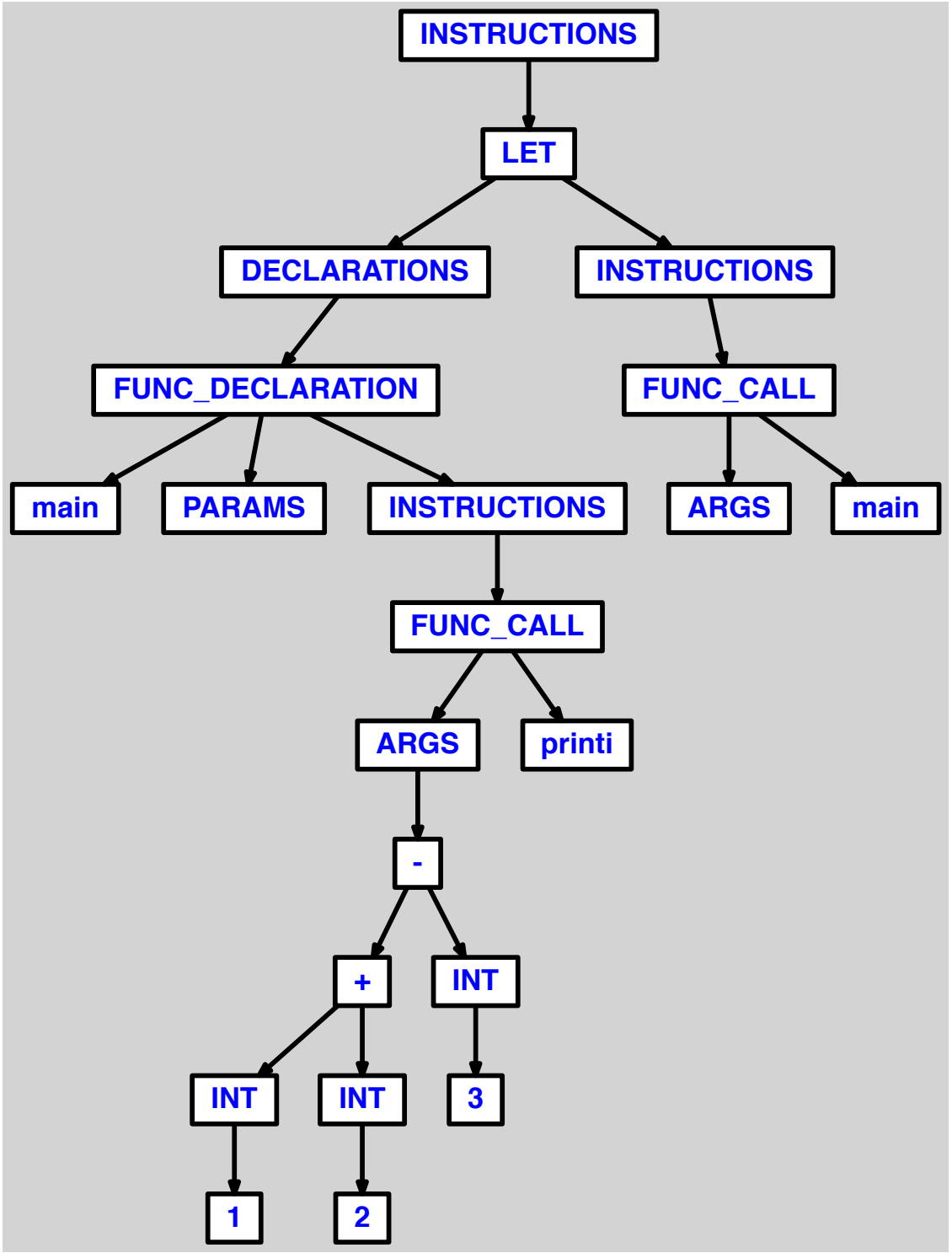
3.2.5 addition a 3 termes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
+
INT
INT
INT
3
1
2

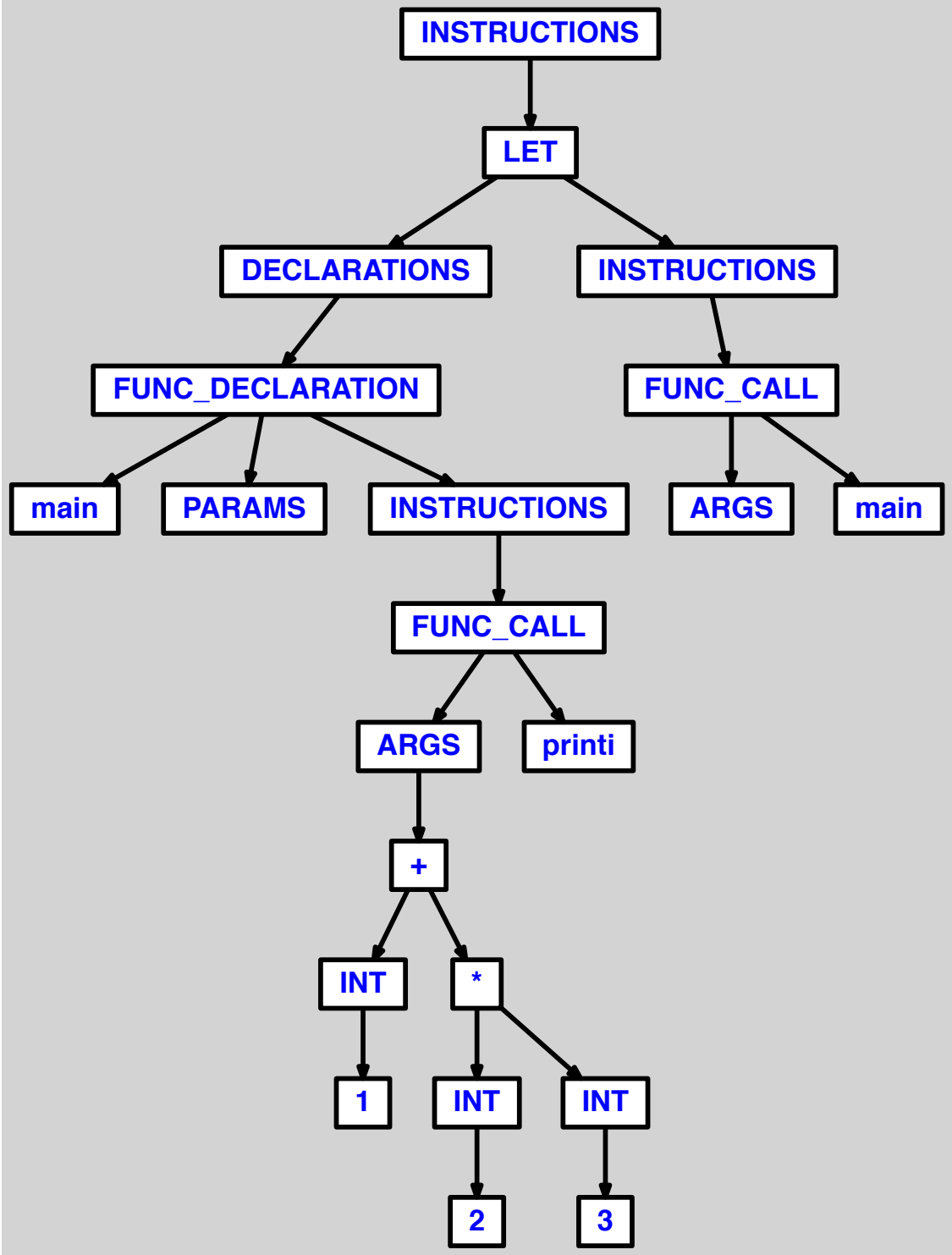
3.2.6 addition suivie de soustraction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
+
INT
INT
INT
3
1
2

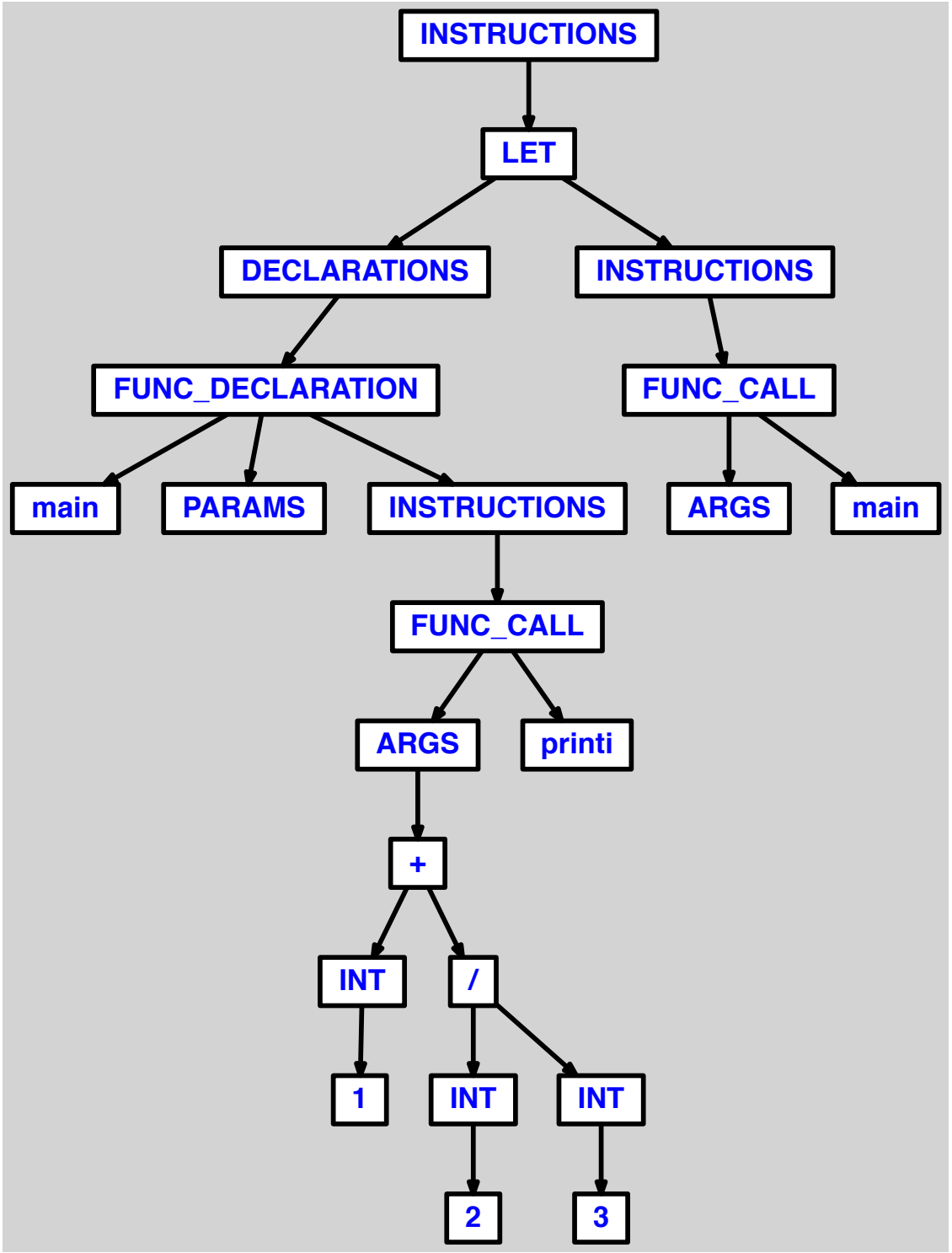
3.2.7 addition suivie de multiplication

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
*
1
INT
INT
2
3

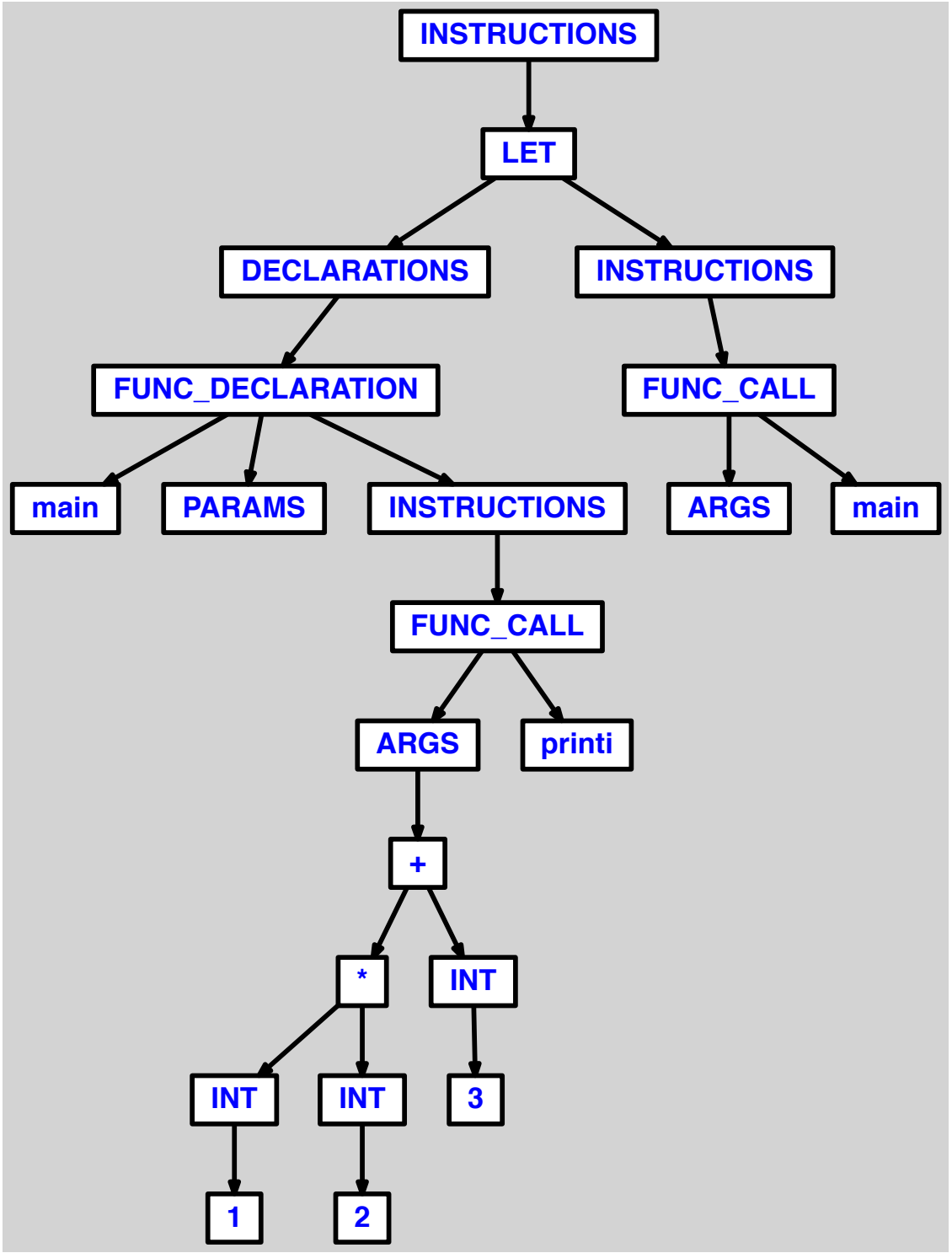
3.2.8 addition suivie de division

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
/
1
INT
INT
2
3

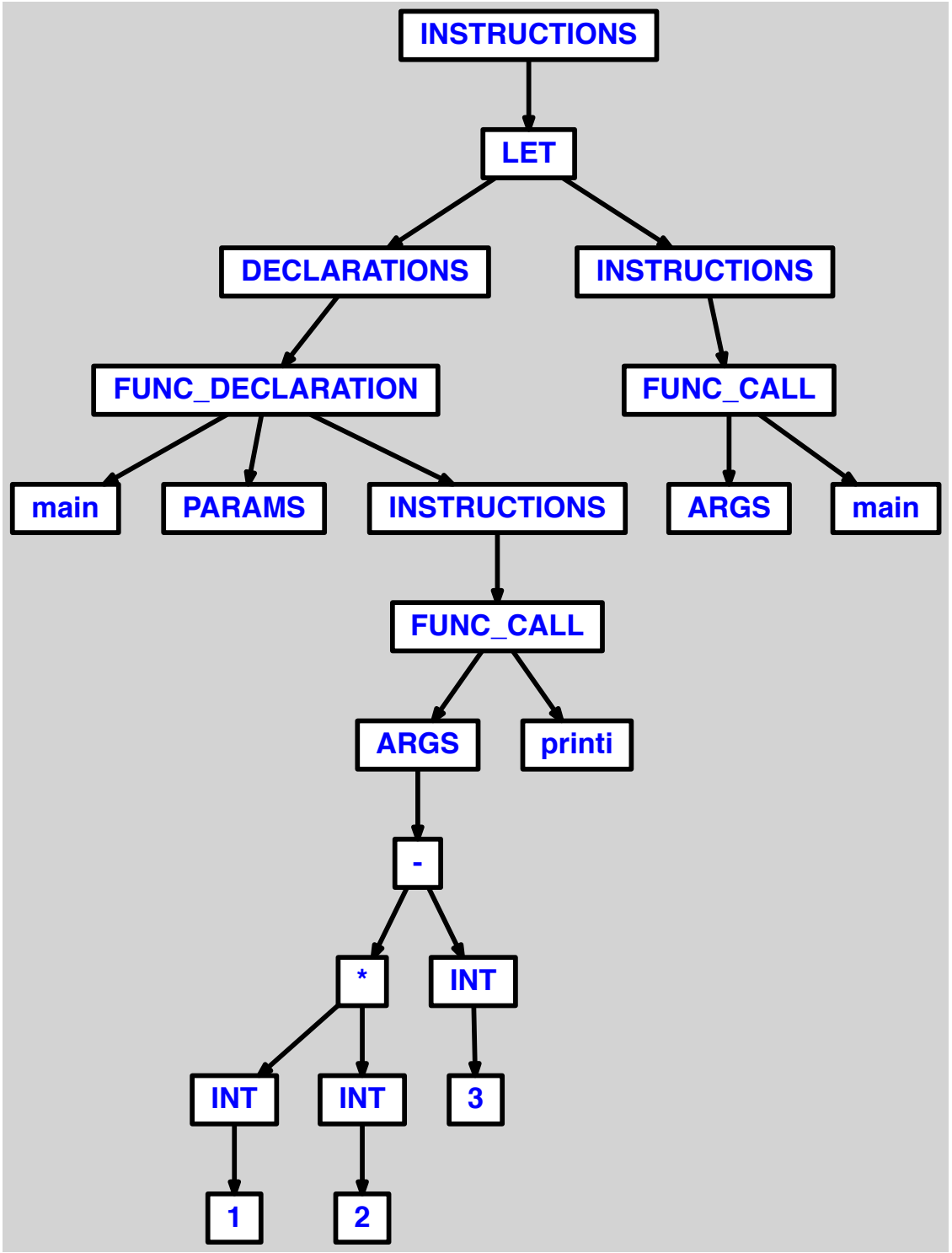
3.2.9 multiplication suivie d'addition

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
*
INT
INT
INT
3
1
2

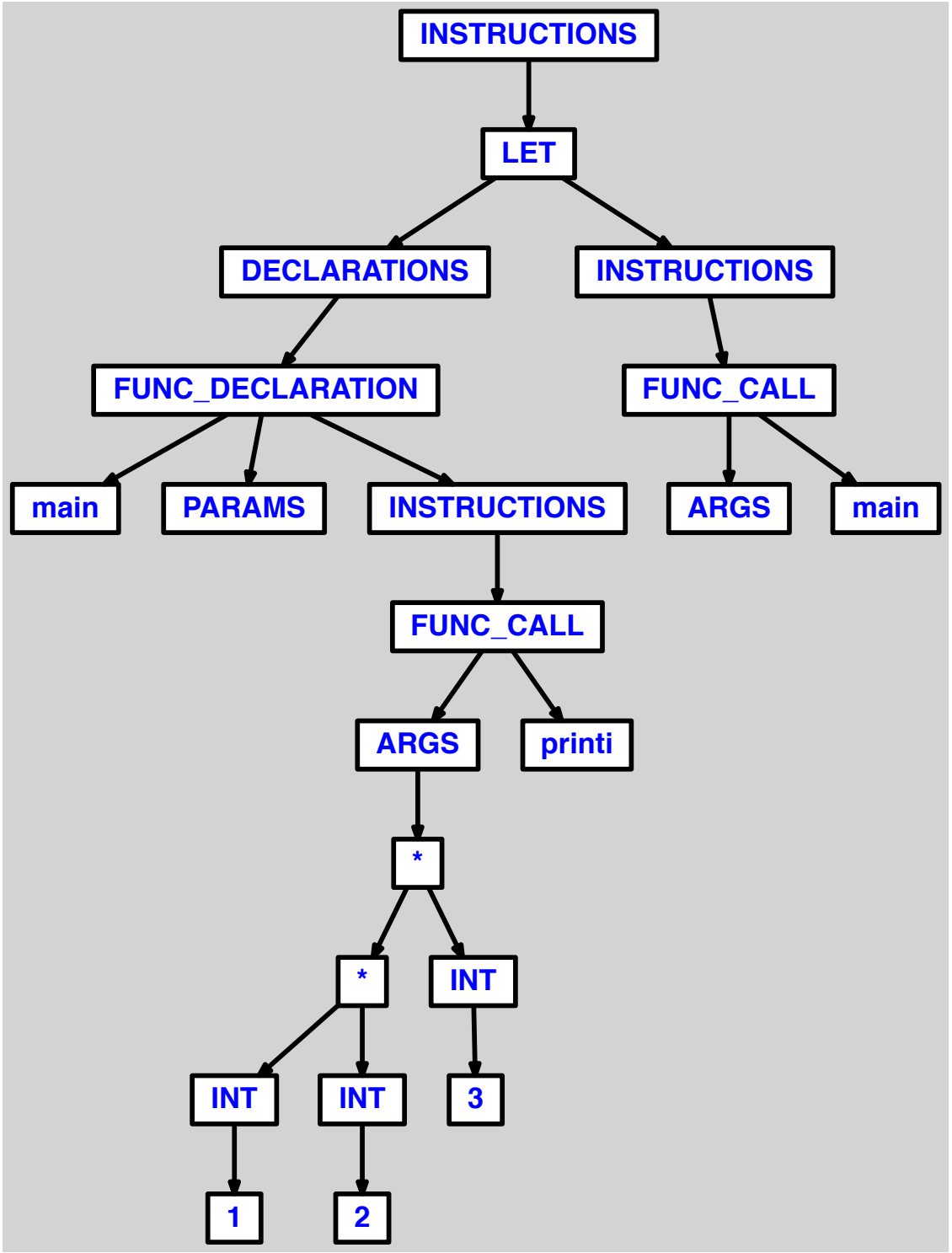
3.2.10 multiplication suivie de soustraction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
*
INT
INT
INT
3
1
2

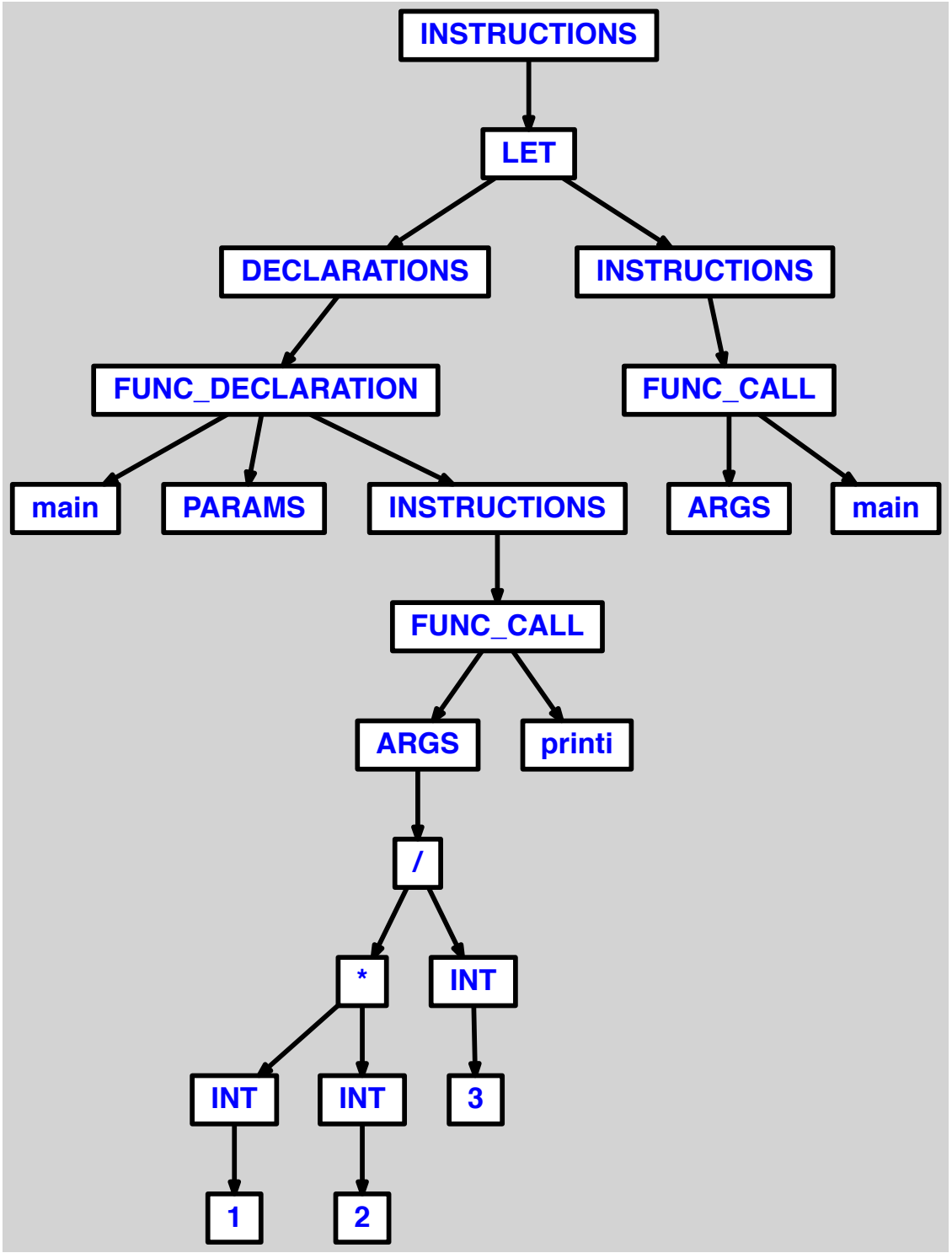
3.2.11 multiplication a 3 termes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
*
INT
INT
INT
3
1
2

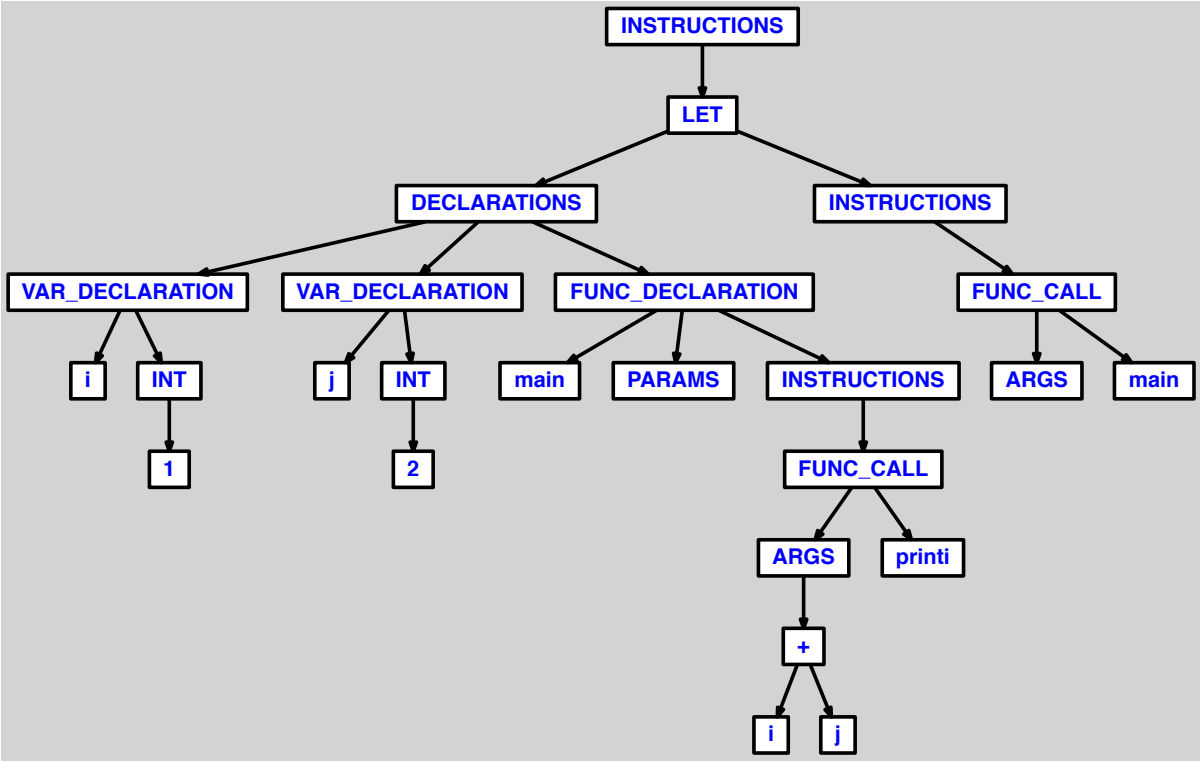
3.2.12 multiplication suivie de division

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
*
INT
INT
INT
3
1
2

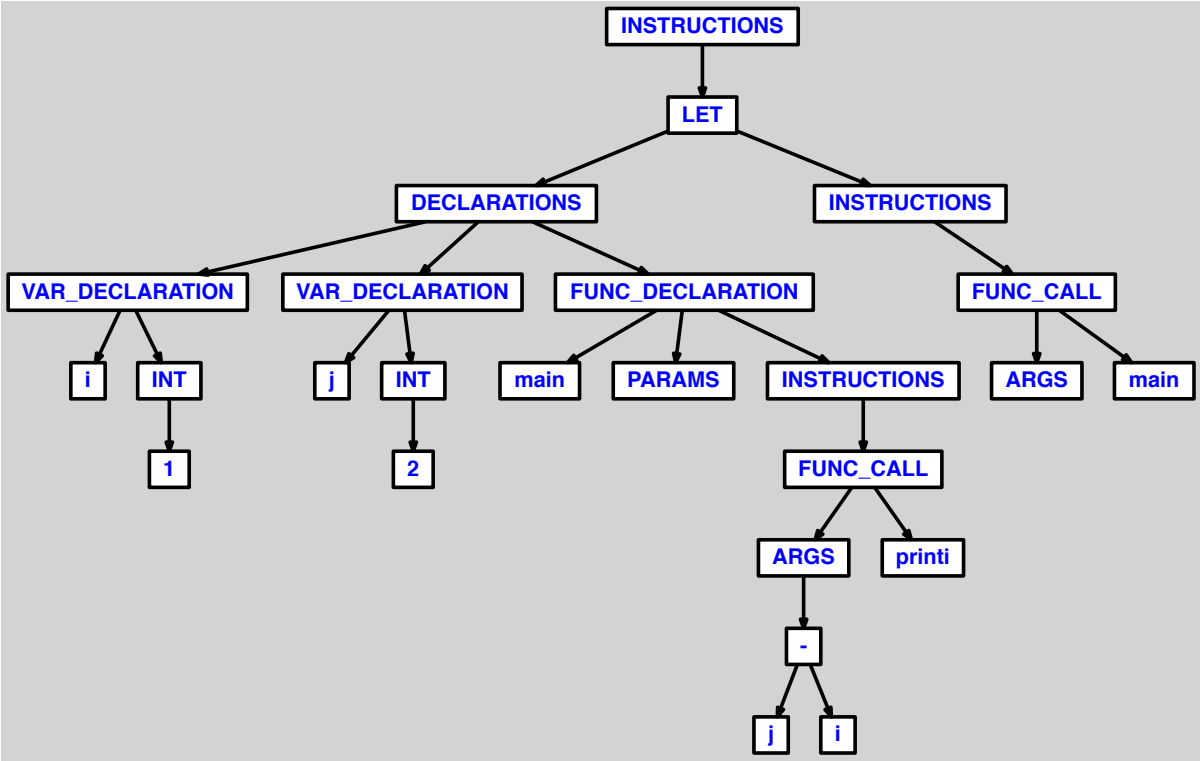
3.2.13 addition simple, a 2 termes, identifies par des variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
+
i
j

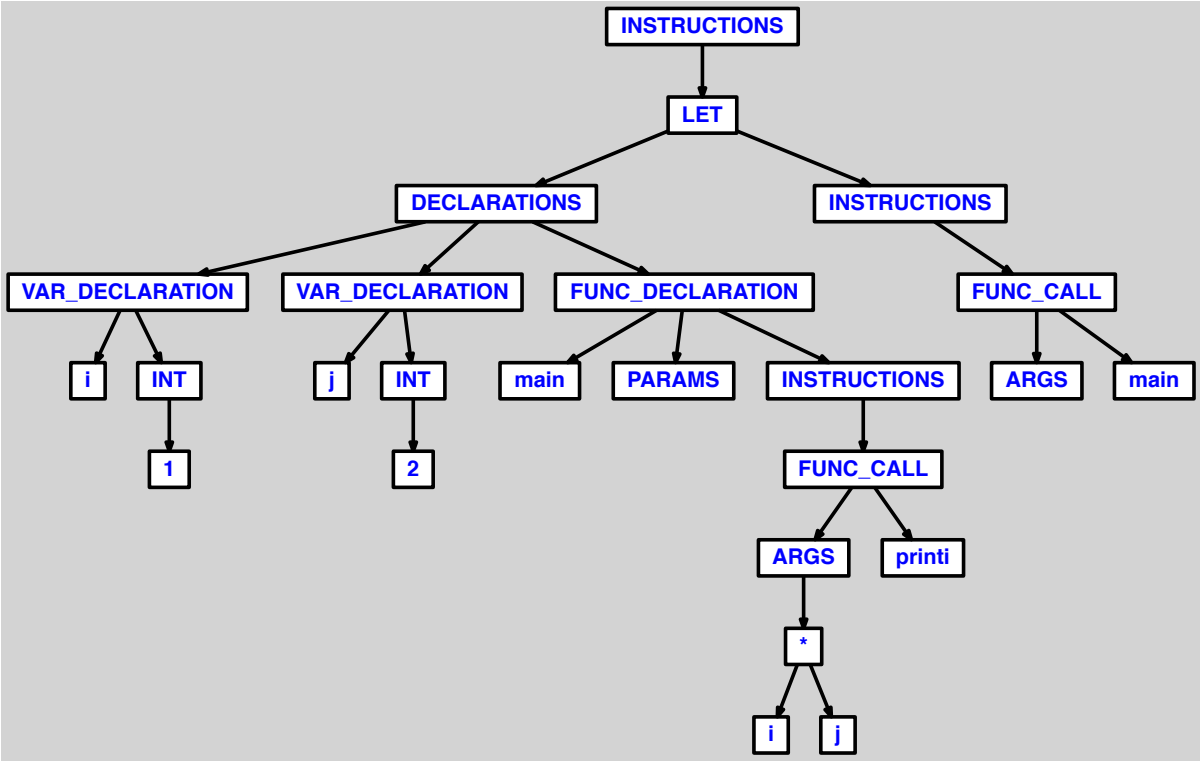
3.2.14 soustraction simple, a 2 termes, identifies par des variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
-
j
i

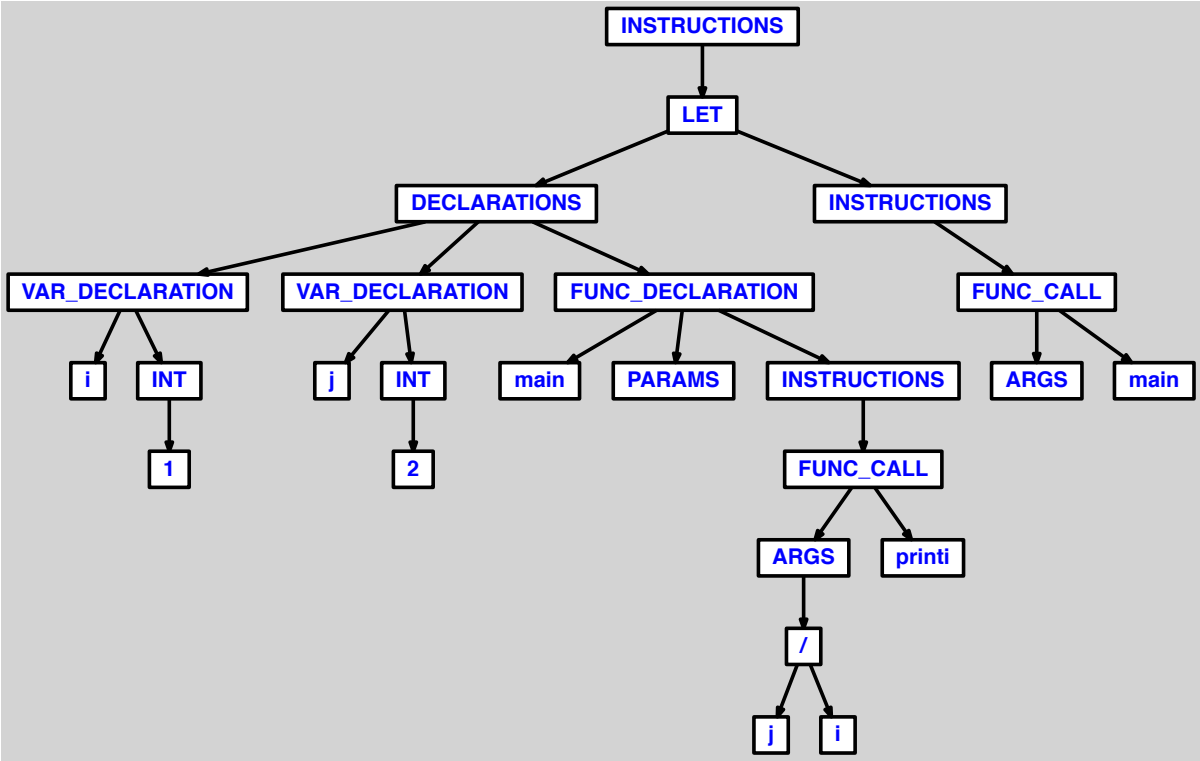
3.2.15 multiplication simple, a 2 termes, identifies par des variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
*
i
j

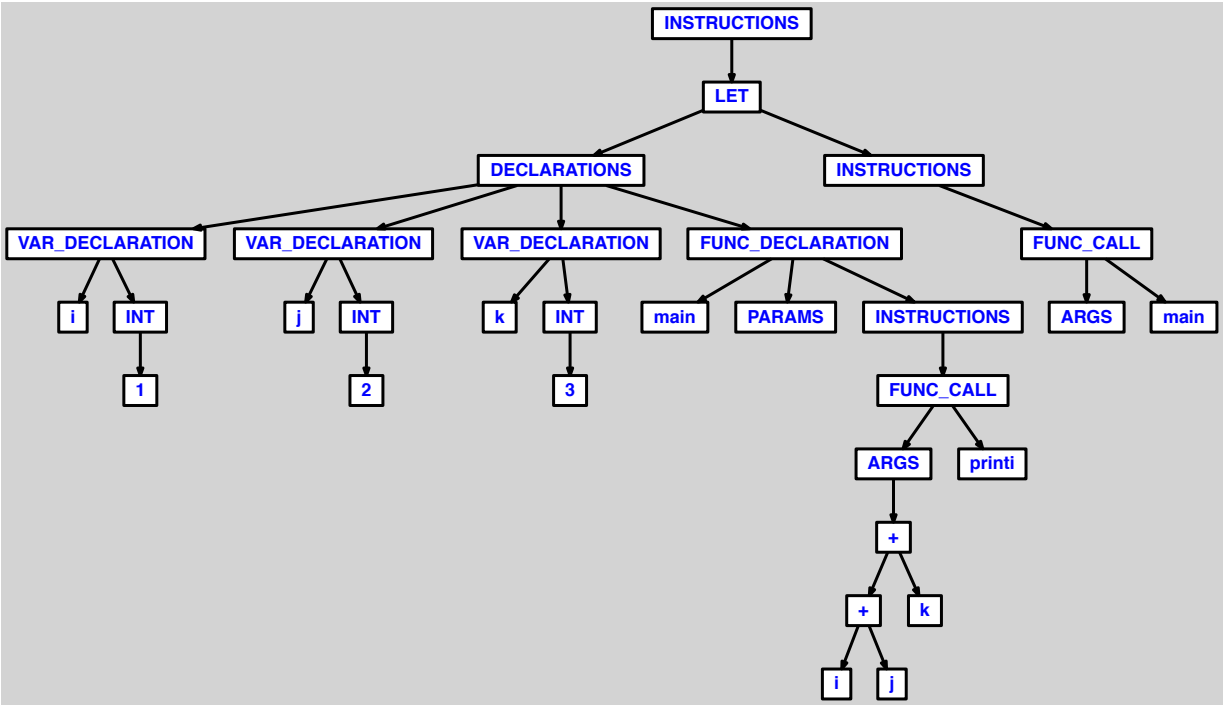
3.2.16 division simple, a 2 termes, identifies par des variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
/
j
i

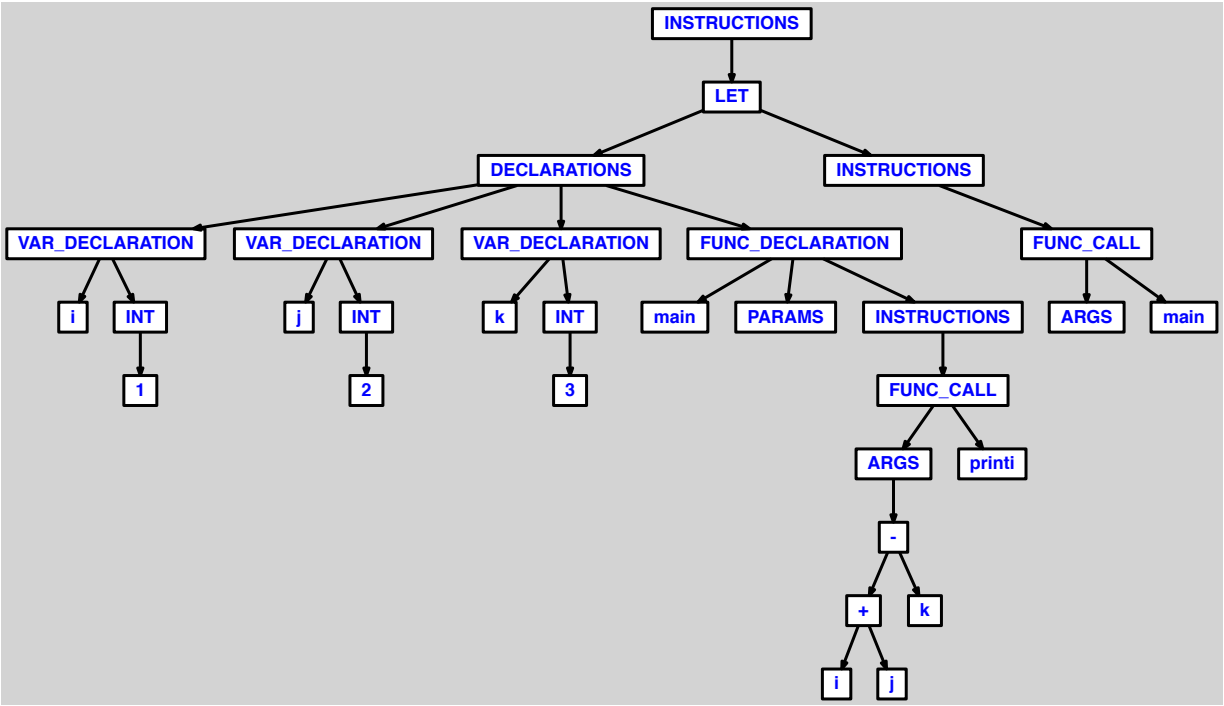
3.2.17 addition a 3 termes, identifies par des variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
+
k
i
j

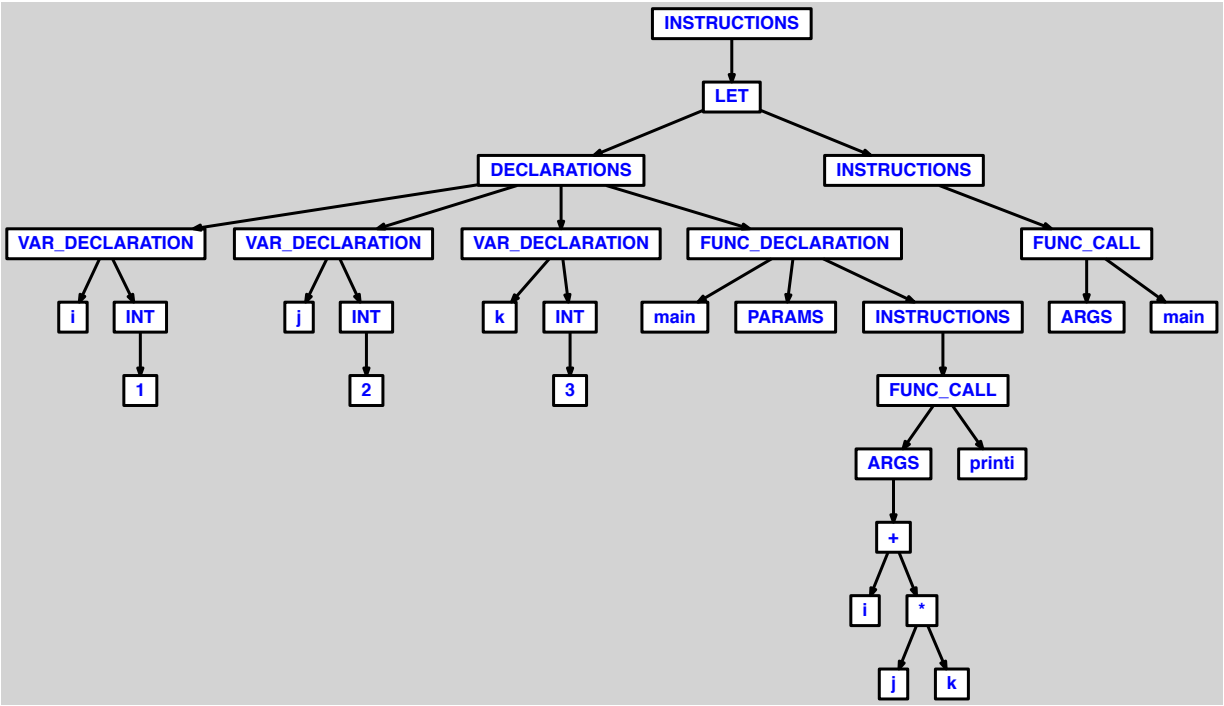
3.2.18 addition suivie de soustraction, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
-
+
k
i
j

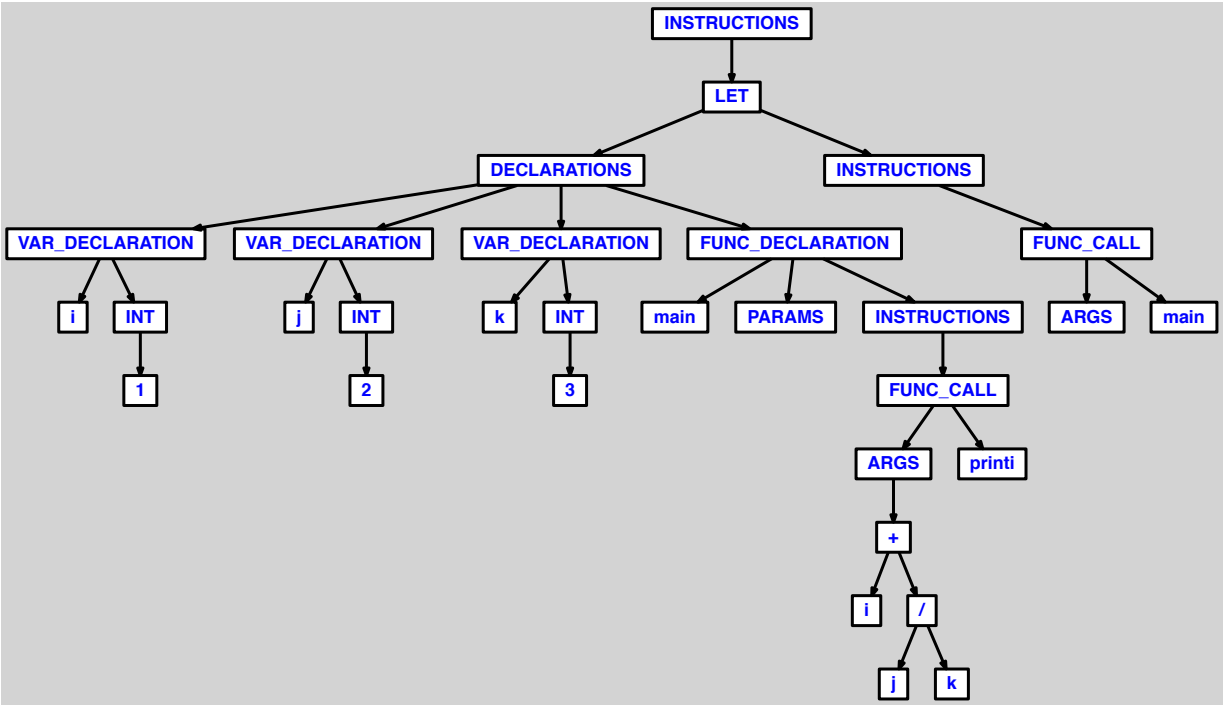
3.2.19 addition suivie de multiplication, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
i
*
j
k

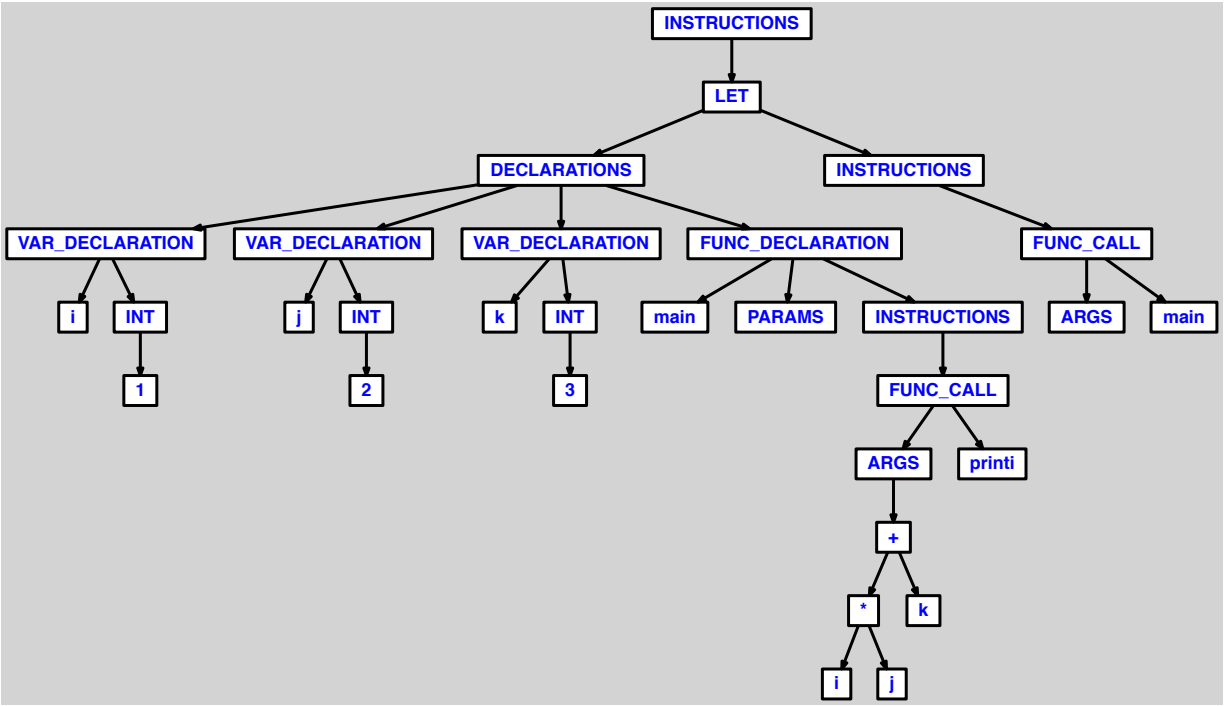
3.2.20 addition suivie de division, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
i
/
j
k

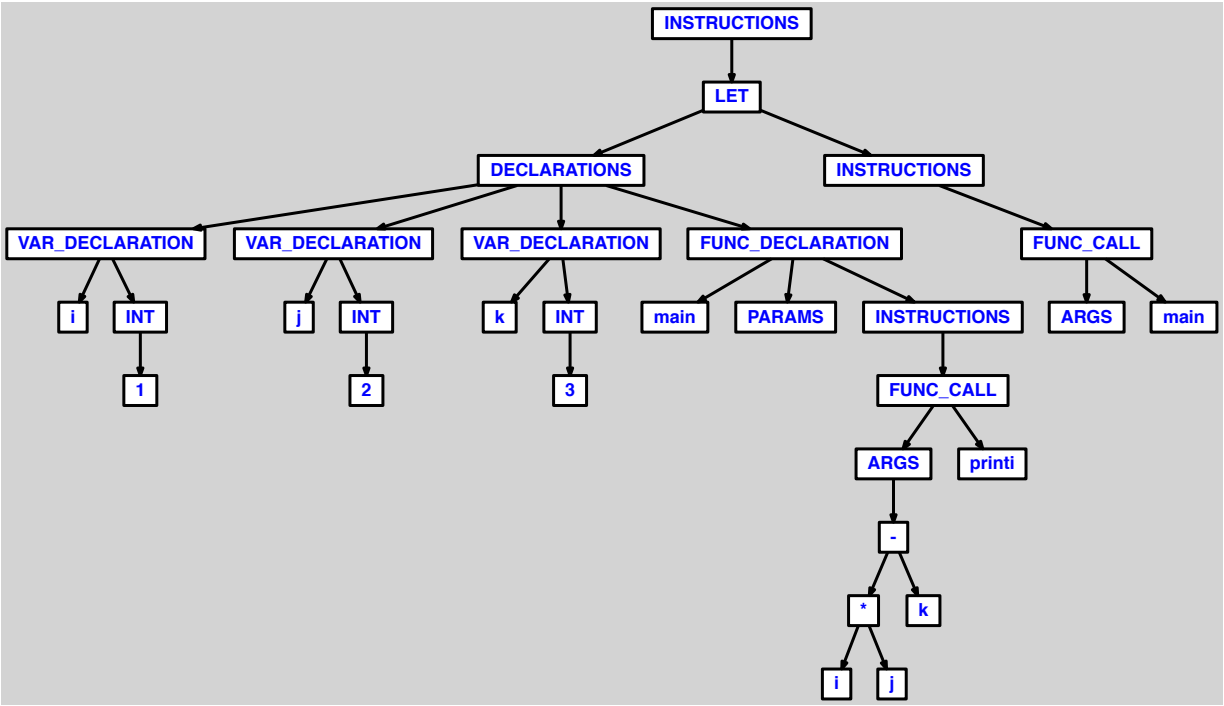
3.2.21 multiplication suivie d'addition, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
*
k
i
j

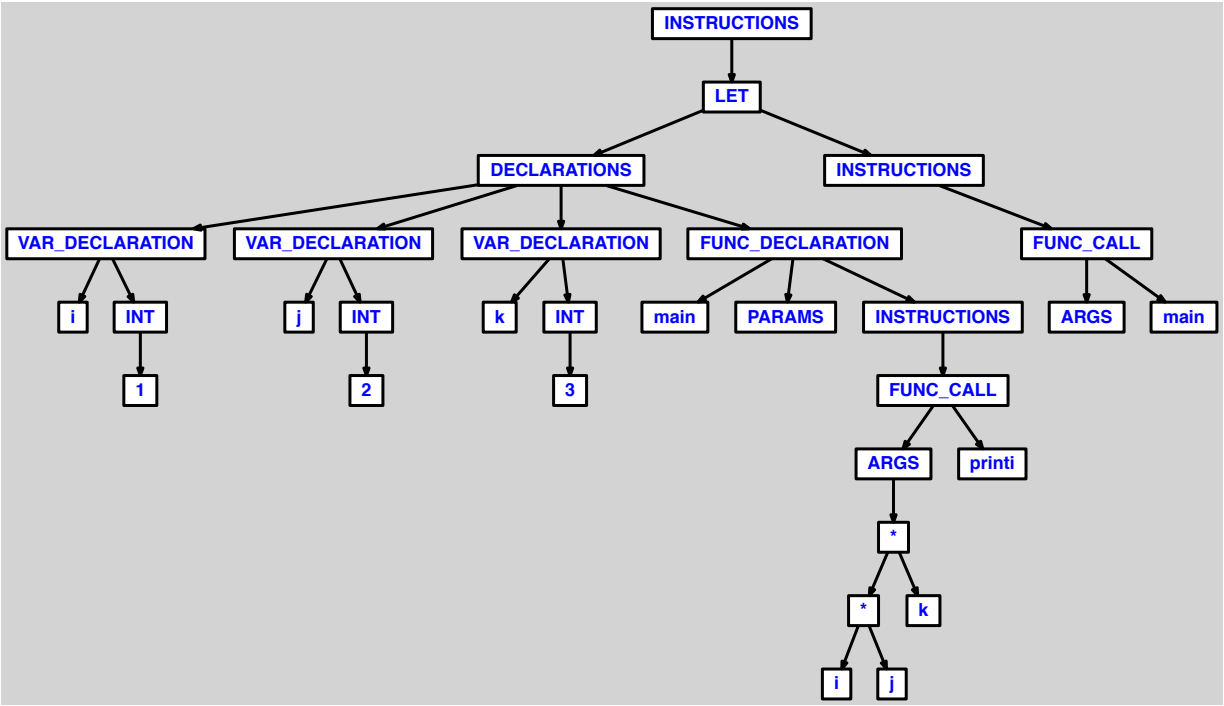
3.2.22 multiplication suivie de soustraction, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
-
*
k
i
j

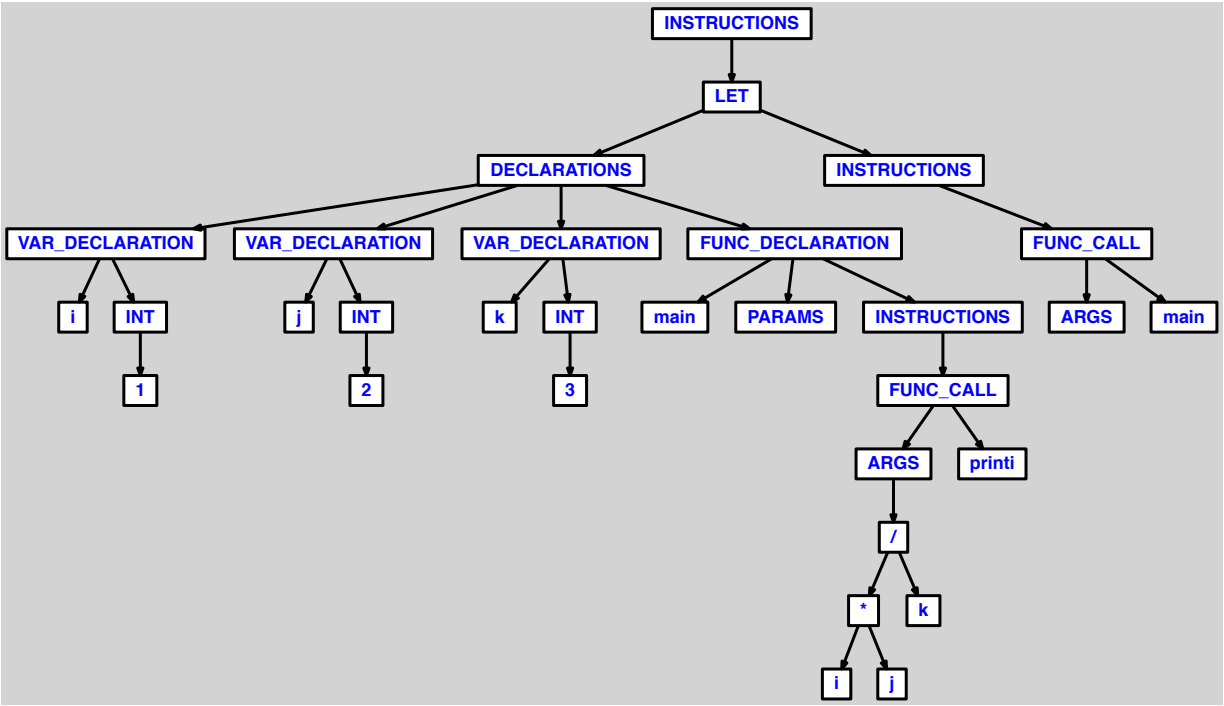
3.2.23 multiplication a 3 termes, identifies par des variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
*
k
i
j

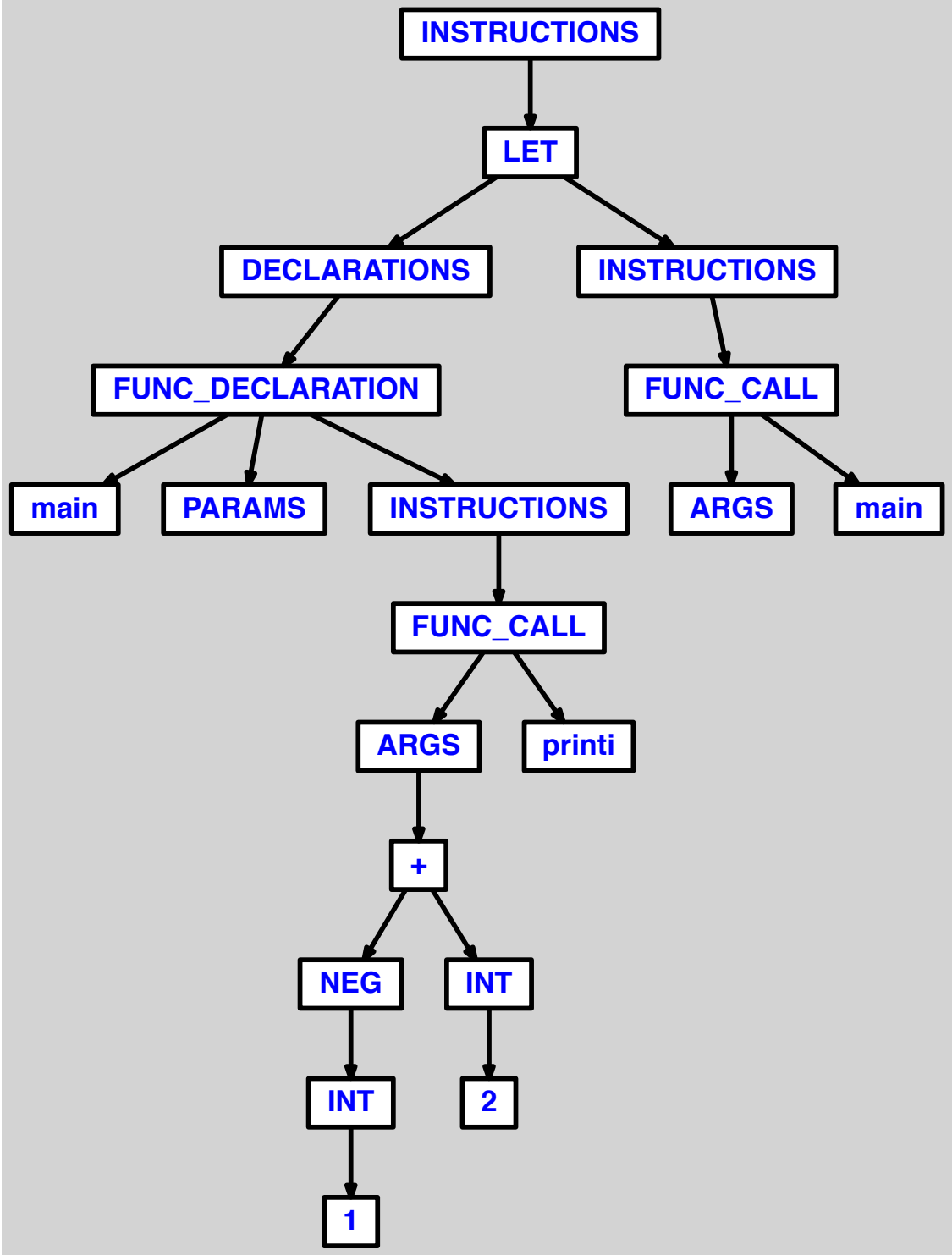
3.2.24 mutliplication suivie de division, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
/
*
k
i
j

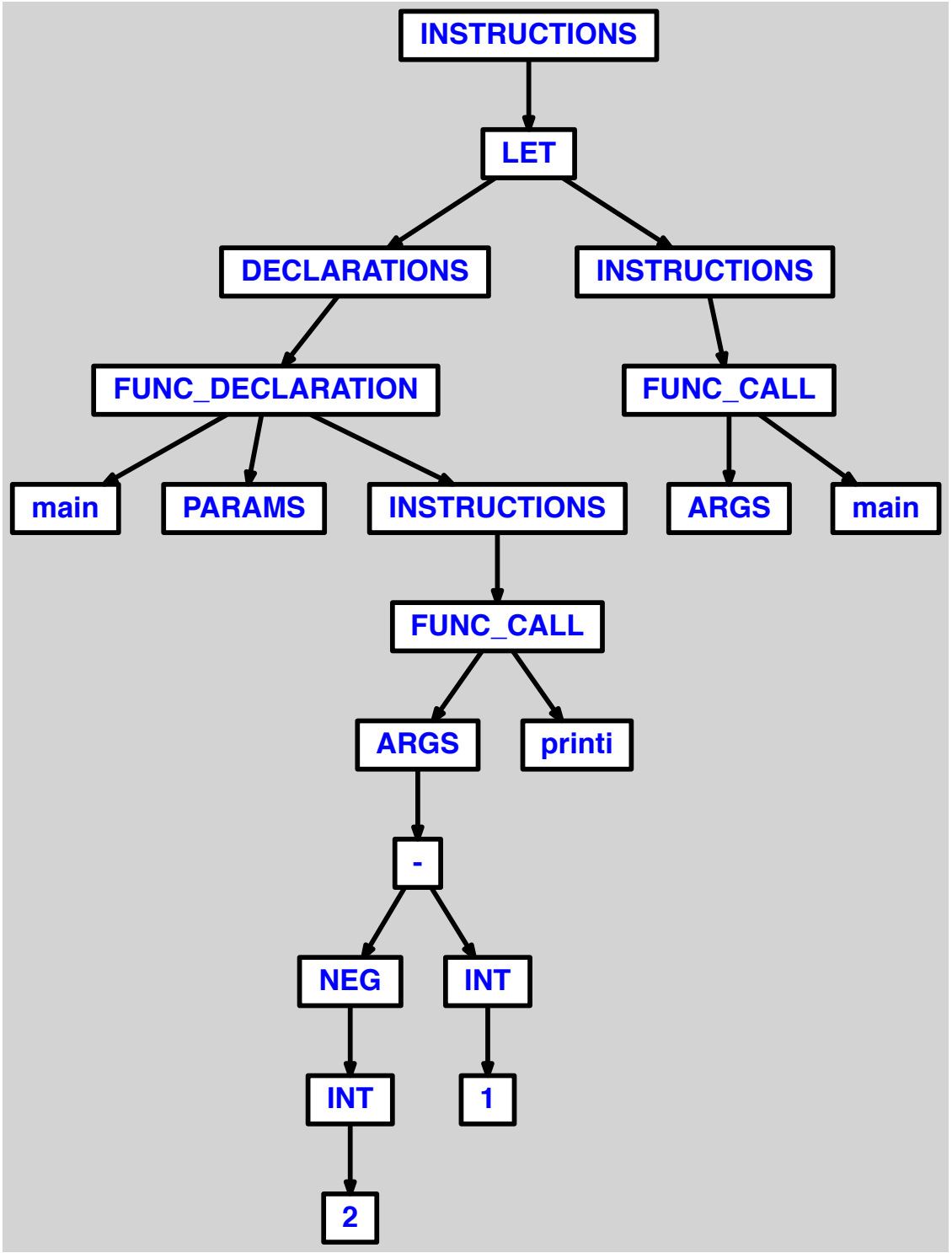
3.2.25 addition simple, a 2 termes, dont un ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
NEG
INT
INT
2
1

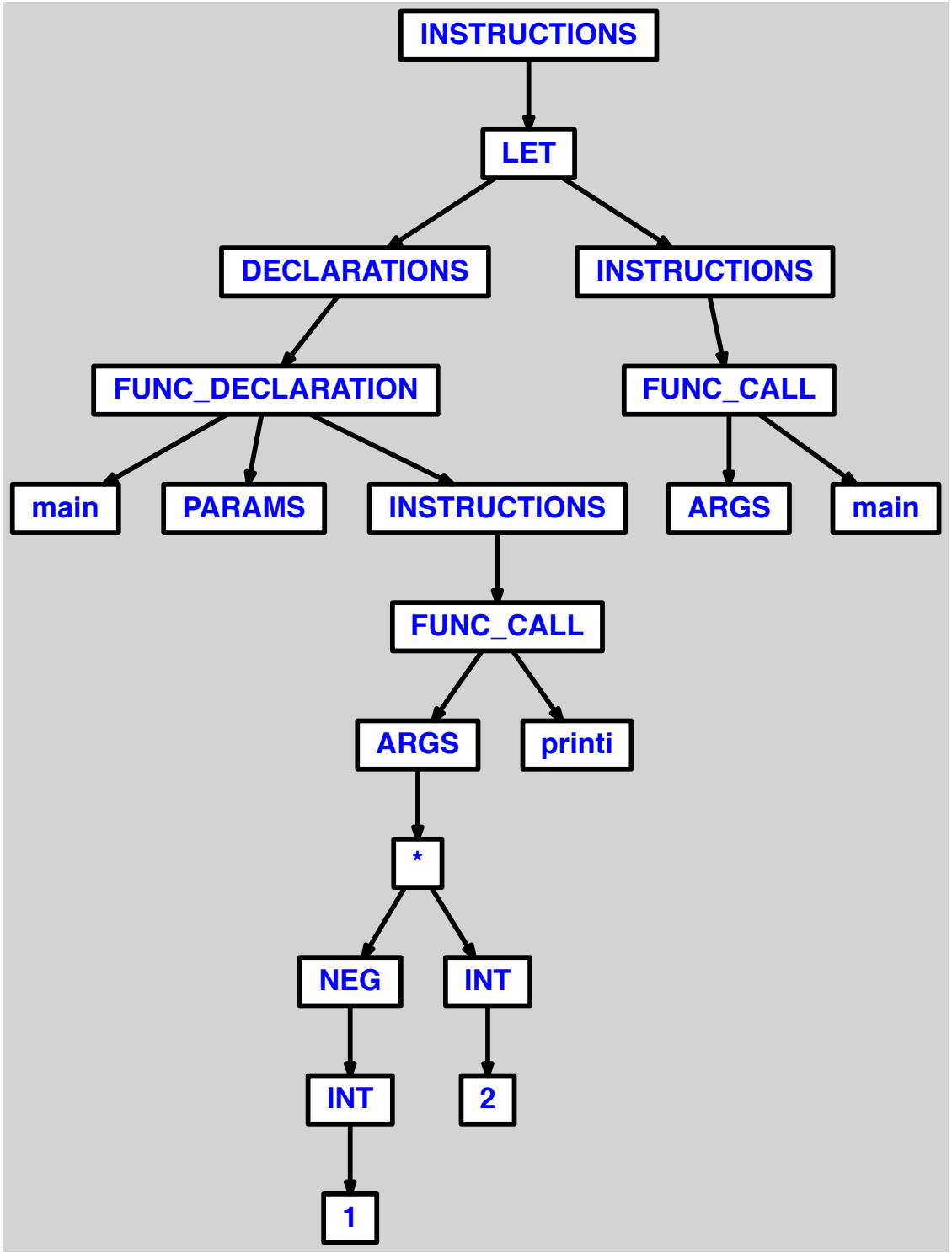
3.2.26 soustraction simple, a 2 termes, dont ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
NEG
INT
INT
1
2

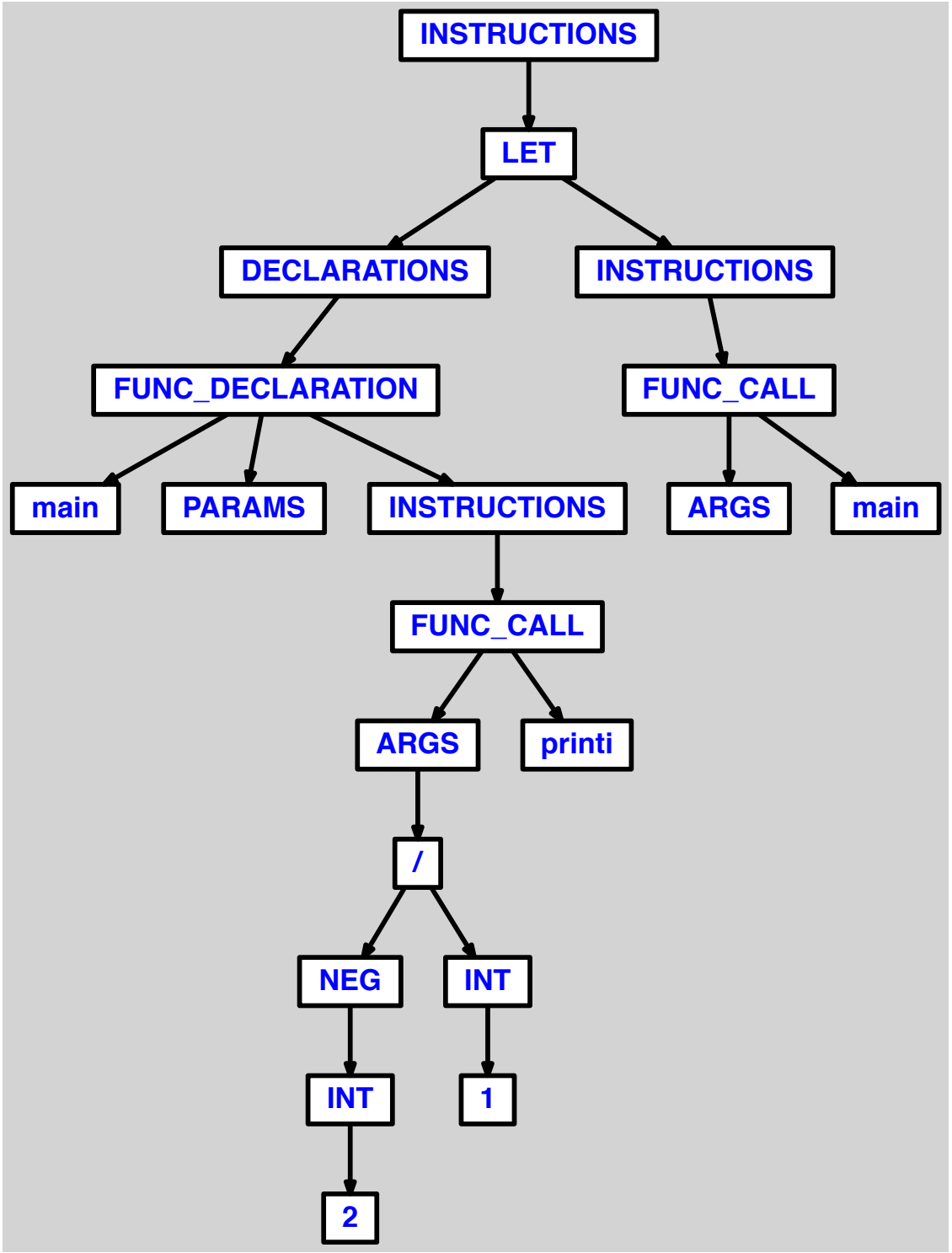
3.2.27 multiplication simple, a 2 termes, dont un ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
NEG
INT
INT
2
1

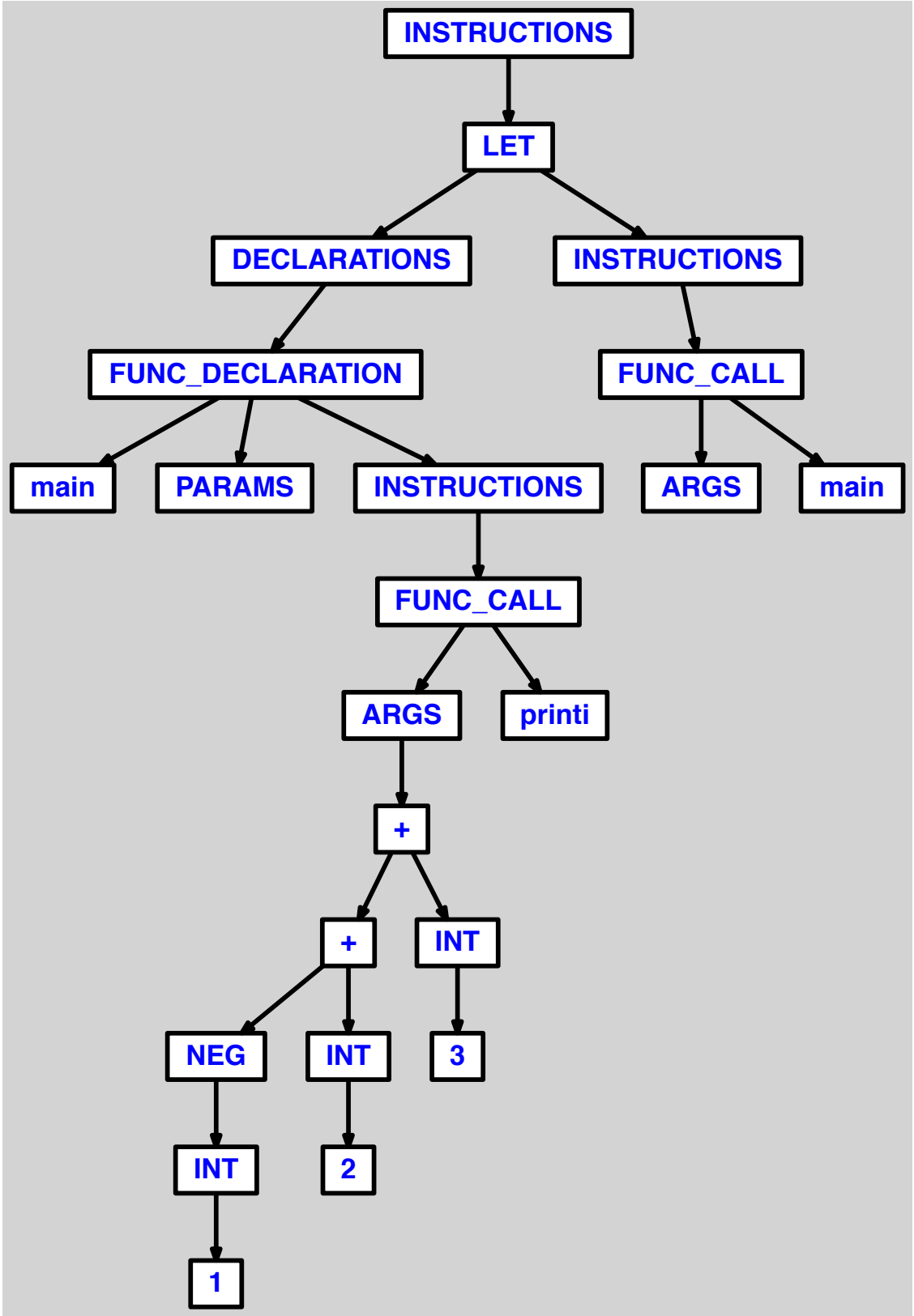
3.2.28 division simple, a 2 termes, dont un ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
NEG
INT
INT
1
2

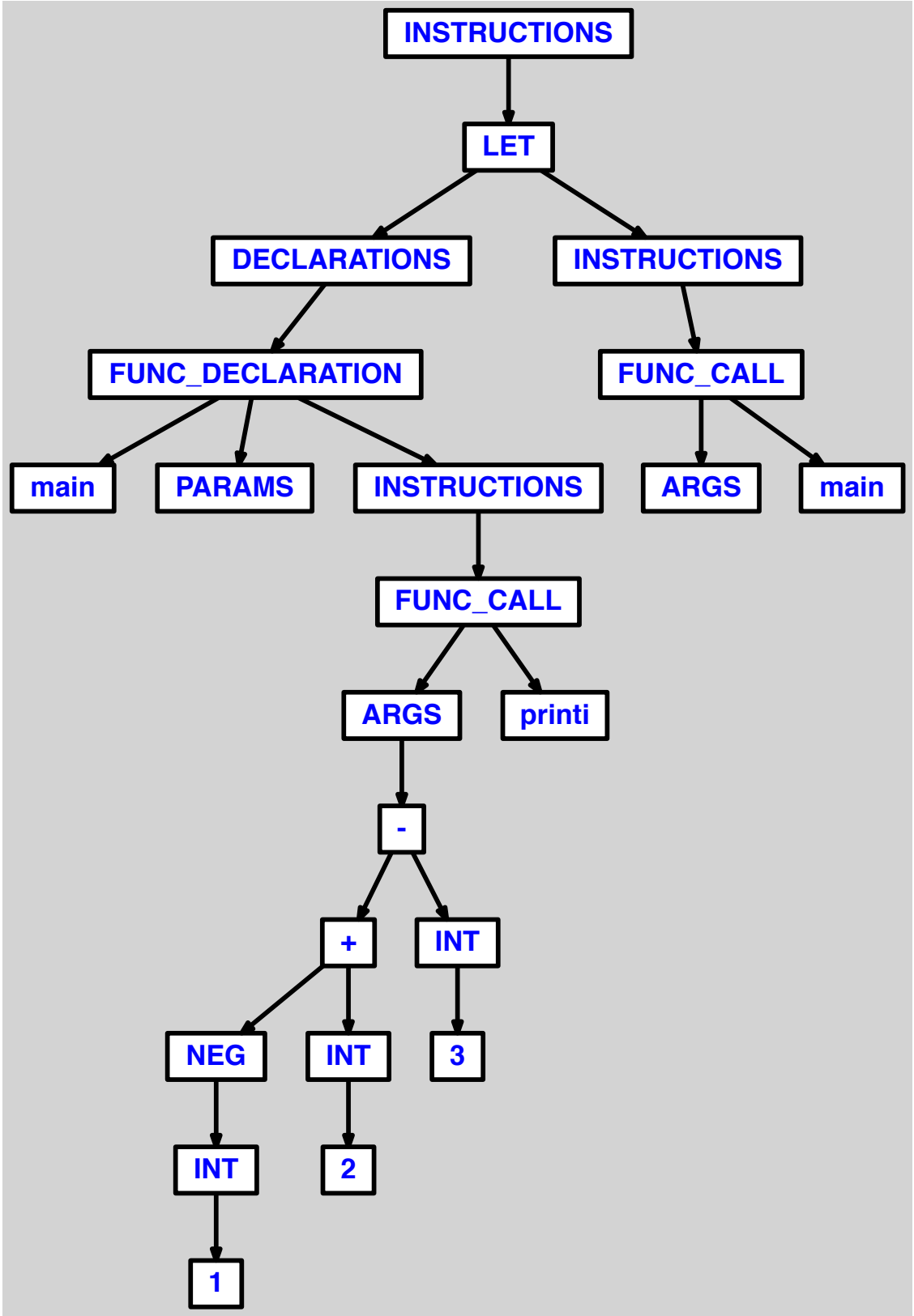
3.2.29 addition a 3 termes, dont un ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
+
INT
NEG
INT
3
INT
2
1

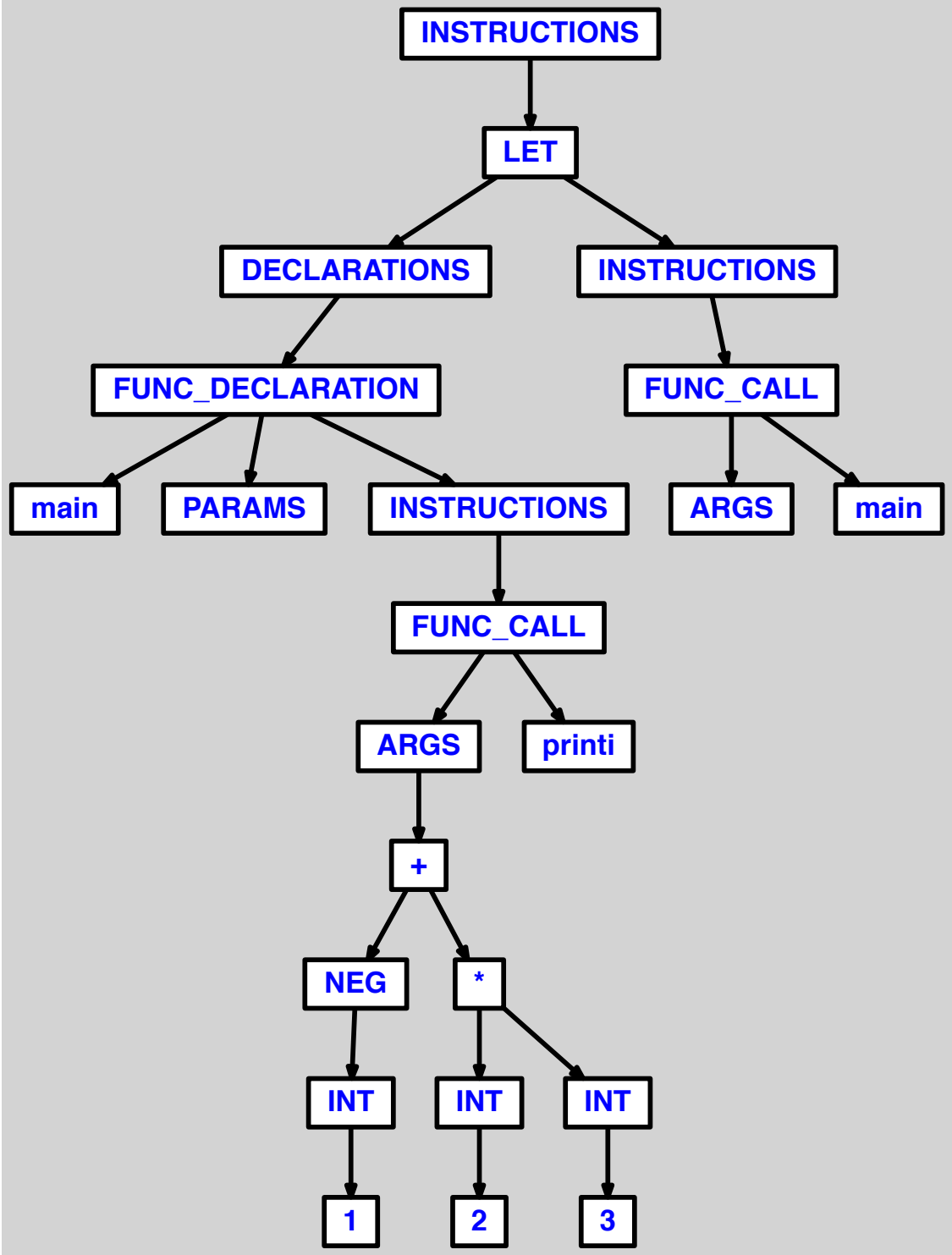
3.2.30 addition suivie de soustraction, avec un terme ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
+
INT
NEG
INT
3
INT
2
1

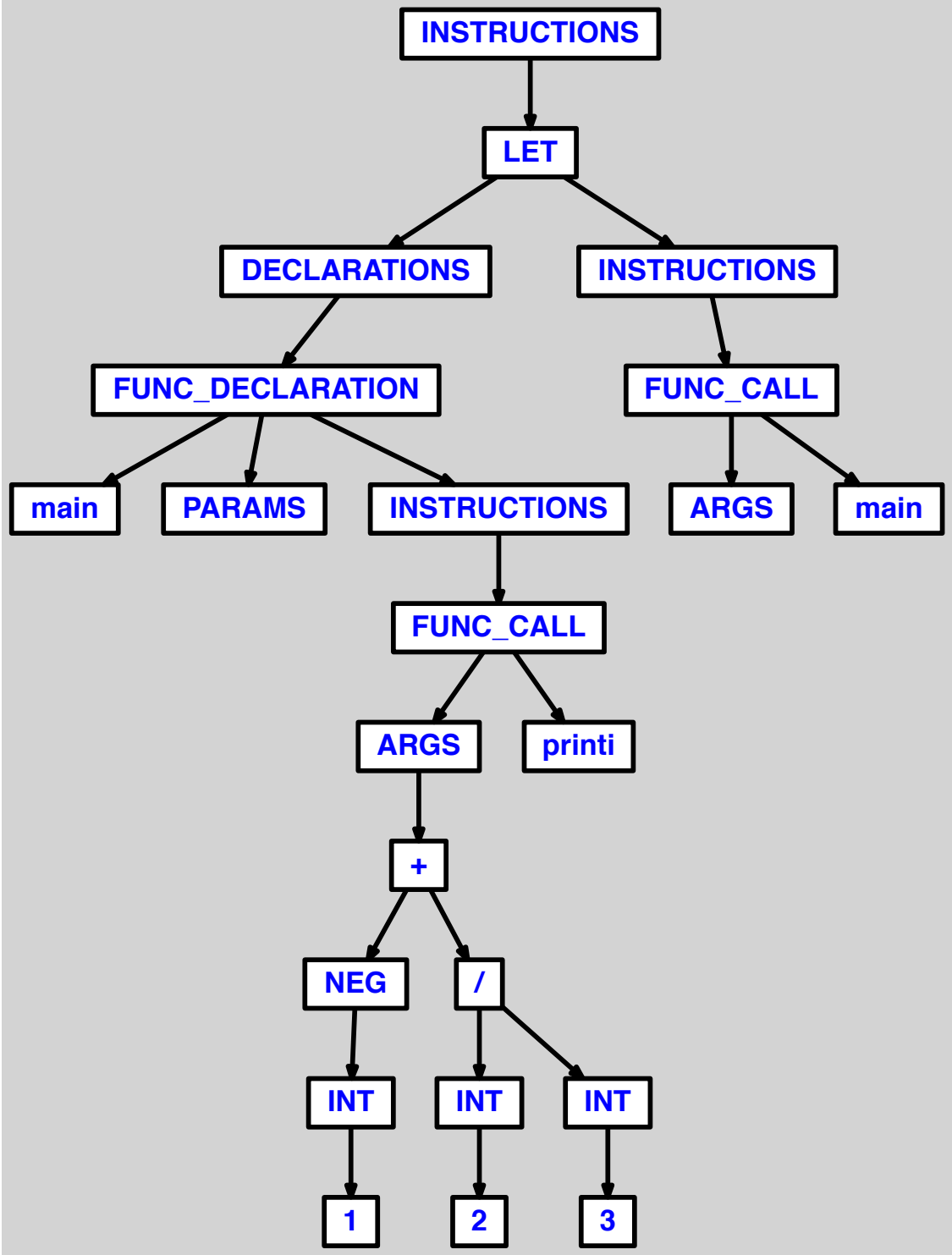
3.2.31 addition suivie de multiplication, avec un terme ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
NEG
*
INT
INT
INT
1
2
3

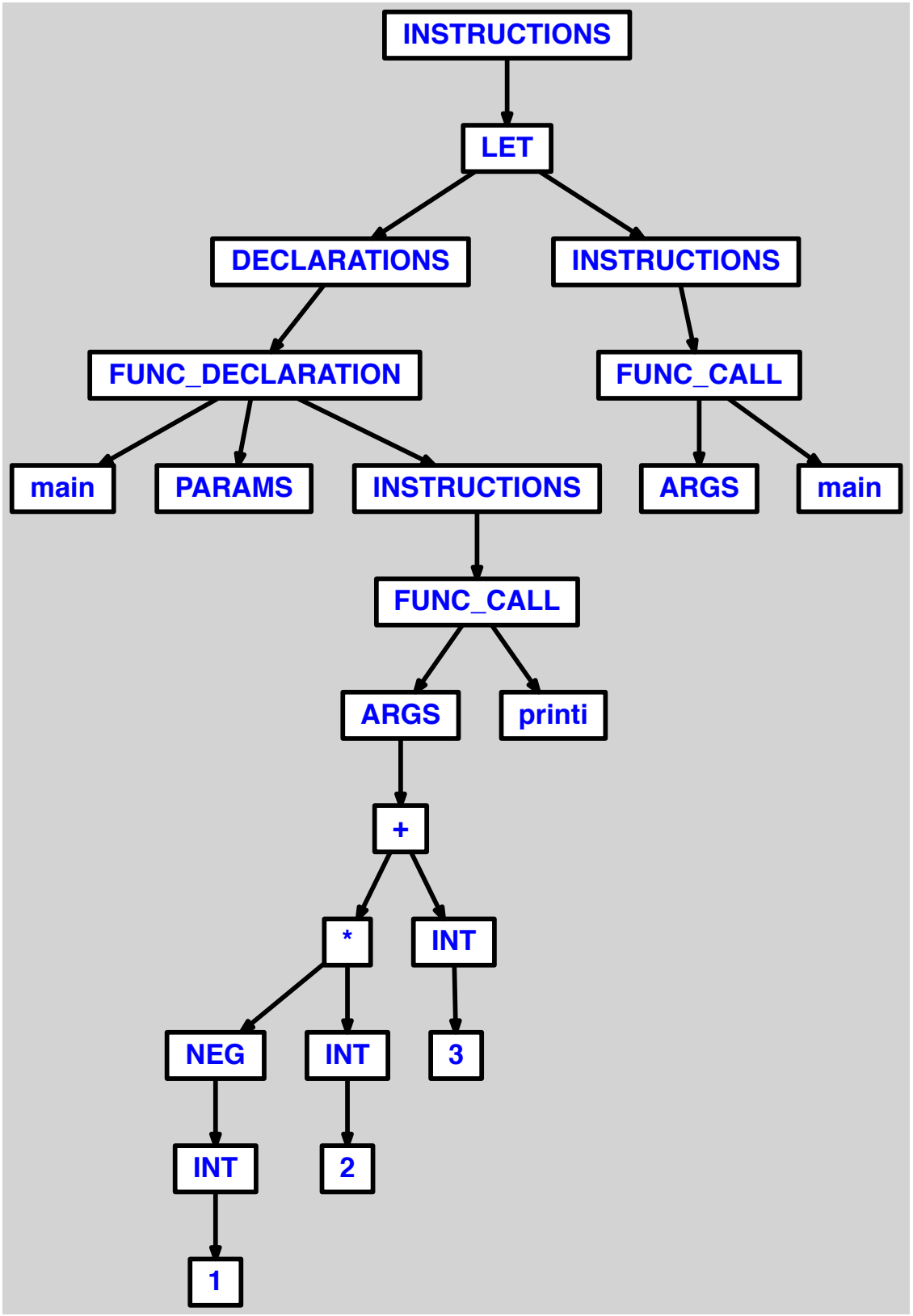
3.2.32 addition suivie de division, avec un terme ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
NEG
/
INT
INT
INT
1
2
3

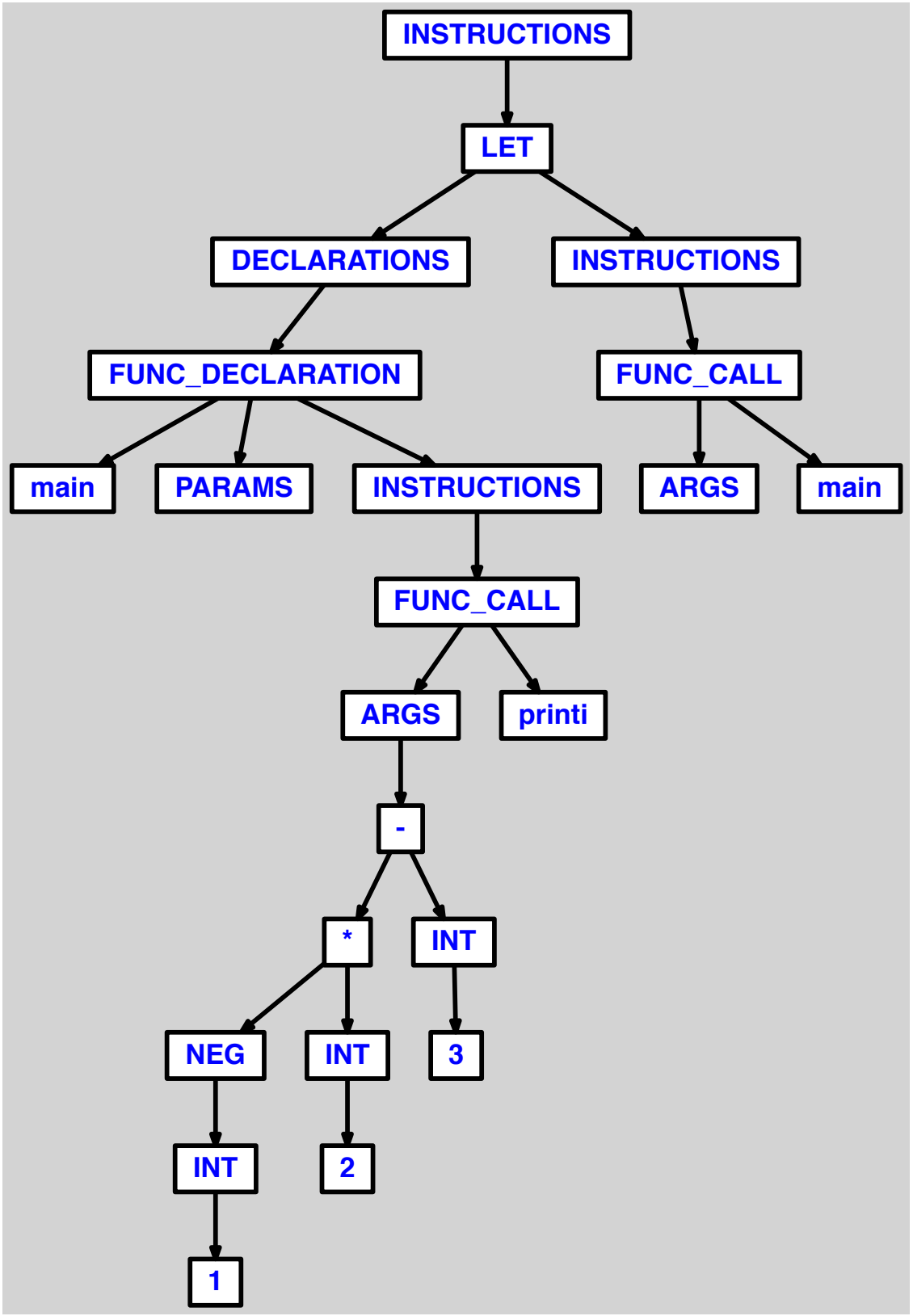
3.2.33 multiplication suivie d'addition, dont un terme ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
*
INT
NEG
INT
3
INT
2
1

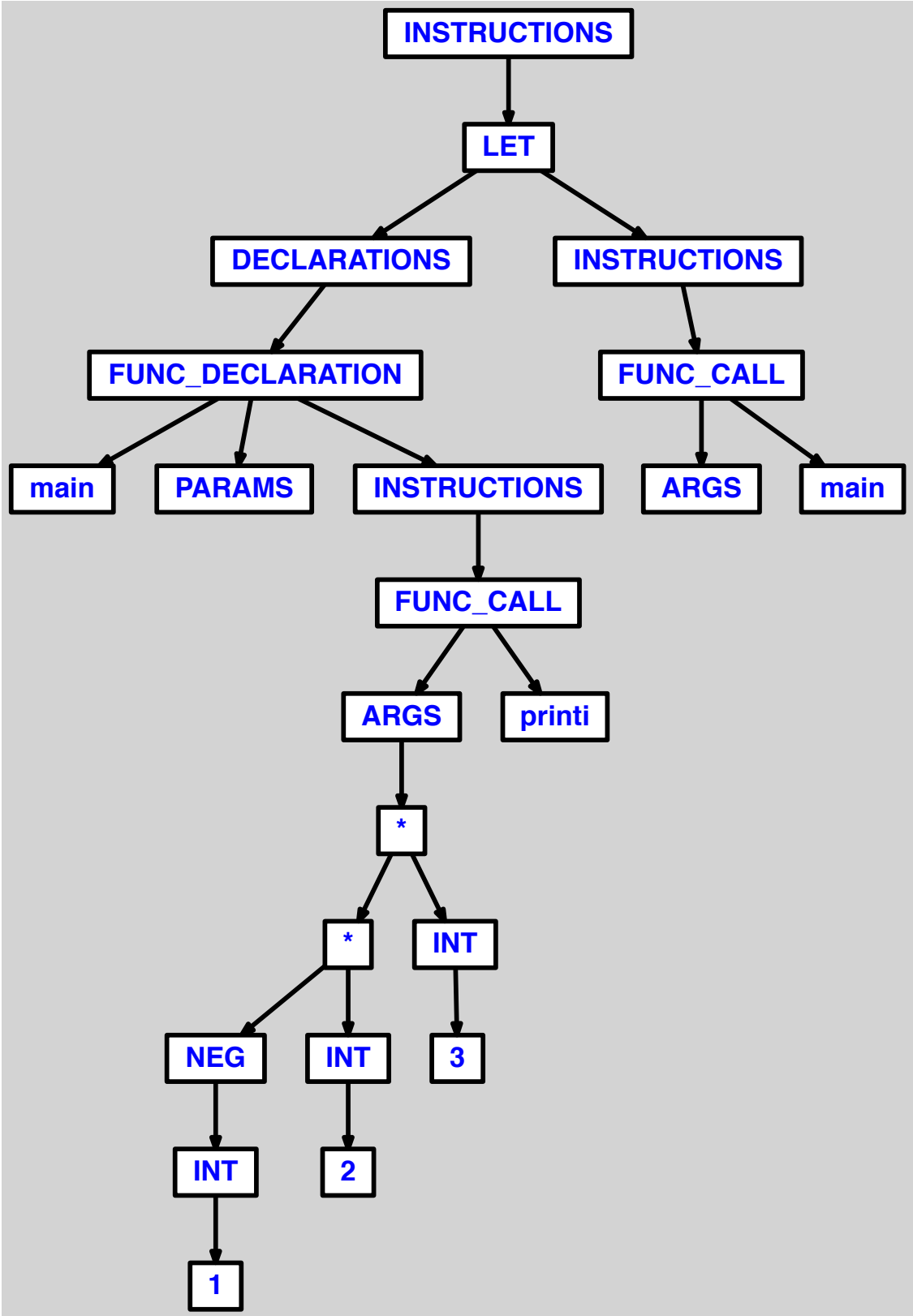
3.2.34 multiplication suivie de soustraction, dont un terme ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
*
INT
NEG
INT
3
INT
2
1

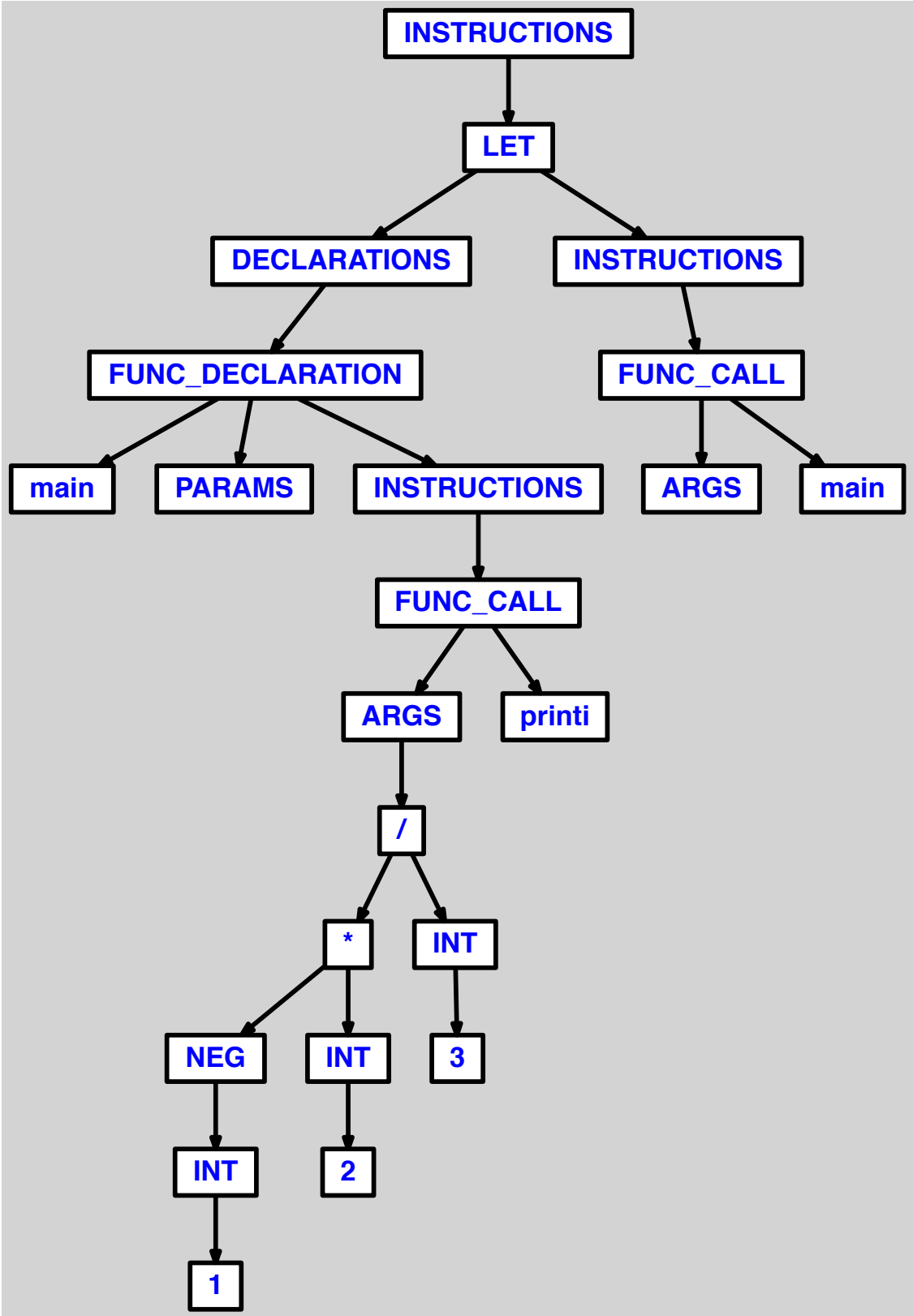
3.2.35 multiplication a 3 termes, dont un ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
*
INT
NEG
INT
3
INT
2
1

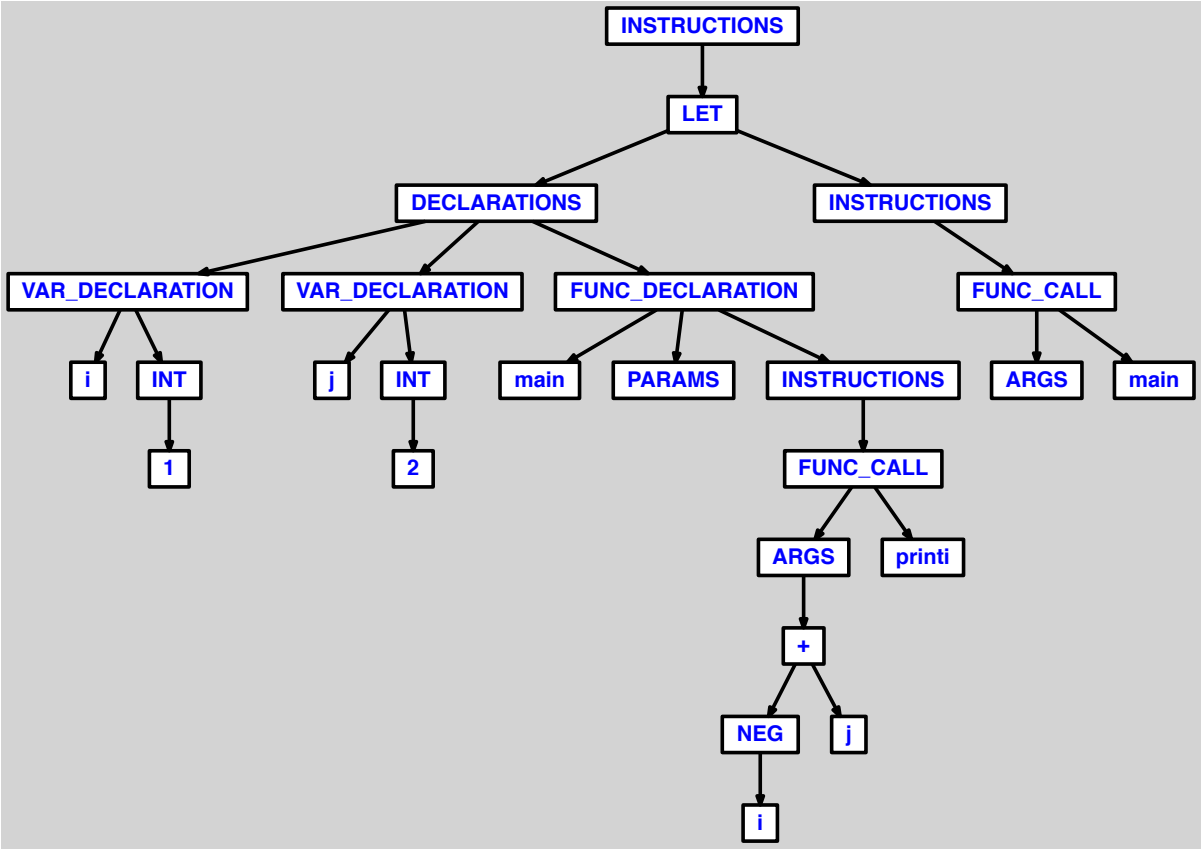
3.2.36 multiplication suivie de division, dont un terme ayant moins unaire

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
*
INT
NEG
INT
3
INT
2
1

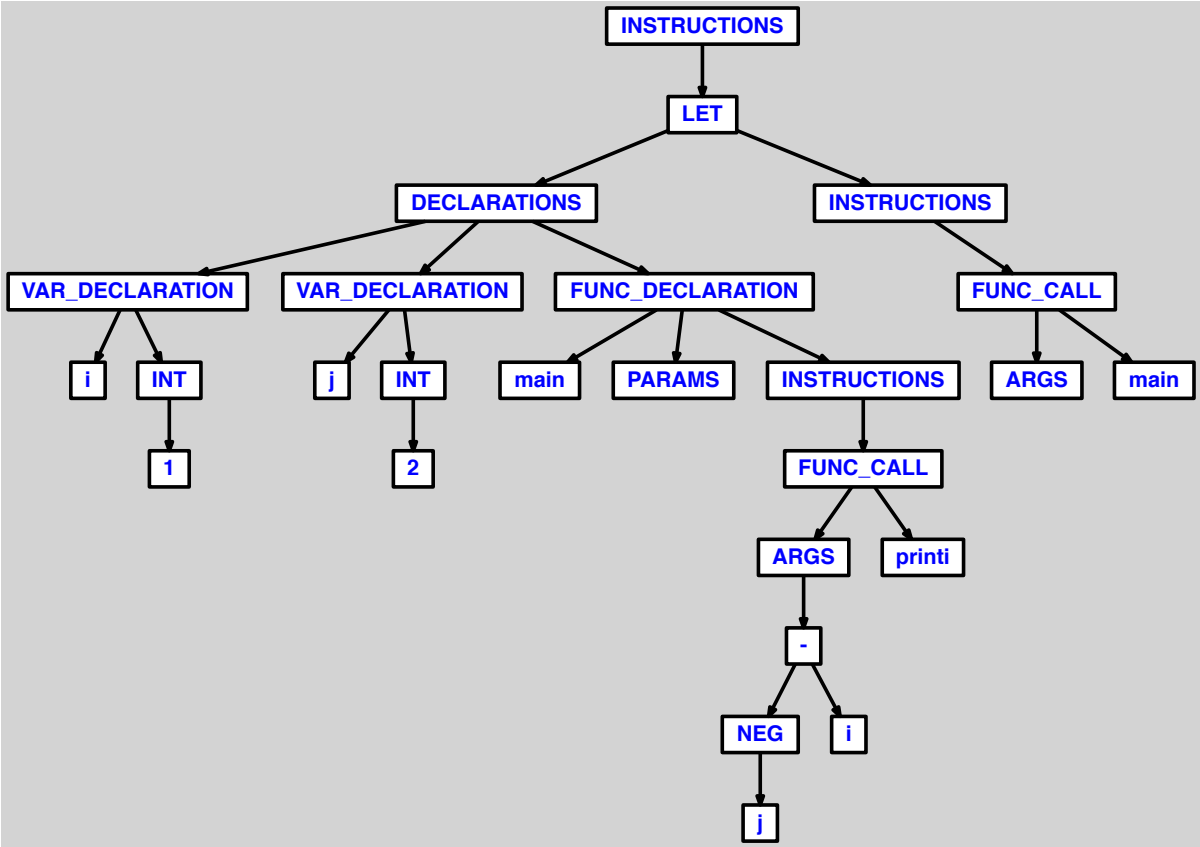
3.2.37 addition simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
+
NEG
j
i

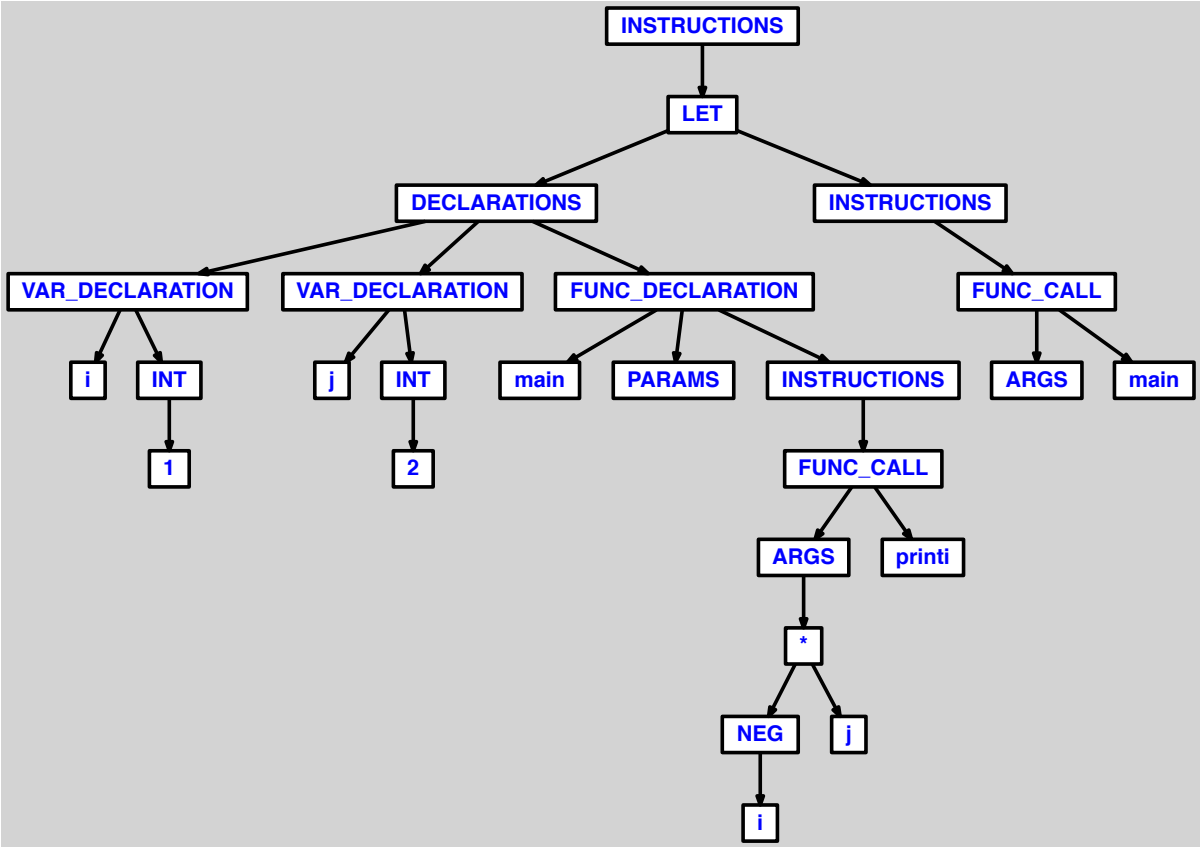
3.2.38 soustraction simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
-
NEG
i
j

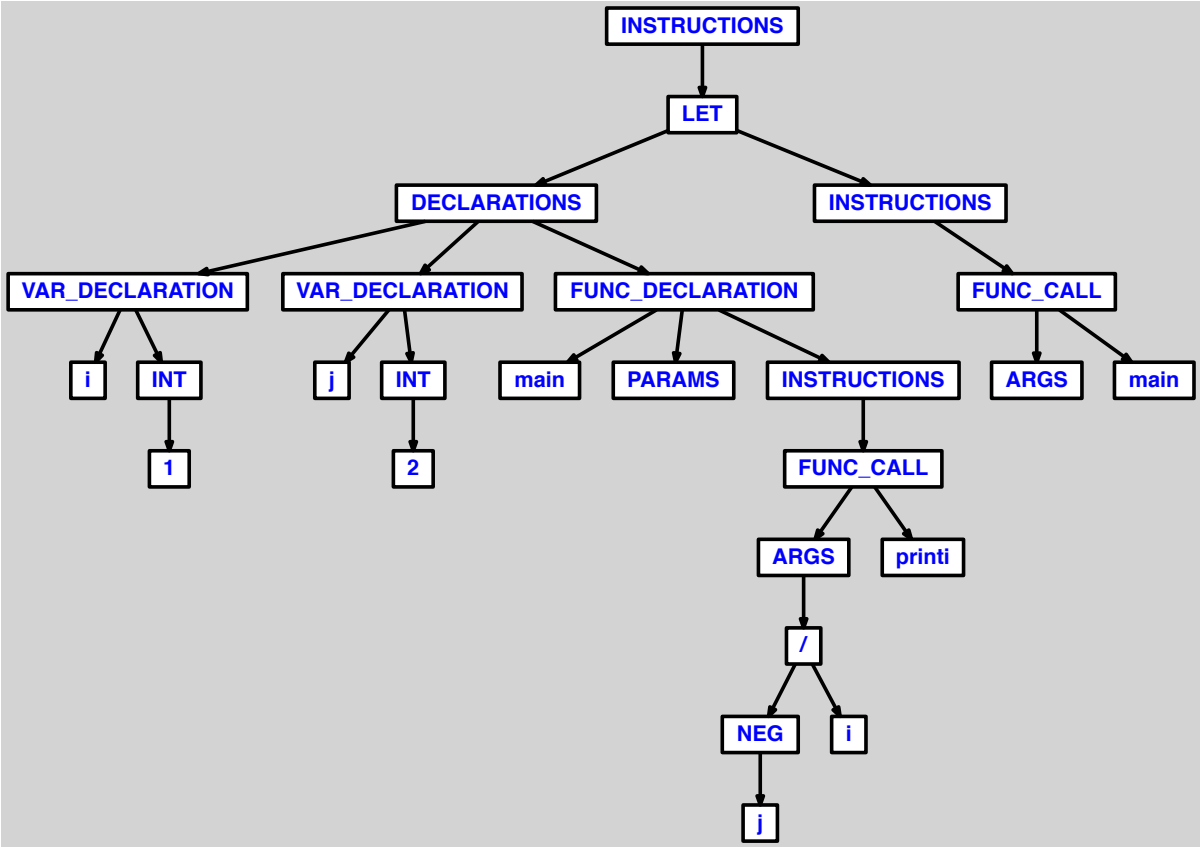
3.2.39 multiplication simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
*
NEG
j
i

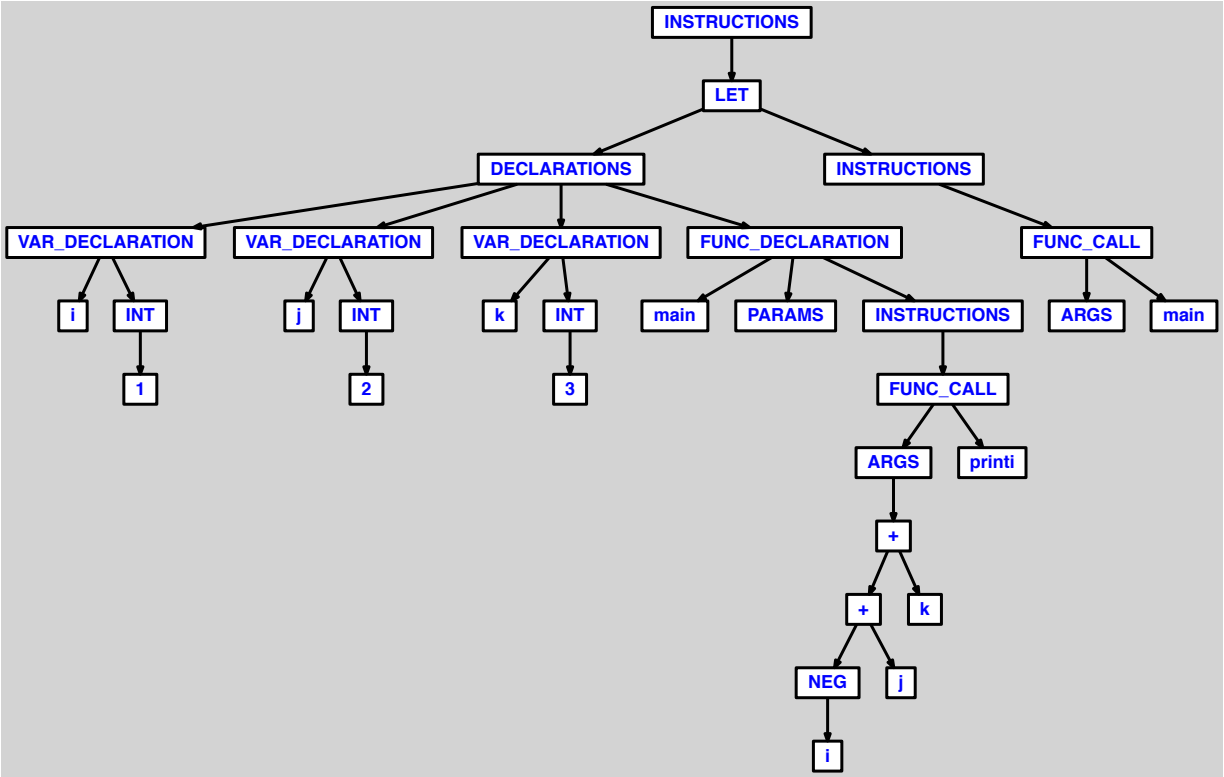
3.2.40 division simple, a 2 termes, identifies par des variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
FUNC_CALL
ARGS
printi
/
NEG
i
j

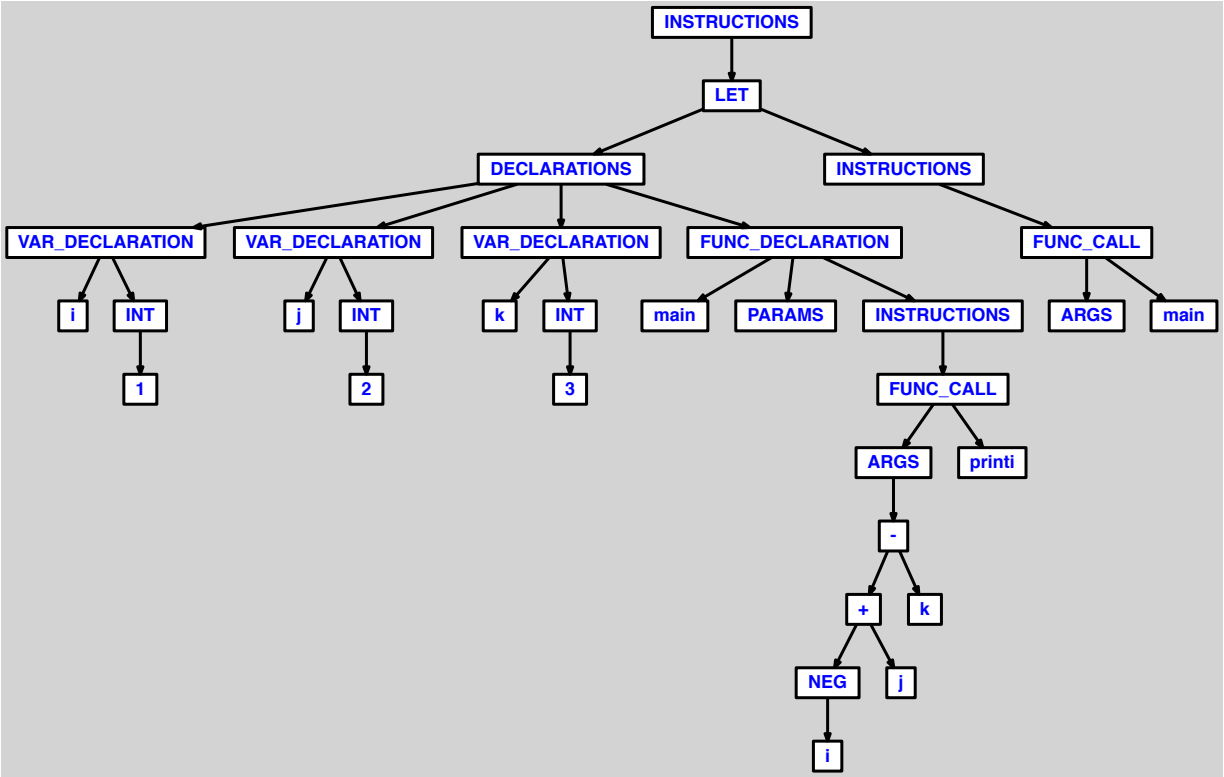
3.2.41 addition a 3 termes, identifies par des variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
+
k
NEG
j
i

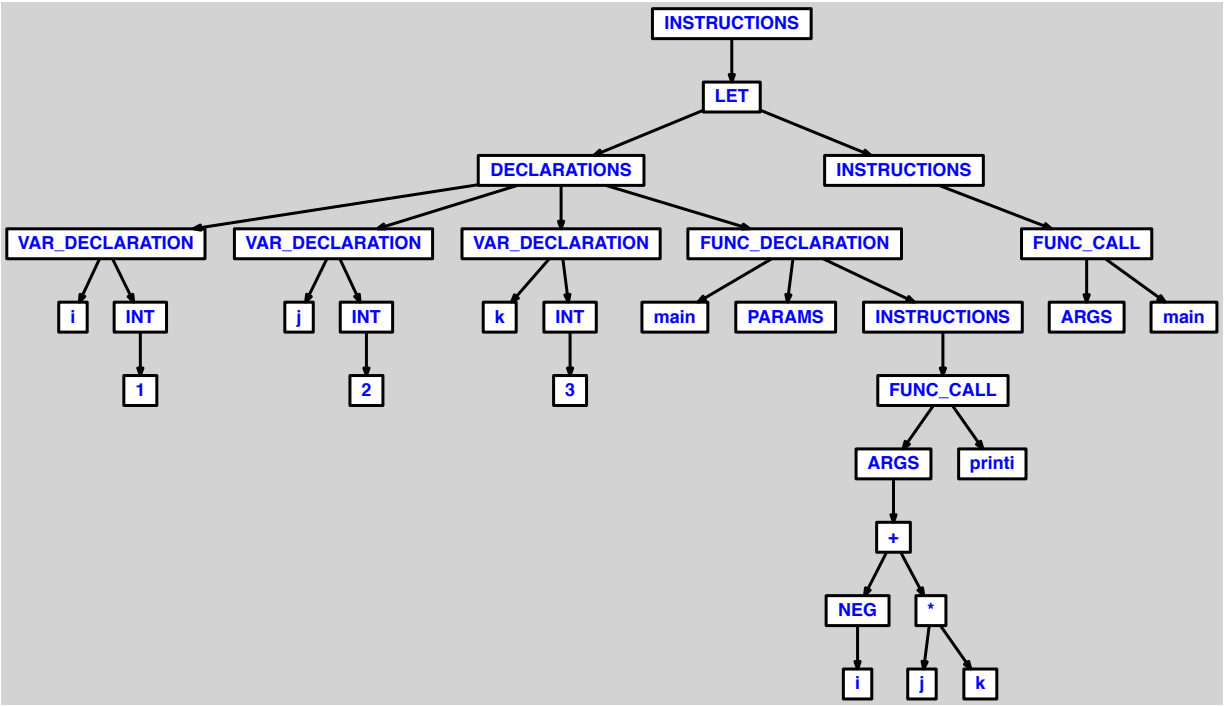
3.2.42 addition suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
-
+
k
NEG
j
i

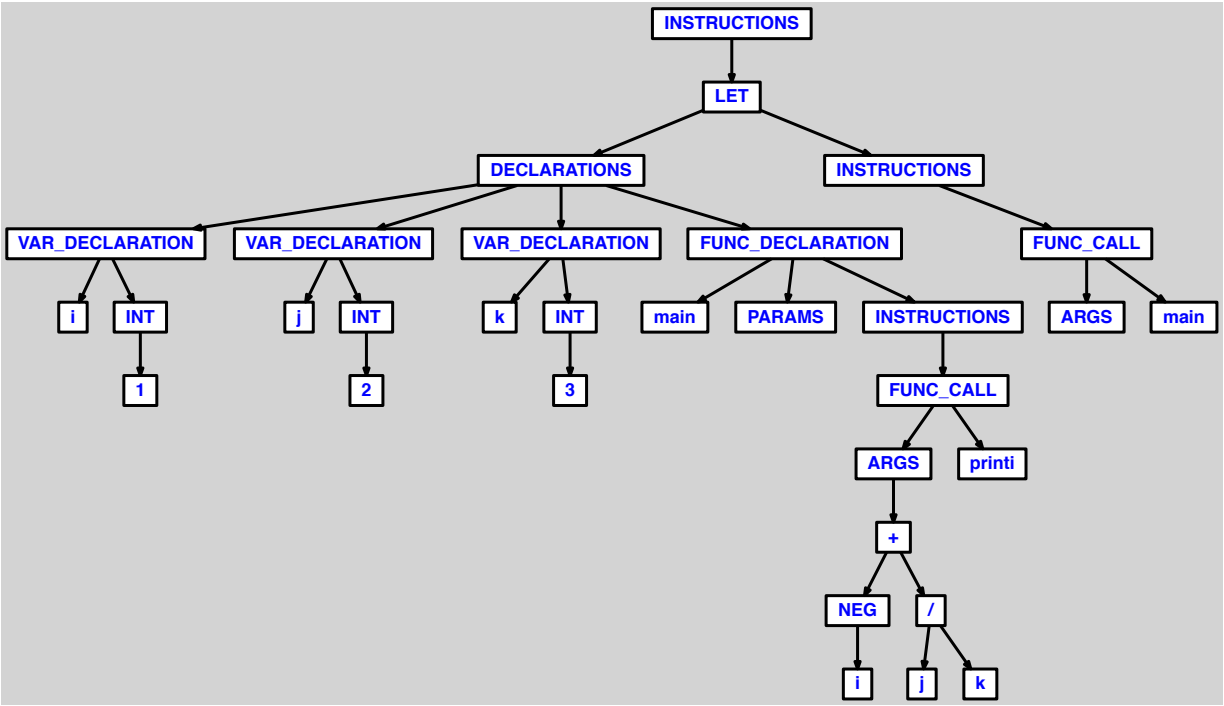
3.2.43 addition suivie de multiplication, avec termes identifies par variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
NEG
*
i
j
k

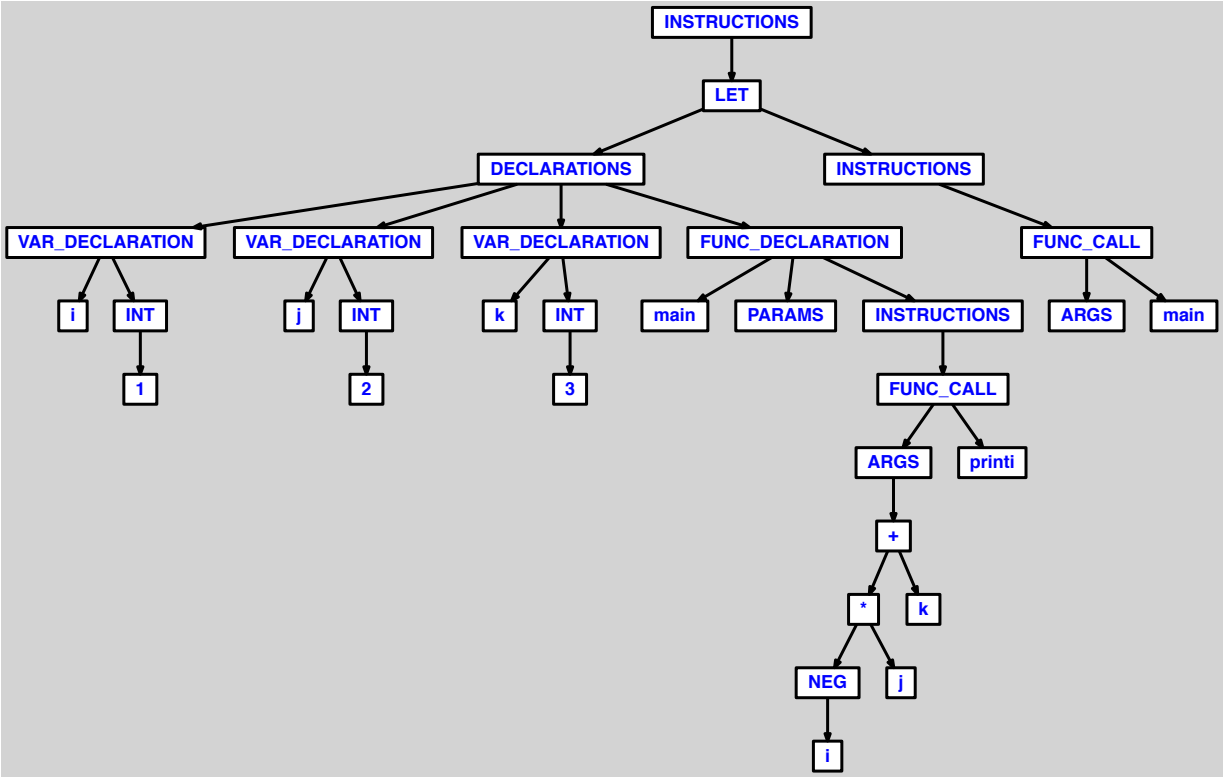
3.2.44 addition suivie de division, avec termes identifies par variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
NEG
/
i
j
k

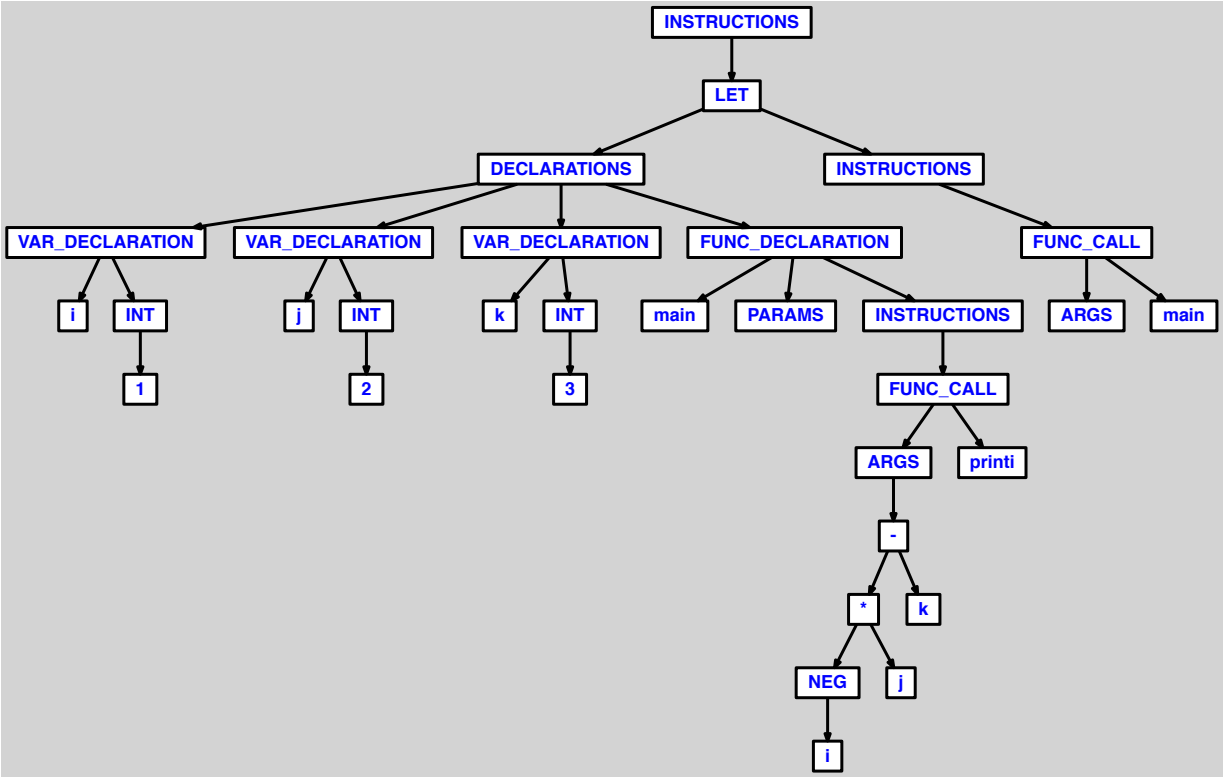
3.2.45 multiplication suivie d'addition, avec termes identifies par variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
*
k
NEG
j
i

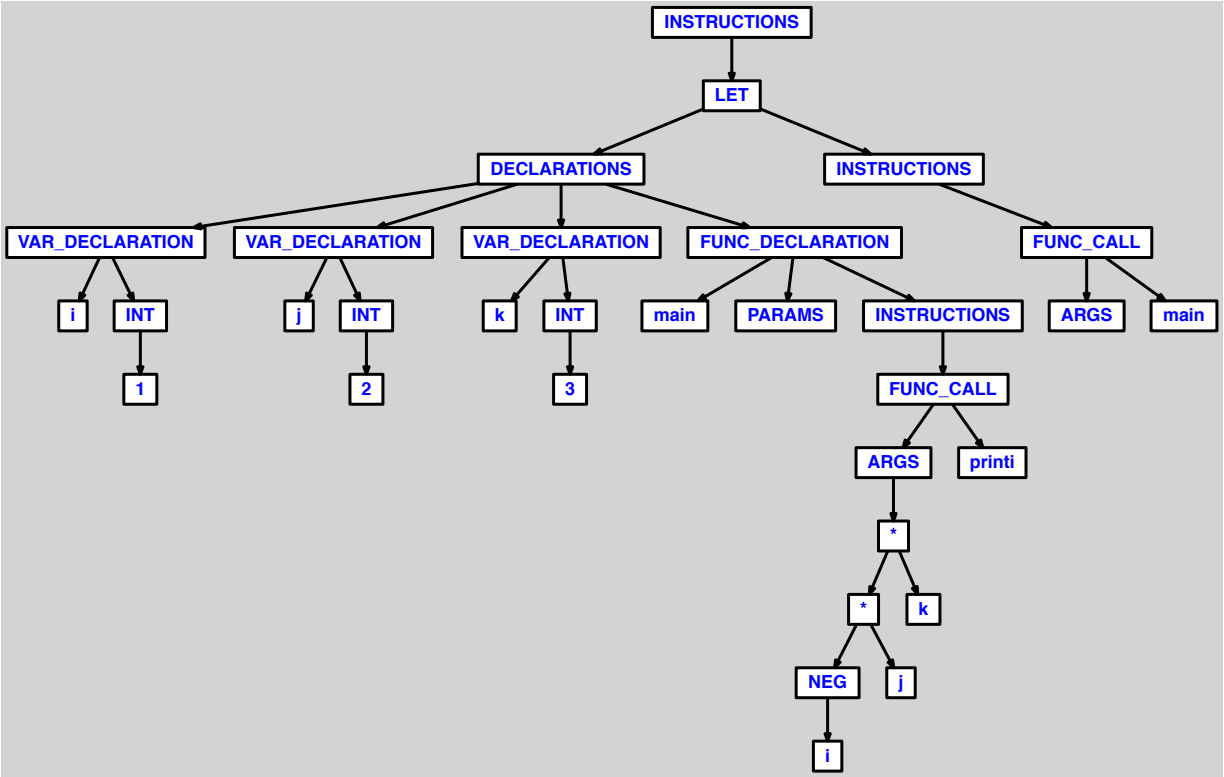
3.2.46 multiplication suivie de soustraction, avec termes identifies par variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
-
*
k
NEG
j
i

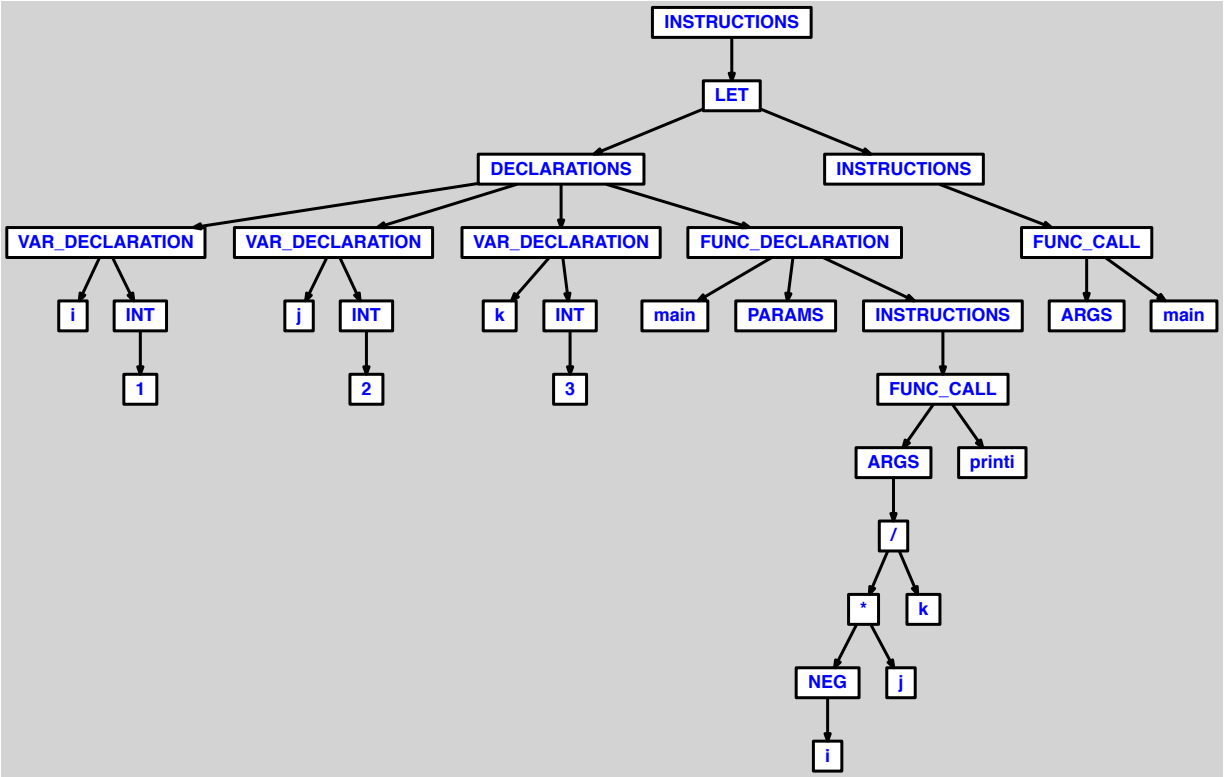
3.2.47 multiplication a 3 termes, identifies par des variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
*
k
NEG
j
i

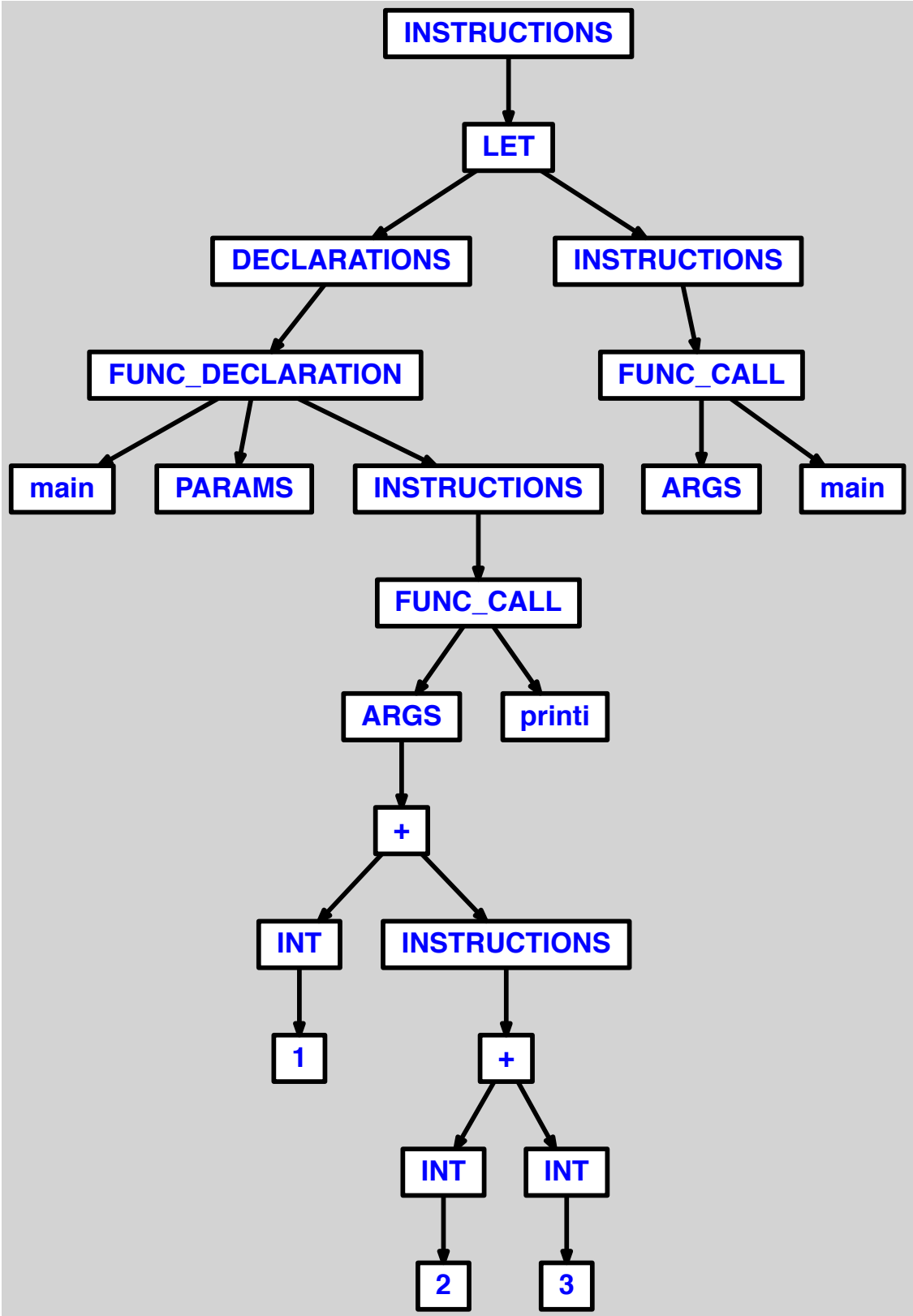
3.2.48 mutliplication suivie de division, avec termes identifies par variables, dont une ayant moins unaire

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
/
*
k
NEG
j
i

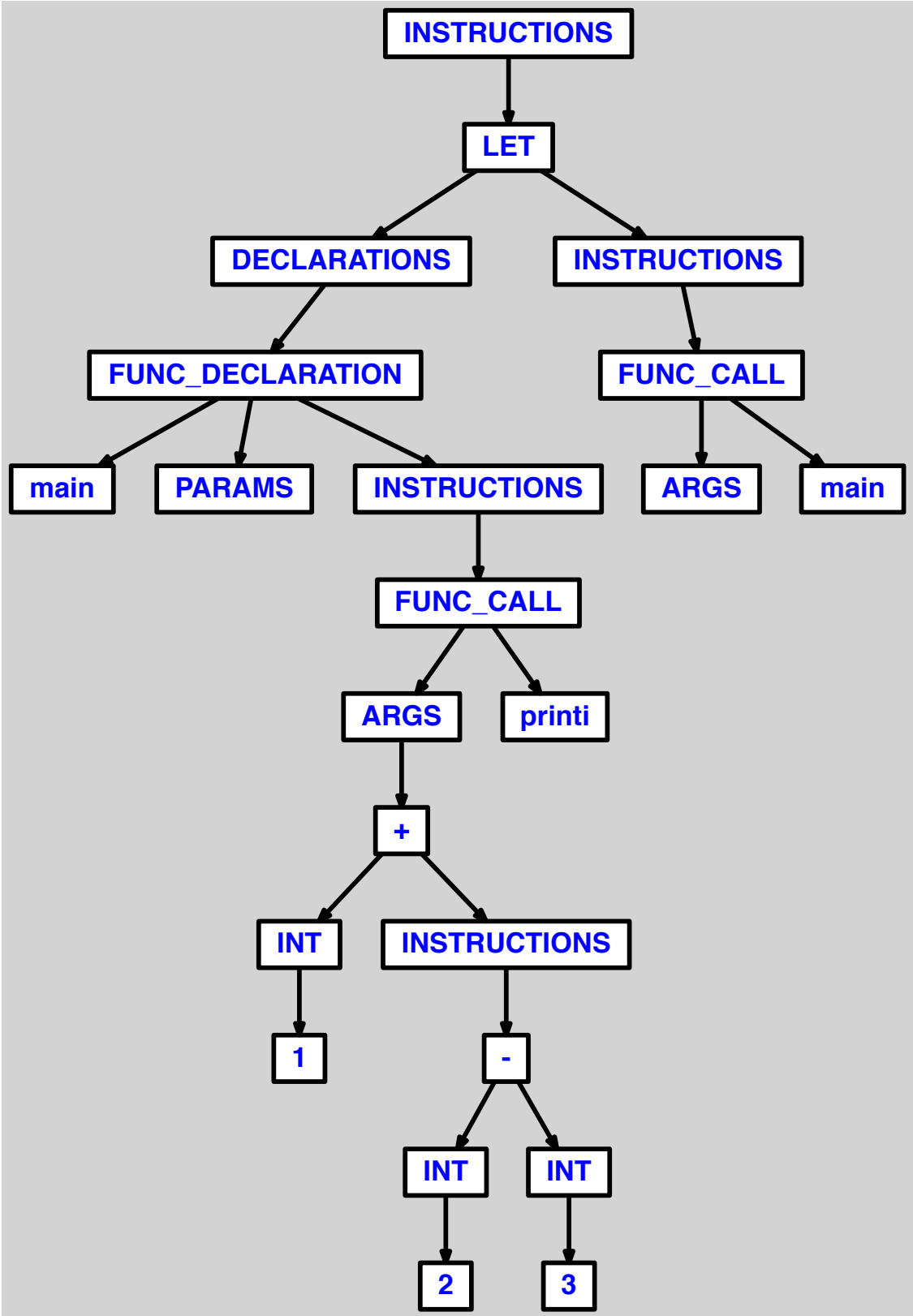
3.2.49 addition a 3 termes, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INSTRUCTIONS
1
+
INT
INT
2
3

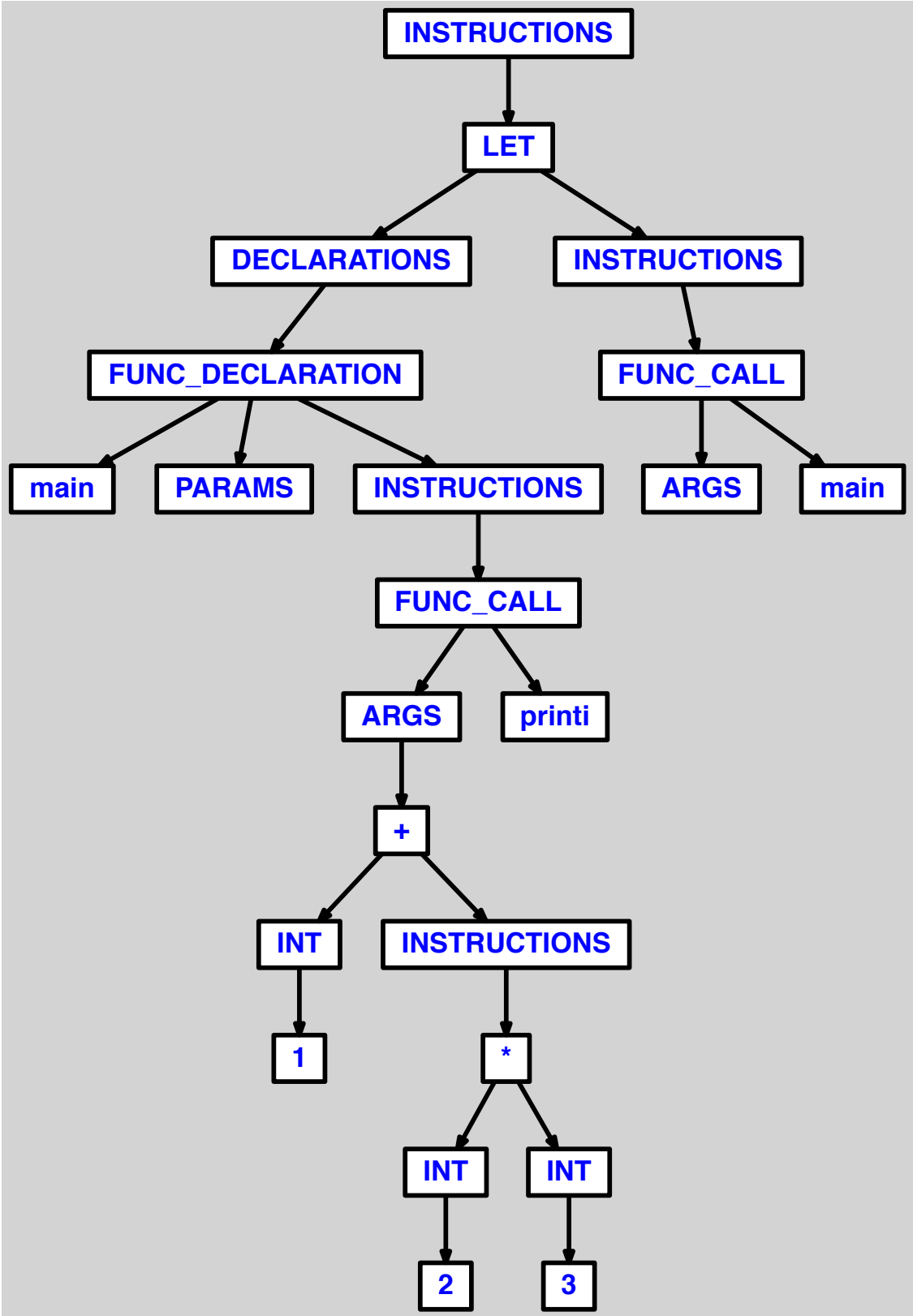
3.2.50 addition suivie de soustraction, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INSTRUCTIONS
1
-
INT
INT
2
3

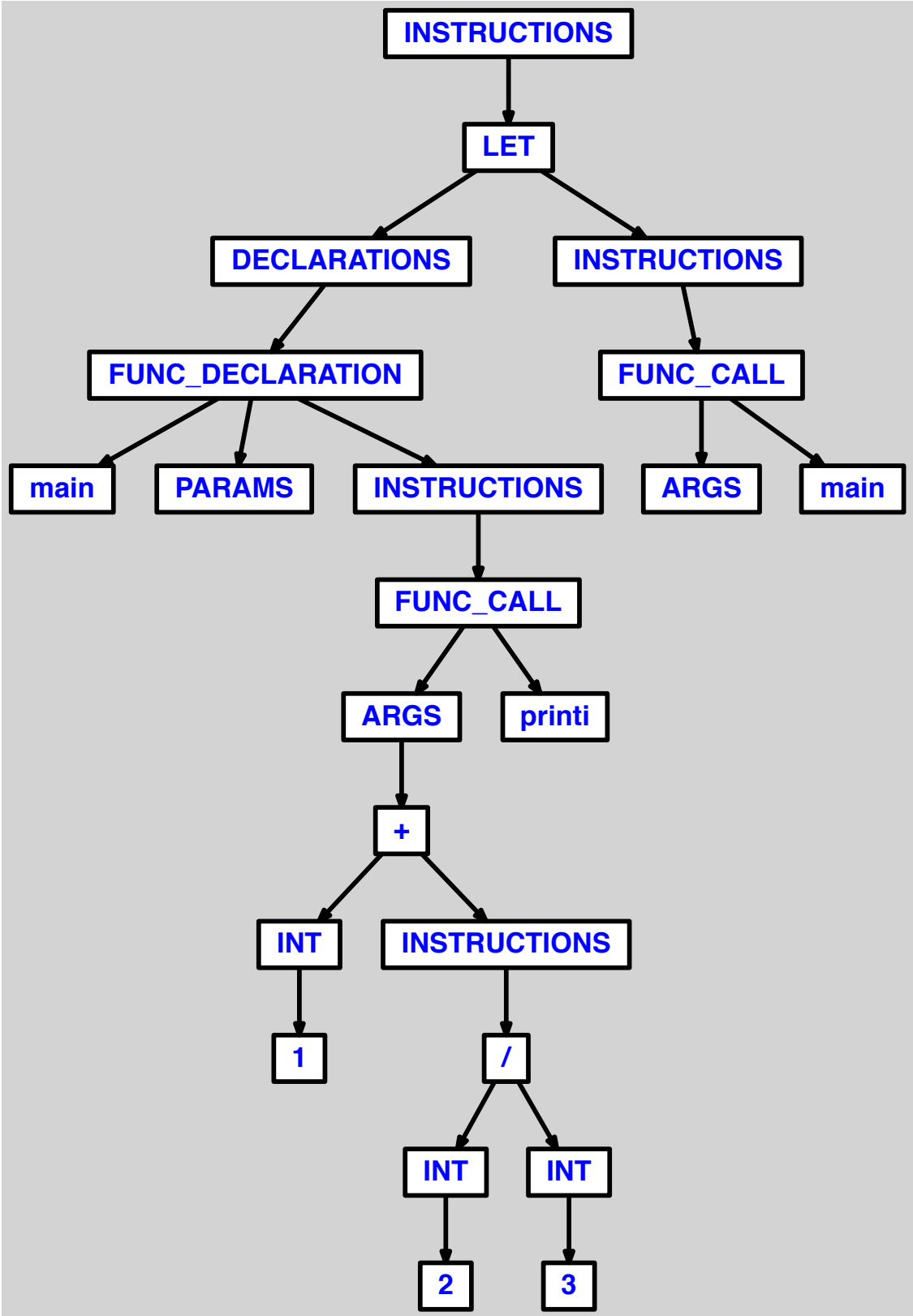
3.2.51 addition suivie de multiplication, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INSTRUCTIONS
1
*
INT
INT
2
3

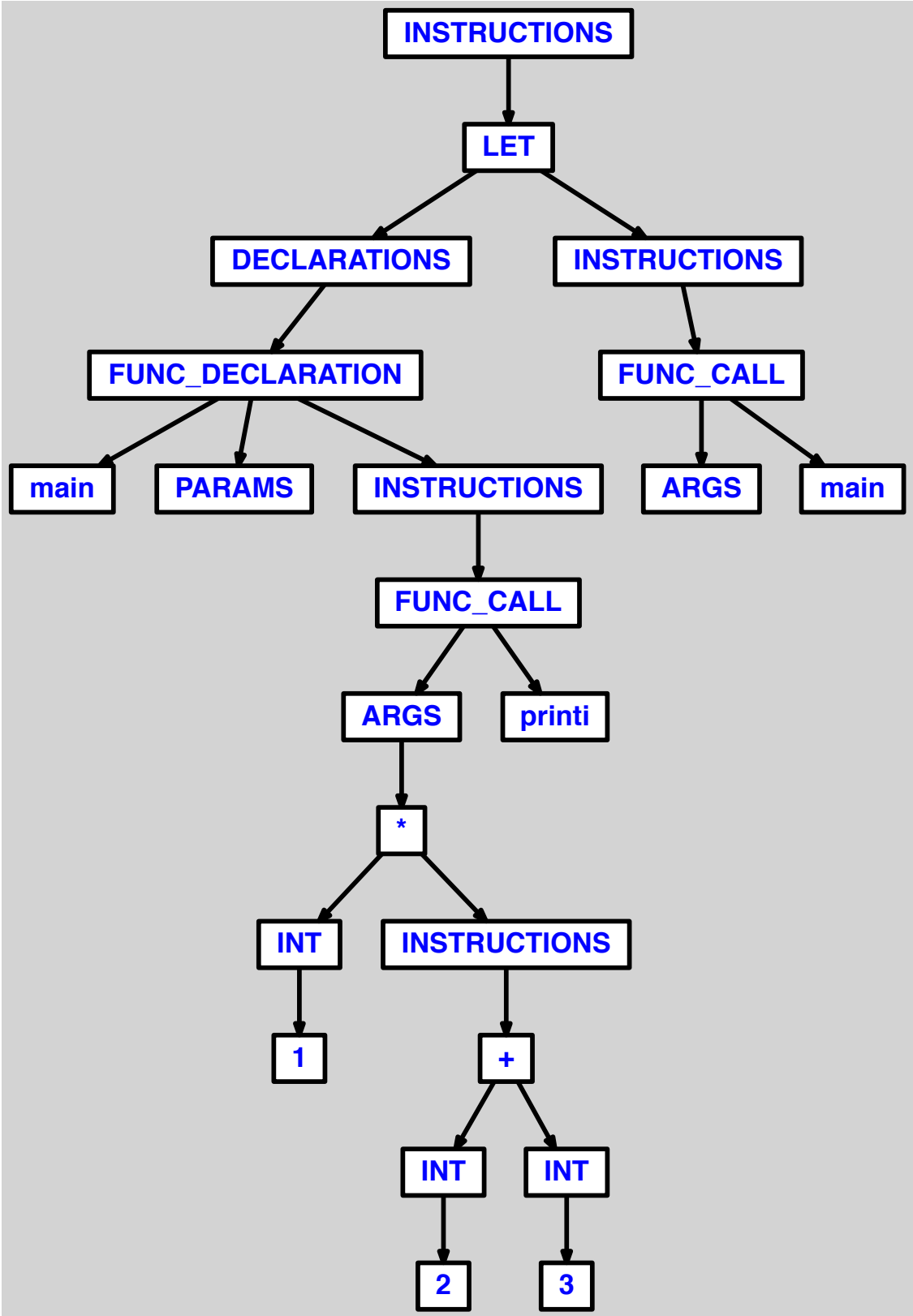
3.2.52 addition suivie de division, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INT
INSTRUCTIONS
1
/
INT
INT
2
3

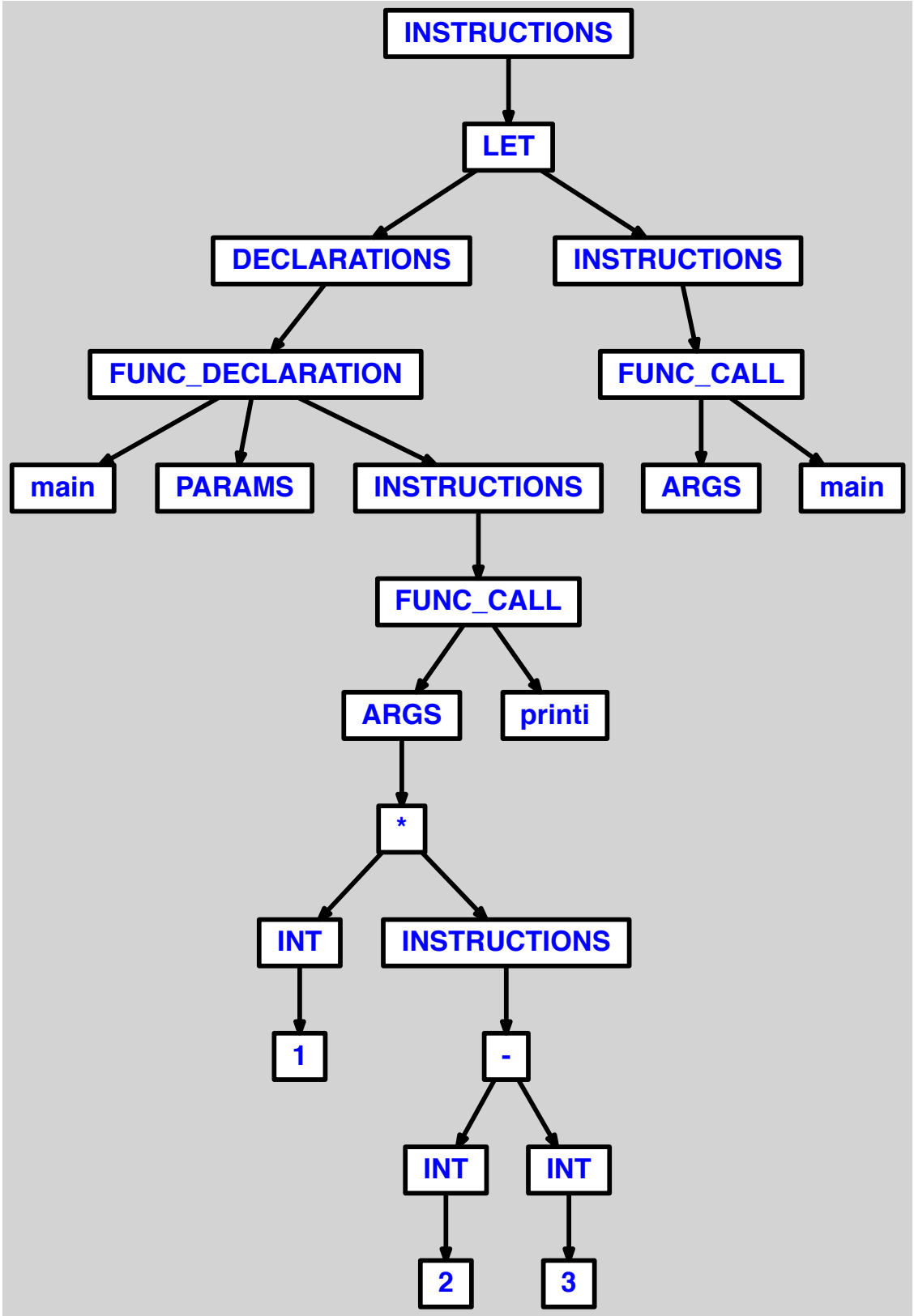
3.2.53 multiplication suivie d'addition, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INT
INSTRUCTIONS
1
+
INT
INT
2
3

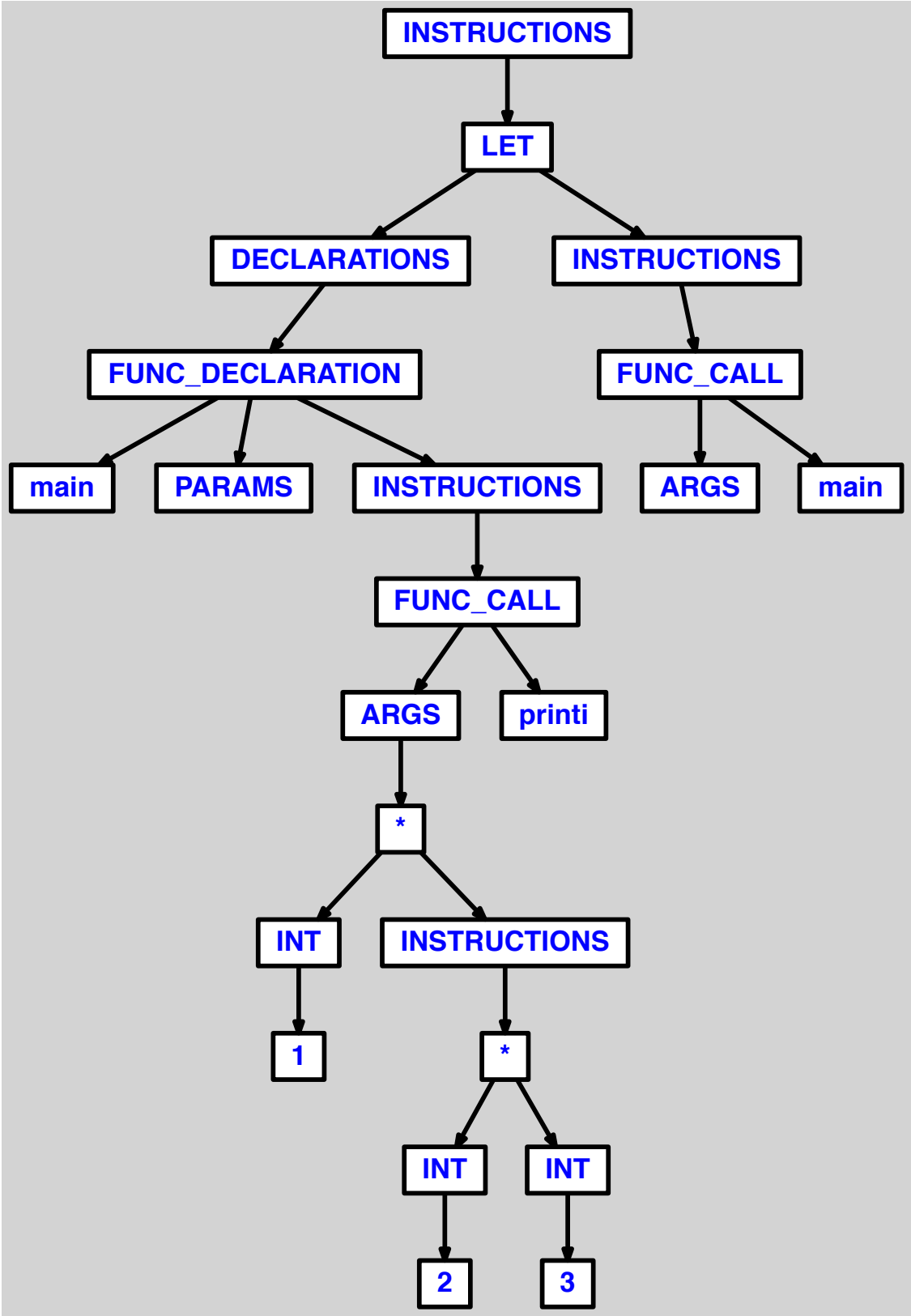
3.2.54 multiplication suivie de soustraction, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INT
INSTRUCTIONS
1
-
INT
INT
2
3

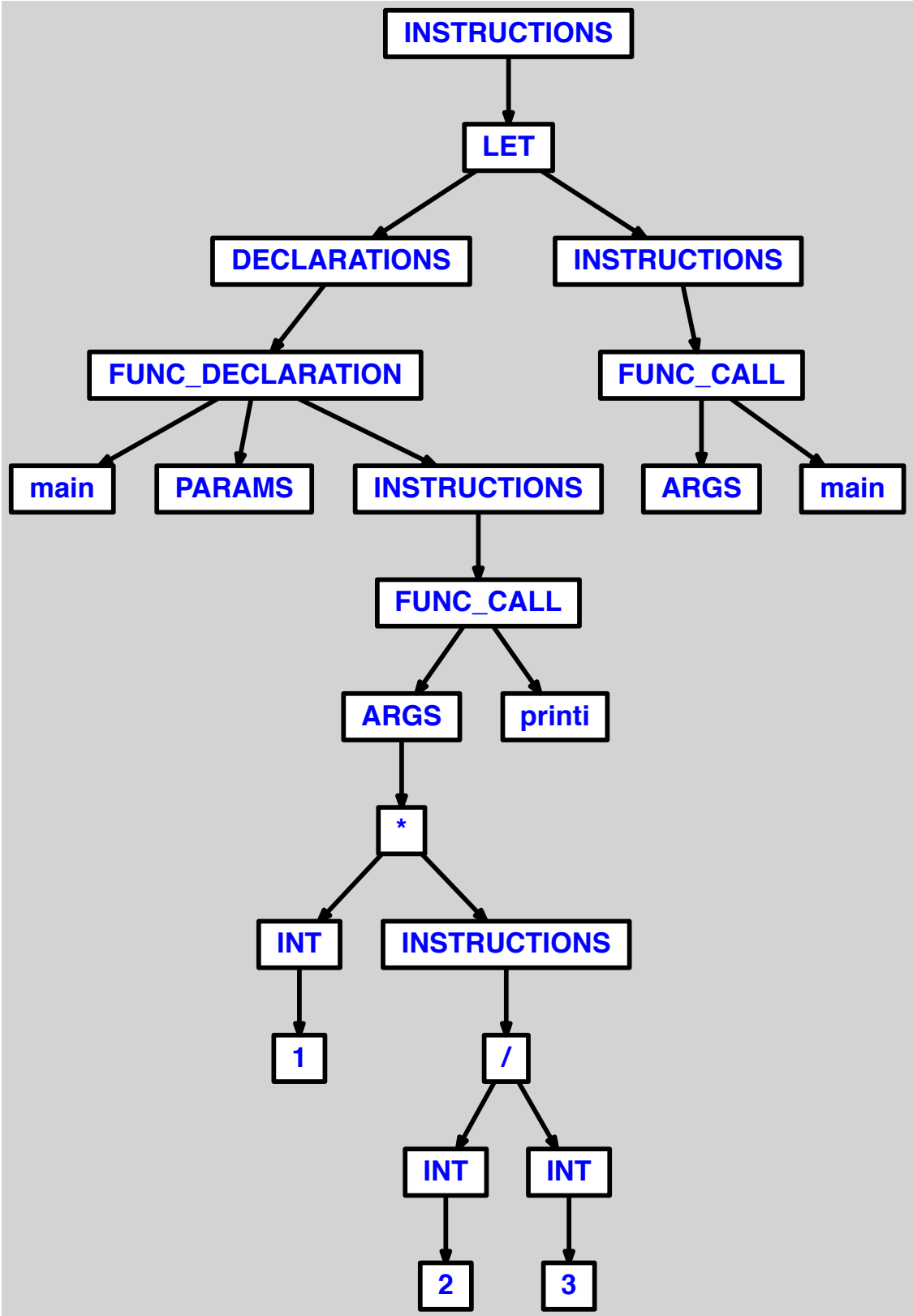
3.2.55 multiplication a 3 termes, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INT
INSTRUCTIONS
1
*
INT
INT
2
3

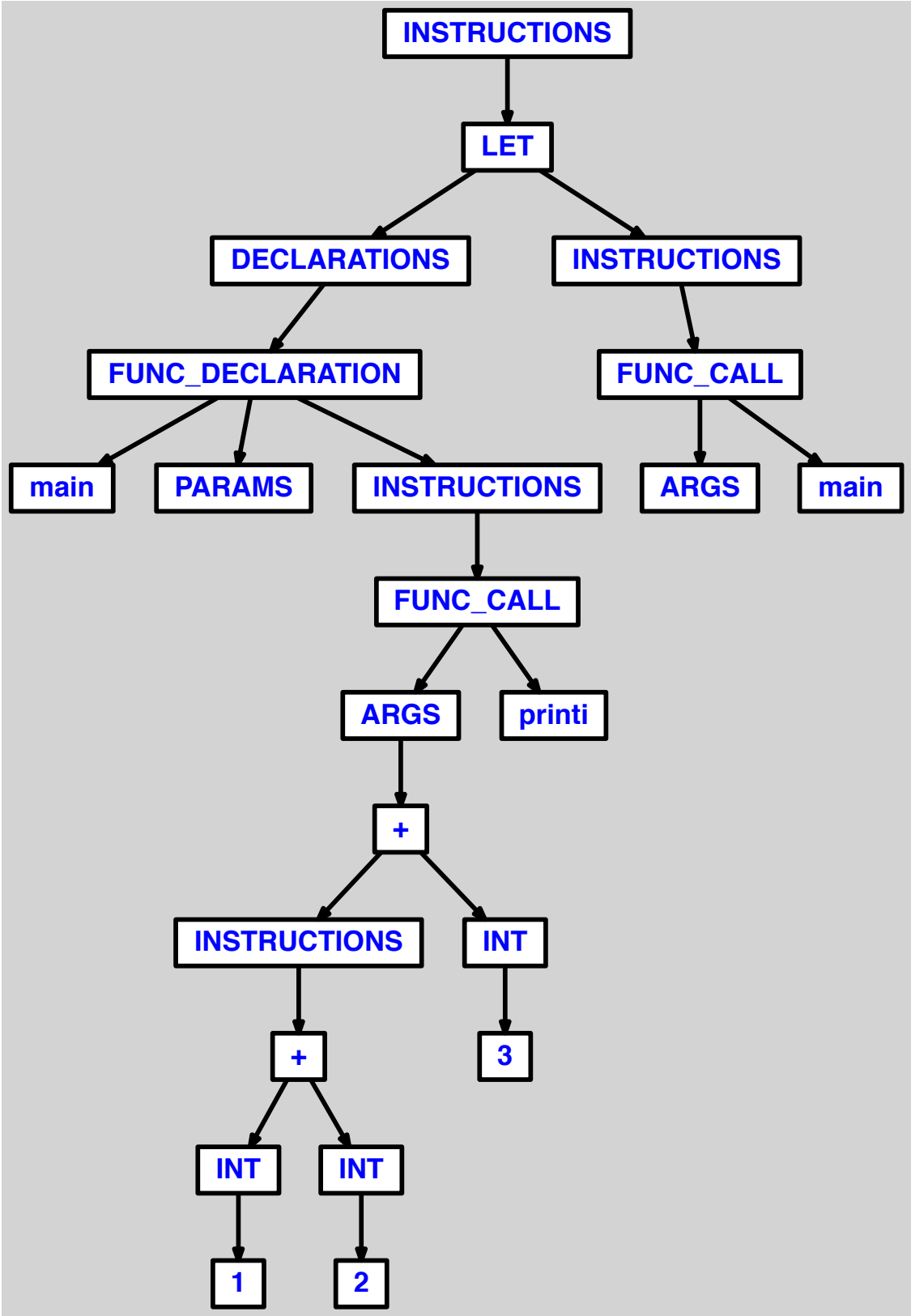
3.2.56 multiplication suivie de vision, avec parenthesage des 2 termes a droite

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INT
INSTRUCTIONS
1
/
INT
INT
2
3

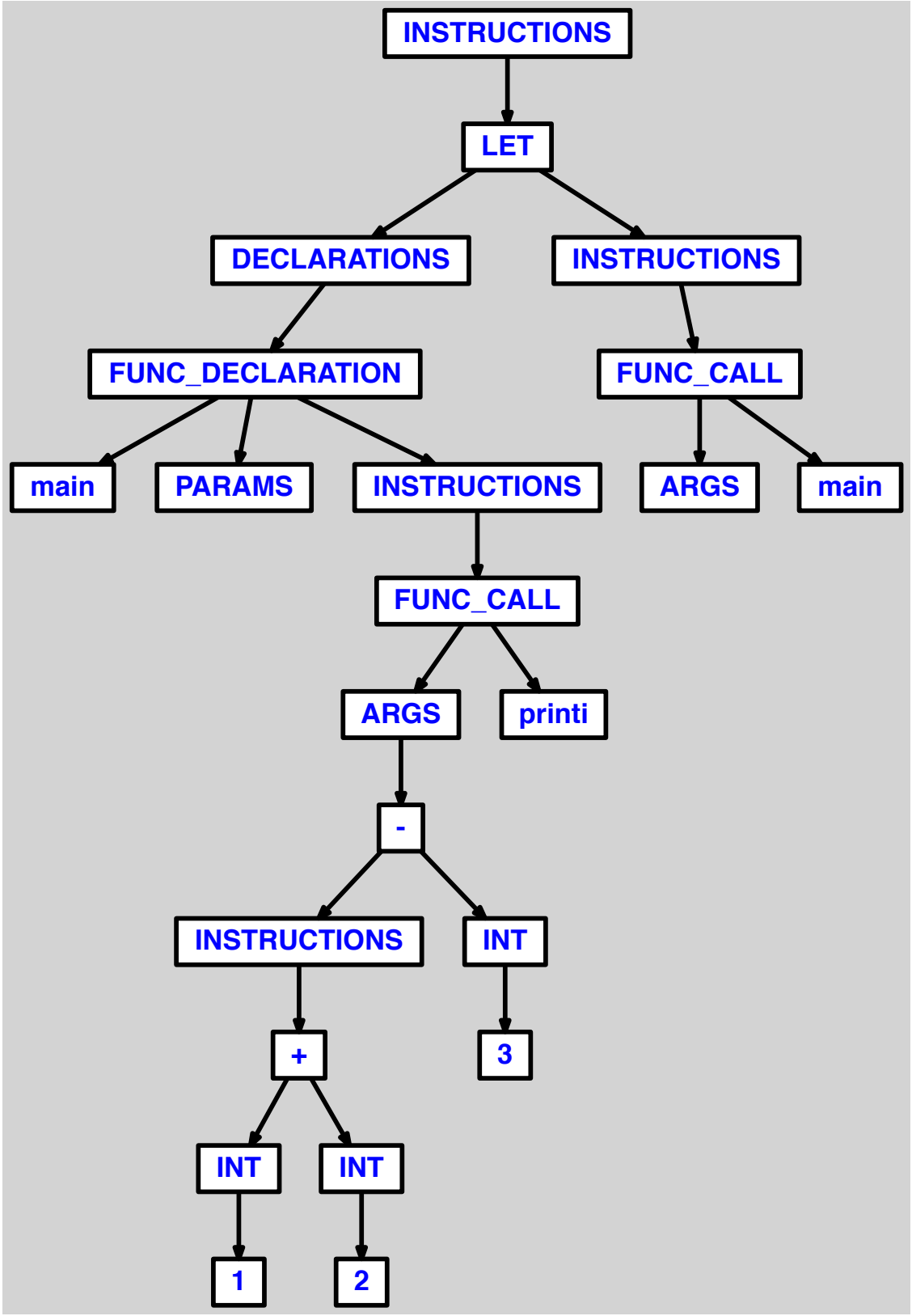
3.2.57 addition a 3 termes, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INSTRUCTIONS
INT
+
3
INT
INT
1
2

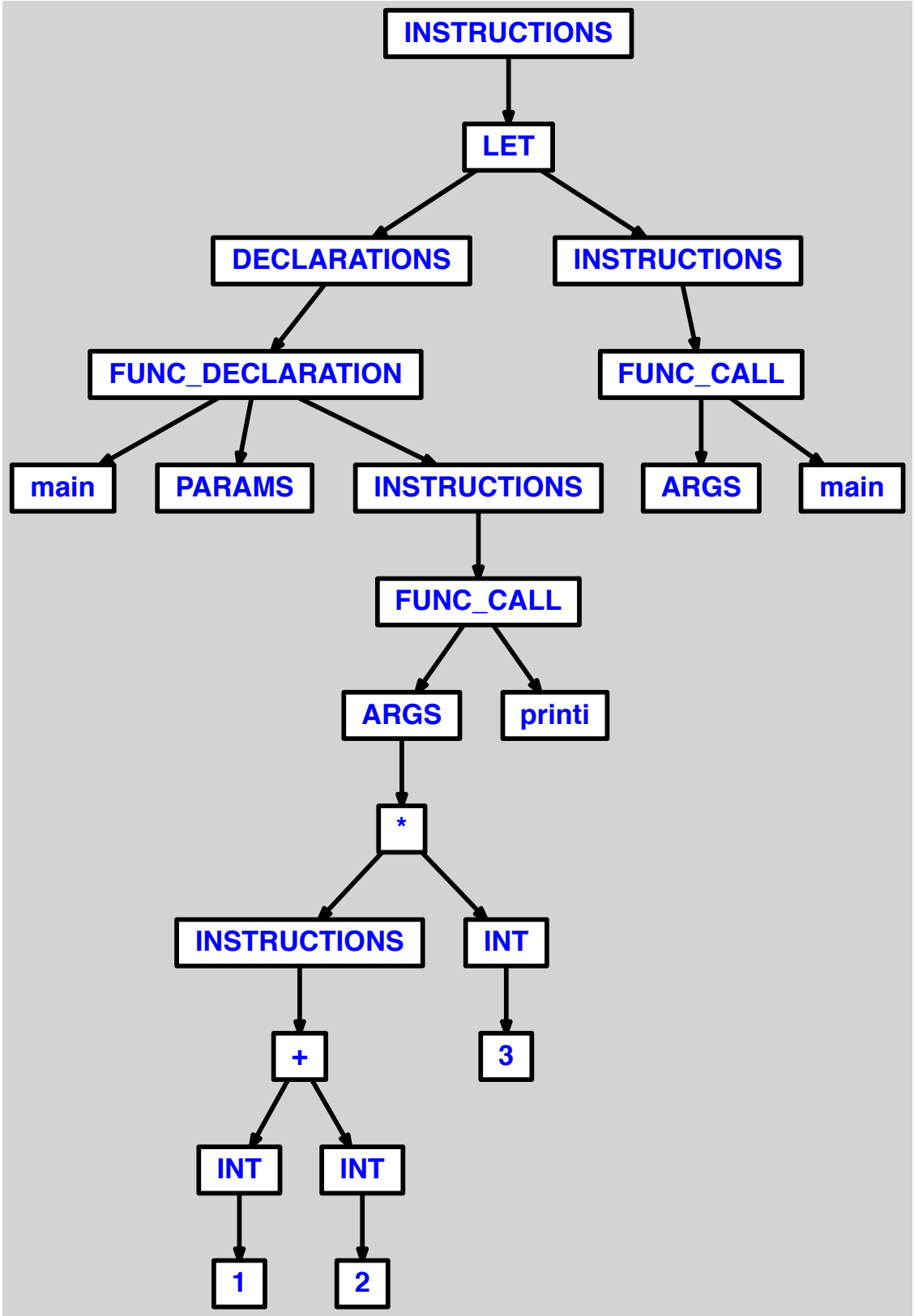
3.2.58 addition suivie de soustraction, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
INSTRUCTIONS
INT
+
3
INT
INT
1
2

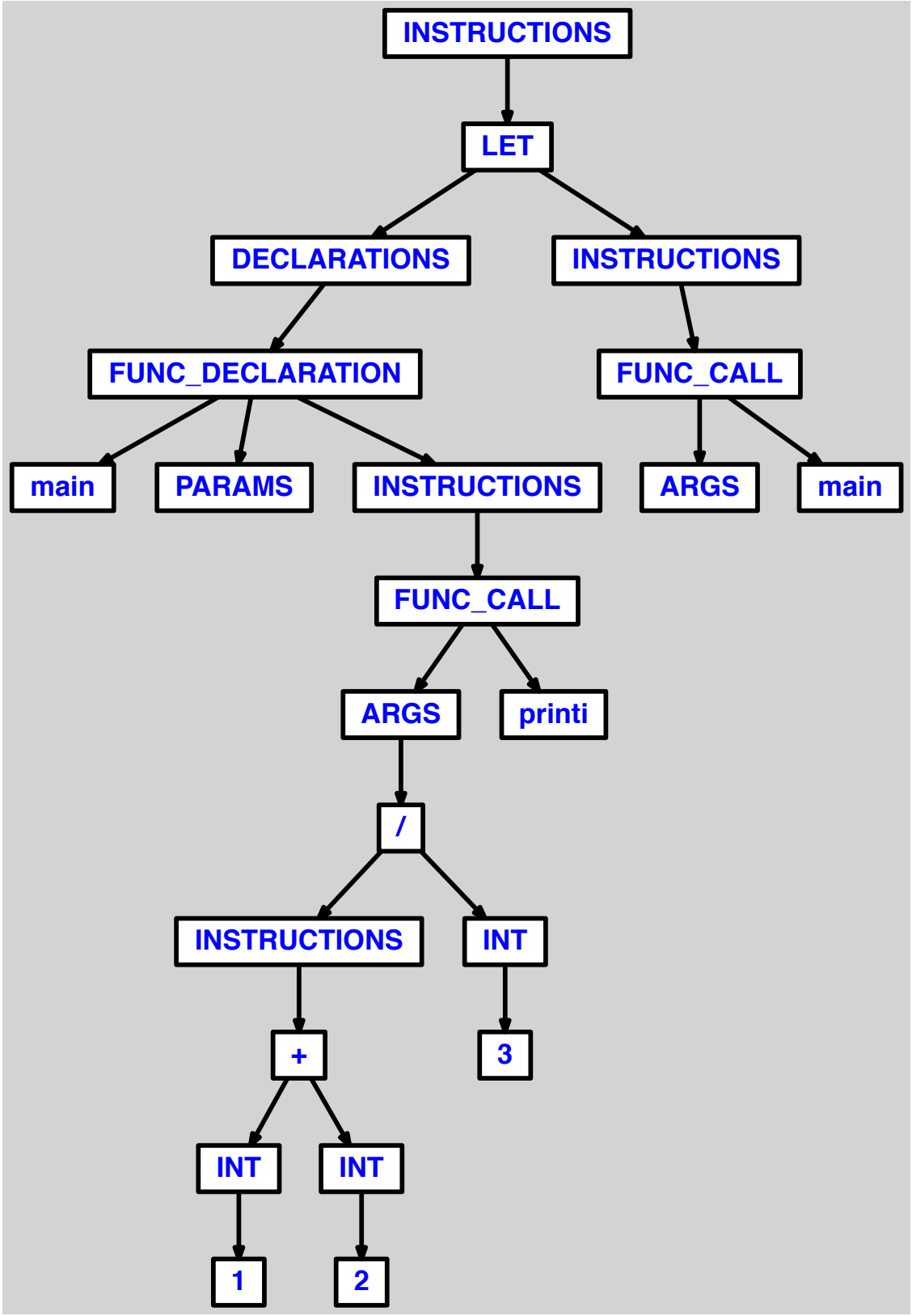
3.2.59 addition suivie de multiplication, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INSTRUCTIONS
INT
+
3
INT
INT
1
2

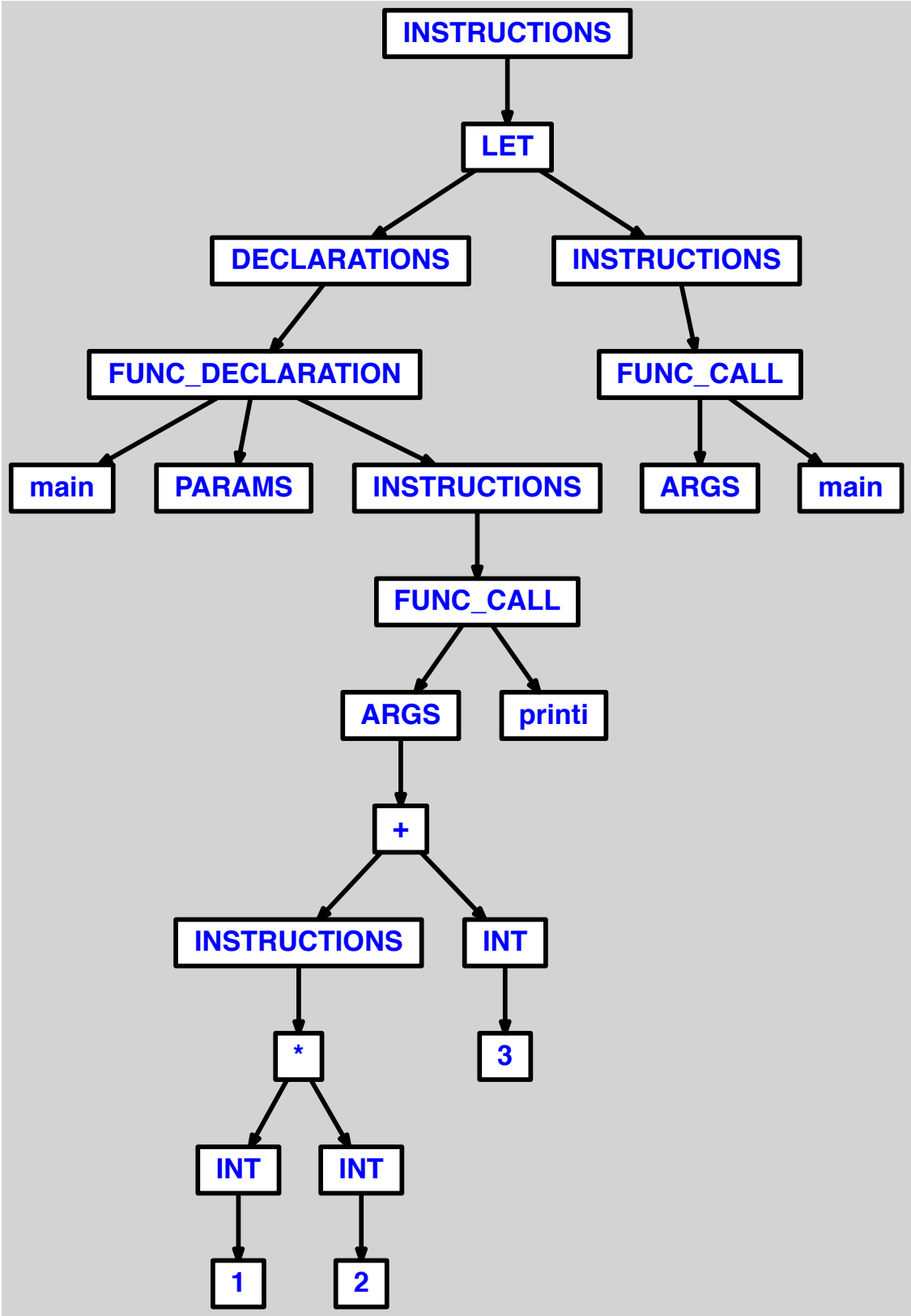
3.2.60 addition suivie de division, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
INT
+
3
INT
INT
1
2

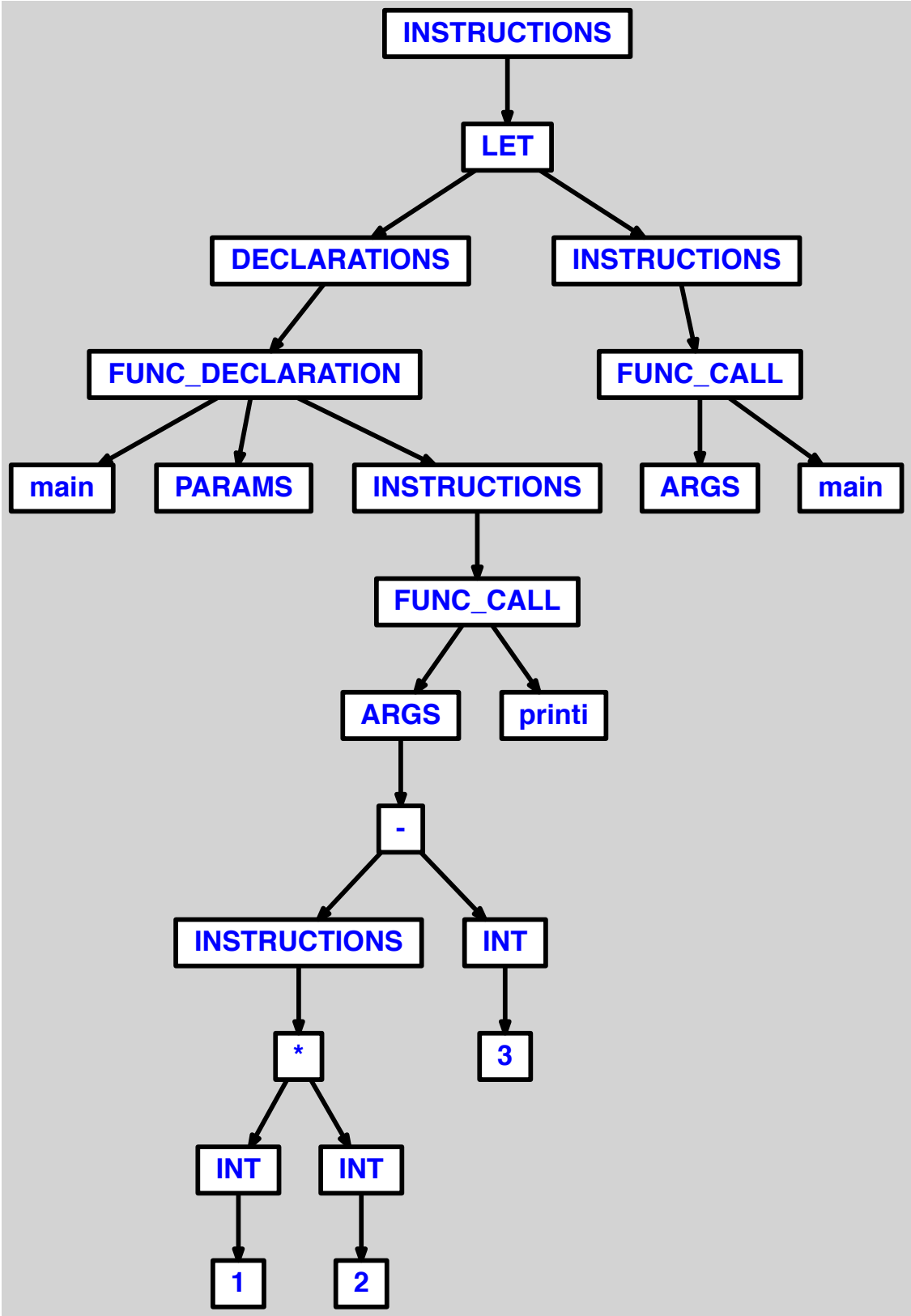
3.2.61 multiplication suivie d'addition, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INSTRUCTIONS
INT
*
3
INT
INT
1
2

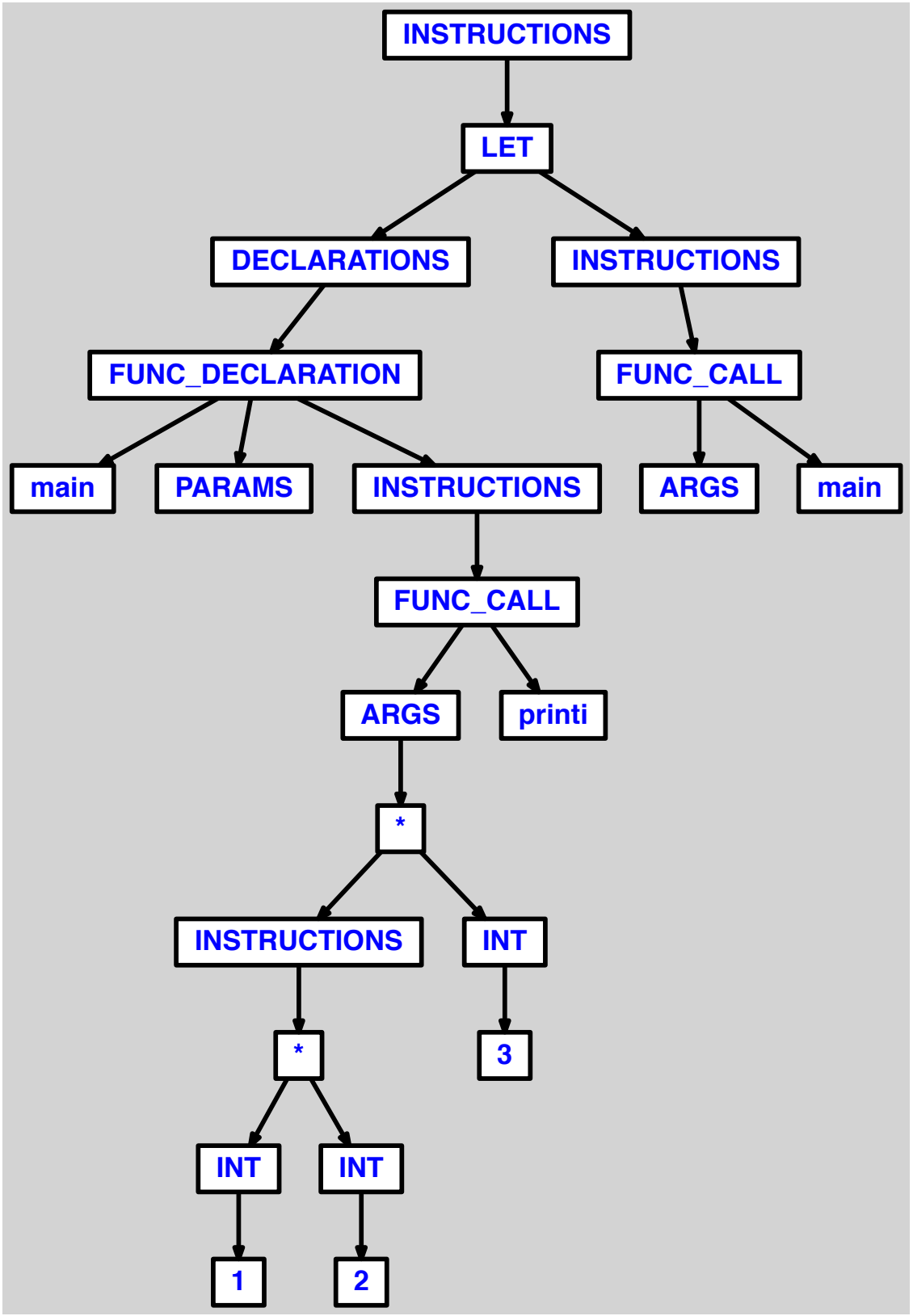
3.2.62 multiplication suivie de soustraction, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
INSTRUCTIONS
INT
*
3
INT
INT
1
2

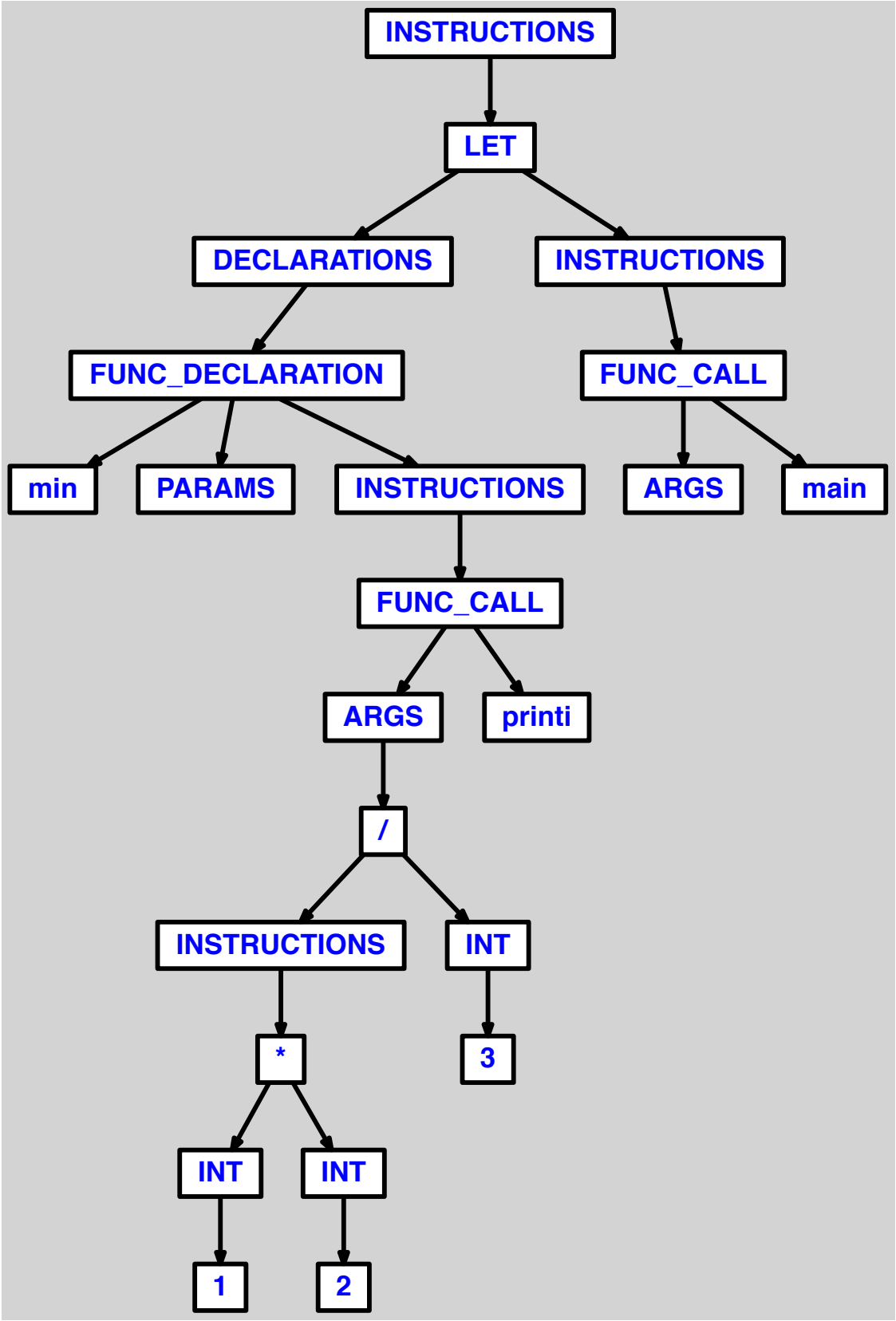
3.2.63 multiplication a 3 termes, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INSTRUCTIONS
INT
*
3
INT
INT
1
2

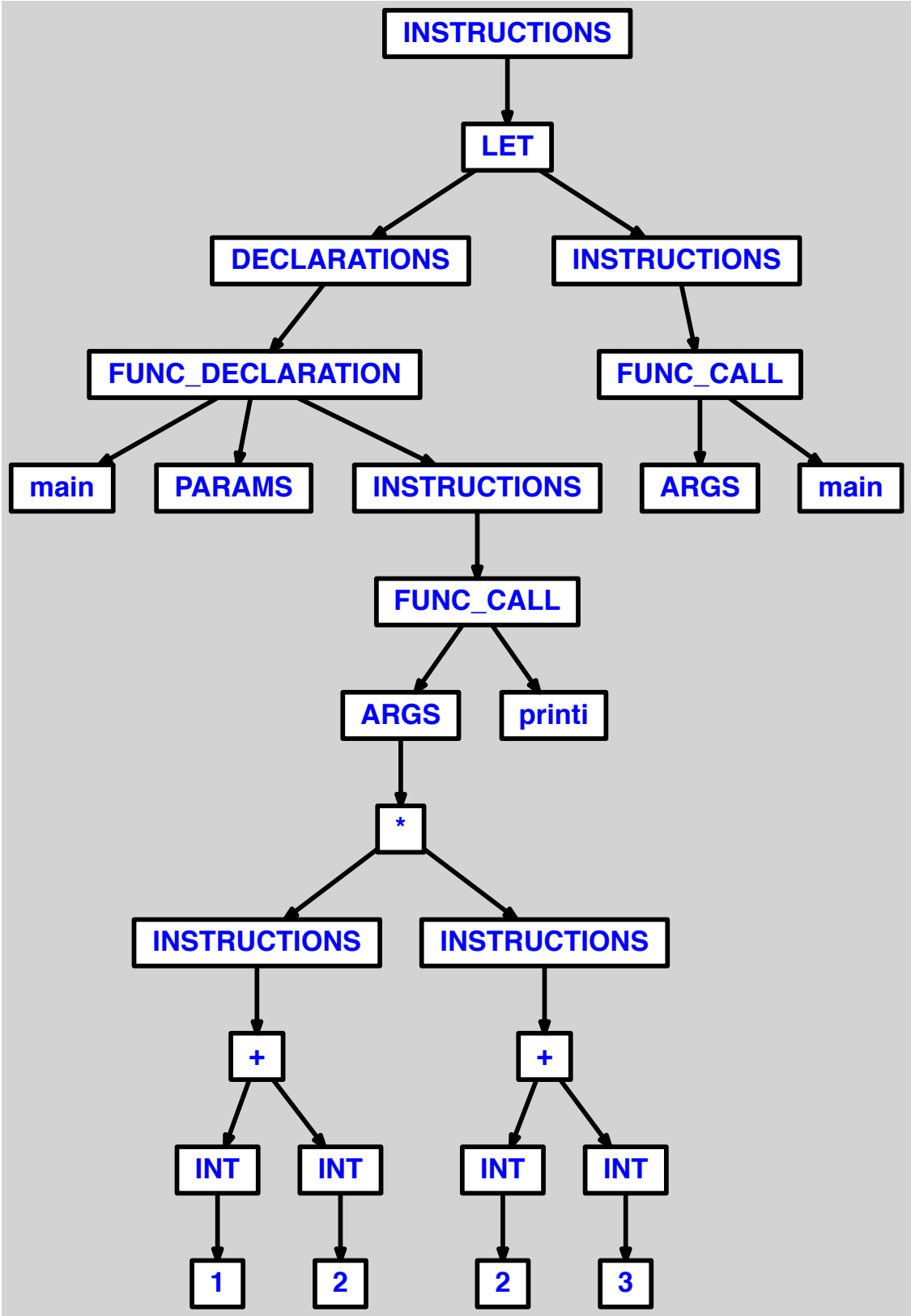
3.2.64 multiplication suivie de division, avec parenthesage des 2 termes a gauche

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
min
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
INT
*
3
INT
INT
1
2

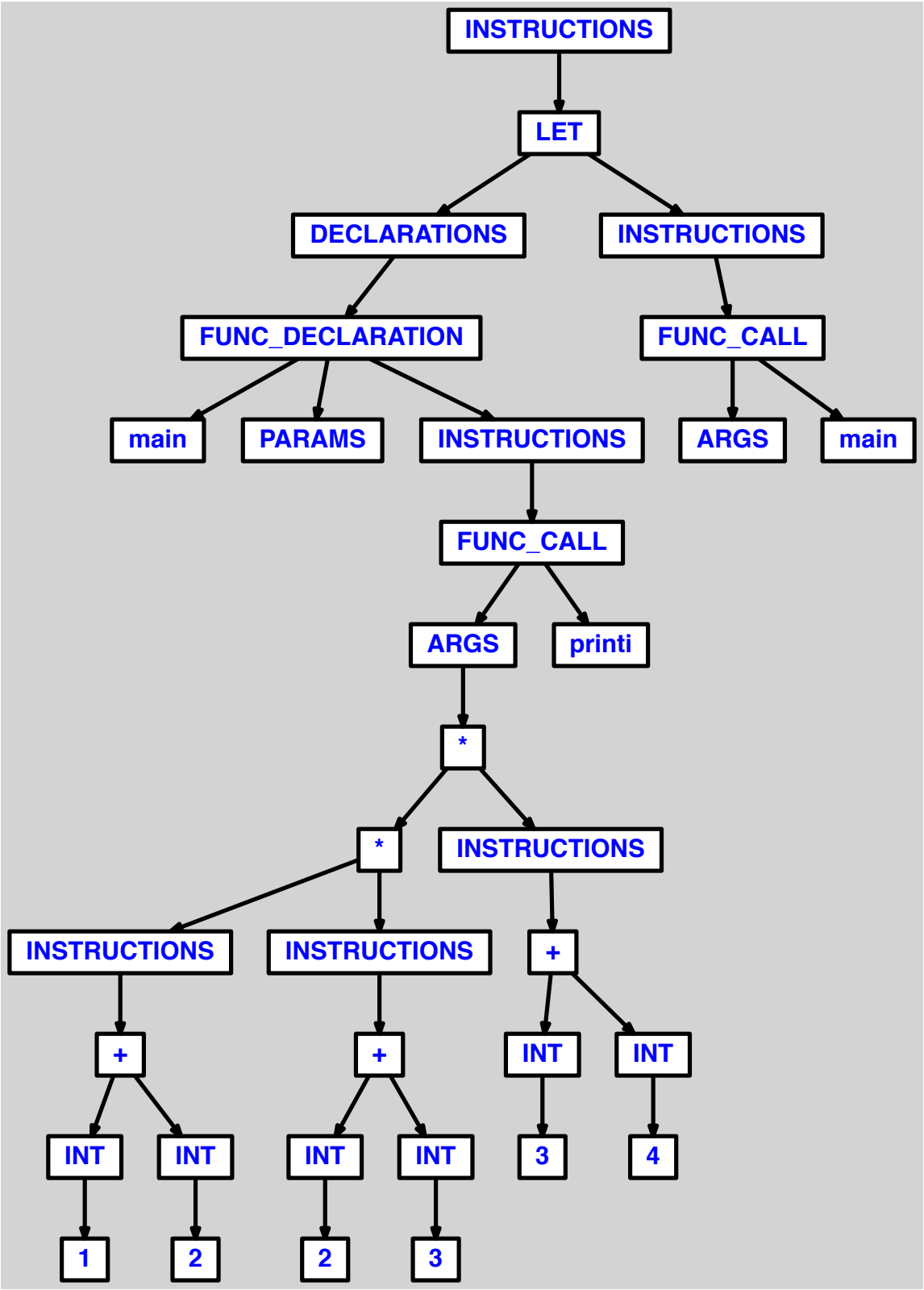
3.2.65 multiplication de 2 additions parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
INSTRUCTIONS
INSTRUCTIONS
+
+
INT
INT
INT
INT
1
2
2
3

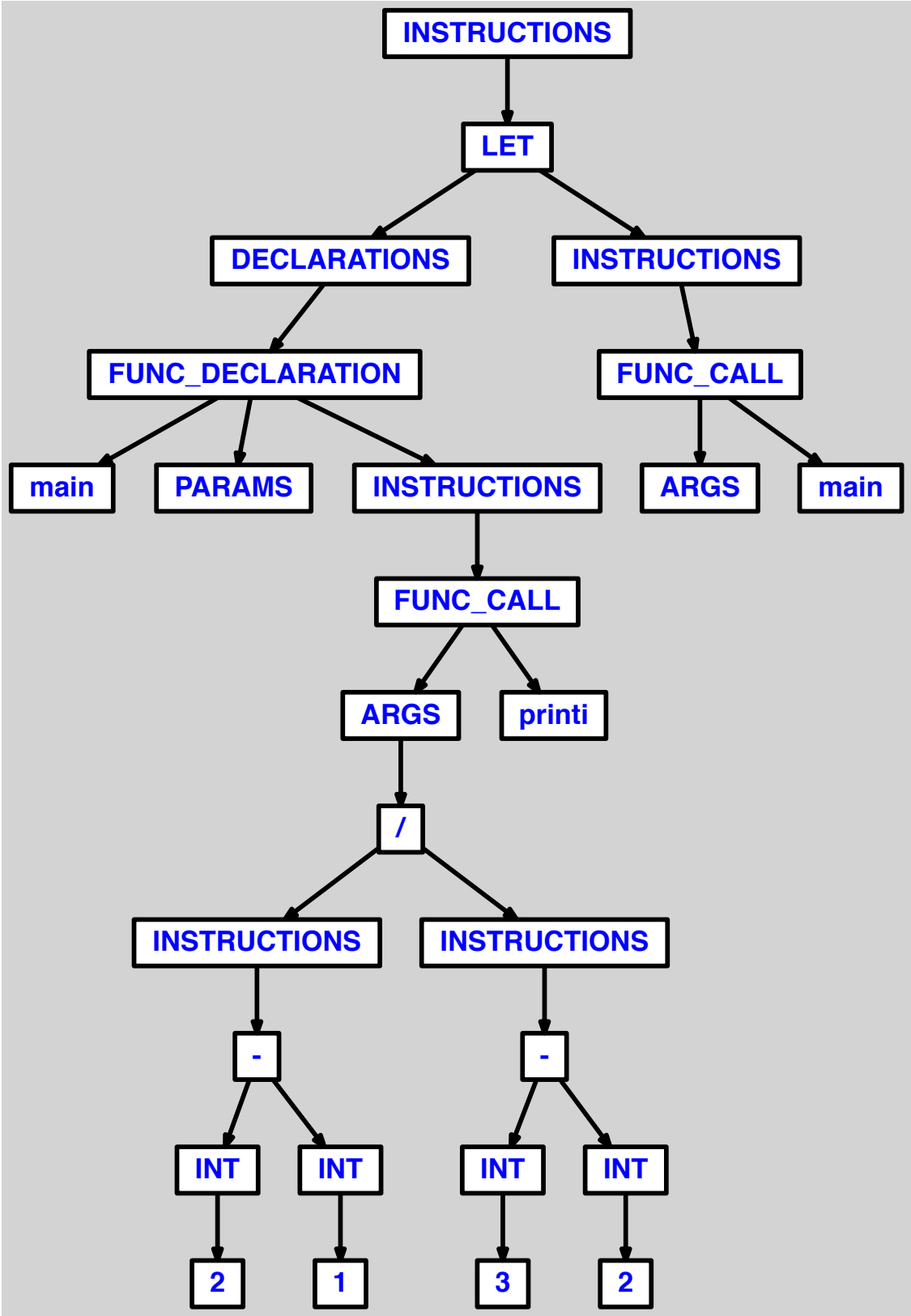
3.2.66 multiplication de 3 additions parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
*
*
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
+
+
+
INT
INT
INT
INT
INT
INT
3
4
1
2
2
3

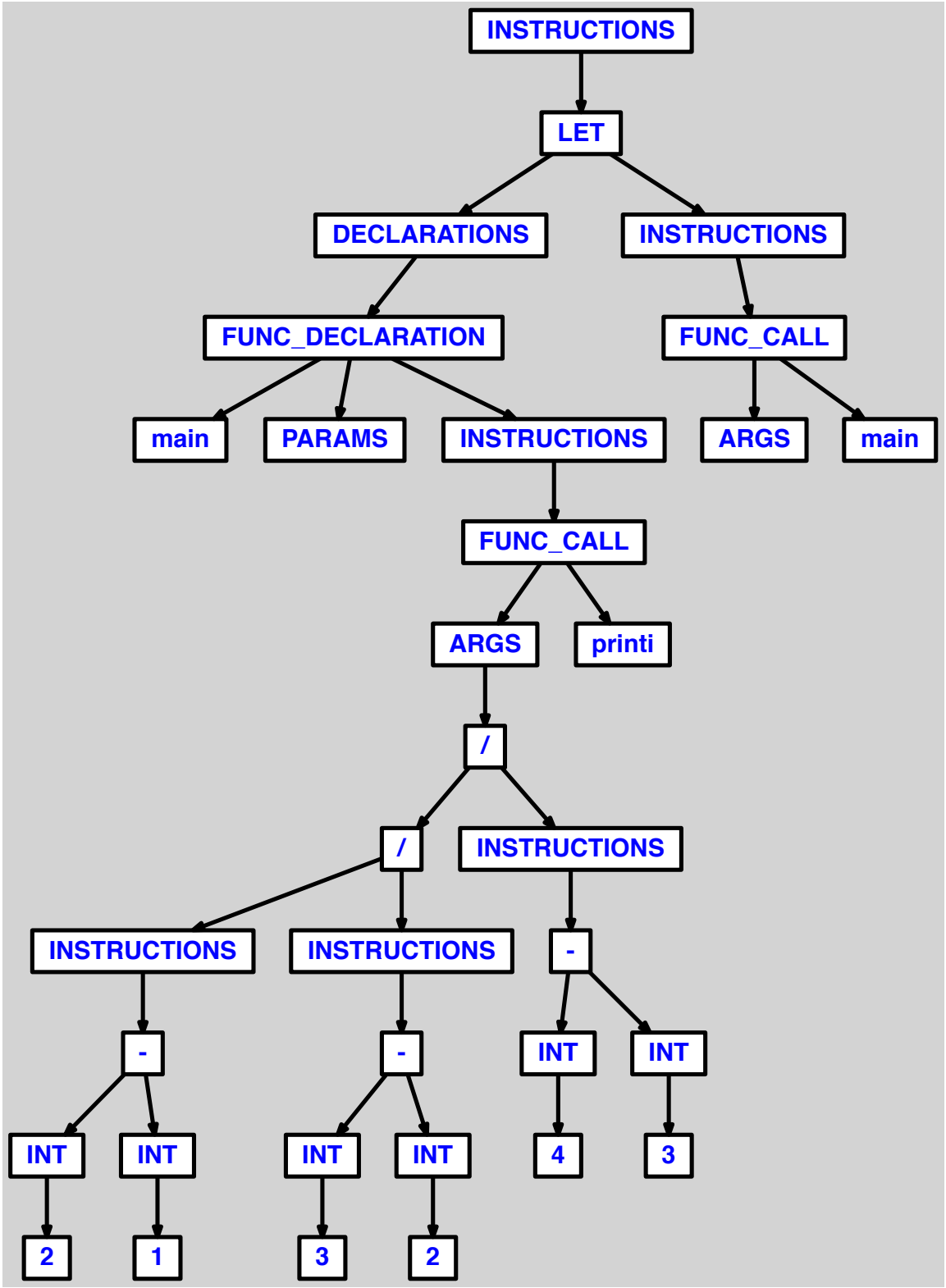
3.2.67 division de 2 soustractions parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
INSTRUCTIONS
-
-
INT
INT
INT
INT
2
1
3
2

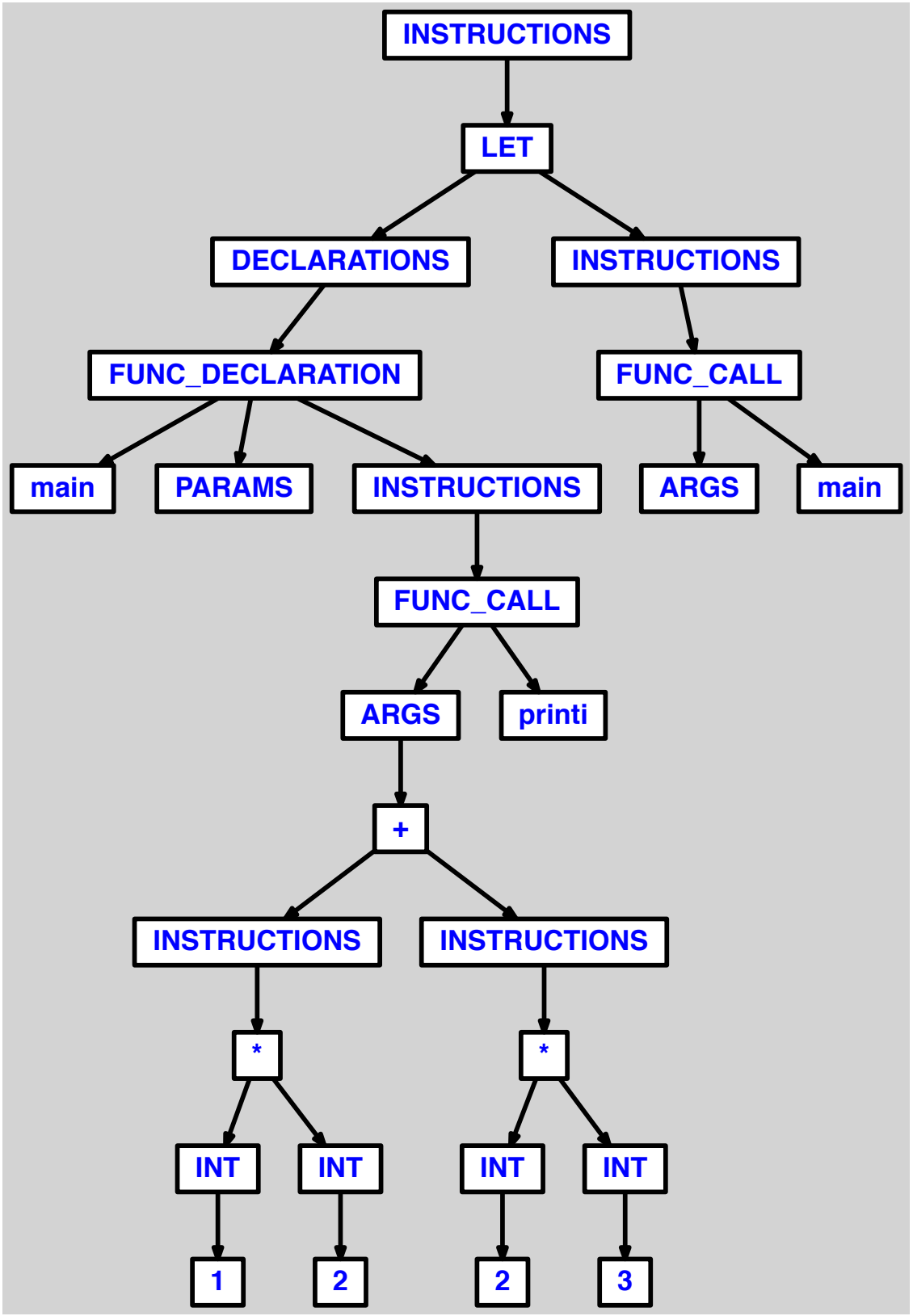
3.2.68 division de 3 soustractions parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
/
/
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
-
-
-
INT
INT
INT
INT
INT
INT
4
3
2
1
3
2

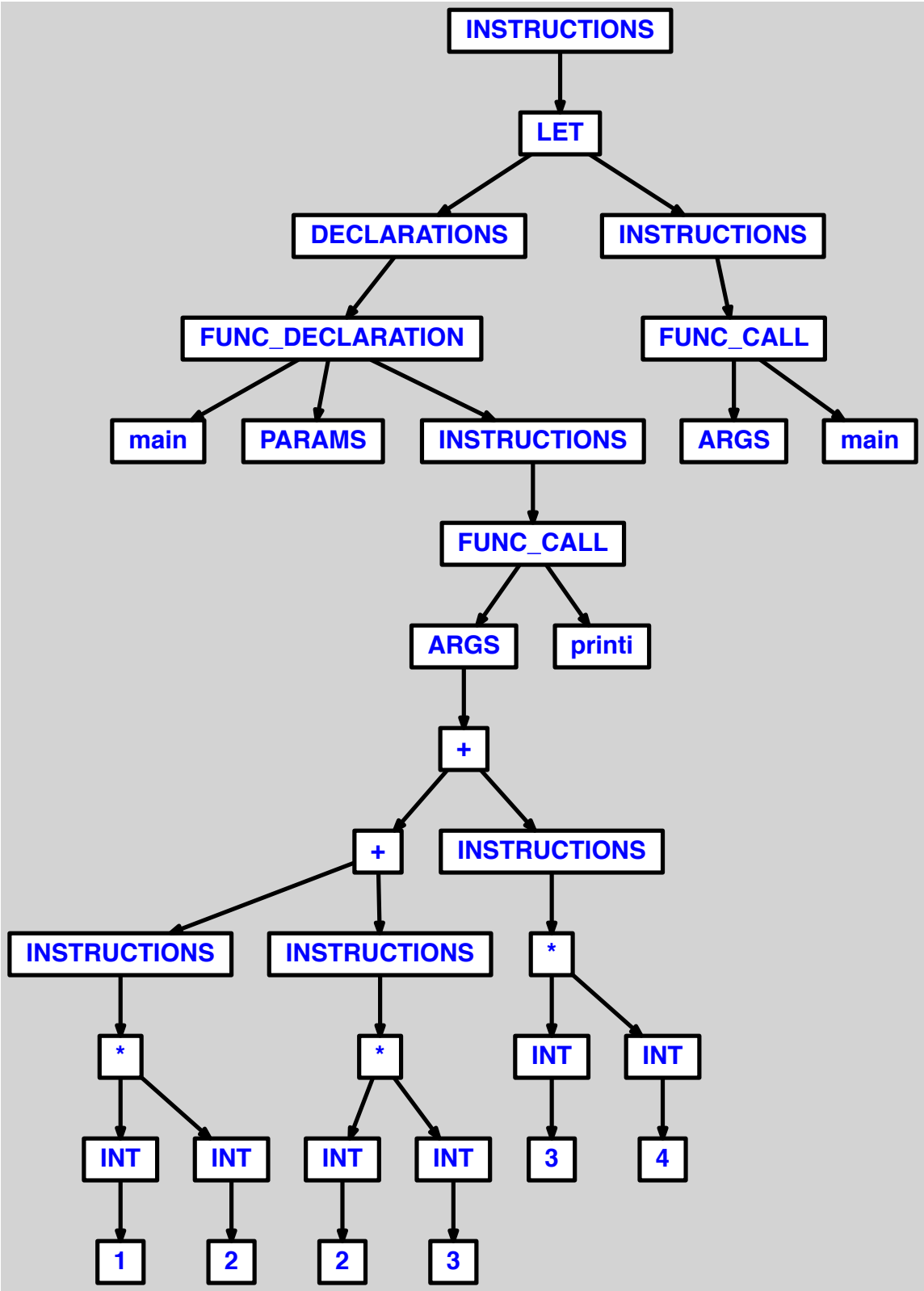
3.2.69 addition de 2 multiplications parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
INSTRUCTIONS
INSTRUCTIONS
*
*
INT
INT
INT
INT
1
2
2
3

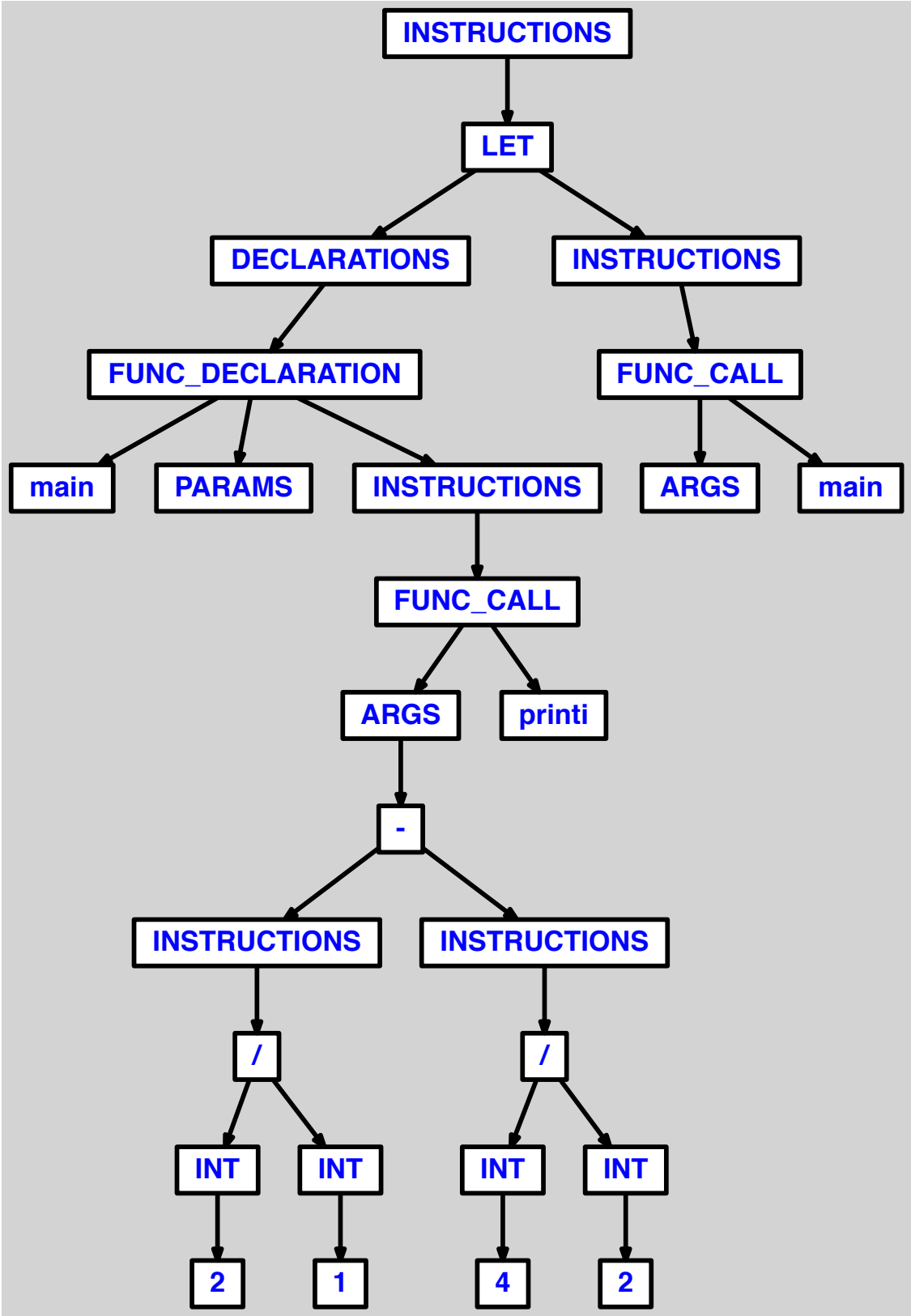
3.2.70 addition de 3 multiplications parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
+
+
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
*
*
*
INT
INT
INT
INT
INT
INT
3
4
1
2
2
3

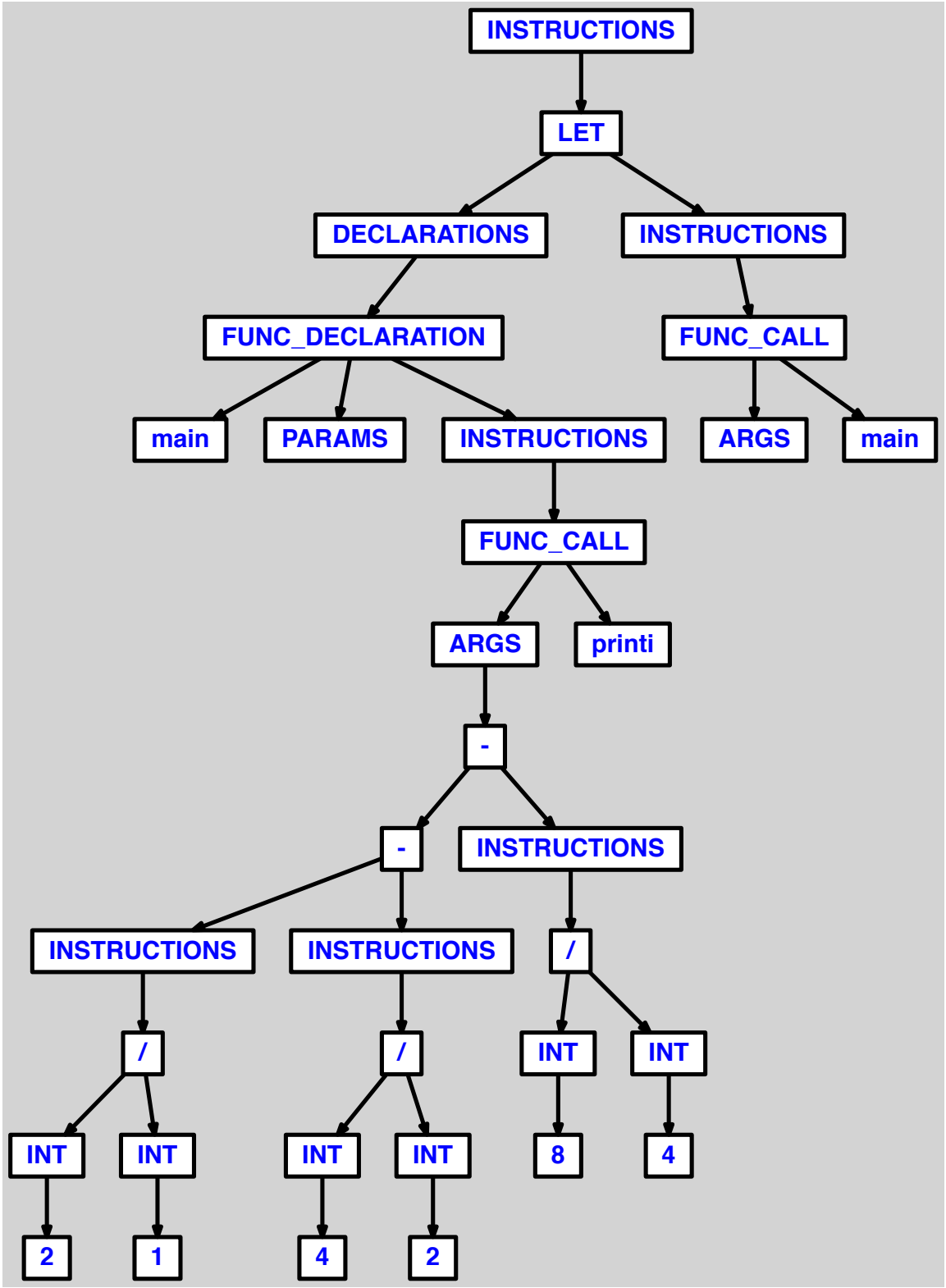
3.2.71 soustraction de 2 divisions parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
INSTRUCTIONS
INSTRUCTIONS
/
/
INT
INT
INT
INT
2
1
4
2

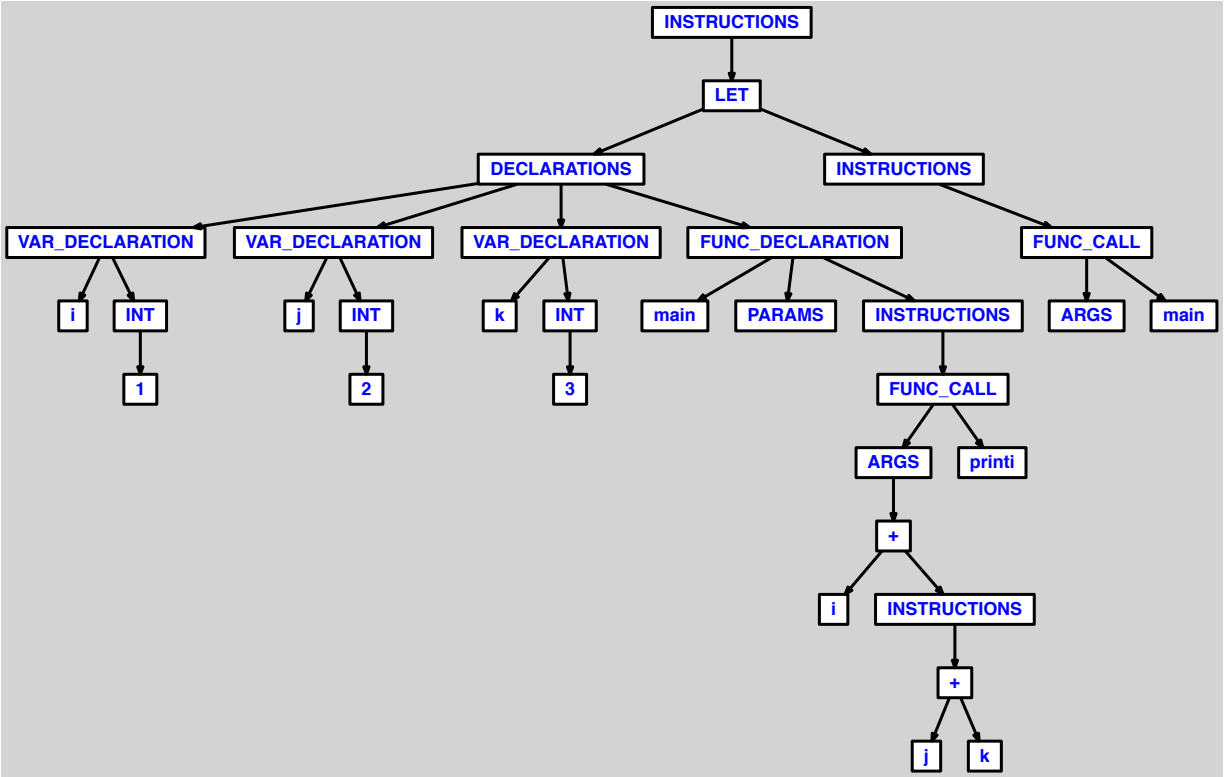
3.2.72 soustraction de 3 divisions parenthesees

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
-
-
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
/
/
/
INT
INT
INT
INT
INT
INT
8
4
2
1
4
2

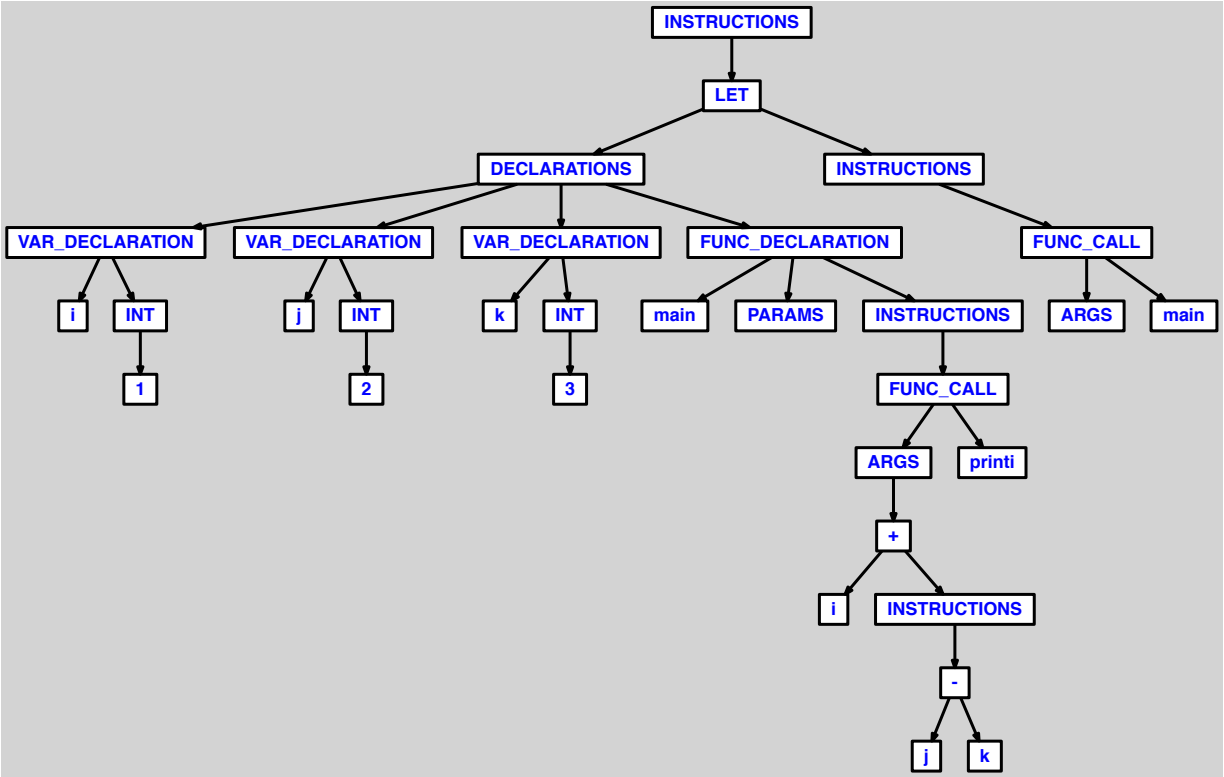
3.2.73 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
i
INSTRUCTIONS
+
j
k

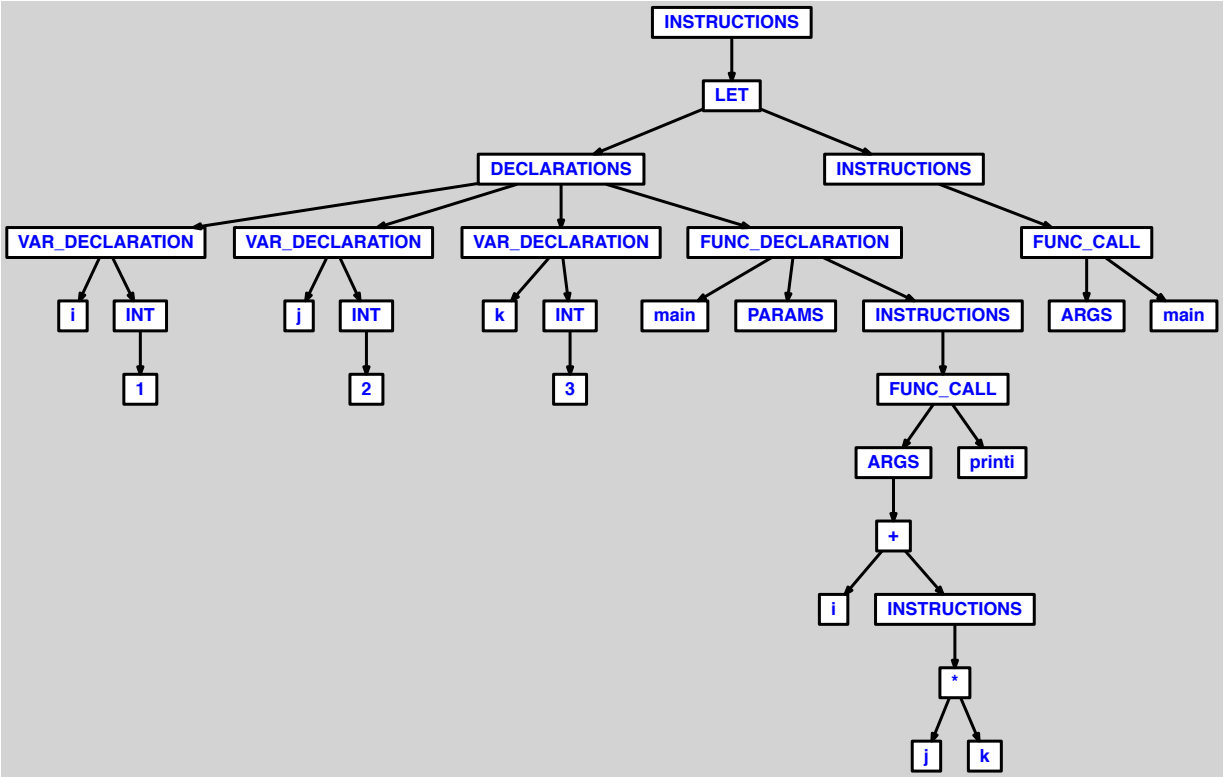
3.2.74 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
i
INSTRUCTIONS
-
j
k

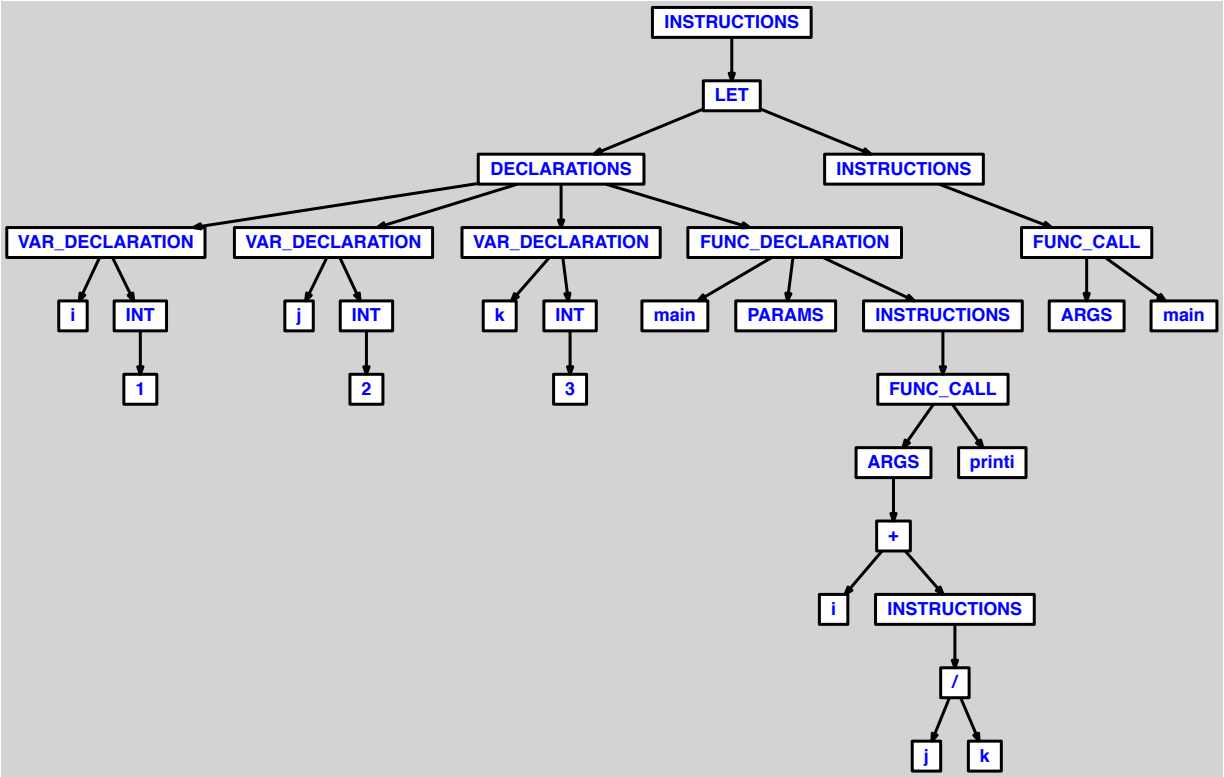
3.2.75 addition suivie de multiplication, avec termes identifies par variables et parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
i
INSTRUCTIONS
*
j
k

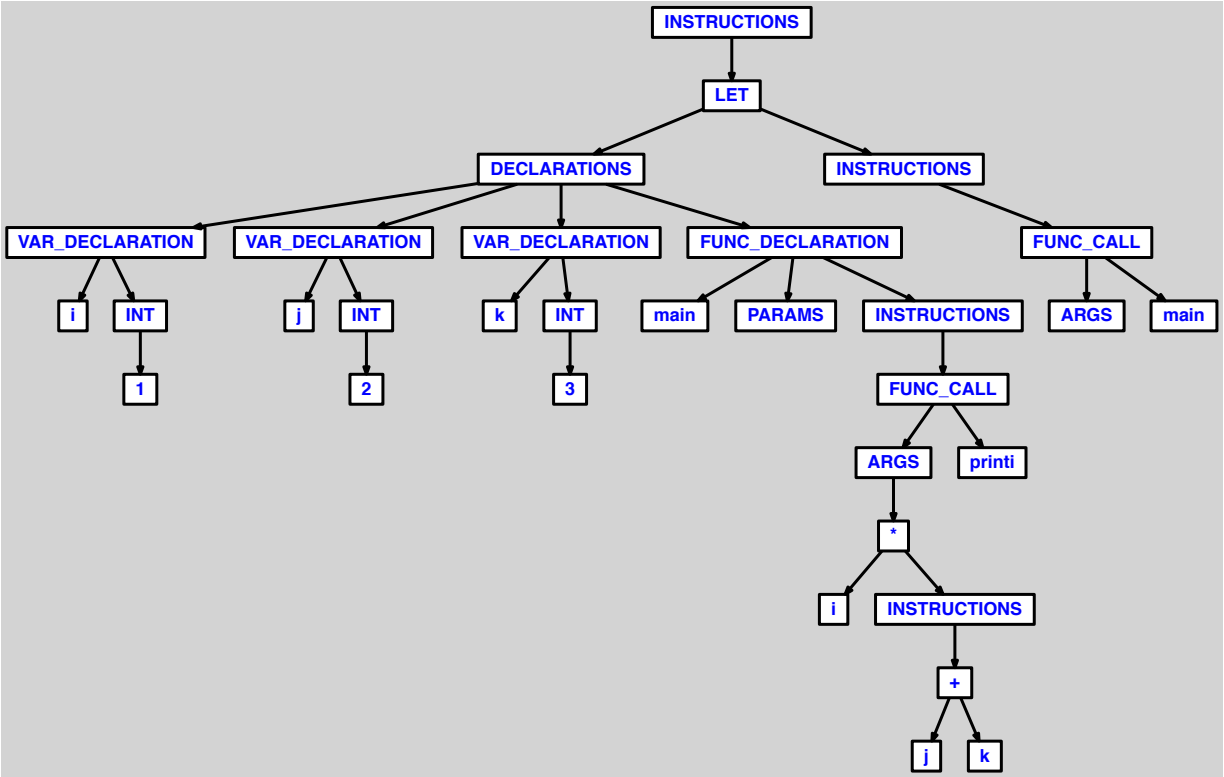
3.2.76 addition suivie de division, avec termes identifies par variables et parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
i
INSTRUCTIONS
/
j
k

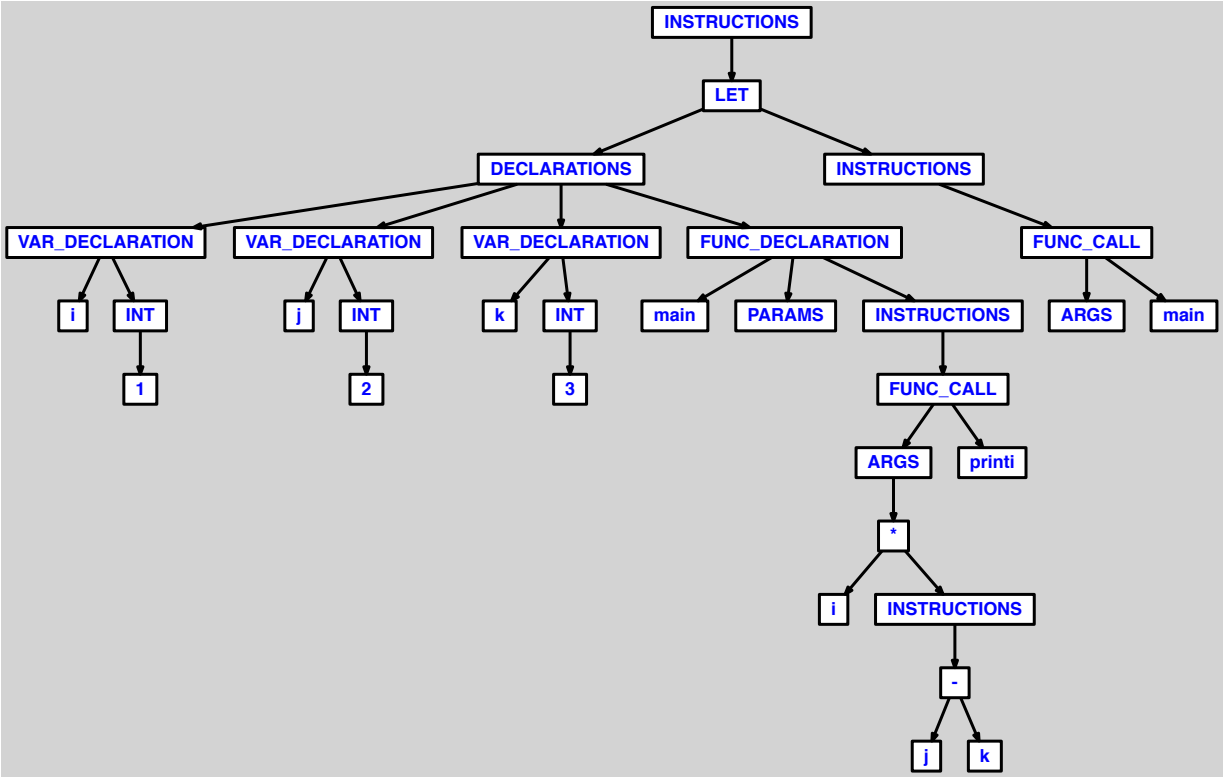
3.2.77 multiplication suivie d'addition, avec termes identifies par variables et parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
i
INSTRUCTIONS
+
j
k

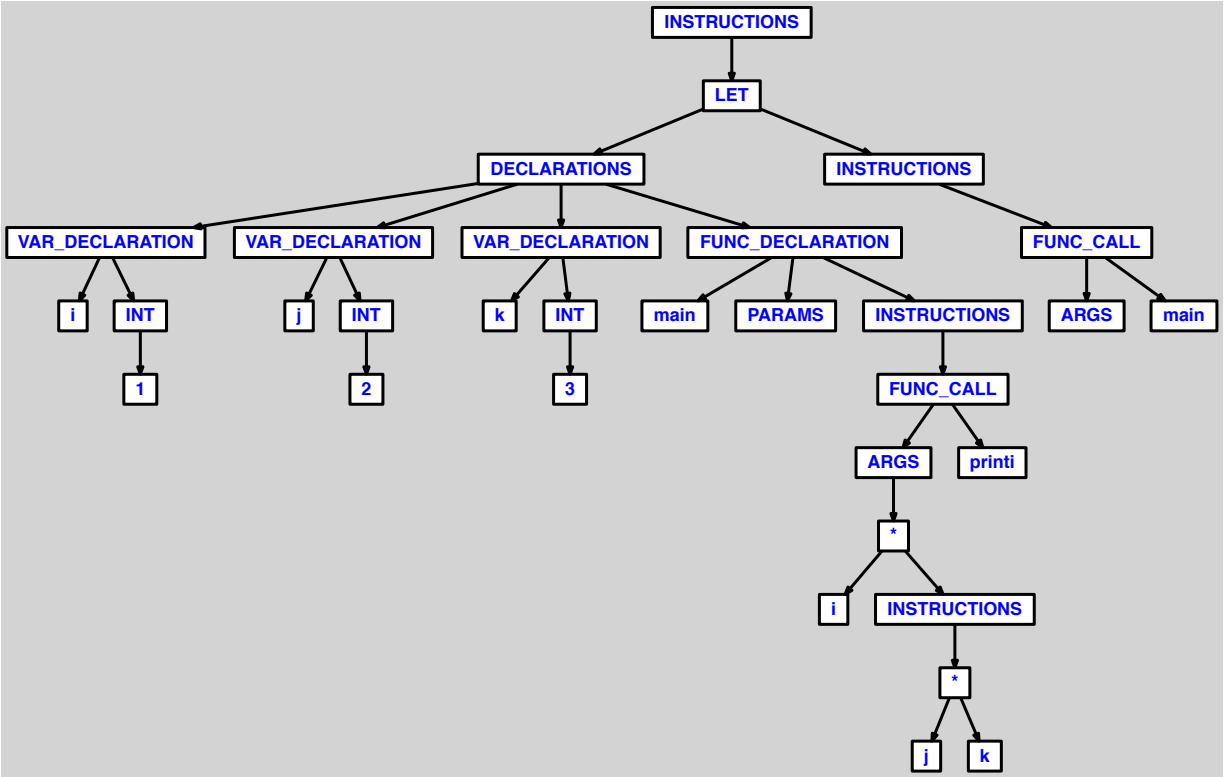
3.2.78 multiplication suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
i
INSTRUCTIONS
-
j
k

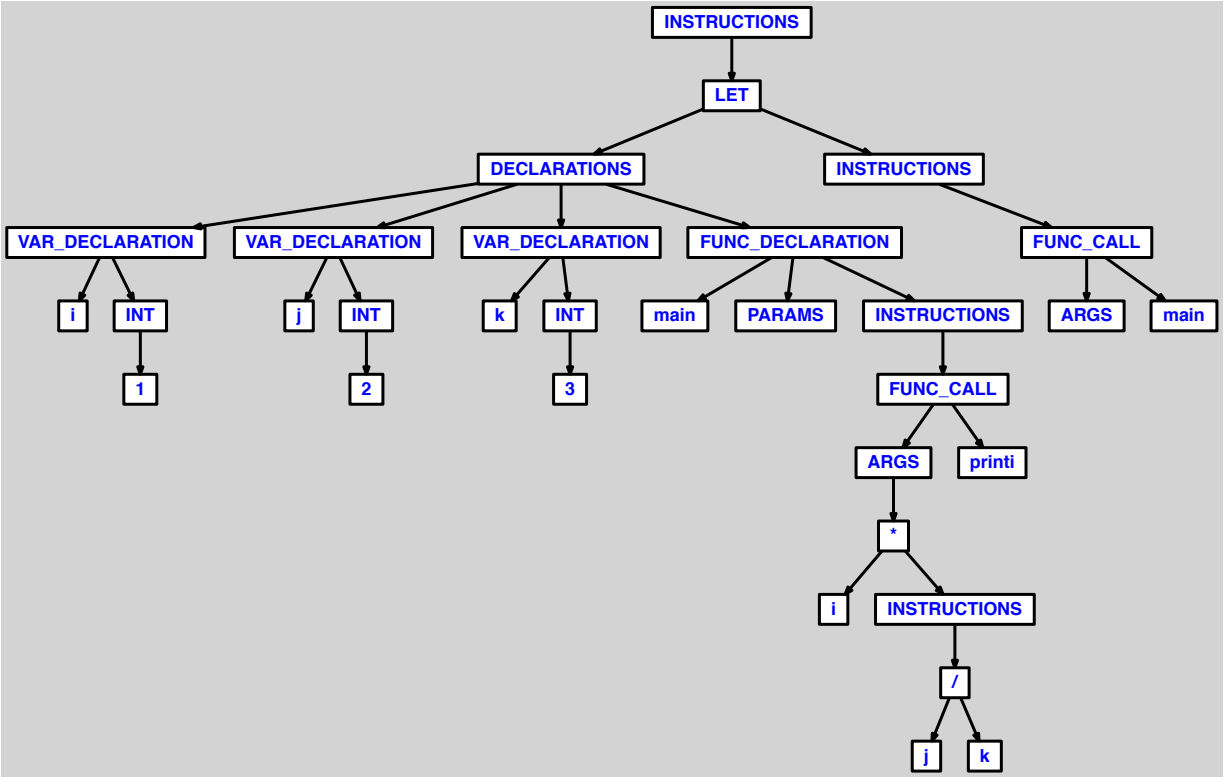
3.2.79 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
i
INSTRUCTIONS
*
j
k

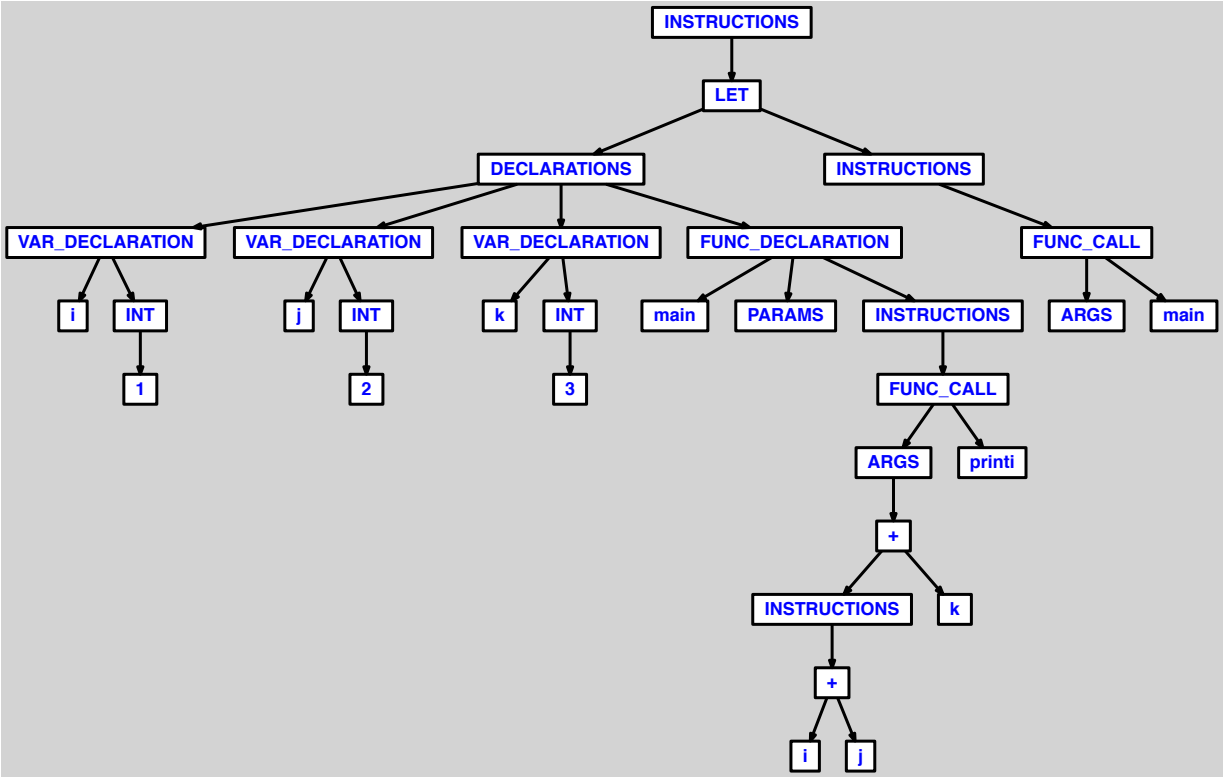
3.2.80 multiplication suivie de division, avec termes identifies par variables et parenthesage des 2 variables a droite

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
i
INSTRUCTIONS
/
j
k

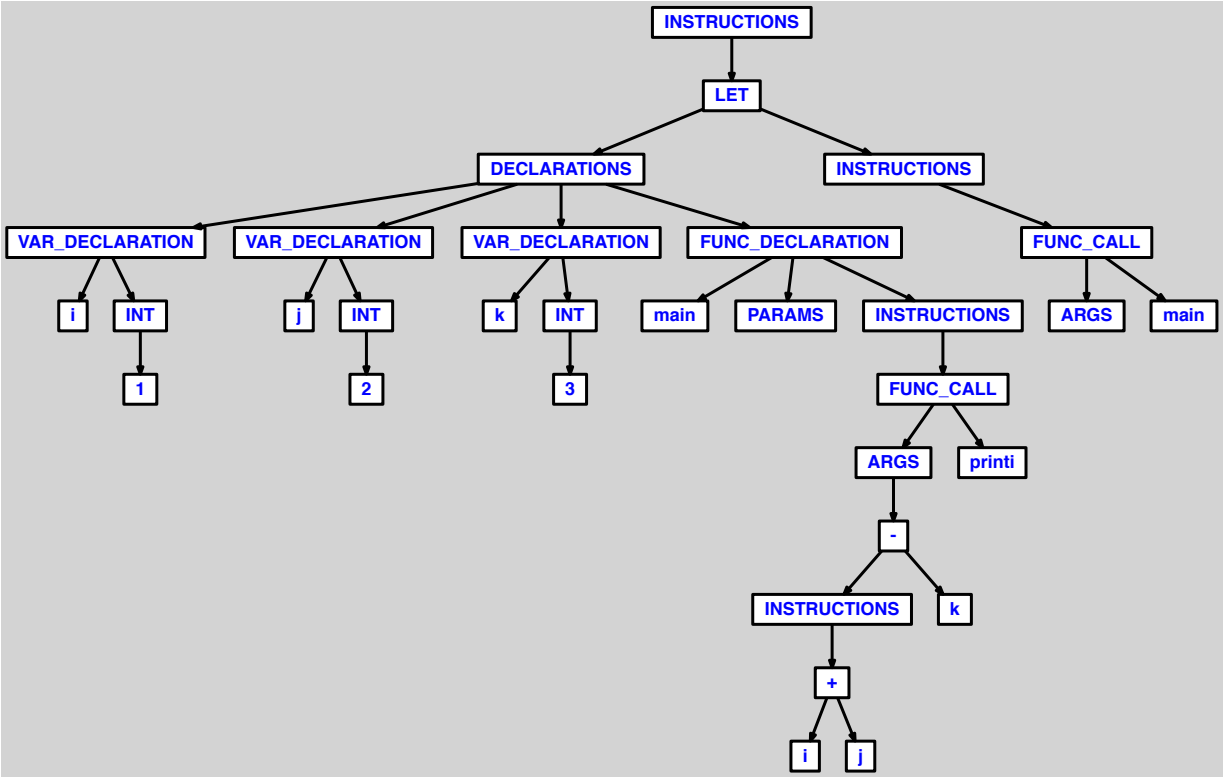
3.2.81 addition a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
INSTRUCTIONS
k
+
i
j

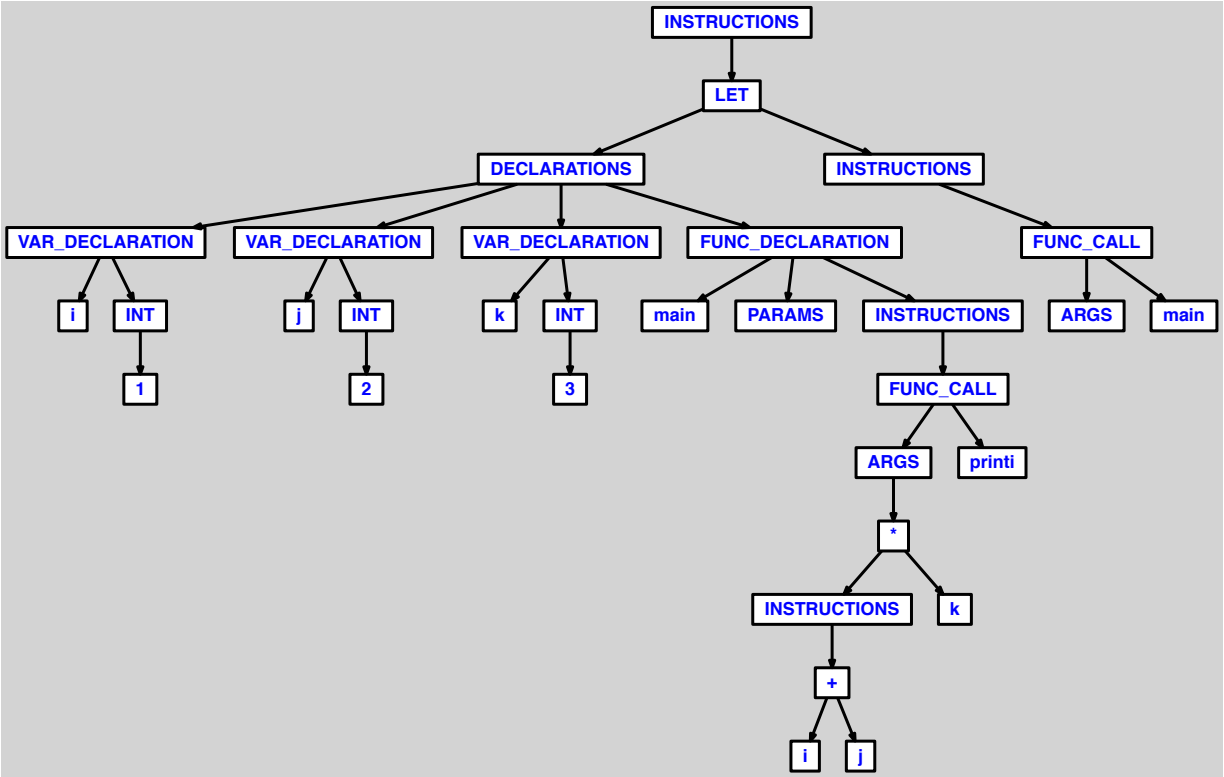
3.2.82 addition suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
-
INSTRUCTIONS
k
+
i
j

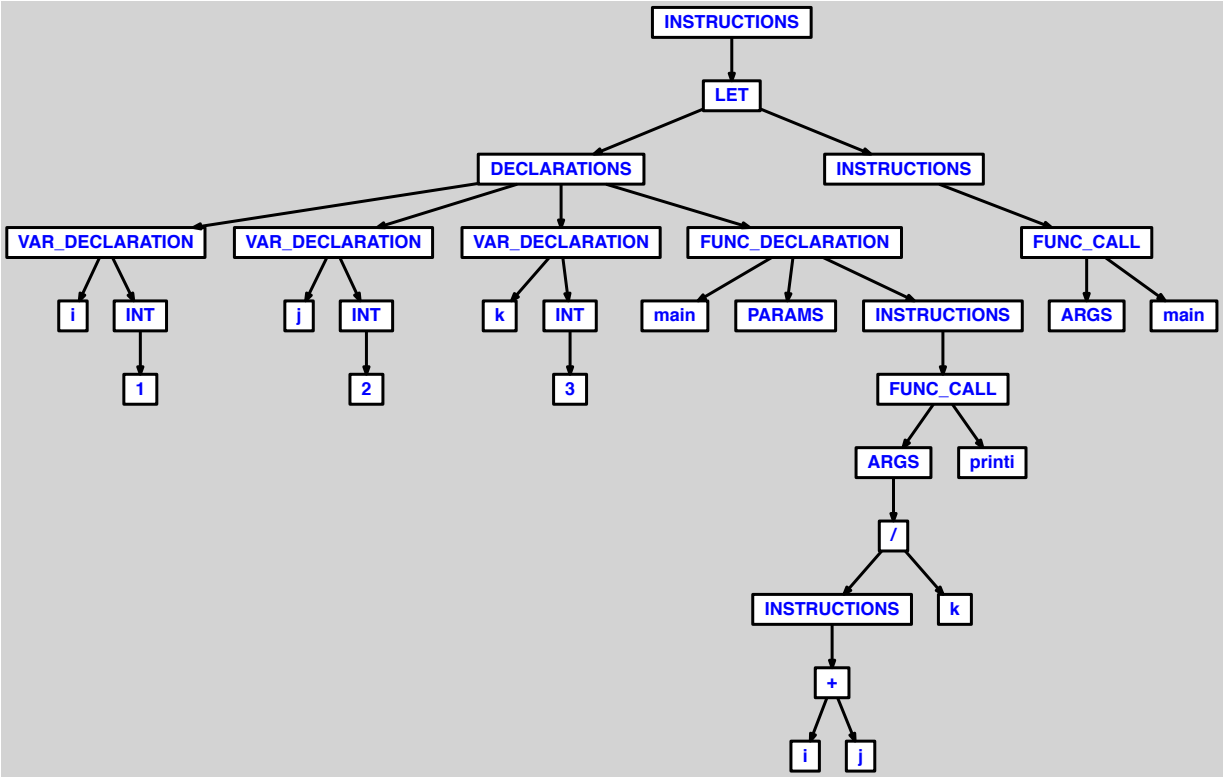
3.2.83 addition suivie de multiplication, avec termes identifies par variables et parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
INSTRUCTIONS
k
+
i
j

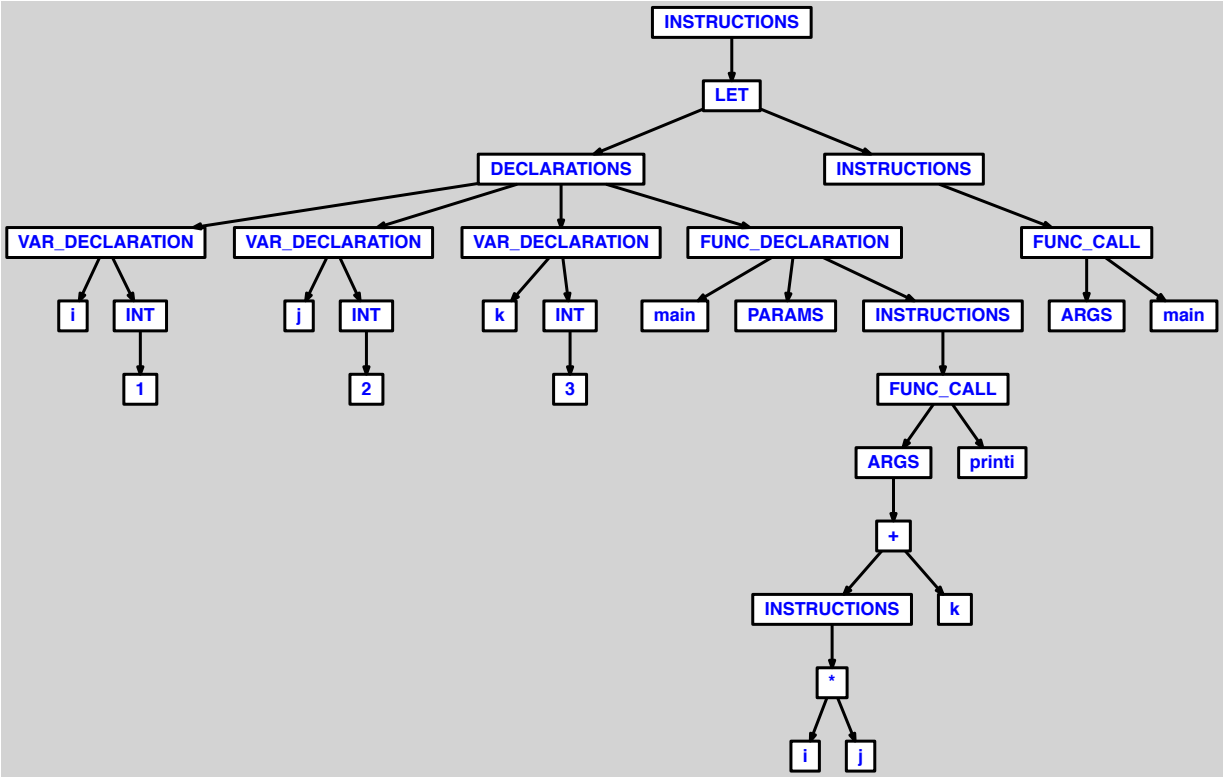
3.2.84 addition suivie de division, avec termes identifies par variables et parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
k
+
i
j

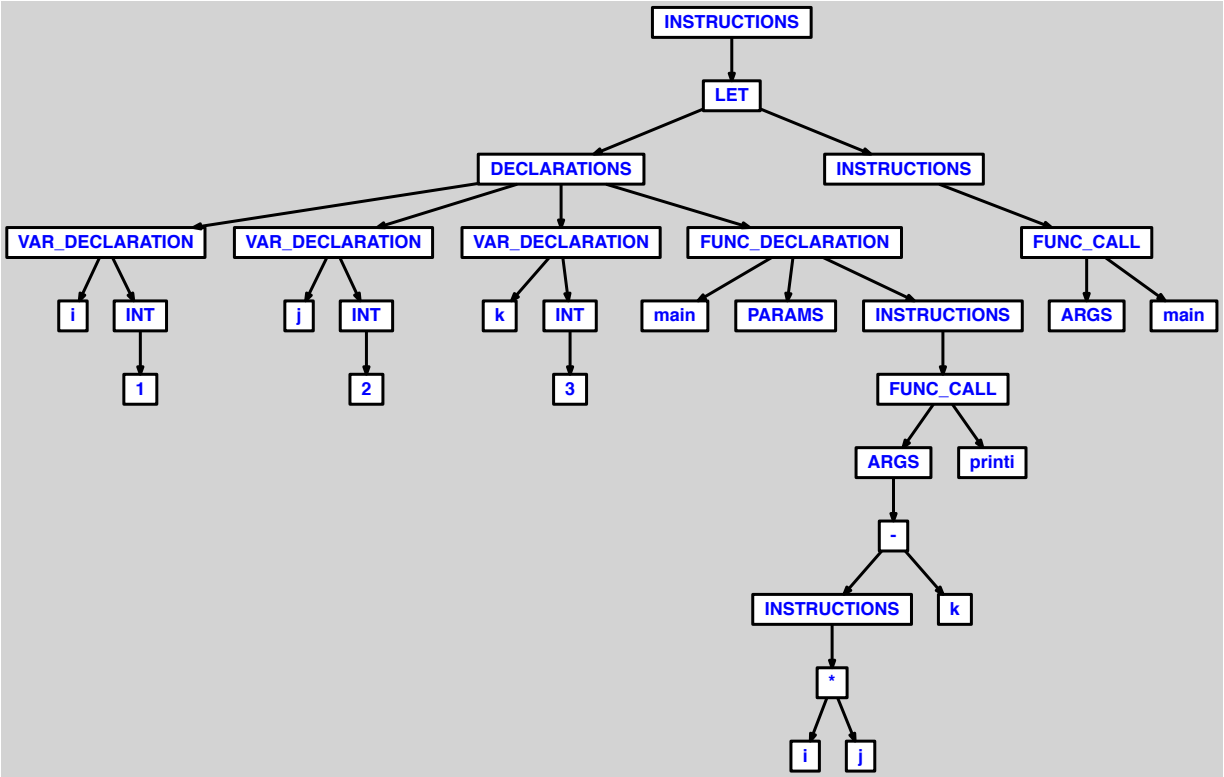
3.2.85 multiplication suivie d'addition, avec termes identifies par variables et parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
INSTRUCTIONS
k
*
i
j

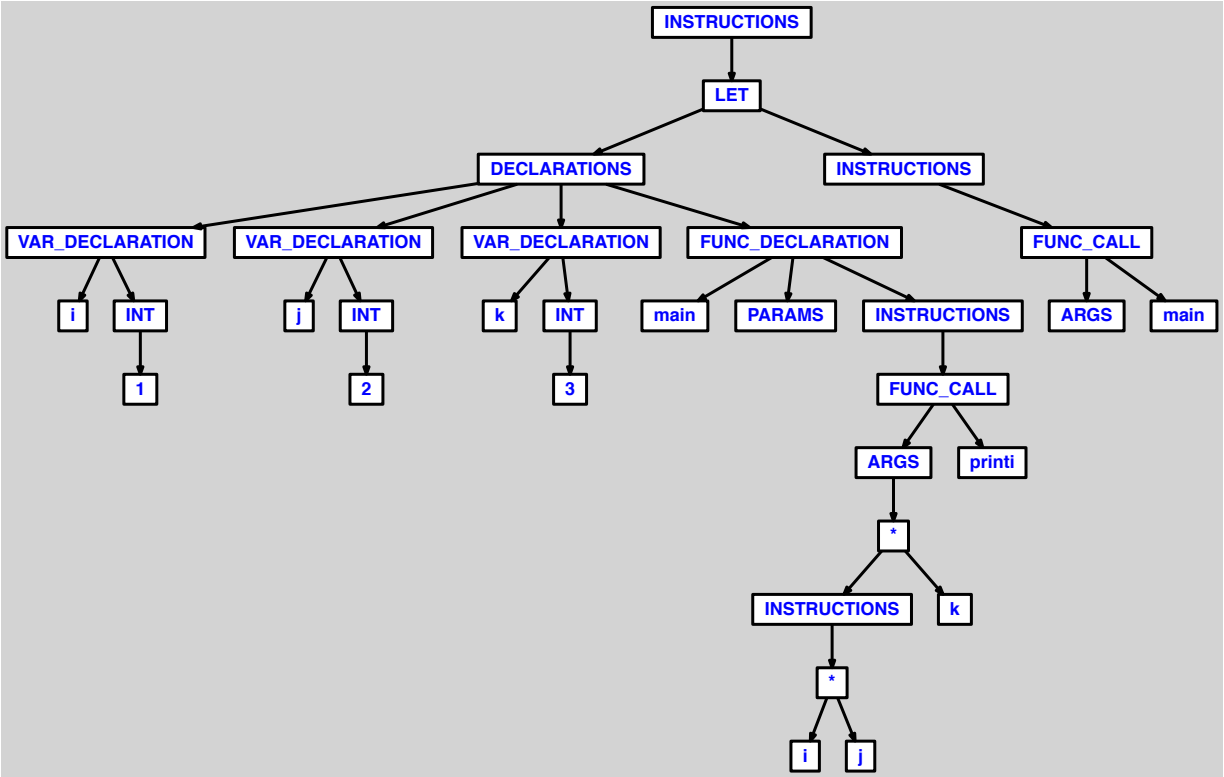
3.2.86 multiplication suivie de soustraction, avec termes identifies par variables et parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
-
INSTRUCTIONS
k
*
i
j

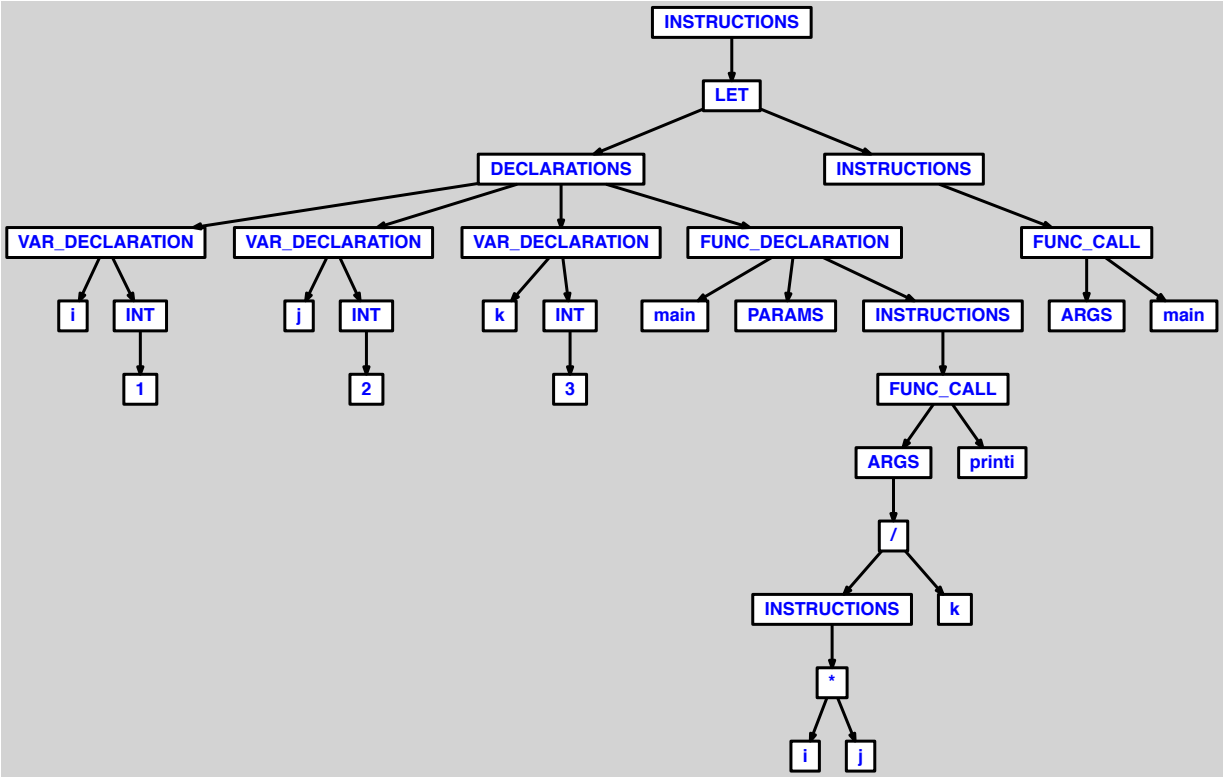
3.2.87 multiplication a 3 termes identifies par variables, avec parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
INSTRUCTIONS
k
*
i
j

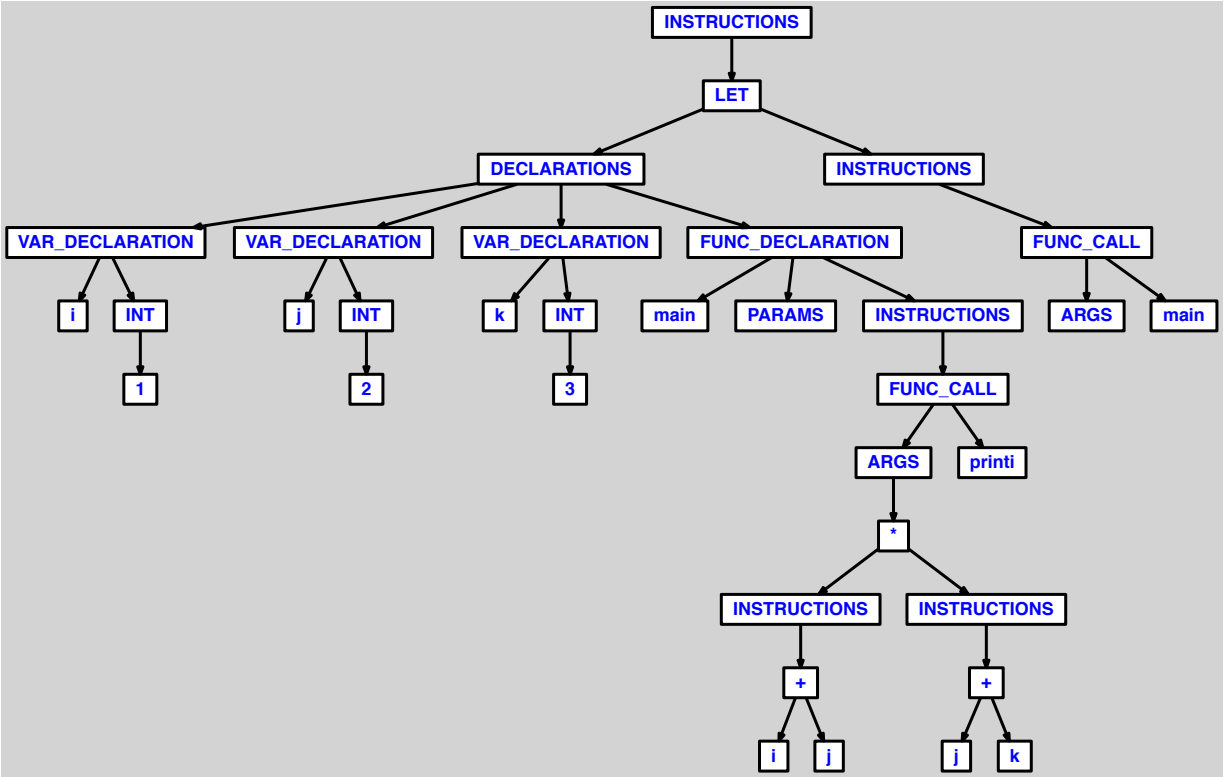
3.2.88 multiplication suivie de division, avec termes identifies par variables et parenthesage des 2 variables a gauche

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
k
*
i
j

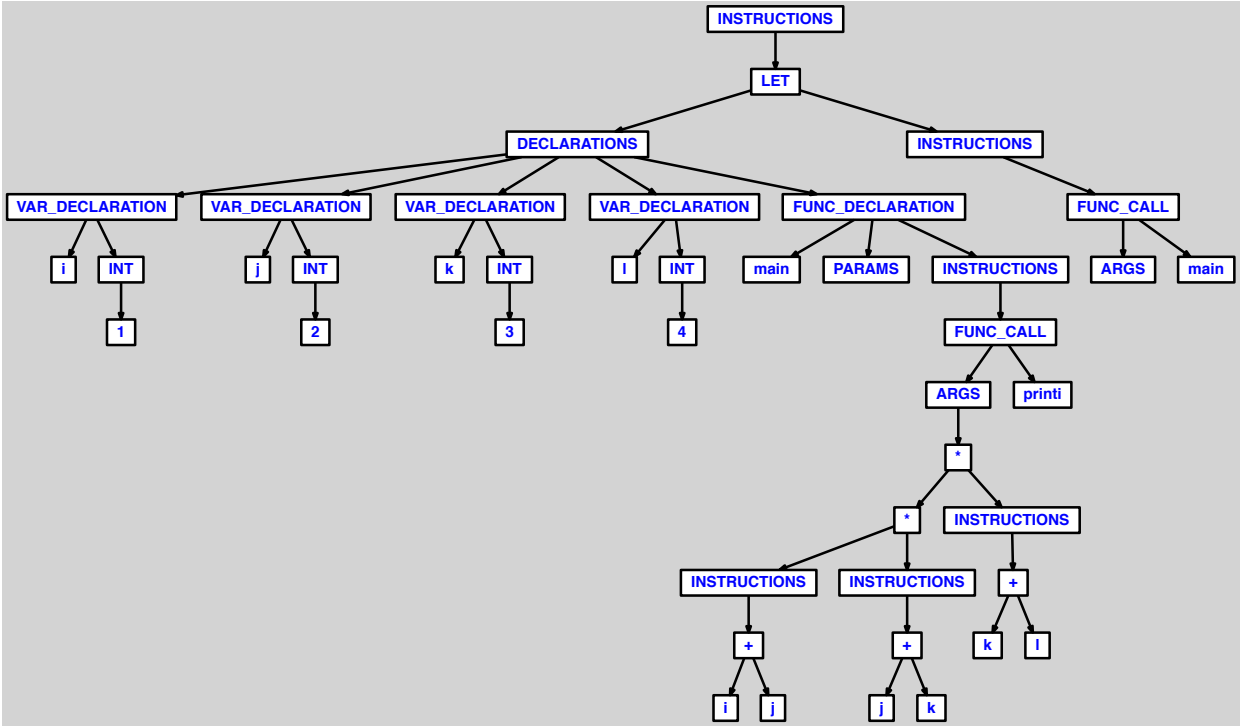
3.2.89 multiplication de 2 additions parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
*
INSTRUCTIONS
INSTRUCTIONS
+
+
i
j
j
k

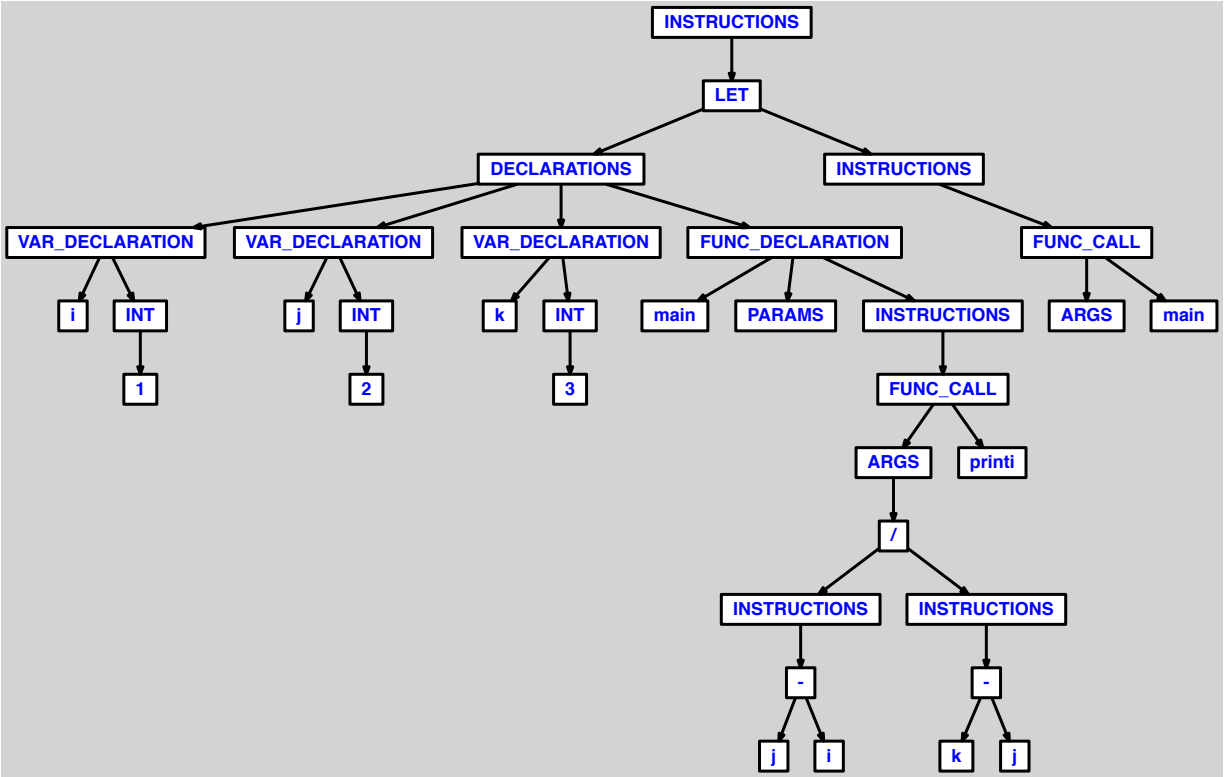
3.2.90 multiplication de 3 additions parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
l
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
4
FUNC_CALL
ARGS
printi
*
*
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
+
+
+
k
l
i
j
j
k

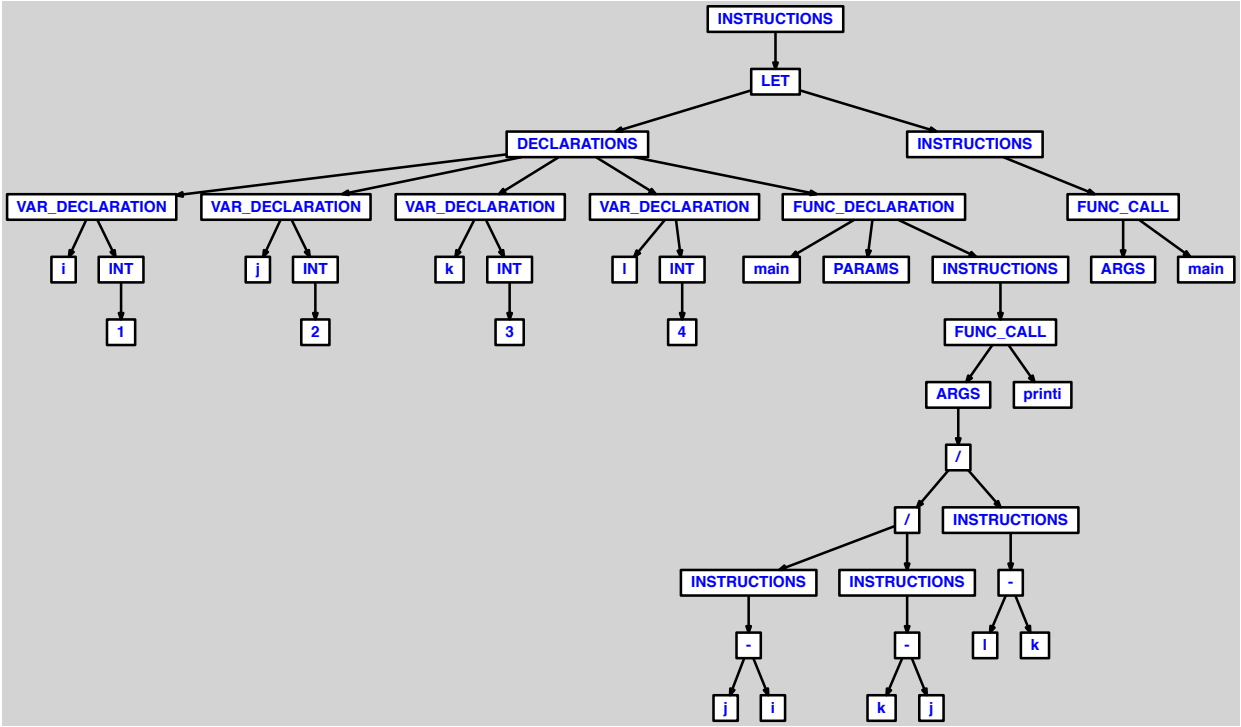
3.2.91 division de 2 soustractions parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
/
INSTRUCTIONS
INSTRUCTIONS
-
-
j
i
k
j

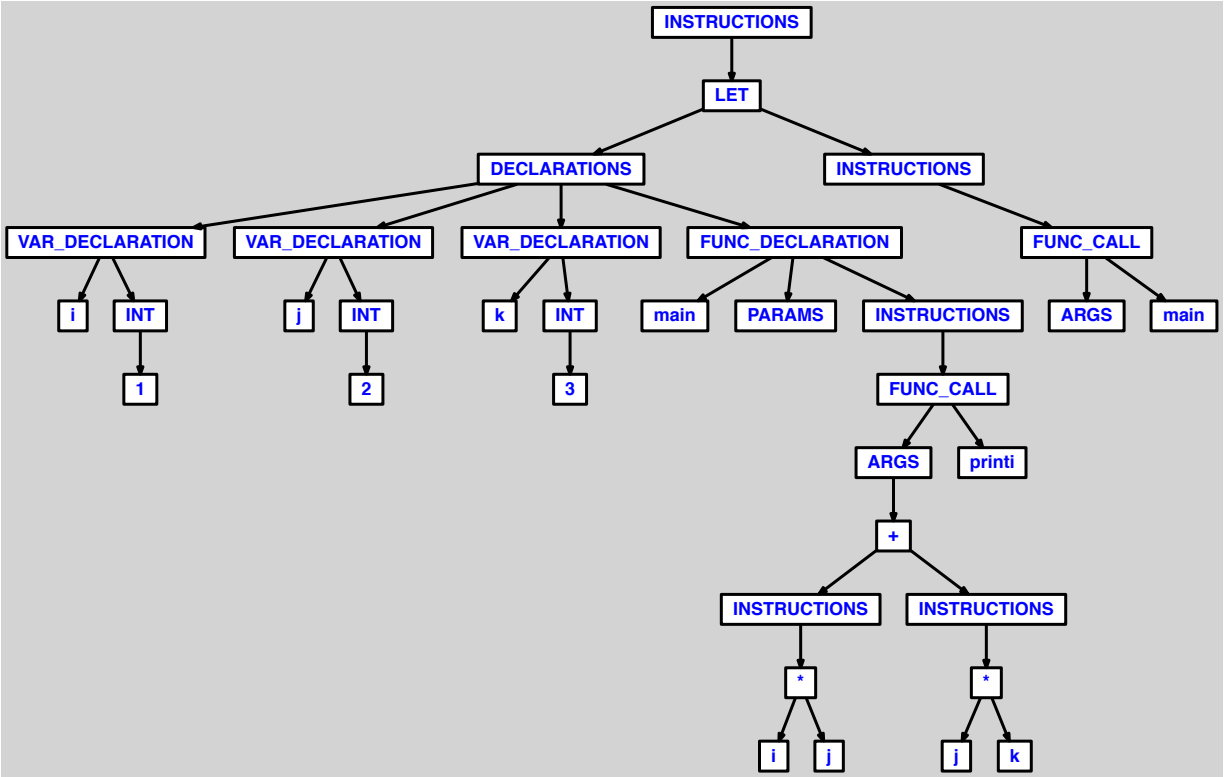
3.2.92 division de 3 soustractions parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
l
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
4
FUNC_CALL
ARGS
printi
/
/
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
-
-
-
l
k
j
i
k
j

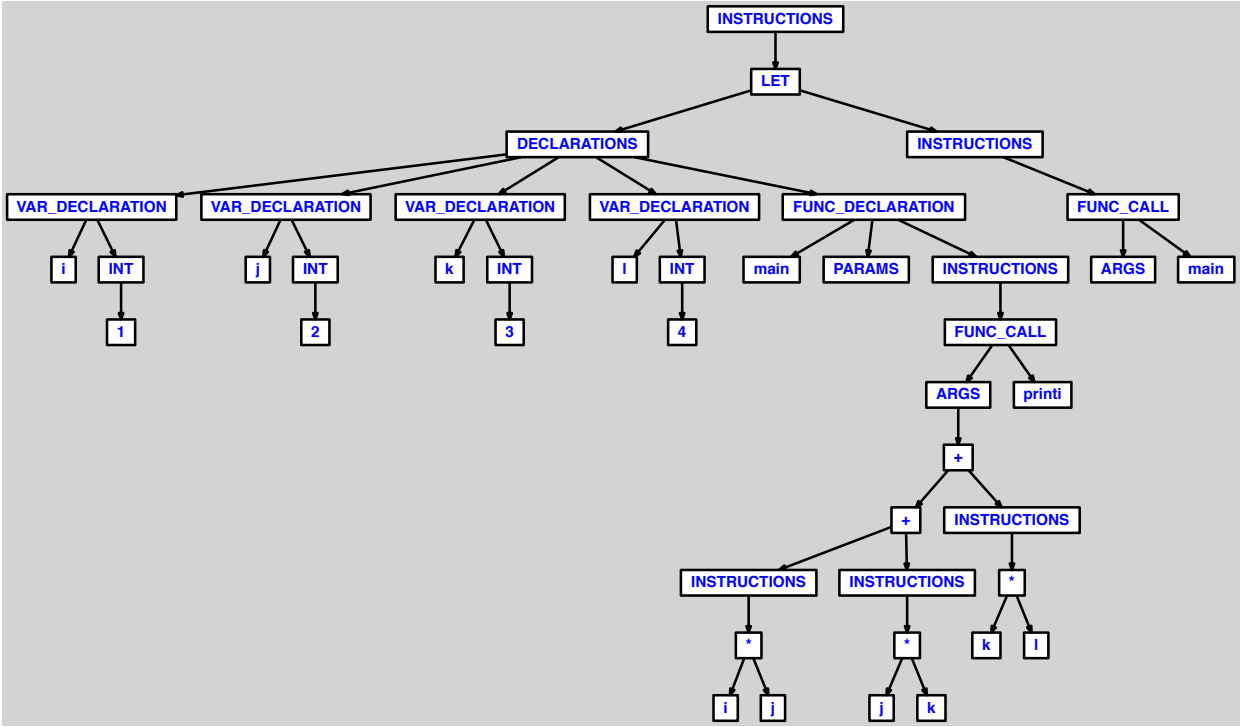
3.2.93 addition de 2 multiplications parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
FUNC_CALL
ARGS
printi
+
INSTRUCTIONS
INSTRUCTIONS
*
*
i
j
j
k

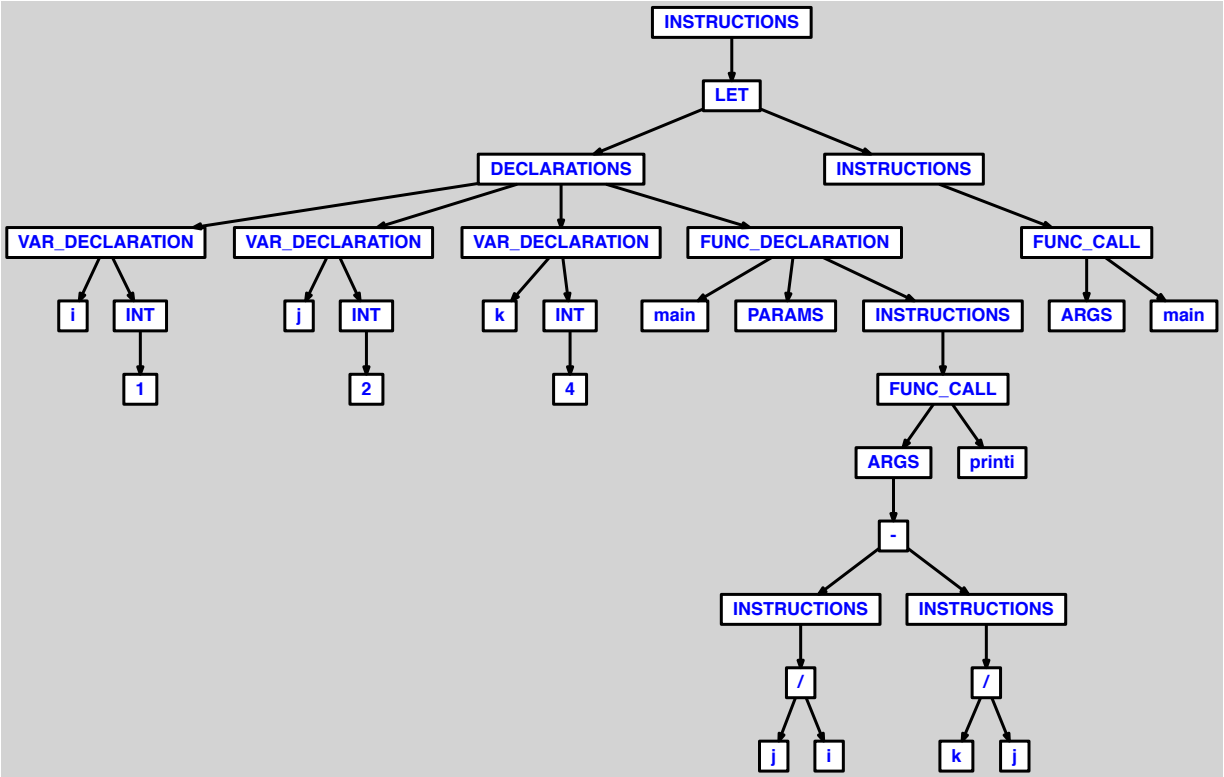
3.2.94 addition de 3 multiplications parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
l
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
3
4
FUNC_CALL
ARGS
printi
+
+
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
*
*
*
k
l
i
j
j
k

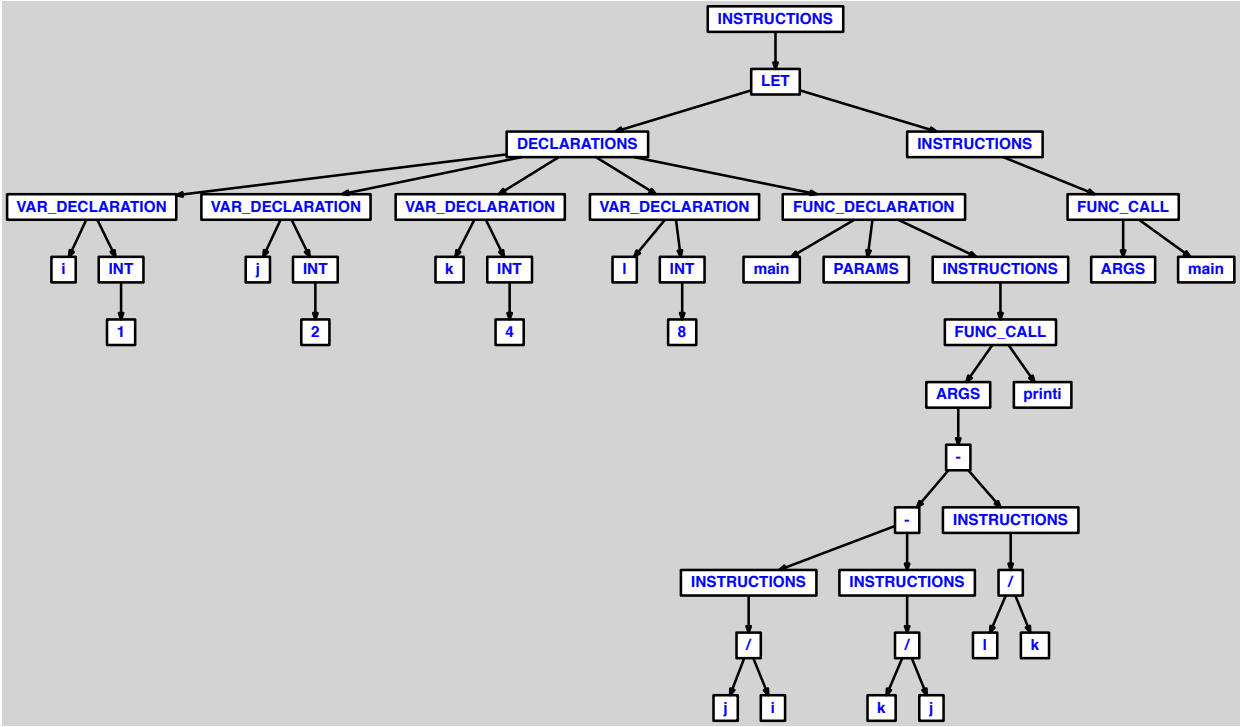
3.2.95 soustraction de 2 divisions parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
4
FUNC_CALL
ARGS
printi
-
INSTRUCTIONS
INSTRUCTIONS
/
/
j
i
k
j

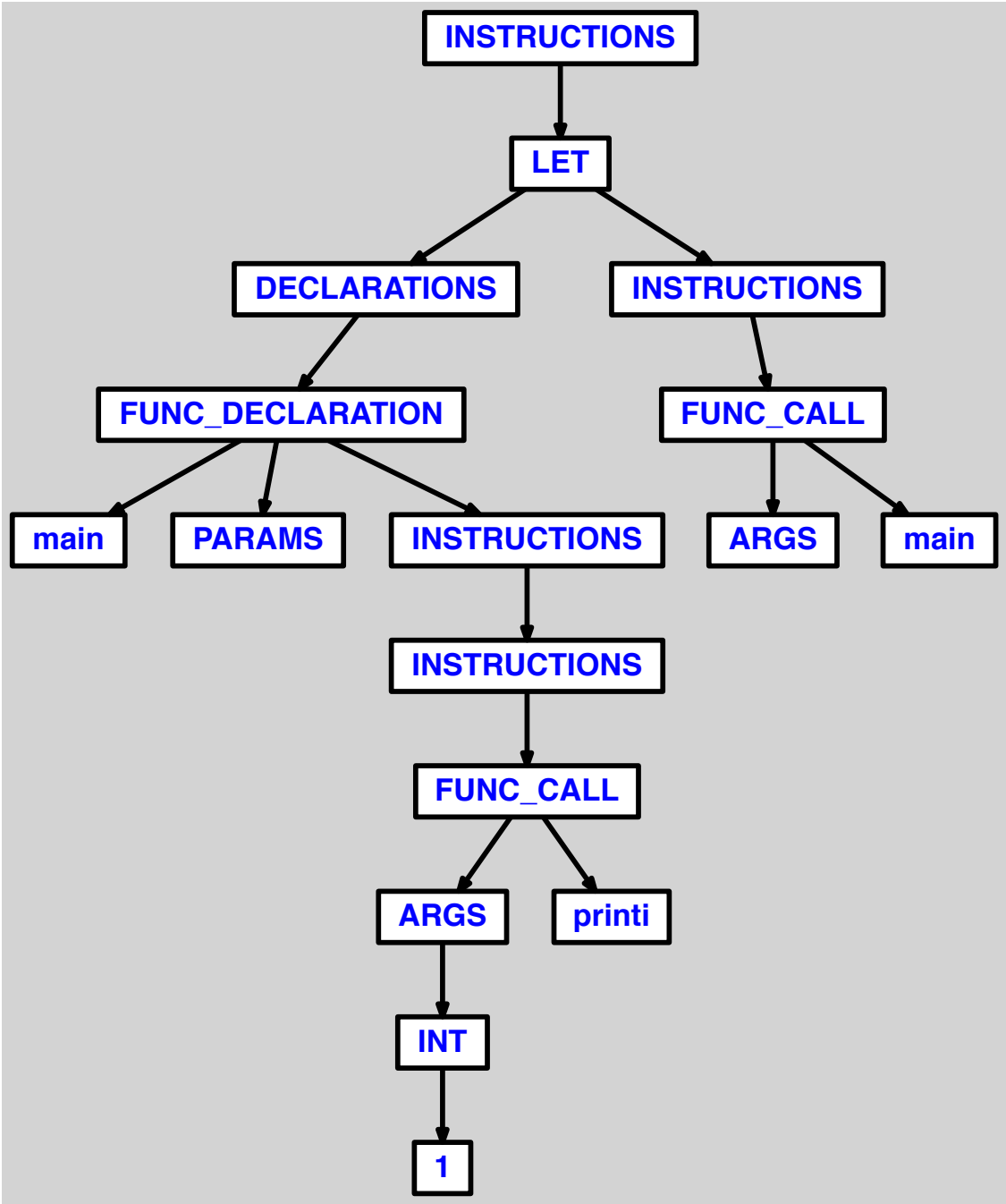
3.2.96 soustraction de 3 divisions parenthesees, avec termes identifies par variables

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
l
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
2
4
8
FUNC_CALL
ARGS
printi
-
-
INSTRUCTIONS
INSTRUCTIONS
INSTRUCTIONS
/
/
/
l
k
j
i
k
j

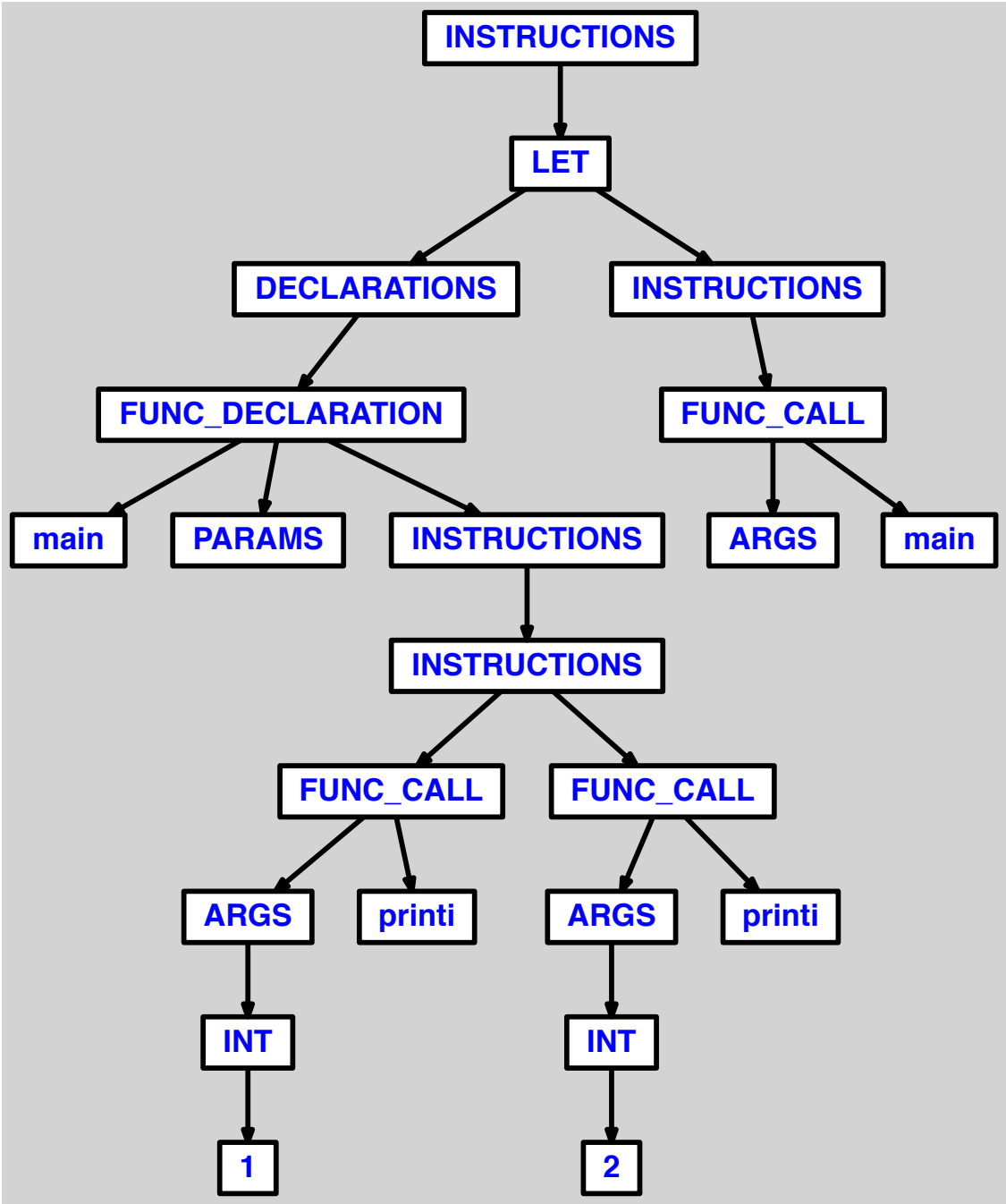
3.2.97 parenthesage d'une instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
ARGS
printi
INT
1

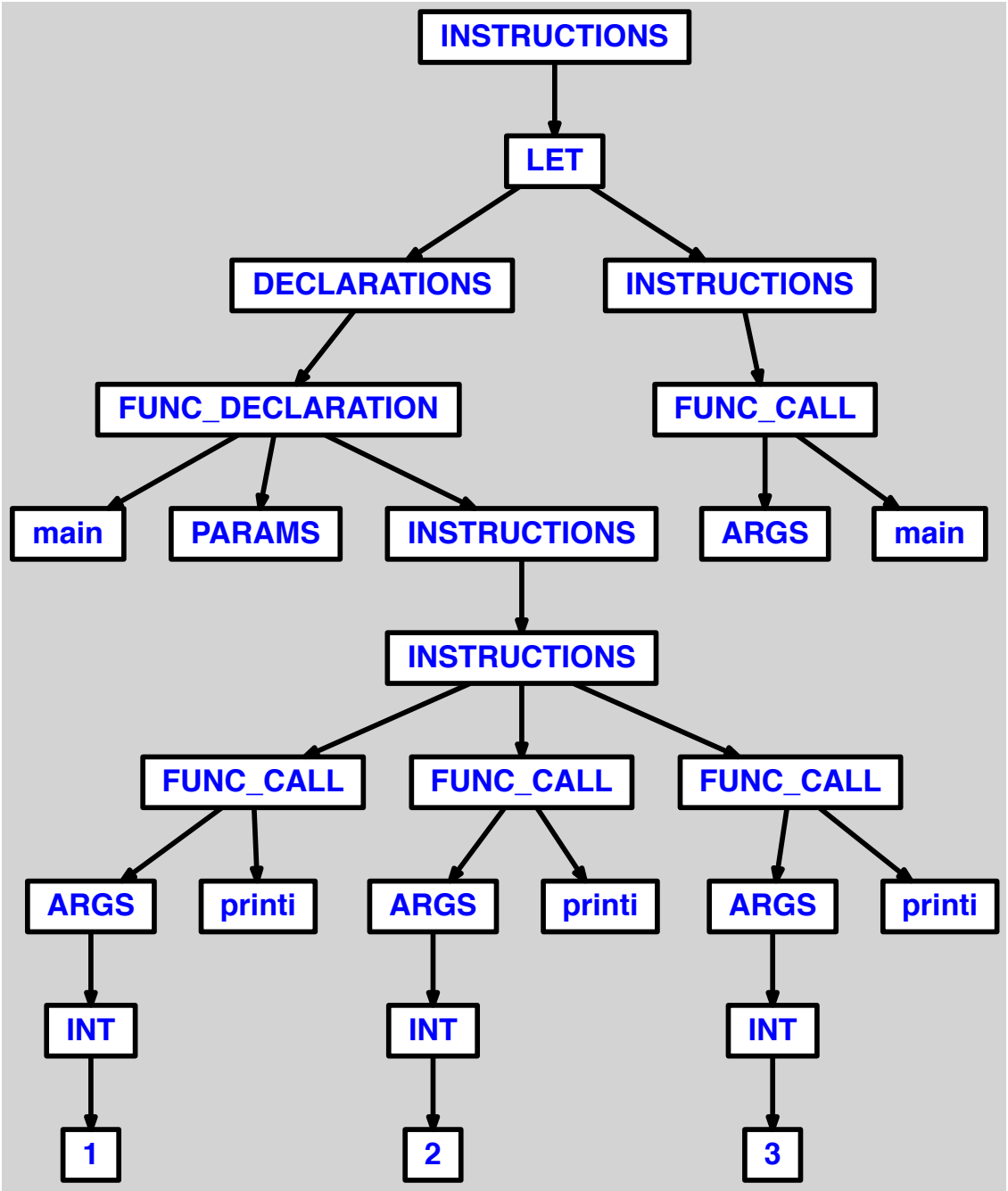
3.2.98 parenthesage de 2 instructions

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
ARGS
printi
ARGS
printi
INT
INT
1
2

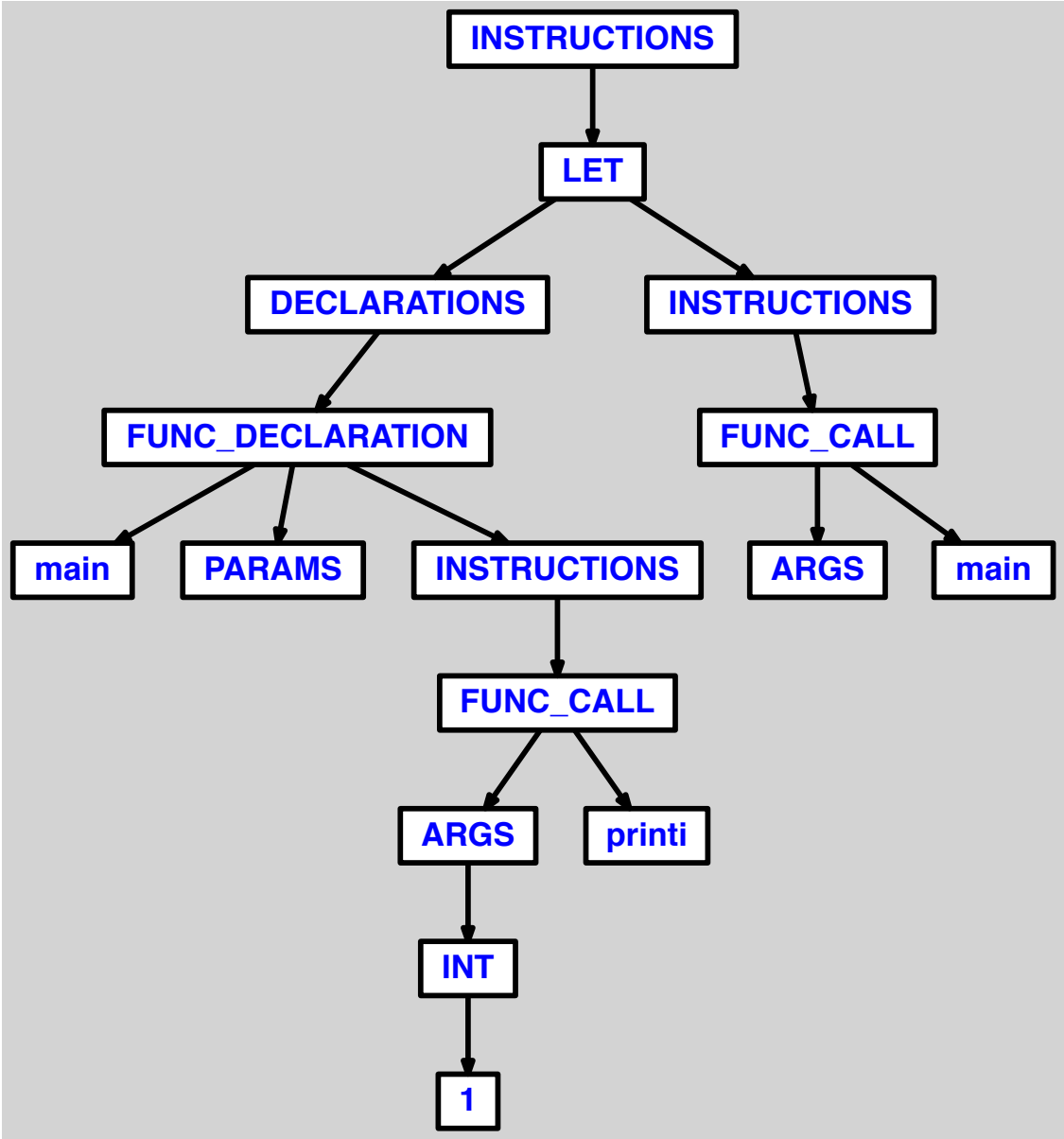
3.2.99 parenthesage de 3 instructions

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
printi
ARGS
printi
ARGS
printi
INT
INT
INT
1
2
3

3.2.100 1 instruction sans parentheses

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

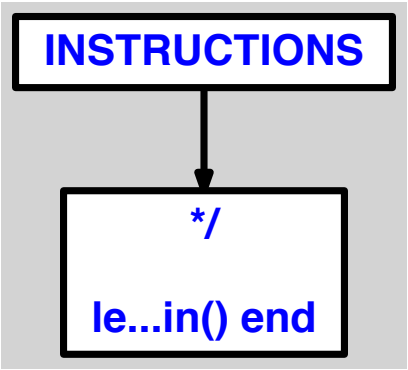

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
INT
1

4 commentaire

4.1 KO

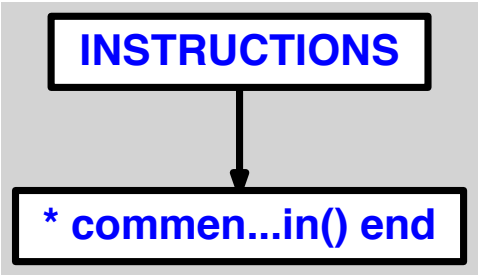
4.1.1 imbrication

```
/*
  un commentaire
  /* imbrique dans ce commentaire */
*/

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


INSTRUCTIONS
*/
le...in() end

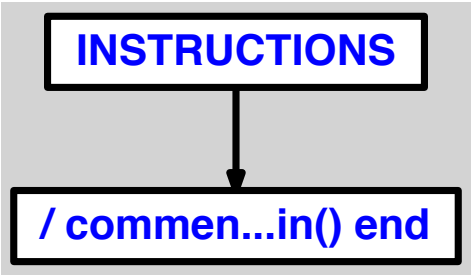
4.1.2 oubli de </> en debut de commentaire

```
* commentaire */

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


INSTRUCTIONS
* commen...in() end

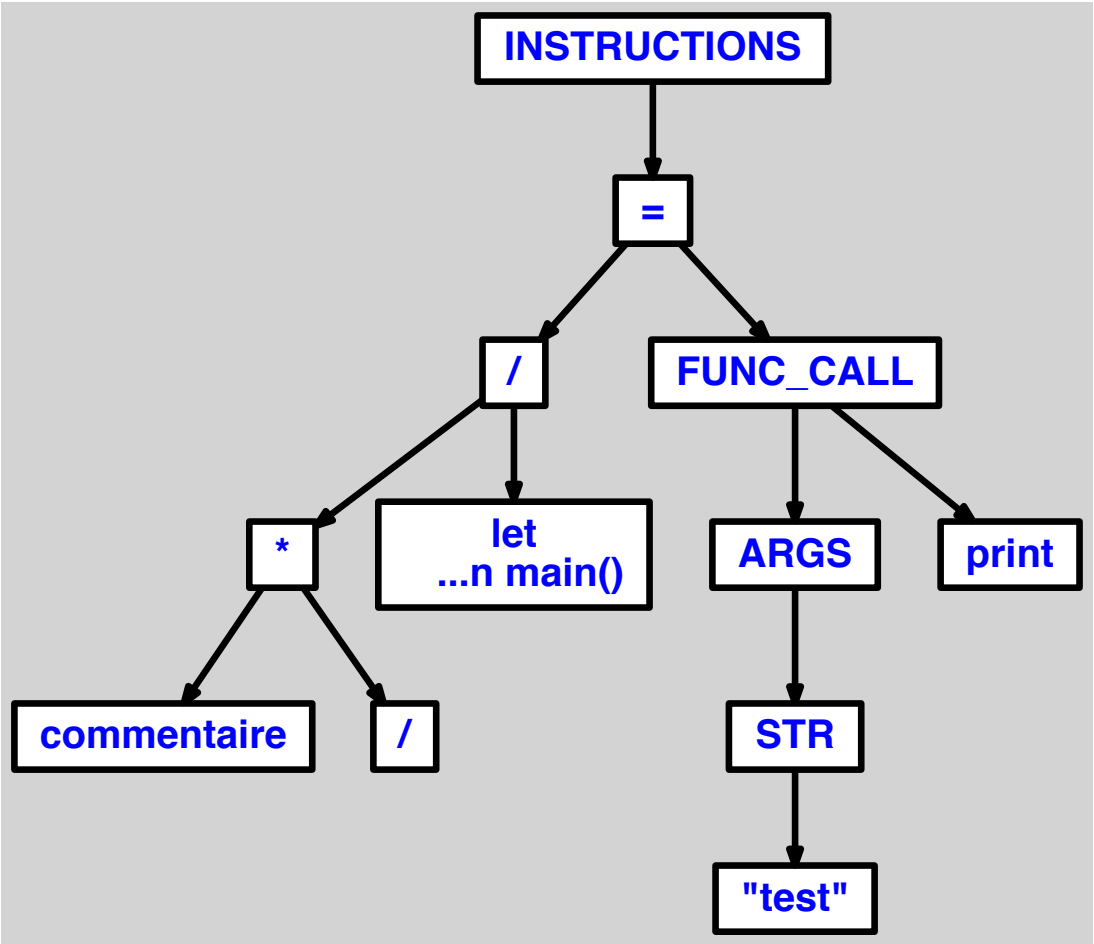
4.1.3 oubli de <*> en debut de commentaire

```
/ commentaire */

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


INSTRUCTIONS
/ commen...in() end

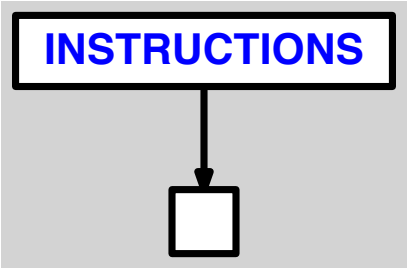
4.1.4 oubli de </*> en debut de commentaire

```
commentaire */

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


INSTRUCTIONS
=
/
FUNC_CALL
*
let
...n main()
ARGS
print
commentaire
/
STR
"test"

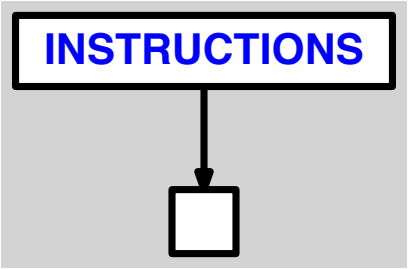
4.1.5 oubli de <*> en fin de commentaire

```
/* commentaire /

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


INSTRUCTIONS

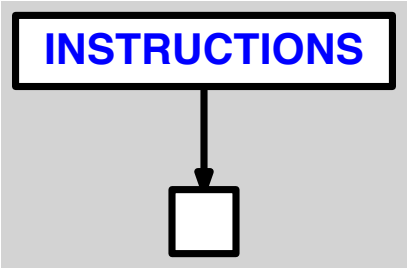
4.1.6 oubli de </> en fin de commentaire

```
/* commentaire *

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


INSTRUCTIONS

4.1.7 oubli de <*/> en fin de commentaire

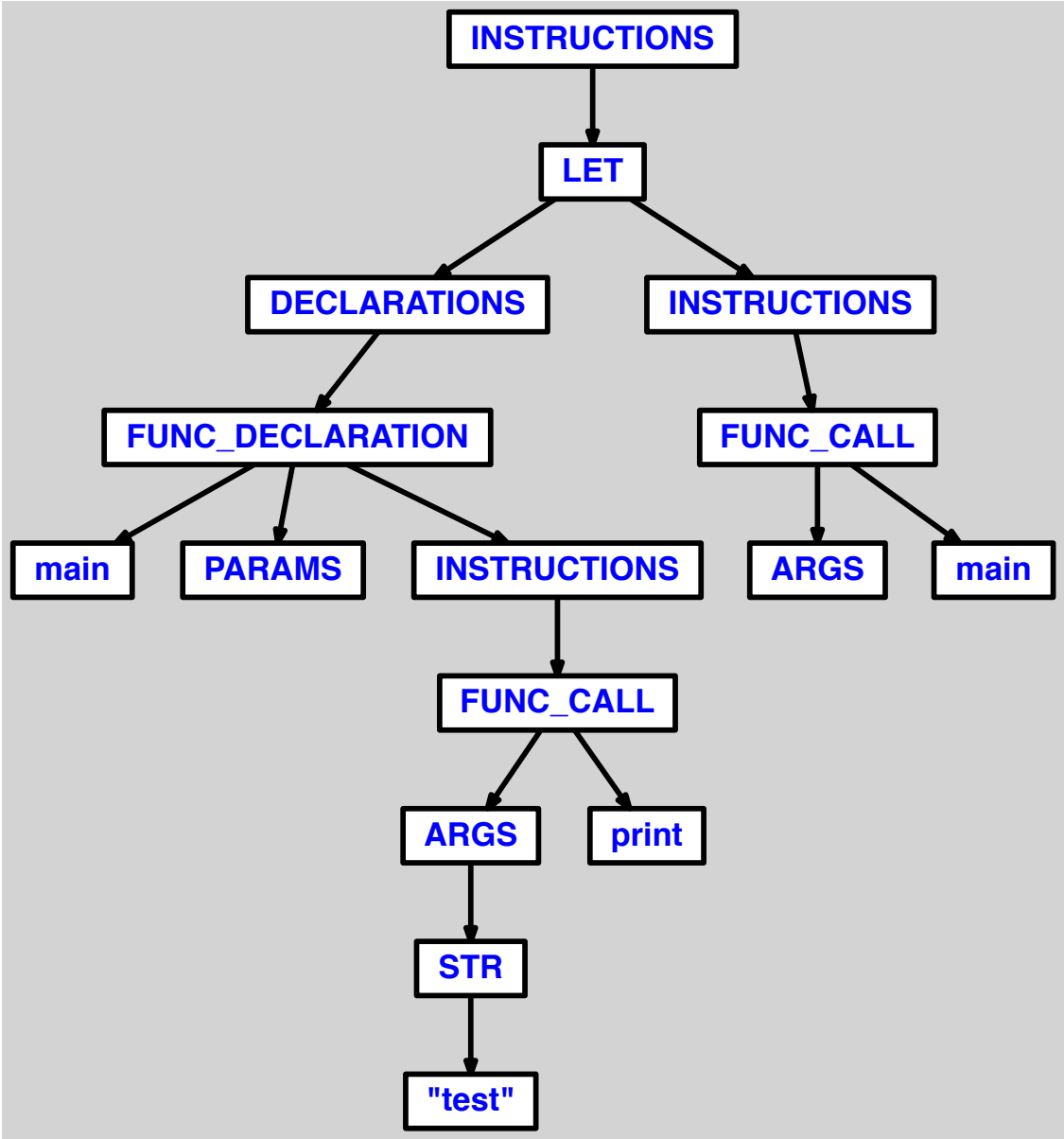
```
/* commentaire

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


INSTRUCTIONS

4.2 OK

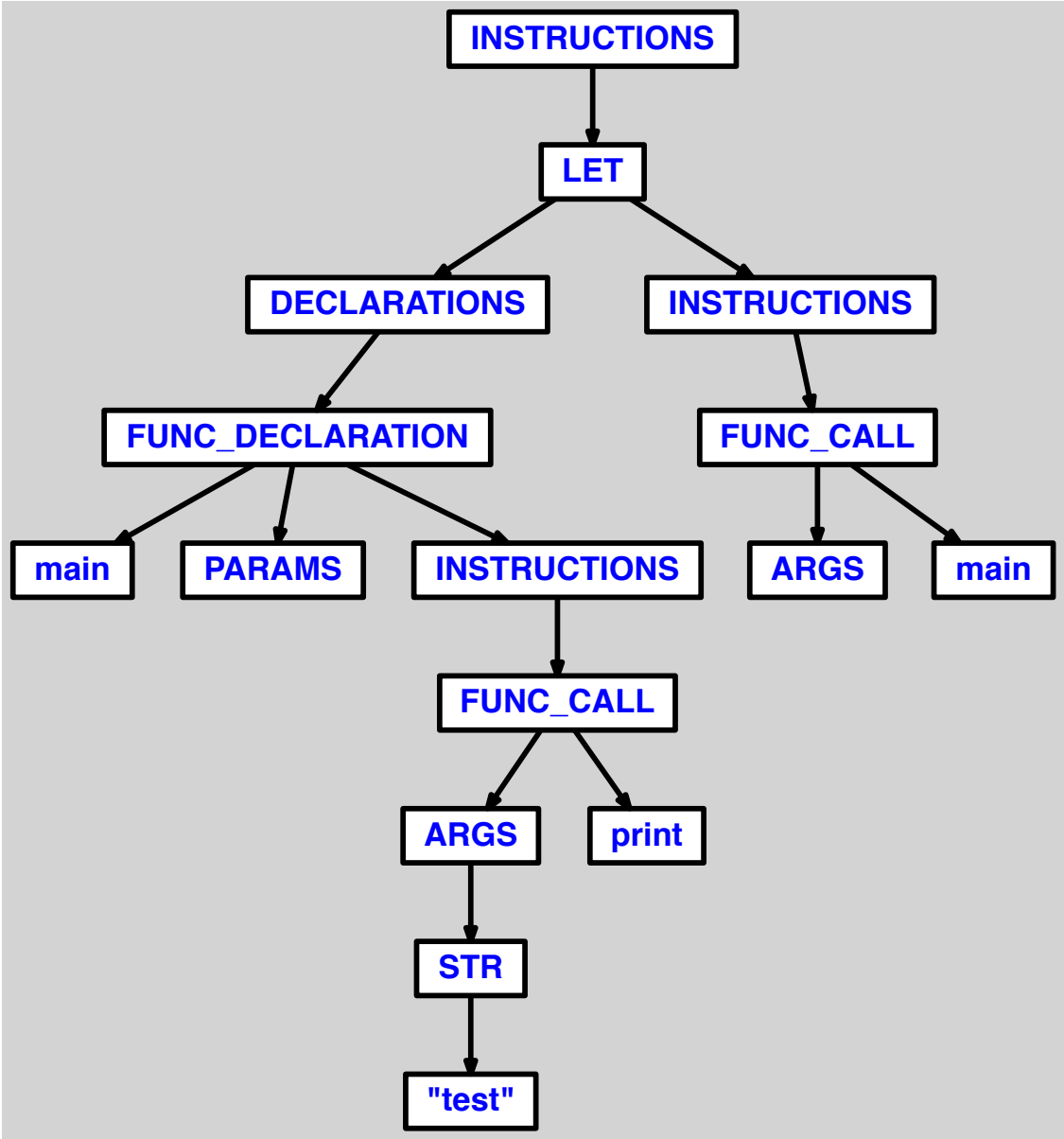
4.2.1 1 commentaire, sans instruction avant, sur 1 ligne

```
/* commentaire */

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

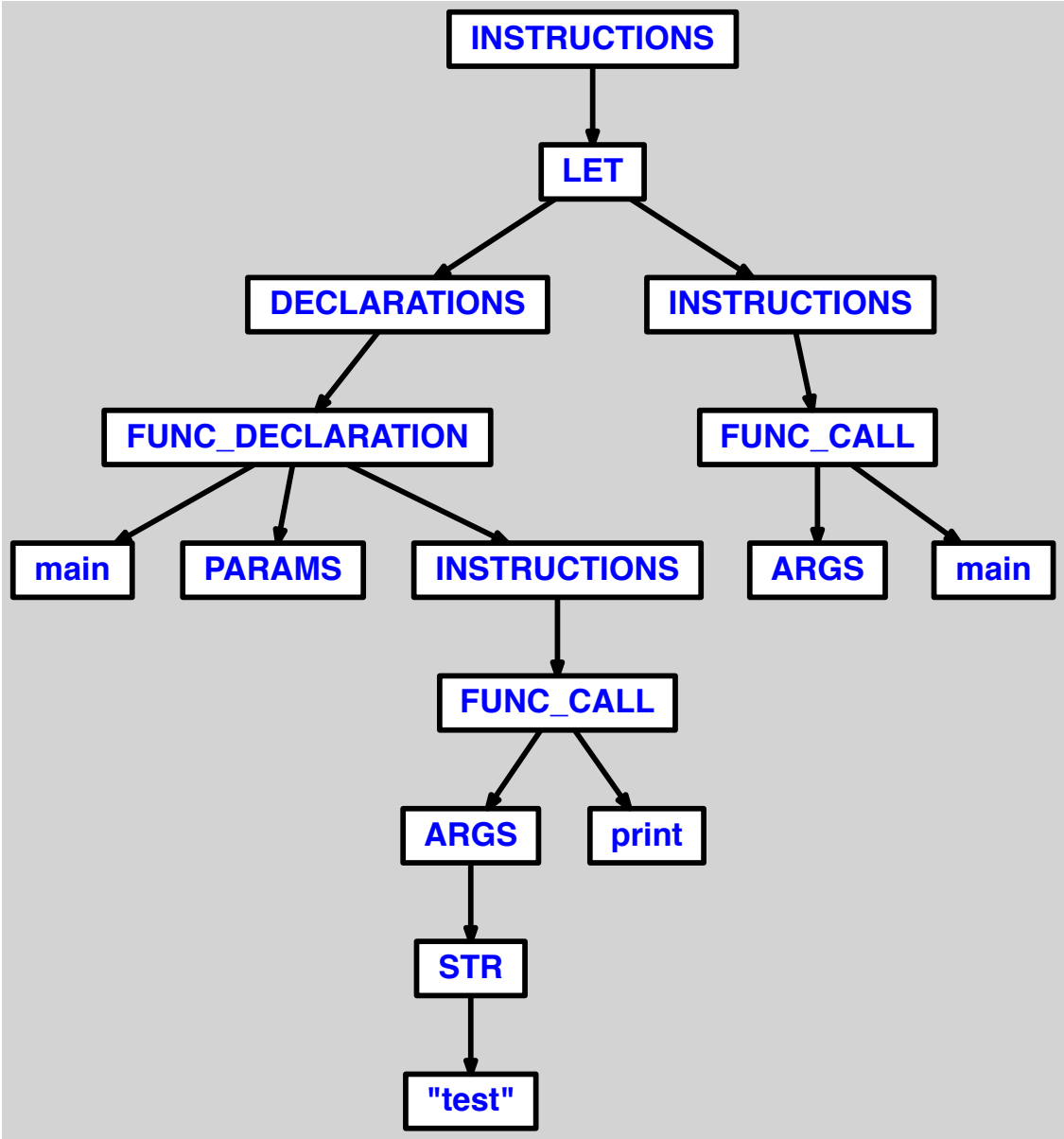
4.2.2 1 commentaire, sans instruction avant, sur plusieurs lignes

```
/*
1
2
3
*/

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

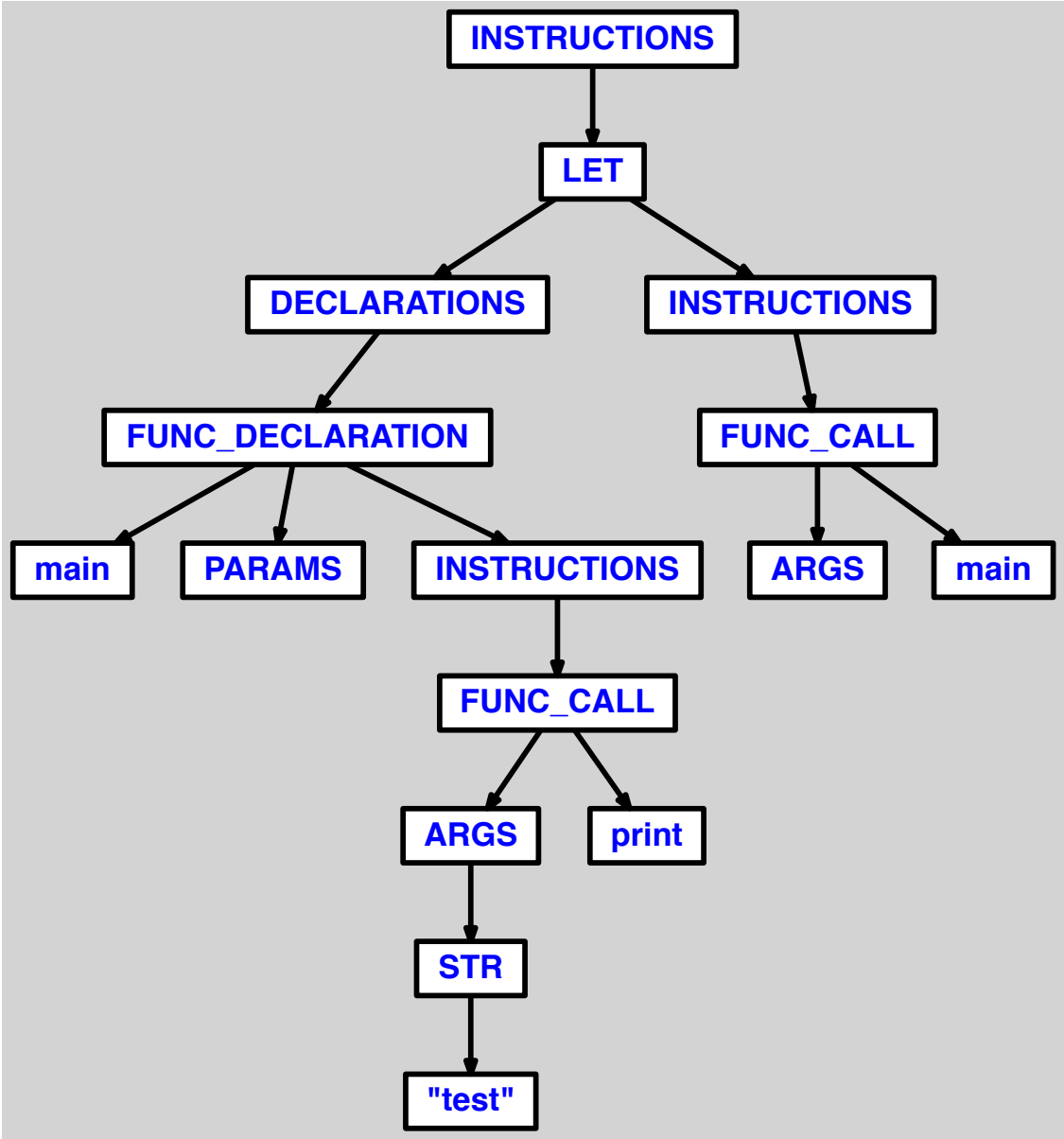
4.2.3 1 commentaire, apres instruction, sur 1 ligne

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

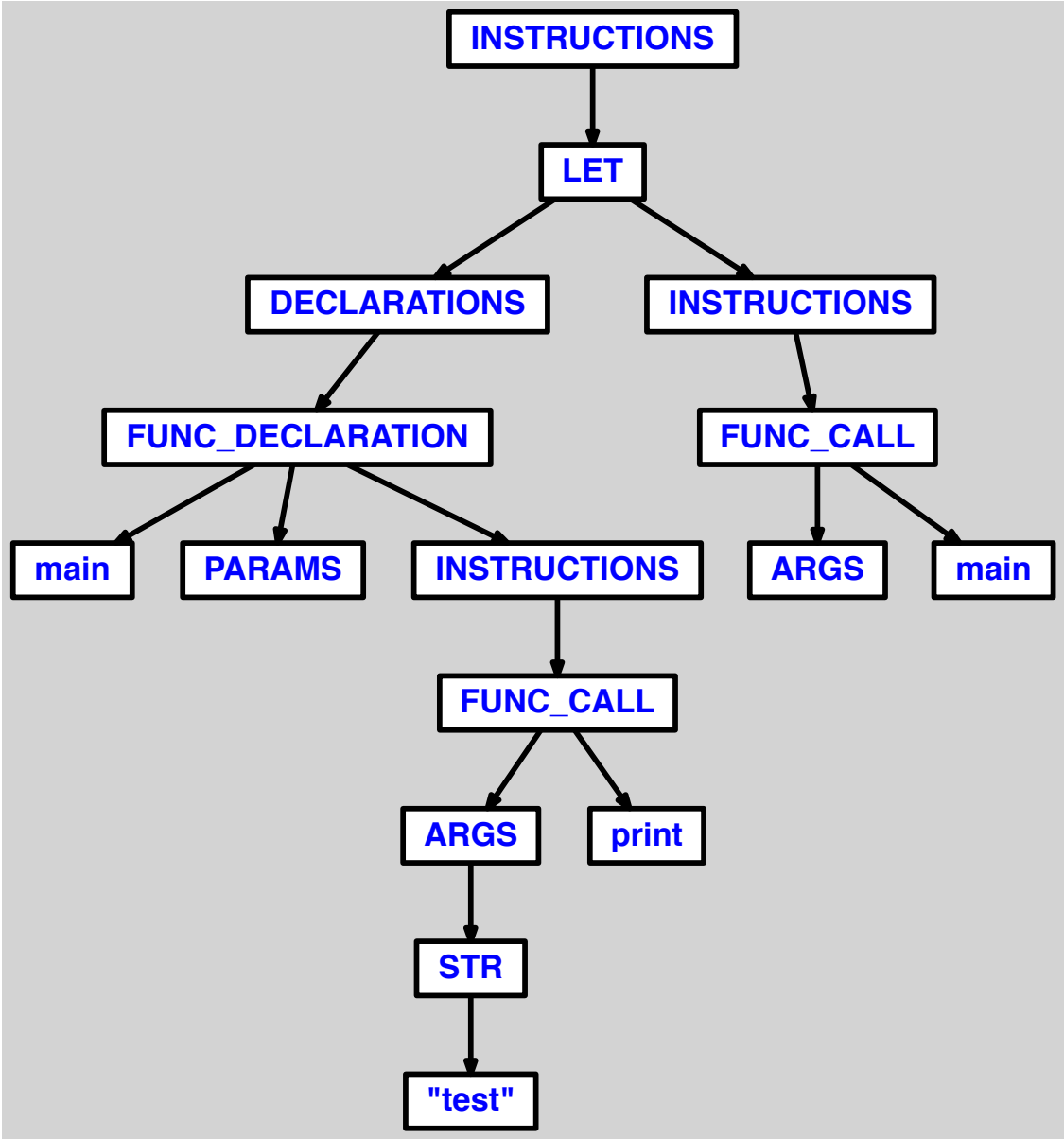
4.2.4 1 commentaire, apres instruction, sur plusieurs lignes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

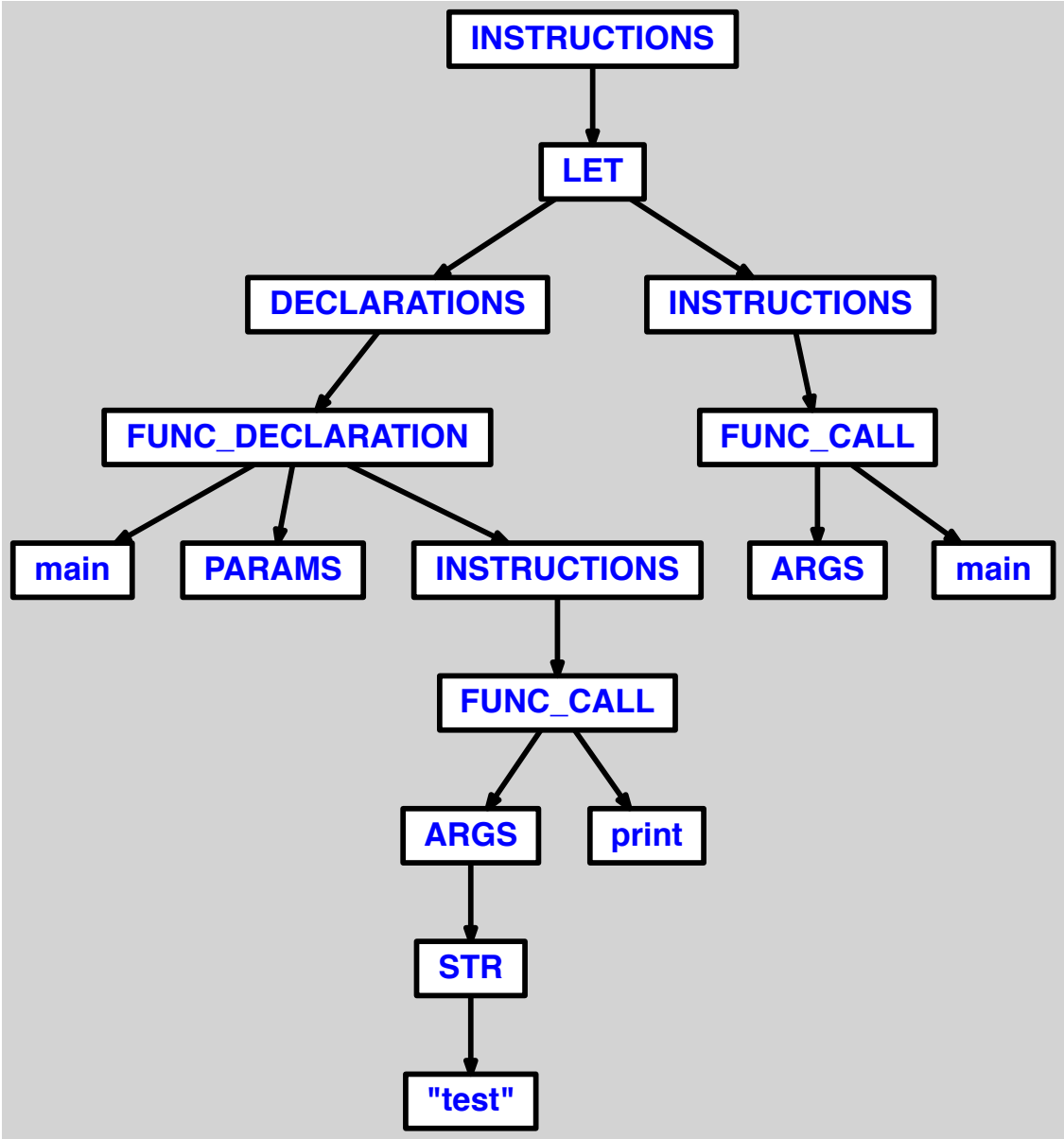
4.2.5 2 commentaires, sans instruction avant, sur 1 ligne

```
/* commentaire 1 */

/* commentaire 2 */

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

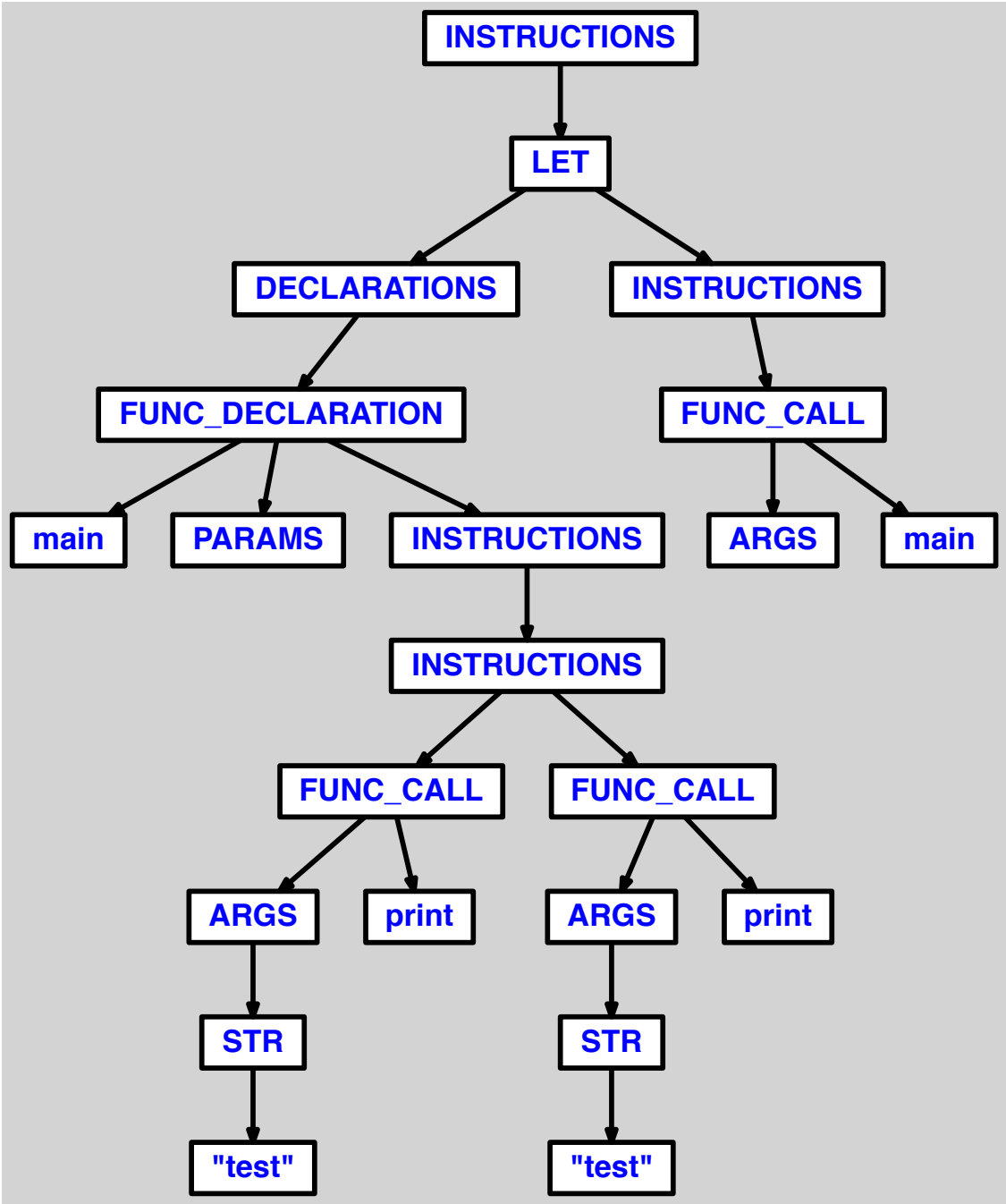
4.2.6 2 commentaires, sans instruction avant, sur plusieurs lignes

```
/*
1
2
3
*/

/*
4
5
6
*/

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

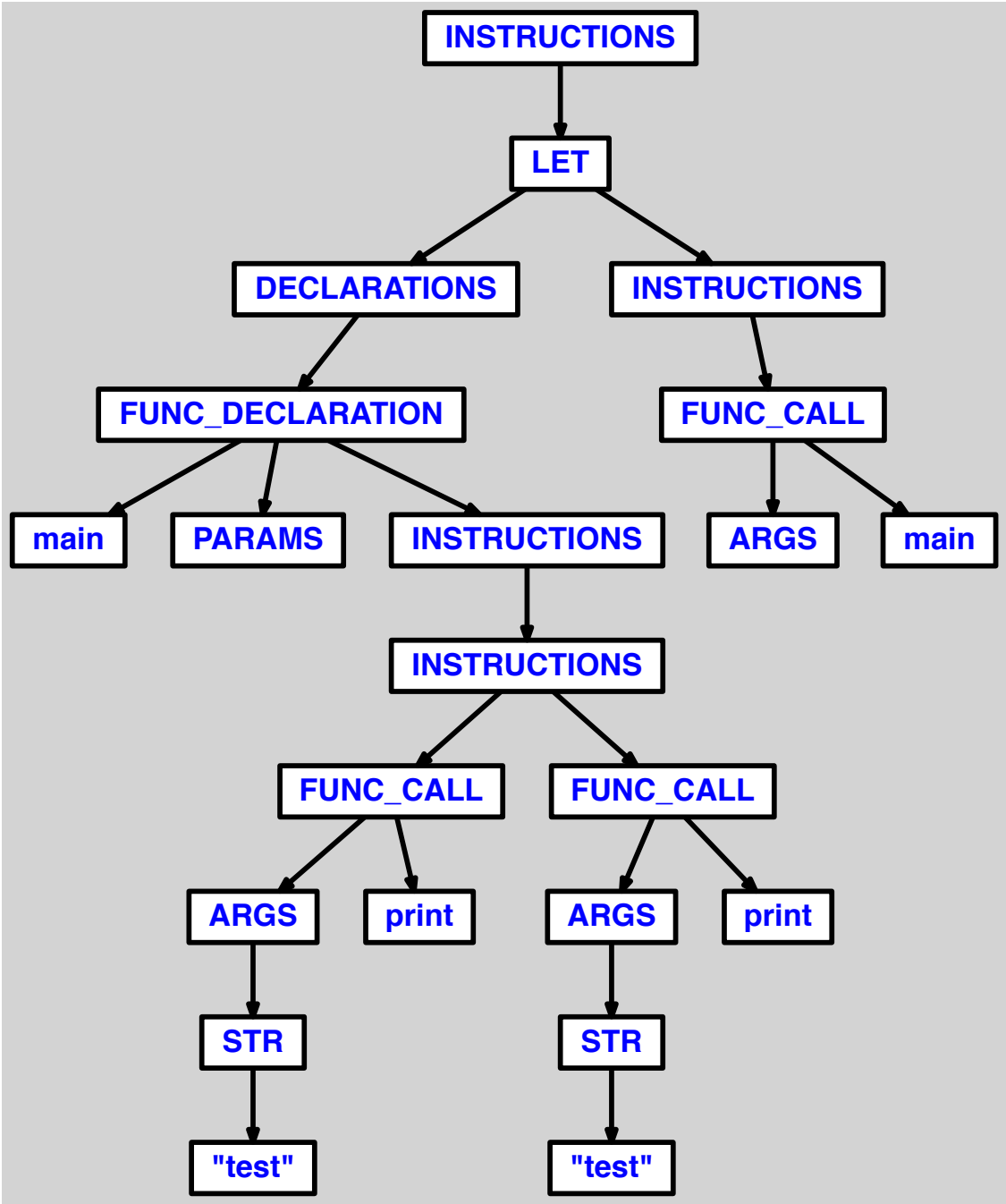
4.2.7 2 commentaires, apres instructions, sur 1 ligne

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
STR
STR
"test"
"test"

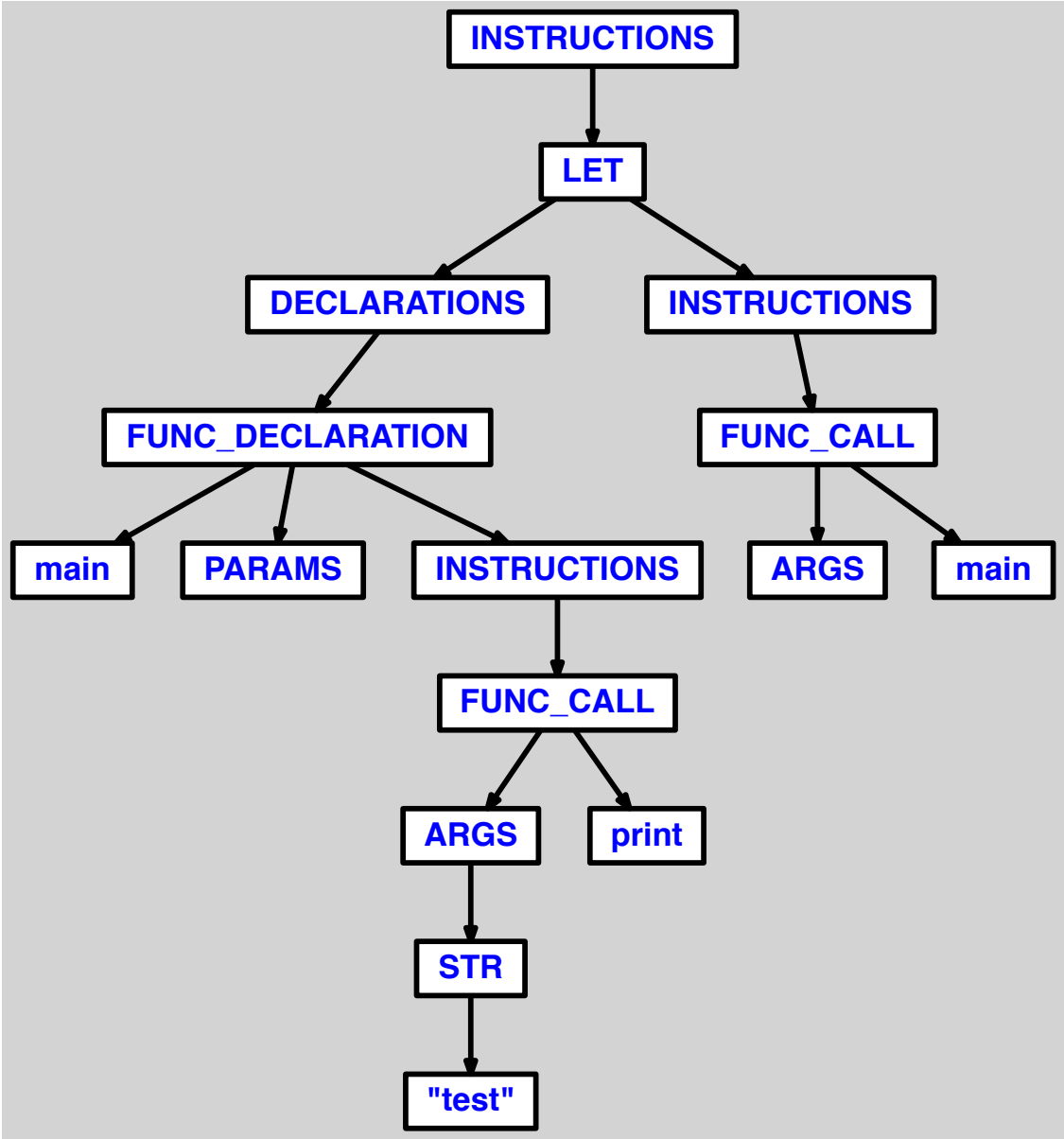
4.2.8 2 commentaires, apres instructions, sur plusieurs lignes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
STR
STR
"test"
"test"

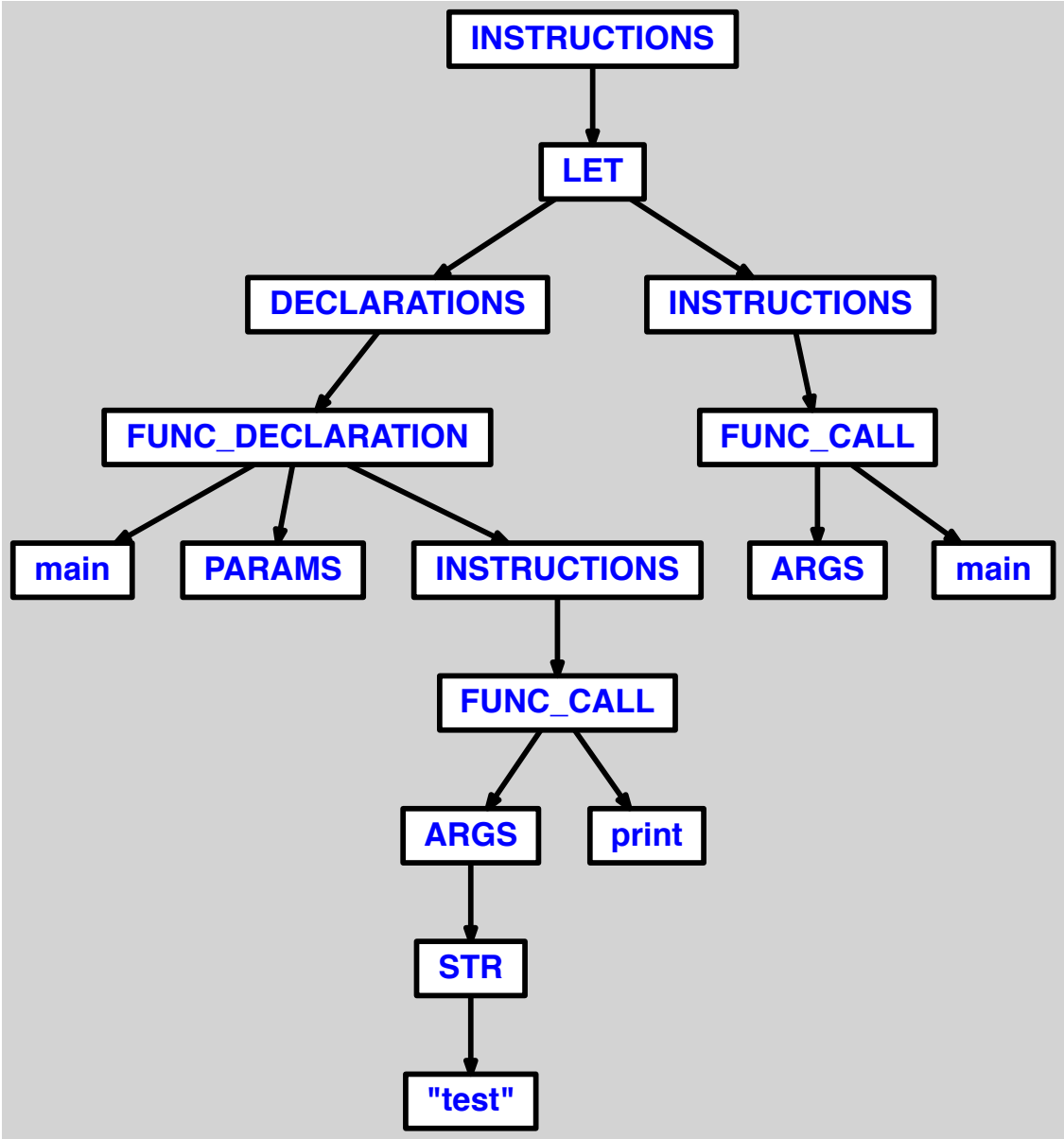
4.2.9 3 commentaires, sans instruction avant, sur 1 ligne

```
/* commentaire 1 */

/* commentaire 2 */

/* commentaire 3 */

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

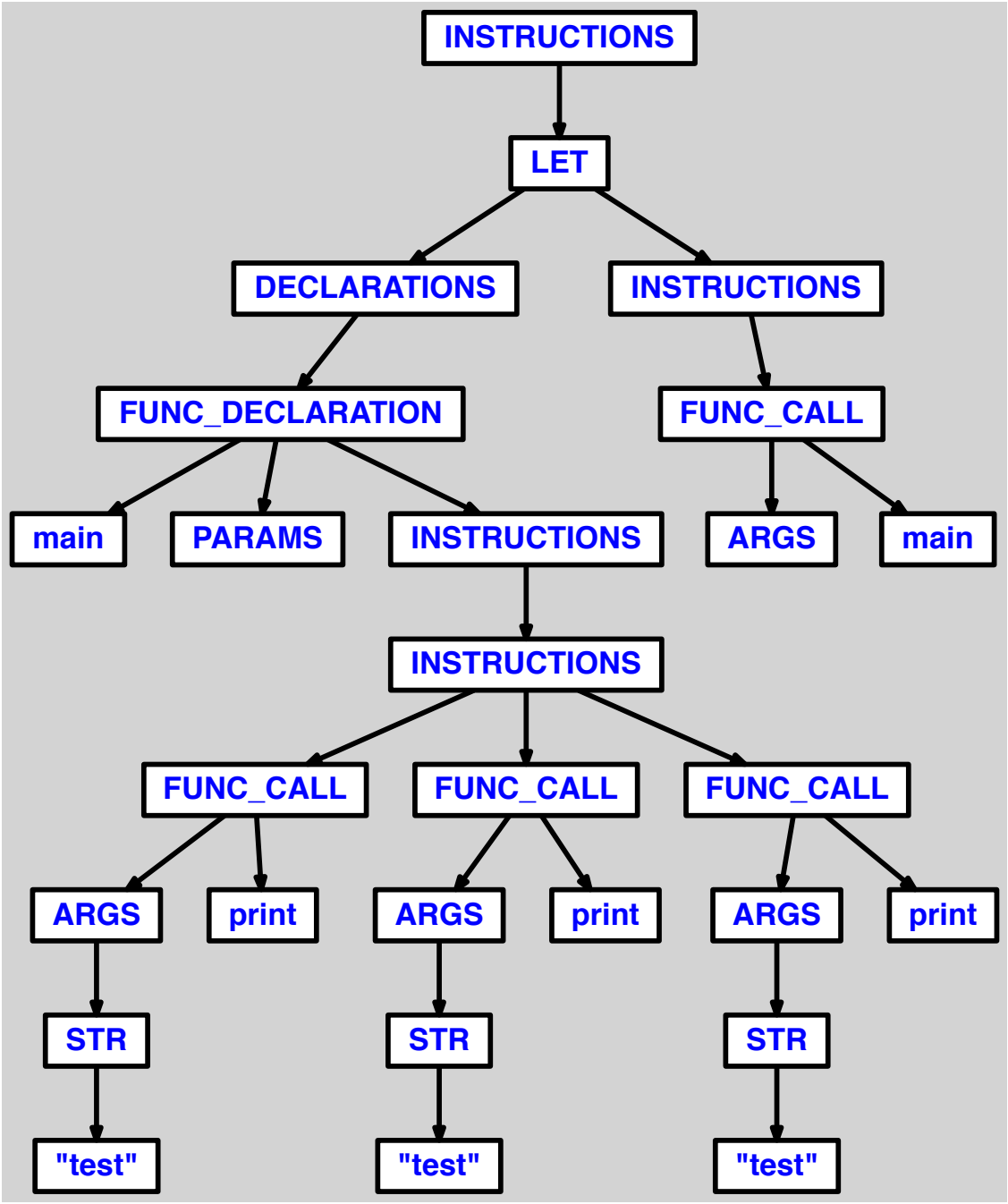
4.2.10 3 commentaires, sans instruction avant, sur plusieurs lignes

```
/*
1
2
3
*/

/*
4
5
6
*/

/*
7
8
9
*/

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

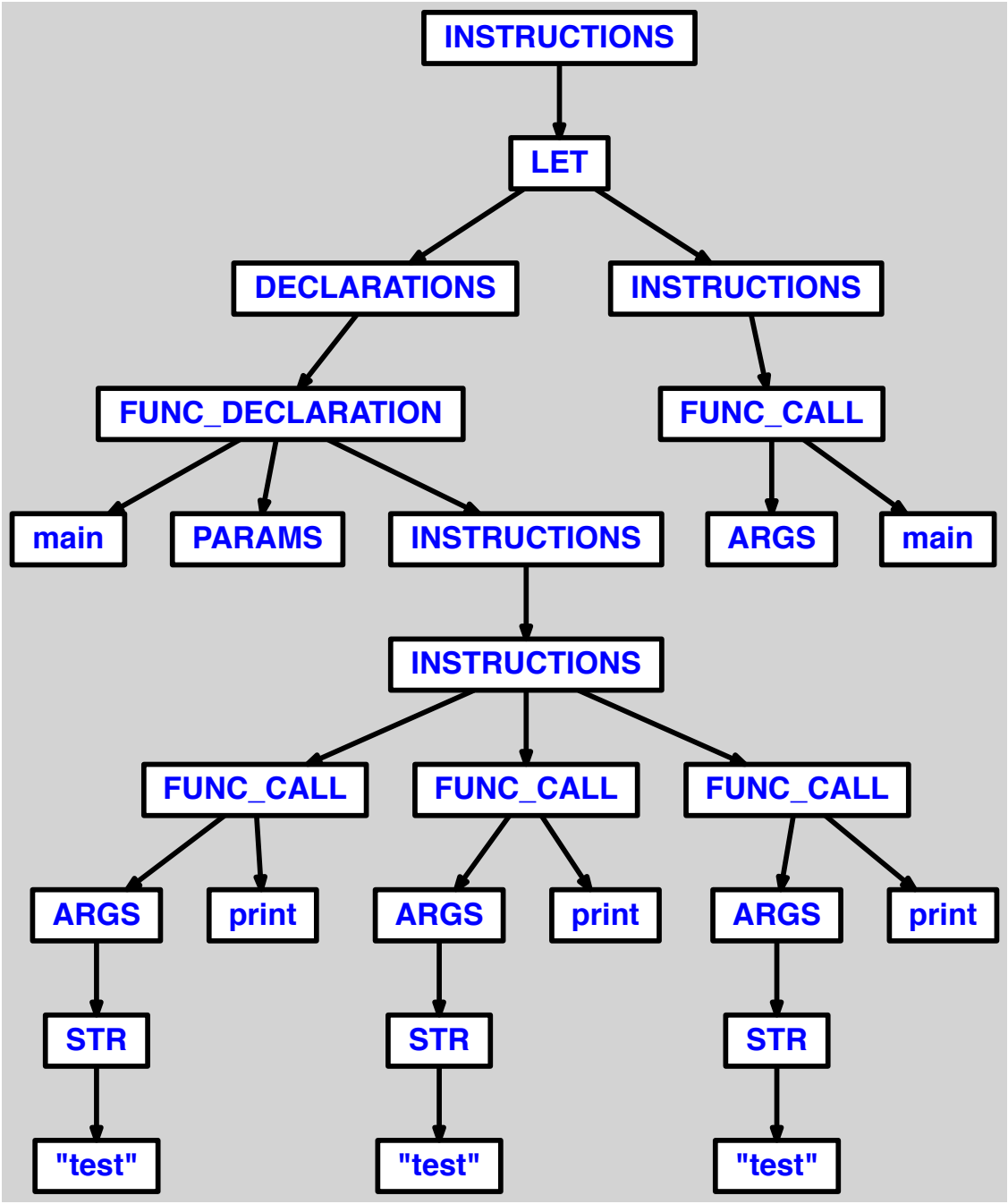
4.2.11 3 commentaires, apres instructions, sur 1 ligne

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
ARGS
print
STR
STR
STR
"test"
"test"
"test"

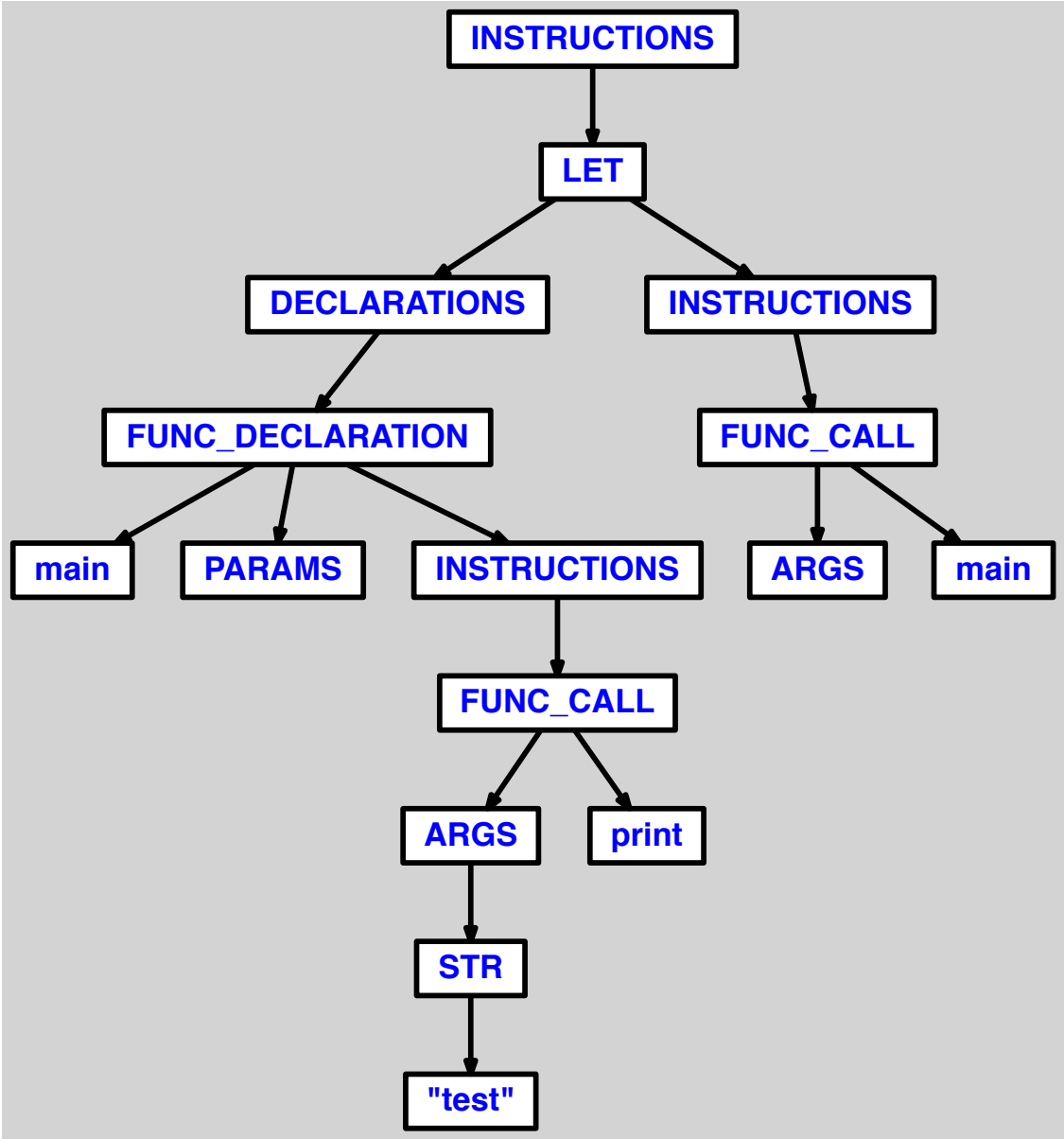
4.2.12 3 commentaires, apres instructions, sur plusieurs lignes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
ARGS
print
STR
STR
STR
"test"
"test"
"test"

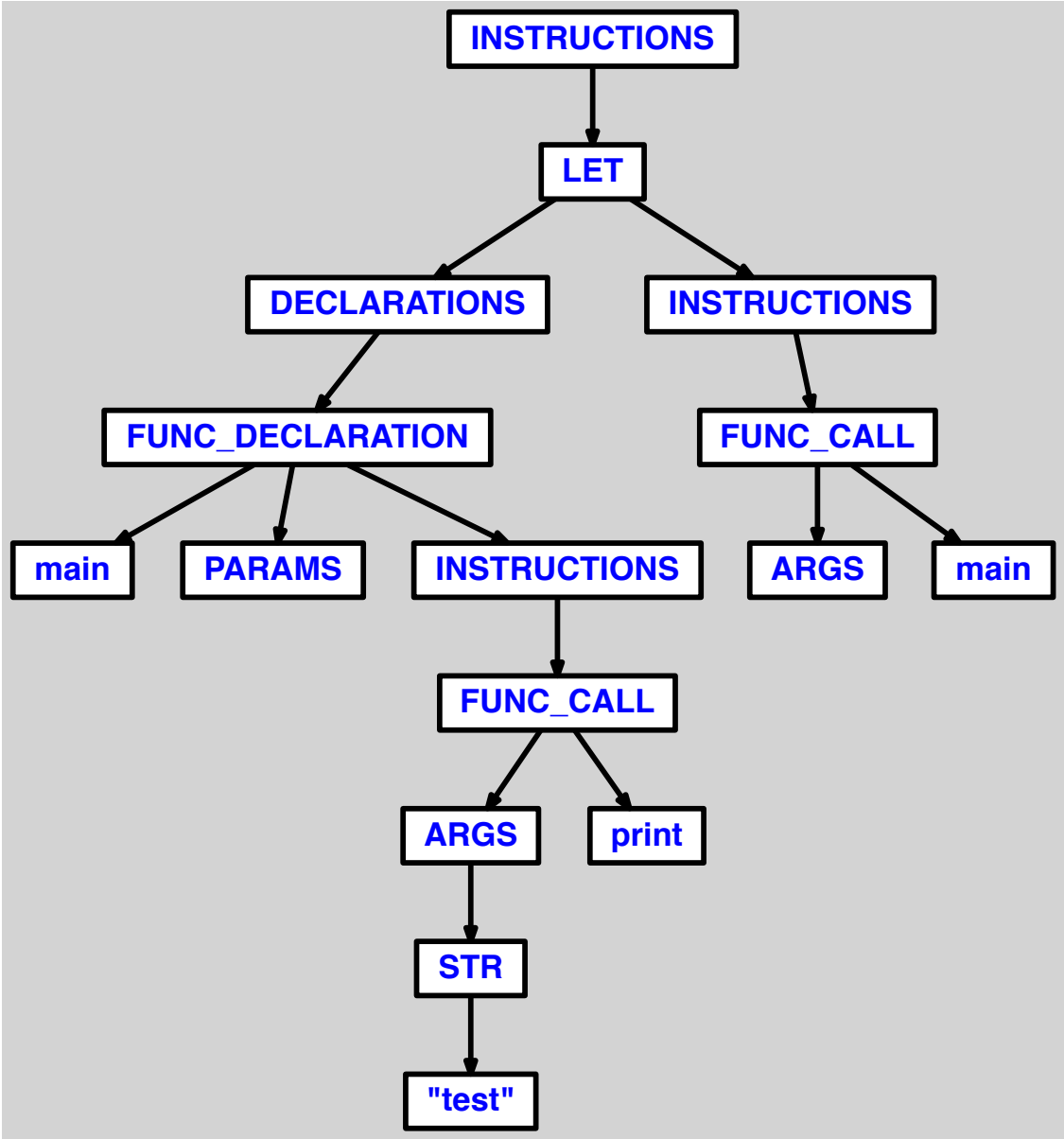
4.2.13 commentaire sans contenu sur 1 ligne

```
/* */

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

4.2.14 commentaire sans contenu sur plusieurs lignes

```
/*



*/

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

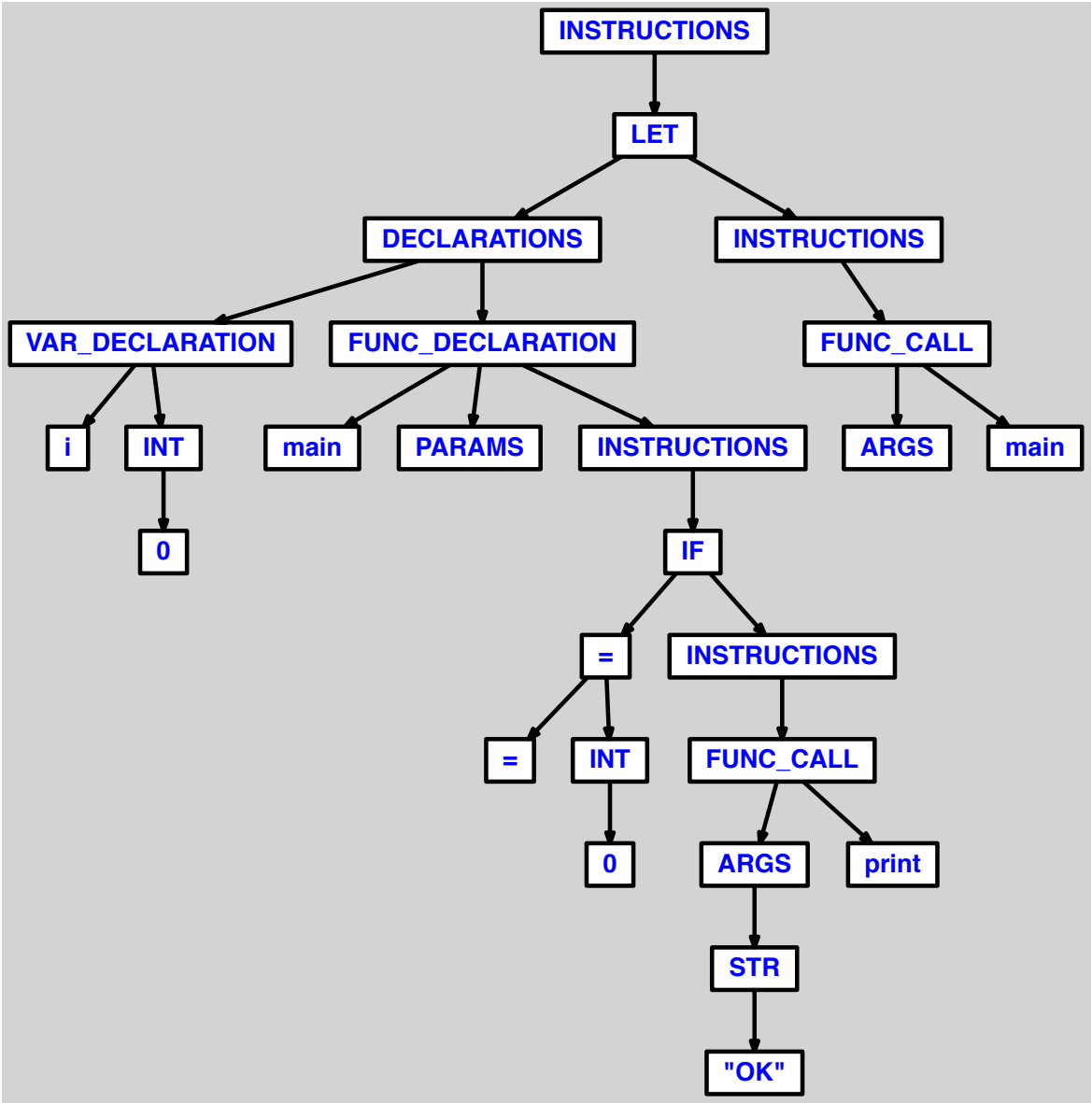

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

5 comparaison

5.1 KO

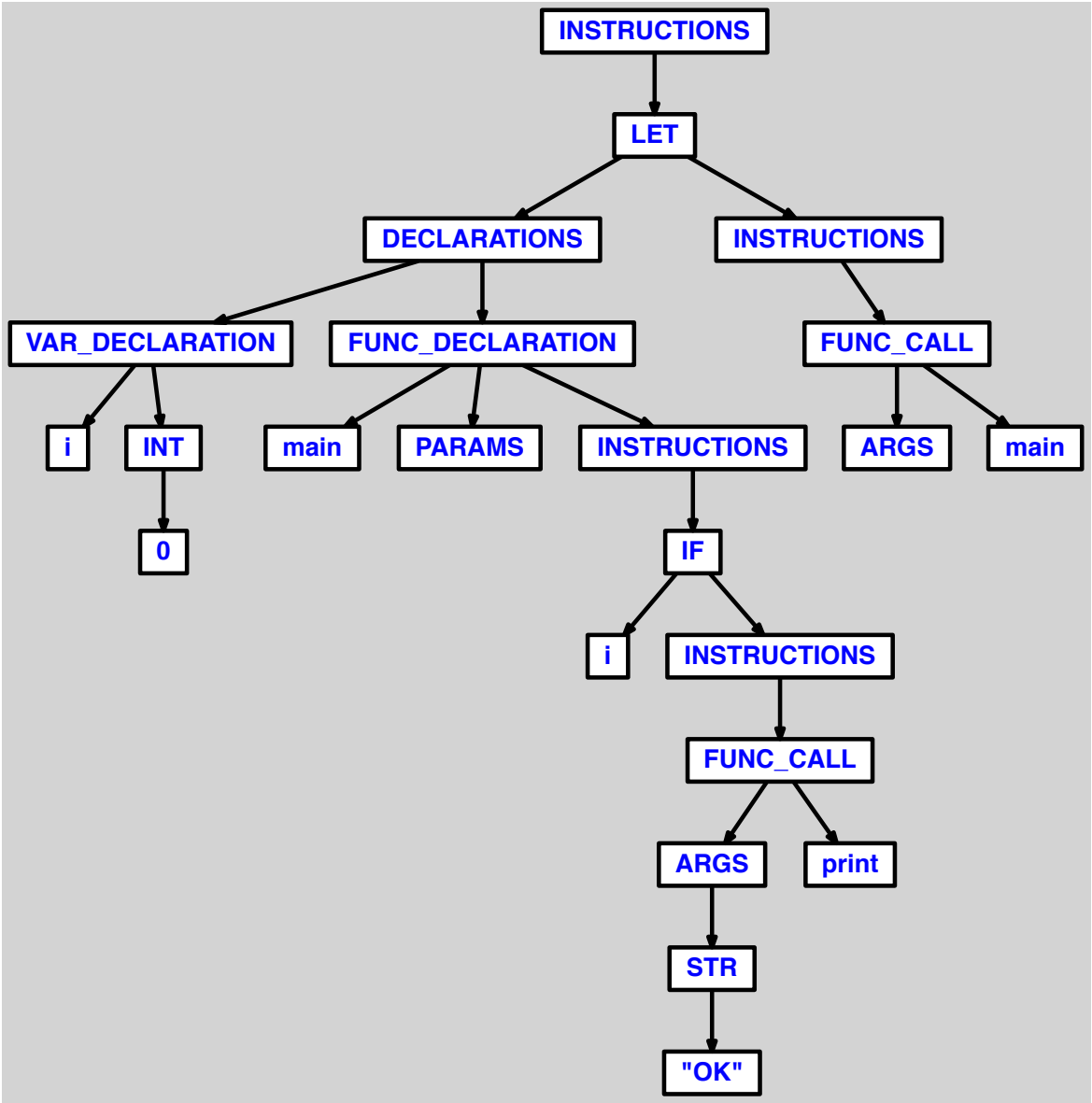
5.1.1 comparaison avec oubli d'operande gauche

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
=
INSTRUCTIONS
=
INT
FUNC_CALL
0
ARGS
print
STR
"OK"

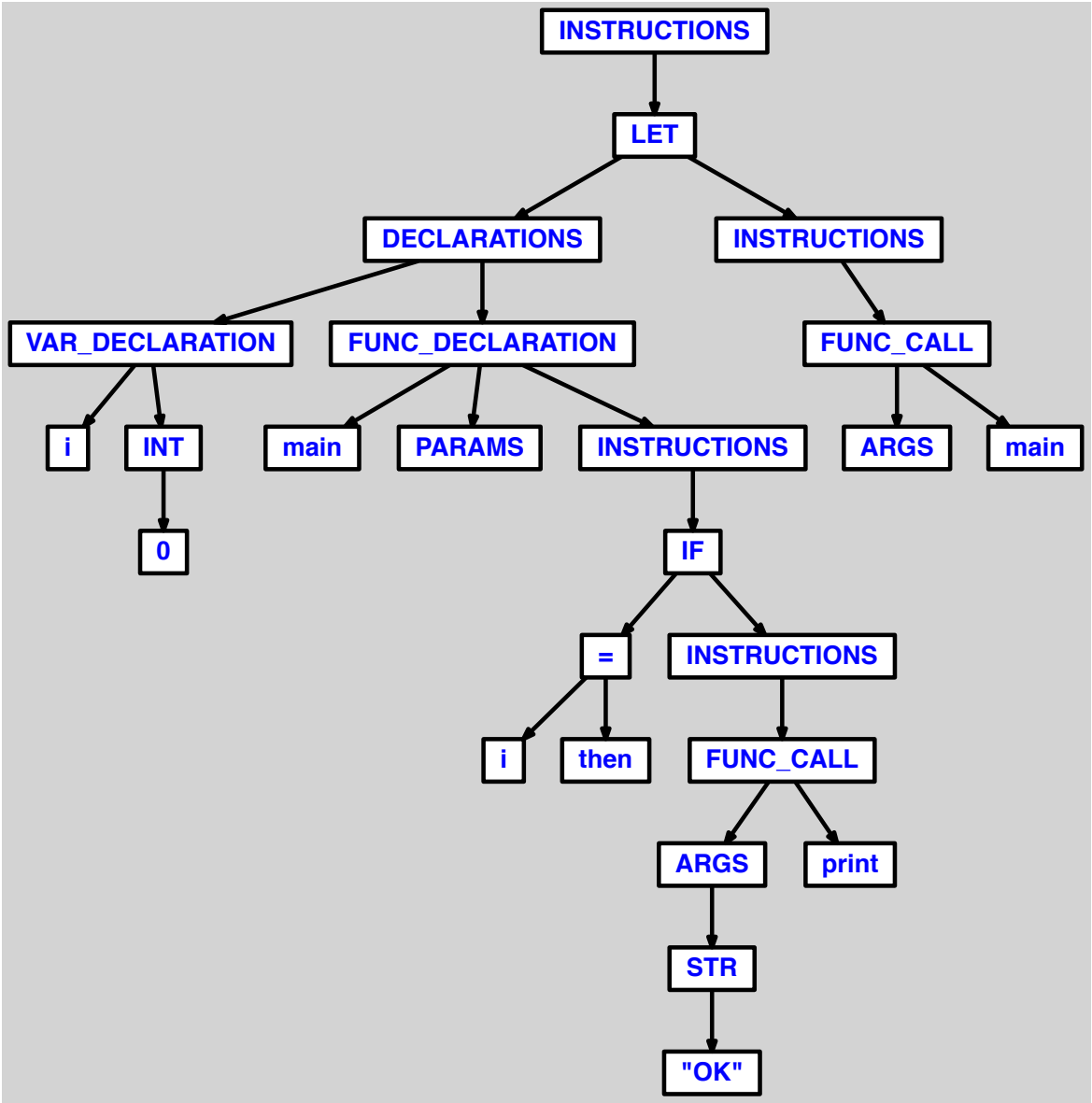
5.1.2 comparaison avec oubli d'operateur

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
i
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"OK"

5.1.3 comparaison avec oubli d'operateur droit

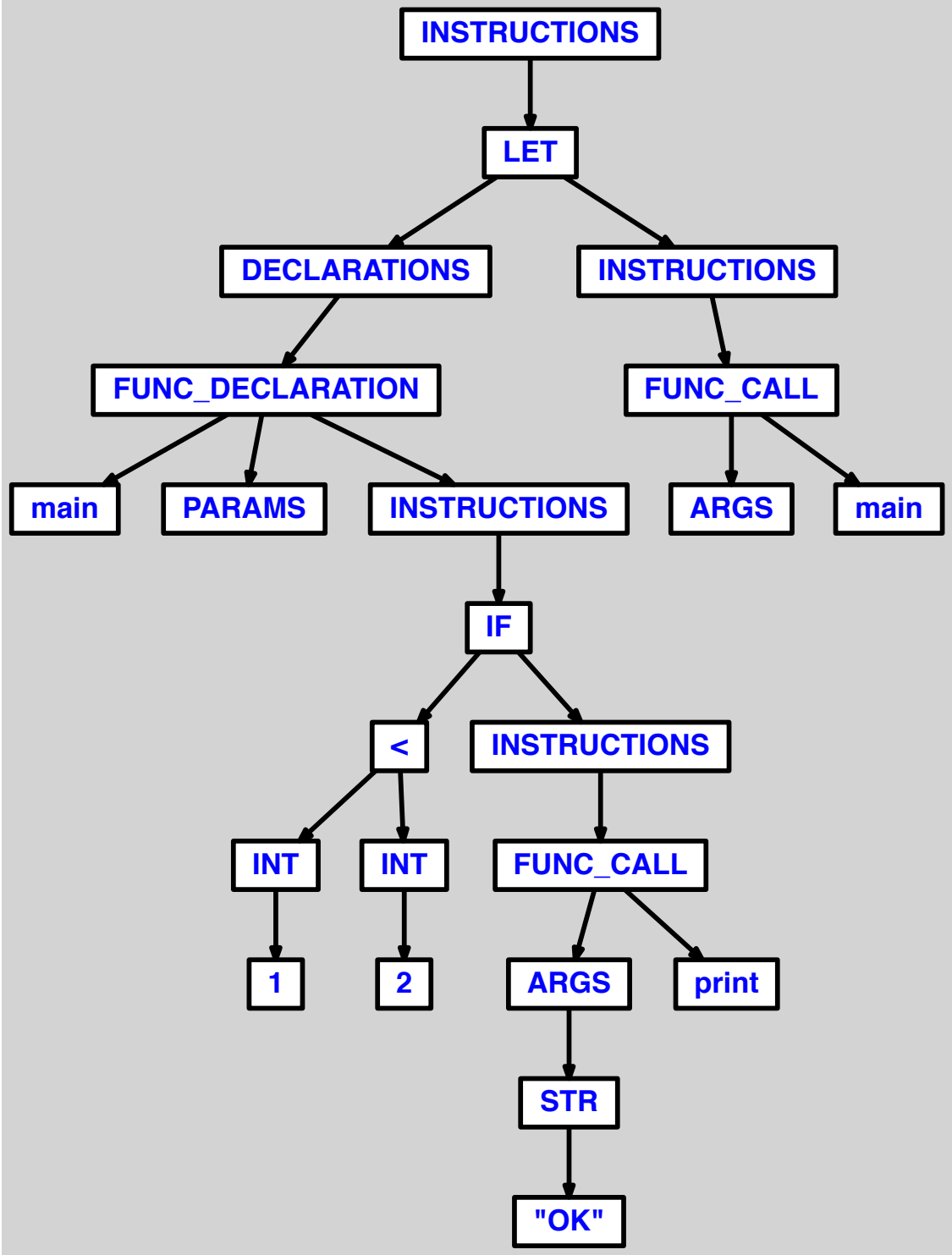
```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
=
INSTRUCTIONS
i
then
FUNC_CALL
ARGS
print
STR
"OK"

5.2 OK

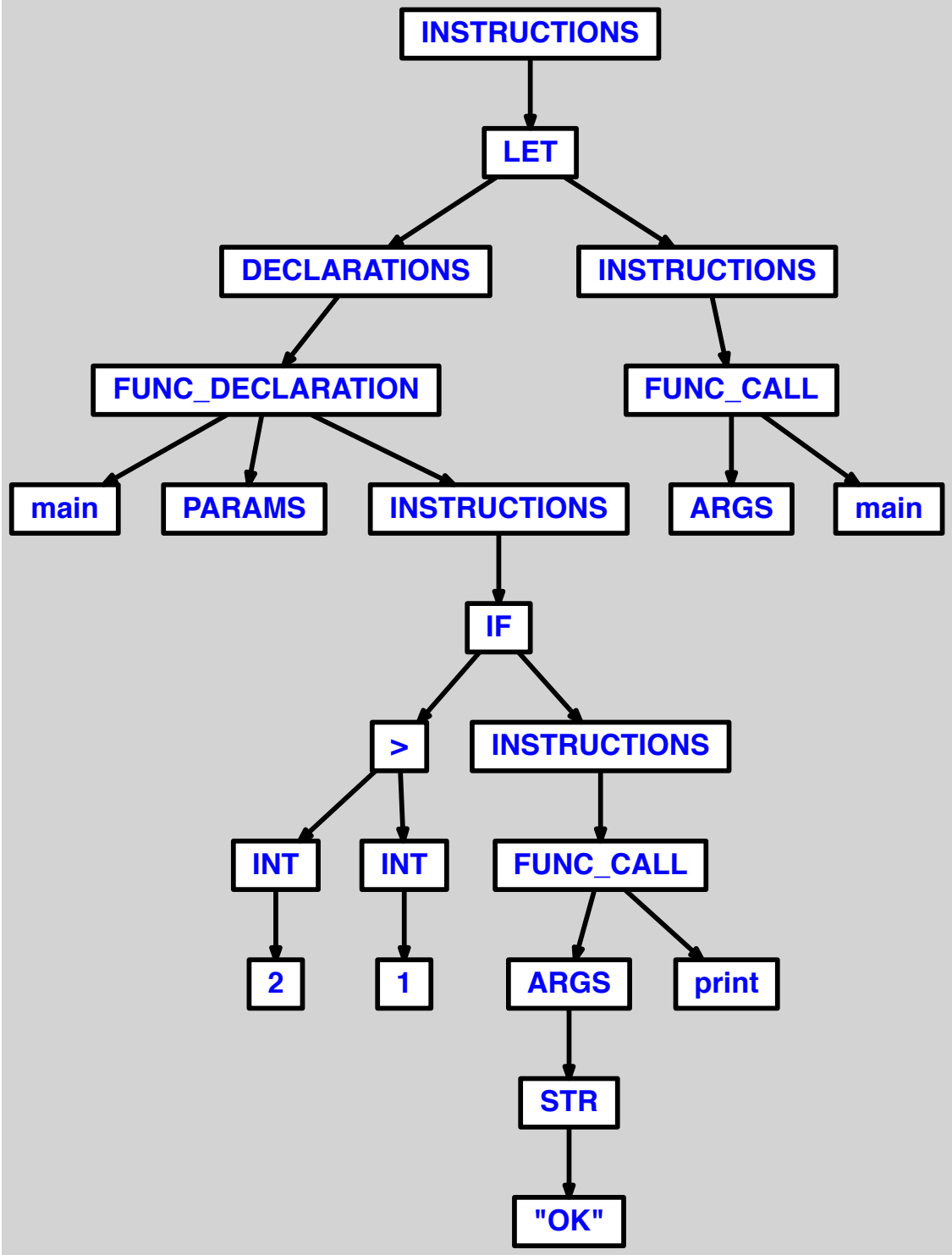
5.2.1 comparaison simple d'entiers avec <"<">

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
<
INSTRUCTIONS
INT
INT
FUNC_CALL
1
2
ARGS
print
STR
"OK"

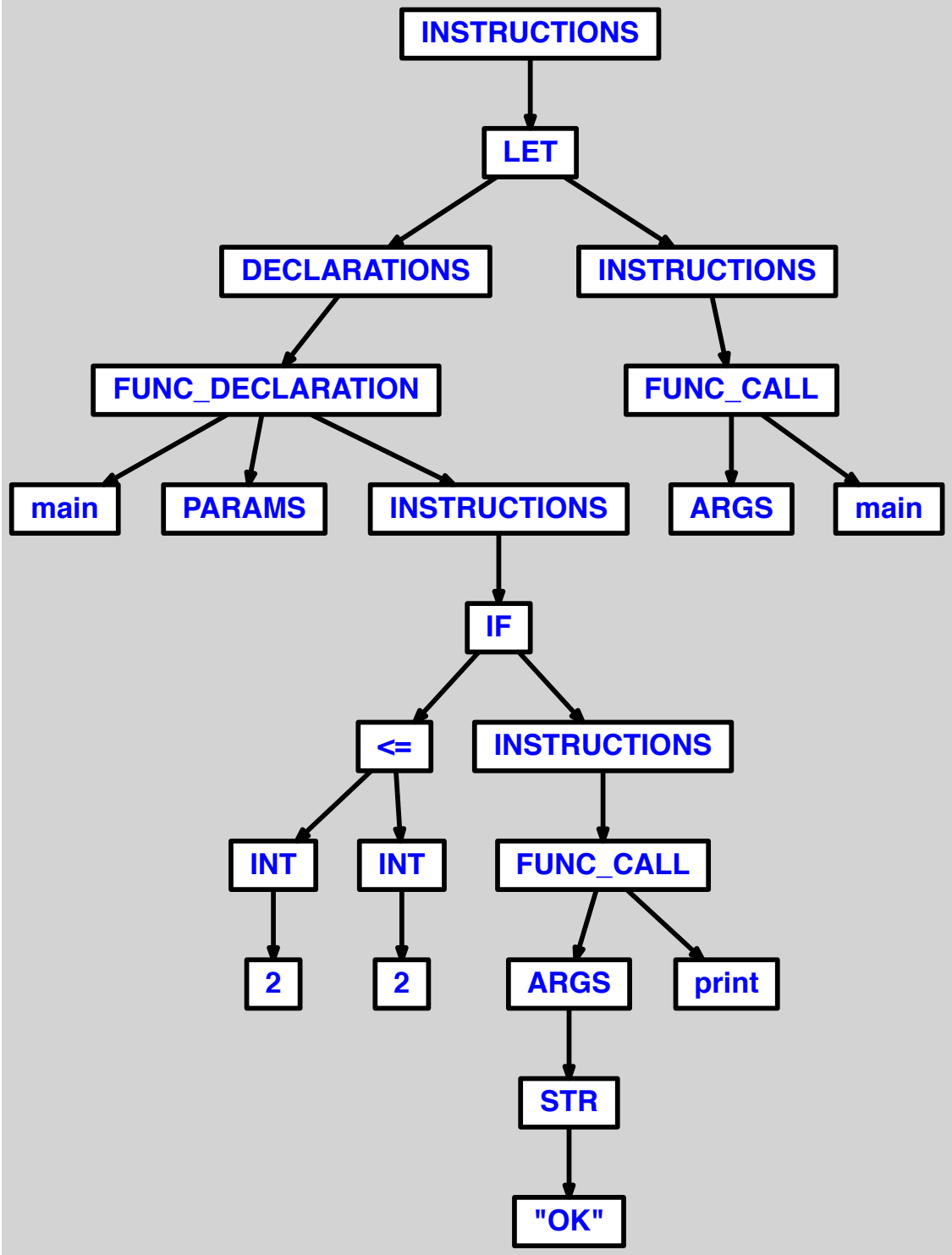
5.2.2 comparaison simple d'entiers avec <">">

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
>
INSTRUCTIONS
INT
INT
FUNC_CALL
2
1
ARGS
print
STR
"OK"

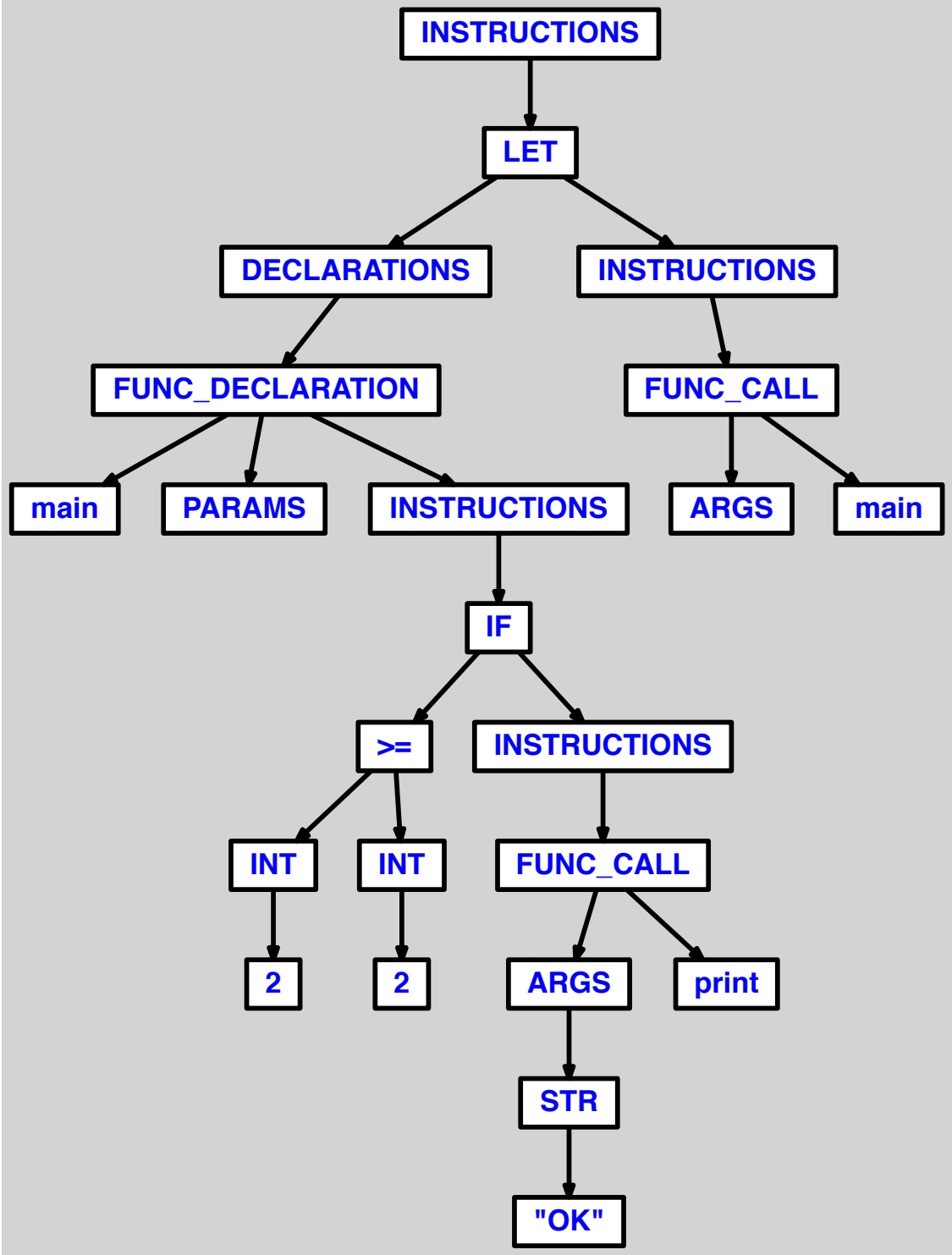
5.2.3 comparaison simple d'entiers avec <"<="><

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
<=
INSTRUCTIONS
INT
INT
FUNC_CALL
2
2
ARGS
print
STR
"OK"

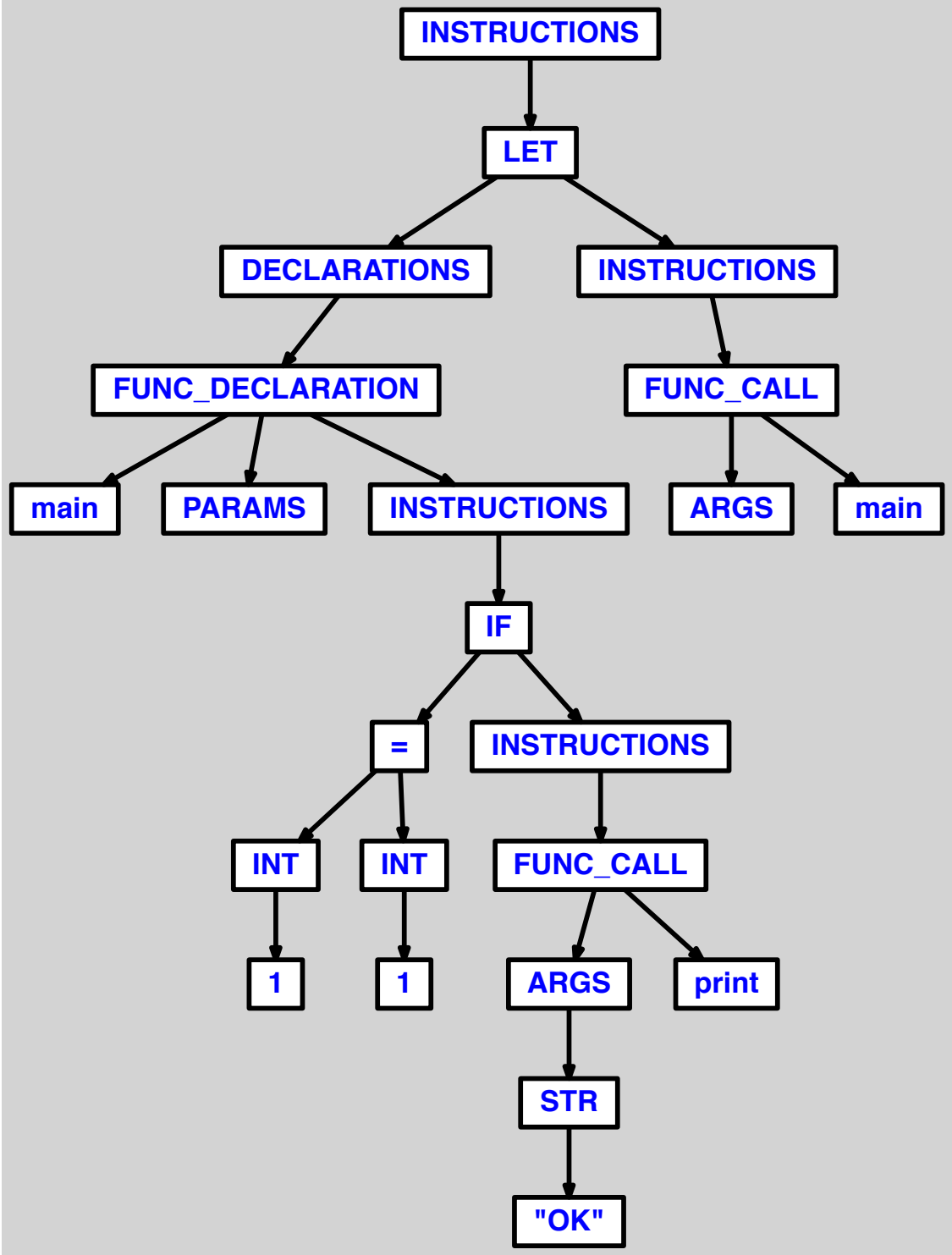
5.2.4 comparaison simple d'entiers avec <">="><

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
>=
INSTRUCTIONS
INT
INT
FUNC_CALL
2
2
ARGS
print
STR
"OK"

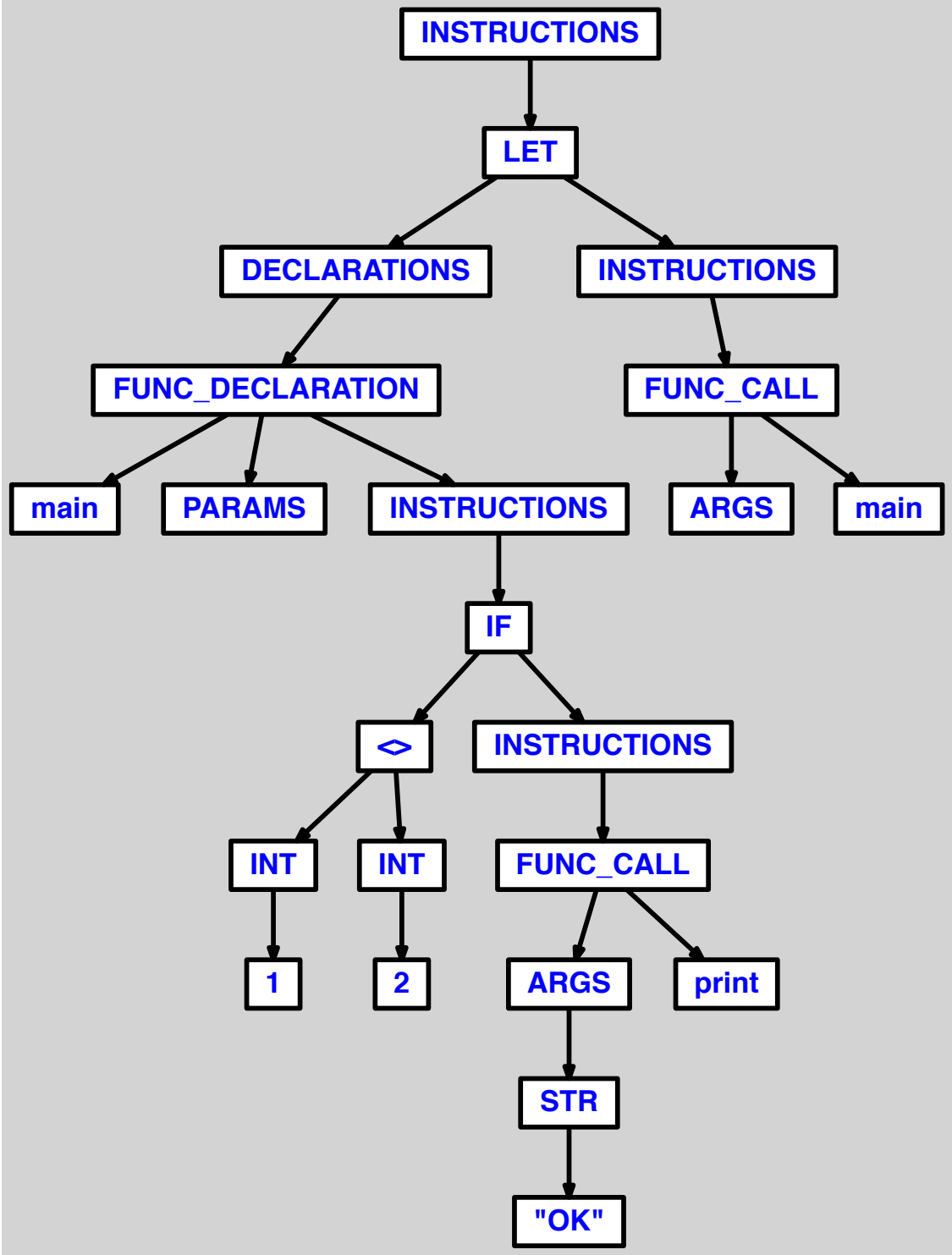
5.2.5 comparaison simple d'entiers avec <=>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
=
INSTRUCTIONS
INT
INT
FUNC_CALL
1
1
ARGS
print
STR
"OK"

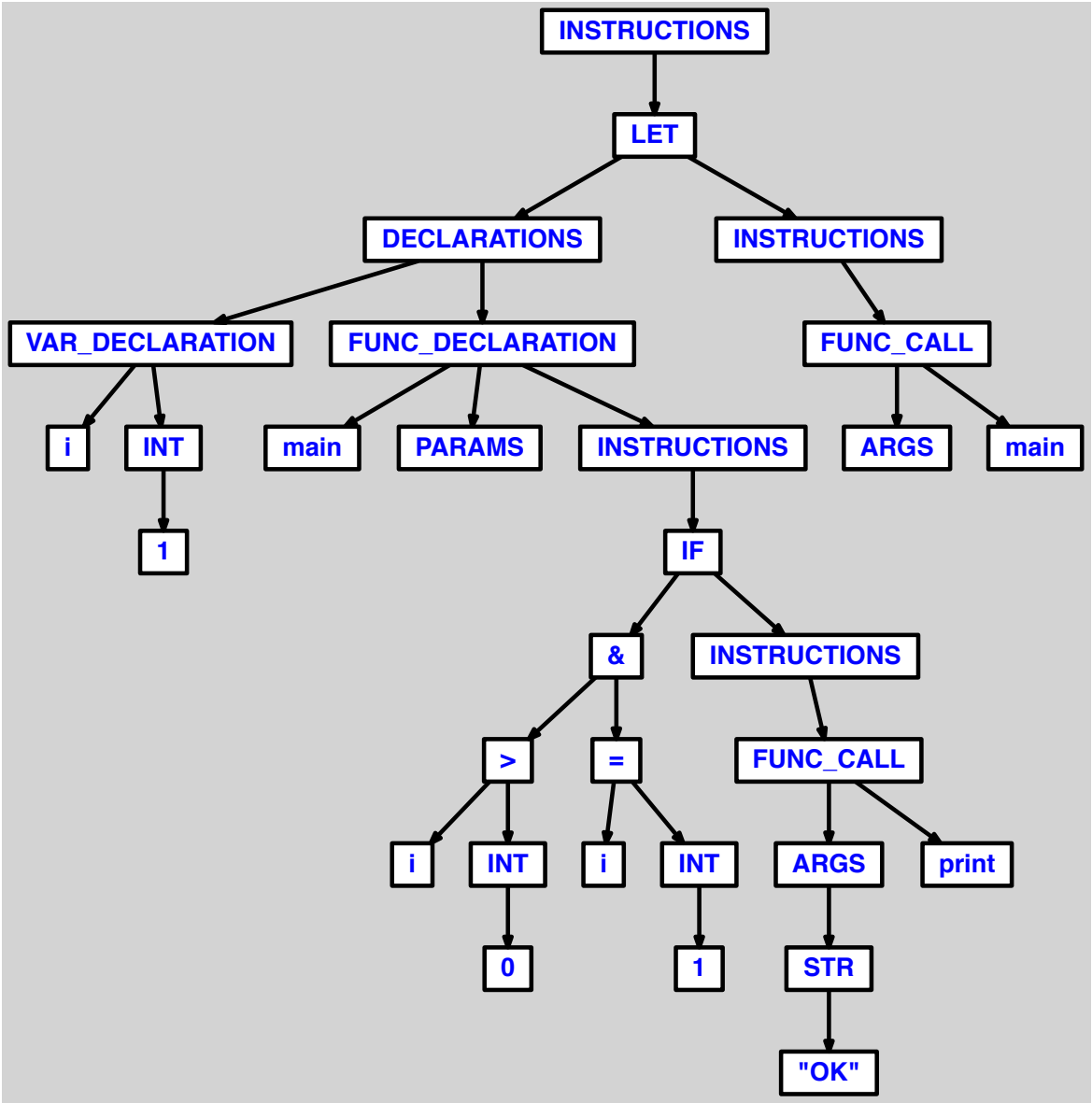
5.2.6 comparaison simple d'entiers avec <"<>">

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
<>
INSTRUCTIONS
INT
INT
FUNC_CALL
1
2
ARGS
print
STR
"OK"

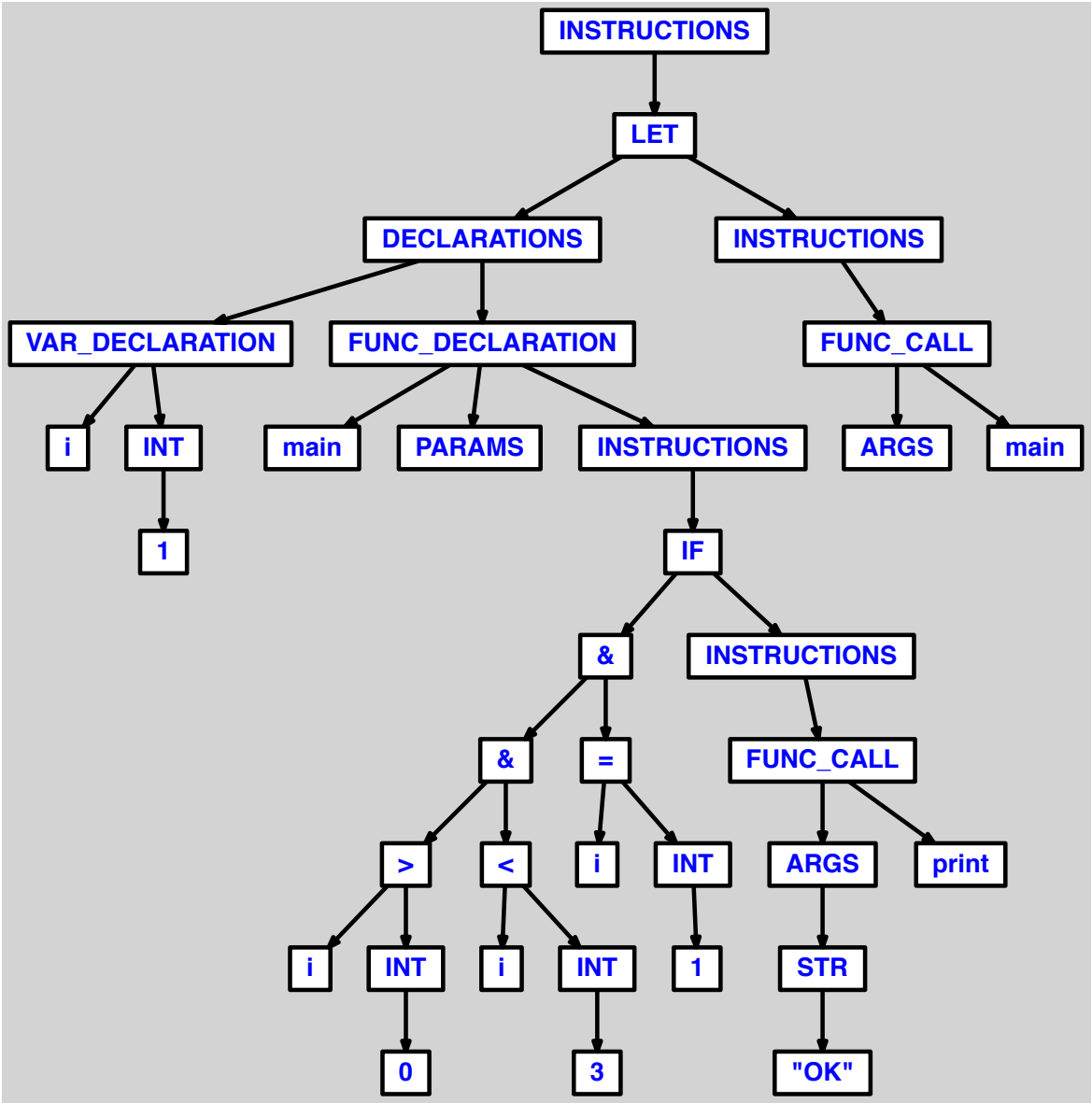
5.2.7 comparaison double d'entiers

```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
&
INSTRUCTIONS
>
=
FUNC_CALL
i
INT
i
INT
ARGS
print
0
1
STR
"OK"

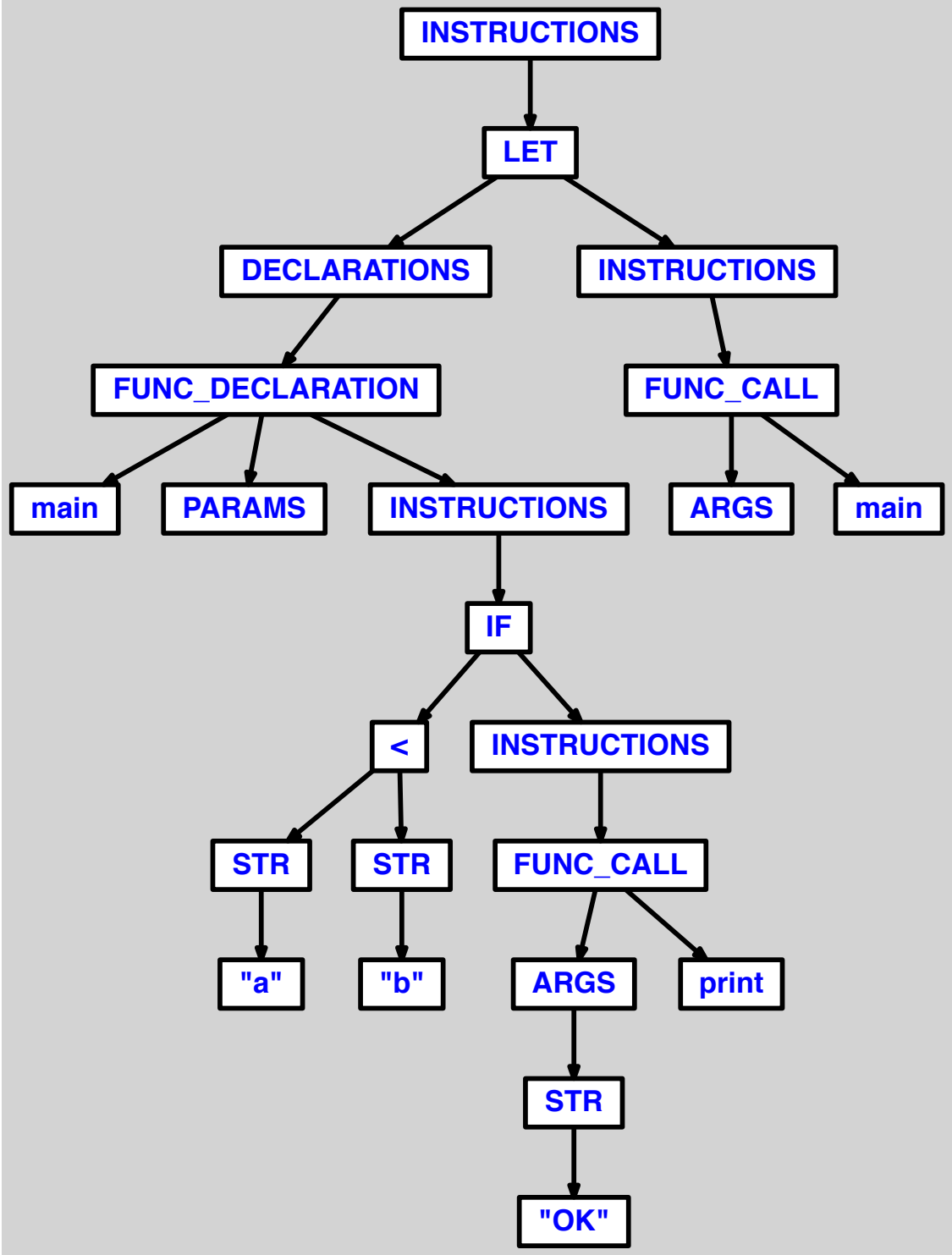
5.2.8 comparaison triple d'entiers

```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
&
INSTRUCTIONS
&
=
FUNC_CALL
>
<
i
INT
ARGS
print
i
INT
i
INT
1
STR
0
3
"OK"

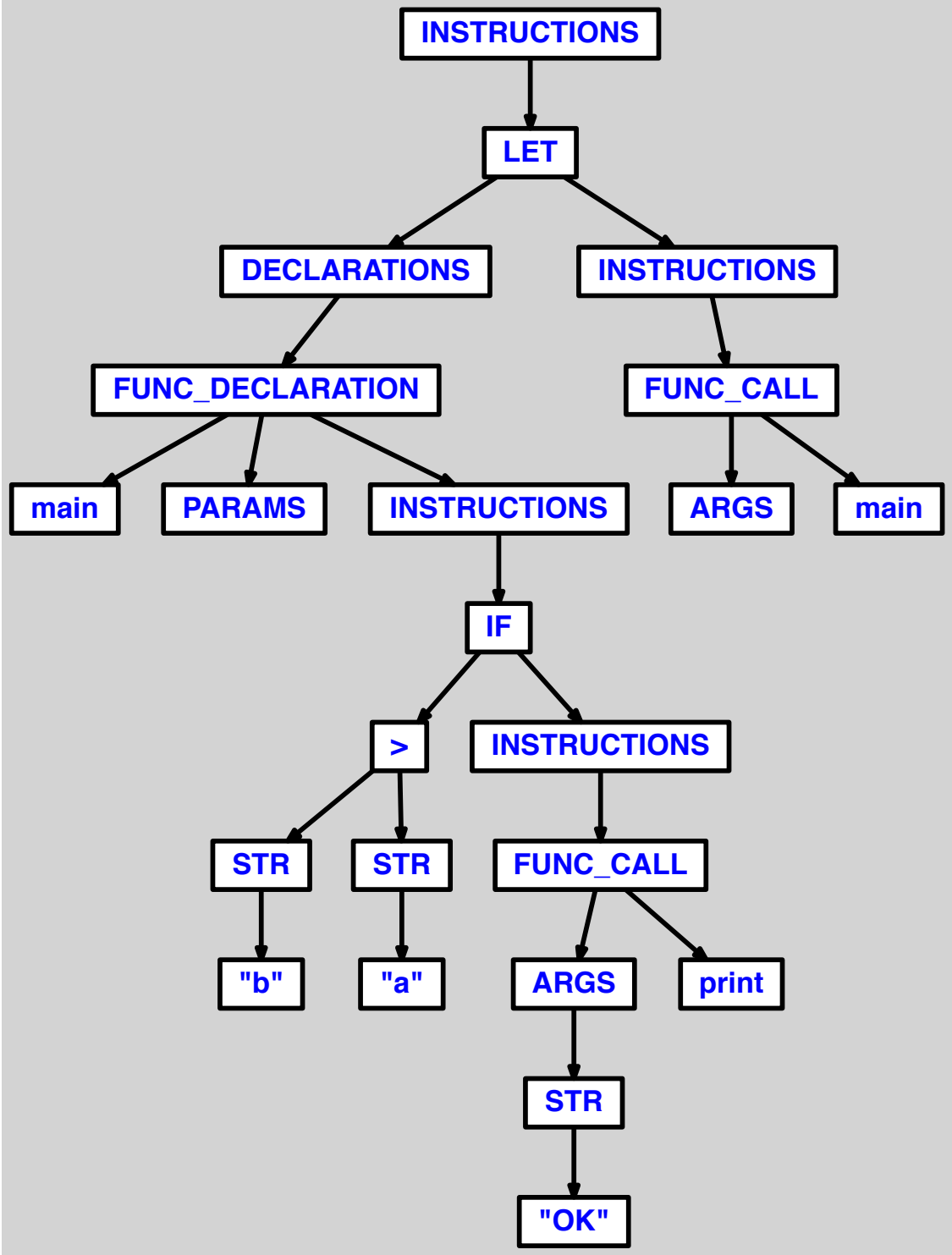
5.2.9 comparaison simple de chaines avec <"<">

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
<
INSTRUCTIONS
STR
STR
FUNC_CALL
"a"
"b"
ARGS
print
STR
"OK"

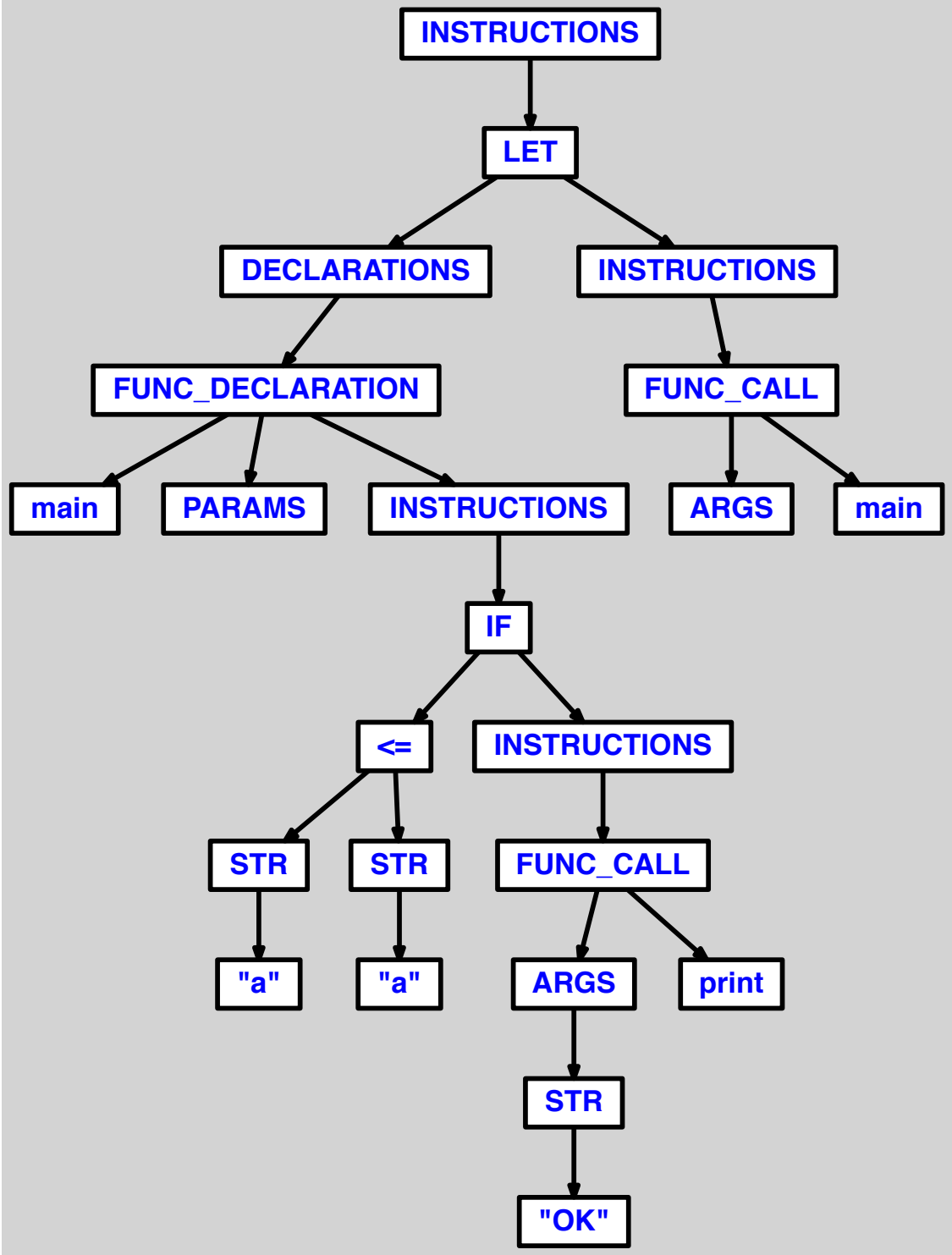
5.2.10 comparaison simple de chaines avec <">">

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
>
INSTRUCTIONS
STR
STR
FUNC_CALL
"b"
"a"
ARGS
print
STR
"OK"

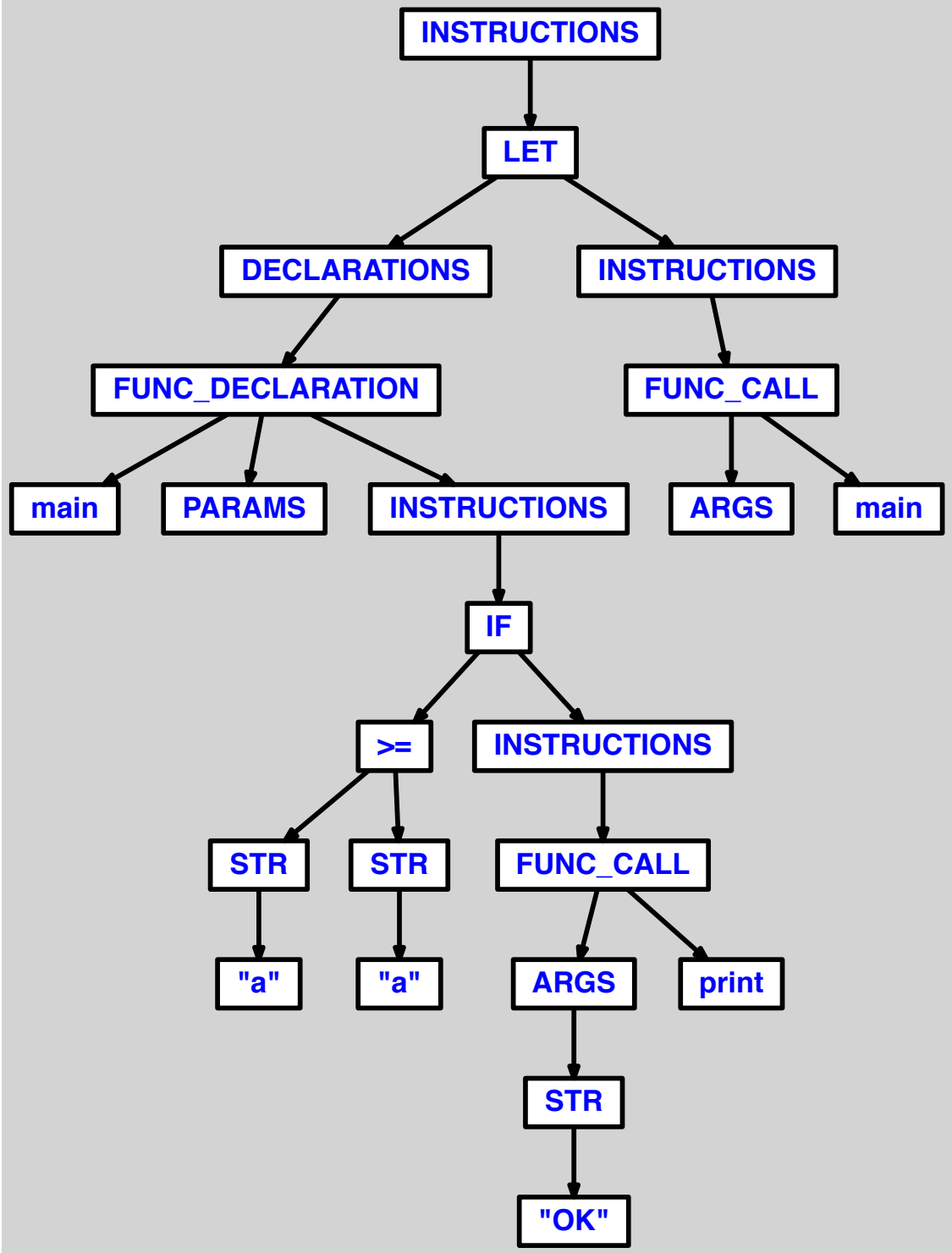
5.2.11 comparaison simple de chaines avec <"<="><

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
<=
INSTRUCTIONS
STR
STR
FUNC_CALL
"a"
"a"
ARGS
print
STR
"OK"

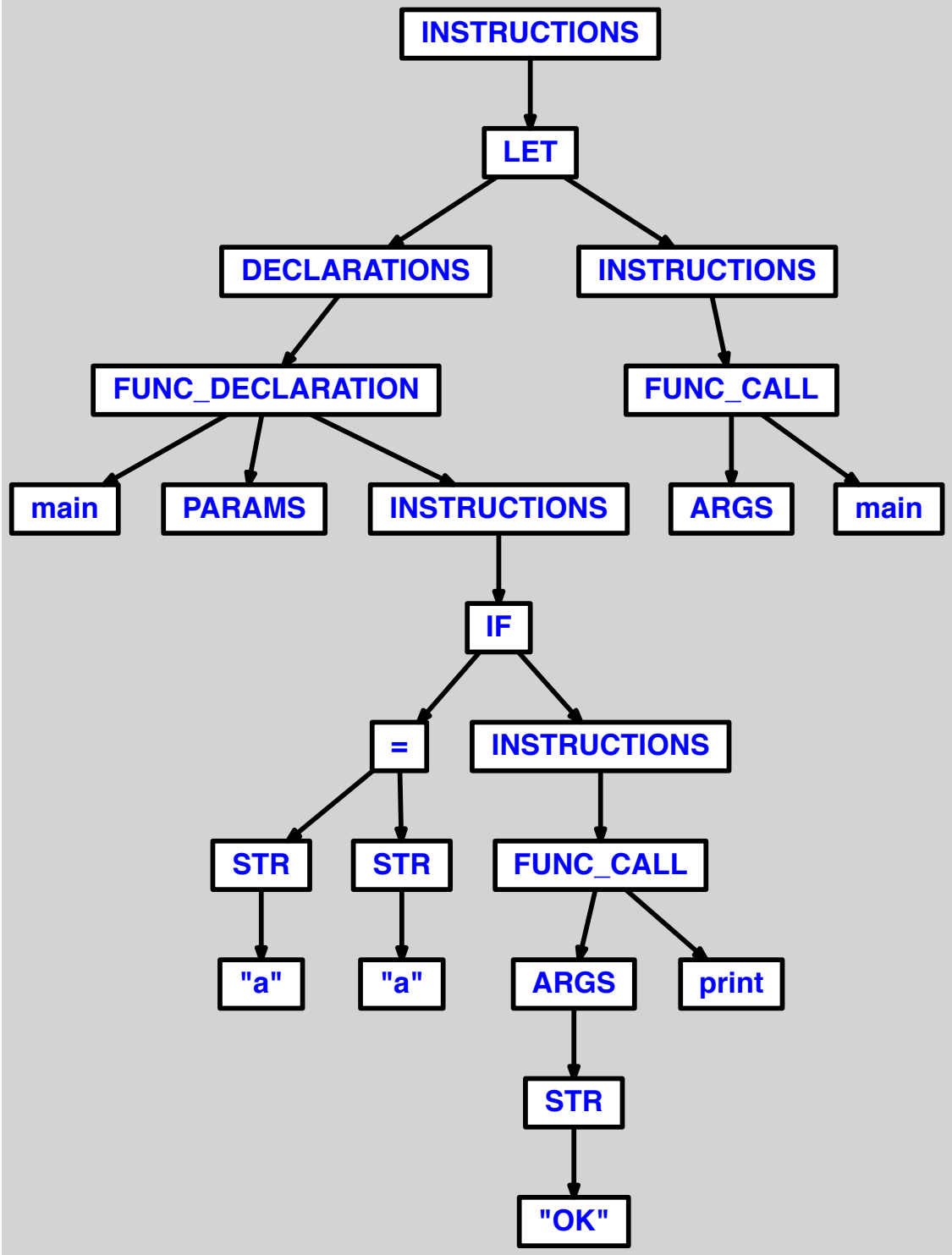
5.2.12 comparaison simple de chaines avec <">="><

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
>=
INSTRUCTIONS
STR
STR
FUNC_CALL
"a"
"a"
ARGS
print
STR
"OK"

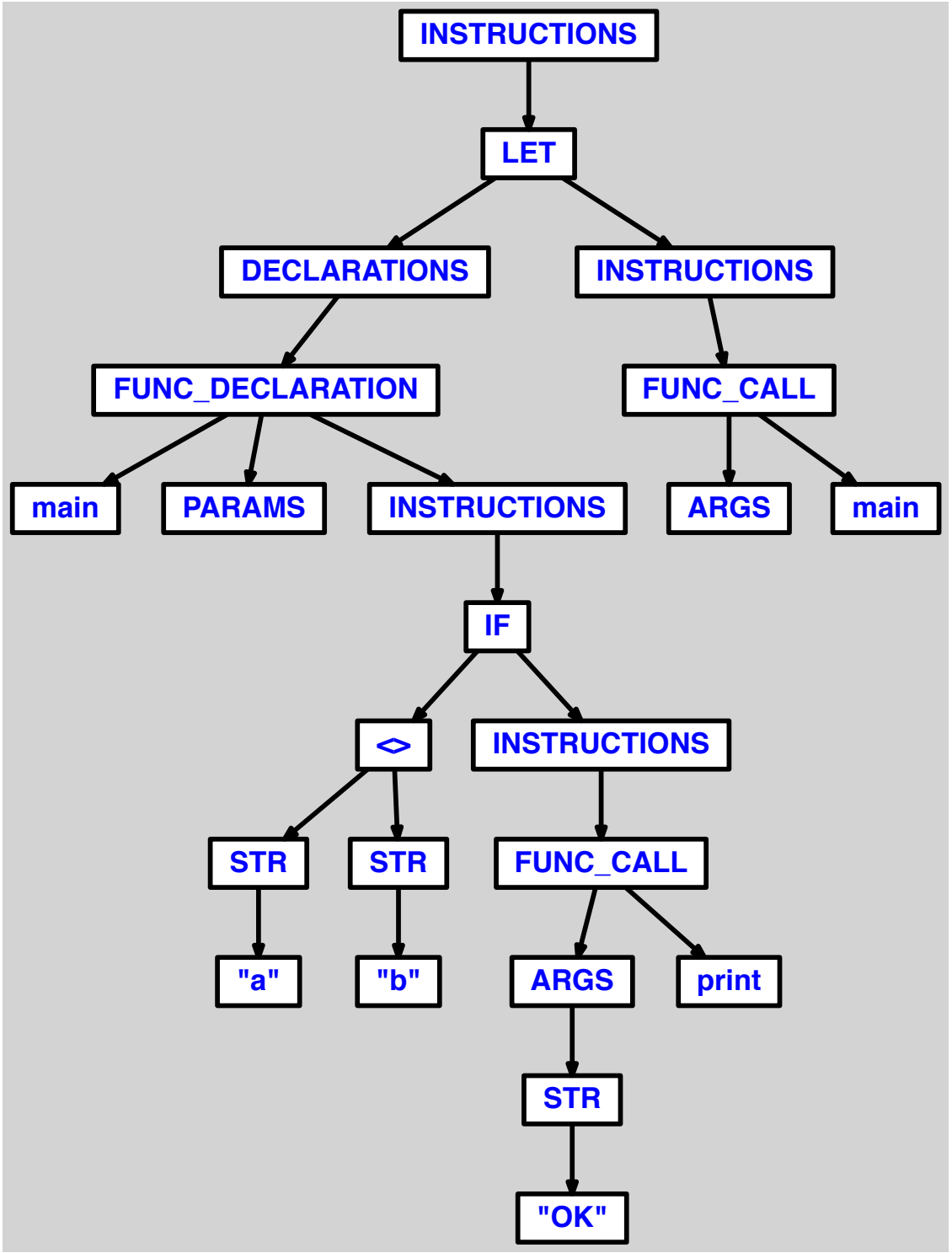
5.2.13 comparaison simple de chaines avec <=>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
=
INSTRUCTIONS
STR
STR
FUNC_CALL
"a"
"a"
ARGS
print
STR
"OK"

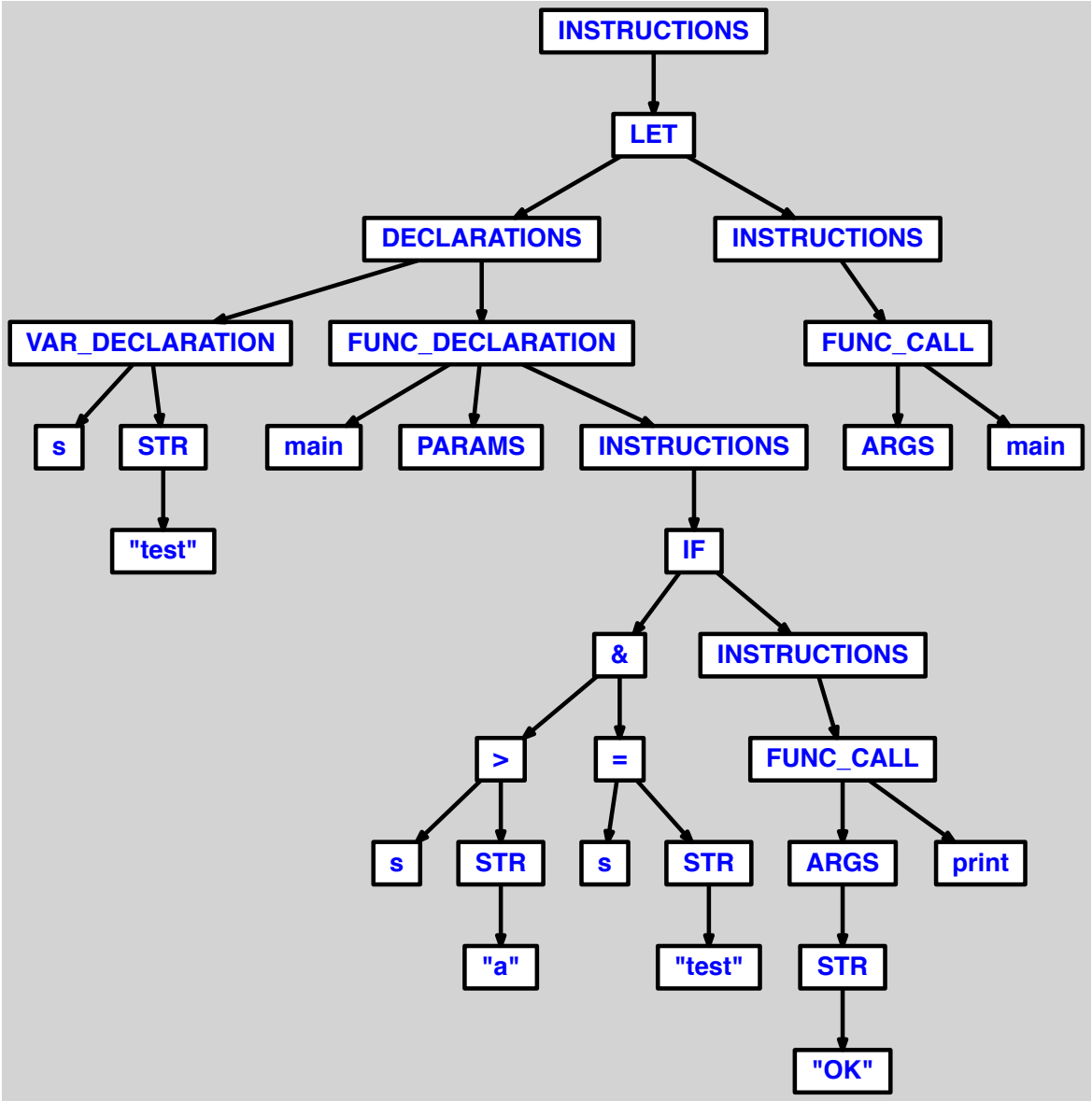
5.2.14 comparaison simple de chaines avec <"<>">

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
<>
INSTRUCTIONS
STR
STR
FUNC_CALL
"a"
"b"
ARGS
print
STR
"OK"

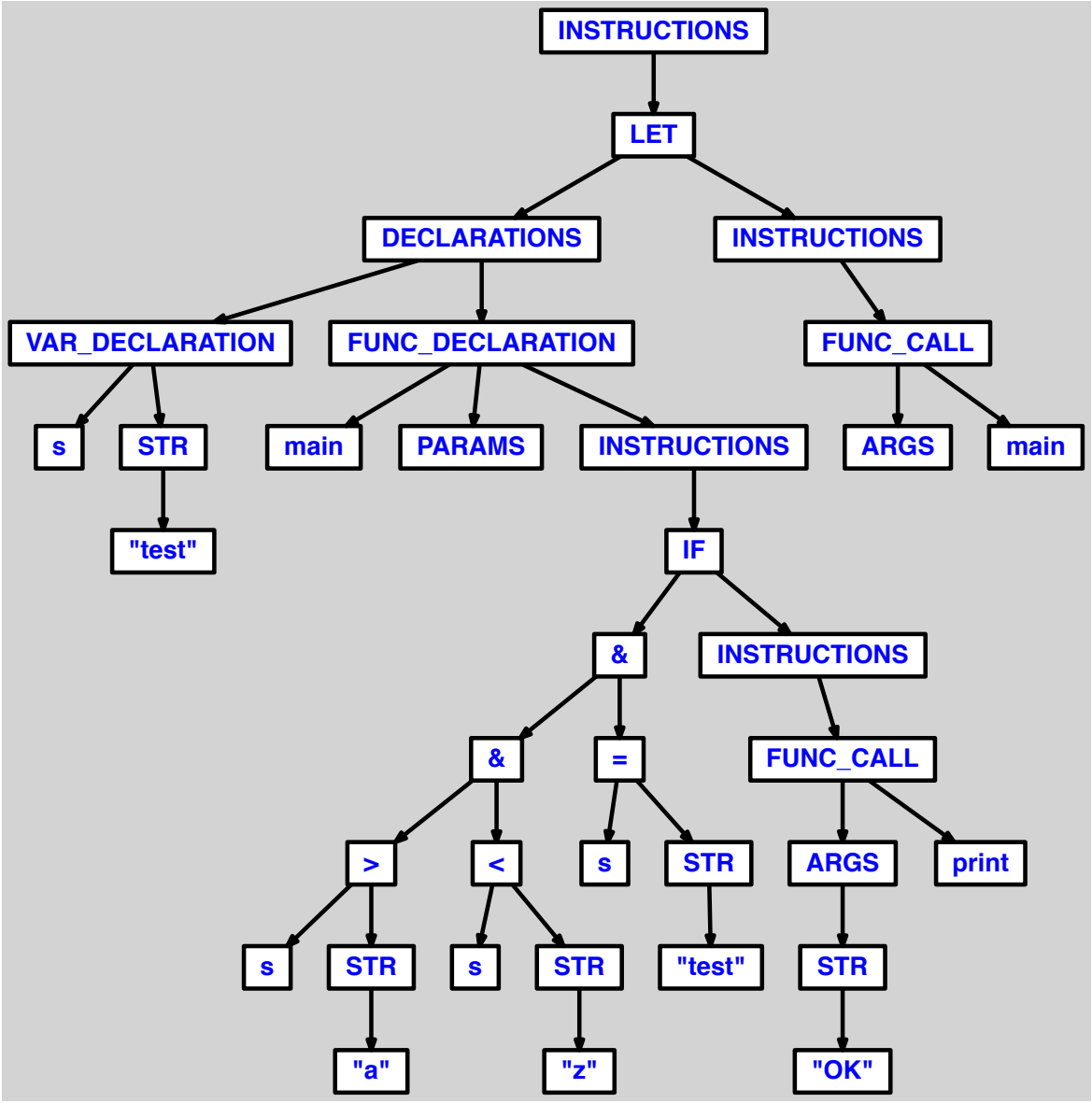
5.2.15 comparaison double de chaines

```
let
  var s := "test"

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
s
STR
main
PARAMS
INSTRUCTIONS
ARGS
main
"test"
IF
&
INSTRUCTIONS
>
=
FUNC_CALL
s
STR
s
STR
ARGS
print
"a"
"test"
STR
"OK"

5.2.16 comparaison triple de chaines

```
let
  var s := "test"

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
s
STR
main
PARAMS
INSTRUCTIONS
ARGS
main
"test"
IF
&
INSTRUCTIONS
&
=
FUNC_CALL
>
<
s
STR
ARGS
print
s
STR
s
STR
"test"
STR
"a"
"z"
"OK"

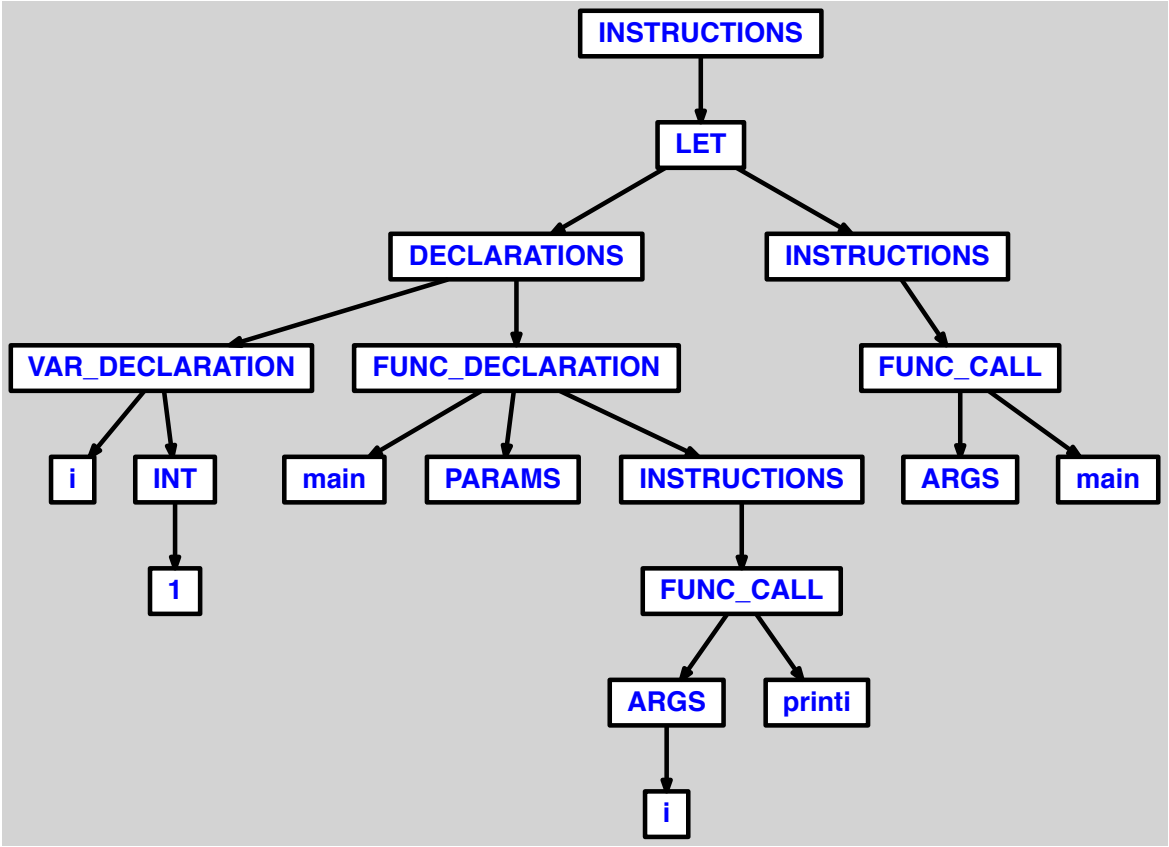
6 espace

6.1 KO

6.2 OK

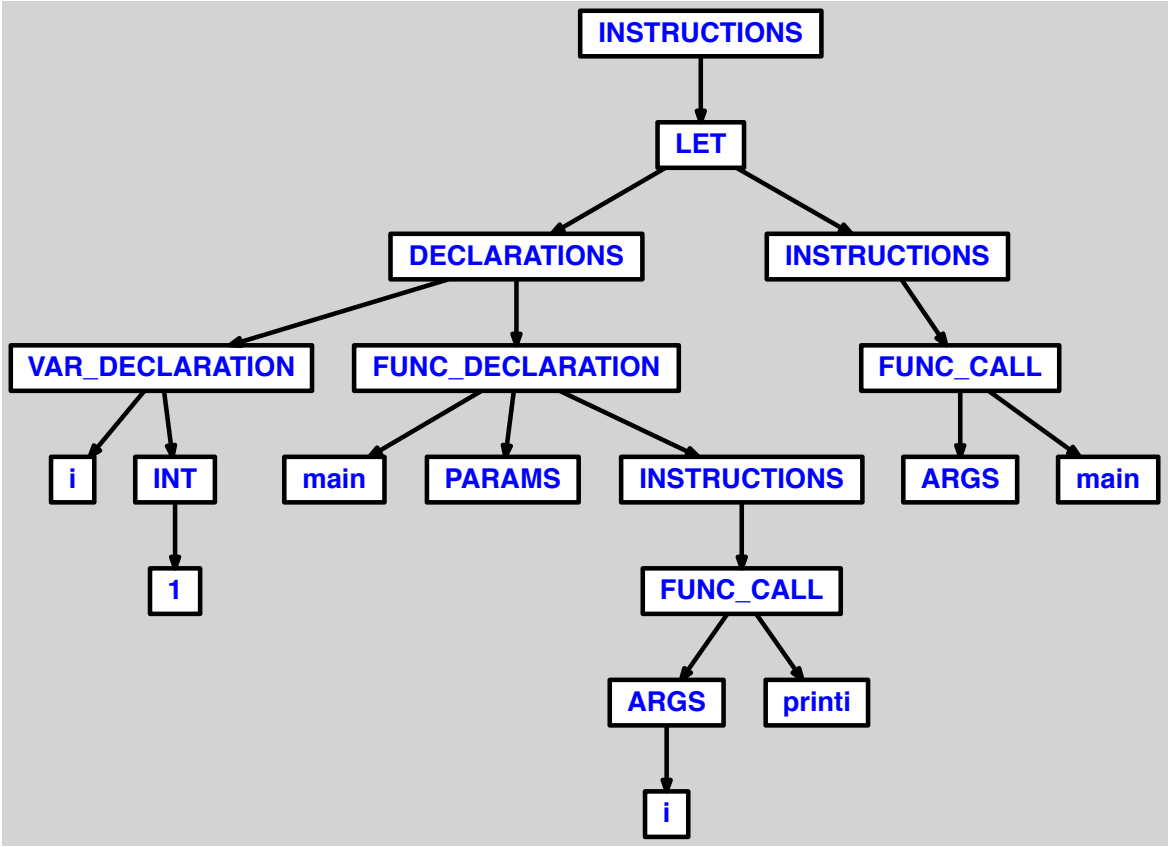
6.2.1 1 espace entre <var> et <i>

```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

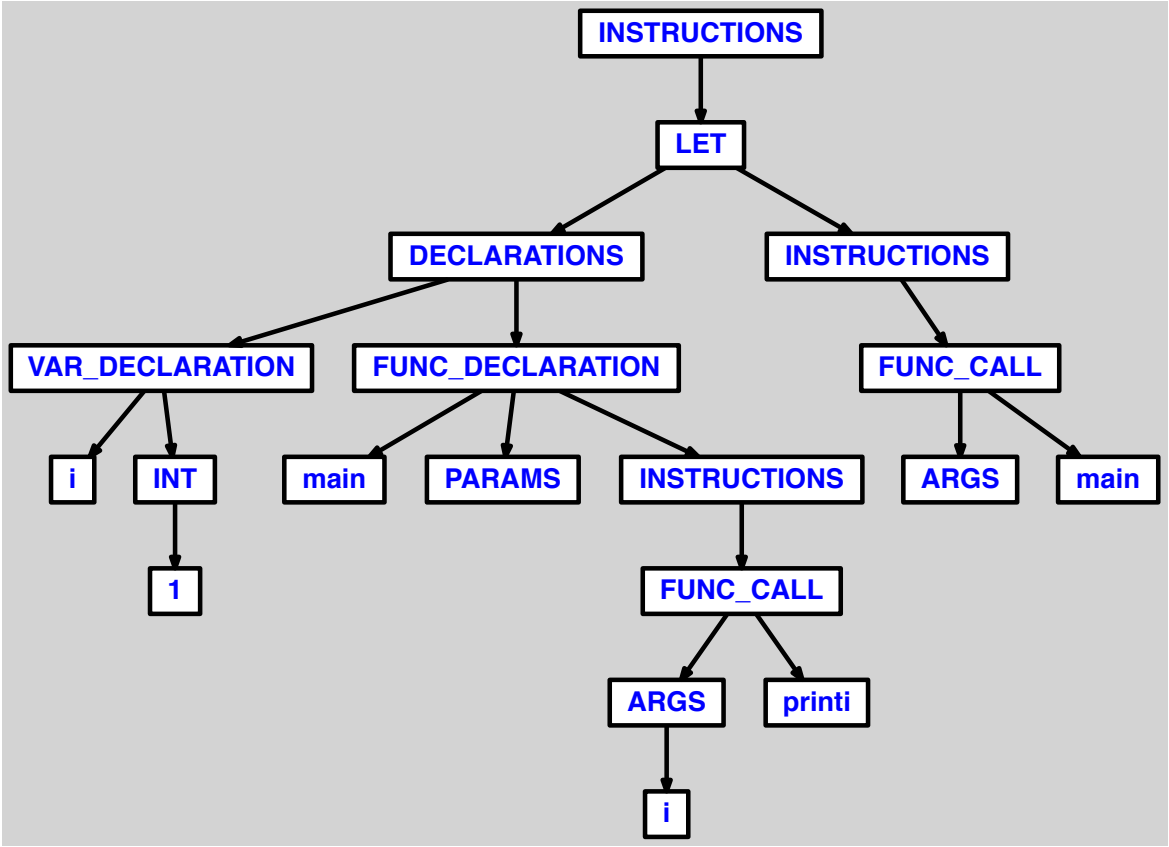
6.2.2 1 tabulation entre <var> et <i>

```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

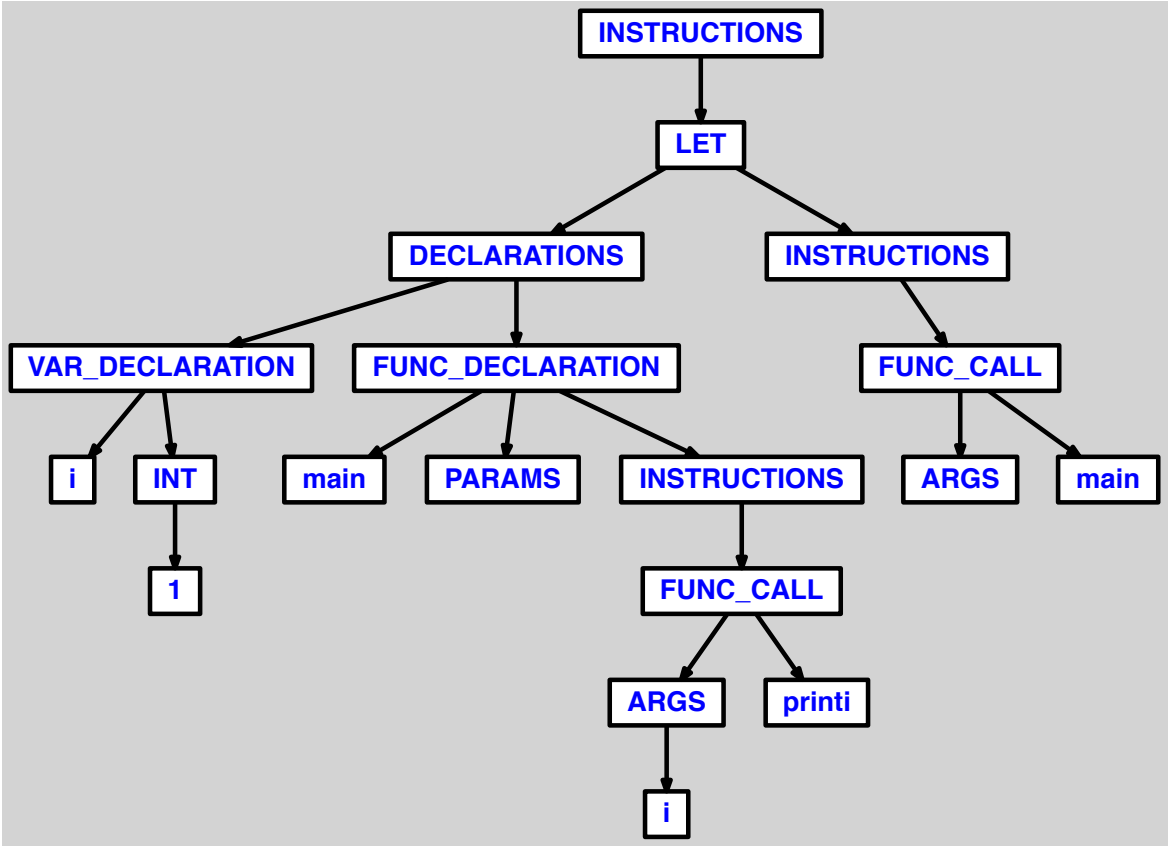
6.2.3 1 saut de ligne entre <var> et <i>

```
let
  var
  i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

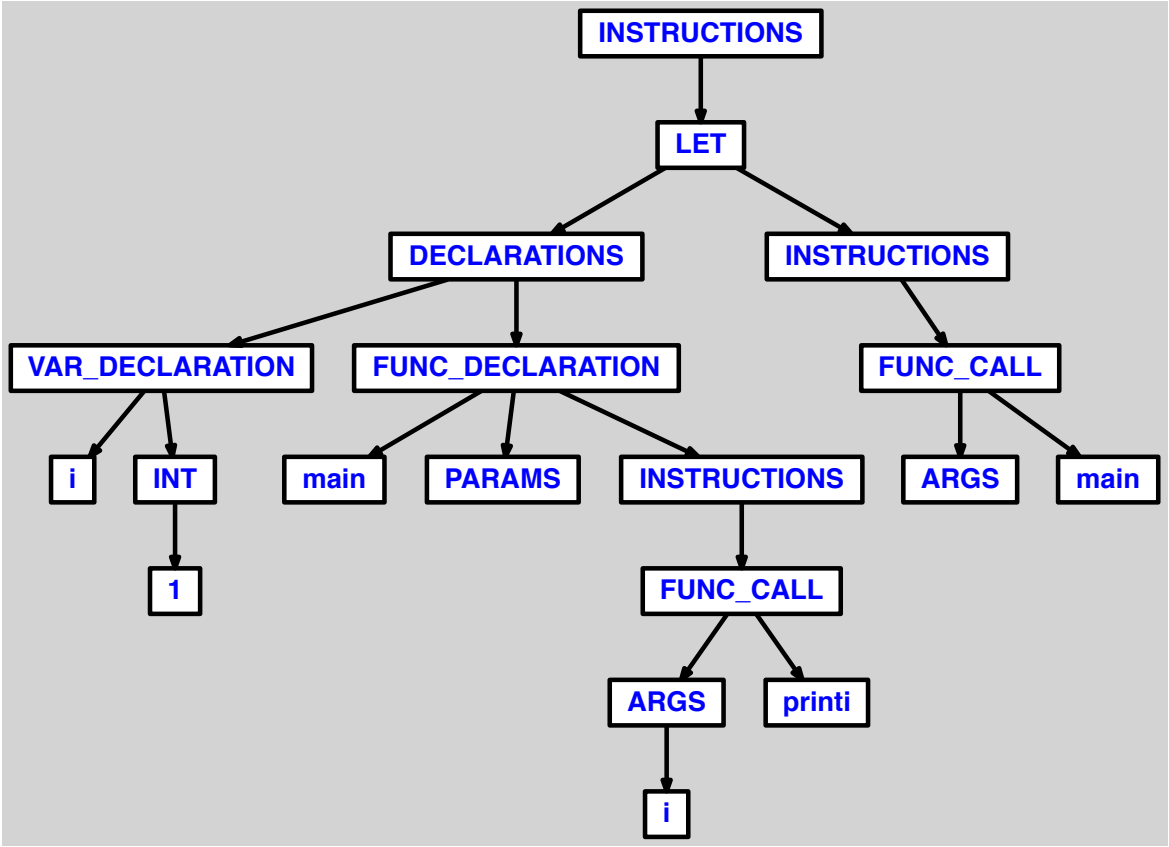
6.2.4 2 espaces entre <var> et <i>

```
let
  var  i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

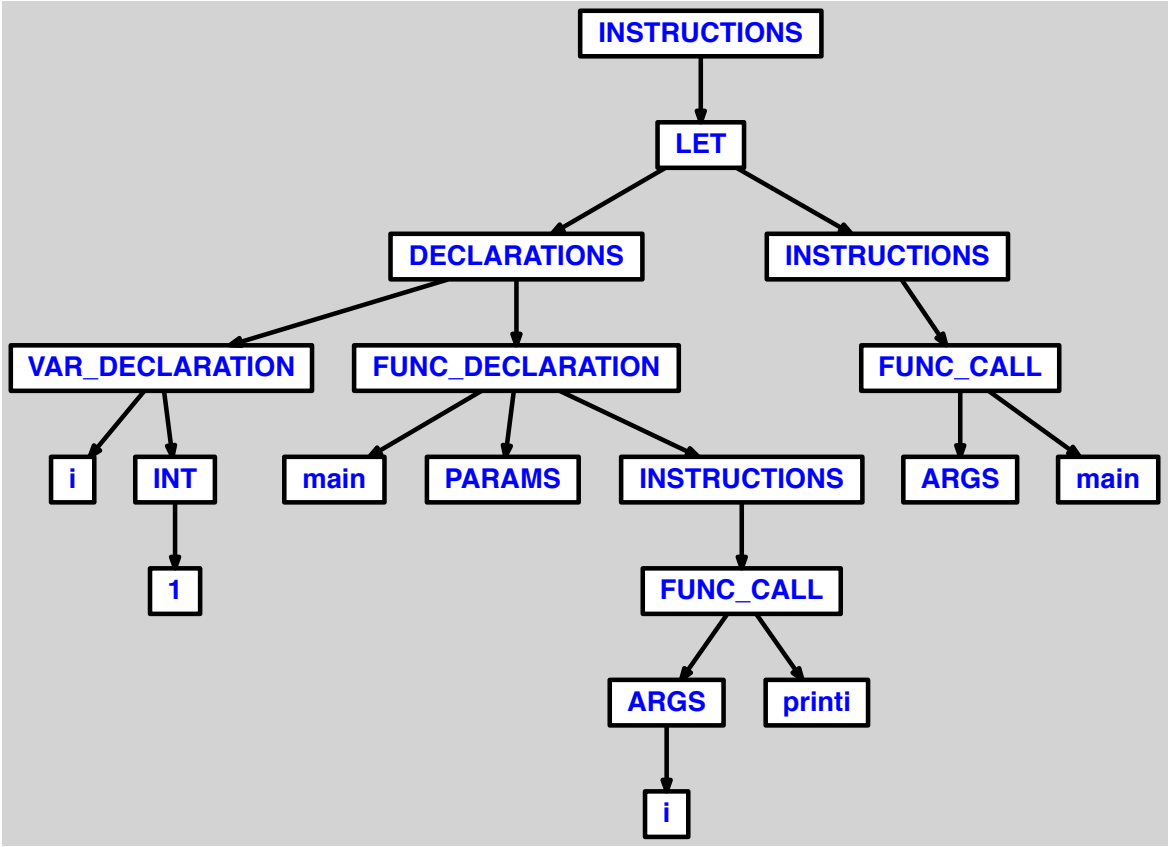
6.2.5 2 tabulations entre <var> et <i>

```
let
  var   i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

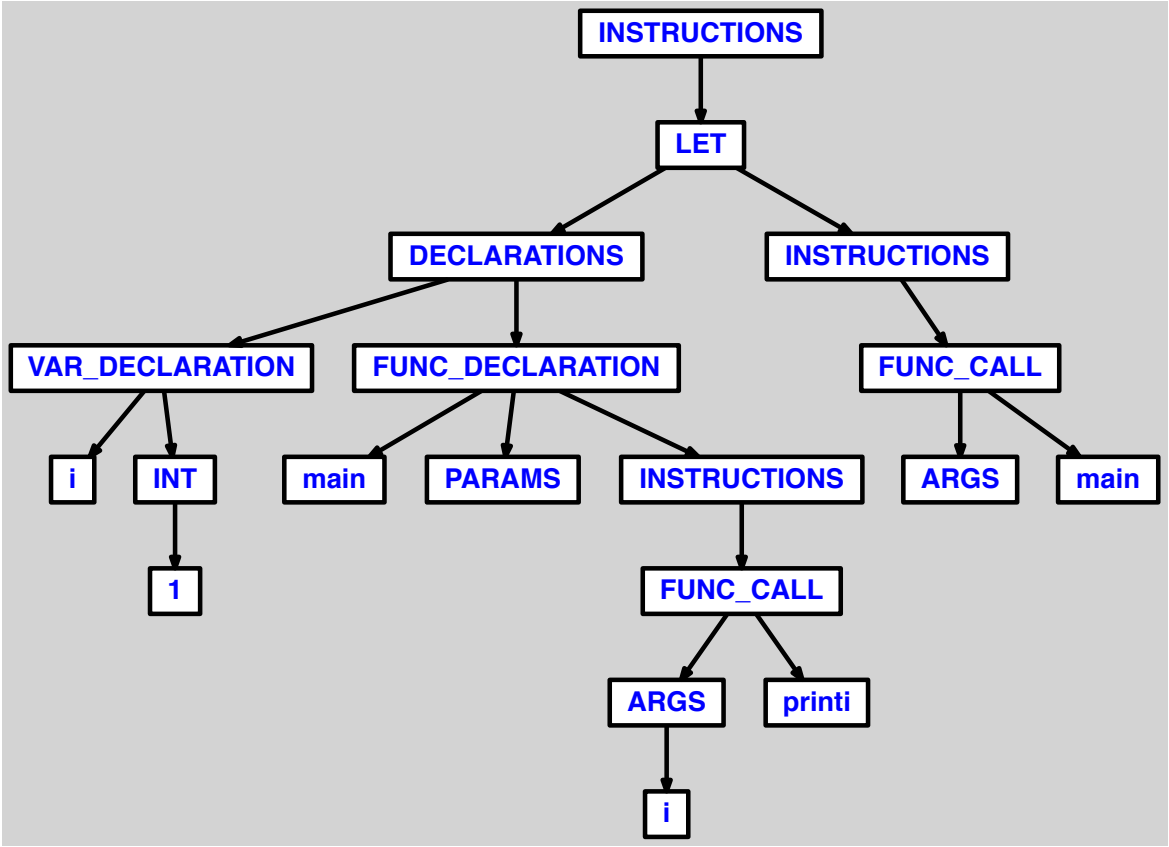
6.2.6 2 sauts de ligne entre <var> et <i>

```
let
  var

  i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

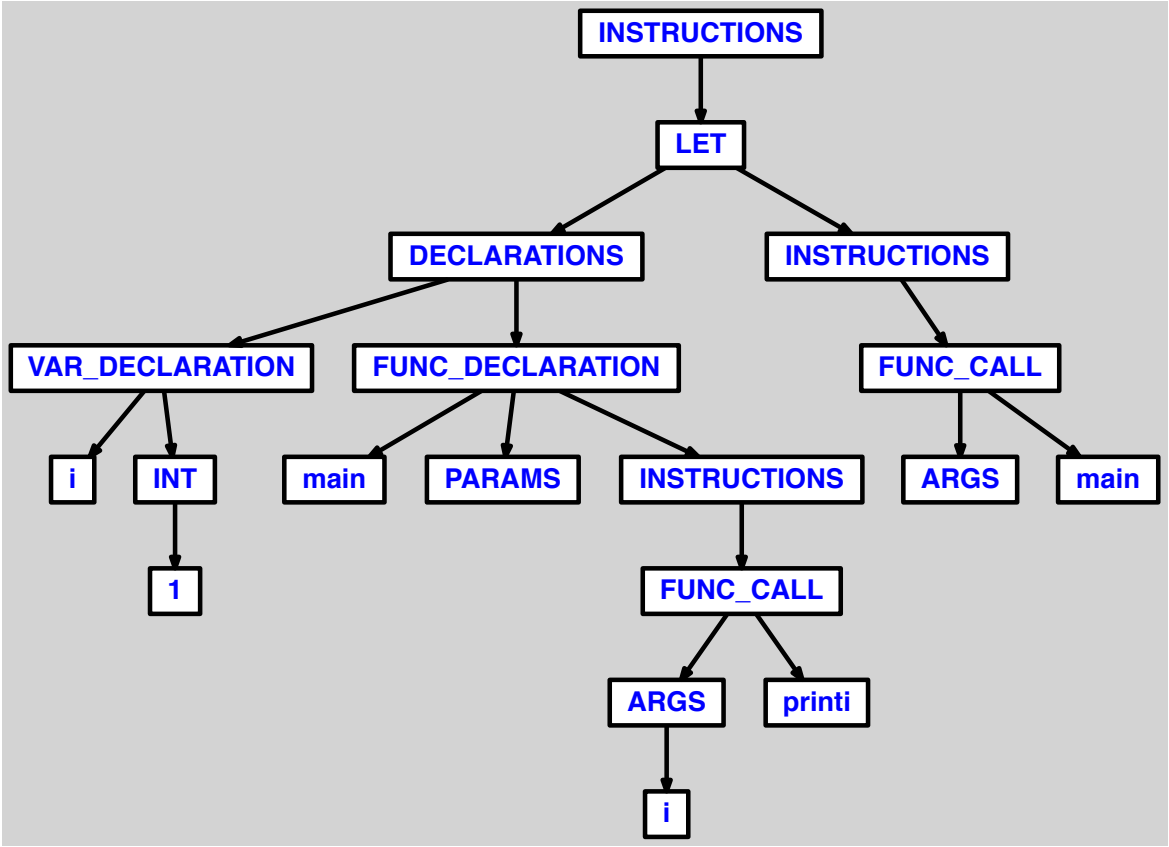
6.2.7 3 espaces entre <var> et <i>

```
let
  var   i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

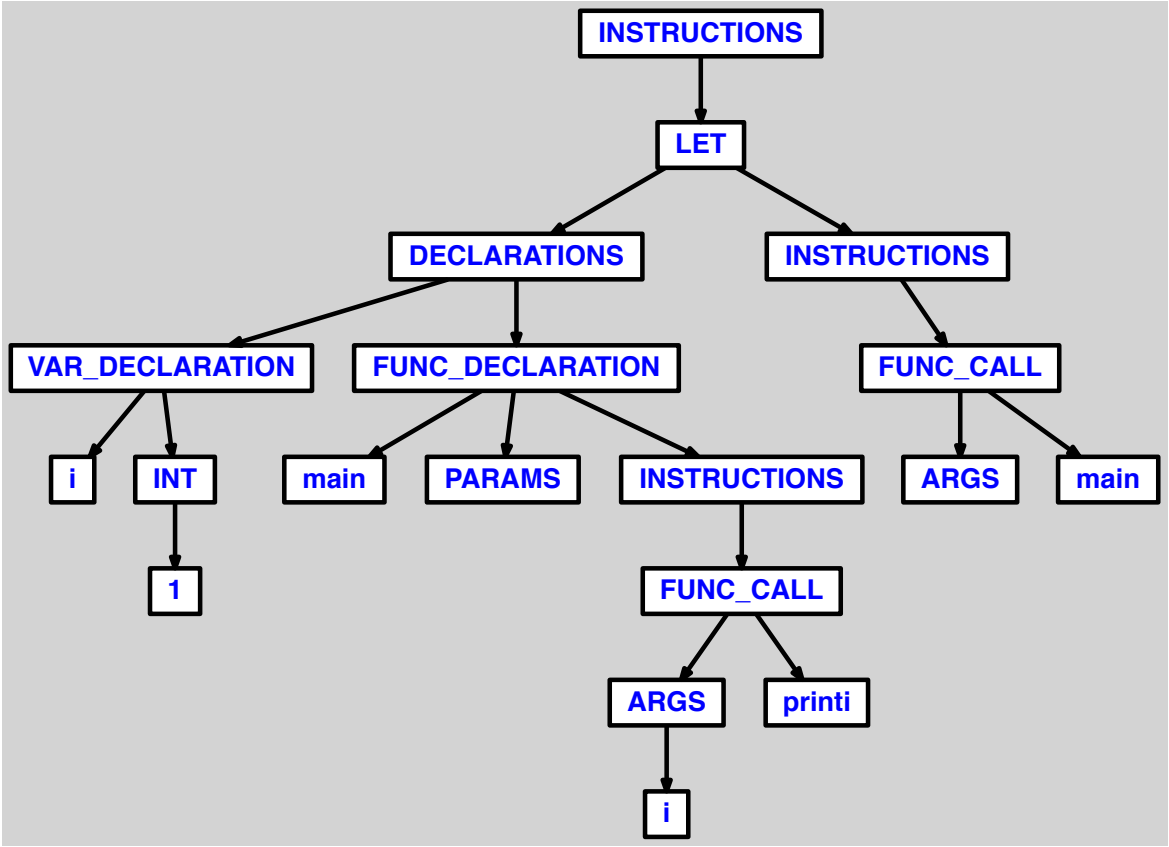
6.2.8 3 tabulations entre <var> et <i>

```
let
  var     i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

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

```
let
  var


  i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
i

7 for

7.1 KO

7.1.1 <for> avec affectation de chaine pour le compteur

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

Pas d'AST, problème de syntaxe.

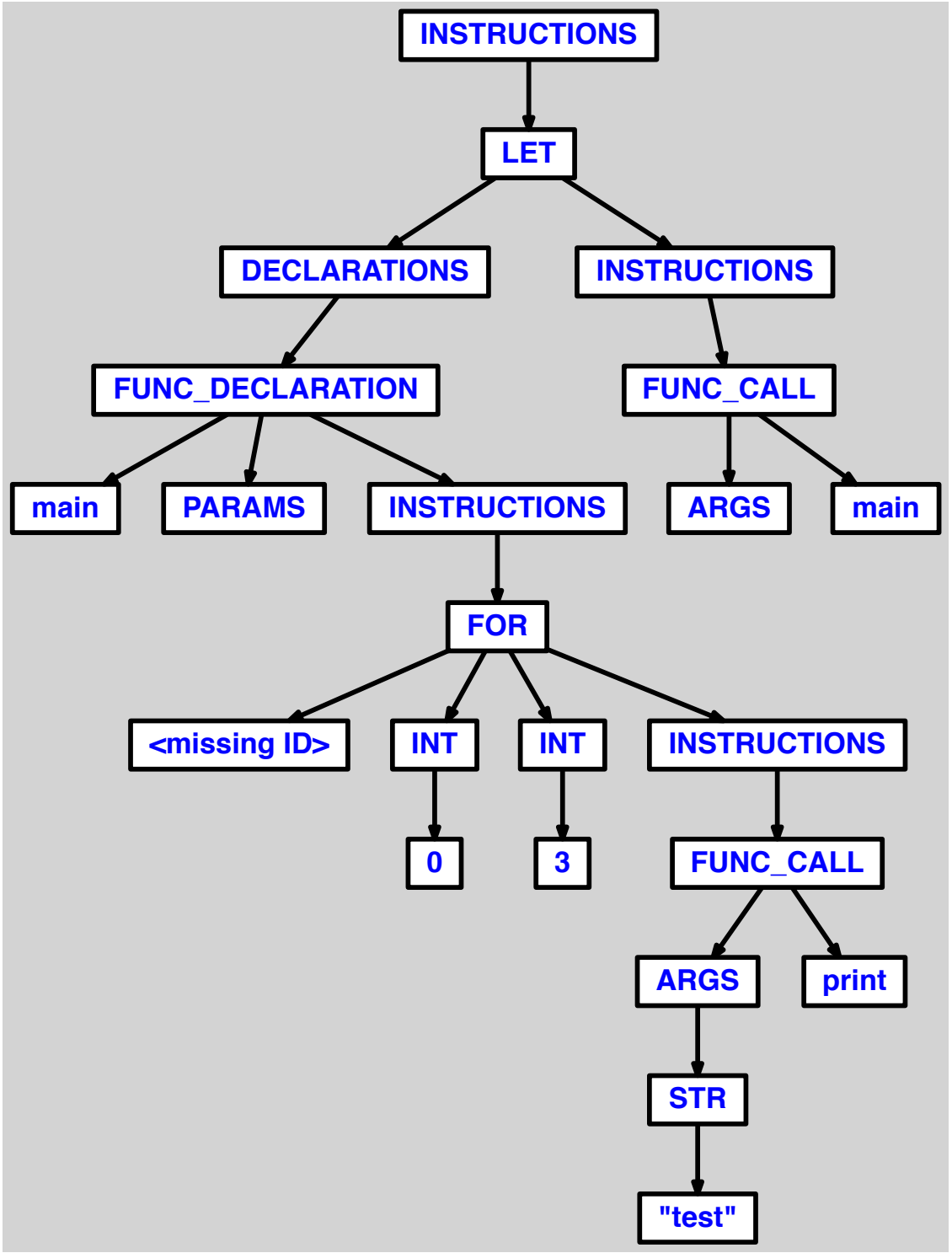
7.1.2 <for> avec affectation de chaine pour la borne superieure de l'iteration

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

Pas d'AST, problème de syntaxe.

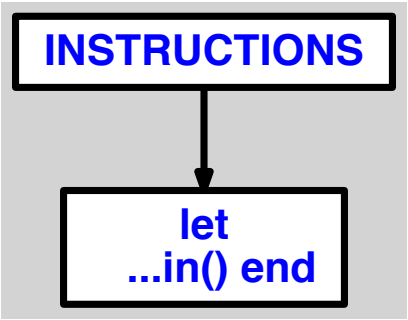
7.1.3 <for> avec oubli de l'identifiant du compteur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
<missing ID>
INT
INT
INSTRUCTIONS
0
3
FUNC_CALL
ARGS
print
STR
"test"

7.1.4 <for> avec oubli du <for>

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


INSTRUCTIONS
let
...in() end

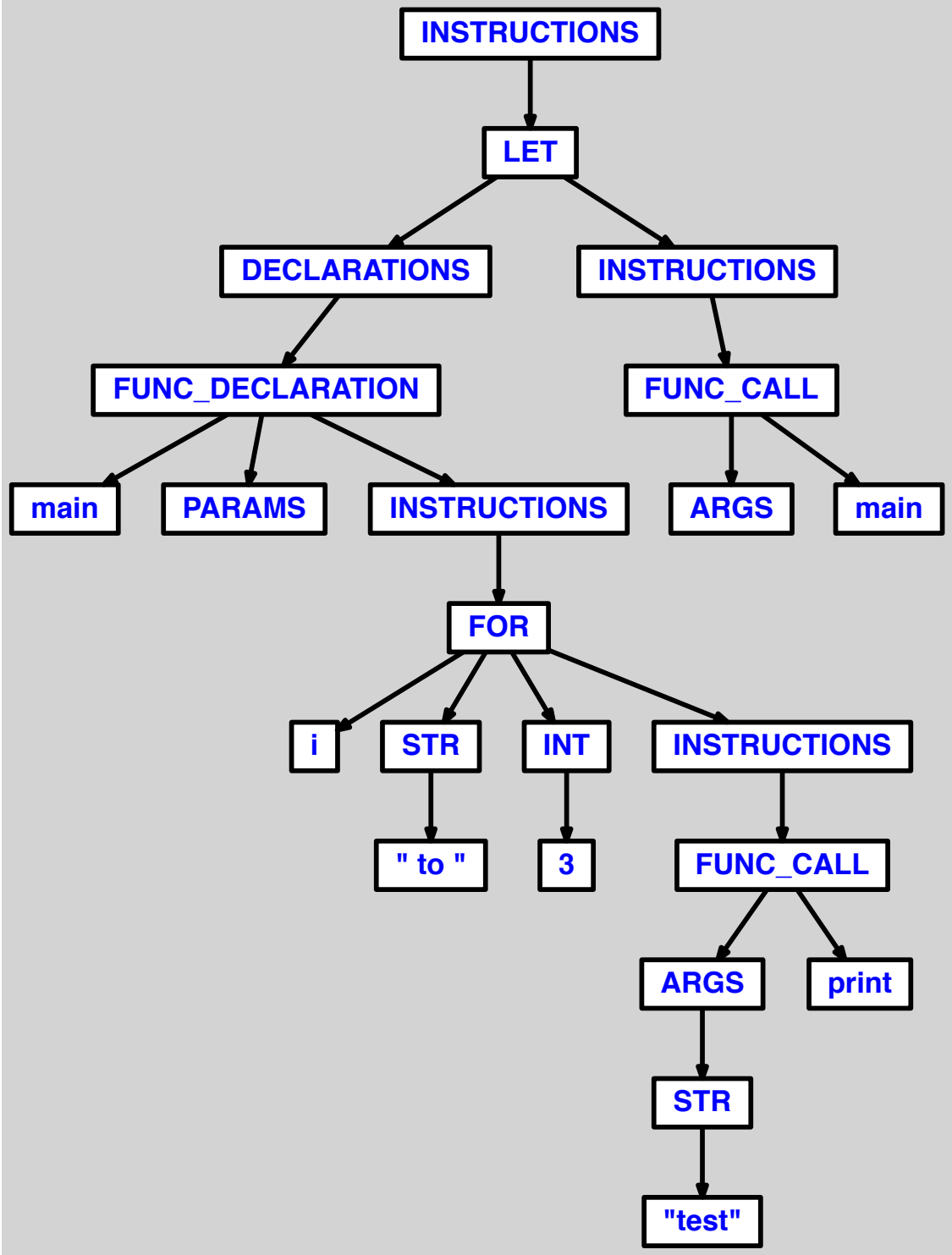
7.1.5 <for> "express"

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

Pas d'AST, problème de syntaxe.

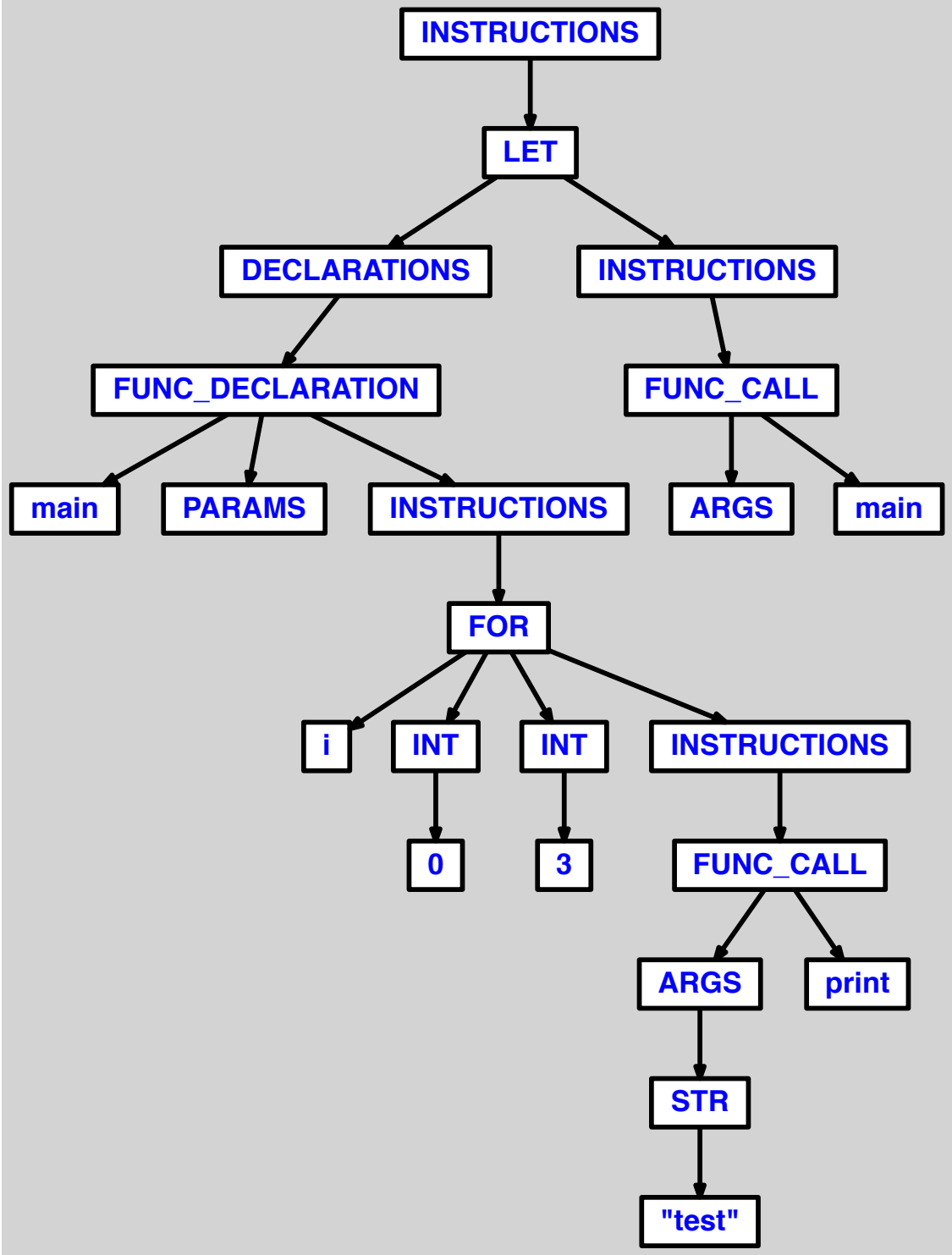
7.1.6 <for> avec affectation de chaines pour le compteur et la borne maximale de l'iteration

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
STR
INT
INSTRUCTIONS
" to "
3
FUNC_CALL
ARGS
print
STR
"test"

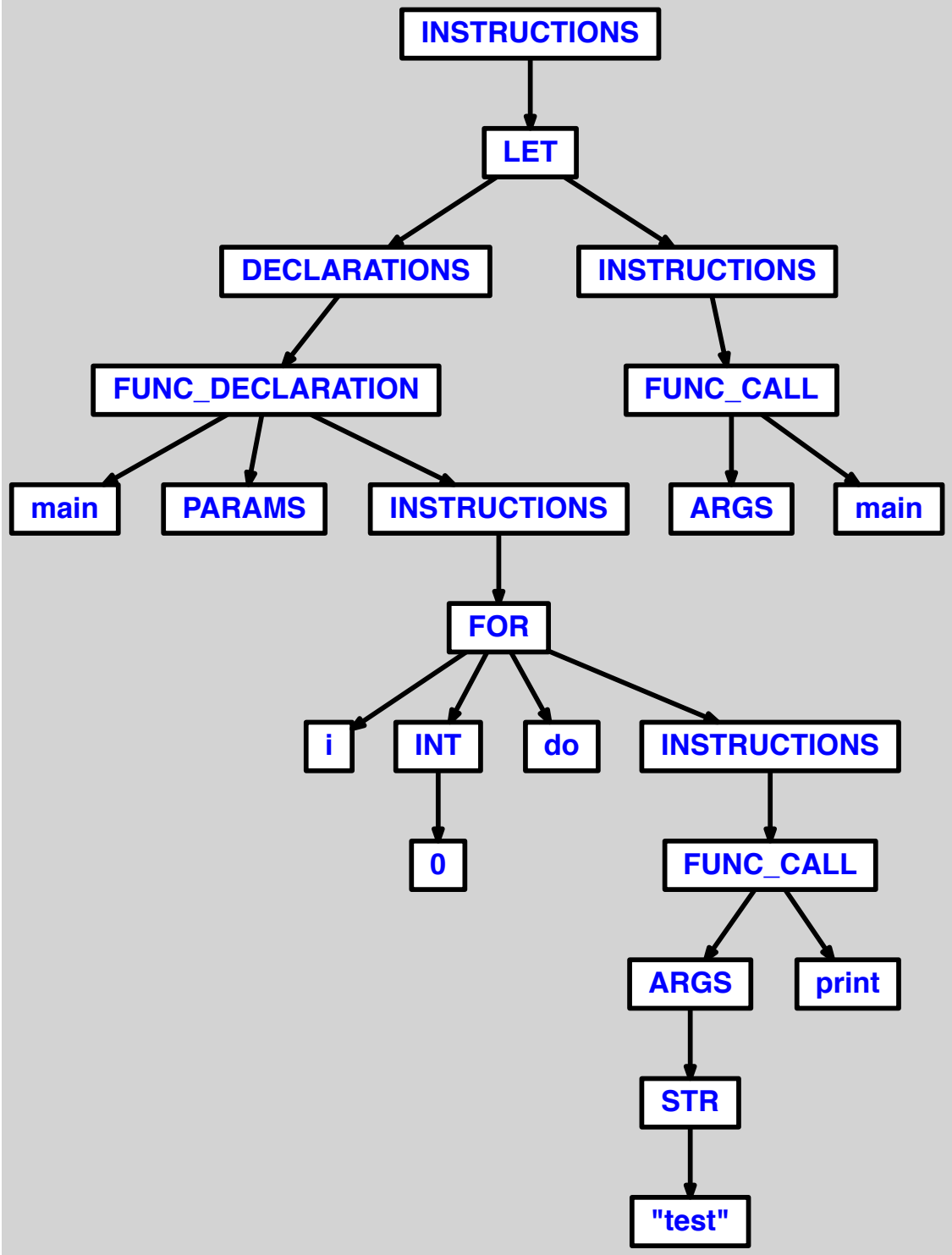
7.1.7 <for> avec oubli du <do>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FUNC_CALL
ARGS
print
STR
"test"

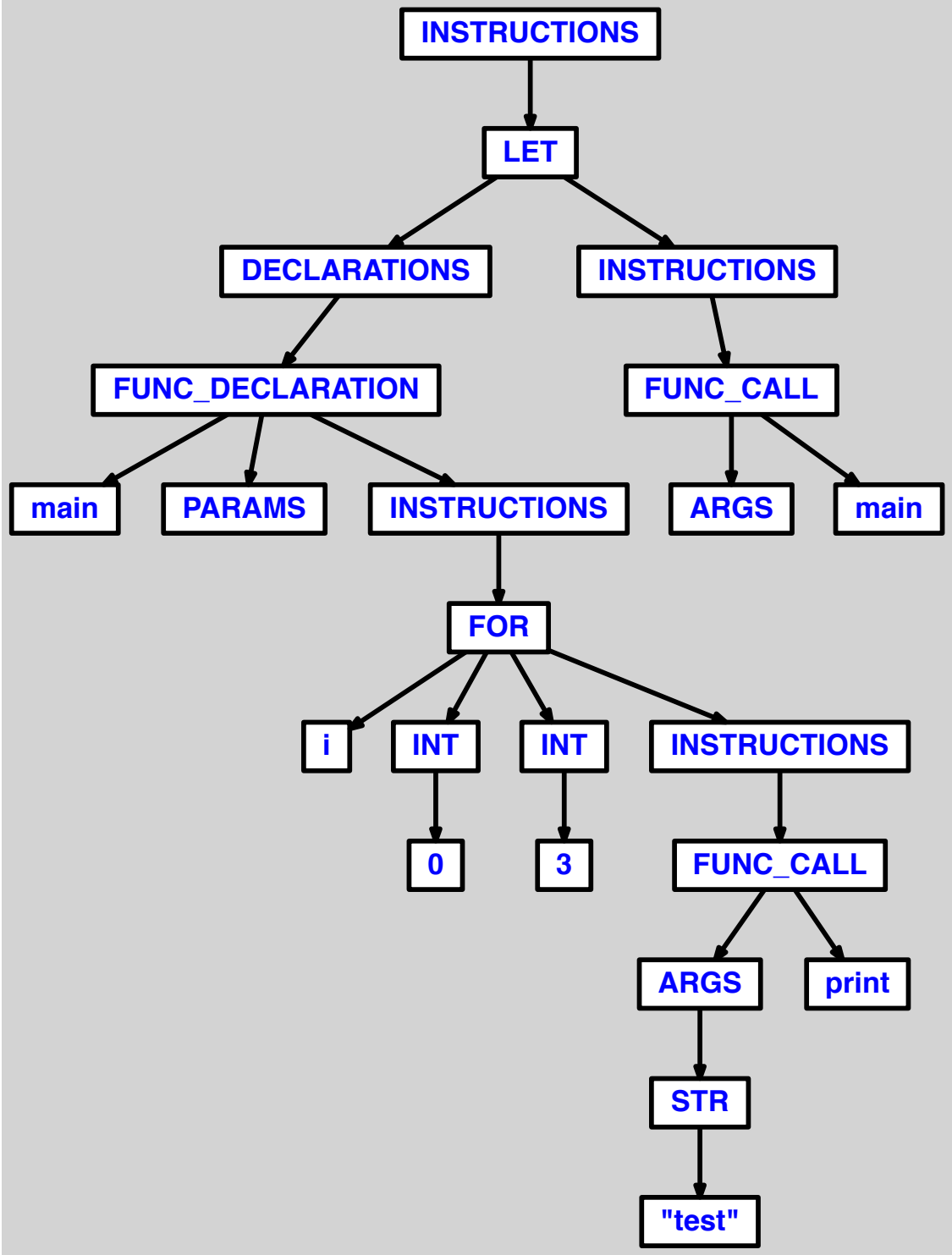
7.1.8 <for> avec oubli de la borne maximale de l'iteration

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
do
INSTRUCTIONS
0
FUNC_CALL
ARGS
print
STR
"test"

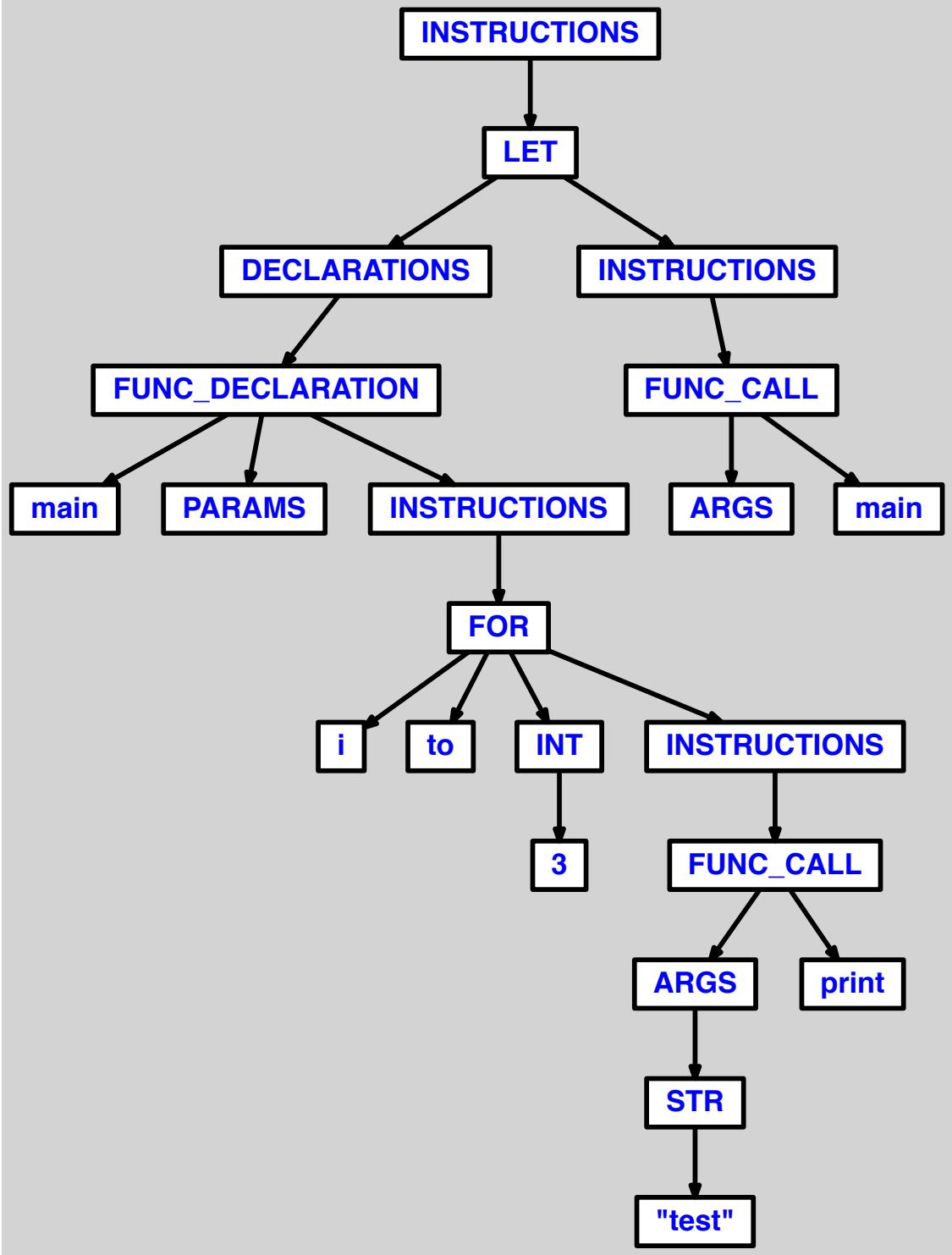
7.1.9 <for> avec oubli du <to>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FUNC_CALL
ARGS
print
STR
"test"

7.1.10 <for> avec oubli de la borne inferieure de l'iteration

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
to
INT
INSTRUCTIONS
3
FUNC_CALL
ARGS
print
STR
"test"

7.1.11 <for> avec oubli du <=>

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

Pas d'AST, problème de syntaxe.

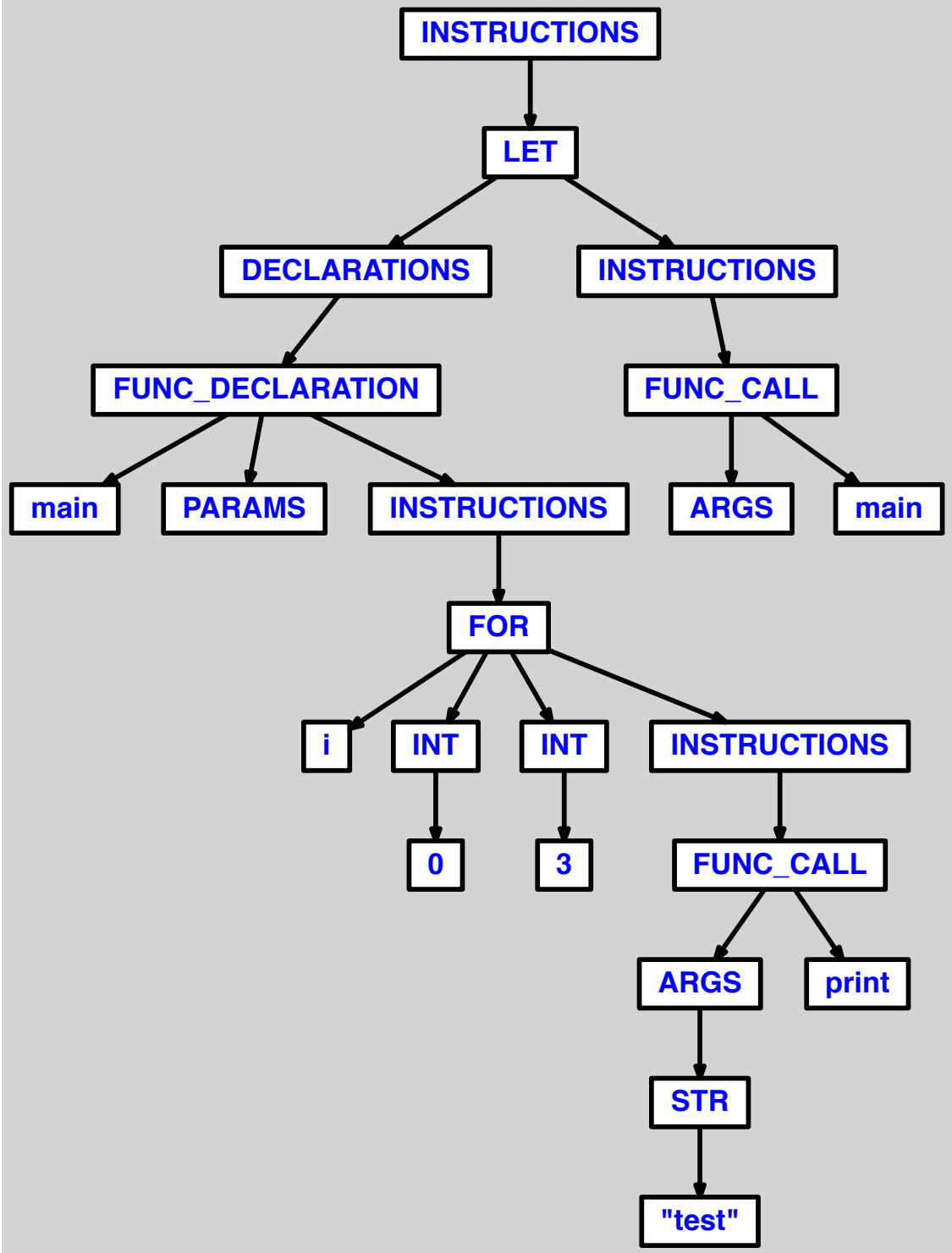
7.1.12 <for> avec oubli du < :>

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

Pas d'AST, problème de syntaxe.

7.1.13 <for> avec oubli du < :=>

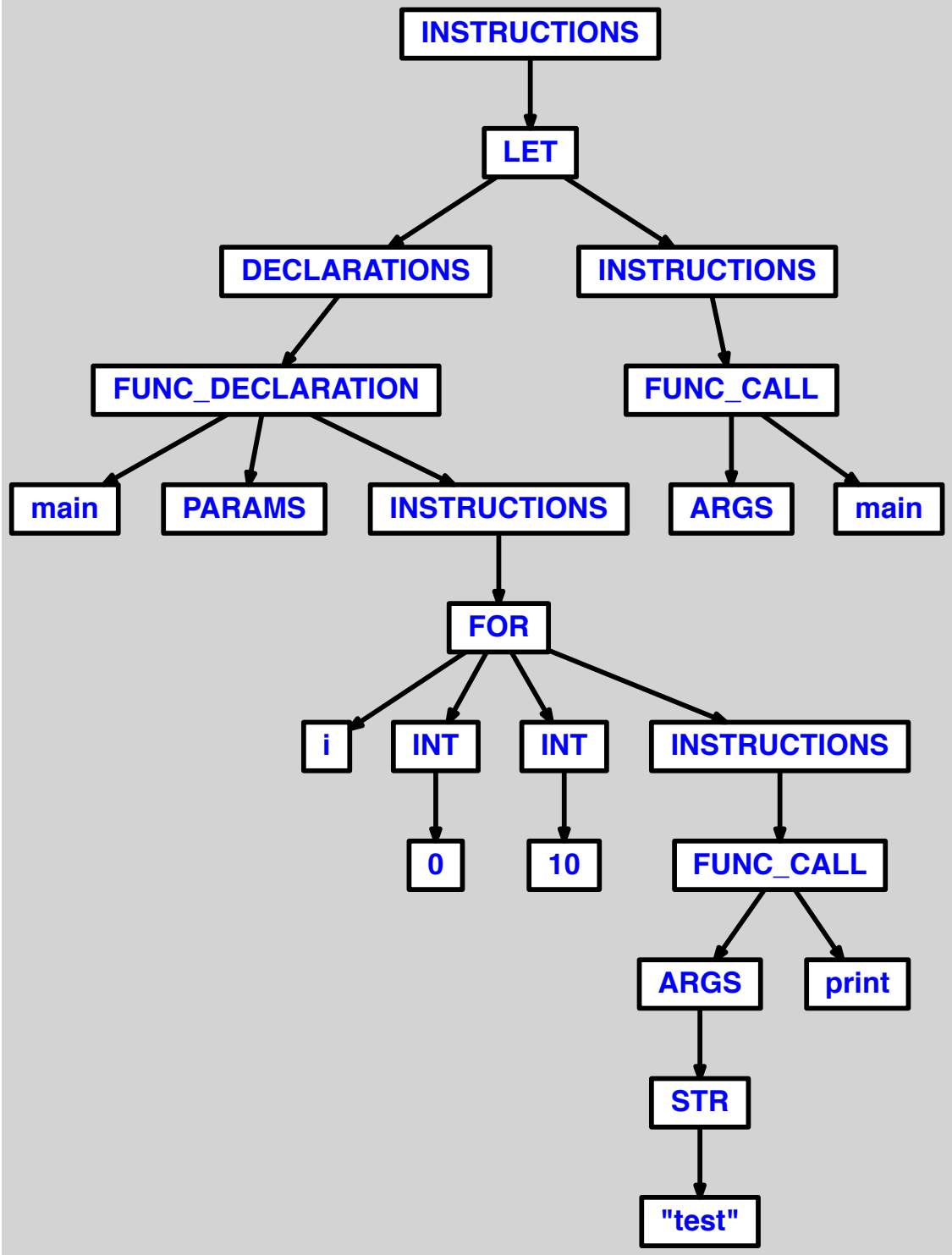
```
let
  function main() =
    for i 0 to 3 do
      print("test")
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FUNC_CALL
ARGS
print
STR
"test"

7.2 OK

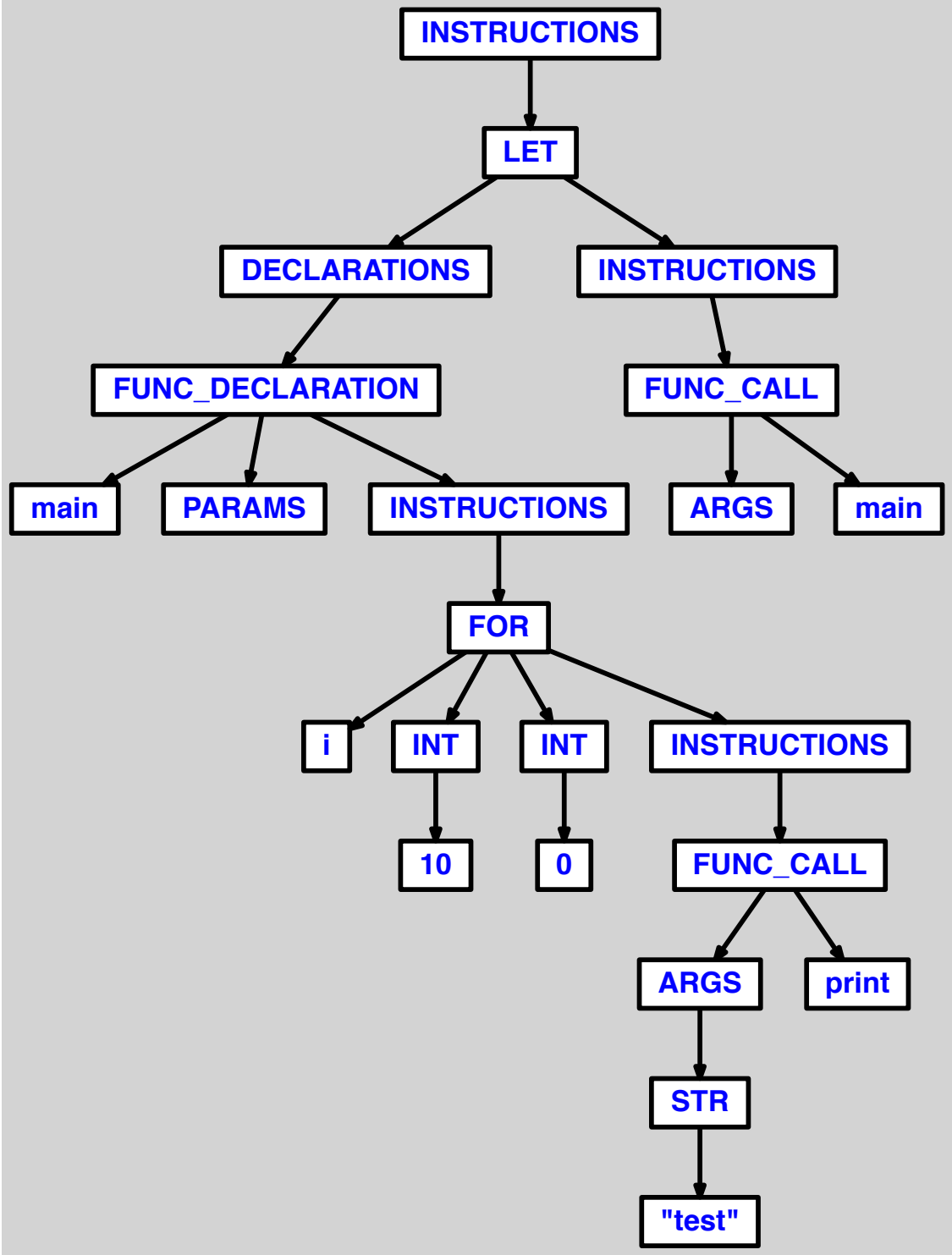
7.2.1 <for> simple croissant

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
10
FUNC_CALL
ARGS
print
STR
"test"

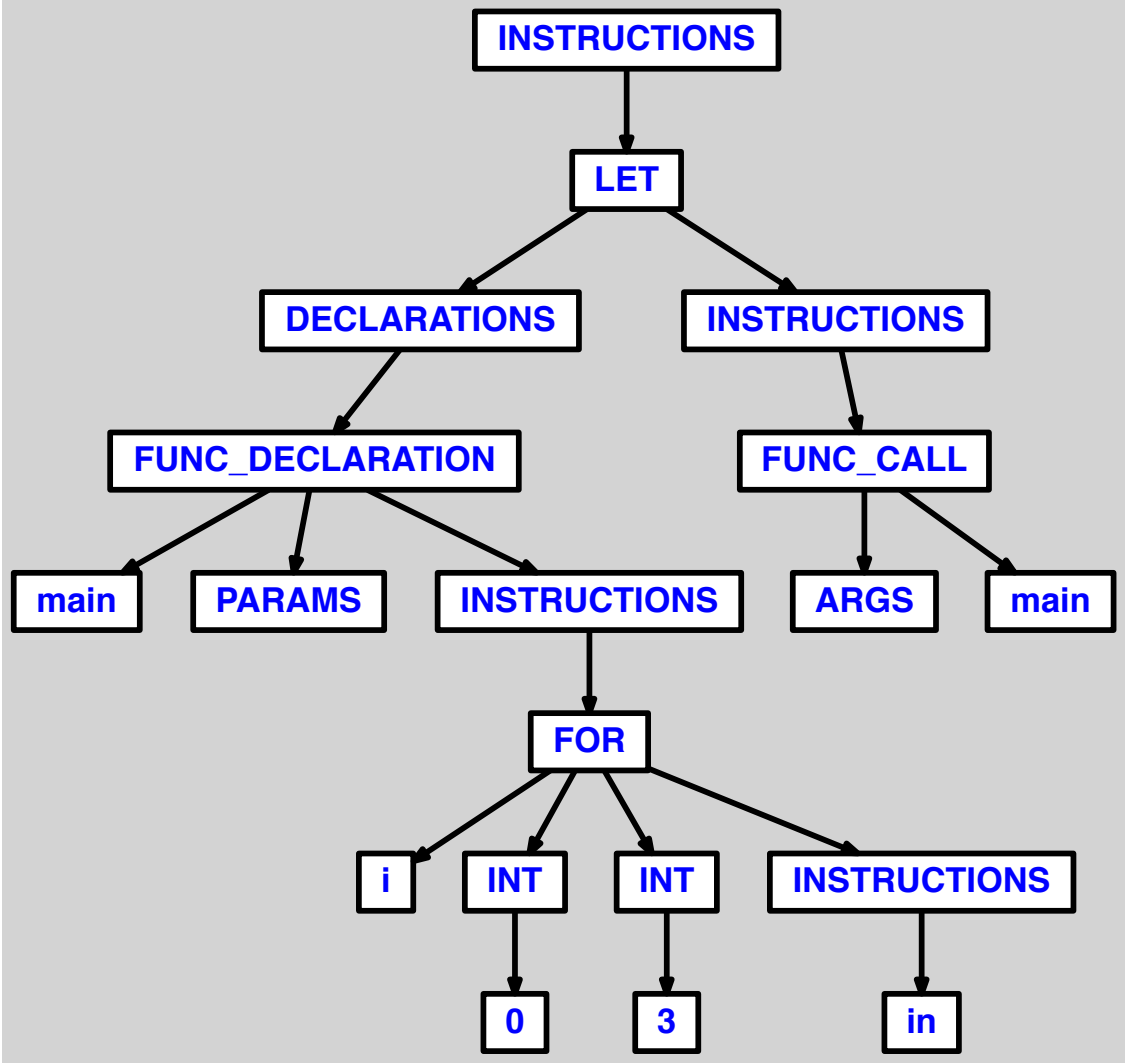
7.2.2 <for> simple decroissant

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
10
0
FUNC_CALL
ARGS
print
STR
"test"

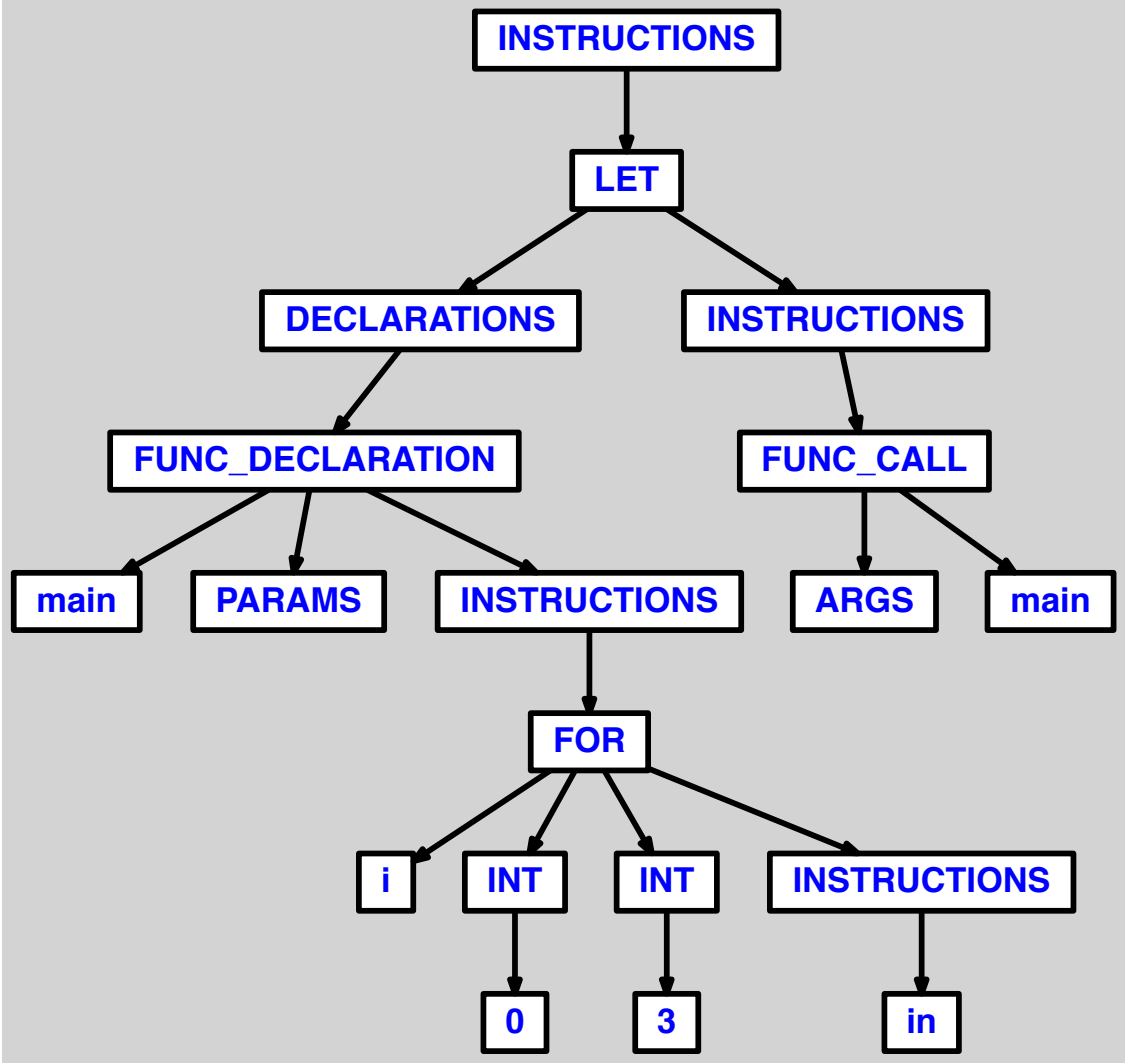
7.2.3 <for> sans instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
in

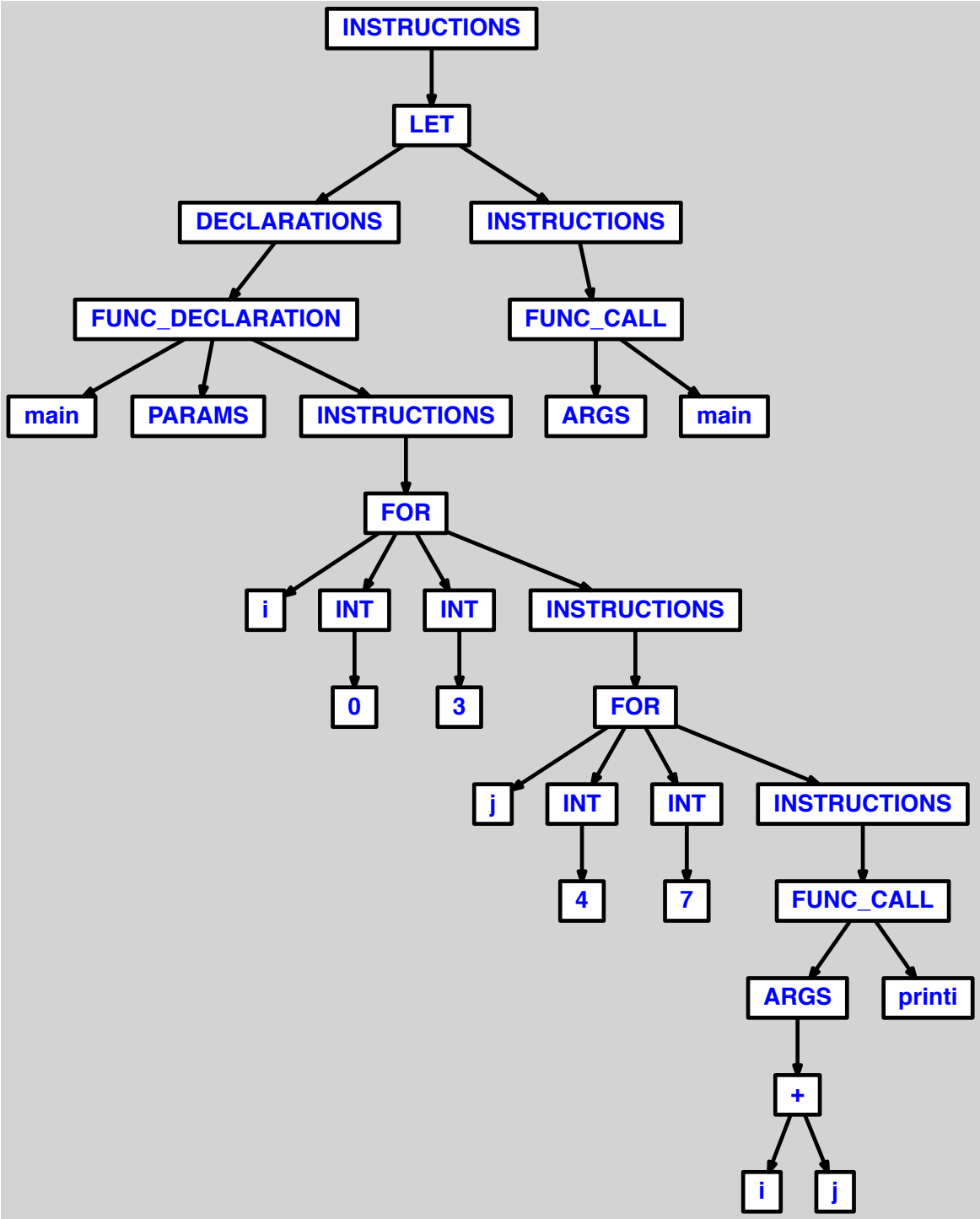
7.2.4 <for> avec ligne vide

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

in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
in

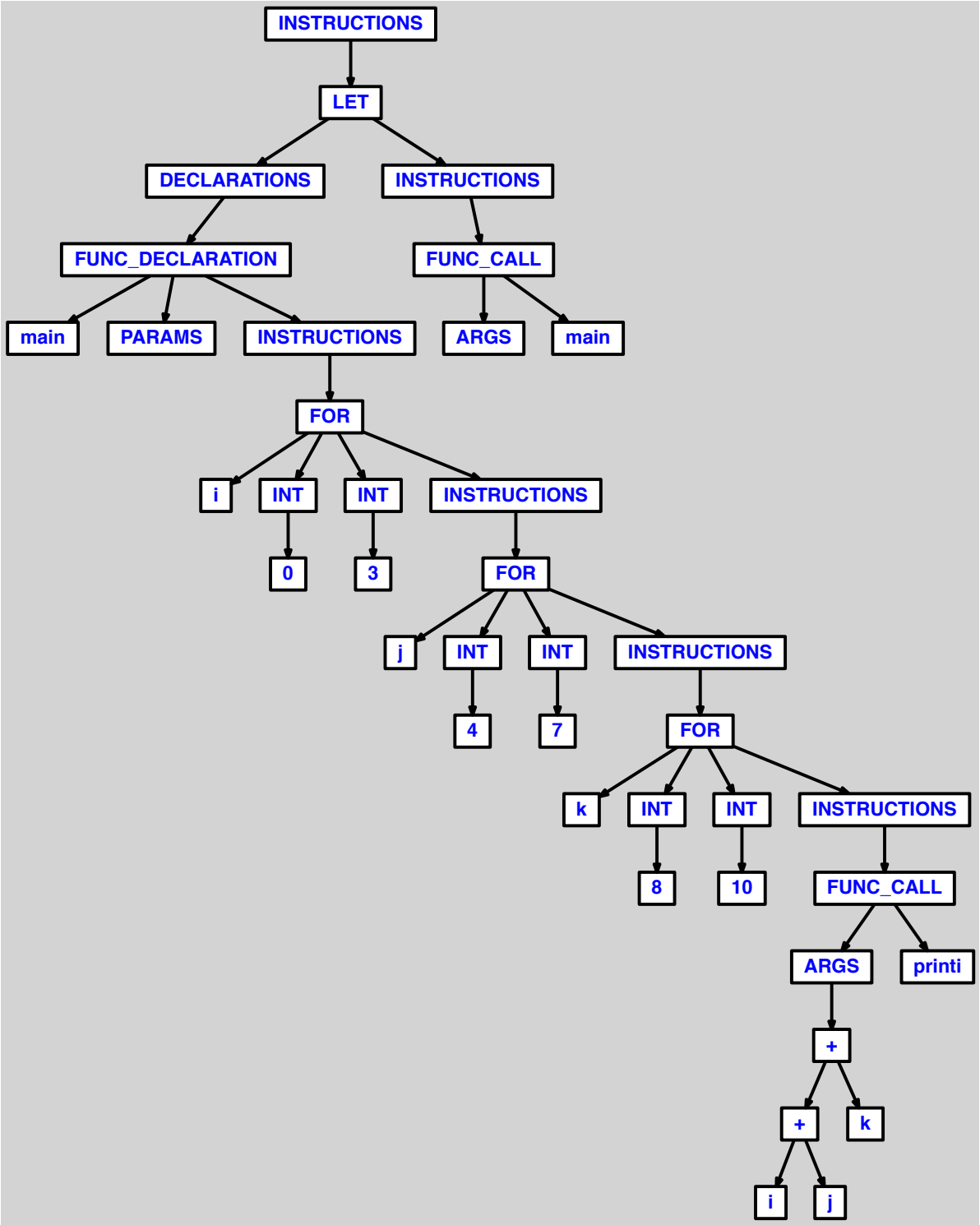
7.2.5 double-imbrication de <for>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
INT
INT
INSTRUCTIONS
4
7
FUNC_CALL
ARGS
printi
+
i
j

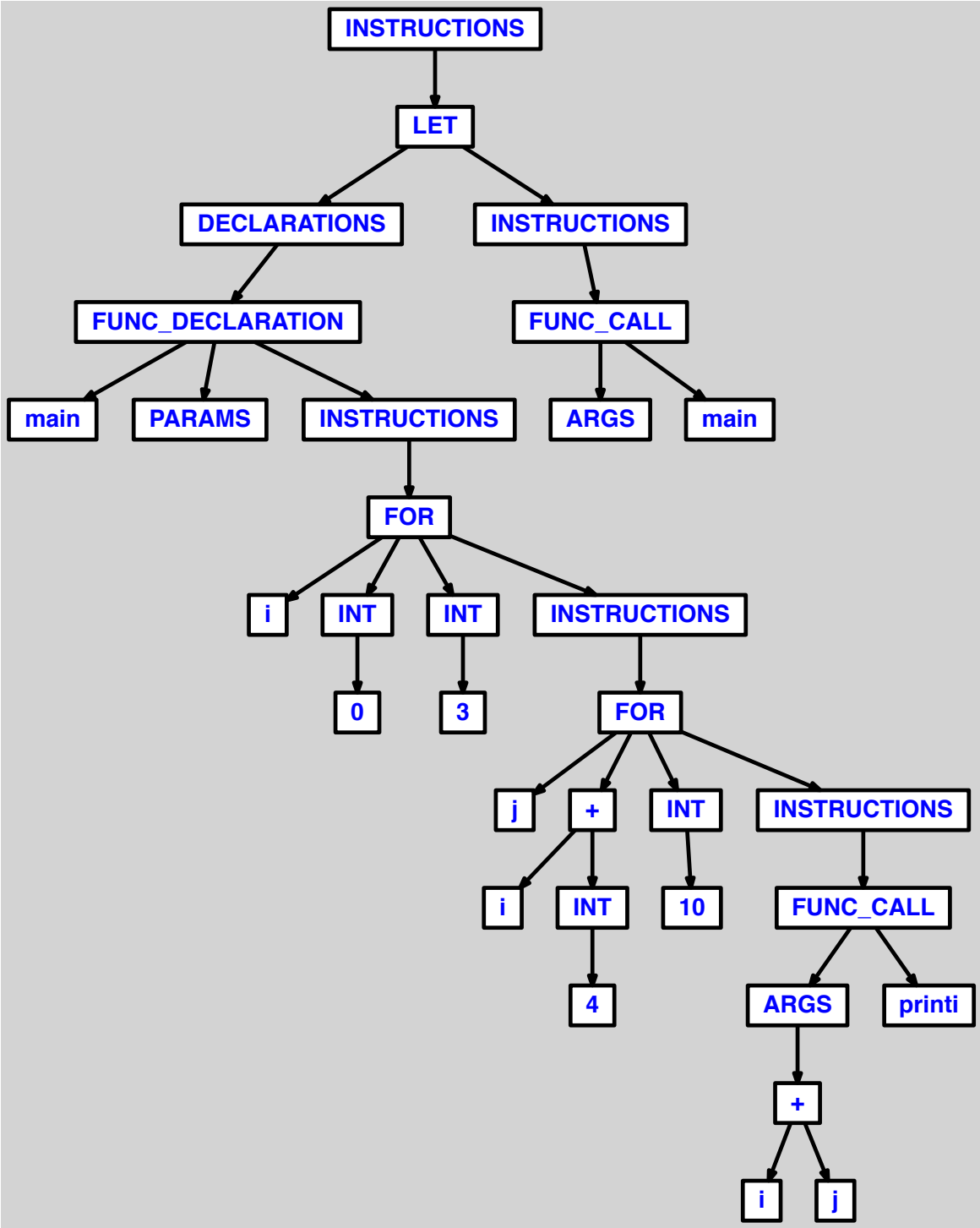
7.2.6 triple-imbrication de <for>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
INT
INT
INSTRUCTIONS
4
7
FOR
k
INT
INT
INSTRUCTIONS
8
10
FUNC_CALL
ARGS
printi
+
+
k
i
j

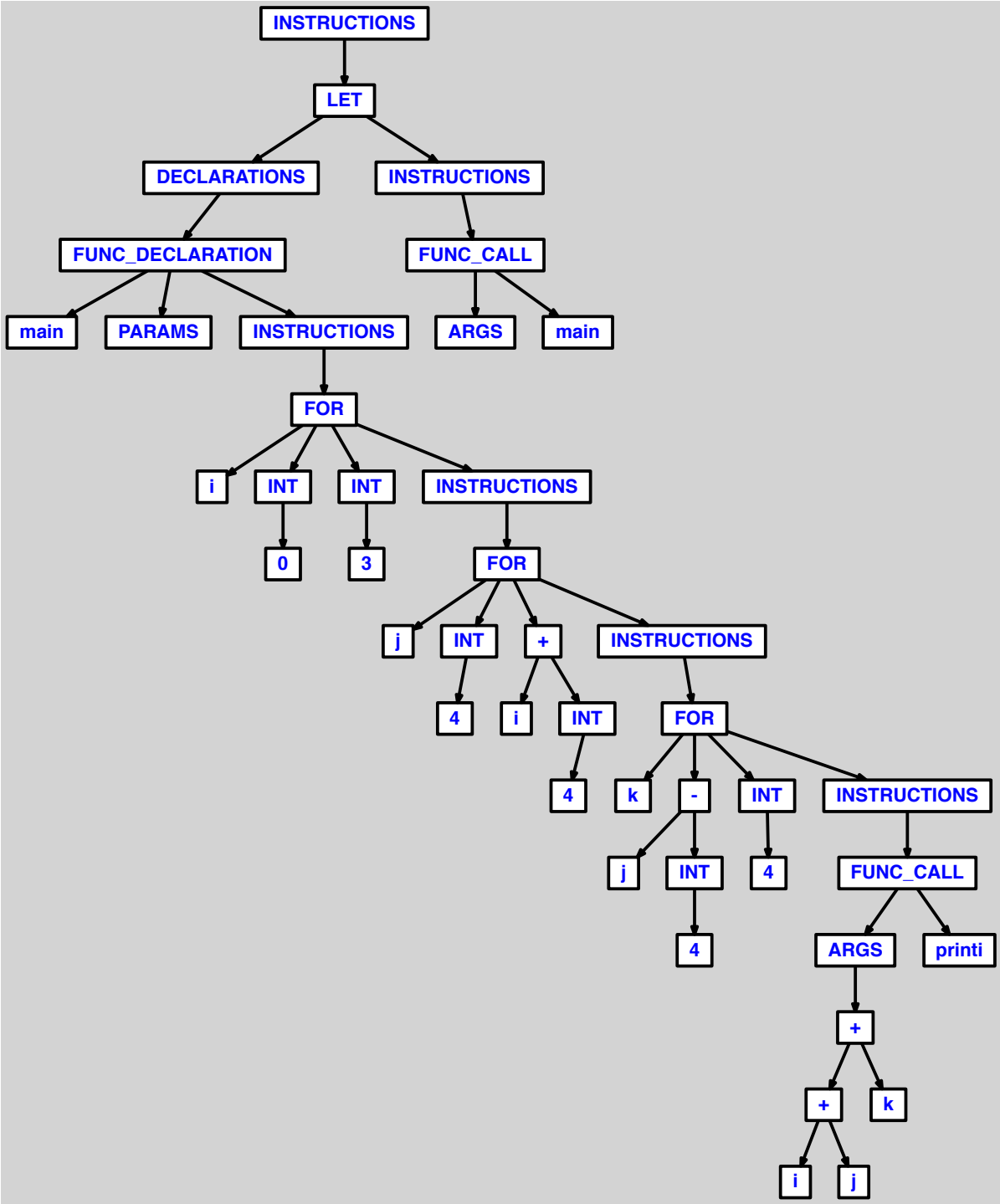
7.2.7 <for> avec reutilisation de compteur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
+
INT
INSTRUCTIONS
i
INT
10
FUNC_CALL
4
ARGS
printi
+
i
j

7.2.8 <for> avec reutilisation de compteurs

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

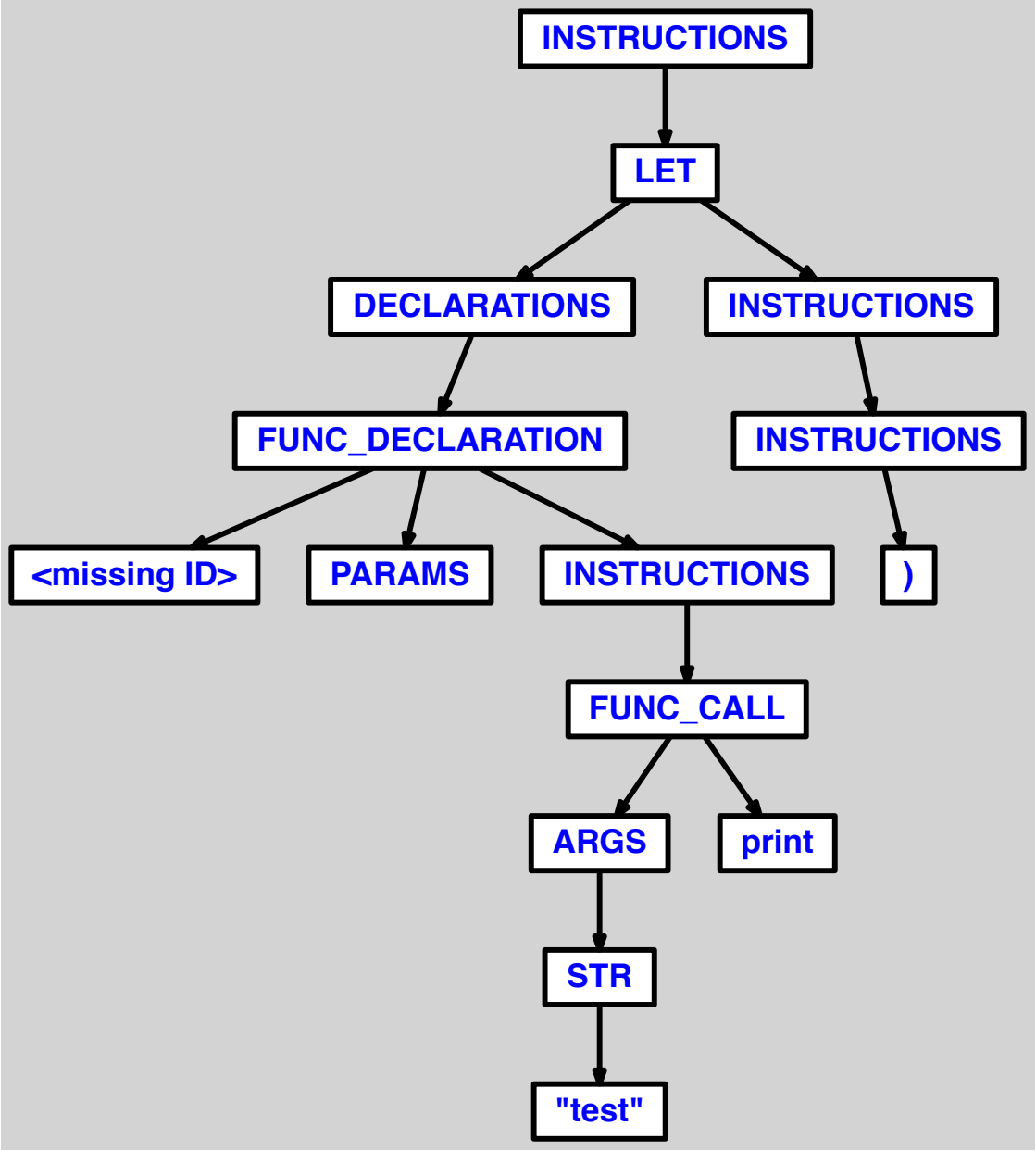

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FOR
i
INT
INT
INSTRUCTIONS
0
3
FOR
j
INT
+
INSTRUCTIONS
4
i
INT
FOR
4
k
-
INT
INSTRUCTIONS
j
INT
4
FUNC_CALL
4
ARGS
printi
+
+
k
i
j

8 function

8.1 KO

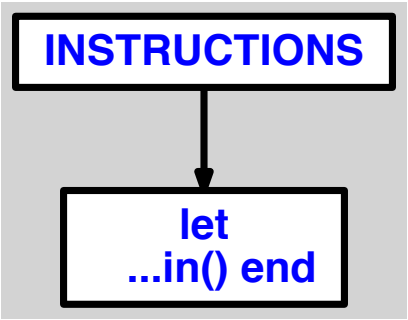
8.1.1 fonction identifiee par caractere special

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
INSTRUCTIONS
<missing ID>
PARAMS
INSTRUCTIONS
)
FUNC_CALL
ARGS
print
STR
"test"

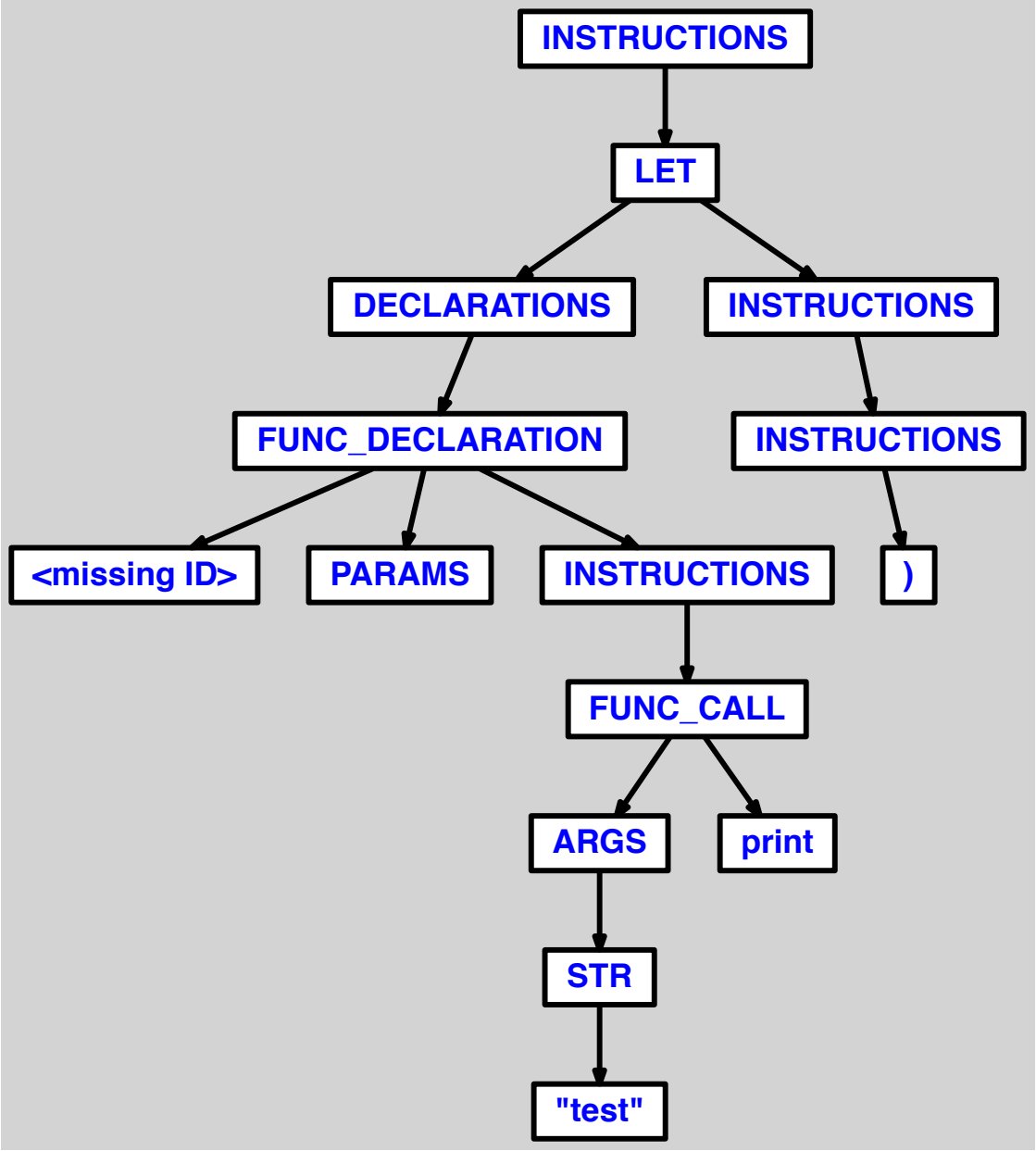
8.1.2 oubli de <function>

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


INSTRUCTIONS
let
...in() end

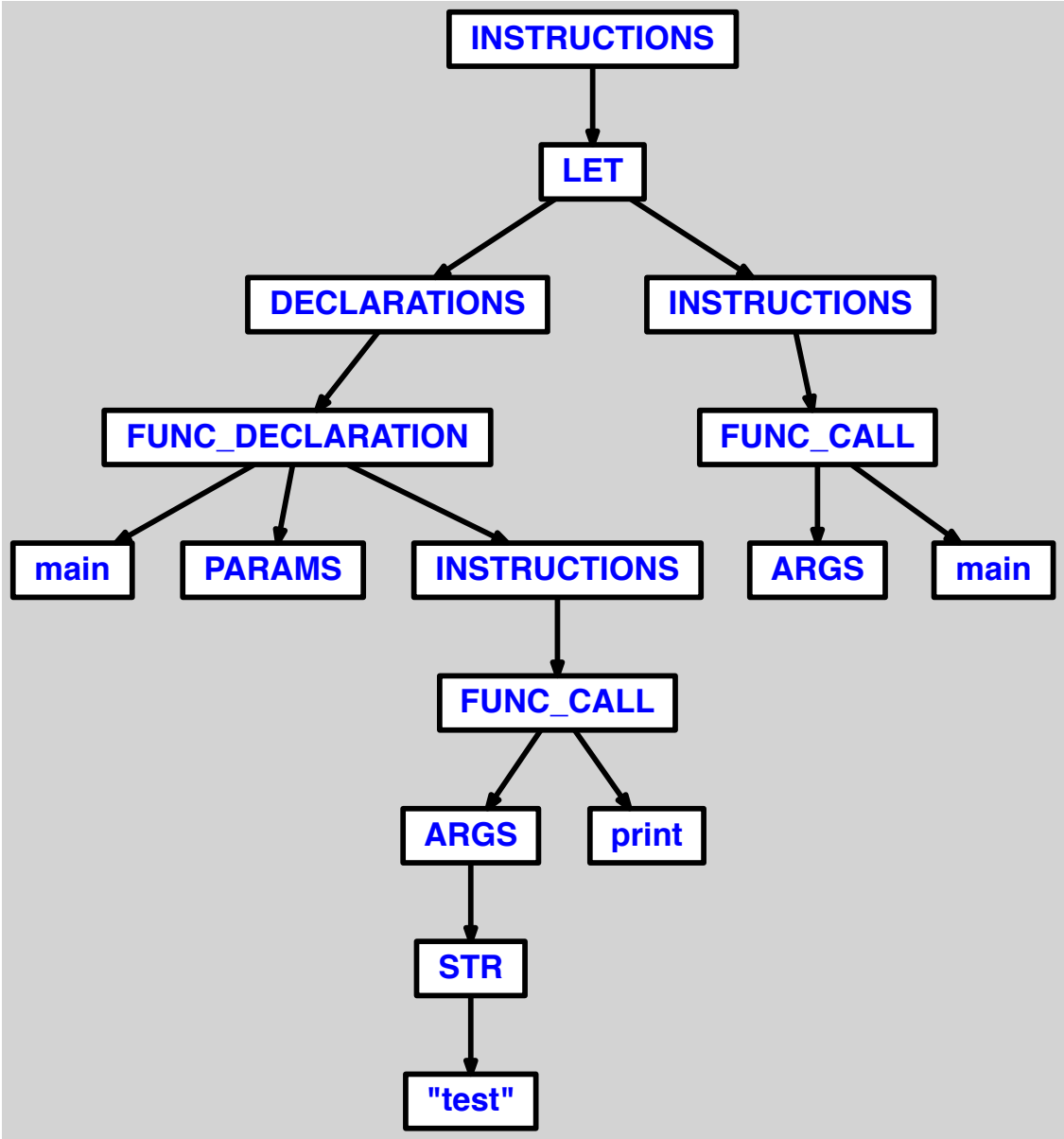
8.1.3 fonction sans identifiant

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
INSTRUCTIONS
<missing ID>
PARAMS
INSTRUCTIONS
)
FUNC_CALL
ARGS
print
STR
"test"

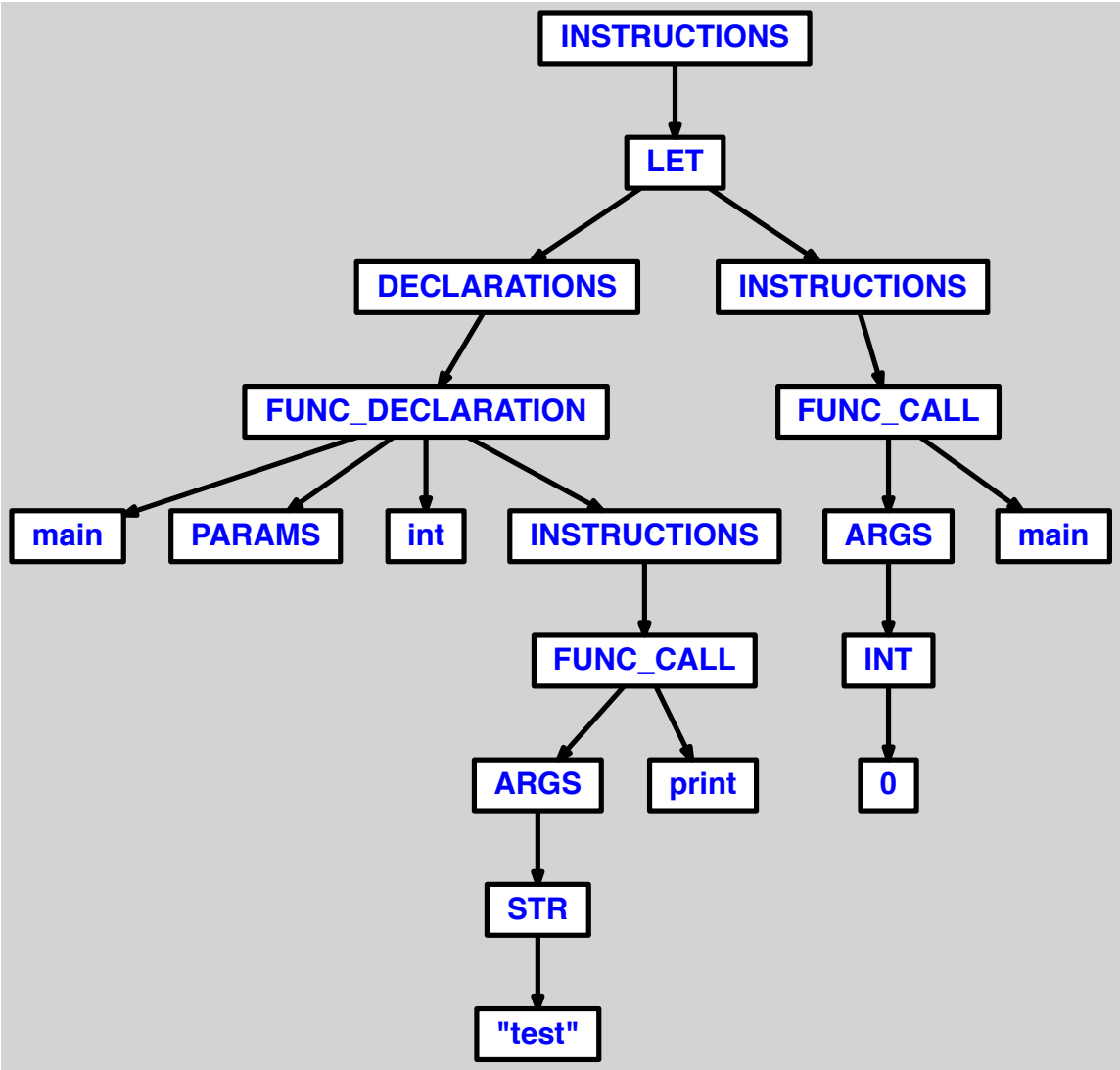
8.1.4 oubli de <(>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

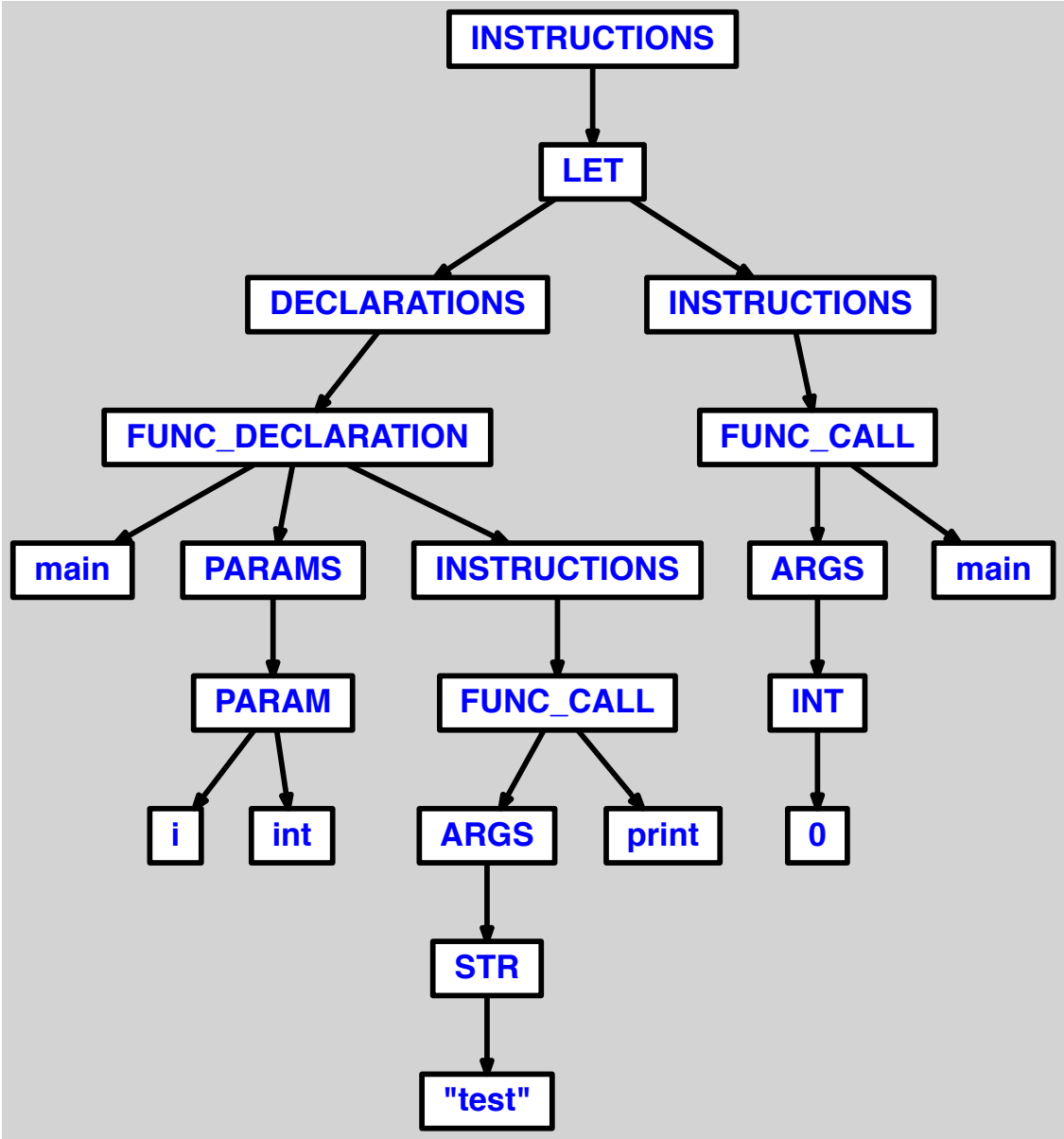
8.1.5 oubli d'identifiant de parametre

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
int
INSTRUCTIONS
ARGS
main
FUNC_CALL
INT
ARGS
print
0
STR
"test"

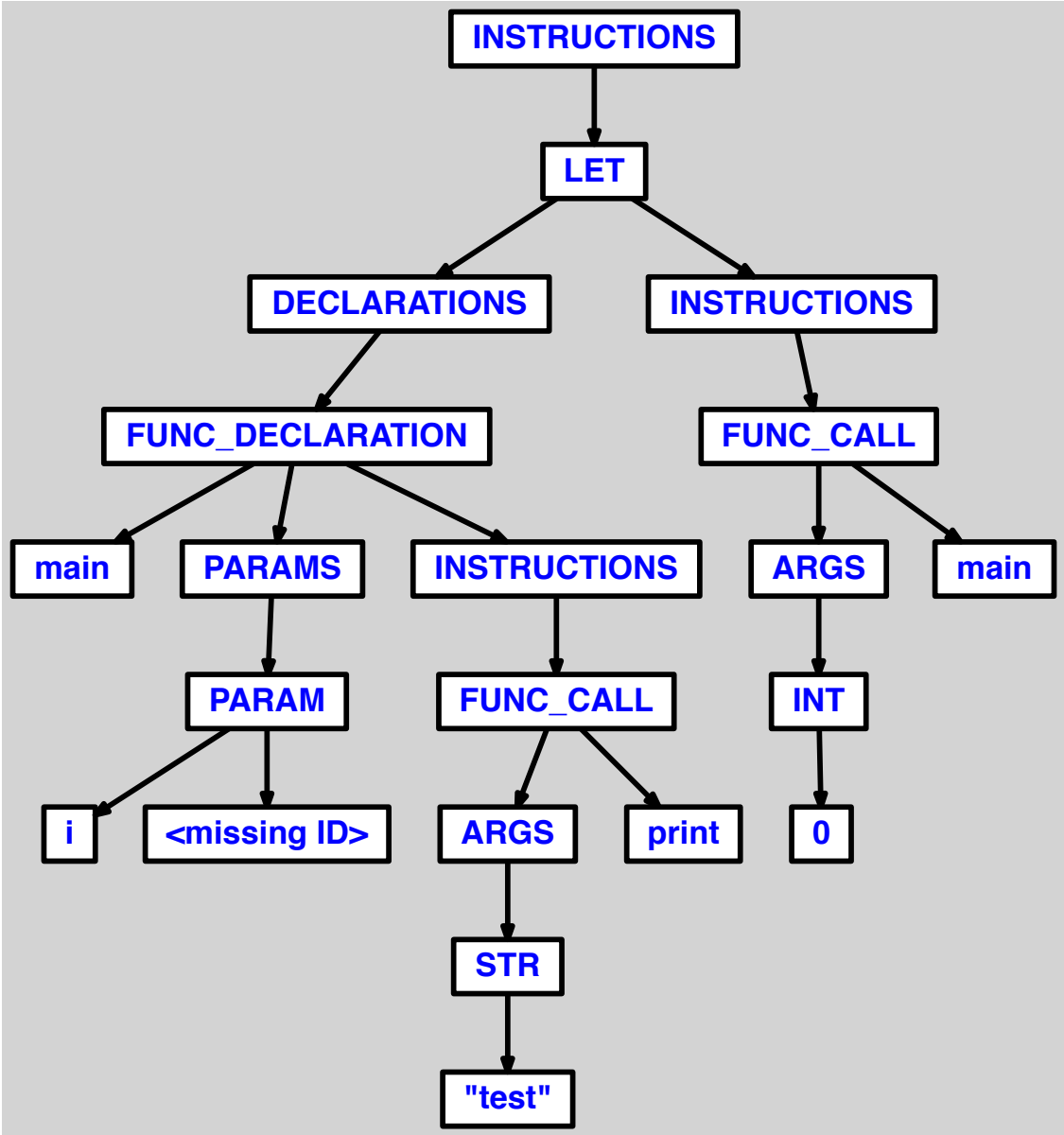
8.1.6 oubli de < :> pour parametre

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
FUNC_CALL
INT
i
int
ARGS
print
0
STR
"test"

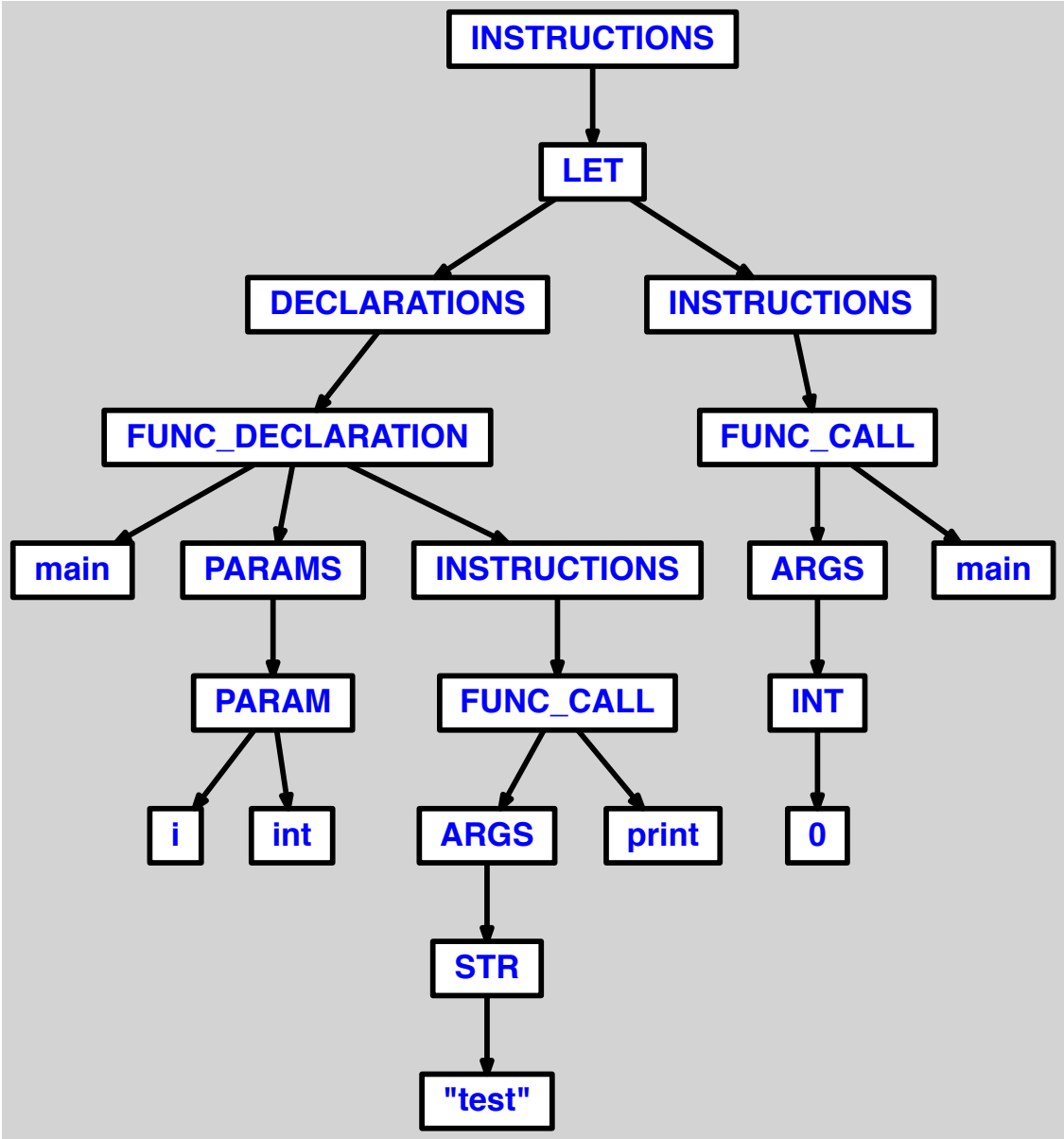
8.1.7 oubli de type de parametre

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
FUNC_CALL
INT
i
<missing ID>
ARGS
print
0
STR
"test"

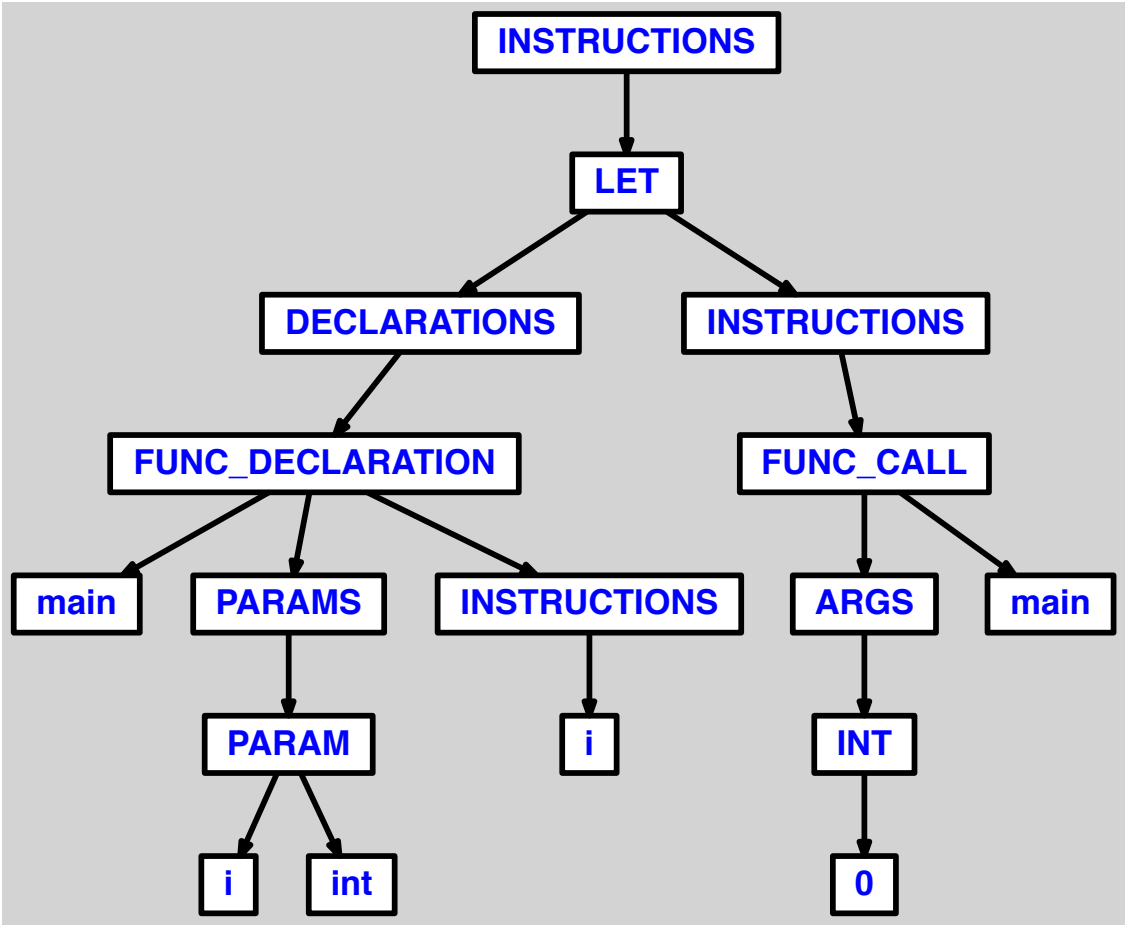
8.1.8 oubli de <)>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
FUNC_CALL
INT
i
int
ARGS
print
0
STR
"test"

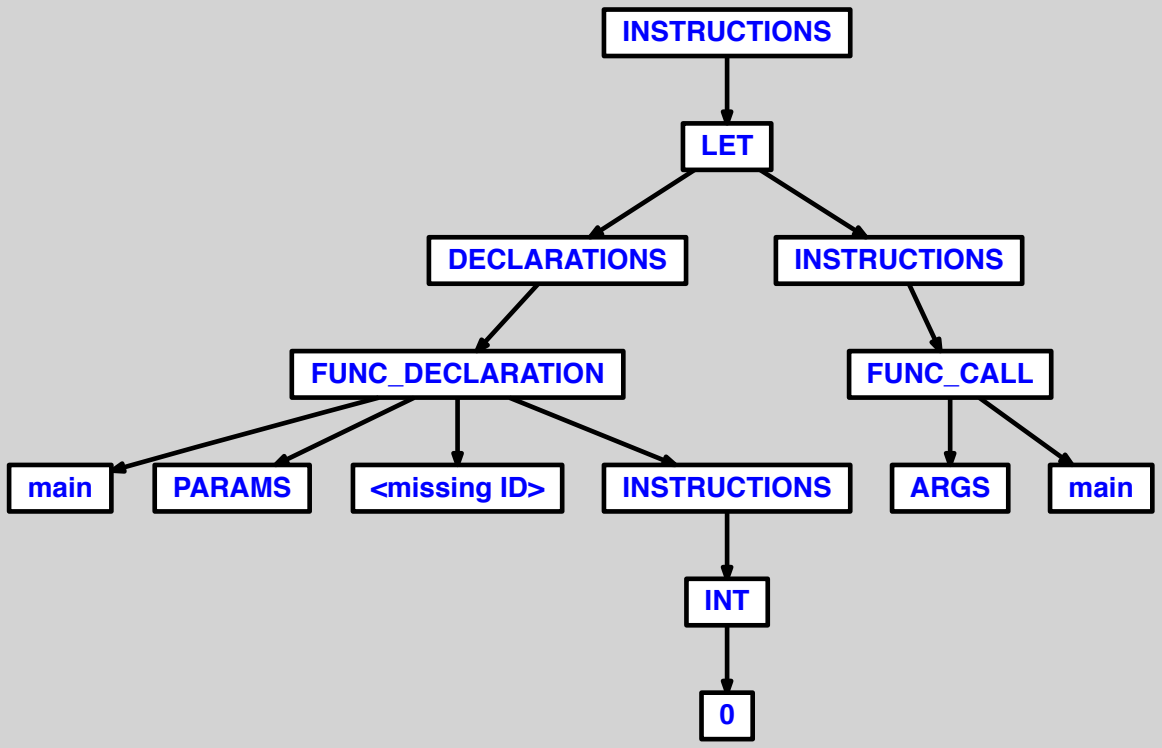
8.1.9 oubli de < :> pour type de fonction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
i
INT
i
int
0

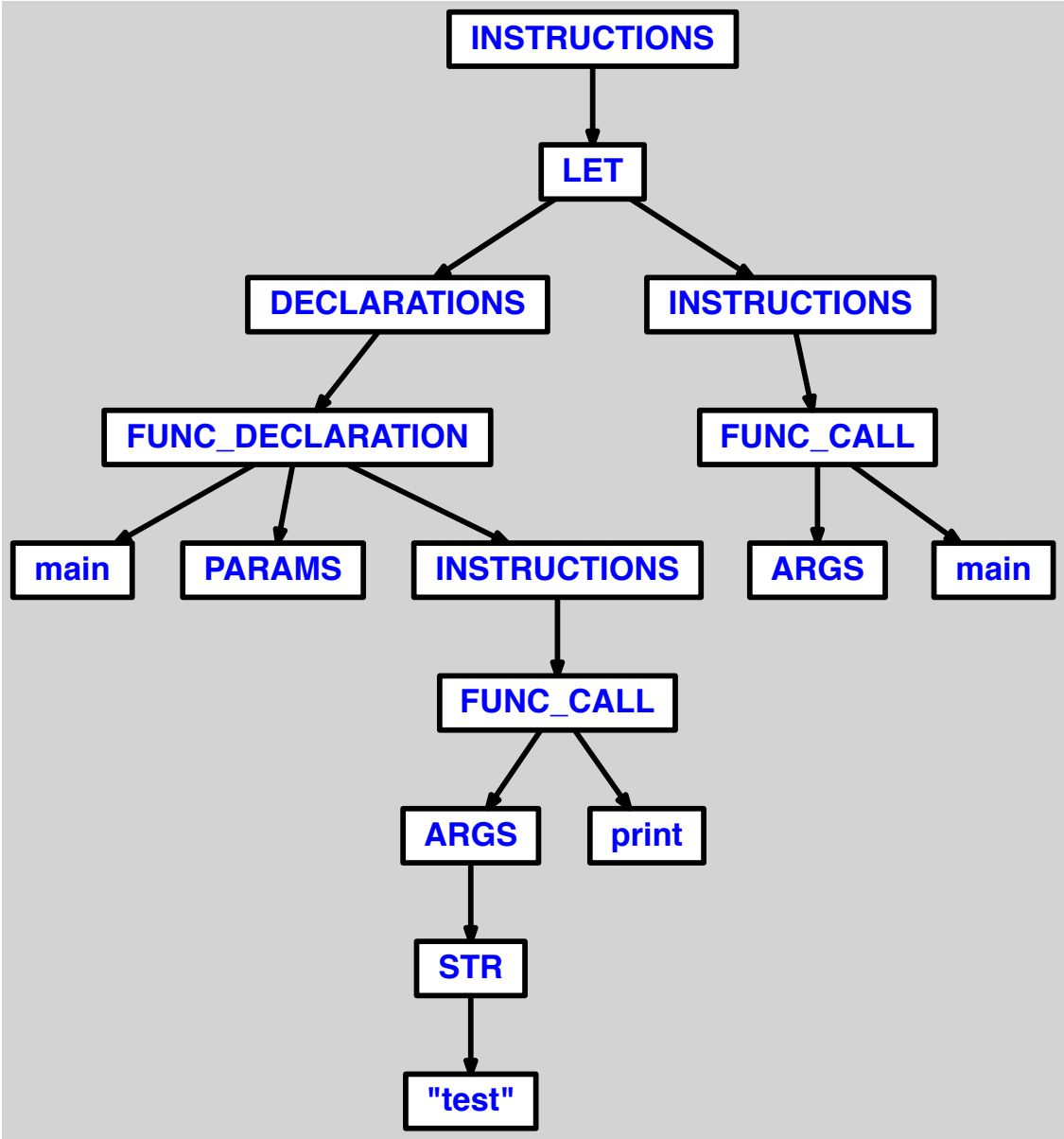
8.1.10 < :> pour type de fonction present mais pas de type

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
<missing ID>
INSTRUCTIONS
ARGS
main
INT
0

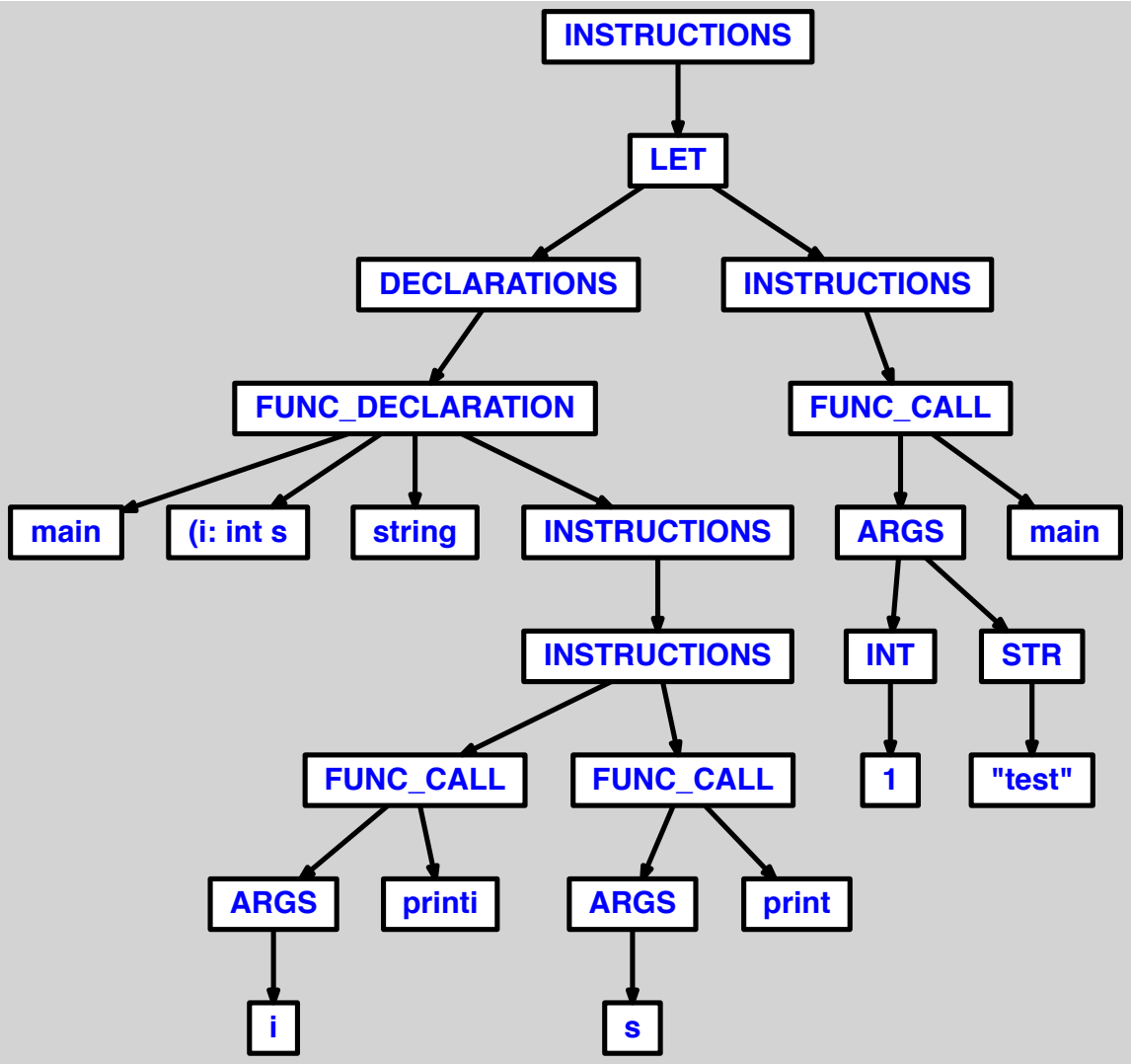
8.1.11 oubli de <=>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

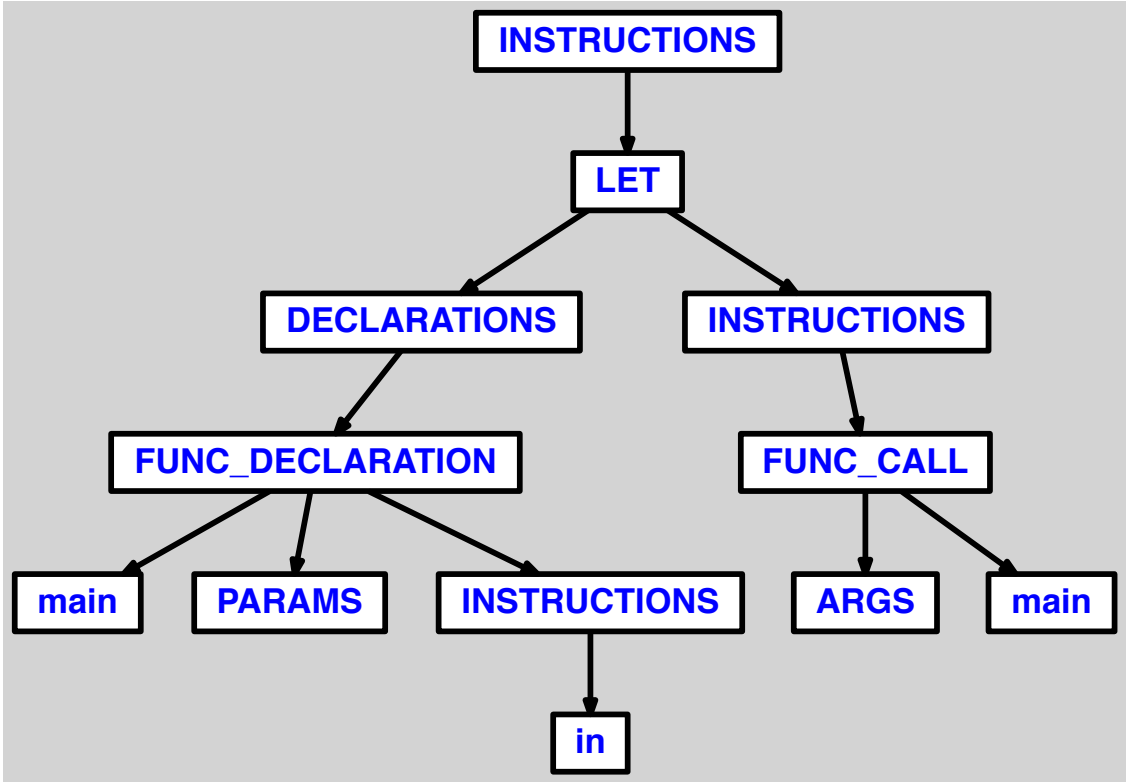
8.1.12 parametres mal separes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
(i: int s
string
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
INT
STR
FUNC_CALL
FUNC_CALL
1
"test"
ARGS
printi
ARGS
print
i
s

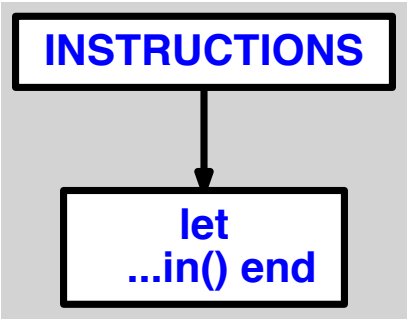
8.1.13 fonction sans instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
in

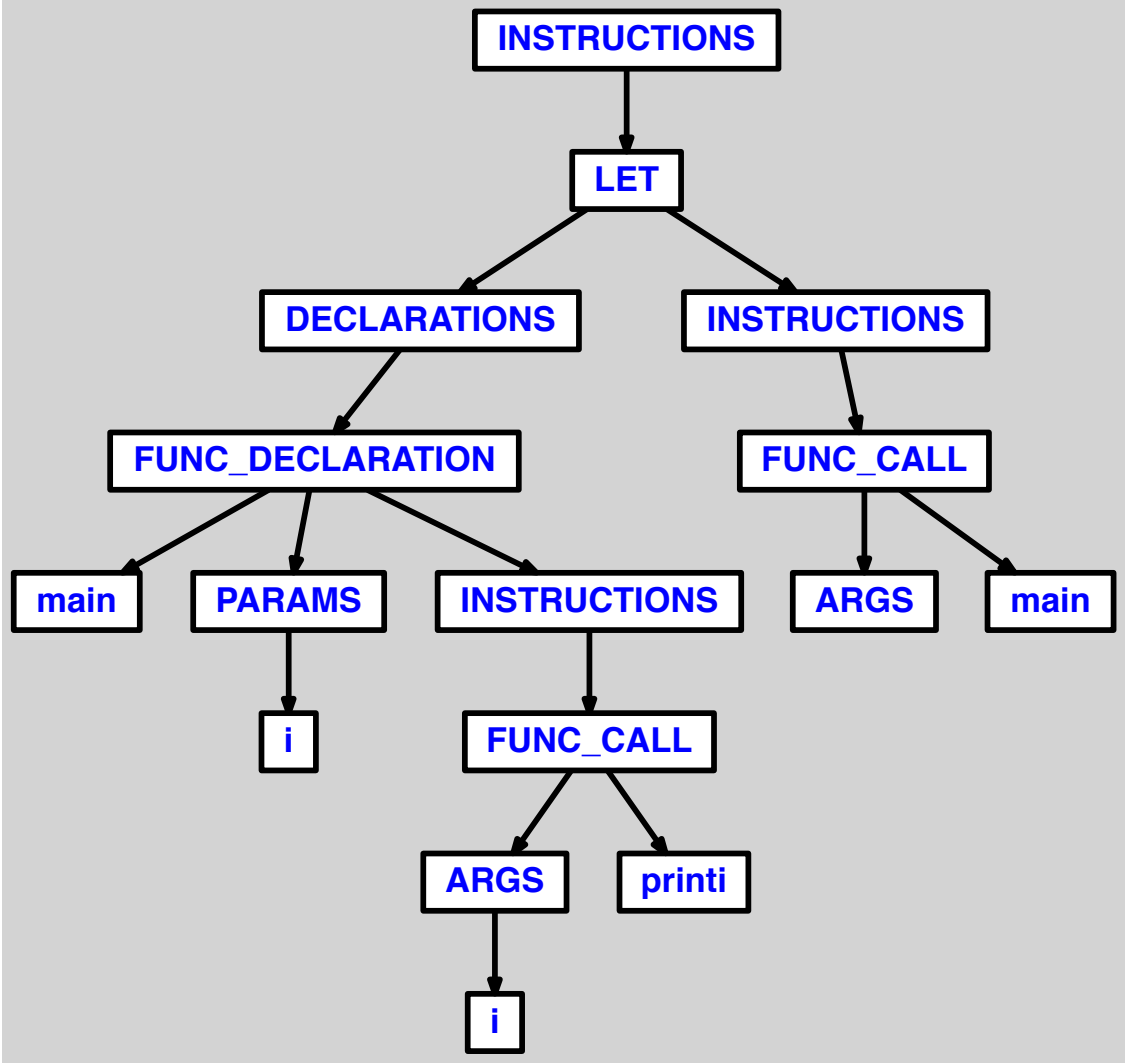
8.1.14 <function> mal ecrit

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


INSTRUCTIONS
let
...in() end

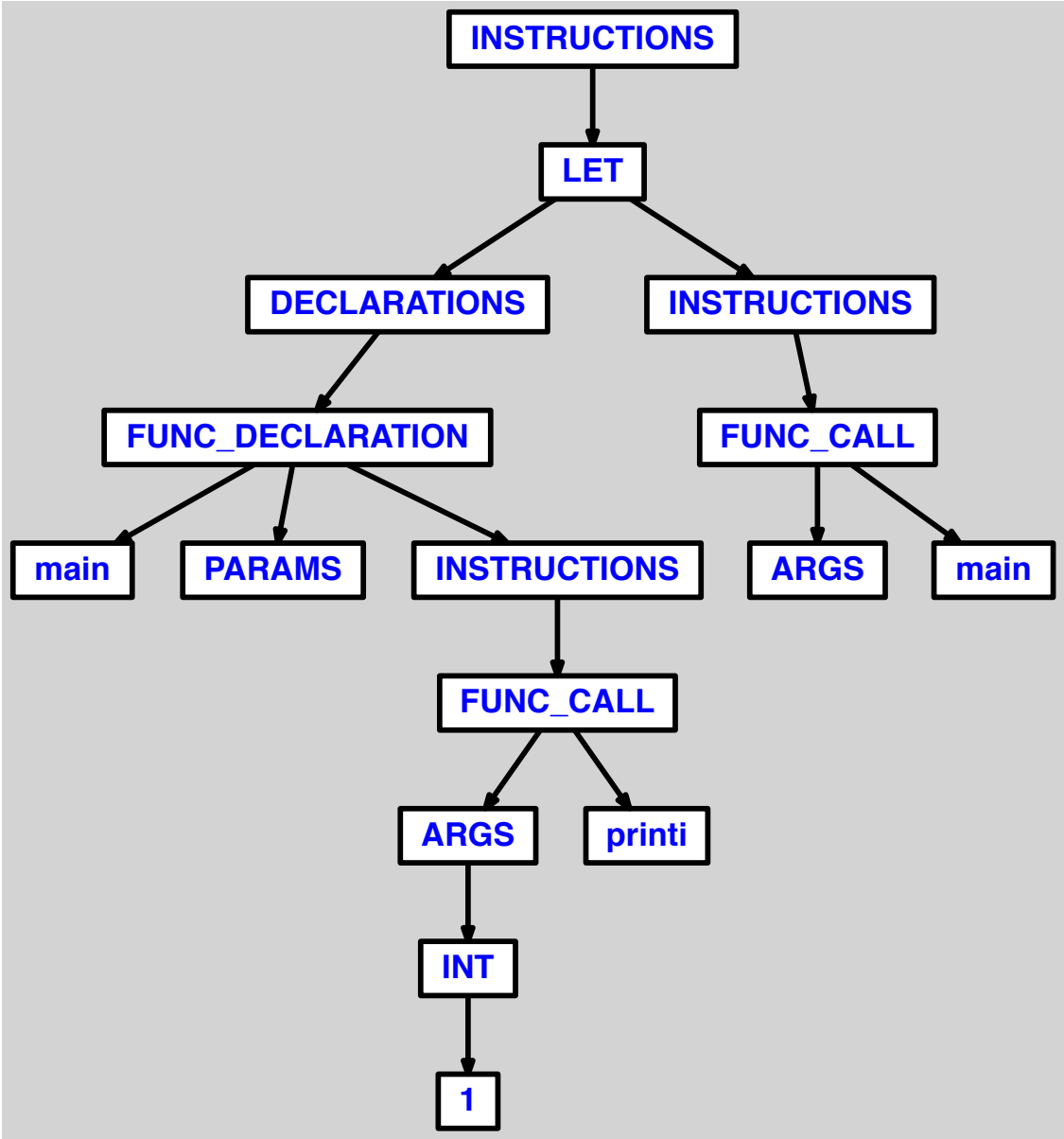
8.1.15 fonction avec identifiant seulement en parametre

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
i
FUNC_CALL
ARGS
printi
i

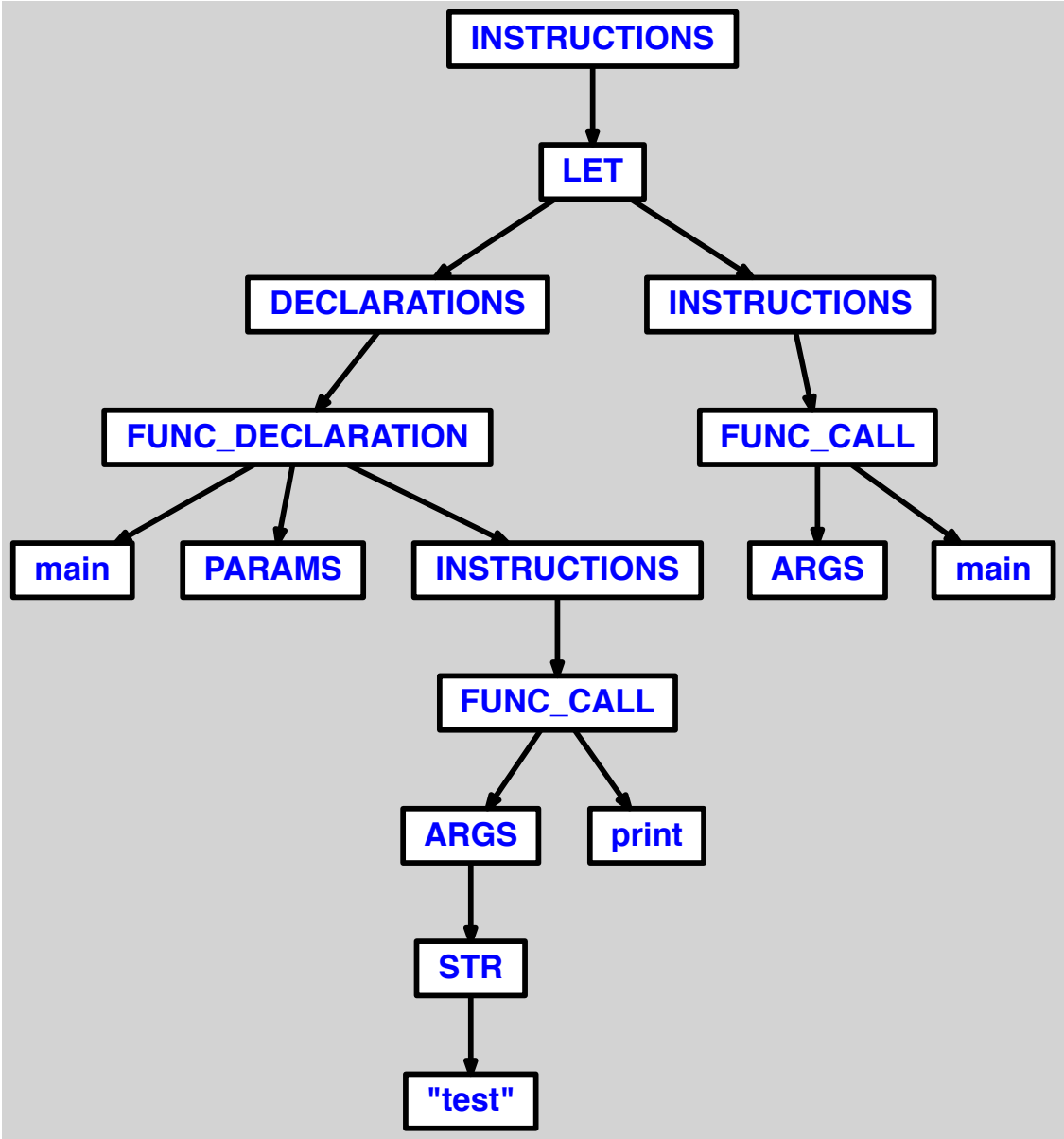
8.1.16 fonction avec entier directement en parametre

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
INT
1

8.1.17 fonction avec chaine directement en parametre

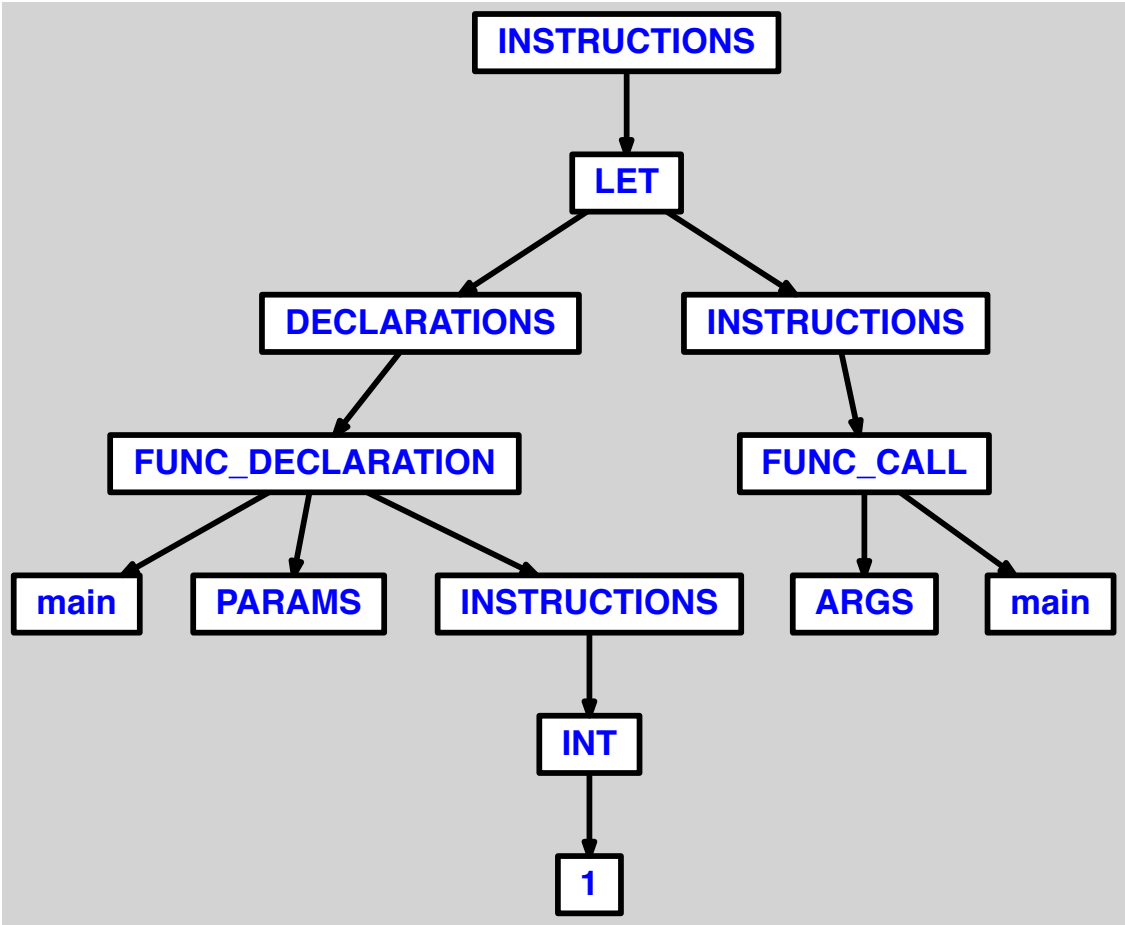
```
let
  function main("test") = print("test")
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

8.2 OK

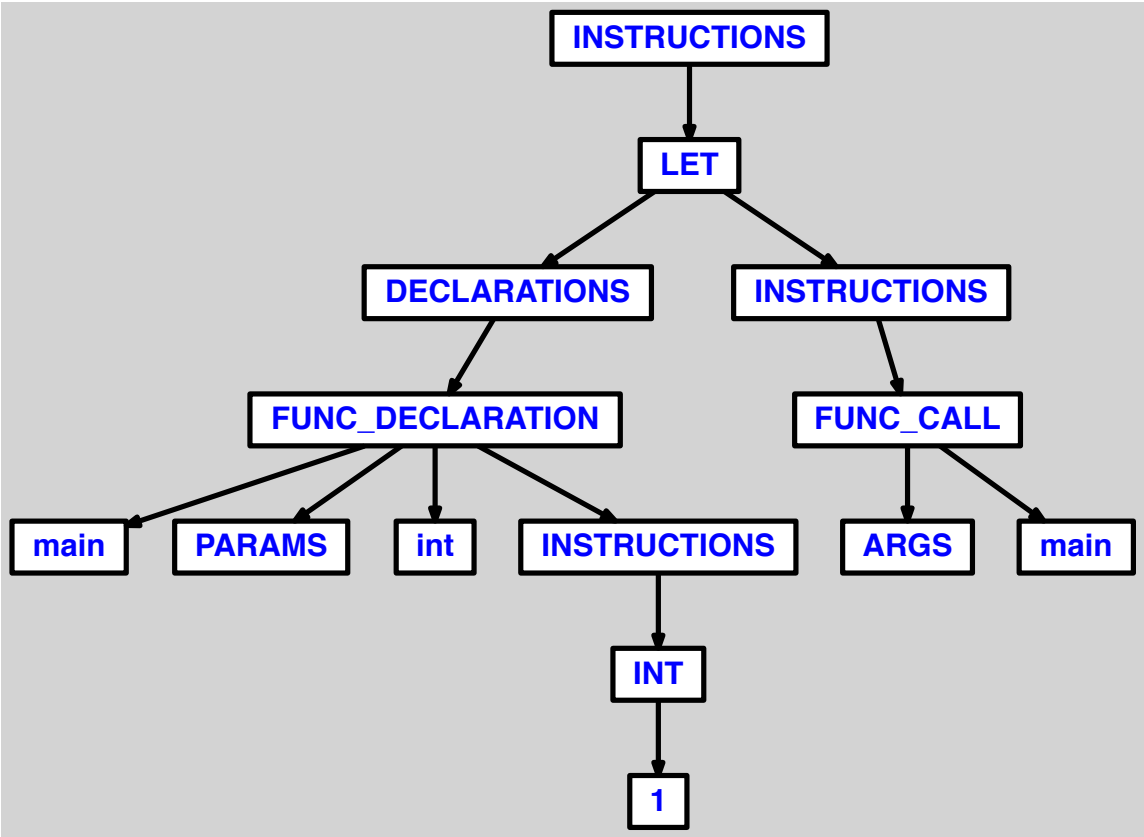
8.2.1 fonction definissant un entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INT
1

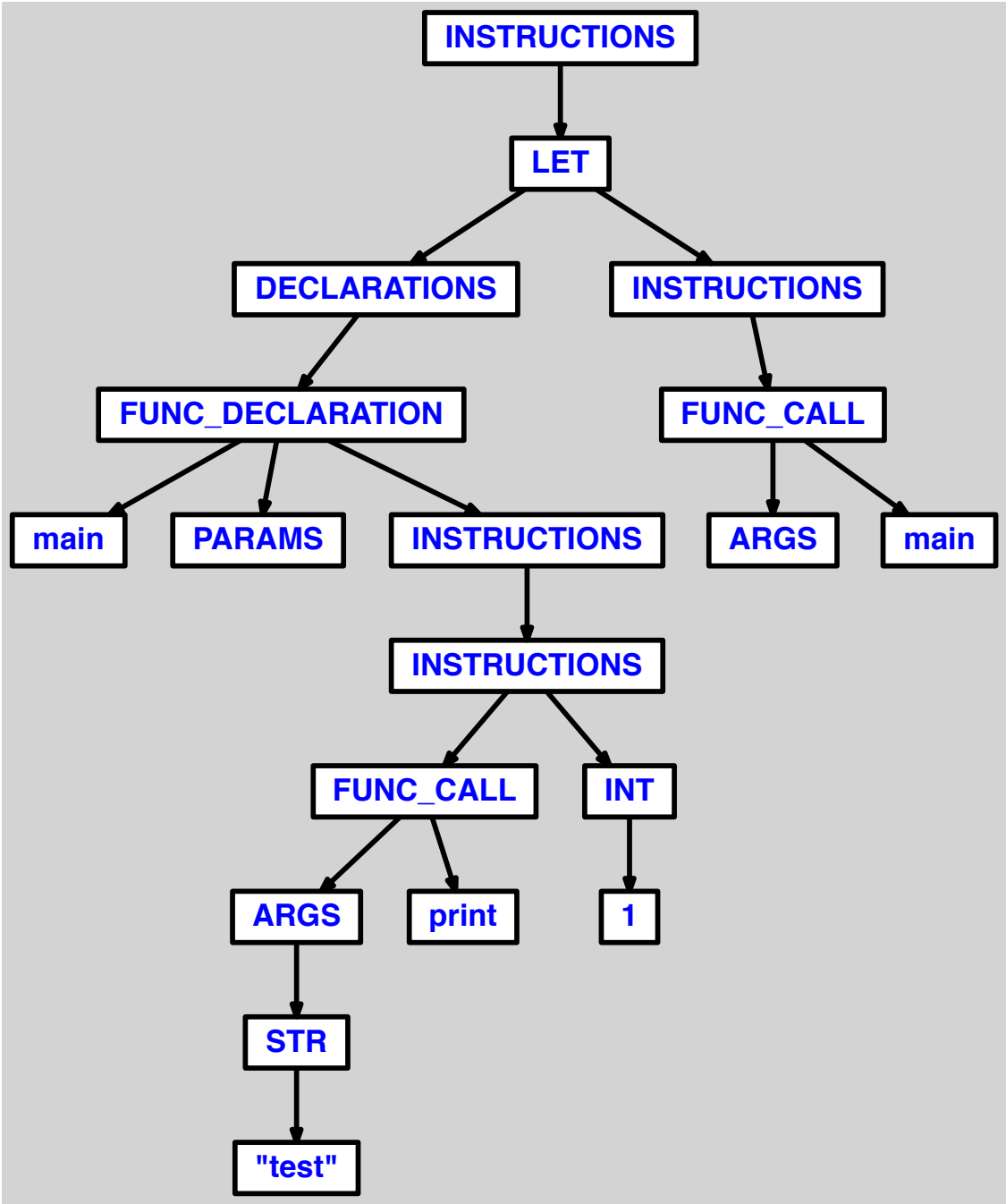
8.2.2 fonction simple, avec declaration du type de retour <int> et instruction de retour

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
int
INSTRUCTIONS
ARGS
main
INT
1

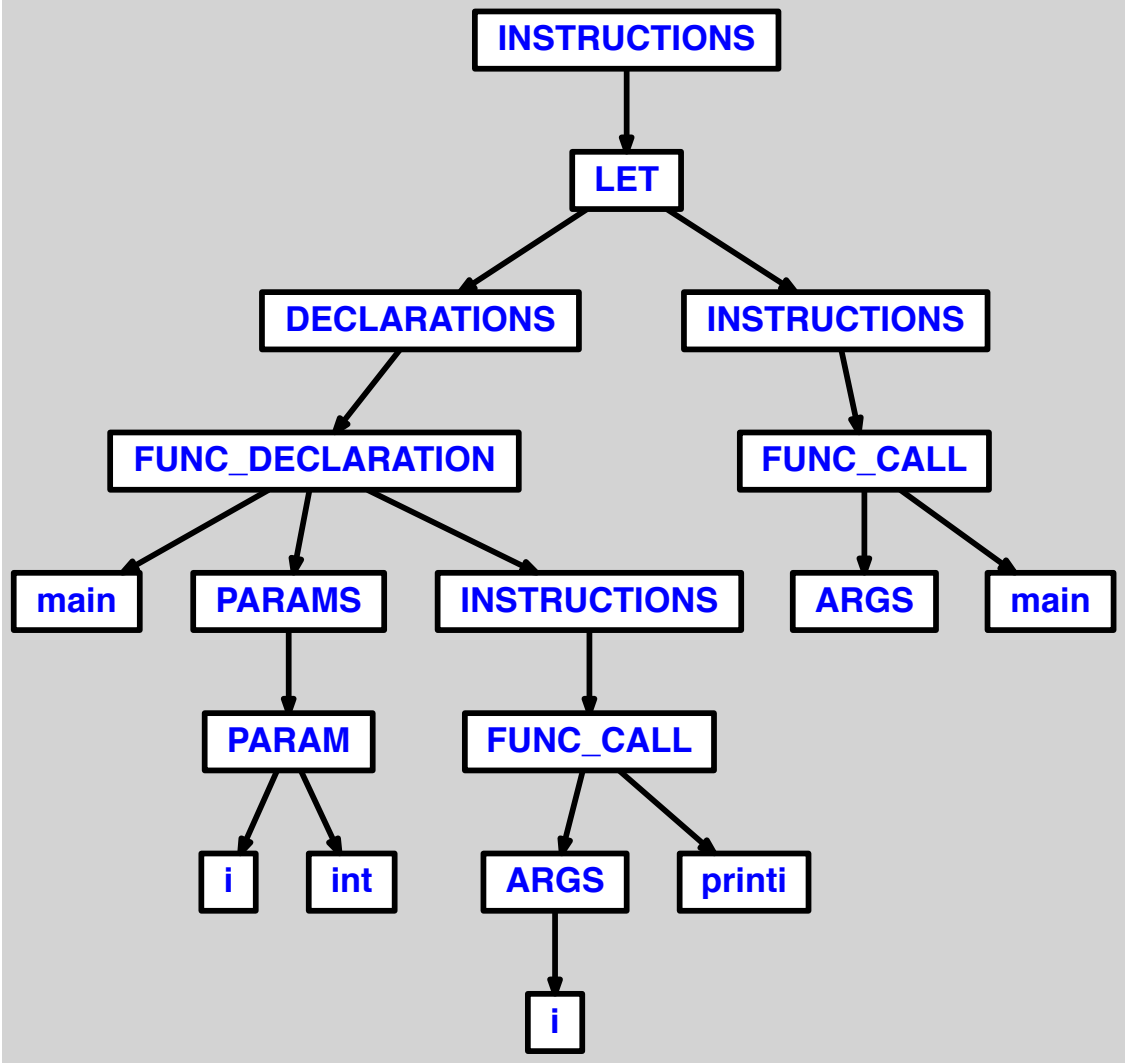
8.2.3 fonction simple, avec instruction simple et instruction retournant un entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
INT
ARGS
print
1
STR
"test"

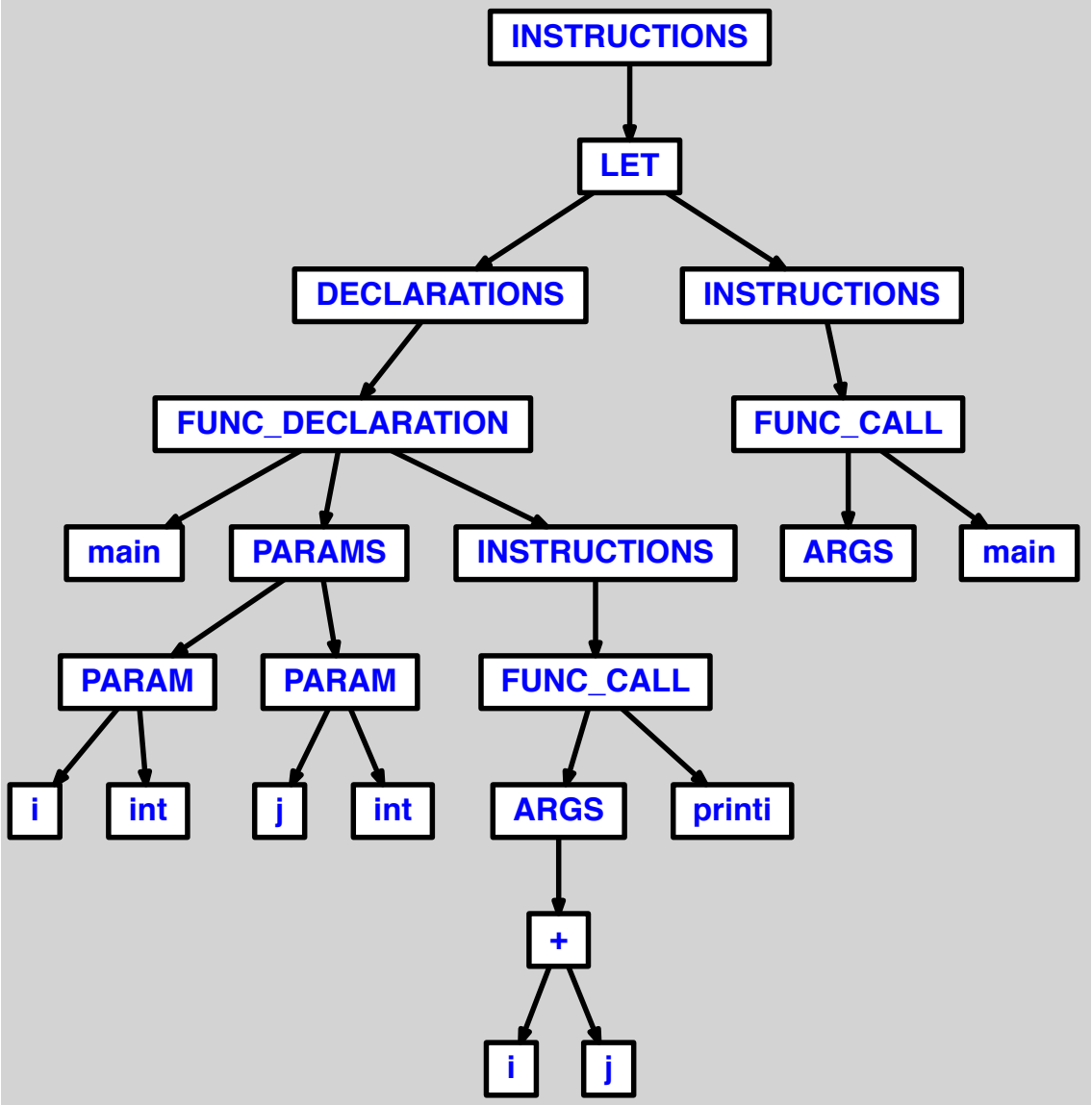
8.2.4 fonction avec 1 parametre entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
FUNC_CALL
i
int
ARGS
printi
i

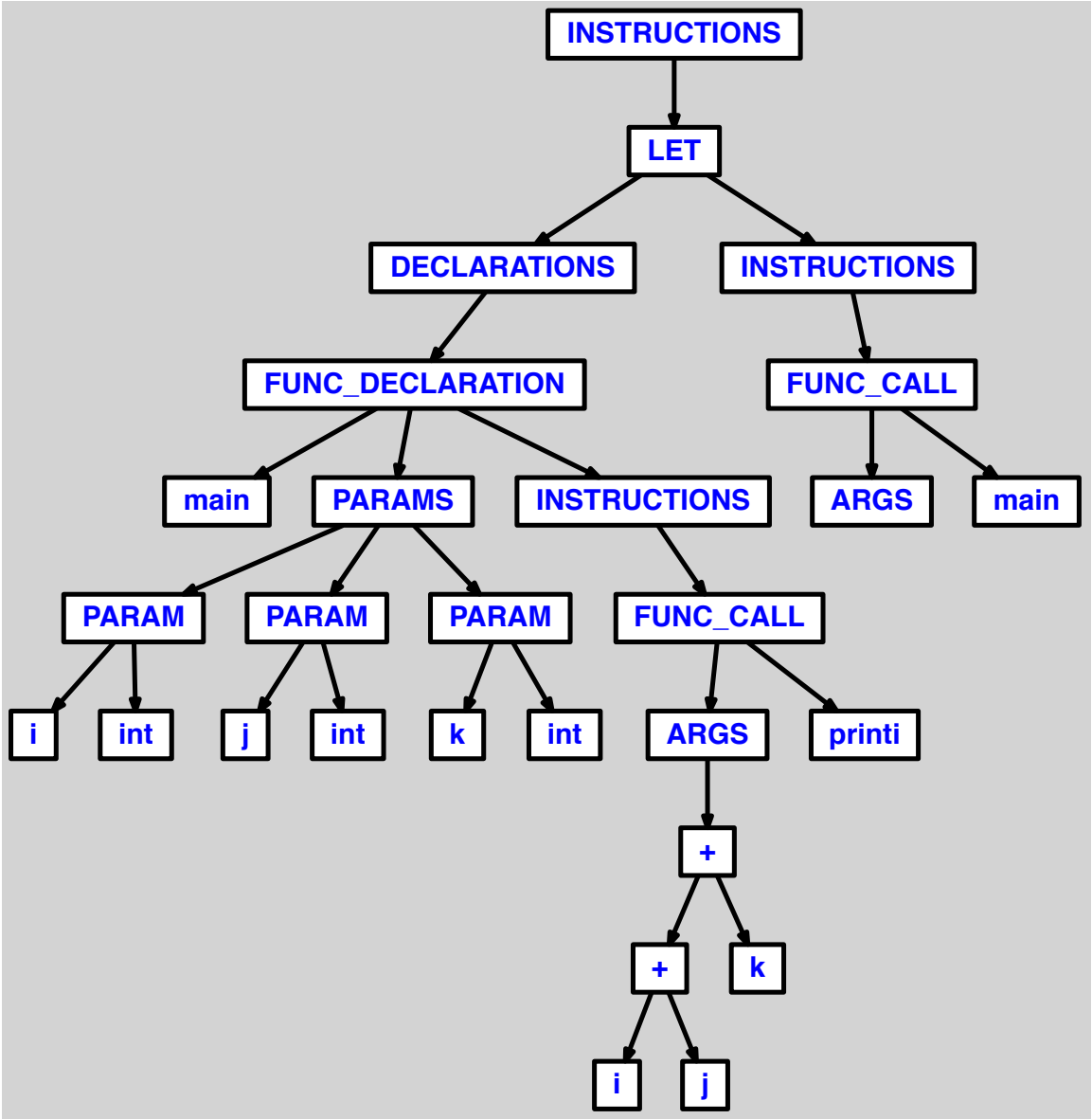
8.2.5 fonction avec 2 parametres entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
FUNC_CALL
i
int
j
int
ARGS
printi
+
i
j

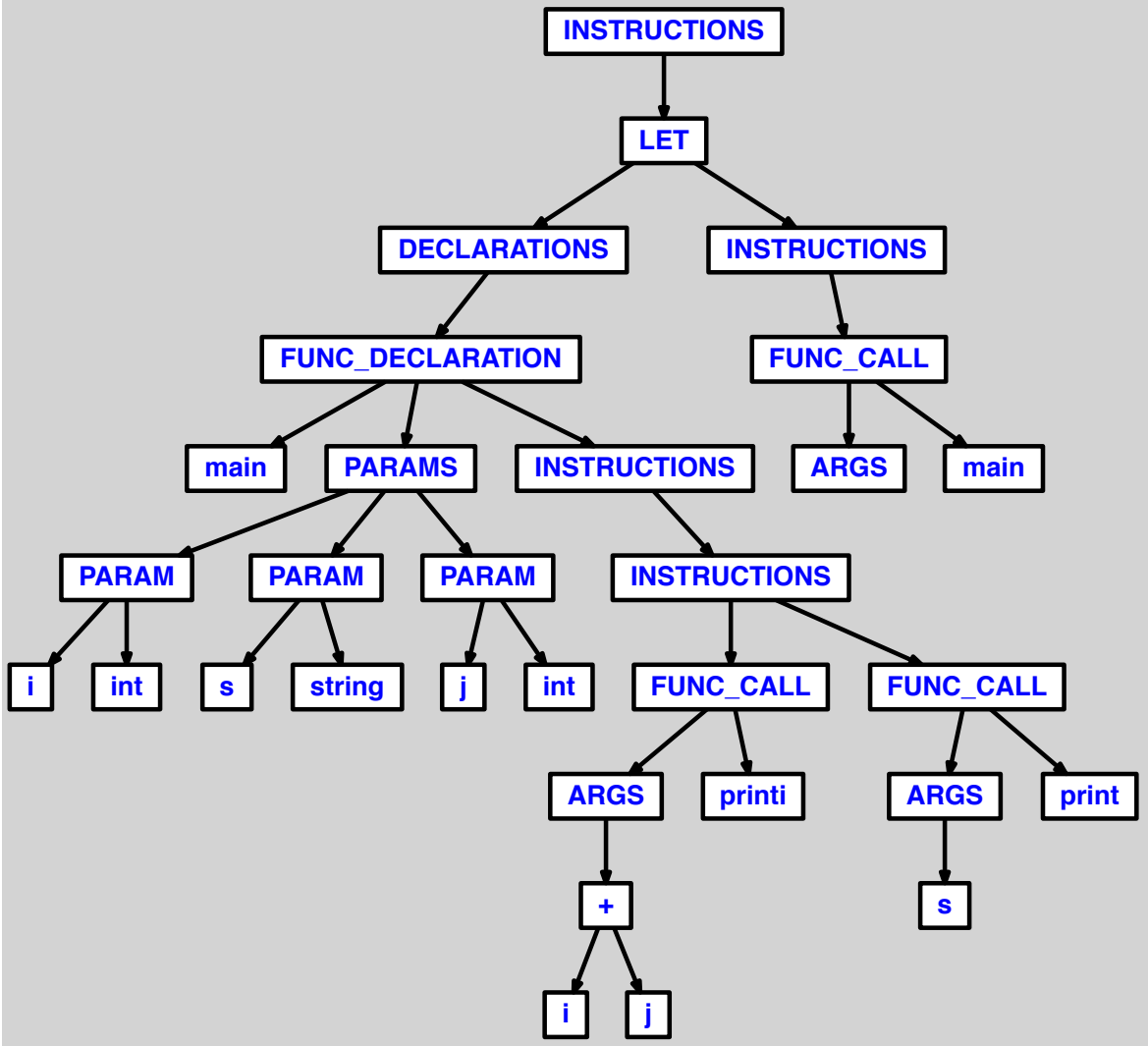
8.2.6 fonction avec 3 parametres entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
PARAM
FUNC_CALL
i
int
j
int
k
int
ARGS
printi
+
+
k
i
j

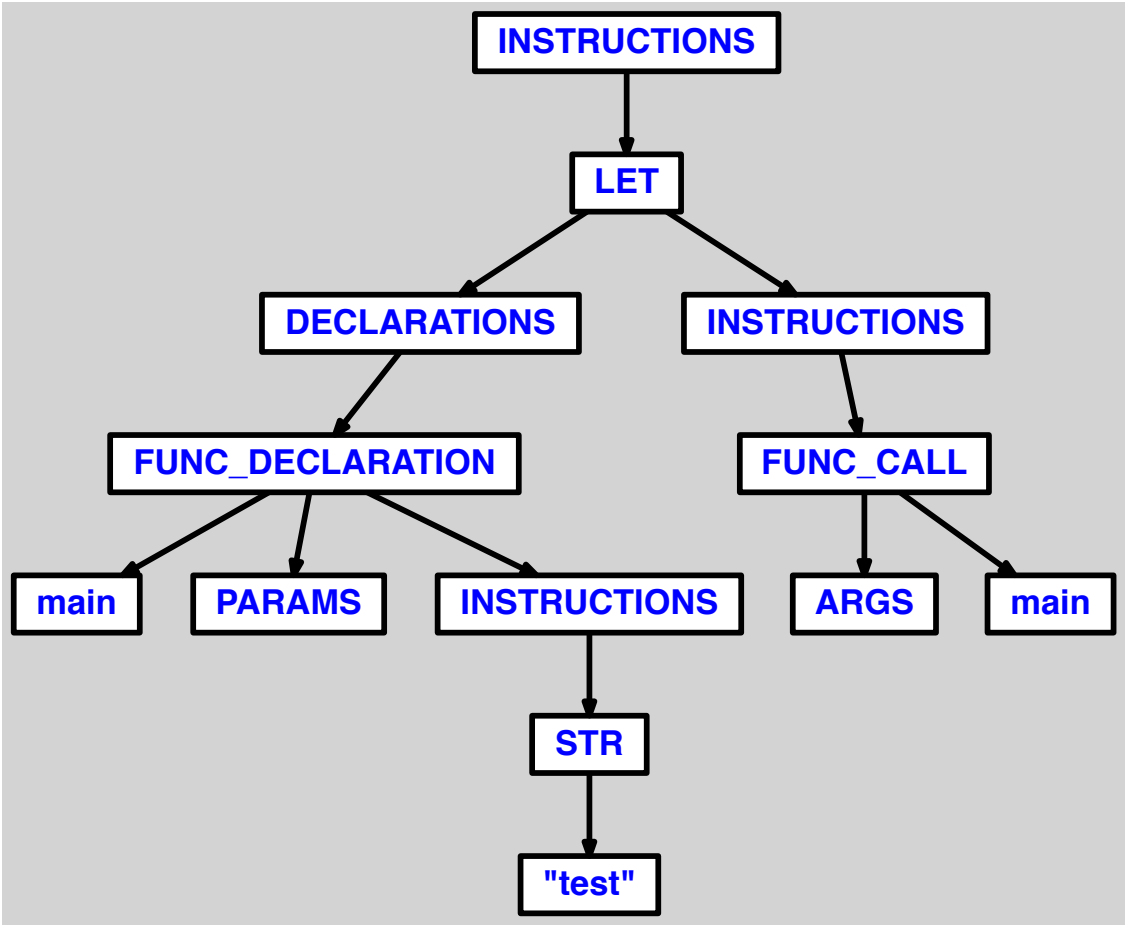
8.2.7 fonction avec 2 parametres entiers et 1 parametre de type <string>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
PARAM
INSTRUCTIONS
i
int
s
string
j
int
FUNC_CALL
FUNC_CALL
ARGS
printi
ARGS
print
+
s
i
j

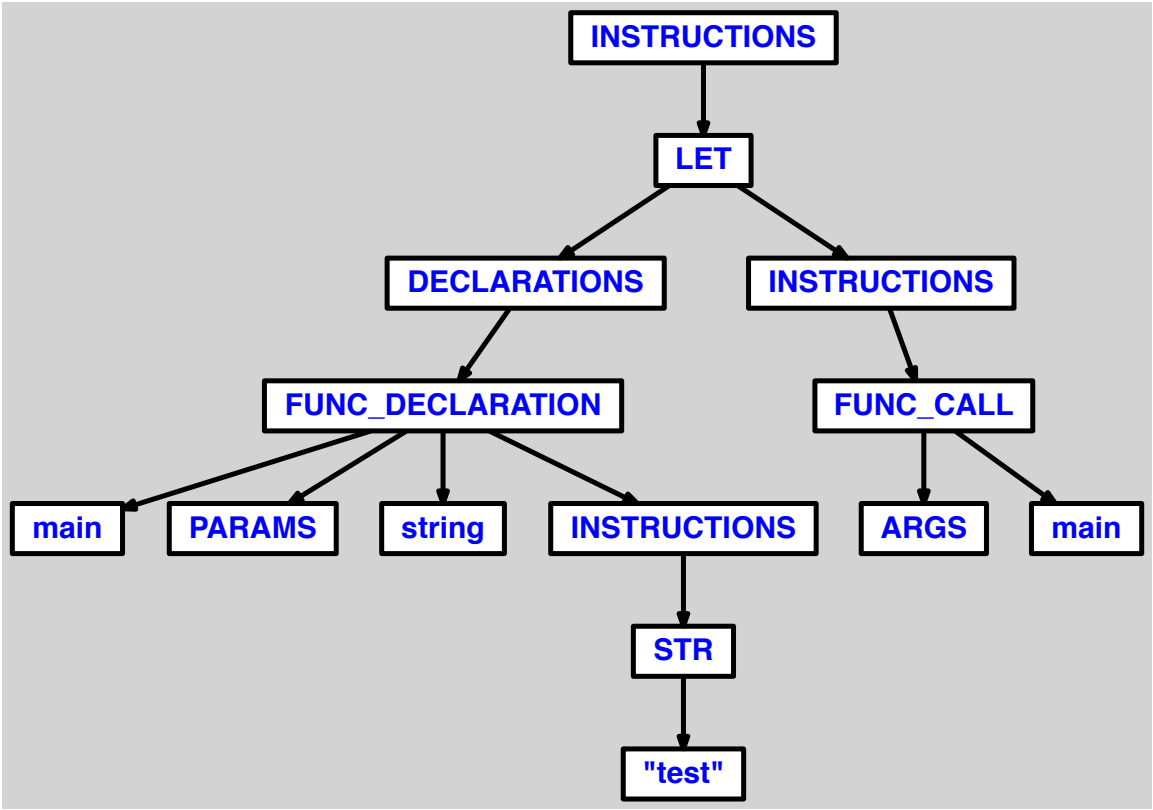
8.2.8 fonction definissant une chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
STR
"test"

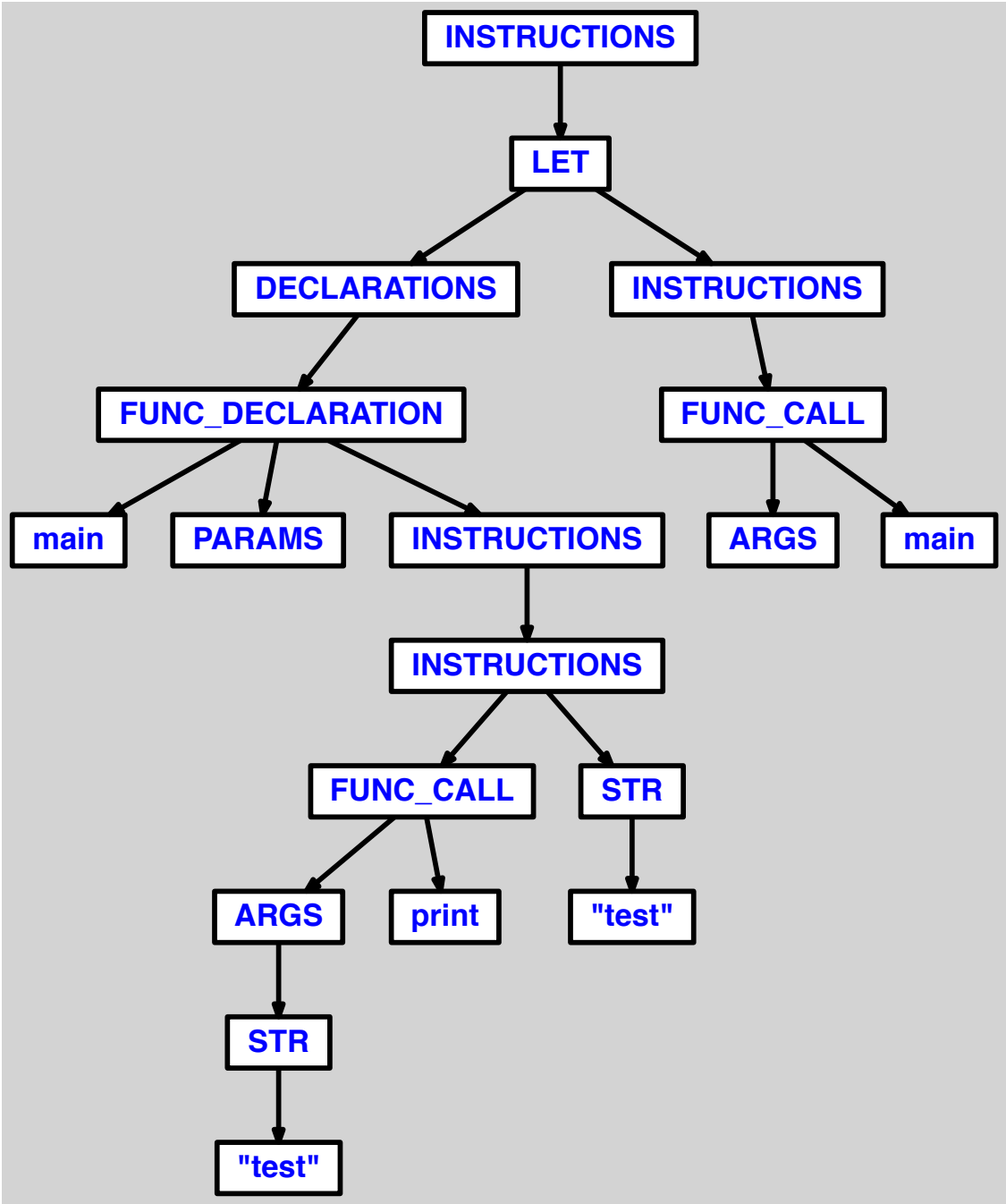
8.2.9 fonction simple, avec declaration du type de retour <string> et instruction de retour

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
string
INSTRUCTIONS
ARGS
main
STR
"test"

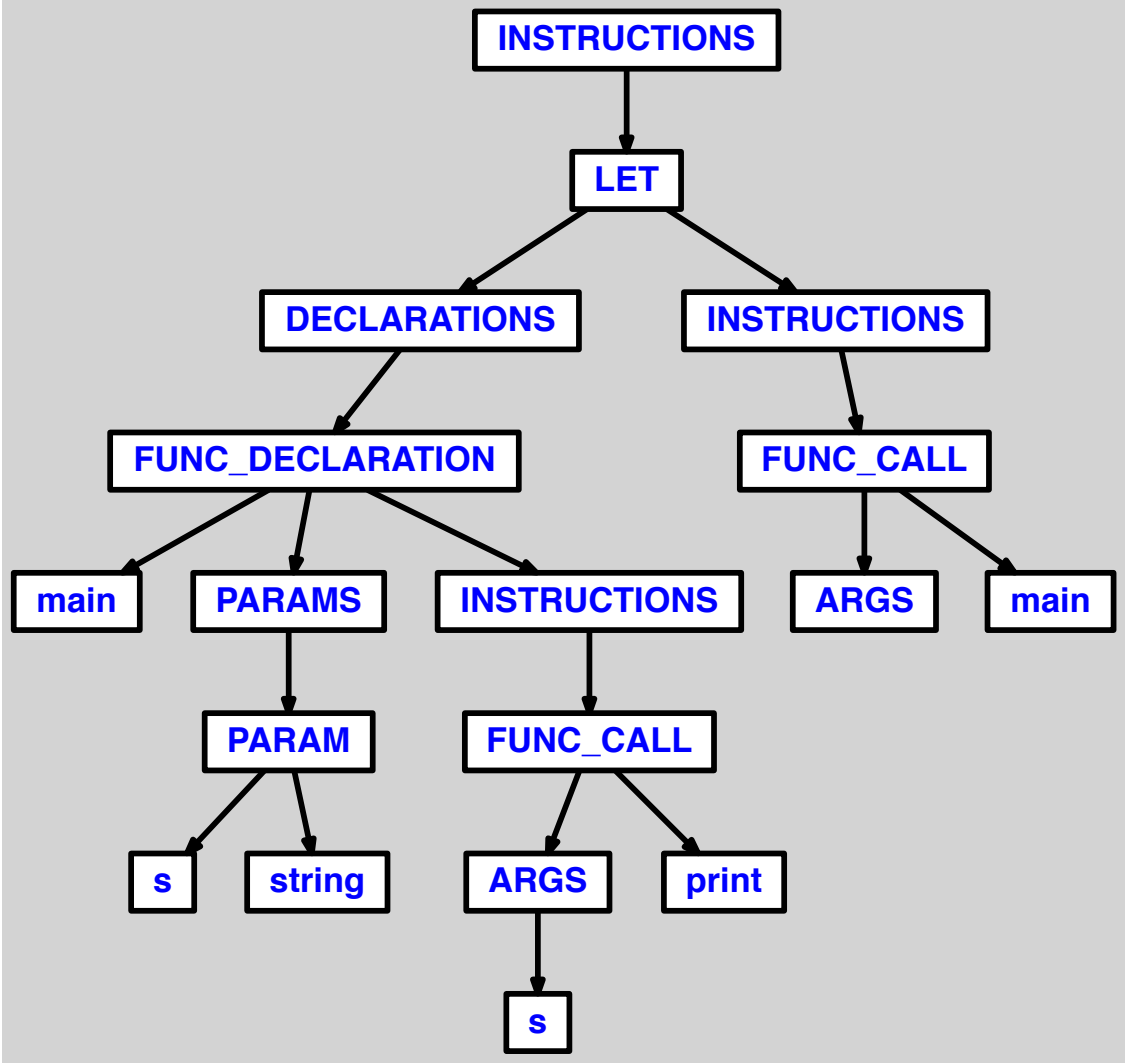
8.2.10 fonction simple, avec instruction simple et instruction retournant une chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
INSTRUCTIONS
FUNC_CALL
STR
ARGS
print
"test"
STR
"test"

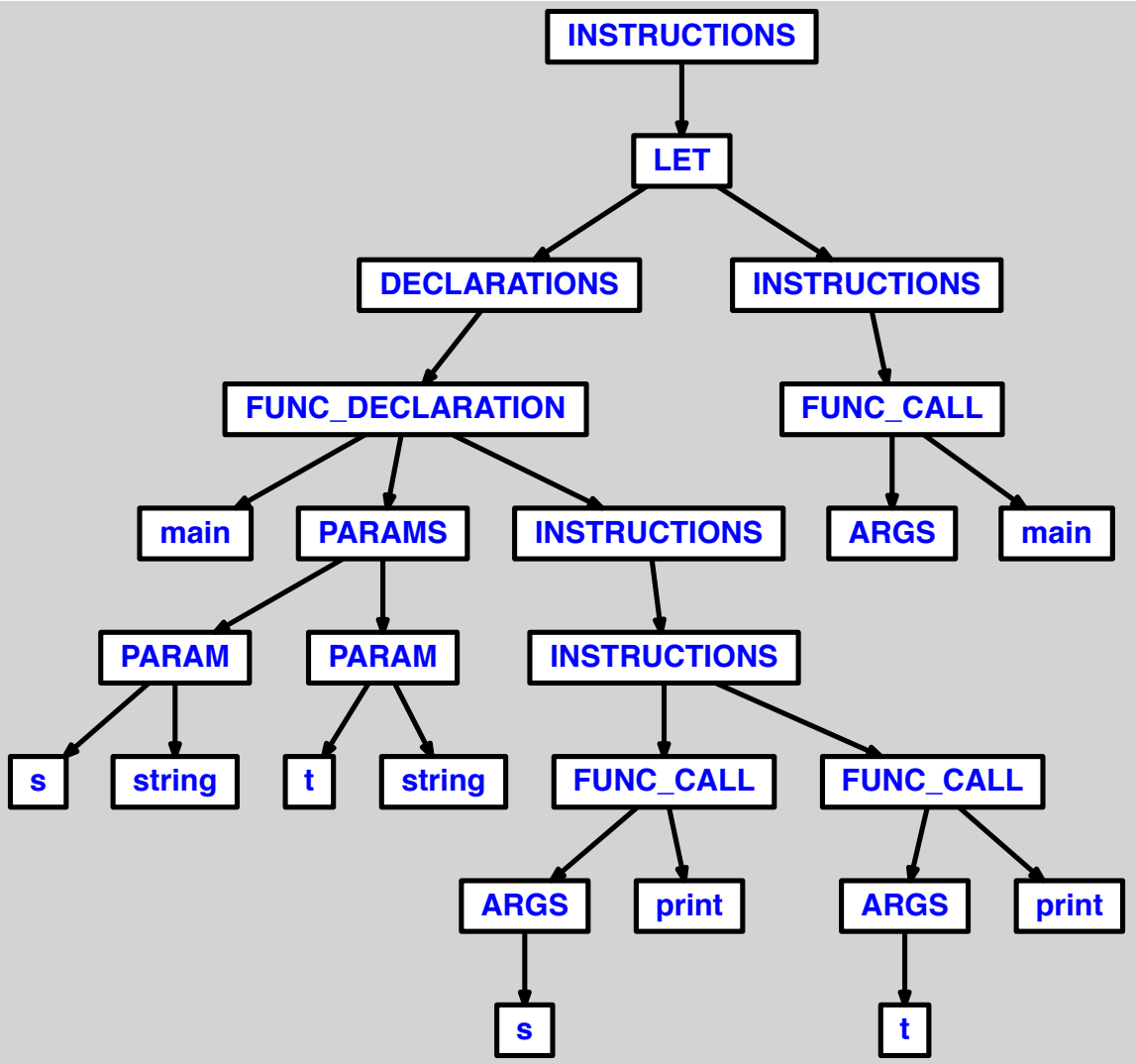
8.2.11 fonction avec 1 parametre de type <string>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
FUNC_CALL
s
string
ARGS
print
s

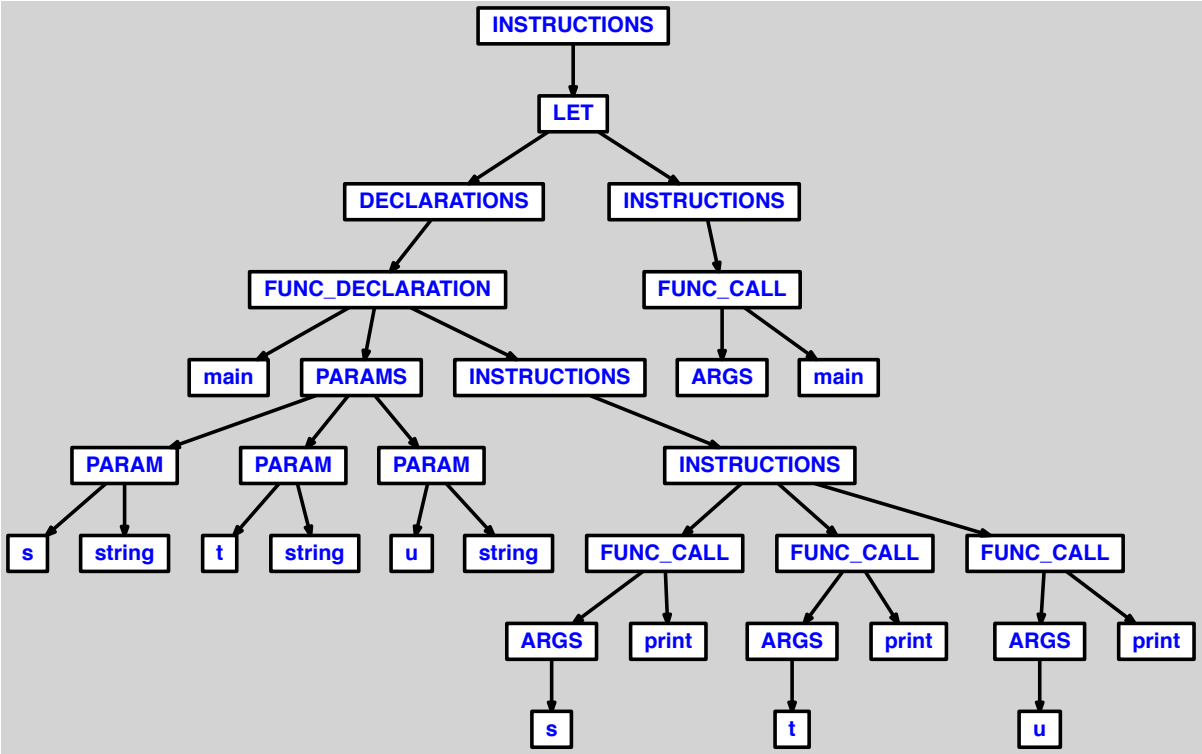
8.2.12 fonction avec 2 parametres de type <string>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
INSTRUCTIONS
s
string
t
string
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
s
t

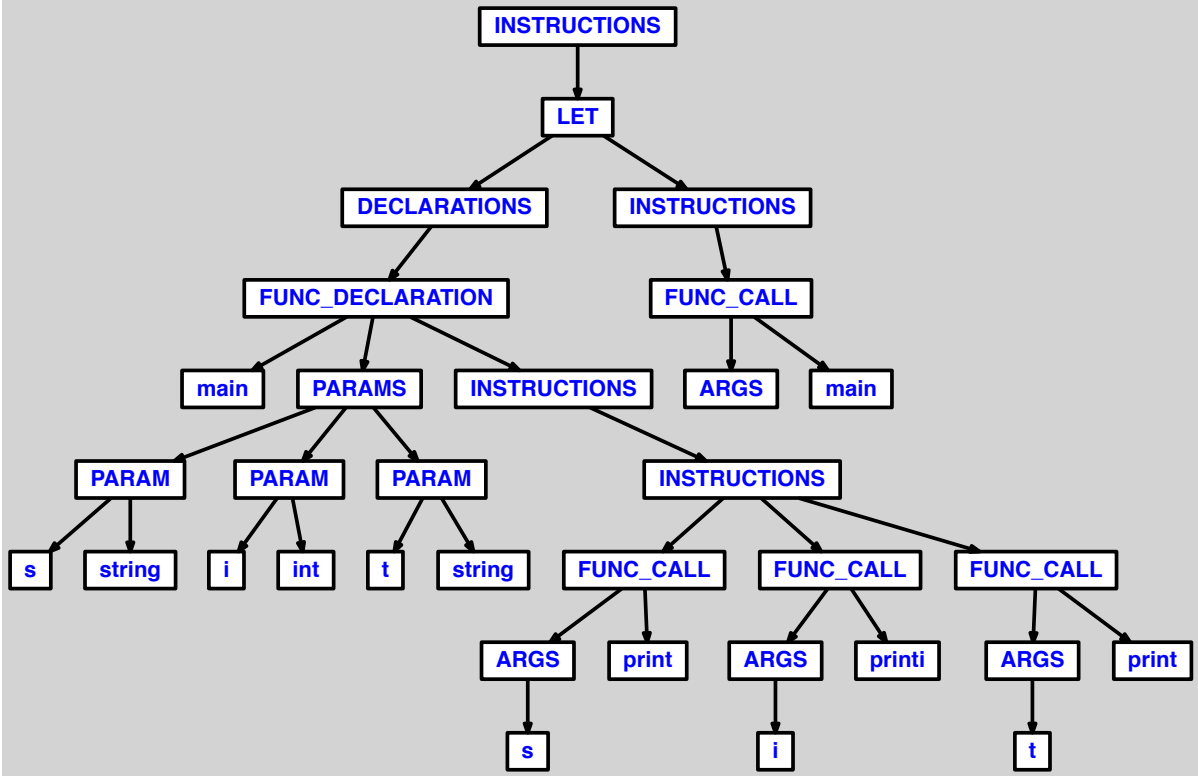
8.2.13 fonction avec 3 parametres de type <string>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
PARAM
INSTRUCTIONS
s
string
t
string
u
string
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
ARGS
print
s
t
u

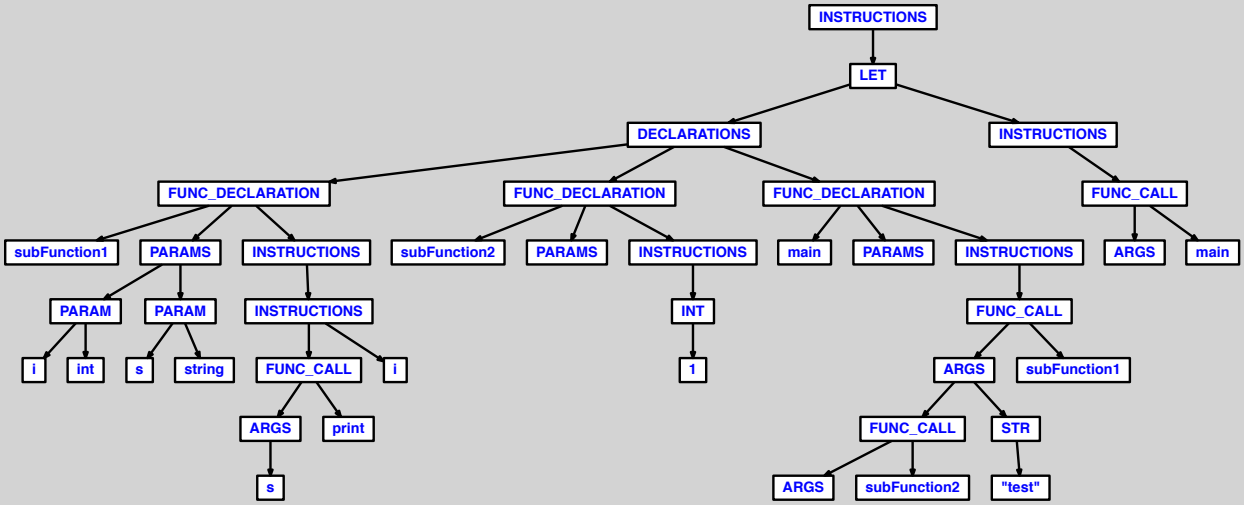
8.2.14 fonction avec 2 parametres de type <string> et 1 parametre entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
PARAM
INSTRUCTIONS
s
string
i
int
t
string
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
printi
ARGS
print
s
i
t

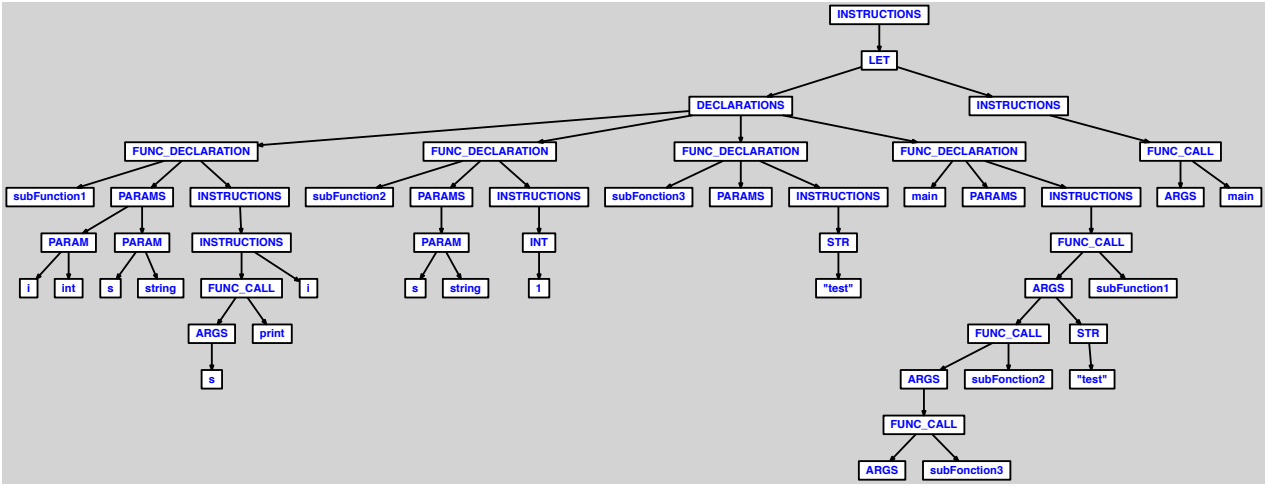
8.2.15 double-imbrication de fonction

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

  function subFunction2() = 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_CALL
subFunction1
PARAMS
INSTRUCTIONS
subFunction2
PARAMS
INSTRUCTIONS
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
INSTRUCTIONS
INT
FUNC_CALL
i
int
s
string
FUNC_CALL
i
1
ARGS
subFunction1
ARGS
print
FUNC_CALL
STR
s
ARGS
subFunction2
"test"

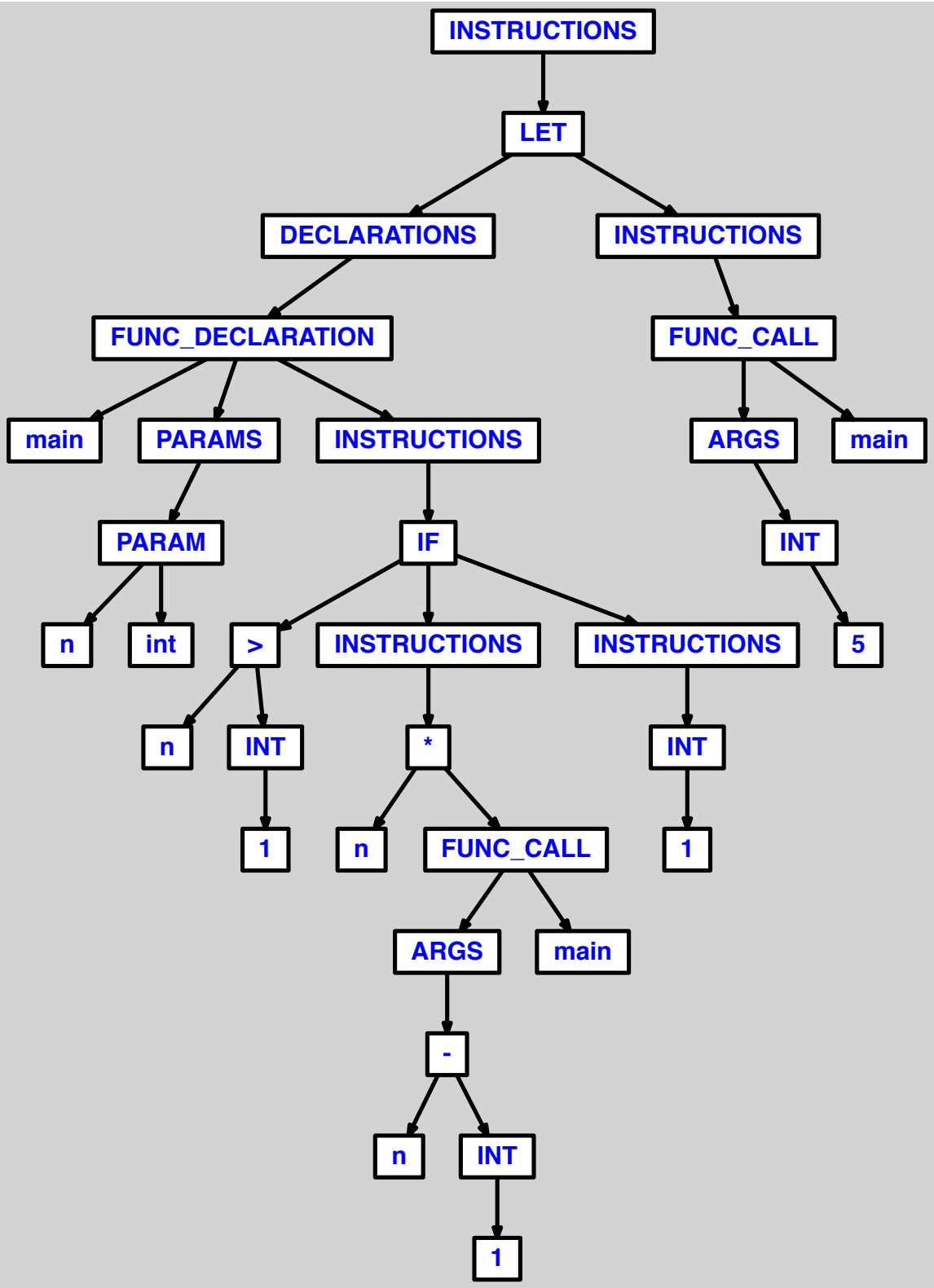
8.2.16 triple-imbrication de fonction

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

  function subFunction2(s: string) = 1

  function subFonction3() = "test"

  function main() = subFunction1(subFonction2(subFonction3()), "test")
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_CALL
subFunction1
PARAMS
INSTRUCTIONS
subFunction2
PARAMS
INSTRUCTIONS
subFonction3
PARAMS
INSTRUCTIONS
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
PARAM
INSTRUCTIONS
PARAM
INT
STR
FUNC_CALL
i
int
s
string
FUNC_CALL
i
s
string
1
"test"
ARGS
subFunction1
ARGS
print
FUNC_CALL
STR
s
ARGS
subFonction2
"test"
FUNC_CALL
ARGS
subFonction3

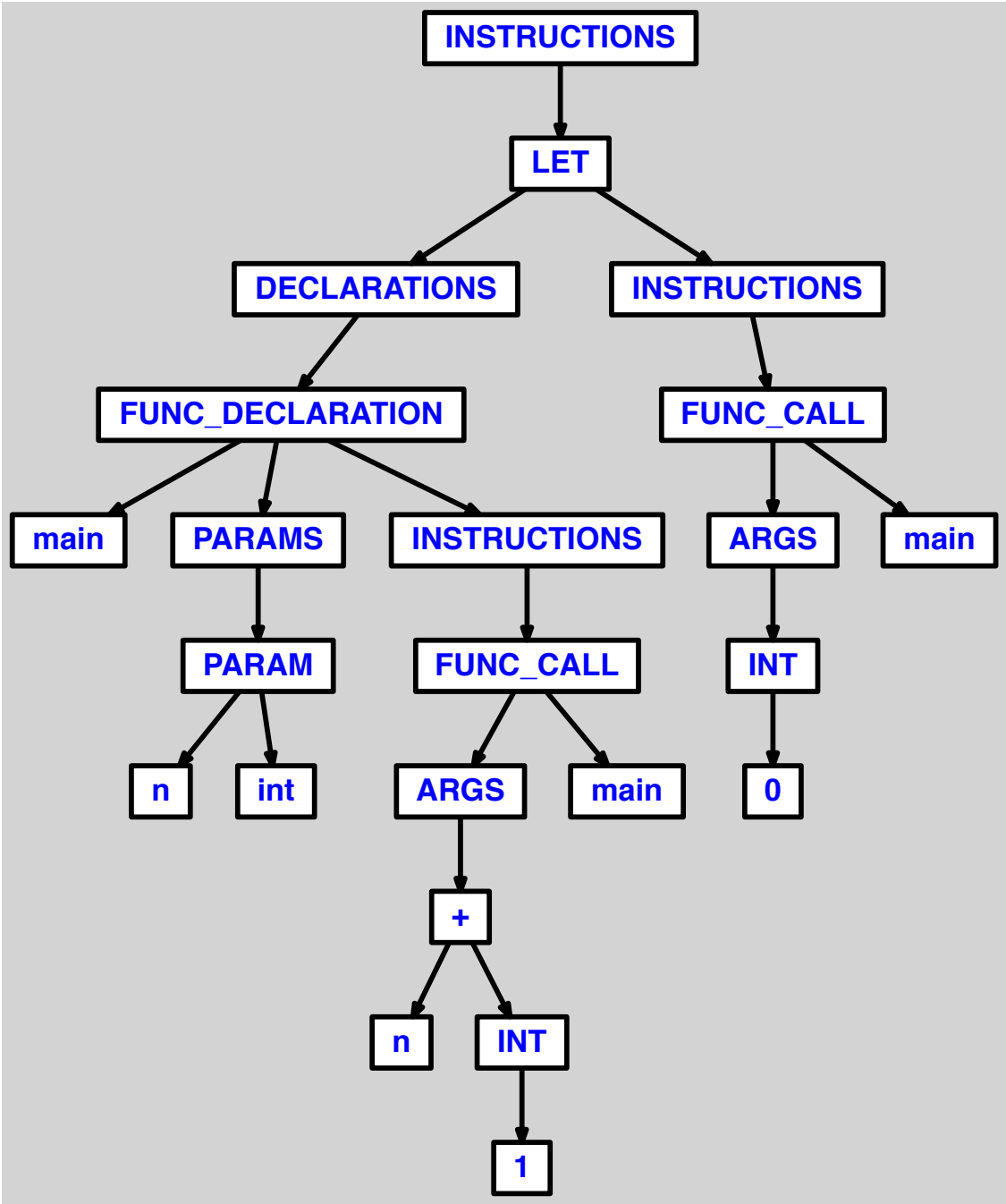
8.2.17 recursion finie

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
IF
INT
n
int
>
INSTRUCTIONS
INSTRUCTIONS
5
n
INT
*
INT
1
n
FUNC_CALL
1
ARGS
main
-
n
INT
1

8.2.18 recursion infinie

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

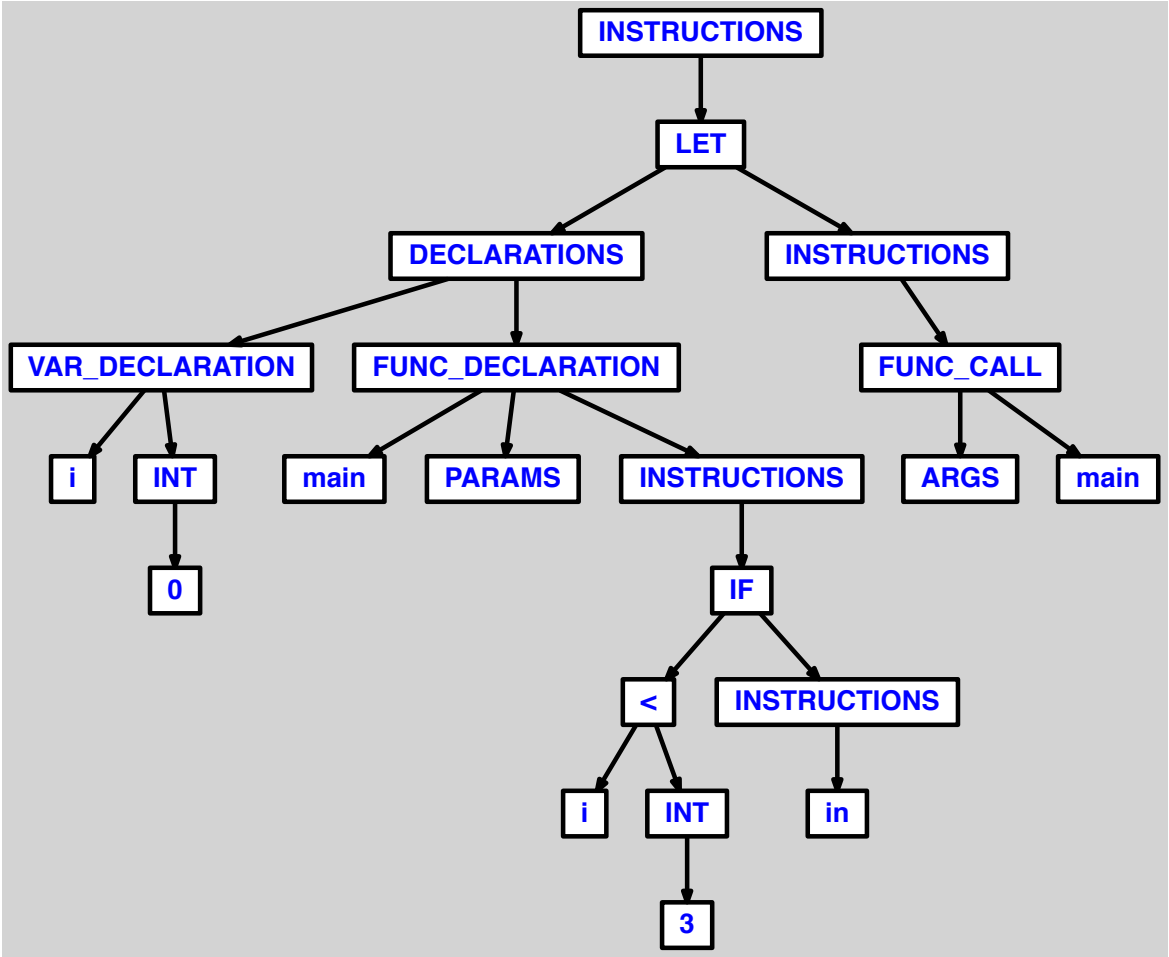

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
PARAM
FUNC_CALL
INT
n
int
ARGS
main
0
+
n
INT
1

9 if

9.1 KO

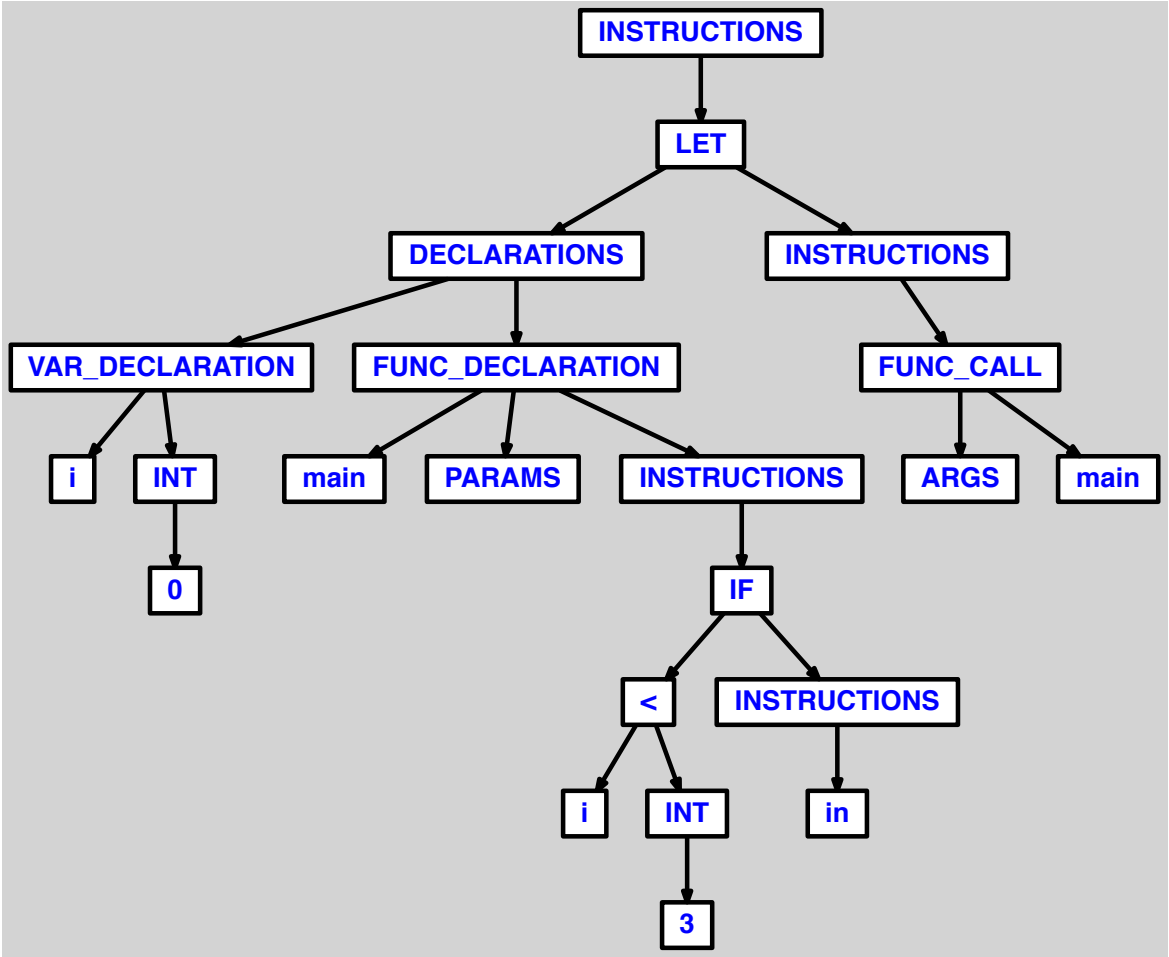
9.1.1 <if then> sans instruction

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
<
INSTRUCTIONS
i
INT
in
3

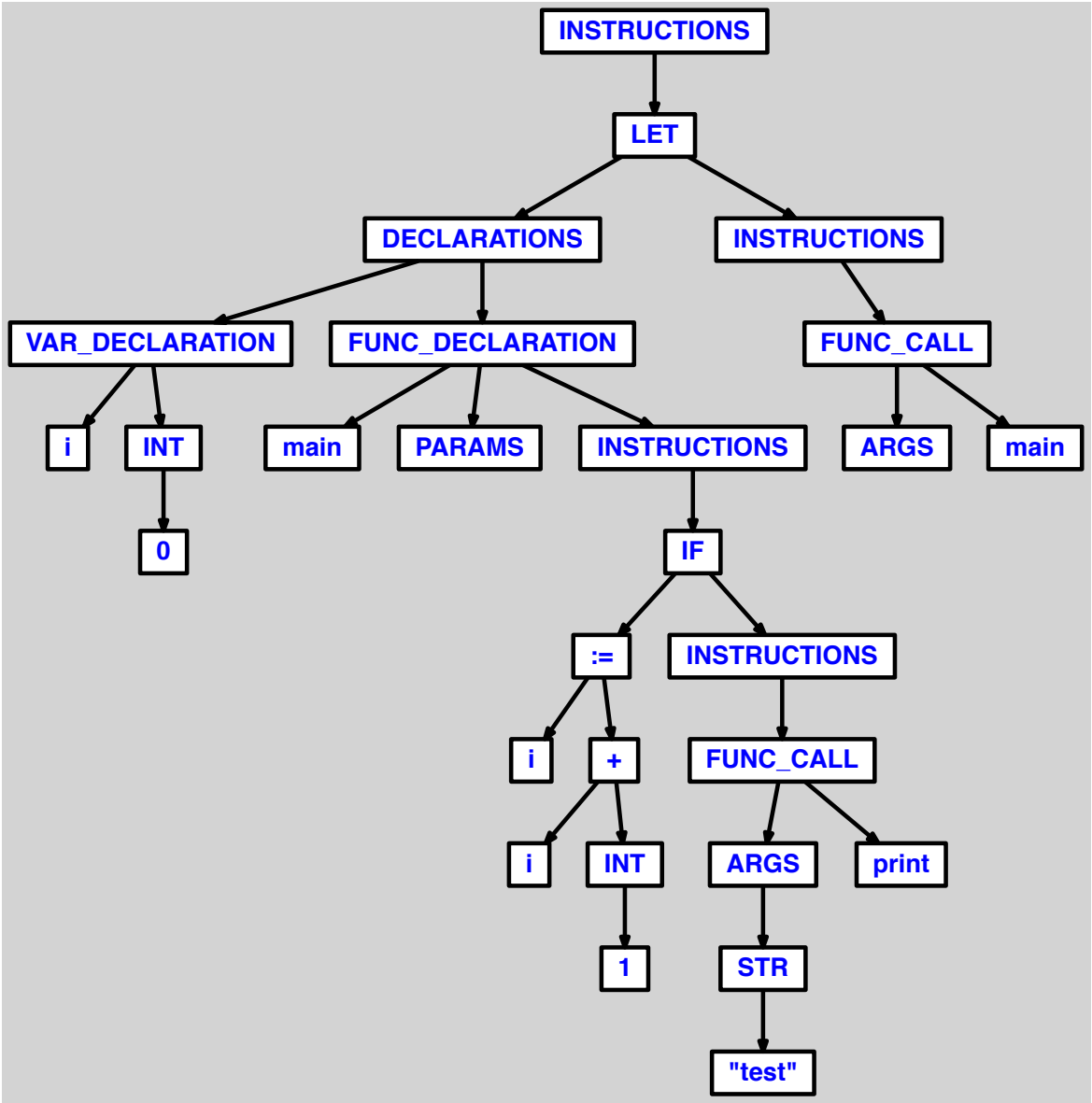
9.1.2 <if then> avec ligne vide

```
let
  var i := 0

  function main() =
    if i < 3 then

in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
<
INSTRUCTIONS
i
INT
in
3

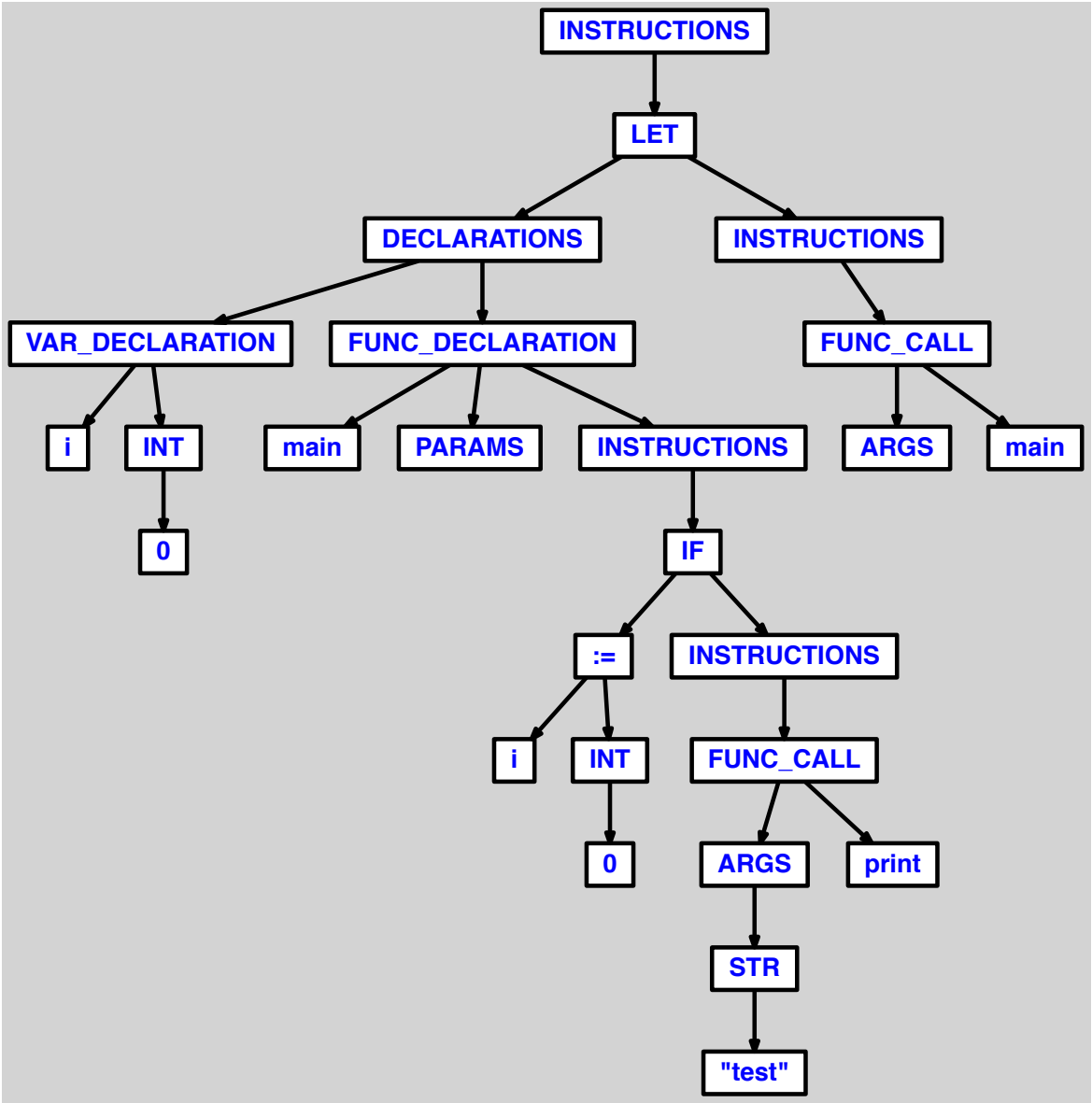
9.1.3 incrementation dans la condition <if then>

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
:=
INSTRUCTIONS
i
+
FUNC_CALL
i
INT
ARGS
print
1
STR
"test"

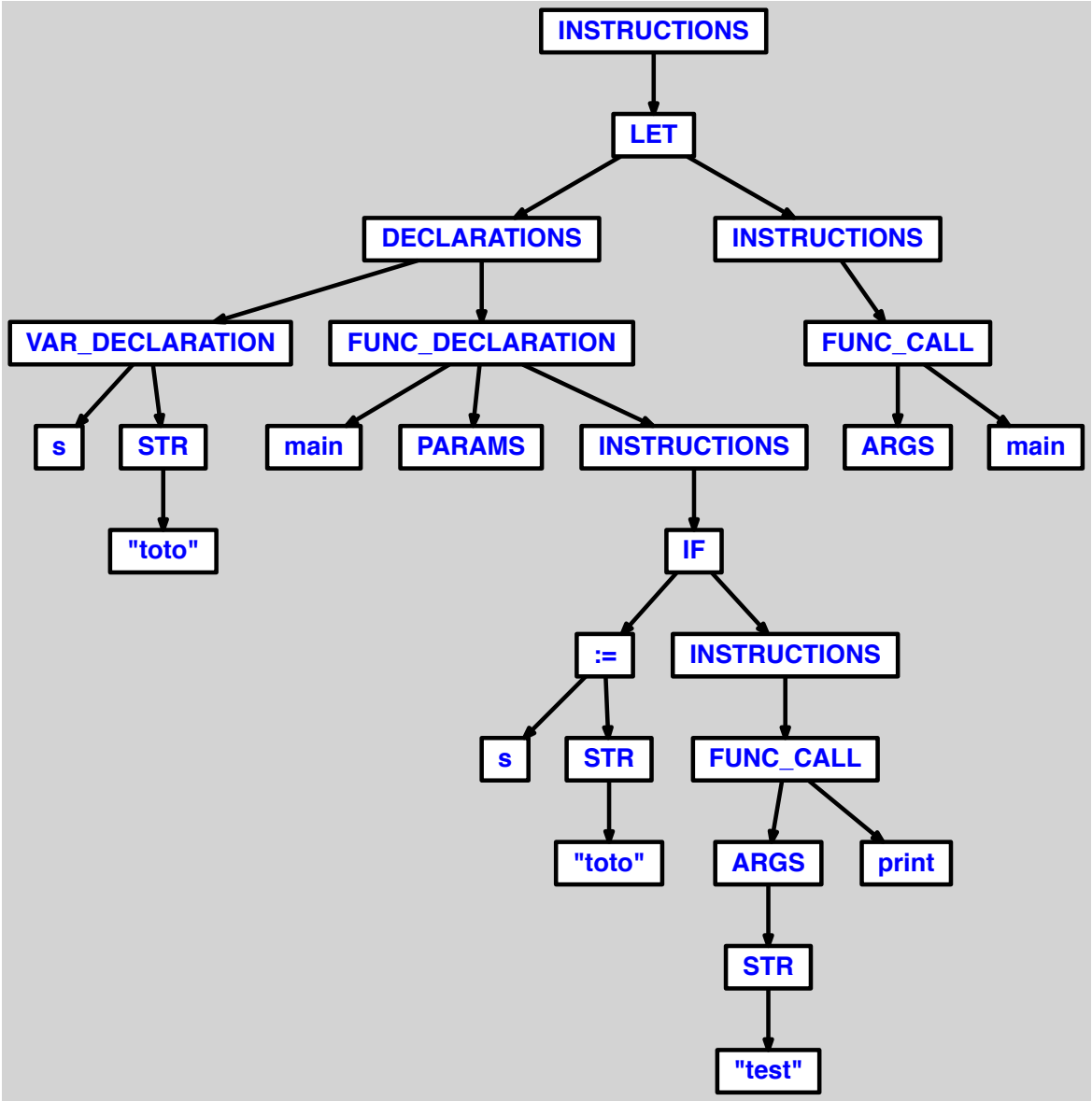
9.1.4 affectation d'entier dans la condition <if then>

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
:=
INSTRUCTIONS
i
INT
FUNC_CALL
0
ARGS
print
STR
"test"

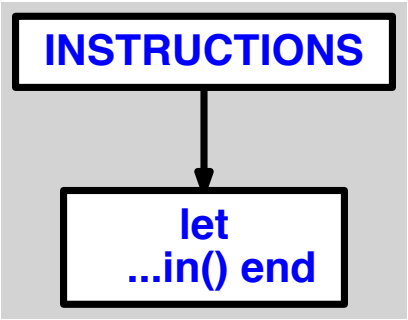
9.1.5 affectation de chaine dans la condition <if then>

```
let
  var s := "toto"

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
s
STR
main
PARAMS
INSTRUCTIONS
ARGS
main
"toto"
IF
:=
INSTRUCTIONS
s
STR
FUNC_CALL
"toto"
ARGS
print
STR
"test"

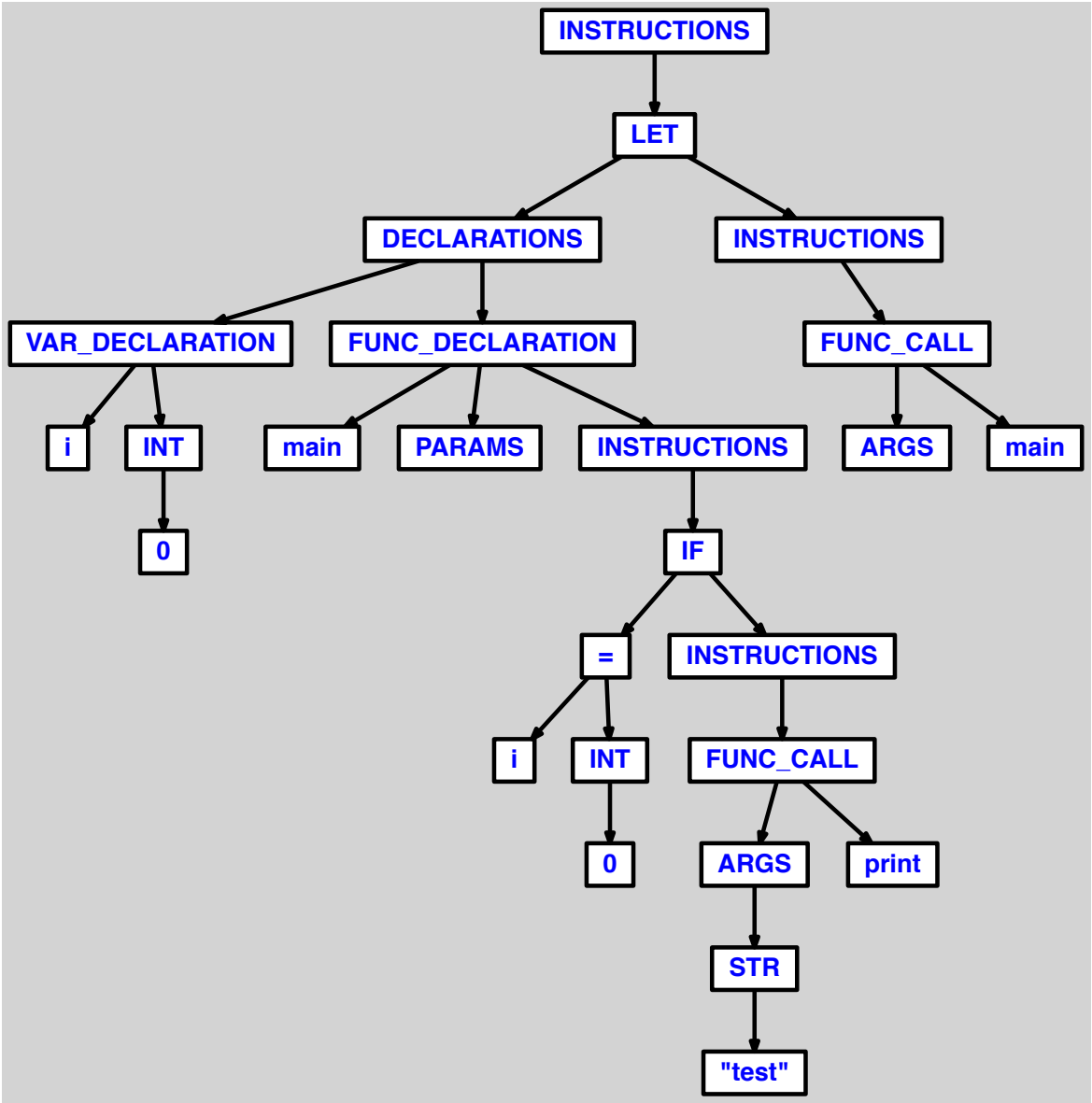
9.1.6 <if then> avec oubli du <if>

```
let
  var i := 0

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


INSTRUCTIONS
let
...in() end

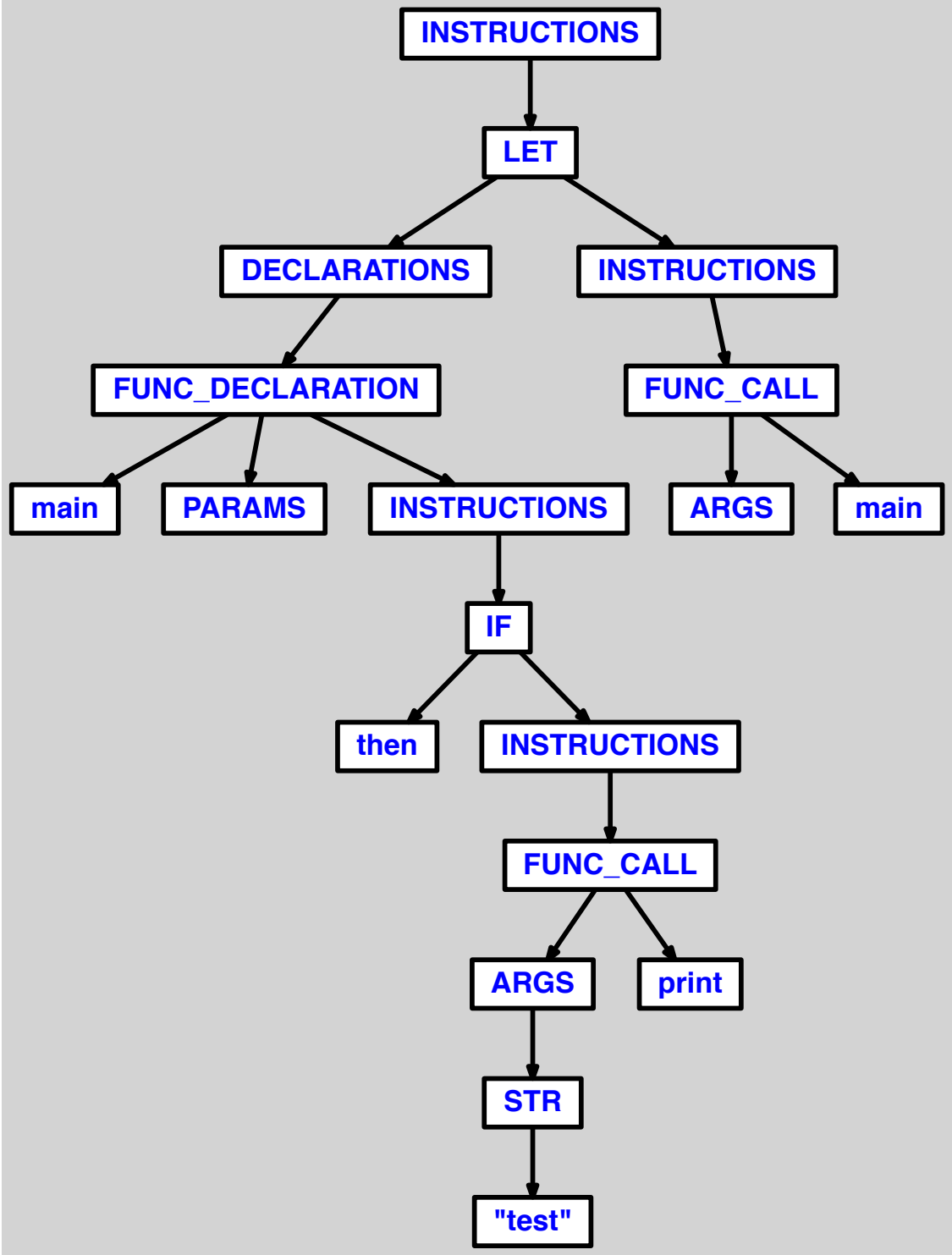
9.1.7 <if then> avec oubli du <then>

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
=
INSTRUCTIONS
i
INT
FUNC_CALL
0
ARGS
print
STR
"test"

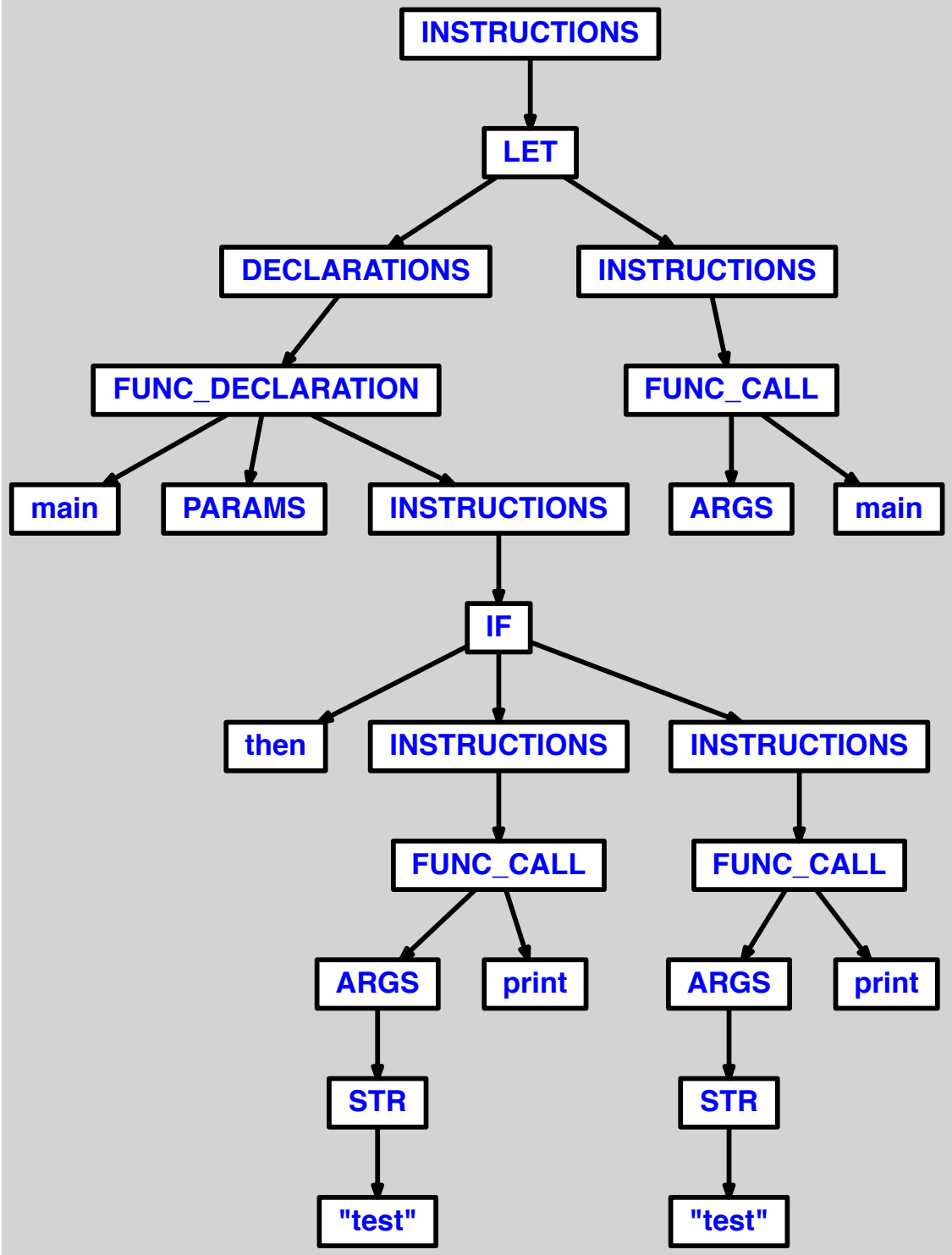
9.1.8 <if then> avec oubli de la condition

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
then
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"test"

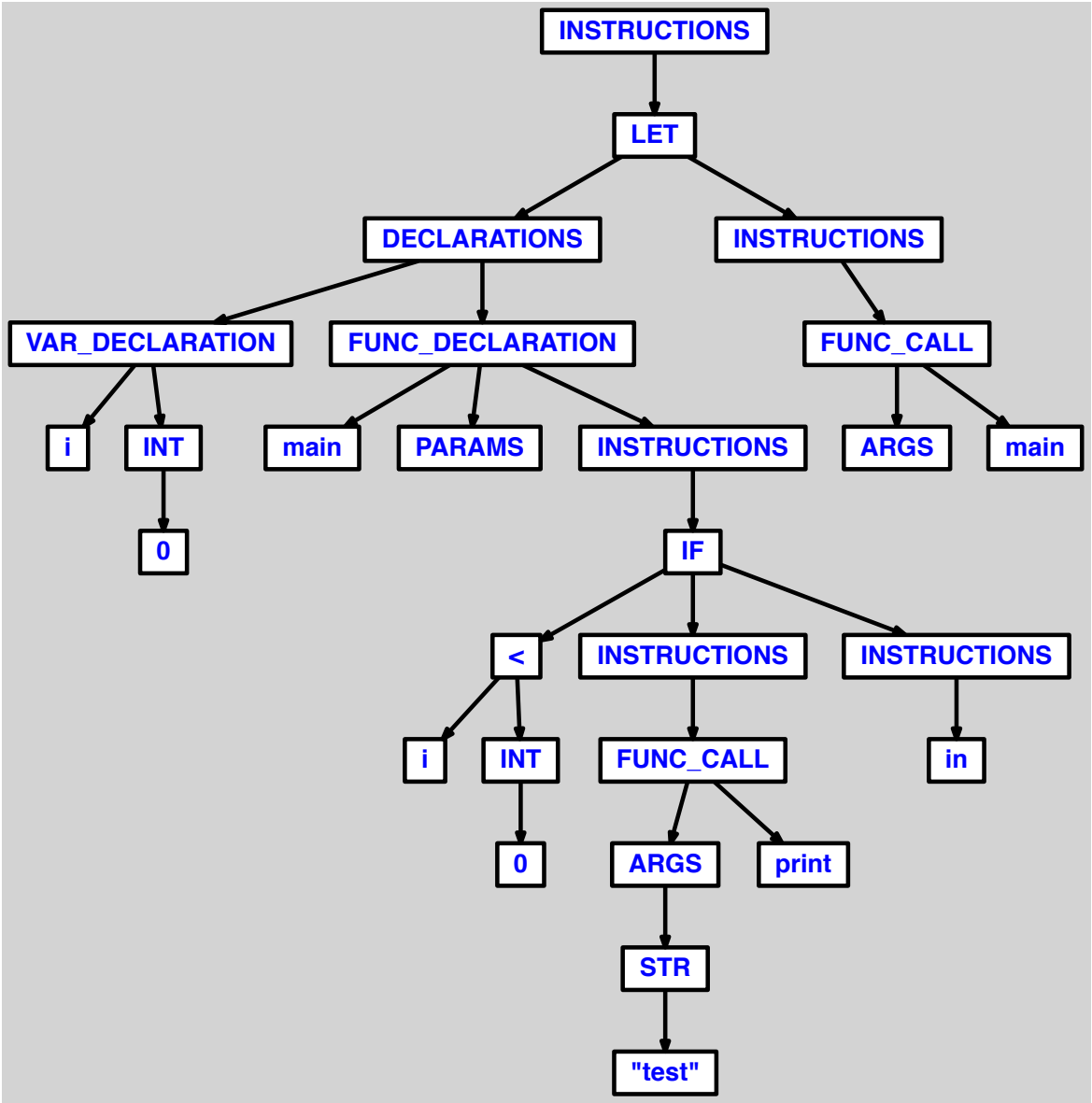
9.1.9 <if then else> avec oubli de la condition

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
then
INSTRUCTIONS
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
ARGS
print
ARGS
print
STR
STR
"test"
"test"

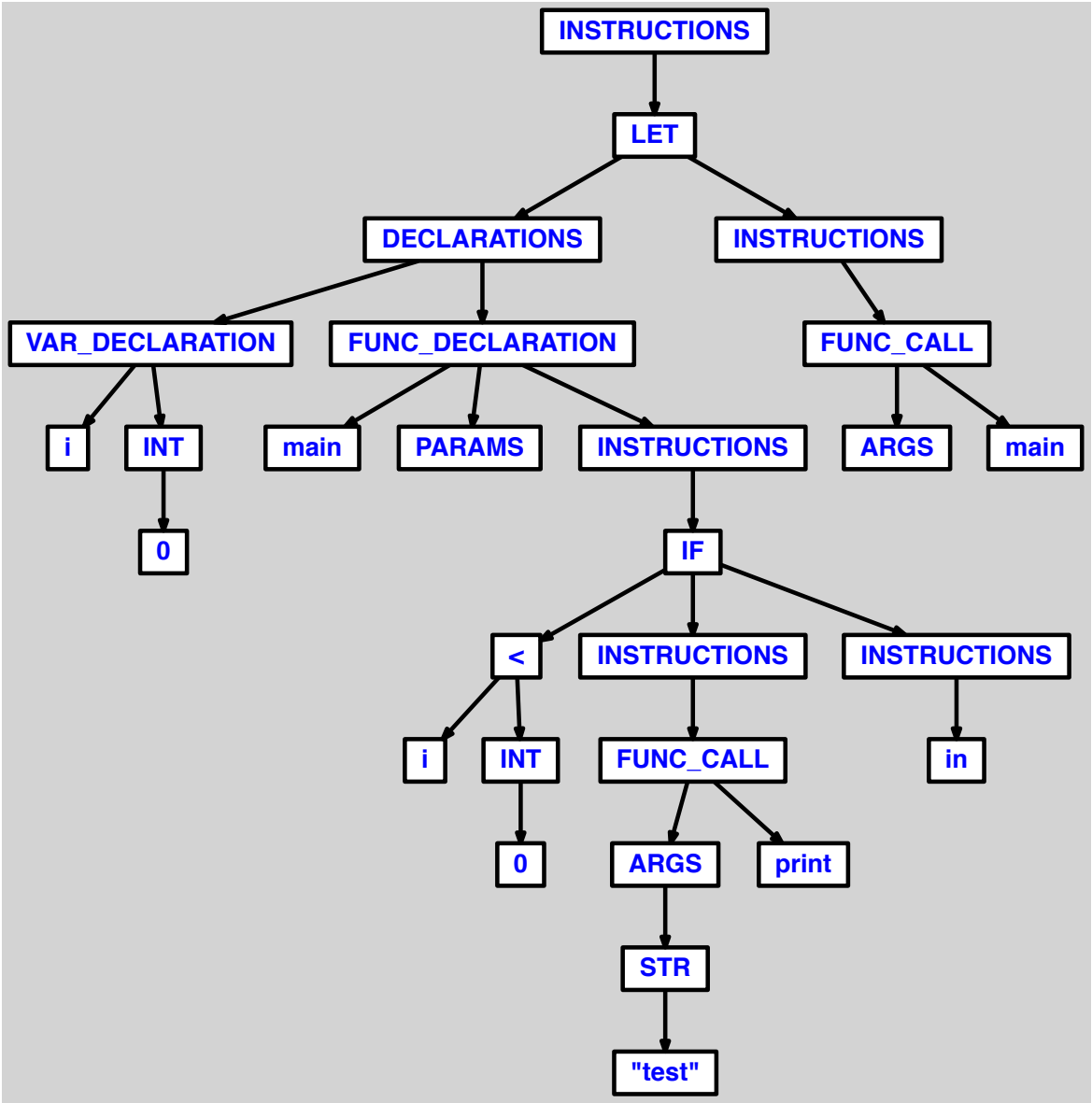
9.1.10 <if then else> sans instruction dans <else>

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
<
INSTRUCTIONS
INSTRUCTIONS
i
INT
FUNC_CALL
in
0
ARGS
print
STR
"test"

9.1.11 <if then else> avec ligne vide dans <else>

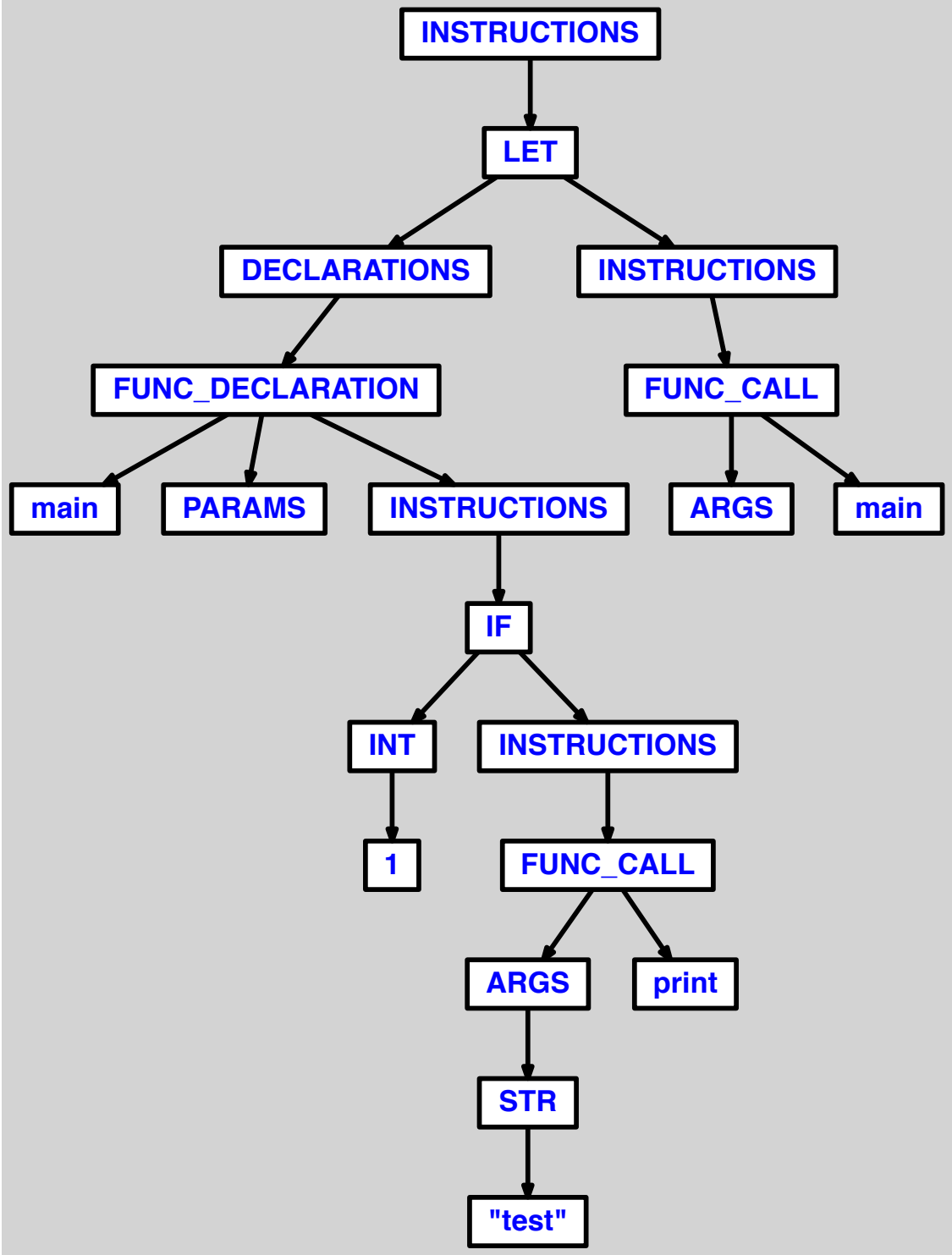
```
let
  var i := 0

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

in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
<
INSTRUCTIONS
INSTRUCTIONS
i
INT
FUNC_CALL
in
0
ARGS
print
STR
"test"

9.2 OK

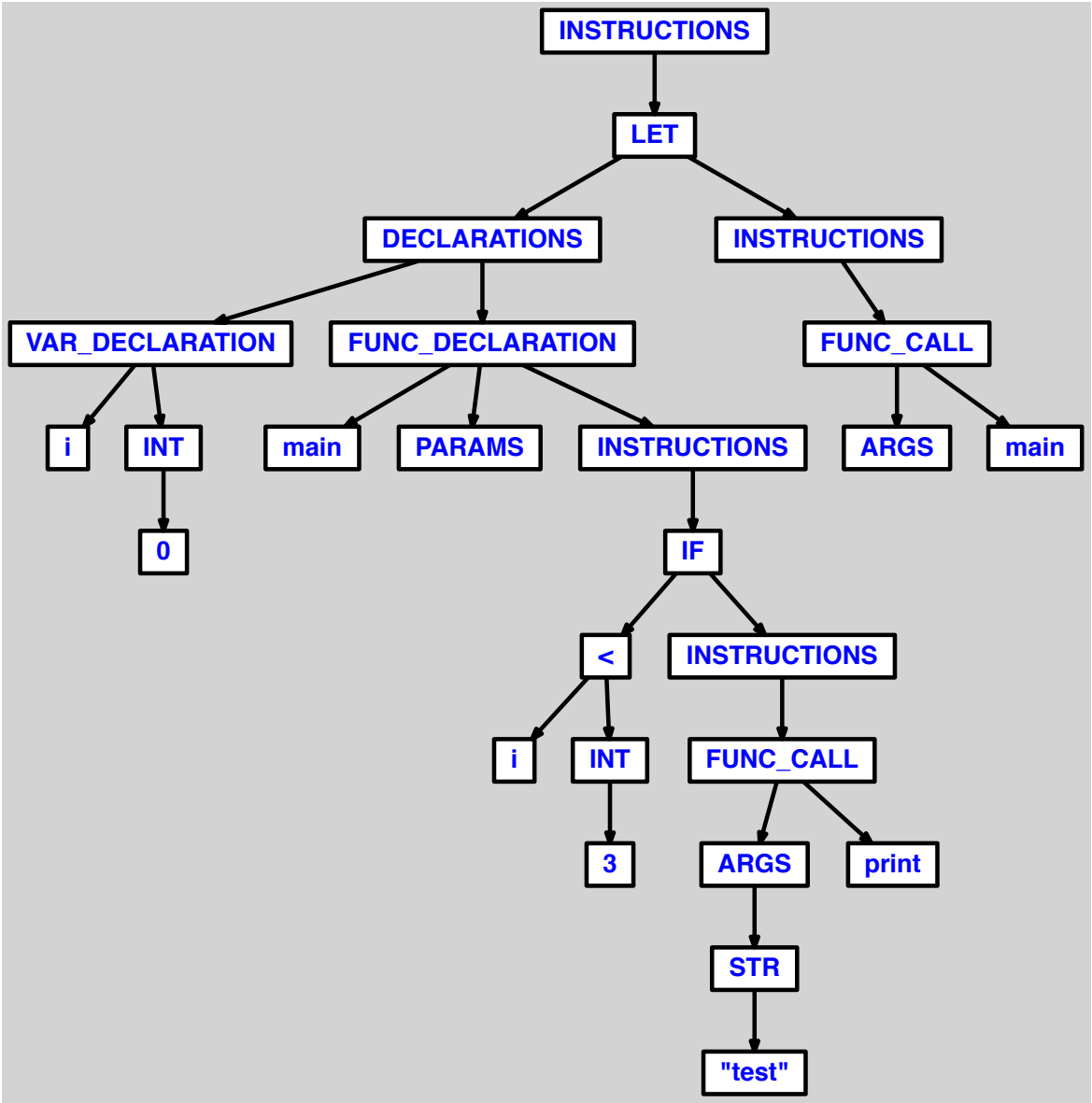
9.2.1 <if then> avec condition entiere

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
INT
INSTRUCTIONS
1
FUNC_CALL
ARGS
print
STR
"test"

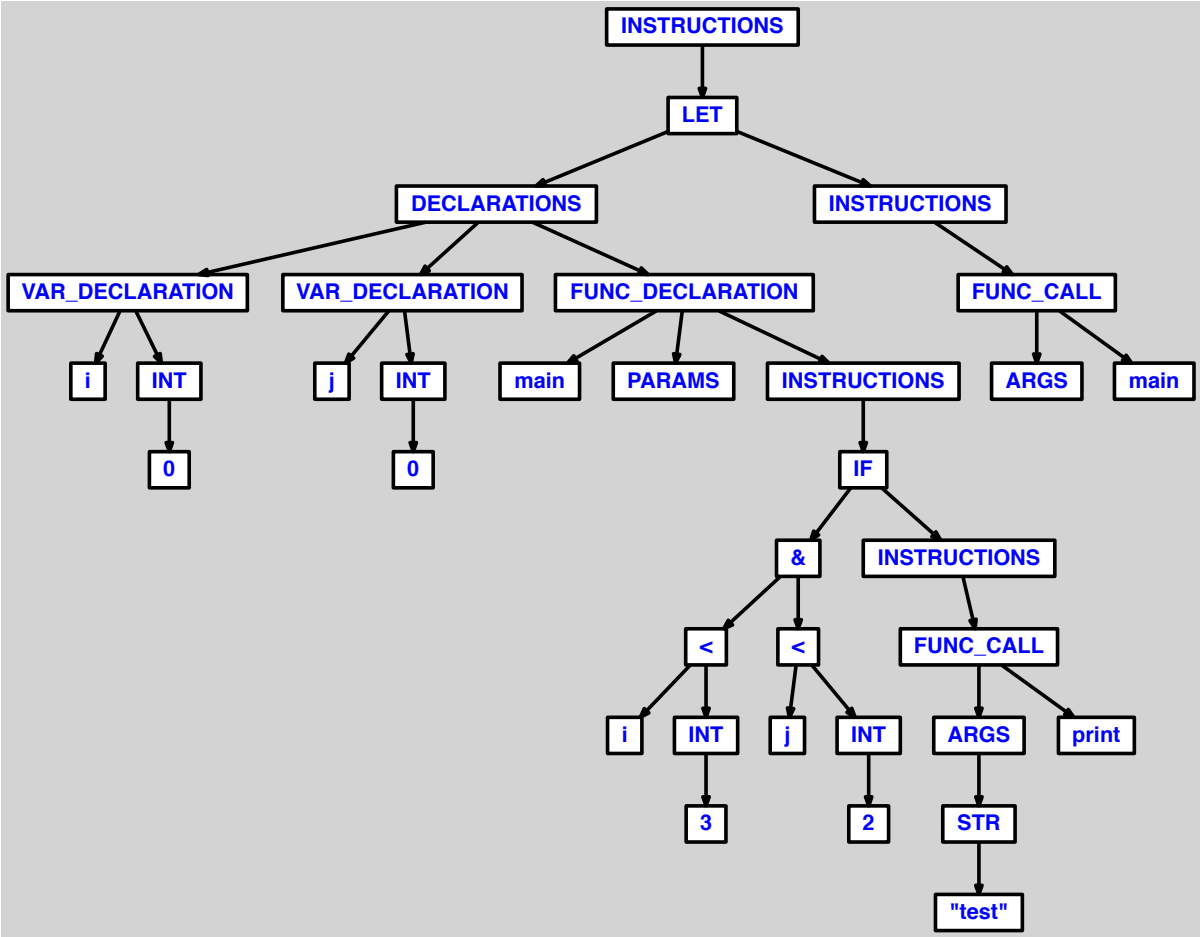
9.2.2 <if then> avec simple condition

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
IF
<
INSTRUCTIONS
i
INT
FUNC_CALL
3
ARGS
print
STR
"test"

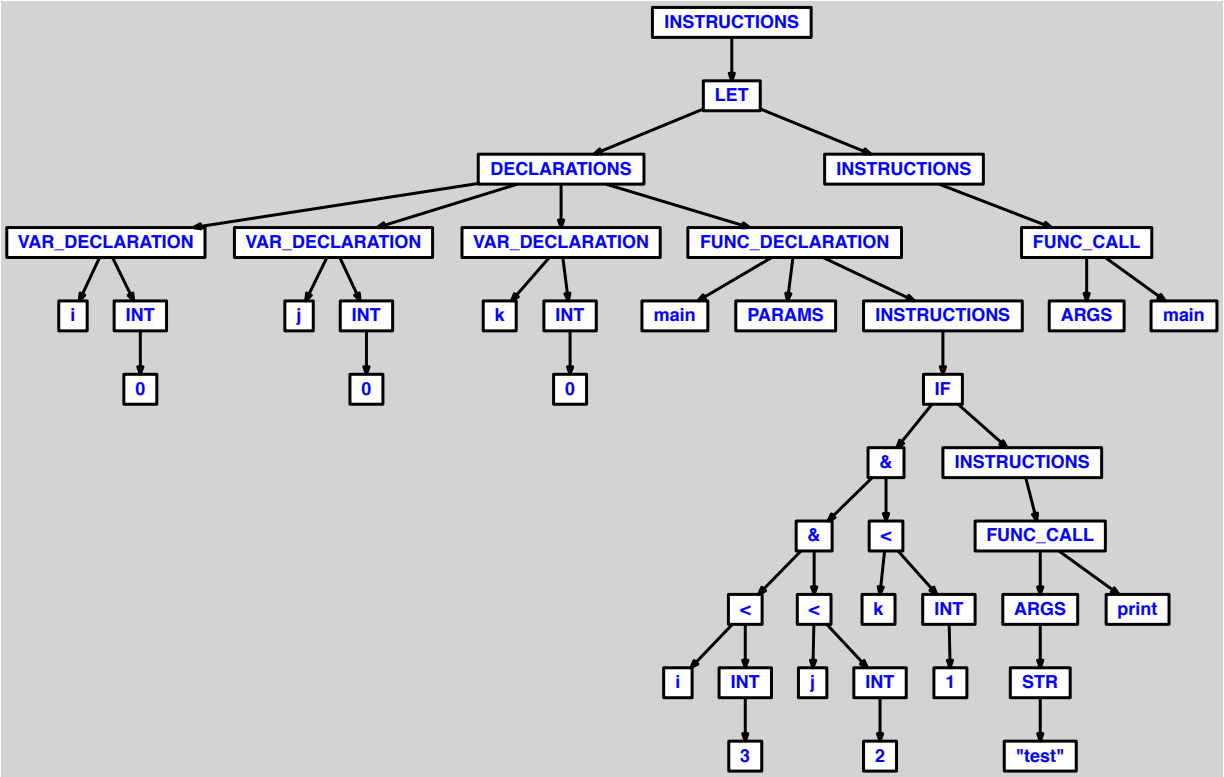
9.2.3 <if then> avec double-condition

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
IF
&
INSTRUCTIONS
<
<
FUNC_CALL
i
INT
j
INT
ARGS
print
3
2
STR
"test"

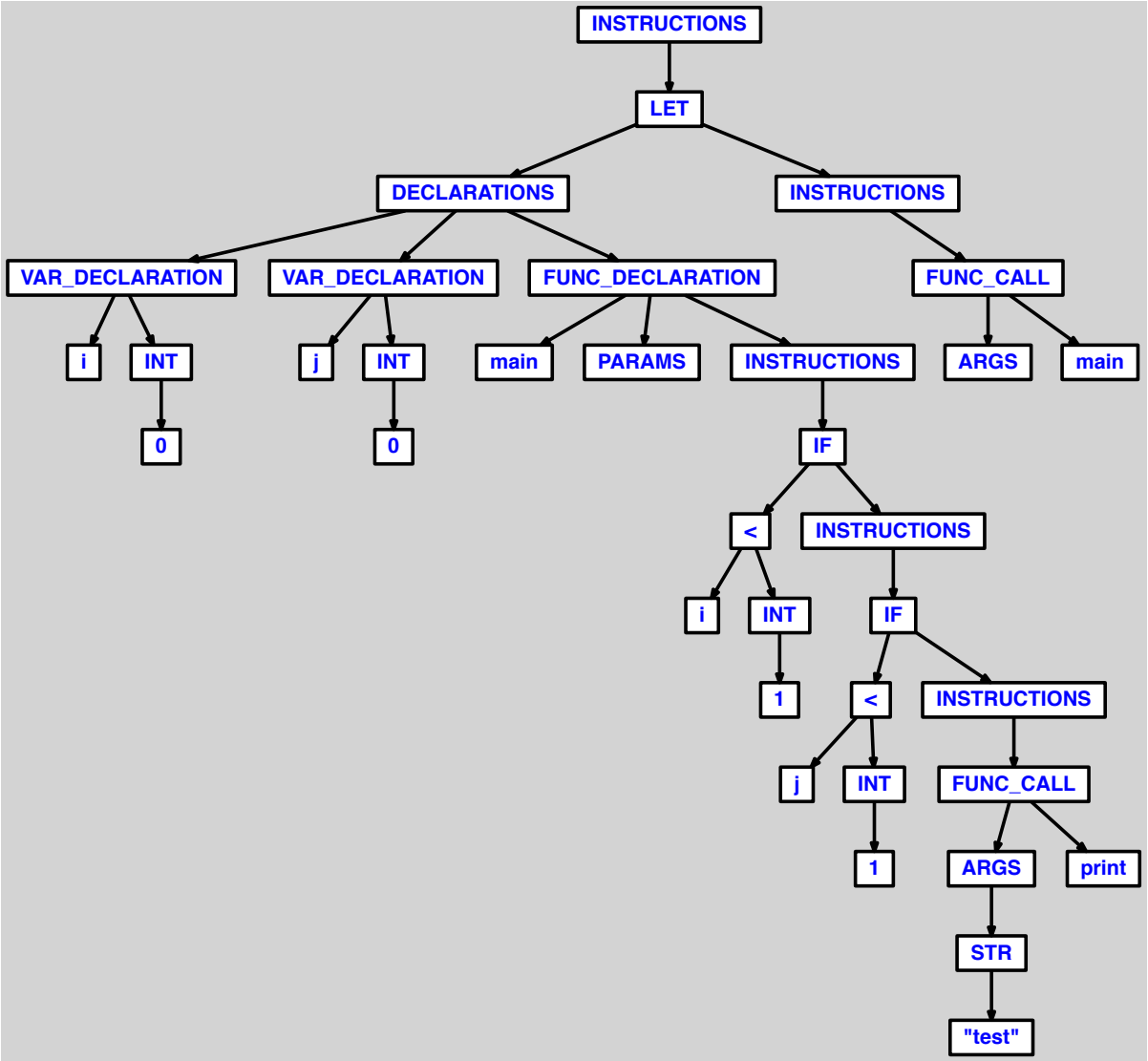
9.2.4 <if then> avec triple-condition

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
0
IF
&
INSTRUCTIONS
&
<
FUNC_CALL
<
<
k
INT
ARGS
print
i
INT
j
INT
1
STR
3
2
"test"

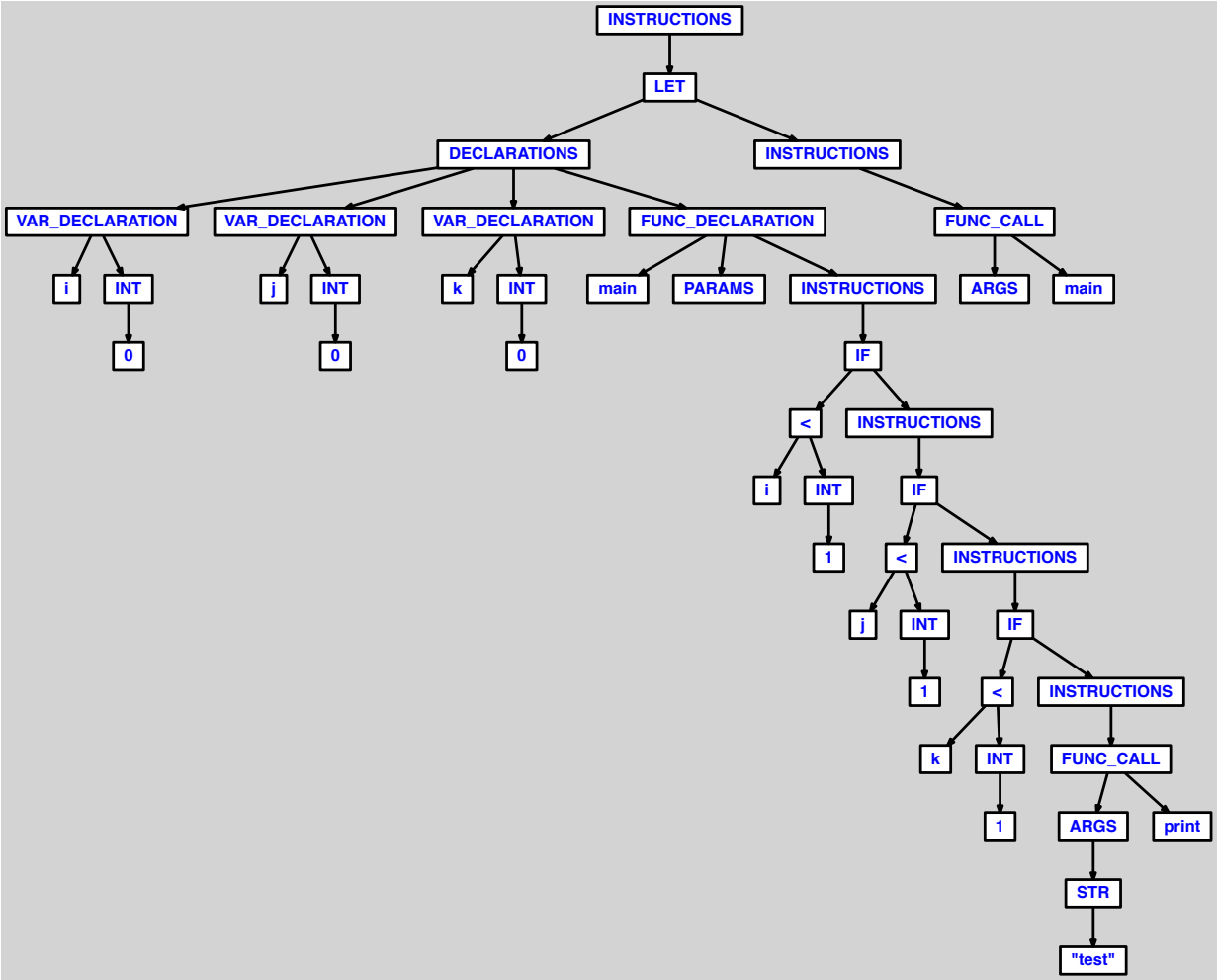
9.2.5 double-imbrication de <if then>

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
IF
<
INSTRUCTIONS
i
INT
IF
1
<
INSTRUCTIONS
j
INT
FUNC_CALL
1
ARGS
print
STR
"test"

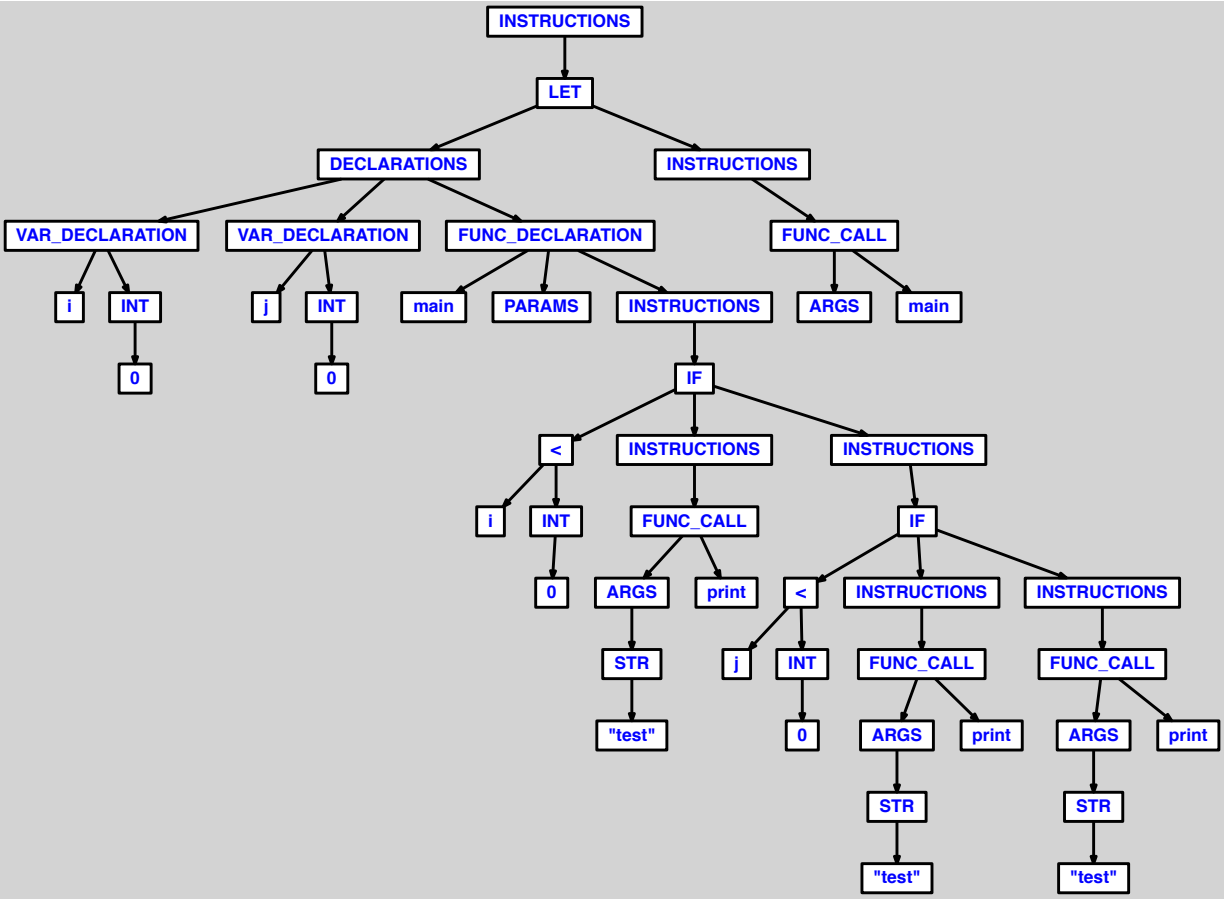
9.2.6 triple-imbrication de <if then>

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
0
IF
<
INSTRUCTIONS
i
INT
IF
1
<
INSTRUCTIONS
j
INT
IF
1
<
INSTRUCTIONS
k
INT
FUNC_CALL
1
ARGS
print
STR
"test"

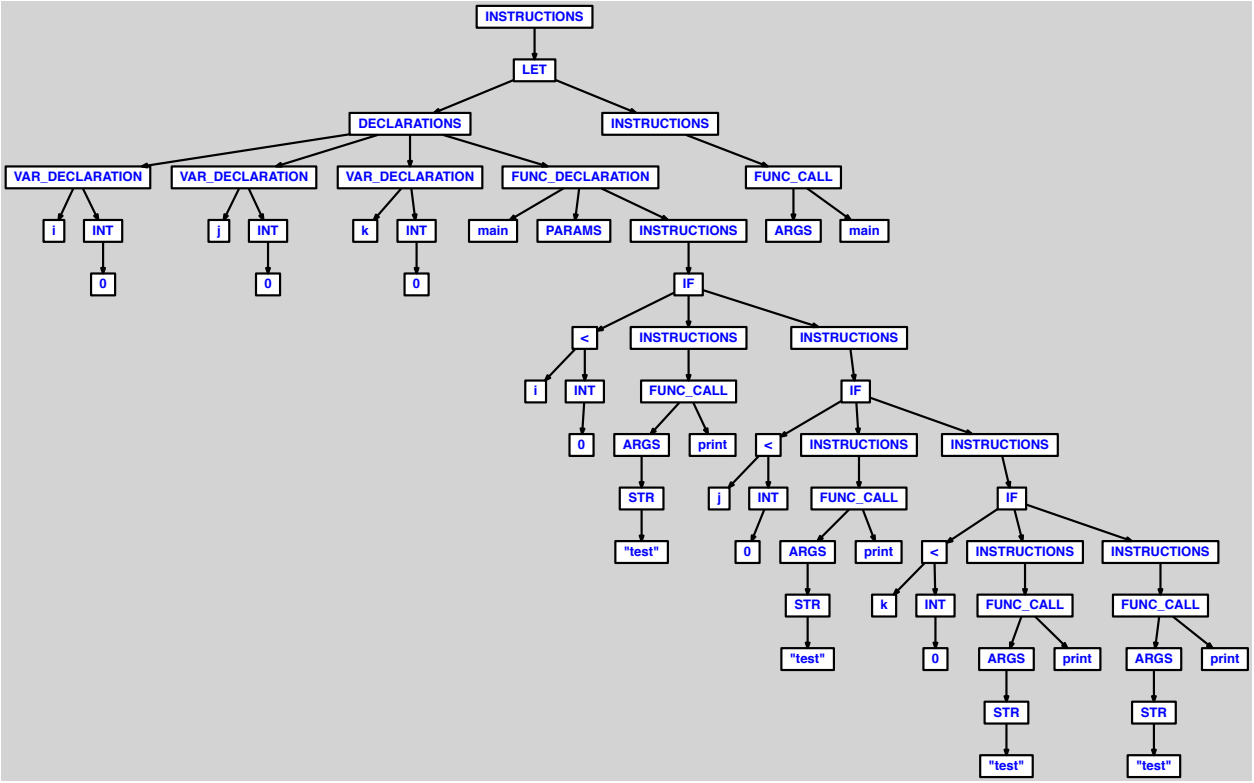
9.2.7 double-imbrication de <if then else>

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
IF
<
INSTRUCTIONS
INSTRUCTIONS
i
INT
FUNC_CALL
IF
0
ARGS
print
<
INSTRUCTIONS
INSTRUCTIONS
STR
j
INT
FUNC_CALL
FUNC_CALL
"test"
0
ARGS
print
ARGS
print
STR
STR
"test"
"test"

9.2.8 triple-imbrication de <if then else>

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

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

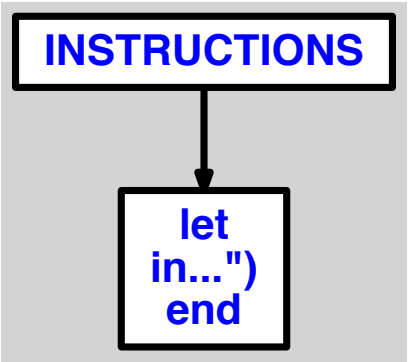

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
0
IF
<
INSTRUCTIONS
INSTRUCTIONS
i
INT
FUNC_CALL
IF
0
ARGS
print
<
INSTRUCTIONS
INSTRUCTIONS
STR
j
INT
FUNC_CALL
IF
"test"
0
ARGS
print
<
INSTRUCTIONS
INSTRUCTIONS
STR
k
INT
FUNC_CALL
FUNC_CALL
"test"
0
ARGS
print
ARGS
print
STR
STR
"test"
"test"

10 let

10.1 KO

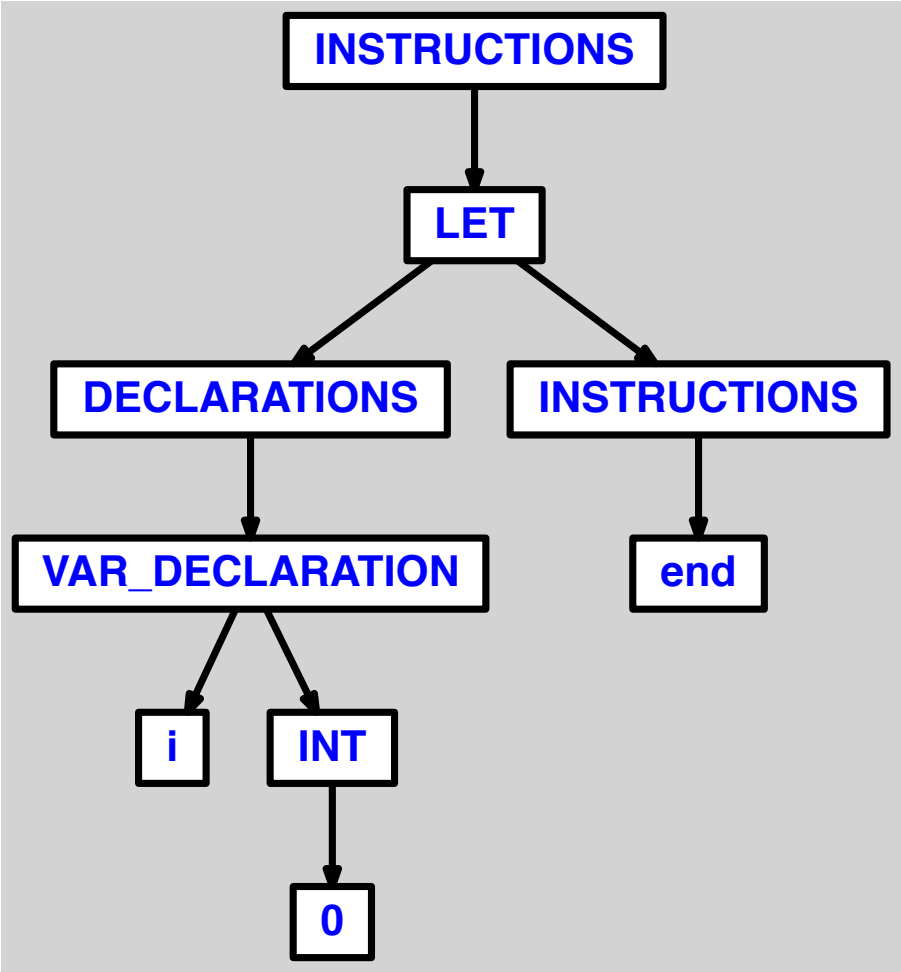
10.1.1 <let> sans declaration

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


INSTRUCTIONS
let
in...")
end

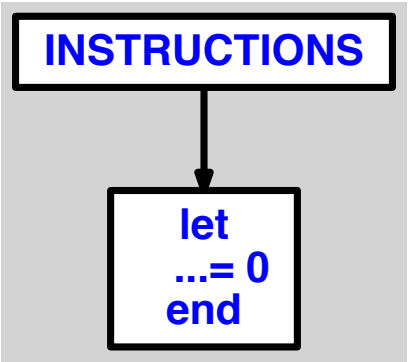
10.1.2 <let> sans instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
end
i
INT
0

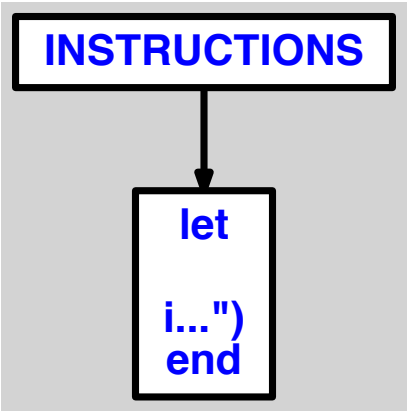
10.1.3 <let> avec inversion de declaration et instruction

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


INSTRUCTIONS
let
...= 0
end

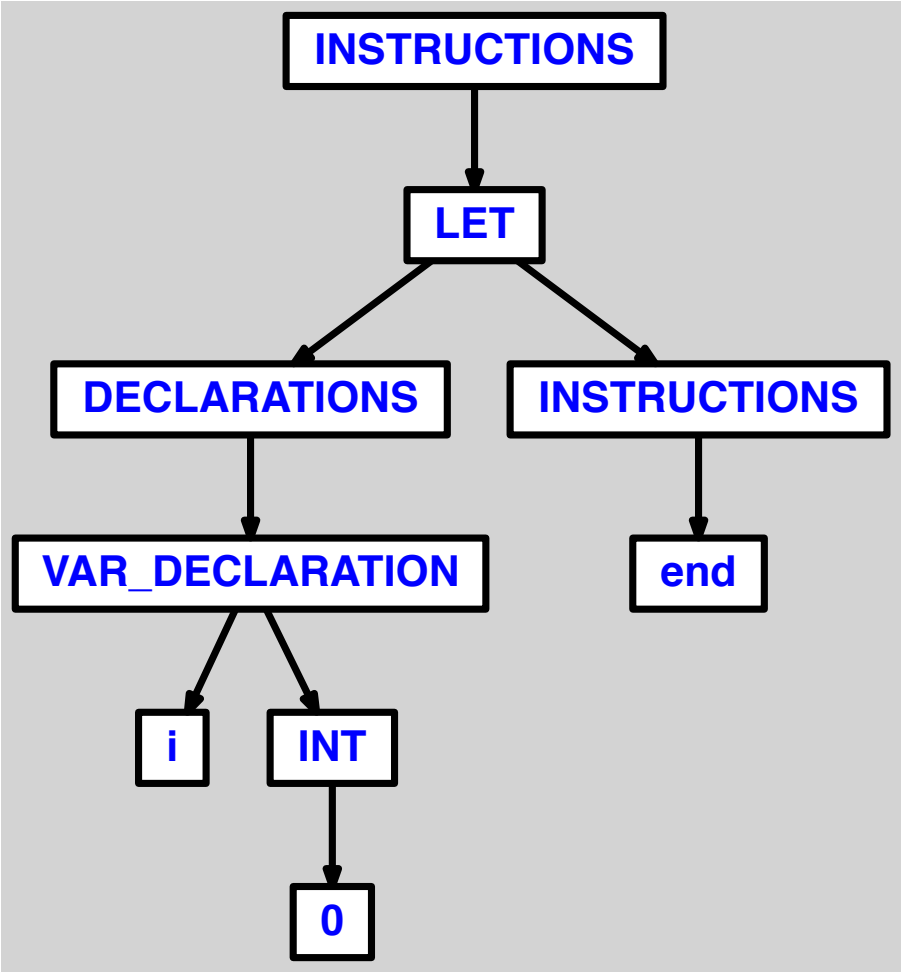
10.1.4 <let> avec declaration vide

```
let

in
  print("test")
end
```


INSTRUCTIONS
let
i...")
end

10.1.5 <let> avec instruction vide

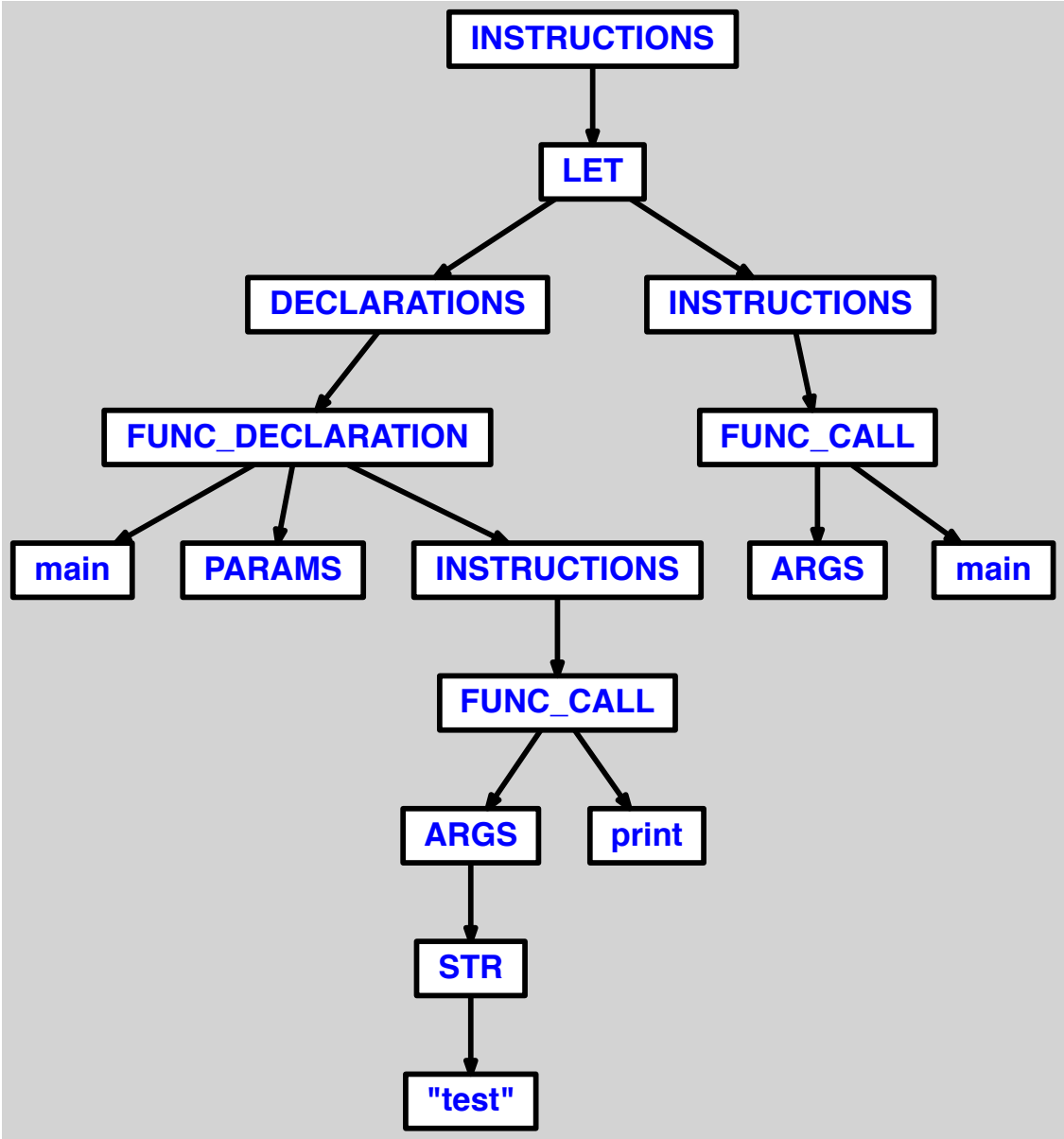
```
let
  var i := 0
in

end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
end
i
INT
0

10.2 OK

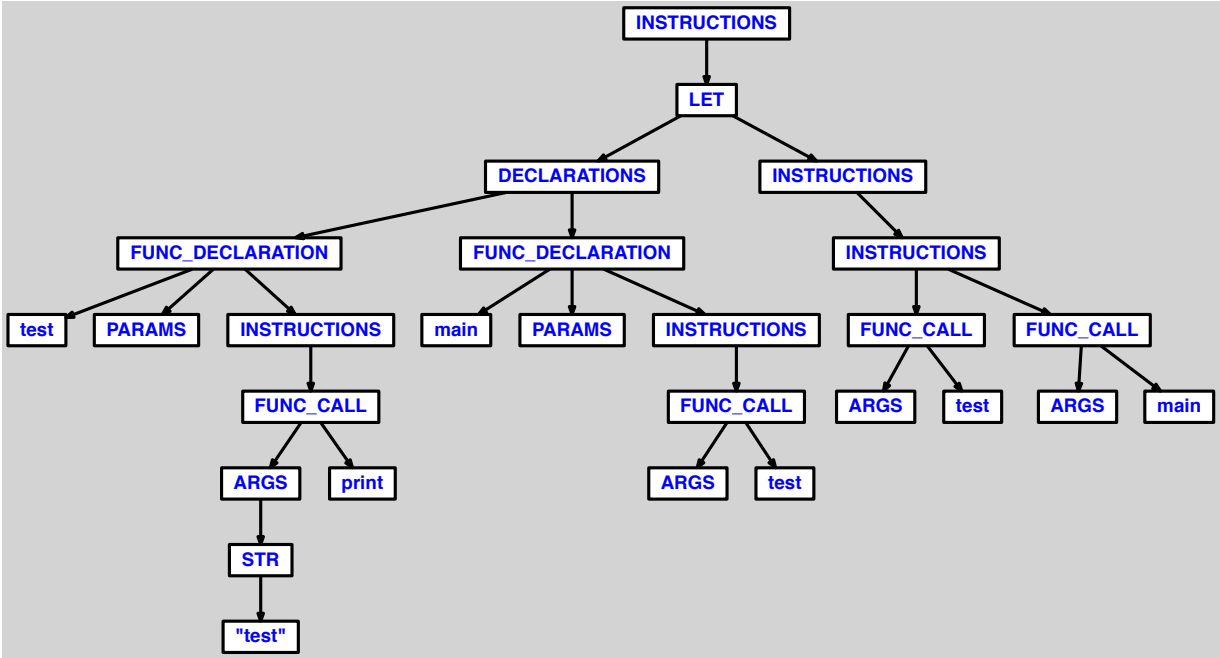
10.2.1 <let> avec 1 declaration de fonction et 1 appel de fonction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
print
STR
"test"

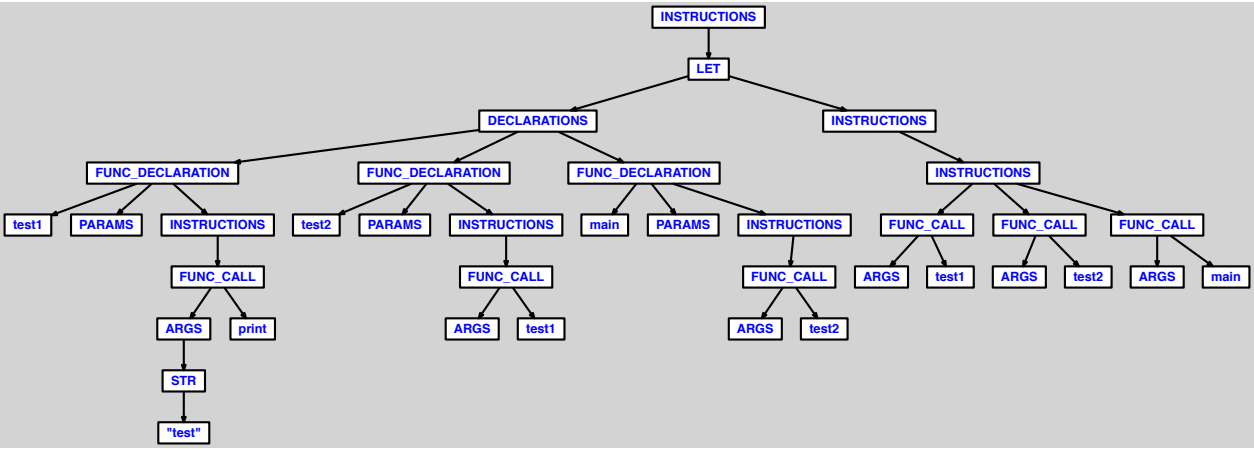
10.2.2 <let> avec 2 declarations de fonction et 2 appels de fonction

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_DECLARATION
INSTRUCTIONS
test
PARAMS
INSTRUCTIONS
main
PARAMS
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
test
ARGS
main
ARGS
print
ARGS
test
STR
"test"

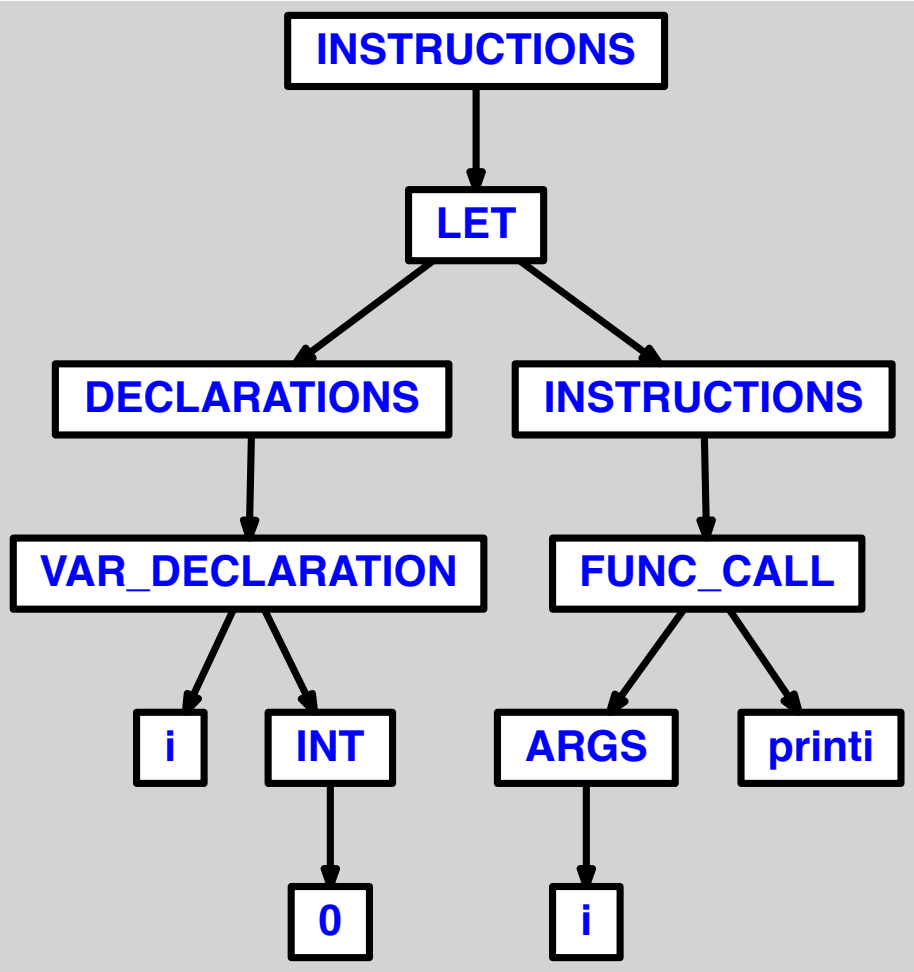
10.2.3 <let> avec 3 declarations de fonction et 3 appels de fonction

```
let
  function test1() = print("test")

  function test2() = test1()

  function main() = test2()
in
  (test1(); test2(); main())
end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_DECLARATION
INSTRUCTIONS
test1
PARAMS
INSTRUCTIONS
test2
PARAMS
INSTRUCTIONS
main
PARAMS
INSTRUCTIONS
FUNC_CALL
FUNC_CALL
FUNC_CALL
FUNC_CALL
FUNC_CALL
FUNC_CALL
ARGS
test1
ARGS
test2
ARGS
main
ARGS
print
ARGS
test1
ARGS
test2
STR
"test"

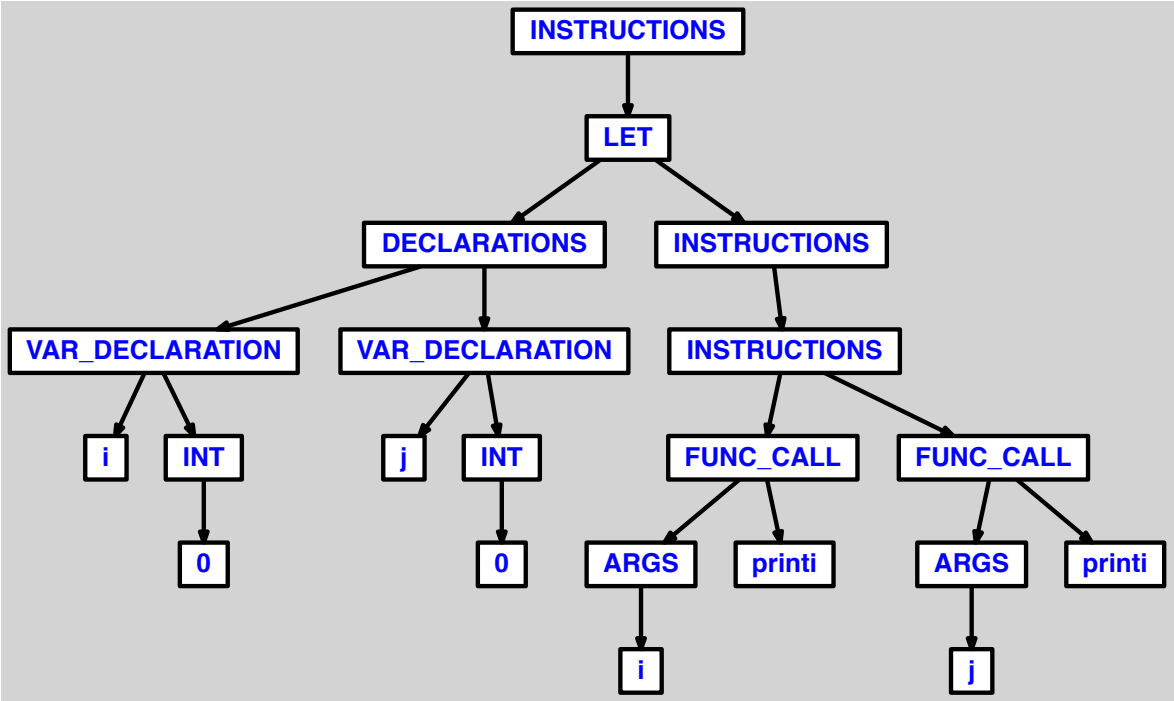
10.2.4 <let> avec 1 declaration de variable et 1 instruction

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
i
INT
ARGS
printi
0
i

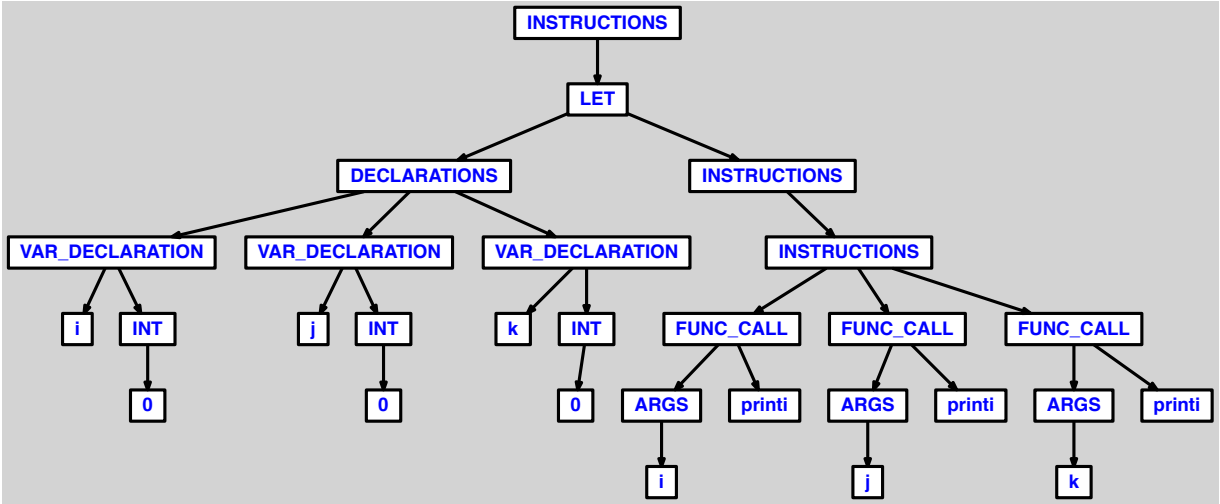
10.2.5 <let> avec 2 declarations de variable et 2 instructions

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
INSTRUCTIONS
i
INT
j
INT
FUNC_CALL
FUNC_CALL
0
0
ARGS
printi
ARGS
printi
i
j

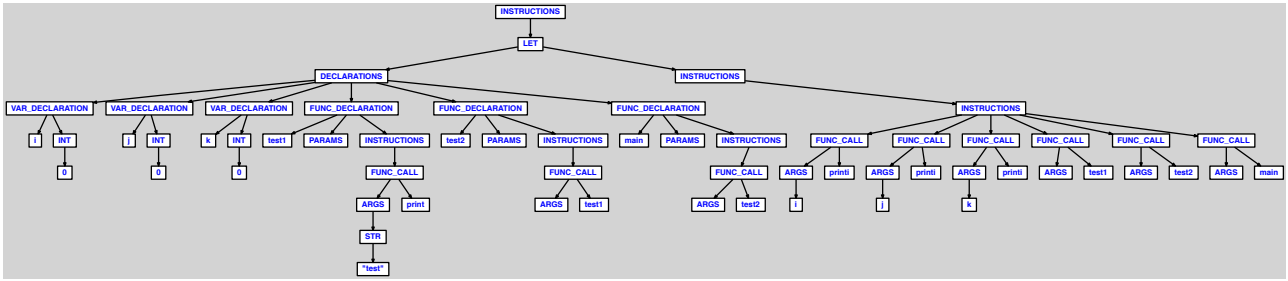
10.2.6 <let> avec 3 declarations de variable et 3 instructions

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
INSTRUCTIONS
i
INT
j
INT
k
INT
FUNC_CALL
FUNC_CALL
FUNC_CALL
0
0
0
ARGS
printi
ARGS
printi
ARGS
printi
i
j
k

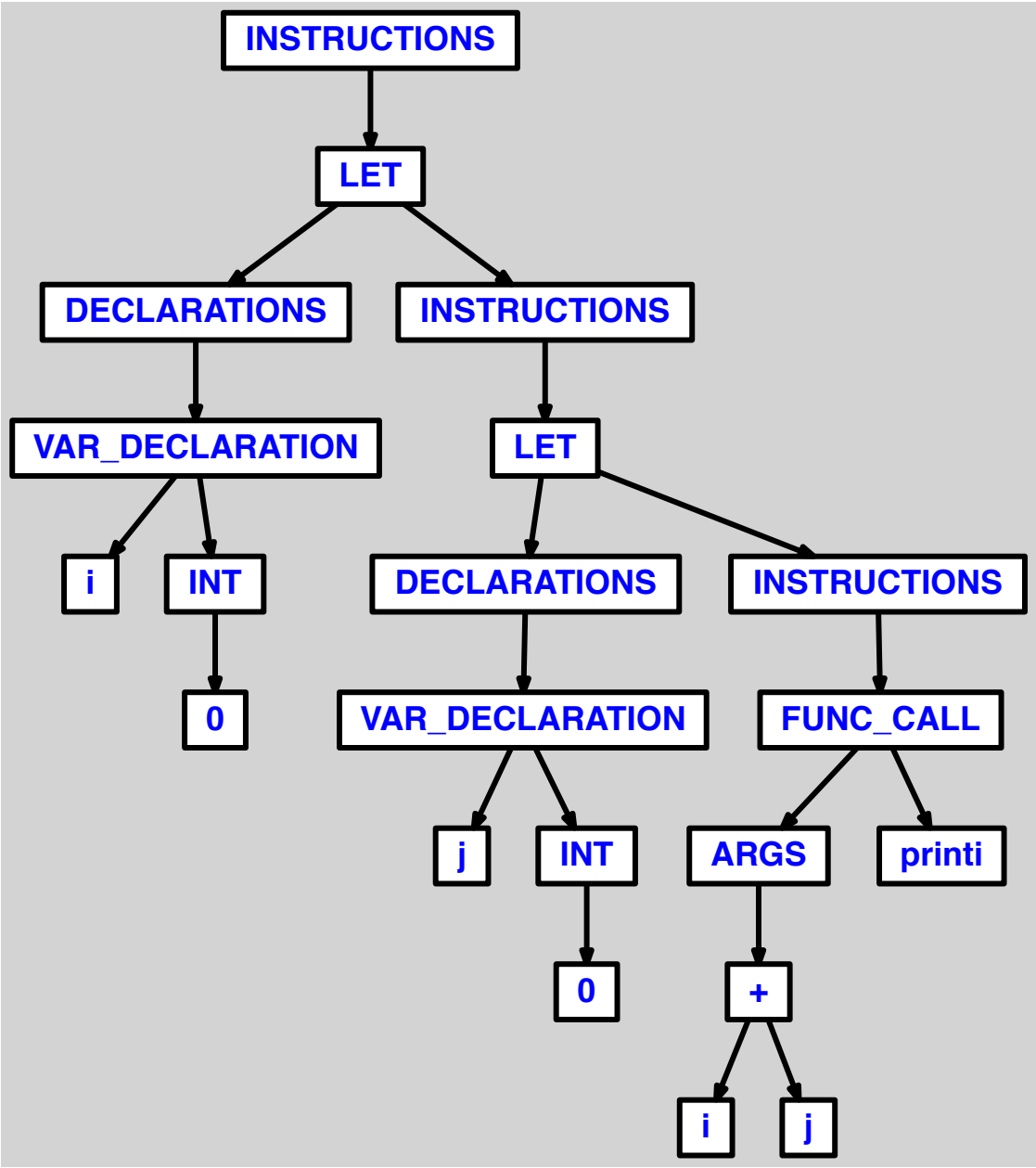
10.2.7 <let> avec 3 declarations de variable et fonction, et 3 instructions et appels de fonction

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

  function test1() = print("test")

  function test2() = test1()

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_DECLARATION
INSTRUCTIONS
i
INT
0
j
INT
0
k
INT
0
test1
PARAMS
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"test"
test2
PARAMS
INSTRUCTIONS
FUNC_CALL
ARGS
test1
main
PARAMS
INSTRUCTIONS
FUNC_CALL
ARGS
test2
FUNC_CALL
ARGS
printi
i
FUNC_CALL
ARGS
printi
j
FUNC_CALL
ARGS
printi
k
FUNC_CALL
ARGS
test1
FUNC_CALL
ARGS
test2
FUNC_CALL
ARGS
main

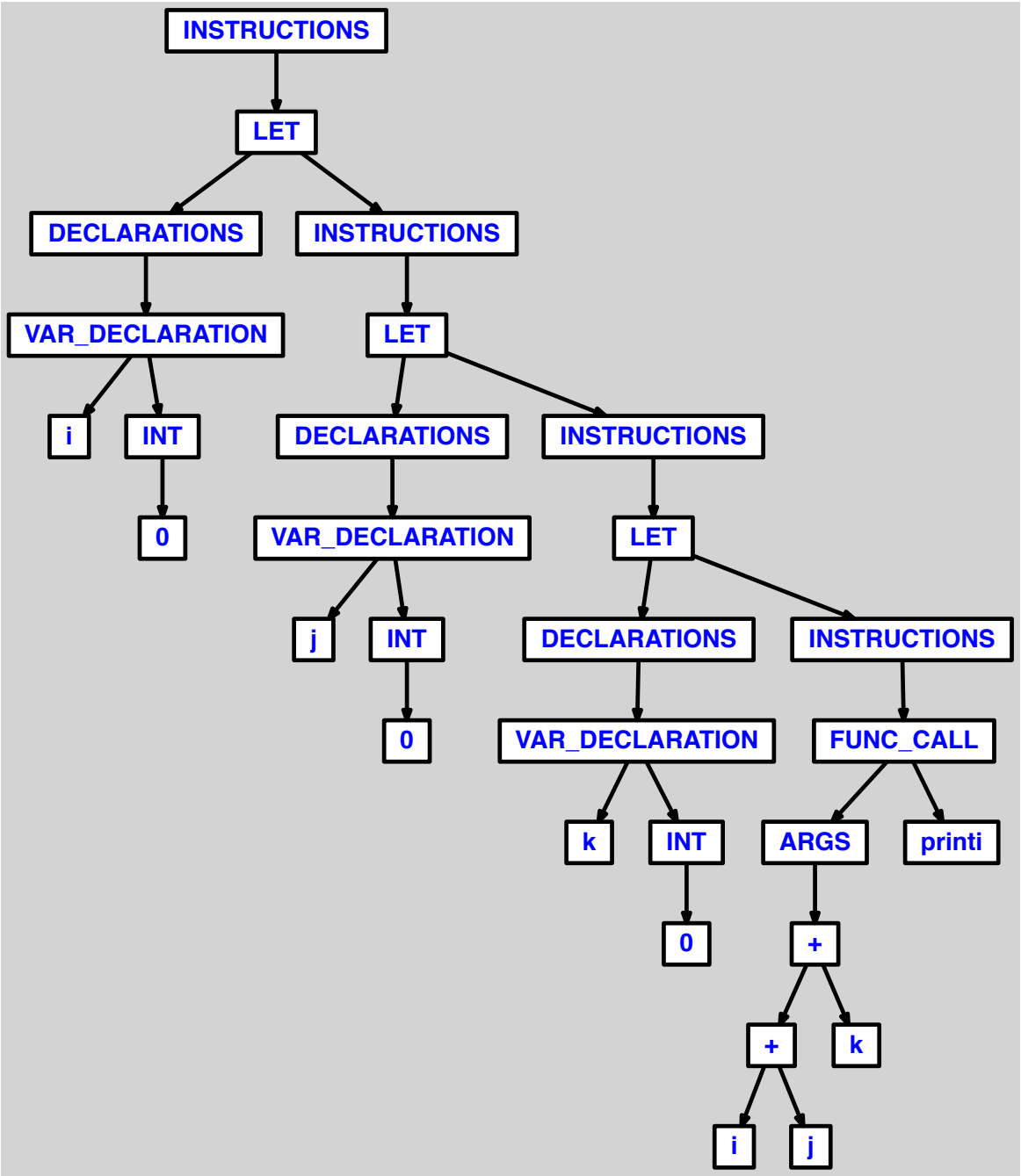
10.2.8 <let> avec double-imbrication

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
LET
i
INT
DECLARATIONS
INSTRUCTIONS
0
VAR_DECLARATION
FUNC_CALL
j
INT
ARGS
printi
0
+
i
j

10.2.9 <let> avec triple-imbrication

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

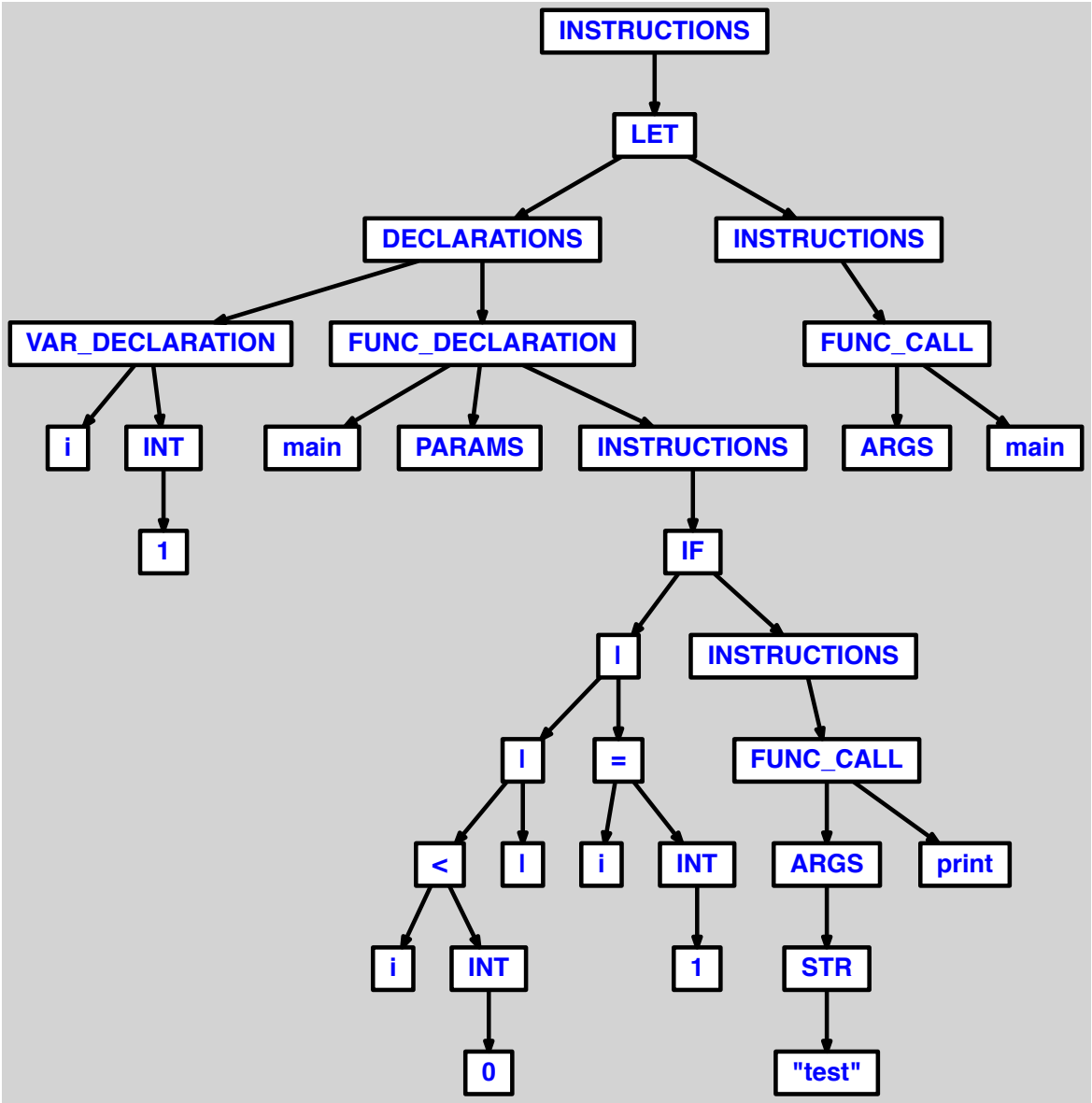

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
LET
i
INT
DECLARATIONS
INSTRUCTIONS
0
VAR_DECLARATION
LET
j
INT
DECLARATIONS
INSTRUCTIONS
0
VAR_DECLARATION
FUNC_CALL
k
INT
ARGS
printi
0
+
+
k
i
j

11 or

11.1 KO

11.1.1 <|> mal ecrit

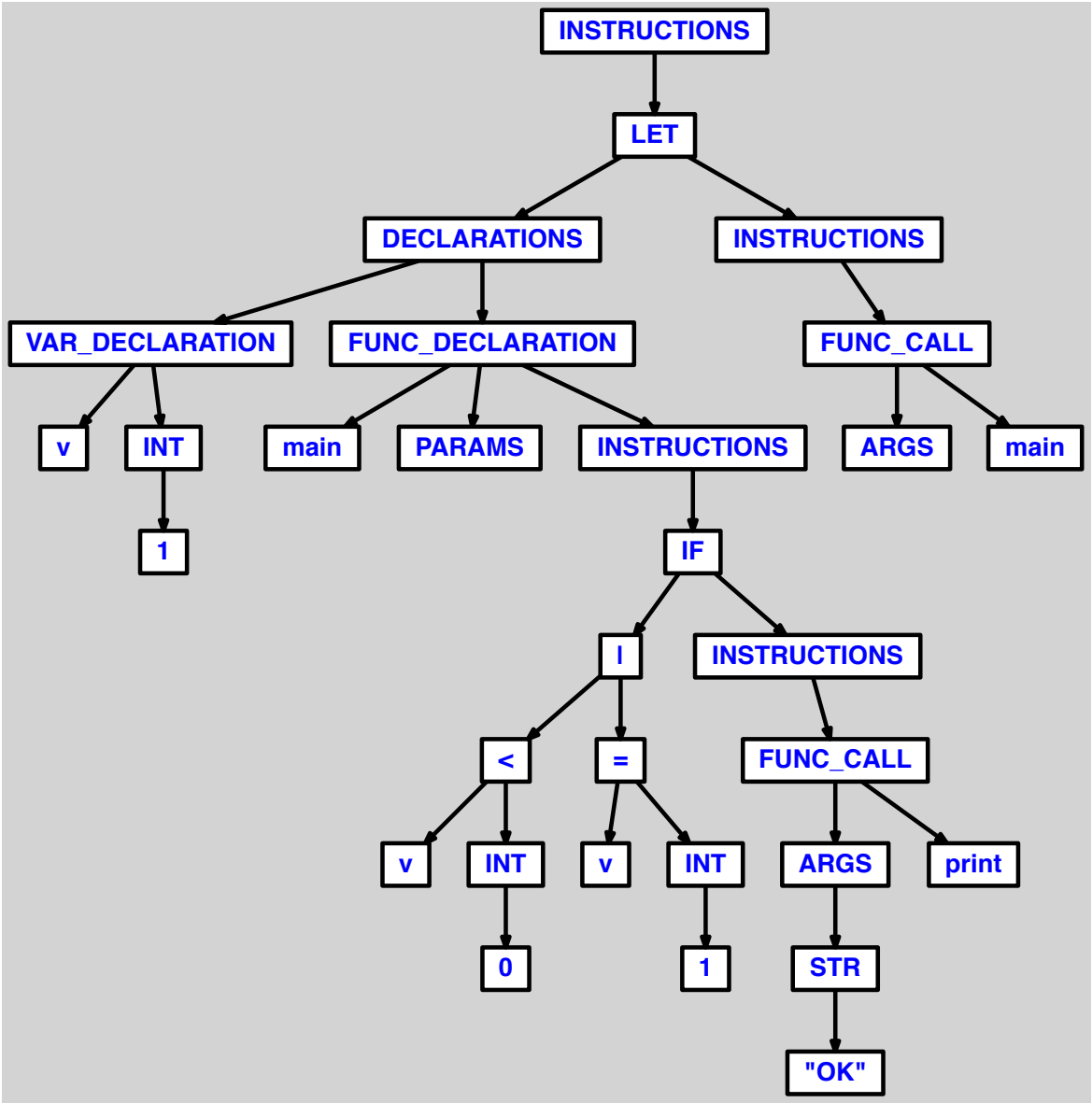
```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
|
INSTRUCTIONS
|
=
FUNC_CALL
<
|
i
INT
ARGS
print
i
INT
1
STR
0
"test"

11.2 OK

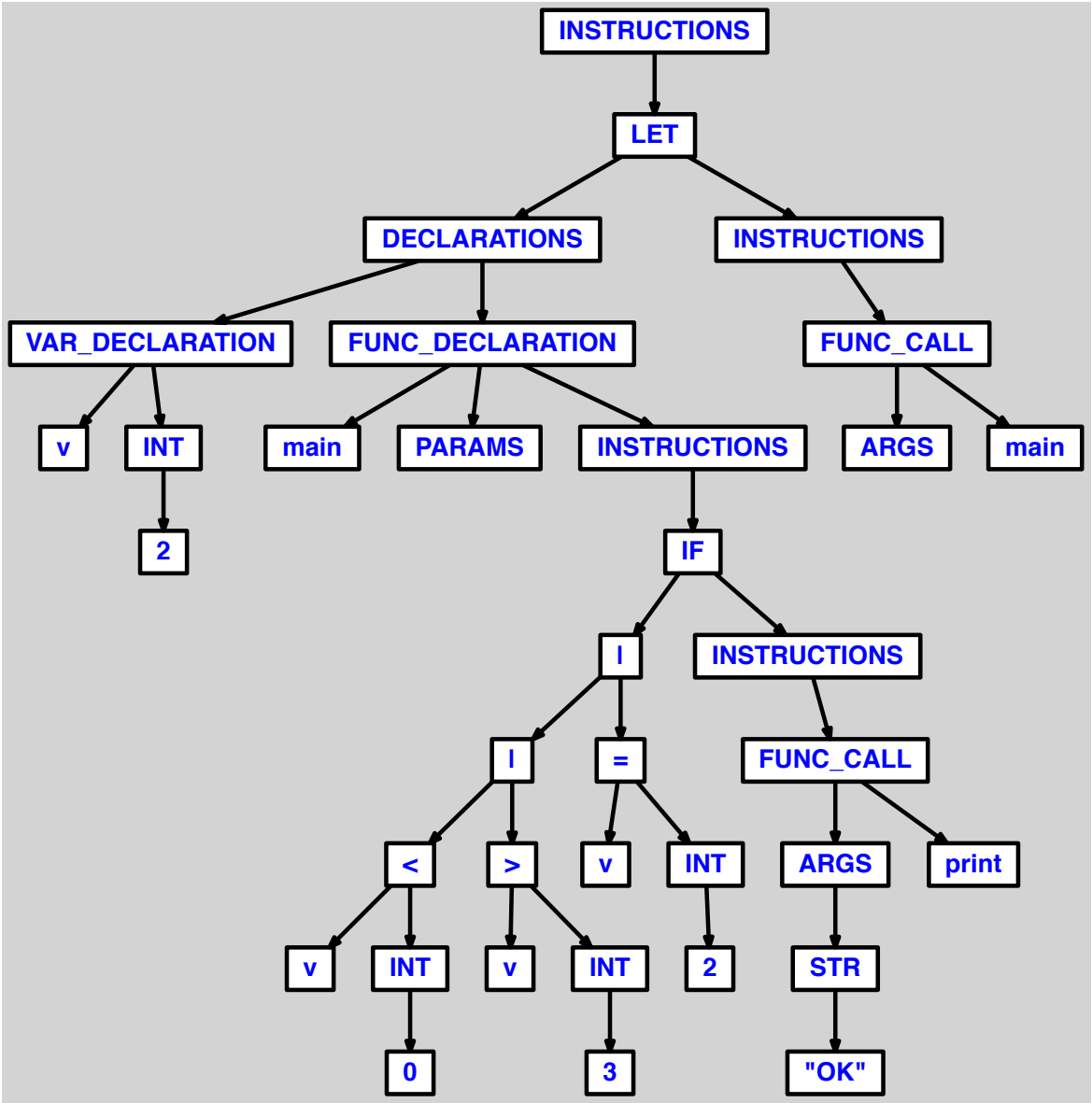
11.2.1 condition avec 1 <|>

```
let
  var v := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
|
INSTRUCTIONS
<
=
FUNC_CALL
v
INT
v
INT
ARGS
print
0
1
STR
"OK"

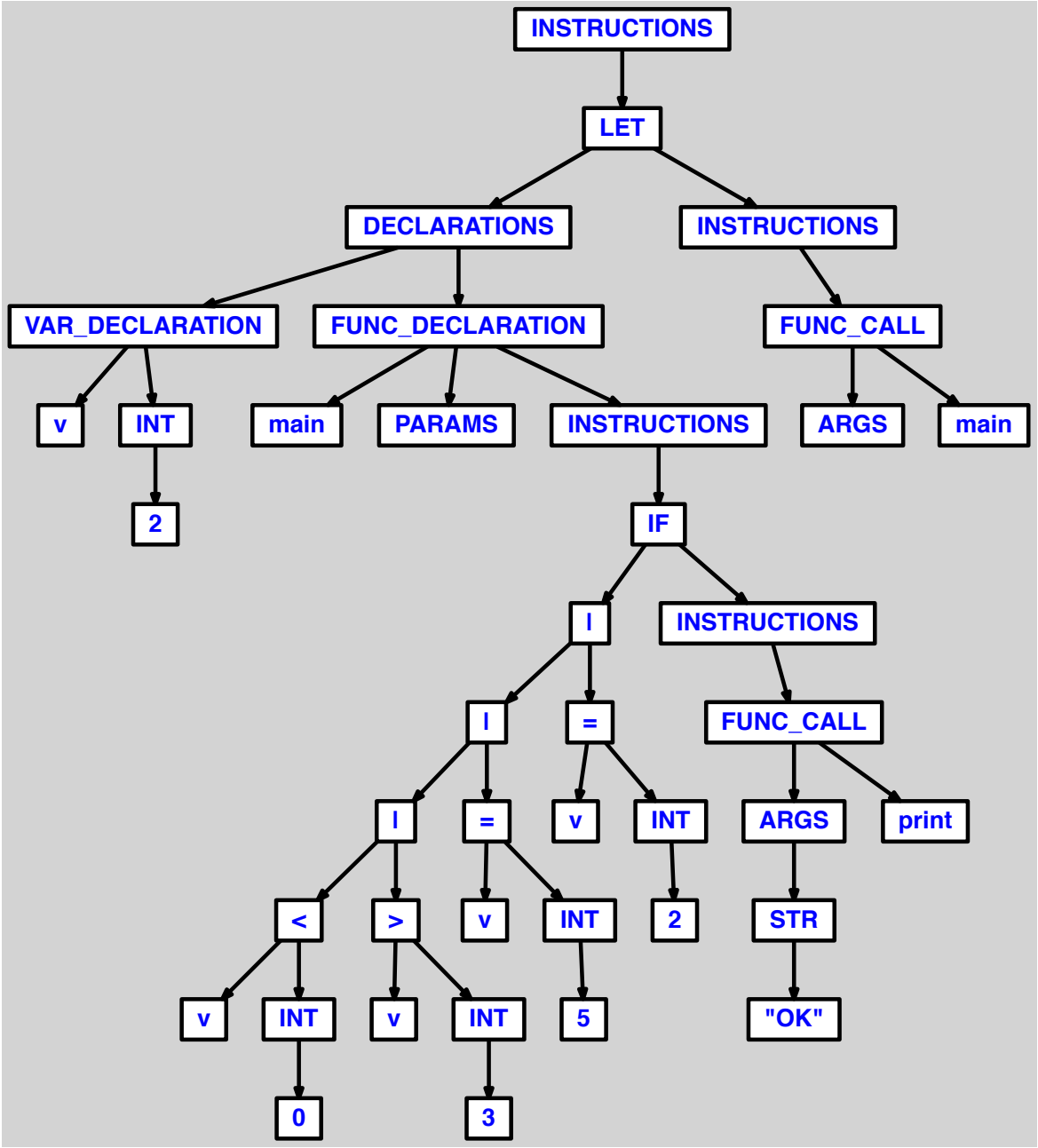
11.2.2 condition avec 2 <|>

```
let
  var v := 2

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
2
IF
|
INSTRUCTIONS
|
=
FUNC_CALL
<
>
v
INT
ARGS
print
v
INT
v
INT
2
STR
0
3
"OK"

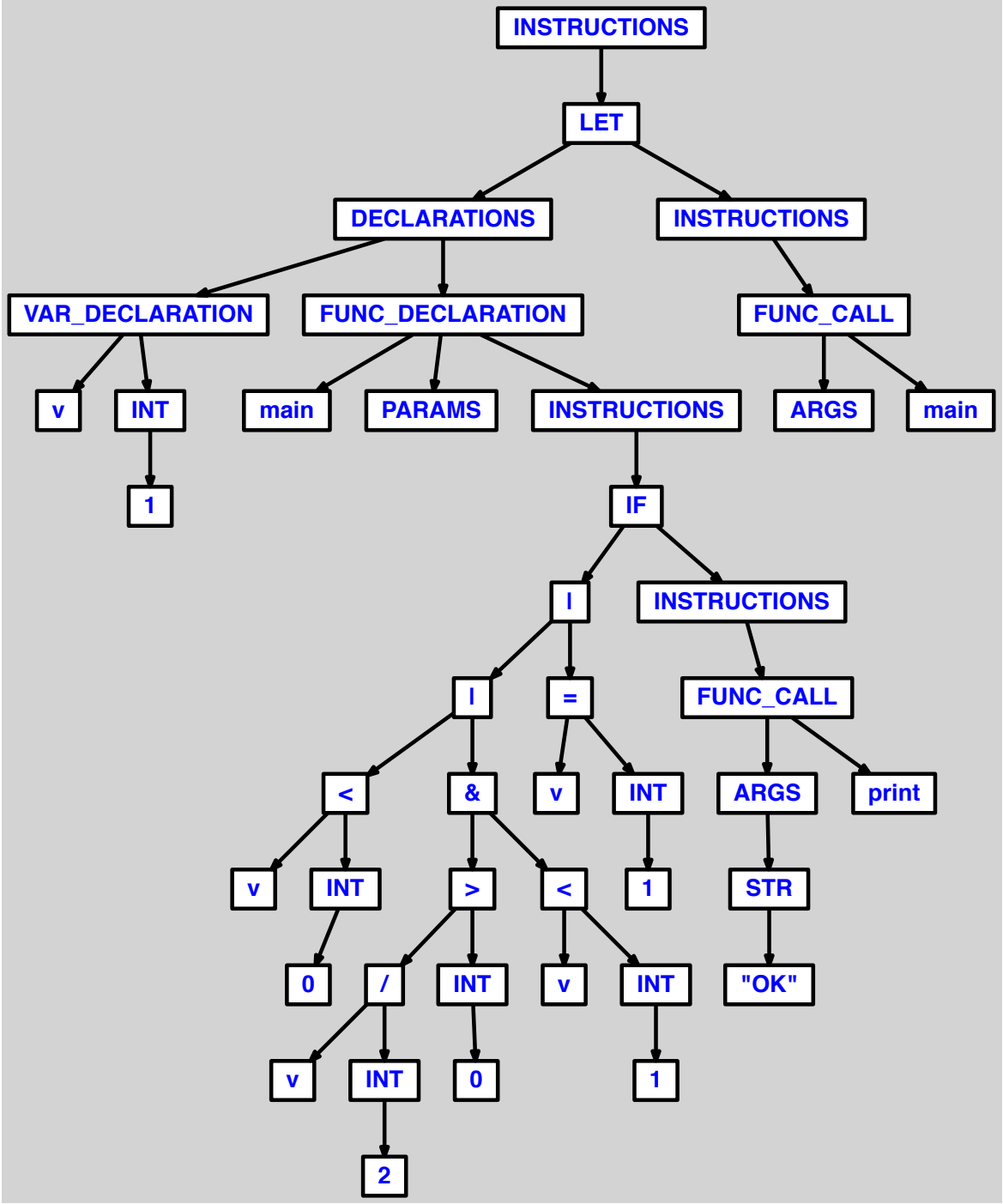
11.2.3 condition avec 3 <|>

```
let
  var v := 2

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
2
IF
|
INSTRUCTIONS
|
=
FUNC_CALL
|
=
v
INT
ARGS
print
<
>
v
INT
2
STR
v
INT
v
INT
5
"OK"
0
3

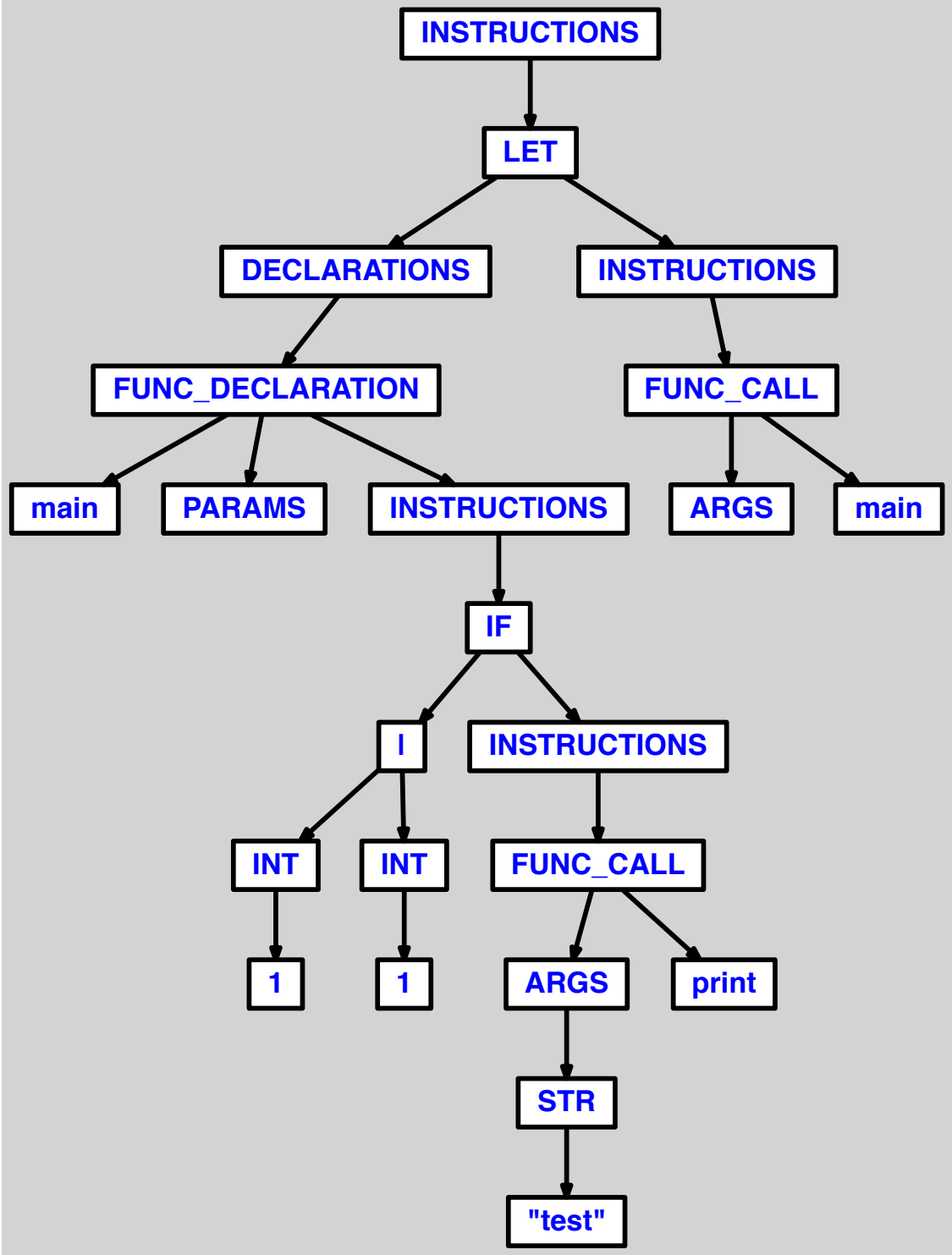
11.2.4 condition avec 2 <|> et 1 <&>

```
let
  var v := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
IF
|
INSTRUCTIONS
|
=
FUNC_CALL
<
&
v
INT
ARGS
print
v
INT
>
<
1
STR
0
/
INT
v
INT
"OK"
v
INT
0
1
2

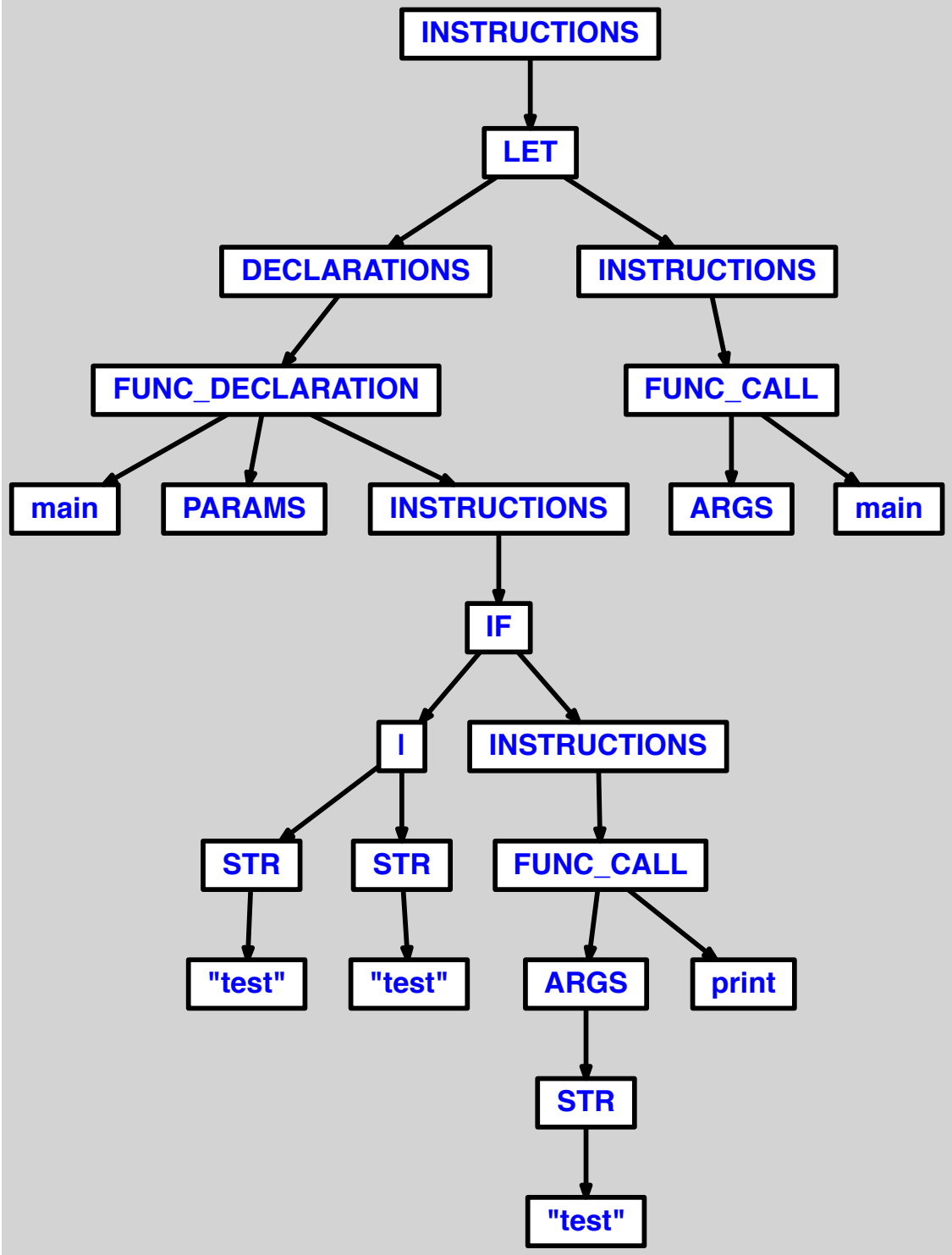
11.2.5 <|> avec des entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
I
INSTRUCTIONS
INT
INT
FUNC_CALL
1
1
ARGS
print
STR
"test"

11.2.6 <|> avec des chaines

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

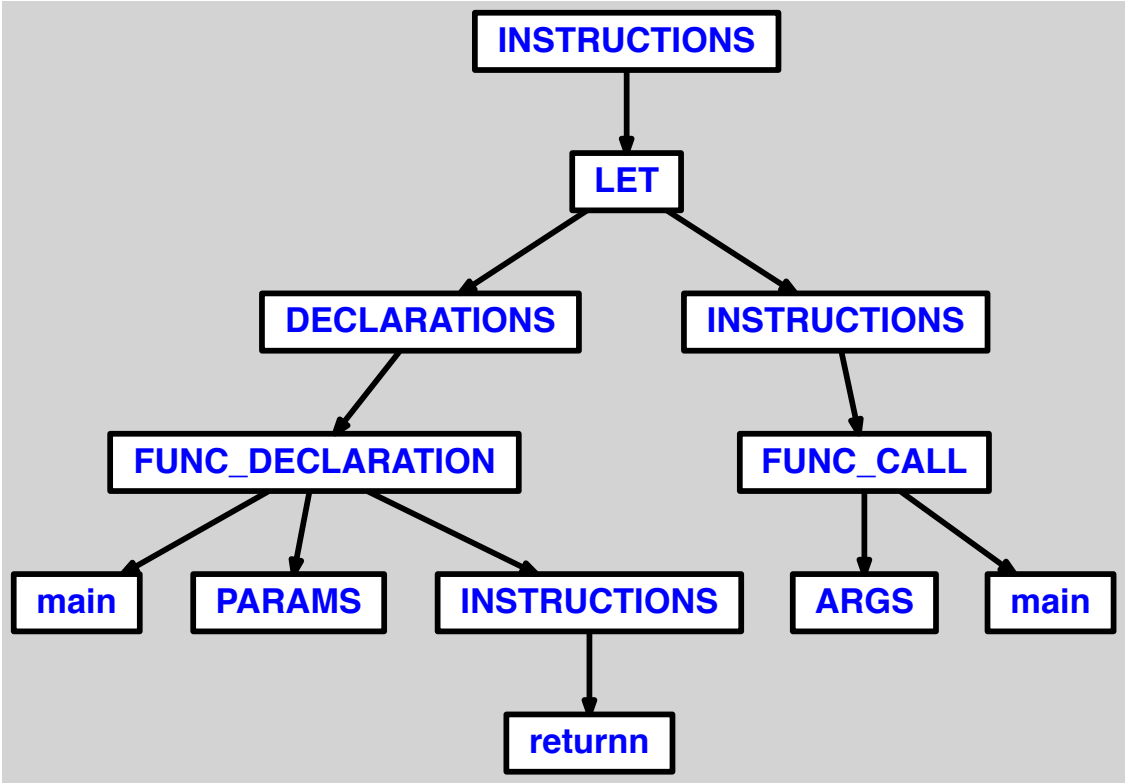

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
IF
I
INSTRUCTIONS
STR
STR
FUNC_CALL
"test"
"test"
ARGS
print
STR
"test"

12 return

12.1 KO

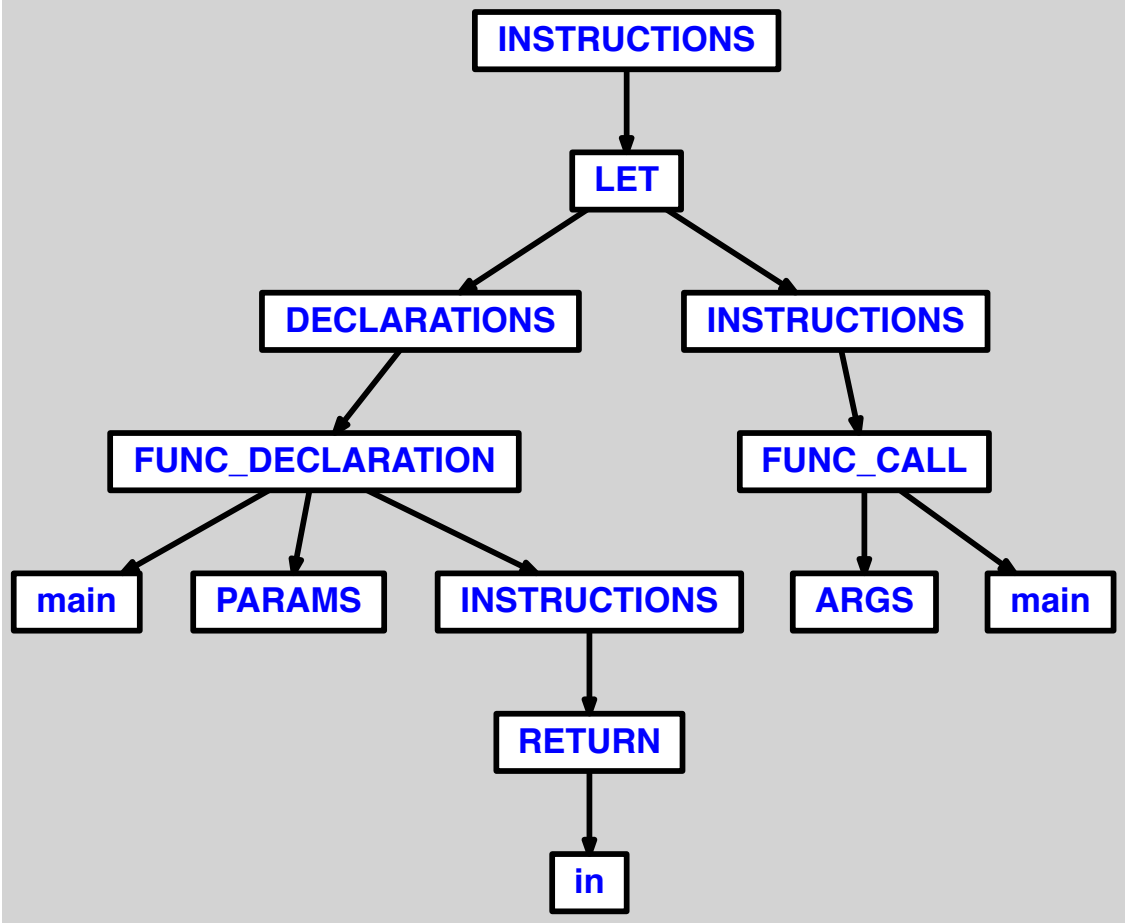
12.1.1 <return> mal ecrit

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
returnn

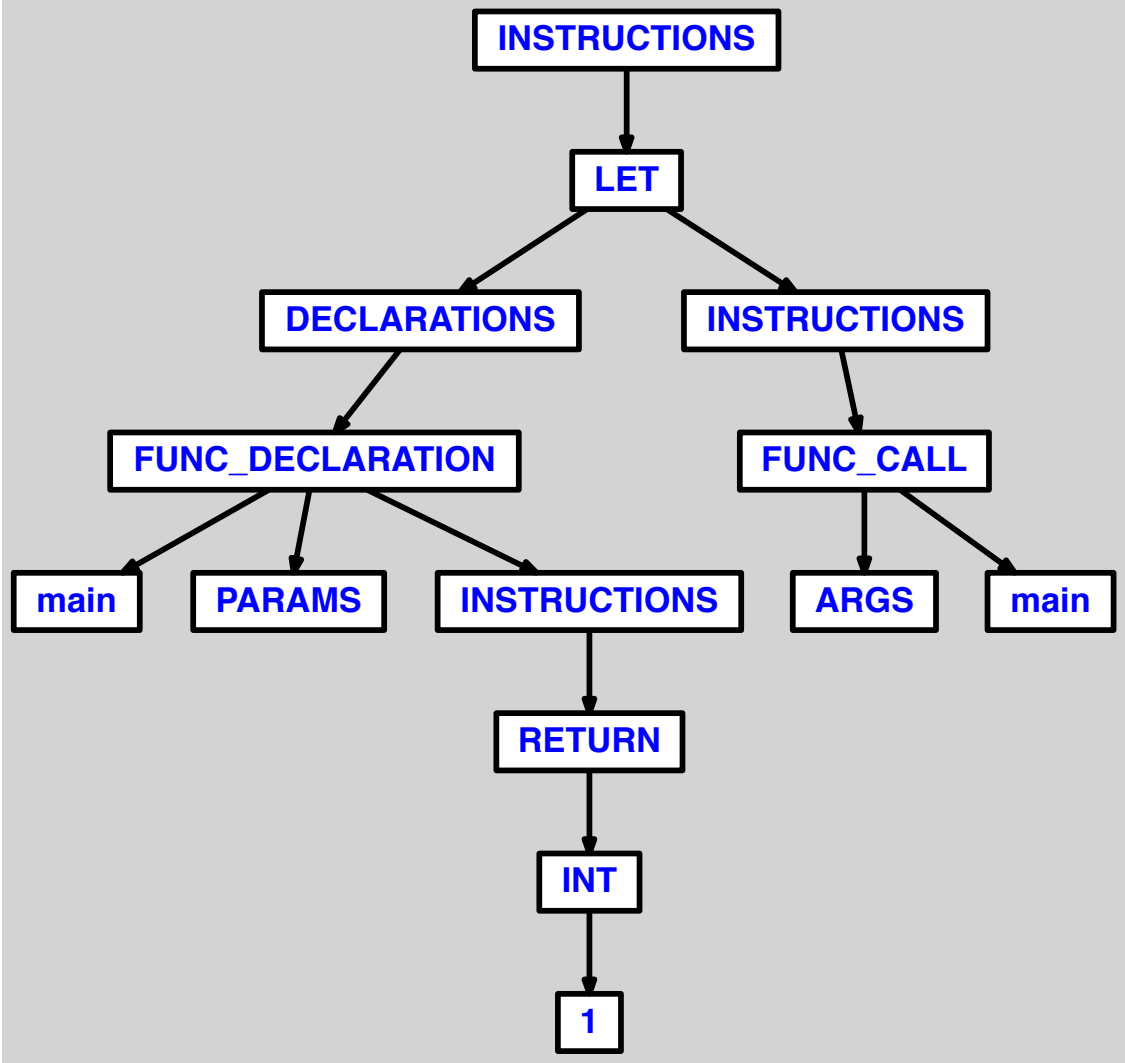
12.1.2 <return> sans valeur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
in

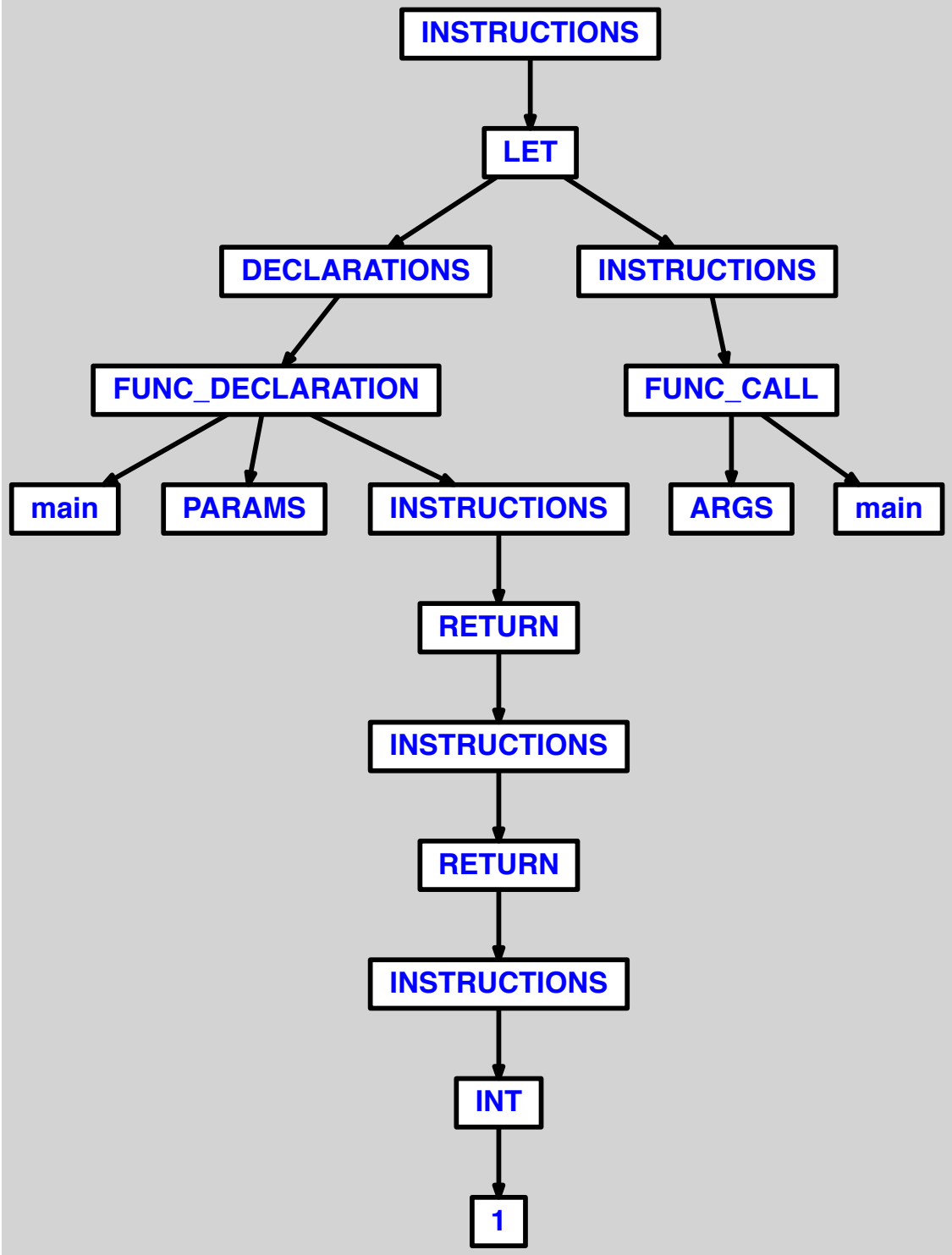
12.1.3 <return> avec oubli de parentheses

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INT
1

12.1.4 2 <return>

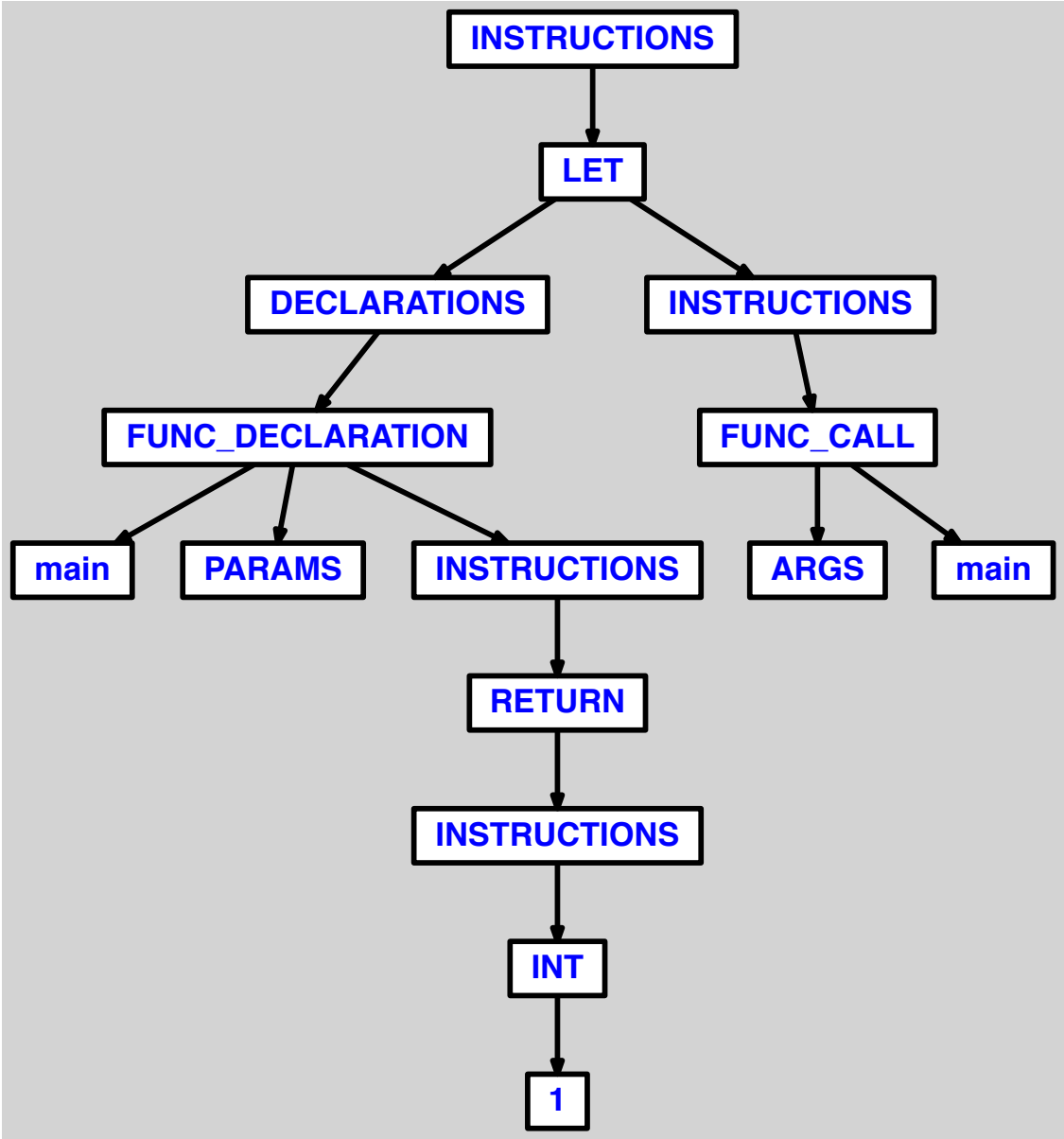
```
let
  function main() = return (return (1))
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INSTRUCTIONS
RETURN
INSTRUCTIONS
INT
1

12.2 OK

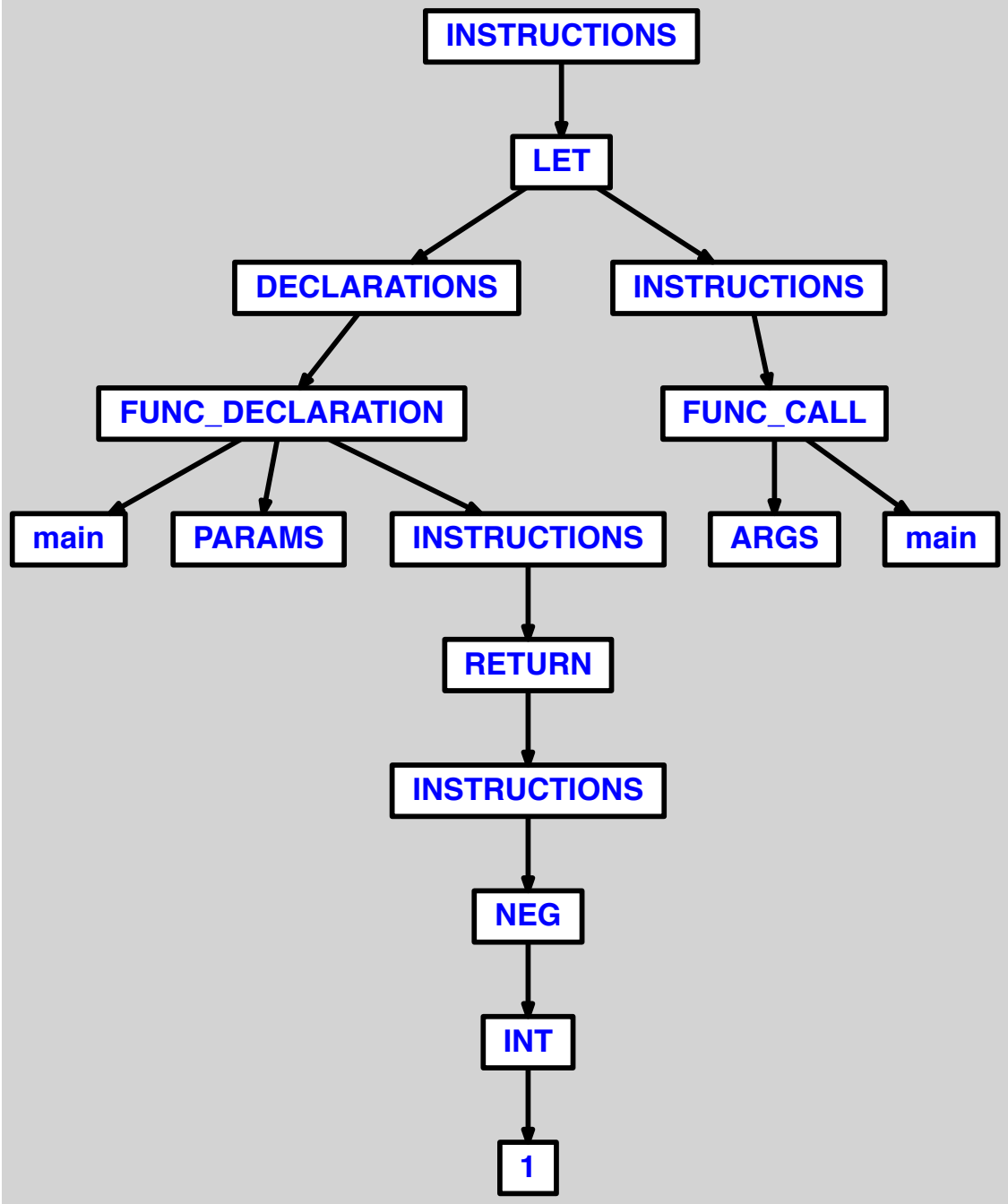
12.2.1 retour d'entier positif

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INSTRUCTIONS
INT
1

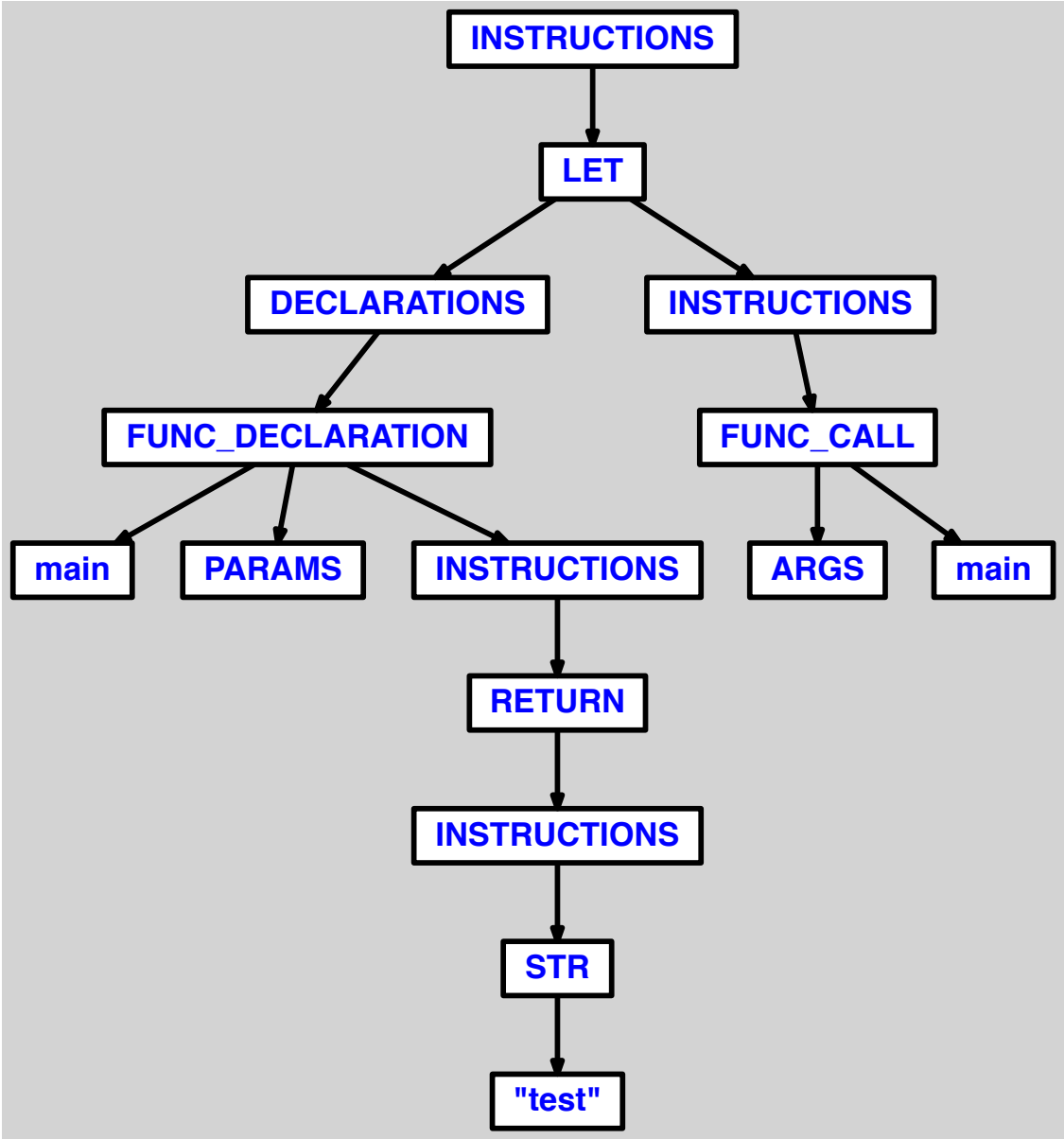
12.2.2 retour d'entier negatif

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INSTRUCTIONS
NEG
INT
1

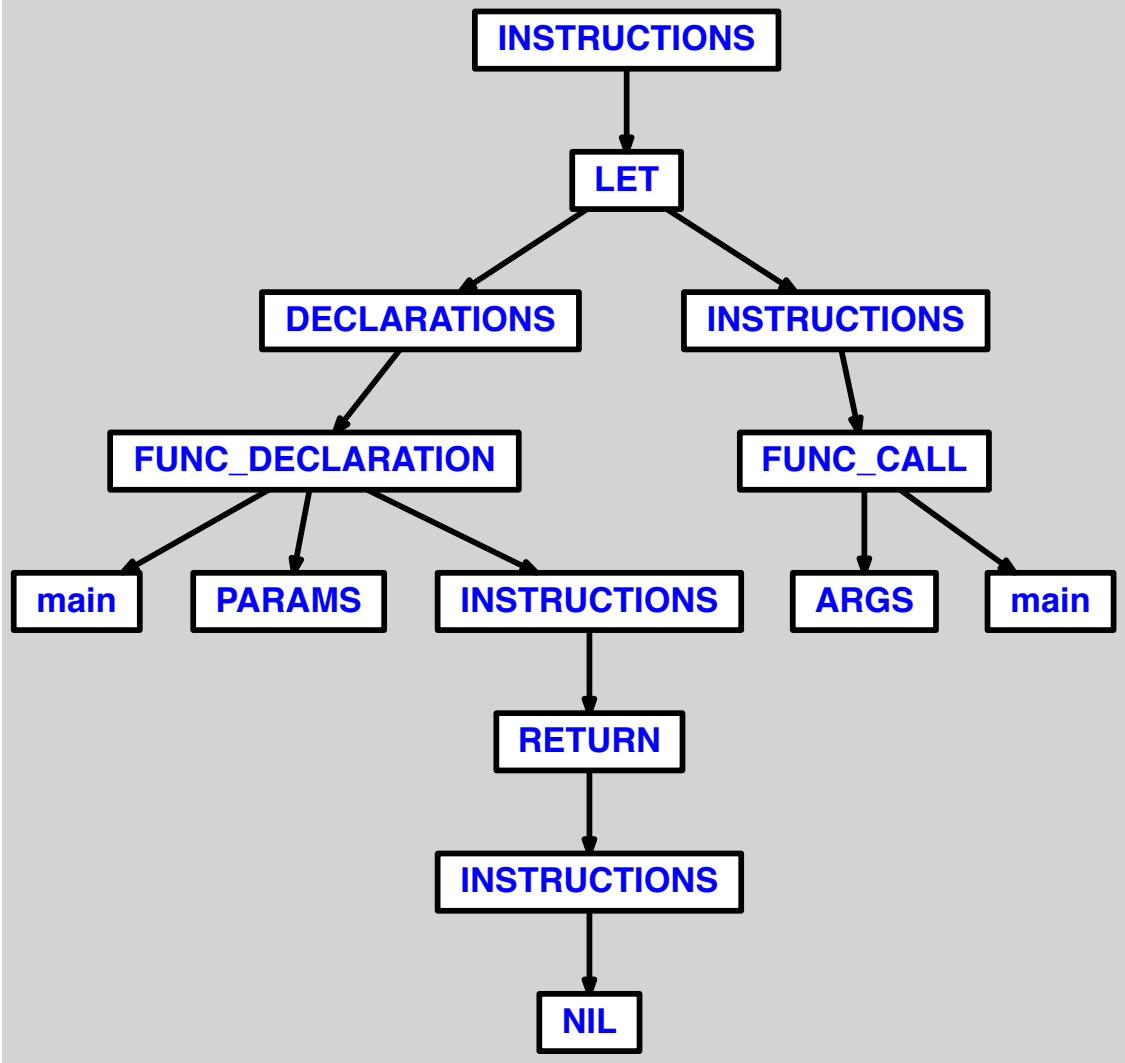
12.2.3 retour de chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INSTRUCTIONS
STR
"test"

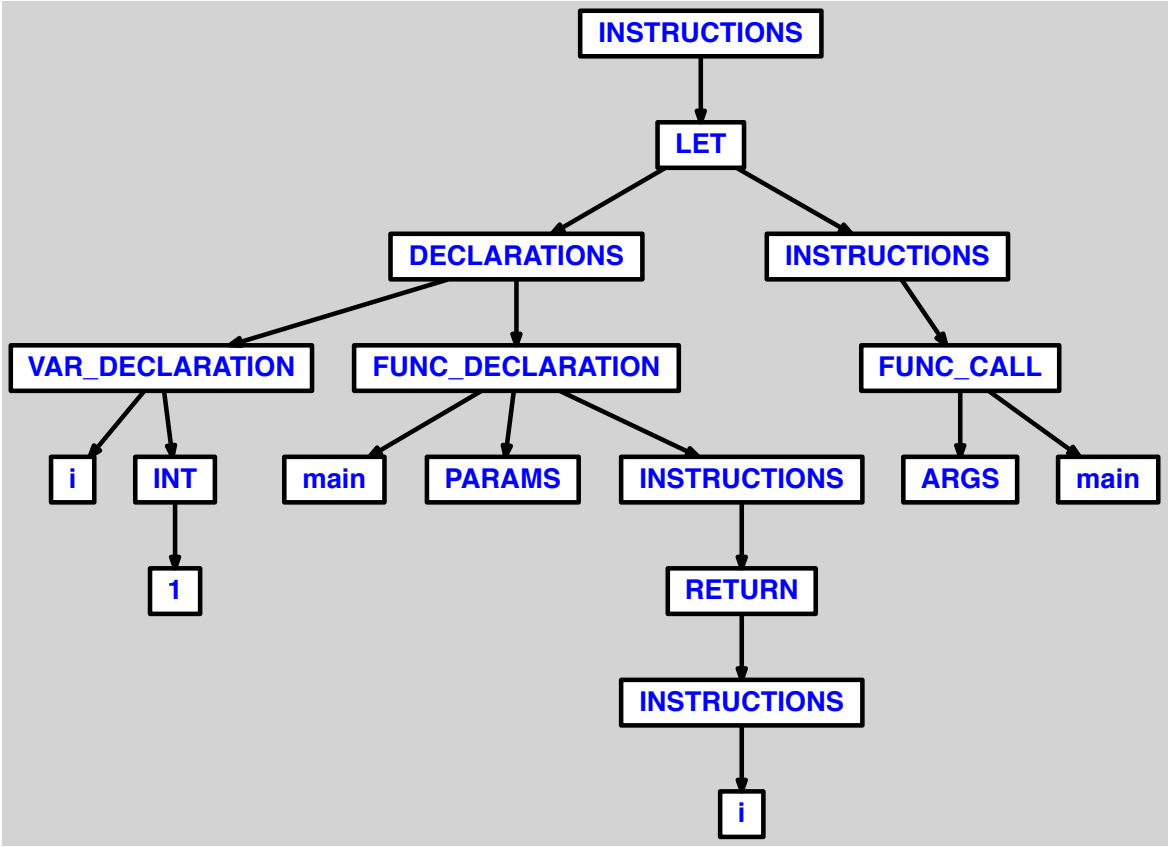
12.2.4 retour de <nil>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INSTRUCTIONS
NIL

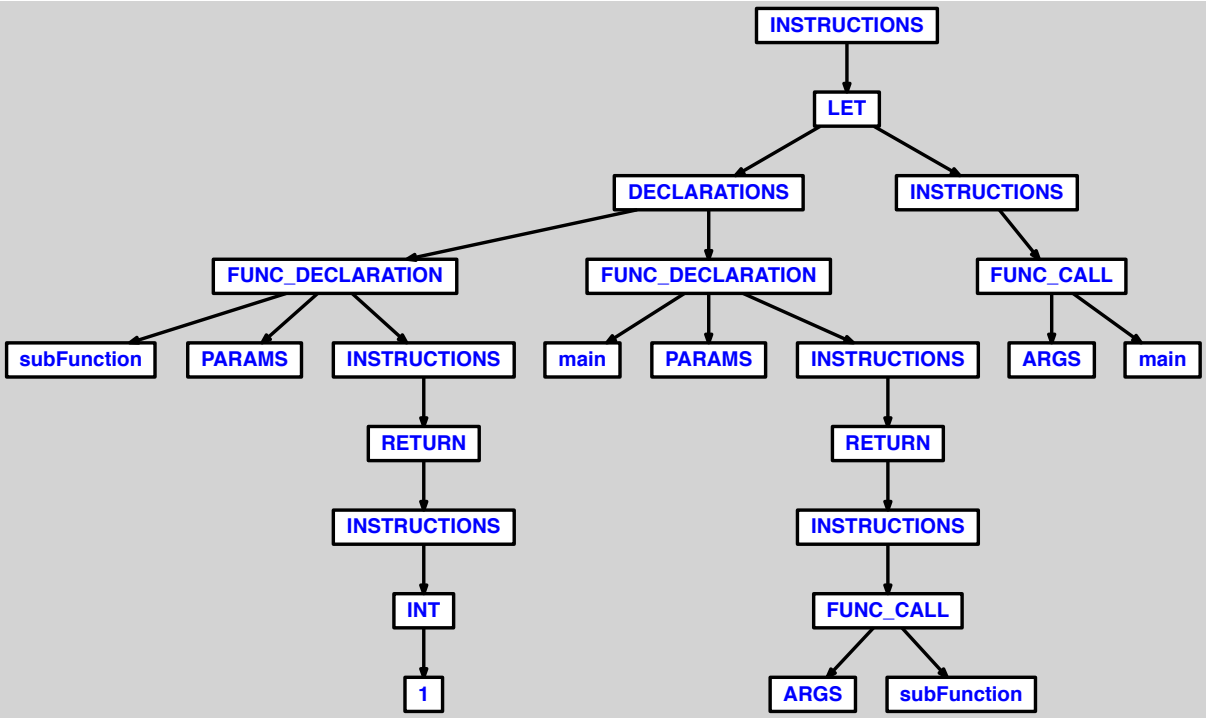
12.2.5 retour de variable

```
let
  var i := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
RETURN
INSTRUCTIONS
i

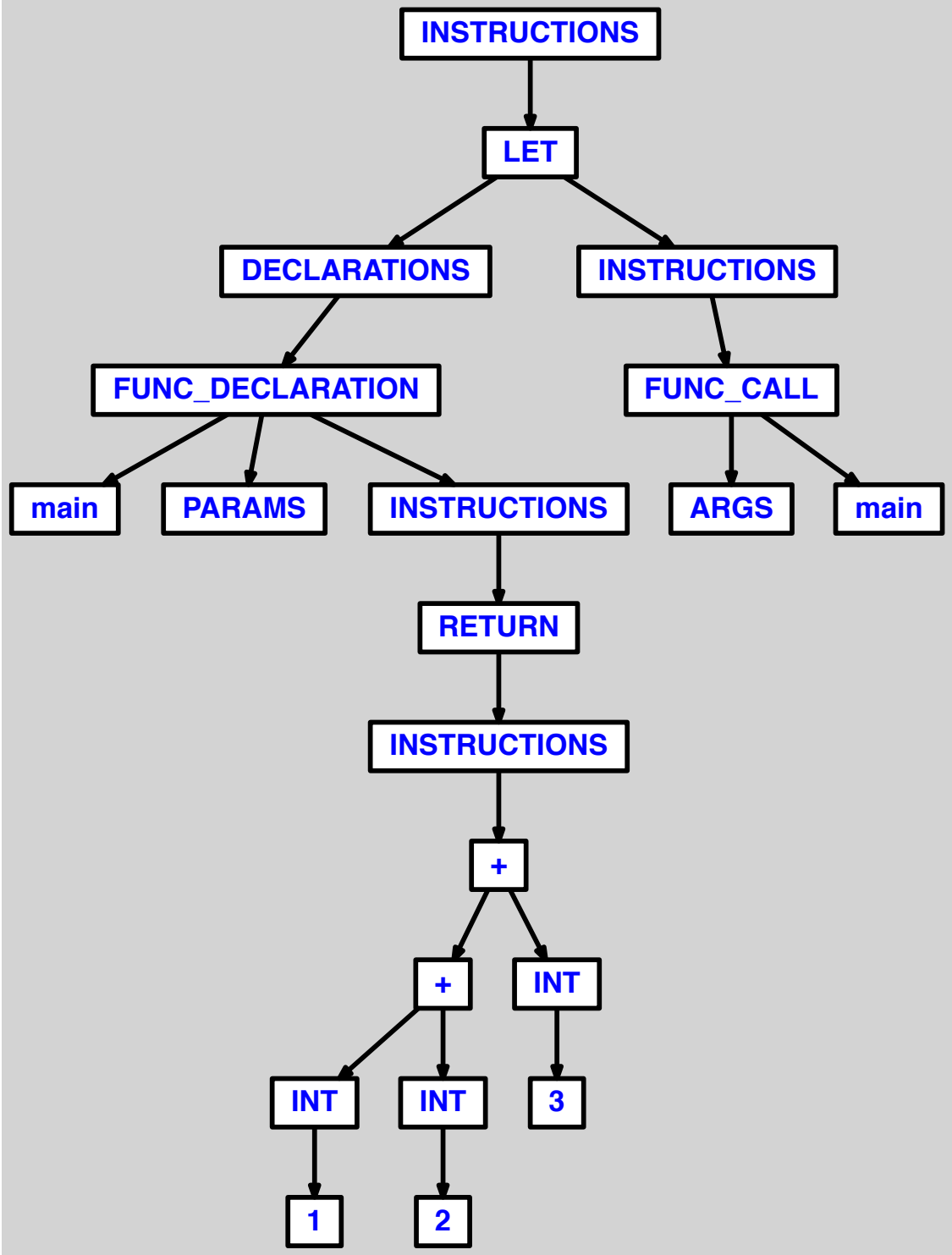
12.2.6 retour de fonction

```
let
  function subFunction() = return (1)

  function main() = return (subFunction())
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_DECLARATION
FUNC_CALL
subFunction
PARAMS
INSTRUCTIONS
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
RETURN
INSTRUCTIONS
INSTRUCTIONS
INT
FUNC_CALL
1
ARGS
subFunction

12.2.7 retour d'expression

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

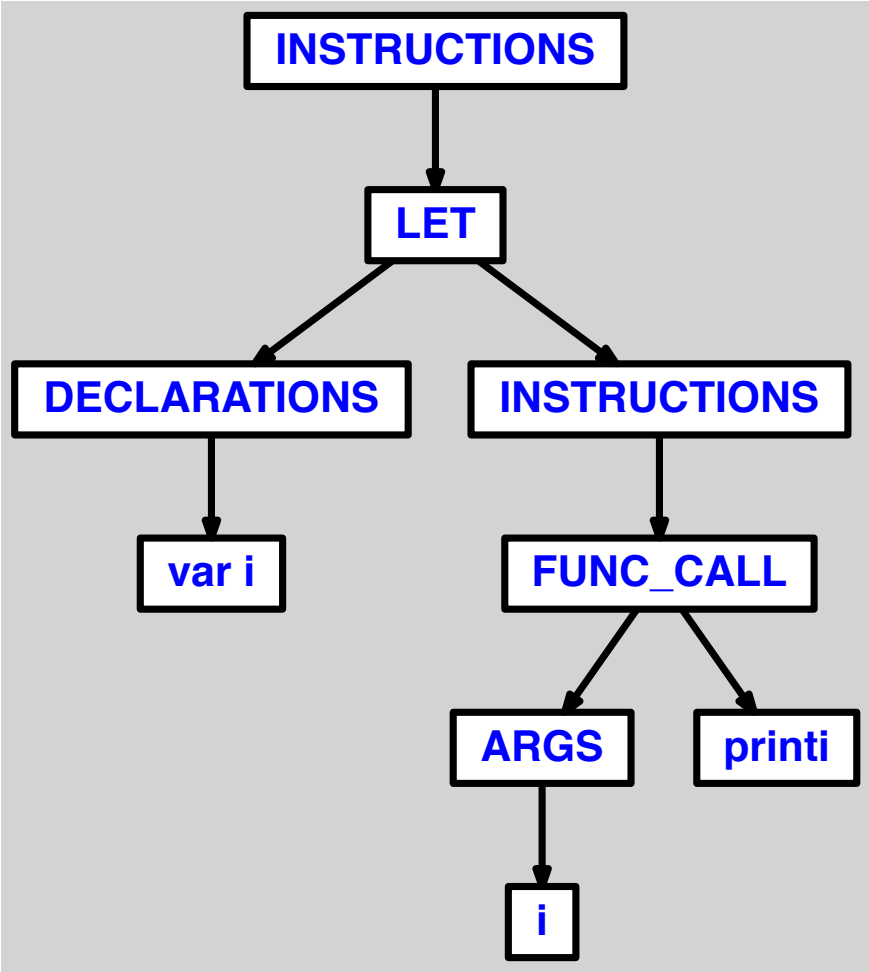

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
RETURN
INSTRUCTIONS
+
+
INT
INT
INT
3
1
2

13 var

13.1 KO

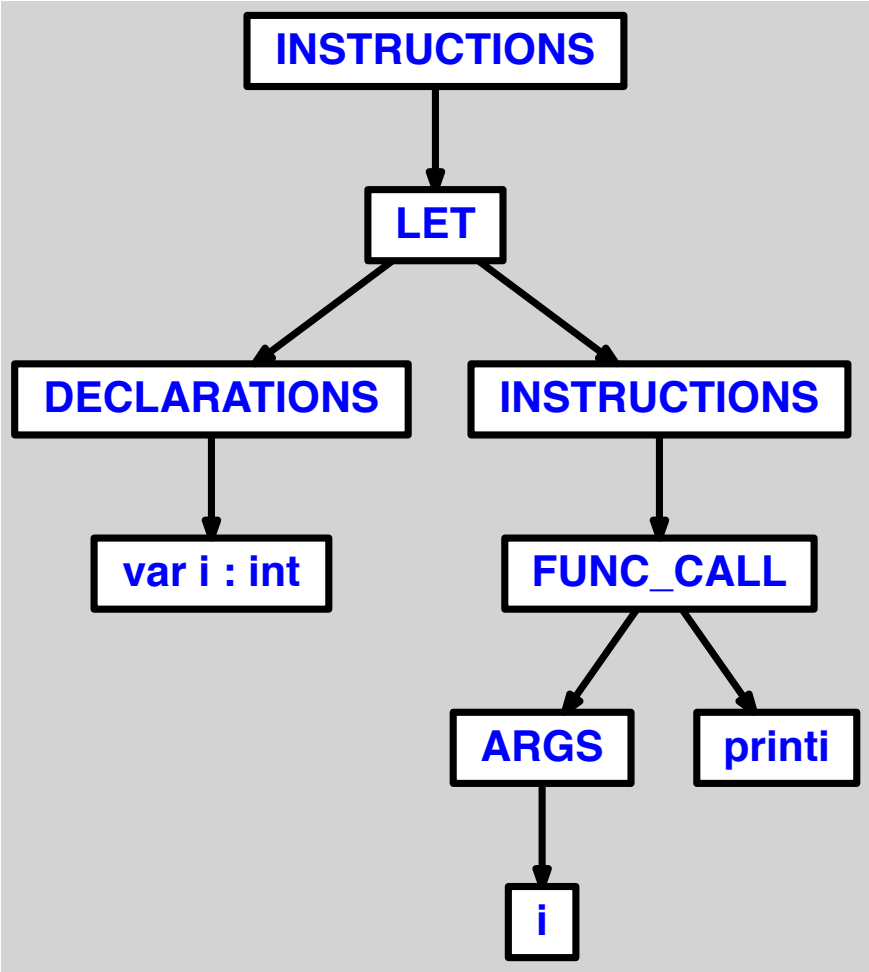
13.1.1 declaration sans affectation

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var i
FUNC_CALL
ARGS
printi
i

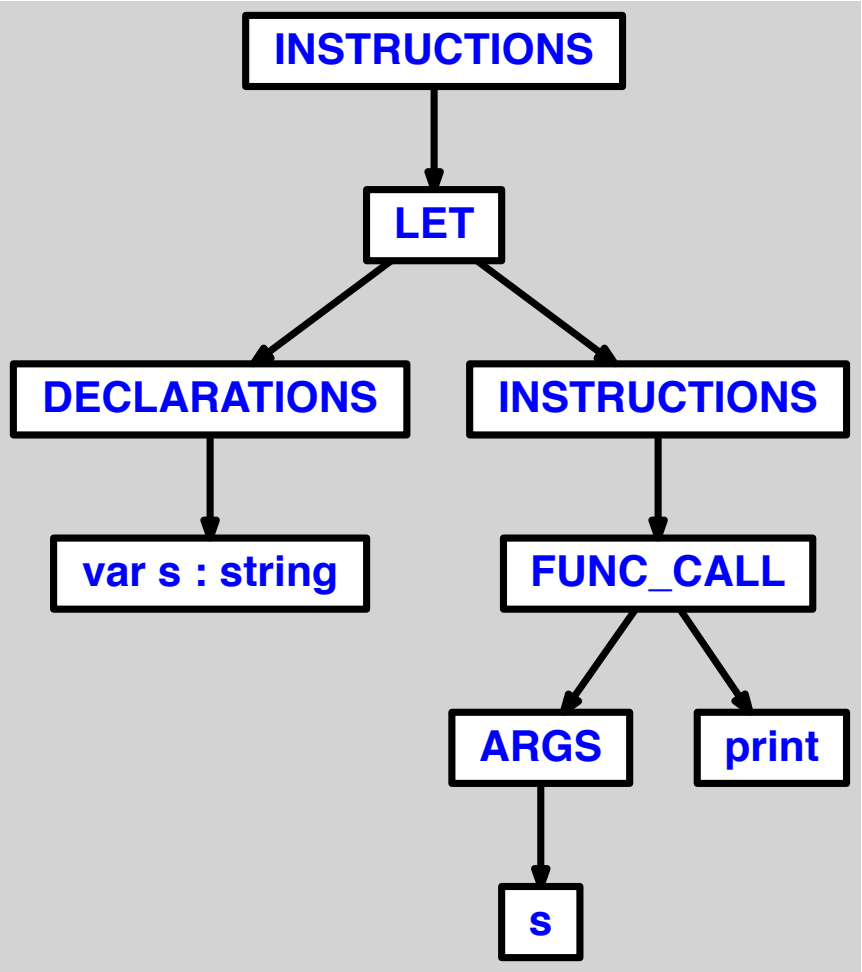
13.1.2 declaration d'entier avec type sans affectation

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var i : int
FUNC_CALL
ARGS
printi
i

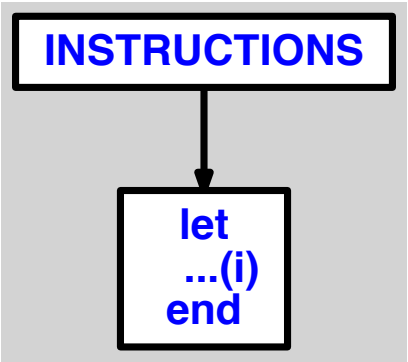
13.1.3 declaration de chaine avec type sans affectation

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var s : string
FUNC_CALL
ARGS
print
s

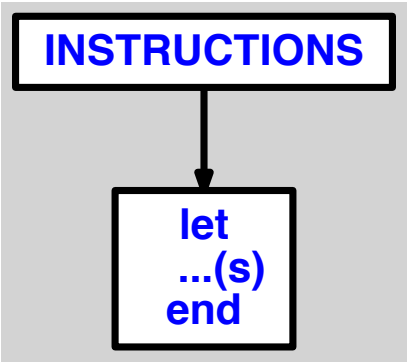
13.1.4 affectation d'entier sans declaration

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


INSTRUCTIONS
let
...(i)
end

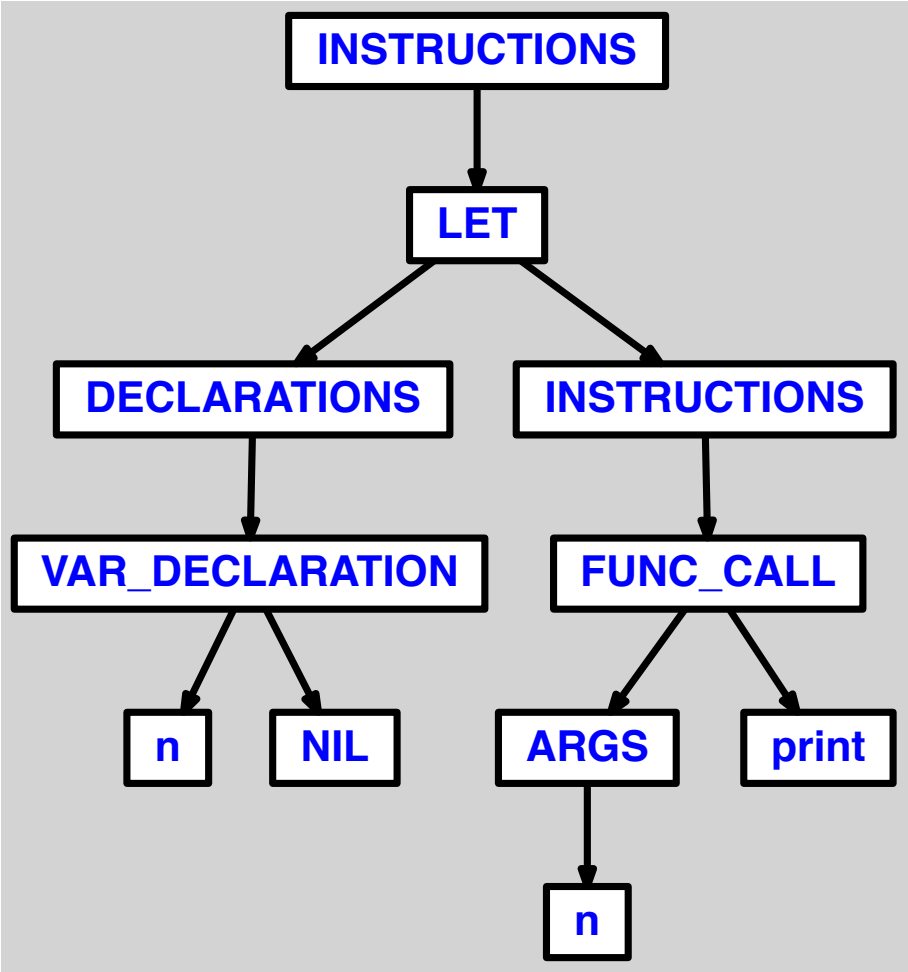
13.1.5 affectation de chaine sans declaration

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


INSTRUCTIONS
let
...(s)
end

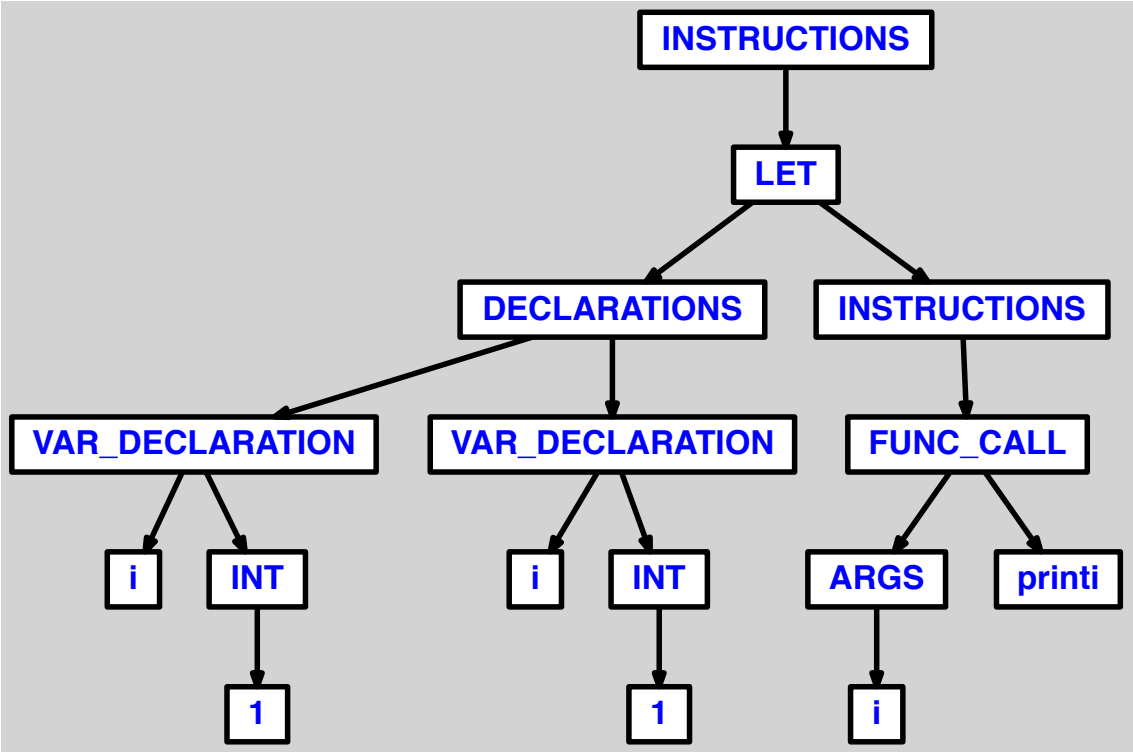
13.1.6 affectation de <nil>

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
n
NIL
ARGS
print
n

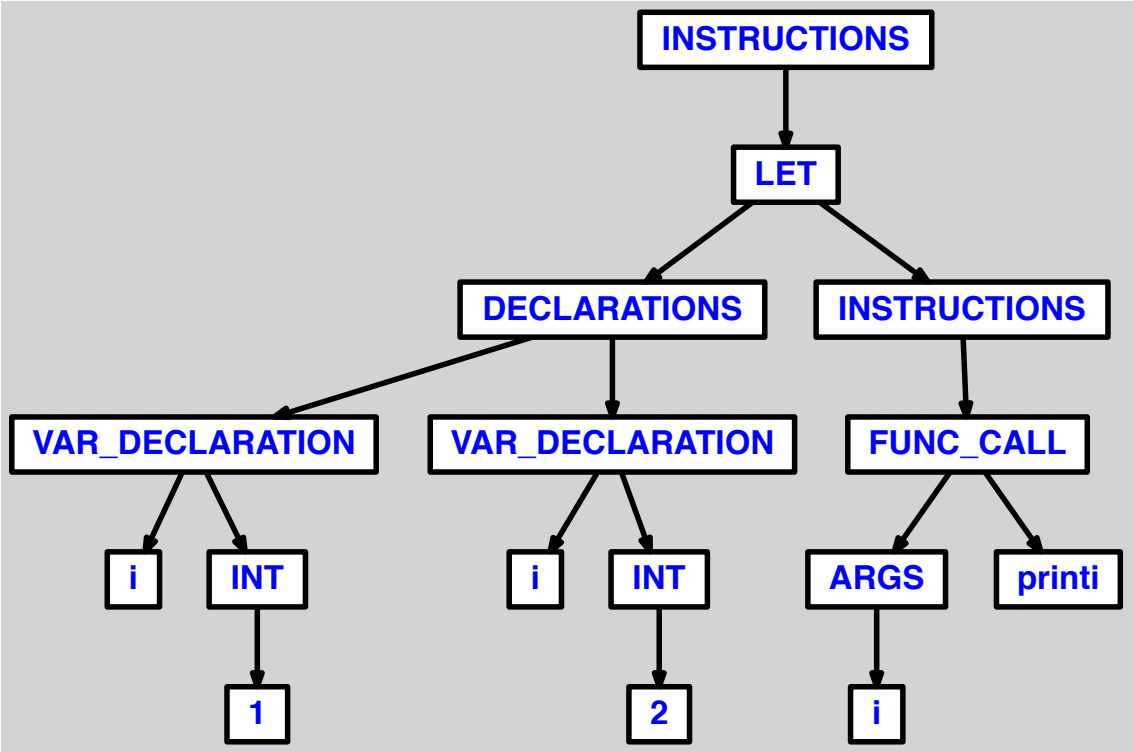
13.1.7 double-declaration avec valeurs egales

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
i
INT
i
INT
ARGS
printi
1
1
i

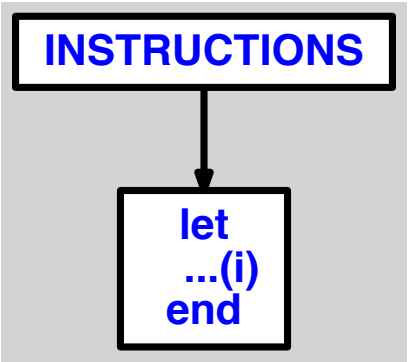
13.1.8 double-declaration avec valeurs differentes

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
i
INT
i
INT
ARGS
printi
1
2
i

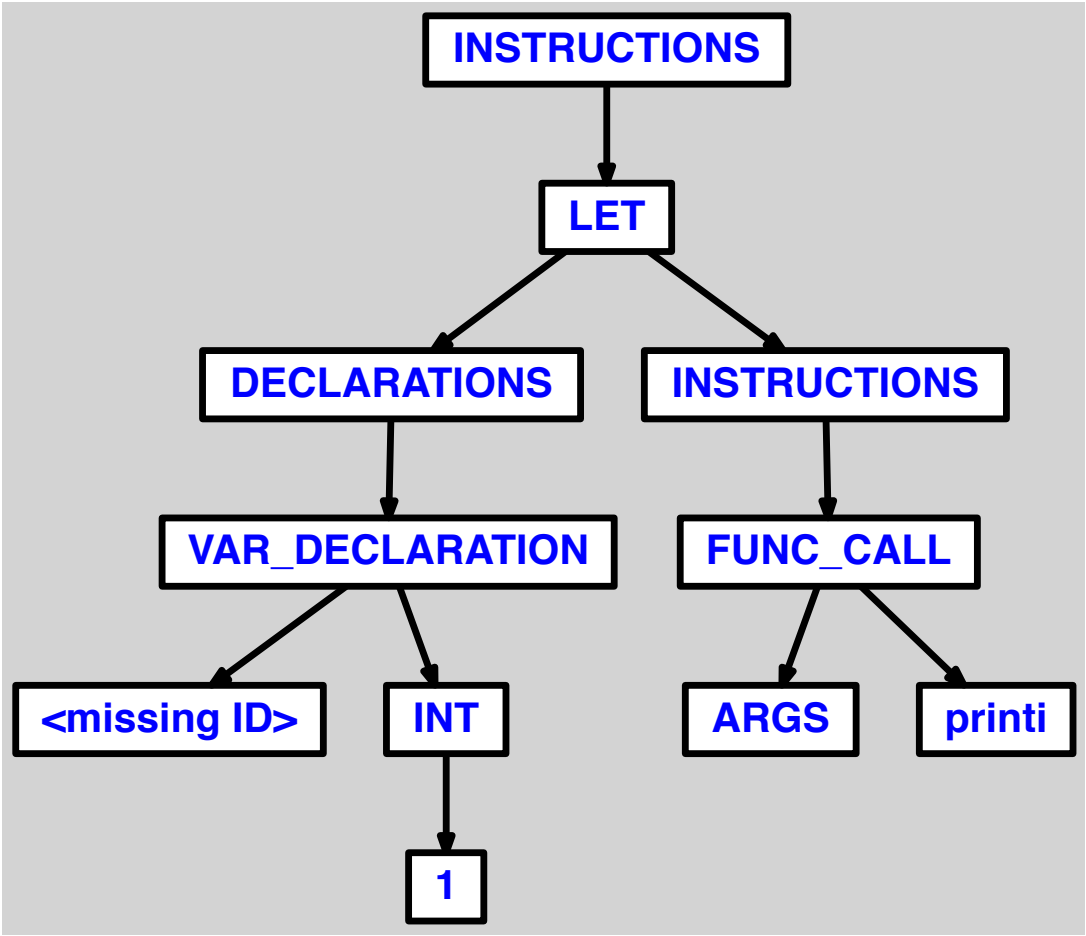
13.1.9 <var> mal ecrit

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


INSTRUCTIONS
let
...(i)
end

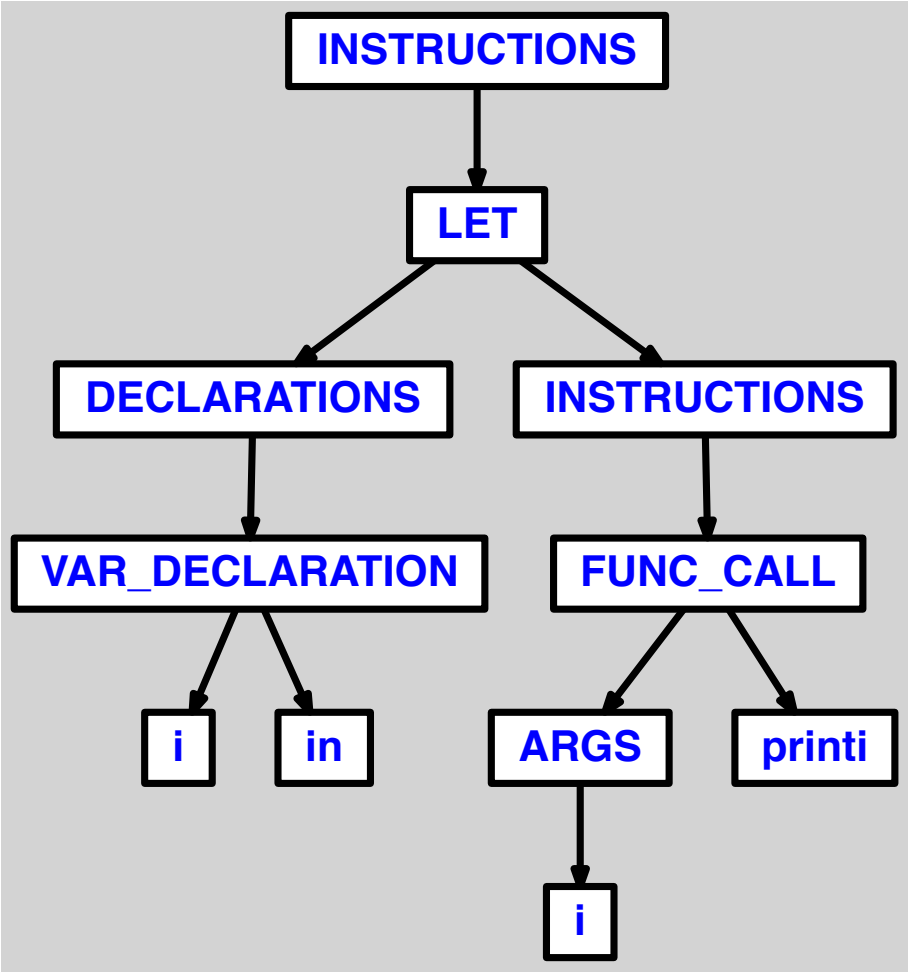
13.1.10 declaration de caractere special

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
<missing ID>
INT
ARGS
printi
1

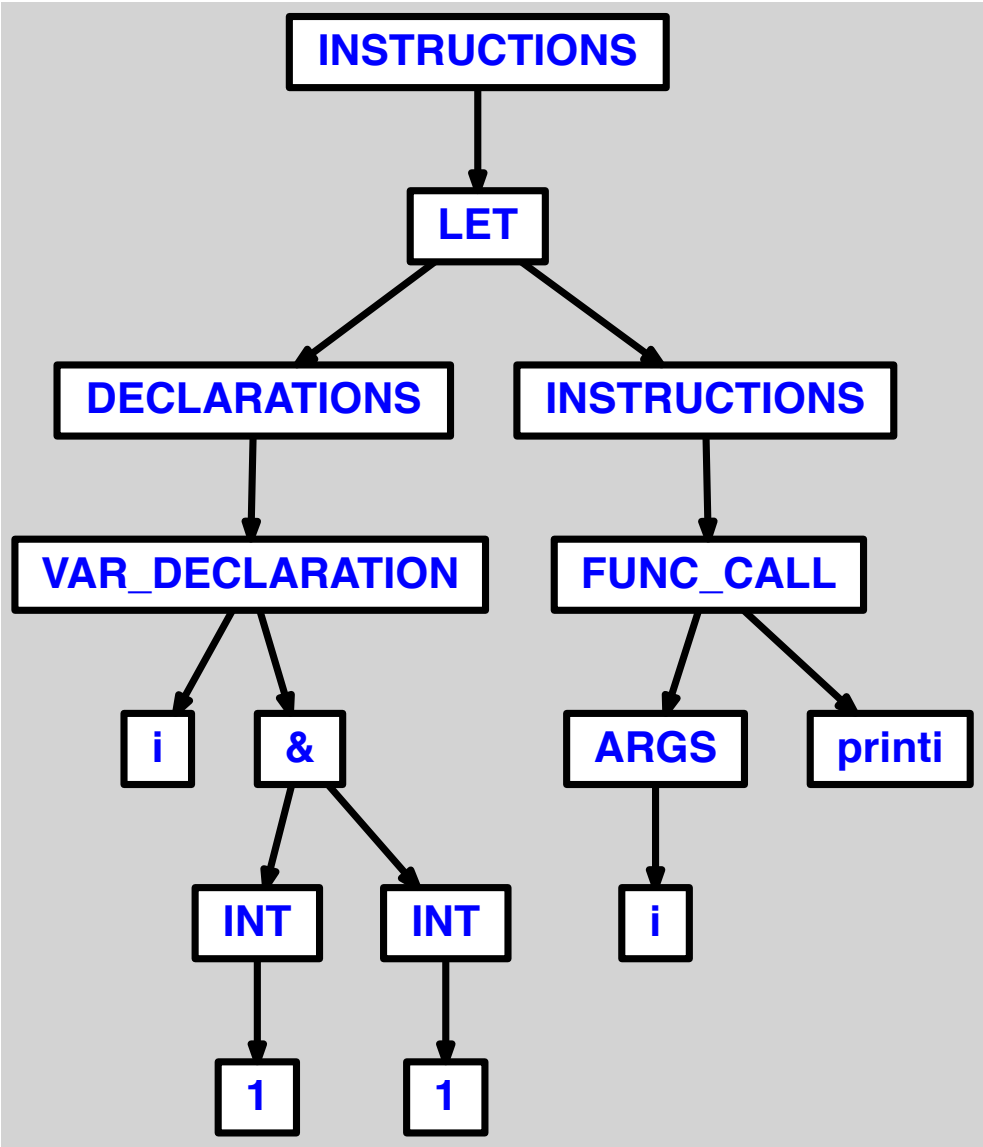
13.1.11 affectation de caractere special

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
i
in
ARGS
printi
i

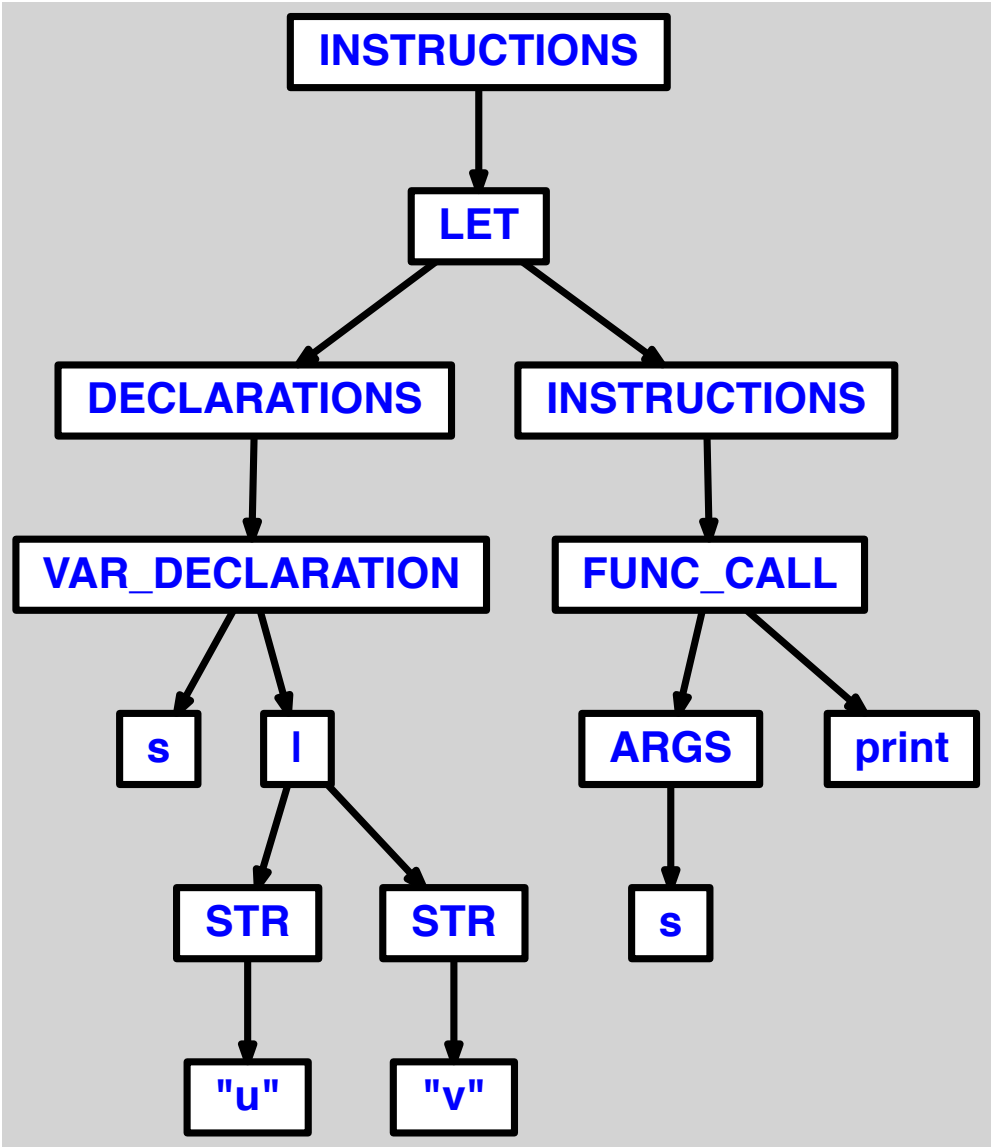
13.1.12 affectation d'operation logique sur entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
i
&
ARGS
printi
INT
INT
i
1
1

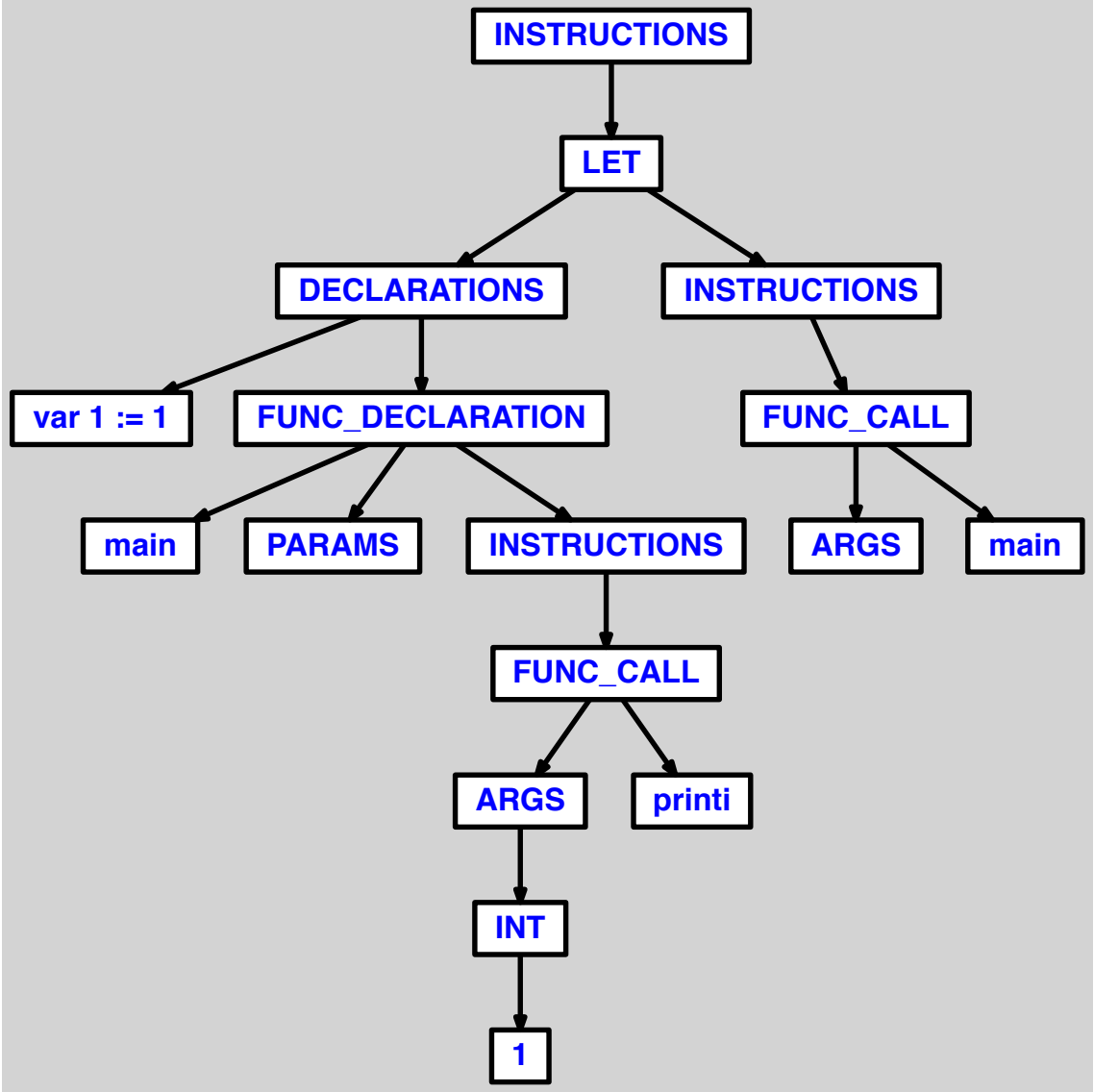
13.1.13 affectation d'operation logique sur chaines

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
s
l
ARGS
print
STR
STR
s
"u"
"v"

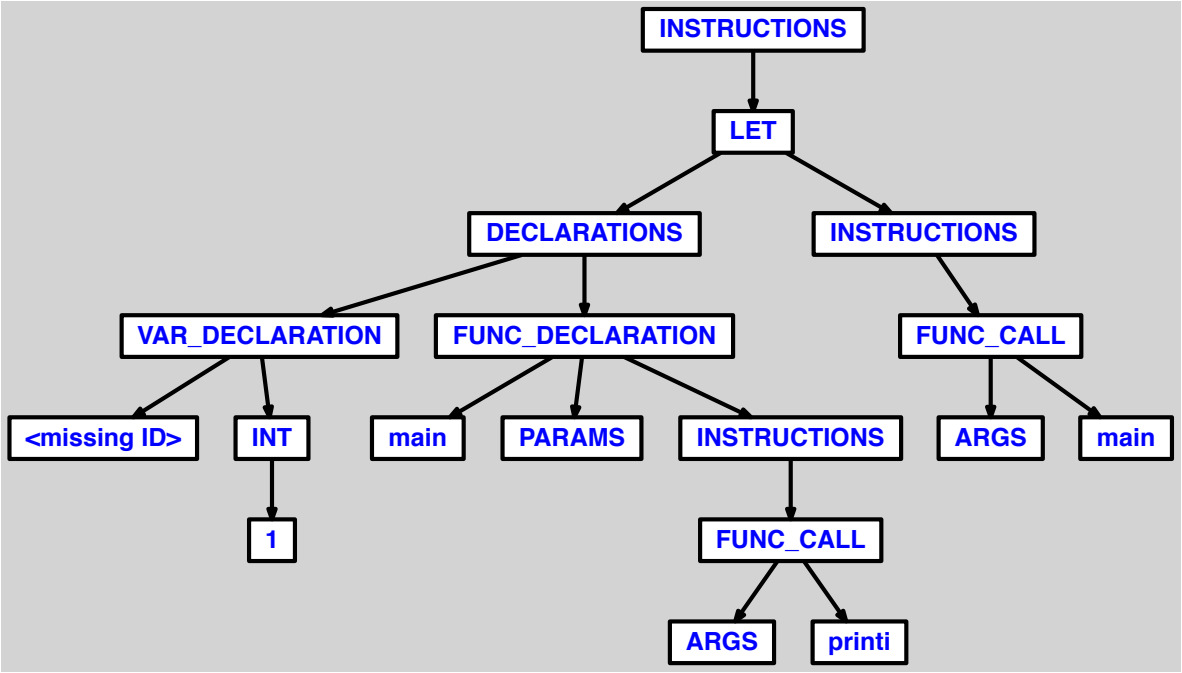
13.1.14 chiffre comme nom de variable

```
let
  var 1 := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
var 1 := 1
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
FUNC_CALL
ARGS
printi
INT
1

13.1.15 <_> comme nom de variable

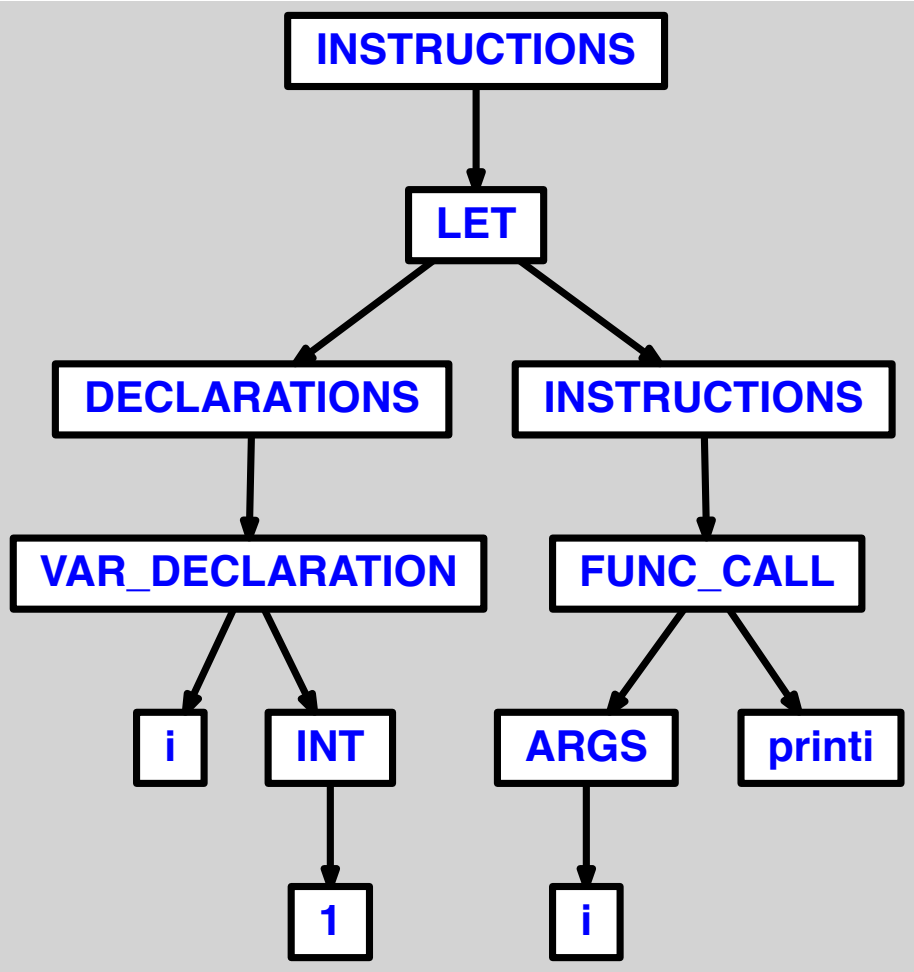
```
let
  var _ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
<missing ID>
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi

13.2 OK

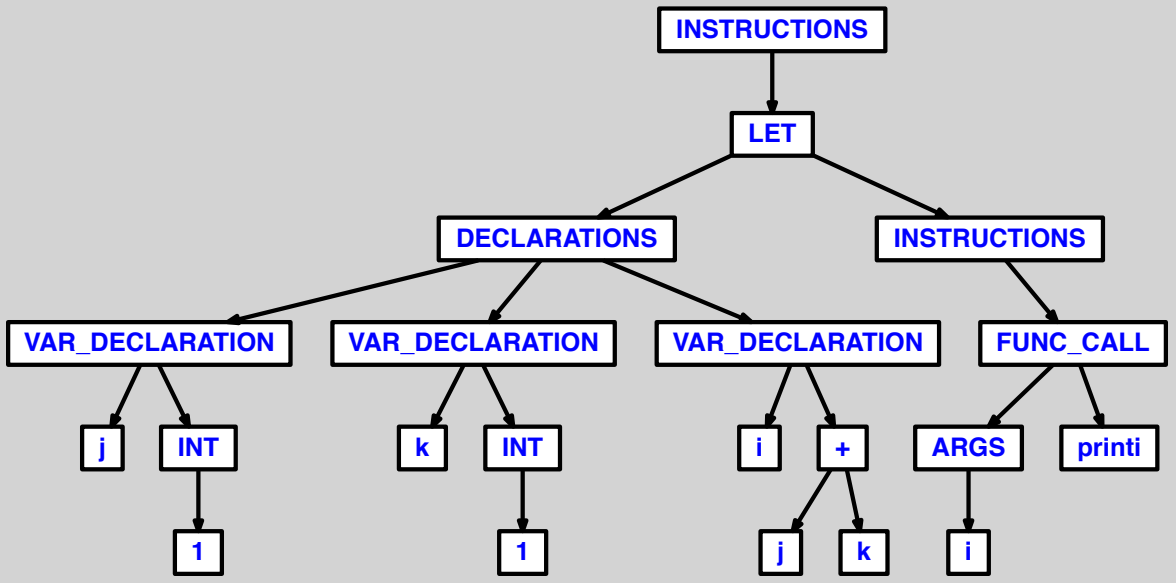
13.2.1 declaration d'entier simple

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
i
INT
ARGS
printi
1
i

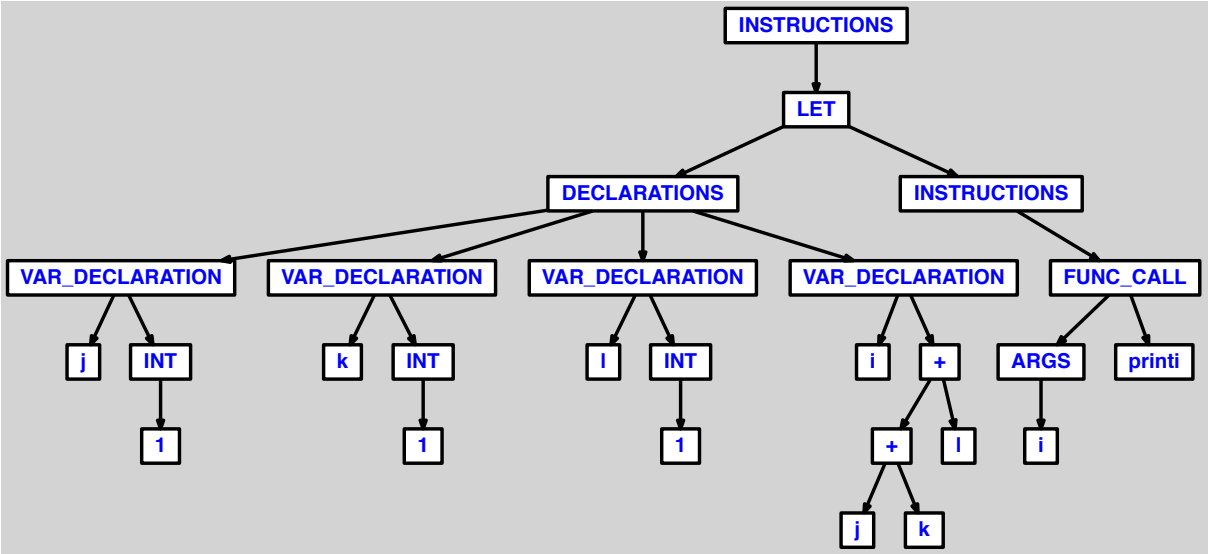
13.2.2 declarations d'entier avec reutilisation de 2 autres entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
j
INT
k
INT
i
+
ARGS
printi
1
1
j
k
i

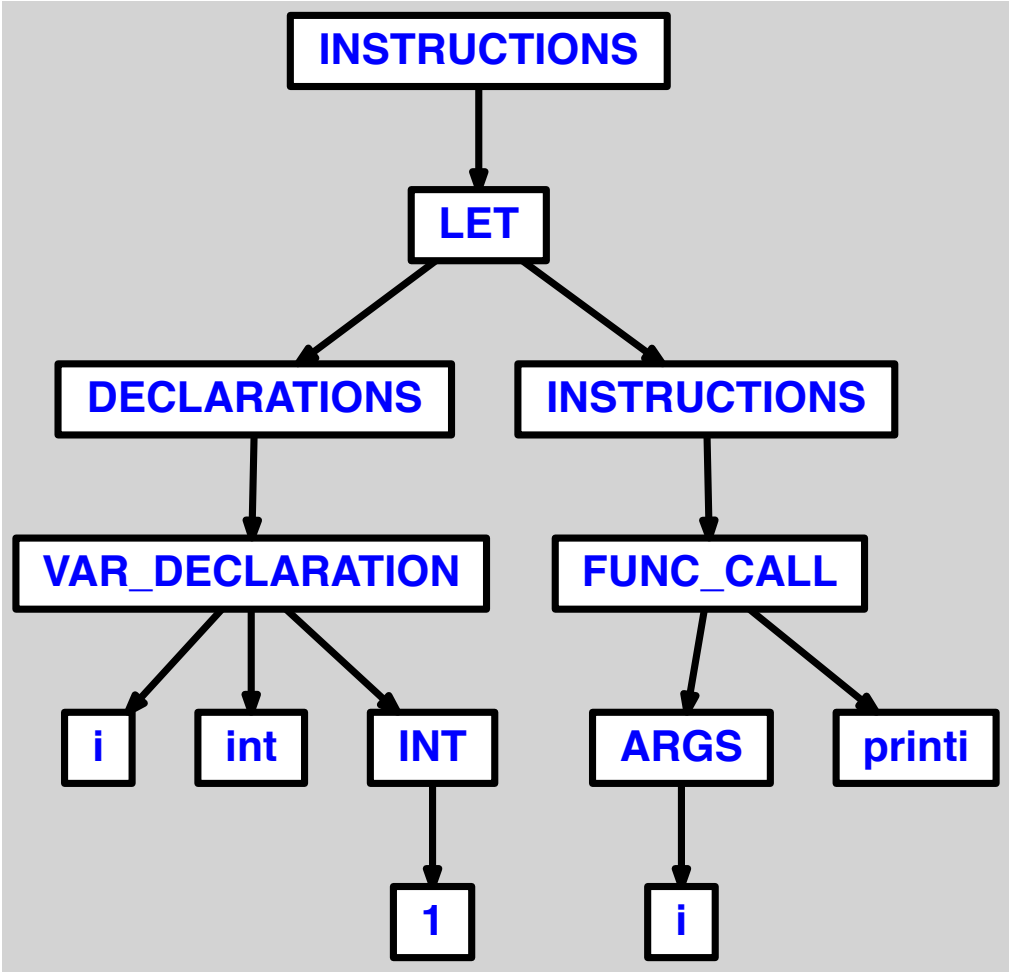
13.2.3 declarations d'entier avec reutilisation de 3 autres entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
j
INT
k
INT
l
INT
i
+
ARGS
printi
1
1
1
+
l
i
j
k

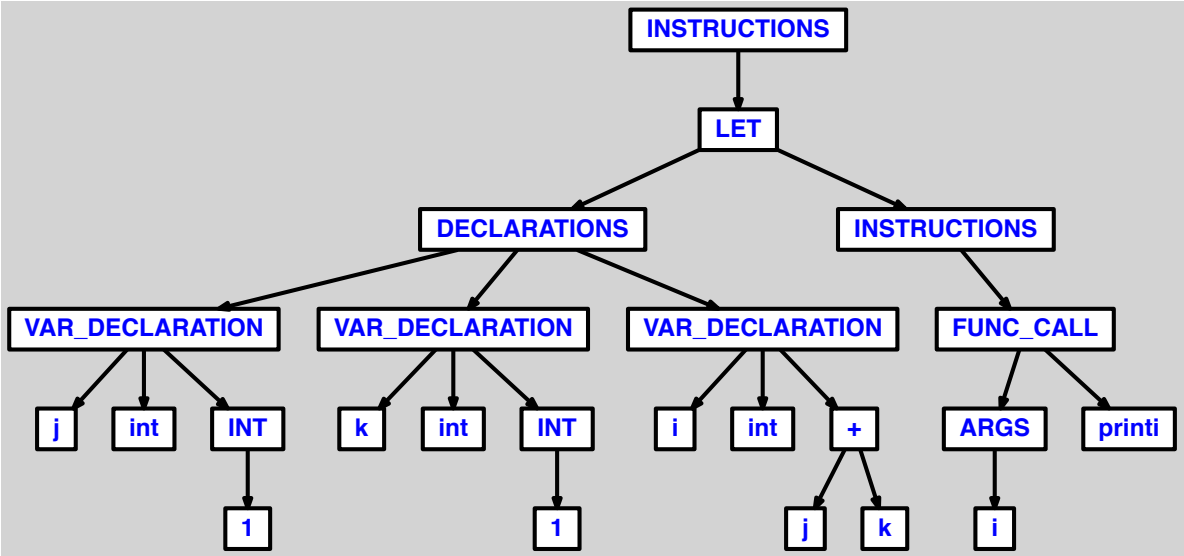
13.2.4 declaration simple d'entier avec type

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
i
int
INT
ARGS
printi
1
i

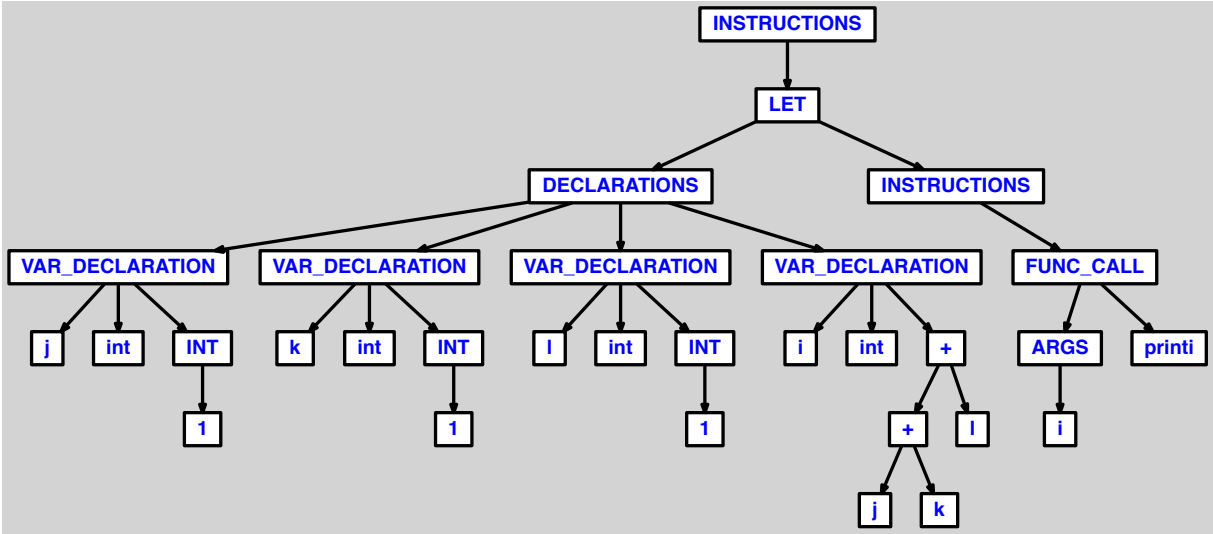
13.2.5 declarations d'entier avec types et reutilisation de 2 autres entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
j
int
INT
k
int
INT
i
int
+
ARGS
printi
1
1
j
k
i

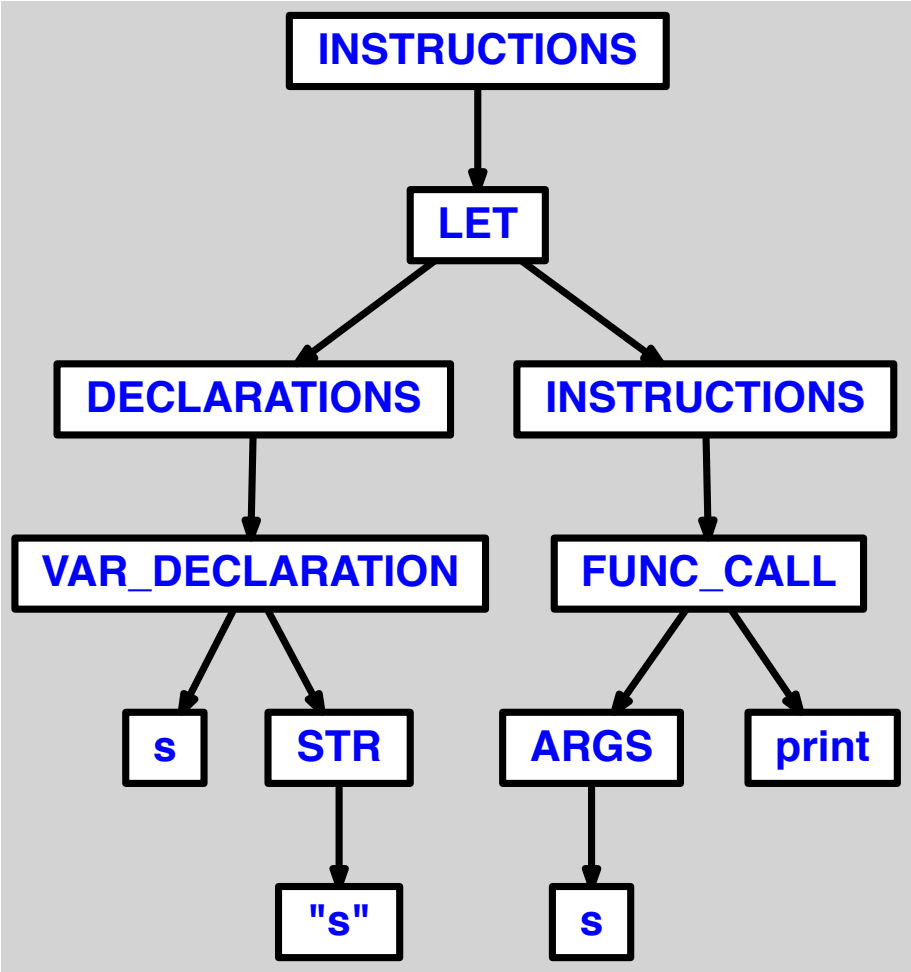
13.2.6 declarations d'entier avec types et reutilisation de 3 autres entiers

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
j
int
INT
k
int
INT
l
int
INT
i
int
+
ARGS
printi
1
1
1
+
l
i
j
k

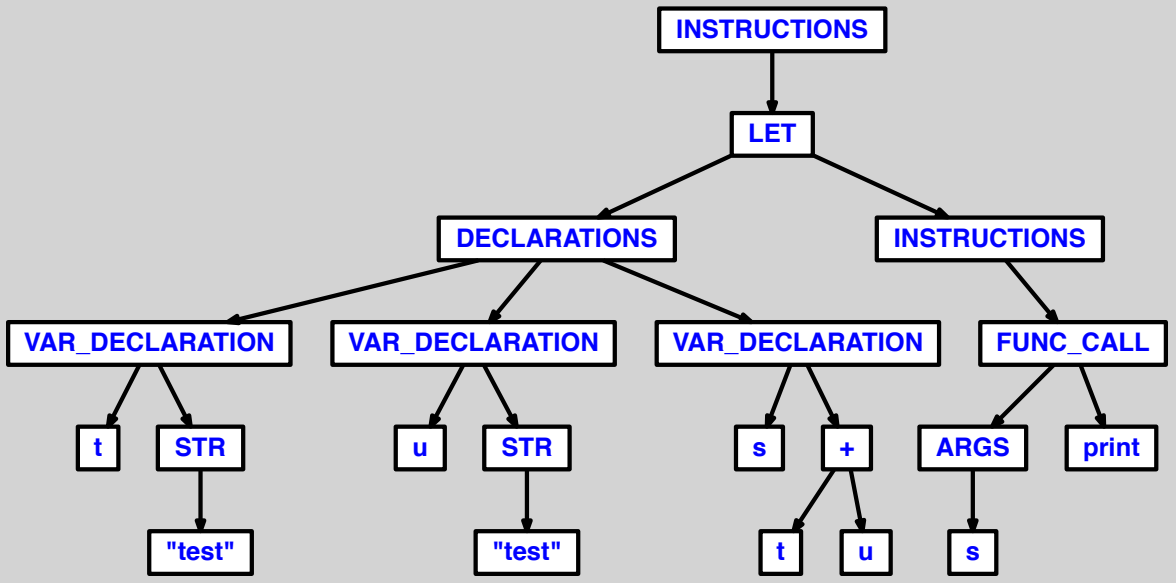
13.2.7 declaration de chaine simple

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
s
STR
ARGS
print
"s"
s

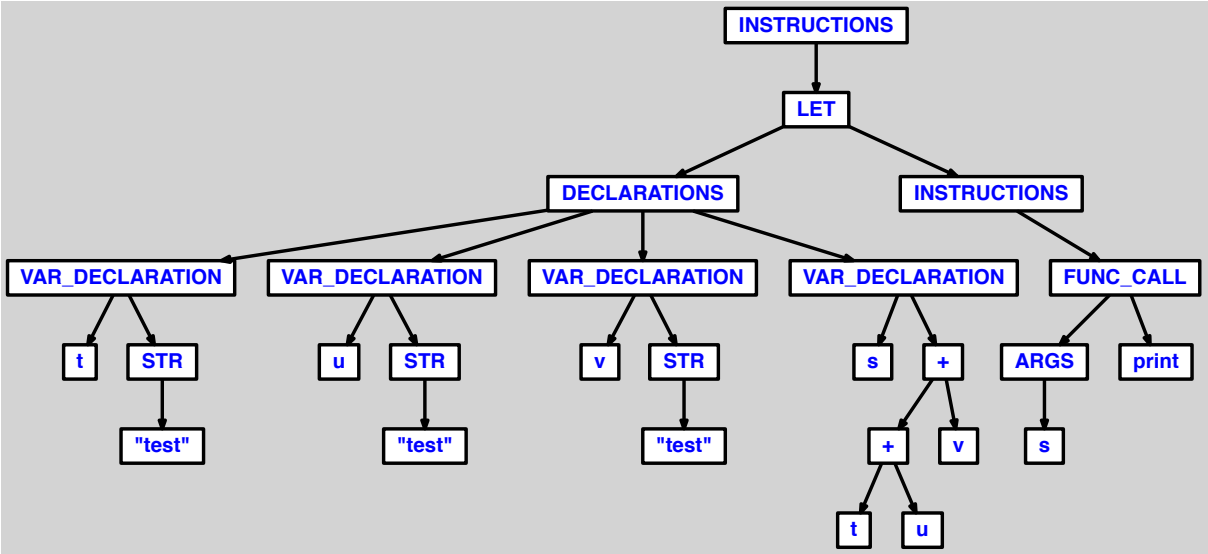
13.2.8 declarations de chaine avec reutilisation de 2 autres chaines

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
t
STR
u
STR
s
+
ARGS
print
"test"
"test"
t
u
s

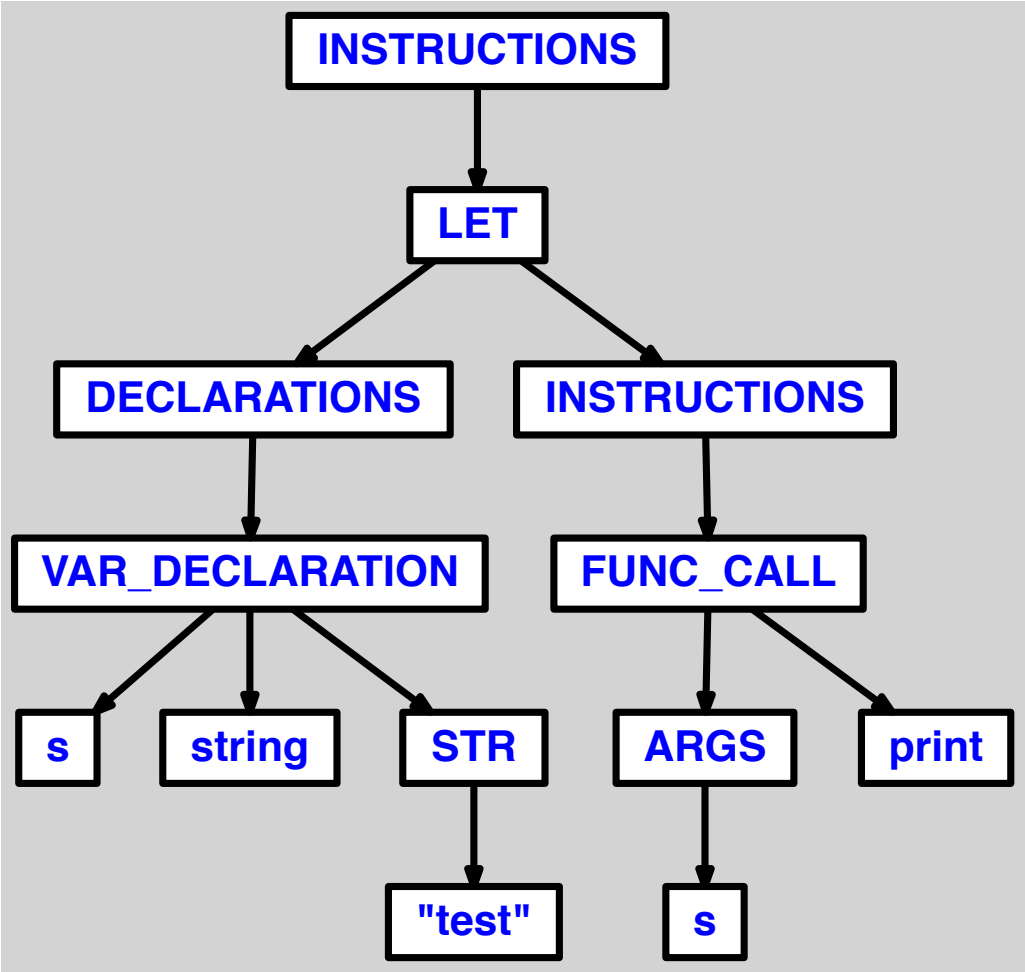
13.2.9 declarations de chaine avec reutilisation de 3 autres chaines

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
t
STR
u
STR
v
STR
s
+
ARGS
print
"test"
"test"
"test"
+
v
s
t
u

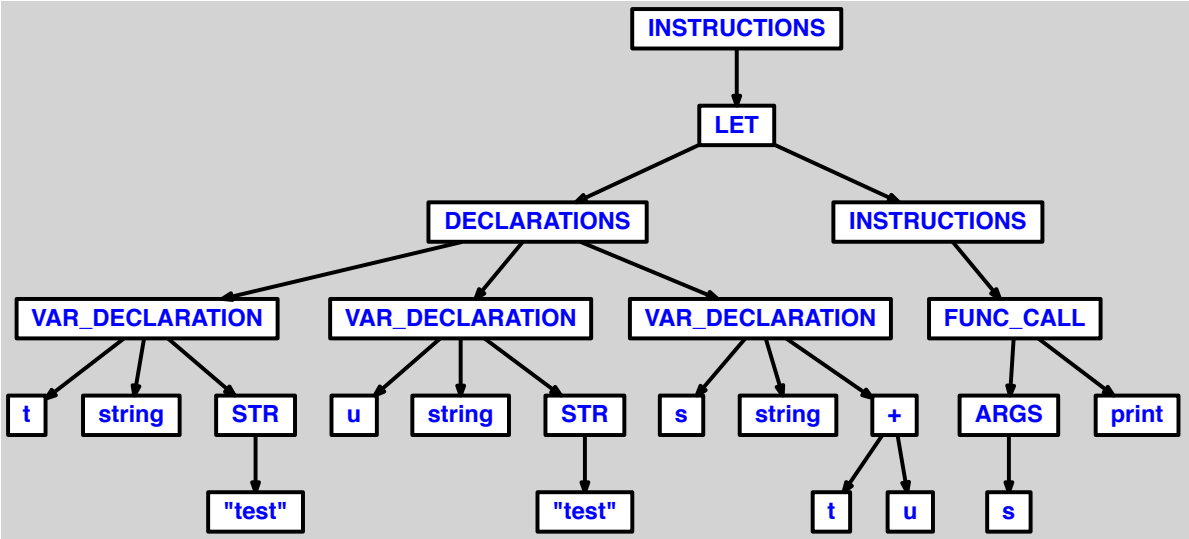
13.2.10 declaration de chaine avec type

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_CALL
s
string
STR
ARGS
print
"test"
s

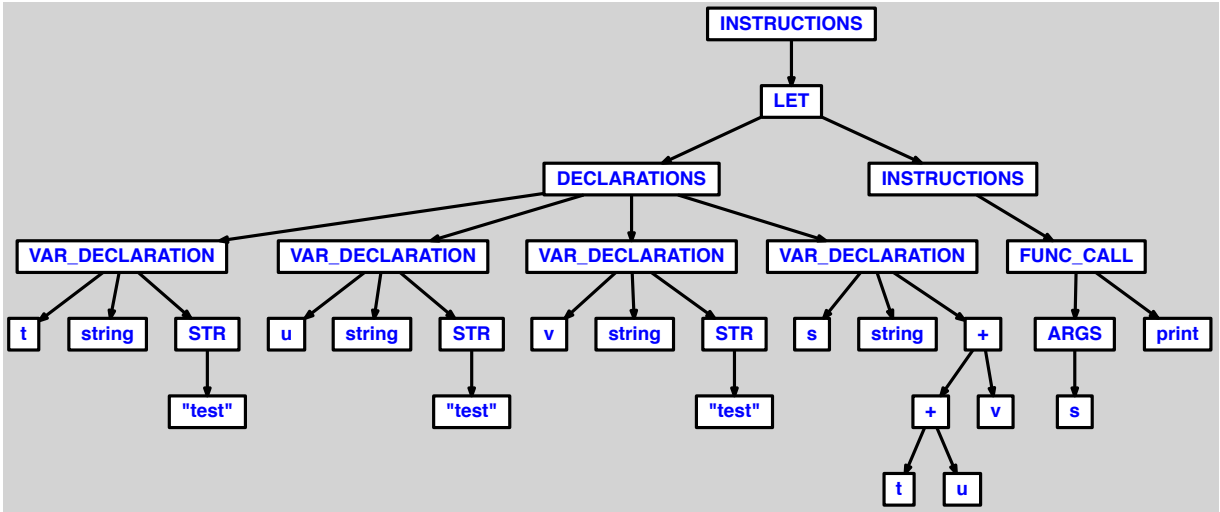
13.2.11 declarations de chaine avec types et reutilisation de 2 autres chaines

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
t
string
STR
u
string
STR
s
string
+
ARGS
print
"test"
"test"
t
u
s

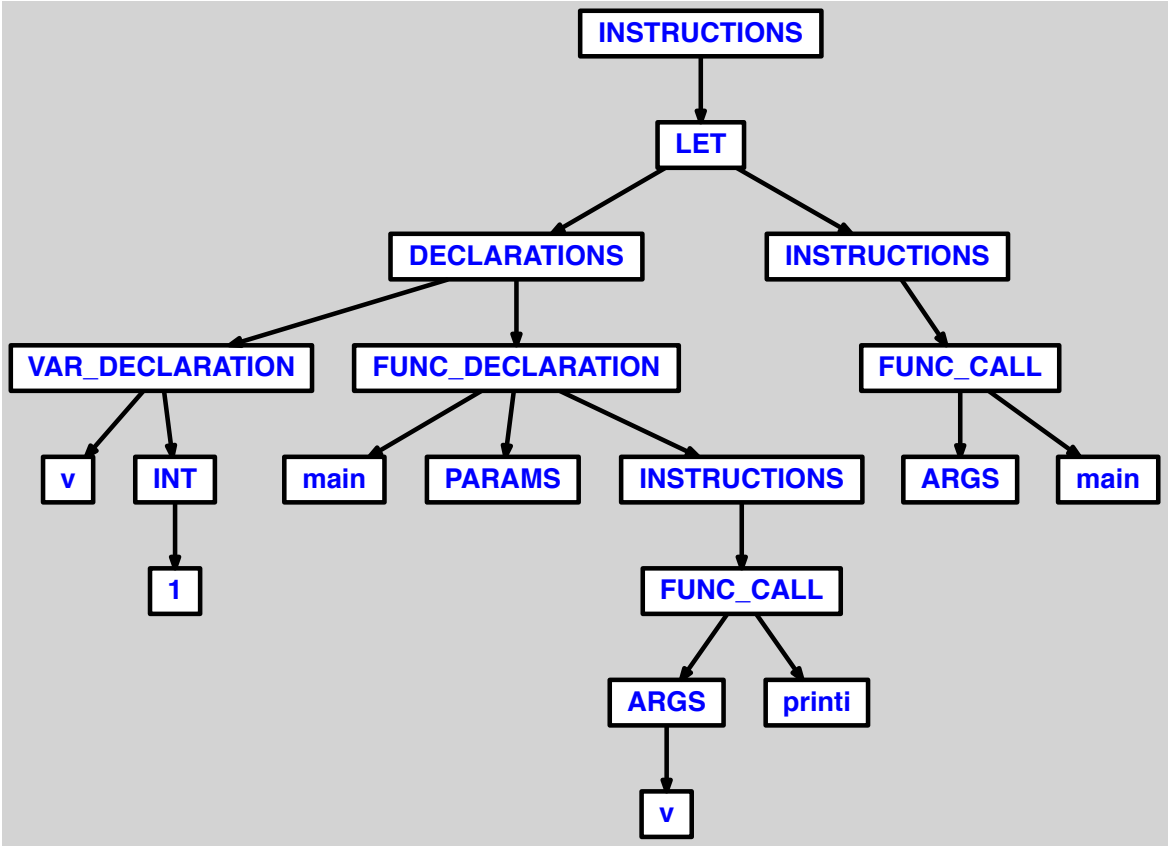
13.2.12 declarations de chaine avec types et reutilisation de 3 autres chaines

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_CALL
t
string
STR
u
string
STR
v
string
STR
s
string
+
ARGS
print
"test"
"test"
"test"
+
v
s
t
u

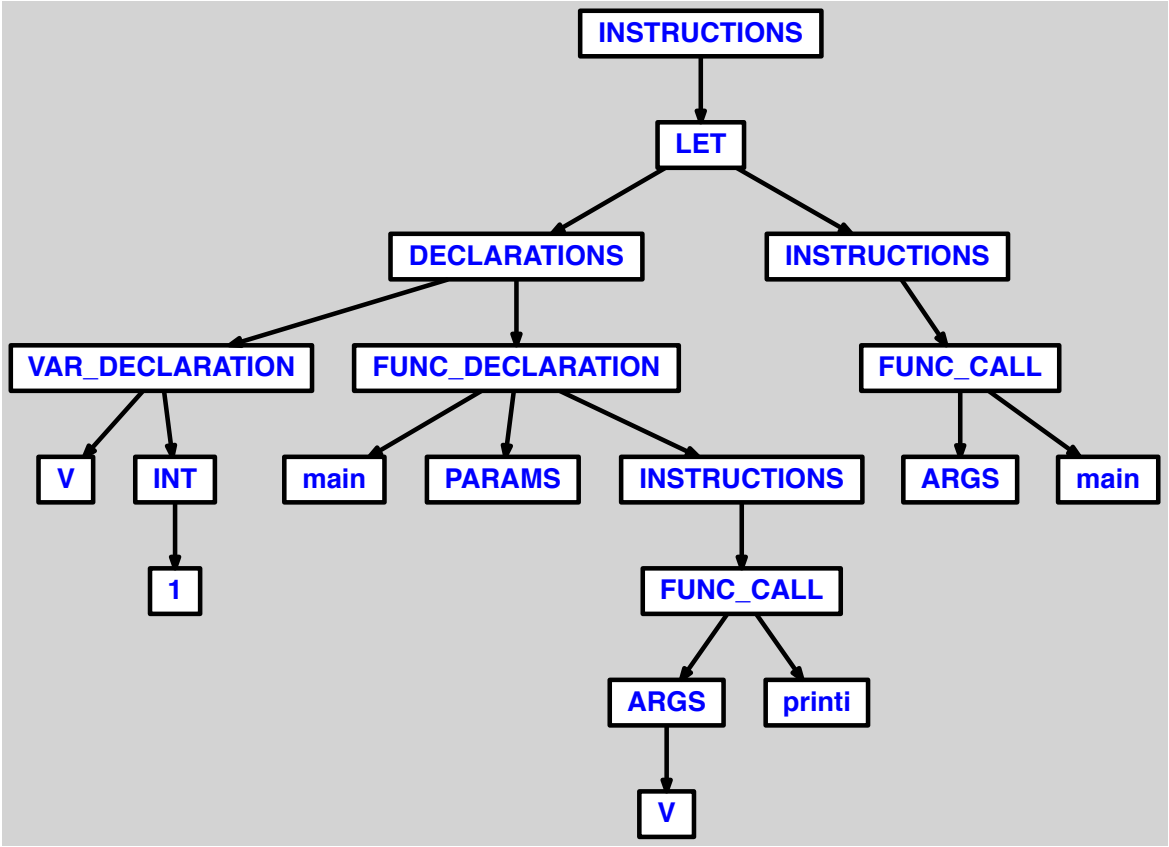
13.2.13 lettre minuscule comme nom de variable

```
let
  var v := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
v

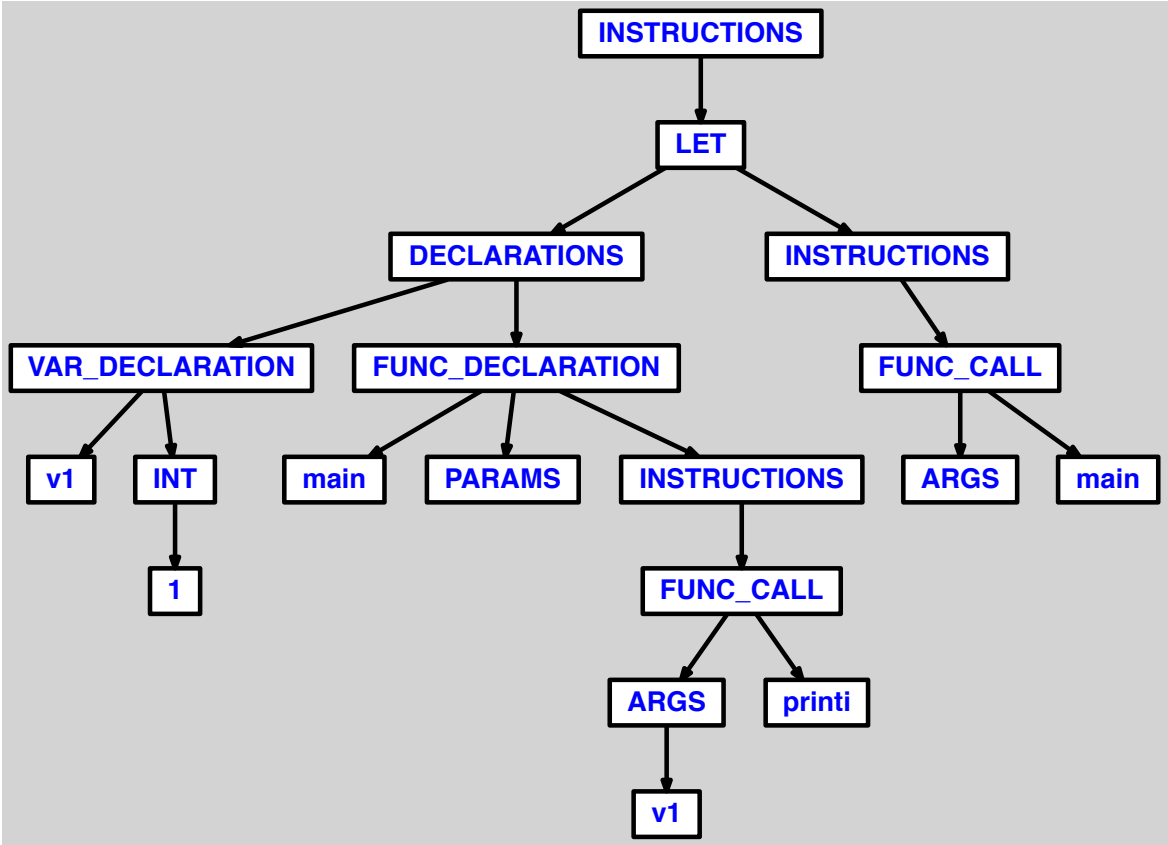
13.2.14 lettre majuscule comme nom de variable

```
let
  var V := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
V
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
V

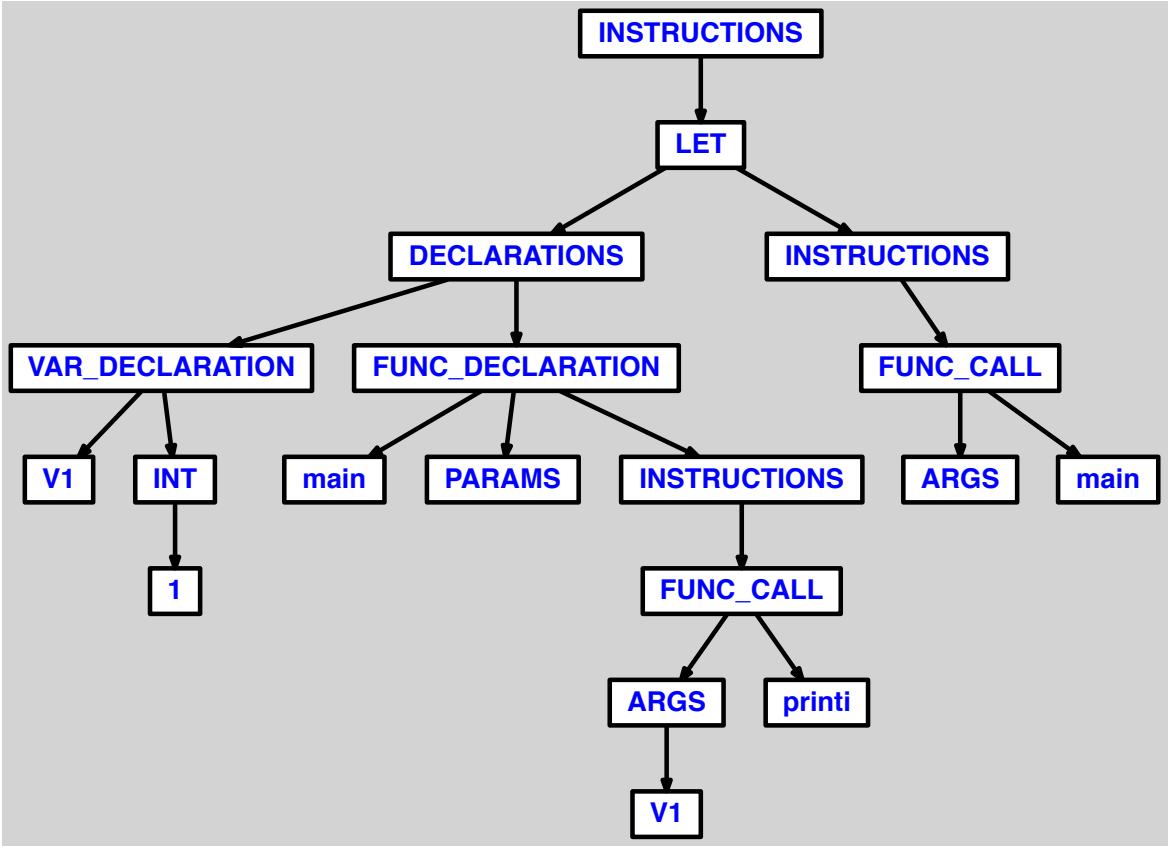
13.2.15 lettre minuscule et chiffre dans nom de variable

```
let
  var v1 := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v1
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
v1

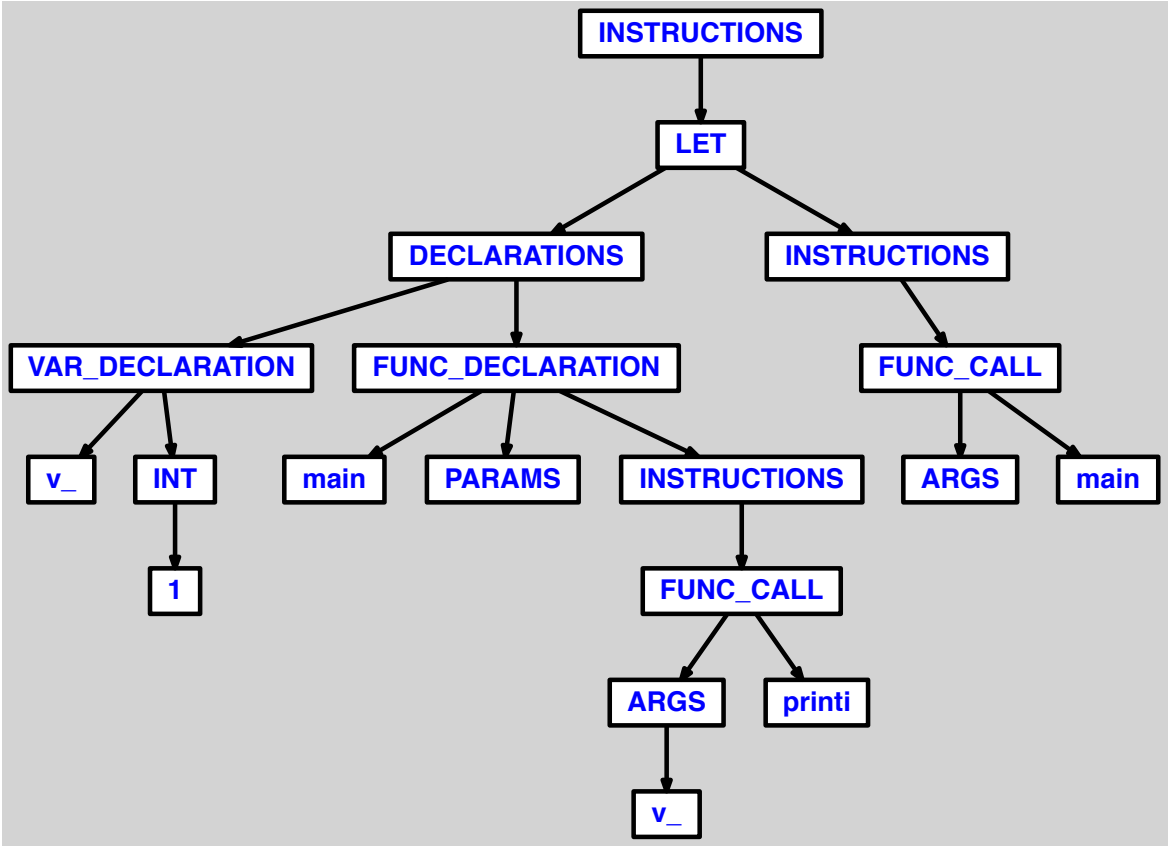
13.2.16 lettre majuscule et chiffre dans nom de variable

```
let
  var V1 := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
V1
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
V1

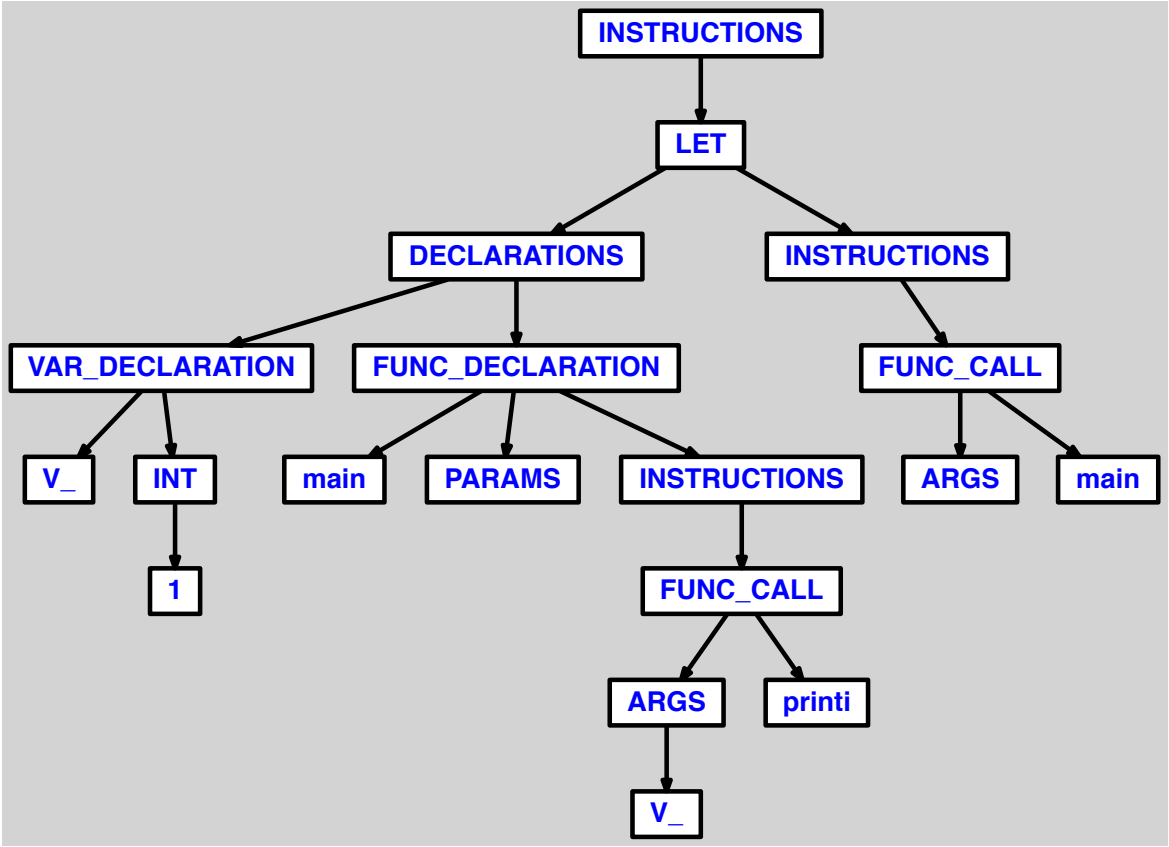
13.2.17 lettre minuscule et <_> dans nom de variable

```
let
  var v_ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v_
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
v_

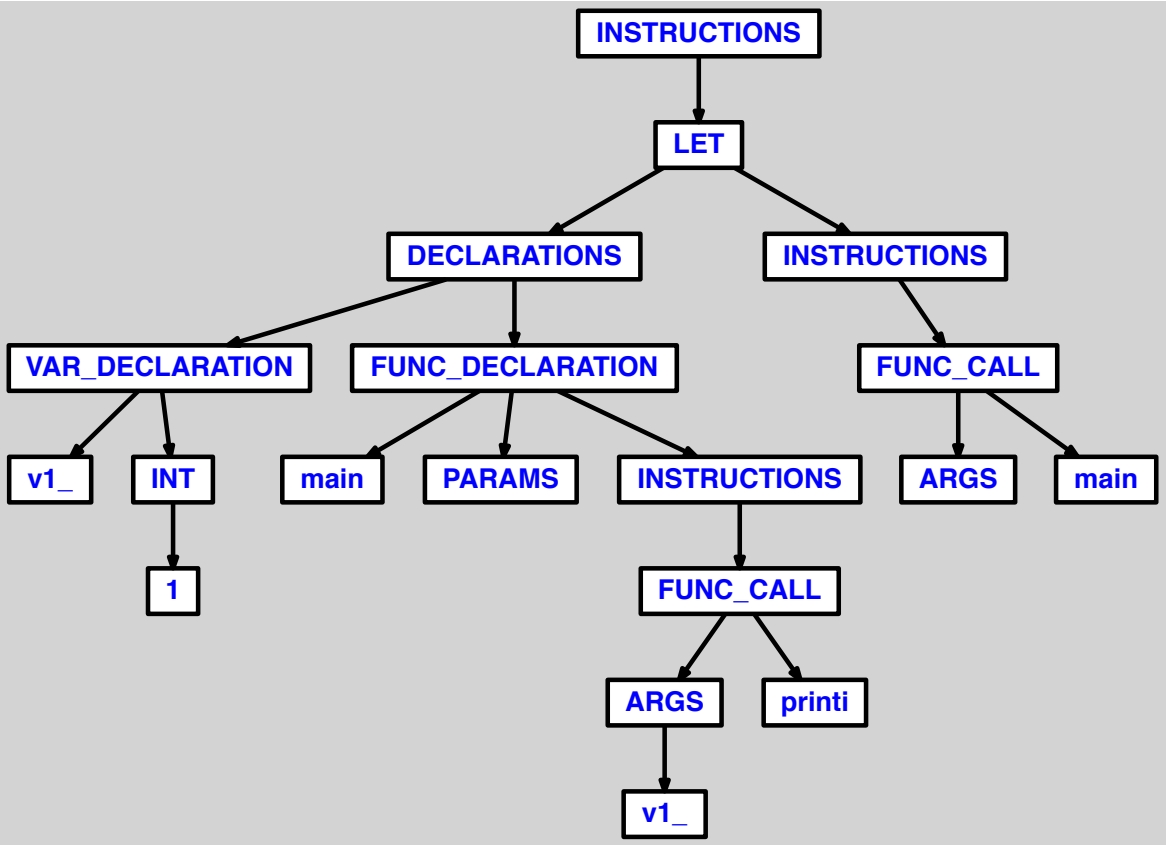
13.2.18 lettre majuscule et <_> dans nom de variable

```
let
  var V_ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
V_
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
V_

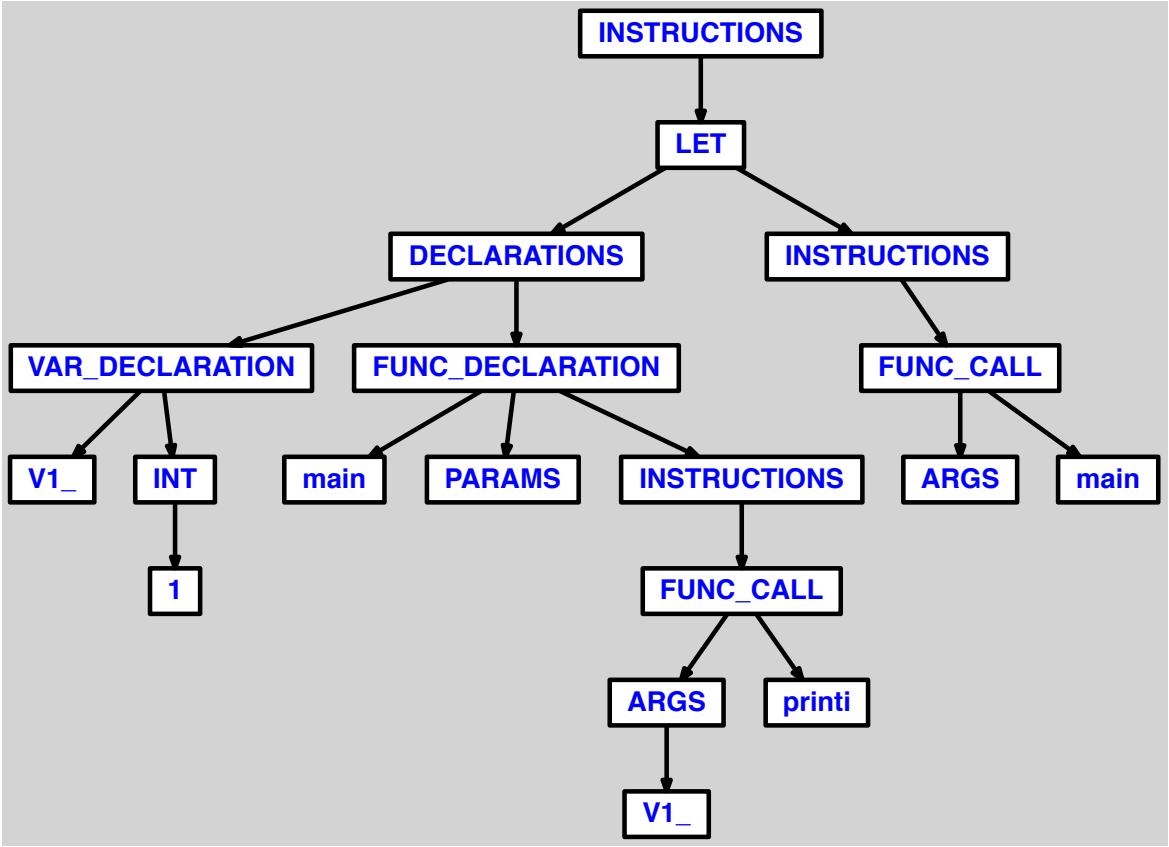
13.2.19 lettre minuscule, chiffre et <_> dans nom de variable

```
let
  var v1_ := 1

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
v1_
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
v1_

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

```
let
  var V1_ := 1

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

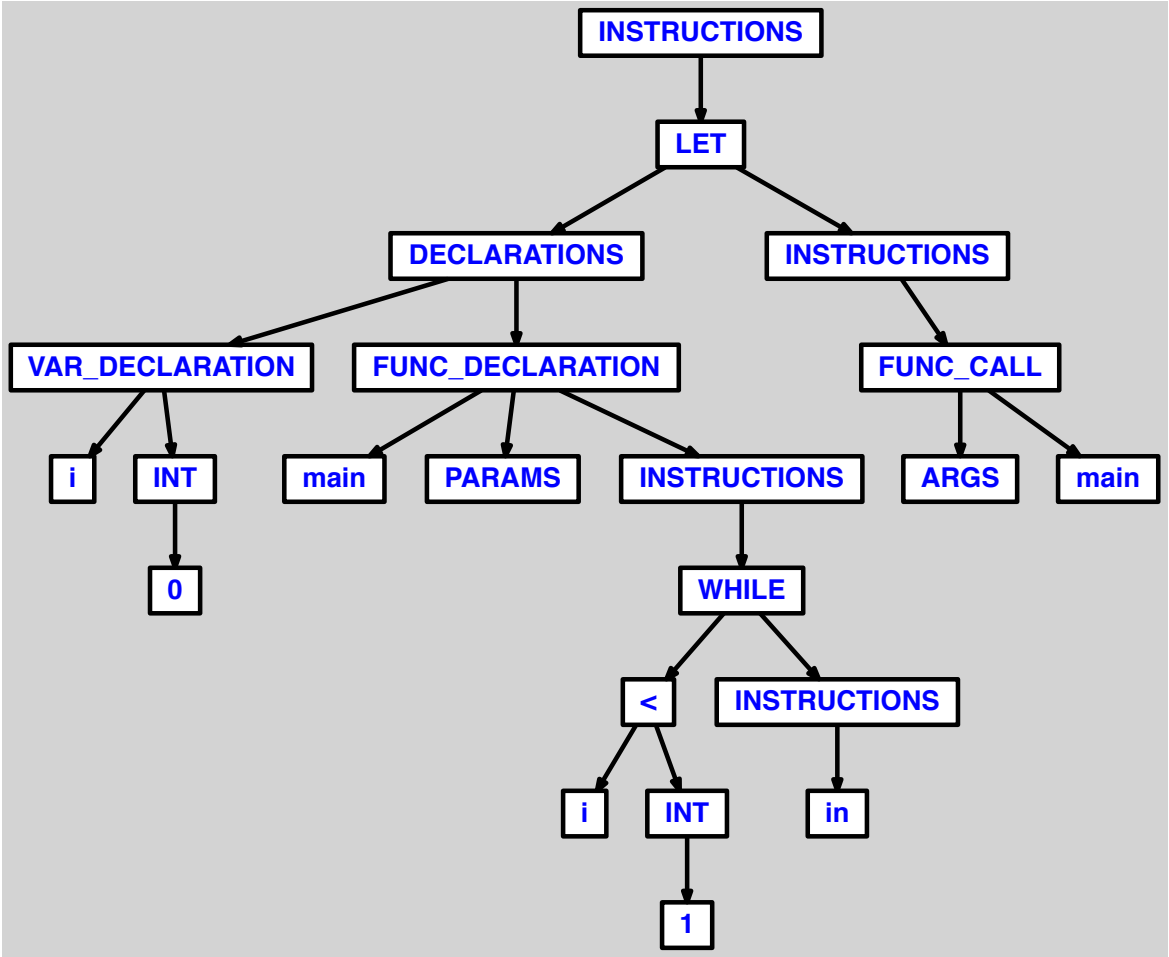

INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
V1_
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
1
FUNC_CALL
ARGS
printi
V1_

14 while

14.1 KO

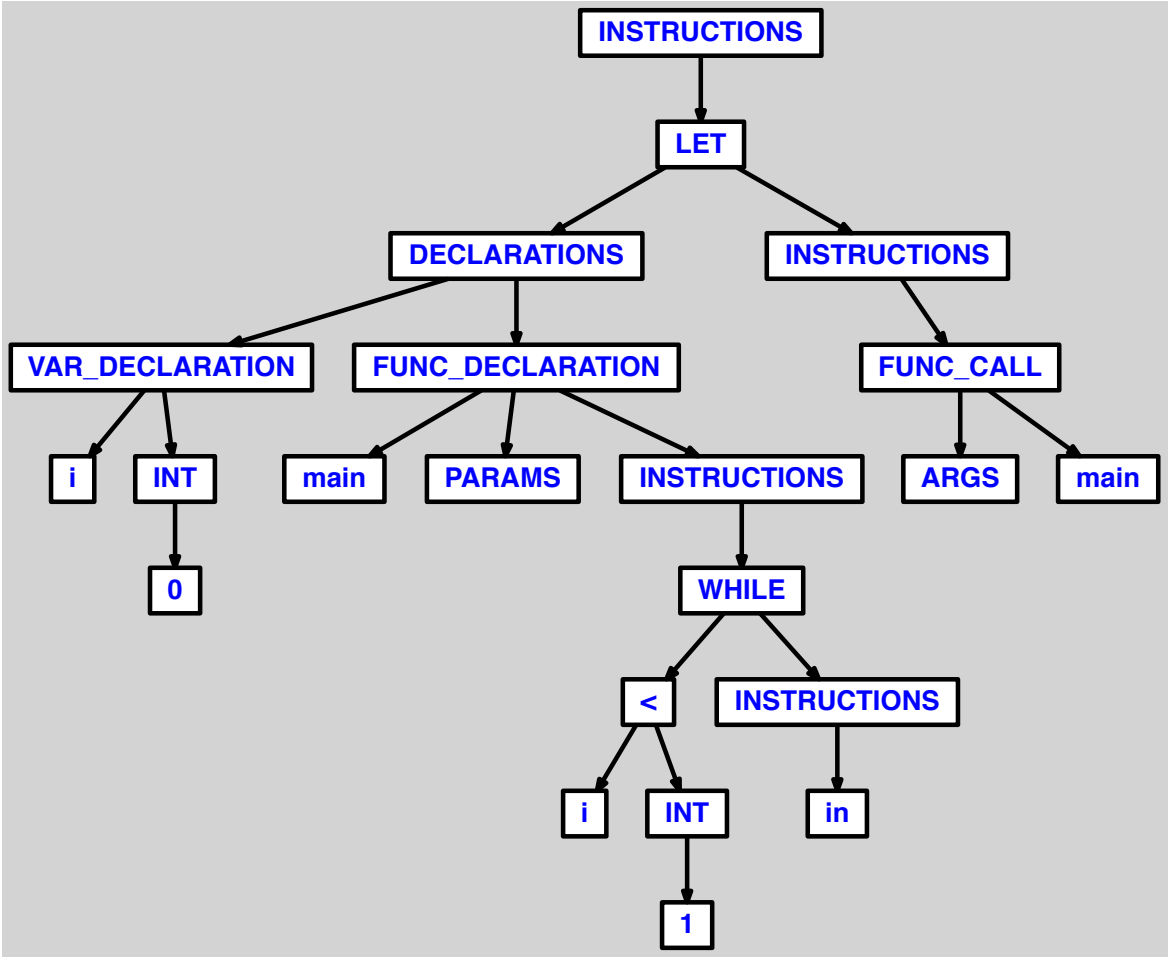
14.1.1 <while> sans instruction

```
let
  var i := 0

  function main() =
    while i < 1 do
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
WHILE
<
INSTRUCTIONS
i
INT
in
1

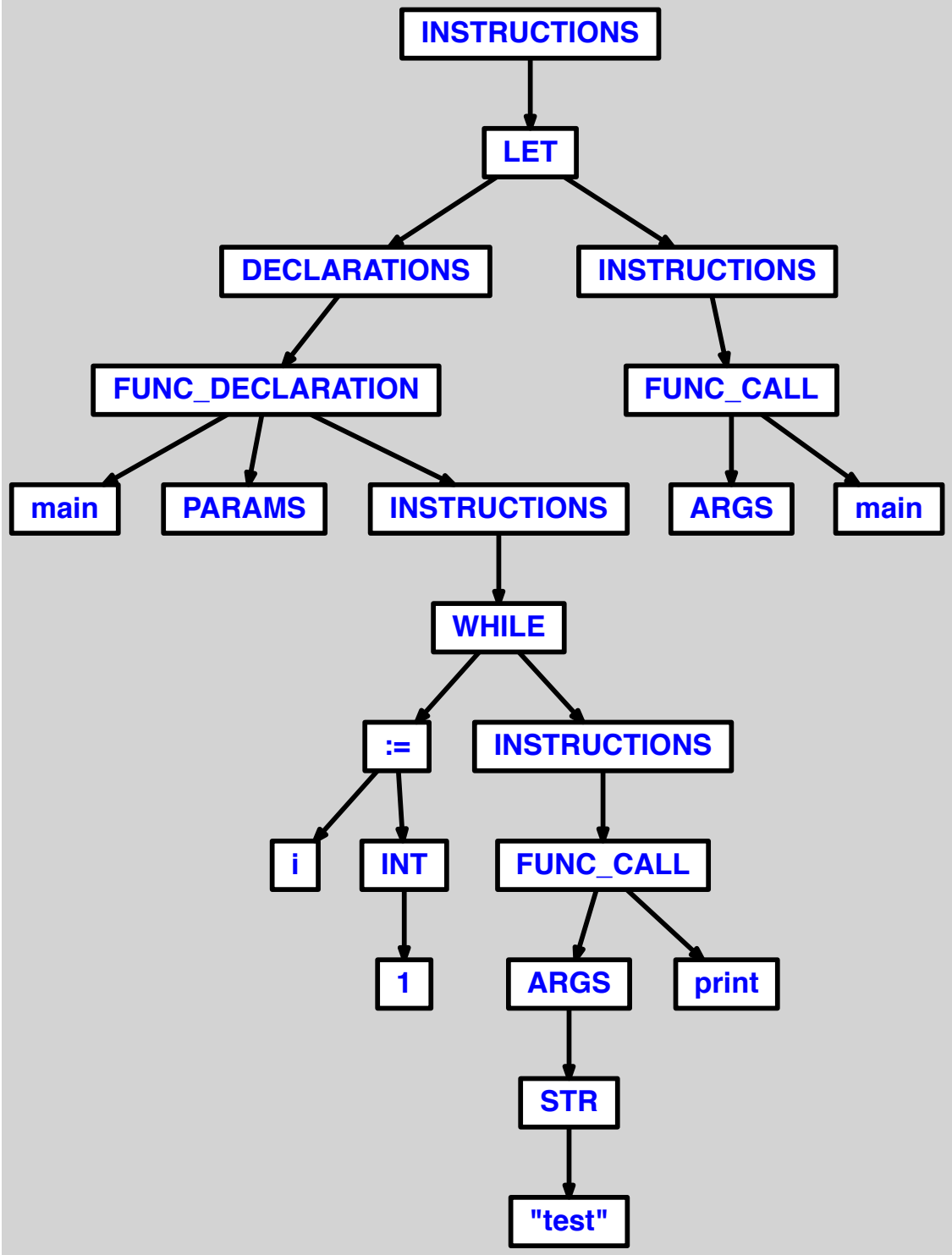
14.1.2 <while> avec ligne vide

```
let
  var i := 0

  function main() =
    while i < 1 do

in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
WHILE
<
INSTRUCTIONS
i
INT
in
1

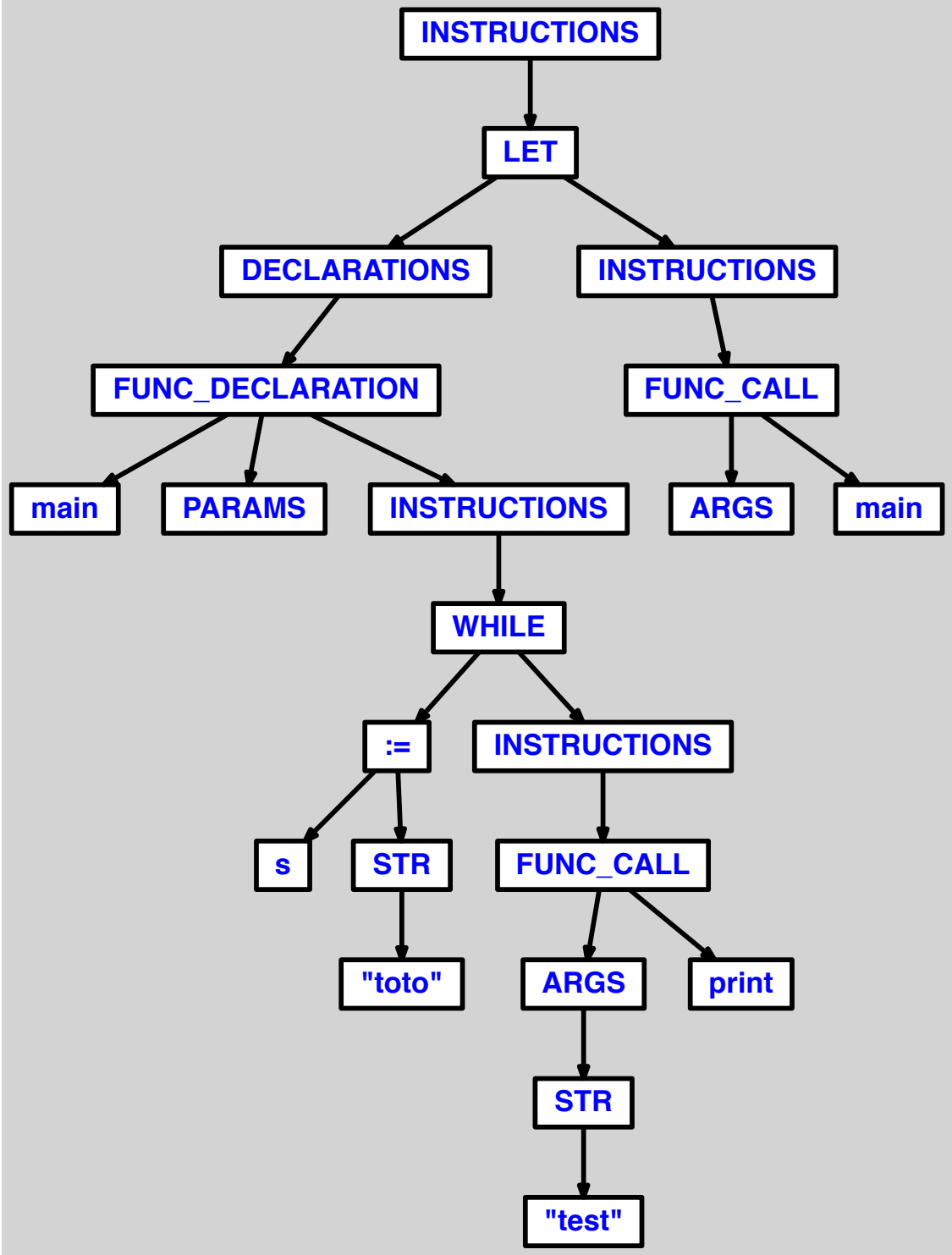
14.1.3 <while> avec affectation d'entier

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
:=
INSTRUCTIONS
i
INT
FUNC_CALL
1
ARGS
print
STR
"test"

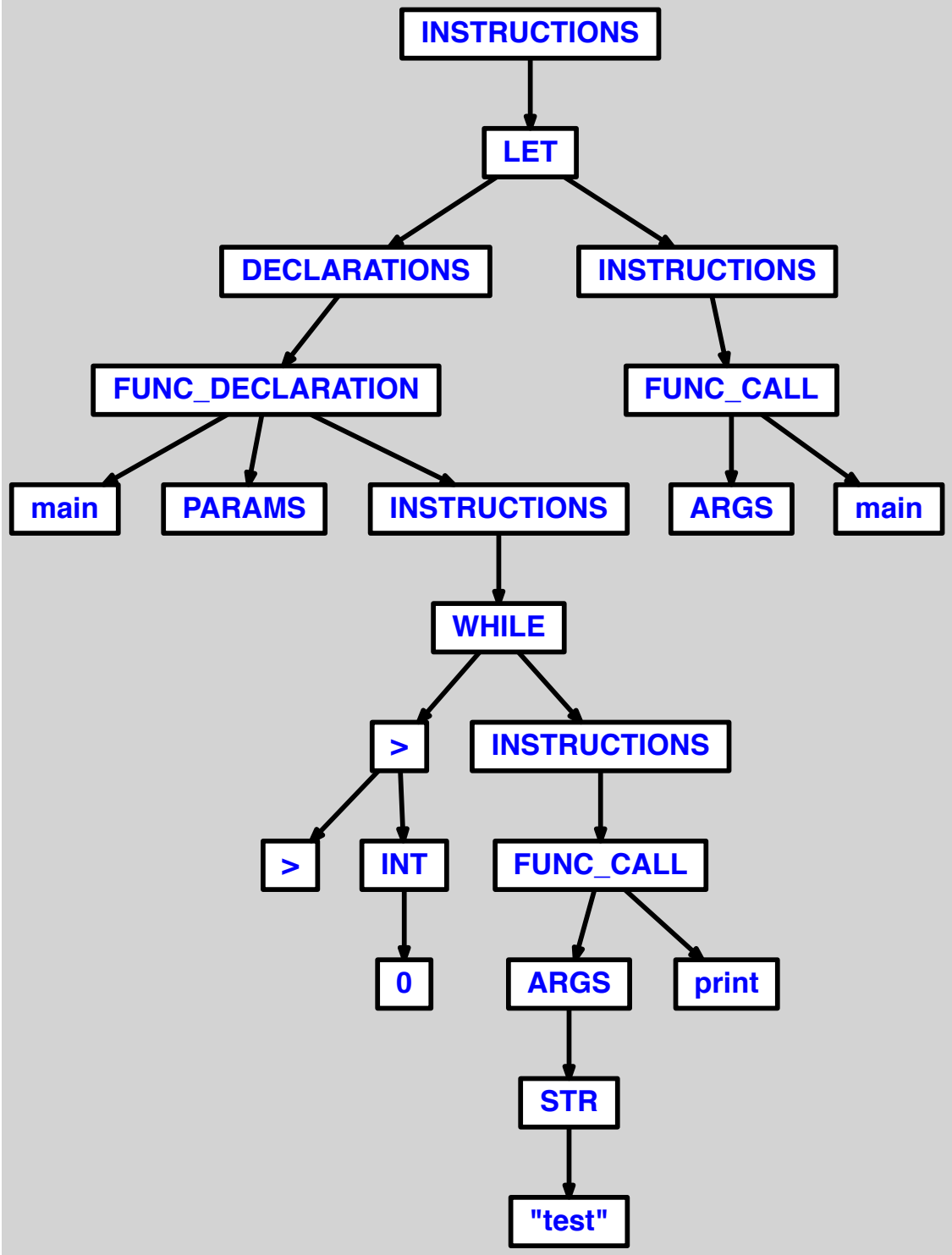
14.1.4 <while> avec affectation de chaine

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
:=
INSTRUCTIONS
s
STR
FUNC_CALL
"toto"
ARGS
print
STR
"test"

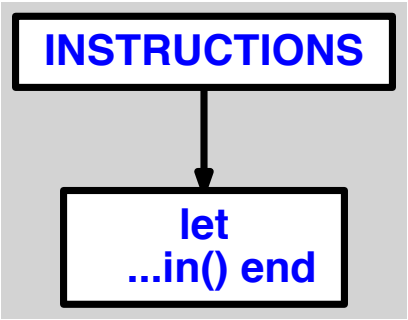
14.1.5 <while> avec oubli du compteur

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
>
INSTRUCTIONS
>
INT
FUNC_CALL
0
ARGS
print
STR
"test"

14.1.6 <while> avec oubli du <while>

```
let
  var i := 0

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


INSTRUCTIONS
let
...in() end

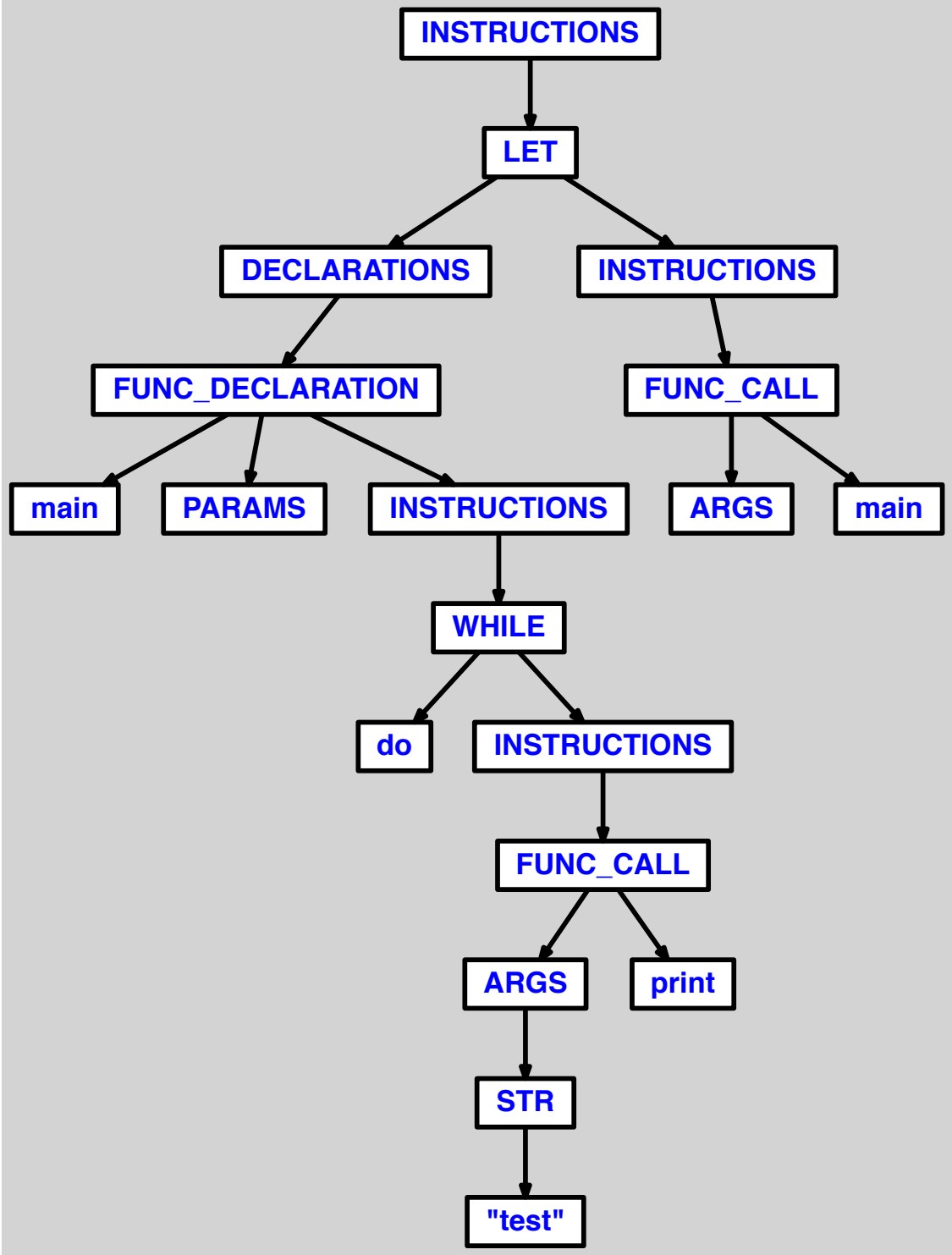
14.1.7 <while> avec oubli du <do>

```
let
  var i := 0

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

Pas d'AST, problème de syntaxe.

14.1.8 <while> avec oubli de la condition

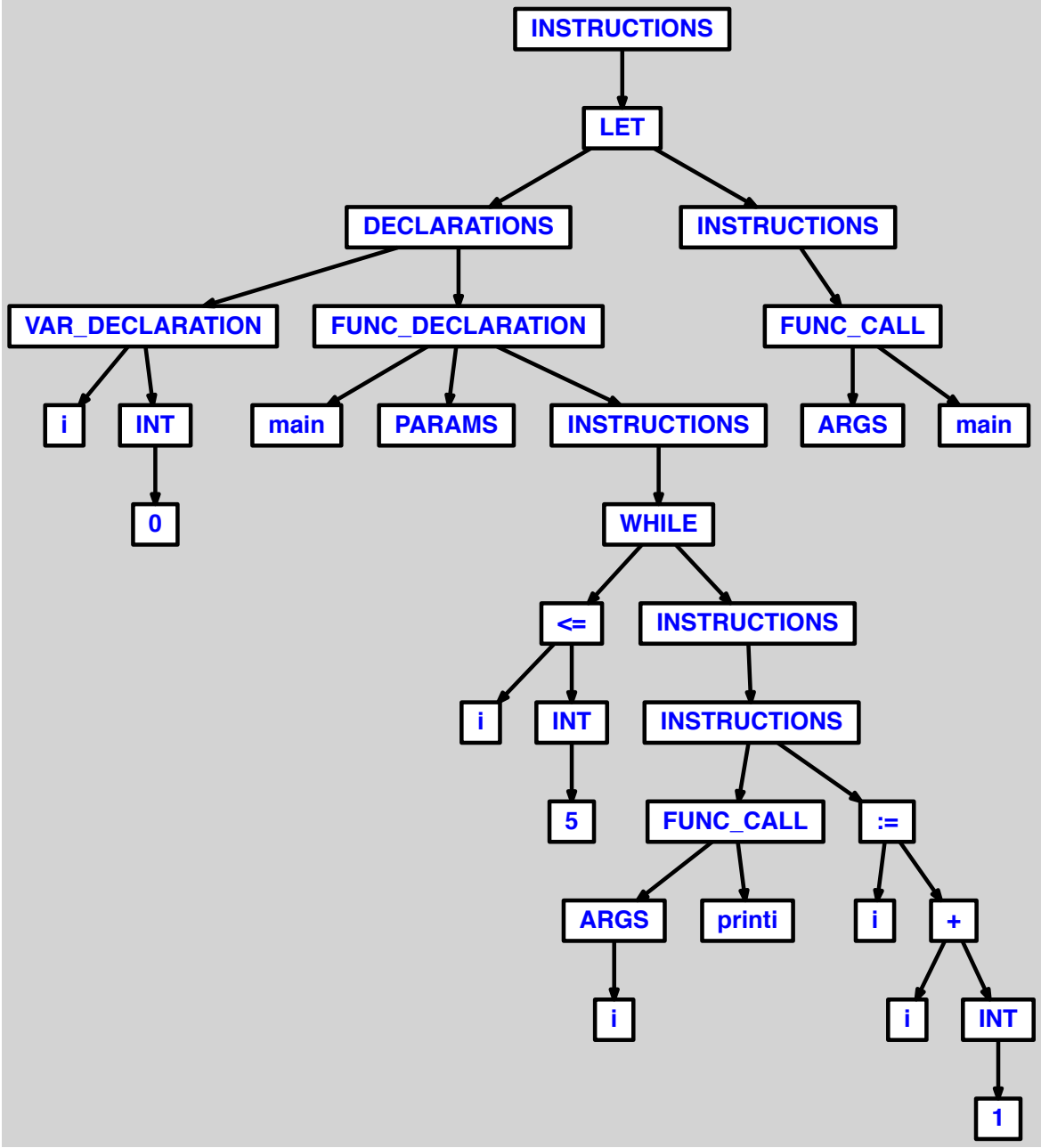
```
let
  function main() =
    while do
      print("test")
in main() end
```


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
do
INSTRUCTIONS
FUNC_CALL
ARGS
print
STR
"test"

14.2 OK

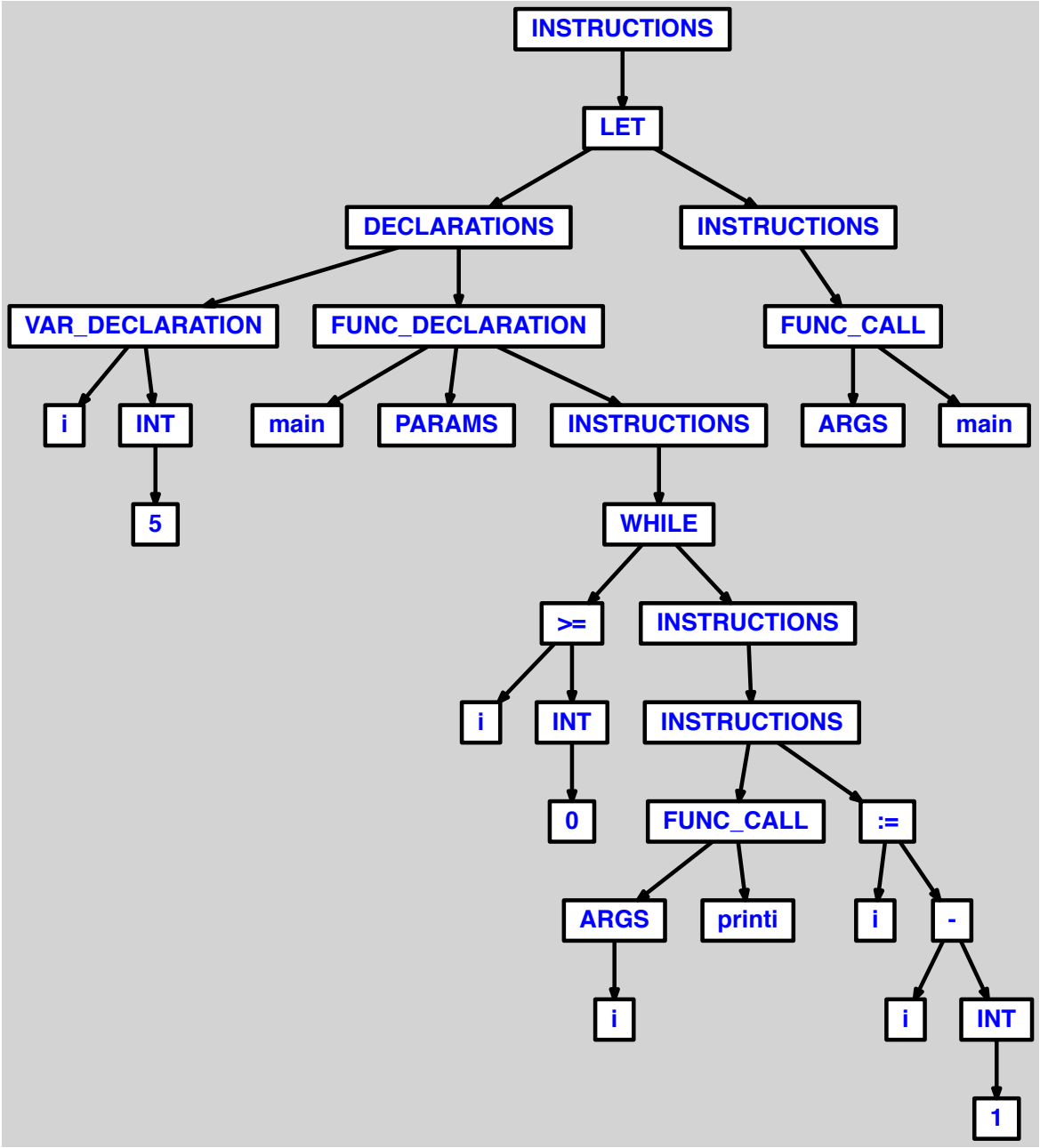
14.2.1 <while> avec iteration croissante

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
WHILE
<=
INSTRUCTIONS
i
INT
INSTRUCTIONS
5
FUNC_CALL
:=
ARGS
printi
i
+
i
i
INT
1

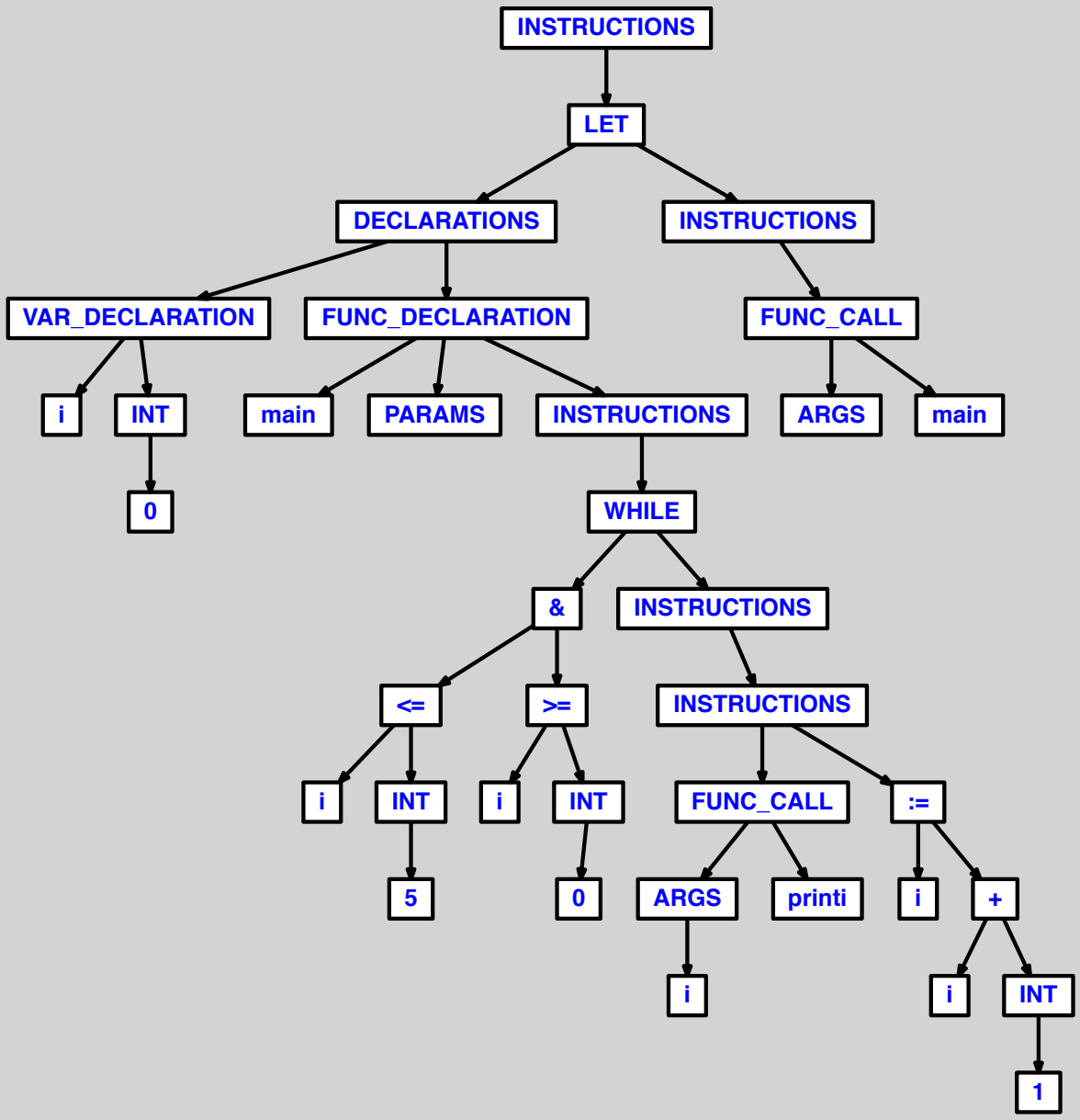
14.2.2 <while> avec iteration decroissante

```
let
  var i := 5

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
5
WHILE
>=
INSTRUCTIONS
i
INT
INSTRUCTIONS
0
FUNC_CALL
:=
ARGS
printi
i
-
i
i
INT
1

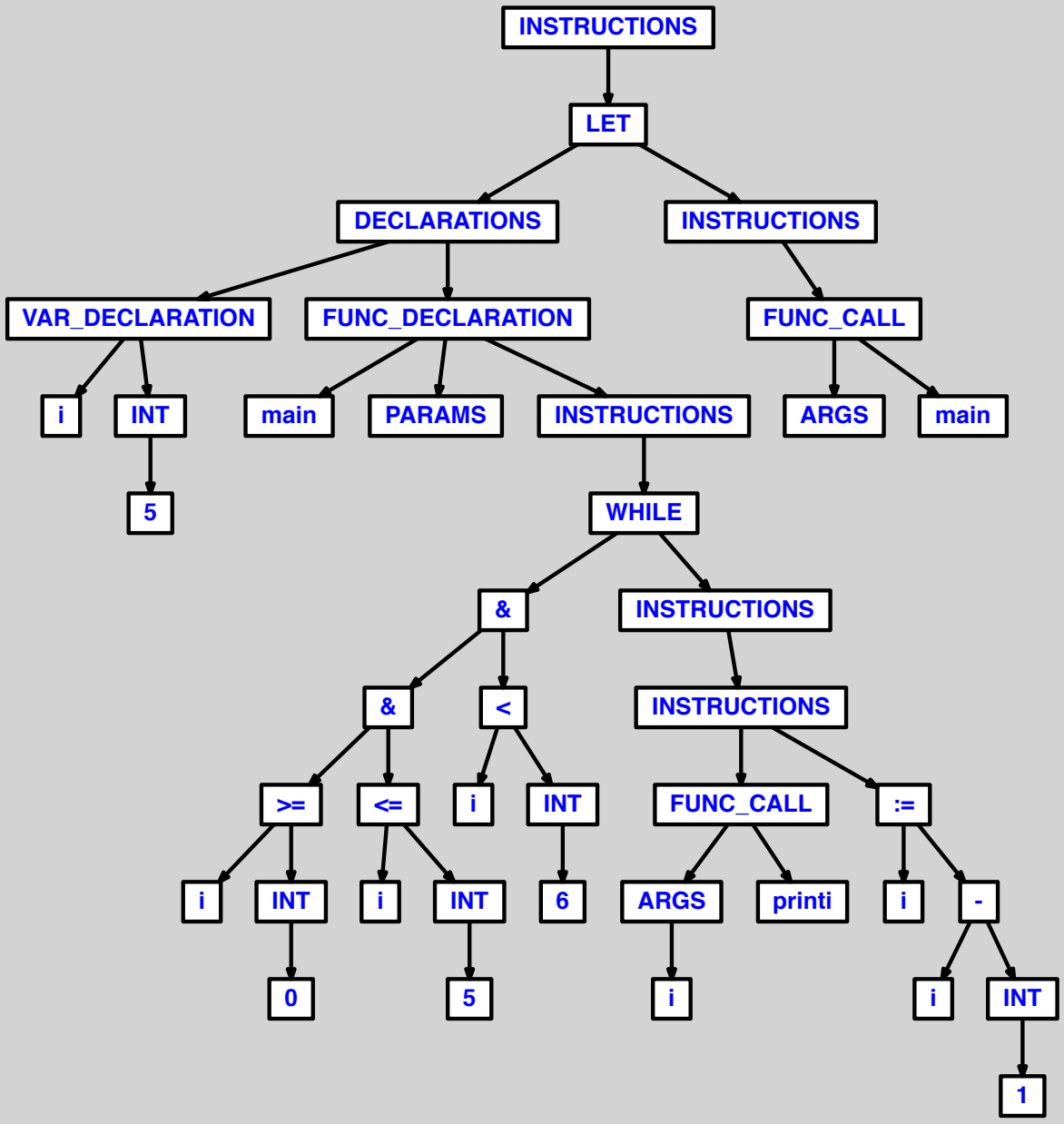
14.2.3 <while> avec double-condition

```
let
  var i := 0

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
WHILE
&
INSTRUCTIONS
<=
>=
INSTRUCTIONS
i
INT
i
INT
FUNC_CALL
:=
5
0
ARGS
printi
i
+
i
i
INT
1

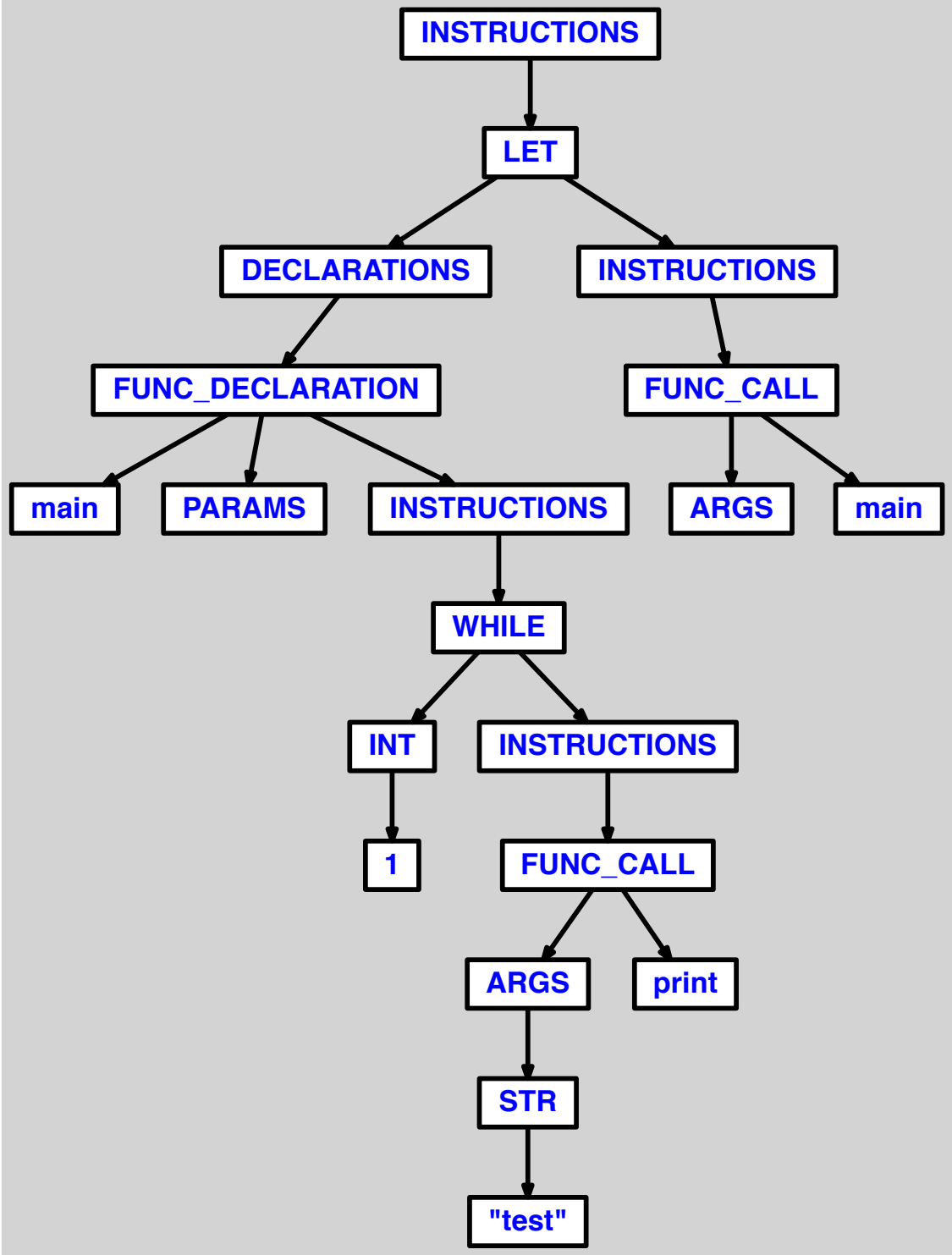
14.2.4 <while> avec triple-condition

```
let
  var i := 5

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
5
WHILE
&
INSTRUCTIONS
&
<
INSTRUCTIONS
>=
<=
i
INT
FUNC_CALL
:=
i
INT
i
INT
6
ARGS
printi
i
-
0
5
i
i
INT
1

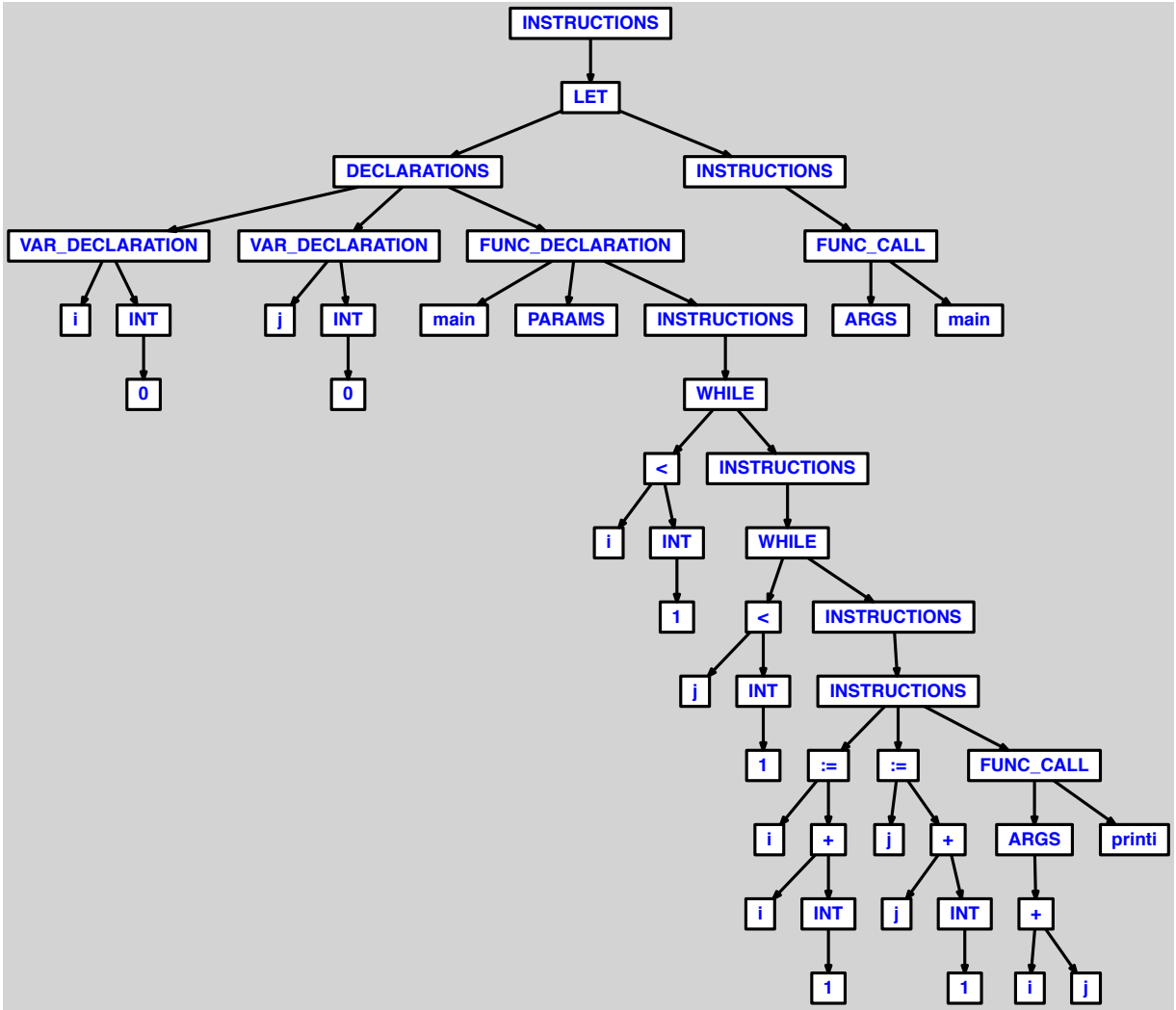
14.2.5 <while> avec condition entiere

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
FUNC_DECLARATION
FUNC_CALL
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
INT
INSTRUCTIONS
1
FUNC_CALL
ARGS
print
STR
"test"

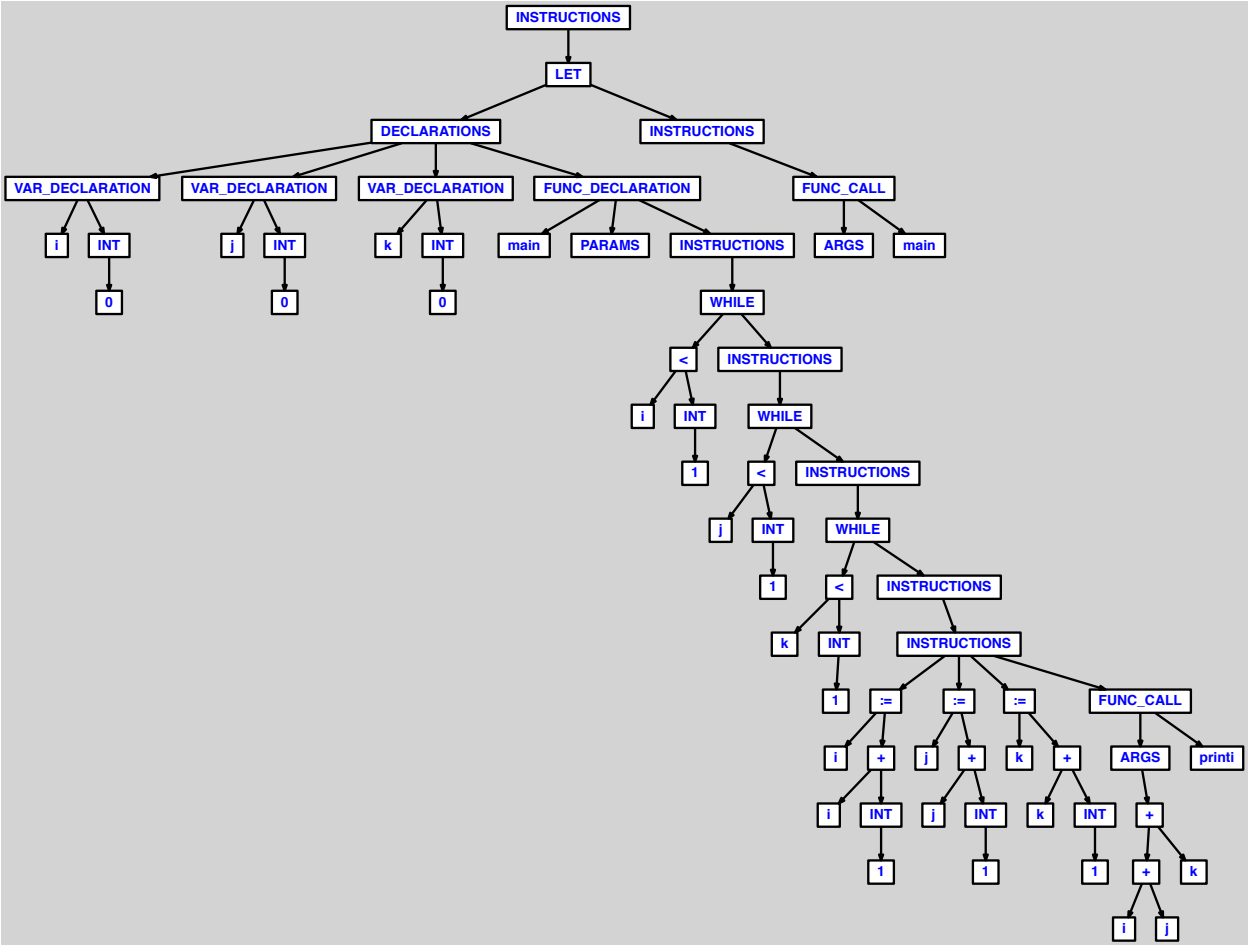
14.2.6 <while> avec double-imbrication

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
WHILE
<
INSTRUCTIONS
i
INT
WHILE
1
<
INSTRUCTIONS
j
INT
INSTRUCTIONS
1
:=
:=
FUNC_CALL
i
+
j
+
ARGS
printi
i
INT
j
INT
+
1
1
i
j

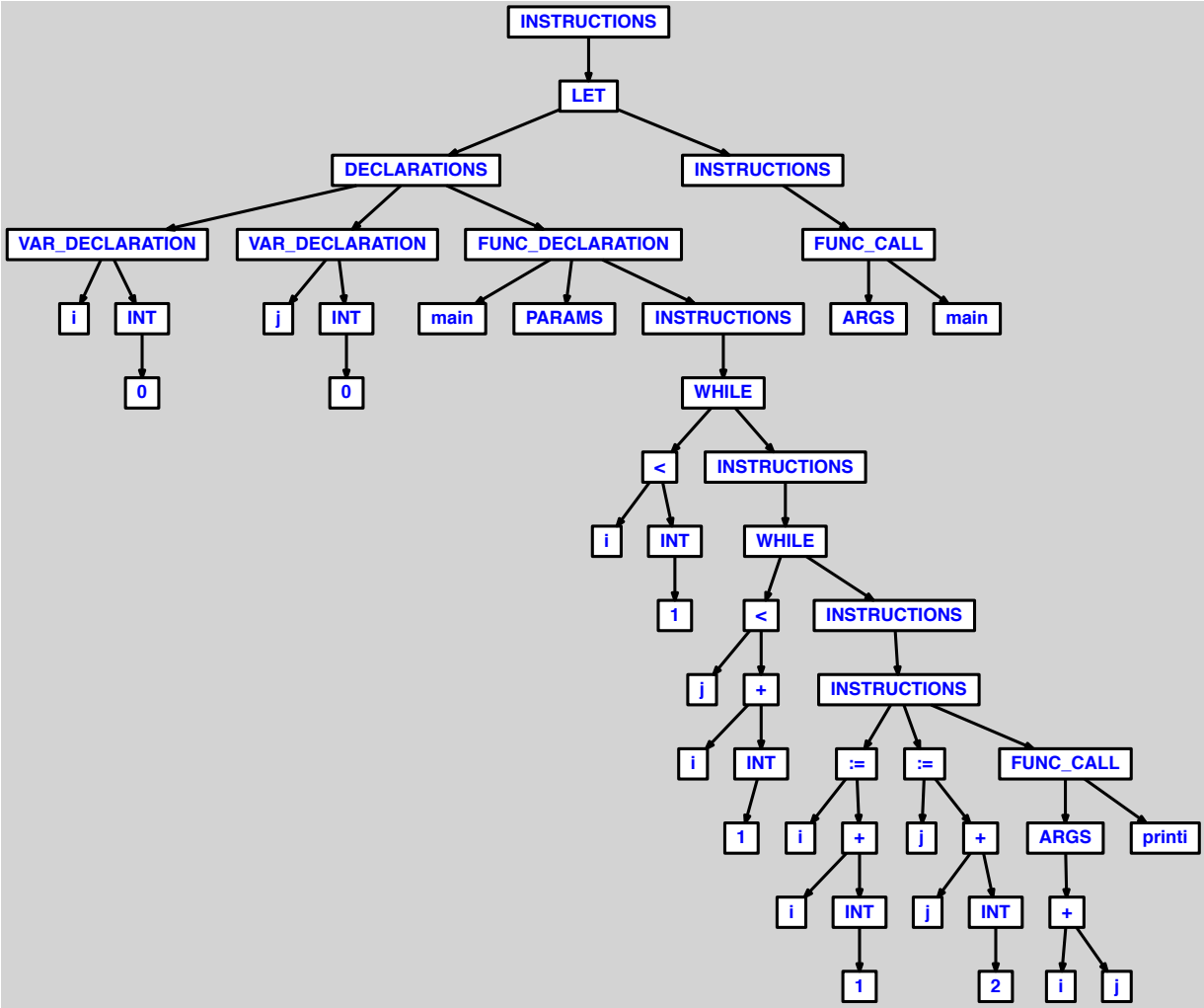
14.2.7 <while> avec triple-imbrication

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
k
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
0
WHILE
<
INSTRUCTIONS
i
INT
WHILE
1
<
INSTRUCTIONS
j
INT
WHILE
1
<
INSTRUCTIONS
k
INT
INSTRUCTIONS
1
:=
:=
:=
FUNC_CALL
i
+
j
+
k
+
ARGS
printi
i
INT
j
INT
k
INT
+
1
1
1
+
k
i
j

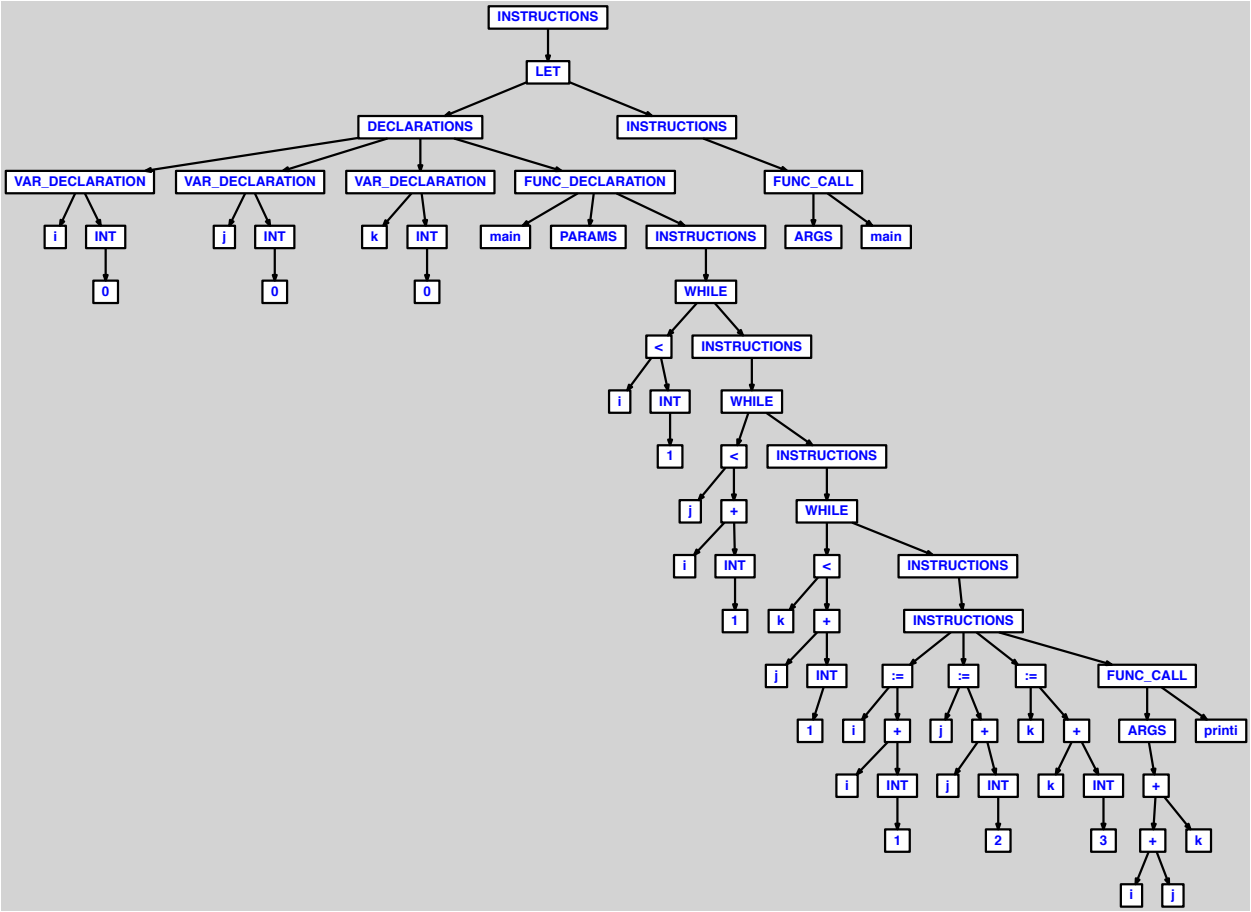
14.2.8 <while> avec reutilisation de compteur

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
j
INT
main
PARAMS
INSTRUCTIONS
ARGS
main
0
0
WHILE
<
INSTRUCTIONS
i
INT
WHILE
1
<
INSTRUCTIONS
j
+
INSTRUCTIONS
i
INT
:=
:=
FUNC_CALL
1
i
+
j
+
ARGS
printi
i
INT
j
INT
+
1
2
i
j

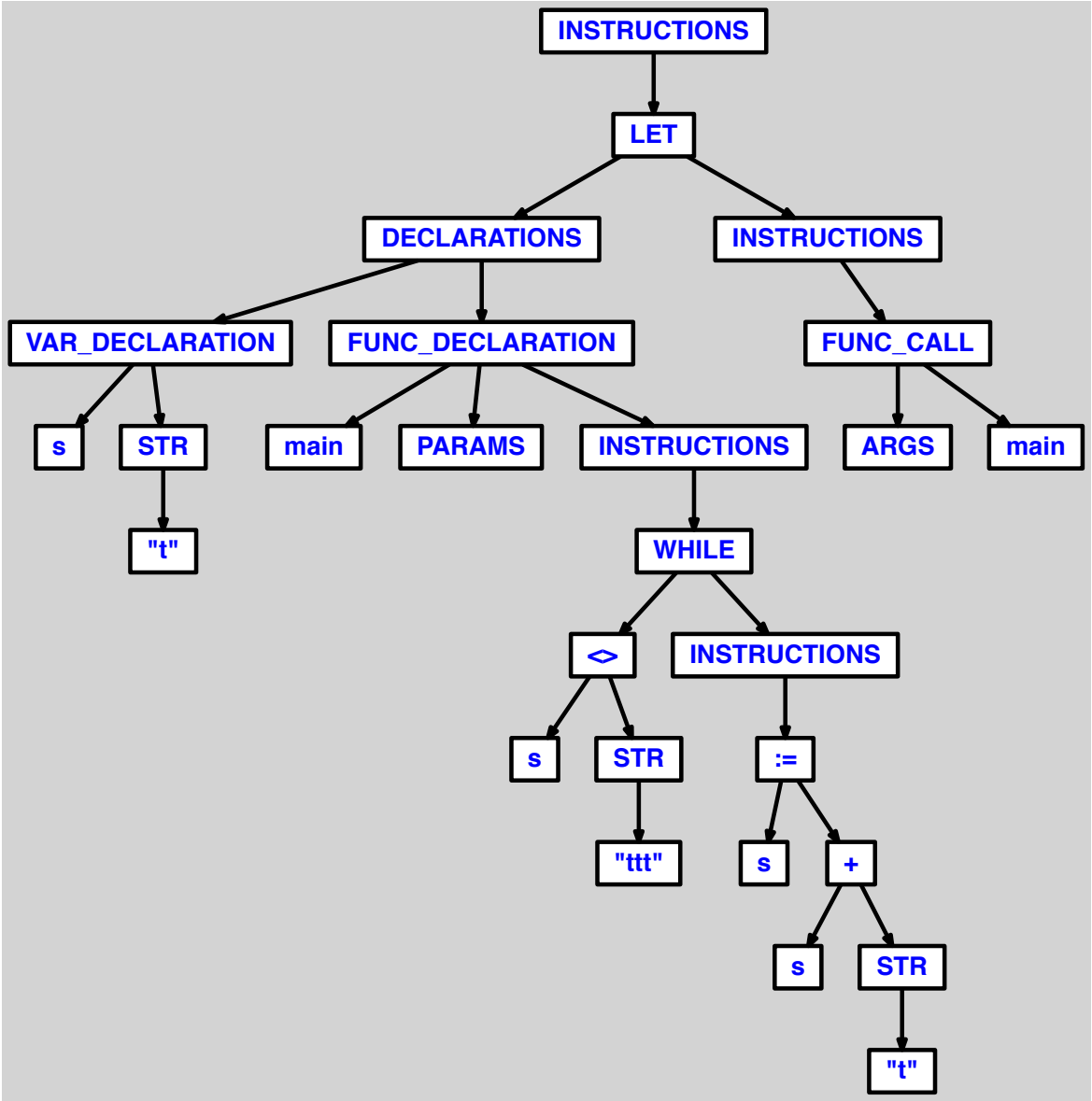
14.2.9 <while> avec reutilisation de compteurs

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

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
VAR_DECLARATION
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
i
INT
0
j
INT
0
k
INT
0
main
PARAMS
INSTRUCTIONS
ARGS
main
WHILE
<
INSTRUCTIONS
i
INT
1
WHILE
<
INSTRUCTIONS
j
+
i
INT
1
WHILE
<
INSTRUCTIONS
k
+
INSTRUCTIONS
j
INT
1
:=
:=
:=
FUNC_CALL
i
+
i
INT
1
j
+
j
INT
2
k
+
k
INT
3
ARGS
printi
+
+
i
j
k

14.2.10 <while> avec iteration sur une chaine

```
let
  var s := "t"

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


INSTRUCTIONS
LET
DECLARATIONS
INSTRUCTIONS
VAR_DECLARATION
FUNC_DECLARATION
FUNC_CALL
s
STR
main
PARAMS
INSTRUCTIONS
ARGS
main
"t"
WHILE
<>
INSTRUCTIONS
s
STR
:=
"ttt"
s
+
s
STR
"t"